Fast Constant Weight Codeword to Index Converter

J. T. Butler

Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA U.S.A.

T. Sasao

Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka, Fukuoka, JAPAN

***Abstract*—Constant weight codewords, in which the number of 1's is fixed, are essential to many coding applications. In this paper, we show an efficient circuit that converts a constant weight codeword into a unique index of that codeword. For example, this circuit is necessary when constant weight codewords are used to transmit data on and off chip. Our circuit is based on the combinatorial number system in which the digits are binomial coefficients. It has $O(n^3)$ area complexity and $O(n)$ delay, where n is the number of variables. Two types of circuits are proposed. Various constant weight codes are implemented on an FPGA, including a 64-out-of-128 code. These implementations support our complexity analysis.**

Keywords: **constant weight codewords, converters, encoding**

I. Introduction

A constant weight code to index converter is needed when constant weight Gray codes are used to encode data in flash memory. In local rank modulation [2], data stored in flash memory is viewed as an n-bit constant weight codeword that differs in exactly two bits from an adjacent memory location (because of overlap). All codewords in this encoding have the same weight (number of 1's).

Balanced codes, with as many 0's as 1's, can be used to transfer data on and off VLSI chips so that the current fluctuations are minimized [7]. On the other hand, codes with small weight are desired in this application because they yield faster and more compact circuits [7]. Constant weight codewords can be used to counter "side-channel" attacks against secure systems [4]. Such attacks use data dependent differences in power consumption to extract hidden information. Constant weight codewords have been used in asynchronous logic to implement delay-insensitive codewords [8].

The use of constant weight codewords requires two parts, an index to constant weight code converter and a constant weight code to index converter. We considered the first part in [1]. However, we have not seen a hardware implementation of the second part, except for an implementation that requires $O(2^n)$ complexity [6]. In this paper, we propose an implementation with $O(n^3)$ complexity. In Section II, we discuss the combinatorial number system. We show how it can be used to convert a constant weight codeword to an index, and we present its circuit implementation. Then, in Section III, we show an improvement to this circuit that significantly reduces delay for large n. Finally, in Section IV, we give concluding remarks.

II. The Combinatorial Number System

A. Introduction

The basis for our constant weight code to index converter is the combinatorial number system [3].

Definition 1. *In an $\binom{n}{r}$* **combinatorial number system** *[5], an integer $N < \binom{n}{r}$ is represented as $N = c_r c_{r-1} \ldots c_1$, where*

$$N = \binom{c_r}{r} + \binom{c_{r-1}}{r-1} + \ldots + \binom{c_1}{1}, \qquad (1)$$

and $c_r > c_{r-1} > \ldots > c_1 \geq 0$.

Example 1. *Table I shows the representation of integers in the $\binom{6}{3}$ combinatorial number system. The leftmost column shows the integer's value in decimal and its vector representation. The middle column shows how this value is computed according to (1). The rightmost column of Table I shows the corresponding 6 bit constant weight code. Note that the three elements of the vector representation shown in the leftmost column correspond to the positions of the 1's in the constant weight codeword. For example, $19 = 5\ 4\ 3$ corresponds to 111000, there being 1's in positions 5, 4, and 3.*

(End of Example)

B. Circuit Implementation

A major contribution of this paper is to show how the combinatorial number system can be used to realize an efficient circuit that transforms a constant weight codeword to the index for that codeword. Such a circuit has for inputs the values of the rightmost column of Table I (the bits of the constant weight code) and has as outputs the standard binary number representation of the numbers shown in the leftmost column (the values of the index N). As shown in Table I, the 1-bits in the constant weight code contribute a value to its corresponding index depending on the 1 bit's position in the codeword. For example, from Table I, the 1's in the codeword 111000 contribute $\binom{5}{3}$, $\binom{4}{2}$, and $\binom{3}{1}$, from left to right. This can be seen in Fig. 1, which shows a circuit that converts a 3-out-of-6 constant weight codeword into the corresponding index of that codeword.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **AUG 2011**	2. REPORT TYPE	3. DATES COVERED
4. TITLE AND SUBTITLE **Fast Constant Weight Codeword to Index Converter**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Constant weight codewords, in which the number of 1?s is fixed, are essential to many coding applications. In this paper, we show an efficient circuit that converts a constant weight codeword into a unique index of that codeword. For example this circuit is necessary when constant weight codewords are used to transmit data on and off chip. Our circuit is based on the combinatorial number system in which the digits are binomial coefficients. It has O(n3) area complexity and O(n) delay, where n is the number of variables. Two types of circuits are proposed. Various constant weight codes are implemented on an FPGA including a 64-out-of-128 code. These implementations support our complexity analysis.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**4**	

TABLE I
THE $\binom{6}{3}$ COMBINATORIAL NUMBER SYSTEM FOR $0 \le N \le 19$

N	Computing the Value of N	Const Wght Code
	for $r = 3$	543210
19 = 5 4 3	$\binom{5}{3}+\binom{4}{2}+\binom{3}{1}=10+6+3$	111000
18 = 5 4 2	$\binom{5}{3}+\binom{4}{2}+\binom{2}{1}=10+6+2$	110100
17 = 5 4 1	$\binom{5}{3}+\binom{4}{2}+\binom{1}{1}=10+6+1$	110010
16 = 5 4 0	$\binom{5}{3}+\binom{4}{2}+\binom{0}{1}=10+6+0$	110001
15 = 5 3 2	$\binom{5}{3}+\binom{3}{2}+\binom{2}{1}=10+3+2$	101100
14 = 5 3 1	$\binom{5}{3}+\binom{3}{2}+\binom{1}{1}=10+3+1$	101010
13 = 5 3 0	$\binom{5}{3}+\binom{3}{2}+\binom{0}{1}=10+3+0$	101001
12 = 5 2 1	$\binom{5}{3}+\binom{2}{2}+\binom{1}{1}=10+1+1$	100110
11 = 5 2 0	$\binom{5}{3}+\binom{2}{2}+\binom{0}{1}=10+1+0$	100101
10 = 5 1 0	$\binom{5}{3}+\binom{1}{2}+\binom{0}{1}=10+0+0$	100011
9 = 4 3 2	$\binom{4}{3}+\binom{3}{2}+\binom{2}{1}=4+3+2$	011100
8 = 4 3 1	$\binom{4}{3}+\binom{3}{2}+\binom{1}{1}=4+3+1$	011010
7 = 4 3 0	$\binom{4}{3}+\binom{3}{2}+\binom{0}{1}=4+3+0$	011001
6 = 4 2 1	$\binom{4}{3}+\binom{2}{2}+\binom{1}{1}=4+1+1$	010110
5 = 4 2 0	$\binom{4}{3}+\binom{2}{2}+\binom{0}{1}=4+1+0$	010101
4 = 4 1 0	$\binom{4}{3}+\binom{1}{2}+\binom{0}{1}=4+0+0$	010011
3 = 3 2 1	$\binom{3}{3}+\binom{2}{2}+\binom{1}{1}=1+1+1$	001110
2 = 3 2 0	$\binom{3}{3}+\binom{2}{2}+\binom{0}{1}=1+1+0$	001101
1 = 3 1 0	$\binom{3}{3}+\binom{1}{2}+\binom{0}{1}=1+0+0$	001011
0 = 2 1 0	$\binom{2}{3}+\binom{1}{2}+\binom{0}{1}=0+0+0$	000111

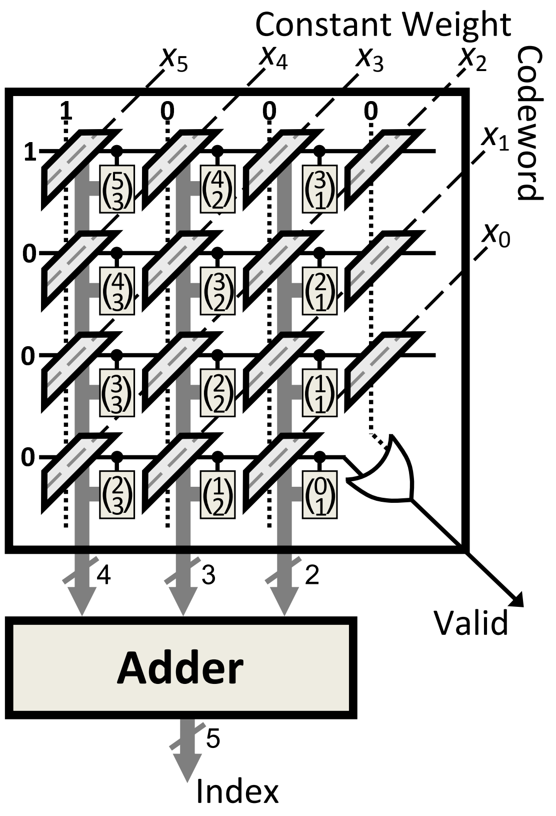

Fig. 1. Constant Weight Codeword to Index Converter Circuit.

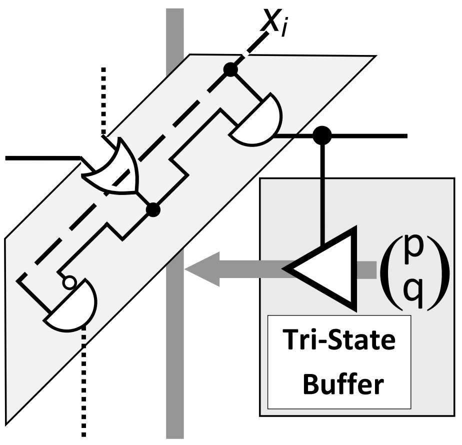

Fig. 2. Decoder and Binomial Constant Generator.

This circuit contains an array of decoders that control which digits occur in the combinatorial number. Fig. 2 shows the detail of the decoders and the tri-state circuit that provides constants for the combinatorial number. We can make the following observations.

1) If a 1 occurs on either or both inputs to the OR gate in the upper left hand corner of the decoder, then a 1 is produced at exactly one of the two outputs. Specifically, if x_i, the input driving the decoder is 1, then a 1 appears at the horizontal output (solid line), and a 0 appears at the vertical output (dotted line). On the other hand, if x_i is 0, then a 0 appears at horizontal output and a 1 appears at the vertical output. However, if *both* inputs to the OR gate in the upper left hand corner are 0, then both the horizontal and vertical outputs produce 0.
2) There is a path of 1's through the array of decoders in Fig. 1 beginning at the upper left hand corner. The path is determined by the values of x_i in the constant codeword. For example, if $x_5x_4x_3x_2x_1x_0 = 111000$, then the three decoders along the top of the array of decoders starting from the upper left hand decoder all produce 1 at their horizontal outputs. This is because their inputs, x_5, x_4, and x_3, are 1. However, the decoder in the upper right hand corner is driven by x_2, which is 0. So, the 1 at its OR gate input is directed now to its vertical output (while its horizontal output is 0). Because x_1 and x_0 are both 0, this 1 is directed downward (along dotted lines) through two decoders into the 2-input OR gate that drives `Valid`. That is, when $x_5x_4x_3x_2x_1x_0 = 111000$, `Valid` is 1, indicating the input codeword is a valid 3-out-of-6 codeword.
3) All other valid codewords result in a path of 1's from the upper left hand corner to the lower right hand corner, causing `Valid` to be 1. Conversely, a non-codeword causes `Valid` to be 0.
4) All horizontal lines from decoders drive binomial coefficient generators which apply to one of three bus lines that drive inputs of an adder whose output is the `Index`. Specifically, a 1 on the horizontal line causes the

corresponding binomial coefficient generator to drive its line. A 0 disconnects the binomial coefficient generator. For example, in the case of $x_5x_4x_3x_2x_1x_0 = 111000$, the three horizontal lines driven by decoders cause $\binom{5}{3}$, $\binom{4}{2}$, and $\binom{3}{1}$ to be applied to the three adder inputs resulting in 19 at the output, which is the index of 111000.

5) As shown in Fig. 1, each input to the adder is driven by four binomial coefficient generators. The first (leftmost) 1 in the constant weight codeword specifies which binomial coefficient generator drives the left input of the adder. The second 1 determines which drives the middle adder input, and the third (rightmost) 1 determines which drives the right adder input.

C. Complexity of Implementation

The complexity of the constant weight code to index converter, is dominated by the array of binomial coefficient generators and decoders. This array is a rectangle of $r+1$ by $n-r+1$ cells, for a total of $(r+1)(n-r+1)$ cells. With $r = \frac{n}{2}$, the (worst case) number of cells is $O(n^2)$. The decoder has a complexity that is independent of n. However, the binomial coefficient generator requires $O(n)$ tri-state buffers. That is, the binomial coefficient with the most tri-state buffers is the one in the upper left hand corner; it realizes $\binom{n-1}{r}$, which requires no more that $O(n)$ tri-state buffers. Thus, the total complexity is $O(n^3)$. And so, the constant weight codeword to index converter has complexity polynomial in n. Table II shows the exact number of tri-state buffers and decoders needed in the proposed constant weight codeword to index converter. In the case of the tri-state buffers, the array cell at the top of each column corresponds to the largest binomial coefficient in that column and thus determines the number of bits needed for that adder input.

TABLE II
THE NUMBER OF TRI-STATE BUFFERS AND DECODERS NEEDED IN THE CONSTANT WEIGHT CODEWORD TO INDEX CONVERTER

n	r	# of tri-state	# of decoders
4	2	12	9
6	3	36	16
8	4	90	25
10	5	162	36
12	6	266	49
14	7	424	64
16	8	648	81
18	9	920	100
20	10	1254	121
22	11	1680	144
24	12	2184	169

The longest path in the circuit is from x_{n-1} through the array to the `Valid` output, and it is $O(n)$. The delay of the adder can be neglected, since it is $O(\log n)$. Thus, the overall delay is $O(n)$.

D. FPGA Resources Used

To understand how the complexity of a $\binom{n}{r}$ combinatorial number system constant weight code to index converter depends on n and r, we implemented this system for various n and r on the 40 nm Altera Stratix IV EP4SE530F43C3NES FPGA. Table III shows the delay obtained and the resources used in this implementation. The leftmost column shows the constant weight code as a binomial number. For example, $\binom{128}{64}$ corresponds to a 64-out-of-128 bit code. The second column shows how many bits in the output `Index` are needed to represent the largest this code. The third column gives the delay achieved, which is inversely proportional to the frequency of the circuit. The rightmost column gives the number of ALMs needed to realize this circuit, which a measure of the area. Although this table shows only balanced constant weight code generators where the number of bits is a power of 2, our approach applies to any number of bits and to any weight.

TABLE III
DELAY AND RESOURCES USED TO REALIZE COMBINATORIAL NUMBER SYSTEM CONSTANT WEIGHT CODE GENERATORS ON THE ALTERA STRATIX IV EP4SE530F43C3NES FPGA.

Con. Wgt. Code $\binom{n}{r}$	# Bits `Index`	Freq. (MHz	Delay (ns.)	Est. # of Packed ALMs
$\binom{4}{2}$	3	261.6	3.8	2 (0%)
$\binom{8}{4}$	7	178.7	5.6	17 (0%)
$\binom{16}{8}$	14	104.4	9.6	120 (0%)
$\binom{32}{16}$	30	57.5	17.4	647 (0%)
$\binom{64}{32}$	61	31.5	31.7	3,203 (1%)
$\binom{128}{64}$	125	15.2	65.8	20,497 (9%)

Our circuit was synthesized using Synplify Pro and modeled using ModelSim. A large codeword is achievable; a 64-out-of-128 bit converter uses only 9% of the available ALMs. The large values of n required special Verilog programming. For example, to implement the 64-out-of-128 bit constant weight codeword to index converter requires that the binary value of $\binom{127}{64}$ be applied to the adder circuit. This value is much too large for Synplify Pro. To overcome this deficiency, we computed the binary value of $\binom{127}{64}$ and other values of $\binom{n}{r}$ in a MATLAB program and wrote it to a header file that was `include`d in the Verilog code.

III. Complex Disjoint Decomposition Solution

It can be see from Fig. 1 that the longest path through the array of the constant weight codeword converter has length n, where n is the number of bits in the constant weight code. In computing the index, each 1 contributes a value that depends on the number of 1's that preceded it. A 1 in the leftmost bit position is an exception to this. This 1 always contributes $\binom{6}{3} = 10$. This can be seen in Table I; the constant weight codewords with a 1 in the leftmost bit corresponds to a combinatorial number in which the most significant digit is $\binom{5}{3}$. However, a 1 in the second bit from the left contributes a different value to its combinatorial number representation depending on whether the leftmost bit is 1 or 0. If 1, then the second 1 contributes $\binom{4}{2}$. If 0, then it contributes $\binom{4}{3}$.

A similar phenomena exists at the right side. Interestingly, the least significant bit, whether 0 or 1 contributes 0 to the combinatorial number's value. This is because that bit is "forced" to be 0 or 1 depending on whether there are six or five 1 bits to its left. However, note that the right digit of the combinatorial number's value is 1 iff the least significant bit of the constant weight code is 0. This can also be seen in the circuit of Fig. 1. Here, if x_0 is 1, then 0 drives the least significant digit, and none of the other three binomial coefficients can drive this least significant digit. That is, x_0 and the only decoder it drives is the "mirror" image of x_5 and the only decoder it drives. Similarly, x_1 and the two decoder it drives are the mirror image of x_4 and the two decoder it drives. Therefore, we can realize the same circuit by reversing the decoders in Fig. 1 that are driven by x_2, x_1, and x_0. The new circuit is shown in Fig. 3.

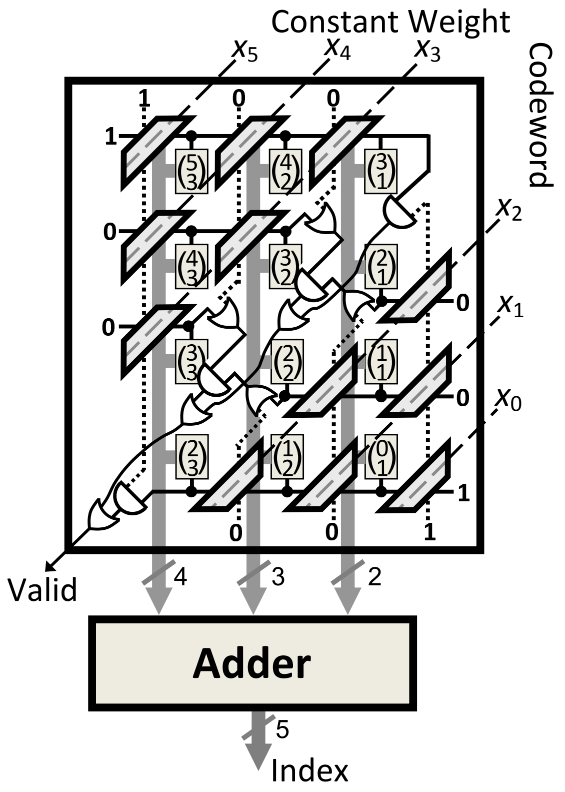

Fig. 3. Constant Weight Codeword to Index Converter Circuit Consisting of Two Subcircuits.

In the new circuit, the inputs are divided into two parts $\{x_5, x_4, x_3\}$ and $\{x_2, x_3, x_1\}$, where each part drives a separate subcircuit. The two subcircuits, in turn, drive inputs to the adder, which, in turn, drives the `Index` output. Such a circuit is said to have a **complex disjoint decomposition** (**CDD**).

Table IV shows the delay achieved and the resources used for the CDD circuit. The benefit of the new circuit is its reduced delay, especially in large circuits. This can be seen by comparing Tables IV and III. For example, for 64-out-of-128 codes, the delay of circuit consisting of two subcircuits is 64% that of the full rectangle circuit.

TABLE IV
DELAY AND RESOURCES USED FOR THE CDD CONSTANT WEIGHT CODEWORD TO INDEX CONVERTER ON THE ALTERA STRATIX IV EP4SE530F43C3NES FPGA.

Con. Wgt. Code $\binom{n}{r}$	# Bits `Index`	Freq. (MHz)	Delay (ns.)	Est. # of Packed ALMs
$\binom{4}{2}$	3	262.9	3.8	3 (0%)
$\binom{8}{4}$	7	193.5	5.2	16 (0%)
$\binom{16}{8}$	14	127.9	7.8	116 (0%)
$\binom{32}{16}$	30	80.8	12.4	657 (0%)
$\binom{64}{32}$	61	47.8	20.9	3,503 (1%)
$\binom{128}{64}$	125	23.6	42.4	20,723 (9%)

IV. CONCLUDING REMARKS

Although there is a need for a circuit that computes an index from a constant weight codeword, we have not seen a simple implementation. We show a circuit based on the combinatorial number system that has complexity $O(n^3)$, where n is the number of bits in the code. Our circuit is useful, for example, in the encoding/decoding of data, such as between on-chip and off-chip and in delay-insensitive logic for asynchronous circuits. It has only $O(n)$ delay. We also show an improvement that reduces by about half the delay that still has $O(n^3)$ complexity. We have implemented our designs on an Altera Stratix IV EP4SE530F43C3NES FPGA. This has shown that both circuits are efficiently implemented.

V. ACKNOWLEDGMENTS

This research is supported by a Knowledge Cluster Initiative (the second stage) of MEXT (Ministry of Education, Culture, Sports, Science and Technology).

REFERENCES

[1] J. T. Butler and T. Sasao, "High-speed constant weight code generators," *7th Inter. Symp. on Applied Reconfigurable Computing (ARC2011), Proceedings Lecture Notes in Computer Science (LNCS 6576)*, Springer-Verlag Berlin Heidelberg, 2011, A. Koch et al (eds.), Belfast, N. Ireland, Mar. 23-25, 2011, pp. 193-204.
[2] E. Gad, M. Langberg, M. Schwartz, and J. Bruck, "Constant-weight Gray codes for local rank modulation," *Proc. of the 2010 IEEE International Symposium on Information Theory, ISIT2010*, pp. 869-873, Austin, TX, U.S.A., June 2010.
[3] Combinadic - `http://en.wikipedia.org/wiki/Combinadic`.
[4] `http://en.wikipedia.org/wiki/Side_channel_attack`.
[5] D. E. Knuth, *The Art of Computer Programming*, "Generating all combinations and partitions," Vol. 4, **Fascicle 3**, Addison-Wesley, pp. 5-6, ISBN 0-321-58050-8, (2009).
[6] T. Sasao, *Memory Based Logic Synthesis*, 1st Edition, Springer, ISBN 978-1-4419-8103-5, (2011)
[7] L. G. Tallini and B. Bose, "Design of balanced and constant weight codes for VLSI systems," *IEEE Trans. on Computers*, Vol. 47, No. 5, pp. 556–572, (1998)
[8] T. Verhoeff, "Delay-insensitive codes - an overview," *Distr. Comp*, 3(1):1-8, (1988)