Content Addressable Parallel Processors

Authors: Valentin DARMON, Maxime GRAS-CHEVALIER, Kevin ROLANDO and Louis SAUBOLE. \
Based on Camille Leroux's minimalist CPU architecture

The cape branch is organized as in the following :

> The **PROG** folder contains programs for the CPU, the compiler and some helper scripts\
> The **VHDL** folder contains the HDL code of the project\
> The **XDC** folder contains the constraints file required to implement the design on a Nexys 4 board\
> The **DOC** folder contains more documentation about each part of this project

---

The project

This GitHub repository represents the fruits of a 3rd year project by embedded systems students at ENSEIRB-MATMECA.

This project aims to address traditional CPU architecture limitations. The idea is to implement an architecture suggested in the late 70s by Foster Caxton. This architecture, named **Content Addressable Parallel Processors (CAPP)**, sometimes referred as Content Addressable Parallel Engine (CAPE), uses **Content Addressable Memory cells (CAM)** as the basic memory cell and as a fundamental element of a massively parallel co-processor.

This architecture will allow us to perform a single operation on every CAM word simultaneously, thus reducing the time and energy required to perform some specific operations. This also makes compute time dependent of the size of words only, not the number of words to be processed.

Because there is very little literature on the topic, we designed the architecture ourselves and implemented it on a FPGA using VHDL.

Our goal is to design a simple CAPP co-processor as a proof of concept. It will be running alongside a traditionally-architectured CPU.
We managed up to 4.5x speedup when incrementing 96 16-bit words. The following table presents these results in number of cycles to perform one incrementation on 96 words.

	No load/store	**With load/store**
CPU	~600	~4000
CAPP	132	1284

Our architecture

Our architecture is based on a 16-bit simple CPU design working alongside a 96-CAM 16-bit words CAPP.

- **The CPU** handles 11 instructions. It internally uses one instruction register and one accumulator.\
  More detailed information on the CPU can be found in this file : DOC/CPU/CPU.md

- **The CAPP** is made of 3 different blocs : the **Finite State Machine (FSM)**, the **CAM words** and an interface block to command the CAM words individually from the FSM output signals.

  As of now, the only operational instruction that the CAPP can perform is incrementing the CAM words. It can also load a data to a specific word from CPU, and write back to the CPU the data from a specific word.

  - The **FSM** sees CPU instructions as input and output CAM control commands.
    Thus, it can stall the CPU so that the CPU respects the timings of the FSM and CAM cells.
     It is also responsible for translating CPU instruction (write to CAM, read from CAM, increment) into search controls. As such, it implements the bitwise incrementation algorithm. The algorithms for bitwise addition and subtraction of two words are ready and tested but not implemented since it would require compatible CAM words. More information in the PROG/micro_instr/ folder. \
     More detailed information on the FSM can be found in this file : DOC/CAPP/FSM.md
  - The **CAM** stores data and performs bitwise operations on the stored data to perform search and update steps.
    It uses two inputs (mask and comparand) to achieve computation. The stored data is masked using AND operations on each bit and then strict equality is tested between the masked data and the comparand to tag words for future update.
    Updating also uses these inputs, where the mask is used as a way to enable the update of a bit and the comparand contains the values bits are to be updated to.
    The mask and comparand needed for these steps are not stored and are provided by the FSM.\
    More detailed information on the CAM in DOC/CAPP/CAM.md

Results & possible improvements

- Functional co-processor
- Some problems either reading or writing the data to and from the CPU (blocks of 4 identical words)
- Up to 4.5x faster than CPU for specific tasks on 96 words, more CAM cells would provide more speedup
- To implement multi-vector operations CAM cells need to be enhanced and share information between themselves
- Thus vAdd and vSub instructions are not implemented (but have been developed)
- Future tasks would include : the CAM enhancements brought up earlier, LUT use optimization (mainly in the CAM and FSM)

Some literature on the subject

> - H Caminal et al, *CAPE: A Content-Addressable Processing Engine*, 27th IEEE Symposium on High-Performance Computer Architecture (HPCA-27), *Feb 2021*, available here
> - A. Salik et al, *Content Addressable Parallel Processors on a FPGA*, arXiv:2106.11376v2, *Jun 2021*, available here
> - C. C. Foster, *Content addressable parallel processors*, ser. Computer science series. New York: Van Nostrand Reinhold, *1976*

Authors contact in case you need it:

- Valentin Darmon (*worked on the FSM & CAPP architecture*): valentin.darmon2@gmail.com
- Maxime Gras-Chevalier (*worked on the CPU & programming*) : mgraschevalier@icloud.com
- Kevin Rolando (*worked on CAM mechanisms*) : kevin.rolando.pro@gmail.com
- Louis Saubole (*worked on CAM & CAPP architecture*) : louissaubole@gmail.com