American Journal of Applied Sciences 8 (11): 1169-1175, 2011
ISSN 1546-9239

A Single Core Hardware Module of a Data Compression Scheme Using Prediction by Partial Matching Technique

[1]Jubayer Jalil, [2]Md. Mamun, [1]Mohd. Marufuzzaman and [1]Hafizah Husain
[1]Department of Electrical, Electronic and Systems Engineering,
[2]Systems Design Lab,
Universiti Kebangsaan Malaysia, 43600, UKM, Bang, Selangor, Malaysia

Abstract: Problem statement: Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth. For effective data compression, the compression algorithm must be able to predict future data accurately in order to build a good probabilistic model for compression. Lossless compression is essential in cases where it is important that the original and the decompressed data be identical, or where deviations from the original data could be deleterious. **Approach:** Prediction by Partial Matching (PPM) data compression technique had utmost performance standard and capable of very good compression on a variety of data. In this research, we had introduced PPM technique to compress the data and implemented the algorithm on Altera FLEX10K FPGA device that allows for efficient hardware implementation. The PPM algorithm was modeled using the hardware description language VHDL. **Results:** Functional simulations were commenced to verify the functionality of the system with both 16-bit input and 32-bit input. The FPGA utilized 1164 logic cells with a maximum system frequency of 95.3MHz on Altera FLEX10K. **Conclusion:** The proposed approach is computationally simple, accurate and exhibits a good balance of flexibility, speed, size and design cycle time.

Key words: Prediction by Partial Matching (PPM), expensive resources, statistical modeling technique, original algorithm, lossless compression especially, Field-Programmable Gate Arrays (FPGA)

INTRODUCTION

Data compression is an essential process due to the need to reduce the average time required to send messages and reduce the data size for storage purposes. The ultimate goal of data compression is to represent an information source, e.g., a text file, binary information, an image or a video signal, as accurately as possible using the fewest possible number of bits (Cleary and Witten, 1984). There is a vital need for lossless compression especially for text and binary compression as it is important to ensure that the restructured text is identical to the original text, because a very small difference or variation in statement can lead to total different meaning. PPM is a "finite context" statistical modeling technique that can be viewed as blending several "fixed-order context" models to predict the next character in the input sequence. The "Prediction by Partial Matching" data compression scheme is capable of very good compression on a wide variety of source data. The adaptive nature of the scheme and the flexibility afforded by arithmetic coding, mean that an effective compression model will be built for any input file that is reasonably homogeneous (Moffat, 1990). The original algorithm was first published in 1984 by Cleary and Witten (1984) and a series of improvements was described by Moffat (1990) culminating in a careful implementation, called PPMC, which has become the benchmark version. This still achieves results superior to virtually all other compression methods, despite many attempts to better it.

Developed PPMZ which uses an adaptive second level model to estimate the optimum value as a function of the order, the total character count, number of unique characters and the last one or two bytes of context. Shkarin (2002) developed PPMII which is similar to PPMZ as it also uses a secondary escape model. PPMII does not use statistics from the longest matching context. Instead, PPMII inherits the statistics of shorter contexts to set the initial estimate when a longer context is encountered for the first time. PPMONSTR and PPMD are based on PPMII. PPMONSTR is a variation of PPMD that trades compression rate for execution speed. This project will concentrate on the original

Corresponding Author: Jubayer Jalil, Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia

version of PPM. Other methods such as those based on Ziv and Lempel (1977; 1978) are more commonly used in practice, but their attractiveness lies in their relative speed rather than any superiority in compression. Indeed, their compression performance generally falls distinctly below that of PPM in practical benchmark tests (Bell *et al*., 1990).

The Field-Programmable Gate Arrays (FPGA) offers a potential alternative to speed up the hardware realization (Coussy *et al*., 2009; Marufuzzaman *et al*., 2010; Reaz *et al*., 2007a). From the perspective of computer-aided design, FPGA comes with the merits of lower cost, higher density and shorter design cycle (Choong *et al*., 2005; Akter *et al*., 2008). It comprises a wide variety of building blocks. Each block consists of programmable look-up table and storage registers, where interconnections among these blocks are programmed through the hardware description language (Reaz *et al*., 2003; 2004; 2005). This programmability and simplicity of FPGA made it favorable for prototyping digital system. FPGA allows the users easily and inexpensively realize their own logic networks in hardware. FPGA also allows modifying the algorithm easily and the design time for the hardware becomes shorter by using FPGA (Ibrahimy *et al*., 2006).

In this study, a unified framework for FPGA realization of PPM is designed by means of using a standard hardware description language VHDL for two different input sizes 16-bit and 32-bit. The use of VHDL for modeling is especially appealing since it provides a formal description of the system and allows the use of specific description styles to cover the different abstraction levels (architectural, register transfer and logic level) employed in the design (Reaz *et al*., 2006; 2007b). In the computation of method, the problem is first divided into small pieces; each can be seen as a submodule in VHDL. Following the software verification of each submodule, the synthesis is then activated. It performs the translations of HDL code into an equivalent netlist of digital cells. The synthesis helps integrate the design work and provides a higher feasibility to explore a far wider range of architectural alternative (Mohd-Yasin *et al*., 2004). The method provides a systematic approach for hardware realization, facilitating the rapid prototyping of the data compression system.

METERIALS AND METHODS

For effective data compression, the compression algorithm must be able to predict future data accurately in order to build a good probabilistic model for compression (Bell *et al*., 1990). The PPM compression scheme would operate on binary data as computer-based data is represented and transmitted using binary digits. The PPM involves two steps, the generation of the adaptive model and the compression/decompression using arithmetic coding (Cleary and Witten, 1984). The generation of the adaptive model uses Markov modeling to build a probabilistic distribution of binary digits. The Markov predictor of an order j predicts the next bit based on the j preceding bits. Adaptive coding allows the model to be constructed dynamically by both encoder and decoder during the course of the transmission and has been shown to incur a smaller coding overhead than explicit transmission of the model's statistics (Cleary and Witten, 1984). A block diagram of a complete data compression system is shown in Fig 1.

Data that has been compressed need to be decompressed to return it to its original form. Therefore, a decompressor comes hand-in-hand with a compressor. Compression is dependent upon the fact that data is redundant and that its generation was based on a fixed set of rules. If those rules are known, we can accurately predict the data. The data compression can be viewed as a branch of information theory whose primary objective is to minimize the sum of data to be transmitted.

PPM Implementation: As mentioned earlier, PPM involves two steps, the generation of an adaptive model (predictor stage) and the compression or coding stage (Cleary and Witten, 1984).

Both the encoder and decoder of a PPM system, adapts the model of coding dynamically to the message statistics as the transmission proceeds (Langdon and Rissanen, 1981). Adaptive coding is effective because what is happening is counted and used as the basis of subsequent coding so that the counts only needs to be incremented in the event that a character is correctly predicted (Moffat, 1990). It should also be noted that text statistics in reality are not homogeneous and so well behaved, thus requiring a need for the adaptive coding method (Cleary and Witten, 1984). In PPM, the adaptive model is generated using a Markov predictor.

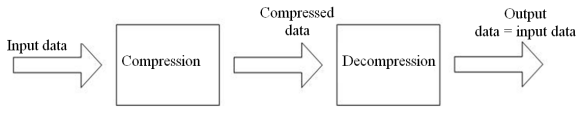

Fig. 1: A basic compression system

The bases of the PPM algorithm of order m are set of (m + 1) Markov predictors. A Markov predictor of order j predicts the next bit based upon the j immediately preceding bits. It is just a very simple Markov chain. There will be 2j possible patterns if there is j number of bits. The transition frequency is built by the predictor in proportion to the observed frequencies of a '1' or a '0' that occur, given that the predictor has seen the bit pattern associated with that state. The predictor builds the transition frequency just by recording the number of times a '1' or a '0' occurs in the (j+1) th bit following the preceding j bits. A Markov chain is built at the same time that it is used for prediction and often the chain is incomplete. Figure 2 shows an example of the 4-state Markov chain. Let there be an input sequence of '010101101' bits and the order of the Markov predictor are to be 2. The next bit is to be predicted based on the two immediately preceding bits of '01'. From observation, it can be noted that the pattern '01' occurs three times throughout the current input sequence. The frequency counts of the bit following '01' are as such: '0' follows '01' twice and '1' follows '01' once. Therefore, the predictor should predict the next bit to be '0' with a probability of 2/3 (Mudge *et al*., 1996).

A 0th order Markov predictor simply predicts the next bit based on the relative frequency in the input sequence. For simplicity, the 0th order Markov predictor is adopted for this project and it assumed that each bit encountered by the Markov predictor is novel. Figure 3 shows the flowchart for a Markov predictor.

Arithmetic encoder and decoder: Arithmetic coding is one approach to generate variable length codes and is one of the best algorithms that can be used in lossless data compression. For the case of PPM, where the modeling and coding stages of the lossless data compression have to be kept separate, arithmetic coding is a particularly well suited method to adopt.

Arithmetic coding replaces a sequence of symbols with a coding range of real numbers between 0 and 1. The range accorded to a symbol will depend on the probability of that particular symbol, the higher the probability, the higher the range, which assigns to it. The gist of arithmetic coding is the generation of a tag from that range for a sequence encoded. The tag is a floating-point value that corresponds to a binary fraction. Eventually this binary fraction will become the binary code for the sequence. In practice, the generation of the tag and the binary code are one in the same process. The arithmetic coding is divided into two phases. The first phase generates a unique identifier or tag for a given sequence of symbols and the second phase gives a unique binary code to the tag (Sayood, 1996).

The arithmetic encoder is used to generate a tag. The tag generated by the arithmetic encoder has to fall within the probability line of 0-1. The symbols are given a range based on their probability. The higher the probability, the higher is the range that is given to it. Once the ranges and the probability line are defined, the symbols are then encoded where each symbol defines where the output floating point number lands. The algorithm for arithmetic encoding is:

Low = 0
High = 1

Loop for all symbols:

Range = High – Low
High = Low + Range * Q(x)
Low = Low + Range * Q(x-1)

Where:

Q(x) = High range of symbol being encoded
Q(x- 1) = Low range of symbol being encoded

The tag can actually be any real number between 0.47265625 and 0.578125. For the purpose of this project, the tag is taken as the value of the low range, 0.47265625. The tag is useless if it cannot be deciphered. The decoder is used to recover t he original data.

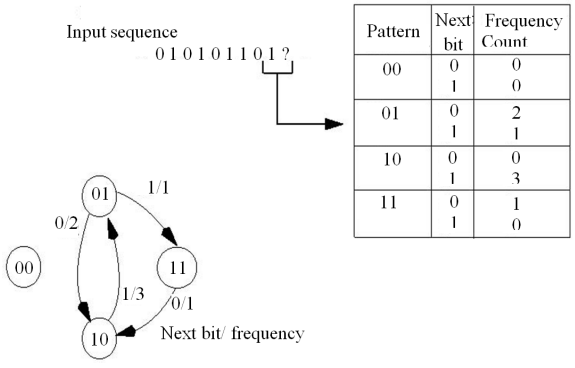

Fig. 2: The (incomplete) 4-state markov chain

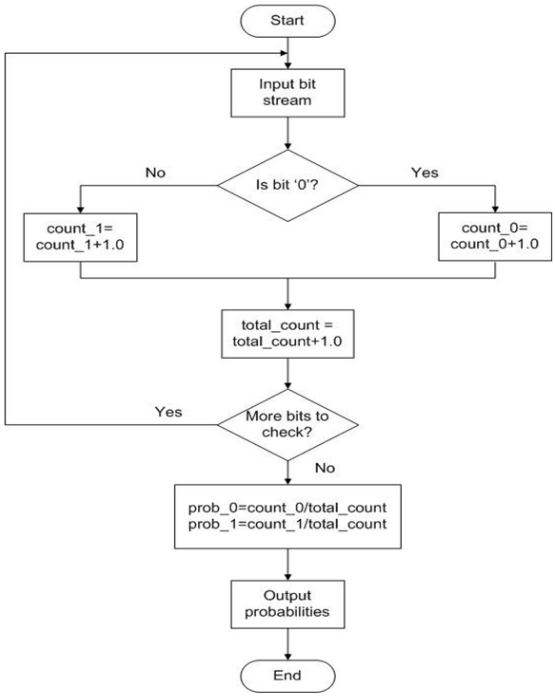

Fig. 3: Flowchart of the markov predictor

The algorithm for decoding is:

Loop for all symbols:

Range = Q(x) – Q(x – 1)
Tag = Tag-Q(x – 1)
Tag = Tag / Range

Where:
Q(x) = High range of symbol being decoded
Q(x- 1) = Low range of symbol being decoded

Software implementation: The whole design is described using IEEE-compliant VHDL language. Optimization to the VHDL code was performed to reach an even higher speed. As described earlier, the PPM module carries out probability distribution using Markov predictor followed by arithmetic encoding and decoding. A bottom-up approach is adopted here, whereby the PPM module is split into sub-modules with each sub-module being coded, verified and validated separately before being combined to produce the final design. The three sub-modules are the Markov predictor, arithmetic encoder and arithmetic decoder. However, for the design of this project, the Markov predictor would be combined with the arithmetic encoder and referred to as the Markov model. Thus, the total number of sub-modules is two. Each distinct module performs a specific function as stated in the next sub-sections.

Markov model: The Markov predictor builds the probability distribution from the input symbols seen and the arithmetic encoder generates a unique identifier or tag based on the probability distribution of symbols. Figure 4 shows the block diagram of a Markov model.

The input to the Markov model sub-module is bit_stream1_1, where a string of binary bits would be read in accordance with the clock cycle, clock1. The control signal enable1 is used to select between the Markov model sub-module and the arithmetic decoder sub-module. The probability of '1' and '0' occurring will then be calculated and the probability of both will be outputted separately in real values (floating-point values) through two output ports, prob0_0 and prob1_0. The values of these probabilities are universal to all sub-modules. The Markov predictor needs to calculate all probabilities before the arithmetic encoder can start encoding.

After the probabilities have completely been calculated and the values have been outputted, the arithmetic encoding would be carried out to produce the tag in real value. The string of bits is then encoded based on the probability according to the clock cycle, clock1. The outcome is a series of ranges in real values signifying the high and low range of each bit occurring at the input. For these ranges of highs and lows, the tag is generated and outputted at output_tag1. After the tag has successfully been generated, the sub-module would output a high bit to port trigger_decoder for the decoder to start decoding.

Arithmetic decoder: The arithmetic decoder takes the tag generated by the encoder, decodes it and output the results as a string of bits. The output string of bits for the decoder should be similar to the input string of bits for the Markov model. This sub-module consists of four input ports, one clock signal, one control signal and one output port. Figure 5 shows a block diagram of an arithmetic decoder.

The input to the arithmetic decoder is the tag in real value together with the probability of '1' and '0' (prob0_1 and prob1_1) also in real values and the trigger signal, decode. The tag is then decoded with reference to clock2. Once again, the control signal

enable2 is used to select between the two sub-modules. The output is a string of bits, output_bits similar to that of the input string of bits of the Markov model.

PPM model: The PPM main module consists of the two sub-modules; Markov model and arithmetic decoder. It does not have any input ports corresponding to clock signals or control signals but has one output port. The output is a value that needs to be checked to ensure correct operation of the module. The output is a string of bits, output_bit. Figure 6 below shows the complete top-level view of a PPM module.

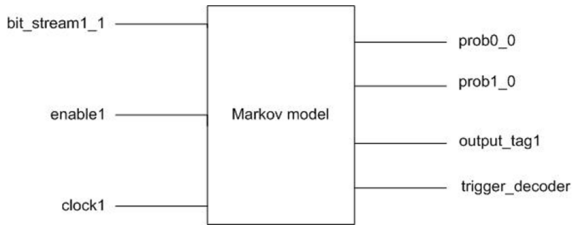

Fig. 4: Block diagram of Markov model

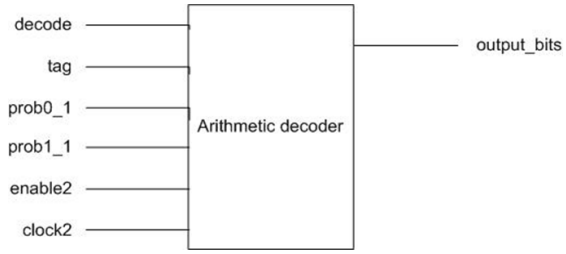

Fig. 5: Block diagram of arithmetic decoder

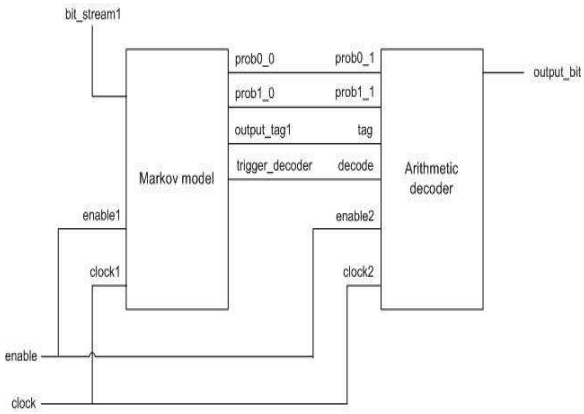

Fig. 6: Top-level view of PPM model

Hardware implementation: Hardware implementation is the unique abstract of this study, it is not sufficient by only performing software simulations. The physical hardware layout is generated using the synthesis tool Quartus II version 5.0. The compilation process is repeated with different synthesis options in order to tradeoff between area and speed properly. The project was successfully configured and downloaded to the FLEX10K EPF10K30BC356-3 FPGA, tested and validated. The FLEX10K family provides the density, speed and features to integrate entire systems, including multiple 32-bit buses into a single chip.

RESULTS AND DISCUSSION

An extensive experimental investigation was conducted in order to demonstrate the efficiency and feasibility of the proposed method. Functional simulation is performed to test the logic function of the hardware design and it is presented to verify the correctness of all the modules involved. The results of classification used different 16-bit inputs and later expanded to using a few different 32-bit inputs.

16-bit input: The test vectors have been predetermined before the simulation was carried out. Therefore the first 16-bit input to the PPM module to test its functionality is "1101011011011000" in binary or "D6D8" in hexadecimal. The simulation results are shown in Fig. 7, using the following input/output:

bit_stream1 = 16-bit input
s01 = probability of '0'
s02 = probability of '1'
s03 = output tag
output_bit = 16-bit output

At the start of the clock, the probability of '0' is 0.43750 and the probability of '1' is 0.56250. From observation, the number of occurrences of '0' in the 16-bit input stream is 7 and the number of occurrences of '1' is 9. The total number of bits is 16. Thus:

$$P('0') = 7/16 = 0.43750$$
$$P('1') = 9/16 = 0.56250$$

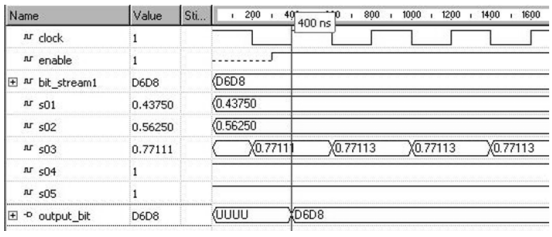

Fig. 7: Simulation results for16-bit PPM module

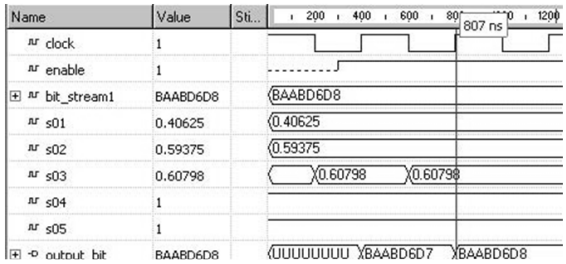

Fig. 8: Simulation results for 32-bit PPM module

Table 1: The usage of resources

Logic resources (EPF10K30BC356-3)	Logic resources:
	1164 LCs of 1728 (67.36%)
	Number of Nets: 386
	Number of Inputs: 1066
	I/O cells: 57
	Cells in logic mode: 312
	Cells in cascade mode: 45

Since both simulated and computed results are similar, it can be concluded that the Markov predictor portion of the PPM module is functioning correctly.

At time 200 ns, the tag value is 0.77111 as shown in Figure 7. The subsequent values that appear at s03 are due to the fact that the operation of the arithmetic encoding runs based on the clock event. Therefore, since the clock continues to run there will still be values at signal s03. The only way to prevent values from appearing at s03 after the first value of the tag has been outputted is to find a way to stop the clock from running after that. At time 400ns, the 16-bit output "D6D8" is available at the port output_bit. By comparing, the initial 16-bit input with the current 16-bit output, it can be observed that both of them are similar. Therefore, it can be concluded that the arithmetic encoding portion of the PPM module is functioning correctly.

32-bit input: The PPM module is expanded to accept 32-bit inputs. The first 32-bit input to the PPM module to test its functionality is "10111010101010111101011011011000" in binary or "BAABD6D8" in hexadecimal. The simulation result is shown in Fig. 8.

At the start of the clock, the probability of '0' is 0.40625 (13/32) and the probability of '1' is 0.59375 (19/32). The values are correct since '0' has a count of 13 and '1' has a count of 19. At 200ns, the tag is outputted as 0.60798.

Finally, at time 800ns, the 32-bit output appearing on the output ports is "BAABD6D8. Comparing that with the 32-bit input shows that they are similar. Therefore, the PPM module functions properly with a 32-bit input.

Synthesis results: A comparatively low critical path frequency was achieved which was 25.1 MHz. The design took a minimum resource i.e., 1164 logic cells, which is 67.36% of the EPF10K30BC356-3 device. A maximum frequency of 95.3MHz was achieved. Table 1 shows a detailed report of the usage of resources.

CONCLUSION

In this study project, the FPGA prototyping of data compression using partial matching algorithm that allows for efficient hardware implementation had been implemented. It was successfully simulated and generated acceptable results for a 16-bit input as well as for a 32-bit input. The modules were successfully compiled and simulated. The hardware implementation demonstrated complete, correct functionality and met all the initial system requirements. The performance of the hardware prototype is encouraging. The results reveal that the proposed approach is computationally simple, accurate and exhibits a good balance of flexibility, speed, size and design cycle time. Comparison and results presented validate the successful compression of data using partial matching.

REFERENCES

Akter, M., M.B.I. Reaz, F. Mohd-Yasin and F. Choong, 2008. Hardware implementations of an image compressor for mobile communications. J. Commun. Technol. Electr., 53: 899-910. DOI: 10.1134/S106422690808007X

Bell, T.C., J.G. Cleary and I.H. Witten, 1990. Text Compression. 1st Edn., Prentice Hall, Englewood Cliffs, N.J., ISBN: 0139119914, pp: 318.

Choong, F., M.B.I. Reaz and F. Mohd-Yasin, 2005. Power quality disturbance detection using artificial intelligence: A hardware approach. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Apr. 4-8, IEEE Xplore Press, Denver, USA., pp: 146a-146a. DOI: 10.1109/IPDPS.2005.348

Cleary, J. and I. Witten, 1984. Data compression using adaptive coding and partial string matching. IEEE Trans. Commun., 32: 396-402, DOI: 10.1109/TCOM.1984.1096090

Coussy, P., D.D. Gajski, M. Meredith and A. Takach, 2009. An introduction to high-level synthesis. IEEE Design Test Comput., 26: 8-17. DOI: 10.1109/MDT.2009.69

Ibrahimy, M., M.B.I. Reaz, M.A.M. Ali, T.H. Khoon and A.F. Ismail, 2006. Hardware realization of an efficient fetal QRS complex detection algorithm. WSEAS Trans. Circuits Syst., 5: 575-581.

Langdon, G. and J. Rissanen, 1981. Compression of black-white images with arithmetic coding. IEEE Trans. Commun., 29: 858-867 DOI: 10.1109/TCOM.1981.1095052

Marufuzzaman, M., M.B.I. Reaz, M.S. Rahman and M.A.M. Ali, 2010. Hardware prototyping of an intelligent current dq PI controller for FOC PMSM Drive. Proceedings of the International Conference on Electrical and Computer Engineering, Dec. 18-20, IEEE Xplore Press, Dhaka, pp: 86-88. DOI: 10.1109/ICELCE.2010.5700559

Moffat, A., 1990. Implementing the PPM data compression scheme. IEEE Trans. Commun., 38: 1917-1921. DOI: 10.1109/26.61469

Mohd-Yasin, F., A.L. Tan and M.I. Reaz, 2004. The FPGA prototyping of iris recognition for biometric identification employing neural network. Proceedings of the 16th International Conference on Microelectronics, Dec. 6-8, IEEE Xplore Press, Malaysia, pp: 458-461. DOI: 10.1109/ICM.2004.1434697

Mudge, T.N., I.C.K. Cheng and J.T. Coffey, 1996. Limits to branch prediction. The University of Michigan.

Reaz, M.B.I., M.T. Islam, M.S. Sulaiman, M.A.M. Ali and H. Sarwar *et al*., 2003. FPGA realization of multipurpose FIR filter. Proceedings of the 4th International Conferecne on Parallel and Distributed Computing, Applications and Technologies, Aug. 27-29, IEEE Xplore Press, Malaysia, pp: 912-915. DOI: 10.1109/PDCAT.2003.1236448

Reaz, M.B.I., F. Mohd-Yasin, M.S. Sulaiman, K.T. Tho and K.H. Yeow, 2004. Hardware prototyping of boolean function classification schemes for lossless data compression. Proceedings of the 2nd IEEE International Conference on Computational Cybernetics, Aug. 30-Sep.1, IEEE Xplore Press, Vienna, pp 47-51. DOI: 10.1109/ICCCYB.2004.1437664

Reaz, M.B.I., F. Mohd-Yasin, S.L. Tan, H.Y. Tan and M.I. Ibrahimy, 2005. Partial encryption of compressed images employing FPGA. Proceedings of the IEEE International Symposium on Circuits and Systems, May 23-26, IEEE Xplore Press, Malaysia, pp: 2385-2388. DOI: 10.1109/ISCAS.2005.1465105

Reaz, M.B.I., F. Choong and F. Mohd-Yasin, 2006. VHDL modeling for classification of power quality disturbance employing wavelet transform, artificial neural network and fuzzy logic. Simulation, 82: 867-881. DOI: 10.1177/0037549707077782

Reaz, M.B.I., F. Choong, M.S. Sulaiman and F. Mohd-Yasin, 2007. Prototyping of wavelet transform, artificial neural network and fuzzy logic for power quality disturbance classifier. J. Electric Power Components Syst., 35: 1-17. DOI: 10.1080/15325000600815431

Reaz, M.B.I., M.I. Ibrahimy, F. Mohd-Yasin, C.S. Wei and M. Kamada, 2007. Single core hardware module to implement encryption in TECB mode. Inform. Midem -Ljubljana, 37: 165-171.

Sayood, K., 1996. Introduction to Data Compression. 1st Edn, Morgan Kaufmann Publishers, San Francisco, ISBN-10: 1558603468, pp: 475.

Shkarin, D., 2002. PPM: One step to practicality. Proceedings Data Compression Conference, Apr. 4-4, IEEE Xplore Press, Moscow, pp: 202-211. DOI: 10.1109/DCC.2002.999958

Ziv, J. and A. Lempel, 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory, 23: 337.343, DOI: 10.1109/TIT.1977.1055714

Ziv, J. and A. Lempel, 1978. Compression of individual sequences via variable-rate coding. IEEE Trans. Inform. Theory, 24: 530-536. DOI: 10.1109/TIT.1978.1055934