Predictive Control for Spacecraft Rendezvous in an Elliptical Orbit using an FPGA

Edward N. Hartley and Jan M. Maciejowski

Abstract—A field programmable gate array (FPGA)-based predictive controller for a spacecraft rendezvous manoeuvre is presented. A linear time varying prediction model is used to accommodate elliptical orbits, and a variable prediction horizon is used to facilitate finite time completion of manoeuvres. The resulting constrained optimisation problems are solved using a primal dual interior point algorithm. The majority of the computational demand is in solving a set of linear equations at each iteration of this algorithm. To accelerate this operation, a custom circuit is implemented, using a combination of Mathworks HDL Coder and Xilinx System Generator for DSP, and used as a peripheral to a MicroBlaze soft core processor. The system is demonstrated in closed loop by linking the FPGA with a simulation of the plant dynamics running in Simulink on a PC, using Ethernet.

I. INTRODUCTION

IMPLICIT generation of a control law through solution of a constrained optimisation problem makes model predictive control (MPC) particularly suitable for control scenarios where reconfiguration is necessary. The prediction model, cost function and constraints can be updated online to reflect changes in system parameters or control objectives. In [1]–[4], linear time-varying (LTV) prediction models are used to perform spacecraft manoeuvres in elliptical orbits. Simulations show that fuel consumption using MPC is favourable in comparison to other methods. Variable horizon predictive control (VH-MPC) is a variant of MPC, suited to “completion” type control problems, such as aerial vehicle manoeuvres [5] and spacecraft rendezvous [4], [6], [7] due to its finite-time completion properties. Exploiting this, [4] presents a family of VH-LTV-MPC controllers to perform a spacecraft rendezvous operation, with a controlled chaser spacecraft performing rendezvous with a passive target in an orbit around Mars [8].

VH-LTV-MPC has a high computational burden, and a contemporary guidance, navigation and control computer would require a dedicated co-processor to support the MPC. Instead of using an extra high-performance general purpose microprocessor, an alternative would be to employ a custom peripheral circuit, designed to carry out all or part of the predictive control function, trading high clock frequencies (problematic in space applications due to radiation exposure) for parallelism, and offering opportunities to reduce power consumption.

A. Background: FPGA based predictive control

Field programmable gate arrays (FPGAs) are configurable silicon devices that enable a system designer to implement custom circuits without the need to produce custom chips.

Recently there has been interest in using FPGAs to exploit opportunities for parallelism in the numerical algorithms used to implement MPC, with a variety of approaches proposed. A hybrid software-hardware design is advocated by [9]–[11]. The first uses a primal barrier method and the custom accelerator is used to calculate the gradient vector and Hessian matrix of the augmented cost function at each iteration, whilst the latter two use an active set method, with a custom circuit accelerating matrix/vector-vector multiplication. Full hardware implementations of MPC controllers are documented in [12] (active set), [13]–[16] (interior point), although [16] uses a soft-core microprocessor to bridge between the MPC and the outside world. At the other end of the spectrum, [17] implements MPC using a custom soft-core processor with non-standard numerical representations to minimise the FPGA resource (and power) requirements. With the exception of [15], [16], systems with one or two inputs are considered, and no prior FPGA-based predictive controllers consider variable prediction horizons and time-varying prediction models.

B. Summary

This paper presents an FPGA-based implementation of a variable horizon predictive controller, for a phase of the spacecraft rendezvous application in an elliptical orbit, based on a simplified version of one modes of operation from [4]. The design is implemented and tested on a Xilinx ML605 Evaluation Board [18], using Ethernet to communicate with a simulation of the spacecraft dynamics running in Simulink on a PC. Resource usage and power consumption are analysed.

An uncondensed MPC form is used. This scales favourably in terms of prediction horizon [19] and is advantageous for model reconfiguration, since once the time varying state space matrices are computed, the remaining effort in constructing the optimisation problem is data transfer rather than the multiplication of large, dense matrices.

A primal-dual interior point (PD-IP) algorithm [20] is implemented to solve the constrained optimisation problems. Following the hybrid software-hardware paradigm, the algorithm is co-ordinated by a Xilinx MicroBlaze soft-core processor, and a custom peripheral circuit is designed to assist in the construction and to perform the solution of the set of linear equations that arise at each iteration of the PD-IP algorithm. This is the bottleneck in optimisation problems.

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).
of the scale considered. In contrast to prior designs, where register transfer languages (RTL) such as Verilog [10], VHDL [12], [14]–[16], or high level C-like languages (System-C, [13]) have been used, the custom peripheral presented here is designed using a combination of Mathworks HDL Coder 2012a and Xilinx System Generator for DSP 14.2. This gives almost the same fine-grained clock-cycle accurate control of the design that an RTL would offer, but, being based upon Simulink, enables high-level visualisation of connectivity between components and rapid simulation.

The remainder of the paper is organised as follows: Section II develops the control problem; Section III describes the FPGA-based implementation; Section IV analyses the FPGA resource usage and power consumption; Section V describes the test-bench setup and presents closed-loop performance results; and Section VI concludes.

II. PREDICTIVE CONTROL PROBLEM

The scenario considered here is the medium-short range phase of the rendezvous in the Mars Sample Return mission [4], [8], starting at the point where the controlled spacecraft (chaser) has been brought into approximately the same orbit as the spacecraft with which it must rendezvous (target), but with a significant separation in true anomaly (i.e. an in-track separation of a few tens of kilometres). The control objective is to guide the chaser towards the vicinity of the target via a sequence of pre-determined “holding points” at which it must remain until given clearance. The time spent at holding points is unknown a priori, so at a given instant the goal is to enter the next holding point. The scenario terminates once the final holding point has been reached.

The Yamanaka-Ankersen (Y-A) equations [21] describe the dynamics of the relative position and velocity of the chaser with respect to the target in a local-vertical-local-horizontal (LVLH) reference frame centred on the target, with the z-axis pointing towards the central body, the y-axis normal to the orbital plane, and the x-axis completing a right-handed set. Unlike the LTI Clohessey-Wiltshire (C-W) equations (e.g. [22]), these apply to elliptical orbits, but are parameter varying in terms of the true anomaly, \( \nu \) of the target. In the MSR capture scenario, the target is passive, so the Y-A equations are LTV, since \( \nu \) is only a function of time.

As in [4], [6], a 1-norm cost is used to reflect propellant consumption being directly proportional to delivered thrust. A variable control horizon is used to enable a compromise between completion time and propellant usage. Letting \( x_i \in \mathbb{R}^n \) and \( u_i \in \mathbb{R}^m \) denote the predicted state and control input \( i \) time steps into the future, \( x_T \in \mathbb{R}^n \) denote a target setpoint, \( x(k) \) denote the actual measured (estimated) state at time \( k \in \mathbb{Z}_+ \), \( \gamma > 0 \in \mathbb{R} \) be a weighting parameter, and \( \theta(k) \triangleq (x_0, u_0, \ldots, x_N) \), the optimisation problem posed is

\[
J^* = \min_{\theta} \sum_{i=0}^{N-1} (1 + \gamma \|u_i\|_1) \\
\text{s.t. } x_0 = x(k), \quad x_{i+1} = A_i x_i + B_i u_i, \quad i \in \{0, \ldots, N-1\}, \quad x_N = x_T (k+N), \quad 0 \leq u_i \leq u_{\text{max}}, \quad 0 \leq N \leq N_{\text{max}}.
\]

where \( A_i \) denotes the Y-A state transition matrix corresponding to a prediction from time step \( i \) to \( i+1 \). The plant input is modelled as an impulsive change in velocity, partitioned into positive and negative components aligned with the three axes, so \( B_i = A_i \begin{bmatrix} 0 & 0 \\ I_3 & -I_3 \end{bmatrix} \). Control \( u(k) = u_0 \) is applied to the plant, the rest of \( \theta(k) \) is discarded and the process is repeated at the next sampling instant. To improve numerical conditioning of the matrices within the PD-IP algorithm, the state and input vectors are scaled so that relative positions are in units of 20 m, velocities in units of 1/60 m/s and inputs in units of 1/330 m/s.

The variable horizon problem (1) can be posed as a sequence of \( N_{\text{max}} \) linear programs (LPs), each with fixed \( N \). An LP is QP with a zero Hessian matrix, and the PD-IP algorithm for QP does not require a strictly positive definite Hessian matrix. Therefore, to facilitate future component reuse (for example for trajectory tracking during the final few metres of the rendezvous, for which a quadratic cost is more appropriate), a QP solver is implemented. The QP representation of (1), with \( N \) fixed is

\[
\min_{\theta} \frac{1}{2} \theta^T H \theta + h^T \theta
\]

subject to \( F \theta = f \) and \( G \theta \leq g \) where, letting \( \oplus \) denote the direct matrix sum, \( \otimes \) denote the Kronecker product, \( I_p \) denote the \( p \)-dimensional identity matrix, \( 0_{q\times r} \) denote a \( q \times r \) matrix of zeros and \( 1_{q\times r} \) denote a \( q \times r \) matrix of ones, \( H = (\bigoplus_{i=0}^{N-1} H_i) \oplus H_N \), \( h = \begin{bmatrix} I_N \otimes h_1 \\ h_N \end{bmatrix} \), \( F = \begin{bmatrix} I_N \otimes [-I_{n_1}, 0_{N_n \times n_1}] & 0_{N_n \times n_1} \\ 0_{n_1 \times N_n} & -I_{n_1} \otimes I_{N_n} + \bigoplus_{i=0}^{N-1} \begin{bmatrix} A_i & B_i \\ 0_{n_1 \times N_n} & F_N \end{bmatrix} & 0_{n_1 \times N_n} \\ 0_{n_1 \times N_n} & F_N \end{bmatrix} \),

\[
f = \begin{bmatrix} -x(k) \\ 0_{N_n \times 1} \end{bmatrix}, \quad G = \begin{bmatrix} \bigoplus_{i=0}^{N-1} G_i \\ 0_{N_n \times n_1} \end{bmatrix}, \quad g = [I_N \otimes g_1].\]

with fixed \( N \), every \( H_i = 0_{n_1 \times n_1}, h_i = [0_{1 \times n_1} \gamma 1_{n_1 \times 1}]^T, F_N = I_{n_1}, f_N = x_N, G_i = \begin{bmatrix} 0_{n_1 \times n_1} & -I_{n_1} \\ 0_{n_1 \times n_1} & I_{n_1} \end{bmatrix}, \quad \text{and } g_i = [0_{n_1 \times 1} \ | u_{\text{max}}].

III. MPC CONTROLLER ARCHITECTURE

Since the cost function (1a) in problem (1) is not convex with respect to the \( N \), each of the convex constrained optimisation problems corresponding to horizons \( N = \{1, \ldots, N_{\text{max}}\} \) is solved in sequence according to Algorithm [1] A Xilinx MicroBlaze soft-core processor (with floating point unit) is used to calculate the LTV prediction matrices, and provide data to a custom peripheral core designed to solve the set of linear equations that occur at each iteration of the PD-IP algorithm, which solves the optimisation problem occurring at step 3 of Algorithm [1] Using the annotation (M) to denote a process performed on the MicroBlaze and (F) to indicate that the process happens in the custom logic implemented in the FPGA fabric, Algorithm [2] shows the software-hardware
Algorithm 1: Variable horizon MPC controller

Initialisation:
1. $J_{opt} = \infty$, $N_{opt} = 0$, $\theta_{opt} = 0$;
for $j = 1$ to $N_{max}$
2. Calculate $A_j$, $B_j$ using [21];
3. Solve (1) s.t. $N = j$. If feasible, $J_j = J^*$, else $J_j = \infty$;
if $J_j < J_{opt}$
4. $J_{opt} = J_j$, $N_{opt} = j$, $\theta_{opt} = \theta^*$;
end if
end for.
5. Return $u_0$ from $\theta_{opt}$.

Algorithm 2: PD-IP Algorithm

1. Initialisation: $\theta_0$, $\lambda_0$, $s_0$, $z_0$, $\sigma$, $l_{IP}$
for $k = 0$ to $l_{IP} - 1$

Linearisation:
2. $\mu_k = z_k^T s_k/N_{n_k}$
3. $w_k = z_k/s_k$ (elementwise)
4. $m_k = \begin{bmatrix} \theta_k \\ \lambda_k \\ \sigma \mu_k \mathbf{l}_{n_k}/s_k - \text{diag}(w_k)g_k + z_k \end{bmatrix}$
5. $\Phi_k = \begin{bmatrix} H + \Phi_k & F^T \\ F & 0 \end{bmatrix}$
6. $\omega_k = \begin{bmatrix} -h & -H + \Phi_k & F^T & G^T \end{bmatrix} m_k$

Solve:
7. $\omega_k c_k = b_k$, where $c_k = \begin{bmatrix} \Delta \theta_k \\ \Delta \lambda_k \end{bmatrix}$
8. Calculate $G(\theta_k + \Delta \theta_k) - g$
9. $\Delta s_k = \text{diag}(w_k)(G(\theta_k + \Delta \theta_k) - g) + \sigma \mu_k s_k^{-1}$
10. $\Delta s_k = -G(\theta_k + \Delta \theta_k) - s_k$

Line search:
11. $\alpha_k = \max_{[0,1]} \alpha: \frac{\lambda_k + \alpha \Delta \lambda_k}{s_k + \alpha \Delta s_k} > 0$

Update:
12. $(\theta, \lambda, z, s)_{k+1} = (\theta, \lambda, z, s)_k + \alpha \Delta(\theta, \lambda, z, s)_k$ (M)
end for

The number of iterations is fixed, since in a real time setting, the time to obtain a solution (even if not optimal) should be bound. Some of the optimisation problems might be infeasible for shorter horizons $N$. Three heuristics are used to attempt to identify infeasibility and/or a poor solution after $l_{IP}$ iterations: the solution is assumed infeasible if for any $j$, $|\|b_j\|_\infty + N_{n_k} \mu_j| > 10^4 \min_j (\|b_j\| + N_{n_k} \mu_j)$. [23] and the solution is also rejected as likely infeasible if $\mu_{l_{IP}-1} > 10^{-3}$ or $|\|b_{l_{IP}-1}\|_\infty > 0.01|\|b_0\|_\infty$ (i.e. the result has not converged sufficiently).

As in [15], [16], the system of linear equations is solved using the minimum residual (MINRES) algorithm. In the present context, this iterative method can be considered preferable to factorisation-based methods, being dominated by matrix-vector multiplications, which are very easily parallelised, and the number of iterations (i.e. the solution time) can be traded for solution accuracy.

At a high level, the custom peripheral core (PCORE) is split into two major parts. The first component performs steps (5-7) of the Algorithm 2 building the linear system, whilst the second, more complex component performs step 8.

Fig. 1. High-level architecture of controller implementation

The PCORE is connected to the MicroBlaze using the AXI4-lite bus, which presents data and status registers as shared memory locations (Figure 1).

A. Linear system builder

The linear system builder is designed using Xilinx System Generator for DSP, and consists of four main modes of operation. In the first mode, the vectorised matrices $A_i$, $B_i$, $G_i$, $F_i$, $H_i$ (in row major form) and vectors $w_k$, $z_k$ and $[-h^T, f^T]^T$ are communicated from the MicroBlaze to the peripheral via a shared FIFO RAM. Five additional shared RAMs of length $(N_{max} + 1)$ are used, with element $i$ of each of these RAMs containing the address of the first element of $A_i$, $B_i$, etc. (Figure 2). Consequently $A_i$ etc. do not need to be stored sequentially and can be repeated (e.g. for LTI cases) without the overhead of re-writing the time invariant matrices to the PCORE.

The second mode triggers a state machine which calculates $G^T \text{diag}(w_k)G + H$ and stores this row-wise in a further RAM of size $N_{max}(n_k + n_u)^2 + n_\sigma^2$. This calculation requires $N_{max}(n_k + n_u)^2$ dot products per PD-IP iteration. To achieve a throughput of one scalar addition per clock cycle within each dot product (after an initial latency), elementwise multiplication is performed in single precision floating point using a pipelined IP-core, then converted to 64-bit signed fixed point with a 32-bit fractional part [11]. The accumulated result is transformed back to single precision floating point.

The third mode triggers a second state machine which addresses the RAMs to construct a sequence of numbers comprising the rows of a compact block representation of
\[ A_k: \]

\[
\begin{bmatrix}
H_{\Phi x, 0} & H_{\Phi y, 0} & -I & A_f^T \\
H_{\Phi x, 0} & H_{\Phi y, 0} & 0 & B_f^T \\
H_{\Phi x, 1} & H_{\Phi y, 0} & -I & A_f^T \\
H_{\Phi x, 1} & H_{\Phi y, 0} & 0 & B_f^T \\
\vdots & \vdots & \vdots & \vdots \\
H_{\Phi x, N} & 0 & -I & F_N^T \\
-I & 0 & 0 & 0 \\
-A_o & -B_o & -I & 0 \\
\vdots & \vdots & \vdots & \vdots \\
F_N & 0 & 0 & 0 \\
\end{bmatrix}
\]

(3)

Whilst less generic than interleaving the primal, dual and slack variables corresponding to each stage of the prediction horizon forming a banded matrix [19], to perform dot products of the rows of (3) with appropriately sized vectors with a throughput of one dot product per clock cycle requires a bank of \((3n_u + n_a)\) multipliers in parallel, in contrast to \((2n_u + 4n_s - 1)\) if a band structure were exploited.

The fourth mode triggers the second state machine again with an additional configuration flag enabled, and in combination with a second multiply-accumulate device and a subtractor to calculate \(b_k\) as in step 7, and presents this sequence as a separate output.

\section{B. MINRES solver}

The majority of the computation in the MINRES algorithm comprises the Lanczos process. This is implemented using fixed-point arithmetic, enabled by the preconditioner proposed by [24], [25], which ensures that key variables are in the range \([-1, 1]\). Using the notation \(Z_{k,m}\) to denote the \(m\)th element of the \(n\)th row of matrix \(Z\) at iteration \(k\), by letting

\[
\mathcal{M}_{k,m} = \begin{cases} 
\sum_{n} |A_{k,m,n}|^{1/2} & \text{if } m = n \\
0 & \text{otherwise}
\end{cases}
\]

(4)

instead of directly solving \(A_k c_k = b_k\), MINRES is used to solve \(A_k c_k = (\mathcal{M}_k b_k) / \|\mathcal{M}_k b_k\|_2\). This is implemented as state machines using M-code. The result of the equation is stored as type sFix64_32.

Outside of the Lanczos process, the remainder of the algorithm is implemented using floating point arithmetic since the bounds on the magnitudes are not known \(a \text{ priori}\). This is designed using Xilinx System Generator for DSP to instantiate pipelined IP-cores performing floating point operations and type conversions, with the HDL description of the Lanczos process imported as a “black box”. A single precision (32-bit) floating point data type is used in preference to the more common double precision (64-bit) representation, this decision being motivated by FPGA resource usage.

\section{IV. Resource and Power Usage}

Table I shows the resource usage of the MicroBlaze and custom circuit on the FPGA on the ML605 Evaluation Board. The resources used specifically in the Lanczos process are also shown in italic. An estimate of utilisation for a selection of smaller FPGAs is also presented.

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|}
\hline
Component & FF & LUT & BRAM \\
\hline
Microblaze & 8908 & 9664 & 6 & 41 \\
Custom PCORE & 28873 & 26389 & 199 & 67 \\
(Lanczos) & (9806) & (8198) & (89) & (34) \\
\hline
Total & 37481 & 36223 & 205 & 108 \\
\hline
\end{tabular}
\caption{FPGA resource usage}
\end{table}

(FF: Flip-flop, LUT: Look-up table, BRAM: Block RAM, DSP48E: Hardware multiplier)

Using Xilinx Power Analyzer’s default operating conditions, power consumption of the system is estimated. This shows
that the majority of power consumption is due to “leakage” (i.e. due to quiescent currents) and the custom PCORE. The power consumption of the PCORE and leakage are estimated for a selection of smaller FPGAs, using the Timing and Power Analysis workflow in System Generator for DSP and presented in Table II (Totals in brackets exclude the MicroBlaze). The estimated consumption of the PCORE is consistent over the FPGA models considered, so to gain a further reductions in power consumption, improving the sharing of resources between tasks would be worthwhile.

### Table II

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Total (W)</th>
<th>PCORE (W)</th>
<th>Leakage (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex 6 LX240T</td>
<td>5.247</td>
<td>0.6</td>
<td>3.481</td>
</tr>
<tr>
<td>Virtex 6 LX75TL</td>
<td>(2.176)</td>
<td>0.6</td>
<td>1.081</td>
</tr>
<tr>
<td>Artil XCA100TL</td>
<td>(0.988)</td>
<td>0.6</td>
<td>0.041</td>
</tr>
<tr>
<td>Kintex XC7K70TL</td>
<td>(0.965)</td>
<td>0.6</td>
<td>0.042</td>
</tr>
</tbody>
</table>

V. CLOSED-LOOP SIMULATION

The design is used to control a simulation of the plant dynamics on a PC connected to the FPGA using UDP/IP over 100 Mbit ethernet. The target orbital parameters, the chaser relative position and velocity and parameters configuring the number of algorithm iterations are transmitted to the FPGA, which returns the optimal cost, optimal prediction horizon and the full predicted state and input trajectory.

A. Simulator and scenario

Continuous-time target orbit dynamics and chaser relative dynamics are simulated in Simulink. The target is modelled as being in a Keplerian orbit [22] around Mars, with semimajor axis \( a = 4643 \text{ km} \) and eccentricity \( e = 0.2044 \). The orbital radius at perigee is \( r_p = 3.6939 \times 10^6 \text{ m} \). The open-loop chaser relative dynamics are modelled using the continuous time-varying equations from which the Y-A equations [21] derive. Navigation uncertainty is modelled as zero-mean additive Gaussian white noise on each channel. With \( r \) denoting the Euclidean distance from the target, the standard deviation is 0.35/1000r + 0.0028 (m) for position, and 0.0115/1000r + 0.003 (m/s) for velocity measurements [4]. The impulsive thrust input request from the predictive controller delivered as a 1 s pulse of acceleration, 1 s after the control instruction is received from the FPGA (this delay is unmodelled in the prediction).

In elliptical scenarios, points on the x-axis in the relative reference frame, with zero instantaneous velocity, are not unforced equilibria. The terminal state \( x_T(k + N) \) is chosen as a point on a periodic holding trajectory. Letting \( h_p \) be the average distance the target, \( v_N = v(k + N) \), and \( \rho_N = (1 + e \cos \nu_N) \), then \( x_T(k + N) = h_p \left[ \rho_N, 0, -e \sin \nu_N, -k_2 e \rho_N^2 \sin \nu_N, 0, k_2 (\rho_N^2 - \rho_N^3) \right]^T \), where \( k_2 = \mu^2 \left( r_p \sqrt{2\mu (r_p^{-1} - 0.5a^{-1})} \right)^3 \), and \( \mu \) is the gravitational parameter.

The scenario demonstrated starts at a position of \( x(0) = [10000 \text{ m}, 0 \text{ m}, 0 \text{ m}, 0 \text{ m/s}, 0 \text{ m/s}, 0 \text{ m/s}]^T \) and the hold points are scheduled to be centred at 5000 m, 2000 m, 1000 m, 500 m, 200 m and 100 m in front of the target. The chaser is released from a holding point when \( N_{\text{opt}} \leq 2 \) for at least 5 sampling instants. Based on [4], the sampling period is chosen to be \( T_s = 300 \text{ s} \), and the weighting parameter \( \gamma = 35 \). The impulsive control input bound \( \Delta u = 1 \text{ m/s} \).

B. Control results

Table III shows metrics for the quality of the control performance, with respect to variation in the number of interior point iterations \( I_{IP} \), and the number of MINRES iterations, \( I_{MR} = \lceil \eta / N \rceil \). These include the completion time of the whole scenario, the solution time for the full VH-LTV-MPC problem at each sampling instant, the cumulative impulsive velocity change \( \Delta v \) applied, and the maximum and mean 2-norm of the difference between the control action from the FPGA-based strategy and an “ideal” solution solved on the PC using GLPK [26]. (For context, the MicroBlaze runs at 100 MHz, and the peripheral MINRES accelerator at 200 MHz). Figure 3 shows an example closed loop trajectory with \( I_{IP} = 22 \) and \( \eta = 1.1 \).

![Fig. 3. Closed loop trajectory with FPGA in the loop](image)

C. Computational load (for one fixed horizon)

The apportionment of computation time for a fixed horizon \( N = 20 \) is presented in Table IV obtained empirically using a 100 MHz Timer Counter peripheral connected to the MicroBlaze.

### Table IV

<table>
<thead>
<tr>
<th>Operation</th>
<th>Runs</th>
<th>Ticks</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction model</td>
<td>1</td>
<td>16829</td>
<td>16829</td>
</tr>
<tr>
<td>Prediction model transfer</td>
<td>1</td>
<td>6867</td>
<td>6867</td>
</tr>
<tr>
<td>Memory initialization</td>
<td>1</td>
<td>3594</td>
<td>3594</td>
</tr>
<tr>
<td>Vector transfer</td>
<td>IP</td>
<td>31403</td>
<td>690866</td>
</tr>
<tr>
<td>Microblaze computation</td>
<td>IP</td>
<td>86676</td>
<td>1906872</td>
</tr>
<tr>
<td>PCORE computation</td>
<td>IP</td>
<td>224464</td>
<td>4938208</td>
</tr>
</tbody>
</table>

| Total                     | 7563236 | 75.6  |

In comparison, for a fixed horizon \( N = 20 \), an LP solver generated using CVXGEN [27], configured with prediction matrices starting from \( v = \pi / 2 \) with \( x_0 \) on the x axis at 10000 m, and scheduled holding point at 5000 m takes 314 ms to run 23 iterations directly on the MicroBlaze using single precision arithmetic. The FPGA-based solver offers a 4× speedup (or 2.5× cycle-per-cycle due to the different clock rates). The FPGA-based solver can also handle dense quadratic stage and input costs with no modifications. A QP solver for such costs from CVXGEN takes 1.79 s with single precision arithmetic (14× more cycle-per-cycle).
VI. CONCLUSIONS AND FUTURE DEVELOPMENTS

This paper has presented the design and evaluation of a predictive controller for a phase of spacecraft rendezvous on an FPGA using a mixed hardware-software approach, with the custom hardware component accelerating the critical path of the algorithm. The controller supports a LTV prediction model and a variable horizon, and the custom hardware components are implemented using Simulink HDL Coder and Xilinx System Generator for DSP. Further work will investigate the application to additional rendezvous phases, and improving the efficiency of the custom hardware component.

REFERENCES


