2016

Design and implementation of an FPGA-based piecewise affine Kalman Filter for Cyber-Physical Systems

Aaron Mills
Iowa State University

Design and implementation of an FPGA-based piecewise affine Kalman Filter for Cyber-Physical Systems

by

Aaron Joseph Mills

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Joseph Zambreno, Major Professor

Phillip Jones

Diane Rover

Arun Somani

Elia Nicola

Iowa State University

Ames, Iowa

2016

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Kalman Filter is a robust tool often employed as a process observer in Cyber-Physical Systems. However, in the general case the high computational cost, especially for large plant models or fast sample rates, makes it an impractical choice for typical low-power microcontrollers. Furthermore, although industry trends towards tighter integration are supported by powerful high-end System-on-Chip software processors, this consolidation complicates the ability for a controls engineer to verify correct behavior of the system under all conditions, which is important in safety-critical systems and systems demanding a high degree of reliability.

Dedicated Field-Programmable Gate Array (FPGA) hardware can provide application speedup, design partitioning in mixed-criticality systems, and fully deterministic timing, which helps ensure a control system behaves identically to offline simulations. This dissertation presents a new design methodology which can be leveraged to yield such benefits. Although this dissertation focuses on the Kalman Filter, the method is general enough to be extended to other compute-intensive algorithms which rely on state-space modeling.

For the first part, the core idea is that decomposing the Kalman Filter algorithm from a strictly linear perspective leads to a more generalized architecture with increased performance compared to approaches which focus on nonlinear filters (e.g. Extended Kalman Filter). Our contribution is a broadly-applicable hardware-software architecture for a linear Kalman Filter whose operating domain is extended through online model swapping. A supporting application-agnostic performance and resource analysis is provided.

For the second part, we identify limitations of the mixed hardware-software method and demonstrate how to leverage hardware-based region identification in order to develop a strictly hardware-only Kalman Filter which maintains a large operating domain. The resulting hardware processor is partitioned from low criticality software tasks running on a supervising software processor and enables vastly simplified timing validation.

CHAPTER 1. INTRODUCTION

The Kalman Filter is a robust algorithm often employed as a means to estimate the state of a physical process in a wide array of sensing and control tasks in Cyber-Physical Systems, ranging from radar[1] to battery monitoring for planetary rovers [2] to virtual reality [3]. Cyber-Physical System (CPS) have been described as the integration of computing with physical processes, where correct behavior requires correct timing [4]. Lee, in his position paper [5], describes the limitations of current design approaches and architectures which support CPS, which includes how the many advances in software processors (deep pipelines, speculation, etc.) makes software timing prediction difficult or even impossible. The author makes a call for research on new ways of approaching CPS compute architecture, including mixed hardware-software design methods, which better support the unique constraints of the field.

Taking a broad view, CPS usually involve process control functionality for which the Kalman Filter plays only one part. In control theory, a discrete-time control process consists of 3 tasks: input, computation, and output. The input task typically involves an analog to digital conversion. For model-driven controllers, the computation task typically of consists state estimation (e.g. through a Kalman Filter) followed by control value computation. Lastly, the output task involves converting the control value into a physical signal (ie digital to analog conversion). From a theoretic standpoint, the standard assumption is that the input occurs at a constant interval, and the computation and output process take zero time.

However, in practice, nonfunctional system engineering concerns (e.g. desire for low cost, low power, small size) coupled with the need for a system to perform potentially many functions with different timing requirements makes it challenging to even approximate the standard control theoretic assumptions in a validatable fashion. These issues are discussed further in the following section.

1.1 Motivation

In the general case the high computational cost, especially for large system models (e.g. consisting of many states) or fast sample rates, makes the Kalman Filter an impractical choice for typical low-power 8-bit or 16-bit microcontrollers. The work in [6] estimated that applying a particular variant to the Kalman Filter–the UD Filter–to a 21-state model for an Inertial Measurement Unit (IMU) with a 10ms update rate, would require an estimated 2.8MFLOPs. Of course, this estimate does not include other instructions such as branching and memory access, as well as kernel overhead and time spent tending to other tasks sharing the same processor. Fixed-point math reduces the runtime complexity, allowing the filter to be run on microcontrollers lacking floating-point support, but imposes an additional design hurdle on the controls engineer, since numerical precision must be accounted for as well.

Other examples of high-speed Kalman Filtering exist in literature. An even more complex 36-state IMU model appears in [7]. The high-speed running robot in [8] has a highly-complex electromechanical model using 16 states at 1kHz sampling. Applications involving the monitoring or simulation of switching circuits rank among those with the shortest update deadline, such as a proposed Kalman Filter-based fault detection system for DC-DC power converters in Hybrid Electric Vehicles [9], which requires sample rates greater than 10kHz with low jitter. Simultaneous location and mapping (SLAM) for mobile robotics is a common application for the Kalman Filter which normally must be computed on a PC attached to the robot [10] due to the large size of the covariance matrix, which may have millions of entries [11].

However, most literature on Kalman Filter acceleration tends to ignore systems engineering level issues, which may be fundamentally more challenging. Industry trends towards tighter integration and subsystem consolidation are supported by powerful 32-bit System-on-Chip (SoC) processors, giving rise to "mixed-criticality systems" in which high criticality tasks are time-multiplexed on the same compute resource as low criticality tasks. Compared to the "air gap" model of separate processing units, this approach complicates the ability for a controls engineer to verify correct behavior of the critical tasks, whose timing may be negatively impacted by the inherently non-deterministic timing of a complex software processor. Thus the primary concern of mixed-criticality systems is interference between tasks [12]. Many such applications may be subject to strict certification requirements such as DO-178B for aerospace applications or ISO 26262 for road vehicles.

Several ongoing research areas exist to help address this issue, each with strengths and weaknesses. Formal verification methods, which seek to validate a system on an analytical or mathematical basis, have made great advancements in the past decade, but still have many barriers to integration into existing design workflows[13] and tend to lack consideration for interactions with the environment [14]. High-confidence Worst-Case Execution Time (WCET) analysis requires a detailed hardware-timing model, which is very hard to obtain for processors designed primarily for completing a high number of instructions per clock [15]. Carefully designed scheduling methods, which seek to impose additional determinism on task runtime characteristics, often impose additional runtime overhead, and still result in a stochastic computation model [16, 17]. Architecture extensions to software processor architectures are also gaining interest, but at best reduce–not eliminate–software execution time uncertainty [18]. In all cases, the best-achievable runtime behavior is still stochastic in nature.

Finally, for mobile applications or applications operating on battery power there is clear motivation for minimizing total system power consumption: if one can halve the power consumed, battery runtime is doubled. The hardware-based Kalman Filter accelerator in [19] is targeted at mobile robotics and seeks to balance computational power and power consumption. Interest in SLAM for mobile robotics–a very computationally intensive task–is also driving computer architecture research. The work in [11] demonstrated a hardware architecture which can process 1.5K features, consisting of 3003 states and 34MB data, at an 14Hz update rate–not only faster, but also at substantially lower power consumption compared to a reference Pentium M processor.

Field-programmable gate arrays (FPGAs) are of growing interest in the area of applied control theory [20]. 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). This work is primarily interested in using dedicated FPGA hardware to provide computational speedup, firm design partitioning in mixed-criticality systems, and also fully deterministic timing, which helps ensure a control system behaves as close as possible to offline simulations. In addition, at the system engineering level a critical advantage for FPGAs in CPS is the ability to reduce board space and design cost by incorporating many different digital peripherals (e.g. audio/video codecs, signal multiplexers, DSPs, custom I/O drivers) into the same package without sacrificing performance.

1.2 Dissertation

The overall dissertation has two major components: developing a mixed hardware-software hardware-accelerated Kalman Filter for CPS, and extending that design to enable a fully hardware-based compute platform for more effective architectural partitioning.

For the first part, the core idea is that of approaching the design process from the perspective of a linear Kalman Filter rather than the Extended Kalman Filter leads to a more generalizable architecture with increased performance. Coupled with a piecewise-affine modeling methodology, operating domain can be expanded beyond the basic linear Kalman Filter. Our contribution is (1) a novel mixed hardware-software architecture for a linear Kalman Filter which we describe as the hardware-software Piecewise-Affine Kalman Filter. This approach achieves a speedup over existing methods by reducing computational complexity of software and communication overhead between the processor and the hardware. This work is supported by performance and resource analyses which approaches the problem from a novel, application-agnostic perspective.

For the second part, the core idea is that adding architecture support for hardware-based region identification leads to a fully hardware-based Piecewise-Affine Kalman Filter which achieves timing-determinism and full partitioning from software tasks. Our contribution is (2) an approach that integrates hardware architecture and a hyperrectangular model partitioning scheme to eliminate the non-accelerated, application-specific software code stub that exists in all mixed hardware-software Kalman Filter accelerators to date. This leads to a slight speedup over the mixed hardware-software Piecewise Affine Kalman Filter in part 1, but more importantly unlocks the capability for fully time-deterministic control loops. Supporting analysis to help determine the overall memory requirements of such an approach is also included.

An additional contribution is (3) a set of case studies which illustrate the model transformation process and overall design workflow for the tools proposed in this dissertation.

1.3 Organization of the Report

The remainder of this report is organized as follows. Chapter 2 provides background research motivating the use of dedicated FPGA architectures for use in real-time systems. Chapter 3 describes existing works in the space of mixed hardware-software Kalman Filtering, the architecture for our Piecewise-Affine Kalman Filter, and a detailed characterization of the design. Chapter 4 extends the architecture of the Piecewise-Affine Kalman Filter, enabling a completely hardware-oriented design which is still supervised by a traditional software processor. Chapter 5 delves into several specific case studies to illustrate the design process as it pertains to the Piecewise-Affine Kalman Filter. Finally, Chapter 6 provides a summative discussion and future avenues for research.

CHAPTER 2. HARDWARE/SOFTWARE FOUNDATIONS

2.1 Introduction

This chapter lays the conceptual groundwork for our work on FPGA-based Kalman Filtering. It describes the foundational motivation for considering an application-agnostic design methodology, the advantages of a mixed hardware-software architecture, and an illustration of the timing-determinism which dedicated hardware can bring to timing-sensitive applications. We present our implementation of an Linear Quadratic Regulator (LQR) originally proposed in [21], and more aggressively pipelined in [22].

2.2 FPGAs for Sensing and Control

Kozak, in [23], 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 [24], 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 [25], 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 [26], which presents an application-specific design that used machine vision to control an inverted pendulum. In [27], 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 [28], the design of a high-speed, hardware-only, fixed-point MPC is discussed. Finally, in [29], 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.

Another interesting application space for FPGAs is Hardware-In-the-Loop (HIL) testing, in which an FPGA may stand-in for another circuit during the design process. This form of testing is especially of interest in power electronics, as it allows real hardware to be tested in conjunction with a high-fidelity, realtime simulation of an electrical circuit, without the dangers of high power. When equipped with a Digital to Analog converter, an FPGA can approximate the output of a analog or mixed-signal circuit–examples appear in [30, 31, 32]. The work in [33] was able to simulate a model of an Electric Vehicle (EV) drive with a 200ns timestep.

2.3 Software-Configurable Hardware

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 [34] and [35], 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 [36]. In [36] 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,

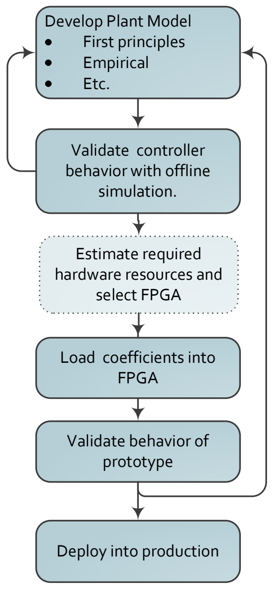

Develop Plant Model
First principles
Empirical
Etc.
Validate controller behavior with offline simulation.
Estimate required hardware resources and select FPGA
Load coefficients into FPGA
Validate behavior of prototype
Deploy into production

Figure 2.1: Application-Agnostic Workflow Supported by our Architectural Vision

unlike our fully software-configurable solutions. Garbergs, in [37, 38], presents the closest vision and architecture to this flexible approach, but there are several differences. Their design does not take into account scaling to different sized controllers (e.g. if controller coefficients change the design must be re-implemented). Also, their design is intended to be standalone, as compared to being a memory-mapped co-processor. Finally, 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.

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. This design methodology helps bridge the gap between controls and embedded system engineering by reducing the barriers for engineers unfamiliar with hardware architecture design. A plant-agnostic workflow is illustrated in Fig. 2.1.

In this workflow, the hardware architecture is not a purpose-built, but rather constructed as an IP Block to facilitate quick prototyping and deployed in a wide range of applications.

The primary configuration data consists of the model data which the controls engineer will generate during the planning and simulation phase of the project.

2.4 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, C, and D[1] and is formulated as follows:

$$x_{k+1} = Ax_k + Bu_k \tag{2.1}$$

$$y_k = Cx_k + Du_k \tag{2.2}$$

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. The state-space method of modeling is central to this dissertation as it leads to straightforward, generalizable hardware implementations.

Typical methods for linearization include Gaussian regression (e.g. least squares method), and the Taylor series method, which requires that the function is differentiable. Although broadly applicable, the process of converting complex, nonlinear plant behavior into a linear model is clearly a lossy one. With this issue in mind, we will extend this model in Section 3.4.

2.5 FPGA-based LQR Coprocessor

The proposed coprocessor is designed to control physical processes representable by a linear state-space model, and illustrates the advantages of the overall design philosophy proposed in the previous sections. It implements a Linear Quadratic Regulator (LQR) coupled with a Luenberger Observer. An LQR controller can also be coupled with a Kalman Filter to produce an Linear Quadratic Gaussian (LQG) controller, which allows a model of (Gaussian) noise to be included in the overall system. Such an effort is out of scope for the present dissertation.

[1] D is rarely required, and therefore often omitted in literature.

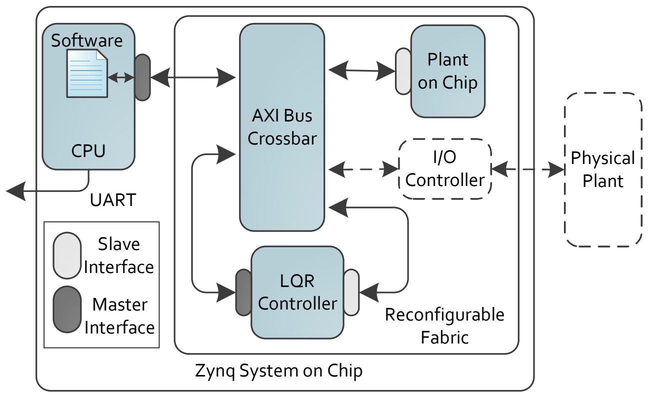


Figure 2.2: System Overview. Both the controller and Plant on Chip (PoC) are fully software configurable over the shared AXI bus. A software or hardware controller can control the PoC without requiring hardware reconfiguration.

The physical process which a control system seeks to control is typically termed a plant. The inverted pendulum on a cart is a well-understood plant model which is often used as a reference plant in the controls community. It is therefore 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, as well as reducing the impact that the compute environment has on performance characteristics.

For the purpose of evaluation, the controller can be interfaced to a hardware-based emulation of a physical plant (i.e. Plant on Chip) [39]. This arrangement is depicted in Fig. 2.2. 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 for initialization and supervision. The software is also free to perform other tasks as required by the application–for example, task scheduling, path planning, video processing, and interactive communications.

2.5.1 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

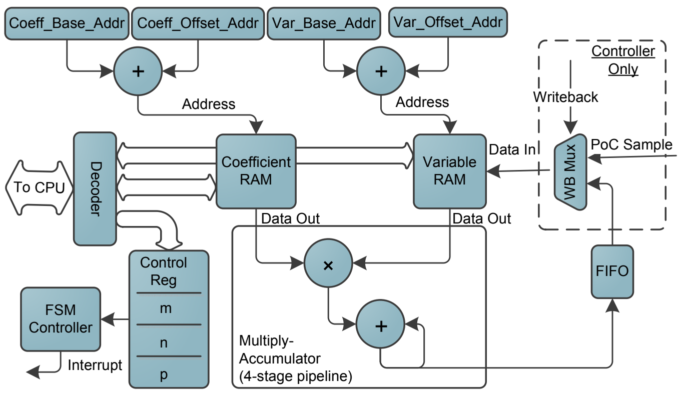


Figure 2.3: Architecture datapath. The dot-product result for each row is stored in the FIFO. After all rows are processed, the FIFO writes results back in the same cycle as they are read back out for the next update operation. An interrupt signal can be used to notify the CPU when values are updated.

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. 2.2, 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. 2.3. 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 2.1; in total 5% of available Slices are consumed. Thus the approach is not only flexible but compact.

Table 2.1: 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**

2.5.1.1 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. 2.4. 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.

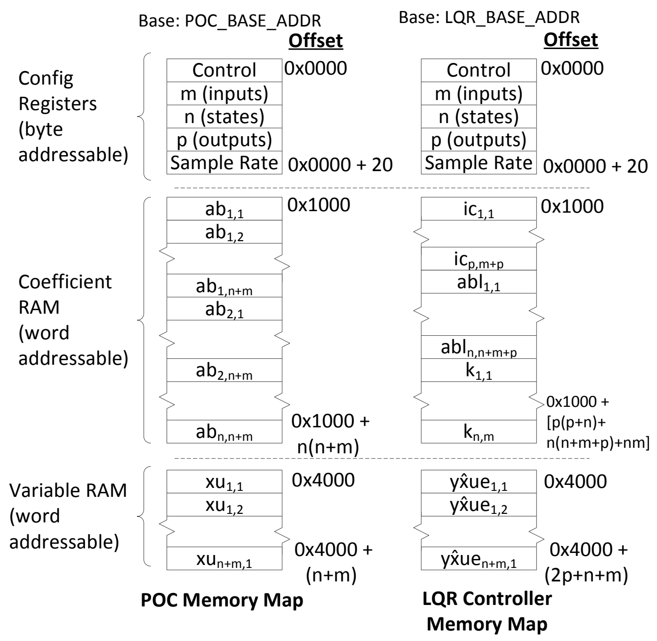


Figure 2.4: Coefficients and variables occupy independent, packed memory spaces, with matrices laid out in row-major order.

2.5.1.2 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 2.3 and 2.4.

$$\begin{bmatrix} x_{1|k+1} \\ \vdots \\ x_{n|k+1} \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} & b_{1,1} & \cdots & b_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} & b_{n,1} & \cdots & b_{n,m} \end{bmatrix} \times \begin{bmatrix} x_{1|k} \\ \vdots \\ x_{n|k} \\ u_{1|k} \\ \vdots \\ u_{m|k} \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{p,1} & \cdots & c_{p,n} \end{bmatrix} \times \begin{bmatrix} x_{1|k} \\ \vdots \\ x_{n|k} \end{bmatrix} \tag{2.4}$$

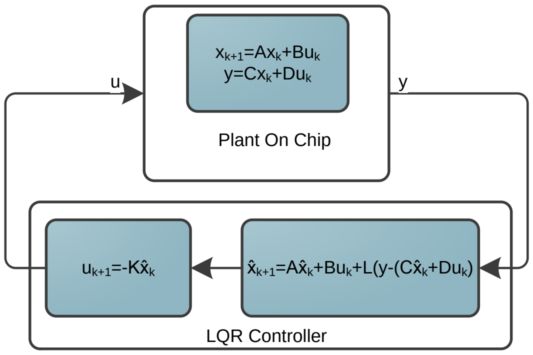


Figure 2.5: LQR controller with observer controlling an emulated plant (PoC).

2.5.1.3 LQR Controller Algorithm

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 Equation 2.5 [40].

$$J(u) = \sum_{1}^{\infty} x_k^T Q x_k + u_k^T R u_k \tag{2.5}$$

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.

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + L(y - C\hat{x}_k) \tag{2.6}$$

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. 2.5.

The control update operation is split into three phases, shown in Equations 2.7-2.9. 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.

$$E_{p\times n} = \begin{bmatrix} I_{p\times p} & -C_{p\times n} \end{bmatrix} \times \begin{bmatrix} y_{p\times 1} \\ \hat{x}_{n\times 1|k} \end{bmatrix} \tag{2.7}$$

$$\hat{x}_{n\times 1|k+1} = \begin{bmatrix} A_{n\times n} & B_{n\times m} & L_{n\times p} \end{bmatrix} \times \begin{bmatrix} \hat{x}_{n\times 1|k} \\ u_{m\times 1} \\ E_{p\times 1} \end{bmatrix} \tag{2.8}$$

$$u_{m\times 1} = \begin{bmatrix} -K_{m\times n} \end{bmatrix} \times \begin{bmatrix} \hat{x}_{n\times 1|k+1} \end{bmatrix} \tag{2.9}$$

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.

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 multiplexer).
 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 $j \leftarrow 0 \ldots 2$ do                      ▷ For each computation phase...
 8:         $sum \leftarrow 0$
 9:         for $c_i \leftarrow 0 \ldots c_{max}[j]$ do                ▷ For each coefficient...
10:             $sum \leftarrow sum + \text{COEF\_MEM}[c_{base}[j]+c_i] \times \text{VAR\_MEM}[v_{base}[j]+v_i]$
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:             $v_i \leftarrow v_i + 1$
```

2.5.2 Evaluation

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 typical low-powered, integer-only 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 [41]. In this experiment, we only consider delay, since the software controller is single-threaded, and the hardware controller is naturally jitter-free (e.g. approximately down to the level of jitter on the system oscillator). 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 which is designed to require 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.

2.5.2.1 Inverted Pendulum Model

The inverted pendulum configuration is illustrated in Fig. 2.6, and the model parameters are shown in Table 2.2. 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 [42, 43] are shown below:

$$(M + m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\text{cos}\theta - ml\dot{\theta}^2\text{sin}\theta = u \tag{2.10}$$

$$(I + ml^2)\ddot{\theta} + mgl\text{sin}\theta = -ml\ddot{x}\text{cos}\theta \tag{2.11}$$

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.

$$(M+m)\ddot{x}+b\dot{x}-ml\ddot{\phi}=u \tag{2.12}$$

$$(I+ml^2)\ddot{\phi}-mgl\phi=ml\ddot{x} \tag{2.13}$$

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

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ \phi_{k+1} \\ \dot{\phi}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ \phi_k \\ \dot{\phi}_k \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u \tag{2.14}$$

Note that the output expression in Equation 2.15 is configured to reflect the fact that only position x and angle ϕ are directly observable.

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ \phi_k \\ \dot{\phi}_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \tag{2.15}$$

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

2.5.2.2 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

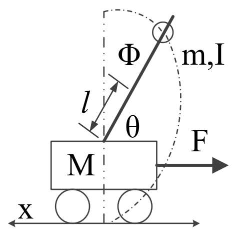


Figure 2.6: Inverted Pendulum Model

Table 2.2: 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	$0.006kg \cdot m_2$
u	applied force	0
x	position displacement	0
θ	angle from downward vertical axis	N/A
ϕ	angle from upward vertical axis	$-5°$

controller workload over repeated trials. Extra zeroed 'dummy' states are added to the 4 base states in order to modulate the controller delay without impacting the model itself.

Fig. 2.7 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. 2.8 shows that increasing the computational workload has very little observable effect on the stability of the plant. Table 2.3 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:

Table 2.3: Execution Time Comparison (μs)

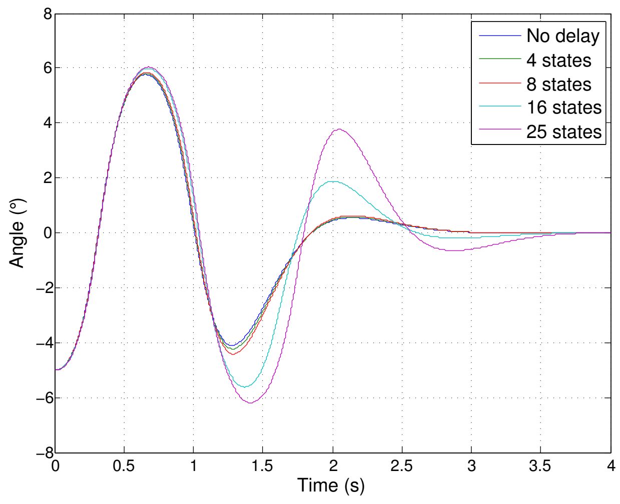
States	Software	Hardware
4	183	9.9
8	428	20.66
16	1254	48.64
32	4071	127.14

$$f(m,n,p) = 2n^2 + p^2 + 3nm + 2pn + 2(p+n+m) \tag{2.16}$$

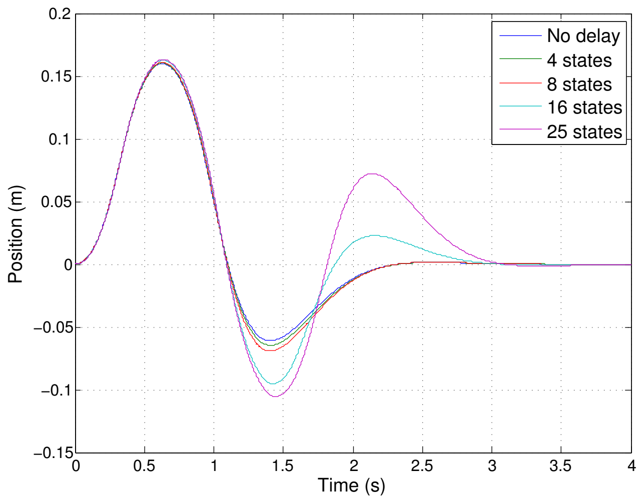
Given that total amount of block RAM on the target device is 560kB, we consider two cases: one actuator and many actuators. Letting $n = p$, which is worst-case memory wise, if $m = 1$ then the maximum number of states is approximately 166. If we constrain $m = n = p$ then the maximum number of states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

2.6 Conclusion

In this chapter 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. In the following chapters, we take these concepts and study how we can apply them to a significantly more complex algorithm: the Kalman Filter.

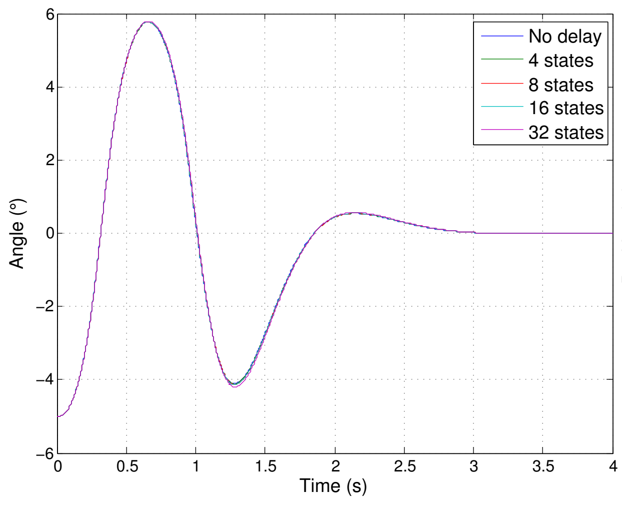


(a) Pendulum angle response

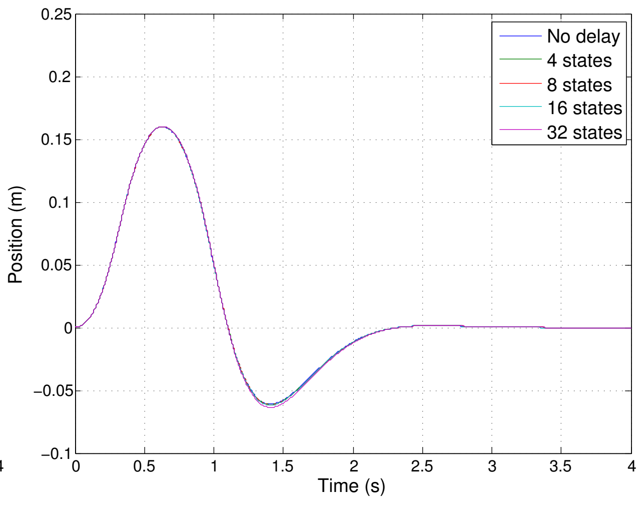


(b) Pendulum position response

Figure 2.7: Impact of computation delay on plant stability (software controller)



(a) Pendulum angle response



(b) Pendulum position response

Figure 2.8: Impact of computation delay on plant stability (hardware controller): the plant responses are nearly indistinguishable with increasing delay.

CHAPTER 3. MIXED HARDWARE-SOFTWARE KALMAN FILTER

3.1 Introduction

The Kalman filter is a common means of accurately estimating the state of a plant in the presence of noise in either measurements or the model itself. As discussed in [6], the computational complexity of the Kalman Filter, when coupled with the high-dimensionality of models and high sample rates required in applications such as inertial navigation systems (INS), would tend to overwhelm processing resources available in the low power microcontrollers used in typical automotive or industrial settings.

Although several algorithmic variations exist, such as Sigma-Point Kalman Filters [44, 45], the Extended Kalman Filter (EKF) is the most commonly used extension to the Kalman Filter, as it consists of only a relatively small modification to the original Kalman Filter algorithm to support non-linear models. Mixed hardware-software implementations of the Extended Kalman Filter (EKF) have been proposed in existing literature [6, 11, 46, 47, 1], which focus on computational acceleration. These designs consist of an application-specific, nonlinear, non-acceleratable part that is computed in software, and a generic matrix-math part which is computable in hardware. We claim that the construction of existing approaches creates an artificial upper bound on both the overall speedup, and fail to explore the broader architectural advantages of dedicated hardware for control systems and mixed-criticality systems.

Instead of focusing on the EKF algorithm, we propose a piecewise affine[1] modeling approach which uses the standard linear Kalman Filter. This not only offers an application-layer speedup over the EKF approach, but allows the complete plant observer to be computed on the coprocessor in a general way, thereby dramatically simplifying the analysis of control loop

[1]Note on terminology: affine indicates a linear relationship plus a translation. Affine is a more flexible form than linear, but these terms are often used interchangeably.

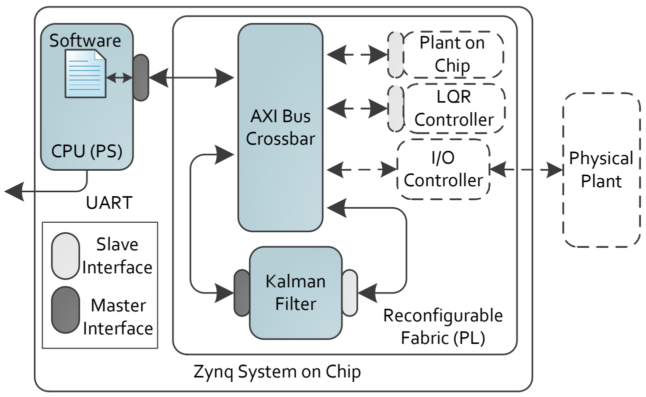


Figure 3.1: System context and hardware-software interconnects.

timing. The high level structure is illustrated in Fig. 3.1, along with functional units proposed in [21].

In the following sections, which elaborate on the work from [48], we describe an implementation of the FPGA architecture supporting the piecewise affine approach, provide an assessment of the scalability of the implementation, and explore the performance tradeoffs between the typical mixed hardware-software EKF approach and the proposed piecewise affine approach. The majority of the literature available on hardware-accelerated Kalman Filtering is application-specific. In this chapter, in order to maintain broad relevancy, we take a unique approach by performing analyses in an application-agnostic manner.

3.2 Kalman Filter and Extended Kalman Filter Algorithms

The Kalman filter, which is optimal only for linear models, involves the repeated application of the following steps at a particular sampling rate:

1. Estimate state
2. Calculate error covariance
3. Calculate Kalman gain
4. Update state estimate based on measurement
5. Update error covariance based on measurement

This process is shown more formally in lines 5 through 9 of Algorithm 2. In this paper, the subscript k indicates the discrete time-step in question. Furthermore the notation x_k^- denotes the state vector *before* performing the measurement update, and x_k^+ to denote the state vector *after* performing the measurement update.

Algorithm 2 Kalman Filter [49]

```
1: procedure INITIALIZE
2:     $\hat{x}_0^- \leftarrow E[x_0]$
3:     $P_{\tilde{x},0}^- \leftarrow E[(x_0 - \hat{x}_0^-)(x_0 - \hat{x}_0^-)^T]$
4: procedure UPDATE
5:     $\hat{x}_k^- \leftarrow f(\hat{x}_{k-1}^+, u_{k-1})$                      ▷ Estimate state
6:     $P_k^- \leftarrow A_{k-1}P_{k-1}^+A_{k-1}^T + Q$                      ▷ Calc. error cov.
7:     $K_k \leftarrow P_k^- C_k^T[C_k P_k^- + R]^{-1}$                      ▷ Calc. gain
8:     $\hat{x}_k^+ \leftarrow \hat{x}_k^- + K_k[y_k - g(\hat{x}_k^-, u_k)]$          ▷ Update state
9:     $P_k^+ \leftarrow (I - K_k C_k)P_k^-$                                  ▷ Update cov.
```

In the EKF, which extends the Kalman Filter to non-linear models, the derivative of the state update and measurement equations $f(\hat{x}_k, u_k)$ and $g(\hat{x}_k, u_k)$ (Equations 3.1 and 3.2) must be determined in advance analytically. During each update step, they are evaluated at the current state estimate [50].

$$A_{k-1}^x = \left.\frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\right|_{x_{k-1}=\hat{x}_{k-1}^+} \tag{3.1}$$

$$C_k^x = \left.\frac{\partial g(x_k, u_k)}{\partial x_k}\right|_{x_k=\hat{x}_k^-} \tag{3.2}$$

Overall these expressions allow us to produce a first-order (linear) estimate of function behavior at the current state value, but increase the computational effort. In cases where the derivative cannot be determined analytically (or it is very difficult), numerical differentiation might be necessary (e.g. finite difference method), which increases the cost yet further since the state update expression (and possibly the measurement update expression) must be evaluated twice per time step.

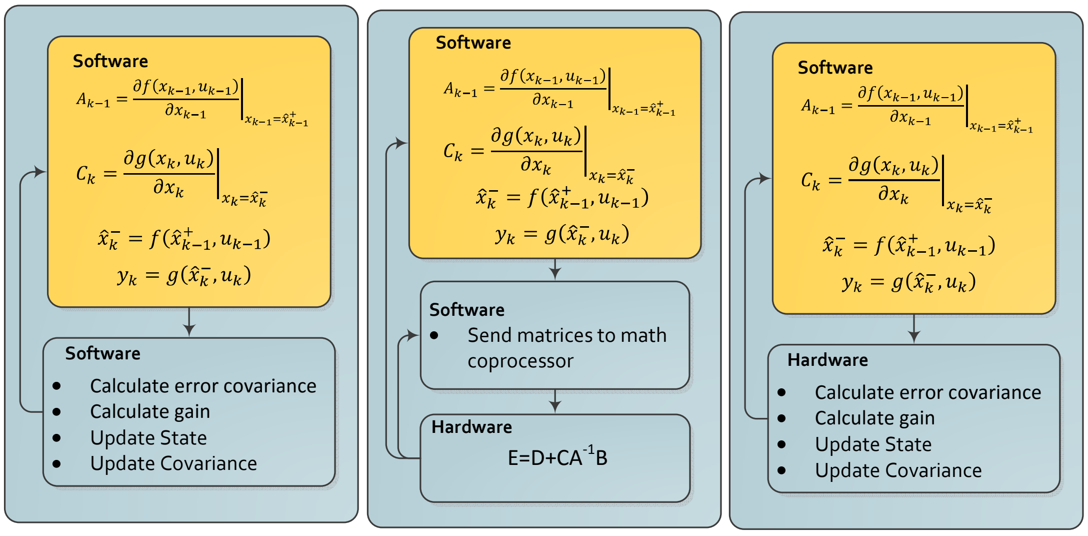
3.3 Hardware Accelerated Kalman Filtering

Hardware-based solutions to manage the computational complexity of the EKF have previously been proposed. An early effort [51, 52] offered substantial speedup over software, but required multiple FPGAs to implement and lacked flexibility. More recently an alternative implementation of the EKF [6] has been proposed for enhanced numerical properties. There has also been proposed a specialized FPGA architecture to support Kalman Filtering for SLAM applications normally processed on a higher power processor such as a Pentium M [11].

Another, more application-agnostic group of implementations employ architectures based on the systolic array. The systolic array is a well-known structure which, when implemented on hardware, can support substantial parallelism while reducing signal fanout and other common bottlenecks to design scaling. It has been recognized for some time as an efficient basis for a hardware-based implementation for matrix math, and subsequently as a means to accelerate the Kalman Filter, since at least the early 90's [53]. In addition to a few other architectural approaches, the work in [53] describes the Fadeev algorithm, which performs modified Gauss-Jordan elimination on an input block matrix, M, comprised of four specially-chosen submatrices, **A**,**B**,**C**, and **D**. The bold typeface has been applied to the submatrix names to differentiate them from those used for state-space notation.

$$M_{2n\times 2n} = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right) \tag{3.3}$$

In Equation 3.3 (and throughout this paper), n is the number of states in the specific state-space model of plant behavior. The work in [46] extends [53] by providing extensive FPGA implementation details, as well as describing a folding approach to reduce spacial complexity (at the expense of increasing time complexity). The approach is partially motivated by the need to maintain high power efficiency for battery-powered applications. The work in [47] adds even more array folding, essentially decoupling the required number of math operations from the required number of processing elements–but in general the implementation complexity is very high.


Software
$A_{k-1} = \frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\Big|_{x_{k-1}=\hat{x}^+_{k-1}}$
$C_k = \frac{\partial g(x_k, u_k)}{\partial x_k}\Big|_{x_k=\hat{x}^-_k}$
$\hat{x}^-_k = f(\hat{x}^+_{k-1}, u_{k-1})$
$y_k = g(\hat{x}^-_k, u_k)$
Software
Calculate error covariance
Calculate gain
Update State
Update Covariance
Software
Send matrices to math coprocessor
Hardware
E=D+CA^{-1}B
Hardware
Calculate error covariance
Calculate gain
Update State
Update Covariance

(a) Software (b) Mixed Hardware/Software (c) Mixed Hardware/Software (with sequencer overlay)

Figure 3.2: Extended Kalman Filter Algorithm Partitioning

Each of these approaches use the Fadeev algorithm to implement a general-purpose matrix algebra functional unit, and then rely on software to provide the application-specific matrices to the hardware at every step within the Kalman Filter algorithm. A recent architectural variation in [1] attempts to reduce the communication overhead (on the order of 30% of the total update time) to the coprocessor by introducing a hardware sequencer to automatically compute the application agnostic steps of the algorithm. However, the matrices representing the linearized model must still be sent to the coprocessor on every iteration. Furthermore, in all aforementioned works the software-based application-specific portion of the update process still requires evaluation of the full non-linear function in the prediction step (e.g. $f(\hat{x}_k, u_k)$ and $g(\hat{x}_k, u_k)$) , and the evaluation of the Jacobians of the non-linear functions, every iteration.

The structural comparison of the software-based EKF and mixed hardware-software implementations is illustrated in Fig. 3.2. The upper (orange-colored) box describes the application-specific portion of the algorithm, while the lower (blue-colored) box describes the generic matrix-math portion which is common to all implementations. The software-only version appears in (a), the existing hardware-software approach from previous publications appears in (b), and the extension described in [1] appears in (c).

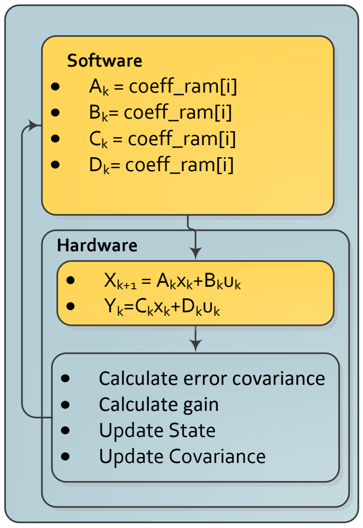


Figure 3.3: Mixed Hardware/Software PWAKF

Our work also employs the Fadeev algorithm to implement the matrix-math portion of the Kalman Filter. However, we also study the use of a piecewise-affine system modeling approach as a means to develop a coprocessor with significantly less communication requirements compared to EKF-based approaches, while maintaining a similar operating domain. For brevity we term this the Piecewise-Affine Kalman Filter (PWAKF), whose partitioning is depicted in Fig. 3.3. The modifications necessary for a fully hardware-based solution (similar to the work in Chapter 2) will appear in the next Chapter.

3.4 Piecewise Affine Model of a Plant

Piecewise affine modeling is a well-known approach to linearizing a non-linear model, as discussed in the summative works [54] and [55]. Rather than linearize a non-linear model around a particular operating point, which yields a single system of equations, it is possible to divide the non-linear space into regions, wherein all points within the region are computed via a linear (or affine) function. The goal is to obtain a much wider operational range, while making a tradeoff of computations for memory. In terms of applications, it is often used as a way to model the nonideal behavior of many engineered systems, such as tire slip in an unmanned ground vehicle [56]. Other systems may explicitly require a piecewise modeling method such as for simulation of switching circuits [57, 58], or Kalman-filter based fault detection for switching circuits [9],which requires very high sample rates. More broadly speaking, it is also an efficient means to model "nearly" linear nonlinear systems.

One issue with piecewise affine modeling may be the difficulty in finding good partitions for large state-space models. It is worth mentioning there is existing work on automated system identification, for example [8].

We assume that non-linear models of interest can be mapped into the following generic state-space form.

$$x_{k+1} = \begin{bmatrix} x_{1|k+1} \\ x_{2|k+1} \\ \vdots \\ x_{n|k+1} \end{bmatrix} = f(x_k, u_k) = \begin{bmatrix} f_1(x_k, u_k) \\ f_2(x_k, u_k) \\ \vdots \\ f_n(x_k, u_k) \end{bmatrix} \tag{3.4}$$

$$y = \begin{bmatrix} y_{1|k+1} \\ y_{2|k+1} \\ \vdots \\ y_{p|k+1} \end{bmatrix} = g(x_k, u_k) = \begin{bmatrix} g_1(x_k, u_k) \\ g_2(x_k, u_k) \\ \vdots \\ g_p(x_k, u_k) \end{bmatrix} \tag{3.5}$$

In the general case, partitioning such a multivariate function into q linear regions will yield a set of equations of the form in Equation 3.6. This formulation extends the basic linear model introduced in Section 2.4.

$$\begin{aligned} x_{k+1} &= A_i x_k + B_i u_k \\ y_k &= C_i x_k + D_i u_k \\ \forall k &= 1, 2, ..., \infty, \forall i = 1, 2, ..., q \end{aligned} \tag{3.6}$$

Each of q regions can each be identified by a globally unique identifier i, which we may treat like an additional hidden state. Our PWAKF approach assumes that the plant is modeled in this form. Note that although Equation 3.6 is strictly linear, affine expressions can be refactored into the linear state-space form. Consider the following affine expression in which c is assumed to be a column vector with as many entries as there are rows in B_i.

$$x_{k+1} = A_i x_k + B_i u_k + c \tag{3.7}$$

This can be normalized using augmented matrices as shown below.

$$x_{k+1} = A_i x_k + \left[\begin{array}{c|c} B & c \end{array} \right] \begin{bmatrix} u_k \\ 1 \end{bmatrix} \tag{3.8}$$

This is an important consideration, since for a piecewise-defined model most regions (besides that at the origin) will require a nonzero translation term.

3.5 System Architecture

Fig. 3.1 shows the general structure of the hardware. The targeted hardware platform for implementation is the Xilinx Zynq 7z020 system-on-a-chip (SoC). The Zynq SoC consists of an ARM Cortex A9 processor (termed the Processing System, or PS) coupled with an FPGA (termed the Programmable Logic, or PL). 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 coprocessor to communicate with the PS, or allows independent communication among co-processors. As shown in Fig. 3.1, the coprocessor has a slave interface, which allow their internal memory spaces to appear to the PS as memory-mapped peripherals–an interface very familiar to the embedded engineer. Meanwhile, the AXI Bus master interface on the coprocessor allows it to read from sensors or other functional units within the FPGA while the system is running.

The architecture within the coprocessor is shown in Fig. 3.4. The regular structure of the systolic array in Fig. 3.5 allows us to use VHDL's *generate* and *loop* statements in such a way that the systolic array can be produced to support processing of a state-space model of arbitrary size n. This enormously simplifies the work needed to test the design at various scales.

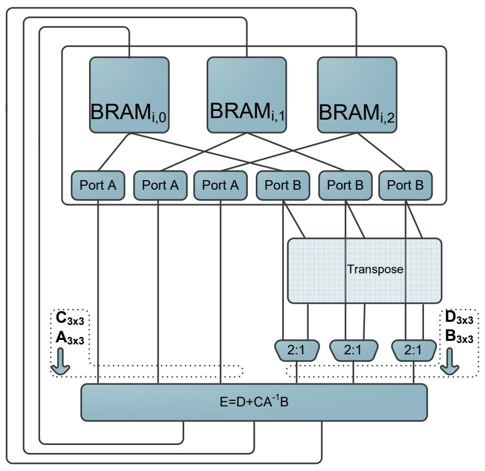


Figure 3.4: Architecture datapath (n = 3) to support the Fadeev algorithm math unit. All routes shown have 32-bit width. The columns of block matrix M emerge from each BRAM port and enter the Fadeev unit at the bottom.

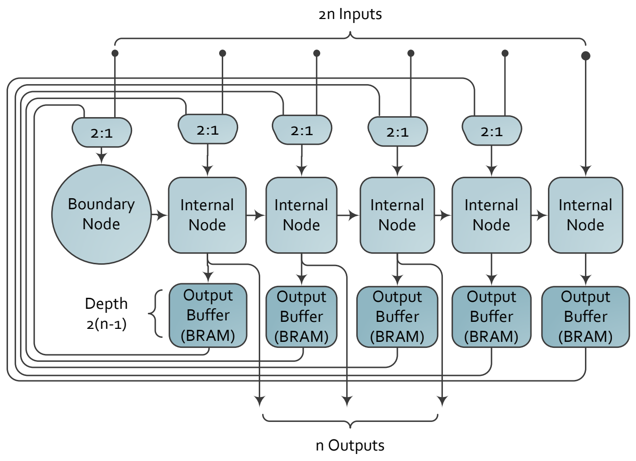


Figure 3.5: Internal systolic structure of Fadeev functional unit (n=3).

3.5.1 Fadeev Algorithm

As the Fadeev algorithm has already been implemented in the previous work described in Section 3.3, we only describe the approach abstractly here; more details on the theory of the algorithm appear in [59]. The algorithm is implemented as a non-homogeneous array consists of a node which performs division, which is referred to as a Boundary Node (the pivoting column in Gauss-Jordan elimination) and $2n - 1$ Internal Nodes, which perform multiplication and addition (where n is the number states). The result is the Schur Complement of the **A** matrix

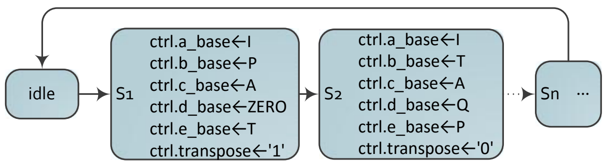


Figure 3.6: The state machine for the control unit is simple and flexible enough to implement a variety of algorithms. In each state, a set of base pointers into memory are set to the appropriate constant value at that step in the algorithm.

of Equation 3.3. The Schur Complement is shown in Equation 3.9, and the hardware structure appears in Fig. 3.5.

$$E = \mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \tag{3.9}$$

As an example (similar to step 5 of Table 3.1, for instance), the following demonstrates the block matrix inputs needed to compute the matrix inverse of some matrix $\mathbf{A}$, using identity matrix I and zero matrix $\mathbf{0}$.

$$M = \left(\begin{array}{c|c} \mathbf{A} & I \\ \hline I & \mathbf{0} \end{array}\right) \tag{3.10}$$

The Fadeev algorithm then consumes input matrix M and effectively computes the Schur Complement of the $\mathbf{A}$ submatrix, which yields $\mathbf{A}^{-1}$.

$$E = 0 + I\mathbf{A}^{-1}I = \mathbf{A}^{-1} \tag{3.11}$$

Our work thus far uses [46] as a basis for implementing the Fadeev algorithm, as it scales better in hardware resources than a full systolic array implementation (linear vs. quadratic). The tradeoff is that runtime is a function of n^2, as inputs must be iteratively cycled through a single row of nodes rather than a full 2-dimensional array. A schedule of inputs is developed for this basic hardware element to produce a complete hardware-accelerated Kalman Filter (Tables 3.1 and 3.2). The control block consists of a state machine which need only set the correct set of base pointers for reading and writeback during each algorithm step (Fig. 3.6). Therefore it is straightforward to implement different variations of the same algorithm.

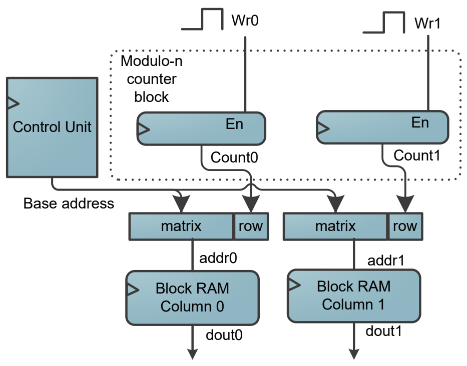


Figure 3.7: Example address generation circuit ($n = 2$) for writeback. Read address generation uses an identical circuit driven by a set of "Rd" signals.

3.5.2 Memory Interface

In the worst case the Fadeev systolic array needs to write back n words simultaneously, and read $2n$ words simultaneously; thus we need to place matrix columns in separate dual-port block RAMs (BRAMs). For each BRAM, Port A is attached to the columns 0 to $n/2 - 1$ of the Fadeev input bus (left side), and Port B is attached to columns $n/2$ to $n - 1$ (right side).

From the perspective of the PS, the global address to locate a specific element within a matrix is formed by concatenating the matrix (base) address, the matrix row address, and the matrix column address. The latter two address components are $ceil(log_2(n))$ bits wide. Columns are located in separate BRAMs, and rows are accessed in parallel by applying the same address to the n column BRAMs. Non-matrix variables (such as x, y, etc) in the PWAKF occupy the same sized memory block as any other matrix, and the unused matrix elements are simply filled with zeros.

Within the coprocessor itself, it is not necessary to specify the row column address, as it is implicit in the structure of the interconnects. The signals from the Fadeev unit signaling that the writeback data is valid drive n instances of *modulo-n* counters, which produce the row address within each BRAM. An example of address generation for writeback is shown in Fig. 3.7. Independent selection of rows is needed to accommodate the skewed input-output pattern of the systolic array.

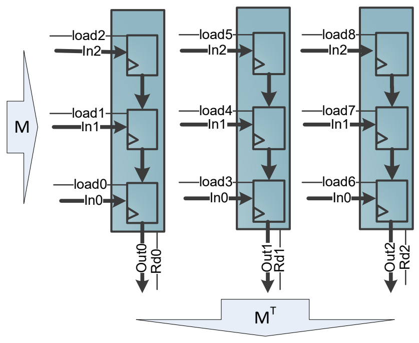


Figure 3.8: Architecture of transpose functional unit ($n = 3$), comprised of parallel shift registers.

3.5.3 Matrix Transpose

As we are attempting to implement the full Kalman Filter in hardware, we need to consider how to handle the matrix transposes. The obvious solution is to pre-compute them and store them in memory. However, after partitioning the algorithm there are several temporary matrices for which the transpose cannot be pre-computed–it has to be done online.

Another option is to simply manipulate the address provided to the BRAMs so that we sequentially fetch all elements in a column of our input matrix to produce the row of our transposed matrix. This is not a scalable solution, however, since multiplexers will be needed between the BRAMs and the Fadeev input so that each BRAM output port can be re-routed to every array input column. For large n and 32-bit words, this quickly results in thousands of multiplexed connections.

Instead, we propose a simple functional consisting of shift registers to transform the rows output from the BRAMs into columns (Fig. 3.8). Matrix property $AB^T = (BA^T)^T$ allows us to move all transposes in the Kalman algorithm to the **B** position within our input block matrix (Equation 3.3). As a result, we need only provide this unit on the right-hand side datapath. Furthermore, due to the skewed input pattern characteristic of systolic arrays, inputs on the left-hand side will be consumed first, so that inputs on the right-hand side do not need to be available immediately (ie instead, n cycles after the first element on the furthest left-hand side is input). This effectively hides the extra clock latency involved in loading the transpose unit.

Table 3.1: Input Schedule for Extended Kalman Filter

Step	**Computation**	**Fadeev Input** (M)
1	$T = AP^T$	$\begin{bmatrix} I & P^T \\ A & 0 \end{bmatrix}$
2	$P = AT^T + Q$	$\begin{bmatrix} I & T^T \\ A & Q \end{bmatrix}$
3	$T = CP^T$	$\begin{bmatrix} I & P^T \\ C & 0 \end{bmatrix}$
4	$K = CT^T + R$	$\begin{bmatrix} I & T^T \\ C & R \end{bmatrix}$
5	$K = TK^{-1}$	$\begin{bmatrix} K & I \\ T & 0 \end{bmatrix}$
6	$P = P - KT^T$	$\begin{bmatrix} I & T^T \\ -K & P \end{bmatrix}$
7	$x = x + KE$	$\begin{bmatrix} I & E \\ K & x \end{bmatrix}$

3.5.4 Hardware-Software Partition

We next break down the workload distribution between the PS (i.e. software) and the PL (i.e. hardware) to distinguish the EKF (based on paritioning in Fig. 3.2c) and PWAKF approaches.

3.5.4.1 EKF Approach

For the EKF, the PS needs to obtain the previous state estimate from the PL ($\hat{x}_{k-1}$), then uses it along with the non-linear state update equation $f(x_k, u_k)$ and measurement equation $g(x_k, u_k)$ to compute the following in software.

$$A^x_{k-1} = \frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\bigg|_{x_{k-1}=\hat{x}^+_{k-1}} \tag{3.12}$$

$$\hat{x}^-_k = f(\hat{x}^+_{k-1}, u_{k-1}) \tag{3.13}$$

$$C^x_k = \frac{\partial g(x_k, u_k)}{\partial x_k}\bigg|_{x_k=\hat{x}^-_k} \tag{3.14}$$

$$y_k = g(\hat{x}^-_k, u_k) \tag{3.15}$$

$$E_k = z_k - y_k \tag{3.16}$$

The resulting A_k and C_k matrices, along with the state estimate $\hat{x}_k$ and vector E_k need to be written back to the PL. Vector E_k from Equation 3.16 is the error between the sensor measurement and estimated measurement value based on the plant model. The PL sequence of computations using the Fadeev hardware are listed in Table 3.1, and correspond to lines 6-9 in Algorithm 1. In the table, subscripts are omitted for simplicity. At the completion of this computation, the PS will read the updated state estimate from the PL in preparation for the next iteration.

3.5.4.2 PWAKF Approach

For the PWAKF, the PS must obtain the previous state estimate from the PL ($\hat{x}_{k-1}$). It then needs to determine the region index i within the piecewise-affine state-space that this state lies (see Equation 3.6). If there are relatively few regions, a simple region by region bounds check would suffice. For more complex systems, a binary search tree might be used such as in [60]. If hyperrectagular partitions are used, constant-time lookup is possible as the current region identifier can be used as an index into a multi-dimensional array.

The PS then writes to the PL a subset of elements of A_k, B_k, C_k or D_k as required by the linear model, but *only if we have entered a new region*, which reduces AXI bus communication overhead. This can be done simply by maintaining the index of the current region in software.

The PL sequence of computations using the Fadeev hardware are listed in Table 3.2. Note the additional steps 7-10, which are the linearized computations corresponding to the non-linear function computed in software for the EKF approach. Another difference is that the error term $z - y$ is computed in software for the EKF approach (and later transmitted over the AXI bus), but is computed in hardware (step 11) in the PWAKF approach. At the completion of these computations, the PS will read the updated state estimate from the PL in preparation for the next iteration.

Table 3.2: Input Schedule for Piecewise-Affine Kalman Filter

Step	Computation	Fadeev Input (M)
1	$T = AP^T$	$\begin{bmatrix} I & P^T \\ A & 0 \end{bmatrix}$
2	$P = AT^T + Q$	$\begin{bmatrix} I & T^T \\ A & Q \end{bmatrix}$
3	$T = CP^T$	$\begin{bmatrix} I & P^T \\ C & 0 \end{bmatrix}$
4	$K = CT^T + R$	$\begin{bmatrix} I & T^T \\ C & R \end{bmatrix}$
5	$K = TK^{-1}$	$\begin{bmatrix} K & I \\ T & 0 \end{bmatrix}$
6	$P = P - KT^T$	$\begin{bmatrix} I & T^T \\ -K & P \end{bmatrix}$
7	$T = Ax$	$\begin{bmatrix} I & x \\ A & 0 \end{bmatrix}$
8	$x = T + Bu$	$\begin{bmatrix} I & u \\ B & T \end{bmatrix}$
9	$T = Cx$	$\begin{bmatrix} I & x \\ C & 0 \end{bmatrix}$
10	$y = T + Du$	$\begin{bmatrix} I & u \\ D & T \end{bmatrix}$
11	$T = z - y$	$\begin{bmatrix} I & I \\ -y & z \end{bmatrix}$
12	$x = x + KT$	$\begin{bmatrix} I & T \\ K & x \end{bmatrix}$

3.6 Implementation Results

3.6.1 Resources

The Zynq chip has $n_{dspmax} = 220$ available DSPs, and our usage directly impacts our space-time tradeoff. These are a critical resource for floating point-heavy designs and should be used strategically. There are $2n - 1$ Internal Nodes in the array (where n is the number of states), each containing a multiplier and an adder. If we allocate n_{dsp}=4 DSPs per Internal Node, the maximum value of n is 28. If we allocate 5 DSPs per Internal Node, the maximum value of n

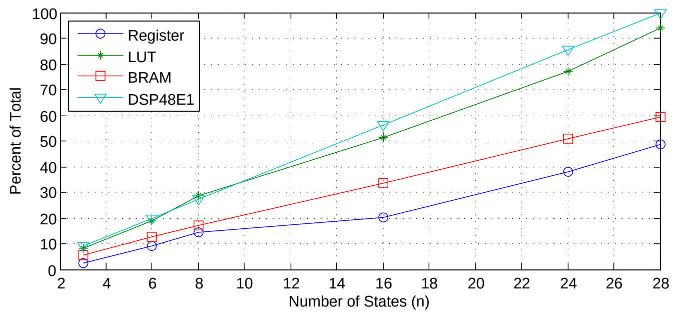


Figure 3.9: Architecture resource consumption for various values of n. Scaling halts at $n = 28$ due to exhaustion of DSP units, although there are additional SoCs in the Zynq line with significantly more DSPs. Additional PEs would be implemented in reconfigurable logic, and would be quickly exhausted.

is 21 (without increasing FF/LUT usage). Thus to ensure all nodes use only DSPs, we must observe the following constraint.

$$n_{dsp}(2n - 1) \leq n_{dspmax} \tag{3.17}$$

The system resources were assessed at various sizes of n in Fig. 3.9. The difference between the EKF and PWAKF approach lies only in the controller steps; thus the impact on LUT usage is negligible. Resource growth is primarily linear in nature: LUT growth is a linear function of n, BRAM growth is $2n$, DSP growth is $n_{dsp}(n - 1)$. The transpose unit consists of an array of registers and grows as a factor of n^2.

3.6.2 Performance

The performance of the PWAKF approach is compared to that of the EKF approach: first at the hardware level, which makes for easier comparison across implementations, and then at the mixed hardware-software level, for which most existing literature presents highly application-specific results. Software-based EKF benchmarks use GNU Scientific Library–a well-known library with optimized matrix algebra routines–and use *gcc*'s flag $-O2$. The ARM NEON (SIMD) instructions are not activated at this optimization level, since NEON is not IEEE 754 compliant and may result in loss of precision–not desirable for critical code. The FPGA implementations are clocked at 45Mhz, while the ARM processor is operated at 200Mhz which lies in the spectrum of microcontroller frequencies deployed widely in embedded systems [6]. Mi-

Table 3.3: Hardware-Only PWAKF & Hardware-Only EKF vs. Software-Only EKF

n	EKF SW 200Mhz Time (ms)	PWAKF HW (45Mhz) Time (ms)	PWAKF HW (45Mhz) Speedup	EKF HW (45Mhz) Time (ms)	EKF HW (45Mhz) Speedup
2	0.073	0.011	6.70	0.006	10.10
4	0.069	0.019	3.57	0.011	5.46
6	0.156	0.043	3.64	0.025	5.82
8	0.258	0.075	3.45	0.044	5.52
10	0.430	0.115	3.73	0.067	6.12
12	0.692	0.164	4.21	0.096	6.95
14	0.985	0.222	4.44	0.130	7.40
16	1.396	0.288	4.85	0.168	8.11
18	2.020	0.363	5.57	0.212	9.33
20	2.681	0.446	6.01	0.260	10.11
22	3.422	0.538	6.36	0.314	10.74
24	4.264	0.638	6.68	0.372	11.27
26	5.283	0.747	7.07	0.436	11.92
28	6.409	0.864	7.42	0.504	12.51

crocontrollers in this performance range are more likely to contain a memory management unit (MMU) and other such peripheral support which makes them more attractive for multitasking under a real-time operating system.

3.6.2.1 Hardware Level

A performance comparison of the PWAKF hardware and EKF hardware vs. a direct software implementation of the EKF algorithmic steps appears in Table 3.3. The PWAKF coprocessor shows less speedup compared to the EKF coprocessor, because it performs several additional computation steps. However, we have not yet considered the additional software processing and communication time that the EKF requires. This is explored in the following section.

Of interest is the elevated speedup around $n = 2$ to $n = 4$ both in Table 3.3 and Fig. 3.11. This is due to the fact that the software processor has a relatively fixed runtime overhead which the dedicated hardware does not–e.g. function calls, branch tests, etc. which represents a large percent of the overall runtime for small values of n.

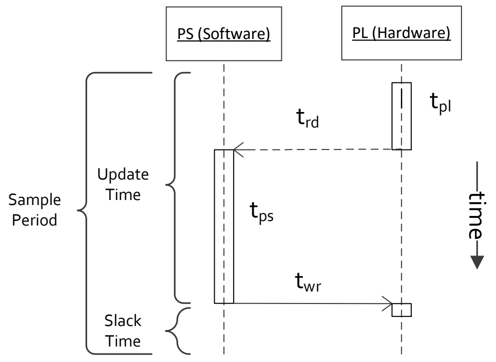


Figure 3.10: Sequence Diagram showing the set of delays considered in this study. With this configuration, the update timing is driven by the hardware. The software prepares the matrix values needed for the next iteration.

The number of clock cycles for a complete algorithm iteration can be analytically expressed as a function of the latency of the divider, multiplier, and adder units. This allows us to determine the number of clocks needed for various values of n, shown in Equation 3.18.

$$k = (D_{ma}(2((n^2) - 1)) + D_{ma}(2n - 2))K_s + D_d \tag{3.18}$$

The relevant symbols are as follows: D_{ma} is the clocks needed to complete either multiplication or addition (they are the same), D_d is the clocks needed to complete division, n is the number of states, and K_s is the number of steps in the Kalman algorithm.

3.6.2.2 Mixed Hardware-Software Level

The advantage of the PWAKF approach does not become apparent without analyzing the mixed hardware-software performance. In this section we analyze how the PWAKF can provide a speedup over the EKF, while also shifting the majority of the computation (greater than 99%) to timing-deterministic hardware.

The general idea is that the PWAKF approach will present less communication overhead and less overhead on the application-specific part of the computation in software, because these processes need only occur when the state of the plant under observation migrates into a new state-space region. For example, a plant which is operating in steady-state may incur no matrix-update communication overhead at all.

Since the performance of both the EKF and PWAKF approaches depend on the characteristics of the application, in order to make generalized analysis, we introduce some abstract application parameters. Fig. 3.10 summarizes the sources of processing delay which are considered during this analysis, and analysis parameters are described in Table 3.4.

As the number of states n needed in the plant model increases, so does the hardware-software communication time ($t_{rd} + t_{wr}$), the software processing time (t_{ps}) and the hardware processing time (t_{pl}). For the EKF approach, the communication time is rather predictable: during each update, the PS will read the n-word state vector from the PL, and write back the $n^2 + nm + p$ words of A,C, and E.

For the PWAKF approach, the communication time depends on the number of elements in the linear state-space model which are not constant across regions. This ranges from 0 (a trivial case in which the plant model was already linear), to $n(n+m) + p(n+m)$, meaning the entire model (A,B,C and D) needs to be transmitted during every region transition. This latter case is quite unlikely as it requires every state variable in the model to appear in all state update equations, all output equations, *and* be involved in a non-linear operation. For example, consider the non-linear state update equation for a system consisting of two states, and arbitrary constants c_x.

$$x_{k+1} = f(x_k, u_k) = \begin{bmatrix} c_1 x_1^2 + c_2 x_2^2 \\ c_3 x_1 + c_4 x_2 + u \end{bmatrix} \tag{3.19}$$

Based on Equation 3.6, the non-linear model can be expressed as a piecewise affine model with q regions, shown below.

$$x_{k+1} = \begin{bmatrix} m_{i,1} & m_{i,2} \\ c_3 & c_4 \end{bmatrix} x + \begin{bmatrix} b_{i,1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ u \end{bmatrix} \qquad \forall i = 1, 2, ..., q \tag{3.20}$$

Slope ($m_{i,x}$) and offset ($b_{i,x}$) elements are introduced, and the vector containing the single input u has been augmented with a constant 1 to accommodate the offset. Specific values are determined after partitioning and system identification; for example a simple point-to-point procedure is described in [61]. During a region transition, only these entries need to be updated in the PL memory, as they differ among regions.

For analysis, we use Equation 3.19 as a template for our non-linear plant model, and modulate the "complexity" of the model by adding additional non-linear (quadratic) terms based on parameter r_{nc}, which ranges from 0 to 1. For example, if r_{nc}=0.5, this indicates half of all elements in the linear model are variables and therefore must be transfered during a region transition. Plant model profiles based on this template consist of Low Complexity (r_{nc}=0.1), Moderate Complexity (r_{nc}=0.5), and High Complexity (r_{nc}=0.9).

Since the *rate* of region transition varies by application, we also introduce a transition rate parameter r_t which varies from 0 (i.e. there are no region transitions) to 1 (i.e. a region transition occurs every timestep). Finally, overall mixed hardware-software speedup is assessed using Equation 3.21.

$$speedup = \frac{t_{swb}}{t_{pl} + t_{ps} + t_{rdb}n + t_{wrb}r_t n_{wb}} \tag{3.21}$$

We further define n_{wr} (number of words to write back to the PL) as shown in Equation 3.22.

$$n_{wb} = \lfloor (n(n+m) + p(n+m))r_{nc} \rfloor \tag{3.22}$$

Analysis results based on Equation 3.21 and the application profiles are summarized in Table 3.5. To manage the number of variables, we assume $m = p = 1$, which indicates a single-input single-output system. The speedup of the hardware-software PWAKF in the best case ($r_t = r_{nc} = 0$) and worst case ($r_t = r_{nc} = 1$) is compared to the hardware-software EKF in Fig. 3.11. On average, the speedup for the PWAKF HW-SW approach is 62% larger than the EKF HW-SW approach. Many times it is more useful to consider the maximum update frequency, which can be determined by computing the inverse of the sum of all delays in Fig. 3.10. Thus these results also indicate that the maximum update frequency will be in the average case 62% higher using the PWAKF.

Table 3.4: Performance Analysis Parameters

Symbol	**Description**
n	Number of states in linear state-space model
m	Number of control inputs in linear state-space model
p	Number of measurable outputs in linear state-space model
r_{nc}	Rate of non-constant entries in the linear state-space model
r_t	Plant region transition rate (transitions per unit timestep)
t_{ps}	Time spent on the software processor (PS)
t_{pl}	Time spent on the hardware processor (PL)
t_{rd}	Time spent by the PS reading from the PL
t_{wr}	Time spent by the PS writing to the PL
t_{rdb}	Benchmarked time for the PS to read a word from the PL
t_{wrb}	Benchmarked time for the PS to write a word to the PL
t_{swb}	Benchmarked software-only implementation execution time

Table 3.5: PWAKF HW-SW Speedup vs. Software-Only Approach, with Varying Model Characteristics

	Low Complexity (r_{nc}=0.1)					Moderate Complexity (r_{nc}=0.5)					High Complexity (r_{nc}=0.9)				
n \ r_t	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1
2	13.70	13.32	12.96	12.62	12.30	13.70	12.62	11.70	10.90	10.20	13.70	11.99	10.66	9.59	8.72
4	4.17	4.09	4.00	3.92	3.85	4.17	3.95	3.75	3.57	3.41	4.17	3.82	3.53	3.28	3.06
6	4.23	4.17	4.10	4.04	3.98	4.23	4.04	3.86	3.70	3.55	4.23	3.92	3.65	3.41	3.21
8	4.03	3.97	3.91	3.86	3.80	4.03	3.86	3.70	3.56	3.42	4.03	3.75	3.51	3.30	3.11
10	4.33	4.27	4.22	4.17	4.11	4.33	4.16	4.00	3.85	3.71	4.33	4.05	3.80	3.58	3.38
12	4.80	4.74	4.69	4.64	4.58	4.80	4.61	4.44	4.28	4.13	4.80	4.49	4.22	3.98	3.76
14	5.03	4.98	4.92	4.87	4.82	5.03	4.84	4.67	4.50	4.35	5.03	4.72	4.44	4.19	3.97
16	5.44	5.38	5.33	5.27	5.22	5.44	5.24	5.05	4.88	4.72	5.44	5.10	4.80	4.54	4.30
18	6.17	6.11	6.04	5.98	5.93	6.17	5.94	5.74	5.54	5.36	6.17	5.79	5.46	5.16	4.90
20	6.60	6.53	6.47	6.41	6.34	6.60	6.36	6.14	5.93	5.74	6.60	6.20	5.84	5.53	5.25
22	6.95	6.88	6.82	6.75	6.69	6.95	6.70	6.47	6.26	6.06	6.95	6.53	6.16	5.83	5.53
24	7.28	7.21	7.15	7.08	7.02	7.28	7.02	6.79	6.56	6.36	7.28	6.85	6.46	6.12	5.81
26	7.65	7.58	7.51	7.45	7.38	7.65	7.38	7.14	6.90	6.69	7.65	7.20	6.79	6.43	6.11
28	8.00	7.93	7.86	7.79	7.72	8.00	7.72	7.46	7.22	7.00	8.00	7.53	7.11	6.73	6.40

3.6.2.3 Discussion

Although in the hardware-only implementation the PWAKF does not offer as large a speedup as the EKF approach when compared to the equivalent software-only implementation (Table 3.3), when the timing of a *complete application* is considered, the PWAKF approach outperforms the EKF approach due to the reduced time spent in the PS and on communication. At the hardware level, speedup for the hardware-software PWAKF could be further increased by reconsidering steps 7-10 of the PWAKF algorithm, which are rather trivial with respect to the time complexity of the Fadeev algorithm and would be significantly faster if computed in a separate functional unit. This approach was used in [1], reporting an additional 62% performance increase compared to the Fadeev-only hardware unit.

Table 3.6: PWAKF HW-SW Update Time (μs), with Varying Model Characteristics

	Low Complexity (r_{nc}=0.1)					Moderate Complexity (r_{nc}=0.5)					High Complexity (r_{nc}=0.9)				
n \ r_t	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1	0	0.25	0.5	0.75	1
2	4.8	5.0	5.1	5.2	5.4	4.8	5.2	5.7	6.1	6.5	4.8	5.5	6.2	6.9	7.6
4	19.7	20.1	20.6	21.0	21.4	19.7	20.8	21.9	23.0	24.1	19.7	21.5	23.3	25.1	26.9
6	43.2	43.8	44.5	45.2	45.9	43.2	45.2	47.3	49.4	51.4	43.2	46.6	50.1	53.5	57.0
8	75.2	76.3	77.4	78.5	79.6	75.2	78.5	81.8	85.1	88.4	75.2	80.7	86.2	91.7	97.2
10	115.6	117.1	118.6	120.1	121.7	115.6	120.4	125.2	130.1	134.9	115.6	123.7	131.9	140.0	148.2
12	164.8	166.7	168.6	170.6	172.5	164.8	171.4	178.0	184.6	191.3	164.8	176.1	187.4	198.7	210.0
14	222.4	224.8	227.3	229.8	232.3	222.4	231.1	239.7	248.4	257.1	222.4	237.3	252.2	267.1	282.0
16	288.5	291.5	294.6	297.6	300.6	288.5	299.5	310.6	321.6	332.7	288.5	307.5	326.6	345.7	364.7
18	363.2	366.9	370.6	374.3	378.1	363.2	376.8	390.5	404.2	417.8	363.2	386.8	410.4	434.0	457.6
20	446.4	450.8	455.2	459.6	464.0	446.4	462.9	479.5	496.1	512.6	446.4	475.1	503.8	532.5	561.2
22	538.0	543.2	548.3	553.4	558.5	538.0	557.8	577.5	597.3	617.0	538.0	572.4	606.8	641.2	675.6
24	638.4	644.3	650.2	656.2	662.1	638.4	661.5	684.7	707.9	731.1	638.4	678.8	719.3	759.7	800.2
26	747.2	753.9	760.7	767.5	774.2	747.2	774.1	801.0	827.9	854.9	747.2	794.2	841.3	888.4	935.5
28	864.5	872.2	879.9	887.7	895.4	864.5	895.4	926.3	957.3	988.2	864.5	918.6	972.7	1026.9	1081.0

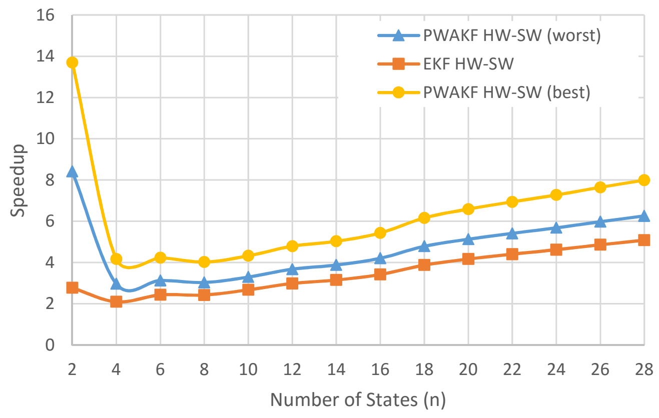


Figure 3.11: Mixed hardware-software speedup over software vs. number of states: PWAKF shows an average 62% performance increase over hardware-software EKF. The large speedup for small n is due to unavailability of NEON instructions at that problem size, highlighting the high impact that these instructions have on software execution time.

Theoretically, for the EKF linearization occurs at runtime (during each discrete time step), while for the PWAKF the function is linearized offline and stored in state-space form. Accurate state tracking for the EKF depends (in part) on the time step being sufficiently small to support the assumption that the plant evolves linearly within the time step; likewise for the PWAKF, good tracking behavior depends on the regions being sufficiently small to assume linearity within that region of the plant's state-space. Therefore, EKF may be preferable if the number of regions needed to achieve the desired accuracy for the PWAKF method induces substantial memory overhead, such as for systems with very fast oscillations. As a straightforward enhancement to the PWAKF, introducing a small amount of hysteresis (e.g. overlap) in the region boundaries could further reduce overhead for noisy or oscillatory systems.

In terms of speedup, it is evident larger problems will benefit more than smaller ones. However, for any application, time-deterministic computations allow a control engineer to focus on plant dynamics rather than worry about (and compensate for) non-idealities in the computational platform.

3.6.3 Power Consumption

The Xilinx Power Estimator tool [62], combined with signal activity rates obtained by simulating our largest hardware design ($n = 28$), estimates the Zynq chip to consume 722mW overall, including dynamic and static power. Of this, the coprocessor (PL) alone consumes 149mW. Therefore the approach is power-efficient and is suitable for many embedded systems with limited power budgets.

3.7 Conclusion

The PWAKF coprocessor is a novel approach to hardware acceleration for Kalman-filter state estimation, establishing a new reference point in the mixed hardware-software design continuum. By replacing the standard EKF methodology with a fully linearized one, it offers a speedup over both the pure software approach, as well as the hardware-software EKF approach, without sacrificing the modeling expressiveness that software enjoys. There are several avenues for further exploration. Placing constraints on the model partitioning scheme (e.g. hypercubes [60]) may enable the region-identification task to be placed in hardware. Of additional interest is the possibility to integrate the Kalman Filter with prior work in FPGA-based LQR control [21] to form more advanced FPGA-based controllers. The ability to develop a complete system on an FPGA brings with it the promise of high-speed, low-power, tightly integrated control systems.

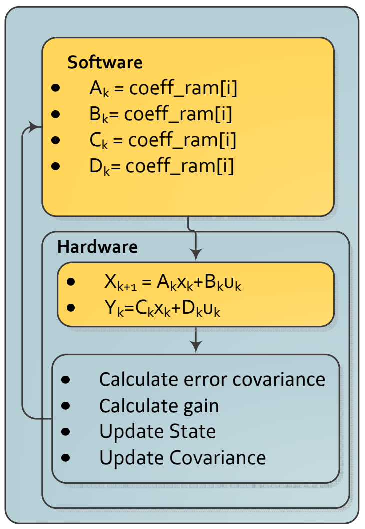
CHAPTER 4. HARDWARE BASED KALMAN FILTER

4.1 Introduction

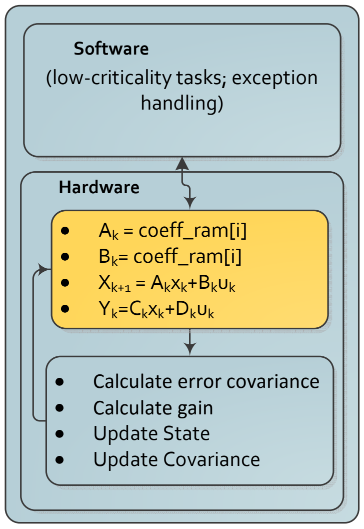
Real time systems are subject to random delays form a variety of sources: network, shared compute resource (task switching), etc. Sampling or actuation (e.g. control) jitter causes control system degradation due to the control signal not arriving at the right time. This can lead to an increase in the cost (i.e. energy) required to control system–or in the worst case, instability. Constant delay can easily be compensated for at design time, but jitter cannot. A study on the impact of jitter on motion controllers[17] found that 8% control jitter (that is, the magnitude of the jitter is 8% of the sample period) can result in regulation error twice as much as sensor noise, causing a 90% increase in regulation error. Overall this jitter contributes significant error in high frequency tracking. To mitigate the issue in [17], calibrating factors are added to the controller to reduce the impact of jitter.

Online compensators are often proposed as means to reduce the impact of jitter, for example using timestamps[16] or a mixture of control theoretic and scheduling compensations [63]. The work in [63] reported a $O(n^4)$ runtime complexity overhead, and significant memory requirements (relative to common microcontroller) for a table-oriented version. High-overhead compensatory measures tend to place limitations on the maximum sample rate which can be reached in such a system.

The alternative vision in this work, which we reiterate here, is to begin with a time-deterministic compute platform rather than adding compensatory measures after issues are observed. As demonstrated in Chapter 2, hardware oriented designs provide higher update rates and lower jitter than software or mixed approaches, ensuring system behavior matches more closely with simulation. In fact, the magnitude of jitter for the main digital signal process-



(a) Mixed hardware-software PWAKF


Software
(low-criticality tasks; exception handling)
Hardware
A_k = coeff_ram[i]
B_k= coeff_ram[i]
$X_{k+1} = A_k x_k + B_k u_k$
$Y_k = C_k x_k + D_k u_k$
Calculate error covariance
Calculate gain
Update State
Update Covariance

(b) Hardware-Only PWAKF

Figure 4.1: Algorithm Refactoring for Fully Hardware-Based PWAKF

ing component of control loop is reduced to the jitter on the system oscillator (picosecond-scale). By contrast, the degree of jitter in commercially available real-time controllers is typically ranging from hundreds of microseconds to tens of microseconds [17]. Note that we do not propose eliminating the software processor altogether, as it is needed to process lower-criticality tasks and handle exceptional conditions in the hardware.

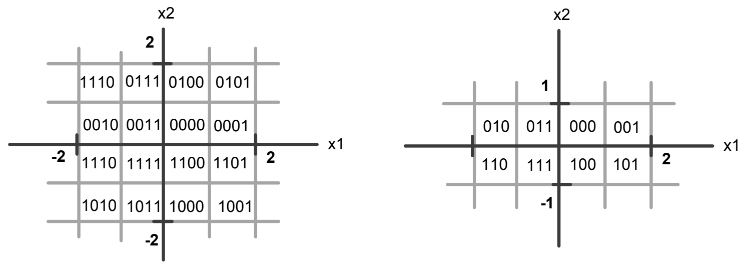
In this chapter we describe the concepts and supporting hardware needed to transition the mixed hardware-software PWAKF architecture from the previous chapter to a fully hardware-oriented design which retains the expressiveness of piecewise-affine modeling. This transition is illustrated in Fig. 4.1, in which (a) depicts the architecture resulting from the previous chapter. Meanwhile, (b) depicts the result of the current chapter. Note that the software box is empty– the software processor is still present, but the resources used to compute the software portion of (a) can be reallocated to the other tasks which the system must perform (e.g. non-critical or non-timing sensitive tasks).

4.2 Point Location Algorithm

We recall the basic mathematical representation of a piecewise-defined linear model from Equation 3.6. The key challenge in implementing this piecewise-defined modeling approach can be termed as the "point location problem", a term which arises in computational geometry [64]. That is, if we interpret the current state vector x_k as point $(x_1, x_2, ...x_n)$ in n-dimensional geometric space, we would like to determine i, the unique index of the region to which point x_k belongs. Once we have i, we have access to the A,B,C and D matrices which describe the region's linear geometry, and we can evaluate the linear state-space function in the normal way to obtain x_{k+1} or y.

Discussion of hardware-based implementations of piecewise-affine functions appears in [65], using an in-hardware transformation step to allow non-uniformly sized regions. More implementations appears in [60] and [66], including comparisons of several ways to partition a multi-dimensional function into regions. The work in [60] demonstrates how a hyperrectangular partitioning approach which allows rectangles of differing size ("multi-resolution" hyperrectangles) leads to a more memory-efficient implementation, since regions which change slowly can be represented by large hyperrectangles and regions which change quickly by small ones. The FPGA-based implementation uses a search tree methodology to enable a fast solution to the point location problem. In general, the main difference among implementations concerns the targeted function partitioning scheme. The partitioning scheme directly impacts the hardware resources and memory resources required to implement an arbitrary function. Region shapes include polytopes, simplicies, hyperrectangles, and other minor variants.

The multi-resolution method in [60] effectively exchanges FPGA memory for FPGA reconfigurable resources, as the number of LUTs and Flipflops increases with the complexity of the model. Furthermore, the number of clock cycles needed to identify region i is not constant due to the search tree structure. This introduces a small amount of timing uncertainty into or model of computation, which we wish to avoid. In this work we uses single-resolution hyperrectangular regions. This scheme is less memory-efficient than the multi-resolution case, but dramatically simplifies the hardware design and ensures a constant time $O(1)$ solution to the point-location



(a) Partition with $t_1 = t_2 = 2$ (b) Partition with $t_1 = 2, t_2 = 1$

Figure 4.2: State-space partitioning for a system with two state variables.

problem. In contrast to earlier work in the area, the input to our hardware functional unit is in floating-point representation to allow easy integration with our existing floating-point based pipeline discussed in earlier chapters. This requires a straightforward float-to-integer step as appearing in Algorithm 3.

Consider the partitioning scheme for a two-state model illustrated in Fig. 4.2 which uses a two's-complement binary value to identify each region. We define symbol t_d to describe the number of bits allocated to uniquely identify each region for a particular dimension (or axis) d of n-dimensional space. Thus for a particular value of t_d there are $r_d = 2^{(t_d)}$ partitions in a given dimension.

For Fig. 4.2a, the address used to locate the active coefficient set will contain two bits for the x_1 state and two bits for the x_2 state. For Fig. 4.2b, the address will have two bits for the x_1 state and one bits for the x_2 state. If $t_d = 0$, this indicates a degenerate case in which state variable x_d is mapped to a domain consisting of a single region: no bits are used from x_d to produce the address.

In this approach we allow each dimension to have different value of t_d. If we were to require them to be the same–e.g. r_d partitions in all dimensions–the storage requirements becomes $r_d^n n^2$ which is highly inefficient, especially for models which are only non-linear on one or a few dimensions. Thus, avoiding this restriction the regions become *hyperrectangles* rather than hypercubes. We do place some constrains on the model and the partition in order to simplify the hardware implementation. Given r_d partitions for dimension d, the state variable x_d has a domain in $r_d/2 \leq x_d \leq r_d/2$.

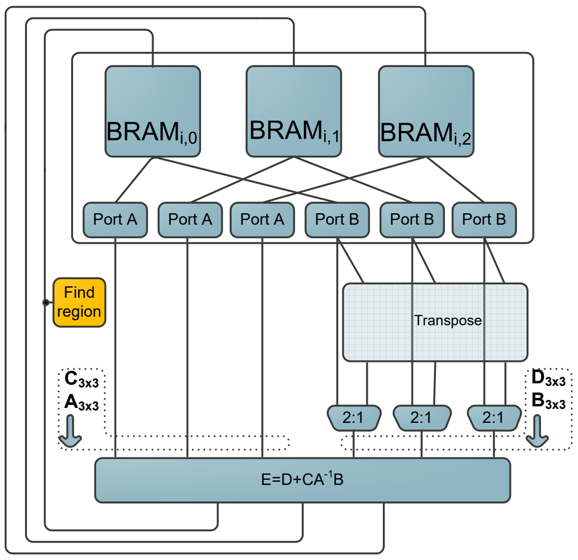
The algorithm for solving the point location problem therefore becomes quite simple. Whereas [60] effectively concatenates input bits based on values of t_i known at synthesis time, Algorithm 3 uses a "sliding window" using multiplexers to construct the base address in the coefficient memory based on runtime values of t_i. This runtime-variable approach may be of interest for rapid prototyping; for IP block generation, it makes more sense to hardcode the various problem parameters and thereby produce a more resource-optimized implementation. In either case, this approach is directly analogous to how a software system computes an address for an n-dimensional array which is arranged contiguously in memory.

Algorithm 3 Solution to point location problem

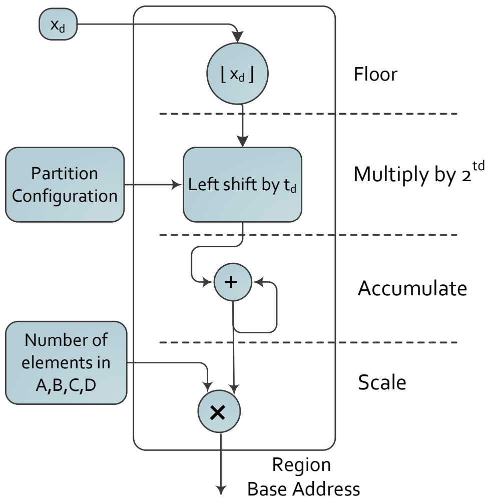
```
1:  procedure FindRegion
2:      $out \leftarrow 0$
3:      $lb \leftarrow 0$
4:      $acc \leftarrow 0$
5:      for $i \leftarrow 0$ ... n do
6:          $xf \leftarrow floor(state[i])$
7:          $acc \leftarrow acc + t[i] - 1$
8:          $ub \leftarrow lb + acc$
9:          $out(ub : lb) \leftarrow xf(ub : lb)$
10:         $lb \leftarrow ub$
11:     $baseaddr \leftarrow out * sizeof_model$
```

The hardware which supports this is shown in Fig. 4.3. The pipeline in Fig. 4.3b implements the concept appearing in Algorithm 3, while Fig. 4.3a illustrates how this functional unit integrates into the broader architecture. It is only necessary to instantiate a single such functional unit because we have, by convention, left-aligned column vectors within their respective memory block. That is, when the final updated state vector x_{k+1} emerges from the Fadeev systolic array, it will consist of a column vector emerging sequentially, word-by-word, from the left-most output bus. Our new functional unit need only monitor this bus, and feed the new region base address back to the central controlling state machine.

In summary, the process of mapping a nonlinear function into the piecewise-affine Kalman Filter is as follows.



(a) Modification to support point location problem.



(b) Pipeline of "FindRegion" functional unit.

Figure 4.3: Hardware Solution to Point Location Problem

1. Determine an appropriate number of partitions t_d per state-space dimension. This is dependent on the on the structure of the model. The choice represents a tradeoff between overall fitting error and the memory required to store the resulting coefficient sets.

2. Map the original function domain onto the new indexed domain using a transformation factor T_f. For example, let $t_d = 2$: if $f(x_d) = sin(x_d)$,$-\pi \leq x_d \leq \pi$, then using $T_f = 2\pi/8$ we can create a function with transformed domain $f'(x_d) = sin(T_f x_d)$, $-4 \leq x_d \leq 4$. Doing this offline reduces hardware complexity.

3. Determine the coefficients for each region. For a nonlinear function $f(x_1, x_2, ..., x_n)$ we need to develop an approximation $\hat{f}(x_1, x_2, ..., x_n) = m_1x_1 + m_2x_2 + ... + m_nx_n + b$. This process is detailed in the following section.

4. Reform the coefficients into our standard A, B, C and D matrices. There will be one set per hyperrectangle.

5. If it has not been done so, transform the model from continuous-time domain to the discrete time domain (e.g. using zero-order hold) with a sampling period equal to the desired sample period of the Kalman Filter.

6. Map the set of A, B, C and D matrices into a 1-dimensional memory array. Careful memory mapping enables a constant-time solution to the point location problem (Algorithm 3). More specifically, the coordinate $(x_1, x_2, ..., x_n)$ which yields the coefficients for region i defines the memory address for that region: the least-significant t_d bits of each component x_d are concatenated to form a globally unique address.

4.3 Piecewise Affine Conversion Algorithm

The "point-to-point" method is normally only discussed in the context of simple 2- or 3-dimensional functions. In this section we describe the model conversion scheme in the most general way–that is, for n-dimensional space–in order to encompass the broadest possible set of models. One of the features of this approach is that the application designer has exact control over how many partitions are generated for each dimension of a multidimensional model, and therefore can directly make tradeoffs between overall accuracy and memory consumption.

We start with a continuous-time model, such as that which is developed through fundamental differential equations.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \mathbf{f}(x_k, u_k) = \begin{bmatrix} f_1(x,u) \\ f_2(x,u) \\ \vdots \\ f_n(x,u) \end{bmatrix} \tag{4.1}$$

We wish, through some procedure, to transform this expression into a set of state-space models, each one having the following continuous-time piecewise-affine form.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} & a_1 \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{n,1} & A_{n,2} & \dots & A_{n,n} & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} + \begin{bmatrix} B_{1,1} & B_{1,2} & \dots & B_{1,n} & b_1 \\ B_{2,1} & A_{2,2} & \dots & B_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ B_{n,1} & B_{n,2} & \dots & B_{n,n} & b_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \\ 1 \end{bmatrix} \tag{4.2}$$

We make an observation to simply discussion. Equation 4.2 can be simplified to the following using block matrices.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \left[\begin{array}{c|c|c|c} A_{n\times n} & a_{n\times 1} & B_{n\times m} & b_{n\times 1} \end{array} \right] \begin{bmatrix} x_{n\times 1} \\ 1 \\ u_{m\times 1} \\ 1 \end{bmatrix} \tag{4.3}$$

Therefore we may target, without loss of generality, the following much-simplified form.

$$\dot{x} = M_i x \tag{4.4}$$

Discussion up to this point applies identically to the output expression (e.g. $\mathbf{g}(x, u)$, and the corresponding C and D matrices). We can now divide the algorithm into two main tasks.

1. Sample the nonlinear vector-valued function $\mathbf{f}(x, u)$ (that is, evaluate each $f_j(x, u), \forall j = 1, 2, \ldots, n$) at integral intervals while applying a selected (invertible) scaling factor T_f to each x. These points should cover the complete domain of interest.

2. For each of the n functions, construct an n-dimensional plane using each sample point and its n neighbors. The equation of each plane contains the coefficients needed to produce M_i.

To reduce notation, we define a set of functions f'_j through which each of the domain scaling factors T_f can be applied to each nonlinear state transformation function f_j.

$$\begin{gathered} f'_j(x_1, x_2, \ldots, x_n) = f_j(T_{f1}x_1, T_{f2}x_2, \ldots, T_{fn}x_n) \\ \forall j = 1, 2, \ldots, n \end{gathered} \tag{4.5}$$

Then, given a particular sample point at coordinates $(x_1, x_2, \ldots, x_n)$, determining the set of corresponding n-dimensional planes consists of solving the set of equalities described by Equation 4.6 by using the point's n direct neighbors. Each solution will consist of a set of slope terms m and an intercept term b. As an example, Matlab can be used to solve each equality by employing the *null* function.

$$\begin{bmatrix} x_1 & x_2 & \dots & x_{n-1} & x_n & f'_j(x_1, x_2, \dots, x_n) & 1 \\ (x_1+1) & x_2 & \dots & x_{n-1} & x_n & f'_j((x_1+1), x_2, \dots, x_n) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \dots & (x_{n-1}+1) & x_n & f'_j(x_1, x_2, \dots, (x_{n-1}+1), x_n) & 1 \\ x_1 & x_2 & \dots & x_{n-1} & (x_n+1) & f'_j(x_1, x_2, \dots, (x_n+1)) & 1 \end{bmatrix} \begin{bmatrix} m_{j,1} \\ m_{j,2} \\ \vdots \\ m_{j,n+1} \\ b_j \end{bmatrix} = 0$$
$$\forall j = 1, 2, \dots, n \tag{4.6}$$

The solutions to these equalities are collected to form the hyperplane expressions.

$$m_{j,1}x_1 + m_{j,2}x_2 + \ldots + m_{j,n}x_n + (m_{j,n+1})f'_j(x_1, x_2, \dots, x_n) + b_j = 0$$
$$\forall j = 1, 2, \dots, n \tag{4.7}$$

We refactor the expression to produce the classic affine form.

$$f'_j(x_1, x_2, \dots, x_n) = \frac{m_{j,1}}{-(m_{j,n+1})}x_1 + \frac{m_{j,2}}{-(m_{j,n+1})}x_2 + \ldots + \frac{m_{j,n}}{-(m_{j,n+1})}x_n + \frac{b_j}{-(m_{j,n+1})}$$
$$\forall j = 1, 2, \dots, n \tag{4.8}$$

Finally, we may populate Equation 4.4 with the computed coefficients for region with index i.

$$\dot{x} = M_i x = \begin{bmatrix} \frac{m_{1,1}}{-(m_{1,n+1})} & \frac{m_{1,2}}{-(m_{1,n+1})} & \dots & \frac{m_{1,n}}{-(m_{1,n+1})} & \frac{b_1}{-(m_{1,n+1})} \\ \frac{m_{2,1}}{-(m_{2,n+1})} & \frac{m_{2,2}}{-(m_{2,n+1})} & \dots & \frac{m_{2,n}}{-(m_{2,n+1})} & \frac{b_2}{-(m_{2,n+1})} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{m_{n,1}}{-(m_{n,n+1})} & \frac{m_{n,2}}{-(m_{n,n+1})} & \dots & \frac{m_{n,n}}{-(m_{n,n+1})} & \frac{b_n}{-(m_{n,n+1})} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} \tag{4.9}$$

This procedure is replicated for each sample point $(x_1, x_2, \dots, x_n)$ in the transformed domain of the vector-valued function $\mathbf{f}$(x,u). Since region boundaries fall on integer-value coordinates, each sampled coordinate yields a unique coefficient matrix described by Equation 4.9. Once the continuous-time matrices are computed, they can be converted into discrete time through usual means (e.g. zero-order hold).

4.4 Implementation Results

4.4.1 Memory Requirements

Most of the schemes described in this dissertation make tradeoffs between memory and computation. Therefore it is necessary to analyze the overall memory requirements for the fully hardware-based solution described in this Chapter. The memory required to represent the complete state-space model in the hypercube case M_{hc} (that is, requiring $t_1 = t_2 = ... = t_n = t$) is shown in Equation 4.10. The symbols n,m, and p are as originally defined in Section 2.5.1 and allow us to compute the number of words needed for a single state-space region. The symbol M_k is the additional "overhead" memory for the various Kalman Filter matrices (e.g. K, P, Q, R, matrix scratch space, etc). We let $M_k = 10n^2$ based on our implementation.

$$M_{hc} = 2^{nt}(n^2 + nm + np + mp) + M_k \tag{4.10}$$

Meanwhile the memory required for hyperrectangles M_{hr} is shown below.

$$M_{hr} = 2^{(\sum_{d=1}^{n} t_d)}(n^2 + nm + np + mp) + M_k \tag{4.11}$$

The latter case will always consume less or equal memory than the former; more importantly it will consume much less memory in the average application. More realistically, however, it must be recalled that the Fadeev algorithm operates on square matrices. Therefore without additional architectural optimization, each matrix will consume an equal-sized cell of size n^2, with many zeroed memory locations for simple models. The term $4n^2$ in Equation 4.12 assumes that the full A,B,C and D matrices are included. In most typical cases there is no need to reserve space for the D matrix; it is factored in an effort to identify upper bounds.

$$M_{hr} = 2^{(\sum_{d=1}^{n} t_d)}4n^2 + M_k \tag{4.12}$$

In Fig. 4.4, Equation 4.12 is applied using the total available block RAM on two available Xilinx FPGAs [67, 68] to determine the maximum number of partitions which can fit given a particular size state-space model n. The Spartan 6 in (a) will be used as a test-bed for the examples in the next chapter, and the Zynq in (b) is the same as that used in the previous chapter. For analysis purposes we constrain $m = p = 1$ (e.g. single-input single-output system) since n is the dominating term.

The results of the analysis indicate that rather complex models can be implemented depending on the amount of block RAM available on the selected chip. For example, the selected Zynq can contain 8192 regions for a 12-state model whereas the much smaller Spartan 6 can only contain 256 regions for a 12-state model. Within both the Zynq and Spartan 6 product families there are a number of model numbers with significantly more block RAM, so these results merely provide a reference point. The application designer can use the results in this section to estimate the required block RAM needed to support a given application.

4.4.2 Performance

With the modification described in this chapter, the overall runtime of the hardware is analytically determined by Equation 3.18 from the previous chapter. The software workload has been eliminated, as all computation is performed in the hardware. The elimination of the software and communication portion from the algorithm does contribute to a small additional speedup vs. the reference software implementation described in the previous chapter (typically 1% faster than the mixed hardware-software PWAKF). In Section 3.6.2.2 the transition rate r_t was introduced to parameterize the amount of data which needs to be transferred between the software and hardware components of the algorithm. The performance of the hardware-only PWAKF is therefore equal to the hardware-software PWAKF with $r_t = 0$. The final runtime comparison is shown in Table 4.1.

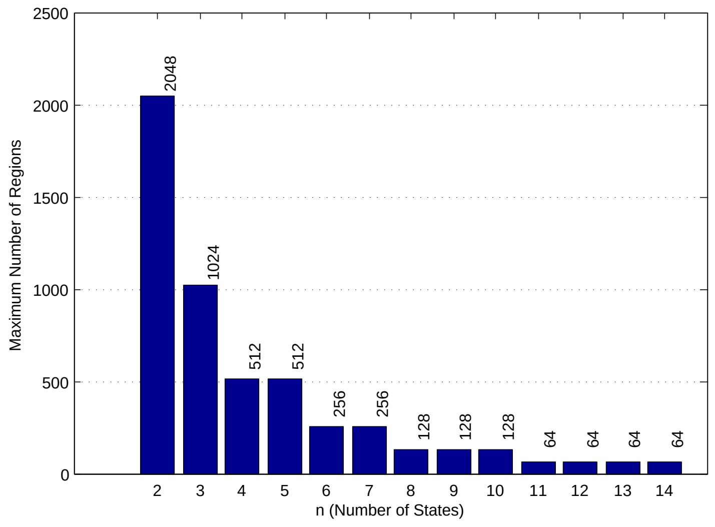
4.5 Conclusion

Our approach is quite simple and generic, and with the on-chip memory it is possible to implement a wide range of models. The additional logic to implement Algorithm 3 consumed just 440 LUTs–for the approach in which partition parameters (t_d) and problem size parameters (n,m,n) are encoded at synthesis time the design would be even smaller. However, for models consisting of a very large number states or requiring many regions to reach the desired state

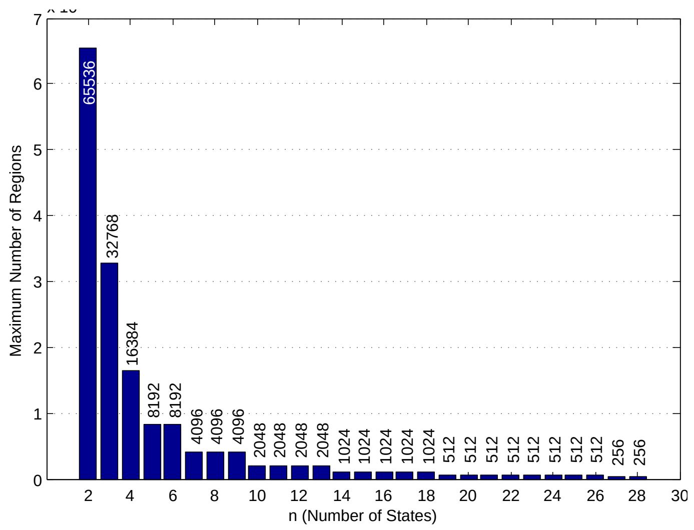
Table 4.1: Hardware PWAKF & Hardware-Software EKF vs. Software EKF

n	EKF SW 200Mhz Time (ms)	PWAKF HW (45Mhz) Time (ms)	PWAKF HW (45Mhz) Speedup	EKF HW-SW (45Mhz) Time (ms)	EKF HW-SW (45Mhz) Speedup
2	0.066	0.004	14.89	0.024	2.78
4	0.082	0.019	4.24	0.039	2.09
6	0.182	0.043	4.26	0.075	2.43
8	0.302	0.075	4.04	0.12	2.42
10	0.500	0.115	4.33	0.19	2.68
12	0.790	0.164	4.80	0.26	2.99
14	1.119	0.222	5.03	0.35	3.15
16	1.568	0.288	5.44	0.46	3.42
18	2.240	0.363	6.17	0.58	3.88
20	2.943	0.446	6.59	0.71	4.17
22	3.737	0.538	6.95	0.85	4.41
24	4.646	0.638	7.28	1.00	4.62
26	5.716	0.747	7.65	1.17	4.86
28	6.914	0.864	8.00	1.36	5.08

tracking performance, on-chip FPGA memory may be exhausted. For example, many partitions may be needed to retain accuracy for fast oscillating behavior of non-constant amplitude. In this case it will be necessary to explore a solution which involves access to off-chip memory. Use of off-chip memory would support models of nearly arbitrary complexity at the expense of higher power consumption and more complex hardware design. The key design tasks for the application developer is determining which dimensions of a multi-dimensional model to partition, and how many partitions to use for these. This requires an application-specific sensitivity analysis (e.g. via simulation) to balance model complexity with the overall tracking performance of the Kalman Filter.



(a) Spartan 6 XC6SLX45



(b) Zynq XC7Z020

Figure 4.4: Estimating Maximum Model Complexity

CHAPTER 5. CASE STUDIES

5.1 Introduction

Up to this point we have characterized the PWAKF from an application-agnostic viewpoint. We now consider three applications-specific case studies in the following sections: a rotating pendulum, a quadrotor helicopter, and a battery monitoring application. The goal of multiple examples is to highlight different aspects of the design process.

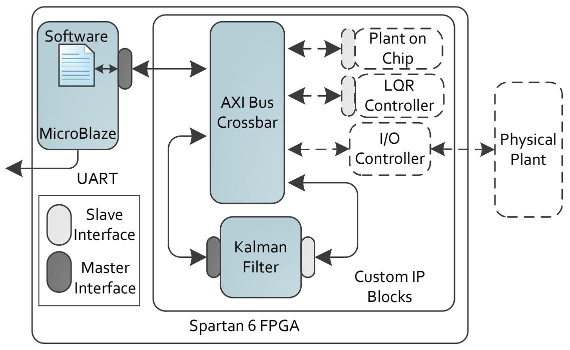


Figure 5.1: Architecture used for Case Studies.

In order to enable power analysis, as described in the next section, we employed Digilent's Atlys board, which features the Spartan 6 XC6SLX45. The XC6SLX45 contains 58 DSP units. Equation 3.17 suggests that if 4 DSPs are allocated per node, the largest size state-space model is 7, with 6 left over for a soft processor. If 1 DSP is allocated per node, the maximum value of n increases, but the operating frequency of the circuit decreases. The XC6SLX45 is one of the smallest variants in the Spartan 6 line; therefore a larger FPGA will support larger models.

The ARM processor from the previous chapters is replaced by a soft processor, shown in Fig. 5.1. Soft processors are becoming more popular for not only computer architecture research, but also within commercial product designs. Xilinx's MicroBlaze RISC soft processor requires 1 cycle for most instructions and supports a maximum frequency of 210Mhz. Use of the soft processor coupled to the PWAKF hardware within the Spartan 6 also highlights the possibility of a single-chip design using a commodity FPGA–that is, one which does not require additional power-hungry external DDR memory or other external I/O bridge components, as do non-SoC processors. Although, with the addition of external DDR memory, Linux can be targeted to the soft processor as well, which makes the support of advanced communication interfaces such as Ethernet much easier. MicroBlaze was configured for performance, and consumed 2366 LUTs, or roughly 9% of the available total.

5.2 Assessing System Power

For the case studies, we wanted the ability to accurately estimate the power consumption of the processor. The Zedboard includes a single shunt resistor for measuring the current of the overall board, but this does not provide the ability to estimate power for the different components of the board. By contrast the Atlys board provides facilities for measuring each of the power rails on the board (3.3V, 2.5V, 1.2V, 1.8V, 0.9V) with accuracy better than 1%. We are most interested in the 1.2V rail which powers the FPGA core logic. This is also the power rail which would be directly influenced by the problem size scaling considered in this work. Power related to I/O is negligible in these tests as we do not make use of external peripherals. Digilent's Adept software [69] is used to retrieve current and power readings from the board. The power analysis results follow.

- Intel Pentium M: 20.8W to 35W TDP [70].
- Intel Atom: 1.3W to 8.5W TDP [71].
- ARM A9 (within Zynq SoC) : 0.6-1.5W
- Spartan 6 (+MicroBlaze): 0.315W

The referenced Intel processors are common for embedded platforms where performance is critical. It is important to consider that the Intel processors in question do not contain internal memory and require additional integrated circuits to interface with basic peripheral devices.

Therefore, Thermal Design Power (TDP) figures tend to under-estimate the total power in a practical implementation.

The power consumption of the Spartan 6 was found to vary negligibly with design size (less than 0.010W across all design sizes which fit in the device) and therefore the figure represents an average. The power consumption appeared much more strongly impacted by the choice of operating frequency. In short, the conclusion is that it is possible to maintain high Kalman Filter performance at a level of power consumption which is 2 to 100 times lower than typical processors, which in mobile applications can lead to longer battery runtime or enable the use of smaller, lighter batteries.

5.3 Rotating Pendulum

Models based on the motion of a pendulum are often used as examples in control theory literature. The motion is based on elementary physical laws and is relatively easy to test in a laboratory setting. The plant discussed in this example consists of a electric servo motor with a weighted rod attached. Much more complex pendulum arrangements exist; for example a computer-based controller was designed in [72] to stabilize a triple inverted pendulum, analogous to a model of a human standing on one leg.

5.3.1 Plant Model

The model of a rotating pendulum originally appearing in [73] is illustrated in Fig. 5.2. This example is used to highlight the design process targeting the PWAKF from the control engineer's perspective. The differential equations which describe the plant motion are reproduced in Equation 5.1.

$$\begin{aligned} \ddot{\theta} &= -\frac{B_\theta}{ml^2 + J_m}\dot{\theta} + \frac{K_2}{ml^2 + J_m} I_\alpha - \frac{mgl}{ml^2 + J_m} sin(\theta) \\ \dot{I}_\alpha &= -\frac{R_\alpha}{L_\alpha} I_\alpha - \frac{K_1}{L_\alpha}\dot{\theta} - \frac{1}{L_\alpha} u \end{aligned} \tag{5.1}$$

This model contains a number of constants which are derived from characterization data obtained from the physical plant. For designing a Kalman Filter to track the state of the pendulum, the measurement uncertainty (statistical variance of the noise) which accompanies

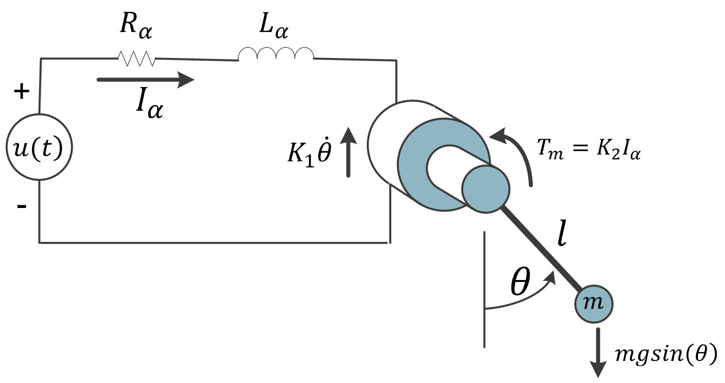


Figure 5.2: Model of rotating pendulum

this characterization procedure can be used to guide the selection for the system noise covariance matrix Q, and the measurement uncertainty which characterizes the position sensor of the pendulum will guide the selection of the measurement noise covariance matrix R (see Algorithm 2).

The expressions of Equation 5.1 can be refactored into a more explicit three state format as follows. Specific constants are obtained from [73] and will not be derived. We let $x_1 = \theta$, the pendulum angle, $x_1 = \dot{\theta}$, the the pendulum angular velocity, and $x_3 = I_\alpha$, the motor current.

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -40.69sin(x_1) - 0.1x_2 + 8.996x_3 \\
\dot{x}_3 &= -28.89x_2 - 722.2x_3 + 555.5u \\
y &= x_1
\end{aligned} \tag{5.2}$$

5.3.2 Conversion to Piecewise Affine Model

Examining Equation 5.2 we consider the trigonometric function *sin*, the only nonlinear function in the model. In typical linearization processes this could be replaced by a small-angle approximation. However, here we would like to maintain modeling accuracy over the full domain of the *sin* function and therefore consider the piecewise-affine approach. By replacing the expression $sin(x_1)$ with another of the form $m_i x_1 + b_i$ We can produce a set of matrices to fit the form described by Equation 3.6. Notice the symbols m_i and b_i which describe slope and intercept coefficients unique to each region denoted by index i.

$$\begin{aligned}
&A_i = \begin{bmatrix} 0 & 1 & 0 \\ -40.69m_i & -0.1 & 8.996 \\ 0 & -28.89 & -722.2 \end{bmatrix}, B_i = \begin{bmatrix} 0 & 0 \\ -40.69b_i & 0 \\ 0 & 555.5 \end{bmatrix}, \\
&C_i = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, D_i = \begin{bmatrix} 0 \end{bmatrix} \\
&\quad \forall i = 1, 2, \ldots, 8
\end{aligned} \tag{5.3}$$

We must determine an appropriate number of index bits t_d to allocate to each state-space dimension. Since only the second dimension (state) contains a nonlinearity we select $t_1 = 0$,$t_2 = 3$ and $t_3 = 0$, leading to a state space consisting of 8 regions (ie $q = 8$ as per Equation 3.6).

We must now identify specific values for each $[m_i, b_i]$ pairing, and do so in a way which remaps the *sin* function onto our new indexed domain. In general, for a nonlinear function $f(x_1, x_2, ..., x_n)$ we need to develop an approximation $\hat{f}(x_1, x_2, ..., x_n) = m_1x_1 + m_2x_2 + ... + m_nx_n + b$. For a function of a single variable this leads to a set of line segments.

A critical divergence from the work in [73] is the domain transformation leading to integral breakpoints at region boundaries. This is the required scheme for targeting the architecture described in Section 4.2. If $f(x_2) = sin(x_2)$,$-\pi \leq x_2 \leq \pi$, then we can create a function with transformed domain $f'(x_2) = sin(2\pi x_2/8)$, $-4 \leq x_2 \leq 4$. Thus our domain transformation factor is $2\pi/8$. To determine the coefficients b_i and m_i for each region we can follow the generalized procedure from Section 4.3. However, this example is quite simple so we need only evaluate $y = sin(x)$ at $x = -4, -3, \ldots, 2, 3$ and determine the equation of the lines between each (x, y) pairs. For software-based simulations, the simple Matlab example in Listing 5.1 demonstrates one method, where the structure *memory* contains the state-space model for each of the 8 regions. Wrapping on the edges can also be handled here.

```
function [ss] = findregion(x2)
i=0;
if( -4<= x2 && x2 <-3)
    i=1;
elseif( -3<= x2 && x2 <-2)
    i=2;
elseif( -2<= x2 && x2 <-1)
    i=3;
elseif( -1<= x2 && x2 <0)
    i=4;
elseif( 0<= x2 && x2 <1)
    i=5;
elseif( 1<= x2 && x2 <2)
    i=6;
elseif( 2<= x2 && x2 <3)
    i=7;
elseif( 3<= x2 && x2 <=4)
    i=8;
end
ss = memory(i).ss;
```

Listing 5.1: “FindRegion” Matlab example

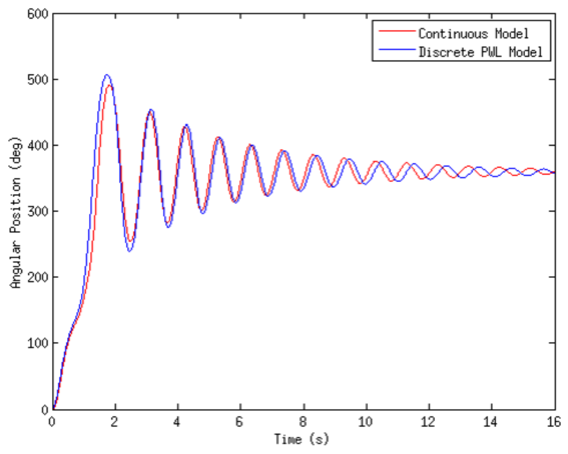
For loading into hardware, the A, B, C and D matrices are mapped into into a memory array for loading into the FPGA block RAMs. Note that in Table 5.1, region models are not necessarily laid out sequentially in memory when negative values are included in the domain. It is this careful memory mapping which enables a simple solution to the point location problem (Algorithm 3).

5.3.3 Performance

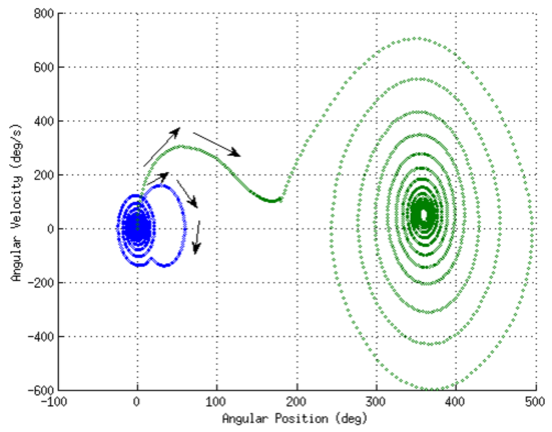
Using Equation 3.18 we know, without the need for benchmarking, that the update time for the hardware-based PWAKF would be 10.8μs, and this figure can be fed into the control engineer's simulation for determining system performance before requiring physical hardware.

Table 5.1: Pendulum Model Memory Arrangement

i	Model Address	m_i	b_i
0	000_b	0.707	0.0
1	001_b	0.293	0.414
2	010_b	-0.293	1.586
3	011_b	-0.707	2.828
-4	100_b	-0.707	-2.828
-3	101_b	-0.293	-1.586
-2	110_b	0.293	-0.414
-1	111_b	0.707	0.0



(a) Comparison of piecewise-linear behavior vs. original differential equations.



(b) Motion through state-space with a small impulse (blue/left) and a large impulse (green/right)

Figure 5.3: Pendulum Motion

Based on Equation 4.12, the estimated memory requirements is 1008 words; therefore the design easily fits into the selected Spartan 6 device.

An interesting point to make is that due to the natural wrapping behavior of binary numbers of limited width, we need only implement regions for one period of the sinusoidal wave, since region traversal will automatically wrap from edge to edge. For example, the region denoted by $i = -5$, which appears to be out of bounds, is implicitly aliased to the region at $i = 3$.

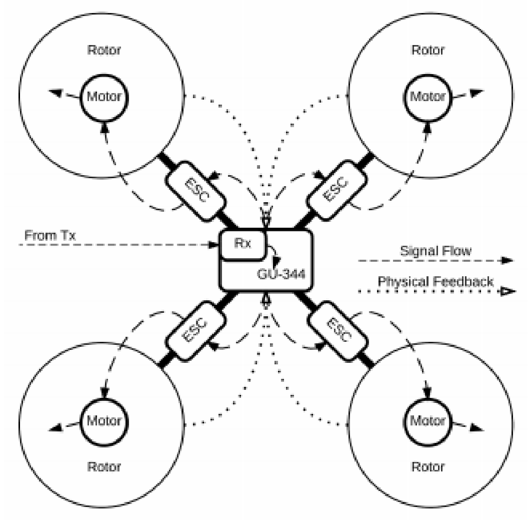


Figure 5.4: Quadrotor helicopter architecture (reproduced from [77]).

5.4 Quadrotor Helicopter

The quadrotor helicopter is an example of a complex electromechanical system consisting of four rotors which relies on an Inertial Navigation System (INS) as a means to maintain flight stability and enable complex aerial maneuvers. These systems often rely on PID based control [74, 75] but increasing interest is vested in a state-space model-based approach. Helicopter flight dynamics are in general very complicated and therefore modeling and simulation is critical in the design process. A general overview of this topic appears in [76].

5.4.1 Plant Model

A methodology was proposed in [77] to characterize the various components of the commercially-available GAUI 330X-S quadrotor helicopter (Fig. 5.4) with a 2100mAh (23.3Wh nominal) battery. A major result consisted of determining an reduced-order 12-state model and using it to develop an LQR controller. The number of states depends on the complexity of the sensor system and the intended degree of maneuverability.

The experimental setup in [77] consists of 12 infrared cameras for motion capture, having a 10ms update rate, which are send to a PC where quaternions are converted to Euler angles in

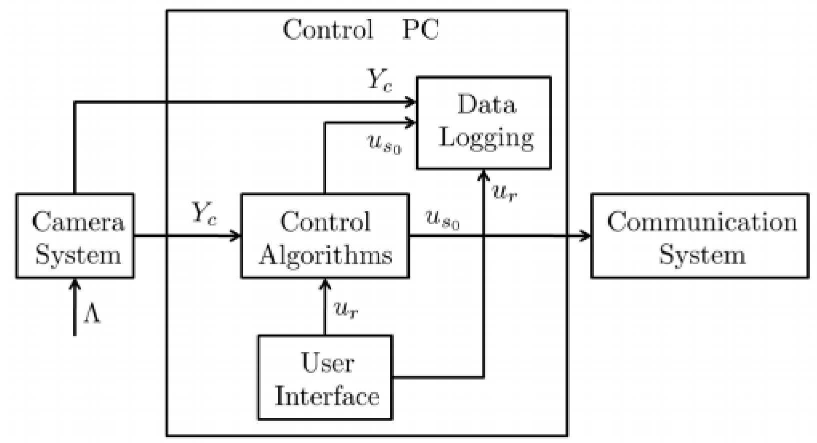


Figure 5.5: Overall control system setup. The control signal u_{S_0} is computed on the PC and then transmitted to the helicopter (reproduced from [77]).

specialized control software. In addition, the control value u_{S_0} is computed on the PC, which includes throttle, aileron, elevator, and rudder values (Fig. 5.5).

In order to execute the control loop on the quadrotor itself, which would enabling more complex autonomous behavior such as optimal path planning [78], the computation should meet the 10ms deadline and also maintain low power usage in order to maximize battery life.

5.4.2 Conversion to Piecewise Affine Model

In the PWAKF approach, the use of a piecewise-defined model is supported, but not required. In this case study we elected to maintain the single-region linear model proposed by the original author–that is, for each model dimension (state) we have $td = 0$ and $q = 1$ as per Equation 3.6.

It is possible to increase the model accuracy by partitioning non-linear behavior into multiple piecewise models. For example, the relationship between battery State of Charge and terminal voltage was modeled in a stateless way with a simple linear fit, due to a lack of sensors on the hardware platform. Otherwise, the accuracy could be significantly enhanced by using a more sophisticated model such as that discussed in Section 5.5. For a fully autonomous platform, accurate assessment of available energy is critical; for example, the study in [79] seeks to select a safe landing space for a quadrotor helicopter based on energy expenditure. That being said, ultimately it would be up to the application developer to determine whether the increased accuracy is beneficial from a control standpoint.

5.4.3 Performance

Based on Equation 3.18 the runtime for a 12 state model using the PWAKF coprocessor is 164μs, which easily meets our 10ms deadline, and based on Equation 4.12 the memory requirements is 2016 words. Although this model fits in memory, if each node consumes 4 DSPs, a 12-state model will not fit into the XC6SLX45. It will however fit into any of the several Spartan 6 devices which contain more DSPs. Adding additional model regions would not impact the runtime, but would impact the memory requirements. Based on Section 5.2 the power consumption of the processor is negligible compared to typical processors used in mobile robotics, and certainly compared to the 4 motors of the helicopter, which are capable of consuming up to a peak 130W each.

5.5 Battery Monitoring

The following section details an extended case study on piecewise-affine Kalman Filtering for tracking cell voltages in large batteries. It is based on results appearing in [80] and [81]. Unlike the previous examples, first-principles battery models rooted in chemical processes are rather difficult to understand by non-domain experts. Therefore, empirical models which rely on general data-fitting tools are overwhelmingly preferred in the engineering community.

5.5.1 Introduction

The advent of rechargeable, high energy density batteries has revolutionized our concepts of portability and mobility. The continued effort to maximize the capacity and life of those batteries enables an ever-expanding array of applications, as well as increasing levels of sophistication. As the demands of the end-application increase, so does the need for a robust battery management system (BMS) to maximize longevity, safety and fault-tolerance.

To guide the actions of any battery management system, typically a model of cell behavior is used as a means to estimate battery State of Charge (SOC) and possibly State of Health

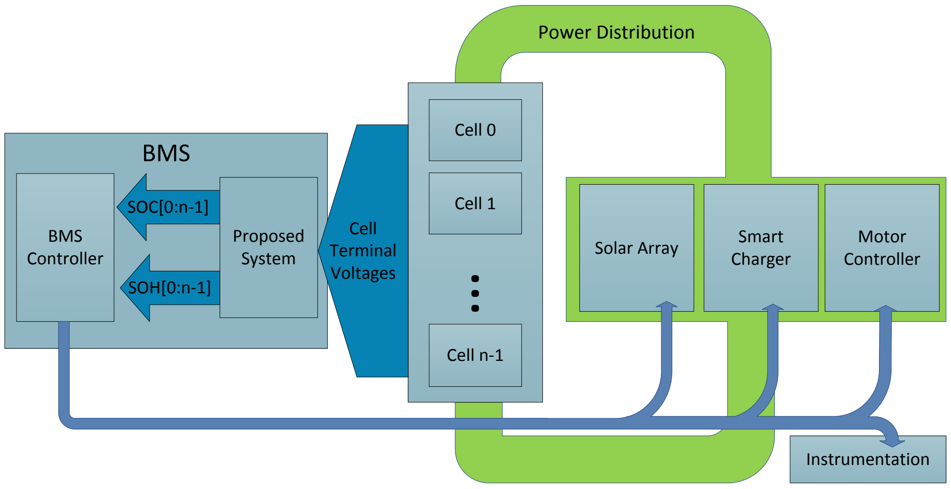


Figure 5.6: The context of the proposed system, including the control feedback signals

(SOH). We use these terms to indicate a 0% to 100% fuel-gauge of the energy contained in the cell, and the current capacity of the cell (based on its age), respectively. For example, a cell could be fully charged (100% SOC), but have severely degraded runtime (low SOH).

However, developing technology to accurately monitor and manage these batteries in applications with heavy or unpredictable energy demands requires knowledge in multiple domains, including material physics, mathematical modeling, statistical methods, and control systems. This complexity is further compounded by the fact that such a system will likely need to maintain a certain level of reliability over potentially years of deployment, during which the characteristics of the battery itself are gradually changing due to manufacturing defects, physical aging, and environmental effects. Thus, monitoring capabilities must be dynamic and adaptive. An generalized illustration of a system powered by a multi-cell battery appears in Fig. 5.6. For electric vehicles, for instance, the number of cells connected in series can exceed 80, with battery currents easily exceeding 200A. Thus great care must be made in the BMS design for such an application.

A number of papers have been published which employ Kalman filters to estimate SOC for rechargeable cells such as Li-ion or LiPo [80, 82, 83, 84]. This method has been shown capable of very accurate cell SOC estimation in real time. The use of the Kalman filter in battery monitoring enables State of Charge estimates which are less sensitive to inaccurate initial cell parameters and are less influenced by long-term drift. In short, by accurately modeling and predicting battery aging effects in-system, using a model such as the one proposed in [85], one can maintain high operational efficiency over the full lifetime of the battery pack. This not only enables more efficient control of cell balancing systems, which addresses the problems of cell parameter mismatch and drift, but also increases the accuracy of the system's SOC and SOH reporting to external entities. Broadly speaking, if system designers can be confident about a battery's SOC to within just a few percent error, and they know that the SOC estimation will maintain tracking even under extremely dynamic loads (e.g.for electric vehicles), they can use the available energy more efficiently and more aggressively.

5.5.2 Plant Model

The ability for model-based SOC and SOH tracking depends to a great extent on developing an accurate behavioral model of the rechargeable cell. There are a number of ways to model a cell. Some approaches such as [2, 86] rely on a deep analysis of the physical and chemical properties of the cell in order to derive a model of its high-level behavior. Despite the high degree of accuracy, typically reduced-order versions of a chemical models must be identified in an order to reduce the computational complexity to a level which can be handled in online applications[87]. Broader coverage of estimation methods appear in [88].

An often-overlooked aspect of SOC estimation is that accurately estimating the SOC for an entire *battery*, even with a highly accurate cell model, cannot be done by simply considering the terminal voltage of that battery. This approach assumes that the constituent cells are always at the same SOC, and also have the same physical cell attributes, such as capacity or internal resistance. Research on model-building has shown this is not the case [89, 90]. This approach is especially ineffective when one wants to accurately predict available power, as over- or under-estimates near the top or bottom of the SOC range could result in some cells being forced outside their safe operational ranges. The unfortunate implication (from a system complexity standpoint) is that the state of each cell in a battery must be considered in some way.

The most straightforward method to estimate SOC/SOH for individual cells in a battery is to develop a model of an "average" cell, and deploy one or more Kalman Filter which use this model to track the states of a set of cells. This allows research on single-cell methods to

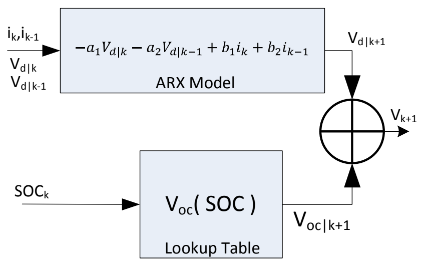


Figure 5.7: Cell model consisting of a linear ARX filter and a non-linear (chemistry-dependent) characteristic curve.

directly translate to a multi-cell approach. A black-box view of the proposed system context is shown in Fig. 5.6. The per-cell information can then be fed to battery protection logic, cell balancing logic, or summarized for instrumentation.

In this work, we have opted for a more abstract, system-identification driven approach rather than a principles-first physical approach, as the former is generally less complex, and easier to generalize to other cell chemistries. However, for our PWAKF solution, any model which can be represented in state-space notation can be utilized. The model used for the case study is a that of a rechargeable cell (Fig. 5.7) and is based on prior work on the subject [81]. It is only non-linear in one dimension, and that non-linearity must be identified empirically; thus it is well-suited for piecewise-affine approximation, using a simple point-to-point partitioning procedure similar to that described in [61]. As for cell characterization procedures, more details appear in [91].

5.5.2.1 State Update Expression

The discrete-time state-space representation will only be summarized here. The structure of the model consists of two superimposed behaviors: an autoregressive exogenous (ARX) model [92] to describe the dynamic behavior, and an empirically obtained curve to describe the relationship between the cell's State of Charge (SOC: an unmeasurable, hidden state) and its open-circuit voltage V_{oc} (directly measurable).

The state update equations below (ie $f(\hat{x}_k, u_k)$) are refactored so that they can be written in standard state space form. They are fully linear as a result of the ARX filter.

$$\begin{bmatrix} SOC_{k+1} \\ V_{d|k+1} \\ V_{d|k} \\ u_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -a_1 & -a_2 & b_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} SOC_k \\ V_{d|k} \\ V_{d|k-1} \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} 0 & 1/C_k \\ 0 & b_1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ u_k \end{bmatrix} \tag{5.4}$$

The constants appearing in the state update matrices are as follows: the ARX constants a_1, a_2, b_1, and b_2 are extracted from cell cycling data, Δt is our simulation timestep (i.e. sensor sample period), and C_k is the nominal cell capacity. Finally, i_k is the measured battery current.

5.5.2.2 Measurement Expression

The system nonlinearity appears in the measurement expression, a nonlinear function of SOC called V_{oc}.

$$y = g(x_k, u_k) = V_{oc}(SOC_k) + V_{d|k} \tag{5.5}$$

We fit a $n_d = 9^{th}$-order polynomial to the sampled data to obtain a continuous, smooth curve with a sum of squares due to error (SSE) of 5mV.

$$y = g(x_k, u_k) = \sum_{i=1}^{n_d} p_i SOC_k^{n_d - i} + V_{d|k} \tag{5.6}$$

This curve is differentiable which allows it to be use it in the standard EKF procedure.

5.5.3 Conversion to Piecewise Affine Model

The nonlinear relationship which appears in the measurement expression is plotted in Fig. 5.8, which compares a 9^{th}-order polynomial fit to a simple point-to-point linearized method. The piecewise-affine expression for each region is shown below, assuming a simple point-to-point

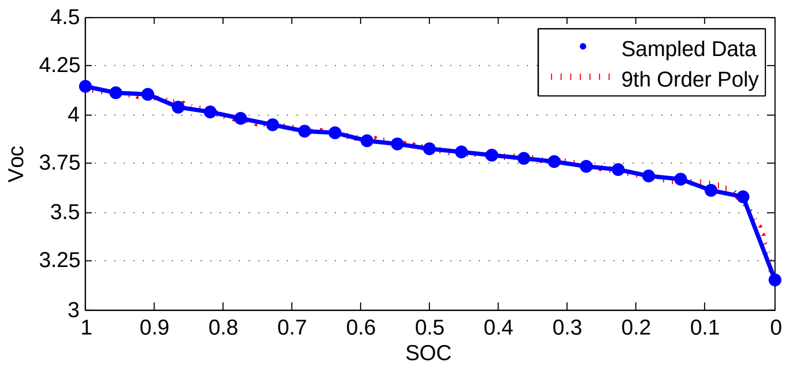


Figure 5.8: Nonlinear relationship between V_{oc} and SOC.

partitioning scheme with $q = 16$ as per Equation 3.6. Four bits are needed to uniquely identify these regions.

$$y = g(x_k, u_k) = \begin{bmatrix} m_i & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} SOC_k \\ V_{d|k} \\ V_{d|k-1} \end{bmatrix} + \begin{bmatrix} b_i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ u \end{bmatrix} \tag{5.7}$$

$$\forall i = 1, 2, \dots, 16$$

Based on this scheme, each region will have a specific value of m_i and b_i which are the slope and y-intercept of the line connecting each pair of sampled data points on the SOC-V_{oc} curve. For $q = 16$, internally SOC is represented by a floating-point value in the range 0 to 16; domain transformation $T_f = 16$ can be used to stretch a $[0, 1]$ domain to $[0, 15]$. In this case, unlike the Pendulum example, wrapping behavior at the edges is likely to be undesirable; instead the "FindRegion" functional unit from Chapter 4 should be configured for saturating behavior.

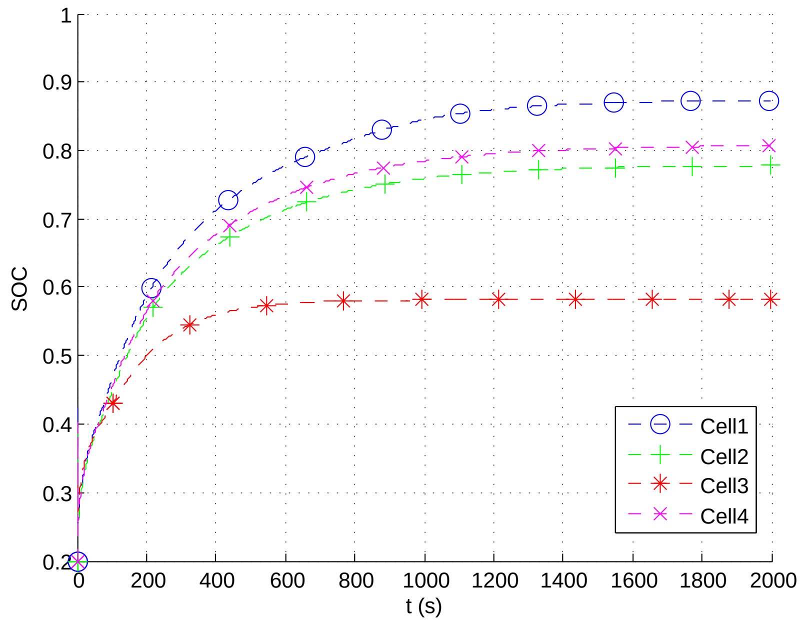
5.5.4 Performance

Based on Equation 3.18 the runtime for a 4 state model using the PWAKF coprocessor is 19.4μs, and based on Equation 4.12 the memory requirements is 3584 words. The most straightforward means of tracking multiple cells is to overlay an additional sequencer which iteratively updates each cell state. This approach does not significantly increase memory consumption, as only the state vector will need to be duplicated for each cell. As an example, a large battery with 80 series-connected cells would require 1.6ms for full-battery state update.

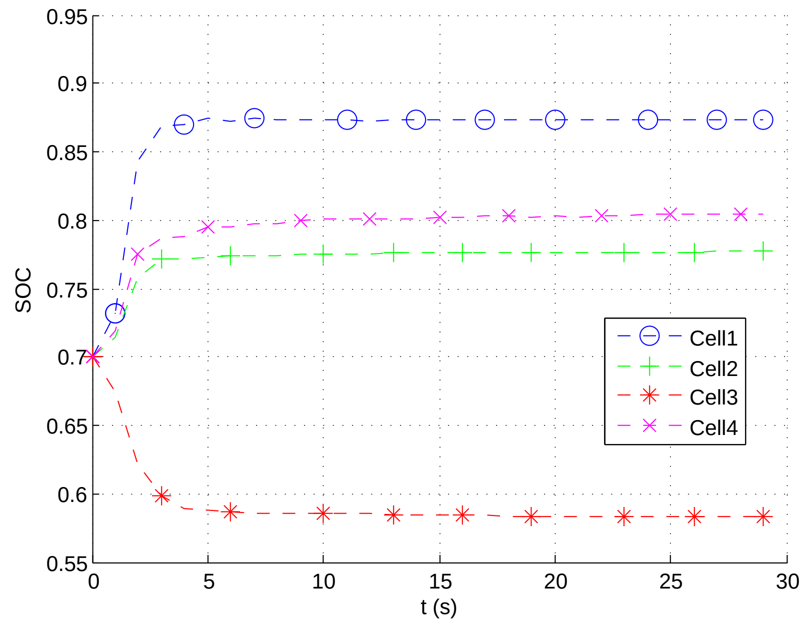
In the discussion that follows, regardless of internal representation we will describe SOC as a real value in the range $[0, 1]$. Thanks to basic Kalman Filter behavior the system state will eventually settle on the true SOC value even when it is initialized poorly. However, if the sample rate is low, for example once per second, poor initialization can drastically increase the delay before the SOC measurement is ready for use by the system. As seen in Fig. 5.9a, when all cells are incorrectly initialized at 0.20 SOC, at 1s sampling, cell 1 takes almost 30 minutes to reach steady-state. The state initialization vector should thus be saved in non-volatile memory periodically and then recovered at system reset to minimize startup delay. As seen in Fig. 5.9b, even initializing the states to 0.70 reduces the time to steady-state to less than 20s. Conveniently, since the Kalman Error Covariance matrix (P_k) is updated each iteration, it is also possible to determine when SOC has been acquired when the uncertainty reduces to a given threshold. The uncertainty (variance) for each state is contained in the diagonals of the matrix. The plot in Fig. 5.10 shows how the uncertainty reduces over the first 10 iterations (seconds) of the algorithm. The bounds represent two standard deviations. Unfortunately, it is possible to make the state "over confident" by underestimating the system noise.

5.6 Multiple Kalman Filters

Some applications may require multiple Kalman Filters to be implemented. It is useful to think of the PWAKF architecture proposed in this work as a single IP Block which may be instantiated more than once times as required, each configured for the characteristics of the model for which it is intended to track. In other cases, the states of each plant in a large array of plants having identical models must be computed–perhaps too large for the resource available on modern FPGAs. For example, the cells in a large battery back will have the same model, but different values for state of charge, state of health, etc at any given moment. For tracking objects in an image, each object will have a position, velocity, acceleration, etc. For these cases it makes sense to simply perform time-multiplexing on the same Kalman Filter processor. Note that this approach does not suffer the same problems as time-multiplexing on a software processor, since each task is processed sequentially in a fixed order and at fixed intervals using a time-deterministic state machine.



(a) Acquisition delay for SOC(0)=0.2



(b) Acquisition delay for SOC(0)=0.7

Figure 5.9: Impact of SOC state initialization over time

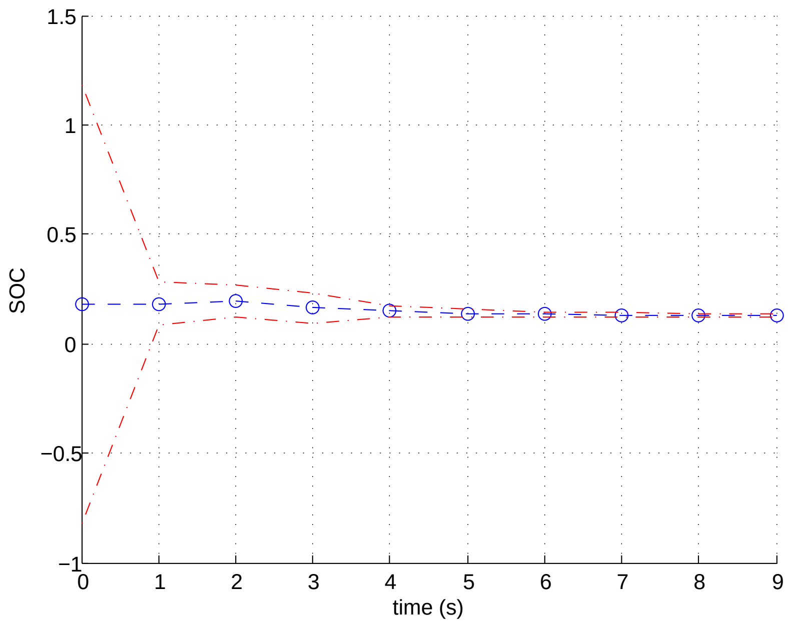


Figure 5.10: Reduction in State Uncertainty Over Time

Another interesting problem arises in applications where one modeling goal is to assess the "health" of a plant while simultaneously monitoring its state. For example, there is interest in monitoring the health of aircraft engines [93, 94]. The goal of monitoring the engine health is to lengthen the lifespan of an aircraft and save money on repairs and replacements. Piece-wise Kalman filters must be designed at different operating conditions–for example, one model is used when the engine is in "good" health, one model when it is in "fair" health, and so on. A similar example appears in [95, 81] which is concerned with assessing the state of health (e.g. degree of degradation from nominal capacity) of rechargeable cells. In both cases, the health parameter varies on a different timescale (i.e. slower) than the state; therefore it should be implemented in a separate filter loop. Since health parameters usually vary at very slow rates (e.g. days, months, years), it makes sense to implement such a loop in software since any delay or jitter will comprise an extremely small percent of the overall sampling period. Fig. 5.11 shows two Kalman Filters running in parallel with an $L - fold$ difference in sample rate. The "micro-scale" filter in this case estimates State of Charge while the "macro-scale" filter estimates State of Health.

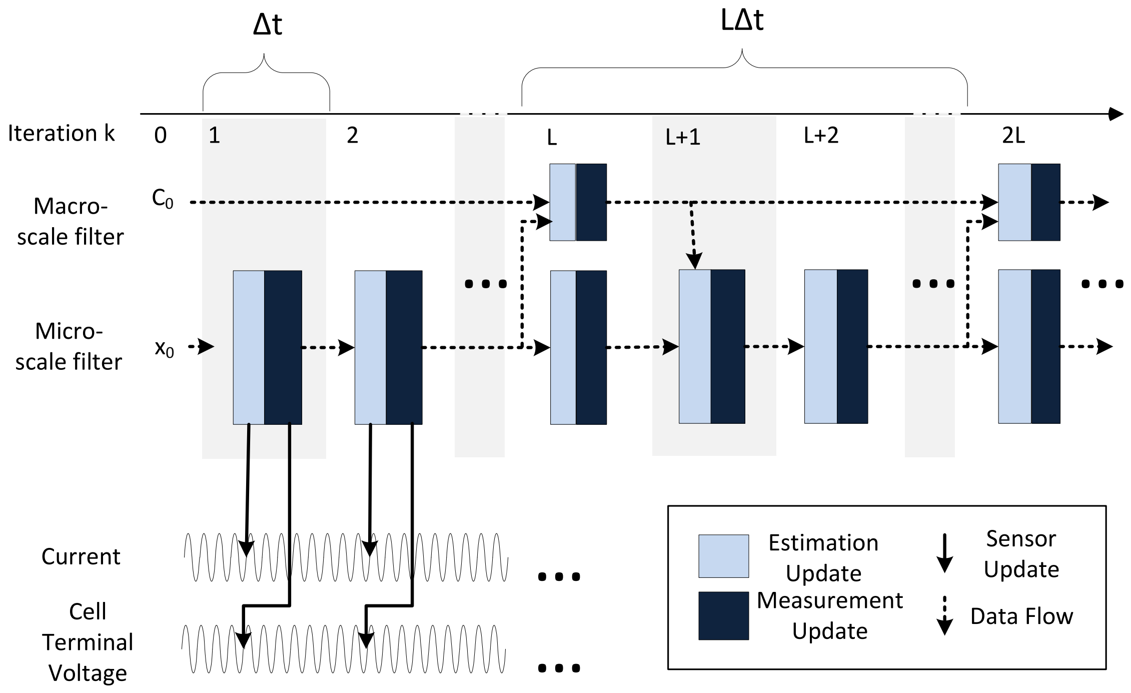


Figure 5.11: Multiscale Kalman Filter

In both examples, the health parameter varies on a different timescale (i.e. slower) than the state; therefore it should be implemented in a separate filter loop. Since health parameters usually vary at very slow rates (e.g. days, months, years), it makes sense to implement such a loop in software since any delay or jitter will comprise an extremely small percent of the overall sampling period.

5.7 Conclusion

In this section we have explored implementation power usage and showed several examples of linear or piecewise-affine defined plant models, along with performance analyses as pertaining to our hardware implementation.

CHAPTER 6. CONCLUSION

6.1 Design Evolution

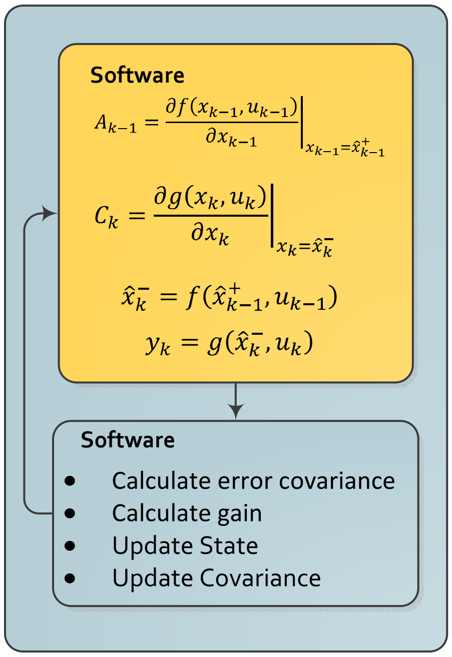
The location of the hardware-software partition for each of the approaches to Kalman Filtering discussed in this work are illustrated in Fig. 6.1.

The software-only Kalman Filter (Fig. 6.1a) is fast to implement, but imposes heavy compute requirements with for embedded hardware. Within any typical multiprocess software processor it will also contribute to stochastic timing characteristics, which complicates analysis for the broader control loop which it may be feeding.

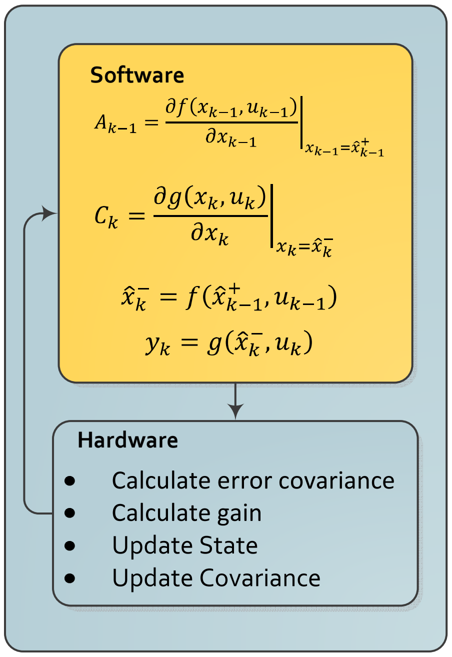
The existing mixed hardware-software approaches to Kalman Filtering (Fig. 6.1b) are effective at speeding up an application, but also exhibit high communication cost for applications requiring fast update times or complex models. This approach also inherits software scheduling issues from the software-only approach, since the application-specific code must still share software processor time with other lower or higher priority tasks.

The mixed hardware-software PWAKF (Fig. 6.1c) approach enables a reduced communication cost, but it still does not completely eliminate scheduling-induced timing issues resulting from shared compute resources. A potential drawback is that additional planning is needed for model partitioning.

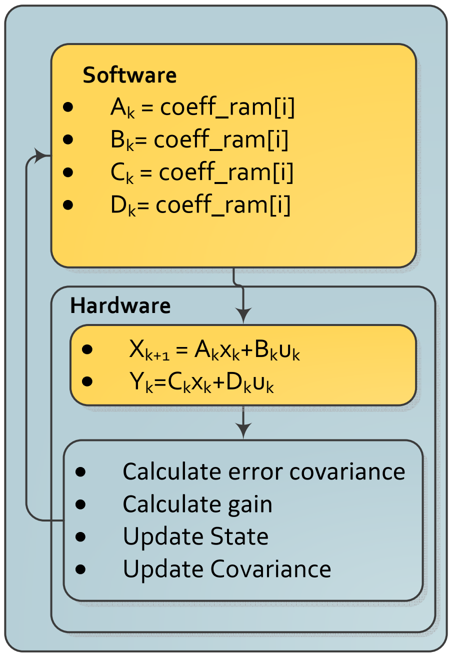
The remaining limitations in the mixed hardware-software PWAKF lead us to the hardware-only PWAKF (Fig. 6.1d), which eliminates coefficient transfers and timing uncertainty. Like the hardware-software PWAKF, applications sensitive to modeling accuracy will require extra planning to implement model partitioning. However, in this case the extra design time may be offset by the fact that less time needs to be spent developing complex computational models and estimating WCET, since timing is guaranteed by design.


Software
$A_{k-1} = \frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\Big|_{x_{k-1}=\hat{x}^+_{k-1}}$
$C_k = \frac{\partial g(x_k, u_k)}{\partial x_k}\Big|_{x_k=\hat{x}^-_k}$
$\hat{x}^-_k = f(\hat{x}^+_{k-1}, u_{k-1})$
$y_k = g(\hat{x}^-_k, u_k)$
Software
Calculate error covariance
Calculate gain
Update State
Update Covariance

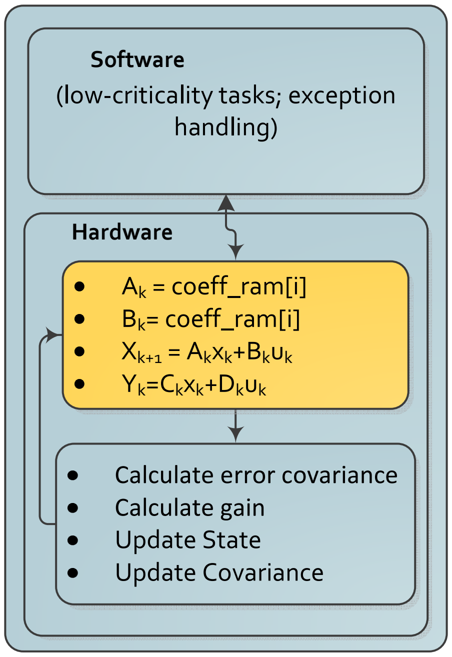
(a) Software Extended Kalman Filter


Software
$A_{k-1} = \frac{\partial f(x_{k-1}, u_{k-1})}{\partial x_{k-1}}\Big|_{x_{k-1}=\hat{x}^+_{k-1}}$
$C_k = \frac{\partial g(x_k, u_k)}{\partial x_k}\Big|_{x_k=\hat{x}^-_k}$
$\hat{x}^-_k = f(\hat{x}^+_{k-1}, u_{k-1})$
$y_k = g(\hat{x}^-_k, u_k)$
Hardware
Calculate error covariance
Calculate gain
Update State
Update Covariance

(b) Hardware-Software Extended Kalman Filter



(c) Hardware-Software PWAKF


Software
(low-criticality tasks; exception handling)
Hardware
A_k = coeff_ram[i]
B_k= coeff_ram[i]
$X_{k+1} = A_k X_k + B_k U_k$
$Y_k = C_k X_k + D_k U_k$
Calculate error covariance
Calculate gain
Update State
Update Covariance

(d) Hardware PWAKF

Figure 6.1: Hardware-Software Partitioning of Kalman Filter. Orange boxes indicate processes with application-specific data.

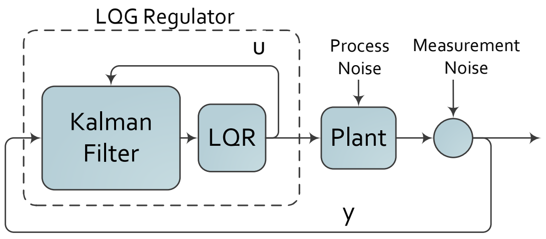


Figure 6.2: LQG Regulator Comprised of Kalman Filter and LQR Regulator

Of course, any such approach constitutes merely one tool in a CPS designer's toolbelt–one which must be selected carefully based on the application requirements. It is hoped that this dissertation has provided the necessary design characterization and workflow examples to assist in such a decision-making process.

6.2 Future Work

There are a number of avenues for future work, several of which relate to relatively minor variations on the architecture. It is possible to obtain additional hardware performance by adding one or more dedicated matrix math units which have lower time complexity than the Fadeev algorithm (e.g. for simple matrix multiplication); from this perspective, the Fadeev algorithm is mainly reserved for the matrix inversion expression in the Kalman Filter algorithm. This modification will also eliminate the Fadeev algorithm's constraint that A,B,C and D must be square matrices, which would reduce memory overhead for simple models. It is also of interest to more deeply compare the efficacy of alternative partitioning schemes. The partitioning scheme impacts hardware and memory resources, and possibly computation time.

Finally, although we focus on applying piecewise modeling approach to the Kalman Filter, it is also possible to extend the method to other state-space based algorithms such as LQR or MPC. For example it is possible to develop a highly accelerated Linear Quadratic Gaussian (LQG) regulator by attaching the input of the LQR controller to the output of the Kalman Filter. This structure is shown Fig. 6.2, and an example design process for a 10-state LQG controller for an aircraft appears in [96]. Last, but not least, we would seek to integrate these techniques into a complete Cyber-Physical System, such as a quadrotor helicopter.

BIBLIOGRAPHY

[1] D. Pritsker, "Hybrid implementation of Extended Kalman Filter on an FPGA," in *IEEE Radar Conference (RadarCon)*, May 2015, pp. 0077–0082.

[2] M. Daigle and C. Kulkarni, "A battery health monitoring framework for planetary rovers," in *Aerospace Conference, 2014 IEEE*, March 2014, pp. 1–9.

[3] G. F. Welch, "History: The use of the kalman filter for human motion tracking in virtual reality," *Presence: Teleoperators and Virtual Environments*, vol. 18, no. 1, pp. 72–91, 2009.

[4] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach, Second Edition.* LeeSeshia.org, 2011.

[5] E. Lee, "Cyber-Physical Systems - Are Computing Foundations Adequate?" *NSF Workshop on Cyber-Physical Systems*, 2006.

[6] R. Gonzalez, G. Sutter, C. Sistema, and H. Patino, "FPGA-based floating-point UD filter coprocessor for integrated navigation systems," in *Argentine Conference on Embedded Systems (CASE)*, Aug 2015, pp. 7–12.

[7] J. A. Rios and E. White, "Fusion filter algorithm enhancements for a MEMS GPS/IMU," *Crossbow Technology, Inc*, pp. 1–12, 2002.

[8] A. Buchan, D. Haldane, and R. Fearing, "Automatic identification of dynamic piecewise affine models for a running robot," in *International Conference on Intelligent Robots and Systems (IROS)*, Nov 2013, pp. 5600–5607.

[9] H. Al-Sheikh, G. Hoblos, and N. Moubayed, “Piecewise switched Kalman filtering for sensor fault diagnosis in a DC/DC power converter,” in *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2015 Third International Conference on*, April 2015, pp. 62–67.

[10] B. Vincke, A. Elouardi, and A. Lambert, “Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems,” *EURASIP Journal on Embedded Systems*, vol. 2012, no. 1, pp. 1–14, 2012.

[11] V. Bonato, E. Marques, and G. Constantinides, “A Floating-Point Extended Kalman Filter Implementation for Autonomous Mobile Robots,” in *International Conference on Field Programmable Logic and Applications*, Aug 2007, pp. 576–579.

[12] A. Burns and R. Davis, “Mixed criticality systems-a review,” *Department of Computer Science, University of York, Tech. Rep*, 2013.

[13] J. C. Bicarregui, J. S. Fitzgerald, P. G. Larsen, and J. Woodcock, “Industrial practice in formal methods: A review,” in *FM 2009: Formal Methods.* Springer, 2009, pp. 810–813.

[14] M. Batra, “Formal methods: Benefits, challenges and future direction,” *Journal of Global Research in Computer Science*, vol. 4, no. 5, pp. 21–25, 2013.

[15] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, “The worst-case execution-time problemoverview of methods and survey of tools,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008.

[16] B. Lincoln, “Jitter compensation in digital control systems,” in *American Control Conference, 2002. Proceedings of the 2002*, vol. 4, 2002, pp. 2985–2990 vol.4.

[17] K. Smeds and X. Lu, “Effect of sampling jitter and control jitter on positioning error in motion control systems,” *Precision Engineering*, vol. 36, no. 2, pp. 175–192, 2012.

[18] C. Kumar, S. Vyas, R. Cytron, C. Gill, J. Zambreno, and P. Jones, "Hardware-Software Architecture for Priority Queue Management in Real-time and Embedded Systems," *International Journal of Embedded Systems (IJES)*, vol. 6, no. 4, pp. 319–334, 2014.

[19] V. Bonato, R. Peron, D. Wolf, J. de Holanda, E. Marques, and J. Cardoso, "An FPGA implementation for a Kalman Filter with application to mobile robotics," in *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, July 2007, pp. 148–155.

[20] E. Monmasson and M. Cirstea, "Guest Editorial Special Section on Industrial Control Applications of FPGAs," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1250–1252, Aug 2013.

[21] A. Mills, P. Zhang, S. Vyas, J. Zambreno, and P. H. Jones, "A software configurable coprocessor-based state-space controller," in *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015, pp. 1–6.

[22] P. Zhang, A. Mills, J. Zambreno, and P. H. Jones, "A software configurable and parallelized coprocessor architecture for lqr control," in *Proceedings of the IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–8.

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

[24] E. Monmasson, L. Idkhajine, and M. W. Naouar, "FPGA-based Controllers," *IEEE Industrial Electronics Magazine*, vol. 5, no. 1, pp. 14–26, March 2011.

[25] 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*, May 2011, pp. 6186–6191.

[26] 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*, vol. 21, no. 7, pp. 1170 – 1182, 2011.

[27] 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.

[28] 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.

[29] 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*, June 2011, pp. 209–218.

[30] T. Ould-Bachir, C. Dufour, J. Belanger, J. Mahseredjian, and J.-P. David, “Effective floating-point calculation engines intended for the fpga-based hil simulation,” in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*, May 2012, pp. 1363–1368.

[31] C. Dufour, S. Cense, and J. Belanger, “An fpga hil reconfigurable testing platform for vehicular traction systems,” in *Vehicle Power and Propulsion Conference (VPPC), 2014 IEEE*, Oct 2014, pp. 1–4.

[32] A. Penczek, R. Stala, Ł. Stawiarski, and M. Szarek, “Hardware-in-the-loop fpga-based simulations of switch-mode converters for research and educational purposes,” *Przeglad Elektrotechniczny (Electrical Review)*, vol. 87, no. 11, pp. 194–200, 2011.

[33] L. Herrera and J. Wang, “FPGA based detailed real-time simulation of power converters and electric machines for EV HIL applications,” in *2013 IEEE Energy Conversion Congress and Exposition*, Sept 2013, pp. 1759–1764.

[34] 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*, vol. 17, no. 5, pp. 1006–1017, Sept 2009.

[35] 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.

[36] 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)*, June 2010, pp. 1436–1441.

[37] 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*, vol. 2, Nov 1996, pp. 895–899 vol.2.

[38] B. Garbergs and B. Sohlberg, “Implementation of a state space controller in a FPGA,” in *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean*, vol. 1, May 1998, pp. 566–569 vol.1.

[39] 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*, vol. 6, no. 1, pp. 4–7, March 2014.

[40] T. Chen, B. Francis, and T. Hagiwara, “Optimal sampled-data control systems,” *Proceedings of the IEEE*, vol. 86, no. 4, pp. 741–741, 1998.

[41] 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)*, March 2014, pp. 1–6.

[42] 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)*, June 2009, pp. 2869–2874.

[43] B. Messner and D. Tilbury, “Inverted Pendulum: System Modeling,” http://ctms.engin.umich.edu/CTMS, University of Michigan, 2012.

[44] R. Van Der Merwe, “Sigma-point Kalman filters for probabilistic inference in dynamic state-space models,” Ph.D. dissertation, Oregon Health & Science University, 2004.

[45] E. Wan and R. Van der Merwe, “The unscented Kalman filter for nonlinear estimation,” in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000, pp. 153–158.

[46] A. Bigdeli, M. Biglari-Abhari, Z. Salcic, and Y. T. Lai, “A New Pipelined Systolic Array-based Architecture for Matrix Inversion in FPGAs with Kalman Filter Case Study,” *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 75–75, Jan. 2006.

[47] A. Sudarsanam, “Analysis of Field Programmable Gate Array-Based Kalman Filter Architectures,” in *All Graduate Theses and Dissertations*, 2010. [Online]. Available: http://digitalcommons.usu.edu/etd/788

[48] A. Mills, P. Jones, and J. Zambreno, “Parameterizable FPGA-based Kalman Filter Coprocessor Using Piecewise Affine Modeling,” in *Proceedings of the Reconfigurable Architectures Workshop (RAW)*, May 2016.

[49] M. Grewal and A. Andrews, *Kalman Filtering: Theory and Practice with MATLAB*. Wiley, 2015.

[50] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering: A Practical Approach*. Virginia: American Institute of Aeronautics and Astronautics, Inc, 2009.

[51] C. Lee and Z. Salcic, “A fully-hardware-type maximum-parallel architecture for Kalman tracking filter in FPGAs,” in *International Conference on Information, Communications and Signal Processing (ICICS)*, Sep 1997, pp. 1243–1247 vol.2.

[52] C. Lee and Z. Salcic, “High-performance FPGA-based implementation of Kalman filter,” *Microprocessors and Microsystems*, vol. 21, no. 4, pp. 257–265, 1997.

[53] F. Gaston and G. Irwin, “Systolic Kalman filtering: an overview,” *Control Theory and Applications*, vol. 137, no. 4, pp. 235–244, Jul 1990.

[54] M. Johansson, *Piecewise Linear Control Systems: A Computational Approach.* Springer, 2002.

[55] J. Xu and L. Xie, *Control and Estimation of Piecewise Affine Systems*, ser. Woodhead Publishing Reviews Mechanical Engineering. Elsevier Science & Technology, 2014.

[56] A. Benine-Neto and C. Grand, "Piecewise affine control for fast unmanned ground vehicles," in *International Conference on Intelligent Robots and Systems (IROS)*, Oct 2012, pp. 3673–3678.

[57] P.-K. Luk, S. Aldhaher, W. Fei, and J. Whidborne, "State-Space Modeling of a Class $\mathbf{e^2}$ Converter for Inductive links," *IEEE Transactions on Power Electronics*, vol. 30, no. 6, pp. 3242–3251, June 2015.

[58] B. Farhangi and H. A. Toliyat, "Piecewise linear modeling of snubberless dual active bridge commutation," in *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sept 2014, pp. 2065–2071.

[59] M. Zajc, R. Sernec, and J. Tasic, "An efficient linear algebra SoC design: implementation considerations," in *Mediterranean Electrotechnical Conference (MELECON)*, 2002, pp. 322–326.

[60] F. Comaschi, B. Genuit, A. Oliveri, W. Heemels, and M. Storace, "FPGA Implementations of Piecewise Affine Functions Based on Multi-Resolution Hyperrectangular Partitions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 12, pp. 2920–2933, Dec 2012.

[61] G. Lowe and M. Zohdy, "A technique for using H_2 and H_∞; robust state estimation on nonlinear systems," in *International Conference on Electro/Information Technology (EIT)*, June 2009, pp. 109–115.

[62] Xilinx, "Xilinx Power Estimator (XPE)," http://www.xilinx.com/products/technology/power/xpe.html, Xilinx, 2016.

[63] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, “Jitter compensation for real-time control systems,” in *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, Dec 2001, pp. 39–48.

[64] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong, *Computational Geometry: Algorithms and Applications, 3rd ed.* Springer-Verlag, 2008.

[65] T. Poggi, F. Comaschi, and M. Storace, “Digital circuit realization of piecewise-affine functions with nonuniform resolution: Theory and fpga implementation,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 2, pp. 131–135, Feb 2010.

[66] M. Storace and T. Poggi, “Digital architectures realizing piecewise-linear multivariate functions: Two FPGA implementations,” *International Journal of Circuit Theory and Applications*, vol. 39, no. 1, pp. 1–15, 2011. [Online]. Available: http://dx.doi.org/10.1002/cta.610

[67] Xilinx, “Spartan-6 Family Overview (DS160),” http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf, Xilinx, 2011.

[68] Xilinx, “Zynq-7000 All Programmable SoC Overview (DS190),” http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, Xilinx, 2016.

[69] Digilent, “Digilent Adept 2,” https://reference.digilentinc.com/digilent_adept_2, Digilent, 2016.

[70] Intel, “Mobile Intel Pentium 4 Processor M Datasheet,” download.intel.com/design/mobile/datashts/25068607.pdf, Intel, 2003.

[71] Intel, “Intel Atom Processsor,” ark.intel.com/products/family/29035/Intel-Atom-Processor, Intel, 2016.

[72] V. A. Tsachouridis, “Robust control of a triple inverted pendulum,” in *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, vol. 2. IEEE, 1999, pp. 1235–1240.

[73] G. Lowe and M. Zohdy, "Modeling nonlinear systems using multiple piecewise linear equations," *Nonlinear Analysis: Modelling and Control*, vol. 15, no. 4, pp. 451–458, 2010.

[74] T. Bresciani, "modeling, identification and control of a quadrotor helicopter, masters thesis," Ph.D. dissertation, Lund University, 2008.

[75] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proc. of the AIAA Guidance, Navigation, and Control Conference*, vol. 2, 2007.

[76] G. D. Padfield, *Helicopter flight dynamics.* John Wiley & Sons, 2008.

[77] M. Rich, "Model development, system identification, and control of a quadrotor helicopter," Ph.D. dissertation, Iowa State University, 2012.

[78] J. F. Whidborne, I. D. Cowling, and A. K. Cooke, "Optimal Trajectory Planning and LQR Control for a Quadrotor UAV," in *Proceedings of the International Conference Control*, vol. 30, 2006.

[79] J. Park, Y. Kim, and S. Kim, "Landing site searching and selection algorithm development using vision system and its application to quadrotor," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 488–503, March 2015.

[80] A. Mills and J. Zambreno, "Towards scalable monitoring and maintenance of rechargeable batteries," in *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT)*, June 2014, pp. 624–629.

[81] A. Mills and J. Zambreno, "Estimating state of charge and state of health of rechargable batteries on a per-cell basis," in *Proceedings of the Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, April 2015, pp. 1–6.

[82] D. Haifeng, W. Xuezhe, and S. Zechang, "State and parameter estimation of a HEV Li-ion battery pack using Adaptive Kalman Filter with a new SOC-OCV concept," in *Measuring Technology and Mechatronics Automation, 2009. ICMTMA '09. International Conference on*, vol. 2, 2009, pp. 375–380.

[83] G. L. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 1. background," *Journal of Power Sources*, vol. 134, no. 2, pp. 252 – 261, 2004.

[84] S. Yuan, H. Wu, and C. Yin, "State of charge estimation using the Extended Kalman Filter for battery management systems based on the ARX battery model," *Energies*, vol. 6, no. 1, pp. 444–470, 2013.

[85] L. Lam and P. Bauer, "Practical capacity fading model for Li-ion battery cells in electric vehicles," *IEEE Transactions on Power Electronics*, vol. 28, no. 12, pp. 5910–5918, 2013.

[86] K. Thirugnanam, H. Saini, and P. Kumar, "Mathematical modeling of Li-ion battery for charge/discharge rate and capacity fading characteristics using genetic algorithm approach," in *Transportation Electrification Conference and Expo (ITEC), 2012 IEEE*, June 2012, pp. 1–6.

[87] S. Dey, B. Ayalew, and P. Pisu, "Nonlinear robust observers for state-of-charge estimation of lithium-ion cells based on a reduced electrochemical model," *Control Systems Technology, IEEE Transactions on*, vol. 23, no. 5, pp. 1935–1942, Sept 2015.

[88] W.-Y. Chang, "The state of charge estimating methods for battery: A review," *ISRN Applied Mathematics*, no. 7, 2013.

[89] D. Shin, M. Poncino, E. Macii, and N. Chang, "A statistical model of cell-to-cell variation in Li-ion batteries for system-level design," in *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 94–99.

[90] M. Dubarry, N. Vuillaume, and B. Y. Liaw, "Origins and accommodation of cell variations in Li-ion battery pack modeling," *International Journal of Energy Research*, vol. 34, no. 2, pp. 216–231, 2010. [Online]. Available: http://dx.doi.org/10.1002/er.1668

[91] Idaho National Engineering and Environmental Laboratory. (2001) PNGV Battery Test Manual . [Online]. Available: http://avt.inel.gov/battery/pdf/pngv_manual_rev3b.pdf

[92] E. J. Hannon, *Multiple time series*. New York: John Wiley and Sons, Inc, 1970.

[93] T. Kobayashi, D. L. Simon, and J. S. Litt, "Application of a constant gain extended kalman filter for in-flight estimation of aircraft engine performance parameters," in *ASME Turbo Expo 2005: Power for Land, Sea, and Air.* American Society of Mechanical Engineers, 2005, pp. 617–628.

[94] D. L. Simon and T. Kobayashi, "Hybrid kalman filter approach for aircraft engine in-flight diagnostics: Sensor fault detection case," NASA/TM2006-214418 technical report, Tech. Rep., 2006.

[95] H. Dai, X. Wei, Z. Sun, J. Wang, and W. Gu, "Online cell SOC estimation of Li-ion battery packs using a dual time-scale Kalman filtering for EV applications," *Applied Energy*, vol. 95, no. 0, pp. 227 – 237, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306261912001432

[96] J. Zarei, A. Montazeri, M. R. J. Motlagh, and J. Poshtan, "Design and comparison of lqg/ltr and h controllers for a vstol flight control system," *Journal of the Franklin Institute*, vol. 344, no. 5, pp. 577–594, 2007.