World Applied Sciences Journal 22 (1): 133-139, 2013
ISSN 1818-4952
© IDOSI Publications, 2013
DOI: 10.5829/idosi.wasj.2013.22.01.201

Hardware Approach of Lempel-Ziv-Welch Algorithm for Binary Data Compression

Md. Mamun, Md. Arif Sobhan, Ahmad Ashrif A. Bakar and Hafizah Hussain

Department of Electrical Electronics and Systems Engineering,
Universiti Kebangsaan Malaysia, 43600, Bangi, Malaysia

Abstract: In a distributed environment, large data files remain a major bottleneck. Compression is an important component of the solutions available for creating file sizes of manageable and transmittable dimensions. When high-speed media or channels are used, high-speed data compression is desired. Software implementations are often not fast enough. In this paper, we present the very high speed hardware description language (VHDL) modeling environment of Lempel-Ziv-Welch (LZW) algorithm for binary data compression to ease the description, verification, simulation and hardware realization. The VHDL model defines a main block, which describe the LZW algorithm for binary data compression through a behavioral and structural description. The LZW algorithm for binary data compression comprises of two modules compressor and decompressor. The input of compressor is 1-bit bit stream read in according to the clock cycle. The output is an 8-bit integer stream fed into the decompressor, which is an index that represents the memory location of the bit string stored in the dictionary. The output of decompressor is 1-bit bit stream. Once detecting the particular approaches for input, output, main block and different modules, the VHDL descriptions are run through a VHDL simulator, followed by the timing analysis for the validation, functionality and performance of the designated model that supports the effectiveness of the model for the application.

Key words: Binary Data Compression · LZW · VHDL · Simulation

INTRODUCTION

Despite the fact that computer memory costs have decreased dramatically over the past few years, data storage still remains, and will probably always remain, an important cost factor for many large scale database applications [1-3]. Compressing data in a database system is attractive for two reasons: data storage reduction and performance improvement. Storage reduction is a direct and obvious benefit, while performance improves because smaller amounts of physical data need to be moved for any particular operation on the database [4-6].

One of the most important developments in the study of data compression is the modern paradigm first presented by Rissanen and Langdon [7]. This paradigm divides the process of compression into two separate components: modeling and coding. Modeling is the process of constructing representation. Coding entails mapping the modeler's representation of the source into a compressed representation. The process of converting a text, such as a file on disk, a string in memory, or a stream of characters, into a compact representation is called encoding or compression. Decoding or decompression restores the original text from its compressed representation [8-9].

Huffman coding is constrained to represent every event using an integral number of bits [10-11]. Updating a Huffman tree is much more time consuming. Like Huffman, Shannon-Fano coding requires to recognize the statistical properties of the source file i.e. the probability of occurrence of a code must be found first before the compression start [12-13]. But in most cases, it is not possible to determine the statistical properties of the file due to the randomness of the occurrence.

Dictionary compression (also referred to as Ziv-Lempel compression or textual substitution) removes data redundancy by replacing repeated input substrings by references (also called indices or pointers) to earlier copies of the identical substring [14-17]. A dictionary of characters, words or phrases that are expected to occur

Corresponding Author: Md. Arif Sobhan, Department of Electrical Electronics and Systems Engineering, Universiti Kebangsaan Malaysia, 43600, Bangi, Malaysia.

frequently is maintained and a recurring substring is encoded by the index of its corresponding dictionary entry. Compression is achieved by choosing indices so that on average they require less space than the phrase they encode.

LZW compression implements a variation of LZ78 due to Welch and initializes the dictionary with the input character set [18]. LZW compression is a lossless, adaptive and dynamic dictionary compression technique [19-20]. LZW compress text, executable code, and similar data files to about one-half their original size. LZW also performs well when presented with extremely redundant data files, such as tabulated numbers, computer source code, and acquired signals. Compression ratios of 5:1 are common for these cases.

In this paper we present a model of LZW algorithm for binary data using VHDL. The goal of this work is to ease hardware realization to achieve high-speed data compression and to evaluate the feasibility of using VHDL for rapid design and prototyping of the same.

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 [21-23]. 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 hardware description language 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 [24-27]. The method provides a systematic approach for hardware realization, facilitating the rapid prototyping of the energy meter.

MATERIALS AND METHODS

This research is to develop a model of LZW algorithm for binary data compression using VHDL. The model is capable of compressing 1-bit bit stream of binary data. The application of the model can be used in various data compression devices thus saving memory space and reduce the transmission time. A structural view of the model is shown in Fig. 1.

The input of compressor is a bit streams, where binary data is read in one by one according to the clock cycle. The input bit stream is compressed, or encoded to another form. The output is an index that represents the memory location of the bit string stored in the dictionary called a codeword. The output of compression is an integer stream, which consists of a predefined number of bits. The number of bits used in the compression determines the number of memory locations allocated in the dictionary. The selection of number of bits is important as the memory locations are filled up before all the input bits are read if the number of memory locations allocated is not enough. This make the dictionary "congested" and as a result the compression cease. The number of bits used is directly proportional to the file size to be compressed. Unlike design in software compressor, the software is able to check the file size and allocate more bits for bigger file size. Since the compressor is hardware, where the compressor itself unable to check the file size, the number of bits used is predefined. In this research, 8 bits are used to design an 8-bit compressor thus the memory locations allocated is 28=256 memory locations.

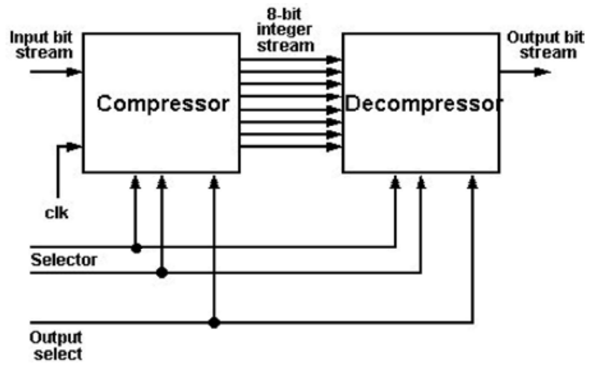

Fig. 1: Structural view of VHDL model

Since the decompressor decompress the output of compression, the compression output is used as the input for decompression. The decompressor adds a new string to the dictionary each time it reads a new code. It translates each incoming code into a string and sends it to the output. The input to the decompressor is an integer stream consists of 8 bits. The output of the decompression is a bit streams. The desired output of decompression is binary data, which is same as the original binary data.

Three control signals, clock, selector and output select control the operation. Clock signal is used to control the timing. Selectors are used to select the functions, reset, compression, or decompression. Table 1 shows the function and its corresponding control signal.

Table 1: Selector and its corresponding function

X	Y	Function
0	0	Reset
0	1	Compression
1	0	Decompression
1	1	Unused

The last control signal is output select. When output select is “1”, the output is enabled and the compressor output the current value of the output, whether it is in compression or decompression. This signal is necessary because when the compressor searches the dictionary to find whether the input string is in the dictionary, the searched result is either found or not found. If the string is not found, the compressor input next byte, concatenates it with the previous string, and searches the dictionary again. Hence, the compressor does not have an output after every new cycle. The output is only enabled when a string is not found in the dictionary.

VHDL language is able to design in hierarchies. Reusing components, error management and verification allows describing complex circuitry efficiently. In VHDL model, three packages std_logic_1164, std_logic_arith and std_logic_unsigned are defined from the IEEE library. These packages include definition of standard logic data types, arithmetic operations involving standard logic data types, integer data types, integer to standard logic vector conversion and vice versa. The standard logic is the type declaration for bit stream and standard logic vector is the type declaration for integer stream.

The VHDL design entity for the LZW algorithm for binary data compression is built by constructing a number of entities with certain behavior associated with them. The modeling styles of entities that are being used: structural, dataflow, and behavioral. The model is basically consists of 4 main parts, i.e. entity declaration, architecture declaration, signal declaration, and clock and control setting.

The entity declaration specifies the name of the entity being modeled. All the interface ports, which include input and output, are declared in the entity declaration. The entity name of the model is lzw. There are 7 interface ports that are declared which contributes a total of 22 input and output. Bit_stream is the input of 1-bit bit stream for compression. Integer_stream consists of 8-bit for compression output. Integer_stream_de consists of 8-bit for decompression input and Bit_stream_de is the output of 1-bit bit stream. The following VHDL code is the entity lzw declaration.

```
entity lzw is
        port (
                Bit_stream: in STD_LOGIC;
                Integer_stream: out STD_LOGIC_VECTOR (7 downto 0);
                Integer_stream_de: in STD_LOGIC_VECTOR (7 downto 0);
                Bit_stream_de: out STD_LOGIC;
                output_sel: out STD_LOGIC;
                clock: in STD_LOGIC;
                selector: in STD_LOGIC_VECTOR (1 downto 0)
        );
end lzw:
```

Architecture defines a body for a component entity. An architecture body specifies the behavior between inputs and outputs. The architecture name is not the same as the component name. The architecture name of the model is lzw_archi. Architecture lzw_archi is tied to entity lzw.

All the signals are declared and also the type of variable is defined. A partial signal declaration VHDL code is given below.

```
variable element  : integer := 0;
variable counter  : integer := 1;
variable ML_max   : integer := 2;
variable ele_num  : integer := 0;
variable found    : integer := 1;
variable ML       : array256;
variable ocode    : integer;
variable ncode    : integer;
variable ML_de    : array256slv;
variable str      : std_logic_vector(7 downto 0);
variable bitcnt   : integer;
variable endpro   : integer := 0;
```

The program starts its execution with process “clock” high. The control signal “selector” has three functions run at any one time. These functions are reset, compression and decompression. The VHDL code formation of clock and control setting is given below.

```
begin
     process(clock)
          .
          .
     begin
          if (clock'event and clock='1')then
                    .
                    .
                 if(selector="00")then -- reset case
                         .
                         .
                 elsif (selector="01")then – compression
                         .
                         .
                 elsif (selector="10")then – decompression

                 end if;
          end if;
```

RESULTS AND DISCUSSIONS

A test bench is written to perform simulation by adding the stimulus and defining the clock cycle to verify the correctness of the model.

A part of the VHDL code of stimulus is given below. At the initialization of the simulation, the selector is set to “00” to reset the operation. When the time run until 210ns, the selector selects either compression or decompression depends on which simulation is desired.

```
selector <= "00" after 0 ns,"01" after 210 ns;

clock <= not(clock) after 100 ns;

Bit_stream <= '1' after 210 ns,'0' after 410 ns,'0' after 610 ns,'1' after 810 ns
```

The simulation is performed using the string of 50 binary bits. The simulation waveform of compression block and decompression block is shown in Fig. 2 and Fig. 3 respectively and their corresponding variables and functions are tabulated in Table 2 and Table 3 respectively.

In Fig. 2, the clock cycle is 200 ns per cycle. The selector is “00” from 0ns to 210ns. Selector “00” is the reset case. After 210 ns, the selector is set to “01” to switch to compression mode. Hence, the first input bit is read at 210 ns, and after every new cycle, the next bit is read. The output is enabled when output_sel is high, or equal to “1”. For every high state of output_sel, the corresponding value of Integer_stream is the output of compression.

Table 2: Variable and its function in compression

Name	Function
Bit_stream	Input bit stream
Integer_stream	Output integer stream
Clock	Clock signal
Output_sel	Output enable
Selector	Function selector

Table 3: Variable and its function in decompression

Name	Function
Bit_stream_de	Output bit stream
Integer_stream_de	Input integer stream
Clock	Clock signal
Output_sel	Output enable
Selector	Function selector

Input:
1,0,0,1,0,1,1,0,1,1,0,1,1,0,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,1,1,0,
1,0,1,1,1,0,1,1,1,0,1,0,0,1,0,1,1

Output: 1,0,0,2,1,5,7,3,4,3,2,10,12,7,5,4,6,17,16,13,1

In Fig. 3, the clock cycle is 200 ns per cycle. The selector is “00” from 0ns to 110ns. After 110 ns, the selector is set to “10” to switch to decompression mode. Hence, the first input integer is read at 110 ns and after

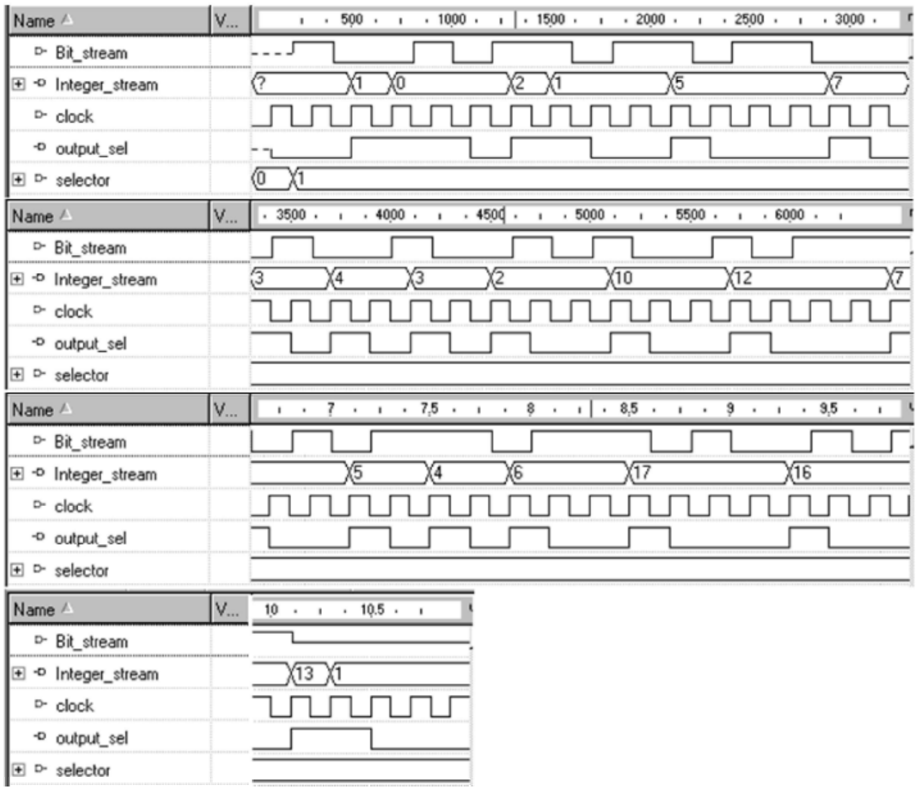

Fig. 2: The Simulation waveform of compression block.

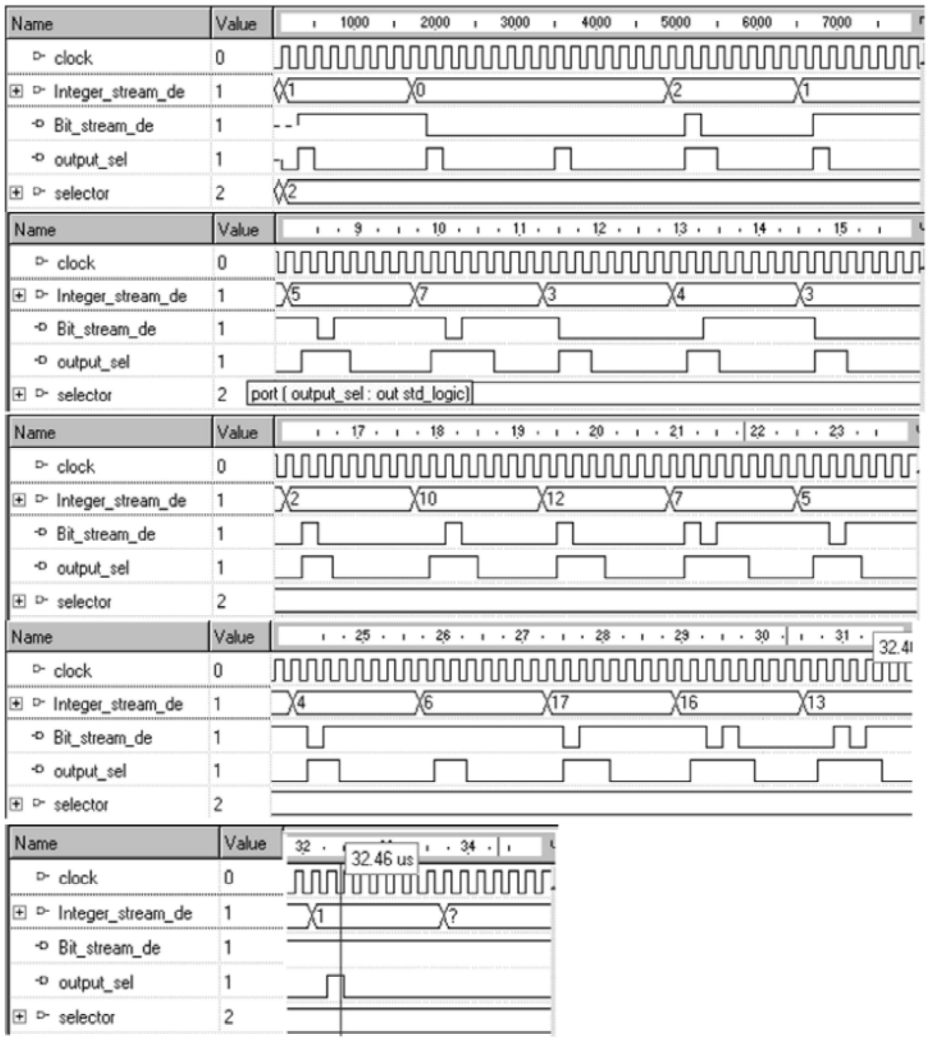

Fig. 3: Simulation waveform of decompression block.

every 8 clock cycles, or 1600 ns, the next integer is read.. The gap between subsequent inputs is 8 clock cycles due to 8-bit integer. The decompressor read one bit in one clock cycle. Like compression, the output is enabled when output_sel is high, or equal to "1". For every high state of output_sel, the corresponding value of Bit_stream_de is the output of decompression.

Input:1,0,0,2,1,5,7,3,4,3,2,10,12,7,5,4,6,17,16,13,1

Output:
1,0,0,1,0,1,1,0,1,1,0,1,1,0,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,1,1,0,
1,0,1,1,1,0,1,1,1,0,1,0,0,1,0,1,1

The simulation results of input and output of each block confirms the correctness of the code. The model is also compared with a C model of LZW algorithm where the same input is being fed and the output is synonymous with the result obtained from VHDL model.

CONCLUSION

In this paper, we present the very high speed hardware description language (VHDL) modeling environment of Lempel-Ziv-Welch (LZW) algorithm for binary data compression to ease the description, verification, simulation and hardware realization. By simulating with binary data and comparing with C model of LZW algorithm the proposed approach of VHDL modeling of LZW algorithm for binary data compression is successfully designed, implemented and tested. Currently, we are conducting the further research that considers further reductions in the hardware complexity in terms of synthesis and ease hardware realization.

REFERENCES

1. Reaz, M.B.I., Choong F. and Mohd-Yasin F. 2006. VHDL modeling for classification of power quality disturbance employing wavelet transform, artificial neural network and fuzzy logic. Simulation, 82(12): 867-881.
2. Mohd-Yasin, F., A.L. Tan and M.I. Reaz, 2004. The FPGA prototyping of Iris recognition for biometric identification employing neural network. In the Proceedings of the International Conference on Microelectronics, ICM, pp: 458-461.
3. Marufuzzaman Mohd., M.B.I. Reaz, M.S. Rahman and M.A. Mohd. Ali, 2010. Hardware prototyping of an intelligent current dq PI controller for FOC PMSM drive. In the Proceedings of the 6th International Conference on Electrical and Computer Engineering (ICECE 2010), pp: 86-88.
4. Arifin, M.A.B.T., M. Mamun, M.A.S. Bhuiyan and H. Husain, 2012. Design of A Low Power and Wide Band True Single-Phase Clock Frequency Divider. Australian Journal of Basic and Applied Sciences, 6(7): 73-79.
5. Kader, W.M., H. Rashid, M. Mamun and M.A.S. Bhuiyan, 2012. Advancement of CMOS Schmitt Trigger Circuits. Modern Applied Science, 6(12): 51-58.
6. Rosli, K.A., M. Mamun, M.A.S. Bhuiyan and H. Husain, 2012. A Low Loss Wide Swing Cascode Current Mirror in 0.18-ìm CMOS Technology. Journal of Applied Sciences Research, 8(8): 4096-4102.
7. Rissanen, J.J. and G.G. Langdon, 1981. Universal Modeling and Coding. IEEE Trans. Inf. Theory, 27(1): 12-23.
8. Bhuiyan, M.A.S., M. Reaz and M. Ali, 2012. A Review On On-Chip Antenna For 2.4 Ghz Ism Band Rfid Tag. In the Proceedings of the 1st International Conference on Engineering and Built Environment (ICEBE 2012), pp: 1-6.
9. 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. Informacije MIDEM, 37(3): 165-171.
10. Huffman, D.A., 1952. A Method for the Construction of Minimum-Redundancy Codes. In the Proceedings of the Inst. Radio Eng., pp: 1098-1101.
11. Jing, Y., 2011. The Combinational Application of LZSS and LZW Algorithms for Compression Based on Huffman. In the Proceedings of the International Conference on Electronics and Optoelectronics (ICEOE-2011), pp: 397-399.
12. Shannon, C.E., 1948. A Mathematical Theory of Communication. Bell System Technical Journal, 27: 379-423, 623-656.
13. Fano, R.M., 1961. Transmission of Information. Wiley, New York.
14. Rodeh, M., V.R. Pratt and S. Even, 1981. Linear Algorithm for Data Compression via String Matching. J. ACM, 28(1): 16-24.
15. Storer, J.A. and T.G. Szymanski, 1982. Data Compression via Textual Substitution. J. ACM, 29(4): 928-951.
16. Ziv, J. and A. Lempel, 1977. A Universal Algorithm for Sequential Data Compression. IEEE Trans. Inf. Theory, 23(3): 337-343.
17. Ziv, J. and A. Lempel, 1978. Compression of Individual Sequences via Variable-Rate Coding. IEEE Trans. Inf. Theory, 24(5): 530-536.
18. Welch, T.A., 1984. A Technique for High-Performance Data Compression. IEEE Computer, 17(6): 8-19.
19. Bin, L., N. Guiqiang, L. Jianxin and Z. Xue, 2011. BWT-based Data Preprocessing for LZW. . In the Proceedings of the International Conference on Multimedia and Signal Processing (CMSP- 2011), pp: 37-40.
20. Tao, T. and A. Mukherjee, 2005. Pattern Matching in LZW Compressed Files. IEEE Transactions on Computers, 54(8): 929-938.
21. Romli, N.B., M. Mamun, M.A.S. Bhuiyan and H. Husain, 2012. Design of a Low Power Dissipation and Low Input Voltage Range Level Shifter in Cedec 0.18-μm CMOS Process, World Applied Sciences Journal, 19(8): 1140-1148.
22. Yasin, F.M., A.L. Tan and M.I. Reaz, 2004. The FPGA Prototyping of Iris Recognition for Biometric Identification Employing Neural Network. . In the Proceedings of the 16th IEEE International Conference on Microelectronics, pp: 458-461.
23. Chen, S., B. Mulgrew and P.M. Grant, 1993. A Clustering Technique for Digital Communications Channel Equalization using Radial Basis Function Networks. IEEE Trans. Neural Networks, 4(4): 570-590.
24. Amin, M.S., M.B.I. Reaz and J. Jalil, 2013. Design of a novel adder-less Barker matched filter for RFID. Int. J. Circ. Theor. Appl.. doi: 10.1002/cta.1895.
25. Reaz, M.B.I., F. Choong, M.S. Sulaiman and F.M. Yasin, 2007. Prototyping of Wavelet Transform, Artificial Neural Network and Fuzzy Logic for Power Quality Disturbance Classifier. Journal of Electric Power Components and Systems, 35(1): 1-17.

26. Akter, M., M.B.I. Reaz, F.M. Yasin and F. Choong, 2008. Hardware Implementations of Image Compressor for Mobile Communications. Journal of Communications Technology and Electronics, 53(8): 899-910.

27. Reaz, M.B.I., M.T. Islam, M.S. Sulaiman, M.A.M. Ali, H. Sarwar and S. Rafique, 2003. FPGA Realization Of Multipurpose FIR Filter, . In the Proceedings of the Parallel and Distributed Computing, Applications and Technologies (PDCAT-2003), pp: 912-915.