A Software Configurable Coprocessor-Based State-Space Controller

Aaron Mills, Pei Zhang, Sudhanshu Vyas, Joseph Zambreno, Phillip H. Jones
Iowa State University, Ames, Iowa, USA
{ajmills,peizhang,spvyas,zambreno,phjones}@iastate.edu

***Abstract*—We present a software configurable coprocessor-based state-space controller that can control physical processes representable by a linear state-space model. Our proposed architecture has distinct advantages over purely software or purely hardware approaches. It differs from other hardware controllers in that it is not hardwired to control one or a small range of plant types (e.g. only electric motors). Via software, an embedded systems engineer can easily reconfigure the controller to suit a wide range of controls applications that can be represented as a state-space linear model. Additionally, we introduce a novel design methodology to help bridge the gap between controls and embedded system engineering. Control of the well-understood inverted pendulum on a cart is used as an illustrative example of how the proposed hardware accelerator architecture supports our envisioned design methodology for helping bridge the gap between controls and embedded software engineering.**

I. Introduction

Field-programmable gate arrays (FPGAs) are of growing interest in the area of applied control theory [12]. In addition to the massive parallelism available on FPGAs that can potentially be utilized to obtain high controller update rates, software-hardware co-design using FPGAs can help separate embedded software concerns (e.g. real-time scheduling feasibility), from controls concerns (e.g. accounting for update-rate jitter).

Implementing an efficient controller on an FPGA can be challenging for engineers unfamiliar with hardware architecture design. One solution is software programmable hardware. In our work, we describe a software-configurable FPGA co-processor architecture that can implement a wide range of linear state-space controllers, up to the complexity of a Linear Quadratic Regulator (LQR) coupled with a Luenberger Observer.

For the purpose of evaluation, the controller can be interfaced to a hardware-based emulation of a physical plant (i.e. Plant on Chip) [18]. This arrangement is depicted in Fig. 1. The Plant on Chip (PoC) allows for rapid, and consistent testing of control algorithms and system platform configurations. Once stability of the emulated plant is achieved, it can be replaced with an interface to the actual plants sensors and/or actuators. All control computations are done in hardware, while software running on the CPU is used to initialize the co-processor. The software is also free to perform other tasks, for example, task scheduling, path planning, video processing, and interactive communications.

The remainder of this paper is organized as follows. In Section II, we discuss and compare related works in this problem space. In Section III, we describe the detailed design of our software configurable co-processor based state-space controller, and provide a brief overview of our plant on chip architecture. In Section IV, we present an illustrative example of using our co-processor to evaluate the use of hardware verses software for an embedded controls application, and explore the performance and scaling of our coprocessor-based controller. Section V concludes this paper and provides avenues of future work.

II. Related Work

Kozak, in [10], surveys trends in the field of applied controls, in which we see controls have evolved from manually-tuned single-input single-output (SISO) controllers to multiple-input multiple output (MIMO) H_∞ controllers and model-predictive controllers (MPC). The latter types of algorithms are computationally intense and can introduce significant latencies when implemented with off-the-shelf processing platforms. Kozak additionally suggests that a software-hardware co-design approach for implementing advanced controllers (e.g. H_∞) in FPGAs would enable designers to make better use of these complex controllers in high-speed systems. Monmasson, in [13], makes a similar suggestion, pointing out how different parts of a control algorithm are better suited for different types of hardware. However, locating the optimal software-hardware partition is still a challenge.

There are numerous examples of application-specific FPGA-based controllers in the literature. An example of a system requiring very fast control update rates appears in [15], in which a high-speed pan/tilt camera is designed to track objects. In order to reach the 3.5ms update rate, a dedicated PC is used to perform image processing and produce motor control signals. It is noted that the PC introduced considerable delay in the feedback loop. Another application requiring very high update rates appears in [16], which presents an application-specific design that used machine vision to control an inverted pendulum. In [7], the authors developed a self-tuning state-space controller using a multiply-accumulate unit which is interfaced with a digital signal processor (DSP). This paper demonstrated the use of FPGAs to control a plant with non-constant plant parameters. In [2], the design of a high-speed, hardware-only, fixed-point MPC is discussed. Finally, in [9], an MPC is implemented on an FPGA and is shown to allow for a significantly faster sample rates than a PC running at a higher clock frequency.

Compared to software, implementing high-performance control algorithms in hardware is relatively rather time consuming and leads to application-specific solutions. A proposed solution to this issue appears in [17] and [3], which use a co-processor to perform low-level repetitive matrix operations for MPC. This allows control designers to use software for the high-level logic; however, to do so they must work with a custom floating-point format and instruction set.

A general summary of approaches used to implement controllers on FPGAs appears in [14]. In [14] a call is made for designs that make efficient use of the massive parallelism available on FPGAs, while retaining the generality and flexibility available to software solutions. Our work pursues this goal. In summary, a number of works exist describing controllers that achieve reduced computational delay. However, these controllers are designed to solve specific problems, unlike our fully software configurable solution. Garbergs, in [5], [6], presents the closest vision and architecture to our approach. The main differences being: 1) their design does not take into account scaling to different sized controllers (e.g. if controller coefficients change the design must be re-implemented), 2) their design is intended to be standalone, as compared to being a memory-mapped co-processor, and 3) their vision focuses more on developing a fast hardware controller as opposed to supporting a design methodology that bridges the gap between embedded software developers and controls engineers.

III. System Architecture

The targeted hardware platform is the Xilinx Zynq 7000 series system-on-a-chip (SoC). The Zynq SoC consists of an ARM Cortex A9 processor coupled with an FPGA. The Xilinx toolchain for the Zynq directly supports hardware-software co-design, making the platform ideal for developing co-processor based applications. FPGA components are written in VHDL and software components are written in C.

The AXI bus is a 32-bit wide standardized interface which allows a co-processor to communicate with the CPU, or allows independent communication among co-processors. As shown in Fig. 1, the PoC and the controller both have slave interfaces, which allow their internal memory spaces to appear to the CPU as memory-mapped peripherals. Meanwhile, the AXI Bus master interface on the LQR controller allows it to sample the output of the PoC while the system is running.

Besides the model coefficients, both controller and PoC must be configured with three constants which represent the size of the state-space model, and allow boundaries to be computed for internal memory fetches. The controller also is configured with a particular sample rate, which represents the number of clocks it will wait before sampling the PoC output memory.

1) m: the number of plant model inputs.
2) n: the number of plant model states.
3) p: the number of plant model outputs.

Dot-products between the coefficient rows and variable vectors are performed in a straightforward manner using a pipelined multiply-accumulate unit, as shown in Fig. 2. These operations are performed as single-precision floating point to avoid the limitation on precision and tedious preprocessing work associated with fixed-point math, as well as to increase the ease with which the co-processor integrates with software. The resource usage of the system is summarized in Table I; in total 5% of available Slices are consumed.

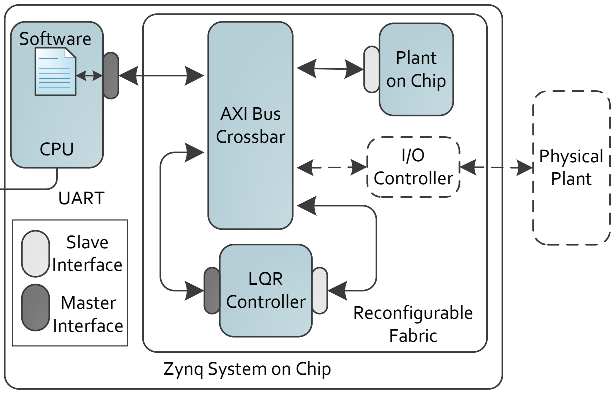


Fig. 1. System Overview. Both the controller and Plant on Chip (PoC) are fully software configurable over the shared AXI bus. A software or hardware controller can control the PoC without requiring hardware reconfiguration.

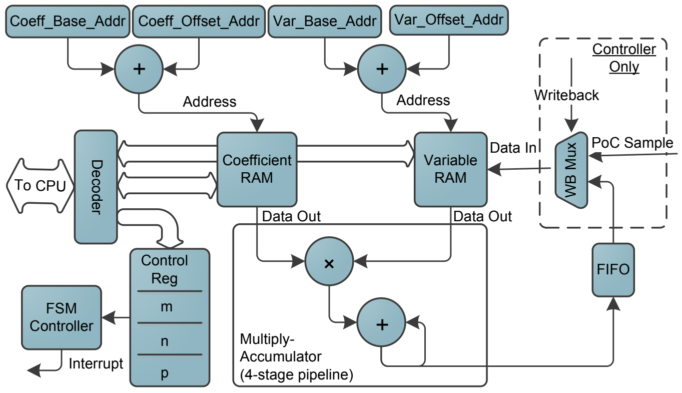


Fig. 2. Architecture datapath. The dot-product result for each row is stored in the FIFO. After all rows are processed, the FIFO writes results back in the same cycle as they are read back out for the next update operation. An interrupt signal can be used to notify the CPU when values are updated.

TABLE I
System Resource Usage on Zynq XC7Z020

	Slices	LUTs	BRAM/FIFO	DSP48E1
PoC	443	1376	4	7
Controller	447	1378	5	3
Total	**890**	**2754**	**9**	**10**

A. General Linearized Model of Plant

Both the PoC and controller computations are centered on a standard linear state-space model of a physical plant. This generic system model consists of matrices A,B, and C[1] and is formulated as follows:

[1]We omit matrix D which is rarely required.

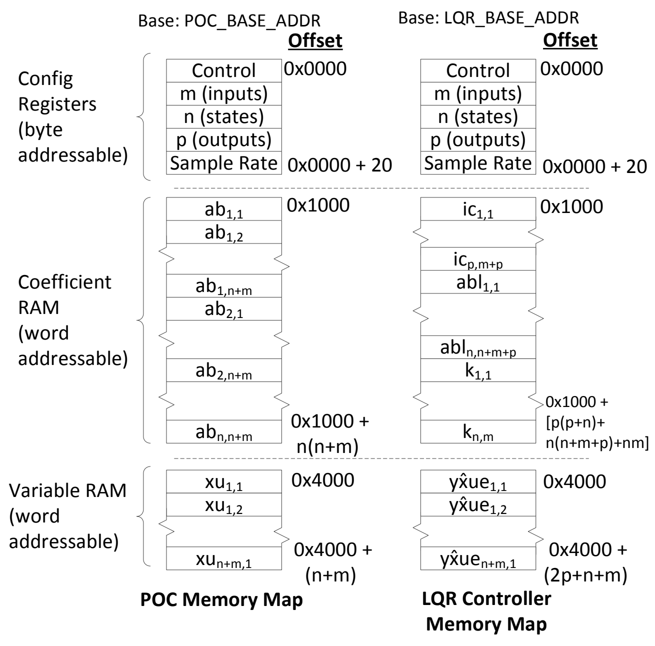


Fig. 3. Coefficients and variables occupy independent, packed memory spaces, with matrices laid out in row-major order.

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

$$y_k = Cx_k \tag{2}$$

These equations allow one to compute the next system state x_{k+1} based on the current state x_k, given a particular input u_k. Additionally the output y_k is computed at each time step k based on the current state and input value.

B. Co-processor Memory Space

The controller and PoC each have two separate dual-ported memories to facilitate parallel data access: one for coefficients (A, B, C, L, K), and one for variables (x, y, u). The memory map as seen by the CPU is shown in Fig. 3. Each memory is dual ported with independent outputs. One port is dedicated to interfacing with the CPU for memory initialization and read back, and the other port dedicated to internal operations.

As a simple optimization, matrices which are involved in the same dot-product are concatenated in memory where it is possible. This prevents two extra, unnecessary load-store cycles, and is explained in greater detail in subsequent sections. All data is loaded by the CPU into a contiguous block at the coefficient or variable memory base address, with only the values of m, n and p being needed to compute arbitrary matrix boundaries. Compared to providing a fixed address for each matrix, this scheme maximizes memory efficiency and plant model flexibility, since there is no real architectural constraint on m, n or p other than the overall memory size of the target FPGA.

C. Plant on Chip Algorithm

The A and B matrix, and the x and u vectors, are concatenated in order to allow more efficient pipelining of multiply-accumulate operations. All computations are performed in single precision floating point format. The PoC computes the equations in Equations 3 and 4. In most applications the D matrix is not needed and therefore excluded.

$$\begin{bmatrix} x_{1|k+1} \\ \vdots \\ x_{n|k+1} \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} & b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} & b_{n,1} & \cdots & b_{n,m} \end{bmatrix} \times \begin{bmatrix} x_{1|k} \\ \vdots \\ x_{n|k} \\ u_{1|k} \\ \vdots \\ u_{m|k} \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{p,1} & \cdots & c_{p,n} \end{bmatrix} \times \begin{bmatrix} x_{1|k} \\ \vdots \\ x_{n|k} \end{bmatrix} \tag{4}$$

D. LQR Controller Algorithm

An LQR controller is implemented to create a complete closed-loop plant control system. An LQR controller is based around the gain matrix K. A particular K is sought such that the feedback law $u_k = -Kx_k$ minimizes the quadratic cost function in Eq. 5 [4].

$$J(u) = \sum_{1}^{\infty} x_k^T Q x_k + u_k^T R u_k \tag{5}$$

Generating K requires the controls engineer to select state-cost matrix Q and performance index matrix R which work well for a given plant.

Although the complete value of state vector x can be read from the PoC at any time, many plant models include internal states which cannot be directly measured. Therefore, to increase its flexibility our controller also integrates a Luenberger-type observer model. Here L denotes the gain matrix of the observer, and y is the measurable states sampled from the plant.

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + L(y - C\hat{x}_k) \tag{6}$$

This addition allows the controller to estimate the value of the unmeasured plant states as $\hat{x}$; if all states are observable then one merely configures n equal to p for the co-processor. It is this estimated state vector which is used to generate the control vector u. This process is shown in Fig. 4.

The control update operation is split into three phases, shown in Equations 7-9. Matrix $I_{p\times p}$ is the identity matrix, and matrices K and C are negated before loading into memory to eliminate the need for subtraction. Matrix sizes in terms of the user-configured constants m, n and p are included as subscripts.

$$E_{p\times n} = \begin{bmatrix} I_{p\times p} & -C_{p\times n} \end{bmatrix} \times \begin{bmatrix} y_{p\times 1} \\ \hat{x}_{n\times 1|k} \end{bmatrix} \tag{7}$$

$$\hat{x}_{n\times 1|k+1} = \begin{bmatrix} A_{n\times n} & B_{n\times m} & L_{n\times p} \end{bmatrix} \times \begin{bmatrix} \hat{x}_{n\times 1|k} \\ u_{m\times 1} \\ E_{p\times 1} \end{bmatrix} \tag{8}$$

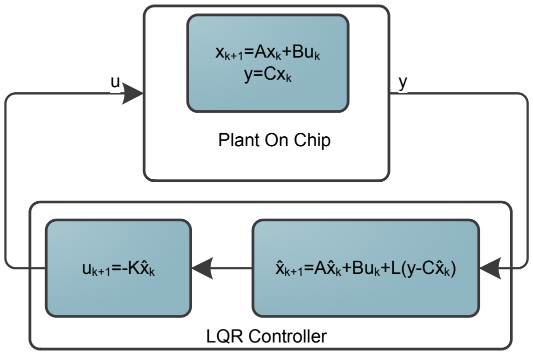


Fig. 4. LQR controller with observer controlling an emulated plant (PoC).

$$u_{m\times 1} = \left[-K_{m\times n}\right] \times \left[\hat{x}_{n\times 1|k+1}\right] \tag{9}$$

Additionally, the addresses applied to the Coefficient RAM and Variable RAM are split into base and offset addresses in order to increase flexibility. The pseudocode in Algorithm 1 demonstrates the computation of memory addresses for the dot product operation. Note that address calculation arithmetic is comprised of only small integers, and is only computed once during initialization.

IV. Experimentation

The performance of the co-processor is tested in this section. For the experimental setup, the Zynq's ARM processor and FPGA fabric are both clocked at 50Mhz, and memory caching is disabled. We configure the ARM processor to 50 Mhz to represent a lower-powered embedded processor, and disable cache to emulate safety critical systems that require highly deterministic timing. Traditionally, both *delay* and *jitter* are considered as critical parameters for determining the stability of a controller [1]. In this paper, we only consider delay, since the software controller is single-threaded, and the hardware controller is naturally jitter-free. In particular we are concerned with the effect that controller update delay has on plant stability. The experiment performed in this section presents a test plant requiring a sample rate of 2ms to maintain stability. As shown, the hardware controller shows a clear advantage over the software controller, as the former can maintain plant stability for large state-space models, whereas the latter cannot due to computational delay.

A. Test Plant

The plant used during testing is the classic inverted pendulum, a non-linear model illustrated in Fig. 5. The model parameters are shown in Table II. Meanwhile, the state vector consists of four states: x, $\dot{x}$,ϕ, and $\dot{\phi}$. This model requires the CPU set co-processor parameters $n = 4$, $m = 1$, and $p = 2$.

A partial derivation of the system follows. The two nonlinear equations which describe the physics of the pendulum [8], [11] are shown below:

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = u \tag{10}$$

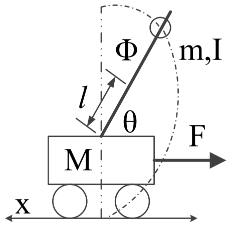


Fig. 5. Inverted Pendulum Model

$$(I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta \tag{11}$$

We linearize the equations around the upright equilibrium position of the pendulum ($\theta = \pi$), assuming that the system will maintain a small deviation from this position. We introduce ϕ as this deviation (that is, $\theta = \phi + \pi$). We use small angle approximations of the trigonometric functions in the system equations to obtain a linearized version.

$$(M+m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u \tag{12}$$

$$(I+ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x} \tag{13}$$

Finally we rearrange the expressions as a set of first-order differential equations so they can be put into state-space form.

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ \phi_{k+1} \\ \dot{\phi}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ \phi_k \\ \dot{\phi}_k \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u \tag{14}$$

Note that the output expression in Equation 15 is configured to reflect the fact that only position x and angle ϕ are directly observable.

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ \phi_k \\ \dot{\phi}_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \tag{15}$$

At this point, Matlab is used for discretization and to solve the LQR minimization problem, thereby providing LQR gain matrix K.

B. Performance Analysis

The effect of controller computational delay on plant behavior is tested by setting the pendulum vertical displacement angle ϕ to $-5°$ so that it is initially unstable, and then increasing controller workload over repeated trials. Extra zeroed 'dummy' states are added to the 4 base states in order to

Algorithm 1 LQR with Observer using variable m, n, and p

```
 1: procedure INITIALIZE
 2:     $v_{base} \leftarrow \{0, p, p\}$                # Compute memory boundaries (implemented as mux).
 3:     $c_{base} \leftarrow \{0, 2p(p+n), (2p+2n+m)(2p+3n)\}$
 4:     $v_{max} \leftarrow \{(p+n)-1, (n+m+p)-1, n-1\}$
 5:     $c_{max} \leftarrow \{2p(p+n)-1, (n+m+p)(3n)-1, mn-1\}$
 6: procedure UPDATE
 7:     for $j \leftarrow 0$ ... 2 do                # For each computation phase...
 8:         $sum \leftarrow 0$
 9:         for $c_i \leftarrow 0$ ... $c_{max}[j]$ do                # For each coefficient...
10:             $sum \leftarrow sum +$ COEF_MEM$[c_{base}[j] + c_i] \times$ VAR_MEM$[v_{base}[j] + v_i]$
11:         if $v_i = v_{max}[j]$ then
12:             WBFIFO $\leftarrow sum$                # Save dot product for this row into writeback FIFO.
13:             $v_i \leftarrow 0$
14:             $sum \leftarrow 0$                # Reset accumulator.
15:         else
16:             $v_i \leftarrow v_i + 1$
```

TABLE II
INVERTED PENDULUM MODEL SYMBOLS

Symbol	Meaning	Initialization
M	cart mass	2.725kg
m	pendulum mass	1.09kg
b	coefficient of friction	0.1 N/m/sec
l	length to pendulum center of mass	0.2 m
I	pendulum moment of inertia	$0.006 kg \cdot m_2$
u	applied force	0
x	position displacement	0
θ	angle from downward vertical axis	N/A
ϕ	angle from upward vertical axis	$-5°$

TABLE III
EXECUTION TIME COMPARISON (μS)

States	Software	Hardware
4	183	9.9
8	428	20.66
16	1254	48.64
32	4071	127.14

modulate the controller delay without impacting the model itself.

Fig. 6 shows the response of the inverted pendulum plant as the number of control states is increased. At 16 states, the response shows decaying oscillations, and beyond 25 states the plant becomes unstable. However, Fig. 7 shows that increasing the computational workload has very little observable effect on the stability of the plant. Table III compares the software execution time to the hardware execution time as the number of states is increased.

It is also possible to estimate the maximum number of states supported by this architecture. Based on the memory map, the total memory required for both the PoC and controller can be computed as follows:

$$f(m,n,p) = 2n^2 + p^2 + 3nm + 2pn + 2(p+n+m) \quad (16)$$

Given that total amount of block RAM on the target device is 560kB, we consider two cases: one actuator and many actuators. Letting $n = p$, which is worst-case memory wise, if $m = 1$ then the maximum number of states is approximately 166. If we constrain $m = n = p$ then the maximum number of states is approximately 131.

V. CONCLUSION

We have presented a novel approach to designing a hardware-based co-processor for control applications, and have illustrated how this approach could be used to ease the transition from control theory to embedded control implementation. Avenues of future work include: 1) increase the modest parallelism exploited by our current co-processor implementation and preform more aggressive pipelining, 2) describe a more formalized procedure for moving controller designs from theory to implementation, and 3) explore the support of higher complexity controllers, such as, H_∞. Algorithm steps which involve irregular data or computation, and are therefore usually pre-computed offline, could be transfered to the CPU. Meanwhile, steps which involve regular data or computation are ideal for hardware accelleration.

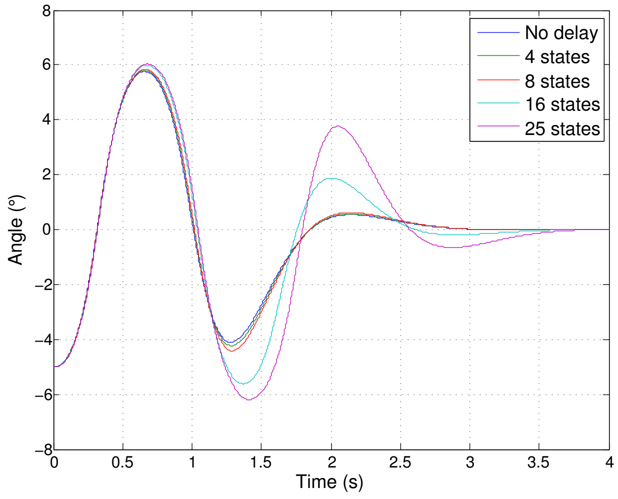
REFERENCES

[1] A. Aminifar, E. Bini, P. Eles, and Z. Peng. Bandwidth-efficient controller-server co-design with stability guarantees. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, March 2014.

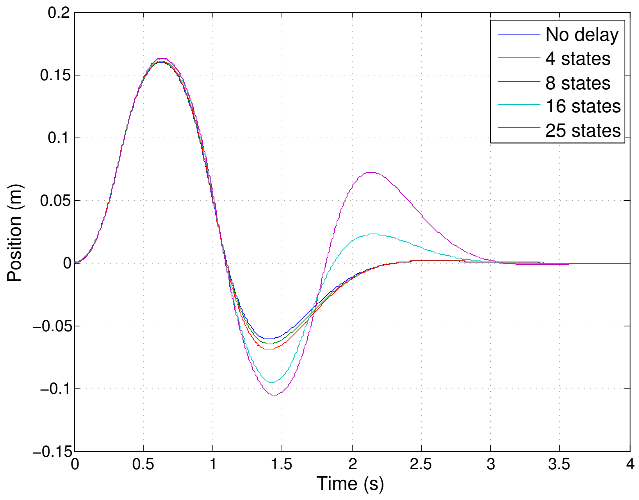
[2] K. Basterretxea and K. Benkrid. Embedded high-speed model predictive controller on a FPGA. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011.

[3] L. Bleris, P. Vouzis, M. Arnold, and M. Kothare. A co-processor FPGA platform for the implementation of real-time model predictive control. In *American Control Conference*, June 2006.

[4] T. Chen, B. Francis, and T. Hagiwara. Optimal sampled-data control systems. *Proceedings of the IEEE*, 86(4):741–741, 1998.

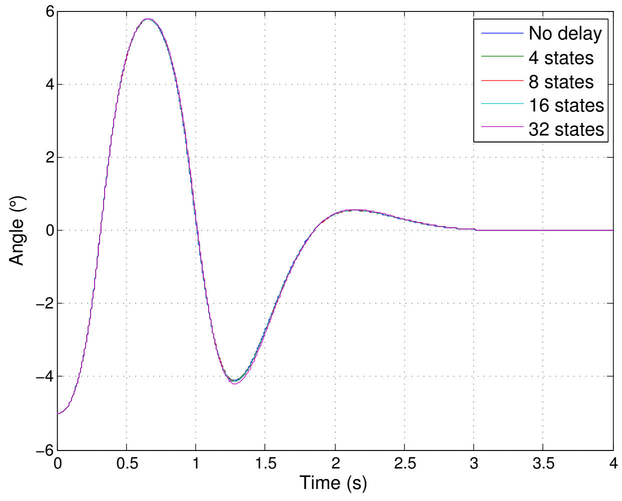


(a) Pendulum angle response

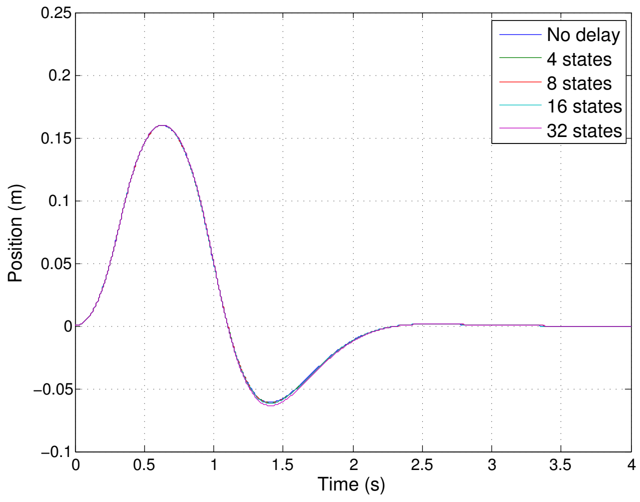


(b) Pendulum position response

Fig. 6. Impact of computation delay on plant stability (software controller)



(a) Pendulum angle response



(b) Pendulum position response

Fig. 7. Impact of computation delay on plant stability (hardware controller): the plant responses are nearly indistinguishable with increasing delay.

[5] B. Garbergs and B. Sohlberg. Specialised hardware for state space control of a dynamic process. In *TENCON '96. Proceedings., 1996 IEEE TENCON. Digital Signal Processing Applications*, volume 2, pages 895–899 vol.2, Nov 1996.

[6] B. Garbergs and B. Sohlberg. Implementation of a state space controller in a fpga. In *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean*, volume 1, pages 566–569 vol.1, May 1998.

[7] D. A. Gwaltney, K. D. King, K. J. Smith, and J. Montenegro. Implementation of Adaptive Digital Controllers on Programmable Logic Devices. *Military and Aerospace Programmable Logic Devices (MAPLD)*, Sept 2002.

[8] C. Hu and F. Wan. Parameter identification of a model with Coulomb friction for a real Inverted Pendulum System. In *Control and Decision Conference (CCDC)*, pages 2869–2874, June 2009.

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

[10] S. Kozak. Advanced control engineering methods in modern technological applications. In *Carpathian Control Conference (ICCC)*, pages 392–397, May 2012.

[11] B. Messner and D. Tilbury. Inverted Pendulum: System Modeling. http://ctms.engin.umich.edu/CTMS, 2012.

[12] E. Monmasson and M. Cirstea. Guest editorial special section on industrial control applications of fpgas. *Industrial Informatics, IEEE Transactions on*, 9(3):1250–1252, Aug 2013.

[13] E. Monmasson, L. Idkhajine, and M. W. Naouar. FPGA-based Controllers. *Industrial Electronics Magazine, IEEE*, 5(1):14–26, March 2011.

[14] B. Mutlu and M. Dolen. Implementations of state-space controllers using Field Programmable Gate Arrays. In *International Symposium on Power Electronics Electrical Drives Automation and Motion (SPEEDAM)*, pages 1436–1441, June 2010.

[15] K. Okumura, H. Oku, and M. Ishikawa. High-Speed Gaze Controller for Millisecond-Order Pan/Tilt Camera. In *IEEE International Conference on Robotics and Automation*, pages 6186–6191, May 2011.

[16] Y. Tu and M. Ho. Design and implementation of robust visual servoing control of an inverted pendulum with an FPGA-based image co-processor. *Mechatronics*, 21(7):1170 – 1182, 2011.

[17] P. Vouzis, M. Kothare, L. Bleris, and M. Arnold. A System-on-a-Chip Implementation for Embedded Real-Time Model Predictive Control. *IEEE Transactions on Control Systems Technology*, 17(5):1006–1017, Sept 2009.

[18] S. Vyas, N. Chetan Kumar, J. Zambreno, C. Gill, R. Cytron, and P. Jones. An FPGA-Based Plant-on-Chip Platform for Cyber-Physical System Analysis. *IEEE Embedded Systems Letters*, 6(1):4–7, March 2014.