Partly Parallel Overlapped Sum-Product Decoder Architectures for Quasi-Cyclic LDPC Codes

Ning Chen, Yongmei Dai, and Zhiyuan Yan
Department of Electrical and Computer Engineering, Lehigh University, PA 18015, USA
E-mail: {nic6, yod304, yan}@lehigh.edu

***Abstract*— In this paper, we propose partly parallel architectures based on optimal overlapped sum-product (OSP) decoding. To ensure high throughput and hardware utilization efficiency, partly parallel parity check and pipelined access to memory are utilized. Impacts of different node update algorithms and quantization schemes are studied. FPGA implementation of our proposed architectures for a $(1536, 768)$ $(3, 6)$-regular QC LDPC code can achieve an estimated $61\,\mathrm{Mbps}$ decoding throughput at SNR$= 4.5\,\mathrm{dB}$. Finally, noncoherent OSP decoder, which does not always satisfy the data dependency constraints, is proposed to ensure that the maximum throughput gain 2 of the OSP decoding is achieved for *all* QC LDPC codes.**

***Index Terms*— low-density parity-check (LDPC) codes, quasi-cyclic (QC) codes, turbo decoding, sum-product decoding, FPGA.**

I. Introduction

Low-density parity-check (LDPC) codes have attracted much attention due to their spectacular error performance and inherent parallelism in their decoding algorithms. The latter is significant since it is conducive to high throughput implementations. Quasi-cyclic (QC) LDPC codes [1], [2] not only have relatively good error performance [1] but also are suitable for hardware implementation due to their regular structure: they can be encoded efficiently with shift registers [2] and can be decoded with partly parallel decoder architectures that require simpler address generation, less memory, and localized memory access [3], [4]. Thus, QC LDPC codes are of great interest and we focus on the decoding of QC LDPC codes in this paper.

An LDPC code can be represented by its Tanner graph, wherein the codeword bits and the parity check equations are represented as variable and check nodes respectively. The sum-product (SP) algorithm [5], sometimes referred to as the message passing algorithm, is an efficient iterative decoding algorithm for LDPC codes based on belief propagation. Each iteration of the SP algorithm has two sequential updates: check node updates (CNUs) and variable node updates (VNUs). Due to the data dependency between the VNUs and CNUs, partly parallel decoder architectures based on direct implementation of the SP algorithm either require twice as much memory or lead to low throughput and hardware utilization efficiency (HUE). Overlapped sum-product (OSP) decoder has been proposed for QC LDPC codes [4] to improve throughput and HUE. Optimal OSP decoder is proposed in [6] for regular QC LDPC codes, which can improve throughput and HUE by up to 100%. In this paper, we further study the OSP decoding of QC LDPC codes. The main contributions of this paper are:

- We utilize partly parallel parity checks (PPPCs) and pipelined memory access to ensure high throughput and HUE for our OSP decoder architectures;
- We evaluate the performances of the SP and the min-sum algorithms as well as different quantization schemes for our OSP decoder architectures;
- We implement our decoder architectures in FPGA and compare the implementation results with those of previously proposed decoder architectures;
- Noncoherent OSP decoder of QC LDPC codes is proposed, where data dependency is not always satisfied, so as to improve throughput and HUE.

The rest of the paper is organized as follows. In Section II, we briefly review QC LDPC codes and the SP algorithm and its variants. Section III presents our partly parallel OSP decoder architectures and evaluates the impacts of different design choices. In Section IV, we compare the FPGA implementation results of our decoder architectures with those of previously proposed decoder architectures. We also propose noncoherent OSP decoding of QC LDPC codes in Section V.

II. Background

A. Quasi-Cyclic LDPC Codes

A subclass of (j, k)-regular QC LDPC codes can be represented by a parity check matrix of the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{d_{0,0}} & \mathbf{I}_{d_{0,1}} & \cdots & \mathbf{I}_{d_{0,k-1}} \\ \mathbf{I}_{d_{1,0}} & \mathbf{I}_{d_{1,1}} & \cdots & \mathbf{I}_{d_{1,k-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{d_{j-1,0}} & \mathbf{I}_{d_{j-1},1} & \cdots & \mathbf{I}_{d_{j-1,k-1}} \end{bmatrix},$$

where $\mathbf{I}_{d_{s,t}}$ denotes an $m \times m$ identity matrix with all the rows cyclically shifted to the right by $d_{s,t}$ positions ($0 \le s \le j-1$, $0 \le t \le k-1$). The (j, k)-regular QC LDPC code defined by $\mathbf{H}$ has block length $k \cdot m$ and rate $\ge 1 - j/k$. Various ways to define the values of $d_{s,t}$ have been proposed [1], [3].

B. Sum-Product Decoding

Given the Tanner graph of an LDPC code, let $N(c)$ and $M(v)$ denote the sets of the variable and check nodes that are adjacent to the check node c and variable node v respectively. In CNUs of the i-th iteration, the check-to-variable message $R_{cv}^{(i)}$ from each check node c to any variable node $v \in N(c)$ is updated as follows

$$R_{cv}^{(i)} = - \prod_{n \in N(c)}^{n \neq v} \mathrm{sign}(L_{cn}^{(i-1)}) \psi \left(\sum_{n \in N(c)}^{n \neq v} \psi(L_{cn}^{(i-1)}) \right), \quad (1)$$

where $L_{cv}^{(i)}$ denotes the variable-to-check message from the variable node v to the check node c, and is updated in VNUs of the i-th iteration by $L_{cv}^{(i)} = -\frac{2r_v}{\sigma^2} + \sum_{m \in M(v)}^{m \neq c} R_{mv}^{(i)}$. Note that $\psi(x) = \ln \frac{e^x+1}{e^x-1}$, σ stands for the estimated standard deviation of the additive white Gaussian noise (AWGN) channel, and $2r_v/\sigma^2$ is the intrinsic information.

A well-known variant of the SP algorithm is the min-sum (MS) algorithm, where VNUs remain the same and the CNU in (1) is approximated as

$$R_{cv}^{(i)} = - \prod_{n \in N(c)}^{n \neq v} \text{sign}(L_{cn}^{(i-1)}) \min_{n \in N(c)}^{n \neq v} |L_{cn}^{(i-1)}|. \quad (2)$$

Obviously, the update in (2) involves less computation. It is also shown that the MS algorithm can outperform the SP algorithm when there are many short cycles [7].

Since $\psi(x)$ function is typically implemented using look-up tables, CNUs involve two look-up operations and are more complex than VNUs, often leading to low throughput in hardware implementation due to the unbalanced computation. A variant of the SP algorithm (see, for example, [8]), referred to as the *balanced SP algorithm* henceforth, redistributes the computation between CNUs and VNUs. CNUs and VNUs of the balanced SP algorithm are respectively given by

$$R_{cv}^{(i)} = - \prod_{n \in N(c)}^{n \neq v} \text{sign}(L_{cn}^{(i-1)}) \sum_{n \in N(c)}^{n \neq v} \psi(L_{cn}^{(i-1)}) \quad (3)$$

and $L_{cv}^{(i)} = -\frac{2r_v}{\sigma^2} + \sum_{m \in M(v)}^{m \neq c} \left[-\text{sign}(R_{mv}^{(i)}) \psi \left(R_{mv}^{(i)} \right) \right]$.

C. Overlapped Sum-Product

Overlapped sum-product (OSP) algorithm [4], where CNUs and VNUs are partially overlapped, improves both throughput and HUE while introducing no error performance loss. If dual-port memory is used, CNUs and VNUs can work on the same memory blocks so as to reduce the memory requirement. By choosing the addressing schemes of CNUs and VNUs properly, throughput of OSP decoder can be optimized [6]. For partly parallel decoder architectures, the optimal throughput gain of overlapping is bounded by 2, and whether the maximum throughput gain of 2 can be achieved depends on the given code itself [6].

III. Partly Parallel OSP Decoder Architectures

In this section, we propose partly parallel OSP decoder architectures based on the optimal OSP decoding scheme [6], and discuss various implementation issues.

A. Partly Parallel Architectures

Similar to the partly parallel architecture in [4], our partly parallel OSP architectures include j check node function units (CNFUs), k variable node function units (VNFUs), $j \times k$ message memory banks, k intrinsic information memory banks, parity check function units (PCFUs), and memory banks for hard decisions. CNFUs, VNFUs, and PCFUs are the hardware units that implement CNUs, VNUs, and parity checks, respectively. The numbers of PCFUs and hard decision memory banks depend on implementation, and are discussed below. Fig. 1 illustrates our OSP architectures for $(2, 3)$ QC LDPC codes. Each message memory bank consists of m units, indexed by row and column addresses [6]. The j CNFUs update the j block rows simultaneously, while the k VNFUs update the k block columns simultaneously. CNFU-s updates the m rows in the s-th block row of $\mathbf{H}$ sequentially, one row each step. VNFU-t updates the m columns of the t-th block column of $\mathbf{H}$ sequentially, one column each step.

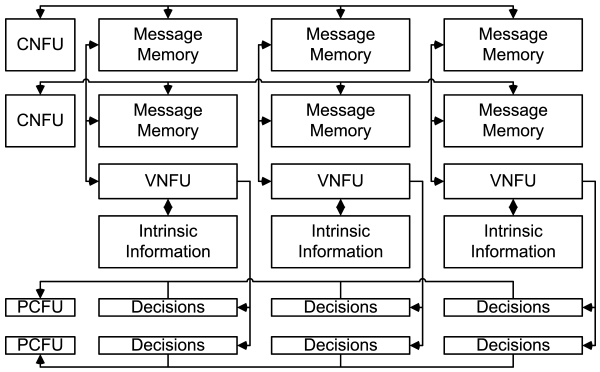

Fig. 1. OSP decoder architecture with PPPCs for (2,3) QC LDPC codes

B. PCFUs and Hard Decision Memory

The iterative decoding stops when either all parity checks are satisfied or the maximum iteration number is reached. During VNUs of the i-th iteration in the SP and MS algorithms, hard decision on the bit corresponding to the variable node v is made based on the soft output[1] $L_v^{(i)} = -\frac{2r_v}{\sigma^2} + \sum_{m \in M(v)} R_{mv}^{(i)}$. Fully parallel implementation of the jm parity checks achieves high throughput and small latency, but it leads to very high hardware cost (km bit registers to store the hard decisions and jm PCFUs) and routing congestion due to the typically large size of $\mathbf{H}$. This is also unnecessary due to limited throughput achieved by partly parallel decoders. It is natural to utilize partly parallel parity checks with j PCFUs so as to match throughput of CNFUs and VNFUs. This enables us not only to reduce the number of PCFUs, but also to use memory to store the hard decisions. Similar to the CNFUs, the j PCFUs check the parity of the j block rows simultaneously and PCFU-s check the parity check equations in the s-th block row sequentially. Clearly, parity checks run concurrently[2] with CNUs because when VNUs generate the messages ($L_{cv}^{(i)}$s) for CNUs, they also generate soft outputs ($L_v^{(i)}$s) and the hard decisions needed by parity checks.

The amount of hard decision memory depends on implementation. A straightforward implementation needs k memory banks, each of m bits, to store the hard decisions. Each memory bank corresponds to one block column and the hard decisions come from the VNFU for this block column. If each parity check takes one clock cycle, since each memory bank is accessed by all j PCFUs and this VNFU simultaneously, it

[1] A minor modification is necessary for the balanced SP algorithm.

[2] This possibility of concurrency was mentioned in [4].

is necessary to use $(j+1)$-port memory, which is generally unavailable in RAM compilers supplied by standard cell library vendors. An alternative solution is to use j dual-port memory banks, one for each PCFU, instead of one $(j+1)$-port memory bank. The VNFU writes the decisions into all j banks, and each bank is accessed by only one PCFU.

In fact, CNUs and parity checks may be combined by merging the PCFU and CNFU as well as the k dual-port memory banks for messages and k dual-port memory banks for hard decisions corresponding to the same block row. The advantage of this approach is that only one set of addressing mechanisms is shared for both CNUs and parity checks. However, this approach is not adopted in our decoder architectures since it does not allow the improvements described in Section III-D. Since PPPCs perform parity check concurrently with CNUs, it is checking the decisions from VNUs of the previous iteration. Thus the latency introduced by PPPCs is $m - w_{cv}$, where w_{cv} is the intra-iteration waiting time in the optimal OSP [6].

C. Pipelined Access to Message Memory

In FPGA and RAM compilers for ASIC with deep submicron technology, only synchronous static memory is available [9]. A drawback of typical synchronous memory is that one or two waiting clock cycles are needed in every read operation. We assume exactly one waiting clock cycle is added for each read operation, but our approach can be easily extended to synchronous memory that requires two clock cycles. To fully exploit the bus of message memory and achieve high throughput, we consider two different pipelined access schemes: one is to let the functional units read message for the next step during the waiting clock cycle, while the other is to let VNFUs read during the waiting clock cycle of CNFUs. They will be referred to as *intra-unit* and *inter-unit* pipelined memory access schemes respectively.

Intra-Unit Pipelined Access: For both CNFUs and VNFUs, two adjacent update steps can be pipelined as in Fig. 2. In comparison to non-pipelined access, the throughput gain of the intra-unit pipelined access is $6/4 = 1.5$. Dual-port memory is still needed in this case. However, to facilitate this pipelined access, slightly more complicated addressing mechanisms than those in [6] are needed for CNFUs and VNFUs.

Inter-Unit Pipelined Access: The scheduling for inter-unit pipelined message access is depicted in Fig. 3. In the OSP architecture proposed in [4], dual-port memory is required. With inter-unit pipelined message access, they can be replaced by single-port memory to reduce area and routing congestion at the expense of throughput. However, CNFUs and VNFUs need to share the read/write and address ports by simply adding some multiplexers. The scheduling scheme for inter-unit pipelined message access remains the same as in [6]. Throughput of the inter-unit pipelined access is one half of that of the intra-unit pipelined access due to idle clock cycles.

D. Folded Operations

As in the pipelined memory access schemes, each step of VNFUs and CNFUs may need multiple clock cycles. This enables us to fold operations, leading to hardware savings. If each step of CNFUs and VNFUs takes p clock cycles, the LUTs for $\psi(x)$ function and q-bit adders (q is the wordlength of messages) in CNFUs and VNFUs can be folded by a factor of p. Each VNFU requires only $\lceil j/p \rceil$ LUTs and $\lceil j/p \rceil$ q-bit adders while each CNFU requires $\lceil k/p \rceil$ LUTs and $\lceil k/p \rceil$ q-bit adders. For both LUTs and adders, the total number required is reduced from $k \times j + j \times k$ to $\lceil k/p \rceil \times j + \lceil j/p \rceil \times k$ by folding. But at the same time, the folded operations require more registers to store the intermediate results (one extra q-bit register is needed to save one LUT) and involve more complicated control. Folding also reduces the critical path. Folding can be combined with the intra-unit pipelined access as depicted in Fig. 4. This combined approach, referred to as the *combined access* henceforth, not only keeps the high throughput and memory access efficiency from the intra-unit pipelined access, but also reduces hardware by folding.

CNFU	READ 2n-1th	READ 2nth	WRITE 2n-1th	WRITE 2nth
VNFU	READ 2p-1th	READ 2pth	WRITE 2p-1th	WRITE 2pth

Fig. 2. Intra-unit pipelined message access scheme

CNFU	READ	WAIT	WRITE	IDLE
VNFU	IDLE	READ	WAIT	WRITE

Fig. 3. Inter-unit pipelined message access scheme

CNFU	READ nth LOOK-UP and ADD n-1th	WRITE n-1th	READ n+1th LOOK-UP and ADD nth	WRITE nth
VNFU	READ mth LOOK-UP and ADD m-1th	WRITE m-1th	READ m+1th LOOK-UP and ADD mth	WRITE mth

Fig. 4. The combined access with $p = 2$

PPPCs can also take advantage to share hard decisions and PCFUs to save area. By accessing different addresses of hard decision memory, one PCFU can perform p different parity checks in parallel with each step of CNFUs(VNFUs). It reduces the numbers of required PCFUs and dual-port memory banks from j and jk to $\lceil j/p \rceil$ and $\lceil jk/p \rceil$ respectively.

E. Message Quantization

The quantization scheme affects not only area and power consumption of decoder architectures but also error performance due to the finite precision effect. We assume the messages and intrinsic information have the same wordlength, and consider two types — uniform and nonuniform — of quantization schemes.

For uniform quantization schemes, there is a tradeoff between dynamic range and quantization step. In [10], the 8-bit uniform quantization scheme uses a quantization step 0.125 so as to obtain a large dynamic range up to 16. This is achieved by dividing eight quantization bits into one sign bit, four integer bits, and three fraction bits. We can also divide them into one sign bit, three integer bits, and four fraction bits. We refer to these quantization schemes as 843bu and 834bu schemes respectively henceforth.

Nonuniform quantization method [11] can provide a different tradeoff between dynamic range and quantization step.

In q-bit quantization, a message is denoted as $v_q v_{q-1} \dots v_1$, where v_q is the sign and v_{q-1} is used as a flag. If the magnitude of the message value is less than 1, then v_{q-1} is set to 0 and the decimal point lies between v_{q-1} and v_{q-2}. When $q = 8$, in the fraction range the quantization step is $1/64 = 0.0156$, smaller than those of the two uniform quantization schemes. If the magnitude of the message is at least 1, v_{q-1} is set to 1 and the decimal point moves to a proper position for desired dynamic range. In our implementation, the decimal point lies between v_{q-4} and v_{q-5}, and the message value of integer part is interpreted as

$$v_{q-1} \cdot \bar{v}_{q-2} \cdot \bar{v}_{q-3} \cdot \bar{v}_{q-4} \cdot 2^3 + v_{q-2} \cdot 2^2 + v_{q-3} \cdot 2 + v_{q-4},$$

where $\bar{v}_i$ represents the bitwise complement of v_i. When $q = 8$, the dynamic range is roughly the same as that of the uniform scheme with 3 integer bits and 4 fraction bits. Though the quantization step is bigger when the message value is greater than 1, it does not matter because $\psi(x)$ function will be less sensitive to the quantization step when the message value is greater. We also experimented with an nonuniform quantization scheme where the decimal point is between v_{q-5} and v_{q-6} when the message value is greater than 1. For $q = 8$, we refer to these two schemes as 833bnu and 842bnu schemes respectively below. The main disadvantage of nonuniform quantization scheme is that it requires more complicated adders than the uniform scheme, leading to bigger area and larger critical path delay as seen below in Section IV.

The BER performance and average iteration numbers of the SP and MS algorithms using different 8-bit uniform and nonuniform quantization schemes are compared with simulation results of full floating point precision in Figs. 5 and 6 respectively. In our simulations, and the maximum iteration number is set to 20. From Figs. 5 and 6, we observe that between the two uniform quantization schemes, 834bu scheme for the SP algorithm and 843bu scheme for the MS algorithm have slightly better error performance in high SNR range and hence they are adopted in our implementations in Section IV. For nonuniform schemes, 833bnu has slightly better performance than 842bnu, and hence is adopted in our implementations in Section IV. In the simulation of the SP algorithm, 834bu scheme has slightly better BER performance than 833bnu and 842bnu schemes. The MS algorithm using uniform quantization has better BER performance than the others in high SNR range, including those with full floating point precision (This was also observed in [12]). On the convergence rate, all simulations are fairly close when SNR is high. We remark that for four curves in Fig. 5, we do not give bit error rates at 4.5dB. Although we are certain bit error rates for these points are all below 10^{-7}, not enough frames are simulated to report reliable bit error rates.

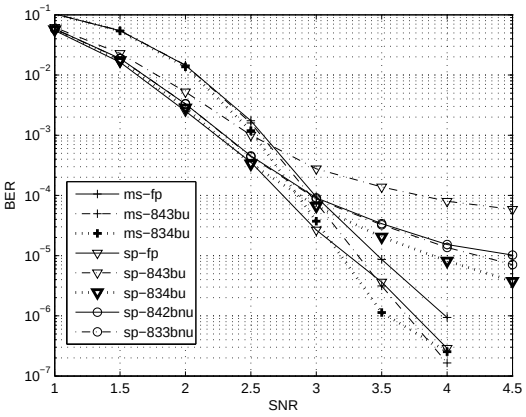

Fig. 5. BER performance of a $(1536, 768)$ $(3, 6)$-regular code

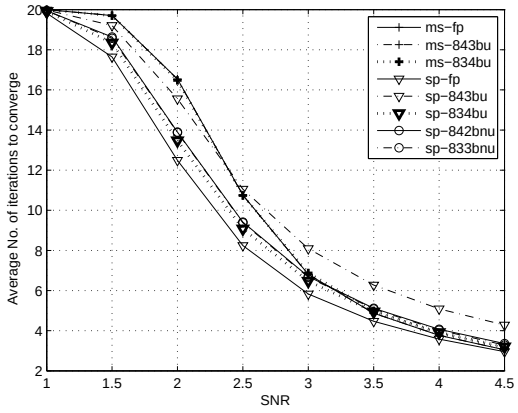

Fig. 6. Average iteration number of a $(1536, 768)$ $(3, 6)$-regular code

IV. FPGA Implementation Results

We implement five different partly parallel OSP decoder architectures proposed above — one based on the MS algorithm and four based on the balanced SP algorithm (using $\psi(x)$) — using a Xilinx Vertex II xc2v6000-5ff1152 FPGA, which has 33792 slices and 144 block RAMs (each block RAM is 18 Kbits). The synthesis and implementation are carried out using Xilinx ISE Version 8.1. Our decoder architecture based on the MS algorithm uses the intra-unit pipelined message access, whereas our four decoder architectures based on the balanced SP algorithm use the intra-unit pipelined access and the combined access respectively. Among the four architectures implementing $\psi(x)$ function, two are using uniform quantization scheme and the other two are using nonuniform quantization scheme. Note that PPPCs are used in all five of our decoder architectures. In Table I, we specify the numbers of occupied slices, flip flops, 4-input LUTs, and block RAMs as well as critical path delays, average iteration numbers, and estimated decoding throughput of our decoder architectures. The first two rows of Table I, reproduced from [10, Table I], itemize implementation details of the two decoder architectures in [10] using *the same FPGA*. These two architectures, referred to as the *MS* and *HEMS* architectures respectively henceforth, are based on the MS algorithm and the high-efficiency min-sum algorithm proposed in [10] respectively. The HEMS algorithm in [10], a variant of the MS algorithm, uses extra VNUs so as to achieve better message passing, leading to faster convergence rate. The estimated decoding throughputs for all five decoder architectures are computed based on *the same assumptions* made in [10]: The input

TABLE I

FPGA IMPLEMENTATION RESULTS OF A $(1536, 768)$ $(3, 6)$-REGULAR CODE (THE FIRST TWO ROWS ARE REPRODUCED FROM [10, TABLE I])

Architecture	Quantization	Slice	F/F	4-input LUT		Block	Critical Path	Average Iteration	Throughput
				logic	ROM	RAM	Delay (ns)	Number at $4.5\,\mathrm{dB}$	at $4.5\,\mathrm{dB}$ (Mbps)
MS [10]	843bu	1144	1089	1705	0	102	9.894	3.4	23
HEMS [10]	843bu	1296	1358	1785	0	102	9.900	2.0	38
intra-unit OSP MS	843bu	1167	664	2066	0	36	10.893	3.0	61.20
intra-unit $\psi(x)$	834bu	2166	778	1885	2016	36	11.898	3.2	53.65
combinational $\psi(x)$	834bu	1919	1141	2227	1176	36	12.645	3.2	50.48
intra-unit $\psi(x)$	833bnu	2808	1240	2892	2016	36	12.426	3.3	51.04
combinational $\psi(x)$	833bnu	2914	1793	3379	1176	36	15.885	3.3	39.35

interface can get intrinsic information by groups of k nodes from different block columns, and the clock frequency is the inverse of the critical path delay. The decoding throughput of our five decoder architectures is given by

$$\text{Throughput} = \frac{k \times m \times f}{m + 2 \times m \times (l + 1)}, \tag{4}$$

where f is the clock frequency and l is the average number of iterations. The difference between Eq. (4) and [10, (8)] is due to the different scheduling schemes.

Comparing all three MS-based decoder architectures itemized in the first three rows of Table I, it is clear that our OSP decoder architecture requires *less hardware* and achieves a *higher throughput*. Although our OSP decoder architecture based on the MS algorithm requires slightly more combinational logic than the two architectures in [10], the number of flip-flops is reduced to 61% of the MS architecture and 49% of the HEMS architecture, and the overall number of slices is less than the HEMS architecture and roughly the same as the MS architecture. Furthermore, our decoder architecture requires 65% fewer block RAMs than those in [10]. Memory efficiency of our decoder architecture will enable implementation on small FPGAs and simplify the placement and routing in ASIC implementation. As shown in Table I, the critical path delay of our decoder architecture based on the MS algorithm is close to those of the MS and HEMS architectures. More than double the throughput of the MS architecture, the throughput of our OSP decoder architecture based on the MS algorithm is 61% higher than that of the HEMS architecture despite fewer iterations required by the latter.

Comparisons among our five decoder architectures illustrate the advantage of the MS algorithm over the SP algorithm involving $\psi(x)$. We note that the extra combinational logic and ROMs required by our decoder architectures using $\psi(x)$ are for the implementation of LUTs and adders, which are not needed by our decoder architecture based on the MS algorithm. That is also why our decoder architecture based on the MS algorithm has a shorter critical path delay. We note that the significant difference in the number of flip-flops is mostly due to duplicate registers to reduce fan-out and routing congestion for shorter critical path delay. As shown in Figure 5, the MS algorithm is also less vulnerable to the effects of finite precision implementation, and has better BER performance in high SNR range than the SP algorithm. Our decoder architectures using the combined access require about half as much ROM as their counterparts using the intra-unit scheme, but they have longer critical path delays and use more combinatorial logic and flip-flops. Also, our decoder architectures with nonuniform quantization scheme are more complex, require more area, and have longer critical path delays than their counterparts with uniform schemes.

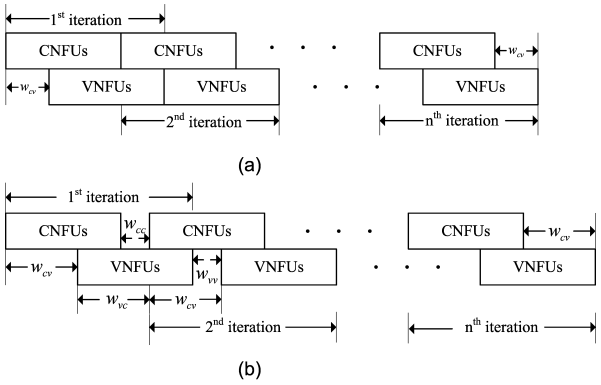

Fig. 7. Scheduling scheme of optimal OSP decoder

V. NONCOHERENT OSP DECODER

Depending on a quantity w_{cv}, which is determined by the parity check matrix of a QC LDPC code, the scheduling scheme for the optimal OSP decoder falls into two scenarios [6] shown in Figs. 7(a) and 7(b) respectively. Under the first scenario where $w_{cv} > \lceil \frac{m}{2} \rceil$, after finishing CNUs (VNUs) of any iteration, CNFUs (VNFUs) have to wait $w_{cc} = w_{vv} = 2w_{cv} - m$ cycles before they can start CNUs (VNUs) of the next iteration. In this case, the HUE $\frac{m}{w_{cc}+m}$ is clearly smaller than 1 and the throughput gain of the OSP decoder [6] is less than 2. Under the second scenario that corresponds to $w_{cv} \leq \lceil \frac{m}{2} \rceil$, $w_{cc} = w_{vv} = 0$, after finishing CNUs (VNUs) of any iteration CNFUs (VNFUs) can start CNUs (VNUs) of the next iteration right away. Thus, the HUE is 100%, and the maximum throughput gain of $G \approx 2$ is indeed achieved.

Note that w_{cv} and hence the throughput gain as well as HUE of the optimal OSP decoder in [6] is determined by the given QC LDPC code itself. For LDPC codes with $w_{cv} > \lceil \frac{m}{2} \rceil$, we propose noncoherent OSP (NOSP) decoder so as to achieve high throughput and HUE. The basic idea is to utilize a scheduling scheme as in Fig. 7(a) with $w_{cv} = 1$ for these codes. Since the starting addresses for CNFUs and VNFUs need to be increased by $w_{cv} - 1$ for each iteration [6], setting the w_{cv} to 1 implies that the starting addresses for

CNFUs and VNFUs are the same for each iteration, thereby simplifying control mechanisms. This scheduling scheme is called noncoherent OSP decoding since data dependencies of node updates are no longer guaranteed. That is, some of the variable-to-check messages used in (1) (or (2)) are from the $(i-1)$-th iteration and the rest are from the $(i-2)$-th iteration. Similarly, some of the check-to-variable messages used to compute $L_{cn}^{(i)}$ are from the i-th iteration and the rest are from the $(i-1)$-th iteration.

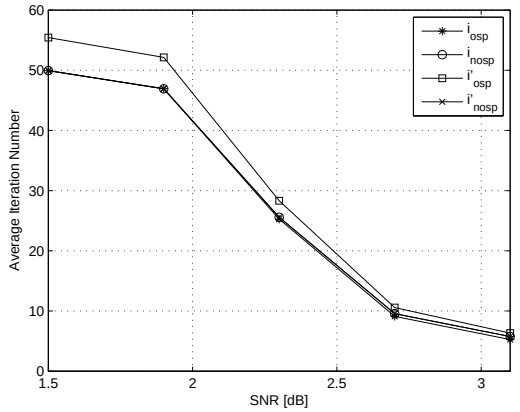

Fig. 8. Average iteration number of a $(5, 13)$-regular QC LDPC code (maximum iteration number is 50)

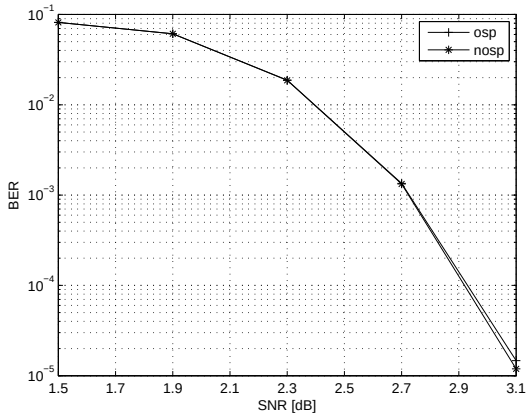

Fig. 9. BER performance of a $(5, 13)$-regular QC LDPC code (maximum iteration number is 50)

NOSP decoder eliminates the idle clock cycles of CNFUs and VNFUs in the OSP decoder of these codes, which obviously improves HUE to 100%. By setting w_{cv} to 1, NOSP decoder also reduces the number of clock cycles used to decode a block slightly. These two factors, however, do not necessarily improve throughput gain since NOSP decoding may need more iterations. To account for all the factors, we define an identity called *effective iteration number* that is inversely proportional to the decoding throughput[3]. Suppose it takes the (coherent or noncoherent) OSP decoder i iterations to converge, then the effective iteration number i' is defined to be

$$i' = i(1 + \frac{w_{cc}}{m}) + \frac{w_{cv}}{m}.$$

For a $(5, 13)$-regular QC LDPC code [1, Table I] with $w_{cv} = 72$, we compare the average of both iteration numbers and effective iteration numbers by coherent and noncoherent OSP decoders ($w_{cv} = 1$) in Fig. 8. For this code, NOSP decoding improves decoding throughput under any SNR by about 10% (see, or example, $i'_{\text{osp}} = 6.32$ and $i'_{\text{nosp}} = 5.76$ at SNR$= 3.1$ dB). Also, the BER performances of coherent and noncoherent OSP decoders for this code, shown in Fig. 9, are very close. Our simulation results with other codes suggest that NOSP decoder improves throughput for all the codes with $w_{cv} > m/2$ with negligible difference in error performance.

The architectural difference between coherent and noncoherent OSP decoders is their scheduling schemes, including starting addresses and address generation, which are easily configurable in our architectures. Hence, our decoder architectures described above for coherent OSP decoder can be readily adapted to noncoherent OSP decoder. In fact, since the starting addresses of noncoherent OSP decoder are the same for each iteration, noncoherent OSP decoder has simpler control than coherent OSP decoder, leading to reduced hardware cost.

[3]The latency associated with parity checks as described in Section III-B is not considered for simplicity.

VI. Acknowledgment

We thank the authors of [10] for providing details of the $(3, 6)$-regular QC LDPC code used in [10].

References

[1] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja and D. J. Costello, "LDPC Block and Convolutional Codes Based on Circulant Matrices," *IEEE Trans. Info. Theory*, vol. 50, pp. 2966–2984, Dec. 2004.

[2] M. P. C. Fossorier, "Quasi-Cyclic Low-Density Parity-Check Codes From Circulant Permutation Matrices," *IEEE Trans. Info. Theory*, vol. 50, pp. 1788–1793, Aug. 2004.

[3] M. M. Mansour and N. R. Shanbhag, "Low Power VLSI Decoder Architecture for LDPC Codes," *in Proc. IEEE Int. Symp. on Low Power Electronics and Design (ISLPED)*, pp. 284–289, Aug. 2002.

[4] Y. Chen and K. K. Parhi, "Overlapped Message Passing for Quasi-Cyclic Low Density Parity Check Codes," *IEEE Trans. Circuits and Systems*, vol. 51, pp. 1106–1113, June 2004.

[5] F. R. Kschischang, B. J. Frey and H. A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Info. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[6] Y. Dai and Z. Yan, "Coset-Based Quasi-Cyclic LDPC Codes for Optimal Overlapped Message Passing Decoding," *in Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Nov. 2005.

[7] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier and X. Hu, "Reduced-Compelxity Decoding of LDPC Codes," *IEEE Trans. Commun.*, vol. 53, pp. 1288–1299, Aug. 2005.

[8] Z. Wang, Y. Chen and K. K. Parhi, "Area Efficient Decoding of Qausi-Cyclic Low Density Parity Check Codes," *in Proc. IEEE Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. V49–V52, May 2004.

[9] Xilinx, "Xilinx libraries guide," 2005.

[10] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga and S. Goto, "Partially-Parallel LDPC Decoder Based on High-Efficiency Message-Passing Algorithm," *in Proc. Int. Conference on Computer Design (ICCD)*, pp. 503–510, Oct. 2005.

[11] T. Zhang, Z. Wang and K. K. Parhi, "On Finite Precision Implementation of Low Density Parity Check Codes," *in Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, vol. 4, pp. 202–205, 2001.

[12] J. Zhao, F. Zarkeshvari and A. H. Banihashemi, "On Implementation of Min-Sum Algorithm and Its Modifications for Decoding Low-Density Parity-Check (LDPC) Codes," *IEEE Trans. Commun.*, vol. 53, pp. 549–554, April 2005.