Technische Universität Ilmenau

Fakultät für Maschinenbau

Pontificia Universidad Católica del Perú

Escuela de Posgrado

Master Thesis

Implementation of a High Performance Embedded MPC on FPGA using High-Level Synthesis

To achieve the Degree of:

Master of Science (M. Sc.)

In Mechatronics

Submitted by: Antonio Araujo Barrientos

Supervisor (TU Ilmenau): Dr. rer. nat. Abebe Geletu

Supervisor (PUCP): Dr. Elizabeth Villota Cerna

Date and Place: 13/04/2017, Ilmenau, Germany

Implementation of a High Performance Embedded MPC on FPGA using High-Level Synthesis

Master Thesis in Mechatronics

submitted by

Antonio Araujo Barrientos

born on 02 April 1992
in Lima

in the

Simulation and Optimal Processes Group

Department of Computer Science and Automation
Technische Universität Ilmenau

Advisors: **Dr. rer. nat. Abebe Geletu**
Dr. Elizabeth Villota Cerna (PUCP)

Submission Date: **13 April 2017**

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Antonio Araujo Barrientos)
Ilmenau, 13 April 2017

Abstract

Model predictive control (MPC) has been, since its introduction in the late 70's, a well accepted control technique, especially for industrial processes, which are typically slow and allow for on-line calculation of the control inputs. Its greatest advantage is its ability to consider constraints, on both inputs and states, directly and naturally. More recently, the improvements in processor speed have allowed its use in a wider range of problems, many involving faster dynamics. Nevertheless, implementation of MPC algorithms on embedded systems with resources, size, power consumption and cost constraints remains a challenge.

In this thesis, High-Level Synthesis (HLS) is used to implement implicit MPC algorithms for linear (LMPC) and nonlinear (NMPC) plant models, considering constraints on both control inputs and states of the system. The algorithms are implemented in the Zynq® -7000 All Programmable System-on-a-Chip (AP SoC) ZC706 Evaluation Kit, targeting Xilinx's Zynq®-7000 AP SoC which contains a general purpose Field Programmable Gate Array (FPGA). In order to solve the optimization problem at each sampling instant, an Interior-Point Method (IPM) is used. The main computation cost of this method is the solution of a system of linear equations. A minimum residual (MINRES) algorithm is used for the solution of this system of equations taking into consideration its special structure in order to make it computationally efficient. A library was created for the linear algebra operations required for the IPM and MINRES algorithms.

The implementation is tested on trajectory tracking case studies. Results for the linear case show good performance and implementation metrics, as well as computation times within the considered sampling periods. For the nonlinear case, although a high computation time was needed, the algorithm performed well on the case study presented. Because of resources constraints, implementation of the nonlinear algorithm on higher order systems was precluded.

Kurzfassung

Modellprädiktive Regelung (engl: Model Predictive Control (MPC) ist, seit der Einführung in den späten 70er Jahren, eine gut angenommene Regelungstechnik, insbesondere für industrielle Prozesse, die typischerweise langsam sind und die online Steuergröße Berechnung ermöglichen. Ihr größter Vorteil ist die Fähigkeit, Beschränkungen bezüglich der Steuergrößen und der Regelgrößen zu berücksichtigen. In letzter Zeit hat die Verbesserung der Geschwindigkeit der Prozessoren den Einsatz in einer breitere Problemreichweite mit einer schnelleren Dynamik ermöglicht. Allerdings bleibt die MPC Algorithmus-Implementierung in eingebetteten Systeme mit beschränkte Ressourcen, Größe, Energieverbrauch und Kosten eine Herausforderung.

In dieser Arbeit wird die High-Level Synthesis (HLS) benutzt, um implizit MPC Algorithmen für lineare (LMPC) und nichtlineare (NMPC) Regelstrecken zu implementieren, wobei Steuergröße- und Regelgrößenbeschränkungen berücksichtigt werden. Die Algorithmen sind im Zynq®-7000 AP SoC ZC706 Auswertungskit implementiert, wobei auf der Xilinxs Zynq®-7000 AP SoC, der ein allgemeiner Zweck FPGA enthält, abgezielt wird. Ein innere-Punkte Verfahren (engl: Interior-Point Method (IPM)) wird für die Lösung des Optimierungsproblems in jedem Sampling benutzt. Die größte Berechnungskomplexität bei dem IPM ist die Lösung eines linearen Gleichungssystems. Ein minimaler Residuum-Algorithmus (MINRES) wird für die Lösung dieses Gleichungssystem benutzt, wobei die spezielle Struktur berücksichtigt wird, um das Verfahren recheneffizient zu machen. Es wurde eine Bibliothek mit Funktionen für die benötigten linearen Algebra Operationen in den IPM und MINRES Verfahren entwickelt.

Die Implementierung wird in Trajektorieverfolgung Fallstudien getestet. Die Ergebnisse für den linearen Fall zeigen gute Leistungen und Metriken, sowie Rechenzeiten innerhalb des berücksichtigten Taktzeiten. Für den nichtlinearen Fall wurde eine hohe Rechenzeit benötigt. Trotzdem hat der Algorithmus für die vorgestellte Fallstudie gut funktioniert. Infolge der Ressourcenbeschränkungen war die Implementierung des nichtlinearen Algorithmus für Systeme höherer Ordnung verhindert.

Acknowledgments

Special acknowledgment to my supervisor Dr. Abebe Geletu for the valuable guide throughout the development of my thesis. I also thank professor Pu Li for the interest shown and the support. My thankfulness also to Dr. Elizabeth Villota for the comments on my work. Finally, I would like to thank my family and all my friends with whom I spent this year in Ilmenau for the support provided.

Contents

Chapter 1

Introduction

1.1 Motivation

Embedded control is necessary to adequately fulfill the functionality of an embedded system. This control strategy has to be implemented on the chosen processor, satisfying any resources constraints and performance requirements, like for example area constraints, computation time, which is vital for real time applications, and power consumption requirements. Embedded control is ubiquitous, for example in the automotive and aircraft industries for applications like autonomous driving, in robotics or even in more simple systems like household appliances.

Typical processors for embedded applications include CPUs, Graphics Processor Units (GPUs) and Digital Signal Processors (DSPs). These processors execute instructions in a sequential way. Multi core processors can be used to achieve parallelism, as is, in fact, the case for GPUs which contain a large number of programmable cores; however, despite the improvements in performance, these processors continue to possess a fixed hardware architecture. On the other hand, hardware-customizable integrated circuits, namely Field Programmable Gate Arrays (FPGAs), are much more flexible and a wide level of parallelism can be achieved, allowing to trade off resource usage for higher throughput and lower latency in order to meet resource constraints, while achieving a higher power efficiency [14], which is an important issue since most fast autonomous systems are battery driven. In this context, the use of hardware implementations on FPGAs is of great interest and the main topic of this thesis.

Hardware programming is traditionally performed based on Hardware Description Languages (HDLs) like VHDL or Verilog, targeting Application-Specific Integrated Circuits (ASICs) or FPGAs. The latter require less design time, effort and costs, and offer the capability of being reprogrammed at the expense of a lower achievable

performance. The design and verification of embedded systems with this approach demands a lot of time compared with pure software implementations. In order to increase productivity, lot of effort has been put into the development of High-Level Synthesis (HLS) tools, which allow the user to specify the behavior of the algorithm at a higher level of abstraction, commonly using C/C++ language. Implementing the algorithms at this higher level of abstraction permits more complex algorithms to be accelerated in hardware without any knowledge of HDLs being required. This reduces significantly the design time, at the expense of a lower control over the final synthesized design.

Real systems may have constraints on both control inputs and states. For example, in the injection system of a car, the valves have a maximum opening gap and due to consumption matters, the flow through the valves may be restricted to be between certain values; this would represent respectively constraints on a control input and a state for this particular system. Classical control techniques, namely PID control, fail to take these constraints into consideration and therefore operate the plant far from the constraints in order to allow for disturbances to take place, making the operation inefficient [45]. Model Predictive Control (MPC) is one of the few advanced control techniques that has had an important and widespread impact on industrial process engineering, precisely because of its ability to naturally take the constraints of the system into consideration, as well as being easily applied to MIMO systems.

Implicit MPC requires the control input to be calculated at every sampling instant, which initially confined its implementation to systems with slow dynamics, especially in the chemical industry. With the growing improvements in processor speed and growing transistor count, MPC is in recent years being applied to systems with faster dynamics. This poses the challenge of calculating the control input to be applied to the system within a shorter period of time.

MPC involves the solution at every sampling period of an optimization problem. This problem resulting from the control formulation possesses a well defined structure, which can be used to tailor the algorithms for the solution in order to improve the efficiency and speed, while reducing memory requirements.

1.2 Problem Statement

The on-line computation of the MPC problem solution demands a high computational effort and must be obtained within the sampling period of the system under consideration. For fast dynamic systems, this is usually in the order of milliseconds. For the case of embedded control, this solution must be also computed in an efficient way so that

timing, resource, and power consumption requirements are met. An FPGA is chosen to accelerate the most computationally demanding tasks of the MPC problem solution.

1.3 Objectives

This thesis aims to design a high performance embedded MPC implementation accomplishing the following objectives:

- To study FPGAs hardware architecture for efficient implementations.
- To study the use of HLS tools on modern FPGA hardware.
- To design a high-level performance implementation of an MPC scheme for the embedded control of fast autonomous systems.
- To validate the viability of the implementation through case studies and real-time MPC control of systems with fast dynamics.

1.4 Thesis Organization

The content of this thesis is organized as follows:

- Chapter 2 starts with a brief overview of the different processor types available for embedded control. It gives then an introduction to FPGAs and the ways they can be programmed, namely using HDLs or HLS, expatiating on the use of the latter as an important topic of this work.
- Chapter 3 presents the principle of MPC and the formulations for Linear Model Predictive Control (LMPC) and Nonlinear Model Predictive Control (NMPC) considered in this thesis. Special care is taken in describing the structure of the corresponding Quadratic Programming (QP) and Nonlinear Programming (NLP) problems that result from the approach chosen for the formulation.
- Chapter 4 describes the primal-dual Interior-Point Method (IPM) for the solution of QP and NLP problems, and the special structure arising from the control approach, which is exploited to improve computation time and memory requirements.
- Chapter 5 describes in detail the implementation of the IPM solvers. The chapter starts with an overview of the related work done regarding hardware MPC implementations. Then, a description of the compressed storage formats used for memory utilization reduction is given, the employed hardware and software tools

are then introduced. Next, LMPC and NMPC algorithm implementations are described step by step. An explanation of the optimization directives used for exploiting parallelization in the FPGA is then given. This chapter ends with a description of how the solvers implemented using HLS are integrated on a design targeting a Zynq® All Programmable System-on-a-Chip (AP SoC).

- Chapter 6 presents a description of the models considered for the case studies. Trajectory tracking problems were solved and the results obtained are shown and described, together with performance and utilization metrics.
- Chapter 7 concludes this thesis with a summary of the work done and the obtained results, as well as possible future research directions.

Chapter 2

High-Level Synthesis

ASICs and FPGAs are the most popular hardware programming technologies. They are traditionally programmed with Hardware Description Languages (HDLs) like VHDL or Verilog, which make the design process long and require expertise. Another approach is the use of HLS tools. This chapter introduces the use of HLS for FPGA programming.

2.1 Embedded Systems Processors

This section provides a brief overview of the processor choice for embedded systems, based on the description by Vahid and Givargis presented in [64].

An embedded system is any computer system different from a PC with constrained features like size, cost, performance and power consumption designed to perform a single task. Its most important part is the processor, which performs the required computations and manages the entire operation. Depending on the architecture used for the desired functionality it can be one of the following:

- General-purpose processor, suits a wide range of applications, which reduces design time and related costs at the expense of achievable performance. Reduced instruction set computing (RISC) processors, like those with the ARM architecture widely used in embedded applications, are an example of this category.

- Single-purpose processor, is customized for a single specific functionality. Is also called accelerator and will be considered in this thesis for the hardware implementation. This type of processor allows for fast performance and small size and power consumption, at the expense of higher development costs and lower flexibility.

- Application-specific processor, developed for a particular type of applications that have similar characteristics. Properties of this processor type can be placed in-between the previous two types. An important example of an application-specific processor are DSPs, which allow math intensive operations to be executed on digital signals, like video and audio.

Beside the processor technology previously described, the Integrated Circuit (IC) technology also plays an important role in determining how the digital gate-level implementation is mapped to the IC. Bottom layers on an IC form transistor-based logic gates and top layers connect these gates through wires. Three different technologies are to be considered:

- In full-custom IC technology, all layers are optimized for a particular implementation. Excellent performance can be achieved while retaining a small size and power consumption at the expense of elevated development costs and design time. Only economically viable for large production volumes, like commercially available microprocessors.

- Semi-custom, for this case lower layers are fully or partially built, remaining the upper layers for the developer to finish. Present good performance and size, with smaller costs as full-custom ICs. ASICs are an example of this technology.

- In programmable logic devices all layers already exist, circuits are programmed by creating or destroying connections between gate-connecting wires. FPGAs are the most popular devices of this type. As an important topic of this thesis, they will be treated separately in section 2.2.

Any type of processor technology can be implemented on any type of IC technology, for this thesis, a hardware accelerator was implemented on an FPGA.

2.2 Field Programmable Gate Array (FPGA)

An FPGA is a semiconductor device, namely an IC, that can be programmed after fabrication and can be dynamically reconfigured for different tasks. This distinguishes them from ASICs, which are more application-specific [72]. The fabrication cost of an ASIC for a given application is still very high, making it economically viable only for applications demanding a lot of units, sometimes in the range of millions.

On normal processors, instructions are executed in a sequential manner. Initial attempts to improve execution runtime for a given application relied on the increase

in the processor clock frequency so that instructions would execute faster. Afterwards, attempts considering specialized processors like DSPs or GPUs became popular. The problem with these approaches is that regardless of a growing processor speed the memory accesses are limited by a fixed unified memory space. This led to the introduction of multi core processors and parallel execution which are preferred today. Unlike processors, FPGAs don't execute programs as a sequence of instructions, they allow for parallel implementation of an algorithm without the restriction of cache and a fixed memory space, FPGAs are therefore ideal for implementing hardware accelerators to improve algorithm performance [74].

FPGAs give the designer the opportunity to obtain the power consumption savings and the performance of a custom implementation without the cost and complexity of developing an ASIC. In addition, reprogrammability and the increasing logic density, consisting up to two million logic cells on modern devices allows for the implementation of complex algorithms while exploiting the inherently parallel nature of a custom circuit [74]. All these properties make FPGAs a good choice for numerous applications in different markets like aerospace, military and defense, audio and video processing, automotive, medical, communications and industrial applications [18, 72].

In this thesis, a Xilinx® device is utilized for the implementation of an MPC algorithm. Using Xilinx®-specific terms, general purpose FPGA logic fabric is composed basically of slices, Configurable Logic Blocks (CLBs) and Input/Output Blocks (IOBs) as well as special resources like DSP48E1 and Block RAMs (BRAMs). Figure 2.1 shows the basic architecture of the Programmable Logic (PL) part for the Zynq® device used for this work. Next, a brief explanation extracted from [21] is given:

- Lookup Table (LUT): Resource capable of implementing simple logic functions, small ROM, small RAM or a shift register. LUTs can be combined to form larger units.
- Slice: Unit that contains resources for implementing combinatorial and sequential logic circuits. Composed basically by LUTs and Flip-Flops (FFs).
- CLB: Each CLB contains two logic slices and is positioned next to a switch matrix, which makes possible the connection between elements within a CLB and from one CLB to other resources.
- IOB: Are the interface between the FPGA resources and the physical pads of the device. Each IOB can handle a 1-bit input or output signal.
- BRAM: This special purpose component is thought for dense memory requirements. Can implement RAM, ROM, and FIFO buffers. Using BRAMs allows a large

amount of data (up to 36Kb for Zynq® devices) to be stored in a small physical space on the device. The alternative, implemented only with LUTs is called distributed memory.

- DSP48E1 Are dedicated silicon resources for implementing high-speed arithmetic on signals with medium to long arithmetic word lengths. Comprise a pre-adder/subtractor, a multiplier, and a post-adder/subtractor with a logic unit.

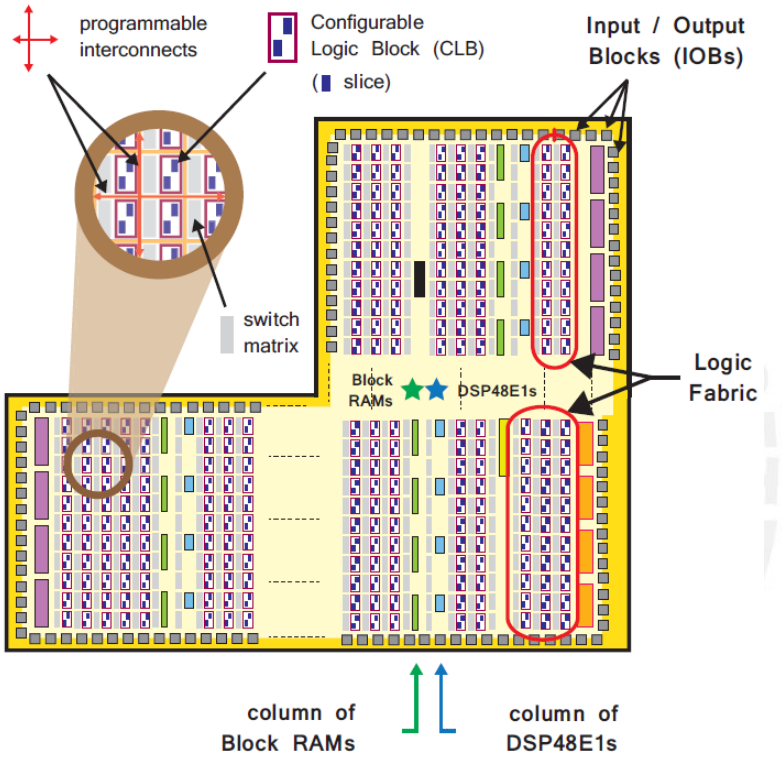


Figure 2.1 – Logic fabric on the Zynq 7000 AP SoC[21]

2.3 Traditional FPGA Programming Languages

VHDL and Verilog are the most widely used HDLs, both are IEEE standards and are supported by synthesis tools for ASICs and FPGAs [17]. VHDL was developed in the 70's and 80's by the U.S. Department of Defense and was proposed as an IEEE standard in 1986, being adopted one year later [55]. Verilog was adopted as a standard in 1995 [65].

Hardware description involves two major aspects: behavior and structure. Behavioral description requires a language which allows the behavior or functionality to be declared

independent of structural or design aspects. The description of structure requires the language to express this hardware structure irrespective of the behavioral models that might be applied to it [48]. HDLs allow the design to be portable and independent of technology, allowing the engineer to focus on functionality. After an HDL design is completed it allows for simulation, synthesis, place and route procedures and finally it can be simulated again to verify results as well as timing and design constraints fulfillment [55].

2.4 High-Level Synthesis (HLS)

A great disadvantage of FPGA programming using the languages described in section 2.3 is the long design and implementation times when compared to alternative fixed architectures like GPUs or DSPs which are programmed only in software using languages with a higher level of abstraction. The continuously increasing complexity of functionalities and applications implemented, and the additional complexity as the result of intending to meet design constraints make necessary the use of tools that increase design productivity [20]. In this context, HLS tools have been developed to ease and accelerate the hardware design process and have started gaining acceptance since the 2000's. A detailed study on HLS history and the reasons why early attempts failed to gain acceptance is presented in [46], a review of its benefits is presented in [20]. A detailed overview of the HLS tools available up to 2011 can be found in [47]. For the work presented in this thesis, Xilinx® hardware was used, consequently, Vivado® HLS software was employed for generating a Register Transfer Level (RTL) model which was later programmed in the FPGA. For a detailed description of the tool used for this work refer to [76]. Following, a brief introduction to HLS extracted from [76] is presented.

Vivado® HLS transforms a description specified in C, C++ or System C into an RTL implementation. This higher level of abstraction programming allows both software and hardware programmers to benefit from both domains easing the design process and increasing productivity.

The HLS process consists basically of scheduling and binding operations. During scheduling, it is defined in which clock cycle each operation will be performed and during binding hardware resources to implement the scheduled operations are assigned. This process is done taking into consideration the target device, timing constraints and any optimization directive specified. Optimization directives give the programmer the possibility to influence the final RTL implementation in the desired way to improve the results. Unlike in HDLs where the interface of the design has to be explicitly specified by the programmer, in designs specified with HLS it is inferred from the arguments and

return variables of the function to be synthesized, also called top function, so care must be taken in the programming style.

Vivado® HLS uses three metrics to measure the performance of a design:

- Area: Quantifies the resources (BRAMs, DSP48Es, FFs and LUTs) required for the implementation considering the target device.
- Latency: Number of clock cycles required for the function to compute all output values.
- Initiation Interval (II): Number of clock cycles before new data can be input to the function.

When designing with Vivado® HLS special care must be taken as not all C/C++ constructs are supported for synthesis; for example, no memory allocation is permitted, so all memory requirements must be known at compile time.

After developing the code with the desired functionality, it can be simulated to verify correctness. The first step is to perform a *C simulation*. For this purpose, an appropriate *test bench* written also in a high-level language is required, which should provide inputs to the top function and compare the outputs with known solutions to verify the operations are correctly performed. Once the functionality has been verified the *C synthesis* is performed. In this step, all user-defined optimization directives are considered and different solutions can be created to facilitate the comparison process. *C synthesis* outputs an RTL design which can again be verified during *C/RTL cosimulation*. A great advantage of HLS is that the same *test bench* used for *C simulation* is used in this step, which reduces significantly the verification time [47]. Finally, the design can be exported as an Intellectual Property (IP) block for its integration in a more complex design using Xilinx®'s IP Integrator.

Apart from the metrics used by Vivado HLS presented previously in this section, there are also three important concepts to take into consideration [74]:

- Clock Frequency: Is defined as the longest time it takes a signal to travel from a source register to a sink register.
- Throughput: Expresses the rate at which data can be passed through the system.
- Pipelining: Is the process of inserting registers between computation blocks in order to get smaller segments and allow for concurrent operation execution. Can increase latency in absolute number of clock cycles but increases performance by allowing a higher clock frequency and improving throughput. The latency caused by pipelining is a trade-off to consider during FPGA design.

2.5 Summary

This chapter presented a brief description of the processor choice on embedded applications focusing on the architecture and the IC technology. A description of FPGA technology and why it is a good choice for parallel computing was given. HDLs were briefly described, giving more emphasis on HLS as a productivity oriented alternative for hardware programming.

Chapter 3

Model Predictive Control (MPC)

MPC is a model-based control technique which determines the optimal control inputs, based on the current measurement (or estimation) of the states, such that a desired reference is reached or followed. This is accomplished by predicting the system's future response and minimizing a cost function, while having the constraints on the states and control inputs under consideration.

MPC was originally developed in the petrochemical industry. The first description of MPC applications was presented in 1976 by Richalet et al. at a conference and then summarized in 1978 in an *Automatica* paper [60]. The software proposed was called IDCOM, an acronym for Identification and Command. Later, at the 1979 National AIChE meeting, engineers of Shell Oil presented their MPC technology named dynamic matrix control (DMC) [57]. In recent years the scope of applications has increased drastically, covering areas including chemicals, food processing, automotive and aerospace applications; a detailed overview of MPC and industrial applications can be found in [56, 57].

3.1 Principle of MPC

MPC has had a widespread impact on industrial process control and is recently being adopted for a much wider range of applications, made possible by the constant increase in computing speed and power [45]. Its main advantage over other control techniques is the ability to naturally handle the constraints on states and control inputs of the system, allowing for a more efficient operation. It is also easily extended to multivariable systems.

The principle of MPC consists in finding, at each sampling instant, the optimal control inputs along a predefined time horizon, often called prediction horizon, in order to follow the desired reference. This is achieved by predicting the future outputs based on an

explicit model while minimizing a cost function and having constraints on the system under consideration. Once the optimal control inputs for the current prediction horizon have been computed, only the inputs corresponding to the first interval of the prediction horizon are applied to the system. The process is repeated on every sampling instant shifting the prediction horizon accordingly, an approach known as *receding horizon* strategy [45]. At every sampling instant, the new measurement available, or estimation if not all states are measured, is used as input for the optimization problem.

3.2 Linear Model Predictive Control (LMPC)

In case the system to be controlled is linear—or a linearization around an operating point is considered—, the cost function is quadratic, and both the states and control inputs constraints are linear, then the MPC problem is an LMPC problem. The last two conditions are normally easily satisfied since the cost function is chosen to be quadratic and the constraints on the system typically appear as box constraints (simple bounds). The advantage of this formulation is that the optimization problem can be formulated as a QP problem, which is convex and therefore assures a global minimum can be found [52].

3.2.1 LMPC Formulation

Consider the following linear time-varying discrete system:

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k \,, \\ y_k &= C x_k \,, \end{aligned} \tag{3.1}$$

where $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$ and $y_k \in \mathbb{R}^p$ represent the states, inputs, and outputs of the system, respectively. Throughout this thesis, the notation $x_k = x(k) = x(t = kT_s)$, for $k = 0, 1, 2, ...$ and T_s representing the sampling period will be considered.

The LMPC algorithm will compute the optimal control inputs u^* over a predefined control horizon (N_u), required to follow the desired reference, and use the model in (3.1) to predict the future states over the prediction horizon (N_p). Applied inputs to the system are considered to be piecewise constant over the sampling period. N_u is normally chosen to be less than or equal to N_p, in the following, they will be considered

to be equal and will be denoted with the letter N. In order to find the optimal solution, the following cost function is to be minimized at each sampling instant k:

$$J(k) = \frac{1}{2}\left(\sum_{i=1}^{N}\|x(k+i|k) - x_r(k+i|k)\|^2_{Q_i} + \sum_{i=0}^{N-1}\|u(k+i|k) - u_r(k+i|k)\|^2_{R_i}\right) \quad (3.2)$$

where x_r and u_r denote the reference vectors for the states and control inputs respectively. The expression $\|x\|^2_Q$ represents the quadratic form x^TQx and $x(k+i|k)$ denotes the values predicted for the states at the instant $k+i$ using the measurement or estimation available at instant k, the same applies for inputs u. It may be preferable to consider the change in the control input Δu instead of the control input itself; in that case, the cost function (3.2) can be easily modified. Considering the dynamics of the system and the constraints the following optimization problem is solved every time a new measurement is available:

$$\begin{aligned}\underset{u,x}{\text{minimize}}\, J(k) = &\frac{1}{2}\sum_{i=1}^{N}\|x(k+i|k) - x_r(k+i|k)\|^2_{Q_i} + \\ &\frac{1}{2}\sum_{i=0}^{N-1}\|u(k+i|k) - u_r(k+i|k)\|^2_{R_i}\,,\end{aligned} \quad (3.3a)$$

subject to:

$$x(k|k) = x_0\,, \quad (3.3b)$$

$$x(k+i+1|k) = A_i x(k+i|k) + B_i u(k+i|k)\,, \qquad i = 0,1,..,N-1\,, \quad (3.3c)$$

$$D_i x(k+i|k) \leqslant d_i\,, \qquad i = 1,2,...N\,, \quad (3.3d)$$

$$F_i u(k+i|k) \leqslant f_i\,, \qquad i = 0,1,...N-1\,. \quad (3.3e)$$

Equation (3.3c) considers the general case of a linear time-varying system, where the subindex in the matrices A and B indicates their value at instant i in the prediction horizon, starting from the actual sampling period k. Equations (3.3d) and (3.3e) impose constraints on the states and control inputs respectively, they appear usually as box constraints ($min \leq var \leq max$) and can be formulated with the matrices and vectors shown in equation (3.4). In case a given state or input is not constrained then the corresponding terms in the matrices and vectors in (3.4) are simply set to zero without altering the problem structure.

$$D_i = \begin{bmatrix} I \\ -I \end{bmatrix} \quad , \quad d_i = \begin{bmatrix} x_{max} \\ -x_{min} \end{bmatrix} , \tag{3.4a}$$

$$F_i = \begin{bmatrix} I \\ -I \end{bmatrix} \quad , \quad f_i = \begin{bmatrix} u_{max} \\ -u_{min} \end{bmatrix} . \tag{3.4b}$$

For the case of LMPC the optimization problem to solve at each sampling interval can be formulated as a QP problem of the form:

$$\underset{\xi}{\text{minimize}}\, J(k) = \frac{1}{2}\xi^T Q \xi + q^T \xi \,, \tag{3.5a}$$

subject to:

$$A\xi = b \,, \tag{3.5b}$$

$$C\xi \leqslant d \,. \tag{3.5c}$$

Where the vector ξ represents the optimization variables and its structure depends on the solution approach. For the problem to be convex, matrix Q must be positive semidefinite [52].

3.2.2 Explicit LMPC

From the LMPC formulation (3.3) is clear that when considering a time-invariant system and assuming constraints and references remain constant, only the initial state will change at every sampling instant. In explicit LMPC the optimization problem (3.3) is solved off-line for a number of initial states of interest in order to obtain an explicit dependence between the control action u and the states x of the system [8]. This approach alleviates the on-line computational cost of MPC allowing for faster systems to be considered; nevertheless, the memory requirements increase with the size of the problem. It is, therefore, more appropriate for small problems.

3.2.3 Implicit LMPC

Implicit LMPC requires the solution of (3.3) to be computed on-line on every sampling instant, which demands a significant computational effort. For many years, this led MPC to be applied only to slow processes which allowed this computation to be performed within the sampling period [28]. In recent years, the application range of MPC has

widened as the computational capacity of controllers improved. Two approaches for the formulation of (3.5) will be considered next.

3.2.3.1 Dense Approach

In the dense approach, only the control inputs over the prediction horizon are considered as optimization variables, and the states have to be expressed as a function of the former and the initial state. This decision for ξ results in a QP problem with dense matrices Q, A and C (refer to (3.5)), with the advantage of a reduced order as in comparison with the case when both the control inputs and the states are considered for the optimization as explained next. For a detailed description of this approach refer to [51].

3.2.3.2 Sparse Approach

In the sparse approach, the states are considered together with the control inputs as optimization variables. This leads to a bigger optimization problem, but with the advantage of having sparse matrices. This sparsity and the well defined structure of the problem can be easily exploited to develop tailored algorithms for its solution in order to improve efficiency and reduce memory requirements, as presented in [28, 58].

The vector of optimization variables ξ can be selected as follows

$$\xi = \begin{bmatrix} u(k|k) \\ x(k+1|k) \\ u(k+1|k) \\ x(k+2|k) \\ \vdots \\ x(k+N-1|k) \\ u(k+N-1|k) \\ x(k+N|k) \end{bmatrix}. \tag{3.6}$$

With this selection, formulation (3.5) results in matrices and vectors of the following dimensions: $Q \in \mathbb{R}^{N_{OV} \times N_{OV}}$, $A \in \mathbb{R}^{N_{EC} \times N_{OV}}$, $C \in \mathbb{R}^{N_{IC} \times N_{OV}}$, $q \in \mathbb{R}^{N_{OV}}$, $b \in \mathbb{R}^{N_{EC}}$ and $Q \in \mathbb{R}^{N_{IC}}$, where N_{OV}, N_{EC} and N_{IC} represent the number of optimization variables, number of equality constraints and number of inequality constraints respectively and are defined in (3.7).

$$N_{OV} = N(n+m)\,, \tag{3.7a}$$
$$N_{EC} = Nn\,, \tag{3.7b}$$
$$N_{IC} = 2N(n+m)\,. \tag{3.7c}$$

The initial state vector $x(k|k)$ is not considered as optimization variable because it is known at every sampling instant and is regarded as an input. In order to have equation (3.3) in the form of (3.5), considering the optimization variables in the order previously presented, the problem has to be reformulated. The cost function (3.3a) can be put in the form (3.8), where the linear terms appear in case the desired reference is different to zero. The weight matrices are usually held constant over the prediction horizon, except for the penalty on the states deviation at the end of it; the cost function is therefore written with the penalty terms at the end of the horizon isolated and their weight matrix and vector represented by P and p respectively.

$$\begin{aligned}\underset{u,x}{\text{minimize}}\, J(k) = &\sum_{i=1}^{N-1}\left(\frac{1}{2}\|x(k+i|k)\|^2_{Q_i} + q_i^T x(k+i|k)\right) + \\ &\sum_{i=0}^{N-1}\left(\frac{1}{2}\|u(k+i|k)\|^2_{R_i} + r_i^T u(k+i|k)\right) + \\ &\frac{1}{2}\|x(k+N|k)\|^2_P + p^T x(k+N|k)\,.\end{aligned} \tag{3.8}$$

In the following the matrices and vectors for the equation (3.5) are derived.

Cost Function

Expanding the cost function in (3.8) gives:

$$\begin{aligned}J(k) = &\frac{1}{2}u_0^T R_0 u_0 + r_0^T u_0 + \\ &\frac{1}{2}x_1^T Q_1 x_1 + \frac{1}{2}u_1^T R_1 u_1 + q_1^T x_1 + r_1^T u_1 + \\ &\frac{1}{2}x_2^T Q_2 x_2 + \frac{1}{2}u_2^T R_2 u_2 + q_2^T x_2 + r_2^T u_2 + \\ &\qquad\qquad\vdots \\ &\frac{1}{2}x_{N-1}^T Q_{N-1} x_{N-1} + \frac{1}{2}u_{N-1}^T R_{N-1} u_{N-1} + q_{N-1}^T x_{N-1} + r_{N-1}^T u_{N-1} + \\ &\frac{1}{2}x_N^T P x_N + p^T x_N\,,\end{aligned}$$

where x_i and u_i represent $x(k+i|k)$ and $u(k+i|k)$ respectively.

This equation can be rearranged and put in the quadratic form $\frac{1}{2}\xi^T Q\xi + q^T\xi$ with matrix Q and vector q as expressed in (3.9) and (3.10).

$$Q = \begin{pmatrix} R_0 & & & & & & & \\ & Q_1 & & & & & & \\ & & R_1 & & & & & \\ & & & Q_2 & & & & \\ & & & & \ddots & & & \\ & & & & & Q_{N-1} & & \\ & & & & & & R_{N-1} & \\ & & & & & & & P \end{pmatrix}, \tag{3.9}$$

$$q = \begin{bmatrix} r_0^T & q_1^T & r_1^T & q_2^T & \dots & q_{N-1}^T & r_{N-1}^T & p^T \end{bmatrix}^T. \tag{3.10}$$

Equality Constraints

Expanding the equality constraints (3.3c) corresponding to the dynamics of the plant:

$$\begin{aligned} x(k+1|k) &= A_0 x(k|k) + B_0 u(k|k)\,, \\ x(k+2|k) &= A_1 x(k+1|k) + B_1 u(k+1|k)\,, \\ &\vdots \\ x(k+N-1|k) &= A_{N-2} x(k+N-2|k) + B_{N-2} u(k+N-2|k)\,, \\ x(k+N|k) &= A_{N-1} x(k+N-1|k) + B_{N-1} u(k+N-1|k)\,. \end{aligned}$$

This equations can be rearranged and put in the form of a system of linear equations $Ax = b$ with matrix A and vector b as expressed in (3.11) and (3.12).

$$A = \begin{bmatrix} -B_0 & I & & & & & & \\ & -A_1 & -B_1 & I & & & & \\ & & & -A_2 & -B_2 & I & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & -A_{N-2} & -B_{N-2} & I \\ & & & & & & -A_{N-1} & -B_{N-1} \; I \end{bmatrix}, \tag{3.11}$$

$$b = \begin{bmatrix} A_0 x(k|k) \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.12}$$

Inequality Constraints

The inequality constraints on states and control inputs appear as box constraints of the form $x_{min} \leq x \leq x_{max}$. Considering equation (3.4) the matrix and vector for $Cx \leq d$ can be formed.

$$C = \begin{pmatrix} F_0 & & & & & & \\ D_1 & & & & & & \\ & F_1 & & & & & \\ & & D_2 & & & & \\ & & & \ddots & & & \\ & & & & D_{N-1} & & \\ & & & & & F_{N-1} & \\ & & & & & & D_N \end{pmatrix}, \tag{3.13}$$

$$d = \begin{bmatrix} f_0 \\ d_1 \\ f_1 \\ d_2 \\ \vdots \\ d_{N-1} \\ f_{N-1} \\ d_N \end{bmatrix}. \tag{3.14}$$

3.3 Nonlinear Model Predictive Control (NMPC)

The LMPC technique described previously in this chapter relies on a linear model of the system; however, most real systems are nonlinear. As increasingly higher performance specifications and constraints imposed not only by the system itself but also by environmental regulations and economical considerations are considered, systems are required to operate closer to the constraints, precluding an approach based on a linearization to be applied, making the use of a nonlinear approach, namely NMPC necessary to achieve

the desired performance [23]. In addition to nonlinear dynamics, NMPC considers the general case of nonlinear constraints and a non-quadratic objective function. As in the case of LMPC, an optimization problem must be solved on every sampling interval, but with the important difference that in this case the problem may be non convex and stability and robustness are not easy to study. For a more detailed study on NMPC refer to [2, 23, 25] and the references therein.

3.3.1 NMPC Formulation

Consider the following nonlinear system:

$$\dot{x} = f(x(t), u(t), t)\,, \tag{3.15}$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$ represent the states and inputs of the system respectively. The main idea, as in the linear case, is to obtain the optimal control inputs required to follow a desired trajectory while minimizing a scalar valued cost function, having the constraints on inputs and states under consideration and following the receding horizon strategy. As a result, the following Optimal Control Problem (OCP) is to be solved on every sampling interval:

$$\underset{u(t)}{\text{minimize}}\, J(t) = \phi(x(t_k + NT_s), t_k + NT_s) + \int_{t_k}^{t_k+NT_s} f_0(x(t), u(t), t)dt \tag{3.16a}$$

subject to:

$$x(t_k) = x_0\,, \tag{3.16b}$$

$$\dot{x} = f(x(t), u(t), t)\,, \qquad t \in [t_k, t_k + NT_s]\,, \tag{3.16c}$$

$$g(x(t), u(t), t) \leq 0\,, \qquad t \in [t_k, t_k + NT_s]\,. \tag{3.16d}$$

Equation (3.16c) represents the nonlinear dynamics of the plant over the prediction horizon N and (3.16d) represents the inequality constraints on both states and inputs of the system. Function ϕ is a function of the state vector at the end of the prediction horizon. This continuous-time OCP can be solved using dynamic programming [7, 9], direct or indirect methods [11]. A survey of different numerical methods for the solution of OCPs can be found in [10]. Direct methods are preferred as they allow for larger problems to be solved and can easily deal with inequality constraints.

Direct methods solve problem (3.16) by discretizing it over the considered time interval, in this case, the prediction horizon. This step can be done using any discretization approach. The main task is the discretization of the system dynamics (3.16c), which

together with the initial state (3.16b) form an Initial Value Problem (IVP). The solution of this IVP can be obtained through one-step methods like Euler, Runge-Kutta or collocation methods, or through multi-step methods like Backward Differentiation Formula (BDF) [22]. The method chosen has a direct influence on the performance of the controlled system. The Euler explicit method, for example, is the simplest to implement but requires a smaller step size to achieve a good performance as compared with other methods which may preclude its usage as this increases the computation requirements. As a result of discretization, problem (3.16) is turned into the Nonlinear Programming (NLP) problem shown in equation (3.17).

$$\underset{\xi}{\text{minimize}}\, f(\xi)\,, \tag{3.17a}$$

subject to:

$$C_E(\xi) = 0\,, \tag{3.17b}$$

$$G(\xi) \leq 0\,. \tag{3.17c}$$

As in the LMPC formulation, vector ξ can be chosen to consider only the control inputs over the prediction horizon (sequential approach) or to consider both inputs and states, which is called simultaneous approach. The latter approach is used in this thesis, which leads to a vector of optimization variables of the form (3.6). Vectors C_E and G are formed by scalar functions and have dimensions N_{EC} and N_{IC} respectively as shown in (3.7). The cost function f is normally chosen to be quadratic, of the form (3.8) which can be put in the form of (3.5a). The function C_E will depend on the discretization approach chosen as previously mentioned. For this thesis, inequality constraints will be considered as in the LMPC formulation, arising from box constraints on the inputs and the states.

3.4 Summary

This chapter presented the basic MPC formulation and its advantages. This control technique has a wide range of applications, mostly on industrial processes. In recent years, however, systems with faster dynamics are being also considered for its implementation and computationally efficient solvers are therefore needed to deal with faster sampling rates. Sparse formulations where described in detail for both LMPC and NMPC approaches. The solution approach for both the QP and the NLP problems formulated in this chapter will be explained in chapter 4.

Chapter 4

Interior-Point Method (IPM)

The term interior-point method was introduced in the 1980s as new methods were developed for solving large linear optimization problems efficiently, which were characterized for requiring the inequality constraints to be strictly satisfied on all iterates. In the early 1990s a subclass of these methods, the primal-dual methods, appeared as the most efficient practical approaches. IPMs are characterized for presenting computationally expensive iterations while making significant progress toward the solution [52]. A more detailed description of the origins and impact of IPMs can be found in [69]. As mentioned before, IPMs generate iterates that lie strictly inside the region described by the inequality constraints. If the constraints are satisfied throughout all the iterations then the method is considered feasible, whereas infeasible IPMs only guarantee constraints satisfaction at the solution [27].

As explained in chapter 3, an LMPC problem can be formulated as a typical QP problem to be solved at every sampling instant. The two most used approaches to solve a QP problem when inequality constraints are present are Active-Set Method (ASM) and IPM. A description of both methods for solving QP problems can be found in [52, 70]. In general, for a small number of variables and constraints ASM has lower complexity and converges faster than IPM; however, IPM performs better for larger problems [37]. This is due to the mathematical complexity of ASMs growing exponentially with the problem size, as opposed to IPMs which present a polynomial complexity [27]. Additionally, for the case of ASM, the size of the linear system of equations to be solved at each iteration changes depending on which constraints are active (inequality constraints satisfied as equalities, which define the active set on each iteration), which is not the case for IPM where the problem structure is fixed. For embedded FPGA-based implementations, IPMs are therefore a better choice since the algorithm needs to be executed on a fixed architecture.

An NMPC problem, considering the solution using direct methods, has to be formulated as an NLP problem through discretization. The two main solution approaches for NLP problems are Sequential Quadratic Programming (SQP) and IPM for NLP [52]. SQP methods solve a QP problem at each iteration for the step direction search and then a merit function is used for the step length. The QP problem is usually solved using active-set methods; therefore, IPM for NLP was implemented in this thesis as it better suits the FPGA platform as previously explained.

4.1 Primal-Dual IPM for QP

The formulation of primal-dual IPM as presented in [52, 70] is considered in this section. Consider the following QP problem with both equality and inequality constraints (same as (3.5)):

$$\underset{\xi}{\text{minimize}}\, J(k) = \frac{1}{2}\xi^T Q\xi + q^T\xi\,, \tag{4.1a}$$

subject to:

$$A\xi = b\,, \tag{4.1b}$$

$$C\xi \leqslant d\,. \tag{4.1c}$$

The Lagrange function of this QP problem is:

$$L = \frac{1}{2}\xi^T Q\xi + q^T\xi + \lambda(A\xi - b) + v(C\xi - d)\,, \tag{4.2}$$

where $\lambda \in \mathbb{R}^{N_{EC}}$ and $v \in \mathbb{R}^{N_{IC}}$ represent the Lagrange multipliers for the equality and inequality constraints respectively. Assuming Q is positive semidefinite and the constraints are linear, then the Karush-Kuhn-Tucker (KKT) conditions are sufficient conditions for finding a global minimum:

$$Q\xi + q + A^T\lambda + C^T v = 0\,, \tag{4.3a}$$

$$A\xi - b = 0\,, \tag{4.3b}$$

$$C\xi - d + s = 0\,, \tag{4.3c}$$

$$v_i s_i = 0\,, \quad i = 1, ..., N_{IC}\,, \tag{4.3d}$$

$$(v, s) \geq 0\,, \tag{4.3e}$$

where $s \in \mathbb{R}^{N_{IC}}$ represents a vector of slack variables and N_{IC} corresponds to the number of inequality constraints. Newton-like methods are used to compute the solution to Equations (4.3a) to (4.3d), and a line search is then applied to adjust the step length such that equation (4.3e) is satisfied. Applying Newton method to the equality KKT conditions equation (4.4) is obtained, where e is a vector of ones, V and S are diagonal matrices formed from vectors v and s respectively.

$$\begin{bmatrix} Q & A^T & C^T & 0 \\ A & 0 & 0 & 0 \\ C & 0 & 0 & I \\ 0 & 0 & S & V \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\lambda \\ \Delta v \\ \Delta s \end{bmatrix} = \begin{bmatrix} -Q\xi - q - A^T\lambda - C^T v \\ -A\xi + b \\ -C\xi + d - s \\ -VSe \end{bmatrix}, \tag{4.4a}$$

$$(v, s) \geq 0\,. \tag{4.4b}$$

It should be noticed that the restriction (4.4b) makes this a nonlinear system of equations. Additionally, in the system matrix, only the terms on the last row change within iterations. Terms in the vector on the right side are updated on every iteration. The solution of equation (4.4a) gives a search direction for the next step; usually, taking a full step in this direction would violate the condition (4.4b), so a step length along this direction must be calculated. Only small steps can be achieved before violating (4.4b), which makes this approach impractical. For this reason, primal-dual methods use a Newton-like approach by making (4.3d) equal to a small number instead of equal to zero [52], such that:

$$VSe = \sigma\mu e\,, \tag{4.5}$$

where the reduction factor σ, and the complementarity measure μ defined in equation (4.8) are scalar values. Making this change and eliminating Δs from (4.4a) we obtain:

$$\begin{bmatrix} Q & A^T & C^T \\ A & 0 & 0 \\ C & 0 & -V^{-1}S \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\lambda \\ \Delta v \end{bmatrix} = \begin{bmatrix} -rd \\ -rp \\ -rc + s - \sigma\mu V^{-1}e \end{bmatrix}, \tag{4.6a}$$

$$(v, s) > 0\,, \tag{4.6b}$$

$$\Delta s = -s - V^{-1}S\Delta v + \sigma\mu V^{-1}e\,, \tag{4.6c}$$

where rd, rp and rc are defined as:

$$rd = Q\xi + q + A^T\lambda + C^T v\,, \tag{4.7a}$$
$$rp = A\xi - b\,, \tag{4.7b}$$
$$rc = C\xi - d + s\,. \tag{4.7c}$$

$$\mu = \frac{v^T s}{N_{IC}}\,. \tag{4.8}$$

The system can be further simplified by eliminating Δv from (4.6a):

$$\begin{bmatrix} Q + C^T S^{-1} V C & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -rn \\ -rp \end{bmatrix}\,, \tag{4.9a}$$
$$(v, s) > 0\,, \tag{4.9b}$$
$$\Delta v = S^{-1} V \left[C\Delta\xi + rc - s + \sigma\mu V^{-1} e \right]\,, \tag{4.9c}$$
$$\Delta s = -s - V^{-1} S \Delta v + \sigma\mu V^{-1} e\,. \tag{4.9d}$$

With rn defined as:

$$rn = rd + C^T S^{-1} V rc - C^T v + \sigma\mu C^T S^{-1} e\,. \tag{4.10}$$

The main computational task in IPMs is solving the linear system of equations (4.6a) or (4.9a). Notice that the system matrices on both cases are symmetric, indefinite and sparse, which is important for choosing the right solver. A widely used algorithm for solving a linear system of equations when the matrix has these properties is the minimum residual (MINRES) algorithm presented in Algorithm 1 [24]. The MINRES algorithm is a Krylov subspace method for iteratively solving a linear system of equations when the system matrix is symmetric, presenting advantages especially if it is also large and sparse. For the solution of a system of the general form $J\chi = y$, the objective is to iteratively minimize the residual $r = y - J\chi$. The symmetry of the system allows the solution to be performed recursively, being only two previous steps required on each iteration. The iterative nature of this method makes it possible to stop the algorithm at a fixed number of iterations to improve computation time, although a trade-off with precision is to be considered. This method uses only vectors resulting from matrix-vector products with the system matrix J, regarding it as an operator [54], this will be exploited as matrix-vector products are easily parallelized.

The IPM algorithm for the solution of system of equations (4.9) is presented in Algorithm 2 [52].

Algorithm 1: MINRES algorithm for solution of $Ax = b$ [24]

input : A, b, x_0, N_iter

output : x_sol

`/* Initialization:` `*/`

$n \leftarrow 1$, $N \leftarrow length(x_0)$;

$v^{(k)} \leftarrow zeros(N, 1)$, $\hat{v}^{(k)} \leftarrow b - Ax_0$, $\beta^{(k)} \leftarrow ||\hat{v}^{(k)}||$;

$\gamma^{(k)} \leftarrow 1$, $\gamma^{(k-1)} \leftarrow 1$, $\sigma^{(k)} \leftarrow 0$, $\sigma^{(k-1)} \leftarrow 0$;

$\omega^{(k)} \leftarrow zeros(N, 1)$, $\omega^{(k-1)} \leftarrow \omega^{(k)}$;

$\eta \leftarrow \beta^{(k)}$, $x \leftarrow x_0$;

while $n \leq N\_iter$ **do**

Lanczos:

$v^{(k-1)} \leftarrow v^{(k)}$, $v^{(k)} \leftarrow \frac{\hat{v}^{(k)}}{\beta^{(k)}}$;

$\alpha^{(k)} \leftarrow {v^{(k)}}^T A v^{(k)}$;

$\hat{v}^{(k)} \leftarrow Av^{(k)} - \alpha^{(k)} v^{(k)} - \beta^{(k)} v^{(k-1)}$;

$\beta^{(k-1)} \leftarrow \beta^{(k)}$, $\beta^{(k)} \leftarrow ||\hat{v}^{(k)}||$;

QR Factorization:

$\gamma^{(k-2)} \leftarrow \gamma^{(k-1)}$, $\gamma^{(k-1)} \leftarrow \gamma^{(k)}$;

$\sigma^{(k-2)} \leftarrow \sigma^{(k-1)}$, $\sigma^{(k-1)} \leftarrow \sigma^{(k)}$;

$\hat{r}_1 \leftarrow \gamma^{(k-1)}\alpha - \gamma^{(k-2)}\sigma^{(k-1)}\beta^{(k-1)}$;

$r_1 \leftarrow \sqrt{\hat{r}_1^2 + {\beta^{(k)}}^2}$;

$r_2 \leftarrow \sigma^{(k-1)}\alpha + \gamma^{(k-2)}\gamma^{(k-1)}\beta^{(k-1)}$;

$r_3 \leftarrow \sigma^{(k-2)}\beta^{(k-1)}$;

Givens rotation:

$\gamma^{(k)} \leftarrow \frac{\hat{r}_1}{r_1}$;

$\sigma^{(k)} \leftarrow \frac{\beta^{(k)}}{r_1}$;

Update:

$\omega^{(k-2)} \leftarrow \omega^{(k-1)}$, $\omega^{(k-1)} \leftarrow \omega^{(k)}$;

$\omega^{(k)} \leftarrow \frac{v^{(k)} - r_3\omega^{(k-2)} - r_2\omega^{(k-1)}}{r_1}$;

$x \leftarrow x + \gamma^{(k)}\eta\omega^{(k)}$;

$\eta \leftarrow -\sigma^{(k)}\eta$;

$n \leftarrow n + 1$;

end

$x\_sol \leftarrow x$

Algorithm 2: IPM algorithm for QP problem solution [52]

input : Feasible initial vectors $\xi^{(0)}$, $\lambda^{(0)}$, $v^{(0)}$, $s^{(0)}$, such that $(v^{(0)}, s^{(0)}) > 0$

output : Solution ξ

`/* Initialization: */`

$\mu^{(0)} \leftarrow \frac{s^{(0)^T} v^{(0)}}{mc}$, $k \leftarrow 0$;

repeat

choose $\sigma^{(k)} \in (0, 1)$;

solve for $\Delta\xi^{(k)}$ and $\Delta\lambda^{(k)}$:

$$\begin{bmatrix} Q + C^T S^{(k)^{-1}} V^{(k)} C & A^{(k)^T} \\ A^{(k)} & 0 \end{bmatrix} \begin{bmatrix} \Delta\xi^{(k)} \\ \Delta\lambda^{(k)} \end{bmatrix} = \begin{bmatrix} -rn^{(k)} \\ -rp^{(k)} \end{bmatrix};$$

get $\Delta v^{(k)}$:

$$\Delta v^{(k)} \leftarrow S^{(k)^{-1}} V^{(k)} \left[C\Delta\xi^{(k)} + rc^{(k)} - s^{(k)} + \sigma^{(k)} \mu^{(k)} V^{(k)^{-1}} e \right];$$

get $\Delta s^{(k)}$:

$$\Delta s^{(k)} \leftarrow -s^{(k)} - V^{(k)^{-1}} S^{(k)} \Delta v^{(k)} + \sigma^{(k)} \mu^{(k)} V^{(k)^{-1}} e;$$

find $\alpha^{(k)}$ such that $(v^{(k+1)}, s^{(k+1)}) > 0$;

update vectors:

$$(\xi^{(k+1)}, \lambda^{(k+1)}, v^{(k+1)}, s^{(k+1)}) \leftarrow$$
$$(\xi^{(k)}, \lambda^{(k)}, v^{(k)}, s^{(k)}) + \alpha^{(k)} (\Delta\xi^{(k)}, \Delta\lambda^{(k)}, \Delta v^{(k)}, \Delta s^{(k)});$$

update complementarity measure $\mu^{(k)}$;

set $k \leftarrow k + 1$;

until *stopping criteria satisfied*;

4.2 Primal-Dual IPM for NLP

The formulation of IPM considering the continuation approach as presented in [52] is considered in this section. The NLP problem shown in (4.11) is considered, which differs from (3.17) only in the inequality constraints. The form considered here will be useful for the implementation.

The Lagrange function is given in equation (4.12) and the KKT conditions in (4.13), where $\lambda \in \mathbb{R}^{N_{EC}}$ and $v \in \mathbb{R}^{N_{IC}}$ represent the Lagrange multipliers for the equality and inequality constraints respectively, $s \in \mathbb{R}^{N_{IC}}$ represents a vector of slack variables, e is a vector of ones, V and S are diagonal matrices formed from vectors v and s respectively. $A_E(\xi)$ and $A_I(\xi)$ represent the Jacobian matrices of $C_E(\xi)$ and $C_I(\xi)$ respectively. In this case, the KKT conditions are necessary conditions only. For equation (4.13d), the original condition of being equal to zero ($\mu = 0$) is perturbed and this leads to the

inequality (4.13e) being strictly satisfied. The continuation approach is to solve the perturbed system (4.13) for a sequence of positive scalar parameters μ_k, also called barrier parameters, that converges to zero [52].

$$\underset{\xi}{\text{minimize}}\, f(\xi)\,, \tag{4.11a}$$

subject to:

$$C_E(\xi) = 0\,, \tag{4.11b}$$
$$C_I(\xi) \leq d\,. \tag{4.11c}$$

$$L(\xi, \lambda, v) = f(\xi) + \lambda C_E(\xi) + v(C_I(\xi) - d)\,. \tag{4.12}$$

$$\nabla f(\xi) + A_E^T(\xi)\lambda + A_I^T(\xi)v = 0\,, \tag{4.13a}$$
$$C_E(\xi) = 0\,, \tag{4.13b}$$
$$C_I(\xi) - d + s = 0\,, \tag{4.13c}$$
$$VSe = \mu e\,,\ \mu > 0\,, \tag{4.13d}$$
$$(v, s) > 0\,. \tag{4.13e}$$

Newton method is used to compute the solution to Equations (4.13a) to (4.13d), and a line search is then applied to adjust the step length such that equation (4.13e) is satisfied. Applying Newton method to the equality KKT conditions, equation (4.14) is obtained.

$$\begin{bmatrix} \nabla^2_{\xi\xi} L & A_E^T(\xi) & A_I^T(\xi) & 0 \\ A_E(\xi) & 0 & 0 & 0 \\ A_I(\xi) & 0 & 0 & I \\ 0 & 0 & S & V \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\lambda \\ \Delta v \\ \Delta s \end{bmatrix} = \begin{bmatrix} -\nabla f(\xi) - A_E^T(\xi)\lambda - A_I^T(\xi)v \\ -C_E(\xi) \\ -C_I(\xi) + d - s \\ -VSe + \mu e \end{bmatrix}, \tag{4.14a}$$

$$(v, s) > 0\,. \tag{4.14b}$$

Eliminating Δv and Δs system (4.15) is obtained, with rd, rp and rn defined in (4.16). It should be noticed that matrix $A_I^T(\xi)S^{-1}VA_I(\xi)$ will be diagonal if the inequality constraints are in the form of box constraints.

$$\begin{bmatrix} \nabla^2_{\xi\xi} L + A_I^T(\xi) S^{-1} V A_I(\xi) & A_E^T(\xi) \\ A_E(\xi) & 0 \end{bmatrix} \begin{bmatrix} \Delta\xi \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -rn \\ -rp \end{bmatrix}, \tag{4.15a}$$

$$(v, s) > 0\,, \tag{4.15b}$$

$$\Delta v = S^{-1} V \left[A_I(\xi)\Delta\xi + C_I(\xi) - d + \mu V^{-1} e \right], \tag{4.15c}$$

$$\Delta s = -s - V^{-1} S \Delta v + \mu V^{-1} e\,, \tag{4.15d}$$

$$rd = \nabla f(\xi) + A_E^T(\xi)\lambda + A_I^T(\xi) v\,, \tag{4.16a}$$

$$rp = C_E(\xi)\,, \tag{4.16b}$$

$$rn = rd + A_I^T(\xi) S^{-1} V (C_I(\xi) - d) + \mu A_I^T(\xi) S^{-1} e\,. \tag{4.16c}$$

As in the IPM for QP, the main computational task is solving the linear system of equations (4.15a). A MINRES algorithm will also be used for its solution. Once the step direction has been found the step length must be found such that condition (4.15b) is satisfied. A basic IPM algorithm for NLP is presented in algorithm 3[52].

It should be noticed that derivative information of both equality and inequality constraints functions, as well as the Hessian of the Lagrange function, must be available for the solution of the NLP problem, and that this information must be updated on every iteration of the algorithm.

As a consequence of the problem having non-convexities and nonlinearities additional modifications to algorithm 3 are required; for example, the addition of line search and control features for the decrease rate in slack variables and Lagrange multipliers, a detailed description can be found in [52].

4.3 Summary

This chapter presented the primal-dual IPM for the solution of QP and NLP problems. The KKT conditions and the Newton-like solution approach to solve the linear system of equations was detailed for both cases. This linear system will be solved using the iterative MINRES method because of the characteristics of the primal-dual system. Iterative methods additionally allow to trade off accuracy for computation time by reducing the required number of iterations.

Algorithm 3: Basic IPM algorithm for NLP problem solution [52]

input : Feasible initial vectors $\xi^{(0)}$, $\lambda^{(0)}$, $v^{(0)}$, $s^{(0)}$, such that $(v^{(0)}, s^{(0)}) > 0$

output : Solution ξ

`/* Initialization: */`

Select $\mu^{(0)} > 0$, and $\sigma \in (0, 1)$;

$k \leftarrow 0$;

repeat

solve (4.15a) for $\Delta\xi^{(k)}$ and $\Delta\lambda^{(k)}$;

get $\Delta v^{(k)}$ from (4.15c);

get $\Delta s^{(k)}$ from (4.15d);

find $\alpha_{v^{(k)}}$ and $\alpha_{s^{(k)}}$ such that:

$v^{(k)} + \alpha_{v^{(k)}} \Delta v^{(k)} > 0$,

$s^{(k)} + \alpha_{s^{(k)}} \Delta s^{(k)} > 0$;

update vectors:

$\xi^{(k+1)} \leftarrow \xi^{(k)} + \alpha_{s^{(k)}} \Delta\xi^{(k)}$,

$\lambda^{(k+1)} \leftarrow \lambda^{(k)} + \alpha_{v^{(k)}} \Delta\lambda^{(k)}$,

$v^{(k+1)} \leftarrow v^{(k)} + \alpha_{v^{(k)}} \Delta v^{(k)}$,

$s^{(k+1)} \Leftarrow s^{(k)} + \alpha_{s^{(k)}} \Delta s^{(k)}$;

update μ :

$\mu_{k+1} \leftarrow \sigma\mu_k$;

set $k \leftarrow k + 1$;

until *stopping criteria for problem* (4.11) *satisfied*;

Chapter 5

Implementation of Embedded MPC Algorithms on FPGA Using HLS

An MPC problem can be expressed as a QP problem for LMPC and as anNLP problem for NMPC as explained in chapter 3. For both cases the primal-dual IPM was chosen because it better suits FPGA-based implementations when compared with the ASM, as described in chapter 4. The main computational effort in IPM algorithm resides in the solution of the linear system of equations resulting from the KKT conditions; based on the symmetry and indefiniteness of the system matrix the MINRES algorithm was chosen for its solution. The MINRES algorithm implemented was tailored to the corresponding approach (LMPC or NMPC) in order to reduce computation time and allow for implementation on fast dynamics systems.

5.1 Related Work

A lot of research has been made regarding hardware implementations of MPC. Explicit piecewise-linear MPC was implemented on an ASIC in [32], reporting sampling intervals in the order of milliseconds and pointing out the increasing memory requirements as the problem dimension grows. Feasibility of QP solver implementation on FPGA using primal-dual IPM was demonstrated in [41] with an aircraft example; a sequential Handel-C model obtained from MATLAB® code was used, achieving 20 MHz clock frequency. This work was improved in [40] using parallel computation and solving the linear system of equations with Gaussian elimination, achieving 25 MHz operation frequency. A comparison between implementation of Mehrotra's IPM algorithm on a GPU using CUDA® and on FPGA using both single precision floating-point and 32 bits fixed-point is presented in [39], analyzing the trade-offs between these number representation formats. 88.2 MHz and 56.7 MHz were reported for floating-point and

fixed-point implementations respectively, obtaining 11x speed-ups on the FPGA for small size problems when compared with the GPU. A soft processor, i.e. a processor implemented using the FPGA's logic fabric, running a C/C++ model of the MPC algorithm at 150 MHz was used in [15], considering dual IPM for the QP solver.

A mix of software and hardware implementation was considered in [12], achieving 25 MHz clock frequency on a Verilog based design. This work was further improved in [66] using logarithmic number systems arithmetic for the co-processor, running the accelerator at 50 MHz. Another mixed software/hardware implementation was reported in [80], where the intensive floating-point operations were performed in software while remaining operations were accelerated in hardware. Verilog was used as input language for the hardware accelerator running at 100 MHz.

Non-standard numerical representation for MPC was studied in [16] considering a hybrid fixed-point floating-point architecture, and in [6] using only fixed-point representation. Implementation of the Lanczos algorithm used in iterative methods like MINRES was proposed in [29, 30] using fixed-point arithmetic, where a preconditioner was proposed to guarantee overflow errors will not occur. Additionally, [34] showed that the discretization method employed is critical for guaranteeing good numerical behavior with low-precision number representation. In [44], a method to include auxiliary decision variables for the solution of QP problems with low-precision arithmetic was proposed, employing delta-domain formulation as discretization method for numerical stability.

Very fast computation times were achieved in [67] using primal barrier IPM and reduced precision floating-point arithmetic, allowing systems with sampling rates of 200 µs to be controlled. A dual active-set algorithm was used in [68] achieving similar computation time results. Both designs achieved 70 MHz clock frequency. Sparse formulation of LMPC for FPGA implementation using primal-dual IPM and MINRES algorithm was presented in [28], where inherent parallelism and problem structure in IPM was exploited to achieve high throughput and a reduction in memory requirements of up to 75%, achieving clock frequencies above 150 MHz. Custom architectures for first-order optimization methods were proposed in [31] for the solution of constrained MPC problems, achieving good performance for sampling periods beyond 1 MHz, running the FPGA at 400 MHz. The latter two works were further detailed in [27]. A system-on-a-chip MPC system considering also the sparse approach and primal-dual IPM as well as an additional fast gradient QP solver for steady state target calculation was presented in [26]. A soft processor was used to communicate the two custom QP solvers implemented, running the whole system at 250 MHz. Single precision floating-point Cholesky factorization was employed in [43] for the predictor-corrector IPM algorithm

reporting also frequencies up to 250 MHz. VHDL programming was used for the last six works cited.

Implementations of NMPC on FPGAs have also been reported. SQP algorithm was used in [35] to solve the optimization problem and QR factorization was used for the linear solver; however, no performance results were reported. A vector-MPC approach was used in [38] for the control of permanent magnet synchronous motors with interior magnets for electric vehicles, achieving control cycle times in the range of µs. MATLAB® Simulink® and Xilinx® System Generator were used to program the FPGA. Particle swarm optimization (PSO) algorithm on fixed-point arithmetic was used in [79] to deal with the nonlinear optimization problem, because of its naturally parallel capabilities. HLS tools were used for the implementation, achieving 40 MHz clock frequency.

For this thesis, primal-dual IPM algorithms for the solution of QP and NLP problems were implemented on an FPGA using HLS with C++. The remaining operations for the LMPC and NMPC algorithms were implemented on an ARM processor using C descriptions.

5.2 Compressed Storage Formats

As explained in chapter 3, the sparse or simultaneous approach for the formulation of the MPC problem was considered, which leads to sparse matrices. Therefore it is not memory and computationally efficient to store their non-zero values and perform operations on them. Four compressed storage formats were considered: Compressed Row Storage (CRS), Compressed Column Storage (CCS), Compressed Diagonal Storage (CDS) and left-shift storage.

Compressed Row Storage

This compressed storage approach makes no assumptions on the matrix structure. Only the non-zero values are stored in a vector, scanning the original matrix from one row to the next, from left to right. Two more vectors are required, one for the column indexes of each non-zero value and one for specifying the number of non-zero values up to each row, starting with zero. This way of storing the matrix is not very efficient because of a required indirect access to the vectors for operations like matrix-vector multiplication [5]. An example of this storage format:

$$M = \begin{bmatrix} 1 & 5 & -2 & 0 & 0 & 0 & 1 \\ -8 & 0 & 0 & 0 & 4 & -2 & 0 \\ 0 & 0 & 9 & -1 & 0 & 0 & 2 \end{bmatrix},$$

CRS format:

$$\begin{aligned} val &= \begin{bmatrix} 1 & 5 & -2 & 1 & -8 & 4 & -2 & 9 & -1 & 2 \end{bmatrix}, \\ col\_ind &= \begin{bmatrix} 0 & 1 & 2 & 6 & 0 & 4 & 5 & 2 & 3 & 6 \end{bmatrix}, \\ row\_ptr &= \begin{bmatrix} 0 & 4 & 7 & 10 \end{bmatrix}. \end{aligned}$$

Compressed Column Storage

This compressed storage approach makes no assumptions on the matrix structure. Only the non-zero values are stored in a vector, scanning the original matrix from one column to the next, from top to bottom starting at the column on the left. Two more vectors are required, one for the row indexes of each non-zero value and one for specifying the number of non-zero values up to each column, starting with zero. This format equals the CRS representation of the transpose of the considered matrix [5]. An example of this storage format:

$$M = \begin{bmatrix} 1 & 5 & -2 & 0 & 0 & 0 & 1 \\ -8 & 0 & 0 & 0 & 4 & -2 & 0 \\ 0 & 0 & 9 & -1 & 0 & 0 & 2 \end{bmatrix},$$

CCS format:

$$\begin{aligned} val &= \begin{bmatrix} 1 & -8 & 5 & -2 & 9 & -1 & 4 & -2 & 1 & 2 \end{bmatrix}, \\ row\_ind &= \begin{bmatrix} 0 & 1 & 0 & 0 & 2 & 2 & 1 & 1 & 0 & 2 \end{bmatrix}, \\ col\_ptr &= \begin{bmatrix} 0 & 2 & 3 & 5 & 6 & 7 & 8 & 10 \end{bmatrix}. \end{aligned}$$

Compressed Diagonal Storage

This format is appropriate for banded matrices. It represents the original matrix with one smaller matrix and one vector. The matrix stores the sub-diagonals, considering

leading zeros for those corresponding to the lower triangular part and trailing zeros for the upper triangular part. This means some unnecessary zeros are stored, however, this format allows for a more efficient matrix-vector multiplication than the CRS format. The vector stores the columns indexes of the sub-diagonals, considering the main diagonal as zero-indexed [5][13]. For symmetric matrices, only the upper (or lower) triangular part may be stored. An example of this storage format:

$$M = \begin{bmatrix} 1 & 5 & -2 & 0 & 0 & 0 & 0 \\ 5 & 7 & 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & -2 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & -7 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 8 & -1 \\ 0 & 0 & 0 & 0 & 3 & -1 & 9 \end{bmatrix},$$

CDS format:

$$val = \begin{bmatrix} 1 & 5 & -2 \\ 7 & 1 & 0 \\ -2 & 0 & 4 \\ 1 & 0 & 0 \\ -7 & 0 & 3 \\ 8 & -1 & 0 \\ 9 & 0 & 0 \end{bmatrix},$$

$$col\_ind = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix}.$$

Left-Shift Storage

Having matrix-vector multiplication as target operation to be performed on a block structured banded matrix like $\hat{A}_{ip}$ (refer to (5.8) on page 41), as will be the case for the implemented MINRES algorithm, a better approach is to store the blocks on the band of the matrix in a left-shifted manner. In this case, non-useful zeros are stored but has the advantage that all elements in each row remain in the same row and in the same order in the new reduced matrix. Additionally, a vector is required to specify the offset for each row, that is the number of zeros on the left of the band suppressed during the left-shift operation.

5.3 Hardware and Software

The algorithms were implemented on a Zynq® -7000 AP SoC ZC706 Evaluation Kit, developed by Xilinx®, which contains an XC7Z045 FFG900 -2 AP SoC with characteristics as shown in table 5.1, for more detailed information refer to [78]. Zynq® devices possess two main sections: the Processing System (PS) and the Programmable Logic (PL), which can be used independently or together. The PS contains a dual-core ARM® Cortex®-A9 processor and associated resources that form an Application Processing Unit (APU), together with peripheral and memory interfaces, interconnects and clock generation circuitry. Figure 5.1 shows a diagram of the basic structure of the PS. The PL is basically a general purpose FPGA (refer to section 2.2). Communication between PS and PL can be done with the use of the Advanced eXtensible Interface (AXI) standard through AXI interconnects and interfaces or with Extended Multiplexed Input/Output (EMIO) interfaces [21].

The RTL design for the PL was obtained using Vivado® HLS [76] and then exported to a design in the Vivado® IDE [77] for its integration on a Zynq®-based design. Synthesis, implementation, and bitstream generation were also run in the Vivado® IDE and then exported to Xilinx® SDK environment as a hardware platform for the development of a bare metal application for the PS. Both PS and PL were programmed from Xilinx® SDK.

Table 5.1 – XC7Z045 FFG900 -2 AP SoC characteristics [78], upper characteristics correspond to the PS and lower to the PL

Characteristic	Description
Device Name	Z-7045
Part Number	XC7Z045
Processor Core	Dual-Core ARM® Cortex®-A9 MP-Core Up to 1GHz
On-Chip Memory	256KB
Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO
PS to PL Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave, 4x AXI 64b/32b Memory, AXI 64b ACP , 16 Interrupts
7 Series PL Equivalent	Kintex®-7
Logic Cells	350K
Look-Up Tables (LUTs)	218,600
Flip-Flops	437,200
Total Block RAM (36Kb)	19.1Mb

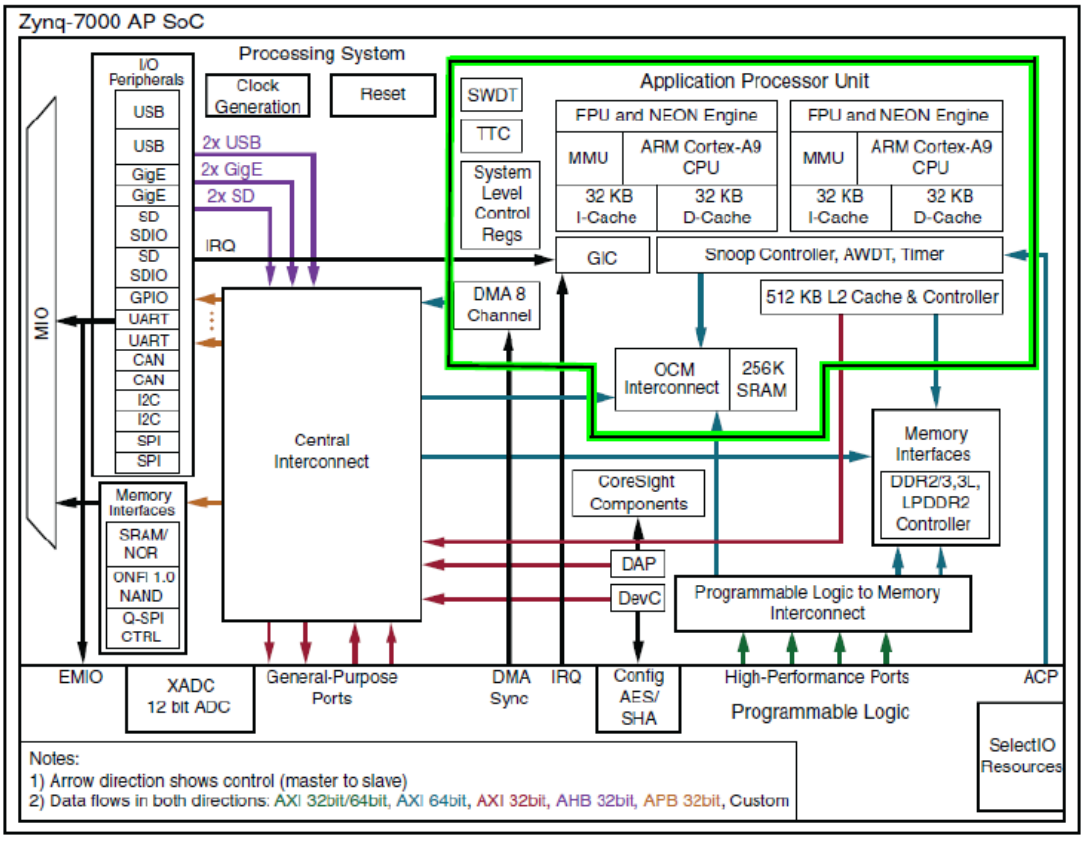


Figure 5.1 – Block diagram for Zynq® PS [21]

5.4 LMPC Algorithm Implementation

5.4.1 Block Structure and Sparsity

Matrices Q, A and C in equation (3.5) are sparse and have a well defined structure when using the implicit sparse approach as described in section 3.2.3.2. This can be easily exploited as proposed in [28] and [70]. Using the primal-dual IPM for solving this problem equation 4.9 is obtained, a MINRES algorithm was implemented to solve system (4.9a) iteratively. Regarding this linear system of equations, also called primal-dual system, let's denote the system matrix as A_{ip}.

$$A_{ip} = \begin{bmatrix} Q + C^T S^{-1} V C & A^T \\ A & 0 \end{bmatrix} = \begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix}. \tag{5.4}$$

Within an LMPC iteration, only the term $C^T S^{-1} V C$ changes from iteration to iteration of the MINRES algorithm. As matrices S and V are diagonal matrices formed from vectors s (slack variables) and v (Lagrange multipliers for inequality constraints)

respectively, this term has diagonal components only. This leads to the important fact that only the diagonal components of A_{ip} need to be updated when solving (4.9a). For the case of a time-invariant system, this remains so for all iterations of the LMPC problem. If the system is time-varying, then matrix A needs to be updated for the next LMPC iteration. The constant part of matrix A_{ip} is denoted as $\hat{A}$, and has the structure shown in (5.5).

$$\hat{A} = \begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix},$$

$$\hat{A} = \begin{bmatrix}
R_0 & & & & & & & -B_0^T & & & \\
& Q_1 & & & & & & I & -A_1^T & & \\
& & \ddots & & & & & & -B_1^T & & \\
& & & \ddots & & & & & I & \ddots & -A_{N-1}^T \\
& & & & R_{N-1} & & & & & & -B_{N-1}^T \\
& & & & & P_N & & & & & I \\
-B_0 & I & & & & & & 0 & \dots & \dots & 0 \\
& -A_1 & -B_1 & I & & & & \vdots & \ddots & & \vdots \\
& & & \ddots & & & & \vdots & & \ddots & \vdots \\
& & & -A_{N-1} & -B_{N-1} & I & & 0 & \dots & \dots & 0
\end{bmatrix}. \tag{5.5}$$

Matrix $\hat{A}$ is symmetric and sparse, but not banded. Through row and column reordering it was made banded to ease calculations and reduce memory requirements [27, 28, 58]. After reordering, the system of linear equations is the following:

$$\hat{A}_{ip}\Delta_{\xi,\lambda} = \hat{b}_{ip}\,, \tag{5.6}$$

where matrix $\hat{A}_{ip}$, which is banded and remains symmetric, was obtained by reordering the vector of optimization variables and Lagrange multipliers for the equality constraints from the system (4.9a), such that vector $\Delta_{\xi,\lambda}$ and matrix $\hat{A}_{ip}$ have the structure presented in equation (5.7) and (5.8) respectively.

As can be seen from (5.8) the elements in the band maintain the order along the prediction horizon. The terms in $C^T S^{-1} V C$ must still be added to $\hat{A}_{ip}$ for the MINRES algorithm, and have the same distribution as those in matrix Q, along $\hat{A}_{ip}$ diagonal. This means that the terms R_i and Q_i in $\hat{A}_{ip}$ are replaced by the corresponding terms of H. Vector $\hat{b}_{ip}$ was constructed from vectors rn and rp as shown in equation (5.9),

where the notation $rn(x_i)$ refers to the terms in vector rn that correspond to vector x on the prediction horizon interval defined by subindex i.

$$\Delta_{\xi,\lambda} = \begin{bmatrix} \Delta u_0 \\ \Delta\lambda_0 \\ \Delta x_1 \\ \Delta u_1 \\ \Delta\lambda_1 \\ \Delta x_2 \\ \vdots \\ \Delta u_{N-1} \\ \Delta\lambda_{N-1} \\ \Delta x_N \end{bmatrix}, \tag{5.7}$$

$$\hat{A}_{ip} = \begin{bmatrix} R_0 & -B_0^T & & & & & & & & \\ -B_0 & 0 & I & & & & & & & \\ & I & Q_1 & 0 & -A_1^T & & & & & \\ & & 0 & R_1 & -B_1^T & & & & & \\ & & -A_1 & -B_1 & 0 & I & & & & \\ & & & & & \ddots & & & & \\ & & & & & I & Q_{N-1} & 0 & -A_{N-1}^T & \\ & & & & & & 0 & R_{N-1} & -B_{N-1}^T & \\ & & & & & & -A_{N-1} & -B_{N-1} & 0 & I \\ & & & & & & & & I & P_N \end{bmatrix}, \tag{5.8}$$

$$b_{ip} = \begin{bmatrix} -rn(u_0) \\ -rn(x_1) \\ -rn(u_1) \\ -rn(x_2) \\ \vdots \\ -rn(u_{N-1}) \\ -rn(x_N) \\ -rp(\lambda_0) \\ -rp(\lambda_1) \\ \vdots \\ -rp(\lambda_{N-1}) \end{bmatrix} \quad , reordered: \quad \hat{b}_{ip} = \begin{bmatrix} -rn(u_0) \\ -rp(\lambda_0) \\ -rn(x_1) \\ -rn(u_1) \\ -rp(\lambda_1) \\ -rn(x_2) \\ \vdots \\ -rn(u_{N-1}) \\ -rp(\lambda_{N-1}) \\ -rn(x_N) \end{bmatrix}. \tag{5.9}$$

As mentioned in section 5.2, the left-shift format is more appropriate for matrix-vector operations on banded matrices. Matrix $\hat{A}_{ip}$ is shown in this storage format in equation (5.10), where n is the system order, m is the number of inputs and NR is the number of rows of $\hat{A}_{ip}$, which equals $N(2n+m)$. The number of columns of the reduced matrix $ValA_{iph}$ is $3n+m$.

$$ValA_{iph} = \begin{bmatrix} R_0 & -B_0^T & & \\ -B_0 & 0 & I & \\ I & Q_1 & 0 & -A_1^T \\ R_1 & -B_1^T & & \\ -A_1 & -B_1 & 0 & I \\ & \vdots & & \\ I & Q_{N-1} & 0 & -A_{N-1}^T \\ R_{N-1} & -B_{N-1}^T & & \\ -A_{N-1} & -B_{N-1} & 0 & I \\ I & P_N & & \end{bmatrix}, col\_indA_{iph} = \begin{bmatrix} 0 \\ 0 \\ m \\ 2n+m \\ n+m \\ \vdots \\ NR-(4n+m) \\ NR-(2n+m) \\ NR-(3n+m) \\ NR-2n \end{bmatrix}. \tag{5.10}$$

5.4.2 LMPC Formulation

The implemented LMPC algorithm solves the QP problem (3.5) that results from formulating problem (3.3) using the implicit sparse approach, i.e. considering both the system states and control inputs over the corresponding prediction horizon as optimization variables, as presented in section 3.2.3.2. This approach results in sparse matrices and vectors of a well defined structure of which advantage was taken to reduce memory requirements and computation time.

Because of the wide dynamic range required for the solution of the IPM algorithm single precision floating-point representation (32 bits) was preferred for the implementation [34][30]. Despite modern FPGAs present double precision support, the use of single precision arithmetic allows a reduction in resources utilization and therefore lower power consumption. Although single precision was preferred for the reasons mentioned above, for some study cases it was necessary to use double precision as the dynamic range precluded the use of the former. Algorithm 4 presents a general LMPC formulation.

Algorithm 4: LMPC algorithm

input : Prediction horizon N, sampling period T_s, plant model, constraints

foreach *Sampling instant* **do**

Measure or estimate plant states at current sampling instant;

Formulate QP problem;

Solve QP problem to get optimal solution ξ^* for the optimization variables;

Apply first optimal input(s) u^* to the plant, corresponding to the first interval of the prediction horizon;

end

For this work, trajectory tracking study cases were considered and therefore the implementation was tailored to this kind of application. The considered models were nonlinear, so they were linearized in order to apply LMPC. As a consequence, putting problem (3.3) in the form of (3.5) resulted in the matrices and vectors of the QP problem having the following characteristics:

- Matrix Q: This matrix was considered to be constant, that is the differences between the variables considered in the cost function and the desired references were equally penalized on each sampling period over the corresponding prediction horizon (does not mean that within the prediction horizon the weight matrices were the same for every time interval). This matrix may be block diagonal, for example in case linear combinations of the states or inputs are penalized in the cost function; therefore, to allow the implemented design to take this case into consideration, this matrix was stored in CDS format using on-chip memory. Diagonal matrices were considered for the case studies.

- Matrix A: Is a constant matrix for the special case of a time-invariant system; however, for a model linearized along different operation points, it does not remain constant for different sampling instants. Due to its well defined block structure, all operations with this matrix were performed only storing the corresponding system matrices $A(k+i|k)$ and $B(k+i|k)$ (abbreviated as A_i, B_i) from the state space representation along the prediction horizon.

- Matrix C: This matrix, which only contains 1 and -1 as non-zero values, was considered to be constant, i.e. the constrained variables remain the same for every instant. This allowed to desist from storing this matrix as all operations were translated into code. For implementation it was assumed that all control inputs

are constrained, and a vector (C_p) indicating which states are constrained must be specified before synthesis.

- vector q: This vector depends on the reference trajectory, is zero in case all states and inputs have desired value zero, as was the case for the LMPC case studies. Was stored using on-chip memory.

- vector b: Updates at every sampling instant with a new states measurement/estimation (x_0).

- vector d: Changes only if the constraints change or if the operation point for a linearized model changes.

5.4.3 Primal-dual IPM implementation

Primal-dual IPM was used for the solution of problem (3.5) as presented in section 4.1, considering the solution of the nonlinear system of equations (4.9). A function named QP_Solver was created for the solution of the QP problem. As described previously, constant vectors and parameters were stored using on-chip memories. Instead of the sparse matrix A, information about the system model was given through input matrices A_{dN} and B_{dN}, which contain the matrices A_i and B_i for the whole prediction horizon, $i = 0, 1, 2..., N$ as presented in equation (5.11); it was considered that these matrices would change within the prediction horizon because of a nonlinear model being linearized. Information about constraints was given through vector d. The current measured or estimated state $x(k|k)$ abbreviated as x_0 was also input to the solver. Figure 5.2 shows a block diagram of function QP_Solver. The inputs and outputs shown were chosen to be streamed in and out using AXI4-Stream [71] protocol for communication with other IP blocks, for this purpose a top function was created with input and output streams as arguments such that they are automatically implemented as data ports for the RTL design during synthesis by Vivado® HLS.

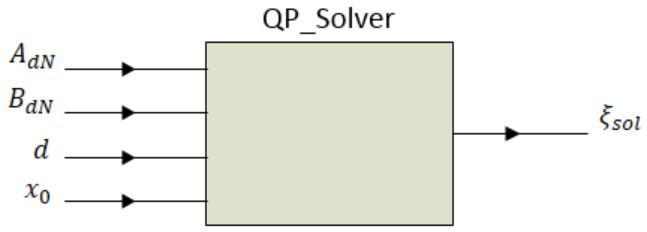


Figure 5.2 – Block diagram for QP_Solver function

$$A_{dN} = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_N \end{bmatrix} \quad , \quad B_{dN} = \begin{bmatrix} B_0 \\ B_1 \\ \vdots \\ B_N \end{bmatrix} . \tag{5.11}$$

Next, the implemented function QP_Solver is explained in detail:

- The system order, number of inputs, prediction horizon and number of iterations for the IPM and MINRES algorithms are specified for each particular case study before synthesis.
- Matrix Q (in CDS format as matrix Q_{cds} and vector col_indQ) and vectors q, C_p and col_indA_{iph} (refer to (5.10)) are stored using ROMs during the implementation of the synthesized design. Vector C_p indicates which states are constrained with a non-zero value (1) in the corresponding position of the states vector.
- Matrix $ValA_{iph}$ (from $\hat{A}_{ip}$ in left-shift format) is obtained from matrices A_{dN}, B_{dN} and Q, see equation (5.10).
- Even though matrix A can be formed from matrices A_{dN} and B_{dN} (see equation (3.11)), it is not necessary as operations on this matrix can be performed without reconstructing it, taking only its structure into consideration.
- Vector b is calculated from the initial state condition x_0, see equation (3.12).
- Near to optimal solution is found with IPM, as explained below.

Once the steps explained above are completed, the nearly optimal solution is found with the primal-dual IPM. For this purpose, a function IP_algorithm was implemented, figure 5.3 shows the block diagram of the function inputs and outputs.

Next, the implemented function IP_algorithm is explained in detail:

- Vectors ξ (feasible), λ, v and s are initialized, the latter two element-wise positive.
- The reduction factor σ and constant β (for the step length computation) are specified in the interval $(0, 1)$.
- The complementarity measure μ is computed (refer to equation (4.8)).
- Main loop of IPM algorithm. A fixed number of iterations was chosen as stopping criteria for the IPM algorithm, since the real-time nature of the implementation

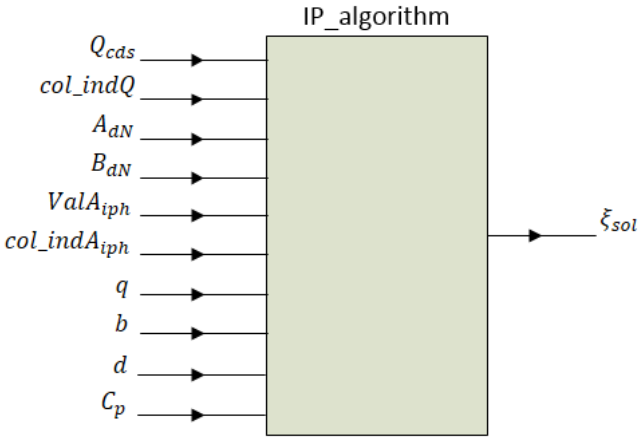


Figure 5.3 – Block diagram for IP_algorithm function

favors a deterministic timing; it also allows to trade off accuracy for computation time [62] [27]. This approach does not guarantee that an optimal solution will be found; in order to obtain a good behavior, the minimum number of iterations required was determined with off-line simulations. Next, the operations in this step are detailed.

- The first step is to compute vectors rd, rp and rc (refer to equation (4.7)). For computation of rd, three matrix-vector multiplications are required:
  - $Q\xi$ is performed with a function that implements an algorithm for multiplication of a symmetric matrix in CDS format with a vector as shown in algorithm 5.
  - $A^T\lambda$ requires a function that performs the operation on matrix A using matrices A_{dN} and B_{dN} as shown in algorithm 7.
  - $C^T v$ uses the vector C_p instead of matrix C as shown in algorithm 6
- For computation of rp, matrix A has to be multiplied with the optimization variables vector ξ and then vector b has to be subtracted. A function was implemented for this purpose as shown in algorithm 8 in page 52.
- For computation of rc, matrix C has to be multiplied with the optimization variables vector ξ and then vectors d and s have to be subtracted and added respectively. The function shown in algorithm 9 in page 53 was implemented for this purpose.
- For the computation of rn (refer to equation (4.10)), matrices V and S are not computed; instead, only vectors v and s are employed. For the term $C^T S^{-1} e$, algorithm 6 in page 50 is used with a vector formed by the inverse

of each term of s as input. For the term $C^T S^{-1} V$ a similar algorithm is used, but considering that instead of a matrix-vector multiplication it is a matrix-matrix multiplication, where the matrix on the right is diagonal, with the diagonal formed by the terms $v[i]/s[i]$.

- Only the terms in the diagonal of Matrix H (refer to equation (5.4)) vary from iteration to iteration of the IPM, and therefore only these terms are calculated and stored in a vector H_{diag}. Algorithm 11 shown on page 54 was used for the implementation.
- Terms in the diagonal of $\hat{A}_{ip}$ are updated with the calculated terms of H_{diag}. This is done taking the compressed structure in equation (5.10) under consideration. H_{diag} will replace the corresponding terms of Q.
- Vector $\hat{b}_{ip}$ is formed from vectors rn and rp (refer to equation (5.9)).
- As explained before, at each IPM iteration a linear system of equations is solved using MINRES algorithm, which is sensitive to the condition number of the system matrix. As the solution is approached this matrix becomes ill-conditioned and inaccuracies appear, leading to wrong solutions even when increasing the number of MINRES iterations [26, 27]. The complementary nature of S and V in $\hat{A}_{ip}$ results in very large and very small values in H, which may additionally exceed single precision [59]. In this context, preconditioning is required in order to accelerate convergence and reduce the error on the solution. Linearization of the systems regarded on the case studies along the given reference trajectory leads to a time-varying model, therefore only on-line preconditioning was implemented. The approach considered in [34] and [26] was adopted, where a diagonal precondition matrix M is used such that:

$$M_{ii} = \frac{1}{\sqrt{\sum_{j=1}^{Z} |ValA_{iph_{ij}}|}} \quad , for \quad i = 0, 1, \ldots, N(2n+m) - 1 \,, \tag{5.12}$$

where Z is the number of columns in $ValA_{iph}$ and equals $3n + m$. As M is a diagonal matrix, only the diagonal terms are stored in vector M_{diag}. This vector is used to form the new preconditioned system of linear equations:

$$\tilde{A}_{ip} y = \tilde{b}_{ip} \,, \tag{5.13}$$

where $\tilde{A}_{ip}$ and $\tilde{b}_{ip}$ are defined as:

$$\tilde{A}_{ip} = M\hat{A}_{ip}M\,, \tag{5.14a}$$

$$\tilde{b}_{ip} = M\hat{b}_{ip}\,. \tag{5.14b}$$

For the case of $\tilde{b}_{ip}$, it is computed by multiplying vectors M_{diag} and $\hat{b}_{ip}$ in an element-wise manner. Algorithm 10, shown on page 53 , was implemented considering the left-shift compressed format for computation of $\tilde{A}_{ip}$.

- Preconditioned system (5.13) is solved using MINRES algorithm 1 (refer to page 27). The implementation will be explained later in this chapter.
- Once the linear system is solved, the solution for the system before preconditioning is obtained multiplying y with M_{diag} element-wise, such that:

$$\Delta_{\xi\lambda} = My\,. \tag{5.15}$$

- Vectors $\Delta\xi$ and $\Delta\lambda$ are obtained from $\Delta_{\xi\lambda}$ (refer to equation (5.7)).
- Vectors Δv and Δs are computed (refer to equations (4.9c) and (4.9d)). For the computation of these vectors, the same considerations are taken on matrices C, V and S as explained in previous steps.
- The step size or step length α is found following the criteria presented in [52] and [70]. The step length is chosen such that the dual feasibility constraint (4.9b) from the KKT conditions is satisfied, this allows the evaluation to be reduced to the terms in Δv and Δs that are negative as shown in equation (5.16), where β is a constant to guarantee the constraint is strictly satisfied and ς represents elements in vectors v and s.

$$\alpha = min(\frac{-\beta\varsigma}{\Delta\varsigma}, 1)\,, \text{ for } \Delta\varsigma \text{ negative}\,. \tag{5.16}$$

- Vectors ξ, λ, v and s are updated, each in its respective search direction computed previously, considering the same step length for all vectors.
- Finally, the complementarity measure μ is updated (refer to equation (4.8)).

Figure 5.4 shows a scheme of the implemented primal-dual IPM algorithm.

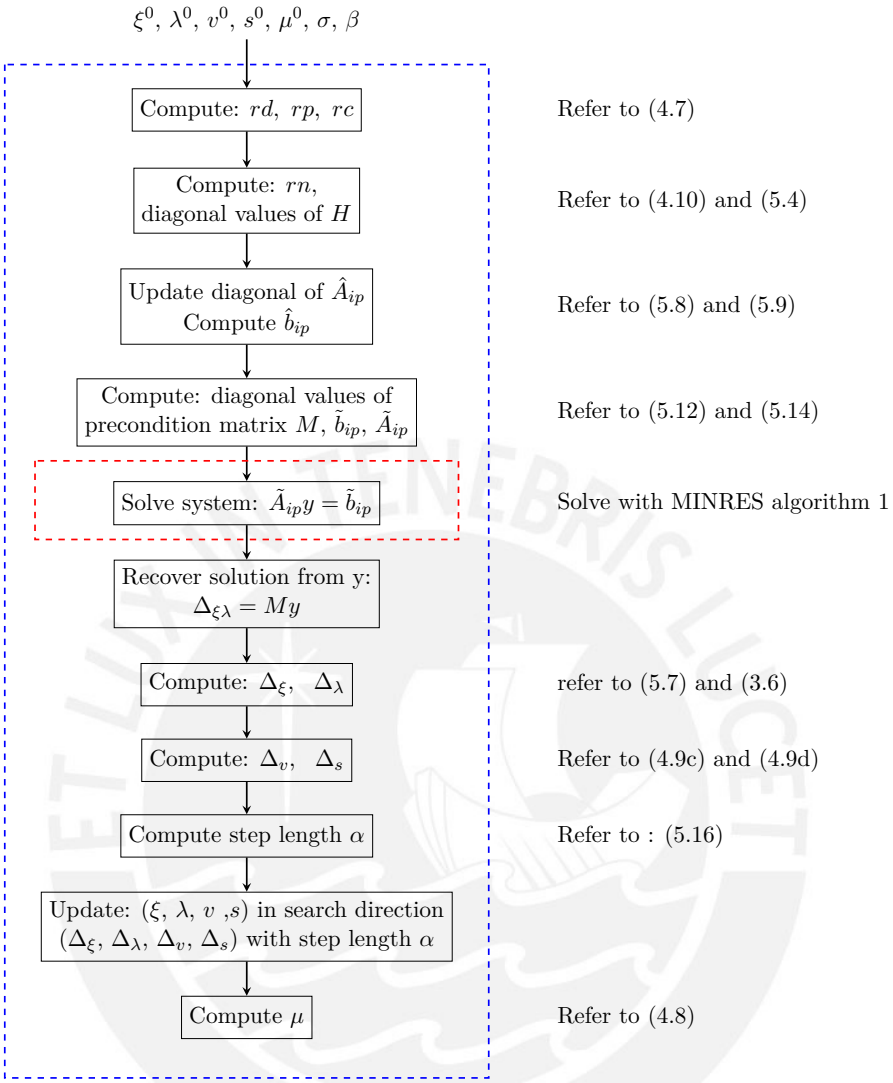


Figure 5.4 – Scheme for the implementation of IPM for solution of a QP problem.

Algorithm 5: Symmetric CDSmatrix-vector multiplication

input : Val, *col_ind*, *vect*
output : *out*
for *i=0* **to** *sizeof(vect)-1* **do**
  $aux \leftarrow 0$;
  $i_{row} \leftarrow i$;
  $i_{col} \leftarrow 0$;
  for *j=0* **to** *sizeof(col_ind)-1* **do**
    if *i+col_ind[j]* $<$ *sizeof(vect)* **then**
      $aux \leftarrow aux + Val[i][j]vect[i + col\_ind[j]]$;
    end
    if $i_{row} > 0$ **and** $i_{row} >$ *i - (sizeof(col_ind)-1)* **then**
      $aux \leftarrow aux + Val[i_{row} - 1][i_{col} + 1]vect[i_{row} - 1]$;
      $i_{row} \leftarrow i_{row} - 1$;
      $i_{col} \leftarrow i_{col} + 1$;
    end
  endfor
  $out[i] \leftarrow aux$;
endfor

Algorithm 6: Transpose of C matrix-vector multiplication

input : C_p, *vect*
output : *out*
/* N, n and m are prediction horizon, system order and number of inputs respectively */
for *i=0* **to** *N-1* **do**
  for *j=0* **to** *m-1* **do**
    $out[i(n + m) + j] \leftarrow vect[2i(n + m) + j] - vect[2i(n + m) + m + j]$;
  endfor
  for *j=0* **to** *n-1* **do**
    $out[i(n + m) + m + j] \leftarrow$
     $C_p[j](vect[2i(n + m) + 2m + j] - vect[2i(n + m) + 2m + n + j])$;
  endfor
endfor

Algorithm 7: Transpose of A matrix-vector multiplication

input : A_{dN}, B_{dN}, $vect$
output : out
/* N, n, m, N_{OV} and N_{EC} are prediction horizon, system order, number of inputs, number of optimization variables and number of equality constraints respectively */
/* Initial block: */
for i=0 **to** m-1 **do**
  $sum \leftarrow 0$;
  for j=0 **to** n-1 **do**
    $sum \leftarrow sum - B_{dN}[j][i]vect[j]$;
  endfor
  $out[i] \leftarrow sum$;
endfor
/* Repetitive block: */
for ite=0 **to** N-2 **do**
  for i=0 **to** n-1 **do**
    $sum \leftarrow 0$;
    for j=0 **to** n-1 **do**
      $sum \leftarrow sum - A_{dN}[(ite+1)n+j][i]vect[(ite+1)n+j]$;
    endfor
    $out[m+ite(m+n)+i] \leftarrow vect[ite(n)+i] + sum$;
  endfor
  for i=0 **to** m-1 **do**
    $sum \leftarrow 0$;
    for j=0 **to** n-1 **do**
      $sum \leftarrow sum - B_{dN}[(ite+1)n+j][i] * vect[(ite+1)n+j]$;
    endfor
    $out[(ite+1)(m+n)+i] \leftarrow sum$;
  endfor
endfor
/* Final block: */
for i=0 **to** n-1 **do**
  $out[N_{OV}-n+i] \leftarrow vect[N_{EC}-n+i]$;
endfor

Algorithm 8: $A\xi - b$ implementation

input : A_{dN}, B_{dN}, ξ, b
output : rp
/* N, n, m are prediction horizon, system order and number of inputs respectively */
/* Initial block: */
for *i=0* **to** *n-1* **do**
  $sum \leftarrow 0$;
  for *j=0* **to** *m-1* **do**
    $sum \leftarrow sum - B_{dN}[j][i]\xi[j]$;
  endfor
  $rp[i] \leftarrow sum + \xi[i+m] - b[i]$;
endfor
/* Repetitive block: */
for *ite=0* **to** *N-2* **do**
  for *i=0* **to** *n-1* **do**
    $sum \leftarrow 0$;
    for *j=0* **to** *n-1* **do**
      $sum \leftarrow sum - A_{dN}[(ite+1)n+i][j]\xi[m+ite(m+n)+j]$;
    endfor
    for *j=0* **to** *m-1* **do**
      $sum \leftarrow sum - B_{dN}[(ite+1)n+i][j]\xi[(ite+1)(m+n)+j]$;
    endfor
    $rp[(ite+1)n+i] \leftarrow x[(ite+1)(m+n)+m+i] + sum - b[(ite+1)n+i]$;
  endfor
endfor

Algorithm 9: Computation of vector rc

input : C_p, ξ, d, s
output : rc
/* N, n and m are prediction horizon, system order and number of inputs respectively */
for *i=0* **to** *N-1* **do**
  for *j=0* **to** *m-1* **do**
    $rc[2i(m+n)+j] \leftarrow \xi[i(m+n)+j] - d[2i(m+n)+j] + s[2i(m+n)+j];$
    $rc[2i(m+n)+j+m] \leftarrow s[2i(m+n)+j+m] - \xi[i(m+n)+j] - d[2i(m+n)+j+m];$
  endfor
  for *j=0* **to** *n-1* **do**
    $rc[2i(m+n)+j+2m] \leftarrow$
    $Cp[j](\xi[i(m+n)+j+m]) - d[2i(m+n)+j+2m] + s[2i(m+n)+j+2m];$
    $rc[2i(m+n)+j+2m+n] \leftarrow$
    $s[2i(m+n)+j+2m+n] - Cp[j](\xi[i(m+n)+j+m]) - d[2i(m+n)+j+2m+n];$
  endfor
endfor

Algorithm 10: Computation of vector $\tilde{A}_{ip}$

input : $ValA_{iph}$, col_indA_{iph}, M_{diag}
output : $ValA_{tilde}$
/* n, *rows* and *cols* are the system order and number of rows and columns in $ValA_{iph}$ respectively */
for *i=0* **to** *rows-1* **do**
  for *j=0* **to** *cols-1* **do**
    if $i <$ *rows-n* **or** $j < 2n$ **then**
      $ValA_{tilde}[i][j] \leftarrow ValA_{iph}[i][j] M_{diag}[i] M_{diag}[j + col\_indA_{iph}[i]];$
    end
  endfor
endfor

Algorithm 11: Computation of vector H_{diag}

input : C_p, Q_{cds}, v, s
output : H_{diag}
/* n, m and N_{OV} are system order, number of inputs and number of optimization variables respectively */
for *i=0* **to** N_{OV}*-1* **do**
  $j \leftarrow modulo(i, n+m)$;
  $k \leftarrow j - m$;
  if $j < m$ **then**
    $H_{diag}[i] \leftarrow Q_{cds}[i][0] + v[2i-j]/s[2i-j] + v[2i-j+m]/s[2i-j+m]$;
  else if *Cp[j-m] != 0* **then**
    $H_{diag}[i] \leftarrow Q_{cds}[i][0] + v[2i-k]/s[2i-k] + v[2i-k+n]/s[2i-k+n]$;
  else
    $H_{diag}[i] \leftarrow Q_{cds}[i][0]$;
  end
endfor

5.4.4 MINRES Algorithm Implementation for QP

As explained in chapter 4 the most computationally demanding part of the IPM algorithm is the solution of a linear system of equations, which was chosen to be solved using MINRES method. For this reason, special care was taken for the implementation of the MINRES algorithm for the solution of the preconditioned system (5.13).

Figure (5.5) shows a block diagram of the inputs and outputs of the implemented algorithm. Next, an explanation of the most important facts of algorithm 1 implementation is given:

- The termination criteria was set to a fixed number of iterations for the same reason as in the IPM. The fact that this is a Krylov subspace method and that preconditioning was considered suggests a similarity with the conjugate gradient method may apply, where the required number of iterations is fixed for a well conditioned problem. This was confirmed through trial and error, such that the number of iterations chosen equals the order of the system matrix $\tilde{A}_{ip}$.
- The main operation in the algorithm is matrix-vector multiplication regarding the system matrix. As this matrix is in left-shift format a function was implemented for this operation as shown in algorithm 12, where matrix Val and vector col_ind

represent the system matrix. For this algorithm to work properly, the input vector has to be padded with m zeros at the end. This allows the operation to be performed without the need of a conditional statement, improving latency.

- On each iteration, the previous solution vector is used as input for the solver, as initial guess, to improve convergence.

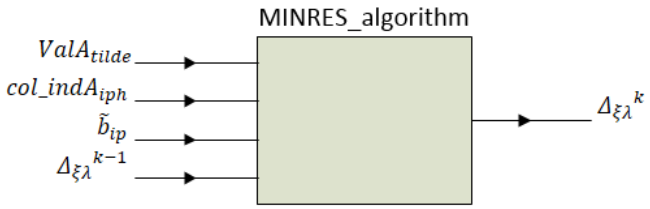


Figure 5.5 – Block diagram for MINRES_algorithm function

Algorithm 12: Matrix-vector multiplication for matrix in left-shift format

input : Val, col_ind, $vect$
output : out
/* *rows* and *cols* are the number of rows and columns in Val respectively */
for *i=0* **to** *rows-1* **do**
  $sum \leftarrow 0$;
  for *j=0* **to** *cols-1* **do**
    $sum \leftarrow sum + Val[i][j]vect[col\_ind[i] + j]$;
  endfor
  $out[i] \leftarrow sum$;
endfor

5.5 NMPC Algorithm Implementation

5.5.1 Block Structure and Sparsity

The NLP problem (4.11) was considered for the implementation. The same cost function in the form (4.1a) as for the linear case was used, which eases the computations as the gradient and Hessian are $Q\xi + q$ and Q respectively. It was considered that only

box constraints appear on the system, therefore inequality (4.1c) was employed, which means the terms $C_I(\xi)$ and $A_I(\xi)$ in the NLP IPM formulation in section 4.2 can be replaced by $C\xi$ and C respectively. This leaves the difference between the LMPC and NMPC formulations to the equality constraints only. The solution of the primal-dual system of equations (4.15a) must be found on every sampling instant. The system matrix, which will be referred to as A_{ip}, has a similar structure as in the linear case, but differs in the following terms:

- $A_E(\xi)$, this term corresponds to the Jacobian matrix of the vector of equality constraints $C_E(\xi)$
- $\nabla^2_{\xi\xi}L$, this term corresponds to the Hessian matrix of the Lagrange function (4.12) and equals: $Q + \nabla_\xi(A_E^T(\xi)\lambda)$

The fourth order Runge-Kutta method was employed to solve the IVP for the discretization of the equality constraints along the prediction horizon, which results in function C_E. This function must be evaluated, on each iteration of the IPM, on the current optimization variables vector ξ and the initial state x_0 measured at the current sampling instant. The same applies for the Jacobian and the Hessian matrices $A_E(\xi)$ and $\nabla_\xi(A_E^T(\xi)\lambda)$; the latter requires additionally the current multipliers vector λ. These matrices and vector are obtained using the tool CasADi [3], which outputs C code functions for this purpose. As these matrices are sparse, CasADi uses the CCS format and outputs only the non-zero values vector for each matrix, as the row and column indexes vectors remain constant because of the fixed structure of the problem.

As the data previously mentioned is available in CCS format the same storage format was used for the whole system matrix A_{ip}. The fixed structure of this matrix allows for off-line computation of row and column indexes vectors and only the non-zero values are computed on-line. Using this compressed storage format, bandedness of the matrix is not relevant, therefore the structure in (4.15a) was not modified.

5.5.2 NMPC Formulation

The implemented NMPC algorithm solves NLP problem (4.11) that results from the simultaneous approach, i.e., considering both the system states and control inputs as optimization variables. This approach results in sparse matrices and vectors of a well defined structure, which allows the algorithms to be tailored to reduce memory requirements and computation time.

The compressed storage format used for the primal-dual system matrix precluded the use of the preconditioner used in the linear case. Double precision floating-point

representation (64 bits) was used for the implementation due to the wide dynamic range required for the solution of the IPM algorithm. Algorithm 13 presents a general NMPC formulation.

Algorithm 13: NMPC algorithm

input : Prediction horizon N, sampling period T_s, nonlinear plant model, constraints

foreach *prediction horizon* **do**

Measure or estimate plant states at current sampling instant;
Formulate NLP problem;
Solve NLP problem to get optimal solution ξ^* for the optimization variables;
Apply first optimal input(s) u^* to the plant, corresponding to the first interval of the prediction horizon;

end

Trajectory tracking was also considered for this approach and therefore the implementation was tailored to this kind of application. Matrices and vectors of the NLP problem (4.11) have the following characteristics:

- Matrix Q: Was assumed to be a constant diagonal matrix, only the diagonal values were stored in the FPGA.

- Matrix C: This matrix as in the linear case was assumed to be constant and that all control inputs were constrained. Only a vector (C_p) indicating which states are constrained was specified before synthesis, all operation on this matrix were translated into code considering its special structure.

- vector q: This vector depends on the reference trajectory, was considered an input to the solver.

- vector d: As constraints were assumed constant this vector is also constant and was stored in the FPGA.

- vector $C_E(\xi)$: this vector was implemented as a function, the input parameters are the current optimization variables vector ξ which updates at every iteration of the IPM algorithm and the initial state x_0 measured or estimated at each sampling instant.

5.5.3 Primal-dual IPM implementation for NLP

Primal-dual IPM was used for the solution of problem (4.11), considering the solution of the nonlinear system of equations (4.15a). A function named NLP_Solver was created for the solution of the NLP problem. As described previously, constant vectors and parameters were stored using on-chip memories. Information about the desired reference trajectory was given through input vector q, the current state measurement or estimation x_0 is also input to the solver. Figure 5.6 shows a block diagram of function NLP_-Solver. The inputs and outputs shown were chosen to be streamed in and out using AXI4-Stream [71] protocol for communication with other IP blocks, for this purpose a top function was created with input and output streams as arguments such that they are automatically implemented as data ports for the RTL design during synthesis by Vivado® HLS as was done for the linear case. As previously explained, the structure of the primal-dual matrix A_{ip} allows, in addition to matrix Q (only diagonal terms) and vector d, the following vectors to be computed off-line and stored using on-chip memories: row index and column pointer vectors considering the CCS storage format for $A_E(\xi)$, $A_E^T(\xi)$, $\nabla_\xi(A_E^T\lambda)$, and A_{ip}.

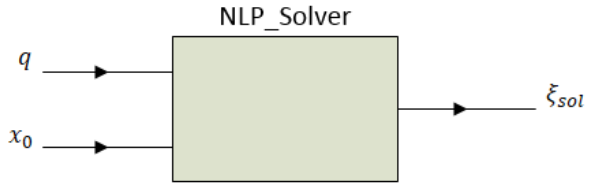


Figure 5.6 – Block diagram for NLP_Solver function

Once the steps explained above have been completed the near to optimal solution is found with the primal-dual IPM. For this purpose, function IP_algorithm was implemented. Figure 5.7 shows the block diagram of the function inputs and outputs, where the input matrices indicated refer only to the row index and column pointer vectors for CCS format as previously explained. The non-zero values vectors for this matrices are updated on every iteration of the algorithm as local variables.

Next, the implemented function IP_algorithm is explained in detail:

- Vectors ξ (feasible), λ, v and s are initialized, the latter two element-wise positive.
- The reduction factor σ and constant β (for the step length computation) are specified in the interval $(0, 1)$.
- Main loop of IPM algorithm. A fixed number of iterations was chosen as stopping criteria for the IPM algorithm since the real-time nature of the implementation

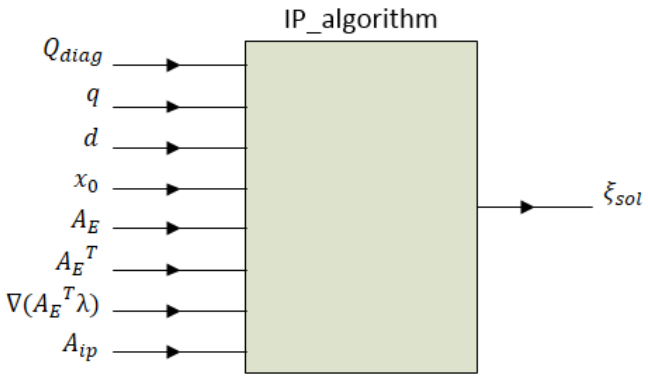


Figure 5.7 – Block diagram for IP_algorithm function for NLP

favors a deterministic timing; it also allows to trade off accuracy for computation time [62] [27].

- The first step is to compute vectors rd and rp (refer to equation (4.16)). For computation of rd there are three matrix-vector multiplications required:
  - $Q\xi$ is performed element-wise, between vector Q_{diag} and ξ.
  - $A_E^T(\xi)\lambda$, for this operation the transpose of the Jacobian of the equality constraints function must be evaluated at the current value of ξ and x_0, this is done using a function obtained with CasADi. The multiplication of this matrix with vector λ is performed with a function that performs matrix-vector multiplication in CCS format as shown in algorithm 14.
  - $C^T v$ uses the vector C_p instead of matrix C as shown in algorithm 6
- For computation of rp, a function obtained with CasADi was employed to evaluate the function of equality constraints at the current value of ξ and x_0.
- Matrix C has to be multiplied with the optimization variables vector ξ. A function similar to the one shown in algorithm 9 was implemented for this purpose, without considering the operations on vectors d and s.
- Computation of rn (refer to equation (4.16c)) is performed in a similar way as for the linear case, recall that $A_I(\xi)$ is simply replaced by matrix C because of the assumptions made for the inequality constraints.
- Matrix $A_E(\xi)$ is computed with a function generated with CasADi tool, with the current value of ξ and x_0 as inputs. Matrix $\nabla_\xi(A_E^T(\xi)\lambda)$ is also computed in a similar way, requiring ξ, x_0 and λ as inputs.
- Vector b_{ip} (right side vector of primal-dual system (4.15a)) is formed from vectors rn and rp.

- Matrix $A_I^T(\xi)S^{-1}VA_I(\xi)$ has diagonal terms only, these terms are stored in a vector using an algorithm similar to algorithm 11, without considering the terms in Q.
- Non-zero values of matrix A_{ip} are computed from vector Q_{diag}, matrix $\nabla_\xi(A_E^T(\xi)\lambda)$, the vector computed in the previous step and the Jacobian matrix $A_E(\xi)$. This step is done considering the special structure of this matrices in the CCS format.
- System (4.15a) is solved using MINRES algorithm 1. The actual implementation will be explained later in this chapter.
- Vectors $\Delta\xi$ and $\Delta\lambda$ are obtained from the solution of the primal-dual system.
- Vectors Δv and Δs are computed (refer to equations (4.15c) and (4.15d)). For the computation of these vectors, the same considerations are taken on matrices C, V and S as explained in previous steps.
- The step size or step length α is found following the criteria presented in [52]. The step length is chosen such that the dual feasibility constraint (4.15b) from the KKT conditions is satisfied, this allows the evaluation to be reduced to the terms in Δv and Δs that are negative. In this case two step lengths were considered, one for the primal variables and one for the dual variables as shown in equation (5.17), where β is a constant to guarantee the constraint is strictly satisfied.

$$\alpha_v = min(\frac{-\beta v}{\Delta v}, 1)\,, \text{ for } \Delta v \text{ negative} \tag{5.17a}$$

$$\alpha_s = min(\frac{-\beta s}{\Delta s}, 1)\,, \text{ for } \Delta s \text{ negative} \tag{5.17b}$$

- Vectors ξ, λ, v and s are updated, each in its respective search direction computed previously, considering step length α_s for vectors ξ and s and α_v for vectors λ and v.
- Finally, the barrier parameter μ is updated, refer to algorithm 3.

5.5.4 MINRES Algorithm Implementation for NLP

Figure (5.8) shows a block diagram of the inputs and outputs of the implemented algorithm. Next, an explanation of the most important components of the implementation is given:

- The termination criteria was set to a fixed number of iterations for the same reason as in the IPM; the number of iterations chosen equals the order of the system matrix A_{ip}.
- The main operation in the algorithm is matrix-vector multiplication regarding the system matrix. As the input matrix is in CCS format a function was implemented for this operation as shown in algorithm 14, where vectors *val*, *row_ind* and *col_ptr* represent the system matrix.
- The previous solution vector is used as input for the solver, as an initial guess, to improve convergence. This implies that the solution vector must be initialized a feasible starting point before the first iteration is performed.

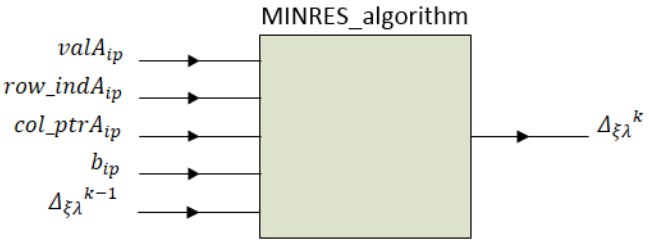


Figure 5.8 – Block diagram for MINRES_algorithm function for NLP

Algorithm 14: Matrix-vector multiplication in CCS format

input : *val*, *row_ind*, *col_ptr*, *vect*
output : *out*
/* *rows* and *cols* are the dimensions of the input matrix. *maxit* is the maximum difference between two adjacent values in *col_ptr* */
for *i=0* **to** *rows-1* **do**
  $out[i] \leftarrow 0$;
endfor
for *i=0* **to** *cols-1* **do**
  for *k=0* **to** *maxit-1* **do**
    if $k <$ *(col_ptr[i+1]-col_ptr[i])* **then**
      $out[row\_ind[k + col\_ptr[i]]] \leftarrow$
      $out[row\_ind[k + col\_ptr[i]]] + val[k + col\_ptr[i]]vect[i]$;
    end
  endfor
endfor

5.6 Optimization Directives

The use of HLS for FPGA programming allows the programmer to influence the way a design is synthesized through the use of optimization directives in the form of pragmas directly inserted in the C-code. For the current implementation, the following directives provided by Vivado® HLS were used:

- Loop Unrolling: This directive allows for parallel execution of operations in a loop by creating copies of the loop body such that operations corresponding to different loop indexes are performed in parallel, this means multiple hardware implementations of the loop body are created trading off resources utilization for higher throughput and performance. The unrolling can be full or partial, with the desired factor specified by the programmer [75]. Loop unrolling was used mainly on operations involving vectors in the MINRES_algorithm function, as these operations do not present data dependencies.

- Loop Pipelining: This directive allows for the concurrent implementation of the code, such that operations are not forced to occur in a sequential manner. It was used in most for-loops as it allows operations corresponding to the next iteration of the loop to begin prior the completion of the current iteration [75]. For nested loops, this directive automatically unrolls loops underneath the loop where it is applied. So care must be taken as this can significantly increase the resources utilization.

- Array Partition: Special attention was given to array handling as they can cause performance bottlenecks. Arrays are implemented as RAMs, ROMs or FIFO memories using either BRAMs, LUTs or registers [76]. If BRAMs are used, memory accesses in each clock cycle are limited by the number of ports of the block (maximum 2) and can, therefore, preclude parallel execution of operations. To solve this issue arrays can be partitioned on three different forms: cyclic, block or complete. Cyclic partition was employed on the vectors in MINRES implementation, to be able to take advantage of the loop unrolling.

- Inline: When applied to a function, this directive eliminates the function call replacing it with a copy of the function body. This allows optimizations with surrounding operations to occur more effectively and can, therefore, improve latency [73], while maintaining the original function-based C code.

- Interface: This directive allows the programmer to specify how the RTL ports will be created. It was used to specify the I/O ports of the QP_Solver and NLP_Solver

functions to be implemented using an AXI4-Stream interface for AXI4-Stream protocol [71].

5.7 Integration in Vivado IDE

As mentioned in sections 5.4.3 and 5.5.3 the functions QP_Solver and NLP_Solver were implemented in Vivado® HLS. After the RTL design was obtained during the synthesis stage it was exported as an IP block so it could be integrated into a more complex design using the software Vivado® IDE. During RTL export, VHDL was chosen as target language. As explained previously the interface for these functions was implemented for the use of AXI4-Stream protocol, therefore the AXI4-Stream interface was used. Figure 5.9 shows the symbolic representation of the exported IP when used in the Vivado® IP integrator, which corresponds to the hardware accelerator for the solution of the QP problem. Matrices A_{dN}, B_{dN} and vectors d and x_0 are streamed in through IN_STREAM input; the output ξ_{sol} is streamed out through OUT_STREAM. Data is streamed in and out for the hardware accelerator for the NLP problem in a similar way. As mentioned previously, Vivado® HLS automatically synthesizes the top-level function parameters as data ports; additionally, the top-level function itself is also implemented with an input/output protocol that allows for operation start and indicates when the operations have been completed and new input data can be accepted [21]. These control ports are represented by the input s_axi_CONTROL_BUS in figure 5.9 and are managed by the PS.

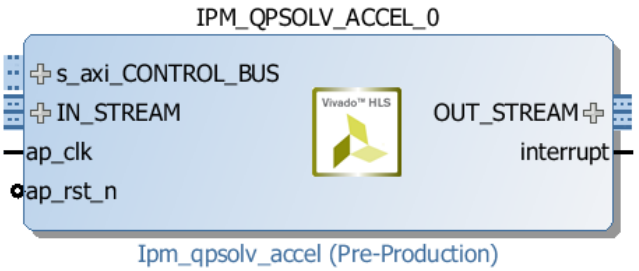


Figure 5.9 – Block diagram for MINRES_algorithm function

For the case studies regarding LMPC, the linearization of the models along the reference trajectory leads to a time-varying system, the operating points were obtained off-line for the given trajectory, and were stored in flash memory on the Zynq® ZC706 Evaluation Kit. The process of reading the flash memory for the corresponding prediction horizon on each sampling period to form matrices A_{dN} and B_{dN}, as well as updating vector d and providing vector x_0 to the accelerator is carried out in the PS part of

the Zynq® -7000 AP SoC, as the computation cost for these operations is very low compared with that of the solution of the QP. For the nonlinear case, the reference trajectory is also stored in flash memory and vector q is updated in the PS before being streamed to the accelerator. Therefore, a communication between the PS and the PL is required as data must be streamed into the accelerator from the PS and then streamed out from the accelerator to the PS.

The PS communicates with the PL using the AXI4 protocol, designed for memory-mapped links, which means that an address in the processor memory space must be specified by the master for read or write operations [21]. On the other hand, the IP core corresponding to the hardware accelerator was implemented using AXI4-Stream interfaces, which allow high-speed point-to-point data transfer between two cores on the PL using a producer-consumer model by streaming data, that is without the need for addressing (not memory-mapped) [4]. As the protocols used are different, the Xilinx® IP core AXI DMA was used to allow for high-bandwidth direct memory access between the memory on the PS and the accelerator via an AXI4-Stream interface [4]. The Accelerator Coherency Port (ACP) on the PS was used to achieve coherency between the caches on the PS and the elements in the PL, providing a low latency path for the communication to take place [21]. Figure 5.10 shows a simplified scheme of the implementation. A representation of the block diagram design considering interface connections only for the LMPC implementation is shown in figure 5.11, where letters M and S at the beginning of the port names mean Master and Slave port respectively, AXIS means AXI4-Stream interface, MM2S means memory-mapped to stream and S2MM means stream to memory-mapped. For the NMPC implementation, only the hardware accelerator block must be replaced by the corresponding solver.

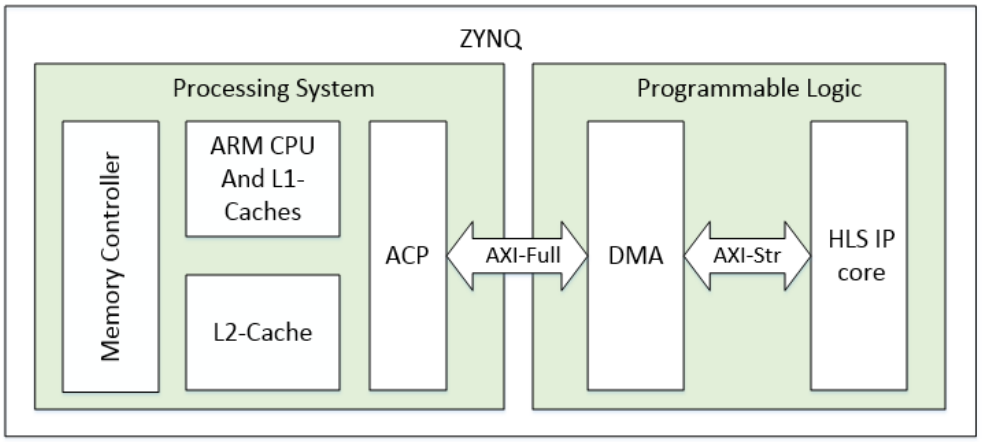


Figure 5.10 – Block diagram for PS-PL communication [4]

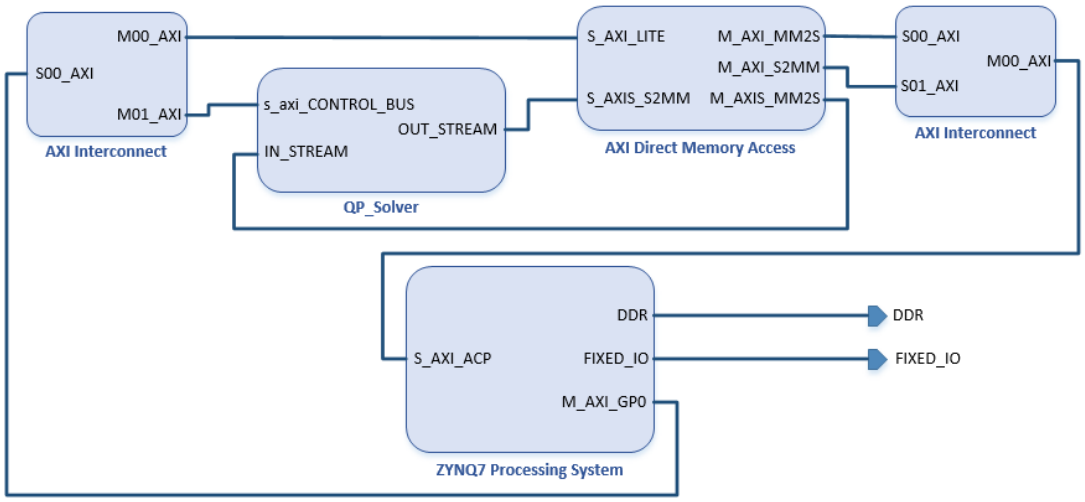


Figure 5.11 – Block design in Vivado IDE

After completion of the block design on the IP Integrator, the following steps were executed using Vivado® IDE: output products generation, HDL wrapper creation, synthesis, implementation and bitstream generation. After this, the created hardware platform and bitstream were exported to Vivado® SDK where the PS was programmed to read the flash memory, stream data in and out the PL and communicate through a serial interface for sending back the computed results of the simulation to a desktop computer. Both the ARM® processor and the FPGA were programmed using the Vivado® SDK environment.

5.8 Summary

This chapter described in detail the implementation as hardware accelerators of the primal-dual IPM solvers for the solution of the QP and the NLP problems arising from the LMPC and the NMPC formulation respectively. Special considerations and assumptions, as well as how the structure of the problem was exploited were detailed. Optimization directives employed to take advantage of the parallelization capabilities of the FPGA were also explained. The integration of the accelerator in a design targeting the Zynq® device to take advantage of the system-on-a-chip architecture was also detailed.

Chapter 6

Case Studies

The implemented MPC is coded in the C++ language using HLS to obtain an RTL model for its implementation on a Zynq® device, which contains an FPGA along with an ARM® processor, as explained in chapter 5. Trajectory tracking problems were chosen as case studies to test the performance and computational cost of the implemented design.

Trajectory tracking is a common problem presented in mobile robot applications and autonomous vehicles. The fast dynamics of these systems make the embedded application of MPC a challenge regarding computation time. Furthermore, these systems are always nonlinear, which implies that a linearization along the trajectory must be considered in order to apply LMPC as presented in [19, 36, 50, 53].

6.1 Case Studies for LMPC

6.1.1 Linearization Along Trajectories

Assume the following nonlinear continuous-time model of the system to be controlled:

$$\dot{x} = f(x, u)\,, \tag{6.1}$$

where $x \in \mathbb{R}^n$ represents the system states and $u \in \mathbb{R}^m$ the controlled inputs, both time dependent. From this model, a virtual system is considered to obtain the desired reference states and inputs along the desired trajectory. Considering this reference points the nonlinear system (6.1) is successively linearized using first order Taylor expansion at each operating point (x_{ref}, u_{ref}):

$$f(x, u) = f(x_{ref}, u_{ref}) + J_f(x_{ref}, u_{ref}) \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix}\,, \tag{6.2}$$

where $J_f(x_{ref}, u_{ref})$ represents the Jacobian of the function f evaluated at the operating point and $\Delta x = x - x_{ref}$. Reordering the terms into the standard linear state space formulation yields

$$\Delta \dot{x} = A_{lc}\Delta x + B_{lc}\Delta u\,. \tag{6.3}$$

For the LMPC problem formulation, a discrete-time model is required, the following system will be considered after model discretization:

$$\Delta x_{k+1} = A_d \Delta x_k + B_d \Delta u_k\,. \tag{6.4}$$

Since the operation point will vary along the reference trajectory, a time-varying system is obtained. For the sake of coherency with the description in chapter 3, the following simplified notation can be adopted for equation (6.4) (see equation (3.1)):

$$x_{k+1} = A_k x_k + B_k u_k\,. \tag{6.5}$$

Matrices A_k and B_k for a corresponding prediction horizon are stored in matrices AdN and BdN, respectively, for the IPM algorithm as explained in chapter 5.

6.1.2 Three-state Bicycle Model

The car-like mobile vehicle model treated in this section considers kinematic relationships and a two dimensional movement only, neglecting slip. The name bicycle model arises from the fact that the wheels on both rear and front axles are merged so that only one wheel is considered for each axle. For the two dimensional case, because of the number of actuators being less than three, it is a non-holonomic system and not all trajectories are possible as the vehicle moves along the direction determined by the heading wheels and the longitudinal orientation [81]. The model considered in [42] is presented in figure 6.1, with rear wheel traction and front wheel steering. The nonlinear model equations are presented in equation (6.6)[42], where x and y are the vehicle's position, θ is the orientation with respect to the X axis, v is the longitudinal velocity and φ is the steering angle with respect to the vehicle's longitudinal axis. Both v and φ are considered inputs to the system.

$$\begin{aligned} \dot{x} &= v cos(\theta)\,, \\ \dot{y} &= v sin(\theta)\,, \\ \dot{\theta} &= \frac{v}{l} tan(\varphi)\,. \end{aligned} \tag{6.6}$$

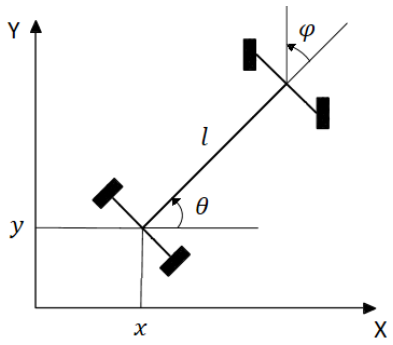


Figure 6.1 – Three-state bicycle model [42]

Linearizing the model near an arbitrary operating point leads to the system in equation (6.7), with $\tilde{x}$ and $\tilde{u}$ as presented in (6.8).

$$\dot{\tilde{x}} = \begin{bmatrix} 0 & 0 & -v_{ref}sin(\theta_{ref}) \\ 0 & 0 & v_{ref}cos(\theta_{ref}) \\ 0 & 0 & 0 \end{bmatrix} \tilde{x} + \begin{bmatrix} cos(\theta_{ref}) & 0 \\ sin(\theta_{ref}) & 0 \\ \frac{1}{l}tan(\varphi_{ref}) & \frac{v_{ref}}{l}(1+tan(\varphi_{ref})^2) \end{bmatrix} \tilde{u}\,, \tag{6.7}$$

$$\tilde{x} = \begin{bmatrix} x - x_{ref} \\ y - y_{ref} \\ \theta - \theta_{ref} \end{bmatrix}, \quad \tilde{u} = \begin{bmatrix} v - v_{ref} \\ \varphi - \varphi_{ref} \end{bmatrix}. \tag{6.8}$$

After discretization using forward Euler method also called forward rectangular method, the following model is obtained:

$$\tilde{x}_{k+1} = \begin{bmatrix} 1 & 0 & -T_s v_{ref}sin(\theta_{ref}) \\ 0 & 1 & T_s v_{ref}cos(\theta_{ref}) \\ 0 & 0 & 1 \end{bmatrix} \tilde{x}_k + \begin{bmatrix} T_s cos(\theta_{ref}) & 0 \\ T_s sin(\theta_{ref}) & 0 \\ \frac{T_s}{l}tan(\varphi_{ref}) & \frac{T_s v_{ref}}{l}(1+tan(\varphi_{ref})^2) \end{bmatrix} \tilde{u}_k\,, \tag{6.9}$$

where T_s is the sampling period. For the implementation, the parameters presented in table 6.1 are considered. Weighting matrices were chosen as given in (6.10). As can be seen, differences in the vehicle's position with respect to the desired reference are strongly penalized.

The reference trajectory is shown in dotted red line and the calculated trajectory in solid blue in figure 6.2. Figure 6.3 shows the calculated control inputs as well as their constraints, it can be seen that the constraints are respected. Figure 6.4 shows the computation time required for prediction horizon values of: $N = 5,\ 6,\ 7,\ 8,\ 9,\ 10$;

as expected, the computation time increases with the value of the prediction horizon and for 10 or more it surpasses the sampling period for this particular problem. This means that the implemented MPC is not able to control the system in real-time if the prediction horizon considers 10 or more sampling instants.

Table 6.1 – Parameters for LMPC implementation of bicycle like model

Parameter	Value
l	0.2 m
Sampling period (T_s)	0.01 s
Prediction horizon (N)	5
IPM iterations	12
State constraints	$-\pi \leq \theta \leq \pi$ [rad]
Input constraints	$-1.5 \leq v \leq 1.5$ [m/s]
	$-0.65 \leq \varphi \leq 0.65$ [rad]
Initial state	$x_0 = [-1 \quad -1 \quad 0]^T$
Initial reference state	$x_0 = [0 \quad 0 \quad \pi/2]^T$

$$
\begin{aligned}
Q_i &= \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \quad \text{, for } i = 1, 2, ..., N-1\,, \\
R_i &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{, for } i = 0, 1, ..., N-1\,, \\
P &= 20 Q_1\,.
\end{aligned}
\tag{6.10}
$$

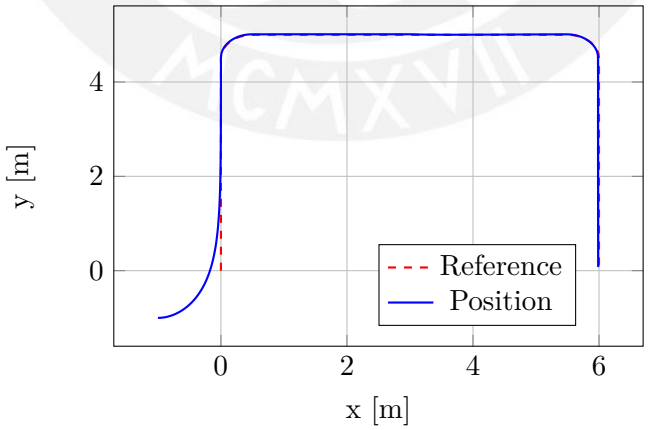


Figure 6.2 – Trajectory in the X-Y plane followed by the three-state vehicle model for parameters as shown in table 6.1

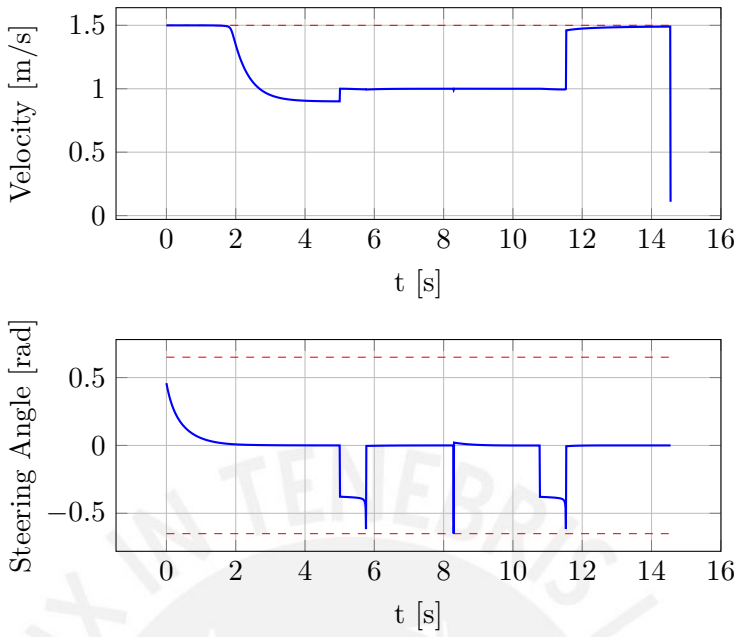


Figure 6.3 – Control inputs for the three-state vehicle model considering parameters given in table 6.1

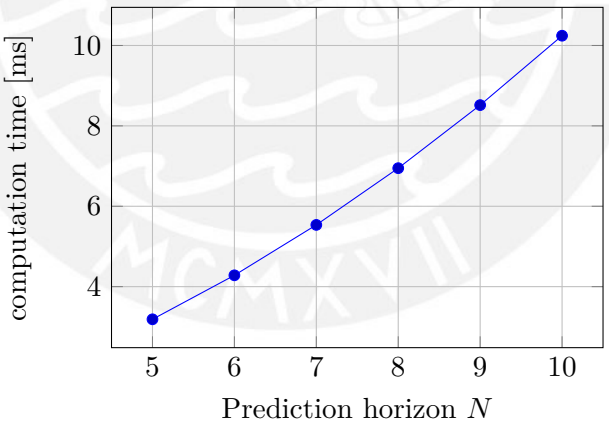


Figure 6.4 – Computation times for the bicycle model 1 problem.

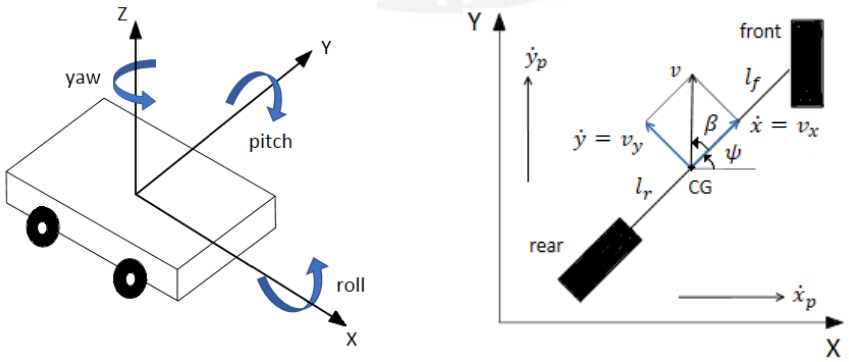
Resource utilization, timing achieved after implementation, power consumption and the computation time for the solution of the LMPC problem are summarized in table 6.2, where latency and initiation interval (II) are given in number of clock cycles.

Table 6.2 – Metrics for three-state bicycle-like model after implementation, $N = 5$

Metric	Value
Clock Period	10 ns
Max. Latency	318767
Max. II	318768
Computation Time	3.2 ms
BRAM (%)	9
DSP48E (%)	26
FF (%)	16
LUT (%)	34
LUTRAM (%)	7
Total On-Chip Power	4045 W

6.1.3 Five-state Bicycle Model

A better kinematic model for a car-like mobile vehicle takes into consideration roll, pitch and yaw movements in a three-dimensional plane as shown in figure 6.5a. For this case study a simplified kinematic lateral bicycle model is considered, neglecting roll, pitch and tire slip angle as presented in [1] and used in [19, 53]. The model considered is presented in figure 6.5b. The nonlinear model is presented in equation (6.11) [1], with state and input vectors as shown in equation (6.12), where β is the vehicle side slip angle, ψ is the yaw angle, $\dot{\psi}$ is the yaw rate, x_p and y_p are the coordinates of the vehicle's center of gravity (CG) in an inertial frame, v_x is the velocity component along the vehicle's longitudinal axis and δ_f is the front tire steering angle. The parameters used for the model and their values are presented in table 6.3 [1].



(a) Roll, pitch and yaw movements **(b)** Bicycle model

Figure 6.5 – Five-states bicycle model

$$
\begin{aligned}
\dot{\beta} &= \frac{2}{Mv_x}\left(C_f(\delta_f - \beta - \frac{l_f\dot{\psi}}{v_x}) + C_r(-\beta + \frac{l_r\dot{\psi}}{v_x})\right) - \dot{\psi}\,, \\
\dot{\psi} &= \dot{\psi}\,, \\
\ddot{\psi} &= \frac{2}{I_z}\left(l_f C_f(\delta_f - \beta - \frac{l_f\dot{\psi}}{v_x}) - lrC_r(-\beta + \frac{l_r\dot{\psi}}{v_x})\right)\,, \\
\dot{x}_p &= v_x cos(\psi) - v_x tan(\beta) sin(\psi)\,, \\
\dot{y}_p &= v_x sin(\psi) + v_x tan(\beta) cos(\psi)\,.
\end{aligned}
\tag{6.11}
$$

$$
x = \begin{bmatrix} \beta \\ \psi \\ \dot{\psi} \\ x_p \\ y_p \end{bmatrix}\,, \qquad u = \begin{bmatrix} v_x \\ \delta_f \end{bmatrix}\,, \tag{6.12}
$$

Table 6.3 – Parameters for five-state bicycle-like model [1]

Parameter	Value	Description
C_f	66 900 N/rad	Cornering stiffness considering lateral force in the front tire
C_r	62 700 N/rad	Cornering stiffness considering lateral force in the rear tire
l_f	1.232 m	Longitude from CG to front tire
l_r	1.468 m	Longitude from CG to rear tire
I_z	4175 kg m^2	Yaw moment of inertia
M	1723 kg	Vehicle mass

Linearizing the model near an arbitrary operating point leads to the system and input matrices presented in equation (6.13), where the values of the variables at the operating point are designated with a p subscript, s, c and t_a mean sine, cosine and tangent functions respectively, $C_r l_r - C_f l_f$ has been replaced with b and $C_r l_r^2 + C_f l_f^2$ has been replaced with d. After discretization using forward Euler method, the model matrices in (6.14) are obtained, where $T_s v_{xp}$ has been replaced with a_p. For the implementation,

parameters as presented in table 6.4 were considered. Weighting matrices were chosen as shown in equation (6.15).

$$A_{lc} = \begin{bmatrix} \frac{-2}{Mv_{xp}}(C_f + C_r) & 0 & \frac{2b}{Mv_{xp}^2} - 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \frac{2b}{I_z} & 0 & \frac{-2d}{I_z v_{xp}} & 0 & 0 \\ -v_{xp}s(\psi_p)(1 + t_a(\beta_p)^2) & -v_{xp}(s(\psi_p) + t_a(\beta_p)c(\psi_p)) & 0 & 0 & 0 \\ v_{xp}c(\psi_p)(1 + t_a(\beta_p)^2) & v_{xp}(c(\psi_p) - t_a(\beta_p)s(\psi_p)) & 0 & 0 & 0 \end{bmatrix},$$

$$B_{lc} = \begin{bmatrix} \frac{2}{M}v_{xp}^{-2}(2C_f l_f \dot{\psi}_p v_{xp}^{-1} - 2C_r l_r \dot{\psi}_p v_{xp}^{-1} + C_f(\beta_p - \delta_{fp}) + C_r\beta_p) & \frac{2C_f}{Mv_{xp}} \\ 0 & 0 \\ \frac{2}{I_z}(C_f l_f^2 \dot{\psi}_p v_{xp}^{-2} + C_r l_r^2 \dot{\psi}_p v_{xp}^{-2}) & \frac{2}{I_z}C_f l_f \\ c(\psi_p) - t_a(\beta_p)s(\psi_p) & 0 \\ s(\psi_p) + t_a(\beta_p)c(\psi_p) & 0 \end{bmatrix}. \tag{6.13}$$

$$A_d = \begin{bmatrix} 1 - \frac{2T_s}{Mv_{xp}}(C_f + C_r) & 0 & T_s(\frac{2b}{Mv_{xp}^2} - 1) & 0 & 0 \\ 0 & 1 & T_s & 0 & 0 \\ \frac{2bT_s}{I_z} & 0 & 1 - \frac{2dT_s}{I_z v_{xp}} & 0 & 0 \\ -a_p s(\psi_p)(1 + t_a(\beta_p)^2) & -a_p(s(\psi_p) + t_a(\beta_p)c(\psi_p)) & 0 & 1 & 0 \\ a_p c(\psi_p)(1 + t_a(\beta_p)^2) & a_p(c(\psi_p) - t_a(\beta_p)s(\psi_p)) & 0 & 0 & 1 \end{bmatrix},$$

$$B_d = \begin{bmatrix} \frac{2T_s}{M}v_{xp}^{-2}(2C_f l_f \dot{\psi}_p v_{xp}^{-1} - 2C_r l_r \dot{\psi}_p v_{xp}^{-1} + C_f(\beta_p - \delta_{fp}) + C_r\beta_p) & \frac{2T_s C_f}{Mv_{xp}} \\ 0 & 0 \\ \frac{2T_s}{I_z}(C_f l_f^2 \dot{\psi}_p v_{xp}^{-2} + C_r l_r^2 \dot{\psi}_p v_{xp}^{-2}) & \frac{2T_s}{I_z}C_f l_f \\ T_s(c(\psi_p) - t_a(\beta_p)s(\psi_p)) & 0 \\ T_s(s(\psi_p) + t_a(\beta_p)c(\psi_p)) & 0 \end{bmatrix}. \tag{6.14}$$

$$\begin{aligned} Q_i &= \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad , \text{for } i = 1, 2, ..., N-1\,, \\ R_i &= \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix} \quad , \text{for } i = 0, 1, ..., N-1\,, \\ P &= 10Q_1\,. \end{aligned} \tag{6.15}$$

Table 6.4 – Parameters for LMPC implementation of five-state bicycle-like model

Parameter	Value
Sampling period (T_s)	0.01 s
Prediction horizon (N)	5
IPM iterations	12
State constraints	$-0.3 \leq \beta \leq 0.3$ [rad]
	$-2\pi \leq \psi \leq 2\pi$ [rad]
Input constraints	$-3 \leq v_x \leq 3$ [m/s]
	$-0.3 \leq \delta_f \leq 0.3$ [rad]
Initial state	$x_0 = [0 \quad 0 \quad 0 \quad 15 \quad 29]^T$
Initial reference state	$x_0 = [0 \quad 0 \quad 0 \quad 20 \quad 30]^T$

The reference trajectory is shown in dotted red line and the calculated trajectory in solid blue in figure 6.6. Figures 6.7 and 6.8 show the calculated control inputs for the desired trajectory, which adequately respect the constraints.

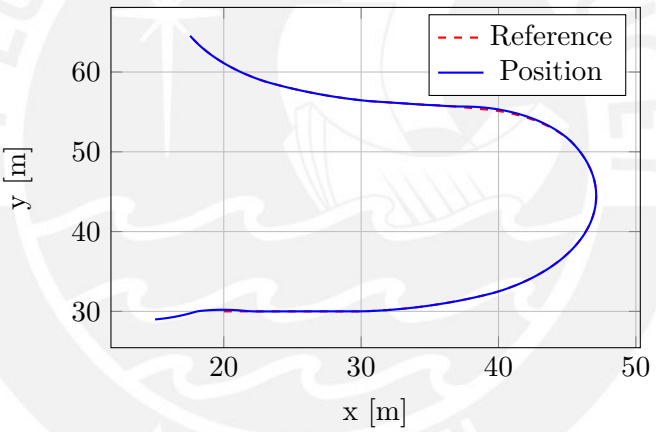


Figure 6.6 – Trajectory in the X-Y plane followed by the five-state vehicle model

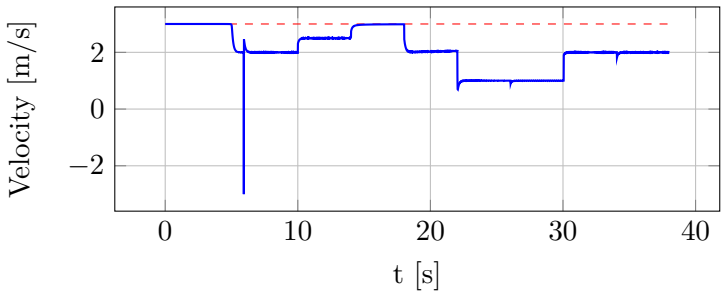


Figure 6.7 – Velocity considering parameters given in table 6.4

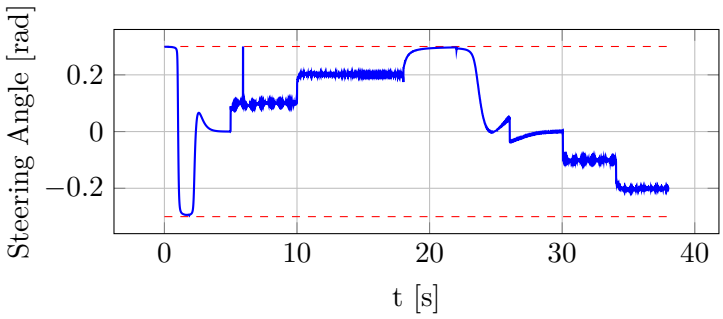


Figure 6.8 – Steering angle considering parameters given in table 6.4

Figure 6.9 shows the computation time required for prediction horizon values of: $N = 3,\ 5,\ 6,\ 8,\ 10$. The computation time for this case exceeds the sampling period, that is real-time is not guaranteed, for prediction horizon values greater than 5.

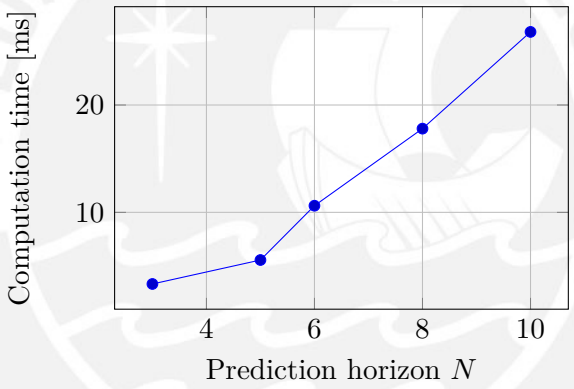


Figure 6.9 – Computation times for the five-state bicycle model problem.

Resource utilization, timing, power consumption and the computation time for the solution of the LMPC problem achieved after implementation are summarized in table 6.5, where latency and II are given in number of clock cycles. Figure 6.10 shows the effect of the size of the linear system of equations to solve, namely system (4.9a), on the computation time of the implemented LMPC algorithm. The size of this system depends on the prediction horizon (N) chosen, and the number of control inputs (m) and states (n) of the model and equals $N(2n+m)$. This is important because even if the number of optimization variables ($N(n+m)$) is the same for two different problems, the number of states plays an important role and can make the computation time for these systems differ. This is because the sparse approach was considered as explained in section 3.2.3.2.

Table 6.5 – Metrics for five-state bicycle-like model after implementation, $N = 5$

Metric	Value
Clock Period	10 ns
Max. Latency	555950
Max. II	555951
Computation Time	5.6 ms
BRAM (%)	8
DSP48E (%)	26
FF (%)	19
LUT (%)	44
LUTRAM (%)	13
Total On-Chip Power	4233 W

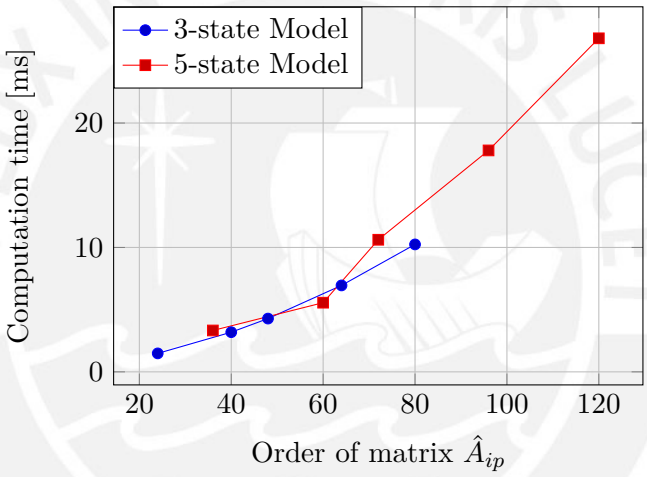


Figure 6.10 – Computation time for different orders of matrix $\hat{A}_{ip}$, refer to equation (5.8), for prediction horizon $N = 3,\ 5,\ 6,\ 8,\ 10$

As can be observed in figures 6.7 and 6.8, the rate of change of the calculated control inputs is high for some sections of the trajectory, which may not be physically realizable on a real system. To overcome this inconvenient, constraints must be also set on the change in the control inputs. As the implementation only considers box constraints on the optimization variables, the changes in the inputs must be considered as new input signals. For this purpose, the order of the system was augmented such that the original control inputs are treated as states and their variation is considered as a control input. The new state and input vectors are as shown:

$$x = \begin{bmatrix} \beta_k & \psi_k & \dot{\psi}_k & x_{p_k} & y_{p_k} & v_{x_{k-1}} & \delta_{f_{k-1}} \end{bmatrix}^T , \tag{6.16}$$

$$u = \begin{bmatrix} \Delta v_{x_k} & \Delta \delta_{f_k} \end{bmatrix}^T , \tag{6.17}$$

where the change in the original inputs, denoted with operator Δ, means the difference between their current and previous values. The parameters as given in table 6.6 were considered, all other parameters were set as given in table 6.4. The results obtained are shown in figure 6.11, obtaining a smoother curve.

Table 6.6 – Parameters for augmented five-state bicycle-like model

Parameter	Value
Prediction horizon (N)	3
Input constr.	$-0.2 \leq \Delta v_x \leq 0.2$ [m/s]
	$-0.1 \leq \Delta \delta_f \leq 0.1$ [rad]

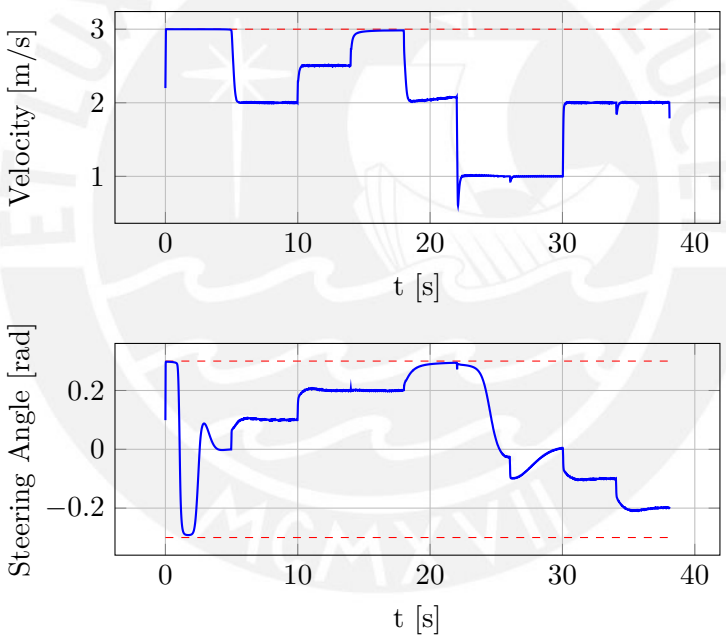


Figure 6.11 – Control inputs for the augmented five-state vehicle model considering parameters given in table 6.6

6.1.4 Satellite Model

A nonlinear model for an asymmetric, tumbling spacecraft is considered, as in [33] and [19]; a similar model is also considered in [61] and [63]. The dynamics of the system are presented in equation (6.18), where the states β_1, β_2, β_3, β_4 represent the Euler

parameters for the orientation description and $\omega_1,\ \omega_2,\ \omega_3$ represent the angular velocities of the satellite's frame relative to an inertial reference frame. The inputs $T_1,\ T_2,\ T_3$ represent the components of the control torque and $I_1,\ I_2,\ I_3$ represent the principal moments of Inertia.

$$\begin{aligned}
\dot{\beta_1} &= \frac{1}{2}\left(\omega_1\beta_4 - \omega_2\beta_3 + \omega_3\beta_2\right) , \\
\dot{\beta_2} &= \frac{1}{2}\left(\omega_1\beta_3 + \omega_2\beta_4 - \omega_3\beta_1\right) , \\
\dot{\beta_3} &= \frac{1}{2}\left(-\omega_1\beta_2 + \omega_2\beta_1 + \omega_3\beta_4\right) , \\
\dot{\beta_4} &= -\frac{1}{2}\left(\omega_1\beta_1 + \omega_2\beta_2 + \omega_3\beta_3\right) , \\
\dot{\omega_1} &= \frac{(I_2 - I_3)\omega_2\omega_3 + T_1}{I_1} , \\
\dot{\omega_2} &= \frac{(I_3 - I_1)\omega_1\omega_3 + T_2}{I_2} , \\
\dot{\omega_3} &= \frac{(I_1 - I_2)\omega_1\omega_2 + T_3}{I_3} .
\end{aligned} \tag{6.18}$$

Linearizing the model near an arbitrary operating point leads to the system and input matrices presented in equation (6.19), where the variable values at the operating point are designated with a p subscript.

$$A_{lc} = \begin{bmatrix}
0 & \frac{1}{2}(\omega_{3_p}) & -\frac{1}{2}(\omega_{2_p}) & \frac{1}{2}(\omega_{1_p}) & \frac{1}{2}(\beta_{4_p}) & -\frac{1}{2}(\beta_{3_p}) & \frac{1}{2}(\beta_{2_p}) \\
-\frac{1}{2}(\omega_{3_p}) & 0 & \frac{1}{2}(\omega_{1_p}) & \frac{1}{2}(\omega_{2_p}) & \frac{1}{2}(\beta_{3_p}) & \frac{1}{2}(\beta_{4_p}) & -\frac{1}{2}(\beta_{1_p}) \\
\frac{1}{2}(\omega_{2_p}) & -\frac{1}{2}(\omega_{1_p}) & 0 & \frac{1}{2}(\omega_{3_p}) & -\frac{1}{2}(\beta_{2_p}) & \frac{1}{2}(\beta_{1_p}) & \frac{1}{2}(\beta_{4_p}) \\
-\frac{1}{2}(\omega_{1_p}) & -\frac{1}{2}(\omega_{2_p}) & -\frac{1}{2}(\omega_{3_p}) & 0 & -\frac{1}{2}(\beta_{1_p}) & -\frac{1}{2}(\beta_{2_p}) & -\frac{1}{2}(\beta_{3_p}) \\
0 & 0 & 0 & 0 & 0 & \frac{(I_2-I_3)\omega_{3_p}}{I_1} & \frac{(I_2-I_3)\omega_{2_p}}{I_1} \\
0 & 0 & 0 & 0 & \frac{(I_3-I_1)\omega_{3_p}}{I_2} & 0 & \frac{(I_3-I_1)\omega_{1_p}}{I_2} \\
0 & 0 & 0 & 0 & \frac{(I_1-I_2)\omega_{2_p}}{I_3} & \frac{(I_1-I_2)\omega_{1_p}}{I_3} & 0
\end{bmatrix} ,$$

$$B_{lc} = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\frac{1}{I_1} & 0 & 0 \\
0 & \frac{1}{I_2} & 0 \\
0 & 0 & \frac{1}{I_3}
\end{bmatrix} . \tag{6.19}$$

After discretization using forward Euler method, the model matrices shown in (6.20) are obtained.

$$A_d = \begin{bmatrix} 1 & \frac{T_s}{2}(\omega_{3_p}) & \frac{-T_s}{2}(\omega_{2_p}) & \frac{T_s}{2}(\omega_{1_p}) & \frac{T_s}{2}(\beta_{4_p}) & \frac{-T_s}{2}(\beta_{3_p}) & \frac{T_s}{2}(\beta_{2_p}) \\ \frac{-T_s}{2}(\omega_{3_p}) & 1 & \frac{T_s}{2}(\omega_{1_p}) & \frac{T_s}{2}(\omega_{2_p}) & \frac{T_s}{2}(\beta_{3_p}) & \frac{T_s}{2}(\beta_{4_p}) & \frac{-T_s}{2}(\beta_{1_p}) \\ \frac{T_s}{2}(\omega_{2_p}) & \frac{-T_s}{2}(\omega_{1_p}) & 1 & \frac{T_s}{2}(\omega_{3_p}) & \frac{-T_s}{2}(\beta_{2_p}) & \frac{T_s}{2}(\beta_{1_p}) & \frac{T_s}{2}(\beta_{4_p}) \\ \frac{-T_s}{2}(\omega_{1_p}) & \frac{-T_s}{2}(\omega_{2_p}) & \frac{-T_s}{2}(\omega_{3_p}) & 1 & \frac{-T_s}{2}(\beta_{1_p}) & \frac{-T_s}{2}(\beta_{2_p}) & \frac{-T_s}{2}(\beta_{3_p}) \\ 0 & 0 & 0 & 0 & 1 & \frac{T_s(I_2-I_3)\omega_{3_p}}{I_1} & \frac{T_s(I_2-I_3)\omega_{2_p}}{I_1} \\ 0 & 0 & 0 & 0 & \frac{T_s(I_3-I_1)\omega_{3_p}}{I_2} & 1 & \frac{T_s(I_3-I_1)\omega_{1_p}}{I_2} \\ 0 & 0 & 0 & 0 & \frac{T_s(I_1-I_2)\omega_{2_p}}{I_3} & \frac{T_s(I_1-I_2)\omega_{1_p}}{I_3} & 1 \end{bmatrix},$$

$$B_d = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{T_s}{I_1} & 0 & 0 \\ 0 & \frac{T_s}{I_2} & 0 \\ 0 & 0 & \frac{T_s}{I_3} \end{bmatrix}. \tag{6.20}$$

In the references mentioned above, this model is used for the solution of a nonlinear optimal control problem, where the required torques are to be found in order to drive the system to a desired final state in a given period of time. For this thesis, however, a reference tracking problem is considered, as the implemented algorithm corresponds to an LMPC formulation. Furthermore, the discretization over the specified period of time required for the solution of the nonlinear optimal control problem increases significantly the number of optimization variables, precluding an FPGA implementation because of the resources that would be required.

The parameters' values used for the implementation are shown in table 6.7. For the implementation, parameters as presented in table 6.8 are considered. Weighting matrices were chosen as shown in equation (6.21). As can be seen from the weighting matrices, the difference between the actual angular velocities and the reference is highly penalized, as the considered desired behavior is the satellite to move with the specified velocities. On the other hand, deviation on the Euler parameters is not strongly penalized. For the control inputs, as an initial point different from that of the reference trajectory is considered, the weights on the matrix R_i are given a very small value; even if these values could be zero, a penalization value of 10^{-6} is considered as this leads to a smoother

response. The very high difference on the values in matrices Q_i and R_i is due to the high ratio between applied torque and angular velocity variations obtained.

Table 6.7 – Parameters for satellite model [33]

Parameter	Value
I_1	$1 \times 10^6\,\mathrm{kg\,m^2}$
I_2	$833\,333\,\mathrm{kg\,m^2}$
I_2	$916\,667\,\mathrm{kg\,m^2}$

Table 6.8 – Parameters for LMPC implementation of satellite model

Parameter	Value
Sampling period (T_s)	$0.05\,\mathrm{s}$
Prediction horizon (N)	4
IPM iterations	20
State constraints	$-1 \leq \omega_1 \leq 1$ [rad/s]
	$-1 \leq \omega_2 \leq 1$ [rad/s]
	$-1 \leq \omega_3 \leq 1$ [rad/s]
Input constraints	$-300 \leq T_1 \leq 300$ [N m]
	$-800 \leq T_2 \leq 800$ [N m]
	$-50 \leq T_3 \leq 50$ [N m]
Initial state	$x_0 = [0 \quad 0 \quad 0 \quad 1 \quad 0.009 \quad 0 \quad 0.00120]^T$
Initial reference state	$x_0 = [0 \quad 0 \quad 0 \quad 1 \quad 0.01 \quad 0.005 \quad 0.001]^T$

$$
\begin{aligned}
Q_i &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^6 \end{bmatrix}, \text{ for } i = 1, 2, ..., N-1\,, \\
R_i &= \begin{bmatrix} 10^{-6} & 0 & 0 \\ 0 & 10^{-6} & 0 \\ 0 & 0 & 10^{-6} \end{bmatrix}, \text{ for } i = 0, 1, ..., N-1\,, \\
P &= 10 Q_1\,.
\end{aligned}
\tag{6.21}
$$

The desired references for the angular velocities are shown in dotted red line and the calculated states after running the simulation on the FPGA in solid blue in figure 6.12. Calculated inputs are shown in figure 6.13, which adequately respect the constraints.

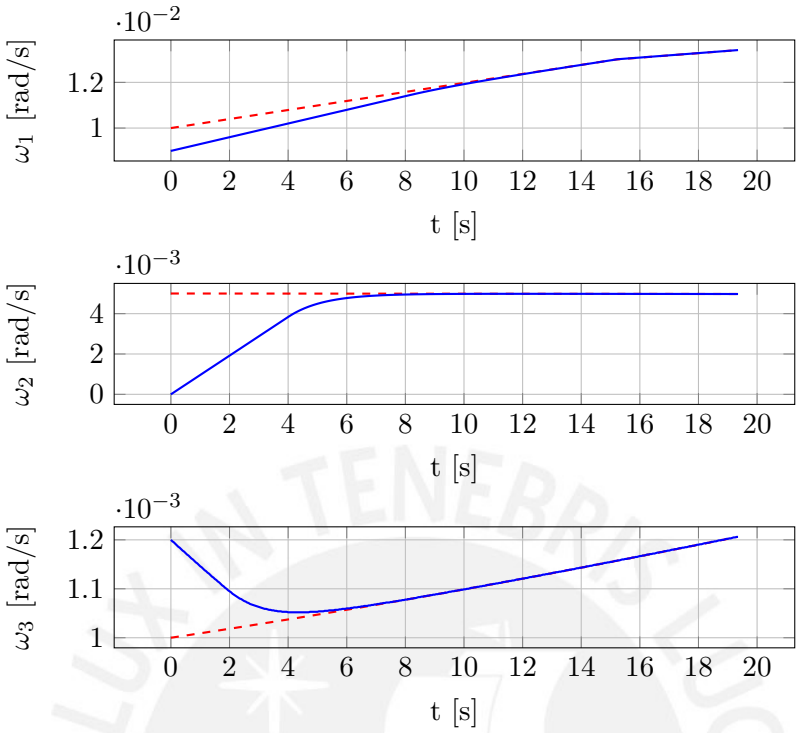


Figure 6.12 – Desired and computed angular velocities for the satellite model

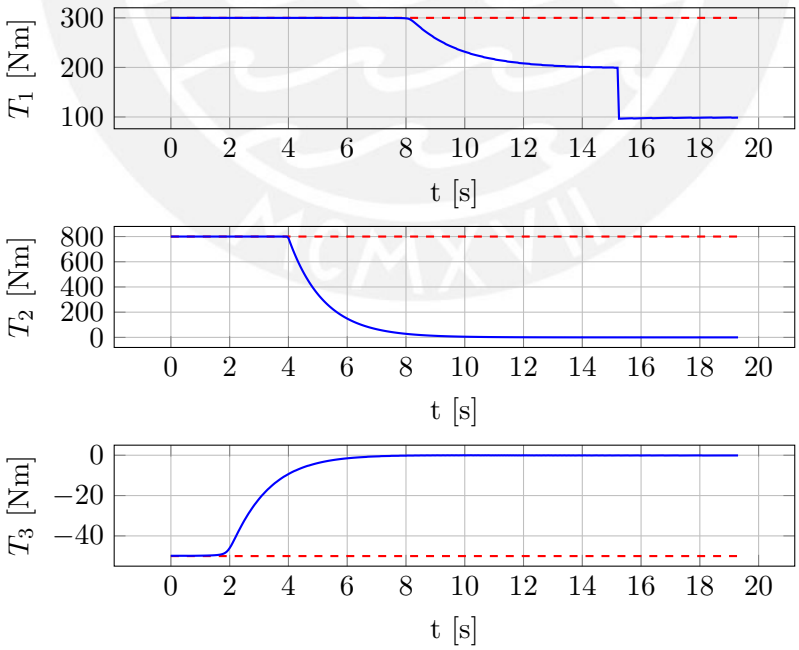


Figure 6.13 – Input torques for the satellite model

For this case study, the LMPC algorithm was implemented using double precision floating-point representation, as the on-line preconditioning employed was not sufficient to obtain satisfactory results using single precision. Figure 6.14 shows the computation time required for prediction horizon values of: $N = 3,\ 4,\ 6,\ 8$. The computation time for this case exceeds the sampling period for prediction horizon values greater than 6. Resource utilization, timing and power consumption achieved after implementation, for both single and double precision floating-point are summarized in table 6.9 for $N = 4$, where latency and Initiation Interval (II) are given in number of clock cycles. This table shows that the considered performance metrics are almost twice for double as for single precision. As mentioned previously good results are obtained with double precision only.

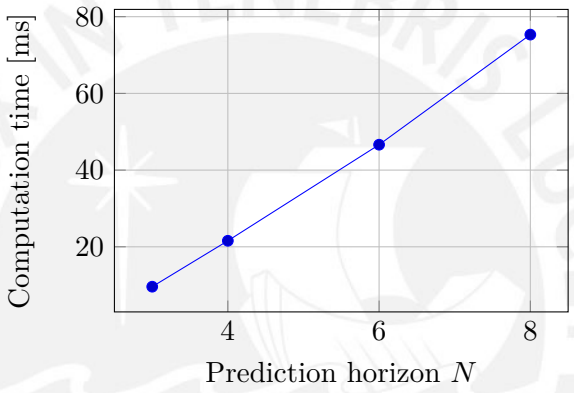


Figure 6.14 – Computation times for the satellite model problem.

Table 6.9 – Metrics for satellite model after implementation, $N = 4$

Metric	Value (double)	Value (float)
Clock Period	10 ns	10 ns
Max. Latency	2157119	1186388
Max. II	2157120	1186389
Computation Time	21.6 ms	11.9 ms
BRAM (%)	22	10
DSP48E (%)	56	27
FF (%)	40	21
LUT (%)	74	50
LUTRAM (%)	8	15
Total On-Chip Power	7531 W	4634 W

6.1.5 Comparison Between FPGA and Pure Software Implementation

The IPM solvers implemented as hardware accelerators on the PL part of the Zynq® device run at 100 MHz, at a clock speed of 10 ns. The same C code used for the implementation in Vivado HLS was then compiled and tested on a laptop computer running Windows®, with an Intel® Core™ i7-5500U processor running up to 2.4 GHz, with 8GB of RAM. A comparison of the time required for an IPM iteration for different prediction horizons for the case studies considered is shown in figure 6.15, where bike 1 and bike 2 refer to the three-state and five-state models respectively. These results show that the time the hardware accelerator requires to complete an IPM iteration is approximately twice the time required by the laptop computer; considering that the computer is running at more than 20 times the frequency of the FPGA, the degree of parallelization achieved can be considered to be approximately 10 times. The lower frequency implies a lower power consumption [49], which is significant for embedded applications, especially for battery driven systems.

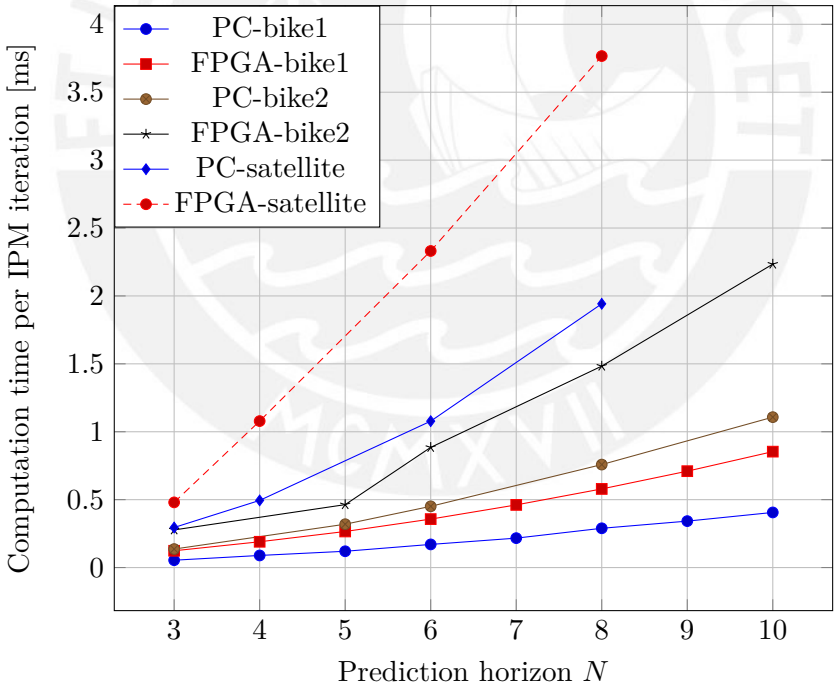


Figure 6.15 – Comparison of computation times on FPGA and a PC.

The implemented LMPC algorithm was also tailored to run entirely on the PS of the Zynq® device, namely on the ARM® Cortex® -A9 processor which can run up to 1 GHz. Figure 6.16 shows the computation times for each IPM iteration for the implementation

of the three-states bicycle model case study when targeting a laptop PC, the FPGA and the ARM® processor. As can be seen, the computation time for the ARM® processor is much higher, approximately 8 times higher than that for the FPGA, which shows the achieved parallelization and the advantage of the hardware implementation versus a pure software implementation.

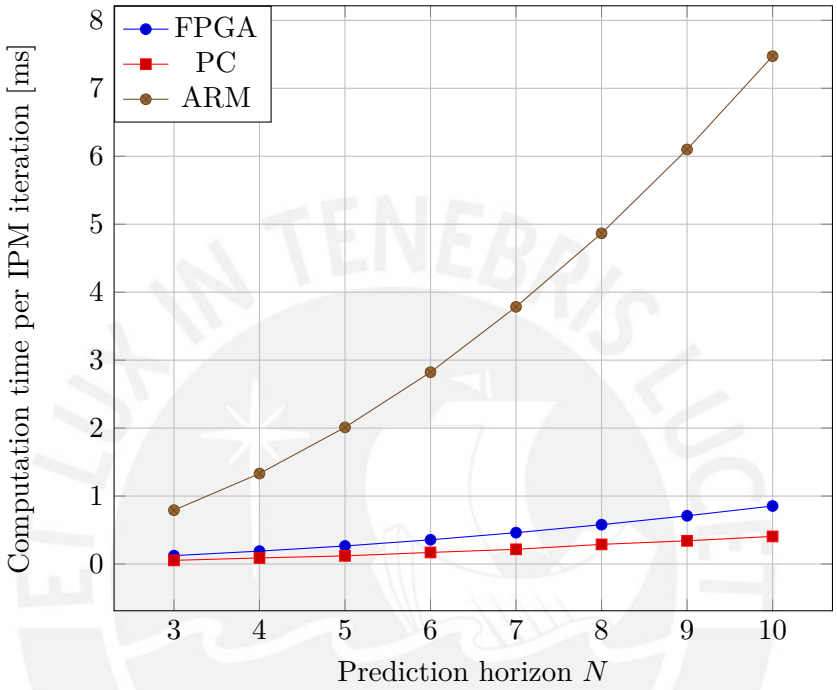


Figure 6.16 – Computation times for three-state bicycle model problem implemented on FPGA, PC and ARM® processor.

6.2 Case Studies for NMPC

For the case of NMPC, no linearization is required and therefore only the reference trajectory corresponding to the states is necessary.

6.2.1 Three-state Bicycle Model

The same three-state bicycle model presented in section 6.1.2 was considered, with nonlinear equations (6.6). The use of fourth order Runge-Kutta method for the solution of the IVP for generation of the NLP problem allows to increment the sampling period from 0.01 to 0.05 seconds, as this method is more accurate than the Euler method.

Parameters used for the simulation are as shown in table 6.10. Weight matrices shown in (6.22) were considered.

Table 6.10 – Parameters for NMPC implementation of bicycle like model

Parameter	Value
l	0.2 m
T_s	0.05 s
Prediction horizon (N)	3
IPM iterations	14
State constraints	$-\pi \leq \theta \leq \pi$ [rad]
Input constraints	$-1.5 \leq v \leq 1.5$ [m/s]
	$-0.65 \leq \varphi \leq 0.65$ [rad]
Initial state	$x_0 = [-1 \quad -1 \quad 0]^T$
Initial reference state	$x_0 = [0 \quad 0 \quad \pi/2]^T$

$$
\begin{aligned}
Q_i &= \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{, for } i = 1, 2, ..., N-1\,, \\
R_i &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \quad \text{, for } i = 0, 1, ..., N-1\,, \\
P &= 20Q_1\,.
\end{aligned}
\tag{6.22}
$$

The same trajectory is used as in the LMPC case for comparison purposes. Figure 6.17 shows the reference trajectory in dotted red line, the calculated trajectory in solid blue and the trajectory obtained for the LMPC approach in green for comparison. Resource utilization, timing, computation time and power consumption achieved after implementation are summarized in table 6.11, where latency and II are given in number of clock cycles. Computation time for this case is six times higher than the one achieved for LMPC when considering the same prediction horizon. The results obtained after synthesis in Vivado HLS for $N = 5$ are also shown in table 6.11, as can be seen, the resources of the target FPGA are not sufficient and in fact four times more LUTs would be required in comparison with the case when $N = 3$; this precluded the implementation and therefore only the metrics after synthesis are shown for this case. This high increment in resources utilization is due to the number of operations required for the calculation of the Jacobian and Hessian matrices to be evaluated at every iteration of the IPM algorithm, which increases with the size of the problem. Attempts to introduce optimization directives to the functions obtained with CasADi increased the minimal achievable clock period significantly, forcing a sequential execution of these functions,

which is not efficient in an FPGA implementation as the achievable clock frequency is much lower than in traditional processors.

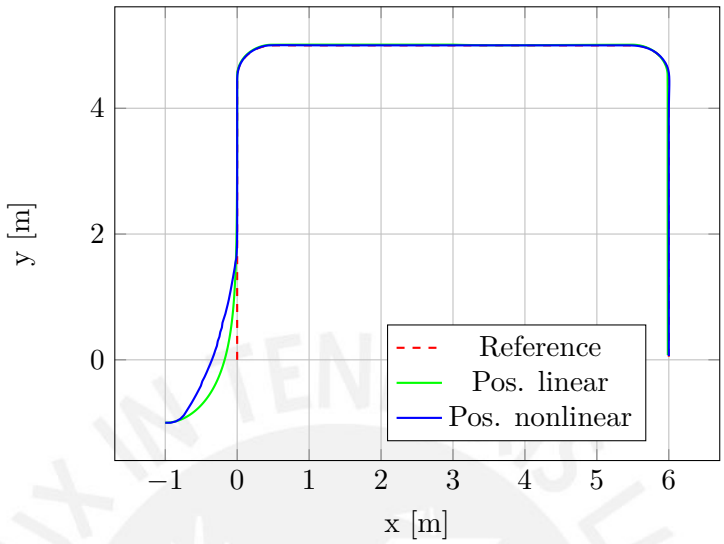


Figure 6.17 – Trajectory in the X-Y plane followed by the vehicle model

Table 6.11 – Metrics for satellite model after implementation (N=3) and after synthesis (N=5)

Metric	Value ($N = 3$)	Value ($N = 5$)
Clock Period	10 ns	11 ns
Max. Latency	1203277	3060077
Max. II	1203278	3060078
Avg. Computation Time	9 ms	-
BRAM (%)	12	26
DSP48E (%)	46	97
FF (%)	20	39
LUT (%)	53	209
LUTRAM (%)	6	-
Total On-Chip Power	5301 W	-

6.3 Summary

This chapter presented the results obtained with the implemented algorithms for LMPC and NMPC tested on trajectory tracking problems for mobile vehicle systems. This application requires the desired trajectory to be known a priori or be calculated on-line for

the corresponding prediction horizon. The linearization approach has the disadvantage that not only the reference states but also the reference inputs must be given, but require less computation time for the solution and resources and power consumption are significantly lower. Highly nonlinear systems may require an NMPC approach, the results obtained show that the current implementation fails to deal with problems bigger than that considered for this thesis, so further research is needed to improve computation time and area utilization.

Chapter 7

Conclusions and Future Work

This thesis focuses on the implementation of MPC algorithms using high-level synthesis on an FPGA for hardware acceleration. Computational burden is significant for the on-line implementation of MPC. When systems with high sampling rates are considered the problem must be solved efficiently within the available time and additionally meeting any given resource, cost, and power consumption constraints. The optimization problem that arises from the MPC formulation is usually solved with IPMs or ASMs. The main advantage of IPM, as opposed to ASM, is the little variations in the problem structure, which is important for a priori FPGA resource allocation.

For the case of LMPC, a primal-dual IPM algorithm was tailored for the solution of the QP that arises as a result of the sparse formulation. Sparsity and the fixed well defined structure of the QP matrices were exploited to reduce memory requirements and improve computation time. For the implementation, a linear algebra library was implemented with the compressed matrix storage formats. Parallelization of the operations is achieved through the use of pragmas available with the Vivado® HLS tool. The algorithm was tested on three trajectory tracking case studies. Results show that the use of HLS for FPGA programming of an IPM solver gives promising results, obtaining the solutions within the sampling period considered. However, computation time grows with the problem size and, because of area limitations on the FPGA, further parallelization to reduce latency is not possible. The choice of setting the number of iterations for both the IPM and MINRES algorithms to a fixed number proved good results, having a significant impact on the computation time. These values must be determined through an off-line implementation. Numerical precision also plays an important role in embedded applications as a reduced demand for precision (number of bits) implies a reduction in computation time, resources utilization, and power consumption. The linear system of equations solved at every MINRES iteration gets ill-conditioned as the solution

is approached, and therefore a wide dynamic range is required; however, the use of preconditioning allows for single precision to be considered as shown in two study cases.

For the case of NMPC, a primal-dual IPM algorithm is also implemented for the solution of the NLP problem using the simultaneous approach. The algorithm was tailored to the trajectory tracking case study, assuming the same cost function and inequality constraints as in the linear case. Only the equality constraints introduce nonlinearities to the optimization problem. A fourth-order Runge-Kutta method was used to turn the OCP problem into an NLP problem. Jacobian and Hessian matrices required for the IPM were obtained from functions generated by CasADi tool. A CCS format was considered for the primal-dual system to be solved by the MINRES algorithm. Results showed that a much greater computation time and resources utilization is required for the solution of the NLP as compared to the QP problem, in great part because of the computation of derivative information at each iteration of the IPM. Further research is needed to improve the current implementation for its use in a wider range of problems.

Results presented in this work proved that LMPC can be successfully implemented on FPGA-based systems. However, further improvements can still be made on the implemented design to improve computation time and reduce resources utilization and energy consumption. One important factor, for example, is the use of low-precision data representation, e.g. based on fixed-point arithmetic, for the solution of the QP problem. Another important factor which has a great influence on the quality of the results is the use of on-line and off-line preconditioning. Some works have already been made regarding these topics but further research is still needed. Further research is also needed for efficient implementation of NMPC on FPGA-based systems.

Regarding the work on this thesis, communication between PL and PS on the Zynq® device should be further studied to allow for 64 bits data streaming. This would allow to exploit the architecture of the Zynq® device for the NMPC problem, in order to accelerate only the MINRES algorithm on hardware, which represents the main computational burden, leaving the other operations to the ARM® controller and in this way to improve latency.

Parallelization of the algorithm itself, dividing the operations on parallel solvers running on the FPGA as well as exploring new multi-core hardware accelerator architectures can also be topics of further research. The use of OpenCL framework appears as a good possibility, using a CPU as host and one or more FPGAs or other accelerators like GPUs or DSPs for task parallelism.

List of Acronyms

ACP	Accelerator Coherency Port
AP SoC	All Programmable System-on-a-Chip
APU	Application Processing Unit
ASIC	Application-Specific Integrated Circuit
ASM	Active-Set Method
AXI	Advanced eXtensible Interface
BDF	Backward Differentiation Formula
BRAM	Block RAM
CCS	Compressed Column Storage
CDS	Compressed Diagonal Storage
CLB	Configurable Logic Block
CRS	Compressed Row Storage
DMA	Direct Memory Access
DSP	Digital Signal Processor
EMIO	Extended Multiplexed Input/Output
FF	Flip-Flop
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GPU	Graphics Processor Unit

HDL	Hardware Description Language
HLS	High-Level Synthesis
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
II	Initiation Interval
IOB	Input/Output Block
IP	Intellectual Property
IPM	Interior-Point Method
IVP	Initial Value Problem
KKT	Karush-Kuhn-Tucker
LMPC	Linear Model Predictive Control
LUT	Lookup Table
MIMO	Multiple-Input-Multiple-Output
MINRES	MINimum RESidual
MPC	Model Predictive Control
NLP	Nonlinear Programming
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
PL	Programmable Logic
PS	Processing System
QP	Quadratic Programming
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read-only Memory
RTL	Register Transfer Level
SQP	Sequential Quadratic Programming

List of Figures

List of Tables

Bibliography

[1] Abbas, M. A. (2011). Non-linear model predictive control for autonomous vehicles. Master's thesis, University of Ontario Institute of Technology, Ontario, Canada.

[2] Allgöwer, F. and Zheng, A. (2012). *Nonlinear model predictive control*, volume 26. Birkhäuser.

[3] Andersson, J., Åkesson, J., and Diehl, M. (2012). Casadi: A symbolic package for automatic differentiation and optimal control. In *Recent advances in algorithmic differentiation*, pages 297–307. Springer.

[4] Bagni, D., Di Fresco, A., Noguera, J., and Vallina, F. (2016). A zynq accelerator for floating point matrix multiplication designed with vivado hls. Technical report, Technical Report# XAPP1170. Xilinx.

[5] Barrett, R., Berry, M. W., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. (1994). *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam.

[6] Basterretxea, K. and Benkrid, K. (2011). Embedded high-speed model predictive controller on a fpga. In *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, pages 327–335. IEEE.

[7] Bellman, R. (1957). Dynamic programming. *Princeton, USA: Princeton University Press*, 1(2):3.

[8] Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.

[9] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA.

[10] Betts, J. T. (1998). Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207.

[11] Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*. SIAM.

[12] Bleris, L. G., Vouzis, P. D., Arnold, M. G., and Kothare, M. V. (2006). A co-processor fpga platform for the implementation of real-time model predictive control. In *American Control Conference, 2006*, pages 6–pp. IEEE.

[13] Center, I. K. Compressed-diagonal storage mode. Available at `http://www.ibm.com/support/knowledgecenter/SSFHY8_5.3.0/com.ibm.cluster.essl.v5r3.essl100.doc/am5gr_cdst.htm`. [Accessed: October 10, 2016].

[14] Che, S., Li, J., Sheaffer, J. W., Skadron, K., and Lach, J. (2008). Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101–107. IEEE.

[15] Chen, H., Xu, F., and Xi, Y. (2012). Field programmable gate array/system on a programmable chip-based implementation of model predictive controller. *IET control theory & applications*, 6(8):1055–1063.

[16] Chen, X. and Wu, X. (2011). Design and implementation of model predictive control algorithms for small satellite three-axis stabilization. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 666–671. IEEE.

[17] Ciletti, M. D. (2003). *Advanced digital design with the Verilog HDL*. Prentice Hall Upper Saddle River.

[18] Corporation, I. Industry solutions. Available at `https://www.altera.com/solutions/industry.html?_ga=1.232792025.657609052.1486913302`. [Accessed: January 10, 2017].

[19] Correa Córdova, M. L. (2016). High Performance Implementation of MPC Schemes for Fast Systems. Master's thesis, Ilmenau University of Technology.

[20] Coussy, P. and Takach, A. (2009). Guest editors' introduction: Raising the abstraction level of hardware design. *IEEE Design & Test of Computers*, 26(4):4–6.

[21] Crockett, L. H., Elliot, R. A., Enderwitz, M. A., and Stewart, R. W. (2014). *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media.

[22] Deuflhard, P. and Bornemann, F. (2002). *Scientific computing with ordinary differential equations*, volume 42. Springer Science & Business Media.

[23] Findeisen, R. and Allgöwer, F. (2002). An introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, volume 11, pages 119–141. Technische Universiteit Eindhoven Veldhoven Eindhoven, The Netherlands.

[24] Fischer, B. (1996). *Polynomial based iteration methods for symmetric linear systems*. Springer.

[25] Grüne, L. and Pannek, J. (2011). Nonlinear model predictive control: Theory and algorithms.

[26] Hartley, E. N., Jerez, J. L., Suardi, A., Maciejowski, J. M., Kerrigan, E. C., and Constantinides, G. A. (2014). Predictive control using an fpga with application to aircraft control. *IEEE Transactions on Control Systems Technology*, 22(3):1006–1017.

[27] Jerez, J. L. (2013). *Custom optimization algorithms for efficient hardware implementation*. PhD thesis, Imperial College London.

[28] Jerez, J. L., Constantinides, G. A., and Kerrigan, E. C. (2011). An fpga implementation of a sparse quadratic programming solver for constrained predictive control. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 209–218. ACM.

[29] Jerez, J. L., Constantinides, G. A., and Kerrigan, E. C. (2012a). Fixed point lanczos: Sustaining tflop-equivalent performance in fpgas for scientific computing. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 53–60. IEEE.

[30] Jerez, J. L., Constantinides, G. A., and Kerrigan, E. C. (2012b). Towards a fixed point qp solver for predictive control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 675–680. IEEE.

[31] Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., and Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251.

[32] Johansen, T. A., Jackson, W., Schreiber, R., and Tondel, P. (2007). Hardware synthesis of explicit model predictive controllers. *IEEE Transactions on Control Systems Technology*, 15(1):191–197.

[33] Junkins, J. L. and Turner, J. D. (1980). Optimal continuous torque attitude maneuvers. *Journal of Guidance, Control, and Dynamics*, 3(3):210–217.

[34] Kerrigan, E. C., Jerez, J. L., Longo, S., and Constantinides, G. A. (2012). Number representation in predictive control. *IFAC Proceedings Volumes*, 45(17):60–67.

[35] Knagge, G., Wills, A., Mills, A., and Ninness, B. (2009). Asic and fpga implementation strategies for model predictive control. In *Control Conference (ECC), 2009 European*, pages 144–149. IEEE.

[36] Künhe, F., Gomes, J., and Fetter, W. (2005). Mobile robot trajectory tracking using model predictive control. In *II IEEE latin-american robotics symposium*.

[37] Lau, M. S., Yue, S., Ling, K., and Maciejowski, J. (2009). A comparison of interior point and active set methods for fpga implementation of model predictive control. In *Control Conference (ECC), 2009 European*, pages 156–161. IEEE.

[38] Leuer, M. and Bocker, J. (2014). Real-time implementation of an online model predictive control for ipmsm using parallel computing on fpga. In *Power Electronics Conference (IPEC-Hiroshima 2014-ECCE-ASIA), 2014 International*, pages 346–350. IEEE.

[39] Leung, B., Wu, C.-H., Memik, S. O., and Mehrotra, S. (2010). An interior point optimization solver for real time inter-frame collision detection: Exploring resource-accuracy-platform tradeoffs. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 113–118. IEEE.

[40] Ling, K.-V., Wu, B. F., and Maciejowski, J. (2008). Embedded model predictive control (mpc) using a fpga. *IFAC Proceedings Volumes*, 41(2):15250–15255.

[41] Ling, K. V., Yue, S. P., and Maciejowski, J. M. (2006). A fpga implementation of model predictive control. In *2006 American Control Conference*, pages 6 pp.–.

[42] Liu, C., Chen, W.-H., and Andrews, J. (2010). Optimisation based control framework for autonomous vehicles: Algorithm and experiment. In *Mechatronics and Automation (ICMA), 2010 International Conference on*, pages 1030–1035. IEEE.

[43] Liu, J., Peyrl, H., Burg, A., and Constantinides, G. A. (2014). Fpga implementation of an interior point method for high-speed model predictive control. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–8. IEEE.

[44] Longo, S., Kerrigan, E. C., and Constantinides, G. A. (2013). A predictive control solver for low-precision data representation. In *Control Conference (ECC), 2013 European*, pages 3590–3595. IEEE.

[45] Maciejowski, J. M. (2002). *Predictive control: with constraints*. Pearson education.

[46] Martin, G. and Smith, G. (2009). High-level synthesis: Past, present, and future. *IEEE Design & Test of Computers*, 26(4):18–25.

[47] Meeus, W., Van Beeck, K., Goedemé, T., Meel, J., and Stroobandt, D. (2012). An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, 16(3):31–51.

[48] Mermet, J. (1993). *Fundamentals and Standards in Hardware Description Languages*. Springer Science & Business Media.

[49] Miyoshi, A., Lefurgy, C., Van Hensbergen, E., Rajamony, R., and Rajkumar, R. (2002). Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35–44. ACM.

[50] Mousavi, M. A., Heshmati, Z., and Moshiri, B. (2013). Ltv-mpc based path planning of an autonomous vehicle via convex optimization. In *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, pages 1–7. IEEE.

[51] Muske, K. R. and Rawlings, J. B. (1993). Model predictive control with linear models. *AIChE Journal*, 39(2):262–287.

[52] Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

[53] Oyerele Sunday, S. (2013). A Study of Model Predictive Control Schemes for Autonomous Ground Vehicles and Implementations. Master's thesis, Ilmenau University of Technology.

[54] Paige, C. C. and Saunders, M. A. (1975). Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629.

[55] Perry, D. L. (2002). *VHDL: programming by example*. McGraw-Hill.

[56] Qin, S. J. and Badgwell, T. A. (2000). An overview of nonlinear model predictive control applications. In *Nonlinear model predictive control*, pages 369–392. Springer.

[57] Qin, S. J. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764.

[58] Rao, C. V., Wright, S. J., and Rawlings, J. B. (1998). Application of interior-point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757.

[59] Rees, T. and Greif, C. (2007). A preconditioner for linear systems arising from interior point optimization methods. *SIAM Journal on Scientific Computing*, 29(5):1992–2007.

[60] Richalet, J., Rault, A., Testud, J., and Papon, J. (1978). Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14(5):413–428.

[61] Robinett, R., Wilson, D., Eisler, G., and Hurtado, J. (2005). *Applied Dynamic Programming for Optimization of Dynamical Systems:*. Advances in Design and Control. Society for Industrial and Applied Mathematics.

[62] Shahzad, A., Kerrigan, E. C., and Constantinides, G. A. (2010). A fast well-conditioned interior point method for predictive control. In *49th IEEE Conference on Decision and Control (CDC)*, pages 508–513. IEEE.

[63] Tamimi, J. (2011). *Development of Efficient Algorithms for Model Predictive Control of Fast Systems*. PhD thesis, Technische Univertsität Ilmenau.

[64] Vahid, F. and Givargis, T. (1999). Embedded system design: A unified hardware/-software approach. *Department of Computer Science and Engineering University of California*.

[65] Verilog.com. Ieee p1364-2005. Available at `http://www.verilog.com/IEEEVerilog.htmll`. [Accessed: December 7, 2016].

[66] Vouzis, P. D., Bleris, L. G., Arnold, M. G., and Kothare, M. V. (2009). A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5):1006–1017.

[67] Wills, A., Mills, A., and Ninness, B. (2011). Fpga implementation of an interior-point solution for linear model predictive control. *IFAC Proceedings Volumes*, 44(1):14527–14532.

[68] Wills, A. G., Knagge, G., and Ninness, B. (2012). Fast linear model predictive control via custom integrated circuit architecture. *IEEE Transactions on Control Systems Technology*, 20(1):59–71.

[69] Wright, M. (2005). The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1):39–56.

[70] Wright, S. J. (1997). Applying new optimization algorithms to model predictive control. In *AIChE Symposium Series*, volume 93, pages 147–155. Citeseer.

[71] Xilinx, I. Axi reference guide. Available at `https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf`. [UG761, V13.1, March 7,2011, Accessed: January 25, 2017].

[72] Xilinx, I. Field programmable gate array (fpga). Available at `https://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm`. [Accessed: January 10, 2017].

[73] Xilinx, I. Function inlining. Available at `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/sdsoc_doc/topics/calling-coding-guidelines/concept_function_inlining.html`. [Accessed: January 25, 2017].

[74] Xilinx, I. Introduction to fpga design with vivado high-level synthesis. Available at `http://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf`. [UG998, v1.0, July 2, 2013, Accessed: November 6, 2016].

[75] Xilinx, I. Loop pipelining and loop unrolling. Available at `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/sdsoc_doc/topics/calling-coding-guidelines/concept_pipelining_loop_unrolling.htmll`. [Accessed: January 24, 2017].

[76] Xilinx, I. Vivado design suite user guide, high-level synthesis. Available at `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug902-vivado-high-level-synthesis.pdf`. [UG902, v2012.2, July 25, 2012, Accessed: December 7, 2016].

[77] Xilinx, I. Vivado design suite user guide, using the vivado ide. Available at `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/ug893-vivado-ide.pdf`. [UG893, v2014.3, October 10, 2014, Accessed: December 20, 2016].

[78] Xilinx, I. Zynq-7000 all programmable soc family product tables and product selection guide. Available at `https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf`. [XAPP1170 (v2.0) January 21, 2016, Accessed: December 19, 2016].

[79] Xu, F., Chen, H., Gong, X., and Mei, Q. (2016). Fast nonlinear model predictive control on fpga using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, 63(1):310–321.

[80] Yang, N., Li, D., Zhang, J., and Xi, Y. (2012). Model predictive controller design and implementation on fpga with application to motor servo system. *Control Engineering Practice*, 20(11):1229–1235.

[81] Zapata, R., Thevenon, J., Perrier, M., Pommier, E., and Badi, E. (1989). Path planning and trajectory planning for non holonomic mobile robots. In *Intelligent Robots and Systems' 89. The Autonomous Mobile Robots and Its Applications. IROS'89. Proceedings., IEEE/RSJ International Workshop on*, pages 323–330. IEEE.