Some Improved Encoding and Decoding Schemes for Balanced Codes *

Jong-Hoon Youn and Bella Bose
Dept. of Computer Science
Oregon State University
Corvallis, OR 97331
{jhyun, bose}@cs.orst.edu
Tel. 541-737-5573
FAX. 541 737-3014

August 13, 2001

Abstract

A binary code of length n is called a *balanced code* if each codeword contains exactly $\lfloor n/2 \rfloor$ (or $\lceil n/2 \rceil$) ones and $\lceil n/2 \rceil$ (or $\lfloor n/2 \rfloor$) zeros. In this paper, we give two improved methods for encoding and decoding the balanced codes. The first one, called improved single map, improves the computation complexity of the complementation methods, first proposed by Knuth. This method, instead of complementing one bit at a time as done in Knuth's method, complements several appropriate bits at a time. Some simulation results show the improvement of this scheme over the previously known methods. The second one is a parallel implementation of this method.

*This work is supported by the U.S. National Science Foundation grant under MIP-9705738.

1 Introduction

The binary code B is a balanced code with r check bits and k information bits if and only if,

- B is a block code of length $n = k + r$,
- Each word $X \in B$ has $\lfloor n/2 \rfloor$ (or $\lceil n/2 \rceil$) ones and $\lceil n/2 \rceil$ (or $\lfloor n/2 \rfloor$) zeros,
- B has 2^k codewords.

Balanced codes have the property that no codeword is contained in another; that is the positions of the ones in one codeword will never be a subset of the positions of the ones in a different codeword. This property makes balanced codes attractive for certain applications.

Balanced codes are

- capable of detecting all unidirectional errors. In the case of unidirectional errors, both $1 \rightarrow 0$ and $0 \rightarrow 1$ errors are possible; however, in any particular received word all the errors are of the same type. These types of errors are predominant in VLSI circuits and memories [7, 9].
- used in achieving the data integrity of write-once memories such as laser disks, where a 0 can be changed to 1, but a written 1 cannot be changed to 0 [5, 6].
- used in state assignments in fault-tolerant and fail-safe sequential circuit design [12].
- used in fiber optics and magnetic and optical storage media [4, 8].
- useful to achieve delay insensitive communication [13].
- useful to accomplish noise reduction in VLSI chips [10].

Balanced codes with serial encoding schemes and parallel and serial decoding schemes are given by Knuth in [5]. Using r check bits, the parallel decoding scheme can have up to $k = 2^r - r - 1$ information bits; the serial decoding scheme can have up to $k = 2^r$ information bits. In both methods, for each given information word, some appropriate number of bits, starting from the first bits, are complemented; then a check

symbol of r-bits is assigned to this modified information word to make the entire word balanced. In the sequential decoding scheme the check represents the weight of the original information word whereas in the parallel decoding scheme the check directly indicates the number of information bits complemented. Al-Bassam and Bose in [1] improved the parallel decoding scheme by presenting a construction with $k = 2^r - (r \bmod 2)$ information bits using r-check bits. They showed this construction is optimal when Knuth's complementation method mentioned above is used. The serial balanced coding scheme has been extended to $k = 2^{r+1} - r - 2$ information bits by Bose in [3] and further extended to $k = 2^{r+1} - \lceil 0.8\sqrt{r} \rceil - 2$ information bits by Al-Bassam and Bose in [2] again using r-check bits. The latter result was also shown to be optimal when Knuth's complementation method is used. In [11], Tallini, Capocelli and Bose have designed new methods to construct efficient balanced codes based on the concept of tail-map. A tail-map is an injective function from the set of unbalanced words to the set of the balanced words. Three different tail-map constructions were presented. These three methods, using r check bits, can encode up to k information bits as follows.

1. Method 1: $k = 2^{r+1} - 2$,
2. Method 2: $k = 3 \times 2^r - 8$,
3. Method 3: $k = 5 \times 2^r - 10r + C(r)$, where $C(r) = \{-15, -10, -5, 0, +5\}$.

In this paper, we propose two implementation methods for balanced codes which improve the computational complexity of the complementation method given in [1, 2, 3, 5, 11].

The following notations are used in this paper.

k	number of information bits
r	number of check bits
$n = k + r$	length of the codeword
Z_2	$\{0, 1\}$
Z_2^k	the binary string of length k
	$Z_{2,1} \times Z_{2,2} \times \ldots \times Z_{2,k}$
$\overline{X}$	complement of X
$\omega(X)$	weight or the number of 1's in X
S_ω^k	$\{X \in Z_2^k : \omega(X) = \omega\}$
	the binary string of length k and weight w.

The remainder of the paper is organized as follow. Section 2 reviews the complementation method for encoding and decoding of the balanced codes, and Section 3 describes the new coding schemes for the balanced codes. Finally, conclusion are drawn in Section 4.

2 Knuth's complementation method

Since most of the balanced code design methods are based on Knuth's complementation method [5], we briefly describe the method here. In the construction, the encoding and decoding require only a complementation of some appropriate number of bits in the information word (starting from the beginning). If $X = x_1\ x_2\ \ldots\ x_k$ is a binary information word, the balanced codeword encoding of X consists of X with the first j bits complemented, and an appropriate check symbol of length r appended, as depicted in the following: $x_1x_2\ldots x_k \longrightarrow \overline{x_1x_2}\ldots\overline{x_j}x_{j+1}\ldots x_k \quad c_1c_2\ldots c_r$.

In serial decoding, the check symbol specifies the original weight of the information word, so the decoder complements one bit at a time until it recovers the original information word. On the other hand, in parallel decoding the check symbol specifies the number of bits complemented, so the decoder simultaneously complements that many bits to recover the original information word.

In order to elaborate Knuth's serial complementation method, some notation is needed. Given $X = x_1x_2\ldots x_k \in Z_2^k$, let $X^{(j)}$ be X with the first j bits complemented, i.e. $X^{(j)} = \overline{x_1x_2}\ldots\overline{x_j}x_{j+1}\ldots x_k$. Further, let $\sigma_j(X) = \omega(X^{(j)}) = \omega(\overline{x_1x_2}\ldots\overline{x_j}x_{j+1}\ldots x_k)$, where $\omega(X)$ is the weight of X.

For example, if $X = 1001\ 0000$ then $X^{(5)} = 0110\ 1000$ and $\sigma_5(X) = 3$. As a function of j, $\sigma_j(X)$ satisfies the following properties:

$\sigma_0(X) = \omega(X)$,

$\sigma_k(X) = k - \omega(X)$, where k is the length of X.

$\sigma_j(X) = \sigma_{j-1}(X) \pm 1 \qquad$ for $j = 1, 2, \ldots, k$,

$\sigma_i(X) - (j-i) \le \sigma_j(X) \le \sigma_i(X) + (j-i) \qquad$ for any i and j, where $0 \le i \le j \le k$.

As a function of j, $\sigma_j(X)$ goes from $\sigma_0(X) = \omega(X)$ to $\sigma_k(X) = k - \omega(X)$ by unit steps. Thus, it is possible to obtain a word of any weight between $\omega(X)$ and $k - \omega(X)$ by complementing the first j bits of X, for some j where $0 \le j \le k$.

In particular, there always exists a j such that $\sigma_j(X) = \lceil \frac{k}{2} \rceil$ (or $\lfloor \frac{k}{2} \rfloor$). In this sense, $\sigma_j(X)$ represents a 'random walk' from $\omega(X)$ to $k-\omega(X)$. For example, if $X = 0101\ 0000$ then $\omega(X) = 2$ and $k - \omega(X) = 8 - 2 = 6$ and so we can obtain words of weight 2,3,4,5, and 6 by complementing the first 0,1,6,7 and 8 bits respectively. There may be several j's which give the same weight. However, all methods use the smallest such j.

Knuth's method with serial encoding and decoding scheme relies on the properties of $\sigma_j(X)$. A check symbol is an element of Z_2^r. The general strategy of the encoding scheme is that first partitions the set of information words into 2^r sets, one for each check symbol, say $\{S^C\}_{C \in Z_2^r}$; then using the 2^r one-to-one functions

$\varphi : S^C \longrightarrow S_v^k$, where S_v^k is a set of binary strings of length k and weight v,
maps the words of S^C to the words of weight $v = \lfloor \frac{k+r}{2} \rfloor - \omega(C)$.

In order to encode the generic information word X, the encoder performs the following steps:

1. Compute $\omega(X)$ and decide which set of the partition X belongs to, say $X \in S^C$.
2. Compute $v = \lfloor \frac{k+r}{2} \rfloor - \omega(C)$.
3. Complement X one bit at a time until the weight v is obtained. Let this word be Y.
4. Append the check symbol C to Y to obtain the encoded word: $YC = \varphi(X)C$.

In *setp* 3, we should complement one bit, change the current weight and compare it to v. At the worst, we perform these operations k times. Thus, *step* 3 takes $\Theta(k)$.

Since v is chosen so that $v = \lfloor \frac{k+r}{2} \rfloor - \omega(C)$, we have $\omega(YC) = \omega(Y) + \omega(C) = v + \omega(C) = \lfloor \frac{k+r}{2} \rfloor - \omega(C) + \omega(C) = \lfloor \frac{k+r}{2} \rfloor$.

Decoding is straightforward. If YC is received, the decoder, reading the check symbol C, knows that the information word lies in the set S^C. So it computes $X = \varphi^{-1}(Y)$, and complements Y until the proper weight is obtained.

Example 1: A balanced code with r=3 check bits and k=8 information bits can

be encoded as follows. Assume the information word $X = 1101\ 1111 \in S_7^8$ needs to be encoded, and the check symbol for S_7^8 is 110. The encoder performs the following steps.

1. Compute $\omega(X) = 7$. Since $X \in S_7^8$, the check symbol for X is 110.
2. Compute $v = \lfloor \frac{8+3}{2} \rfloor - \omega(110) = 5 - 2 = 3$.
3. Complement X one bit until weight 3 is obtained. Since the smallest j such that $\omega(X^{(j)}) = 3$ is 6, the computed word Y is
   $Y = X^{(6)} = 0010\ 0011$.
4. Append the check symbol C to Y to obtain the encoded word: $YC = 0010\ 0011\ \ 110$. Now $\omega(YC) = 5$ and so, the codeword is balanced.

On receiving $YC = 0010\ 0011\ \ 110$, the decoder, reading the check symbol $C = 110$, knows that the original information word came from the set S_7^8. So it complements Y one bit at a time until the weight 7 is reached. Since weight 7 is first reached by complementing the first 6 bits, it decodes Y as $X = Y^6 = 1101\ 1111$.

3 The Proposed Schemes

In this section, we proposed two improved methods for encoding and decoding the balanced codes; Improved Single Map and Parallel Single Map. The first method improves the computation complexity of the complementation method by complementing several appropriate bits at a time. The second one is a parallel implementation of the complementation method. Although we compare our schemes to Knuth's methods, the proposed methods can be applied to all the complementation methods given in [1, 2, 3, 5, 11].

3.1 Improved Single Map

Knuth's single map relies on the properties of $\sigma_j(X)$. Let $v = \lfloor \frac{k+r}{2} \rfloor - \omega(X)$ and $v' = |v - \omega(X)|$. Since $\sigma_j(X) = \sigma_{j-1}(X) \pm 1$, $\sigma_j(X) - v' \leq \sigma_{j+v'}(X) \leq \sigma_j(X) + v'$.

Since v' is the absolute value of the difference between the desired weight(v) and the current weight ($\sigma_j(X)$), it is impossible to reach weight v by complementing less than v' bits. Thus, we can complement v' bits simultaneously. Our new serial coding scheme

is based on this idea. In order to encode the generic information word X, the encoder performs the following steps:

1. Compute $\omega = \omega(X)$ and decide which set of the partition X belongs to, say $X \in S^C$.
2. Compute $v = \lfloor \frac{k+r}{2} \rfloor - \omega(C)$ and initialize $j = 0$.
3. Compute $v' = |\omega - v|$.
4. If $v' = 0$, go to *step* 8.
5. Complement $X^{(j)}$ from the $(j+1)th$ bit to the $(j+v')th$ bit.
6. Update the weight ω and compute $j = j + v'$.
7. Go to *step* 3.
8. Append the check symbol C.

Decoding is straightforward. In order to decode the generic codeword YC, the decoder performs the following steps:

1. Read the check symbol C and decide the weight of X.
2. Compute $\omega = \omega(Y) = \lfloor \frac{k+r}{2} \rfloor - \omega(C)$ and initialize $j = 0$.
3. Compute $v' = |\omega(X) - \omega|$.
4. If $v' = 0$, end.
5. Complement $Y^{(j)}$ from the $(j+1)th$ bit to the $(j+v')th$ bit.
6. Update the weight ω and compute $j = j + v'$.
7. Go to *step* 3.

Example 2. A balanced code with r=3 check bits and k=8 information bits can be encoded as follows. Assume the information word $X = 1101\ 1111 \in S_7^8$ needs to be encoded, and the check symbol for S_7^8 is 110. The encoder performs the following steps.

1. Compute $\omega = \omega(X) = 7$. Since $X \in S_7^8$, the check symbol for X is 110.
2. Compute $v = \lfloor \frac{8+3}{2} \rfloor - \omega(110) = 5 - 2 = 3$ and initialize j to 0.
3. Compute $v' = |v - \omega| = |3 - 7| = 4$.
4. Complement the first 4 bits of X. The computed word is $X^{(4)} = 0010\ 1111$.
5. Update ω and j: $\omega = 5$ and $j = 0 + 4 = 4$.
6. Compute $v' = |3 - 5| = 2$.
7. Complement the 5th and 6th bits of X^4. The computed word is $X^6 = 0010\ 0011$.
8. Update ω and j: $\omega = 3$ and $j = 4 + 2 = 6$.
9. Compute $v' = |3 - 3| = 0$. Since $v' = 0$, the word Y is equal to $Y = 0010\ 0011$.
10. Append the check symbol $C = 110$ to obtain the encoded word of X: $YC = 0010\ 0011\ \ 110$.
    Now $\omega(YC) = 5$ and so, the word is balanced.

On receiving $YC = 0010\ 0011\ \ 110$, the decoder, reading the check symbol $C = 110$, knows that the original information word came from S_7^8, i.e. $\omega(X) = 7$. It performs the following steps.

1. Compute $\omega = \omega(Y) = 3$ and initialize j to 0.
2. Compute $v' = |\omega(X) - \omega| = |7 - 3| = 4$.
3. Complement the first 4 bits of Y. The computed word is $Y^4 = 1101\ 0011$.
4. Update ω and j: $\omega = 5$ and $j = 0 + 4$.
5. Compute $v' = |7 - \omega| = |7 - 5| = 2$.
6. Complement the 5th and 6th bits of Y^4. The computed word is $Y^{(6)} = 1101\ 1111$.
7. Update ω and j: $\omega = 7$ and $j = 4 + 2 = 6$.

8. Compute $v' = |7 - 7| = 0$. Since $v' = 0$, it decodes Y as $X = 1101\ 1111$

In order to compare the efficiency of the proposed method to that of Knuth's method, we have simulated both Knuth's single map (KSM) and improved single map (ISM) for k=8, 16 and 32. In this simulation, we generate all the possible binary strings (2^k binary strings) and encode them using both KSM and ISM methods. The simulation results are shown in Table 1, Table 2 and Table 3. These results show that ISM outperforms KSM for all simulated cases. It is noteworthy that the ratio between KSM's execution time and ISM's execution time increases as k increases. This is because, as k increases, v' also increases, and so ISM complements a large number of bits at a time. For large k ($k > 32$), we expect ISM to perform much better than KSM.

Average # of bit complementation							
Weight	Check	KSM	ISM	Weight	Check	KSM	ISM
0	001	4.00	1.00	5	000	0.00	0.00
1	011	2.50	1.25	6	010	3.29	1.61
2	111	0.00	0.00	7	110	5.00	1.50
3	101	0.00	0.00	8	001	4.00	1.00
4	100	0.00	0.00	Overall		0.62	0.27

Table 1: Comparison on the average number of bit complementations for $k = 8, r = 3$ and $v = 5$.

3.2 New parallel coding scheme

By extending the results given by Knuth in [5], many improved coding techniques have been designed [1, 2, 3, 11, 14]. These methods usually require serial encoding and decoding. However, in some applications (for example, noise reduction in VLSI Systems) it is desirable to have parallel encoding and parallel decoding schemes for balanced codes. In this section, a fast parallel implementation of these serial encoding and decoding schemes is presented.

In the complementation method, if the next bit to be complemented is a '1', then the resultant weight of the word will decrease by 1 after complementation, whereas it will increase by 1 if it is a '0'. Thus, if the accumulated variations of weight from the first bit to the k−th bit are calculated in parallel, then the proper number of bits, j, to be complemented can be found in one step. The proposed method is based on this idea.

Average # of bit complementation							
Weight	Check	KSM	ISM	Weight	Check	KSM	ISM
0	0011	8.00	1.0	9	0001	0.00	0.00
1	0101	7.88	1.44	10	0000	0.00	0.00
2	0110	7.70	1.73	11	0010	4.30	2.01
3	0111	5.96	1.83	12	0100	5.30	1.97
4	1011	5.30	1.97	13	1000	5.96	1.83
5	1101	4.30	2.01	14	1010	7.70	1.73
6	1111	0.00	0.00	15	1100	7.88	1.44
7	1110	0.00	0.00	16	0011	8.00	1.00
8	1001	0.00	0.00	Overall		1.00	0.42

Table 2: Comparison on the average number of bit complementations for $k = 16, r = 4$ and $v = 10$.

In order to elaborate the new coding method, we introduce a new notation. Let $\psi_j(X)$ be the variation of weight of X with the first j bits complemented;

$\psi_j(X) = \sigma_j(X) - \omega(X)$.

For example, if $X = 1101\ 1111$, then $\psi_2(X) = \sigma_2(X) - \omega(X) = \omega(0001\ 1111) - \omega(1101\ 1111) = 5 - 7 = -2$, $\psi_5(X) = \sigma_5(X) - \omega(X) = \omega(0010\ 0111) - \omega(1101\ 1111) = 4 - 7 = -3$, and $\psi_7(X) = \sigma_7(X) - \omega(X) = \omega(0010\ 0001) - \omega(1101\ 1111) = 2 - 7 = -5$.

The encoder performs the following steps using $\Theta(k)$ adders to encode the given information word $X = x_1\ x_2\ \ldots\ x_k$.

Average # of bit complementation							
Weight	Check	KSM	ISM	Weight	Check	KSM	ISM
0	00111	15.00	1.0	17	00011	0.00	0.00
1	01011	14.88	1.44	18	00000	0.00	0.00
2	01101	14.73	1.71	19	00010	7.32	3.01
3	01110	14.56	1.92	20	00100	9.29	3.24
4	10011	14.36	2.12	21	01000	10.67	3.27
5	10101	14.12	2.32	22	10000	11.68	3.19
6	10110	13.82	2.51	23	00101	14.50	3.25
7	11001	13.46	2.70	24	01100	14.87	3.03
8	11010	13.01	2.88	25	01001	15.14	2.82
9	01111	10.36	2.83	26	01010	15.36	2.60
10	10111	9.34	2.89	27	01100	15.53	2.39
11	11011	8.00	2.86	28	10001	15.66	2.18
12	11101	6.19	2.65	29	10010	15.77	1.97
13	11111	0.00	0.00	30	10100	15.86	1.75
14	11110	0.00	0.00	31	11000	15.94	1.47
15	11100	0.00	0.00	32	00111	17.00	1.00
16	00011	0.00	0.00	Overall		2.54	0.89

Table 3: Comparison on the average number of bit complementations for $k = 32, r = 5$ and $v = 18$.

1. Compute $\omega(X)$ and decide which set of the partition X belongs to, say $X \in S^C$.
2. Compute $v' = \lfloor \frac{k+r}{2} \rfloor - \omega(C) - \omega(X)$.
3. Find $X' = \phi(X)$, where $\phi(x_i) = -1$ if $x_i = 1$, and $\phi(x_i) = 1$ if $x_i = 0$ for $1 \leq i \leq k$.
4. Using parallel prefix additions for X', compute $\psi_j(X)$ for $1 \leq j \leq k$.
5. Find the smallest j such that $\psi_j(X) = v'$, and complement the first j bits of X.
6. Append the check symbol C.

Note that the parallel prefix computation in *Step 4* can be computed in $\Theta(\log k)$ steps. An example of parallel prefix addition is shown in Figure 1.

Decoding is straightforward. In order to decode the given codeword YC, the decoder performs the following steps using $\Theta(k)$ adders:

1. Read the check symbol C and decide the weight of X.
2. Compute $v' = \omega(X) - \omega(Y)$.
3. Find $Y' = \phi(Y)$, where $\phi(y_i) = -1$ if $y_i = 1$, and $\phi(y_i) = 1$ if $y_i = 0$ for $1 \leq i \leq k$.
4. Using parallel prefix additions for Y', compute $\psi_j(Y)$ for $1 \leq j \leq k$.
5. Find the smallest j such that $\psi_j(Y) = v'$, and complement the first j bits of Y.

Example 3: A balanced code with r=3 check bits and k=8 information bits can be encoded as follows. Assume the information word $X = 1101\ 1111 \in S_7^8$ needs to be encoded, and the check symbol for S_7^8 is 110. The encoder performs the following steps.

1. Since $\omega(X) = 7$, the check symbol for X is 110.
2. Compute $v' = \lfloor \frac{8+3}{2} \rfloor - \omega(C) - \omega(X) = -4$.
3. $X' = \phi(1101\ 1111) = -1 - 1 \quad 1 - 1 - 1 - 1 - 1 - 1 - 1$.
4. Compute $\psi_j(X)$ for $1 \leq j \leq k$ (see Fig. 1).
5. Since the smallest j such that $psi_j(X) = -4$ is 6, complement the first 6 bits of X.
6. Append the check symbol $C = 110$ to get the codeword 0010 0011 110.

On receiving $YC = 0010\ 0011\ \ 110$, the decoder, reading the check symbol $C = 110$, knows that the original information word came from S_7^8. Then, it performs the following.

1. Compute $v' = \omega(X) - \omega(Y) = 7 - 3 = 4$.

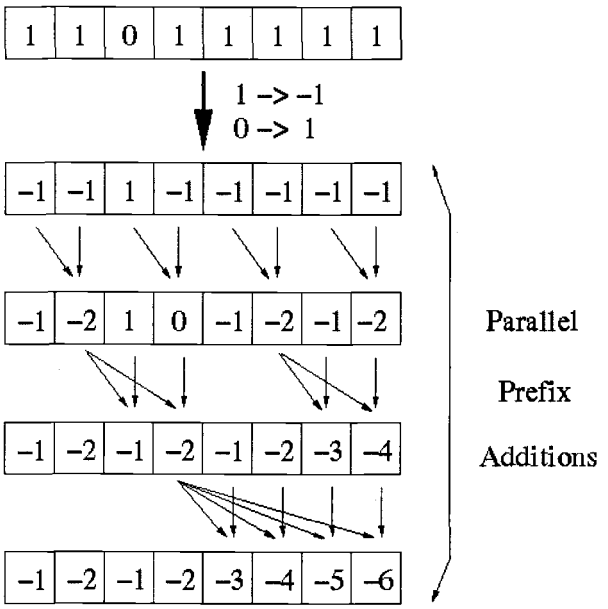

Figure 1: Parallel computation of $\psi_j(X)$ where $1 \leq j \leq 8$, in a given word X=1101 1111.

2. $Y' = \phi(0010\ 0011) = 1 \quad 1-1 \quad 1 \quad 1 \quad 1-1-1.$

3. Compute $\psi_j(Y)$ (see Fig. 2).

4. Since the smallest j such that $\psi_j(Y) = 4$ is 6, complement the first 6 bits of Y.

Computing $\psi_j(X)$ can be done in $\Theta(\log k)$ using $\Theta(k)$ adders, and all other steps can be done in $\Theta(1)$. Thus, the overall time complexity of the proposed encoding scheme is $\Theta(\log\ k)$. With regard to the decoding complexity, a similar argument can be applied. So, the time complexity of decoding scheme is also $\Theta(\log\ k)$.

4 Conclusions

In this paper, by extending the results given by Knuth, we describe two improved coding schemes for designing balanced codes. Our coding scheme does not save any bit, but they perform Knuth's serial encoding and decoding scheme more efficiently. The first method, called improved single map, improves the computation complexity of the complementation methods by complementing several appropriate bits at a time instead

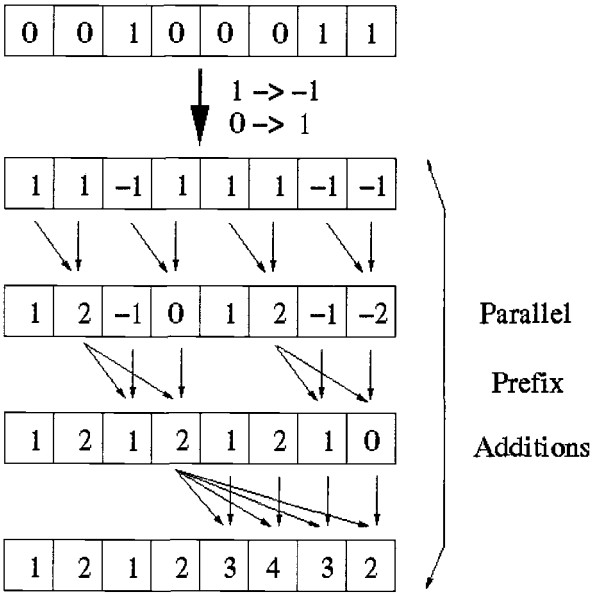

Figure 2: Parallel computation of $\psi_j(X)$ where $1 \leq j \leq 8$, in a given word X=0010 0011.

of complementing one bit at a time as done in Knuth's method. The simulation results given in Section 3 show the superiority of the proposed serial coding scheme over Knuth's scheme. It can be seen that, as the information length, k, increases, the ratio between KSM's execution time and the proposed coding scheme's execution time increases. The second method, called parallel single map, is a parallel implementation of the complementation method. In general, the time complexity of Knuth's serial encoding scheme is $\Theta(k)$. Our parallel encoding scheme performs the same operations in $\Theta(\log k)$. Thus, our encoding scheme is computationally faster than Knuth's.

References

[1] S. Al-Bassam and B. Bose, On balanced codes, *IEEE Trans. Inform. Theory*, vol. 36, pp. 406-408, Mar. 1990.

[2] S. Al-Bassam and B. Bose, Design of efficient balanced codes, *IEEE Trans. Comput.*, vol. 43, no. 3, pp. 362-365, Mar. 1994.

[3] B. Bose, On unordered codes, *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 125-131, Feb. 1991.

[4] R. Karabed and P.H. Siegel, Matched spectral-null codes for partial-response channels, *IEEE Trans. Inform. Theory*, vol. 37, pp. 818-855, May 1991.

[5] D. E. Knuth, Efficient balanced codes, *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 51-53, Jan. 1986.

[6] E. L. Leiss, Data integrity in digital optical disks, *IEEE Trans. Comput.*, vol. c-33, pp. 818-827, Sept. 1984.

[7] D.K. Pradhan and J.J. Stiffler, Error correcting codes and self-checking circuits in fault-tolerant computers, *IEEE Comput. Mag.*, vol. 13, pp. 27-37, Mar. 1980.

[8] R. M. Roth, P. H. Siegel and A. Vardy, High-order spectral-null codes-constructions and bounds, *IEEE Trans. Inform. Theory*, vol. 40, pp. 1826-1840, Nov. 1994.

[9] S. J. Piestrak, Design of self-testing checkers for unidirectional error detecting codes, *Scientific papers of the institute of technical cybernetics of the technical university of wroclaw*, No. 92, monograph No. 24, 1995.

[10] J. Tabor, Noise reduction using low weight and constant weight coding techniques *Technical report* AI-TR 1232, MIT Artificial Intelligence Laboratory, Jun. 1990.

[11] L. G. Tallini, R. M. Capocelli, and B. Bose, Design of some new efficient balanced codes, *IEEE Trans. Inform. Theory*, vol. 42, pp. 790-802, May 1996.

[12] Y. Tohma, R. Sakai and R. Ohyama, Realization of fail-safe sequential machines by using k-out-of-n code, *IEEE Trans. Comput.*, vol. c-20, pp. 1270-1275, Nov. 1971.

[13] T. Verhoeff, Delay-insensitive codes-an overview, *Distributed Computing*, vol. 3, pp. 1-8, 1988.

[14] J. Youn and B. Bose, Some improved encoding and decoding schemes for balanced codes, *Proceedings Pacific Rim International Sym. on Dependable Comput.*, pp. 103 - 109, Dec. 2000.