


<b>ITC 1/50</b> <b>Information Technology and Control</b> <b>Vol. 50 / No. 1 / 2021</b> <b>pp. 28-44</b> <b>DOI 10.5755/j01.itc.50.1.25321</b>	<b>Packet Analysis: A Model Implementation of a Special Communication Protocol Using Micro Programming</b>	
	Received 2020/02/17	Accepted after revision 2021/01/20
	 <a href="http://dx.doi.org/10.5755/j01.itc.50.1.25321">http://dx.doi.org/10.5755/j01.itc.50.1.25321</a>	

**HOW TO CITE:** Chaaban, Y. (2021). Packet Analysis: A Model Implementation of a Special Communication Protocol Using Micro Programming. *Information Technology and Control*, 50(1), 28-44. <https://doi.org/10.5755/j01.itc.50.1.25321>

# Packet Analysis: A Model Implementation of a Special Communication Protocol Using Micro Programming

**Yaser Chaaban**

Assistant Professor; Department of Computer & Information Sciences; Faculty of Science and Arts - AlUla Branch; Taibah University; Saudi Arabia; e-mail: [yshaaban@taibahu.edu.sa](mailto:yshaaban@taibahu.edu.sa)

Corresponding author: [yshaaban@taibahu.edu.sa](mailto:yshaaban@taibahu.edu.sa)

Communication protocols are used in telecommunication systems. These protocols are defined as rules, which enable the entities of communicating systems to transfer information usually as packages. Additionally, each standard communication protocol has a uniquely-defined structure and consequently, a special pattern of network packets. It is worth mentioning that communication protocols are implemented in different ways, in the system software layer or the hardware layer (i.e., silicon chipsets). This paper presents the implementation of a special communication protocol called "Packet Analysis", which is used in the Hardware project "Minimax machine". This implementation is a software that can be written using a special simulator, "Minimax simulator", which is the target execution environment. That simulator was successfully developed for micro-programming and hardware simulations. In this regard, this study develops an algorithm that represents a step toward simulating communication protocols using micro-programming. The flow chart designed here gives an overview of how the "Packet Analysis" algorithm works (designed protocol), which in turn describes all steps in detail. As a result, the entire system of this research paper was implemented and tested with various input values. Additionally, the implemented proposed solution (implemented protocol) was evaluated by two metrics (quantitative measures) using test-benches so that its statistics will be trustworthy for research. Other results

of this study showed that there is a lot of scope for optimization in the solution presented in this research paper. This leads in turn to optimization of the proposed implementation and to consideration of implementing alternatives.

**KEYWORDS:** Communication, protocol, communication protocols implementation, simulation, micro-programming.

---

## 1. Introduction

The implementation stage of protocols gets less attention in "Protocol Engineering" research than other phases of the protocol development process [14]. Based on that, this section introduces a survey of the most closely related work, which was published in the domain of communication protocols implementation. This overview of existing related work intends to highlight the need for developing a novel methodology to cover the gap recognized in this field (research problem).

Chaaban introduced a computer hardware project that designs a special simulator for micro-programming and hardware simulation [2]. Furthermore, he presented two formal measures and metrics to evaluate the implemented programs. Based on that work, this paper presents a model implementation of a special communication protocol called "Packet Analysis". The implementation will be conducted on the micro-programming level utilizing the programming environment (simulation tool) designed by Chaaban. This means that in this research paper, the design and implementation of special hardware architectures will be developed. In this regard, the implemented solution/system performance/designed algorithm has to be measured, aiming to ensure a robust solution.

Therefore, different given test benches, which are test-files as input for the simulator, will be carried out. To achieve this purpose, this paper introduces a practically applicable methodology that integrates various concepts from different research areas, as described later in this research paper.

### 1.1. Related Work

Protocol Engineering is an important discipline. It covers the design, validation, and implementation of communication protocols [14]. Accordingly, different communication protocols are defined for various systems and components. These protocols are usually rules or steps in to achieve the goal of communicat-

ing between system entities. Here, communication devices (entities) should perform those rules (steps) automatically so that a connection can be made and data can be exchanged between them. Additionally, communication protocols are usually implemented differently on both a hardware and/or software level.

When considering sensor communication protocols, some protocols define both levels, hardware and software, i.e., SDI-12. Other protocols specify only the hardware level, i.e., RS-485. Finally, some sensor communication protocols define only the software level, i.e., Modbus RTU [19]. Here it is worth mentioning that communication protocols are implemented, in practice, for the most part in software [14]. In this context, a protocol development has main stages, in which the development phases start from design to integration or installation. These phases are equivalent to those of software development [24].

In database systems, MySQL provides connections between clients and the server using several networking protocols. MySQL communication protocols are needed to create a connection aiming at the information exchange of clients and the server. The main protocols used by clients to connect with MySQL server are TCP/IP, Unix socket file, Named pipe, Shared memory [13]. These protocols are usually implemented by various libraries and program drivers [20]. In this regard, all client programs included in various MySQL distributions (MySQL, mysqladmin, etc) can establish connections to the server using the native C client library [20].

In the industrial field, there are many popular communication protocols and networks. The most famous and widely used industrial communication protocols are standards, such as Profi-Bus, Profi-Bus DP, Profi-Net-IO, Mod-Bus, Mod-Bus/RTU, Mod-Bus/TCP, Can-Bus, CAN-OPEN, Field-Bus, Ethernet, EtherNet/IP [21]. Details of some of these communication protocols and networks are:

- Modbus RTU and Modbus published by Modicon company in 1979 [17]. Modbus RTU is an open serial protocol which is widely used in today's industrial equipment of monitoring and control. This protocol uses the serial interface RS-232 or RS-485 for communications [5]. On the other side, Modbus is a communication protocol used for transmitting information between electronic devices via the Ethernet or over serial lines [18].
- Profibus DP used by Siemens. Profibus DP is the second type of PROFIBUS that is for Decentralized Periphery. It is a much simpler and faster protocol used in most of PROFIBUS application profiles today [22]. The hardware structure of the Profibus-DP and its application in DP slave development are presented in [12].

In this context, an interesting research field, called the Industrial Internet of Things (IIoT), was identified. Here, [11] presented a comparative study of communication protocols for industrial internet of things environment (IIoT). Accordingly, polling-based and event-based protocols were investigated so that an open and interoperable IIoT environment can be realized. The author compared various Internet of Things (IoT) protocols and consequently chose the message queuing telemetry transport (MQTT) as the event-based, publish–subscribe protocol. Additionally, the study found that the MODBUS protocol has an optimized message structure in the application layer that concentrates on industrial applications. As a result, an event-oriented IoT protocol will not replace the MODBUS TCP, but completes it. Based on that, the study introduced two different scenarios to build the IIoT environment. Firstly, building the environment using the MODBUS TCP alone so that the MODBUS TCP can be considered as an IoT protocol. Secondly, using MQTT simultaneously with the MODBUS TCP [11].

Other work related to the Protocol Engineering [14] proposed in order to deal with authentication protocols were presented in literature [15, 10]. These protocols are communication or cryptographic protocols, which take into account the transferring of authentication data between communicating entities. An improved delegation-based authentication protocol for Portable Communication Systems (PCSs) was presented in [15]. The authors demonstrated that Lee-Yeh's protocol has an instinctive design flaw. Based

on that, they proposed a modification to overcome the protocol weaknesses and provided the anonymity service.

On the other side, the work in [10] concentrated on multi-server authentication. It presented an improved biometric multi-server authentication scheme, which is designed for Chang et al.'s protocol. Here, the authors investigated a few multi-server authentication schemes in the literature, although security with loopholes. They reviewed the protocol thoroughly, and consequently proposed their improved model. It is resistant to all known and identified attacks. Furthermore, they presented the formal and informal security analysis, performance, and evaluation analysis.

Although there are various works made towards implementing communication protocols, such as [1, 16, 9, 8, 25, 3, 6, 23], a study of developing a novel methodology to develop an algorithm, which simulates communication protocols using micro-programming, does not exist yet (at least it is extremely rare published as a study).

To the best of my knowledge, this paper represents the first study towards introducing an applicable methodology by integrating several concepts from different research areas to define a model implementation of communication protocols using micro-programming.

---

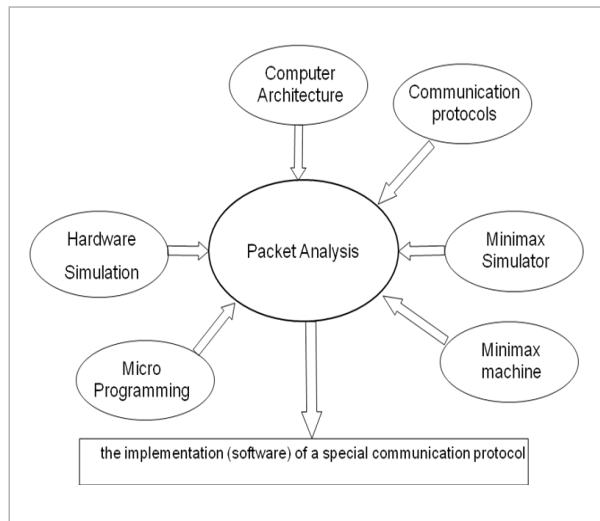
## 2. Concept and Goals of "Packet Analysis"

The main contribution of this paper is the integration of concepts from different research areas into a practically applicable methodology. Here, Figure 1 summarizes the methodologies integrated within "Packet Analysis".

The main goal of the "Packet Analysis" is to implement a special communication protocol. It integrates many concepts from different research areas such as computer architecture, communication protocols, micro-programming, and hardware simulations. Additionally, it uses a special simulator called "Minimax simulator", which was developed for a micro-programming project called "Minimax machine". Consequently, "Packet Analysis" produces software that

**Figure 1**

The methodologies integrated within "Packet Analysis"



implements a special communication protocol following the concepts mentioned above.

To emphasize this concept, it is worth mentioning some examples of the practical utilization of the proposed protocol. Data transmission/transport systems could utilize packet analysis protocol, especially in embedded systems, where limited resources and time are considered. Here, the exact size of packet fields can be re-adjusted (re-configured) to allow for an optimal implementation to be utilized in practice. That means, the "packet analysis" protocol allows different data lengths in communication protocols used in embedded systems. One of the practical application of this protocol could be wireless sensor networks, more precisely, communication protocols used in embedded sensor networks. Moreover, the protocol proposed in this research paper concentrates on software implementations of communication protocols used in hardware architectures. In this context, "packet analysis" demonstrates an example scenario of common communication protocols designed especially for simplified/ abstracted hardware architectures. Therefore, this concept developed an algorithm, which could be utilized in practice, to simulate communication protocols using micro-programming.

The next section will summarize the concepts, mentioned above, from different research areas that form "Packet Analysis".

## 2.1. Computer Architecture

**Defining Computer Architecture:** Computer designers encounter a complex task in specifying the most significant attributes need for designing a new computer so that a trade-off exists between high performance and efficient energy on one hand and reasonable cost, power, and availability on the other [7]. In this context, instruction set design, logic design, functional organization, and implementation are important issues. Additionally, developing a better design depends on various technologies, such as operating systems, compilers, logic design, and packaging [7]. Hennessy and Patterson in [7] believe that the term computer architecture referred not only to instruction set design but also to other aspects of computer design such as implementation. They wrote: "Several years ago, the term computer architecture often referred only to instruction set design. Other aspects of computer design were called implementation, often insinuating that implementation is uninteresting or less challenging. We believe this view is incorrect. The architect's or designer's job is much more than instruction set design, and the technical hurdles in the other aspects of the project are likely more challenging than those encountered in instruction set design."

## 2.2. Communication Protocols

**Defining Communication protocols:** Communication protocols used in telecommunication systems are norms or rules. The role of these rules is to allow units of the designed system to communicate and exchanging information (structured messages called packages) over a network. In this regard, communication protocols may be implemented in software or hardware chipsets or both. Internet Protocol (IP) and Transmission Control Protocol (TCP) is the most widely used protocols on the Internet and Intranet networks so that data between two computers (machines) can be transmitted, and consequently communication between them is realized. In this context, Request For Comments (RFCs) of the Internet Engineering Task Force (IETF) is supported by the Internet Society (ISOC) to describe (define) the most widely used communication protocols of the Internet. Moreover, network protocols consist of many layers called protocol stack. Here, a reference model (OSI), which is a software architecture, plays the role to

specify each layer with its provided services. This OSI model has the classic seven layers (protocol stack).

### 2.3. Micro-Programming and Hardware Simulations

Recently, micro-programming of machines have become more common, because it is used in several fields, e.g., as Computer Engineering. Additionally, micro-programming is employed throughout the design process. It is an important technique to understand how a compiler construction and computer works. In this regard, Computer Simulation Tool (CST) was introduced to develop a suitable tool helping in simulating machines (hardware) and micro-programming so that programmers can test their program execution in a user-friendly way. CSTs use simulators (software packages) to emulate the behavior of machines and hardware systems (hardware simulation). One of the most important tools is VHDL (VHSIC Hardware Description Language). It is a hardware description language used in electronic design automation and parallel programming languages.

The next section presents the "Minimax simulator", which is used in the "Minimax project". This simulator was developed for micro-programming and hardware simulation. It facilitates the process of micro-programming, significantly enabling programmers to understand easily how their programs work.

Chaaban in his work in [2] presents a survey of related work that was published in the domain of micro-programming or hardware simulation. It can be considered as an overview of some architectures or approaches introduced concerning the micro-programming and hardware simulation.

### 2.4. Minimax Machine

Project Computer Engineering (mini-project: Minimax machine) was created and carried out at Department of Systems and Computer Architecture [4] at Leibniz University Hannover (LLH), Germany. Here, the general task is to solve group-based programming tasks based on the Minimax machine known from the lecture "Introduction to Computer Engineering". To solve the tasks, it is necessary to suitably expand the given basic structure of the machine (new ALU commands, additional registers). Programming for problem-solving will be done later on the micro-program as well as the assembler level.

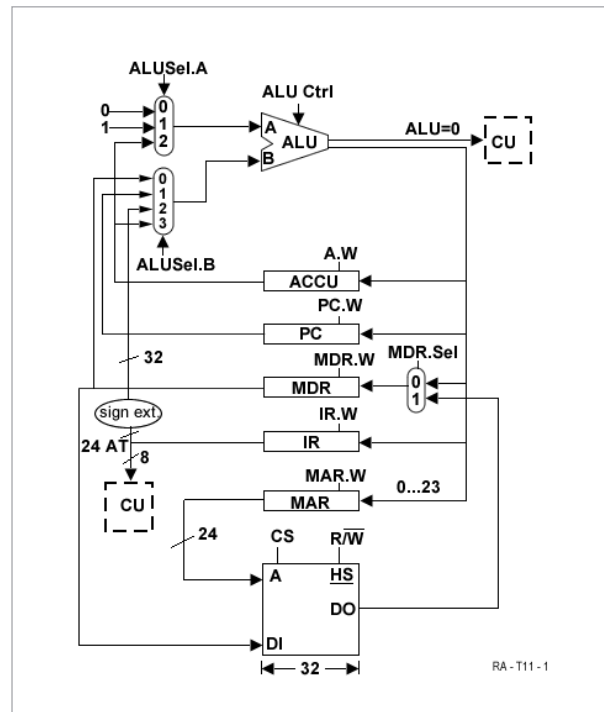
To solve the tasks, a Minimax simulator (as a Java program) is available, with which the project participants can also test their architectural extensions, micro- and assembler programming at home.

#### 2.4.1. Minimax Architecture

As part of the lecture "Fundamentals of Computer Engineering", basic knowledge of computer architecture is taught using the example of the Minimax machine (see Figure 2).

**Figure 2**

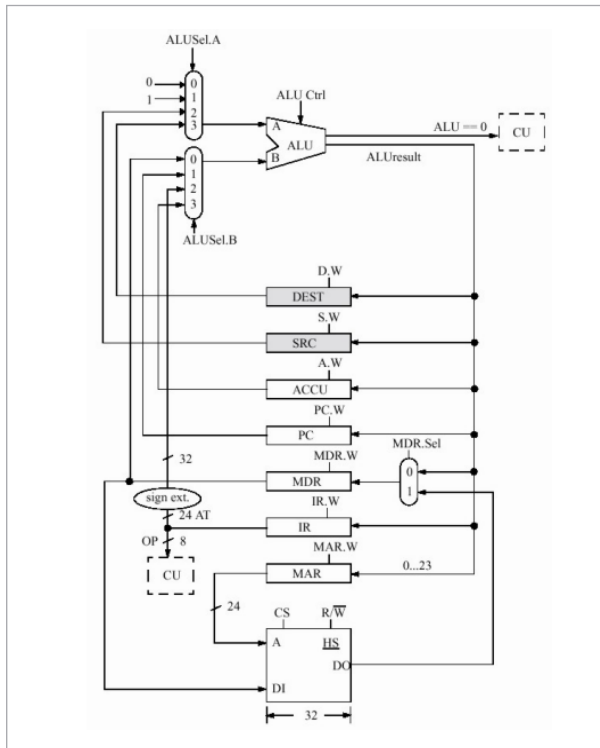
The basic Minimax machine



This machine consists essentially of some registers, an ALU and a memory in which code and data are stored together. The command sequence (instruction cycle) is determined by a micro-program. The architecture of the Minimax machine is largely fixed but is extended for exam and practice tasks by additional registers or Sign Extension Units. Likewise, the ALU may be supplemented with additional functionality (e.g., DIV command). Figure 3 shows an extended Minimax machine.

By extending the basic structure of the Minimax machine, the specific assignment of the Minimax ma-

**Figure 3**  
The Extended Minimax machine



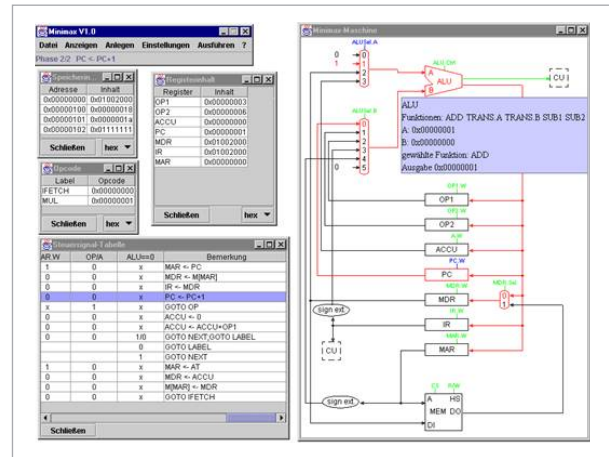
chine with control signals changes. With the Minimax simulator, it is possible to take these changes into account and to simulate the modified data path.

For the concrete solution of the design task, the programmers can decide which parts of the functionality should be moved into the micro-program and how the machine program, based on it, looks like. Assessment standards are used for the programmers in addition to the error-free execution and the number of required clock cycles. The amount of resources used (additional registers and ALU commands) is also included in the evaluation.

### 2.5. The Minimax Simulator

For the Minimax machine, there is a visualization environment for displaying the execution behavior of micro programs (see Figure 4). The visualization environment reads a textual description of the specific Minimax-shaping (Minimax characteristic) and generates there from a clear graphical representation of the architecture to be simulated.

**Figure 4**  
The Minimax simulator during the execution of a machine program



The user then has the opportunity to read micro programs and pre-configuration (default value(s) of registers and main memory addresses) to simulate the execution of machine programs then.

During this execution, which can be switched on step by step through pressing a button (Enter button and Return key), the occupied (allocated) resources are colored in the visualization. This allows a step-by-step simulation whereby the register and memory contents can be monitored at any time. It is also possible to set breakpoints in the simulator.

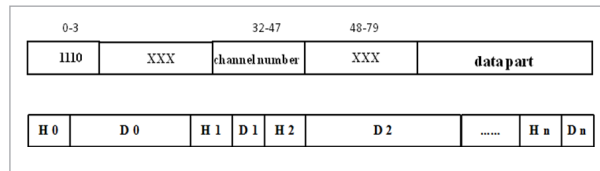
To ensure platform independence and use over the Internet, the implementation has been carried out as an application for Java 2 (Standard Edition, at least V1.3.1).

## 3. Problem Description (Task) of the "Packet Analysis"

Different data packets are stored in the memory of the Minimax machine. Each packet contains a header of (80) bits and a data portion of variable length. A packet starts with the specified pattern (template) 1110. Within the header, the channel number is represented in two consecutive bytes from bit number 32 onwards. A channel includes several packets with a unique channel number. The Figure 5 shows the special communication protocol "Packet Analysis" Format.

**Figure 5**

The special communication protocol "Packet Analysis" Format



Now the request (problem description) can be drafted as follows:

Implement the "Packet Analysis" algorithm, which creates a memory (storage) table of the channel numbers and the length (in bits) of the data portion (data part) of all packets. The ACCU initially contains the memory address X, where the packet field begins, and the length of the field (in bits) is passed in the command (in the instruction as a parameter by calling the program). The memory (storage) table should be stored from any memory address outside the packet field area.

Here, the basic Minimax machine, depicted in Figure 2, should be extended (Figure 3 is an example of an extended Minimax machine) by the implementation of the "Packet Analysis" algorithm. That is because the architecture of the basic Minimax machine is largely fixed, but is extended for big tasks (by additional registers, additional Sign Extension Units, additional functionality of ALU, e.g., DIV command, etc.).

## 4. The Basic Minimax Machine

This basic machine is a simple one-address example machine that is used for a more detailed understanding of the data path structure and then to develop the controller. It has the following properties:

- 1 ALU for arithmetic operations, also used for address calculation.
- 1 bit status signal: COND = 1 if ALU == 0.
- Arithmetic Operations Combine ACCU and MDR (MDR: Memory Data Register)
- ALU inputs A and B are occupied by MUXes (Multiplexer); no buses.
- HS: address 24 bit, data 32 bit.

This predefined architecture (basic machine) of the Minimax machine has only four arithmetic operations,

i.e., the operations set of Minimax-ALU are: ADD, SUB, TRANS.A, TRANS.B as depicted in Table 1.

**Table 1**

The four basic arithmetic operations (operations set) of Minimax-ALU

Symbol	ALU operation	ALU Ctrl
ADD	ALUresult $\leftarrow$ A + B	00
SUB	ALUresult $\leftarrow$ -A + B	01
TRANS.A	ALUresult $\leftarrow$ A	10
TRANS.B	ALUresult $\leftarrow$ B	11

While the MINIMAX registers are shown in Table 2.

**Table 2**

The basic MINIMAX registers

Register	Width (W)	
ACCU	32	Accumulator
PC	32	Program Counter; $PC_{31..24} = 0$
MDR	32	Memory Data Register
IR	32	Instruction Register
MAR	24	Memory Address Register

MINIMAX selectors can be seen in Table 3.

**Table 3**

The basic MINIMAX selectors

ALUSel.A	A	ALUSel.B	B
0	0	0	MDR
1	1	1	PC
2	ACCU	2	$(IR_{23})^s @ IR_{23..0}$
		3	ACCU

MDR.Sel	MDR
0	ALUresult
1	HS.DO

Additionally, Control HS and write signals are explained in Table 4.

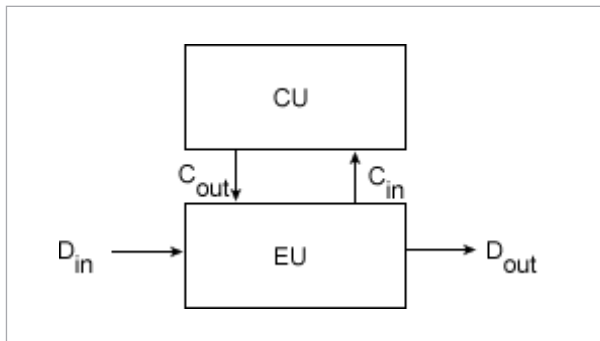
**Table 4**  
The Control HS and write signals of basic MINIMAX

Write signals A.W, PC.W, MDR.W, IR.W, MAR.W		Operations
0		-
1		write
HS		Operations
CS	R/W	
0	0	-
0	1	-
1	0	M(MAR) ← DI
1	1	DO ← M(MAR)

### 4.1. Control Unit (CU)

The task of the control unit is to activate the control lines  $C_{out}$  in the correct sequence, taking into account the signals  $C_{in}$  received from the execution unit (EU) as depicted in the Figure 6.

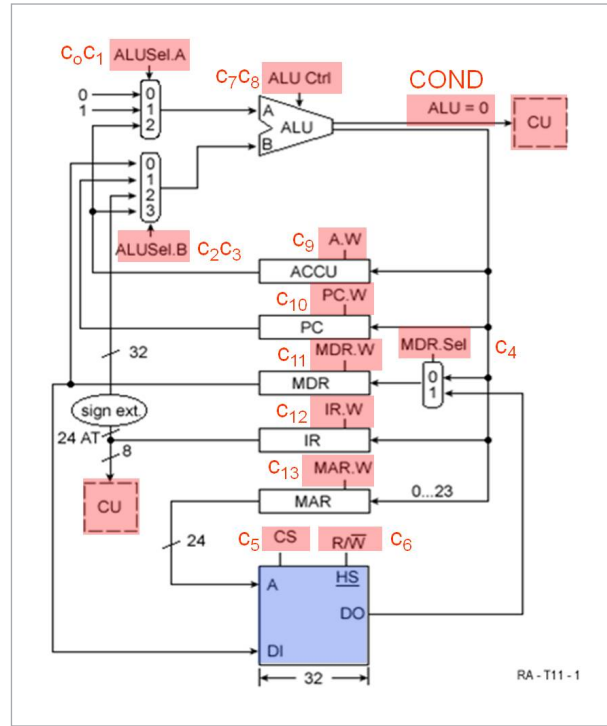
**Figure 6**  
The task of the CU



Here, a typical micro processor needs about 100 to 150 control lines  $C_i^{out}$ . Each control line is responsible (possibly together with others) for the execution of a basic RT operation. The next figure shows the data path of the example machine MINIMAX.

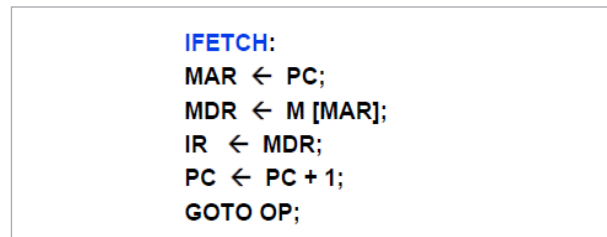
Controllers can be hardwired (hardwired control) or realized by micro-programming. In both cases, the *machine instructions* are to be represented in the form of sequences of RT-operations. This means

**Figure 7**  
The data path of the example machine MINIMAX



that machine instructions should be translated into RT-operations. For example, all commands have the IFETCH phase as follows:

**Figure 8**  
The IFETCH phase



Additionally, each RT-operation is activated by setting certain control signals. The following table shows the control signals  $c_0 \dots c_{14}$  for some machine instructions, where a micro-program determines the instructions sequence.

In this context,  $c_0 \dots c_{14}$  are output signals (for the CU), where COND (ALU == 0) is an input signal.



**Figure 9**  
Control signals for example machine (MINIMAX Micro-program)

Label	Adr. CM	ALU Sel. A	ALU Sel. B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OPI/A	COND (ALU==0)	A	Bemerkung	
Signalnr.	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14		
IFETCH:	0	x	0	1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x		1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0	0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0	1	0	1	x	0	x	ADD	0	1	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1	0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0	0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU ← ACCU + MDR;
	8	x	x	x	0	x	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1	0	x	x	0	x	TRANS.A	0	0	0	0	0	0	1	10	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1	0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;

Furthermore, each row of the control signal table is referred to as a micro-instruction. It consists of one or more micro-operations (RT-operations). The number of micro-instructions required for a particular machine instruction is determined by the CPI (for this instruction), where CPI (Cycles Per Instruction) means the number of cycles per instruction in the case of processing (processor's performance).

## 5. The Algorithm "Packet Analysis"

Now, the algorithm "Packet Analysis" will be explained by finding a solution for a "Packet Analysis" problem.

### 5.1. Initial Situation

The Minimax basic machine consists of four 32 bit registers, which are in detail ACCU, PC, MDR, and IR. There is also a 24-bit address register for memory access. The memory contains both the code and the data. There is an ALU with the operations ADD, TRANS.A, TRANS.B, and SUB.B. A branch in the program sequence is achieved by the ALU result ALU == 0. A Control Unit (CU) activates the control lines.

A micro-program determines the instruction sequence. The architecture is largely fixed but can be

extended by additional registers or sign extension units. Moreover, the ALU can be supplemented with additional functionality.

It is required to extend the basic machine described by a "Packet Analysis" algorithm. This algorithm is used to create a table of different data packets stored in memory, containing their channel numbers and the length (in bits) of the data portion (data part) of all packets of a channel number. This table should be placed at the end of the packet field area. A packet of this packet field area is characterized in that it starts with the specified pattern 1110. Within the header, the channel number is stored in two consecutive bytes from bit number 32 onwards, as depicted in Figure 5. Then, the data portion of the packet follows, starting at bit number 80. This data portion ends where either a new packet with the specified pattern 1110 begins or the entire packet field area ends. In other words, each data portion length (L) of every packet is arbitrary, for example, 2, or 5, or... (later it should also be implemented in that way). Therefore, the following cases are possible:

---

$H0 D0(L= 5 bit) H1 D1(L = 0bit) H2 D2(L = 40bit) H3 D3(L=8 bit) \dots \dots \dots HN DN(L=20bit)$

---

or

---

$H0 D0(L= 0 bit) H1 D1(L = 80bit) H2 D2(L = 80bit) H3 D3(L=144 bit) \dots \dots \dots HN DN(L=80bit)$

---

Here, the designed algorithm must have all possible cases, and the implementation of the algorithm should take into account all these cases. Figure 5 shows the format the special communication protocol designed in this paper, which is called "Packet Analysis".

## 5.2. The Proposed Algorithm

Packet field processing begins by searching for pattern 1110, which is at the beginning of each packet header. From the point where the pattern 1110 was found, 32 bits are jumped further forward, and the 16-bit channel number that begins there is copied into a separate register. After another 32-bit jump, the quasi-read header is on the first data bit of this packet. Now the data bits are counted under consistent pattern search. If the pattern is found, the channel number and the data bit number of the just read packet are determined and can be stored. For this purpose, the first memory address after the packet field is jumped to. Next, the channel number is counted down from there in 64-bit steps (increments) to arrive at a channel-identifying memory location, at which the channel number is entered.

Second, the counted number of data bits is added to the content of the following data word (where is the data length of the channel). Consequently, the processing of the packet is completed. Thus, the Pseudo read head jumps back to the place of the last found occurrence of the pattern +32 bits. Here begins the channel number of the next packet.

There are some notes about the processing mentioned above. Firstly, when counting the data, the field length must be kept in mind so that it is "not counted into the memory or storage table". Consistent decrementing of the length ensures that the end of the packet field is not exceeded except for storage. Once it has been reached, the processing of the packet field is complete. Secondly, in addition to the field length, the current position in the field during processing is saved. For this, the address of the current data word, as well as the index of the current bit are stored in the word. Thirdly, by storing the bit position, the 'data word change' is still guaranteed. If all bits of a data word have been read and used, the next data word must be loaded. This case can occur when counting the data and copying the channel number. Finally, the maximum space required is  $2^{16} * 64$  bits, a value smaller than the available memory of  $2^{24}$  bits, which is initially empty (i.e., 0).

## 5.3. A Detailed Description of the Algorithm

### 5.3.1. Initializing Step

Firstly, the address located in the ACCU, which indicates the beginning of the field, is loaded into the IR. The length of the field is stored in the register FLENGTH. This size is initially used to determine the memory address for the result table, and then it is consistently reduced to be able to determine with their help when the field end is reached. This is followed by writing the address of the data word after the field end in the register ACCU. After the execution of analysis (x), the address, at which the result table starts, will be in the ACCU.

In the BITPOS register, a 0 value is written, in the MEMPOS register, the address from the IR. In the following, the position in the field is stored with these two registers.

For the first packet, the algorithm starts directly when storing the channel number, a data word (32 bits) is continued. We assume that the field is not empty and properly formatted.

### 5.3.2. Copying the Channel Number Step

The channel number is stored in the register KANNUM. It is copied bit by bit by shifting the data word 16 times to the right. If the channel number extends over two data words, this is taken into account and treated accordingly (see remark 1). At each step, the remaining part of the data word is connected via AND with  $2^0$ , so that only the least significant bit is original, the rest is 0. The result is checked for equality with 0. If there is a match, the register KANNUM is shifted to the right and the process starts again after increment of COUNTER and BITPOS. If the result is not equal to 0 (that is, 1), the value is shifted first and then KANNUM is connected to the constant  $2^{31}$  via OR. This corresponds to a left-side insertion of a one value in the register KANNUM, whereas the already stored bits of the channel number remains untouched.

At the end of this loop, which is terminated by simply counting the passes, and after a subsequent 16 times shift, the channel number will be in the 32-bit register KANNUM.

Now, it is moved to the beginning of the data part in order to proceed with the counting of the data bits and the search for the pattern.

### 5.3.3. Working on the Data (Counting and Searching) Step

The counting here is very simple. Since we work bit by bit, the register COUNTER is incremented per loop. In this phase, by comparing the contents of FLENGTH with 0, it is checked whether the machine has reached the end of the field. The search is carried out with the aid of two registers (MDR and TEMP) and a constant 14 (binary: 1110), which are used to process the data word. First, the data word (in MDR) is copied to register TEMP. There the word is reduced to its four least significant bits ( $0^{\wedge}28@MDR_{3,0}$ ) and then compared with the pattern. A negative comparison results in the right-hand shift of the word in the register MDR, as well as the adaptation of COUNTER, FLENGTH and BITPOS. After that, the word is copied, cut and compared again. If BITPOS reaches the value 31, a new word must be retrieved from the memory. A positive comparison means that the beginning of the next packet has been found, and thus the channel number and the data length of the previous one must be stored. The next step is to jump in the memory to the address stored in the register ACCU (beginning of the table) and to start the search for adequate storage space for the information.

### 5.3.4. Saving Step

The quasi-read header is above the first bit of the first word of the memory table. After that, the channel number as a counter for 64 bit jumps in Memory is used. If the counter (the content of TEMP) is zero, the channel number is written at this point, and the data bit number in the following data word is added to the existing content of the word. Thus, after processing the field, e.g., the information for channel ten at the tenth place (in 64-bit increments) in the memory table.

### 5.3.5. The "Back-to-Field-Jumping" Step

Now, BITPOS is going from the memory table directly to the starting bit of the next channel number. The register MEMPOS contains the address of the data word, in which the last occurrence of pattern 1110 was found. This is incremented by 1 before reading the contents of this address into the MDR. At this time, BITPOS will be copied to TEMP. This register is used this time to shift the data word by the appropriate number of bits. After that, TEMP will be decremented per step and check for equality with 0. If this equality is given, the rest of the current memory word

is in the MDR, and the least significant bit is the first bit of the next Channel number (step 2).

#### Remarks:

- 1 End of a data word reached: In the case that  $BITPOS - 31 = 0$ , an address is preceded ( $MAR + 1$ ,  $MEMPOS + 1$ ) and BITPOS is reset to 0. After that, the algorithm continues in the appropriate place.
- 2 Unclean searching: When searching for the pattern, it can be canceled when the 28 bits of a data word is reached because then the inserted zeros are also searched with. This will be not done because there is no danger of mistakenly finding the pattern and eliminating the problem of looking at the beginning of counting, which BITPOS has for content (29 or 30 or 31). Therefore, it is 3 superfluous pattern comparisons per data word.

## 5.4. The Personalized Registers

**FLENGTH:** stores one-time the passed field length X and is counted down to determine when the field was 'expired'.

**BITPOS:** counts the bits in a data word and, together with MEMPOS, saves the current position in the field.

**MEMPOS:** specifies the address of the current data word.

**KANNUM:** stores the channel number of the current packet.

**TEMP:** reduces the contents of MDR to the four least significant bits.

**COUNTER:** adds up the data bits (pay attention that the channel number is 16 bits long and that it is shifted 16 digits in the 32-bit register).

## 5.5. Constants

- 0 to reset a register.
- 1 for incrementing and decrementing.
- 14 corresponds to the binary number 1110.
- 16 used in case of loop break when copying the channel number (see COUNTER).
- 31 BITPOS limit for channel number memory (when data word ends).
- Decimate 32 FLENGTH by scrolling one word at the same BITPOS.
- $2^{\wedge}31$  in binary notation (32 bit) 10..0 and is used to set an inserted bit.

### 5.6. The Machine Instructions

The machine instructions used are the instructions of the basic machine.

**ADD** ALUResult  $\leftarrow$  A+B

**SUB.B** ALUResult  $\leftarrow$  -A+B

**TRANS.A** ALUResult  $\leftarrow$  A

**TRANS.B** ALUResult  $\leftarrow$  B

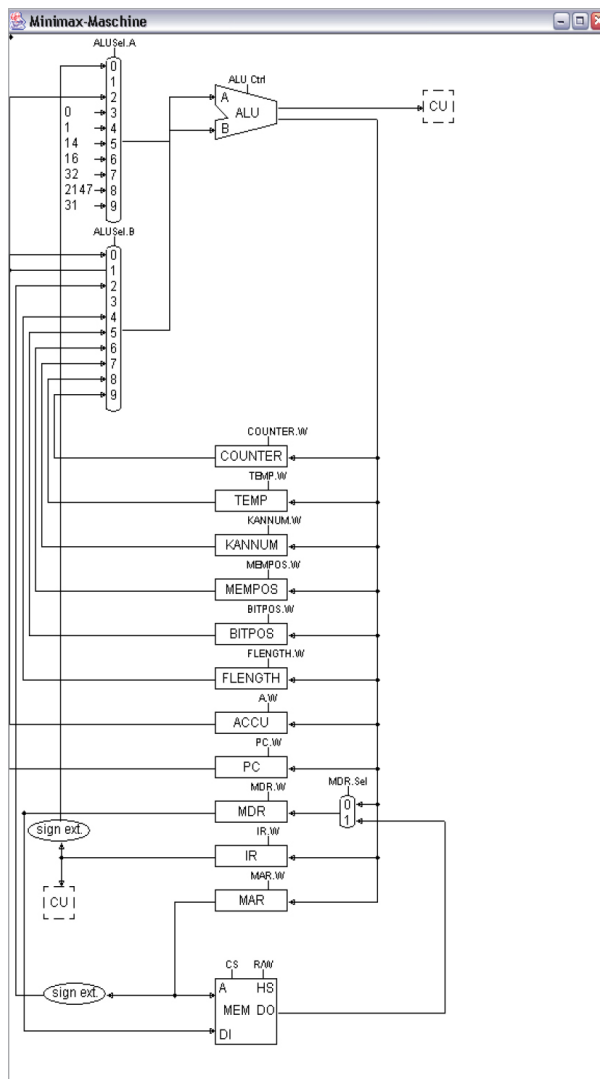
Additionally used ALU- operations:

**OR** A OR B

**AND** A AND B

**Figure 10**

The used (implemented) architecture



**DIV.A** ALUResult  $\leftarrow$  B DIV A

**SR.B** ALUResult  $\leftarrow$  0@B<sub>31:0</sub>

Figure 10 shows the used architecture. It is an extension of the Minimax machine so that an implementation of this developed architecture presents solutions to how the special communication protocol (Packet Analysis) introduced in this paper can be implemented.

### 5.7. Algorithm Flowchart

The flowchart of the presented algorithm is shown in Figure 11. It serves to give an overview of how the "Packet Analysis" algorithm works.

Here, the diagram will show a detailed description for each step of this algorithm (Initializing step, copying the channel number step, working on the data step, saving step, and "back-to-field-jumping" step).

**Figure 11**

The flow chart of the presented algorithm

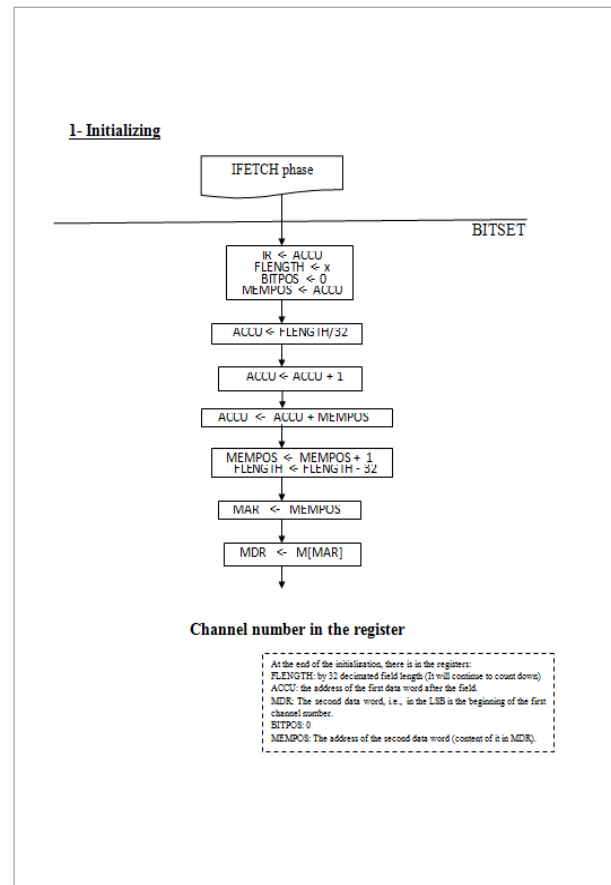
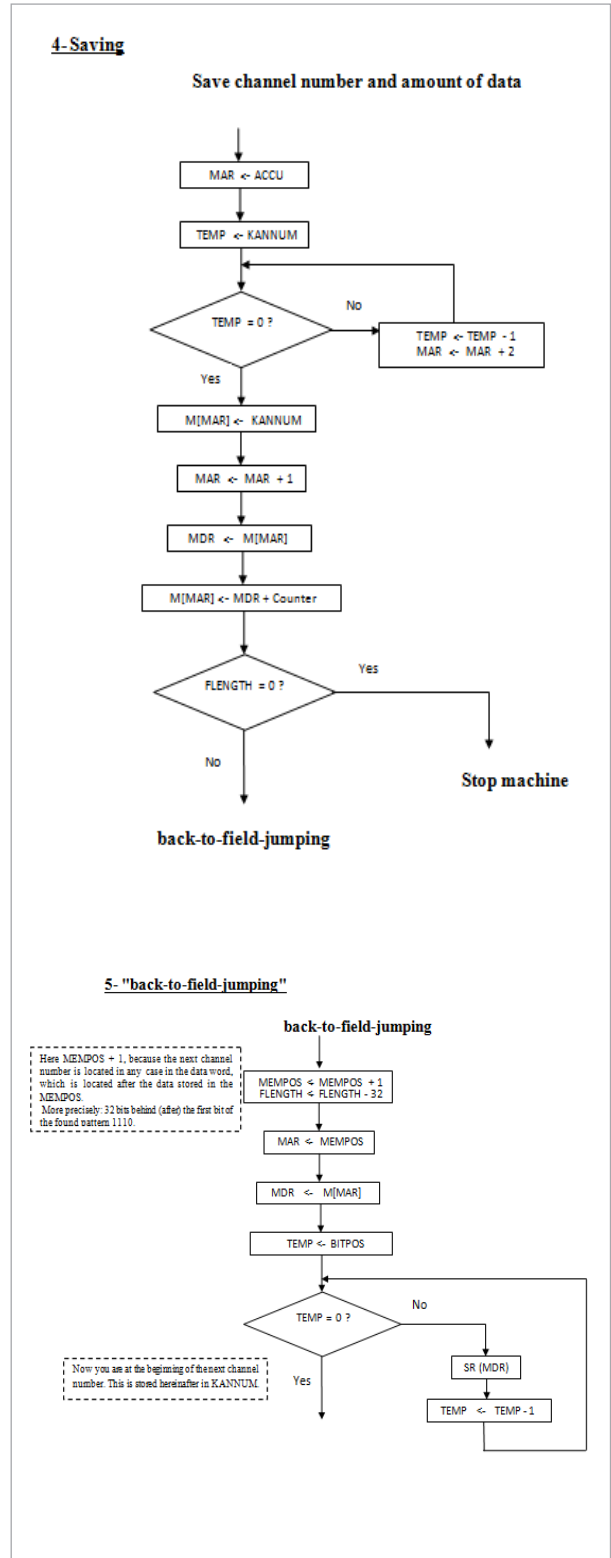
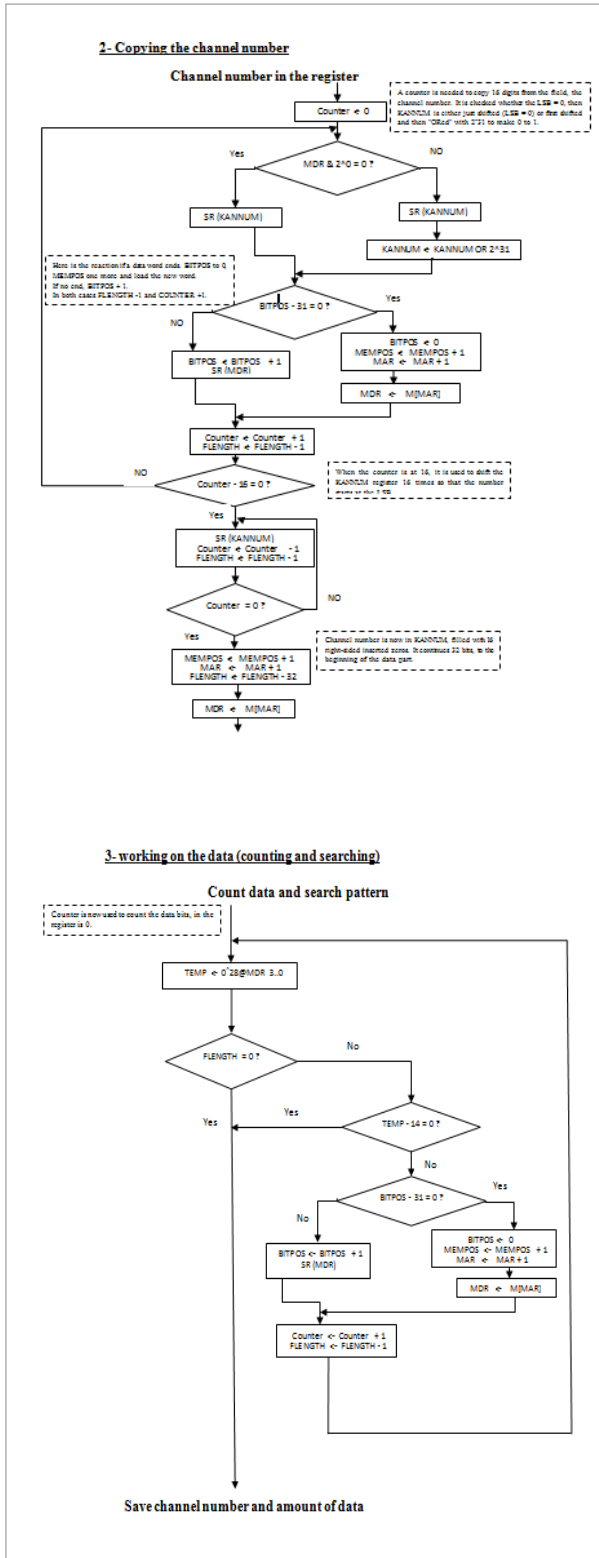


Figure 11 (continued)



## 6. Benchmark Results

This entire program must be implemented and tested for error-freeness (accuracy) with suitable input values. Here, an assessment of the implemented solution is needed. In this paper, two metrics defined by Chaaban [2] are used. Firstly, the execution time in clock cycles on the Minimax machine. These times have to be determined using given test-benches. Secondly, the (weighted) runtime and program length of the solution. These values must be calculated using a given suitable formula. In this context, two formulas were defined for these metrics to calculate  $t\_evaluated$  (runtime) and  $n\_evaluated$  (program length).

### 6.1. Evaluation of the Solution

- reg: number of supplemented Minimax registers
- se: number of supplemented sign extension units (0 or 1)
- const: number of added constants (one "0" and one "1" free)
- alu\_add: Penalty sum for all added ALU commands
- alu\_use: Penalty sum the used ALU commands in the program.

### 6.2. Runtime in Minimax-Clocks

The weighted runtime of the solution in Minimax-clocks should be calculated using the next formula [2]:

$$t\_evaluated = (t\_bench * (1 + 0.1 * reg + 0.15 * se + 0.015 * alu\_add + 0.05 * const))$$

In this formula,  $t\_bench$  is the execution speed, which is the number of the Minimax-clocks that is needed to complete the running of the solution (program).

In this paper, runtimes for the packet field size 1120 bytes are to be determined in three cases:

- 1- All packets have a length of (80) bits (i.e., packets without data).
- 2- Average data length of (80) bits.
- 3- Average data length of (144) bits.

### 6.3. Program Length (Length of the Algorithm)

The program length (length of the algorithm) of the

solution should be calculated using the next formula [2]:

$$n\_evaluated = n\_algorithm + 5 * reg + 10 * se + 3 * alu\_use + 5 * const$$

In this formula,  $n\_algorithm$  is the number of lines of the algorithm (solution).

### 6.4. Additional ALU Operations

The penalty of all added (alu\_add) and used (alu\_use) ALU commands (additional ALU operations) in the program can be taken from Table 5.

**Table 5**

Penalty of additional ALU operations

Additional ALU operations	SUB1	INC/DEC	S.L, S.R	AND, OR, NOT	XOR	DIV	Custom
Penalty (1..20)	1	4	5	6	8	10	Up to 20

## 7. Results and Discussion

In this section, some captures of simulation results (inputs and outputs) are presented and discussed. It aims to explain how packets are analyzed and processed by "packet analysis".

As mentioned early in this paper, the measurement of the implemented system performance aims to ensure solution robustness. Therefore, simulation outputs are collected and reported based on the formula mentioned above to detect if the implemented system robust, weakly robust, or maybe even not robust.

Table 6 shows the execution speeds (runtimes)  $t\_bench$  (simulation outputs) of the implemented algorithm (the micro-program) for the three cases mentioned above. Here, the packet field size was chosen 1120 bytes (simulation inputs) as it represents a suitable size to the memory size of the Minimax machine used in this paper.

After measuring  $t\_bench$ , the  $reg$ ,  $se$ ,  $alu\_add$ ,  $const$  have to be determined to calculate the  $t\_evaluated$  according to the formula mentioned above. Table 7 shows these values.

**Table 6**Runtimes ( $t_{bench}$ ) measured for three cases

Benchmark	$t_{bench}$ (runtime in Minimax-clocks)
All packets have a length of (80) bits (i.e., packets without data)	4775
Average data length of (80) bits	5760
Average data length of (144) bits	6730

**Table 7**Calculating the elements of  $t_{evaluated}$ 

Element	Number	Penalty
<b>Reg:</b> number of supplemented Minimax registers.	6 (FLENGTH, BITPOS, MEMPOS, KANNUM, TEMP, COUNTER)	$0.1*6$
<b>Se:</b> number of supplemented sign extension units (0 or 1)	1	$0.15*1$
<b>Const:</b> number of added constants (one "0" and one "1" free).	5 (14, 16, 31, 32, $2^{*31}$ )	$0.05*5$
<b>alu_add:</b> Penalty sum for all added ALU commands	4 (OR, AND, DIVA, SR.B)	$0.015*(6+6+10+5)$

Based on both Tables 6 and 7, the  $t_{evaluated}$  can be calculated for the three cases, as mentioned above. Table 8 shows the results of  $t_{evaluated}$ .

**Table 8**Calculating  $t_{evaluated}$  in three cases

Benchmark	$t_{evaluated}$
Case 1: no data (0 bits)	11483
Case 2: data (80 bits)	13852
Case 2: data (144 bits)	16185

After counting  $n_{algorithm}$ , which is the number of lines of the algorithm (solution), the  $reg$ ,  $se$ ,  $alu_{use}$ ,  $const$  have to be determined to calculate the  $n_{evaluated}$  according to the formula mentioned above. Table 9 shows these values.

**Table 9**Calculating the elements of  $n_{evaluated}$ 

Element	Number	Penalty
<b>Reg</b>	6	$5*6$
<b>Se</b>	1	$10*1$
<b>Const</b>	5	$5*5$
<b>alu_use:</b> Penalty sum the used ALU commands in the program.	OR: 5 times AND: 7 times DIVA: 2 times SR.B: 3 times	$3*(5*6+7*6+2*10+3*5)$
<b>n_algorithm:</b> is the number of lines of the algorithm (solution).	75	75

Based on Table 9, the  $n_{evaluated}$  can be calculated for the three cases as mentioned above. Table 10 shows the results of  $n_{evaluated}$  (same value for the three cases).

**Table 10**Calculating  $n_{evaluated}$  in three cases

Benchmark	$n_{evaluated}$
Case 1: no data (0 bits)	461
Case 2: data (80 bits)	461
Case 2: data (144 bits)	461

A comparison with other model implementations of this special communication protocol is useful. By making this comparison, it could be noticed that there is a lot of scope for optimization in the solution presented in this paper.

It is clear that the machine operations, which are executed in a continuous loop (e.g., increment and decrement counters), have to be minimized. Similarly, the loop itself has to be avoided by using more so-

phisticated algorithms. Furthermore, constants can be saved by calculating the required numerical values from a few integrated constants at runtime. This was not done here to avoid further registries.

A comprehensive revision of the presented solution is to be recommended, because unfortunately, it could not be started directly with the optimal implementation, as in most cases. Thus, this paper presents a fully functional but not optimal solution, which implements a special communication protocol.

## 8. Conclusion

Communication protocols can be implemented in different manners. That may be in the system software layer or the hardware layer. In "Protocol Engineering" research, it can be noticed that the implementation phase of communication protocols is not considered as much as other phases of the protocol development process.

In this paper, the implementation of a special communication protocol called "Packet Analysis" was introduced. The focus was the study of designing a novel methodology for defining a model implementation of communication protocols using micro-programming. This methodology is an applicable approach, in which various concepts from different research areas were integrated.

Based on that, an algorithm and its flow chart were introduced. It explains all working phases of the "Packet Analysis" algorithm in detail. This algorithm simulates the communication protocol using micro-programming.

Consequently, the development and the evaluation of the entire system of this research paper were made using different metrics of system performance. In this regard, two formal quantitative measures and metrics were used to evaluate the implemented protocol/solution. These metrics were the execution speed and program length. Here, different given test files (test-benches) were used as input for the simulator.

Additionally, the analysis of the results led to the conclusion that an optimization of the presented solution can be achieved.

Finally, after the conclusion of this paper was drawn, a peek at future trends for follow-up research papers can be given.

In this paper, the idea of designing and implementing of special hardware architectures was introduced. It measured and reported the implemented solution/system performance based on standards. Therefore, the next step is to continue with the optimization process of the presented solution/algorithm so that its results will be more trustworthy for research.

## References

1. Bochmann, G., Gerber, G., Serre, J. Semiautomatic Implementation of Communication Protocols. *IEEE Transactions on Software Engineering*, 1987, 13(9), 989-1000. <https://doi.org/10.1109/TSE.1987.233521>
2. Chaaban, Y. A Simulator for Micro Programming and Hardware Simulation Integrated in a Computer Hardware Project. *Journal of Engineering Sciences and Information Technology*. Arab Journal of Sciences & Research Publishing, 2020, 4(1), 107-121. DOI: 10.26389/AJSRP.Y201019.
3. Csofaki G., Horváth G. A., Kovács G. Communication Protocol Implementation in Java. In: *International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS 2000)*. Lecture Notes in Computer Science, (LNCS, volume 1905), Springer, Berlin, Heidelberg, 2000, 254-265. [https://doi.org/10.1007/3-540-40002-8\\_24](https://doi.org/10.1007/3-540-40002-8_24)
4. Department of Systems and Computer Architecture, Leibniz University Hannover, Germany. Available from: [https://www.sra.uni-hannover.de/Lehre/WS19/L\\_MMM/](https://www.sra.uni-hannover.de/Lehre/WS19/L_MMM/). Accessed on December 3, 2019.
5. DGH Corp. Modbus RTU Protocol Overview, 2015. <https://www.dghcorp.com/modbus/modbusrtuovr.asp>. Accessed on December 10, 2019.
6. George, B., Raja, T. Implementation of Communication Protocols in FPGA for Testing of Microcontrollers. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2015, 4(7), 6001-6009. <https://doi.org/10.15662/ijareeie.2015.0407026>
7. Hennessy, J., Patterson, D. *The 5th Edition of Computer Architecture: A Quantitative Approach*. ISBN: 978-0-12-383872-8. Elsevier, 2012. Available from: [https://www.bau.edu.jo/UserPortal/UserProfile/PostsAttach/66220\\_4281\\_1.pdf](https://www.bau.edu.jo/UserPortal/UserProfile/PostsAttach/66220_4281_1.pdf)



8. Hoffmann, M. G. Hardware Implementation of Communication Protocols: A Formal Approach. In Proceedings of the 7th annual symposium on Computer Architecture (ISCA '80). Association for Computing Machinery, New York, NY, USA, 1980, 253-263. <https://doi.org/10.1145/800053.801933>
9. Ideguchi, T., Mizuno, T., Matsunaga, H. Automatic Implementation of Communication Protocols. ACM SIGCOMM Computer Communication Review, 1981, 11(1), 40-56. <https://doi.org/10.1145/1040114.1040117>
10. Irshad, A., Chaudhry, S. A., Shafiq, M., Usman, M., Asif, M., Ali, S., Kumari, S. An Improved Biometric Multi-Server Authentication Scheme for Chang et al.'s Protocol. Information Technology and Control, 2019, 48(2), 211-224. <https://doi.org/10.5755/j01.itc.48.2.17417>
11. Jaloudi, S. Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study. Future Internet 2019, 11(3), 66. DOI:10.3390/fi11030066 <https://doi.org/10.3390/fi11030066>
12. Jun, X., Yan-Jun F. Profibus Automation Technology and Its Application in DP Slave Development. Proceedings of IEEE International Conference on Information Acquisition, Hefei, 2004, 155-159. <https://doi.org/10.1109/ICIA.2004.1373430>
13. Kodinov, G. MySQL Server Blog, News from the MySQL Server Team. Track and Optimize Server Connection Methods, 2015. <https://mysqlservertime.com/track-and-optimize-server-connection-methods/>. Accessed on December 19, 2019.
14. König, H. Protocol Engineering. Springer, Berlin-Heidelberg, 2012. <https://doi.org/10.1007/978-3-642-29145-6>
15. Lee, C. C., Chang, R. X., Chen, T. Y., Chen, L. An Improved Delegation-Based Authentication Protocol for PCSs. Information Technology and Control, 2012, 41(3), 258-267. <https://doi.org/10.5755/j01.itc.41.3.857>
16. Merlin, P. A Methodology for the Design and Implementation of Communication Protocols. IEEE Transactions on Communications, 1976, 24(6), 614-621. <https://doi.org/10.1109/TCOM.1976.1093347>
17. MODICON. Modicon Modbus Protocol Reference Guide. MODICON, Inc., Industrial Automation Systems, June, 1996.
18. Mostia, W. Introduction to Modbus: Why the Grandfather of Modern Fieldbus is Still on the Job After 40 Years. In Control Global, 2018, 31(12), 35-39.
19. Nexsens Technology. Sensor Communication Protocols. <https://www.nexsens.com/knowledge-base/x2-data-logger/sensor-interface-x2-data-logger/sensor-communication-protocols.htm>. Accessed on December 15, 2019.
20. Paul, D., Stefan, H., Carsten, P. MySQL 5.0 Certification Study Guide (MySQL Press), MySQL Press, 2005.
21. Pereira, C., Neumann, P. Industrial Communication Protocols. Chapter In: Springer Handbook of Automation, Springer, Berlin-Heidelberg, 2009, 981-999. [https://doi.org/10.1007/978-3-540-78831-7\\_56](https://doi.org/10.1007/978-3-540-78831-7_56)
22. PROFIBUS. Comprehensive Protocol Overview. Real Time Automation, Inc. <https://www.rtautomation.com/technologies/profibus/>. Accessed on December 1, 2019.
23. Río, T., Moreno, L., Gracia, A. Implementation of the Communication Protocols SPI and I2C Using a FPGA by the HDL-Verilog Language. Research in Computing Science, 2014, 75(1), 31-41. <https://doi.org/10.13053/rcs-75-1-3>
24. Sommerville, I. Software Engineering. Addison-Wesley, Reading, 2000.
25. Truman, T. E., Brodersen, R. W. A Methodology for the Design and Implementation of Communication Protocols for Embedded Wireless Systems. Ph.D. Dissertation. University of California, Berkeley, 1998. ISBN: 978-0-591-99366-0.

