# An Embedded Scalable Linear Model Predictive Hardware-based Controller using ADMM

Pei Zhang, Joseph Zambreno and Phillip H. Jones

*Presenter: Pei Zhang*

Iowa State University

*peizhang@iastate.edu*

July 5, 2017

# Overview

# Quadratic Programming (QP) solutions

MPC can be posed as a Quadratic Programming problem.

QP problems can be solved reliably via various iterative methods.

- Interior-Point Method (IPM)
- Active Set Method (ASM)
- Splitting Method

# FPGA-based QP solutions

Compare IPM and ASM in FPGA

- ASM gives lower computing complexity and converges faster when the number of decision variables and constraints are small.
- IPM is a better choice when considering scalability.

# State Space Model

A discrete state-space model defines what state a system will be in one-time step into the future:

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

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

- $x_k$ represents the state of the system at time $k$
- $u_k$ represents the input acting on the system at time $k$
- $y_k$ represents outputs of the system at time $k$
- $A$ is a matrix that defines the internal dynamics of the system
- $B$ is a matrix that defines how the input acting upon the system impact its state
- $C$ is a matrix that transforms states of the system into outputs $(y_k)$

## Augmented Vector

$$U_k = \begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+H_u} \end{bmatrix}, \quad \Delta U_k = \begin{bmatrix} \Delta u_k \\ \Delta u_{k+1} \\ \vdots \\ \Delta u_{k+H_u-1} \end{bmatrix}, \quad X_k = \begin{bmatrix} x_k \\ x_{k+1} \\ \vdots \\ x_{k+H_p} \end{bmatrix} \tag{3}$$

Where:

- $H_u$: changeable future input horizon. We assume input $u_k$ will be constant after $H_u$ time steps.
- $H_p$: prediction horizon. Normally, $H_p \geq H_u$.
- $U_k \in \mathbb{R}^{M(H_u+1)}$, $\Delta U_k \in \mathbb{R}^{MH_u}$, $X_k \in \mathbb{R}^{N(H_p+1)}$.

# Cost Function

Cost function:

$$\mathbb{C}(k) = \frac{1}{2}\Big( \sum_{i=k}^{k+H_p} (x_i^T q_i x_i - 2r_i^T q_i x_i) + \sum_{i=k}^{k+H_u} u_i^T p_i u_i + \sum_{i=k}^{k+H_{u-1}} \Delta u_i^T s_i \Delta u_i \Big) + Const \qquad (4)$$

Compact matrix format:

$$\mathbb{C}(k) = \frac{1}{2} \begin{bmatrix} X_k \\ U_k \\ \Delta U_k \end{bmatrix}^T \begin{bmatrix} Q & & \\ & P & \\ & & S \end{bmatrix} \begin{bmatrix} X_k \\ U_k \\ \Delta U_k \end{bmatrix} - R_k^T Q X_k \qquad (5)$$

# Box Constraints
Constraints in the QP Problem

The state $X_k$, input $U_k$ and input rate of change $\Delta U_k$ are constrained by its lower and upper value respectively:

$$\begin{cases} min(x) \leq x_k \leq max(x) \\ min(\Delta u) \leq \Delta u_k \leq max(\Delta u) \\ min(u) \leq u_k \leq max(u) \end{cases}$$

## Consensus Form

One technique for partitioning variables in ADMM is writing the convex
QP problem into consensus form:

$$minimize : \quad \mathbb{1}_{\mathcal{D}}(\chi) + \phi(\chi) \; + \; \mathbb{1}_{\mathcal{C}}(\zeta)$$

$$subject\ to : \; \chi = \zeta$$

## Consensus Form

One technique for partitioning variables in ADMM is writing the convex QP problem into consensus form:

$$minimize : \quad \mathbb{1}_{\mathcal{D}}(\chi) + \phi(\chi) \; + \; \mathbb{1}_{\mathcal{C}}(\zeta)$$
$$subject\ to : \quad \chi = \zeta$$

$$g(\chi) \qquad\qquad\qquad f(\zeta)$$

## Consensus Form

$$g(\chi) = \mathbb{1}_{\mathcal{D}}(\chi) + \phi(\chi)$$
$$f(\zeta) = \mathbb{1}_{\mathcal{C}}(\zeta) \tag{6}$$

$$\chi^{i+1} := prox_{g,\rho}(\zeta^i + v^i) \tag{7}$$
$$\zeta^{i+1} := prox_{f,\rho}(\chi^{i+1} + v^i) \tag{8}$$
$$v^{i+1} := v^i + \rho(\chi^{i+1} - \zeta^{i+1}) \tag{9}$$

Here, $i$ is the iteration counter, $prox_{f,\rho}(\chi)$ is the proximal mapping (or proximal operator) of a convex function $f$:

$$prox_{f,\rho}(\chi) = arg \min_u (f(u) + \frac{\rho}{2}\|\chi - u\|_2^2)$$

$\rho > 0$ is the dual update step length.

# Solve $\chi^{i+1}$
## Matrix-vector Multiply (MvM)

KKT Condition

$$minimize: \frac{1}{2}(\chi^{i+1})^T E \chi^{i+1} + l^T \chi^{i+1}$$
$$subject\ to: G\chi^{i+1} = h \tag{10}$$

# Solve $\zeta^{i+1}$
## Saturation Function

$$f(\zeta^{i+1}) = \begin{cases} \chi^{i+1} - \upsilon^i & \text{, if } min(\zeta) \leq \zeta^{i+1} \leq max(\zeta) \\ f(min(\zeta)) & \text{, if } \zeta^{i+1} \leq min(\zeta) \\ f(max(\zeta)) & \text{, if } \zeta^{i+1} \geq max(\zeta) \end{cases}$$
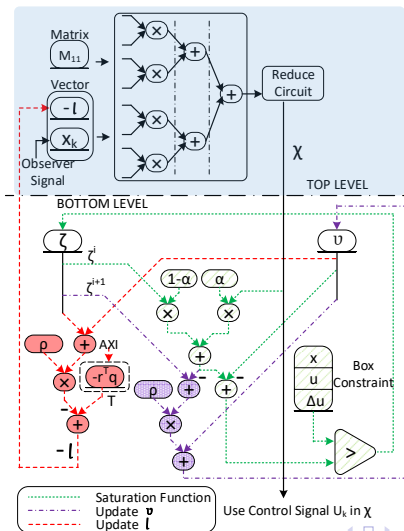
# Solve $v^{i+1}$

Vector Plus Vector

# ADMM Algorithm

**Algorithm 1:** ADMM algorithm

**1** Start from $i = 0$ with arbitrary $\zeta^0$ and $\upsilon^0$.

**2 do**

**3** $\quad l := \begin{bmatrix} Q * R_k \\ \mathbf{0} \end{bmatrix} - \rho(\zeta^i + \upsilon^i)$

**4** $\quad \chi^{i+1} := M_{11} * \begin{bmatrix} -l & x_k \end{bmatrix}^T$

**5** $\quad \zeta^{i+1} := sat(\chi^{i+1} - \upsilon^i, \mathbf{dom} \, \mathcal{C})$

**6** $\quad \upsilon^{i+1} := \upsilon^i + \rho(\zeta^{i+1} - \chi^{i+1})$

**7** $\quad i := i + 1$

**8 until** *stopping criterion is satisfied*;

# Hardware Architecture

Hardware Architecture for ADMM with Relaxation Parameter $\alpha$.

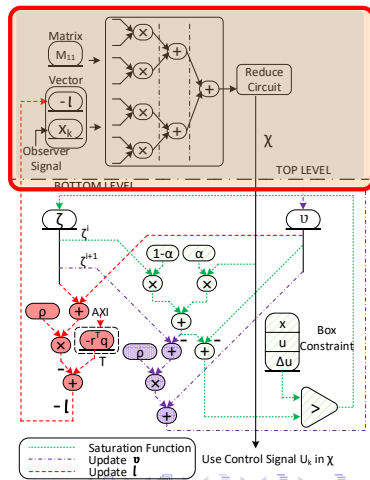# Hardware Architecture
## Processing Flow

### Step 1
### Solve KKT

### Step 2
### Saturation Function

### Step 3
### Update $\upsilon$

### Step 4
### Update $l$

# Hardware Architecture
## Processing Flow

### Step 1
Solve KKT

### Step 2
Saturation Function

### Step 3
Update $\upsilon$
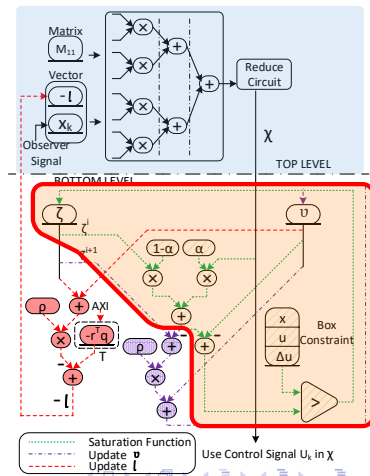
### Step 4
Update $l$

# Hardware Architecture
## Processing Flow

**Step 1**
Solve KKT

**Step 2**
Saturation Function

**Step 3**
Update $\upsilon$
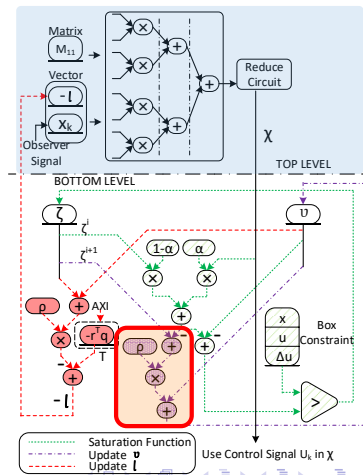
**Step 4**
Update $\iota$

# Hardware Architecture
## Processing Flow

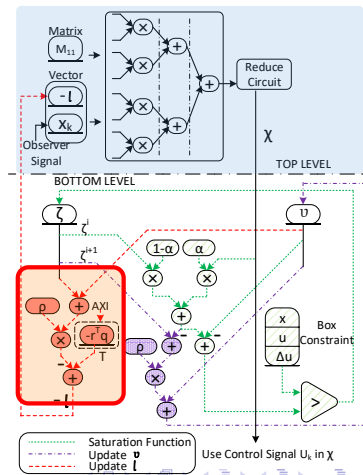### Step 1
Solve KKT

### Step 2
Saturation Function

### Step 3
Update $\upsilon$

### Step 4
Update $l$

# Hardware Architecture
Reduce Circuit
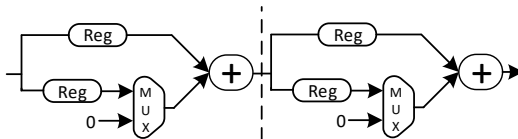


Figure: Reduce Circuit Architecture with Two Cascaded Adders

# Hardware Architecture

Reduce Circuit
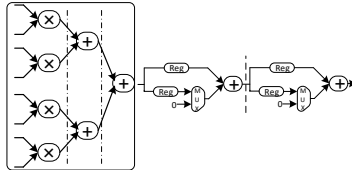


$$G = \begin{bmatrix} I & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A & -I & \cdots & \cdots & \cdots & \cdots & B & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & A & -I & \cdots & \cdots & \cdots & \cdots & B & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & B & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & A & -I & \cdots & \cdots & \cdots & \cdots & B & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & I & -I & \cdots & \cdots & \cdots & \cdots & I & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & I & -I & \cdots & \cdots & \cdots & \cdots & I & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & I & -I & \cdots & \cdots & I \end{bmatrix}$$

# Hardware Architecture
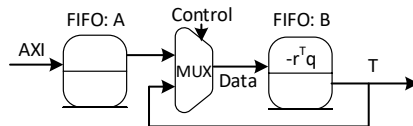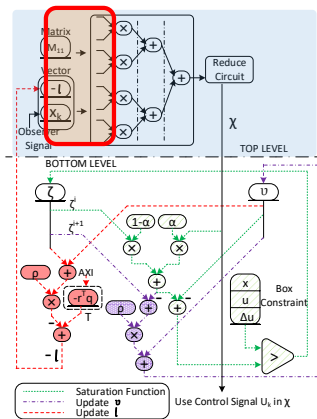Runtime Trajectory Planning



Figure: Runtime Trajectory Planning

# Latency of Each Pipeline Stages



The number of clock cycles to merge all the matrix and vector data into the MVM pipeline is:
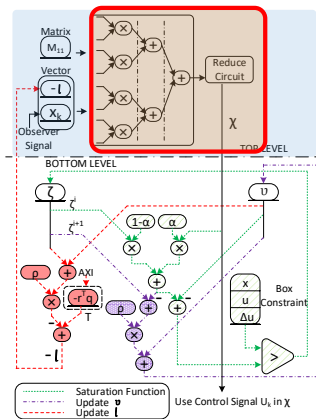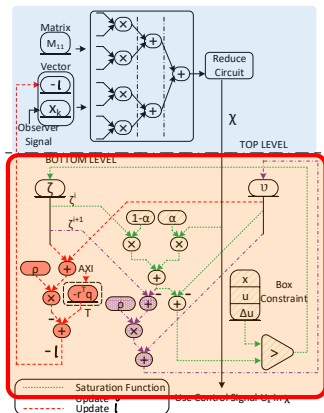
$$L_{read\_M_{11}} = N_{ROW} * (N_R + 1)$$

$$BinaryTree(L_{bt}) = L_M + D_p L_A + N_R(L_A + 2)$$

$$BottomLevel(L_{bl}) = 6L_A + 3L_M + L_C$$

Total Latency:

$$L_{ADMM} = L_{bt} + L_{bl} + L_{read\_M_{11}}$$

# Latency of Each Pipeline Stages



The number of clock cycles to merge all the matrix and vector data into the MVM pipeline is:
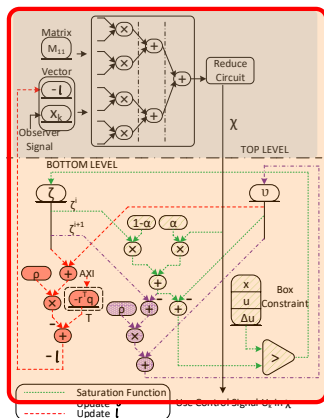
$$L_{read\_M_{11}} = N_{ROW} * (N_R + 1)$$

$$BinaryTree(L_{bt}) = L_M + D_p L_A + N_R(L_A + 2)$$

$$BottomLevel(L_{bl}) = 6L_A + 3L_M + L_C$$

Total Latency:

$$L_{ADMM} = L_{bt} + L_{bl} + L_{read\_M_{11}}$$

# Latency of Each Pipeline Stages



The number of clock cycles to merge all the matrix and vector data into the MVM pipeline is:

$$L_{read\_M_{11}} = N_{ROW} * (N_R + 1)$$

$$BinaryTree(L_{bt}) = L_M + D_p L_A + N_R(L_A + 2)$$

$$BottomLevel(L_{bl}) = 6L_A + 3L_M + L_C$$

Total Latency:

$$L_{ADMM} = L_{bt} + L_{bl} + L_{read\_M_{11}}$$

# Latency of Each Pipeline Stages



The number of clock cycles to merge all the matrix and vector data into the MVM pipeline is:

$$L_{read\_M_{11}} = N_{ROW} * (N_R + 1)$$

$$BinaryTree(L_{bt}) = L_M + D_p L_A + N_R(L_A + 2)$$

$$BottomLevel(L_{bl}) = 6L_A + 3L_M + L_C$$

Total Latency:

$$L_{ADMM} = L_{bt} + L_{bl} + L_{read\_M_{11}}$$
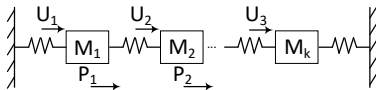
# Mass-spring System (Testbench)



Figure: Mass-spring System

---

- Objective: moving masses to desired positions by applying a force to each mass.
- State Variable: position ($P$) and speed ($\dot{P}$) of each mass[1]
- Constraints:

$$\begin{cases} -0.5m \leq P \leq 0.5m \\ -0.5N \leq u \leq 0.5N \\ -0.1N/s \leq \Delta u \leq 0.1N/s \end{cases}$$

---

[1]$P$ is the position relative to the initial position

# Plant on Chip

A hardware component that emulates the physical behavior of a linear system.
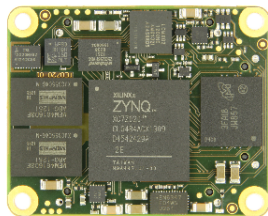


$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

Input: $u_k$                Sensor: $y_k$

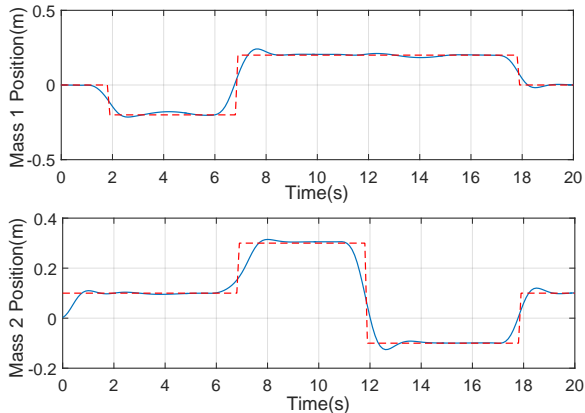# Emulation using Plant on Chip
## Mass Position



Figure: Mass Position Change with respect to Planned Trajectory. Red dashed line is the planned trajectory, and the blue line is the actual trajectory.

# Emulation using Plant on Chip
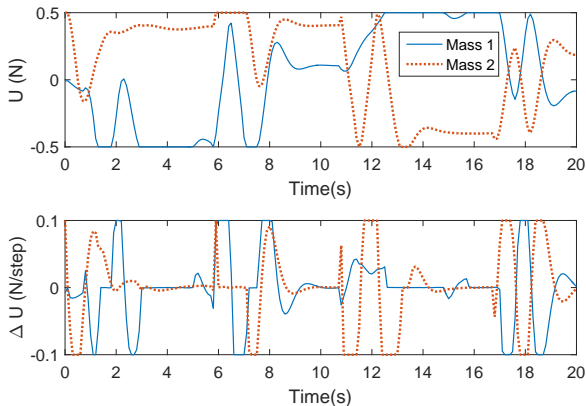
Constraints on Force and its Rate of Change



Figure: Control Signal $U$ and $\Delta U$. Blue line is the input force and the force rate of change for $M_1$, red dashed line is for $M_2$.
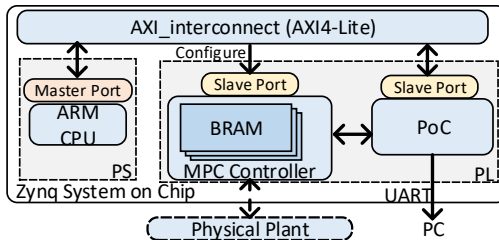
# SW/HW Co-design



Figure: Top Level System Overview
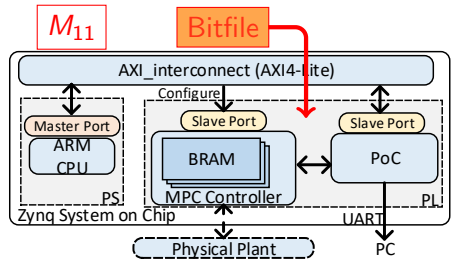
# Deploy Steps

### Step 1

According to system requirements, generate the bitstream in Vivado, and $M_{11}$ matrix in Matlab.

### Step 2

Store $M_{11}$ to BRAM via AXI bus using ARM software.

### Step 3

ARM software configures trajectory and box constraints.



### Step 4

Send start signal to the PoC, and collect data through UART to external computer and plot graph.

# Deploy Steps

## Step 1

According to system requirements, generate the bitstream in Vivado, and $M_{11}$ matrix in Matlab.

## Step 2

Store $M_{11}$ to BRAM via AXI bus using ARM software.

## Step 3

ARM software configures trajectory and box constraints.



## Step 4

Send start signal to the PoC, and collect data through UART to external computer and plot graph.
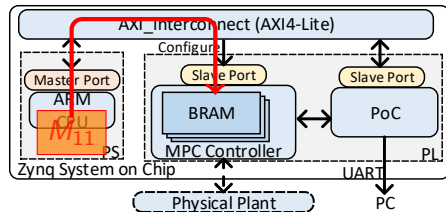
# Deploy Steps

### Step 1

According to system requirements, generate the bitstream in Vivado, and $M_{11}$ matrix in Matlab.

### Step 2

Store $M_{11}$ to BRAM via AXI bus using ARM software.

### Step 3

ARM software configures trajectory and box constraints.



### Step 4

Send start signal to the PoC, and collect data through UART to external computer and plot graph.
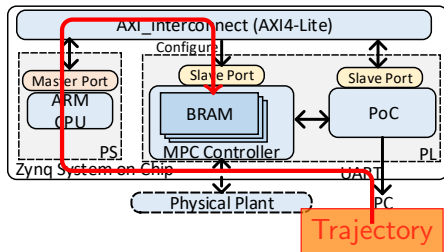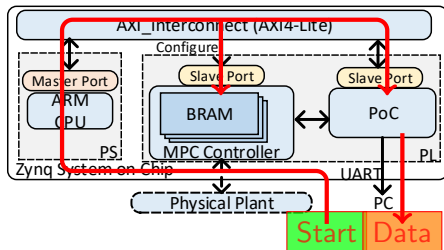
# Deploy Steps

### Step 1

According to system requirements, generate the bitstream in Vivado, and $M_{11}$ matrix in Matlab.

### Step 2

Store $M_{11}$ to BRAM via AXI bus using ARM software.

### Step 3

ARM software configures trajectory and box constraints.



### Step 4

Send start signal to the PoC, and collect data through UART to external computer and plot graph.

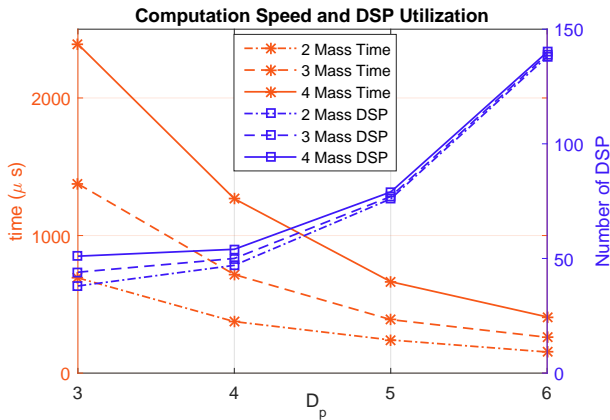# Computation Speed Versus Hardware Resources
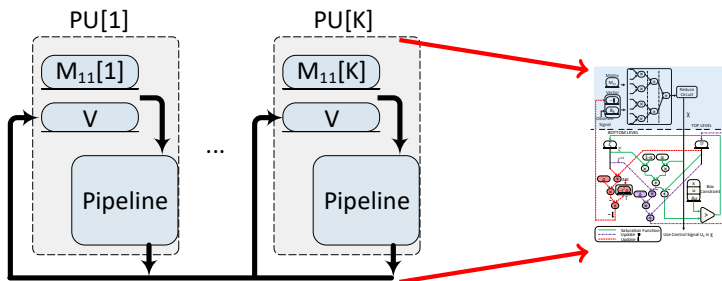


Figure: Computation time of 40 converge iteration loops and DSP usage for different system configurations from simulation. Computation time is marked by ∗, number of DSPs is marked by □. Hardware speed is 100MHz.

# Potential Computation Parallism



$L_{read\_M_{11}}$ is inverse proportional to parallelism $K$.

$$L_{read\_M_{11}} = \frac{N_{ROW}}{K} * (N_R + 1) \qquad (11)$$

# Resource Utilization

Table: Zynq-7020 Hardware Resource Usage

| MVM Size $D_p$ | Flip-Flops (106400 total) | LUTs (53200 total) | 18Kb BRAM (280 total) | DSP48E (220 total) | Maximum Frequency |
|---|---|---|---|---|---|
| 3 | 18147 | 12746 | 55 | 38 | 151.149MHz |
| 4 | 21058 | 15103 | 87 | 47 | 144.885MHz |
| 5 | 32425 | 23391 | 151 | 76 | 143.699MHz |
| 6 | 57167 | 41273 | 279 | 138 | 133.298MHz |

# Timing Summary

Table: Hardware Computation Time per Iteration between Related Work.

| | Method | Data Format | Chip Series | $f_{clk}$ | #Multipliers | Iteration | #Opt Var | Running Time |
|---|---|---|---|---|---|---|---|---|
| This Paper | ADMM | floating-point | Zynq-7020 | 130MHz | 72 ($D_p$=6, K=1) | 40 | 204 | 314.2 $\mu s$ |
| | | | | | | | 350* | 717.2 $\mu s$ |
| | | | | | 80 ($D_p$=5, K=2) | | | 291.4 $\mu s$ |
| | | | ZU9EG (Zynq UltraScale+) | 340MHz | 264 ($D_p$=8, K=1) | | 204 | 46.1 $\mu s$ |
| | | | | | 792 ($D_p$=8, K=3) | | | 30.1 $\mu s$ |
| HW[?] | ADMM | fixed-point | Virtex-6 (LX75) | 400MHz | 216 (K=1) | 40 | 216 | 23.4$\mu s$ |
| | | | Virtex-6 (SX475) | | 1512 (K=7) | | | 4.90$\mu s$ |
| HW[?] | IPM | floating-point | Virtex-7 (XC7VX485T) | 200MHz | 448 | 10 | 240 | 2,650 $\mu s$ |
| SW[?] | ADMM | floating-point | Quad-core Intel Xeon | 3.4GHz | n/a | 35.1 | 525 | 3,400 $\mu s$ |

# Summary of Work

The primary contribution of this work is a software/hardware (SW/HW) co-design that allows:

- configuring an MPC controller for a wide range of plants;
- updating at run-time the desired trajectory to track;
- the flexibility to trade off hardware resources for computing speed;
- easing controller deployment by introducing an SW/HW co-design to decouple hardware details from control and embedded software engineers.

# Citation

An example of the \cite command to cite within the presentation:

This statement requires citation [Smith, 2012].

# References

John Smith (2012)

Title of the publication

*Journal Name* 12(3), 45 – 678.

# The End