Power and Energy Efficiency Evaluation for HW and SW Implementation of nxn Matrix Multiplication on Altera FPGAs

Abdelghani Renbi
Embedded Systems Department
Jönköping University, Sweden
mems07reab@jth.hj.se

Lennart Lindh
Embedded Systems Department
Jönköping University, Sweden
lennart.lindh@jth.hj.se

ABSTRACT

Matrix multiplication is most often involved in graphics, image processing, digital signal processing, robotics and control engineering applications. In this paper we compared and analyzed the power and energy consumption in three different designs, which multiply two matrices A and B of nxn 32-bit items and store the result in C matrix of nxn 64-bit items. The first two designs use FPGA HW with different number of storage registers 2n and $2n^2$ and the third design uses a computer system piloted by NIOS II\e processor with On-Chip memory. We showed that NIOS II\e is not an energy efficient alternative to multiply nxn matrices compared to HW matrix multiplier on FPGA.

Since our target FPGA is the Altera cyclone II family, we also had to find one acceptable method to measure the real power consumption in the FPGA device.

Categories and Subject Descriptors

B.2.1 [**Hardware**]: Arithmetic and Logic structures, Design Styles; B.6.1 [**Hardware**]: Logic Design, Design Styles; C.1.1 [**Processor Architecture**]: Single Data Stream Architectures; C.1.2 [**Processor Architecture**]: Multiple Data Stream Architectures (Multiprocessors)

General Terms

Design, Algorithms, Performance, Measurement

Keywords

FPGA, Low Power, Matrix multiplication, Energy efficiency, Altera, NIOS Processor

1. INTRODUCTION

In addition to performance, energy efficiency has become an important issue in the design process of mobile embedded systems. Mobile electronics with rich features most often involve complex computation and intensive processing which result in short battery lifetime especially when energy efficiency is not taken in consideration. Both under-availability and over-availability of system resources have a high impact on power and energy dissipation. The work aimed to raise the intensity of this impact by comparing three designs with different resources for matrix multiplication.

Short time to market pressure, high ASIC design costs have made ASIC designers switch to programmable logic devices which have seen an important progress in terms of performance, resources and design size [1]. Today's FPGAs are easily customizable for DSP, high-speed IO standards and other application. The Altera EP2C35F672C6 Cyclone II FPGA was targeted for use in our research work. It is a relatively simple, low cost FPGA that is widely available.

The Cyclone II uses general purpose logic resources called Logic Elements (LEs), and has 4 kb RAM, embedded multiplier and PLL embedded functional blocks [2].

2. RELATED WORK

The area of matrix multiplication has been extensively explored due to the evolution of DSP applications. Similar work has been done by Seonil Choi and Viktor K. Prasanna in [3] within three different platforms: The Xilinx Virtex-II Pro, the Texas Instruments DSP (TMS32OC6415) and the Intel's XScale based PXA250 processor. The Xilinx FPGA platform was the ideal choice for nxn matrix multiplication in terms of energy and latency efficiency between the Texas Instruments DSP and the Intel processor, while the latter lead to the longest latency and offers a slight gain in energy as compared to the Texas Instruments DSP. The Xilinx FPGA matrix multiplication design uses the linear array architecture with 15 processing elements and the algorithm presented in [4].

Other related work is done by Oskar Mencer, Martin Morf and Michael J. Flynn in [5], within Xlinx platform where they compared the throughput of 4x4 matrix multiplier in three different designs. The first design which suffices with the Synopsys FPGA Express II complier and multiple bit-serial multipliers using booth encoding performs the matrix multiplication in 39 clock ticks and allow 15 MHz as the maximum operating frequency, while the other last two designs which employ the parameterized circuit generators with different multiplication algorithm reduce the latency by 31% and 51% of the first design and permit 33 MHz as the maximum operating frequency.

3. POWER MEASURMENT METHODS

Efficient and accurate power estimation tools allow designers to check several implementations for power efficiency before going to the fabrication phase.

Power estimation techniques either use simulation, statistics or probabilistic models to estimate the average dissipated power [6].

Simulation method is characterized by its accuracy and can be applied to any type of technology and architecture, however it is computationally expensive and the result may not reflect the reality due to the pattern-dependence issue, therefore, realistic input patterns are required.

Statistical techniques came to solve the pattern-dependence problem, the main idea of these statistical techniques is common, it is based on applying randomly generated input patterns at the primary inputs and monitoring the convergence of the power dissipation. The simulation is stopped when the measured power is close enough to the true average power.

One of the disadvantages of this approach is that the estimated average power corresponds to the circuit as a whole, while we do not know the dissipated power in each gate or in a group of a set of gates.

Probabilistic models are characterized by their low runtime and weak pattern-dependence as the user is still required to feed the estimator by the information about the typical behavior of the system in terms of probabilities. Probabilistic methods also allow localizing the power concentration in a system, however they are less accurate.

In this section we have evaluated the accuracy of PowerPlay Power Analyzer result against the physical power measurement for cyclone II device where we engaged our HW designs for matrix multiplication for nxn size and a basic NIOS II\e computer system in the comparison.

3.1 Physical Power Measurement

The Altera DE2 development board that we used for this comparison does not facilitate a direct total power measurement, however we found a workaround method to measure accurately the currents drawn by the FPGA device.

Figure 1 shows that the total dissipated power can be divided in two components where the first is due to the IOs and it is contributed by the currents drawn by VCCIO pins, which are connected to 3.3V power supply. The total IOs current has been measured directly by placing the Ammeter instead of the null resistor, which connects the VCCIO pins to 3.3 power supply. This resistor is added in the FPGA board to facilitate measuring the current absorbed by the IOs [2].

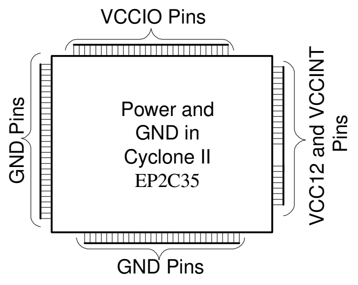

Figure 1. Power Supply Pins in Cyclone II EP2C35.

The second component is due to the currents drawn by VCCINT and VCC12 pins in the FPGA device. VCCINT and VCC12 pins are connected directly to the output of 1.2V regulation circuit. To measure the total current drawn by VCCINT and VCC12 pins, we had to insert a small shunt with 1.25 Ω at the input of the regulator circuit as the complexity of the board does not ease placing the shunt at the output. Measuring the input current instead of the output current makes no difference since the current flowing into the regulator's ground pin is only few micro amps. The below figure illustrates a schematic used for measuring the real drawn currents by VCCINT and VCC12 pins.

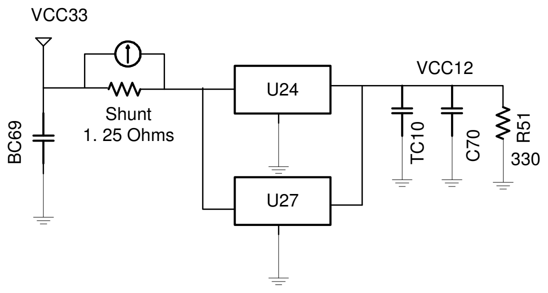

Figure 2. Physical measurement of the power drawn by VCC12 and VCCINT pins.

The measured voltage across the shunt represents the current drawn by VCCINT and VCC12 pins in the FPGA core.

3.2 CAD Power Estimation

Altera provides FPGA designers with a CAD tool for power estimation called PowerPlay Power Analyzer, which can be used to estimate the power after the design is completed. The below figure illustrates the high-level flow of PowerPlay Power Analyzer.

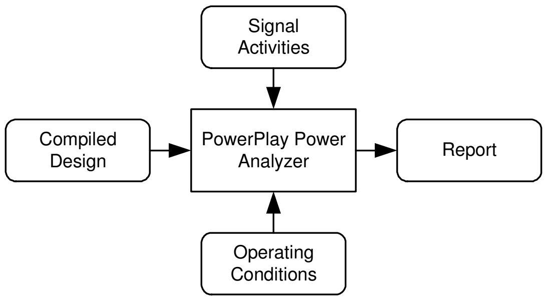

Figure 3. PowerPlay Power Analyzer high-level flow.

As mentioned in figure 3, after the design is synthesized and fit in the target device for place and route information, PowerPlay Power Analyzer requires the operating conditions such as the ambient temperature and other cooling information, it also requires the user to simulate the design for the specified test vectors. During the system simulation, the simulator computes the toggle rate at each node in the design. Once PowerPlay Power Analyzer collects those three inputs, it generates the system power report.

Figure 4 shows that Quartus II PowerPlay Power Analyzer is an accurate tool for power estimation with 7% absolute error in average compared to the real measurement.

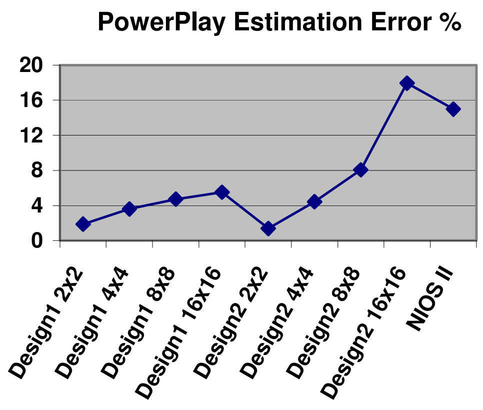

Figure 4. PowerPlay Estimation Error compared to the real measurement.

For PowerPlay Power Analyzer evaluation, we utilized our designs for matrix multiplication, which will be described in the next section. Design1 corresponds to the HW design which is described in section (4.2), Design2 corresponds to the HW design which is discussed in section (4.3) and NIOS II design corresponds to the solution which uses NIOS II/e processor, it is described in section (4.1).

We applied the physical power measurement for the rest of our experiments since we had the opportunity to measure the real power consumption in the FPGA device.

4. THREE DESIGNS

In this section we will present a high-level architecture, algorithm and resources that have been utilized in three different solutions for matrix multiplication. Our solutions fall into two design categories:

- General Purpose Processor based design.
- Single Purpose Processor based design.

Altera NIOS design - C solution, which uses a simple general purpose processor and On-Chip memory to perform the matrix multiplication.

HW design 1 - VHDL program and *HW design 2 - VHDL program,* both solutions will not employ a program memory to perform the matrix multiplication, however they will use different number of resources. *HW design 1 - VHDL program* employs 2n registers for temporary storage elements, n multipliers and n-1 cascaded adders, on the other hand *HW design 2 - VHDL program* uses $2n^2$ registers for temporary storage elements, n multipliers and n-1 cascaded adders.

4.1 Altera NIOS design - C program

In general purpose processor based solution for matrix multiplication, we employed a basic computer system based on a simple core architecture processor, which is NIOS II\e with On-Chip memory and the summation convention algorithm listed in figure 5.

```
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
for (k = 0; k < n; k++) {
C[i][j] = C[i][j] +A[i][k]*B[k][j];
} } }
```

Figure 5. Matrix multiplication using summation convention algorithm.

The power cost due to the above piece of code remains almost constant regardless of the changes in data to be computed and the size of the matrices, on the other hand if we only guess that the design latency is different from the other HW designs, the power will not be a good measure to evaluate the design cost and we have to look at the energy.

NIOS II\e has no HW multiplier, for multiplication purpose, NIOS II compiler engages the Shift-Add algorithm which causes an enormous difference in latency as compared to the HW solutions for matrix multiplication.

The difference springs from the complexity order which is always 3 and the content of the left operand a[i][k]. By the word "content" we refer to the number of 1's and the highest weight in the left operand a[i][k]. Based on NIOS II\e instructions set performance, we built the latency formula for the code illustrated in figure 5 after studying the assembly code generated by the NIOS II compiler. The latency can be written as:

$$L = 366n^3 + 76n^2 + 76n + 28 + n \sum_{i=1}^{n} \sum_{j=1}^{n} \begin{cases} 32(W_{A_{ij}} + 1) + 6N_{A_{ij}} \\ 0 \quad if \quad A_{ij} = 0 \end{cases} \tag{1}$$

where n is the size of the matrix, W is the highest weight and N is the number of 1's in the A_{ij} respectively.

The above equation is fully accurate and it has been verified and crosschecked against the latency computed by the timer method for the program performance.

The best latency case is when multiplying a zero matrix:

$$L = 366n^3 + 76n^2 + 76n + 28 \tag{2}$$

while the worst case for 32–bit multiplication is when all weights are equal to 31 and the number of 1's in all items is equal to 32:

$$L = 1582n^3 + 76n^2 + 76n + 28 \tag{3}$$

Based on the above analysis, the Shift-Add algorithm latency depends only on the content of the left operand a[i][k], one can always exchange the operands for better latency [7].

4.2 HW design 1 - VHDL program

In the first HW solution for matrix multiplication, we employed 2n registers for temporary storage elements with n multipliers and n-1 cascaded adders. Figure 6 illustrates the high-level architecture of the design where the matrices A, B and C use separate memory blocks.

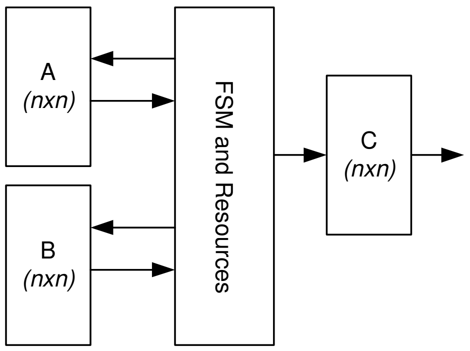

Figure 6. High- level architecture for HW solution for matrix multiplication.

The below figure shows the used algorithm, which allows a full exploitation of the resources and produce the matrix C items as soon as data is available from A rows and B columns, each C matrix item is computed after each n clock ticks.

```
For i =1 to n
Read A_Row$_i$ and B_ Column$_1$
Compute C$_{i1}$
For j =2 to n
Read B_Column$_j$
Compute C$_{ij}$
 Next j
 Next i
```

Figure 7. High-level algorithm for the first solution.

To compute a row in C Matrix we need n readings from A and $(n-1)n$ readings from B matrix as the first column in B matrix is always read concurrently with any row of A matrix. For n rows the latency is given by:

$$L = [n + (n-1)n]n = n^3 \tag{4}$$

The complexity order remains 3 due to the memory access bottleneck as we are shorthanded in terms of temporary storage elements, the first n registers set are dedicated to A rows and the second set of n registers are dedicated to B columns. To compute the whole matrix C, each row in A needs to be fetched only once while B columns need to be fetched n times. The matrices separation in different memory blocks permits writing the computed items concurrently and reading the first column of B matrix with any row of A matrix in parallel. This parallelism contributes to reduce the latency by eliminating the exponentiation component n^2 from the latency equation, this exponentiation part would appear if we are restricted to write C matrix in a separate clock cycle when using only one memory block.

The below figure show the timing diagram of the availability of C matrix items at the output of the adder and at the memory locations of C block.

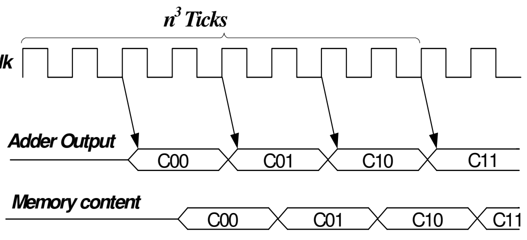

Figure 8. Timing Diagram for 2x2 size matrix multiplication.

4.3 HW design 2 - VHDL program

In this HW solution for matrix multiplication, we utilized the same architecture been used in the previous solution, we also kept n multipliers and n-1 cascaded adders, however we increased the resources in terms of temporary storage registers instead of 2n registers we employed $2n^2$ to prevent the memory access bottleneck. The below figure illustrates the high-level algorithm been used to compute the C matrix.

```
For i =1 to n
Store A_ Row_i and Store B_Column_i
Compute C Matrix items as soon as data is available
Next i
```

Figure 9. High-level algorithm for the second solution.

By the use of the above algorithm and $2n^2$ registers we managed to reduce the order of the complexity to 2. During the first n^2 cycles, we store A rows and B columns and at the same time we compute the first C row as soon as the data is available to perform the computation.

During this period, the storage of one row of A matrix and one column of B matrix is done after each n cycle, as well one item of the first row of C matrix is computed.

After n^2 clock ticks, A and B are already stored and no further memory access is needed to fetch A and B items. This time we produce the rest of C matrix items every clock cycle.

The latency can be given as:

$$L = n^2 + n^2 - n = 2n^2 - n \qquad (5)$$

Figure 10 shows the timing diagram of the availability of C matrix items at the output of the adder and at the memory locations of C block.

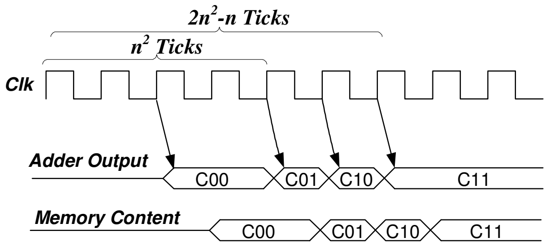

Figure 10. Timing Diagram for 2x2 size matrix multiplication.

5. RESULTS AND ANALYSIS

In this section we will analyze the result and discuss the sources of power and energy in our designs. We will also observe the huge difference between the power and energy, as these two terms have been freely interchanged in the area of low power design.

5.1 Power

For fair comparison, all these three designs perform equal matrix multiplication at 50 MHz.

It appears that the NIOS II processor dissipates the same amount of power regardless of the problem size and switching activity as shown in figure 11.

Obviously the dissipated power depends only on the switching activity in this case, the question might be posed why the consumed power remains almost constant. This is due to the fact that the switching activity happens in the core, and the majority of the power is contributed by the holding logics involved. The increase in the dynamic switching activity due to data changes is not dominant compared to the static power dissipated by the whole FPGA core and the dynamic power consumed by the functional units and the IOs, on the other hand, when feeding the processor with different problem size and different data content, the same functional units will operate at any case. We can conclude that the NIOS II processor is not an extreme case design to observe the effect of the internal switching activities [2].

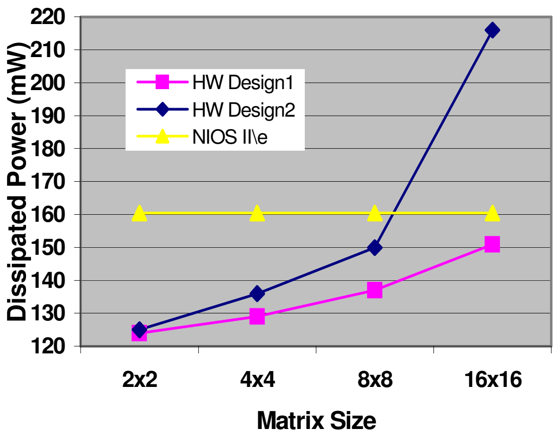

Figure 11. Dissipated power versus matrix size in the three different designs.

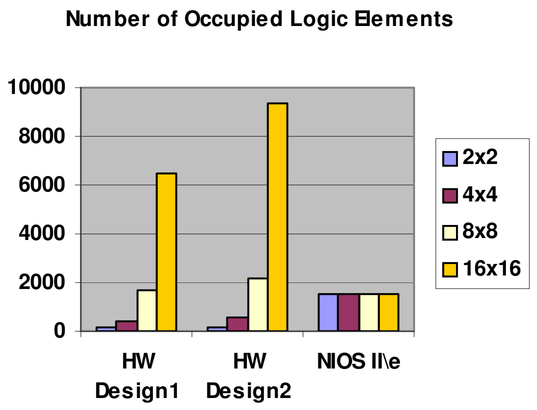

Figure 12. Number of occupied logic elements in the three different designs.

The experiment also shows that HW Design1 uses less power as compared to NIOS II and HW Design2 solutions. It dissipates 6% to 23% less power than NIOS II and 16% less in average for all problem sizes. By increasing the problem size we increase the number of logic signals and therefore the dissipated power increases due to the increase of the number of switching activities per second. Although NIOS II area is smaller than the HW Design1 for the matrix size 8x8 and 16x16, the consumed power is bigger, this also means that NIOS II solution results in higher effective capacitance, one can deduce that a bigger design does not always signify higher dissipated power. If we take two designs, which are supplied by the same voltage and operate at the same frequency, the higher dynamic power design will be the one with higher total effective capacitance, which means the sum of the load capacitances multiplied by the number of transitions per clock cycle as it is given in the following equation:

$$P_{Dynamic} = \frac{1}{2} V_{DD}^{2} f_{Clk} \sum_{i=1}^{n} C_i \alpha_i \qquad (6)$$

where f_{Clk} is the operating clock frequency, V_{DD} is the operating voltage, C_i and α_i are the load capacitance and the number of transitions per clock cycle of the i_{th} node respectively.

The smaller design will dissipate higher power, if it has a higher total effective capacitance as compared to the bigger design [7].

For small sizes matrix multiplication, such as 2x2 and 4x4, Design2 dissipates almost equal power as compared to Design1, precisely it dissipates 0.8% and 5% more power for 2x2 and 4x4 respectively. On the other hand the effect of $2n^2$ temporary storage registers shows up only in bigger sizes such as 8x8 and 16x16 where Design2 dissipates 9% and 43% more power as compared to Design1 for 8x8 and 16x16 respectively.

For the sizes 2x2 and 4x4 matrix multiplication, Design2 consumes 22% and 15% less power as compared to NIOS II solution respectively. However for bigger sizes, the dissipated power in Design2 almost reaches NIOS II solution for 8x8 size with 7% less dissipated power and surpasses the NIOS II consumed power by 35% for 16x16 size, this is due to the high number of clocked registers, which are forming $2n^2$ temporary storage elements [8].

5.2 Energy

The energy can be computed by multiplying the average power by the latency. As we mentioned in section 4.1, if the designs have different computation time and especially if they are targeted for mobile embedded systems, which are often battery powered, the average dissipated power is not a measure for battery consumption concern. Of course power is always good for heat dissipation concern. The below figure shows how much energy is dissipated per design for all matrix sizes multiplication.

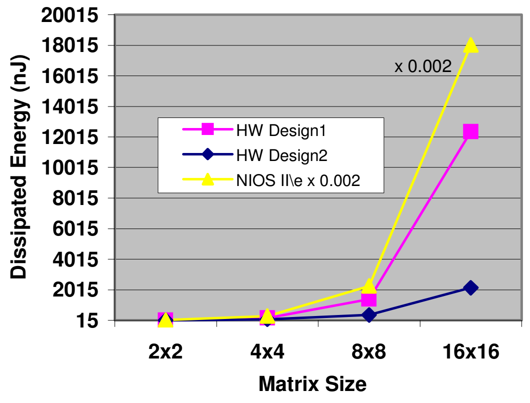

Figure 13. Dissipated energy versus matrix size in Design1, Design2 and NIOS II\e solution.

Latency imparts a high impact on energy efficiency. Design2, which draws higher current and occupies more area as compared to Design1, is more economical for performing the matrix multiplication.

The saved energy follows the difference in latency $n^3 - 2n^2 - n$ and it is increasing dramatically with n.

The below graph illustrates how much energy is saved when moving from Design1 to Design2.

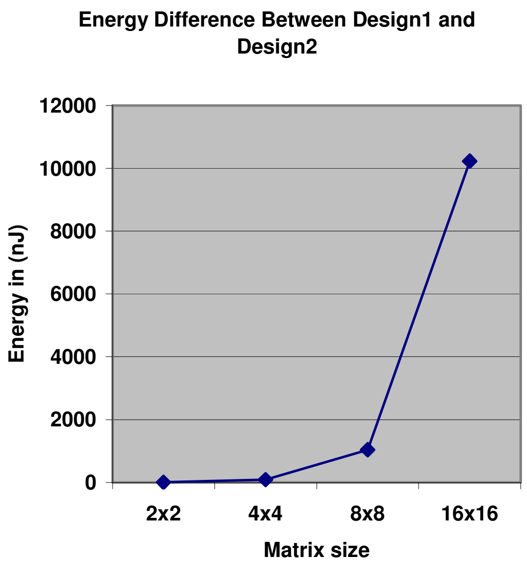

Figure 14. Saved energy versus matrix size when using Design2.

We can observe that the second design offers an important energy saving especially for bigger sizes matrix multiplication such as 8x8 and 16x16.

Figure 13 also shows that NIOS II\e is far to compete with both HW designs for energy saving as the difference is huge due to the lack of embedded HW multiplier in the processor. The energy is wasted mainly when performing the multiplication using the Shift-Add algorithm. NIOS II\e dissipates 1271 to 4208 times more as compared to Design2 and dissipates 961 to 729 times more as compared to Design1. These differences in energy dissipation are valid only for the matrices used in our experiments as the difference depends on the number of 1's and the highest weight in the left operand a[i][k], although if we look at the best latency case, where A is a zero matrix with 2x2 size, the gap is still large where NIOS II\e will consume 21 and 28 times more as compared to Design1 and Design2 respectively [8].

6. CONCLUSION AND FUTURE WORK

Based on the conducted research work, we can draw a conclusion that Design2 which uses $2n^2$ temporary storage elements is a fast and energy efficient design for nxn matrix multiplication compared to Design1, which uses 2n temporary storage elements. Design2 is more rapid than Design1 because of the enough storage registers to hold the matrix items and minimize the memory access. Design1 latency order is 3 due to the memory access bottleneck as it employs only 2n temporary storage elements. However for big sizes matrix multiplication, Design2 is not a good choice under heat constraints because of the high number of the clocked registers. Moreover, based on the design requirements we can always make the trade-off between the design area, the dissipated power, the computation latency and the energy. On the other hand NIOS II\e processor is not a good choice for matrix multiplication especially for battery powered embedded systems due to its slowness when performing the multiplications.

We also witnessed that PowerPlay Power Analyzer is reliable tool for power estimation with 7% absolute error in average. However for accurate power awareness, the physical power measurement always leads to the best result as it is recommended in [2].

For future work, it is very much worthwhile to use the NIOS C2H acceleration compiler and see how much it will impact the energy dissipation using NIOS II\e. On the other hand, to enrich our research with more information, it is important to involve other processors which come with their own embedded HW multiplier and other architecture rather than sufficing with a simple RISC processor. Clock gating is a possibility to reduce the power and in a future work this would be interesting to study.

One can also evaluate the energy efficiency for the component library for matrix multiplication provided by different vendors such as Altera and Xilinx.

7. ACKNOWLEDGMENTS

The authors thank all the reviewers from the FPGAWorld academic program committee for valuable feedback on earlier version of this paper.

8. REFERENCES

[1]. Behrooz Zahiri. (2003). Structured ASICs: Opportunities and Challenges. *IEEE Computer Society.* ISBN: 0-7695-2025-1

[2]. Altera Corporation. www.altera.com.

[3]. Seonil Choi and Viktor K. Prasanna. (2002). Energy Efficiency of FPGAs and Programmable Processors for Matrix Multiplication. *IEEE International Conference*. ISBN: 0-7803-7574-2.

[4]. Ju-Wook Jang, Seonil Choi and Viktor K. Prasanna. (2005). Energy- and Time-Efficient Matrix Multiplication on FPGAs. *IEEE International Conference.* IISSN: 1063-8210.

[5]. Oskar Mencer, Martin Morf, Michael J. Flynn.(1998). PAM-Blox: High Performance FPGA Design for Adaptive Computing. *IEEE Computer Society.* ISSN: 1082-3409.

[6]. Farid N. Najm.(1994). A Survey of Power Estimation Techniques in VLSI Circuits. *IEE Tanrs.* ISSN:1063-8210.

[7]. Kaushik Roy, Sharat C. Prasad. (1999). Low-Power CMOS VLSI Circuit Design. A Wiley-Interscience publication. ISBN-0-471-11488-X. Pages 143-200, 321-348.

[8]. Abdelghani Renbi. (2009). Power and Energy Efficiency Evaluation for HW and SW Implementation of nxn Matrix Multiplication on Altera FPGAs. Master's thesis, Jönköping University.

[9]. Ricardo Gonzalez and Mark Horowitz. (1996). Energy Dissipation In General Purpose Microprocessor. *Journal Of Solid-State Circuits.* ISSN: 0018-9200.