Parameterizable FPGA-based Kalman Filter Coprocessor Using Piecewise Affine Modeling

RAW 2016

Department of Electrical and Computer Engineering
Iowa State University

Aaron Mills, Phillip Jones, Joseph Zambreno

Date: 05/23/2016

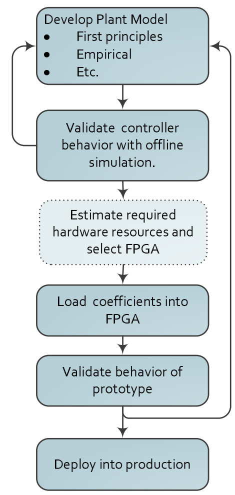
Outline

- Background
  - FPGAs for Control and Sensing
  - Kalman Filter
- Mixed Hardware/software Kalman Filtering
  - Hardware Accelerated Kalman Filtering
  - Limitations of Existing Methods
  - Architecture
- Implementation Results
  - Hardware Resources
  - Performance
- Conclusion

FPGAs for Control and Sensing

- Standard ideal assumptions in control theory
  - Input: Should be performed at the same sampling instant each iteration
  - Computation: complete ASAP after sample is available
  - Output: physical layer signal drivers
- Advantages of FPGAs
  - Application acceleration (most well-known)
  - Massive parallelism can be exploited to create more ideal compute platform
    - Timing guaranteed by design (pico-second scale jitter)
    - Design consolidation (integrate codecs, custom IO, etc)
    - Low power
  - **But:** How to maintain flexibility of software?

Engineering Workflow



Kalman Filter

- What is it?
  - Model-based algorithm to estimate plant (e.g. physical process) state
  - Includes **noise model**.
  - Gain updated dynamically (e.g. online)
  - Very broad applicability
    - Aerospace, robotics, image processing, virtual reality, stock market
  - Often coupled with Linear Quadratic Regulator (LQR)

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(z - C\hat{x}_k)$$

X: state vector
A: state transition model
B: control model
K: gain
Z: measurement
C: measurement model

Linear State Estimator

Initialize

$$\hat{x}_k^- = E[x_0]$$

$$P_{x,0}^- = E[(x_0 - \hat{x}_k^-)(x_0 - \hat{x}_k^-)^T]$$

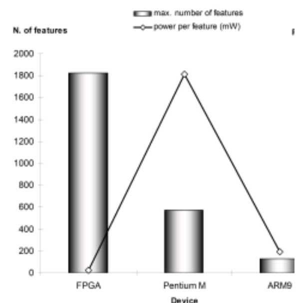
Update

$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1})$	State Prediction
$P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + Q$	Calc. Error Covariance
$K_k = P_k^- C_k^T [C_k P_k^- + R]^{-1}$	Calc. Gain
$\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - g(\hat{x}_k^-, u_k)]$	Update State
$P_k^+ = (I - K_k C_k)P_k^-$	Update Covariance

Extended Kalman Filter

FPGA Kalman Filtering

- Early groundwork
  - Early systolic array work; Fadeev Algorithm [Gaston 1990]
  - special-purpose; poor scalability [Lee 1997]
- Mixed hardware-software approaches
  - Mobile robotics [Bigdeli et al, 2006]
  - Kalman Filtering for SLAM [Bonato et al 2007]
  - Additional systolic array folding [Sudarsanam 2010]
  - UD Filter coprocessor [Gonzalez et al, 2015]
  - EKF coprocessor with heterogeneous PEs [Pritsker 2015]
  - For the most part these works follow a similar design template



[Bonato et al 2007]

Systolic Array for Matrix Math

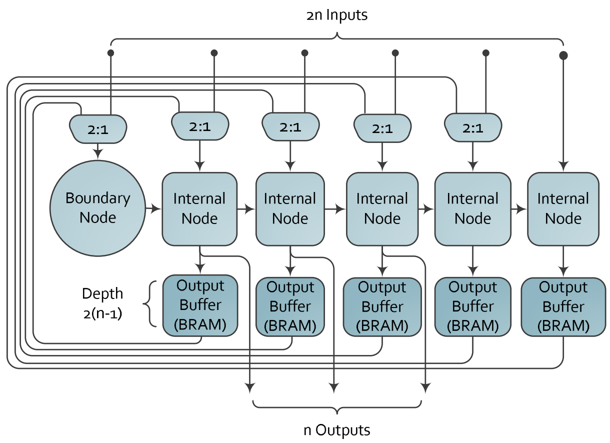
- Fadeev Algorithm: exploits properties of block matrices to perform matrix addition, subtraction, multiplication and inverse
- Uses form of Gaussian Elimination to produce **Shur Complement** ($E=D+CA^{-1}B$) of selected A matrix

Input Example

$$\begin{array}{c} A \quad\quad\quad B \\ \left(\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 1 & 4 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right) \\ C \quad\quad\quad D \end{array}$$

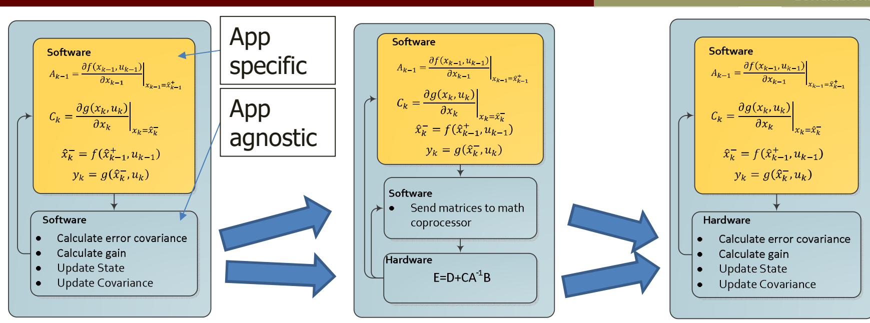
A	B	C	D	E
x	I	I	0	x^{-1}
I	x	y	0	xy
I	I	x	y	x+y
I	x	y	z	x+yz

Typical Refinement (n=3)



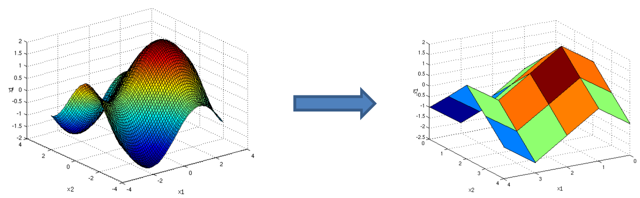
- Node reuse reduces nodes from factor of n^2 to 2n
- However processing time becomes a function of n^2

Typical Codesign Approach



- Limitations
  - **Communication**: bottleneck for large *n* or high update frequency (more than 30% of total iteration time)
  - **Ignores system engineering issues--MCS, jitter**: resource contention still invokes stochastic scheduling behavior
- **Observation:** A linearized modelling approach can eliminate the model-specific software portion → leads to fully hardware-oriented loop
  - Maintains fast, yet flexible hardware
- How to maintain the same or similar operating domain?

Plant Modeling



- Piecewise Affine Modelling
  - Reduces non-linear functions to a set of linear state-space models (one per region i)

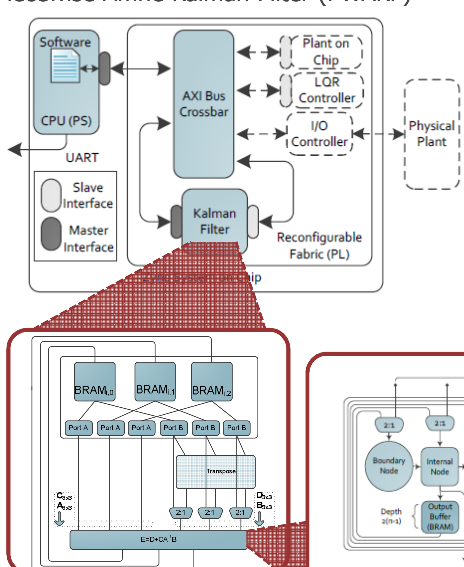
$$x_{k+1} = A_i x_k + B_i u_k$$
$$y_k = C_i x_k + D_i u_k$$

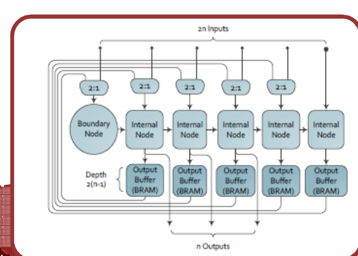
$$\forall k = 1, 2, \ldots \infty$$
$$\forall i = 1, 2, \ldots, q$$

  - accurately model plant behavior over an expanded domain
- Kalman Filter reduces to pure matrix math

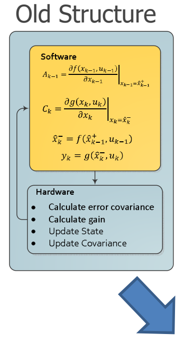
First Design Steps

Piecewise-Affine Kalman Filter (PWAKF)

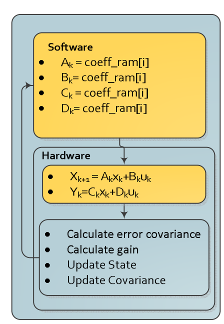




Old Structure



New Structure



Observation:
- Coefficients only sent on transition
- No need to compute f, g

Sequencer Design

Original Algorithm

Initialize

$\hat{x}_k^- = E[x_0]$

$P_{x,0}^- = E[(x_0 - \hat{x}_k^-)(x_0 - \hat{x}_k^-)^T]$

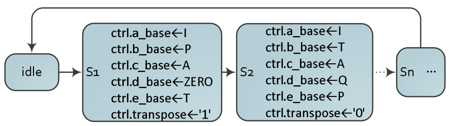
Update

$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1})$	State Prediction
$P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + Q$	Calc. Error Covariance
$K_k = P_k^- C_k^T [C_k P_k^- + R]^{-1}$	Calc. Gain
$\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - g(\hat{x}_k^-, u_k)]$	Update State
$P_k^+ = (I - K_k C_k)P_k^-$	Update Covariance

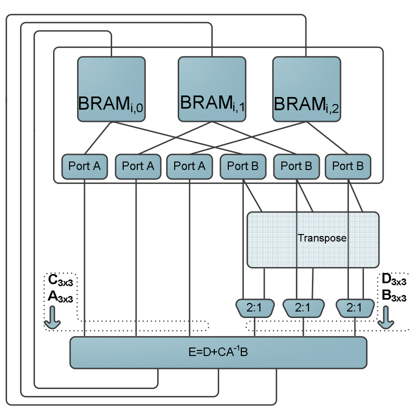
Decomposition

Step	Computation	Input
1	$T = AP^T$	$\begin{bmatrix} I & P^T \\ A & 0 \end{bmatrix}$
2	$\hat{P} = AT^T + Q$	$\begin{bmatrix} I & T^T \\ A & Q \end{bmatrix}$
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	$x = x + KT$	$\begin{bmatrix} I & T \\ K & x \end{bmatrix}$

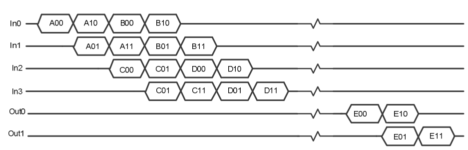
State Machine



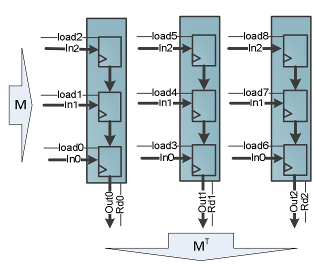
Integrating Systolic Array



Memory Interconnects (n=3)



Systolic Array Input Pattern



Transpose Unit (n=3)

- Additional latency hidden by skewed array input

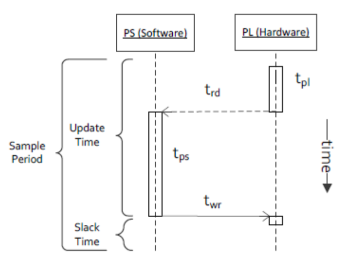
Performance Analysis

- Number of clocks can be determined analytically from hardware:
  - D_{ma} is clocks needed for multiplication/add
  - D_d is clocks needed for division
  - K_s is the number of Kalman algorithm steps
  - Can determine computation delay for pre-hardware simulation

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

- **Analysis Obstacle**: existing work is based on EKF and therefore highly application-specific.
- We propose a generalized approach to enable performance comparisons for piecewise Kalman Filter.

Delay Model



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

1. How much to send? (r_{nc})
   - From 0 (plant model already linear) to 1 (all coefficients need to be updated every region transition
2. How often to send? (r_t)
   - From 0 (no region transitions) to 1 (a region transition every time step)

Nonlinear Template function:

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

Performance Analysis

- Hardware running at 45Mhz vs 200Mhz ARM-A9 ref.
- Speedup vs Software-based EKF (-O2 flag; Gnu Scientific Lib)
- Performance assessed by manipulating n, r_{nc}, r_t

PWA KF 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

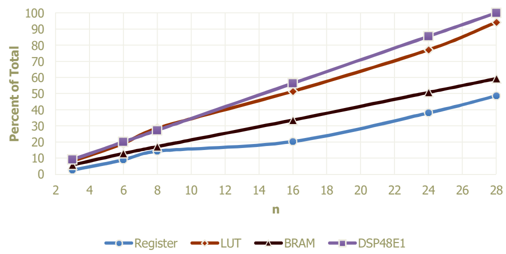
Observations

- ARM-A9 must be run at least 800Mhz to achieve equivalent performance
- For r_t =0, Model Complexity is irrelevant

Comparison to EKF HW-SW

- Performance bounds assessed by manipulating r_{nc}, r_t
- PWAKF exhibits ave. 62% performance increase vs. EKF HW-SW
  - Simplified software; reduced communication (mostly on transitions)

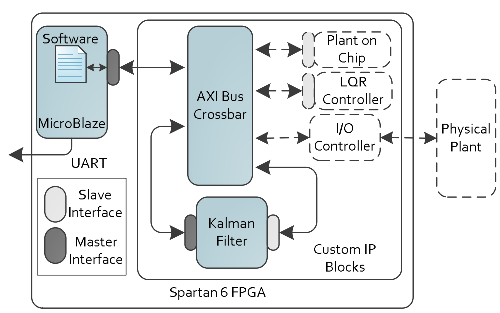
Hardware Resources



- Best power usage & performance when nodes make maximal use of hardware DSPs
- Hardware planning: $n_{dsp}(2n-1) \leq n_{dspmax}$
- For Zynq (equiv. Artex 7): 220 DSPs → n = 28

Case Studies Hardware Setup

- Digilent Atlys Board
  - Better support for power estimation than Zedboard (Zynq)
- MicroBlaze RISC soft processor
  - Configured for performance (2366 LUTs → ~9%)
  - Single-chip solution



- Power Analysis
  - Intel Pentium M*: 20.8W to 35W TDP
  - Intel Atom*: 1.3W to 8.5W TDP
  - ARM9 (within Zynq): 0.6W-1.5W
  - Spartan 6 (+Microblaze): 0.315W (<1% variation vs. n)

* Figures tend to underestimate actual system power requirements

Conclusion

- Looking Back
  - Approach exhibits speedup vs prevalent hardware-software methods.
  - Maintains application-agnostic design.
  - Support for piecewise modeling allows tracking to maintain accuracy at multiple bias points.
  - hardware-based processing dramatically simplifies timing analysis for verification.
  - Low power appealing for battery operated applications.
- Limitation
  - Still requires software stub
  - Communication time impacts maximum update rate
- Future Work
  - Region ID in hardware
  - Impact of additional specialized matrix math units
  - System integration