An application specific instruction set processor based implementation for signal detection in multiple antenna systems

An Application Specific Instruction Set Processor Based Implementation for Signal Detection in Multiple Antenna Systems

M. Tamagnone[a], M. Martina[a], G. Masera[a]

[a]*Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy*

Abstract

In comparison to single antenna systems, a wireless multiple-input multiple-output (MIMO) system provides higher throughput at no additional cost of bandwidth, but the high complexity of the detection algorithms poses a major challenge to the hardware implementation. Maximum likelihood (ML) MIMO detection guarantees optimal performance but implies huge processing complexity, which makes acceptable this approach only when the number of transmitting antennas is low and the adopted modulation scheme has a small cardinality. Sphere decoding (SD) is an efficient method that significantly reduces the average processing complexity with no performance penalty.

Most of known sphere decoders have been implemented as **application specific integrated circuits (ASICs)**, but the need for high degree of flexibility in MIMO detection makes interesting also **application specific instruction set processor (ASIP)** implementations. A single programmable ASIP can hardly reach the same processing speed as a fully dedicated ASIC, thus parallel architectures with multiple concurrent ASIPs must be conceived to guarantee sufficient data throughput.

The objective of this paper is to present a new ASIP-based implementation for the detection of MIMO signals. The processor supports multiple lattice modulation schemes (up to 64-QAM) and up to 4 transmitting antennas and it is able to run both ML and close to ML algorithms. A parallel architecture has been also designed with multiple ASIPs, which concurrently execute the detection algorithm on received symbols, a central unit acting as task scheduler, and a buffer for the compensation of non constant throughput. A dedicated bus handles the communication among allocated units. Each ASIP occupies a silicon **area of 0.093 mm^2 and runs at 400 MHz when implemented on a 90 nm CMOS technology**. Achievable throughput depends on the adopted MIMO system and on the number of allocated ASIPs: a detector with 10 ASIPs programmed to run ML detection on a 4×4 MIMO system with 64-QAM modulation offers a **throughput of 78 Mbps** at signal–to–noise ratio SNR=18 dB.

1. Introduction

Multiple-input multiple-output (MIMO) systems [1] are digital transmission systems with more than one transmitting and receiving antenna. Such systems, in combination with space–time codes, offer both multiplexing capabilities and transmit diversity, which can be exploited to achieve increased channel capacity and robustness against multipath fading channels. On the receiver side each antenna receives a linear combination of transmitted symbols. Provided that the receiver is able to reconstruct the originally sent information from received signals, such systems have the potential to multiply the achievable data rate by the number of transmitting antennas at no cost of additional bandwidth. MIMO systems are also known to be able to increase the robustness to combat the fading in wireless channels.

The drawback of MIMO systems is the computational complexity required to perform the detection of the originally transmitted symbols, which is the task implemented by MIMO detectors. In the direct implementation of maximum likelihood (ML) detection, the computational complexity grows exponentially with the order of the system, which depends on the size of the selected constellation and on the number of antennas. Therefore, computationally efficient algorithms are required to achieve acceptable bit rates in the detection process, especially for high order MIMO systems. Sphere Decoding (SD) techniques [2] are a class of efficient detection algorithms that solve the ML problem with polynomial average complexity in the rate [3]. SD techniques are based on the association of all the possible transmitted vectors to the leaves of a decision

tree, but each SD algorithm is different in terms of tree structure and exploring procedure. Since all modern high performance receivers include iterative soft–input soft–output channel decoders, soft–output detection on MIMO channels is also required [4]. Thus, several SD algorithms have been extended to provide soft-output information instead of hard bits.

This paper mainly deals with depth–first algorithms, both hard-output (HO) and soft-output (SO), as the ones presented in [5, 6] respectively. The aim of the paper is to present a new implementation of these algorithms based on an array of Application Specific Instruction set Processors (ASIPs) [7]. Each ASIP is optimized to perform the SD on a single vector symbol, and the parallelization is required to achieve acceptable throughput level.

The proposed system also includes a central processor (CP) that works as scheduler, assigning the received symbols to the allocated ASIPs and retrieving the results from them. The scheduling policies employed in the CP are crucial to find practical solutions to the main problem of the SD: its non–constant throughput. This paper covers the ASIP structure and the proposed scheduling policies as well. Notice that the CP has been implemented only on a behavioral level, while the ASIP has been fully synthesized for both FPGA and CMOS standard cell technologies.

The rest of the paper is organized as follows. The MIMO model and the sphere decoding algorithms are introduced in Section 2 together with the notation used in the paper. Section 3 introduces the designed ASIP architecture, while the structure of the multi-ASIP system and the adopted scheduling policies are detailed in Section 4. Simulation results are given in Section 5, where the effects of different architectural choices are evaluated in terms of throughput and BER (Bit Error Rate) performance. Section 6 provides ASIP synthesis results as well as comparisons with other SD implementations. Finally conclusions are drawn in Section 7.

2. Depth First Sphere Decoding

This section briefly introduces HO and SO algorithms that employ the depth–first sphere decoding. A complete derivation can be found in [5, 6].

2.1. MIMO Link Model

Let us consider a MIMO channel with M_T transmitting antennas and M_R receiving antennas. The same carrier signal is used for every transmitting antenna, each of which transmits synchronously a different digital symbol belonging to the constellation $\mathscr{O}$. Let $\mathbf{s}$ be the vector of the complex transmitted symbols (one entry for each transmitting antenna) and $\mathbf{y}$ the vector of the complex received signals (one entry for each receiving antenna). Then, the model of the channel is given by:

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \quad \text{with} \quad \mathbf{s} \in \mathscr{O}^{M_T} \tag{1}$$

where $\mathbf{H}$ is the channel matrix ($M_R \times M_T$) and $\mathbf{n}$ is a vector of independent complex gaussian random variables with variance $N_0/2$, which models the noise at the receiver. The purpose of Sphere Decoding is to find the most likely vector symbol $\hat{\mathbf{s}}$, i.e. the one that minimizes the Euclidean distance from $\mathbf{y}$:

$$\hat{\mathbf{s}} = \arg\min_{\mathbf{s}\in\mathscr{O}^{M_T}} d(\mathbf{s}) = \arg\min_{\mathbf{s}\in\mathscr{O}^{M_T}} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 \tag{2}$$

2.2. Preprocessing and Decision Trees

A required preliminary step in sphere decoding is QR decomposition, which is a method to triangularize the $M_R \times M_T$ matrix $\mathbf{H}$ [8]:

$$\mathbf{H} = \mathbf{Q}\mathbf{R} \tag{3}$$

where $\mathbf{R}$ is a $M_T \times M_T$ upper triangular matrix, while $\mathbf{Q}$ is a $M_R \times M_T$ matrix with orthonormal columns.

Using this decomposition, a squared Euclidean Distance (ED) $d(\mathbf{s})$ can be associated to each symbol vector $\mathbf{s}$:

$$d(\mathbf{s}) = c + \|\hat{\mathbf{y}} - \mathbf{R}\mathbf{s}\|^2 \tag{4}$$

where $\hat{\mathbf{y}} = \mathbf{Q}^H\mathbf{y} = \mathbf{R}\mathbf{s}^{ZF}$, **the superscript** $(\cdot)^H$ **stands for the Hermitian transpose**, $\mathbf{s}^{ZF}$ is the zero–forcing solution [5] and $c = \|\mathbf{y}\|^2 - \|\hat{\mathbf{y}}\|^2$ is a constant with respect to the symbol $\mathbf{s}$. Problem described by (2) is then equivalently formulated as the problem of finding the $\mathbf{s}$ symbol with the minimum ED. It is worth noticing that constant c does not influence the metric computation expressed in (2) and will then be omitted from now on.

We introduce now the partial symbol vector $\mathbf{s}^{(i)}$, defined as

$$\mathbf{s}^{(i)} \triangleq [s_i, s_{i+1}, \ldots, s_{M_T}]' \tag{5}$$

where $[\cdot]'$ indicate a transposed vector. These partial symbols can be organized in a tree [5], where $i = M_T$ is the level of the tree root, $i = 1$ is the level of tree leaves, and each possible value of partial symbol $\mathbf{s}^{(i)}$ is associated to one node at level i. A node at level i inherits from its parent at the upper level $i-1$ the corresponding partial symbol $\mathbf{s}^{(i-1)}$ and obtains $\mathbf{s}^{(i)}$ by appending one more vector element, corresponding to a specific choice for s_i.

The leaves of the tree are all the possible $\mathbf{s} \in \mathscr{O}^{M_T}$. Therefore ED values $d(\mathbf{s})$ are associated to tree leaves,

while intermediate tree nodes can be associated to Partial Euclidean Distances (PED) $T_i(\mathbf{s}^{(i)})$. PEDs can be additively updated when moving from the root towards the leaves of the tree:

$$\begin{aligned} T_{M_T+1}(\cdot) &\triangleq 0 \\ T_i(\mathbf{s}^{(i)}) &\triangleq T_{i+1}(\mathbf{s}^{(i+1)}) + |e_i(\mathbf{s}^{(i)})|^2 \end{aligned} \quad (6)$$

with $i = M_T, M_T - 1, \ldots, 1$; amounts $|e_i(\mathbf{s}^{(i)})|^2$ are called *distance increments*:

$$|e_i(\mathbf{s}^{(i)})|^2 = \left| \hat{y}_i - \sum_{j=i}^{M_T} R_{ij} s_j \right|^2 = \left| \psi_{i+1}(\mathbf{s}^{(i+1)}) - R_{ii} s_i \right|^2 \quad (7)$$

where

$$\psi_{i+1}(\mathbf{s}^{(i+1)}) = \hat{y}_i - \sum_{j=i+1}^{M_T} R_{ij} s_j \quad (8)$$

From (6) it is evident that the PED of a node is larger than the one of its parent node, and this property is exploited to achieve an efficient tree exploration. For example, to explore only leaves such that $d(\mathbf{s}) < r$, with a certain r (called *sphere constraint*), we can prune the tree under an intermediate node whose PED violates the constraint, so reducing the number of searched nodes, **with no penalty in terms of quality of the found solution**.

2.3. Hard Output SD

By exploring the tree in a depth first way, we can determine the leaf with the smaller ED, which corresponds to the ML solution. The result is given by the bits associated to the found leaf, hence this algorithm provides output sequences of zeros and ones (hard bits) without any information on the reliability of the detected bits.

During tree exploration, when a leaf is reached with ED value smaller than previously calculated, the constraint on the sphere radius is updated, so as the search space is reduced to all tree nodes with PED lower that the current radius. This allows pruning the tree without the risk of eliminating the ML solution. In exploring the sons of a node, the Schnorr-Euchnerr (SE) enumeration [9] can be used: in this approach, the sons are explored in ascending order of their PEDs. In the SE enumeration, the first explored node at tree level l corresponds to the Babai point [10], which can be calculated by means of a rounded division [11]:

$$\left\lfloor 0.5 \frac{\psi_{i+1}(\mathbf{s}^{(i+1)})}{R_{l,l}} + 1 \right\rfloor \quad (9)$$

where $\lfloor 0.5x + 1 \rfloor$ rounds argument x to the nearest odd integer value (this is equivalent to select the nearest point in a PAM constellation). Therefore, if a node violates the current radius constraint, all succesive siblings according to the SE enumeration also violate the constraints and can be pruned with no further processing.

When all the nodes have been explored or pruned, the found leaf with the minimum ED corresponds to the wanted ML solution.

2.4. Soft Output Single Tree Search SD

A larger amount of information can be provided as detector output by also calculating an estimate of the reliability of each detected bit. This leads to the concept of "soft bit", a numerical value whose sign represents the decoded bit (minus for "0", plus for "1") and whose magnitude represents the corresponding reliability.

Thanks to the max-log approximation [4], the bth soft bit of the jth antenna can be expressed by:

$$L(x_{j,b}) \cong \min_{\mathbf{s} \in \chi_{j,b}^{(0)}} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 - \min_{\mathbf{s} \in \chi_{j,b}^{(1)}} \|\mathbf{y} - \mathbf{H}\mathbf{s}\|^2 \quad (10)$$

where $\chi_{j,b}^{(x)}$ is the set of the vector symbols with the bth bit of the jth antenna equal to $x \in \{0, 1\}$. Obviously for each bit (b, j) one of the two terms in (10) is the ED of the ML solution (λ^{ML}), while the other is the one of the so called *best counterhypothesis* for the same bit, normally indicated as $\lambda_{j,b}^{\overline{ML}}$. More details can be found in [4].

The Single Tree Search (STS) SD algorithm presented in [6] can find both the ML solution and the best counterhypotheses for all the bits in the label associated to the vector symbol, so that (10) can be used to generate soft bits. While in other soft–output MIMO detection algorithms multiple visits of the tree are necessary to apply (10), the key characteristic of STS SD is that a single tree search is enough, so as each node is explored at most once. The pruning criterion has to be modified in this case to avoid pruning the best counterhypotheses. In particular the algorithm explores the sons of a node if:

$$d(\mathbf{s}^{(i)}) < \max_{a_l \in \mathcal{A}(\mathbf{x}^{(i)})} a_l \quad (11)$$

where $\mathbf{x}^{(i)}$ is the partial bit sequence associated to symbol vector $\mathbf{s}^{(i)}$, $\{a_l\} \in \mathcal{A}(\mathbf{x}^{(i)})$ and $\mathcal{A}(\mathbf{x}^{(i)})$ is a set including all PED values associated to the counter-hypotheses to be checked:

$$\begin{aligned} \mathcal{A}(\mathbf{x}^{(i)}) = &\left\{ \lambda_{j,b}^{\overline{ML}} | (j \geq i, b = 1, \ldots, M_C) \wedge (x_{j,b} = \overline{x_{j,b}^{ML}}) \right\} \\ &\cup \left\{ \lambda_{j,b}^{\overline{ML}} | j < i, b = 1, \ldots M_C \right\} \end{aligned} \quad (12)$$

If a node violates the constraint

$$d(\mathbf{s}^{(i)}) < \max_{b_l \in \mathcal{B}(\mathbf{x}^{(i+1)})} b_l \tag{13}$$

where $\mathcal{B}(\mathbf{x}^{(i)})$ is defined similarly to (12)

$$\mathcal{B}(\mathbf{x}^{(i+1)}) = \left\{ \lambda_{j,b}^{\overline{ML}} | (j > i, b = 1, \ldots, M_C) \wedge (x_{j,b} = \overline{x_{j,b}^{ML}}) \right\}$$

$$\cup \left\{ \lambda_{j,b}^{\overline{ML}} | j \leq i, b = 1, \ldots M_C \right\} \tag{14}$$

then also its siblings can be pruned.

Each time a leaf is reached, the list of the $\lambda_{j,b}^{\overline{ML}}$ is updated according to the algorithm 1 (expressed here in C-like pseudo code):

Algorithm 1 List administration

```
if PED ≤ λ^ML then
    λ^ML ← PED
    for j ← 1 to M_T do
        for b ← 1 to M_C do
            λ^ML_{j,b} ← min(λ^ML_{j,b}, λ^ML + L_max)
            if x_{j,b} ≠ x^ML_{j,b} then
                λ^ML_{j,b} ← λ^ML
            end if
        end for
    end for
else if PED ≤ max_A{λ^ML_{j,b}} then
    for j ← 1 to M_T do
        for b ← 1 to M_C do
            if x_{j,b} ≠ x^ML_{j,b} then
                if λ^ML_{j,b} > PED then
                    λ^ML_{j,b} ← PED
                end if
            end if
        end for
    end for
end if
```

It can be seen that the chosen range for soft bit is $[-L_{max}, L_{max}]$. While the value of soft bits is theoretically unbounded, in several cases introducing a clipping level has the effect of reducing the number of counter hypotheses that have to be checked and this results in an increased data rate [12].

3. The designed ASIP architecture

Most of sphere decoder implementations are based on dedicated ASICs, that can be optimized both in terms of area and throughput: some examples of dedicated implementations are found in [5, 6, 13, 14, 15]. Although many of these implementations are parameterized and support multiple modulation schemes and antenna numbers, they suffer from limited flexibility. The growing demand for modern wireless systems capable of dynamically adapting to different channel conditions and user constraints raises the need to also explore flexible implementations, where several system level choices can be freely and dynamically changed in a wide range to achieve the wanted performance and quality of service.

High level of flexibility can be obtained resorting to software implementation on general purpose processors. However, **as pointed out in [16]**, the requested level of throughput can be hardly achieved with this kind of implementation. This is mainly due to the high number of elementary operations necessary to carry out the single node exploration. In fact, ASIC based sphere decoders are generally able to explore a tree node in a clock cycle (one node per cycle architectures) [5] or even multiple nodes per cycles [17]. This is obtained by allocating dedicated hardware resources specifically optimized with the aim of exploiting all potential parallelism in the operations that have to be performed in the node visit.

On the other hand, implementations based on general purpose processors do not include customized hardware resources and the number of elementary operations they require to explore a node tends to be very high. This is particularly true in soft-output SD, where the evaluation of the two sets $\mathcal{A}$ and $\mathcal{B}$ defined in (12) and (14) involves several compare and branch instructions.

In ASIPs, the flexibility coming from the processor programmability is coupled to customized instruction set and datapath, where fully dedicated hardware units can be included to achieve better processing efficiency on a specific class of applications than general purpose processors. The novel ASIP architecture described in this Section has been conceived for depth–first Sphere Decoding algorithms. The ASIP contains dedicated hardware resources to perform in a faster way some of the operations involved in the tree exploration, and these functionalities can be invoked by means of dedicated instructions. The designed ASIP can explore a node in an average number of cycles equal to 12, when running the soft–output version of the SD algorithm.

3.1. The Fundamental Structure of the ASIP

Each ASIP is characterized by a plain pipelined architecture based on four stages: the *instruction fetch* (IF), the *instruction decode* (ID), the *execution* (EX) and the *write back* (WB), as shown in Figure 1. A data memory

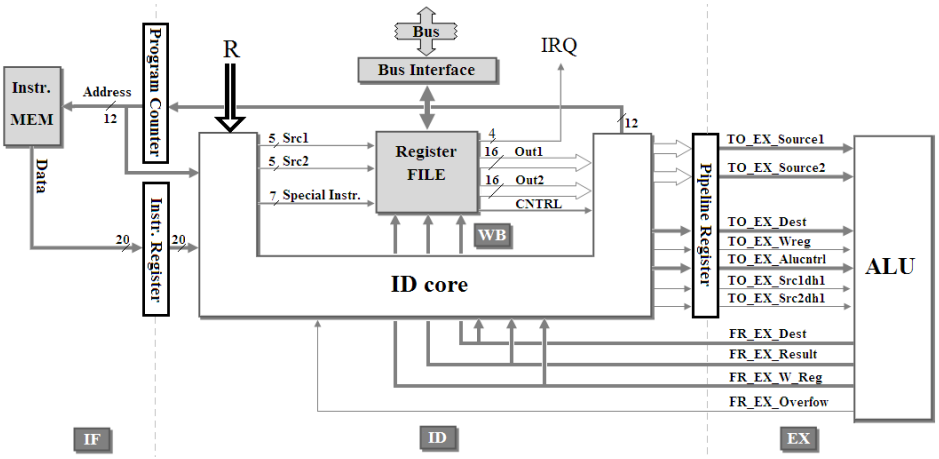

Figure 1: The fundamental structure of the ASIP

is not allocated, so that the algorithm only works on the values stored in the registers. This allows the elimination of a pipeline stage, simplifying the architecture and reducing the data and branch hazards to be handled.

Moreover IF, EX and WB stages are characterized by a very low complexity, with a reduced set of instructions and addressing modes to be supported. The key novelty of the proposed processor comes from the choice of embedding algorithm–specific hardware units in the register file, where several frequently required functions are directly executed. This approach is motivated by the intensive use of registers and allows for a simple datapath. In other words some registers, in addition to the normal read and write operations, also support the execution of specific processing tasks, which are made possible through additional local hardware resources and without involving the execution stage. These special functions can be accessed using dedicated instructions optimized for the tree exploration.

The size of the register file has an impact on the detector flexibility in terms of supported signal constellation and number of antenna. The ASIP described in this paper includes a register file sized to support QAM constellations up to 64-QAM and 4 antennas at both transmitting and receiving sides: **the overall register file size is equal to 187 bytes**. However the flexibility of the ASIP could be easily extended to higher order systems by allocating a larger register file.

All the numerical values are represented using a **16 bit fixed–point format with 6 decimal digits. Two's complement is adopted. This choice guarantees almost the same performance as the floating point model [11]**. The datapath is designed to handle real data and hence it requires a real–value decomposition of the sphere decoding algorithm. While several authors proved that complex-valued model results in a lower number of visited nodes with respect to real-valued one [5], it was shown in [18, 19] that the latter offers two advantages: (i) it involves a lower global number of elementary real-valued operations and (ii) it allows for simple enumeration techniques. Moreover a real–valued datapath occupies a lower silicon area.

3.2. The ASIP Registers and Special Instructions

Both the code running on the ASIP and the CP can access the registers. To this purpose registers are classified in two types according to the access mode: read-only registers, whose content can not be modified;read-write registers whose content can be modified. The access mode to a register depends on both the selected register and who is accessing it (the code or the CP). However, it never happens that a register can be accessed in a read-write way by both the code and the CP.

Several registers can be written neither by the CP nor by the write-back phase of a standard instruction. However their contents can be updated by one or more special instructions. Furthermore some configuration registers can be set by the CP, but the code cannot access directly to their contents. These registers are initialized to properly drive the enumeration process according to the selected modulation and number of antenna.

Finally array of registers are implemented using dedicated index registers to select the element inside the array.

In the following list, the ASIP registers and the special instructions they support are described.

1. **LEV register stores the current level in the tree. It can be incremented or decremented by means of two special instructions, *LEVINC* and *LEVDEC*. The end of the decoding is automatically detected when LEVDEC is performed at the root node (LEV = 0).**
2. **SI registers and enumeration: the SI register array stores the current partial symbol. The update of SI is handled by the SE real enumerator. When the INITENUM instruction is executed, the enumerator is initialized by performing the rounded division in (9): the resulting value is stored in the LEV^{th} component of SI, which is visible from the program. Then, each time the ENUM instruction is called, the same component is set to the following value according to the SE enumeration order. In case all the values have been enumerated, a branch is performed.**
3. **LANDAML, RESULT registers and list administration: the RESULT register stores several values used to compute the final soft bits. In particular it stores the current ML candidate symbol (gray-encoded), together with accumulated PEDs for ML solutions and best counter-hypothesis, λ^{ML} and $\lambda_{j,b}^{\overline{ML}}$. The ML value can be read by means of special instruction LANDAML. Soft bits are sent to the bus and retrieved by the CP after the end of the decoding. A fast procedure is also available in order to retrieve hard bits in a lower number of cycles than required for soft bits. Finally these registers support the procedure for list administration: special instructions UPDCHY and UPDML allow updating the counter-hypotheses and the ML candidate respectively.**
4. **LWBND register initially stores the highest possible value allowed for the $\lambda_{j,b}^{\overline{ML}}$, i.e. the distance of the counter-hypotheses from the received signal. When a new ML solution is reached, the special instruction UPDML is executed on this register to update the sum of the current λ^{ML} and the LLR clipping value.**
5. **SC, SCPRE registers are used to store and update the maximum elements in both sets $\mathcal{A}$ and $\mathcal{B}$. This is done by means of two dedicated comparators.**
6. **ψ register contains ψ values defined in (8); when the level in the tree is increased or decreased, ψ amounts are updated by means of dedicated multipliers and adders working in parallel. The same technique is used in several ASIC implementations [11]. The ψ registers are initialized by the CP with $\hat{\mathbf{y}}$ (4).**

An example of special instruction embedded in the register file is given in Figure 2, where the update of ψ registers is detailed. The R matrix is initially written by the scheduler entry by entry. The ψ vector is also initialized to $\hat{y}_i$ values, according to (7). The shown architecture updates in parallel all ψ values: at each executed LEVINC instruction, the current symbol s_i is selected and each ψ is updated by subtracting the right Rs_i product. Indicated multipliers are easily implemented by means of add & shift operations, due to the values assumed by s_i symbols [11]. It is worth noticing here that the reported architecture is sized for handling r matrices up to 8×8. However, if larger constellations or numbers of antenna are required, the update of ψ registers can be accomodate in multiple cycles.

The other special functions are implemented in a similar way. This embedding approach allows a simplification of the architecture, since it eliminates several pipeline registers. It also offers high processing throughput, as long as the critical path delay across units that implement special functions is kept limited.

3.3. Additional registers and execution unit

The standard instructions are implemented in the EX stage. They include arithmetic operations, immediate value loading, the NOP (no operation), the RST (reset to default), the SLEEP (freeze execution) and jump/branch instructions. The branch instructions perform both the comparison between two registers and the jump. **As the most important operations are executed in the RF stage, the arithmetic unit in the EX stage is not a critical element of the processor and it is not part of the critical path. For this reason, no specific optimization has been necessary for the EX stage. On the contrary, special instructions associated to previously described ASIP registers required carefull optimization at the RTL level, to reduce critical path delay.**

Additional registers that are not associated to special instructions are:

- **R0, R1, R2, R3 registers are general purpose registers, not accessible from the bus and used to store temporary values produced by the algorithm.**
- **A0, A1, SO registers are general purpose registers that can be read by the CP. They can be**

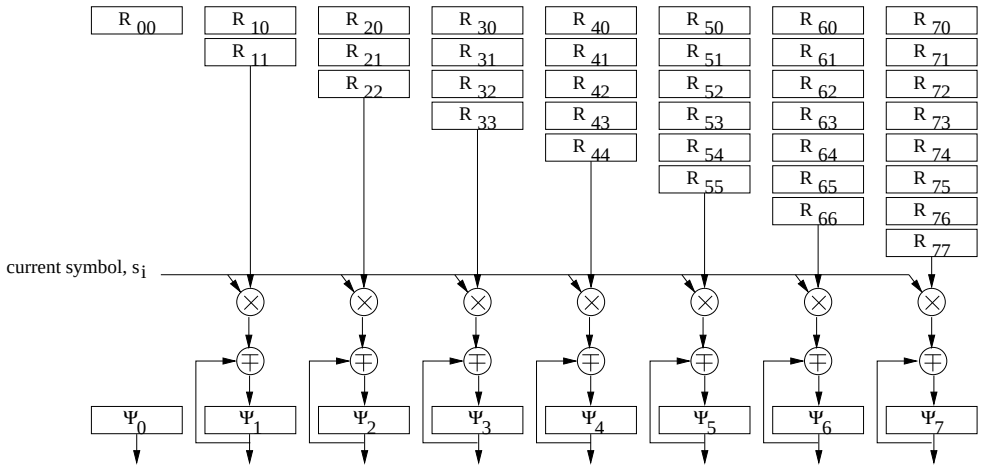

Figure 2: Embedding ψ update in the register file

used to make available custom values to the outside of the ASIP. The four most significant bits of the SO (Status in Output) are brought directly out of the ASIP and are used to implement interrupts.

- **C0, C1 registers are general purpose registers that can be read and written by the CP, but are read-only from the point of view of the ASIP.**

- **TSR (Tree Size Register) is a configuration register written from the bus to set the size of the system (number of transmitting antennas and signal constellation size). A dedicated register, LEVMAX, is loaded with the value $2M_T - 1$, which is frequently used by the algorithm.**

- **PED and PEDPRE registers form an array of registers indexed by LEV. PED registers store the PED values of nodes explored while descending the tree in the depth–first sequence; PEDPRE registers contain the PED values for the parent nodes.**

- **R registers contain the elements of the R matrix. In a multi–ASIP architecture, these registers are shared among the ASIPs and duplicated in order to handle the channel updating phase. While current elements of R are read, new values derived from the last channel estimation are computed and stored. A multiplexer for each ASIP selects which of the two channel matrices has to be used. These multiplexers are driven by a configuration bit in the BUS accessible MUX register.**

- **LMAX register stores L_{max}, the clipping value for computed LLR.**

3.4. The Soft Output SD Assembly Code

The core assembly code for both hard–output and soft–output SD algorithms are reported as Listings 1 and 2 respectively.

Listing 1: Example of ASIP programming code: hard–output SD decoding

```
1  start:       RST
                LDIH        0x7F
3               LDIL  LWBND 0xFF
   loop:        INITENUM
5  nextson:     ENUM  nomoresons
                MUL   R1    RDI   SI
7               SUB   R1    B     R1
                MUL   R1    R1    R1
9               ADD   PED   PEDPRE      R1
                BGE   PED   LANDAML     nomoresons
11              BEQ   LEV   LEVMAX      leaf
                LEVINC      loop
13 leaf:        UPDML
   nomoresons:  LEVDEC      nextson
```

Note that in the arithmetic instructions, the first register is the destination, while the following one (ones) is (are) the source(s).

The initialization phase sets the register LWBND to the highest possible value, that will be reduced in the soft bit case each time a new ML candidate is found.

The main loop starts with the INITENUM instruction, that finds the Babai point and resets the enumerator for the current level. This loop starts whenever the exploration of the sons of a given node begins.

An inner cycle that starts form the ENUM instruction enumerates all the sons. For each son, the PED is computed (instructions from 6 to 9 in both listings), and

then the two algorithms becomes different. In the hard-output case, if the sphere constraint is violated, the code returns to the parent node level, and selects the following node based on the enumeration order. However the soft output case distinguishes two cases accordingly to Equations (11) and (13).

Listing 2: Example of ASIP programming code: soft–output SD decoding

```
start:      RST
            LDIH      0x7F
            LDIL      LWBND  0xFF
loop:       INITENUM
nextson:    ENUM      nomoresons
            MUL       R1     RDI     SI
            SUB       R1     B       R1
            MUL       R1     R1      R1
            ADD       PED    PEDPRE  R1
            BGE       PED    SCPRE   nomoresons
            BGE       PED    SC      nextson
cont:       BEQ       LEV    LEVMAX  leaf
            LEVINC    loop
leaf:       BLT       PED    LANDAML newML
            UPDCHY
            JMP       nextson
newML:      ADD       LWBND  PED     LMAX
            UPDML
            JMP       nextson
nomoresons: LEVDEC    nextson
```

When a leaf is reached, an additional portion of code is executed. In Listing 1 a check is performed to determine if the leaf is a new ML candidate, and in that case the candidate is updated. In Listing 2, also the case of a new counter-hypothesis is checked, and the code calls the appropriate special instruction.

4. Multi-ASIP architecture

The depth–first SD detection can be formulated as an iterative tree search algorithm. However it is characterized by heavy data dependencies that leave almost no instruction level parallelism to be exploited in the processor design. Look-ahead unrolling, coupled to a modified search strategy, was successfully adopted in ASIC implementation to enable higher clock frequency with respect to traditional SD implementations [20]. Such solution provides better throughput at the cost of additional processing complexity, which is not affordable in the ASIP approach. Therefore, in order to achieve a sufficient throughput, the final detector has to use more ASIPs working in parallel, whose activity is scheduled by the CP. Some structures are shared among the ASIPs, such as the program memory and the R matrix, to reduce the overall occupied area.

Figure 3 shows the complete system, with particular attention to the interconnection bus between the CP and the ASIPs. The CP operates as the master, while ASIPs

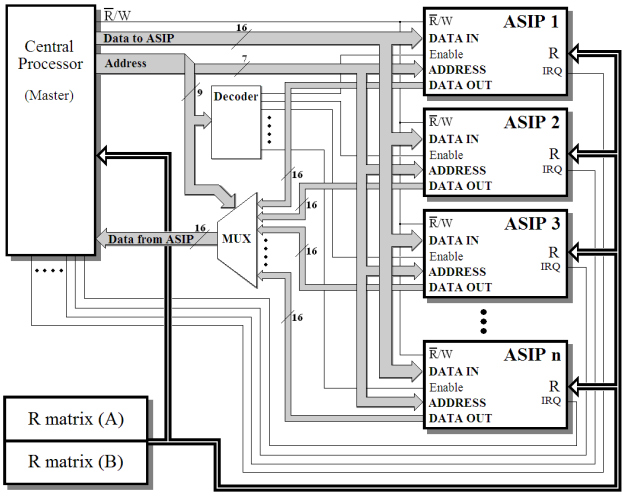

Figure 3: The architecture of the proposed system

are slaves. The upper part of the address generated by the CP is used to select the recipient of the transaction among the ASIPs (Decoder block in Fig. 3). The lowest part of the generated adress is sent to each ASIP. However only the actual recipient uses this part of the address to access internal ASIP registers that are mapped on the bus: this allows for data or command transactions between CP and the selected slave ASIP. The highest part of the address is also used by the bus multiplexer to send the read values returned by the selected ASIP to the CP. The bus also includes interrupt signals (IRQ) coming from each ASIP to the CP. These signals are used when the detection of a symbol has been completed by an ASIP to request the transfer of the achieved result and the assignment of a new symbol vector to be detected.

4.1. Scheduling Policy

Most of MIMO detection algorithms show a non constant detection time. In particular both hard–output and soft–output SD algorithms perform tree search visits that require unpredictable amounts of cycles, with a statistical distribution depending on the signal to noise ratio. The variable detection time results into a variable throughput. The scheduling policy proposed in this Section allows to achieve a constant throughput performance.

The CP preprocesses the incoming symbols (computation of $\hat{\mathbf{y}}$) and then stores them into a circular buffer that is administered by the CP itself. The symbols in the buffer are processed in a pipeline fashion, through a three state FSM (Finite State Machine):

1. State 1: The symbol is not yet processed and it is waiting for ASIP assignment
2. State 2: The symbol is being processed by the assigned ASIP
3. State 3: The symbol has been processed and it is ready to be extracted from the buffer and made available as an output of the MIMO detector.

When an ASIP completes the detection and becomes available again for a new symbol vector, the CP chooses a state 1 symbol from the buffer (giving priority to the older ones), sends it to the ASIP through the bus and starts the detection. Hence the state of the symbol becomes 2.

At the end of the processing, the ASIP raises its interrupt request (IRQ) and waits for the answer from the CP. When the CP is available, it retrieves the results using the bus and stores them in the correspondent buffer element: therefore the detected symbol enters the state 3. As a consequence of the variable detection time, a state 3 element cannot be immediately extracted from the buffer: completion of all previously started symbols needs to be achieved, in order to preserve the correct order of returned symbols. As a consequence it is possible that the several ASIPs remain in an inactive condition, waiting for a single symbol, which requires a long processing time.

To prevent this negative effect, the CP monitors the time elapsed since the assignement of each symbol vector to an ASIP. If this time exceeds a certain threshold, the ASIP is stopped (early detection stopping) and the symbol is retrieved. In this case, only a partial tree exploration is performed by the SD algorithm and there is no guarantee that the returned solution is the ML one. As a consequence the early detection stopping has an impact on the BER performance. This effect has to be controlled by carefully selecting the value of the threshold. In the proposed approach, the time threshold is dynamically chosen based on the number of elements currently stored in the buffer: no limit to the detection time is applied if more than half of the buffer is empty, while a linearly decreasing threshold is adopted from increasing percentages of buffer occupation: the threshold ranges between $10 \cdot T_{safe}$ and $0.5 \cdot T_{safe}$ when 50 % to 90% of the buffer is used. T_{safe} is defined as the ratio between the number of allocated ASIPs, N_{ASIP}, and the requested rate of detected symbols, f_{symb}: $T_{safe} = N_{ASIP}/f_{symb}$. So it is equal to the average time that can be allowed to each ASIP to cope with the symbol rate. Finally the time threshold is set to $0.5 \cdot T_{safe}$ if more than 90% of the buffer is occupied. These choices have been selected by means of extensive simulations performed under different conditions of requested throughput and signal to noise ratio.

The L_{max} clipping level introduced in Section 2.4 can also be used as a parameter to control the detection time. Larger values of L_{max} provide a wider soft bit range, while lower values offer a faster execution of the detection. A method has been developed in the implemented CP to progressively reduce L_{max} value while the number of state 1 and 2 elements in the buffer increases over a certain threshold. A large number of symbols stored in the buffer and waiting for ASIP assignement or completion of the detection indicates that the detection time needs to be reduced; in this kind of situation, the current value of L_{max} is decreased by multiplying it by 0.99. On the contrary, if the number of stored symbols that are not yet processed is under the threshold, current L_{max} is multiplied by 1.01. The threshold has been chosen as a fifth of the maximum buffer size.

The presented parameters have been found empirically, by performing simulations with different values and policies, in order to optimize BER and throughput performances.

The effects of the proposed scheduling policy and L_{max} control technique are described in the following Section.

5. Simulation Results

In this Section, simulation results are reported for a 4×4 MIMO system with 64-QAM modulation. The complex–valued 64-QAM constellation has been transformed into a real–valued 8 PAM. As a consequence, the tree has $M = 16$ levels, with 8 sons per node. **An i.i.d. Rayleigh fading channel is assumed, with perfect channel knowledge and unsorted QR decomposition. The signal-to-noise ratio (SNR) is defined as $SNR = M_T E_s/N_0$, with $E_s = \mathrm{E}[|s|^2]$, $s \in O$. In order to evaluate the achievable throughput, it is assumed that system clock runs at 400 MHz: as shown in Section 6, this is the best frequency value resulting from the detector synthesis performed on a 90 nm CMOS technology**.

As a first simulation result, Figure 4 shows the average throughput achieved while executing the soft–output algorithm (with clipping level set to $L = 3$) on both the designed ASIP and a well–known general purpose processor, namely the StrongARM SA-110. The two processors execute the same version of the detection algorithm and reported throughput values have been obtained by averaging

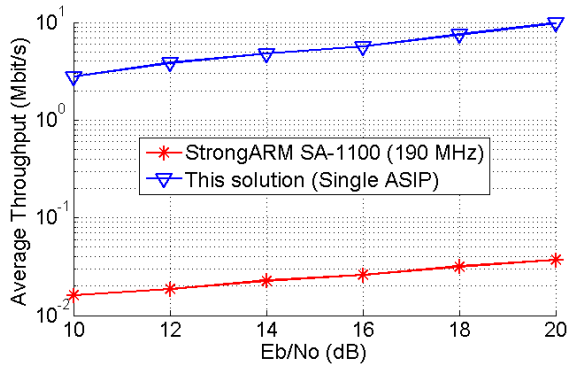

Figure 4: Througput performances offered by a single ASIP compared to a general purpose processor (StrongARM SA-1100). Clipping level is set to L=3.

the processing time taken for the detection of 800 symbols. Berkeley SIMIT 2.1 cycle accurate simulator [21] was used to evaluate the StrongARM execution time, assuming the maximum supported clock frequency (190 MHz).
From the comparison in Figure 4, it can be seen that obtained results for the two implementations are separated by more than two orders of magnitute. Although higher throughput could be obtained by executing the same application on a different general purpose processors, maybe clocked at a higher frequency, the large difference in Figure 4 clearly shows the potential of the ASIP approach, which enables efficient execution while still offering software flexibility.

In order to further show the performance provided by the designed ASIP, two types of simulations have been performed. In the *unconstrained* simulations, the early detection stopping and the L_{max} control are disabled: this means that the ASIP is free to complete the tree exploration according to the described algorithm and a constant L_{max} value is adopted. Full ML detection is achieved in this case, but the resulting throughput is not constant. By using a virtually unlimited buffer, unconstrained simulations allow measuring the average throughput of the detector.

In the *constrained* simulations, both early detection stopping and L_{max} control are enabled: in this case, a thoughput constraint can be set and the simulation outcome is the effect of adopted scheduling techniques on the BER performance.

Figures 5 and 6 show the average throughput achievable with a single ASIP detector, running hard–output

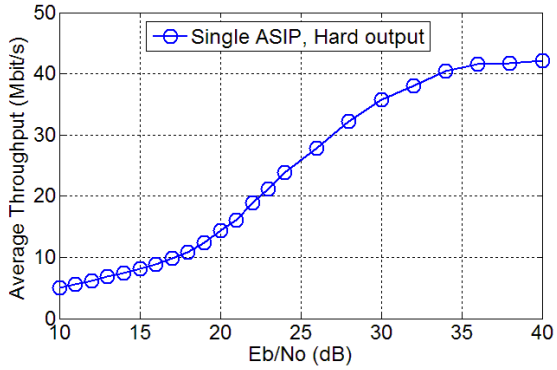

Figure 5: Throughput of a single ASIP with the hard-output algorithm (unconstrained simulation)

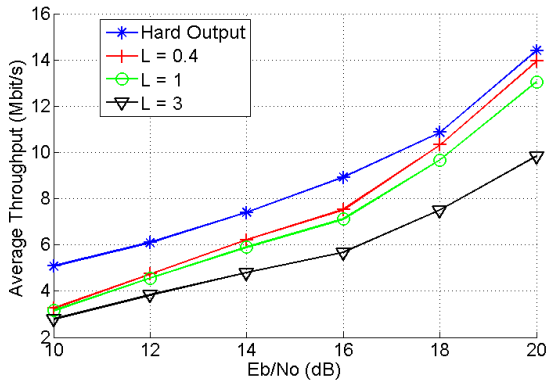

Figure 6: Throughput of a single ASIP with the soft-output algorithm (unconstrained simulation). Here L represents the selected clipping level L_{max}.

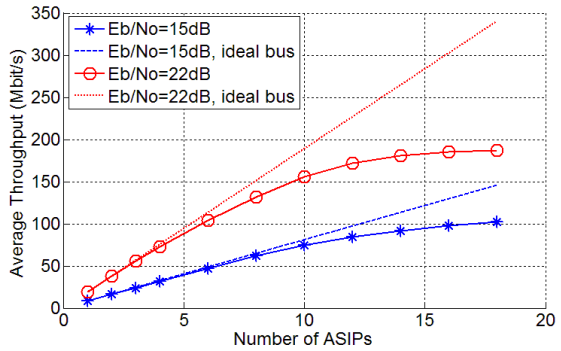

Figure 7: Throughput performances vs. the number of ASIPs, unconstrained hard-output simulation. Dotted lines indicate the throughput performance achievable assuming unlimited bandwidth for the bus.

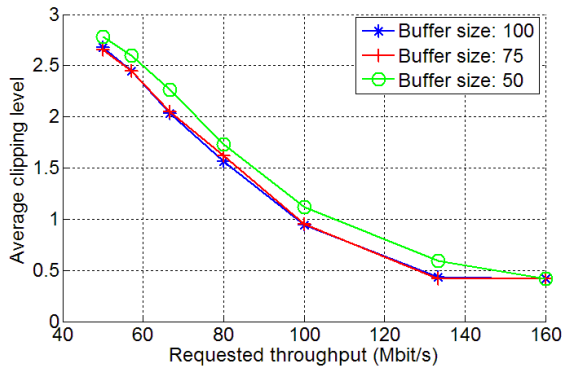

Figure 8: Average clipping level achieved by the constrained system vs the requested throughput (10 ASIPs, E_b/N_0 = 20dB)

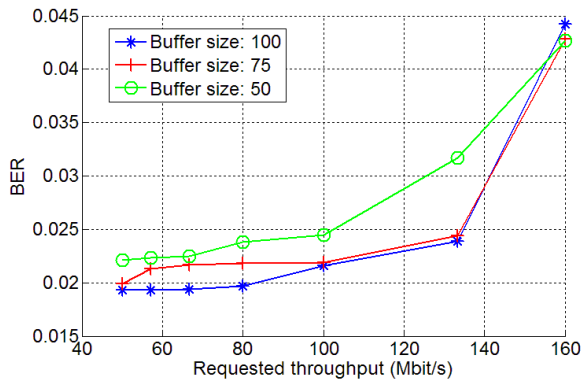

Figure 9: Hard BER achieved by the constrained system vs the requested throughput (10 ASIPs, E_b/N_0 = 20 dB)

and soft–output algorithms respectively. The throughput increases with the signal to noise ratio, E_b/N_0, in both cases and it is generally higher with hard–output detection. The average throughputs obtained with three different values of L_{max} (0.4, 1 and 3) are given in Figure 6: it can be seen that soft–output decoding can approach hard– output version in terms of throughput if the clipping level is lowered from 3 to 0.4.

Figure 7 reports the average throughput obtained with unconstrained simulations of several detectors with different numbers of allocated ASIPs. The Figure shows the effect of the generated bus traffic on the throughput performance. As a consequence of the overhead due to symbol and command transactions, the total achievable throughput with n allocated ASIPs (solid lines) is lower than n times the throughput of a single ASIP detector (dotted lines). This is particularly evident when the bus is working close to its bandwidth limit, which is approximately equal to 130 10^6 soft–bits per second, for the 4×4, 64-QAM MIMO system. **The adopted communication bus between CP and ASIPs tends to become a performance bottleneck when n is large. However results reported in Figure 7 show that the average throughput achieved with $n = 10$ ASIPs is close to 100 Mbps with signal to noise ratio of 22 dB and the performance penality introduced by the bus is still acceptable. More efficent bus architectures would be required for larger numbers of allocated ASIPs.**

Results obtained from constrained simulations performed on a 10 ASIP detector operating with signal to noise ratio equal to E_b/N_0 = 20 dB are given in Figures 8 to 10. In particular, Figure 8 shows the average value of the clipping level L_{max} for different values of the requested throughput and buffer size. The average clipping level strongly depends on the requested throughput, while it is almost independent of the number of elements in the buffer. In fact a large buffer effectively compensates for long detection times, but this does not result in an increased clipping level.

The size of the buffer also affects the occupied area and the detection latency. Moreover, due to the described scheduling policy, the buffer size has an effect on the probability of early detection stopping and hence on the BER performance. Figure 9 reports the BER performance measured at the output of the detector in the case of unconstrained simulation of hard–output detection. It can be seen that the BER level grows worse with requested throughput, especially after 140 Mbit/s, when the bus traffic comes close to bandwidth limits. The BER plot also shows that performance can be im-

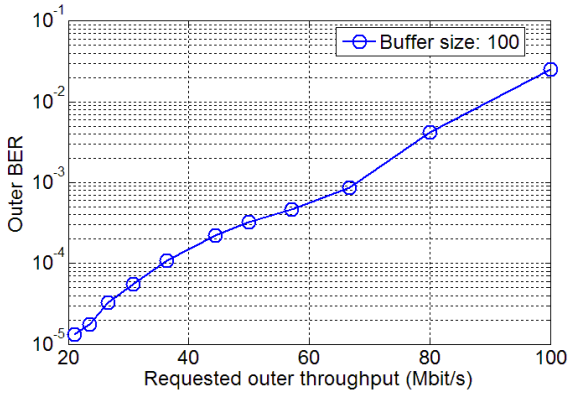

Figure 10: BER achieved at the output of the Viterbi decoder (constrained system with 10 ASIPs, E_b/N_0 = 20 dB)

proved by increasing the buffer size. However, increasing the buffer size above a certain limit does not provide additional benefit: in fact, when this saturation limit is reached, almost all the symbols are detected with no early detection stopping. For instance, it can be seen from Figures 8 and 9 that there is little advantage in choosing size 100 instead of a 75, so a good value for the buffer size is 75 in the considered case.

Finally Figure 10 gives the BER performance obtained with the 10 ASIP soft–output detector concatenated with a Viterbi decoder. In this case, at the transmitted side, a convolutional encoder (rate 1/2, constraint length 3, and generator polynomials $G_1 = (1,1,1)$ $G_2 = (1,0,1)$ receives source bits and feeds the MIMO modulator with generated codewords; at the receiver side, soft information available at the output of the MIMO detector is processed by a Viterbi decoder [22]. Buffer size is here set to 100 and $E_b/N0$ = 20 dB. A wide range of trade-offs between requested throughput and obtained BER performance can be achived with the described scheduling policy.

6. ASIP synthesis and comparisons

The ASIP unit was modelled using VHDL language and synthesized with Synopsys Design Compiler using a 90 nm CMOS technology. Achievable clock frequency is equal to 400 MHz, while obtained area occupation is 0.093 mm^2, corresponding to approximately 43,000 equivalent gates (GE) [1]. Most of ASIP area is occupied by the RF stage (Table 1). Dissipated power at the clock frequency of 400 MHz and with Supply voltage of 1 V has also been estimated after logic synthesis with Synopys Design Compiler and is equal to 14.1 mW. Critical path delay is located in the RF unit and equals to 2.3 ns.

[1] One GE is the area of a two input NAND gate, equal to 2.15 μm^2 on the considered technology

Table 1: Occupied area for ASIP components with 90nm CMOS standard cell technology

ASIP stage	Area (μm^2)
IF	2860
RF	85534
EX	4908
whole ASIP	93504

Table 2: Stratix III implementation

$F_{clk,max}$	117.52 MHz
Family	Stratix III
Device	EP3SL50F484C2
Timing model	Final
Logic Utilization	19%
Combinational ALUTs	6,189/38000 (16%)
Memory ALUTs	0/19000 (0%)
Dedicated logic registers	1981/38000 (5%)
Total pins	79
DSP block 18-bit elements	2/216 (<1%)

The ASIP was also implemented for FPGA technology using the software Quartus II 9.0: synthesis results are shown in Table 2: achievable clock frequency is 117 MHz and 6,189 ALUTs are used in a Stratix III device.

Comparison to other similar implementations of MIMO detectors is extremely difficult. Fair comparisons imply that the number of antennas, the modulation method, the channel model, the finite precision arithmetic and the signal to noise ratio are the same between considered systems. Moreover only architectures synthetized on the same type of implementation platform should be compared, as different platforms have drastically different potentials in terms of cost, power dissipation and flexibility. As a matter of fact, only few ASIP architectures are available in the literature for MIMO detection. The key characteristics of two ASIP based detectors [23, 16] are reported in Table 3, to enable comparison with the implementation described in this paper. Table 3 also includes a recent ASIC detector [24], in order to evaluate the cost and throughput overhead of the ASIP approach with respect to a fully dedicated solution. When programmed to handle the 4 × 4 16-QAM

Table 3: Comparison with other works

Reference	[23]	[24]	[16]	this work		
type	ASIP	ASIC	ASIP	ASIP		
Antennas	4×4		2×2	4×4	4×4	2×2
Modulation	16-QAM	64-QAM	64-QAM	64-QAM	16-QAM	64-QAM
Algorithm	KB LSD	FSD	KB LSD	SESD STS		
BER performance	close to ML			ML		
Technology	130 nm	45 nm	130 nm	90 nm		
Area (KGE)	26.6	70	25	43		
f_{CK} (MHz)	100	500	280	400		
T (Mbps)	1	187	6.8	7.8	12.6	17.4
Cycles per bit	100	2.67	41.18	51.28	31.74	22.99

system, the proposed ASIP achieves better throughput than the ASIP in [23], at the cost of a larger implementation complexity. If one wants to evaluate the VLSI efficiency of an implementation by means of a unique figure of merit, the throughput to area ratio (TAR) can be used: for the 4×4 16-QAM system with signal–to–noise ration $E_b/N_0 = 18$ dB, the presented ASIP achieves **a TAR of 293 bps/GE (bit per second over gate equivalent count), while 37.6 is obtained for the ASIP in [23]**. Lower area complexity (24 kGE) is required by the ASIP described in [16], which achieves a throughput of 6.8 Mbps on a 2×2 64-QAM system: these figures correspond to a TAR value of 272 bps/GE, which is lower than **the 404 bps/GE resulting from the 17.4 Mbps and 43 kGE reported in Table 3** for the ASIP proposed in this paper. Moreover it must be considered that the processor in [16] only supports 2×2 MIMO systems, **while the proposed ASIP was sized to support up to the** 4×4 **64-QAM system**.

The average number of cycles required per detected bit is also reported in the last row of the Table. This value provides a technology independent measure of the architecture efficiency in terms of processing time. The proposed ASIP outperforms the two previously published implementations; however a large gap remains with respect to fully dedicated ASICs, such as the one given in [24].

Current wireless communication standards require throughput values higher than reported in Table 3 for ASIP implementations. So multi–processor architectures are necessary to achieve higher data rates. Table 4 shows the performance that can be obtained with **four** multi-processor architectures: the second column refers to a GPU–based implementation **[25], using an NVIDIA Tesla C1060 graphic card with 240 stream processors running at 1300MHz**. The third column gives results for the 5 interconnected ASIPs introduced in [23]. **Column four summarizes results from a GPU implementation based on an NVIDIA Quadro FX1700 card [26], with four stream multi–processors running at 920 MHz. For this implementation, only performance achieved for a** 2×2 **16-QAM system are available, while the other results reported in the Table are related to** 4×4 **systems.** Finally complexity and throughput results obtained with the parallel architecture proposed in this paper, targeted to 10 interconnected ASIPs are provided in the last column. The global complexity of this multi–ASIP architecture is lower than 10 times the complexity of the single ASIP solution, as some resources are shared among the processors, such as the **R** memory and the CP. It can be seen from the Table that despite of the huge processing power available in the Nvidia cards, **the proposed multi–ASIP architecture provides competitive throughput (77 Mbps, at SNR=18 dB), larger than shown in [16] and lower than in [25].** Finally, when compared to the implementation in [23], the proposed multi–ASIP architecture requires 139% additional resources, but offers a 14.7 times higher throughput.

Table 4: Complexity and throughput obtained with multi-processor architectures

Reference	[25]	[23]	[26]	this work
type	GPU	ASIP	GPU	ASIP
Antennas	4×4		2×2	4×4
Modulation	64-QAM		16-QAM	64-QAM
Area (KGE)	-	170	-	407
f_{CK} (MHz)	1300	100	920	400
T (Mbps)	120	5.3	36	77

7. Conclusions

A multi–ASIP architecture for MIMO detection is proposed in this paper. The pipeline structure of the ASIP is very simple to limit hardware complexity and special instructions are associated to several internal registers to achieve efficient computation of key processing steps. Dynamic techniques are proposed to schedule symbols in the multi–ASIP version of the detector and to achieve a constant throughput. The ASIP architecture can be reconfigured on the fly and offers a large potential for flexibility. Not only the proposed system is able to easily adapt to different system level parameters, but it can also run different algorithms, such as hard–output and soft–output SD algorithms. The designed detector proves that the ASIP based approach is a viable solution to achieve high throughput and flexibility at the same time, in both hard–output and soft–output MIMO detectors.

References

[1] A. Paulraj, R. Nabar, D. Gore, Introduction to Space-Time Wireless Communications, Cambridge Univ. Press, 2003.

[2] E. Viterbo, J. Boutros, A universal lattice code decoder for fading channels, Information Theory, IEEE Transactions on 45 (5) (1999) 1639 –1642. doi:10.1109/18.771234.

[3] B. Hassibi, H. Vikalo, On the expected complexity of integer least-squares problems, in: Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on, Vol. 2, 2002, p. II. doi:10.1109/ICASSP.2002.1006038.

[4] B. Hochwald, S. ten Brink, Achieving near-capacity on a multiple-antenna channel, Communications, IEEE Transactions on 51 (3) (2003) 389 – 399. doi:10.1109/TCOMM.2003.809789.

[5] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, H. Bolcskei, Vlsi implementation of mimo detection using the sphere decoding algorithm, Solid-State Circuits, IEEE Journal of 40 (7) (2005) 1566 – 1577. doi:10.1109/JSSC.2005.847505.

[6] C. Studer, A. Burg, H. Bolcskei, Soft-output sphere decoding: algorithms and VLSI implementation, Selected Areas in Communications, IEEE Journal on 26 (2) (2008) 290 –300. doi:10.1109/JSAC.2008.080206.

[7] J. Van Praet, G. Goossens, D. Lanneer, H. De Man, Instruction set definition and instruction selection for ASIPs, in: High-Level Synthesis, 1994., Proceedings of the Seventh International Symposium on, 1994, pp. 11 –16. doi:10.1109/ISHLS.1994.302348.

[8] M. Damen, H. El Gamal, G. Caire, On maximum-likelihood detection and the search for the closest lattice point, Information Theory, IEEE Transactions on 49 (10) (2003) 2389 – 2402. doi:10.1109/TIT.2003.817444.

[9] C. Schnorr, M. Euchner, Lattice basis reduction: improved practical algorithms and solving subset sum problems, Math. Computat. 66 (2) (2004) 181–191.

[10] E. Agrell, T. Eriksson, A. Vardy, K. Zeger, Closest point search in lattices, Information Theory, IEEE Transactions on 48 (8) (2002) 2201 – 2214. doi:10.1109/TIT.2002.800499.

[11] B. Cerato, G. Masera, E. Viterbo, Decoding the Golden Code: A VLSI design, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 17 (1) (2009) 156 –160. doi:10.1109/TVLSI.2008.2003163.

[12] E. Zimmermann, D. Milliner, J. Barry, G. Fettweis, Optimal llr clipping levels for mixed hard/soft output detection, in: Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, 2008.

[13] J. Lee, S.-C. Park, S. Park, A pipelined VLSI architecture for a list sphere decoder, in: Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, 2006.

[14] K. Lee, B. Daneshrad, VLSI implementation of a quasi-ML, energy efficient fixed complexity sphere decoder for MIMO communication system, in: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, 2010.

[15] C.-H. Liao, T.-P. Wang, T.-D. Chiueh, A 74.8 mW soft-output detector IC for 8x8 spatial-multiplexing MIMO communications, Solid-State Circuits, IEEE Journal of 45 (2) (2010) 411 –421. doi:10.1109/JSSC.2009.2037292.

[16] J. Antikainen, P. Salmela, O. Silveny, M. Juntti, J. Takala, M. Myllyla, Fine-grained application-specific instruction set processor design for the k-best list sphere detector algorithm, in: Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on, 2008, pp. 108 –115. doi:10.1109/ICSAMOS.2008.4664853.

[17] B. Wu, G. Masera, A novel VLSI architecture of fixed-complexity sphere decoder, in: Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on, 2010, pp. 737 –744. doi:10.1109/DSD.2010.10.

[18] S. Lee, J. Lee, S. Seo, S. C. Park, VLSI implementation of area-efficient list sphere decoder, in: International Symposium on Intelligent Signal Processing and Communications, ISPACS 2006, 2006, pp. 610 – 613.

[19] M. Myllyla, M. Juntti, J. R. Cavallaro, Implementation aspects of list sphere detector algorithms, in: Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE, 2007, pp. 3915 –3920. doi:10.1109/GLOCOM.2007.744.

[20] M. Troglia Gamba, G. Masera, Look-ahead sphere decoding: algorithm and VLSI architecture, Communications, IET 5 (9) (2011) 1275 –1285. doi:10.1049/iet-com.2010.0523.

[21] W. Qin, S. Malik, Simit-ARM: A series of free instruction-set simulators and microarchitecture simulators.
URL `http://embedded.eecs.berkeley.edu/mescal/forum/2.html`

[22] S. Benedetto, E. Biglieri, V. Castellani, Digital Transmission Theory, Prentice Hall, 1987.

[23] J. Antikainen, P. Salmela, O. Silven, M. Juntti, J. Takala, M. Myllyla, Application-specific instruction set processor implementation of list sphere detector, in: Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on, 2007, pp. 943 –947. doi:10.1109/ACSSC.2007.4487358.

[24] P. Bhagawat, R. Dash, G. Choi, Systolic like soft-detection architecture for 4x4 64-QAM MIMO system, in: Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09., 2009, pp. 870 –873.

[25] M. Wu, Y. Sun, S. Gupta, J. Cavallaro, Implementation of a high throughput soft MIMO detector on GPU, Journal of Signal Processing Systems 64 (2011) 123–136, 10.1007/s11265-010-0523-4.
URL `http://dx.doi.org/10.1007/s11265-010-0523-4`

[26] T. Nylånden, J. Janhunen, O. Silven and, M. Juntti, A GPU implementation for two MIMO-OFDM detectors, in: Embedded Computer Systems (SAMOS), 2010 International Conference on, 2010, pp. 293 –300. doi:10.1109/ICSAMOS.2010.5642054.