

# ASIC and FPGA Implementation Strategies for Model Predictive Control

Geoff Knagge, Adrian Wills, Adam Mills, Brett Ninness

**Abstract**—This paper considers the system architecture and design issues for implementation of on-line Model Predictive Control (MPC) in Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). In particular, the computationally intensive tasks of fast matrix QR factorisation, and subsequent sequential quadratic programming, are addressed for control law computation. An important aspect of this work is the study of appropriate data word-lengths for various essential stages of the overall solution strategy.

## I. INTRODUCTION

Model predictive control (MPC) techniques have recently enjoyed an upsurge of interest within the automatic control community, due to their ability to handle non-linear systems and constraints on allowable control inputs and system states [15], [12].

The essential idea underpinning the method is that a constrained optimisation problem is solved in order to generate a required control action. Unfortunately, the requirement for on-line solution of the optimisation problem is a main impediment to widespread implementation of the approach. As such, despite the important attractive features of MPC, its use to date has been largely restricted to chemical process control problems that operate on sufficiently slow time scales [15].

This paper is directed at this difficulty by detailing methods for using specialised hardware that is capable of fast solution of the associated optimisation problems. The hardware design is captured here via an generalised architecture targeted at implementation in a field programmable gate array (FPGA), or as part of an Application Specific Integrated Circuit (ASIC).

Hardware designs supporting fast MPC solution are currently generating increasing interest, and the reader is referred to several other important contributions on the topic [1], [11], [10]. The work here is discriminated from these previous contributions by paying particular attention to encompassing non-linear systems, and considering the finer detail of appropriate numerical precision.

All authors are with the School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW, 2308, Australia. Corresponding author [adrian.wills@newcastle.edu.au](mailto:adrian.wills@newcastle.edu.au)

An essential aspect of the hardware design is the consideration of trade-offs between data word size and computation speed, versus numerical precision and effectiveness of the computed control action. Previous work by the authors has demonstrated an implementation on a software programmed digital signal processor (DSP) device [17]. However, DSPs are typically restricted to standard, non-customisable, floating point numerical systems.

The paper is organised into two main parts. Part I includes Sections II–V and is concerned with the broad MPC algorithm. In this part, we identify numerical operations that are key to the efficient implementation of MPC in hardware. Part II of the paper is encompassed by Section VI, where these key numerical operations are discussed in terms of their hardware implementation.

## II. NONLINEAR MODEL PREDICTIVE CONTROL

Consider a system whose dynamic behaviour can be described via a discrete-time nonlinear state-space model

$$x_{k+1} = f_k(x_k, u_k) \quad (1)$$

where the state  $x_k \in \mathbb{R}^n$ , the input  $u_k \in \mathbb{R}^m$  and the function  $f_k(\cdot, \cdot) \in \mathbb{R}^n$  maps the current state and input to the next state  $x_{k+1}$ . It is assumed that  $f_k$  is twice continuously differentiable in both the state and input arguments, and that  $f_k(0, 0) = 0$  for all  $k$ .

Given an initial state value  $x_1$ , control of the state is desired over subsequent time intervals, to a target region in the state space (for example the origin). As a first step, observe that the model (1) may be used to predict future state values over a any prediction horizon  $N$ , based on an initial state  $x_1$  and future input moves  $\{u_1, \dots, u_N\}$ . More precisely, if the current state  $x_1$  is known, then the state trajectory is given by

$$x_2 = f_1(x_1, u_1), \dots, x_{N+1} = f_N(x_N, u_N)$$

Therefore, the state at any time in the future is a function of the initial state  $x_1$  and all the inputs  $\{u_1, \dots\}$ . Provided that the input has sufficient control authority, it is possible to choose an input sequence

$$\mathbf{u} = \{u_1, \dots, u_N\} \quad (2)$$

that moves the initial state  $x_1$  towards a desired region, e.g. the origin. This aim is typically achieved by minimising a cost function

$$V(\mathbf{u}) = \sum_{k=1}^N x_{k+1}^T Q x_{k+1} + u_k^T R u_k \quad (3)$$

where  $Q \in \mathbb{R}^{n \times n}$  is assumed to be a positive semi-definite matrix used to penalise state-movements about the origin, and  $R$  is assumed to be a positive definite matrix that penalises input movements from the origin.

In addition to minimising this cost, a key benefit of the MPC approach is that physical limits on the system inputs and states can also be directly included into the optimisation problem. More precisely, if the constraints are describe via  $c(\mathbf{u}) \leq 0$  where  $c(\cdot) : \mathbb{R}^{Nm} \rightarrow \mathbb{R}^{n_c}$ , then the MPC optimisation problem becomes

$$\mathbf{u}^* = \arg \min_{\mathbf{u}} V_N(\mathbf{u}) \quad \text{s.t.} \quad c(\mathbf{u}) \leq 0. \quad (4)$$

### III. SOLVING THE MPC PROBLEM

In order to implement the MPC algorithm, it is necessary to solve non-convex optimisation problems of the type in (4) online. This, in general, is a difficult task. Amongst the many approaches used to solve these types of problems, Sequential Quadratic Programming (SQP) methods are very competitive [2]. This approach is discussed below, with a view to highlighting some of the computationally demanding numerical operations, which are the subject of the remainder of this paper.

The SQP approach to solving (4) is as follows (see e.g. Ch. 18 in [13]). In a first step the cost function and constraints are replaced with local approximations, which lead to a Quadratic Programming (QP) subproblem

$$p^* = \arg \min_p p^T \nabla_{\mathbf{u}} L(\mathbf{u}, \lambda) + \frac{1}{2} p^T \nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda) p \quad \text{s.t.} \quad c(\mathbf{u}) + p^T \nabla_{\mathbf{u}} c(\mathbf{u}) \leq 0 \quad (5)$$

where  $\nabla_{\mathbf{u}}$  and  $\nabla_{\mathbf{u}}^2$  refer to the operation of taking first and second order derivatives, respectively, and the Lagrangian function  $L$  is defined as

$$L(\mathbf{u}, \lambda) = V(\mathbf{u}) + \lambda^T c(\mathbf{u}). \quad (6)$$

Computing the Hessian  $\nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda)$  of the Lagrangian function is usually computationally intensive. One common approach to alleviate this problem is to update an approximate Hessian at each iteration of the method (such as the BFGS method - see Ch. 18 in [13]). However, if the sum-of-squares formulation in (3) is exploited, then (3) can be expressed as

$$V(\mathbf{u}) = e^T(\mathbf{u})e(\mathbf{u}) \quad (7)$$

where  $e$  is a column vector and is defined as

$$e^T(\mathbf{u}) = [(Q^{1/2}x_2)^T, \dots, (Q^{1/2}x_{N+1})^T, (R^{1/2}u_1)^T, \dots, (R^{1/2}u_N)^T]. \quad (8)$$

Therefore, the Hessian of the Lagrangian is given by

$$\nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda) = J^T(\mathbf{u})J(\mathbf{u}) + S(\mathbf{u}) \quad (9)$$

where  $J(\mathbf{u}) = \nabla_{\mathbf{u}} e(\mathbf{u})$  is the Jacobian matrix of  $e$  and  $S(\mathbf{u})$  is defined as

$$S(\mathbf{u}) = \left( \sum_{k=1}^{N(n+m)} e_k(\mathbf{u}) \nabla_{\mathbf{u}}^2 e_k(\mathbf{u}) \right) + \sum_{i=1}^{n_c} \lambda_i \nabla_{\mathbf{u}}^2 c_i(\mathbf{u}) \quad (10)$$

where  $c_i(\cdot)$  refers to the  $i$ 'th element in  $c(\cdot)$ . From (10), it can be seen that part of the Hessian depends on terms that are scaled by  $e_k(\mathbf{u})$ . However, as the state approaches the origin  $e_k(\mathbf{u}) \rightarrow 0$  (via assumptions on (1)) and we therefore these terms in the Hessian calculation can be neglected (this is a standard approach for nonlinear least squares - see Ch. 10 in [4]). This results in the following Hessian approximation

$$\nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda) \approx J^T(\mathbf{u})J(\mathbf{u}) + \sum_{i=1}^{n_c} \lambda_i \nabla_{\mathbf{u}}^2 c_i(\mathbf{u}).$$

The Jacobian term is assumed to be available and it remains to approximate the second order derivatives of the constraints. This can be achieved via the BFGS approach (see Chs. 8–9 in [4]), but we have modified the update to ensure that the approximation is positive semi-definite (note that the Lagrange multipliers will be maintained as non-negative numbers). This latter requirement implies that the full Hessian approximation is positive definite since  $J^T(\mathbf{u})J(\mathbf{u})$  is positive definite by construction; recall that  $R$  is assumed positive definite.

Once the QP subproblem in (5) is solved for  $p^*$  with associated Lagrange multipliers  $\lambda^*$ , a second stage of the (line-search) SQP approach tries to find a suitable step-length  $\alpha$  such that  $\mathbf{u} + \alpha p^*$  reduces a merit function, which typically uses  $\lambda^*$  as part of a penalty term. For this purpose, the well known one-sided  $\ell_1$  merit function is used - see Ch. 18 in [13].

The above SQP approach has been implemented on a standard 2GHz Laptop for the purposes of controlling an inverted pendulum apparatus. The prediction horizon in that case was  $N = 60$  and sampling interval was 25msec. Both input and state constraints are present in the formulation and a video that demonstrates this can be found at <http://sigpromu.org/mpc>.

### A. Key Numerical Operations

From the above approach, the key numerical operations that need to be performed on-line are:

- 1) Evaluate the cost  $V(\mathbf{u})$  and constraints  $c(\mathbf{u})$ ;
- 2) Compute Jacobians  $\nabla_{\mathbf{u}}e(\mathbf{u})$  and  $\nabla_{\mathbf{u}}c(\mathbf{u})$ ;
- 3) Compute an approximation to  $\nabla_{\mathbf{u}}^2L(\mathbf{u}, \lambda)$ ;
- 4) Solve the QP subproblem in (5);
- 5) Evaluate a merit function;

It is the experience of the authors that items 2–4 require the vast majority of computational effort. In the next section we discuss the QP subproblem solution, which leads to a discussion on items 2–3.

### IV. QUADRATIC PROGRAMMING SUBPROBLEM

Direct online QP methods are of primary interest due to the flexibility they offer. The two most commonly used variants are active-set and interior-point methods, and robust implementations of both methods are available. However, for time-critical on-line optimisation it is often necessary to adapt such tools in order to exploit problem structure.

In light of this and with knowledge of the types of QP problems we encounter, we have implemented an active-set method based on the work of [6], [14]. While this method requires a positive definite Hessian matrix  $H$ , as discussed in Section III, it does not require a primal feasible initial point and this greatly simplifies the algorithm [17].

For the purposes of discussion, consider the following QP problem

$$p^* = \arg \min_p \frac{1}{2} p^T H p + 2g^T p, \quad \text{s.t. } A p \leq b$$

which corresponds to (5) with

$$\begin{aligned} H &= \nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda), & g &= \nabla_{\mathbf{u}} L(\mathbf{u}, \lambda) \\ A &= \nabla_{\mathbf{u}} c(\mathbf{u}), & b &= -c(\mathbf{u}) \end{aligned}$$

where  $p \in \mathbb{R}^{Nm}$  and  $A \in \mathbb{R}^{n_c \times Nm}$ . The algorithm starts with the unconstrained solution  $(-H^{-1}g)$ , each subsequent iteration adds a violated constraint, if any, to the active-set of constraints and then solves an equality constrained problem, where the equality constraints are those listed in the active-set. An already active constraint may also be dropped if it is no longer needed (i.e. the associated Lagrange multiplier is negative).

The method maintains two matrices  $Z$  and  $U$  such that  $ZZ^T = H^{-1}$  and  $Z^T A_a = U$ , where the columns of  $A_a$  hold the normals to the active constraints and  $U$  is an upper triangular matrix. The implementation uses Givens rotations (see Section VI-B) to update the matrices  $Z$  and  $U$  in a numerically robust fashion. If there are no more constraints in violation then the algorithm terminates and the solution is optimal.

The key points of interest are:

- The need for a fast and numerically robust means to compute an initial  $Z$  matrix that satisfies  $ZZ^T = H^{-1}$  (discussed in Section V);
- The need for a numerically fast and stable Givens implementation for updating  $Z$  and  $U$  when adding or dropping a constraint from the active-set (discussed in Section VI-B).

### V. COMPUTING $Z$

Recall that the Hessian matrix under consideration has the structure

$$\nabla_{\mathbf{u}}^2 L(\mathbf{u}, \lambda) \approx J^T(\mathbf{u})J(\mathbf{u}) + \sum_{i=1}^{n_c} \lambda_i \nabla_{\mathbf{u}}^2 c_i(\mathbf{u})$$

where  $J(\mathbf{u})$  is the Jacobian matrix of  $e(\mathbf{u})$  in (8). By focusing on the  $J^T(\mathbf{u})J(\mathbf{u})$  term of the above Hessian, and neglecting the latter term for ease of exposition, the Jacobian can be straightforwardly expressed as

$$J(\mathbf{u}) = \begin{bmatrix} Q^{1/2} \frac{\partial x_2}{\partial u_1^T} & 0 & \dots & 0 \\ Q^{1/2} \frac{\partial x_3}{\partial u_1^T} & Q^{1/2} \frac{\partial x_3}{\partial u_2^T} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ Q^{1/2} \frac{\partial x_{N+1}}{\partial u_1^T} & Q^{1/2} \frac{\partial x_{N+1}}{\partial u_2^T} & \dots & Q^{1/2} \frac{\partial x_{N+1}}{\partial u_N^T} \\ R^{1/2} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & R^{1/2} \end{bmatrix} \quad (11)$$

where the recursion for state derivatives can be exploited :

$$\frac{\partial x_{k+1}}{\partial u_j^T} = \frac{\partial f_k(x_k, u_k)}{\partial x_k^T} \frac{\partial x_k}{\partial u_j^T} + \frac{\partial f_k(x_k, u_k)}{\partial u_k^T} \frac{\partial u_k}{\partial u_j^T} \quad (12)$$

for  $j = 0, \dots, k$ . By computing the QR factorisation (see [7]) of  $J$

$$QR = J(\mathbf{u}) \quad (13)$$

where  $Q$  is an orthonormal matrix and  $R$  is upper triangular, it is possible to express the Hessian matrix as

$$H = J^T(\mathbf{u})J(\mathbf{u}) = \mathcal{R}^T Q^T Q \mathcal{R} = \mathcal{R}^T \mathcal{R}. \quad (14)$$

Therefore, identifying  $Z = \mathcal{R}^{-1}$  gives the property that

$$ZZ^T = \mathcal{R}^{-1} \mathcal{R}^{-T} = (\mathcal{R}^T \mathcal{R})^{-1} = H^{-1}. \quad (15)$$

The important feature is that  $\mathcal{R}$  is upper triangular and quite easily inverted to give  $Z$ . However, much of the computation effort involves the QR factorisation of a potentially large Jacobian matrix, as discussed in the following section.

## VI. MPC HARDWARE SOLUTION

The MPC approach presented above is applicable to a wide range of control problems, with many variables that will affect a particular implementation and the optimisations that can be found. These include the prediction horizon and number and type of constraints in the MPC problem, the required speed and precision of the calculation, and the allowable size and power consumption of the circuit.

To address a wide range of configurations, it is necessary to develop a set of alternate implementation strategies and scalable building blocks. For a given problem configuration, the most appropriate methods can be applied to a generalised architecture.

Figure 1 summarises such an architecture for the implementation of the active set method. This consists of a small number of memories and simple parallel arithmetic elements, which can be carefully coordinated by a state machine to perform the algorithm in an efficient manner. As a scalable architecture, it can be reconfigured to meet the requirements of a specific application.

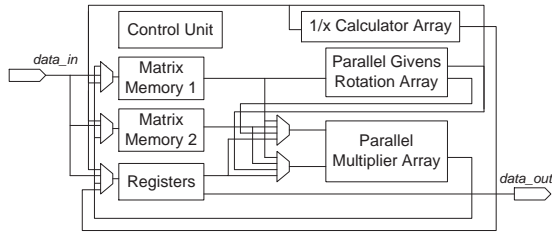


Fig. 1. Simplified depiction of an architecture that uses parallelism to implement the active set method.

The efficient implementation of MPC techniques is challenging, requiring a large amount of intensive processing in an efficient manner to maximise throughput. It also requires the handling of non-trivial operations such as square-roots, divisions, and specialised Givens rotations.

These will be examined in the following sections, beginning with Givens rotations and Householder reflections, which are key parts of the active-set algorithm (Section IV) and the computation of  $Z$  (Section V).

### A. QR Factorisation

The application of Givens rotations and Householder reflections are the two main methods for computing the QR factorisation [7]. Generally, Householder reflections are suited to the case of zeroing many entries in a vector (which is useful in the case of computing  $Z$ ), while Givens rotations are more suited

to zeroing out selected entries in a vector (which is useful in the active-set algorithm).

Fast QR factorisation in hardware has been explored in the literature, with a broad range of implementation features. Many implementations are based on the COordinate Rotation DIgital Computer (CORDIC) algorithm (see e.g. [16]), both Givens [5], [8] and Householder [9].

Of particular interest here is the scaled and decoupled Givens rotation discussed in [3]. In that algorithm, the square root and division operations are minimised by decoupling the numerator and denominator calculations and applying a scaling factor to ensure numerical stability. This type of approach will be adopted here and is discussed in the next section.

### B. Givens Rotations

The active-set algorithm mentioned in Section IV is heavily dependant on the use of rotations to maintain an upper triangular matrix  $U$ . On each iteration of the algorithm, either

- A constraint is added to the right of the triangular matrix, requiring the new column (constraint) to be rotated to meet the upper-triangular requirement, or
- A constraint is removed from anywhere in the matrix, requiring columns to the right to be adjusted by rotating the lowermost element.

For either case, the calculation of a Given's rotation of  $\begin{bmatrix} a \\ b \end{bmatrix}$  results in a Given's matrix

$$G = \begin{bmatrix} \frac{a}{\sqrt{a^2+b^2}} & \frac{b}{\sqrt{a^2+b^2}} \\ \frac{-b}{\sqrt{a^2+b^2}} & \frac{a}{\sqrt{a^2+b^2}} \end{bmatrix}. \quad (16)$$

However, by isolating the denominators, and an initial diagonal  $K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ , it is possible to express this as

$$G = \hat{K}^{-\frac{1}{2}} \hat{G} \quad (17)$$

$$\hat{G} = \begin{bmatrix} aK_{2,2} & bK_{1,1} \\ -b & a \end{bmatrix} \quad (18)$$

$$\hat{K} = \begin{bmatrix} \hat{K}_1 & 0 \\ 0 & \hat{K}_2 \end{bmatrix} \quad (19)$$

$$\hat{K}_1 = K_{1,1}K_{2,2}(K_{2,2}a^2 + K_{1,1}b^2) \quad (20)$$

$$\hat{K}_2 = K_{2,2}a^2 + K_{1,1}b^2. \quad (21)$$

The expressions (18,19) involve only multiplication and addition operations, which may be computed relatively quickly in a hardware implementation.

To obtain the original value of  $G$ , reciprocal square roots need to be applied to find  $K^{-\frac{1}{2}}$ , however this operation may be delayed until a convenient point

in the algorithm. By using such a technique on a QR decomposition of a matrix  $A$ , the decomposition becomes

$$A = \left(QK^{-\frac{1}{2}}\right) \left(K^{-\frac{1}{2}}R\right) = QK^{-1}R. \quad (22)$$

In the case of the active set algorithm, we have observed that continually delaying the application of  $K^{-\frac{1}{2}}$ , until later operations in the algorithm, results in the eventual cancellation of all such  $K^{-\frac{1}{2}}$  elements. Consequently, only the original diagonals of  $K$  are required in the modified algorithm, and the square-root need not be calculated.

### C. Householder Reflections

Householder reflections, denoted here by the matrix  $P \in \mathbb{R}^{n \times n}$ , transforms a vector  $x \in \mathbb{R}^n$  into a new vector  $y = Px = [\alpha, 0, \dots, 0] \in \mathbb{R}^n$  [7]. That is, it zeros out all but the first element of a vector. By successively applying such reflections to the columns of a matrix  $A \in \mathbb{R}^{n \times m}$ , it is possible to obtain the QR factorisation. The Householder reflection vector  $v$  can be calculated from  $x$  as follows.

```

function:  $[v, \beta] = \mathbf{house}(x)$ 
 $n = \text{length}(x), \sigma = x(2:n)^T x(2:n)$ 
 $v = \begin{bmatrix} 1 \\ x(2:n) \end{bmatrix}$ 
if  $(\sigma = 0), \beta = 0$ 
else
 $\mu = \sqrt{x(1)^2 + \sigma}$ 
if  $(x(1) \leq 0), v(1) = x(1) - \mu$ 
else,  $v(1) = \frac{-\sigma}{x(1) + \mu}$ 
end
 $\beta = \frac{2v(1)^2}{\sigma + v(1)^2}, v = \frac{v}{v(1)}$ 
end

```

The main operations in calculating  $v$  are multiplication and addition. However, operations of square-root and divide are more difficult to implement in hardware and cannot be eliminated from the Householder method, as was done for the Givens method.

In Section V it was seen that the Jacobian matrix has a specific structure, which can be exploited to approximately halve the computational effort. By observing the locations of the zeros in the Jacobian matrix, it is possible to compute the Householder vector more efficiently and also apply the reflection in an economical manner. Specifically, computation of the Householder vector  $v$  requires the norm of  $x$ , and the non-zero elements of  $x$  have a predetermined structure that can be utilised to skip the zero entries. In addition, the Householder vector will also have a number of zeros that can be exploited in the inner product calculation when applying the Householder reflection to the matrix  $A$ . Similar gains can be made

for the outer product term of applying a Householder reflection.

### D. Use of Parallelism

Significant speed increases may be achieved in an FPGA or ASIC implementation through the exploitation of parallelism that is not possible on typical DSP devices. A simple example, outlined in Figure 2, is the simultaneous calculation of some or all of the multiplications in a vector inner product. Such operations are required in the active-set method during the inclusion of new constraints to the set, and during the Givens rotations and Householder reflections.

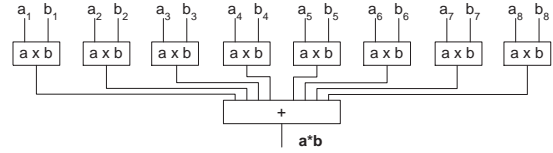


Fig. 2. Conceptual example of the use of parallelism. The product between a  $1 \times 8$   $8 \times 1$  vector can be found in only 2 steps, instead of the 8 accumulate and add steps required in a serial approach.

Further parallel optimisations are less evident, but may be found by careful examination of the algorithm. For example, Figure 3 demonstrates a typical method of performing a column rotation in a DSP, where pairs of cells are chosen for Givens rotations and one result depends on the result from the previous calculation. Combined with the post-rotation adjustment of affected matrices, this represents a significant processing time when done sequentially.

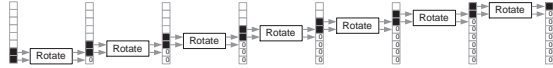


Fig. 3. Typical rotation method using a serial processing approach, requiring up to  $n$  iterations.

However, Figure 4 reveals that, by selecting the rotation points carefully, multiple rotations may be performed in parallel. Hence, a column rotation that previously required  $n$  iterations of single rotations, only requires  $\log_2 n$  iterations of parallel rotations. Furthermore, the post-rotation adjustment of the affected matrices may also be performed in parallel, greatly reducing the execution time. This technique can reduce execution time by 50% to 75%, with the effect more significant on larger sized problems (e.g. prediction horizon of 25) that cannot normally be implemented by a DSP.

Parallelism may also be applied to minimise latency by calculating certain data before it is required. This includes the reciprocal of data elements, and the implied inverse of the rotated matrix.

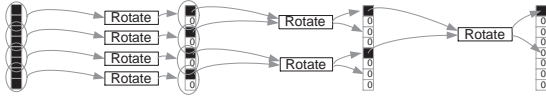


Fig. 4. By using parallel hardware and selecting rotation points carefully, the time required can be reduced to  $\log_2 n$  iterations.

### E. Numerical representation

The MPC algorithm may be implemented using a floating point numeric system, customised to maximise efficiency of implementation. This is used in favour over a fixed point system, to allow a larger dynamic range to be represented with a minimal number of bits. This in turn decreases circuit size and power consumption, and increases the speed of functions such as multiplication and division. While floating point systems require additional complexity for the rescaling of the mantissa to the  $[1..2)$  range after an operation, this is not dissimilar in complexity to the range checking and shifting required in a fixed point system.

In an attempt to gauge the required data word size for the MPC algorithm, we compared an IEEE-754 floating-point (denoted as single-precision, using 23 mantissa bits) implementation of the active-set algorithm with that of reduced mantissa sizes. The active-set algorithm comprises the most work in the algorithm, and is most affected by these optimisations.

In particular, we recorded several hundred QP's that arise in the computation of a control action of the inverted pendulum apparatus described at the end of Section III; it has 61 variables and 242 constraints. These QP's were solved using a single-precision version of the active-set method, and reduced precision versions of the active-set algorithm. The results in Table I demonstrate that a degradation in precision occurs as expected, and at the same time acceptable results are obtained even when using a mantissa with as low as 12-bits. However, this result will vary, depending on the characteristics of an individual application.

Bits	22	20	18	16	14	12	10
Max	5.2e-3	2.2e-2	1.8e-2	6.2e-2	2.9e-1	8.9e-1	5.2
Mean	5.7e-4	2.4e-3	5.1e-3	1.9e-2	8.4e-2	2.9e-1	1.5
St. Dev.	1.2e-3	5.0e-3	4.8e-3	1.8e-2	8.3e-2	2.8e-1	1.6

TABLE I

MAXIMUM, MEAN, AND STANDARD DEVIATION OF THE ERROR  $\|x_s - x_r\|_2 / \|x_s\|_2$ , BETWEEN THE IEEE-754 SINGLE-PRECISION ACTIVE-SET RESULT  $x_s$  AND THE REDUCED PRECISION RESULT  $x_r$ .

## VII. CONCLUSION

This paper has described an architecture and implementation strategies for performing MPC on FPGA and ASIC devices, by taking advantage of parallelism and numerical customisation that is possible on such devices. This is done with the aim of providing an efficient and cost effective means for physically realising high performance MPC controllers. Further work is underway to build and test various configurations of this design.

## REFERENCES

- [1] L.G. Bleris, P.D. Vouzis, M.G. Arnold, and M.V. Kothare. A Co-Processor FPGA Platform for the Implementation of Real-Time Model Predictive Control. In *Proceedings of the 2006 American Control Conference*, pages 1912–1917, Minneapolis, Minnesota, USA, June 14–16 2006.
- [2] P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1996.
- [3] L.M. Davis. Scaled and decoupled cholesky and qr decompositions with application to spherical mimo detection. *IEEE Wireless Communications and Networking*, 1:326–331, March 2003.
- [4] J.E. Dennis and B. Schnabel R. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, 1983.
- [5] K. Dickson, Z. Liu, and J. V. McCanny. Programmable processor design for givens rotations based applications. *Proc. 4th IEEE Workshop on Sensor Array and Multichannel Processing*, pages 84–87, 2006.
- [6] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1983.
- [7] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The John Hopkins University Press, 1996.
- [8] J. Gotze. Parallel methods for iterative matrix decompositions. *Proc. IEEE International Symposium on Circuits and Systems*, 1:232–235, June 1991.
- [9] S.F. Hsiao and J.M. Delosme. Householder CORDIC Algorithms. *IEEE Transactions on Computers*, 44(8):990–1001, August 1995.
- [10] T.A. Johansen, W. Jackson, R. Schreiber, and P. Tondel. Hardware synthesis of explicit model predictive controllers. *IEEE Transactions on Control System Technology*, 15(1):191–197, January 2007.
- [11] K.V. Ling, S.P. Yue, and J.M. Maciejowski. A FPGA Implementation of Model Predictive Control. In *Proceedings of the 2006 American Control Conference*, pages 1930–1935, Minneapolis, Minnesota, USA, June 14–16 2006.
- [12] David Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.
- [13] J. Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.
- [14] M.J.D. Powell. On the Quadratic Programming Algorithm of Goldfarb and Idnani. *Mathematical Programming Study*, 25:46–61, 1985.
- [15] S. Joe Qin and T.A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11:733–764, 2003.
- [16] J.E. Volder. The Birth of CORDIC. *Journal of VLSI Signal Processing*, 25(3):101–105, June 2000.
- [17] A.G. Wills, D. Bates, A.J. Fleming, B. Ninness, and S.O.R. Moheimani. Model predictive control applied to constraint handling in active noise and vibration control. *IEEE Transactions on Control Systems Technology*, 16(1):3–12, 2008.