Deterministic micro-controller selection method for complex control algorithms

JHD Bloem

orcid.org/0000-0002-2561-9198

Dissertation submitted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor:	Dr AJ Grobler
Co-supervisor:	Dr H Marais
Co-supervisor:	Dr RR le Roux

Graduation ceremony: May 2019

Student number: 24119490

Abstract

This dissertation presents a method for selecting a micro-controller unit (MCU). The driving force behind this method is to determine whether or not a MCU will have enough processing power for a complex control algorithm to execute within its available control period. The first iteration of the method can be used to redefine the criteria that the MCU must adhere to. The method has been implemented, and a MCU was evaluated. The evaluation of the MCU gave enough information to estimate additional parameters for the criteria. From this updated criteria list a second MCU was chosen that adhered to all the items of criteria except the operating frequency. This was intentionally disobeyed to show that the evaluation part of the selection method would conclude that this MCU will not be able to execute the complex control algorithm within the available control period. The first MCU that was evaluated was then used to implement the control algorithm for the intended application. The selection of a MCU and the timing estimations serves as verification of the selection method. The successful control of the plant and the time the control algorithm took to execute which converged with the estimated timings serves as validation of the selection method.

Keywords— AMB, PMSM, Period, MCU, Control, Frequency, Selection, Peripherals, Simulink, Matlab, Embedded, Coder, STM32, Controller, Discrete, Method

Acknowledgements

I want to acknowledge Keith Swanepoel who designed the AMB system for the TWINS at the NWU.

I want to thank my supervisor Prof. Andre Grobler who guided me during the process my masters. I will always remember the quote which he regularly uses to encourage me or any other student: "Just do it"!

I would also like to acknowledge my co-supervisor Dr. Henri Marias, his knowledge and insight was greatly appreciated and widely used.

I would like to thank my parents Dirk and Bonita Bloem who supported me during the course of my Masters. I would also like to thank my fiancée Larisha Kuhn, who gave me the strength and the will to complete my Masters.

Lastly I want to give thanks to the Lord for providing me with both the talents and opportunity to do this masters.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	analog to digital converter
AF	alternative function
AMB	active magnetic bearing
API	application programming interface
ART	adaptive real-time
AXI	advanced eXtensible interface
CMSIS	cortex microcontroller software interface standard
CPU	central processing unit
CS	chip select
DAC	digital to analog converter
DC	direct current
DMA	direct memory access
DMIPS	Dhrystone mega instructions per second
DSP	digital signal processing
EEMBC	Embedded Microprocessor Benchmark Consortium
FPU	floating point unit
FSO	full scale output
GPIO	general purpose input output
HAL	hardware abstraction layer
HDD	hard disk drive
I2C	inter-integrated circuit
I/F	interface
IC	integrated circuit
IDE	integrated development environment
LL	low layer
LPF	low pass filter
ISS	instruction set simulation
MCU	micro-controller unit
MFLOPS	mega floating operations per second
MOSFET	metal oxide semiconductor field effect transistor
MSPS	mega samples per second
NWU	North-West University
OS	operating system
PA	power amplifier
PCB	printed circuit board
PI	proportional integrate
PID	proportional integrate derivative
PLL	phase lock loop
PMSM	permanent magnet synchronous motor

PSU	power supply unit
PWM	pulse width modulation
RAM	random access memory
RCC	reset and clock control
RTOS	real-time operating system
SBC	single board computer
SPI	serial peripheral interface
TCM	tightly coupled memory
UI	user input
USART	universal asynchronous receiver transmitter

List of Symbols

$f_{control}$	control frequency
$T_{control}$	control period
$V_{disturbance}(n)$	The n'th sample in voltage of a disturbance

Chapter 1

Introduction

1.1 Background

1.1.1 Introduction

Selecting a MCU for a control project can be a daunting task. A guide on how to select a MCU for an application seen in [8] falls short when it comes to determining if a specific controller will be able to support the intended application. Often an overpowered MCU is selected to ensure successful operation. This is especially true for complex control algorithms. Selecting a MCU that will not be able to support the application can become an expensive venture. It would be beneficial if it can be said with certainty that a specific MCU will run the application with the required performance.

1.1.2 Example

Introduction

Consider a system where a ferromagnetic material needs to be levitated in mid-air using electromagnets. There is extensive research on systems like these especially in the field of electric machines. Instead of using ball bearings (between the rotor and the stator) researchers in this field replaced it with an AMB. AMBs are essentially electromagnets configured in a specific manner to establish levitation of the rotor.

Technical details to example

An author who designed such an AMB system determined that the control loop of his system need to be executed at a frequency of 10 kHz [9]. He needed to implement his control on a MCU or single board computer (SBC) in order to verify his AMB controller. The author broke the control loop down to very basic assembler code to determine the specifications of the processor within the controller needed [9]. This gave a rough estimation of the minimum mega floating operations per second (MFLOPS) the controller needed to execute. These estimations do not consider the architecture of the controller or any time delays between the memory and central processing unit (CPU), but purely the computational timings. This method also doesn't consider the delays associated with the peripherals or delays of any interface (I/F) electronics. From the results of the mentioned author, the estimated execution time does not match the implemented execution time. It was estimated that the control of the AMB system would execute within 5 μs whereas the implementation of the AMB control was executed in 18.18 μs. The implemented control took a factor of 3.6 longer than calculated. The estimations and recorded timings in this example were done on an AMB system with 5 AMB stators [9]. This author did not know how to specify the processing requirements for a MCU correctly. The reasoning behind the assembler decomposition was a step in the right direction.

Reason for more background information

To solve this large deviation, it is important to note at what stage in the author's methodology this calculation has been used.

1.1.3 Systems design

Developing a system for a specific application which involves electronics has a methodology to it. A proven and most widely used methodology is the systems design approach. This methodology was also used by the previously mentioned author [9]. The methodology for a project which involves the control of a plant will look more or less like this:

Methodology:

1. Introduction
2. Problem Statement
3. Plant
4. Controller
5. Simulations
6. Implementation
7. Control verification

The big deviation between the estimated execution time and the implemented execution time would form part of the implementation stage of the methodology as mentioned above. However, to move on to the implementation stage, information from the controller stage is needed.

1.1.4 Control loop and control frequency

The control loop forms part of the "Controller" section of the systems design approach. Figure 1.1 is a typical example of a control loop. The control algorithm can be seen as the controller subsystem, the reference, the feedback from the system and the output to the plant.

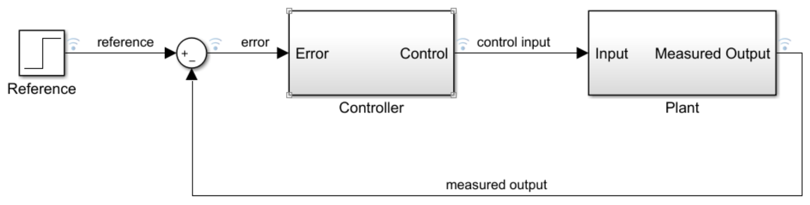


Figure 1.1: Typical Control Loop

A control law will include the following stages: feedback stage, processing stage and an output stage. Concerning digital control, the control law will need to be executed at a designed frequency. The feedback, processing and output stage will each consume a portion of the control period. The control period is defined as:

$$T_{control} = \frac{1}{f_{control}}, \tag{1.1}$$

where $T_{control}$ is the control period and $f_{control}$ is the control frequency.

The control period executed at a designed control frequency is illustrated in Figure 1.2. A single control period can also be seen as a control step. Each control step will include the previously mentioned feedback, processing and output stages. Considering a single control step as a function, the control frequency can be established by calling the function at the designed control frequency. If the function is called in a continuous polling manner and the accumulated time that each stage consumes exceeds the control period, the designed control frequency will not be met. This phenomenon is called an over-run. If the accumulated time does not exceed the control period, the CPU of the MCU will do nothing within that remaining time of the control period. The concept explained above is captured by the timeline shown in Figure 1.2.

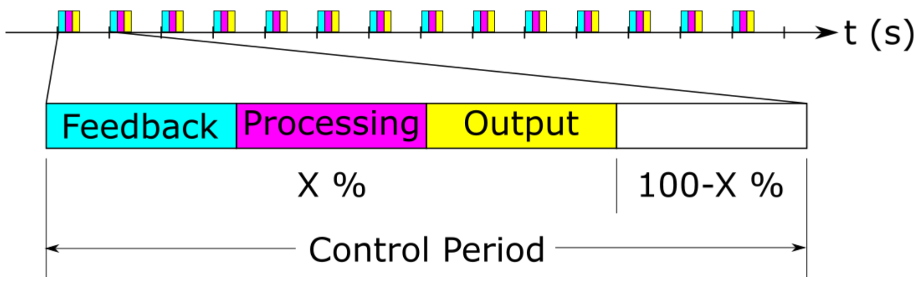


Figure 1.2: Control Period

1.1.5 Implementation

MCU selection

The selection of a MCU forms part of the Implementation section of the systems design approach. The information from the control section will assist in compiling a list of specifications that the MCU must adhere to. The methodology of selecting a micro-controller can be seen below:

1. Get requirements for MCU.
2. Compiling a list of MCUs with requirements from step 1.

This list of requirements must include the peripherals that the MCU will need. The requirements should also include the type of CPU the MCU should utilise. This implies whether or not the CPU should include a digital signal processing (DSP) unit or a floating point unit (FPU).

Examples

In the academic literature there seems to be a trend regarding the implementation of a control law. A short list of control research for AMBs and PMSMs can be see in Table 1.1. Most academics use Simulink® to design and test a control concept. The advantage to using Simulink® is that the implementation of this control law is usually done on a dSpace controller. The dSPACE® company has made it very easy for control theory academics to implement their designed controller onto the dSPACE® hardware. The hardware support package that dSPACE® made for Simulink® has made it easy to implement a controller on their hardware. More details on hardware support packages for Simulink® can be seen in Section 2.7.2.

Table 1.1: Summary of simulation packages and controllers used

Author	**Control application**	**Simulation package used**	**Implemented controller**
	AMB		
Steyn [10]	H AMB	Simulink®	dSPACE®
Myburgh [7]	AMB	Simulink®	dSPACE®
Kiani [11]	Hybrid 3 pole AMB	?	Pentium 4 CPU
Aucamp [12]	Model predictive AMB	Simulink®	dSPACE®
Le Roux [13]	AMB	ModelSim and Matlab®	FPGA
	PMSM control		
Zolfaghari [14]	Neural PMSM control	Simulink®	dSPACE®
De Klerk [15]	PMSM	Simulink®	dSPACE®
Qutubuddin [16]	Brain emotional PMSM	Simulink®	FPGA
Guo [17]	In wheel PMSM	?	TI
Kruger2011 [6]	Vector PMSM	Simulink®	dSPACE®
	AMB and PMSM control		
Kruger2014 [18]	AMB, PMSM control	Simulink®	dSPACE®
Baumgartner [19]	Slotless self-bearing PMSM	ANSYS and COMSOL	FPGA
Herbst [9]	AMB and PMSM	Simulink®	TI

Current selection helpers

The big-O notation is a good estimate to determine how complex and computationally expensive an algorithm is. However, it can not give a clear answer to whether or not a MCU will be able to execute the control within the required timeframe of the control frequency. More information about the big-O notation can be seen in Section 2.6. Benchmark software for embedded MCUs also exists but requires the MCU in order to classify the computational intensity of the control algorithm to the benchmark. The Embedded Microprocessor Benchmark Consortium (EEMBC) has a variety of benchmark suits which can be used. More information about this can be seen in Section 2.4.

1.2 Problem statement

No clear method can precisely determine if a MCU would be able to execute a specific control algorithm within a certain time “budget”. There is a trend where the engineer selects an overpowered controller for the application or sticking with what he/she knows. The engineer is used to working with a specific brand and MCU line with regards to his/her experience with that micro-controller. The use of a dSPACE® controller or an overpowered MCU, however, is not possible in all scenarios. This is especially true for the world outside academia. The cost of a dSPACE® controller is expensive, and an overpowered MCU is unnecessary if the cost can be reduced by selecting a cheaper MCU that will fit the application and its cost. A solution between the big-O-notation and EEMBC benchmark software is needed.

1.3 Issues to address

1.3.1 Introduction

A method for selecting a MCU which will be used for a real-world application (not just for academic purposes) is needed. A method that will provide certainty about a possible MCU. Determining how much delay a peripheral will add to the control period can be calculated.

Issues that will be addressed can be seen in the list below:

- Find a way to select a MCU which will be able to execute the intended application within its time budget.
- Determine how to validate that the selection method has successfully provided a MCU that can execute the intended application within the required time budget.

1.4 Methodology

During this dissertation a MCU selection method will be designed, implemented, verified and validated. For a selection method to be designed and implemented, a proposal on how the method will work can be seen in Section 1.4.1. The verification and validation for this dissertation is explained in Section 1.4.2 and 1.4.3 respectively.

1.4.1 Proposal

Table 1.1 shows examples of complex control algorithms. These algorithms can be used as a starting point for the selection method. The proposal commences just before the implementation stage. The control chosen from the table is already simulated and verified within Simulink®. This is where the plant’s time constant can be obtained as well as the controller’s time constant. It is recommended that the control in Simulink® be tested with a discrete controller. It is also recommended to test the effects of double, single, fixed point and integer finite word length effects. These effects can be investigated, and ultimately the data type can be chosen accordingly. The discrete controller will have a cycle time or execution frequency(In most cases this will be equal to each other). Call this Y.

The next step would be to identify MCU candidates for the implementation of the discrete controller. This can be done by compiling a list of required hardware peripherals that the controller must include. The specification of the hardware peripherals can also be calculated from the information obtained by the simulations. It is also recommended to include the effects that these hardware peripherals will have on the controller in the simulations.

The list of candidates will most likely have different architectures, integrated development environment (IDE)’s, compilers, etc. All of the supporting software should be downloaded at this stage. The supporting IDE will most likely have an instruction-set simulation program embedded into it. These programs will also most likely have the functionality of a "virtual stopwatch".

The next step will be to use Simulink® to compile the control algorithm to c/c++ code. If Simulink® was not used, the next step will be to convert the simulations into use-able code for a specific MCU.

The stopwatch can then be used to obtain the total time the controller will take to execute the coded control algorithm. Call this X.

The answer to whether or not the chosen MCU will be able to execute and control the plant can be answered in the following manner:

MCU load is the time of the executed algorithm normalised to the control period:

$$Load = \frac{X}{Y} \times 100\%$$

The output of this calculation should be less than 100%. The engineer should use his/her own discretion when selecting a MCU which operates close to 100% load.

1.4.2 Verification

Verification is concerned with determining whether or not a specific algorithm or equation has been implemented correctly. This is typically done by means of comparisons to literature or empirical data. For this work, the verification is mainly concerned with the estimated execution time which the selection method is based on. Detailed results of the verification process can be found in Chapter 4 Section 4.3.9 and 4.3.10.

1.4.3 Validation

The main purpose of validation is to determine whether a solution meets the initial requirements.

This implies that, if the MCU identified by the selection method is acceptable, then the estimated performance specifications would be achievable in practice.

Thus, in order to validate this research, an identified MCU will be used to realise automatic control of an AMB system (as a case-study problem). Details of the validation, as well as the performance measurements of the implemented system, can be found in Chapter 5 Section 5.2.

1.5 Dissertation overview

Chapter 2: Literature
Chapter 3: Method Design
Chapter 4: Method Implementation
Chapter 5: Results and verification
Chapter 6: Conclusion and recommendations

Chapter 2

Literature

2.1 Introduction

The research needed for the successful design of the selection method was documented in this chapter. To design a MCU selection method a deep understanding of the MCU is needed. It is also recommended to understand how the instruction sets work and a how a control algorithm will utilise the instruction set and the MCU architecture. Methods to express the control algorithm will be investigated. Methods to express a MCU's processing power will also be investigated. Finally, the literature will present authors that implemented some selection method strategy.

2.2 Micro-controllers

A MCU is a device which can be programed to control a wide variaty of devices. The use of a MCU is needed in applications which requires a bit more complexity to it or has more than one function it needs to execute. A MCU consists of the following parts [1]:

- The CPU
- Memory
- And the peripherals

2.2.1 CPU

Figure 2.1 is an example of how a CPU works. The program is stored in the program memory. With each clock cycle, the program counter is incremented. The instruction that needs to be executed is located in the memory that the program counter is pointing to [1]. The contents of this memory are the instruction that needs to be executed along with any additional parameter(s) that the instruction might need. The instruction at this memory location is copied to the instruction register where it is decoded and executed. The execution of the instruction is realised through the use of predefined logic function blocks. The output of these logic blocks is routed to the control lines of the CPU [1]. Each logic block can be seen as an instruction. A collection of instructions is called an instruction set.

2.2.2 Memory

Memory is divided into two categories namely: volatile and non-volatile [1]. Volatile memory loses its data when it is switched off. Non-volatile memory retains its data when it is switched off. Program memory is located in non-volatile memory. Each MCU will make use of both types of memory. The non-volatile memory will be used for the program memory whereas the volatile memory will be used as random access memory (RAM) [1].

2.2.3 Peripherals

Alongside the CPU and the memory, a MCU will have peripherals. Peripherals can be seen as highly specialised function blocks which are separate from the CPU. Communication and use of the peripherals are established with a data bus. Each peripheral will have registers in the memory space which will be used to establish the control and use of the peripheral. The peripheral uses these registers to execute their specified function. A list of peripherals commonly found in MCUs can be seen in the list below:

- Timers
- analog to digital converter (ADC)

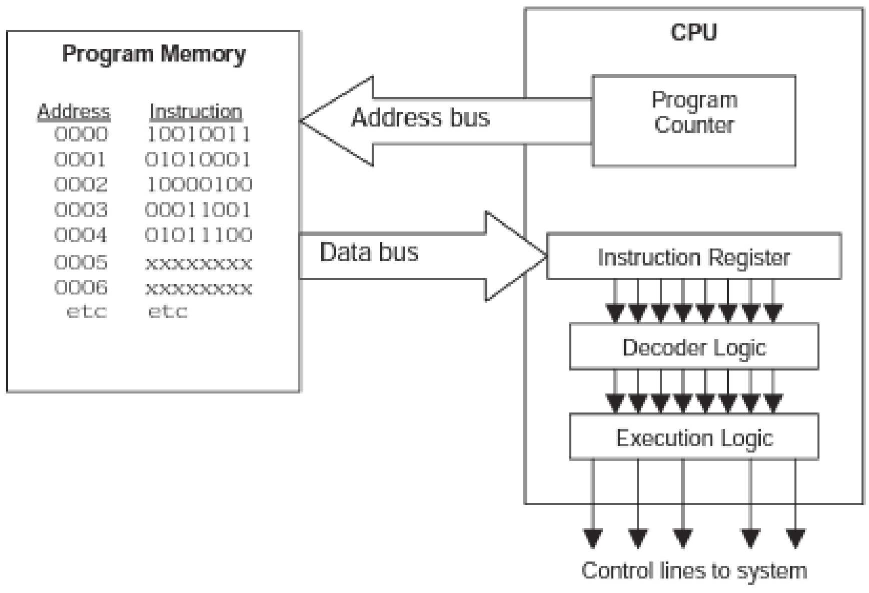


Figure 2.1: How MCUs execute programs [1]

- digital to analog converter (DAC)
- serial peripheral interface (SPI)
- inter-integrated circuit (I2C)
- universal asynchronous receiver transmitter (USART)
- direct memory access (DMA)

2.3 Instruction sets

As mentioned in Section 2.2 a MCU has a CPU embedded in it. A CPU has registers which temporarily hold information [20]. These registers are used by the instructions that will be executed. The program counter and the DPTR registers also play an important role in the use of instruction sets [20]. A CPU only understand binary numbers. The use of assembly language makes it easier for the programmer to understand and program the MCU. The assembly language is a text-based language which can be converted to binary code for the CPU to execute [20]. The assembly language has structures which need to be adhered to in order for it to be compiled to machine code [20]. The instructions that a CPU can execute is given by the manufacturer to explain the functions of each instruction that the CPU can execute [20].

The structure of an assembly instruction can be seen in Listing 2.1 [20]. The square brackets indicate that its contents are conditional to the instruction. The `label:` is used to obtain the pointer address of the current line/instruction. This `label` can then be used at a later stage to address this line. The `mnemonic` is the word that represents an instruction. The mnemoic is a descriptive text based word of the instruction. The `operands` is the location where the function's parameters will be entered. These operands will be provided by the manufacturer's description of the function. The `comment` serves no purpose to the instruction. The programmer can use this to provide additional information about the program at this instruction or line.

Listing 2.1: Assembly instruction structure [20]

```
[label:] mnemonic [operands] [;comment]
```

Each instruction requires a certain amount of cycles to execute. The manufacturer will provide this amount along with the description. This cycle count per instruction (n_{cycles}) can also indicate the time ($t_{duration}$) the instruction will used to execute, as a cycle has a direct relation to the operating frequency($f_{operating}$). This relation can be seen in (2.1).

$$t_{duration} \propto n_{cycles} \propto f_{operating} \tag{2.1}$$

Equation (2.1) is very important when it comes to determining the performance of a MCU. This relationship can be used to not only determine the time a single instruction takes execute but can be used to convert a number of cycles to a time.

2.4 Benchmarking

Benchmarking uses the concept which was communicated by (2.1). It uses this concept and a number of algorithms written in c-code and compiled for the specific MCU to the specific instruction set that it uses. Benchmarks use eight different categories to score a MCU's architecture, peripherals and the compiler used for programming [21]. The benchmark will run a series of code testing each category. The benchmark will then evaluate that category based on how long the MCU took to execute it. A weighted sum of the categories is then used to compile a single score that can be seen as the benchmark score [21]. These categories can be seen in the list below [21].

- fixed-point mathematical algorithms
- floating-point math. algorithms
- logical calculations
- digital control
- loops and conditional jumps
- polynomial calculations
- fast Fourier transform
- lookup tables

As stated previously the benchmark scores the MCU on each of these categories. Using these scores, the benchmark is calculated and provided as a single score. Benchmarking of a MCU gives a clear answer to the performance of the MCU.

Benchmark scores of a MCU can help select whether or not a MCU has enough processing power to execute the intended application [22]. In terms of the Dhrystone benchmark, MCUs running an operating system (OS) requires 300-400 Dhrystone mega instructions per second (DMIPS) whereas a real-time operating system (RTOS) would require as little as 50 DMIPS [22]. This article provides estimates to what benchmark scores is for these applications, but a more concrete answer for a specific application is needed.

2.5 Digital control theory

A typical second order plant can be represented by (2.2) [2].

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.2}$$

Where ω_n is the natural frequency, ζ is the damping ratio, and s is the s-domain variable.

The time domain equivalent of this plant for a step input can be seen in (2.3) [2].

$$y(t) = 1 - \frac{1}{\beta} e^{-\zeta\omega_n t} \sin(\omega_n \beta t + \theta) \tag{2.3}$$

Where t is time in seconds, $\beta = \sqrt{1-\zeta^2}$, and $\theta = \cos^{-1}\zeta$, and $0 < \zeta < 1$

The effects of varying the damping ratio from 0.1 to 2 can be seen in Figure 2.2.

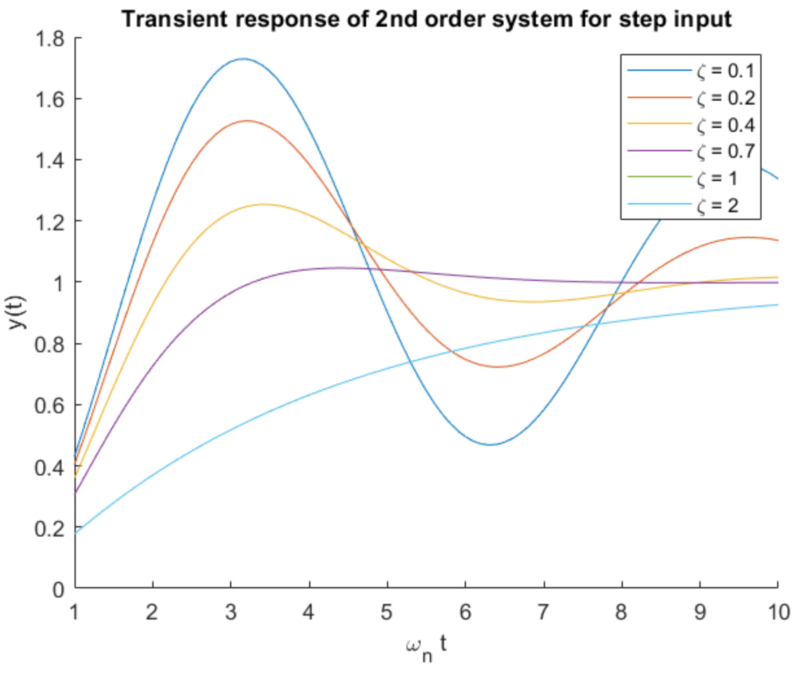


Figure 2.2: Transient response of a 2nd order system for a step input [2]

The settling time of a control system can be seen in Figure 2.3. Equation (2.4) can be used to calculate the settling time (T_s).

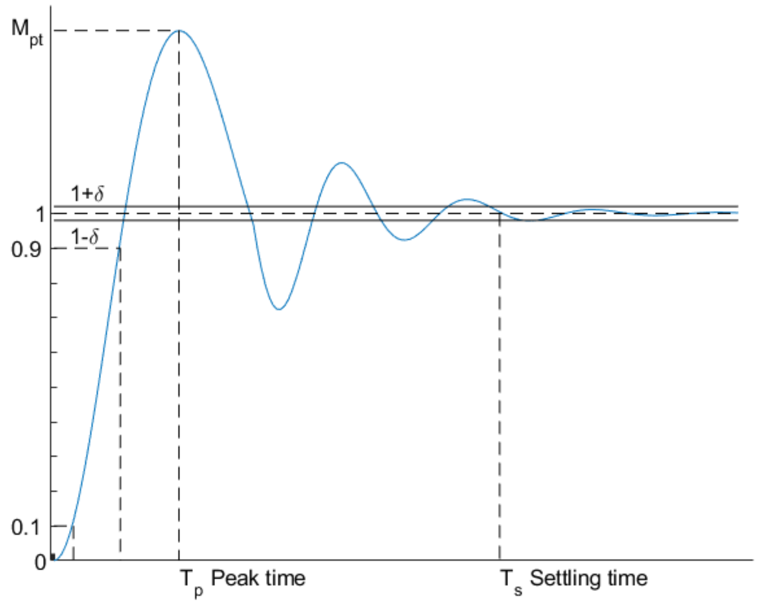


Figure 2.3: Step response of a control system [2]

$$\zeta \omega_n T_s = 4 \tag{2.4}$$

The settling time is equal to 4 time constants. Thus the time constant can be calculated in (2.5). The settling time is derived from (2.3) by finding the point in time where the equation stabilizes within a 2% margin (δ) [2].

$$\tau = \frac{1}{\zeta \omega_n} \tag{2.5}$$

Digitising the control of a plant has a destabilising effect. The destabilising effect of a zero-order hold with regards to a step input can be seen in Figure 2.4. The sampling period of the digital controller in this figure is equal to the time constant of the continuous system.

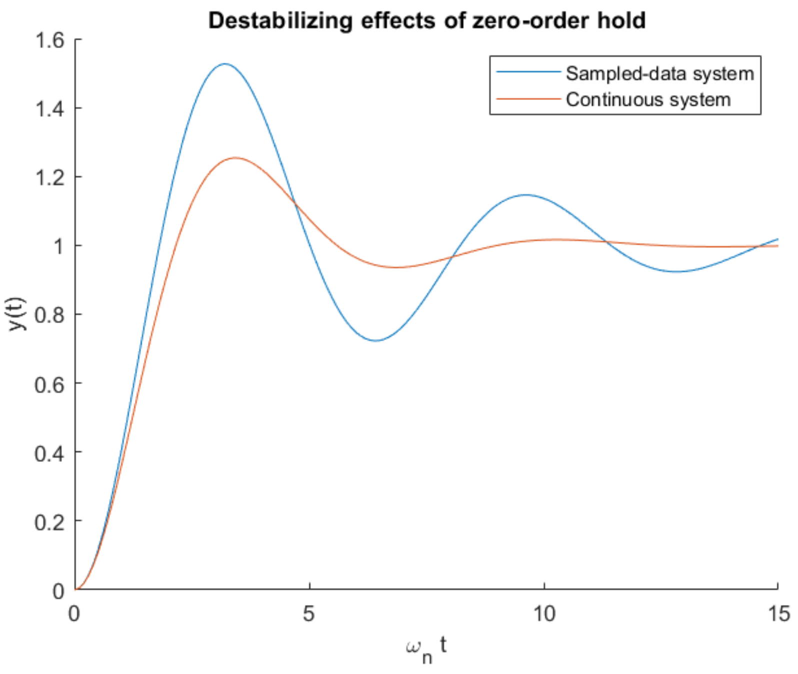


Figure 2.4: Destabilizing effects of zero-order hold [3]

Table 2.1 shows the effects that a controller's sampling frequency (time constant) has on the control of the plant. The closer the values in the first column is to 1 the more accurate the digital controller will control the plant with respect to the equivalent s-domain controller. τ in the second column refers to the time constant of the plant whereas T refers to the sampling period of the digital controller.

Table 2.1: Samples per time constant [3]

Accuracy relation to s-domain	τ/T
0.999	999.5
0.99	99.5
0.9	9.5
0.8	4.48
0.4	1.09

Using this table the digital control theory book [3] concludes that a general rule of thumb is that the digital controller needs a time constant of at least five times smaller than the plant's time constant to accurately control the plant.

2.6 Big-O-notation

The big-O-notation in computer science provides a formulation of time taken by the algorithm vs the number of inputs into the algorithm [23]. The big-O-notation does not provide the exact timing of an algorithm executed on an architecture but rather provides the relationship between the number of inputs and the output time. In a control scheme, the number of inputs will always be fixed. The big-O-notation can provide an answer to how much longer a control scheme will take if the number of inputs increases. Thus the big-O-notation will only provide a classification to how complex a control scheme is [23].

2.7 Code generation software

The use of code generation software enables rapid prototyping by generating software which will be used to program the MCU according to the project's needs. Two code generation programs will be discussed in this section. The output of these code generation programs can then use the relationship mentioned in (2.1) to calculate the timings of the MCU or the Big-O-notation can be applied to the generated algorithms.

2.7.1 CubeMX

STM32CubeMX is a STMicroelectronics® product [24]. The software is a graphical software configuration tool used to generate code to initialize a STM32 MCU and it's peripherals. The configuration tool makes use of STMicroelectronics®'s hardware abstraction layer (HAL) or low layer (LL) application programming interface (API), and the ARM cortex microcontroller software interface standard (CMSIS) to initialize the MCU. The configuration tool can create a project for a variety of IDEs. The project file is used to setup the compiler and the programming toolchain to program the MCU. This tool: "make developers' lives easier by reducing the development effort, time and cost" [24].

The generic structure of HAL API for a peripheral has functions that will initialise the resource, start the functionality of the resource and finally terminate the resource if need be. All necessary initialization is done by the STM32CubeMX configuration tool [24]. The programmer can start the resource when needed and ultimately terminate it if need be. The HAL API do, however, have a detailed description on how to use the API at the beginning of each source file, if further detail of software is needed.

2.7.2 Matlab®/Simulink®

The Simulink® Coder add-on is for Simulink® is used to generate C and C++ code from the Simulink® models, Stateflow charts and Matlab® functions. The source code generated by this add-on can be used for real-time and non-real-time applications. This enables functionalities like simulation acceleration, rapid prototyping and hardware-in-the-loop testing [5].

Key features of Simulink® Coder will include can be seen in the list below [5]:

- ANSI/ISO C and C++ code and executables for discrete, continuous, or hybrid Simulink® and Stateflow models
- Integer, floating-point, and fixed-point data types using row- and column-major layout
- Code generation for single-rate, multi-rate, and asynchronous models
- Single-task, multitask, and multicore code execution with or without an RTOS
- External mode simulation for parameter tuning and signal monitoring using XCP, TCP/IP, and serial communication protocols
- Incremental and parallel code generation builds for large models

The Embedded Coder® add-on can extend the functionality of the Simulink® Coder. The Embedded Coder® will generate C and C++ code which is faster, more readable and compact [5]. This add-on allows for more control over generated functions, files and data [5].

Key Features the the Embedded coder® has is listed below [5]:

- Optimization and code configuration options extending Matlab Coder™ and Simulink® Coder
- Storage class, type, and alias definition using data dictionaries
- Multirate, multitask, and multicore code execution with or without an RTOS
- Code verification, including SIL and PIL testing, custom comments, and code reports with tracing of models to and from code and requirements
- Standards support, including ASAP2, AUTOSAR, DO-178, IEC 61508, ISO 26262, and MISRA C (with Simulink®)
- Advanced code optimizations and device drivers for specific hardware, including ARM®, Intel®, NXP®, STMicroelectronics®, and Texas Instruments®

2.8 Interface electronics

Introduction

Communication with external components is sometimes needed. There are different ways of communicating with external components. The need to communicate with external could be because the internal components of the MCU are absent or it is not good enough. Using external components can be time-consuming with respect to the time that the MCU has available within the control period. The use of external components needs to be carefully chosen to fit the time constraints of the project.

External peripheral interfaces

Parallel

A parallel interface is where several bits (usually 4 or 8) is clocked to the IC at once. This is an extremely fast I/F but is limited to the speed at which external component can be updated. This is usually given in mega samples per second (MSPS) [25].

SPI

The SPI peripheral is a serial communication device. A serial communication device clocks the bits of the data one at a time. A SPI peripheral makes use of a 3 wire system [25]. The 3 wires consist of a MISO, MOSI and clock ports. The SPI device can either be a master or a slave. The MOSI port of a SPI device can be seen as the outgoing data port. The MISO port of a SPI device can be seen as the incoming data port. Each pulse of the clock port indicates to the SPI devices that there is a bit on the incoming/outgoing port. In a network of SPI devices, there will be a single master device and multiple slave devices [25]. The master device is responsible for the generation of the clock port. A 4 wire SPI system is exactly like the 3 wire system. However, the 4th port on the 4 wire system is the slave select port (SS). In a network of 4 wire SPI devices, the master is also responsible for providing the slaves with their SS signals [25].

I2C

The I2C interface also uses a clock and a data line to establish communication between devices [25]. A data transfer on the I2C I/F starts when a master device produces a start condition [25]. The start condition is followed by one or two bytes that contain the address and control information. All the I2C devices on the I/F has an address [25]. If the I2C device is not acting as the master, the devices will listen to the communicated data. The device would act upon the communication if the data were intended for it [25].

Evaluation of technologies

A parallel I/F will be the fastest I/F to use. This I/F does, however, need a large amount of general purpose input output (GPIO) pins to be realised. The SPI I/F is very fast and can be used to communicate data with speeds up to 50 Mbps. The use of a SPI I/F with a large number of SPI slaves can become expensive in terms of GPIO use. The slaves that are being communicated to need to be selected. Thus the required GPIO pin count increases as the number of slave devices increases. The use of an I2C I/F on the other hand does not use the slave select method. The I2C I/F uses an addressing method to communicate with multiple devices on the same bus. This method does increase the total number of bits that need to communicate on the I/F which leads to a larger delay between the start of communication and the start of the intended function of the I2C device. In general, I2C devices in comparison to SPI devices has a slower data transfer speed.

2.9 Current selection methods

Morishima [26] simulated the control algorithm with a combination of Simulink® and Synopsys CoMET. CoMET is an instruction set simulation (ISS) which is used to simulate the instructions generated by Matlab®. CoMET accumulates the cycles that the control algorithm uses to execute and control the plant completely. The article used this method to compare different control techniques for motor control of a hard disk drive (HDD). The cycle counts between the different control techniques were compared and evaluated [26]. This article showed that constraints liken RAM size, ROM size and operating frequency can be obtained through code simulation.

A constraint satisfaction algorithm can be used to assist in MCU selection by matching design requirements to the capabilities of individual MCU pins [27].

A neural network can be used to extract relevant information from large MCU databases [28]. An article presents a method for self-organising maps to help assist in the selection of a MCU.

Another way on deciding what MCU needs to be selected is by using a weighted decision matrix [29]. Using a weighted decision matrix requires experiments/experience with said MCUs to provide a score for the matrix accurately.

2.10 Conclusion

A MCU consists of several parts. One of these parts is the CPU. The CPU may or may not include a FPU or DSP core. Each MCU has an instruction set that it uses. These instruction sets will include functions to utilise all the functions of the MCU. This implies that if the CPU has a FPU or DSP core that the instruction set will include instructions for these cores. The compiler that will be used to convert c-code to binary code will be specially designed to utilise the special instructions if need be. Thanks to programs like STM32CubeMX and Matlab®/Simulink® the c-code can be automatically generated for the control algorithm designed in said package. This enables a developer to concentrate on the control of the plant and the digitisation of it. The developer can then focus more on the selection of the MCU and the I/F electronics needed. Benchmarking, the big-O-notation and the methods mentioned in Section 2.9 can be used to assist in the selection process.

Chapter 3

Method design

In this chapter, the controller selection method will be designed. This chapter will also cover simulation recommendations for the intent to convert the simulation to program code which can be used for the embedded application. This chapter will also cover simulation recommendations that will assist the selection method to accurately estimate the parameters required for the MCU and detailed steps of the selection method.

3.1 Introduction

A controller simulation will start with the modelling of a plant. The plant will then be tested with ideal theoretical sources. A controller will then be designed to utilise these sources to control the plant. When the controller shows promising results, the ideal sources can be replaced with more realistic sources that will most likely be used at implementation. The effects of these realistic sources can (if necessary) be compensated for by the controller. Controlling these realistic sources might also have some effects on the plant and the control thereof. Compensating for these effects will ensure successful implementation of the controller. The gist of this is that any hardware that will be used to control the plant, which will affect the control, need to be simulated for the controller to control the plant.

3.2 Methods overview

3.2.1 Selection method

The proposal is repeated and summarised here for convenience. These steps will be the basis of the method that will be designed. These steps are done after the simulation of the discrete controller in Simulink® and trade-off studies for the hardware of the project. This implies that both the control and all the hardware to control the plant has been decided on. The method to choose a MCU is:

1. Obtain all necessary plant information from discrete controller simulations done in Simulink®.
2. Using the information from step 1, compile a list of possible MCUs to use for the project.
3. Obtain the supporting software for the MCUs
4. Compile the discrete controller of the simulations to code which will be used on the MCU.
5. Use supporting software in simulated debugging mode and time the control step.
6. Evaluate how much did the algorithm use of the control period.

Please note that a detailed explanation of step 1 in the list above is discussed in Section 3.4.1.

3.2.2 Method validation

A more traditional approach that most people use for validation that the MCU will be able to implement a control algorithm can be seen below. This method can also be used to validate the selection method that will be designed.

Method:

1. Set an unused GPIO pin high at the beginning of the algorithm
2. Set the same unused GPIO pin low at the end of the algorithm.
3. Follow the necessary steps in the IDE to flash the code to the MCU.
4. Use an oscilloscope to view the waveform generated by the GPIO pin.

The waveform generated can be used to provide two critical details to the execution of the control algorithm. The period of the waveform will represent the chosen control period(also equal to the sampling period). The timing of the duty cycle is the time the MCU took to execute the control algorithm. This time can be used to validate the selection method as mentioned in Section 1.4.3.

3.3 Considerations

3.3.1 Using Simulink® for the intend to program an embedded system

A good simulation will lead to a better specification estimation for a MCU. After this, the controller can be converted to a discrete controller (assuming the control was simulated in a continuous Simulink® model) and the effects of this can be compensated for. A trade of study can be made on the effects that data types have on the controller.

Introduction to an example

A good example of a simulation can be seen in Figure 3.1. This simulation is done by MathWorks [4]. The simulation simulates the physical parameters in order to realise a linear actuated motion by means of a direct current (DC) motor given an input command.

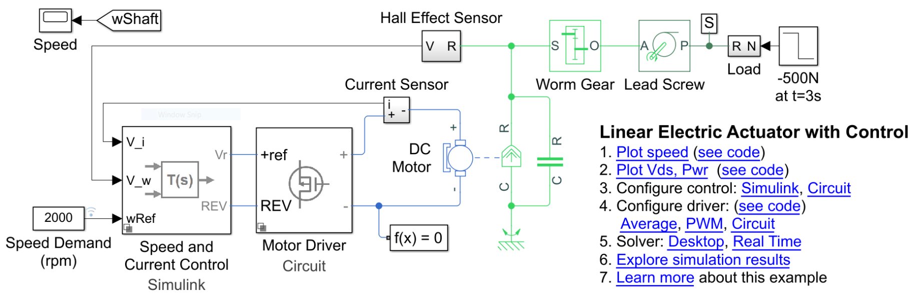


Figure 3.1: Screen-shot of MathWorks example simulation of a DC motor linear actuator [4]

Goal of example

From this simulation, the specifications of the motor, electronics and other hardware components can be obtained. The goal of this simulation was to obtain information like this. This simulation was also created to implement the speed and current control with analog electronics.

Recommendations for simulation

This example also shows how a simulation can grow during the development of the simulation. It also shows how the hardware affects the control and the plant. It is recommended to develop the simulation over time in a similar manner in order to make a better-informed decision when it comes to hardware and MCU selection. The growth of a simulation will follow.

Extending example

However, it is possible to control this system with a MCU instead of analog electronics. All the specifications for the hardware have been obtained by the simulation so far, but if we want to implement a MCU instead of the analog electronics, it is recommended to simulate additional discrete controller effects. The "Simulink®" variant is selected for the speed and current control subsystem. It is recommended to test the effects that fixed point, and floating point data types have on the system. The effects that these data types can be evaluated in order to determine if a FPU with a MCU is needed. All of this can be done within the speed and current control subsystem.

Preparing extended example for MCU implementation

The speed and current control subsystem can then be used to generate code for a MCU. The inputs and outputs of the MCU from and to the plant need to be established by means of communication with the MCU's peripherals.

Growth of a simulation

The simulation starts with a plant. This can be seen in Figure 3.2.

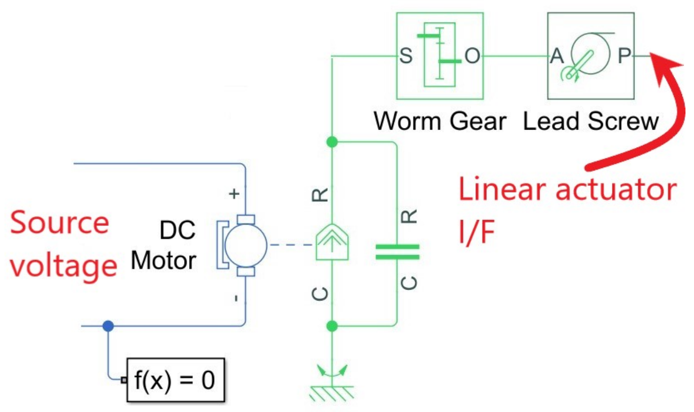


Figure 3.2: A DC motor linear actuator

This plant is controlled by an ideal voltage source. This can be seen in Figure 3.3.

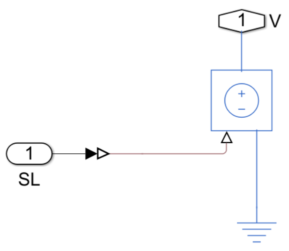


Figure 3.3: Ideal Simscape voltage source

The controller needs to control both the current and the speed. The ideal motion and current sensors for this can be seen in Figure 3.4 and 3.5 respectively.

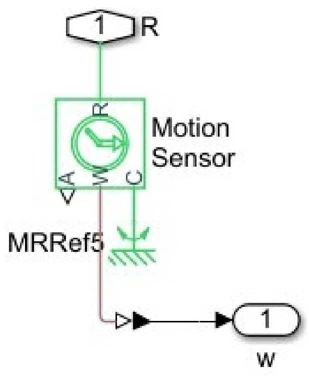


Figure 3.4: Ideal Simscape motion sensor

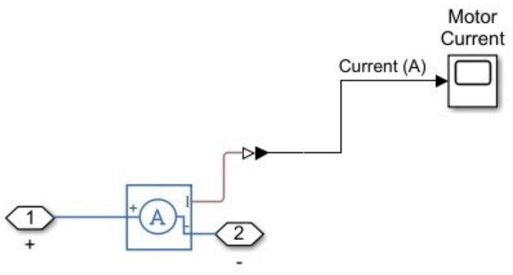


Figure 3.5: Ideal Simscape current sensor

These sources and sensors were then used to test the proportional integrate (PI) current and PI speed control. This control can be seen in Figure 3.6.

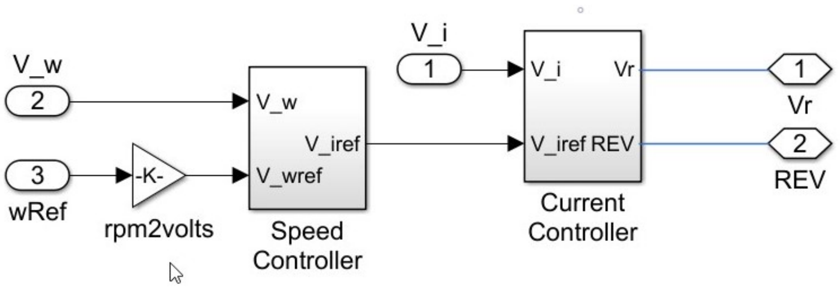


Figure 3.6: Speed and current control subsystem

The contents of the current and speed control subsystems can be seen in Figure 3.7 and 3.8 respectively.

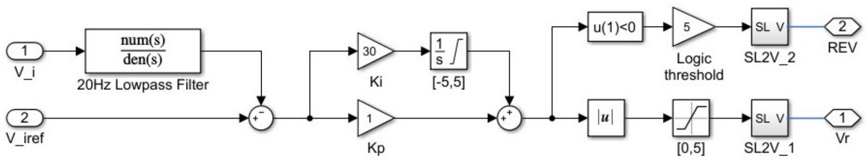


Figure 3.7: Current control subsystem

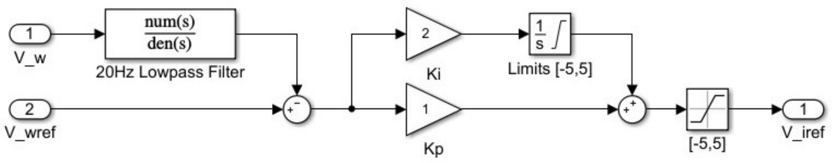


Figure 3.8: Speed control subsystem

This will verify that the PI control for both the current and the speed will work. It would be beneficial if the control can compensate for any hardware that will be used. This example extends the simulation to include the hardware that will be used to realise this system. The current sensor is replaced with an equivalent model which has a limited bandwidth of sensing. This can be seen in Figure 3.9.

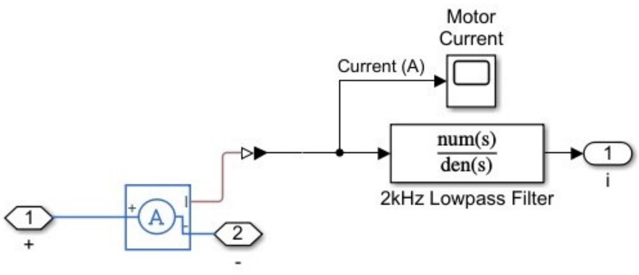


Figure 3.9: Model of current sensor

The ideal motion sensor of Figure 3.4 is replaced with a digital encoder and can be seen in Figure 3.10.

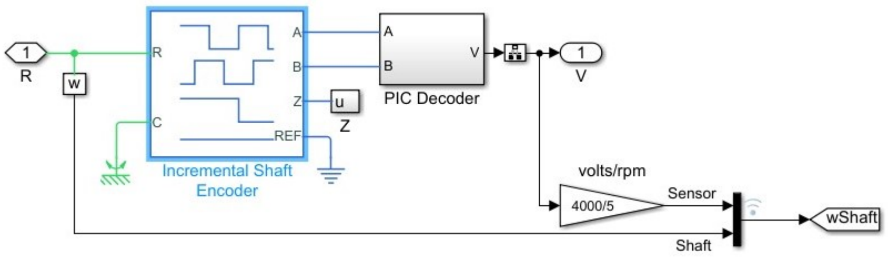


Figure 3.10: Model of digital Encoder

The ideal voltage source was also replaced with an H-Bridge circuit using metal oxide semiconductor field effect transistor (MOSFET)s. This can be seen in Figure 3.11.

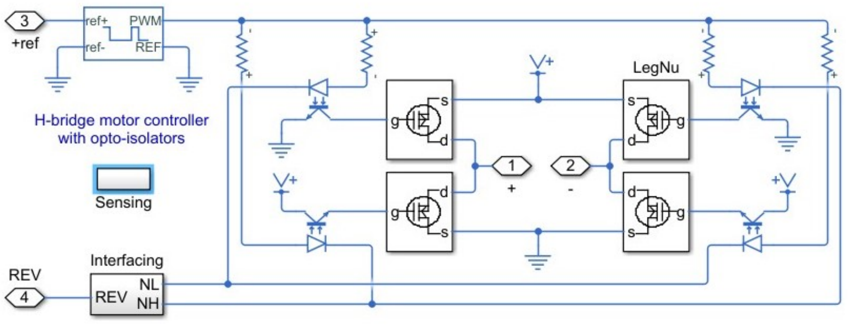


Figure 3.11: H-Bridge Circuit

The P-channel and N-channel MOSFETs is also modelled and can be seen in Figure 3.12 and 3.13 respectively.

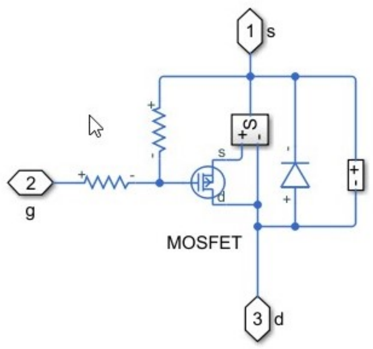


Figure 3.12: P-Channel MOSFET model

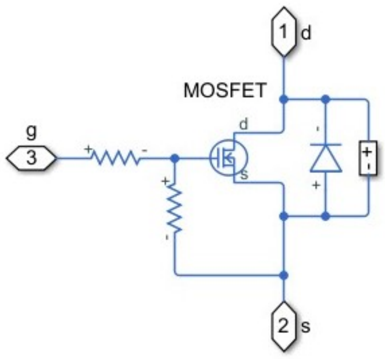


Figure 3.13: N-Channel MOSFET model

The contents of the interface block (seen in Figure 3.11) can be seen in Figure 3.14.

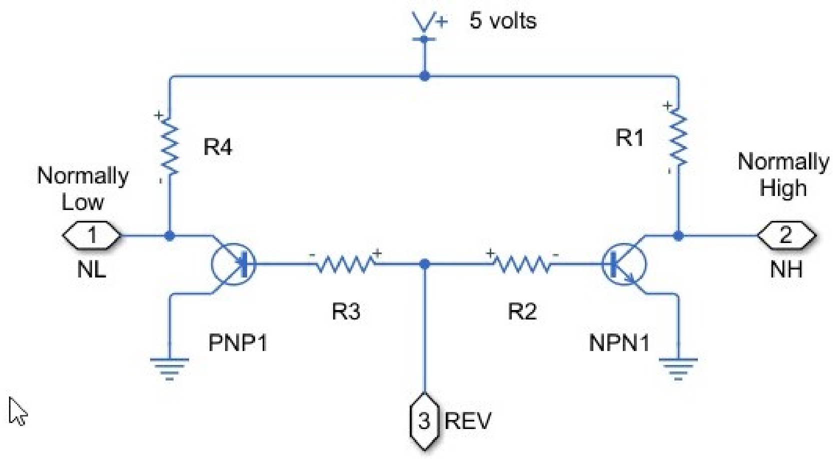


Figure 3.14: Direction control

This interface control block is used to control the direction of the motor. Now that the hardware has been modelled the effects that they have on the plant can be compensated for by the control seen in Figure 3.6.

Recommendations for design in Simulink® with end code in mind

When it comes to hardware implementation, there are things to note. Consider GPIO pin that needs to be toggled in the middle of a control loop and note the problem with this scenario. Normal hardware support software for a GPIO toggle has a sink block configuration to it. The problem in this scenario is that Simulink® uses a diagram structure. Any disjointed diagrams will be executed sequentially. Because a GPIO toggle block is not connected to the control loop diagram, the GPIO block will, in code, be executed after the control loop diagram. The use of subsystems, tokens and Simulink® functions needs to be used to overcome this problem. Using an atomic subsystem and isolating the section where the GPIO pin needs to be toggled can be used. A token system can be used when multiple sink blocks need to be set within a control loop. The token system is useful when the sink blocks need to be executed in a specific sequence. An example of this can be seen in Figure 3.15. The contents of the "Processing" subsystem is the reason for using the token system. The contents of this subsystem is a Stateflow diagram seen in Figure 3.16. It was required that the kernel Stateflow diagram be executed after the Input subsystem of Figure 3.15. When this diagram is compiled to c-code, the token system will not be coded, but the order in which the subsystems execute is maintained. When the same sink block or smaller disjointed Simulink® diagram needs to be used multiple times throughout the control loop, the use of a Simulink® function is then recommended. An example of this can be seen in Figure 3.17. The CSBus function is called more than once in this example. The settings for this block can be seen in Figure 3.18. The contents of the Simulink® function can be seen in Figure 3.19.

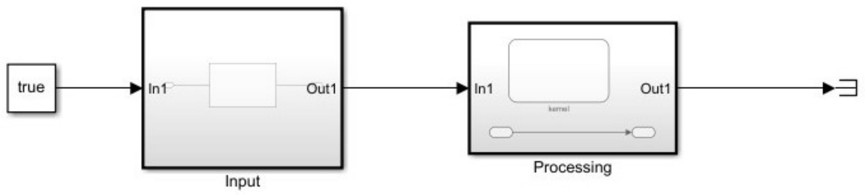


Figure 3.15: Token example

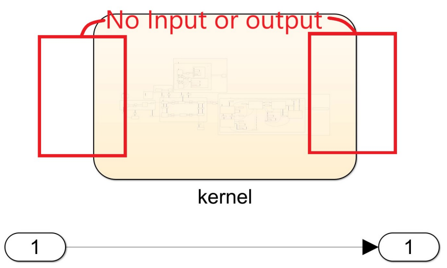


Figure 3.16: Reason for using tokens

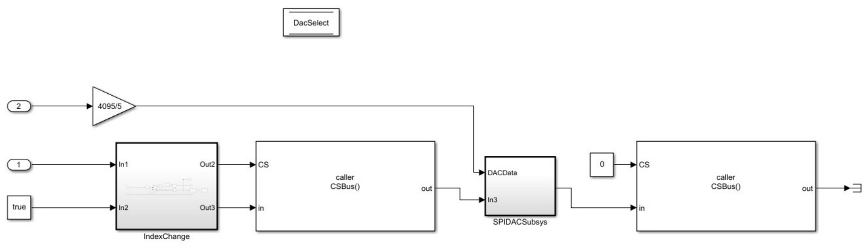
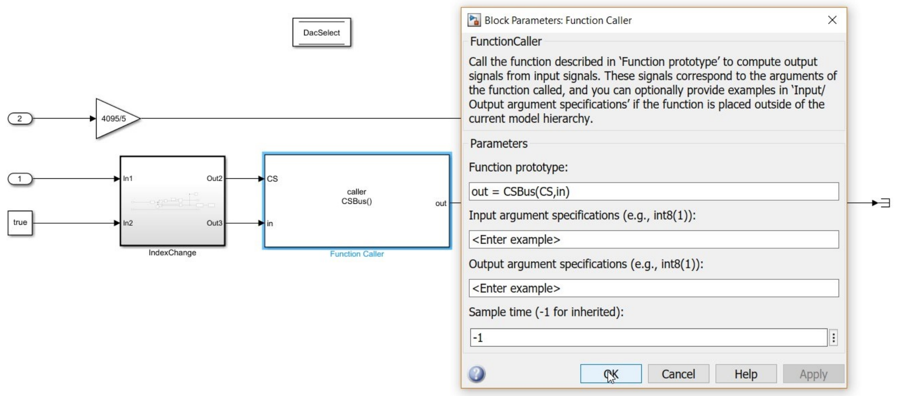


Figure 3.17: A Simulink® diagram utilizing a Simulink® function call



Block Parameters: Function Caller

FunctionCaller

Call the function described in 'Function prototype' to compute output signals from input signals. These signals correspond to the arguments of the function called, and you can optionally provide examples in 'Input/ Output argument specifications' if the function is placed outside of the current model hierarchy.

Parameters

Function prototype:

out = CSBus(CS,in)

Input argument specifications (e.g., int8(1)):

<Enter example>

Output argument specifications (e.g., int8(1)):

<Enter example>

Sample time (-1 for inherited):

-1

Figure 3.18: Simulink® function call dialog

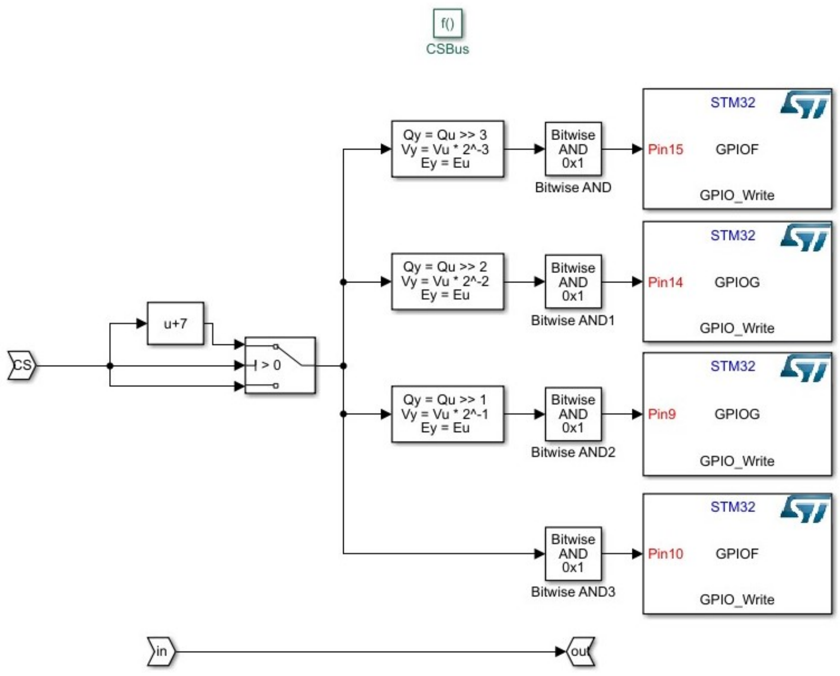


Figure 3.19: A Simulink® function block contents

Conclusion

This section gave recommendations for controller design and simulations in Simulink®. The recommendations is summarized and listed below:

1. Start the simulation with a good model of the plant.
2. Do the controller design using ideal sensors and sources.
3. Do trade-off studies to determine what hardware will be used when the system will be implemented.
4. Extend the simulation by including the models of the sensors that will be used.
5. Include models of the sources that will be used.
6. Adjust the controller designed in step 2 to compensate for simulation adjustments made in steps 4 to 5.

3.3.2 Algorithm performance improvements

Because of the nature of the controller it is crucial to utilised the control period as efficient as possible. This section will demonstrate a few recommendations to improve the time used by the algorithm.

PID implementation variants

There are different methods to implement the proportional integrate derivative (PID) control using Simulink®. The first method is to use a diagram realization of a PID control. This can be seen in Figure 3.20.

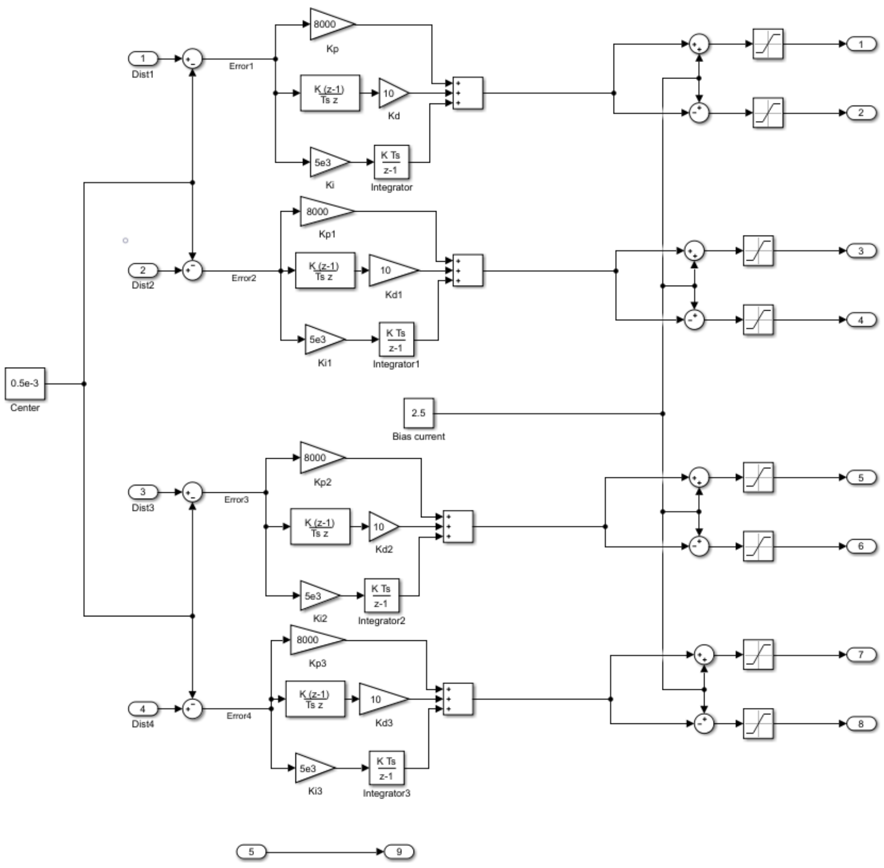
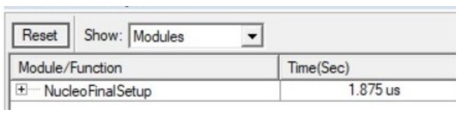


Figure 3.20: PID control using a diagram realization

The time this PID control realization took to execute can be seen in Figure 3.21.



Reset Show: Modules

Module/Function	Time(Sec)
NucleoFinalSetup	1.875 us

Figure 3.21: Execution time of the diagram PID variant

The second method to use is Simulink®'s PID controller block. This can be seen in Figure 3.22.

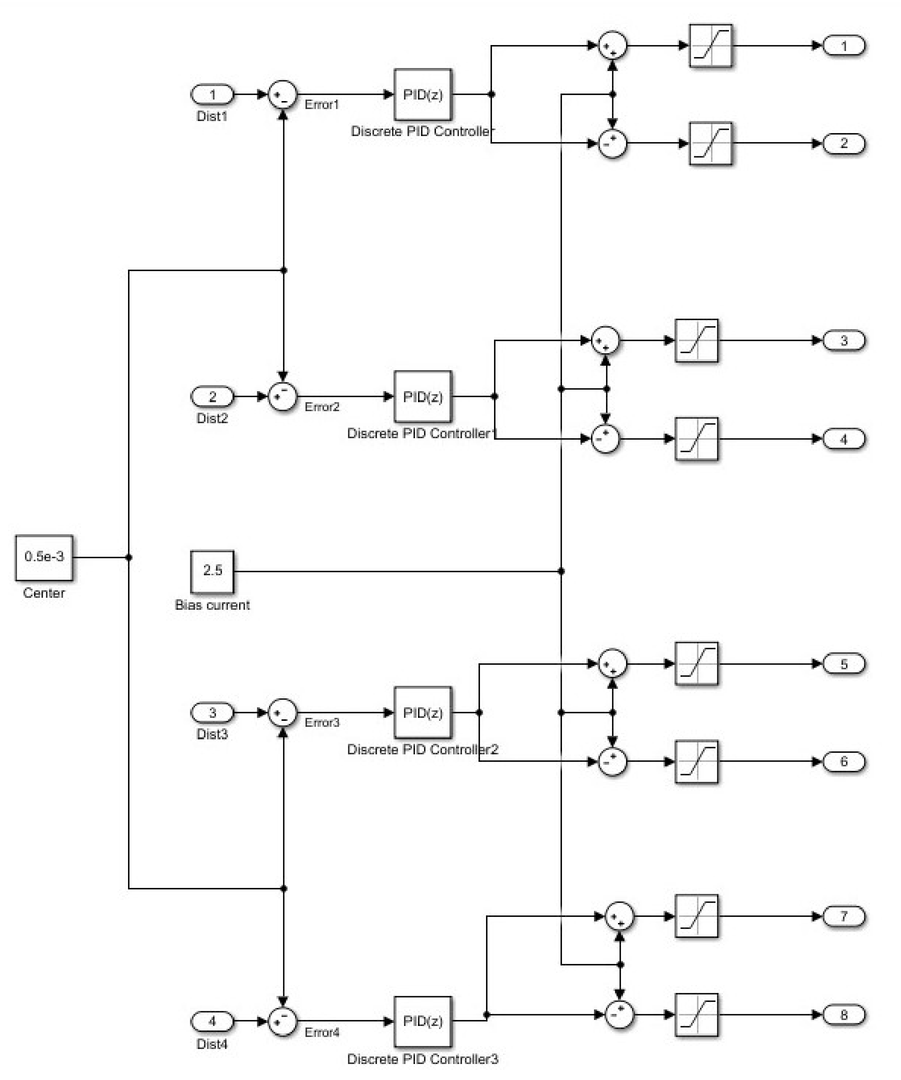
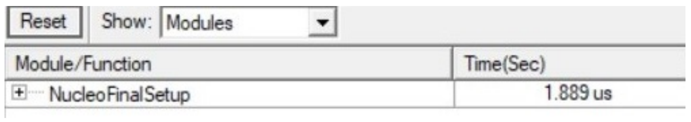


Figure 3.22: PID control using Simulink®'s PID block

The time this PID control realization took to execute can be seen in Figure 3.23.



Module/Function	Time(Sec)
NucleoFinalSetup	1.889 us

Figure 3.23: Execution time of Simulink®'s PID block

The third method to use is the CMSIS DSP library. This library includes software for PID control. This method to execute this method can be seen in Figure 3.24.

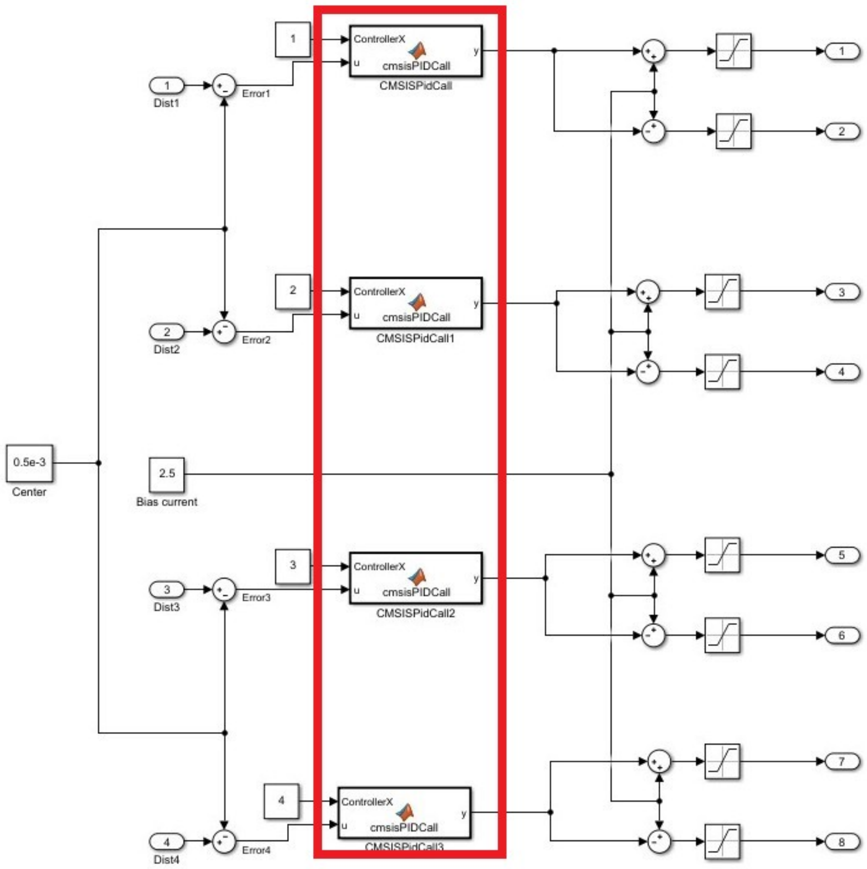
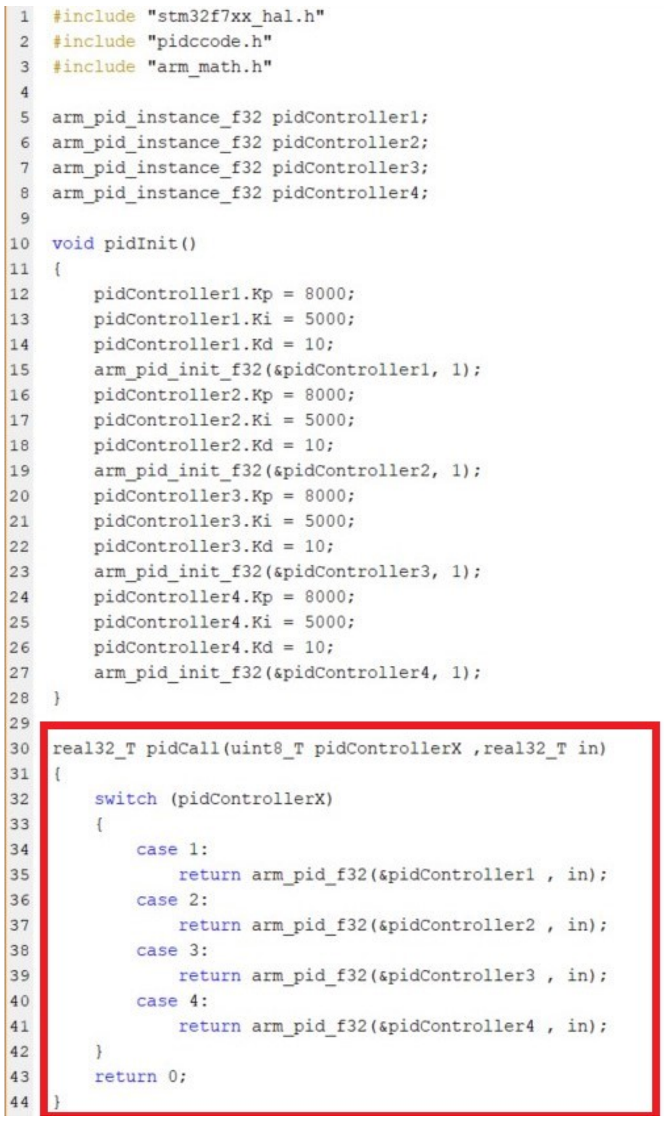


Figure 3.24: PID control using CMSIS's DSP library

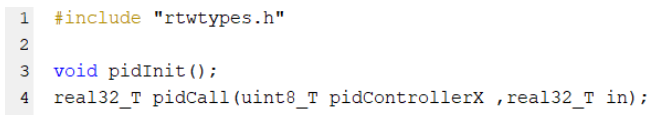
The Matlab® function blocks seen in Figure 3.24 is used to call the C-Code function "pidCall()". The C-Code for this function can be seen in Figure 3.25.



```c
#include "stm32f7xx_hal.h"
#include "pidccode.h"
#include "arm_math.h"

arm_pid_instance_f32 pidController1;
arm_pid_instance_f32 pidController2;
arm_pid_instance_f32 pidController3;
arm_pid_instance_f32 pidController4;

void pidInit()
{
    pidController1.Kp = 8000;
    pidController1.Ki = 5000;
    pidController1.Kd = 10;
    arm_pid_init_f32(&pidController1, 1);
    pidController2.Kp = 8000;
    pidController2.Ki = 5000;
    pidController2.Kd = 10;
    arm_pid_init_f32(&pidController2, 1);
    pidController3.Kp = 8000;
    pidController3.Ki = 5000;
    pidController3.Kd = 10;
    arm_pid_init_f32(&pidController3, 1);
    pidController4.Kp = 8000;
    pidController4.Ki = 5000;
    pidController4.Kd = 10;
    arm_pid_init_f32(&pidController4, 1);
}

real32_T pidCall(uint8_T pidControllerX ,real32_T in)
{
    switch (pidControllerX)
    {
        case 1:
            return arm_pid_f32(&pidController1 , in);
        case 2:
            return arm_pid_f32(&pidController2 , in);
        case 3:
            return arm_pid_f32(&pidController3 , in);
        case 4:
            return arm_pid_f32(&pidController4 , in);
    }
    return 0;
}
```

Figure 3.25: C-code to initialize and call the PID controller

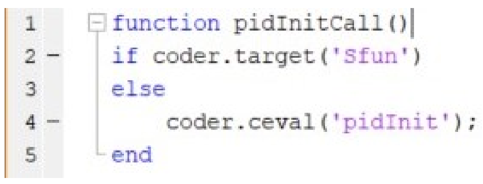
The header file for the code seen in Figure 3.25 can be seen in Figure 3.26.



```c
#include "rtwtypes.h"

void pidInit();
real32_T pidCall(uint8_T pidControllerX ,real32_T in);
```

Figure 3.26: The header file of c-code used to initialize and call PID controller

In order to utilize the CMSIS PID controller software it must first be initialize. A Matlab® function block is included initialize subsystem in the root of the Simulink® model. This Matlab® function block is used to call the initialized function "pidInit" seen in Figure 3.25. The Matlab® code to call this c-function can be seen in Figure 3.27.



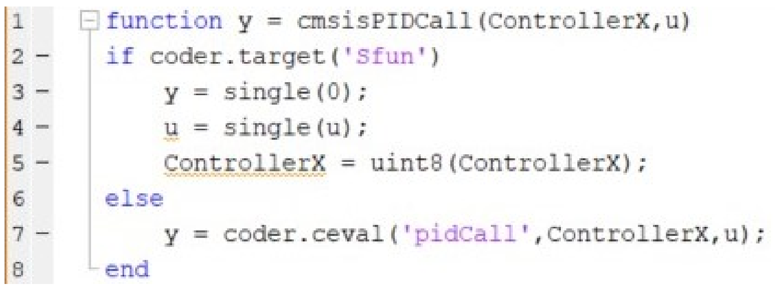
```matlab
function pidInitCall()
if coder.target('Sfun')
else
    coder.ceval('pidInit');
end
```

Figure 3.27: Matlab® code to call the PID initiate c-function

The block used to call the Matlab® code seen in Figure 3.27 can be seen in Figure 3.28.

Figure 3.28: Block used to include Matlab® code for initiation of CMSIS PID controller

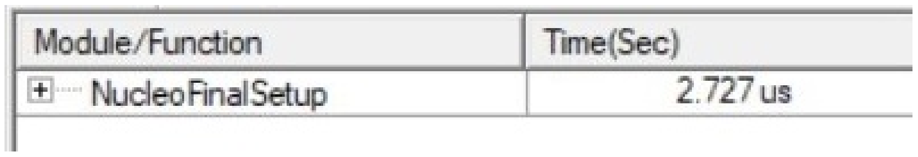
Once the CMSIS PID controllers is initialized the "pidCall" function can be called. This function is called using the Matlab® code seen in Figure 3.29. This Matlab® code is used in the Matlab® function block seen in Figure 3.24.



```
function y = cmsisPIDCall(ControllerX,u)
if coder.target('Sfun')
    y = single(0);
    u = single(u);
    ControllerX = uint8(ControllerX);
else
    y = coder.ceval('pidCall',ControllerX,u);
end
```

Figure 3.29: Matlab® code to call the PID step c-function

The time this PID control realization took to execute can be seen in Figure 3.30.



Module/Function	Time(Sec)
+ NucleoFinalSetup	2.727 us

Figure 3.30: The execution time the CMSIS PID variant used

The results of the 3 PID implementation methods is summarized in Table 3.1. These results is normalised to a 100 μs control period and displayed in a bar graph seen in Figure 3.31.

Table 3.1: PID implementations execution timings

Implementation	**Timing**
Diagram	1.875μs
Simulink® Block	1.889μs
CMSIS PID	2.727μs

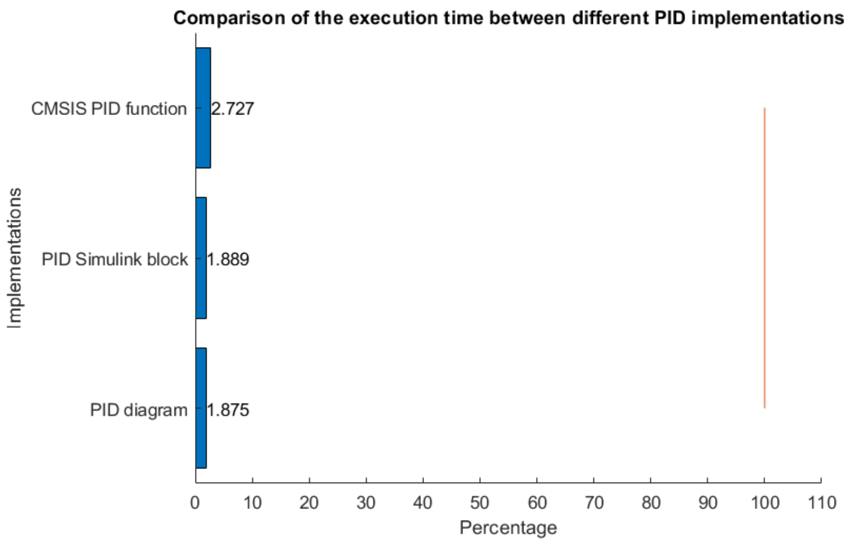
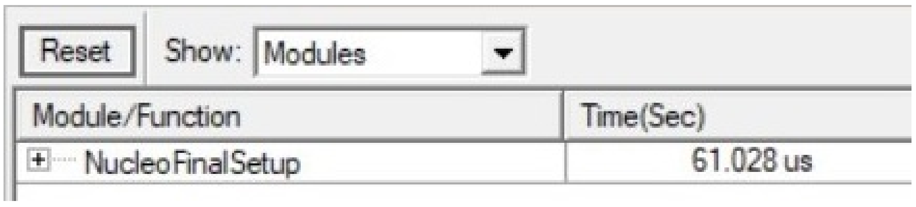


Figure 3.31: Comparison between variants normalized to the required control frequency

Table 3.1 shows the estimated execution timings of the 3 different methods of implementing a PID controller on a MCU. It is important to note that every micro-second that can be spared in the control period is crucial. The results show that the PID diagram has the fastest execution. The reason why the Simulink® PID block takes longer to execute is due to the fact that the Simulink® block includes additional code compare to the PID diagram variant. The additional code is due to the fact that the Simulink® PID block has additional PID settings that can be adjusted. The CMSIS PID software takes even longer, and it is due to the manner the PID control is implemented with their version of for loops. Analysis of the CMSIS implementation of the PID control shows that there is more lines of code used compared to the Simulink® version. Each extra line of code used will lead to longer execution times of the algorithms.

Floating point operations

The default variable type that Matlab®/Simulink® uses is double precision floating point. In the simulation models, a data type should be explicitly defined if a data type other than a double is used. Using a data type that is not supported by the MCU can drastically increase the computational time of the algorithm. This implies that double floating point operations executed on a single point FPU will use more considerably more processing time. The PID control seen in Figure 3.20 is executed using double variables. This execution time of this PID control can be seen in Figure 3.32. The results seen in Figure 3.21 is obtained with the same PID control but with single variables. The results of the single vs double variables are normalised to a 100 μs control period and can be seen in Figure 3.33.



Module/Function	Time(Sec)
NucleoFinalSetup	61.028 us

Figure 3.32: PID execution time with double variables

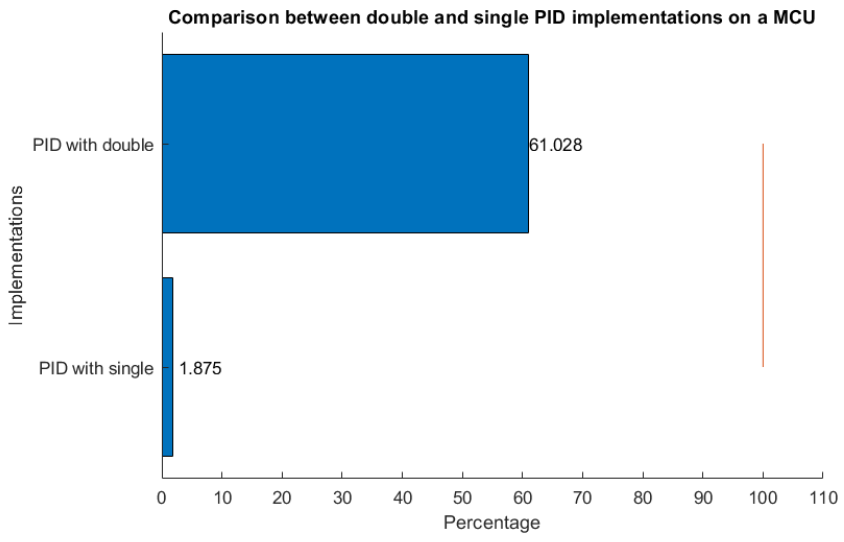


Figure 3.33: Comparison between single vs. double variables normalized to the required control frequency

The results seen in Figure 3.33 shows the negative effects of choosing a data type which is not supported by the MCU. The MCU will do the double point calculations with a single point FPU. Because the double point variable has twice the amount of bit as a single point, the single point will execute each instruction for both the lower and higher word of the double word variable.

MCU settings for performance improvements

The initialisation of a MCU can be a daunting process. Frameworks like Arduino, MBED and the dSPACE Simulink® blocks initialises the MCU to operate at maximum capacity. In cases where frameworks like these are not used it is important to know how to initialise a MCU to operate at maximum capacity. The first and most important peripheral to initialise is the clock. If the clock is operating at maximum frequency, the MCU will most likely operate at its maximum capacity. To demonstrate this the control in Figure 3.20 is executed on a Nucleo-F746ZG at the default clock frequency of 16 MHz. The waveform generated by the GPIO for this result can be seen in Figure 3.34 as the "Default" graph. The pulse width of this can be seen in Figure 3.36 under the default description. Increasing the clock frequency to 216 MHz reduces the time drastically. The result of this can be seen in Figure 3.34 as the Max Clock graph. The pulse timing of this can be seen in Figure 3.36 under the Max clock description. The MCU also have an instruction and data instruction pipeline caches. Enabling these caches will improve the MCU's performance. The waveform and pulse timing of this can be seen in Figure 3.34 and 3.36 respectively under the Cache description. This specific MCU has a tightly coupled memory (TCM) bus interface. Using the TCM interface instead of the advanced eXtensible interface (AXI) interface will also reduce the computational time. The waveform and pulse timing of this can be seen in Figure 3.34 and 3.36 respectively under the TCM description. The results show that in order to obtain maximum performance from the MCU the clock frequency needs to be set to maximum, the instruction and data caches need to be enabled, and the TCM bus needs to be used instead of the AXI bus.

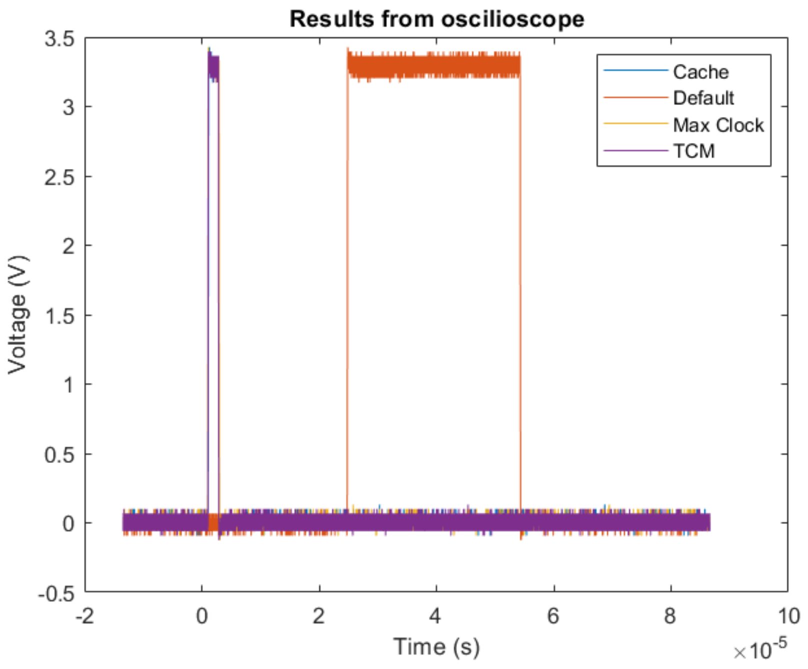
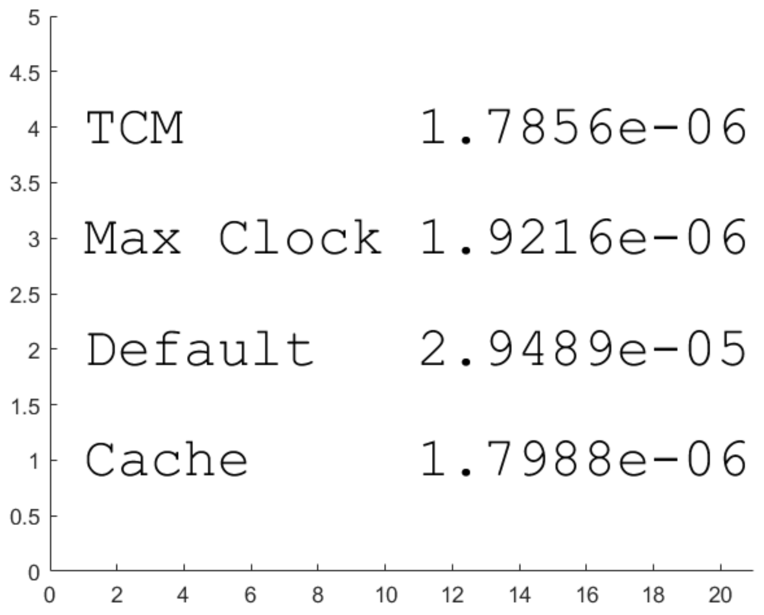


Figure 3.34: Occilioscope results of the Nucleo-F746ZG tests



TCM	1.7856e-06
Max Clock	1.9216e-06
Default	2.9489e-05
Cache	1.7988e-06

Figure 3.35: Pulse width timings of occilioscope results

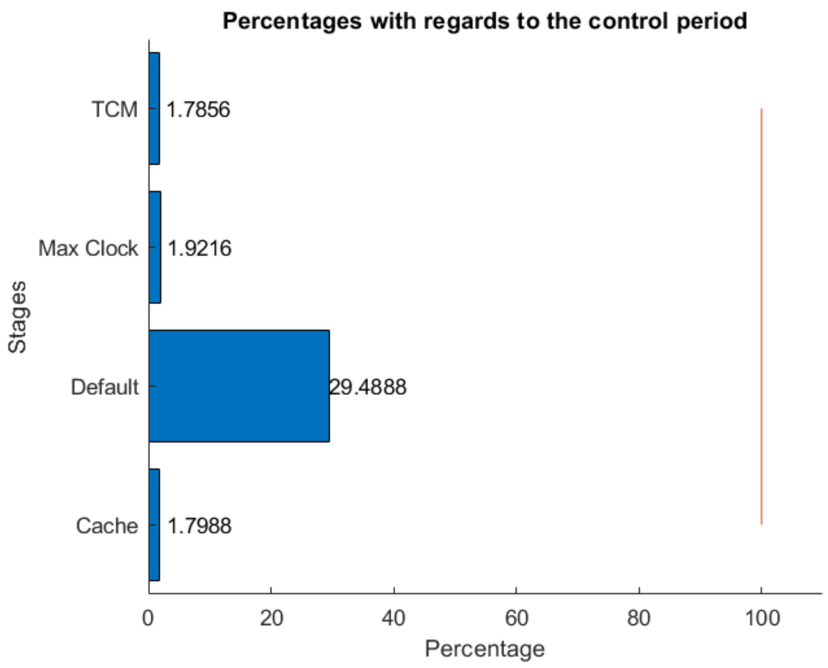


Figure 3.36: Comparison between of the timings normalized to 100 μs

3.4 MCU selection method

In order to use the method, an extensive Simulink® simulation model is needed. The model will need to include the effects that the hardware will have on the plant so that the MCU can compensate for it. All of the hardware choices need to be final at this point, meaning that it will be clear which peripheral will be used to control the hardware. The only question that must remain for this project is the choice of the MCU. Figure 3.37 shows the flow diagram of the method that will be used. The sections that follow will explain in detail what must be done during each step. The basic steps seen in Figure 3.37 is listed below:

1. Obtain information from the simulation.
2. Create a list of MCU candidates.
3. Convert the simulation to code.
4. Simulate the code for the MCU from the list.
5. Use the code to obtain the needed memory and operating frequency requirements and add it to the criteria for the list of MCU candidates.
6. Repeat step 4 in order to verify that the code executes within the time constraint.

Please note that the method is not a linear approach and might require some iteration within the steps.

3.4.1 Obtain information from simulations

Information that needs to be obtained from the simulations is the plant's time constant. This time constant will be used to determine at what frequency the discrete controller needs to operate. The information from the simulation will also determine the data type that will be used and ultimately the MCU.

Simulink®/Matlab® has built-in tools to linearise a plant or model. The reason why the plant needs to be linearised is because the mathematical model of the plant is not linear the controller will not be able to control the plant. This is because the controller is a linear controller and will not be able to control a non-linear plant. The linear controller can, however, control the plant at a specific pre-defined operating point where the non-linear model is linearised. To linearize the plant the open-loop input and open-loop output need to be identified. The open-loop input and open-loop output is simply the input and output of the plant. In the case of the linear actuator example mentioned in Section 3.3.1 the open-loop input needs to be defined as seen in Figure 3.38.

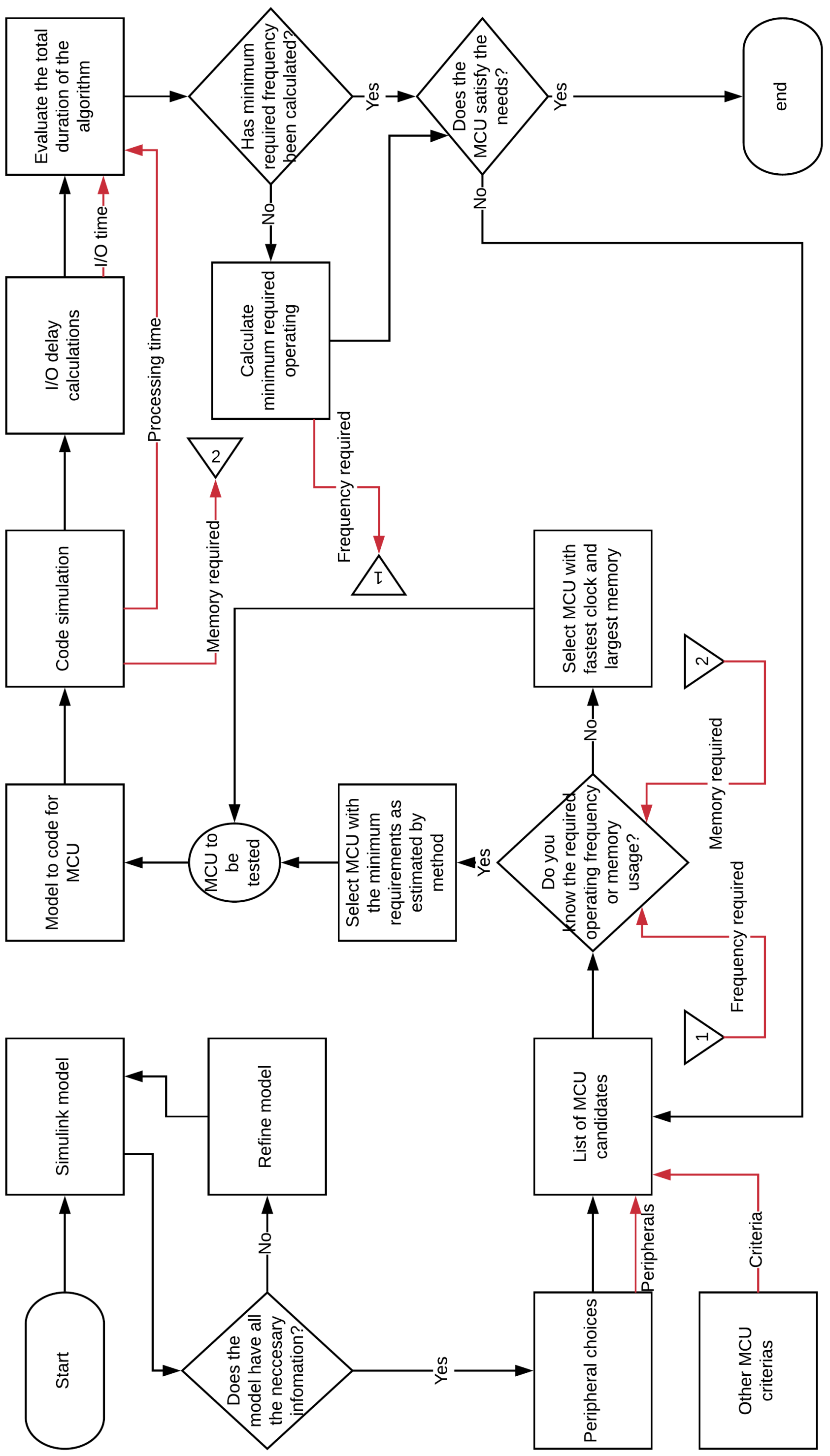

Start
Simulink model
Does the model have all the neccesary infomation?
No
Refine model
Yes
Peripheral choices
Other MCU criterias
Peripherals
Criteria
List of MCU candidates
Do you know the required operating frequency or memory usage?
Frequency required
Memory required
1
2
No
Select MCU with fastest clock and largest memory
Yes
Select MCU with the minimum requirements as estimated by method
MCU to be tested
Model to code for MCU
Code simulation
Memory required
2
Processing time
I/O delay calculations
I/O time
Evaluate the total duration of the algorithm
Has minimum required frequency been calculated?
No
Calculate minimum required operating
Frequency required
1
Yes
Does the MCU satisfy the needs?
No
Yes
end

Figure 3.37: High level flow chart of the designed selection method

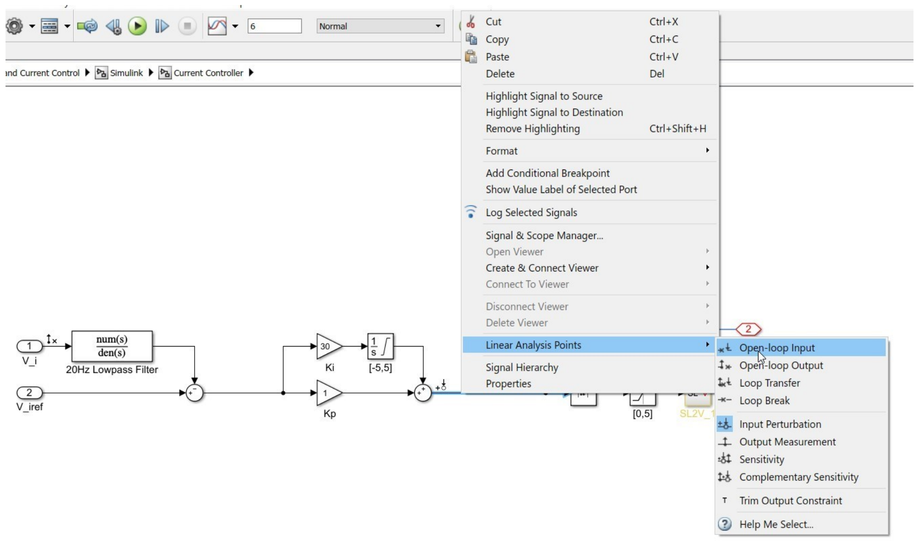


Figure 3.38: Specifying the input for linearisation

The open-loop output needs to be defined as seen in Figure 3.39. Thus the input and output locations for the linearisation can be seen in Figure 3.40. These locations can be specified within the current control loop of the example.

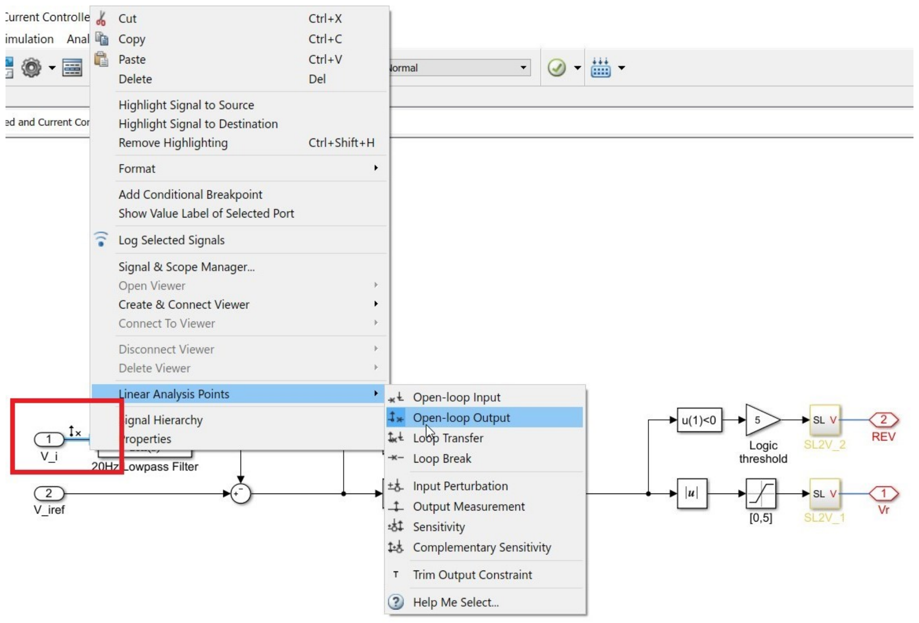


Figure 3.39: Specifying the output for linearisation

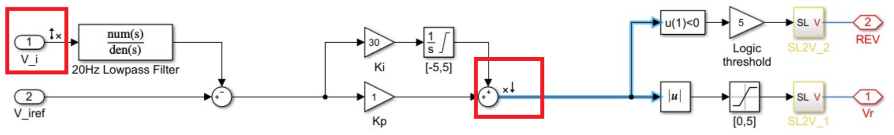


Figure 3.40: Locations of I/O for linearisation

Once the input and output are defined the linearise tool can be launched. The tool can be found under

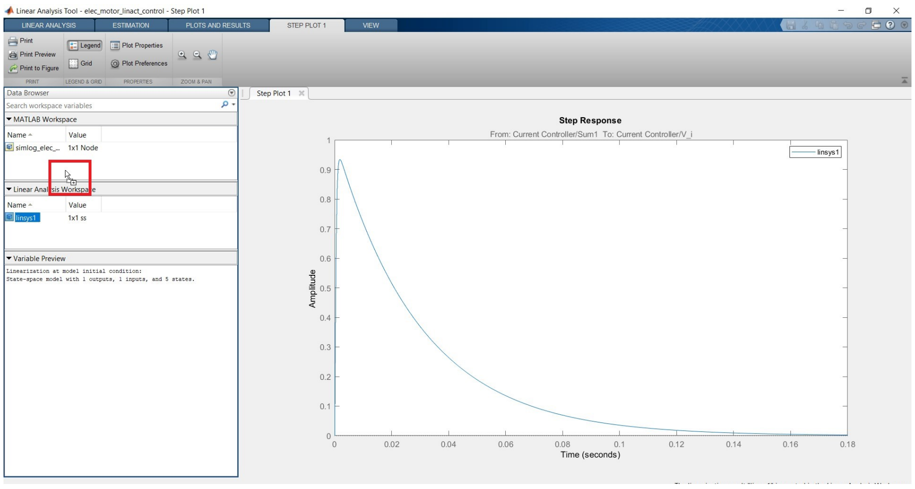

Linear Analysis Tool - elec_motor_linact_control - Step Plot 1
Data Browser
MATLAB Workspace
simlog_elec_... 1x1 Node
Linear Analysis Workspace
linsys1 1x1 ss
Variable Preview
Linearization at model initial condition:
State-space model with 1 outputs, 1 inputs, and 5 states.
Step Response
From: Current Controller/Sum1 To: Current Controller/V_i
linsys1
Amplitude
Time (seconds)

Figure 3.43: Linearisation result

the Analysis>Control Design menu as seen in Figure 3.41.

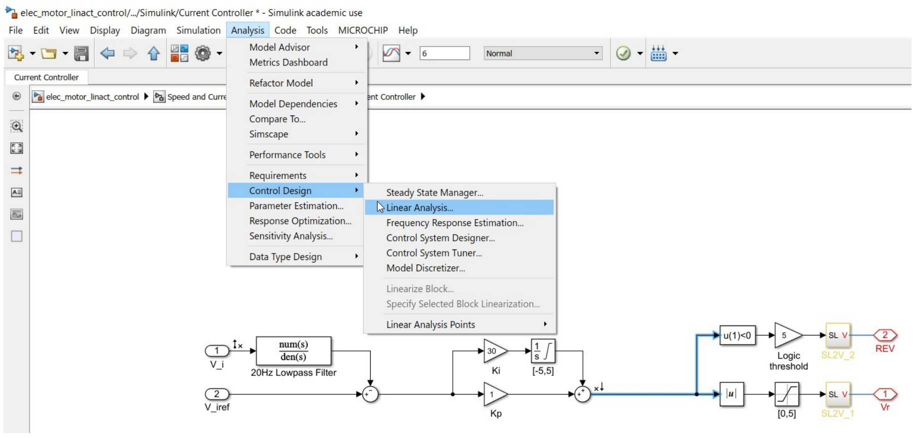


Figure 3.41: Location of the linearise tool

Once the tool is launched the system will be linearised once the step (or any other methods) is selected. This can be seen in Figure 3.42.

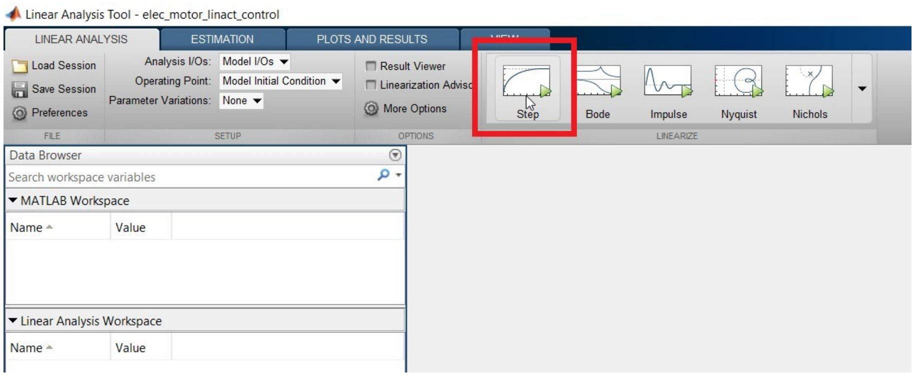


Figure 3.42: Location of the linearise tool

The linearized plant model can now be copied to the Matlab® workspace for further analysis. This step can be seen in Figure 3.43. By clicking and dragging the plant model from the Linear Analysis Workspace section to the Matlab® workspace section within the linearization tool.

Once the plant(in this case the electric linear actuator) is in the Matlab® workspace the necessary controller frequency can be calculated. The Matlab® live script seen in Figure 3.44 was used to extract the time constant from the plant and ultimately determine the minimum controller frequency needed. Please note that these calculations are based on the content discussed in Section 2.5.

The minimum control frequency obtained from the Matlab® script seen in Figure 3.44 can now be used as the sample/control frequency for a digital controller used to control the current of the electric linear actuator.

3.4.2 MCU peripheral choices and I/F electronics

The choice of peripherals needed in a MCU and the I/F that will need to be used are interdependent. The peripherals needed is mostly reliant on how the plant is going to be controlled and how the plant will provide feedback to the controller. It is thus recommended to do an extensive and in-depth trade-off study that will ensure a fast and cost-effective implementation. What is implied by fast is the fact that the delay the peripheral adds to the overall control step execution needs to be considered. Trade-off study does not form part of this selection method, but it is highly recommended to do this because the uncertainty of a peripheral will halter the selection process.

3.4.3 Peripheral requirements

The peripheral requirements must now be calculated. Taking classic control theory into account the resolution of the peripheral needs to be calculated. This can be done by deciding the acceptable margin of error for the controller. Once this margin of error has been decided on the resolution can be calculated by using the margin, the time constant of the plant (with hardware effects taken into account) and the control period. Using these three variables the minimum resolution required to digitise the margin of error can be calculated. The sample rate that the peripherals need to achieve must at least be equal to the control frequency. A communication peripheral needs to be at least be eight times faster than the control frequency because if the data to be communicated is at least 8-bits long it will take the peripheral the whole control period to transfer the data. This is assuming the peripheral specification is given in bits per second.

As seen in Section 2.5, the deviation can be configured as 2% of the voltage output range. The frequency can be chosen as $1/\tau$. Using these values, the resolution can be calculated starting with Equation (4.1).

$$V_{disturbance}(n) = Deviation@Frequency \tag{3.1}$$

The absolutely minimum voltage differences that the ADC needs to be able to pick up can be calculated with equation (3.2):

$$dv_{array}(n) = V_{disturbance}(n \times T) - V_{disturbance}((n-1) \times T)), n \in \mathbb{N} \cup n \geq 1 \tag{3.2}$$

Where: T is the control period and dv_{array} is an array of voltage differences between samples.

$$dv_{min} = min(dv_{array}) \tag{3.3}$$

Taking the minimum value of Equation (3.2)(using Equation (3.3)) we can determine the minimum resolution required to digitize the disturbance. Equation (3.4) to (3.6) are used determine the resolution:

$$numLevels = \frac{V_{sensmax}}{dv_{min}} \tag{3.4}$$

Where: $V_{sensmax}$ = is the maximum voltage value the sensor will provide.

$$n_{resbits} = log_2(numLevels) \tag{3.5}$$

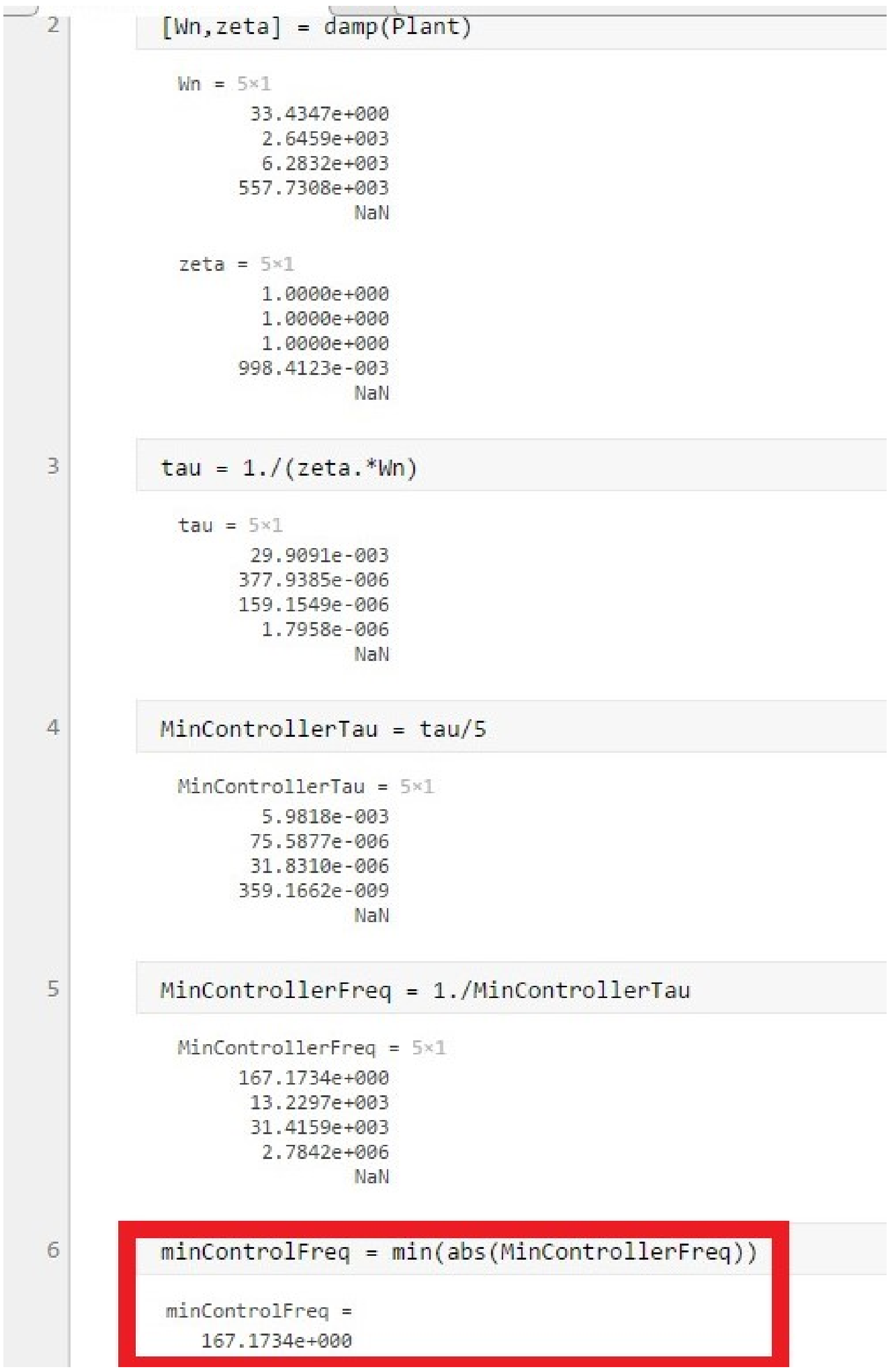
$$n_{minRequired} = ceiling(n_{resbits}) \tag{3.6}$$

Where: $n_{resbits}$ is the resolution in bits and $n_{minRequired}$ is the resolution in bits rounded up to the nearest integer.

3.4.4 MCU criteria

Using the information of the previous sections a criteria list can be compiled. MCU candidates need to adhere to all the items on the list in order to be considered. The criteria list will include the following:

- The peripheral's required, their resolution and sampling rate.
- What data type will be used and whether or not the MCU needs to include a FPU or not.
- The number of GPIO pins required.



```
[Wn,zeta] = damp(Plant)

 Wn = 5×1
        33.4347e+000
         2.6459e+003
         6.2832e+003
       557.7308e+003
                 NaN

 zeta = 5×1
         1.0000e+000
         1.0000e+000
         1.0000e+000
       998.4123e-003
                 NaN

tau = 1./(zeta.*Wn)

 tau = 5×1
        29.9091e-003
       377.9385e-006
       159.1549e-006
         1.7958e-006
                 NaN

MinControllerTau = tau/5

 MinControllerTau = 5×1
         5.9818e-003
        75.5877e-006
        31.8310e-006
       359.1662e-009
                 NaN

MinControllerFreq = 1./MinControllerTau

 MinControllerFreq = 5×1
       167.1734e+000
        13.2297e+003
        31.4159e+003
         2.7842e+006
                 NaN

minControlFreq = min(abs(MinControllerFreq))

minControlFreq =
   167.1734e+000
```

Figure 3.44: Linearisation result

3.4.5 List of MCU candidates

The list of MCU candidates needs to be ordered in descending order of clock speed, flash and RAM. For simulation purposes, the best MCU in this list will be chosen in order to determine the required clock speed, flash and RAM. Some aspects of the simulation results can be scaled. Thus the time that this MCU uses to execute a control step can be used to calculate clock speed. Once the code has been compiled the required flash and RAM can be used to update the criteria list.

3.4.6 Algorithm to code

Simulink® has an extensive library of hardware support packages. The chance that a MCU candidate has a hardware support package is very likely. Using this hardware support packages, the algorithm that needs to be coded can be compiled to c-code that the specific MCU will be able to utilise.

Simulink® can program other embedded targets as well. STMicroelectronics® is another vendor that developed Simulink® blocks. Figure 3.45 provides an example of the blocks used to interface with a STM32 controller.

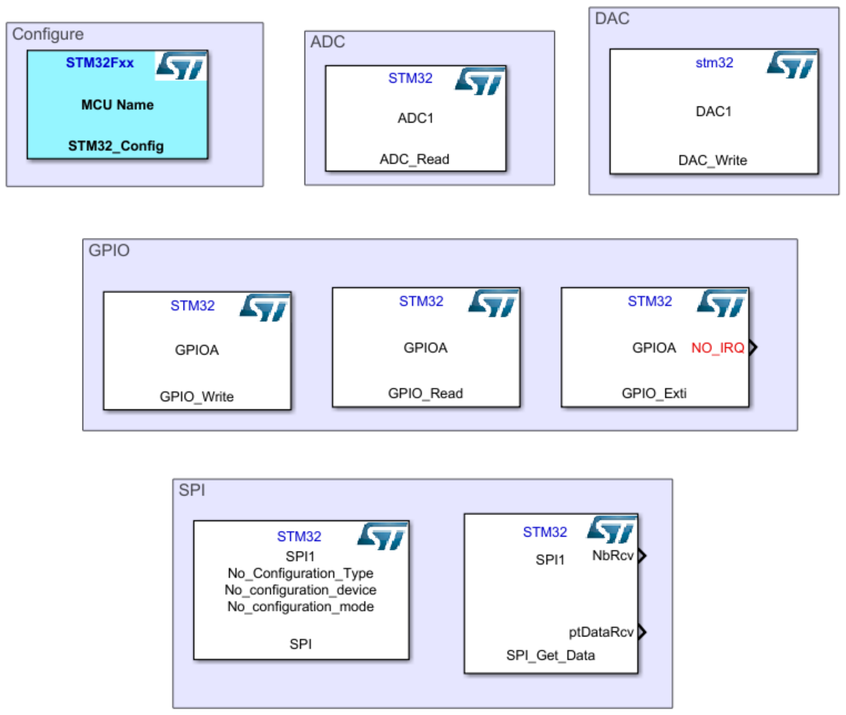
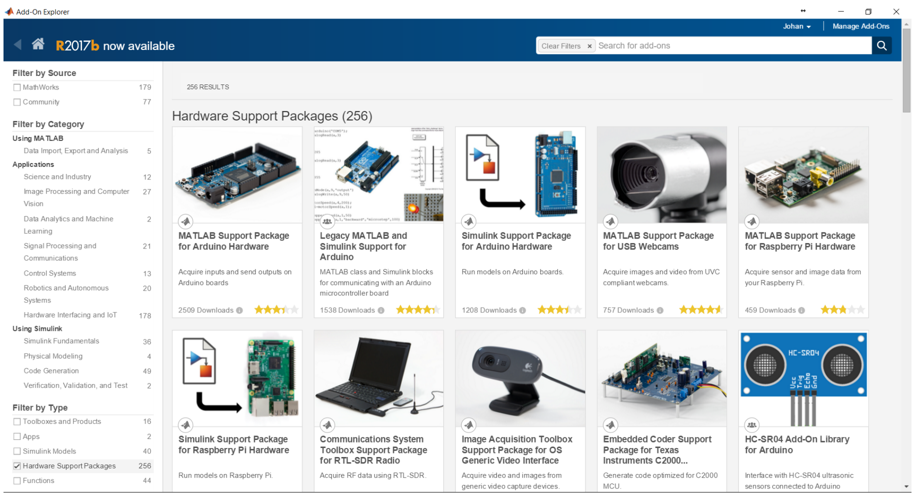


Figure 3.45: STM32 Interface Blocks

Other well known controllers are also supported, these include Arduino, Raspberry Pi, Lego Mindstorm and much more. These controllers can be interfaces with Simulink® by downloading the hardware support packages through the Matlab®/Simulink® add-on manager. Figure 3.46 shows the hardware support page within the add-on manager.



256 RESULTS

Hardware Support Packages (256)

Filter by Source

☐ MathWorks	179
☐ Community	77

Filter by Category

Using MATLAB	
Data Import, Export and Analysis	5
Applications	
Science and Industry	12
Image Processing and Computer Vision	27
Data Analytics and Machine Learning	2
Signal Processing and Communications	21
Control Systems	13
Robotics and Autonomous Systems	20
Hardware Interfacing and IoT	178
Using Simulink	
Simulink Fundamentals	36
Physical Modeling	4
Code Generation	49
Verification, Validation, and Test	2

Filter by Type

☐ Toolboxes and Products	16
☐ Apps	2
☐ Simulink Models	40
✓ Hardware Support Packages	256
☐ Functions	44

MATLAB Support Package for Arduino Hardware — Acquire inputs and send outputs on Arduino boards — 2509 Downloads

Legacy MATLAB and Simulink Support for Arduino — MATLAB class and Simulink blocks for communicating with an Arduino microcontroller board — 1538 Downloads

Simulink Support Package for Arduino Hardware — Run models on Arduino boards. — 1208 Downloads

MATLAB Support Package for USB Webcams — Acquire images and video from UVC compliant webcams. — 757 Downloads

MATLAB Support Package for Raspberry Pi Hardware — Acquire sensor and image data from your Raspberry Pi. — 459 Downloads

Simulink Support Package for Raspberry Pi Hardware — Run models on Raspberry Pi.

Communications System Toolbox Support Package for RTL-SDR Radio — Acquire RF data using RTL-SDR.

Image Acquisition Toolbox Support Package for OS Generic Video Interface — Acquire video and images from generic video capture devices.

Embedded Coder Support Package for Texas Instruments C2000... — Generate code optimized for C2000 MCU.

HC-SR04 Add-On Library for Arduino — Interface with HC-SR04 ultrasonic sensors connected to Arduino

Figure 3.46: Simulink® Hardware Support [5]

Using STMicroelectronics® in conjunction with Matlab®/Simulink® allows one to program the controller to the MCU. STMicroelectronics® also allows the user to program the MCU by means of a third party application. Additional evaluations can be done to the converted control program when using third party applications, like using the ISS of said application.

3.4.7 Code simulation

The code simulation is done by utilising an ISS. Instructions and the properties thereof are explained in Section 2.3. An ISS is a simulation program which accumulates the cycle count as the program is being virtually executed. The MCU will have a constant clock pulse per cycle. Using this information with the total cycle count and operating frequency, the real-time duration in seconds can be calculated. Keil's MDK-ARM V5 IDE has a simulator function which can be used for ARM MCUs. The simulator in combination with the performance analyser can be used to obtain execution timings of the code.

3.4.8 Peripheral delay calculations

The flow diagram seen in Figure 3.47 is used to determine how to calculate the delay the peripheral will cause. The figure has three possible outcomes which involve calculating delays for a peripheral as is, a peripheral using a DMA controller and an external peripheral.

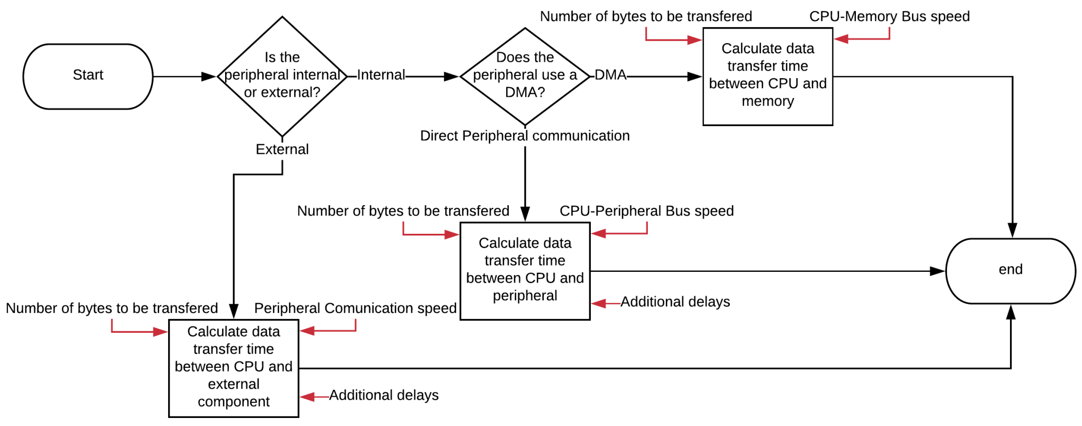

Start
Is the peripheral internal or external?
Internal
External
Does the peripheral use a DMA?
DMA
Direct Peripheral communication
Number of bytes to be transfered
CPU-Memory Bus speed
Calculate data transfer time between CPU and memory
Number of bytes to be transfered
CPU-Peripheral Bus speed
Calculate data transfer time between CPU and peripheral
Additional delays
end
Number of bytes to be transfered
Peripheral Comunication speed
Calculate data transfer time between CPU and external component
Additional delays

Figure 3.47: I/O delay calculations flow chart

Peripheral communication calculations

To start the function of a peripheral, the necessary bit in the peripheral's control register needs to be set. Setting this bit is done by transferring a byte between the CPU and the peripheral. The bus between the

CPU and the peripheral will be clocked at a pre-defined speed. The peripheral might have some delay time before the peripheral responds with the necessary value which the CPU is waiting for, before continuing with the program. In the case of an ADC, there is the sampling time and the conversion time that need to be taken into account. Thus the delay of a peripheral can be calculated in the following manner:

1. The number of bytes transferred between the CPU and peripheral for the initialisation of peripheral.

2. The number of cycles/ the duration of any delays the peripheral might have.
3. The number of bytes being transferred between the CPU and the peripheral after the peripheral has executed its task.

Equation (3.7) can be used to calculate the timing of item 1 and 3. Equation (3.8) can be used to convert the number of cycles to a timing in seconds. This equation might be used in step 2 if the peripheral's specification is given in cycles. The accumulative time of steps 1 to 3 can be seen as the peripheral's delay.

$$t_{BusDataTransfer} = n_{Bytes}/f_{BusSpeed} \tag{3.7}$$

$t_{BusDataTransfer}$ is the time which the number of bytes (n_{Bytes}) will take to be transferred at specific bus speed ($f_{BusSpeed}$).

$$t_{cycle2time} = n_{Cycles}/f_{Operating} \tag{3.8}$$

$t_{cycle2time}$ is the time which a number of cycles (n_{Cycles}) will take at specific operating frequency ($f_{Operating}$).

DMA calculations

The use of a DMA controller will decrease the delay of a peripheral. Consider a peripheral that is used as an input (like an ADC). The use of a DMA will only use step 3 in the process mentioned in the previous section. The trigger that is set up for the peripheral will be used to execute steps 1 and 2 without the CPU being halted. Once the peripheral has reached step 3 the DMA controller will transfer the response value to a pre-defined byte(s) in the memory. The program will transfer the data from the memory to the CPU once the information from the peripheral is needed. The DMA can also be used to serve data to the peripheral via the memory. This implies that the data that the peripheral will use can be written to a section in the memory. The DMA controller, the location of the data in the memory and a hardware trigger will ensure that the peripheral will execute its function.

The time delay of a peripheral using a DMA controller can be calculated using Equation (3.7). The bus speed in this equation will be the speed of the bus between the memory and the CPU.

External component calculation

Calculating the time a external component will use to execute it's function can be done by using the flow diagram seen in Figure 3.48. The peripheral will also use the steps seen in Section 3.4.8 under the heading **Peripheral communication calculations**. The additional time delay that a serial communication peripheral will have in step 2 can be calculated using Equation (3.9). The additional time delay that a parallel device adds during step 2 can be calculated with Equation (3.10).

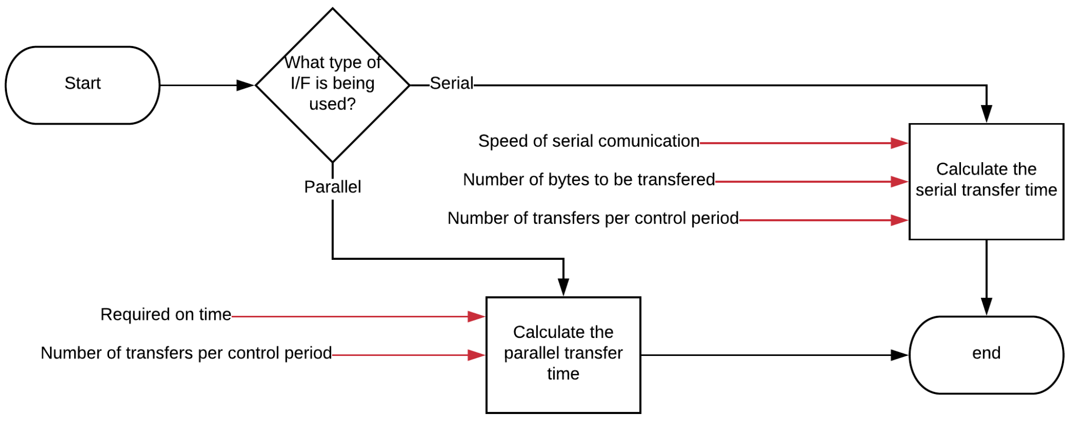

Start
What type of I/F is being used?
Serial
Parallel
Speed of serial comunication
Number of bytes to be transfered
Number of transfers per control period
Calculate the serial transfer time
Required on time
Number of transfers per control period
Calculate the parallel transfer time
end

Figure 3.48: I/O delay calculations of external peripheral flow chart

$$t_{serialCom} = n_{bits}/f_{baudrate} \times n_{transfersPerControl} \tag{3.9}$$

$t_{serialCom}$ is the time it takes for a number of bits (n_{bits}) to be transferred with the peripheral device at a specific baudrate ($f_{baudrate}$) for a $n_{transfersPerControl}$ number of times.

$$t_{parallelCom} = n_{transfersPerControl} \times t_{reqOnTime} \tag{3.10}$$

$t_{parallelCom}$ is the time it takes for the MCU to transfer the data to the external device for a $n_{transfersPerControl}$ number of times. The $t_{reqOnTime}$ is the time required for the data to be read by the external device.

3.4.9 Evaluate total duration

The estimated timings of the input, processing and output stages need to be accumulated. The accumulated value can then be used to compare it to the control period. If the accumulated value is less than the control period the MCU will control the plant and no overruns will occur. Step 5 seen in the introduction of Section 3.4 calculates additional parameters for the criteria list. Equation (3.12) can be used to calculate this parameter. The required memory parameter information can be obtained from the IDE's build log after a successful compilation of the code.

$$Workload\% = \frac{T_{exetime}}{T_{control}} \times 100\% \tag{3.11}$$

$$f_{minRequired} = Workload\% \times f_{operatingFrequency} \tag{3.12}$$

3.5 Conclusion

The flow chart seen in Figure 3.37 gave a detailed overview of the steps mentioned in Section 3.4. Section 3.4.1 to 3.4.9 were used to explain how the flow diagram is used in much more detail. It is important to note that step 6 in the steps mentioned in Section 3.4 is an iterative step and can be evaluated until all requirements are satisfied. The whole method is an iterative procedure. The procedure will not provide the perfect solution because of the uniqueness of each MCU and the vendor. The procedure will however give a very clear answer whether or not the MCU under investigation will be able to execute the intended control application.

Chapter 4

Method implementation

The controller selection method designed in Chapter 3 will be implemented in this Chapter. The selection method will commence after a description of the plant, and the controller that will be used is given. This Chapter and the choice final choice of MCU will serve as the verification of this dissertation.

4.1 Plant

A PMSM at the North-West University (NWU)[7] cannot operate at rated speed for long periods of time because of heat generated by the motor. The developed system is called the TWINS motors. A source of heat within this system is the ball bearings between the rotor and stator. To reduce heat generated in the system, it was decided to fit the TWINS with AMBs.

The layout of the system is comprised of a rotor, stator and two radial, hetero-polar, four pole electromagnetic bearings. The layout of the PMSM with AMB can be seen in Figure 4.1.

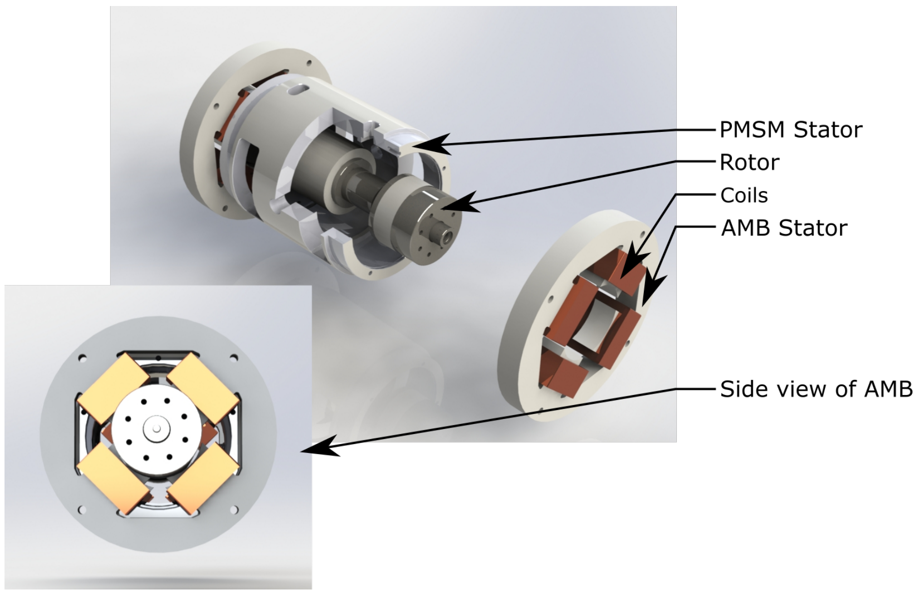


Figure 4.1: PMSM with AMB

The AMB system consists of the following components:

- The two radial AMBs is comprised of two four-pole hetero-polar electromagnetic stators.
- 8 Current controlled power amplifier (PA)s.
- four position sensors to measure the rotor's radial position.
- The rotor to be levitated.

The peripheral requirements for the controller can be derived from the list above. These peripherals can be determined by establishing how the sensors, power and controllable source are interfaced with the controller. The AMB system has already been designed[?] The design of the AMB lead to the choice of

PAs. A Eddy current sensor was chosen to act as the rotor position feedback. The peripherals required are:

- 8 DACs which can output 0 to 10 volts for the power amplifiers' current reference.
- 4 ADCs which will read the position sensors' feedback.

The PA that was chosen will need a DC voltage input between -10V and 10V. This voltage is used as input and converted to a current. This input voltage has a 1:1 ratio with the current which is internally controlled. The negative side of the PA's spectrum will not be used as this will influence the flow of magnetic flux within the AMB. Thus an analog signal between 0 and 10V needs to be generated. The eddy current sensor will be used to provide feedback of the rotor's position. The eddy current sensor has an output of 0V to 10V which translate to 0mm to 1mm between the sensor and the surface in question.

4.2 Discrete Controller

The control loop seen in Figure 4.2 has been categorised into four subsystems. The entirety of the control loop has been used by multiple masters' dissertations at the NWU [18, 7, 13]. The Figure has been modified for presentation purposes. The first subsystem called "ADC2Sensor" seen in Figure 4.2 is used to obtain the position of the rotor from the sensor using ADC's. The value obtained by the sensors needs to be converted into useful information for the control. The PID control uses this distance value (in meters) as input. The PID block can be seen in Figure 4.3. This block calculates the error and then implements a standard PID control law. The output of the PID law, in this case, is the command current. A bias current is added to the command current and sent to the next part of the loop. The low pass filter (LPF) sub-system filters unwanted audible frequencies out of the biased command current. The last subsystem that the controller will need to execute is the conversion of the biased command current to a digital value. The DACs subsystem will use this value to convert it into a voltage which the PA will use as a reference. The PA will use its own internal control loop to provide the AMB with the biased command current. Each sensor input controls 2 PAs.

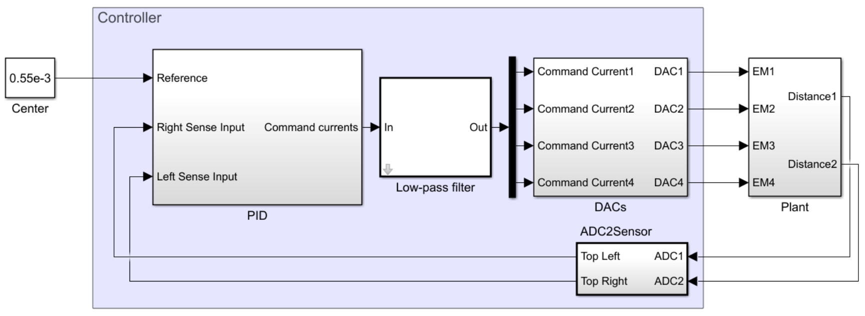


Figure 4.2: Screen-shot of control loop in Simulink®

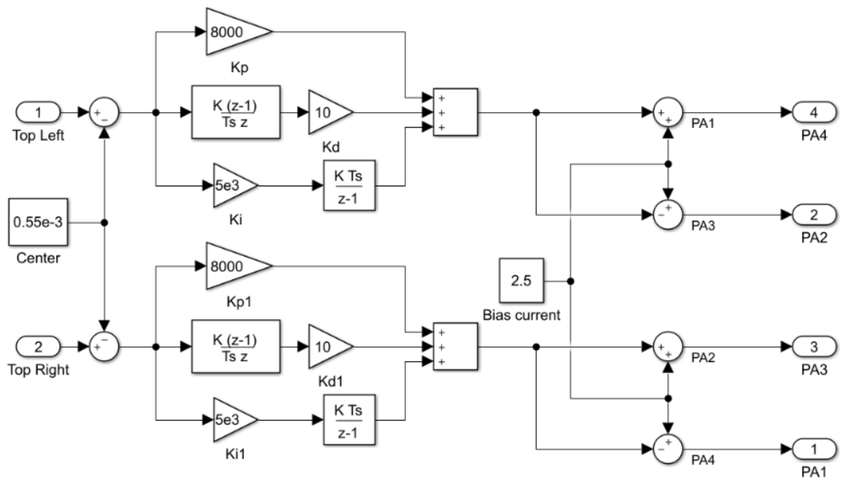


Figure 4.3: Screen-shot of PID Subsystem in Simulink®

4.3 Implementing Method

In this section the steps mentioned in Section 3.2.1 will be followed to determine the time a MCU will use to execute the PID control described in Section 4.2 on the PMSM with AMBs mentioned in Section 4.1. This section will also serve as verification for this dissertation.

The first step is to obtain information from the simulations to calculate the resolution needed for the peripherals and the necessary control frequency. In most cases, some interface electronics is needed which will protect and translate the inputs and outputs to the correct measures. The interface electronics is chosen alongside the MCU.

4.3.1 Simulation information

The necessary simulation information from x is extracted and provided here. Simulations with a mass-spring-damper system we found that there will be a max disturbance of 105N will displace the rotor with 0.2mm occurring at a frequency of 500 Hz. According to control theory book [3], a digital controller needs to sample and execute the control at least five times faster than the plant's time constant. Faster than a factor of 5 will improve the MCU's ability to control the plant. The control of this plant has been previously verified by [7]. A control frequency of 10 KHz was used to control this plant [7]. The AMB structure has been designed and a appropriate PA and Eddy current sensor has been chosen [7]. From this information, the peripherals and I/F electronics can be chosen.

4.3.2 Interface electronics and peripheral choices

The interfaces that is required for the MCU is mentioned in Section 4.1. The STM32 MCUs from STMicroelectronics has GPIO pins operating at 3.3 V. This implies that I/F electronics will need to be used to convert the input and output voltages of the MCU to the appropriate levels.

It is not very common to find a MCU with 8 DACs. Thus other technologies to consider is:

- SPI DACs
- I2C DACs
- pulse width modulation (PWM) with filter to convert Duty cycle to an average value.
- Parallel DACs

Considering the advantages and disadvantages of each of the above methods to convert digital data into analog values (see Section 2.8), the SPI DACs were chosen.

The behaviour of the eddy current sensor and the voltage outputs that it will provide can degrade the effective resolution of the ADC. The surface that the eddy current sensor is going to sense is rounded. This will cause DC voltage offsets. However, the linear response of the sensor is not affected. An I/F circuit using digital pots was designed to overcome this problem. This circuit in its reset state passes the voltage through to the ADC. The circuit and parts used for this I/F circuit can be seen in Appendix B. The MCU will then use the AMBs to determine the minimum and maximum voltage output of the sensors by activating the appropriate AMBs. The minimum value will be used to subtract the minimum DC voltage offset from the sensor input using a digital pot with a subtracting op-amp configuration. The output of the subtracting op-amp will be amplified using a combination of the minimum and maximum voltage. The calculation procedure will look like the diagram seen in Figure 4.4. The I/F circuit used for the ADCs can be seen in Figure 4.5.

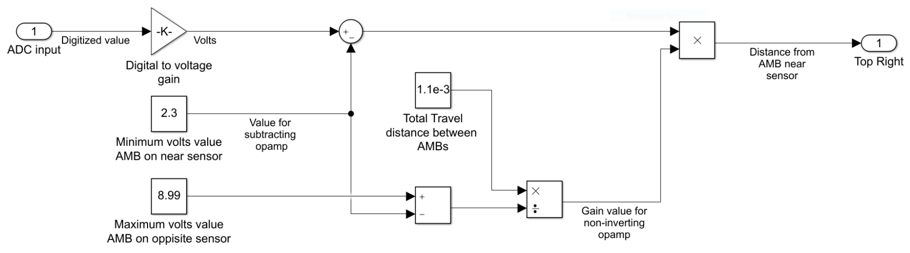

1
ADC input
Digitized value
-K-
Volts
Digital to voltage gain
2.3
Minimum volts value AMB on near sensor
Value for subtracting opamp
8.99
Maximum volts value AMB on oppisite sensor
1.1e-3
Total Travel distance between AMBs
Gain value for non-inverting opamp
Distance from AMB near sensor
1
Top Right

Figure 4.4: I/F calibration method

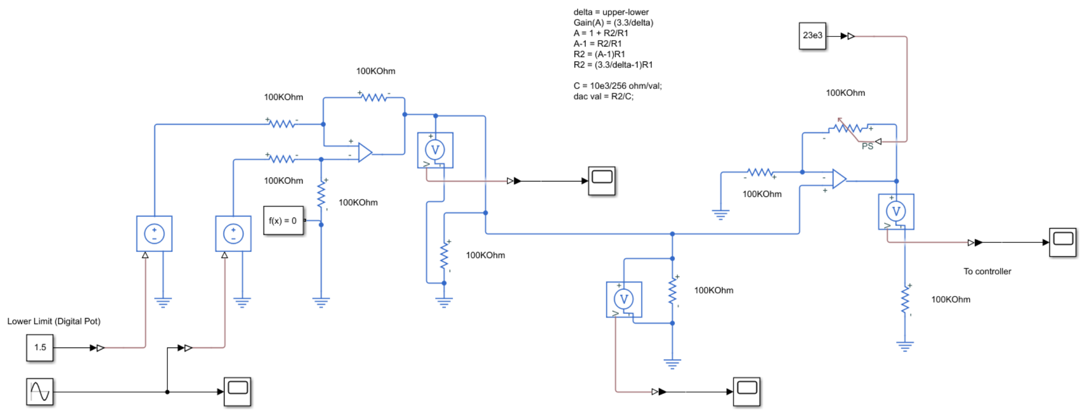


Figure 4.5: ADC I/F circuit

The output to the PAs will be realised 4 dual channel SPI DACs. There will thus be 8 SPI components that will be addressed (4 dual channel 100 KOhm digital pots and four dual-channel DACs).

An LCD keypad shield will be used as user input (UI). This shield needs an additional channel of an ADC and 7 GPIO pins to communicate with. The decoder will be used for the chip select (CS) channels needs 4 GPIO pins, 3 to address the 8 SPI components and 1 to enable/disable the CS channels.

4.3.3 Peripheral requirements

The simulations produced a disturbance occurring at a frequency of 500Hz for a G6 unbalanced load[7]. In essence, this can be seen as the plant's time constant.

The required resolution for the peripherals needs to be calculated to keep the rotor within a 0.01 mm margin of the nominal airgap at a 500 Hz. This margin was chosen according to the control theory book [3]. Taking the designed sampling frequency of 10 kHz, the sensor will translate the disturbance to a voltage output represented by Equation 4.1. This equation is derived from Equation 3.1.

$$V_{disturbance}(n) = 10_{v/mm} * 0.01 * cos\left(\frac{2\pi 500n}{T_{controlPeriod}}\right), n \in \mathbb{N} \tag{4.1}$$

The voltage output of this equation can be seen in Figure 4.6. Figure 4.6 shows the sample and hold characteristics of the ADC and Equation 4.1 is defined for positive integers. Equation 3.2 to 3.6 is used to calculated the resolution needed.

The same method is used for the DAC's. The input to the equations for the DAC was calculated by the PID control. Equation 3.2 to 3.6 was implemented for both the ADC and DAC in Simulink® and the diagram used can be seen in Figure 4.7. The sensor gain block seen in this figure was changed to 1 for the calculations of the DAC. The resolutions required is listed below (See Figure 4.8 for results from equations):

- a ADC of at least 11 bits resolution sampling at a frequency of 10kHz.
- a DAC of at least 12 bits resolution updating it's value at a frequency of 10kHz.

A higher resolution peripheral will lead to a tighter control margin. A tighter control margin implies that the rotor will have a smaller deviation from the centre/reference point. This bare minimum calculation of the peripherals does not take into account the noise that might be added into the system. This is based on the smallest difference in voltage the peripheral needs to pick up or produce within a control step. These equations can be seen as the minimum resolution needed to keep the rotor within a 0.01mm margin from the reference.

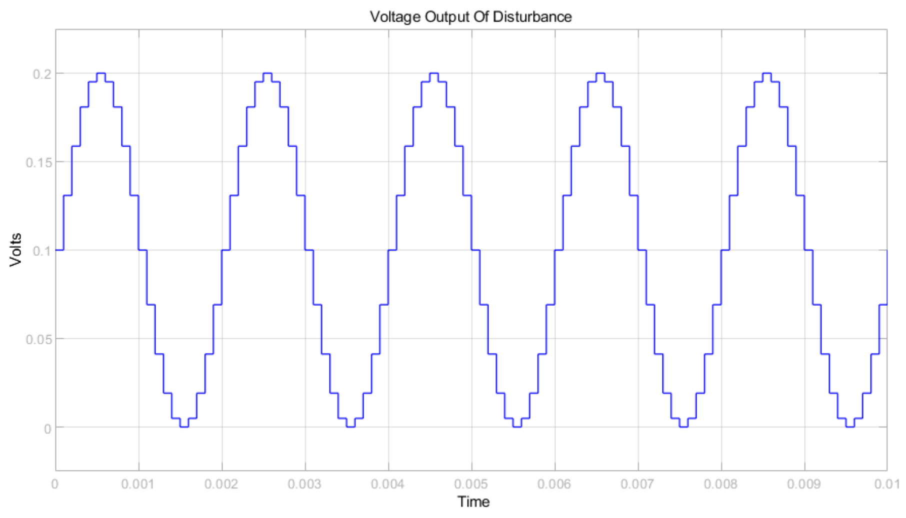


Figure 4.6: Voltage output of 0.01mm disturbance

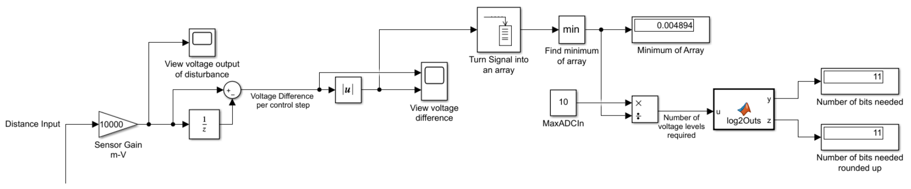

Distance Input
10000
Sensor Gain m-V
View voltage output of disturbance
Voltage Difference per control step
|u|
View voltage difference
Turn Signal into an array
min
Find minimum of array
0.004894
Minimum of Array
10
MaxADCIn
Number of voltage levels required
log2Outs
11
Number of bits needed
11
Number of bits needed rounded up

Figure 4.7: Resolution Calculations

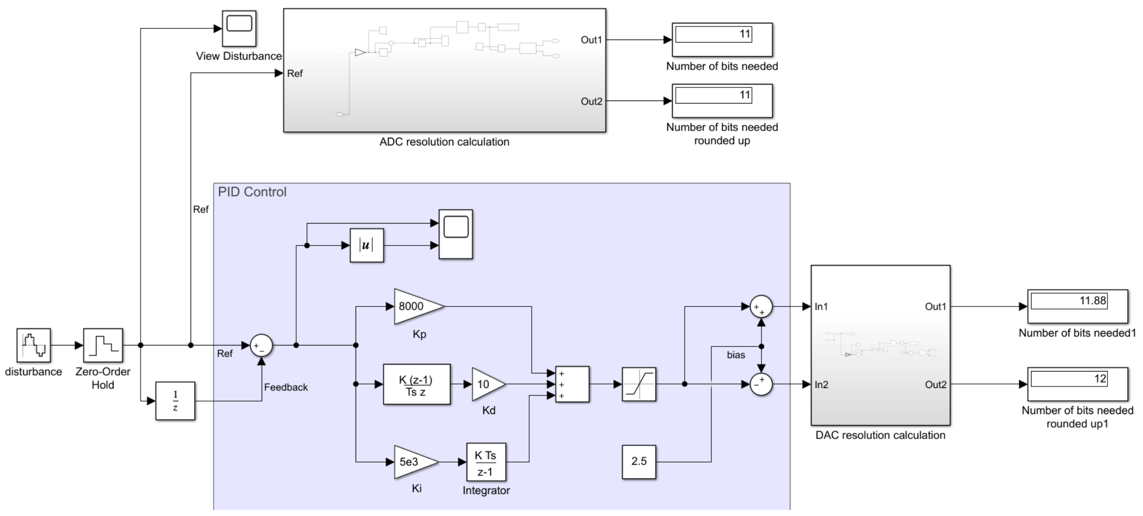


Figure 4.8: Resolution Calculations Method

4.3.4 MCU criteria

The MCU must be able to do floating point operations. Double floating point precision is not necessary (see Section 3.3.2 for reason why it is not necessary). This equates to the following specifications for the MCU:

- 12-bit ADC with 5 channels (5 alternative function (AF) pins)
- SPI channel (2 AF pins for clock and data and 4 GPIO pins for CS decoder)
- 7 Digital Out pins for LCD (7 GPIO pins)

A MCU with 30 pins or more where 19 of these pins is function specific outputs and 11 are for GPIO purposes.

4.3.5 Shortlisting

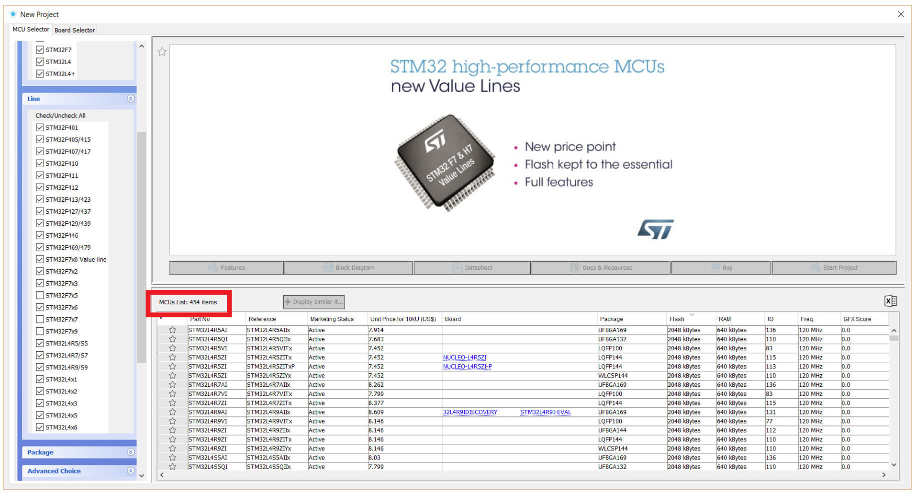
Using the criteria of Section 4.3.4 a list of possible MCU can be compiled. From the STMicroelectronics website the following MCU lines has single precision FPUs:

- STM32F4
- STM32F7
- STM32L4
- STM32L4+

From this list, the F7 line has a few MCUs that have double precision FPUs which need to be excluded. These rages are listed below::

- STM32F7x5
- STM32F7x7
- STM32F7x9

There are 454 different STMicroelectronics MCUs that meets our criteria (see Figure 4.9). For testing purposes, the MCU(s) operating at max frequency in this list will be considered. The test will provide us with an executed time which can be used in relation to the operating frequency to determine the minimum operating frequency needed of the MCU. The next step is to narrow down the MCUs by displaying all the MCUs which can operate at the max frequency possible. This gives a narrowed down list of only 60 MCUs. Note that these 60 MCUs are of the STM32F7 line(See Figure 4.10).



- [✓] STM32F7
- [✓] STM32L4
- [✓] STM32L4+

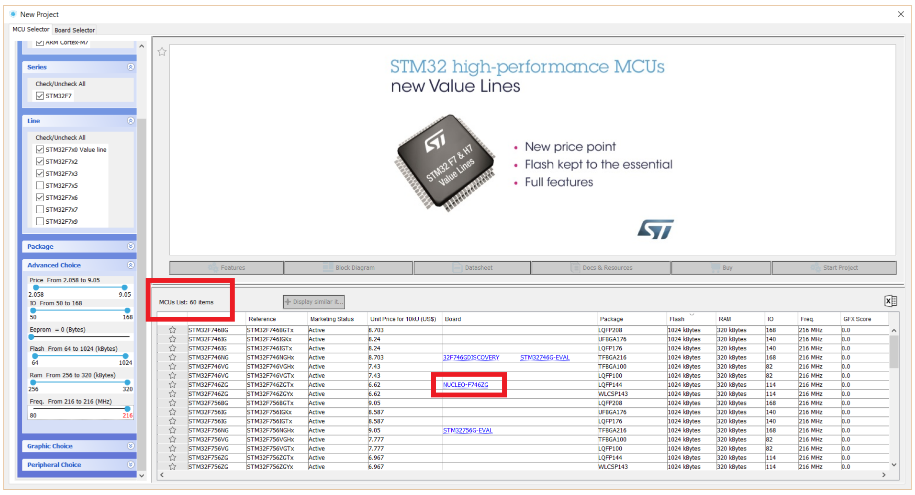
Line

Check/Uncheck All
- [✓] STM32F401
- [✓] STM32F405/415
- [✓] STM32F407/417
- [✓] STM32F410
- [✓] STM32F411
- [✓] STM32F412
- [✓] STM32F413/423
- [✓] STM32F427/437
- [✓] STM32F429/439
- [✓] STM32F446
- [✓] STM32F469/479
- [✓] STM32F7x0 Value line
- [✓] STM32F7x2
- [✓] STM32F7x3
- [ ] STM32F7x5
- [✓] STM32F7x6
- [ ] STM32F7x7
- [ ] STM32F7x9
- [✓] STM32L4R5/S5
- [✓] STM32L4R7/S7
- [✓] STM32L4R9/S9
- [✓] STM32L4x1
- [✓] STM32L4x2
- [✓] STM32L4x3
- [✓] STM32L4x5
- [✓] STM32L4x6

MCUs List: 454 items

*	Part No	Reference	Marketing Status	Unit Price for 10kU (US$)	Board		Package	Flash	RAM	IO	Freq.	GFX Score
☆	STM32L4R5AI	STM32L4R5AIIx	Active	7.914			UFBGA169	2048 kBytes	640 kBytes	136	120 MHz	0.0
☆	STM32L4R5QI	STM32L4R5QIIx	Active	7.683			UFBGA132	2048 kBytes	640 kBytes	110	120 MHz	0.0
☆	STM32L4R5VI	STM32L4R5VITx	Active	7.452			LQFP100	2048 kBytes	640 kBytes	83	120 MHz	0.0
☆	STM32L4R5ZI	STM32L4R5ZITx	Active	7.452	NUCLEO-L4R5ZI		LQFP144	2048 kBytes	640 kBytes	115	120 MHz	0.0
☆	STM32L4R5ZI	STM32L4R5ZITxP	Active	7.452	NUCLEO-L4R5ZI-P		LQFP144	2048 kBytes	640 kBytes	113	120 MHz	0.0
☆	STM32L4R5ZI	STM32L4R5ZIYx	Active	7.452			WLCSP144	2048 kBytes	640 kBytes	110	120 MHz	0.0
☆	STM32L4R7AI	STM32L4R7AIIx	Active	8.262			UFBGA169	2048 kBytes	640 kBytes	136	120 MHz	0.0
☆	STM32L4R7VI	STM32L4R7VITx	Active	7.799			LQFP100	2048 kBytes	640 kBytes	83	120 MHz	0.0
☆	STM32L4R7ZI	STM32L4R7ZITx	Active	8.377			LQFP144	2048 kBytes	640 kBytes	115	120 MHz	0.0
☆	STM32L4R9AI	STM32L4R9AIIx	Active	8.609	32L4R9IDISCOVERY	STM32L4R9I-EVAL	UFBGA169	2048 kBytes	640 kBytes	131	120 MHz	0.0
☆	STM32L4R9VI	STM32L4R9VITx	Active	8.146			LQFP100	2048 kBytes	640 kBytes	77	120 MHz	0.0
☆	STM32L4R9ZI	STM32L4R9ZIJx	Active	8.146			UFBGA144	2048 kBytes	640 kBytes	112	120 MHz	0.0
☆	STM32L4R9ZI	STM32L4R9ZITx	Active	8.146			LQFP144	2048 kBytes	640 kBytes	110	120 MHz	0.0
☆	STM32L4R9ZI	STM32L4R9ZIYx	Active	8.146			WLCSP144	2048 kBytes	640 kBytes	110	120 MHz	0.0
☆	STM32L4S5AI	STM32L4S5AIIx	Active	8.03			UFBGA169	2048 kBytes	640 kBytes	136	120 MHz	0.0
☆	STM32L4S5QI	STM32L4S5QIIx	Active	7.799			UFBGA132	2048 kBytes	640 kBytes	110	120 MHz	0.0

Figure 4.9: List of MCU's meeting our criteria



ARM Cortex-M7

Series
Check/Uncheck All
[✓] STM32F7

Line
Check/Uncheck All
[✓] STM32F7x0 Value line
[✓] STM32F7x2
[✓] STM32F7x3
[ ] STM32F7x5
[✓] STM32F7x6
[ ] STM32F7x7
[ ] STM32F7x9

Package

Advanced Choice
Price From 2.058 to 9.05 (2.058 – 9.05)
IO From 50 to 168 (50 – 168)
Eeprom = 0 (Bytes)
Flash From 64 to 1024 (kBytes) (64 – 1024)
Ram From 256 to 320 (kBytes) (256 – 320)
Freq. From 216 to 216 (MHz) (80 – 216)

Graphic Choice

Peripheral Choice

MCUs List: 60 items

	Reference	Marketing Status	Unit Price for 10kU (US$)	Board		Package	Flash	RAM	IO	Freq.	GFX Score
STM32F746BG	STM32F746BGTx	Active	8.703			LQFP208	1024 kBytes	320 kBytes	168	216 MHz	0.0
STM32F746IG	STM32F746IGKx	Active	8.24			UFBGA176	1024 kBytes	320 kBytes	140	216 MHz	0.0
STM32F746IG	STM32F746IGTx	Active	8.24			LQFP176	1024 kBytes	320 kBytes	140	216 MHz	0.0
STM32F746NG	STM32F746NGHx	Active	8.703	32F746GDISCOVERY	STM32746G-EVAL	TFBGA216	1024 kBytes	320 kBytes	168	216 MHz	0.0
STM32F746VG	STM32F746VGHx	Active	7.43			TFBGA100	1024 kBytes	320 kBytes	82	216 MHz	0.0
STM32F746VG	STM32F746VGTx	Active	7.43			LQFP100	1024 kBytes	320 kBytes	82	216 MHz	0.0
STM32F746ZG	STM32F746ZGTx	Active	6.62	NUCLEO-F746ZG		LQFP144	1024 kBytes	320 kBytes	114	216 MHz	0.0
STM32F746ZG	STM32F746ZGYx	Active	6.62			WLCSP143	1024 kBytes	320 kBytes	114	216 MHz	0.0
STM32F756BG	STM32F756BGTx	Active	9.05			LQFP208	1024 kBytes	320 kBytes	168	216 MHz	0.0
STM32F756IG	STM32F756IGKx	Active	8.587			UFBGA176	1024 kBytes	320 kBytes	140	216 MHz	0.0
STM32F756IG	STM32F756IGTx	Active	8.587			LQFP176	1024 kBytes	320 kBytes	140	216 MHz	0.0
STM32F756NG	STM32F756NGHx	Active	9.05	STM32756G-EVAL		TFBGA216	1024 kBytes	320 kBytes	168	216 MHz	0.0
STM32F756VG	STM32F756VGHx	Active	7.777			TFBGA100	1024 kBytes	320 kBytes	82	216 MHz	0.0
STM32F756VG	STM32F756VGTx	Active	7.777			LQFP100	1024 kBytes	320 kBytes	82	216 MHz	0.0
STM32F756ZG	STM32F756ZGTx	Active	6.967			LQFP144	1024 kBytes	320 kBytes	114	216 MHz	0.0
STM32F756ZG	STM32F756ZGYx	Active	6.967			WLCSP143	1024 kBytes	320 kBytes	114	216 MHz	0.0

Figure 4.10: List of MCU's with clock frequency of 216MHz sorted by Flash size in decending order

The major differences within these lines are the peripherals that are packed into the MCU. Another difference is the size in flash available for programming etc. Sort the MCUs in descending order of flash memory size and RAM size.

From these steps, the STM32F746ZG stood out. The reason being there is a development board available with this MCU on it, the Nucleo-F746ZG (See Figure 4.10). For fast deployment and prototyping purposes, it would be beneficial to use a development board. This eliminates the process of developing a printed circuit board (PCB) to interface external component with the MCU. The PCB will also need to perform functions like programming the MCU. These development boards provide a STLink. The STLink can program and debug the MCU.

The purpose of choosing the MCU with maximum frequency, maximum flash and RAM is for testing purposes only, as the next step we will extract the recommended frequency, flash and RAM from the coded algorithm.

4.3.6 Algorithm to code

The next step is to set up the MCU. Simulink® uses a project that has been set up in CubeMX in order to generate the code needed for the MCU to boot and be configured according to your specifications. For more information about CubeMX see Section 2.7.1. Simulink® or the vendor of the MCU will be able to provide the hardware support package for your MCU. This support package will enable Simulink® to convert the Simulation (discrete controller) into C/C++ code specifically for the targeted MCU. See Section 2.7.2 for more information on Matlab®/Simulink® coder. In case of a situation where there is no support package available, the controller can be compiled to C/C++ code which can be imported into the embedded C environment. It is then up to the programmer to call the appropriate functions at the appropriate times.

This is followed by setting up the software interfaces between the algorithm and the MCU. Once these interfaces are done the algorithm can be compiled to C-code using the appropriate settings in Simulink®(See Figure 4.11). The hardware support package used for this project will need the user to compile the code with an external IDE. This external IDE is specified in the CubeMX project file. The compiling process will generate all the files needed to the current folder(see Figure 4.12). During this compile process, the Simulink® coder calls the CubeMX interface to compile and generate the correct project files within the same directory.

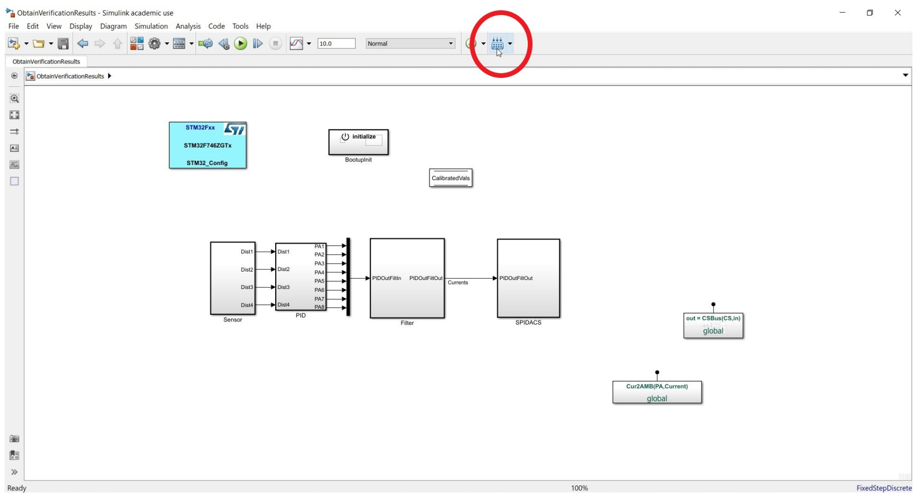


Figure 4.11: PID Control Algorithm to be converted to C-Code

CubeMX can create project files for the following IDEs:

- EWARM (30-trail)
- MDK-ARM (linker free for code up to 32k)
- TrueStudio (free)
- SW4STM32 (free)

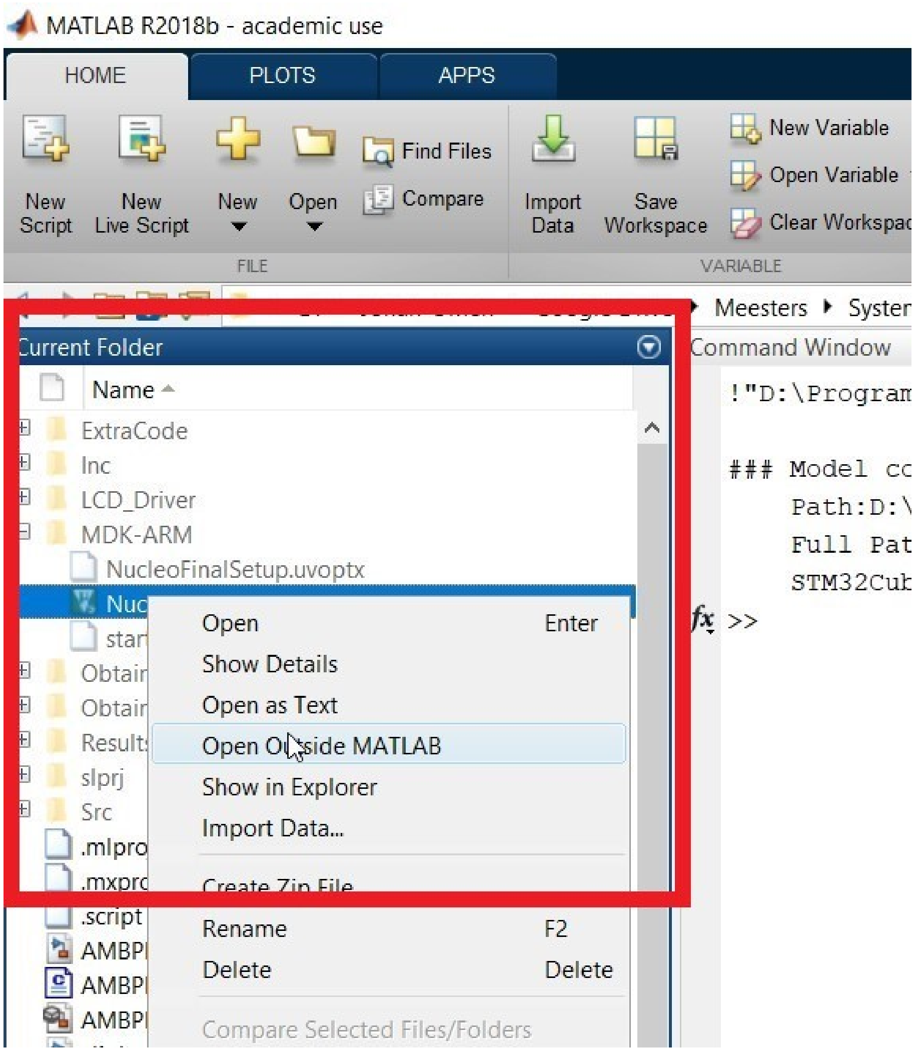
Of these IDEs, Keil's MDK-ARM is the only IDE which has a simulation and is free to use for projects up to 32KB in size. This is useful in our case as we want to simulate the control algorithm and calculate the time it took to be executed. This feature uses the principle that each assembler instructions takes a set time to be executed. The simulator analyses the assembler instructions and calculates the time sequentially. The settings of the CubeMX project can be seen in Appendix A.

4.3.7 Software Simulation

Software Setup

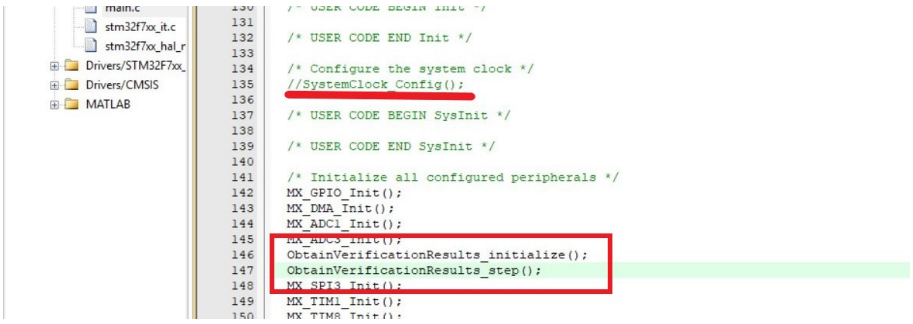
The Simulink® coder produces c/c++ code in a similar way to how Simulink® models are simulated. Simulink® initialises each block or model at the beginning of the simulation. This is followed by a step function of said block or model. The step function is then executed at the desired (solver) frequency. The embedded application uses the solver frequency as the implemented control frequency.

Before compiling the instructions in Keil, there are a few limitations to note. The simulator does not simulate the hardware. Simulation signals can be created in such a case. Some of the STM32 HAL software relies heavily on hardware responses. This includes the internal phase lock loop (PLL) of the reset and clock control (RCC) peripheral. Because the RCC is responsible for handling the system tick and thus the main control loop timing this poses a problem for the simulation. Preparing the program for the simulator can help us overcome some of these problems(Figure 4.13). By commenting out the system tick setup function, the HAL software that waits for the PLL response will not be executed. This will not affect the output timings of the simulation as the simulation timings rely on the frequency set in Keil's project settings. This setting was configured by the CubeMX software. The CubeMX software has set the frequency to 216 MHz. The step function in question is also added into the main function. This can be seen in Figure 4.13. The initialise function needs to be called before the step function in order for the step function to work. The reason for placing these functions at this specific point in the code is because the GPIO and ADC with the DMA configuration do not rely on hardware responses. This implies that they can be simulated by the simulator.



```
!"D:\Program

### Model cc
    Path:D:\
    Full Pat
    STM32Cub
fx >>
```

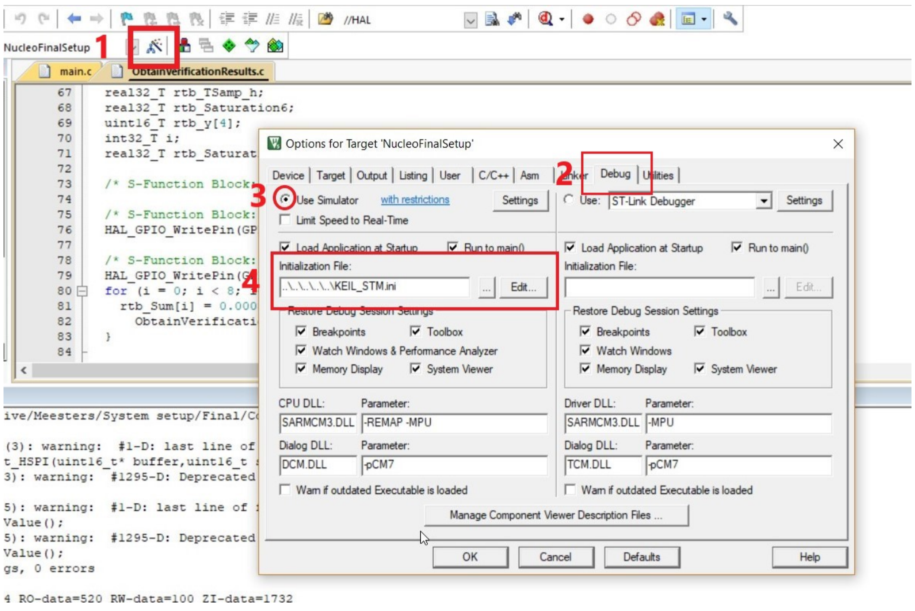
Figure 4.12: Open generated Keil Project



```
/* USER CODE END Init */

/* Configure the system clock */
//SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();
MX_ADC3_Init();
ObtainVerificationResults_initialize();
ObtainVerificationResults_step();
MX_SPI3_Init();
MX_TIM1_Init();
MX_TIM8_Init();
```

Figure 4.13: Code setup for Simulation

Figure 4.14 shows how to set up the Simulator. Step 4 in this figure can be seen in Figure 4.15. This step is necessary because another problem with the simulator is the read and write permissions of the flash memory that needs to be simulated. This can be done by writing a .ini file and including it in the simulation settings page(See Figure 4.15). The contents of this file can be seen in Figure 4.16.


1
2
3
4
NucleoFinalSetup
main.c
ObtainVerificationResults.c
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Settings
OK
Cancel
Defaults
Help
Manage Component Viewer Description Files ...

Options for Target 'NucleoFinalSetup'

[✓ Use Simulator] with restrictions
☐ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File: ..\..\..\..\..\KEIL_STM.ini
Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows & Performance Analyzer ☑ Memory Display ☑ System Viewer

CPU DLL:	Parameter:
SARMCM3.DLL	-REMAP -MPU
Dialog DLL:	Parameter:
DCM.DLL	-pCM7

☐ Warn if outdated Executable is loaded

Use: ST-Link Debugger
☑ Load Application at Startup ☑ Run to main()
Initialization File:
Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows ☑ Memory Display ☑ System Viewer

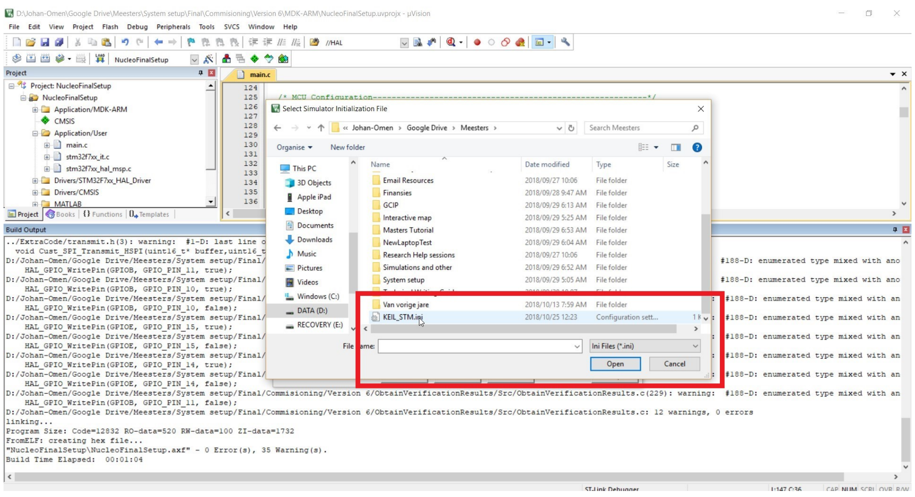
Driver DLL:	Parameter:
SARMCM3.DLL	-MPU
Dialog DLL:	Parameter:
TCM.DLL	-pCM7

☐ Warn if outdated Executable is loaded

```
real32_T rtb_TSamp_h;
real32_T rtb_Saturation6;
uint16_T rtb_y[4];
int32_T i;
real32_T rtb_Saturat

/* S-Function Block

/* S-Function Block:
HAL_GPIO_WritePin(GP

/* S-Function Block:
HAL_GPIO_WritePin(G
for (i = 0; i < 8;
  rtb_Sum[i] = 0.000
    ObtainVerificati
}
```

```
ive/Meesters/System setup/Final/C

(3): warning:  #1-D: last line of
t_HSPI(uint16_t* buffer,uint16_t
3): warning:  #1295-D: Deprecated

5): warning:  #1-D: last line of
Value();
5): warning:  #1295-D: Deprecated
Value();
gs, 0 errors

4 RO-data=520 RW-data=100 ZI-data=1732
```

Figure 4.14: Enable simulation mode

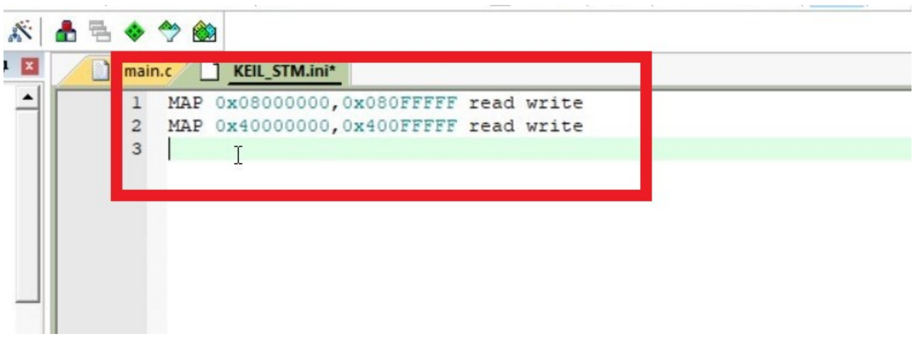

D:\Johan-Omen\Google Drive\Meesters\System setup\Final\Commisioning\Version 6\MDK-ARM\NucleoFinalSetup.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project: NucleoFinalSetup
NucleoFinalSetup
Application/MDK-ARM
CMSIS
Application/User
main.c
stm32f7xx_it.c
stm32f7xx_hal_msp.c
Drivers/STM32F7xx_HAL_Driver
Drivers/CMSIS
MATLAB
Select Simulator Initialization File
« Johan-Omen › Google Drive › Meesters ›
Search Meesters
Organise
New folder
This PC
3D Objects
Apple iPad
Desktop
Documents
Downloads
Music
Pictures
Videos
Windows (C:)
DATA (D:)
RECOVERY (E:)
Open
Cancel

Name	Date modified	Type	Size
Email Resources	2018/09/27 10:06	File folder	
Finansies	2018/09/28 9:47 AM	File folder	
GCIP	2018/09/29 6:13 AM	File folder	
Interactive map	2018/09/29 5:25 AM	File folder	
Masters Tutorial	2018/09/29 6:53 AM	File folder	
NewLaptopTest	2018/09/29 6:04 AM	File folder	
Research Help sessions	2018/09/27 10:06	File folder	
Simulations and other	2018/09/29 6:52 AM	File folder	
System setup	2018/09/29 5:05 AM	File folder	
Van vorige jare	2018/10/13 7:59 AM	File folder	
KEIL_STM.ini	2018/10/25 12:23	Configuration sett...	1 K

File name: Ini Files (*.ini)

```
../ExtraCode/transmit.h(3): warning:  #1-D: last line o
  void Cust_SPI_Transmit_HSPI(uint16_t* buffer,uint16_t
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with ano
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, true);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with ano
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, true);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, false);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, true);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, false);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_14, true);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/    #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_14, false);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/Commisioning/Version 6/ObtainVerificationResults/Src/ObtainVerificationResults.c(229): warning:  #188-D: enumerated type mixed with an
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, false);
D:/Johan-Omen/Google Drive/Meesters/System setup/Final/Commisioning/Version 6/ObtainVerificationResults/Src/ObtainVerificationResults.c: 12 warnings, 0 errors
linking...
Program Size: Code=12832 RO-data=520 RW-data=100 ZI-data=1732
FromELF: creating hex file...
"NucleoFinalSetup\NucleoFinalSetup.axf" - 0 Error(s), 35 Warning(s).
Build Time Elapsed:  00:01:04
```

Figure 4.15: Simulation initiation file settings



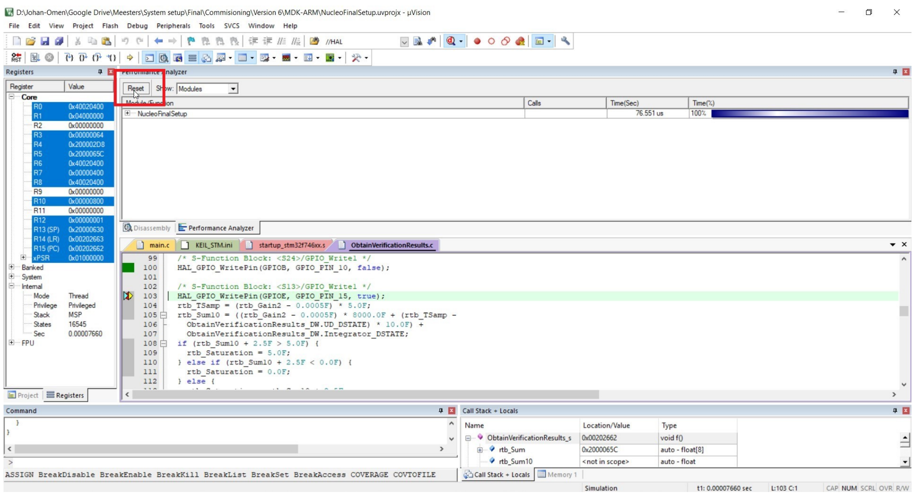
```
MAP 0x08000000,0x080FFFFF read write
MAP 0x40000000,0x400FFFFF read write
```

Figure 4.16: Contents of .ini file

Because the timings of the hardware can be calculated we only need to use the simulation software to calculate the processing part of the control algorithm. Thus the code is edited and compiled in such a manner that the processing timing can be obtained before the simulator "hangs" at the software that relies on hardware responses. The simulator is also configured in such a manner that timings are calculated with the MCU final operating frequency that is configured with the CubeMX interface. This setting in CubeMX defines the clock frequency in the Keil project. The Keil IDE uses this value for the simulator. The simulator does not take into account that the MCU starts up at a slower clock frequency.

Software simulation procedure

This information is repeated here to justify the two functions (initialise and step) that is being called at the start of the main loop for the simulation purposes. Within the step function, a GPIO pin is set before the processing stage. The same GPIO pin is reset after the processing stage. Debug breakpoints are set on these two lines of code. The simulation will start and the time between these two points are taken. Breakpoint 1 can be seen in Figure 4.17. The performance analyser's stopwatch needs to be reset(Figure 4.18) before continuing to breakpoint 2. The resulting time shows how fast the MCU will take for the PID algorithm to be executed once. The PID loop takes 1.875 us to execute. This result can be seen in Figure 4.19.



D:\Johan-Omen\Google Drive\Meesters\System setup\Final\Commisioning\Version 6\MDK-ARM\NucleoFinalSetup.uvprojx - µVision

Module/Function	Calls	Time(Sec)	Time(%)
NucleoFinalSetup		76.551 us	100%

Register	Value
R0	0x40020400
R1	0x04000000
R2	0x00000000
R3	0x00000064
R4	0x200002D8
R5	0x2000065C
R6	0x40020400
R7	0x00000400
R8	0x40020400
R9	0x00000000
R10	0x00000800
R11	0x00000000
R12	0x00000001
R13 (SP)	0x20000630
R14 (LR)	0x00202663
R15 (PC)	0x00202662
xPSR	0x01000000

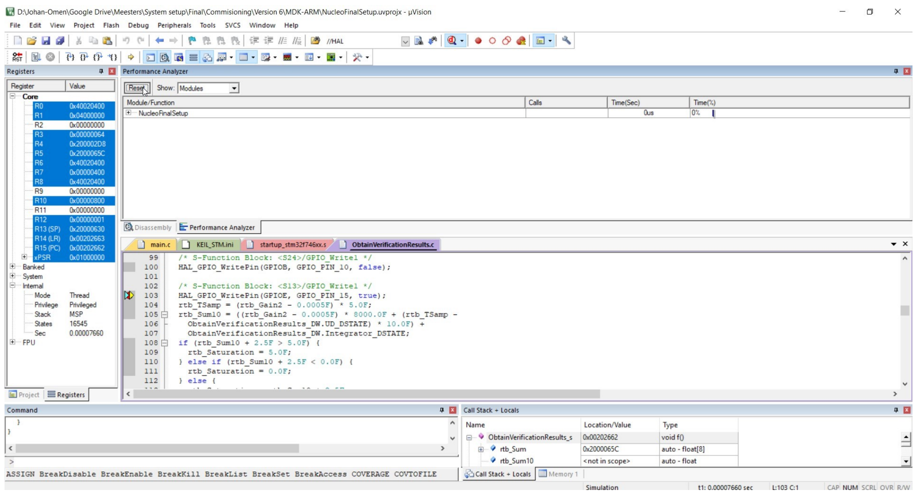
```
/* S-Function Block: <S24>/GPIO_Write1 */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, false);

/* S-Function Block: <S13>/GPIO_Write1 */
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, true);
rtb_TSamp = (rtb_Gain2 - 0.0005F) * 5.0F;
rtb_Sum10 = ((rtb_Gain2 - 0.0005F) * 8000.0F + (rtb_TSamp -
  ObtainVerificationResults_DW.UD_DSTATE) * 10.0F) +
  ObtainVerificationResults_DW.Integrator_DSTATE;
if (rtb_Sum10 + 2.5F > 5.0F) {
  rtb_Saturation = 5.0F;
} else if (rtb_Sum10 + 2.5F < 0.0F) {
  rtb_Saturation = 0.0F;
} else {
```

Name	Location/Value	Type
ObtainVerificationResults_s	0x00202662	void f()
rtb_Sum	0x2000065C	auto - float[8]
rtb_Sum10	<not in scope>	auto - float

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
```

Simulation t1: 0.00007660 sec L:103 C:1

Figure 4.17: Breakpoint at start of PID calculations



D:\Johan-Omen\Google Drive\Meesters\System setup\Final\Commisioning\Version 6\MDK-ARM\NucleoFinalSetup.uvprojx - µVision

Registers

Register	Value
Core	
R0	0x40020400
R1	0x04000000
R2	0x00000000
R3	0x00000064
R4	0x200002D8
R5	0x2000065C
R6	0x40020400
R7	0x00000400
R8	0x40020400
R9	0x00000000
R10	0x00000800
R11	0x00000000
R12	0x00000001
R13 (SP)	0x20000630
R14 (LR)	0x00202663
R15 (PC)	0x00202662
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	16545
Sec	0.00007660
FPU	

Performance Analyzer

Module/Function	Calls	Time(Sec)	Time(%)
NucleoFinalSetup		0us	0%

```
/* S-Function Block: <S24>/GPIO_Write1 */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, false);

/* S-Function Block: <S13>/GPIO_Write1 */
HAL_GPIO_WritePin(GPIOE, GPIO_PIN_15, true);
rtb_TSamp = (rtb_Gain2 - 0.0005F) * 5.0F;
rtb_Sum10 = ((rtb_Gain2 - 0.0005F) * 8000.0F + (rtb_TSamp -
  ObtainVerificationResults_DW.UD_DSTATE) * 10.0F) +
  ObtainVerificationResults_DW.Integrator_DSTATE;
if (rtb_Sum10 + 2.5F > 5.0F) {
  rtb_Saturation = 5.0F;
} else if (rtb_Sum10 + 2.5F < 0.0F) {
  rtb_Saturation = 0.0F;
} else {
```

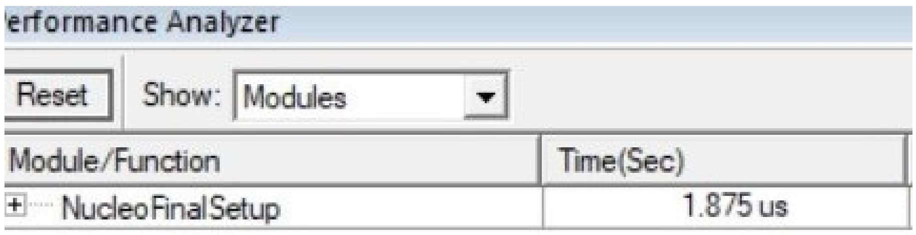
```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE
```

Call Stack + Locals

Name	Location/Value	Type
ObtainVerificationResults_s	0x00202662	void f()
rtb_Sum	0x2000065C	auto - float[8]
rtb_Sum10	<not in scope>	auto - float

Simulation t1: 0.00007660 sec L:103 C:1

Figure 4.18: Reset stopwatch at first break point



Performance Analyzer

Show: Modules

Module/Function	Time(Sec)
NucleoFinalSetup	1.875 us

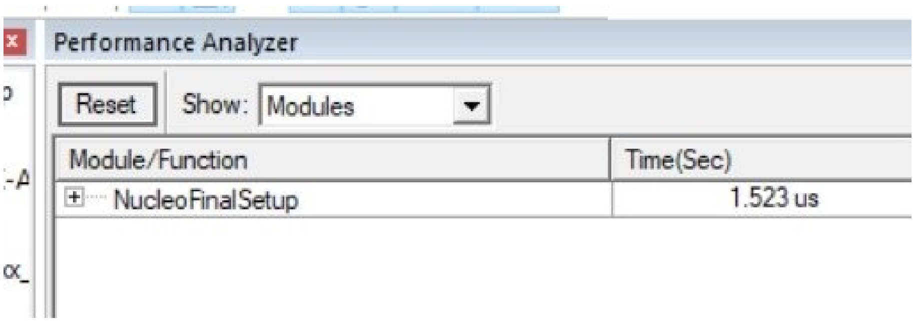
Figure 4.19: Break point at end of PID and LPF

4.3.8 Input and output peripherals delay

A closer look on how to calculate the delays the peripherals take see Section 3.4.8. The ADC in this example has been configured to work with the DMA controller. As mentioned in Section 3.4.8 the delay for this configuration only depends on the bus speed between the CPU and the RAM memory.

The bus between the RAM and CPU is clocked at 216 MHz. This equates to a single variable to arrive at the CPU in 4.63 ns. Multiplying this number by 4 for the 4 ADC channels equates to 18.52 ns. The interface created in Simulink® for this needs to convert these values to usable distance measures for the PID control. The time required to convert these values can also be simulated because the use of the DMA eliminates the use of the HAL's hardware response procedure. The software simulation takes into account the delay of the values being transferred from the memory to the CPU. The timing of this conversion equates to 1.53 μs (See Figure 4.20). The value seen in Figure 4.20 takes the 18.52 ns into account.

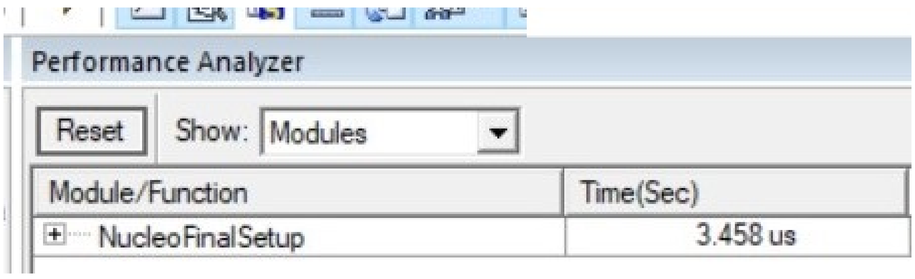
The DAC outputs will be realised by the dual channel SPI DACs. The timing to convert the PID control output into a 16-bit integer for the DAC can be simulated by the simulation software. The timing of the CS GPIO toggle (for both set and reset) and SPI data transfer needs to be calculated. The toggle of a GPIO pin can be calculated by the software simulation as the software only needs to set or reset a bit in a register. The SPI channels utilise its own PLL peripheral in order to obtain high transfer rates. Thus this cannot be simulated by the software simulator. The transfer rate of the SPI has to be configured as 6.75 MBits/s. Each transfer transfers 16 bits at a time. One transfer equates to 2.37 μs. The total transfer time for the SPI 8 channels is 18.96 μs. Before transmitting the data to the SPI DACs, the output of the PID loop needs to be converted to digital values that the SPI DAC can use. This can be simulated. This conversion was timed at 3.458 μs for one DAC and can be seen in Figure 4.21. Multiplying this value by 8 gives the total conversion for the values to be transmitted through the SPI channel. This equates to 27,872 μs. Adding the transfer time the time used by the MCU and SPI DACs to output to the PAs is around 46.835 μs.



Modules

Module/Function	Time(Sec)
NucleoFinalSetup	1.523 us

Figure 4.20: ADC conversion timing delay



Modules

Module/Function	Time(Sec)
NucleoFinalSetup	3.458 us

Figure 4.21: SPI DAC conversion timing delay

4.3.9 Results

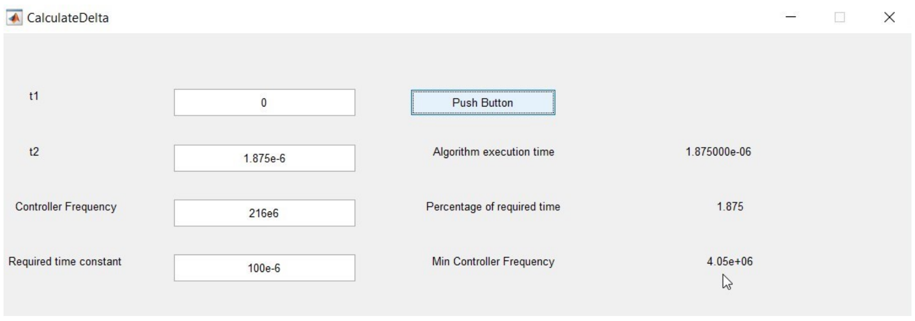
The table below provides the timings simulated and calculated for the STM32F746ZG MCU.

Table 4.1: Simulated timings of code and peripherals

Stage	**Timing**
ADC input and processing	1.523μs
PID and LPF execution	1.875μs
Output prepreation and SPI DACs timing	48.835μs
Total	52.233μs

$$Workload\% = \frac{1.874\mu s}{100\mu s} \times 100\% = 1.874\%$$

Figure 4.22 shows a program created in Matlab® to calculate the minimum required operating frequency needed of the MCU in order to use 100% of the control period. The calculation for this minimum frequency can be seen in Equation 3.12.



t1: 0

t2: 1.875e-6

Controller Frequency: 216e6

Required time constant: 100e-6

Algorithm execution time: 1.875000e-06

Percentage of required time: 1.875

Min Controller Frequency: 4.05e+06

Figure 4.22: Minimum frequency of MCU required to run PID control of AMB's at 10KHz

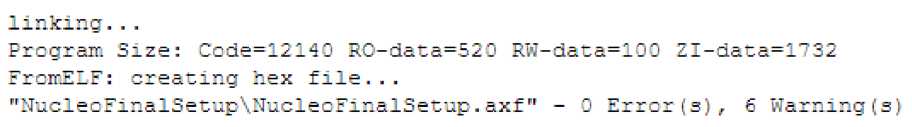
Using equation 3.11 and equation 3.12 the minimum operating frequency needed for this control can be calculated with Equation 4.2.

$$f_{minRequired} = \frac{T_{exetime}}{T_{control}} \times f_{operatingFrequency} \tag{4.2}$$

The use of an external peripheral needs to be taken into account when calculating the minimum required frequency for the MCU. The time the SPI peripheral uses to communicate the data to the external peripheral's was calculated as 18.96 μs. Subtracting this from the control period leaves the total processing time which was available for the MCU. This equates to 81.04 μs. The time the output stage used to process the data before transmitting it was 27.872 μs. Using this value for the output stage processing along with the 1.523 μs input time and 1.875μs PID time the total processing time is: 31.27 μs. External peripherals will not make use of the operating frequency of the MCU and thus the time used by these components should be subtracted from the control period as the MCU will only have that time available for processing. The communication to the external peripherals cannot be directly scaled as it has fixed discrete increments. The available processing time will be used for the control period in the equation. Using a 31.27 μs as the execution time and 81.04 μs as the control period, the minimum required operating frequency can be calculated as 83.346 MHz(See Equation 4.3).

$$f_{minRequired} = \frac{31.27\mu s}{81.04\mu s} \times 216MHz = 83.346MHz \tag{4.3}$$

This operating frequency needs to be met by the MCU as well as a SPI transfer rate of 6.75 MHz to realise this control procedure. During the compilation of the project the compiler output the size of the code. This can be seen in Figure 4.23. The flash memory required for this project will need to be at least larger than 12.72 KB (Code+RO-data+RW-data). The required RAM size can is 1.832 KB (RW-data+ZI-data). The operating frequency and SPI transfer rate also need to be equal to or higher than the values mentioned above.



```
linking...
Program Size: Code=12140 RO-data=520 RW-data=100 ZI-data=1732
FromELF: creating hex file...
"NucleoFinalSetup\NucleoFinalSetup.axf" - 0 Error(s), 6 Warning(s)
```

Figure 4.23: Compiler output of program size

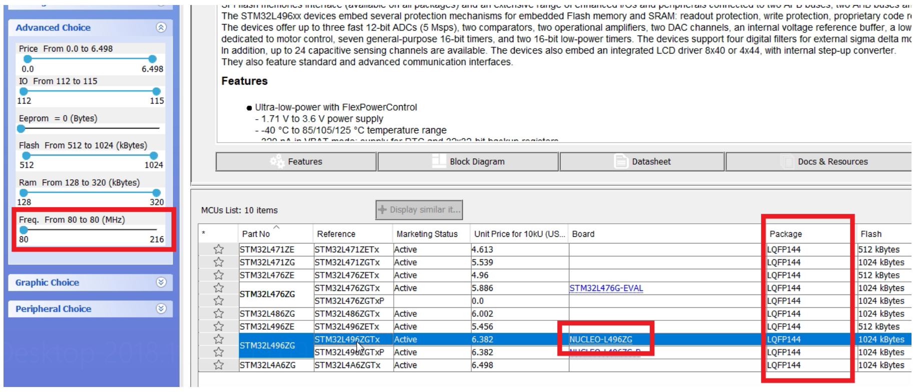
4.3.10 Final Results

The steps initial steps of Figure 3.37 was done in Section 4.3.1 to Section 4.3.9. The information obtained from these Sections can be used to redefine the MCU criteria. The new criteria can be seen in Table 4.2.

Table 4.2: Updated criteria

Description/Requirement	**Specification**
FPU	single precision
ADC	≥12 bit @ ≥0.01 MSPS
SPI	≥6.75 Mbps
Operating frequency	≥83.346 MHz
Form factor	Nucleo 144
Flash memory	≥12.72 KB
RAM memory	≥1.832 KB

The reason why the Nucleo 144-pin form factor is included in the criteria is to allow easy migration of the generated code to a new MCU. The same steps will be followed as done in Section 4.3.1 to 4.3.9 but with the new updated criteria (Table 4.2). Using this criterion but not adhering to the minimum operating frequency, the NUCLEO-L496ZG was chosen. This can be seen in Figure 4.24. The red blocks in this figure highlight the settings used for the filter and to highlight the chosen MCU. The operating frequency of this MCU is 80 MHz. The use of all the right peripherals and set-up of this MCU was done by using the Migration tool of STM32CubeMX. Figure 4.26 shows the migration process used to import the MCU settings from the previously used CubeMX project.



Advanced Choice

Price From 0.0 to 6.498 — 0.0 … 6.498
IO From 112 to 115 — 112 … 115
Eeprom = 0 (Bytes)
Flash From 512 to 1024 (kBytes) — 512 … 1024
Ram From 128 to 320 (kBytes) — 128 … 320
Freq. From 80 to 80 (MHz) — 80 … 216

Graphic Choice
Peripheral Choice

The STM32L496xx devices embed several protection mechanisms for embedded Flash memory and SRAM: readout protection, write protection, proprietary code r
The devices offer up to three fast 12-bit ADCs (5 Msps), two comparators, two operational amplifiers, two DAC channels, an internal voltage reference buffer, a low
dedicated to motor control, seven general-purpose 16-bit timers, and two 16-bit low-power timers. The devices support four digital filters for external sigma delta mo
In addition, up to 24 capacitive sensing channels are available. The devices also embed an integrated LCD driver 8x40 or 4x44, with internal step-up converter.
They also feature standard and advanced communication interfaces.

Features

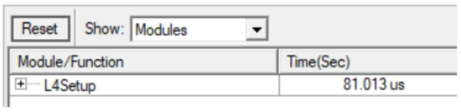
- Ultra-low-power with FlexPowerControl
  - 1.71 V to 3.6 V power supply
  - -40 °C to 85/105/125 °C temperature range

MCUs List: 10 items

*	Part No	Reference	Marketing Status	Unit Price for 10kU (US...	Board	Package	Flash
☆	STM32L471ZE	STM32L471ZETx	Active	4.613		LQFP144	512 kBytes
☆	STM32L471ZG	STM32L471ZGTx	Active	5.539		LQFP144	1024 kBytes
☆	STM32L476ZE	STM32L476ZETx	Active	4.96		LQFP144	512 kBytes
☆	STM32L476ZG	STM32L476ZGTx	Active	5.886	STM32L476G-EVAL	LQFP144	1024 kBytes
☆		STM32L476ZGTxP		0.0		LQFP144	1024 kBytes
☆	STM32L486ZG	STM32L486ZGTx	Active	6.002		LQFP144	1024 kBytes
☆	STM32L496ZE	STM32L496ZETx	Active	5.456		LQFP144	512 kBytes
☆	STM32L496ZG	STM32L496ZGTx	Active	6.382	NUCLEO-L496ZG	LQFP144	1024 kBytes
☆		STM32L496ZGTxP	Active	6.382	NUCLEO-L496ZG-P	LQFP144	1024 kBytes
☆	STM32L4A6ZG	STM32L4A6ZGTx	Active	6.498		LQFP144	1024 kBytes

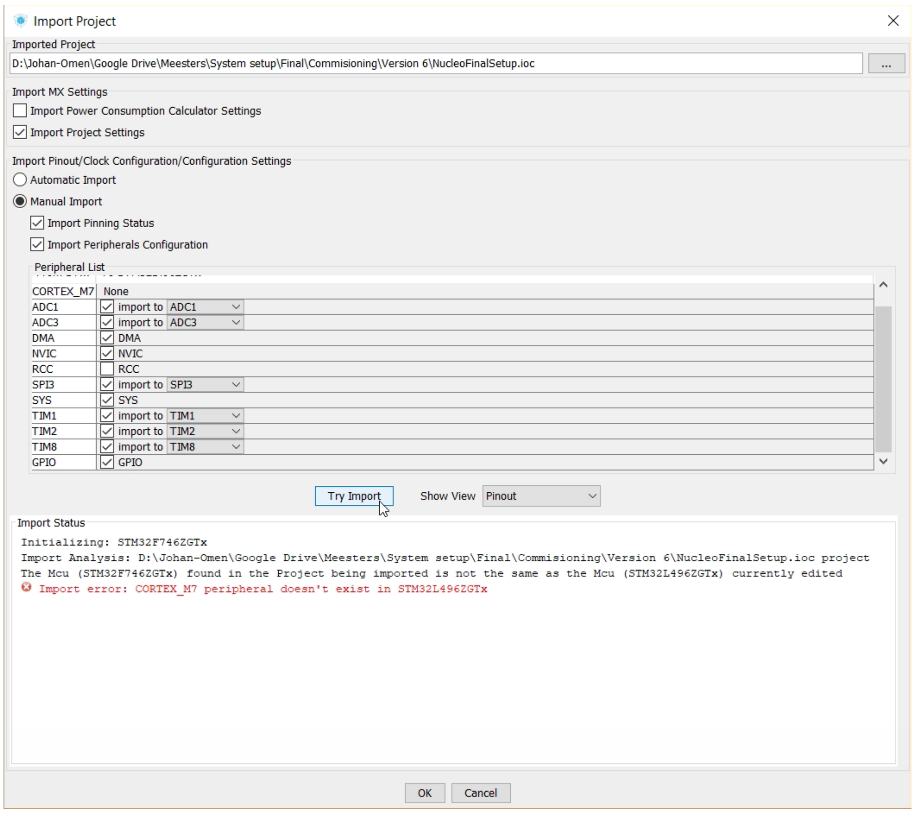
Figure 4.24: Filter settings that matches criteria exactly

The total processing time obtained by the code simulation is 81.113 μs. This can be seen in Figure 4.25. If the SPI peripheral is clocked at 6.75 MHz, the data transfer will take 18.963 μs. This will calculate to a total algorithm execution time of approximately 100.076 μs. One might expect a larger overrun, but there is a slight difference in architecture. This implies that the HAL drivers are slightly different. The difference between the F7 HAL library and the L4 HAL library is the reason why the overrun of this MCU is not as big as expected. This second iteration shows the importance of simulating the software of a MCU beforehand. The NUCLEO-L496ZG does, however, have a faster SPI peripheral. This would imply that the control algorithm will execute within the control period by increasing the SPI speed. This MCU has a SPI peripheral which is capable of speeds up to 50 Mbps.



Module/Function	Time(Sec)
L4Setup	81.013 us

Figure 4.25: STM32L496ZG processing duration



Import Project

Imported Project
D:\Johan-Omen\Google Drive\Meesters\System setup\Final\Commisioning\Version 6\NucleoFinalSetup.ioc

Import MX Settings
☐ Import Power Consumption Calculator Settings
☑ Import Project Settings

Import Pinout/Clock Configuration/Configuration Settings
○ Automatic Import
● Manual Import
☑ Import Pinning Status
☑ Import Peripherals Configuration

Peripheral List

CORTEX_M7	None
ADC1	☑ import to ADC1
ADC3	☑ import to ADC3
DMA	☑ DMA
NVIC	☑ NVIC
RCC	☐ RCC
SPI3	☑ import to SPI3
SYS	☑ SYS
TIM1	☑ import to TIM1
TIM2	☑ import to TIM2
TIM8	☑ import to TIM8
GPIO	☑ GPIO

Import Status

```
Initializing: STM32F746ZGTx
Import Analysis: D:\Johan-Omen\Google Drive\Meesters\System setup\Final\Commisioning\Version 6\NucleoFinalSetup.ioc project
The Mcu (STM32F746ZGTx) found in the Project being imported is not the same as the Mcu (STM32L496ZGTx) currently edited
Import error: CORTEX_M7 peripheral doesn't exist in STM32L496ZGTx
```

Figure 4.26: STM32CubeMX project migration tool

4.4 Conclusion

The NUCLEO-F746ZG would be the better choice for the implementation of the control algorithm. Additional software might be added later for better user experience. Given that the MCU has 45% of the control period available the additional software can use this available time. The NUCLEO-L496ZG can be used to control the plant, but no additional software can be added, only the core control algorithm will be able to run on this MCU. The steps followed in Section 4.3.9 and 4.3.10 can be iterated until the MCU under investigation satisfies the all the requirements and needs of the engineer. In this case the NUCLEO-F746ZG left with additional computation time is chosen for UI and future development purposes. This Chapter serves as the verification to this dissertation.

Chapter 5

MCU Implementation and results

In this Chapter the MCU chosen in Chapter 4 will be implemented with the same controller and plant described in Section 4.2 and 4.1 respectively. More practical details of the plant, PAs, sensors and the MCU are given in this Chapter. The time that the MCU took to the control algorithm will be measured along with the step response of the controller. The results of this Chapter will assist in the validation of the selection method.

5.1 Plant

The plant were constructed [7] as mentioned in Section 4.1 for the evaluation of a AMB control algorithm. The selection method was implemented in Section 4.3 and the code generated during the course of the selection method will now be implemented on the Nucleo-F746ZG. The selection method produced estimated durations of each stage of the control algorithm. These estimations will be validated by the use of the validation method mentioned in Section 3.2.2. The plant can be seen in Figure 5.1 and 5.2.

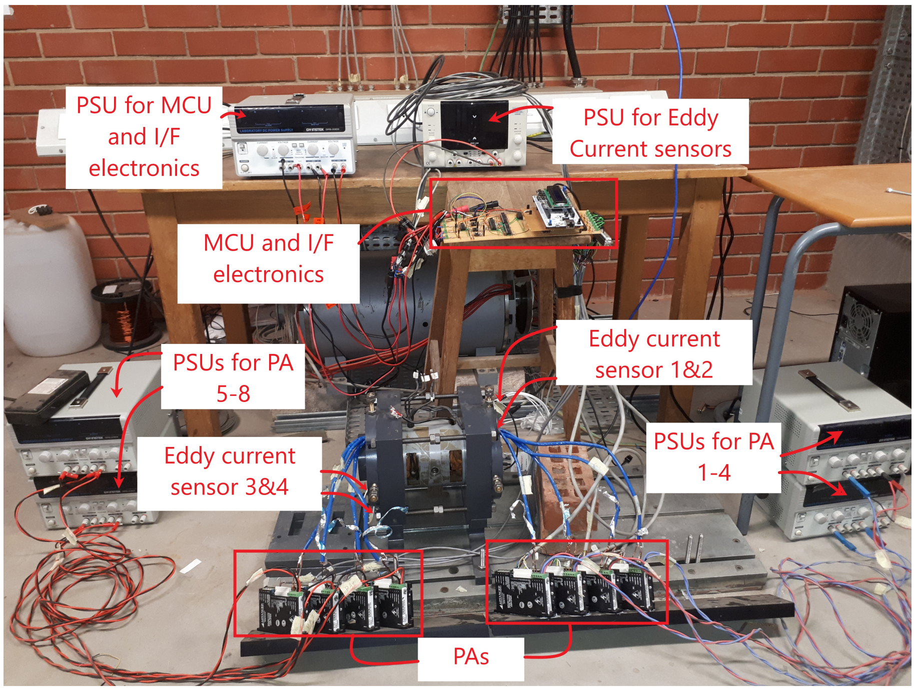


Figure 5.1: The implemented AMB system on PMSM

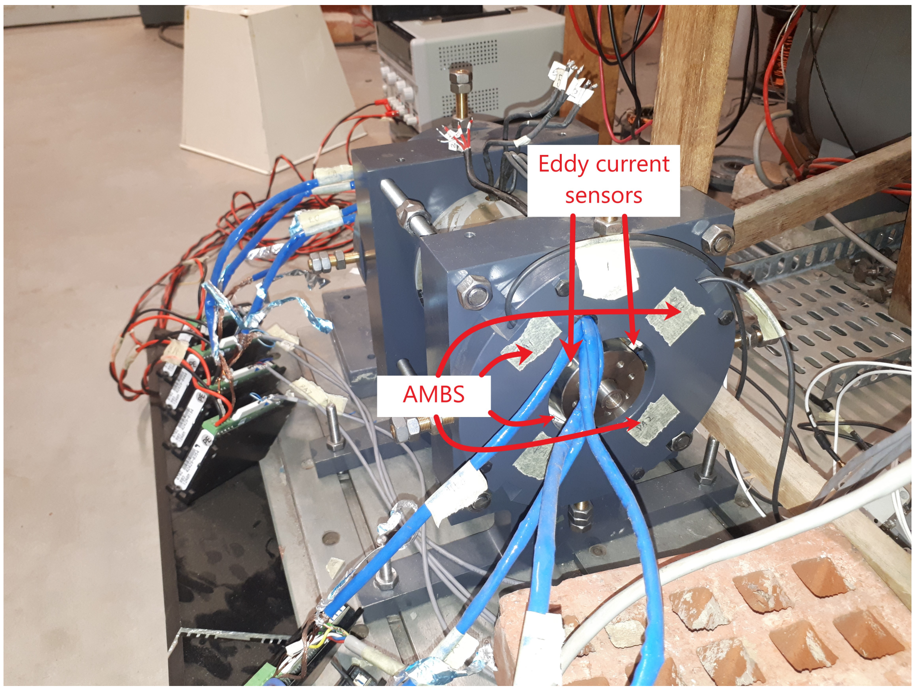


Figure 5.2: Close-up view of the sensors and PAs

The choice of the MCU and the development of the I/F electronics has been done during the process of this masters project. The implemented MCU and I/F electronics can be seen in Figure 5.3.

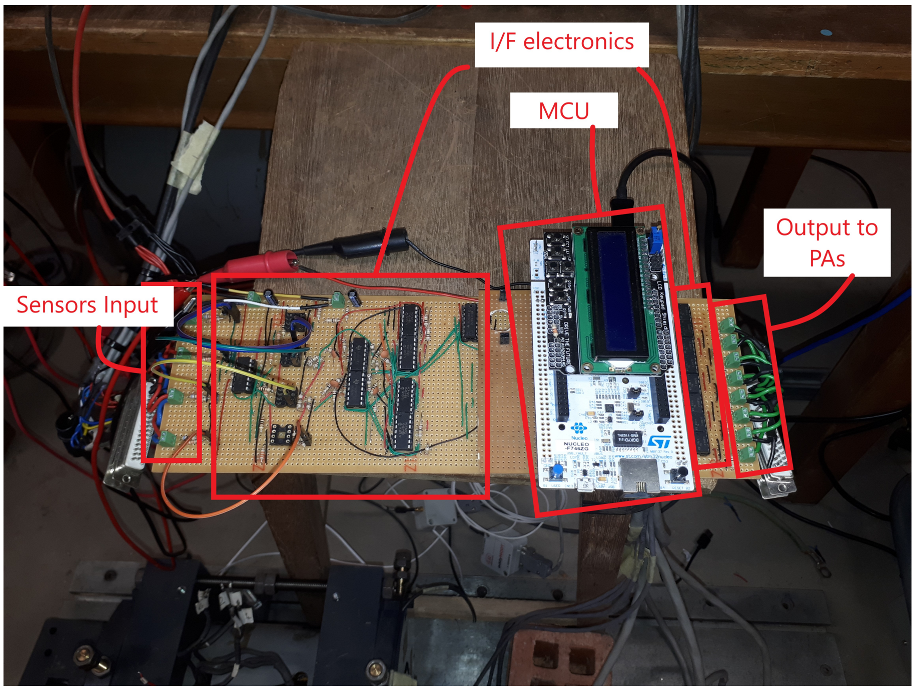


Figure 5.3: Close-up view of the MCU and the I/F electronics

Details of the AMB plant, PAs, eddy current sensors, the I/F electronics and the MCU will be discussed in the sections to follow.

5.1.1 The AMBs for the PMSM

The specifications of the AMBs can be seen in Table 5.1[7]. The structure of the AMB can be seen in Figure 5.4. The symbols shown in round brackets in Table 5.1 correspond with the symbols seen in Figure 5.4.

Table 5.1: AMB specifications [7]

Description	Value
Coil Inductance	6 mH
Coil Resistance	150 mΩ
Designed force (F_{max})	150 N
Coil length(l_c)	26 mm
Coil thickness(t_c)	7 mm
Coil radius(r_c)	65 mm
Number of turns(N)	58
Peak flux density(B)	0.7236 T
Pole width(w)	36 mm
Axial length(l)	20 mm
Stator outer radius(r_s)	85 mm
Pole radius(r_p)	32 mm
Journal radius(r_j)	32.5 mm
Shaft radius(r_r)	9 mm
Rotor weight	4.44kg
Bias current	2.5 A
Calculated current gain	30.4 N/A
Calculated open loop stiffness	152.2 N/m

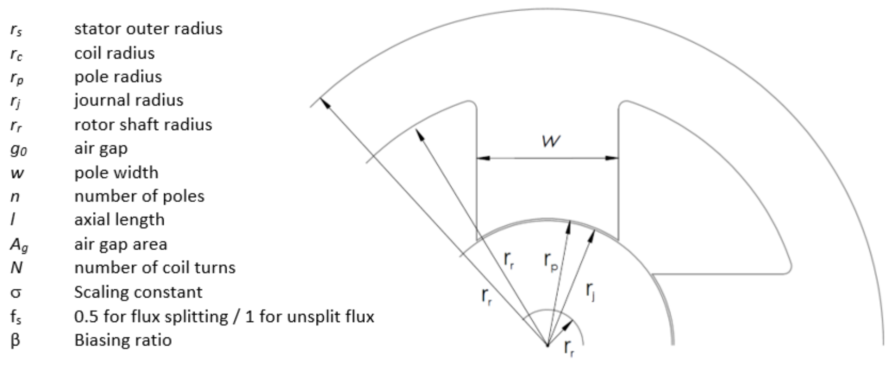


Figure 5.4: Structure of the AMB

A detailed discussion of the plant can be seen in Section 4.1.

5.1.2 PA and Eddy current sensors

The PA that was used is the Analog Servo Drive from Advanced Motion Controls (12A8). It has a peak current rating of 12 A, a continuous current rating of 6 A and an input supply of 20-80 VDC. It was used in its current control configuration. A differential voltage input will be used by the PA's internal control loop to convert the voltage value to a current. The PA has a voltage to current ratio of 1. The PA can be seen in Figure 5.5

Figure 5.5: Power Amplifier used for this project

The Eddy current sensor that was used for this project can be seen in Figure 5.6 (DT3005-U1-A-C1). The sensor has a measuring range of 1 mm on aluminium objects. It has an offset distance of 0.1 mm, a linearity of 2.5 μm ($\leq$ 0.25% full scale output (FSO)) and a resolution of 0.5 μm($\leq$ 0.05% FSO).

Figure 5.6: Eddy current sensor used for this project

5.1.3 MCU

The MCU that will be used is the STM32F746ZG. This MCU is available on a development board. The Nucleo-F746ZG will be used for the implementation of the controller algorithm on the plant. This MCU has a ARM 32-bit Cortex-M7 CPU with a single precision FPU, adaptive real-time (ART) Accelerator, L1-cache and DSP instructions. It has a 1 MB flash memory, 6 SPIs, 3x12-bit 2.4 MSPS ADC with up to 24 channels and a general purpose DMA. The Nucleo-F746ZG can be seen in Figure 5.7.

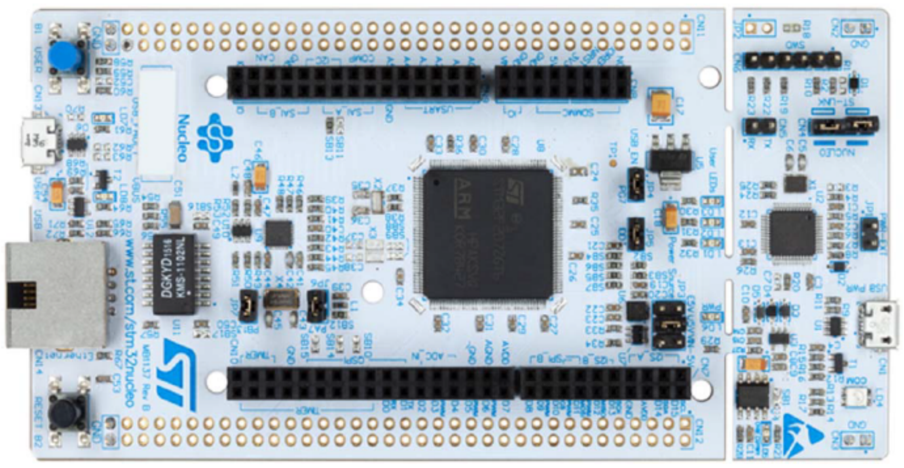


Figure 5.7: Eddy current sensor used for this project

5.1.4 I/F electronics

The functionality of the I/F circuit has been explained and can be seen in Section 4.3.2. The schematic of the I/F electronics can be seen in Appendix B. The list of components that were used can be seen in Table 5.2.

Table 5.2: Components used for the I/F electronics

Component	**Total number used**
2 port screw terminals	15
LM2902N 14-pin integrated circuit (IC) with quad op-amps	5
IL300 linear opto-coupler	12
MCP4251-104E/P dual digital SPI pot	4
SN74HCT138 8 channel decoder	1
MCP4922 dual channel SPI DAC	4

5.2 Results

5.2.1 Method validation

The code that was used in Section 4.3.7 is implemented on the Nucleo-F746ZG development board. The two functions called at the beginning of the main loop are removed, and the clock initialisation function is uncommented. The GPIO pin which is set at the beginning of a stage and reset after the stage can be used to obtain the execution duration of each stage. An oscilloscope was then used to capture the waveform generated by the GPIO pin of each stage. The resulting waveforms can be seen in Figure 5.8.

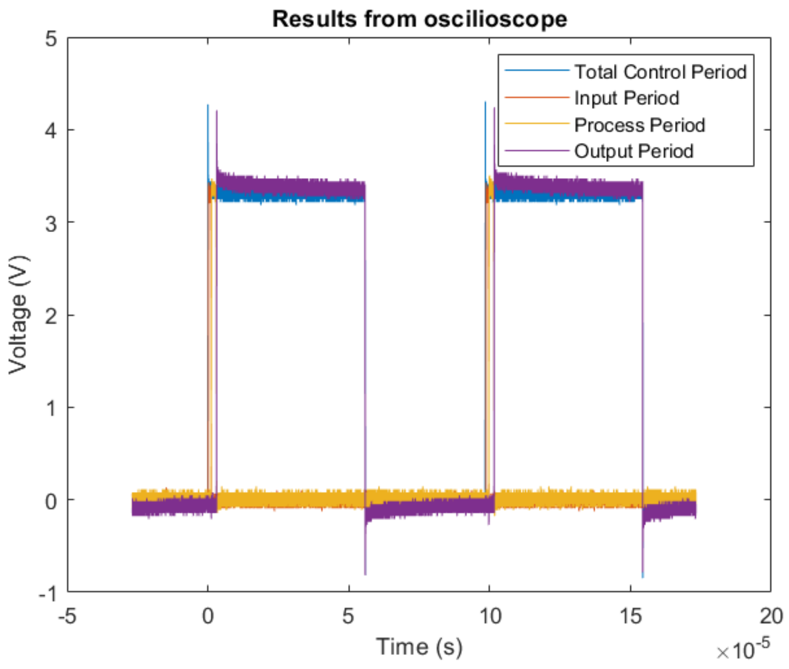


Figure 5.8: Scope waveforms

The pulse width timings of these waveforms can be seen in Table 5.3.

Table 5.3: Stage execution duration

Stage	**Duration**
Input duration	1.008 μs
Process duration	1.7668 μs
Output duration	52.715 μs
Total duration	**55.871 μs**

The percentage that each stage uses of the control period can be seen in Figure 5.9.

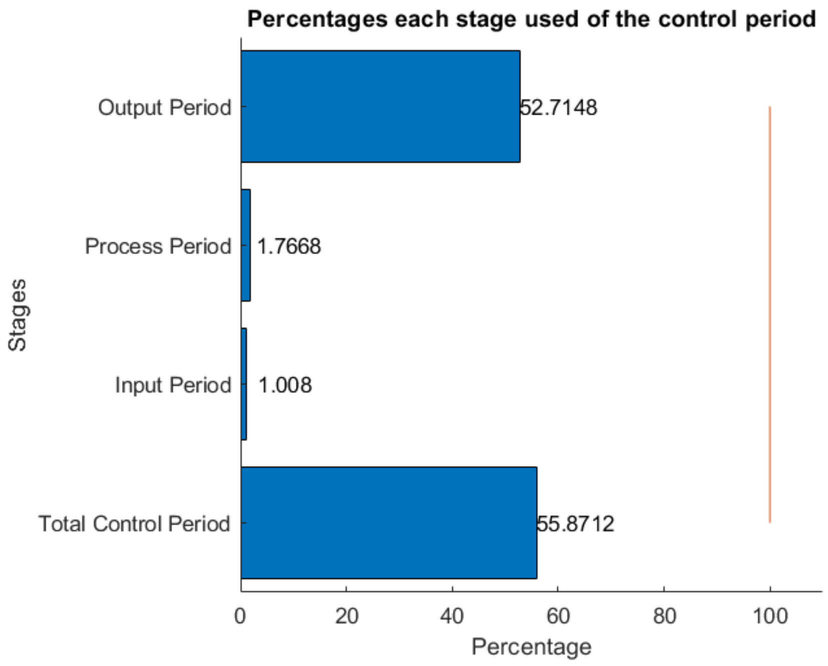


Figure 5.9: Percentage of control period used

5.2.2 Controller validation

Figure 5.10 shows the ADC values during the operation of the AMB system. The ADC values represent the position of the rotor. An ADC value of approximately 2048 is the centre point between the two opposing AMBs. The 0.5 mm reference equates to an equivalent ADC value of 2048. Figure 5.11 is an image overlay of 2 still images taken of the system at 1000 ms and 3000 ms. The blue and red circles seen in this figure are the positions of the rotor in each still image. The blue circle is the resting position(taken at 1000 ms) whereas the red circle is the levitating position(taken at 3000 ms).

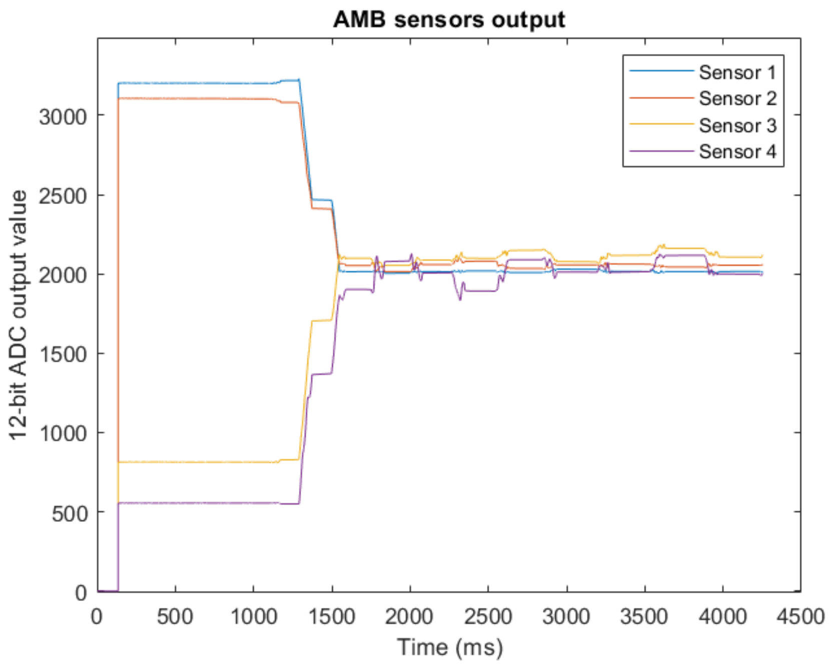


Figure 5.10: Sensor feedback during control execution

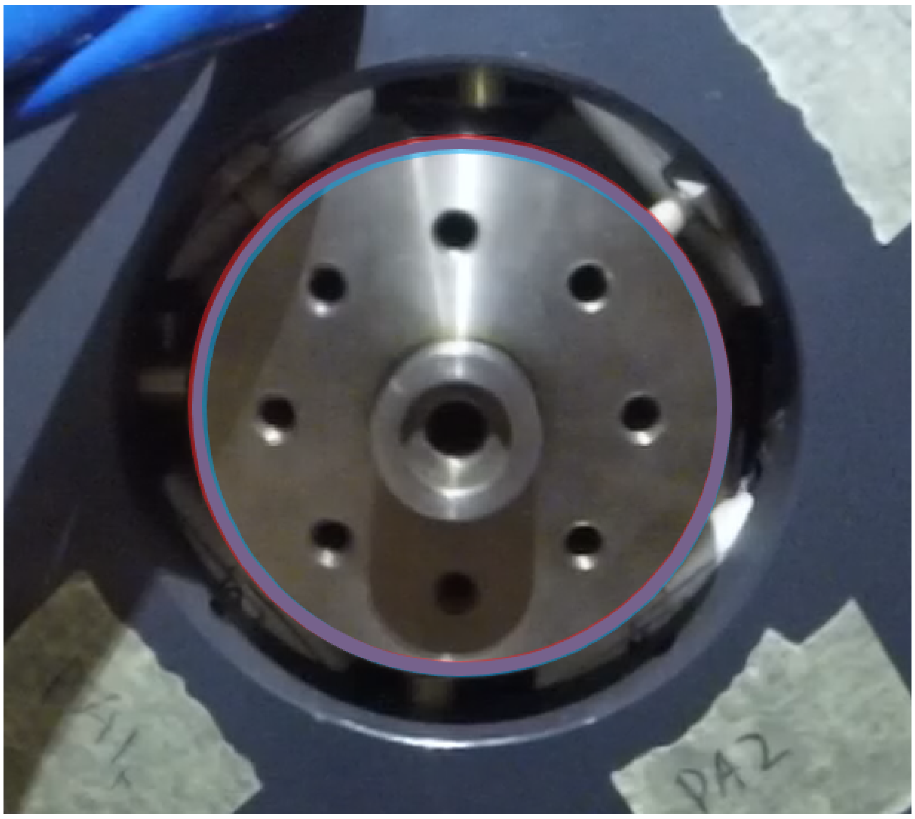


Figure 5.11: Image overlay of the system's resting(blue) and levatation(red) states

5.3 Result discussion

The estimations timings match the implemented timings within an acceptable margin. The estimated and practical durations can be seen in Table 5.4. The negative values in the deviation column is an indication that practical durations were faster than the estimated durations. Large negative deviation values are good news, but it is an indication that the method of estimating the value can be improved or investigated. The deviation of the input stage had a positive 0.52% impact on the time available in the control period. The process stage had a positive 0.11% impact, whereas the output stage was underestimated and had a negative 3.88% impact on the available time of the control period.

Table 5.4: Comparison between estimated and practical timings

Stage	**Estimation**	**Practical**	**Deviation**	**Deviation's impact on control period**
Input duration	1.523 μs	1.008 μs	-33.81%	-0.52%
Process duration	1.875 μs	1.7668 μs	-5.77%	-0.11%
Output duration	48.835 μs	52.715 μs	7.95%	3.88%
Total duration	52.233 μs	55.871 μs	6.96%	3.64%

5.3.1 Output stage deviations

Upon examining the SPI HAL library there was code not taken into account because of reasons mentioned in Section 4.3.8. Every line of code uses the time available of the control period. This code(which is not simulated) is the reason why the implemented output stage duration is longer than the estimated duration.

5.3.2 Processing and input stage deviations

The Keil simulation software simulates the architecture of a Cortex M4 processor even when the target uses a Cortex M7 as specified in the Keil project file. The Cortex M4 processor and the Cortex M7 processor has the same architectures, but the M7 has some additional features. The reason the input and processing stage executed faster than estimated is because the simulator did not simulate the additional features found in the cortex M7. The effects the additional features has at this small computational window is noticeably small. This is clearly shown by the last column seen in Table 5.4. The M7 would perform much better than the M4 when large amounts of data need to be processed. The paid version of the Keil MDK-ARM IDE has more advanced models which can be used for simulation purposes. These models will include models like the Cortex M7. The additional features of the M7 include TCM interface and instruction and data caches which improves the execution duration of instructions.

5.3.3 Future development

The AMBs levitates the rotor of a PMSM. Vector control can be used for controlling the speed of a PMSM. Vector control is a closed loop control for PMSM machines. Vector control uses the machine's current, voltage and rotor position as input. A vector control algorithm is given in Figure 5.12 [6].

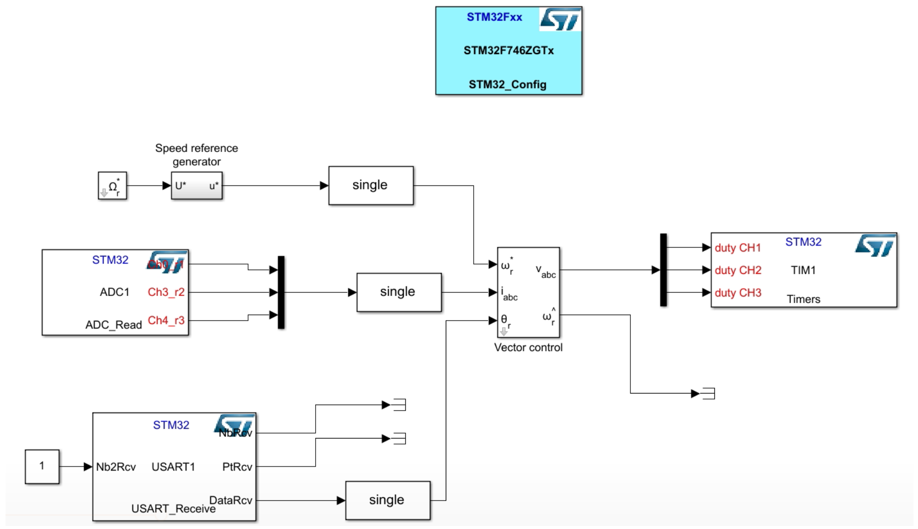
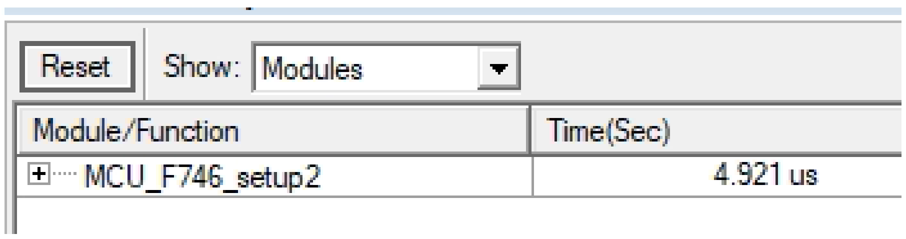


Figure 5.12: Vector control of a PMSM[6]

The selection method was used to evaluate the time needed for this control algorithm to execute. The evaluation of the timing was done on the NUCLEO-F746ZG. The time the NUCLEO-F746ZG will take to execute this control algorithm is 4.921 μs. The results of this can be seen in Figure 5.13.



Module/Function	Time(Sec)
MCU_F746_setup2	4.921 us

Figure 5.13: Vector control of a PMSM timing on NUCLEO-F746ZG[6]

Chapter 6

Conclusion and recommendations

6.1 Method selection

6.1.1 Method selection implementation

This dissertation presents a MCU selection method for complex control algorithms. The selection method is presented, and a flowchart of this can be seen in Figure 3.37. This method was used to select a MCU for the control of an AMB which levitates the rotor of a PMSM. The method requires a complete Simulink® simulation of the controller. The goal of the simulation is to test and design the controller, but it can also be used to program and estimate the requirements of the MCU needed to implement this controller. Initially, a MCU with the fastest operating speed and largest flash and RAM size was selected to determine the minimum required operating speed, flash and RAM. The simulation of the control algorithm was evaluated for code build and compiled for the NUCLEO-F746ZG. The simulation of the code along with additional peripheral delay calculations concluded that the control algorithm would take 52.233 μs to execute on the NUCLEO-F746ZG. This calculates to a workload of 53%. The method concluded that this MCU would be able to execute the control algorithm and ultimately control the plant. The evaluation information of this controller was then used to calculate that the minimum operating frequency required for a similar MCU is 83.346 MHz. The criteria for compiling a list of MCU candidates was updated as seen in Table 4.2. The method has provided a MCU which will have enough processing power to execute the control algorithm, however, selecting a MCU with lower specifications will reduce the cost to the system. Using the criteria, a new MCU was chosen, adhering to all the parameters of the criteria except the operating frequency. This was intentionally done to show that the selection method concludes that this MCU will not be able to execute the control algorithm within the control period. The fact that the selection method provided a discrete answer whether or not the MCUs evaluated can be used serves as verification for the selection method.

6.1.2 MCU implementation

The MCU which was chosen at the end of Chapter 4 was implemented on the AMB system. This implementation was documented in Chapter 4.2. The resulting execution time of the control algorithm on the MCU was 55.871 μs. This is only a 3.64% deviation from the estimated timing. The reason behind the deviations was explained in Section 5.3. The fact that the deviations is less than 5% and the fact that the rotor can be levitated (see Figure 5.10 and 5.11) validates the MCU selection method presented in this dissertation.

The reason why STMicroelectronics was considered in this dissertation is because they were the only vendor that enables the use of a third party application with all the tool-chain parameters already compiled. In other words, STMicroelectronics® offered the best support for the method created and presented during this dissertation.

6.2 Recommendations

6.2.1 Selection method

Selection method improvements

The selection method is implemented in such a way that the use of data type is already known. This implies that before using the selection method, the engineer already knows if the MCU needs a FPU. The selection method can be improved to include trade-off studies between data types and whether or not the MCU needs a DSP, single point or double FPU.

For this method to be supported by vendors other than STMicroelectronics® further knowledge is needed for setting up and importing the Matlab conversion code into other vendor's IDEs.

Other IDEs

The use of other IDEs like MPLab and Atmel Studio could also be used for the method presented in this dissertation. The results obtained from these IDEs should, however, be verified before it can be recommended to use them for this method.

6.2.2 Kernel

Any additional actions that the MCU must be able to do during a control step needs to be included in the software simulation. It is important to note that every single line of code uses the available time of the control period. An example of additional program code will be if feedback during operation needs to be given to the operator. The method of feedback should not cause an over-run. This could be accomplished by only using a fraction of the available time in the control period or using a peripheral like a DMA controller which will not disturb the operation of the CPU.

6.2.3 Hardware improvements

Output stage improvements

In this project, only one SPI peripheral was used in combination with a CS decoder. The use of more than one SPI in combination with the DMA controller will have a faster output stage duration. The use of the DMA controller, in this case, enables the possibility that the SPI peripherals can be used simultaneously. This can be implemented on this MCU as it has 6 SPI peripherals. One SPI peripheral can be used to set up the 4 SPI digital pots for the interfacing circuit. 4 SPI channels will then be used in combination with a DMA to achieve parallel communication to the SPI DACs. The duration of the pre-processing of the data will still take 27.872 μs. The total data transfer time will then be reduced from 18.96 μs to 4.74 μs. This will reduce the estimated output stage duration from 48.835 μs to 23.70 μs.

Further investigations

Figure 5.10 shows the output of the four sensors during the controller execution. From this graph, it can be seen that Sensor 4 deviates a lot more from the reference than the other sensors. Sensor 3 and four has been reused from a previous project whereas sensor 1 and 2 is brand new. There is a loud audible hum which is produced on the side of the PMSM which has the AMB that utilises this sensor. Further investigation into these problems is required to draw a reasonable conclusion to both the deviation and the loud hum.

Power supplies

The bench power supplies used for this project can be replaced. A power supply and power distribution network can be designed to replace the bench power supplies that are currently being used. Replacing these power supplies will take the system one step closer to being used in a real-world application.

PMSM control

The AMBs is used to levitate the rotor of a PMSM. The controller has the timer peripherals to provide an inverter with the needed signals to produce 3 phase power for the PMSM. The Nucleo-F746ZG has two timer peripherals which can be used for 3 phase applications. These timer peripherals can also be used in combination with the DMA. The MCU has enough time of the control period available to implement a closed-loop vector control. This is proven by the results shown in Figure 5.13 in Section 5.3.3.

Bibliography

[1] M. P. Bates, *Interfacing PIC microcontrollers: Embedded design by interactive simulation.* Newnes, 2013.

[2] R. C. Dorf and R. H. Bishop, *Modern control systems.* Pearson, 2011.

[3] C. L. Phillips and H. T. Nagle, *Digital control system analysis and design.* Prentice Hall Press, 2007.

[4] Linear electric actuator with control. [Online]. Available: https://www.mathworks.com/help/releases/R2017b/physmod/elec/examples/linear-electric-actuator-with-control.html

[5] *Simulink Coder User's Guide*, MathWorks Inc., 3 Apple Hill Drive Natick MA, September 2018.

[6] G. L. Kruger, "Implemention and evaluation of v/f and vector control in high-speed pmsm drives," 2011.

[7] S. Myburgh, "The development of a fully suspended amb system for a high-speed flywheel application," Master's thesis, North-West University, 2007.

[8] J. Beningo, "10 steps to selecting a microcontroller."," *Community. Arm (blog). January*, vol. 13, 2014.

[9] D. Herbst, "Single board computer based control of an active magnetic bearing," Master's thesis, North-West University, 2008.

[10] S. J. M. Steyn, P. van Vuuren, and G. van Schoor, "Multivariable h∞ control for an active magnetic bearing flywheel system," Master's thesis, 2010.

[11] M. Kiani, H. Salarieh, A. Alasty, and S. M. Darbandi, "Hybrid control of a three-pole active magnetic bearing," *Mechatronics*, vol. 39, pp. 28–41, 2016.

[12] C. D. Aucamp, "Model predictive control of a magnetically suspended flywheel energy storage system," Master's thesis, North-West University, 2012.

[13] R. R. Le Roux, "An embedded controller for an active magnetic bearing and drive electronic system," Master's thesis, North-West University, 2009.

[14] M. Zolfaghari, S. A. Taher, and D. V. Munuz, "Neural network-based sensorless direct power control of permanent magnet synchronous motor," *Ain Shams Engineering Journal*, vol. 7, no. 2, pp. 729–740, 2016.

[15] A. De Klerk, "Drive implementation of a permanent magnet synchronous motor," Ph.D. dissertation, Citeseer, 2007.

[16] M. Qutubuddin and N. Yadaiah, "Modeling and implementation of brain emotional controller for permanent magnet synchronous motor drive," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 193–203, 2017.

[17] Q. Guo, C. Zhang, L. Li, D. Gerada, J. Zhang, and M. Wang, "Design and implementation of a loss optimization control for electric vehicle in-wheel permanent-magnet synchronous motor direct drive system," *Applied Energy*, 2017.

[18] G. L. Kruger, G. Van Schoor, and P. A. Van Vuuren, "Control of magnetically suspended rotor combined with motor drive system," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 7252–7257, 2014.

[19] T. Baumgartner, R. M. Burkart, and J. W. Kolar, "Analysis and design of a 300-w 500 000-r/min slotless self-bearing permanent-magnet motor," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4326–4336, 2014.

[20] M. A. Mazidi, J. G. Mazidi, and R. D. McKinlay, *The 8051 Microcontroller: A Systems Approach.* Pearson, 2013.

[21] K.-D. Kramer, T. Stolze, and T. Banse, "Benchmarks to find the optimal microcontroller-architecture," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 2. IEEE, 2009, pp. 102–105.

[22] F. Gaillard, “Microprocessor (mpu) or microcontroller (mcu)? what factors should you consider when selecting the right processing device for your next design,” *URL http://ww1. microchip. com/downloads/en/DeviceDoc/MCU_vs_ MPU_Article. pdf*, 2013.

[23] I. Chivers and J. Sleightholme, *Introduction to programming with Fortran.* Springer, 2011.

[24] *STM32CubeMX for STM32 configuration and initialization C code generation*, 27th ed., STMicroelectronics, November 2018.

[25] R. Toulson and T. Wilmshurst, *Fast and effective embedded systems design: applying the ARM mbed.* Newnes, 2016.

[26] K. Morishima and S. Oho, “Implementation analysis of control schemes for choosing a microcontroller,” in *SICE Annual Conference (SICE), 2011 Proceedings of.* IEEE, 2011, pp. 2438–2441.

[27] J. A. Berlier and J. M. McCollum, “A constraint satisfaction algorithm for microcontroller selection and pin assignment,” in *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the.* IEEE, 2010, pp. 348–351.

[28] B. Martín-del Brío, A. Bono-Nuez, and N. Medrano-Marqués, “Self-organizing maps for embedded processor selection,” *Microprocessors and Microsystems*, vol. 29, no. 7, pp. 307–315, 2005.

[29] A. Noroozi, “Evaluation methodology and systematic selection of microcontrollers for delfi-n3xt nanosatellite,” 2010.

Appendices

Appendix A

Nucleo-F746ZG CubeMX setup document

1. Description

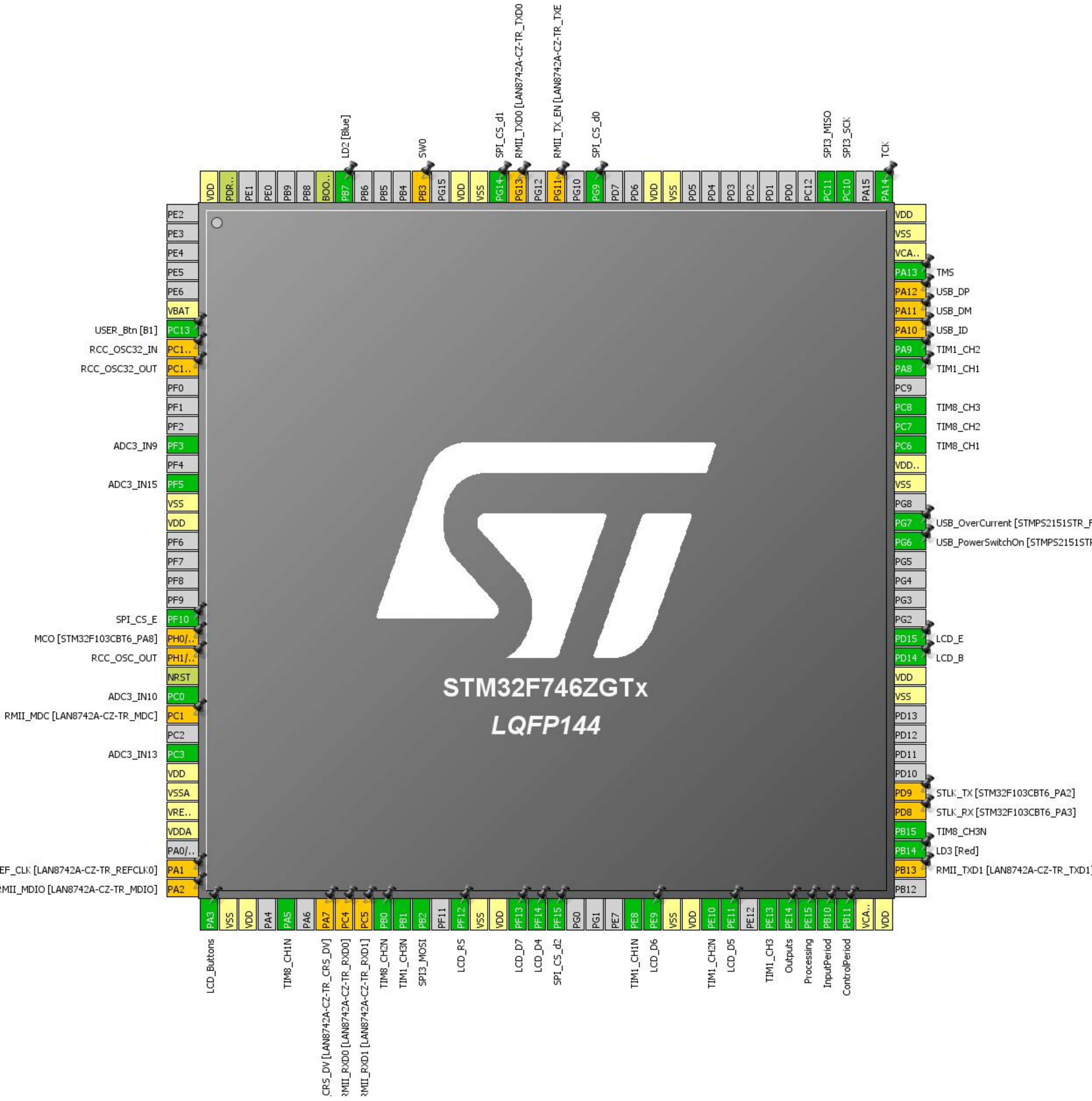
1.1. Project

Project Name	NucleoFinalSetup
Board Name	NUCLEO-F746ZG
Generated with:	STM32CubeMX 4.27.0
Date	10/25/2018

1.2. MCU

MCU Series	STM32F7
MCU Line	STM32F7x6
MCU name	STM32F746ZGTx
MCU Package	LQFP144
MCU Pin number	144

2. Pinout Configuration



3. Pins Configuration

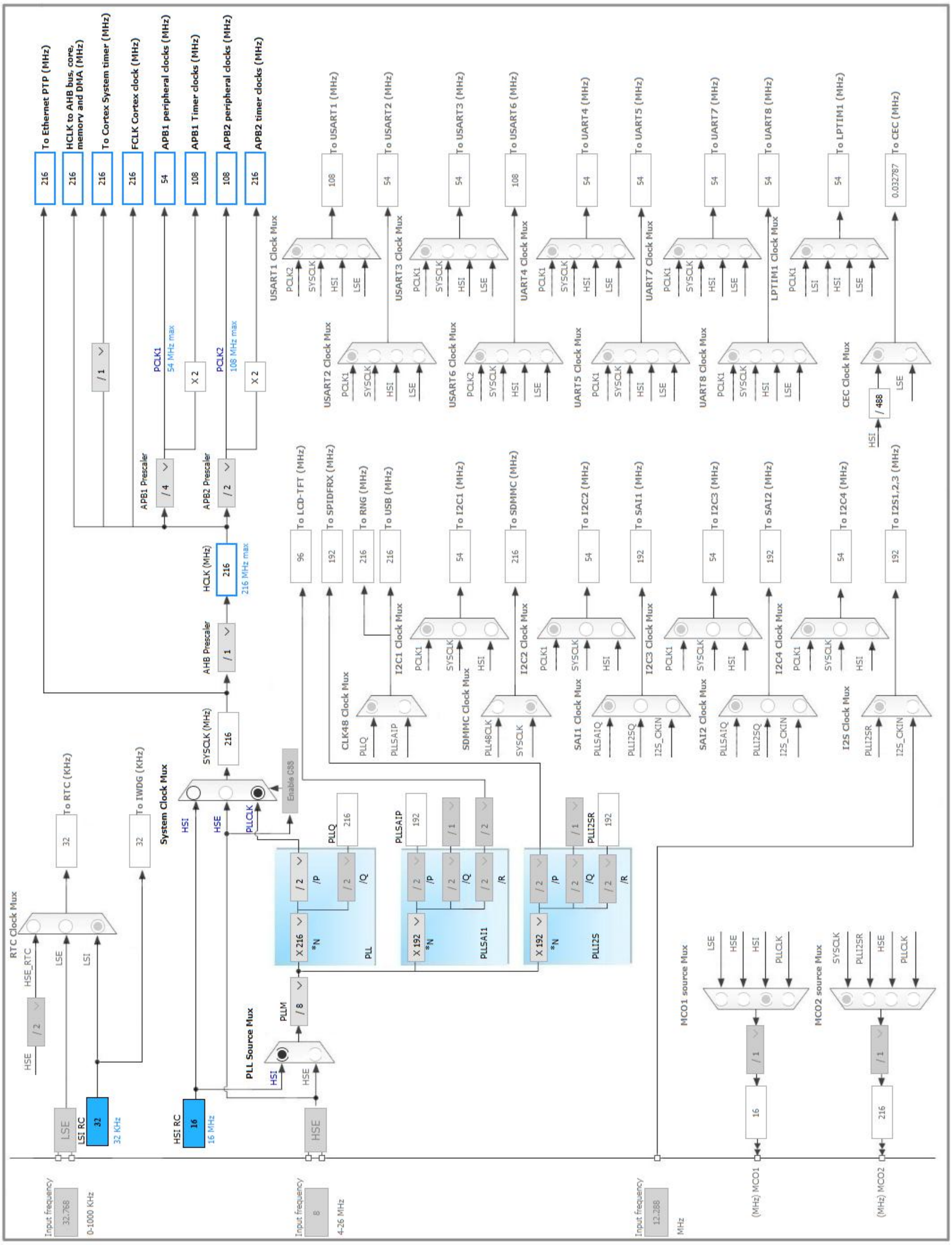
Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
6	VBAT	Power		
7	PC13	I/O	GPIO_EXTI13	USER_Btn [B1]
8	PC14/OSC32_IN *	I/O	RCC_OSC32_IN	
9	PC15/OSC32_OUT *	I/O	RCC_OSC32_OUT	
13	PF3	I/O	ADC3_IN9	
15	PF5	I/O	ADC3_IN15	
16	VSS	Power		
17	VDD	Power		
22	PF10 **	I/O	GPIO_Output	SPI_CS_E
23	PH0/OSC_IN *	I/O	RCC_OSC_IN	MCO [STM32F103CBT6_PA8]
24	PH1/OSC_OUT *	I/O	RCC_OSC_OUT	
25	NRST	Reset		
26	PC0	I/O	ADC3_IN10	
27	PC1 *	I/O	ETH_MDC	RMII_MDC [LAN8742A-CZ-TR_MDC]
29	PC3	I/O	ADC3_IN13	
30	VDD	Power		
31	VSSA	Power		
32	VREF+	Power		
33	VDDA	Power		
35	PA1 *	I/O	ETH_REF_CLK	RMII_REF_CLK [LAN8742A-CZ-TR_REFCLK0]
36	PA2 *	I/O	ETH_MDIO	RMII_MDIO [LAN8742A-CZ-TR_MDIO]
37	PA3	I/O	ADC1_IN3	LCD_Buttons
38	VSS	Power		
39	VDD	Power		
41	PA5	I/O	TIM8_CH1N	
43	PA7 *	I/O	ETH_CRS_DV	RMII_CRS_DV [LAN8742A-CZ-TR_CRS_DV]
44	PC4 *	I/O	ETH_RXD0	RMII_RXD0 [LAN8742A-CZ-TR_RXD0]
45	PC5 *	I/O	ETH_RXD1	RMII_RXD1 [LAN8742A-CZ-TR_RXD1]
46	PB0	I/O	TIM8_CH2N	

Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
47	PB1	I/O	TIM1_CH3N	
48	PB2	I/O	SPI3_MOSI	
50	PF12 **	I/O	GPIO_Output	LCD_RS
51	VSS	Power		
52	VDD	Power		
53	PF13 **	I/O	GPIO_Output	LCD_D7
54	PF14 **	I/O	GPIO_Output	LCD_D4
55	PF15 **	I/O	GPIO_Output	SPI_CS_d2
59	PE8	I/O	TIM1_CH1N	
60	PE9 **	I/O	GPIO_Output	LCD_D6
61	VSS	Power		
62	VDD	Power		
63	PE10	I/O	TIM1_CH2N	
64	PE11 **	I/O	GPIO_Output	LCD_D5
66	PE13	I/O	TIM1_CH3	
67	PE14 **	I/O	GPIO_Output	Outputs
68	PE15 **	I/O	GPIO_Output	Processing
69	PB10 **	I/O	GPIO_Output	InputPeriod
70	PB11 **	I/O	GPIO_Output	ControlPeriod
71	VCAP_1	Power		
72	VDD	Power		
74	PB13 *	I/O	ETH_TXD1	RMII_TXD1 [LAN8742A-CZ-TR_TXD1]
75	PB14 **	I/O	GPIO_Output	LD3 [Red]
76	PB15	I/O	TIM8_CH3N	
77	PD8 *	I/O	USART3_TX	STLK_RX [STM32F103CBT6_PA3]
78	PD9 *	I/O	USART3_RX	STLK_TX [STM32F103CBT6_PA2]
83	VSS	Power		
84	VDD	Power		
85	PD14 **	I/O	GPIO_Output	LCD_B
86	PD15 **	I/O	GPIO_Output	LCD_E
91	PG6 **	I/O	GPIO_Output	USB_PowerSwitchOn [STMPS2151STR_EN]
92	PG7 **	I/O	GPIO_Input	USB_OverCurrent [STMPS2151STR_FAULT]
94	VSS	Power		
95	VDDUSB	Power		
96	PC6	I/O	TIM8_CH1	

Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
97	PC7	I/O	TIM8_CH2	
98	PC8	I/O	TIM8_CH3	
100	PA8	I/O	TIM1_CH1	
101	PA9	I/O	TIM1_CH2	
102	PA10 *	I/O	USB_OTG_FS_ID	USB_ID
103	PA11 *	I/O	USB_OTG_FS_DM	USB_DM
104	PA12 *	I/O	USB_OTG_FS_DP	USB_DP
105	PA13	I/O	SYS_JTMS-SWDIO	TMS
106	VCAP_2	Power		
107	VSS	Power		
108	VDD	Power		
109	PA14	I/O	SYS_JTCK-SWCLK	TCK
111	PC10	I/O	SPI3_SCK	
112	PC11	I/O	SPI3_MISO	
120	VSS	Power		
121	VDD	Power		
124	PG9 **	I/O	GPIO_Output	SPI_CS_d0
126	PG11 *	I/O	ETH_TX_EN	RMII_TX_EN [LAN8742A-CZ-TR_TXEN]
128	PG13 *	I/O	ETH_TXD0	RMII_TXD0 [LAN8742A-CZ-TR_TXD0]
129	PG14 **	I/O	GPIO_Output	SPI_CS_d1
130	VSS	Power		
131	VDD	Power		
133	PB3 *	I/O	SYS_JTDO-SWO	SW0
137	PB7 **	I/O	GPIO_Output	LD2 [Blue]
138	BOOT0	Boot		
143	PDR_ON	Reset		
144	VDD	Power		

** The pin is affected with an I/O function
 * The pin is affected with a peripheral function but no peripheral mode is activated

4. Clock Tree Configuration



5. IPs and Middleware Configuration

5.1. ADC1

mode: IN3

5.1.1. Parameter Settings:

ADCs_Common_Settings:	
Mode	Independent mode
ADC_Settings:	
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	**Enabled ***
Continuous Conversion Mode	**Enabled ***
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	**Enabled ***
End Of Conversion Selection	EOC flag at the end of single channel conversion
ADC_Regular_ConversionMode:	
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 3
Sampling Time	**56 Cycles ***
ADC_Injected_ConversionMode:	
Number Of Conversions	0
WatchDog:	
Enable Analog WatchDog Mode	false

5.2. ADC3

mode: IN9

mode: IN10

mode: IN13

mode: IN15

5.2.1. Parameter Settings:

ADCs_Common_Settings:

Mode	Independent mode

<html><img src='jar:file:D:/Programs/STM32CubeMx/plugins/ipmanager.jar!/com/st/microxplorer/plugins/ipmanager/util/error10x10.png' <font color=red><b> &nbsp ADC_Settings</font></b></html>:

Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	**Enabled ***
Continuous Conversion Mode	**Enabled ***
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	**Enabled ***
End Of Conversion Selection	EOC flag at the end of single channel conversion

ADC_Regular_ConversionMode:

Number Of Conversion	**4 ***
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	**Channel 10 ***
Sampling Time	3 Cycles
Rank	**2 ***
Channel	**Channel 13 ***
Sampling Time	3 Cycles
Rank	**3 ***
Channel	Channel 9
Sampling Time	3 Cycles
Rank	**4 ***
Channel	**Channel 15 ***
Sampling Time	3 Cycles

ADC_Injected_ConversionMode:

Number Of Conversions	0

WatchDog:

Enable Analog WatchDog Mode	false

5.3. SPI3

Mode: Full-Duplex Master

5.3.1. Parameter Settings:

Basic Parameters:

Frame Format	Motorola
Data Size	**16 Bits ***
First Bit	MSB First
Clock Parameters:	
Prescaler (for Baud Rate)	**8 ***
Baud Rate	**6.75 MBits/s ***
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge
Advanced Parameters:	
CRC Calculation	Disabled
NSSP Mode	Enabled
NSS Signal Type	Software

5.4. SYS

Debug: Serial Wire

Timebase Source: SysTick

5.5. TIM1

Clock Source : Internal Clock

Channel1: PWM Generation CH1 CH1N

Channel2: PWM Generation CH2 CH2N

Channel3: PWM Generation CH3 CH3N

5.5.1. Parameter Settings:

Counter Settings:	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	0
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 16 bits value)	0
auto-reload preload	Disable
Trigger Output (TRGO) Parameters:	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)
Trigger Event Selection TRGO2	Reset (UG bit from TIMx_EGR)
Break And Dead Time management - BRK Configuration:	
BRK State	Disable

BRK Polarity	High
BRK Filter (4 bits value)	0

Break And Dead Time management - BRK2 Configuration:

BRK2 State	Disable
BRK2 Polarity	High
BRK2 Filter (4 bits value)	0

Break And Dead Time management - Output Configuration:

Automatic Output State	Disable
Off State Selection for Run Mode (OSSR)	Disable
Off State Selection for Idle Mode (OSSI)	Disable
Lock Configuration	Off
Dead Time	0

PWM Generation Channel 1 and 1N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

PWM Generation Channel 2 and 2N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

PWM Generation Channel 3 and 3N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

5.6. TIM2

Clock Source : Internal Clock

5.6.1. Parameter Settings:

Counter Settings:	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 32 bits value)	**107 ***
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters:	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)

5.7. TIM8

Clock Source : Internal Clock

Channel1: PWM Generation CH1 CH1N

Channel2: PWM Generation CH2 CH2N

Channel3: PWM Generation CH3 CH3N

5.7.1. Parameter Settings:

Counter Settings:	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	0
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 16 bits value)	0
auto-reload preload	Disable
Trigger Output (TRGO) Parameters:	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)
Trigger Event Selection TRGO2	Reset (UG bit from TIMx_EGR)
Break And Dead Time management - BRK Configuration:	
BRK State	Disable
BRK Polarity	High
BRK Filter (4 bits value)	0
Break And Dead Time management - BRK2 Configuration:	
BRK2 State	Disable
BRK2 Polarity	High

BRK2 Filter (4 bits value)	0

Break And Dead Time management - Output Configuration:

Automatic Output State	Disable
Off State Selection for Run Mode (OSSR)	Disable
Off State Selection for Idle Mode (OSSI)	Disable
Lock Configuration	Off
Dead Time	0

PWM Generation Channel 1 and 1N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

PWM Generation Channel 2 and 2N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

PWM Generation Channel 3 and 3N:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

*** User modified value**

6. System Configuration

6.1. GPIO configuration

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
ADC1	PA3	ADC1_IN3	Analog mode	No pull-up and no pull-down	n/a	LCD_Buttons
ADC3	PF3	ADC3_IN9	Analog mode	No pull-up and no pull-down	n/a	
	PF5	ADC3_IN15	Analog mode	No pull-up and no pull-down	n/a	
	PC0	ADC3_IN10	Analog mode	No pull-up and no pull-down	n/a	
	PC3	ADC3_IN13	Analog mode	No pull-up and no pull-down	n/a	
SPI3	PB2	SPI3_MOSI	Alternate Function Push Pull	No pull-up and no pull-down	**Very High** *	
	PC10	SPI3_SCK	Alternate Function Push Pull	No pull-up and no pull-down	**Very High** *	
	PC11	SPI3_MISO	Alternate Function Push Pull	No pull-up and no pull-down	**Very High** *	
SYS	PA13	SYS_JTMS-SWDIO	n/a	n/a	n/a	TMS
	PA14	SYS_JTCK-SWCLK	n/a	n/a	n/a	TCK
TIM1	PB1	TIM1_CH3N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PE8	TIM1_CH1N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PE10	TIM1_CH2N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PE13	TIM1_CH3	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PA8	TIM1_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PA9	TIM1_CH2	Alternate Function Push Pull	No pull-up and no pull-down	Low	
TIM8	PA5	TIM8_CH1N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PB0	TIM8_CH2N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PB15	TIM8_CH3N	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PC6	TIM8_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PC7	TIM8_CH2	Alternate Function Push Pull	No pull-up and no pull-down	Low	
	PC8	TIM8_CH3	Alternate Function Push Pull	No pull-up and no pull-down	Low	
Single Mapped Signals	PC14/OSC32_IN	RCC_OSC32_IN	n/a	n/a	n/a	
	PC15/OSC32_OUT	RCC_OSC32_OUT	n/a	n/a	n/a	
	PH0/OSC_IN	RCC_OSC_IN	n/a	n/a	n/a	MCO [STM32F103CBT6_PA8]
	PH1/OSC_OUT	RCC_OSC_OUT	n/a	n/a	n/a	
	PC1	ETH_MDC	Alternate Function Push Pull	No pull-up and no pull-down	**Very High**	RMII_MDC [LAN8742A-CZ-TR_MDC]

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
					*	
	PA1	ETH_REF_CLK	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_REF_CLK [LAN8742A-CZ-TR_REFCLK0]
	PA2	ETH_MDIO	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_MDIO [LAN8742A-CZ-TR_MDIO]
	PA7	ETH_CRS_DV	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_CRS_DV [LAN8742A-CZ-TR_CRS_DV]
	PC4	ETH_RXD0	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_RXD0 [LAN8742A-CZ-TR_RXD0]
	PC5	ETH_RXD1	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_RXD1 [LAN8742A-CZ-TR_RXD1]
	PB13	ETH_TXD1	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_TXD1 [LAN8742A-CZ-TR_TXD1]
	PD8	USART3_TX	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	STLK_RX [STM32F103CBT6_PA3]
	PD9	USART3_RX	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	STLK_TX [STM32F103CBT6_PA2]
	PA10	USB_OTG_FS_ID	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	USB_ID
	PA11	USB_OTG_FS_DM	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	USB_DM
	PA12	USB_OTG_FS_DP	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	USB_DP
	PG11	ETH_TX_EN	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_TX_EN [LAN8742A-CZ-TR_TXEN]
	PG13	ETH_TXD0	Alternate Function Push Pull	No pull-up and no pull-down	**Very High ***	RMII_TXD0 [LAN8742A-CZ-TR_TXD0]
	PB3	SYS_JTDO-SWO	n/a	n/a	**n/a**	SW0
GPIO	PC13	GPIO_EXTI13	External Interrupt Mode with Rising edge trigger detection	No pull-up and no pull-down	n/a	USER_Btn [B1]
	PF10	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	SPI_CS_E
	PF12	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_RS
	PF13	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_D7

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
	PF14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_D4
	PF15	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	SPI_CS_d2
	PE9	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_D6
	PE11	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_D5
	PE14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	Outputs
	PE15	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	Processing
	PB10	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	InputPeriod
	PB11	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	ControlPeriod
	PB14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD3 [Red]
	PD14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LCD_B
	PD15	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	LCD_E
	PG6	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USB_PowerSwitchOn [STMPS2151STR_EN]
	PG7	GPIO_Input	Input mode	No pull-up and no pull-down	n/a	USB_OverCurrent [STMPS2151STR_FAULT]
	PG9	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	SPI_CS_d0
	PG14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	**Very High ***	SPI_CS_d1
	PB7	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD2 [Blue]

6.2. DMA configuration

DMA request	Stream	Direction	Priority
ADC1	DMA2_Stream0	Peripheral To Memory	Low
ADC3	DMA2_Stream1	Peripheral To Memory	Low

ADC1: DMA2_Stream0 DMA request Settings:

Mode:	**Circular ***
Use fifo:	Disable
Peripheral Increment:	Disable
Memory Increment:	**Enable ***
Peripheral Data Width:	Half Word
Memory Data Width:	Half Word

ADC3: DMA2_Stream1 DMA request Settings:

Mode:	**Circular ***
Use fifo:	Disable
Peripheral Increment:	Disable
Memory Increment:	**Enable ***
Peripheral Data Width:	Half Word
Memory Data Width:	Half Word

6.3. NVIC configuration

Interrupt Table	Enable	Preenmption Priority	SubPriority
Non maskable interrupt	true	0	0
Hard fault interrupt	true	0	0
Memory management fault	true	0	0
Pre-fetch fault, memory access fault	true	0	0
Undefined instruction or illegal state	true	0	0
System service call via SWI instruction	true	0	0
Debug monitor	true	0	0
Pendable request for system service	true	0	0
System tick timer	true	0	0
TIM2 global interrupt	true	0	0
EXTI line[15:10] interrupts	true	0	0
DMA2 stream0 global interrupt	true	0	0
DMA2 stream1 global interrupt	true	0	0
PVD interrupt through EXTI line 16	unused		
Flash global interrupt	unused		
RCC global interrupt	unused		
ADC1, ADC2 and ADC3 global interrupts	unused		
TIM1 break interrupt and TIM9 global interrupt	unused		
TIM1 update interrupt and TIM10 global interrupt	unused		
TIM1 trigger and commutation interrupts and TIM11 global interrupt	unused		
TIM1 capture compare interrupt	unused		
TIM8 break interrupt and TIM12 global interrupt	unused		
TIM8 update interrupt and TIM13 global interrupt	unused		
TIM8 trigger and commutation interrupts and TIM14 global interrupt	unused		
TIM8 capture compare interrupt	unused		
SPI3 global interrupt	unused		
FPU global interrupt	unused		

*** User modified value**

7. Power Consumption Calculator report

7.1. Microcontroller Selection

Series	STM32F7
Line	STM32F7x6
MCU	STM32F746ZGTx
Datasheet	027590_Rev4

7.2. Parameter Selection

Temperature	25
Vdd	3.6

8. Software Project

8.1. Project Settings

Name	Value
Project Name	NucleoFinalSetup
Project Folder	D:\Johan-Omen\Google Drive\Meesters\System
Toolchain / IDE	MDK-ARM V5
Firmware Package Name and Version	STM32Cube FW_F7 V1.12.0

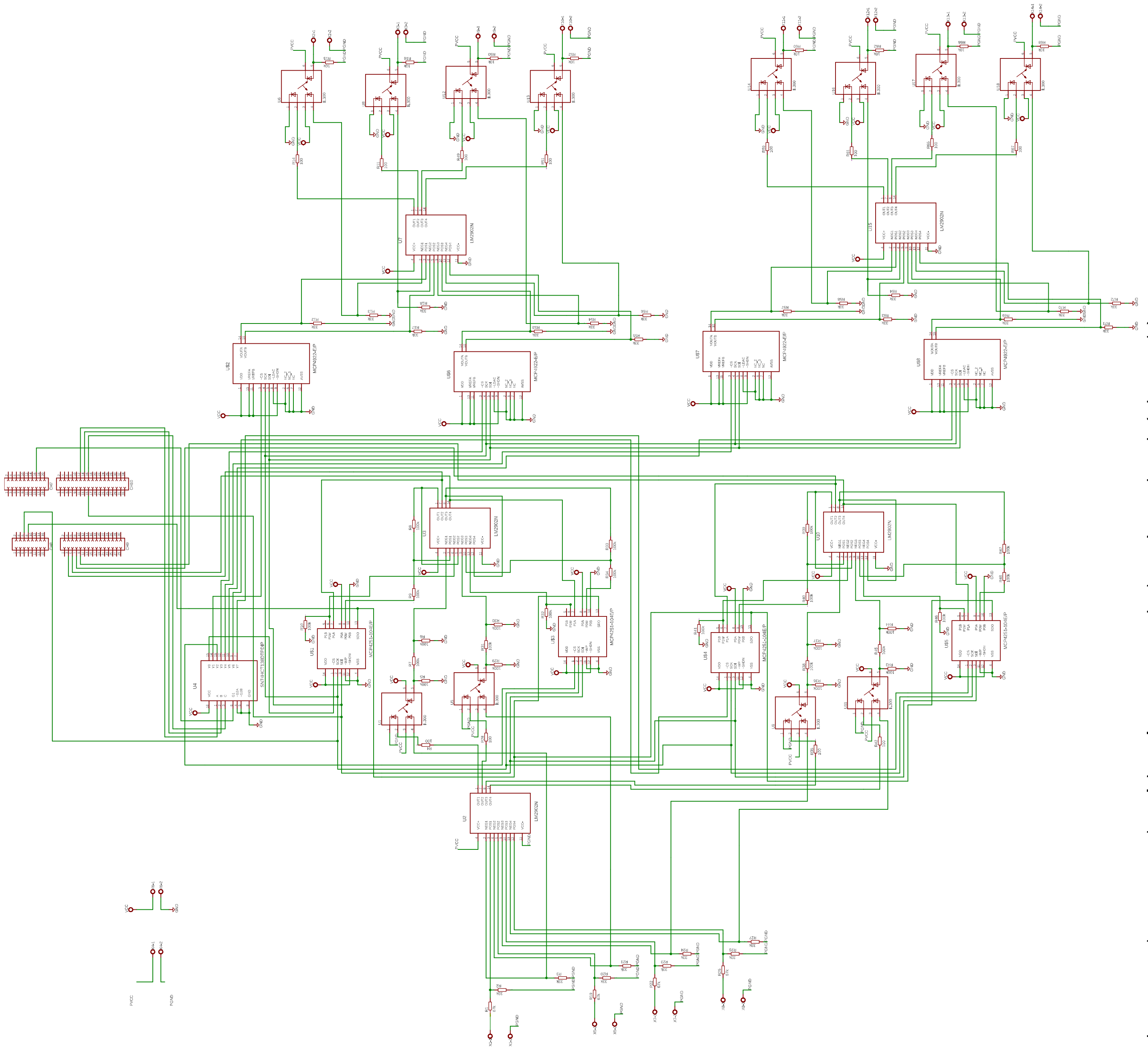
8.2. Code Generation Settings

Name	Value
STM32Cube Firmware Library Package	Add necessary library files as reference in the toolchain project configuration file
Generate peripheral initialization as a pair of '.c/.h' files	No
Backup previously generated files when re-generating	No
Delete previously generated files when not re-generated	Yes
Set all free pins as analog (to optimize the power consumption)	No

9. Software Pack Report

Appendix B

I/F circuit



2018/08/26 11:57 AM f=0.27 D:\Johan-Omen\Documents\eagle\Nucleo IF Protection\IFSchematic.sch (Sheet: 1/1)