Design and Implementation of a Belief Propagation Detector for Sparse Channels

Yanjie Peng, Kai Zhang, Andrew G. Klein and Xinming Huang
Department of Electrical and Computer Engineering
Worcester Polytechnic Institute
Worcester, MA 01609

***Abstract*—In this paper, we address the design and implementation of the symbol detector for sparse channels which are described as having long spanning durations but sparse multipath structure. The traditional maximum-likelihood (ML) algorithm provides an optimal performance to eliminate the multipath effect, however its complexity scales exponentially with the channel length. As a more efficient symbol detection algorithm through sparse channels, the iterative belief propagation (BP) algorithm has a complexity merely dependent on the number of nonzero channel coefficients, while achieving a near-optimal error performance. We present the architecture design for a reconfigurable low-complexity high-throughput BP detector. As an example, we implement a BP detector for quadrature phase-shift keying (QPSK) modulation on Xilinx Virtex 5 FPGA with a maximum frequency of 252 MHz and equivalently a throughput of 100.8 Mb/s at 5 iterations.**

***Keywords*-Belief propagation algorithm, sparse channel, symbol detection, architecture design.**

I. Introduction

Sparse channels are characterized as long multipath delay spreads but with few nonzero coefficients. Such channels arise in a number of wireless communication applications recently, for instance, in underwater acoustic communications and high-definition television (HDTV).

In these applications, intersymbol interference (ISI) is one of the major impairments to reliable symbol detection. Traditionally the optimal maximum-likelihood (ML) detector based on the Viterbi algorithm (VA) may be used to combat the ISI. However, the computational complexity of the ML detector increases exponentially with the delay spread of the channel. Thus, the optimal ML detector would be impractical for sparse channels whose delay spread is usually very long. Detectors for sparse channels have been investigated by several researchers. The parallel trellis Viterbi algorithm (PTVA) [7] reformulated the original single trellis into a set of independent trellises which could operate in parallel and have much less complexity. It requires that the sparse channel have equi-spaced coefficients, however, which often cannot be satisfied in practice. A near-optimal detector based on a multitrellis Viterbi algorithm (MVA) [1] was proposed which decomposes the trellis into multiple irregular subtrellises by investigating the dependencies between the received symbols. It was shown that the complexity only depends on the number of nonzero coefficients. However, the trellis decomposition is not straightforward for a general sparse channel. Decision feedback sequence estimation (DFSE) [2], which is a popular scheme for long ISI channels, can also be an alternative for sparse channels. However, its complexity approaches the traditional VA if we require near-optimal performance. Recently iterative belief propagation (BP) detectors have been proposed for ISI channels [4], [8]. It has been shown in [4] that, for the uncoded system the BP detector achieves near-optimal performance over frequency selective ISI channels. Furthermore, the BP detector has a complexity which only depends on the number of nonzero coefficients, and thus is very suited for sparse channels.

The BP algorithm, also known as the message passing algorithm, has been widely used for iterative decoding of low-density parity-check (LDPC) codes [6]. The BP detector is remarkably different from the LDPC decoder due to the following two reasons. Firstly, the parity check matrix for LDPC codes is usually designed for parallel processing, while the channel matrix for symbol detection is determined by the channel, which could be time-variant during detection. Another difference is that the calculating of messages between nodes is more complex for symbol detection. For these reasons, the parallel processing is infeasible for the BP detector. In this paper, we present a feasible and efficient BP detector for sparse channels. The architecture of the proposed BP detector is based on a pipelined layer processing scheme, which allows a high throughput but with low complexity. The detector is also reconfigurable so that it can adapt to time-varying ISI channels. We implement the BP detector for quadrature phase-shift keying (QPSK) modulation on a Xilinx Virtex 5 FPGA, which achieves a maximum frequency of 252 MHz and equivalently 100.8 Mb/s throughput at 5 iterations.

II. System Model

Assume that a sequence of N M-ary symbols $x[k] \in \{a_0, a_1, ..., a_{M-1}\}$ are transmitted through a complex ISI channel whose discrete-time channel impulse response (CIR) is described as $\mathbf{h} = [h[0], h[1], ..., h[L-1]]^T$,where L is the channel length. Letting D be the number of nonzero coefficients of $\mathbf{h}$, then $D \ll L$ for sparse channels. We assume that the channel $\mathbf{h}$ is known at the receiver. The

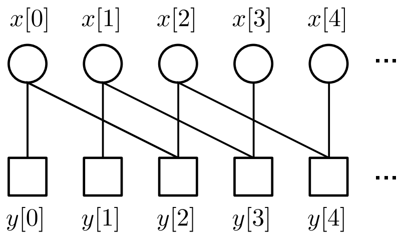

Figure 1. Factor graph of an example channel $[h[0], 0, h[2]]^T$

received signal at time instant k can be expressed as

$$y[k] = \sum_{i=0}^{L-1} x[k-i]h[i] + w[k], \quad (1)$$

where $w[k]$ is complex additive white Gaussian noise (AWGN) with noise power σ^2 at time instant k.

The input-output relationship of an ISI channel can be represented as a factor graph [6]. Fig. 1 shows the factor graph of the channel with a CIR of $[h[0], 0, h[2]]^T$. Channel input symbols are represented by the circle shaped variable nodes, and the channel output signals are denoted by the square shaped check nodes. The connections (edges) between the nodes represent dependencies of input and output. For example, $y[3]$ is connected to $x[1]$ and $x[3]$, since $y[3] = h[0]x[3] + h[3]x[0] + w[3]$ by (1).

III. Belief Propagation Algorithm

In the BP algorithm, check-to-variable (CTV) and variable-to-check (VTC) messages are transmitted along the edges to update each other iteratively. During the first phase, the CTV message is computed at the check nodes based on the CIR coefficients and the *a priori* probability information from the variable nodes. The updated message is then passed from each check node to its connected variable nodes. During the second phase, the variable nodes update their *a priori* information and send it back to their connected check nodes. The same procedure is repeated iteratively. After several iterations, the variable nodes are accumulated with sufficient likelihood information and a hard decision is made for each input symbol. The algorithm in log-likelihood ratio (LLR) domain are summarized as follows.

A. Calculating CTV messages

The message from check node m to variable node n is computed as (2), where $\mathbf{x}$ is the set of symbols connected with check node m and $Q_{j\rightarrow m}$ is the message from variable node j for check node m, and $\mathcal{N}(m)$ is the set of variable nodes connected with check node m. We use the function $\psi : \{0, 1, ...M-1\} \rightarrow \{a_0, a_1, ..., a_{M-1}\}$ to denote the modulation on the M-ary constellation, and the demodulation function is ψ^{-1}. Note that (2) has been simplified by Jacobian logarithm.

B. Calculating VTC messages

The message at variable node n for check node m is

$$Q_{n\rightarrow m}(i) = \sum_{j\in\mathcal{M}(n)\backslash m} R_{j\rightarrow n}(i), i = 0, 1, ..., M-1, \quad (3)$$

where $\mathcal{M}(n)$ is the set of check nodes connected with variable node n. Then we go back to the first step for the next iteration.

C. Summing up and decision

After iterating the above two steps for several times, the accumulated LLRs for variable node n are

$$\Lambda_n(i) = \sum_{j\in\mathcal{M}(n)} R_{j\rightarrow n}(i), i = 0, 1, ..., M-1. \quad (4)$$

By comparing these LLR sums, an estimation on the transmitted symbol $x[n]$ can be made by $\hat{x}[n] = \psi(\arg\max_i \Lambda_n[i]), i = 0, 1, ..., M-1$.

IV. Implementation of BP Detectors

A. Pipelined Layer Processing Scheme

Unlike the trellis-based algorithm, the intrinsic parallel structure of the BP algorithm allows very high-throughput detection with flooding schedule [5], in which all variable nodes and subsequently all the check nodes pass updated messages (LLRs) to their connected nodes in each iteration. However, in BP detectors this scheme is constrained by the large computing resource required by LLRs calculation as in (2). Recalling that there are D elements in $\mathbf{x}$, M^D values of $\left\{\frac{-|y[m]-\mathbf{h}^T\mathbf{x}|^2}{2\sigma^2} + \sum_{j\in\mathcal{N}(m)\backslash n} Q_{j\rightarrow m}(\psi^{-1}(x[j]))\right\}$ should be calculated to obtain the LLRs from check node m to D connected variable nodes. Assuming that three multiplications are needed in each calculation, and N symbols are to be processed, there are $3NM^D$ multiplications required for flooding schedule. In a typical case, for example, $D = 3$, $N = 1K$, and $M = 4$ (QPSK), we need $3\times 1K\times 4^3 = 196K$ multiplications to calculate the CTV LLRs.

Since the parallel message passing schedule consumes too much computing resource, we adopt a sequential message passing schedule to reduce the cost of implementation. The sequential schedule is performed in a way that each check node and its connected variable nodes are treated as a layer, and within each layer processing, we update the LLRs from the check node (for example the check node m) to its neighboring variable nodes $\mathcal{N}(m)$, and then update the accumulated LLRs associated to these variable nodes. These updated messages will be used in next layer processing. Since each layer updates the messages sequentially instead of simultaneously, computing resource can be reused among layers, so that number of multiplications reduces remarkably

$$R_{m\to n}(i) \approx \max_{\forall \mathbf{x}:x[n]=a_i} \left\{ \frac{-\left|y[m]-\mathbf{h}^T\mathbf{x}\right|^2}{2\sigma^2} + \sum_{j\in\mathcal{N}(m)\backslash n} Q_{j\to m}(\psi^{-1}(x[j])) \right\} - \max_{\forall \mathbf{x}:x[n]=a_0} \left\{ \frac{-\left|y[m]-\mathbf{h}^T\mathbf{x}\right|^2}{2\sigma^2} + \sum_{j\in\mathcal{N}(m)\backslash n} Q_{j\to m}(\psi^{-1}(x[j])) \right\}, i = 0,1,...,M-1. \quad (2)$$

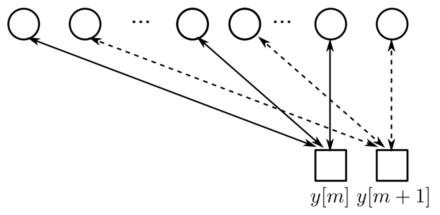

Figure 2. Pipelined layer processing scheme

to $3M^D$, implying a cost of $3 \times 4^3 = 192$ multiplications in the previous case. Studies [3] also showed that sequential BP algorithm improves the convergence speed with respect to number of iterations.

The problem of this scheme is that each layer processing will take a long time to finish due to the high computational complexity of (2). The large latency between layer processings results in a very low throughput rate. The bottleneck can be overcomed by pipelining layer processings as demonstrated in Fig. 2 which shows the factor graph of a typical sparse channel with $D = 3$. The messages on the m-th layer (solid lines) are updated first, and messages on the $(m+1)$-th layer (dashed lines) are processed next. The two successive layer processings can be pipelined as there is no dependency. Note that the independency between layers can be easily satisfied for sparse channels since the nonzero coefficients are often widely separated. It is also observed from the factor graph that the maximum depth of the pipeline is equal to the minimum distance of two adjacent nonzero coefficients. On the other hand, it is possible that a part of nonzero coefficients are closely distributed for some channel. Under this circumstance we can apply a sparse partial response equalizer (sPRE) [8] before the BP detector. The sparse channel is equalized to a channel with target impulse response (TIR), which can be carefully designed to satisfy the need for pipelined implementation. As the sPRE is beyond the scope of this paper, we just assume that the pipelined layer processing scheme is always applicable in our case. Generally when the BP detector works at a frequency of f Hz and the number of iteration is n_{it}, we can achieve a throughput of $f \log_2 M/n_{it}$ bps.

B. Cache-Based Architecture

The overall architecture is shown in Fig. 3. The detector mainly consists of a layer processing unit (LPU) responsible for layer message update, the LLR sum memory and CTV message memory which are used to store the belief propagation messages, a cache which is a temporary storage for the LLRs of the nodes currently being processed, and a decision unit for estimating the input symbols.

The LPU is the core computing unit executing message updating expressed in (2), (3) and (4). We note that the VTC messages Q in (2) can be obtained from LLR sum Λ and CTV messages R by (3) and (4), i.e. $Q_{n\to m}(i) = \Lambda_n(i) - R_{m\to n}(i), i = 0,1,...,M-1$. To minimize the hardware cost, we only store and update Λ instead of Q since the memory for Q is about D times as large as the memory for Λ. In each layer processing, the LPU first calculates the VTC messages Q, and then updates the CTV messages R and LLR sums Λ. The architecture of LPU is given in Fig. 4. The calculation of $\max\left\{ \frac{-\left|y[m]-\mathbf{h}^T\mathbf{x}\right|^2}{2\sigma^2} + \sum_{j\in\mathcal{N}(m)\backslash n} Q_{j\to m}(\psi^{-1}(x[j])) \right\}, \forall \mathbf{x} : x[n] = a_i$ is carried out in M square blocks for $i = 0,1,...,M-1$ respectively. We only show the message updating along one edge within the m-th layer. The hardware should be expanded by D times for the complete layer processing. A pipeline architecture is implemented such that the LPU can take a new signal input every clock cycle.

The updated R from LPU are stored back into the CTV message memory, and they will be fetched from the memory in next iteration. There are $(M-1)$ LLRs for each possible connection, and thus $(M-1)ND$ LLRs for the entire factor graph. The size of CTV message memory is $(M-1)ND \times L_R$where L_R is the word length of the CTV LLR.

The cache is a register bank only containing the accumulated LLRs of L variable nodes being processed. This is similar to a “sliding window” that only needs to fetch one new LLR from the Λ memory when LPU moves to process the next layer. The updated Λ from LPU consists of D sets of LLRs corresponding to D connected variable nodes, which is illustrated in Fig 2. The updated Λ may be used for the following layer processings very soon. For efficient memory operations, they are written back to the cache to replace the old values. Only the Λ related to the leftmost variable node will be used in the next iteration, and they are stored back to the Λ memory.

The LLR sum memory are used to store Λ. There are $(M-1)$ LLRs for each variable nodes. Thus the size of

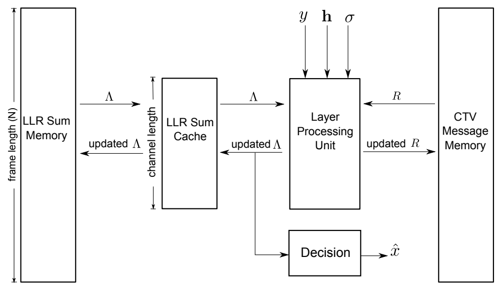

Figure 3. Architecture of the BP detector

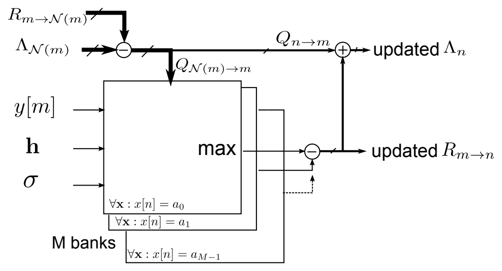
$R_{m\to\mathcal{N}(m)}$
$\Lambda_{\mathcal{N}(m)}$
$Q_{n\to m}$
updated Λ_n
$Q_{\mathcal{N}(m)\to m}$
$y[m]$
$\mathbf{h}$
σ
max
updated $R_{m\to n}$
$\forall\mathbf{x}: x[n]=a_0$
$\forall\mathbf{x}: x[n]=a_1$
M banks
$\forall\mathbf{x}: x[n]=a_{M-1}$

Figure 4. Architecture of layer processing unit

the LLR sum memory is $(M-1)N \times L_\Lambda$ where L_Λ is the word length of the accumulated LLR. Particularly if the number of iterations is reached, the updated Λ from the LPU is delivered to the decision block for decision and output.

In practice, the sparse channel may vary from time to time, in both values and locations of coefficients. The proposed BP detector can adapt to time-varying channels just by changing the coefficient input of LPU and write/read addresses of the cache according to the new $\mathbf{h}$.

V. Simulation and Implementation Results

The performance of the BP detector is evaluated by simulations in terms of symbol error rate (SER) versus signal-to-noise ratio per bit E_b/N_0. We consider a sparse channel with $L = 60$ and $D = 3$. The three nonzero coefficients are $h[0] = 1$, $h[24] = 0.5$ and $h[59] = 0.35$. The modulation is assumed to be QPSK. The BP detector processes $N = 1024$ symbols at a time, and outputs the estimations after $n_{it} = 5$ iterations. In Fig. 5, both floating and fixed point results are given. The performance of the generic linear equalizer is also provided for comparison. At a SER of 10^{-4}, the BP detector exhibits performance 4 dB better than the linear equalizer. The fixed point performance is 1dB away from the floating point result at a SER of 10^{-4} when the word lengths are chosen so $L_\Lambda = L_R = 8$, and about 2.5 dB away when $L_\Lambda = L_R = 7$.

Table I shows the implementation result on Xilinx Virtex 5 xc5vlx110t FPGA. The fixed point parameters are chosen as $L_\Lambda = L_R = 8$. The maximum speed is up to $f = 252$ MHz, corresponding to a throughput rate of $f \log_2 M/n_{it} = 100.8$Mb/s.

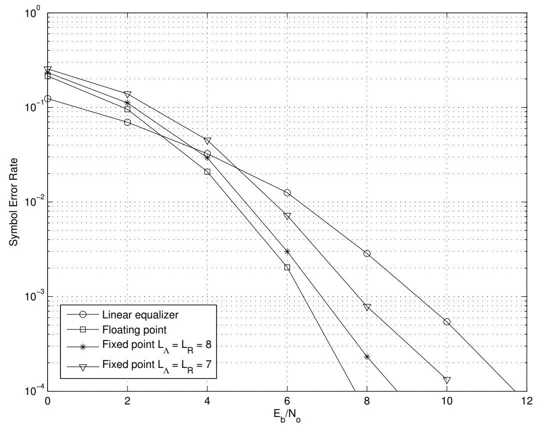

Figure 5. Simulation result

Number of Slice Registers	9660 out of 69120 (13%)
Number of Slice LUTs	17,886 out of 69120 (25%)
Number of Block RAM/FIFO	4 out of 148 (2%)
Number of DSP48Es	64 out of 64 (100%)
Maximum Speed	252 MHz

Table I
FPGA implementation result

References

[1] N. Benvenuto and R. Marchesani. The Viterbi algorithm for sparse channels. *IEEE Trans. Commun.*, 44(3):287–289, Mar. 1996.

[2] A. Duel-Hallen and C. Heegard. Delayed decision-feedback sequence estimation. *IEEE Trans. Commun.*, 37(5):428–436, 1989.

[3] D.E. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *Proc. IEEE Workshop Signal Processing and Systems*, pages 107–112, 2004.

[4] M. N. Kaynak, T. M. Duman, and E. M. Kurtas. Belief propagation over frequency selective fading channels. In *Proc. IEEE Veh. Technol. Conf.*, volume 2, pages 1367–1371, September 26–29, 2004.

[5] F.R. Kschischang and B.J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16(2):219–230, 1998.

[6] H.-A. Loeliger. An introduction to factor graphs. *IEEE Signal Process. Mag.*, 21(1):28–41, January 2004.

[7] N. C. McGinty, R. A. Kennedy, and P. Hocher. Parallel trellis Viterbi algorithm for sparse channels. *IEEE Commun. Lett.*, 2(5):143–145, May 1998.

[8] S. Roy, T. M. Duman, and V. K. McDonald. Error rate improvement in underwater mimo communications using sparse partial response equalization. *IEEE J. Ocean. Eng.*, 34(2):181–201, April 2009.