Power Optimization of the CV32A6 Softcore architecture for CNN aplications

Clément DEMAZEAU[1], David HURTADO[2] and Gerardo NOSTHAS[3]

[1] clement.demazeau@imt-atlantique.net
[2] david-ricardo.hurtado-barreto@imt-atlantique.net
[3] victor.nosthas-vides@imt-atlantique.net

Abstract—This report presents our work corresponding to the A3 project which is delimited within the Thales competition "Power optimisation of the cv32A6a soft-core". To analyse the work done we will begin by giving definitions regarding the contest terminology, discussing about the context and explaining the followed workflow for setting up the environment and replicating the Thales base results. In the second part several optimisation techniques are discussed, and their implementation, as well as the results,are evaluated.

Keywords—RISC-V, CVA6, FPGA, Open Source, Power Optimizations

Introduction

The objective of the project is the reduction of energy consumption of a synthesizable core architecture when running a neural network application.

Nowadays energy consumption is a major issue as risc based processors are used on embedded systems that keep on growing as ioT devices multiply, or are being used on applications that require higher computational power such as artificial intelligence or supercomputers. This is reflected on companies such as ARM which have "low power design"policies for the design of their cores.

For the contest, certain constrains were imposed and they are listed as in the contest announcement:

- The performance of the application must not be changed.
- The FPGA size must not increase more than 10 % (as well as each of the resources types: DSPs, LUTs, BRAMs).
- The CNN application source cannot be changed.
- The pipeline depth must remain the same.
- The features from the core cannot be removed.
- The CNN application can run on the modified processor embedded on the board.
- Only Vivado and Questa (as CAD tools) can be used.

Also, as a reference, the energy consumed will be calculated by:

$$total\_energy = (total\_power)(number\_of\_cycles)(clock\_period) \quad (1)$$

Where the total power will be reported by vivado as it is seen on the section Workspace Installation (see 5)

Definitions and Context

The RISC-V ISA

The RISC-V ISA is a royalty-free risc based instruction set architecture which different companies all around are increasingly using for its simplicity and flexibility. Additionally, due to its open-source nature it presents economic and technological advantages, as well as potentially more transparency and opportunities for collaboration.

Table 1: RISC-V Extensions and their functions

Extension Name	Function
A	Atomic Instructions
C	Compressed Instructions
D	Double-Precision Floating Point
F	Single-Precision Floating Point
M	Integer Multiplication/Division
G	Application Class (A+D+F+M)

The softcore

The cv32A6a core is a 32 bit modified version of the 64 bit “Ariane” softcore which was initially designed to target applications on ASICs 1. This softcore contains a 6 stage pipeline as well as the support for M and C instructions (see 1). Then, with the modified version and as discussed earlier, Thales contest intends to optimize its architecture to run a neuron network application targeting fpgas.

The application

Regarding the application to run, it is a neuron network with two convolutional and two fully connected layers that

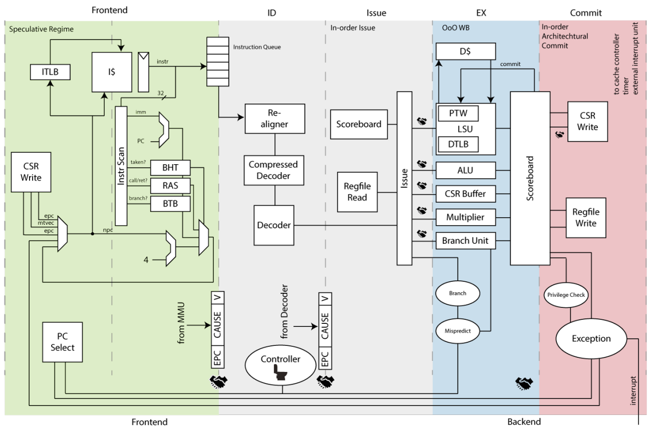


Fig. 1: CVA6 architecture overview [1]

allows to identify numbers from the well known mnist database.

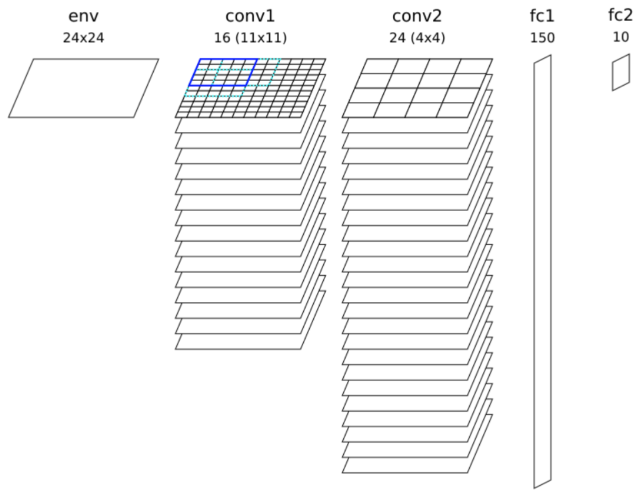


Fig. 2: CNN architecture [2]

Workspace Installation

To set up the environment we were given a GitHub repository with the different steps for the installation and simulation. The workflow for the installation can be observed on figure 3. The first step was to install the tool chain. This contains a group of optimized tools for the specific programming. In this case, GCC, Binutils newlib or gilbc ports. Next, the installation of the specific versions of Questa, Vivado and OCD was done by the school. The latter allow the on-chip programming and supports the JTAG interface.

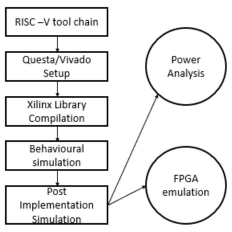
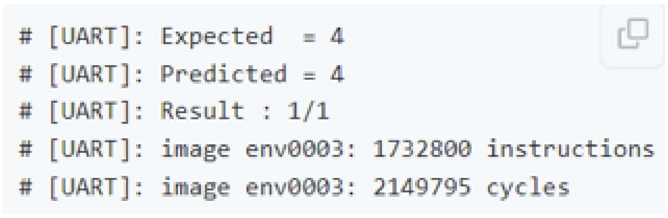


Fig. 3: Workflow Installation of working environment

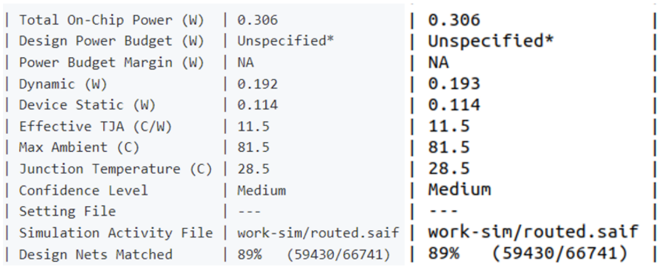
The following step was the Xilinx library Compilation and the Behavioural simulation, which was a first verification to ensure all the installations were done correctly. When this was executed, the Questa simulator appeared as we had to run the given testbench during about 100ms of the application to obtain on the terminal the result shown in figure 4. This tests a 4 is being inputted into the neural network and is indeed recognised as a 4. Additionally, we can observe the instruction and cycle number, which had a slight variation with respect to the ones shown on the provided repository.



```
# [UART]: Expected  = 4
# [UART]: Predicted = 4
# [UART]: Result  : 1/1
# [UART]: image env0003: 1732800 instructions
# [UART]: image env0003: 2149795 cycles
```

Fig. 4: Terminal display after behavioural simulation

Then we ran the post implementation simulation, which means all of the routing on the FPGA is done and therefore the energy consumption information can be extracted. Once again, the Questa tool runs automatically, and a similar image is expected on the terminal to ensure it passes the test. A text file is generated and stored to observe the total consumed power, and more detailed information regarding power consumption and the architecture, such as the number of FF, LUTs or DSPs and their consumption, or the consumption from each block (e.g. The cache subsystem, the frontend, etc.). Figure 5 shows the comparison of the obtained and expected results for the base implementation. It is evident that the results are the same except for a slight discrepancy on the dynamic power.



```
| Total On-Chip Power (W)  | 0.306                   | 0.306                   |
| Design Power Budget (W)  | Unspecified*            | Unspecified*            |
| Power Budget Margin (W)  | NA                      | NA                      |
| Dynamic (W)              | 0.192                   | 0.193                   |
| Device Static (W)        | 0.114                   | 0.114                   |
| Effective TJA (C/W)      | 11.5                    | 11.5                    |
| Max Ambient (C)          | 81.5                    | 81.5                    |
| Junction Temperature (C) | 28.5                    | 28.5                    |
| Confidence Level         | Medium                  | Medium                  |
| Setting File             | ---                     | ---                     |
| Simulation Activity File | work-sim/routed.saif    | work-sim/routed.saif    |
| Design Nets Matched      | 89%   (59430/66741)     | 89%   (59430/66741)     |
```

Fig. 5: Obtained Energetic base consumption vs reference consumption results

The final step was running the app on the FPGA. To do this the zybo board was connected as shown on figure 6. And the bitstream was loaded on to it to obtain a similar display as shown on previous simulations of the terminal window (see fig 4).

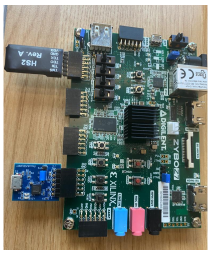


Fig. 6: Zybo board used connected to jtag and USBAUART

Techniques studied for power optimization

Data bus encoding

Given that we're studying a general purpose 32-bit processor, it's safe to assume there's a multitude of data buses connecting the different components of the micro-architecture layer. This is confirmed when looking at the elaborated design on Vivado: all the different blocks effectively communicate 32-bit wide chunks of data every clock cycle. The density of connections begs the question whether pseudo-randomized data transiting these buses could induce energy losses due to excessive switching. As it turns out, this is a well studied subject that has produced new techniques to counter potential losses [3]. To solve this, researchers have created encoding protocols that allow to communicate the same data while reducing switching in the bus. We decided to implement the *inversion encoding* technique for this processor's buses.
This technique consists in first calculating the amount of switched bits on a N-bit bus between clock cycle t and $t+1$ and then inverting the bus's contents if the amount of switched bits is higher than $\frac{N}{2}$. A single "inversion bit" is appended as a MSB, its purpose is to indicate whether the data in the bus is currently inverted. The principle is illustrated in the following diagram presenting an example in 8 bits:

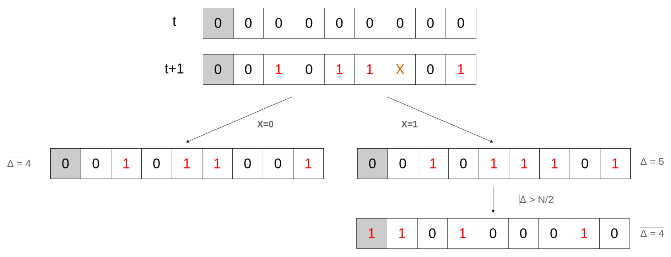


Fig. 7: Description of inversion encoding

The number of switched bits corresponds to the *Hamming Distance* between the bus's contents at t and its contents at $t+1$, and it is calculated via a bit-wise XOR operation between both vectors. This distance corresponds to the value of Δ in the diagram.
In this example, the contents of an 8-bit vector are randomly assigned after a clock cycle, and 4 of the 8 bits are switched. As seen in the diagram, two cases are evaluated here: in one, the X value in the packet is equal to 0, which means the number of swapped bits is equal to $4 = \frac{N}{2}$ thus the bus is not inverted since it's not interesting in this case.
If X = 1 however, 5 bits are swapped, which means a majority of the data bus gets switched in a single clock cycle. This triggers our encoder and the inversion is made, thus leaving 5 out of the 8 data bits as they were in the previous state and switching 3 instead of 5. We can also see the inversion bit is set to 1 to indicate the current data is inverted.
This encoding implies the data should be decoded upon exiting the bus and before entering its target by simply re-inverting the data whenever the inversion bit is set to 1.
In practice we are comparing not 8-bit sized words but rather 32-bit data buses. The principle is the same: we first stock the previous data in a temporary register and compare it to the current data, whenever a majority of the bus is switched, we inverse the signal so the number of swapped bits goes from M to $N-M \leq N-\frac{N}{2}=\frac{N}{2}$. Once the inversion is done, we update the temp register and repeat the comparison at $t+2$
This method presents the drawback of extending the design through the addition of encoder/decoder pairs, which adds a considerable amount of logic gates to our design. We predict that this will mean an increase in static power consumption, but we would hope that taking the stress off the switching FPGA connections would compensate and save a substantial amount of dynamic power.

This method was tested to no avail since the synthesized design of our Encoder component kept consuming large surfaces and consuming lots of power.

Branch prediction

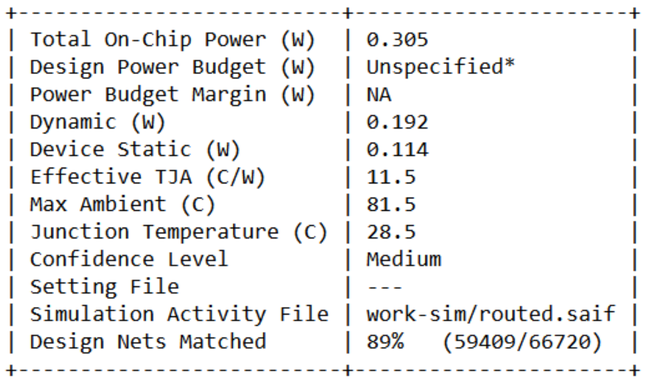
Another area in which we focused our investigations was the system of branch prediction. As we know, branch prediction aims to speed up when executing the instructions on pipelined processors by "guessing"the most probable outcome when encountering conditional situations. In the Ariane architecture inside the PC generation stage, the decision of whether a branch is taken or not is done, as well as the speculation of the branch target address. To perform this a Branch History Table (BTH) and a branch target buffer (BTB) are utilized. The BTB is in charge of decoding PCs to determine if it appears to be a jump and is updated when mis predicting, while the BHT will make the actual decision of taking or not the branch. As it is expected, if the accuracy of the branch prediction, and of the address, are improved, the power consumption can be lowered. This is studied in [4] where several optimization branch prediction architectures are studied as well as the optimal size of BTB to find a balance between power consumption and performance.This paper gave us some insight about the directions to look at in order to attempt the power reduction.
In this order of ideas, as it is also explored in the article, the CNN algorithm used, functions in nested loops. Therefore, we thought of adapting the branch prediction by making a secondary path that is triggered by the conditional opcodes, meaning it will be dedicated to speeding up these conditional jumps. Our solution for this is to profile the algorithm and find the jump addresses, with this information in hand we could create a LUT that holds the probable jump addresses and allows us to access them faster and with less power than it would take to predict the jump address through our traditional branch predictor.
We looked into this option but were unable to profile the code in this way because of time constraints.

Cache memory

Very early during the profiling stage we where able to establish few components that makes up for most of the power consumed by the CV32A6. One of those components is memory and more specifically cache. Because the CVA6 was developed with a general purpose processor in mind there is quite some room for improvement there in order to fit the cache better to bare-metal applications and our neural network implementation. Since cache can be implemented in a plethora of ways and could be the main interest of a whole contest in itself we chose to try and tweak the already existing implementations in order to see if we could make some improvements without having to rewrite how the memory is dealt with. Lucky for us, the CVA6 comes with two different implementations of cache memory, the first one is a Write Back implementation which was the first one to be written for CVA6 the second one is a Write Through which was originally written for the OpenPiton[**OpenPiton**] project which implements many core cache for open sparc but was adapted for the CVA6. Both present a different paradigm, in order to summarize, Write Back stores written values in the cache for a given time before it writes them to the main memory. Write through on the other hand tries to write in the main memory as soon as possible every time a write is issued by the CPU.

After having studied both solutions it is interesting to try and modify cache parameters in order to determine their influence on the power consumption. Settings we can tweak for the data and instruction caches are : The size, the associativity and the cache line width. Generally speaking decreasing the cache size will improve power consumption whilst increasing the cache miss rate and thus increasing the total number of cycles it takes to run our neural network. The goal we aim at is finding the sweet spot where performance and having the cache as small and lightweight as possible.

During our tests we weren't able to gain on power consumption whilst maintaining performance. Two interesting result were that by decreasing the cache size by 256 bytes on the data cache we could gain about $1mW$ of power but also add about 1000 cycles when running the mnist app. Extreme settings could lead to great saves in power : having both data and instruction caches indexed on $6bits$ instead of 12 we could lower down the total consumption to $283mW$ saving up about $34mW$ but this would also imply that the total number of cycles necessary for running the app would increase up to 2867613. We tried to compensate losses that appeared when increasing cache size by modifying other settings but we ran into multiple compatibility issues and bugs, which, because of time constraints forced us to abandon cache optimizations for this contest.



```
+--------------------------+----------------------+
| Total On-Chip Power (W)  | 0.305                |
| Design Power Budget (W)  | Unspecified*         |
| Power Budget Margin (W)  | NA                   |
| Dynamic (W)              | 0.192                |
| Device Static (W)        | 0.114                |
| Effective TJA (C/W)      | 11.5                 |
| Max Ambient (C)          | 81.5                 |
| Junction Temperature (C) | 28.5                 |
| Confidence Level         | Medium               |
| Setting File             | ---                  |
| Simulation Activity File | work-sim/routed.saif |
| Design Nets Matched      | 89%   (59409/66720)  |
+--------------------------+----------------------+
```

Fig. 8: Energy consumption information for modified cache parameters

Ways to improve power consumption that didn't fall under Thales's rules

During the profiling stage we identified quite a few ways to improve the fpga's power consumption unfortunately some of those didn't fall under the the rules of the challenge. Here are a few ideas we explored and eventually didn't implement.

Clock generation

Clock generation is a challenging matter when it comes to fpga particularly in the default configuration of the CV32A6. The needed clock for the softcore as specified by thales is 45Mhz. This clock is generated and dispatched by the MCMM which is the single component that consumes the most power. There exist solutions in order to reduce the impact of clock generation which mostly consists of doing manually the operations of clock division using PLLs, such method could be quite tricky to implement in practice since PLL circuitry needs to be fine tuned in order to achieve the correct clock frequency. Another issue with this method is that 45Mhz isn't a direct divisor of the quartz frequency that drives the FPGA. If we were able to either overclock the processor or decrease the critical path length in order to drive the CV32A6 at 50Mhz [**mrx05**] implementing clock generation using PLLs is estimated to save up to 20 % of the total dynamic power.

Conclusion

To sum things up, our team explored a series of different techniques meant to reduce power consumption, but as a result of the challenge's constraints along with ambitious techniques that are difficult to scale, our research did not come to fruition. We wish we could improve on power consumption but are still glad for the knowledge we acquired through this project

Referencias

[1] OpenHW group. “cva6”. En: (2020).

[2] Thales Group. “cva6-softcore-contest”. En: (2022).

[3] A. Karthik y Siva Sankar Yellampalli. “Comparative analysis of various off-chip bus encoding techniques”. En: *2015 International Conference on Computer Communication and Informatics (ICCCI)* (2015), págs. 1-4.

[4] Mingjian Sun y col. “A Low Power Branch Prediction for Deep Leaning on RISC-V Processor”. En: (2021), págs. 89-92. DOI: 10.1109/CSEPS53726.2021.00025.