Content Addressable Parallel Proccessors (CAPPs) on a FPGA University of Massachusetts, Amherst

Ayush Salik*, Dr. Manor Askenazi †, Dr. Edward Rietman‡
BINDS Lab, CICS,
University of Massachusetts, Amherst
Email: *asalik@umass.edu, †manor@biomedical.hosting, ‡erietman@gmail.com

***Abstract*—We designed a synthesizable module for Content Addressable Memory on a FPGA. The module designed is expandable and uses a very small amount of LUTs. It can also be used as a co-processor for a Von Neumann based CPU machine using serial communication. This type of memory is able to provide constant time complexities for searching, reading and writing by leveraging certain circuity around each cell of memory.**

***Index Terms*—FPGA, Verilog, CAM**

I. Introduction

Content Addressable Parallel Proccessors (CAPPs) are an alternate architecture for memory. Unlike Random Access Memory (RAM) which works by performing operations for a word by using a memory address, a CAPP is able to select multiple memory addresses based on some information about the words. Furthermore, a CAPP can perform operations like multi-writing into and reading from multiple selected cells in constant time. This is the reason CAPPs are so useful in routers and bridges as they are a much faster way to search for IP addresses to send packets to.

CAPPs are usually available in the form of Application Specific Integrated Chipsets (ASICs) and are used in routers, databases and more. They can be divided into two broad categories, one being Binary CAPPs and the other Ternary. Ternary CAPPs come with the added flexibility of searching for words with masked bits, which is very similar to the regex for '.'. Of course, this comes with added circuity to each cell.

The goal of this project was to design a module for an expandable Ternary CAPP that is synthesizable on a FPGA using modern HDL tools. We also integrated a finite state machine that encapsulated the T-CAPP and a Serial UART module. This enabled the researchers to directly communicate with the T-CAPP from a CPU using serial communication. The FSM enabled commands for reading, writing, selecting first as well as searching.

CAPPs have potential in graph theory, neural network caching, high-speed indexing, regex computations and a lot more. We hope that an open-source, expandable CAPP that can be synthesized on a FPGA can fill the gap for researchers to find new applications using an alternate memory architecture.

II. Content Addressable Parallel Processor

The circuitry of a CAPP can be broken down into 3 main parts. The tags, the cells and the search registers. The search registers are the least complex circuits in a CAPP. They consist of 2 different registers, the comparand and the mask. The comparand is the word to search for, and the mask contains locations of bits in the word to ignore during the search. For example, if we want to ignore the third character in our search, we would write mask bits [16:23] as high. The design for these are given in figure 1. (TODO)

The second most complex circuitry is found in the tags. Tags are bits for all cells, if the bit of a cell is high after a search, it means that the word in that cell was found in the search. In other words, after a search, the cells with high tag bits are the ones that were found. As all the tag bits of a CAPP have to change in constant time (or in parallel), match lines from the search registers go through each cell to create mismatch lines which turn irrelevant tags off whenever the search signal is set. The circuitry also includes logic for manually setting all bits high and selecting the first tag bit that is on. The design for these are given in figure 2.(TODO)

The most complex circuitry is found in the memory cells as it encapsulates the main logic for writing, reading and searching. Each bit of each cell is surrounded by 2 write lines, 2 search lines and 1 read line. The nth bits of all cells share these lines. The value of each bit in each cell is changed to 0 or 1 depending on which write line is set high. Inversely, depending on the 2 search lines and the bit value, a mismatch line that is connected to the tag bit of the cell is set high or low. This allows the CAPP to search or write in parallel. The output of all the read lines is the bitwise OR of all cells that have their tag bits on. The circuits as implemented our shown in the figures 3-5 (TODO).

Our design of the CAPP is heavily influenced by Caxton Foster's book on the subject [1]. We used System Verilog for

our design, along with Next PnR. This placing and routing tool was used because it accommodated factors like timing into place. This was crucial for a clock dependent machine with dense circuitry like a CAPP along with the Serial UART module that is used for control. The whole module was synthesized and tested on a TinyFPGA-Bx and a Python library was written to communicate with it. The FSM that acts as a central command center for the CAPP is discussed in the next section.

III. The Finite State Machine

The FSM is the crux of interacting with the CAPP and making it carry out complex arbitrary algorithms by exposing commands to carry out different tasks. It has numerous states, some of them are sending or receiving data, loading data into the CAPP, selecting first, searching, reading, setting the comparand and mask as well as writing. The base UART module used is from David Things' Github repository. [2] This works on a 48MHz clock cycle. To reduce complexities and avoid additional LUTs usage, we used the same clock speed throughout the project.

As shown in figure 4 (TODO), our FSM acts as a way to carry out procedural tasks while staying under the limit of 21 ns by linking states together. Therefore, a task may trigger several states before it returns to the default state. For example, the algorithm for searching has several steps, it comprises of

- Setting the comparand
- Setting the mask
- Sending the SET signal
- Sending the SEARCH signal

Notice that the first two steps use several clock cycles as only one byte of data flows through the UART each clock cycle. This is due to its pipeline design.

The states transitions for the SET signal is given below:

- SET 1: change SET to high, set delay to 5 clock cycles
- IDLE: wait for delay, go to SET 0
- SET 0: change SET to low, listen for new command

This is similar to the SEARCH signal:

- SEARCH 1: change SEARCH to high, set delay to 5 clock cycles
- IDLE: wait for delay, go to SEARCH 0
- SEARCH 0: change SEARCH to low, listen for new command

The tasks that involve transmitting follow a similar state transition procedure. Here, a state send one character to the host through the uart each clock cycle. Examples of these are sending comparand, tags and mask. The state transition diagram of sending the comparand is shown in Figure 3. The procedures for the other commands are similar in the way that different states just change the value of a register. The send state just transmits data from this register. This design decision was made to reduce the memory as well as number of combinatorial circuits.

- GET COMPARAND: set output text equat to comparand
- SEND: Send the next character until EOL. Listen for next command

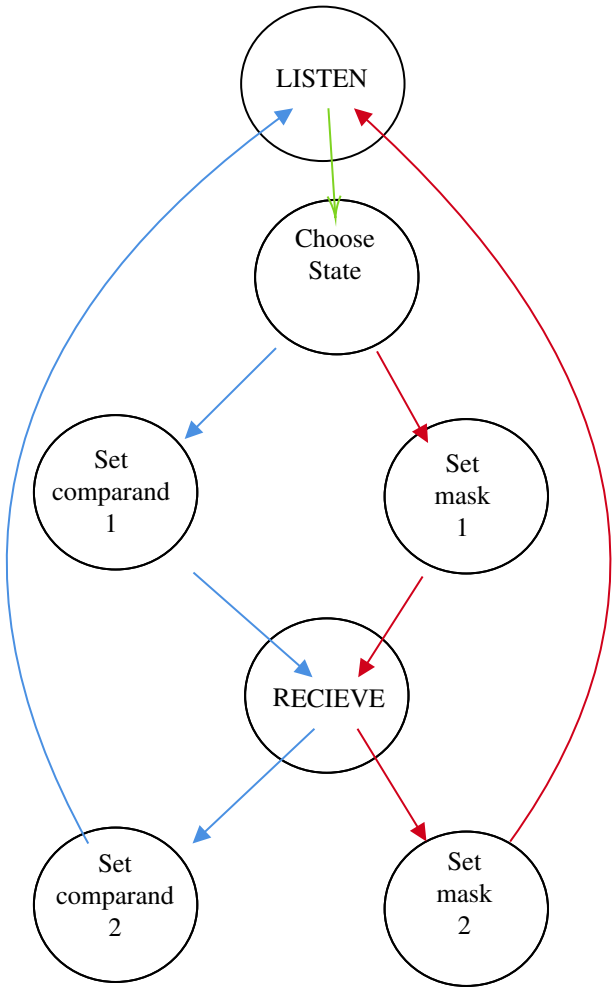

Figure 1. The transitions of the FSM for setting the comparand (blue) and the mask (red). Each transition happens on a different clock cycle to maintain the inner state of the memory and to allow time for the data to complete its flow.

IV. Results

A Content Addressable Parallel Processor was successfully completed. The machine works in conjunction with a machine with Von Neumann architecture that is responsible for streaming commands to the CAPP which enacts them on the CAM. The commands are sent over serial using a UART module on the FPGA

As the CAPP works on the principles of a CAM, it can search for matching cycles in constant time, and write to those in constant time too.

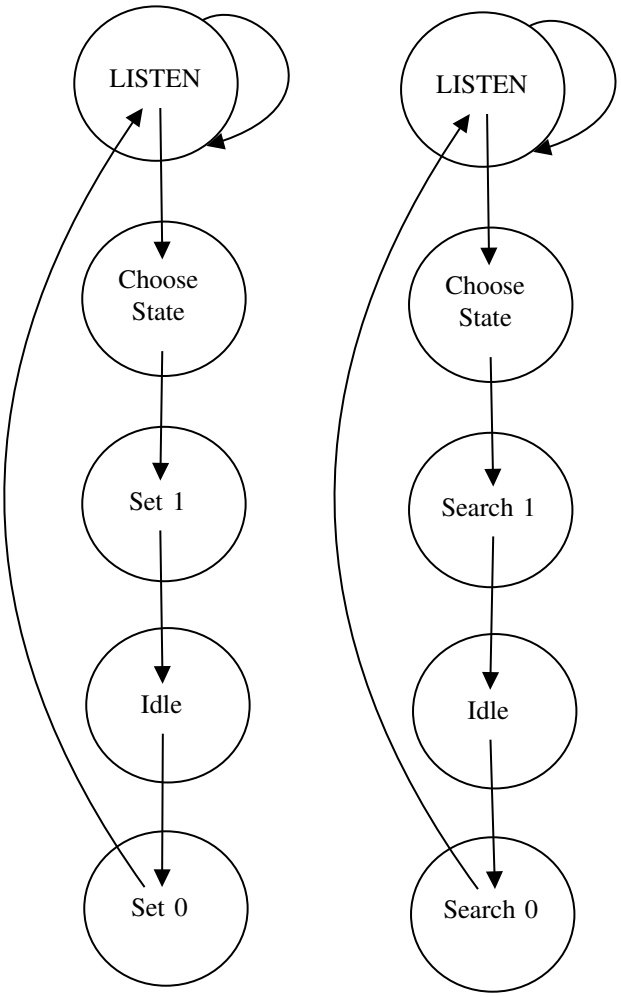

Figure 2. The transitions of the FSM for (left) sending signals for seeting all tag bits high, as well as (right) performing search using the comparand and the mask.

We also wrote a Python library that is used to initialize the CAPP and send commands to it from the external Von Neumann machine. This makes it easy to transmit commands and read cells in the CAM.

V. Discussion

Earlier on, in the late 20th century, CAPPs were disregarded as compared to Von Neumann Machines. This was because Von Neumann machines were relatively cheaper and simpler, they didn't have elaborate control circuits to each cell of memory.

Over the years, manufacturing and designing costs have gotten lower which makes it a good time to reconsider them. We think there lies great opportunity in CAPPs. This alternate architecture can read, search and write within the worst complexity of O(1). In the modern computer, that is based on Von Neumann machines, these operations have the worst time complexity of O(n).

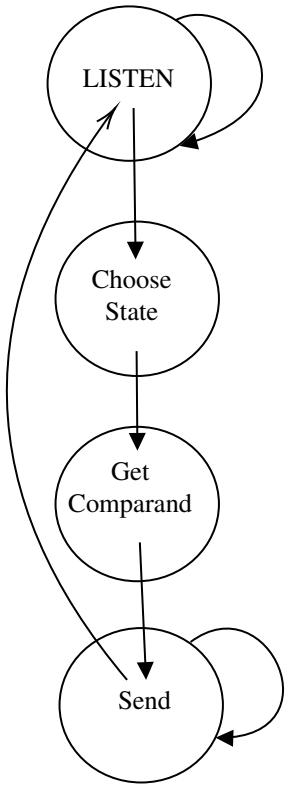

Figure 3. State transition diagram for sending data to the host. The Get Comparand state sets output text as comparand and moves on to the Send state which transmits the data byte by byte.

CAPPs are widely recognized as a way to do fast lookups. Our perspective is that they can used in other ways too. One of them is already proposed in this paper (neural network caching paper citation). Additionally, it can also be used for parallel regex operations, testing reachability in graph and much more. A CAPP can also be used to run a truly parallel interpreted language, where each command would be stored in a cell and a portion of each cell is dedicated to the order of commands.

Acknowledgment

The authors would like to thank ... for his/her/their help and support during the process of writing this paper. [**?**], [**?**], [**?**], [**?**]

References

[1] C. Fostor, *Content Addressable Parallel Processors*. Litton Educational Publishing, Inc., 1929.

[2] "TinyFPGA BX USB Serial." [Online]. Available: https://github.com/davidthings/tinyfpga_bx_usbserial