COOPERATIVE ACTOR ORIENTED SYNTHESIS FOR FPGA-BASED MIMO-OFDM DETECTION

Yun Wu, John McAllister, Peng Wang

eStreams,
Institute of Electronics, Communications and Information Technology (ECIT),
Queens University Belfast, UK

ABSTRACT

Massively parallel networks of fine-grained software programmable processors on FPGA have enabled the only software-defined architectures supporting real-time implementation of Sphere Decoding (SD) detection of Multiple-Input Multiple-Output (MIMO) channels. However, research to date has only proven the viability of the architectures to support real-time software-defined SD; the adoption of this approach is highly dependent on the availability of synthesis methodologies and tool-sets to support the designer in programming for these the target multi-core architectures. No such approach which results in the derivation of real-time solutions exists. In this paper a designer directed synthesis approach is proposed which creates and maps SD applications specified as Dataflow Graphs (DFGs) onto such architectures. When applied to synthesis of Fixed-complexity Sphere Decoder (FSD) and Selective Spanning Fast Enumeration (SSFE) detectors, multi-SIMD architectures on FPGA are derived for 4×4 MIMO $16/64$QAM scheme, which are the only automatically derived software-defined implementations on record which meets the 802.11n throughput requirements of 480 Mbps.

Index Terms— Soft-Core, FPGA, ESL, MIMO, OFDM, Dataflow, Mapping and Scheduling

1. INTRODUCTION

Sphere Decoding is a highly attractive detection scheme for Multiple-Input Multiple-Output (MIMO) communications systems, offering quasi-ML (Maximum Likelihood) performance with considerably reduced computational complexity as compared to the ideal ML detector [1]. However, for Software-Defined Radio (SDR) systems, supporting real-time quasi-ML SD for standards such as 802.11n or LTE is a substantial implementation challenge.

This work is supported by Xilinx Inc., National Instruments Inc. and CapnaDSP Ltd. under UK Engineering and Physical Sciences Research Council contract number EP/H051155/1

Indeed, the recent work in [2] is the first demonstrated feasible real-time SDR solution. By exploiting multi-SIMD (Single Instruction, Multiple Data) architectures on Field-Programmable Gate Array (FPGA) it has enabled the only recorded software-defined quasi-ML MIMO detector for 4×4 16-QAM 802.11n MIMO. However, to do so necessitated manual architecture design in VHDL and assembly-level programming, negating the flexibility and rapid deployment benefits of software radio platforms [3]. Hence, whilst the architectural foundations are in place to enable such solutions, platform synthesis methodologies and tool-sets are still absent.

This paper resolves this problem. By extending work in [4], it presents a designer-guided semi-automated *cooperative synthesis* process which enables almost automatic derivation of real-time multi-SIMD SD architectures on FPGA. Specifically, it makes three novel contributions.

1) A novel cooperative synthesis approach for SDR is proposed.

2) A toolset to automate 1) is described.

3) The tool-set from 2) is applied to Fixed Complexity SD (FSD) and Selective Spanning Fast Enumeration (SSFE) detectionof ynthesis of 4×4 16/64QAM 802.11n, enabling the first recorded automatically generated real-time implementations.

The rest of this paper is organized as follows. Section 2 motivates the choice of synthesis approach, before the synthesis process and toolset described in Section 3 are applied to FSD and SSFE detectors in Section 4.

2. MOTIVATION

MIMO communications standards, such as 802.11n and LTE, employ multiple antennas at both the transmit and receive ends of the communications channel to enable high spectral efficiency in multipath communications environments [5]. For $M \times N$ MIMO communication (i.e. M transmit and N receive antennas) over a multipath fading channel,

$\mathbf{H} \in \mathbb{C}^{\mathbf{N}\times\mathbf{M}}$, with Additive White Gaussian Noise (AWGN), $\mathbf{v} \in \mathbb{C}^{\mathbf{N}\times\mathbf{1}}$, a set of transmitted symbols, $\mathbf{s} \in \mathbb{C}^{\mathbf{M}\times\mathbf{1}}$, are transformed to a received symbol vector, $\mathbf{r} \in \mathbb{C}^{\mathbf{N}\times\mathbf{1}}$, according to

$$\mathbf{r} = \mathbf{H} \cdot \mathbf{s} + \mathbf{v}. \tag{1}$$

Detection algorithms such as SD attempt to retrieve an estimate of $\widehat{\mathbf{s}}$ from $\mathbf{r}$. In particular, SD approaches are highly attractive due to their ability to enable quasi-ML detection with significantly reduced computational complexity as compared to the ideal ML detector. Whilst many SD schemes have been proposed, FSD [6] offers a unique combination of low complexity, quasi-ML performance and deterministic behavior that makes it ideal for embedded implementation. Fig. 1 illustrates the structure of the FSD algorithm.

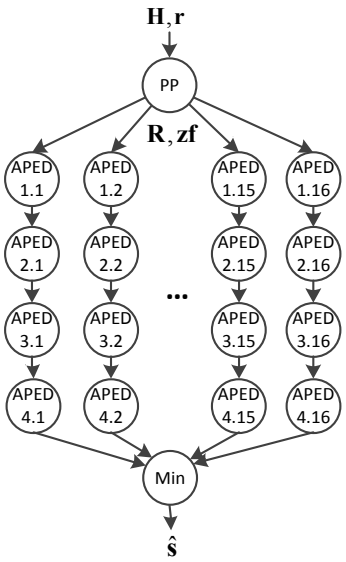


Fig. 1. The FSD Algorithm Example of $16QAM$ and $M = 4$

As this shows, during Preprocessing (PP) the received symbol vector $\mathbf{r}$ undergoes Zero-Forcing (ZF) equalization according to:

$$\mathbf{zf} = (\mathbf{H}^H \cdot \mathbf{H})^{-\mathbf{1}} \cdot \mathbf{H}^{\mathbf{H}} \cdot \mathbf{r} \tag{2}$$

whilst the channel matrix $\mathbf{H}$ is ordered based on the distortion experienced by each path through the multipath environment, via process such as Sorted QR decomposition (SQRD) [7], to produce an upper triangular matrix $\mathbf{R}$. The $\mathbf{zf}$ symbol vector then undergoes processing by an N-level detection tree. During the first $NFS = \left\lceil \sqrt{M} - 1 \right\rceil$ levels, each input symbol is enumerated to M_c successor symbols, where M_c is the number of constellation points in the transmission modulation scheme, whilst during the remaining $M - NFS$ levels, only a single successor is maintained. At each level, the Accumulated Partial Euclidean Distance (APED) is calculated according to (3) and (4).

$$PED_{n,i} = \sum_{m=n}^{M} r_{n,m}^2 \left\| \widehat{zf}_{m,i} - \widehat{s}_{m,i} \right\|^2 \tag{3}$$

$$\begin{aligned} APED_{n,i} &= APED_{n,i} + PED_{n-1,i}, \\ APED_{M,i} &= PED_{M,i} \end{aligned} \tag{4}$$

where i is the index of search branch, $r_{n,m}$ is the element of $\mathbf{R}$, $\widehat{s}_{n,i}$ is the n^{th} detected symbol for i^{th} branch and $\widehat{zf}_{m,i}$ is the center of the constrained sphere defined as (5).

$$\begin{aligned} \widehat{zf}_{m,i} &= zf_{m,i} - \sum_{l=m+1}^{M} \frac{r_{m,l}}{r_{l,l}} \left(zf_{l,i} - \widehat{s}_{l,i} \right), \\ \widehat{zf}_{M,i} &= zf_{M,i}. \end{aligned} \tag{5}$$

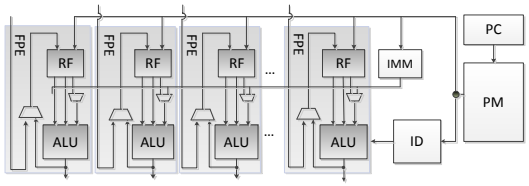
The estimated $\widehat{\mathbf{s}}$ is then selected by *Min* according to

$$\widehat{\mathbf{s}} = arg\ min_{i\in[1,L]}(APED_{N,i}), L = M_c^{NFS}. \tag{6}$$

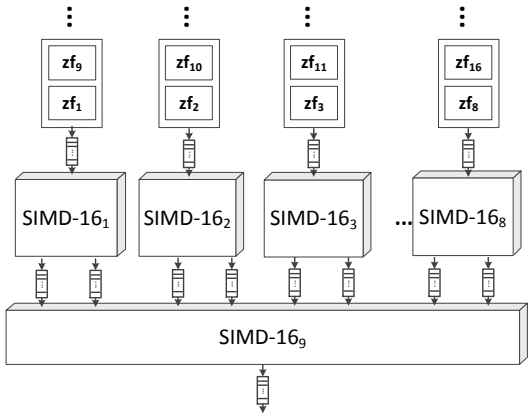
FSD is the lowest complexity quasi-ML detection algorithm on record, however real-time implementation for standard such as 802.11n is a challenging implementation problem. Specifically, the duplication of the FSD tree in Fig. 1 for each of the 108 OFDM sub-carriers in 4×4 16-QAM 802.11n [8] to achieve the requisite 480 Mbps is highly challenging SDR engineering problem. Indeed the FPGA-based solution in [2] is the only approach to achieve it, by exploiting a high performance network of SIMD processors, as shown in Fig. 2(a).

As shown in Fig. 2, each SIMD way is composed of an FPGA Processing Element (FPE), with a centralized Program Counter (PC), Program Memory (PM), Instruction Decoder (ID) and Immediate Memory (IMM). Both the FPE and the SIMD conglomerate are highly configurable in terms of, for instance, the number of SIMD ways, PM and Register File (RF) sizes [2]. This has allowed the architecture to be tuned at design time to enable such high performance but also imposes a heavy manual development load on the designer, in terms of manual Register Transfer Level (RTL) design and assembly level program of highly parallel architectures (up to 400 FPEs and 9 SIMDs in the architecture in Fig. 2(b)). This necessity for manual design clearly negates the flexibility and productivity benefits of software-radio applications and can only be resolved by automated hardware and software design technology.

Unfortunately, such approaches are currently lacking. Leading approaches to 'system-level' design of embedded streaming architectures, such as as Daedalus [9] and Nucleus [10], try to close the gap between an application model and its embedded implementation but none is well suited



(a) FPGA SIMD Processor Architecture [2]



(b) Multi-SIMD Architecture for 802.11n

Fig. 2. The FPE-based FSD for 802.11n

to the SDR-based SD problem at hand. Both Daedalus and Nucleus map Kahn Process Network (KPN) models to target platforms, with Nucleus targeting pre-allocated platforms and Daedalus composing platforms from library components before optimizing the solution via automated Design Space Exploration (DSE). Neither of these could avoid the manual architecture design effort currently required. Hence a new approach to synthesizing multi-SIMD architectures for FPGA-based SDR is required.

Although ideally such an approach would generate an absolutely optimal implementation from an application specification, this is in general not feasible and a designer-guided *cooperative* approach is required [11]. Such an approach is described in Section 3.

3. COOPERATIVE SYNTHESIS: METHODOLOGY

3.1. Overview

Any synthesis process which targets FPGA and multi-core processing technologies needs to address a number of key architecture synthesis problems:

- ♢ *Specification* of the behavior of the application to be realized at an appropriate level of abstraction to enable high productivity, automated synthesis.
- ♢ *Allocation* of a set of processing resources to realize the application
- ♢ *Partitioning* of the application model into subparts
- ♢ *Mapping* of the partitioned application onto the allocation
- ♢ *Scheduling* of the operations mapped to each architecture element
- ♢ *Code Generation* of suitable source to allow, for instance, compilers or RTL synthesis tools to create the final architecture.

Whilst these requirements are quite general, there are specific requirements imposed by the use of SIMD processing architectures which the design process must meet:

- ☐ The load assigned to each FPE (i.e. each way of each SIMD) during mapping must be symmetric and load balanced.
- ☐ Given the absence of forwarding logic in the target SIMD architecture [2], interleaved scheduling of the operations mapped to each FPE is required [4].

The cooperative synthesis methodology in Fig. 3 is proposed to achieve these objectives. As this shows, three key pieces of information are input:

- ○ *Application*: A specification of the application behavior
- ○ *FPE*: A specification of the FPE interface and configurable characteristics [2]
- ○ *Constraints*: A set of real-time performance constraints (specifically, a target throughput) which the realisation must achieve

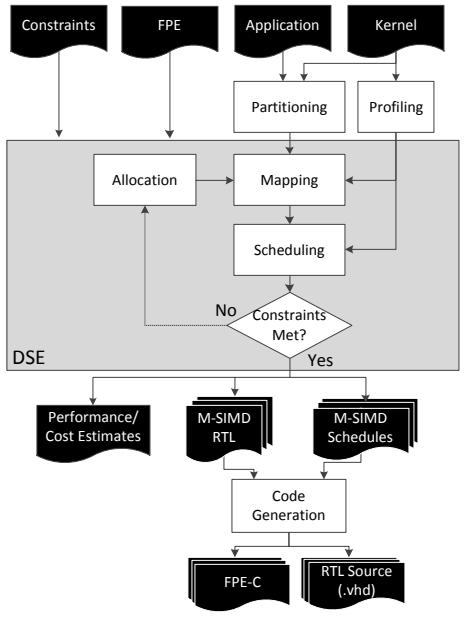


Fig. 3. The Cooperative Synthesis Process

The application is specified using a Synchronous Dataflow (SDF) [12] specification of the application - $G = \{V, E\}$,

where V represents the set of vertices/actors and E represents the set of First-In First-Out (FIFO) edges connecting the actors. When an actor $v \in V$ fires, it consumes tokens from input edges through input ports, transforms them in some way and produces the results onto output edges through output ports. Fig. 1 is an example SDF model of the FSD algorithm.

Note that Fig. 3 also requires a '*kernel*' definition input. This input is the designer's key cooperative step in the design process, identifying a subgraph of the application SDF around which the SIMD implementations are built, which is used to 'anchor' the partitioning and mapping process. Hence, in this case it is a sub-graph of the SDF application model which describes the designer's choice of optimal SIMD processing unit. It should be carefully chosen to maximise SIMD operation and allow effective generation of balanced, workloads and highly interleaved schedules, as described in [4].

As Fig. 3 shows, the application DFG is partitioned according to the kernel structure identified by the designer, as described in Section 3.2, and a platform allocation generated as described in Section 3.4. The kernel DFG is then profiled to derive metrics of the computational and communication costs (see [4] for further details), with this information used to drive the mapping of the partitioned application DFG onto the allocated platform, as described in Section 3.3. Post-mapping, the application partitions mapped to each processor are scheduled, and the performance estimated. If performance is sufficient to meet the real-time constraints specified in the input, the implementation proceeds to code generation, otherwise and iterative re-allocation, mapping and scheduling Design Space Exploration (DSE) is used until the constraints are met.

3.2. Application SDF Clustering

Application Clustering serves to partition the application SDF model into two parts:

- **S**: The SDF subgraph to be implemented on SIMD structures
- **Z**: The SDF subgraph which is not to be implemented using SIMD structures

The process of deriving **S** and **Z** from **G** and **C** (the kernel SDF) is illustrated in Fig.4 where the step indexes are marked in each block.

a) A trivial sub-graph **Q** is derived from **G** and subtracted from it. The index for the set group **S**, i, is initialized.

b) The intersection set of **Q** and **C** is removed from **Q** and added to $\mathbf{S_i}$ after adding an empty set.

c) By forming the intersection of **Q** and **C**, b) is repeated until the intersection set is empty.

d) If the intersection set of **Q** and **C** is empty, no kernels remain in **Q**, and the union of **Q** and **Z** are formed.

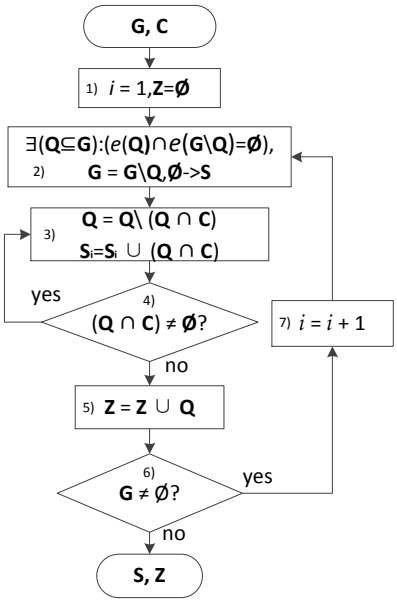


Fig. 4. The Application SDF Clustering Flowchart

e) By judging the input graph set **G**, a) - d) are repeated by increasing the index i until **G** is empty; **S** and the **Z** are subsequently output.

The effect of this clustering on the 802.11n FSD application DFG [1] is shown in Fig. 5 represented as **G**. As this shows, a single branch of the FSD tree has been identified as **C**, the DFG is factored to identify these kernels, mapping $APED_{1.i} - APED_{4.i}$ to $C.i$ from each branch and including these in **S**, with the remaining unclustered actors PP and Min included in **Z**.

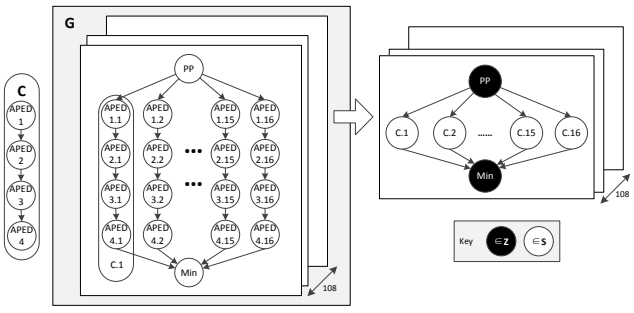


Fig. 5. 802.11n FSD Clustering

After this clustering, the elements of **Z** are mapped to SISD FPE architectures [2][2], with the members of **S** mapped to multi-SIMD architectures. The mapping of the clustered

[1]Note that there are 108 FSD operations in the FSD DFG for 802.11n, 1 per OFDM subcarrier.

[2]For brevity, this straightforward process is not described in this paper.

application DFG to SIMD processors is described in Section 3.3.

3.3. The Application Partitioning

After clustering the application DFG, the kernel set $\mathbf{S}$ is symmetric, and the processing load (i.e. the sets of kernels $\mathbf{S_i} \in \mathbf{S}$) must be mapped to SIMD processors such that the final load is balanced. Essentially this means that the kernels from $\mathbf{S}$ must be selected and assigned to groups corresponding to the target SIMD processors, in a manner which results in an equal number of kernels being contained in each group.

To ensure completely balanced loads for each SIMD processor, a number of kernels n must be mapped per SIMD lane; since the number of SIMD (and hence the total number of SIMD lanes) may vary according to the allocation process, this number may be variable and cannot be statically defined. However, two facts may be discerned about the number of kernels mapped to each SIMD lane:

- ⋆ It will be an integer
- ⋆ It will be a factor of the total number of kernels such that an equal number of kernels are mapped per lane

Accordingly, a set of division factors can be defined based on $|\mathbf{S}|$ and $|\mathbf{P}|$, the number of SIMD lanes. The division factor set $\mathbf{D}$ is union set of two sub-sets $\mathbf{D_1}$ and $\mathbf{D_2}$ indicating all the kernels and part of kernels of the trivial sub-graph of $\mathbf{G}$ to be exactly divided for all the SIMD lanes given by Eq. (7) and Eq. (8) respectively.

$$\left\{\mathbf{D_1} \subseteq \mathbb{Z} : \sum \mathbf{D_1} = |\mathbf{S_i}| \cdot \left\lfloor \frac{|\mathbf{S}|}{|\mathbf{P}|} \right\rfloor, \forall \mathbf{d} \in \mathbf{D_1}, \mathbf{d} = |\mathbf{S_i}| \right\}, \tag{7}$$

$$\left\{\mathbf{D_2} \subseteq \mathbb{Z} : \sum \mathbf{D_2} = \frac{|\mathbf{S}| \cdot |\mathbf{S_i}|}{|\mathbf{P}|} \% \, |\mathbf{S_i}|, \forall \mathbf{d} \in \mathbf{D_2}, |\mathbf{S_i}| \, | \mathbf{d} \right\}. \tag{8}$$

The division factors $\mathbf{D} = \mathbf{D_1} \cup \mathbf{D_2}$ and Fig. 6 shows the procedure for mapping of the clustered SDF expressed in $\mathbf{S}$ by adding $d \in \mathbf{D}$ kernels in $\mathbf{S}$ to a group of set $\mathbf{P}$ representing the SIMD partitions.

a) Initialise i & j in 1) & 2).

b) In stage 3), $\mathbf{K} \in \mathbf{S_i}$ is mapped to $\mathbf{P_j}$, where $|\mathbf{K}| = d \in \mathbf{D}$. i is increased in 5) only if $\mathbf{S_i}$ is empt and hence there are no kernels available for mapping.

c) Repeat until all the elementary sequences of $\mathbf{P}$ contain $\mathbf{K}$, after which the adopted element d is removed from set $\mathbf{D}$.

d) b) - d) are repeated until either $\mathbf{D}$ or $\mathbf{S}$ is empty.

The output $\mathbf{P}$ defines the mapping of kernels in $\mathbf{S}$ to ways of each SIMD. When applied to the FSD example from Section 3.2, the final partitioning is shown in Fig. 7. From Fig. 5,

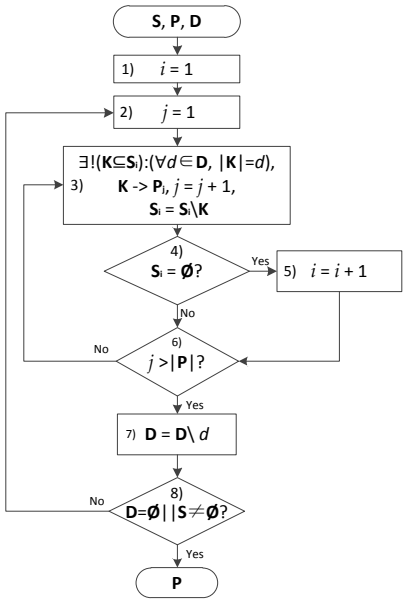


Fig. 6. Application SDF Partitioning

$\mathbf{D} = \{\mathbf{2}\}$ by (8). Hence, each kernel set $\mathbf{K} \in \mathbf{S}$ with $|\mathbf{K}| = 2$ is mapped onto a sequence $\mathbf{P_i} \in \mathbf{P}$, dividing $\mathbf{S}$ into 16 parts (for $|\mathbf{P}| = 16$, two kernel per SIMD lane).

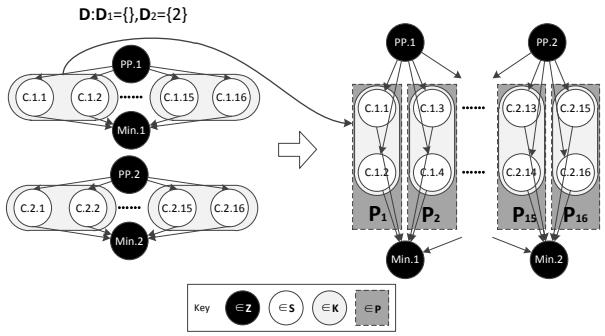


Fig. 7. FSD Application Partitioning

3.4. Allocation, Mapping and Scheduling

After partitioning, the application kernel workload has been exactly balanced across lanes of a single SIMD as defined by $\mathbf{P}$ - all that remains is to create an allocation of multiple SIMD units and map the total application workload thereon. This is performed by an iterative allocation, kernel mapping and scheduling approach outlined in Fig. 8. As Fig. 8 shows, the definitions of the SIMD partition $\mathbf{P}$ and the kernel $\mathbf{C}$ are processed along with constraints on the throughput T and the register file capacity of each FPE R^3.

[3]The register file size is capped since this is most resource expensive aspect of the processor architecture.

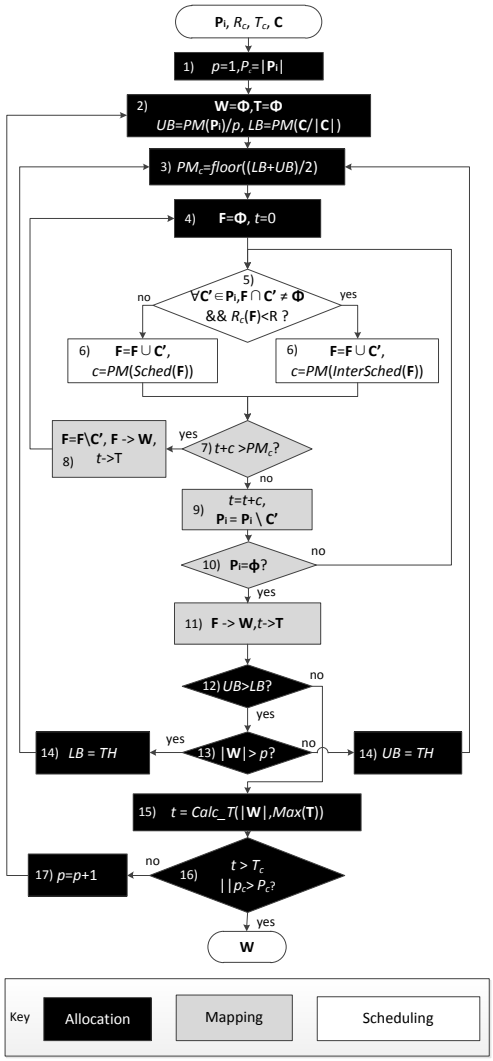


Fig. 8. Iterative Allocation, Mapping and Scheduling

The outermost allocation process (tasks denoted by white text on black background) initally allocates a single SIMD and increases the number via Binary-Constrained Search (BCS) [13] mapping, scheduling and reallocation constrained by the PM cost[4] until either the throughput constraint T_c is met, or a maximum number of processors, P_c reached. The final allocation is stored in the set $\mathbf{W}$, where $|\mathbf{W}|$ is the number of allocated SIMDs, with each element $w \in \mathbf{W}$ a sequence describing the processing schedule for SIMD w. The allocation process proceeds as follows:

a) The initial allocation p and upper bound on the number of SIMDs P_c are initialized as 1 and $|\mathbf{P_i}|$ respectively. The upper and lower bounds UB and LB respectively on program memory for the BCS are initialized profiling the kernels in $\mathbf{P_i}$ and the definition in $\mathbf{C}$ (step 2)).

b) The PM constraint on the BCS process PM_c is updated (3)) and the allocation set $\mathbf{G}$ initialized (4)), after which the allocation $\mathbf{G}$ and mapping $\mathbf{P}$ are used to map and schedule the implementation.

c) On completion of mapping and scheduling, if the PM requirements are in excess of the program threshold PM_c, (7)), a new allocation is generated. Alternatively, by comparing $|\mathbf{W}|$ at P_c it may be determined if the number of SIMDs has exceed the constraint, in which case BCS bounds UB and LB are updated in 14) before a new allocation is generated.

d) Once the $LB > UB$ in step 12), the throughput is determined according to $|\mathbf{W}|$ and the maximum recorded program memory size in $\mathbf{T}$ (15)). If throughput exceeds T_c, or $p \geq P_c$, $\mathbf{W}$ is output or reallocation (b) - d) occurs.

Mapping and scheduling (grey and white boxes respectively) form a single conglomerate process which maps $\mathbf{P_i}$ onto $\mathbf{F}$ representing the allocation, subject to constraint by PM_c. The procedures is as follows:

a) Each $\mathbf{C}' \in \mathbf{P_i}$ is mapped onto $\mathbf{F}$, and scheduled according to the process in [4], with estimates of PM cost c derived as in 6).

b) According the the availability of RF resources, and the presence of kernels of the same class on $\mathbf{F}$, the kernels added are either scheduled separately or in an interleaved fashion to those already mapped to $\mathbf{F}$ in 6).

c) If resulting PM cost exceeds the constraint PM_c, $\mathbf{F}$ is added to $\mathbf{W}$ (8)), t recorded in $\mathbf{T}$, the above procedures from a) - c) are repeated.

d) Otherwise, PM cost t is updated in 9) and the kernel $\mathbf{C}'$ from $\mathbf{P_i}$ is mapped onto $\mathbf{F}$ continuously until $\mathbf{P_i}$ is empty, after which $\mathbf{F}$ is added to $\mathbf{W}$ and t is recorded in 11).

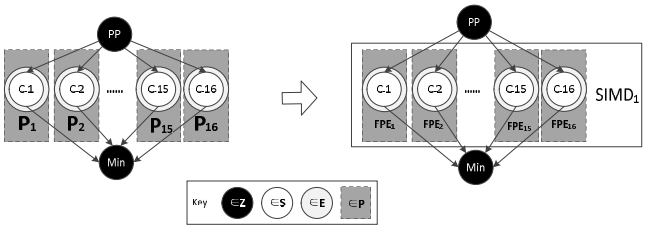


Fig. 9. Iterative FSD Allocation, Mapping and Scheduling Result

Fig. 9 illustrates the result of applying this procedure to Fig. 7. After iterative allocation, mapping and scheduling,

[4]Program Memory is used as a proxy for meeting the throughput constraint, as the number of instructions defines throughput.

the multi-SIMD architectures generated meet the user specified constraints on throughput and program memory size, if indeed such implementations are achievable.

4. AUTOMATED COOPERATIVE SYNTHESIS: FSD AND SSFE

To demonstrate the effectiveness of the proposed cooperative synthesis approach for automatically generating real-time SD implementations using multi-SIMD architectures, it is applied in this section to FSD and SSFE schemes for 802.11n [8].

4.1. Cooperative Synthesis: FSD

When the 4×4 16 QAM FSD algorithm model and kernel definition in Fig. 5 are targeted towards 16 way SIMD processors (up to 64 element RF, 1024 location PM) on Virtex-6 FPGA, the implementations derived are as described in column 2 of Table 1. As this shows, the real-time performance target of 480 Mbps for 40 MHz HT frame 802.11n [8] is comfortably exceeded, with 503.3 Mbps achieved. When extended to 64 QAM, a similar result is observed.

Table 1. 4×4 FSD Implementation Results

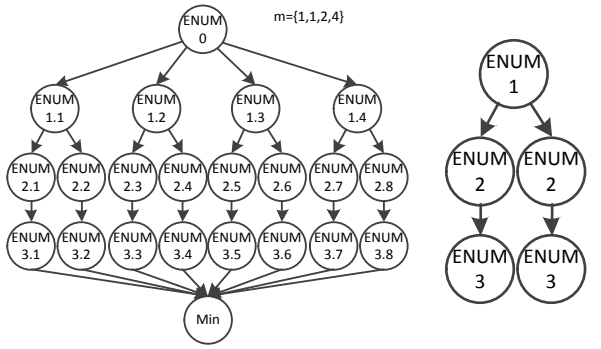
Modulation	16-QAM	64-QAM
SIMDs	9	10
DSP48E1	144	160
Clock (MHz)	322	298
Throughput (Mbps)	503.3	500.9
Instructions	184	184
LUTs ($\times 10^3$)	24.42	27.08

4.2. Cooperative Synthesis: SSFE

SSFE adopts a highly configurable tree structure, where the number of enumerated symbols from each node is expressed using a sequence m. Let the numbers in m also stand for the nodes, Fig. 10(a) describes the structure of the SSFE tree for $m = [1, 1, 2, 4]$. Similarly, the structure of an example SSFE kernel, with structure $[1, 1, 2]$ is shown in Fig. 10(b).

A series of SSFE schemes for 16 and 64 QAM 4×4 MIMO have been implemented on Virtex-6 FPGA exploiting the cooperative synthesis toolset as outlined in Tables 2 and 3 respectively. As this shows, real-time performance for 802.11n is achieved for all SSFE schemes. In addition, as compared to the manually created Very Long Instruction Word (VLIW) implementations in [14], Table 3 demonstrates the substantial throughput increases achieved by utilizing the FPGA-based multi-SIMD implementation approach.

The cooperative synthesis performance results for different SSFE schemes of 16QAM and 64QAM 4×4 MIMO



(a) SSFE DFG (b) $\{1, 1, 2\}$ Kernel

Fig. 10. SSFE Cooperative Synthesis Inputs

all achieve the real-time performance on FPGA based soft-core streaming processor compared to the manual VLIW approaches [14].

Given these FSD and SSFE synthesis results, the effectiveness of the proposed cooperative synthesis scheme is apparent. It has not only demonstrated the unique capability to automatically generate real-time implementations of SD detection for 4×4 802.11n, but that it is highly scalable across SD tree topologies and schemes. Given the inability of any other SDR platform to even support manually created real-time detectors for such schemes, these are truly a unique set of capabilities.

5. CONCLUSION

This paper has presented the first automated approach to generation of real-time software defined MIMO SD detectors for modern standard such as 802.11n and LTE. By targeting multi-SIMD architectures on FPGA via a cooperative, designer guided synthesis process. By exploiting dataflow modeling semantics and designer-identified SIMD processing kernels, real-time 480 Mbps implementations for detection of 4×4 16 QAM 802.11n MIMO systems has been achieved.

It is worth noting that whilst this work has presented how these may be generated automatically, it is highly unique in that respect. Indeed, recorded instances of real-time detectors for 802.11n are absent beyond those supported by the platform targeted here.

6. REFERENCES

[1] 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, Feb. 1981.

[2] X. Chu and J. McAllister, "Software-Defined Sphere

Table 2. 4×4 16-QAM SSFE Implementations

Scheme	[1,1,1,1]	[1,1,1,4]	[1,1,2,4]	[1,2,2,4]	[1,1,1,8]	[1,1,2,8]	[1,2,4,8]
Kernel	[1,1,1,1]	[1,1,1]	[1,1,2]	[1,2,2]	[1,1,1]	[1,1,2]	[1,2,4]
SIMDs	1	3	5	8	5	9	40
DSP48E1	16	48	64	128	64	144	640
LUTs ($\times 10^3$)	2.14	8.53	14.22	22.75	14.22	24.42	113.19
Instructions	141	225	214	308	314	357	545
Clock (MHz)	341	339	318	314	318	322	255
Throughput (Mbps)	618.6	482.3	498.9	521.4	510.1	489.9	500.8

Table 3. 4×4 64-QAM SSFE Implementations

Scheme	[1,1,1,1]	[1,1,1,4]	[1,1,2,4]	[1,2,2,4]	[1,1,1,8]	[1,1,2,8]	[1,2,4,8]
Kernel	[1,1,1,1]	[1,1,1]	[1,1,2]	[1,2,2]	[1,1,1]	[1,1,2]	[1,2,4]
SIMD	1	2	3	5	3	6	20
DSP48E1	16	32	48	80	48	96	320
LUTs ($\times 10^3$)	2.14	5.67	8.53	14.22	8.53	17.02	56.59
Instructions	141	168	357	308	499	357	545
Clock (MHz)	341	339	339	333	339	315	276
Throughput ([14]) (Mbps)	927.9 (125.3)	581.3 (48.5)	547.1 (37.4)	520.1	489.3	508.5	485.9

Decoding for FPGA-based MIMO Detection," *IEEE Trans. Signal Processing*, Accepted. 2012.

[3] J. H. Reed, *Software Radio: A Modern Approach to Radio Engineering*, Prentice Hall, 1 edition, 2002.

[4] C. Zheng, J. McAllister, and Y. Wu, "A Kernel Interleaved Scheduling Method for Streaming Applications on Soft-core Vector Processors," *2011 International Conference on Embedded Computer Systems (SAMOS)*, pp. 278–285, July 2011.

[5] A. Sibille, C. Oestges, and A. Zanella, *MIMO: From Theory to Implementation*, Academic Press, 1st edition, 2010.

[6] L. Barbero and J. Thompson, "Fixing the Complexity of the Sphere Decoder for MIMO Detection," *IEEE Transactions on Wireless Communications*, vol. 7, no. 6, pp. 2131–2142, June 2008.

[7] D. Wübben, R. Böhnke, J. Rinas, K. D. Kammeyer, and V. Kühn, "Efficient algorithm for decoding layered space-time codes," *IEE Electronic Letters*, vol. 37, no. 22, pp. 1348–1350, Nov 2001.

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

[9] H. Nikolov, M. Thompson, and et. al, "Daedalus: toward composable multimedia MP-SoC design," *Proceedings of the 45th annual Design Automation Conference*, pp. 574–579, 2008.

[10] J. Castrillon, S. Schürmans, and et. al, "Component-Based Waveform Development: The Nucleus Tool Flow For Efficient and Portable Software Defined Radio," *Analog Integr. Circuits Signal Process.*, vol. 69, no. 2-3, pp. 173–190, December 2011.

[11] J. Cong, B. Liu, and et. al, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, April 2011.

[12] E.A. Lee, "Synchronous Data Flow," *Proceedings of the IEEE*, vol. 75, pp. 1235 – 1245, September 1987.

[13] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and design of embedded systems*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[14] M. Li, B. Bougard, and et. al, "Selective Spanning with Fast Enumeration: A Near Maximum-Likelihood MIMO Detector Designed for Parallel Programmable Baseband Architectures," *IEEE International Conference on Communications, ICC '08.*, pp. 737 – 741, May 2008.