ELectronics EXpress

IEICE Electronics Express, Vol.9, No.12, 1044–1050

Flexible and high-efficiency turbo product code decoder design

Li Zhou[a)], **Hengzhu Liu, and Botao Zhang**

Computer School, National University of Defense Technology, Changsha, China

a) *zhouli06@nudt.edu.cn*

Abstract: This paper presents a flexible and high-efficiency decoder for turbo product code using extended Hamming code. The supported component code ranges from $(8, 4)$ to $(128, 120)$ to provide enough flexibility for various communication standards. A novel Chase decoder architecture is developed with high efficiency using a low complexity algorithm. Moreover, a conflict free interleave memory access model for variable length is provided. A 90 nm standard cell technology shows that the decoder sustains a maximum throughput of 5.6 Gbps and consumes 300 k gates.

Keywords: turbo product code, high efficiency VLSI, flexibility decoder

Classification: Integrated circuits

References

[1] K. Gracie and M. H. Hamon, "Turbo and Turbo-Like Codes: Principles and Applications in Telecommunications," *Proc. IEEE*, vol. 95, pp. 1228–1254, 2007.

[2] S. A. Hirst, B. Honary, and G. Markarian, "Fast Chase algorithm with an application in turbo decoding," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1693–1699, Oct. 2001.

[3] C. Argon and W. McLaughlin, "An efficient Chase decoder for turbo product codes," *IEEE Trans. Commun.*, vol. 52, no. 6, pp. 896–898, June 2004.

[4] J. Cuevas, P. Adde, and S. Kerouedan, "Turbo decoding of product codes for Gigabit per second applications and beyond," *European Transactions on Telecommunications*, vol. 17, pp. 45–55, Jan.-Feb. 2006.

[5] C. Jego, P. Adde, and C. Leroux, "Full-parallel architecture for turbo decoding of product codes," *Electronics Letters*, vol. 42, pp. 1052–1053, 2006.

[6] C. Leroux, C. Jego, P. Adde, D. Gupta, and M. Jezequel, "Turbo Product Code Decoder Without Interleaving Resource: From Parallelism Exploration to High Efficiency Architecture," *J. Signal Processing Systems for Signal Image and Video Technology*, vol. 64, pp. 17–29, July 2011.

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

1 Introduction

Turbo product code (TPC) performs close to the Shannon limit and has lower complexity than turbo convolutional code [1]. TPC easily achieves throughput of gigabit per second by decoding rows and columns in parallel. Numerous wireless and optical communication protocols have adopted TPC, but their code length varies. For example, TPC in IEEE 802.16 chose the $(16, 11)$, $(32, 26)$, or $(64, 57)$ extended Hamming code as component code in both row and column dimensions. Thus, the TPC decoder urgently needs high throughput and flexibility characteristics for future high data rate communication systems.

A TPC decoder is composed of soft input soft output (SISO) decoder and interleaving resource. Most TPC decoders use Chase algorithm for SISO decoding of block code. Register file, memory, and connection network are candidates for turbo code interleaving resource. Numerous implementations have been proposed, but few have achieved high efficiency while supporting flexibility. In this paper, a high-efficiency Chase decoder for TPC is designed using a low complexity algorithm. Moreover, we propose the conflict free memory access model of using memory as interleaving resource, and support variable code length in our design. We implement the TPC decoder using several Chase decoders and interleaving memory on Chartered 90 nm CMOS technology.

2 TPC decoder design

2.1 Low complexity Chase decoding algorithm

The Chase algorithm of block code includes three steps: least reliable sorting, test vector decoding, and soft output. In the least reliable sorting, input $R = \{r_0, r_1 \dots r_{n-1}\}$ is sorted to determine the p least reliable value in R, and a hard decision is defined by $Y = sign(R)$. Then 2^p test vectors are generated according to p and Y, which traverse all possible codes on p locations. After algebraic decoding of test vectors, a valid code set is generated, where $D = \{d_0, d_1 \dots d_{n-1}\}$ is the code with minimal Euclidean distance to R. In the soft output, the decoder seeks a candidate code C in the valid code set with a minimal distance to R and $d_i \neq c_i$ for every position i. The soft value is then calculated by C and D.

The most complex part of the original Chase algorithm lies in the following procedures. First, decoding each test vector T_i requires syndrome calculation $S_i = T_i \oplus H$. Second, Euclidean distance computing is difficult to implement in hardware, so a simplified expression should be devised. Third, sorting in a candidate code search is necessary for each position i. The complexity of these three tasks grows with the code length n, causing low-efficiency Chase decoding in the case of variable code length. Several modifications have been proposed to address these issues. In [2], test vector decoding is done in Gray code order, which reduces its complexity to 1 vector-matrix multiplication plus $2^p - 1$ vector-vector multiplications in GF(2). In [3], dot product is applied instead of Euclidean distance. Only different bits

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

Algorithm 1: Low-Complexity Chase Algorithm

Input: Received code $R:\{r_0, r_1 ... r_{n-1}\}$; Number of least reliable position: p; Parity check matrix : H

Output: Updated soft information for next iteration $\Lambda:\{\lambda_0, \lambda_1, ... \lambda_{n-1}\}$

1 Find p locations $L:\{l_1, l_2, ... l_p\}$, where $|r_i|, i \in L$ is the p least reliable value in R;
2 Determine hard decode $Y \leftarrow sign(R)$; Test vector $T_0 \leftarrow Y$; syndrome $S_0 \leftarrow Y \oplus H$; valid code set $V \leftarrow \phi$;
3 **for** $j := 0$ to $2^p - 1$
4 Generate a valid code C_j according to T_j and S_j;
5 $V \leftarrow V \cup \{C_j\}$;
6 Compute dot product of C_j relative to Y: $M_j = R \cdot Y - R \cdot C_j$;
7 Invert the i^{th} bit of $T_j, i \in L$, such that a new test vector T_{j+1} forms;
8 $S_{j+1} \leftarrow S_j \oplus h_i$, h_i is the i^{th} column of H;
9 **end for**
10 $D \leftarrow C_t$ where $C_t \in V$ and $M_t = \min_i M_i$
11 **for** $k := 0$ to $n - 1$
12 **if** $k \in L$
13 Find candidate code C_q in V, whose M_q is minimal and whose k th bit is different from d_k;
14 $\lambda_k \leftarrow (2d_k - 1)(M_q - M_t) - r_k$;
15 **else**
16 $\lambda_k \leftarrow (2d_k - 1)(\max M_i - M_t) / p$;
17 **end if**
18 **end for**
19 $\Lambda \leftarrow 0.5\Lambda + R$;

between a valid code and Y contribute to the metric.

For the least reliable position i, there are at least two test vectors with an ith bit of 1 if $p \geq 2$. They cannot be corrected both at the same time; thus $c_i = 1$ exists in the valid code set. Similarly, $c_i = 0$ also exists in the valid code set, so we can always find candidate codes for least reliable positions. The extended Hamming code only corrects one error; thus, there are at most $p + 2$ different bits and at least $p - 2$ between a valid code and Y. Therefore, extrinsic information differs in the same range. Averaging p is suitable for estimating extrinsic information when using maximum metric. This approach is useful because searching candidate codes is not necessary for all bits; only for the p least positions. This premise is true for all code lengths.

Algorithm 1 is the modified algorithm with less complexity than the original one and much less dependency on code length. Section 3 shows that the coding gain of low-complexity algorithm only slightly declines, or becomes even better in certain code lengths.

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

2.2 High-efficiency Chase decoder architecture

Figure 1 shows the proposed Chase decoder architecture, which takes input r_i and generates output λ_i serially. Three modules make up the algorithm that implements three modules: least reliable sorting (lines 1–2); test vector decoding (lines 3–9); and soft output (lines 10–19). Latency of the Chase decoder includes input time n cycles and test vector decoding time 2^p cycles. Output also takes n cycles. The complexity which varies with n is hidden by the input and output latency; only bit width of registers and combination logic are relevant.

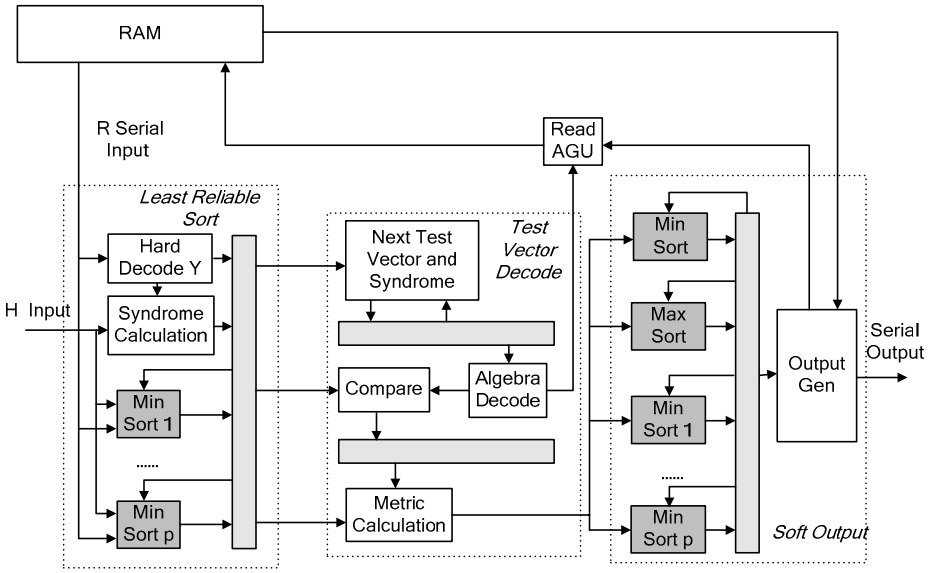

Fig. 1. Chase decoder architecture

Least reliable sorting module takes r_i in from RAM, each with read address increment 1. RAM will be read later by the other two modules. Sorting occurs simultaneously with r_i input. Hard decode Y with its syndrome, p locations with r_i, and corresponding p columns in H are obtained at the end of input, and then stored in registers.

Test vector decoding module is divided into three stages in the pipeline. The first stage inverts a bit of T_j in each cycle to generate a new test vector and syndrome. Then, algebraic decoding corrects at most 1 bit of test vector according to the syndrome. If necessary, RAM read request is sent to read address generation unit (AGU) for access to r_i at the corrected position. The decoding result is compared with Y, and differences are recorded in stage 2. Stage 3 uses the comparison to compute relative dot product M by adding r_i or $-r_i$ where different bits lie.

In the soft output module, sorting is performed once M_j is prepared in the test vector module. There are $p+1$ minimal sort logics for D and unreliable bits, as well as 1 maximum sort logic for reliable bits. When the test vector decoding is complete, minimal and maximal value in registers are ready for soft output generation. Thus, we can use r_i in RAM and then compute for λ_i immediately.

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

2.3 Conflict-free memory access model

In Turbo code, interleaving resource is necessary to rearrange data in a particular order between iterations. Usually, three kinds of hardware are adopted: register, memory, and connection network. Register file is the simplest way, but hardware overhead caused by large code matrix is intolerable. The $(128, 120)^2$ code needs $8 \times 128 \times 128$ bit storages if symbols are quantized by 8 bits. Some studies [4, 5] have addressed issues of using connection networks, such as the Omega network. However, connection networks lack the capability of supporting variable length. Suppose the SISO decoder contains eight Chase decoders in TPC, so that eight rows or columns can be processed in parallel. While decoding $(8, 4)^2$ code matrix, the output of row SISO can be delivered through interleaving connection network to column SISO directly in a delicate manner and utilized without latency. However, in the case of $(16, 11)^2$ code, two subprocedures consisting of eight rows each are needed for row SISO. Column SISO has to be divided as well. The rows are separated, and thus, output of row SISO cannot ensure continuity to form a whole column. So, the connection network cannot rearrange data into a column order without storage. In our design, we use memory as interleaving resource. Figure 2 shows a full iteration of TPC.

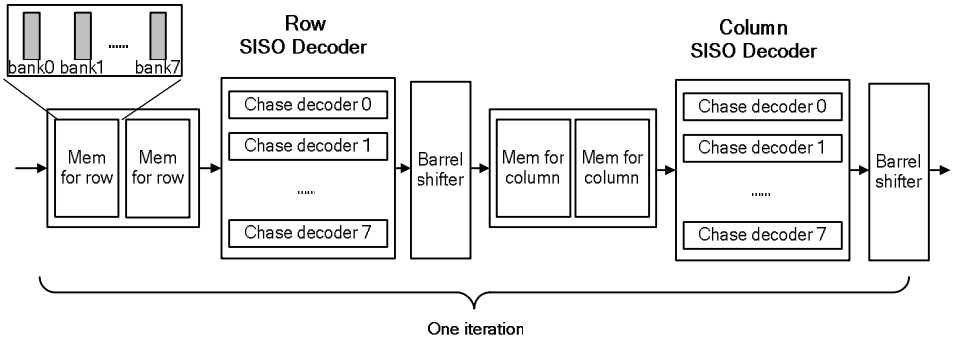

Fig. 2. A full iteration of TPC decoder

SISO decoder consists of eight Chase decoders. Two memory blocks alternately read and write. Each memory block contains eight single port 8 bit widths, 2 k depth RAM 0-7, which provide data to Chase decoder 0–7. Conflict-free memory access concerns data location and rearrangement issues. TPC matrix is denoted by $[D_{x,y}]$ in our design, and $0 \le x \le 2^{q_1} - 1$, $0 \le y \le 2^{q_2} - 1$, $q_1, q_2 \in \{3, 4, 5, 6, 7\}$. Data $D_{x,y}$ is located at $A_{x,y} = (k, l)$, where k is bank index and l is internal address in the bank. More than one subprocedure is needed if $q_1 > 3$ or $q_2 > 3$. For example, when the TPC matrix has 16 rows, row SISO decoder first processes rows 0–7, and then processes rows 8–15. In the memory for row, we place $D_{x,y}$ at $A_{x,y} = (x \bmod 8, y + \lfloor x/8 \rfloor \times 2^{q_2})$. Any two data are stored in the same bank only if they are within the same row or belong to two subprocedures. In the memory for column, a similar scheme is applied to avoid reading conflict, where $A_{x,y} = (y \bmod 8, x + \lfloor y/8 \rfloor \times 2^{q_1})$. This address scheme ensures conflict-free memory reading, because any two

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

data in a bank would not be read at the same time.

We solve data rearrangement issues and avoid memory write conflicts in a way inspired by data interleaving with connection network. Eight outputs of row SISO are to be written in memory for column according to the address described above. Write conflict is avoided by the ith Chase decoder generating output data $\Lambda : \{\lambda_0, \lambda_1, \ldots \lambda_{n-1}\}$ in the order of $output_0 = \lambda_i$, $output_{t+1} = output_t + 1 \bmod n$. For example, Chase decoder 0's output order is $\{D_{0,0}, D_{0,1}, \ldots\}$, Chase decoder 1's output order is $\{D_{1,1}, D_{1,2}, \ldots\}$. This setup makes the eight outputs of row SISO different from each other in the sense of mod 8; thus, they will be stored in different banks. Barrel shifter is inserted to direct data to the bank where it belongs. The output order of Column SISO rearranges data similarly.

3 Result and comparison

The proposed TPC decoder is implemented in HDL and synthesized on Chartered 90 nm technology by Synopsys Design Compiler. The whole decoder has four duplications of architecture in Fig. 2, consisting of eight SISO decoders (with eight Chase decoders each) and eight interleaving memories ($2 * 8 * 2$ k bit each). Ignoring latency of fulfilling four iterations, throughput is calculated by $T = f \times P$, where f is the decoder frequency and P is the number of Chase decoders in an SISO. Area A is represented by the number of equivalent two input NAND gates. If the Chase decoder supports a maximum code length of 2^q, the complexity of decoding a bit is $o(q)$. A fair comparison of TPC decoders with different code length is obtained using efficiency $E = \frac{qT}{A}$. Table I compares the synthesized result at 700 MHz with the related TPC decoders. Our proposed decoder shows high efficiency while supporting variable code length compared with other architectures. Coding gain only slightly declines and is even better in $(32, 26)^2$. Throughput can be improved by duplicating more Chase decoders that do not affect efficiency.

Table I. Comparison of related decoders

Design	Flexibility Support	Area (Mgates)	Throughput (Gbps)	Efficiency (operation/gate/second)	Coding Gain at 10e-9 (dB)
[4]	$(32,26)^2$	3.5	10.7	15.3	8.4
[5]	$(64,57)^2$	2.0	10.7	32.1	8.6
[6]	$(32,26)^2$	0.42	10.7	127.3	8.0
Proposed	From (8,4) to (128,120) both in row and column	0.30	5.6	130.7	$(8,4)^2$: 8.8
					$(16,11)^2$: 8.7
					$(32,26)^2$: 8.6
					$(64,57)^2$: 8.5
					$(128,120)^2$: 8.1

4 Conclusions

This paper proposes a high-efficiency Chase decoder using low complexity algorithms. Moreover, conflict-free memory access model is provided to avoid

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012

decoder read and write congestion. The TPC decoder exhibits high efficiency in cases of variable code length. Proven to be highly flexible and more efficient than others, our decoder is suitable for future wireless and optical communication.

Acknowledgments

This work was supported in part by the National NFSC of China (No. 06970037).

DOI: 10.1587/elex.9.1044
Received May 07, 2012
Accepted June 04, 2012
Published June 22, 2012