Implementation and Design of High Speed FPGA-based Content Addressable Memory

Anupkumar Jamadarakhani[1] Shailesh Kumar Ranchi[2]
[1, 2] M. Tech (VLSI)
[1, 2]VTU RC Gulbarga

Abstract— CAM stands for content addressable memory. It is a special type of computer memory used in very high speed searching application. A CAM is a memory that implements the high speed lookup-table function in a single clock cycle using dedicated comparison circuitry. It is also known as associative memory or associative array although the last term used for a programming data structure. Unlike standard computer memory (RAM) in which user supplies the memory address and the RAM returns the data word stored in that memory address, CAM is designed in such a way that user supplies data word and CAM searches its entire memory to see if that data word stored anywhere in it. If the data word is found, the CAM returns a list of one or more storage address where the word was found. This design coding, simulation, logic synthesis and implementation will be done using various EDA tools.

Key words: CAM, Data word, RAM, EDA tools, NAND cells, NOR cells.

I. INTRODUCTION

A Content- Addressable memory (CAM) compares input search data against a table of stored data, and returns the address of the matching data [1]–[5]. CAMs have a single clock cycle throughput making them faster than other hardware- and software-based search systems. CAMs can be used in a wide variety of applications requiring high search speeds. The time required to find an item stored in memory can be reduced considerably if the item can be identified for access by its content rather than by its address. CAM provides a performance advantage over other memory search algorithms, such as binary or tree-based searches or look-aside tag buffers, by comparing the desired information against the entire list of pre-stored entries simultaneously, often resulting in an order-of-magnitude reduction in the search time. The primary commercial application of CAMs today is to classify and forward Internet protocol (IP) packets in network routers [15]–[20]. CAM is ideally suited for several functions; including Ethernet address lookup, data compression, pattern-recognition, Cache tags, high bandwidth address filtering, fast lookup of routing, user privilege and security or encryption information on a packet-by-packet basis for high-performance data switches, firewalls, bridges and routers.

The basis for the interconnections of the Internet is IP (Internet Protocol). IP applications grow very rapidly with requirements doubling every three months. In the future, IP will not only be used to interconnect computers, but all kinds of equipment will use this protocol to communicate with each other including base stations for cellular communication. Due to the increasing demand for high bandwidth, many efforts are made to make faster IP handling systems. Not only speed, but also flexibility is an important factor here, since new standards and applications have to be supported at all times. A way to gain speed and flexibility is to move critical software functions to reconfigurable hardware.

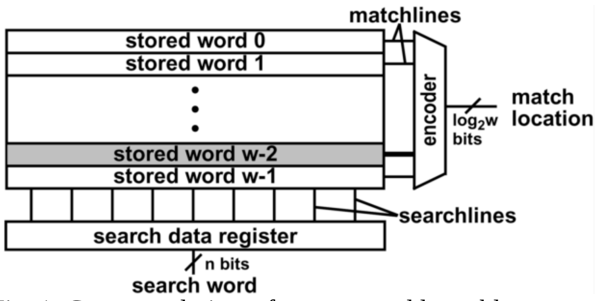


Fig. 1: Conceptual view of a content-addressable memory containing words. In this example, the search word matches location as indicated by the shaded box. The match lines provide the row match results. The encoder outputs an encoded version of the match location using bits.

Fig. 1 shows a simplified block diagram of a CAM. The input to the system is the search word that is broadcast onto the search lines to the table of stored data. The number of bits in a CAM word is usually large, and may range from 36 to 144 bits. A typical CAM employs a table size ranging between a few hundred entries to 32K entries, corresponding to an address space ranging from 7 bits to 15 bits. Each stored word has a match line that indicates whether the search word and stored word are same that is the match case or are different that is the mismatch case. The match lines are fed to an encoder that generates a binary match location corresponding to the match line that is in the match state. An encoder is used in systems where only a single match is expected. In CAM applications where more than one word may match, a priority encoder is used instead of a simple encoder. A priority encoder selects the highest priority matching location to map to the match result, with words in lower address locations receiving higher priority. In addition, there is often a hit signal (not shown in the figure) that flags the case in which there is no matching location in the CAM. The overall function of a CAM is to take a search word and return the matching memory location. This way matching is done purely in hardware and is therefore faster than the software solution.

A. *Basics of CAM*

Since CAM is an outgrowth of Random Access Memory (RAM) technology, in order to understand CAM, it helps to contrast it with RAM. A RAM is an integrated circuit that stores data temporarily. Data is stored in a RAM at a particular location, called an address. In a RAM, the user supplies the address, and gets back the data. The number of address line limits the depth of a memory using RAM, but the width of the memory can be extended as far as desired.

With CAM, the user supplies the data and gets back the address. The CAM searches through the memory in one clock cycle and returns the address where the data is found. The CAM can be preloaded at device start-up and also be rewritten during device operation. Because the CAM does not need address lines to find data, the depth of a memory system using CAM can be extended as far as desired, but the width is limited by the physical size of the memory.

CAM can be used to accelerate any application requiring fast searches of data-base, lists, or patterns, such as in image or voice recognition, or computer and communication designs. For this reason, CAM is used in applications where search time is very critical and must be very short. For example, the search key could be the IP address of a network user, and the associated information could be user's access privileges and his location on the network. If the search key presented to the CAM is present in the CAM's table, the CAM indicates a 'match' and returns the associated information, which is the user's privilege. A CAM can thus operate as a data-parallel or Single Instruction/Multiple Data (SIMD) processor.

B. *Technical aspect of packet forwarding using CAM*

Network routers forward data packets from an incoming port to an outgoing port, using an address-lookup function. The address- lookup function examines the destination address of the packet and selects the output port associated with that address. The router maintains a list, called the routing table that contains destination addresses and their corresponding output ports. An example of a simplified routing table is displayed in Table I. All four entries in the table are 5-bit words, with the don't care bit, "X", matching both a 0 and a 1 in that position. Because of the "X" bits, the first three entries in the Table represent a range of input addresses, i.e., entry 1 maps all addresses in the range 10100 to 10111 to port A. The router searches this table for the destination address of each incoming packet, and selects the appropriate output port. For example, if the router receives a packet with the destination address 10100, the packet is forwarded to port A. In the case of the incoming address 01101, the address lookup matches both entry 2 and entry 3 in the table. Entry 2 is selected since it has the fewest "X" bits, or, alternatively, it has the longest prefix, indicating that it is the most direct route to the destination. This lookup method is called longest-prefix matching.

Fig. 2 illustrates how a CAM accomplishes address lookup by implementing the routing table shown in Table I. On the left of Fig. 2, the packet destination-address of 01101 is the input to the CAM. As in the table, two locations match, with the (priority) encoder choosing the upper entry and generating the match location 01, which corresponds to the most-direct route.

Entry no.	Address	Output
1	101XX	A
2	0100X	B
3	011XX	C
4	10011	D

Table. 1: Example of Routing Table

This match location is the input address to a RAM that contains a list of output ports, as depicted in Fig. 2. A RAM read operation outputs the port designation port B, to which the incoming packet is forwarded. We can view the match

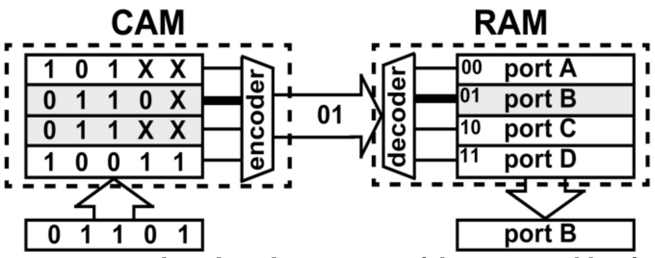


Fig. 2: CAM-based implementation of the routing table of Table 1

Location output of the CAM as a pointer that retrieves the associated word from the RAM. In the particular case of packet forwarding the associated word is the designation of the output port. This CAM/RAM system is a complete implementation of an address-lookup engine for packet forwarding.

II. CAM CELL DESIGN

A CAM cell serves two basic functions: bit storage, as in RAM; and bit comparison, which is unique to CAM. Although there are various cell designs, for bit storage, typically a CAM cell uses an SRAM internally. The bit comparison function, which is logically equivalent to an XOR of the stored bit and the search bit, can be implemented two different approaches: the NOR cell and the NAND cell. Both approaches have their advantages, and there are various design optimizations that apply to each. Although some CAM cell implementations use lower area DRAM cells [5], [15], typically, CAM cells use SRAM storage.

A. *NOR Cell*

The NOR cell implements the comparison between the complementary stored bit, D (and D), and the complementary search data on the complementary search line, SL (and SL), using four comparison transistors M1-4, which are all typically minimum size to maintain high cell density. These transistors implement the pull down path of an XNOR gate to compare SL and D, such that a mismatch of SL and D creates a discharge path for the match line, ML, which was precharged high before the evaluation phase. For the match case, both pull down paths are disabled, and the ML stays high. A mismatch in any of the cells creates a path to ground and the match line is discharge.

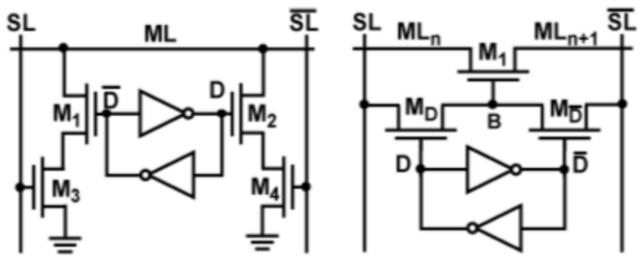


Fig. 3: Standard 10-T NOR cell and Standard 9-T NAND cell

A typical NOR search cycle operates in three phases: search line precharge, match line precharge, and match line evaluation. First, the search lines are precharged low to disconnect the match lines from ground by disabling the pull-down paths in each CAM cell. Second, with the pull down paths disconnected, a pull-up transistor precharges the match lines high. Finally, the search lines are driven to the search word values, triggering the match line evaluation phase. The main feature of the NOR match line is its high speed of operation. In the slowest case of a one-bit miss in a word, the critical evaluation path is through the two series transistors in the cell that form the pull down path.

B. NAND CELL

The NAND cell implements the comparison between the stored bit, D, and corresponding search data on complementary search lines, (SL,SL), using the three comparison transistors , M1, MD and MD, which are all typically minimum size to maintain high cell density. One can recognize node B as the PTL implementation of the XNOR function of inputs SL and D. In the case of a match, node B becomes high, turning transistor M1 on, which connects MLn and MLn+1 in series. The mismatch case leaves node B low, disabling transistor M1 and disconnecting MLn and MLn+1. The NAND nature of this cell is better understood when multiple cells are connected in series to for a CAM word, for which the ML resembles the pull down path of a CMOS NAND gate.

The NAND search cycle starts with precharging ML with a PMOS transistor. Next, unlike the NOR cell, the NAND structure includes an NMOS evaluation transistor which is activated after precharge phase. In the case of a match, all XNORs in each CAM cell in the word evaluate to 1, and transistors M1 through Mn form a discharge path for ML. For the mismatch case, ML remains high. A sense amplifier detects the low (i.e. match) or high (i.e. miss) cases on ML.

An important feature of the NAND cell is that a miss case in a cell stops the signal propagation to following cells [7]. This means no power consumption after the final matching transistor in the series MLi chain. If we consider the whole CAM array, typically only one word is in the match state, which means only a small number of cells consume power in the rest of the array? A downside to the NAND cell is that its delay grows qudratically with the number of cells. Using NMOS pass transistors also cause a Vtn drop for the gate voltage applied to the access transistors Mi, which further limits the highest voltage on the match line to V DD - 2Vtn.

III. VIRTEX ARCHITECTURES

This section describes the architecture of the Xilinx Virtex FPGA series. Xilinx Virtex has a regular structure, consisting of configurable logic blocks (CLBs) surrounded by programmable input/output blocks (IOBs) [11]. This is shown in figure 4.

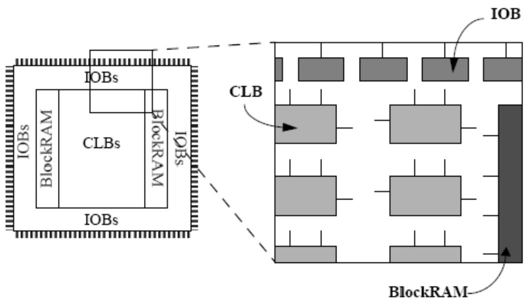


Fig. 4: Global Architecture of Virtex FPGA

The CLBs provide the functional elements for constructing logic and IOBs provide the interface between the package pins and the CLBs. By connecting the IOBs and CLBs together using general routing resources, a complex circuit can be built.

Except for the CLBs and IOBs, Virtex also contains integrated SRAM blocks, called Block RAM and 3-State Buffers (TBUFs). The CLBs, Block RAM and TBUFs will be discussed in the next paragraphs.

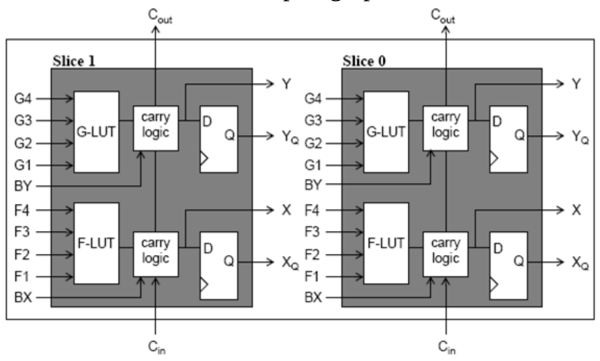


Fig. 5: Virtex CLB

A. Configuration Logic Blocks

A schematic view of the Virtex CLB is given in figure 5. It consists of two identical parts, called slices and each slice has two logic cells (LCs). An LC includes a 4-input look-up table (LUT), carry logic and a storage element.

The LUTs can be configured in any combinatorial function of four inputs or 16x1 synchronous RAM or 16 bit shift register.

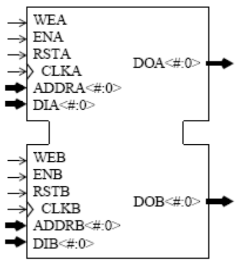
The storage elements in the Virtex slice can be configured either as edge-triggered D-flip-flops or as level-sensitive latches. The carry-logic, shown in figure 5, can be used for fast arithmetic functions, but also for cascading LUTs for implementing wide logic functions. The Virtex 1000 has an array of 64 x 96 CLBs.

B. Tri-state Buffers

Each Virtex CLB contains two 3-State buffers (TBUFs) that can drive on-chip buses. These on chip busses are provided by horizontal routing resources and four bus lines are provided per CLB row, as shown in figure 6.

The TBUFs as implemented on the Virtex are no true TBUFs, but instead they are implemented using a logical circuit that emulates the behaviour of a true 3-State buffer. This way, several TBUFs may drive a line simultaneously without the device getting damaged. When at least one TBUF drives a ‘0’ on a line, the logic value of that line becomes ‘0’ no matter what the output values of the other TBUFs are.

C. Block RAM



Width	Depth
1	4096
2	2048
4	1024
8	512
16	256

BlockRAM port configurations

Fig. 6: Dual-Port Block RAM with possible port configurations

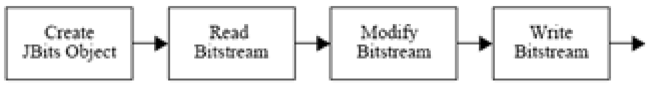


Fig. 7: JBits Programming model

The Virtex contains 32 Block RAMs, organized in two columns along each vertical edge of the chip. Each such memory cell is a dual ported 4096-bit RAM with independent control signals for each port as illustrated in figure 6. The data widths of the two ports can be configured independently according to the table in the figure.

D. *About JBits*

JBits is a set of Java classes which provide an Application Program Interface (API) into the Xilinx FPGA bit stream [13]. This interface operates either on bit streams generated by design tools, or on bit streams read back from actual hardware. This provides the capability of designing, modifying and dynamically modifying the logic on an FPGA. JBits gives the possibility to manually place, route and reconfigure the FPGA on a CLB level with relatively simple commands. This makes it very suitable for dynamically reconfiguring regular structures, such as the CAM. The diagram in figure 7 illustrates the essential steps involved in the development of a JBits application.

IV. CAM STRUCTURES AND THEIR TYPES

This includes general overview on CAMs, including the definition of explicit priority. Then two different CAM structures are discussed and the way they can be mapped onto the Virtex architecture. In the first structure, the same amount of area is reserved for all entries. This will be referred to as fixed length CAM. The second structure shows much similarity with the fixed length CAM, but instead the area that each entry occupies is variable and depends on the number of 'don't cares'. This structure is called variable length CAM. Both these implementations utilize dynamic reconfiguration for updating their content. Other CAM structures that do not use dynamic reconfiguration, but are suitable for implementation on FPGA can be found in [14]-[17]. Next the structure of an explicit priority encoder is described, that can be integrated with one of the earlier described CAM implementations.

A. *Binary versus Ternary CAMs*

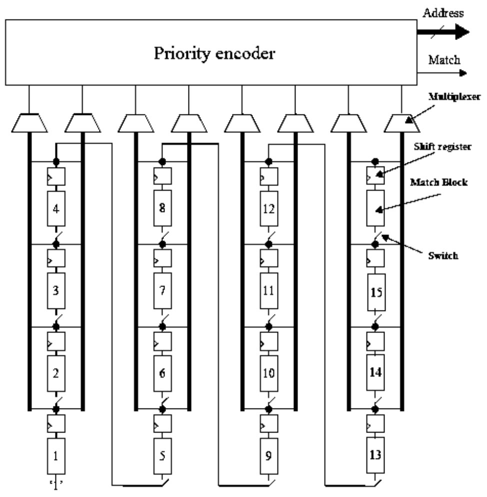


Fig. 8: Schematic view of CAM with Entries of variable length

A binary CAM stores only one of two states ('0' and '1') in each memory location (i.e. in each bit of a word); a ternary CAM stores one of three states in each memory location. These three states are represented by: '0', '1', and 'X'. Ternary CAMs may have a global mask as well. This allows also the search pattern (i.e. the bit vector that is used as an input of the CAM) to contain 'X's. This is especially useful when the width of the search pattern is small, such that two or more entries can be stored in the same CAM location.

B. *Fixed length CAMs*

The global structure of the fixed length CAM is given in figure 1. It is a PLA structure, consisting of matching lines and an encoder. The encoder is used to translate the outputs of the matching lines to the address of the line that gave a match. This can either be an inherent or an explicit priority encoder.

C. *Variable length CAMs*

The variable length CAM is a CAM where the stored entries have variable length, depending on the number of 'don't cares' they contain. The reason for implementing this kind of CAM is that the number of 'don't cares' is quite large in general.
The global structure of the variable length CAM with a maximum of 16 entries is given in figure 8. It consists of a long chain of match blocks and shift registers, separated by switches and placed into match lines of four blocks each.
Programming the CAM is done by mapping entries on match blocks and shift registers and placing the entries on the chain starting at match block 1. Blocks within an entry are connected together by closing a switch, which causes the carry signal of a block to be propagated to the next block. An open switch on the input of a block means that its input becomes '1' and starts a new entry. The multiplexers are used to connect the outputs of the entries to the priority encoder.
Programming the CAM consists of two steps:

1) Mapping: Dividing entries into 64 bits blocks separated by delays, leaving out blocks that merely contain don't cares'.
2) Placing: Placing the mapped entries on the actual CAM structure and connecting their output to the priority encoder.

There are a few special cases that need to be looked at separately:

1) Empty entries: If a whole entry consists of don't cares, no match blocks are used to store this entry. Instead the corresponding multiplexer to which the entry would have been connected is programmed to output '1' independent of the input.
2) Beginning of entry is empty: If one or more blocks in the beginning of the entry are empty, then these blocks do not cause any delay to be incremented.
3) More than two outputs in a line**:** Since there are two multiplexers available per match line, a maximum of two entries can have their output in a line. When a new entry is added, which would cause the number of outputs in a match line to become three, the output of this entry is shifted to the next line. This leads to match blocks that are not used.

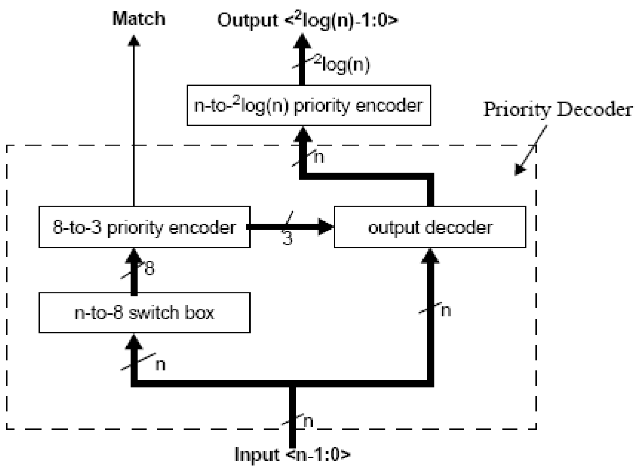


Fig. 9: Global structure of an n-to-2log (n) explicit priority encoder with eight priority classes.

D. *Explicit Priority Encoder*

As described before, there are two mechanisms for prioritizing: inherent priority and explicit priority. In the latter case, not only the words that are searched are added to the CAM, but also their priority. This way, new words are always added in the end or at empty places and shifting other entries are not necessary.

1) *Different Implementation*

To implement explicit priority, several schemes are possible. The common way to do explicit encoding is by adding an explicit priority field to each CAM word [17]. Each cycle, the system combines the search word with a different priority word. In the first cycle of the search, the system sets the priority word to the highest priority. If a match occurs, the address of the matching entry is returned; else the system combines the search word with the next highest priority. This procedure is repeated until either a match occurs, or the lowest priority is reached. The advantage of this algorithm is that it is easy to implement and no extra hardware design effort is needed. On the other hand, the algorithm is not very efficient and the matching process can take many clock cycles, depending on the number of possible priority values.

Using dynamic reconfiguration, other implementations are possible. One of these possibilities is using a regular priority encoder in combination with a switch box. This switch box routes every output of the CAM to the correct input of the priority encoder and the configuration of the switch box is controlled by JBits. Although this method is efficient in time, it would consume too much hardware for the CAM size at hand. This problem can be solved by reducing the number of priority classes. The number of priority classes is defined as the number of explicit priority values that a search word can have. In case of an inherent priority encoder, this value is equal to the number of entries. By reducing this number, the amount of hardware is reduced, but there is a risk that more priority classes than available are needed for a certain CAM configuration. To solve this, a combined explicit/inherent priority encoder is proposed, where the priority can be set explicitly for each entry, but in case two entries have the same explicit priority, their priority is determined inherently. This way, entries can be added even when all priority classes have been used, just as with the inherent priority encoder.

2) *Global Structure*

The global structure of the n-to-2log (n) explicit priority encoder with eight priority classes is given in figure 9. Estimating the number of priority classes that is needed is difficult and requires again information about the actual content of the CAM. A number of eight has been chosen, since this can be mapped efficiently on the Virtex architecture as will be demonstrated. The priority encoder consists of two basic blocks. First there is a priority decoder. The input of this block is coming from the match lines and contains ‘1’s at all matching positions. The output is a bit vector with ‘1’s only at those positions that match and have highest priority. In case different priority classes have been used for all ‘overlapping’ entries (i.e. entries that may match simultaneously), then the output of the priority decoder contains no more than one ‘1’.

The output of the priority decoder is connected to a regular n-to-2log (n) priority encoder that is needed to decide the return value when two or more overlapping entries with the same explicit priority match simultaneously. The priority decoder consists of three parts, as shown in figure 9 and works as follows. The n-to-8 switch box connects each of the n input lines to one of 8 priority lines. These lines, each representing a priority class are connected to an 8-to-3 priority encoder that looks if there is a match and if there is, it extracts the value of the highest priority of all the input lines that match. The output decoder propagates the value of each input line only if the priority that this line is set with corresponds with the highest priority, else ‘0’ is propagated at this bit position.

To program the explicit priority encoder, the switch box and the output decoder need to be programmed using JBits and the implementation of these blocks is discussed next.

V. DEVICE UTILIZATION

For Fixed length CAM, each Virtex Slice is able to match 8 bits. The fixed length CAM consists of 128 match lines, each containing 5 blocks matching 64 bits each. The number of slices taken by the match field is then equal to: 128 x 5 x 8 = 5120 slices. Since the total number of slices is 12288, 42% of the slices are consumed, not counting the encoder. The flip flops on the output of each match block have been neglected, because these consume little resources and can be combined with other logic in the same slice.

For Variable length CAM, The number of match lines, blocks and multiplexers were chosen from a design point of view without taking into account what the actual format of the entries is. In case there are a lot of entries containing only one block after mapping, the number of multiplexers should be increased to minimize the number of unused blocks. Another problem is choosing the size of the priority encoder. If there are many long expressions, there are a lot of inputs that are not used and it’s necessary to have a large priority encoder or longer match lines. If there are many short expressions, a smaller encoder can be used.

From this it becomes clear that knowledge about the actual content of the CAM is necessary to implement it in an efficient way, which is not available for IP version 6 yet.

	Inherent priority	Inherent priority
Fixed length CAM	43%	45%
Variable length CAM	44%	48%

Table. 2: Device Utilization

Port	Direction	Function	No. of bits
clk	In	System clock	1
Reset	In	Reset CAM	1
Ctrl_Reg	In	Start Matching	8
Stat_Reg	Out	Matching Ready	8
Data 0	In	Indata [31:0]	32
Data 1	In	Indata [63:32]	32
Data 2	Out	Match, Address	32

Table. 3: Function Assignments for FGPA Ports

The number of match lines has been chosen to be 128, and the CAM can therefore store up to 256 entries. The variable length CAM has 128 match lines of 4 match blocks each. In the worst case, all entries that are stored have 5 blocks after mapping and only 128 x 4/5 = 102 entries can be stored, but this is very unlikely.

Each Virtex Slice is able to match 8 bits, so that 8 slices are needed per match block. The CAM consists of 128*4 = 512 blocks, so a total of 512 x 8 = 4096 Virtex slices (33%) are consumed. Each shift register consumes one slice, since it's not possible to use the second slice for something else. There are 512 shift registers in the design, meaning a utilization of 4%. The two multiplexers that each match line has to select the output have to be mapped in separate slices. These multiplexers therefore consume 2 x 128 = 256 slices (2%). The total slice utilization of the variable length CAM without the encoder is then 39%.

For Inherent priority Encoder, the device utilization of the inherent priority encoder depends on the CAM structure it is used with. The fixed length CAM requires a 128-to-7 priority encoder, while the variable length CAM requires an encoder that is twice as wide. The utilization by the inherent priority encoder has been determined by synthesizing both sizes and examining the mapping report. The results are:
128-to-7 bits: 179 slice (1%).
256-to-8 bits: 709 slice (5%).
The device utilization of the explicit priority encoder has also been estimated to be used with both fixed and variable length CAM. The fixed length CAM has 128 outputs, i.e. a 128-to-8 explicit priority encoder is needed. The 128-to-8 bits switch box is divided in 32 8-to-8 switch boxes, each containing 8 input multiplexers that use 1 slice each. The total number of slices for the switch box is then 128 = 1%. The output decoder consumes 128 LUTs. Normally two LUTs can be placed in one slice, but it is not possible to constrain a LUT to a certain position within a slice. In the case of a carry chain, the order of the LUTs is set implicitly by the direction of the carry chain. However, there is no carry chain now and it is therefore necessary to place the LUTs in separate slices. The output decoder therefore takes 128 (= 1%) slices.

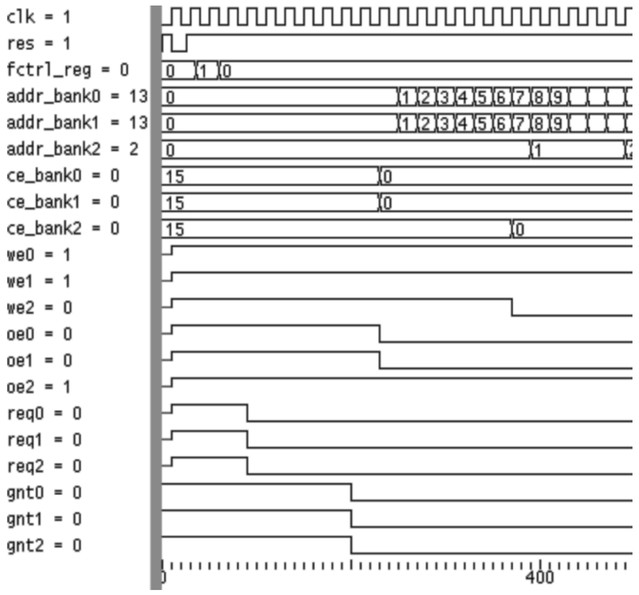


Fig. 10: Simulation results for the board interface

The utilization by the inherent priority encoder is equal to 171 slices = 1%. The total utilization by the 128 bits explicit priority encoder is then 3%, where the slices consumed by the 8 to 3 priority encoder have been neglected. Repeating this calculation for the 256 bits explicit priority encoder leads to a utilization of 9%.

From the table II, it follows, that the hardware utilization of the fixed length CAM and the variable length CAM are about equal for these dimensions. Furthermore it follows that the hardware cost of using an explicit priority encoder instead of using inherent priority is small and therefore interesting.

VI. IMPLEMETATION OF THE BOARD INTERFACE

The board interface takes care of the communication between the FPGA, the host and the on-board memory and is situated on the FPGA. This paragraph contains a description of the board interface and how it has been implemented together with simulation results.

A. *Port Description*

General overview of the board suggest, with a description of the ports that are available for communication between host, memory and FPGA. To use the board as part of the CAM application, these various ports were assigned a function. These functions are summarized in table III, together with the direction of the signals viewed from the FPGA side.
Besides these ports, other signals are necessary for controlling the memory, control register and status register.

B. *VHDL Description*

The VHDL description of the board interface is given in appendix B. The board interface is a finite state machine (FSM) that repeatedly reads data from memory to the CAM and writes the result from the CAM to memory. The behaviour of the board interface has been simulated and the result is given in figure 10.

First a reset is applied, that initializes the control signals and brings the FSM in state 'IDLE'. Then value '1' is written to the control register, which is interpreted as start. The board interface sends a memory request for bank 0, bank 1 and bank 2 to the on-board memory arbiter and waits until all banks have been granted. Next the board interface starts reading from bank 0 and bank 1. After 7 clock cycles (5 for processing by the CAM and 2 for latency due to the registers before and after the CAM) the result is written to bank 2.

From this moment Indata is read every clock cycle and the result is written every 5 clock cycles until Addr0 is greater than Buffer Size. For debugging purposes, LEDs are turned on and off depending on the state of the FSM.

VII. HARDWARE IMPLEMENTATION OF THE CAM

In this section, the VHDL implementations of the fixed length CAM, the variable length CAM and the explicit priority encoder are discussed. This meant to show how the designs have been described in a structural style, by using the hardware primitives available for Virtex. This type of VHDL description is necessary in applications that use dynamic reconfiguration, since full control over the implementation of specific parts of the design is necessary.

A. Implementation of the Fixed Length CAM

The structure of the VHDL description of the fixed length CAM is shown in figure 11. It shows the entities that are used and how they relate to each other.
Entity DecLut defines the LUTs that are part of the match lines and store the actual entries. Encoder is the priority encoder.

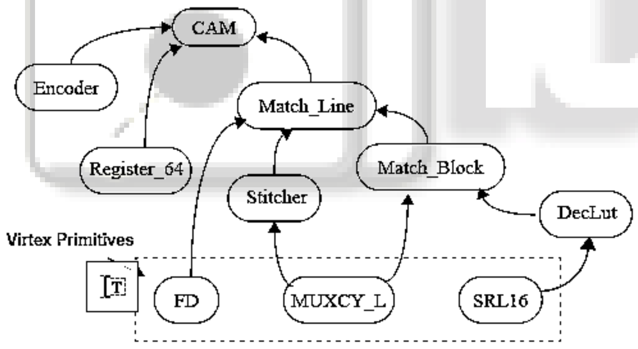


Fig. 11: Structure of the VHDL Description of the Fixed Length Cam

B. Implementation of the Variable Length CAM

The structure of the VHDL code of the variable length CAM is much like the fixed length CAM and is given in figure 12. The main differences are that two multiplexers were added for each match line (entity Mux4) and flip flops were replaced by shift registers.

C. Implementation of the Priority Encoder

The structure of the VHDL description of the explicit priority encoder is given in figure 13, showing all entities and Virtex primitives that have been instantiated. The two priority encoders have been described in a behavioural style, while the switch box and the output decoder were implemented in a structural way, since these need to be configured by JBits. Entity SwitchBox_8x8 uses primitive TBUF, which refers to a tri-state buffer on Virtex.

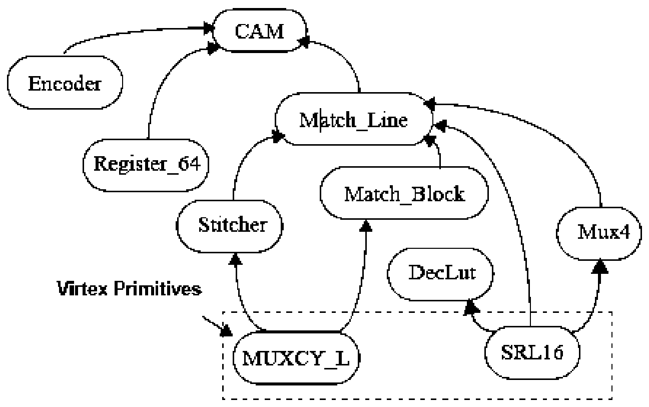


Fig. 12: Structure of the VHDL description of the variable length CAM

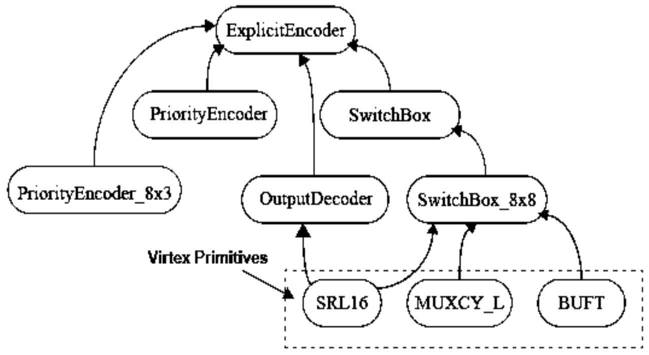


Fig. 13: Structure of the VHDL description of the variable length CAM

VIII. SUMMARY

The goal of this paper was to implement an FPGA-based CAM. This CAM should be able to store at least 128 words with a maximum width of 315 bits and the CAM words may contain 'don't cares'. It should be part of a 622 MBit/sec communication channel, which means that a look up rate of 1.9 million look ups per second is required.

Three different CAM structures were implemented. The first implementation, called fixed length CAM can store 128 entries of 320 bits and all CAM words consume the same amount of resources. In this implementation an inherent priority mechanism is used, meaning that when several CAM words match simultaneously, then the matching word on the lowest address is selected.

The second implementation is called 'variable length CAM' that can store up to twice as many entries on the same area compared to the fixed length CAM. This is done by dividing the CAM words into five match blocks and when such a match block merely contains 'don't cares', then this block is omitted.

The third implementation is based on the variable length CAM, but uses a more advanced priority mechanism where not only the CAM words, but also their priority can be programmed.

To implement the CAMs, a specific design methodology was used, consisting of a static and a dynamic part. The static part is used to implement the basic structure of the CAM, from a VHDL description to the programming bit stream. The dynamic part is a Java application, used to change this bit stream for updating the CAM.

The three implementations were implemented in a Xilinx

Virtex device on a PCI-based board and for each implementation; a Java-based user interface has been developed to configure the CAM.

The fixed length CAM has been implemented successfully, being able to contain 128 CAM words and able to perform 7.1 million lookups/sec. The variable length CAM that has been implemented can contain up to 256 CAM words and searching can be done at a rate of 3.8million lookups/sec.

The explicit priority scheme added in the third implementation allows fast adding/deleting of CAM words and it was shown that this added no significant hardware costs. Searching can be done at a rate of 3.4million lookups/sec.

IX. CONCLUSION

When implementing an FPGA-based CAM, its architecture has to be considered in order to efficiently exploit its resources. This is because, unlike custom circuits, the architecture of the FPGA is fixed a-priori and therefore the permitted programmability, connectivity and rout ability is constrained by that architecture.

Dynamic reconfiguration is a good way to implement FPGA-based CAMs. It was shown that flexible circuits can be implemented, without adding hardware costs. Since critical functions (searching the CAM) and non-critical functions (changing the CAM) can be divided and implemented in respectively hardware and software, the final hardware implementation becomes both faster and smaller than regular FPGA implementations.

The performance of dynamically reconfigurable FPGA-based CAMs is enough for IP characterization.

Since the required search rate is less than 1.9million lookups/s, all three implementation are suitable. With progressing FPGA performance, it is expected that the three CAM structures can be used in future communication channels with more stringent requirements as well.

ACKNOWLEDGMENT

The authors gratefully acknowledge the programming efforts of Mr. Pradeep, Praveen, Shivsharan, Manoj, Revan, Govardhan, Mallu, and Prem from the University of VTU RC GULBARGA. The authors thank for their great feedback on drafts of this paper.

REFERENCES

[1] T. Kohonen, Content-Addressable Memories, 2nd Ed. New York: Springer-Verlag, 1987.
[2] L. Chisvin and R. J. Duckworth, “Content-addressable and associative memory: alternatives to the ubiquitous RAM,” IEEE Computer, vol. 22, no. 7, pp. 51–64, Jul. 1989.
[3] K. E. Grosspietsch, “Associative processors and memories: a survey,” IEEE Micro, vol. 12, no. 3, pp. 12–19, Jun. 1992.
[4] I. N. Robinson, “Pattern-addressable memory,” IEEE Micro, vol. 12, no. 3, pp. 20–30, Jun. 1992.
[5] S. Stas, “Associative processes with CAMs,” in Northcon/93 Conf.
[6] T.-B. Pei and C. Zukowski, “VLSI implementation of routing tables: tries and CAMs,” in Proc. IEEE INFOCOM, vol. 2, 1991, pp. 515–524.
[7] “Putting routing tables in silicon,” IEEE Network Mag., vol. 6, no.1, pp. 42–50, Jan. 1992.
[8] A. J. McAuley and P. Francis, “Fast routing table lookup using CAMs,” in Proc. IEEE INFOCOM, vol. 3, 1993, pp. 1282–1391.
[9] N.-F. Huang, W.-E. Chen, J.-Y. Luo and J.-M. Chen, “Design of multi-field IPv6 packet classifiers uses ternary CAMs,” in Proc. IEEE GLOBECOM, vol. 3, 2001, pp. 1877–1881.
[10] G. Qin, S. Ata, I. Oka, and C. Fujiwara, “Effective bit selection methods for improving performance of packet classifications on IP routers,” in Proc. IEEE GLOBECOM, vol. 2, 2002, pp. 2350–2354.
[11] H. J. Chao, “Next generation routers,” Proc. IEEE, vol. 90, no. 9, pp. 1518–1558, Sep. 2002.
[12] V. Lines, A. Ahmed, P. Ma, and S. Ma, “66 MHz 2.3Mternary dynamic content addressable memory,” in Record IEEE Int. Workshop on Memory Technology, Design and Testing, 2000, pp. 101–105.
[13] R. Panigrahy and S. Sharma, “Sorting and searching using ternary CAMs,” in Proc. Symp. High Performance Interconnects, 2002, pp. 101–106.
[14] “Sorting and searching using ternary CAMs,” IEEE Micro, vol. 23, no. 1, pp. 44–53, Jan.–Feb. 2003.
[15] H. Noda, K. Inoue, M. Kuroiwa, F. Igaue, K. Yamamoto, H. J. Mattausch, T. Koide, A. Amo, A. Hachisuka, S. Soeda, I. Hayashi, F. Morishita, K. Dosaka, K. Arimoto, K. Fujishima, K. Anami, and T. Yoshihara, “Acost-efficient high-performance dynamicTCAMwith pipelined hierarchical search and shift redudancy architecture,” IEEE J. Solid-State Circuits, vol. 40, no. 1, pp. 245–253, Jan. 2005.
[16] J. Brelet, “Using Block Select RAM+ for High-Performance Read/Write CAMs”, Xilinx Application Note 204, 1999.
[17] S. V. Kartalopoulos, “An Associative RAM-based CAM and Its Application to Broad - Band Communications Systems”, IEEE Transactions on Neural Networks, p 1036, vol. 9, 1998.Record, 1993, pp. 161–167. J. Wang, “Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),” IEEE J. Quantum Electron., submitted for publication.