# Implementation of MU-MIMO Schedulers on SoC

Ganesh Venkatraman, Janne Janhunen, and Markku Juntti
Email: {gvenkatr, janne.janhunen, markku.juntti}@ee.oulu.fi

Centre for Wireless Communications (CWC),
Department of Communications Engineering (DCE),
University of Oulu, Oulu, FI-90014

2/21

# Outline

# Abstract

- ▶ Problem Studied - Implementing multi-user MIMO scheduler schemes on ZYNQ ZC702 SoC
- ▶ Issues addressed -
  - ▶ Complexity involved in implementing scheduling algorithms - low complex algorithm design
  - ▶ How to partition processing subsystem (PS) and programming logic (PL) in ZYNQ SoC for implementing scheduler algorithms
- ▶ Summary - Proposed implementation supports up to 50 users in the system with $4 \times 4$ MIMO configuration

# Introduction and Motivation

▶ Current standards are moving towards multi-antenna systems due to its numerous advantages

▶ To avail the benefits, spatially multiplexing multiple user streams are considered

▶ In order to do so, efficient precoding and user subset are to be identified

▶ In this work, we analyze the computational needs of different MU-MIMO scheduling algorithms for a single scheduling block

▶ We evaluate algorithm complexity with ZYNQ ZC702 SoC

# Notations used

- ▶ We consider a single-cell multi-user MIMO scenario
- ▶ Let $N_K$ be the total number of users with $N_R$ antenna elements
- ▶ Let $\kappa$ be the total available spatial streams for a user $k$, given by $\kappa = \min(N_T, N_R)$
- ▶ $\mathbf{H}_{\hat{k}} \in \mathbb{C}^{N_R \times N_T}$ be the channel between BS and user $\hat{k}, \forall k \in \mathcal{U}$
- ▶ Let $\mathcal{A} \subset \mathcal{U}$ be the subset of users chosen by scheduling algorithm

# System Model

- Let $\mathbf{H}_{\hat{k}} = \mathbf{U}_{\hat{k}} \mathbf{D}_{\hat{k}} \mathbf{V}_{\hat{k}}^{\mathrm{H}}$ be singular value decomposition of $\mathbf{H}_{\hat{k}}$

- Let $k = \kappa \hat{k} + i$ be the virtual user corresponding to the spatial stream $i \in \{0, \ldots, \kappa - 1\}$

- Using this, we denote virtual channel $\mathbf{h}_k = \mathbf{U}_{\hat{k}}(i)^{\mathrm{H}} \mathbf{H}_{\hat{k}}$, where $\mathbf{U}_{\hat{k}}(i)$ corresponds to the column $i$ of $\mathbf{U}_{\hat{k}}$

- Now, the received symbol $\hat{d}_k$ of virtual user $k$ is given as

$$\hat{d}_k = \mathbf{h}_k \mathbf{m}_k d_k + \sum_{i \in \mathcal{A} \setminus \{k\}} \mathbf{h}_k \mathbf{m}_i d_i + n_k$$

- where $\mathbf{m}_k \in \mathbb{C}^{N_T \times 1}$ is the transmit precoder of user $k$

# Overview of Scheduling Algorithms

▶ To minimize interference, only a subset of users are allowed for transmission

▶ Subset selection with certain objective requires exhaustive search

▶ Scheduling can inherently be performed by precoder designs - efficient iterative algorithms are available

▶ However, as the user count increases, complexity scales up significantly

▶ Hence, precoders are to be designed only for a subset of users chosen by scheduling algorithms

# Successive Projections[†]

- ▶ Based on Gram-Schmidt Orthogonalization Procedure
- ▶ In each iteration, user channel vectors are projected on to the subspace orthogonal to the span of channel vectors already chosen
- ▶ Upon projecting on to the orthogonal subspace, resulting vector with maximum norm is chosen as the candidate user

$$\mathbf{N}(\mathcal{A}) = \mathbf{I}_{N_T} - \mathbf{F} \left( \mathbf{F}^{\mathrm{H}} \mathbf{F} \right)^{-1} \mathbf{F}^{\mathrm{H}}$$

- ▶ where $\mathbf{F}$ is the matrix formed by stacking channel vector of already chosen users in $\mathcal{A}$

[†]T. Yoo and A. Goldsmith, "On the Optimality of Multi-Antenna Broadcast Scheduling using zero-forcing Beamforming, in *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3. IEEE, march 2006.

# Product of Independent Projections (PIPD)[†]

- ▶ As compared to the subspace projection in previous algorithm, vector projections are considered

- ▶ Each user channel is projected on to unit vector in the direction of already chosen users channel

- ▶ Selection is based on the product of independent vector projections

- ▶ Performs significantly closer to successive projections method

- ▶ Due to vector projections, inverse calculation is not required - low complexity

[†]Venkatraman, G., Tolli, A., Janhunen, J., and Juntti, M. "Low Complexity Multi-User MIMO Scheduling for Weighted Sum Rate Maximization", in *Proc. of European Signal Process. Conference (EUSIPCO)*, pp. 820–824, 2013
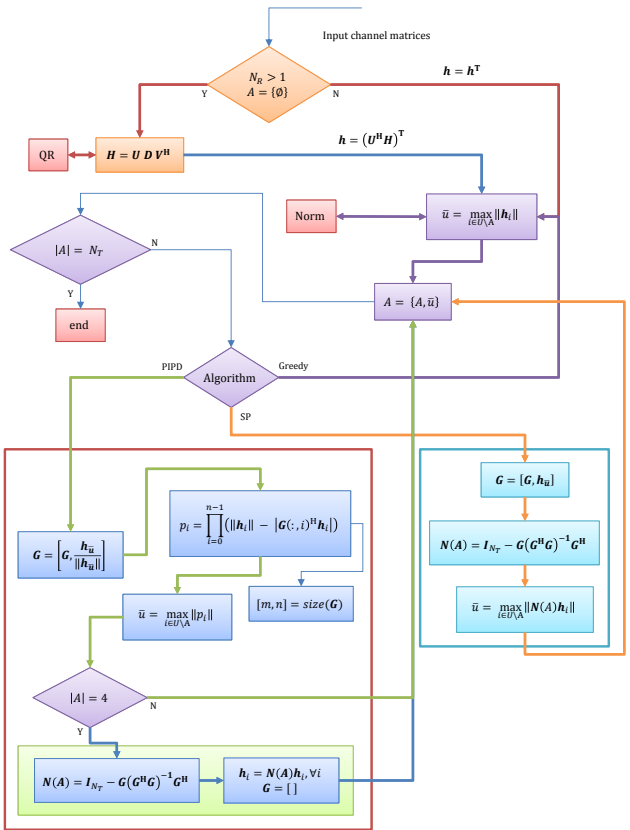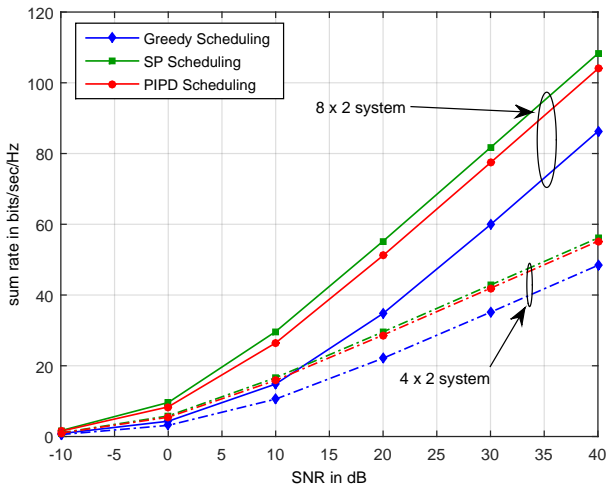
Figure: Block diagram of the scheduler algorithms: greedy, PIPD and successive projections

Figure: Comparison of Scheduler Algorithms for $N_K = 50$ users.

# Overview of ZYNQ ZC702 SoC

▶ ZC702 is a SoC, which includes dual core ARM Cortex A9 and FPGA

▶ It supports AMBA-AXI interface between processing subsystem (PS) and programming logic (PL)

▶ Clock of ARM is 667 MHz and PL operates at 100 MHz

▶ 1GB DDR RAM is accessible by both subsystems through AMBA-AXI

▶ Modes of communication between PS and PL can be
  ▶ B-RAM (block RAM)
  ▶ M_AXI (burst transfer mode)
  ▶ S_AXI (streaming mode of transfer)
  ▶ S_AXI_Lite (address followed by data)

# Algorithm Partitioning

- ▶ Computationally, SVD is the most demanding operation
- ▶ SVD is performed by repeated QR factorization (16 iterations)
- ▶ In order to utilize the SoC efficiently, SVD is performed in PL
- ▶ Since PL operates at 100 MHz, we require parallel SVD operations to increase throughput
- ▶ With the available resources, only 5 SVD operations can be performed in parallel
- ▶ Interface between PL and DDR is with $HP0$ interconnect - high performance AXI_M mode

# FPGA Resource Utilization for 5 SVD's

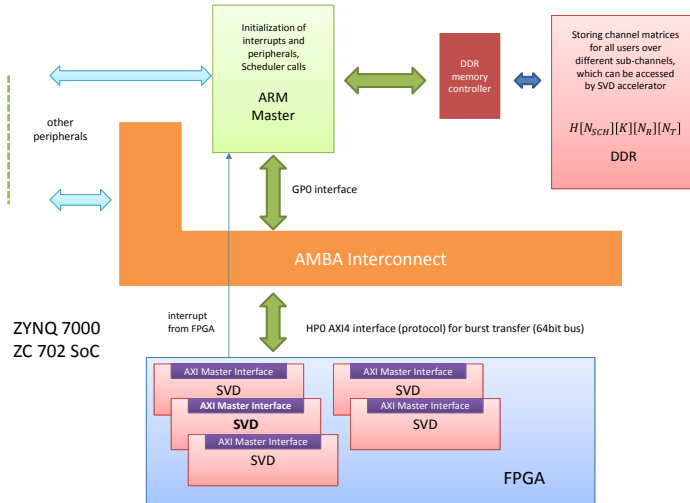| Resource | Utilization % |
|----------|---------------|
| Flip Flops | 39.4 |
| LUT | 72.6 |
| Memory LUT | 6.2 |
| BRAM | 60.7 |
| DSP48 | 86.4 (limiting resource) |
| BUFG | 3.1 |

Figure: Software and Hardware Partitioning on ZYNQ 7ZC702

# Why Scheduler Implementation on ARM

- ▶ SVD is highly complex compared to scheduling complexity

- ▶ Scheduling is sequential, *i.e.*, users are selected based on previous chosen users

- ▶ However, metric calculation for all users and max search over all metrics can be done in parallel

- ▶ To do parallel implementation on PL, it requires more resources, which we are lacking

- ▶ Similar parallelization can also be performed using NEON vector floating point (VFP) co-processor on ARM

- ▶ SVD is required only when channel changes, whereas scheduling is need on per sub-frame basis

# Programming Logic Implementation

- ▶ Real and complex entries - Q3.15 signed format
- ▶ Dynamic range is improved by storing intermediate values in Q9.15
- ▶ Using above Q format, each MAC requires one DSP48 unit, *i.e.* $25 \times 18$ multiplication
- ▶ S_AXI_Lite is used for control command from PS $\rightarrow$ PL, HP0 dedicated lane is used for fetching from DDR
- ▶ Computes SVD for 5 channel matrices in parallel to improve throughput
- ▶ Upon completion, an interrupt is sent to PS

# Processing Subsystem Implementation

- ▶ Signed Q1.15 format is used to represent real and imaginary entries

- ▶ Upon availability of channel, S_AXI_Lite control is used to interrupt PL for SVD

- ▶ On obtaining interrupt from PL, a single ARM core is used to perform scheduling

- ▶ If there are multiple sub-channels, it can be performed over other ARM cores too

Table: Computational Complexity for one SB with $N_K = 50$ users

| $N_T \times N_T$ | $\lambda$ streams | SVD Comp. msec | Greedy msec | SP msec | PIPD msec |
|---|---|---|---|---|---|
| $8 \times 4$ | 4 | 11.427 | 0.098 | 2.029 | 1.280 |
| $8 \times 4$ | 2 | 11.427 | 0.084 | 1.202 | 0.743 |
| $8 \times 2$ | 2 | 3.231 | 0.079 | 1.225 | 0.704 |
| $8 \times 2$ | 1 | 3.231 | 0.072 | 0.796 | 0.421 |
| $4 \times 4$ | 4 | 5.169 | 0.052 | 0.327 | 0.237 |
| $4 \times 4$ | 2 | 5.170 | 0.042 | 0.189 | 0.168 |
| $4 \times 2$ | 2 | 1.481 | 0.038 | 0.202 | 0.139 |
| $4 \times 2$ | 1 | 1.481 | 0.033 | 0.131 | 0.089 |

# Conclusion from Implementation Results

▶ SVD is the computationally expensive part of scheduling

▶ However, due to channel coherency, SVD computation is not per subframe resolution

▶ Current implementation can perform $4,375$ SVD's per second of $8 \times 4$ matrix

▶ Similarly it can handle $33,760$ SVD's per second of $4 \times 2$ matrix

▶ All scheduling schemes satisfy real-time requirement of 1msec with $4 \times 2$ system with $N_K = 50$ users

# Conclusions

▶ We studied the implementation of different state-of-the-art MU-MIMO scheduling algorithms on ZYNQ ZC702 SoC

▶ Complexity is mainly attributed to SVD decomposition of channel matrices

▶ Using parallel implementation, the overall throughput of SVD computation is improved $\approx 4.7$ times (loss is due to memory bottleneck)

▶ We have demonstrated that with the current implementation, all scheduling schemes meet the real-time requirements

▶ As a comparative study, implementing above algorithms on TI C66K2H12 octa-core SoC, supports up to 100 users for $4 \times 4$ MIMO system (to appear in GlobalSIP 2015)