ET4074 - Modern Computer Architectures Laboratory Assignment No. 2

Aniket Ashwin Samant
Delft University of Technology
Delft, The Netherlands
Email: A.A.Samant@student.tudelft.nl

Apoorva Arora
Delft University of Technology
Delft, The Netherlands
Email: A.Arora-1@student.tudelft.nl

Snehal Jauhri
Delft University of Technology
Delft, The Netherlands
Email: S.Jauhri@student.tudelft.nl

I. Introduction

The aim of this lab assignment is to use an actual SoC design with one or more ρVEX VLIW processor cores to run a small workload consisting of 4 Powerstone benchmark applications, namely, *matrix.c, convolution_3x3.c. crc.c, and fir.c*. The task is to create an SoC design configuration that will run the workload as efficiently as possible based on certain criteria - performance, area utilization, and energy consumption. The report describes three designs accordingly: one for achieving the best performance, one with the best energy efficiency, and finally a design which provides a good balance of energy, area, and performance. We run the benchmarks on a real ρVEX core implemented on a Xilinx FPGA board and obtain the execution statistics.

II. Our Benchmarks

A. Matrix

The Matrix benchmark (a C program) tests the result of matrix multiplication of two 64x64 matrices. Three matrices, a[64][64], b[64][64] and Result[64][64] are provided, and the matrix c[64][64] is computed as the product of the multiplication of a and b. c is compared with Result by checking the equality of the matrices, based on which "Matrix Test Passed" is printed (if all elements are equal), else "Matrix Test Failed". The two matrices are multiplied using three nested *for* loops which involve additions and multiplications of the elements of the a, b, and c matrices. Thus as expected, the assembly file showed *large numbers of multiplication operations along with load, compare and addition.*

B. FIR

FIR or finite impulse response system is a filter whose impulse response, that is the response to a unit sample input (an input of finite length) is of finite duration which means it settles to zero in a finite time. Analysis of the assembly file of FIR benchmark showed many functions like sin, cos, square root, tan, absolute, rand and hence a lot of function calls are observed. Each of the functions involves mainly load, compare, store and addition operations. Next to nil multiplication operations are observed which means that there will be a *limited use of multipliers for this benchmark.*

C. Convolution_3X3

A convolution is an integral that gives the amount of overlap of one function as it is shifted over another function. It therefore expresses how the shape of a function f is modified by another function g. The powerstone benchmark-convolution_3x3 is a C program that runs a predefined 3x3 filter over a 64x64 image in order to perform a convolution operation on it. The convolution is carried out using two *for* loops, one for the width of the image and one for the height. The RGB values obtained by convolving the values of the filter and the image's pixels are stored in a new array of unsigned integers (the R,G, and B values are stored in the same location using left shift operators). On completion of the convolution operation, a string is printed to the terminal stating that the convolution is finished. Analyzing the convolution.s assembly file, we identify *'add', 'compare' and 'load' operations are dominating for the common case, that is, Trace 1* (main())

D. CRC

CRC, or Cyclic Redundancy Check, is a technique used for detecting errors in any digital data transmission using communication protocols like Ethernet, Bluetooth, etc. It uses a *checksum*, that is, a frame or a field which is appended to the original transmit data to detect errors introduced while transmitting. In *crc.c*, execution statistics show a large number of shift and other logical operations. On analyzing the assembly code, *a large usage of addition, compare, load and store, move and shift operations is observed.* There are no multiplication operations in the assembly file. Function calls within loops are observed.

III. Initial Design Space Exploration for individual benchmarks

Initially, a design space exploration is performed using the ρVEX simulation method from Lab Assignment 1 to find the optimum values of the issue width and the number of multipliers required, by observing the variation in ILP for each of the benchmarks. The results of this DSE for the new benchmarks are presented in Table I. Since the real ρVEX processor provided on which the benchmarks are finally run does not have as many configuration options as the VEX simulator, we can only vary the issue width (2, 4, or 8) and

Table I
DATA FOR THE FIR AND CRC BENCHMARKS' DSE PROCESS

Issue Wdth	Num Mults	Exec cycs, FIR	ILP, FIR	Exec cycs, CRC	ILP, CRC
2	1	509595	1.31	15745	1.74
2	2	509595	1.31	15745	1.74
4	**1**	**505944**	**1.71**	**13361**	**1.85**
4	2	505945	1.71	13361	1.85
8	1	505179	1.79	13353	1.85
8	2	505157	1.79	13353	1.85

the number of multipliers in the machine configuration file. The number of clusters is kept one.

On analyzing and comparing the assembly files of the benchmarks, we observe the the following:

- *Matrix* is a big program with the maximum number of instructions among the four benchmarks, and involves a lot of multiplication operations. Hence it is the heaviest consumer of processor resources (especially issue width and multipliers.)
- *FIR* has the second largest program length and has a very limited requirement for multipliers.
- *Convolution_3x3* and *CRC* are comparatively smaller programs and hence require less processor resources.

As we can see, the data in Table I obtained from the DSE of the new benchmarks conforms with the above observation, as can also be verified by analyzing the assembly files. From our analysis of the *Matrix* and *Convolution_3X3* benchmarks in Assignment 1 as well as the new initial analysis of *FIR* and *CRC*, we can conclude that:

- Since *Matrix* is a large program which contains a lot of resource hungry multiplications, its performance improves quite significantly on increasing the issue width. Similarly, increasing the number of multipliers also improves performance. Hence, executing it on a core with a large issue width (4 or 8) with around 4 multipliers is expected to work the best for the high performance case.
- For *FIR*, an issue width of 4 seems optimum as seen from Table I. Also, increasing the number of multipliers has no effect. Thus, a single multiplier is allocated to this benchmark in later exploration.
- We know from previous analysis in Assignment 1 that *Convolution_3X3* gives the best performance with a large issue width, and at least one multiplier. However, given that the program is relatively small in size, we could consider allocating fewer resources to this benchmark for an area efficient design.
- Since *CRC* is a relatively small program which requires less processor resources, a minimum issue width of 2 with 1 multiplier is expected to be optimum for an area efficient design.

IV. SEARCHING THE DESIGN SPACE

In the `configuration.rvex` file, we see there are 4 major design parameters for each core, namely the number of multipliers, the issue width of the cores, cache size (instruction cache, data cache) and the number of stop bits. These, along with the number of cores allocated for the workload, affect the performance, area, and energy consumption of the ρVex VLIW processor. Our main strategy involves *choosing the distribution of programs between cores and configuring each core* for performance, area and energy optimization. We came up with this strategy after observing the following:

- Starting with execution of the default configuration, that is, 1 core, 4 issue width, 4 multipliers, 16Kb I\$, 4Kb D\$ and no stop bits, we observed from the generated results that Matrix took the maximum number of execution cycles followed by FIR, Convolution_3X3, and CRC benchmarks, which took significantly fewer execution cycles. From this we understand that Matrix benchmark is the bottleneck. Thus, optimizing other benchmarks individually would not lead to a design improvement. Instead, we need to divide the benchmarks between cores and try to allocate maximum resources to *Matrix* while still running the other cores with the other benchmarks in reasonable time.

In this section, we discuss the effect of each parameter on the execution of the benchmarks. The observations from this section provide inputs to our main Design Space Exploration process whose strategy is discussed in the next section.

A. Issue Width

The issue width roughly represents the number of parallel syllables that can be fetched by the processor at a time, and hence it makes intuitive sense to keep its value as high as possible for benchmarks that can execute a lot of parallel, independent operations; this is to reduce the number of execution cycles taken for running the application (at the cost of increased area and power consumption though). As an example of this, we see that for the *Matrix* benchmark, based on the analysis of its assembly file, there are a lot of parallel multiplication operations that can be performed. Thus, we can say that an issue width with a large value (4 or 8) should be used for it for the best performance. However, for the other three benchmarks, the initial DSE suggests that a relatively lower issue width suffices without sacrificing the performance drastically. The advantage of keeping it low is seen in the form of overall area reduction.

B. Cache Size

Theoretically, applications that require frequent main memory accesses for fetching data should see an improvement with an increase in the data cache size. Similarly, we can reason that the instruction cache size plays an important role in applications that require frequent instruction fetches. Based on this, we analyze our benchmark applications. An example of the Cache Size's impact is illustrated in Figure 1. As shown, for the *Matrix* benchmark, since the data read miss (DRmiss) and write miss (DWmiss) values reduce significantly by increasing the data cache (DCache) from 8K to 16k, we see a drastic improvement in performance (reduction of number of Execution Cycles). Since we do not see much improvement in on increasing it further, we can conclude that a DCache of 16k is optimum for *Matrix* given that the area also shoots up on increasing Data Cache size.

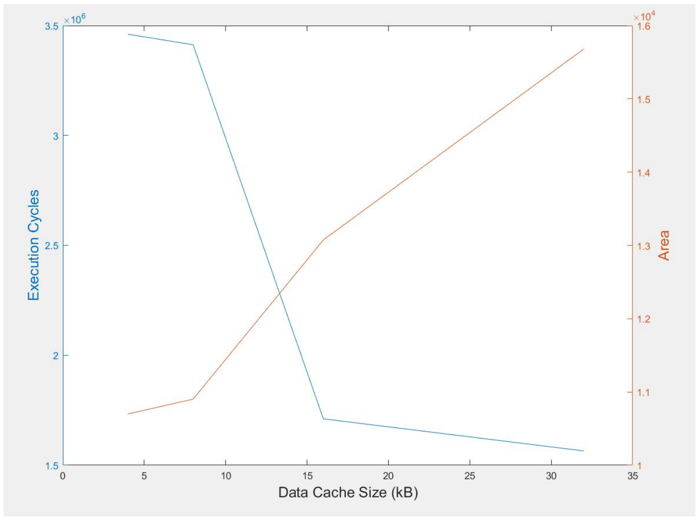


Figure 1. An example of how Data Cache size impacts Performance and Area Consumption (*Matrix.c* with an 8-way core)

C. Number of Stop Bits

The stop bit functionality allows an improvement in performance by enabling the processor to execute a smaller number of instructions when NOPs are detected. Say, when a 4 issue width VLIW has 2 of the instructions to be executed as NOPs, the stop bit is set, which indicates that the remaining 2 instructions are NOPs and can be skipped. Instead, the processor can start executing the next 4-issue width instruction line. This functionality is especially useful in improving performance when dealing with larger issue width cores. The area increase when using stop bits is minimal since extra area is only needed for the logic circuitry to implement the stop bit functionality. The impact of using stop bits is illustrated in Figure 2. We notice a significant reduction in execution cycles but only a tiny increase in area.

We start our exploration on the basis of the above observations and then analyze the effect of varying the parameters on: (i) total no. of execution cycles, (ii) area, and (iii) energy consumption. We vary the issue width from 2 to 16, the number of cores, and the data and instruction cache sizes. The comparison of results for different configurations provides insights into how these affect performance, area and power consumption. For example, **we observe that the *matrix* benchmark takes more executions cycles than the other benchmarks and is also significantly affected by the cache size. This, combined with our initial analysis, suggests that giving it an exclusive core, more multipliers, a higher issue width, and a larger cache is expected to improve the overall performance.**

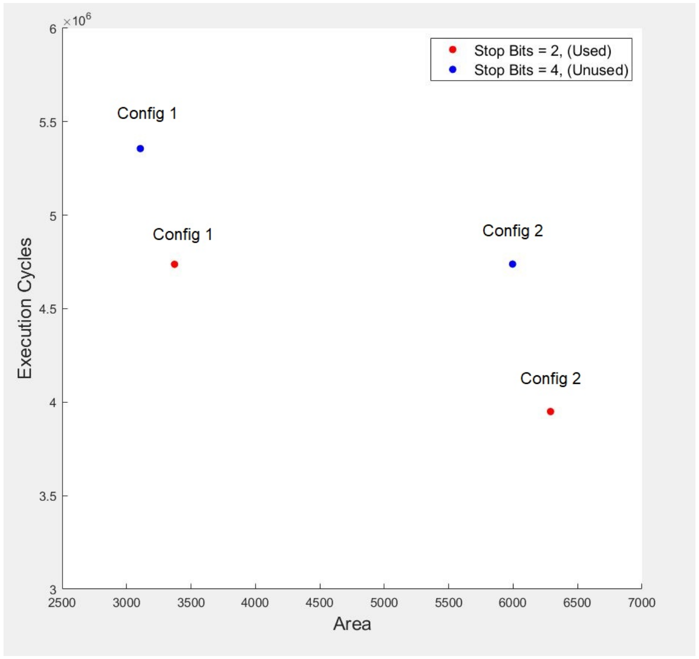


Figure 2. An example showing how usage of Stop bits for the same configuration leads to significant performance improvement while not increasing much.

V. Design Configurations

The main objective is to determine the best configuration(s) for each design approach: (i) performance maximization, (ii) energy minimization, and (iii) a balanced design with a good performance-energy-area trade-off. To achieve this, we create several configurations by varying the parameters discussed in the previous section. The analysis of each of the parameters separately and DSE results for individual benchmarks is used to group benchmarks with similar resource requirements together (especially in multi-core environments) for optimal utilization of resources. We start with the execution of all the benchmarks on a single static core and then move towards multi-core configurations in search of possibly better performances. Some of the configurations that were executed are included in Appendix I. The results of the runs are retrieved from the board server after running the designs on the FPGA bank.

We start with a given default configuration as follows:

- Issue Width = 4
- No. of cores = 1
- No. of Multipliers = 4
- I$ size= 16 KiB
- D$ size= 4 KiB
- No stop bit (i.e. STOPBIT = 4)

The results of this configuration are:

- Total Execution Cycles: 4244362
- Area: 4017.2
- Energy: 2.2 mJ

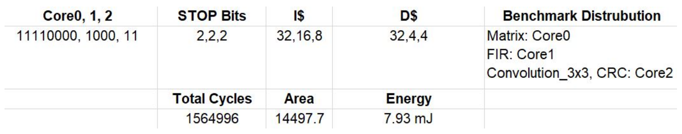
This default configuration is considered the starting point reference for further optimization for each design approach.

A. *Configuration for Highest Performance*

The *Matrix benchmark is the heaviest user of processor resources, followed by FIR. Convolution_3x3 and CRC are comparatively smaller programs and require less resources for functioning*. We exploit this information for optimal distribution of resources - we start with (i) a 4-issue core exclusively for running Matrix with 4 multipliers, and (ii) another 4-way core for the rest of the benchmarks, also with 4 multipliers. To further improve the performance we proceed to a higher issue width of 8, with 8 multipliers for core0 running Matrix. As expected, this improves the performance but also increases the area; but in this design's motive, we consider only the performance numbers. Thus, for a better design, we add another core, ending up with the following optimum three-core design:

1) an 8-way Core0 running Matrix, with 4 multipliers,
2) a 4-way Core1 running FIR with the minimum number of multipliers (that is, 1), and
3) a two-way Core2 for CRC and Convolution_3X3, with two multipliers.

Increasing the issue width beyond 4 does not significantly increase the performance of Core1. We observe that increasing both the instruction and data cache size from 16 KiB to 32 KiB for core0 significantly improves the performance. For core1, a instruction cache size of 16 KiB and Data Cache size of 4 is taken to be appropriate, since increasing it further, while improving core1's performance, does not affect the overall performance (it is lower bounded by the performance of *Matrix* in core0). For instance, in one run, the number of execution cycles with D$ = 4 KiB, I$ = 16 KiB for FIR is 130,000. Increasing the cache to D$ = 8 KiB, I$ = 32 KiB decreases the cycle count to 110,000. But the number of execution cycles for Matrix (around 160,000) determines the overall performance. Core2 is given 8 KiB of I$ and 4 KiB of D$ since the CRC and Convolution benchmarks are small programs which exhibit good performance even with limited resources. Finally, providing 2 stop bits to each of the cores results in a significant decrease in the number of execution cycles as opposed to a higher number.



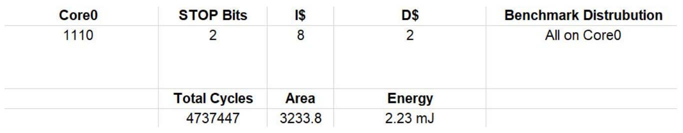
Core0, 1, 2	STOP Bits	I$	D$	Benchmark Distrubution
11110000, 1000, 11	2,2,2	32,16,8	32,4,4	Matrix: Core0 FIR: Core1 Convolution_3x3, CRC: Core2
	Total Cycles	**Area**	**Energy**	
	1564996	14497.7	7.93 mJ	

Figure 3. High Performance Configuration

B. *Configuration for Low Power Consumption*

The aim of this approach is to minimize the FPGA area utilized and energy consumed without a significant degradation of performance.
Starting with the default configuration, we perform the following steps to arrive at an optimum configuration for low power consumption.

- We first reduce the data and instruction cache size by half to 2 KiB and 8 KiB respectively; however, this improves the area only slightly. We set STOPBIT = 2. This improves the performance significantly without increasing the area too much.
- We analyze the energy, area and performance results employing 2 cores (11,11) both having 2 multipliers each and both cores having 16 KiB of I$ and 4 KiB of D$. We observe that the area increases too much and also the performance degrades significantly. The reason for this result for the configuration is that since matrix benchmark is a large program, it requires a high issue width and a large number of multipliers.
- Hence, keeping in mind the above observation, we go with a single core, and optimize the no. of multipliers and cache size for minimal power consumption. We arrive at the following result: a four-way single Core design with three multipliers.



Core0	STOP Bits	I$	D$	Benchmark Distrubution
1110	2	8	2	All on Core0
	Total Cycles	**Area**	**Energy**	
	4737447	3233.8	2.23 mJ	

Figure 4. Low Power Configuration

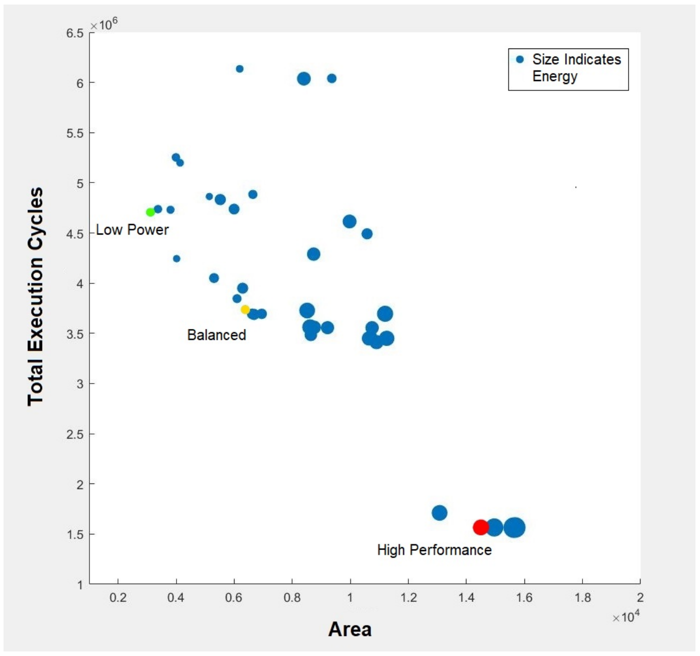


Figure 5. **Overall Design Space Exploration**. Size of the circles indicates Energy consumption. The chosen configurations for Low Power, High Performance and Balanced Design are highlighted.

C. *Energy-Area-Performance Compromise*

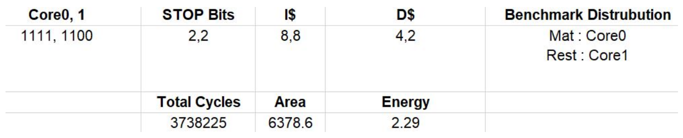
We use the results of our overall Design Space Exploration (shown in Figure 5) to arrive at a well balanced design.

We notice for this Overall DSE plot (Figure 5) how performance improvement comes at the cost of area increase. Moreover, we also notice how energy consumed and area utilized are closely correlated since the size of the circles (indicating Energy) increases as we move from the left to the right of the plot (increasing area).

The well balanced design is chosen with the objective of achieving a reasonable performance while not increasing the area too much. For this, we finally choose a 2 core design with *Matrix* occupying one core while the rest of the benchmarks (which are smaller) occupying the other core. Issue width for *Matrix* is kept at 4 and the number of multipliers for this issue width is maximized for the balanced design. We arrive at the following result:

- a 4-way Core0 running Matrix, with 4 multipliers,
- a 4-way Core1 running FIR, Convolution_3x3 and CRC with 2 multipliers.

Reasonable performance is reached with a Cache Size of 8 KiB and 4 KiB (Instruction and Data) for core0 and smaller cache sizes for core1 (8 and 2 KiB respectively).



Core0, 1	STOP Bits	I$	D$	Benchmark Distrubution
1111, 1100	2,2	8,8	4,2	Mat : Core0 Rest : Core1
	Total Cycles	**Area**	**Energy**	
	3738225	6378.6	2.29	

Figure 6. Balanced Design Configuration

VI. Conclusions and Chosen Design for the 4 Applications' domain

The Matrix benchmark is the main bottleneck in the workload since in the default configuration it does not get the resources it needs. After adding more resources (multipliers, I- and D-cache, increasing the issue width) and assigning a core exclusively to it, we see that the performance increases by about 50 %. However, the area increases as well since the additional components require a lot of FPGA slices. The other applications require far less in terms of hardware resources, and hence needn't have exclusive cores each. Our decisions to allot cores were made accordingly.

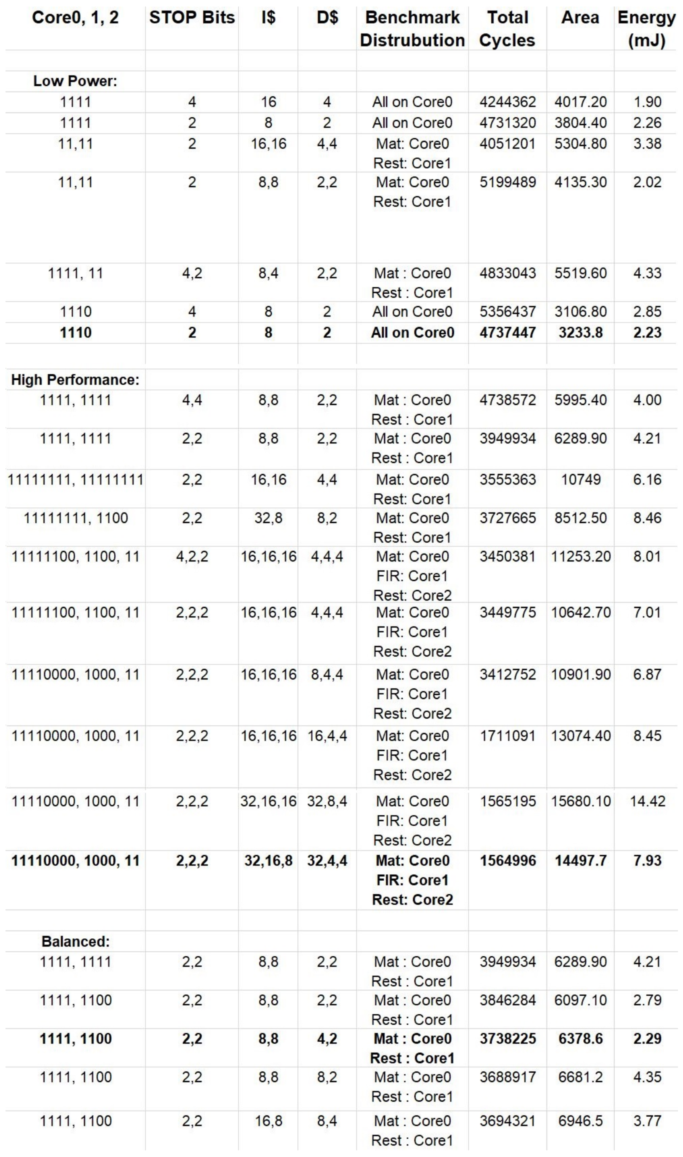
Overall, we see how changing the various parameters can greatly affect the results based on the approach taken (for instance, for maximizing performance, the area increase wasn't taken into consideration in the decisions, whereas for area minimization, the impact on performance wasn't considered.) We thus conclude that choosing a configuration ultimately depends on the intended use case(s). *The applications provided to us (matrix, FIR, CRC and convolution) are all related to communication and signal (image) processing, and probably are ones that run in embedded processors in handheld cameras, cellphones, and the like. Though our High performance configuration minimizes the execution cycles in an optimal way, its area is high and its I$ and D$ numbers of 32 each are unrealistic for such a scenario. Thus, the performance-energy-area compromise design seems best suited for the four applications.*

VII. Reflection: Learnings from this Assignment

This assignment introduced us to the process of choosing hardware resources keeping in mind the requirements of an application (or multiple applications in parallel) intended to be run on a processor, and the impact of varying the parameters associated with it. It also got us familiarized with the reasoning behind the splitting of a processor into multiple cores, and how power savings and performance improvements can be achieved with simple tweaks to the configuration parameters. Moreover, we also learned that performance almost always comes at a cost beyond a certain point of resource optimization, in the form of increased area and/or energy usage. We now know that parallelization is not as straightforward as it may initially seem since it requires thinking from a different paradigm and perspective and depends heavily on the processor of allotting resources to cores, and on the applications as well. Hence it is necessary to gauge the applications' requirements and the design constraints through analyses and simulations before synthesizing the hardware as otherwise a lot of extra time and effort would be consumed. It was a good, hands-on, learning experience in computer engineering, given that we actually synthesized processors on FPGAs and ran applications on them.

VIII. Appendix

List of important configurations:



Core0, 1, 2	STOP Bits	I$	D$	Benchmark Distrubution	Total Cycles	Area	Energy (mJ)
Low Power:							
1111	4	16	4	All on Core0	4244362	4017.20	1.90
1111	2	8	2	All on Core0	4731320	3804.40	2.26
11,11	2	16,16	4,4	Mat: Core0 Rest: Core1	4051201	5304.80	3.38
11,11	2	8,8	2,2	Mat: Core0 Rest: Core1	5199489	4135.30	2.02
1111, 11	4,2	8,4	2,2	Mat : Core0 Rest : Core1	4833043	5519.60	4.33
1110	4	8	2	All on Core0	5356437	3106.80	2.85
1110	**2**	**8**	**2**	**All on Core0**	**4737447**	**3233.8**	**2.23**
High Performance:							
1111, 1111	4,4	8,8	2,2	Mat : Core0 Rest : Core1	4738572	5995.40	4.00
1111, 1111	2,2	8,8	2,2	Mat : Core0 Rest : Core1	3949934	6289.90	4.21
11111111, 11111111	2,2	16,16	4,4	Mat: Core0 Rest: Core1	3555363	10749	6.16
11111111, 1100	2,2	32,8	8,2	Mat: Core0 Rest: Core1	3727665	8512.50	8.46
11111100, 1100, 11	4,2,2	16,16,16	4,4,4	Mat: Core0 FIR: Core1 Rest: Core2	3450381	11253.20	8.01
11111100, 1100, 11	2,2,2	16,16,16	4,4,4	Mat: Core0 FIR: Core1 Rest: Core2	3449775	10642.70	7.01
11110000, 1000, 11	2,2,2	16,16,16	8,4,4	Mat: Core0 FIR: Core1 Rest: Core2	3412752	10901.90	6.87
11110000, 1000, 11	2,2,2	16,16,16	16,4,4	Mat: Core0 FIR: Core1 Rest: Core2	1711091	13074.40	8.45
11110000, 1000, 11	2,2,2	32,16,16	32,8,4	Mat: Core0 FIR: Core1 Rest: Core2	1565195	15680.10	14.42
11110000, 1000, 11	**2,2,2**	**32,16,8**	**32,4,4**	**Mat: Core0 FIR: Core1 Rest: Core2**	**1564996**	**14497.7**	**7.93**
Balanced:							
1111, 1111	2,2	8,8	2,2	Mat : Core0 Rest : Core1	3949934	6289.90	4.21
1111, 1100	2,2	8,8	2,2	Mat : Core0 Rest : Core1	3846284	6097.10	2.79
1111, 1100	**2,2**	**8,8**	**4,2**	**Mat : Core0 Rest : Core1**	**3738225**	**6378.6**	**2.29**
1111, 1100	2,2	8,8	8,2	Mat : Core0 Rest : Core1	3688917	6681.2	4.35
1111, 1100	2,2	16,8	8,4	Mat : Core0 Rest : Core1	3694321	6946.5	3.77