

CVA6 Design Space Exploration on Agilex 7 FPGA

Angela Gonzalez¹, Mustafa Karadayi¹, Franck Jeulin², Christophe Biquard² and Jérôme Quévremont²

¹PlanV, ²Thales

corresponding author: angela.gonzalez@planv.tech

Abstract

CVA6 offers a wide range of configuration parameters that permit to tailor the core to different applications. However, the vast number of existing parameters can be overwhelming, making it difficult to know where to start from, or which are the right choices to make. This work presents the results of design space exploration of CVA6 focusing on FPGA targets, in particular, on the Agilex 7 platform from Altera. Starting from the existing FPGA configuration from OpenHW, we explore two orthogonal directions: (i) maximizing performance and (ii) minimizing resources. We show the results achieved for different configurations, providing insights on the impact of different parameters (e.g. memory architecture, extensions, etc.) Among a variety of combinations, we find a sweet-spot that permits to achieve 30% performance improvement together with 50% reduction of registers, compared to the existing FPGA configuration which is primarily optimized for Xilinx. With this example and other exploratory results, this work aims at simplifying the initial choices in the configuration of new designs based on CVA6.

Introduction

CVA6¹ is a popular open source RISC-V CPU from OpenHW Foundation. It is highly configurable, allowing for designing a wide range of cores: from solutions for applications with Linux support to embedded processors running bare-metal applications. While this offers great flexibility, it brings also the challenge to choose the best configuration for a new design. Besides that, CVA6 was initially designed for ASIC targets, but the interest to have a vendor-independent FPGA version emerged, resulting in the FPGA configuration maintained in CVA6 repository. In this work, we explore the capabilities of CVA6 for an FPGA target (Agilex 7 from Altera²). We start from the existing FPGA configuration, and explore what is possible both in terms of performance and reduction of resources, focusing on embedded configurations.

Porting & Optimizing CVA6 on Altera FPGA

The FPGA configuration defined in previous work, optimizes CVA6 for FPGAs [1]. The optimizations in that work included technology agnostic and technology specific ones (targeting Xilinx). The latter, can be enabled with a configuration flag (FpgaEn). However, they can't be reused in our case because the FPGA fabric does not support the same primitives. The first step in this study was to port the technology specific optimizations to Altera technology. The second one, to create a design equivalent to the Xilinx Application Processing Unit (APU) for Agilex 7 platform. Both items have been contributed to CVA6 repository and are available to the community.

Design Space Exploration

We start with the FPGA configuration as is (Config. A in Table 1), getting the performance (Table 2) and resources usage (Table 3) of CVA6. We will use these results as a reference to benchmark other configurations.

¹ <https://github.com/openhwgroup/cva6>

² <https://www.intel.com/content/www/us/en/docs/programmable/683024/current/overview.html>

Table 1: Configurations Explored

A	OpenHW FPGA config. w/o Xilinx optimizations
B	Altera FPGA optimizations enabled
C	No MMU
D	B + C + best cache performance*
E	D + no privilege levels
F	E + only C extension (remove Zcb, A, B, RVZicnd)
G	F + store and commit buffer depth 2
H	G + reduced cache**
I	H + 2 scoreboard entries
J	I + one single load buffer entry
K	J + SRAM instead of DDR
L	H + SRAM instead of DDR

*best cache performance: 16 KB cache, 512 bits cache line and 4 ways. Same for Data and Instruction caches.

**reduced cache: cache line of 64 (width of AXI bus), with only 1 way. 8KB instruction cache, 4kB data cache.

First of all, we evaluate the impact of the FPGA optimizations that have been ported to Altera technology (Config. B). The results show that the FPGA optimizations permit to reduce the number of Flip-Flops (Ffs) by 34%, and the number of Look Up Tables (LUTs) by 7 %, in exchange for a 33 % of extra Block RAM (BRAM). This was expected since the optimizations focus on moving registers to memory resources, to improve the routing in the FPGA. The overall performance (Coremark) improves slightly, because the maximum frequency achieved on the FPGA is a bit higher (thanks to the better routing in the FPGA when moving registers to memory).

Secondly, we evaluate the impact of the Memory Management Unit (MMU). Overall, removing the MMU reduces resources between 9 and 16%. Since we are focusing on embedded configurations, we are not interested in configurations that offer Linux support, so we continue this exploration without MMU.

Thirdly, we try to push the performance. A typical pain point is the memory access, so we evaluate different cache configurations.

Table 2: HW resources required by each configuration

	LUTs		FFs		BRAM	
A	12530		8959		24	
B	11621	-7%	5874	-34%	32	+33%
C	11411	-9%	7908	-12%	20	-16%
D	15773	+26%	7521	-16%	135	+462%
E	15591	+24%	7409	-17%	135	+462%
F	13738	+10%	7036	-21%	135	+462%
G	13823	+10%	6650	-26%	135	+462%
H+L	7710	-38%	4422	-50%	16	-33%
I	7150	-43%	3874	-56%	16	-33%
J+K	7273	-42%	3837	-57%	16	-33%

The best result was achieved with the cache described in Config. D, improving performance by 25%. The cache line is set according to the throughput of the DDR memory used in this design. However, we see that the use of BRAM has exploded with respect to the original FPGA optimizations. This is because the use of a big cache line infers an inefficient use of the BRAM blocks available in the Agilinx 7 FPGA. We also see an increase in LUTs and FFs compared to the FPGA optimized version of OpenHW configuration (Config. B), related to the bigger cache. As next step we focused on reducing resources without sacrificing performance and find two ways to do it: eliminate privilege levels (Config. E) and extensions (Config. F). After this, changes impact the performance.

The distribution of FFs in Config. F is shown in Fig. 1. As expected, the cache is taking a lot of resources. The next biggest impact is the Load Store Unit (LSU), followed by the scoreboard (SCB). We try first with the LSU to see if we can get some reduction without dropping too much the performance. Reducing the depth of the store and commit buffer (Config. G) lowers the Coremark/MHz to 1.9 and reduces about 400 FFs compared to Config. F.

At this point we decide to prioritize resources reduction, to see where it is possible to get in spite of performance. We reduce the cache (Config. H), the number of scoreboard entries (Config. I) and the number of entries in the load buffer (Config. J). Now, the resources have decreased considerably (up to 57% in FFs), but the performance is also lower (-17%). The new distribution is shown in Fig. 2. Since we previously identified the memory accesses as a pain point, we decide to explore other memory architectures. Until here, the APU design is using a DDR as main memory. In the next experiments, we move from using the DDR to using an internal SRAM.

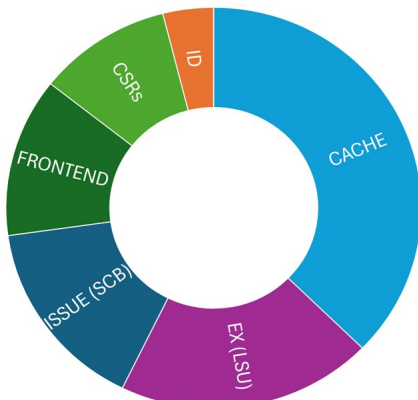


Figure 1: FFs distribution in Config. F

Table 3: Performance obtained for each configuration

	Coremark/MHz	Fmax (MHz)	Coremark Total	
A	2	200	400	
B	2	215	430	+7%
C	2	215	430	+7%
D	2,3	215	498	+25%
E	2,3	217	499	+25%
F	2,3	218	501	+25%
G	2,25	220	495	+24%
H	1,9	220	418	+4%
I	1,5	220	330	-17%
J	1,5	220	330	-17%
K	1,8	210	378	-5%
L	2,5	210	525	+31%

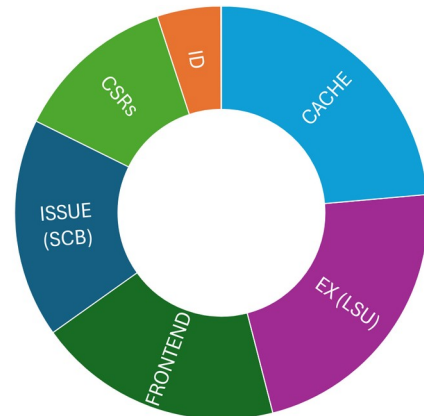


Figure 2: FFs distribution in Config. J

The goal of this change is to evaluate the performance of CVA6 itself, decoupling it from a specific memory architecture. We show the results for the configuration with lowest resources achieved (K) and for the smallest one that does not incur in performance losses (L). We see that, by eliminating the DDR path latency, the performance recovers to values close to the best ones (or even better). The overall results show that it is possible to keep almost the same performance (-5%) with a reduction of 57% in FFs, or to get higher performance (+31%) with a reduction of 50% in FFs.

Conclusion & Future Work

Overall, we have showcased the flexibility of CVA6 and provided results with different example configurations. These results can be used by the community as reference to make informed choices in future FPGA designs. Future work could follow [2], where performance on ASIC achieved a Coremark/MHz of 3.09 with another memory architecture, and estimated an increase to 4.5 in a dual issue version, which could also be optimized for FPGA targets.

References

- [1] Sébastien Jacq et al. Recent achievements of the Open-Source CVA6 Core. RISC-V summit Europe 2023
- [2] Côme Allart et al. Performance Modeling of CVA6 with Cycle-Based Simulation. RISC-V summit Europe 2023