Static Power Optimization for Reconfiguration of Hand Held Devices

Shweta Loonkar
Department of Computer Engineering D.J. Sanghvi College of Engineering Mumbai, India

Lakshmi Kurup
Department of Computer Engineering D.J. Sanghvi College of Engineering Mumbai, India

ABSTRACT
It has been widely seen that multimedia application has increased in hand held devices such as mobile devices, cellular phones, PDA's , mobile audio / video player etc. These embedded devices and applications need a huge amount of power to function so improvement in power in these devices has turned out an important issue. This paper presents a novel approach for reducing the bit-width of the data used for the dynamic reconfiguration of the hand held devices. Run time dynamic reconfiguration of hand held devices to maximize power according to user is a significant area for research. Remote reconfiguration is possible only when Request Processing Time is less. This is achievable only when majority of optimizations is performed statically. The bit streams available after the static analysis and preprocessing are used further for dynamic optimizations which will greatly reduce the runtime of the applications which further reduces the power consumed by the devices. Thus the paper aims to propose a new set of preprocessing algorithm in which the variables are identified based on different usage patterns and the generated bit stream is further compressed using the Huffman compression and Dynamic Huffman Coding.

General Terms
Client-Server Model, Preprocessing Algorithm, Huffman Compression, Dynamic Huffman Coding.

Keywords
Dynamic Reconfiguration, Request Processing Time, QIBO, QDBO, Critical Variables, Non-Critical Variables.

1. INTRODUCTION

Mobile devices work with dynamism i.e. they have to deal with the unpredicted network outage or should be able to switch to a different network, without changing the application. Reconfigurable systems have the potential to operate efficiently in these dynamic environments [1]. Dynamic Reconfiguration allows almost any customized design to be placed inside the hardware or an existing design to be updated or modified [2]. Remote reconfiguration is a strong solution for power optimization. There are various proposed algorithms for power minimization, but the only weak point in these algorithms is that the request processing time is more than required. Request Processing Time is a time required in sending the request, processing at remote server, transmission of new bit stream to mobile device and reconfiguration of mobile device as per the requested quality (RqoS) must be less. Thus the proposed paper exhibits a satisfactory real time solution for operational performance of hand held devices [3].

Contributions: We therefore summarize the contributions of this paper as:
1) Bit-width optimization performed using QIBO and QDBO.
2) Preprocessing algorithm applied to the resulting variables and further compressed using Huffman compression algorithm.
3) Dynamic Huffman Coding has been compared with Static Huffman coding for minimization of execution time of preprocessed variables.

The remainder of the paper is structured as follows. We start by introducing related work in Section 2. Next, Client-server model of reconfigurable hand held framework is described in Section 3. The Preprocessing Algorithm, which identifies the critical and non-critical variables, is described in Section 4. In section 5 experimental results are shown before and after inclusion of preprocessing algorithms using Huffman compression and dynamic Huffman coding, conclusion is presented in Section 6, Section 7 includes the future scope, Section 8 gives the acknowledgement and finally in Section 9 references are mentioned.

2. MOTIVATION AND RELATED WORK

Improving the efficiency of battery usage in hand held devices (mobile devices) is a significant area of research. Some of the known techniques include voltage scaling [4], variable frequency of operation [1], battery aware task scheduling [3] and dynamic power management [5]. Significant improvement in battery efficiency is reported by using these power management techniques for processor or ASIC based portable systems. Another popular approach [6] for power minimization at the same level is dynamic power management, which aims to reduce power consumption of integrated circuits by selectively shutting down idle components at any instant. Extensive research has been carried out on power optimization in high-level synthesis for ASIC designs [7, 8, 9]. It is customary to note that the dynamic

reconfigurability character of FPGA opens several new avenues for power optimization. Reconfigurability can actually be applied at many layers in a system and at multiple levels of granularity, in which each has its own preferred and optimal application domain [10]. FPGAs are particularly useful for applications with bit-level operations [11]. Bit width analysis and optimization of all the operands and operators in any processing algorithm serve as the foundation for data path optimizations in the low power design flow [12]. Optimization of functional unit and data-path width [13] statically and dynamically, also known as static and dynamic bit-width optimizations, has been addressed in various research works. Previous works [13], [12] show that reduction of data-path width can substantially reduce the area and energy required by the system, although the reduction of width beyond a threshold limit may adversely affect the quality and accuracy of processing output. Bit-width analysis and optimization of all the operands and operators in any processing algorithm serve as the foundation for data-path optimizations in the low power design flow [14]. The fact that optimizing bit-width of data-path for each given application is an effective technique to reduce leakage power dissipation of the whole embedded systems has been explored. Bit-width analysis is a concept to circumvent the unnecessary bit computations by determining the Bit-widths of operands and identifying the unused and insignificant bits, defined and used in high-level languages.

3. CLIENT SERVER MODEL OF RECONGIFURABLE MOBILE FRAMEWORK

The network view of framework explains that the client is the mobile device user while remote server provides the reconfiguration support to the user. Whenever user feels that the device is running out-of-power, he, initiates the process by selecting the required level of quality, which is then converted to technical specifications by client model of the framework and the parameters are sent to remote server through communication network as shown in Fig 1. Server calculates the new optimized design and extracts the reconfigured bit-stream, which then sent to the client, reconfigures the device for reduced power consumption and compromised quality. The accuracy is reduced further till the limit of quality parameters is attained.

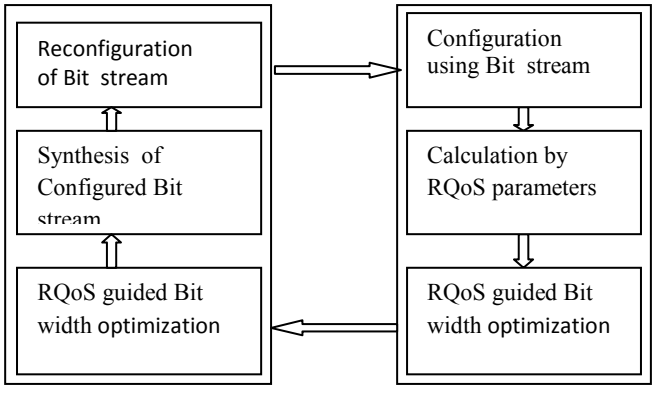

Figure 1 Client-Server Network Model Framework

In this paper it will be seen that often the most efficient implementation of an algorithm in the reconfigurable hardware is one in which the bit width of each internal variable is fine-tuned to the best precision. Bit Width Optimization is done in two

3.1 Quality Independent Bit Width Optimizations (QIBO)

QIBO is performed as a static process i.e. it is done only once for any source application and the results are stored for subsequent use by Quality Driven Optimization phase. QIBO phase is intended to remove the bits from Most Significant (MSB) and Least Significant (LSB) ends of a variable that do not introduce any error or inaccuracy. The QIBO is loss less approach. Algorithm for QIBO is as:

1. Assigning zero to lsb_saving and msb_saving vectors.
2. Removing one bit from LSB and MSB end of any variable (i.e. increasing the value of lsb_saving vector or msb_saving vector at the position corresponding to the variable under consideration, by one) and calculating the mean square error [15].
3. If MSE is found to be lower than the Min_Err value the algorithm proceeds towards the removal of further next bit. The process is repeated until the MSE remains lower than the Min_Err.
4. Similar steps are repeated for MSB end.
5. Repeat the steps 1-4 for all n variables in the application.

3.2 Quality Driven Bit Width Optimizations (QDBO)

The results of QIBO are made available for the QDBO at remote server after receiving the required quality level from the client. QDBO is a lossy approach and intends to enhance the statically determined bit widths (determined through QIBO) by allowing deterioration in quality up to an externally specified acceptable level.

1. In this algorithm n is the total number of variables in the application being optimized? The QDBO uses lsb_saving vector from QIBO.
2. Removing one bit from LSB end of any variable (i.e. increasing the value of lsb_saving vector at the position corresponding to the variable under consideration, by one) and calculating the mean square [15].
3. If MSE is found to be lower than the Min_Err value the algorithm proceeds towards the removal of further next bit. The process is repeated till the MSE remains lower than the Min_Err.
4. Repeat the steps 1-4 for all n variables in the application.

4. PREPROCESSING

To further reduce the preprocessing time of the variables obtained from QIBO and QDBO, pre-processing algorithm has been proposed in this paper. The overall aim of this paper can be stated as "***To generate an optimum bit stream, in response to desired quality request from user, using Preprocessing and Huffman Compression algorithms and Arithmetic Coding, which can dynamically reconfigure the remote runtime of hand held devices for improved power efficiency against quality***

compromise". In this framework source code of algorithm to be optimized is preprocessed before applying QIBO and QDBO (Figure 2).

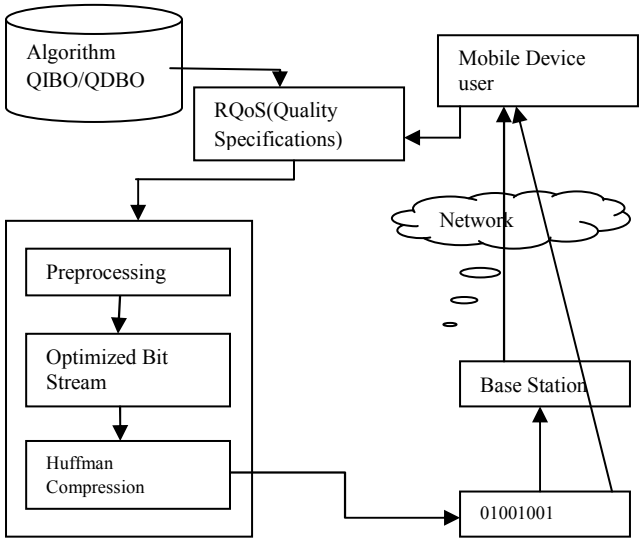

Figure 2 Preprocessing at Remote Server

Initially variables are identified and their bit width is documented (as per the language specification). E.g. source code to be analyzed

Example 1:

```
# d e f i n e W2 2 6 7 6
# d e f i n e W6 1 1 0 8
x8 = x2 + x1;
i f (x0)
{x1 = W6*(x3+x2);
}
x2 = x1 − (W2+W6 ) * x2 ;
x3 = x1 + (W2−W6 ) * x3 ;
x1 = x4 + x6 ;
x4 −= x6 ;
f o r ( k = 0 ; k < x7 ; k++)
{
x6 = x5 << x2 ;
}
```

Identified variables and their bit widths from above source code are documented in the Table 1.

Table 1 Documentation of Variables and their Bit-widths

Variable	X0	X1	X2	X3	X4	X5	X6	X7	X8	W2	W6	k
Bitwidth	32	32	32	32	32	32	32	32	32	12	11	32

Table 1 data can be presented as Initial Variable Set (*Vi*).

Vi = {x0, x1, x2, x3, x4, x5, x6, x7, x8, W6, W2, k} ***(1)***

|Vi| = n = 12 ***(2)***

The optimization time requires by number of variables in
algorithm as per eq. (3).

Execution Time = n *b where ***(3)***

- **n** is the number of variables and
- **b** is the average bit width.

Therefore to reduce the execution time number of variables to be analyzed should be reduced. During preprocessing, variables are flagged as critical and non-critical. Critical variables are variables whose accuracy must not be alerted in order to retain functionality. These critical variables can be avoided in further analysis to reduce the execution time. Their usage patterns can identify critical variables. If variable

1. Participates in comparison statement.
2. is used to calculate at least 80% of output variables.
3. are of Boolean type.
4. are a type on which number of loop iteration depends?

By analyzing variables based on usage pattern some of the variable in *Vi* can be identified as Critical Variable set (*Vcvup*)

Vcvup ={x0, x2, x7, k} ***(4)***

|Vcvup| = m = 4 ***(5)***

The Approximate Effective Variable Set (Vae) is

Vae =Vi−Vcvup ***(6)***

Vae = {x1, x3, x4, x5, x6, x8, W6, W2} ***(7)***

|Vae| = n−m = 12- 4 = 8 ***(8)***

Example 2: For a statement x8 = W3*(x6+x7) the injected code for virtual bit width reduction is

```
Mask(&x6 ) ;
Mask(&x7 ) ;
x8 = W3 * ( x6 + x7 )
Mask(&x8 )
f u n c t i o n Mask ( i n t * x )
{
i f (* x > 0 )
*x = ( * x & Ma s k Pa t t e r n ) ;
e l s e
{
*x = −1 * *x ;
*x = ( −1 ) *(* x & Ma s k Pa t t e r n ) ;
}
}
```

Now one variable from above set is chosen and usefulness of its bits is verified. To verify usefulness, we inject some code to reduce bit width virtually (during calculation a block of bits is masked to produce an effect of bit width reduction). Masking of block of bits can be done from LSB side. By masking we reduce the precision of variable. Always block of bits is masked and block of bits start from LSB as shown in Figure 3. Advantages of masking process is that it provides a simulation of bit width specific calculation at system level, thus a great potential for considering feedback and power saving without going to low- level.

Figure 3 Masking of Variables

The modified program is executed and output is observed. If output is as accurate as with standard code, i.e. ***"Mean Square Error (MSE)"*** remains less than the visualized error threshold **(*MSE Minimum Threshold*),** the bits are flagged as *unused*. This process is called ***"Error Contribution Analysis"*** and repeated for every variable in *Vae*. Output of this process is ***"Error Contribution Matrix",*** as shown in Table 2. Here MSE is calculated by adding difference i.e. error between all pixels of produced original and new image as given by-

$$MSE = \frac{1}{N^2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\{f'(i,j) - f(i,j)\}^2 \quad (9)$$

Here masked column shows the number of bits masked from LSB Side and MSE introduced due to masking of bits. Thus we have an error contribution matrix for all variables *Vae* set. If error contribution of LSB bit is greater than threshold (*TCriticalError*) then it can be flagged as critical variable.

Table 2 Error Contribution Matrix of Variable x0

Masked	MSE	Masked	MSE	Masked	MSE	Masked	MSE
1	0	9	0.1362	17	191.453	25	1959.56
2	0	10	0.2352	18	393.053	26	1959.56
3	0	11	0.4259	19	859.466	27	1959.56
4	0	12	0.7683	20	1309.61	28	1959.56
5	0.0013	13	1.6258	21	1959.56	29	1959.56
6	0.0019	14	4.2247	22	1959.56	30	1959.56
7	0.0039	15	14.2902	23	1959.56	31	1959.56
8	0.0966	16	54.2229	24	1959.56	32	1959.56

E.g. in Table 3 error introduced due to masking of first LSB bit is listed. Here value of *TCriticalError* is 10.0000 hence we can flag *x*5 as critical variable.

Table 3 Error Contribution due to first bit for all variables of Approximate Effective Variable Set (*Vae*)

Variable	X0	X1	X3	X4	X5	X6	X8	W6	W2
Error	0.00	0.00	0.00	0.002	14.9632	0.0011	0.00	0.0016	0.0287

By including all identified critical variables we can reduce *Vae* to Effective Variable set (*Veff*). *Veff* calculated for above example is

Veff =Vae− {x5} **(10)**
Veff = {x1, x3, x4, x6, x8, W6, W2} **(11)**
|Veff | = 7 **(12)**

Thus preprocessing calculation takes significant time but as these are static tasks, these make the dynamic process faster.

5. EXPERIMENTATION AND RESULTS

In this paper we have explored the preprocessing module in which processing of variables are done on various usage patterns mentioned above. In order to demonstrate the importance and benefits of the quality independent and quality driven bit-width optimizations in terms of performance and power consumption we have analyzed Huffman compression and Dynamic Huffman Coding in MPEG-2 decoding algorithm [16] has used.

5.1 Compression Using Huffman Algorithm

Compression is a technology for reducing the quantity of data used to represent any content without excessively reducing the quality of the image. It also reduces the number of bits required to store and/or transmit the digital data (Figure 4).

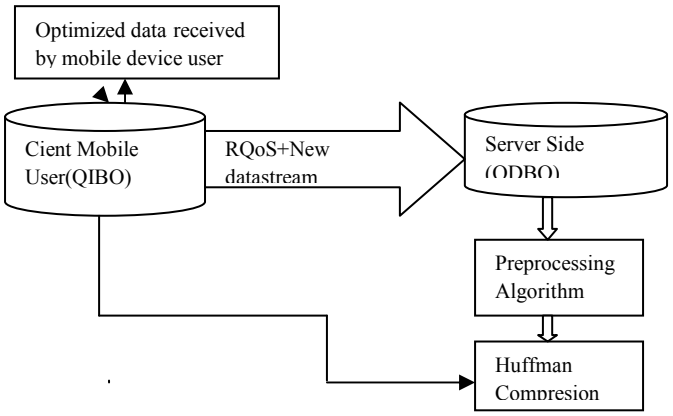

Figure 4 Huffman Compression of Pre-processed variables

Huffman coding [17] is an entropy encoding algorithm used for lossless data compression. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix-free code [18] (that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol). Here our preprocessed sets of variables are
{*x*1, *x*3, *x*4, *x*6, *x*8, *W*6, *W*2}. To generate a Huffman prefix code we traverse the tree to the value we want, outputting a 0 every time we take a left-hand branch and a 1 every time we take a right-hand branch shown in Figure 4. Each preprocessed variable

is an integer value i.e. 32*7=224 bits in length. Considering all the leaf nodes the partial tree for our preprocessed variables can be represented with 010001010110011011110**,** which is 21 bits long. So in our case, use of Huffman codes saved 203 (224-21) bits which is around 80-90% of the total preprocessed data shown in Table 4).

Table 4 Pre-processed Variables and their Huffman Codes

Preprocessed Variables	Frequency of Occurrence	Value	Huffman code
x8	1	1	0100
x4	1	2	0101
w2	2	3	011
x3	3	4	00
x6	3	5	110
w6	3	6	111
x1	5	7	10

5.1 Dynamic Huffman Coding

In this paper we have also done the analysis for Dynamic Huffman coding [19] as the performance of dynamic coding is better, even though the average code length of ordinary Huffman code is less. The problem with static coding is that the tree has to be constructed at the client side and sent to the server. The tree may change because the frequency distribution of the variables may change. Since the tree in dynamic coding is constructed on the server side as well, there is no need to send it again from the client side.

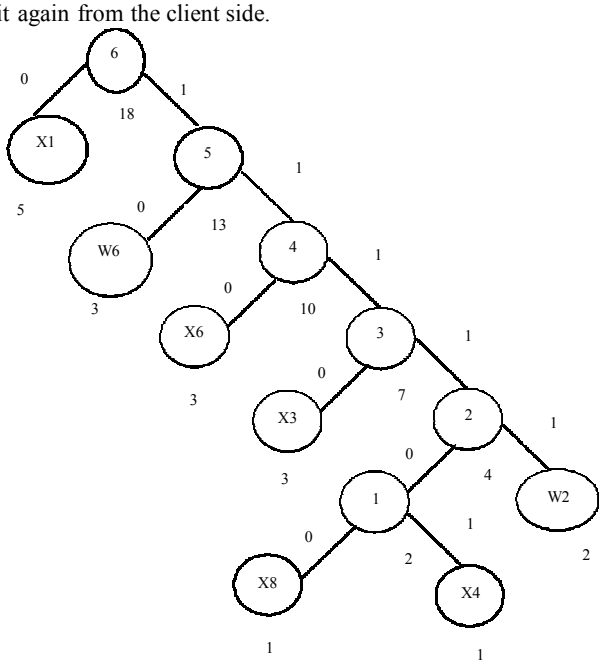

Figure 5 Dynamic Huffman Code Tree

The paths from the root node to the leaf nodes define the code word used for the corresponding symbol based on the tree following code word are generated given in Table 4. Thus the dynamic codeword for the preprocessed variable set is 27 bits long which is actually greater than the average code length of general Huffman code. But as the code words are generated dynamically at the server side, the transmission delay and propagation delay can be reduced to a greater extent.

Table 5 Pre-processed Variables and their Dynamic Huffman Codes

Pre-processed Variable	Frequency	Dynamic Huffman Code Word
X8	**1**	**111100**
X4	**1**	**111101**
W2	**2**	**11111**
X3	**3**	**1110**
X6	**3**	**110**
W6	**3**	**10**
X1	**5**	**0**

6. CONCLUSION

Bit width analysis and compile time optimizations have emerged as a key technique for power and performance aware optimizations in future digital systems design. This paper primarily targets to do a Critical Variable Partitioning using Preprocessing Algorithm and Huffman compression algorithm to minimize the total time taken in execution of a program and thereby minimizing the power consumption. Comparative analysis of dynamic Huffman coding with static Huffman coding states that the former reduces transmission time to a greater extent; even though the latter has a better average compression rate .Reduced power consumption will result into increased standby time of mobile devices.

7. FUTURE SCOPE

This work can be implemented using hardware accelerators in real time application in order to reduce the Request Processing Time of Mobile Devices.

8. ACKNOWLEDGEMENT

We are extremely thankful to Dr. Hiren Joshi for his help in understanding and developing a concept of bit-width analysis and power optimization in mobile devices and for their many discussions and useful feedback in writing this paper.

9. REFERENCES

[1] Martin, T. L., Balancing Batteries, Power, Performance System Issues in CPU Speed-Setting for Mobile Computing. Ph.D. dissertation, Carnegie Mellon University (1999).

[2] Verma, S.S., Joshi, H., Sharma, G.K.: Quality Driven Dynamic low Power Reconfiguration of Handhelds, In

Proceeding of International Workshop on Applied Reconfigurable Computing, ARC 2006, Delft, Netherland (March 2006).

[3] Luo, Jha, N.K., Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems, In Design Automation Conference, pp. 444–449 (2001).

[4] M. Pedram and Q. Wu, "Design Considerations for Battery-Powered Electronics," in Proceedings of the Design Automation Conference, 1999, pp. 861–866.

[5] L. Benini, A. Bogliolo, and G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," in IEEE Trans. on VLSI Systems, vol. 8, issue. 3, 2000, pp. 299–316.

[6] G.A.Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. "Policy Optimization for Dynamic power management", In DAC '98: Proceedings of the 35th annual conference on Designautomation, pages 182–187, New York, NY, USA, 1998, ACM Press.

[7] A. Raghunathan and N.K. Jha, "Behavioral Synthesis for Low-power", Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computer & Processors, pages 318–322, Washington, DC, USA, 1994, IEEE Computer Society.

[8] P. Kollig and B.M. Al-Hashimi,"A New Approach to Simultaneous Scheduling, Allocation and Binding in High-level Synthesis", in Proc. of IEEE Electronics Letters, vol. 33, Aug 1997.

[9] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R.W.Brodersen, "Optimizing Power Using Transformations", in Proc. of IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 14, no. 1,pp. 12-31, Jan. 1995.

[10] Rabaey Jan M., "Reconfigurable Computing: The Solution to Low Power Programmable DSP", *Proceedings 1997 ICASSP Conference*, Munich, April 1997.

[11] Gerard J.M. Smit, Paul J.M. Havinga, Lodewijk T. Smit, Paul M. Heysters, Michel A.J. Rosien "Dynamic Reconfiguration in Mobile Systems", *University of Twente department of Computer Science Enschede, the Netherlands*.

[12] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T.Okuma, and H. Yasuura, "Variable size analysis for data path width optimization". In APCHDL 98: Proceedings of the Asia Pacific Conference on Hardware Description Languages, pages 69–74, July 1998.

[13] S. Mahlke, Bit width Cognizant architecture synthesis of custom hardware accelerators. In Computer-Aided Design of Integrated Circuits and Synthesis, volume 20, pages 1355–1371, November 2001.

[14] Yun Cao Hiroto, Leakage power reduction using bitwidth optimization.

[15] J. Bins, B. Draper, W. Bohm, and W. Najjar, Precision vs. error in jpeg compression, 1999.

[16] JPEG, Joint photographic experts group's image compression standard, http://www.jpeg.org.

[17] Pham, H.-A., Bui, V.-H., Dinh-Duc, A.-V., An Adaptive, Memory-Efficient and Fast Algorithm for Huffman Decoding and Its Implementation, In: ACM International Conference, Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, Seoul, Korea, vol. 403, pp. 275–279 (2009).

[18] Sharma, M., Compression Using Huffman Coding, IJCSNS International Journal of Computer Science and Network Security 10(5) (May 2010).

[19] Binary essence at:
http://www.binaryessence.com/dct/en000097.html