Graphical FPGA Design for a Predictive Controller with Application to Spacecraft Rendezvous

Edward N. Hartley and Jan M. Maciejowski

***Abstract*— A reconfigurable field-programmable gate array (FPGA)-based predictive controller based on Nesterov's fast gradient method is designed using Simulink and converted to VHDL using Mathworks' HDL Coder. The implementation is verified by application to a spacecraft rendezvous and capture scenario, with communication between the FPGA and a simulation of the relative dynamics occuring over Ethernet. For a problem with 120 decision variables and 240 constraints, computation times of $0.95\ \mathrm{ms}$ are achieved with a clock rate of $50\ \mathrm{MHz}$, corresponding to a speed up of more than 2000 over running the algorithm directly on a MicroBlaze microprocessor implemented on the same FPGA.**

I. Introduction

The archetypal predictive controller (e.g. [1], [2]) solves a constrained, finite but receding horizon optimal control problem online at each time step. In its most prevalent forms, this problem is cast as a constrained convex quadratic program (QP) or a linear program (LP). The solution of these has a higher computational burden than linear feedback and this must be accommodated in the overall system design.

FPGAs are programmable silicon chips that allow implementation of custom digital circuits. In contrast to conventional microprocessors, the designer can choose the levels of parallelism and numerical precision at each stage in a numerical algorithm and trade these against hardware resource usage. Parallellism can accelerate the solution of constrained optimisation problems to meet real-time requirements whilst running at comparatively low clock frequencies.

The present paper demonstrates the implementation of a field programmable gate array (FPGA)-based predictive controller applied to the terminal phase of a spacecraft rendezvous and capture mission, as outlined in Section II.

FPGA design can be carried out in a number of ways. These include: directly using a register transfer language (RTL) (e.g. VHDL, Verilog), writing a custom generator to create RTL source from a set of parameters (e.g. [3], [4]), using a high level C-like language (System C, Vivado High Level Synthesis), or graphical methods (e.g. Xilinx System Generator for DSP, Mathworks HDL Coder). Whilst the former allow the most fine-grained control over the design, the latter are more accessible to the control systems designer, who may be an expert with MATLAB and SIMULINK, but only marginally familiar with VHDL or Verilog. Graphical visualisation of the design also helps its documentation and communication, ease of maintenance and re-use.

This work was supported by the EPSRC (Grant EP/G030308/1) as well as industrial support from Xilinx, Mathworks and the European Space Agency.

The authors are with the University of Cambridge (email: {enh20,jmm}@eng.cam.ac.uk).

The main focus of this paper, forming Section III, is the modelling of the circuit design using SIMULINK, to be translated to VHDL using Mathworks' HDL Coder. In [5], an interior point QP solver was implemented solely using built-in SIMULINK blocks to enable automatic code generation and to assist verification and validation processes. Elsewhere, SIMULINK Coder/MATLAB Coder (*née* Real Time Workshop/Embedded MATLAB (EML)) have been used to compile M-code implementations of custom QP/LP solvers to C to accelerate simulation and simplify deployment (e.g. [6]–[8]). For FPGA synthesis the process is rather more complex. It is insufficient to generate RTL from the same M-code used for simulation or C generation. Register level timing, numerical representation and parallelism of computation must also be considered to obtain a design suitable for an FPGA. The design from Section III is implemented on a Xilinx ML605 evaluation board, and results are presented in Section IV.

II. Background

A. Control scenario

The terminal phase of a spacecraft rendezvous and capture scenario is considered, based on the Mars Sample Return mission [6], [9], [10] in a circular orbit with radius of $3893.4\ \mathrm{km}$. The linearised Hill-Clohessy-Wiltshire equations (e.g. [11], [12]) are used for the prediction model, the controlled spacecraft has a nominal mass of $1575\ \mathrm{kg}$.

As in [13], the chaser spacecraft nominally starts at zero velocity, $200\ \mathrm{m}$ behind a passive target and $0.0767\ \mathrm{m}$ below the x-axis (positive z) in a cartesian reference frame centred on the target. The objective is to track a step increase in velocity in the x direction (towards the target) to $0.2\ \mathrm{ms}^{-1}$ until the separation has reduced to $100\ \mathrm{m}$ and then reduce to $0.1\ \mathrm{ms}^{-1}$ whilst holding $0.0767\ \mathrm{m}$ below (positive value) the z-axis, and null out-of-plane (y-axis) separation. At $\leq\ 3\ \mathrm{m}$ separation, the remainder of the manœuvre must be passive and the chaser trajectory will naturally drift upwards to intercept the x axis. Navigation error is modelled as Gaussian and white, with 3σ values of $[0.0247, 0.0247, 0.009, 0.007, 0.007]$ on $[y, z, \dot{x}, \dot{y}, \dot{z}]$ respectively [6]. As well as a minimum impulse bit, below which differential thrust is used to deliver the net commanded value, the thrust on each axis is subject to a multiplicative uncertainty with $\sigma = 0.01$.

An ℓ_1-regularised quadratic (ℓ_{asso}) cost function [13]–[16] is used to engender sparsity in the input trajectory, thus avoiding lengthy periods of continuous low-level thrust. At

each time step k, the MPC minimises the cost function

$$\|x_N - r\|_P^2 + \sum_{i=0}^{N-1} \left(\|x_i - r\|_Q^2 + \|u_i\|_R^2 + \|R_\lambda u_i\|_1 \right)$$

subject to the predicted plant dynamics and any input or state constraints, where $R > 0$ is diagonal and $Q \geq 0$. In the present design there are no state constraints, and inputs are bounded symmetrically: $-u_{\max} \leq u_i \leq u_{\max}$, where $u_{\max} = [8, ..., 8]^T$ (in Newtons). The 1-norm of $R_\lambda u_i$ is encoded by letting $R_\lambda > 0$ be diagonal, and splitting u_i into positive and negative components on each of 3 axes, so that $u_i = u_i^+ - u_i^-$, with $0 \leq u_i^+ \leq u_{\max}$, and $0 \leq u_i^- \leq u_{\max}$. Let $\tilde{u}_i = [u_i^{+T}, u_i^{-T}]^T$ and to ensure strong convexity, replace the input contribution to the stage cost with $u_i^{+T} R u_i^+ + u_i^{-T} R u_i^-$. Due to diagonality of R and positivity of $\tilde{u}_i$, the optimal u_i is equal to that of the original problem. The resulting constrained optimisation can be posed as a dense parametric QP

$$\min \theta^T H \theta + f^T \theta \quad \text{s.t. } \theta_{\min} \leq \theta \leq \theta_{\max}$$

where $\theta^T = [\tilde{u}_0^T, \tilde{u}_1^T, \ldots, \tilde{u}_{N-1}^T]$, $f = (F_r r + F_x x_0 + l)$, and the values of H, F_x, F_r, l, $\theta_{\min}$, $\theta_{\max}$ follow from condensing out the state variables in the usual way (e.g. [1]). The tunings are obtained as described in [13], aiming for a capture tolerance of 7.5 cm (tighter than the 20 cm required by the MSR mission).

In the present scenario, $N = 20$, and $\tilde{u}_i \in \mathbb{R}^6$. Therefore, after condensing, the number of decision variables, $n_\theta = 120$. A sampling period of $T_s = 1$ s is used. To avoid magnification of navigation error it is desirable to *not* allow a full unit delay for computation. A QP of this size is easy to solve in milliseconds on a contemporary desktop computer, but in space environments computer clock rates are between one and two orders of magnitude slower due to radiation hardening and power consumption requirements. Through parallelisation, an FPGA can achieve similar times whilst clocked at a few tens of MHz. The motivation for using the FPGA in this application is to get adequate computational speed at relatively low clock rates.

B. Optimisation algorithm

Nesterov's fast gradient method (FGM) (e.g. [17], [18]) is chosen, since only input constraints are considered in the presented control scenario and this class of algorithm is division-free, with a relatively straightforward data flow, offering natural opportunities for parallelism and pipelining. It is also demonstrably well-behaved with fixed-point arithmetic [4], [19]. Thus it lends itself well to FPGA implementation using the design methodology presented. The algorithm is shown in Fig. 1 for the case of a bound constrained QP, where θ is the decision variable, the superscript $\bullet^{(k)}$ indicates element of a sequence, and L is the largest eigenvalue of H. The value β is taken to be $(1-\alpha^{(\infty)})/(1+\alpha^{(\infty)})$, where $\alpha^{(\infty)} = \sqrt{\mu/L}$ and μ is the smallest eigenvalue of H.

For $k = 0$ **to** $k_{\max}$

1. $\nabla J^{(k)} = H y^{(k)} + f$
2. $\tilde{\theta}^{(k+1)} = y^{(k)} - \frac{1}{L} \nabla J^{(k)}$
3. $\theta^{(k+1)} = \max\{\theta_{\min}, \min\{\tilde{\theta}^{(k+1)}, \theta_{\max}\}\}$
4. $\Delta\theta^{(k+1)} = \theta^{(k+1)} - \theta^{(k)}$
5. $y^{(k+1)} = \theta^{(k+1)} + \beta \Delta\theta^{(k+1)}$

End for

Fig. 1. Fast gradient method for bound constrained QP

TABLE I
DATA TYPE SELECTION (ALL SIGNED)

	Bits	
Variable	Word	Frac.
H	18	16
$y^{(k)}$	25	22
$Hy^{(k)}$	25	22
f	41	22

	Bits	
Variable	Word	Frac.
$\tilde{\theta}^{(k)}$	42	22
$\theta^{(k)}, \theta_{\min\,/\,\max}$	25	22
$\Delta\theta^{(k)}$	25	22
β	18	16

III. Design

A. Scaling and data types

As highlighted in [17], [18], the QP can be scaled to minimise the condition number of H to improve convergence. Alternatively [20], [21] proposes a diagonal preconditioner, originally intended for an interior point QP solver with an iterative MINRES linear solver which puts the numerical values of a matrix into the range $[-1, 1]$ and $\mathrm{spec}(H) \subseteq (0, 1]$. The latter is used in the present implementation, since it simplifies the positioning of the fixed point, and also means that $L = 1$ (Fig. 1, Step 2), therefore eliminating a multiplication by $1/L$ from the final circuit design. To avoid additional notation, we henceforth use H to mean the Hessian term *after* the scaling has been applied.

The choice of fixed point data type is driven by the solution accuracy required, the dynamic range of the numerical values and the hardware available. The Xilinx Virtex 6 FPGA which will be targeted has a number of dedicated 25×18-bit hardware multipliers (DSP48Es). The data types used (shown in Table I) are chosen based on the FPGA resources, empirical testing and knowledge of the bounds of variables, rather than an aim to use the optimum number of resources for a specified accuracy. By application of [20], [21] only one integer bit is needed for H. Similarly, $\beta \in [0, 1]$. During matrix multiplication, elements of H multiply elements of $y^{(k)}$ therefore it is convenient for one of these to be (a multiple of) 18 bits long and the other to be (a multiple of) 25 bits long to optimally use the DSP48E resources. $y^{(k)}$ has a greater dynamic range, so therefore takes the longer word length. Noting that for this scenario $\theta_{\min}$ is a vector of zeros, $y^{(k)}$ and $\theta^{(k)}$ will be bounded above by $\max(\theta_{\max})$. After scaling, $\max(\theta_{\max}) = 1.5453$ so *at least* 1 integer bit is needed. $\beta(\theta^{(k+1)} - \theta^{(k)})$ is added to $\theta^{(k+1)}$ to get $y^{(k+1)}$ for $\beta \in [0, 1]$, so y must have at least one additional integer bit totalling 2 integer bits. β takes the same data type as H, and we let θ take the same type as y with 2 integer bits and 22 fractional bits. The data type of $\tilde{\theta}^{(k)}$ follows from allowing SIMULINK to use its internal rules rather than forcing a data type or using backwards propagation. Since data types are chosen based on pre-determined bounds, *all* summations and products are set to round to "Floor" and to

TABLE II

INPUT AND OUTPUT PORTS

LOADMODE	4-bit unsigned integer indicating which RAM the "write enable" input should be connected to.
DATA_IN	A 41-bit unsigned integer. This is reinterpreted as fractional fixed point data types as appropriate, starting from the least significant bit if the target data type is less than the full 41 bits.
WE_IN	A write enable signal - when high, the value on DATA_IN is written to the RAM indicated by LOADMODE. Each RAM has an counter associated with it that increments when WE_IN is high and resets when LOADMODE corresponds to another RAM.
TRIG_IN	Triggers the QP solver state machine.
ITERMAX_IN	12-bit unsigned integer corresponding to maximum number of QP iterations.
DOUT	Element of final QP solution.
RDY	Boolean signal is high when DOUT contains an element of the final QP solution.

"Wrap" on overflow to minimise hardware resources used.

Experiments using the MATLAB fixed-point toolbox verify sufficient accuracy for the application. Analysis of convergence with fixed point arithmetic is beyond the scope of this paper; for a formal theoretical framework the reader should consult the work of [4].

B. Fast gradient method implementation

The data-flow through the circuit is modelled at a clock-cycle level using SIMULINK. Counters and state machine logic are implemented using the "MATLAB Function Block", from which HDL Coder can also generate VHDL or Verilog (we use the former). It is designed so that rather than being hard-coded, the QP matrices can be loaded upon initialisation. (This is not repeated for each QP solution.)

1) External interface: The QP solver presents five input ports and two output ports (Table II).

2) Control logic: The main control logic consists of two counters and a switch. The first counter is the iteration number. On the final iteration (determined by the input `ITERMAX_IN`) the boolean signal `LASTITER` is set high. The second is the element of the vector y and/or row of H currently being addressed. When an element is being addressed the output `TRIG` is set high. This pulse passes through a series of shift registers with a delay of the same length as the numerical component of the circuit, allowing determination of when the iteration has been completed when it returns as input `we_y` (where "`we`" stands for "Write Enable"). The same happens with the read address — each subsystem outputs a delayed copy of this to enable memory instances in the next subsystem to be presenting the output from the correct memory location at the correct time. The delays take into account the latency of the RAMs later in the pipeline, avoiding extra delays in the main data path. Each of these blocks is implemented in a MATLAB function block, with persistent variables used to model registers storing data between time steps (Fig. 3).

The switch simply determines whether data from outside the circuit, or estimate of y from the previous iterate is passed

```
function [ADDR, TRIG1] = fcn(TRIG, RST, nvar)
%#codegen
% TRIG and RST are boolean block inputs
% nvar is a parameter

nbits = ceil(log2(nvar));     % Number of bits needed to address RAM
T = numerictype(0,nbits,0);
F = fimath('Overflowmode', 'wrap', ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', T.WordLength, ...
    'SumFractionLength', 0);

% Persistent variables
persistent k1; if isempty(k1), k1 = fi(0,T, F); end
persistent trigprev; if isempty(trigprev), trigprev = logical(0); end;
persistent running; if isempty(running), running = logical(0); end;

% Output counter and trigger if running
ADDR = k1;
TRIG1 = running;


if RST
    % Set running to 0 and k1 to 0 if reset is triggered
    running = logical(0);
    k1 = fi(0,T,F);
elseif running
    % Else if counter is running increment k1 until it equals nvar-1.
    % When it reaches nvar-1, set running to 0 and reset k1 to 0.
    if k1 < nvar-1
        k1 = k1+1;
    else
        k1 = fi(0,T,F);
        running = logical(0);
    end
end

% On a rising edge of TRIG, set running to 1
if TRIG && ~trigprev
    running = logical(1);
end
trigprev = TRIG;
```

Fig. 3. M-code for `Read_counter`

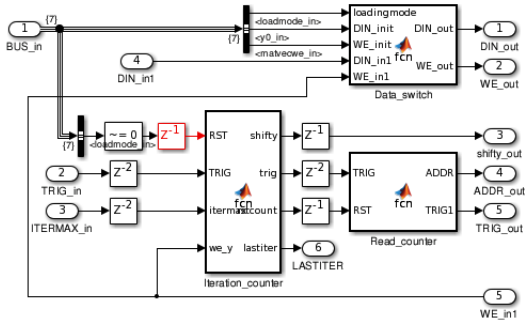


Fig. 4. Control logic

through to the next (gradient calculation) stage, based on the value of the external control signal "`loadmode_in`".

3) Gradient calculation: The gradient calculation (Step 1 in Fig. 1 and Fig. 5) comprises a matrix-vector multiplication followed by a vector-vector addition. This is performed in a pipelined manner, with a (maximum) throughput of one dot product of a matrix row with the vector per clock cycle. The version of HDL Coder used (R2012b) does not support matrix valued signals. Instead, the (scaled) matrix H is explicitly stored as an array of single-port RAMs (generated using the template in `hdldemolib`) each containing a column. All of these (i.e. the matrix row) are indexed by the same signal. Since the size of H depends on the problem formulation, a self-modifying masked subsystem is created. To load data into H, switching logic (not shown) is included so that when "`WE`" is high, the address is taken from a pair of resettable counters (determining the row address, which is connected to all RAMs (columns), and determine which column to set Write Enable high for) rather than from `RADDR`

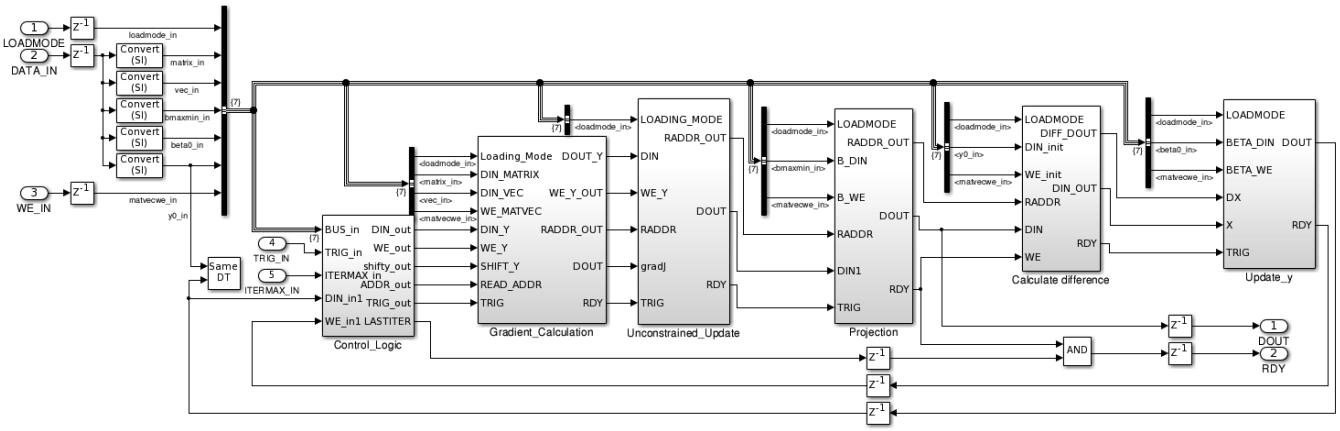


Fig. 2. Simulink block diagram implementation of fast gradient method

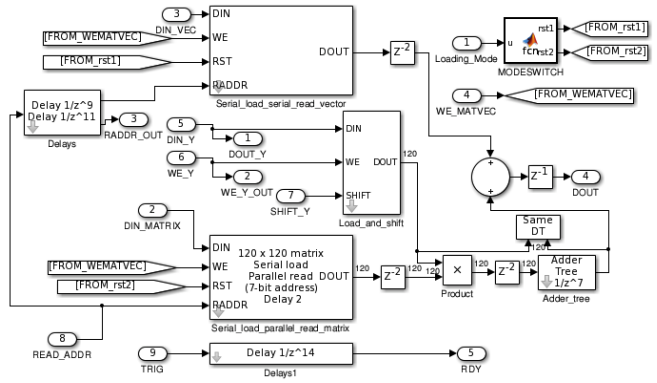


Fig. 5. Gradient calculation

(Read Address). The (scaled) vector f is stored in a single single-port RAM, with a similar arrangement to enable data to be initially loaded. The vector $y^{(k)}$ is stored in a tapped shift register. On the block "load_and_shift" when `WE` is high, the end of the vector takes on the value `DIN` and all previous values are shifted. When `SHIFT` is high, the values from the first shift register transfer in parallel into a bank of parallel registers. (This avoids replacing $y^{(k)}$ with elements of $y^{(k+1)}$ before it is finished with.)

The parallel element-by-element multiplication of each row of H with $y^{(k)}$ only requires a single SIMULINK block. The data type is allowed to propagate via internal rule at this point. The elements are then added together (as in the matrix-vector multiplication in [3]) by an adder reduction tree. The latter is generated using the SIMULINK "Sum of Elements" block, configured to use a "Tree" architecture (in HDL Block Properties) in series with a delay block. By placing this in a subsystem with HDL Coder's "Distributed Pipelining" enabled for that subsystem, the delay is then automatically distributed between the stages of the tree in the generated VHDL. $Hy^{(k)}$ is then converted back to the same data type as $y^{(k)}$ and will not overflow because $\mathrm{spec}(H) \subseteq (0, 1]$.

4) Gradient step: The gradient step (Step 2 in Fig. 1 and Fig. 6) employs a second copy of $y^{(k)}$ stored in a simple dual port RAM (again with an additional counter inside the

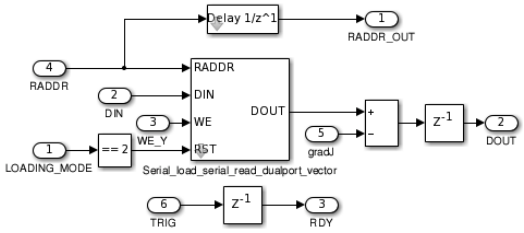


Fig. 6. Gradient step

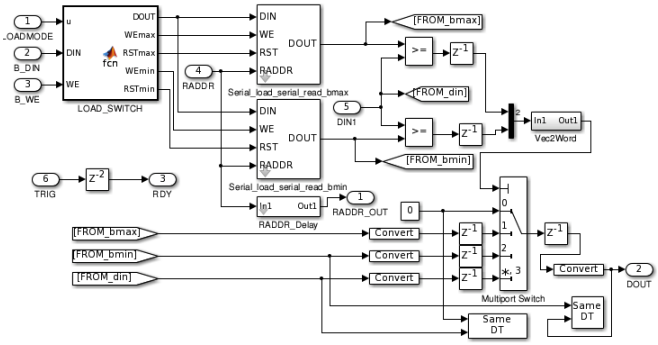


Fig. 7. Projection

subsystem for the write address) and a subtractor. The dual port RAM means that a write of element $y_i^{(k+1)}$ will not interfere with a read of element $y_j^{(k)}$.

5) Projection: The projection stage (Step 3 in Fig. 1 and Fig. 7) employs two single port RAMs, two comparators in parallel, and a multiport switch indexed by word obtained by concatenating the bits from the comparators.

6) Calculate difference: The next stage (Step 4 in Fig. 1 and Fig. 8) employs a further dual port RAM (with counter) which buffers $y^{(k)}$ and a subtractor.

7) Update y: The final stage (Step 5 in Fig. 1 and Fig. 9) contains a register (containing β), a multiplier and an adder.

C. Calculation of f

Vector f is a function of contant matrices F_x, F_r, vector l, and varying parameters x and r. Letting $\hat{F} = [F_x, F_r, l]$, and $\hat{x} = [x_0^T, r^T, 1]^T$, $f = \hat{F}\hat{x}$ — another matrix multiplication.

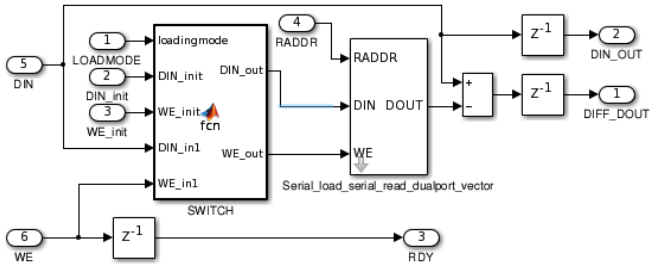


Fig. 8. Calculate difference

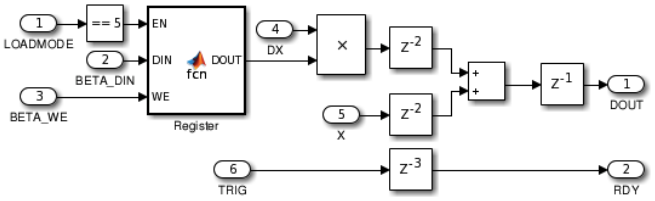


Fig. 9. Update y

This is implemented as a separate block outside of the main algorithm (not shown for space reasons). To simplify the external interface, the first three input ports of the FGM block are replicated (although DIN is here a 32-bit unsigned integer, extended to 41-bit before being re-output), and passed through modified or unmodified depending on the value of LOADMODE. The vector $\hat{x}$ is input in serial as a 32-bit signed data type with 20-bit fraction length. $\hat{F}$ is stored as a sequence of its row vectors with a 25-bit signed data type with 20-bit fraction length. The matrix-vector multiplication is performed serially since the vector is only of length 11 (5 states, 5 references and a constant of unity) and this only happens once per QP. When complete, the "TRIG" output, connected to the FGM state machine, is raised. The 32×25 bit multiplication requires two DSP48E units. The 32-bit word is manually split into two parts and long multiplication carried out with a pipeline stage in-between. The latency of calculating f is (after loading) $(n_\theta \times 11) + 9$ cycles, where n_θ is the number of decision variables.

D. General design considerations

Addition, subtraction, comparison and switching are chained with a delay of one cycle. Each multiplier is chained with a pipeline delay of two cycles, since the DSP48E units contain two built in pipeline stages. Whilst HDL Coder generates generic VHDL, this construct is inferred by the Xilinx ISE 14.4 toolchain used to synthesise and implement the generated design. The total latency of a single FGM iteration is $n_\theta + \log_2(n_\theta) + 19$. (For $n_\theta = 120$, the main path latency is 24, with an extra delay of 2 cycles in the iteration counter.)

HDL Coder is configured so that any "reset" signals are synchronous and active high. All delay blocks and "MATLAB function" blocks are configured with reset type "none" to allow the Xilinx toolchain to efficiently synthesise the design.

The resulting VHDL synthesises with an estimated feasible operating clock frequency of 261 MHz on the Virtex 6 LX240T FPGA on the ML605 Evaluation Board targeted with default settings in the Xilinx ISE toolchain. Whilst for the proposed application this is inappropriately high, there is flexibility (due to pipeline registers) to route the design in a smaller FPGA, or around another circuit on the same FPGA.

TABLE III
RESOURCE UTILISATION FGM (MICROBLAZE)

Resource	Register	LUT	Block RAM	DSP48E
Total	8119(19108)	3852(17047)	66(24)	123(5)
V6 LX240T	2.7%(6.3%)	2.6%(11.3%)	15.9%(5.8%)	16.0%(0.7%)
V6 LX75T	8.7%(20.5%)	8.3%(36.6%)	42.3%(15.4%)	42.7%(1.7%)
A7 50SL	1.9%(4.6%)	1.8%(8.1%)	69.5%(25.3%)	68.3%(2.8%)
K7 70T	9.9%(23.3%)	9.4%(41.6%)	48.9%(17.8%)	51.3%(2.1%)
V7 585T	1.1%(2.6%)	1.1%(4.7%)	8.3%(3.0%)	9.8%(0.4%)
Zynq 7020	7.6%	7.2%	47.1%	55.9%

IV. INTEGRATION AND TESTING

The designed circuit is connected as a peripheral core to a Xilinx MicroBlaze soft core processor in a similar way to [22] (although here, the generated VHDL code is imported into Xilinx System Generator for DSP, which is used to generate the memory-mapped interface logic). The MicroBlaze has two tasks. The first is as a convenient bridge to the outside world using UDP over 100 MBit/s Ethernet. The second is to initialise the QP matrices with their operational values, by loading them into the DATA_IN port with an appropriate schedule of values of LOADMODE. The FGM circuit and the MicroBlaze are clocked at 50 MHz.

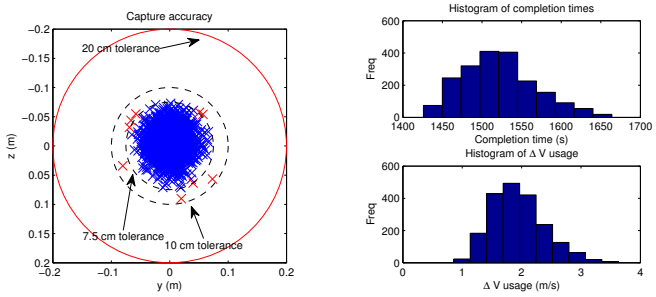
FPGA resource usage is presented in Table III, based on synthesis estimates. The design comfortably fits inside the smallest Virtex 6 FPGA (V6 LX75T). The register and lookup-table (LUT) usage of the custom FGM circuit is small compared with that used by the MicroBlaze, whilst the Block RAM and DSP48E usage is substantially greater.

For a nominal scenario with measurement and thrust uncertainty, but the simulation model otherwise matching the prediction model including an assumption of instantaneous control on sampling, a comparison of the FPGA-based solver with EML-based FGM implementations and an FGM solver synthesised using FiOrdOs (version R20121210) [23] is presented in Table IV (all FGM solvers carry out 300 iterations). The loss of accuracy for the fixed point implementations is small. The FGM/EML solver on the desktop is faster than FiOrdOs due to use of BLAS (without this enabled, they are comparable) but slightly slower on the MicroBlaze since BLAS is not used. The fixed point software implementations are slow due to use of multi-word arithmetic. The MicroBlaze has a single precision hardware floating point unit — if emulation is used instead, the QP takes almost 40 seconds. The computation time for the FPGA-based design is more than 2000 times faster than the fastest software implementation on the MicroBlaze. The FPGA clock frequency could be halved or even quartered, whilst keeping computation time insignificant with respect to T_s. However, communication time is significant for the FPGA-based implementation and other interfacing technologies would be more appropriate if used with a real plant.

TABLE IV

QP SOLVER TIMING AND ACCURACY COMPARISON

Solver	Precision	Mean Time (ms)	Norm err (u_0) Max	Norm err (u_0) Mean
2.8 GHz Desktop (via MEX)				
CPLEX	Dbl	7.9	–	–
FGM/EML	Dbl	3.6	2.7×10^{-4}	2.7×10^{-7}
FGM/EML	Sgl	2.7	2.7×10^{-4}	7.5×10^{-7}
FGM/EML	Fix	26.0	3.4×10^{-3}	9.9×10^{-5}
FGM/FiOrdOs	Sgl	5.6	2.4×10^{-4}	4.0×10^{-7}
50 MHz MicroBlaze				
FGM/EML	Sgl	2334	*(Same as desktop)*	
FGM/EML	Fix	7426	*(Same as desktop)*	
FGM/FiOrdOs	Sgl	2003	*(Same as desktop)*	
50 MHz Custom FPGA Circuit				
FGM	Fix	0.95 + 0.04 ms transfer Microblaze to FGM + 0.92 ms Ethernet communication	3.4×10^{-3}	6.7×10^{-5}



(a) $z - y$ position at capture (b) Histograms

Fig. 10. FPGA-in-the-loop control performance over 2000 simulations with perturbed parameters

To verify the control system, a Monte-Carlo simulation with the FPGA-based controller in the loop (communicating using UDP/IP over Ethernet) is run from a range of initial conditions surrounding the nominal condition. Mismatch is introduced into the simulation model parameters. The seed used for thrust and navigation error is also varied. The initial position, velocity, orbital radius and mass are sampled from uniform distributions of range $(\pm 10\ \mathrm{m}, \pm 0.3\ \mathrm{ms}^{-1}, \pm 50\ \mathrm{km}, \pm 235\ \mathrm{kg})$ respectively. Figure 10 shows the position in the $y - z$ plane at capture for each of the simulations and histograms of capture time and cumulative impulsive velocity change (ΔV). All are well within the $20\ \mathrm{cm}$ tolerance considered in [6]. All are within a $10\ \mathrm{cm}$ tolerance with 99.55% within $7.5\ \mathrm{cm}$ of the origin.

V. CONCLUSIONS

This paper has presented the design of a constrained MPC controller with a ℓ_{asso} cost on an FPGA, using fixed point arithmetic and implemented using SIMULINK and HDL Coder. The design was tested on an FPGA evaluation board, connected to the outside world using a MicroBlaze soft core processor and UDP/IP over Ethernet. The system is demonstrated in a closed-loop with a simulated plant, and delivers the solution within $1.91\ \mathrm{ms}$ (of which $0.96\ \mathrm{ms}$ was communication) despite a clock frequency of only $50\ \mathrm{MHz}$.

REFERENCES

[1] J. M. Maciejowski, *Predictive Control with Constraints*. Harlow, U.K.: Prentice Hall/Pearson Education, 2002.

[2] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Madison, WI: Nob Hill Publishing, 2009.

[3] J. L. Jerez, K. V. Ling, G. A. Constantinides, and E. C. Kerrigan, "Model predictive control for deeply pipelined field-programmable gate array implementation: Algorithms and circuitry," *IET Control Theory Appl.*, vol. 6, no. 8, pp. 1029–1041, May 2012.

[4] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded predictive control on an FPGA using the fast gradient method," in *Proc. European Control Conf.*, Zurich, July 17–19 2013, pp. 3614–3620.

[5] A. Richards, W. Stewart, and A. Wilkinson, "Auto-coding implementation of model predictive control with application to flight control," in *Proc. European Control Conf.*, Budapest, Hungary, Aug. 23–26 2009, pp. 150–155.

[6] E. N. Hartley, P. A. Trodden, A. G. Richards, and J. M. Maciejowski, "Model predictive control system design and implementation for spacecraft rendezvous," *Control Eng. Pract.*, vol. 20, no. 7, pp. 695–713, Jul. 2012.

[7] M. Kögel and R. Findeisen, "Fast predictive control of linear systems combining Nesterov's gradient method and the method of multipliers," in *Proc. 50th Conf. Decision and Control and European Control Conf.*, Orlando, FL, Dec. 12–15 2011, pp. 501–506.

[8] G. Binet, R. Krenn, and A. Bemporad, "Model predictive control applications for planetary rovers," in *Proc. Int. Symp. Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, Turin, Italy, Sep. 4–6 2012.

[9] F. Mura, "Mars Sample Return: ENG-02 Mission Architecture Definition," European Space Agency, Tech. Rep., 2007.

[10] D. Beaty, M. Grady, L. May, and B. Gardini, "Preliminary planning for an international Mars Sample Return mission," Report of the International Mars Architecture for the Return of Samples (iMARS) Working Group, Jun. 2008.

[11] W. Fehse, *Introduction to Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 2003.

[12] M. J. Sidi, *Spacecraft dynamics and control: A practical engineering approach*. Cambridge University Press, 1997.

[13] E. N. Hartley, M. Gallieri, and J. M. Maciejowski, "Terminal spacecraft rendezvous and capture using LASSO MPC," *To Appear in Int J. Control*, 2013.

[14] H. Ohlsson, F. Gustafsson, L. Ljung, and S. Boyd, "Trajectory generation using sum-of-norms regularization," in *Proc. 49th IEEE Conf. Decision and Control*, Dec. 15–17 2010, pp. 540–545.

[15] M. Nagahara and D. E. Quevedo, "Sparse representations for packetized predictive networked control," in *Proc. 18th IFAC World Congress*, Milano, Italy, 2011, pp. 84–89.

[16] M. Gallieri and J. M. Maciejowski, "LASSO MPC: Smart regulation of over-actuated systems," in *Proc. American Control Conf.*, Montreal, Canada, Jun. 27–29 2012, pp. 1217–1222.

[17] S. Richter, C. N. Jones, and M. Morari, "Real-time input-constrained MPC using fast gradient methods," in *Proc. 48th IEEE Conf. Decision and Control and 28th Chinese Control Conf.*, Shanghai, China, Dec. 15–18 2009, pp. 7387–7393.

[18] ——, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Trans. Automat. Control*, vol. 57, no. 6, pp. 1391–1403, 2012.

[19] P. Zometa, M. Kögel, T. Faulwasser, and R. Findeisen, "Implementation aspects of model predictive control for embedded systems," in *Proc. American Control Conf.*, Montreal, Canada, Jun. 27–29 2012, pp. 1205–1210.

[20] E. C. Kerrigan, J. L. Jerez, S. Longo, and G. A. Constantinides, "Number representation in predictive control," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Noordwijkerhout, NL, Aug. 23–27 2012, pp. 60–67.

[21] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "Towards a fixed point QP solver for predictive control," in *Proc. 51st IEEE Conf. Decision and Control*, Maui, HI, USA, Dec 2012.

[22] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control of a Boeing 747 aircraft using an FPGA," in *Proc. IFAC Conf. Nonlinear Model Predictive Control*, Noordwijkerhout, NL, Aug. 23–27 2012.

[23] F. Ullmann, "FiOrdOs: A Matlab Toolbox for C-Code Generation for First Order Methods," Master's thesis, ETH Zurich, Zurich, 2011.