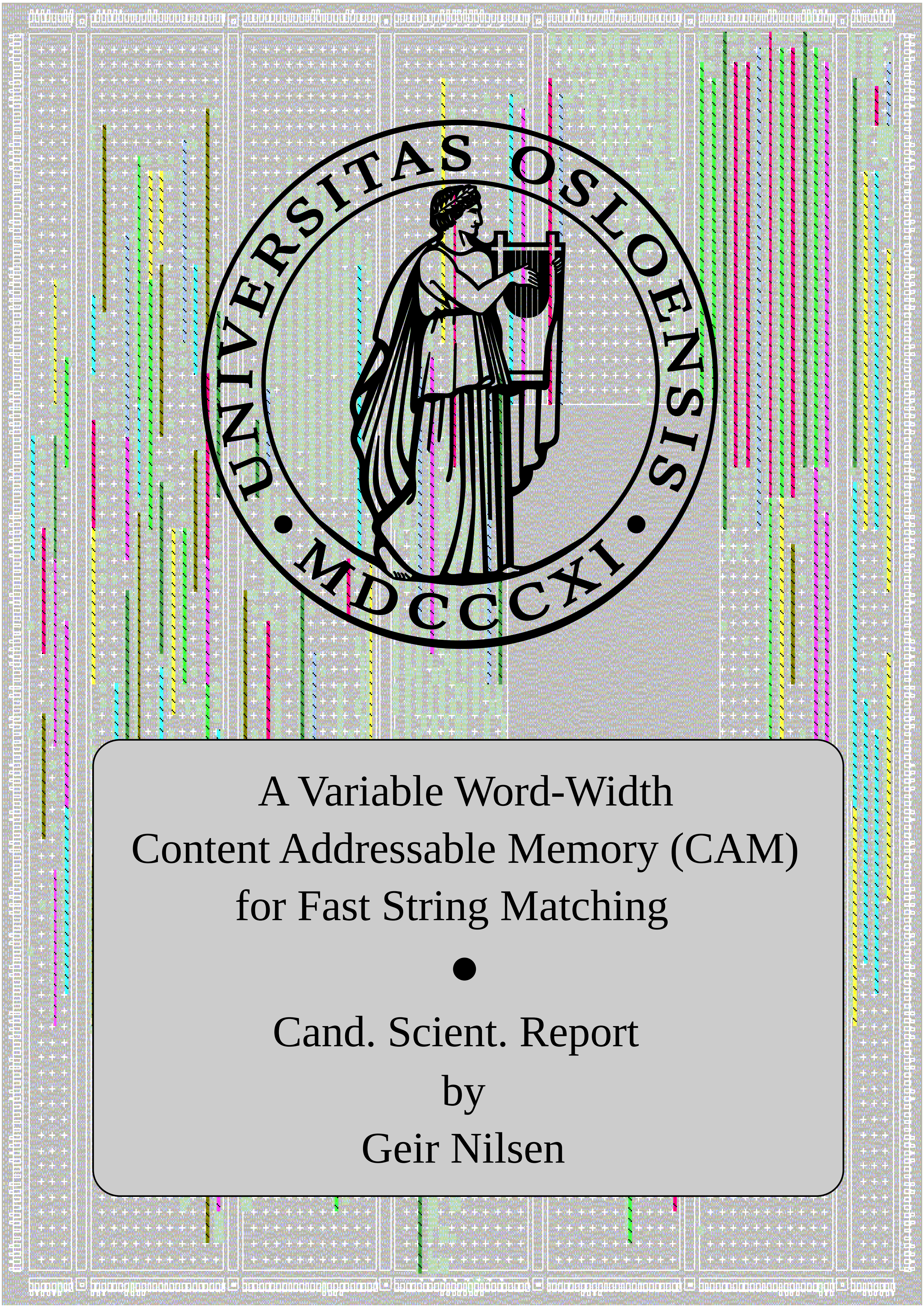


UNIVERSITAS OSLOENSIS • MDCCCXI •

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

●

Cand. Scient. Report

by

Geir Nilsen

Abstract

This work deals with off-loading some 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. To be able to effectively reconfigure the CAM, a software technique has been developed for creating the VHDL code. The CAM design has been simulated with Model Technology ModelSim 5.6f, and synthesized by Xilinx ISE 6.1.03i. It was then loaded into a Virtex-II Pro (P7) FPGA. 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 throughput of 800 Mbit/s. One important part of this work has also been to develop circuits for hardware testing purposes.

Acknowledgements

Unconditional thanks goes to my supervisors *Oddvar Søråsen* and *Jim Torresen*. First of all for accepting me as their student. Then for all the highly intelligent conversations along the way through this project. For every heavy step in the creative process of this project, my courage was always increased after a discussion with my supervisors. These conversations are, without any doubt, the main reason that this project is now completed. All my fellow students and the employees that make up the good working environment we have at this group, deserves a thank also. The time spent here will be remembered as a good time.

Good Luck to you all!!!

Contents:

Figures:

Tables

1 INTRODUCTION

The speed of a network is today 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]. The focus of this project is intrusion detection.

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 for these patterns. For this purpose we use Intrusion Detection Systems (IDS).

Snort [10] is a popular Network IDS (NIDS) because it has an open source and runs under most versions of Linux and Windows. It also offers full control over its rule set configuration [3]. A rule is also known as a signature and may contain a string that must be compared with the contents of an incoming packet.

IDS rely on exact string (content) matching [2][3]. String matching based on software has not been able to keep up with high network speeds, and hardware solutions are needed [4]. Content Addressable Memory (CAM) may be used for high performance systems, but it is known to offer little or no flexibility [5]. Available CAMs are not suited for implementations with Snort rules. Making a more flexible CAM is therefore the primary target of this project. The CAM design presented in this report also solves the limitations of CAM as reported in [6].

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 project a part of Snort, the string matcher, will be implemented in hardware. A part of the open source of Snort will have to be changed and recompiled in order to use the hardware string matcher. The string matcher is then supposed to run in hardware in combination with the recompiled software. Making changes in the source of Snort is beyond the scope of this project.

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. A suited hardware technology for this purpose seems to be Field Programmable Gate Arrays (FPGA). An FPGA is easily reconfigured to accommodate these changes.

In order to make a design that can be used with an FPGA, a Hardware Description Language (HDL) is needed. HDLs known to the author are Vendors HDL (VHDL), Verilog and Handle-C. VHDL is the only HDL from which a first hand knowledge is available. Describing a flexible CAM with a programming language would be of a trivial matter. However, VHDL lacks programming abilities. It is possible that Verilog or Handle-C has the desired programming abilities, but VHDL was still chosen for this project. The main reason for this is that VHDL is well known at the University of Oslo. If there should be any problems with developing the design presented in this report it is more likely that useful hints could be received when using VHDL. The secondary target of this project is therefore to come up with a solution on how to take advantage of *all* programming abilities that are common to any programming language, in relation with VHDL.

The CAM will be simulated with a software called *ModelSim 5.6f.* When the simulation is done and confidence to the design has been achieved, it will be implemented in FPGA hardware. The VHDL-files describing the CAM and the control logic needed to make the CAM work will be implemented in hardware by using *Xilinx ISE 6.1.03i.* With this tool various reports are generated. These reports will be used for giving an estimate of how much of the *resources* in the FPGA that will be used in relation with one specific design. A *maximum performance* estimate will be provided also.

In summary:
The primary target of this project is to make a CAM that is capable of doing string matching with Snort rules. The secondary target of this project is to solve limitations in relation with programming abilities in VHDL.

Chapter 2 introduces IDSs.
Chapter 3 introduces reconfigurable hardware and chooses the hardware platform for this project.
Chapter 4 presents the solution for the primary target.
Chapter 4.2.1 presents the solution for the secondary target.
Chapter 5 illustrates some details necessary to test the solution of the primary target.
Chapter 6 presents the results, and a brief discussion of these.
Chapter 7 gives a conclusion.

2 INTRUSION DETECTION SYSTEMS (IDS)

One event for motivating research on IDS took place February 9, 2000. Amazon.com, E-trade, and other pioneering electronic commerce companies got hit with a distributed denial of service attack that collectively cost several million dollars. This is believed to have changed the nature of electronic commerce for all future by highlighting the importance of effective detection and response in any successful on-line business [7].

2.1 Introduction

Intrusion detection has existed as a research area since the middle of the 1980s. Early systems had bad user interfaces. They were also unable to be used in environments outside those for which they were designed and could monitor a very small number of targets.

As electronic commerce dominates the economic landscape and drives the growth of the Internet, the interconnectivity of computers for businesses has become an important factor for success. We are connected to our partners, suppliers, customers and even our competitors through the Internet. To live with all of these connections, businesses have to develop a degree of trust based on computer security controls. Trust is enhanced by verification that the control system works properly. Verification is provided by intrusion detection.

The disciplines of computer security address three fundamental needs: *Prevention*, *Detection* and *Response*. They are all important for a reliable protection. However, for the last 30 years, most of the resources have been spent on *prevention*. The attempt has been to prevent threats so that *detection* and *response* would not be necessary. Unfortunately, prevention methods have not been able to give an acceptable level of security [7].

2.2 Categories of IDS

Three categories of intrusion detection technologies are host, network and distributed intrusion detection [8]. Common to all intrusion detection technologies is that they are based on analyzing a set of discrete, time-sequenced events for patterns of misuse [7].

2.2.1 Network-Based IDS (NIDS)

Intrusion detection is network-based when the system is used to analyze network packets. Network packets can be derived from the output of routers and switches, but they are usually *sniffed* off the network. The most common protocol targeted in commercial products is TCP/IP, but others may be available [7]. Figure 2.1 shows an example NIDS [8].

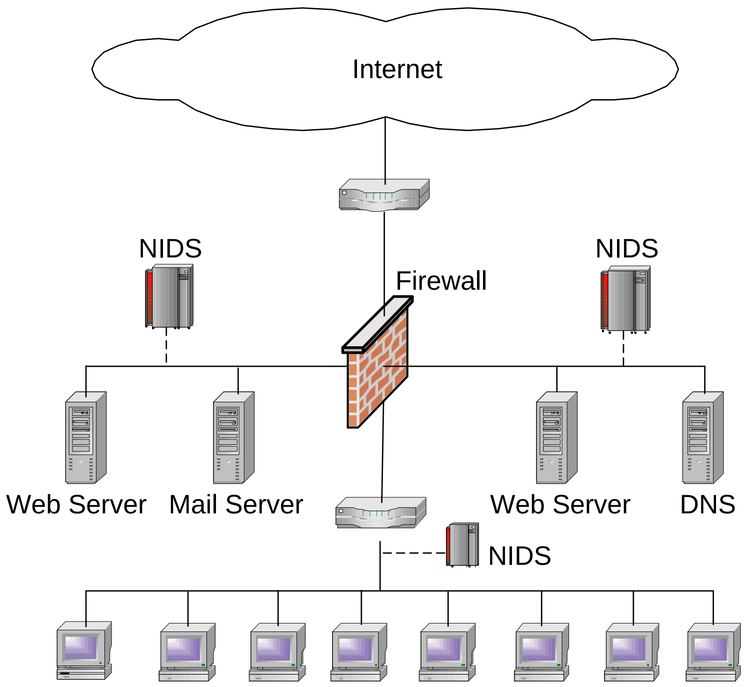


Figure 2.1: An Example of a NIDS[8]

The following attacks are some of the most common ones related to network traffic. Most network-based attacks are directed at operating system vulnerabilities. In most cases these attacks would be impossible to detect with host-based technologies [7].

Denial of Service (DoS)

DoS attacks are named so because they result in a resource not being available to service its users. DoS attacks come in many forms and different levels of severity. Insiders can cause DoS attacks as well as outsiders, but these types of attacks usually leave many clues, so malicious DoS attacks are usually initiated by anonymous outsiders. The packets that deliver the attack usually carry many characteristics that can be detected with a NIDS, thereby making it an effective tool for detecting these attacks.

Packet flooding is a simple DoS technique that involves sending as many packets as you can to a single network device. This is done until the device either crashes because it can’t handle the load or becomes so slow that legitimate user requests can’t get through. This is not a very sophisticated attack, and it is easy to detect. Defending is done by denying access to the source computer sending the packets. However, if the attacker is spoofing the source address, it may be very hard to find out were the packets are coming from [7]. Spoofing is the creation of TCP/IP packets by using somebody else's IP address. Routers use the *Destination IP address* in order to forward packets through the Internet, but ignore the *Source IP address*, which is only used by the destination machine when it responds back to the source [9].

A special case of packet flooding is the *Distributed DoS* (DDoS) attack in which a number of computers are used to attack at the same time. Defending against DDoS can be difficult if the IP addresses are spoofed. NIDS is not a perfect tool for this type of attack, but it is still vital in both detection and response [7].

There are a number of DoS tools that uses a technique with malformed packets. They are available on the Internet and go by names like *land*, *bank* and *bink*. Malformed packets come in a variety of shapes and sizes with intent of causing a protocol stack to crash. Network protocols are made of complicated pieces of code, and it is difficult to handle all the different types of error conditions that can arise. In most cases, programmers do not attempt to handle impossible situations such as null arguments in critical fields. Hackers take advantage of this by creating null arguments in these fields, causing the protocol to fail. Results of doing so range from hung networks to machines that crash.

Unauthorized Access

Unauthorized password file downloads gives attackers the ability to compromise other systems. This is one of the traditional data thefts that a NIDS is capable of detecting. The *Network Security Monitor*, one of the first NIDS available, looked for the pattern "/etc/passwd" in FTP traffic from the outside of the network. It was simple but effective.

Outsiders rarely break into intended targets from their home machines. Usually there is a complicated path via computers that have been hacked, leading from the attacker's machine to the intended target of misuse. Also, once a computer has been compromised it usually contains information that opens up several other computers within the same organization. These types of attacks are identifiable by the patterns of traffic leading out of the network.

Unauthorized access occurs when outsiders come in over the network and log into a system uninvited. Once they have logged in they can be tracked more effectively with a Host-Based IDS (HIDS). The goal in this case is to detect the outsider with a NIDS before access is given, or in the process of giving access. Unauthorized access over the network should not be possible. Unfortunately, the tools and programs used to share resources and information over networks has a number of security vulnerabilities that can be exploited to allow access. Many older programs were not designed with any security in mind, leaving them wide open for abuse [7].

Theft

There are countries that have trained cyber spies who steal data by committing industrial espionage against other nations. There have also been cases of freelance information brokers who steal information and sell it to the highest bidder.

Another type of theft is stealing bandwidth and disk storage. Big companies tend to have a lot of bandwidth that may not be used at all times. Clever attackers will take over a machine and run whole businesses from networks they do not own. Attackers usually get caught when their business gets too popular and the traffic becomes noticeable [7].

2.2.2 Host-Based IDS (HIDS)

Intrusion detection is host-based when the system is used to analyze data that originates on computers (hosts), such as application and operating system event logs [7]. Figure 2.2 shows an example of a HIDS. A HIDS protects only *the one* computer at which it operates. This is a major difference from NIDS [8].

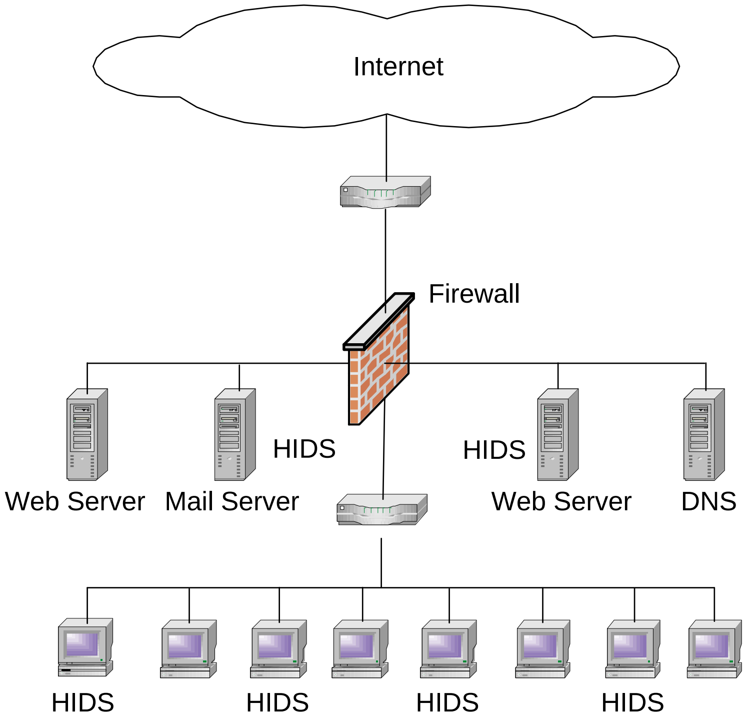


Figure 2.2: An Example of a HIDS[8]

Despite the popularity of NIDS, host-based monitoring is becoming more important because of the threat from the insider. Each of the following examples represents a measurable loss and would be nearly impossible to detect with a NIDS [7].

Security Hazards

Abuse of privilege is when a user has root, administrative or some other privilege and uses it in an unauthorized manner. The distribution of privileges in a system is a security risk by itself. A HIDS is operating on the host where the privileges are granted to the user.

Procedures usually exist to request, document, and create new accounts. Still, administrators have the ability to create accounts without going through these procedures. For example, while installing a software package, the instructions may suggest that the software agent needs an account added to operate successfully. Most administrators will just add the account using their privilege without going through the formal procedures. It is now an undocumented account on the system that only the administrator knows about. If the administrator has to leave the company and his accounts were to be disabled as he walked out the door, there would still be one active account left.

Most organizations have policies in operation to delete or disable accounts when individuals leave. But these procedures may take time, leaving a possibility for the ex-employee to access the account for still some time.

Sometimes contractors get elevated privileges. This usually happens when an administrator gives a contractor elevated privileges to install an application. Most security policies restrict non-employees from having root or administrator privileges, but sometimes it is easier to elevate the user and then reduce privileges later. The security hazard here is that the administrator easily forgets to remove the privileges.

In large companies the lock of screen savers keeps sensitive data safe when people get up from their desks for a short period of time. As a result, many security policies require that the lock of screen savers should be enabled. Unfortunately, it can be annoying if every time you turn around you have to type your password again. This is why many choose to disable the lock. A HIDS can be used to detect users who turn off their screen lock [7].

Changing Contents of Data

Some hacks, for example against government agencies, can result in nude pictures and uncomplimentary remarks posted to their Web sites. Although these attacks originate outside the network, they are executed on the machine itself through the hard disk. This does not always mean that there has been a login. If the NIDS set up to protect the Web site does not detect an unauthorized change on the site, a HIDS is the only way to determine that your Web site is now insulting your customers rather than inviting them in.

A system hastily rebuilt can end up with the registry open to the network. In early versions of Windows NT the default state of the system was to have the registry open to the network. This has been corrected in later versions, but it is still wise to monitor for any default configurations that are considered insecure [7].

Theft

Personnel records are a significant concern of responsibility. Unauthorized release of personal records of any kind, including medical records, can result in lawsuits. All accesses to sensitive records should be monitored by a HIDS.

Observing the access patterns to selected files can indicate users who are scanning the network for interesting information. The net result of these attacks can be very minor, such as a user pushing the limits of his privilege to gather information for a proposal. They can also be very severe, such as an information broker or any other person conducting industrial espionage [7].

2.2.3 Distributed IDS (DIDS)

Groups of IDSs functioning as remote sensors and reporting to a central management station are known as DIDS. In Figure 2.3 we can see a DIDS with *four sensors* and a *central management station*. The individual sensors in a DIDS can be NIDS, HIDS or a combination of both. The rules for each sensor can be chosen independently from the others. Alerts are forwarded to the central management station, thereby notifying the administrator. Common to all DIDSs are that the distributed sensors report to a central management station [8].

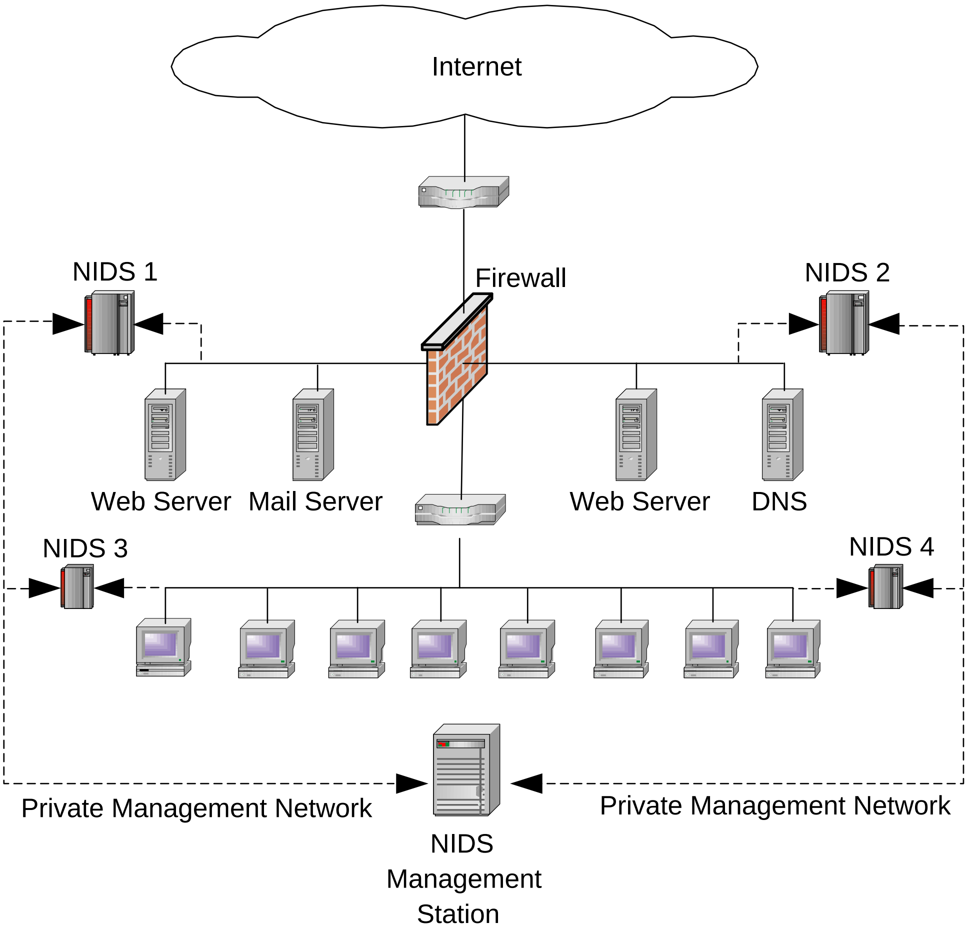


Figure 2.3: An Example of a DIDS[8]

2.3 Signatures

Signatures are deterministic because they identify patterns that are predefined. This makes signatures an interesting field of research. Signatures are also known as rules or rule-based systems. When rules are triggered, an alarm is generated, a response is executed, a notification is sent, or some other action takes place. The characteristics that make up a good rule mechanism are customizability and ease of use [7].

2.3.1 NIDS Signatures

NIDS signatures have two basic forms; patterns within the packet contents and patterns within the header information. Encryption eliminates the ability to see the packet contents. If the system being monitored uses encryption, header analysis is regarded to be the most reliable choice.

Packet Content Signatures are basically *string matches* with the packet contents; Chapter 4. Packet contents, also known as payload, are the data of the network packet that is being communicated from the source to the destination machines. Content signatures are the most common and provide the greatest detail in detection.

FTP Site Execution is an attempt to execute programs on the FTP server during an FTP session. Executing programs remotely that lie outside the FTP root directory is an activity commonly used to access privileged resources. In general, a computer that allows FTP access should not allow FTP site executions.

Packet Header (Traffic) Analysis is a method to detect suspicious network activity without needing to look at the packet contents. Packet headers include the routing information for the packet. There is a surprisingly large amount of detection information that may be derived by using traffic analysis.

Broadcasts are a class of attacks that causes machines to crash. Sending a packet to a system with the source and destination fields identical will cause the protocol stack to fail in most IP implementations [7].

2.3.2 HIDS Signatures

Signature recognition is the most common detection mechanism in a HIDS. An administrator of the network will define which signatures that is of interest.

HIDS signatures are rules that define a sequence of events and a set of transitions between the events. Noteworthy activities may not necessarily be considered misuse or an intrusion because they may be used for other reasons. There are several types of signatures available, including *single-event*, *multi-event* and *multi-host*.

Ninety percent of HIDS signatures are *single-event*. This is because the most interesting activities can be represented in single events. However, single event signatures should not be considered simple just because there is only one event. There are many fields in a single event, and the combinations of field data can be as complicated as multiple event signatures. From a security point of view, executable files are not often written. This usually happens during controlled software updates and other scheduled administrative activities. Attackers who plant trojan horses and viruses that are infecting executable files are detectable with a simple single event signature.

Multi-event signatures are sequences that include two ore more events and a set of transitions between the events. One simple example of a multi-event signature is *Three Failed Logins* which is based on password guessing. Although this attack is relatively low tech, it can still be very effective because there are always users that make poor password choices. It is rumored that half the passwords in the Dallas area are some derivate of “Cowboys”. This is a relevant signature because password guessing is still common. The *Three Failed Logins* signature will create many alerts. An administrator account should make this particular alert more interesting than a normal account. One way to tune this signature is to specify “administrator” in the signature definition so that only failed logins from an administrator will trigger.

Multi-host signatures are signatures that are an aggregation of events from multiple hosts that indicate a noteworthy action. Multi-host signatures are useful for detecting stealth attacks. Stealth attacks are when an attacker does only a little bit of an attack on each machine in order to stay “under the radar” of the IDS. There are a number of challenges related to implementing and configuring multi-host signatures. For example, consider a network where you would have to correlate data from Solaris, HP/UX, Windows NT/2000 and Netware to look for this type of attack. It is obviously a challenge to make these signatures work, but a good IDS should be able to do so anyway [7].

2.4 Detection Mechanisms

IDS technologies offer both *signature* and *statistical anomaly detection*. *Artificial intelligence* (AI) and *metalanguage* have been used in research systems but is not commercially available [7].

2.4.1 Signature Detection Mechanisms

Although there is a rich set of signature types, the administrator must be conservative in establishing rule sets for a network because too many signatures will result in poor performance and lower manageability. Most commercial IDSs are delivered with predefined signatures. The administrator can then choose to customize some of the standard rules, or even create new rules from scratch [7].

2.4.2 Statistical Analysis

Statistics only reflect behavior, not definitive activity. The nondeterministic nature of statistical models makes them most useful in assisting an administrator with broad investigations.

Statistical analysis has a long history in IDS. The first IDSs were designed to automatically distinguish users from each other by using statistical behavior models. This was originally known as *automated anomaly detection* and was intended to detect users who pretended to be other users by logging into somebody else’s account. The early systems scratched the surface of this capability and even showed some level of success.

Statistical analysis provides some of the most powerful features in intrusion detection, but there is a value in these detection models only if their use is kept in perspective. Effectively identifying users by their behavioral characteristics will probably never be possible. Statistics can assist an operator in detecting misuse but they are not very effective as automated detection mechanisms [7].

2.4.3 Metalanguage

Metalanguage is a special case of a rule set that typically consists of thousands of rules that describe the behavior of a user or system. Misuse is detected through combinations of rule triggering that indicate behavior outside normal behavior patterns. Metalanguage is interesting because it uses a rule-based technology to perform a task that is usually reserved for statistical methods [7].

2.4.4 Artificial Intelligence (AI)

A computer is said to have AI if a program running on it is made in such a manner that it can be said to have similarities to the human thinking processes. The goal in applying AI to the intrusion detection problems is to automate the correlation processes that a human brain can perform much better than any computer.

We can differ between strong and weak AI:

- Strong AI
  - Claim that computers can be made to think just like human beings do. More precisely said there is a class of computer programs such that any implementation of such a program is really thinking.
- Weak AI
  - Claim that computers are important tools in the modeling and simulation activity.

This differentiation puts expert systems and statistical models in the weak AI category. Neural networks are the best candidates for strong AI. Neural networks were first used in IDS in the late 1980s [7].

2.5 A Lightweight NIDS called Snort

One popular type of NIDS is a manual router and firewall log analysis and the use of a shareware package called *Snort* developed by Martin Roesch [10]. Snort is a packet sniffer/logger that can be used as a lightweight NIDS. It features rules-based logging and can perform protocol analysis and content searching/matching. A variety of attacks can be detected also. Snort has a real-time alerting capability, with alerts being sent to a separate *alert file* [7].

As an aside, the name Snort came from the fact that the application is a *sniffer and more*. That is, the application *snorts* also (packet logging). Also, Roesch felt that he had too many programs called “a.out”, and that all the popular names for sniffers called TCP-something were already taken.

Figure 2.4 gives an overview of Snort. A packet *Sniffer* is a device used to tap into networks. This is similar to a telephone wiretap. The effect of this is that the entire communication being tapped can be monitored and logged. A simple way of preventing anyone from retrieving information out of network packets, is to use encryption; Chapter 2.3.1. The packets can still be monitored, but the encrypted data will be useless without the proper decryption key.

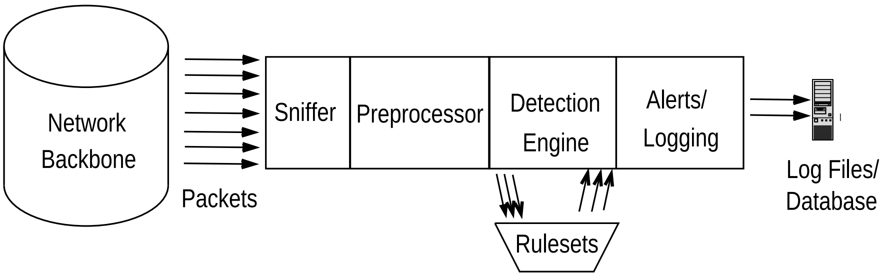


Figure 2.4: Snort Architecture[8]

After having packets sniffed off the network, they are passed on to the *Preprocessor*; see Figure 2.5. The Preprocessor reassembles the packets. By using *Plug-ins,* it then determines which protocol that has been used. Only packets identified by such a plug-in are passed on to the *Detection Engine*. As an example, if you don’t want Remote Procedure Calls (RPC) packets sent to the detection engine, then simply remove the RPC Plug-in.

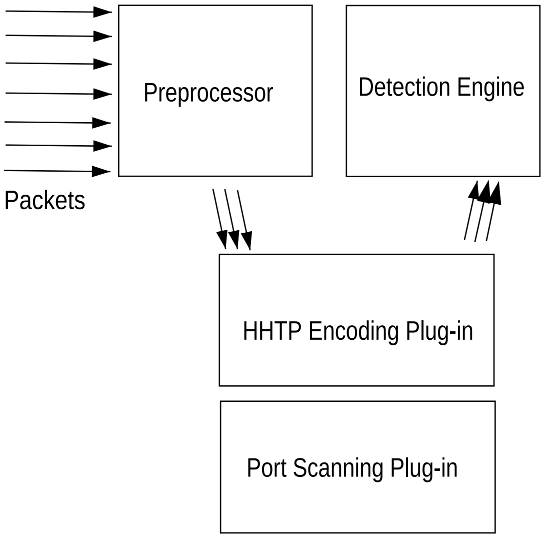


Figure 2.5: The Preprocessor of Snort[8]

Figure 2.6 gives an overview of the Detection Engine. The packet is checked according to a set of *rules*. These rules may include strings that must be compared with the packet content; treated in Chapter 4. If the packet matches a rule, an action will be taken as indicated by the *Logging/Alert* component; see Figure 2.7 [8].

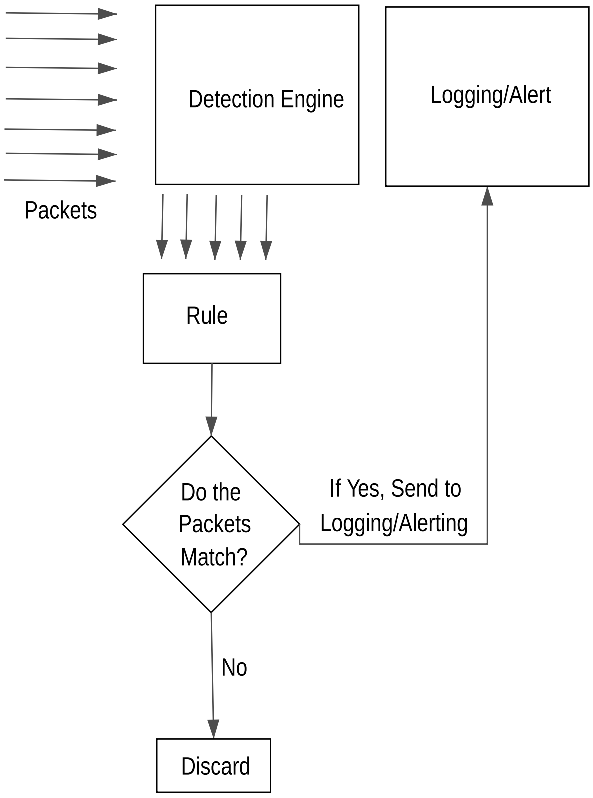


Figure 2.6: The Detection Engine of Snort[8]

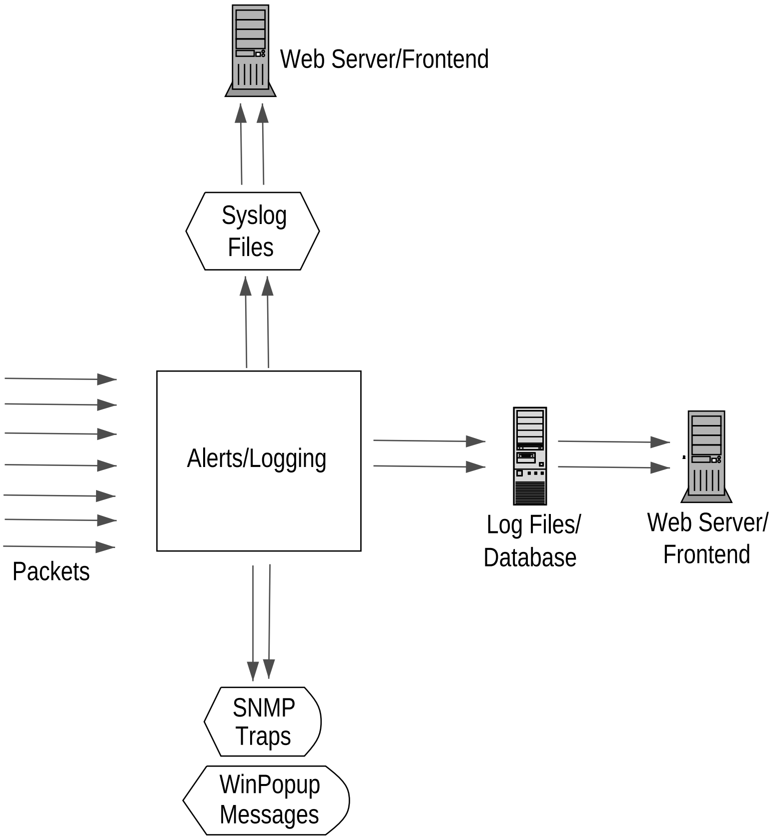


Figure 2.7: The Alerting Component in Snort[8]

String matching, as described in the previous paragraph, is one of the main bottlenecks when running Snort in software. It would therefore be advantageous to implement the string matching part of Snort in hardware. More specifically, reconfigurable hardware is suited for this purpose because the strings will change over time.

A hardware implementation that *scans* the contents of packets (strings) has been implemented in [11]. The hardware chosen for this is an FPGA. The implementation has been combined with other modules such as CAM. The scanner receives 32 bits of data per clock cycle, but can process only 8 bits of data per clock cycle. *One* scanner can operate at 37 MHz. Thus, it can check an input data stream at speed *8 bits x 37 MHz = 296 Mbits/s*. By running four of these scanners in parallel, the entire input data of 32 bits can be processed each clock cycle. This gives a throughput of *4 x 296 Mbit/s = 1.184 Gbit/s*. *Regular expressions* have been used in this solution. Such expressions give a capability of storing more data per byte than *exact string matching*, for example by using wildcards such as ‘*’, ‘?’ etc. Regular expressions are not a topic of this project. A CAM capable of doing exact string matching is presented in Chapter 4.

A string matcher that searches through the content part (strings) of *all* Snort rules has been developed in [3]; see Figure 4.1 for a typical Snort rule. These strings are then converted into a regular expression that matches all the strings. An FPGA has been used to implement this string matcher, and it exceeds the performance of a system based on software by 600x for large patterns. For a small pattern of 47 bytes the hardware throughput was 862 KByte/s, (6.8 Mbit/s) while the software throughput was 884 KByte/s. For a large pattern of 4971 bytes the hardware throughput was 784 KByte/s, while the software throughput was 1.72 KByte/s. As can be seen from these two examples are that the larger pattern we have, the more advantageous it is to implement a hardware string matcher.

3 RECONFIGURABLE HARDWARE

Reconfigurable hardware is still a young field of research. Although Gerald Estrin of the University of California at Los Angeles proposed reconfigurable hardware in the late 1960s, the first demonstrations did not occur until the middle of the 1980s [12].

3.1 Brief History

To understand how the reconfigurable circuits that are in use today works, we must take a short tour through history. We will start in the early 1970s by taking a look at *Programmable Logic Arrays* (PLA) before continuing with *Programmable Array Logic* (PAL) and *Programmable Logic Devices* (PLD). These devices are commonly called *Simple Programmable Logic Devices* (SPLDs). We then end up with the two most common categories of reconfigurable devices that are in use today; the *Complex Programmable Logic Device* (CPLD) and the *Field Programmable Gate Arrays* (FPGA). These devices are collectively called *Field Programmable Logic Devices* (FPLDs) [13].

It is natural to ask how hardware devices can be electronically programmed to perform any possible logic function. These devices evolved from the PLA devices of the early 1970s. The basic PLA structure is shown in Figure 3.1. It consists of a layer of *AND gates* succeeded by a layer of *OR gates*, interconnected through programmable switch arrays. In the PLA, every input and its logical inversion is passed into an *AND array* on the *horizontal wires*. The *vertical wires* in the AND array are inputs to a row of *AND gates*. The AND gates receive input signals by tying the horizontal and vertical wires together as illustrated by the *black dots*. Thus, in Figure 3.1, the leftmost AND gate receives the *logical inverse of the C signal* and ANDs it with the *A signal*.

The *OR array* has a function similar to the AND array. The vertical wires are outputs from the AND gates into the OR array. There they can be connected to the horizontal wires, which are inputs to a *column of OR gates*. By connecting the outputs of the AND gates to the inputs of the OR gates, a *sum of products* can be created at each output of the PLA.

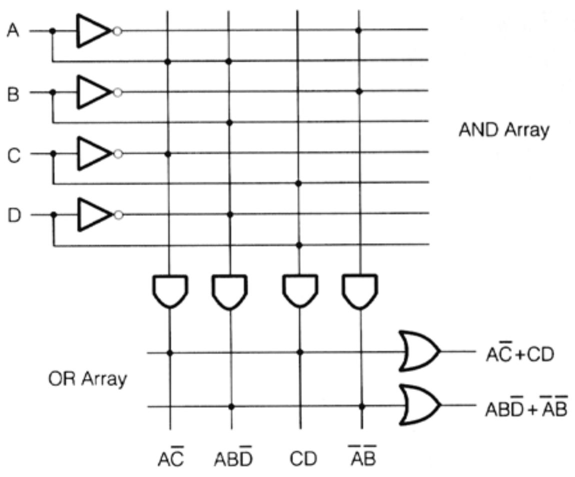


Figure 3.1: PLA [13]

The flexibility provided by both a programmable AND and OR array often went unused, so engineers came up with the simpler PAL structure; Figure 3.2. The programmable OR array were replaced with a set of fixed connections from *AND gates* into the *OR gates*. The PAL also outputs *feedback* into the AND array. The feedback terms are used to build multilevel logic functions. Thus, you can program the switches to form any product term you want. In Figure 3.2 the output from each OR gate is fixed to be the sum of two product terms.

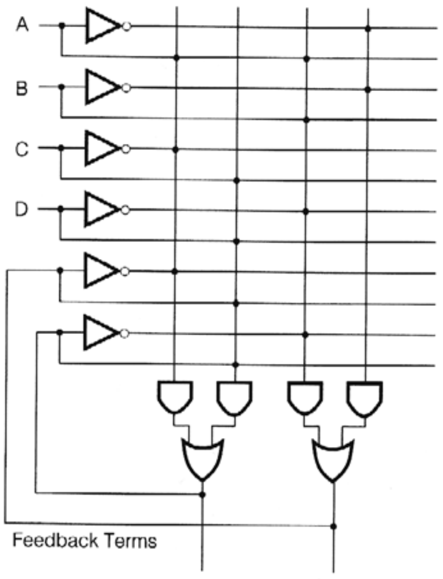


Figure 3.2: PAL [13]

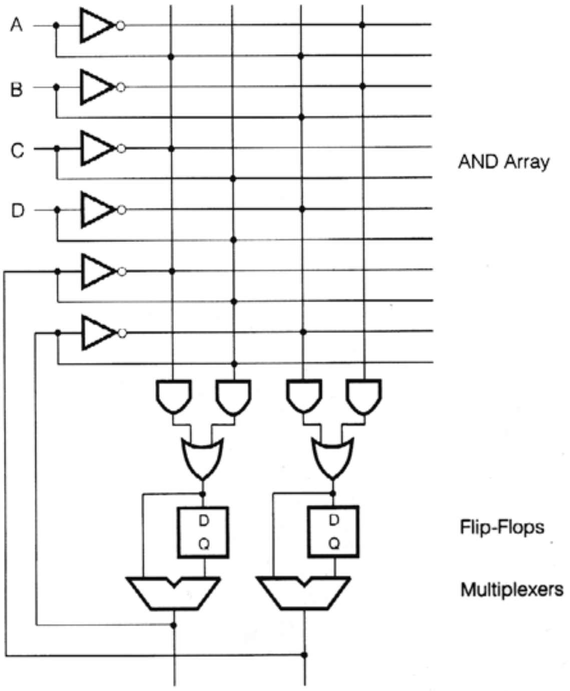


Figure 3.3: PLD [13]

PALs and PLAs are good for combinational logic, but they cannot be used for sequential logic without adding external *flip-flops*. So flip-flops were added to the PAL structure; Figure 3.3. This circuit is called a PLD. *Multiplexers* were added to each output in order to select either the flip-flop output or the combinational output as the actual output. The AND gates, OR gates, flip-flops and multiplexers that drive each output are collectively known as a *macrocell* (Macrocells are used in CPLDs which will be presented below). Modern PLDs have a variety of programmable circuit structures with many options that can be enabled to increase the usefulness of these devices.

The PLAs, PALs and PLDs had to be placed on a Printed Circuit Board (PCB) and then wired to each other and other components. The PLDs could be replaced if small errors were found on the PCB. However, large errors could only be corrected by manually changing the wiring pattern on the PCB. Another disadvantage with these devices is that they can be programmed only once. By combining several PLDs into a single device, it was possible to create CPLDs. An alternative architecture was used to construct the FPGA. This solved the problems related to PCBs.

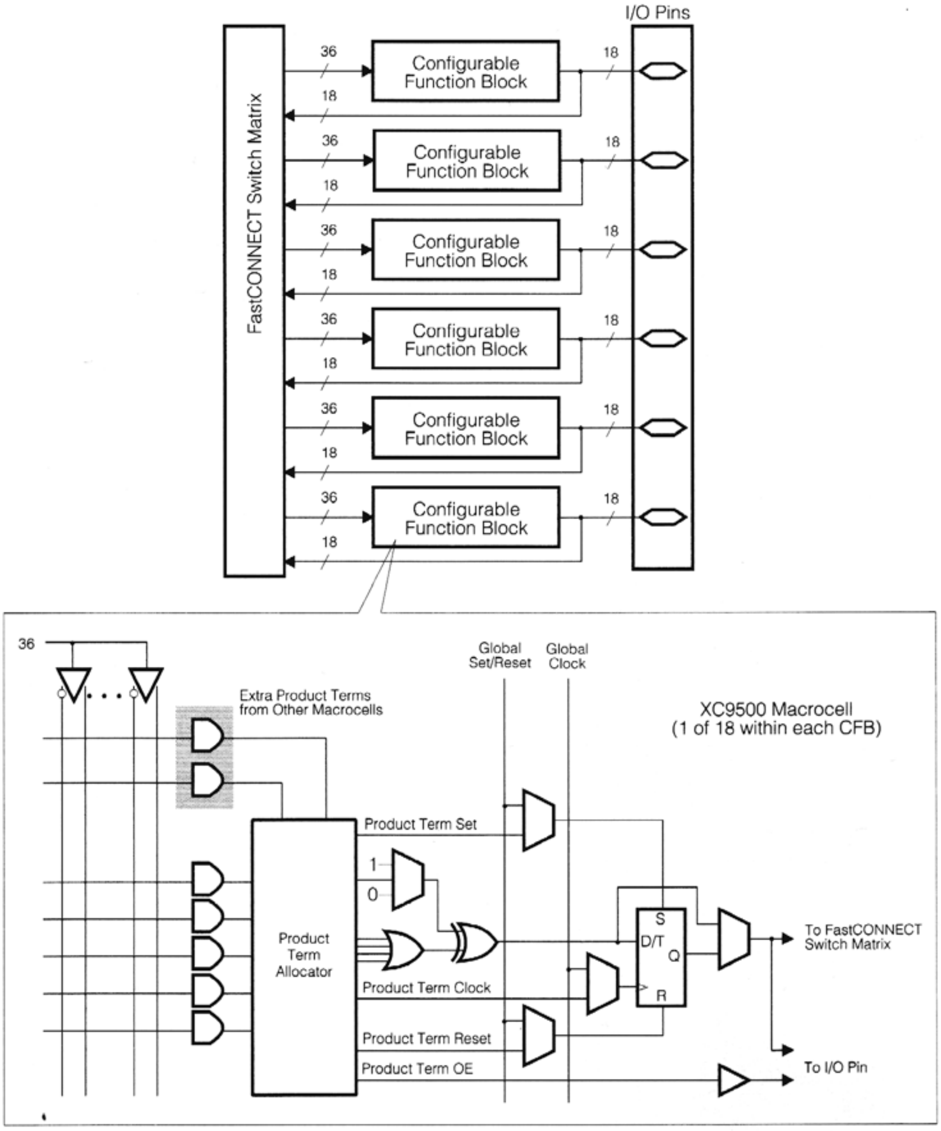
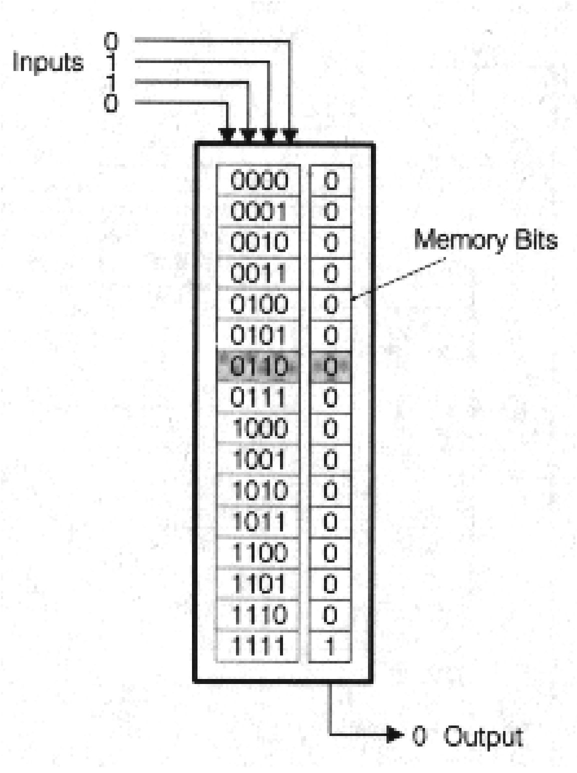

I/O Pins
FastCONNECT Switch Matrix
36
18
Configurable Function Block
36
Extra Product Terms from Other Macrocells
Global Set/Reset
Global Clock
XC9500 Macrocell (1 of 18 within each CFB)
Product Term Set
1
0
Product Term Allocator
Product Term Clock
Product Term Reset
Product Term OE
S
D/T
Q
R
To FastCONNECT Switch Matrix
To I/O Pin

Figure 3.4: XC95108 CPLD [13]

The Xilinx XC9500 series of CPLDs is an example of such a CPLD; Figure 3.4. For example the XC95108 contains *six Configurable Function Blocks* (CFBs); upper half of Figure 3.4. Each of these CFBs is equivalent to an *18-macrocell PLD* with 36 inputs and 18 outputs. The bottom half of Figure 3.4 shows *one* of these 18 macrocells. Each macrocell is connected to an I/O pin on the chip. Complex multilevel logic functions can be built by programming the individual logic functions of each macrocell in each CFB and connecting them through a switch matrix. The result is a design where each pin on a CPLD is driven by a macrocell that implements a wide logic function of a combination of many inputs. The CPLDs use nonvolatile FLASH-based storage cells so the device retains its programming even if the power is turned off.

The FPGAs employ Static RAM (SRAM) storage cells so they need to be reprogrammed each time power is interrupted. The basic building block for the FPGA is the *LookUp Table* (LUT); Figure 3.5. A typical LUT has four inputs and one output. It has a memory containing 16 bits. Applying a binary combination to the *inputs* (such as "*0110*") will match the address of a particular *memory bit* and make it *output* its value. Any four-input logic function can be built by programming the LUT memory with the appropriate bits. For example, a four-input AND gate is made by loading the entire memory with '0's except for a '1' that is placed in the cell that is activated when all the inputs are '1', as is done in Figure 3.5.



0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	0
0111	0
1000	0
1001	0
1010	0
1011	0
1100	0
1101	0
1110	0
1111	1

Figure 3.5: LUT [13]

In FPGAs such as the Xilinx XC4000 series, *three LUTs* are combined with *two flip-flops* and some additional steering circuitry to form a *Configurable Logic Block* (CLB); Figure 3.6. Then the CLBs are arranged in an array with *Programmable Switch Matrices* (PSMs) between the CLBs; Figure 3.7. The PSMs are used to route outputs from neighboring CLBs to the inputs of a CLB. The FPGA I/O pins can be attached to the PSMs and CLBs. Most FPGAs have a lot more CLBs than I/O pins. Thus, each CLB cannot have a direct connection to the outside world, as is the case with macrocells in a CPLD.

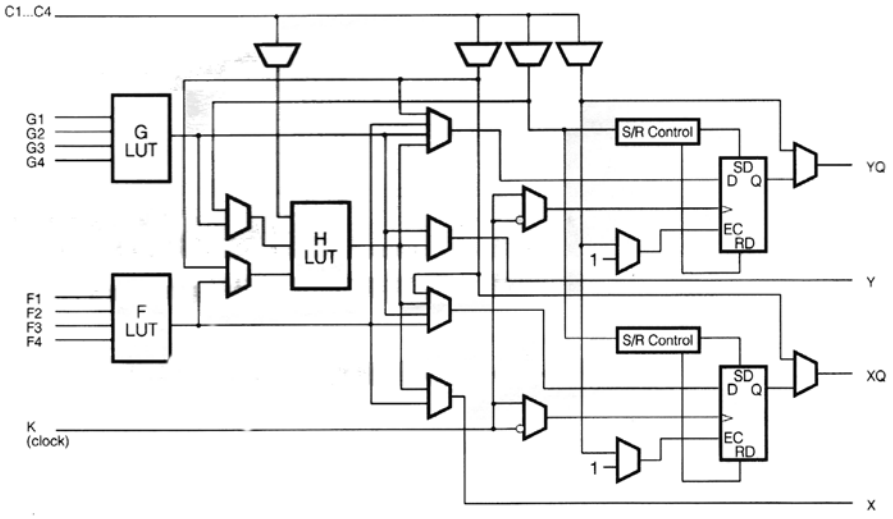


Figure 3.6: XC4000 CLB [13]

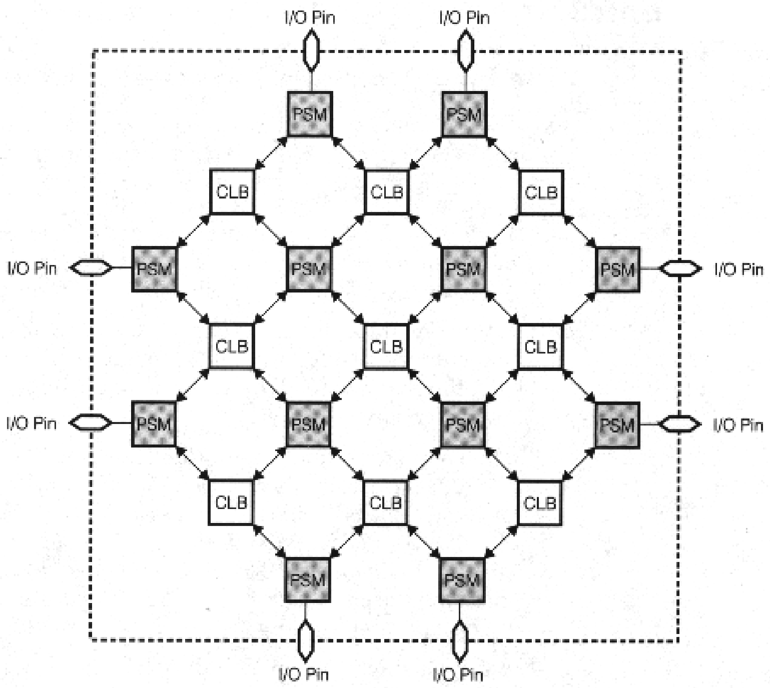


Figure 3.7: A Generic FPGA Architecture [13]

All the wiring in FPLDs is internal to the device, so there is no way an engineer can physically change any connections. Instead, the connections are programmed electrically. In SPLDs the switch arrays are manufactured with fuses at every cross points such that every input is connected to each logic gate. A “burner” is used to program an SPLD. High voltages are set on selected vertical and horizontal wires. The high voltage burns out the fuse at the cross point between the two wires. This operation is performed until all the unwanted connections are burned out. At the end of the process, only the connections needed to build the desired logic functions are left.

The disadvantage with fuses is that once they are blown, they stay blown. When a bug is found, the programmable device has to be discarded and a new one must be programmed. It is more convenient if the connections can be erased and reprogrammed. This is a major advantage of the CPLDs and FPGAs. They contain reprogrammable switches where the fuses would normally be. Each switch is controlled by a storage element that records whether the attached switch is opened or closed. Changing the values in these storage elements changes the state of the switches and alters the functions of the programmable device. These switches can be repeatedly programmed to implement new designs, or repair faulty designs. This is eliminating the need to buy a new device for each design modification. [13]

A brief overview of (re)programmable devices is given in Table 3.1 [14].

				Vendor
FPLD	SPLD	PAL	Programmable Array Logic	Vantis
		GAL	Generic Array Logic	Lattice
		PLA	Programmable Logic Array	
		PLD	Programmable Logic Device	
		The smallest and cheapest way of programmable logic. Programming is done by fuses or non-volatile memory like EPROM, EEPROM or FLASH.		
	CPLD	EPLD	Erasable PLD	
		PEEL		
		EEPLD	Electrically EPLD	
		MAX	Multiple Array matriX	Altera
		A typical CPLD has 2 to 64 times as much logic as an SPLD. Programming is done by non-volatile memory like EPROM, EEPROM or FLASH.		
	FPGA	LCA	Logic Cell Array	Xilinx
		pASIC	programmable ASIC	
		FLEX, APEX		Altera
		ACT		Actel
		ORCA		Lucent
		Virtex		Xilinx
		pASIC		QuickLogic
		Typically offers more logic than a CPLD. Programming is done by Static RAM (SRAM) or antifuses.		
	CSoC		Configurable System-on Chip	

Table 3.1: Summary of (Re)Programmable Devices

3.2 The Choice of a Reconfigurable Hardware Platform

As stated in Chapter 1, reconfigurable hardware is well suited for this project. In Chapter 3.1, we could see that the most common types of reconfigurable hardware are CPLDs and FPGAs. The most outstanding advantage of using an FPLD is its ability for parallel processing. The advantage of this ability in relation with a CAM will become obvious in Chapter 4.

In practice we can say that an FPGA can be reconfigured an infinite number of times and that it is capable of being programmed with far more complex designs than a CPLD. A CPLD can typically be configured a limited number of times. For a large design an FPGA has far better resources for parallel processing than a CPLD. A CPLD on the other hand has far better resources than an FPGA when implementing large boolean expressions, but this is not needed for this project. An FPGA was therefore chosen for this project.

The FPGA for this project, and the board it is attached to, was chosen by the following criteria:

- Debugging possibilities must be available so that prototypes can be developed fast. A Development Board was chosen for this purpose; see Figure 5.1.
- An integrated CPU should be present for future use. This CPU must be able of running Snort software; Chapter 1.
- The possibility of fast communication with other devices must be present. Xilinx offers FPGA with Rocket I/Os capable of baud rates from 600 Mbit/s to 3.125 Gbit/s.
- It must be able to process data in as high a speed as possible. That is, the best possible speed grade must be selected.
- It had to fit with the economical budget that was given to the project.

A Xilinx Virtex-II Pro FPGA that met all of the above criteria's was chosen. The FPGA chosen has one PowerPC 405 CPU, four Rocket I/Os capable of baud rates up to 2.5 Gbit/s each and a speed grade of -7. The use of the CPU and the Rocket I/Os are out of scope in this project. The speed grade will be explained in Chapter 6.

3.3 Some Details of Xilinx Virtex-II Pro FPGAs

Figure 3.8 gives a general overview of a Virtex-II Pro FPGA. As can be seen from the figure, the *CLBs* take up most of the area in the FPGA. It further indicates the placement of the *Processors* (PowerPC 405) and the *Rocket I/Os.*

Figure 3.9 illustrates *one* CLB in a Virtex-II Pro FPGA. The *LUTs*, *MUXCYs* and the *carry chain* will be used in Chapter 4 to make wide AND gates. Note that the carry chain goes upwards in columns. Each CLB in a Virtex-II Pro is subdivided into *four slices*.

Figure 3.10 shows a block diagram of the available logic in *one slice*. The possible connections are not shown here. There are two *function generators*, *F* and *G*. Each function generator is capable of generating any 4-input Boolean function. A function generator can be configured as RAM, a ShiftRegister or a LUT.

The top/bottom half of a slice is called a *Logic Cell* (LC). Figure 3.11 shows the details of the top half of a slice. The resources used in one LC when making a wide AND gate as in Chapter 4, is indicated by the grey area.

Figure 3.12 shows the basic contents of an I/O Block (IOB). As can be seen, each IOB can be configured as either input or output. The need of the *Pullup Resistor* is indicated in Chapter 5.1.3 and Chapter 5.1.4.

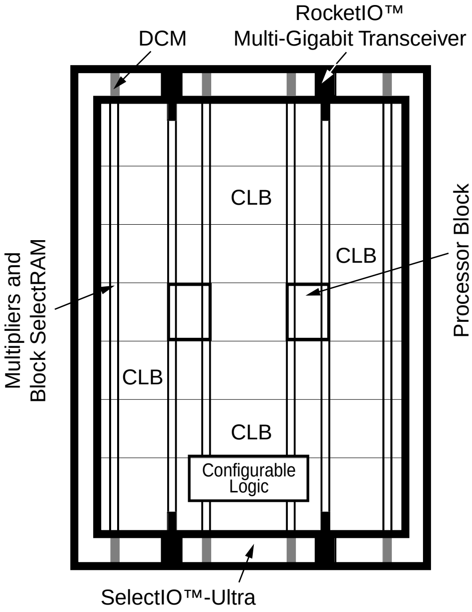


Figure 3.8: An Overview of a Virtex-II Pro FPGA [17]

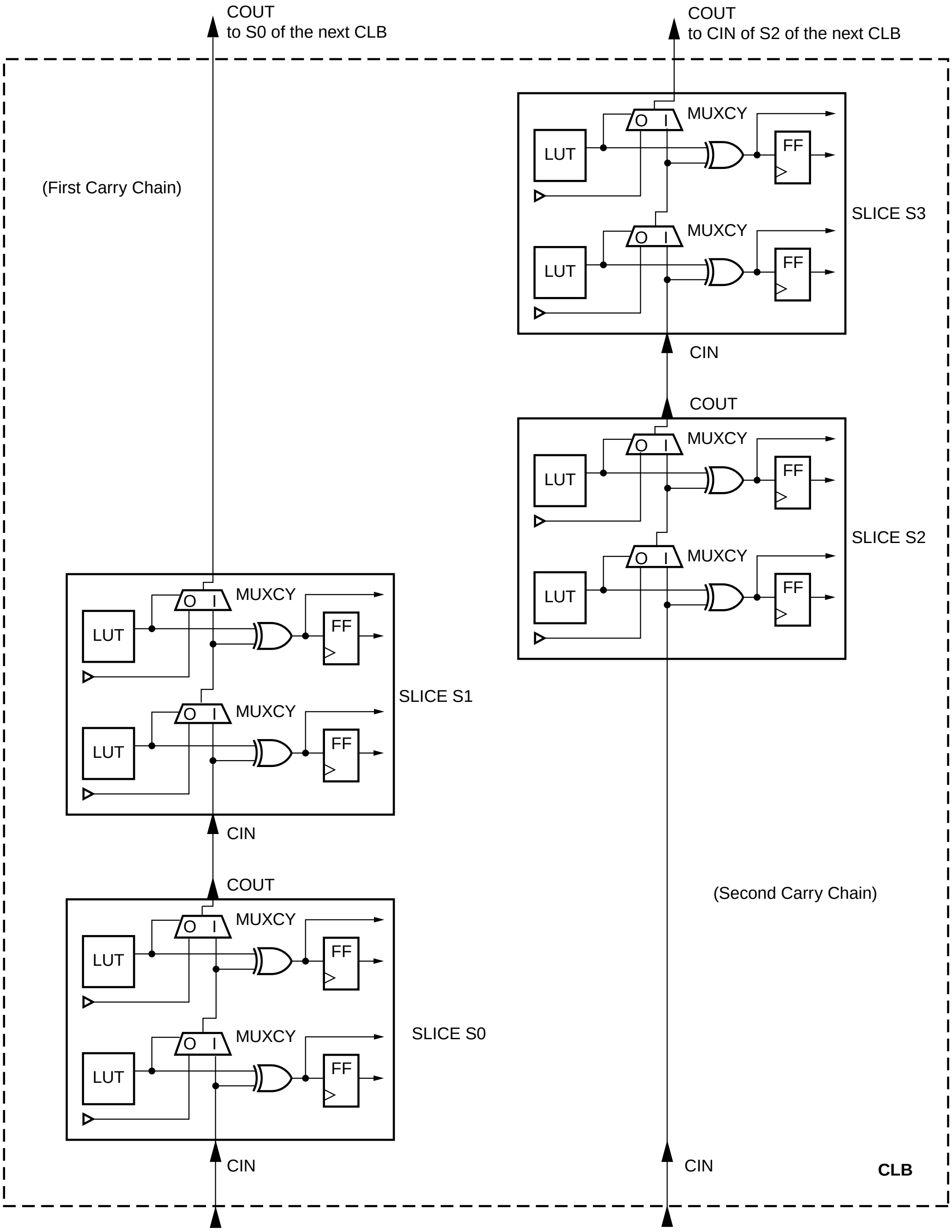


Figure 3.9: Fast Carry Logic Path in a CLB [17]

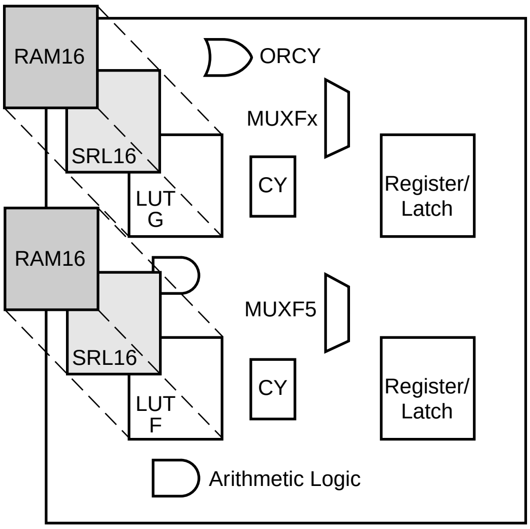


Figure 3.10: Slice [17]

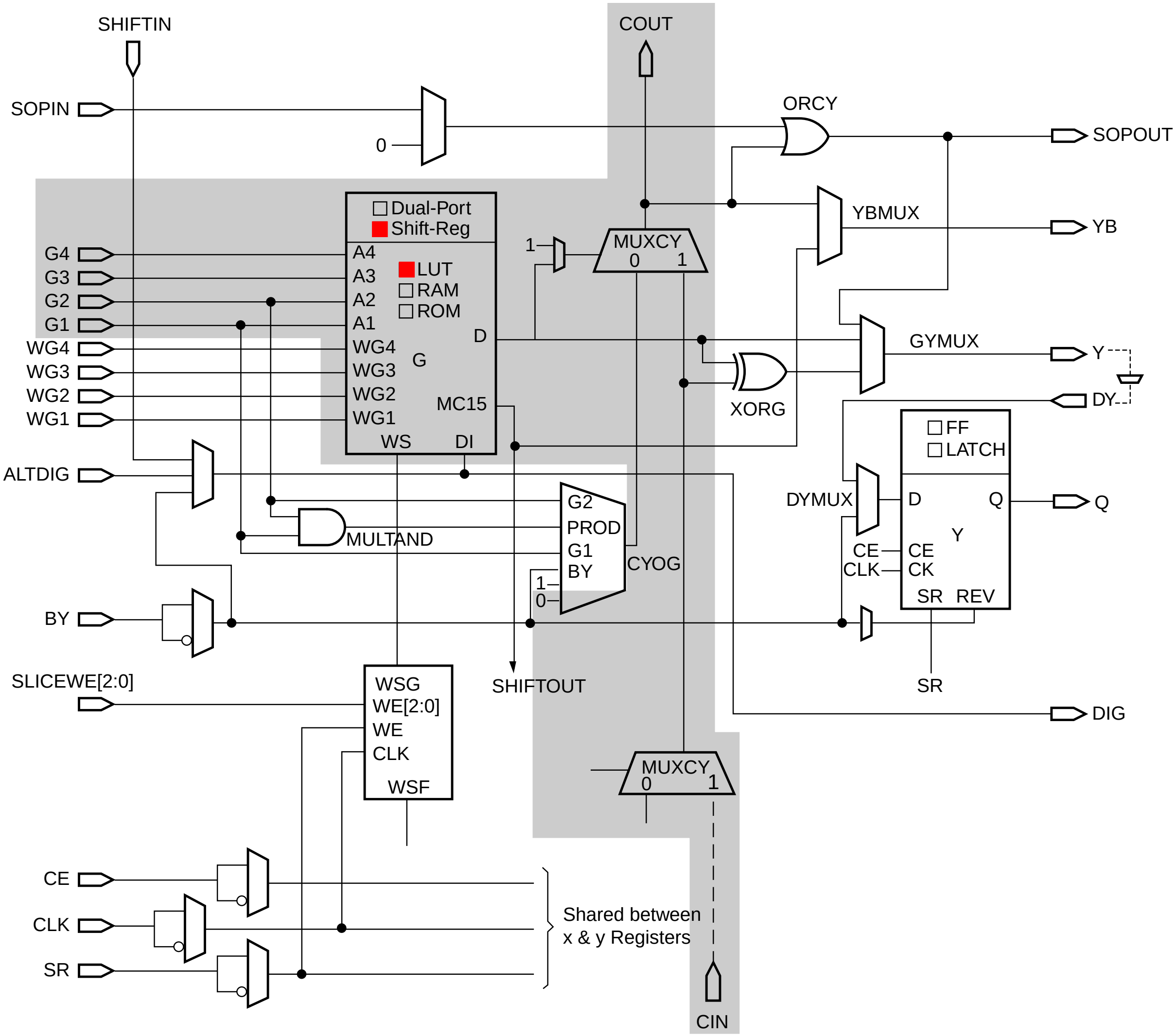


Figure 3.11: Logic Cell [17]

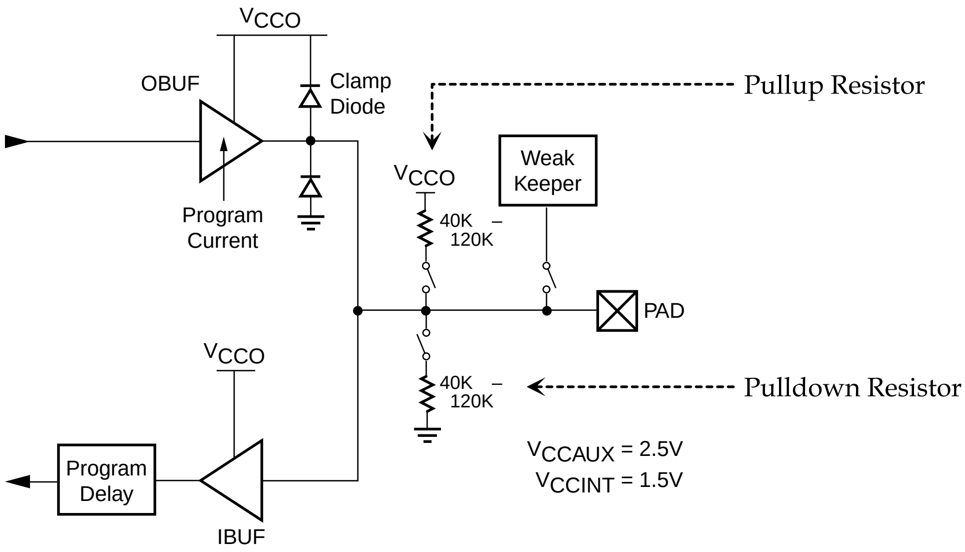


Figure 3.12: Input/Output Block [17]

4 MAKING A STRING MATCHER

Figure 4.1 shows a typical Snort rule (signature). The emphasized text illustrates a string that will be compared to the payload of an incoming packet over the network. These strings may have any length. Snort 2.0 has well over 1400 rules which may contain such strings. String matching therefore becomes a major bottleneck of the performance; Chapter 2.5. To remove this bottleneck, a hardware solution is proposed in this chapter by using an FPGA; Chapter 3.2.

```
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 4.1: A typical Snort rule (signature)

The top of Figure 4.3 illustrates Snort when running string matching on a PC. The middle figure shows how a string matcher in hardware can be used to speed up the performance of Snort. The illustration at the bottom shows how a string matcher has been implemented and tested in this project.

4.1 String Matching

Figure 4.2 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. For a string of *n* bit, we would need an *n*-input AND-gate.

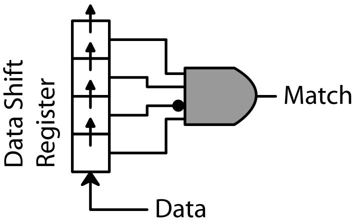


Figure 4.2: The Basic Idea of a String Matcher by Using an AND gate

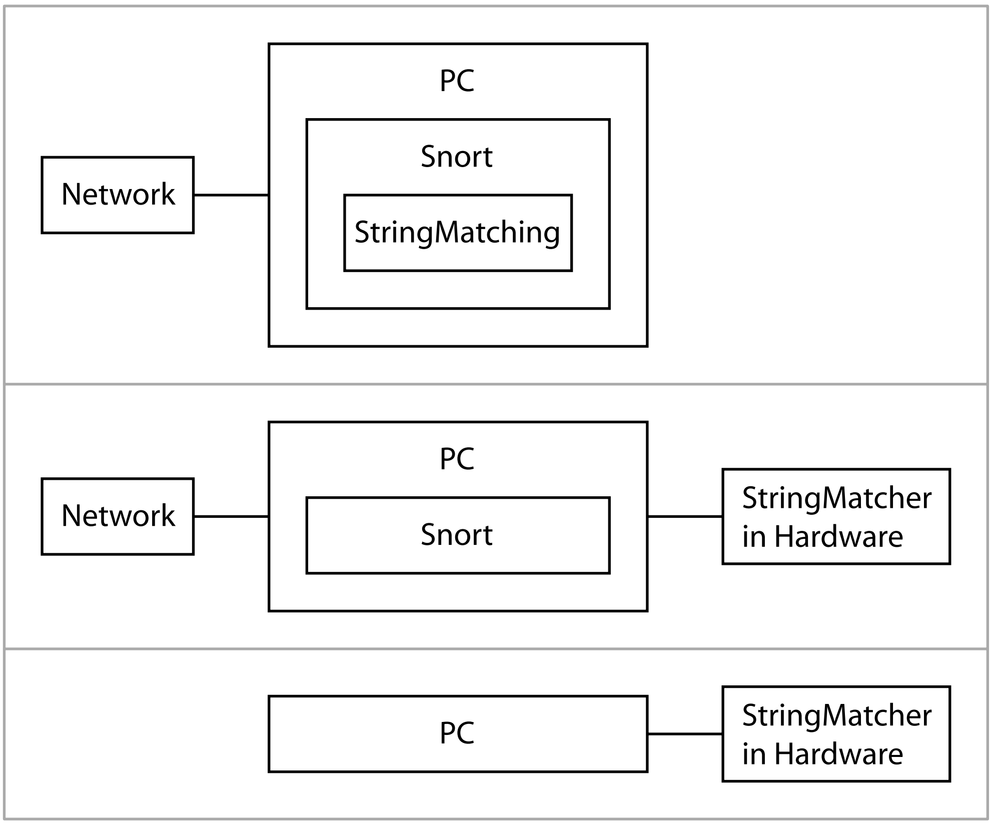


Figure 4.3: String Matching Targeted for IDS

With signature based IDS, it is an advantage of having a fast string matcher. By using a Content Addressable Memory (CAM), we can check the content part of one signature against numerous strings in *one* clock cycle. Figure 4.4 shows a comparison of RAM and CAM in read mode. They both store 1024 words of width 8. Similar to a RAM, a CAM stores words in an array. It should be noted that a CAM also has an address bus to be able to access every word in write mode. Also, the CAM shown here has a possibility of making only 256 (2^8) unique words, while there are 1024 (2^{10}) words available. This means that there will be some multiple matches no matter what data this CAM stores. The handling of multiple matches will not be a topic in this project.

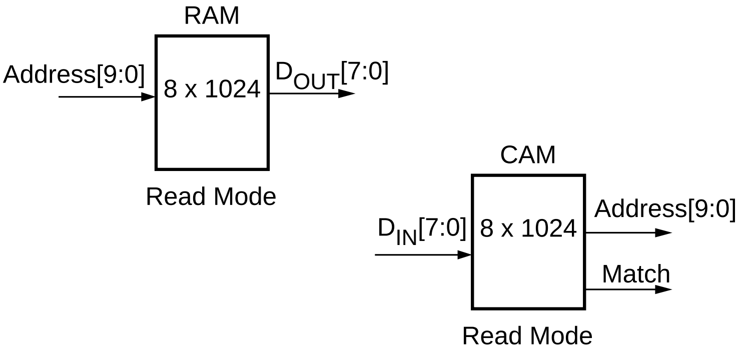


Figure 4.4: CAM/RAM Read Mode [15]

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 CAM we need a *Match* bit to indicate if the data at the input exists in the CAM or not.

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 4.5 shows how a CAM has been applied for string matching in this project. The data to be matched is sent to the CAM (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. Note that Snort has not been used, although the string matcher is intended for use with Snort.

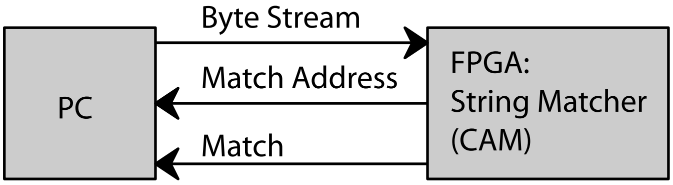


Figure 4.5: A CAM Applied in a String Matching System

4.2 Designing a CAM for String Matching with Snort

Virtex (-E / II / II Pro) FPGAs are suited for making logic that is equivalent to wide AND-gates. The components chosen for this purpose are LUTs configured as shift registers (SRL16E) and multiplexers (MUXCY); see Figure 3.11.

4.2.1 Requirements for a CAM used as a Snort String Matcher

The following properties are desired for the string matching:

1. The length of a string should be independent of the others. 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 more 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 1Gbit/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 is kept in order to make as flexible a CAM as possible. As seen in item 6 below, the need to make a write is obvious.
5. The time spent for making the VHDL code of a CAM should be small. Also, it should not be necessary to go into the details whenever a new CAM is acquired.

6. Future work: It should be possible to change the number and length of words in 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 programming capabilities in VHDL makes it a greater challenge of designing 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 acquired. The solution for this is provided by the following scheme:

Perl → (Generate VHDL source code)
VHDL → (Synthesis tools takes 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 project, 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 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 project is capable of generating VHDL files describing *any* CAM based on the SRL16E and MUXCY in less than one second. 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 project is described below.

To obtain a realistic dataset for testing the CAM (by simulation and by hardware), the following choices was 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 4.1 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 lest 4 bytes and no more than 32 bytes.
5. Write these strings to file.

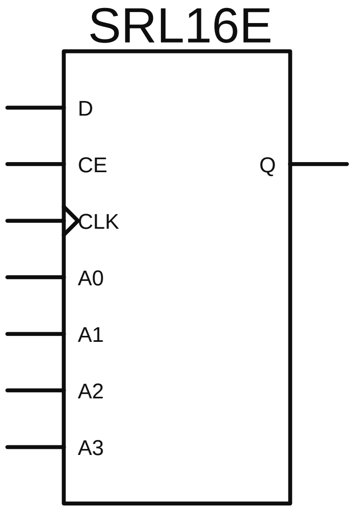
There were 1083 strings that matched the above criteria’s. As can be seen from Table 6.1, only 256 of these were used. A discussion for this is left to chapter 6.

A CAM with variable word-width it 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.

4.2.2 SRL16E

Because the SRL16E is the basic building block of this design, it is clearly an advantage to know in detail how it works. Figure 4.6 shows the block level symbol of this component. As a trivial matter, we know that there are 16 flip-flops connected in serial to make a shift register. The questions that remain to be answered are how the data is shifted in, how the Clock Enable

(CE) and address (A3-A0) affect the shifts, and how to control the output Q. However, information about the basic construction of this component has not been available.



D	**Data In**
CE	**Clock Enable**
CLK	**Clock**
A0-A3	**Address**
Q	**Data Out**

Figure 4.6: SRL16E Block Level Symbol

Experiments by simulation are a good alternative to get detailed knowledge about a component if little or no documentation about it is available. Appendix B shows details of a simulation of an SRL16E. Xilinx is providing libraries to make simulation of their components possible. Based on this simulation, we now have good reasons to believe that an SRL16E might look like the one shown in Figure 4.7. As can be seen from this figure, the only purpose of the address is control the output. Also, the data-bit is always shifted into the least significant address (the flip-flop at the top). A shift occurs at the rising edge of the clock (CLK), and all bits are then shifted one address higher (downwards in the figure). The bit in the most significant address (at the bottom) is overwritten, and lost. A shift can only occur when the clock is enabled; that is, when CE is high. With this knowledge it is easier to make a correct design, and less time will be spent on debugging it.

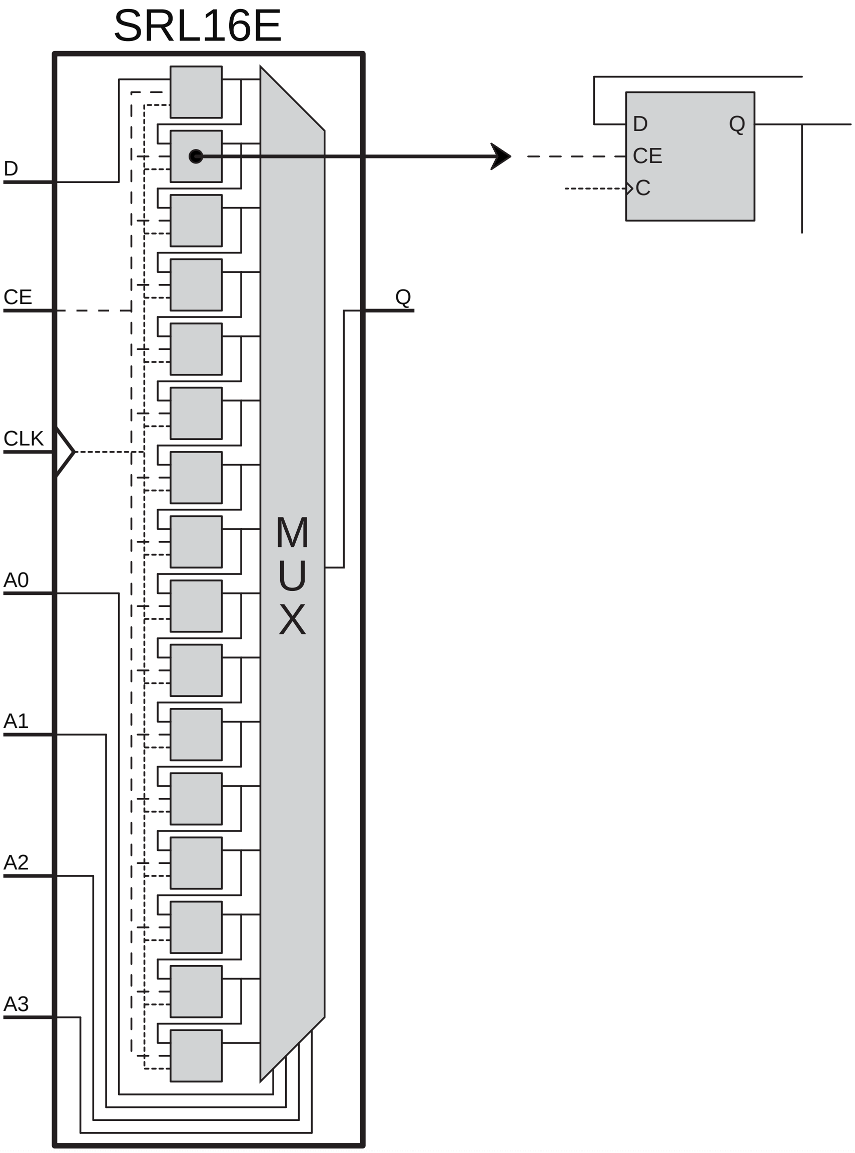
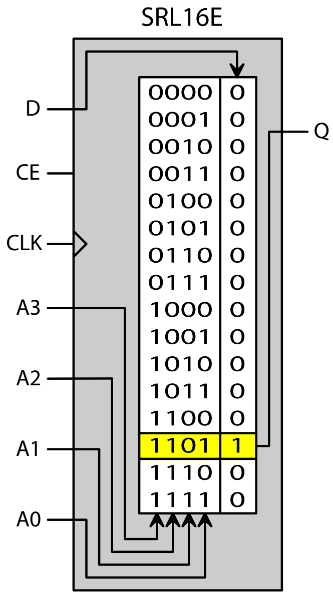


Figure 4.7: Looking Inside of an SRL16E

4.2.3 Configuring SRL16Es as CAM-Words

By configuring an SRL16E as shown in Figure 4.8 and disabling the shift, we get an equivalent logic to that shown in Figure 4.2. By connecting a data shift register to the address bus of the SRL16E, we get a 4-bit CAM [16]. Note that only one ‘1’ has been written to the SRL16E. For this reason, an SRL16E-based CAM consumes a large area in an FPGA. It is typical that CAMs need more logic per bit than RAM.



0000	0
0001	0
0010	0
0011	0
0100	0
0101	0
0110	0
0111	0
1000	0
1001	0
1010	0
1011	0
1100	0
1101	1
1110	0
1111	0

Figure 4.8: SRL16E Configured as an AND-gate

It is possible to connect SRL16Es in series through a carry-chain as shown in Figure 4.9 [17]. 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 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.

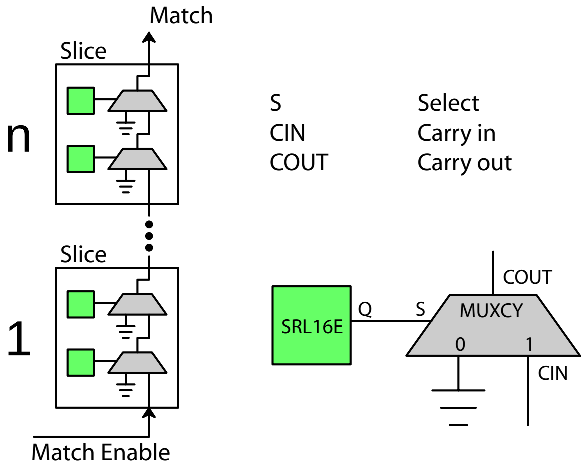


Figure 4.9: Serial Connection of SRL16Es to Make One CAM-Word

4.2.4 CAM Read Mode

Figure 4.10 illustrates a (5,9,4,6) byte CAM in read mode. Each location in the 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.

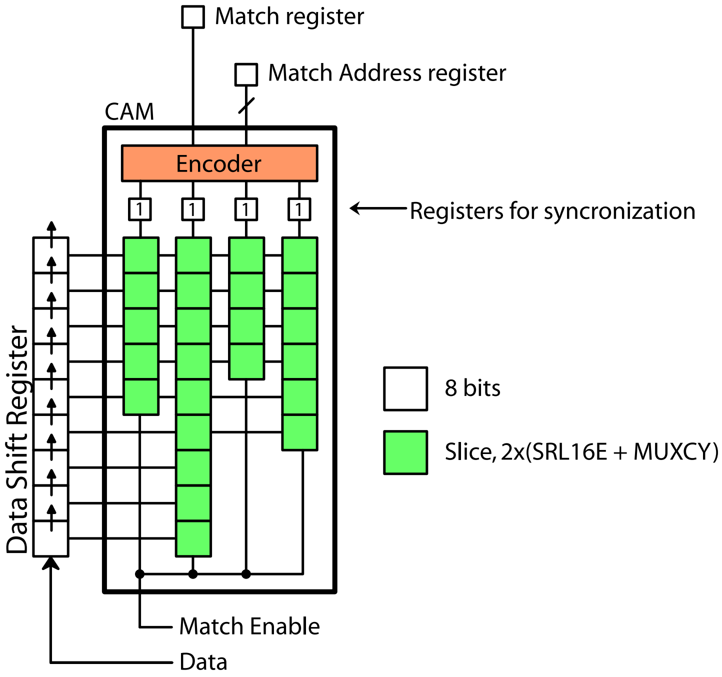


Figure 4.10: CAM Read Mode

4.2.5 CAM Write Mode

During a write (shifting data into SRL16Es) there should be made no shifts in the data shift register, and the *Match Enable* (as seen in Figure 4.9) should be low. This will ensure that the outputs of all words are kept low.

To make an "AND gate", a '1' should be written to the address of the SRL16E that contains the bit pattern to match. Because there are 16 locations to address in the SRL16E, 16 bits must be shifted in during one write. Only one address will have a '1' written to it, on the *last* shift of the write operation; the other addresses will have '0's. That is, if we want to make an AND gate to match the pattern "1101", a '1' should be written to address "1101" of the SRL16E, as seen in Figure 4.8. All other locations should store a '0'. Since a write to an SRL16E is always made to address "0000", we need two extra components to control a write; a counter and a comparator; see Figure 4.11. The counter will count the 16 clock cycles for one write. The comparator will determine whether to write a '0' or a '1' during a write. A '1' will be written only to the address that equals the data at the *Data Shift Register*.

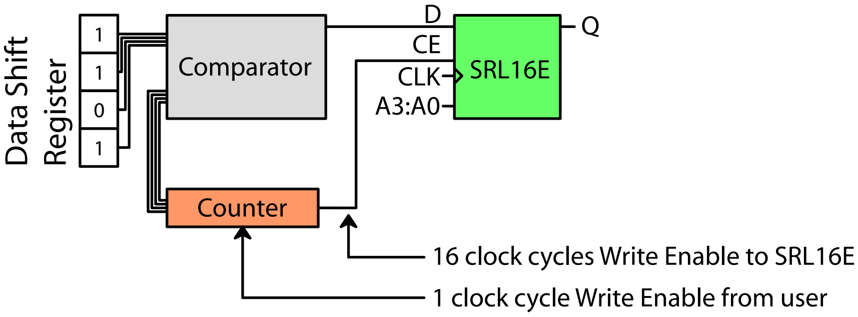

Data Shift Register
1
1
0
1
Comparator
Counter
D
CE
CLK
A3:A0
SRL16E
Q
16 clock cycles Write Enable to SRL16E
1 clock cycle Write Enable from user

Figure 4.11: Detailed Look of a CAM in Write Mode

The counter in Figure 4.11 is illustrated in detail in Figure 4.12. It generates a *16 clock cycles Write Enable* signal for the SRL16E, and also gives the counter value, *count(3:0)*, to each comparator; see Figure 4.13. As the *1 clock cycle Write Enable* goes high to start the counting, "1111" (D) is loaded into the counter. The CE goes high and the counter starts. As long as there is at least one '1' in the counter output (Q), the output "Write Enable"-signal will be high. When Q is at "0000", CE will go low and the counter will stop.

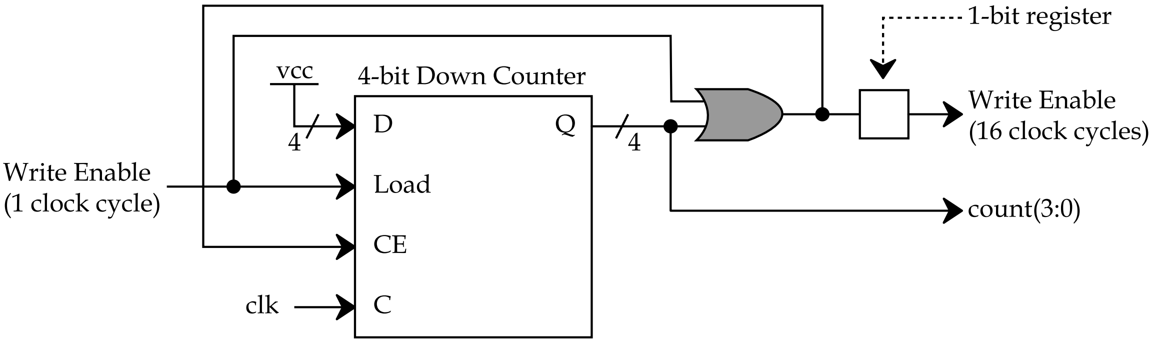


Figure 4.12: Counter

Figure 4.13 shows in details how a comparator is constructed. The input is constant during the write. The address being written to is equal to the counter value. This value can therefore be compared to the input from the shift register. If the input equals the counter value, a '1' will be written. In all other cases a '0' will be written. The XNOR (identity) function is suited for this comparison. The "Compare result"-bit is shifted into the SRL16E.

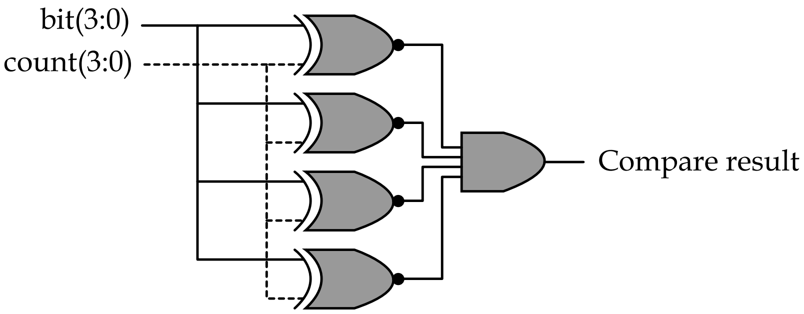


Figure 4.13: 4-Bit Comparator

Figure 4.14 gives an overview of a (2,3,1,3,1) byte CAM in write mode. Each comparator is connected to all matching units (SRL16Es) in the corresponding row. Only one word can be written to at a time. This word is chosen by the decoder. Note that the decoder passes on the *Write Enable* signal — output by the *Counter*, to the chosen word.

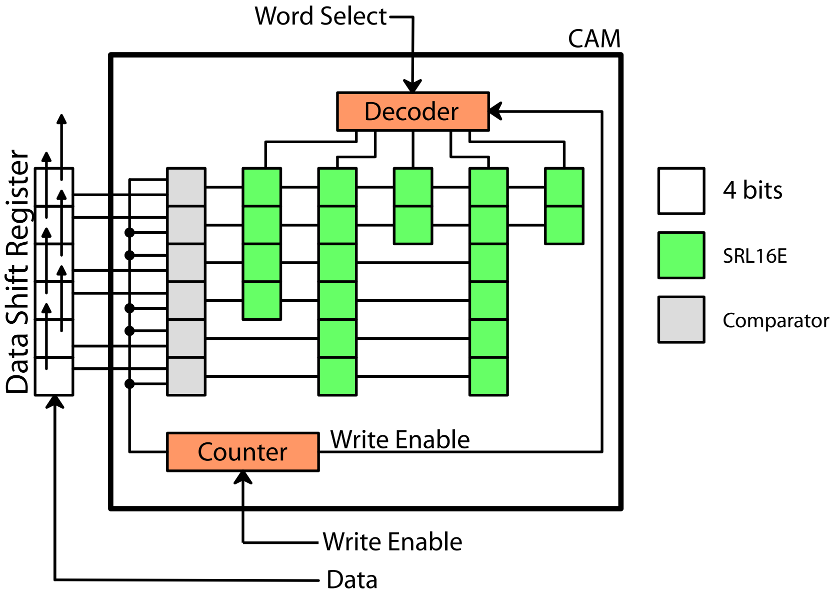


Figure 4.14: CAM Write Mode Overview

4.3 Test of the Design by Simulation

Simulation is entirely done by software; *ModelSim 5.6f*. For this project, VHDL-testbenches have been chosen for simulation. A testbench holds all properties of the design to be tested. It is possible to give *stimuli* to the design. Stimuli are plain text in the testbench, and acts as input to the design. By reading the outputs, we may predict how the design will behave in hardware. The modules in any design (counter, decoder etc. as in this design) could be simulated individually, thereby increasing the level of confidence to one specific component.

Figure 4.15 shows a plot where an 8-word (134 byte) CAM has been simulated. Only the last part of the simulation is shown. The upper five signals are stimuli from the testbench. The outputs (the two signals at the bottom) are then triggered. The stimuli in this simulation have been made by the following criteria:

- Write to all words in CAM (one at a time). Use the dataset that was created from Snort-rules; Chapter 4.2.1.
- Set *Match Enable* high. This will enable read mode.
- For each clock cycle, change the input data. Use the same data that was used during write, except the 2nd last one. It is then expected that the *Match Address Register* should count from lowest address to highest address (except from the second last read).
- For the 2nd last read, use data that is not written to CAM. The *Match Register* should then go low. Observe that it goes high the next clock cycle. This is done to verify that the *Match Register* is updated every clock cycle.

Similar simulations have been made for various CAMs. Some carefully chosen numbers of words in CAMs that have been simulated are {7, 8, 9, 15, 16, 17, 31, 32, 33, 63, 64, 65, 127, 128}. Numerous bugs of the CAM-design were found by observing the outputs generated of (various) stimuli. For this reason, it should be obvious that simulation is highly important when making any design targeted for hardware.

Almost all bugs can be detected by simulation. Detecting bugs by hardware is more complicated. How many bugs we must search for in hardware depend of how well the design was simulated (how cleverly the stimuli were chosen). Still, in large designs it is impossible to simulate all possible combinations of inputs, and it should therefore be expected that some of the bugs must be searched for in hardware.

Professional designers are known to use as much as 70% of the time spent on a project for testing their design [18]. To stay at a professionally competitive level, various ways of testing a design must be known. To be able to combine methods gives further advantage over competitors.

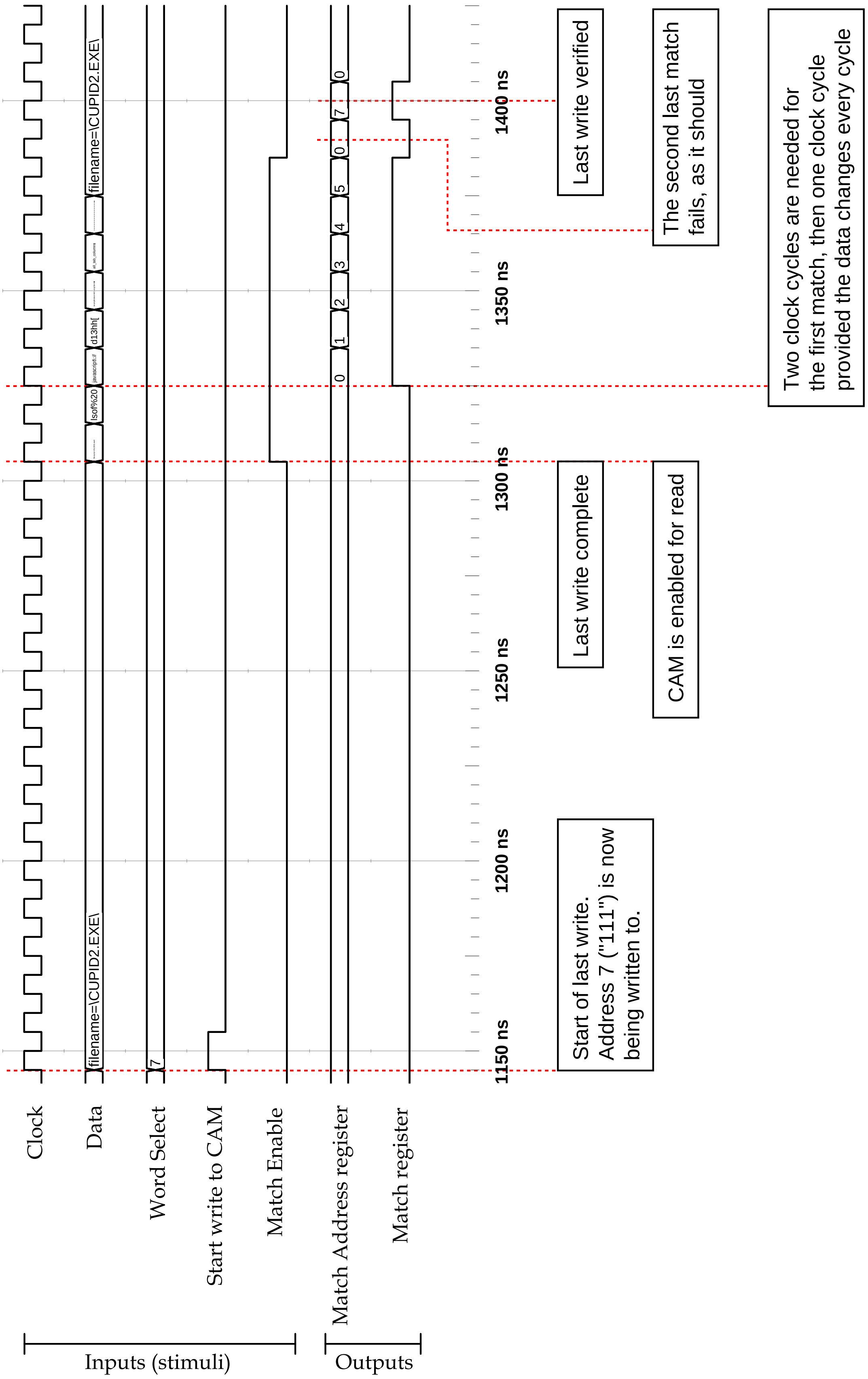


Figure 4.15: Simulation of a 134 Byte (8-Word) CAM Design

4.4 Functional Verification of the CAM Design in FPGA

Figure 4.16 illustrates how an 1822 bytes (128 words) CAM was verified in hardware. The *User Interface* communicates with the FPGA through the Universal Asynchronous Receiver-Transmitter (UART). The ports of the UART are commonly called Serial Ports, but they may also be called RS232 ports. The *Development Board* does not have a UART. Thus, it was necessary to make modules in the FPGA that could communicate with the UART. RS232 modules (*RS232RX* and *RS232TX*) were designed to handle this communication. One *RS232RX* module receives one byte at a time from the *User Interface*. Sending data the other way, the *User Interface* receives data, one byte at a time, from the *RS232TX* module. The *MAX3223C* and *MAX3221C* are driving the signals between the UART and the I/Os of the FPGA. Details of how to make the RS232 modules are left to Chapter 5.

The FPGA is entirely controlled by the *User Interface*. All input operations to the CAM are communicated through the *Instruction Register Control*. As can be seen, there are seven instructions needed to control the CAM. The *Instruction Register Control* receives one byte for every instruction. It is therefore possible to expand the number of instructions to 2^8 without introducing a shift register.

The instructions in the *Instructions Register* are the following:

•	*Match Enable*	on	Enable read mode
•		off	Disable read mode
•	*Write Enable*		Write enable signal
•	*CAM Address Shift*	on	Shift (write) enable of *Word Select Register*
•		off	Shift (write) disable of *Word Select Register*
•	*Data Shift*	on	Shift (write) enable of *Data Shift Register*
•		off	Shift (write) disable of *Data Shift Register*

The Word Select Register is set to exactly the number of bits needed; see Figure 4.14. When the number of address bits to the CAM is less or equal to 8, only one byte has to be uploaded in order to select a word in CAM.

The *Data Shift Register* is configured to have the exact length of the longest string in the CAM. Data is being shifted into the least significant bits, one byte at a time, and then shifted upwards as a new byte is received; see Figure 4.10 and Figure 4.14.

The outputs of the CAM (*Match* and *Match Address*) go into a *Match Detection Unit*; see Figure 4.10. If there is a match, it will be indicated by the *Match* bit. The *Match Address* will be sent to the *User Interface* as shown.

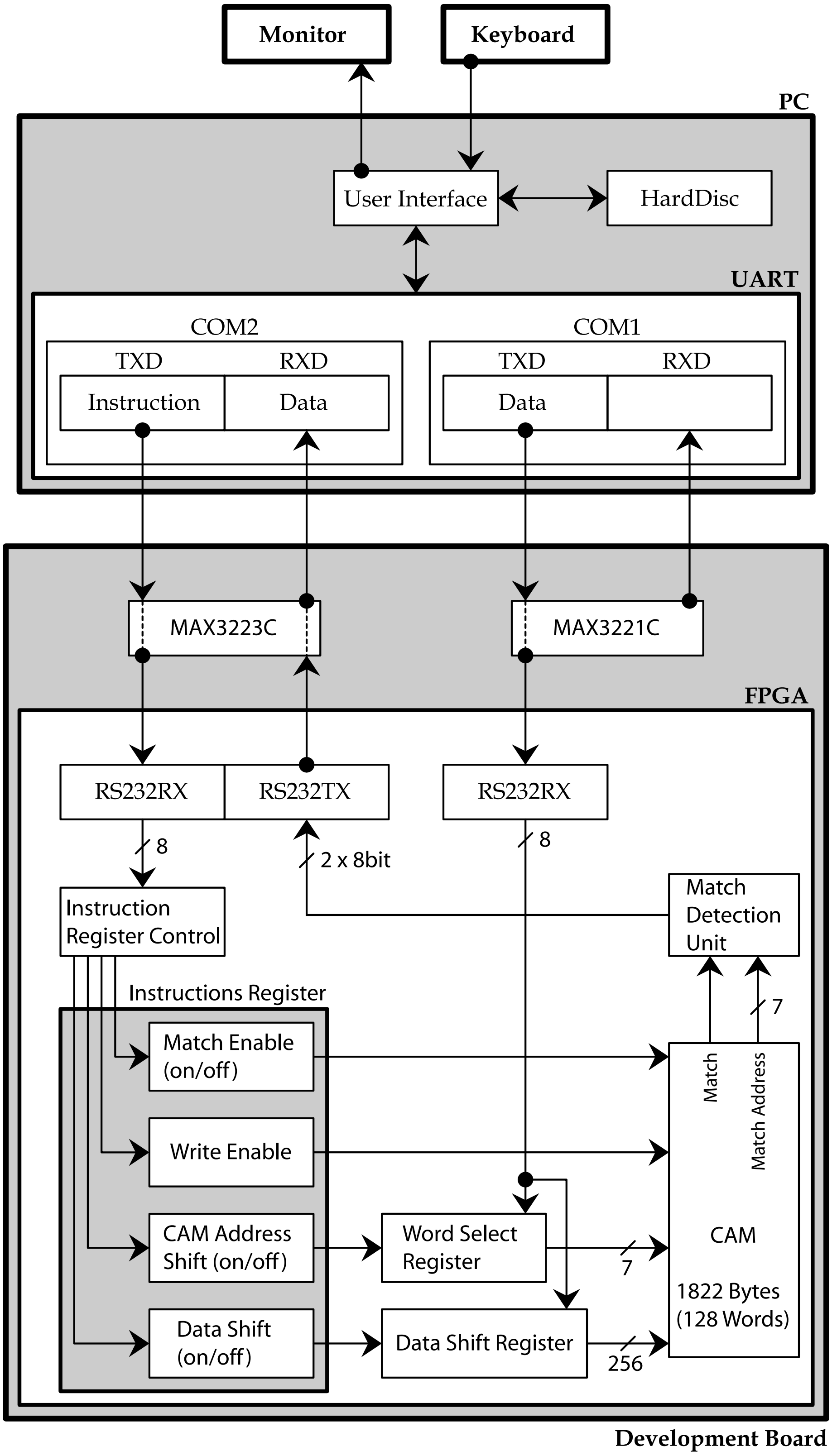


Figure 4.16: Verification of CAM in Hardware

Description of the User Interface:

- First, the CAM-data are read into an array from file. The data are taken from the dataset described in Chapter 4.2.1.
- Write procedure
    - Select one address in CAM to write to. The address in CAM corresponds to the index in the array held by the *User Interface*
        - Turn *CAM Address Shift* on
        - Write address to *Word Select Register*. Eight bits are sent at a time. Thus, for CAMs smaller or equal to 256 words, only one write is needed
        - Turn *CAM Address Shift* off
        - Turn *Data Shift* on
        - Upload word to *Data Shift Register*, one byte at a time
        - Fill data shift register with zeroes so that the word is positioned at the most significant bits in the shift register; see Figure 4.14
        - Turn *Data Shift* off
        - Give a *Write Enable* signal
- Enabling Read Mode. Matches in CAM will then be displayed on the monitor.
    - Turn *Data Shift* on
    - Fill *Data Shift Register* with data
    - Turn *Match Enable* on when *Data Shift Register* is filled up
    - Continue uploading data continuously to the Data Shift Register

Figure 4.17 gives an illustration of the used resources in the FPGA when configured with the design as shown in Figure 4.16. 93% of the resources are used; see Table 6.1, row 3. In both images, on the right-half, the rectangle represents the *IBM PowerPC 405 RISC CPU (PPC405)*. *Xilinx Floorplanner* (right) illustrates the various widths of the words. One colored line illustrates one word. *Xilinx FPGA Editor* (left) illustrates the resources that are used. Future work includes running Snort software on the *PPC405*.

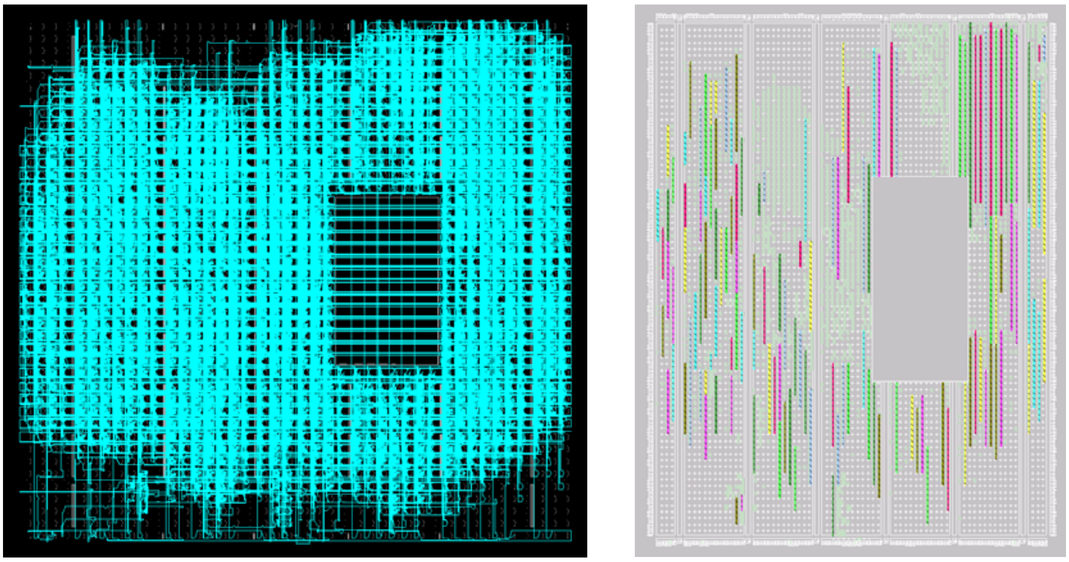


Figure 4.17: An XC2VP7 Configured With a CAM of 1822 Bytes (128 Words)

5 MAKING AN RS232 CONNECTION

However simple it may seem to make an RS232 connection as the one shown in Figure 4.16, there are a lot of practical challenges to overcome in order to make it work properly. In this chapter the hardware and software solutions for this purpose are described. Experiences made along the way are included in such a manner that the reader should not need to spend much time to understand the solutions presented here.

5.1 Hardware Solutions

The hardware chosen for this project is the *Development Board* shown in Figure 5.1, and the *P160 Communications Module* shown Figure 5.2. This Module is a plug-in board that goes into the *P160 Expansion Slots* of the Development Board. This hardware is well suited for prototyping advanced designs in micro electronics. In the following, the on-board debugging possibilities on the Development Board will be introduced. The purpose of using these debugging options is to make reliable RS232 modules (*RS232RX* and *RS232TX*) in order to communicate with a PC through the two RS232 ports available. One RS232 port is available on the Development Board; the other is available on the P160 Communications Module.

The *Parallel IV Cable Port* is used to program the FPGA directly. It is also used to program the *ISP PROMs* that will program the FPGA automatically when the power is turned on. Also, when activating the *Program Button*, the FPGA is programmed with the configuration stored in the *ISP PROMs*. The *Program Status LED* will glow when the programming of the FPGA is done. The remaining components that have been used are described in details in the following subchapters.

Detailed simulation of a design inside an FPGA is possible. Simulation of components that are external to the FPGA cannot be simulated well without having a simulation model for that component; *LCD*, *SDRAM* etc. Only a simple simulation is available without simulation models, but they are still useful in order to remove the most critical bugs. The complete debugging of these components was therefore done by experiments in hardware, as will be described in the following subchapters.

Finite State Machines (FSMs) are used to describe the desired behavior of the circuitry in this chapter. Appendix C shows how to describe a non-sequential FSM graphically. Appendix D shows a way of describing a sequential FSM. In the latter case the FSM is designed in conjunction with a timing diagram. In both cases the VHDL code is easily derived from the descriptions of the FSMs. In the case where an FSM is regarded trivial, as in Chapter 5.1.1, the FSM is not outlined. Refer to Appendix F for source code. All FSMs are designed from scratch.

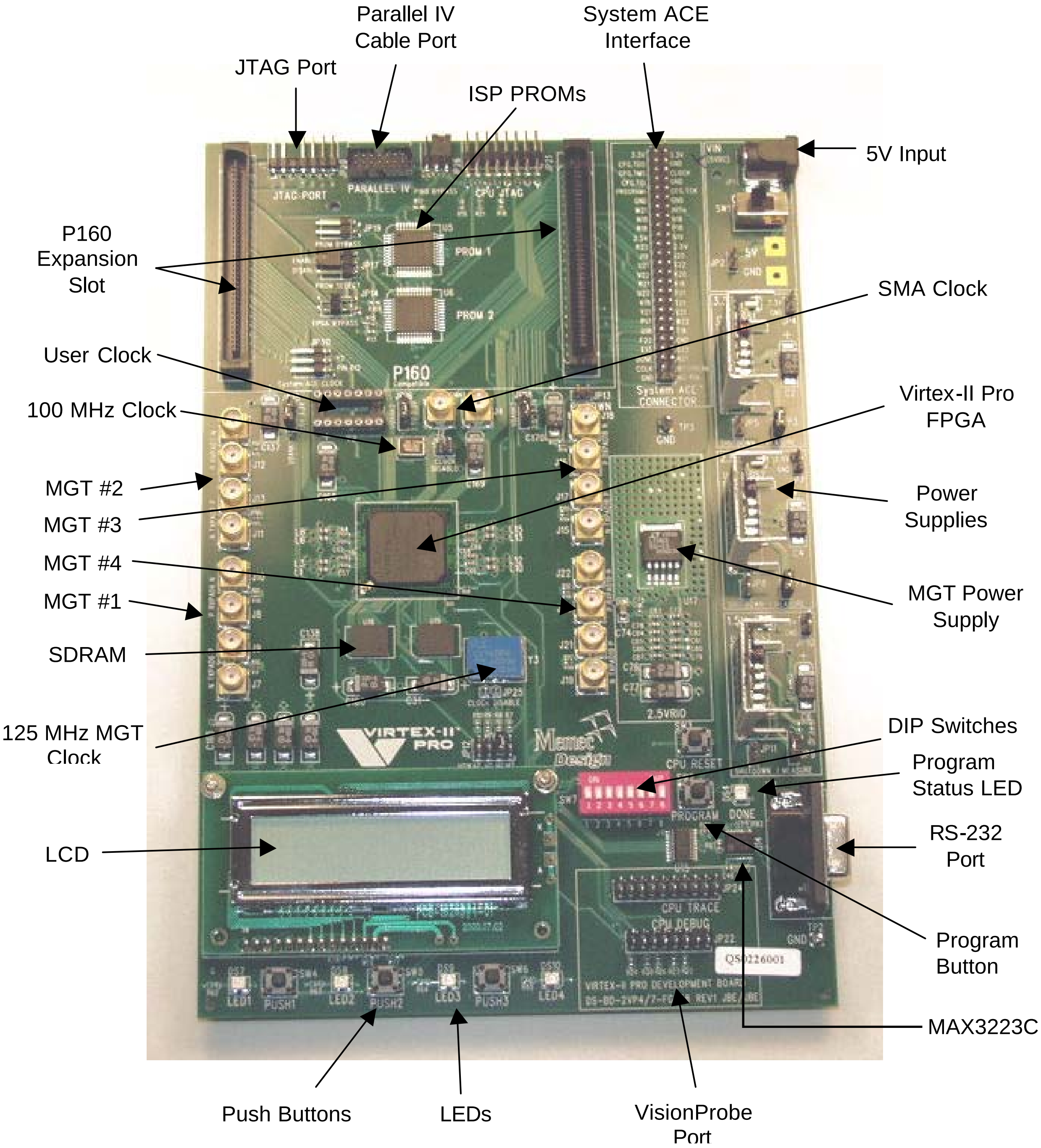


Figure 5.1: Vitex-II Pro Development Board [19]

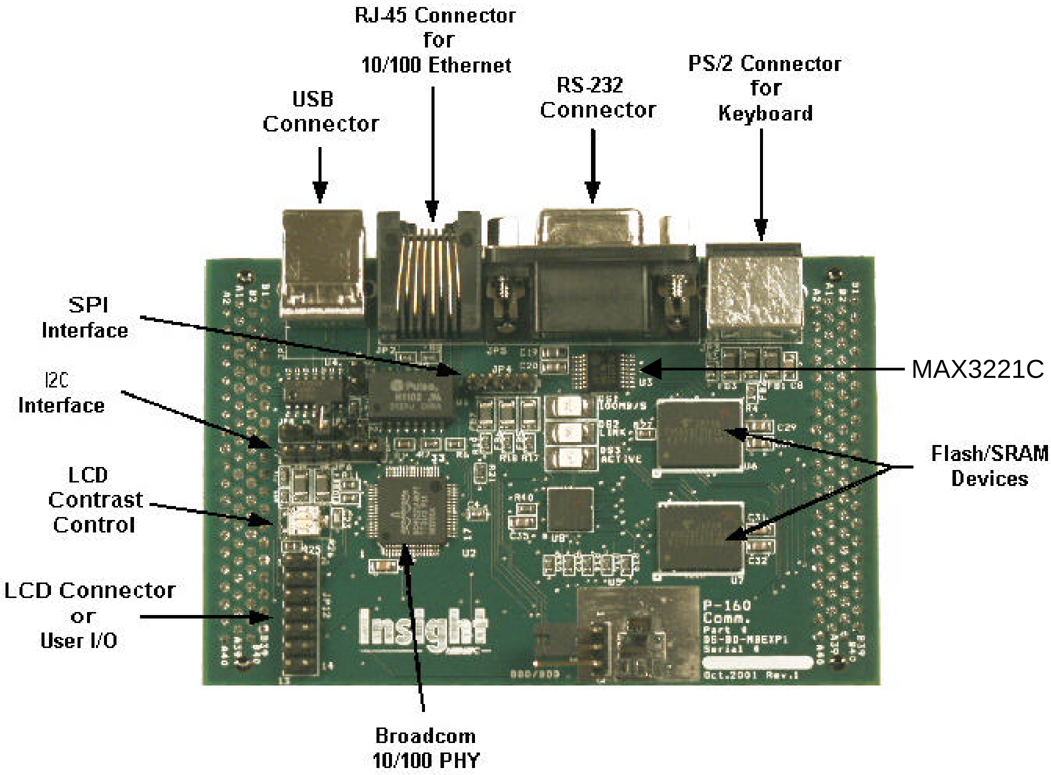


Figure 5.2: P160 Communications Module [20]

5.1.1 Light Emitting Diodes (LEDs)

The easiest component to use on the development board is a LED. Figure 5.3 shows how one LED is connected to the FPGA. As can be seen from the figure, the LED will turn on when the I/O outputs a current less than 2.5V. A resistor is connected in serial with the LED, to limit the current.

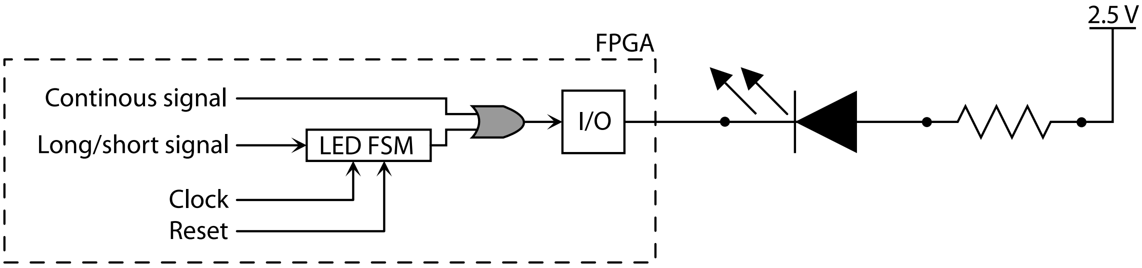


Figure 5.3: One LED Connected to an I/O

When giving a long ‘0’-signal to the *I/O*, the LED will glow bright. The shorter the signal is, the weaker (and shorter) the led will glow. If the signal to the *I/O* is too short, the LED will not glow at all. To accommodate this, we can make a design that makes the LED glow for a minimum amount of time. The minimum time chosen for this project is 1/100 second. This design is shown at block level in Figure 5.3; implemented as *LED FSM*.

5.1.2 CPU Reset Push Button

Figure 5.4 shows how the CPU Reset push button is connected to the FPGA. This button is dedicated to the CPU that is integrated on the FPGA, but it may also be use by other modules. Note that it has an external pullup resistor attached to the input. Thus, every time the button has been pushed and released, the wire going into the I/O is pulled up to ‘1’.

This button is in this project used as a global asynchronous reset. Detailed information of how this button is connected to the FPGA has not been found. From experiences, it appears to have a built-in “Push Button FSM”, much the same as the one described in 5.1.3. The CPU Reset push-button works as follows:

- When the button is activated and held down, nothing happens.
- When releasing the button, it is activated and one short low signal is sent to the attached circuitry.

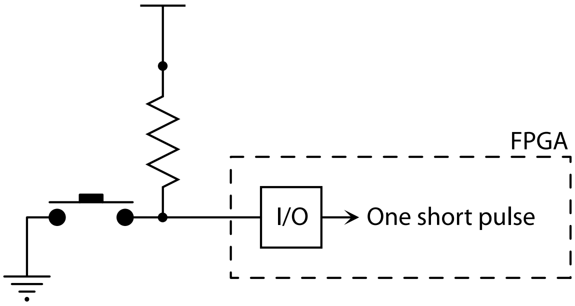


Figure 5.4: CPU Reset Push Button

5.1.3 User Push Buttons

Figure 5.5 shows how one user push button is connected to the FPGA. When the button is pushed down, the wire connected to the I/O is pulled down to ground. A *Continuous Pulse* is made as long as the button is down. When the button is released, the current in the wire to the I/O must be pulled up. A pullup-resistor as shown in Figure 3.12 can be used to accomplish this. If this resistor is going to be used, it must be specified in the design. This can be done either in the VHDL-file where the push-button is used, or in the User Constraint File (UCF). The UCF is also where to associate signals in the VHDL-design, with pins on the FPGA. The pullup-resistor is activated when programming the FPGA.

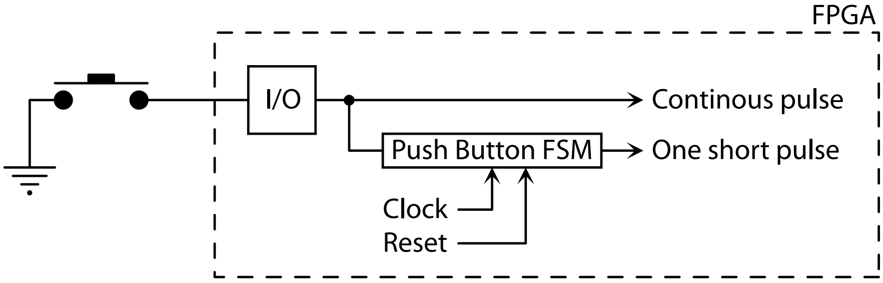


Figure 5.5: One User Push Button

A continuous pulse is not always convenient. We often need a push button that gives one short pulse also. That is, when the button is pushed down it should activate the attached circuit only once. This is accomplished through the *Push Button FSM* as shown in Figure 5.5. For this report, the *one short pulse* is one clock cycle. The details of the *Push Button FSM* are described in Appendix C.

Before continuing, it is important to test the *LED FSM* and the *Push Button FSM*. This is done by making a design that connects one push button to four LEDs; see Figure 5.6. The grey zone indicates the test design (simplified). The design works as follows:

- LED 1: By connecting the *Push Button* directly to the first LED, the LED should glow for as long as the button is pushed down.
- LED 2: By connecting the same input to a *LED FSM*, and then to a second LED, this LED should flash for 1/100 second each time the button is pushed down.
- LED 3: Taking the input from the button through a *Push Button FSM*, the signal to the third LED will be active for only one clock cycle. This led should not glow at any time.
- LED 4: By taking the output from the Push Button FSM, then connecting it to a LED FSM before outputting it to a fourth LED, this LED should flash for 1/100 second each time the button is pushed down.

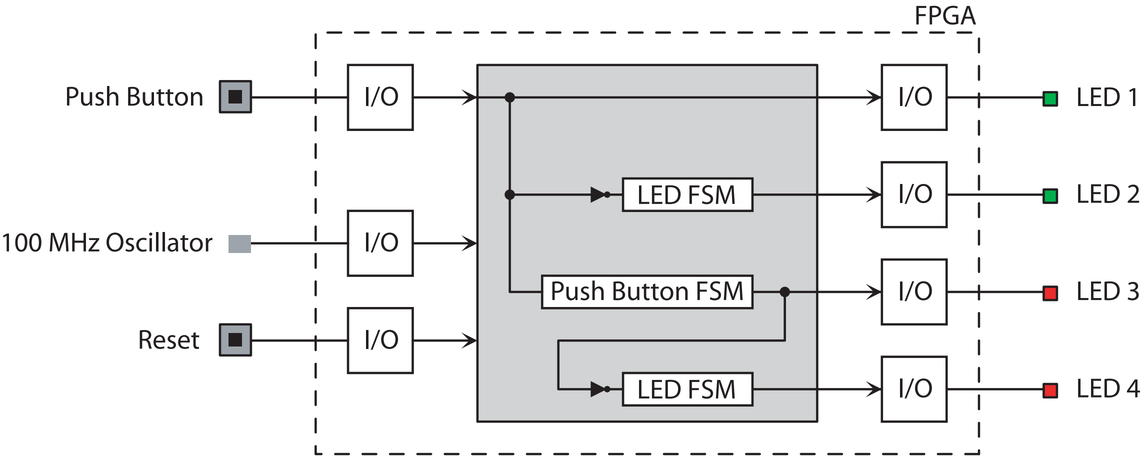


Figure 5.6: Test of LED FSM and Push Button FSM

5.1.4 Dual In-Line Package (DIP) Switches

The DIP contains 8 identical switches, each connected to an I/O. They are connected to the I/Os the same way as the User Push Buttons; see 5.1.3. Thus, the internal pullup resistors in the I/Os must be attached for every switch in the DIP; see Figure 3.12.

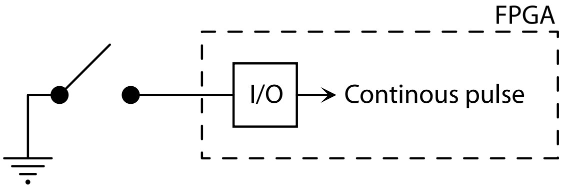


Figure 5.7: Illustration of One Switch in the DIP

5.1.5 Liquid Crystal Display (LCD)

As can be seen from Figure 5.8, the *LCD* acquires 10 I/Os of the FPGA. Details of the *LCD FSM* are left to Appendix D. The LCD FSM controls *one* write to the LCD.

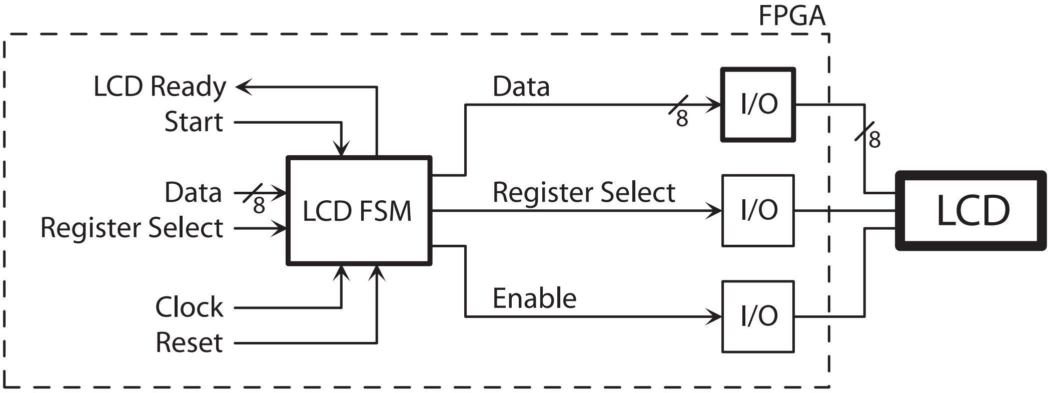


Figure 5.8: LCD FSM

There are two types of registers in the LCD; an instruction register and a data register, each of 8 bits. Thus, all data written to the LCD consists of 8 bits. The *Register Select* determines whether data should be written to the data register or the instruction register. The data register will display a character to the LCD. The instruction register will clear the LCD, set the next position for writing a character etc.

At startup an init sequence is performed. Seven instructions are then written to the instruction register of the LCD. The init sequence is written again every time *Reset* goes low. One write sequence in the LCD FSM works as follows:

- When *LCD Ready* is high, it indicates that the LCD is ready to be written to.
- By setting *Start* high, *Data* and *Register Select* are stored in the *LCD FSM* and then passed on as output to the LCD.
- *LCD Ready* goes low to indicate that the start of a new write is not possible.
- At the appropriate time, the *LCD FSM* enables the *LCD* by setting *Enable* high for a given time, before setting it low again.
- Depending on the kind of instruction that has been written to the LCD, there is a given time to wait before the LCD is updated.
- When done, *LCD Ready* goes high to indicate that the start of a new write is possible.

Figure 5.9 shows how the *LCD FSM* was tested and verified in hardware. The outputs from the *LCD FSM* to the *LCD* are the same as in Figure 5.8 and are omitted here. The grey zone indicates the design needed to put this design to work (simplified).

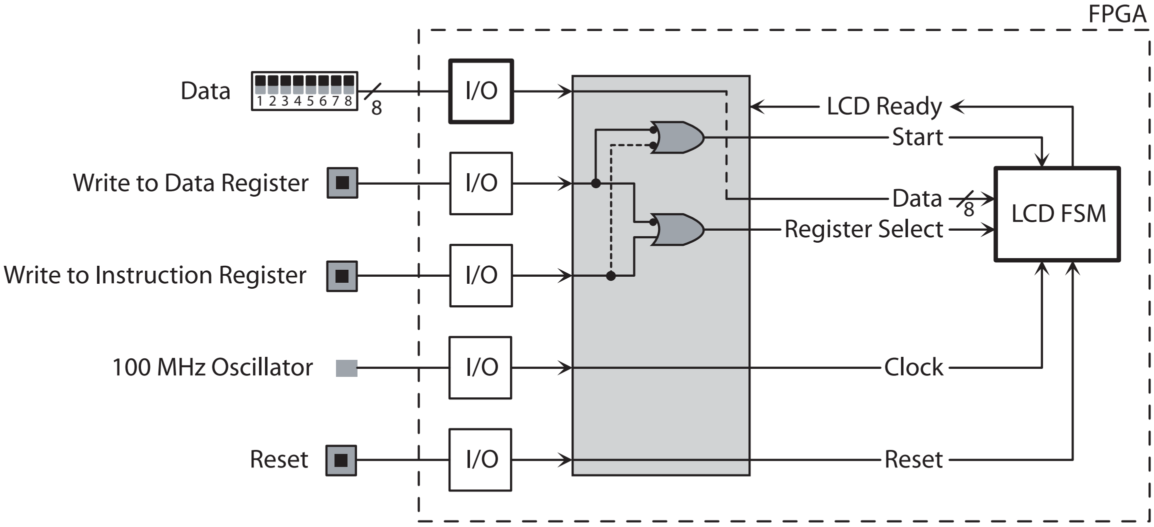


Figure 5.9: Testing the LCD FSM

The *Data* to the LCD is given by the DIP switch. *Start* will be activated whenever one of the two attached push buttons is activated, and *LCD Ready* is high. When the *Write to Data Register* push button is activated, the *Register Select* is set such that a character will be written to the LCD. When activating the *Write to Instruction Register* push button, the *Register Select* is set to write an instruction like “Clear LCD” etc.

5.1.6 RS232 – UART Communication

At the time a communication between the PC and the FPGA was needed, the Communication Module was not available; Figure 5.2. Thus, the RS232 port on the *Development Board* was chosen to establish this communication. Due to limited time of completing this report, the USB and Ethernet connections of the communications module have not been used.

A PC has serial ports that are controlled by a UART. Table 5.1 (second column) shows how the UART was set up for the work in this project. There is no UART at the Development Board, so a receiver module (RS232RX) and a transmitter module (RS232TX) are needed; Chapter 4.4. The last two columns show the parameters chosen for these two modules.

	PC (UART)	FPGA (RS232RX)	FPGA (RS232TX)
Input Clock	UART Clock	100 MHz	100 MHz
Baudrate	115200	> 115200	< 115200
Parity	None	None	None
Data bits	8	8	8
Stop bits	1	0.5	> 1

Table 5.1: The Relation between UART and FPGA Communication

Prior to making the RS232RX and the RS232TX modules, the time (ns/bit) to receive or transmit *one* bit should be known. This must be calculated with respect to the selected UART baudrate. The calculation is simple:

$$\frac{1\,\mathrm{s}}{115200\,\mathrm{bits}} \approx 8{,}680.55\,\mathrm{ns/bit}$$

That is, one UART clock cycle lasts 8,680.55 ns. In addition to the data bits and the stop bit, there is also a startbit. Thus, there are ten bits to be transmitted/received, for each byte.

The *Development Board* has a 100 MHz oscillator. It is not possible to derive a clock signal in the FPGA that exactly matches the UART clock. In both the RS232RX and RS232TX modules the time spent for sending/receiving one databit is 8680 ns. This is 5,55 ns faster than the UART. The time spent for the startbit and the stopbit differ in these two modules.

RS232RX

The top row of Figure 5.10 illustrates a PC transmitting one byte through the UART. When idle, a logical ‘1’ is continuously transmitted. When sending a new byte, a startbit is first transmitted. The startbit is logical ‘0’ for one UART clock cycle. In the following 8 cycles the byte is transmitted, one bit at a time. At the end of the transmission, a stopbit (logical ‘1’) is transmitted.

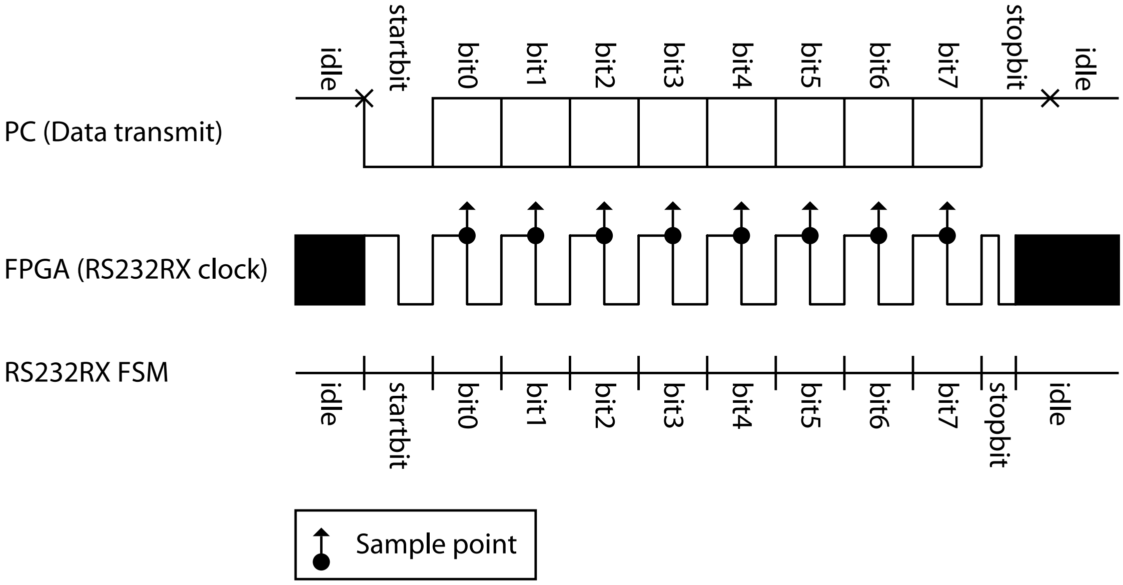


Figure 5.10: RS232RX vs. UART

In the middle row we can see the clock signal used in the RS232RX module. The dark squares indicate a fast oscillating clock. As can be seen, this is a clock with variable speed. It is used to control the *RS232RX FSM*. Arrows indicates at which point the samples are made; half way through one clock cycle.

The *states* of the *RS232RX FSM* are shown at the bottom row of the figure. When *idle*, nothing is done. When a startbit is detected, the RS232RX FSM changes state to *startbit*. The delay from *idle* to *startbit* is less or equal to 20 ns. This is a relatively short time compared to one UART clock cycle, and no special actions are acquired for this. The following RS232RX FSM states are each 0.55 ns shorter than the UART clock. This is a relatively short mismatch.

Because the bits are sampled half way, the mismatch may be ignored. The stopbit does not need to be sampled, so the RS232RX FSM is leaving the *stopbit* state after half a UART clock cycle. Thus, the stopbit is 0.5, as shown in the third column of Table 5.1. Doing it this way will ensure that the FSM is guarantied to be ready for the next startbit from the UART. In summary, the RS232RX has a slightly faster baudrate than the UART as indicated in the third column of Table 5.1.

The RS232RX module is shown at block level in Figure 5.11. The *RS232RX FSM* is automatically started by the incoming startbit through *RXD*. When one byte has been received, it is indicated by the *Byte Ready* signal. The *Incoming Byte* may then be read.

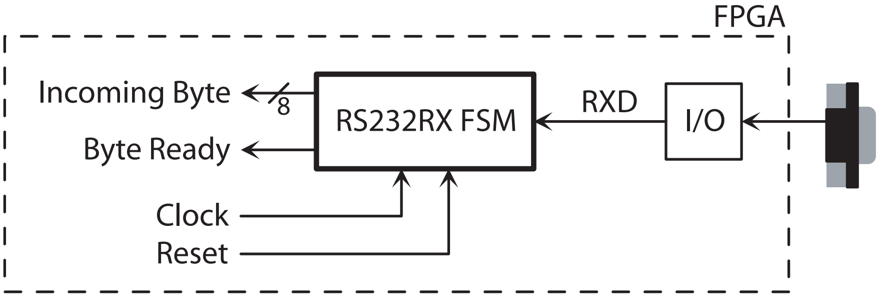


Figure 5.11: Block Level Diagram of the RS232RX Module

RS232TX

The middle row of Figure 5.12 illustrates an *FPGA* when sending one byte. When no byte is transmitted, or when the transmission is completed, a logical ‘1’ is continuously outputted.

The top row illustrates a *UART* when receiving a byte. It is assumed to behave as illustrated in the figure. That is, when it receives a startbit, the *UART clock* is synchronized to rising edge. Then 10 samples are made at the following 10 falling edges, as illustrated by the arrows. Because the samples are made half way through one clock cycle, it doesn’t matter that the RS232TX FSM is slightly faster than the UART.

The *RS232TX FSM* is illustrated in the bottom row. An extra delay is inserted after the stopbit, to ensure that the UART is always ready to receive a new byte.

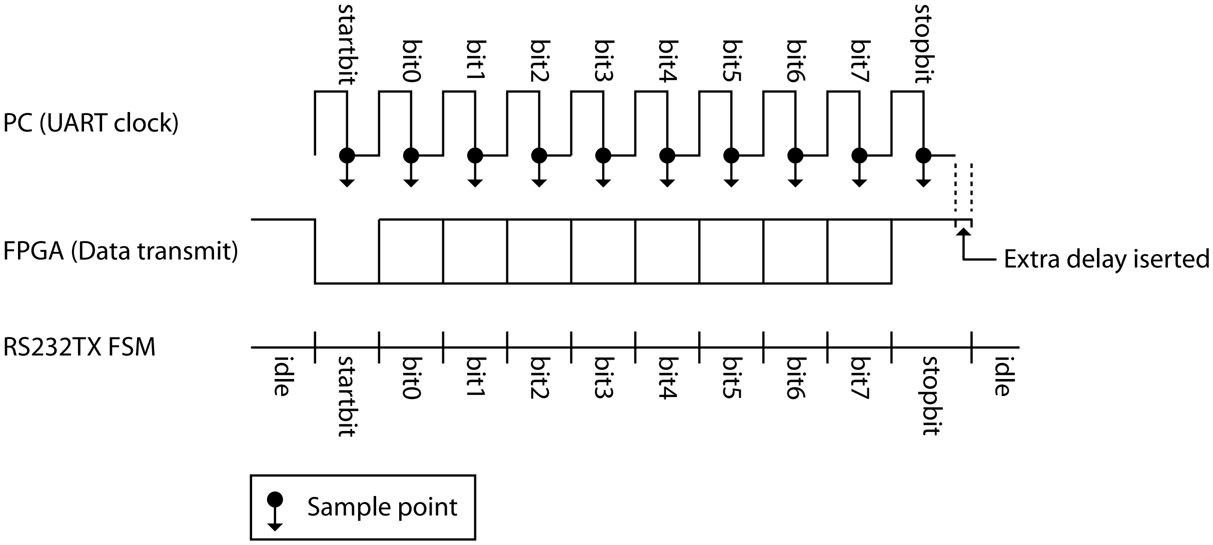


Figure 5.12: RS232TX vs. UART

The RS232TX module is shown at block level in Figure 5.13. When the *RS232TX FSM* is *ready*, a *start* signal may be given. The *Outgoing Byte* to be transmitted is then sampled by the *RS232TX FSM* and transmitted one bit at a time through *TXD*.

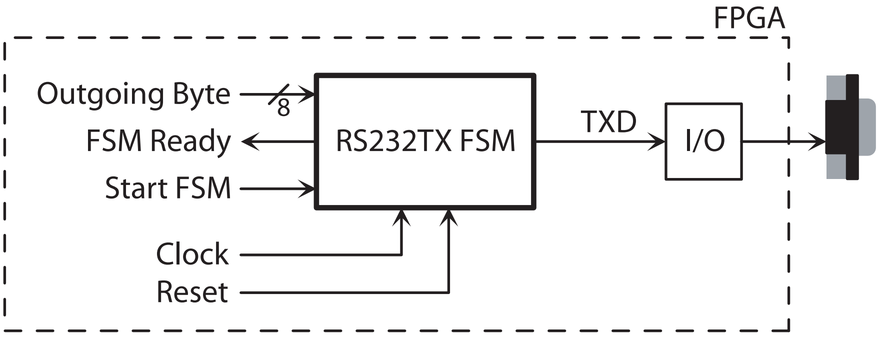


Figure 5.13: Block Level Diagram of the RS232TX Module

Testing the RS232 Communication between the PC and the FPGA

Figure 5.15 illustrates how the RS232 communication was tested and verified in hardware. The grey zone is a simplified view of the test design. All FSMs are attached to the oscillator and the reset push button. A simple user interface was made to run on the PC. This interface was used to send bytes (typed on the keyboard) to the RS232RX module. It also received bytes from the RS232TX module, and displayed on the monitor as characters.

When sending one byte from the *PC*, the byte is transmitted through the *UART* and then into the *RX232RX FSM*. When one byte has been received by the RS232RX FSM, the byte is passed on to the *LCD FSM*. The signal in the RS232RX FSM that indicates when data is ready is used as a *start signal* to the LCD FSM. Also, this signal goes to a *LED FSM*, thereby making the attached LED flash every time a new byte is received. This is done in case the LCD FSM should have been set up wrong. The Register Select (*RS*) of the LCD FSM is set to '1'. Thus, every byte is written to the data register of the LCD and then printed out as a character. The speed of sending a byte from the PC to the FPGA is in this case limited to the rate the keyboard is set up to repeat characters. If it should have been necessary, data could have been read from file and then transmitted at maximum speed.

The *Data* sent from the FPGA to the PC, is determined by the DIP-Switch. To begin with it was advantageous to send one byte at a time. Each byte could then be inspected as it was received by the UART and displayed on the monitor. A push button connected to a *Push Button FSM* was used as the start signal to the *RS232TX FSM* to accomplish this. Also, the start signal was connected to a *LED FSM* so that it was possible to inspect that a start signal had been given.

Sending bytes at maximum speed from the FPGA to the PC is accomplished by another push button. This push button is not connected to a Push Button FSM. When pushed, it is therefore passing on a continuous start signal to the RS232TX FSM. Then every time the RX232TX FSM is ready, the byte specified by the DIP Switch is transmitted to the PC.

In this design it was necessary to do some debugging in hardware, because there was one bug left after simulation. The bug behaved in such a manner that sometimes one byte was not sent or received correctly (both sides of the RS232 connection). The debugging was done such that many identical bytes were transmitted for some time; then observing the results. At first, the startbit seemed to have been lost. Making small changes in the design (RS232 modules) fixed the problem, but introduced a new one. It now appeared as if the stopbit were sampled by the RS232RX. The bug turned out to be a latch. That is, a memory bit that is updated asynchronously rather than on the rising edge of the clock. For synchronization the latch was replaced with a D flip-flop (register), and the bug disappeared.

The time spent for this debugging took a little more than one week. At a later stage of this project, it was proven to be time well spent. In the final design, as shown in Figure 4.16, only one LED was needed for debugging. It was connected to a signal that should have made the LED flash, but it was glowing continuously instead. It could have taken months to have this bug discovered if there should have been bugs in any of the RS232 modules.

Compatibility Mismatch between the UART, RS232RX and RS232TX

The RS232RX and RS232TX modules are each made to be compatible with a UART. Because of differences in the *startbit* and *stopbit* states of the two modules, there is one special case were incompatibility occurs; Figure 5.14.

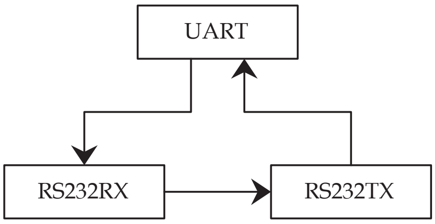


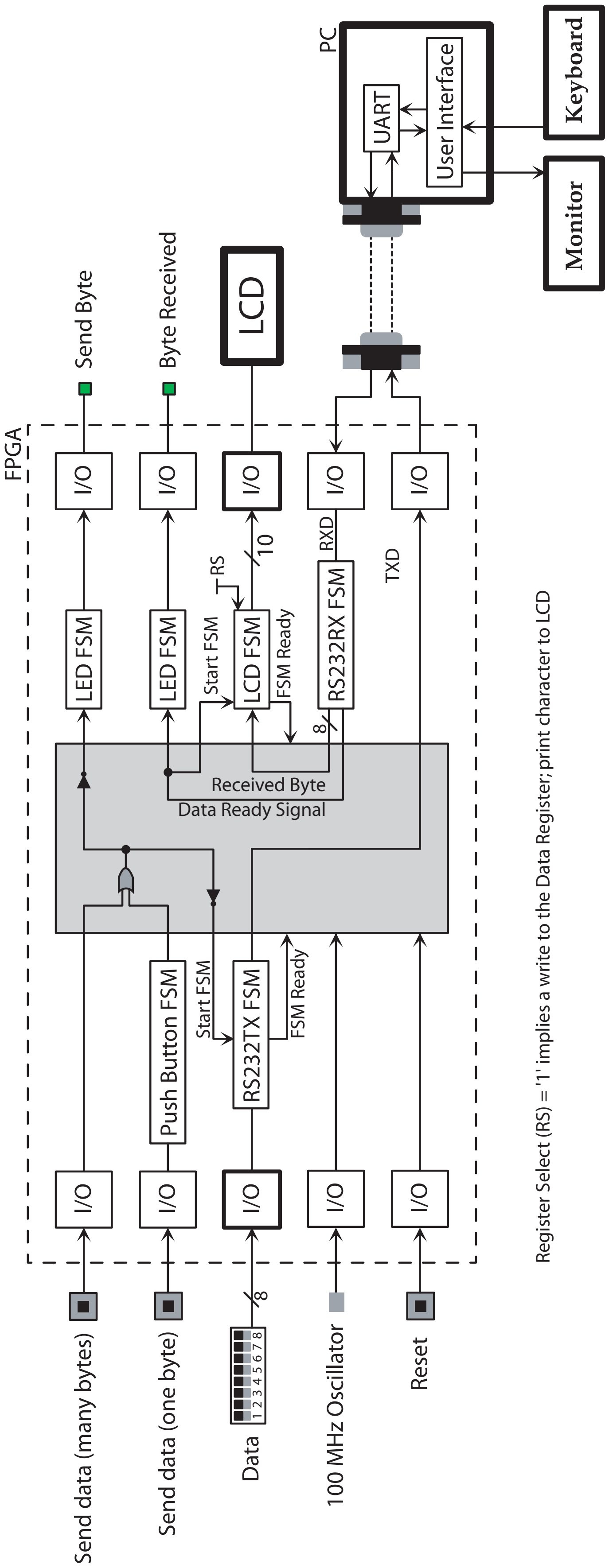
Figure 5.14: Compatibility Mismatch

The figure illustrates a *UART* that sends data at full speed to an *RS232RX* module. The data is then passed on to a *RS232TX* module, before it is sent back to the *UART*. As can be seen from the last column of Table 5.1, the RS232TX has a baudrate slower than 115200. Thus, the mismatch occurs in the RS232TX module. By studying the middle illustrations of Figure 5.10 and Figure 5.12, the mismatch becomes obvious. This mismatch will always occur in the modules presented here, no matter what baudrate is chosen.

However, the special case described above is of no interest for this project. The reason for why this case is mentioned at all is because the setup in Figure 5.14 is one obvious way of testing the relationship between the UART, RS232RX and RS232TX. By sending a continuous datastream from the UART, *data sent* could be compared with *data received* by a simple program on a PC.

5.1.7 MAX3221C / MAX3223C

The *MAX3221C* and *MAX3223C* are Integrated Circuits (ICs) that drives the RS232 signals between the FPGA I/0s and the UART. They can drive signals at a maximum speed of 250 Kbit/s. There is no way to (re)configure these ICs after the have been soldered onto the *Development Board*. The reason for commenting these ICs is only to underscore what components are needed for making an RS232 connection. These ICs are illustrated in Figure 4.16 and Figure 5.1.



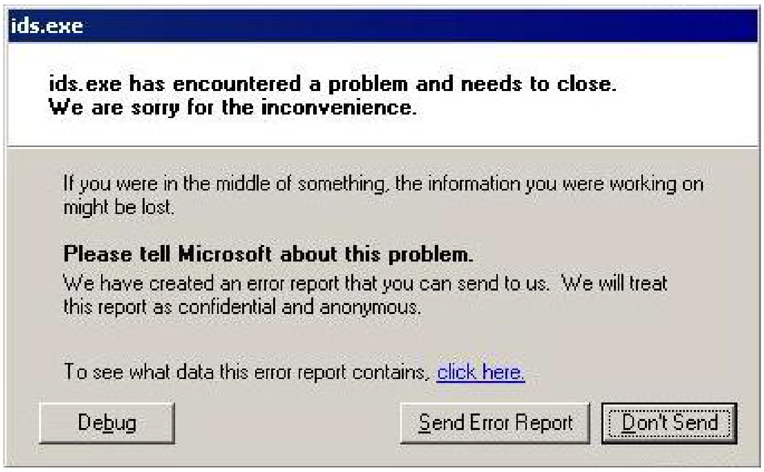
Register Select (RS) = '1' implies a write to the Data Register; print character to LCD

Figure 5.15: Test of the RS232 Communication

5.2 Software Solutions

The Operating System (OS) on the PC running the *User Interface* (*ids.exe*) as shown in Figure 4.16 was *Windows 2000 Professional* (later upgraded to *Windows XP*), while the design shown in Figure 5.15 was run under *Windows 95*. The programming language chosen was *C*, because skills with this language were already at hand. Thus, *Microsoft Visual C++ 6.0* was first chosen as compiler for *ids.exe*.

Unlike *Windows 95/98*, *Windows NT/2000/XP* has strict control over I/O ports [21]. Making a program that uses the serial ports, like *ids.exe*, is not easy to run under *Windows NT/2000/XP*. *ids.exe* is designed to run at the command prompt. When starting *ids.exe* from *Windows XP*, an error message is displayed; see Figure 5.16 (top). Doing a *debug run* from the compiler itself does not work out any better as shown in Figure 5.16 (bottom). Having administrator privileges did not make any difference.

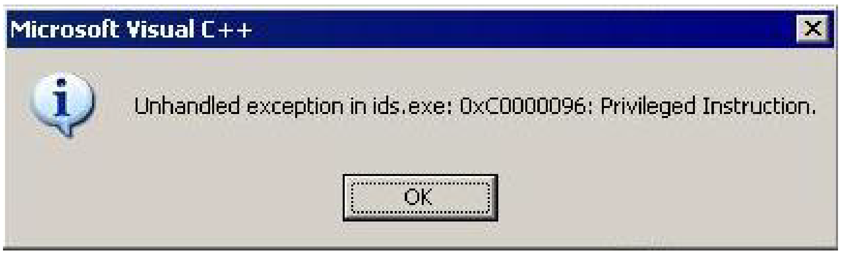


ids.exe has encountered a problem and needs to close.
We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.



Unhandled exception in ids.exe: 0xC0000096: Privileged Instruction.

Figure 5.16: Access to Serial Ports are Denied With Little Informative Messages

Open source code to come around this problem is available. A driver (*porttalk.sys*) and a program that uses the driver (*allowio.exe*) are both available at [21]. In order to use the driver without installing it, administrator privileges are needed. Otherwise, the driver must be installed by an administrator and the machine must be restarted in order to load the driver. It is then available for any user.

An example of how to use these programs is "*allowio.exe ids.exe 0x3f8 0x2f8*". *allowio.exe* will open the ports *0x3f8* (COM1) and *0x2f8* (COM2) for use with *ids.exe*. It uses *porttalk.sys* internally to open the ports specified at the command line for the application specified (*ids.exe*). After sending this command to the OS from the command prompt, it returns to the command prompt rather than printing the menu in *ids.exe*; see Figure 5.18. Then, when giving the first command to *ids.exe*, the OS returns an error message saying that an unknown command was given at the command line. The second command will be sent to *ids.exe* as intended. This process repeats itself as long *ids.exe* is running.

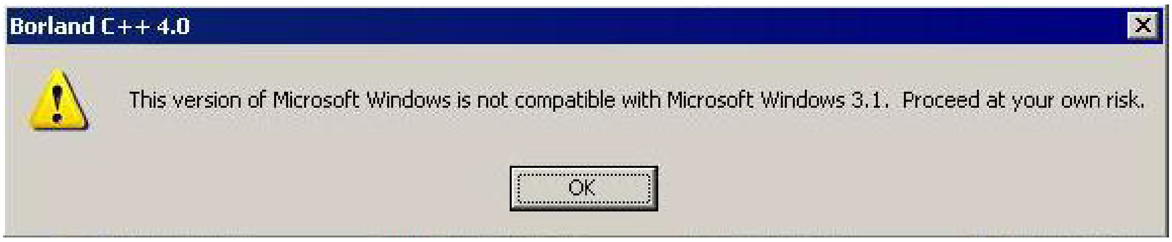
At first this seemed to be a peculiar bug. By doing some experiments, it is appeared that *allowio.exe* expects that a new window is created from *ids.exe*. By changing the source code of *allowio.exe* in order to be compatible with programs running from the command line, could have fixed this problem. Even though the source codes of *allowio.exe* and *porttalk.sys* are short, it takes considerable time to make changes to them because skills in *C++* are acquired. Another way of solving this would be to make a program that creates a new window when started. However, skills in *C++* would have been needed for this.

To come around these problems, a rather unconventional solution was tried out. *Windows 95* was installed at an old PC. The compiler chosen for this PC was *Borland C/C++ 4.0*. This is a compiler that was originally designed for *Windows 3.1*. It compiles 16-bit executables. Making a design that uses the serial ports, with these tools, turned out to be a trivial matter. Now, the menu of ids.exe was printed out; see Figure 5.18. *ids.exe* was executed with no problems at all, and the design in Figure 5.15 was tested and debugged as described in conjunction with that figure.

It was desirable to run *ids.exe* under *Windows XP*. *Borland C/C++ 4.0* has an option that compiles command line programs, like the one shown in Figure 5.18, into an *EasyWin* program. When starting an *EasyWin* program from the command prompt, a new window is created. It contains exactly the same as we would expect from a program designed to be run at the command line. Now, by compiling *ids.exe* with the *EasyWin* option and then copy it over to the PC running *Windows XP*, it was expected to run without problems by using *allowio.exe* and *porttalk.sys*. It did not turn out as expected. A new window was created, the menu was printed out as in Figure 5.18 and no error message was given. However, there was no way of making any communication through the serial ports. Many attempts were made to put the *EasyWin* version of *ids.exe* to work. Compiler options in *Borland C/C++ 4.0* were explored, the source codes of *porttalk.sys* and *allowio.exe* were again examined and the UART setup in *ids.exe* was checked over and over again. Details of UART are given in Appendix E.

Each time a new version of *ids.exe* was made, it had to be copied from the old PC to the new one. By accident, a version of *ids.exe* that was based on running from the command line (not *EasyWin*) was copied. When starting this version under *Windows XP*, it gave no error messages as the ones shown in Figure 5.16. This turned out to be quite a surprise, but it was expected not to be able to making any communication at all because *allowio.exe* had not been used. By testing the communication anyway, it turned out to work perfectly well.

By copying *Borland C/C++ 4.0* over to a hard disk that is available by using *Windows XP*, the old PC could be removed. When starting *Borland C/C++ 4.0* under *Windows XP*, a warning is given; see Figure 5.17. Of course, running old software like this compiler under *Windows XP* is far from ideal. It was good enough for use with this project, but for future projects other solutions should be tried out. A redesign of *porttalk.sys* and *allowio.exe* would be a good (but time consuming) start.


Borland C++ 4.0

This version of Microsoft Windows is not compatible with Microsoft Windows 3.1. Proceed at your own risk.

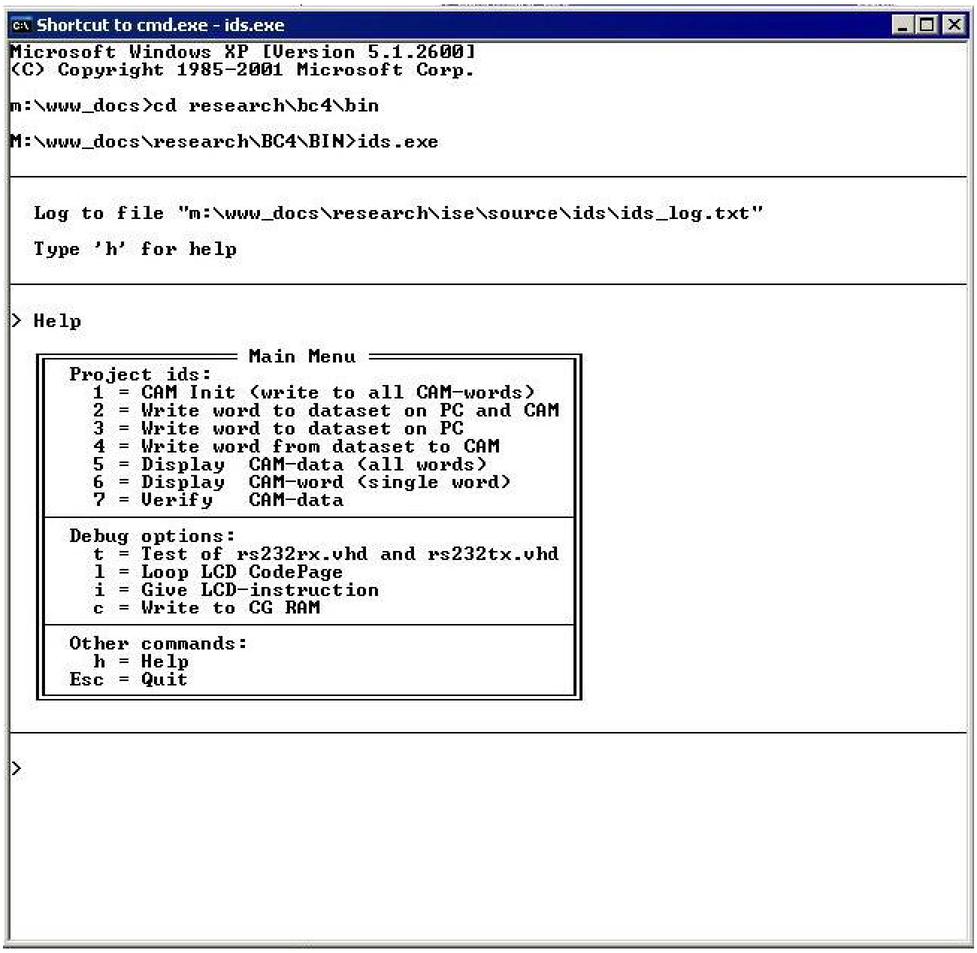
Figure 5.17: Running Borland C/C++ Under Windows XP

By taking a closer look at [21] it was found that 16-bit Windows programs, when running under a 32-bit Windows OS (*Windows NT/2000/XP*), will run as it would be expected to run under a 16-bit Windows OS (*Windows 95*). As noted in [21], there may be timing problems when doing this. By experience from testing the design as shown in Figure 4.16, these timing problems were of little concern because they were easily detected. That is, when a timing problem occurred, nothing seemed to work.

The timing problems occurred (rarely) in two ways

- A codepage can be said to be a driver for the keyboard. There is one codepage for each country. The codepage was not loaded so that the keyboard was left undefined. The way to handle this bug was to end *ids.exe*, then manually change the codepage at the command prompt, and then restart *ids.exe*.
- It was not possible to write to CAM. That is, when trying to verify a write to CAM it failed in all cases. A specific solution for this bug was not found. It is highly possible that the solution mentioned above would have fixed this bug too. Further experiments were not acquired because the CAM had been verified, and it was time to start writing this report.

To remove any doubt about having bugs in the FPGA, *ids.exe* could have been run under Windows 95 at an old PC.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

m:\www_docs>cd research\bc4\bin

M:\www_docs\research\BC4\BIN>ids.exe

  Log to file "m:\www_docs\research\ise\source\ids\ids_log.txt"

  Type 'h' for help

> Help
                    Main Menu
     Project ids:
       1 = CAM Init (write to all CAM-words)
       2 = Write word to dataset on PC and CAM
       3 = Write word to dataset on PC
       4 = Write word from dataset to CAM
       5 = Display  CAM-data (all words)
       6 = Display  CAM-word (single word)
       7 = Verify   CAM-data

     Debug options:
       t = Test of rs232rx.vhd and rs232tx.vhd
       l = Loop LCD CodePage
       i = Give LCD-instruction
       c = Write to CG RAM

     Other commands:
       h = Help
     Esc = Quit

>
```

Figure 5.18: ids.exe Running in 16-bit Mode Under Windows 95/2000/XP

6 RESULTS

Only a functional test of this design has been performed; as described in Chapter 4.4. This was made possible by making RS232 modules that could communicate with a UART; Chapter 5. The UART at the PC used for this project has a maximum speed of 115200 Kbit/s when running from the command prompt; see Figure 5.18. As the speed of a string matcher should be measured in Mbit/s or even Gbit/s, the RS232 connection is useless for the purpose of testing the maximum performance of this design.

Testing the maximum performance could have been done by using the SDRAM available at the *Development Board*; see Figure 5.1. This is left as a suggestion for future work due to the time left of writing this report.

Considerations that must be made when making a fast design are *the area used* and *delays* through various parts of the circuit. The synthesize tool (*Xilinx ISE 6.1.03i* in this case) is providing various *reports* to help us making a better design. The *Synthesize Report* gives a good overview over used resources, but the timing estimates are not expected to be achieved in hardware. These estimates are still useful in order to give hints about where in the design there might be a bottleneck. The *Post Place and Route Static Timing Report* gives timing estimates that can be expected to be reliable when running the design in hardware. However, these estimates are far from as detailed as those given in the *Synthesize Report*.

6.1 Estimates of Maximum Performance

When the synthesize tool optimizes a design for speed, it is important to not use all the resources (area) in the device. That will result in less flexibility for the synthesize tool. In this project it is therefore important to be aware of how much of the slices are used, and then compare it with the predicted maximum speed.

Table 6.1 shows two examples (row 1 and row 3) of CAM-designs that have been functionally tested as described in Chapter 4.4. The others are processed down to the bitstream that is used for programming the FPGA, but not tested in FPGA. Assuming we get a match for every incoming packet, it is possible to predict the worst case that a string matcher should be able to handle. This is an important consideration to make when designing a system for detecting and handling massive attacks. In the column reporting max speed, we get a prediction of the worst case of this design. The numbers have been rounded down to the nearest integer. Note that a lower *speed grade* indicates a faster device (-7 is faster than –6, for example).

The following observations are made:

- Small CAMs (134 bytes in this case), implemented in an FPGA with a fast speed grade, are capable of taking an input bit stream of 1 Gbit/s; *row 1*. By using an FPGA with a slower speed grade, the *Max Speed* of the bit stream is significantly less; *row 2*. In both cases, the *Slices Used* is low, which gives a high flexibility of optimization by the synthesis tool.
- Using the same two devices for a larger CAM (1822 bytes), we can see that the two devices now have the same performance; *row 3* and *row 4*. Note that almost all of the slices are used in *row 3*, and that it will give little flexibility for optimization by the synthesis tool. That is why the device in *row 4* is able to run as fast as the one in *row 3*. The reduction of performance for the device in *row 3* is 22%, while in *row 4* the reduction is only 6%.
- The CAM in *row 5* is faster than the one in *row 4*. This is expected because the device in *row 5* has a faster speed grade than the device in *row 4*.
- By increasing the size of the CAM in *one* type of device from 1822 bytes to 3601 bytes, the performance is reduced by 5%; *row 4* and *row 7*. We find the same reduction by comparing *row 5* with *row 8*.
- By comparing *row 6* with *row 9*, we find that the performance reduction is less than for the comparisons in the previous item; 2%.
- The maximum number of words is 256, and will be further discussed in Chapter 6.2.3.

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

Table 6.1: A Selection of Some Successfully Implemented Designs

These results are achieved by processing 8 bits per clock cycle. By making a design that can process 32 bits per clock cycle, the expected throughput will be *4 x 800 Mbit/s = 3.2 Gbit/s*. The design in [11] as described in Chapter 2.5 also processes 8 bits per clock cycle, but at a rate of only 296 Mbit/s. The key to its high performance of 1.184 Gbit/s was to connect 4 modules in parallel.

A CAM takes up a large area in an FPGA; Chapter 4.2.3. For this reason it is not possible to implement all strings in the Snort rules in one FPGA. The design in [3] uses a method to implement many strings in one expression, thereby saving area in the FPGA. The cost of this is speed; all reported results from this design are less than 7 Mbit/s.

6.2 Considerations of performance

6.2.1 The Delay Through one Word in CAM

The FPGA used in this project has 160 slices in a column, thereby defining the max width of a word to 160 bytes; see Figure 4.9. It would be interesting to see if the path through one word could be optimized. Table 6.2 (refer to Figure 4.8, Figure 4.9 and Figure 4.10) shows delays through a 32-byte word. The data are taken from the *Synthesis Report* where only the CAM files have been synthesized.

	Path	Gate Delay (ns) (logic)	Net Delay (ns) (route)
SRL16E	CLK → Q	2.720	0.360
MUXCY	S→ COUT	0.334	0
Long chain of MUXCYs	CIN → COUT	***0.036***	0
Last MUXCY	CIN → COUT	0.600	0.360
Synchronization Register	-	0.208	-
Sum:		6.844	

Table 6.2: Delays Through One 32-Byte Word

From the table (the green cell) we can se that increasing the length of the carry chain (through the MUXCYs) adds a delay of *0.036 ns* for each MUXCY that is added. This should be taken in consideration when designing a new CAM. By adding any more of the other parts will not affect the total delay through the word. From this table there is no obvious way of further optimization for speed.

6.2.2 The Read and Write Path

The read and write procedures never occur simultaneously in this design. It is therefore of great interest to know the delay through these paths separately. It is the read path that is of greatest interest; as outlined in Chapter 4.2.1. It can probably be read out of the various reports generated by the synthesis tool. However, it takes more experience than what is at hand to read out this information. If it is so that the read path is faster than the write path, these paths should be treated separately. Two different clocks could have been used, for example. Future work will include finding these paths, and then give better estimates for maximum performance.

6.2.3 Optimizing the Encoder

When trying to synthesize a 512-word CAM, the synthesis tool worked for about one hour before giving a strange error message. The message was that an unknown error had occurred and that 2Gbyte of system RAM was not enough to complete this job. The computer was equipped with 1Gbyte system RAM, and the virtual memory was set to 3Gbyte. This problem occurred when synthesizing the encoder.

The encoder is one component that obviously can be optimized; see Figure 4.10. In this project, the encoder is described in VHDL as a *standard encoder*; p109. An encoder like this takes a lot of resources. It outputs an address if there is *exactly one* match. To check if there is exactly one match, it must also be checked that all the other inputs are low. It is this check that consumes resources.

Since the data chosen for CAM in this project only produces *one* match at a time as described in Chapter 4.2.1, the encoder should better be described *"one-hot"*; p176. A "one-hot" encoder is such that it does not check whether the other inputs are low when a match is received. As a little experiment, a 32-word "one-hot" encoder was made and synthesized by itself. Also, a standard 32-word encoder was synthesized by itself. From the *Synthesis Report* of these two encoders, it could be seen that the "one-hot" encoder was about twice as fast as the standard one. When using fewer resources for the encoder, such as a "one-hot" encoder, it is most likely that CAMs as described in this report can be synthesized with many more words than 256. Even better, the designs are expected to perform faster also.

7 CONCLUSION

A variable word-width CAM has been designed that is well suited for 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. The flexibility of such a CAM, and the short time needed to make the VHDL-files, will be important for NIDSs since they need to be changed frequently for research purposes. The implemented architecture functionally tested on the hardware platform that was chosen for this project can process 128 words (1822 bytes) in parallel at 800 Mbit/s. With an optimization of the encoder it is most likely that the speed of this CAM will increase, and also that more words will fit into the FPGA. Future work involves making an even more flexible CAM. It should be able of changing the number of words, and the length of each word runtime. That is, a new CAM should be able to be made without producing new VHDL files.

8 APPENDIXES

A CD-ROM

This document in

- Word 2003
- Pdf

Source code

- All source code used in this project in directory *source*
- All source code converted to HTML in directory *html*

Borland C/C++ 4.0 (Designed for Windows 3.1)

- An installed version is found in directory *BC4*
- Original source for installation is found in directory *BC4_install*

B Exploring the Properties of an SRL16E by Simulation

Figure 8.1 shows the waveforms from the simulation of an SRL16E by using ModelSim 5.6f provided by ModelTech. The simulation is created by a VHDL-design. By giving this design stimuli from a testbench (also written in VHDL), we are able to give input and measure output of the design just as if we were testing it by hardware.

Three observations/conclusions are made:

1. At the first rising edge CE is set high for the next 16 clock cycles. The address is set to 7 ("0111"), and remains at this value as long as CE is high. The output Q asserts the input values after 8 clock cycles. Conclusion: The address controls which of the 16 registers to output on Q.
2. CE is set low, but the address remains the same. If shifting was enabled, we would expect to find a low output of Q at the indicated point. But the output is high. Conclusion: The value found at this point is the value that the register at address 7 had the time CE went low. CE can therefore be named as "Shift Enable" as well as "Clock Enable".
3. At 285 ns the address starts counting from 0 through 15, one count each clock cycle. The values on the output Q appear to be the reversed values to that of input D during the first 16 clock cycles. Conclusion: The data is always shifted in on the least significant address. The chosen address have no influence on which register is being written to or how the shifting is done, it only controls the output Q.

Based on the above observations/conclusions it is now possible to make a more detailed understanding of an SRL16E. Figure 4.7 shows how an SRL16E might look like inside.

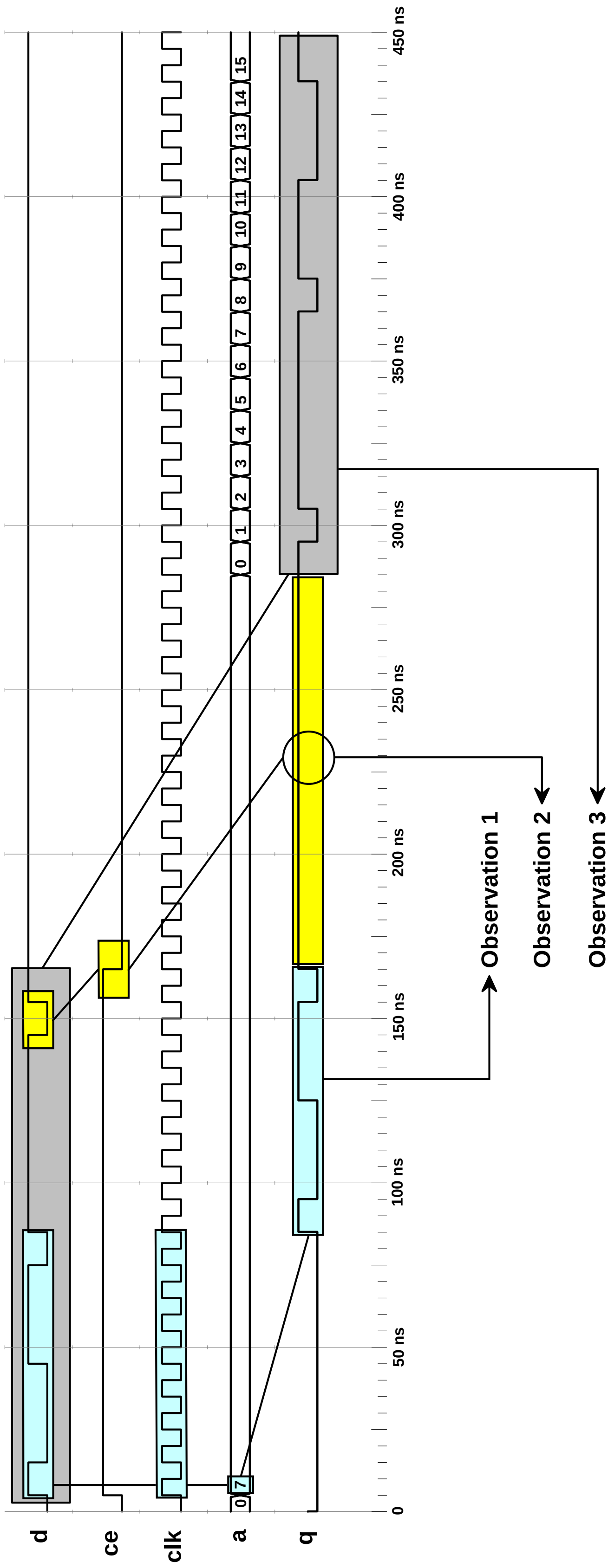


Figure 8.1: Simulation of One SRL16E

C A State Diagram for a Non-Sequential FSM

Figure 8.2 illustrates the FSM from Chapter 5.1.3 by using a state diagram. Whenever we have an FSM that is not sequential, such as this, a state diagram is often needed. It is possible to write the VHDL code (almost) directly out of a state diagram. Also, far less time will be spent on debugging. That is, we will debug the state diagram rather than the VHDL code.

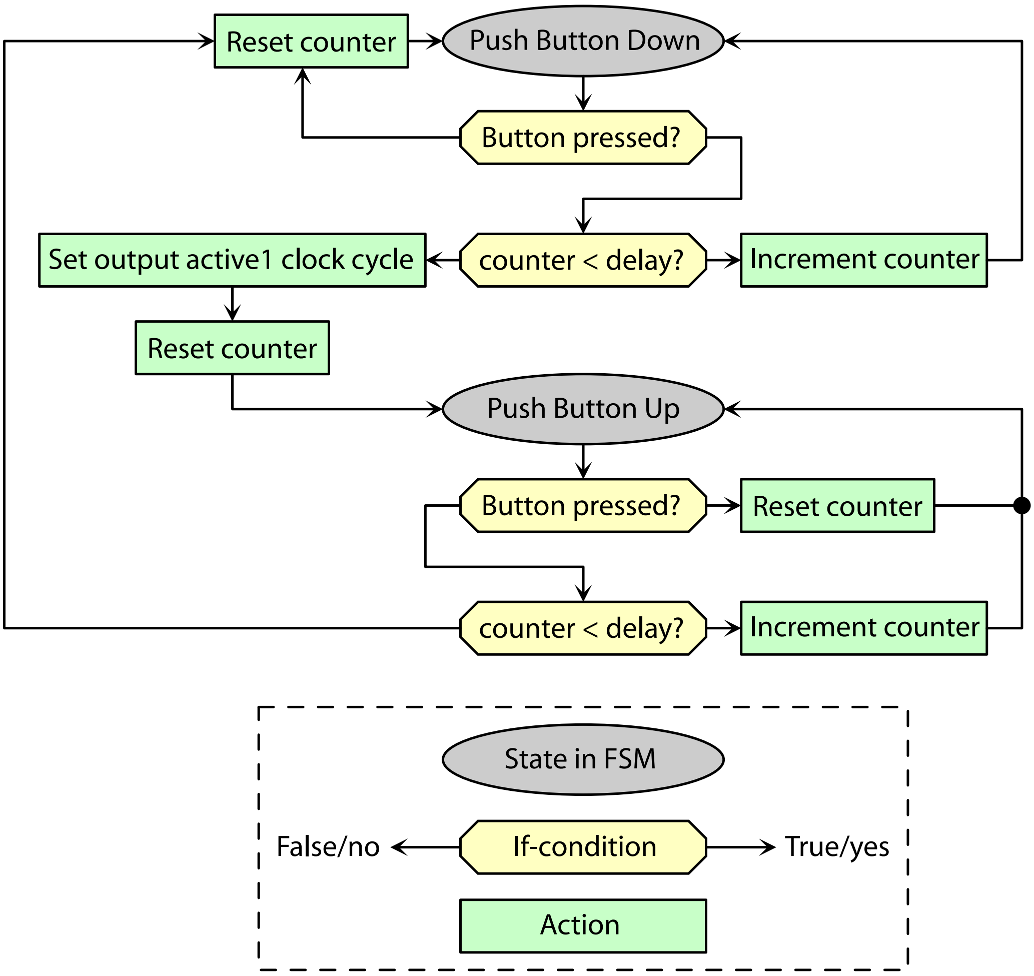


Figure 8.2: Push Button FSM Diagram

D A Way To Describe a Sequential FSM

There is no need to use state diagrams (Appendix C) for FSMs that can be described sequentially. The FSM described in 5.1.5 is sequential. It is not necessarily easier to write a sequential FSM, as is the case with this one.

Table 8.1 gives timing parameters that must be met by the FSM. Figure 8.3 illustrates how to use these parameters [22]. The bottom row of Figure 8.3 illustrates the *LCD FSM*. As with state diagrams, most of the job is done when having illustrated the FSM like this. The tricky part here is that it is easy to code one or more of these parameters wrong in the VHDL code, and nothing works.

	Symbol	Min (ns)	Max (ns)
Enable cycle time	$t_{cyc}E$	500	-
Enable pulse width	PW_{EH}	230	-
Enable rise and fall time	t_{ER}, t_{EF}	-	20
Address setup time (RS, R/W enable)	t_{AS}	40	-
Address Hold Time	t_{AH}	10	-
Data Setup Time	t_{DSW}	80	-
Data Hold Time	t_H	10	-

Table 8.1: LCD Write Timing Parameters

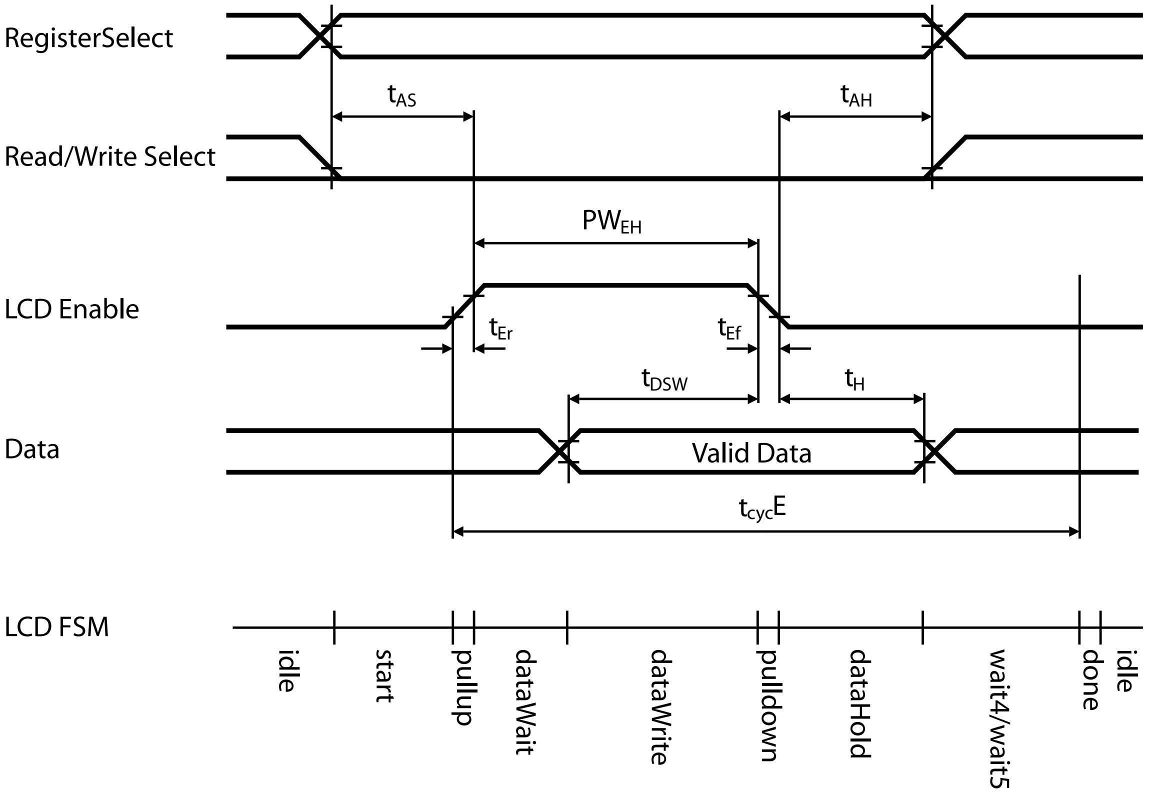


Figure 8.3: LCD Write Timing Diagram

E Details of the UART

I/O Port	R/W	Note	Bit 7	Bit 6	Bit 5
03f8	W	DLAB bit = 0: Serial port 1 transmit holding register	Byte to transmit		
“	R	DLAB bit = 0: Serial port 1 receive buffer register	Receive character		
“	R/W	DLAB bit = 1: LSB of serial port 1 Divisor Latch	LSB of BAUD rate divisor		
03f9	R/W	DLAB bit = 1: MSB of serial port 1 Divisor Latch	MSB of BAUD rate divisor		
“	R/W	DLAB bit = 0: Serial Port 1 interrupt enable register	Reserved		
03fa	R	Serial port 1 interrupt ID register	11 = FIFO feature present 10 = FIFO feature not present		Reserved
“	W	Serial port 1 FIFO control register	Receiver FIFO register trigger 00 = 1 byte 01 = 4 bytes 10 = 8 bytes 11 = 14 bytes		Reserved
03fb	R/W	Serial port 1 line control register	DLAB (Divisor Latch Access Bit). 0 = Receive buffer, transmit holding, interrupt enable register access enabled. 1 = Divisor Latch Access enabled	1 = Set break enabled	1 = Stick parity
03fc	R/W	Serial port 1 modem control register	Reserved		
03fd	R	Serial port 1 line status register	Reserved	1 = Transmit shift and holding registers empty	1 = Transmit holding register empty
03fe	R	Serial port 1 modem status register	1 = Data Carrier Detect (DCD) active	1 = Ring Indicator (RI) active	1 = Data Set Ready (DSR) active
03ff	R/W	Serial port 1 scratch register	Can be used by the microprocessor to hold a byte. It is not used by the serial port		

Table 8.2: UART Serial Ports overview

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1 = Enable modem status interrupt	1 = Enable receiver line status interrupt	1 = Enable transmit holding register empty interrupt	1 = Enable receive data available interrupt. In FIFO mode also enables time-out interrupt
	000 = Modem status interrupt 001 = Transmit holding register empty interrupt 010 = Receive data available interrupt 011 = Receive line status interrupt 110 = FIFO timeout interrupt			0 = Interrupt pending
		1 = Transmit FIFO register cleared; counter cleared bit is self-clearing	1 = Receive FIFO register cleared; bit is self-clearing	0 = Clears receive and transmit FIFO registers and enters character mode 1 = Receive and transmit FIFOs enabled
1 = Even parity enable	1 = Parity enable	Number of stop bits: 0 = 1 stop bit 1 = 1.5 stop bits with char length of 5 bits 1 = 2 stop bits with char length of 6/7/8 bits	00 = Character length is 5 bits 01 = Character length is 6 bits 10 = Character length is 7 bits 11 = Character length is 8 bits	
1 = Loopback mode	1 = Enable OUT2 interrupt	1 = Force OUT1 output active	1 = Request-To-Send (RTS) active	1 = Set Data Terminal Ready (DTR) active
1 = Break interrupt	1 = Framing error	1 = Parity error	1 = Overrun error	1 = Data ready
1 = Clear-to-Send (CTS) active	1 = Change detected on DCD line	1 = Trailing edge of RI signal detected	1 = Change detected on DSR line	1 = Change detected on CTS line

F Source Code

Each subchapter gives a brief description of the files.

F.1 CAM

These files are those needed for simulation and hardware implementation of a CAM.

- *camdata.pl*
    - Parses the Snort rules for content parts. These parts are stored in *camdata.txt*. Only a small part of *camdata.txt* is included here.
- *cam_vhdl.pl* produces the remaining files in this section. In this project *camdata.txt* was given as input to cam_vhdl.pl
    - cam_top.vhd
        - Top level module for a CAM. Binds together the files below.
    - cam_words.vhd
    - cam_word.vhd
    - cam_basic.vhd
    - compare.vhd
    - counter.vhd
    - decode.vhd
    - encode.vhd
    - components.vhd
- tb_cam_top.vhd
    - The testbench. Stimuli are automatically generated with *cam_vhdl.pl.*
- tb_cam_top.fdo
    - Script for running the testbench in ModelSim. Generated with *cam_vhdl.pl* so that it runs exactly the time needed to come through *one* simulation

camdata.pl

```
#!/local/bin/perl5 -w

#
# File   : cam_content.pl
# Author : Geir Nilsen { geirni@ifi.uio.no }
# Created: Sep 3 2003
#
# Description:
#   Extract bit-pattern from content-part of Snort-rules.
#   Choose rules that have only one content-part.
#   Store distinct patterns only.
#   Choose length of shortest and longest pattern to store.
#

$rulesdir  = "/hom/geirni/www_docs/research/snort202_win32/Snort/rules";
@rulefiles = `ls $rulesdir/*.rules`;
$camfile   = "camdata.txt";

$minLength = 4;  # Bytes
$maxLength = 32;

# Find content-part of rules
for $rulefile(@rulefiles){

    open(INFILE, "<".$rulefile) or die
        "Can't open ".$rulefile."\n";
    @rules = <INFILE>;
    close(INFILE);

    for $rule(@rules){

        $contentParts = 0;

        if($rule =~ /content:/){
            @parts = split(/;/, $rule);
            for $part(@parts){
                if($part =~ /content:/){
                    $content = $part;
                    $contentParts++;
                    # Remove anything before content-part
                    $content =~ s/^.*content:.*?\"//i;
                    # Remove anything after content-part
                    $content =~ s/\"$.*//g;
                }
            }
        }

        # Store content-part
        if ($contentParts == 1){
            push(@contents, $content);
        }
    }
}



# Convert content-strings to hex. Store only distinct patterns
for $content(@contents){

    $pipe    = 0;  # hex patterns are limited by pipes; |00 bc 55|
    $char    = ""; # Current character in content; ASCII or hex
    $pattern = ""; # Content converted to hex

    # Loop through current content-string
    for ($i=0; $i<=length($content)-1; $i++){ # -1 for newline

        $char = substr($content, $i, 1);

        # Control over pipes
        if($char =~ /\|/){
            if(!$pipe){
                $pipe = 1;
            }
            else {
                $pipe = 0;
            }
            next; # Skip to next character
        }

        # Convert to lowcase hex
        if(!$pipe){               # ASCII-value
            $pattern .= sprintf("%x", ord($char));
        }
        else {                    # hex-value
            $char =~ s/ //;       # Remove blanks
            $pattern .= "\l$char";
        }
    }

    # Store converted pattern
    if((length($pattern) >= $minLength*2) &&
       (length($pattern) <= $maxLength*2)){
        $hexPatterns{$pattern} = "dummyValue"; # Keys will be distinct
    }
}



# Print patterns, that have no subsets, to file
open(OUTFILE, ">".$camfile) or die
    "Can't open ".$camfile."\n";

@patterns = keys %hexPatterns;
$count = 0; # Count patterns that are written to file

HEXLOOP:
for($i=0; $i<=$#patterns; $i++){
    for($j=0; $j<=$#patterns; $j++){ # Search for subsets

        next if($i==$j); # Do not compare a pattern with itself

        next HEXLOOP if  # Skip if subset is found
            ((length($patterns[$i]) <= length($patterns[$j])) &&
             ($patterns[$j] =~ /$patterns[$i]/));
    }
    print OUTFILE $patterns[$i]."\n";
    $count++;
}

close(OUTFILE);



# msg
print
    "\n".
    "    Wrote ".$count." patterns to file: \"".$camfile."\"\n".
    "\n";
```

camdata.txt

```
66696c656e616d653d5c466978323030312e6578655c
6c736f66253230
6a6176617363726970745c3a2f2f
64313368685b
2f6578616d706c65732f736572766c65742f536e6f6f70536572766c6574
616c6c5f7461625f636f6c756d6e73
0f0000000373686f7720646174616261736573
66696c656e616d653d5c4355504944322e4558455c
2e6173702e
2f766965772d736f75726365
4142434445464748494a4b4c4d4e4f505152535455565741424344454647484849
02010004820100
2e2e5c5c
2f776169732e706c
496e646578206f66202f6367692d62696e2f
ce63d1d216e713cf39a5a586
2f73746f72652e636769
```

```
416d616e6461

.
.
.

2f2e70657266
901ac00f900220089202200fd023bff8
2f7066646973706c61792e636769
2e6874706173737764
6e6f6e676d696e5f636e
2f64666972652e636769
2f616473616d706c65732f636f6e6669672f736974652e637363
3f45646974446f63756d656e74
666574697368
2f69697370726f746563742f61646d696e2f5369746541646d696e2e617370
2f6e7374656c656d657472792e616470
6261636b646f6f72
2f75706c6f61642e706c
616c6c5f636f6e73747261696e7473
```

cam_vhdl.pl

```
#!/local/bin/perl5 -w

################################################################################
################################################################################
##                                                                            ##
##   File   : cam_vhdl.pl                                                     ##
##   Author : Geir Nilsen { geirni@ifi.uio.no }                               ##
##   Created: Sep 4 2003                                                      ##
##                                                                            ##
################################################################################
################################################################################
##                                                                            ##
##   Contents:                                                                ##
##                                                                            ##
##    - Global variables                                                      ##
##    - Functions                                                             ##
##    - Generate variables necessary for creating project files              ##
##    - Create files:                                                         ##
##        Top level module:    cam_top.vhd                                    ##
##                  module:    |-> cam_words.vhd                              ##
##                  module:    |--> cam_word.vhd     (1)                      ##
##                  module:    |---> cam_basic.vhd   (1)                      ##
##                  module:    |-> compare.vhd       (1)                      ##
##                  module:    |-> counter.vhd       (1)                      ##
##                  module:    |-> decode.vhd                                 ##
##                  module:    |-> encode.vhd                                 ##
##                  testbench: tb_cam_srl16e.vhd                              ##
##                  package:   components.vhd                                 ##
##    - Print info to screen                                                  ##
##    - Extra stuff. Delete ???                                               ##
##                                                                            ##
##    (1) File need to be created only once.                                  ##
##        For ease of use it is recreated anyway:                             ##
##        Entities must be given in these files to produce correct code       ##
##        in components.vhd. Doing it this way, it is easy to make changes    ##
##                                                                            ##
################################################################################
```

```
################################################################################
##                                                                            ##
##   General description:                                                     ##
##                                                                            ##
##   Create VHDL source files for a CAM spesified at the command line. There  ##
##   are no restrictions of how many CAM-words to create, and each CAM-word   ##
##   may have any width. The physical dimensions of a spesific hardware gives ##
##   the maximum size.                                                        ##
##                                                                            ##
##   A CAM-word is designed by serialconnecting SRL16Es. The CAM-words are    ##
##   then connected in parallell.                                             ##
##                                                                            ##
##   Choose number of patterns to read from "cam_content.txt", a file that    ##
##   contains patterns extracted form Snort-rulefiles (= number of CAM-words) ##
##   Default startposition in file is 0.                                      ##
##                                                                            ##
##   The testbench is primarily designed to work with Snort-rules or          ##
##   equivalent                                                               ##
##                                                                            ##
################################################################################
################################################################################

################################################################################
################################################################################
##                                                                            ##
##                           Global variables                                 ##
##                                                                            ##
################################################################################
################################################################################

$author            = "Geir Nilsen { geirni\@ifi.uio.no }";
$prjName           = "cam_srl16e";
$filename          = "cam_vhdl.pl";

$sim_path          = "m:/www_docs/research/ise/simulation/cam";

$prjFileCam_top    = "cam_top.vhd";
$prjFileCamCells   = "cam_words.vhd";
$prjFileCamGeneric = "cam_word.vhd";
$prjFileCam_basic  = "cam_basic.vhd";
$prjFileCompare    = "compare.vhd";
$prjFileCounter    = "counter.vhd";
$prjFileDecode     = "decode.vhd";
$prjFileEncode     = "encode.vhd";
$prjFileTb         = "tb_cam_top.vhd";
$prjFileComponents = "components.vhd";
$prjFileFdo        = "tb_cam_top.fdo";

$totNumOfPatterns  = 0;  # Number of patterns found in file
$posOfFirstPattern = 0;  # Pos in file containing patterns
$posOfLastPattern  = 0;

@header            = (); # Contains info about project. Header of project files
%entities          = (); # Stores content of entities for use with
                         # components.vhd

@patterns          = (); # Patterns read from file. Each pattern is one word in
                         # CAM
@lengthOfPatterns  = (); # Length of patterns in LUTs. 1 LUT = 4 bits
$numOfPatterns     = 0;  # Patterns loaded
$numOfAddrBits     = 0;  # #bits needed to represent #patterns loaded
```

```
$longestPattern    = 0;  # Given in #bits
$camSize           = 0;  # Given in bytes

################################################################################
################################################################################
##                                                                            ##
##                           Functions                                        ##
##                                                                            ##
################################################################################
################################################################################

# Takes one argument:
# Number of elements to represent in binary
sub findNumOfAddrBits{
    my $numOfElements = $_[0];
    my $numOfbits = 0;
    while($numOfElements > 1){
        $numOfElements /= 2;
        $numOfbits++;
    }
    return $numOfbits;
}

# Takes two arguments:
# Decimal number and number of bits to convert to
sub dec2bin{
    my($dec, $bits) = @_;
    my $bin = "";
    for(my $i=0; $i<=$bits-1; $i++){
        $bin .= $dec % 2;
        $dec /=  2;
    }
    return reverse($bin);
}

# Takes two arguments:
# Number of colums to use and whitch row to return
sub onehot{
    my ($col, $row) = @_;
    my $head = "0";
    my $tail = "0";
    $head x= $col - $row- 1;
    $tail x= $row;
    return $head."1".$tail; # return row number $row
}

# Takes no arguments
sub getDate{
    use POSIX qw(strftime);
    my $date = strftime " %b %e %Y", localtime; # b=month e=day Y=year
    $date =~ s/\s+//; # Remove blanks returned by strftime
    return $date;
}

# Usage
sub msg{
    print
        "\n".
        "Usage: ".$filename." -f filename numOfPatterns ".
        "[startpos(default=0)]\n".
        "       ".$filename." -c width words\n".
        "       ".$filename." -u 4 8 12 16 ...\n".
        "\n".
        "       -f       : Read length of patterns from file.\n".
        "       filename : Spesify which file to read.\n".
        "                  Assuming one character in file to be 4 bits ".
        "(one LUT),\n".
        "                  and each line being one word.\n".
        "       startpos : Startposition in file.\n".
        "\n".
        "       -c       : Create CAM that have equal width to all words\n".
        "       width    : Width in bits of words in CAM. Must be 4x value of ".
        "integer.\n".
        "       words    : Number of words in CAM.\n".
        "\n".
        "       -u       : User-defined CAM taken from command-line.\n".
        "                  The width of each word is given at the ".
        "command-line.\n".
        "                  All values must be 4x of integer.\n".
        "\n".
        "       NOTE     : When using the -c or -u switch, there is only a ".
        "dummy data-set\n".
        "                  created in the testbench. Each element in the ".
        "pattern-array indicates\n".
        "                  the length of the corresponding word. These values ".
        "must be edited\n".
        "                  manually. Be sure not to create any subsets.\n".
        "\n";
}

################################################################################
################################################################################
##                                                                            ##
##          Generate variables necessary for creating project files           ##
##                                                                            ##
################################################################################
################################################################################

# Parse ARGV.  Like an amateur... TODO stuff here!!!
if($#ARGV==-1){
    msg();
    exit(-1);
}

elsif($ARGV[0] eq "-f"){ # -f option
    if($#ARGV==2){
        $camContentfile=$ARGV[1];
        $numOfPatterns=$ARGV[2];
    }
    elsif($#ARGV==3){
        $camContentfile=$ARGV[1];
        $numOfPatterns=$ARGV[2];
        $posOfFirstPattern=$ARGV[3];
    }
    else{
        msg();
        exit(-1);
    }

    $posOfLastPattern = $numOfPatterns + $posOfFirstPattern;

    # Load patterns from file
    open(INFILE, "<".$camContentfile) or die
        "\n     Can't open file: ".$camContentfile."\n\n";
    @patterns = <INFILE>;
    close(INFILE);

    # Exit if user asks for position in file that exceeds the number of lines
    # in file
    if(($posOfLastPattern > $#patterns+1) || ($numOfPatterns==0)){
        print
            "\n".
            "     Zero patterns or not enough patterns in file".
            $camContentfile."\n".
            "\n";
        exit(-1);
    }

    # Find length of longest pattern
    # Make array over length of patterns
    for($i=$posOfFirstPattern; $i<$posOfLastPattern; $i++){
        $len = length($patterns[$i])-1; # Minus one byte for \n
        push(@lengthOfPatterns, $len);  # Make array over lengths of patterns
                                        # in LUTs
        $camSize += $len;
        if($longestPattern < $len){      # Longest pattern in hex
            $longestPattern = $len;
        }
    }
    $longestPattern *= 4; # Longest pattern in bin
    $totNumOfPatterns = $#patterns + 1;
    $camSize /= 2;

    print
        "\n".
        "     Loading ".$numOfPatterns." patterns (of ".$totNumOfPatterns.
        ") from ".$camContentfile."\n".
        "     Startposition (starting at zero): ".$posOfFirstPattern."\n".
        "\n".
        "     Number of patterns loaded : ".$numOfPatterns."\n".
        "     Longest pattern (bits)    : ".$longestPattern."\n".
        "     Size of CAM (bytes)       : ".$camSize."\n";
}

elsif($ARGV[0] eq "-c"){ # -c option
    if($#ARGV==2){
        $longestPattern=$ARGV[1];
        if(!(($longestPattern % 4) == 0) || $longestPattern==0){
            print "\n     Width must be of 4x value of integer > 0\n\n";
            exit(-1)
            }
        $numOfPatterns=$ARGV[2];
        for($i=0; $i<$numOfPatterns; $i++){
            push(@patterns, $longestPattern/4); # @patterns holds length in LUTs
        }
        @lengthOfPatterns = @patterns;
        print
            "\n".
            "     Words                     : ".$numOfPatterns."\n".
            "     Width(bits)               : ".$longestPattern."\n";
    }
    else{
        msg();
        exit(-1);
    }
}

elsif($ARGV[0] eq "-u"){ # -u option
    if($#ARGV>=1){
        shift;
        @patterns = @ARGV;

        # Find length of longest pattern
        # Make array over length of patterns
        for($i=0; $i <= $#patterns; $i++){
            if(!(($patterns[$i] % 4)==0) || $patterns[$i]==0){
                print "\n     Width must be of 4x value of integer > 0\n\n";
                exit(-1)
                }
            $len = $patterns[$i];
            push(@lengthOfPatterns, $len/4);  # Make array over lengths of
                                              # patterns in LUTs
            if($longestPattern < $len){       # Longest pattern in bin
                $longestPattern = $len;
            }
        }
        print
            "\n".
            "     Words                     : ".($#lengthOfPatterns + 1)."\n".
            "     Longest pattern (bits)    : ".$longestPattern."\n";

    }
    else {
        msg();
        exit(-1);
    }
}



# Various
$totNumOfPatterns = $#patterns + 1;
$numOfPatterns    = $#lengthOfPatterns + 1;
$numOfAddrBits    = findNumOfAddrBits($numOfPatterns);

# Header
@header =
    (
      "--\n",
      "-- File   : ",
      "",  # Filename goes into this line: $header[2]
      "-- Project: ".$prjName."\n",
      "-- Author : ".$author ."\n",
      "--\n",
      "-- This file was created ".getDate()." by using a Perl-script, \"".
      $filename."\"\n",
      "--\n",
      "-- ".$prjName." project files:\n",
      "-- Top level module:  ".$prjFileCam_top."\n",
      "--           module:  |-> ".$prjFileCamCells."\n",
      "--           module:  |--> ".$prjFileCamGeneric."\n",
      "--           module:  |---> ".$prjFileCam_basic."\n",
      "--           module:  |-> ".$prjFileCompare."\n",
      "--           module:  |-> ".$prjFileCounter."\n",
      "--           module:  |-> ".$prjFileDecode."\n",
      "--           module:  |-> ".$prjFileEncode."\n",
      "--        testbench:  ".$prjFileTb."\n",
      "--          package:  ".$prjFileComponents."\n",
      "--\n",
      "\n"
      );

################################################################################
################################################################################
##                                                                            ##
##                          Create file "cam_top.vhd"                         ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCam_top, ">".$prjFileCam_top) or die
    "Can't open ".$prjFileCam_top."\n";

$header[2] = $prjFileCam_top."\n";

print prjFileCam_top @header;
print prjFileCam_top
    "library ieee, unisim;\n".
    "use ieee.std_logic_1164.all;\n".
    "use unisim.vcomponents.all;\n".
    "use work.cam_components.all;\n".
    "\n".
    "entity cam_top is\n";

$entities{cam_top} =
    [
     "  generic(\n",
     "    longestPattern : integer := ".$longestPattern.";\n",
     "    addrBits       : integer := ".$numOfAddrBits.";\n",
     "    numOfPatterns  : integer := ".$numOfPatterns."\n",
     "    );\n",
     "  port(\n",
     "    clk              : in  std_logic;\n",
     "    rst              : in  std_logic;\n",
     "    -- Data to compare or to write\n",
     "    cam_data         : in  std_logic_vector(longestPattern-1 downto 0);".
     "\n",
     "    -- Address when write ONLY\n",
     "    cam_wordaddr_in  : in  std_logic_vector(addrBits-1 downto 0);\n",
     "       -- Match address\n",
     "    cam_wordaddr_out : out std_logic_vector(addrBits-1 downto 0);\n",
     "    cam_write_rdy    : out std_logic;\n",
     "    -- '1' starts a 16 clock cycle write\n",
     "    cam_write_en     : in  std_logic;\n",
     "    -- Enable to find a match, otherwise no change on match bus.\n",
     "    cam_match_en     : in  std_logic;\n",
     "    -- '1' if match found\n",
     "    cam_match        : out std_logic\n",
     "    );\n"
     ];
print prjFileCam_top @{$entities{cam_top}};

print prjFileCam_top
    "end cam_top;\n".
    "\n".
```

```
"architecture cam_top of cam_top is\n".
"  -- Out of decoder. In to CAM\n".
"  signal word_sel_sig   : std_logic_vector(numOfPatterns-1 downto 0);\n".
"  -- Out of compare. In to CAM\n".
"  signal data_sig       : ".
"std_logic_vector(longestPattern/4-1 downto 0);\n".
"  -- Out of CAM. In to encoder\n".
"  signal match_bus_sig  : std_logic_vector(numOfPatterns-1 downto 0);\n".
"  -- Out of counter. In to decoder\n".
"  signal we_sig         : std_logic;\n".
"  -- Out of counter. In to compare\n".
"  signal cnt_sig        : std_logic_vector(3 downto 0);\n".
"begin\n".
"\n".
"cam_write_rdy <= '1' when cnt_sig=\"0000\" else '0';\n".
"\n".
"counter_inst: counter\n".
"  port map(\n".
"    write_en => cam_write_en,\n".
"    clk      => clk,\n".
"    rst      => rst,\n".
"    we       => we_sig,\n".
"    cnt      => cnt_sig\n".
"  );\n".
"\n".
"compare_inst: compare\n".
"  generic map(\n".
"    longestPattern => longestPattern\n".
"  )\n".
"  port map(\n".
"    addr => cam_data,\n".
"    cnt  => cnt_sig,\n".
"    data => data_sig\n".
"  );\n".
"\n".
"decode_inst: decode\n".
"  port map(\n".
"    we       => we_sig,\n".
"    addr     => cam_wordaddr_in,\n".
"    word_sel => word_sel_sig\n".
"  );\n".
"\n".
"encode_inst: encode\n".
"  port map(\n".
"     addr      => cam_wordaddr_out,\n".
"     match     => cam_match,\n".
"     match_bus => match_bus_sig\n".
"  );\n".
"\n".
"cam_inst: cam_words\n".
" generic map(\n".
"    longestPattern => longestPattern,\n".
"    numOfPatterns  => numOfPatterns\n".
"  )\n".
"  port map(\n".
"    addr      => cam_data,\n".
"    data      => data_sig,\n".
"    clk       => clk,\n".
"    rst       => rst,\n".
"    match_en  => cam_match_en,\n".
"    word_sel  => word_sel_sig,\n".
```

```
    "    match_bus => match_bus_sig\n".
    "  );\n".
    "\n".
    "end cam_top;\n";

close(prjFileCam_top);

################################################################################
################################################################################
##                                                                            ##
##                        Create file "cam_words.vhd"                         ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCamCells, ">".$prjFileCamCells) or die
    "Can't open ".$prjFileCamCells."\n";

$colsInArray = 16;
$header[2] = $prjFileCamCells."\n";

print prjFileCamCells @header;
print prjFileCamCells
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "use work.cam_components.all;\n".
    "\n".
    "entity cam_words is\n";

$entities{cam_words} =
    [
     "  generic(\n",
     "    longestPattern : integer;\n",
     "    numOfPatterns  : integer\n",
     "  );\n",
     "  port(\n",
     "    addr      : in  std_logic_vector(longestPattern-1 downto 0);\n",
     "    -- Out of compare. In to CAM\n",
     "    data      : in  std_logic_vector(longestPattern/4-1 downto 0);\n",
     "    clk       : in  std_logic;\n",
     "    rst       : in  std_logic;\n",
     "    -- Enable to find a match, otherwise no change on match bus\n",
     "    match_en  : in  std_logic;\n",
     "    -- Out of decoder. In to CAM\n",
     "    word_sel  : in  std_logic_vector(numOfPatterns-1 downto 0);\n",
     "    -- Out of CAM. In to encoder\n",
     "    match_bus : out std_logic_vector(numOfPatterns-1 downto 0)\n",
     "  );\n"
     ];
print prjFileCamCells @{$entities{cam_words}};

print prjFileCamCells
    "end cam_words;\n".
    "\n".
    "architecture cam_words of cam_words is\n".
    "  type num_of_luts is array(0 to numOfPatterns-1) of integer;\n".
    "  constant luts : num_of_luts := (  -- 1 LUT = 4 bits\n    ";

# Make array of length of patterns
for($i=0; $i <= $numOfPatterns-1; $i++){
    if((($i % ($colsInArray)) == 0)&&!($i==0)){
        print prjFileCamCells "\n    ";
    }

    # Tabulate columns
    print prjFileCamCells
        " " x (length($longestPattern)-length($lengthOfPatterns[$i])-1).
        $lengthOfPatterns[$i];

    if($i == ($numOfPatterns - 1)){
        print prjFileCamCells "\n    );\n";
    }
    else {
        print prjFileCamCells ", ";
    }
}

print prjFileCamCells
    "begin\n".
    "\n".
    "cam_words: for i in numOfPatterns-1 downto 0 generate\n".
    "begin\n".
    "  cam_word_inst: cam_word\n".
    "    generic map(\n".
    "      numOfLuts => luts(i)\n".
    "      )\n".
    "    port map(\n".
    "      -- lowest index(bits) = highest index(bits) - 4*luts+1\n".
    "      addr => addr(longestPattern-1 downto ".
    "(longestPattern-1)-4*luts(i)+1),\n".
    "      -- lowest index(luts) = highest index(luts) - luts+1\n".
    "      data => data(longestPattern/4-1 downto ".
    "(longestPattern/4-1)-luts(i)+1),\n".
    "      write_en  => word_sel(i),\n".
    "      clk       => clk,\n".
    "      rst       => rst,\n".
    "      match_en  => match_en,\n".
    "      match_out => match_bus(i)\n".
    "      );\n".
    "end generate;\n".
    "\n".
    "end cam_words;\n";

close(prjFileCamCells);

################################################################################
################################################################################
##                                                                            ##
##                       Create file "cam_word.vhd"                           ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCamGeneric, ">".$prjFileCamGeneric) or die
    "Can't open ".$prjFileCamGeneric."\n";

$header[2] = $prjFileCamGeneric."\n";

print prjFileCamGeneric @header;
print prjFileCamGeneric
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "use work.cam_components.all;\n".
    "\n".
    "entity cam_word is\n";

$entities{cam_word} =
    [
     "  generic(\n",
     "    numOfLuts : integer\n",
     "  );\n",
     "  port(\n",
     "    addr      : in  std_logic_vector(numOfLuts*4-1 downto 0);\n",
     "    -- Write one bit to X cam_basic cells in parallell\n",
     "    data      : in  std_logic_vector(numOfLuts-1 downto 0);\n",
     "    -- Write Enable during 16 clock cycles\n",
     "    write_en  : in  std_logic;\n",
     "    clk       : in  std_logic;\n",
     "    rst       : in  std_logic;\n",
     "    -- And gate should be disabled during write\n",
     "    match_en  : in  std_logic;\n",
     "    -- '1' is the DATA_IN matches the stored data\n",
     "    match_out : out std_logic\n",
     "    );\n"
     ];
print prjFileCamGeneric @{$entities{cam_word}};

print prjFileCamGeneric
    "end cam_word;\n".
    "\n".
    "architecture cam_word of cam_word is\n".
    "  -- (wires) carry in/out of MUXCY \n".
    "  signal match_in : std_logic_vector(numOfLuts downto 0);\n".
    "begin\n".
    "\n".
    "match_in(numOfLuts) <= match_en;\n".
    "\n".
    "-- Make one cam-word\n".
    "-- Use X cam_basic components\n".
    "cam_basic_X: for i in numOfLuts-1 downto 0 generate\n".
    "begin\n".
    "  -- Instansiate cam_basic one by one\n".
    "  cam_basic_inst: cam_basic\n".
    "    port map(\n".
    "      data      => data(i),\n".
    "      match_in  => match_in(i+1),\n".
    "      -- start with most significant bit\n".
    "      addr      => addr((4*i+3) downto (4*i)),\n".
    "      write_en  => write_en,\n".
    "      clk       => clk,\n".
    "      -- This wire goes to register\n".
    "      match_out => match_in(i)\n".
    "      );\n".
    "end generate;\n".
    "\n".
    "-- Register the result\n".
    "register_match_out: process(rst, clk)\n".
    "begin\n".
    "  if(rst = '0') then\n".
    "    match_out <= '0';\n".
    "  elsif rising_edge(clk) then\n".
    "    match_out <= match_in(0);\n".
    "  end if;\n".
    "end process;\n".
    "\n".
    "end cam_word;\n";

close(prjFileCamGeneric);

################################################################################
################################################################################
##                                                                            ##
##                        Create file "cam_basic.vhd"                         ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCam_basic, ">".$prjFileCam_basic) or die
    "Can't open ".$prjFileCam_basic."\n";

$header[2] = $prjFileCam_basic."\n";

print prjFileCam_basic @header;
print prjFileCam_basic
    "library ieee, unisim;\n".
    "use ieee.std_logic_1164.all;\n".
    "use unisim.vcomponents.all;\n".
    "\n".
    "entity cam_basic is\n";

$entities{cam_basic} =
    [
     "  port(\n",
     "    data      : in  std_logic;    -- Data to write (one bit at a time)\n",
     "    write_en  : in  std_logic;\n",
     "    clk       : in  std_logic;\n",
     "    addr      : in  std_logic_vector(3 downto 0);\n",
     "    match_in  : in  std_logic;    -- Input  to   MUXCY (carry-in)\n",
     "    match_out : out std_logic     -- Output from MUXCY (carry-out)\n",
     "    );\n"
     ];
print prjFileCam_basic @{$entities{cam_basic}};

print prjFileCam_basic
    "end cam_basic;\n".
    "\n".
    "architecture cam_basic of cam_basic is\n".
    "  signal s   : std_logic; -- Select\n".
    "  signal gnd : std_logic;\n".
    "begin\n".
    "\n".
    "gnd <= '0';\n".
    "\n".
    "-- Use one SRL16E to make a 4-bit cam\n".
    "srl16e_inst: srl16e\n".
    "  port map(\n".
    "    d   => data,\n".
    "    ce  => write_en,\n".
    "    clk => clk,\n".
    "    a0  => addr(0),\n".
    "    a1  => addr(1),\n".
    "    a2  => addr(2),\n".
    "    a3  => addr(3),\n".
    "    q   => s\n".
    "    );\n".
    "\n".
    "-- Make wide and gate by using MUXCY\n".
    "muxcy_inst: muxcy\n".
    "  port map(\n".
    "    di => gnd,      -- DataIn\n".
    "    ci => match_in, -- CarryIn \n".
    "    s  => s,        -- Select\n".
    "    o  => match_out -- Output \n".
    "    );\n".
    "\n".
    "end cam_basic;\n";

close(prjFileCam_basic);

################################################################################
################################################################################
##                                                                            ##
##                         Create file "compare.vhd"                          ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCompare, ">".$prjFileCompare) or die
    "Can't open ".$prjFileCompare."\n";

$header[2] = $prjFileCompare."\n";

print prjFileCompare @header;
print prjFileCompare
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "\n".
    "entity compare is\n";

$entities{compare} =
    [
     "  generic(\n",
     "    longestPattern : integer\n",
     "  );\n",
     "  port(\n",
     "    -- Longest pattern to be written\n",
     "    addr : in  std_logic_vector(longestPattern-1 downto 0);\n",
     "    -- Output from 16 bit counter\n",
     "    cnt  : in  std_logic_vector(3 downto 0);\n",
     "    -- LUTs needed for longest pattern\n",
     "    data : out std_logic_vector(longestPattern/4-1 downto 0)\n",
     "    );\n"
     ];
print prjFileCompare @{$entities{compare}};

print prjFileCompare
    "end compare;\n".
    "\n".
    "architecture compare of compare is\n".
    "  -- Output of xnor2\n".
    "  signal bit_xnor : std_logic_vector(longestPattern-1 downto 0);\n".
    "begin\n".
    "\n".
    "-- Compare bits to be written with the counter (xnor gates)\n".
    "-- generate xnor2-gates for comparators\n".
    "comparators: for j in 0 to longestPattern/4-1 generate\n".
    "begin\n".
    "  -- generate xnor2-gates 4 by 4\n".
    "  xnor2_inst:    for i in 0 to 3 generate\n".
    "  begin\n".
    "    bit_xnor(i+j*4) <= not (addr(i+j*4) xor cnt(i));   -- xnor2 gates\n".
    "  end generate;\n".
    "end generate;\n".
    "\n".
    "-- connect xnor2 to and4\n".
    "and_inst: for i in 0 to longestPattern/4-1 generate\n".
    "begin\n".
    "  data(i) <= bit_xnor(i*4  ) and  -- and4-gates\n".
    "             bit_xnor(i*4+1) and\n".
    "             bit_xnor(i*4+2) and\n".
    "             bit_xnor(i*4+3);\n".
    "end generate;\n".
    "\n".
    "end compare;\n";

close(prjFileCompare);

################################################################################
################################################################################
##                                                                            ##
##                         Create file "counter.vhd"                          ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileCounter, ">".$prjFileCounter) or die
    "Can't open ".$prjFileCounter."\n";

$header[2] = $prjFileCounter."\n";

print prjFileCounter @header;
print prjFileCounter
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "use ieee.std_logic_unsigned.all;\n".
    "use ieee.std_logic_arith.all;\n".
    "\n".
    "entity counter is\n";

$entities{counter} =
    [
     "  port(\n",
     "     -- one high write_en starts 16 write cycles when going low\n",
     "     write_en : in  std_logic;\n",
     "     clk      : in  std_logic;\n",
     "     rst      : in  std_logic; \n",
     "     -- Write enable valid during 16 clock cycles\n",
     "     we       : out std_logic;\n",
     "     -- Copy of counter value\n",
     "     cnt      : out std_logic_vector(3 downto 0)\n",
     "     );\n"
     ];
print prjFileCounter @{$entities{counter}};

print prjFileCounter
    "end counter;\n".
    "\n".
    "architecture counter of counter is\n".
    "  -- Count the 16 write clock cycles\n".
    "  signal count    : std_logic_vector(3 downto 0);\n".
    "  signal term_cnt : std_logic;\n".
    "begin\n".
    "\n".
    "-- Generate a Write Enable for the decoder and counter data for ".
    "comparison\n".
    "write_cycle: process(rst, clk, count)\n".
    "begin\n".
    "  if(rst = '0') then       -- Asynchronous reset\n".
    "    count <= (others => '0');\n".
    "  elsif rising_edge(clk) then \n".
    "    if(write_en = '1') then   -- Start counting down\n".
    "      count <= (others => '1');\n".
    "    else\n".
    "      if(term_cnt = '1') then\n".
    "        count <= count - 1;  -- Count 15 downto 0\n".
    "      end if;\n".
    "    end if;\n".
    "  end if;\n".
    "  cnt <= count;\n".
    "end process;\n".
    "\n".
    "-- Terminal count generation: Prevent counter to wrap around\n".
    "term_cnt <= count(3) or count(2) or count(1) or count(0);\n".
    "\n".
    "-- Generate a 16 clock cycles enable signal\n".
    "write_en_register: process(rst, clk)\n".
    "begin\n".
    "  if(rst = '0') then \n".
    "    we <= '0';\n".
    "  elsif rising_edge(clk) then \n".
    "    we <= write_en or term_cnt;\n".
    "  end if;\n".
    "end process;\n".
    "\n".
    "end counter;\n";

close(prjFileCounter);

################################################################################
################################################################################
##                                                                            ##
##                          Create file "decode.vhd"                          ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileDecode, ">".$prjFileDecode) or die
    "Can't open ".$prjFileDecode."\n";

$header[2] = $prjFileDecode."\n";

print prjFileDecode @header;
print prjFileDecode
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "\n".
    "entity decode is\n";
```

```

$entities{decode} =
    [
     "  port(\n",
     "    -- WriteEnable\n",
     "    we       : in  std_logic;\n",
     "    -- Binary address to words\n",
     "    addr     : in  std_logic_vector(".($numOfAddrBits-1)." downto 0);\n",
     "    -- Select one word\n",
     "    word_sel : out std_logic_vector(".($numOfPatterns-1)." downto 0)\n",
     "    );\n"
     ];
print prjFileDecode @{$entities{decode}};

print prjFileDecode
    "end decode;\n".
    "\n".
    "architecture decode of decode is\n".
    "begin\n".
    "\n".
    "-- Create write enable signal\n".
    "decode: process(addr, we)\n".
    "begin\n".
    "  word_sel <= (others => '0');\n".
    "  case addr is\n";

for($i=0; $i<=$numOfPatterns-1; $i++){
    print prjFileDecode
        "    when \"".dec2bin($i, $numOfAddrBits)."\" => word_sel(".
        " " x (length($numOfPatterns-1) - length($i)). # tabulate numbers
        $i.") <= we;\n";
}

print prjFileDecode
    "    when others => word_sel <= (others => '0');\n".
    "  end case;\n".
    "end process;\n".
    "\n".
    "end decode;";

close(prjFileDecode);

################################################################################
################################################################################
##                                                                            ##
##                          Create file "encode.vhd"                          ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileEncode, ">".$prjFileEncode) or die
    "Can't open ".$prjFileEncode."\n";

$header[2] = $prjFileEncode."\n";

print prjFileEncode @header;
print prjFileEncode
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "\n".
    "entity encode is\n";
```

```
$entities{encode} =
    [
     "  port(\n",
     "    -- '1' if match is found\n",
     "    match     : out std_logic;\n",
     "    -- Match address\n",
     "    addr      : out std_logic_vector(".($numOfAddrBits-1)." downto 0);\n",
     "    -- match_bus from CAM-words\n",
     "    match_bus : in  std_logic_vector(".($numOfPatterns-1)." downto 0)\n",
     "    );\n"
     ];
print prjFileEncode @{$entities{encode}};

print prjFileEncode
    "end encode;\n".
    "\n".
    "architecture encode of encode is\n".
    "begin\n".
    "\n".
    "generate_address: process(match_bus)\n".
    "begin\n".
    "  case match_bus is\n";

for($i=0; $i<=$numOfPatterns-1; $i++){
    print prjFileEncode
        "    when \"".onehot($numOfPatterns, $i).
        "\" => addr <= \"".dec2bin($i, $numOfAddrBits)."\";\n";
}

print prjFileEncode
    "    when others => addr <= ( others => '0');\n".
    "  end case;\n".
    "end process;\n".
    "\n".
    "-- Generate the match signal if one or more match(es) is/are found\n".
    "match <= '0' when match_bus =\n".
    "  \""."0" x $numOfPatterns."\"\n".
    "  else '1';\n".
    "\n".
    "end encode;\n";

close(prjFileEncode);

################################################################################
################################################################################
##                                                                            ##
##                        Create file "components.vhd"                       ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileComponents, ">".$prjFileComponents) or die
    "Can't open ".$prjFileComponents."\n";

$header[2] = $prjFileComponents."\n";

print prjFileComponents @header;
print prjFileComponents
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "\n".
    "package cam_components is\n".
    "\n";

for $key(keys %entities){
    print prjFileComponents "component ".$key." is\n";
    print prjFileComponents @{$entities{$key}};
    print prjFileComponents "end component;\n\n";
}

print prjFileComponents "end cam_components;\n";

close(prjFileComponents);

################################################################################
################################################################################
##                                                                            ##
##                         Create file "tb_cam_top"                           ##
##                                                                            ##
################################################################################
################################################################################

open(prjFileTb, ">".$prjFileTb) or die
    "Can't open ".$prjFileTb."\n";

# Estimate of time needed for simulation
$tb_clkCycle     = 10; # 10ns => 100MHz
$tb_rstAndInit   = 3*$tb_clkCycle;
$tb_write        = $numOfPatterns * $tb_clkCycle * 16; # 16 cycles in each write
$tb_matchBefore  = 2*$tb_clkCycle;
$tb_match        = $numOfPatterns*$tb_clkCycle;
$tb_matchAfter   = 2*$tb_clkCycle;
$tb_total = $tb_rstAndInit+$tb_write+$tb_matchBefore+$tb_match+$tb_matchAfter;
# Pattern not in CAM
$tb_errorPattern = "B" x ($longestPattern/4);
# Used to write array in testbench
$tb_pattern       = "";
$tb_patternZeroes = "";
# Write correct filename to header
$header[2] = $prjFileTb."\n";

print prjFileTb @header;
print prjFileTb
    "--\n".
    "-- Behavioral simulation at 10 ns clock cycles.\n".
    "-- 2 clock cycles delay after match enable goes high\n".
    "-- \n";

printf(prjFileTb "-- reset:%26s ns\n", $tb_rstAndInit);
printf(prjFileTb "-- write:     16*160ns = %10s ns\n", $tb_write);
printf(prjFileTb "-- matching:                  %5s ns  Before match\n",
       $tb_matchBefore);
printf(prjFileTb "--          %5s*10ns = %10s ns  Match status\n",
       $numOfPatterns, $tb_match);
printf(prjFileTb "--                            %5s ns  Get last 2 matches\n",
       $tb_matchAfter);
print prjFileTb "--                       _____________\n";
printf(prjFileTb "-- Sum:%28s ns\n", $tb_total);

print prjFileTb
    "--\n".
    "\n".
    "library ieee;\n".
    "use ieee.std_logic_1164.all;\n".
    "use ieee.std_logic_arith.all;\n".
    "use work.cam_components.all;\n".
    "\n".
    "entity testbench is\n".
    "  generic(\n".
    "    longestPattern : integer := ".$longestPattern.";\n".
    "    addrBits       : integer := ".$numOfAddrBits.";\n".
    "    numOfPatterns  : integer := ".$numOfPatterns."\n".
    "    );\n".
    "end testbench;\n".
    "\n".
    "architecture testbench of testbench is\n".
    "  signal   cam_data                                    :\n".
    "           std_logic_vector(longestPattern-1 downto 0) := ".
    "(others => '0');\n".
    "  signal   cam_data_reg                                :\n".
    "           std_logic_vector(longestPattern-1 downto 0) := ".
    "(others => '0');\n".
    "  signal   cam_wordaddr_in                             :\n".
    "           std_logic_vector(addrBits-1 downto 0)       := ".
    "(others => '0');\n".
    "  signal   cam_wordaddr_in_reg                         :\n".
    "           std_logic_vector(addrBits-1 downto 0)       := ".
    "(others => '0');\n".
    "  signal   cam_write_rdy                               :\n".
    "           std_logic                                   := '0';\n".
    "  signal   cam_write_en                                :\n".
    "           std_logic                                   := '0';\n".
    "  signal   clk                                         :\n".
    "           std_logic                                   := '0';\n".
    "  signal   rst                                         :\n".
    "           std_logic                                   := '0';\n".
    "  signal   cam_match_en                                :\n".
    "           std_logic                                   := '0';\n".
    "  signal   cam_match_en_reg                            :\n".
    "           std_logic                                   := '0';\n".
    "  signal   cam_wordaddr_out                            :\n".
    "           std_logic_vector(addrBits-1 downto 0)       := ".
    "(others => '0');\n".
    "  signal   cam_match                                   :\n".
    "           std_logic                                   := '0';\n".
    "  constant half_period                                 :\n".
    "           time                                        := 5 ns;\n".
    "\n".
    "  -- Bitvectors are set to equal size to make the testbench easier to ".
    "read.\n".
    "  -- In hardware the \"_00...\"-part may be omitted\n".
    "  type pattern_array is\n".
    "    array(0 to numOfPatterns-1) of ".
    "bit_vector (longestPattern-1 downto 0);\n".
    "  constant pattern : pattern_array := (  -- Content of CAM\n";

# Write test-patterns
# End patterns that are shorter than the longest pattern with "_00..."

if($posOfLastPattern==0){ # -c or -u switch has been used
    $posOfLastPattern = $#patterns+1;
}

for($i = $posOfFirstPattern; $i < $posOfLastPattern; $i++){
    $tb_pattern = $patterns[$i];
    $tb_pattern =~ s/\n//;
    $tb_patternZeroes = "";
    if(length($tb_pattern) < $longestPattern/4){
        $tb_patternZeroes = "_"."0" x ($longestPattern/4 - length($tb_pattern));
    }
    print prjFileTb "    x\"".$tb_pattern.$tb_patternZeroes."\"";
    if($i < $posOfLastPattern-1){
        print prjFileTb ",";
    }
    print prjFileTb "\n";
}

print prjFileTb
    "    );\n".
    "\n".
    "  -- A (userdefined) pattern not to be found in CAM\n".
    "  constant error_pattern : bit_vector(longestPattern-1 downto 0) :=\n".
    "    x\"".$tb_errorPattern."\";\n".
    "begin\n".
    "\n".
    "uut: cam_top port map(\n".
    "  cam_data         => cam_data_reg,\n".
    "  cam_wordaddr_in  => cam_wordaddr_in_reg,\n".
    "  cam_write_en     => cam_write_en,\n".
    "  cam_write_rdy    => cam_write_rdy,\n".
    "  clk              => clk,\n".
    "  rst              => rst,\n".
    "  cam_match_en     => cam_match_en_reg,\n".
    "  cam_wordaddr_out => cam_wordaddr_out,\n".
    "  cam_match        => cam_match\n".
    "  );\n".
    "\n".
    "clk <= not(clk) after half_period;  \n".
    "\n".
    "tb: process\n".
    "begin\n".
    "\n".
    "  rst <= '0'; wait for 2*half_period;\n".
    "  rst <= '1'; wait for 2*half_period;\n".
    "\n".
    "  -- Syncronize signals to rising edge\n".
    "  if not rising_edge(clk) then\n".
    "    wait for half_period;\n".
    "  end if;\n".
    "\n".
    "  -- write new data to CAM. Write to all CAM-locations\n".
    "  for i in 0 to numOfPatterns-1 loop\n".
    "    -- example: '1' <= \"01\"\n".
    "    cam_wordaddr_in <= conv_std_logic_vector(i,addrBits);\n".
    "    cam_data        <= to_stdLogicVector(pattern(i));\n".
    "    if(cam_write_rdy='0') then\n".
    "      wait until cam_write_rdy = '1';\n".
    "    end if;\n".
    "    -- Start counter\n".
    "    cam_write_en <= '1'; wait for    2*half_period;\n".
    "    -- 15 clock-cycles left of writecycle\n".
    "    cam_write_en <= '0'; wait for 15*2*half_period;\n".
    "  end loop;\n".
    "\n".
    "  -- \"read\" CAM: verify that data has been written to CAM\n".
    "  cam_match_en <= '1';\n".
    "\n".
    "  for i in 0 to numOfPatterns-1 loop\n".
    "    if(i=numOfPatterns-2) then\n".
    "      cam_data <= to_stdlogicvector(error_pattern); -- Make mismatch\n".
    "    else\n".
    "      cam_data <= to_stdlogicvector(pattern(i));\n".
    "    end if;\n".
    "    wait for 2*half_period;\n".
    "  end loop;\n".
    "\n".
    "  cam_match_en <= '0';\n".
    "\n".
    "  wait; -- Prevent simulation to wrap around\n".
    "end process;\n".
    "\n".
    "-- Registered I/O\n".
    "io_register: process(rst, clk)\n".
    "begin\n".
    "  if(rst = '0') then\n".
    "    cam_data_reg         <= (others => '0');\n".
    "    cam_wordaddr_in_reg  <= (others => '0');\n".
    "    cam_match_en_reg     <= '0';\n".
    "  else\n".
    "    if rising_edge(clk) then\n".
    "      cam_data_reg         <= cam_data;\n".
    "      cam_wordaddr_in_reg  <= cam_wordaddr_in;\n".
    "      cam_match_en_reg     <= cam_match_en;\n".
    "    end if;\n".
    "  end if;\n".
    "end process;\n".
    "\n".
    "end testbench;\n";

close(prjFileTb);

################################################################################
################################################################################
##                                                                            ##
##                     Warning: User defined fdo-file                         ##
##                                                                            ##
################################################################################
################################################################################

# Create .fdo-file for easy startup in ModelSim
open(fdoFile, ">".$prjFileFdo) or die
    "Can't open ".$prjFileFdo."\n";

$tb_zoom_from = $tb_total-($numOfPatterns*10)-(24*10);

print fdoFile
    "# User defined fdo-file.\n".
    "# Created " .getDate()."\n".
    "vlib       ".$sim_path."\n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./counter.vhd   \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./compare.vhd   \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./decode.vhd    \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./encode.vhd    \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./cam_basic.vhd \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./components.vhd\n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./cam_word.vhd  \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./cam_words.vhd \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./cam_top.vhd   \n".
    "vcom -work ".$sim_path." -nologo -93 -explicit ./tb_cam_top.vhd\n".
    "vsim -t 1ps -lib ".$sim_path." testbench      \n".
    "view wave\n".
    "onerror {resume}\n".
    "quietly WaveActivateNextPane {} 0\n".
    "add wave -noupdate -format Logic   /testbench/rst             \n".
    "add wave -noupdate -format Logic   /testbench/clk             \n".
    "add wave -noupdate -format Literal -radix ascii ".
                                      " /testbench/cam_data        \n".
    "add wave -noupdate -format Literal /testbench/cam_wordaddr_in \n".
    "add wave -noupdate -format Logic   /testbench/cam_write_rdy   \n".
    "add wave -noupdate -format Logic   /testbench/cam_write_en    \n".
    "add wave -noupdate -format Logic   /testbench/cam_match_en    \n".
    "add wave -noupdate -format Literal /testbench/cam_wordaddr_out\n".
    "add wave -noupdate -format Logic   /testbench/cam_match       \n".
    "TreeUpdate [SetDefaultTree]\n".
    "WaveRestoreCursors {0 ps}\n".
    "WaveRestoreZoom {".$tb_zoom_from." ns} {".$tb_total." ns}\n".
    "configure wave -namecolwidth 160\n".
    "configure wave -valuecolwidth 130\n".
    "configure wave -justifyvalue left\n".
    "configure wave -signalnamewidth 1\n".
    "configure wave -snapdistance 10\n".
    "configure wave -datasetprefix 0\n".
    "configure wave -rowmargin 4\n".
    "configure wave -childrowmargin 2\n".
    "\n".
    "run ".$tb_total."ns";

close(fdoFile);

################################################################################
################################################################################
##                                                                            ##
##                                 Print                                      ##
##                                                                            ##
################################################################################
################################################################################

print
    "     Address bits              : ".$numOfAddrBits."\n".
    "\n".
    "     Wrote Top level module:  ".$prjFileCam_top."\n".
    "     Wrote           module:  |-> ".$prjFileCamCells."   (1)\n".
    "     Wrote           module:  |--> ".$prjFileCamGeneric."   (2)\n".
    "     Wrote           module:  |---> ".$prjFileCam_basic." (2)\n".
    "     Wrote           module:  |-> ".$prjFileCompare."     (2)\n".
    "     Wrote           module:  |-> ".$prjFileCounter."     (2)\n".
    "     Wrote           module:  |-> ".$prjFileDecode."\n".
    "     Wrote           module:  |-> ".$prjFileEncode."\n".
    "     Wrote          package:  ".$prjFileComponents."\n".
    "     Wrote        testbench:  ".$prjFileTb."\n".
    "     Wrote         fdo-file:  ".$prjFileFdo."\n".
    "\n".
    "     1) Contains array over length of words\n".
    "     2) Files need to be created only once\n".
    "\n";
```

```

################################################################################
################################################################################
##                                                                            ##
##                                                                            ##
##             #####  #                     #####               #             ##
##               #    #                     #                   #             ##
##               #    #                     #                   #             ##
##               #    # ###    ####         #      # ###    ### #             ##
##               #    ##   #  #    #        ####   ##   #  #   ##             ##
##               #    #    #  ######        #      #    #  #    #             ##
##               #    #    #  #             #      #    #  #    #             ##
##               #    #    #  #    #        #      #    #  #   ##             ##
##               #    #    #   ####         #####  #    #   ### #             ##
##                                                                            ##
##                                                                            ##
################################################################################
################################################################################
```

cam_top.vhd

```
--
-- File   : cam_top.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee, unisim;
use ieee.std_logic_1164.all;
use unisim.vcomponents.all;
use work.cam_components.all;

entity cam_top is
  generic(
    longestPattern : integer := 256;
    addrBits       : integer := 5;
    numOfPatterns  : integer := 32
    );
  port(
    clk              : in  std_logic;
    rst              : in  std_logic;
    -- Data to compare or to write
    cam_data         : in  std_logic_vector(longestPattern-1 downto 0);
    -- Address when write ONLY
    cam_wordaddr_in  : in  std_logic_vector(addrBits-1 downto 0);
       -- Match address
    cam_wordaddr_out : out std_logic_vector(addrBits-1 downto 0);
    cam_write_rdy    : out std_logic;
    -- '1' starts a 16 clock cycle write
    cam_write_en     : in  std_logic;
    -- Enable to find a match, otherwise no change on match bus.
    cam_match_en     : in  std_logic;
    -- '1' if match found
    cam_match        : out std_logic
    );
end cam_top;

architecture cam_top of cam_top is
  -- Out of decoder. In to CAM
  signal word_sel_sig   : std_logic_vector(numOfPatterns-1 downto 0);
  -- Out of compare. In to CAM
  signal data_sig       : std_logic_vector(longestPattern/4-1 downto 0);
  -- Out of CAM. In to encoder
  signal match_bus_sig  : std_logic_vector(numOfPatterns-1 downto 0);
  -- Out of counter. In to decoder
  signal we_sig         : std_logic;
  -- Out of counter. In to compare
  signal cnt_sig        : std_logic_vector(3 downto 0);
begin

cam_write_rdy <= '1' when cnt_sig="0000" else '0';

counter_inst: counter
  port map(
    write_en => cam_write_en,
    clk      => clk,
    rst      => rst,
    we       => we_sig,
    cnt      => cnt_sig
  );

compare_inst: compare
  generic map(
    longestPattern => longestPattern
  )
  port map(
    addr => cam_data,
    cnt  => cnt_sig,
    data => data_sig
  );

decode_inst: decode
  port map(
    we       => we_sig,
    addr     => cam_wordaddr_in,
    word_sel => word_sel_sig
  );

encode_inst: encode
  port map(
    addr      => cam_wordaddr_out,
    match     => cam_match,
    match_bus => match_bus_sig
  );

cam_inst: cam_words
 generic map(
    longestPattern => longestPattern,
    numOfPatterns  => numOfPatterns
  )
  port map(
    addr      => cam_data,
    data      => data_sig,
    clk       => clk,
    rst       => rst,
    match_en  => cam_match_en,
    word_sel  => word_sel_sig,
    match_bus => match_bus_sig
  );

end cam_top;
```

cam_words.vhd

```
--
-- File   : cam_words.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;
use work.cam_components.all;

entity cam_words is
  generic(
    longestPattern : integer;
    numOfPatterns  : integer
  );
  port(
    addr      : in  std_logic_vector(longestPattern-1 downto 0);
    -- Out of compare. In to CAM
    data      : in  std_logic_vector(longestPattern/4-1 downto 0);
    clk       : in  std_logic;
    rst       : in  std_logic;
    -- Enable to find a match, otherwise no change on match bus
    match_en  : in  std_logic;
    -- Out of decoder. In to CAM
    word_sel  : in  std_logic_vector(numOfPatterns-1 downto 0);
    -- Out of CAM. In to encoder
    match_bus : out std_logic_vector(numOfPatterns-1 downto 0)
  );
end cam_words;

architecture cam_words of cam_words is
```

```
  type num_of_luts is array(0 to numOfPatterns-1) of integer;
  constant luts : num_of_luts := (  -- 1 LUT = 4 bits
    44, 14, 28, 12, 60, 30, 38, 42, 10, 24, 64, 14,  8, 16, 36, 24,
    20, 12, 24, 28, 26, 36, 24, 24, 26, 34, 40, 24, 24, 22, 16, 14
    );
begin

cam_words: for i in numOfPatterns-1 downto 0 generate
begin
  cam_word_inst: cam_word
    generic map(
      numOfLuts => luts(i)
      )
    port map(
      -- lowest index(bits) = highest index(bits) - 4*luts+1
      addr => addr(longestPattern-1 downto (longestPattern-1)-4*luts(i)+1),
      -- lowest index(luts) = highest index(luts) - luts+1
      data => data(longestPattern/4-1 downto (longestPattern/4-1)-luts(i)+1),
      write_en  => word_sel(i),
      clk       => clk,
      rst       => rst,
      match_en  => match_en,
      match_out => match_bus(i)
      );
end generate;

end cam_words;
```

cam_word.vhd

```
--
-- File   : cam_word.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;
use work.cam_components.all;

entity cam_word is
  generic(
    numOfLuts : integer
  );
  port(
    addr       : in  std_logic_vector(numOfLuts*4-1 downto 0);
    -- Write one bit to X cam_basic cells in parallell
    data       : in  std_logic_vector(numOfLuts-1 downto 0);
    -- Write Enable during 16 clock cycles
    write_en   : in  std_logic;
    clk        : in  std_logic;
    rst        : in  std_logic;
    -- And gate should be disabled during write
    match_en   : in  std_logic;
    -- '1' is the DATA_IN matches the stored data
    match_out  : out std_logic
    );
end cam_word;

architecture cam_word of cam_word is
  -- (wires) carry in/out of MUXCY
  signal match_in : std_logic_vector(numOfLuts downto 0);
begin

match_in(numOfLuts) <= match_en;

-- Make one cam-word
-- Use X cam_basic components
cam_basic_X: for i in numOfLuts-1 downto 0 generate
begin
  -- Instansiate cam_basic one by one
  cam_basic_inst: cam_basic
    port map(
      data      => data(i),
      match_in  => match_in(i+1),
      -- start with most significant bit
      addr      => addr((4*i+3) downto (4*i)),
      write_en  => write_en,
      clk       => clk,
      -- This wire goes to register
      match_out => match_in(i)
      );
end generate;

-- Register the result
register_match_out: process(rst, clk)
begin
  if(rst = '0') then
    match_out <= '0';
  elsif rising_edge(clk) then
    match_out <= match_in(0);
  end if;
end process;

end cam_word;
```

cam_basic.vhd

```
--
-- File   : cam_basic.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee, unisim;
use ieee.std_logic_1164.all;
use unisim.vcomponents.all;

entity cam_basic is
  port(
    data      : in  std_logic;    -- Data to write (one bit at a time)
    write_en  : in  std_logic;
    clk       : in  std_logic;
    addr      : in  std_logic_vector(3 downto 0);
    match_in  : in  std_logic;    -- Input  to   MUXCY (carry-in)
    match_out : out std_logic     -- Output from MUXCY (carry-out)
    );
end cam_basic;

architecture cam_basic of cam_basic is
  signal s   : std_logic; -- Select
  signal gnd : std_logic;
begin

gnd <= '0';

-- Use one SRL16E to make a 4-bit cam
srl16e_inst: srl16e
  port map(
    d   => data,
    ce  => write_en,
    clk => clk,
    a0  => addr(0),
    a1  => addr(1),
    a2  => addr(2),
    a3  => addr(3),
    q   => s
    );

-- Make wide and gate by using MUXCY
muxcy_inst: muxcy
  port map(
    di => gnd,       -- DataIn
    ci => match_in, -- CarryIn
    s  => s,         -- Select
    o  => match_out -- Output
    );

end cam_basic;
```

compare.vhd

```
--
-- File   : compare.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;

entity compare is
  generic(
    longestPattern : integer
  );
  port(
    -- Longest pattern to be written
    addr : in  std_logic_vector(longestPattern-1 downto 0);
    -- Output from 16 bit counter
    cnt  : in  std_logic_vector(3 downto 0);
    -- LUTs needed for longest pattern
    data : out std_logic_vector(longestPattern/4-1 downto 0)
    );
end compare;

architecture compare of compare is
  -- Output of xnor2
  signal bit_xnor : std_logic_vector(longestPattern-1 downto 0);
begin

-- Compare bits to be written with the counter (xnor gates)
-- generate xnor2-gates for comparators
comparators: for j in 0 to longestPattern/4-1 generate
begin
  -- generate xnor2-gates 4 by 4
  xnor2_inst:    for i in 0 to 3 generate
  begin
    bit_xnor(i+j*4) <= not (addr(i+j*4) xor cnt(i));   -- xnor2 gates
  end generate;
end generate;

-- connect xnor2 to and4
and_inst: for i in 0 to longestPattern/4-1 generate
begin
  data(i) <= bit_xnor(i*4  ) and  -- and4-gates
             bit_xnor(i*4+1) and
             bit_xnor(i*4+2) and
             bit_xnor(i*4+3);
end generate;

end compare;
```

counter.vhd

```
--
-- File   : counter.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity counter is
  port(
    -- one high write_en starts 16 write cycles when going low
    write_en : in  std_logic;
    clk      : in  std_logic;
    rst      : in  std_logic;
    -- Write enable valid during 16 clock cycles
    we       : out std_logic;
    -- Copy of counter value
    cnt      : out std_logic_vector(3 downto 0)
    );
end counter;

architecture counter of counter is
  -- Count the 16 write clock cycles
  signal count    : std_logic_vector(3 downto 0);
  signal term_cnt : std_logic;
begin

-- Generate a Write Enable for the decoder and counter data for comparison
write_cycle: process(rst, clk, count)
begin
  if(rst = '0') then       -- Asynchronous reset
    count <= (others => '0');
  elsif rising_edge(clk) then
    if(write_en = '1') then   -- Start counting down
      count <= (others => '1');
    else
      if(term_cnt = '1') then
        count <= count - 1;  -- Count 15 downto 0
      end if;
    end if;
  end if;
  cnt <= count;
```

```
end process;

-- Terminal count generation: Prevent counter to wrap around
term_cnt <= count(3) or count(2) or count(1) or count(0);

-- Generate a 16 clock cycles enable signal
write_en_register: process(rst, clk)
begin
  if(rst = '0') then
    we <= '0';
  elsif rising_edge(clk) then
    we <= write_en or term_cnt;
  end if;
end process;

end counter;
```

decode.vhd

```
--
-- File    : decode.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;

entity decode is
  port(
    -- WriteEnable
    we       : in  std_logic;
    -- Binary address to words
    addr     : in  std_logic_vector(4 downto 0);
    -- Select one word
    word_sel : out std_logic_vector(31 downto 0)
    );
end decode;

architecture decode of decode is
begin

-- Create write enable signal
decode: process(addr, we)
begin
  word_sel <= (others => '0');
  case addr is
```

```
    when "00000" => word_sel( 0) <= we;
    when "00001" => word_sel( 1) <= we;
    when "00010" => word_sel( 2) <= we;
    when "00011" => word_sel( 3) <= we;
    when "00100" => word_sel( 4) <= we;
    when "00101" => word_sel( 5) <= we;
    when "00110" => word_sel( 6) <= we;
    when "00111" => word_sel( 7) <= we;
    when "01000" => word_sel( 8) <= we;
    when "01001" => word_sel( 9) <= we;
    when "01010" => word_sel(10) <= we;
    when "01011" => word_sel(11) <= we;
    when "01100" => word_sel(12) <= we;
    when "01101" => word_sel(13) <= we;
    when "01110" => word_sel(14) <= we;
    when "01111" => word_sel(15) <= we;
    when "10000" => word_sel(16) <= we;
    when "10001" => word_sel(17) <= we;
    when "10010" => word_sel(18) <= we;
    when "10011" => word_sel(19) <= we;
    when "10100" => word_sel(20) <= we;
    when "10101" => word_sel(21) <= we;
    when "10110" => word_sel(22) <= we;
    when "10111" => word_sel(23) <= we;
    when "11000" => word_sel(24) <= we;
    when "11001" => word_sel(25) <= we;
    when "11010" => word_sel(26) <= we;
    when "11011" => word_sel(27) <= we;
    when "11100" => word_sel(28) <= we;
    when "11101" => word_sel(29) <= we;
    when "11110" => word_sel(30) <= we;
    when "11111" => word_sel(31) <= we;
    when others => word_sel <= (others => '0');
  end case;
end process;

end decode;
```

encode.vhd

```
--
-- File   : encode.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
```

```
use ieee.std_logic_1164.all;

entity encode is
  port(
    -- '1' if match is found
    match     : out std_logic;
    -- Match address
    addr      : out std_logic_vector(4 downto 0);
    -- match_bus from CAM-words
    match_bus : in  std_logic_vector(31 downto 0)
    );
end encode;

architecture encode of encode is
begin

generate_address: process(match_bus)
begin
  case match_bus is
    when "00000000000000000000000000000001" => addr <= "00000";
    when "00000000000000000000000000000010" => addr <= "00001";
    when "00000000000000000000000000000100" => addr <= "00010";
    when "00000000000000000000000000001000" => addr <= "00011";
    when "00000000000000000000000000010000" => addr <= "00100";
    when "00000000000000000000000000100000" => addr <= "00101";
    when "00000000000000000000000001000000" => addr <= "00110";
    when "00000000000000000000000010000000" => addr <= "00111";
    when "00000000000000000000000100000000" => addr <= "01000";
    when "00000000000000000000001000000000" => addr <= "01001";
    when "00000000000000000000010000000000" => addr <= "01010";
    when "00000000000000000000100000000000" => addr <= "01011";
    when "00000000000000000001000000000000" => addr <= "01100";
    when "00000000000000000010000000000000" => addr <= "01101";
    when "00000000000000000100000000000000" => addr <= "01110";
    when "00000000000000001000000000000000" => addr <= "01111";
    when "00000000000000010000000000000000" => addr <= "10000";
    when "00000000000000100000000000000000" => addr <= "10001";
    when "00000000000001000000000000000000" => addr <= "10010";
    when "00000000000010000000000000000000" => addr <= "10011";
    when "00000000000100000000000000000000" => addr <= "10100";
    when "00000000001000000000000000000000" => addr <= "10101";
    when "00000000010000000000000000000000" => addr <= "10110";
    when "00000000100000000000000000000000" => addr <= "10111";
    when "00000001000000000000000000000000" => addr <= "11000";
    when "00000010000000000000000000000000" => addr <= "11001";
    when "00000100000000000000000000000000" => addr <= "11010";
    when "00001000000000000000000000000000" => addr <= "11011";
    when "00010000000000000000000000000000" => addr <= "11100";
    when "00100000000000000000000000000000" => addr <= "11101";
    when "01000000000000000000000000000000" => addr <= "11110";
    when "10000000000000000000000000000000" => addr <= "11111";
    when others => addr <= ( others => '0');
  end case;
end process;

-- Generate the match signal if one or more match(es) is/are found
match <= '0' when match_bus =
  "00000000000000000000000000000000"
  else '1';

end encode;
```

components.vhd

```
--
-- File   : components.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

library ieee;
use ieee.std_logic_1164.all;

package cam_components is

component compare is
  generic(
    longestPattern : integer
  );
  port(
    -- Longest pattern to be written
    addr : in  std_logic_vector(longestPattern-1 downto 0);
    -- Output from 16 bit counter
    cnt  : in  std_logic_vector(3 downto 0);
    -- LUTs needed for longest pattern
    data : out std_logic_vector(longestPattern/4-1 downto 0)
    );
end component;

component decode is
  port(
    -- WriteEnable
    we       : in  std_logic;
    -- Binary address to words
    addr     : in  std_logic_vector(4 downto 0);
    -- Select one word
    word_sel : out std_logic_vector(31 downto 0)
    );
end component;

component cam_words is
  generic(
    longestPattern : integer;
    numOfPatterns  : integer
  );
  port(
    addr      : in  std_logic_vector(longestPattern-1 downto 0);
    -- Out of compare. In to CAM
    data      : in  std_logic_vector(longestPattern/4-1 downto 0);
    clk       : in  std_logic;
    rst       : in  std_logic;
    -- Enable to find a match, otherwise no change on match bus
    match_en  : in  std_logic;
    -- Out of decoder. In to CAM
    word_sel  : in  std_logic_vector(numOfPatterns-1 downto 0);
    -- Out of CAM. In to encoder
    match_bus : out std_logic_vector(numOfPatterns-1 downto 0)
  );
end component;

component cam_word is
  generic(
    numOfLuts : integer
  );
  port(
    addr       : in  std_logic_vector(numOfLuts*4-1 downto 0);
    -- Write one bit to X cam_basic cells in parallell
    data       : in  std_logic_vector(numOfLuts-1 downto 0);
    -- Write Enable during 16 clock cycles
    write_en   : in  std_logic;
    clk        : in  std_logic;
    rst        : in  std_logic;
    -- And gate should be disabled during write
    match_en   : in  std_logic;
    -- '1' is the DATA_IN matches the stored data
    match_out  : out std_logic
    );
end component;

component cam_top is
  generic(
    longestPattern : integer := 256;
    addrBits       : integer := 5;
    numOfPatterns  : integer := 32
    );
  port(
    clk              : in  std_logic;
    rst              : in  std_logic;
    -- Data to compare or to write
    cam_data         : in  std_logic_vector(longestPattern-1 downto 0);
    -- Address when write ONLY
    cam_wordaddr_in  : in  std_logic_vector(addrBits-1 downto 0);
       -- Match address
    cam_wordaddr_out : out std_logic_vector(addrBits-1 downto 0);
    cam_write_rdy    : out std_logic;
    -- '1' starts a 16 clock cycle write
    cam_write_en     : in  std_logic;
    -- Enable to find a match, otherwise no change on match bus.
    cam_match_en     : in  std_logic;
    -- '1' if match found
    cam_match        : out std_logic
    );
end component;

component counter is
  port(
    -- one high write_en starts 16 write cycles when going low
    write_en : in  std_logic;
    clk      : in  std_logic;
    rst      : in  std_logic;
    -- Write enable valid during 16 clock cycles
    we       : out std_logic;
    -- Copy of counter value
    cnt      : out std_logic_vector(3 downto 0)
    );
end component;

component encode is
  port(
    -- '1' if match is found
    match     : out std_logic;
    -- Match address
    addr      : out std_logic_vector(4 downto 0);
    -- match_bus from CAM-words
    match_bus : in  std_logic_vector(31 downto 0)
    );
end component;

component cam_basic is
  port(
    data      : in  std_logic;    -- Data to write (one bit at a time)
    write_en  : in  std_logic;
    clk       : in  std_logic;
    addr      : in  std_logic_vector(3 downto 0);
    match_in  : in  std_logic;    -- Input  to   MUXCY (carry-in)
    match_out : out std_logic     -- Output from MUXCY (carry-out)
    );
end component;

end cam_components;
```

tb_cam_top.vhd

```
--
-- File   : tb_cam_top.vhd
-- Project: cam_srl16e
-- Author : Geir Nilsen { geirni@ifi.uio.no }
--
-- This file was created Aug  4 2004 by using a Perl-script, "cam_vhdl.pl"
--
-- cam_srl16e project files:
-- Top level module:  cam_top.vhd
--           module:  |-> cam_words.vhd
--           module:  |--> cam_word.vhd
--           module:  |---> cam_basic.vhd
--           module:  |-> compare.vhd
--           module:  |-> counter.vhd
--           module:  |-> decode.vhd
--           module:  |-> encode.vhd
--        testbench:  tb_cam_top.vhd
--          package:  components.vhd
--

--
-- Behavioral simulation at 10 ns clock cycles.
-- 2 clock cycles delay after match enable goes high
--
-- reset:                          30 ns
-- write:     16*160ns =         5120 ns
-- matching:                       20 ns  Before match
--             32*10ns =        320 ns  Match status
--                               20 ns  Get last 2 matches
--                       _____________
-- Sum:                         5510 ns
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.cam_components.all;

entity testbench is
  generic(
    longestPattern : integer := 256;
    addrBits       : integer := 5;
    numOfPatterns  : integer := 32
    );
end testbench;

architecture testbench of testbench is
  signal   cam_data                                    :
           std_logic_vector(longestPattern-1 downto 0) := (others => '0');
  signal   cam_data_reg                                :
           std_logic_vector(longestPattern-1 downto 0) := (others => '0');
  signal   cam_wordaddr_in                             :
           std_logic_vector(addrBits-1 downto 0)       := (others => '0');
  signal   cam_wordaddr_in_reg                         :
           std_logic_vector(addrBits-1 downto 0)       := (others => '0');
  signal   cam_write_rdy                               :
           std_logic                                   := '0';
  signal   cam_write_en                                :
           std_logic                                   := '0';
  signal   clk                                         :
           std_logic                                   := '0';
  signal   rst                                         :
           std_logic                                   := '0';
  signal   cam_match_en                                :
           std_logic                                   := '0';
  signal   cam_match_en_reg                            :
           std_logic                                   := '0';
  signal   cam_wordaddr_out                            :
           std_logic_vector(addrBits-1 downto 0)       := (others => '0');
  signal   cam_match                                   :
           std_logic                                   := '0';
  constant half_period                                 :
           time                                        := 5 ns;

  -- Bitvectors are set to equal size to make the testbench easier to read.
  -- In hardware the "_00..."-part may be omitted
  type pattern_array is
    array(0 to numOfPatterns-1) of bit_vector (longestPattern-1 downto 0);
  constant pattern : pattern_array := (  -- Content of CAM
    x"66696c656e616d653d5c46697832303031 2e6578655c_00000000000000000000",
    x"6c736f66253230_00000000000000000000000000000000000000000000000000",
    x"6a6176617363726970745c3a2f2f_000000000000000000000000000000000000",
    x"64313368685b_0000000000000000000000000000000000000000000000000000",
    x"2f6578616d706c65732f736572766c65742f536e6f6f70536572766c6574_0000",
    x"616c6c5f7461625f636f6c756d6e73_0000000000000000000000000000000000",
    x"0f0000000373686f7720646174616261736573_00000000000000000000000000",
    x"66696c656e616d653d5c4355504944322e4558455c_0000000000000000000000",
    x"2e6173702e_000000000000000000000000000000000000000000000000000000",
    x"2f766965772d736f75726365_0000000000000000000000000000000000000000",
    x"4142434445464748494a4b4c4d4e4f505152535455565741424344454647484 9",
    x"02010004820100_00000000000000000000000000000000000000000000000000",
    x"2e2e5c5c_00000000000000000000000000000000000000000000000000000000",
    x"2f776169732e706c_000000000000000000000000000000000000000000000000",
    x"496e646578206f66202f6367692d62696e2f_0000000000000000000000000000",
    x"ce63d1d216e713cf39a5a586_0000000000000000000000000000000000000000",
    x"2f73746f72652e636769_00000000000000000000000000000000000000000000",
    x"416d616e6461_0000000000000000000000000000000000000000000000000000",
    x"0000000000000002000186a1_0000000000000000000000000000000000000000",
    x"2f6262735f666f72756d2e636769_000000000000000000000000000000000000",
    x"2f75706c6f616465722e657865_00000000000000000000000000000000000000",
    x"65706c792d746f3a20617e2e602f62696e2f_0000000000000000000000000000",
    x"6c000b000000000000000000_0000000000000000000000000000000000000000",
    x"2f7573722f62696e2f637070_0000000000000000000000000000000000000000",
    x"2f72656769737465722e636769_00000000000000000000000000000000000000",
    x"2f70617373776f72642e6367692e746d70_000000000000000000000000000000",
    x"0000000000000000000000000000000000000000_000000000000000000000000",
    x"2f2e2e2563302561662e2e2f_0000000000000000000000000000000000000000",
    x"2f69697373616d706c65732f_0000000000000000000000000000000000000000",
    x"00556e69780053616d6261_000000000000000000000000000000000000000000",
    x"2b06104014d10219_000000000000000000000000000000000000000000000000",
    x"484541442f2e2f_00000000000000000000000000000000000000000000000000"
    );

  -- A (userdefined) pattern not to be found in CAM
  constant error_pattern : bit_vector(longestPattern-1 downto 0) :=
    x"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB";
begin

uut: cam_top port map(
  cam_data          => cam_data_reg,
  cam_wordaddr_in   => cam_wordaddr_in_reg,
  cam_write_en      => cam_write_en,
  cam_write_rdy     => cam_write_rdy,
  clk               => clk,
  rst               => rst,
  cam_match_en      => cam_match_en_reg,
  cam_wordaddr_out  => cam_wordaddr_out,
  cam_match         => cam_match
  );

clk <= not(clk) after half_period;

tb: process
begin

  rst <= '0'; wait for 2*half_period;
  rst <= '1'; wait for 2*half_period;

  -- Syncronize signals to rising edge
  if not rising_edge(clk) then
    wait for half_period;
  end if;

  -- write new data to CAM. Write to all CAM-locations
  for i in 0 to numOfPatterns-1 loop
    -- example: '1' <= "01"
    cam_wordaddr_in <= conv_std_logic_vector(i,addrBits);
    cam_data        <= to_stdlogicvector(pattern(i));
    if(cam_write_rdy='0') then
```

```
      wait until cam_write_rdy = '1';
    end if;
    -- Start counter
    cam_write_en <= '1'; wait for    2*half_period;
    -- 15 clock-cycles left of writecycle
    cam_write_en <= '0'; wait for 15*2*half_period;
  end loop;

  -- "read" CAM: verify that data has been written to CAM
  cam_match_en <= '1';

  for i in 0 to numOfPatterns-1 loop
    if(i=numOfPatterns-2) then
      cam_data <= to_stdlogicvector(error_pattern); -- Make mismatch
    else
      cam_data <= to_stdlogicvector(pattern(i));
    end if;
    wait for 2*half_period;
  end loop;

  cam_match_en <= '0';

  wait; -- Prevent simulation to wrap around
end process;

-- Registered I/0
io_register: process(rst, clk)
begin
  if(rst = '0') then
    cam_data_reg         <= (others => '0');
    cam_wordaddr_in_reg  <= (others => '0');
    cam_match_en_reg     <= '0';
  else
    if rising_edge(clk) then
      cam_data_reg         <= cam_data;
      cam_wordaddr_in_reg  <= cam_wordaddr_in;
      cam_match_en_reg     <= cam_match_en;
    end if;
  end if;
end process;

end testbench;
```

tb_cam_top.fdo

```
# User defined fdo-file.
# Created Aug  4 2004
vlib       m:/www_docs/research/ise/simulation/cam
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./counter.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./compare.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./decode.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./encode.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./cam_basic.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./components.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./cam_word.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./cam_words.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./cam_top.vhd
vcom -work m:/www_docs/research/ise/simulation/cam -nologo -93 -explicit ./tb_cam_top.vhd
vsim -t 1ps -lib m:/www_docs/research/ise/simulation/cam testbench
view wave
onerror {resume}
```

```
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic   /testbench/rst
add wave -noupdate -format Logic   /testbench/clk
add wave -noupdate -format Literal -radix ascii  /testbench/cam_data
add wave -noupdate -format Literal /testbench/cam_wordaddr_in
add wave -noupdate -format Logic   /testbench/cam_write_rdy
add wave -noupdate -format Logic   /testbench/cam_write_en
add wave -noupdate -format Logic   /testbench/cam_match_en
add wave -noupdate -format Literal /testbench/cam_wordaddr_out
add wave -noupdate -format Logic   /testbench/cam_match
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {0 ps}
WaveRestoreZoom {4950 ns} {5510 ns}
configure wave -namecolwidth 160
configure wave -valuecolwidth 130
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2

run 5510ns
```

F.2 Modules for Debugging

These are the modules used for debugging.

- lcd.vhd
    - Controller for the LCD.
- pButton.vhd
    - A “debouncer”. Gives a *one* clock cycle pulse no matter how long the button is activated.
- led_flash.vhd
    - Ensures that the LED flashes for a given time, even if the duration of the active input is too short to make the LED glow.

lcd.vhd

```
--
--  File    : lcd.vhd
--  Author  : Geir Nilsen { geirni@ifi.uio.no }
--  Created : Oct 13 2003
--
--  Description:
--    Controller to LCD (MDL-16265-LV). Specially designed for a Memec Virtex-II
--    Pro Development Board (DS-BD-2VP4/7-FG456 Rev 4) equipped with an XC2VP7
--

library ieee, unisim;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity lcd is
  generic(
    -- The following parameters are given that "clk" is running at 100 MHz
    -- Timing parameters for write operation
    tcycE : integer := 50;  -- Enable cycle time                 (min 500 ns)
    PWEH  : integer := 23;  -- Enable pulse width (high level)   (min 230 ns)
    tEr   : integer :=  1;  -- Time rise                         (max  20 ns)
    tEf   : integer :=  1;  -- Time fall                         (max  20 ns)
    tAS   : integer :=  4;  -- Address setup time (RS, R/W to E) (min  40 ns)
    tAH   : integer :=  1;  -- Address hold time                 (min  10 ns)
    tDSW  : integer :=  8;  -- Data set-up time                  (min  80 ns)
    tH    : integer :=  1;  -- Data hold time                    (min  10 ns)

    -- The instruction delay needs an n+1 bit counter
    upperDelayIndex : integer := 20  -- All bits = '1' => 20.9 ms
    );
  port(
    clk          : in  std_logic;
    rst          : in  std_logic;
    -- lcd_data(8) = RS (register select)
    lcd_data_in  : in  std_logic_vector(8 downto 0);
    lcd_data_out : out std_logic_vector(8 downto 0);
    lcd_en       : out std_logic;
    -- HandShake: LCD is ready to write a new instruction
    lcd_rdy      : out std_logic;
    -- HandShake: A new instruction is given at lcd_data
    lcd_start    : in  std_logic
    );
```

```
end lcd;

architecture lcd of lcd is

  type lcdInit_type is array(0 to 6) of std_logic_vector(8 downto 0);
  constant lcdInitArray : lcdInit_type := (
    -- Function set: 8 bit data length, 2 lines, 5x8 character font
    "000111000",
    "000111000",
    "000111000",
    "000111000",
    -- Entry mode set: Increment one AC (address counter), shift disabled
    "000000110",
    -- Display on/off control: Display on, cursor off, cursor blink off
    "000001100",
    -- Clear display
    "000000001"
    );
  signal cntInitArray : integer range 0 to lcdInit_type'high+1;
  signal cntInitArrayInc : std_logic;

  type lcdStateType is (
    wait1, wait2, wait3, -- Init only
    idle, -- Wait for lcd_start to go high. Set lcd_rdy high
    start, pullup, dataWait, dataWrite, pulldown, dataHold, -- One write
    wait4, wait5, -- Wait for instruction to complete
    done
    );
  signal lcdState, nextState : lcdStateType;

  -- Counts cycles needed for one write operation
  signal cntWrite    : integer range 0 to tcycE+tAS;
  signal cntWriteInc : std_logic;                     -- Increment cntWrite
  signal cntWriteRst : std_logic;                     -- Reset cntWrite

  -- Counter for delays
  signal cntDelay     : std_logic_vector(upperDelayIndex downto 0);
  signal cntDelayRst  : std_logic;

  -- wait1Delay: Power on init (wait more then 15   ms): "11" after 15.7 ms
  -- wait2Delay: Power on init (wait more than  4.1 ms):  '1' after  5.2 ms
  -- wait3Delay: Power on init (wait more than 100  us):  '1' after 16.3 us
  -- wait4Delay: Instruction delay (clear display, return home, entry mode set)
  --             (wait more than 1.64 ms): "11" after 1.96 ms
  -- wait5Delay: Instruction delay (remaining instructions)
  --             (wait more than   40 us): "11" after 61.4 us
  --             NOTE: 40.9 us is not enough
  alias wait1Delay : std_logic_vector(1 downto 0) is cntDelay(20 downto 19);
  alias wait2Delay : std_logic                    is cntDelay(19);
  alias wait3Delay : std_logic                    is cntDelay(14);
  alias wait4Delay : std_logic_vector(1 downto 0) is cntDelay(17 downto 16);
  alias wait5Delay : std_logic_vector(1 downto 0) is cntDelay(12 downto 11);

-- Uncomment these signals for simulation.
-- Comment out the above duplicate signals
--  signal cntDelay  : std_logic_vector(3 downto 0);
--  alias wait1Delay : std_logic_vector(1 downto 0) is cntDelay(3 downto 2);
--  alias wait2Delay : std_logic                    is cntDelay(0);
--  alias wait3Delay : std_logic                    is cntDelay(0);
--  alias wait4Delay : std_logic_vector(1 downto 0) is cntDelay(1 downto 0);
--  alias wait5Delay : std_logic_vector(1 downto 0) is cntDelay(1 downto 0);
```

```vhdl

  -- Register lcd_data
  signal lcd_data_sig : std_logic;
  signal lcd_data_reg : std_logic_vector(8 downto 0);
begin

lcd_data_out <= lcd_data_reg;

lcd_FSM: process(
  clk, rst, lcdState, cntWrite, cntDelay, lcd_start, cntInitArray, lcd_data_reg,
  wait1Delay, wait2Delay, wait3Delay, wait4Delay, wait5Delay )
begin
  lcd_en          <= '0';
  cntWriteInc     <= '0';
  cntWriteRst     <= '0';
  cntDelayRst     <= '0';
  nextState       <= lcdState;
  lcd_rdy         <= '0';
  lcd_data_sig    <= '0';
  cntInitArrayInc <= '0';

  case lcdState is

    when wait1 =>
      if (wait1Delay="11") then
        nextState <= start;
      else
        nextState <= wait1;
      end if;

    when wait2 =>
      if (wait2Delay='1') then
        nextState <= start;
      else
        nextState <= wait2;
      end if;

    when wait3 =>
      if (wait3Delay='1') then
        nextState <= start;
      else
        nextState <= wait3;
      end if;

    when idle =>
      lcd_rdy <= '1';
      if(lcd_start='1') then  -- Write to LCD
        lcd_data_sig <= '1';
        nextState <= start;
      else
        nextState <= idle;
      end if;

    when start =>
      cntWriteInc <= '1';
      if (cntWrite < tAS-tEr) then
        nextState <= start;
      else
        nextState <= pullup;
      end if;

    when pullup =>
      cntWriteInc <= '1';
      if (cntWrite < tAS) then
        nextState <= pullup;
      else
        nextState <= dataWait;
      end if;

    when dataWait =>
      cntWriteInc <= '1';
      lcd_en      <= '1';
      if (cntWrite < tAS+(PWEH-tDSW)) then
        nextState <= dataWait;
      else
        nextState <= dataWrite;
      end if;

    when dataWrite =>
      cntWriteInc <= '1';
      lcd_en      <= '1';
      if (cntWrite < tAS+PWEH) then
        nextState <= dataWrite;
      else
        nextState <= pulldown;
      end if;

    when pulldown =>
      cntWriteInc <= '1';
      if (cntWrite < tAS+PWEH+tEf) then
        nextState <= pulldown;
      else
        nextState <= dataHold;
      end if;

    when dataHold =>
      cntWriteInc <= '1';
      if (cntWrite < tAS-tEr+tcycE) then
        nextState <= dataHold;
      else
        if (lcd_data_reg(8 downto 3)="000000") then
          nextState <= wait4;  -- 1.64 ms instruction
        else
          nextState <= wait5;  -- 40 us instruction
        end if;
      end if;

    when wait4 =>
      if (wait4Delay="11") then
        nextState <= done;
      else
        nextState <= wait4;
      end if;

    when wait5 =>
      if (wait5Delay="11") then
        nextState <= done;
      else
        nextState <= wait5;
      end if;

    when done =>
```

```
      cntWriteRst <= '1';
      cntDelayRst <= '1';

      if (cntInitArray < lcdInit_type'high+1) then
        cntInitArrayInc<='1';
      end  if;

      if (cntInitArray = 0) then
        nextState <= wait2;
      elsif (cntInitArray=1) then
        nextState <= wait3;
      elsif (cntInitArray < lcdInit_type'high) then
        nextState <= start;
      else
        nextState <= idle;
      end if;

    when others => null;
  end case;
end process;



init_counter: process(rst, clk, cntInitArrayInc)
begin
  if (rst='0') then
    cntInitArray <= 0;
  elsif(rising_edge(clk) and cntInitArrayInc = '1') then
    cntInitArray <= cntInitArray + 1;
  end if;
end process;

lcd_data_register: process(clk,rst, lcd_data_sig, lcd_data_in,cntInitArray)
begin
  if (rst='0') then
    lcd_data_reg <= lcdInitArray(0);
  elsif (rising_edge(clk)) then
    if (cntInitArray < lcdInit_type'high+1) then
      lcd_data_reg <= lcdInitArray(cntInitArray);
    elsif (lcd_data_sig='1') then
      lcd_data_reg <= lcd_data_in;
    end if;
  end if;
end process;

writeCounter: process(clk,rst,cntWriteRst,cntWriteInc)
begin
  if (rst ='0' or cntWriteRst='1') then
    cntWrite <= 0;
  elsif (rising_edge(clk) and cntWriteInc='1') then
    cntWrite <= cntWrite + 1;
  end  if;
end process;

delayCounter: process(clk,rst,cntDelayRst)
begin
  if (rst = '0' or cntDelayRst = '1') then
    cntDelay <= (others  => '0');
  elsif rising_edge(clk) then
    cntDelay <= cntDelay + 1;
  end if;
end process;

lcdStateRegister: process (clk,rst,nextState)
begin
  if (rst='0') then
    lcdState <= wait1;
  elsif rising_edge(clk) then
    lcdState <= nextState;
  end if;
end process;

end lcd;
```

pButton.vhd

```
--
--  File  : pButton.vhd
--  Author: Geir Nilsen { geirni@ifi.uio.no }
--
--  Created: Oct  9 2003
--
--  Description: When pushbutton have given a stable signal for "delay" clock
--               cycles, a valid pulse of one clock cycle is passed on to a
--               register. Before making a new valid pulse the pushbutton must
--               have been relased for "delay" clock cycles.
--

library ieee;
use ieee.std_logic_1164.all;

entity pButton is
  generic(
    delay : integer := 1000000 -- 0.01s at 100 MHz
    );
  port(
    clk        : in   std_logic;
    rst        : in   std_logic;
    pButtonIn  : in   std_logic;
    pButtonOut : out  std_logic
    );
end pButton;

architecture pButton of pButton is
  type   PB_STATE_type is (pb_down, pb_up); -- FSM for pButtonIn-button
  signal pb_state, next_pb_state : PB_STATE_TYPE;

  signal pButtonIn_counter     : integer range 0 to delay;
  signal inc_pButtonIn_counter : std_logic;
  signal pButtonIn_counter_rst : std_logic;

  signal pButtonOut_sig : std_logic;
  signal pButtonOut_reg : std_logic;
begin

pButtonOut <= pButtonOut_reg;

PB_state_machine: process(pb_state,pButtonIn,pButtonIn_counter)
begin

  inc_pButtonIn_counter <= '0';
  pButtonIn_counter_rst <= '0';
  next_pb_state         <= pb_state;
  pButtonOut_sig        <= '0';

  case pb_state is

    -- pButtonIn must have been down for a while before
    -- pButtonOut goes high
    when pb_down =>
      if (pButtonIn = '0') then
        if (pButtonIn_counter < delay) then
          inc_pButtonIn_counter <= '1';
          next_pb_state <= pb_down;
        else
          pButtonOut_sig <= '1';
          pButtonIn_counter_rst <= '1';
          next_pb_state <= pb_up;
        end if;
      else
        pButtonIn_counter_rst <= '1';
        next_pb_state <= pb_down;
      end if;

    -- pButtonIn must have been relased for a while before
    -- pButtonOut can go high again
    when pb_up =>
      if (pButtonIn = '0') then
        pButtonIn_counter_rst <= '1';
        next_pb_state <= pb_up;
      else
        if (pButtonIn_counter < delay) then
          inc_pButtonIn_counter <= '1';
          next_pb_state <= pb_up;
        else
          pButtonIn_counter_rst <= '1';
          next_pb_state <= pb_down;
        end if;
      end if;

  end case;
end process;



pButtonOut_register: process (clk,rst,pButtonOut_sig)
begin
  if (rst='0') then
    pButtonOut_reg <= '1';
  elsif (rising_edge(clk)) then
    if (pButtonOut_sig='1') then
      pButtonOut_reg <= '0';
    else
      pButtonOut_reg <= '1';
    end if;
  end if;
end process;

PB_pButtonIn_counter: process(clk,rst,inc_pButtonIn_counter,
                              pButtonIn_counter_rst)
begin
  if (rst = '0') then
    pButtonIn_counter <= 0;
  elsif (rising_edge(clk)) then
    if (pButtonIn_counter_rst = '1') then
      pButtonIn_counter <= 0;
    end if;
    if (inc_pButtonIn_counter = '1') then
      pButtonIn_counter <= pButtonIn_counter + 1;
    end if;
  end if;
end process;

PB_pButtonIn_statereg: process (clk,rst)
begin
  if (rst = '0') then
    pb_state <= pb_down;
  elsif (rising_edge(clk)) then
    pb_state <= next_pb_state;
  end if;
end process;

end pButton;
```

led_flash.vhd

```
--
-- File    : led_flash.vhd
-- Author  : Geir Nilsen { geirni@ifi.uio.no }
-- Created : Mars 5 2004
--
-- Description:
--   Make flashing leds
--

library ieee;
use ieee.std_logic_1164.all;

entity led_flash is
  generic(
    litetime : integer := 1000000       -- 1/100 s @ 100 MHz
    );
  port(
    clk     : in  std_logic;
    rst     : in  std_logic;
    sig_in  : in  std_logic;
    led_reg : out std_logic
    );
end led_flash;



architecture led_flash of led_flash is
  type led_type is (led_off, led_on);

  signal led_current, led_next : led_type;
  signal led_cnt_inc           : std_logic;
  signal led_reg_sig           : std_logic;
  signal led_cnt               : integer range 1 to litetime;
begin



led_FSM: process(led_current, sig_in, led_cnt)
```

```
begin

  led_cnt_inc <= '0';
  led_reg_sig <= '0';

  case led_current is

    when led_off =>
      if (sig_in='1') then
        led_next <= led_on;
      else
        led_next <= led_off;
      end if;

    when led_on =>
      led_reg_sig <= '1';
      led_cnt_inc <= '1';
      if (led_cnt=litetime) then
        if (sig_in='1') then
          led_next <= led_on;
        else
          led_next <= led_off;
        end if;
      else
        led_next <= led_on;
      end if;

    when others => null;
  end case;
end process;

led_counter: process (rst, clk, led_cnt_inc)
begin
  if (rst='0') then
    led_cnt <= 1;
  elsif(rising_edge(clk)) then
    if (led_cnt=litetime ) then
      led_cnt <= 1;
    elsif (led_cnt_inc='1') then
      led_cnt <= led_cnt + 1;
    end if;
  end if;
end process;

led_FSM_state_register: process(rst, clk)
begin
  if (rst='0') then
    led_current <= led_off;
  elsif (rising_edge(clk)) then
    led_current <= led_next;
  end if;
end process;

led_register: process (rst, clk, led_reg_sig)
begin
  if (rst='0') then
    led_reg <= '1';
  elsif (rising_edge(clk)) then
    if (led_reg_sig='1') then
      led_reg <= '0';
    else
```

```vhdl
      led_reg <= '1';
    end if;
  end if;
end process;

end led_flash;
```

F.3 The PC – FPGA Interface

These are the modules that make a communication between the FPGA and the PC possible.

- ids.c
  - The *User Interface* running on the PC.
- rs232rx.vhd
  - The serial receiver module in the FPGA.
- rs232tx.vhd
  - The serial transmitter module in the FPGA
- devboard.vhd
  - A design for testing the RS232 connection, and the hardware debugging options on the Development Board.
- ids128.vhd
  - Top level module for a 128 word CAM.
- components.vhd
  - Factoring out component declarations to keep a better overview in the other files.
- devboard.ucf
  - User Constraints File. Attach a signal in VHDL to a pin on the FPGA. Nothing will work in the FPGA without having a file like this. This specific file is designed for use with devboard.vhd above.

ids.c

```c
//
//  File: ids.c
//
//  Author: Geir Nilsen { geirni@ifi.uio.no }
//
//  Created February 3 2004
//
//  Description:
//    Designed to work with idsXXX.vhd (ids008.vhd, ids128.vhd etc) and
//    devboard.vhd.. Constants LONGESTPATTERN and NUMOFPATTERNS must be changed
//    each time a new idsXXX.vhd is to be tested; ids.c must then be recompiled.
//
//    The two delay functions, delay_u (microseconds) and delay_m
//    (milliseconds), are tested at a PC with a 466 MHz CPU. These two functions
//    must be rewritten to the spesific CPU that are to be used.
//
//    The serial communication will work when compiled to Win16.
//    Borland C/C++ 4.0 was used to accomplish this.
//

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#define COM1 0x3f8
#define COM2 0x2f8

#define LONGESTPATTERN    60 // hexnumbers----------------ARGV
#define NUMOFPATTERNS      8 // ARGV

//#define LONGESTPATTERN    64 // hexnumbers--------------ARGV
//#define NUMOFPATTERNS      128 // ARGV


#define MAXCAMDATALENGTH  64 // 32 byte (hex)


#define ESC         27
#define clear_LCD    1
#define line1pos0  128
#define line2pos0  192
#define cursor_off  12 // Display on
#define cursor_on   15 // Display on
#define display_off  8
#define cgram       64 // Set CGRAM address 0

#define NOP          0 // Set instruction register in FPGA
#define camData      3 // CAM data shift enable
#define camDataOff   8
#define camAddr      4 // CAM addr shift enable
#define camAddrOff   1
#define camWrite     5 // Trigger
#define camMatchOn   6
#define camMatchOff  7

// Needs a terminating character
char camdata[NUMOFPATTERNS][MAXCAMDATALENGTH + 1];

char *file_camdata = "m:\\www_docs\\research\\ise\\source\\cam\\camdata.txt";
char *file_log     = "m:\\www_docs\\research\\ise\\source\\ids\\ids_log.txt";
FILE *logfile;
char s_in[80]; // Input string from keyboard etc
char command = 0;

void init               (void);
// Load defaults and write to all CAM-words
void CAM_init           (void);
// Write new data to single CAM-word. Update dataset on PC
void CAM_write_update   (int);
// Write new data to single CAM-word. Do not update dataset on PC
void CAM_write_noupdate (void);
// DEBUG: ReWrite one word from dataset
void CAM_rewrite        (void);
// Write one byte to CAM (Help function to the four above)
void CAM_write          (int);
void CAM_verify         (void);
// Display CAM-data (all words)
void display_CAM_data   (void);
// Display single word in CAM-data
void display_CAM_word   (void);
void help               (void);
void getText            (void);
void hrule              (void);
void delay_u            (int);
```

```c
void delay_m            (int);
int  hex2dec            (int, int);
// Test rs232 connection
void rs232              (void);
// Give instruction to LCD
void lcd_inst           (void);
void help_inst          (void);
// Loop through LCD codepage
void lcd_loop_cp        (void);
// instruction, single int - string - single char
void write2lcd          (int, char*, int);
void write2cgram        (int);

// -----------------------------------------------------------------------------

int main(){

  init();

  while(1){
    hrule();
    printf("> ");
    fprintf(logfile, "> ");
    command=tolower(getch());

    if      (command=='1') CAM_init();
    else if (command=='2') CAM_write_update(1);
    else if (command=='3') CAM_write_update(0);
    else if (command=='4') CAM_rewrite();
    else if (command=='5') display_CAM_data();
    else if (command=='6') display_CAM_word();
    else if (command=='7') CAM_verify();

    else if (command=='t') rs232();
    else if (command=='l') lcd_loop_cp();
    else if (command=='i') lcd_inst();
    else if (command=='c') write2cgram(1);
    else if (command=='h') help();
    else if (command==ESC){
      printf("Quit\n\n\n");
      fprintf(logfile, "Quit\n\n\n");
      fclose(logfile);
      write2cgram(2);
      return 0;
    }
    else {
      printf("Unknown command. Type 'h' for help\n");
    }
  }
}

// -----------------------------------------------------------------------------

void init(){
  // Set up UART: 115200 baud, no parity, 8 data bits, 1 stop bit
  // COM1
  outp(COM1 + 3, 0x80); // Set DLAB on
  outp(COM1 + 1, 0x00); // MSB of BAUD rate divisor
  outp(COM1 + 0, 0x01); // LSB of BAUD rate divisor
  outp(COM1 + 3, 0x03); // DLAB off. Set: no parity, 1 stop bit, 8 data bits
  outp(COM1 + 2, 0x47); // Enable FIFO
  //COM2
  outp(COM2 + 3, 0x80); // Set DLAB on
  outp(COM2 + 1, 0x00); // MSB of BAUD rate divisor
  outp(COM2 + 0, 0x01); // LSB of BAUD rate divisor
  outp(COM2 + 3, 0x03); // DLAB off. Set: no parity, 1 stop bit, 8 data bits
  outp(COM1 + 2, 0x47); // Enable FIFO

  hrule();
  if((logfile=fopen(file_log, "w"))==NULL){
    printf("  Could not open file \"%s\" for writing\n\n", file_log);
    printf("  No logfile will be written\n");
    return;
  }
  else {
    printf("  Log to file \"%s\"\n\n", file_log);
    printf("  Type 'h' for help\n");
  }

  return;
}

// ---------------------------------------

void CAM_init(){
  FILE *fp;
  char c=0;
  int  i=0;
  int  j=0;
  int  count=0;

  printf("CAM Init\n\n");
  fprintf(logfile, "CAM Init\n\n");

  if((fp=fopen(file_camdata, "r"))==NULL){
    printf("  Could not open file \"%s\"\n\n", file_camdata);
    printf("  Init aborted\n");
    fprintf(logfile, "  Could not open file \"%s\"\n\n", file_camdata);
    fprintf(logfile, "  Init aborted\n");
    return;
  }

  printf("  Reading file \"%s\"\n\n", file_camdata);
  fprintf(logfile, "  Reading file \"%s\"\n\n", file_camdata);

  for(i=0; i<=NUMOFPATTERNS-1; i++){
    while((c=getc(fp)) != '\n'){ // Read line
      camdata[i][j] = c;
      count++;
      j++;
    }
    camdata[i][j] = 0; // Terminate string
    j=0;
  }
  fclose(fp);

  //Call CAM_write for each word
  for(i=0; i<=NUMOFPATTERNS-1; i++){
    if(kbhit()){ // Break
      getch();
      return;
    }
    CAM_write(i);
  }

  printf("\n  Size of CAM is %d bytes, %d words\n\n", count/2, NUMOFPATTERNS);
  printf("  Init Complete\n");
  fprintf(logfile,
          "\n  Size of CAM is %d bytes, %d words\n\n", count/2, NUMOFPATTERNS);
  fprintf(logfile, "  Init Complete\n");
  return;
}



void CAM_write(int word){
  int i = 0;
  int CAM_byte = 0;
  int c = 0;
  int lowbyte = 0;
  int highbyte = 0;
  int sum = 0;

  //printf("  lowbyte(0) %d \n", low_byte);

  outp(COM2, camAddr); // Set FPGA instruction register: CAM address
  outp(COM1, word);    // Set CAM address LSB
  outp(COM2, camAddrOff);
  outp(COM2, camData); // Set FPGA instruction register:
                       //Write to cam_data_shiftreg. MSB first

  for(i=strlen(camdata[word])-2; i>=0; i-=2){
    CAM_byte = hex2dec(word, i);
    outp(COM1, CAM_byte); // Write to datareg
  }

  // Fill LSB with zeroes
  for(i=0; i<(LONGESTPATTERN-strlen(camdata[word]))/2; i++)
    outp(COM1, 0);

  outp(COM2, camDataOff);
  outp(COM2, camWrite); // Set FPGA instruction register:
                        //Enable CAM write (Clears instreg in FPGA)

  // Verify write
  outp(COM2, camMatchOn);



  // lowbyte is sent first from FPGA, but highbyte must be read first
  delay_m(100); // highbyte
  c = inp(COM2 + 5);
  if(c & 1){
    highbyte = inp(COM2);
    //printf("  highbyte %3d  ", highbyte);
  }

  delay_m(100); // lowbyte
  c = inp(COM2 + 5);
  if(c & 1){
    lowbyte = inp(COM2);
    printf("  lowbyte %3d \n", lowbyte);
  }

  sum = highbyte+lowbyte;//------------------------------------------------- FIX
  //printf("Sum: %d\n", sum);

  if(sum==word){
    printf("  Word in address %3d is written to CAM\n", word);
    fprintf(logfile, "  Word in address %3d is written to CAM\n", word);
  }
  else{
    printf("  Error! Word in address %3d is not written to CAM\n", word);
    fprintf(logfile,
            "  Error! Word in address %3d is not written to CAM\n", word);
  }
  outp(COM2, camMatchOff);
  return;
}

void CAM_write_update(int cam){
  int i, j   = 0;
  int c      = 0;
  int word   = 0;
  int update = 1; // Assume no subset
  printf("Write to single CAM-word. Update dataset on PC\n\n");
  printf("  Press ESC to skip char\n");
  printf("  Type any non-hex digit to skip remaining chars\n\n");
  printf("  Address => ");
  fprintf(logfile, "Write to single CAM-word. Update dataset on PC\n\n");
  fprintf(logfile, "  Press ESC to skip char\n");
  fprintf(logfile, "  Type any non-hex digit to skip remaining chars\n\n");
  fprintf(logfile, "  Address => ");
  getText();
  word = atoi(s_in);
  printf("\n");
  printf("  Addr Len Data (hex)\n");
  printf("%6d%4d %s\n", word, strlen(camdata[word]), camdata[word]);
  printf("\n");
  printf("  New   => ");
  fprintf(logfile, "\n");
  fprintf(logfile, "  Addr Len Data (hex)\n");
  fprintf(logfile, "%6d%4d %s\n", word, strlen(camdata[word]), camdata[word]);
  fprintf(logfile, "\n");
  fprintf(logfile, "  New   => ");

  // Get new data
  for(i=0; i<=strlen(camdata[word])-1; i++){
    c = tolower(getch());
    if(c==ESC){
      s_in[i] = camdata[word][i];
      printf("%c", camdata[word][i]);
      fprintf(logfile, "%c", camdata[word][i]);
    } else if(isxdigit(c)){
      s_in[i] = c;
      printf("%c", c);
      fprintf(logfile, "%c", c);
    } else {
      for(j=i; j<=strlen(camdata[word])-1; j++){
        s_in[j] = camdata[word][j];
        printf("%c", camdata[word][j]);
        fprintf(logfile, "%c", camdata[word][j]);
      }
      i=strlen(camdata[word])-1;
      printf("\n");
      fprintf(logfile, "\n");
    }
    s_in[i+1] = '\0';
  }

  // check subsets etc
  printf("\n");
  fprintf(logfile, "\n");
  for(i=0; i<=NUMOFPATTERNS-1; i++){
    if(strlen(s_in) <= strlen(camdata[i])){
      for(j=0; s_in[j]==camdata[i][j]; j++){
        if(s_in[j] == '\0'){
          printf("  Subset found at address %d.", i);
          fprintf(logfile, "  Subset found at address %d.", i);
          update = 0;
          break;
        }
      }
    }
  }

  if(update){
    strcpy(camdata[word], s_in); // Update dataset on PC
    if(cam)
      CAM_write(word);             // Update CAM
    printf("\n");
    printf("  Update in address %d. PC and CAM updated\n", word);
    printf("  Old val  %s\n", camdata[word]);
    printf("  New val  %s\n", camdata[word]);
    fprintf(logfile, "\n");
    fprintf(logfile, "  Update in address %d. PC and CAM updated\n", word);
    fprintf(logfile, "  Old val  %s\n", camdata[word]);
    fprintf(logfile, "  New val  %s\n", camdata[word]);
  } else {
    printf("  No update of dataset\n");
    fprintf(logfile, "  No update of dataset\n");
  }
  return;
}



void CAM_write_noupdate(){
  return;
}



void CAM_verify(){
  char c = 0;
  //int getC = 0;
  int i = 0;
  int j = 0;
  //int k = 0;
  int m = 0;
  int highbyte = 0;
  int lowbyte = 0;
  int sum = 0;

  printf("Verify CAM contents\n\n");
```

```
fprintf(logfile, "Verify CAM contents\n\n");
// Set FPGA instruction register: Write to cam_data_shiftreg. MSB first
outp(COM2, camData);

// Write all words to cam_data_shiftreg
for(i=0; i<=NUMOFPATTERNS-1; i++){

  for(j=strlen(camdata[i])-2; j>=0;  j-=2){
    if(kbhit()){ // Break
      getch();
      return;
    }
    m++;
    if(m==LONGESTPATTERN/2-1)
      outp(COM2, camMatchOn);
    outp(COM1, hex2dec(i, j)); // Write to datareg
    printf("  Word %3d Pos %2d %2d: %c%c\n",
           i, j, j+1, camdata[i][j], camdata[i][j+1]);
    fprintf(logfile, "  Word %d Pos %2d %2d: %c%c\n",
            i, j, j+1, camdata[i][j], camdata[i][j+1]);


    // lowbyte comes first here
    delay_m(100); // highbyte
    c = inp(COM2 + 5);
    if(c & 1){
      lowbyte = inp(COM2);
      //printf("  lowbyte %3d \n", lowbyte);

      c = inp(COM2 + 5);
      if(c & 1){
        highbyte = inp(COM2);
        // printf("  highbyte %3d \n ", highbyte);
      }
      sum = highbyte+lowbyte;
      printf("                          FPGA match address: %3d\n", sum);
      fprintf(logfile,
              "                          FPGA match address: %3d\n", sum);
    }
  }// for j
} // for i

// Shift remaining patterns through CAM by sendig zeroes
for(j=0; j<=LONGESTPATTERN/2; j++){
  printf("                      00\n");
  fprintf(logfile, "                     00\n");
  outp(COM1, 0);

  delay_m(100); // highbyte
  c = inp(COM2 + 5);
  if(c & 1){
    lowbyte = inp(COM2);
    //printf("  lowbyte %3d \n", lowbyte);
    delay_m(100); // lowbyte
    c = inp(COM2 + 5);
    if(c & 1)
      highbyte = inp(COM2);//----------------------------------------------FIX
    //printf("  highbyte %3d \n ", highbyte);

    sum = highbyte+lowbyte;//---------------------------------
    printf("                          FPGA match address: %3d\n", sum);
```

```
      fprintf(logfile,
              "                          FPGA match address: %3d\n", sum);
    }
  }// for j

  outp(COM2, camMatchOff);
  outp(COM2, camDataOff);
  return;
}



void CAM_rewrite(){
  int word = 0;
  //int c, getC;

  printf("Write word from dataset to CAM\n\n");
  printf("  Address => ");
  fprintf(logfile, "Write word from dataset to CAM\n\n");
  fprintf(logfile, "  Address => ");
  getText();
  word = atoi(s_in);
  printf("\n");
  printf("  Write to CAM: \n\n");
  printf("  Addr Len Data (hex)\n");
  printf("%6d%4d %s\n", word, strlen(camdata[word]), camdata[word]);
  printf("\n");
  fprintf(logfile, "\n");
  fprintf(logfile, "  Write to CAM: \n\n");
  fprintf(logfile, "  Addr Len Data (hex)\n");
  fprintf(logfile, "%6d%4d %s\n", word, strlen(camdata[word]), camdata[word]);
  fprintf(logfile, "\n");

  CAM_write(word);
  return;
}



int hex2dec(int word, int pos){
  int dec1, dec2, dec3 = 0;

  if(isdigit(camdata[word][pos]))
    dec1 = camdata[word][pos]   - 48;
  else
    dec1 = camdata[word][pos]   - 87;

  if(isdigit(camdata[word][pos+1]))
    dec2 = camdata[word][pos+1] - 48;
  else
    dec2 = camdata[word][pos+1] - 87;

  dec3 = dec1;
  dec3 <<= 4;
  dec3 += dec2; // Byte to write to CAM
  return dec3;
}



void display_CAM_data(){
  int i=0;
  printf("Display CAM data (all words)\n\n");
  printf("  Addr Len Data (hex)\n");
  fprintf(logfile, "Display CAM data (all words)\n\n");
  fprintf(logfile, "  Addr Len Data (hex)\n");
  for(i=0; i<=NUMOFPATTERNS-1; i++){
    printf("%6d%4d %s\n", i, strlen(camdata[i]), camdata[i]);
    fprintf(logfile, "%6d%4d %s\n", i, strlen(camdata[i]), camdata[i]);
  }
  return;
}



void display_CAM_word(){
  printf("Display CAM word (single word)\n\n");
  printf("  Address => ");
  getText();
  printf("\n");
  printf("  Addr Len Data (hex)\n");
  printf("%6d%4d %s\n",
         atoi(s_in), strlen(camdata[atoi(s_in)]), camdata[atoi(s_in)]);
  fprintf(logfile, "Display CAM word (single word)\n\n");
  fprintf(logfile, "  Address => ");
  fprintf(logfile, "\n");
  fprintf(logfile, "  Addr Len Data (hex)\n");
  fprintf(logfile, "%6d%4d %s\n",
          atoi(s_in), strlen(camdata[atoi(s_in)]), camdata[atoi(s_in)]);
  return;
}

// -----------------------------------------------------------------------------

void help(){ // Use DOS graphics
  printf("Help\n");
  printf("\n");
  printf("  ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ Main Menu ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»\n");
  printf("  º  Project ids:                              º\n");
  printf("  º    1 = CAM Init (write to all CAM-words)   º\n");
  printf("  º    2 = Write word to dataset on PC and CAM º\n");
  printf("  º    3 = Write word to dataset on PC         º\n");
  printf("  º    4 = Write word from dataset to CAM       º\n");
  printf("  º    5 = Display  CAM-data (all words)       º\n");
  printf("  º    6 = Display  CAM-word (single word)     º\n");
  printf("  º    7 = Verify   CAM-data                   º\n");
  printf("  ÇÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¶\n");
  printf("  º  Debug options:                            º\n");
  printf("  º    t = Test of rs232rx.vhd and rs232tx.vhd º\n");
  printf("  º    l = Loop LCD CodePage                   º\n");
  printf("  º    i = Give LCD-instruction                º\n");
  printf("  º    c = Write to CG RAM                     º\n");
  printf("  ÇÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¶\n");
  printf("  º  Other commands:                           º\n");
  printf("  º    h = Help                                º\n");
  printf("  º  Esc = Quit                                º\n");
  printf("  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼\n");
}



void getText(){
  while(strlen(fgets(s_in, 80, stdin))<1);
  s_in[strlen(s_in)-1]='\0';
  fflush(stdin);
}



void hrule(){
  printf("\n");
  printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ");
  printf("ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ\n");
}



void delay_u(int usec){ // usec (based on delay_m)
  int i, j, k;
  for(i=0; i<=46; i++)
    for(j=0; j<=50; j++)
      for(k=0; k<=usec; k++);
  return;
}



void delay_m(int msec){ // msec (based on experiments)
  int i, j, k;
  for(i=0; i<=460; i++)
    for(j=0; j<=500; j++)
      for(k=0; k<=msec; k++);
  return;
}



void rs232(){
  int getC  = 0;
  int sendC = 0;
  int c     = 0;
  int LCD_count = 0;
  printf("Test rs232tx.vhd and\n");
  printf("     rs232rx.vhd\n\n");
  printf("                 ");
  printf("Send \"00011011\" or press ESC to return to main program\n\n");

  write2lcd(clear_LCD, 0, 0);
  write2lcd(line1pos0, 0, 0); write2lcd(0, "    Test of:    ", 0);
  write2lcd(line2pos0, 0, 0); write2lcd(0, "rs232tx  rs232rx", 0);
  write2lcd(line1pos0, 0, 0);

  while(1){

    c = inp(COM1 + 5); // Check to see if new char has been recived from FPGA
    if(c & 1){
      getC = inp(COM1);
      if(getC==ESC){
        printf("\n");
        return;
      }
      else {
        printf("%c", getC);
    LCD_count++;
    if(LCD_count==16)
      write2lcd(line2pos0, 0, 0);
    else if(LCD_count==32){
      write2lcd(line1pos0, 0, 0);
      LCD_count = 0;
    }
   }
  } // FPGA

  if(kbhit()){ // Check to see if new char has been typed at keyboard
   sendC=getch();
   if(sendC==ESC){
    printf("\n");
    outp(COM2, 0);
    return;
   }
   else {
    printf("%c", sendC);
    write2lcd(0, 0, sendC);
    LCD_count++;
    if(LCD_count==16){
      write2lcd(line2pos0, 0, 0);
    }
    else if(LCD_count==32){
      write2lcd(line1pos0, 0, 0);
      LCD_count = 0;
    }
   }
  } // Keyboard

 }
}



void lcd_inst(){
 printf("Give LCD-instruction\n");
 printf("\n");
 help_inst();

 while(1){
  command=tolower(getch());

  if (command=='1'){
   printf("  Clear\n");
   write2lcd(clear_LCD, 0, 0);
  }
  else if (command=='2'){
   printf("  Set DDRAM addr: Line 1 Pos 0\n");
   write2lcd(line1pos0, 0, 0);
  }
  else if (command=='3'){
   printf("  Set DDRAM addr: Line 2 Pos 0\n");
   write2lcd(line2pos0, 0, 0);
  }
  else if (command=='4'){
   printf("  Cursor off\n");
   write2lcd(cursor_off, 0, 0);
  }
  else if (command=='5'){
      printf("  Cursor on\n");
      write2lcd(cursor_on, 0, 0);
    }
    else if (command=='6'){
      printf("  Display off\n");
      write2lcd(display_off, 0, 0);
    }
    else if (command=='h')
      help_inst();
    else if (command==ESC){
      outp(COM2, 0);
      return;
    }
    else {
      printf("  Unknown command. Type 'h' for help\n");
    }
  }
}



void help_inst(){
  printf("  help\n");
  printf("  ÉÍÍÍÍÍÍÍÍÍÍÍ LCD-instruction Menu ÍÍÍÍÍÍÍÍÍÍÍ»\n");
  printf("  º                                            º\n");
  printf("  º    1 = Clear                               º\n");
  printf("  º    2 = Set DDRAM addr: Line 1 Pos 0        º\n");
  printf("  º    3 = Set DDRAM addr: Line 2 Pos 0        º\n");
  printf("  º    4 = Cursor off  (Display on)            º\n");
  printf("  º    5 = Cursor on   (Display on)            º\n");
  printf("  º    6 = Display off                         º\n");
  printf("  º                                            º\n");
  printf("  º    h = help                                º\n");
  printf("  º  Esc = Back to main program                º\n");
  printf("  º                                            º\n");
  printf("  ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼\n");
  return;
}



void lcd_loop_cp(){
  int i=0;
  printf("Loop LCD CodePage\n");
  printf("\n");
  printf("  Press any key to abort\n");

  write2lcd(cursor_off, 0, 0);
  write2lcd(clear_LCD, 0, 0);
  write2lcd(line1pos0, 0, 0);
  write2lcd(0, " Code Page A-00 ", 0);

  for(i=0; i<=255; i++){
    write2lcd(line2pos0, 0, 0);
    write2lcd(0, "    '", 0);
    write2lcd(0, 0, i);
    write2lcd(0, "' ", 0);
    write2lcd(0, itoa(i, s_in, 10), 0);

    if(i==0)
      delay_m(3000);
    else if ( (16<=i && i<=31) || (128<=i && i<=159) ) // Blank columns
      delay_m(50);
    else
      delay_m(500);

    if(kbhit()){
      getch();
      return;
    }
  }

  outp(COM2, 0); // Set instruction register in FPGA
  return;
}



void write2lcd(int cmd, char *s, int c){
  int i=0;
  if(cmd){
    outp(COM2, 1);     // FPGA instruction register: Shift enable lcd_data_in
    outp(COM1, 0);     // MSB lcd_data_in. LCD register select: Register
    outp(COM1, cmd);   // LSB lcd_data_in. D7-D0
    outp(COM2, 2);     // Write lcd_data_in to LCD
    if(cmd==clear_LCD)
      delay_m(2);
    else
      delay_u(50);
  }
  else if(strlen(s)>0){
    for(i=0; i<=strlen(s)-1; i++){
      outp(COM2, 1);
      outp(COM1, 1);    // LCD register select: Data
      outp(COM1, s[i]);
      outp(COM2, 2);
      delay_u(50);
    }
  }
  else {
    outp(COM2, 1);
    outp(COM1, 1);
    outp(COM1, c);
    outp(COM2, 2);
    delay_u(50);
  }
  return;
}



void write2cgram(int choice){
  int i = 0;

  if(choice==1){
    printf("Write to CG RAM\n\n");

    printf("  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ \n");
    printf("  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þþ þþ \n");
    printf("  þ   þ  þ þ þ  þ þ þ  þþ þþ  þþ þþ  þþ þþ  þþ þþ  þ þ þ \n");
    printf("  þ þ þ  þ   þ  þ þ þ  þ   þ  þ þ þ  þþ þþ  þþþþþ  þþ þþ \n");
    printf("  þ   þ  þ þ þ  þ þ þ  þþ þþ  þþ þþ  þþ þþ  þþ þþ  þ þ þ \n");
    printf("  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þ   þ  þþ þþ \n");
    printf("  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ  þþþþþ \n");

    write2lcd(clear_LCD, 0, 0);
    write2lcd(cgram, 0, 0);

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 0); //

    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 27); // 11 11
    write2lcd(0, 0, 31); // 11111
    write2lcd(0, 0,  0); //

    write2lcd(line1pos0, 0, 0);
    write2lcd(0, " Test of CG RAM ", 0);
    write2lcd(line2pos0, 0, 0);
    write2lcd(0, "    ", 0);

    for(i=0; i<=7; i++)
      write2lcd(0, 0, i);

  } // choice==1


  else if(choice==2){
    printf("  Design by:\n\n");

    //      |     |     |     |     |     |     |     |     |
    printf("  þþþþ þþþ þ þþþþ                                \n");
    printf("  þ    þ   þ þ  þ                                \n");
    printf("  þ þþ þþ  þ þþþþ                                \n");
    printf("  þ  þ þ   þ þ þ   þ   þ þ þ    þþþ   þþþ  þ   þ \n");
    printf("  þþþþ þþþ þ þ  þ  þþ  þ þ þ    þ     þ    þþ  þ \n");
    printf("                   þ þ þ þ þ    þþþ   þþ   þ þ þ \n");
    printf("                   þ  þþ þ þ      þ   þ    þ  þþ \n");
    printf("                   þ   þ þ þþþ  þþþ   þþþ  þ   þ \n");

    printf("\n");

    write2lcd(clear_LCD, 0, 0);
    write2lcd(cursor_off, 0, 0);
    write2lcd(cgram, 0, 0);

    //                      |||||
    write2lcd(0, 0, 15); //  1111
    write2lcd(0, 0,  8); //  1
    write2lcd(0, 0, 11); //  1 11
    write2lcd(0, 0,  9); //  1  1
    write2lcd(0, 0, 15); //  1111
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    //                      |||||
    write2lcd(0, 0, 29); // 111 1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 25); // 11  1
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 29); // 111 1
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    //                      |||||
    write2lcd(0, 0, 30); // 1111
    write2lcd(0, 0, 18); // 1  1
    write2lcd(0, 0, 30); // 1111
    write2lcd(0, 0, 20); // 1 1
    write2lcd(0, 0, 18); // 1  1
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    //                      |||||
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 25); // 11  1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 19); // 1  11
    write2lcd(0, 0, 17); // 1   1
    //                      |||||
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0, 20); // 1 1
    write2lcd(0, 0, 20); // 1 1
    write2lcd(0, 0, 20); // 1 1
    write2lcd(0, 0, 20); // 1 1
    write2lcd(0, 0, 23); // 1 111
    //                      |||||
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0, 14); //  111
    write2lcd(0, 0,  8); //  1
    write2lcd(0, 0, 14); //  111
    write2lcd(0, 0,  2); //    1
    write2lcd(0, 0, 14); //  111
    //                      |||||
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0, 14); //  111
    write2lcd(0, 0,  8); //  1
    write2lcd(0, 0, 12); //  11
    write2lcd(0, 0,  8); //  1
    write2lcd(0, 0, 14); //  111
    //                      |||||
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0,  0); //
    write2lcd(0, 0, 17); // 1   1
    write2lcd(0, 0, 25); // 11  1
    write2lcd(0, 0, 21); // 1 1 1
    write2lcd(0, 0, 19); // 1  11
    write2lcd(0, 0, 17); // 1   1
    //                      |||||

    write2lcd(line1pos0, 0, 0);
    write2lcd(0, "   Design by:   ", 0);
    write2lcd(line2pos0, 0, 0);
    write2lcd(0, "    ", 0);

    for(i=0; i<=7; i++)
      write2lcd(0, 0, i);

  } // choice==2


  return;
}
```

rs232rx.vhd

```
--
--  File    : rs232rx.vhd
--  Author  : Geir Nilsen { geirni@ifi.uio.no }
--  Created : Dec 15 2003
--
--  Description:
--     This controller is designed to work with the following parameters:
--     100 MHz input clock
--     Baud:      115200, 9600
--     Parity:    None
--     Data Bits: 8
--     Stop Bits: 1
--     --------------------------------------------------------------------------
--     Calculate ns/cycle @ 115200 baud
--     (1 s) / (115200 cycles) = 8680.55 ns
--
--     Ideel  10 cycles = 86805.55 ns
--     Actual 10 cycles = 86800          => halfperiod = 4340 ns
--     Difference       =     5.55 ns
--     --------------------------------------------------------------------------
--     Calculate ns/cycle @   9600 baud
--     (1 s) / (9600 cycles) = 104166.66 ns
--
--     Ideel  10 cycles = 1041666.66 ns
--     Actual 10 cycles = 1041600    ns  => halfperiod = 52080 ns
--     Difference       =      66.66 ns
--     --------------------------------------------------------------------------
--       baud  divisor   halfperiod (halfperiod must be even (see stopbit))
--     115200        1          434
--       9600       12         5208
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity rs232rx is
  generic(
    divisor    : integer :=   1;
    half       : integer := 434
    );
  port(
    clk         : in  std_logic;                    -- 100 MHz
    rst         : in  std_logic;
    rxd         : in  std_logic;                    -- Bit to FPGA from DP9
    rx_data     : out std_logic_vector(7 downto 0); -- Bits from rxd
    rx_data_rdy : out std_logic                     -- '1' when rx_data is ready
    );
end rs232rx;

architecture rs232rx of rs232rx is
  type rs232_type is ( idle,
                       startbit,
                       bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7,
                       stopbit
                       );
  signal rx_current, rx_next : rs232_type;

  signal rx_shift_en      : std_logic;
  signal rx_data_reg      : std_logic_vector(7 downto 0);
  signal rx_data_rdy_sig  : std_logic;
  signal rx_data_rdy_reg  : std_logic;

  type   rxdatardy_type is ( rxdatardy_idle,
                             rxdatardy_wait );
  signal rxdatardy_current, rxdatardy_next : rxdatardy_type;
  signal rx_data_rdy_sig2 : std_logic;

  signal half_period : integer range 1 to divisor * half;
  signal cnt         : integer range 1 to divisor * half;
  signal rs232clk    : std_logic;

  signal rx_start_reg : std_logic;
begin

--------------------------------------------------------------------------------

rx_data     <= rx_data_reg;
rx_data_rdy <= rx_data_rdy_reg;

--------------------------------------------------------------------------------

rx_FSM:process (rx_current, rxd, rx_start_reg)
begin

  rx_data_rdy_sig <= '0';
  rx_shift_en     <= '0';
  half_period <= divisor * half;

  case rx_current is

    when idle =>                    -- Wait for startbit
      half_period <= 1;
      if (rx_start_reg='1') then
        rx_next <= startbit;
      else
        rx_next <= idle;
      end if;

    when startbit => rx_next <= bit0;
    when bit0     => rx_next <= bit1;    rx_shift_en <= '1';
    when bit1     => rx_next <= bit2;    rx_shift_en <= '1';
    when bit2     => rx_next <= bit3;    rx_shift_en <= '1';
    when bit3     => rx_next <= bit4;    rx_shift_en <= '1';
    when bit4     => rx_next <= bit5;    rx_shift_en <= '1';
    when bit5     => rx_next <= bit6;    rx_shift_en <= '1';
    when bit6     => rx_next <= bit7;    rx_shift_en <= '1';
    when bit7     => rx_next <= stopbit; rx_shift_en <= '1';

    when stopbit  =>
      -- Skip to idle halfway, to be sure not to miss the next startbit
      half_period <= (divisor * half) / 2;
      rx_data_rdy_sig <= '1';
      rx_next         <= idle;

    when others =>
      rx_next <= idle;
  end case;
end process;

-------------------------------------------------------------------------------

rxdatardy_FSM: process (rxdatardy_current, rx_data_rdy_sig)
begin

  rxdatardy_next <= rxdatardy_current;
  rx_data_rdy_sig2 <= '0';

  case rxdatardy_current is

    when rxdatardy_idle =>
      if (rx_data_rdy_sig='1') then
        rx_data_rdy_sig2<='1';
        rxdatardy_next <= rxdatardy_wait;
      else
        rxdatardy_next <= rxdatardy_idle;
      end if;

    when rxdatardy_wait =>
      if (rx_data_rdy_sig='1') then
        rxdatardy_next <= rxdatardy_wait;
      else
        rxdatardy_next <= rxdatardy_idle;
      end if;

  end case;
end process;

-------------------------------------------------------------------------------

rx_start_register: process (rst, clk, rxd, rx_current)
begin
  if (rst='0') then
    rx_start_reg <= '0';
  elsif (rising_edge(clk)) then
    if (rx_current=idle and rxd='0') then
      rx_start_reg <= '1';
    elsif (rx_current=bit7) then
      rx_start_reg <= '0';
    end if;
  end if;
end process;

rx_data_shiftregister: process (rst, rs232clk, rx_shift_en, rx_data_reg)
begin
  if (rst='0') then
    rx_data_reg <= (others => '0');
```

```vhdl
  elsif (falling_edge(rs232clk)) then     -- Make sample halfway
    if (rx_shift_en='1') then
      rx_data_reg <= rxd & rx_data_reg(7 downto 1);
    end if;
  end if;
end process;

rx_data_rdy_register: process (rst, clk, rx_data_rdy_sig2)
begin
  if (rst='0') then
    rx_data_rdy_reg <= '0';
  elsif (rising_edge(clk)) then
    if (rx_data_rdy_sig2='1') then
      rx_data_rdy_reg<='1';
    else
      rx_data_rdy_reg <= '0';
    end if;
  else
    rx_data_rdy_reg <= rx_data_rdy_reg;
  end if;
end process;

rx_stateRegister: process (rst, rs232clk, rx_next)
begin
  if (rst='0') then
    rx_current <= idle;
  elsif (rising_edge(rs232clk)) then
    rx_current <= rx_next;
  end if;
end process;

rxdatardy_statereg: process (clk,rst)
begin
  if (rst='0') then
    rxdatardy_current <= rxdatardy_idle;
  elsif (rising_edge(clk)) then
    rxdatardy_current <= rxdatardy_next;
  else
    rxdatardy_current <= rxdatardy_current;
  end if;
end process;

-------------------------------------------------------------------------------

rs232clk_generator: process (clk, rst, half_period)
begin
  if (rst = '0') then
    cnt <= 1;
    rs232clk <= '1';
  elsif (rising_edge(clk)) then
    if (cnt = half_period) then
      rs232clk <= not rs232clk;
      cnt <= 1;
    else
      cnt <= cnt + 1;
    end if;
  end if;
end process;

end rs232rx;
```

rs232tx.vhd

```vhdl
--
--  File    : rs232tx.vhd
--  Author  : Geir Nilsen { geirni@ifi.uio.no }
--  Created : Dec 15 2003
--
--  Description:
--     This controller is designed to work with the following parameters:
--     100 MHz input clock
--     Baud:      115200, 9600
--     Parity:    None
--     Data Bits: 8
--     Stop Bits: 1
--     -------------------------------------------------------------------------
--     Calculate ns/cycle @ 115200 baud
--     (1 s) / (115200 cycles) = 8680.55 ns
--
--     Ideel  10 cycles = 86805.55 ns
--     Actual 10 cycles = 86800          => halfperiod = 4340 ns
--     Difference       =     5.55 ns
--
--     Delay must be 1 clock cycle due to the difference
--     1 extra clock cycle is added to give the UART at the reciving end time to
--     prepare for the next byte
--     => 1 cycle delay per halfperiod
--     -------------------------------------------------------------------------
--     Calculate ns/cycle @   9600 baud
--     (1 s) / (9600 cycles) = 104166.66 ns
--
--     Ideel  10 cycles = 1041666.66 ns
--     Actual 10 cycles = 1041600    ns  => halfperiod = 52080 ns
--     Difference       =      66.66 ns
--
--     Delay must be 7 clock cycles due to the difference.
--     1 extra clock cycle is added to give the UART at the reciving end time to
--     prepare for the next byte
--     => 4 cycles delay per halfperiod
--     -------------------------------------------------------------------------
--       baud  divisor   halfperiod delay
--     115200        1          434     1
--       9600       12         5208     4
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity rs232tx is
  generic(
    divisor : integer :=   1;     -- 115200 baud
    half    : integer := 434
    );
  port(
    clk       : in  std_logic;                    -- 100 MHz
    rst       : in  std_logic;
    tx_rdy    : out std_logic;                    -- Ready to send data to DP9
    tx_start  : in  std_logic;                    -- Start sendig tx_data
    tx_data   : in  std_logic_vector(7 downto 0); -- Byte to write to DP9
                                                  -- (from FPGA) using txd
    txd       : out std_logic                     -- Bit from FPGA to DP9
    );
end rs232tx;

architecture rs232tx of rs232tx is
  type rs232_type is (idle,
                      startbit,
                      bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7,
                      stopbit
                      );
  signal tx_current, tx_next : rs232_type;

  signal tx_start_reg : std_logic;

  signal tx_data_reg  : std_logic_vector(9 downto 0);
  signal tx_shift_en  : std_logic;
  signal tx_load      : std_logic;

  -- ad hoc ( find max-delay ) Make delay generic ?
  signal delay       : integer range 1 to 10;
  signal half_period : integer range 1 to (divisor * half) + 10;
  signal cnt         : integer range 1 to (divisor * half) + 10;
  signal rs232clk    : std_logic;
begin

txd <= tx_data_reg(0);

delay <= 1 when divisor =  1 else -- 115200 baud
         4 when divisor = 12 else --   9600 baud
         4;                       -- Add delay(s) to other baudrates

rs232_FSM: process (tx_current, tx_start, tx_start_reg)
begin

  tx_rdy       <= '0';
  tx_load      <= '0';
  tx_shift_en  <= '1';
  half_period  <= divisor * half;

  case tx_current is

    when idle =>
      half_period <= 1;
      tx_rdy      <= '1';
      tx_shift_en <= '0';
      if (tx_start_reg='1') then
        tx_load<='1';
        tx_next <= startbit;
      else
        tx_next <= idle;
      end if;

    when startbit => tx_next <= bit0;
    when bit0     => tx_next <= bit1;
    when bit1     => tx_next <= bit2;
    when bit2     => tx_next <= bit3;
    when bit3     => tx_next <= bit4;
    when bit4     => tx_next <= bit5;
    when bit5     => tx_next <= bit6;
    when bit6     => tx_next <= bit7;
    when bit7     => tx_next <= stopbit;
```

```vhdl

    when stopbit  =>
      -- Wait a few ns extra to ensure that the reciver is ready for a new byte
      half_period  <= (divisor * half) + delay;
      tx_next <= idle;

    when others =>
      tx_next <= idle;
  end case;
end process;

tx_start_register: process(rst, clk, tx_start, tx_current)
begin
  if (rst='0') then
    tx_start_reg <= '0';
  elsif (rising_edge(clk)) then
    if (tx_current=idle and tx_start='1') then
      tx_start_reg <= '1';
    elsif (tx_current=bit7) then
      tx_start_reg <= '0';
    end if;
  end if;
end process;

tx_data_ShiftRegister: process (rst, rs232clk, tx_load, tx_current, tx_data)
begin
  if (rst = '0') then
    tx_data_reg <= (others => '1');
  elsif (tx_load='1') then
    tx_data_reg(9) <= '1';
    tx_data_reg(8 downto 1) <= tx_data;
    tx_data_reg(0) <= '0';
  elsif (rising_edge(rs232clk)) then
    if (tx_current=idle and tx_start='0') then
      tx_data_reg(0) <= '1';
    elsif (tx_shift_en='1') then
      tx_data_reg <= '1' & tx_data_reg(9 downto 1);
    end if;
  end if;
end process;

rs232_stateRegister: process (rst, rs232clk, tx_next)
begin
  if (rst='0') then
    tx_current <= idle;
  elsif (rising_edge(rs232clk)) then
    tx_current <= tx_next;
  end if;
end process;

rs232clk_generator: process (clk, rst, half_period)
begin
  if (rst = '0') then
    cnt <= 1;
    rs232clk <= '1';
  elsif (rising_edge(clk)) then
    if (cnt = half_period) then
      rs232clk <= not rs232clk;
      cnt <= 1;
    else
      cnt <= cnt + 1;
    end if;

  end if;
end process;


end rs232tx;
```

devboard.vhd

```
--
--  File  : devboard.vhd
--  Author: Geir Nilsen { geirni@ifi.uio.no }
--  Created: Mars 3 2004
--
--  Description:
--    Test of components on Development Board. These components are external
--    to the FPGA. Use ids.c as a user interface to this design.
--

library ieee, unisim;
use ieee.std_logic_1164.all;
use unisim.vcomponents.all;
use work.components.all;

entity devboard is
  generic(
    longestPattern : integer := 240;---------- Fix: add this file to cam_vhdl.pl
    addrBits       : integer := 3
    );
  port(
    i_clk        : in  std_logic; -- 100 MHz
    rst          : in  std_logic;
    lcd_en       : out std_logic; -- LCD Enable Signal
    -- Goes to LCD-display. lcd_data_out(8): LCD Register Select Signal
    lcd_data_out : out std_logic_vector(8 downto 0);
    push         : in  std_logic_vector(1 to 3);
    dip          : in  std_logic_vector(8 downto 1);
    led          : out std_logic_vector(1 to 4);
    rxd1         : in  std_logic;    -- Data in 1 (COM1)
    txd1         : out std_logic;    -- Data/Instructions out (debug)
    rxd2         : in  std_logic     -- Instruction in (COM2)
--    txd2          : out std_logic      -- Data/Instructions out
  );
end devboard;

architecture devboard of devboard is
  signal clk                       : std_logic;
  signal clk_buf                   : std_logic;

  -- Data transfer through COM1
  signal rx_data1                  : std_logic_vector(7 downto 0);
  signal rx_data_rdy1              : std_logic; --_vector(1 to 2)

  signal tx_rdy1                   : std_logic; --_vector(1 to 2)
  signal tx_start1                 : std_logic; --_vector(1 to 2)
  signal tx_data_reg1              : std_logic_vector(7 downto 0);

  -- Instructions through COM2
  signal rx_data2                  : std_logic_vector(7 downto 0);
  signal rx_data_reg2              : std_logic_vector(7 downto 0);
```

```vhdl
  signal rx_data_reg2_res         : std_logic;
  signal rx_data_rdy2             : std_logic;

--  signal tx_rdy2                  : std_logic;
--  signal tx_start2                : std_logic;
--  signal tx_data_reg2             : std_logic_vector(7 downto 0);

  signal lcd_data_in_shiftreg     : std_logic_vector(8 downto 0);
  signal lcd_start_sig            : std_logic;
  signal lcd_start_reg            : std_logic;
  signal lcd_rdy                  : std_logic;

  -- Wires to pushButtonRegisters
  signal pB_wire                  : std_logic_vector(1 to 2);
  -- pushButtonRegisters
  signal pB_reg                   : std_logic_vector(1 to 2);
  signal dip_reg                  : std_logic_vector(7 downto 0);

  type main_FSM_type is ( idle, push1_st, push_st, lcd_start_st);
  signal main_curr, main_next : main_FSM_type;

  signal led4_reg : std_logic;
begin

--------------------------------------------------------------------------------
-- FSM

main_FSM: process(main_curr, pB_reg, push(3), rx_data_reg2, lcd_rdy, tx_rdy1)
begin

  lcd_start_sig    <= '0';
  rx_data_reg2_res <= '0';
  tx_start1        <= '0';
--  tx_start2        <= '0';

  case main_curr is

    when idle =>
      if (pB_reg(1)='0') then
        main_next <= push1_st;
      elsif ((pB_reg(2)='0' or push(3)='0') and lcd_rdy='1' and tx_rdy1='1') then
         main_next <= push_st;
      elsif (rx_data_reg2="00000010") then  -- Instruction Register: LCD start
        main_next <= lcd_start_st;
      else
        main_next <= idle;
      end if;

    when push1_st =>
      lcd_start_sig <= '1';
      main_next     <= idle;

    when push_st =>
      lcd_start_sig <= '1';  -- Display byte from DIP on LCD
      tx_start1     <= '1';  -- Send byte to PC
      main_next     <= idle;

    when lcd_start_st =>
      lcd_start_sig    <= '1';
      rx_data_reg2_res <= '1'; -- Clear Instruction Register
      main_next        <= idle;
    when others => null;
  end case;
end process;



--
-- Instruction Register In, COM2
-- 0 = NOP                  - Idle/Stop/Break
-- 1 = LCD data shift left  - Shift enable. lcd_data_in_shiftregister
-- 2 = LCD start            - Set lcd_start_reg high (Self-clearing)
-- 4 = led4on
-- 3 = led4off
--
rx_data2_register: process(rst, clk, rx_data_reg2_res, rx_data_rdy2)
begin
  if (rst='0' or rx_data_reg2_res='1') then
    rx_data_reg2 <= (others => '0');
  elsif(rising_edge(clk)) then
    if (rx_data_rdy2='1') then
      rx_data_reg2 <= rx_data2;
    end if;
  end if;
end process;

-- Instruction/Data Out, COM2
--tx_data_reg2_register: process(rst, clk)
--begin
--  if (rst='0') then
--    tx_data_reg2 <= (others => '0');
--  elsif (rising_edge(clk)) then
--    tx_data_reg2 <= "00000" & cam_wordaddr_out_reg;----------- max 255
--  end if;
--end process;

-- Debug Out, COM1
tx_data1_register: process(rst, clk)
begin
  if (rst='0') then
    tx_data_reg1 <= (others => '0');
  elsif (rising_edge(clk) and (pB_reg(2)='0' or push(3)='0')) then
    tx_data_reg1 <= dip;
  end if;
end process;

main_FSM_register: process(rst, clk, main_next)
begin
  if (rst='0') then
    main_curr <= idle;
  elsif (rising_edge(clk)) then
    main_curr <= main_next;
  end if;
end process;

push_register: process (clk, rst, pB_wire)
begin
  if (rst='0') then
    pB_reg <= (others => '1');
  elsif (rising_edge(clk)) then
    pB_reg <= pB_wire;
  end if;
end process;

dip_register: process (clk, rst, dip)
begin
  if (rst='0') then
    dip_reg <= (others => '1');
  elsif (rising_edge(clk)) then
    dip_reg <= dip;
  end if;
end process;

lcd_data_in_shiftregister: process(rst, clk, rx_data_rdy1, rx_data_reg2)
begin
  if (rst='0') then
    lcd_data_in_shiftreg <= (others => '0');
  elsif (rising_edge(clk)) then
    if (rx_data_rdy1='1' and rx_data_reg2="00000001") then
      lcd_data_in_shiftreg <= lcd_data_in_shiftreg(0) & rx_data1;
    elsif (pB_reg(1)='0') then
      lcd_data_in_shiftreg(8 downto 0) <= '0' & dip_reg;
    elsif (pB_reg(2)='0' or push(3)='0') then
      lcd_data_in_shiftreg(8 downto 0) <= '1' & dip_reg;
    end if;
  end if;
end process;

lcd_start_register: process(rst, clk, lcd_start_sig)
begin
  if (rst='0') then
    lcd_start_reg <= '0';
  elsif (rising_edge(clk)) then
    if (lcd_start_sig='1') then
      lcd_start_reg <= '1';
    else
      lcd_start_reg <= '0';
    end if;
  end if;
end process;

-- Assign debouncer to pushButtons 1 and 2
pButton_inst_X: for i in 1 to 2 generate
begin
  pButton_inst: pButton
    port map (
      clk        => clk,
      rst        => rst,
      pButtonIn  => push(i),
      pButtonOut => pB_wire(i)
      );
end generate;


clk_in: IBUFG
  port map (
    I => i_clk,
    O => clk_buf
    );

clk_out: BUFG
  port map (
    I => clk_buf,
    O => clk
    );

lcd_inst: lcd
  port map (
    clk          => clk,
    rst          => rst,
    lcd_data_in  => lcd_data_in_shiftreg,
    lcd_data_out => lcd_data_out,
    lcd_en       => lcd_en,
    lcd_rdy      => lcd_rdy,
    lcd_start    => lcd_start_reg
    );

rs232rx_inst_dataIn: rs232rx
--  generic map(divisor => 1, half => 2) -- Simulation
  port map (
    clk         => clk,
    rst         => rst,
    rxd         => rxd1,
    rx_data     => rx_data1,
    rx_data_rdy => rx_data_rdy1
    );

rs232tx_inst_debugOut: rs232tx
--  generic map(divisor => 1, half => 1) -- Simulation
  port map (
    clk      => clk,
    rst      => rst,
    tx_rdy   => tx_rdy1,
    tx_start => tx_start1,
    tx_data  => tx_data_reg1,
    txd      => txd1
    );

rs232rx_inst_instructionIn: rs232rx
--  generic map(divisor => 1, half => 2) -- Simulation
  port map (
    clk         => clk,
    rst         => rst,
    rxd         => rxd2,
    rx_data     => rx_data2,
    rx_data_rdy => rx_data_rdy2
    );

--rs232tx_inst_out: rs232tx
--  generic map(divisor => 1, half => 1) -- Simulation
--  port map (
--    clk      => clk,
--    rst      => rst,
--    tx_rdy   => tx_rdy2,
--    tx_start => tx_start2,
--    tx_data  => tx_data_reg2,
--    txd      => txd2
--    );

led_flash_inst_dataIn: led_flash
  port map (
    clk     => clk,
    rst     => rst,
    sig_in  => rx_data_rdy1,
    led_reg => led(1)
    );

led_flash_inst_dataOut: led_flash
  port map (
    clk     => clk,
    rst     => rst,
    sig_in  => tx_start1,
    led_reg => led(2)
    );

led_flash_inst_instructionIn: led_flash
  port map (
    clk     => clk,
    rst     => rst,
    sig_in  => rx_data_rdy2,
    led_reg => led(3)
    );

--led_flash_inst_instructionOut: led_flash
--  port map (
--    clk     => clk,
--    rst     => rst,
--    sig_in  => tx_start2,
--    led_reg => led(4)
--    );
--led(4) <= '1';

led(4) <= led4_reg;

led4_register: process(rst, clk, rx_data_rdy2, rx_data_reg2)
begin
  if (rst='0') then
    led4_reg <= '1';
  elsif (rising_edge(clk) and rx_data_rdy2='1') then
    if (rx_data2="00000011") then   -- On
      led4_reg <= '0';
    elsif (rx_data2="00000100") then  -- Off
      led4_reg <= '1';
    end if;
  end if;
end process;


end devboard;
```

ids128.vhd

```
--
--  File   : ids128.vhd
--  Author : Geir Nilsen { geirni@ifi.uio.no }
--  Created: Mars 3 2004
--
--  Description:
--    Top level module of a cam of 128 words. The cam-files must be configured
--    to 128 words. To change this file to match other cam's a change must be
--    made in the tx_data2_register. The current version of this design can
--    take at most 128 words. Use ids.c for a user interface to this design.
--

```

```
library ieee, unisim;
use ieee.std_logic_1164.all;
use unisim.vcomponents.all;
use work.components.all;
use work.cam_components.all;

entity ids is
  generic(
    longestPattern : integer := 256;---------- Fix: add this file to cam_vhdl.pl
    addrBits       : integer := 7
    );
  port(
    i_clk        : in  std_logic; -- 100 MHz
    rst          : in  std_logic;
    led4         : out std_logic;
    rxd1         : in  std_logic;
    rxd2         : in  std_logic; -- Instruction in (COM2)
    txd2         : out std_logic  -- Data/Instructions out
  );
end ids;

architecture ids of ids is
  signal clk                      : std_logic;
  signal clk_buf                  : std_logic;

  -- Data transfer through COM1
  signal rx_data1                 : std_logic_vector(7 downto 0);
  signal rx_data_rdy1             : std_logic;

  -- Instructions through COM2
  signal rx_data2                 : std_logic_vector(7 downto 0);
  signal rx_data_reg2             : std_logic_vector(7 downto 0);
  signal rx_data_rdy2             : std_logic;

  signal tx_rdy2                  : std_logic;
  signal tx_start2                : std_logic;
  signal tx_data_reg2             : std_logic_vector(15 downto 0);
  signal tx_data2                 : std_logic_vector( 7 downto 0);
  signal tx2_start_sig            : std_logic;
  signal tx2_send_sig             : std_logic;

  signal rx_data_reg2_res         : std_logic;

  signal cam_data_shiftreg        : std_logic_vector(longestPattern-1 downto 0);
  signal cam_data_shift_en        : std_logic;
  signal cam_wordaddr_in_shiftreg : std_logic_vector(addrbits-1 downto 0); --NB!
  signal cam_addr_shift_en        : std_logic;
  signal cam_wordaddr_out         : std_logic_vector(addrbits-1 downto 0);
  signal cam_write_rdy            : std_logic;
  signal cam_write_en_sig         : std_logic;
  signal cam_write_en_reg         : std_logic;
  signal cam_match_en_reg         : std_logic;
  signal cam_match_en_sig         : std_logic;
  signal cam_match                : std_logic;

  type write_FSM_type is (idle, write_st);
  signal write_curr, write_next : write_FSM_type;
  type cam_match_FSM_type is (match_off_st, match_on_st, wait_st1, wait_st2);
  signal match_curr, match_next : cam_match_FSM_type;
  type data_FSM_type is (shift_off, shift_on);
  signal data_curr, data_next : data_FSM_type;
```

```vhdl
  type addr_FSM_type is (shift_off, shift_on);
  signal addr_curr, addr_next : addr_FSM_type;
  type tx2_FSM_type is (idle, byte1, byte2);
  signal tx2_curr, tx2_next : tx2_FSM_type;

  signal tx2_start_reg : std_logic;
begin

--------------------------------------------------------------------------------


tx2_start_register: process(rst, clk, tx2_start_sig)
begin
  if (rst='0') then
    tx2_start_reg <= '0';
  elsif (rising_edge(clk)) then
    if (tx2_start_sig='1') then
      tx2_start_reg <= '1';
    else
      tx2_start_reg <= '0';
    end if;
  end if;
end process;

tx_addr_FSM: process(tx2_curr, tx2_start_reg, tx_rdy2)  -- Send 2 bytes to PC
begin
  tx_start2 <= '0';       -- tx_data2 <= tx_data_reg2( 7 downto 0)
  tx2_send_sig <= '0';
  case tx2_curr is

    when idle =>
      if (tx2_start_reg='1') then
        tx2_next <= byte1;
      else
        tx2_next <= idle;
      end if;

    when byte1 =>
      if (tx_rdy2='1') then
        tx_start2 <= '1';
        tx2_next <= byte2;
      else
        tx2_next <= byte1;
      end if;

    when byte2 =>
      tx2_send_sig <= '1';  -- tx_data2 <= tx_data_reg2(15 downto 0)
      if (tx_rdy2='1') then
        tx_start2 <= '1';
        tx2_next <= idle;
      else
        tx2_next <= byte2;
      end if;

    when others => null;
  end case;
end process;

tx_data2 <=
  tx_data_reg2( 7 downto 0) when tx2_send_sig='0' else
  tx_data_reg2(15 downto 8) when tx2_send_sig='1';
```

```

tx_data2_register: process(rst, clk)
begin
  if (rst='0') then
    tx_data_reg2 <= (others => '0');
  elsif (rising_edge(clk) and cam_match='1') then
    --tx_data_reg2 <= "0000000000000" & cam_wordaddr_out; -- Max    8 words
    --tx_data_reg2 <= "000000000000"  & cam_wordaddr_out; -- Max   16 words
    --tx_data_reg2 <= "00000000000"   & cam_wordaddr_out; -- Max   32 words
    --tx_data_reg2 <= "0000000000"    & cam_wordaddr_out; -- Max   64 words
      tx_data_reg2 <= "000000000"     & cam_wordaddr_out; -- Max  128 words
    --tx_data_reg2 <= "00000000"      & cam_wordaddr_out; -- Max  256 words
    --tx_data_reg2 <= "0000000"       & cam_wordaddr_out; -- Max  512 words
    --tx_data_reg2 <= "000000"        & cam_wordaddr_out; -- Max 1024 words
    --tx_data_reg2 <= "00000"         & cam_wordaddr_out; -- Max 2048 words
  end if;
end process;

cam_match_FSM: process(match_curr, rx_data_reg2, cam_match, tx_rdy2,
                       rx_data_rdy1)
begin

  tx2_start_sig <= '0';
  cam_match_en_sig <= '0';

  case match_curr is

    when match_off_st =>
      if (rx_data_reg2="00000110") then
        match_next <= match_on_st;
      else
        match_next <= match_off_st;
      end if;

    when match_on_st =>
      cam_match_en_sig <= '1';
      if (rx_data_reg2="00000111") then
        match_next <= match_off_st;
      elsif (cam_match='1' and tx_rdy2='1') then
        tx2_start_sig <= '1';
        match_next <= wait_st1;
      else
        match_next <= match_on_st;
      end if;

    when wait_st1 =>
      if (rx_data_reg2="00000111") then
        match_next <= match_off_st;
      elsif (rx_data_rdy1='1') then
        match_next <= wait_st2;
      else
        match_next <= wait_st1;
      end if;

    when wait_st2 =>
      cam_match_en_sig <= '1';
      if (rx_data_reg2="00000111") then
        match_next <= match_off_st;
      else
        match_next <= match_on_st;
      end if;

    when others => null;
  end case;
end process;

cam_data_shift_FSM: process(data_curr, rx_data_reg2)
begin
  cam_data_shift_en <= '0';
  case data_curr is
    when shift_off =>
      if (rx_data_reg2="00000011") then
        data_next <= shift_on;
      else
        data_next <= shift_off;
      end if;
    when shift_on =>
      cam_data_shift_en <= '1';
      if (rx_data_reg2="00000111") then
        data_next <= shift_off;
      else
        data_next <= shift_on;
      end if;
    when others => null;
  end case;
end process;

cam_write_FSM: process(write_curr, rx_data_reg2, cam_write_rdy, cam_match,
                       tx_rdy2)
begin
  rx_data_reg2_res <= '0';
  cam_write_en_sig <= '0';
  case write_curr is
    when idle =>
      if (rx_data_reg2="00000101") then
        write_next <= write_st;
      else
        write_next <= idle;
      end if;
    when write_st =>
      if (cam_write_rdy='1') then
        cam_write_en_sig <= '1';
        rx_data_reg2_res <= '1'; -- Clear Instruction Register
      end if;
      write_next <= idle;
    when others => null;
  end case;
end process;

cam_addr_shift_FSM: process(addr_curr, rx_data_reg2)
begin
  cam_addr_shift_en <= '0';
  case addr_curr is
    when shift_off =>
      if (rx_data_reg2="00000100") then
        addr_next <= shift_on;
      else
        addr_next <= shift_off;
      end if;
    when shift_on =>
      cam_addr_shift_en <= '1';
      if (rx_data_reg2="00000001") then
```

```
        addr_next <= shift_off;
      else
        addr_next <= shift_on;
      end if;
    when others => null;
  end case;
end process;

tx_addr_FSM_StateRegister: process(rst, clk)
begin
  if (rst='0') then
    tx2_curr <= idle;
  elsif (rising_edge(clk)) then
    tx2_curr <= tx2_next;
  end if;
end process;

cam_addr_shift_FSM_StateRegister: process(rst, clk, addr_next)
begin
  if (rst='0') then
    addr_curr <= shift_off;
  elsif (rising_edge(clk)) then
    addr_curr <= addr_next;
  end if;
end process;

cam_data_shift_FSM_StateRegister: process(rst, clk, data_next)
begin
  if (rst='0') then
    data_curr <= shift_off;
  elsif (rising_edge(clk)) then
    data_curr <= data_next;
  end if;
end process;

cam_match_FSM_StateRegister: process(rst, clk, match_next)
begin
  if (rst='0') then
    match_curr <= match_off_st;
  elsif (rising_edge(clk)) then
    match_curr <= match_next;
  end if;
end process;

cam_write_FSM_StateRegister: process(rst, clk, write_next)
begin
  if (rst='0') then
    write_curr <= idle;
  elsif (rising_edge(clk)) then
    write_curr <= write_next;
  end if;
end process;
--------------------------------------------------------------------------------
-- Instruction Register In, COM2
--
-- 3 = CAM data shift LSB   - ShiftEnableOn. cam_data_shiftreg
-- 8                        - ShiftEnableOff
-- 4 = CAM addr             - Shift enable on.  Set cam_wordaddr_in
-- 1                        - Shift enable off
-- 5 = CAM write            - Set cam_write_en_reg high (Self-clearing)
-- 6 = CAM match on         - Shift enable cam_data_reg. Recv cam_data and macth
-- 7 = CAM match off

cam_data_shiftregister: process(rst, rx_data_rdy1, cam_data_shift_en)
begin
  if (rst='0') then
    cam_data_shiftreg <= (others => '0');
  elsif (rising_edge(rx_data_rdy1)) then  --------------------------------------
    if (cam_data_shift_en='1') then
      cam_data_shiftreg <=
        cam_data_shiftreg((longestPattern-1)-8 downto 0) & rx_data1;
    end if;
  end if;
end process;

cam_wordaddr_in_shiftregister: process(rst, clk, rx_data_rdy1, cam_addr_shift_en)
begin
  if (rst='0') then
    cam_wordaddr_in_shiftreg <= (others => '0');
  elsif (rising_edge(rx_data_rdy1)) then   -------------------------------------
    if (cam_addr_shift_en='1') then
      -- For use with CAM that has words > 256:
      --cam_wordaddr_in_shiftreg <= cam_wordaddr_in_shiftreg(0) & rx_data1;-----
      -- For use with CAM that has words <= 256:
      cam_wordaddr_in_shiftreg <= rx_data1(addrbits-1 downto 0);
    end if;
  end if;
end process;

cam_write_en_register: process(rst, clk, cam_write_en_sig)
begin
  if (rst='0') then
    cam_write_en_reg <= '0';
  elsif (rising_edge(clk)) then
    if (cam_write_en_sig='1') then
      cam_write_en_reg <= '1';
    else
      cam_write_en_reg <= '0';
    end if;
  end if;
end process;

cam_match_en_register: process(rst, clk, rx_data_reg2)
begin
  if (rst='0') then
    cam_match_en_reg <= '0';
  elsif (rising_edge(clk)) then
    if (cam_match_en_sig='1') then
      cam_match_en_reg <= '1';
    else
      cam_match_en_reg <= '0';
    end if;
  end if;
end process;

rx_data2_register: process(rst, clk, rx_data_reg2_res, rx_data_rdy2)
begin
  if (rst='0' or rx_data_reg2_res='1') then
    rx_data_reg2 <= (others => '0');
  elsif(rising_edge(clk)) then
    if (rx_data_rdy2='1') then
      rx_data_reg2 <= rx_data2;
    end if;
  end if;
end process;

--------------------------------------------------------------------------------
-- Instanitiations

cam_top_inst: cam_top
  port map (
    clk              => clk,
    rst              => rst,
    cam_data         => cam_data_shiftreg,
    cam_wordaddr_in  => cam_wordaddr_in_shiftreg,
    cam_wordaddr_out => cam_wordaddr_out,
    cam_write_rdy    => cam_write_rdy,
    cam_write_en     => cam_write_en_reg,
    cam_match_en     => cam_match_en_reg,
    cam_match        => cam_match
  );

clk_in: IBUFG
  port map (
    I => i_clk,
    O => clk_buf
    );

clk_out: BUFG
  port map (
    I => clk_buf,
    O => clk
    );

rs232rx_inst_dataIn: rs232rx
  --generic map(divisor => 1, half => 2)  -- Simulation
  port map (
    clk         => clk,
    rst         => rst,
    rxd         => rxd1,
    rx_data     => rx_data1,
    rx_data_rdy => rx_data_rdy1
    );

rs232rx_inst_instructionIn: rs232rx
  --generic map(divisor => 1, half => 2)  -- Simulation
  port map (
    clk         => clk,
    rst         => rst,
    rxd         => rxd2,
    rx_data     => rx_data2,
    rx_data_rdy => rx_data_rdy2
    );

rs232tx_inst_out: rs232tx
  --generic map(divisor => 1, half => 1)  -- Simulation
  port map (
    clk      => clk,
    rst      => rst,
    tx_rdy   => tx_rdy2,
    tx_start => tx_start2,
    tx_data  => tx_data2,
    txd      => txd2
    );

led_flash_inst_instructionOut: led_flash
  port map (
    clk     => clk,
    rst     => rst,
    sig_in  => tx_start2,
    led_reg => led4
    );

end ids;
```

components.vhd

```
--
--  File  : components.vhd
--  Author: Geir Nilsen { geirni@ifi.uio.no }
--
--  Created: Mars 3 2004
--

library ieee;
use ieee.std_logic_1164.all;

package components is

component lcd is
  generic(
    -- Assuming 100 MHz clock

    -- Timing parameters for write operation
    tcycE : integer := 50;  -- Enable cycle time                 (min 500 ns)
    PWEH  : integer := 23;  -- Enable pulse width (high level)   (min 230 ns)
    tEr   : integer :=  1;  -- Time rise                         (max  20 ns)
    tEf   : integer :=  1;  -- Time fall                         (max  20 ns)
    tAS   : integer :=  4;  -- Address setup time (RS, R/W to E) (min  40 ns)
    tAH   : integer :=  1;  -- Address hold time                 (min  10 ns)
    tDSW  : integer :=  8;  -- Data set-up time                  (min  80 ns)
    tH    : integer :=  1;  -- Data hold time                    (min  10 ns)

    -- The instruction delay needs an n+1 bit counter
    upperDelayIndex : integer := 20  -- All bits = '1' => 20.9 ms
    );
  port(
    clk          : in  std_logic;
    rst          : in  std_logic;
    -- lcd_data(8) = RS (register select)
    lcd_data_in  : in  std_logic_vector(8 downto 0);
    lcd_data_out : out std_logic_vector(8 downto 0);
    lcd_en       : out std_logic;
    lcd_rdy      : out std_logic;
    lcd_start    : in  std_logic
    );
end component;

component pButton is
  generic(
    delay : integer := 1000000 -- 0.01s at 100 MHz
    );
  port(
    clk        : in   std_logic;
    rst         : in   std_logic;
    pButtonIn   : in   std_logic;
    pButtonOut  : out  std_logic
    );
end component;

component rs232rx is
  generic(
    divisor : integer :=   1;
    half    : integer := 434
    );
  port(
    clk         : in  std_logic;                    -- 100 MHz
    rst         : in  std_logic;
    rxd         : in  std_logic;                    -- Bit to FPGA from DP9
    rx_data     : out std_logic_vector(7 downto 0); -- Bits from rxd
    rx_data_rdy : out std_logic                     -- '1' when rx_data is ready
    );
end component;

component rs232tx is
  generic(
    divisor : integer := 1;
    half    : integer := 434
    );
  port(
    clk       : in  std_logic;                    -- 100 MHz
    rst       : in  std_logic;
    tx_rdy    : out std_logic;                    -- Ready to send data to DP9
    tx_start  : in  std_logic;                    -- Start sendig tx_data
    tx_data   : in  std_logic_vector(7 downto 0); -- Byte to write to DP9
                                                  -- (from FPGA) using txd
    txd       : out std_logic                     -- Bit from FPGA to DP9
    );
end component;

component led_flash is
  generic(
    litetime : integer := 1000000       -- 1/100 s @ 100 MHz
    );
  port(
    clk     : in  std_logic;
    rst     : in  std_logic;
    sig_in  : in  std_logic;
    led_reg : out std_logic
    );
end component;

end components;
```

devboard.ucf

```
#
# User defined constraints
#
# Author: Geir Nilsen { geirni@ifi.uio.no }
#
# Created: Oct  7 2003
#


```

```
  #######################################################
  #    ##########                              #########
#####  ##########      Developement Board      ########
 ###   ##########                              #########
  #    ##################################################

# clk period
net "i_clk" period = 10.00;

# CLK.CAN.HS / Input / On-board 100 MHz LVTTL Oscillator
net "i_clk" loc = "V12";

# User LED
net "led<1>" loc = "V8" ; # DS7  / LED1
net "led<2>" loc = "W6" ; # DS8  / LED2
net "led<3>" loc = "U10"; # DS9  / LED3
net "led<4>" loc = "V10"; # DS10 / LED4

# User Push Button Switches
net "rst"     loc = "V15" ;                       # SW3 / FPGA.RESET
net "push<1>" loc = "V7"  ; net "push<1>" pullup; # SW4 / PUSH1
net "push<2>" loc = "W5"  ; net "push<2>" pullup; # SW5 / PUSH2
net "push<3>" loc = "AA12"; net "push<3>" pullup; # SW6 / PUSH3

# User DIP switch
# NOTE: Reversed to make DIP(8) LSB
net "dip<8>" loc = "W13"; net "dip<8>" pullup; # DIP8 / User Switch 7 - Input 8
net "dip<7>" loc = "Y13"; net "dip<7>" pullup; # DIP7 / User Switch 7 - Input 7
net "dip<6>" loc = "W14"; net "dip<6>" pullup; # DIP6 / User Switch 7 - Input 6
net "dip<5>" loc = "W15"; net "dip<5>" pullup; # DIP5 / User Switch 7 - Input 5
net "dip<4>" loc = "Y15"; net "dip<4>" pullup; # DIP4 / User Switch 7 - Input 4
net "dip<3>" loc = "W16"; net "dip<3>" pullup; # DIP3 / User Switch 7 - Input 3
net "dip<2>" loc = "Y16"; net "dip<2>" pullup; # DIP2 / User Switch 7 - Input 2
net "dip<1>" loc = "V16"; net "dip<1>" pullup; # DIP1 / User Switch 7 - Input 1

# LCD Interface
# NOTE: lcd_data(8) is used as RS
net "lcd_data_out<0>" loc = "D7"; # LCD data bit 0
net "lcd_data_out<1>" loc = "F9"; # LCD data bit 1
net "lcd_data_out<2>" loc = "D5"; # LCD data bit 2
net "lcd_data_out<3>" loc = "D6"; # LCD data bit 3
net "lcd_data_out<4>" loc = "C7"; # LCD data bit 4
net "lcd_data_out<5>" loc = "D8"; # LCD data bit 5
net "lcd_data_out<6>" loc = "C8"; # LCD data bit 6
net "lcd_data_out<7>" loc = "E8"; # LCD data bit 7
net "lcd_data_out<8>" loc = "E6"; # LCD Register Select (RS)
net "lcd_en"          loc = "E7"; # LCD Enable Signal

# RS232
# The names of these two pins have been switched
#net "txd2" loc = "W7"; # Bit from DP9   to FPGA
net "rxd2" loc = "U9"; # Bit from FPGA to DP9

       ##################################################    #
        #########                              ##########   ###
         ########      Developement Board      ##########  #####
        #########                              ##########    #
       #######################################################
  #######################################################
  #    ##########                             #########
#####  ##########  P160 Communications Module  ########
 ###   ##########                             #########
  #    ##################################################

# RS232
net "txd1" loc = "E14"; # RS232_TX / JX1 Pin B12
net "rxd1" loc = "F14"; # RS232_RX / JX1 Pin B13

       ##################################################    #
        #########                             ##########   ###
         ########  P160 Communications Module  ##########  #####
        #########                             ##########    #
       #######################################################
```

F.4 Source2html Converters

The source files in this report have been converted to html and then imported into Word without loosing the formatting.

- src2html.css
  - This is the file that defines the formatting of the html files. It is common to all the files in this Appendix.
- vhdl2html
  - Originally found on the web Major modifications have been made. It now handles *ucf* files also. Takes one file at a time.
- src2html
  - A script that converts C, Perl, VHDL and ucf files to html by invoking vhdl2html and Source-Highlight. Takes one file at a time.
- to_html
  - This is an example of how to convert many files at once.

Source-Highlight is found at:
http://www.gnu.org/software/src-highlite/

src2html.css

```
body { color:#000; background-color:#fff; }

pre .linenum   { color:#000; font-size:0.6em; vertical-align:middle; }
pre .blankLine { color:#fff; } /* For compatibility with MS_Word */
pre .comment   { color:#080; }
pre .keyword   { color:#00f; font-weight:bold; }
pre .type      { color:#00f; }
pre .function  { color:#800; font-weight:bold; }
pre .string    { color:#800; }
pre .number    { color:#000; }
pre .preproc   { color:#008; font-weight:bold; }
pre .normal    { color:#000; }
pre .symbol    { color:#000; } /* .,"#% ... */
pre .cbracket  { color:#000; } /* Curly brackets */
pre .attribute { color:#088; }
pre .pack      { color:#000; font-weight:bold; }
```

vhdl2html

```
#!/usr/bin/perl

# This script converts a VHDL file to HTML with all VHDL keywords
# in bold face type.
#
# Rex Hill
# Applied Research Laboratory
# Washington University in St. Louis
# rex@arl.wustl.edu
#
# Originally found at http://www.veritools-web.com/vhdl2html.htm

#
# Modified for css use and more syntax categories by
# Benj Carson <benjcarson@digitaljunkies.ca>
#
# Note:  style.css must now be stored in the same directory
# as this program's output.
#

#
# This file was found at:
# http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2002_w/misc/vhdl2html/
#
# Major changes have been made to this file.
#
# Modified to handle comments properly.
# Code added to handle quotes. Does not format words in quotes now.
# Trailing blanks are not written to html.
# Switches are added.
# Added support for ucf file format.
#
# A link to "style.css" is added, so that it dosen't have to be in
# the same directory as the HTML-file(s).
#
# Deleted files "vhdl.*". Words to be substituted are now included in this file.
# The substitutions of words are now case insensitive.
# All keywords etc are converted to lowercase.
#
# Geir Nilsen <geirni@ifi.uio.no>
#



# Print usage and exit
sub msg {
    print
        "\n".
        "    Example usage with aquired switches:\n".
        "      \"vhdl2html.pl -s vhdl -i counter.vhd -o counter.vhd.html\n".
        "    Switches:\n".
        "      -n  Write line numbers. Default is no line numbers\n".
        "      -s  Specify source type; vhdl or ucf\n".
        "      -i  Input file\n".
        "      -o  Output file\n".
        "      -T  Text between html title tags\n".
        "      -c  Path to css file. Default:\n".
        "          http://heim.ifi.uio.no/~geirni/src2html.css\n".
        "\n";
    exit(-1);
}

# ucf: Words to format. Lot's of TODO stuff here...
%subst_ucf =
  (
   keyword =>
   ["loc","net","period","pullup","pulldown"]
   );

# vhdl: Words to format
%subst_vhdl =
  (
   keyword =>
   ["abs","access","after","alias","all","and","arch","architecture","array",
    "assert","attr","attribute","begin","block","body","buffer","bus","case",
    "comp","component","cond","conditional","conf","configuration","cons",
    "constant","disconnect","downto","else","elseif","elsif","end","entity",
    "exit","file","for","func","function","generate","generic","group",
    "guarded","if","impure","in","inertial","inout","inst","instance","is",
    "label","library","linkage","literal","loop","map","mod","nand","new",
    "next","nor","not","null","of","on","open","or","others","out","pack",
    "package","port","postponed","procedure","process","pure","range","record",
    "register","reject","rem","report","return","rol","ror","select","severity",
    "shared","sig","signal","sla","sll","sra","srl","subtype","then","to",
    "transport","type","unaffected","units","until","use","var","variable",
    "wait","when","while","with","xnor","xor"],
   attribute =>
   ["active","ascending","base","delayed","driving","driving_value","event",
    "foreign","high","image","instance_name","last_active","last_event",
    "last_value","left","leftof","length","low","path_name","pos","pred",
    "quiet","reverse_range","right","rightof","simple_name","stable","succ",
    "transaction","val","value"],
   pack =>
   ["ieee","math_complex","math_real","numeric_bit","numeric_std",
    "std_logic_1164","std_logic_arith","std_logic_misc","std_logic_signed",
    "std_logic_textio","std_logic_unsigned","textio","unisim","vcomponents"],
   type =>
   ["bit","bit_vector","boolean","character","delay_length","file_open_kind",
    "file_open_status","integer","line","natural","positive","real",
    "severity_level","side","signed","std_logic","std_logic_vector",
    "std_ulogic","std_ulogic_vector","string","text","time","unsigned"],
   function =>
   ["conv_integer","conv_signed","conv_std_logic_vector","conv_unsigned",
    "endfile","ext","falling_edge","is_x","now","read","readline","resize",
    "resolved","rising_edge","rotate_left","rotate_right","shift_left",
    "shift_right","shl","shr","std_match","sxt","to_01","to_bit","to_bitvector",
    "to_integer","to_signed","to_stdlogicvector","to_stdulogic",
    "to_stdulogicvector","to_unsigned","to_ux01","to_x01","to_x01z","write",
    "writeline"]
   );


# Init variables to store arguments given at the command line
$num      = 0;  # 1 when "-n" is given at the command line
$source   = "";
$infile   = "";
$outfile  = "";
$title    = "";
$cssPath  = "http://heim.ifi.uio.no/~geirni/src2html.css";

```

```
# Line numbers will start on 1
$lineNum  = 1;

# Parse ARGV
while(@ARGV) {
    $option = shift;
    if($option eq "-i") {
        $infile = shift;
    }
    elsif($option eq "-o") {
        $outfile = shift;
    }
    elsif($option eq "-n") {
        $num  = 1;
    }
    elsif($option eq "-s") {
        $source  = shift;
    }
    elsif($option eq "-c") {
        $cssPath = shift;
    }
    elsif($option eq "-T") {
        $title  = shift;
    }
    else {
        msg();
    }
}

# Check parameters and determine type of conversion
if($source eq "" | $infile eq "" | $outfile eq ""){
    msg();
}
elsif($source eq "vhdl"){
    %subst = %subst_vhdl;
    $commentType = "--";
}
elsif($source eq "ucf"){
    %subst = %subst_ucf;
    $commentType = "#";      # html escape code for '#'
}
else {
    msg();
}

# Read file that is to be converted
open(SOURCEFILE, "<".$infile) or die
    "  Can't open file for read: ".$infile."\n";
@lines = <SOURCEFILE>;
close(SOURCEFILE);

# Open conversion file for write
open(HTMLFILE, ">".$outfile) or die
    "  Can't open file for write: ".$outfile."\n";

# Print out the HTML header information
print HTMLFILE
    "<html>\n".
    "  <head>\n".
    "    <title>".$title."</title>\n".
    "    <link rel=\"stylesheet\" type=\"text/css\" href=\"".$cssPath."\">\n".
    "  </head>\n".
    "  <body>\n".
    "\n".
    "<pre><tt>"; # No newline here



# Conversion:
for $line(@lines){

    # Global substitutions.
    # 1) Bugfix for ucf:
    #    "<1>" will be invisible when importing the html document into MS-Word.
    #    Fix: Change "<1>" to "&lt;1&gt;"
    # 2) Quotes - " - are messing up colors in Emacs
    $line =~ s/&/&amp;/g;   # &
    $line =~ s/</&lt;/g;    # <
    $line =~ s/>/&gt;/g;    # >
    $line =~ s/\"/&quot;/g; # "

    # Print line numbers
    if($num){
        print HTMLFILE
            "<span class=\"linenum\">".
            "0" x (length($#lines+1)-length($lineNum)).
            $lineNum++."</span> ";
        }

    # Remove newline temporarily; trailing blanks permanently
    $line =~ s/\n//;
    $line =~ s/\s+$//;

    # Add an empty comment to an empty line
    # (for equal line spacing when importing the html-document into MS-Word)
    if($line eq ""){
        print HTMLFILE "<span class=\"blankLine\">".$commentType."</span>";
    }

    # Handle comments
    $comment = "";
    if($line =~ /$commentType/){
        ($line, $comment) = split(/$commentType/, $line, 2);
        $comment = "<span class=\"comment\">".$commentType.$comment."</span>";
    }

    # Restore newline
    $comment .= "\n";

    # Handle quotes and conversion
    while($line ne ""){

        # There may be many quotes for each line
        $quote = "";
        if($line =~ /&quot;/){
            ($line, $quote, $rest) = split(/&quot;/, $line, 3);
            $quote = "<span class=\"symbol\">&quot;</span>".$quote.
                     "<span class=\"symbol\">&quot;</span>";
        }
        else{
            $rest = "";
        }

        # Conversion
        $line =~ s/(\'|\.|:|,|;|\+|\-|\*|=|\/|\\|\{|\}|\[|\]|\(|\)|&(amp|quot|lt|gt);)/<span class=\"symbol\">$1<\/span>/g;
        for $key(keys %subst){
            for($i=0; $i <= $#{$subst{$key}}; $i++){
                $line =~ s/\b($subst{$key}[$i])\b/<span class=\"$key\">\L$1<\/span>/gi;
            }
        }

        print HTMLFILE $line.$quote;
        $line = $rest;
    }

    print HTMLFILE $comment;
}



# Write end tags to HTML file
print HTMLFILE
    "</tt></pre>\n".
    "\n".
    "  </body>\n".
    "</html>\n";

close(HTMLFILE);
```

src2html

```
#!/local/bin/perl5 -w

#
# Created: July 20 2004
#
# Author:  Geir Nilsen <geirni@ifi.uio.no>
#
# Description:
#   See subroutine "msg" for usage and description
#



# Print help/usage
sub msg {
    print
        "\n".
        "  Uses Source-highlight to convert Perl and C to html. The converted\n".
        "  html-file is again converted for compatibility with css".
        "\n".
        "    Example usage (Perl, C) with aquired switches:\n".
        "      src2html -f html -s perl -i infile.pl -o outfile.pl.html\n".
        "  ----------\n".
        "  Uses vhdl2html to convert VHDL and ucf files to html".
        "\n".
        "    Example usage (vhdl, ucf) with aquired switches:\n".
        "      src2html -s vhdl -i counter.vhd -o counter.vhd.html\n".
        "  ----------\n".
        "  Switches:\n".
        "      -f       Source-highlight only: html, xhtml, esc\n".
        "      --no-doc Source-highlight only: Cancel -d switch\n".
        "\n".
        "      -n  Write line numbers. Default is no line numbers\n".
        "      -s  Specify source type; Perl, C, vhdl, ucf\n".
        "      -i  Input file\n".
        "      -o  Output file\n".
        "      -T  Text between html title tags\n".
        "      -c  Path to css file\n".
        "\n";
    exit(-1);
}

# Modify html-file (converted by source-highlight)
sub modify {
    open(HTMLFILE, "<".$outfile) or die
        "     Can't open file for read: ".$outfile."\n";
    @lines = <HTMLFILE>;
    close(HTMLFILE);

    open(HTMLFILE, ">".$outfile) or die
        "     Can't open file for write: ".$outfile."\n";

    if(!$nodoc_sw eq ""){ # Print out the HTML header information
        print HTMLFILE
            "<html>\n".
            "  <head>\n".
            "    <title>".$title."</title>\n".
            "    <link ".
            "rel=\"stylesheet\" type=\"text/css\" href=\"".$cssPath."\">\n".
            "  </head>\n".
            "  <body>\n".
            "\n";
    }

    # Do modifications
    for $line(@lines){

        # A character must be added to blank lines for compatibility with MS-Word
        # if size of linenumbers are smaller than the remaining text; for example
        # 0.6em. In Word, these lines will have hight 0.6 rather than 1.
        $linenumber = "";
        if($line =~ /^\d*:\s/){     # Example: "001: "
            $line =~ s/^(\d*):\s//; # Remove linenumber temporarily
            $linenumber = "<span class=\"linenum\">".$1."<\/span\> "; # "001 "
        }
        $line =~ s/\n//;   # Remove newline temporarily
        $line =~ s/\s+$//; # Remove trailing blanks permanently
        if($line eq ""){
            $line = "<span class=\"blankLine\">";
            if($source eq "perl"){
                $line .= "#";
            }
            elsif($source eq "c") {
                $line .= "//";
            }
            $line .= "</span>";
        }
        $line = $linenumber.$line."\n"; # Restore $line

        # Remove newline #1, #2 and last
        $line =~ s/(<pre>|<tt>|<\/tt>)\n/$1/;
        # Fixed by patch src/genhtml/htmldocgenerator.cc

        print HTMLFILE $line;
    }

    if(!$nodoc_sw eq ""){
        print HTMLFILE # Write end tags to HTML file
            "\n".
            "  </body>\n".
            "</html>\n";
    }
    close(HTMLFILE);
}



$f_sw = $n_sw = $s_sw = $i_sw = $o_sw = $c_sw = $T_sw = $nodoc_sw = "";
$source = $infile = $outfile = $title = $cssPath = "";

# Parse ARGV
while(@ARGV) {
    $option = shift;
    if($option eq "-n") {
        $n_sw .= "-n ";
    }
    elsif($option eq "-f") { # source-highlight only (aquired)
        $f_sw .= "-f ";
        $f_sw .= shift;
        $f_sw .= " ";
    }
    elsif($option eq "--no-doc") { # source-highlight only
        $nodoc_sw .= "--no-doc ";
    }
    elsif($option eq "-s") {
        $s_sw .= "-s ";
        $source = shift;
        $s_sw .= $source." ";
    }
    elsif($option eq "-i") {
        $i_sw .= "-i ";
        $infile = shift;
        $i_sw .= $infile." ";
    }
    elsif($option eq "-o") {
        $o_sw .= "-o ";
        $outfile = shift;
        $o_sw .= $outfile." ";
    }
    elsif($option eq "-c") {
        $c_sw = "-c ";
        $cssPath .= shift;
        $c_sw .= $cssPath." ";
    }
    elsif($option eq "-T") {
        $T_sw = "-T ";
        $title .= shift;
        $T_sw .= $title." ";
    }
    else {
        msg();
    }
}

# Check parameters and convert
if($s_sw eq "" | $i_sw eq "" | $o_sw eq ""){
    msg();
}
elsif($source eq "perl" | $source eq "c"){
    if($f_sw eq ""){
        msg();
    }
    print
        "src2html ".$infile." =>\n".
        "         ".$outfile."\n";
    system("source-highlight ".
           $f_sw.$n_sw.$s_sw.$i_sw.$o_sw.$c_sw.$T_sw.$nodoc_sw);
    modify(); # Modify converted file
}
elsif($source eq "vhdl" | $source eq "ucf"){
    print
        "src2html ".$infile." =>\n".
        "         ".$outfile."\n";
    system("vhdl2html ".
           $n_sw.$s_sw.$i_sw.$o_sw.$c_sw.$T_sw);
}
else {
    msg();
}

system("chmod 755 ".$outfile);
```

to_html

```
#!/local/bin/perl5 -w

#
# Created: June 30 2004
#
# Author:  Geir Nilsen <geirni@ifi.uio.no>
#

use File::Basename;

$html_dir = "/hom/geirni/www_docs/research/ise/html/"; # Output dir
$css      = "http://heim.ifi.uio.no/~geirni/src2html.css";
@files    = `find cam common ids devboard`;

print "\n";

for $file(@files){
    $file =~ s/\n//;

    if (-f $file){
        if ($file =~ /\.vhd$/){ # VHDL
            system("src2html         -n -s vhdl -i ".$file.
                   " -o ".$html_dir.$file."\.html -c ".$css.
                   " -T ".basename($file));
        }
        elsif ($file =~ /\.ucf$/){ # ucf
            system("src2html         -n -s ucf  -i ".$file.
                   " -o ".$html_dir.$file."\.html -c ".$css.
                   " -T ".basename($file));
        }
        elsif ($file =~ /\.pl$/){ # Perl
            system("src2html -f html -n -s perl -i ".$file.
                   " -o ".$html_dir.$file."\.html -c ".$css.
                   " -T ".basename($file)." --no-doc");
        }
        elsif ($file =~ /\.c$/){ # C
            system("src2html -f html -n -s c    -i ".$file.
                   " -o ".$html_dir.$file."\.html -c ".$css.
                   " -T ".basename($file)." --no-doc");
        }
    }
}

$thisfile = "to_html";
system("src2html -f html -n -s perl -i ".$thisfile.
       " -o ".$thisfile."\.html -c ".$css." -T ".$thisfile);

print "\n";

system("chmod 755 -R ".$html_dir);
```

F.5 Optimization of Components in CAM

This is a “one-hot” encoder that can be used with the CAM as described in this project. Due to little time left of writing this report, it has not been used.

encode_opt.vhd

```
--
-- File    : encode_opt.vhd
-- Created : August 3 2004
-- Project : cam_srl16e
-- Author  : Geir Nilsen { geirni@ifi.uio.no }
--
-- Description:
--   An attempt of optimizing the encoder for speed and a large number of
--   inputs. This encoder is "one-hot".
--

library ieee;
use ieee.std_logic_1164.all;

entity encode is
  port(
    -- '1' if match is found
    match     : out std_logic;
    -- Match address
    addr      : out std_logic_vector(4 downto 0);
    -- match_bus from CAM-words
    match_bus : in  std_logic_vector(31 downto 0)
    );
end encode;

architecture encode of encode is
begin

addr <= "00000" when match_bus( 0) = '1' else
```

```
        "00001" when match_bus( 1) = '1' else
        "00010" when match_bus( 2) = '1' else
        "00011" when match_bus( 3) = '1' else
        "00100" when match_bus( 4) = '1' else
        "00101" when match_bus( 5) = '1' else
        "00110" when match_bus( 6) = '1' else
        "00111" when match_bus( 7) = '1' else
        "01000" when match_bus( 8) = '1' else
        "01001" when match_bus( 9) = '1' else
        "01010" when match_bus(10) = '1' else
        "01011" when match_bus(11) = '1' else
        "01100" when match_bus(12) = '1' else
        "01101" when match_bus(13) = '1' else
        "01110" when match_bus(14) = '1' else
        "01111" when match_bus(15) = '1' else
        "10000" when match_bus(16) = '1' else
        "10001" when match_bus(17) = '1' else
        "10010" when match_bus(18) = '1' else
        "10011" when match_bus(19) = '1' else
        "10100" when match_bus(20) = '1' else
        "10101" when match_bus(21) = '1' else
        "10110" when match_bus(22) = '1' else
        "10111" when match_bus(23) = '1' else
        "11000" when match_bus(24) = '1' else
        "11001" when match_bus(25) = '1' else
        "11010" when match_bus(26) = '1' else
        "11011" when match_bus(27) = '1' else
        "11100" when match_bus(28) = '1' else
        "11101" when match_bus(29) = '1' else
        "11110" when match_bus(30) = '1' else
        "11111" when match_bus(31) = '1';

-- Generate the match signal if one or more match(es) is/are found
match <= '0' when match_bus =
  "00000000000000000000000000000000"
  else '1';

end encode;
```

9 BIBLIOGRAPHY

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

2 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.

3 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.

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

5 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.

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

7 Paul E. Proctor. *The Practical Intrusion Detection Handbook.* Prentice Hall PTR, 2001. ISBN 0-13-025960-8.

8 Jay Beale et al. *Snort 2.0 Intrusion Detection[p. 1-60]*. Syngress Publishing, Inc. 2003. ISBN 1-931836-74-4.

9 http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/Spoofing/default.htm

10 http://www.snort.org/

11 James Moscola et al. *Implementation of a Content-scanning Module for an Internet Firewall*. FCCM 2003.

12 John Villasenor, William H. Mangione-Smith. *Configurable Computing*. In *Scientific American, 6/97*.

13 David Van den Bout. *The practical Xilinx designer lab book [p. 23-31]*. Prentice Hall, 1999. ISBN 0-13-021617-8.

14 http://www.optimagic.com/faq.html

15 J-L Brelet. *An Overview of Multiple CAM Designs in Virtex Family Devices*. Xilinx Application Note 201, September 23, 1999 (Version 1.1).

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

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

18 Course at Institute of Physics, University of Oslo.
”Fys329: Data assistert konstruksjon av kretselektronikk”

19 Memec Design. *Virtex-II Pro(P4/P7) Development Board. User Guide, Version 4.0, June 2004. PN# DS-Manual-2VP4/7-FG456.*

20 Memec Design. *P160 Communications Module. User Guide, Version 2.0, December 2002. PN# DS-Manual MBEXP1.*

21 http://www.beyondlogic.org/porttalk/porttalk.htm

22 Memec Design. *Seiko LCD Operating Instructions* (Provided by Memec).