Implementation of Parallel Multiplications on FPGA

Kai Zhi Woo#*, Poh Kit Chong*, *Lee Kong Chian
Faculty of Engineering and Science
Universiti Tunku Abdul Rahman
#Motorola Solutions Penang
stevenwkz@yahoo.com, pkchong@utar.edu.my

***Abstract*—Multiplications are often involved in Digital Signal Processing such as for digital filters and FFT (Fast Fourier Transform). It requires high speed multipliers and logic components (such as adders, subtractors, and shifters). Processing speed is always critical for Digital Signal Processing, and there are many attempts to reduce the processing latency that may cause performance issues on the end product. There has been a number of parallel multiplication approaches proposed to speed up computation. This paper aims to design and implement several parallel multiplication approaches using Field Programmable Gate Array (FPGA). These approaches make use of the resources in FPGA to achieve fast multiplication. The parallel methods implemented and compared in this paper include partitioning of multiplicands for parallel multiplication, hybrid Look-up tables (LUT) parallel multiplication, and Wallace Tree Multiplication algorithm. Comparison is made based on the number of processing cycles and also the amount of resources used in the FPGA through simulation. The proposed designs utilized lesser processing cycles (5 cycles) for a single process by using the FPGA resources effectively.**

Keywords—Digital Signal Processing(DSP); hybrid; parallel; multiplication; multipllier; FPGA; Look-up Table; LUT; Wallace Tree.

I. INTRODUCTION

Multiplications often play an important role in computing systems for security, financial computing, and Digital Signal Processing (DSP) especially in digital filtering (FIR and IIR filters), FFT (Fast Fourier Transform), and DFT (Discrete Fourier Transform). The multiplication operation can be done using embedded multipliers in a processor (e.g.: multipliers in DSP48 slices of Xilinx Virtex series FPGA.), logic resources (e.g.: Configurable Logic Blocks of Xilinx Virtex series FPGA [1].), and also memory based implementation (e.g., Look-up Table (LUT)). An analysis done on the execution time of some applications show that decimal multiplication could reach over 27 percent of the total time of a DSP application. [2].

Field Programmable Gate Array (FPGA) is a hardware technology which offers the capability to develop the circuit architecture for computational, memory and power requirements of the application in a similar way to system-on-chip (SoC) solutions [3]. FPGAs are a collection of components which allow users to create a DSP system easily as it involves multiple hardware and software components and also interconnection fabrics. The availability of multiple, distributed logic resources and registers enable the opportunity for parallelism and pipelining, which results in a highly efficient circuit architecture in a FPGA implementation.

Several attempts on parallel multiplication, such as those in [4], [5], [6] have been proposed for fast multiplication. The parallel floating point multiplication approach presented in [4] makes use of the modular and parallel nature of FPGA architecture. Multiplication is done in three stages which consist of partial product generation, accumulation, and final addition. All operation in each stage is done in parallel as digits are framed into multiple windows and summed up at the final addition stage. Furthermore, the proposed method in [5] is the parallel multiplication of decimal integer words. Partial products are generated in parallel and reduction of partial products is also been done for efficient implementation of decimal parallel multiplication.

A LUT based approach has been used for parallel multiple constant multiplications (MCM) on FPGA as describe in [6]. There are a few proposed methods for optimization of LUT multiplications, one example of which is detailed in [7]. On the other hand, a Wallace Tree multiplier is a type of efficient parallel multiplier. The Wallace Tree multiplier is one of the methods to enhance the speed of multiplication operation. It is also considered to be a high speed multiplier. We have implemented a Wallace Tree multiplier, and it will be discussed further in this paper.

In this paper, three parallel multiplication approaches implemented on FPGA are introduced based on previous works [4], [5], [6], [7], [8] as a starting point. The FPGA used for the implementation is Xilinx Kintex-7 FPGA KC705 (xc7k325tffg900-2).

978-1-4673-6716-5/15/$31.00 ©2015 IEEE

These three parallel approaches are: partitioning of multiplicands/multipliers for parallel multiplication, a hybrid LUT multiplication approach, and the conventional Wallace Tree Multiplication algorithm. The remainder of the paper is organized as follows. In Section II, the proposed design methodology of partitioning of multiplicands/multipliers for parallel multiplication is detailed. The proposed hybrid LUT based parallel multiplication is detailed in Section III and the proposed design methodology of the conventional Wallace Tree Multiplication algorithm is detailed. In Section IV, the component and time complexity of each proposed method is discussed and evaluated. Lastly, we conclude in Section V.

II. Design 1: Parallel Multiplication By Partitioning of Multipliers

Referring to [4], the binary multiplication algorithm contains three stages. We have modified these stages with the following proposed methods to perform parallel multiplication algorithm:

- Partial Products Generation (Multipliers are partitioned and partial product is shifted)
- Accumulation (Accumulated output is shifted)
- Final Addition

The bit width of the binary multipliers plays an important role in binary multiplication. As the bit width increases, time taken to complete a multiplication will also increase. It is a linear relationship between the binary bit width and the multiplication latency. In order to reduce the latency of the multiplication process, the binary multipliers is partitioned into several subsections before the multiplication operation starts. The purpose of the partitioning is to sub divide the multipliers with large bit width into several subsections with lesser bit width. After the sub division process, the multiplication between the multiplicand and sub divided multipliers can be performed in parallel.

Next, there is a shifting process on the output of the even subsections before the accumulation stage. This is to shift the output of the even subsection into the correct position before the accumulation process. It is similar with the shifting process of the partial product of a canonical multiplication operation. In the accumulation stage, all partial products which also include the shifted partial products are summed up. Shifting processes after the summation are also required in order for addition process in final addition stage to take place correctly. Finally the accumulation outputs are added up at the final addition stage to produce the final output of the binary multiplication. All multiplication and addition operations at each stage are done in parallel which exploit the high level of parallelism available on FPGA DSP algorithms.

The proposed design utilizes multipliers and adders of the logic resources in the FPGA. Several LUT slices were used to store the multiplicands for fast multiplication. Figure 1 shows the multiplication of two binary numbers with the word size of 32 bits. Before the multiplication starts, the multiplier (B) is partitioned into 8 subsections with 4 bit word size in each subsection as shown in Figure 2. Figure 3 illustrates the overview of proposed parallel multiplication method.

A = 10 011011 011011 011011 011011 011011
X
B = 11 110110 110110 110110 110110 110110

Figure 1
An example of a 32 x 32 bits binary multiplication

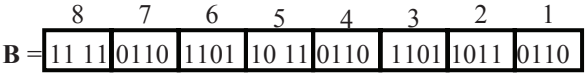

Figure 2
Partitioning of multiplicands into subsections

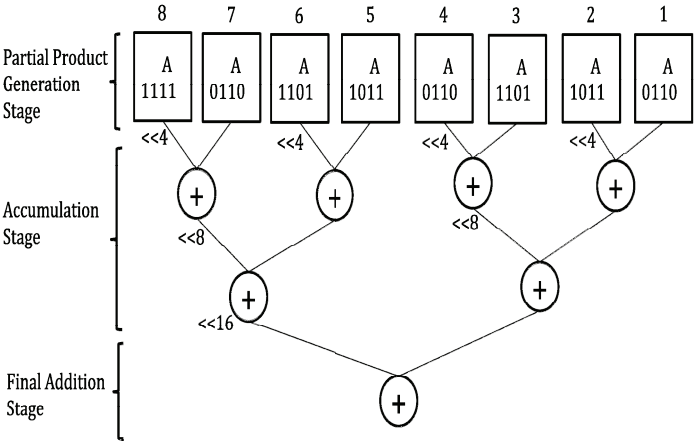

Figure 3
Block diagram for overall operation of Design 1

In Figure 3, the required number of shifting for the partial product of the partial product generation stage is depending on the bit width of multipliers. As stated earlier, shifting is done for even sections only. Subsequently, the required number of bits to be shifted will double up.

III. Design 2: Hybrid Look-Up-Tables (LUT) based Parallel Multiplication

Look-Up-Table (LUT) is an array which stores possible pre-computed data in memory elements. Each pre-computed data is stored in a specific address of the memory element. The memory element will be used to look-up for the output based on the input data. When the input data matches the address of the stored data, the stored data will be used as the output of LUT. Multiplication based on LUT method is where some or all multiplicands, multipliers and pre-computed products are stored in the memory element. The LUT will then look up for the product based on the combination of multiplicand and multipliers. It is suitable for varying word size multiplication, and offers fast and parallel multiplication.

There is work done on optimization of LUT based multiplication as described in [7]. The approach in [7] is to reduce the usage of memory area for memory based multiplication. It is done by downsizing the number of pre-computed products in the memory element and also using logic operations to compute the product for the LUT output. In this paper, the implementation of LUT based multiplication by employing the LUT resources of FPGA is illustrated. Figure 4 embodies the concept of the hybrid LUT based parallel multiplication. This hybrid approach consists of partitioning, look-up method, and shifting and addition.

Firstly, the input data which is the multiplicand and multiplier are partitioned into several subsections. The purpose of partitioning into subsections is to reduce the size of LUT and also to sub divide the multiplication process for binaries with larger bit width into several processes with lesser bit width.

Secondly, the partitioned multiplier values will be used to look–up for the particular product LUT. Each product LUT stored all possible products and the value of multiplier is the address of each product LUT. Once the product LUT is determined, the next look-up process will search for the product values based on the partitioned multiplicand value. Lastly, the shift and add process is used to fine tune all obtained products of respective sub sections to produce the appropriate output products.

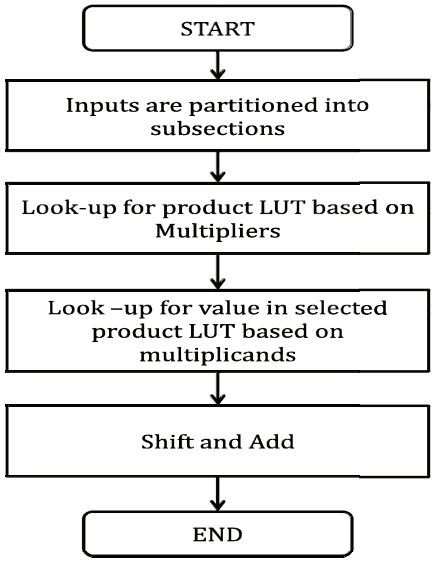

Figure 4
Operation flow for Design 2.

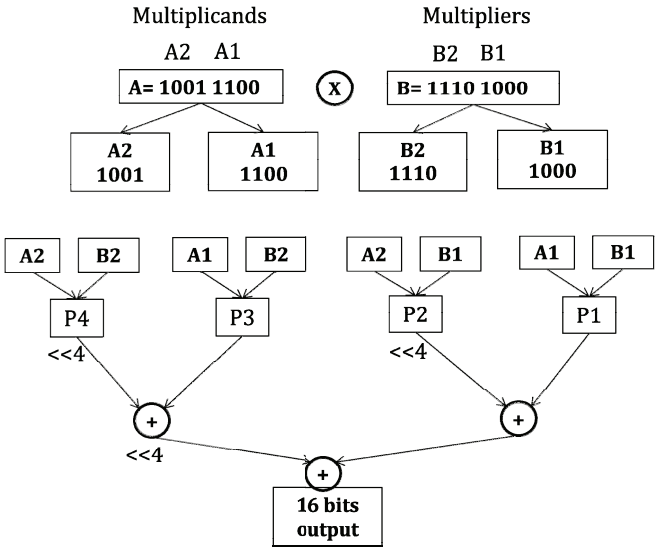

Figure 5
Block diagram for overall operation of Design 2

Figure 5 illustrated an example of an 8x8 bits multiplication. The multiplication is done by several 4x4 bits multiplication processes. A1, A2 are the partitioned multiplicands and B1 and B2 are the partitioned multipliers. There are four 4x4 bits multiplication processes done in parallel (A1 and B2, A1 and B2, A2 and B1 and A1 and B1). P1, P2, P3 and P4 are the products of each multiplication retrieve from the LUT as shown in Figure 6. Shift and add process is done on each product to produce a 16 bits output product. This example is implemented and simulated using Vivado Design Tool Suite. The simulated waveform is shown in section V.

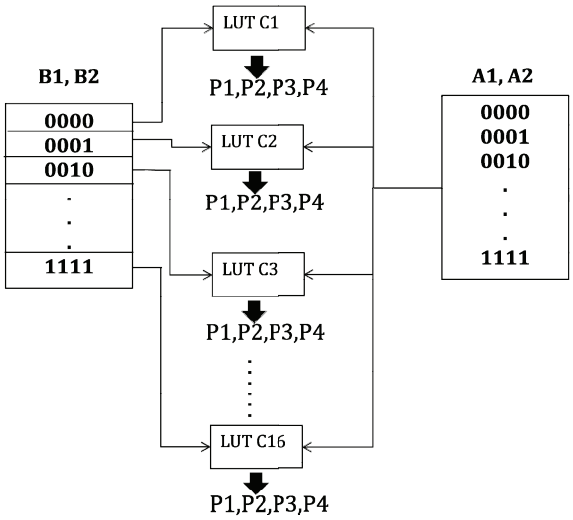

Figure 6
Block Diagram for LUT operation of Design 2

The look-up process of the hybrid LUT based parallel multiplication is illustrated in Figure 6. The look-up process starts when the look-up table receives the partitioned input bits which are A1, A2, B1, and B2. LUT C1 till LUT C16 is the product LUTs which stores all possible products with 8-bit bit widths. The details of the operation are explained as previously. The products will be fetched from the selected product LUT (From LUT C1 till LUTC 16) to the shift and add process.

This hybrid approach employs LUT slices and register/latches such as FDRE (D Flip-Flop with Clock Enable and Synchronous Reset) on the FPGA. The entire look-up process is done in parallel as it will retrieve the values stored in the ROM of the FPGA accordingly based on the multiplicand and multiplier available at the LUT input. It is also suitable for multiple constant multiplications (MCM) as illustrated in [6].

IV. Design 3: Conventional Wallace Tree Multiplication

The architecture of Wallace tree multiplier consists of three phases as shown in Figure 7.

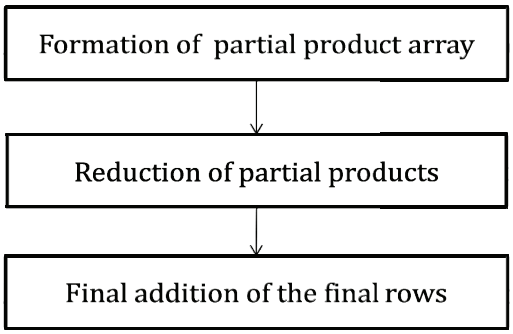

Figure 7
Operation flow for Design 3

Firstly, all partial products are generated in parallel using logic gates or embedded multipliers. The array is formed by grouping the binary bits at each column of the partial product. Next is the reduction of partial products by using full adders and half adders. For an 8x8 bits binary multiplication, there are five reduction stages. All reduction processes in each reduction stage is done in parallel using the logic resources of the FPGA. During the reduction process, three binary bits of the array will be reduced to two bits using a full adder whereas the half adder is used for two bits array. The rest of the binary bits of the array will be fetched to the subsequent stage. The reduction process is to reduce all partial products till the remaining two rows of partial products. Two resulting rows will be summed up to produce the final output.

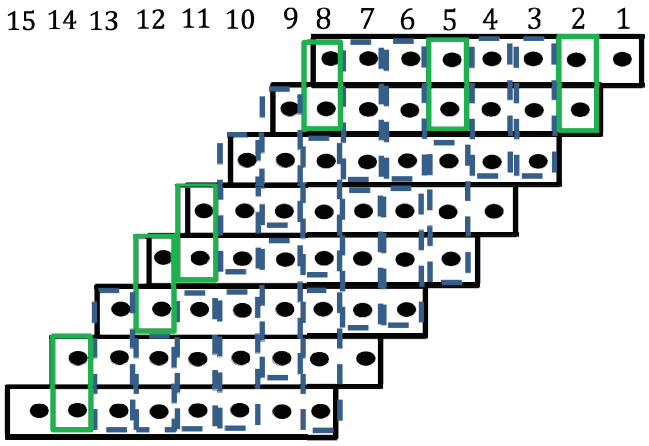

Figure 8
Formation of partial product array of Design 3

Figure 8 shows an example of the formation of partial product array for 8x8 bits binary multiplications. It is also shows the first stage partial product reduction of the Wallace tree multiplication. The partial products are divided into 15 columns. Each column had their own reduction processes using full adders (in dashed boxes) and half adders (in solid boxes). The sum and carry of each adder will be passed on to the subsequent stages together with the remaining bits which do not go through the reduction process.

V. Results And Discussion

The three parallel binary multiplication approaches are synthesized and simulated using Xilinx Vivado Design Tool Suite 2013. As mentioned earlier, the device selected for this implementation is Xilinx Kintex-7 FPGA KC705 (xc7k325tffg900-2).

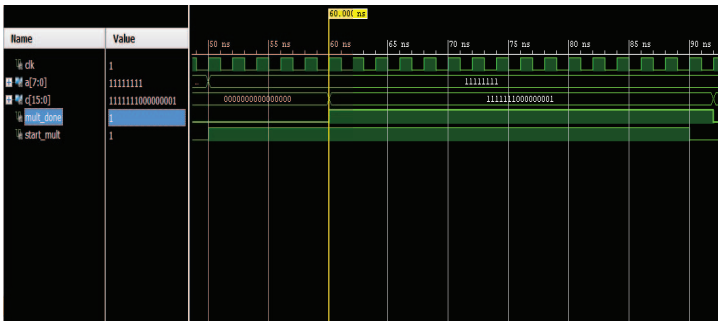

Figure 9
Simulated waveform of Design 1

The 8x8 bits parallel multiplication approach by partitioning of multipliers is simulated and the simulated waveform is shown in Figure 9. This approach utilized 146 logic cells, 28 I/O ports and 206 nets for the connection of the components. The multiplication involved two binary numbers with the same value which is 0b11111111 as the multiplicand and multiplier. The output value is 0b1111111000000001 which is

exactly the same as the calculated product. The multiplier is stored as a constant in the LUT of FPGA. It shows that the proposed multiplication method required 5 clock cycles which is equal to a duration of 10ns at the clock speed of 500MHz (2 ns per cycle) to complete a single multiplication process.

Consequently, the hybrid LUT based parallel multiplication approach (Design 2) utilized 645 logic cells, 35 I/O ports and 708 nets for component connections. This implementation consists of LUT slices with 4 input ports and D-Flip-Flops in the FPGA. Pre-computation work need to be done before the implementation of this LUT multiplication as all possible multiplicands, multipliers and products need to be computed. All pre-computed values are stored in the LUT slices of the FPGA.

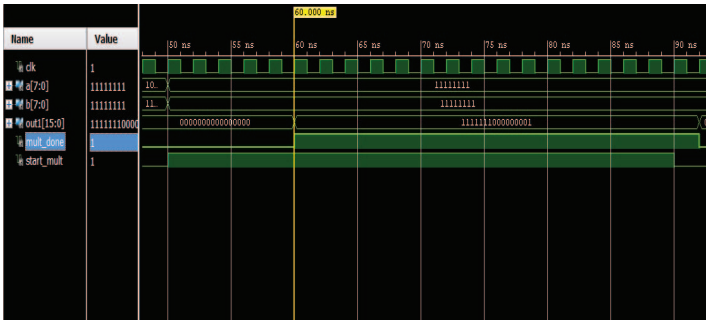

Figure 10
Simulated waveform of Design 2

Figure 10 is the simulated waveform of the hybrid LUT based multiplication. The multiplication involved two binary numbers with the same value which is 0b11111111 as the multiplicand and multiplier. The output value is 0b1111111000000001 which is the same as the calculated product. This approach is required 5 clock cycles to complete a multiplication process which is duration of 10ns at the clock speed of 500MHz.

On the contrary, the 8x8 bits binary multiplication base on the conventional Wallace Tree multiplier (Design 3) utilized a total of 211 logic cells, 34 I/O ports and 383 nets for component connections. It utilized components a large number of components such as full adders, half adders and D flip-flop.

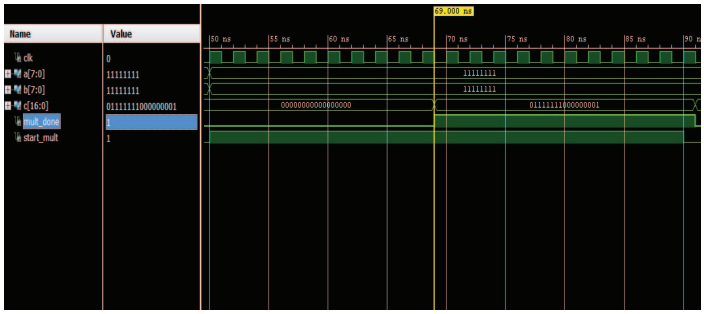

Figure 11
Simulated waveform of Design 3

Figure 11 is the simulated waveform of the 8x8 bits Wallace Tree multiplier. The multiplication also involved two binary numbers with the same value which is 0b11111111 as the multiplicand and multiplier. The output value is 0b1111111000000001 which is the same as the calculated product. It shows that the proposed multiplication method required 9.5 clock cycles which is equals to a duration of 19ns at the clock speed of 500MHz (2 ns per cycle) to complete a single multiplication process.

Implemented Design	Number of cycles to complete
Design 1: Parallel Multiplication by Partitioning of Multipliers	5 cycles
Design 2: Hybrid LUT based Parallel Multiplication	5 cycles
Design 3: Conventional Wallace Tree Multiplication	9.5 cycles

Table 1 Number of cycles for each design.

Implemented Design	Number of logic cells	Number of I/O ports	Number of Nets
Design 1: Parallel Multiplication by Partitioning of Multipliers	146	28	206
Design 2: Hybrid LUT based Parallel Multiplication	645	35	708
Design 3: Conventional Wallace Tree Multiplication	211	34	383

Table 2 Number of resources used for each design.

Table 1 summarizes the processing cycle times consumed to complete a multiplication process. Wallace tree multiplication has the longest processing time among three approaches as there are many logic operations for the partial product reduction. Table 2 tabulated the amount of FPGA resources utilized for each implementation. Hybrid LUT based multiplication utilized the most logic cells as it requires a large number of LUTs to store all possible products but it offers the same number of processing cycles as Design 1 which is the parallel multiplication by partitioning of multiplier to complete a multiplication process. All the approaches above are suitable for multiplication of large bit width multiplicands and multipliers especially Design 1 and Design 2 as they reduce the size of multiplication into smaller parts and complete all the logic operations with shorter processing time by utilizing the parallel environment provided in the FPGA.

VI. CONCLUSION

This paper introduced three parallel multiplication approaches implemented on FPGA. These approaches are suitable for DSP applications such as digital filters, Fast Fourier transform (FFT), etc. Simulation is done on each approach and the output result in the simulated output waveform matches with the pre-computed multiplication output. Therefore, the parallel multiplication by partitioning of multipliers and hybrid LUT based parallel multiplication approaches are more efficient and effective compare to the conventional Wallace Tree multiplications.

VII. ACKNOWLEDGEMENT

The authors would like to acknowledge the support of CREST through the grant number P25C1-13.u

VIII. REFERENCES

[1] Virtex-5 FPGA User Guide, Configurable Logic Blocks (CLBs), UG190 (v5.4) March 16, 2012,

[2] L-K. Wang et al., "Benchmarks and Performance Analysis of Decimal Floating-Point Applications," *Proc. 25th Int'l Conf.Computer Design, pp. 164-170, Oct. 2007.*

[3] Rogers Woods, John Mcallister, Gaye Lightbody, Ying Yi, "FPGA-based Implementation of Signal Processing Systems," *ISBN: 978-0-470-03009-7 2008*

[4] Sandeep K Venishetti, Ali Akoglu, "A highly parallel FPGA Based IEEE-754 Compliant Double-Precision Binary Floating-Point Multiplication Algorithm," *IEEE 2007*

[5] Alvaro V´azquez, Elisardo Antelo, Paolo Montuschi "A New Family of High–Performance Parallel Decimal Multipliers",*18th IEEE Symposium on Computer Arithmetic(ARITH'07)*

[6] Mathias Faust, Chip-Hong Chang, "Bit-Parallel Multiple Constant Multiplication using Look-Up Tables on FPGA", *IEEE 2011*

[7] Pramod Kumar Meher, Senior Member, IEEE "New Approach to Look-Up-Table Design and Memory-Based Realization of FIR Digital Filter", *IEEE 2010*

[8] Jasbir Kaur, Kavita "Structural VHDL Implementation of Wallace Multiplier", *International Journal of Scientific & Engineering Research, Volume 4, Issue 4, April-2013.ISSN 2229-5518*

[9] Himanshu Bansal, K. G. Sharma*, Tripti Sharma "Wallace Tree Multiplier Designs: A Performance Comparison Review" *Innovative Systems Design and Engineering www.iiste.org, ISSN 2222-1727 (Paper) ISSN 2222-2871 (Online), Vol.5, No.5, 2014*