Modern Computer Architectures – Lab Assignment 2

Group 18

Students: Saevar Steinn, Misael Hernandez, Tudor Voicu

Introduction

Assignment 2 consisted on the design space exploration of the programs from Assignment 1 plus two other ones, now on an actual SoC design with a reconfigurable ρ-VEX core. The exploration was done on the 2 assigned programs, pocsag and v42, and two randomly selected ones, crc and blit. Based on the results obtained in Assignment 1, and further analysis of the selected benchmaks, a configuration was selected on the basis of balancing the needs of the 4 programs. Using the given tools, an analysis on performance, and energy consumption was done. Based on that analysis, two other configurations were defined, one with the purpose of improving performance, and a second one with the purpose of reducing area. The same analysis for those 2 configurations was also done, and a conclusion is presented where the all the results are taken in consideration for a final proposal..

Blit is a graphics algorithm for combining two bitmaps. It basically consists on taking a small bitmap 1 that will be introduced into a bigger bitmap 2. The bitmap 1 has a mask that defines which pixels will be taken and pasted into the larger image. With an AND operation of the mask and bitmap 2, the original image is erased on the parts where the bitmap 1 will be introduced. After that, an OR operation is used to "paste" bitmap 1 into bitmap 2. The blit benchmark consists only on this last operation. It will take an array of 32 bit data, and it will copy it into a different address with an offset. If the offset is non-zero, special consideration is taken to leave the data that was already there at the destination address and is not supposed to be overriden. For example if the destination offset is 20 bits, the 20 MSB of the first 32 bits will be left intact, and the amount of bits specified to be copied, will be copied into the destination address starting from the 12th position. All the processing is done in batches of 32, that is why the OR operation is very important as to not modify the data that was already copied. Most of the processing that is done consists of shifts and the OR operation, along with a branch to keep looping until the number of bits to be copied is reached. Data is read and written in batches of 32 bits.

The ***CRC*** (cyclic redundancy check) is a very simple and effective error detection algorithm, used in data transmission and storage devices to detect damaged raw data. CRC is mainly used because of its ease of implementation in binary hardware and its high effectiveness on detecting common errors caused by noise on data transmission channels. This algorithm performs a specific operation on an input block of data, generating a fixed-size value, similar to a hash. The output value is then compared to the attached key (a value attached to the original message), for validating the data. The main processing effort is done iteratively with data dependencies (variables in loops are dependent on their previous versions) and the operations performed are mainly logic (shifts and XOR).

Proposals

Initial configuration

IssueWidth 2 and 4K sets in Instruction cache

For the first proposal, we wanted one that kind of balance the requirements of all our benchmarks. For the benchmarks from Lab 1, the best configuration consisted on issue width 4 with around 4 ALUs and most of the other parameters as 1. However, for the new benchmarks described in the introduction, the best configuration was issue width 2, all of them single core. So our initial proposal was a minimalist design using a single core with an issue-width 2, 1 ALU and one multiplier per issue and 4096 depth instruction cache. With this configuration we expected to have a good performance with few hardware, on which we could build up to a better performance.

On this configuration, running the workload on the FPGA took 4459966 total clock ticks which translates to 59.47 ms at 75 Mhz with a power consumption of 1.99 Watt. As v42 was the heaviest benchmark in the workload the next design step was to aim at the final solution from assignment 1 where the ideal configuration had issue-width 4 and 4 ALUs for the v42 benchmark.

Slice Logic Utilization	Used	Utilization
Number of Slice Registers	9752	3%
Number of Slice LUTs	13083	8%
Number of RAMB36E1/FIFO36E1s	0	0%
Number of RAMB18E1/FIFO18E1s	140	16%
Number of DSP48E1s	4	1%
Power		1.990234 Watt
Cycles		4459966

Table 1: Results for initial configuration

Improving Performance

IssueWidth 4 and 4K sets in Instruction cache

Of the 4 applications that we ran on the processor, the longest one by far was v42. Considering this, for the first improvement we started from the best configuration we had obtained in Lab 1 for v42, with an issue width of 4 and an ALU per issue. Because of the sequential characteristic of our applications, the improvement trend flattens at this point, showing no benefit in further increasing the hardware complexity.

For demonstrating these facts, a new design was made with an issue-width 4, four ALUs and again instruction cache depth of 4096. This increased the overall performance by 3 % or 1.8 ms with a total time of 57.6 ms and a 6% increase in power consumption or 2.11 Watt compared with the 1.99 Watt in the initial design. Because the only application that can benefit from an increased cache size is the blit benchmark, the bigger instruction cache is not feasible for our configuration.

Slice Logic Utilization	Used	Utilization
Number of Slice Registers	10347	3%
Number of Slice LUTs	18288	12%
Number of RAMB36E1/FIFO36E1s	0	0%
Number of RAMB18E1/FIFO18E1s	268	32%
Number of DSP48E1s	8	1%
Power		2.109375 Watt
Cycles		4319318

Table 2: Results for performance-optimized configuration

Reducing Area

IssueWidth 4, 512 sets in Instruction cache

A third design was made to attack the power consumption of the second design without losing too much of it's performance. Starting off, neither of our past proposals can be considered heavyweight. The biggest one has an issue width of 4, with one ALU and one multiply unit per issue but only one branch and memory transfer functional units in total. The instruction cache has just 4K blocks while the data cache has 64 sets with 4 ways, meaning 256 blocks. So this looked like a complicated task.

Our first idea was to reduce the number of multipliers. Neither of our applications make use of them, so we planned on leaving only one multiplier, as a mean to still support the full MUL instruction, but still reduce the area considerable. However, this is not supported by the tools yet, which require each issue to have a multiplier functional unit. So another idea would be needed.

After analyzing the code of our benchmarks and simulation in the HP VEX simulator it was obvious that the combined workload was not using much of the cache provided. As reported in Assignment 1, and a fact that is also true for the two extra benchmarks, blit and crc, these programs are not dependant on cache size. Most of the instructions and data are read in a linear fashion, and they contain a lot of small loops, so having a cache is important for improving performance. However, the code and the amount of memory needed for input and output data is very small. Also, caches tend take a lot of die area. So it looked like the perfect opportunity for improvement.

The question was how much could we reduce the cache. For that, we took on the task of trying out several configurations. So after a few experiments in ModelSim we came down to the solution of issue-width 4 but with an I-cache depth of 512. This results in a significant decrease in area usage and power consumption without a high sacrifice in overall performance. The reduction in performance was just 0.33%. However it also reduced power consumption by 6.8%, and area is greatly reduced as seen in the RAMB utilization.

Slice Logic Utilization	Used	Utilization
Number of Slice Registers	10344	3%
Number of Slice LUTs	18289	12%
Number of RAMB36E1/FIFO36E1s	33	7%
Number of RAMB18E1/FIFO18E1s	8	1%
Number of DSP48E1s	8	1%
Power		2.109375 Watt
Cycles		4319318

Table 3: Results for area-and-power-optimized configuration

Deciding on the best solution

Looking at the results, two of our solutions could be considered the best. If we only care about performance, the proposal with issue width 4 and 4K sets in the Instruction cache would be the winner by a considerable amount. On area and power consumption, the one with issue width 4 and 512 sets in the Icache is could arguably have definitely the least power and the least area, considering that memory blocks are much bigger than LUTs, which would be translated to logic gates in a die..

On a closer look the reduction on performance between the best solution on performance and the best on area is just 0.33%, which is not really considerable. However the power consumption is reduced by 6.8%, so in terms of performance per watt, the third solution is the best one.

In comparison to the first configuration, the one with issue width 4 and 4K sets in I-cache (configuration 3) is 3.15% faster, but consumes 6% more power, and the it is bigger all around. For configuration 3 (issue width 4 and 512 sets), the improvement is 2.83% on cycles, but also power decreases by 1.28%. In terms of area, it is worse all around as well, except for the RAMB blocks, which we think offsets everything else. We can see the whole data on Figure 4, noting that the percentages mean improvement, so a positive improvement means less area, less power and less cycles. Overall, we think Configuration 3 is the best one, as it has a great performance, the best performance per watt, the least power consumption, and arguably the least area.

Slice Logic Utilization	Configuration 1	Improvement of Conf 2	Improvement of Conf 3
Number of Slice Registers	9752	-6.10%	-6.07%
Number of Slice LUTs	13083	-39.78%	-39.79%
Number of RAMB36E1/FIFO36E1s	0	0	(+ 33 units)
Number of RAMB18E1/FIFO18E1s	140	-91.43%	94.286%
Number of DSP48E1s	4	-100%	-100%
Power	2.109375 Watt	-6%	1.28%
Cycles	4319318	3.15%	2.83%

Table 4: Improvement of Configurations 2 and 3 in comparison to Configuration 1

Reflections

During this lab, the first thing we learned were the tools used in a workflow for computer architecture and design. Using the cycle accurate simulator, the Modelsim simulation and ISE synthesizer tools we we're able to get experience on a big part of the whole workflow of architecting and designing an SoC. Next, we learned about making architecture decisions based on heterogeneous loads, where we had to balance the requirements of several types of applications. Mixed with that, we also had to balance between improving performance and maintaining a small and power-efficient design. In general, with this lab we had a big amount of contact with the situations and the kind of decisions that a real computer architect goes through everyday. Finally, we also had a chance to look at and meddle a little bit with a VLIW processor that is currently at the heart of the latest research on this kind of processors.