Software-Defined Sphere Decoding for FPGA-Based MIMO Detection

Xuezheng Chu, *Member, IEEE*, and John McAllister, *Member, IEEE*

***Abstract*—Sphere Decoding (SD) is a highly effective detection technique for Multiple-Input Multiple-Output (MIMO) wireless communications receivers, offering quasi-optimal accuracy with relatively low computational complexity as compared to the ideal ML detector. Despite this, the computational demands of even low-complexity SD variants, such as Fixed Complexity SD (FSD), remains such that implementation on modern software-defined network equipment is a highly challenging process, and indeed real-time solutions for MIMO systems such as 4 × 4 16-QAM 802.11n are unreported. This paper overcomes this barrier. By exploiting large-scale networks of fine-grained software-programmable processors on Field Programmable Gate Array (FPGA), a series of unique SD implementations are presented, culminating in the only single-chip, real-time quasi-optimal SD for 4 ×4 16-QAM 802.11n MIMO. Furthermore, it demonstrates that the high performance software-defined architectures which enable these implementations exhibit cost comparable to dedicated circuit architectures.**

***Index Terms*—FPGA, MIMO, multicore, sphere decoder.**

I. Introduction

Multiple-input, multiple-output (MIMO) communications systems [1] exploit spatial diversity to provide wireless communications channels of unprecedented capacity and throughput, prompting their adoption in wireless communications standards such as 802.11n [2]. A generic MIMO system employing M transmit and N receive antennas is shown in Fig. 1.

Effectively harnessing the benefits of MIMO technology, however, relies on the existence of accurate, high throughput receiver equipment—a very significant embedded architecture design problem. This difficulty is due to two main factors. Firstly, the high computational complexity of accurate detector algorithms such as Sphere Decoders (SDs) is apparent in the current absence of reported real-time implementations for even moderate MIMO systems, such as the 4 × 4 16-QAM topologies employed in 802.11n [3]–[8]. Furthermore, such realizations should ideally be 'software-defined', for integration in modern network equipment and design processes [9], [10], whilst current implementations of parts of SD algorithms require custom circuit architectures to achieve real-time processing. Combining these two to achieve real-time, software-defined detection is a highly challenging implementation problem.

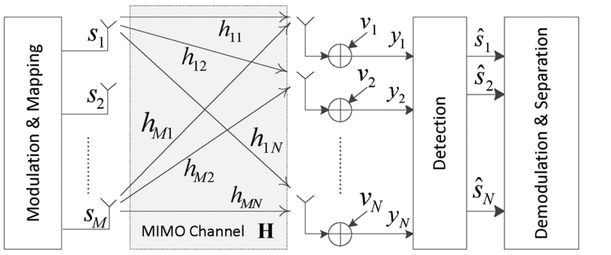


Fig. 1. Generic MIMO communication system.

This paper presents a unique design approach which overcomes these barriers. It extends the work in [11], [12] to create a series of unique real-time, software-defined SD processing architectures for 4 × 4 16-QAM 802.11n MIMO. Specifically, four contributions are made:

1) A highly efficient, software-defined FPGA processing architecture for baseband DSP [11], [12] is presented.
2) The architecture from 1) is used to create the first known real-time preprocessing architecture for 802.11n FSD.
3) It is shown how 1) enables the only known software-defined FSD metric calculation and sorting architecture.
4) The implementations from 2) and 3) are combined to create the only known full FSD detector for 4 × 4 16-QAM 802. 11n.

The remainder of this paper is organized as follows. Section III describes the software-defined FPGA processing paradigm, before it is used to create real-time architectures for preprocessing and metric calculation and sorting in Sections IV and V respectively. Finally, Section VI exploits this approach to create the only recorded single-chip, real-time SD architecture for 4 × 4 16-QAM 802.11n MIMO.

II. Background and Motivation

In an M-transmit, N-receive antenna MIMO system, the M-element transmitted symbol vector $\mathbf{s}$ suffers multipath distortion and noise corruption $(\mathbf{v})$ when propagating across the channel to the receiver. Hence the N-element received symbol vector $\mathbf{y}$ is formulated mathematically as (1), where $\mathbf{H} \in \mathbb{C}^{N\times M}$ represents the MIMO channel, used typically as a parallel set of flat-fading subchannels via Orthogonal Frequency Division Multiplexing (OFDM).

$$\mathbf{y} = \mathbf{Hs} + \mathbf{v} \tag{1}$$

Manuscript received February 09, 2012; revised May 01, 2012 and June 22, 2012; accepted June 24, 2012. Date of publication July 31, 2012; date of current version October 09, 2012. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Jarmo H. Takala. This work is supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC), under Grant EP/F031017/1.

The authors are with the Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, Belfast BT7 1NN, United Kingdom (e-mail: xchu01@qub.ac.uk; jp.mcallister@qub.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2012.2210951

SD is a receiver baseband signal processing approach employed to estimate $\mathbf{s}$. It offers near-ideal detection performance with significantly reduced computational complexity relative to the ideal ML detector [13], [14]. Despite this, SD algorithms in general remain computationally complex and present a significant implementation challenge, particularly in base-station equipment where demanding real-time performance metrics must be met, such as the maximum 480 Mbps, 4 μs latency required by 4×4 16-QAM 802.11n MIMO [2]. In the context of the industry-wide move toward software-programmable or 'software-defined' DSP architectures which emphasize flexibility to support multiple radio standards along with implementation cost and performance [9], this is a challenging real-time implementation problem; even recorded custom circuit architectures have not proven capable of supporting real-time quasi-ML SD for 4×4 16-QAM 802.11n [3], [5], [15]–[18].

A range of simplified SD variants have emerged in an attempt to alleviate this complexity problem whilst maintaining quasi-ML detection accuracy. Amongst these, Fixed-Complexity SD (FSD) is exceptional since it uniquely combines relatively low complexity, deterministic behavior and quasi-ML accuracy [19]. FSD has a two-phase behavior:

1) *Pre-Processing (PP)*: The symbols of $\mathbf{y}$ are ordered for detection and the centre of the decoding sphere is initialized using Zero Forcing detection.
2) *Metric Calculation & Sorting (MCS)*: An M-level decode tree performs a Euclidean distance based statistical estimation of $\mathbf{s}$.

PP orders the received symbols according to the perceived distortion experienced by each. This is achieved by reordering the columns of $\mathbf{H}$ to give $\mathbf{H}^{\dagger}$ (the general form of which is illustrated in Fig. 2(a)) via an M-phase iterative process:

1) Calculate $\mathbf{W}_i$ according to (2), where $\mathbf{H}_i$ is the channel matrix with previously selected columns zeroed.

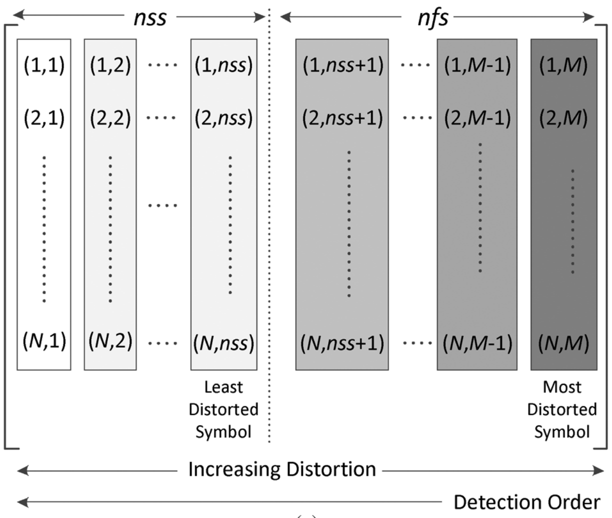
$$\mathbf{W}_i = \left(\mathbf{H}_i^{-H}\right)\mathbf{H}_i^{-1} \tag{2}$$

2) The signal $\hat{s}_k$ to be detected is selected according to

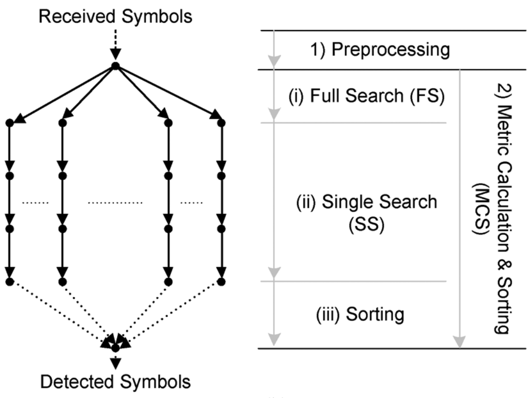
$$k = \begin{cases} \arg\max_j \left\| \left(\mathbf{H}_i^{\dagger}\right)_j \right\|^2 & \text{if } n_i = P \\ \arg\min_j \left\| \left(\mathbf{H}_i^{\dagger}\right)_j \right\|^2 & \text{if } n_i \neq P \end{cases}$$

Post-ordering, groups of M symbols undergo detection via a tree-search structure illustrated in Fig. 2(b). The node distribution at each level in the tree is given by $\mathbf{n}_S = (n_1, n_2, \ldots, n_M)^T$. During the first nfs levels, the worst distorted symbols undergo *Full Search* (FS), where the search space is fully enumerated resulting in P child nodes at level $i+1$ per node at level i, where P is the number of QAM constellation points. The remaining $nss(nss = M - nfs)$ least distorted symbols subsequently undergo *Single Search* (SS), where only a single candidate detected symbol is maintained between layers. For full diversity, nfs is given by (3). At each MCS tree level, (4) and (5) are performed.

$$nfs = \lceil \sqrt{M} - 1 \rceil. \tag{3}$$



(a)



(b)

Fig. 2. FSD algorithm components. (a) General form of $\mathbf{H}^{\dagger}$; (b) FSD tree search structure.

$$\tilde{s}_i = \hat{s}_{ZF,i} - \sum_{j=i+1}^{M_t} \frac{r_{ij}}{r_{ii}} (\hat{s}_{ZF,j} - \hat{s}_j) \tag{4}$$

$$d_i = \sum_{j=i}^{M_t} r_{ij}^2 \| \hat{s}_{ZF,j} - \hat{s}_j \|^2, D_i = d_i + D_{i+1} \tag{5}$$

In (4) and (5), r_{ij} refers to an entry in $\mathbf{R}$, obtained via QR decomposition of $\mathbf{H}$ during PP, $\hat{s}_{ZF}$ is the center of the constrained FSD sphere and $\tilde{s}_j$ is the j^{th} detected data, which is sliced to $\hat{s}_j$ in subsequent iterations of the detection process [20]. Since D_{i+1} can be considered as the Accumulated Partial Euclidean Distance (APED) at level $j = i+1$ of the MCS tree and d_i as the PED in level i, the APED can be obtained by recursively applying (5) from level $i = M$ to $i = 1$. The resulting candidate symbols are sorted based on their Euclidean distance measurements, and the final result produced post-sorting. In 802.11n MIMO, this collective behaviour must be replicated 108 times, once per OFDM subcarrier employed.

FSD is amongst the lowest complexity, quasi-optimal SD algorithms known [21]; along with the highly parallel nature of FSD this has proven effective in enabling real-time FSD MCS [22]. However, three outstanding issues remain in real-time software-defined FSD for standards such as 802.11n:

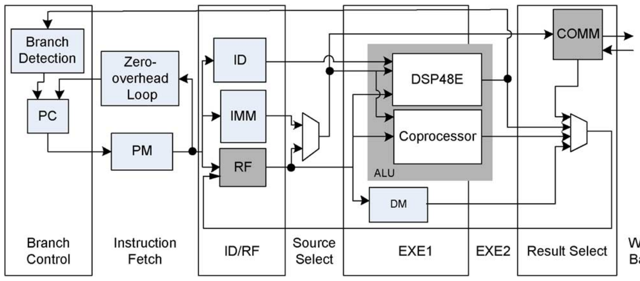


Fig. 3. The FPE architecture.

1) Single-chip detectors remain elusive: there is no recorded real-time architecture which integrates both PP and MCS for all 108 802.11n OFDM subcarriers.
2) The high computational complexity of PP (the M iterations of the $O(M^3)$ pseudo-inverse in (2) results in a $O(M^4)$ algorithm) has, to date, prohibited real-time implementation of even PP.
3) Existing real-time MCS realisations [22] rely on custom dedicated circuits, whilst modern equipment design processes require software-defined architectures.

Whilst technologies such as Field Programmable Gate Array (FPGA) are computationally capable of hosting real-time MCS at least, two key issues currently prevent software-defined FPGA architectures from resolving this problem:

1) Software-defined FPGA architectures, e.g. [23], [24] are too costly and low performance to meet the real-time demands of 802.11n FSD.
2) FSD detection of all 108 OFDM subcarriers in 802.11n is a large scale operation, requiring a highly scalable processing architecture.

To resolve this issue, a new approach to software-defined realization of SD is required. This paper presents a unique solution which demonstrates the viability of real-time, software-defined MIMO detection on FPGA, by realizing FSD PP, MCS and full detector architectures which meet the 480 Mbps, 4 μS latency requirements of 802.11n. Further, we show how the resulting realisations exhibit cost comparable to custom circuit solutions. Section III describes the processing architecture exploited, before its effectiveness for FSD PP, MCS and full detection are described in Sections IV, V and VI respectively.

III. The FPGA-Based Processing Element (FPE)

The emergence of components such as the DSP48E on recent generations of Xilinx FPGA offer unprecedented levels of computational capacity enclosed in programmable datapath components. Their programmability implies the need for data storage and circuitry for datapath control, but despite modern FPGA housing plentiful resources with which to realize these, in the form of Look Up Tables (LUTs) and Block RAM (BRAMs), existing FPGA processors are typically resource hungry and performance limited. Hence whilst modern FPGA house very high levels of programmable computational capacity, software-defined architectures capable of exploiting these resources are lacking. A unique, lean processing architecture known as the *FPGA Processing Element* (FPE) is proposed to resolve this deficiency. The architecture of the FPE is shown in Fig. 3.

TABLE I
FPE Configuration Parameters

Parameter	Meaning	Values
DataWidth	Data wordsize	16/32 bits
DataType	Type of data	Real/complex
ALUWidth	No. DSP48E slices	1 - 4
PMDepth/PMWidth	PM Capacity/Instrn. Width	Unlimited
DMDepth/RFDepth	DM/ RF Capacity	Unlimited
TxCOMM/RxCOMM	No. Tx/Rx ports	$\leq$1024
IMMDepth/IMM Width	No./size immediate data memory locations	Unlimited

TABLE II
FPE Instruction Set

	Instruction	Function
CTRL	LOOP/RPT	loop/repeat
	BEQ/BGT/BLT	branch if equal/greater/less
	JMP	jump
	GET/PUT	load/push data from/to channel
	GETCH/CLRCH	load data from/clear channels
	NOP	no operation
ALU	MUL/ADD/SUB	multiply/add/subtract
	MULADD/MULSUB(FWD)	multiply-add/subtract (& forward)
	COPROC	coprocessor access
MEM	LD/ST	load/store data from/to memory
	LDIMM/STIMM	load/store data from/to IMM
	LDIAR	updata IMM address register

The FPE houses the minimum set of resources required for programmable operation: the instructions pointed to by the *Program Counter* (PC) are loaded from *Program Memory* (PM) and decoded by the *Instruction Decoder* (ID). Data operands are read either from *Register File* (RF), or in the case of immediate data *Immediate Memory* (IMM) and processed by the ALU (implemented using a Xilinx DSP48E). In addition, a *Data Memory* (DM) is used for bulk data storage and a *Communication Adapter* (COMM) performs on/off-FPE communications.

The FPE is *configurable* such that its architecture can be customized pre-synthesis in terms of the aspects listed in Table I; in addition, the ALU can be extended with custom coprocessors, to accelerate critical operations. Further, the FPE is *programmable* via the instruction set described in Table II; it is currently programmed manually at the assembly level, and the instruction set is extensible to incorporate new instructions for specific coprocessors. When implemented on Xilinx Virtex 5 VLX110T FPGA, the computational capability and cost of six FPE configurations—16 bit Real (*16R*), 32 bit Complex (*32C*) and 32 bit Real (*32R*) variants—are as described in Table III[1].

Table III describes a range of performance/cost metrics that, to the best of the authors' knowledge, are unmatched in any other software-defined FPGA architecture; for instance, the FPE occupies only 18% of the resource of a conventional MicroBlaze processor, whilst enabling a factor 2.8 increase in computational capacity. These metrics imply that the FPE is the most

[1]All synthesis results are post place-and-route, employing flat criteria, with neither speed nor area prioritized.

TABLE III
FPE ARITHMETIC PERFORMANCE

Config	Resource		Latency	Clock	Throughput
	LUTs	DSP48Es	(Cycles)	(MHz)	(MMACs/s)
16 R	90	1	4	483	483
16 C	132	1	7	476	119
	172	2	5	453	226.5
	140	4	5	474	474
32 R	185	2	6	431	215.5
	182	3	7	431	431

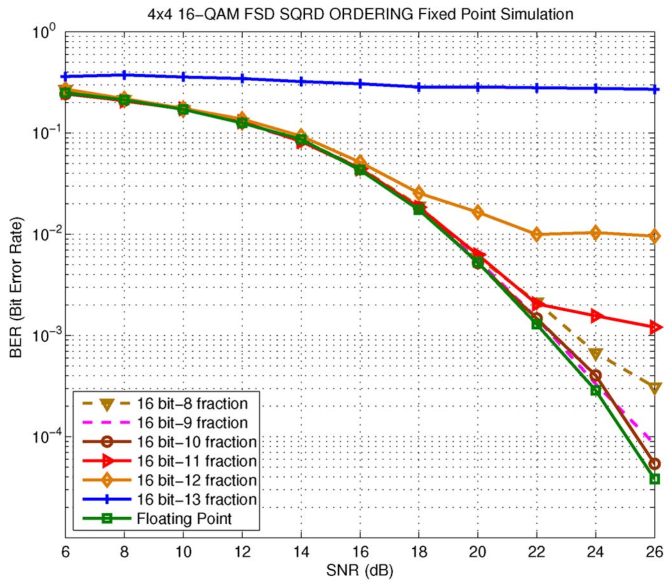


Fig. 4. 4×4 16-QAM fixed point simulation.

likely of any software-defined FPGA architecture to support real-time FSD. Sections IV–VI examine implementation of PP, MCS and full detector architectures using the FPE processing paradigm.

IV. FPE-BASED PRE-PROCESSING USING SQRD

Section II described how the complexity of SD PP has, to date, prohibited real-time implementation for MIMO systems such as 4×4 802.11n. However the recent emergence of sub-optimal PP algorithms such as Sorted QRD (SQRD) [25], [26] can potentially overcome this issue. This section tests the ability of the FPE to support real-time SQRD-based PP for FSD.

SQRD-based ordering for FSD transforms the input channel matrix $\mathbf{H}$ to the product of a unitary matrix $\mathbf{Q}$ and an upper-triangular $\mathbf{R}$ via QR decomposition, whilst deriving $\mathbf{order}$, the order of detection of the received symbols during MCS. It operates in two phases, as described in Algorithm 1 [26]. In Phase 1 $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{order}$, $\mathbf{norm}$ and nfs are initialized as shown in lines 2–5 of Algorithm 1, where $\mathbf{q}_i$ is the i^{th} column of $\mathbf{Q}$. Phase 2 comprises M iterations, in each of which the k^{th} lowest entry in $\mathbf{norm}$ is identified (lines 9 & 10) before the corresponding column of $\mathbf{R}$ and elements in $\mathbf{order}$ and $\mathbf{norm}$ are permuted with the i^{th} (line 11) and orthogonalized (line 12–18). The resulting $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{order}$ are used for FSD MCS as defined in (4) and (5). Note the merged ordering and QRD of $\mathbf{H}$ in Phase 2; this avoids the M iterations of QRD in V-BLAST, enabling an observed order-of-magnitude complexity reduction for FSD PP.

Algorithm 1: Sorted QR decomposition for FSD

```
input: $\mathbf{H}, M$
output: $\mathbf{Q}, \mathbf{R}, \mathbf{order}$
1 Phase 1: Initialization
2 $\mathbf{Q} = \mathbf{H}, \mathbf{R} = \mathbf{0}_M$,
3 $\mathbf{order} = [1, \cdots, M], nfs = \lceil \sqrt{M} - 1 \rceil$
4 for $i \leftarrow 1$ to $M$ do
5     $\mathbf{norm}_i = \|\mathbf{q}_i\|^2$
6 end
7 Phase 2: SQRD ordering
8 for $i \leftarrow 1$ to $M$ do
9     $k = min(nfs + 1, M - i + 1)$
10    $k_i = \arg \overset{k}{\min}_{j=i,\cdots,M} \mathbf{norm}_j$
11    Exchange columns $i$ and $k_i$ in $\mathbf{R}, \mathbf{order}, \mathbf{norm}$ and $\mathbf{Q}$
12    $r_{i,i} = \sqrt{\mathbf{norm}_i}$
13    $\mathbf{q}_i = \frac{\mathbf{q}_i}{r_{i,i}}$
14    for $l \leftarrow i + 1$ to $M$ do
15        $r_{i,l} = \mathbf{q}_i^H \cdot \mathbf{q}_l$
16        $\mathbf{q}_l = \mathbf{q}_l - r_{i,l} \cdot \mathbf{q}_i$
17        $\mathbf{norm}_l = \mathbf{norm}_l - r_{i,l}^2$
18    end
19 end
```

The quasi-ML accuracy of floating-point SQRD-based FSD is demonstrated in [26], however since FPE-based realization requires reduced-precision fixed-point arithmetic, similar verification under these conditions is required. To this end, the BER performance of SQRD-based FSD detection of a 4×4 16-QAM Rayleigh Fading MIMO channel has been performed for 32, 24 and 16 bit fixed-point variants and is compared with the ideal floating-point version in Fig. 4[2]. As Fig. 4 shows, 16 bit arithmetic in the SQRD phase is sufficient to enable detection performance almost equal to that of the ideal floating-point solution, particularly when integer wordsizes of 9 or 10 bits are employed. Hence, FPE-based realization is viable and 16 bit fixed-point arithmetic with a 10 bit fractional part is chosen for FPE-based SQRD.

Despite its relatively low complexity and suitability for fixed-point implementation, there are two major issues that must be resolved to enable FPE-based SQRD PP for 4×4 802.11n:

1) SQRD remains highly computationally demanding, as outlined in Table IV; given the capabilities of a single FPE, it appears that a large-scale multi-FPE architecture is required to enable SQRD for 4×4 802.11n.
2) The square root (line 12) and division (line 13) operations used in SQRD offer very low performance on sequential

[2]For clarity, the behaviors under 32 and 24 bit fixed point conditions are omitted from Fig. 4 due to the high detection performance of 16 bit solutions.

TABLE IV
4 × 4 SQRD OPERATIONAL COMPLEXITY

Operation	**No. per second** $(\times 10^9)$
$+/-$	3.24
$\times$	12.72
$\div$	0.12
$\surd$	0.12

processors [27]; special consideration of these is required for real-time PP for 802.11n MIMO.

A. FPE-Based Division Acceleration

Binary division is usually achieved using digital recurrence or convergence algorithms [27]. Of these alternatives, recurrence algorithms generally exhibit lower complexity and latency, and hence are usually preferred. Non-restoring recurrence algorithms generally enable higher performance FPGA implementations by avoiding sophisticated control overhead [28].

Non-restoring 16 bit division [27] requires 312 cycles on a *16R* FPE. This equates to approximately 1.2 MDIV/s (millions of divisions per second), which means that to achieve the 120 MDIV/s required by real-time 4 × 4 SQRD for 802.11n would require at least 100 FPEs dedicated solely to division. The high resource cost such a solution could entail may potentially be avoided by exploiting the configurable nature of the FPE, specifically its ability to incorporate coprocessors within the ALU, to accelerate FPE-based division.

Radix-2/4 non-restoring division coprocessors [27] are considered in this context—the structure of these coprocessors are outlined in Fig. 5. The performance, cost and efficiency (in terms of throughput per LUT, or TP/LUT) of the programmed FPE implementation $(FPE - P)$ and radix-2/4 coprocessor augmented FPEs $(FPE - R_2, FPE - R_4)$ when implemented on Virtex 5 FPGA is described in Table V. As this shows, $FPE - R_2$ and $FPE - R_4$ increase throughput by factors of 8.9 and 13.3 and hardware efficiency by factors of 9.4 and 10.7 as compared to $FPE - P$ respectively. Given the need for 120 MDIV/s for SQRD-based detection of 4 × 4 802.11n MIMO systems, the implied implementation cost and performance metrics of each option are summarized in Table V. This suggests that $FPE - R_2$ represents the lowest cost real-time solution, enabling a 93.4% reduction in resource cost relative to $FPE - P$. Accordingly, this approach is adopted in the FPE-based SQRD implementation.

B. FPE-Based Acceleration of Square Root Operations

Implementing square root operations poses a similar problem to that of division—120 MSQRT/s (million square root operations per second) are required for real-time SQRD-based detection of a 4 × 4 802.11n system. There are two primary options for achieving this: software-based execution on the native FPE, using the pencil-and-paper method [27], or by using a standard CORDIC component available in vendor IP libraries [29]. The programmed solution $(FPE - P)$ is compared with that incorporating the CORDIC coprocessor $(FPE - C)$ in

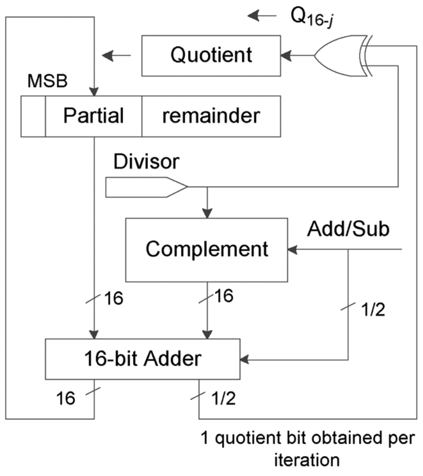


Fig. 5. SQRD divider coprocessor architecture.

TABLE V
SQRD DIVISION IMPLEMENTATIONS

Solution	**Resource**			**Throughput**
	FPEs	**DSP48Es**	**LUTs**	**(MDiv/s)**
FPE-P	100	100	13,600	120
FPE-R_2	5	5	900	120
FPE-R_4	4	4	944	144

TABLE VI
COMPARISON OF 16 BIT PSQRT, CSQRT ON FPE

		FPE-P [27]	***FPE-C*** [29]
Cost	**PM/RF locations**	29/14	8/1
	LUTs	142	330
	DSP48Es	1	0
	Clock (MHz)	367.7	350
	Latency (Cycles)	191	8
	Throughput (MSQRT/s)	1.93	43.6
	TP/LUT $(\times 10^{-}3)$	13.6	132.1

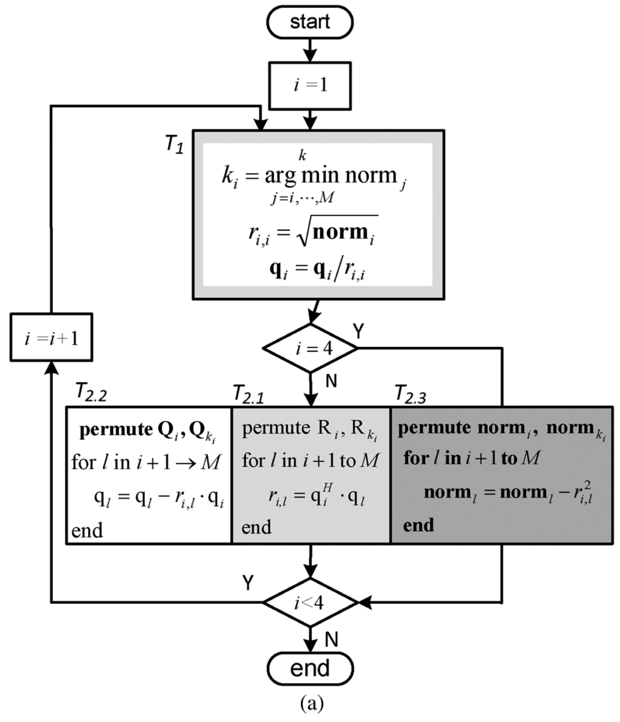
TABLE VII
FPE-BASED SQRT IMPLEMENTATIONS

Solution	**Resource**			**Throughput**
	FPEs	**DSP48Es**	**LUTs**	**(MSqrt/s)**
FPE-P	63	63	8946	121.6
FPE-C	3	3	990	130.8

Table VI. As this shows, $FPE - C$ offers simultaneous increases in throughput and efficiency by factors of 23 and 10 respectively as compared to $FPE - P$. This implies that the resources required to realise real-time square-root for SQRD-based detection of 4 × 4 802.11n are summarized in Table VII. Hence $FPE - C$ enables real-time operation whilst incurring only 11% of the resource required by $FPE - P$, and is adopted for realising FPE-based square root operations.

C. FPE-Based SQRD

Whilst Sections IV-A and IV-B have provided FPE-based components of sufficiently high performance and low cost to


start
$i=1$
T_1
$k_i = \arg\min_{j=i,\cdots,M}^{k} \text{norm}_j$
$r_{i,i} = \sqrt{\mathbf{norm}_i}$
$\mathbf{q}_i = \mathbf{q}_i / r_{i,i}$
$i=i+1$
$i=4$
Y
N
$T_{2.2}$
$T_{2.1}$
$T_{2.3}$
permute $\mathbf{Q}_i, \mathbf{Q}_{k_i}$
for l in $i+1 \rightarrow M$
$q_l = q_l - r_{i,l} \cdot q_i$
end
permute R_i, R_{k_i}
for l in $i+1$ to M
$r_{i,l} = q_i^H \cdot q_l$
end
permute $\mathbf{norm}_i, \mathbf{norm}_{k_i}$
for l in $i+1$ to M
$\mathbf{norm}_l = \mathbf{norm}_l - r_{i,l}^2$
end
Y
$i<4$
N
end
(a)

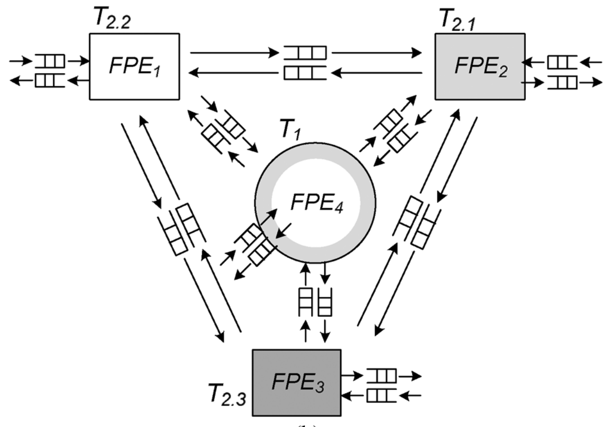


Fig. 6. 4 × 4 SQRD FPE-MIMD mapping. (a) 4 × 4 SQRD; (b) 4 × 4 SQRD architecture.

enable real-time division and square-root, integrating these components into a coherent processing architecture to perform SQRD, and replicating that behaviour to provide PP for the 108 subcarriers of 802.11n MIMO is a large scale, challenging implementation problem.

Fig. 6(a) describes the SQRD algorithm as a flow chart composed of four main iterative tasks $(T_1, T_{2.1} - T_{2.3})$. The first task, T_1, conducts the iterative channel norm ordering, and computes the diagonal elements of $\mathbf{R}$ (lines 11–13 in Algorithm 1), with the subsequent concurrent tasks $T_{2.1} - T_{2.3}$ permuting and updating $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{norm}$ respectively (lines 14–18 in Algorithm 1).

To realise this behaviour a 4-FPE Multiple Instruction, Multiple Data (MIMD) processing architecture, illustrated in Fig. 6(b), is used; the FPEs employ 16 bit datapaths, in accordance with the analysis in Section IV, and are otherwise configured as described in Table VIII(a). $FPE_1 - FPE_3$ perform permutation of $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{norm}$ and iterative updating ($T_{2.1} - T_{2.3}$ in Fig. 6(a)), whilst FPE_4 calculates the diagonal elements of $\mathbf{R}$ (T_1). Across the architecture, SQRD-based PP of a 4 × 4 matrix occurs in three phases. Initially, $\mathbf{H}$ and the calculation of $\mathbf{norm}$ are distributed amongst the FPEs, with the separate parts of $\mathbf{norm}$ gathered by FPE_4 to undergo ordering, division and square root. These resulting metrics are distributed to the outer FPEs for independent iterative permutation and update of $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{norm}$. Inter-FPE communication occurs via point-to-point FIFO links, chosen due to their relatively low cost on FPGA and implicit ability to synchronize the multi-FPE architecture in a data-driven manner whilst avoiding data access conflicts. The performance and cost of the 4-FPE grouping is given in Table VIII(b).

TABLE VIII
4-FPE Based SQRD. (a) FPE Configuration; (b) FPE-SQRD Metrics

(a)

Parameter	Value
PM Depth	350
RFDepth	32
IMMDepth	32
DMDepth	64
TxComm	32
RxComm	32

(b)

	Aspect	Value
Cost	**LUTs**	2109
	DSP48Es	4
	BRAMs	0
	Clock (MHz)	3.15
	T (MSQRD/s)	1.07
	Latency (μS)	0.9

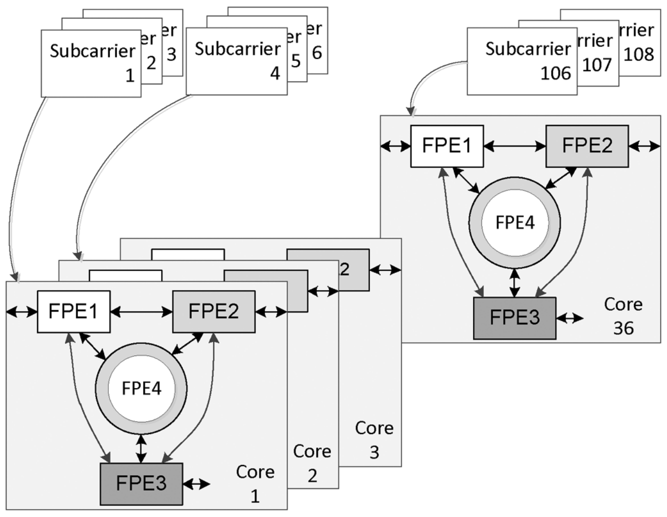


Fig. 7. 4 × 4 SQRD mapping.

According to the metrics quoted in Table VIII(b), the throughput of each 4-FPE group is sufficient to support SQRD-based PP of 3 subcarriers within the real-time constraints of 802.11n. Hence, to implement PP for all 108 subcarriers of 802.11n, the architecture illustrated in Fig. 7, incorporating 36 groups of the 4-FPE array, is used. The mapping of subcarriers to groups is as described in Fig. 7.

D. Implementation Evaluation

When implemented on Xilinx Virtex 5 VSX240T FPGA, the cost and performance of the 802.11n PP architecture $(FPE-SQRD)$ described in Fig. 7 are as quoted in Table IX. These results are notable since, to the best of the authors' knowledge, they constitute the only recorded real-time SQRD PP and FSD

TABLE IX
4 × 4 SQRD IMPLEMENTATIONS

	Ref	FPE-SQRD	[30]
Resource	LUTs	**70,560**	33,512
	DSP48Es	**144**	426
	BRAMs	**0**	N/A
	ELUTs ($\times 10^3$)	**152.5**	276.0
	Clock (MHz)	**265**	87
	T (MSQRD/s)	**32.5**	22
	L (μS)	**1.1**	1.43

TABLE X
802.11N FSD OPERATIONAL COMPLEXITY

Operation	No. per second ($\times 10^9$)
$+/-$	32.37
$\times$	19.20

TABLE XI
FPE-BASED MCS IMPLEMENTATIONS

		16R-MCS	16R + Coprocessors
Cost	PM/RF Locations	4591/32	1420/32
	LUTs	2520	805
	DSP48Es	1	0
	Clock (MHz)	367.7	350
	Latency (Cycles)	3281	1420
	Throughput (MOP/s)	1.9	4.5

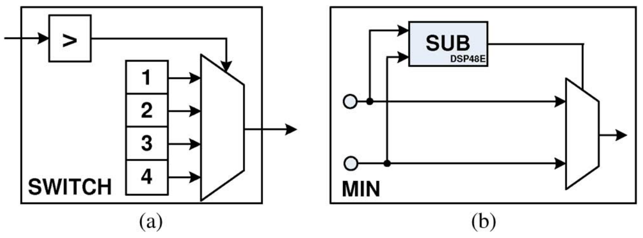


Fig. 8. FPE Coprocessors for switch and min acceleration. (a) Switch coprocessor; (b) Min coprocessor.

PP architectures for 4 × 4 MIMO. They achieve 32.5 MSQRD/s (millions of SQRD operations per second) exceeding the required 30 MSQRD/s.

Given that they are unique in enabling real-time PP for SD-based detection for 4 × 4 MIMO, comparing with other PP realisations is difficult—whilst a number of SD implementations rely on SQRD-based PP, such as [31], [32], they do not implement it. Further, whilst the work in [33] describes an SQRD implementation, balanced objective comparison is difficult since it reports only the resource cost for an ASIC-based 2 × 2 SQRD, giving no measure of real-time performance. Table IX compares $FPE-SQRD$ with a custom circuit-based V-BLAST detector [30]. The implementation in [30] does not achieve the required throughput for real-time processing and whilst $FPE - SQRD$ consumes significantly more LUT resource, it significantly reduces the demand for DSP48E resources by 66%; combining these relates to an overall reduction of 45% in terms of Equivalent LUTs (ELUTs)[3], whilst enabling an increased throughput of almost 50%.

Hence, this analysis shows that the FPE has enabled the only software-defined real-time PP architecture for 4 × 4 16-QAM 802.11n FSD, whilst incurring resource costs comparable to existing circuit architectures. Indeed, further, the FPE-SQRD it is the only real-time realization of any kind. Section V investigates its ability to support the second major suboperation of FSD: MCS.

V. FPGA-BASED FSD MCS FOR 802.11N

The MCS stage of FSD for 4 × 4 16 QAM 802.11n is even more computationally demanding than SQRD-based PP, as described in Table X. When a single 4 × 4 16-QAM FSD MCS is implemented on a *16R* FPE, the performance and cost are as reported as *16R-MCS* in Table XI.

Table XI reports a large increase in resource cost for *16R-MCS* as compared to the basic *16R* reported in Table III. The observed order of magnitude increase is a consequence of the large PM required to house the 4591 instructions required. A significant factor in this large number of instructions are the comparison operations required for slicing (4) and sorting the PED metrics, which require branch instructions. Associated with these branch instructions are NOPs, whose number is swollen by the FPE's deep pipeline [12]; the wasted cycles these NOPs represent dramatically increase cost and reduce throughput—indeed branch and NOP instructions represent 50.7% of the total number of instructions. As a result, optimizing the FPE architecture to reduce the impact of these branch instructions could have a significant impact on the MCS cost/performance.

Employing ALU coprocessors, in a manner similar to that described in Section IV to accelerate division and square root operations, can significantly reduce these penalties. A SWITCH coprocessor (Fig. 8(a)), which compares the input to one of a number of pre-defined options can be used to accelerate slicing, whilst a MIN coprocessor (Fig. 8(b)) can accelerate the sorting operation.

Each of these coprocessors costs 20 LUTs, but their ability to eliminate wasted instructions can significantly reduce the PM size leading to an overall cost decrease and performance increase when these coprocessors are used, as described in column 3 of Table XI. As this shows, including these components results in a 68% reduction in resource cost and a factor 2.3 increase in throughput. This produces an implementation capable of realising FSD MCS for a single 802.11n subcarrier in real-time, providing a good foundation unit for implementing MCS for all 108 subcarriers.

A. SIMD-Based Implementation of 802.11n FSD MCS

A large, coherent collection of FPEs is required to implement FSD MCS for all 108 subcarriers of 802.11n MIMO. Two important observations of the application's behaviour help guide the choice of multiprocessing architecture:

1) In the tree-structured FSD MCS (Fig. 2(b)), each tree branch performs an identical sequence of operations on

[3]ELUTs combine measurement of resource on modern Xilinx FPGA in a single quantity; the reader is referred to [34] for details.

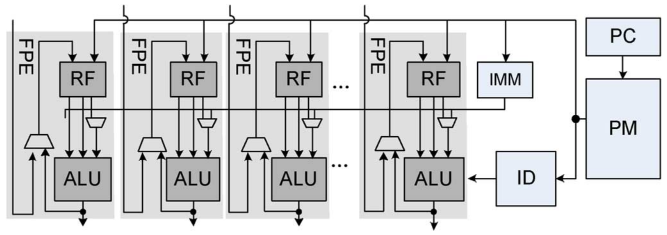


Fig. 9. SIMD processor architecture.

TABLE XII
SIMD PROCESSOR CONFIGURATION PARAMETERS

Parameter	Meaning	Values
SIMDways	No. parallel FPE elements	Unlimited
IMMDepth/IMMWidth	No./width of IMM locations	Unlimited
PMDepth/PMWidth	No./width of PM locations	Unlimited

distinct data streams—the definition of Single Instruction Multiple Data (SIMD) behavior.

2) The number of FPEs required to implement MCS for all 108 OFDM subcarriers on a single, very wide SIMD processor implies limitations on the achievable clock rate as a result of high signal fan-outs to broadcast instructions from a central PM to a very large number of ALUs, restricting performance [11]. Hence, a collection of smaller SIMDs is used.

To enable these multi-SIMD architectures, the FPE is used as a foundation for a configurable SIMD processor component, as illustrated in Fig. 9. Note that the PC, PM, ID and IMM are now all centralized in the SIMD, and hence do not appear in each FPE way. Table XII defines the configurable aspects of the SIMD processor. All of the FPE instructions (except BEQ, BGT and BLT) can be used as SIMD instructions.

The increasing limiting effect of instruction broadcast from the central PM results in 16-way SIMD configurations offering the best cost/performance balance; accordingly FSD MCS for all 108 802.11n subcarriers is implemented on a dual-layer network of such processors, as illustrated in Fig. 10. Level 1 consists of 8 SIMDs. The 802.11n subcarriers are clustered into 8 groups $\{G_i = \{j : (j-1) \bmod 8 = i\}_{j=1}^{108}\}_{i=0}^{7}$, where j is the set of subcarriers processed by Core i. The 16 branches of the MCS tree for each subcarrier are processed in parallel across the 16 ways of the Level 1 SIMD onto which they have been mapped. Sorting for the subcarriers implemented in each Level 1 SIMD is performed by adjacent pairs of ways in the Level 2 SIMD—hence given the 8 Level 1 SIMDs, the Level 2 SIMD is composed of 16 ways.

The analysis in [22] shows that 16 bit data is sufficient for FSD-based detection of 4 × 4 16-QAM 802.11n, hence each FPE is configured to exploit 16 bit real-valued arithmetic. All processors exploit $PMDepth = 128$, $RFDepth = 32$ and $DMDepth = 0$, and communication between the two levels exploit 8-element FIFO queues. The Level 1 SIMDs incorporate SWITCH coprocessors to accelerate the slicing operation, whilst the Level 2 SIMDs support the MIN ALU extension to accelerate the sort operation.

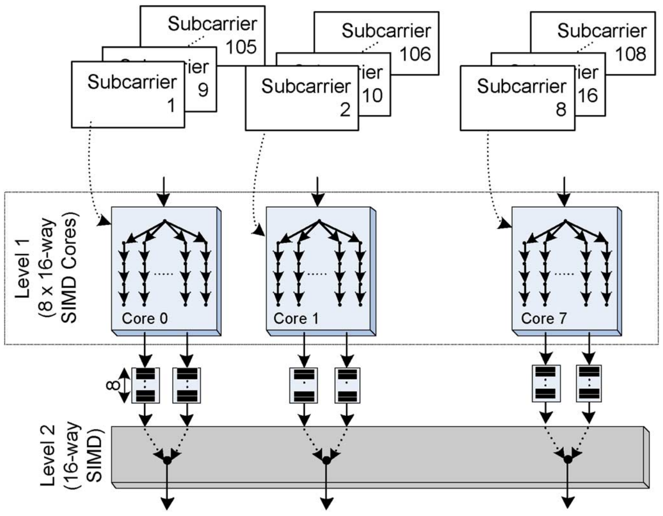


Fig. 10. 802.11n OFDM MCS-SIMD mapping.

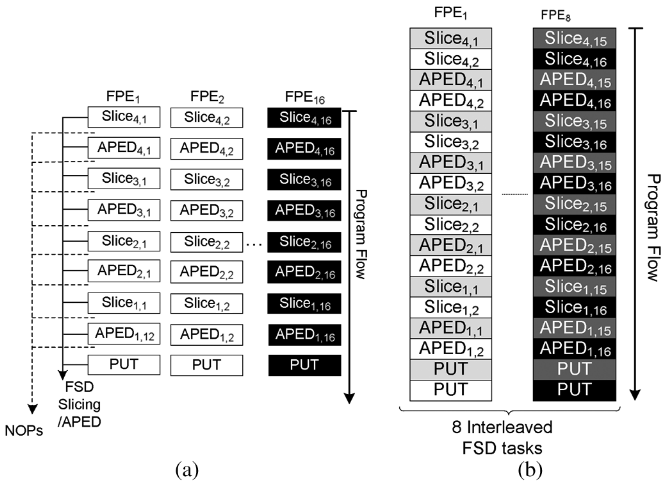


Fig. 11. FPE branch interleaving. (a) Original FSD threads; (b) Interleaved threads.

The program flow for each Level 1 SIMD is as illustrated in Fig. 11(a). As this shows, each FPE performs a single branch of the MCS tree, with the empty parts of the program flow representing NOP instructions, used to properly synchronize movement of data into and out of memory. These NOP cycles represent 29% of the total instruction count but since they represent ALU idle cycles they should preferably be eliminated. To achieve this, the NOP cycles in one branch can be occupied by the useful, independent instructions from another, i.e., the branches may be interleaved as illustrated in Fig. 11(b). As this shows, interleaving branches occupies wasted NOP cycles, to the extent that when two branches are interleaved the proportion of wasted cycles is reduced to 4%.

When implemented on Xilinx Virtex 5 VSX240T FPGA the performance and cost of the FSD-MCS for 802.11n are reported as $FPE-MCS$ in Table XIII. As this shows, it comfortably exceeds the real-time performance criteria of 802.11n, and is the first software-defined implementation to do so.

TABLE XIII
4×4 16-QAM FSD IMPLEMENTATIONS

Ref	FPE-MCS	[22]	[17]	[18]
LUT	**16,601**	13,197	18,893	6,587
DSP48E	**144**	160	64	0
BRAM	**0**	49	12	0
ELUT ($\times 10^3$)	**98.5**	168.0	N/A	N/A
Clock (MHz)	**296**	150	100	52
T (Mbps)	**502.5**	600	200	27.7
L (μS)	**0.9**	N/A	N/A	N/A

TABLE XIV
4×4 SQRD FSD FULL DETECTOR IMPLEMENTATIONS

	Aspect	FPE-FSD
Resource	**LUTs**	96,115
	DSP48Es	408
	BRAMs	N/A
	ELUTs ($\times 10^3$)	328
	Clock (MHz)	189
	T	483
	L (μS)	2.3

Table XIII also compares the FPE-MCS with existing Xilinx FPGA custom circuit SD realisations. The work in [22] displays a slightly lower LUT cost and higher throughput, but the $FPE-MCS$ enables software-programmability whilst maintaining real-time behaviour and comparable cost. The architectures in [17], [18] operate well below real-time, and the architectural changes necessary to enable real-time performance are such that direct comparison is very difficult. The work in [8] presents a very high performance 800 Mbps single subcarrier architecture on Altera FPGA, but in common with [6] the resource and performance implications of adapting this to enable all 108 802.11n subcarriers is unknown, making comparison difficult. Finally, target technology variations between the FPE-MCS and the 2×2 ASIC-based custom circuit in [35] make comparison very difficult also.

Given these comparisons, the novel aspect of the FPE-MCS is clear: it is the only software-defined approach which supports real-time FSD MCS for 4×4, 16 QAM 802.11n. Similarly to the FPE-SQRD presented in Section IV, it shows that massively parallel networks of simple processors (>140 in this case) on FPGA can support real-time processing with resource costs comparable to custom circuits. Like all software-defined radio platforms, it trades absolute performance/cost for flexibility and ease of design: it offers a predominately software-based design approach, and hence is inherently more flexible for adaption to other SD or even other more general DSP algorithms, as well as being more suited to existing software radio design processes. In Section VI, the FPE-based design approach is applied to the design of a full FSD detector.

VI. FPGA-BASED SOFTWARE-DEFINED FSD FOR 802.11N

When the PP and MCS implementation strategies, described in Sections IV and V respectively, are combined to create a full FSD detector implementation, the cost and performance of the implementation are as reported in Table XIV. Given that this implementation realizes the real-time processing requirements of 4×4 16 QAM 802.11n, to the best of the authors' knowledge it is the only single-chip implementation to do so, despite its software-defined nature.

VII. CONCLUSION

This paper has presented a uniquely capable approach for implementing SD detectors for MIMO receivers: it is the first software-defined platform to support real-time detection for applications such as 4×4 16 QAM 802.11n MIMO. This paper has shown how, by composing fine-grained, very high performance programmable components into large scale multiprocessing architectures on FPGA, the resulting software-defined architectures satisfy the demanding real-time performance metrics of modern MIMO standards, whilst incurring resource costs of the order of existing dedicated circuit architectures. We have demonstrated this by creating three unique implementations:

1) The only recorded SQRD-based PP architecture for 4×4 802.11n MIMO.
2) The only recorded real-time software-defined FSD MCS architecture for 4×4 16-QAM 802.11n MIMO.
3) The only recorded single-chip integrated quasi-optimal detector for 4×4 16-QAM 802.11n MIMO.

It is important to note that their software-defined nature implicitly eases the design process for architectures such as these. However, this is only the case given supporting Computer Aided Design (CAD) and software compilation infrastructure. This paper has concentrated on demonstrating the feasibility of the architectures to support such realisations, but has constructed and programmed them manually at the Register Transfer Level (RTL) and assembly level respectively. Creating these technologies is left as future work of similar significance to the demonstration of architectural viability presented here.

ACKNOWLEDGMENT

The authors would like to thank Prof. Roger Woods, Prof. John Thompson, Dr Chengwei Zheng and Mr. Matthew Milford for their valuable assistance in this work.

REFERENCES

[1] P. Wolniansky, G. Foschini, G. Golden, and R. Valenzuela, "V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel," in *Proc. URSI Int. Symp. Signals, Syst., Electron.*, 1998, pp. 295–300.

[2] *802.11n-2009 IEEE Local and Metropolitan Area Networks-Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, IEEE802.11n, 2009.

[3] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI Implementation of MIMO detection using the sphere decoding algorithm," *IEEE J. Solid-State Circuits*, vol. 40, no. 7, pp. 1566–1577, Jul. 2005.

[4] X. Huang, C. Liang, and J. Ma, "System architecture and implementation of MIMO sphere decoders on FPGA," *IEEE Trans. VLSI Syst.*, vol. 16, no. 2, pp. 188–197, 2008.

[5] M. Li, B. Bougard, W. Xu, D. Novo, L. Van Der Perre, and F. Catthoor, "Optimizing Near-ML MIMO detector for SDR baseband on parallel programmable architectures," *Proc. Des., Autom., Test in Eur.(DATE)*, pp. 444–449, Mar. 2008.

[6] P. Bhagawat, R. Dash, and G. Choi, "Dynamically reconfigurable soft output MIMO detector," in *Proc. IEEE Intl. Conf. Comput. Des. (ICCD)*, Oct. 2008, pp. 68–73.

[7] J. Janhunen, O. Silvén, and M. Juntti, "Programmable processor implementations of K-best list sphere detector for MIMO receiver," *Elsevier J. Signal Process.*, vol. 90, no. 1, pp. 313–323, 2009.
[8] M. Khairy, M. Abdallah, and S.-D. Habib, "Efficient FPGA implementation of MIMO decoder for mobile WiMAX system," in *IEEE Intl. Conf. Commun. (ICC)*, Jun. 2009, pp. 1–5.
[9] J. Bard and V. J. Kovarik, Jr., *Software Defined Radio: The Software Communications Architecture*. New York: Wiley, 2007.
[10] J. H. Reed, *Software Radio: A Modern Approach to Radio Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 2002.
[11] X. Chu and J. McAllister, "FPGA based soft-core SIMD processing: A MIMO-OFDM fixed-complexity sphere decoder case study," in *Proc. IEEE Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2010, pp. 479–484.
[12] X. Chu, J. McAllister, and R. Woods, "A pipeline interleaved heterogeneous SIMD soft processor array architecture for MIMO-OFDM detection," in *Proc. 7th Int. Conf. Reconfigurable Comput.: Architect., Tools, Appl. (ARC)*, Mar. 2011, pp. 133–144.
[13] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications," *SIGSAM Bull.*, vol. 15, no. 1, pp. 37–44, 1981.
[14] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Math. Programm.*, vol. 66, no. 1, pp. 181–199, 1994.
[15] J. Antikainen, P. Salmela, O. Silven, M. Juntti, J. Takala, and M. Myllyla, "Application-specific instruction set processor implementation of list sphere detector," in *Conf. Rec. 41st Asilomar Conf. Signals, Syst., Comput.*, Nov. 2007, pp. 943–947.
[16] J. Janhunen, O. Silven, M. Juntti, and M. Myllyla, "Software defined radio implementation of K-best list sphere detector algorithm," in *Proc. Int. Conf. Embedded Comput. Syst.: Architect., Model., Simul. (SAMOS)*, Jul. 2008, pp. 100–107.
[17] Q. Qi and C. Chakrabarti, "Parallel high throughput soft-output sphere decoder," in *Proc. IEEE Workshop Signal Process. Syst. (SIPS)*, Oct. 2010, pp. 174–179.
[18] B. Wu and G. Masera, "A Novel VLSI architecture of fixed-complexity sphere decoder," in *Proc. 13th Euromicro Conf. Digit. Syst. Des.: Architect., , Methods, Tools*, Sep. 2010, pp. 737–744.
[19] L. Barbero and J. Thompson, "Fixing the complexity of the sphere decoder for MIMO detection," *IEEE Trans. Wireless Commun.*, pp. 2131–2142, Jun. 2008.
[20] L. Hanzo, W. Webb, and T. Keller, *Single and Multi-Carrier Quadrature Amplitude Modulation: Principles and Applications for Personal Communications, WATM and Broadcasting*. New York: IEEE Presss-Wiley, 2000.
[21] C. Zheng, X. Chu, J. McAllister, and R. Woods, "Real-valued fixed-complexity sphere decoder for high dimensional QAM-MIMO systems," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4493–4499, Sep. 2011.
[22] L. G. Barbero and J. S. Thompson, "Rapid prototyping of a fixed-throughput sphere decoder for MIMO systems," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2006, pp. 3082–3087.
[23] P. Yiannacouras, J. G. Steffan, and J. Rose, "Fine-grain performance scaling of soft vector processors," in *Proc. Int. Conf. Compilers, Architec., Synthesis Embedded Syst. (CASES)*, Oct. 2009, pp. 97–106.
[24] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core CPU accelerator," in *Proc. Int. ACM/SIGDA Symp. Field Programmable Gate Arrays (FPGA)*, Feb. 2008, pp. 222–232.
[25] D. Wubben, R. Bohnke, V. Kuhn, and K.-D. Kammeyer, "MMSE extension of V-BLAST based on sorted QR decomposition," in *Proc. IEEE Veh. Technol. Conf. (VTC)*, Oct. 2003, vol. 1, pp. 508–512.
[26] X. Chu, J. McAllister, and R. Woods, "A low complexity real-time MIMO-preprocessing for fixed-complexity sphere decoder," in *Proc. Wireless Innovation Forum (SDR'11-WINNComm)*, Nov. 2011, pp. 601–604.
[27] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed. New York: OUP USA, 2010.
[28] N. Sorokin, "Implementation of high-speed fixed-point dividers on FPGA," *J. Comput. Sci. Technol.*, vol. 6, no. 1, pp. 8–11, 2006.
[29] Xilinx Inc., LogiCORE IP CORDIC v4.0 2011.
[30] X. Chu, K. Benkrid, and J. Thompson, "Rapid prototyping of an improved cholesky decomposition based MIMO detector on FPGAs," in *Proc. NASA/ESA Conf. Adapt. Hardware Syst. (AHS)*, 2009, pp. 369–375.
[31] N. Moezzi-Madani, T. Thorolfsson, and W. Davis, "A low-area flexible MIMO detector for WiFi/WiMAX standards," in *Proc. Des., Autom. Test Eur. (DATE)*, Mar. 2010, pp. 1633–1636.
[32] M. Myllyla, J. Cavallaro, and M. Juntti, "Architecture design and implementation of the metric first list sphere detector algorithm," *IEEE Trans. VLSI Syst.*, vol. 19, no. 5, pp. 895–899, May 2011.
[33] J. Im, M. Cho, Y. Jung, and J. Kim, "Low-power low-complexity MIMO-OFDM baseband processor for wireless LANs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2009, pp. 601–604.
[34] D. Sheldon, R. Kumar, R. Lysecky, F. Vahid, and D. Tullsen, "Application-specific customization of parameterized FPGA soft-core processors," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2006, pp. 261–268.
[35] T. Cupaiuolo, M. Siti, and A. Tomasoni, "Low-complexity high throughput VLSI architecture of soft-output ML MIMO detector," in *Proc. Des., Autom. Test Eur. e (DATE)*, Mar. 2010, pp. 1396–1401.

Xuezheng Chu (M'11) received the MRes. degree in electronics engineering from The University of Edinburgh, Edinburgh, U.K. in 2008 and the Ph.D. degree in electronics engineering from Queen's University Belfast, Belfast, U.K. in 2011, for his work on Efficient and Adaptive FPGA-based MIMO Detectors and Massively Parallel Processor Architecture for FPGA-based real-time signal processing systems.

He is currently with ARM, Ltd., Cambridge, U.K. His research interests include SoC, many-core system and on-chip Interconnect designs, wireless signal processing systems, and their implementation.

John McAllister (S'02–M'04) received the Ph.D. degree in electronic engineering from Queen's University Belfast, Belfast, U.K., in 2004.

Since July 2005, he has lead a group of researchers in embedded processing architectures, and electronic system level synthesis tools and technologies for streaming applications, with a specialty particularly high performance software radio transceiver architectures and design tools for MIMO communications applications. In 2008, he co-founded CapnaDSP, Ltd., to commercialize the unique FPGA synthesis tool technology he invented; this currently exists as the Intrinsic Toolsuite. He was Chief Technology Office of CapnaDSP from its foundation until 2011.

Dr. McAllister is a member of the IEEE Signal Processing Society and its Technical Committee on Design and Implementation of Signal Processing Systems (DISPS). He is an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and serves on the program committees of a number of IEEE conferences, including the International Conference on Embedded Computer Systems: Architectures, Modelling and Simulation (SAMOS), the International Conference on Acoustics, Speech and Signal Processing (ICASSP), and the Workshop on Signal Processing Systems: Design and Implementation (SIPS).