Application-Specific Accelerators for Communications

Yang Sun, Kiarash Amiri, Michael Brogioli, and Joseph R. Cavallaro

Abstract For computation-intensive digital signal processing algorithms, complexity is exceeding the processing capabilities of general-purpose digital signal processors (DSPs). In some of these applications, DSP hardware accelerators have been widely used to off-load a variety of algorithms from the main DSP host, including FFT, FIR/IIR filters, multiple-input multiple-output (MIMO) detectors, and error correction codes (Viterbi, Turbo, LDPC) decoders. Given power and cost considerations, simply implementing these computationally complex parallel algorithms with high-speed general-purpose DSP processor is not very efficient. However, not all DSP algorithms are appropriate for off-loading to a hardware accelerator. First, these algorithms should have data-parallel computations and repeated operations that are amenable to hardware implementation. Second, these algorithms should have a deterministic dataflow graph that maps to parallel datapaths. The accelerators that we consider are mostly coarse grain to better deal with streaming data transfer for achieving both high performance and low power. In this chapter, we focus on some of the basic and advanced digital signal processing algorithms for communications and cover major examples of DSP accelerators for communications.

Yang Sun
Rice University, 6100 Main St., Houston, TX 77005, e-mail: ysun@rice.edu

Kiarash Amiri
Rice University, 6100 Main St., Houston, TX 77005, e-mail: kiaa@rice.edu

Michael Brogioli
Freescale Semiconductor Inc., 7700 W. Parmer Lane, MD: PL63, Austin TX 78729,
e-mail: michael.brogioli@freescale.com

Joseph R. Cavallaro
Rice University, 6100 Main St., Houston, TX 77005, e-mail: cavallar@rice.edu

S.S. Bhattacharyya et al. (eds.), *Handbook of Signal Processing Systems*,
DOI 10.1007/978-1-4419-6345-1_13,

1 Introduction

In third-generation (3G) wireless systems, the signal processing algorithm complexity has begun to exceed the processing capabilities of general purpose digital signal processors (DSPs). With the inclusion of multiple-input multiple-output (MIMO) technology in the fourth-generation (4G) wireless system, the DSP algorithm complexity has far exceeded the processing capabilities of DSPs. Given area and power constraints for the mobile handsets one can not simply implement computation intensive DSP algorithms with gigahertz DSPs. Besides, it is also critical to reduce base station power consumption by utilizing optimized hardware accelerator design. Fortunately, only a few DSP algorithms dominate the main computational complexity in a wireless receiver. These algorithms, including Viterbi decoding, Turbo decoding, LDPC decoding, MIMO detection, and channel equalization/FFT, need to be off-loaded to hardware coprocessors or accelerators, yielding high performance. These hardware accelerators are often integrated in the same die with DSP processors. In addition, it is also possible to leverage the field-programmable gate array (FPGA) to provide reconfigurable massive computation capabilities.

DSP workloads are typically numerically intensive with large amounts of both instruction and data level parallelism. In order to exploit this parallelism with a programmable processor, most DSP architectures utilize Very Long Instruction Word, or VLIW architectures. VLIW architectures typically include one or more register files on the processor die, versus a single monolithic register file as is often the case in general purpose computing. Examples of such architectures are the Freescale StarCore processor, the Texas Instruments TMS320C6x series DSPs as well as SHARC DSPs from Analog Devices, to name a few [14, 28, 39].

In some cases due to the idiosyncratic nature of many DSPs, and the implementation of some of the more powerful instructions in the DSP core, an optimizing compiler can not always target core functionality in an optimal manner. Examples of this include high performance fractional arithmetic instructions, for example, which may perform highly SIMD functionality which the compiler can not always deem safe at compile time.

While the aforementioned VLIW based DSP architectures provide increased parallelism and higher numerical throughput performance, this comes at a cost of ease in programmability. Typically such machines are dependent on advanced optimizing compilers that are capable of aggressively analyzing the instruction and data level parallelism in the target workloads, and mapping it onto the parallel hardware. Due to the large number of parallel functional units and deep pipeline depths, modern DSP are often difficult to hand program at the assembly level while achieving optimal results. As such, one technique used by the optimizing compiler is to vectorize much of the data level parallelism often found in DSP workloads. In doing this, the compiler can often fully exploit the single instruction multiple data, or SIMD functionality found in modern DSP instruction sets.

Despite such highly parallel programmable processor cores and advanced compiler technology, however, it is quite often the case that the amount of available instruction and data level parallelism in modern signal processing workloads far

exceeds the limited resources available in a VLIW based programmable processor core. For example, the implementation complexity for a 40 Kbps DS-CDMA system would be 41.8 Gflops/s for 60 users [50], not to mention 100 Mbps+ 3GPP LTE system. This complexity largely exceeds the capability of nowadays DSP processors which typically can provide 1-5 Gflops performance, such as 1.5 Gflops TI C6711 DSP processor and 1.8 Gflops ADI TigerSHARC Processor. In other cases, the functionality required by the workload is not efficiently supported by more general purpose instruction sets typically found in embedded systems. As such the need for acceleration at both the fine grain and coarse grain levels is often required, the former for instruction set architecture (ISA) like optimization and the latter for task like optimization.

Additionally, wireless system designers often desire the programmability offered by software running on a DSP core versus a hardware based accelerator, to allow flexibility in various proprietary algorithms. Examples of this can be functionality such as channel estimation in baseband processing, for which a given vendor may want to use their own algorithm to handle various users in varying system conditions versus a pre-packaged solution. Typically these demands result in a heterogeneous system which may include one or more of the following: software programmable DSP cores for data processing, hardware based accelerator engines for data processing, and in some instances general purpose processors or micro-controller type solutions for control processing.

The motivations for heterogeneous DSP system solutions including hardware acceleration stem from the tradeoffs between software programmability versus the performance gains of custom hardware acceleration in its various forms. There are a number of heterogenous accelerator based architectures currently available today, as well as various offerings and design solutions being offered by the research community.

There are a number of DSP architectures which include true hardware based accelerators which are not programmable by the end user. Examples of this include the Texas Instruments' C55x and C64x series of DSPs which include hardware based Viterbi or Turbo decoder accelerators for acceleration of wireless channel decoding [26, 27].

1.1 Coarse Grain Versus Fine Grain Accelerator Architectures

Coarse–grain accelerator based DSP systems entail a co–processor type design whereby larger amounts of work are run on the sometimes configurable co–processor device. Current technologies being offered in this area support offloading of functionality such as FFT and various matrix-like computations to the accelerator versus executing in software on the programmable DSP core.

As shown in Figure 1, coarse grained heterogeneous architectures typically include a loosely coupled computational grid attached to the host processor. These types of architectures are sometimes built using an FPGA, ASIC, or vendor pro-

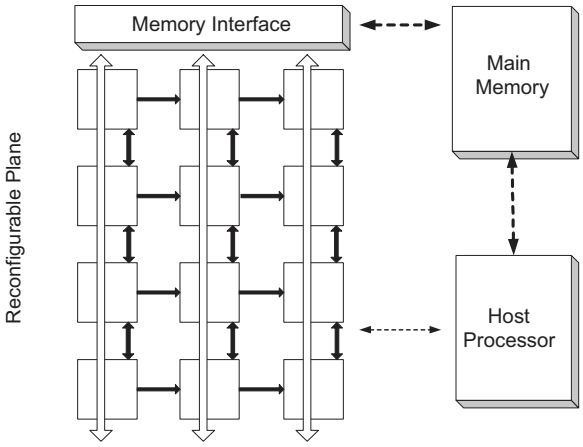

Fig. 1 Traditional coarse grained accelerator architecture. [8]

grammable acceleration engine for portions of the system. Tightly coupled loop nests or kernels are then offloaded from executing in software on the host processor to executing in hardware on the loosely coupled grid.

Fine–grain accelerator based architectures are the flip-side to the coarse grained accelerator mindset. Typically, ISAs provide primitives that allow low cost, low complexity implementations while still maintaining high performance for a broad range of input applications. In certain cases, however, it is often advantageous to offer instructions specialized to the computational needs of the application. Adding new instructions to the ISA, however, is a difficult decision to make. On the one hand they may provide significant performance increases for certain subsets of applications, but they must still be general enough such that they are useful across a much wider range of applications. Additionally, such instructions may become obsolete as software evolves and may complicate future hardware implementations of the ISA [53]. Vendors such as Tensilica, however, offer toolsets to produce configurable extensible processor architectures typically targeted at the embedded community [25]. These types of products typically allow the user to configure a predefined subset of processor components to fit the specific demands of the input application. Figure 2 shows the layout of a typical fine grained reconfigurable architecture whereby a custom ALU is coupled with the host processors pipeline.

In summary, both fine grained acceleration and coarse grained acceleration can be beneficial to the computational demands of DSP applications. Depending on the overall design constraints of the system, designers may chose a heterogeneous coarse grained acceleration system or a strictly software programmable DSP core system.

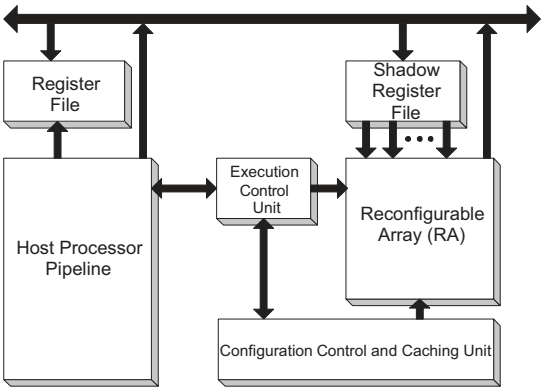

Fig. 2 Example fine grained reconfigurable architecture with customizable ALU for ISA extensions. [8]

1.2 Hardware/Software Workload Partition Criteria

In partitioning any workload across a heterogeneous system comprised of reconfigurable computational accelerators, programmable DSPs or programmable host processors, and varied memory hierarchy, a number of criteria must be evaluated in addition to application profile information to determine whether a given task should execute in software on the host processor or in hardware on FPGA or ASIC, as well where in the overall system topology each task should be mapped. It is these sets of criteria that typically mandate the software partitioning, and ultimately determine the topology and partitioning of the given system.

Spatial locality of data is one concern in partitioning a given task. In a typical software implementation running on a host processor, the ability to access data in a particular order efficiently is of great importance to performance. Issues such as latency to memory, data bus contention, data transfer times to local compute element such as accelerator local memory, as well as type and location of memory all need to be taken into consideration. In cases where data is misaligned, or not contiguous or uniformly strided in memory, additional overhead may be needed to arrange data before block DMA transfers can take place or data can efficiently be computed on. In cases where data is not aligned properly in memory, significant performance degradations can be seen due to decreased memory bandwidth when performing unaligned memory accesses on some architectures. When data is not uniformly strided, it may be difficult to burst transfer even single dimensional strips of memory via DMA engines. Consequently, with non-uniformly strided data it may be necessary to perform data transfers into local accelerator memory for computation via programmed I/O on the part of the host DSP. Inefficiencies in such methods of data transfer can easily overshadow any computational benefits achieved by compute acceleration of the FPGA. The finer the granularity of computation offloaded

for acceleration in terms of compute time, quite often the more pronounced the side effects of data memory transfer to local accelerator memory.

Data level parallelism is another important criteria in determining the partitioning for a given application. Many applications targeted at VLIW-like architectures, especially signal processing applications, exhibit a large amount of both instruction and data level parallelism [24]. Many signal processing applications often contain enough data level parallelism to exceed the available functional units of a given architecture. FPGA fabrics and highly parallel ASIC implementations can exploit these computational bottlenecks in the input application by providing not only large numbers of functional units but also large amounts of local block data RAM to support very high levels of instruction and data parallelism, far beyond that of what a typical VLIW signal processing architecture can afford in terms of register file real estate. Furthermore, depending on the instruction set architecture of the host processor or DSP, performing sub-word or multiword operations may not be feasible given the host machine architecture. Most modern DSP architecures have fairly robust instruction sets that support fine grained multiword SIMD acceleration to a certain extent. It is often challenging, however, to efficiently load data from memory into the register files of a programmable SIMD style processor to be able to efficiently or optimally utilize the SIMD ISA in some cases.

Computational complexity of the application often bounds the programmable DSP core, creating a compute bottleneck in the system. Algorithms that are implemented in FPGA are often computationally intensive, exploiting greater amounts of instruction and data level parallelism than the host processor can afford, given the functional unit limitations and pipeline depth. By mapping computationally intense bottlenecks in the application from software implementation executing on host processor to hardware implementation in FPGA, one can effectively alleviate bottlenecks on the host processor and permit extra cycles for additional computation or algorithms to execute in parallel.

Task level parallelism in a portion of the application can play a role in the ideal partitioning as well. Quite often, embedded applications contain multiple tasks that can execute concurrently, but have a limited amount of instruction or data level parallelism within each unique task [51]. Applications in the networking space, and baseband processing at layers above the data plane typically need to deal with processing packets and traversing packet headers, data descriptors and multiple task queues. If the given task contains enough instruction and data level parallelism to exhaust the available host processor compute resources, it can be considered for partitioning to an accelerator. In many cases, it is possible to concurrently execute multiple of these tasks in parallel, either across multiple host processors or across both host processor and FPGA compute engine depending on data access patterns and cross task data dependencies. There are a number of architectures which have accelerated tasks in the control plane, versus data plane, in hardware. One example of this is the Freescale Semiconductor QorIQ platform which provides hardware acceleration for frame managers, queue managers and buffer managers. In doing this, the architecture effectively frees the programmable processor cores from dealing with control plane management.

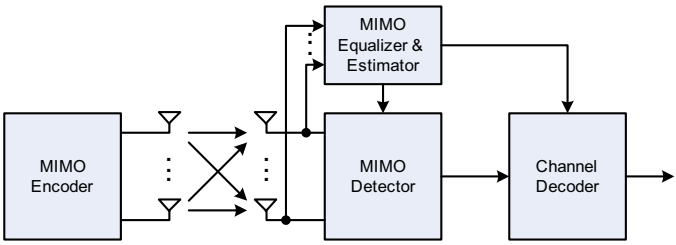

Fig. 3 Basic structure of an MIMO receiver.

2 Hardware Accelerators for Communications

Processors in 3G and 4G cellular systems typically require high speed, throughput, and flexibility. In addition to this, computationally intensive algorithms are used to remove often high levels of multiuser interference especially in the presence of multiple transmit and receive antenna MIMO systems. Time varying wireless channel environments can also dramatically deteriorate the performance of the transmission, further requiring powerful channel equalization, detection, and decoding algorithms for different fading conditions at the mobile handset. In these types of environments, it is often the case that the amount of available parallel computation in a given application or kernel far exceeds the available functional units in the target processor. Even with modern VLIW style DSPs, the number of available functional units in a given clock cycle is limited and prevents full parallelization of the application for maximum performance. Further, the area and power constraints of mobile handsets make a software-only solution difficult to realize.

Figure 3 depicts a typical MIMO receiver model. Three major blocks, MIMO channel estimator & equalizer, MIMO detector, and channel decoder, determine the computation requirements of a MIMO receiver. Thus, it is natural to offload these very computational intensive tasks to hardware accelerators to support high data rate applications. Example of these applications include 3GPP LTE with 326 Mbps downlink peak data rate and IEEE 802.16e WiMax with 144 Mbps downlink peak data rate. Further, future standards such as LTE-Advance and WiMax-m are targeting over 1 Gbps speeds.

Data throughput is an important metric to consider when implementing a wireless receive. Table 1 summaries the data throughput performance for different MIMO wireless technologies as of 2009. Given the current DSP processing capabilities, it is very necessary to develop application-specific hardware accelerators for the complex MIMO algorithms.

Table 1 Throughput performance of different MIMO systems

	HSDPA+ (2×2 MIMO)	LTE (2×2 MIMO)	LTE (4×4 MIMO)	WiMax Rel 1.5 (2×2 MIMO)	WiMax Rel 1.5 (4×4 MIMO)
Downlink	42 Mbps	173 Mbps	326 Mbps	144 Mbps	289 Mbps
Uplink	11.5 Mbps	58 Mbps	86 Mbps	69 Mbps	69 Mbps

2.1 MIMO Channel Equalization Accelerator

The total workload for a given channel equalizer performed as a baseband processing part on the mobile receiver can be decomposed into multiple tasks as depicted in Figure 4. This block diagram shows the various software processing blocks, or kernels, that make up the channel equalizer firmware executing on the digital signal processor of the mobile receiver. The tasks are: channel estimation based on known pilot sequence, covariance computation (first row or column) and circularization, FFT/IFFT (Fast Fourier transform and Inverse Fast Fourier transform) post-processing for updating equalization coefficients, finite–impulse response (FIR) filtering applied on the received samples (received frame), and user detection (despreading–descrambling) for recovering the user information bits. The computed data is shared between the various tasks in a pipeline fashion, in that the output of covariance computation is used as the input to the matrix circularization algorithm.

The computational complexity of various components of the workload vary with the number of users in the system, as well as users entering and leaving the cell as well as channel conditions. Regardless of this variance in the system conditions at runtime, the dominant portions of the workload are the channel estimation, fast

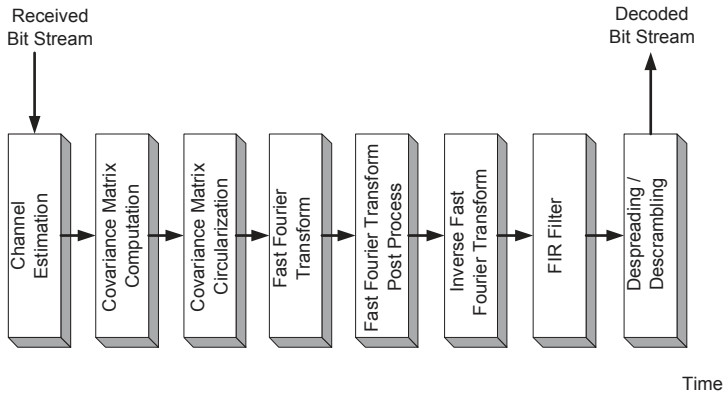

Fig. 4 Workload partition for a channel equalizer.

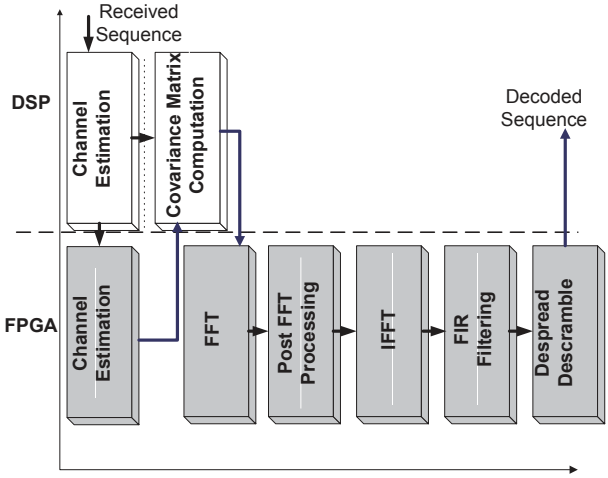

Fig. 5 Channel equalizer DSP/hardware accelerator partitioning.

Fourier transform, inverse fast Fourier transform and FIR filtering as well as despreading and descrambling.

As an example, using the workload partition criteria for partitioning functionality between a programmable DSP core and system containing multiple hardware for a 3.5G HSDPA system, it has been shown that impressive performance results can be obtained. In studying the bottlenecks of such systems when implemented on a programmable DSP core in software, it has been found the key bottlenecks in the system to be the channel estimation, fast fourier transform (FFT), inverse fast fourier transform (IFFT), FIR filter, and to a lesser extent despreading and descrambling as illustrated in Figure 4 [9]. By migrating the 3.5G implementation from a solely software based implementation executing on a TMS320C64x based programmable DSP core to a heterogeneous system containing not only programmable DSP cores but also distinct hardware acceleration for the various bottlenecks, the authors achieve almost an 11.2x speedup in the system [9]. Figure 5 illustrates the system partitioning between programmable DSP core and hardware (e.g. FPGA or ASIC) accelerator that resulted in load balancing the aforementioned bottlenecks.

The arrows in the diagram illustrate the data flow between local programmable DSP core on-chip data caches and the the local RAM arrays. In the case of channel estimation, the work is performed in parallel between the programmable DSP core and hardware acceleration. Various other portions of the workload are offloaded to hardware based accelerators while the programmable DSP core performs the lighter weight signal processing code and book keeping.

Despite the ability to achieve over 11x speedup in performance, it is important to note that the experimental setup used in these studies was purposely pessimistic. The various FFT, IFFT, etc compute blocks in these studies were offloaded to discrete FPGA / ASIC accelerators. As such, data had to be transferred, for example,

Fig. 6 MIMO transmitter and receiver

from local IFFT RAM cells to FIR filter RAM cells. This is pessimistic in terms of data communication time. In most cases the number of gates required for a given accelerator implemented in FPGA/ASIC was low enough that multiple accelerators could be implemented within a single FPGA/ASIC drastically reducing chip-to-chip communication time.

2.2 MIMO Detection Accelerators

MIMO systems, Figure 6, have been shown to be able to greatly increase the reliability and data rate for point-to-point wireless communication [47]. Multiple-antenna systems can be used to improve the reliability and diversity in the receiver by providing the receiver with multiple copies of the transmitted information. This diversity gain is obtained by employing different kinds of space-time block code (STBC) [1, 45, 46]. In such cases, for a system with M transmit antennas and N receive antennas and over a time span of T time symbols, the system can be modeled as

$$\mathbf{Y} = \mathbf{HX} + \mathbf{N}, \tag{1}$$

where $\mathbf{H}$ is the $N \times M$ channel matrix. Moreover, $\mathbf{X}$ is the $M \times T$ space-time code matrix where its x_{ij} element is chosen from a complex-valued constellation Ω of the order $w = |\Omega|$ and corresponds to the complex symbol transmitted from the i-th antenna at the j-th time. The $\mathbf{Y}$ matrix is the received $N \times T$ matrix where y_{ij} is the perturbed received element at the i-th receive antenna at the j-th time. Finally, $\mathbf{N}$ is the additive white Gaussian noise matrix on the receive antennas at different time slots.

MIMO systems could also be used to further expand the transmit data rate using other space-time coding techniques, particularly layered space-time (LST) codes [17]. One of the most prominent examples of such space-time codes is Vertical Bell Laboratories Layered Space-Time (V-BLAST) [20], otherwise known as spatial multiplexing (SM). In the spatial multiplexing scheme, independent symbols are transmitted from different antennas at different time slots; hence, supporting even higher data rates compared to space-time block codes of lower data rate [1, 45]. The spatial multiplexing MIMO system can be modeled similar to Eq. (1) with $T = 1$ since there is no coding across the time domain:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \tag{2}$$

where $\mathbf{H}$ is the $N \times M$ channel matrix, $\mathbf{x}$ is the M-element column vector where its x_i-th element corresponds to the complex symbol transmitted from the i-th antenna, and $\mathbf{y}$ is the received N-th element column vector where y_i is the perturbed received element at the i-th receive antenna. The additive white Gaussian noise vector on the receive antennas is denoted by $\mathbf{n}$.

While spatial multiplexing can support very high data rates, the complexity of the maximum-likelihood detector in the receiver increases exponentially with the number of transmit antennas. Thus, unlike the case in Eq. (1), the maximum-likelihood detector for Eq. (2) requires a complex architecture and can be very costly. In order to address this challenge, a range of detectors and solutions have been studied and implemented. In this section, we discuss some of the main algorithmic and architectural features of such detectors for spatial multiplexing MIMO systems.

2.2.1 Maximum-Likelihood (ML) Detection

The Maximum Likelihood (ML) or optimal detection of MIMO signals is known to be an NP-complete problem. The maximum-likelihood (ML) detector for Eq. (2) is found by minimizing the

$$\left|\left|\mathbf{y} - \mathbf{H}\mathbf{x}\right|\right|_2^2 \tag{3}$$

norm over all the possible choices of $\mathbf{x} \in \Omega^M$. This brute-force search can be a very complicated task, and as already discussed, incurs an exponential complexity in the number of antennas, in fact for M transmit antennas and modulation order of $w = |\Omega|$, the number of possible $\mathbf{x}$ vectors is w^M. Thus, unless for small dimension problems, it would be infeasible to implement it within a reasonable area-time constraint [10, 19].

2.2.2 Sphere Detection

Sphere detection can be used to achieve ML (or close-to-ML) with reduced complexity [15, 23] compared to ML. In fact, while the norm minimization of Eq. (3) is exponential complexity, it has been shown that using the sphere detection method, the ML solution can be obtained with much lower complexity [15].

In order to avoid the significant overhead of the ML detection, the distance norm can be simplified [13] as follows:

$$\begin{aligned} D(\mathbf{s}) &= \| \mathbf{y} - \mathbf{H}\mathbf{s} \|^2 \\ &= \| \mathbf{Q}^{\mathbf{H}}\mathbf{y} - \mathbf{R}\mathbf{s} \|^2 = \sum_{i=M}^{1} |y_i{}' - \sum_{j=i}^{M} R_{i,j} s_j|^2, \end{aligned} \tag{4}$$

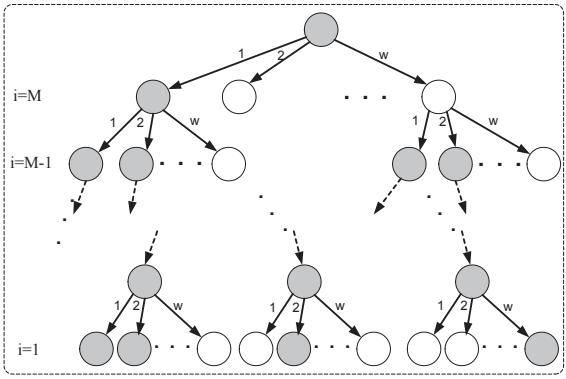

Fig. 7 Calculating the distances using a tree. Partial norms, *PN*s, of dark nodes are less than the threshold. White nodes are pruned out.

where $\mathbf{H} = \mathbf{QR}$ represents the channel matrix QR decomposition, $\mathbf{R}$ is an upper triangular matrix, $\mathbf{QQ^H} = \mathbf{I}$ and $\mathbf{y}' = \mathbf{Q^H y}$.

The norm in Eq. (4) can be computed in M iterations starting with $i = M$. When $i = M$, i.e. the first iteration, the initial partial norm is set to zero, $T_{M+1}(\mathbf{s}^{(M+1)}) = 0$. Using the notation of [11], at each iteration the Partial Euclidean Distances (PEDs) at the next levels are given by

$$T_i(\mathbf{s}^{(i)}) = T_{i+1}(\mathbf{s}^{(i+1)}) + |e_i(\mathbf{s}^{(i)})|^2 \tag{5}$$

with $\mathbf{s}^{(i)} = [s_i, s_{i+1}, ..., s_M]^T$, and $i = M, M-1, ..., 1$, where

$$|e_i(\mathbf{s}^{(i)})|^2 = |{y_i}' - R_{i,i}s_i - \sum_{j=i+1}^{M} R_{i,j}s_j|^2. \tag{6}$$

One can envision this iterative algorithm as a tree traversal with each level of the tree corresponding to one i value or transmit antenna, and each node having w' children based on the modulation chosen.

The norm in Eq. (6) can be computed in M iterations starting with $i = M$, where M is the number of transmit antennas. At each iteration, partial (Euclidian) distances, $PD_i = |{y_i}' - \sum_{j=i}^{M} R_{i,j}s_j|^2$ corresponding to the i-th level, are calculated and added to the partial norm of the respective parent node in the $(i-1)$-th level, $PN_i = PN_{i-1} + PD_i$. When $i = M$, i.e. the first iteration, the initial partial norm is set to zero, $PN_{M+1} = 0$. Finishing the iterations gives the final value of the norm. As shown in Figure 7, one can envision this iterative algorithm as a tree traversal problem where each level of the tree represents one i value, each node has its own PN, and w children, where w is the QAM modulation size. In order to reduce the search complexity, a threshold, C, can be set to discard the nodes with $PN > C$. Therefore, whenever a node k with a $PN_k > C$ is reached, any of its children will

have $PN \geq PN_k > C$. Hence, not only the k-th node, but also its children, and all nodes lying beneath the children in the tree, can be pruned out.

There are different approaches to search the entire tree, mainly classified as depth-first search (DFS) approach and K-best approach, where the latter is based on breadth-first search (BFS) strategy. In DFS, the tree is traversed vertically [2, 11]; while in BFS [22, 52], the nodes are visited horizontally, i.e. level by level.

In the DFS approach, starting from the top level, one node is selected, the *PN*s of its children are calculated, and among those new computed *PN*s, one of them, e.g. the one with the least *PN*, is chosen, and that becomes the parent node for the next iteration. The *PN*s of its children are calculated, and the same procedure continues until a leaf is reached. At this point, the value of the global threshold is updated with the *PN* of the recently visited leaf. Then, the search continues with another node at a higher level, and the search controller traverses the tree down to another leaf. If a node is reached with a *PN* larger than the radius, i.e. the global threshold, then that node, along with all nodes lying beneath that, are pruned out, and the search continues with another node.

The tree traversal can be performed in a breadth-first manner. At each level, only the best K nodes, i.e. the K nodes with the smallest T_i, are chosen for expansion. This type of detector is generally known as the K-best detector. Note that such a detector requires sorting a list of size $K \times w'$ to find the best K candidates. For instance, for a 16-QAM system with $K = 10$, this requires sorting a list of size $K \times w' = 10 \times 4 = 40$ at most of the tree levels.

2.2.3 Computational Complexity of Sphere Detection

In this section, we derive and compare the complexity of the proposed techniques. The complexity in terms of number of arithmetic operations of a sphere detection operation is given by

$$J_{SD}(M,w) = \sum_{i=M}^{1} J_i E\{D_i\}, \tag{7}$$

where J_i is the number of operations per node in the i-th level. In order to compute J_i, we refer to the VLSI implementation of [11], and note that, for each node, one needs to compute the $R_{i,j}s_j$, multiplications, where, except for the diagonal element, $R_{i,i}$, the rest of the multiplications are complex valued. The expansion procedure, Eq. (4), requires computing $R_{i,j}s_j$ for $j = i+1,...,M$, which would require $(M-i)$ complex multiplications, and also computing $R_{i,i}s_i$ for all the possible choices of $s_j \in \Omega$. Even though, there are w different s_js, there are only $(\frac{\sqrt{w}}{2} - 1)$ different multiplications required for QAM modulations. For instance, for a 16-QAM with $\{\pm 3 \pm 3j, \pm 1 \pm 1j, \pm 3 \pm 1j, \pm 1 \pm 3j\}$, computing only $(R_{i,j} \times 3)$ would be sufficient for all the choices of modulation points. Finally, computing the $\| \, . \, \|^2$ requires a squarer or a multiplier, depending on the architecture and hardware availabilities.

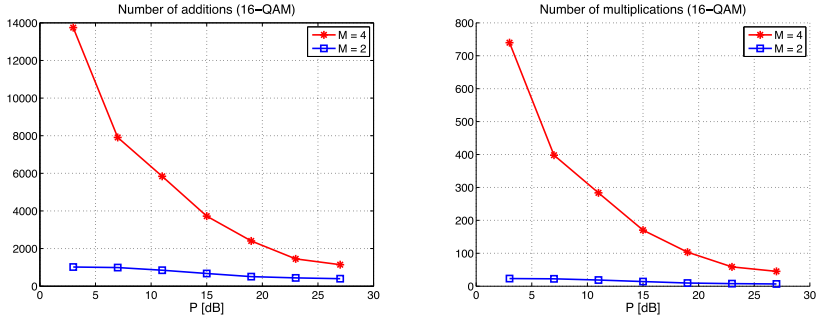

Fig. 8 Number of addition and multiplications operations for 16-QAM with different number of antennas, M.

In order to compute the number of adders for each norm expansion in (4), we note that there are $(M-i)$ complex valued adders required for ${y_i}' - \sum_{j=i+1}^{M} R_{i,j}s_j$, and w more complex adders to add the newly computed $R_{i,i}s_i$ values. Once the w different norms, $\left|{y_i}' - \sum_{j=i}^{M} R_{i,j}s_j\right|^2$, are computed, they need to be added to the partial distance coming from the higher level, which requires w more addition procedures. Finally, unless the search is happening at the end of the tree, the norms need to be sorted, which assuming a simple sorter, requires $w(w+1)/2$ compare-select operations.

Therefore, keeping in mind that each complex multiplier corresponds to four real-valued multipliers and two real-valued adders, and that every complex adder corresponds to two real-valued adders, J_i is calculated by

$$\begin{aligned} J_i(M,w) &= J_{mult} + J_{add}(M,w) \\ J_{mult}(M,w) &= ((\frac{\sqrt{w}}{2} - 1) + 4(M-i) + 1) \\ J_{add}(M,w) &= (2(M-i) + 2w + w) + (w(w+1)/2) \cdot sign(i-1), \end{aligned}$$

where $sign(i-1)$ is used to ensure sorting is counted only when the search has not reached the end of the tree, and is equal to:

$$sign(t) = \begin{cases} 1 & t \geq 1 \\ 0 & otherwise \end{cases}. \tag{8}$$

Moreover, we use θ, β and γ to represent the hardware-oriented costs for one adder, one compare-select unit and one multiplication operation, respectively.

Figure 8 shows the number of addition and multiplication operations needed for a 16-QAM system with different number of antennas.

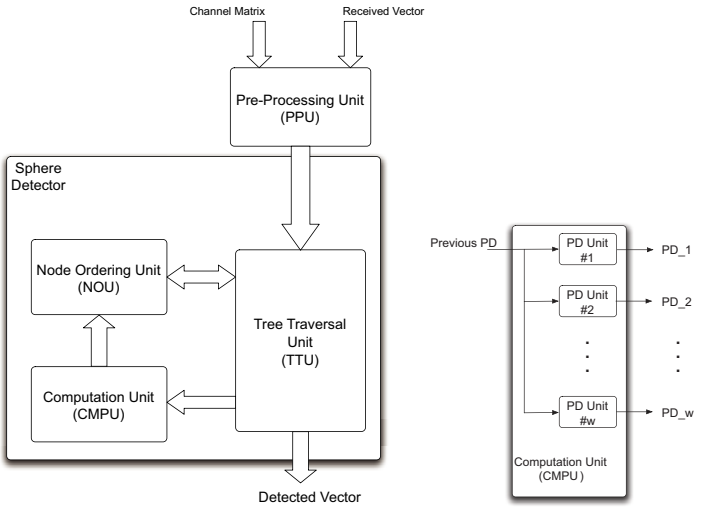

Fig. 9 Sphere Detector architecture with multiple PED function units.

2.2.4 Depth-First Sphere Detector Architecture

The depth-first sphere detection algorithm [11, 15, 19, 23] traverses the tree in a depth-first manner: the detector visits the children of each node before visiting its siblings. A constraint, referred to as radius, is often set on the PED for each level of the tree. A generic depth-first sphere detector architecture is shown in Figure 9. The Pre-Processing Unit (PPU) is used to compute the QR decomposition of the channel matrix as well as calculate $\mathbf{Q}^H\mathbf{y}$. The Tree Traversal Unit (TTU) is the controlling unit which decides in which direction and with which node to continue. Computation Unit (CMPU) computes the partial distances, based on (4), for w different s_j. Each PD unit computes $|y_i' - \sum_{j=i}^{M} R_{i,j}s_j|^2$ for each of the w children of a node. Finally, the Node Ordering Unit (NOU) is for finding the minimum and saving other legitimate candidates, i.e. those inside R_i, in the memory.

As an example to show the algorithm complexity, an FPGA implementation synthesis result for a 50 Mbps 4×4 16-QAM depth-first sphere detector is summarized in Table 2 [2].

Table 2 FPGA Resource Utilization for Sphere Detector

Device	Xilinx Virtex-4 xc4vfx100-12ff1517
Number of Slices	4065/42176 (9%)
Number of FFs	3344/84352 (3%)
Number of Look-Up Tables	6457/84352 (7%)
Number of RAMB16	3/376 (1%)
Number of DSP48s	32/160 (20%)
Max. Freq.	125.7 MHz

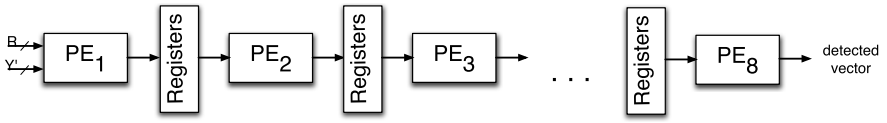

Fig. 10 The K-best MIMO detector architecture. The intermediate register banks contain the sorting information as well as the other values, i.e. $\mathbf{R}$ matrix.

2.2.5 K-Best Detector Architecture

K-best is another popular algorithm for implementing close-to-ML MIMO detection [22, 52]. The performance of this scheme is suboptimal compared to ML and sphere detection. However, it has a fixed complexity and relatively straightforward architecture. In this section, we briefly introduce the architecture [22] to implement the K-best MIMO detector. As illustrated in Figure 10, the PE elements at each stage compute the euclidean norms of (6), and find the best K candidates, i.e. the K candidates with the smallest norms, and pass them as the surviving candidates to the next level. It should be pointed out that the equation (2) can be decomposed into separate real and imaginary parts [22], which would double the size of the matrices. While such decomposition reduces the complex-valued operations of nodes into real-valued operations, it doubles the number of levels of the tree. Therefore, as shown in Figure 10, there are 8 K-best detection levels for the 4-antenna system. By selecting the proper K value, the real-value decomposition MIMO detection will not cause performance degradation compared to the complex-value MIMO detection [32].

In summary, both depth-first and K-best detectors have a regular and parallel data flow that can be efficiently mapped to hardware. The large amount of required multiplications makes the algorithm very difficult to be realized in a DSP processor. As the main task of the MIMO detector is to search for the best candidate in a very short time period, it would be more efficient to be mapped on a parallel hardware searcher with multiple processing elements. Thus, to sustain the high throughput MIMO detection, an MIMO hardware accelerator is necessary.

2.3 Channel Decoding Accelerators

Error correcting codes are widely used in digital transmission, especially in wireless communications, to combat the harsh wireless transmission medium. To achieve high throughput, researchers are investigating more and more advanced error correction codes. The most commonly used error correcting codes in modern systems are convolutional codes, Turbo codes, and low-density parity-check (LDPC) codes. As a core technology in wireless communications, FEC (forward error correction) coding has migrated from the basic 2G convolutional/block codes to more powerful 3G Turbo codes, and LDPC codes forecast for 4G systems.

As codes become more complicated, the implementation complexity, especially the decoder complexity, increases dramatically which largely exceeds the capability of the general purpose DSP processor. Even the most capable DSPs today would need some types of acceleration coprocessor to offload the computation-intensive error correcting tasks. Moreover, it would be much more efficient to implement these decoding algorithms on dedicated hardware because typical error correction algorithms use special arithmetic and therefore are more suitable for ASICs or FPGAs. Bitwise operations, linear feedback shift registers, and complex look-up tables can be very efficiently realized with ASICs/FPGAs.

In this section, we will present some important error correction algorithms and their efficient hardware architectures. We will cover major error correction codes used in the current and next generation communication standards, such as 3GPP LTE, IEEE 802.11n Wireless LAN, IEEE 802.16e WiMax, and etc.

2.3.1 Viterbi Decoder Accelerator Architecture

In telecommunications, convolutional codes are among the most popular error correction codes that are used to improve the performance of wireless links. For example, convolutional codes are used in the data channel of the second generation (2G) mobile phone system (eg. GSM) and IEEE 802.11a/n wireless local area network (WLAN). Due to their good performance and efficient hardware architectures, convolutional codes continue to be used by the 3G/4G wireless systems for their control channels, such as 3GPP LTE and IEEE 802.16e WiMax.

A convolutional code is a type of error-correcting code in which each m-bit information symbol is transformed into an n-bit symbol, where m/n is called the code rate. The encoder is basically a finite state machine, where the state is defined as the contents of the memory of the encoder. Figure 11 (a) and (b) show two examples of convolutional codes with constraint length $K = 3$, code rate $R = 1/3$ and constraint length $K = 7$, code rate $R = 1/2$, respectively.

The Viterbi algorithm is an optimal decoding algorithm for the decoding of convolutional codes [16, 49]. The Viterbi algorithm enumerates all the possible codewords and selects the most likely sequence. The most likely sequence is found by traversing a trellis. The trellis diagram for a $K = 3$ convolutional code (cf. Figure 11(a)) is shown in Figure 12.

In general, a Viterbi decoder contains four blocks: branch metric calculation (BMC) unit, add-compare-select (ACS) unit, survivor memory unit (SMU), and trace back (TB) unit as shown in Figure 13. The decoder works as follows. BMC calculates all the possible branch metrics from the channel inputs. ACS unit recursively calculates the state metrics and the survivors are stored into a survivor memory. The survivor paths contain state transitions to reconstruct a sequence of states by tracing back. This reconstructed sequence is then the most likely sequence sent by the transmitter. In order to reduce memory requirements and latency, Viterbi decoding can be sliced into blocks, which are often referred to as sliding windows. The sliding window principle is shown in Figure 14. The ACS recursion is carried out for the

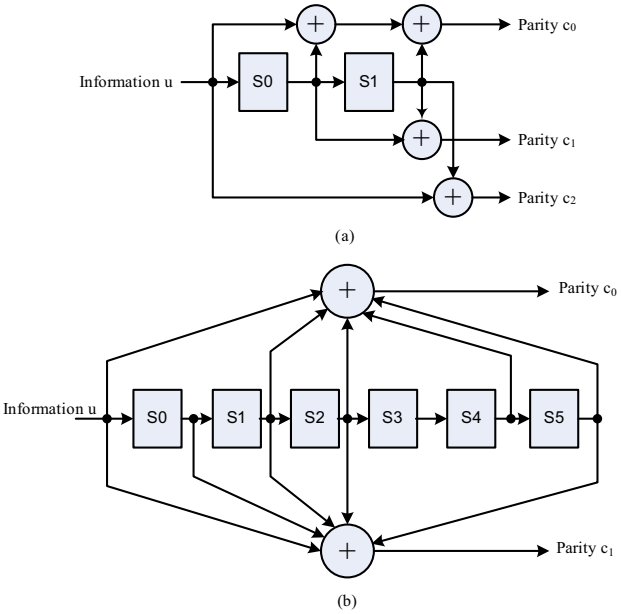

Fig. 11 Convolutional encoder. (a) K=3, R=1/3. (b) K=7, R=1/2 encoder used for WLAN.

entire code block. The trace back operation is performed on every sliding window. To improve the reliability of the trace back, the decisions for the last T (also referred to warmup window) steps will be discarded. Thus, after a fixed delay of $L+2T$, the decoder begins to produce decoded bits on every clock cycle.

Like an FFT processor [4, 12, 33], a Viterbi decoder has a very regular data structure. Thus, it is very natural to implement a Viterbi decoder in hardware to support high speed applications, such as IEEE 802.11n wireless LAN with 300 Mbps peak data rate.

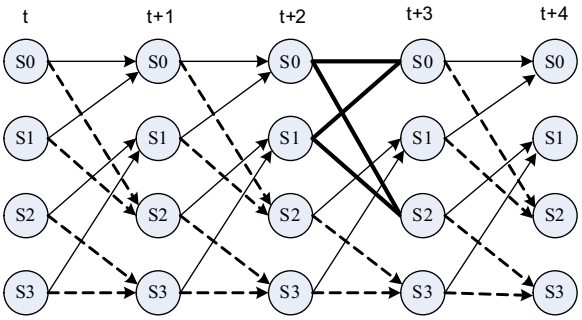

Fig. 12 A 4-state trellis diagram for the encoder in Figure 11(a). The solid lines indicate transitions for a "0" input and the dashed lines for a "1" input. Each stage of the trellis consists of 2^{K-2} Viterbi butterflies. One such butterfly is highlighted at step $t+2$.

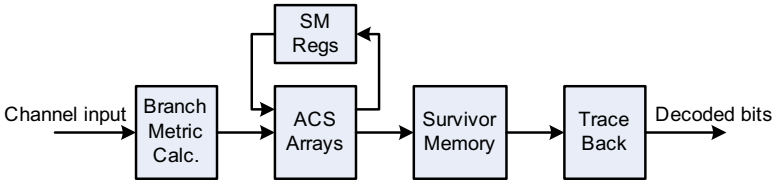

Fig. 13 Viterbi decoder architecture with parallel ACS function units.

The most complex operations in the Viterbi algorithm is the ACS recursion. Figure 15(a) shows one ACS butterfly of the 4-state trellis described above (cf. Figure 12). Each butterfly contains two ACS units. For each ACS unit, there are two branches leading from two states on the left, and going to a state on the right. A branch metric is computed for each branch of an ACS unit. Note that it is not necessary to calculate every branch metric for all 4 branches in an ACS butterfly, because some of them are identical depending on the trellis structure. Based on the old state metrics (SMs) and branch metrics (BMs), the new state metrics are updated as:

$$SM(0) = \min\Big(SM(0) + BM(0,0), SM(1) + BM(1,0)\Big) \tag{9}$$

$$SM(2) = \min\Big(SM(0) + BM(0,2), SM(1) + BM(1,2)\Big). \tag{10}$$

Given a constraint length of K convolutional code, 2^{K-2} ACS butterflies would be required for each step of the decoding. These butterflies can be implemented in serial or parallel. To maximize the decoding throughput, a parallel implementation is often used. The basic parallel ACS architecture can process one bit of the message at each clock cycle. However, the processing speed can be possibly increased by N times by merging every N stages of the trellis into one high-radix stage with 2^N branches for every state. Figure 16 shows the radix-2, radix-4, and radix-8 trellis structures. Figure 17 shows a radix-8 ACS architecture which can process three message bits over three trellis path bits. Generally for a radix-N ACS architecture, it can process $\log_2 N$ message bits over $\log_2 N$ trellis path bits. For high speed applications, high radix ACS architectures are very common in a Viterbi decoder. The top level of a generic Viterbi decoder accelerator is shown in Figure 18. Although a pure software

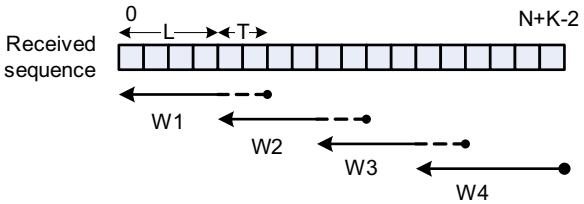

Fig. 14 Sliding window decoding with warmup. Decoding length = L. Warmup length = T.

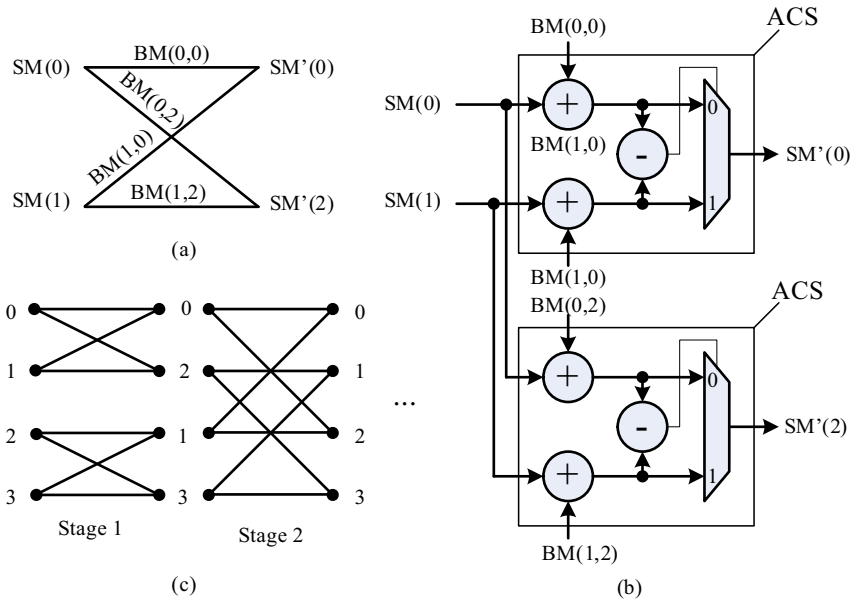

Fig. 15 ACS butterfly architecture. (a) Basic butterfly structure. (b) ACS butterfly hardware implementation. (c) In-place butterfly structure for a 4-state trellis diagram.

approach is feasible for a modern DSP processor, it is much more cost effective to implement the Viterbi decoder with a hardware accelerator. The decoder can be memory mapped to the DSP external memory space so that the DMA transfer can be utilized without the intervention of the host DSP. Data is passed in and out is in a pipelined manner so that the decoding can be simultaneously performed with I/O operations.

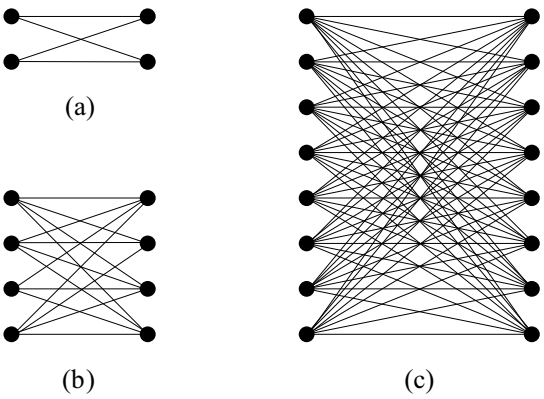

Fig. 16 Different radix trellis structure. (a) Radix-2 trellis. (b) Radix-4 trellis. (c) Radix-8 trellis.

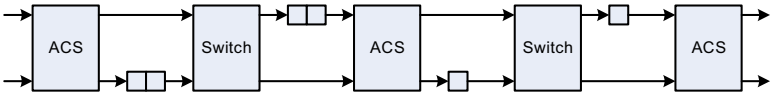

Fig. 17 Radix-8 ACS architecture.

2.3.2 Turbo Decoder Accelerator Architecture

Turbo codes are a class of high-performance capacity-approaching error-correcting codes [5]. As a break-through in coding theory, Turbo codes are widely used in many 3G/4G wireless standards such as CDMA2000, WCDMA/UMTS, 3GPP LTE, and IEEE 802.16e WiMax. However, the inherently large decoding latency and complex iterative decoding algorithm have made it rarely being implemented in a general purpose DSP. For example, Texas Instruments' latest multi-core DSP processor TI C6474 employs a Turbo decoder accelerator to support 2 Mbps CDMA Turbo codes for the base station [27]. The decoding throughput requirement for 3GPP LTE Turbo codes is to be more than 80 Mbps in the uplink and 320 Mbps in the downlink. Because the Turbo codes used in many standards are very similar, e.g. the encoding polynomials are same for WCDMA/UMTS/LTE, the Turbo decoder is often accelerated by reconfigurable hardware.

A classic Turbo encoder structure is depicted in Figure 19. The basic encoder consists of two systematic convolutional encoders and an interleaver. The information sequence **u** is encoded into three streams: systematic, parity 1, and parity 2. Here the interleaver is used to permute the information sequence into a second different sequence for encoder 2. The performance of a Turbo code depends critically on the interleaver structure [36].

The BCJR algorithm [3], also called forward-backward algorithm or Maximum *a posteriori* (MAP) algorithm, is the main component in the Turbo decoding process. The basic structure of Turbo decoding is functionally illustrated in Figure 20.

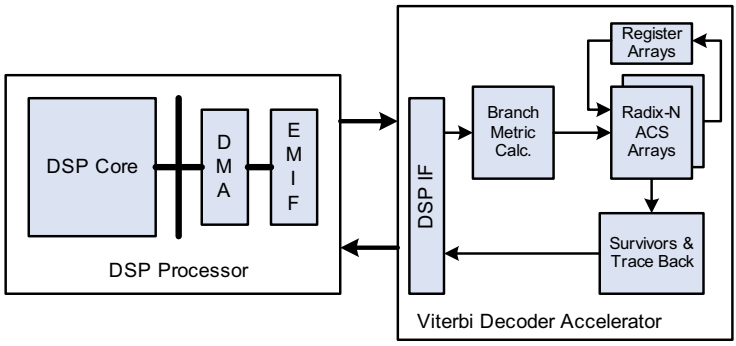

Fig. 18 A generic Viterbi decoder accelerator architecture. Data movement between DSP processor and accelerator is via DMA. Fully-parallel ACS function units are used to support high speed decoding.

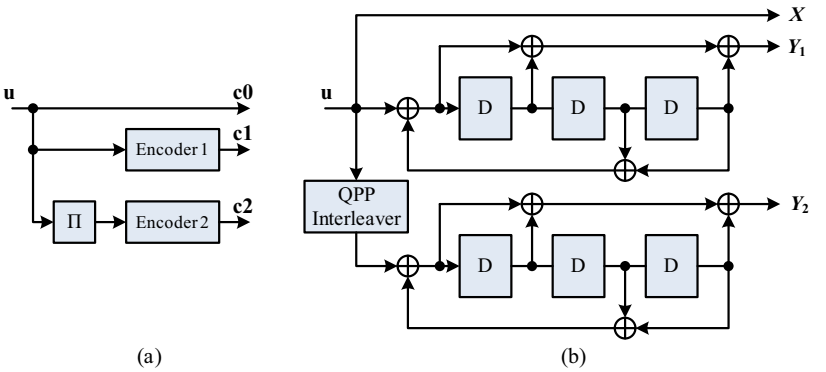

Fig. 19 Turbo encoder structure. (a) Basic structure. (b) Structure of Turbo encoder in 3GPP LTE.

The decoding is based on the MAP algorithm. During the decoding process, each MAP decoder receives the channel data and *a priori* information from the other constituent MAP decoder through interleaving (π) or deinterleaving (π^{-1}), and produces extrinsic information at its output. The MAP algorithm is an optimal symbol decoding algorithm that minimizes the probability of a symbol error. It computes the *a posteriori* probabilities (APPs) of the information bits as follows:

$$\Lambda(\hat{u}_k) = \max_{\mathbf{u}:u_k=1}^{*} \Big\{ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)) \Big\} \tag{11}$$

$$- \max_{\mathbf{u}:u_k=0}^{*} \Big\{ \alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k) + \beta_k(s_k)) \Big\}, \tag{12}$$

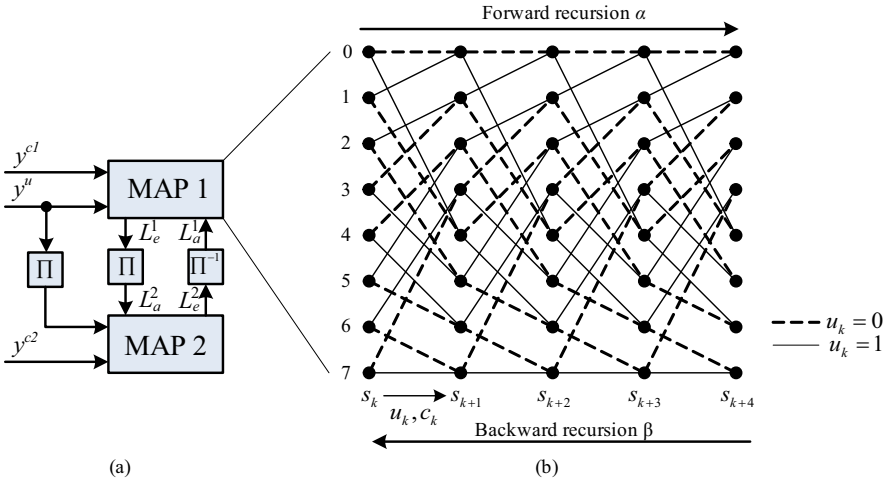

Fig. 20 Basic structure of an iterative Turbo decoder. (a) Iterative decoding based on two MAP decoders. (b) Forward/backward recursion on trellis diagram.

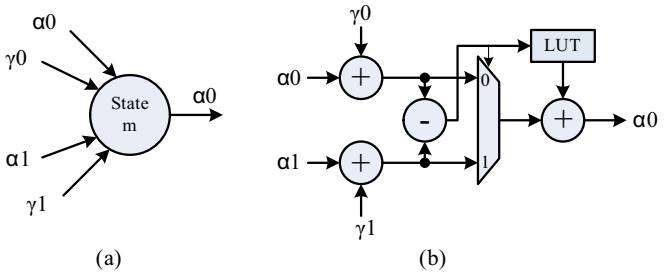

Fig. 21 ACSA structure. (a) Flow of state metric update. (b) Circuit implementation of an ACSA unit.

where α_k and β_k denote the forward and backward state metrics, and are calculated as follows:

$$\alpha_k(s_k) = \overset{*}{\max_{s_{k-1}}}\{\alpha_{k-1}(s_{k-1}) + \gamma_k(s_{k-1}, s_k)\}, \tag{13}$$

$$\beta_k(s_k) = \overset{*}{\max_{s_{k+1}}}\{\beta_{k+1}(s_{k+1}) + \gamma_k(s_k, s_{k+1})\}. \tag{14}$$

The γ_k term above is the branch transition probability that depends on the trellis diagram, and is usually referred to as a branch metric. The max star operator employed in the above descriptions is the core arithmetic computation that is required by the MAP decoding. It is defined as:

$$\overset{*}{\max}(a,b) = \log(e^a + e^b) = \max(a,b) + \log(1 + e^{-|a-b|}). \tag{15}$$

A basic add-compare-select-add (ACSA) unit is shown in Figure 21. This circuit can process one step of the trellis per cycle and is often referred to as Radix-2 ACSA unit. To increase the processing speed, the trellis can be transformed by merging every two stages into one radix-4 stage as shown in Figure 22. Thus, the throughput can be doubled by applying this transform. For an N state Turbo codes, N such ACSA unit would be required in each step of the trellis processing. To maximize the decoding throughput, a parallel implementation is usually employed to compute all the N state metrics simultaneously.

In the original MAP algorithm, the entire set of forward metrics needs to be computed before the first soft log-likelihood ratio (LLR) output can be generated. This results in a large storage of K metrics for all N states, where K is the block length and N is the number of states in the trellis diagram. Similar to the Viterbi algorithm, a sliding window algorithm is often applied to the MAP algorithm to reduce the decoding latency and memory storage requirement. By selecting a proper length of the sliding window, e.g. 32 for a rate 1/3 code, there is nearly no bit error rate (BER) performance degradation. Figure 23(a) shows an example of the sliding window algorithm, where a dummy reverse metric calculation (RMC) is used to get the initial values for β metrics. The sliding window hardware architecture is shown

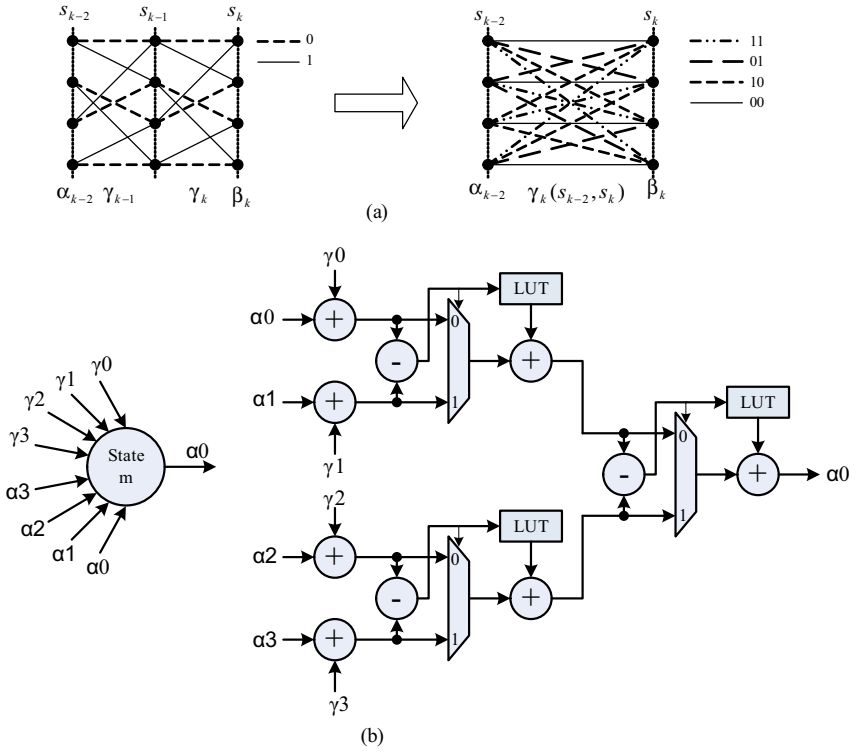

Fig. 22 (a) An example of Radix-4 trellis. (b) Radix-4 ACSA circuit implementation.

in Figure 23(b). The decoding operation is based on three recursion units, two used for the reverse (or backward) recursions (dummy RMC 1 and effective RMC 2), and one for forward recursion (FMC). Each recursion unit contains parallel ACSA units. After a fixed latency, the decoder produces the soft LLR outputs on every clock cycle. To further increase the throughput, a parallel sliding window scheme [6, 30, 34, 44, 48] is often applied as shown in Figure 24.

Another key component of Turbo decoders is the interleaver. Generally, the interleaver is a device that takes its input bit sequence and produces an output sequence that is as uncorrelated as possible. Theoretically a random interleaver would have the best performance. But it is difficult to implement a random interleaver in hardware. Thus, researchers are investigating pseudo-random interleavers such as the row-column permutation interleaver for 3G Rel-99 Turbo coding as well as the new QPP interleaver [41] for 3G LTE Turbo coding. The main differences between these two types of pseudo-random interleavers is the capability to support parallel Turbo decoding. The drawback of the row-column permutation interleaver is that memory conflicts will occur when employing multiple MAP decoders for parallel decoding. Extra buffers are necessary to solve the memory conflicts caused by the row-column permutation interleaver [37]. To solve this problem, the new 3G LTE standard has

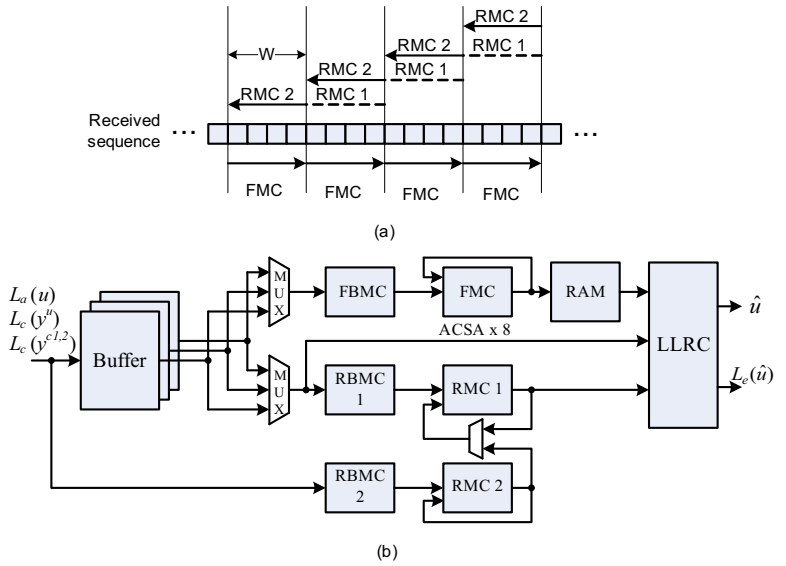

Fig. 23 Sliding window MAP decoder. (a) An example of sliding window MAP algorithm, where a dummy RMC is performed to achieve the initial β metrics. (b) MAP decoder hardware architecture.

adopted a new interleaver structure called QPP interleaver [41]. Given an information block length N, the x-th QPP interleaved output position is given by

$$\Pi(x) = (f_2x^2 + f_1x) \bmod N, 0 \leq x, f_1, f_2 < N. \tag{16}$$

It has been shown in [41] that the QPP interleaver will not cause memory conflicts as long as the parallelism level is a factor of N. The simplest approach to implement an interleaver is to store all the interleaving patterns in non-violating memory such as ROM. However, this approach can become very expensive because it is necessary to store a large number of interleaving patterns to support decoding of multiple block size Turbo codes such as 3GPP LTE Turbo codes. Fortunately, there usually exists an efficient hardware implementation for the interleaver. For example, Figure 25 shows a circuit implementation for the QPP interleaver in 3GPP LTE standard [44].

A basic Turbo accelerator architecture is shown in Figure 26. The main difference between the Viterbi decoder and the Turbo decoder is that the Turbo decoder is based on the iterative message passing algorithms. Thus, a Turbo accelerator may need more communication and control coordination with the DSP host processor. For example, the interleaving addresses can be generated by the DSP processor and passed to the Turbo accelerator. The DSP can monitor the decoding process to decide when to terminate the decoding if there are no more decoding gains. Alternately, the Turbo accelerator can be configured to operate without DSP intervention. To support this feature, some special hardware such as interleavers have to be configurable via DSP control registers. To decrease the required bus bandwidth, interme-

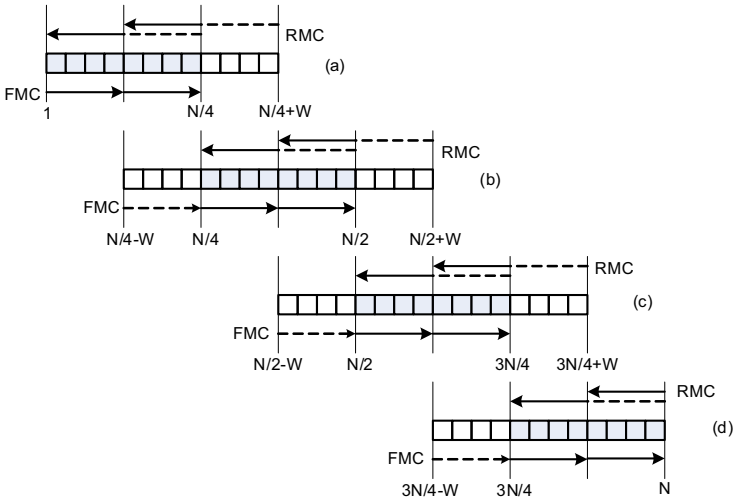

Fig. 24 An example of parallel sliding window decoding, where a decode block is sliced into 4 sections. The sub-blocks are overlapped by one sliding window length W in order to get the initial value for the boundary states.

diate results should not be passed back to the DSP processor. Only the successfully decoded bits need to be passed back to the DSP processor, e.g. via the DSP DMA controller. Further, to support multiple Turbo codes in different communication systems, a flexible MAP decoder is necessary. In fact, many standards employ similar Turbo code structures. For instance, CDMA, WCDMA, UMTS, and 3GPP LTE all use an eight-state binary Turbo code with polynomial (13, 15, 17). Although IEEE 802.16e WiMax and DVB-RCS standards use a different eight-state double binary Turbo code, the trellis structures of these Turbo codes are very similar as illustrated in Figure 27. Thus, it is possible design multi-standard Turbo decoders based on flexible MAP decoder datapaths [31, 40, 44]. It has been shown in [44] that the area overhead to support multi-codes is only about 7%. In addition, when the throughput requirement is high, e.g. more than 20 Mbps, multiple MAP decoders can be activated to increase the throughput performance.

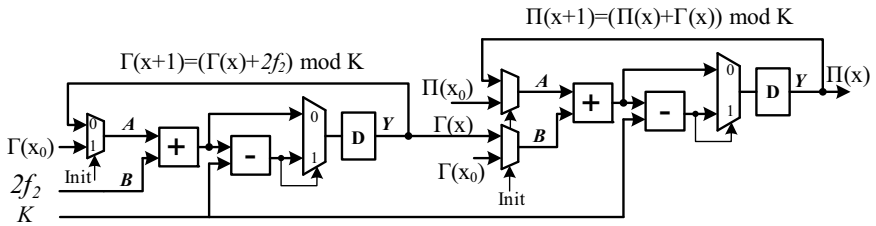

Fig. 25 An circuit implementation for the QPP interleaver $\pi(x) = (f_2x^2 + f_1x) \bmod K$ [44].

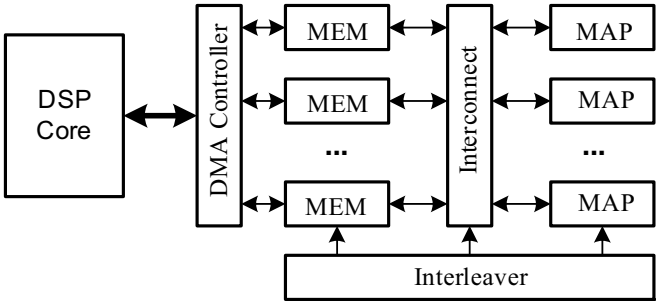

Fig. 26 Turbo decoder accelerator architecture. Multiple MAP decoders are used to support high throughput decoding of Turbo codes. Special function units such as interleavers are also implemented in hardware.

In summary, due to the iterative structures, a Turbo decoder needs more Gflops than what is available in a general purpose DSP processor. For this reason, Texas Instruments' latest C64x DSP processor integrates a 2 Mbps 3G Turbo decoder accelerator in the same die [27]. Because of the parallel and recursive algorithms and special logarithmic arithmetics, it is more cost effective to realize a Turbo decoder in hardware.

2.3.3 LDPC Decoder Accelerator Architecture

A low-density parity-check (LDPC) code [18] is another important error correcting code that is the among one of the most efficient coding schemes discovered as of

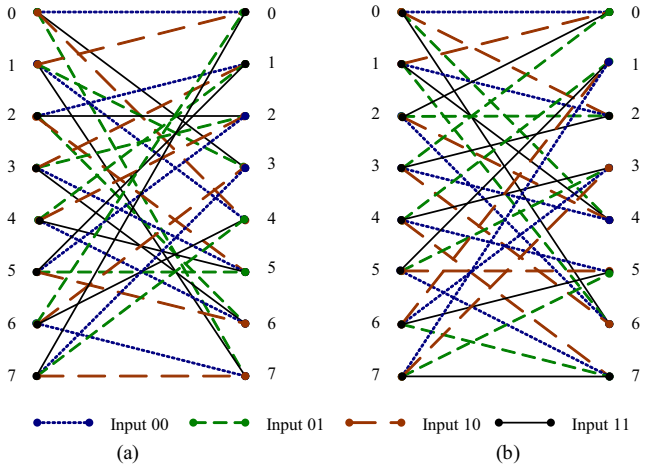

Fig. 27 Radix-4 trellis structures of (a) CDMA/WCDMA/UMTS/LTE Turbo codes and (b) WiMax/DVB-RCS Turbo codes.

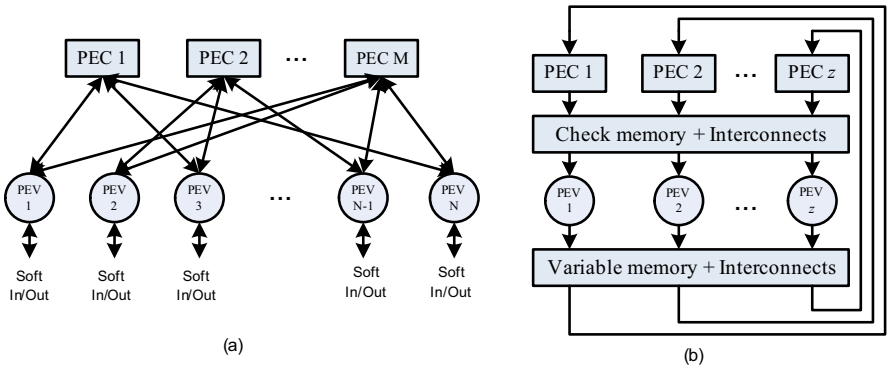

Fig. 28 Implementation of LDPC decoders, where PEC denotes processing element for check node and PEV denotes processing element for variable node. (a) Fully-parallel. (b) Semi-parallel.

2009. The remarkable error correction capabilities of LDPC codes have led to their recent adoption in many standards, such as IEEE 802.11n, IEEE 802.16e, and IEEE 802 10GBase-T. The huge computation and high throughput requirements make it very difficult to implement a high throughput LDPC decoder on a general purpose DSP. For example, a 5.4 Mbps LDPC decoder was implemented on TMS320C64xx DSP running at 600 MHz [29]. This throughput performance is not enough to support high data rates defined in new wireless standards. Thus, it is important to develop area and power efficient hardware LDPC decoding accelerators.

A binary LDPC code is a linear block code specified by a very sparse binary $M \times N$ parity check matrix: $\mathbf{H} \cdot \mathbf{x}^T = 0$, where $\mathbf{x}$ is a codeword and $\mathbf{H}$ can be viewed as a bipartite graph where each column and row in $\mathbf{H}$ represent a variable node and a check node, respectively.

The decoding algorithm is based on the iterative message passing algorithm (also called belief propagation algorithm), which exchanges the messages between the variable nodes and check nodes on graph. The hardware implementation of LDPC decoders can be serial, semi-parallel, and fully-parallel as shown in Figure 28. Fully-parallel implementation has the maximum processing elements to achieve very high throughput. Semi-parallel implementation, on the other hand, has a lesser number of processing elements that can be re-used, e.g. z number of processing elements are employed in Figure 28(b). In a semi-parallel implementation, memories are usually required to store the temporary results. In many practical systems, semi-parallel implementations are often used to achieve 100 Mbps to 1 Gbps throughput with reasonable complexity [7, 21, 35, 42, 43, 54].

In LDPC decoding, the main complexity comes from the check node processing. Each check node receives a set of variable node messages denoted as $\mathcal{N}_m$. Based on these data, check node messages are computed as

$$\Lambda_{mn} = \sum_{j \in \mathcal{N}_m \backslash n} \boxplus \lambda_{mj} = \Big(\sum_{j \in \mathcal{N}_m} \boxplus \lambda_{mj} \Big) \boxminus \lambda_{mn},$$

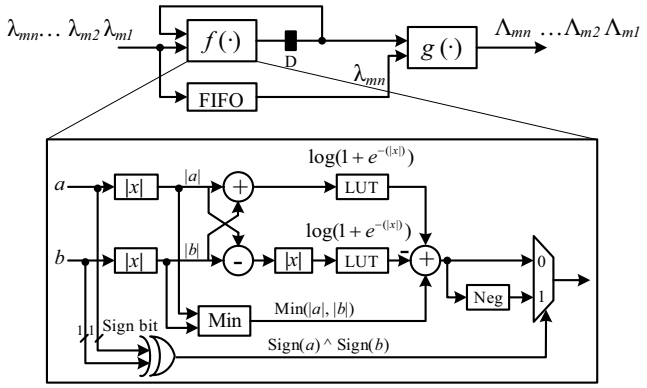

Fig. 29 Recursive architecture to compute check node messages [42].

where Λ_{mn} and λ_{mn} denote the check node message and the variable node message, respectively. The special arithmetic operators $\boxplus$ and $\boxminus$ are defined as follows:

$$
\begin{aligned}
a \boxplus b &\triangleq f(a,b) = \log \frac{1+e^a e^b}{e^a + e^b} \\
&= \operatorname{sign}(a)\operatorname{sign}(b)\Big(\min(|a|,|b|) + \log(1+e^{-(|a|+|b|)}) - \log(1+e^{-\left||a|-|b|\right|})\Big), \\
a \boxminus b &\triangleq g(a,b) = \log \frac{1-e^a e^b}{e^a - e^b} \\
&= \operatorname{sign}(a)\operatorname{sign}(b)\Big(\min(|a|,|b|) + \log(1-e^{-(|a|+|b|)}) - \log(1-e^{-\left||a|-|b|\right|})\Big).
\end{aligned}
$$

Figure 29 shows a hardware implementation from [42] to compute check node message Λ_{mn} for one check row m. Because multiple check rows can be processed simultaneously in the LDPC decoding algorithm, multiple such check node units can be used to increase decoding speed. As the number of ALU units in a general purpose DSP processor is limited, it is difficult to achieve more than 10 Mbps throughput in a DSP implementation.

Given a random LDPC code, the main complexity comes not only from the complex check node processing, but also from the interconnection network between check nodes and variable nodes. To simplify the routing of the interconnection network, many practical standards usually employ structured LDPC codes, or quasi-cyclic LDPC (QC-LDPC) codes. The parity check matrix of a QC-LDPC code is shown in Figure 30. Table 3 summaries the design parameters of the QC-LDPC codes for IEEE 802.11n WLAN and IEEE 802.16e WiMax wireless standards. As can be seen, many design parameters are in the same range for these two applications, thus it is possible to design a reconfigurable hardware to support multiple standards [42].

Table 3 Design parameters for **H** in standardized LDPC codes

	z	j	k	Check node degree	Variable node degree	Max. throughput
WLAN 802.11n	27-81	4-12	24	7-22	2-12	600 Mbps
WiMax 802.16e	24-96	4-12	24	6-20	2-6	144 Mbps

As an example, a multi-standard semi-parallel LDPC decoder accelerator architecture is shown in Figure 31 [42]. In order to support several hundreds Mbps data rate, multiple PEs are used to process multiple check rows simultaneously. As with Turbo decoding, LDPC decoding is also based on an iterative decoding algorithm. The iterative decoding flow is as follows: at each iteration, $1 \times z$ APP messages, denoted as L_n are fetched from the L-memory and passed through a permuter (eg. barrel shifter) to be routed to z PEs (z is the parallelism level). The soft input information λ_{mn} is formed by subtracting the old extrinsic message Λ_{mn} from the APP message L_n. Then the PEs generate new extrinsic messages Λ_{mn} and APP messages L_n, and store them back to memory. The operation mode of the LDPC accelerator needs to be configured in the beginning of the decoding. After that, it should work without DSP intervention. Once it has finished decoding, the decoded bits are passed back to the DSP processor. Figure 32 shows the ASIC implementation result of this decoder (VLSI layout view) and its power consumption for different block sizes. As the block size increases, the number of active PEs increases, thus more power is consumed.

3 Summary

Digital signal processing complexity in high-speed wireless communications is driving a need for high performance heterogenous DSP systems with real-time processing. Many wireless algorithms, such as channel decoding and MIMO detection, demonstrate significant data parallelism. For this class of data-parallel algorithms, application specific DSP accelerators are necessary to meet real-time requirements

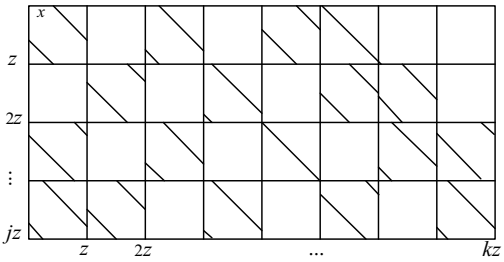

Fig. 30 Structured LDPC parity check matrix with j block rows and k block columns. Each submatrix is a $z \times z$ identity shifted matrix.

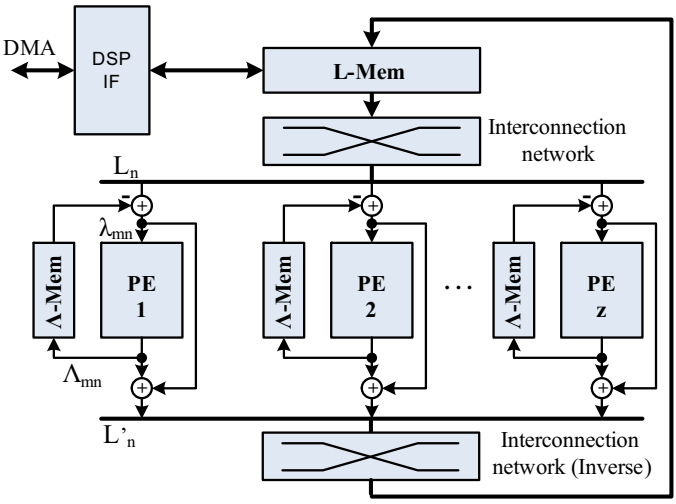

Fig. 31 Semi-parallel LDPC decoder accelerator architecture. Multiple PEs (number of z) are used to increase decoding speed. Variable messages are stored in L-memory and check messages are stored in Λ-memory. An interconnection network along with an inverse interconnection network are used to route data.

while minimizing power consumption. Spatial locality of data, data level parallelism, computational complexity, and task level parallelism are four major criteria to identify which DSP algorithm should be off-loaded to an accelerator. Additional cost incurred from the data movement between DSP and hardware accelerator must be also considered.

There are a number of DSP architectures which include true hardware based accelerators. Examples of these include the Texas Instruments' C64x series of DSPs

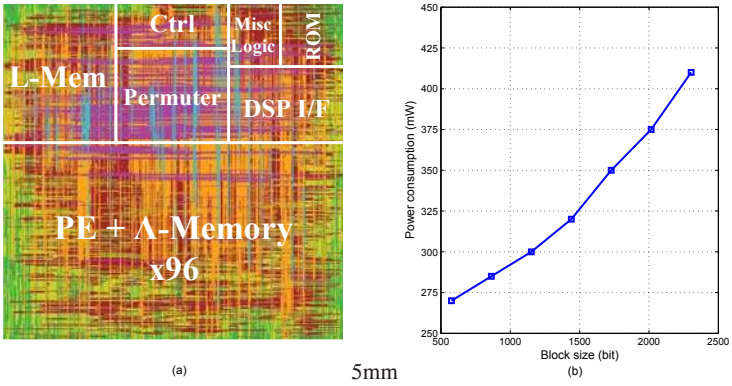

Fig. 32 An example of a LDPC decoder hardware accelerator [42]. (a) VLSI layout view (3.5 mm^2 area, 90nm technology). (b) power consumptions for different block sizes.

which include a 2 Mbps Turbo decoding accelerator [27], and Freescale Semiconductor's six core broadband wireless access DSP MSC8156 which includes a programmable 200 Mbps Turbo decoding accelerator (6 iterations), a 115 Mbps Viterbi decoding accelerator ($K = 9$), an FFT/IFFT accelerator for sizes 128, 256, 512, 1024 or 2048 points at up to 350 million samples/s, and a DFT/IDFT for sizes up to 1536 points at up to 175 million samples/s [38].

Relying on a single DSP processor for all signal processing tasks would be a clean solution. As a practical matter, however, multiple DSP processors are necessary for implementing a next generation wireless handset or base station. This means greater system cost, more board space, and more power consumption. Integrating hardware communication accelerators, such as MIMO detectors and channel decoders, into the DSP processor silicon can create an efficient System on Chip. This offers many advantage: the dedicated accelerators relieve the DSP processor of the parallel computation-intensive signal processing burden, freeing DSP processing capacity for other system control functions that more greatly benefit from programmability.

References

1. Alamouti, S.M.: A simple transmit diversity technique for wireless communications. IEEE Journal on Selected Areas in Communications **16**(8), 1451–1458 (1998)
2. Amiri, K., Cavallaro, J.R.: FPGA implementation of dynamic threshold sphere detection for MIMO systems. IEEE Asilomar Conference on Signals, Systems and Computers pp. 94–98 (2006)
3. Bahl, L., Cocke, J., Jelinek, F., Raviv, J.: Optimal decoding of linear codes for minimizing symbol error rate. IEEE Transactions on Information Theory **IT-20**, 284–287 (1974)
4. Bass, B.: A low-power, high-performance, 1024-point FFT processor. In: IEEE International Solid-State Circuit Conference (ISSCC) (1999)
5. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: Turbo-codes. In: IEEE International Conference on Communications, pp. 1064–1070 (1993)
6. Bougard, B., Giulietti, A., Derudder, V., Weijers, J.W., Dupont, S., Hollevoet, L., Catthoor, F., Van der Perre, L., De Man, H., Lauwereins, R.: A scalable 8.7-nJ/bit 75.6-Mb/s parallel concatenated convolutional (turbo-) codec. In: IEEE International Solid-State Circuit Conference (ISSCC) (2003)
7. Brack, T., Alles, M., Lehnigk-Emden, T., Kienle, F., Wehn, N., Lapos, Insalata, N., Rossi, F., Rovini, M., Fanucci, L.: Low complexity LDPC code decoders for next generation standards. In: Design, Automation, and Test in Europe (DATE), pp. 1–6 (2007)
8. Brogioli, M.: Reconfigurable heterogeneous DSP/FPGA based embedded architectures for numerically intensive embedded computingworkloads. Ph.D. thesis, Rice University, Houston, Texas, USA (2007)
9. Brogioli, M., Radosavljevic, P., Cavallaro, J.: A general hardware/software codesign methodology for embedded signal processing and multimedia workloads. In: IEEE 40th Asilomar Conference on Signals, Systems, and Computers, pp. 1486–1490 (2006)
10. Burg, A.: VLSI circuits for MIMO communication systems. Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, Switzerland (2006)
11. Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W., Bolcskei, H.: VLSI implementation of MIMO detection using the sphere decoding algorithm. IEEE Journal of Solid-State Circuits **40**(7), 1566–1577 (2005)

12. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**, 297–301 (1965)
13. Damen, M.O., Gamal, H.E., Caire, G.: On maximum likelihood detection and the search for the closest lattice point. IEEE Transactions on Information Theory **49**(10), 2389–2402 (2003)
14. Devices, A.: The SHARC Processor Family. http://www.analog.com/en/embedded-processing-dsp/sharc/processors/index.html (2009)
15. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Mathematics of Computation **44**(170), 463–471 (1985)
16. Forney, G.D.: The Viterbi algorithm. Proceedings of the IEEE **61**(3), 268–278 (1973)
17. Foschini, G.: Layered space-time architecture for wireless communication in a fading environment when using multiple antennas. Bell Labs. Tech. Journal **2**, 41–59 (1996)
18. Gallager, R.: Low-density parity-check codes. IEEE Transactions on Information Theory **8**, 21–28 (1962)
19. Garrett, D., Davis, L., ten Brink, S., Hochwald, B., Knagge, G.: Silicon complexity for maximum likelihood MIMO detection using spherical decoding. IEEE Journal of Solid-State Circuits **39**(9), 1544–1552 (2004)
20. Golden, G., Foschini, G.J., Valenzuela, R.A., Wolniansky, P.W.: Detection algorithms and initial laboratory results using V-BLAST space-time communication architecture. Electronics Letters **35**, 14–15 (1999)
21. Gunnam, K., Choi, G.S., Yeary, M.B., Atiquzzaman, M.: VLSI architectures for layered decoding for irregular LDPC codes of WiMax. In: IEEE International Conference on Communications, pp. 4542–4547 (2007)
22. Guo, Z., Nilsson, P.: Algorithm and implementation of the K-best sphere decoding for MIMO detection. IEEE Journal on Selected Areas in Communications **24**(3), 491–503 (2006)
23. Hassibi, B., Vikalo, H.: On the sphere-decoding algorithm I. Expected complexity. IEEE Transactions on Signal Processing **53**(8), 2806–2818 (2005)
24. Hunter, H.C., Moreno, J.H.: A new look at exploiting data parallelism in embedded systems. In: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 159–169 (2003)
25. Tensilica Inc.: http://www.tensilica.com (2009)
26. Texas Instruments: TMS320C55x DSP CPU Programmer's Reference Supplement. http://focus.ti.com/lit/ug/spru652g/spru652g.pdf (2005)
27. Texas Instruments: TMS320C6474 high performance multicore processor datasheet. http://focus.ti.com/docs/prod/folders/print/tms320c6474.html (2008)
28. Instruments, T.: TMS320C6000 CPU and Instruction Set Reference Guide. http://dspvillage.ti.com (2001)
29. Lechner, G., Sayir, J., Rupp, M.: Efficient DSP implementation of an LDPC decoder. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 665–668 (2004)
30. Lee, S.J., Shanbhag, N.R., Singer, A.C.: Area-efficient high-throughput MAP decoder architectures. IEEE Transactions on VLSI Systems **13**, 921–933 (2005)
31. Martina, M., Nicola, M., Masera, G.: A flexible UMTS-WiMax turbo decoder architecture. IEEE Transactions on Circuits and Systems II **55**, 369–273 (2008)
32. Myllylä, M., Silvola, P., Juntti, M., Cavallaro, J.R.: Comparison of two novel list sphere detector algorithms for mimo-ofdm systems. IEEE International Symposium on Personal Indoor and Mobile Radio Communications (2006)
33. Parhi, K.K.: VLSI Digital Signal Processing Systems Design and Implementation. Wiley (1999)
34. Prescher, G., Gemmeke, T., Noll, T.G.: A parametrizable low-power high-throughput turbo-decoder. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 5, pp. 25–28 (2005)
35. Rovini, M., Gentile, G., Rossi, F., Fanucci, L.: A scalable decoder architecture for IEEE 802.11n LDPC codes. In: IEEE Global Telecommunications Conference, pp. 3270–3274 (2007)

36. Sadjadpour, H., Sloane, N., Salehi, M., Nebe, G.: Interleaver design for turbo codes. IEEE Journal on Seleteced Areas in Communications **19**, 831–837 (2001)
37. Salmela, P., Gu, R., Bhattacharyya, S., Takala, J.: Efficient parallel memory organization for turbo decoders. In: Proc. European Signal Processing Conf., pp. 831–835 (2007)
38. Freescale Semiconductor: MSC8156 six core broadband wireless access DSP. www.freescale.com/starcore (2009)
39. Semiconductor, F.: Freescale Starcore Architecture. www.freescale.com/starcore (2009)
40. Shin, M.C., Park, I.C.: A programmable turbo decoder for multiple 3G wireless standards. In: IEEE Solid-State Circuits Conference, vol. 1, pp. 154–484 (2003)
41. Sun, J., Takeshita, O.: Interleavers for turbo codes using permutation polynomials over integer rings. IEEE Transactions on Information Theory **51**(1) (2005)
42. Sun, Y., Cavallaro, J.R.: A low-power 1-Gbps reconfigurable LDPC decoder design for multiple 4G wireless standards. In: IEEE International SOC Conference (SoCC), pp. 367–370 (2008)
43. Sun, Y., Karkooti, M., Cavallaro, J.R.: VLSI decoder architecture for high throughput, variable block-size and multi-rate LDPC codes. In: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2104–2107 (2007)
44. Sun, Y., Zhu, Y., Goel, M., Cavallaro, J.R.: Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standards. In: IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 209–214 (2008)
45. Tarokh, V., Jafarkhani, H., Calderbank, A.R.: Space-time block codes from orthogonal designs. IEEE Transactions on Information Theory **45**(5), 1456–1467 (1999)
46. Tarokh, V., Jafarkhani, H., Calderbank, A.R.: Space time block coding for wireless communications: Performance results. IEEE Journal on Selected Areas in Communications **17**(3), 451–460 (1999)
47. Telatar, I.E.: Capacity of multiantenna Gaussian channels. European Transactions on Telecommunications **10**, 585–595 (1999)
48. Thul, M.J., Gilbert, F., Vogt, T., Kreiselmaier, G., Wehn, N.: A scalable system architecture for high-throughput turbo-decoders. Journal of VLSI Signal Processing pp. 63–77 (2005)
49. Viterbi, A.: Error bounds for convolutional coding and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory **IT-13**, 260–269 (1967)
50. Wijting, C., Ojanperä, T., Juntti, M., Kansanen, K., Prasad, R.: Groupwise serial multiuser detectors for multirate DS-CDMA. In: IEEE Vehicular Technology Conference, vol. 1, pp. 836–840 (1999)
51. Willmann, P., Kim, H., Rixner, S., Pai, V.S.: An efficient programmable 10 Gigabit Ethernet network interface card. In: ACM International Symposium on High-Performance Computer Architecture, pp. 85–86 (2006)
52. Wong, K., Tsui, C., Cheng, R.S., Mow, W.: A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels. IEEE International Symposium on Circuits and Systems **3**, 273–276 (2002)
53. Ye, Z.A., Moshovos, A., Hauck, S., Banerjee, P.: CHIMAERA: A High–Performance Architecture with a Tightly–Coupled Reconfigurable Functional Unit. In: Proceedings of the 27th Annual International Symposium on Computer Architecture, pp. 225–235 (2000)
54. Zhong, H., Zhang, T.: Block-LDPC: a practical LDPC coding system design approach. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **52**(4), 766–775 (2005)