ET4074: Lab Assignment 2

Kaustubh Agarwal(4823168) and Yuanhao Xie(4624424)

I. INTRODUCTION

The aim of this assignment is to design a preliminary architecture for a VLIW Single/Multi-core processor optimal in terms of performance, area utilization and energy consumption for the given four applications- Matrix.c, Pocsag.c, Fir.c and Bcnt.c. This report documents the Design Space Exploration process by varying architecture configuration parameter values in the configuration file of the ρ-VEX platform and identifying the optimal design for the execution of the given benchmarks. We also detail the reasons which led us to choose the respective options.

The report is divided into sections where section II gives a description of the new Benchmarks (FIR and Bcnt). In section III, we provide a basic analysis of the design and the assumptions we take for finding the required design points. Section IV describes our design space exploration process for the three designs and documents the constraints for the design process. We illustrate our observations and justifications with the help of tables and graphs. Section V documents our conclusions on the whole Design space exploration process and the learning from this assignment.

II. BENCHMARK DESCRIPTION

We were assigned the following benchmarks namely - Matrix, Pocsag, Fir and Bcnt. Matrix and Pocsag were also our benchmarks for the first assignment and hence we only detail the high-level observation and the possible bottlenecks of the Fir and Bcnt benchmark. After going through the respective C codes for both applications, we made the following observations-

A. FIR

- This Benchmark computes the Finite impulse response of a particular signal. After looking at the C code we observed that there are many looping statements and a lot of numeric calculations. We assume that it would use a high number of ALU units. There are a lot of function calls in the code and a lot of compare loops which we think will increase the branch operations in the assembly code. A lot of instructions will be dependent on each other and hence there won't be much instruction level parallelism. However, by looking at the code we assume it will take a lot of cycles to execute as there will be a high number of ALU operations which cannot be executed in parallel.

B. Bcnt

- This benchmark is used to calculate the number of bits in a particular program and in order to achieve that it uses an array named poptab which contains 256 1-byte elements and an src array which contains 1024 4-byte elements. After going through the C code it can easily be seen that this benchmark uses a large number of additions and bitwise shift operations which can prove to be a bottleneck for this benchmark. The main loop iterates through all elements of src, and for every element loops through each byte of the element. The value located in poptab at the location denoted by the byte is then added to a total t.

III. ANALYSIS OF THE SYSTEM

A. Benchmark Analysis

With a similar approach from the first assignment, we went through C code and ran all the benchmarks with respect to a default configuration (IW=4, MUL=2, ALU=4) to get an idea of their typical execution cycles and size relative to each other. After going through the assembly code of the four benchmarks, we drew the following inferences from the VEX Simulation-

- The matrix (734237 execution cycles) and the Fir (568948 execution cycles) benchmarks are rather large in size when compared to the other benchmarks Pocsag (22567 execution cycles) and Bcnt (4110 execution cycles) as also shown in Figure 1. Hence, we would like to use a larger Issue Width on Matrix and Fir benchmark to take benefit of the Instruction level parallelism.

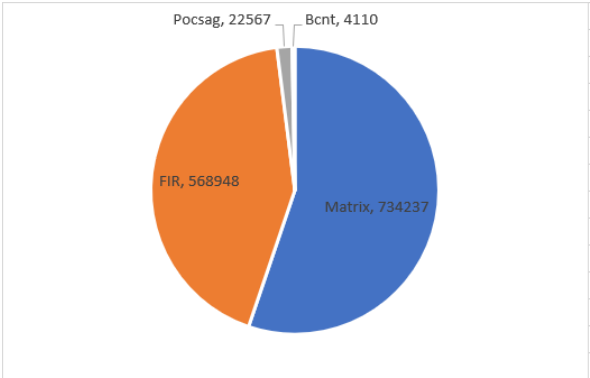

Fig. 1. Number of Execution cycles of all benchmarks

- Matrix code has a total of 16 multiplication instructions and hence will need a large number of multipliers
- Benchmarks Pocsag and Bcnt don't have any multiplication commands and hence don't need any multipliers in their design

- FIR benchmark has only two multiplication instructions but the code is quite large in size. We can also use a larger Issue width on this benchmark to improve its performance
- Matrix benchmark is the largest in size when compared to the other benchmarks hence we propose that a Multi-core design with Matrix benchmark on one core and the other three on other cores should be a better design than a single core design for a balanced or high-performance requirement
- As there is a large difference between the size of the different benchmarks(Matrix>Fir>Pocsag>Bcnt), the scope of using run-time adaptable cores is high as we can judiciously allocate resources to the larger benchmark when the smaller benchmark completes its execution

B. CALCULATION OF AREA OF THE SYSTEM

In this section, we provide an overview of the area model used for the comparison of resource utilization of the explored hardware configurations. The components considered are the number of slice registers, slice LUTs, RAM blocks (RAMB36E1 and RAMB18E1) and the DSP element (DSP48E1). Note that all the components are in reference to the component A_{1CLB}

$$AREA = A_{1S}*N_1+A_{1RAM1}*R_1+A_{1RAM2}*R2+A_{1DSP}*D1$$

where:

A_{1S}	= 0.5 * A_{1CLB}
N_1	= number of occupied slices
A_{1RAM1}	= 2.4 * A_{1CLB}
R_1	= number of RAMB36E1 components
A_{1RAM2}	= 1.2 * A_{1CLB}
R_2	= number of RAMB18E1 components
A_{1DSP}	= 0.7 * A_{1CLB}
D_1	= number of DSP48E1 components

IV. DESIGN SPACE EXPLORATION

In this assignment, we are supposed to design three configurations which correspond to an Embedded Low Power design, High-performance design and a design which compromises between Energy, area and the overall performance of the system. In the "configuration.rvex" file, the parameters which we can vary are -

- The maximum number of cores is 4 1,2,3,4
- The only possible values for issue width is 2,4,8
- The stop bit can only have 3 values 2,4,8 and must be less than the issue width
- The number of ALU units is equal to the issue width, hence there range is 2,4,8
- The minimum number of multipliers is 1 and the maximum is equal to the issue width [1, IW]
- The Instruction cache must be a power of two and should not be greater than 64k [8k,16k,32k,64k]
- The Data cache must also be a power of two and should not be greater than 32k [4k,8k,16k,32k]

memory units[0,1]

A. EMBEDDED LOW POWER

For this design, we were supposed to minimize the energy consumption of our system. We choose a suitable initial design(Config 1 in Table I) considering the following points-

- The usage of a single core was preferred over multiple cores as multiple static cores could lead to a lot of static power leakage and hence consume more power
- We used the Issue width as 2 as it's the minimum configuration and we assume that executing 2 commands in a syllable will take less power than executing 4 or 8 commands in a syllable
- The only possible value of Stop bit that we could use is 2 so we used Stop bit as 2
- We kept the cache at a low configuration, Instruction cache = 16k and data cache = 4k

After running the above-mentioned configuration we got the following results after examining the "boardserver log" file and-

- The total cycle count was 4356115
- the Total energy consumed was 1.47mJ which is very less as compared to the size of our applications.

After Config 1, we decided to model a design with less instruction and data cache so that the design might save power as it would not have to search a large area. We reported the result in Table I. Also evident from Fig 2, the energy consumption increased as the number of data and instruction misses increased by almost 55% from the previous configuration(Config 1).

TABLE I
EMBEDDED LOW POWER TEST CONFIGURATIONS

Configuration	1	2	3	4	5
Physical Cores	1	1	1	1	1
Issue Width	2	2	2	4	4
Config	11	11	01	0111	0111
Stopbit	2	2	2	2	4
Icache	16k	8k	16k	16k	16k
Dcache	4k	4k	4k	4k	4k
Area	3853	3347	3671	4013	3969
Execution cycles	4356115	4569235	4382835	4109715	3971417
Energy Consumed	1.47	5.07	2.52	2.44	1.91

As also evident from Table I we considered two configurations with issue width as 4 to see the effect of Stop bit on the total energy consumption and area of the system. Config 4 consumes more power (2.44 mJ) as the core is ideal and consumes static power while in config 5 the same slots are taken up by NOPs which consume less power (1.91 mJ). This configuration also has less area (3969) than config 4 as the code is more efficiently packed.

We also examined a single core configuration in which we increased the cache(IC=64k, DC=16k) in order to reduce the number of cache misses and save some energy of the system which it would waste by accessing the main memory but to

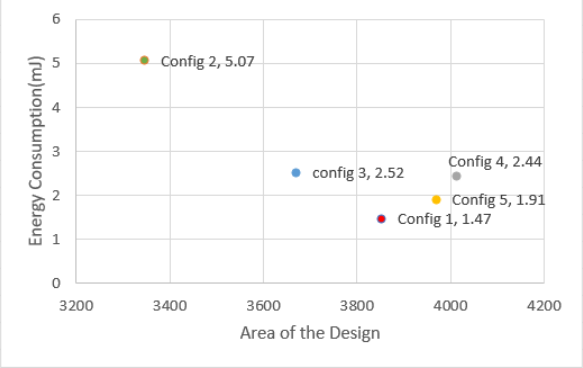

Fig. 2. Energy consumption vs Area for the low power configuration

our surprise the energy shot up to 20.2 mJ as the system had to search a much larger area for finding the correct data and instructions. We haven't mentioned this configuration in Figure I and Table I in order to provide better visualization of the other data points.

Hence out of all the configuration we considered for these benchmarks we would select Config 1 as shown in Figure 2 (in red color) as our favoured design as it consumes the least power which was the aim of this design exploration.

B. High Performance Design

For this design, we were supposed to minimize the total execution cycles of our system. Since we had very contrasting benchmark in terms of execution cycles we tried many configurations trying to optimize every single application and finally presenting some designs in Table II. We preferred a Multi-core architecture over a single core architecture simply because there would be little to no dependence on the completion of one benchmark for the other benchmark to start executing. Also in a single core architecture, the execution cycles of every benchmark will add sequentially as they will execute one after other and not independently in separate cores. We also favored a large issue width (especially for Matrix benchmark) and high values of Data and instruction cache as we suppose that would lead us to the most optimal point. We chose Config 6 as a starting reference point as shown in II. For Config 7, we choose the issue width to be maximum(8) as we wanted to decrease the number of execution cycles of the matrix as much as possible as it is the biggest benchmark and a possible source of a bottleneck. The total cycle count was 1954895. We then also used a Multi core architecture with issue width 8 and four cores as shown in Table II. We increased the instruction cache and data cache memory of cores running Matrix and FIR to 64k and 16k respectively. We used the same number of multipliers and issue width for matrix benchmark as with Config 6 as the performance of our application mainly depended on the Matrix benchmark. We optimized the other cores to better use the resources and reduce design complexity. The configuration accounted 1375234 number of execution cycles a speedup of 29%. This was due to the decrease in data and instruction misses from Config 6 by 18%.

TABLE II
HIGH PERFORMANCE TEST CONFIGURATIONS

Configuration	6	7	8	9
Physical Cores	4	4	2	3
Issue Width	8,8,8,8	8,2,4,2	8,4	8,8,4
Config 1	01111111	0111111	01010101	11111111
Config 2	01100000	01	0101	01100000
Config 3	01000000	0111		0100
Config 4	01000000	01		
Stopbit	2,2,2,2	2,2,2,2	2,2	2,2,2
Icache 1	32k	64k	64k	64k
Icache 2	32k	32K	64k	64k
Icache 3	32k	64k		32k
Icache 4	32k	32k		
Dcache 1	8k	16k	16k	16k
Dcache 2	8k	8k	16k	16k
Dcache 3	8k	16k		16k
Dcache 4	8k	8k		
Area	17465	19039	14414	18549
Execution cycles	1954895	1375234	1360825	1367873
Energy Consumed	20.2	19.54	9.34	19.6

In config 8 we still have a dual-core design with same cache memory for both benchmarks. Matrix was run on core 1 while Pocsag, FIR, and Bcnt were run on core 2. This designed isolated the biggest benchmark and assigned large resources to optimize it. The total reported cycles were 1360825 with the matrix benchmark having execution cycles in order of 1350k while the other benchmarks had execution cycles in order of 873k which is a significant improvement. We also tested a 3 core design (Config 9) with Matrix on core 1, FIR on core 2 and the other two benchmarks on core 3. We used different issue width to better use the resources The value were chosen according to the size of the respective benchmarks. We got execution cycles as 1367873 execution cycles (a degradation of 0.5% from Config 8) with an increase in the area of 29% and energy consumption by 110%. Also, there was no significant improvement in Cache misses for Config 9.

We assume the results are such as we cannot optimize the benchmarks further and increasing the resources will only increase the design complexity and sometimes degrading the performance. We also thought of having a quad-core design with maximum memory for every core. Sadly the design was complex to be synthesized by our systems and hence we did not pursue it further. We believe that even if we would have been able to synthesize the design we would not have achieved a significant improvement in overall performance combined data and instruction misses were around 2500 instruction cycles. Hence for all the above-mentioned points we choose Config 8 as our high-performance design as it isolates the bottleneck benchmark and optimizes it.

We chose Config 8 as an ideal point for our benchmarks as this fulfills the criteria of being the fastest among the configurations that we tested. It uses a dual-core architecture as we expected and performs particularly well as it isolated the largest benchmark on one core while the other three benchmarks of the application run on a different core.

C. Balanced Design

After designing an architecture for an embedded low power application and a different design for a high-performance application, we had to design a balanced architecture which compromises between Area, energy consumption and performance of the system. We began our design space exploration by starting with a dual-core architecture as this should perform better than a low power design and will have a smaller area than a high-performance design. We considered Config 10 from Table III as our initial design in which we mapped Matrix and Pocsag benchmark on Core 1 and on Core 2 - Fir and Bcnt benchmarks. We did this as to balance the overall design by keeping a big benchmark with a small one. We recorded 3266136 execution cycles for this config.

TABLE III
BALANCED DESIGN TEST CONFIGURATIONS

Configuration	10	11	12	13
Physical Cores	2	2	2	3
Issue Width	4,2	8,4	4,4	4,2,2
Config 1	0111	01111000	0111	0111
Config 2	01	0100	0100	0100
Config 3				01
Config 4				
Stopbit	2,2	4,4	4,4	4,2,2
Icache 1	32k	32k	32k	32k
Icache 2	32k	32k	32K	16k
Icache 3				16k
Icache 4				
Dcache 1	8k	8k	16k	16k
Dcache 2	8k	8k	8k	4k
Dcache 3				4k
Dcache 4				
Area	9415	9552	10332	11677
Execution cycles	3266136	3132860	1595795	1633450
Energy Consumed	7.67	7.02	8.07	6.25

Next in order to increase our performance we considered Config 11 from Table III where we mapped the matrix benchmark on core 1 and on core 2- the other three benchmarks as the matrix benchmark was taking more execution cycles (3233654) than the other three benchmarks combined (around 3000k). We kept the cache sizes same but increased the issue width to better suit the benchmarks. We also increased the Stop bit to 4 for both cores. We accounted the execution cycles as 3132860 (speedup of 4% from config 10) with the area as 9552 (an increase of 1.45% from config 10) and the energy consumption as 7.02 (a decrease of 8.5%from config 10). Config 11 did manage to provide a speedup but the data misses were quite high. Matrix benchmark (around 50%) for this point to consider as a well-balanced design. For the next configuration we chose to increase the data cache of the matrix core to improve the performance as shown in Config 12 of Table III. We achieved a speedup of 49% with an area increase of 8% and energy consumption of 15%. The performance increase can be attributed to very fewer data read misses to 32691 from 290174 from Config 11. The area and energy consumption didn't rise to the same magnitude as the performance. As shown in Config 13 in table III a three core design was explored and in order to keep the area in check cache memory was reduced. We recorded a decrease in performance by 2.35% and increase in area by 13%. According to our application, we would choose config 12 as the optimal configuration for a balanced design as it properly manages all the benchmarks according to there needs (issue width and memory demands).

For a single optimal design we would consider Config 12 as the preferred design for the four benchmark applications. Config 12 achieves a decrease of execution cycles from 4356115 to 1595795 from Config 1(low power) while decreasing the area by 28% from Config 7(High performance). This design also uses the resources judiciously while resulting in a good performance for the whole application. It isolates the biggest benchmark(Matrix) and uses the right amount amount of cache without increasing the area proportionally. This design also also allocates the expensive data cache judiciously to the smaller benchmarks Pocsag and Bcnt hence making this design more optimal. This design also uses a high Stop bit to efficiently pack the code size and reduce area. For this application, the matrix benchmark was the most vital benchmark and Config 12 manages it and the other benchmarks very efficiently. This dual core architecture makes sure that both cores complete there execution almost at the same time hence also minimizing static power loss.

V. CONCLUSIONS

In this assignment, we learned how to efficiently model a SOC design for three scenario's - Embedded Low Power, High performance and a balanced design which compromises between Area, energy consumption and the performance of the system. The four benchmarks from the Powerstone suite that we worked with are Matrix, Pocsag, FIR and Bcnt. After running the design on an actual FPGA board we were able to understand how the workload was being distributed between the four benchmarks and how to make educated guesses for testing optimal configurations. We learned about the effect of issue width, Stop bit and the cache memory on the four benchmarks. Tinkering with these variables broadened our understanding of the ρ-VEX core and how we can optimize it for a particular application.

We also learned the value of resources as there were instances when we wanted to increase the cache to the maximum to minimize Instruction misses and data misses but that just made our design very complex and we were not able to successfully implement it (and sometimes we accounted a degradation in performance). We also learned how to efficiently allocate resources to all of the benchmarks and also explored many Multi-core configurations to figure out when to use a single core architecture and when it's wise to use a Multi-core architecture. In our case, Matrix was the largest and the most resource hungry benchmark, hence we tried to isolate it to its own private core while keeping the benchmarks to other cores to optimize the whole application for the high-performance design. We learned to use to the stop bit to more efficiently pack our code and also in general to decrease the area of the design.

We learned that a configuration with minimal components does not always lead to a better design for a low power design (Config 2 from Table I). The design process is very subjective and varies on the benchmarks dependence on memory. For a low power design, we tried to keep the cores, caches and Issue width to the minimum. This did lead to degradation of performance but the FPGA spent very less energy to execute the application which was our final aim. For the high-performance design we kept the matrix benchmark on a separate core than the other benchmarks and set his issue width and resources high as a high proportion of the performance of the application was dependent on the Matrix benchmark. We also used a three core architecture, keeping the two smaller application on one core and the other two benchmarks on separate cores. We also learned how hard it is to obtain an optimal configuration as there are factors like the surface temperature of the die affecting the energy consumption of the design. Finally with the help of tables, graphs and our understanding of Computer architecture we presented our design space exploration process for the three design constraints in this report.

REFERENCES

[1] ρ-VEX project. The Dynamically Reconfigurable VLIW Processor. http://rvex.ewi.tudelft.nl/

[2] ρ-VEX user manual. Jeroen van Straten, TU Delft, 2017.