Ethernet Packet Inspection

Marius-Cristian ANDREI
cristyhj91@gmail.com

Submitted for the 2019 Digilent Design Contest Europe

03.05.2019

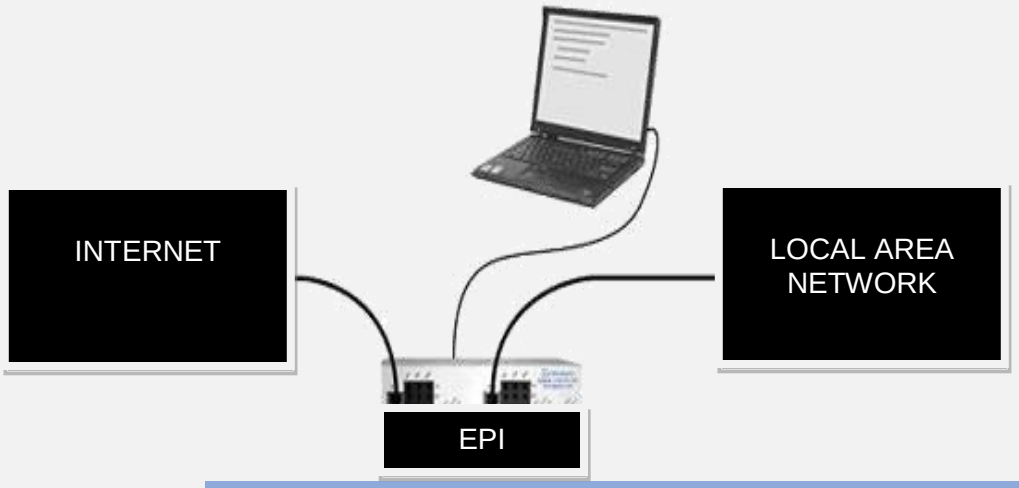
Advisor: Conf. Univ. Dr. Ing. Laurențiu MĂRGĂRIT

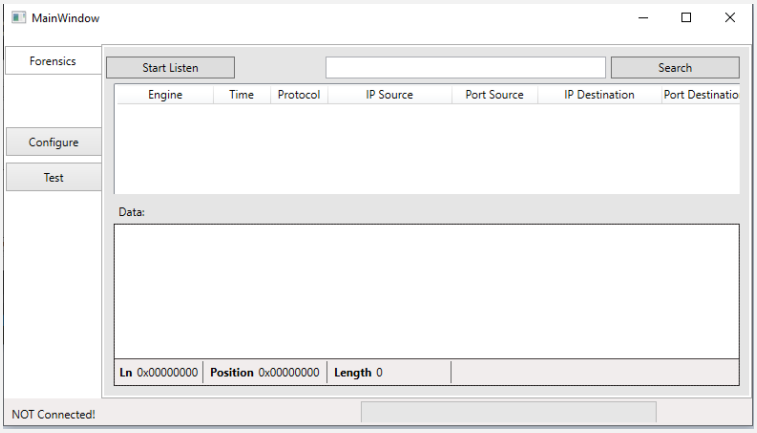
MILITARY TECHNICAL ACADEMY "FERDINAND I"
Bucharest, Romania

Security

Better than a firewall, EPI system is a good IDS solution to protect your Local Area Network.

Fast and powerful, can integrate up to 16 rules and still maintaining the same speed.





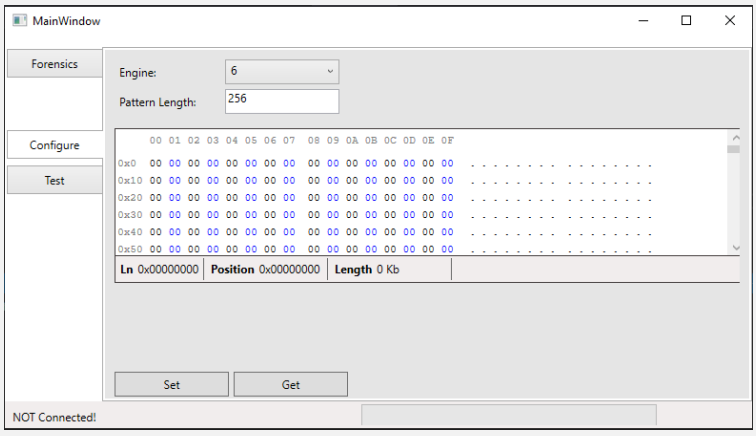
Information

With an intuitive interface, a network administrator can easily detect any intrusion by analyzing every Ethernet packet.

Easy to use

Having a simple structure and intuitive user interface, EPI system is very easy to configure.

With powerful Hex Editor, the user can easily insert any desired data.



Introduction

Abstract

Mostly, harmful software infiltrates in personal computers via local network. This can be represented by viruses, worms, ransomwares, spywares, rootkits and many more. Best way to stop or detect intrusion is to analyze network traffic at kernel-space level.

Hybrid architectures with microcontroller and field programmable gate array (FPGA) are an efficient solution relative to execution speed, implementing parallel computation units that can detect specific byte sequences.

Objectives

IDS (intrusion detection system) is an equipment or a software application which monitors a network or a system, and its purpose is to detect a malicious activity or violation of certain policies.
This project aims to identify, using the FPGA capabilities, an unauthorized data and to notify the user to take action.

Any Ethernet packet received will be redirected to the PL and further analyzed. Using a linear string matching algorithm, the PL will take the decision to send or not to send the packet to the PS. If the second option is chosen, then the user will be notified and some important information about the packet will be stored.

Features-in-Brief

The project aims to speed up the process of finding malicious data at least two times. Theoretically, it can compute 16 times faster, but in practice, due to preparation and data transfers, one expects a speed up around 3 times.

Project Summary

The hardware project has been developed using Xilinx Design Tools Vivado HL Design Edition 2018.2 and the designs have been described in verilog.

The software project has been developed using Microsoft Visual Studio Community 2017 in WPF.

Its main component is a module with a specific function and it has scaling and reuse capabilities.
The design implements the Knuth–Morris–Pratt string matching algorithm with linear approach, parallelism coming in a matter of multiple searching engines.

Digilent Products Required

- Zybo Z7 development board
- Micro SD card with minimum 8 GB memory
- USB cable
- Network cable

Tools Required

- Xilinx Design Tools Vivado HL Design Edition 2018.2
- Microsoft Visual Studio Community 2017
- Linux machine (with ARM cross compiler)

Design Status

The design is complete and meets its objectives. Although, the entire system is still in development and can be enchanted with new capabilities. The hardware design has been verified through simulation and physical implementation.

Background

Why This Project?

IPS/IDS Systems are very common in network security. The most popular is Snort, which is a free open source network intrusion detection system (IDS) and intrusion prevention system (IPS) created in 1998 by Martin Roesch. Other free and open source systems are ACARM-ng, AIDE, Bro NIDS, Fail2ban, OSSEC HIDS, Prelude Hybrid IDS, Sagan, Samhain and Suricata.

In the usual approach, this kind of system can become slowly and inefficient. Using the FPGA capabilities, we can develop a solution to this problem, moreover, a better and a faster system.

The system uses the fastest linear string matching algorithm. Knuth–Morris–Pratt string-searching algorithm (or KMP algorithm) searches for occurrences of a ”word” within a main ”text string” by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

Reference Material

The project was initially developed by Andrei Georgian, as a part of his dissertation diploma, having just the main functionalities.
The next reference materials were used:

- For the string matching algorithm: https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm#Example_of_the_search_algorithm
- For Linux driver: “Linux Device Drivers, Third Edition” - Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman
- For DMA driver: “Linux DMA in Device Drivers” - John Linn

Design

Features and Specifications

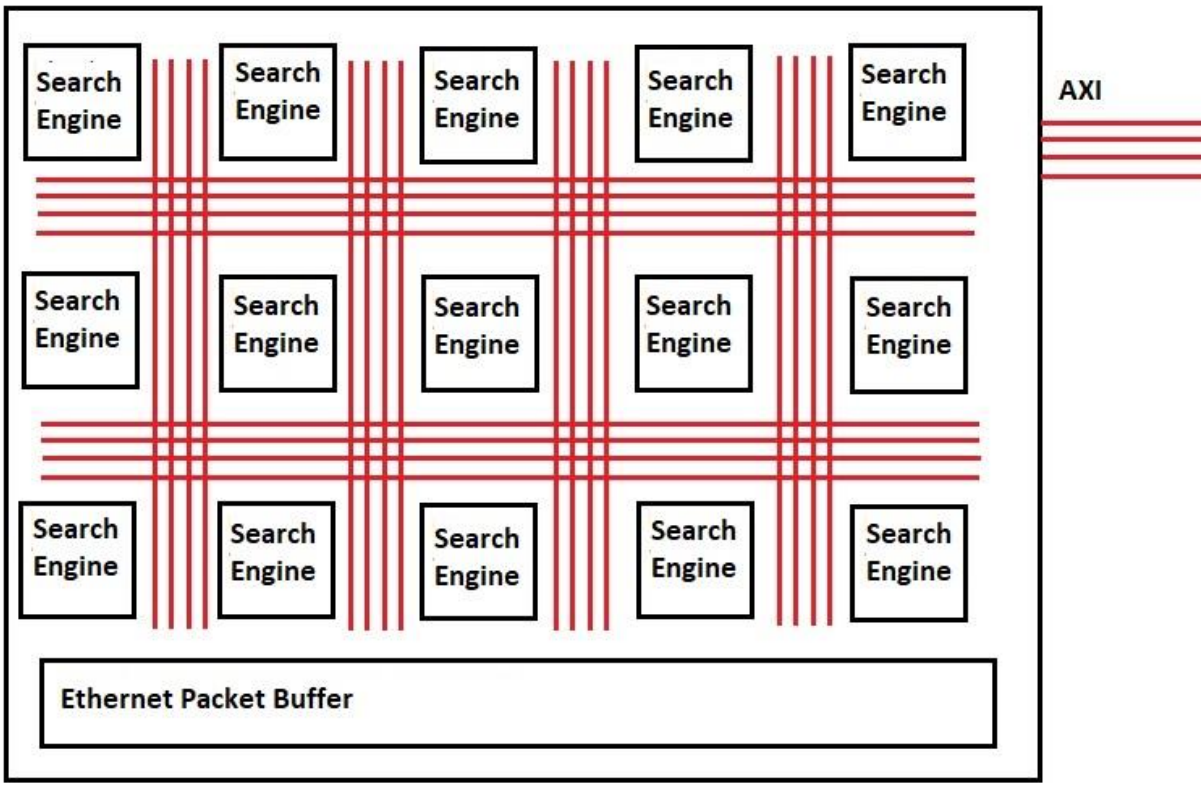
The hardware design is looking forward to implement as many search engines as possible.

Design Overview

The design is capable to generate 16 blocks, each containing one 32 bits configuration register and one 256 bytes pattern buffer. Each one of these registers can be accessed with AXI Lite protocol.
One memory buffer is generated in order to transfer the Ethernet packet into the hardware design. One can access this buffer using AXI Stream protocol. DMA capabilities are used.
The design has:

- one AXI Lite interface
- one AXI Stream Master interface
- one AXI Stream Slave interface

The inner design implements one AXI Lite interface used to transfer bytes from memory buffer to each engine.



Detailed Design Description

In order to use the system, it has to be configured. Each engine responds to an address computed by the formula:

$$engine\ address = engine\ id * (256 + 4)$$

Therefore, beginning with that offset address, it has 4 bytes for configuration register and 256 bytes for pattern configuration. In order to successfully set an engine one has to set the pattern configuration first, then to set the configuration register.

The configuration register contains the following information:

x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	x
24 bits representing pattern length																					reserved							*

*last bit is set if the engine is active or clear if the engine is inactive.

After setting the configuration register it will be increased as follows:

$$configuration\ register = (pattern\ length + 4) \ll 8 + engine\ on\ bit$$

in order to set the entire engine space (pattern length + 4 bytes configuration register). At start-up everything is set to 0.

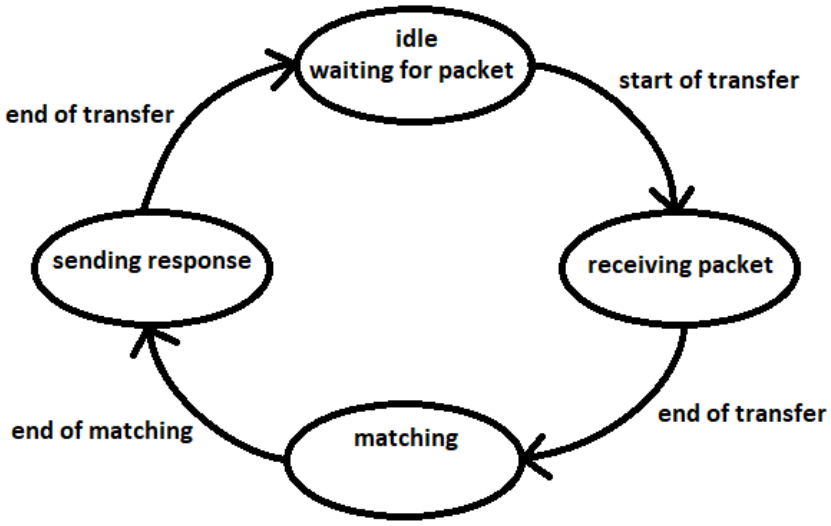
After all the configurations were made, the system is waiting to receive data on AXI Stream Slave interface, the state being called „idle”. Begging with the first byte, the system enters in „receiving packet” state. After all the data were sent, it enters in „matching” state until one engine finds the pattern or all the engines finished without success the string matching. Then enters in „sending response” state, where the AXI Stream Master interface send 4 bytes as response, then enters again in „idle” state.

The 4 bytes configuration is:

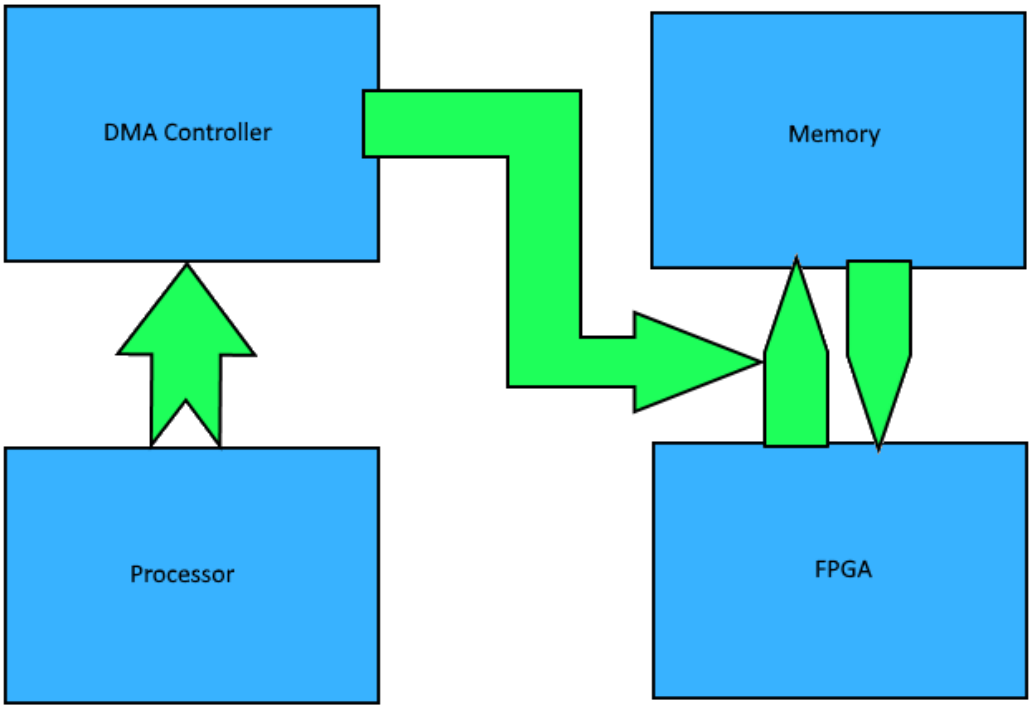
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x
reserved																												*	

*2 bits complementary one from another. If the less significant bit is set, the other is clear, meaning that one engine found the pattern. Otherwise, the less significant bit is clear and the other is set, the system did not find anything.

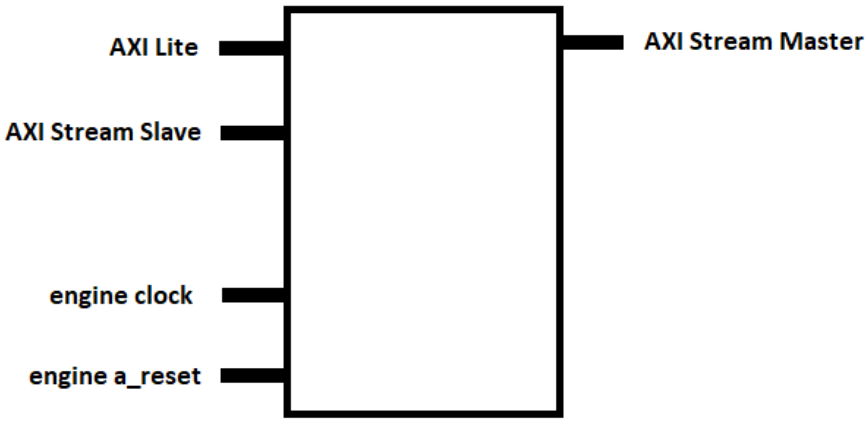
The state diagram is:



The top level block diagram instantiates the inspection unit block (the one described earlier). It takes advantages of DMA transfers through DMA IP block, that is directly connected. The top level design looks like this:



The main actor of the top level module is the Inspection Unit within the FPGA. It interacts with memory using the three AXI interfaces as described in the next block diagram:



The „engine clock" signal is the clock at which rate the engines are computing. It is configured to run twice as fast as the Processing System's clock (200 MHz while the Processing System's clock is 100 MHz).

The software design consists of three modules:

- Linux device driver
- Linux user-space application
- Windows interface application

Linux device driver has the capabilities to control the DMA transfers to send big amount of data to the FPGA, set engines via „ioctl" function and implements net filter hook.
File interaction functions has the next functionalities:

- „write" – send the data received to FPGA to search patterns in it
- „read" – read the data saved by the net filter
- „ioctl" – set the engine and get the information about them and about the system

All of these implementations can be found in „epi_device.c" file.
DMA functions can be found in „epi_dma.c" and the main actor is:

```
void axidma_transfer
      (char *src, int length, struct epi_dev *private_data, int
force_no_wait)
```

which takes 4 arguments. The first two describe the data, the address offset and the length. The next argument is the device structure which contains the information about the DMA driver. The last argument can be 0 if the transfer needs to wait to finish the transfer or 1 otherwise.

The net filter's duty is to capture the packets at kernel space level. It implements a hook function which is called when an Ethernet packet arrives. It is hooked after it passes the simple sanity checks and before routing the packet forward. The net filter sends the packet to the FPGA and waits for response. If the response tells that an intrusion is detected, the net filter logs the data into a circular buffer. The net filter always let the packet pass.

The main purpose of the Linux user-space application is to test the system in real time conditions. It implements 6 types of tests running from the command line.

```
./test
```

This command tests the main purpose of the system: the string matching algorithm. It takes an ASCII string from the user input and sends it to the Linux driver using „write" system call. Then the driver program the DMA to transfer it into the FPGA. After the FPGA finishes searching the pattern the response is received via DMA by the driver. The „write" system call returns 1 if the pattern was found or 2 otherwise.

```
./test stress
```

This command tests the speed of the system. Firstly, it transfers 1000 sample packets into the system and computes the average time taken for the string matching algorithm. Then it computes the time taken for a linear string matching algorithm that search a 256 bytes long pattern in an 1024 bytes long string 16 times. Then it computes the speed-up by the formula:

$$speed\_up = \frac{average\ linear\ time}{average\ parallel\ time}$$

```
./test set
```

This command tests the configuration of the engines. It sets all the engines with patterns read from „patterns.txt" file. Then waits for user input, a number from 0 to 15 meaning the engine index whose pattern will be read from the FPGA and printed on the stdout.

```
./test uart
```

This command opens the UART communication with the host computer. Setting up the connection, it disables the „getty service", then opens the „/dev/ttyACM0" file. The communication protocol is managed by „uart_send" and „uart_recv" functions. It supports 3 commands:

- SetEnginePattern
- GetEnginePattern
- Exit

This command needs to end with the „Exit" message to restart the „getty" service.

```
./test logs
```

This command print to „stdout" the logs captured by the net filter logic. It parses the circular buffer reading the logs and clearing the space to be reused. The logs format is ready to be send via UART to the host computer.

```
./test listen
```

This command executes the „logs" command and sends the logs to the host computer. It is waiting forever for logs to appear in the circular buffer, or stops if an error occurs, signalized by the response received from the host computer.

The Windows interface application in written in C# using WPF. It represents a SIEM (Security Information and Event Management) for EPI system. In the main screen the user can start reading the logs from the board. Selecting one log it displays information about it and the packet's data.

The configure screen is able to configure the engines and get the information about actual configuration. To set the engine the user has to select the engine index from the combo box, the to set the engine pattern length in the text box and press enter. Using the Hex Editor, the user enters the pattern data. After everything is done, the Set button starts the transfer. The result is displayed in the status area, at the bottom of the page.

Discussion

Problems Encountered

The main design issue is the net filter capabilities. The hook is called from a soft interrupt request and it must not be scheduled. The DMA transfer waits for an interrupt to signal the end of transfer. Then, between transfers, while the engines are searching the patterns, the DMA transfer function can be scheduled, resulting in a kernel panic. The system then needs to boot again.

Due to the issue up mentioned, the system will transfer to the SIEM two sample data, one for an ICMP packet and one for an TCP packet.

Engineering Resources Used

- Linux Kernel Xilinx – alongside with Digilent's online documentation
- U-Boot Xilinx – alongside with Digilent's online documentation

Marketability

Marketable Securities of the products relies on the high demand of security equipment and the high price on the market comparing with the small production price of the system.

As the Grand View Research site states, "the global security equipment market is expected to witness significant growth owing to the increasing demand for enhanced security coupled with easy installation". The initial project design was meant to follow the demand of the market.
Although there are already some major companies selling security systems on the market, like Bosch Security Systems and Cisco Systems, EPI project aims to compete with them, being modularized and scalable.

References

- "The Zynq Book Tutorials" – Louise H. Crockett, Ross A. Eliot, Martin A. Enderwitz, Robert W. Stewart
- Zynq-7000 All Programmable SoC: Embedded Design Tutorial
- https://en.wikipedia.org/ Knuth Morris Pratt_algorithm