Low Power Huffman Coding for High Performance Data Transmission

Chiu-Yi Chen, Yu-Ting Pai, and Shanq-Jang Ruan†
Department of Electronic Engineering
National Taiwan University of Science and Technology
No.43, Sec.4, Keelung Rd., Taipei, 106, Taiwan, R.O.C.
†E-mail: sjruan@mail.ntust.edu.tw

Abstract

In this paper, we propose a low power Huffman coding by reducing the number of switching activity during data transmitting. The low power Huffman coding consists of the following two steps: Firstly, reduce the switching activity between each symbol by changing the locations of leaves node in Huffman tree. Secondly, reconstruct the Huffman tree by properly changing interior nodes to decrease switching activities of each codeword. The experimental results show that the proposed Huffman coding algorithm can reduce switching activity up to 7.25% on average and 18.04% for the best case compared with traditional Huffman coding approach in executable files. Besides, the average power consumption reduction ratio is 3.81% for other files. As a result, the proposed algorithm is very useful for data transmissions over large self-capacitances wire.

1. Introduction

Huffman coding is one of the popular compression techniques [2] [14] [13]. Nowaday, many researches still discuss Huffman coding on different applications. For example, it can be applied in text, image, and video compression such as JPEG, MPEG2, and MPEG4, etc. Lakhani presented a minor modification to the Huffman coding of the JPEG baseline compression algorithm to exploit this characteristic [8]. Lee *et al.* proposed a new Huffman decoding method for the MPEG-2 AAC audio [9]. Bauer and Vinton addressed the optimization problem of minimizing the distortion subject to a rate constraint for an MPEG-4 Advanced Audio Coding encoder [1]. Therefore, Huffman coding plays a significant role in multimedia applications.

Recently data transmission and compression in portable device has become a popular technique, such as laptop computer, personal digital assistant (PDA), microdrive, and cellular phone. Because these applications are battery powered, reducing power dissipation in a transmission system is critical. Although many investigations have studied the various low power technologies on data transmission [15] [3] [4], little research has been devoted to Huffman coding for low power data transmission. In this paper, we will present a low power Huffman coding by reducing transmission power dissipation. Power dissipation is proportion to the average number of switching activity directly. Note that switching activity represents the logic transition (i.e. 0 to 1 or 1 to 0). Hence, the aim of the proposed algorithm is to decrease the number of switching activity in Huffman coding. The algorithm can be divided into two parts. Each part is designed for the different place of the transmitted-binary string. In the first part, the algorithm reduces the switching activity between each pair of symbols. We use the directed graph to analyze the relationship of each symbol. The location of each leaf node in the Huffman tree is corresponding to a symbol. Therefore, change the locations of leaves node by referring the result of the analysis can reduce the switching activity between each symbol. The propose of the second part is optimizing switching activities of each codeword in the binary string. We analyze the relationship between the location of interior nodes and codewords. Codewords in Huffman coding are determined by the structure of Huffman tree. Hence, the structure of Huffman tree will be reconstructed by properly changing interior nodes to reduce switching for low power.

2. Background

2.1. Source of Power Dissipation

The average power dissipation in a CMOS circuit can be described by a simple equation that summarizes the three most important contributors to its final value [12].

$$P_{avg} = P_{dynamic} + P_{short} + P_{leakage} \tag{1}$$

The three components are $dynamic$ ($P_{dynamic}$), $shortcircuit$ (P_{short}), and $leakage$ ($P_{leakage}$) power dissipation respectively.

```
FUNCTION Optimize leaf node ()
BEGIN
  construct_directedgraph();
  statistic_simplemodel();
  Hamilton path(node);
FOR i IN 1 TO totalnode − 1 LOOP
  IF node[i].front_bit == node[i].headbit
    node[i].inputform = condition1;
  ELSE
    node[i].inputform = condition2;
  END IF;
  IF node[i].tailbit == node[i].back_bit
    node[i].inputform = condition3;
  ELSE
        node[i].inputform = condition4;
  END IF;
END LOOP
FOR i IN 0 TO totalnode − 1 LOOP
   CASE (node[i].inputform)
    condition1:
    IF node[i].outputform == condition4
          find_to_exchang(node[i]);
        END IF;
        BREAK;
    condition2:
        IF node[i].outputform == condition3
          find_to_exchange(node[i]);
        END IF;
        IF node[i].outputform == condition4
          find_to_exchange(node[i]);
        END IF;
        BREAK;
   END CASE
END LOOP
RETURN(TREE);
END FUNCTION
```

Figure 1. Algorithm of optimizing.

Dynamic power dissipation, $P_{dynamic}$ is obtained from the switching activity of gate outputs. In today's VLSI circuits, dynamic power dissipation is the dominant fraction of average power dissipation(70%-90%) and can be described as follows [12]:

$$P = 0.5 \times C_L \times V_{dd}^2 \times E(sw) \times f_{clk} \tag{2}$$

where C_L denotes the loading capacitance, V_{dd} denotes the supply voltage, and f_{clk} denotes the clock frequency. These three parameters are primarily determined by the fabrication technology and circuit layout. $E(sw)$ is the average transition number per clock cycle (referred as the transition density). Obviously, equation 2 shows that the average power dissipation is proportional to the average switching activity. Hence, we focus on reducing the switching activity in this paper.

2.2. Construction of Huffman Tree

The process of Huffman coding will pass the given data twice. In the first pass, the statistic information of the transmitted data is collected and calculated to construct a tree. In the second pass, all symbols would be encoded and transmitted accordingly. In first pass, Huffman's algorithm is used to construct a tree. Initially, each node contains a symbol and its probability. There are four steps involved for the construction of the tree:

1. The two free nodes/trees with the lowest probability are combined to one tree.

2. A parent node is created, which is assigned to the sum of the probability these two child nodes.

3. One of the child nodes is designated as the path taken from the parent node when decoding the value 0 and other is arbitrarily set to the value 1.

4. Steps 1 to 4 are repeated until there is only one tree left.

2.3. The relationship between switching activity and Huffman coding

There is still some space to discuss in the structure of Huffman tree. Obviously, the different structures of Huffman tree will produce the different codewords for each symbol. Those codewords in the string will produce switching activities during data transmission. Take 2 bits codewords "00" and "01" for example, the switching number of "00" is 0, while the switching number of "01" is 1. According to the switching activity will dissipate the serious power in the technique of deep sub-micron, we exchange sibling nodes or equivalent probability nodes in Huffman tree to get a less-switching-activity encoding result. Therefore, we will apply this concept in the following section to construct the low power Huffman tree.

3. Preliminaries

The terms used in this paper are defined here: A set of binary sequences is defined as *code*. The individual members of the set called *codewords*. The *head bit* means the first bit in the codeword, and the last bit of the codeword is called the *tail bit*. For the adjacent symbols, the head bit in the back codeword follows the last bit of the front codeword.

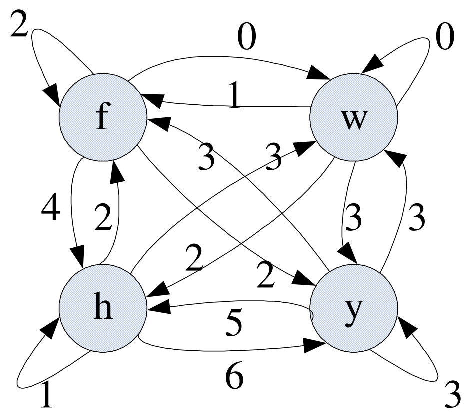


Figure 2. A directed graph.

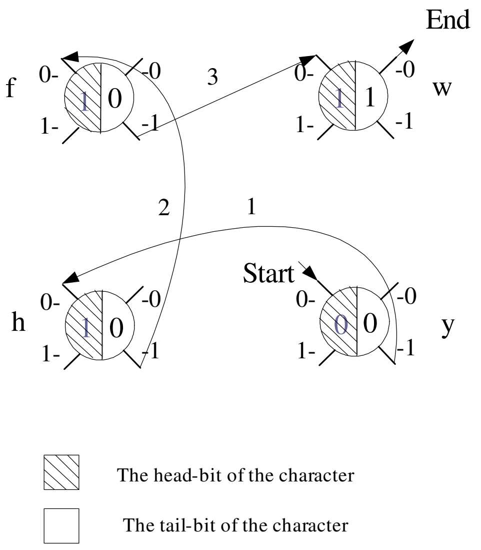


Figure 3. Simple statistical model.

4. Proposed scheme

4.1. Optimize leaves node

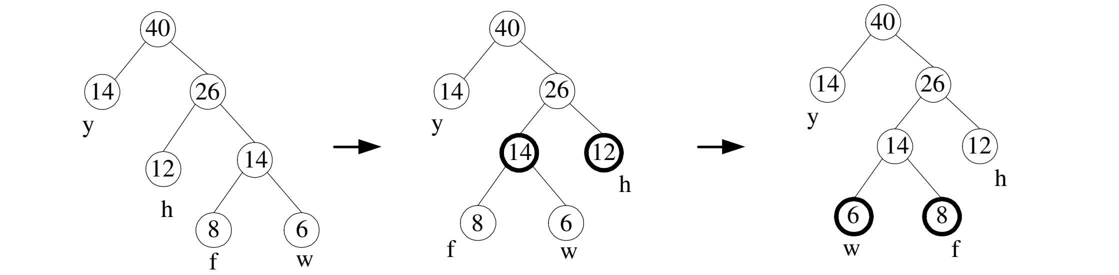
Before we discuss the switching activity between each pair of symbols, let us begin with a simple observation about the relationship between leaf nodes and the switching activity. When a file was encoded in Huffman code, each codeword of symbol is obtained by traversing the Huffman tree from the root node to the leaf node with respect to the symbol. Thus, these codewords will compose a long binary string. It is clear that changing the location of leaf nodes will affect the switching activity between any two adjacent symbols in the string. Therefore, the location of leaf nodes can be properly changed to reduce switching activities. For example, if the codeword of symbol 'h' is "0001" and 'e' is "010". The string of "he" well be "0001010". As a result, the there exist a switching activity between the tail bit of 'h' and the head bit of 'e'. In this condition, the sibling nodes of 'h' or equivalent probability nodes which tail bit is '0' will be found to exchange with the present location for 'h'. Consequently, it will reduce the number of switching activity. But, the symbol 'h' connects with many symbols in a encoded file. We have to analyze all associations between two adjacent symbols for finding the optimal coding. To solve this problem, we recognize all the head bit of the next codeword following the 'h', and figure out the amount of '1' and '0' following the 'h' respectively. If the result shows that the amount of '0' is more than '1', a node which tail bit is '0' will be found to exchange with 'h'.

However, each symbol may be connected with others. It will have a inconsistent problem in the association of each symbols, while we change the head bit or tail bit of a symbol. In the above example, the leaf node of 'h' has found the suitable location for less switching activities, after its location was changed. When other symbol changes its location in the next changing process, the codeword of the symbol would change their head bit or tail bit. It will cause the different association of 'h' in different places where the symbol connects with 'h'. Suppose that the amount of '1' is more than '0' in this condition for the symbol 'h', the analysis of this condition will be mutually inconsistent for the previous condition. It will result in the different outcomes of changing process.

To solve the inconsistency characteristic problem, the major part of the head bit and tail bit will be fixed by processing the highest probability symbol in the priority. The highest probability symbol affects the switching activity more than other lower ones. By fixing the major part in the binary string, it will lower the inconsistent condition for symbols. Therefore, each leaf node will be processed from the high probability to the low probability in this scheme.

The algorithm described above is represented in Figure 1. First, the association of each symbol can be described as the directed graph having the weight associated with each edge as shown in Figure 2.

Second, the directed graph is turned into the simple statistical model as shown in Figure 3. It is a convenient model to traverse for obtaining a Hamiltonian path. We will discuss it in detail later. The concept of the simple statistical model is to simplify all associations between adjacent symbols. There are two forms for each vertex/symbol: input form, output form. For the input form, the bit which exists in front of one symbol could be '1' or '0'. Hence, the input form will be divided into two types: 1-head bit (the tail bit of the front symbol is '1') and 0-head bit (the tail bit of the front symbol is '0'). For the output form, the bit which follows one symbol could be '1' or '0'. Therefore, the output form will be divided into two types: tail bit-1 (the head



Character	Probability	Code
y	14	0
h	12	10
f	8	110
w	6	111

Character	Probability	Code
y	14	0
h	12	11
f	8	100
w	6	101

Character	Probability	Code
y	14	0
h	12	11
f	8	101
w	6	100

Figure 4. Optimization of leaf node

bit of the back symbol is '1') and tail bit-0 (the head bit of the back symbol is '0'). It is obvious that all associations of each symbol could be simply described into four types. After that, each form's probability is counted by classifying the symbol's probability as shown in Table 1.

Third, each vertex is visited in Hamiltonian path to find a processed order of symbols. In the simple statistical model, a vertex is picked with highest probability input form. After that, a higher probability output (tail-0 or tail-1) of this vertex will be selected as its output. Each vertex will be selected by repeatedly following above steps until all vertices have been visited.

All relationships between two adjacent symbols will be explored, when all vertices have been visited in the Hamiltonian path. The first step to reduce the switching activity is to analyze each symbol's input form and output form as the following four conditions.

1. The front bit that exists in the input form is equal to the head bit of symbol.
2. The front bit that exists in the input form is not equal to the head bit of symbol.
3. The back bit that exists in the output form is equal to the tail bit of symbol.
4. The back bit that exists in the output form is not equal to the tail bit of symbol.

The condition 1 and 3 are the best association for symbols. The locations of leaf nodes do not need to be exchanged in the tree, since two adjacent bits are equal between two symbols. However, for the condition 2 and 4, the adjacent bits are not equal to each other. Hence, another interior/leaf node which has same conditions will be found to exchange with the present node in condition 2 or 4.

Table 1. Probability of different forms

symbol		**y**	**h**	**f**	**w**
Probability		14	12	8	6
codeword		0	10	110	111
Probability of inputform	**0-Head bit**	11	10	7	6
	1-Head bit	3	2	1	0
Probability of outputform	**Tail bit-0**	3	6	2	3
	Tail bit-1	11	6	6	3

In Figure 4, the location of leaf node was changed to reduce the number of switching activity base on Hamiltonian path. To begin with checking the symbol 'y', the input form and output form of symbol 'y' meets the condition 1 and 4. Hence, the tail bit of symbol 'y' needs to be changed. Unfortunately, there is no another node which has the same condition with the symbol 'y' in the tree. Therefore, we skip the symbol 'y' and pick the next symbol 'h'. According to the output form of 'h' meets condition 4, the symbol 'h' needs to be exchanged with its sibling node to reform its output. Next, the symbol 'f' is picked to check its condition. Then, we found that 'w' has the same condition with 'f'. As noted above, the symbol 'f' is exchanged with 'w'. In the long run, all the symbols in the tree have been checked, and the Huffman tree is reconstructed as a new structure. It will generate the best association of each symbol to reduce switching activities.

4.2. Optimize interior nodes

In the section 4.1, the association of each symbol was discussed to reduce switching activities. But there are still more space to reduce switching activities of each codeword. In the section 2.2, we have mentioned the construc-

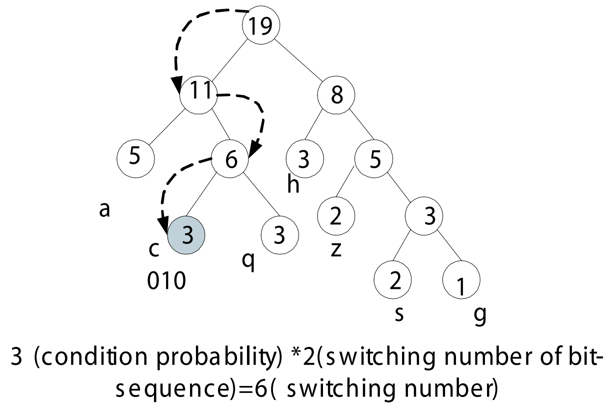
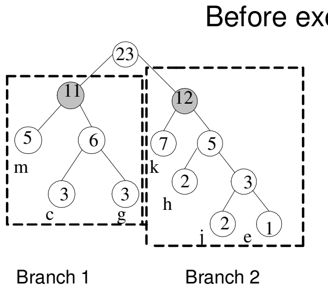

19
11
8
5
6
3
5
a
h
c
3
3
2
3
q
z
010
2
1
s
g
3 (condition probability) *2(switching number of bit-sequence)=6(switching number)

Figure 5. Calculate the switching number



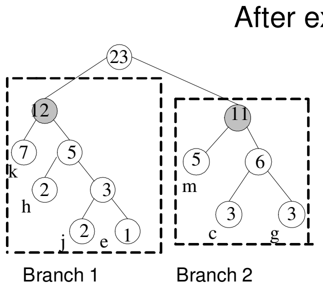
Character	Probability	Code
m	5	00
c	3	010
g	3	011
k	7	10
h	2	110
j	2	1110
e	1	1111

Branch 1: 5*0+3*2+3*1=9
Branch 2: 7*1+2*1+2*1+1*0= 11
Total switching number=9+11=20

Figure 6. Total switching number before exchanged

tion of Huffman tree. Codewords produced by Huffman approach coding are determined by the structure of Huffman tree. Furthermore, an interior node to be the parent contains many relative nodes which belong to the same branch. Suppose the interior node changes its location in the tree, it will affect switching activities of codewords which belong to the same branch. The algorithm in Figure 8 is used to analyze the switching number of two branches, and properly change interior nodes to reduce the switching activity of each codeword. Note that pre_o() means the preorder function.

We start at the root node and pick two children interior nodes to calculate every leaf node's switching number under these two interior nodes respectively, as shown in Figure 5 and Figure 6. Next, exchange the sibling nodes to produce another new structure. Switching number is recounted in this structure as shown in Figure 7. If the total switching number of the original structure is more than that of the exchanged. The exchanged structure will be adopted instead of the original one. Take Figure 6 and Figure 7 for exam-



Character	Probability	Code
m	5	10
c	3	110
g	3	111
k	7	00
h	2	010
j	2	0110
e	1	0111

Branch 1: 7*0+2*2+2*2+1*1=9
Branch 2: 5*1+3*1+3*0=8
Total switching number=9+8=17

Figure 7. Total switching number after exchanged

ple, the interior node "11" and "12" are exchanged, because the total switching number after exchange is smaller. After that, the following interior nodes will be picked to analyze in preorder until all interior nodes have been traversed. When this procedure is done, the path from root node to the corresponding leaf node will be optimized for each codeword. Therefore, this approach can be used to reduce the switching activity of each codeword.

5. Experimental results

The proposed method is used to reduce the number of switching activity during the data transmission. To assess the effectiveness of our proposed method, we implement our algorithm and the original Huffman coding in C++ respectively. Therefore, we can get the power reduction rate by comparing both results of switching activity.

In our experiments, we compare the original Huffman code with our proposed method in both the switching number and power consumption. In order to calculate the realistic value, power consumption of the bus line is measured by HSPICE. Note that the RLC model used were characterized by FastHenry [6] and FastCap [7]. For this experiment, the bus is modeled with 1000 μm long, width 0.14 μm wide and 0.25 μm height, where our bus line is partitioned into four segments of RLC model for accuracy. We encode several files of different nature by the original Huffman code and our proposed method respectively. In the Table 2, the encoded file length is measured by bytes, and the data of each byte is considered to be a symbol. The reduction ratio is defined as follow:

$$Reduction\ Ratio = 1 - \frac{Optimized\ Switching\ Number}{Original\ Switching\ Number} \quad (3)$$

```
FUNCTION pre_o (val root <node pointer>)
BEGIN
  IF root isn't null
    IF root has child nodes
      proc(root);
      proc(root.leftsubtree);
      proc(root.rightsubtree);
    END IF;
  END IF;
  RETURN;
END FUNCTION

FUNCTION proc (val root <node pointer>)
BEGIN
  branch1 = switching_number(root.left);
  branch2 = switching_number(root.right);
  total1 = branch1 + branch2;
  exchange_node(root.left,root.right);
  branch1 = switching_number(root.left);
  branch2 = switching_number(root.right);
  total2 = branch1 + branch2;
  IF total1 < total2
    exchange_node(root.left,root.right);
  END IF;
  RETURN(TREE);
END FUNCTION
```

Figure 8. Interior node optimization algorithm.

Besides, the TRR and RRR are the abbreviations of "Theoretical Reduction Ratio" and "Realistic Reduction Ratio", respectively.

As shown in Table 2, the realistic value is close to the theoretical value. It means that our proposed algorithm is very accurate. In addition, the experimental results show that the efficiency of executable files is better than others. For executable files, this algorithm can reduce the switching number up to 7.25% on average and 18.04% for the best case compared with traditional Huffman coding approach. This is because that every executable file has the same amount of symbols, and the average length of those codewords is shorter than other files. In general case, the average reduction ratio is 3.81% for other files. We may say that the proposed method can reduce the power consumption effectively.

5.1. Conclusion and Future works

In this paper, we proposed a technique to to reduce the transmission power dissipation of Huffman code. The leaf nodes and interior nodes were changed by analyzing the association of each symbol and the structure of Huffman tree respectively. The major contribution of this paper is that proposed method can decrease the number of switching activity to achieve low power dissipation during data transmission. The experimental results show that the average reduction ratio of switching activity is 7.25%, and the best case would be up to 18.04% for executable files.

Furthermore, The adaptive Huffman coding (AHC) has been discussed widely [11] [5] [10]. In AHC, the coded tree is reformed after input a symbol. Thus, we can refine our algorithm to the dynamic coding for low power dissipation in the future work.

Table 2. Switching number compared with the original Huffman coding

File Name	File length	Original switching number	Optimized switching number	TRR (%)	RRR (%)
INSTALL.EXE	34844	85017	69685	18.03	17.91
Java.exe	45162	124003	101693	18.04	17.95
k1_upx.exe	221185	692761	643252	7.14	7.11
DVDX.exe	2134017	5528575	5120638	7.37	7.12
LENA.RAW	65537	245233	236186	3.68	3.66
file1.txt	14244	31603	29994	5.09	5.05
file2.txt	22099	48956	47082	3.82	3.81
testmusic.mp3	4709441	18734698	18349266	2.65	2.61

References

[1] C. Bauer and M. Vinton. Joint optimization of scale facters and huffman code books for mpeg-4 aac. *IEEE Transactions on signal processing*, 54(1):177–189, Jan. 2006.

[2] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text compression*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[3] S. Gomi, K. Nakamura, H. Ito, H. Sugita, K. Okada, and K. Masu. High speed and low power on-chip micro network circuit with differential transmission line. In *Proceedings of the 2004 International Symposium on Low Power Electronics*, pages 173–176, Nov. 2004.

[4] S. Gregori, Y. Li, H. Li, J. Liu, and F. Maloberti. 2.45 ghz power and data transmission for a low-power autonomous sensors platform. In *Proceedings of the 2004 International Symposium on Low Power Electronics*, pages 269–273, Aug. 2004.

[5] M. Y. Javed and A. Nadeem. Data compression through adaptive huffman coding scheme. In *Proceedings of TENCON 2000*, volume 2, pages 187–190, Sep. 2000.

[6] M. Kamon and Fasthenry. Fasthenry: A multiple-accelerated 3-d inductance extraction program. *IEEE Transactions on Microwave Theory Technology*, 42(9):1750–1758, Sep. 1994.

[7] M. Kamon, M. Tsuk, and J. White. Fastcap: a multipole accelerated 3-d capacitance extraction program. *IEEE Transactions on Computer-Aided Design of Integrated*, 10(11):1447–1459, Nov. 1991.

[8] G. Lakhani. Optimal huffman coding of dct blocks. *IEEE Transactions on circuits and systems for video technology*, 14(4):522–527, April 2004.

[9] J. S. Lee, J. H. Jeong, and T. G. Chang. An efficient method of huffman decoding for mpeg-2 aac and its performance analysis. *IEEE Transactions on speech and audio processing*, 13(6):1206–1209, Nov. 2005.

[10] L. Y. Liu, J. P. Wang, R. Jen, and J. Y. Lee. On the concurrent updata and generation of the dynamic huffman code. *IEEE Transactions on signal procession*, 44(8):2082–2085, Aug. 1996.

[11] W. W. Lu and M. P. Gough. A fast-adaptive huffman coding algorithm. *IEEE Transactions on communications*, 41(4):535–538, April 1993.

[12] M. Pedram. Power minimization in ic design: Principles and applications. *ACM Transactions on Design Automation of Electronic Systems*, 1(1):3–56, Jan. 1996.

[13] W. B. Pennebacker and J. L. Mitchell. *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.

[14] I. H. Witten, A. Moffat, and T. C. Bell. Managing gigabytes: Compressing and indexing documents and images. *IEEE Transactions on Information Theory*, 41(6):2101–2102, Nov. 1994.

[15] D. C. Yates, A. S. Holmes, and A. J. Burdett. Optimal transmission frequency for ultralow-power short-range radio links. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 51(7):1405–1413, July 2004.