# Algorithm-Architecture Co-Design of Soft-Output ML MIMO Detector for Parallel Application Specific Instruction Set Processors

Min Li, Robert Fasthuber, David Novo, Bruno Bougard, Liesbet Van Der Perre, Francky Catthoor
IMEC, Leuven, Belgium
EAST, K.U.Leuven, Leuven, Belgium
{limin, fasthubr, novo, bougardb, vdperre, catthoor}@imec.be

## Abstract

*Emerging SDR baseband platforms are usually based on multiple DLP+ILP processors with massive parallelism [10]. Although these platforms would theoretically enable advanced SDR signal processing, existing work implemented basic systems and simple algorithms. Importantly, MIMO is not fully supported in most implementations [7][9][11]. [1] implemented MIMO but with a simple linear detector. Our work explores the feasibility for SDR implementations of soft-output ML MIMO detectors, which brings 6-12 dB SNR gains when compared to popular linear detectors. Although soft-output ML MIMO detectors are considered to be challenging even for ASICs [3][4], we combine architecture-friendly algorithms, application specific instructions, code transformations and ILP/DLP explorations to make SDR implementations feasible. In our work, a 2×4 ADRES based ASIP with 16-way SIMD can deliver 193Mbps for 2×2 64QAM, and 368Mbps for 2×2 16QAM transmissions. To the best of our knowledge, this is the first work exploring SDR based soft-output ML MIMO detectors.*

## 1. Introduction

With the exploding design cost in the deep sub-micron era, the current trend is to implement most baseband functionalities on programmable or reconfigurable platforms. The SDR (Software Defined Radio) paradigm, which was mainly successful in the base-station and military segments, is currently emerging in the handset market as well. Especially, massively parallel instruction set architectures, as those combining ILP (Instruction Level Parallel) and DLP (Data Level Parallel) features [1][7][9][10][11], are becoming prevailing. These architectures offer impressive dimensions of parallelism. For instance, the NXP EVP processor has 10 FUs (Function Units) and 6 of them support 16-way SIMD (Single Instruction Multiple Data)[11]. The SODA processor supports even 32-way SIMD instructions[7]. In addition, multiple processors are combined to further strengthen the massive parallelism.

Theoretically, the massive parallelism would enable SDR implementations of advanced wireless signal processing, such as soft-output ML (Maximum Likelihood) MIMO (Multiple Input Multiple Output) detectors. Unfortunately, only simple systems and algorithms have been demonstrated in reported SDR baseband implementations. For instance, [7][9][11] do not support MIMO. [1] demonstrated SDM (Space Division Multiplexing) based MIMO, but with the simplest MMSE (Minimum Mean Square Error) linear detector, which significantly falls behind the recent research progresses of wireless signal processing [5].

MIMO is an essential component in future wireless communication systems, supporting advanced MIMO is doubtlessly important for future SDR. When working with the SDM, a MIMO detector is the key component to recover multiple transmitted data streams. The aforementioned linear MMSE detector is computational efficient, but suffers from poor BER (Bit Error Rate). Compared to linear detectors, hard output ML MIMO detectors brings 4-8 dB SNR gains[4]. Recently, a SSFE (Selective Spanning with Fast Enumeration) hard output ML detector has been proposed and implemented for SDR systems[6].

However, the remarkable potential of MIMO is still *not* fully delivered with hard output ML detectors. Importantly, both Turbo and LDPC (Low Density Parity Check) decoders in emerging standards (WiMAX, 3GPP LTE, etc.) require *soft information* as input to achieve the best BER. Hence, SOMLDs (Soft-output ML MIMO detectors) are highly desired. In fact, SOMLDs bring 2-4 dB SNR gains comparing to the hard-output counterparts, and 6-12 dB comparing to linear detectors. This has been reported to approach the limit of Shannon bounds [5].

Our work explores the feasibility for SOMLDs implementations on massive parallel ASIPs (Application Specific Instruction Processors). To the best of our knowledge, this is the first work exploring SOMLDs for SDR. With explicit architecture/compiler friendliness, a low complexity algorithm is designed for SDR. The algorithm allows to fully exploit the massive parallelism on a SDR platform. Second, we combine ASIs (Application Specific Instructions)



978-3-9810801-5-5/DATE09 © 2009 EDAA

and code transformations to significantly reduce the number of instructions and required memory accesses. Importantly, kernels of the proposed algorithm are mostly based on area efficient operators, so that the cost of ASIs is very low. The dimensioning of an ADRES based ILP+DLP ASIP is explored to achieve at least 120+ Mbps throughput.

The remaining part of this paper consists of the following sections: section 2 briefs the background; section 3 introduces the architecture-friendly low complexity algorithm; section 4 presents the ASIs and architecture explorations of an ADRES based ILP+DLP ASIP; section 5 gives results; Section 6 concludes the paper.

## 2. Background

A MIMO system can be viewed as transmitting an $Nt \times 1$ vector signal $\mathbf{s}$ through an $Nr \times Nt$ matrix channel $\mathbf{H}$, with $Nr \times 1$ Gaussian noise vector $\mathbf{n}$ added at the received vector signal $\mathbf{y}$: $\mathbf{y} = \mathbf{Hs} + \mathbf{n}$. Both Turbo and LDPC decoders require LLR (Log-Likelihood Radio) as soft information input from a soft-output detector. Let $LLR(j, b)$ denote the LLR of the $bth$ bit of the $jth$ transmitted scalar symbol in $\mathbf{s}$. Most practical SOMLD implementations are decomposed into two parts: (a) A *list generator* that gives a list of candidate symbol vector, denoted by $\mathcal{L}$; (b) A *LLR generator* that search over the list $\mathcal{L}$ and calculate the LLR for all $(j, b)$. List generators are often based on the principle that give the set $\mathcal{L} = \{\mathbf{s}\}$ with $\|\mathbf{y} - \mathbf{Hs}\|^2$ minimized. The list sphere decoder and various soft-output K-Best variants [3] [4] fall in this category.

With the generated list $\mathcal{L}$, $LLR(j, b)$ can be calculated as [5]:

$$(\min_{s \in \mathcal{L} \cap \chi^0_{j,b}} \|\mathbf{y} - \mathbf{Hs}\|^2 - \min_{s \in \mathcal{L} \cap \chi^1_{j,b}} \|\mathbf{y} - \mathbf{Hs}\|^2)/2\sigma^2 \qquad (1)$$

where $\chi^0_{j,b}$ and $\chi^1_{j,b}$ are the disjoint sets of symbol vectors that have the $b$th bit in the $j$the scalar symbol equal to 0 and 1, respectively; $\sigma^2$ is the variance of noise.

Instead of searching in the aforementioned joint sets, the bit-flipping strategy is proposed to simplify the operation. When generating $LLR(j, b)$, the corresponding bit of symbol vectors in $\mathcal{L}$ is flipped to 0 or 1 to introduce two new sets $\mathcal{L}^0_{j,b}$ and $\mathcal{L}^1_{j,b}$. Hence, Eqn.(1) is further transformed as:

$$(\min_{s \in \mathcal{L}^0_{j,b}} \|\mathbf{y} - \mathbf{Hs}\|^2 - \min_{s \in \mathcal{L}^1_{j,b}} \|\mathbf{y} - \mathbf{Hs}\|^2)/2\sigma^2 \qquad (2)$$

We can apply the QRD (Orthogonal-Triangular-Decomposition): $\mathbf{H} = \mathbf{QR}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. It can be shown that $\|\mathbf{y} - \mathbf{Hs}\|^2 = c + \|\hat{\mathbf{y}} - \mathbf{Rs}\|^2$, $\hat{\mathbf{y}} = \mathbf{Q}^H\mathbf{y}$, where $c$ is a constant. Hence, Eqn.(2) can be transformed as:

$$(\min_{s \in \mathcal{L}^0_{j,b}} \|\hat{\mathbf{y}} - \mathbf{Rs}\|^2 - \min_{s \in \mathcal{L}^1_{j,b}} \|\hat{\mathbf{y}} - \mathbf{Rs}\|^2)/2\sigma^2 \qquad (3)$$

Eqn.(3) shows that, for *each* bit of the transmitted MIMO symbol, a SOMLD needs to perform two hard-output ML detections over $\mathcal{L}^0_{j,b}$ and $\mathcal{L}^1_{j,b}$. Hence, a SOMLD is much more complex than hard-output ML detectors, the LLR generation is computationally dominant.

To reduce the complexity of Eqn.(3), the squared Euclidean-norm $\|\phi\|^2$ in Eqn.(3) can be replaced by the Manhattan-norm $\|\phi\|_1 = |\Re(\phi)| + |\Im(\phi)|$, which is multiplication-free [2]. Then, $LLR(j, b)$ becomes:

$$(\min_{s \in \mathcal{L}^0_{j,b}} \|\hat{\mathbf{y}} - \mathbf{Rs}\|_1) - \min_{s \in \mathcal{L}^1_{j,b}} \|\hat{\mathbf{y}} - \mathbf{Rs}\|_1)/2\sigma^2 \qquad (4)$$

In our work, LLR is generated with Eqn.(4).

## 3. Architecture/Compiler Friendly SOMLD

Our algorithm explicitly targets massive parallel ASIPs with ILP+DLP. First, the algorithm is optimized to be friendly to architecture and compiler, so that the massive parallelism of SDR processors can be efficiently utilized. Second, the algorithm is designed to contain mostly area-efficient and low latency operators. This enables low cost ASIs.

### 3.1. List Generation

We use the SSFE algorithm [6] for list generation. In the SSFE, high level algorithmic transformations make the dataflow structure fit ILP/DLP architectures very well. Abundant *vector-parallelism* is enabled in the SSFE. Memory rearrangement, shuffling and non-deterministic behaviors are all excluded. Comparing to the ASIC-minded K-Best, the SSFE results in a completely regular and deterministic dataflow structure. Hence, it can be easily parallelized and efficiently mapped onto architectures. Standard compiler optimizations, such as the software pipelining and vectorization, can easily apply. Besides the efficient vectorization, the SSFE also has many other architecture friendly features. In this paper we can only briefly introduce the characteristics of the SSFE. Details can be found in [6].

### 3.2. LLR Generation

#### 3.2.1 Partial and Selective Updating

The SSFE list generator with the Euclidean-norm gives the set $\mathcal{L}$ of $\mathbf{s}$ with $\|\mathbf{y} - \mathbf{Hs}\|^2$ approximated minimized. With the QRD $\mathbf{H} = \mathbf{QR}$, the minimization of $\|\mathbf{y} - \mathbf{Hs}\|^2$ is equivalent to minimizing $\|\hat{\mathbf{y}} - \mathbf{Rs}\|^2$. A spanning-tree is constructed for this purpose. Level of the tree is $Nt + 1$; mark the root-level as $i = Nt + 1$ and the leaf-level as $i = 1$. Each node at level $i \in \{2, .., Nt + 1\}$ is expanded to $\mathcal{C}$ nodes at level $i + 1$, where $\mathcal{C}$ is the constellation size. In this tree each node at level $i \in \{1, 2, .., Nt\}$) is uniquely described by a partial symbol vector $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$, the leaves at level $i = 1$ correspond to all possible vector-symbols $\Omega^{Nt}$. Annotate the root node with $T_{Nt+1} = 0$ and

starting from Level $i = Nt$, the PED (Partial Euclidean Distance) of a partial symbol vector $\mathbf{s}^i = [s_i, s_{i+1}, .., s_{Nt}]$ is $T_i(\mathbf{s}^i) = T_{i+1}(\mathbf{s}^{i+1}) + \|e_i(\mathbf{s}^i)\|^2$, where the PED-increment $\|e_i(\mathbf{s}^i)\|^2$ is $\|e_i(\mathbf{s}^i)\|^2 = \|\hat{y}_i - \sum_{k=i}^{Nt} R_{ik} s_k\|^2$.

When flipping the $b$th bit of the $j$th scalar symbol in $\mathcal{L}$ to get $\mathcal{L}^1_{j,b}$ and $\mathcal{L}^0_{j,b}$, an original partial symbol vector $\mathbf{s}^i = [s_i, \ldots, s_j, \ldots, s_{Nt}]$ is flipped to $\mathbf{s}^{i(0)}_{j,b} = [s_i, \ldots, s^0_{j,b}, \ldots, s_{Nt}]$ and $\mathbf{s}^{i(1)}_{j,b} = [s_i, \ldots, s^1_{j,b}, \ldots, s_{Nt}]$. $s^0_{j,b}$ and $s^1_{j,b}$ mean that the $b$th bit of the $j$th scalar symbol $s_j$ is flipped to 0 and 1, respectively. If the LLR generation is based on Manhattan-norm, essential operation blocks in the LLR generation are: (a) calculating the PMD (Partial Manhattan Distance) increments, $\|e_i(\mathbf{s}^{i(0)}_{j,b})\|_1$ and $\|e_i(\mathbf{s}^{i(1)}_{j,b})\|_1$; (b) updating PMD 0 and PMD 1, $T^0_i(\mathbf{s}^{i(0)}_{j,b})$ and $T^1_i(\mathbf{s}^{i(1)}_{j,b})$. The above calculations are formulated as:

$$\begin{aligned}
\|e_i(\mathbf{s}^{i(0)}_{j,b})\|_1 &= ||\hat{y}_i - \sum_{k=i}^{j-1} R_{ik} s_k - R_{ij} s^0_{j,b} - \sum_{k=j+1}^{Nt} R_{ik} s_k||_1 \\
\|e_i(\mathbf{s}^{i(1)}_{j,b})\|_1 &= ||\hat{y}_i - \sum_{k=i}^{j-1} R_{ik} s_k - R_{ij} s^1_{j,b} - \sum_{k=j+1}^{Nt} R_{ik} s_k||_1 \\
T^0_i(\mathbf{s}^{i(0)}_{j,b}) &= T^0_{i+1}(\mathbf{s}^{i+1(0)}_{j,b}) + \|e_i(\mathbf{s}^{i(0)}_{j,b})\|_1 \\
T^1_i(\mathbf{s}^{i(1)}_{j,b}) &= T^1_{i+1}(\mathbf{s}^{i+1(1)}_{j,b}) + \|e_i(\mathbf{s}^{i(1)}_{j,b})\|_1 \qquad (5)
\end{aligned}$$

Importantly, when flipping the $b$th bit of the $j$th scalar symbol in partial symbol vectors, only $\{\mathbf{s}^i\}$ with $i \in \{1, \ldots, j\}$ are influenced, and $\{\mathbf{s}^i\}$ with $i \in \{j+1, \ldots, Nt\}$ remain unchanged. Correspondingly, we only need to calculate Eqn.(5) for $i \in \{1, \ldots, j\}$. Such a selective updating significantly reduces the number of times that Eqn.(5) are calculated.

Furthermore, we can rewrite a part of Eqn.(5) as:

$$\begin{aligned}
\|e_i(\mathbf{s}^{i(0)}_{j,b})\|_1 &= \|\hat{y}_i - \sum_{k=i}^{Nt} R_{ik} s_k + R_{ij}(s_j - s^0_{j,b})\|_1 \\
&= \|e_i(\mathbf{s}^i) - \underbrace{R_{ij}(s^0_{j,b} - s_j)}_{\triangle e^{i(0)}_{j,b}}\|_1 \\
\|e_i(\mathbf{s}^{i(1)}_{j,b})\|_1 &= \|e_i(\mathbf{s}^i) - \underbrace{R_{ij}(s^1_{j,b} - s_j)}_{\triangle e^{i(1)}_{j,b}}\|_1 \qquad (6)
\end{aligned}$$

The motivation behind Eqn.(6) is that $e_i(\mathbf{s}^i)$ has already been calculated by the list generator . In addition, only one scalar symbol in partial symbol vectors is modified. If the intermediate results of $e_i(\mathbf{s}^i)$ are stored in a buffer accessible by the LLR generator, we only need to calculate $\triangle e^{i(0)}_{j,b}$, $\triangle e^{i(1)}_{j,b}$ and the intermediate results of $e_i(\mathbf{s}^i)$ can be reused. With the partial updating, the complexity for calculating $\|e_i(\mathbf{s}^{i(0)}_{j,b})\|_1$ and $\|e_i(\mathbf{s}^{i(1)}_{j,b})\|_1$ is further substantially reduced.

#### 3.2.2 Algebraic Simplifications and Strength Reductions

Practical communication systems adopt gray-coded modulation schemes. For most popular gray-coded modulation schemes, a specific bit of the input data changes only the I or Q branch of the modulated/mapped signal. For instance, in 3GPP LTE and IEEE802.15-2006, the $N_b/2$ most significant ones of the $N_b$ total bits determine the position of the modulated signal on the I-axis, and the $N_b/2$ least significant bits the position on the Q-axis. Hence, $s^0_{j,b} - s_j$ and $s^1_{j,b} - s_j$ both have *at least* real or image part being zero. We can exploit the above observation to apply algebraic simplifications.

Let $\beta_b(s_j)$ denote the $b$th bit in the scalar symbol $s_j$, and $\gamma(b)$ denote the *shifted distance* of constellations due to flipping the $b$th bit of $s_j$ from 0 to 1. Note that $\gamma(b)$ is just a real number. As $\beta_b(s_j) \in \{0, 1\}$, $\triangle e^{i(0)}_{j,b}$ or $\triangle e^{i(1)}_{j,b}$ must have one of them being 0. Hence, we have only 4 possible cases:

- $b < N_b/2$ and $\beta_b(s_j) = 0$: $\triangle e^{i(0)}_{j,b} = 0$, $\Re(\triangle e^{i(1)}_{j,b}) = -\Im(R_{ij})\gamma(b)$, $\Im(\triangle e^{i(1)}_{j,b}) = \Re(R_{ij})\gamma(b)$
- $b < N_b/2$ and $\beta_b(s_j) = 1$: $\Re(\triangle e^{i(0)}_{j,b}) = \Im(R_{ij})\gamma(b)$, $\Im(\triangle e^{i(0)}_{j,b}) = -\Re(R_{ij})\gamma(b)$, $\triangle e^{i(1)}_{j,b} = 0$
- $b \geq N_b/2$ and $\beta_b(s_j) = 0$: $\triangle e^{i(0)}_{j,b} = 0$, $\Re(\triangle e^{i(1)}_{j,b}) = \Re(R_{ij})\gamma(b)$, $\Im(\triangle e^{i(1)}_{j,b}) = \Im(R_{ij})\gamma(b)$
- $b \geq N_b/2$ and $\beta_b(s_j) = 1$: $\Re(\triangle e^{i(0)}_{j,b}) = -\Re(R_{ij})\gamma(b)$, $\Im(\triangle e^{i(0)}_{j,b}) = -\Im(R_{ij})\gamma(b)$, $\triangle e^{i(1)}_{j,b} = 0$

The major computations in above are $\Re(R_{ij})\gamma(b)$ and $\Im(R_{ij})\gamma(b)$. The multiplications are be converted to simple bit-shifts and additions if the original input signal $\mathbf{y}$ is properly scaled. In commercial systems, modulations constellations of QAM are often scaled for normalized average power. If we cancel the scaling at receiver side and restore the original constellations, $\Re(R_{ij})\gamma(b)$ and $\Im(R_{ij})\gamma(b)$ can be converted to bit-shifts and additions. For gray-coded 16QAM and 64QAM schemes, $|\gamma(b)| \in \{2, 4, 6, 8, 10, 12, 14\}$, so that $\times|\gamma(b)|$ can be efficiently implemented with at most two bit-shifts and one addition/substraction. The above algebraic simplifications and strength reductions require corresponding ASIs supported in targeted architectures. We will present ASIs in the coming section.

## 4. ASIs and Explorations of ADRES Template

In this section, we will introduce optimizations and explorations on the architecture side. First we will present the design of ASIs, and then ILP/DLP explorations of ADRES ASIP template.

**Table 1. Application Specific Instructions**

| | ASI0 | ASI1 | ASI2 | ASI3 |
|---|---|---|---|---|
| 16b MUL | 0 | 0 | 0 | 2 |
| 16b ADD | 8 | 7 | 8 | 12 |
| 16b ABS | 2 | 2 | 0 | 0 |
| 4b SHIFT | 4 | 4 | 8 | 4 |
| MUX(2 to 1) | 4 | 0 | 0 | 4 |

### 4.1. Application Specific Instructions

An ASI implements a large cluster of connected operators. Since the computations of a SOMLD are dominated few equations such as Eqn.(5) and Eqn.(6), we can design ASIs to greatly improve the efficiency of implementation. First, comparing to performing only one operator in each cycle, packing cascaded operators into one instruction allows much more operators in one clock cycle. In addition, both the number of instructions and intermediate storage requirement will be significantly reduced. Furthermore, ASIs can reduce the size of optimization problems for compilers, so that compilers can allocate and schedule resources in a more effective and efficient way.

#### 4.1.1 Overview of The Design

In our work, ASIs are designed based on algorithmic insights. Computations in the critical path and dominant computations are packed as ASIs. In total we designed 4 ASIs (ASI0-3): ASI0 is designed for the LLR generation; ASI1 is designed for symbol generation at antenna $Nt$; ASI2 and ASI3 are designed for symbol generation at other antennas.

The number of components in required to implement each ASI is summarized in Table 1. We can notice that these ASIs pack a large number of operators inside. In addition, due to the architecture friendly algorithm design, these ASIs are based on mostly low cost operators. They require no multipliers except ASI3. Although ASI0-3 all appear in the critical path of the proposed SOMLD, ASI0 is the most important one in terms of number of executions. Specifically, when the detector is configured for 2×2 64QAM, to detect one MIMO symbol, ASI0 is executed 1152 times whereas ASI1-3 are executed 64 times each. Hence, in the following, we will introduce ASI0 in detail and link it to the algorithmic descriptions in section 3.2.

#### 4.1.2 Details of ASI0

The micro-architecture of ASI0 are illustrated in Fig.1. The Re() and Im() denote the real and image part of a variable, respectively. PMD_inc_c denotes $e_i(\mathbf{s}^i)$. With converted shift-add operations, block0 calculates $\Re(R_{ij})|\gamma(b)|$ and $\Im(R_{ij})|\gamma(b)|$ as required for $\triangle e_{j,b}^{i(0)}$ and $\triangle e_{j,b}^{i(1)}$. Designed for Eqn.(6), block1 can calculate $|(\Re(e_i(\mathbf{s}^i)) \pm \Re(R_{ij})|\gamma(b)|)|$, $|(\Im(e_i(\mathbf{s}^i)) \pm \Im(R_{ij})|\gamma(b)|)|$ or $|(\Im(e_i(\mathbf{s}^i)) \pm \Re(R_{ij})|\gamma(b)|)|$, $|(\Re(e_i(\mathbf{s}^i)) \pm \Im(R_{ij})|\gamma(b)|)|$. Block2 calculates $T_{j,b}^{i(0)}$ and $T_{j,b}^{i(1)}$ as formulated by a part of Eqn.(5).

Inside ASI0, there are 7 control signals enabling the required flexibility. These signals are generated according to

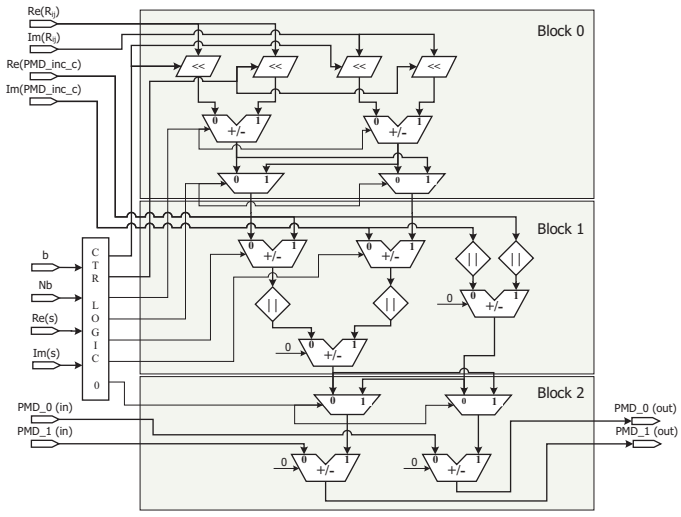


**Figure 1. ASI for LLR Generation (ASI0)**

$Nb$, $b$ and $s_{j,b}$. Importantly, a control signal generator is included as a part of the ASI. The generator can be implemented as combinational logics or lookup tables, and in our work we use combinational logic. Since ASI0 is completely based on low cost and low latency operators such as shift and addition, the latency of ASI0 is very short. In the next section, we will present detailed results synthesized with TSMC 90nm technology libraries.

### 4.2. The ADRES ASIP Template and ILP/DLP Explorations

Our work is based on the ADRES ASIP template [8]. As shown in Fig.2, the parameterizable template consists of an array of densely interconnected FUs that have local RF (Register Files) and configuration memory (loop buffer). A limited subset of those units is connected to a global (shared) RF, enabling their operation also as a standard VLIW processor. All FUs support SIMD. A retargetable C compiler, named as DRESC, targets both the VLIW and CGA modes. DRESC maps loops on the CGA whereas schedules the rest code on the VLIW part.

With the ADRES template, we can design an ASIP with massive parallelism by combining ILP and DLP. The designer needs to dimension ILP and DLP according to application requirement. By changing the size of the array (number of FUs), we can tune the amount of supported ILP. By changing the number of SIMD slots in each FU, we can tune the amount of supported DLP. In our work, the ILP/DLP exploration targets a minimum throughput (Mbps) delivered by the ASIP. In the next section, we will present detailed results.

## 5. Experiment Results

The proposed SOMLD is implemented as manually written low level C code. In experiments, first we investigate the proposed algorithms and the reference algorithm on

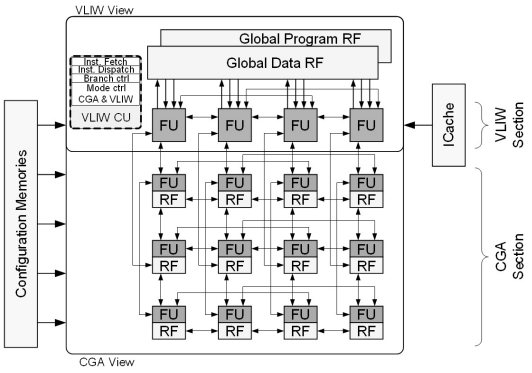


**Figure 2. The ADRES ASIP Template**

TMS320C6416. This is to study the algorithmic improvement without ASIs on a state of the art commercial VLIW DSP. Then, we investigate the SOMLD on ADRES ASIP with ASIs. The latency and cycles of ASIs are estimated with synthesis results on TSMC 90nm libraries.

Our work targets minimum 120 Mbps throughput for 2×2 64QAM transmissions. A previously reported linear MIMO receiver based on ADRES delivers similar throughput [1]. When the ASIP is designed for 120+ Mbps with 2×2 64QAM, it will deliver 150% to 200% throughput when configured for 2×2 16QAM. As that in [1], we focus on 2×2 64QAM and 16QAM systems because their popularity. Although 4×4 systems are also specified in some standards, but the cost of RF components, analog circuits and antennas is much higher.

Our work targets both high throughput and strong communication performance (BER). Hence, the search range (size of the list) in the SOMLD is configured to make sure that the SNR degradation is less than 0.1dB when compared to an exhaustive search, which achieves the best BER with soft-output. Specifically, the generated list by the SSFE has 16 candidates for 16QAM, and 64 candidates for 64QAM. Tested MIMO channel models are 3GPP/3GPP2 SCM (Spatial-Channel-Models): Suburban macro, Urban macro and Urban micro. Due to the limited space, BER curves are not included in this paper.

## 5.1. On TMS320C6416

TMS320C6416 is a representative VLIW DSP in the market. It supports 8 32b instructions per-cycle, controlling 8 parallel FUs organized as 2 clusters. Level-1 memory consists of 16 K-Byte direct-mapped instruction cache (L1P) and 16 K-Byte 2-way set-associative data cache (L1D). In Table 2, we list the statistics for decoding one 2×2 64QAM MIMO symbol. LLR V0 is the LLR generator based on the triangular form and bit-flipping techniques as shown in Eqn.(4), LLR V1 is the optimized LLR generator in our work. All components are mapped on the TMS320C6416 with compiler-scheduled software pipelining.

First, from the number of instructions, we can observe that that the complexity of LLR generation is indeed dominant. Second, the proposed algorithm significantly reduces the number of instructions, L1D accesses and L1D misses. Remarkably, the number of L1D accesses is reduced by a factor of 100+. We can notice that the cycle count of LLR V1 is actually higher than that of LLR V0, despite the reduced number of instructions in LLR V1. The reason is that, without ASIs, the innermost loop of LLR V1 requires too many instructions and too many live registers, so that the TI compiler failed to optimize it with software pipelining. However, if we assume that the TI tool can map it very efficiently with IPC≈6 (as that in the list generator), the SOMLD on TMS320C6416 still requires 13841 cycles to decode one MIMO symbol. This reaches only 2% of the required throughput (120+ Mbps) even with the 4-way SIMD supported by TMS320C6416. This clearly shows that we need an ASIP with ASIs and more parallelism.

**Table 2. Statistics on TMS320C6416**

| | List Generator | LLR V0 | LLR V1 |
|---|---|---|---|
| Instructions | 6209 | 234898 | 76834 |
| Cycles | 1057 | 57189 | 59479 |
| L1D Accesses | 791 | 52651 | 11236 |
| L1D Misses | 96 | 1260 | 12 |

## 5.2. On ASIP

### 5.2.1 ASIs, Code Transformations and Effects

We implement ASIs with VHDL and then synthesize with TSMC 90nm libraries (TCBN90LPHP). ASIs in our work are based on area efficient operators. The dominant ASI (ASI0) has no multipliers. Hence, we mainly study their latency and cycle counts. In Table 3, the listed cycle counts are calculated with ADRES clocked at 400Mhz, as demonstrated with synthesis results in [1] and recent measurements. The calculated cycle counts take into account the overhead for integrating ASIs into FUs, such as large multiplexers inside FUs.

**Table 3. ASIs with TSMC90nm(TCBN90LPHP)**

| | ASI0 | ASI1 | ASI2 | ASI3 |
|---|---|---|---|---|
| Delay | 2.73ns | 2.71ns | 2.48ns | 3.02ns |
| Cycles@400Mhz | 2 | 2 | 2 | 3 |

Since ASIs significantly reduce the number of instructions and live registers, we can further apply pre-compiler code transformations to improve the efficiency. Instead of separating the list generator from the LLR generator and connect them by buffers, we merge loops in the two components aided by loop collapsing. In this way, the locality of data access is substantially improved. For decoding one 2×2 64QAM MIMO symbol, the transformed code with ASIs requires only 2577 instructions and 36 L1D accesses, whereas the original code requires 76834 instructions and 11236 L1D accesses on TMS320C6416. This shows nearly 30× improvement for the number of instructions, and more than 300× improvement for memory accesses. The comparison strongly proves the advantages of using ASIs and accordingly applying code transformations.

**Table 4. ILP/DLP Explorations Targeting 120+ Mbps Throughput with 2×2 64QAM**

| | | 64QAM | | | | | 16QAM | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Array | SIMD | Cycles | CGA Cycles | IPC | Total Mbps | Mbps/FU/SIMD | Cycles | CGA Cycles | IPC | Total Mbps | Mbps/FU/SIMD |
| 2×4 | 16 | 397 | 355 | 6.8 | 12.1×16=192.6 | 1.51 | 139 | 99 | 5.6 | 23.0×16=368.0 | 2.88 |
| 4×4 | 8 | 276 | 227 | 11.4 | 17.4×8=139.2 | 1.09 | 118 | 75 | 8.2 | 27.1×8=216.8 | 1.69 |
| 6×4 | 8 | 228 | 183 | 15.5 | 21.1×8=168.8 | 0.88 | 105 | 61 | 10.6 | 30.5×8=244.0 | 1.27 |
| 8×4 | 8 | 212 | 163 | 18.1 | 22.6×8=180.8 | 0.71 | 98 | 54 | 11.4 | 32.7×8=261.6 | 1.02 |

#### 5.2.2 The ILP/DLP Exploration

Although emerging SDR platforms are based on multiple processors, our work combines the architecture friendly algorithm, ASIs, code transformations to enable a rate-compatible SOMLD implementation on only one processor. Other processors can be used to run the rest of a transceiver. Results of ILP/DLP exploration are summarized in Table 4. The ILP/DLP exploration is combined with loop transformations to improve the scheduling density on a chosen architecture. During the exploration, all FUs in the array are enabled to support ASI0, whereas ASI0-3 are supported in only 1 VLIW FU. Such a design is based on the fact that ASI0 is dominant in terms of the number of executions. We use very wide SIMD slots like that in EVP (16-way) [11] SODA (32-way) [7]. Since the proposed algorithm is explicitly designed for such DLP architectures, the SOMLD implementation can be easily vectorized for SIMD with zero overhead. Adding more SIMD slots brings linear throughput growth.

In Table 4, we list cycle count, CGA cycle count, IPC (Instruction Per-Cycle), throughput (Mbps) and average Mbps achieved by each SIMD slot and each FU (Mbps/FU/SIMD). As a part of the total cycle count, CGA cycles are spent on the CGA section (refer to Fig.2). IPC indicates how much ILP is actually utilized. The metric Mbps/FU/SIMD indicates at system level how efficiently architecture resources are utilized.

First of all, we can achieve 120+ Mbps an ASIP with its scale comparable to other SDR processors. For instance, the NXP EVP processor has 10 FUs and 6 of them support 16-way SIMD. In our work, a 2×4 array with 16-way SIMD gives 368Mbps for 2×2 16QAM and 193Mbps for 2×2 64QAM. In addition, we can observe that the efficiency of resource utilization decreases when enlarging the array. For instance, For 2×2 64QAM, a 2×4 array achieves IPC=6.8 with scheduling density 85%. However, a 4×4 array achieves IPC=11.4 with with scheduling density 71.25%. The Mbps/FU/SIMD metric gives a similar indication as well. Such a phenomenon is due to the fact that, the increment of array size brings exponentially increased complexity for the compiler, so that generating optimized resource allocation and schedule is more difficult. Among the explored options, the 2×4 array with 16-way SIMD gives the best throughput and the best scheduling density (85%).

## 6. Conclusions and Future Work

In this paper, we presented the algorithm/architecture co-design of a SOMLD on massive parallel ASIPs. We combine architecture friendly algorithm design, ASI design, code transformations and ILP/DLP explorations to enable efficient implementations. On an ADRES based ASIP with its scale comparable to existing SDR processors, the proposed SOMLD can achieve throughput compatible with emerging standards. To the best of our knowledge, this is the first work exploring SDR based SOMLD implementations.

## References


[1] B. Bougard, B. De Sutter, S. Rabou, D. Novo, O. Allam, S. Dupont, and L. Van der Perre. A coarse-grained array based baseband processor for 100mbps+ software defined radio. *Design, Automation and Test in Europe, 2008. DATE '08*, pages 716–721, 10-14 March 2008.

[2] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei. Vlsi implementation of mimo detection using the sphere decoding algorithm. *Solid-State Circuits, IEEE Journal of*, 40(7):1566–1577, July 2005.

[3] S. Chen, T. Zhang, and Y. Xin. Relaxed k -best mimo signal detector design and vlsi implementation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(3):328–337, March 2007.

[4] Z. Guo and P. Nilsson. A vlsi architecture of the schnorr-euchner decoder for mimo systems. *Emerging Technologies: Frontiers of Mobile and Wireless Communication, 2004. Proceedings of the IEEE 6th Circuits and Systems Symposium on*, 1:65–68 Vol.1, May-2 June 2004.

[5] B. Hochwald and S. ten Brink. Achieving near-capacity on a multiple-antenna channel. *Communications, IEEE Transactions on*, 51(3):389–399, March 2003.

[6] M. Li, B. Bougard, W. Xu, D. Novo, L. Van Der Perre, and F. Catthoor. The optimization of near-ml mimo detector for sdr baseband on parallel programmable architectures. *The IEEE/ACM Design Automation and Test in Europe (DATE) 2008*, pages 1–6, Nov. 2006.

[7] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. Soda: A high-performance dsp architecture for software-defined radio. *IEEE Micro*, 27(1):114–123, 2007.

[8] B. Mei, A. Lambrechts, D. Verkest, J.-Y. Mignolet, and R. Lauwereins. Architecture exploration for a reconfigurable architecture template. *IEEE Des. Test*, 22(2):90–101, 2005.

[9] A. Nilsson, E. Tell, and D. Liu. An 11mm2 70mw fully-programmable baseband processor for mobile wimax and dvb-t/h in 0.12m cmos. *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 266–612, Feb. 2008.

[10] U. Ramacher. Software-defined radio prospects for multi-standard mobile phones. *Computer*, 40(10):62–69, 2007.

[11] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP J. Appl. Signal Process.*, 2005(1):2613–2625, 2005.