A Variable Word-Width Content Addressable Memory for Fast String Matching

Geir Nilsen, Jim Torresen and Oddvar Søråsen

Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail: {geirni, jimtoer, oddvar}@ifi.uio.no

Abstract:

This work deals with off-loading some time critical parts in the process of performing intrusion detection from software to reconfigurable hardware (FPGA). Signatures of known attacks must typically be compared to high speed network traffic, and string matching becomes a bottleneck. Content Addressable Memories (CAMs) are known to be fast string matchers, but offer little flexibility. For that purpose a Variable Word-Width CAM for fast string matching has been designed and implemented in an FPGA. A typical feature for this CAM is that the length of each word is independent from the others, in contrast to common CAMs where all words have the same length. The design has been functionally tested on a development board for a CAM of size 1822 bytes (128 words). This design processes 8 bits per clock cycle and has a reported maximum clock speed of 100 MHz. This gives a thoughput of 800 Mbit/s.

1. Introduction

The speed of todays networks is of such a kind that a general purpose CPU must struggle to process the network data. The CPU must also have resources left for other application processes. The amount of processing required on network data is increasing due to the need for intrusion detection, cryptographic processing and more [1].

One way to free the CPU from heavy tasks is to convert some of the software or parts of a given software, into hardware. In this work a part of an Intrusion Detection System (IDS), Snort [2], the string matcher will be implemented in hardware.

As time goes by, we can expect that new patterns will be discovered and a reconfiguration of the hardware will then be required. For research purposes it is expected that patterns will change frequently, and that they are of different sizes. A suited hardware technology is Field Programmable Gate Arrays (FPGA) which is easily reconfigured to accommodate these changes.

The main contribution of this paper is to implement a Variable Word-Width Content Addressable Memory (CAM) in FPGA for string matching. Thereby great flexibility is obtained with respect to how the Snort rule set is defined.

In order to design a system for an FPGA, a Hardware Description Language (HDL) is needed. VHDL was chosen for this work in spite of its somewhat restricted programming features. The main reason for this is that VHDL is well known at the University of Oslo. One result from this work is that it is now possible to describe the actual CAM design in VHDL by taking advantage of *all* programming abilities that are common to any programming language. Thereby reconfigurability of the CAM has been made a simple task.

The CAM is implemented in hardware by using *Xilinx ISE 6.1.03i*. With this tool various reports are generated. These reports were be used for giving estimates of how much of the *resources* of an FPGA will be used by each specific design and the *speed* obtained.

Section 2 gives an overview of IDS, whereas the CAM design is described in section 3. The results are summarized in section 4 and the conclusion is given in section 5.

2. Intrusion Detection Systems (IDS)

It turns out that there are certain patterns that occur more often than others in the cases of intrusion. By simulating attacks it is possible to identify patterns that are well suited for detection. The next challenge is to monitor a high speed network looking for these patterns. For this purpose we use IDS.

Snort is a popular Network IDS (NIDS) because it has an open source code and runs under most versions of Linux and Windows. It also offers full control over its rule set configuration [5]. A rule is also known as a signature and may contain a string that must be compared with the contents of an incoming packet. Although Snort has over 1400 rules it is not common to activate all of them at the same time. An IDS that uses 105 rules is presented in [3].

IDS rely on exact string (content) matching [4][5]. String matching based on software has not been able to keep up with high network speeds, and hardware solutions are needed [6]. A CAM may be used for high performance systems, but it is known to offer little or no flexibility [7]. Thus, available CAMs are not suited for implementations with Snort rules. The CAM design presented here is well suited for Snort rules and also solves the limitations reported in [8].

3. Making a String Matcher by using a CAM

A CAM is used to store data, much the same as a RAM. The write mode of CAM and RAM is similar to some degree, but the read mode differs significantly. With RAM we input an *address*, and get *data* out. With CAM we input *data*, and if this data is stored in the CAM, we get the *address* of that data out. There is an address at the output even if there is no match, so with a CAM we need a *Match* bit to indicate if the CAM contains the input data.

A traditional way of describing the size of a CAM is given by "width * words". The width tells the size in bits of one storage location in the CAM, while words give the number of storage locations. The advantage with CAM is that all of its words can be looked up in *parallel*.

Figure 1 illustrates the basic idea of a string matcher. A 4-input AND-gate, with optional inverters on the inputs, is capable of matching any 4-bit string. In this case we could say that we have a CAM of one word that is 4 bits wide. To match a string of *n* bits, we would need an *n*-input AND-gate.

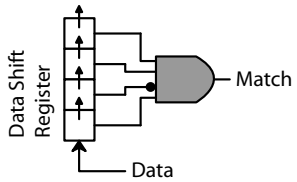

Figure 1: The Basic Idea of a String Matcher by Using an AND Gate

The following properties are desired for the string matching of this work:

1. The length of one string should be independent of the other strings. That is, the CAM should be able to compare strings of different lengths at the same time. As a string has a smallest element of one character (one byte), the smallest element in the CAM should be no less than one byte.
2. The number of words should not be restricted; for example to 2^n. It should be possible to specify the number of strings by any integer.
3. The comparison between an incoming packet and the strings must be fast, preferably around 1 Gbit/s.
4. The time spent for changing the content of the CAM is insignificant, because Snort rules are rarely changed. Still, the possibility of writing to the CAM should be kept in order to make the CAM as flexible as possible.
5. The time spent for making the VHDL code of a CAM should be short. Also, it should not be necessary to go into the details whenever a new CAM is required.
6. Future work: It should be possible to change the number and length of words in a CAM without having to reconfigure the FPGA. This will add further flexibility to the CAM and the previous item would then be eliminated.

Making a CAM where the width of each word is equal is easily achieved in VHDL. The lack of common programming capabilities in VHDL makes it a greater challenge to design a CAM where each word may have any given width. Even more complexity is added if the number of words could be any integer. Many details in the VHDL code describing such a CAM will have to be changed each time a new CAM is required. The solution for this is provided by the following scheme:

Perl → (Generate VHDL source code)
VHDL → (Synthesis tools take care of the remaining steps)
(…) →
FPGA Bitstream

All the details that need to be changed for each possible configuration of a CAM are handled by a programming language. Perl has been chosen for this work, but any other programming language would do just as well. The content of the VHDL CAM files that do not need to be changed are simply stored as text in the Perl script and will be written to files at the appropriate locations. The parts of the VHDL CAM files that need changes are treated as variables in Perl. For a given CAM these variables are calculated and then converted to text before being written to the corresponding VHDL file. In between the variables, the content that does not need changes is written directly to file. The Perl script made in this work is capable of generating VHDL files describing *any* CAM (as described in this paper) in less than one second on a PC with a 466 MHz CPU. The number of words may be given as an integer input to the script. The length of each word may be read from file. The file used for this work is described below.

To obtain a realistic dataset for testing the CAM (by simulation and by hardware), the following choices were made:

1. Make a Perl script to scan all Snort rules for strings that are to be matched.
2. Do not store a string if there are more than one "content" part in the rule; see Figure 2 for an example of a Snort rule with *one* content part.
3. Do not store a string that could generate a multiple match.
4. Store strings that are at least 4 bytes and no more than 32 bytes.
5. Write these strings to file.

There were 1083 Snort rule strings that matched the above criterias.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET
12345:12346 (msg:"BACKDOOR netbus getinfo";
flow:to_server,established; content:"GetInfo|0d|";
reference:arachnids,403; classtype:misc-activity; sid:110;
rev:3;)
```

Figure 2: A Typical Snort Rule

A CAM with variable word-width is better described in bytes, rather than width * words. The amount of logic in a given FPGA is the only limitation for the CAM as specified to the Perl script. It is therefore up to the designer to define the organization of the CAM for the given application/FPGA, when designing a CAM this way.

Virtex (-E / II / II Pro) FPGAs are suited for making logic that is equivalent to wide AND-gates. The basic components used to make a CAM are LookUp Tables (LUTs) configured as shift registers (SRL16Es) and multiplexers (MUXCYs) [10].

By configuring a SRL16E as shown in Figure 3 and disabling the shift, we get an equivalent logic to that shown in Figure 1. By connecting a data shift register to the address bus of the SRL16E, we get a 4-bit CAM [9]. Note that only one '1' has been written to the SRL16E. For this reason, an SRL16E-based CAM consumes a relatively large area in an FPGA. It is typical that a CAM needs more logic per bit than a RAM.

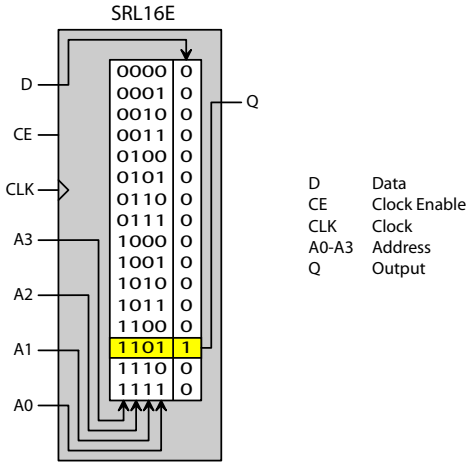

Figure 3: SRL16E Configured as an AND-Gate

The SRL16Es are connected in series through a carry-chain as shown in Figure 4 [10]. Each slice is capable of representing a CAM of 1 byte (two SRL16Es). The number of slices required for a word is therefore equal to the number of bytes in the word. Note that the words must be stored upwards in columns due to the direction of the carry-chain.

By connecting the output (Q) of the SRL16E to the Select (S) input of a MUXCY, we can make a wide AND gate suited for string matching. If S receives a '0', input 0 (which is grounded) of the MUXCY will be selected. Otherwise, the CarryIn (CIN) signal is selected. This signal is then passed on to the next MUXCY. A *Match Enable* signal is connected to the first MUXCY. If only *one* of the MUXCYs receive a '0' on its CIN, the match result will be zero. As seen on the figure, we can now make words (AND gates) of width *n* bytes.

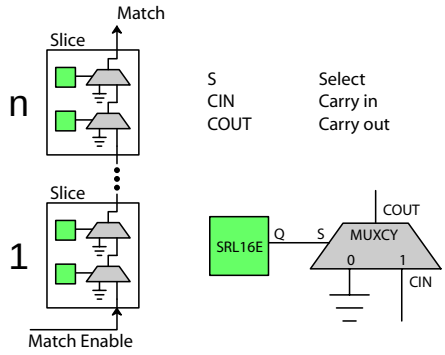

Figure 4: Serial Connection of SRL16Es to Make One CAM-Word

Figure 5 illustrates a (5,9,4,6) byte CAM in read mode. Each location in the *Data Shift Register* is connected to all matching units in the corresponding row as indicated by horizontal lines in the figure. The CAM has a latency of two clock cycles from *Match enable* goes high, until the output registers of the encoder are updated. By giving new data to the shift register each clock cycle, we will get a new valid output for each clock cycle.

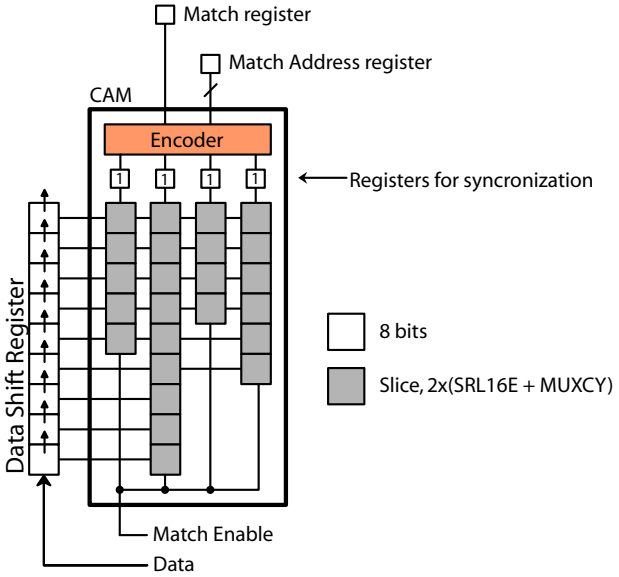

Figure 5: CAM Read Mode

Because there are 16 locations to address in the SRL16E, 16 bits must be shifted in during one write. That is, 16 clock cycles are needed to write *one* word in a CAM. The write to the CAM is of less interest in this work, and details are therefore omitted here.

Figure 6 shows how a CAM has been applied for string matching in this work. The data to be matched is sent to the CAM as a *Byte Stream*. In parallel, it is compared to all strings (i.e. words) stored in the CAM. If a match is found, it is indicated by the *Match* bit. The *Match Address* reports the "address" of the string that matched in the CAM. Exact string matching is performed and thus, only one (or none) string will give a match. The string matcher has not yet been integrated with the Snort program, although this is the intention.

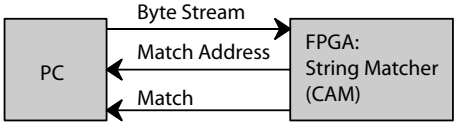

Figure 6: A CAM Applied in a String Matching System

4. Results

Table 1 shows two examples (column 2 and column 4) of CAM-designs that have been functionally tested in hardware. The others are synthesized down to the bitstream that is used for programming the FPGA. The numbers in row 1, 2 and 3 have been rounded down to the nearest integer. As the *Slices Used* increases (row 2) the room for speed optimization decreases. Note that a lower *speed grade* indicates a faster device (-7 is faster than -6, for example).

Max speed of incoming bit stream (Mbit/s)	1032	864	808	808	840	816	768	800	800
Slices Used (%) (Synthesis Report)	10	1	93	13	13	9	27	27	19
Reported Max Speed (MHz) (Post-Place and Route Static Timing Report)	129	108	101	101	105	102	96	100	100
Bytes	134	134	1822	1822	1822	1822	3601	3601	3601
Words	8	8	128	128	128	128	256	256	256
Package	fg456	bf957	fg456	bf957	bf957	ff1517	bf957	bf957	ff1517
Speedgrade	-7	-4	-7	-4	-6	-5	-4	-6	-5
Device	xc2vP7	xc2v6000	xc2vP7	xc2v6000	xc2v6000	xc2v8000	xc2v6000	xc2v6000	xc2v8000

Table 1: A Selection of Some Successfully Implemented Designs

Assuming we get a match for every incoming packet, it is possible to predict the worst case that this string matcher should be able to handle; see the upper row of Table 1. This is an important consideration to make when designing a system for detecting and handling massive attacks.

From column 2 we can see that small CAMs are capable of handling a bitstream of 1 Gbit/s. Column 4 shows an implementation of a 1822 byte (128 word) CAM that is capable of processing data at 800 Mbit/s. The remaining columns shows estimates from other devices.

IDSs based on *software only* can process at most 100 Mbit/s [11]. From the Table 1 we can see that the string matcher presented in this paper can perform about 8 times faster than software IDSs. As the string matcher is often the main bottleneck of an IDS, this design clearly offers an improvement to IDS.

5. Conclusion

A Variable Word-Width CAM has been designed that is well suited for string matching with Snort rules. When taking advantage of a programming tool, the time needed for making a new CAM, described by VHDL, is less than one second on a PC with a 466 MHz CPU. The flexibility of such a CAM, and the short time needed to make the VHDL-files, will be important for NIDSs since they may need to be changed frequently. The implemented architecture functionally tested on the hardware platform that was chosen for this work can process 128 words (1822 bytes) in parallel at 800 Mbit/s. Future work involves making an even more flexible CAM. It should be able to changing the number of words, and the length of each word at runtime. That is, a new CAM could be made without producing new VHDL files.

6. References

1 Peter Bellows et al. *GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing.* FCCM 2002.

2 http://www.snort.org

3 Y. H. Cho et al. *Specialized Hardware for Deep Network Packet Filtering*. FPL 2002.

4 C. Jason Coit et al. *Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort.* In Proc. of DARPA Information Surviability Conference and Exposition, DISCEXII, 2001.

5 B. L. Hutchings et al. *Assisting Network Intrusion Detection with Reconfigurable Hardware.* In Proc. of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. FCCM 2002.

6 S. Dharmapurikar et al. *Deep packet inspection using parallel Bloom filter*. In Proc. of Hot Interconnections 11 (HotI-11), Stanford, CA, 2003.

7 D. E. Tylor et al. *Scalable IP Lookup for Internet Routers*. In IEEE journal on selected areas in communications, Vol. 21, No. 4, May 2003.

8 Shaomeng Li et al. *Exploiting Reconfigurable Hardware for Network Security*. FCCM 2003.

9 J-L Brelet and B. New. *Designing Flexible, Fast CAMs with Virtex Family FPGAs.* Xilinx Application Note 203, September23, 1999 (Version 1.1).

10 Xilinx Virtex-II Pro Platform FPGAs. *Functional Description*. Datasheet ds083

11 M. Gokhale et al. *Granidt: Towards Gigabit Rate Network Intrusion Detection Technology*. FPL 2004.