Hardware Implementation of a Model Predictive Controller for Hybrid Systems

By

Eng. Mohamed Fatouh Mahmoud Fouda

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the
Faculty of Engineering at Cairo University
In Partial Fulfillment of the
Requirement for the Degree of
MASTER OF SCIENCE
In
ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

January 2011

Hardware Implementation of a Model Predictive Controller for Hybrid Systems

By

Eng. Mohamed Fatouh Mahmoud Fouda

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the

Faculty of Engineering at Cairo University

In Partial Fulfillment of the

Requirement for the Degree of

MASTER OF SCIENCE

In

ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of

Prof. Dr. Mohamed Abuelseod Sultan

Electronics and Communications Engineering Department

Faculty of Engineering, Cairo University

Associate Prof. Dr. Hossam Ali Fahmy Elsayed

Electronics Engineering Department

Faculty of Engineering

Cairo University

Dr. Hany Mohamed

Electronics Engineering Department

Faculty of Engineering

Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

January 2011

Hardware Implementation of a Model Predictive Controller for Hybrid Systems

By

Eng. Mohamed Fatouh Mahmoud Fouda

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the

Faculty of Engineering at Cairo University

In Partial Fulfillment of the

Requirement for the Degree of

MASTER OF SCIENCE

In

ELECTRONICS AND COMMUNICATIONS ENGINEERING

Approved by the

Examining Committee

Prof. Dr. Abd-Elmonem Wahdan

____

Prof. Dr. Serag. E.-D Habib

____

Associate Prof. Dr. Hossam Ali Fahmy

____

Prof. Dr. Mohamed Sultan **Thesis Main Advisor**

____

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

January 2011

Table of Contents

List of Tables

List of Figures

List of Symbols

Numerical sets

Notation	Meaning
$\Re$	Set of scalar real numbers
$\Re^n$	Set of real vectors of length n
$\Re^{m\times n}$	Set of real matrices with dimension m×n

Linear state space

Notation	Meaning
x(k)	State vector at instant k
u(k)	Input (control action) vector at instant k
y(k)	Output vector at instant k
$\underline{y}$	Vector of predicted outputs up to time T
$\underline{u}$	Vector of future control inputs up to time T
M	System matrix
N	Input matrix
C	Output matrix
F	Matrix relating $\underline{y}$ to state x(k)
H	Matrix relating $\underline{y}$ to state $\underline{u}$

Mixed Logical Dynamical systems

Notation	Meaning
k	Current sampling instant
x(k)	State vector at instant k
u(k)	Input (control action) vector at instant k
y(k)	output vector at instant k
z(k)	auxiliary real variables vector at instant k
δ(k)	auxiliary binary variables vector at instant k
A_x	System dynamics matrix (associated with states)
B_1, B_2, B_3	System dynamics matrices (associated with input, auxiliary binary variables, auxiliary . real variables respectively).
E_1, E_2, E_3, E_4	MLD constraint matrices (associated with input, auxiliary binary variables, auxiliary real variables, states respectively)
E_5	MLD constraints constant term vector.
n_c, n_b, n_x	Number of states (continuous, binary , total)
p_c, p_b, p_y	Number of outputs (continuous, binary , total)

m_c, m_b, m_u	Number of inputs (continuous, binary , total)
r_c, r_b	Number of auxiliary (continuous , binary) variables

Model predictive control

Notation	**Meaning**
J	Control objective function
Q_x, Q_u, Q_y, Q_z	Objective function weighting matrices for (states, input, output, auxiliary real variables)
Q_{xT}	weighting matrix for final state xT
$\lVert . \rVert^2$	norm 2
T	prediction horizon (MPC) Or final time (optimal control)
T_u	Control horizon
y_{ref}, x_{ref}	Required output , states references respectively
q	optimization vector
$\bar{G}$	quadratic cost matrix
$\bar{d}$	linear cost matrix
$\bar{D}$	parameter – optimization variable cost matrix
$\bar{U}$	quadratic parameter cost matrix
$\bar{V}$	linear parameter cost matrix
$\bar{f}$	objective function constant term
$\bar{A}$	constraints LHS matrix
$\bar{b}$	constraints RHS matrix
$\bar{C}_x$	states weighting matrix in constraints RHS
Θ	parameter vector

Interior point method

Notation	**Meaning**
q	Optimization vector
G	Quadratic cost matrix
d	Linear cost matrix
A	Inequality Constraints LHS matrix
b	Inequality Constraints RHS matrix
n	Number of optimization variables
m	Number of constraints
λ	Lagrange multipliers
s	Slack variables

Λ	Diag(λ)
S	Diag(s)
Δq, Δs, $\Delta\lambda$	Search direction of q, s and λ respectively
α	Step length
σ	Centering parameter
μ	Duality measure
q^+, s^+, λ^+	New iteration value
$\mathcal{N}_{-\infty}$	One sided infinity neighborhood
γ	Parameter of the one sided infinity neighborhood

Mixed integer quadratic programming

Notation	**Meaning**
q	Optimization vector
G	Quadratic cost matrix
d	Linear cost matrix
A	Inequality Constraints LHS matrix
b	Inequality Constraints RHS matrix
A_{eq}	equality Constraints LHS matrix
b_{eq}	equality Constraints RHS matrix
n_{bin}	number of binary optimization variables
n_{real}	number of real optimization variables
n	Total number of optimization variables
m	Total number of constraints

Stick-Slip Inertial Drive (SSID)

Notation	**Meaning**
m_s	SSID small mass
M_m	SSID main mass
k_s	Spring constant of SSID transmission elements
u_p	Force of piezoelectric element
Fc	Coulomb friction force
x_{ms}	Displacement of small mass
x_{Mm}	Displacement of main mass
rx_M	Reference of main mass position
v_{ms}	Speed of small mass
v_{Mm}	Speed of main mass
m	Total number of constraints
ρ	Friction force at interface of two masses

List of Abbreviations

ASIC	Application Specific Integrated Chip
FPGA	Field Programmable Gate Array
HYSDEL	HYbrid System DEscription Language
MLD	Mixed Logical Dynamical
MPC	Model predictive Control
MBQP	Mixed Binary Quadratic Programming
MIQP	Mixed Integer Quadratic Programming
QP	Quadratic Programming
SSID	Stick Slip Inertial Drive

Abstract

The area of hybrid systems modeling and control had great research interest in recent years. Hybrid systems are defined as systems combining both continuous dynamics and discrete events. Such systems appear in different applications like robotic systems and automotive applications. Hybrid systems can be used to model various types of systems like systems with switching dynamics, systems interacting with logic elements or systems with non-linearities like saturation and dead-zone.

One of the most important achievements of research in this area is arrival to a standard modeling framework for hybrid systems. Tools were provided to automate the process of modeling hybrid systems. These tools provided also a standard approach to control hybrid systems using optimal control and model predictive control. Some simulation abilities were provided to simulate the closed loop performance of the controlled hybrid system.

Model Predictive Control (MPC) is one of the most successful control techniques that can be used with hybrid systems. MPC was applied with great success on linear systems and it has many industrial applications. Recently, there were some efforts to use MPC for embedded systems and system-on-chip applications. In this thesis, we consider the application of MPC to hybrid embedded systems. The main barrier is the complexity of system as it requires online solution of a mixed binary optimization problem in each sampling instant. This problem limits the domain of application to very slow systems. We work to overcome this barrier by designing a fast hardware mixed binary quadratic solver. This solver will be used to solve the required optimization problem and calculate the optimal control action. Having a high speed solver will make it possible to apply MPC on hybrid embedded systems with sampling time in range of few milliseconds. To achieve the required speed, the concepts of pipelining and parallel design are used. A study is performed to measure the speed improvement achieved by applying these concepts.

Finally, to evaluate the controller performance, the implemented controller is used to control a practical hybrid system. The controlled hybrid system is the Stick-Slip Inertial drive. This application showed that the proposed design enabled MPC to be applied successfully to control hybrid systems with relatively small sampling time.

Chapter 1 Introduction and Background

1.1 Introduction

This thesis studies the control problem of a special class of systems called Hybrid Systems. Hybrid systems are generally defined as systems that combine both continuous dynamics and discrete events. Systems with multiple switching dynamics represent a main class of hybrid systems, in which the system operates in multiple modes. Each mode has different dynamics. The operating mode is selected due the achievement of a certain conditions. Conditions that alter the system mode are discrete events, while the system operates –in a certain mode– according to continuous dynamics. This hybrid nature of such systems makes them complex to model and control [1].

Hybrid systems may originally have this hybrid nature like mechanical systems that distinguish between stick and slip friction. Hybrid systems may also result from the control of continuous systems using digital logic. Examples of this type of hybrid systems include: temperature control using thermostat, hydraulic systems with on-off pumps or valves and systems controlled by nonlinear elements like saturation and dead-zone [2]. Hybrid systems can be also used the operation systems controlled by digital commands.

Hybrid systems appear in various applications like robotic systems and automotives. In such applications, systems should be controlled to achieve the required performance. However due to the hybrid nature of the system, reaching a systematic modeling and control technique has represented a challenge.

There were many trials to provide modeling frameworks for hybrid systems [3]. However, most of these modeling techniques ware based on the combination of logic rules to describe discrete events and algebraic expressions to describe continuous dynamics. Hence, these modeling techniques did not allow for a systematic control approach. Efforts in the direction resulted in the proposal of a unified modeling framework for hybrid systems. It was called Mixed-Logical dynamical (MLD) modeling [4]. In this modeling framework, only algebraic expressions were used to describe the complete dynamics of a hybrid system. The discrete events occurring in a hybrid system were described by inequality constraints. These inequality constraints have some of the variables restricted to be binary variables. The use of binary variables allowed the description of logic events available in hybrid systems. The resulting dynamic equations were combining real and binary “logic” variables, hence the name “Mixed Logical Dynamical”. The hybrid nature was maintained by constraining the dynamic equations by a set of inequalities.

The introduction of MLD modeling allowed the proposal of systematic control approaches. However, as the system dynamics contain constrains, the proposed control strategy must be able to deal with constraints. As classical control design techniques like PID and Lead-lag compensation were not able to explicitly handle constraints, they were not suitable for this type of problems. To handle constraints explicitly, Model Predictive Control (MPC) was proposed as a suitable control technique [4].

MPC is one of the most successful control techniques used for linear systems and it has many industrial applications. MPC is a form of closed loop optimal control. It works to achieve required performance by the solution of an optimization problem. This optimization problem works to minimize a certain required objective function while satisfying any constraints on the system operation. The application of MPC requires the availability of a mathematical model to describe system behavior. This model is required to predict the system future behavior. This is required to provide an optimal control signal that guarantees satisfaction of requirements and preventing any possible constraints violation.

As MPC only requirement is the availability of a model to use for prediction, the introduction of MLD modeling allowed the use of MPC to control hybrid systems. However, this extension to hybrid system will result in a harder control problem than the case of linear system. The difficulty is that the generated optimization problem has some of its variable restricted to be binary. This converts the optimization problem to a mixed-binary optimization problem. This type of problem is more complex to solve than optimization problems with real variables. Moreover, its complexity increases exponentially as the number of binary variables increases [5]. This complexity limits the domain of application of MPC to only slow hybrid systems as MPC requires the solution of the generated optimization problem every sampling instant. The problem is more difficult when it is related to hybrid embedded systems, where the use of powerful computers to perform the calculations is not practical.

There were trials to apply MPC to embedded linear systems and system-on-chip applications [6-8]. In this thesis, we consider the use of MPC in hybrid embedded systems. We try to exploit recent developments in hardware technology to design a MPC that is able to control fast embedded hybrid systems. This is achieved by designing a custom hardware to solve the MPC mixed binary optimization problem in a short time to allow its application to fast hybrid systems.

To reach small solution time the concepts of parallel design and pipelining were used in the optimizer design. An ALTERA FPGA was used to prototype the hardware design.

The thesis will be organized as following; the rest of chapter one will provide the necessary background. This background is divided to four main sections. Section 1.2 will give a detailed description of Model Predictive control, its advantages and its basic formulation in the case of linear systems. Then, we discuss in section 1.3 the details of hybrid systems, its applications, the MLD modeling framework and hybrid control using MPC strategy. Sections 1.4 and 1.5 review the optimization problem solution algorithms. Section 1.4 reviews the solution algorithms of Quadratic Programming (QP) problems and section 1.5 reviews the solution algorithms of Mixed Binary Quadratic Programming (MBQP) problems.

In chapter 2 the hardware design ideas and implementation are discussed. The results of the hardware design are given in chapter 3. In chapter 4 a practical hybrid system is proposed. Its modeling is discussed. Then MPC is used to control the system. Finally, chapter 5 concludes the thesis and sheds some light on the future work.

1.2 Model Predictive Control

1.2.1 Introduction

Model predictive control (MPC) is a computer control technique that aims to control a process by online optimization. The control signal generation involves the online solution of an optimization problem. The optimization problem involves the minimization of a cost function that tries to drive the future process outputs towards desired trajectories while respecting any existent constraints. The obtained optimal solution is the required control action to be applied to the process to push its future outputs as near as possible to the required references. To build the control cost function the values of the future process outputs need to be predicted. This Prediction requires the knowledge of the process dynamic model, which explains the name “Model Predictive Control" or MPC in short.

The first published work that introduced the concept of control by online optimization appeared at 1963 [9]. The synthesis of a feedback controller using this idea was reported in [10]. Although these works have not introduced deep theoretical analysis of the proposed ideas, these ideas had found a place in industrial applications. Industry was the main driver of the development and application of MPC technology. The first description of MPC control applications appeared in [11], where MPC was considered as a heuristic control technique.
From the time when MPC has first appeared it was used in many industrial applications. Its implementations had different names according to the providing vendor. Different implementations of MPC had different features and used different modeling techniques to describe industrial process dynamics. A survey of different implementations of industrial MPC and their differences can be found at [12].

Despite the early implementation of MPC, its computational complexity has restricted it to processes with slow dynamics like chemical processes. The successive improvements in electronic systems which led to higher computation capabilities opened the door for application of MPC to systems with faster dynamics to the degree that it could be used for embedded systems that require advanced control strategies.

1.2.2 Advantages over other control techniques

Model predictive control outperforms other control techniques like PID control due to the following reasons:

- Its ability to deal with complex multivariable systems.
- Its ability to explicitly deal with constraints. This feature is highly required by process industry due to process and actuator limitations. Besides, it is known that the optimum economical operating point lies within the intersection of economic constraints.
- Its general nature which can be used to control wide range of systems like time variant and nonlinear systems and systems with time delays.

1.2.3 Basic formulation and methods of solution

The basic formulation of model predictive control for a SISO linear system [13] is to find the control sequence u to minimize the following cost function

$$\min_{u(k|k),\ldots u(k+T-1|k)} \left(\sum_{i=1}^{T} w_{y_i}\left(y(k+i\,|\,k)-y_{ref}(k+i)\right)^2+\sum_{i=1}^{T} w_{u_i} u(k+i-1\,|\,k)^2 \quad (1.1)$$

Subject to:

$$y_{\min} \le y(k+i\,|\,k) \le y_{\max}, i=1,\ldots.T \qquad (1.2a)$$

$$u_{\min} \le u(k+i\,|\,k) \le u_{\max}, i=1,\ldots.T \qquad (1.2b)$$

where :

k	current sampling instant
y (k + i | k)	predicted output at instant k+i calculated at instant k
y_{ref} (k + i)	desired output value at instant k+i
u (k + i -1 | k)	control sequence applied to process at instant k+i-1
T	prediction horizon (time window within which the output is predicted)
w_{y_i}, w_{u_i}	Weighting factors for system output and control input respectively

When optimization problem (1.1) is solved, the whole control sequence is obtained. Only the first control move u (k | k) of the control sequence is applied to the process. Then, new measurements are performed at the following sampling instants. The optimization problem is solved again at each sampling instant to calculate the new required control move.

It is clear that MPC formulation (1.1) requires the prediction of future process outputs. This prediction is performed by the use of model for the process. The final problem formulation differs based on the model used to describe the process. The process can be described by an input-output model like FIR model, or it can be described by a more general state space model or an ARMAX model.

The formulation (1.1) represents the constrained MPC problem which in general doesn't have an explicit solution and needs numerical solutions to obtain the control sequence.

Usually the MPC cost function can be formed into a quadratic form, hence we convert the optimization problem into a quadratic programming problem in the form

$$\min_{q} J(q) \equiv \min_{q} q^T \overline{G} q + \overline{d}^T q + \overline{f} \qquad (1.3a)$$

$$S.T \qquad \overline{A} q \le \overline{b} \qquad (1.3b)$$

Where q is the vector collecting all optimization variables u(k | k) to u(k+T-1 | k), $\overline{G}$ is the quadratic cost matrix, $\overline{d}$ is the linear cost matrix, $\overline{f}$ is the objective function constant term, $\overline{A}$ is the inequality constraints coefficients matrix and finally $\overline{b}$ is the constraints constant terms matrix.

The optimization vector q in (1.3) represents the sequence of future control moves from u(k | k) to u(k+T-1 | k). Constrained Quadratic programming problems in the form (1.3) has several numerical methods of solutions. These methods will be described in detail in section 1.5.

Process modeled by state space model

In this section we will describe in detail the MPC formulation for one of the most common process models which is state space model.

In this case the system is described by the state space equations

$$x(k+1) = Mx(k) + Nu(k) \tag{1.4a}$$

$$y(k) = Cx(k) \tag{1.4b}$$

Hence the predicted output at instant k+i is given by

$$\begin{aligned} y(k+i\,|\,k) &= Cx(k+i) = CMx(k+i-1) + CNu(k+i-1) \\ &= CM(Mx(k+i-2) + Nu(k+i-2)) + CNu(k+i-1)) \\ &= CM^{i}x(k) + CM^{i-1}Nu(k) + CM^{i-2}Nu(k+1) + \ldots\ldots\ldots\ldots + CNu(k+i-1) \end{aligned} \tag{1.5}$$

For the T output predictions

$$\underline{y} = \begin{bmatrix} y(k+1|k) \\ y(k+2|k) \\ \vdots \\ y(k+T\,|\,k) \end{bmatrix} = \begin{bmatrix} CM \\ CM^{2} \\ \vdots \\ CM^{T} \end{bmatrix} x(k) + \begin{bmatrix} CN & 0 & \cdots & 0 \\ CMN & CN & \cdots & 0 \\ \vdots & & \ddots & 0 \\ CM^{T-1}N & \cdots & \cdots & CN \end{bmatrix} \begin{bmatrix} u(k) \\ u(k+1) \\ \\ u(k+T-1) \end{bmatrix} \tag{1.6}$$

or

$$\underline{y} = Fx(k) + H\underline{u}$$

The optimization problem (1.1) can be formed in matrix form as

$$\min_{\underline{u}} J = (\underline{y} - \underline{y}_{ref})^{T} Q_{y} (\underline{y} - \underline{y}_{ref}) + \underline{u}^{T} Q_{u} \underline{u} \tag{1.7}$$

Substituting by (1.6) the objective function J takes the form

$$\begin{aligned} J &= (Fx(k) + H\underline{u} - \underline{y}_{ref})^{T} Q_{y} (Fx(k) + H\underline{u} - \underline{y}_{ref}) + \underline{u}^{T} Q_{u} \underline{u} \\ &= \underline{u}^{T}(Q_{y} + H^{T} Q_{y} H)\underline{u} + 2(Fx(k) - \underline{y}_{ref})^{T} Q_{y} H\underline{u} + (Fx(k) - \underline{y}_{ref})^{T} Q_{y} (Fx(k) - \underline{y}_{ref}) \end{aligned} \tag{1.8}$$

The resulting objective function has the standard form of the quadratic programming problem (1.3). When the optimization problem is unconstrained the control moves can be computed explicitly by

$$\underline{u} = -(Q_{u} + H^{T} Q_{y} H)^{-1} H^{T} Q_{y}^{T} (Fx(k) - \underline{y}_{ref}) \tag{1.9}$$

It is clear that the computed control depends linearly on the value of the current state x(k) then MPC can be considered as a state feedback control. Therefore, the implementation of the MPC algorithm requires the use of a state observer for the deterministic control problems or a state estimator (Kalman Filter) for the stochastic problems.

When the previous optimization problem has constraints, the solution cannot be computed explicitly. There exist a number of numerical techniques to solve this problem [14] like:

1. active set methods
2. interior point
3. feasible point improvements
4. pivoting methods

1.2.4 Explicit Solution of the MPC problem

Explicit solution of the MPC problem [15] is an alternative method to avoid the need of online solution of the QP associated with control problem. In this case the problem is solved offline for all possible state values within a selected range. The different solutions of the problem are stored in a memory as a function of the states. During the online control of the system, the controller only searches the memory to get control gains corresponding to current system states. In this manner, instead of solving complex QP online we only perform search in memory. Explicit MPC has some drawbacks when problem size increases or long prediction horizons are used. Tables I and II in [16] shows that the number of stored solutions in explicit MPC -and hence the required memory-increases exponentially when problem size or number of constraints increases. The large number of stored solutions makes the time required to search for the solution longer than the time needed for the online solution.

1.2.5 MPC in embedded systems

The attractive features of model predictive control recommended it as a reliable control technique for complex embedded systems that requires accurate performance while considering the constraints of manipulated and controlled variables. Examples of such systems include: Robotics, Automotive applications [17], Avionics , Biomedical systems such as Prosthetics[18],[19], Rotational antennas,…etc.

The main consideration for using MPC for embedded applications is the computational complexity of the MPC algorithm and whether it would be fast enough to handle the fast dynamics of such systems.

While the industrial MPC algorithms used for process control are implemented as software running on workstations. The implementations of MPC for embedded applications vary from a software running on an embedded processor as proposed in [6], to a complete hardware realization of the algorithm as proposed in [7]. Also, Hardware / Software co-designs of the algorithm are inspected as in [8]. In these designs the hardware is used to speed up the calculations of the matrix operations required during the solution of the optimization problem.

1.3 Hybrid systems

1.3.1 Introduction

Hybrid systems are systems that combine continuous dynamics and discrete event dynamics. Hybrid systems appear in many areas. The most famous examples of hybrid systems are continuous processes controlled by discrete event controllers like on-off relays and switches. This leads to a system with a number of modes. The continuous dynamics of the system depends on its mode of operation. Thus hybrid systems can be viewed as multi-model systems.

Not all hybrid systems result from the application of digital controllers to continuous systems. There are also systems that are hybrid in nature, like systems that have switching dynamics for example:1) friction models which distinct between stick and slip phases and 2) mechanical systems dynamics switching due to load/no-load variations. Also some nonlinear elements can be considered as linear multimodal systems like saturation and dead-zone.

As hybrid systems appear frequently in practical applications, they have attracted researchers' attention. They began to study the methods for modeling such systems. Such modeling is required for performance evaluation of hybrid systems and afterwards to build control schemes to operate these systems as required.

The complex nature of hybrid systems including interacting continuous dynamics -which can be modeled by differential or difference equations- and discrete event dynamics -which can be modeled by finite automata - complicates the task of modeling the overall hybrid system. The research community has provided multiple modeling schemes of hybrid systems. One of the earliest surveys of different hybrid modeling techniques was presented in [20]. This survey distinguishes four hybrid modeling techniques until the last century mid-nineties which are:

1) Automata and transition systems.
2) Dynamic systems.
3) Algebraic structures models.
4) Programming languages models.

The models described in this survey were mainly used for verification of hybrid systems. The control strategies proposed based on these models were not systematic and applicable only for certain classes of hybrid systems. Better modeling techniques, namely; mixed logical dynamical (MLD) modeling and Piece-Wise Affine modeling (PWA), were proposed in 1999. Equivalence between both models was later proved in [21]. An overview of the recent important hybrid modeling techniques and the design of stabilizing controllers was given in [22].

One of the most formal methods of hybrid systems modeling is the Hybrid Automaton, which can be simply defined as a state machine augmented with differential or difference equations. There are many methods to describe a hybrid automaton; one of the most famous is the Alur-Henzinger hybrid automaton [2], [23].

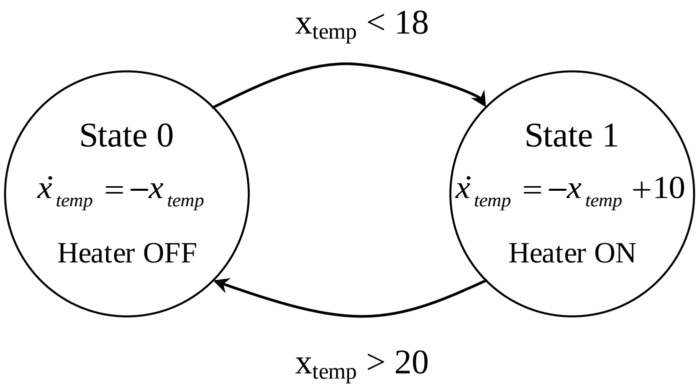


Figure 1.1: Temperature control using thermostat hybrid automaton

In general, hybrid automaton models a hybrid system as a number of logic states, with each logic state associated with continuous dynamics that describe the continuous system behavior in this state. Transitions between states are based on the value of continuous system states, continuous system inputs and discrete system inputs. The system outputs are determined based on the system current state and the continuous dynamics associated with that state. An example hybrid automaton is shown in figure 1.1. It shows the temperature control using thermostats. The system regulates temperature around 20 degree by turning on the

heater is when temperature falls below 18 degree and turns it off when temperature exceeds 22 degree.
A detailed study of hybrid automaton, its basic components and its mathematical formulation is found in [2].

Although the hybrid automaton was a general method to model a wide class of hybrid systems. It was not so much suitable for the control implementation of hybrid systems. This is because hybrid automaton uses both mathematical functions and logic equations to describe the hybrid system. Controlling hybrid systems -on the other hand- requires a unified mathematical model that can be used for performance evaluation and for control synthesis.

This need for a unified framework for hybrid systems modeling and control was addressed by modeling of such systems as Mixed Logical Dynamical (MLD) systems [4].
In this modeling framework propositional logic – used to describe performance of hybrid system - is transformed into linear inequalities involving integer and continuous variables. The resulting MLD system models hybrid system by linear dynamics –difference or differential equations- constrained by linear mixed-integer inequalities.
MLD modeling strategy is not only able to model hybrid systems described by hybrid automata but also able to cover a wide range of systems including constrained linear systems , finite state machines, systems with piece-wise linear dynamics and nonlinear systems whose nonlinearities can be expressed (or properly approximated) as piece-wise linear functions. MLD modeling will be discussed in detail in section 1.3.2.

Before reaching a standard modeling technique for hybrid systems, the task of building a control for a hybrid system was a complex task. This complexity arises from the fact that there were no systematic approaches to build such controllers. At this period, control of hybrid systems was based on heuristic rules that were proposed by experience of the practical plant operation and hence it was highly related to the nature of the hybrid system at hand. Performance validation of the proposed controller was performed by expensive plant tests that consume much time and money.

Modeling hybrid systems as MLD systems opened the door to think for systematic control algorithms for hybrid systems. However, we should note that the resulting models have constraints. These constrains arise from the modeling methodology that converts logic propositions to mixed-integer inequalities and also from the operation constraints on the original system.

Due to this constrained nature of the resulting models, the classical control approaches -such as PID and frequency domain control techniques- won't be suitable. This results from the inability of such techniques to explicitly handle constraints. As a result, optimal control methodologies were thought to be suitable for this type of control problems [4].

As optimal control problems use an objective function which usually takes a quadratic form, optimization problem takes the form of mixed-integer quadratic programming. This problem doesn't have an explicit solution, which forces us to calculate the control sequence off-line and store it to be applied to process during operation. This makes the control scheme an open loop control which has the many disadvantages compared to feedback control. The most important advantage of feedback control is its ability to reject disturbances.

In order to apply optimal control in a feedback fashion, the optimization problem has to be solved online. Control algorithm will use state measurements obtained each sampling instant to form an optimization problem. The problem is solved online and the optimal solution obtained is applied to the system. This procedure is repeated each sampling interval.

The feedback optimal control just described creates optimization problems based on all future system response. Hence, the generated optimization problems are very complex to solve. To simplify the formed optimization problem, we can limit the optimization to only a small window of the system future response. This window is called control horizon. The proposed control technique is called receding horizon control or model predictive control [4].

The control algorithm predicts process performance only during the selected horizon. The predictions obtained are used to calculate the optimal control sequence. Of this sequence, only the first control move is implemented. Then new measurements take place and new optimization iteration is performed based on new predicted response. The MPC approach for hybrid system requires the solution of MIQP problem at each sampling instant. The computational complexity of solution of such a problem is high and it increases exponentially with the number of variables to be optimized. Despite this computational complexity, MPC is considered as one of the most successful control algorithms for hybrid systems. There exist many reported applications where MPC has proved success in the context of hybrid systems [24]. Various studies on the feasibility, optimality, robustness and stability of hybrid systems controlled by MPC exist [25], [26].

1.3.2 MLD systems

In this section we introduce the mathematical formulations of MLD systems [4]. Mixed logical dynamical model integrates system dynamics, logic rules and operation constraints of a system to a unified linear dynamics subject to linear mixed integer inequalities. This modeling can be done by using some basic steps as following:

1- Hybrid system dynamics should be carefully studied. The sources of discrete event dynamics should be distinguished. A group of logic variables - δ - is declared to be used to express the discrete event dynamics. We reach to a form where system dynamics are controlled by some logic rules.
2- The stated logic propositions should be transformed to mixed-integer linear inequalities. This is done based on the propositional calculus and its equivalence to linear inequalities. A complete study of this topic can be found in [27].

3- Some auxiliary states - z - may be introduced to express relations between original system states - x - and the logic variables declared – δ. This results in new state equations which relate original states, auxiliary states and logic variables. Additional linear inequalities may be used to maintain the equivalence between original hybrid system dynamics and resulting MLD system dynamics.

4- The final MLD model is formed by the combination of the resulting linear dynamics –which may include continuous/discrete states, continuous/discrete inputs, continuous/discrete outputs and continuous/discrete auxiliary variables- and all derived linear inequalities.

The general formulation of the MLD system takes the form

$$x(k+1) = A_x x(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) \qquad (1.10a)$$

$$y(k) = Cx(k) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \qquad (1.10b)$$

$$E_2 \delta(k) + E_3 z(k) \le E_1 u(k) + E_4 x(k) + E_5 \qquad (1.10c)$$

where

$x = \begin{bmatrix} x_c \\ x_b \end{bmatrix}, x_c \in \Re^{n_c}, x_b \in \{0,1\}^{n_b}, n_x = n_c + n_b$ is the system states, combining continuous states x_c and logical 0-1 states x_b.

$y = \begin{bmatrix} y_c \\ y_b \end{bmatrix}, y_c \in \Re^{p_c}, y_b \in \{0,1\}^{p_b}, p_y = p_c + p_b$ is the output vector , combining continuous outputs y_c and logical 0-1 ouputs y_b.

$u = \begin{bmatrix} u_c \\ u_b \end{bmatrix}, u_c \in \Re^{m_c}, u_b \in \{0,1\}^{m_b}, m_u = m_c + m_b$ is the control input vector , combining continuous control commands u_c and discrete controls u_b.

$z \in \Re^{r_c}, \delta \in \{0,1\}^{r_b}$ are the continuous and logical (binary) auxiliary variables.

To illustrate this modeling procedure let's consider the example used in [4]

Example 1.1 Consider the system

$$x(k+1)=\begin{cases} 0.8x(k)+u(k) & x(k)\geq 0 \\ -0.8x(k)+u(k) & x(k)<0 \end{cases}$$
$$y(k)=x(k)$$

Where k represents the time kT, where T is the sampling time.

In this system the discrete dynamics come from a condition on the value of the continuous state x(k). This condition will be associated with a logic variable δ(k) such that:.

$$\delta(k)=1 \leftrightarrow x(k)\geq 0$$

The introduction of this logic variable allows us to model the system dynamics by the single equation

$$x(k+1)=1.6\delta(k)x(k)-0.8x(k)+u(k)$$

subject to the logic proposition

$$\delta(k)=1 \leftrightarrow x(k)\geq 0$$

This logic proposition can be transformed into the following inequalities

$$-m_x\delta(k)\leq x(k)-m_x$$
$$-(M_x+\varepsilon)\delta(k)\leq -x(k)-\varepsilon$$

Where ε is a small positive integer to represent measurement precision

And M_x, m_x are defined as

$$M_x=\max_{x\in X} x(k)$$
$$m_x=\min_{x\in X} x(k)$$

Where *X* is the given bounded set that state x belong to.

The term δ(k)x(k) that exist in the new state equation represents a product between continuous and logical variables. This makes the resulting dynamics nonlinear. To overcome this nonlinearity this term can introduce an auxiliary variable $z(k) = \delta(k)x(k)$. the use of this variable will result in a linear state equation. To have a correct modeling of the hybrid system the introduced auxiliary variable should satisfy the following set of inequalities.

$$\begin{aligned}
&z(k) \le M_x \delta(k) \\
&z(k) \ge m_x \delta(k) \Rightarrow -z(k) \le -m_x \delta(k) \\
&z(k) \le x(k) - m_x(1-\delta(k)) \\
&z(k) \ge x(k) - M_x(1-\delta(k)) \Rightarrow -z(k) \le -x(k) + M_x(1-\delta(k))
\end{aligned}$$

The new state equation takes the form

$$x(k+1) = 1.6z(k) - 0.8x(k) + u(k)$$

The system dynamics obtained is a linear dynamics constrained by a number of mixed-integer linear inequalities.

The final step of modeling is to organize obtained equations in the general form of MLD system

$$\begin{aligned}
&x(k+1) = A_x x(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) \\
&y(k) = Cx(k) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \\
&E_2 \delta(k) + E_3 z(k) \le E_1 u(k) + E_4 x(k) + E_5
\end{aligned}$$

where

$A_x = -0.8$, $B_1 = 1$, $B_2 = 0$, $B_3 = 1.6$, $C = 1$, $D_1 = D_2 = D_3 = 0$

$$E_2 = \begin{bmatrix} -m_x \\ -(M_x+\varepsilon) \\ -M_x \\ m_x \\ -m_x \\ -M_x \end{bmatrix}, E_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, E_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, E_4 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}, E_5 = \begin{bmatrix} -m_x \\ -\varepsilon \\ 0 \\ 0 \\ -m_x \\ M_x \end{bmatrix}$$

■

Comment on MLD modeling

An important note about MLD modeling of hybrid system is that the auxiliary variables (z and δ) appear in the MLD equations as if they are independent variables that are used to calculate next value of the system states x(k+1). However, any choice of these variables must satisfy the inequalities associated with MLD modeling. This group of inequalities is what maintains equivalence between original hybrid system and resulting MLD modeling. For example, in original hybrid systems: if the values of current states x(k) and current inputs u(k) are known, auxiliary variables (z and δ) should be uniquely defined according to their dependence on x(k) and u(k). On the other hand, in MLD modeling: if x(k) and u(k) are known and used for substitution in MLD inequalities, the resulting inequalities will be have only auxiliary variables (z and δ) remaining. If the inequalities are tested to obtain a feasible solution that satisfy all inequalities, the obtained feasible point will be the same as the values of z, δ obtained from hybrid system relations.

This previous discussion shows that MLD modeling has used linear inequalities to hide logic rules of original hybrid system. This – somehow- created an inverted problem as z and δ have been converted from depending variables to – only apparently – independent variables. To run a hybrid system simulation using MLD modeling, z and δ should be determined first based on current states x(k) before the new set of states x(k+1) can be calculated.
Finally, we should indicate that MLD modeling will be suitable for control synthesis of hybrid systems as will be shown later.

1.3.3 Control of hybrid systems

In this section we will study some of the control techniques that are suitable for hybrid systems. We have shown before that classical control methods cannot be used for hybrid system due to 1) existence of discrete nature in system performance and 2) the existence of constraints on inputs, outputs or states. Some researchers suggested a de-hybridization technique for hybrid system to allow for system control using classical approaches [28]. However, the most suitable control techniques are those that can explicitly handle constraints like optimal control and model predictive control [4]. The application of these control techniques to hybrid systems will be described in detail in the following section.

1.3.3.1 Optimal control

The optimal control problem of a MLD system can be formulated as follows [4].
Given an initial state x(0) and a final time T, find (if possible) the control sequence $u^{T-1} = \{u(0),u(1),\ldots\ldots,u(T-1)\}$that transfers the system states from x(0) to $x_{ref}(T)$ while minimizing the objective function

$$J = \|x(T) - x_{ref}(T)\|^2_{Q_{xN}} + \sum_{k=1}^{T-1} \|x(k) - x_{ref}(k)\|^2_{Q_x} +$$
$$\sum_{k=0}^{T-1} \|u(k) - u_{ref}(k)\|^2_{Q_u} + \|y(k) - y_{ref}(k)\|^2_{Q_y} + \|z(k) - z_{ref}(k)\|^2_{Q_z} \quad (1.11a)$$

Subject to the MLD dynamics (1.10) and any required limits on the states x, outputs y, control action u

$$\begin{aligned} &x_{\min} \le x(k) \le x_{\max} && k = 1,2,\ldots..,T \\ &y_{\min} \le y(k) \le y_{\min} && k = 0,1,\ldots..,T \qquad\qquad (1.11b) \\ &u_{\min} \le u(k) \le u_{\min} && k = 0,1,\ldots..,T \end{aligned}$$

Where

$$\|(x - x_{ref})\|^2_{Q_x} = (x - x_{ref})^T Q_x (x - x_{ref})$$

$Q_{x,}$ Q_u ,…….. Q_y are given weight matrices

X_{ref}(k),…….. y_{ref}(k) are the values of required references at instant k.

To cast this problem in the standard form of optimization problems, we need to change the objective function to make it a function of the independent variables only. We will use state and output equations of (1.10) to remove the dependent variables (x, y) from the objective function and from inequality constraints (1.10c) and (1.11b).

From (1.10a)

$$x(1) = A_x x(0) + B_1 u(0) + B_2 \delta(0) + B_3 z(0) \qquad (1.12a)$$

Similarly,

$$x(2) = A_x x(1) + B_1 u(1) + B_2 \delta(1) + B_3 z(1) \qquad (1.12b)$$

Hence,

$$x(2) = A_x{}^2 x(0) + A_x (B_1 u(0) + B_2 \delta(0) + B_3 z(0)) + B_1 u(1) + B_2 \delta(1) + B_3 z(1) \quad (1.12c)$$

By successive substitution we reach that a at any future instant k, the predicted state x(k) can be obtained from

$$x(k) = A_x{}^k x(0) + \sum_{i=0}^{k-1} A_x{}^{k-i-1} (B_1 u(i) + B_2 \delta(i) + B_3 z(i)) \qquad (1.13)$$

Consequently, at final time instant (k = T)

$$x(T) = A^T x(0) + \sum_{i=0}^{T-1} A^{T-1-i} (B_1 u(i) + B_2 \delta(i) + B_3 z(i)) \qquad (1.14)$$

We can notice that x(T) – along with all other states x(k) – depends only on x(0) – current measured state – which is known and appears as constant value in optimization and on the independent variables $u(i), z(i), \delta(i); i = 0,1,....T-1$.

By similar mathematical manipulation we find that y(k) depends only on x(0) and the same set of independent variables.

$$y(k) = Cx(k) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \qquad (1.15a)$$

Substituting by (1.13) we reach

$$y(k) = C(A_x{}^k x(0) + \sum_{i=0}^{k-1} A_x{}^{k-i-1} (B_1 u(i) + B_2 \delta(i) + B_3 z(i))) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \qquad (1.15b)$$

Substituting by (1.13) and (1.15b) in objective function we find that the only independent optimization variables are the future values of u, z and δ from the instant k = 0 to T – 1.

These set of variables are collected in the optimization vector q.

$$q = [u(0) \quad u(1) \quad \cdots \quad u(T-1) \;\vdots\; \delta(0) \quad \ldots \quad \delta(T-1) \;\vdots\; z(0) \quad \ldots \quad z(T-1)]^T \qquad (1.16)$$

The objective function (1.11) can be reshaped in the following form

$$\min_q \; J = \frac{1}{2} q^T \bar{G} q + (\bar{d}^T + \theta^T \bar{D}) q + \frac{1}{2} \theta^T \bar{U} \theta + \bar{V}^T \theta + \bar{f} \qquad (1.17a)$$

$$subject \;\; to \;\; \bar{A} q \leq \bar{b} + \bar{C}_x x(0) \qquad (1.17b)$$

The vector θ is related to required references and the value of states at the current sampling instant x(0).

$$\theta = \left[x(0) \quad x_{ref} \quad y_{ref} \quad u_{ref} \quad z_{ref} \right]^T \qquad (1.17c)$$

We can notice that using equality constraints for substitution leaves only the inequality constraint. Hence the problem is converted to inequality constrained optimization problem.

As the current state x(0) and all the references are determined by control requirements, the vector θ is known. By substitution in the objective function, we find that last three terms of objective function will evaluate to a constant. This constant will not affect the optimization problem and can be ignored. The final form of the objective function can be expressed as

$$\min \frac{1}{2} q^T G q + d q \tag{1.18a}$$

$$A q \le b \tag{1.18b}$$

where

$$G = \bar{G} \tag{1.19a}$$

$$d = \bar{d}^T + \theta^T \bar{D} \tag{1.19b}$$

$$A = \bar{A} \tag{1.19c}$$

$$b = \bar{b} + \bar{C}_x x(0) \tag{1.19d}$$

This form has the same standard form of quadratic programming problems. However, we should note that some of the optimization variables – δ(0) to δ(T - 1) – are binary variables. This converts optimization problem to a mixed binary quadratic programming MBQP problem or generally a mixed-integer quadratic programming MIQP. This type of problems is much more complex in solution than normal quadratic programming. The problem computational complexity increases exponentially with the number of binary variables.

There are a number of algorithms that can solve such problems [29]. The main algorithms are:

1) Cutting plane methods.

2) Decomposition methods.

3) Logic based methods.

4) Branch and bound methods.

It was found by [30] that the branch and bound algorithm is the best algorithm for mixed integer quadratic programming problems. Some recent works [31] introduce some preprocessing techniques in order to reduce the complexity of the mixed integer predictive control problems. Detailed discussion about algorithms used in the solution of the above problem is given in section 1.4.

1.3.3.2 Model predictive control of Hybrid systems

The optimal control strategy -discussed above- works as open loop control strategy. This is because the optimization problem is solved only once. The complete control sequence is computed before system is operated. During system operation, previously calculated control actions u(k) are applied to the system.

Open loop control has many drawbacks. The most critical drawback is that the generated control sequence cannot deal with disturbances in the system. This may lead to an output that don't regulate to the required steady state operating point. It may even lead to system instability. To overcome these drawbacks we need to apply a feedback control strategy. This is done by solving the optimization problem (1.18) online at each sampling instant. Solving such a problem online is not practically possible due to the large number of optimization variables and constraints. The online solution can be made possible by simplifying the optimization problem to be solved. This is done by limiting the prediction operation to a small future horizon instead of performing predictions until the final time instant of the system response. This proposed simplification is the main concept of model predictive control. In MPC the symbol T will represent the prediction horizon of the MPC controller and x(0) represent the value of the last reached state. The optimization problem in this case is based on the actually measured or estimated system states. Hence, the control algorithm can counterpart any disturbances affecting the system. The hybrid system model is used online by the controller to predict future system outputs and states. This makes the created optimization problem related to most recent measurements, hence it will address any variations during system operation. The execution of the MPC control strategy is illustrated in figure 1.2.

The mathematical formulation of the MPC problem for hybrid system is the same as (1.11) and can be similarly transformed to the form (1.19). The only difference is that T here represents the selected prediction horizon. The value of T is chosen based on the studied system and the required performance. Generally, increasing the prediction horizon results in better performance. On the other hand increasing T creates larger and more complex optimization problem that may prevent practical implementation. For example, if the MLD modeling of a hybrid system has i optimization variables (binary variables and auxiliary variables) and j constraints. The created MPC optimization problem has around i*T optimization variables and j*T constraints. In addition, the number of binary variables in optimization problem is multiplied by T. This makes the MPC problems with large T very much harder to solve. Methods of solution of such problems is discussed in section 1.4.

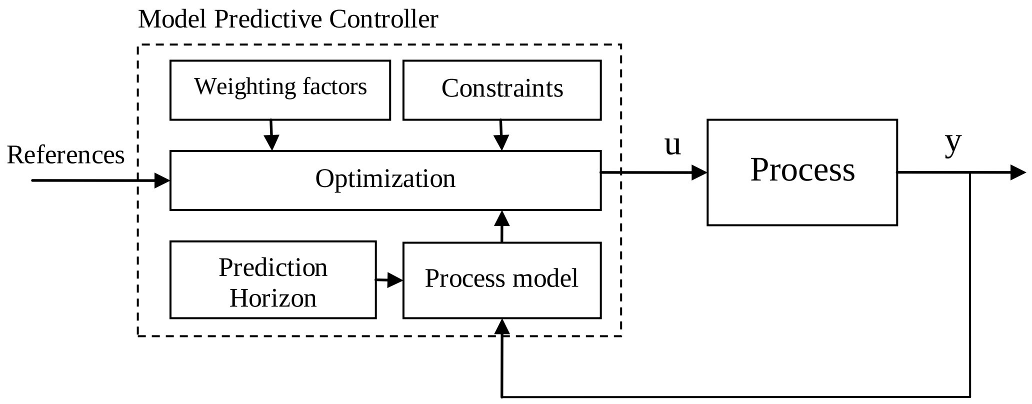


Figure 1.2: Execution of MPC control strategy

1.3.3.2 Explicit MPC control of hybrid systems

The idea of explicit solution of the online optimization problem discussed in section (1.2.3) was extended to the domain of hybrid MPC. The idea first appeared in [32]. In this paper, MPC was based on infinity norms instead of quadratic norms. It was shown that an equivalent piecewise affine explicit reformulation can be obtained through off-line multi-parametric mixed-integer linear programming techniques. To extend this explicit solution to the case of quadratic norms the, conversion of MLD modeling of hybrid system to equivalent piecewise affine systems was proposed in [33]. This allowed the introduction of an explicit solution of quadratic weighted MPC problem using dynamic programming iterations [34]. The solutions were obtained by solving multi-parametric quadratic programming problems. The obtained solutions represent optimal solutions of the hybrid MPC optimization problem for different subdivisions of the state space. These solutions are stored and during system operation the proper solution is used based on the subdivision where the current system states lie in. Some techniques were proposed to reduce the number of regions in the explicit solution [35]. A different approach of solving the explicit control problem is described in [36].

1.3.3.3 Comparison of online and explicit MPC control of hybrid systems

Explicit model predictive control is useful when we deal with small problems with small prediction horizon. When the problem size increases or large prediction horizon is used, the number of regions obtained by the explicit controller increases dramatically. This in turn makes the memory requirements for the implementation of the explicit controller prohibitive. Also the worst case search time to find the correct region becomes larger than the online solution case [16].

1.3.4 Tools available for analysis and control of hybrid systems

Research efforts in the field of analysis and control of hybrid systems resulted out the creation of a number of tools to help modeling, analysis and control of hybrid systems. One of these results was the introduction of HYSDEL [37] which stands for "**Hy**brid **S**ystem **De**scription **L**anguage". This language provides a frame to describe a wide class of hybrid systems. It allows declaration of continuous and discrete system states, input and outputs. It provides methods to describe logic variables based on continuous signals – i.e. events that depend on the value of continuous variables passing a certain threshold- and allows the use of these logic variables to perform dynamics switching. Moreover, it allows the declaration of constraints on both continuous and discrete variables. These capabilities of the HYSDEL modeling language allow the description of different hybrid system dynamics in a standard manner. A HYSDEL compiler accepts hybrid system descriptions in .hys files and converts it to a corresponding MLD model.

Another result was the development of a Matlab Hybrid Toolbox [38]. This toolbox integrates the HYSDEL compiler in Matlab environment. This way, we can create HYSDEL models and use the Hybrid Toolbox to convert it to MLD model in Matlab environment. It also allows the generation of an equivalent PWA models [33]. The toolbox provides routines for the open loop and closed loop simulation of the generated MLD models. It also provides routines for the synthesis and simulation of optimal and MPC controllers of hybrid systems.

The controller generation tools permits the declaration of different weighting factors for each term in the objective function, the introduction of hard or soft constraints, the introduction of limits on values of states, inputs or outputs and the use of 2 or infinity norms in the objective function. The toolbox also integrates some MILP solvers – for infinity norm optimizations – and MIQP solvers – for 2 norm optimizations.

When the objective function is based on 2-norm the optimization problem becomes a mixed integer quadratic programming problem (MIQP) in the form (1.17). The hybrid control routine “hybcon” generates all the related optimization matrices based on the provided MLD model, the required prediction horizon T and the weighing matrices Q_x, Q_u, Q_y, Q_z and Q_{xT} provided by control designer. The “hybcon” function allows the ability to introduce additional constraints on the system inputs, outputs and states.
The generated controller object is used by a closed loop simulation routine where all the required references are declared. The simulation routine uses the MLD model to predict the system future performance. It uses the declared references to calculate the vector θ. By using this vector, problem (1.17) is transformed to the standard form (1.18). A MBQP solver is called to solve the generated optimization problem. The optimal solution is returned and the first control action is applied to the MLD model to predict system new states and outputs. The operation repeats to perform a closed loop simulation of the controlled systems. The hybrid control toolbox was used in this work for simulation of the studied examples before practical implementation on hardware.

1.4 Quadratic programming methods

Optimization problems which have quadratic objective function and linear constraints are called quadratic programs. The general form of a quadratic program is

$$\min \quad J(q) = \frac{1}{2} q^T G q + q^T d \qquad (1.19a)$$

$$a_i^T q = b_i \qquad i \in \mathrm{E} \qquad (1.19b)$$

$$a_i^T q \geq b_i \qquad i \in \mathrm{I} \qquad (1.19c)$$

Where q is an n-element optimization vector, d and {a_i} are n-element vectors. G is a square n×n matrix. E is the set of equality constraints and I is the set of inequality constraints. The number of constraints (1.19b), (1.19c) is equal to m.

The matrix G plays a critical role in determining the complexity of solution of (1.19). If G is positive semi-definite the problem is said to be convex QP problem. Any local minimum of a convex problem is the global minimum. If G is indefinite the problem is said to be non-convex QP. In this case the problem may have multiple local minima. The solution of non-convex QPs is more challenging than the solution of convex QPs.

There exist many methods for the solution of a QP problem. Of those, we will discuss the two most famous techniques which are active set method and interior point method. Before getting in the details of both methods, optimality conditions of inequality constrained problem are reviewed.

1.4.1 Optimality conditions

Optimality conditions are obtained by introducing of Lagrange multipliers λ and forming the Lagrangian

$$L(q,\lambda) = \frac{1}{2} q^T G q + q^T d - \sum_{i \in \mathrm{I} \cup \mathrm{E}} \lambda_i (a_i^T q - b_i) \qquad (1.20)$$

For the optimal solution q* we define the optimal active set A*(q*) which includes the indices of the active constraints – constraints satisfying equality– at q*.

$$A^*(q^*) = \{ i \in \mathrm{E} \cup \mathrm{I} : a_i^T q^* = b_i \} \tag{1.21}$$

The first order optimality conditions are

$$Gq^* + d - \sum_{i \in A^*(q^*)} \lambda_i^* a_i = 0 \tag{1.22a}$$

$$a_i^T q^* = b_i, \quad for \quad all \quad i \in A^*(q^*) \tag{1.22b}$$

$$a_i^T q^* \geq b_i, \quad for \quad all \quad i \in \mathrm{I} \setminus A^*(q^*) \tag{1.22c}$$

$$\lambda_i^* \geq 0, \qquad for \quad all \quad i \in \mathrm{I} \cap A^*(q^*) \tag{1.22d}$$

1.4.2 Active set method

Active set method is one of the most successful methods of solution of small QP problems [14]. The active set method solves the inequality constrained quadratic program (1.19) by solving a series of simpler equality constrained problems.

The main idea of the active set method is to scan the boundary of the problem feasible region. This is done by assuming that some constraints are active (constraints satisfy equality), i.e., we form an active set. This active set along with the objective function are solved as equality constrained problem. This is done by solving the first order optimality condition. The obtained solution is studied to inspect whether it is optimal for the whole problem or not. If the used active set didn't result in the optimal solution, the active set is changed by adding or removing one of the constraints and the operation is repeated until the optimal solution is reached.

As the optimal active set is not initially known, we go through a number of iterations. Each iteration, we select some constraints and assume they are active. Hence we form an active set $A_s(q)$ of these constraints.

$$A_s(q) = \{ i \in \mathrm{E} \cup \mathrm{I} : a_i^T q = b_i \} \tag{1.23}$$

We then solve the equality constrained problem given by

$$\min \quad J(q) = \frac{1}{2} q^T G q + q^T d \tag{1.24a}$$

$$A_{active} q = b_{active} \tag{1.24b}$$

Where

$$A_{active} = [a_i]^T_{i \in A_s(q)} \qquad (1.24c)$$

$$b_{active} = [b_i]^T_{i \in A_s(q)} \qquad (1.24d)$$

This equality constrained problem can be solved by solving the first order conditions (1.24). These conditions are called *Karush–Kuhn–Tucker* conditions, or KKT conditions for short. The condition (1.24a) and (1.24b) can be written in the following matrix form

$$\begin{bmatrix} G & -A_{active}{}^T \\ A_{active} & 0 \end{bmatrix} \begin{bmatrix} q^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -d \\ b_{active} \end{bmatrix} \qquad (1.25)$$

This KKT conditions can be introduced in a more suitable form by introducing any initial feasible point $q_{0,}$ the required optimum solution can be expressed as $q^* = q_0 + p$. where p is the step from q_0 to q*. The KKT conditions convert to

$$\begin{bmatrix} G & A_{active}{}^T \\ A_{active} & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ c \end{bmatrix} \qquad (1.26a)$$

Where

$$g = d + Gq_0 \text{ , } c = A_{active}q_0 - b_{active} \qquad (1.26b)$$

By solving this set of linear equation we reach an optimum pair (p, λ*) which optimizes the equality constrained problem. Studying the obtained optimum pair, two cases arise:

- The first case is that p is equal to zero. This means that the initial value q_0 is the optimum value of the equality constrained problem. It is also the optimal value of the inequality constrained problem provided that all Lagrange multipliers λ* are positive. We hence inspect the signs of the inequality constraints in the active set. If all are positive then we have reached the solution of the inequality constrained problem. If any of the Lagrange λ* multipliers is negative, then its corresponding constraint doesn’t belong to the optimal active set. So, this constraint is removed from the current working active set and a new iteration is performed.
- The second case is that the resulting p doesn't equal zero. In this case, the reached point $q = q_0 + p$ is not the optimal solution and there should be another point q* that optimizes the equality constrained problem. The case here is that

during the movement from q_0 to the optimal point q* we may violate one of the original problem constraints that are not included in the active set. In this case the violated constraint – called blocking constraint – is added to the active set and a new iteration is performed with the new defined active set.

To determine the blocking constraint that will be added to the active set, all constraints that are not included in the active set are examined. The blocking constraint is recognized by being the nearest constraint to the current point q_0 that will turn negative as we move in the direction of P. So, we calculate a distance α to all constraints that are not included in the active set that may turn negative. The constraint with minimum α is the blocking constraint.

α can be calculated by

$$\alpha = \min(1, \min(\frac{b_i - a_i^T q_0}{a_i^T P})), \quad , i \notin A_s(q) \qquad (1.27)$$

After α is calculated the blocking constrained is added to the active set and we start a new iteration. The initial point in the new iteration is calculated by

$$q' = q_0 + \alpha P \qquad (1.28)$$

It should be noted that if α is equal to one, that means that no blocking constraint exists in our way to the optimum q* and a new iteration is performed taking

$$q' = q0 + P \qquad (1.29).$$

Where q' is the initial point of the new iteration.

The active set method can be implemented by the following algorithm [14].

Algorithm 1.1 (Active Set method for Convex QP)

```
Compute a feasible start point $q_0$;
Get W0 to be a subset of the active constraints at $q_0$;
For k = 0, 1, 2, …..
    Solve KKT conditions (1.26) to find $p_k$;
    If $p_k = 0$
        Compute Lagrange multipliers $\lambda$ associated with the solution
        If $\lambda_i \geq 0$ for all $i \in W_k \cap I$ ;
```

```
                **STOP** with solution $x^* = x_k$;
            **else**
                set $j = \min_{j \in W_k \cap I} \lambda_j$ ;
                $q_{k+1} = q_k$;
                $w_{k+1} \leftarrow w_k \setminus \{ j \}$;
        **else** (* $p_k \neq 0$ *)
            compute $\alpha_k$ using (1.27)
            $q_{k+1} \leftarrow q_k + \alpha_k p_k$;
            **If** there are blocking constraints
                Obtain $w_{k+1}$ by adding one of the blocking constraints to $w_{k+1;}$
            **else**
                $w_{k+1} \leftarrow w_k$;
**end** (**for**)
```

Disadvantages of Active Set method

1. Active set method needs a feasible starting point for its implementation. So, it requires a preceding algorithm to obtain a feasible starting point before the active set method is applied.
2. As Active Set scans the boundary of the feasible region it may need a large number of iterations when problem size increases.

1.4.2 Interior point methods

The interior point methods represent another approach to solve quadratic programming problem (1.20). There are two main differences between interior point methods and active set method [14].

1. Interior point methods try to reach the solution by using a lower number of iterations. This is done by performing more complex calculations in each iteration.
2. Active set method reaches optimum solution by scanning the boundary of the problem's feasible region. It moves from a point on the boundary to another

point that has better value of the objective function. This process is repeated until the optimal solution is found. In interior point method, the solution obtained each iteration never lies on the boundary of the feasible region. It approaches the boundary only at the point where optimal solution exists. Interior point iterations can lie in either the interior or the exterior of the feasible region.

Because of these differences authors of [14] find that interior point method is considered more efficient and more suitable for implementation than Active Set method.

Interior point methods solve the general inequality constrained problem by converting it to an equality constrained problem. Solution is reached by solving KKT conditions –or a modified version of them- of this problem. There are two famous types of interior point methods; Barrier method and Primal-Dual interior point method. According to [39] primal-dual method is more efficient than barrier methods especially when high accuracy is required. Moreover, primal-dual method has exhibit better than linear convergence. The focus in this thesis will be on primal-dual method.

1.4.2.1 Primal-dual interior point

Primal-dual interior point method was first introduced to solve linear programming problems in [40]. Most of the main concepts of the method either in application to linear or quadratic programming problems originate from this paper. As convex QP problems represent a subset of Linear Complementarity Problems (LCP), the basics of primal-dual interior point method for QP problems was introduced in [41]. This method solves QP problems by applying some variations to Newton methods. Newton's method is used to solve the KKT conditions of the constrained QP problem. The concept of Newton’s method is to calculate a direction that points toward improving the value of the objective function. The solution is improved by moving a certain step in this direction. It is better to take long steps in the calculated direction. Long steps mean that we can reach optimal solution in a fewer number of iterations. A usual disadvantage of the basic Newton method is that the taken steps are usually very small. This occurs as large steps usually result in a solution that violates the problem constraints. Taking such short steps will make convergence to optimal solution very slow.

Primal-dual interior point method suggests some modifications to Newton's method like central path and path-following. These modifications improve the speed of convergence by allowing the algorithm to take large steps in the calculated search direction.

1.4.2.2 Problem formulation of primal dual-interior point method

Consider the inequality constrained QP problem

$$\min J(q) = \frac{1}{2} q^T G q + q^T d \qquad (1.30a)$$

$$Aq \geq b \qquad (1.30b)$$

Where G is a symmetrical positive semi-definite matrix and A and b contain the coefficients of system inequality constraints. The vector q collects n optimization variables. The number of constraints equal m.

Forming the lagrangian function by adding Lagrange multiplier vector λ, the KKT conditions for the optimality of this system are:

$$Gq - A^T \lambda + d = 0 \qquad (1.31a)$$

$$Aq - b \geq 0 \qquad (1.31b)$$

$$(Aq - b)_i \lambda_i = 0 \qquad (1.31c)$$

$$\lambda \geq 0 \qquad (1.31d)$$

By introducing a vector of slack variables $s = Aq - b$, the optimality conditions can be formed as

$$Gq - A^T \lambda + d = 0 \qquad (1.32a)$$

$$Aq - s - b = 0 \qquad (1.32b)$$

$$s_i \lambda_i = 0 \qquad (1.32c)$$

$$s, \lambda \geq 0 \qquad (1.32d)$$

Newton’s method can be used to solve these KKT conditions. Solution is performed on two steps.

First, calculation of a search direction (Δq, Δs, $\Delta\lambda$) by

$$\begin{bmatrix} G & -A^T & 0 \\ A & 0 & -I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta s \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Lambda S e \end{bmatrix} \tag{1.33}$$

Where $\Lambda = \text{diag}(\lambda)$, $S = \text{diag}(s)$ and $e = [1\ 1\ 1\ \ldots..\ 1]_m^T$

Then, we estimate the maximum step length that can be taken in these directions. This step length α should ensure that the non-negativity constrains of s, λ are not violated. The step length α takes value in the range (0,1].

Finally, the new iterate is calculated by

$$(q^+, s^+, \lambda^+) = (q, s, \lambda) + \alpha(\Delta q, \Delta s, \Delta\lambda) \tag{1.34}$$

Iterations continue until we reach the required accuracy. The problem with this basic Newton’s algorithm is that the taken step is usually very small $\alpha << 1$. This prevents the algorithm from achieving fast progress toward the final solution.

Primal dual interior point method improves this behavior by introducing some bending to the search direction. This bending is done toward the interior of the nonnegative orthant $(s, \lambda) \geq 0$. This way we can move farther in the calculated search direction before one of the components of s or λ turns negative.

This modification of the algorithm is performed by introducing what is called the “Central Path”. Central path is defined in [42] as the points that satisfy the solution of

$$Gq - A^T\lambda + d = 0 \tag{1.35a}$$
$$Aq - s - b = 0 \tag{1.35b}$$
$$s_i\lambda_i = \tau \tag{1.35c}$$
$$s, \lambda \geq 0 \tag{1.35d}$$

Where τ is a positive integer $\tau > 0$.

It can be noticed easily that the central path contains the optimal solution if $\tau = 0$. Consequently, the central path can be considered as a path that respects the non-negativity constraints of s, λ in all of its points and moves toward optimal solution as τ → 0. Definition of central path and the proof of its existence first appeared at [43].

Primal-dual method takes Newton like steps toward points of central path instead of pure Newton steps. It can be seen intuitively that as we move toward points on the central path, the non-negativity constraint of s, λ is respected which allows for longer steps.

To introduce these concepts in the primal-dual interior point algorithm two parameters are introduced [40], namely σ (centering parameter) and μ (duality measure).

The centering parameter σ describes the amount of bending we take toward the central path. If σ = 0 then we take pure Newton steps. If σ = 1, we take Newton steps toward points on the central path. Points on the central path do not give the optimal solution. Hence, taking $\sigma = 1$ may not give a good progress toward optimum. Consequently, σ takes intermediate values in the range [0,1]. σ can be changed during progress of the problem to allow faster progress toward optimum.

The duality measure μ is defined by

$$\mu = \frac{\sum_{i=1}^{m} s_i \lambda_i}{m} = \frac{s^T \lambda}{m} \tag{1.36}$$

As the optimal solution requires $s^T\lambda = 0$, the duality measure μ represents an indicator to our progress toward optimum. As μ decreases we get closer to optimal solution. The centering parameter and duality measure are related to the central path by taking τ = σ μ. In this way, each iteration both μ and τ decrease to approach optimality and at the same time the violation of the non-negativity constraints is avoided.

The modified search direction of the primal-dual method is calculated by the following matrix equation

$$\begin{bmatrix} G & -A^T & 0 \\ A & 0 & -I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta s \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Lambda S e + \sigma \mu e \end{bmatrix} \tag{1.37}$$

1.4.2.3 Infeasible primal-dual interior point method

The previously described method assumes that an initial feasible solution to the problem exists (satisfying both equality and non-negativity constraints). All following iterations will also be feasible. However, finding such a feasible starting point is not an easy task. This problem was first solved in [44], [45] in the context of linear programming by allowing the primal-dual method to start at any infeasible point. The only condition on that starting point is that it satisfies the non-negativity constraints. This condition is simple to achieve. The modified algorithm is called infeasible primal-dual interior point method. The paper [46] was the first to introduce a rule for step length that ensures global convergence. Superlinearly convergent variants of the algorithm were later proposed in [47], [48].

The first extension of primal-dual path-following methods to convex optimization was given by [49] with superlinear convergence being established in [50].

Infeasible primal-dual interior point method starts by any random starting point that satisfies non-negativity constraints, i.e., we can start with any positive initial value of s, λ and any initial value of q. Then, the algorithm calculates residues of this starting point with respect to the equality constraints of KKT conditions. It works to minimize these residues as algorithm execution proceeds. The search direction in infeasible interior point method is calculated by

$$\begin{bmatrix} G & -A^T & 0 \\ A & 0 & -I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_b \\ -\Lambda S e + \sigma\mu e \end{bmatrix} \tag{1.38}$$

Where r_d, r_b are the residues w.r.t the equality constraints and are calculated by

$$r_d = Gq - A^T\lambda + d, \qquad r_b = Aq - s - b \tag{1.39}$$

After calculation of the search direction, we search for the maximum possible step length α that maintain all components of s, λ positive. For α to ensure some progress of solution toward optimality path following algorithms are used. These algorithms relate the calculated α to the main parameters of primal-dual method σ and μ to speed up the convergence of the solution. One of the most successful path following algorithms – namely, long step path following – is described in the following section.

To finish a single iteration the new solution step is calculated by

$$(q^+, s^+, \lambda^+) = (q, s, \lambda) + \alpha(\Delta q, \Delta s, \Delta\lambda) \tag{1.40}$$

The main computational complexity in this flow is the solution of the linear system of equations (1.38). However, this set of equations can be simplified by elimination of variables to the following equations

$$(G + A^T (S^{-1}\Lambda)A)\Delta q = -r_d + A^T (S^{-1}\Lambda)[-r_b - s - \sigma\mu\Lambda^{-1}e] \tag{1.41a}$$

$$\Delta s = A\Delta q + rb \tag{1.41b}$$

$$\Delta\lambda = S^{-1}(\sigma\mu e - S\lambda - \Lambda\Delta s) \tag{1.41c}$$

The infeasible primal-dual interior point method can be implemented by the following algorithm [14].

Algorithm 1.2 (infeasible primal-dual interior point method for Convex QP)

```
Choose an initial point at which all components of s, λ are positive.
While error > required accuracy
    Compute residues
        $r_d = Gq - A^T\lambda + d$
        $r_b = Aq - s - b$
    Compute σ and μ
        $\mu = \frac{s^T\lambda}{m}$, σ is chosen using path following algorithm
    Compute search direction using the equations
        $\Delta q = (G + A^T (S^{-1}\Lambda)A)^{-1}(-r_d + A^T (S^{-1}\Lambda)[-r_b - s - \sigma\mu\Lambda^{-1}e])$
        $\Delta s = A\Delta q + rb$
        $\Delta\lambda = S^{-1}(\sigma\mu e - S\lambda - \Lambda\Delta y)$
    Compute step length α using path following according to σ, μ used
    Update the current iterate value by
        $(q^+, s^+, \lambda^+) = (q, s, \lambda) + \alpha(\Delta q, \Delta s, \Delta\lambda)$
    Calculate the error by
```

```
        $error = \max(abs((q^+, s^+, \lambda^+) - (q, s, \lambda)))$
    end (while)
```

It is worth mentioning here that infeasible primal-dual interior point method is considered one of the most successful methods for the solution of convex optimization problems. Hence, it was chosen for the implementation of an FPGA QP solver [7]. This solver was used to implement a hardware model predictive controller for linear systems.

1.4.2.4 Long-step path following

Path following algorithms were adopted in primal-dual interior point methods from the early publications in that topic. It was proposed for linear programming in [51] and for quadratic programming in [50]. The computational complexity of the algorithm was studied in [52]. Path following allows the complexity of primal-dual methods to be reduced to polynomial complexity. The main idea of path following algorithm is to restrict solution iterations to certain neighborhood around the central path and following it to the optimal solution. In this manner, as we always near the central path the primal-dual steps will be longer which means faster convergence to the solution. Long-step path following is implemented by first defining a neighborhood around central path. Each iteration, the step length is calculated such that the obtained solution not only avoids non-negativity constraint violation, but also remains in the selected neighborhood. This way, it ensures that successive iterations don't get so close to the boundary of the feasible region. Hence, longer steps will be possible. One of the most used neighborhoods in path following [14] is the one sided ∞- norm neighborhood $\mathcal{N}_{-\infty}(\gamma)$ defined by

$$\mathcal{N}_{-\infty}(\gamma) = \{(q, \lambda, s) \in \mathcal{F}^0 \mid \lambda_i s_i \geq \gamma\mu \quad all \quad i = 1, 2, \ldots, m\} \qquad (1.42)$$

Where the parameter $\gamma \in (0,1]$ and typical value of it is 10^{-3}. The set $\mathcal{F}^0$ is the strictly feasible set defined by

$$\mathcal{F}^0 = \{(q, \lambda, s) \mid Aq = b, A^T\lambda + s = c, (s, \lambda) > 0\} \qquad (1.43)$$

In the ∞- norm neighborhood each product $\lambda_i s_i$ must be at least some small multiple

γ of their average value μ. By achieving this we ensure that each pair $\lambda_i s_i$ approaches zero at approximately the same rate.

The described long-step path-following is known practically to achieve good performance. Its implementation uses two parameters σ_{min} and σ_{max} which are bounds on the value of the centering parameter σ. The algorithm calculates the step length such that each iteration solution remains in the defined neighborhood. Long step path following can be implemented according to the following algorithm [14].

Algorithm 1.3 (long-step path-following)

```
Given $\gamma$, $\sigma_{min}$ ,$\sigma_{max}$ with $\gamma \in (0,1]$, $0< \sigma_{min} < \sigma_{max} <1$,
    and $(q,s,\lambda) \in \mathcal{F}^0$
for k = 0, 1, 2, …
    chooses $\sigma_k \in [\sigma_{min} ,\sigma_{max}]$;
    solve (1.41) to calculate $(\Delta q^k, \Delta s^k, \Delta\lambda^k)$;
    choose $\alpha_k$ as the largest value of $\alpha$ in [0, 1] such that
        $(q^k, s^k, \lambda^k) + \alpha_k (\Delta q^k, \Delta s^k, \Delta\lambda^k) \in N_{-\infty}(\gamma)$;
        Set $(q^{k+1}, s^{k+1}, \lambda^{k+1}) = (q^k, s^k, \lambda^k) + \alpha_k (\Delta q^k, \Delta s^k, \Delta\lambda^k)$
    end (for)
```

1.5 Mixed Binary Quadratic Programming

Mixed Binary Quadratic Programs (MBQP) represents a subset of Mixed Integer Quadratic Programs (MIQP). A MIQP program is similar to quadratic program (QP). Their main difference is that some optimization variables are not allowed to take real values, instead, they are restricted to be integer values. Although the simplicity of solution of QP problems. The integrality constraints convert the problem to what is known by NP-hard problem [5], which means that -in worst case- solution complexity increases exponentially with problem size. MIQP problems appears usually in management science. In the context of optimal and MPC control of hybrid dynamical systems a variant of MIQP problem appear. This variant is the MBQP problem. The main feature of MBQP is that the integer variables are restricted to be binary {0, 1}. Hence the name Mixed Binary Quadratic Programming. This work will focus on the MBQP as it is the one that appears in the MPC control of hybrid dynamical system.

The MBQP problem can be formulated as following

$$\min_q J = \frac{1}{2} q^T G q + dq \qquad (1.44a)$$

Subject to

$$A_{eq} q = b_{eq} \qquad (1.44b)$$

$$A\, q \geq b \qquad (1.44c)$$

Where $q \in \Re^{n_{real}} \times \{0,1\}^{n_{bin}}$, G is a symmetrical positive semi-definite matrix of size $n = n_{bin} + n_{real}$, $d \in \Re^{n_{bin}+n_{real}}$.

There are many proposed algorithms for the solution of this problem [53]. The most common techniques are

1) Cutting plane methods
2) Decomposition methods
3) Logic based methods
4) Branch and bound methods

It was found that in most cases the branch and bound method gives superior performance over other methods of solving MBQP [30]. With a few exceptions branch and bound method was found to be an order of magnitude better than other methods. Details of branch and bound are given in the following section.

Branch and Bound

Branch and bound is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. It is main idea is to split the solution space into different subsets. During the scan of these subsets large subsets of fruitless candidates can be discarded by estimation of upper and lower bounds of the objective function being optimized. The method was first proposed by in 1960 [54] for discrete programming.

The efficiency of the method depends strongly on the splitting procedure used and on the methods of estimation of upper and lower bound of the studied objective function. Intuitively, it is better to split the solution space to disjoint subsets.

The structure of the MBQP problem is suitable to be solved by branch and bound. The reason for that is that the binary variables that exist in optimization can be divided to non-overlapping subsets easily. The solution of MBQP can be performed by studying all possible combinations of binary variables involved in the problem. Doing so, the remaining variables are real ones and hence there will be a QP associated with each combination of binary variables. Solving these different QPs, we can take the one that results out the minimum value to be the optimal solution of the MBQP problem. The associated binary variables combination with that QP will be the binary part of the optimum solution. However, this method of solution is very time consuming especially when the number of binary variables increases as the number of binary combinations and hence the number of QP solved is exponentially related to the number of binary variables. The concept of branch and bound provides a path where not all binary combinations are studied. The algorithm solves the MBQP problem by building a tree. In each level of this tree one of the binary variables is enumerated to its possible values -we call this process “Branching”.

By branching the solution space is divided into two subsets. We then try to estimate which of them will result better objective function value. We try to

calculate upper and lower bounds to both sets. An upper bound of a subset can be obtained if this subset includes a feasible solution to the MBQP problem. A lower bound can be obtained by finding a solution to a relaxed problem in the subset. This relaxed problem is formed by relaxation of the remaining binary variables as real variables in the range [0, 1]. The resulting subproblem is solved as a QP. The objective function value obtained from the relaxed problem can be considered as a lower bound for the value of the MIQP in the studied subset.
In this manner- as we search for the minimum achievable value- if it was found that the lower bound of a subset ‘i’ is larger than the upper bound of another subset ‘j’ , then subset ‘i’ can be safely rejected as it will never contain minimum “optimal” solution –we call this process “Pruning” or “Bounding”.

If the resulting solution has all relaxed variables satisfying integrality constraints. Then, the obtained solution is also a feasible solution to the original MBQP. The value of the relaxed objective function in this case can be considered as an upper bound for the solution of the MBQP problem. If the resulting solution had some of the relaxed variables not satisfying the integrality constraints, we continue the branching process by choosing one of these variables for further enumeration. This creates child subproblems for the current subproblem. These child problems form the following level of the tree.

While solving the generated tree of subproblems, we have some situations in which we can ignore the solution of all subproblems that form a complete branch of the tree. First of these situations occurs when the lower bound for a certain subproblem is greater than an already known feasible solution (upper bound). This means that neither this problem nor any of its child problems will have a better solution than the already known one. Hence, all problems that exist in this branch of the tree are neglected.

Another situation in which many subproblems can be ignored is when the solution of a relaxed subproblem is infeasible. If the relaxed problem is infeasible, then the

original MIQP subproblem is infeasible and all its child subproblems are infeasible. A third situation is that when a subproblem is solved to optimality, i.e., the relaxed subproblem resulted in a solution which has all relaxed variables with integer values. In this case none of the child subproblems can have a better solution. Consequently, branching is stopped at this level.
The branch and bound algorithm can be implemented according to the following algorithm [5].

Algorithm 1.4 Branch and bound for binary variables

```
upper_bound = inf
add the current MIQP problem to List
while length (List) > 0 do
    pop a problem P from List
    relax the binary conditions and solve the QP problem
    if P has no feasible solution
        stop branching on this variable
    else if objective function value > upper_bound
        stop branching on this variable
    else if p has a solution with all relaxed variables having binary values
        - stop branching on this variable
        if objective function value < upper_bound
                upper_bound = value of the objective function
        end if
    else
        - choose one of the relaxed variables for branching
        - create two problems $P_0$, $P_1$ by fixing the selected variables with 0
          and with 1 respectively.
        - push the problems $P_0$, $P_1$ to List
    end if
end while
```

Conclusion

In this chapter a short overview of the main topics in this work was presented. It has started in section 1.1 by an introduction to the problem and how it will be handled. Then a review to the main topics was presented in the following sections. Section 1.2 aims to review the model predictive control strategy and explore its advantages. The mathematical formulation of the control strategy was shown in the case of a linear system, to illustrate the basic steps of building the required optimization problem. In section 1.3, hybrid dynamical systems were discussed. Their possible modeling and control strategies were discussed. The mixed logical dynamical “MLD” general modeling framework was reviewed in section 1.3.3. The application of MPC to hybrid system was discussed in section 1.3.4. The different algorithms used in the solution of the resulting optimization problems were reviewed in sections 1.4 and 1.5. First, the solution of a quadratic programming problem was considered in section 1.4. Then, the solution of a mixed-binary quadratic programming was reviewed in section 1.5.

In the discussion of QP methods of solution, two main algorithms were compared: active set method and primal-dual interior point method. The advantages of interior point method were presented. A fast convergence variant of the primal-dual interior point method- namely the path following infeasible primal-dual interior point method – was discussed in detail. Algorithms for practical implementation of these methods were also presented.

In the discussion of MBQP solution method different solution algorithms were considered. The literature in that field showed the superiority of the Branch and Bound method. The concepts and advantages of this method were reviewed. This section was concluded by an algorithm for practical implementation of the branch and bound algorithm.

Chapter 2 Hardware Implementation

2.1 Design objective

The objective of this work is to implement a hardware model predictive controller for hybrid embedded systems. The main part of this controller is a fast MBQP solver. The implementation of such a solver on hardware requires in the first place implementing an efficient hardware QP solver. The reason is that most algorithms solve the MBQP problem by solving a series of QP problems. This shows that the efficiency of a MBQP solver depends basically on the implementation of an efficient QP solver.

The implementation of a QP solver may have different forms as there are various algorithms to solve a QP problem as shown in section 1.4. Moreover, there are many different approaches to implement this solver. For example, the solver can be implemented using an on-board processor and memory as in [6]. In this publication a real time MPC was implemented using a single board computer. That board utilized a high speed 32-bit Motorola microcontroller that contains a 64-bit floating point unit to speed up calculations.

Another approach that allows the execution of parallel floating point operations is to use Graphical Processing Unit (GPU). These approaches can result in fast solver implementation. However, in this thesis as we target embedded systems, we try to implement the hybrid controller on-chip.

One of the approaches to implement the controller on-chip is to implement an on-chip processor and use this processor to run software that executes the solver algorithm. This approach can be referred to as “software approach”. This approach gives larger flexibility in modifying the code of the solver and improving it. It also requires short design time as there are many ready embedded processor implementations that can be used directly like Altera NIOS II processor [55] and Xilinix MicroBlaze processor [56]. However, the performance of such systems may not be satisfactory for many reasons. These processors – generally – are general purpose processors and may not be suitable to perform fast floating point operations. Even when the performance is improved by using processors with dedicated floating points units, the sequential nature of software execution will require all floating point operations to be executed in series. This sequential execution will make the software implementation of the QP solver slow. To have a good performance using the “software approach” custom processor architecture may be required as in [8]. In this paper an application specific matrix coprocessor was implemented. In this implementation a 16 bit logarithmic number system arithmetic unit was designed to perform calculations. The designed coprocessor was used along with a limited resource host processor in a co-design framework to implement a real time MPC for linear and nonlinear systems. The main advantage of the use of the logarithmic number system is the area and power saving.

The performance of “software approach” can also be improved by applying the concept of parallel processing. This is done by implementing multiple instances of the on-chip processor and developing solver software that can use multiple processors. Such an implementation reduces the execution time significantly compared to sequential software execution.

A different approach for the on-chip implementation of the system is to build a custom hardware to solve the QP problem. This “Hardware approach” requires larger design effort and time. However, it allows much more flexibility in controlling the solver speed. An example of the custom hardware implementation of the QP solver appears in [7]. In this paper, the authors introduce an FPGA implementation of the MPC strategy for linear systems. The introduced design has applied interior-point method to solve the resulting QP problem. Single precision floating point representation was used to represent numbers in the design. The design has proved good performance. However, its main disadvantage is that it has not introduced any sort of pipelining or parallelism to improve the performance.

For a custom hardware implementation, the solver speed can be increased by using the concepts of parallelism and pipelining. Consequently, the hardware approach allows the implementation of QP solvers with short execution time. Fast QP solvers will enable us to implement fast MBQP solvers to control fast hybrid dynamical systems.

Recently, there were some efforts to build fast MBQP solvers as in [57]. In this paper, a Mixed Integer (Binary) Quadratic programming solver was implemented for the hybrid control of mobile robot. The implemented design used a previously described C++ code to implement the QP solver. The dual active set algorithm was used in the QP solver implementation. The design applies some concepts like pipelining to improve the performance.

3.2 Design Environment

The hybrid MPC controller was implemented by using VHDL hardware description language. The design currently targets an ALTERA FPGA for prototyping purposes. The final target of implementation is to implement the controller as an Application Specific Integrated Chip (ASIC). Implementing the controller as ASIC will allow its application to much faster hybrid systems. The design currently uses the floating point adders, multipliers, dividers and comparators available in ALTERA Megafunction library. All floating point numbers in the design follow the IEEE-754 32-bit floating point representation.

The functional behavior of the design is verified by using Modelsim simulator. The Synthesis of the VHDL code and the layout on the specified FPGA was performed by ALTERA Quartus II software. The timing simulation of the design was performed by Modelsim in combination with delay information obtained from Quartus II timing analyzer.

To facilitate the functional simulation of the system, a subset of the VHDL-2008 standard [58] is used. The used subset includes a floating point package [59]. This package includes some functions to convert a 32 bit std_logic_vector in the IEEE-754 to a real number. These conversion functions were very useful in simplifying the simulation process.

To allow testing hybrid MPC controller with multiple examples, a link between Modelsim and Matlab was established [60]. This link was used for two purposes:

1. To verify the performance of the hardware controller main blocks, which are the hardware interior point solver and the branch and bound MBQP solver. In this case, Matlab passes the optimization problem matrices to Modelsim. A

dedicated part of the hardware receives this data and stores it in its corresponding location in the on-chip memory. Then, the solver is started. It solves the problem and stores the result back to the on-chip memory. Finally, another part of the hardware reads the memory and passes the solution back to Matlab. The result obtained by the hardware solver is compared with the result of a Matlab function implementing the same solver.

2. The Matlab-Modelsim link was also used to simulate a control system along with the proposed hybrid MPC controller.

2.3 Design main concepts

This section discusses how hardware design concepts like pipelining and parallelism can be used to implement high speed QP and MBQP solvers to be used within the hybrid MPC controller. Different hardware architectures and design options are studied and compared to select the best one for the implementation of the required system. One of these studies is to inspect the effect of using pipelined floating point components instead of non-pipelined components on the solver speed. Another study tries to find the most suitable number of parallel units as a function of the problem parameters.

The concepts of pipelining and parallelism are also applied on function level. For example, when the interior point algorithm was studied, some operations were found to be independent. These independent operations were implemented to be executed in parallel to shorten the algorithm execution time. Another example will be shown later to describe the ability of using pipelining in function level to speed up the calculation of matrix operations required by the algorithm.

Before going to the details of implementation of main system blocks, the concepts of parallelism and pipelining is revisited in the following sections. Some examples will be given to show how these concepts are used to speed up the operation of the system.

2.3.1 Concept of parallelism

The concept of parallelism or parallel design means the usage of multiple instances of the same block to perform similar independent operations in the same time. Parallelism will be useful in the context of implementation of the QP solver, as the QP solution algorithm is based entirely on matrix and vector operations. Vector operations by nature allow the use of parallel components. Example 2.1 shows how the concept of parallelism can be used to achieve fast vector addition by increasing the number of floating point adders used.

Example 2.1

Consider Figure 2.1 where the case of simple vector addition is studied, i.e., we need to add two vectors of length ℓ. The addition of these two vectors can be performed either by using a single adder for ℓ cycles, or by using ℓ adders for only one cycle. The second solution is a clear example of the concept of parallel design. In this situation, we sacrifice more hardware to have a faster solution. An intermediate solution between the above two extremes is to use a number of adders equal *P* where *P* is less than ℓ. In this case the completion of the vector addition requires ceil (ℓ/*P*) cycles. The same discussion applies to the cases of simple vector by vector point multiplication, vector by scalar multiplication, vector by vector point division and vector by scalar division. ■

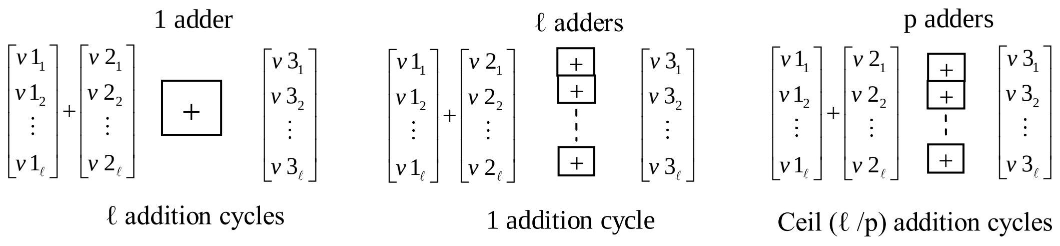


Figure 2.1: using of parallel units in vector addition

The concept of parallelism can be also used on function level. Multiple units of the same block are used to perform similar independents calculations at the same time to speed up operation. Example 2.2 illustrates this concept.

Example 2.2

Consider the solution of a MBQP problem using branch and bound algorithm. The solution requires solving a group of QP problems. These problems can be solved in series by using a single QP solver. Another approach is to apply the concept of parallelism where multiple QP solvers are implemented as in figure 2.2. Each of these solvers is used to solve a different QP problem. All solvers work in parallel in the same time. The MBQP control unit controllers the distribution of the QP problems to the different solvers. This reduces the execution time of the MBQP problem by a factor proportional to the number of implemented QP solvers.

■

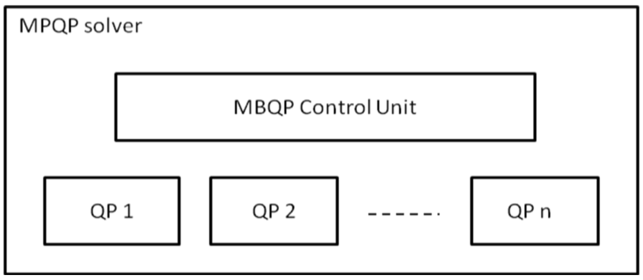


Figure 2.2: Block parallelism in MBQP hardware

3.3.2 Concept of pipelining

The concept of pipelining allows increasing of a hardware design throughput. Pipelining is performed by dividing a complex operation into smaller independent operations "stages". These stages can be operated sequentially i.e., each stage waits for the result of execution of the preceding stage. However, this approach is not efficient. Only one stage is active at a time and all other stages are idle. To increase the throughput of the design, we allow all stages to work in the same time. This is done by applying a new input to the system in each cycle. In this case, each stage holds the data related to a different input. Hence, we will have a new output each cycle. Figure 2.3 illustrates the concept of pipelining when a task is divided to three smaller tasks. The cost paid to have higher throughput is the increased hardware needed. The main cause of the hardware increase is the need of more registers to store the information obtained from each stage to be available for processing by the following stage.

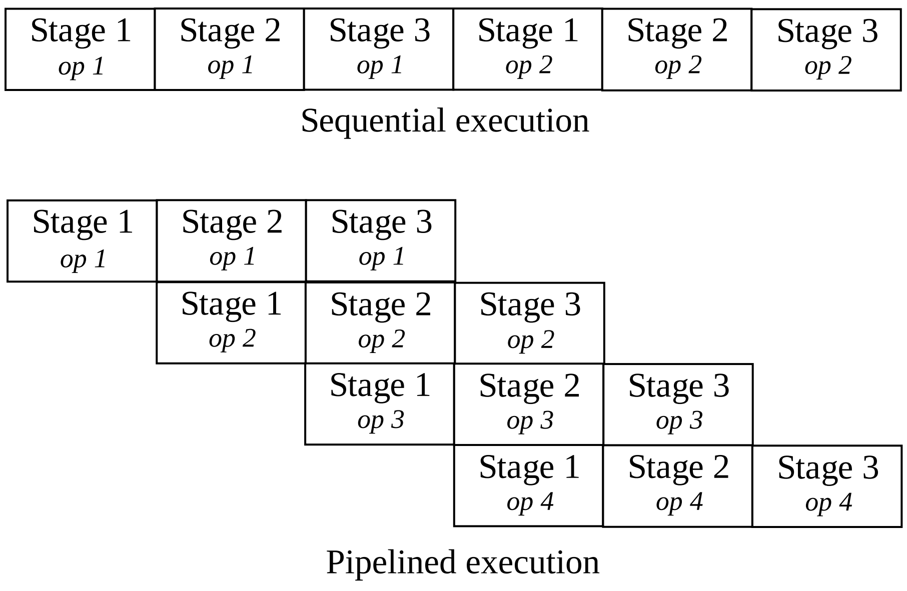


Figure 2.3: sequential execution versus pipelined execution.

The concept of pipelining can be applied in many ways to reduce the time of the required calculations. One of these ways is the use of pipelined floating point calculation units instead of sequential execution units, i.e. pipelined floating point adders and pipelined floating point multipliers. Another way to make use of pipelining is to apply it on function levels where a complex calculation is divided to small independent parts which can be calculated in the pipelining fashion. Example 2.3 shows how the use of a pipelined adder can be used to achieve a fast vector addition using only one pipelined FP adder.

Example 2.3

Consider the simple vector addition described in example 2.1, where two vectors of length ℓ are added. As the operation of adding a pair of elements of the two vectors is independent on the addition of any other pair of elements, a pipelined adder can be used to add the two vectors as shown in figure 2.4. The adder is assumed to have D_A pipelining stages. Figure 2.4 shows that the addition of the two vectors ends in (ℓ +D_A) clock cycles using only a single floating point adder.

■

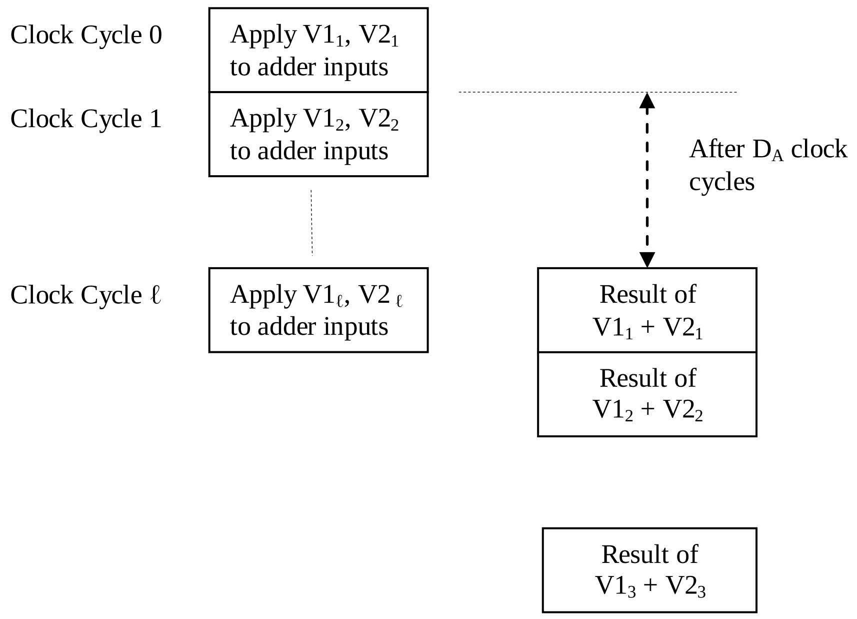


Figure 2.4: two vector addition with a single pipelined adder with D_A stages

When results of examples 2.1 and 2.3 are compared, we find that a parallel design will give a faster result if it is possible to use a large number of parallel FP adders. If the number of adders used is equal to or greater than ℓ, the vector addition task will end in D_A clock cycles. However, for large values of ℓ, the use of an equal number of FP adders will not be practical due to hardware area limitations. In this case, if the number of used adders is equal to P which is less than ℓ, the task will take (D_A×ceil(ℓ/P)) clock cycles. On the other side, the use of a single pipelined adder will require (ℓ +D_A) clock cycles for any value of ℓ. Equation 2.1 compares the latency of both architectures. It sets a condition of the number of parallel adders P that should be used to guarantee better performance for parallel adders' architecture. In this equation ceil(ℓ/P) is replaced by ℓ/P for simplification.

$$D_A\left(\frac{\ell}{P}\right) < \ell + D_A \tag{2.1a}$$

$$\left(\frac{\ell}{P} - 1\right)D_A < \ell \tag{2.1b}$$

$$(\ell - P)D_A < \ell P \tag{2.1c}$$

$$P(\ell + D_A) > \ell D_A \tag{2.1d}$$

$$P > \frac{\ell D_A}{(\ell + D_A)} \tag{2.1e}$$

$$P > \frac{D_A}{(1+\frac{D_A}{\ell})} \qquad (2.1f)$$

Equation 2.1 shows that for large values of ℓ, the performance of a single pipelined adder surpasses the performance of P parallel sequential adders if the number of P is smaller than the adder delay D_A.

Example 2.4 shows how the concept of pipelining can be applied on functional level. This example applies pipelining in the domain of matrix calculations. In specific, the complex task of matrix vector multiplication will be considered.

Example 2.4

Consider we have a matrix X and a vector V and we need to calculate X*V. Assuming that both X and V are stored in memory and the result will be stored back to memory. The steps of calculation of this multiplication process requires three operations

1. loading of operands
2. processing of operands which includes multiplications and additions
3. storing of results.

To achieve a short calculation time, we need to pipeline the previous steps. Naive matrix-vector multiplication is not perfectly suitable for pipelining as will be shown later. So an alternative method of multiplication is used.

Assume the matrix X has dimensions of $n_1 \times n_2$ and the vector V has the dimension of $n_2 \times 1$.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n_2} \\ x_{21} & x_{22} & & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ x_{n_1 1} & \cdots & & x_{n_1 n_2} \end{bmatrix}, \; V = \begin{bmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{n_2 1} \end{bmatrix} \qquad (2.2)$$

The naïve matrix vector multiplication calculates the i_{th} element of the result vector on two steps. The first is to multiply the n_2 elements of the i_{th} row of the matrix X by the n_2 elements of the vector V. The second step is to sum all the n_2 multiplication results. Hence, the calculation of each element of the n_1 elements of the result vector requires n_2 multiplications followed by $n_2 - 1$ additions. The multiplications can be done in parallel by using n_2 multipliers. The result can be available in one multiplication cycle. However, the additions require n_2-1 additions. If the additions are performed sequentially, the addition step will require n_2-1 addition cycles. If it is assumed that both multiplication cycle and addition cycle requires the same number of clock cycles D_A, then the calculation of an element of the result vector needs n_2D_A cycles. Then the matrix vector multiplication requires $n_1n_2D_A$ clock cycles, using n_2 multipliers and a single adder.

In this approach we can notice that during the calculation of an element of the result vector, the n_2 multiplier units were active for 1 multiplication cycle only and idle for n_2-1 addition cycles. Also during additions, only one adder is used for n_2–1 cycles and no parallelism is applied to improve performance.

If multiple adder units can be used, it is possible to speed up the addition task of the naïve matrix vector multiplication. The used adders can be formed in a tree as shown in figure 2.5. Applying this adder organization, each addition task is completed in ceil(log2(n_2)) addition cycles by using n_2-1 adder units. Moreover, the used organization allows for pipelining to be applied. The time required to complete a matrix-vector multiplication using n_2 multipliers and n_2-1 adders with the application of pipelining is (n1+ ceil(log2(n_2))) addition/multiplication cycles. If both multiplier and adder require the same number of clock cycles D_A, then we need (n_1+ ceil(log2(n_2))×D_A) clock cycles.

The addition scheme used in figure 2.5 has reduced the computation delay of the matrix-vector multiplication. Moreover, it has improved utilization. However, obtained utilization is not perfect as some levels of the adder tree will be idle during the calculation of the first elements of the result vector and also during the calculation of the last elements in that vector.

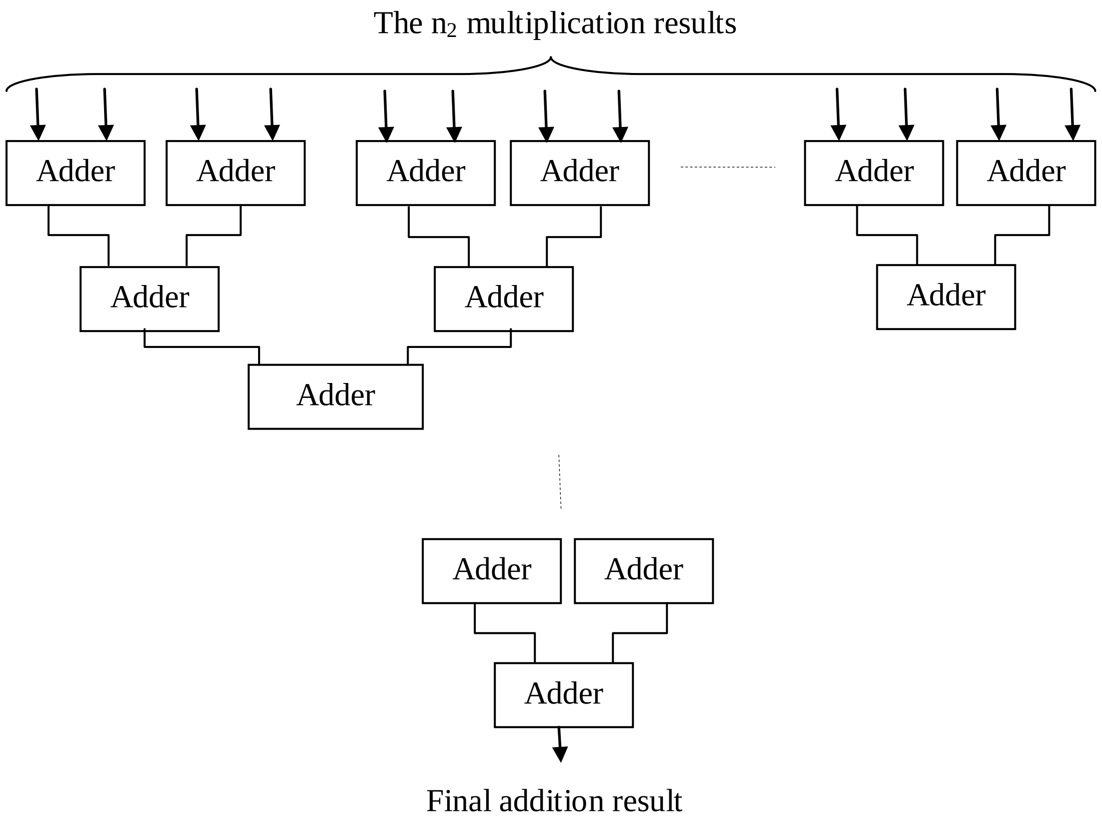


Figure 2.5: Adder tree

In this thesis, an alternative method for matrix vector multiplication is adopted to increase hardware utilization.

This method calculates the result vector by the following equation

$$\text{Result vector} = v_{11} \times X\ (col 1) + v_{21} \times X\ (col 2) + \cdots\cdots + v_{n_2 1} \times X\ (col\ \mathrm{n}_2) \qquad (2.3)$$

Each term of equation (2.3) requires a $n_1 \times 1$ vector by scalar multiplication. Hence, each term requires n_1 multiplications that can be performed in parallel and take one multiplication cycle if n_1 multipliers are used. Equation (2.3) has n_2 terms and each term of this equation is a $n_1 \times 1$ vector. The calculation of the result vector requires the addition of these n_2 vectors. Hence we have $n_2 - 1$ vector additions. In each of these additions we add two $n_1 \times 1$ vectors which can be done in parallel by the use of n_1 adders. This way, each vector addition will take only one addition cycle.

What makes this method more suitable for pipelining is the independency of the added terms. For example, while the n_1 adder units are used to add the first two terms of equation (2.3), the n_1 multiplier units can be used to calculate the third term of the same equation. As we assume that both addition and multiplication use the same number of clock cycles D_A, the addition of the first two terms and the multiplication required to calculate the third term will end at the same time. The operation is continued by adding the third term to the result of the last summation. During this addition the fourth term of the equation can be calculated by the multipliers. The same process is repeated until we add all terms of equation 2.2. This method of calculation is illustrated by the Gantt's chart in figure 2.6.

The discussion above shows that concept of pipelining has been applied efficiently to the matrix vector multiplication. This matrix operation -represented by equation (2.3)- was divided to simpler independent operations (multiplication and addition) that can be operated at the same time. Operating these multiplication and addition processes in the pipelined manner described in figure 2.6 reduces the required time of calculation. Moreover, the hardware utilization was improved as multipliers and adders are active nearly for the whole time of operation.

	Multiplier	Adder
Cycle 0	$v_{11} \times X\ (col\,1)$	
Cycle 1	$v_{21} \times X\ (col\,2)$	Add mult. out with zeros (first cycle)
	$v_{31} \times X\ (col\,3)$	Add mult. out with prev. adder result
	……………	………………
Cycle n_2	$v_{n_2 1} \times X\ (col\ \mathrm{n}_2)$	Add mult. out with prev. adder result
Cycle n_2+1		Add mult. out with prev. adder result

Figure 2.6: Pipelined operation of matrix vector multiplication

It is clear from figure 2.6 that completing a matrix vector multiplication by using the proposed pipelining scheme requires only n_2 +1 multiplication/addition cycles or equivalently $(n_2 +1)\times D_A$ clock cycles compared to $n_1n_2D_A$ clock cycles if pipelining is not used.

	Memory loading	Multiplier	Adder	Memory storage
Cycle 0	Load V_{11} , X(col 1)			
Cycle 1	Load V_{21} , X(col 2)	$V_{11}\times$X(col 1)		
	Load V_{31}, X(col 3)	$V_{21}\times$X(col 2)	Mult_out + 0	
	…………	$V_{31}\times$X(col 3)	Mult_out + adder_out	
Cycle n_2	Load $V_{n2\,1}$, X(col n_2)	…………	Mult_out + adder_out	
Cycle n_2+1		$V_{n2\,1}\times$X(col n_2)	…………	
Cycle n_2+2			Mult_out + adder_out	
Cycle n_2+3				Store result

Figure 2.7: Pipelined matrix-vector multiplication with memory access

It should be noted also that memory access required to fetch operands and store results can also be pipelined in the same way. A Gantt's chart showing the matrix vector multiplication along with the required memory access is shown in figure 2.7.

It can be noted that during the last cycles of execution the loading hardware is not used. To improve utilization these cycles can be used to load the data that will be used for the following calculation. Same concept is also applied on multiplier units and adder units.

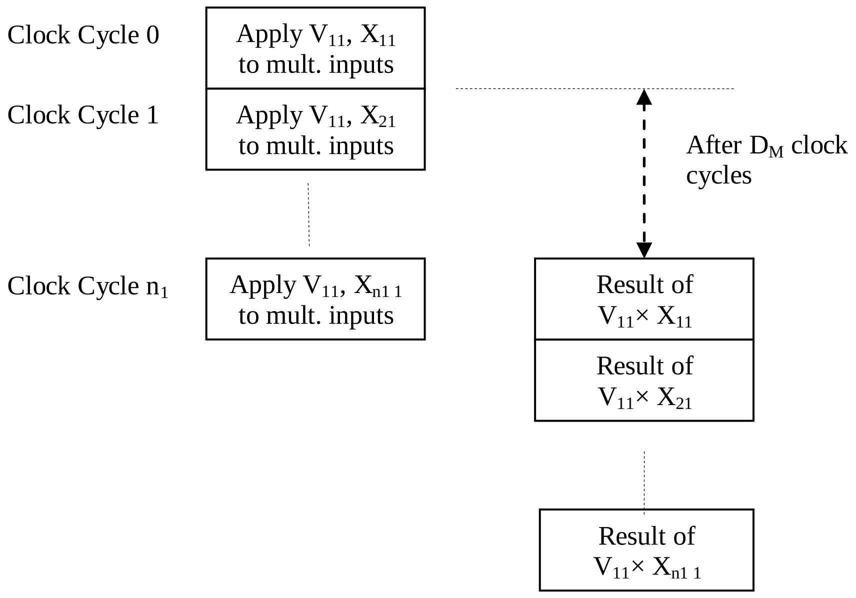
Using pipelining has reduced the number of cycles required to complete the execution to $(n_2 +3)\times D_A$ clock cycles . However, it requires the use of n_1 parallel adders and multipliers. If we use a number of multipliers and adders P less than n_1, the matrix vector calculation will require $((n_2+3)\times\text{ceil}(n_1/P))$ multiplication cycles or equivalently $((n_2+3)\times\text{ceil}(n_1/P)\times D_A)$ clock cycles.

■

It was shown in example 2.4 that the concept of pipelining along with parallel design units can be used to significantly reduce the computation time of the process of a matrix vector multiplication. However, the description of this example has not make advantage of the ability to use pipelined adders and multipliers. If such advantage is used, we can reach a comparable computation time for the same task while using much smaller number of adders and multipliers. Example 2.5 proposes the calculation of a matrix-vector multiplication task using only one pipelined adder and one pipelined multiplier.

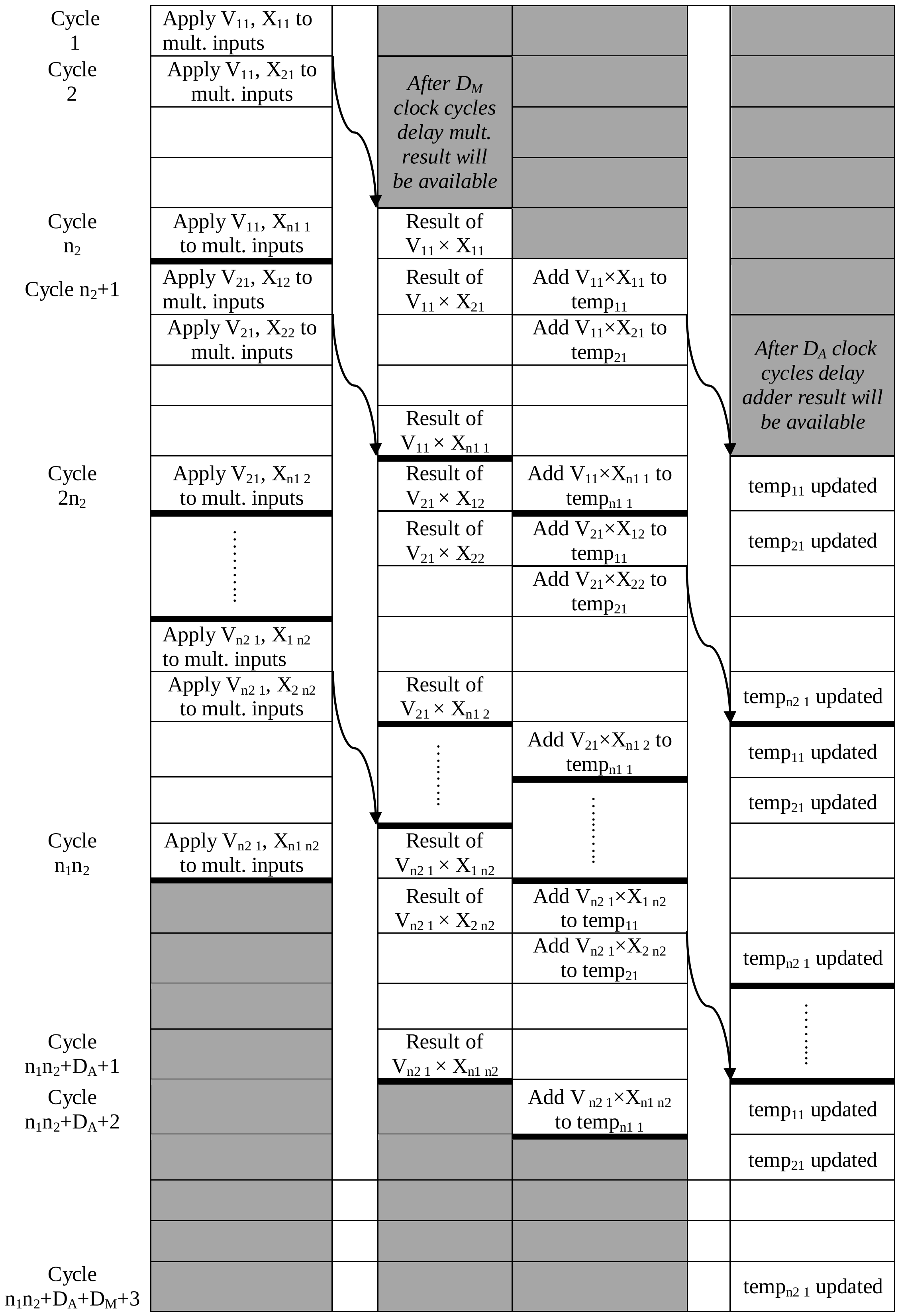
Example 2.5

Consider the multiplication of the matrix and vector shown in equation (2.2) using the multiplication method described by equation 2.2. Each term in equation 2.2 is a vector by scalar multiplication that results in a $n_1 \times 1$ vector. Each of these multiplications can be performed by a single pipelined multiplier in $(n_1 + D_M)$ clock cycle, where n_1 is the number of rows of the matrix X and D_M is the number of pipelining stages of the pipelined multiplier. An example of the execution of such multiplication is shown in figure 2.8, where the calculation of the first term of equation 2.2 ($V_{11} \times X(Col_1)$) is considered.



Clock Cycle 0
Apply V_{11}, X_{11} to mult. inputs
Clock Cycle 1
Apply V_{11}, X_{21} to mult. inputs
Clock Cycle n_1
Apply V_{11}, $X_{n1\,1}$ to mult. inputs
After D_M clock cycles
Result of $V_{11} \times X_{11}$
Result of $V_{11} \times X_{21}$
Result of $V_{11} \times X_{n1\,1}$

Figure 2.8: A vector by scalar multiplication using a single pipelined multiplier with D_M stages



Cycle 1
Apply V_{11}, X_{11} to mult. inputs
Cycle 2
Apply V_{11}, X_{21} to mult. inputs
After D_M clock cycles delay mult. result will be available
Cycle n_2
Apply V_{11}, $X_{n1\,1}$ to mult. inputs
Result of $V_{11} \times X_{11}$
Cycle n_2+1
Apply V_{21}, X_{12} to mult. inputs
Result of $V_{11} \times X_{21}$
Add $V_{11}{\times}X_{11}$ to $temp_{11}$
Apply V_{21}, X_{22} to mult. inputs
Add $V_{11}{\times}X_{21}$ to $temp_{21}$
After D_A clock cycles delay adder result will be available
Result of $V_{11} \times X_{n1\,1}$
Cycle $2n_2$
Apply V_{21}, $X_{n1\,2}$ to mult. inputs
Result of $V_{21} \times X_{12}$
Add $V_{11}{\times}X_{n1\,1}$ to $temp_{n1\,1}$
$temp_{11}$ updated
Result of $V_{21} \times X_{22}$
Add $V_{21}{\times}X_{12}$ to $temp_{11}$
$temp_{21}$ updated
Add $V_{21}{\times}X_{22}$ to $temp_{21}$
Apply $V_{n2\,1}$, $X_{1\,n2}$ to mult. inputs
Apply $V_{n2\,1}$, $X_{2\,n2}$ to mult. inputs
Result of $V_{21} \times X_{n1\,2}$
$temp_{n2\,1}$ updated
Add $V_{21}{\times}X_{n1\,2}$ to $temp_{n1\,1}$
$temp_{11}$ updated
$temp_{21}$ updated
Cycle n_1n_2
Apply $V_{n2\,1}$, $X_{n1\,n2}$ to mult. inputs
Result of $V_{n2\,1} \times X_{1\,n2}$
Result of $V_{n2\,1} \times X_{2\,n2}$
Add $V_{n2\,1}{\times}X_{1\,n2}$ to $temp_{11}$
Add $V_{n2\,1}{\times}X_{2\,n2}$ to $temp_{21}$
$temp_{n2\,1}$ updated
Cycle $n_1n_2+D_A+1$
Result of $V_{n2\,1} \times X_{n1\,n2}$
Cycle $n_1n_2+D_A+2$
Add $V_{n2\,1}{\times}X_{n1\,n2}$ to $temp_{n1\,1}$
$temp_{11}$ updated
$temp_{21}$ updated
Cycle $n_1n_2+D_A+D_M+3$
$temp_{n2\,1}$ updated

Figure 2.9: Pipelined matrix vector multiplication using pipelined adder/multiplier

To calculate the matrix vector multiplication result vector we need to add the n_2 terms of equation 2.2. Each of the required addition tasks is a two $n_1 \times 1$ vector addition task. This addition task can be performed in (n_1 + D_A) clock cycle by the use of a single pipelined adder as was shown before in example 2.3.
Using the same calculation strategy described in example 2.4, the task of addition of any term i to the previous addition result can be executed in the same time as the term i+1 is calculated by multiplier. A detailed illustration of the proposed method of execution is shown in figure 2.9. We assume here that we have an $n_1 \times 1$ temporary vector that is used to store intermediate addition results. This vector will be initially set to zeros.
Figure 2.9 shows that the calculation requires (n_1n_2+D_A+D_M+3) clock cycles using only one pipelined adder and one pipelined multiplier. This result may be better than the result obtained by example 2.4 if the hardware area limits the number of parallel adders/multipliers P to a small value compared to n_1.

■

2.4 Hardware implementation of Quadratic Programming solver

In this section we will give some details about the implementation of the Quadratic Programming solver that is used in the hardware implementation of hybrid model predictive controller. It was discussed in section 2.1 that an efficient implementation of a QP solver plays a crucial rule in the implementation of a fast hybrid MPC. In the design of the QP solver the concepts of parallel design and pipelining -discussed in section 2.3- were applied wherever possible to achieve a fast QP solver implementation. The Algorithm that was used for hardware implementation is the Primal-Dual infeasible interior point method discussed in section 1.4.2. This choice is based on many factors. The first factor is its good performance of this method. It was proved in [50] that this method has a superlinear convergence rate. Another factor is that interior point methods was found by authors of [14] more suitable for practical implementations. A hardware QP solver design based on interior point method was presented [7]. The design proved good efficiency in the solution of QP problems. However, the mentioned design has not applied pipelining or parallel design to speed up the solution. In this work, we try to improve the solver latency by applying these concepts.

Infeasible primal dual method solves a QP problem in a few iterations. However, the main complexity in this method is that; in each iteration we need to solve a complex system of equations. By a proper hardware design this system of equations can be handled efficiently to have a fast solution.

This section is organized as follows; in section 2.4.1 some modifications to the equations of algorithm 1.2 are presented to be more suitable for hardware implementation. In section 2.4.2 the Gauss-Jordan method for solving a linear system of equations is revisited. The hardware architecture of the QP solver is described in section 2.4.3.

2.4.1 Changes in the Algorithm for hardware implementation

The implementation of the QP solver in this thesis is based on the infeasible primal dual interior point method discussed in section 1.4.2.3. The equations of algorithm 1.2 – which is used to implement this method – concentrate on the procedure and do not consider computational complexity. Some of these equations require computation expensive operations that can make any implementation of the algorithm inefficient. To improve the performance of the algorithm, some of its equations can be reformulated to be more suitable for implementation. This section shows some examples of these simplifications.

For example, consider equation (1.41a) that calculates Δq

$$(G + A^T (S^{-1}\Lambda) A)\Delta q = -r_d + A^T (S^{-1}\Lambda)[-r_b - s - \sigma\mu\Lambda^{-1}e] \qquad (2.4)$$

This equation requires the calculation of the inverse of matrix S – which is an m×m matrix- and multiplication of the result with matrix Λ – also an m×m matrix. As m – the number of constraints – is usually large, the calculation of the inverse of S and the multiplication of S and Λ will be very time consuming. These complex calculations can be significantly simplified by noting that both S and Λ are diagonal matrices with diagonal elements coming from vectors s and λ respectively. Hence, the calculation $(S^{-1}\Lambda)$ which results in a diagonal matrix can be performed by calculating only the diagonal elements of the result matrix. These elements are the result of division of vector λ by the vector s element by element. This simplification avoids the calculation of a matrix inverse and a matrix-matrix multiplication by a much simpler division process.

The simplification is continued as the matrix A^T should then be multiplied by the result matrix of $(S^{-1}\Lambda)$. This matrix-matrix multiplication is also avoided. As the matrix $(S^{-1}\Lambda)$ is diagonal then the result $A^T(S^{-1}\Lambda)$ can be simply calculated column by column as follows: the i_{th} column of the result $A^T(S^{-1}\Lambda)$ is the result of multiplying the i_{th} column of A^T by the i_{th} diagonal element of $(S^{-1}\Lambda)$. Moreover, as the term $A^T(S^{-1}\Lambda)$ appears in both sides of the equation, it is then stored in a temporary matrix to avoid repeating the calculation.

Another simplification is performed in the term $(\sigma\mu\Lambda^{-1}e)$. In this term, the matrix Λ is diagonal with diagonal elements coming from vector λ, the vector e is an $m\times1$ vector of all ones. The result of matrix vector multiplication is then multiplied by the constant $\sigma\mu$. This calculation can be performed by simply dividing $\sigma\mu$ on each element of the vector λ.

The same concept of simplification is used to simplify the calculation of $\Delta\lambda$

$$\Delta\lambda = S^{-1}(\sigma\mu e - S\lambda - \Lambda\Delta s) \qquad (2.5)$$

As S is a diagonal matrix, the vector resulting from the multiplication $S\lambda$ equals the diagonal elements of S multiplied by the elements of the vector λ, or simply the result of multiplying the vectors s and λ element by element. Hence, the matrix vector multiplication can be simplified to a two vector point multiplication. Similarly, as Λ is a diagonal matrix then $\Lambda\Delta s$ can be calculated as a point multiplication of the two vectors λ and Δs. The calculation of the matrix inverse S^{-1} can be avoided by dividing the elements of the vector $(\sigma\mu e - S\lambda - \Lambda\Delta s)$ by the elements of the vector s.

2.4.2 Algorithm of solving linear system of equations

Equation 2.3 requires the solution of a linear system of equations to calculate the search direction Δq. This system of equations has n unknowns, where n is the number of optimization variables of the QP problem. There are several methods to solve this linear system of equations like matrix inversion, elimination of variables, Cramer's method, Gauss elimination and Gauss-Jordon, matrix decomposition and iterative methods [61-63]. The method used for hardware implementation is the Gauss-Jordan method [62] because if its simplicity. Gauss-Jordan method is known to have a complexity of O(n^3). The basic idea of Gauss-Jordan is to use elementary row operations to transform the coefficients matrix of equation 2.4 $(G + A^T(S^{-1}\Lambda)A)$ to the unity matrix. Performing the same elementary row operations on the right-hand side vector of equation 2.4, transforms this vector to the solution vector Δq.

To transform the coefficients matrix to a unity matrix, a pivot element is selected in each row. The pivot element is transformed to 1 by dividing the complete row by the value of the pivot element. Then, all elements above or below the pivot element are transformed to zero by the use of row operations. This operation is repeated for all rows until the coefficients matrix is transformed completely to the unity matrix.

During the calculation of Gauss-Jordan special care should be given to the selection of pivot elements. Improper choice of pivot elements may lead to wrong solution. To ensure numerical stability pivot elements are chosen by using partial pivoting technique [63]. Partial pivoting chooses pivot element to be the element with maximum absolute value in the current column. This choice of pivot elements will make the error due to approximation or truncation small. Hence, the approximation error will not affect the correctness of solution. Algorithm 2.1 describes an implementation of Gauss-Jordan method with partial pivoting.

Algorithm 2.1 (Gauss-Jordan for solving linear system of equation with partial pivoting)

Given a linear system of equations $A_{coeff}\times q=b_{rhs}$ with n unknowns.

```
i := 1
j := 1
while (i ≤ n and j ≤ n) do
  generate an augmented matrix $A_{aug}$= [$A_{coeff}$|$b_{rhs}$], $A_{aug}$ has n rows and n+1 coulmns
  Find pivot in column j, starting in row i:
  maxi := i
  for k := i+1 to n do
    if abs($A_{aug}$ [k,j]) > abs($A_{aug}$ [maxi,j]) then
      maxi := k
    end if
  end for
  if $A_{aug}$ [maxi,j] $\neq$ 0 then
    swap rows i and maxi, but do not change the value of i
    Now $A_{aug}$ [i,j] will contain the old value of $A_{aug}$ [maxi,j].
    divide each entry in row i by $A_{aug}$ [i,j]
    Now $A_{aug}$ [i,j] will have the value 1.
    for v := i+1 to n do
      subtract $A_{aug}$ [v,j] × row i from row v
      Now $A_{aug}[v,j]$ will be 0, since $A_{aug}[v,j]-A_{aug}[i,j]\times A_{aug}[v,j]=A_{aug}[v,j] -1\times A_{aug}[v,j] = 0$.
    end for
    i := i + 1
  end if
  j := j + 1
end while
last column of the augmented matrix $A_{aug}$ is the solution vector q
```

2.4.3 Hardware architecture of QP solver

The hardware of the QP solver can be divided into three parts. Arithmetic computation hardware, storage hardware and control hardware. The arithmetic computation consists of floating point adders, multipliers and dividers used to perform arithmetic operations. Storage hardware is the memories and registers that are used to store QP problem matrices and intermediate calculation results. Control hardware is the hardware that organizes the operation of the solver and controls the flow of operands between different calculation units and controls the operation of storage memories.

As indicated in section 2.2 the arithmetic computation units currently used are provided by ALTERA Megafunctions library. The arithmetic units used are pipelined units which allow the use of small number of units to perform complex calculations. For example, the matrix vector multiplication procedure described in example 2.5 was used to implement all the matrix vector multiplications found in the interior point algorithm. Reason of preference of this procedure is that it uses the concept of pipelining efficiently to have the required calculation done in a short time. Moreover, it uses only a single FP adder and a single FP multiplier which saves design area. The saved area is used to implement additional adders and multipliers to perform other calculations in parallel to the currently performed calculation. For example, consider equation (1.39) which calculates the residuals r_d and r_b in the infeasible primal-dual interior point method. The calculation of r_d requires the multiplication of the matrix G and the vector q while the calculation of r_b requires the multiplication of the matrix A and the vector q. Both calculations of $G\times q$ and $A\times q$ can be performed in parallel provided that the vector q can be accessed by the two calculating units at the same time. Each calculating unit will require an adder/multiplier pair to perform the multiplication. Hence, the concept of parallel design is applied by implementing two adder/multiplier pairs. Each of these pairs is used in the same time to perform a different multiplication. This parallelism reduces the algorithm execution time significantly especially when the number of parallel computation units increase. The infeasible primal-dual interior point method was analyzed to obtain all possible independent operation sequences. Each independent sequence of operations will be performed –as the hardware area allows- using a separate hardware. Figure 2.10 shows an example of independent sequence of operations found in the implemented algorithm for QP problem solution.

It is worth mentioning that the hardware used for a certain calculation is reused again in the following calculations. For example the adder/multiplier pair used to calculate $G\times q$ is used again in the addition of $G\times q$ and $A^T\times\lambda$ and also in the addition of d to calculate r_d. All operations of the system components are controlled by the control unit. The control unit is designed as a finite state machine that operates different arithmetic units and maintains synchronization between different calculations.

Calculation of r_d		Calculation of r_b		Calculation of $G+A^T(S^{-1}\Lambda)A$
$G\times q$	$A^T\times\lambda$	$A\times q$	$b+s$	$S^{-1}\Lambda = \lambda/s$
$G\times q- A^T\times\lambda$		$r_b = A\times q -(b+s)$		$temp\_mat=A^T(S^{-1}\Lambda)$
$r_d = (G\times q - A^T\times\lambda)+d$				$A^T(S^{-1}\Lambda)A = temp\_mat\times A$
				$G+A^T(S^{-1}\Lambda)A$

Figure 2.10: Example of parallel operation in interior point algorithm

The last part of the QP solver hardware is the storage part. All the information needed by the QP solver is stored on-chip. On-chip storage allows faster access of data compared to off-chip storage. On-chip storage hardware takes two forms; memories and registers. Registers are used in the design to store important variables that are required to be accessed by multiple sources in the same time like the optimization vector q. Variables that require large amount of storage like matrices G and A or variables that are not accessed by multiple sources in the same time are stored in dedicated on-chip memories. However, each of these variables is stored in a separate memory to allow them to be accessed simultaneously. Large variables -like matrix A- that are needed to be accessed simultaneously are implemented as two-port memories or three-port memories depending on the number of units requiring simultaneous access. This allows overcoming the need to wait for memory access and allows parallel execution of different calculations. The matrices that are implemented as on-chip memory are stored column major. A small combinational hardware is used to translate the row and column indices to an explicit memory address. This allows dealing with matrices using usual row and column indices.

It should be noted here that the number of available memory bits on a certain target FPGA sets the limit of maximum size of the QP problem that can be handled. For a certain target FPGA, the current design implements the largest possible QP solver. This solver can be used to solve any QP problem of that maximum size or of a smaller size. This feature in the QP solver is important if it will be used in a MBQP solver. A MBQP solver –as will be explained later– generates QP subproblems with different sizes. The different size QP problems generated should be handled using the implemented QP solvers, which means it must be able to deal with different problem sizes.

2.4.4 Notes about design parameters

In the current design single precision floating point number representation was used instead of double precision representation. This choice aims to compromise between solver accuracy and the solver and the amount of memory storage required by the design. However, it should be noted that using single precision representation may degrade the performance of the QP solver. For some QP problems with very large dynamic range the solver may fail to reach the optimal solution due to the loss of precision imposed by the singe precision representation. The solver may also need a larger number of iterations to reach the optimal solution. Example 2.6 shows an example QP problem where enlarging the dynamic range results in failure of the solution.

Example 2.6

Consider the inequality constrained QP problem [14]

$$\min J(q) = \frac{1}{2} q^T G q + q^T d$$
$$Aq \geq b$$

where

$$G = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad d = \begin{bmatrix} -2 \\ -5 \end{bmatrix}, \quad A = \begin{bmatrix} -1 & 2 \\ 1 & 2 \\ 1 & -2 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 6 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

The optimal solution of this problem is q* = (1.4, 1.7). If the dynamic range of the problem in increased -for example by multiplying all the coefficients of the first constraint by a factor of 10^{12}- we should obtain the same optimal solution. However, due to the loss of accuracy imposed by the use of single precision representation the solver will fail to reach the optimal solution.

This problem can be handled by providing a preprocessing stage to provide some sort of normalization to the problem constraints. The current design does not include such a preprocessing stage.

2.4.5 Comparison between target platforms

The current design targets FPGA as a fast prototyping method. However, FPGAs – in general- have some characteristics that limit the performance of the system. These limitations can be avoided if the design was implemented as an ASIC. One of the FPGA characteristics that limit the performance is its Logic Element (LE) nature. Each FPGA logic element consists of logic generating hardware and a number of flip-flops. If a LE flip-flops are used -for example- to generate registers to store data, the rest of the logic element hardware cannot be used. In the current design implementation, it is required to store intermediate calculation results for further processing. These intermediate results usually take vector form. When the storage is performed using registers, a large number of LEs is used. Most of them are used due to the use of its flip-flops only. This results in lack of utilization. Moreover, the FPGA area is consumed which prevents the implementation of more arithmetic components to speed up calculations.

To avoid this problem, intermediate variables are implemented as memories. Such implementation saves the number of LEs used as registers. However, it adds memory loading and storage delays. These delays reduce system speed.

If ASIC is used for implementation instead of FPGA, then it will be possible to implement the calculation intermediate results as registers without loss of utilization. It will save the time lost in memory loading and storing.

Another advantage of ASIC over FPGA appears in the implementation of arithmetic calculation units. In FPGA, these units are implemented using logic elements which reduce their speed. The speed is improved on FPGA by the use of dedicated multipliers implemented on the FPGA. However, if ASIC is used, it will be possible to implement faster arithmetic units which will significantly improve the solver performance.

2.5 Hardware implementation of MBQP solver

The success of the online implementation of MPC on hybrid systems depends basically on the implementation of a fast MBQP solver. If such a solver exists, the controller will be able to solve the complex optimization problem online and provide the optimal control move before the end of the sampling interval. Different algorithms that solve MBQP problem were discussed in section 1.5. The superiority of the branch and bound algorithm was shown therein. The details of the branch and bound implementation were given in algorithm 1.4. In this section we examine how this algorithm can be tailored to be suitable for hardware implementation.

The solution of a MBQP using branch and bound requires some basic operations which are: relaxation of binary constraints, calling of a QP solver, examination of the QP solution to check branching possibility, choosing a binary variable for branching, ability to branch on the selected variable, storing branched problems to the list for future solution and finally loading one of the problems in the list for solution. The hardware implementation of each of these operations will be discussed in detail.

2.5.1 Relaxation of binary constraints

One of the simple relaxations to the MBQP problem is the QP relaxation. In this relaxation the MBQP is relaxed to a QP problem by converting the binary variable $q_i \in \{0,1\}$ to a real variable in the range [0, 1]. This consequently adds the following two constraints to the problems

$$q_i \geq 0 \qquad (2.6a)$$

$$q_i \leq 1 \quad \Rightarrow \quad -q_i \geq -1 \qquad (2.6b)$$

Both inequality constraints are converted to the greater than or equal sign to match the formulation of the interior-point QP solver used.

It should be noted here that these two constraints are added for each relaxed binary variable. Hence, the relaxation of binary variables increases the number of constraints in the QP subproblem. The number of added constraints is twice the number of relaxed binary variables. The hardware implementation of this operation is done by modifying the matrices A and b that are passed to the QP solver. The constraint $q_i \geq 0$ is implemented by adding a complete row of zeros to the matrix A except at column i which corresponds to the relaxed variable q_i where 1 is added. The vector b is changed by adding 0 as a new row. The same is done for the constraint $-q_i \geq -1$. The difference is that -1 is stored instead of 1 in column i of the added row to the matrix A and also -1 is added instead of 0 in the vector b. The number of constraints m passed to the QP solver is increased by 2 for each relaxed binary variable.

2.5.2 Calling of QP solver

The relaxed MBQP problem should be passed to the interior point QP solver. The QP solver hardware is called as a component within the interior point hardware. To start the interior point solver, the matrices of the QP problem must be loaded to the interior point solver memories before starting the solver operation. To simplify memory loading, the interior point memories were shared between interior point hardware and MBQP hardware as shown in figure 2.11. The proposed memory sharing was implemented by multiplexing memory address, data and control signals from both sources to control the same memory. The multiplexers' selection signal is controlled by the MBQP hardware to determine which hardware has access to the shared memories. When the MBQP hardware processes a problem from the list and prepares for calling the interior point hardware, the shared memory is accessed by the MBQP hardware to update the matrices A, b, G, d that will be used by the QP solver. Once the interior point solver is started to solve the relaxed QP problem the memory access is transferred to the interior point hardware allowing it to access the QP problem matrices to start algorithm execution.

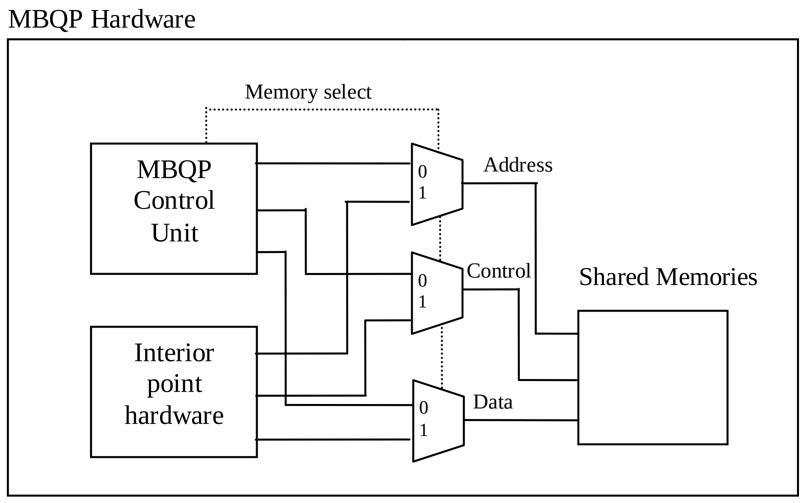


Figure 2.11: sharing of memories between interior point and MBQP

The shared memories architecture has two main advantages:

1) Reducing the number of the required memory bits. If memories are not shared, we would need separate memories for MBQP hardware to store processed A, b, G and d matrices. All these memories will be duplicated in the QP solver hardware.
2) Saving the time needed to pass quadratic programming matrices from MBQP hardware to QP solver Hardware. By simply changing the value of "Memory select" signal, the quadratic programming matrices are available for the QP solver to start its operation. We also save the time required to pass the solution obtained from the interior point solver to MBQP hardware.

We should also indicate here that the proposed design allows the use of parallel QP solvers as indicated before in figure 2.2. This property would have great importance if the average number of QP problems required for solution during MBQP is very large and consumes a large period of time. In this case, using multiple QP solvers will reduce the time for MBQP solution by a factor proportional to the number of the used QP solvers.

2.5.3 Examination of the QP solution

When the QP solver finishes its operation, the reached solution should be examined to determine whether we will continue or stop branching on this branch of the enumeration tree. At first, the QP solver result is checked if it is valid or not. This is done by reading the value of exit flags provided by the QP solver. If exit flags showed that the problem is infeasible then we stop branching in this branch as all subproblems will be also infeasible.

If the solution obtained from the QP solver is feasible, we check the value of the objective function obtained from QP solver. If the value of the objective function is larger than a previously reached upper bound, the solution is rejected and we stop branching on this branch. The reason behind this rejection is that the relaxed problem solution acts as a lower bound for all subproblems in that branch of the enumeration tree. If this lower bound is higher than a currently available upper bound of the MBQP problem, then all subproblems in that branch will not give any better solution to the MBQP problem. Hence, all subproblems of this branch are ignored. It should be noted here that the upper bound must be initially set to infinity to explore all possible problems.

Finally, if the QP solution is valid and has an objective function value less than the available upper bound, the solution vector is tested to examine if it is integer feasible or not. Testing is done by reading the locations in the solution vector corresponding to binary variables. The read elements are examined to determine if they can be considered binary (0 or 1) or not. Each element is compared with a threshold very near to one, i.e., $(1-\varepsilon)$. If the read element is greater than this threshold then it can be considered as one. Otherwise, the element is compared with a threshold very near to zero, i.e., (ε). If the element is smaller than this threshold, it will be considered as zero. The value of ε is the max error allowed in the QP solver that was used to calculate the relaxed solution.
This process is repeated for all binary relaxed variables. If all of them are binary, the obtained solution is a feasible solution to the MBQP problem. The value of the

objective function in this case can be used as an upper bound to the MBQP problem and to the relaxed QP problems and is used in the comparisons discussed before.
If any of the binary relaxed variables is neither near to 1 nor near to 0, then it is not satisfying the constraint of being binary. This variable is used for further branching as will be discussed in the following section.

2.5.4 Branching of a binary variable

Branching operation is one of the most important operations in the solution of a MBQP problem. Branching is the operation of selection of a binary variable and creating two subproblems with one having the chosen variable as 1 and the other having this variable as zero. Branching can be simply performed by adding an equality constraint to the problem. However, this will increase the number of constraints of the subproblem which means that it will need longer time to be solved compared to parent problem. Another problem is that adding this equality constraint may make the constraints dependent and this will cause a problem during QP solution. A third problem is that the existence of equality constraints along with inequality constraints will require a more complex algorithm than the case of inequality constraints only. This will cause the implementation to have more area and may lead to slower execution.
Instead of adding an equality constraint to perform branching, we can remove the branched variable from the optimization problem. By removing this variable, the resulting optimization problem will have a smaller size. Consequently, it requires shorter solution time. As a result, the solution becomes faster as we go down in the enumeration tree which improves the system behavior.

The process of removing a fixed variable from the optimization problem is very simple as the removed variable takes a value of 0 or 1. This method results in some change in the optimization problem matrices as will be shown in the following example.

Example 2.6

Consider the following optimization problem with three optimization variables (n=3) and four constraints (m = 4).

$$\min J(q) = \frac{1}{2}\begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}\begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{12} & g_{22} & g_{23} \\ g_{13} & g_{23} & g_{33} \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} + \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

If the variable q_2 is a binary variable and we need to perform branching on this variable. Then, the optimization problem can be written as

$$\min J(q) = \frac{1}{2}\begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} g_{11} & g_{13} \\ g_{13} & g_{33} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} + \begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} d_1 \\ d_3 \end{bmatrix} + \frac{1}{2} g_{22} q_2^{\,2} + \frac{1}{2} \times 2 \begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} g_{12} \\ g_{23} \end{bmatrix} q_2 + d_2 q_2$$

$$\begin{bmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \\ a_{31} & a_{33} \\ a_{41} & a_{43} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} + \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \end{bmatrix} q_2 \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

In this form we separated the variable q_2 as a preparation to its removal from the problem. If the variable q_2 is branched to zero, the last three terms of the objective function will vanish. Moreover, the second term of the LHS of the constraints inequality will vanish. The optimization problem reduces to an equivalent form which has only two optimization variables (n = 2) and the same number of constraints (m = 4). The resulting equivalent optimization problem has the form

$$\min J(q) = \frac{1}{2}\begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} g_{11} & g_{13} \\ g_{13} & g_{33} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} + \begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} d_1 \\ d_3 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \\ a_{31} & a_{33} \\ a_{41} & a_{43} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

The resulting equivalent optimization problem can be formed simply when the binary variable is fixed to 0. The new G matrix is formed by removing a row and a column from the original G matrix. The removed row and column are those related to the branched binary variable – second row and second column in the previous example. Similarly, the new vector d is formed by removing a row from the original d vector. The new matrix A is formed by removing the column corresponding to branched variable from the original A matrix. The vector b is not changed.

The case when the binary variable is branched to 1 differs slightly from the previous discussion. The difference is that the terms related to the branched variable will not vanish. Instead, they will change the optimization problem matrices. For the same example, when the branched variable is set to 1 the resulting optimization problem will be

$$\min J(q) = \frac{1}{2}\begin{bmatrix} q_1 & q_3 \end{bmatrix}\begin{bmatrix} g_{11} & g_{13} \\ g_{13} & g_{33} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} + \begin{bmatrix} q_1 & q_3 \end{bmatrix}\left(\begin{bmatrix} d_1 \\ d_3 \end{bmatrix} + \begin{bmatrix} g_{12} \\ g_{23} \end{bmatrix}\right) + \frac{1}{2}g_{22} + d_2$$

$$\begin{bmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \\ a_{31} & a_{33} \\ a_{41} & a_{43} \end{bmatrix}\begin{bmatrix} q_1 \\ q_3 \end{bmatrix} \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} - \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \end{bmatrix}$$

In this case, matrices G and A are the same as the case when $q_2 = 0$. However, vectors d and b have changed. The new d vector is the addition of the vector d in the previous case and a part of the removed column of the matrix G. The later column is the complete removed column except the row in the position related to the branched variable. The new b vector is the original b vector minus the removed column of the matrix A.

We should also note that a constant term $(\frac{1}{2}g_{22} + d_2)$ appeared in the objective function. This term affects the value of the objective function. But this term will not affect the value of the optimal solution vector q* and hence it is ignored during the optimization. As this term affects the value of the objective function and to have a correct operation, the value of this term should be computed and saved. It will be used when the QP solver finishes optimization. This constant term is added to the value of the objective function obtained from QP solver. This is done before comparing the value of the objective function with the available upper bound.

2.5.5 Storing and loading problem from list

The number of subproblems created during solution of MBQP increases exponentially as we move from a level to the lower level in the branch and bound enumeration tree. These subproblems have to be stored for future solution. A list of the subproblems to be solved is created. The main problem of this list is its large memory requirements. If we store the matrices of each relaxed subproblem, the memory needed will be huge. As the implementation currently targets FPGA, such a memory cannot be provided. Any FPGA has limited memory bits that cannot be exceeded.

To overcome this critical storage problem an alternative method is used. In this method, instead of storing the matrices of each subproblem, the list stores only the status of the binary variables in the subproblem. Also, a copy of the original MBQP problem matrices is stored in a separate memory. This copy is not altered and is always available.

During the processing of any subproblem, the status of the binary variables is loaded from the list. Each of the variables is processed according to its status. If the current manipulated variable is relaxed, we add the corresponding relaxation constraints to the original constraints of the problem. The new problem constraints are stored to the memory containing the matrix A of the shared memories. The storage of the new matrix A prepares for the solution of the resulting QP problem. If the current manipulated variable is fixed either by 1 or by 0, the original problem matrices are processed as described in example 2.6. The processed matrices are also stored to the shared memories.

The advantage of this storage method is that the memory storage needed by each subproblem becomes very small. Each binary variable needs only 2 bits to store its status which has only three options (relaxed, fixed with 0 or fixed with 1). Hence, the number of memory bits needed by each subproblem is $2n_b$. Reducing required design memory will allow the use of saved memory bits in implementing parallel QP solver units which can have a great impact on system's performance.

If we study the latency of the proposed MBQP solver, we find that it will require a small time. The reason is that most of the operations needed are memory read/write operations which can be executed quickly. Moreover, access to different matrices can be performed in parallel as we have each matrix stored in a separate memory.

The number of calculations required by the proposed branch and bound implementation is small. Calculations are needed only when a binary variable is fixed to 1. Moreover, these calculations are performed quickly by exploiting the pipelined structure of the floating point adder and multiplier used as described in example 2.3.

Due to the small computation requirements of the MBQP algorithm, its hardware uses a single floating point adder, single floating point multiplier and a single floating point comparator. It also uses some additional memories in addition to the shared memory. A group of small memories are used to store the original MBQP problem during the whole period of solution. These memories are used to calculate QP matrices. Additional small memory is used as the list used to store information about QP subproblems. The length of the list memory is related to the number of binary variables in the problem n_b and the method used in scanning the list.

In the proposed implementation, we use depth first branching policy. To perform this policy the list memory is implemented as last-in first-out LIFO memory. Depth first policy is selected as it moves fast toward the bottom of the enumeration tree where the probability to find integer feasible solution -hence an upper bound less than infinity- increases. Finding such an upper bound early can help us to ignore the solution of complete branches of the tree. Hence, it reduces the number of QP problems which in turn leads to a faster solution of the MBQP problem.

Conclusion

Chapter 2 gives the details of the hardware implementation of the main parts of the hybrid model predictive controller. The ideas introduced in this chapter aims basically to help the implementation of a fast hardware hybrid controller.

The chapter begins by explanation of the main parts of the hybrid controller and their role in determining the controller speed in section 2.1. The design environment details are then given in section 2.2. The concepts of pipelining and parallelism were reviewed in section 2.3. Various examples were given to illustrate how these concepts can be used to speed up calculations. Special interest was given to matrix operations as they represent the dominant calculations in the hybrid controller. The ideas of these examples were applied practically in the system hardware design.

The details of the hardware implementation of the interior point QP solver were given in section 2.4. The modifications performed to the algorithm to make it more suitable for hardware implementation were discussed. The implementation of the MBQP solver was discussed in detail in section 2.5.

Chapter 3 Results

In this chapter the performance of the hardware hybrid model predictive controller is evaluated. This evaluation is important as we target the control of embedded hybrid systems. This embedded nature sets some constraints on the controller performance. One constraint is related to the amount of memory needed by the controller in comparison to the hardware platform available memory. Another more important constraint is related to the controller speed. As the MPC control strategy is applied online during system operation, the time needed by the controller to calculate the control signal must be less than the sampling interval of the controlled system.

The main component that affects the performance of the hybrid controller is the MBQP solver. The operation of MBQP optimization is the most computation expensive in the algorithm of hybrid MPC. As we have discussed in section 1.5, the MBQP optimization problem is solved by the branch and bound algorithm which requires the solution of a sequence of QP subproblems. This shows that the performance of the MBQP solver depends mainly on the performance of the used QP solver.

In this thesis the branch and bound algorithm is used for the implementation of the MBQP solver. The QP solver was implemented using path following infeasible primal-dual interior point algorithm. The reasons of these selections were discussed in chapter 1.

In this chapter the performance of the implemented QP solver is discussed in section 3.1 followed by the performance of the MBQP solver in section 3.2.

3.1 Performance of QP solver

In this section we evaluate the performance of the QP solver by reporting its area requirements and its latency. We study various implementations of the QP solver and compare their area requirements and latency.

As we prototype our design using an FPGA, the area requirements can be divided to two categories: logic elements (LEs) requirements and memory requirements.

Logic elements are used to implement any arithmetic\control functions. Hence, the LE requirements are related mainly to the number of arithmetic units implemented and the design of the control unit. On the other hand, memory requirements are related to the amount of data storage needed. The number of required memory bits depends mainly on the size of QP matrices and hence depends on the QP problem size. In a certain FPGA, the number of available memory bits determines the maximum size of the QP solver that can be implemented. In section 3.1.1, we study a number of FPGAs and we list the size of the QP solver that can be implemented using each FPGA.

The latency of the QP solver depends on two factors; number of iterations performed by the QP solution algorithm and the single iteration time. The number of iterations depends on the problem structure, the level of accuracy required and on the QP solution algorithm used. It is not affected by the hardware design. However, the single iteration time depends basically on the hardware design as it depends on the method of the execution of the algorithm equations. Reducing the latency of the single iteration time by proper hardware design will improve the performance of the QP solver.

To test the implemented quadratic programming solver, some benchmark problems were used. The problems used for performance testing are part of a benchmark [64] provided by Tomlab Optimization for testing quadratic programming problems. The benchmark problems were provided in MPS (QPS) format. The MPS format is a standard file format for expressing mathematical programming systems. The MPS format targets basically Linear Programming (LP) problems and Mixed Integer Linear Programming (MILP) problems. The QPS format [65] is an extension to the MPS format to handle Quadratic Programming (QP) problems and Mixed Integer Quadratic Programming (MIQP) problems. There exist a number of tools to export problems in MPS (QPS) format to be available for use inside Matlab environment [66]. Then, the test problems are passed to the Modelsim hardware solver using the Matlab-Modelsim link [60].

3.1.1 Selecting an FPGA

It was shown in section 3.1 that the number of available memory bits determines the maximum size of the QP solver that can be implemented. Table 3.1 shows a list of FPGAs with the max QP size that can be implemented

Table 3.1: Maximum QP problem parameters for different FPGAs

FPGA part number	Available memory bits	Number of variables	Number of constraints
EP3C25F324	608256	32	128
EP3C40F324	1161216	32	256
EP3C120F484	3981312	64	256
EP2S180F1020	9383040	128	512

3.1.2 Comparison with respect to iteration time

The iteration time of the QP solver will depend on the hardware architecture used in the implementation. In this section, we will compare a number of architectures and report the iteration time obtained during the solution of a set of test problems.

Architecture I

In this architecture we try to minimize the used hardware. The minimum number of floating point arithmetic units is used. We use only a single FP adder/subtractor, single FP multiplier, single FP divider and single FP comparator. The used arithmetic units are pipelined units. Table 3.2 gives the characteristics of the units used when synthesized on EP3C25F324 FPGA.

In this architecture all steps of the interior point algorithm 1.3 will be executed in sequence. We use the pipelined execution manner described in examples 2.2 and 2.5 to speed up calculations. For example, to calculate $r_d = G\times q - A^T\times\lambda - d$, $G\times q$ is calculated until the execution ends and the result is stored. Then the same hardware is used to calculate $A^T\times\lambda$. The obtained result is added to the result of $G\times q$. Finally, d is subtracted and the final result of r_d is stored. After rd is calculated, the same hardware is used to calculate r_b. All steps of the algorithm will be calculated sequentially. Parallel execution cannot be used unless the calculated quantities use different arithmetic components. For example, the multiplication = G×q -in the calculation of r_d- can be executed in parallel with the division of λ/s -needed to calculate $A^T(S^{-1}\Lambda)A$- because these calculations use different hardware components. The synthesis report of the implementation of the QP solver using architecture I is given in Table 3.3. Table 3.4 gives the execution time of the QP solver for a set of test problems from the benchmark [64].

Table 3.2: Floating Points components parameters

Unit	FP adder	FP multiplier	FP divider	FP comparator
pipelining stages	7	5	6	1
Logic cells	843	196	224	85
registers	372	152	159	2
Memory bits	0	0	18	0
Multipliers	0	7	16	0

Table 3.3: synthesis report of architecture I QP solver

Logic cells	Registers	Memory bits	Multipliers	Maximum frequency
12236	2897	171538	23	99.6 MHz

Table 3.4: performance of architecture I QP solver

Test problem	n	m	Single iteration time	No. of iterations	Solution time
HS268	5	5	17.637 µs	26	0.458 ms
HS118	15	59	0.327 ms	20	6.45 ms

Results in table 3.4 were obtained using the cyclone III FPGA EP3C25F324 for implementation with a clock frequency of 80 MHz.

Architecture II

In this architecture, we use more hardware to reduce the execution time of the algorithm. The added arithmetic units are used to execute parallel calculations. For example, in figure 2.10, calculation $G \times q$, $A^T \times \lambda$ and $A \times q$ can be executed in parallel. Hence we can use three multiplier/adder pairs to perform the three mentioned calculations in parallel. Each multiplier/adder pair will work in the pipelined fashion described in example 2.5.

This architecture is very useful in reducing the time of the algorithm especially at calculations that are considered as a "bottleneck". A "bottleneck" calculation is a complex computation that takes a long time compared to all other calculations of the algorithm. The execution of this type of calculations requires a number of clock cycles -at least- an order of magnitude higher than other calculations. In the implemented interior point QP algorithm there exist two of such calculations; the matrix-matrix multiplication ($\text{temp\_mat} \times A = A^T(S^{-1}\Lambda)A$) in figure 2.10 and the Gauss-Jordan calculation. These calculations are the most time consuming calculations during the execution of the algorithm. Reducing execution time of both calculations using parallel design has a direct impact on reducing QP solver latency. The execution of a QP iteration using architecture II is illustrated in figure 3.1.

The execution of the calculations in step 1 requires 5 FP adders, 5 FP multipliers and one FP divider. Step 2 requires less numbers of adders and divider. Step 3 represents a bottleneck in calculation. The matrix-matrix calculation can be performed as a group of matrix-vector multiplication. Using parallel adder/multiplier pairs, these matrix-vector multiplications can be performed in parallel. If it is available to implement n adder/multiplier pairs the matrix-matrix multiplication is executed nearly in the same time of a matrix-vector multiplication. If the number of adder/multiplier pairs is equal to PU where PU is less than n, the matrix-matrix multiplication time will equal the time of matrix-vector multiplication multiplied by ceil (n/PU). Step 5 where Gauss-Jordan Algorithm is executed has similar execution behavior to that of step 3. All other steps require a number of adders and multipliers less than 5.

For the current implementation the number of clock cycles required to complete a QP solver iteration -using architecture II- is related to the number of the optimization variables (n) and the number of constraints (m) by a complexity of O((m + n)×n×ceil(n/PU)).

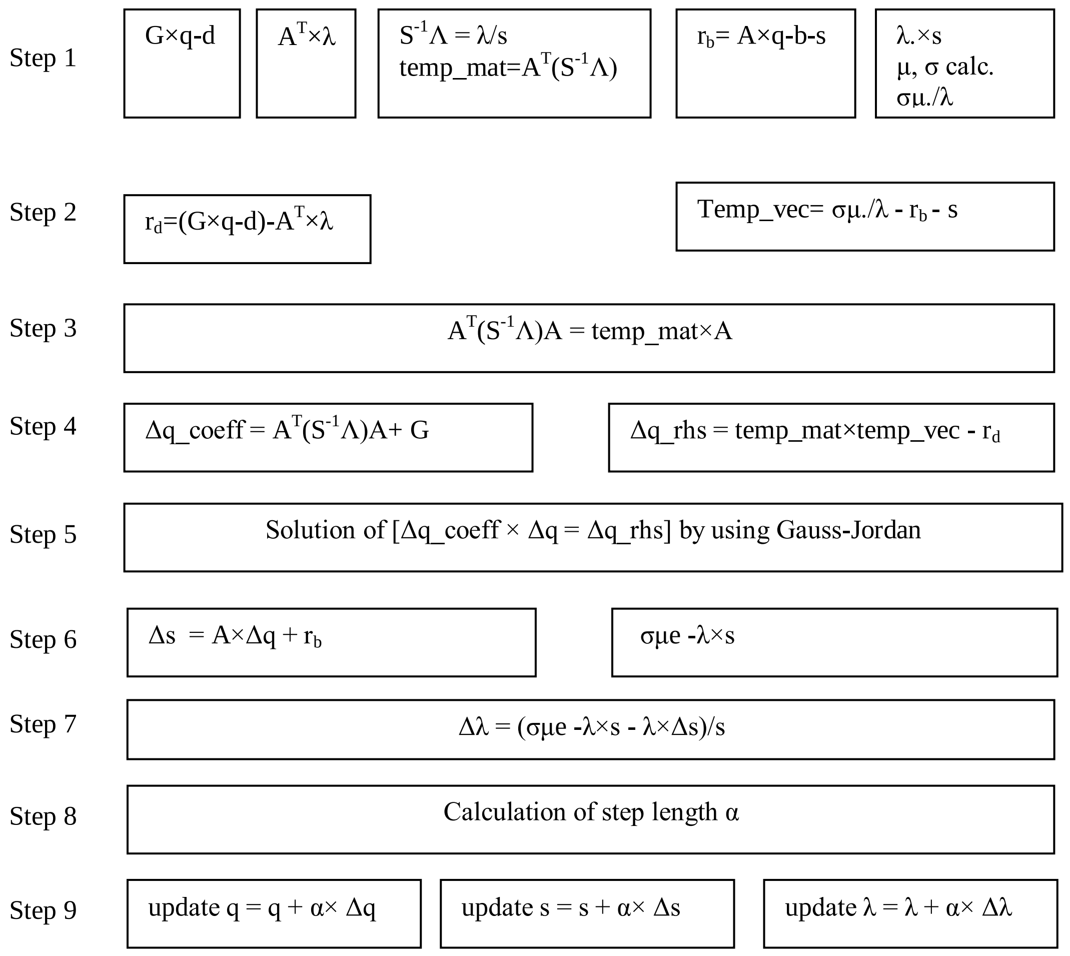


Figure 3.1: Execution of a QP solver iteration using architecture II

The implemented QP solver uses 5 FP adders, 5 FP multipliers, one FP divider and one FP comparator. It is designed to solve a QP with maximum size (n=16 and m=64).The parameters of the arithmetic components used are the same as those in table 3.1. Table 3.5 gives synthesis report for the system. The hardware execution time for a set of test problems – same as those used to evaluate architecture I- is shown in table 3.6.

Table 3.5: synthesis report of architecture II QP solver

Logic cells	Registers	Memory bits	Multipliers	Maximum frequency
18613	7039	171538	51	91.46 MHz

Table 3.6: performance of architecture II QP solver

Test problem	n	m	Single iteration time	No. of iterations	Solution time
HS268	5	5	6.45 μs	26	0.167 ms
HS118	15	59	0.105 ms	20	2.11 ms

3.1.3 Evaluation of solver performance

The performance of the hardware QP solver is compared with the performance of a Matlab function that implements the same algorithm. Table 3.7 reports the execution time of the Matlab function solving the same test problems. The function is implemented using Matlab version 2007B that runs on windows 7 on a labtop with intel I5 processor with clock frequency 2.26 GHz.

Table 3.7: Performance of Matlab function of the QP solver

Test problem	n	m	Average Solution time
HS268	5	5	4.4 ms
HS118	15	59	8.8 ms

It is clear that the execution time obtained by the hardware QP solver is less than the time obtained by the QP Matlab function. Another main advantage of the hardware implementation is the power consumption. The estimated FPGA power consumption does not exceed 0.25 W while the power consumption of the PC is several watts. The FPGA power consumption is estimated using the Quartus II software version 8.1 assuming a toggle rate of 80×10^6 transitions/sec for the input signals and vectorless estimation of the toggle rate of the remaining signals.

Our proposed QP solver is compared to the QP solver implemented in [7]. The implemented solver in that paper solves a QP problem with n = 3 and m = 60 in about 20 ms. Our solver solves the HS118 test problem which has larger size in a much shorter time.

Note about Matlab function results

Results of table 3.7 show that although the HS118 problem is about 30 times larger in size than the HS268 problem, its execution time on Matlab is only twice longer. The reason is that HS118 matrices contain a large number of zeros. It was found by experiment that Matlab performs faster when working with zeros than when working with any other numbers. A problem of the same size as the HS118 will have a longer execution time on Matlab if its matrices have a fewer number of zeros than the HS118 problem.

3.2 Performance of MBQP solver

When we study the performance of the MBQP solver, we study the hardware requirement of the MBQP solver independent on the embedded QP solver. Also, we measure the time required by the MBQP solver only.

For hardware area, the MBQP uses very simple hardware. It uses single floating point adder, multiplier and comparator. The control unit that controls the operation of MBQP solver is a simple FSM. Memory requirements of the MBQP solver are minimized by using the same memory of the interior point unit. The additional memory requirements are represented by two memories. One to store original MBQP matrices, the other is the list that stores the MBQP subproblems. The hardware used by the MBQP solver is shown in table 3.8

Table 3.8: synthesis report of MBQP solver

Logic cells	Registers	Memory bits	Multipliers	Maximum frequency
3869	1426	77824	7	91.46 MHz

For time requirements, it was found that the percentage of time needed by MBQP hardware with respect to the time needed by the QP solver does not exceed 1% in all test problems used.

Conclusion

In this chapter the performance of the hardware hybrid controller is evaluated. Two main factors are used in the performance evaluation: the amount of required hardware resources and the solver latency. Hardware resources requirements are obtained from the design synthesis report. This report determines the FPGA device that should be used. The solver latency determines the maximum sampling time that can be used. Hence, it determines the class of hybrid systems that can be controlled using the proposed controller.

The chapter begins by the evaluation of the performance of the hardware QP solver. QP problem solution is the most time consuming operation during optimization. Moreover, the MBQP problem is solved by solving a series of QP subproblems. To allow the implementation of the proposed solver using different types of FPGA, two architectures were studied. One of these architectures minimizes the needed hardware. The other applies more hardware resources to reduce the QP solution time. The latency of the QP solver implementation using both architectures was reported for a set of test problems. The obtained results are compared to a previously implemented solver [7]. Results are also compared to the execution time of the same algorithm on a PC.

The chapter concludes by reporting the performance of the MBQP solver. Its performance is evaluated in terms of hardware requirements and solution latency.

Chapter 4: Practical Application: Control of a Stick-Slip Drive

4.1 Introduction

Stick-slip drives are becoming popular in the domain of micro and nanotechnologies because of their simple and compact structure, fast response and high resolution [67]. As can be understood from their names, these drives work in two distinct operation modes 'stick' and 'slip'. Stick and slip behaviors due to the friction force in mechanical systems. As friction can be modeled as a hybrid system, consequently, stick-slip drives can also be modeled in a hybrid framework. In fact, hybrid modeling is very suitable for these kinds of systems, since it clearly reflects their operation principle.

In this chapter, a hybrid modeling for the Stick-Slip Inertial Driver "SSID" is proposed. The resulting model is used to build an MPC for the system to achieve a certain required performance. The rest of this chapter is organized as follows. First, in Section 4.2 a brief representation of such drives will be presented. Then, the description of a particular stick-slip setup and its working principle will be explained. In Section 4.3 the hybrid modeling of a stick-slip drive will be introduced. The control design of the drive via MPC will be discussed in Section 4.4. Simulation as well as experimental results will be presented in Section 4.5. Finally, some details about the hardware setup is introduced in section 4.6.

4.2 Impact and Stick-Slip Drives

Inertial or impact drives are well developed in the domains of Microsystems and micro-manipulators. They are intensively used in precise positioning and especially for the probe positioning in scanning tunneling microscopes (STM). These drives have a simple and compact structure, provide high resolution up to some nanometers, and generate long range movements with relatively high velocity. In addition, mechanisms with multiple degrees of freedom can be easily constructed. In Figure 4.1, an impact drive is illustrated. It consists of three basic parts: a main body to be moved, an actuator (piezoelectric actuator) and an inertial mass. When the actuator makes a rapid extension, a strong force is generated which exceeds the friction force between the moving object and the table. Thus the main body will be displaced. When the actuator makes a slow contraction, the force is smaller than the static friction so that the main body does not move. Repeating these fast and slow actuator movements carries out the motion.

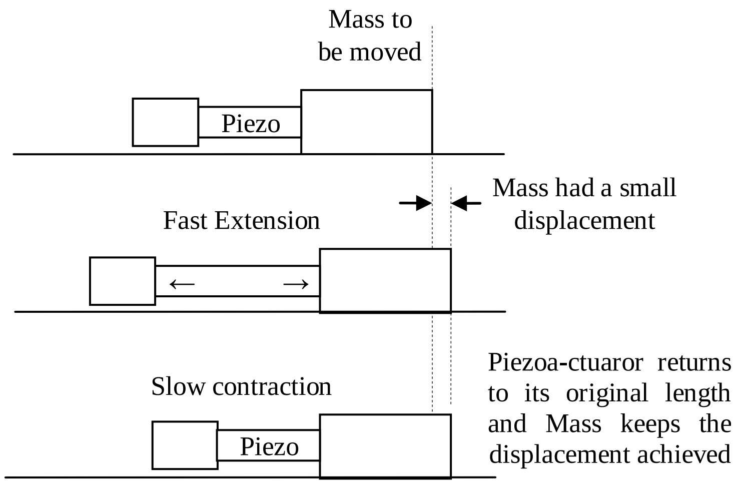

Mass to be moved
Piezo
Fast Extension
Mass had a small displacement
← →
Slow contraction
Piezo
Piezoa-ctuaror returns to its original length and Mass keeps the displacement achieved

Figure 4.1: Operation of Inertial Driver

Stick-slip actuators work by a similar principle and thus can be considered as a special case of impact drives. Stick-slip drives are distinguished from impact drives when one of the masses in the system is negligible compared to the other. Different applications and prototypes of such a technology can be found in the literature [68]. For instance, stick slip micromanipulators which give resolutions of some nanometers were designed and proposed in [69]. A prototype of a stick-slip drive [70] will be used an example for practical control of hybrid system. The description of this drive and its working principle will be explained next.

4.2.1 Description of a Stick-Slip Inertial Drive

A model for the stick-slip inertial drive [70] is given in Figure 4.2. It consists of a main body (M_m) whose position is to be controlled, a small mass (m_s) which has contact with M_m, a piezoelectric actuator and transmission elements that link the piezoactuator to the small mass. These transmission elements can be modeled as a spring with a high spring constant (K_s). The main mass is placed on a rolling table.

4.2.2 Working Principle

The drive operates in two different modes: 'stick' and 'slip'. Existence of these two distinct operational modes is due to the friction force between the small mass and the main mass. The two operational modes of the SSID are:

Stick mode: By applying a voltage input to the piezoactuator, the piezoelectric crystal expands and generates a force u_p on the small mass m_s. If the force generated is smaller than the friction force between the small mass m_s and inertial mass M_m, both masses will move together.

Slip mode: After the expansion of the piezoelectric crystal, if the input voltage is reduced suddenly, the piezoactuator force drops quickly. Consequently, the force acting at the frictional interface exceeds the static friction and the small mass slides on the inertial mass back to its initial position $x_{ms} = 0$. Since the small mass is lighter than the main mass, during the same time interval the inertial main has a much shorter backward movement.

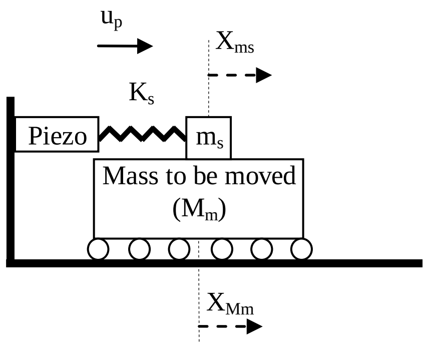


Figure 4.2: Model of SSID

To be able to move the main mass to a certain position we notice that during mode 'stick' both masses are attached and thus, they move together. However, as the range of elongations of the piezoactuator and consequently the small mass is limited, the main mass cannot be displaced more than by a certain limit. Then a resetting action is needed. The act of resetting occurs in the 'slip' period during which the small mass slides on the main mass back to its initial position. After resetting the position of the small mass position, the main mass can be displaced more by operating the system in mode ‘stick’ again. This way the main mass can be freely displaced on the rolling table even outside the movement range of the piezoelectric actuator.

4.3 Hybrid Modeling of the SSID

The source of the hybrid behavior of the SSID is its differentiation between stick and slip friction states. This hybrid behavior can be considered in system modeling by the use of Coulomb friction model [71]. This model defines friction force between the main mass and small mass ($\mathcal{F}$) as a function of their relative velocity ($v_r = v_{ms} - v_{Mm}$), where $\mathcal{F} = Fc*\mathrm{sgn}(v_r)$ and Fc is coulomb friction limit. This behavior is shown in Figure 4.3.

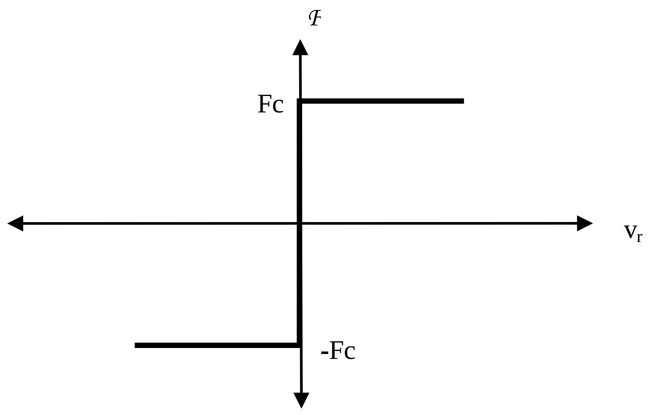


Figure 4.3: Coulomb Friction force model

Other Dynamics of the SSID can be derived from basic principles of mechanical systems. Some assumptions will be made to simplify system modeling and to express some practical system limitations. These assumptions are:

1. The friction between the main mass and the rolling table is neglected.
2. The dynamics of the piezoelectric actuator are much faster than the dynamics of the rest of the system and thus the input voltage to the piezoelectric actuator and the resulting generated force u_p are simply related by a gain.
3. The force generated by the piezoactuator u_p is related to the position of the piezo crystal, x_{piezo}, by the equality $u_p = k_p*x_{piezo}$. Since the maximum elongation of the piezo crystal, $\max\{x_{piezo}\}$, is limited by the dimension and the

load of the system, it will be assumed that the input force is bounded $u_{p\,max} = \max\{k_p * x_{piezo}\}$.

The model of the SSID is given by

$$\dot{x}_{ms} = v_{ms} \tag{4.1a}$$

$$\dot{v}_{ms} = \frac{1}{m_s}(u_p - k_s x_{ms} - \mathcal{F}) \tag{4.1b}$$

$$\dot{x}_{Mm} = v_{Mm} \tag{4.1c}$$

$$\dot{v}_{Mm} = \frac{1}{M_m}\mathcal{F} \tag{4.1d}$$

where x_{ms} and x_{Mm} represent the positions and v_{ms} and v_{Mm} represent the velocities of the small mass and main mass respectively, with m_s and M_m being their respective masses. u_p is the force generated by the piezoelectric actuator, k_s is the spring constant of the transmission elements, and $\mathcal{F}$ is the friction force between the two masses.

The hybrid modeling is achieved by applying the Coulomb friction model and its relation to the external force applied to the system. Using the coulomb model we find that friction on the interface between main mass and small mass has different definitions and the system can be in one of three different modes (discrete states): stick, slip-, and slip+.

To get the conditions of transition between these modes we can denote ρ as the force acting at the frictional interface due to the external input force u_p. If ρ does not exceed the Coulomb friction level Fc, then both the main and the small mass move together with $v_r = 0$. In this case mode 'stick' is active and friction force $\mathcal{F}$ is defined by ρ. As soon as ρ exceeds the Coulomb friction level, one mass slips over the other with $v_r = v_{ms}$-$v_{Mm} \neq 0$. The two modes slip+ and slip- are respectively distinguished by positive and negative relative velocities, i.e. $v_r > 0$ or $v_r < 0$.

The expression of ρ can be derived from evolution of the relative velocity, $v_r = 0$. Setting $\dot{v}_{ms} = \dot{v}_{Mm} = 0$ leads to

$$\rho = \frac{M_m}{M_m + m_s}(u_p - k_s x_{ms}) \tag{4.2}$$

In mode stick the friction force is defined by $\mathcal{F} = \rho$ and in modes slip- and slip+ by $\mathcal{F} = \text{-Fc}$ and $\mathcal{F} = \text{Fc}$ respectively. By using the definition of friction force in each mode, the vector fields

F_p (x; u) with x = $(x_m\ \ v_m\ x_M\ \ v_M)^T$, p= {stick, slip-, slip+} are defined by

$$F_{stick} = \begin{bmatrix} v_{ms} \\ \dfrac{(u_p - k_s x_{ms})}{M_m + m_s} \\ V_{Mm} \\ \dfrac{(u_p - k_s x_{ms})}{M_m + m_s} \end{bmatrix} \tag{4.3a}$$

$$F_{slip-} = \begin{bmatrix} v_{ms} \\ \dfrac{(u_p - k_s x_{ms} + Fc)}{m_s} \\ V_{Mm} \\ \dfrac{-Fc}{M_m} \end{bmatrix}, F_{slip+} = \begin{bmatrix} v_{ms} \\ \dfrac{(u_p - k_s x_{ms} - Fc)}{m_s} \\ V_{Mm} \\ \dfrac{Fc}{M_m} \end{bmatrix} \tag{4.3b,c}$$

The transitions between these modes and the definition of the friction force $\mathcal{F}$ in each mode are given by the hybrid automaton of Figure 4.4.

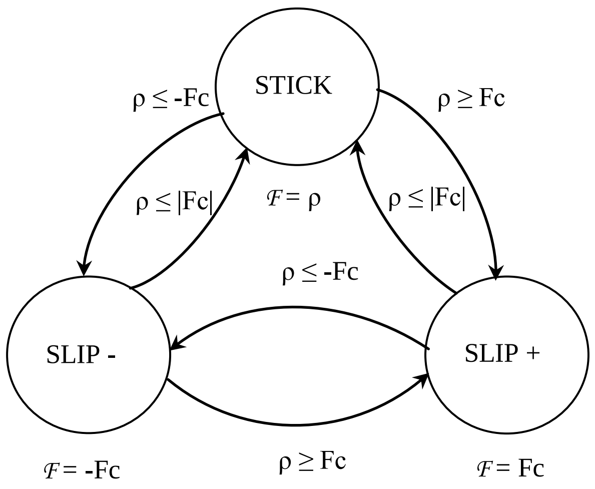


Figure 4.4: SSID Hybrid Automaton

4.4 Control of the SSID

As the main application of inertial drive systems is in micro positioning especially for the probe positioning in scanning tunneling microscopes (STM), The SSID control problem discussed here aims to regulate the position of main mass from an initial position to a certain required reference position. Usually, the required reference position is in order of micrometers or even parts of a micrometer depending of the parameters of SSID used. In this example, we will consider the control design required to move the main mass M_m from its initial position (X_{Mm} = 0) to a reference position r_{xM}.

The control strategy used should be able to handle the hybrid nature of the system. Moreover, it should respect the physical constraints of the system operation, for instance, the constraint that the input external force u_p cannot exceed a certain limit. In [71] the control design of SSID was based on a "dehybridization" approach where a cascade control scheme is applied for the problem of set point tracking of the SSID. In this method, an inner loop switching controller is designed to handle the system hybrid nature. Another outer loop controller is designed based on averaging the inner loop response. The outer loop control design is performed using classical control methods.

In this thesis, we propose the application of MPC to the set point tracking problem of the SSID. This proposal was inspired by the restrictions that exist on the control algorithm especially the need to handle explicit constraints ….. The application of such a control technique requires in the first place the modeling of the targeted hybrid system in the MLD framework, which in turn requires the description of the hybrid system behavior in HYSDEL format.

As MLD modeling deals with discrete time hybrid system, the previously mentioned model of the SSID should be discretized using a proper sampling interval. A straight non-optimized HYSDEL modeling of the SSID system with a sampling interval of 1ms is given in appendix A.1. This straight HYSDEL model uses large number of binary and continuous auxiliary variables, and then it will have also a large number of mixed-binary inequality constraints. Such a modeling approach will make the MLD model and the constructed controller inefficient for practical implementation. The MLD model can be simplified by changing the definition of the used binary variables– as shown in the HYSDEL model in appendix A.2. The proposed change reduced the number of binary variables from 3 variables to only 2 variables. This reduction of the number of the binary variable is logical as we have only 3 states. This change reduces the number of inequality constraints significantly which results in a much simplified MLD model. This discussion shows that a slight change in the definition of binary variables can result in an optimized MLD model.

It should be clear that the performance of the MPC algorithm and the ability of its practical implementation will basically depend on the efficiency of the MLD modeling. Simple MLD models will allow for more efficient control algorithms. It will allow also for faster implementations of the MPC controller.
By studying the modes of the SSID system and its dynamic properties two simplifications can be made to the system MLD modeling.

Simplification 1: reducing the number of modes (discrete states)
By inspecting the modes of the SSID system, we can find that the main mode that can contribute in increasing the position of the main mass is the stick mode. It was shown in section 4.2.2 that after stick mode is applied a reset of the small mass position should be performed. This reset is performed by "slipping" the small mass over the surface of the main mass. Mathematical modeling shows that there exist two slip modes: 'slip+' and 'slip-'. By inspecting the conditions of both slip modes, we find that 'slip+' mode requires positive relative velocity v_r. This means that the small mass will slip to the positive direction of motion in a velocity faster than that of the main mass. This means that the small mass will not return to its initial position ($x_{ms} = 0$). Moreover, this direction of slip can result in the violation

of the constraint of maximum allowed displacement allowed for small mass which is implied from the maximum allowed elongation of piezoelectric actuator.
On the other side, the ‘slip-’ mode requires negative relative velocity. This means the small mass will slip in the negative direction of motion with a velocity greater than that of the main mass. This direction of slipping will make the small mass return to its initial position and allow for a new ‘stick’ mode to push the main mass farther.
This discussion shows that the two modes that will tend to increase the distance of the main mass is the ‘stick’ and ‘slip-’ modes. The ‘slip+’ will not help increasing x_{Mm} and may lead to violation of the system operation constraints. Hence, the system modeling can be simplified by avoiding the ‘slip+’ mode. This is done by imposing an additional constraint on the system operation. The new constraint limits the friction force at the interface between the two masses ρ to be always less than the positive limit of the coulomb friction force +Fc. Although adding a constraint adds to the complexity of the MLD model, the new added constraint will limit the number of modes to two modes instead of three modes which has a greater impact on reducing the complexity of the MLD model.

Simplification 2: reducing the number of continuous time states
The original model of the SSID contains four states: x_{ms}, v_{ms} ,x_{Mm} and v_{Mm}. Four state equations are used to describe the behavior of both the small and main masses. Two state equations are associated with each mass. The most important dynamics of the system is the main mass dynamics as it is the quantity to be controlled. By providing a deeper study to the small dynamics in different modes of operation we can have two observations:

1. During ‘stick’ mode both masses all linked together and move as a single mass. This means that during the ‘stick’ mode the dynamics of the small mass and dynamics of main mass are the same. These dynamics can be obtained by the introduction of a combined mass – a mass whose value is the sum of both masses. As this combined mass has a much larger value than the small mass,

the system dynamics in this mode is slower than the dynamics of the small mass alone.

2. During ‘slip-’ mode the small mass slips over the main mass to its initial position. Hence, at the end of the operation of ‘slip-’ mode we have $x_{ms} = 0$. The small mass dynamics in this case is much faster than the ‘stick’ mode. The reason is that the dynamics here depend only on the value of the small mass only which is much smaller than the main mass. Consequently, the small mass will return from its current position to its initial position in a very small time. This time is much smaller than the time needed to reach the current position in stick mode. This discussion shows that the dynamics of the small mass in ‘slip-’ mode is approximately instantaneous with respect to its dynamics in ‘stick’ mode.

   Mathematical studies on the hybrid model of the SSID in [71] shows that the time that the system can stay in ‘slip-’ mode cannot exceed a certain limit given by

$$T_{\max}^{slip-} = 2\pi\sqrt{\frac{m_s}{k_s}} \qquad (4.4)$$

   This study shows that choosing a sampling interval longer that $T_{\max}^{slip-}$ guarantees that the system will stay in the ‘slip-’ mode for a single sampling instant. At the end of this sampling instant we are guaranteed that $x_{ms} = 0$.

These two observations will help us reduce the complexity of the SSID model by reducing the number of equations needed to model the dynamics of the small mass in different system modes. In ‘stick’ mode, the small mass position is controlled by the dynamics of main mass. In ‘slip-’ mode the small mass position is guaranteed to be zero after one sampling instant, i.e., $x_{ms}(k+1)= 0$ provided that the sampling interval used exceeds $T_{\max}^{slip-}$. By applying these ideas to the behavior of the small mass, the system can be modeled using three state equations only; two state equations describing behavior of main mass and a single equation describing the simplified behavior of small mass position x_{ms}.

Applying the proposed simplifications the resulting hybrid automaton is shown in figure 4.5.

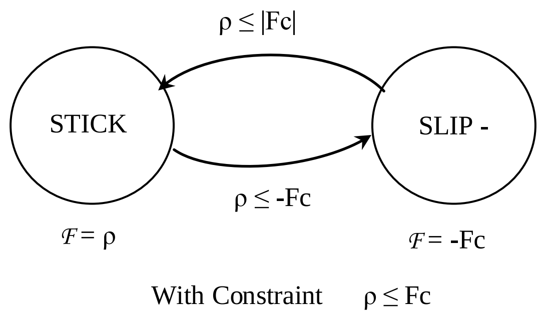


Figure 4.5: Simplified SSID Hybrid Automaton

It should be stated here that in addition to the constraint on ρ shown in the automaton 4.5, there are some other constraints resulting from the nature of the SSID system which are

$$x_{ms} \geq 0 \tag{4.5a}$$
$$u_p \geq 0 \tag{4.5b}$$
$$u_p \leq u_{p_{\max}} \tag{4.5c}$$

The first constraint shows that the small mass cannot get beyond its reference position. The second constraint shows that the force generated by the piezoelectric actuator is always positive. The third constraint shows that the piezoelectric actuator force cannot exceed a certain limit. A HYSDEL model describing the simplified SSID is provided in appendix A.3. The resulting MLD model is given by

$$x(k+1) = A_x x(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) \tag{4.6a}$$
$$y(k) = Cx(k) + D_1 u(k) + D_2 \delta(k) + D_3 z(k) \tag{4.6b}$$
$$E_2 \delta(k) + E_3 z(k) \leq E_1 u(k) + E_4 x(k) + E_5 \tag{4.6c}$$

where

$$A_x = \begin{bmatrix} -0.02667 & 0.00035 & 0 \\ -5866.667 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad B_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad B_3 = \begin{bmatrix} 1 & 0 & 1.02667 \\ 0 & 1 & 5866.667 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad D_1 = 0 \quad D_2 = 0 \quad D_3 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$E_1 = \begin{bmatrix} -0.95238 \\ 0.95238 \\ -5.8333e-8 \\ 5.8333e-8 \\ 0 \\ 0 \\ -3.333e-4 \\ 3.333e-4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ -0.95238 \end{bmatrix} \quad E_2 = \begin{bmatrix} -1747.33 \\ 1662.19 \\ 4.3575e-6 \\ -8.575e-7 \\ 8.575e-7 \\ -4.3575e-6 \\ 0.0249 \\ -0.0049 \\ 0.0049 \\ -0.0249 \\ 0.0001 \\ 0.0001 \\ -0.0001 \\ -0.0001 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad E_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$E_4 = \begin{bmatrix} 1.676e7 & 0 & -1.676e7 \\ -1.676e7 & 0 & 1.676e7 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1.676e7 & 0 & -1.676e78 \end{bmatrix} \quad E_5 = \begin{bmatrix} -14 \\ 1.676e3 \\ 4.3575e-6 \\ -8.575e-7 \\ 8.575e-7 \\ -8.575e-7 \\ 0.0249 \\ -0.0049 \\ 0.0049 \\ -0.0049 \\ 0.0001 \\ 0.0001 \\ 0 \\ 0 \\ 0 \\ 60 \\ 14 \end{bmatrix}$$

4.4.1 Hybrid MPC control problem formulation

After reaching a MLD model describing the hybrid system to be controlled, the construction of an MPC control strategy to control the system will not be a hard task.

The objective of the control problem here is to force the position of the large mass x_{Mm} toward a certain required reference r_{xM}. Hence, the objective function used for optimization in the MPC control strategy should penalize the difference (x_{Mm} - r_{xM}). The objective function will be in the form

$$\min_{u_p(k|k),\ldots u_p(k+T-1|k)} J = (x_{Mm} - r_{xM})^T Q_x (x_{Mm} - r_{xM}) \qquad (4.7)$$
$$\textit{subject to} \quad (4.12)$$

The hybrid control toolbox can be used to transform the system to the form (1.17). The matrices $\bar{G}, \bar{d}, \bar{D}, \bar{A}, \bar{b}$ *and* $\bar{C}_x$ of the form (1.17) are stored in the hybrid MPC controller. Every sampling instant, the values of the system states are measured. These values along with the required references are passed to the MPC controller. The Controller uses these values to form the optimization problem in the MBQP standard form (1.18) according to equations (1.19). This problem is then solved by the MBQP solver to obtain the control sequence u_p that minimizes the required objective function. Model parameters, controller parameters along with the simulation results are presented in section 4.5. Detailed hardware setup is discussed in section 4.6.

4.5 SSID system Simulation Results

The parameters of the simulated SSID model are given in Table 4.1 [71]. The Parameters of Model predictive controller are given in Table 4.2. Hardware implementation results are shown in Table 4.3.

Table 4.1: parameters of the SSID system

M_m	M_s	k_s	Fc	u_{pmax}	T^{slip-}_{max}	Ts
1 kg	0.05 kg	1.76×10^7 Nm^{-1}	14 N	60 N	0.335 ms	1 ms

Table 4.2: parameters of the MPC controller

Q_x	Prediction horizon T
1.9×10^7	1

Table 4.3: Hardware results of SSID problem

FPGA Type	parallel QP solvers	MBQP Solution time
EP3C25F324	1	0.39 ms
EP3C40F324	2	0.205 ms

The system is required to track a position reference r_{xM}. The reference starts by a value of 40 μm then the reference is changed to 80 μm. Simulation results are given in Figure 4.6.

The obtained results show that the proposed MPC control strategy was able to handle the hybrid nature of the system as it selects the correct mode of operation. The SSID system was made to track the required reference in a very short time. Moreover, The MPC control respected the physical constraints of the system.
To evaluate the performance of the MPC approach, the obtained performance is compared to the results obtained by the dehybridization approach used in [71]. The performance obtained by the dehybridization approach is shown in figures 4.7 and 4.8 with the SSID is required to track a reference of 50 μm.

By comparing both responses we find that both respect system constraints. However, the MPC approach achieved much faster tracking to the required set point.

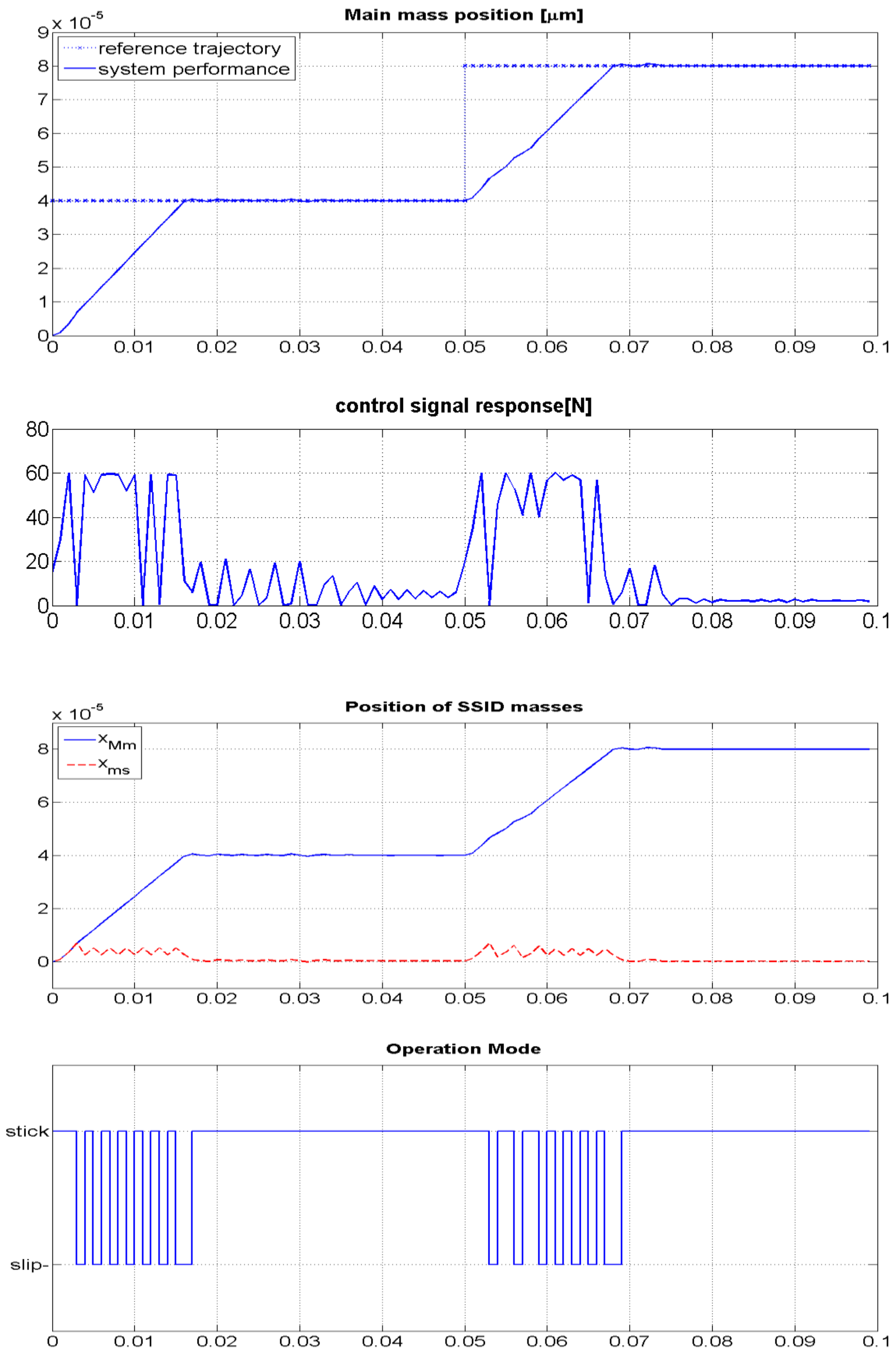


Figure 4.6: simulation results of SSID system

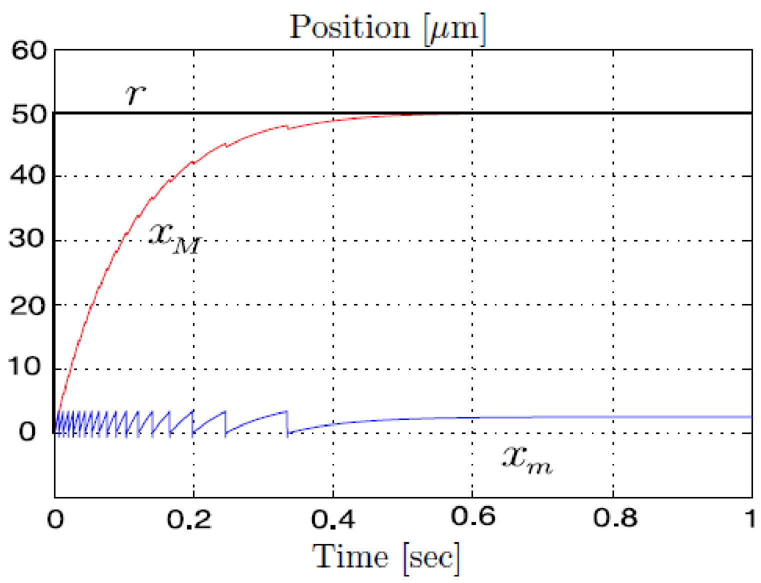


Figure 4.7: Masses positions of SSID with dehybridization control

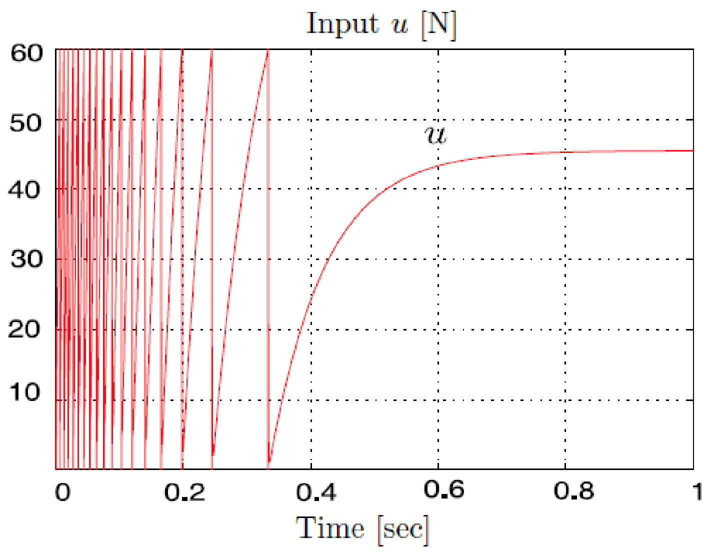


Figure 4.8: Control signal response using dehybridization

4.6 Detailed hardware setup

In this section we give more details about the hardware setup of the implemented hybrid model predictive controller. We clarify the control parameters that should be determined prior to FPGA programming and those that are used during online operation.

1. **Parameters that are determined prior to FPGA programming**

   FPGA programming is performed after reaching form (1.17). When this form is obtained, the size and the matrices of the MBQP problem are known. The matrices $\bar{G},\bar{d},\bar{D},\bar{A},\bar{b}$ *and* $\bar{C}_x$ of the form (1.17) are stored in controller memory prior to the online operation of the controller.

   Form (1.17) is obtained by the hybrid control toolbox after the MLD modeling of the hybrid system and determining all the parameters of the control objective function. These parameters include the weighting matrices, required references, limits on the controlled or manipulated variables, the prediction horizon and the sampling time. The most critical of these parameters are the prediction horizon, the number of constraints introduced in the control objective and the sampling time. The reason of their importance is that any change of these parameters results in a change in the hardware implementation. For example, any change in the prediction horizon or the number of constraints will change the size of the resulting MBQP problem. A change of the MBQP problem size may require changing the size of the implemented storage memory. Hence, it may require selecting a different FPGA for implementation. Also, the required sampling time determines the adopted hardware architecture and the number of required parallel units.

   As a result, the most important parameters to determine before FPGA programming are the prediction horizon, the number of constraints in the control objective and the sampling time. Other parameters like weighting matrices and type of constraints does not change MBQP problem size. However, they change only the values of the matrices $\bar{G},\bar{d},\bar{D},\bar{A},\bar{b}$ *and* $\bar{C}_x$. A change of these parameters will not require a reconfiguration of the FPGA. It will require only a reloading phase where the mentioned matrices are rewritten to the controller memory.

2. **Parameters used during online optimization**

   During online operation of the MPC controller, the system states are measured and the references are passed to the controller. The controller hardware receives these data and generates the standard MBQP problem (1.18) according to equations (1.19). The generated MBQP problem is then solved by the MBQP solver to generate the optimal control action which is applied to the hybrid system.

Conclusion

In this chapter the model predictive control of a practical hybrid system is introduced. The Stick-Slip Inertial Drive (SSID) is introduced as an example of hybrid systems where MPC can be applied to control the system while satisfying its operation constraints. The chapter starts by showing the importance of the SSID system and it practical applications. Then in section 4.2 the physical structure of the system is shown with a description of its working principle. Section 4.3 gives the modeling equations of the SSID system. The hybrid nature of the system was model in the form of a hybrid automaton. In section 4.4 the control problem of the SSID is discussed. Previous control strategies were reviewed. Model predictive control was shown to be a suitable control approach to the proposed control problem. For sake of implementation of MPC, the SSID system was modeled using MLD modeling framework. Some simplifications to the model were proposed to allow for efficient MPC control design. The simulation results and hardware results of the implementation were presented in section 4.5. These results show that the proposed implementation of hybrid MPC was applied successfully to the SSID control problem. Finally, section 4.6 gives some details about the hardware implementation.

Chapter 5 Conclusion

The objective of this thesis is to study the control problem of hybrid embedded systems. The proposed controller implementation should consider the size limitation of the embedded system. It should be easily integrated within the hybrid embedded system. This thesis proposes an on-chip implementation of a hybrid controller.

In this thesis, the Stick-Slip Inertial Derive was used as an example of a practical hybrid embedded system. To design a suitable controller for the used hybrid system, a suitable modeling framework should be used. The used modeling technique should be able to express both the continuous and the discrete dynamics of the hybrid system in a unified mathematical form. In this thesis, mixed-logical dynamical modeling framework was used to model the hybrid system. It was able to accurately describe all aspects of the hybrid system dynamics.

To control the example system, model predictive control was proposed. MPC technique was successful in achieving the required performance while dealing with the hybrid system nature. It was also able to respect all the constraints of the system operation. In order to practically apply MPC technique to the SSID hybrid system, a hardware implementation of the controller was achieved.

In this thesis we aim to make the hybrid model predictive controller hardware implementation general for any hybrid system and any MPC objective function formulation. In other words, we aim to make this design an off-the-shelf design such that control engineers do not have to go in the details of the hardware implementation issues and allowing them to concentrate only in improving the performance of the control problem at hand.

There are some main difficulties in achieving this objective. First, hybrid control problems have large variation in size. The proposed design should have the ability to handle different problem sizes with the same hardware implementation. The second - and more important problem - is the large complexity of the system. This complexity results from the need of solving a MBQP optimization problem each sampling instants. If the proposed system could not handle this optimization problem in a fast manner, the application domain of the controller will be limited only to slow systems with large sampling times – in order of seconds. In this work we aim to improve the controller performance to solve the required optimization problem quickly. Hence, we can target relatively fast system with sampling intervals in range of few milliseconds.

To handle the problem size variation we should handle two issues; 1) storage of large problems data and 2) processing of all problems using the same hardware. The first issue is handled by trying to make an efficient use of the limited memory bits available on FPGA. This is done by reducing the number of memories required by sharing the available memories between different design components. An example is the sharing of memory between interior point hardware and MBQP hardware. This solution allows the same FPGA platform to handle larger problems. However, the maximum size of problem that can be handled depends on the number of memory bits available on target FPGA.

The issue of processing different problem sizes by the same hardware is solved by the proper design of the system control unit. The design control unit accepts problem size as an input from user. It uses this information to process system matrices correctly. When the problem size changes during operation – like the case of different QP problem sizes generated by MBQP solver – the control unit adapts hardware components to work with the new problem size. The control unit design has another feature that allows dealing with different problem sizes. This feature is its ability to partition a calculation to smaller parts. This is required when the number of arithmetic processing units do not allow for the completion of a whole calculation at the same time. The control unit divides the calculation into smaller parts. Each part is processed independently and its result is stored to memory. Then, other parts are calculated in the same manner.

The problem of large solution time is solved by introducing the concepts of parallel design and pipelining. Parallel design concept is applied by building multiple floating point calculation units instead of one unit. These parallel units are used simultaneously to perform different calculations. This parallel execution reduces the algorithm execution time. Another form of parallel processing is applied by the use of separate floating point adders, multipliers and dividers instead of using a composite floating point unit that can perform all the three operations. This implementation allows the use of different types of FP components to perform different operations. For example, multipliers can be used to perform a calculation while adders or dividers perform another calculation. This in turn helps to reduce execution time.

Pipelining was applied by the use of pipelined floating point units and by executing complex calculations in a pipelined fashion. Pipelined FP components allow the design to perform a new calculation each clock cycle. Its use allows also reducing the number of arithmetic units used as shown in example 2.3.
An example of how pipelining is used to speed up complex calculations was given in example 2.4. In this example, the multiplication of a matrix and a vector was executed in a pipelined fashion. Similarly, many calculations in the QP algorithm were executed in the same manner like matrix-matrix multiplication and Gauss-Jordan algorithm execution. This method of execution allowed the reduction of the QP algorithm execution time.

To allow the implementation of the hybrid MPC using different FPGA platforms, different architectures were studied. One of these architectures is suitable for FPGAs with small number of logic elements as it minimizes the used hardware. This architecture uses single FP adder, single FP multiplier, single FP divider and single FP comparator. If the used FPGA has a large number of logic elements, a better architecture can be used. In this architecture more adders and multipliers are implemented to allow parallel execution which reduces solver latency. The more the number of parallel units, the shorter the execution time achieved.
In our implementation we used a low cost ALTERA FPGA family (Cyclone III). The used FPGA kit was shipped with an EP3C25F324 FPGA which has more than

24000 logic elements. This number of LEs allows the use of parallel arithmetic units to reduce algorithm execution time.

The introduced design concepts were able to significantly improve the performance of the MBQP solver and consequently the hybrid model predictive controller. The current hybrid MPC controller implementation can target medium scale hybrid systems with sampling time in the range of few milliseconds.

Future work

Hybrid systems modeling and control represents a currently active research topic. The proposed modeling and control techniques need more improvement. For example, the complexity of the MLD model for a certain hybrid system may vary according to the HYSDEL model used. Inefficient HYSDEL description of the hybrid systems dynamics will result in complex MLD model that would be not useful for system simulation and control design. This leads to a need for more effort to be able to describe hybrid dynamics using the optimal HYSDEL description. MLD modeling will be much easier if some sort of optimization is performed during HYSDEL model compilation. These optimizations should work to minimize the complexity of the generated MLD model of a given HYSDEL model.

For the control design procedure, the design task can be much easier by automating the choice of the weighting matrices of the hybrid MPC objective function. Currently, the weighting matrices are selected by the control engineer by trial and error. Improper choice of weighting matrices can make the optimization problem too complex to solve. As MPC requires the solution of the optimization problem online, using complex optimization problem will require long solution time which means that MPC will be impractical to fast hybrid systems.

Regarding the hardware implementation of the hybrid MPC technique, there are a number of improvements that can be applied to improve the hybrid controller performance. Some of these improvements target the reduction of the memory used by the controller. This reduction of memory will allow increasing the maximum problem size that can be handled by a certain FPGA. It can also allow the implementation of more parallel QP solvers to be used by MBQP solver which will reduce MBQP solver latency.

One of the suggestions to reduce required memory is to use reduced precision floating point representation. In the current design we use standard single precision floating point representation. Each floating point number needs 32 bit of storage. If reduced precision can be used, i.e., each floating point is represented by 16 bits only, the amount of required storage will be reduced. Reducing precision will also reduce the operand width of the arithmetic units which mean we will reduce the number of logic elements used. It may also lead to higher operating frequency.

Another suggestion to reduce the required memory is to store the matrices in the design in a sparse form. Sparse matrix storage stores only the non-zero elements of the matrix along with its row and column indices. As the matrices used in optimization –especially the constraints coefficient matrix A-usually have few non-zero elements, sparse storage will reduce the required memory significantly. The implementation of sparse matrix storage will require the implementation of a special hardware component. This component is required to link the memory and design control unit that use full matrix representation. Other solution is to change the control unit such that is uses also the sparse matrix format.

Another class of improvements can be applied to target the reduction of the solver latency. One of these improvements is to use fixed-point arithmetic instead of floating point arithmetic. Fixed-point arithmetic units have simpler designs than floating point units. Hence, the use of fixed-point units will reduce the number of required logic elements will allow the introduction of more parallel units. Moreover, the operating frequency of fixed points units is usually higher than that of floating point units. Increasing the operating frequency will reduce the execution time.

Another suggestion to improve performance is to search for faster algorithms for the design complex operations like matrix-matrix multiplication and linear system solution. In the current design matrix-matrix multiplication was implemented using naïve method with some modifications to make it suitable for pipelining and parallelism. The linear system solution was implemented using basic Gauss-Jordon method. The method was adjusted to use parallel hardware and perform in a pipelined fashion. Exploring faster algorithms for the later complex calculation will be useful as these calculations represent the most time consuming operations in the design.

Another modification that may allow the application of the hybrid controller to very fast hybrid systems is to implement it as an ASIC. ASICs have much higher operating frequency than FPGAs which will allow the achievement of very small execution times.

Appendix A.1

Direct HYSDEL description of the SSID system

```
SYSTEM SSID {

INTERFACE {
    STATE { REAL x1 [0,0.0001];
        REAL x2 [-10,10];
        REAL x3 [0,0.0001];
        REAL x4 [-10,10];
        }
    INPUT { REAL u [0,60];
        }
    OUTPUT{ REAL y;
        }
    PARAMETER {
        REAL M = 1;
        REAL m_s = 0.05;
            REAL k  = 1.76e7;
            REAL FC = 14;

        REAL a11 = -0.050279;
        REAL a12 = 0.00024394;
        REAL a13 = 0;
        REAL a14 =0;
        REAL a21 = -4089;
        REAL a22 = -0.050279;
        REAL a23 = 0;
        REAL a24 = 0;
        REAL a31 = -1.0503;
        REAL a32 = -0.0047561;
        REAL a33 = 1;
        REAL a34 = 0.005;
        REAL a41 = -4089;
        REAL a42 = -1.0503;
        REAL a43 = 0;
        REAL a44 = 1;
        REAL b11 = 5.9675e-8;
        REAL b12 = 0.00023233;
        REAL b13 = 5.9675e-8;
        REAL b14 = 0.00023233;
        REAL G11 = 0.90498;
        REAL G12 = -2.2677e-5;
        REAL G13 = 0;
        REAL G14 = 0;
        REAL G21 = 7982.2;
        REAL G22 = 0.90498;
        REAL G23 = 0;
        REAL G24 = 0;
        REAL G31 = 0;
        REAL G32 = 0;
        REAL G33 = 1;
        REAL G34 = 0.005;
        REAL G41 = 0;
        REAL G42 = 0;
        REAL G43 = 0;
        REAL G44 = 1;
        REAL b21 = 5.3989e-9;
        REAL b22 = -0.00045354;
        REAL b23 = 0;
        REAL b24 = 0;
             }
    }

IMPLEMENTATION {
        AUX { REAL z1,z2,z3,z4,z5,z6,z7,z8,z9,z10,z11,z12;
              BOOL sm,sp; }
        AD  { sm = (M*(u-k*x1)/(M+m_s))<=-FC;
              sp = (M*(u-k*x1)/(M+m_s))>=FC;
              stick = ~sm & ~sp;
             }
        DA  { z1 = {IF stick THEN a11*x1+a12*x2+a13*x3+a14*x4+b11*u
                    ELSE 0 };
            z2 = {IF sm THEN G11*x1+G12*x2+G13*x3+G14*x4+b21*u+7.5584e-8
                    ELSE 0 };
            z3 = {IF sp THEN G11*x1+G12*x2+G13*x3+G14*x4+b21*u-7.5584e-8
                    ELSE 0 };
              z4 = {IF stick THEN a21*x1+a22*x2+a23*x3+a24*x4+b12*u
                    ELSE 0 };
              z5 = {IF sm THEN G21*x1+G22*x2+G23*x3+G24*x4+b22*u-0.0063
                    ELSE 0 };
              z6 = {IF sp THEN G21*x1+G22*x2+G23*x3+G24*x4+b22*u+0.0063
                    ELSE 0 };
              z7 = {IF stick THEN a31*x1+a32*x2+a33*x3+a34*x4+b13*u
                    ELSE 0 };
              z8 = {IF sm THEN G31*x1+G32*x2+G33*x3+G34*x4+b23*u-1.75e-4
                    ELSE 0 };
              z9 = {IF sp THEN G31*x1+G32*x2+G33*x3+G34*x4+b23*u+1.75e-4
                    ELSE 0 };
              z10 = {IF stick THEN a41*x1+a42*x2+a43*x3+a44*x4+b14*u
                    ELSE 0 };
              z11 = {IF sm & ~sp THEN G41*x1+G42*x2+G43*x3+G44*x4+b24*u-
                        0.07
                    ELSE 0 };
              z12 = {IF sp THEN  G41*x1+G42*x2+G43*x3+G44*x4+b24*u+0.07
                    ELSE 0 };
  }
        CONTINUOUS {
             x1 = z1+z2+z3;
             x2 = z4+z5+z6;
             x3 = z7+z8+z9;
             x4 = z10+z11+z12;}
        MUST {
                x1 >= 0;
                x3 >= 0;
                u  >= 0;
                u  <= 60;
                ~(sp & sm);
                ~(sp & stick)
                ~(stick & sm)
            }

        OUTPUT { y = x3;  }
      }
}
```

Appendix A.2

HYSDEL description of the SSID system with reduced number of binary variables.

```
SYSTEM SSID {

INTERFACE {
    STATE { REAL x1 [0,0.0001];
        REAL x2 [-10,10];
        REAL x3 [0,0.0001];
        REAL x4 [-10,10];
        }
    INPUT { REAL u [0,60];
        }
    OUTPUT{ REAL y;
        }
    PARAMETER {
        REAL M = 1;
        REAL m_s = 0.05;
            REAL k  = 1.76e7;
            REAL FC = 14;

        REAL a11 = -0.050279;
        REAL a12 = 0.00024394;
        REAL a13 = 0;
        REAL a14 =0;
        REAL a21 = -4089;
        REAL a22 = -0.050279;
        REAL a23 = 0;
        REAL a24 = 0;
        REAL a31 = -1.0503;
        REAL a32 = -0.0047561;
        REAL a33 = 1;
        REAL a34 = 0.005;
        REAL a41 = -4089;
        REAL a42 = -1.0503;
        REAL a43 = 0;
        REAL a44 = 1;
        REAL b11 = 5.9675e-8;
        REAL b12 = 0.00023233;
        REAL b13 = 5.9675e-8;
        REAL b14 = 0.00023233;
        REAL G11 = 0.90498;
        REAL G12 = -2.2677e-5;
        REAL G13 = 0;
        REAL G14 = 0;
        REAL G21 = 7982.2;
        REAL G22 = 0.90498;
        REAL G23 = 0;
        REAL G24 = 0;
        REAL G31 = 0;
        REAL G32 = 0;
        REAL G33 = 1;
        REAL G34 = 0.005;
        REAL G41 = 0;
        REAL G42 = 0;
        REAL G43 = 0;
        REAL G44 = 1;
        REAL b21 = 5.3989e-9;
        REAL b22 = -0.00045354;
        REAL b23 = 0;
        REAL b24 = 0;
            }
    }

IMPLEMENTATION {
        AUX { REAL z1,z2,z3,z4,z5,z6,z7,z8;
              BOOL s1,s2; }
        AD  { s1 = (M*(u-k*x1)/(M+m_s))<=-FC;
              s2 = (M*(u-k*x1)/(M+m_s))<= FC; }
        DA  { z1 = {IF s1 THEN
                  G11*x1+G12*x2+G13*x3+G14*x4+b21*u+7.5584e-8-
                  (a11*x1+a12*x2+a13*x3+a14*x4+b11*u)
                    ELSE 0 };
              z2 = {IF s2 THEN a11*x1+a12*x2+a13*x3+a14*x4+b11*u
                    ELSE G11*x1+G12*x2+G13*x3+G14*x4+b21*u-7.5584e-8 };
              z3 = {IF s1 THEN G21*x1+G22*x2+G23*x3+G24*x4+b22*u-0.0063-
                        (a21*x1+a22*x2+a23*x3+a24*x4+b12*u)
                    ELSE 0 };
              z4 = {IF s2 THEN  a21*x1+a22*x2+a23*x3+a24*x4+b12*u
                    ELSE G21*x1+G22*x2+G23*x3+G24*x4+b22*u+0.0063 };
           z5 = {IF s1 THEN G31*x1+G32*x2+G33*x3+G34*x4+b23*u-1.75e-4-
                  (a31*x1+a32*x2+a33*x3+a34*x4+b13*u)
                    ELSE 0 };
              z6 = {IF s2 THEN a31*x1+a32*x2+a33*x3+a34*x4+b13*u
                    ELSE G31*x1+G32*x2+G33*x3+G34*x4+b23*u+1.75e-4};
              z7 = {IF s1 THEN G41*x1+G42*x2+G43*x3+G44*x4+b24*u-0.07-
                  (a41*x1+a42*x2+a43*x3+a44*x4+b14*u)
                    ELSE  0};
              z8 = {IF s2 THEN a41*x1+a42*x2+a43*x3+a44*x4+b14*u
                    ELSE  G41*x1+G42*x2+G43*x3+G44*x4+b24*u+0.07};
            }
       CONTINUOUS {
                    x1 = z1+z2;
                    x2 = z3+z4;
                    x3 = z5+z6;
                    x4 = z7+z8;}
        MUST {
                x1 >= 0;
                x3 >= 0;
                u  >= 0;
                u  <= 60;
                ~(s1 & ~s2);
            }
         OUTPUT { y = x3;  }
      }
}
```

Appendix A.3

HYSDEL modeling of the SSID with the proposed simplifications

```
SYSTEM SSID {
INTERFACE {
    STATE { REAL x1 [0,0.0001];
        REAL x2 [-10,10];
        REAL xMold [0,0.0001];
        }
    INPUT { REAL u [0,60];
        }
    OUTPUT{ REAL y;
        }
    PARAMETER {
        REAL M = 1;
        REAL m_s = 0.05;
        REAL k  = 1.76e7;
        REAL FC = 14;
        REAL T = 0.001;
        REAL a11 = 1;
        REAL a12 = T;
        REAL a21 = 0;
        REAL a22 = 1;
        REAL b11 = 0.5*T*T/(M+m_s);
        REAL b12 = T/(M+m_s);
        REAL b21 = -0.5*T*T*k/(M+m_s);
        REAL b22 = -k*T/(M+m_s);
        REAL b31 = -0.5*T*T*FC/(M);
        REAL b32 = -T*FC/(M);
             }
    }
IMPLEMENTATION {
        AUX { REAL z1,z2,xMold_aux;
              BOOL s1; }
        AD  { s1 = (M*(u-k*(x1-xMold))/(M+m_s))>=-FC;
             }
        DA  { z1 = {IF s1 THEN b11*u+b21*(x1-xMold)
                    ELSE b31 };
              z2 = {IF s1 THEN b12*u+b22*(x1-xMold)
                    ELSE  b32};
              xMold_aux = {IF s1 THEN xMold
                    ELSE  a11*x1+a12*x2+b31};
  }
        CONTINUOUS { x1 = a11*x1+a12*x2+z1;
                    x2 =  a21*x1+a22*x2+z2;
                    xMold = xMold_aux;
}
        MUST {
                u  >= 0; u  <= 60;
                M*(u-k*x1+k*xMold)/(M+m_s)<= FC;
            }
         OUTPUT { y = x1;  }
    }
}
```

References

[1] R. L. Grossmann, A. Nerode, A. P. Ravn and H. Rischel, " Hybrid Systems" Springer Verlag , New York, 1993.

[2] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine "The algorithmic analysis of hybrid systems". Theoretical Computer Science, volume 138(1), pages 3-34,1995.

[3] P. J. Antsaklis, Xenofon D. Koutsoukos, "On hybrid control of complex systems: A survey", Technical Report of the ISIS Group at the University of Notre Dame ISIS-97-017, December, 1997.

[4] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints." Automatica, vol. 35, no. 3, pp. 407–427, March 1999.

[5] L. A. Wolsey, "Integer Programming", John Wiley & Sons, Inc., 1998.

[6] L. G. Bleris and M. V. Kothare, "Real-time implementation of model predictive control," in Proceedings of American Control. Conference, 2005, pp. 1752–1757.

[7] K. Ling, S. Yue, and J. Maciejowski, "An FPGA Implementation of Model Predictive Control," In proceedings of the American Control Conference, Minneapolis, MN, June 2006.

[8] P. D. Vouzis, M. V. Kothare, L. G. Bleris and M. G. Arnold, "A System-on-a-Chip Implementation for Embedded Real-Time Model Predictive Control", in IEEE Transactions on Control Systems Technology, pp. 1006 – 1017, 2009.

[9] A. I. Propoi, "Use of linear programming methods for synthesizing sampled-data automatic systems". Automatic Remote Control, 24(7), 837–844, 1963.

[10] E. B. Lee, L. Markus, "Foundations of optimal control theory", New York, Wiley, 1967.

[11] J. A. Richalet, A. Rault, J. D. Testud, and J. Papon, "Model Predictive Heuristic Control: Application to Industrial Processes", Automatica, 13, 413,1978.

[12] S. Joe Qin, Thomas A. Badgwell, “A survey of industrial model predictive control technology”, Control Engineering Practice 11, pp. 733–764, 2003.

[13] Nikolaou, M., “Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs”, Advances in Chemical Engineering Series, Academic Press, 2001.

[14] J. Nocedal and S. J. Wright, “Numerical Optimization”, Second Edition, Springer Verlag 2006.

[15] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, “The explicit linear quadratic regulator for constrained systems,” Automatica, vol. 38, no. 1, pp. 3–20, 2002.

[16] A. Bemporad, “Model-based predictive control design: New trends and tools”, In Proceedings of 45th IEEE Conference on Decision and Control, San Diego, CA, pp. 6678–6683 (2006).

[17] F. Borrelli, A. Bemporad, M. Fodor, D. Hrovat, “An MPC/hybrid system approach to traction control”, IEEE Transactions on Control Systems Technology, pp 541 – 552, May 2006.

[18] R. S. Parker, F. J. Doyle III, and N. A. Peppas, “A Model-Based Algorithm for Blood Glucose Control in Type I Diabetic Patients,” IEEE Transactions on Biomedical Engineering, vol. 46, pp. 148–156, Feb. 1999.

[19] L. G. Bleris, P. Vouzis, M. G. Arnold, and M. V. Kothare, “Pathways for Optimization-Based Drug Delivery Systems and Devices,” In proceedings of the International Symposium on Advanced Control of Chemical Processes (ADCHEM) 2006, Gramado, Brazil. April 2006.

[20] G. Labinaz, M.M. Bayoumi, K. Rudie, “Modeling and control of Hybrid Systems: A Survey”, IFAC 13th Triennial World Congress, 1996.

[21] W. P. M. H. Heemels, B. De Schutter and A. Bemporad, “Equivalence of hybrid dynamical models”, Automatica, Volume 37, Issue 7, Pages 1085-1091, July 2001.

[22] R. Goebel, R. Sanfelice, A. Teel, “Hybrid dynamical systems”, IEEE Control Systems Magazine, April, 2009.

[23] Thomas A. Henzinger, “The Theory of Hybrid Automata”, Proceedings of the 11th Annual Symposium on Logic in Computer Science, pp. 278-292, 1996.

[24] M. Morari, J. Buisson, B. de Schutter, and G. Papafotiou, “Final report on modeling tools, benchmarks and control methods,” HYCON Deliverable, Tech. Rep. IST contract number 511368, 2006.

[25] M. Lazar, “Model predictive control of hybrid systems: Stability and robustness,” Ph.D. Thesis , Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2006.

[26] M. Lazar and W.P.M.H. Heemels, “Predictive control of hybrid systems: Input-to-state stability results for sub-optimal solutions”, Automatica, Volume 45, Issue 1, p. 180-185, January 2009.

[27] H.P. Williams, "Model Building in Mathematical Programming", John Wiley & Sons, Third Edition,1993.

[28] B. Sedghi, B. Srinivasan, R. Longchamp, “Control of Hybrid Systems via Dehybridization", Proceedings of the American Control Conference, 2002.

[29] C. A. Floudas, “Nonlinear and Mixed Integer Optimization”, Oxford University Press, 1995.

[30] R. Fletcher and S. Lyffer, “Numerical experience with lower bounds for MIQP branch and bound”, SIAM Journal on Optimization, 8(2):604 – 616, May 1998.

[31] Daniel Axehill, “Integer Quadratic Programming for Control and Communication”, PhD thesis , Linköping University, 2008.

[32] A. Bemporad, F. Borrelli, and M. Morari, “Piecewise linear optimal controllers for hybrid systems”, in American Control Conference, Chicago, IL, June 2000, pp. 1190–1194.

[33] A. Bemporad, “Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form,” IEEE Transactions on Automatic Control, vol. 49, no. 5, pp. 832–838, 2004.

[34] F. Borrelli, M. Baotic, A. Bemporad and M. Morari, “Dynamic programming for constrained optimal control of discrete-time linear hybrid systems”, Automatica, vol. 41, no. 10, pp. 1709–1721, Oct. 2005.

[35] A. Alessio and A. Bemporad, “Feasible mode enumeration and cost comparison for explicit quadratic model predictive control of hybrid systems,” in 2nd IFAC Conference on Analysis and Design of Hybrid Systems, pp. 302–308, Alghero, Italy, 2006.

[36] D. Mayne and S. Rakovic, “Optimal control of constrained piecewise affine discrete time systems using reverse transformation”, in Proceedings of the 41th IEEE Conference on Decision and Control, pp. 1546–1551, Las Vegas, Nevada, USA, Dec. 2002.

[37] F.D. Torrisi and A. Bemporad, “HYSDEL - A tool for generating computational hybrid models,” IEEE Transactions on Control Systems Technology, vol. 12, no. 2, pp. 235–249, Mar. 2004.

[38] A. Bemporad, “Hybrid Toolbox user's guide”, www.dii.unisi.it/hybrid/toolbox.

[39] Stephen Boyd, Lieven Vandenberghe, “Convex Optimization”, Cambridge University Press, 2004.

[40] N. Megiddo, “Pathways to the optimal set in linear programming”, in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed. , ch. 8, pp. 131–158, Springer-Verlag, New York, N.Y., 1989.

[41] Masakazu Kojima, Nimrod Megiddo, Toshihito Noma, Akiko Yoshise, “A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems: A Summary”, of Lecture Notes in Computer Science, 1991.

[42] S. J. Wright, “Primal-Dual Interior-Point Methods”, SIAM Publications, Philadelphia, Pa, 1997.

[43] A. V. Fiacco and G. P. McCormick, “Nonlinear Programming: Sequential Unconstrained Minimization Techniques”, John Wiley & Sons, New York, NY, 1968. Reprinted by SIAM Publications, 1990.

[44] I.J. Lustig, “Feasibility issues in a primal-dual interior-point method for linear programming”, Mathematical Programming, 49(2):145–162, 1990.

[45] K. Tanabe, “Centered Newton method for linear programming: exterior point method” (in japanese). In Proc. Inst. Stat. Mathematics, volume 37, pages 146–148, 1989.

[46] M. Kojima, N. Megiddo, S. Mizuno, "A primal-dual infeasible-interior-point algorithm for linear programming", Mathematical Programming, 61:263–400, 1993.

[47] S.J.Wright, "An infeasible-interior-point algorithm for linear complementarity problems", Mathematical Programming, 67:29–52, 1994.

[48] F.A. Potra, "A quadratically convergent predictor-corrector method for solving linear programs from infeasible starting points", Mathematical Programming, 67:383–406, 1994.

[49] R.D.C.Monteiro, I. Adler, "Interior path following primal-dual algorithms. part ii: Convex quadratic programming", Mathematical Programming, 44:43–66, 1989.

[50] R.D.C. Monteiro, F. Zhou, "On superlinear convergence of infeasible-interior-point algorithms for linearly constrained convex programs", Computational Optimization and Applications, 8:245–262, 1997.

[51] R.D.C.Monteiro, I. Adler, "Interior path following primal-dual algorithms. part i: Linear programming", Mathematical Programming, 44:27–41, 1989.

[52] I. J. Lustig, R. E. Marsten, D. F. Shanno, "Computational experience with a primal-dual interior point method for linear programming", Journal of Linear Algebra and Its Applications, 152:191–222, 1991.

[53] O.V. Volkovich, V.A. Roschhin, and I. V. Sergienko, "Models and methods of solution of quadratic integer programming problems", Cybernetics, 23:289-305. 1987.

[54] A. H. Land and A. G. Doig (July 1960). "An automatic method of solving discrete programming problems". Econometrica 28 (3): pp. 497-520.

[55] Nios II Processor Reference Handbook , ver 10.0, Jul 2010, www.Altera.com.

[56] Microblaze Processor reference guide, www.Xilinx.com.

[57] Y. Shimai, J. Tani, H. Noguchi, H. Kawaguchi, and M. Yoshimoto, "FPGA Implementation of Mixed Integer Quadratic Programming Solver for Mobile Robot Control", International Conference on Field-Programmable Technology, Sydney, December 2009.

[58] Institute of Electrical and Electronics Engineers, “1067-2008 IEEE standard VHDL language reference manual”, Jan, 2009.

[59] David Bishop, “Floating point package user’s guide”, http://www.vhdl.org/fphdl/vhdl.html.

[60] http://www.mathworks.com/products/modelsim/.

[61] David C. Lay, “Linear Algebra and Its Applications”, 3rd edition, Addison Wesley, 2005.

[62] Carl D. Meyer, “Matrix Analysis and Applied Linear Algebra”, Society for Industrial and Applied Mathematics (SIAM), 2001.

[63] R. L. Burden, J. D. Faires, “Numerical Analysis”, 8th edition, Thompson Brooks/Cole, 2005.

[64] http://tomopt.com/docs/qp_prob.zip

[65] I. Maros, C. Meszaros, “A Repository of Convex Quadratic Programming Problems”, Optimization Methods and Software, p. 671-681, 11-12, 1999.

[66] Kenneth Holmström, Anders O. Göran and Marcus M. Edvall”, User's Guide for TOMLAB /CPLEX v12.1”, August 14, 2009.

[67] J. Breguet, M. Bregue, W. Driesen, F. Kaegi, T. Cimprich, “Applications of Piezo-Actuated Micro-Robots in Micro-Biology and Material Science”, International Conference on Mechatronics and Automation, 2007.

[68] J. M. Breguet, “Actioneurs `Stick-Slip' pour Micro-Manipulateurs”, PhD thesis, Ecole Polytechnique Federale de Lausanne, 1998.

[69] J. M. Breguet and Ph. Renaud, “A 4-degree-of-freedom microrobot with nanometer resolution”, Robotica, 14:199-203 , 1996.

[70] T. Conus, “Projet de 8eme semestre: Entrainement `stick-slip' avec actionneur piezo”, Technical Report 50324, Laboratoire d'Automatique, Ecole Polytechnique Federale de Lausanne, 1997.

[71] Babak Sedghi “Control Design Via Dehybridization”, PhD thesis, Laboratoire d'automatique, Lausanne, EPFL, 2003.

ملخص البحث

جذبت الانظمة الهجينة الكثير من الاهتمام في السنوات الاخيرة. وتعرف الانظمة الهجينة بانها النظم التي تشتمل على معادلات متصلة واخري متقطعة. تظهر الانظمة الهجينة في العديد من التطبيقات مثل التحكم بالمتحركات وتطبيقات السيارات. ويمكن استخدام الانظمة الهجينة في وصف انواع متعددة من النظم مثل النظم التي تتغير طريقة عملها مع الوقت والنظم المتصلة التي يتم التحكم بها باستخدام قواعد منطقية, كما يمكن استخدامها في وصف بعض الانظمة اللاخطية.

احد ابرز التطورات التي حدثت في مجال وصف تصرف الانظمة الهجينة هو الوصول الى طريقة نمذجة قياسية. باستخدام هذه النمذجة تم تقديم العديد من الادوات التي تقوم باستنتاج نموذج النظام بشكل منهجي. وقد سمحت ايضا باستخدام طريقة منهجية للتحكم في هذه النوعية من النظم باستخدام التحكم الامثل والتحكم باستخدام التوقع المبني على النموذج. وتشتمل هذه الادوات على مكونات يمكنها محاكاة النظام الهجين اثناء تطبيق اسلوب التحكم المقترح.

واسلوب التحكم المبني على التوقع باستخدام النموذج هو من اكثر الاساليب نجاحا للتحكم في الانظمة الهجينة. وقد سبق تطبيقه بنجاح كبير في الانظمة الخطية ويوجد العديد من التطبيقات الصناعية التي تستخدمه. في الفترة الاخيرة كان هناك بعض المجهودات البحثية لاستخدام هذا الاسلوب للتحكم في الانظمة المدمجة والانظمة الموجودة علي شريحة الكترونية. في هذه الرسالة نقوم بدراسة استخدام التحكم المبني على التوقع باستخدام النموذج للتحكم في النظم الهجينة المدمجة. العائق الرئيسي لهذا الاستخدام هو التعقيد الكبير للنظام. حيث أن اسلوب التحكم المقترح يتطلب حل مسألة برمجة تربيعية ذات متغيرات حقيقية ومتغيرات ثنائية القيمة في كل لحظة من لحظات التحكم. هذا التعقيد يقصر تطبيق نظام التحكم المقترح على الانظمة الهجينة البطيئة فقط.

للتغلب على هذا العائق تم تصميم عتاد يقوم بحل مسألة البرمجة التربيعية المذكورة وحساب اشارة التحكم المثلي في وقت قصير. وقد تم تطبيق بعض مفاهيم التصميم الرقمي مثل التنفيذ المتوازي والتنفيذ التجزيئي للعمليات حتى يمكن الوصول بالتصميم للسرعة المطلوبة.

وقد تم تنفيذ نموذج لهذا العتاد باستخدام مصفوفات البوابات المبرمجة حقليا . يسمح تنفيذ هذا العتاد السريع بتطبيق اسلوب التحكم المبني على التوقع باستخدام النموذج على الأنظمة الهجينة والتي تكون فيها فترات التحكم في حدود أجزاء قليلة من الالف من الثانية.

وقد تم اختبار المتحكم الذي تم تصميمه على احد الانظمة الهجينة المدمجة. هو نظام المحرك المبني على التمييز بين الالتصاق والانزلاق. وقد اثبت هذا المثال قدرة المتحكم المصمم على التحكم في نظم هجينة سريعة.