VECTOR-BASED ACCELERATION IN THE IBM POWEREN™ PROCESSOR TO ENABLE SOFTWARE-DEFINED RADIO

Jeff Derby (IBM, Research Triangle Park, NC; jhderby@us.ibm.com); Timothy Heil (IBM, Rochester, MN; timothy.heil@us.ibm.com); Michele Franceschini (IBM, Yorktown Heights, NY; franceschini@us.ibm.com); Anil Krishna (IBM, Research Triangle Park, NC; krishnaa@us.ibm.com); Robert Montoye (IBM, Yorktown Heights, NY; montoye@us.ibm.com); Dheeraj Sreedhar (IBM, Bangalore, India; dhsreedh@in.ibm.com); Augusto Vega (IBM, Yorktown Heights, NY; ajvega@us.ibm.com); Hangu Yeo (IBM, Yorktown Heights, NY; hangu@us.ibm.com); Charles Johnson (IBM, Yorktown Heights, NY; charliej@us.ibm.com)

ABSTRACT

This paper describes a hardware processor architecture that can achieve the holy grail of SDR: a general-purpose processor that enables high-throughput, low-latency processing for layer 1 through layer 3 for a complete basestation on a chip, while meeting the highest demands of WCDMA and LTE-advanced.

The starting point for this architecture is IBM's PowerEN processor, a multicore, massively multithreaded platform that employs general-purpose Power processor cores and includes extensions that address functions appropriate for wired network-edge applications. We consider here a potential evolution of PowerEN to address layer 1 functions, implementing SDR on a general-purpose processor platform for 3G and 4G basestations. Incorporation of a vector-based accelerator (VBA) enables an enhanced PowerEN to support the evolving 3G and 4G standards at a sufficient level of performance for LTE-advanced and beyond, in a fully programmable and scalable fashion. VBA has the appearance of a traditional SIMD unit attached to each general-purpose processor, but includes features that dramatically increase its processing capability for SDR. We provide details of the VBA architecture and describe aspects of programming models for its use. In addition, we describe aspects of the implementation of a 4x4 MIMO LTE uplink receive chain on an enhanced PowerEN platform

1. INTRODUCTION

The rapid proliferation of tablets, smartphones, and other user devices that support high-speed wireless connectivity has led to an explosion of traffic types and traffic volume in wireless networks. It also has the potential to drive a tighter coupling between the signal- and network-processing functions traditionally implemented in basestations using application-specific hardware platforms, and information-technology (IT) functions that have usually been implemented in or behind the core network using general-purpose processing platforms. This latter factor significantly increases the attractiveness of software-defined radio (SDR) with the layer 1 functions running on a general-purpose processor platform. This would enable the signal processing for layer 1 as well as higher-layer functions and IT-oriented functions to be run on a common computing platform with a single architecture, a single toolset, and a single programming model.

The explosion of traffic volume and the associated demand for higher data rates per user have led to new OFDM-based standards as well as evolution of existing WCDMA-based standards; this also increases the attractiveness of SDR, since ideally an SDR-based basestation should have the capability to support different layer 1 mechanisms in a flexible and scalable fashion. At the same time however, the new standards, and their projected evolution over the next 10 years, drive performance requirements for any implementation of layer 1 function, traditional or SDR-based, that will be difficult to meet, and the possibility of satisfying them with a fully programmable SDR solution, especially one based on use of a general-purpose processor platform, may seem remote at best.

This paper describes a hardware processor architecture that can achieve the holy grail of SDR: a general-purpose processor that enables high-throughput, low-latency processing for layer 1 through layer 3 for a complete basestation on a chip, while meeting the highest demands of WCDMA and LTE-advanced. The starting point for this architecture is IBM's PowerEN processor [1],[2], a multicore, massively multithreaded platform that employs general-purpose Power processor cores and includes extensions that address functions appropriate for wired network-edge applications.

The layer 1 functions implemented in an SDR platform can be viewed as a set of “network-edge” functions specific

to the boundary between wireless and core networks. We describe here a set of possible extensions to the PowerEN platform that would enable it to support the evolving 3G and 4G standards at a level of performance appropriate for what is projected through LTE-advanced and beyond, in a fully programmable fashion. The key extension is a vector unit added as an "auxiliary execution unit" (AXU) to each of the Power processor cores on the chip and providing vector-based acceleration (VBA). The vector unit has the appearance of a traditional SIMD unit incorporated in a general-purpose processor, but it includes a number of features that dramatically increase its processing capability for SDR. These include a very large architected register file, a means for dynamically addressing data elements in this register file, gather and scatter functions for data in the register file, and high-bandwidth, low-latency data movement into and out of the register file.

Following an overview of the PowerEN platform and a detailed description of the VBA unit, we discuss in Section 4 the implementation using VBA of two classes of algorithms of particular importance for LTE, namely FFT and matrix inversion; we also present a brief summary of results developed in [3] for the VBA implementation of the LTE turbo decoder. An assessment of the processing load of an LTE digital baseband implementation on an evolved PowerEN platform, for a single sector supporting 4x4 MIMO in a 20MHz channel, is presented in Section 5. These results, together with results presented in [3] for WCDMA chip-rate processing implemented on VBA, indicate that the essentially general-purpose processor platform proposed here is capable of supporting both WCDMA and LTE at appropriate performance levels in a fully programmable and scalable fashion.

2. PowerEN OVERVIEW

The PowerEN processor is a multicore, massively multithreaded platform that employs general-purpose Power processor cores and includes extensions that address functions appropriate for wired network-edge applications. A photograph of the chip is shown in Fig. 1. The chip integrates 16 Book-E 64-bit Power processor cores, identified as A2 cores in the figure, along with a set of hardware accelerators supporting symmetric and asymmetric cryptography, multi-pattern search for deep-packet inspection, gzip-style compression and decompression, and XML processing. Also included in the chip are four 10Gbits/s Ethernet interfaces with a controller (HEA) that provides frame classification and protocol acceleration capabilities. The chip is implemented in 45nm and in typical applications runs at 2.3GHz. Details on chip area, and on power consumption in different configurations, can be found in [2].

The A2 cores support simultaneous multithreading (SMT) with four threads per core. The A2 is a small, efficient, scalar, integer core, but includes an AXU interface that

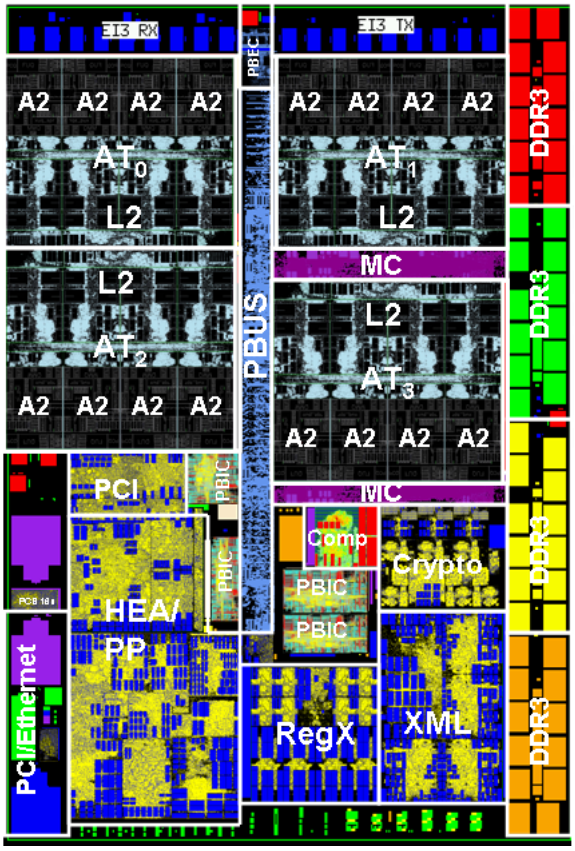

Figure 1. PowerEN die photo

permits attachment of execution units providing additional capabilities. In PowerEN, the AXUs support scalar floating-point instructions.

The A2 cores are organized into AT nodes of four cores each, sharing a 2MB L2 cache. The Pbus in PowerEN connects the four AT nodes, the accelerators and HEA noted above, interfaces to external memory, and other I/O. It provides hardware-managed coherence for all accesses to data by any of the attached entities. It also implements an architecturally defined mechanism for communication between A2 cores and accelerators, with accelerators running in the same virtual address space as the invoking threads, and each accelerator supporting complete address translation. In addition, the Pbus supports *cache injection*, which allows data from an accelerator or an Ethernet interface to be written directly into the L2 cache of a target thread, thereby avoiding unnecessary traffic to and from external memory, with a significant reduction in latency. Supporting low-latency inter-thread communication is important as well. The Pbus provides for low-latency data transfers between L2 caches. For even lower latency, applications can group related threads on top of the same L2 cache.

The latency to transfer data from one thread to another on the same node is trivial compared to the processing on each thread. On PowerEN, this has been measured at around

300 processor cycles with a lockless single-produce, single-consumer message queue.

In the Power architecture this can be done using normal loads and stores to access the message. The producer still requires a light-weight sync when writing a message, to ensure all prior data is written first. The consumer must use an isync to ensure loads after the message is received do not execute out-of-order early, and receive stale data.

The message transfer time of 300 cycles is measured from the start of message send on the producer, until end of message receive on the consumer on the PowerEN chip. The consumer is not stalled for most of the transfer time, and spends about 130 cycles sending the message.

This assumes the worst case, in which the consumer must wait for the producer. Preferably, the producer runs ahead of the consumer, and the consumer finds the data available immediately, in which case the consumer takes about 70 cycles to determine that data is present and receive a message. The remainder of our analysis conservatively assumes the longer scenario.

Several features aid communication between L2 caches. To aid with streaming data, cores can issue special touch instructions which mark cache lines as least-recently-used. By marking lines as ready for replacement, the streaming data is confined to a single way of the L2 cache. This can be used either prior to accessing the data, as an actual prefetch, or after using the data, as a replacement hint. The Replacement Management Table is a similar but more automated mechanism where particular address ranges can be confined to one or more ways of the L2 cache.

Prefetching, whether software-driven or hardware-driven, also aids in shipping data streams between L2 caches. In particular, prefetching helps when it is most needed — when the producer is getting ahead of the consumer.

Given PowerEN as an existing platform, we consider a possible evolution that would enable it to address a broader set of network-edge functions, and in particular to support SDR in wireless basestations. The basic platform architecture would remain as described above, with nodes consisting of a small number (perhaps two or four) of simple, efficient cores with a shared L2 cache. Key extensions will include:

- use of vector-based acceleration, with a vector unit attached to each processor core as an AXU;
- potential modification of the processor cores to maximize the efficiency with which the vector AXUs are used (thus the cores are referred to be below as A2+);
- incorporation of appropriate antenna interfaces (e.g. CPRI) attached to the Pbus with cache-inject support (so that antenna data can be moved directly between the interface and L2 caches);
- possible inclusion of hardware accelerators for certain functions specific to basestation applications (but ideally all functions would be implemented in software).

It is anticipated that an enhanced PowerEN incorporating these extensions would be implemented in 22nm and would run at 2.3GHz.

3. VECTOR-BASED ACCELERATION

VBA is an AXU attached to an A2+ core. It takes and executes instructions fetched and passed to it by the A2+ core. VBA is derived from VMX, the Power SIMD architecture [4], but incorporates several significant extensions and innovations. To begin with, the SIMD with is increased to 32 bytes, with corresponding subword parallelism (e.g. 16-wide for 16-bit halfwords, 8-wide for 32-bit fullwords). Second, native complex-arithmetic is included, together with a set of special instructions useful for despreading and related functions; these are described below in Section 3.1. Most significantly, the vector unit includes an extremely large register file, the *vector string register file* (VSRF), consisting of 2K 32-byte registers, together with an indirection mechanism for dynamically addressing data contained in the register file; these features are discussed in Section 3.2. The benefits provided by use of the large register file and associated indirection are discussed in the context of an "in-line" programming model for use of VBA in Section 3.4 and are also highlighted by the algorithm examples in Section 4.

3.1. Computational Facilities

VBA begins with VMX. The VBA doubles the vector size, from 16B-wide to 32B-wide, and the subword parallelism as noted above. Support for fixed-point and single-precision floating-point arithmetic are as in VMX, including fused multiply-add instructions for floating point and a suite of fused multiply-add and "multiply-sum" instructions for fixed-point. The standard VMX permute facilities, permitting shuffling of data from two source registers, is also available. However, VBA introduces a much more general and powerful mechanism for reorganization of data in its register file, as described below in Section 3.2.

The vector unit introduces native support for complex fixed-point arithmetic. Real and imaginary parts are assumed to be interleaved for data in memory and also for data in the VSRF. Among the instructions provided is a set that implements fused complex fixed-point multiply-add operations. The vector unit also introduces instructions that implement correlations with sequences of ± 1 , e.g. for spreading and despreading in WCDMA. Use of these instructions assumes that the sequence is stored as a bit-vector (say 0 for +1, 1 for -1) in the VSRF and leverages the ability to dynamically address data in the VSRF to "walk through" the bit-vector some number of bits at a time as appropriate. An example of the use of these instructions for WCDMA despreading is presented in [3].

3.2. The VSRF with Indirect Access

The use of a large register file with software-managed indirect access was introduced previously in the context of a SIMD DSP architecture [5],[6]. Derby et al. described a different form of indirection that enabled a 5-bit register-operand field in a standard VMX instruction to specify one out of 1K or more architected registers [7]. These two concepts are combined in the indirection architecture employed with VBA.

The VSRF consists of 2K 256-bit registers, providing an aggregate 64KB of storage. It is physically arranged as a set of eight subarrays, each containing 256 registers, and each with four independent read ports and one write port. While the VSRF contains more storage than the L1 data cache in most processors, it is in fact a register file. It can supply the contents of up to four registers per cycle as input operands to instructions, and access latency is completely hidden by pipelining and bypassing.

Access to data in the VSRF is via an indirection mechanism, which uses a special set of 32 *map registers* (MRs) that contain addresses that are offsets from the VSRF origin. MRs are 128 bits wide and support subword parallelism. A map register may contain eight 16-bit byte pointers or four 19-bit bit-pointers; the pointers contain byte or bit offsets from the origin of the register file. The contents of the map registers are managed by software in SIMD fashion using a set of new "map management" instructions; these include arithmetic operations on the entries in an MR, and moves between an MR and the upper 16B in a register in the VSRF.

The indirection mechanism has two basic forms, referred to as "operand-associated indirection" and "generalized indirection". Operand-associated indirection enables the specification of one out of 2K registers in a 5-bit register operand field. A 5-bit operand selects one 16-bit pointer in an MR, and the pointer indicates the VSRF register accessed. Since there are eight 16-bit pointers in one MR, a 5-bit operand indexes four MRs. Four MRs are therefore logically grouped into an *operand map*. There are four operand maps, one for each of the four operand positions in a VBA instruction (three vector sources and one destination). Defining separate maps for inputs and outputs allows register specifiers to be reused in different contexts, easing register specifier selection. The four operand maps use the first 16 MRs, with the remaining 16 MRs available for generalized indirection, temporaries and immediate values.

An example of the use of operand-associated indirection is shown in Fig. 2. The four maps, each with 32 entries, are indicated in the figure. MR0 through MR3 contain the 'VT map' for the target register operand, MR4 through MR7 contain the 'VA map' for the A source register operand, MR8 through MR11 contain the 'VB map' for the B source register operand, and MR12 through MR15 contain the 'VC map' for

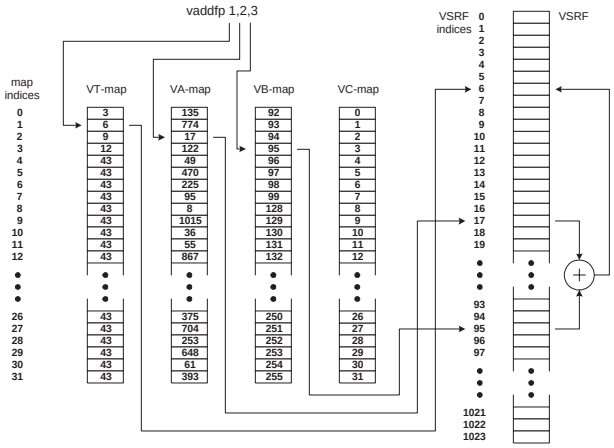

Figure 2. An example of operand-associated indirection

the C source register operand. The map register entries for operand-associated indirection all point to registers in the VSRF and so as byte counts from the origin of the VSRF must be equal to 0 mod 32. The example shown is simplified, in that the map entries are shown as being VSRF register indices, and also in that the VSRF is shown as having 1K registers. The instruction at the top of the figure is a VMX (and so also VBA) floating-point add instruction, with '1' encoded in the target register operand field, '2' encoded in the A source register operand field, and '3' encoded in the B source register operand field; for this instruction, there is no C source operand. As shown in the figure, the '1' from the target operand field is used as an index into the corresponding map ('VT map'); the referenced map entry contains a pointer to VSRF register 6. Similarly, the 'A' source register is identified as VSRF register 17 using the 'VA map', and the 'B'source register is identified as VSRF register 95 using the 'VB map'. Thus the instruction computes the 8-wide floating-point addition of the contents of VSRF registers 17 and 95 and places the result in VSRF register 6.

Generalized indirection is an extremely powerful mechanism for reorganizing data in the VSRF, permitting access to up to eight data elements at arbitrary locations in the VSRF with a single instruction. The gather instructions will place the addressed data elements in a specified order in a target register in the VSRF. For example, a *gather words* instruction, '`vgetw VT,MA,MB`', will take eight pointer values from map register MA and interpret the eight values in map register MB as lengths (in bits), extract eight data elements with the specified lengths from the specified locations in the VSRF, and place them in 32-bit slots in target register VT in the VSRF in the order in which the pointer values appeared in map register MA. It is also important to note that for certain gather instructions, the map-register operands are interpreted as having four entries in 32-bit slots, with the entries in the MA operand being bit counts rather than byte counts from the origin of the VSRF. For these instructions, the fields to be gathered need not be aligned on byte boundaries in the VSRF.

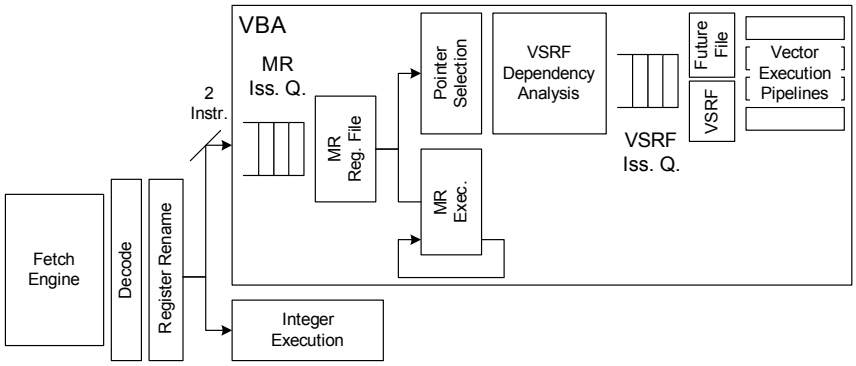

Figure 3. Microarchitecture model of the VBA

This can be particularly useful for implementing bit interleavers, e.g. as part of a turbo encoder.

3.3. VBA Microarchitecture

Performance analysis is based on the A2+ core and attached VBA, as shown in Fig. 3, a two-wide out-of-order processor. Because VSRF accesses depend on MR values, VBA execution proceeds in two stages. First the MR registers are accessed to determine which VSRF registers will be read or written. Vector execution then proceeds in the second stage.

The fetch engine can enqueue two instructions per cycle into the MR issue queue inside the VBA. After issuing and reading their input MRs, map management instructions are executed immediately. Other VBA instructions proceed through the select pointers stage, which determines the actual VSRF registers read/written, based on the pointers in the MR values read from the register file. Dependency analysis then determines which prior instructions, if any, each vector instruction depends on. Instructions are then enqueued a second time to wait for vector register dependencies and an appropriate execution pipeline. Once instructions issue, they read their input values and execute.

Map management instructions may issue out-of-order to the MR execution pipe. These instructions have single-cycle latency, and one may issue each cycle. Due to out-of-order execution, map management instructions issue early, and are rarely a source of stalls. Other VBA instructions must issue in-order because the accessed vector registers are not yet known.

Since integer and VBA instruction can be freely intermixed, the integer execution engine can, in general, flush and restart the instruction stream at any point. The VBA must be able to roll-back register values until instructions are committed. Normal register renaming provides this capability for MR instructions. The A2+ core renames the 32 architected MRs to 64 physical MRs prior to handing the instructions to the VBA.

Register renaming is impractical for VSRF registers because the 2K architected registers would require a highly-ported rename table with 2K entries. Instead, VBA employs a future file. The future file holds vector register values until the producer commits. Register state may be rolled back by invalidating future file entries after the flush point. After commit, future file entries are spilled to the VSRF, which holds only committed architected state.

During dependency analysis, instructions are assigned a slot in the future file for their output value. Instructions that produce no VSRF value (e.g. stores) need no future file entry. Instructions that produce multiple output values (e.g. 64B cache line loads, which produce two 32B registers) receive multiple future file entries.

Instructions determine the future file entry of prior in-flight instructions that produce needed input values. After issue, instructions read their inputs from the future file, the bypass network or the VSRF as needed.

Out-of-order vector execution is a real possibility, since most of the mechanisms to support out-of-order vector issue exist. However, the SDR algorithms that VBA targets can be well-scheduled. Since out-of-order execution provides little advantage, projections are based on in-order vector issue.

The performance projections reported below are based on a VBA capable of issuing 2 vector instructions per cycle. The four VSRF ports limit issue to four total register inputs between the two instructions. This limit is rarely exceeded. Generalized indirection operations that read more than four registers use two issue slots. Reasonable structural limitations are also assumed. In particular, the single permute unit limits the number of gather operations per cycle to one.

3.4. Programming Model

Although the VBA can be programmed using a completely traditional programming model, with memory-resident data loaded and stored as needed, this underutilizes the VSRF. Substantial performance gains can be obtained by loading relatively large blocks of data into the VSRF a cache-line at a time, operating on the entire block of data, keeping interme-

diate results in the VSRF, and storing the final results to memory a cache-line at a time. For example, a 2K-point FFT can be implemented with the data array and the twiddle factors loaded to the VSRF at the outset, the data array maintained in the VSRF as it is updated through the FFT stages, and the output data array stored to memory at the very end. The pointer values that implement the varying access patterns to the data and twiddle arrays are maintained in map registers and updated using map management instructions. This example is discussed in more detail below in Section 4.1.

The VBA provides further advantage when the produced result is used as the input to another function, for instance, passing the FFT output into subcarrier demapping in an LTE uplink receiver. In this case there is no need to store the output of the FFT; the next function is merely given a pointer to its input within the VSRF using an MR.

This in-line acceleration model is the ideal that VBA enables. A sequence of functions, which traditional implementations would offload to hardware, with frequent coordination and data movement between processor cores and accelerators, are instead implemented in software in a single A2+ core with VBA. Data remains local, ideally in the VSRF and if necessary in the local L2 cache, and coordination is handled through normal program flow.

Three additional points with respect to the programming model are worth noting. First, it remains a load/store model, with load and store instructions used to move data into and out of a vector unit's VSRF, and, given the platform's hardware-managed coherency, with no need for software to have knowledge of the precise location of a block of data to be accessed. Second, capabilities needed to program VBA using high-level language, including allocating blocks of registers to be used by arrays and indexing arrays in the VSRF in the usual way, would be available in compilers targeting the VBA [8],[9]. Finally, use of VBA enables solutions that are highly scalable, since each VBA can provide acceleration for any of the desired functions, depending on the code it executes.

4. ALGORITHM EXAMPLES

In this section we discuss the VBA implementations of several algorithms whose performance will have a significant impact on the performance of an LTE digital baseband. We focus in particular on FFT algorithms, which are pervasive in LTE, and on matrix inversion, which is employed for channel estimation and MIMO processing. We also refer to some results presented in [3] on the implementation of the LTE turbo decoder using VBA.

4.1. FFT Algorithms

Power-of-2 size FFT algorithms are used extensively in LTE. In addition, the SC-FDMA format used in the LTE uplink requires the use of non-power-of-2 size FFT algorithms. All these algorithms are commonly implemented using fixed-point arithmetic, with 16 bits each for the real and imaginary parts of the data array, and 16 bits each for the real and imaginary parts of the twiddle array. The fixed-point FFT algorithms implemented using VBA take advantage of its native support for fixed-point complex arithmetic noted above in Section 3.1.

With interleaved real and imaginary parts at 16 bits each, the FFT algorithms see the VBA as providing an 8-wide SIMD. There is a natural affinity between the radix-8 FFT and an 8-wide SIMD, in that eight radix-8 butterflies can be executed in parallel in place throughout the FFT, with just one step of shuffling the data array with base-8-digit-reversed indexing. Moreover, a very simple and clean implementation of the shuffling is possible given the capabilities of the VBA.

Consider a 512-point FFT, with three radix-8 stages. Each stage has eight 8-wide radix-8 butterflies. The data array occupies 64 vectors. A decimation-in-time implementation on VBA proceeds as follows:

1. The data array is accessed in sequential fashion for the first stage, which requires no twiddle factors. The first stage requires about 120 cycles.
2. Groups of eight vectors are transposed in the VSRF, using a sequence of gather instructions. Each 8x8 transpose requires 8 cycles.
3. The shuffled intermediate data array from step 2 is the input to the second stage. The access pattern for the array completes the base-8-digit-reversed indexing. This access pattern is implemented by construction and updating of appropriate sets of pointers in map registers. The second stage, including multiplication by twiddle factors and the necessary map management, completes in about 160 cycles. The data array at the output of this stage is in the VSRF in natural order.
4. The third stage completes the FFT. Including multiplication by twiddle factors and the appropriate map management, it completes in about 160 cycles

The complete 512-point FFT executes in about 550 cycles. This assumes that the data and twiddle arrays are already in the VSRF at the outset and that the transformed data array remains in the VSRF at the end. The overhead to load the data and twiddles and to store the result may increase the cycle count by perhaps 15%. However, in the LTE layer 1 FFTs represent one step, or a set of steps, in a sequence of functions applied to the baseband signals. With the in-line programming model outlined in Section 3.4, it is realistic to consider the FFT requiring neither loads from memory nor stores to memory.

Extension of the approach described above to a 1K-point or 2K-point FFT is straightforward, with the addition of a radix-2 stage or radix-4 stage, respectively, following the third stage. The only qualitative difference in the application of steps 1 through 4 is the pattern used at the input to the second stage to access the transposed data in the VSRF. Cycle counts follow approximately the usual $N \log N$ scaling. The 1K-point and 2K-point FFTs are projected to execute in about 1150 cycles and 2500 cycles, respectively.

Extension of the approach described above to non-power-of-2 FFTs is less straightforward. However, it can be shown that if the FFT size is a multiple of eight, then the first two steps from the pure radix-8 algorithm above can be maintained, with construction of the access pattern used at the input to the second stage to access the transposed data based on the set of radices employed in the second and succeeding stages. In this case, there is exactly one data-shuffling step between the first and second stages of the FFT algorithm. If the FFT size is not a multiple of eight, then a second data-shuffling step becomes necessary, between the second and third stages. In all cases, however, the data shuffling operates efficiently within the VSRF, using the permute and gather mechanisms that the VBA makes available for reorganization of data within the VSRF, and with pointer values used by these mechanisms maintained in map registers and managed in SIMD fashion.

4.2. Matrix Inversion

MIMO configurations require matrix inversion for channel estimation and MIMO detection. For 4x4 MIMO, the matrices to be inverted are 4x4 matrices. We consider using floating-point rather than fixed-point arithmetic for matrix inversion.

VBA carries forward the single-precision floating-point capabilities of VMX [4], extended to be 8-wide. These include instructions that implement the following:

- fused multiply-add
- conversion between floating-point and fixed-point
- reciprocal estimate
- reciprocal square root estimate

These instructions are all useful in implementing matrix-inversion algorithms. Note that native support for complex arithmetic is not included in VBA for floating-point operations, based on an assessment of performance vs. area tradeoffs.

Because in the LTE uplink receiver there is nominally one matrix to be inverted for each data subcarrier in each SC-FDMA symbol, the matrix inversion can be implemented efficiently with eight matrices being inverted in parallel, one in each of the eight lanes of the SIMD. If the matrices are not generated in this format, i.e. with the (1,1) elements of eight matrices in one vector, the (1,2) elements of eight matrices in a second vector, etc., then the permute and gather facilities can be used to reorganize them into this form.

The projected performance for inversion of 4x4 matrices is equivalent to about 120k matrix inversions per second. This estimate includes loading the matrices from the L2 to the VSRF, as well as the potential need to reorganize the matrix elements as noted above.

4.3. Turbo Decoder

The turbo decoder is one element that rarely has a true software implementation in SDR platforms that address 4G wireless, because of the throughput that must be sustained. The VBA incorporates hardware that leverages parallelism that is made available by the physical structure of the VSRF, thereby enabling an implementation of the LTE turbo decoder with very high throughput.

The VSRF is partitioned into eight banks, each with its own set of read and write ports. Logic that implements the standard algorithm based on the Max-Log-Map approximation [10] for BCJR [11] is associated with each bank of the VSRF. In addition, the received data block to be decoded is partitioned into partially overlapped sub-blocks, with one sub-block in each bank of the VSRF. In this way, eight decoders can be running in parallel, one associated with each bank of the VSRF and working on one received sub-block. Together they form the equivalent of a single constituent decoder working on the complete received data block

The interleaver and de-inerleaver for the turbo decoder are implemented using the VBA gather capability.

Each iteration of the turbo decoder involves two calls to the constituent-decoder function and two calls to the shuffling function that implements the interleaver and de-interleaver. Current projections indicate that, assuming six iterations of the turbo decoder, a throughput of about 230Mbits/s is achieved for a single A2+ core with VBA. Finally, it is important to note here that all of the necessary arrays, including pointer arrays for the interleavers, are maintained in the VSRF throughout the operation of the turbo decoder. Moreover, the LLRs that are input to the decoder can already be in the VSRF, as the output of a demodulator or similar process, when the decoder is invoked.

Details of the VBA implementation of the turbo decoder are available in [3].

5. LTE DIGITAL BASEBAND

The results presented in Section 4 will now be used to assess the processing load of an LTE digital baseband implementation on an evolved PowerEN platform, and also to consider alternatives for organizing the work given the opportunity to employ an in-line programming model. The configuration to be considered represents a single sector supporting 4x4 MIMO in a 20MHz channel. We focus on the uplink receive chain, since that is where the bulk of the cycles are con-

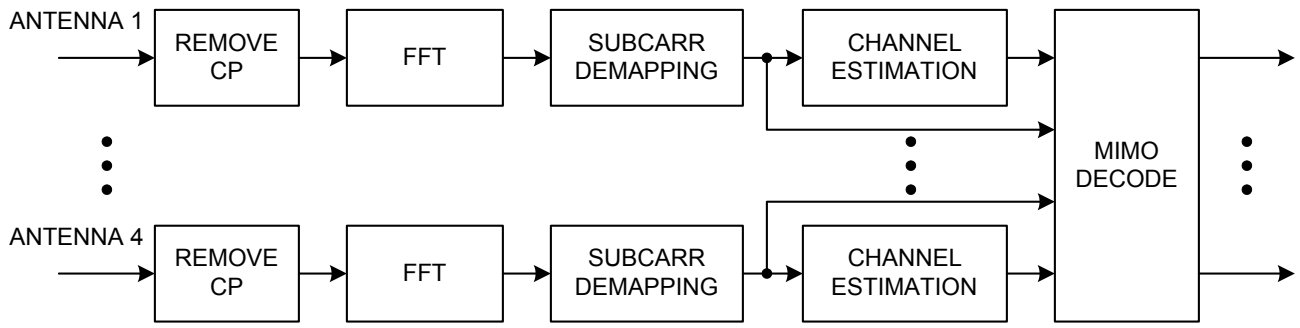

Figure 4. Simplified partial block diagram of an LTE uplink receive chain

sumed.

Fig. 4 shows a simplified block diagram of the LTE uplink receive chain for a 4x4 MIMO configuration, through the MIMO decode step. The IDFT for SC-FDMA, demodulation block, and turbo decoder are not shown. The block labeled "remove CP" may also include compensation for carrier frequency offset.

We consider that in the LTE uplink for a 20MHz channel, seven SC-FDMA symbols, each consisting of 2K+ complex samples with 16 bits for the real part and 16 bits for the imaginary part of each sample (2k+ because of the cyclic prefix), are received from each antenna in every 500μsec slot. These samples would be moved from the antenna interface directly into the L2 cache of the node where they will be processed using the cache-inject mechanism.

For the blocks shown in Fig. 4, we will focus first on those for which performance projections were provided in Section 4, namely the FFT and the MIMO frequency-domain equalizer (MIMO decode).

For the 4x4 MIMO configuration, 28 symbols are received altogether in every 500μsec slot, and each of these is processed by a 2K-point FFT. The projection of 2500 cycles for a 2K-point FFT is uplifted by 20% to account for data movement and scaling. At 2.3GHz, the aggregate time for executing the 28 2K-point FFTs on a single A2+ with VBA is under 40μsec, or less than 10% of the slot time.

For the 4x4 MIMO configuration, using the common MMSE approach for the MIMO decode block, the number of 4x4 matrix inversions per slot is equal to the number of data subcarriers per slot; this is 1200×6 , or 7200. In addition, there may be 1200 inversions per slot for channel estimation. Using the projection from Section 4.2, this will take about 140μsec on a single A2+ with VBA, or about 28% of the slot time.

The focus here has been on two specific algorithms out of many that are employed in the LTE digital baseband: the FFT because it is used so heavily, and matrix inversion because, except for turbo decoder, it is perhaps the most significant consumer of processor cycles. In addition, there are FFT instances not counted in the above analysis (e.g. to complete the SC-FDMA processing) in the uplink receive chain, as well as a 2K-point FFT per symbol in the downlink transmit chain. We have carried out a preliminary assessment of the processing load for a complete LTE digital baseband., including the turbo decoder. The results indicate that processing for one sector with one 20MHz channel and 4x4 MIMO can be supported by three A2+ cores, each with VBA.

How the aggregate processing load is optimally partitioned across the cores will depend on several factors, including the programming model and the relative timing between uplink and downlink. Discussion of the latter is beyond the scope of this paper. With respect to the programming model, key objectives include minimization of data movement, e.g. between the L2 and a VBA's register file within a node. Use of the in-line programming model discussed in Section 3.4 can provide significant benefits here.

Consider following a symbol through the blocks shown in Fig. 4. It has already been observed that the FFT is implemented with the data array maintained in the VBA's register file throughout. In fact, once the symbol is loaded at the outset, the first three blocks, namely CP removal, FFT, and subcarrier demapping, can be implemented with the received samples remaining in the VSRF as they are processed, and with amplitudes of the 1200 used subcarriers stored after the demapping step. The in-line programming model is also easily applied to the FFTs that complete the SC-FDMA processing (following the MIMO decode block) through the demodulation step to the turbo decoder input. How to maintain blocks of data in the VSRF through the channel estimation and MIMO decode blocks will have a strong interaction with how the aggregate processing load is partitioned across multiple cores, since these blocks look at the signals from all four receive antennas.

In fact, there are a number of viable alternatives for partitioning the work, taking into account the different processing requirements in the uplink and downlink. For any of these, there will almost certainly be at least one point, and perhaps two, in the uplink receive chain where processing moves from one core to another. It can be assumed that the three cores supporting the aggregate processing load will be

in the same node sharing the same L2, so the necessary inter-thread synchronization can be efficient and relatively straightforward, as described above in Section 2.

Finally, we note that extension to support multiple sectors and more than one 20MHz channel per sector can be straightforward, given the multicore parallelism available on the enhanced PowerEN platform under consideration. For example, channels from different sectors can be assigned to different nodes. Since the antenna interface can communicate directly with the L2 for the appropriate node and the associated data can remain local to the node, the work being done for a channel in one sector sees no interference form work being done for a channel is a different sector.

6. CONCLUSION

We have presented in this paper a potential enhanced version of the IBM PowerEN chip, with extensions to support complete layer 1 processing in basestations for 3G and 4G standards. The key new element would be the augmentation of each Power processor core with an AXU providing vector-based acceleration. As examples of the use of VBA, implementations of FFT algorithms and matrix inversion were presented, as well as a key aspect of the implementation of an LTE turbo decoder. Based on the results for those algorithms and additional analysis, a preliminary assessment indicates that the complete digital baseband including the turbo decoder for one sector with one 20MHz channel supporting LTE with 4x4 MIMO can be implemented with 3 A2+ cores, each with VBA Details of the turbo decoder implementation, as well as implementation using VBA of a generic despreading algorithm for WCDMA, are presented in [3]. Studies are currently underway to estimate the number of A2+ cores with VBA that might be required to support complete basestation function for 3G including HSPA+ and 4G through LTE-Advanced. Our aggregate results point to the feasibility of an essentially general-purpose computing platform supporting SDR in a fully programmable and scalable fashion at the highest levels of performance required for basestations through LTE-Advanced and including high-throughput 3G configurations.

7. REFERENCES

[1] J. D. Brown, S. Woodward, B. M. Bass, and C. L. Johnson, "IBM Power edge-of-network processor: a wire-speed system on a chip", *IEEE Micro,* vol. 31 no. 2, pp. 76-85, March/April 2011.

[2] C. Johnson et al., "A wire-speed Power^{TM} processor: 2.3GHz 45nm SOI with 16 cores and 64 threads", in *Proc. ISSCC 2010,* San Francisco, CA, pp.104-106, Feb. 2010.

[3] A. J. Vega et al., "Processor and platform architecture for software-defined radio using vector-based acceleration in the IBM PowerEN^{TM}", submitted to ICC 2012.

[4] IBM Corporation, *PowerISA^{TM} Version 2.06 Rev. B.* available from power.org, 23 July 2010.

[5] J. H. Moreno et al., "An innovative low-power high-performance programmable signal processor for digital communications", *IBM J. Res. Devel.* vol. 47 no. 2/3, pp. 299-326, March/May 2003.

[6] J. H. Derby and J. H. Moreno, "A high-performance embedded DSP core with novel SIMD features", in *Proc. ICASSP 2003,* Hong Kong, April 2003.

[7] J. H. Derby, R. K. Montoye, and J. Moreira, "VICTORIA - VMX indirect compute technology oriented towards in-line acceleration", in *Proc. 3rd Conf Computing Frontiers, CF'06,* Ischia, Italy, pp. 303-311, May 2006.

[8] D. Nuzman, M. Namolaru, A. Zaks, and J. H. Derby, "Compiling for an indirect vector register architecture", in *Proc. 2008 Conf. Computing Frontiers, CF'08,* Ischia, Italy, pp. 199-208, May 2008.

[9] M. Namolaru et al., "The efficient use of plentiful indirect vector registers", submitted to HiPEAC 2012.

[10] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", in *Proc. ICC'95,* Seattle, WA, pp. 1009-1013, June 1995.

[11] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Trans Info. Th.* vol. 20 no. 2, pp. 284-287, 1974.