The Journal of Supercomputing

Efficient Design and Hardware Implementation of the OpenFlow v1.3 Switch on the Virtex-6 FPGA ML605

--Manuscript Draft--

Manuscript Number:	SUPE-D-17-00317
Full Title:	Efficient Design and Hardware Implementation of the OpenFlow v1.3 Switch on the Virtex-6 FPGA ML605
Article Type:	Manuscript
Keywords:	OpenFlow switch; Software-defined network (SDN); FPGA; OpenFlow Protocol.
Abstract:	Software defined network (SDN) has had the evolution of the current network with the aim of removing its restrictions so that the data plane has been separated from its control plane. In the architecture of the SDN, the most controversial device is the OpenFlow Switch in that in the OpenFlow Switch it is packets which are processed and investigated. Now, OpenFlow Switch versions 1.0 and 1.1 have been implemented on hardware platforms and support Limited specifications of the OpenFlow. The present article is to design and implement the architecture of the OpenFlow v1.3 Switch on the Virtex®-6 FPGA ML605 board because the FPGA platform has high flexibility, processing speed, and re-programmability. Although little research investigated performance parameters of the OpenFlow Switch, in the present study, the OpenFlow system (switch and controller) is to be implemented on the FPGA via the VHDL Language on the one hand, and performance parameters of the OpenFlow Switch and its stimulation performance is to be investigated via the ISE Design Suite. In addition to enjoying high flexibility, this architecture has a consumer hardware at the level of other startups. It supports features of the OpenFlow v 1.3 for increasing speed and packet pipeline processing in flow tables switch are used. Its parser supports 40 packet headers in the network and provides the possibility of switch development for next versions of the OpenFlow as easily as possible.

Efficient Design and Hardware Implementation of the OpenFlow v1.3 Switch on the Virtex-6 FPGA ML605

Abbas Yazdinejad
Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran
abbasyazdinejad@eng.ui.ac.ir

Ali Bohlooli
Faculty of Computer Engineering, University of Isfahan Isfahan, Iran
bohlooli@eng.ui.ac.ir

Kamal Jamshidi
Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran
jamshidi@eng.ui.ac.ir

Abstract

Software defined network (SDN) has had the evolution of the current network with the aim of removing its restrictions so that the data plane has been separated from its control plane. In the architecture of the SDN, the most controversial device is the OpenFlow Switch in that in the OpenFlow Switch it is packets which are processed and investigated. Now, OpenFlow Switch versions 1.0 and 1.1 have been implemented on hardware platforms and support Limited specifications of the OpenFlow. The present article is to design and implement the architecture of the OpenFlow v1.3 Switch on the Virtex®-6 FPGA ML605 board because the FPGA platform has high flexibility, processing speed, and re-programmability. Although little research investigated performance parameters of the OpenFlow Switch, in the present study, the OpenFlow system (switch and controller) is to be implemented on the FPGA via the VHDL Language on the one hand, and performance parameters of the OpenFlow Switch and its stimulation performance is to be investigated via the ISE Design Suite. In addition to enjoying high flexibility, this architecture has a consumer hardware at the level of other startups. It supports features of the OpenFlow v 1.3 for increasing speed and packet pipeline processing in flow tables switch are used. Its parser supports 40 packet headers in the network and provides the possibility of switch development for next versions of the OpenFlow as easily as possible.

Keywords: OpenFlow switch; Software-defined network (SDN); FPGA; OpenFlow Protocol.

1. Introduction

Today the issue of computer networks has astonished the whole world so that the Internet, as the largest network fabric of the world, has penetrated in computers, houses, shops, universities, companies, and public sectors, and made an unbreakable link with technology, development, and advancement. Large hardware companies are consistently producing network chips, switches, routers and other network devices because the data transfer rate is increasing and users' needs and demands are being added day by day. However, the architecture of the current network cannot fulfil needs of corporates, telecom operators, and users. The increase in the number of mobile devices, development in Cloud services, and stability of bandwidth have caused the occurrence of complicatedness in the network architecture. All in all, the current network architecture suffers from constraints such as complexity, inconsistent policies, stagnation, dependence on developers, lack of scalability and programmability resulting in users' dissatisfaction with the network architecture [1]. The evolution of devices and equipment of the network, virtualization of servers, the advent of could services, changes in traffic patterns, etc. have resulted in revision of the current network architecture [1, 2]. In addition, developers of the current network are dissatisfied in that they do not have the possibility of testing new ideas and protocols in the Internet environment; therefore, the issue of innovation in the network is felt. At this time, a new architecture of the network, called the SDN architecture, was presented in which the traffic transfer is separate and is directly programed [2]. The SDN architecture was developed by the ONF which is a private organization [1]. The ONF consists of a series of firms responsible for standardizing, controlling, and granting standards for the SDN architecture and the OpenFlow to all corporates [3] the issue of infrastructures and equipment of the network is a vital one because the current network is static and unprogrammable. Recently, a concept called the SDN has been presented with programmability in the network. In general, the SDN has evolved the current network with the aim of removing its limitations.

In the SDN architecture, the most controversial device is switch, more specifically called the OpenFlow Switch. During the years that the network has been working, the speed of switches has significantly increased. In other words, all advances in networks are abed on the network switch. A switch has always had the key role in the network. All in all, hardware and virtual switches plays significant and basic roles in the SDN architecture in that they are directly responsible for implementing tables scheduled by the controller. Virtual switches such as OpenFlow Switch are placed on the edge of the network, a place for installing and adding software and control functionality of the network. Hardware switches of the network focus on Fast data transfer and traffic. They have simpler run configuration than virtual switches [4].

The OpenFlow is the first standard communication interface in the SDN architecture which is defined between the control and forwarding layers. The OpenFlow is a protocol of the SDN which enjoys flexibility in the network management and attracted all attentions to itself. The OpenFlow protocol make the links between the data plane and control plane in the switch and allows network administrators to change and develop algorithms for controlling flows and packets at the data plane [1].

One of the advantages of the OpenFlow for researchers and developers is presenting new services to users' demands in network equipment [5]. With a close attention to network equipment in the SDN architecture, particularly the OpenFlow Switch, on can observe multiple implementations of the OpenFlow Switch on hardware and software platforms. The OpenWRT and Linux are implementations on the software platform and FPGA, ASIC, and NetFPGA are implementations on the hardware platform. Implementation on the FPGA platform enjoys higher accessibility and comprehensiveness than other platforms such as the ASIC which is special-purpose. The FPGA-based switches have open-source hardware and enjoy more speed than hardware switches. In addition, the FPGA enjoys high flexibility, re-programmability, and processing speed and provides the possibility of project development to researchers. According to multiple implementations of the OpenFlow Switch on the FPGA, little research has investigated performance and evaluation of the OpenFlow Switch. In the present study, the stimulation and implementation of the designed architecture of the OpenFlow v 1.3 Switch are investigated. Moreover, the present study is to investigate performance parameters of the OpenFlow Switch and compare it with other evaluated performances. The data plane and control plane of the OpenFlow Switch are implemented on the FPGA platform. The hardware description language, i.e. the VHDL, and Xilinx ISE 14.7 the design suite is used in designing the OpenFlow v 1.3 Switch.

2. Background and related works

2.1. The SDN architecture

The architecture of the current network is static and unprogrammable in such a way that it prevents designers of the network to add new services to the network architecture. The current network suffers from a lot of limitations such as complexity, inconsistent policies, stagnation, dependence on developers, lack of scalability and programmability [1, 4]. The new architecture, i.e. the SDN architecture, has been developed with the aim of removing the restrictions. It has received a lot of attentions from academic and industrial environments in that it has enabled researchers and developers of the network to test their ideas. The SDN is not a software/hardware product or concept, but it is a new architecture and approach for more flexibility and controllability of the network, applications, and services. The following cases are some main features of the SDN architecture [1, 3, 4, and 6].

1. Separation of the data plane from the control plane;
2. Logical centralization of the network and an integrated overview of network;
3. The possibility of development of different applications;
4. Programmability of the data plane for more flexibility of the network; and
5. Rapid innovation in the network.

For more information about the SDN architecture and related features, see [1, 4, 7-9].

2.2. The OpenFlow architecture

The first standard for the SDN was the OpenFlow. Because of having features such as flexibility, the OpenFlow has more popularity in academic and industrial environments than other protocols such as IP and Ethernet. All in all, using the OpenFlow has a lot of advantages in network virtualization and routing distribution [10]. The OpenFlow protocol makes the link between the OpenFlow Switch and controller via the Secure Socket Layer (SSL). The OpenFlow Switch basically consists of tables called flow tables. The controller's commands to those tables are performed via the OpenFlow protocol [11]. A profile of the OpenFlow Switch is observed in figure 1. This figure, the controller is connected to the OpenFlow Switch via the SSL. It modifies and updates flow tables of the switch via the OpenFlow protocol [11, 12].

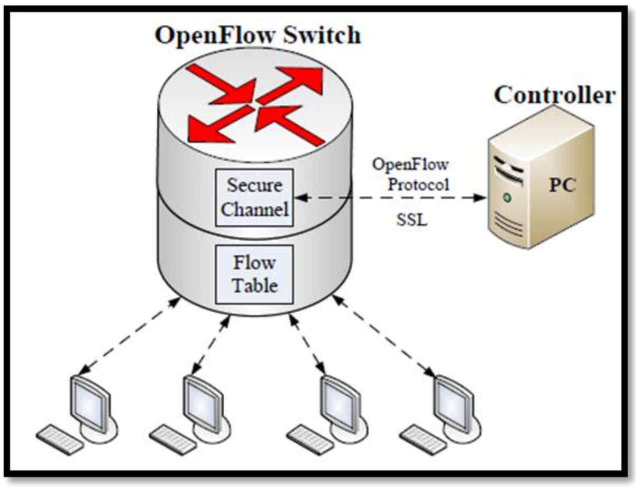


Figure 1: An OpenFlow Switch [13]

The controller is basically the main intelligence of the network and acts as a network operating system. In fact, the OpenFlow controller takes the responsibility of controlling the network hardware and facilitates network management [14]. More information about the OpenFlow controller can be searched in the OpenFlow standard and other resources such as [15-17].

The OpenFlow Switch is one of the key components of the SDN architecture. After entering the OpenFlow Switch, packets are investigated in order that necessary actions can be applied on them. As observed in figure 2, the switch consists of three basic components of the OpenFlow Table, OpenFlow Protocol, and Source Channel [11, 12]. The control plane in the switch is responsible for decision making, policy making, while the data plane has the duty of traffic transfer.

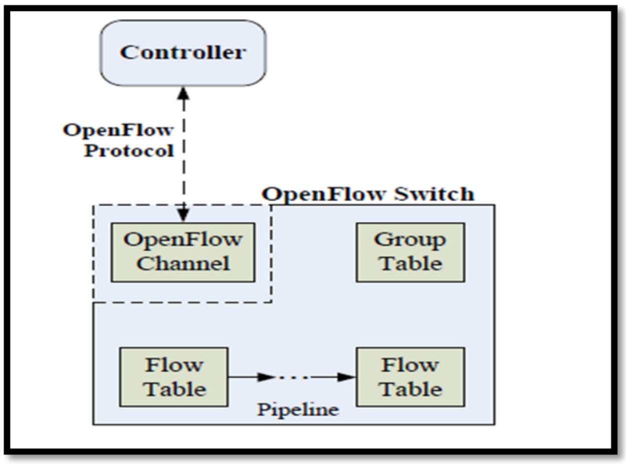


Figure 2: Basic components of the OpenFlow switch [18]

2.3. Review of related works

Currently, the OpenFlow Switch has been implemented on hardware platforms such as NetFPGA, ASIC, FPGA, and software ones such as Linux and OpenWRT. NEC IP8800, HP PROCURVE, FM 6000 Intel are some of the commercialized OpenFlow Switches.

One of the studies related to the subject is Noaus [19] investigating the implementation of the OpenFlow Switch on the NetFPGA platform. In this implementation, a few packet header fields were considered for being matched in the flow tables of the switch. There was a high latency in entering a new flow into the OpenFlow Switch. This is considered as a challenge. Among other implementations, one can refer to [20] indicating the implementation of the DMP (Distributed Multimedia Plays) in the network nodes. It showed less latency and higher scalability on the FPGA platform. Bosshart et al. [21] presented the RMT architecture of the OpenFlow Switch on the ASIC in order to promote the OpenFlow model. Developers of this architecture stated that their project is limited because of stages of the pipeline, action unit, and match tables on the ASIC. Khan [22] implemented the OpenFlow Switch v 1.0 on platforms such as the NetFPGA, ML605, and DE4 by a high-level hardware description language (BSV). It supports a few features of the OpenFlow and a small number of packet headers (12 headers) in the network. Li [23] investigated the implementation of the OpenFlow 1.1 Switch on the ML605FPGA board. This switch supports limited features of the OpenFlow, the controller, and a number of packet headers (15 headers). In that study, the architecture of the OpenFlow v 1.3 Switch was implemented on the FPGA via the VHDL language. In addition to enjoying high flexibility, sustainable hardware consumption, and supporting the features of the OpenFlow v 1.3 Switch, it works for increasing the speed and processing of pipelines of the flow tables of the switch whose parser supports 40 packet heads and easily provides the possibility of switch development to next versions of the OpenFlow. Performance parameters are evaluated and compared to other actions.

3. Architecture of the designed OpenFlow v 1.3 Switch

Figure 3 indicates the block architecture of the OpenFlow v 1.3 Switch. The switch architecture consists of modules of FIFO ‹Packet-Buffer, Packet Processing, Execute Action Set, Flow Table, Packet Forwarding, and Controller OpenFlow.

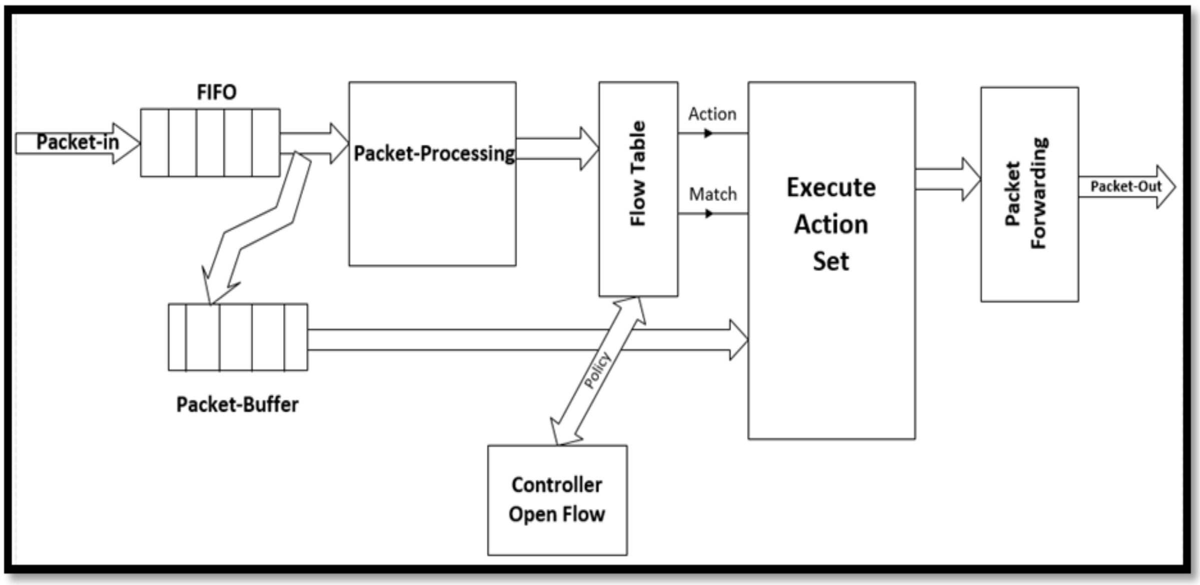


Figure 3 the designed block architecture of the OpenFlow v 1.3 Switch

Each block has its own performance and role in the switch performance. When packets are read in the input ports and enter the switch, they are put in the FIFO queue. After that the packet comes into the FIFO queue, they will be copied in to the queue of Packet-Buffer and Packet Processing modules. In the Packet Processing module, processing is executed on packets because the data of the input packets are important for the switch. In addition, some fields of the packet header must be dropped so as to a series of data can be generated for the Flow Table and the match action. At the same time, packets existing in the Packet-Buffer module comes into the Execute Action Set. In the Flow Table module, the match action of the fields of packet headers and data extracted from the packet are compared with entries of match tables in a pipeline form. In fact, this module works in a pipeline form and the high processing speed in the process of matching flows with entries of the flow tables was considered. After matching fields, data such as action and matched fields are forwarded to the Execute Action Set so as to the desired action can be applied on the packet for those packets coming into the Packet-Buffer module. If fields extracted from the packet are not matched with entries of the flow tables in the Flow Table module, a request is sent to the Controller OpenFlow in order that a decision can be made for the packet. This decision may be a command to delete the packet or update flow tables of the switch. The algorithm of packets entering the OpenFlow v 1.3 Switch is illustrated in a flowchart in figure 4.

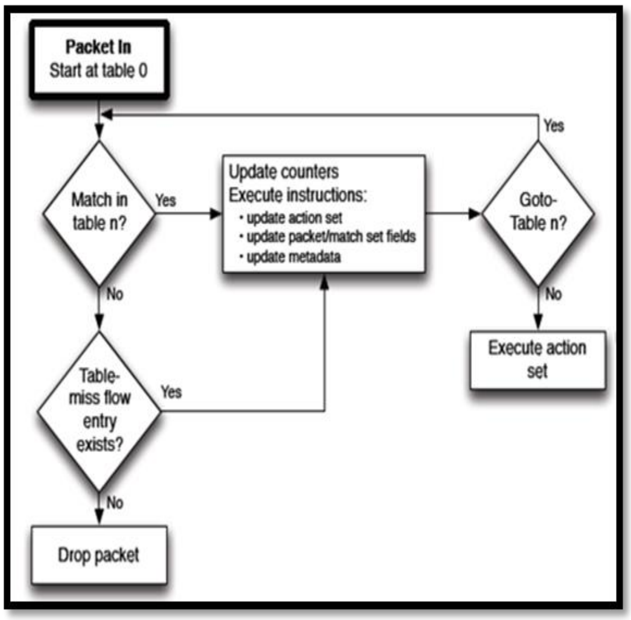


Figure 4: the flowchart of the entrance of packets into OpenFlow v 1.3 Switch

3.1. Investigating the components of the designed OpenFlow V 1.3 Switch

The FIFO queue and the Packet-Buffer module are generated by IP Core, FIFO Generate [24], and Block RAM [25] which are design suites of the Xilinx as well as ISE 14.7 [26]. These two modules plays the role of the Input queue for receiving packets and buffering input packets for using in other parts of the switch. The size of the FIFO queue and Packet-Buffer is as 64 With*1024 Depth.

Packet processing: this module receives packets entering the switch in order that processing can be executed on packets. In fact this part identifies and extracts packet headers and other fields. This part plays the vital role in the switch. These modules, composed of parts such as parser and composer, are illustrated in figure 5.

Parser: it plays the role of identifying and analyzing packet headers and finally extracting the identified fields from the packet. This module supports 40 different packet headers in the network.

Composer: this part is responsible for organizing data and fields extracted from the parser. The composer receives extracted fields and data and changes them into formats appropriate for the flow tables of the switch. Then, it forwarded them to the Flow Table module in the specified time and place.

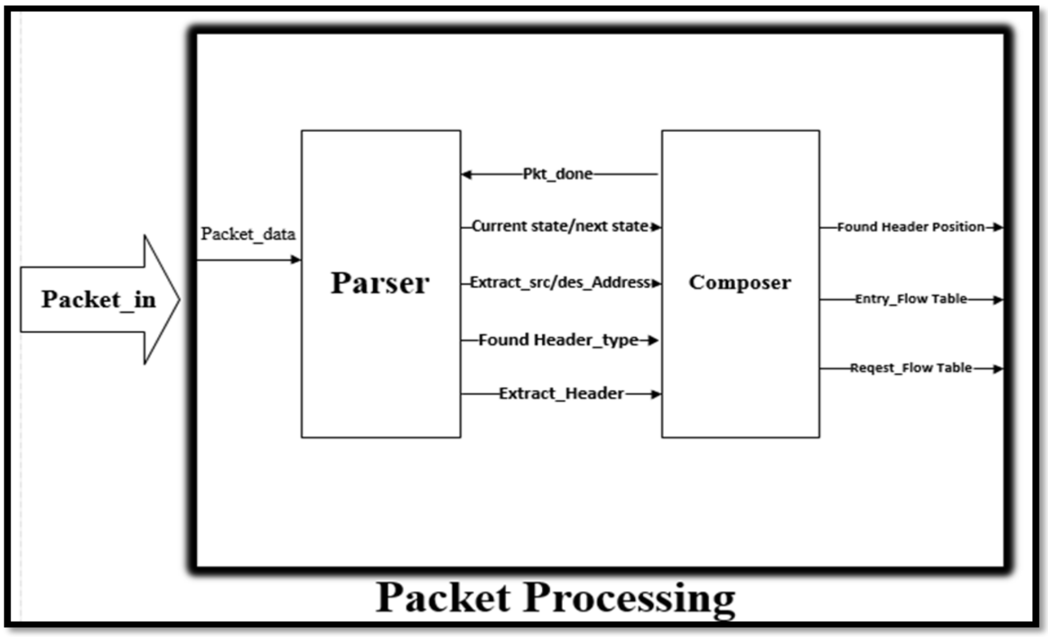


Figure 5: the packet Processing module

Flow table: this module, locating after the Packet Processing module, consists of flow tables of the switch which include flows using those tables for matching packet headers and fields. In addition to including flow tables, this part consists of group flow tables as well. In designing this module, matching packet fields was implemented in the pipeline form and high processing speed was the main issue. In figure 6, main components of this module are exact match table, wildcard match table, and action and flow entries are saved in the exact match table; flow mask in the wildcard match table, and data related to actions of each packet are saved in the action part. Generally, comparing flows and applying controller policies are conducted in the Flow Table module.

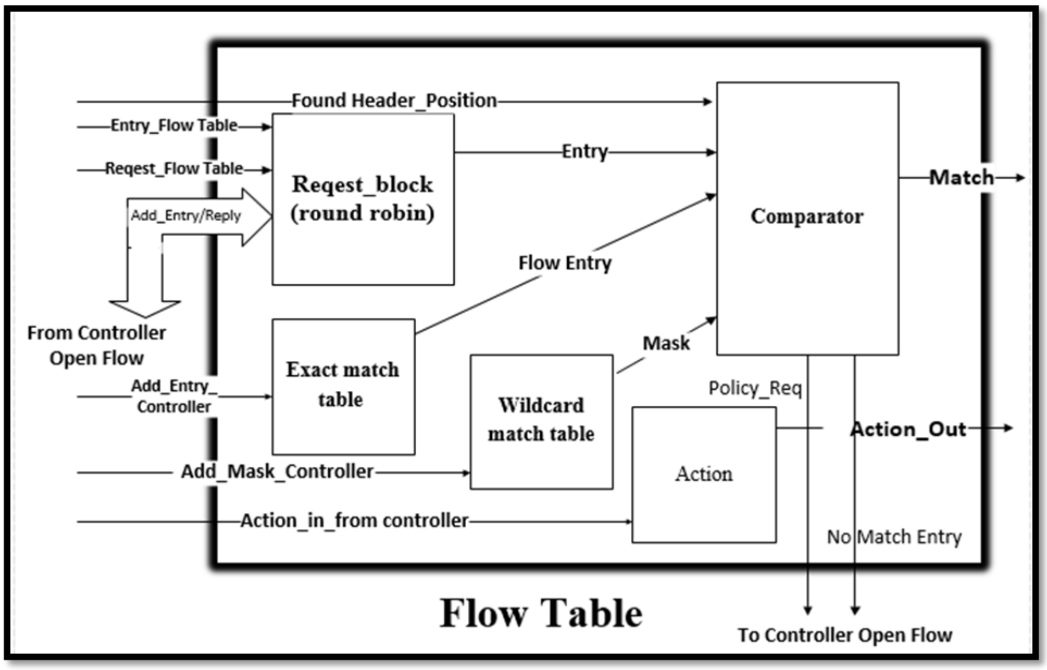


Figure 6: the Flow Table Module

Execute Action Set: after being buffered packets are directly forwarded to this module in the entry in order that actions can be applied to them. In fact, this module is located after the Flow Table and after match operation, the Execute Action Set takes the responsibility of selecting the forwarded port, updating packet headers, and identifying the packet length based on the data extracted from the Flow Table module. It also has features of the OpenFlow Switch. It finally forwards the packet to the Packet Forwarding unit based on the related Action and the output port number.

Packet Forwarding: this module forwards packets on a certain output port based on the identified Action and determined output port in the Execute Action Set. The Action includes data such as the Output Port and Action Flag.

Controller: this part is in fact the network intelligence and has the responsibility of applying necessary policies for the Flow Table switch. This action is executed via forwarding principles and rules to flow tables of the switch. In general, this module is responsible for managing flow tables and handling the received requests from the switch.

3.2. The features attributed to the design of the OpenFlow v 1.3 Switch

In the present study, the architecture of the OpenFlow Switch was designed and implemented on the hardware platform of the ML605 board with the FPGA Vertix6 Xilinx. In addition to flexibility the hardware efficiency and high speed are fully compatible with the SDN and does not support the standard features of the OpenFlow [12].

Due to the re-configurability that the FPGA provides for us, it provides the possibility changes and development in the switch. So far, several versions of the OpenFlow have been published. Table 2 compares some of those versions and their features [12, 27-29]. Compared to other versions (v 1.0 and v. 1.1), the architecture of this switch supports the features of the OpenFlow v 1.3 Switch. The parser of this switch supports 40 packet header fields in the Packet-Processing module, while in studies [22 and 23], it parses 12 and 15 packet header fields.

Table 1: features of different versions of the OpenFlow

Versions of OpenFlow	Release date	Number of headers	Features
1,0 OF	Dec 2009	12	Supporting a flow table, Matching in the fixed, portfolio, IPv4
1,1 OF	Feb 2011	15	Multiple tables,Supporting MPLS,VLAN, unicast-multicast
1,2 OF	Dec 2011	36	Multiple controllers, IPV6
1,3 OF	Jun 2012	40	Multiple parallel channels between the switch and controller, Supporting 802.1ah PBB
1,4 OF	Oct 2013	41	Supporting optical ports Gradual development and evolution

The OpenFlow Switch consists of a set of flow tables containing some entries. With regard to the features of the OpenFlow, the number of those entries are increased in the version 1.3 compared to the version 1.1. Table 1 illustrates the structure of Flow Table entries in the version 1.3 [12].

Table 2: entries of the flow table of the OpenFlow v 1.3

Match field	Priority	Counters	Instruction	Timeouts	Cookie

Match field: it determines the state of a packet in certain conditions. This field is composed of packet headers, input ports, Ethernet, IPV4, IPV6, and high-level ports.

Priority: it refers to prioritization in matching flows

Counters: it is matched with statistics of packet fields and keep inputs of the flow table in the switch.

Instruction: it determines an action which must be executed on match packets.

Timeouts: they refer to the maximum of time that a flow is outdated due to the passage of time.

Cookie: it refers to the unique value of data selected by the controller. It may be used by the controller for filtering, updating, or deleting the flow.

In the Flow Table module of the OpenFlow v 1.3 Switch, 13 fields are matched. Match fields consist of fields layer 2 to 4. These fields are indicated in table 3 [12].

Table 3: fields matched in the flow table of the OpenFlow v 1.3 Switch

Field	Description
IN_PORT	Ingress port. This may be a physical or switch defined logical port.
ETH_DST	Ethernet destination address. Can use arbitrary bitmask.
ETH_SRC	Ethernet source address. Can use arbitrary bitmask.
ETH_TYPE	Ethernet type of the OpenFlow packet payload, after VLAN tags.
IP_PROTO	IPV4 or IPV6 Protocol number
IPV4_SRC	IPV4 source address. can use subnet mask or arbitrary bitmask
IPV4_DST	IPV4 destination address. can use subnet mask or arbitrary bitmask
IPV6_SRC	IPV6 source address. can use subnet mask or arbitrary bitmask
IPV6_DST	IPV6 destination address. can use subnet mask or arbitrary bitmask
TCP_SRC	TCP source port
TCP_DST	TCP destination port
UDP_SRC	UDP source port
UDP_DST	UDP destination port

The pipeline processing in the Flow Table module is another feature considered in the architecture of the designed switch. This module has a series of routing flow tables. Each table consists of a series of flow entries including a series of fields. Several entries may be used for determining the packet states. That entry with the most packet states is used for matching and its reaction is accepted. In figure 7, the pipeline processing is observed. It provides the possibility of configuration and flexibility of flow tables.

There is a set of tables in the pipeline. All packets entering the switch must investigate table 0 for matching its fields with the entries of the table which are concurrently forwarded to tables with higher numbers in order that they construct a match table. The action related to the packet in the table with the highest match is accepted. If the packet is not matched with none of the tables, it is forwarded to the controller which updates tables or deletes the packet. The Group Table, added to the features of the OpenFlow 1.3, is used for a number of entries of the flow table to which an output instruction set is applied.

4. ` Stimulation of the architecture of the OpenFlow v 1.3 Switch

In this section, stimulation of the architecture of the OpenFlow v 1.3 Switch. To investigate the accuracy of the performance and accuracy of the results obtained from stimulation of this architecture, the Xilinx ISE 14.7 was employed.

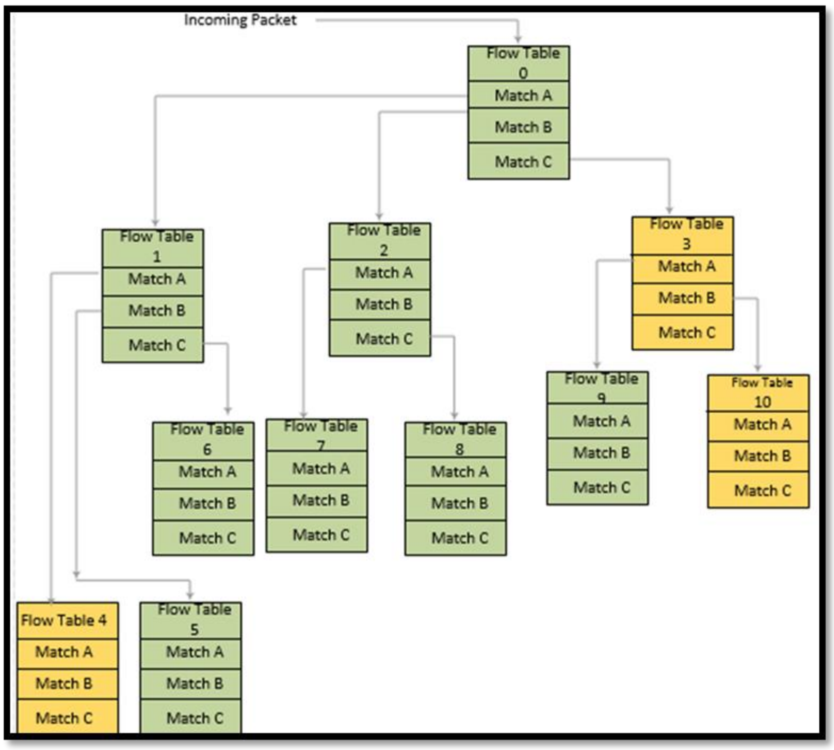
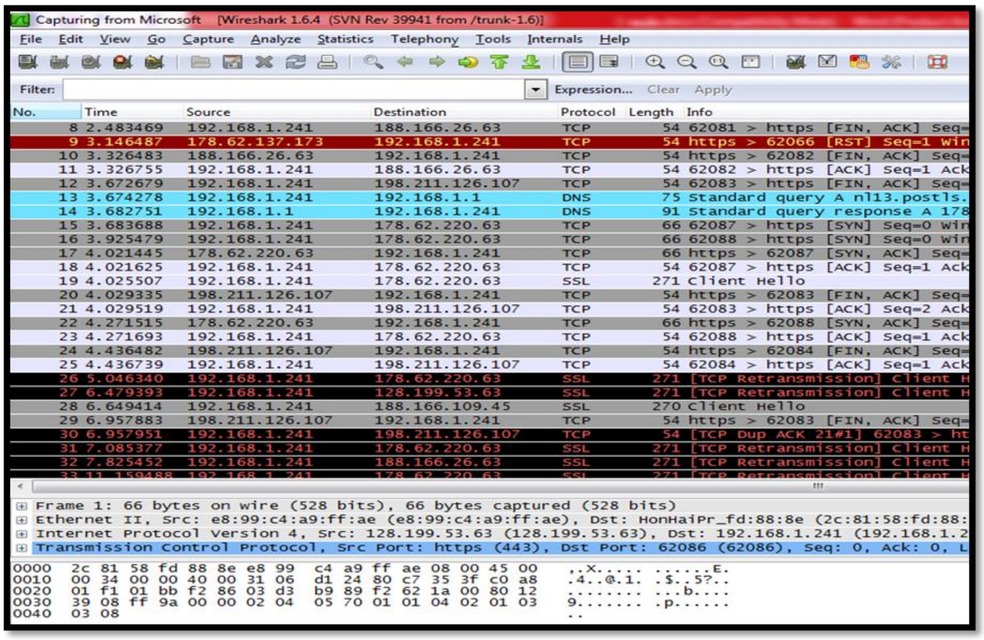


Figure 7: pipeline processing in the Flow table module

In this stimulation, those packets with different headers such as Ethernet, IPV4, IPV6, ICMP, TCP, UDP, ARP, and VLAN, matched with the OpenFlow v 1.3, are used. For producing packets with different headers, the Wireshark Software was used. in figure 8, This software that tries to record the network packets is a packet parser in the network [30].


Capturing from Microsoft [Wireshark 1.6.4 (SVN Rev 39941 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
No.     Time        Source            Destination       Protocol Length Info
      8 2.483469    192.168.1.241     188.166.26.63     TCP        54 62081 > https [FIN, ACK] Seq=
      9 3.146487    178.62.137.173    192.168.1.241     TCP        54 https > 62066 [RST] Seq=1 Win
     10 3.326483    188.166.26.63     192.168.1.241     TCP        54 https > 62082 [FIN, ACK] Seq=
     11 3.326755    192.168.1.241     188.166.26.63     TCP        54 62082 > https [ACK] Seq=1 Ack
     12 3.672679    192.168.1.241     198.211.126.107   TCP        54 62083 > https [FIN, ACK] Seq=
     13 3.674278    192.168.1.241     192.168.1.1       DNS        75 Standard query A nl13.postls.
     14 3.682751    192.168.1.1       192.168.1.241     DNS        91 Standard query response A 178
     15 3.683688    192.168.1.241     178.62.220.63     TCP        66 62087 > https [SYN] Seq=0 Win
     16 3.925479    192.168.1.241     178.62.220.63     TCP        66 62088 > https [SYN] Seq=0 Win
     17 4.021445    178.62.220.63     192.168.1.241     TCP        66 https > 62087 [SYN, ACK] Seq=
     18 4.021625    192.168.1.241     178.62.220.63     TCP        54 62087 > https [ACK] Seq=1 Ack
     19 4.025507    192.168.1.241     178.62.220.63     SSL       271 Client Hello
     20 4.029335    198.211.126.107   192.168.1.241     TCP        54 https > 62083 [FIN, ACK] Seq=
     21 4.029519    192.168.1.241     198.211.126.107   TCP        54 62083 > https [ACK] Seq=2 Ack
     22 4.271515    178.62.220.63     192.168.1.241     TCP        66 https > 62088 [SYN, ACK] Seq=
     23 4.271693    192.168.1.241     178.62.220.63     TCP        54 62088 > https [ACK] Seq=1 Ack
     24 4.436482    198.211.126.107   192.168.1.241     TCP        54 https > 62084 [FIN, ACK] Seq=
     25 4.436739    192.168.1.241     198.211.126.107   TCP        54 62084 > https [ACK] Seq=1 Ack
     26 5.046340    192.168.1.241     178.62.220.63     SSL       271 [TCP Retransmission] Client H
     27 6.479393    192.168.1.241     128.199.53.63     SSL       271 [TCP Retransmission] Client H
     28 6.649414    192.168.1.241     188.166.109.45    SSL       270 Client Hello
     29 6.957883    198.211.126.107   192.168.1.241     TCP        54 https > 62083 [FIN, ACK] Seq=
     30 6.957951    192.168.1.241     198.211.126.107   TCP        54 [TCP Dup ACK 21#1] 62083 > ht
     31 7.085377    192.168.1.241     178.62.220.63     SSL       271 [TCP Retransmission] Client H
     32 7.825452    192.168.1.241     188.166.26.63     SSL       271 [TCP Retransmission] Client H
     33 11.159488   192.168.1.241     178.62.220.63     SSL       271 [TCP Retransmission] Client H

⊞ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: e8:99:c4:a9:ff:ae (e8:99:c4:a9:ff:ae), Dst: HonHaiPr_fd:88:8e (2c:81:58:fd:88:
⊞ Internet Protocol Version 4, Src: 128.199.53.63 (128.199.53.63), Dst: 192.168.1.241 (192.168.1.2
⊞ Transmission Control Protocol, Src Port: https (443), Dst Port: 62086 (62086), Seq: 0, Ack: 0, L

0000  2c 81 58 fd 88 8e e8 99  c4 a9 ff ae 08 00 45 00   ..X..... ......E.
0010  00 34 00 00 40 00 31 06  d1 24 80 c7 35 3f c0 a8   .4..@.1. .$..5?..
0020  01 f1 01 bb f2 86 03 d3  b9 89 f2 62 1a 00 80 12   ........ ...b....
0030  39 08 ff 9a 00 00 02 04  05 70 01 01 04 02 01 03   9....... .p......
0040  03 08                                              ..
```

Figure 8: produced packets by the Wireshark Software

A test bench used for stimulation of this switch was in the VHDL language. It tests performances of parts such as Packet Buffer, Packet Processing (Parser + Composer), Execute Action Set, Flow Table, Controller, and Packet Forwarding. In fact, the data of 1000 different packets of the input file are read and then enter the switch where they are processed and forwarded to an appropriate output port. Here, 4 output ports are considered, on which the packet is forwarded.

After stimulating, the field Pkt_in [63:0] reads contents of the incoming packets as 64 bits per clock in order to process and investigated packet headers in the Parser module (in figure 9).

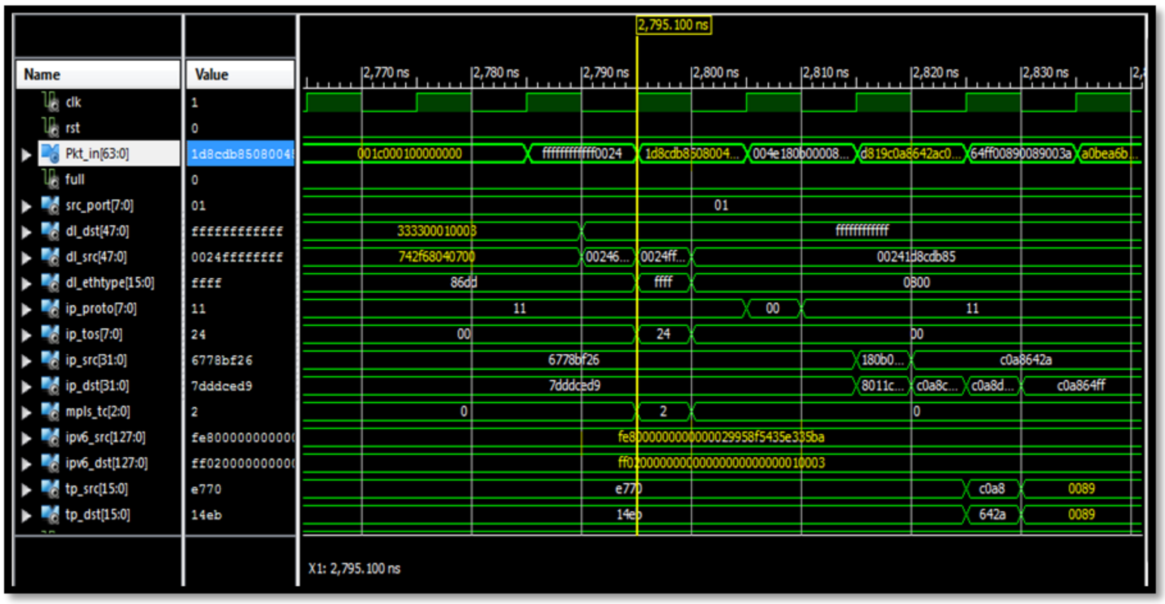


Figure 9: stimulation of the Parser module in reading packets

Data such as the number of the input port, the source and destination addresses, Ethernet type, the VLAN label, packets of IP(Tos, Src, Des, type(TCP,UDP,STCP,ICMP))، IPV6(Src, Des)،MPLS(Lable, Tc, TTl), ARP(Src, Des, Opcode) are extracted in order that other information can be generated for parts such as the Composer module. In figure 10, the fields are extracted from the incoming packets by the Parser module.

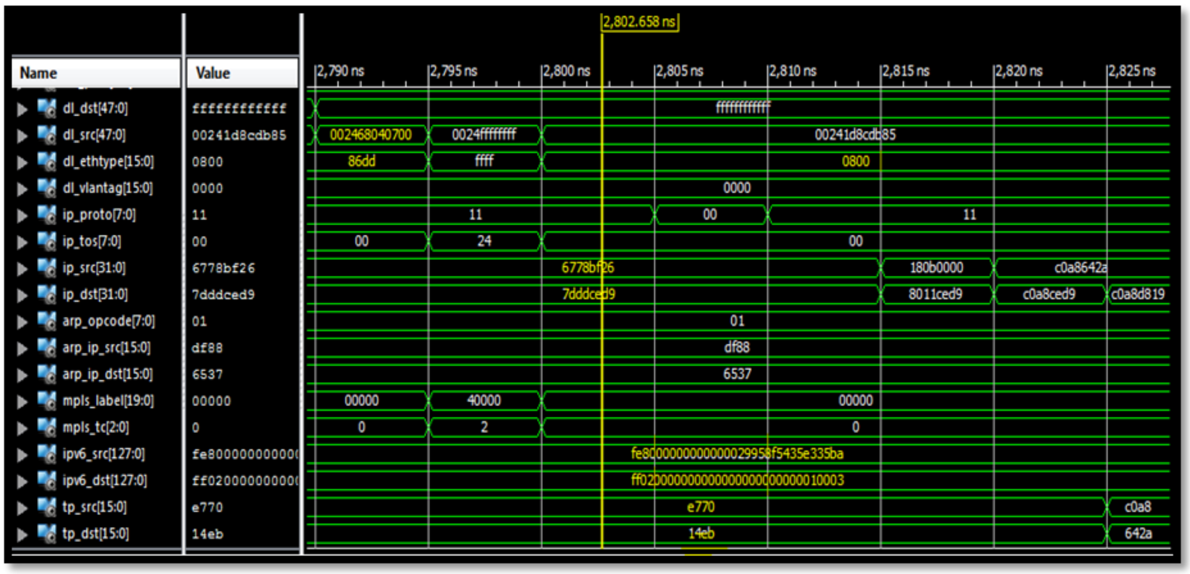


Figure 10: stimulation of the Parse module when removing packet fields

Figure 11 indicates the results obtained from stimulating the Packet Processing top module consisting of the two Consumer and Parser modules. In addition, it has some data for entries of the Flow Table module, the Action part, and Buffer.

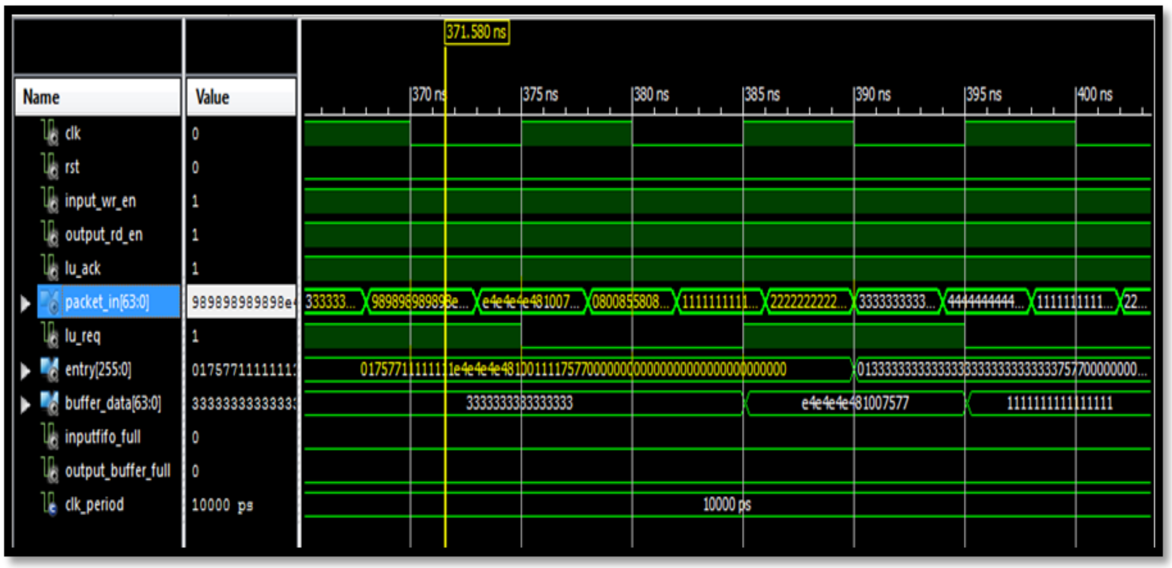


Figure 11: stimulation of the Packet Processing module

In figure 12, stimulation of the Flow Table module shows that the data of packets entering the flow table, fields are matched and the action related to each packet is observed. In this module, commands related to the controller are executed.

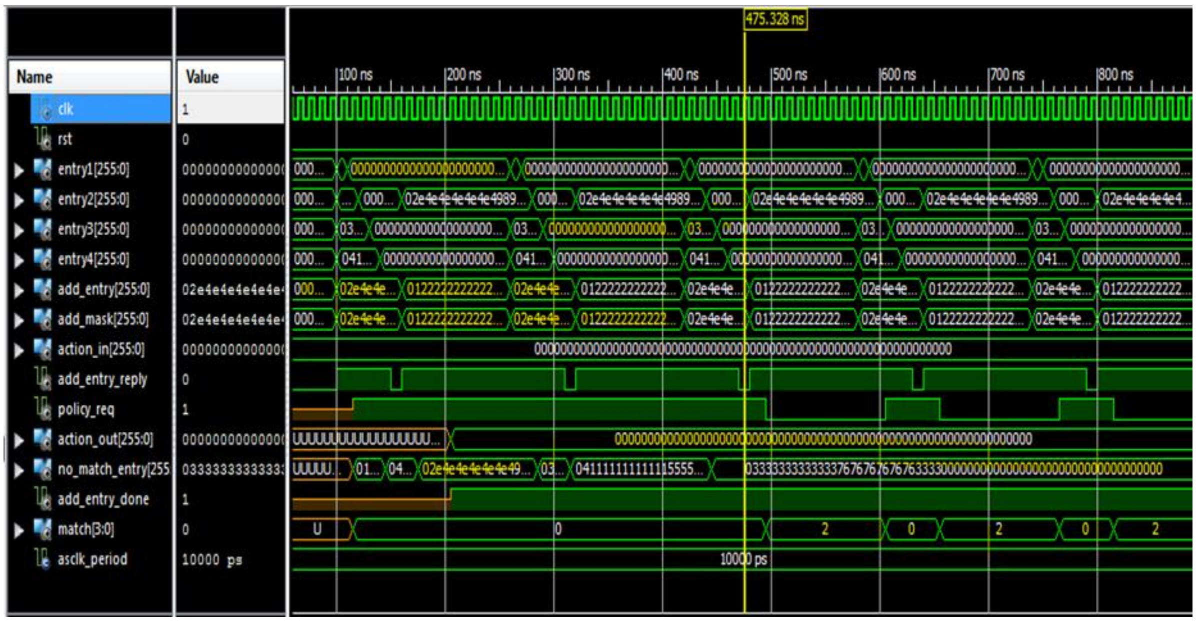


Figure 12: stimulation of the Flow Table module

Figure 13 illustrates stimulation of the Controller module. This module updates the data related to the entries of the flow table and answers the requests received via the fields of Match, Action, and entry to the Flow Table module.

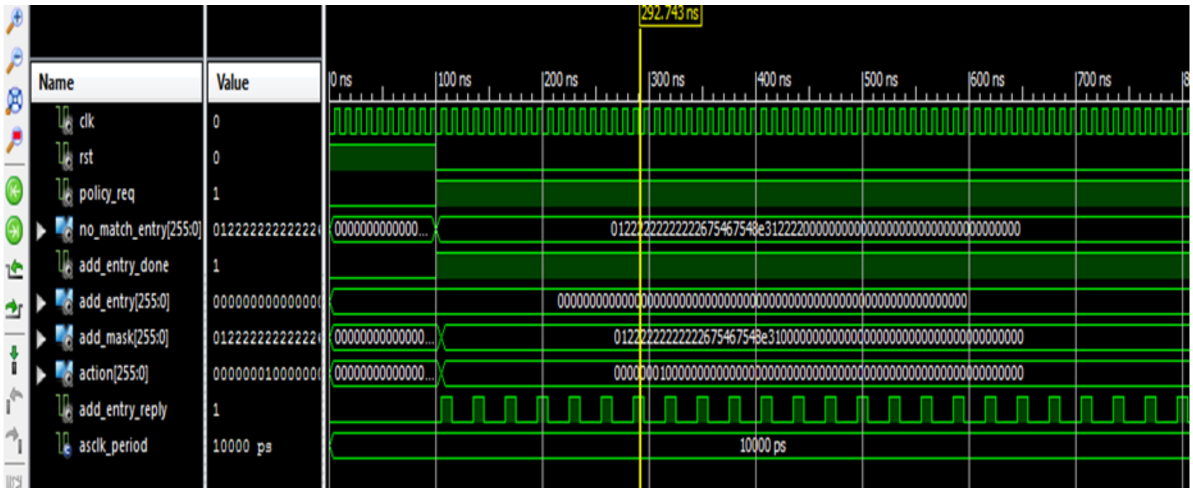


Figure 13: stimulation of the Controller module

Figure 14 shows stimulation of the Execute Action Set module. With regard to the data extracted from the Flow Table module and the incoming packets, this module determines the action related to a packet in order that it can send the packet on an appropriate output port.

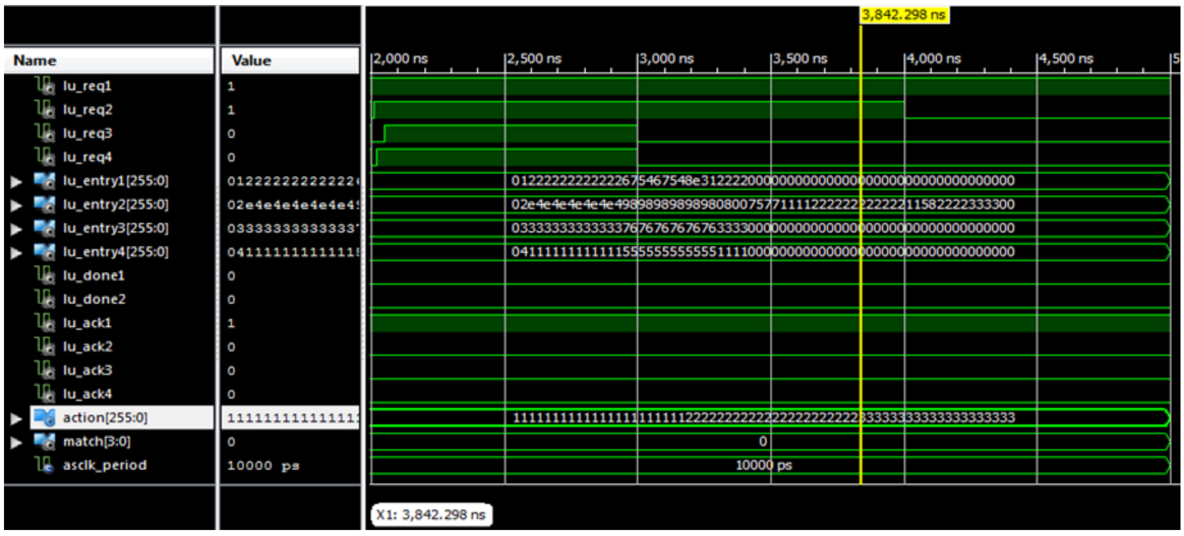


Figure 14: stimulation of the Execute Action Set

Finally, the Packet Forwarding module forwards packets received from the Execute Action Set on te output port in figure 15.

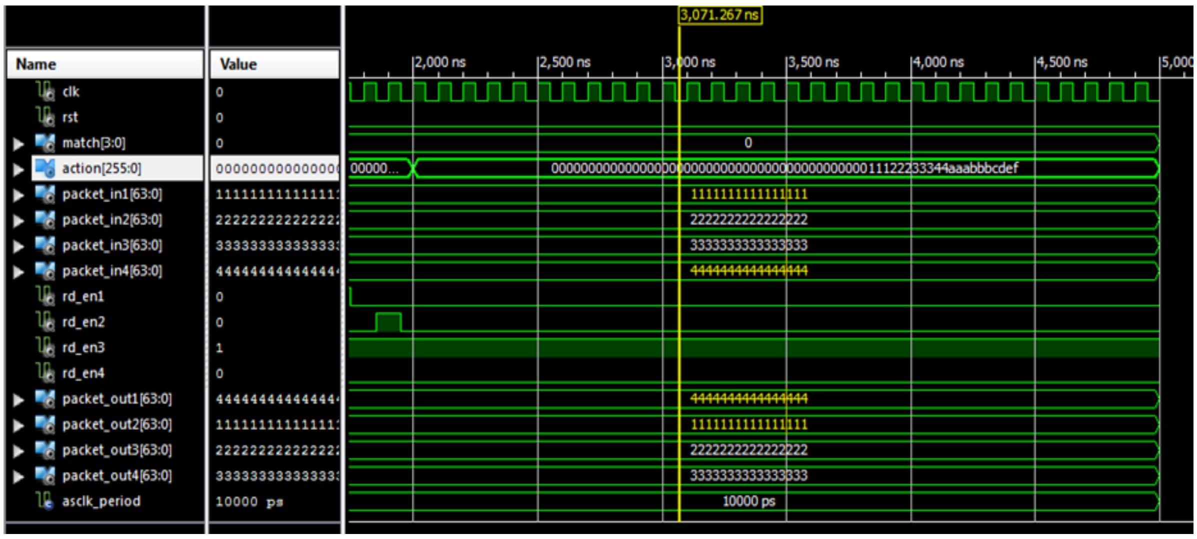


Figure 15: stimulation of the Packet Forwarding module

5. Implementation of the architecture of the OpenFlow v 1.3 Switch

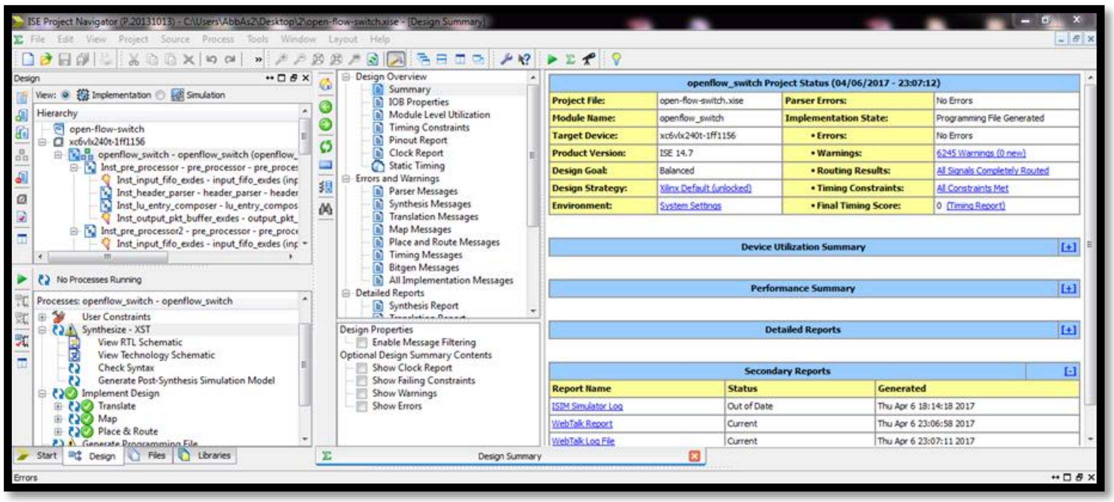
The software architecture of the OpenFlow was implemented on the ML605 (XC6VLX240T) board. This board utilizes the Xilinx FPGA, Virtex-6. Resources such as [22, 23] refer to implementation of this switch on the mentioned board. The ML605 board has features such as High-speed interface (SFP), the clock MHZ 200, compatibility with 10\100\1000 Ethernet PHY(MII/GMII/RMII), and the support of the PCIe*8 Edge Connector [31].

In addition to these features, it also enjoys high efficiency logic of the FPGA, Virtex-6 which consists of a lot of logical and programmable blocks in the system. This feature allows designers to apply a high level of efficiency and performance inside the FPGA-based systems. More features of the FPGA are described in [32]. The amount of hardware resources of the FPGA, Virtex-6 are illustrated in table 4.

Table 4: the amount hardware resources existing in the FPGA, Virtex-6

Logic cells	Configurable Logic Blocks		BRAM	PCIe	Ethernet port	I/O
	Slices	DRAM (Kb)	Max (Kb)			
241,152	37,680	3,650	14,976	2	1	720

The hardware description of the OpenFlow v 1.3 Switch is in the VHDL language. After successful synthesis of this architecture by the ISE 14.7 Xilinx, its implementations is done on the ML605 board (see figure 16).



openflow_switch Project Status (04/06/2017 - 23:07:12)			
Project File:	open-flow-switch.xise	Parser Errors:	No Errors
Module Name:	openflow_switch	Implementation State:	Programming File Generated
Target Device:	xc6vlx240t-1ff1156	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	6245 Warnings (0 new)
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary [+]

Performance Summary [+]

Detailed Reports [+]

Secondary Reports		
Report Name	Status	Generated
ISIM Simulator Log	Out of Date	Thu Apr 6 18:14:18 2017
WebTalk Report	Current	Thu Apr 6 23:06:58 2017
WebTalk Log File	Current	Thu Apr 6 23:07:11 2017

Figure 16: implementation via the ISE 14.7 Xilinx

The amount of hardware resources of this architecture from the Ml605 board is illustrated in table 5 are the amount of resources used in implementing the switch such as resource efficiency, Clock Speed, and consumed power are mentioned. Given that the amount of the consumed resources such as slice registers, slice LUTs, Block RAM/FIFO is low. All performances of the switch controller have not implemented in the OpenFlow v 1.3 Switch designed by the researchers of the present study. The performances forwarded to the switch, adding new flows to the flow table, and implementing performances such as updating the source and destination addresses of the Ethernet are executed in this switch.

Table 5: Design Summary/Reports

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	**5,423**	**301,440**	**1%**
Number used as Flip Flops	648		
Number used as Latches	4,775		
Number of Slice LUTs	**6,971**	**150,720**	**4%**
Number of occupied Slices	**2,266**	**37,680**	**6%**
Number of fully used LUT-FF pairs	**4,308**	**7,041**	**61%**
Number of bonded IOBs	**496**	**600**	**80%**
Number of BRAM/FIFO	**33**	**416**	**6%**
Number of BUFG/BUFGCTRLs	**16**	**32**	**50%**
Number used as BUFGs	16		
Number of slice register sites lost to control set restrictions	**939**	**301,440**	**1%**
Clock Speed	**109.384MHz**		
consumption power	**293 mW**		

5.1. Comparing the evaluation of the presented architecture with other works

The OpenFlow Switch implemented by Khan [22] on the NetFPGA-10G, ML605, and DE4. It was designed by high-level hardware description languages (Blue spec System Verilog (BSV)) which only supports features of the OpenFlow v 1.0. Li [23] investigated the implementation of the OpenFlow v 1.1 Switch on the Ml605 board with the VHDL language, while the switch designed by the researchers of the present study supports the OpenFlow v 1.3, and its implementation of it on the ML605 board has been done by the VHDL language. In table 6, the comparison of architectures presented for the OpenFlow Switch and the amount of resources consumed from hardware platforms are observed.

Table 6: comparison of architectures presented for the OpenFlow Switch

	NetFPGA-10G [22]	DE4 [22]	ML605 [22]	ML605 [23]	ML605
LUTs	**24009**	**11131**	**12062**	**6870**	**6971**
Flips Flops	**29326**	**40287**	**15469**	**4604**	**4308**
Block RAMs	**159**	**1.1 Mb**	**85**	**31**	**33**
OpenFlow versions	**V 1.0**	**V 1.0**	**V 1.0**	**V 1.1**	**V 1.3**
Clock speed	**160 MHz**	**100 MHz**	**100 MHz**	**100 MHz**	**110.38 MHz**
Power	**876 mW**	**442 mW**	**275 mW**	**275 mW**	**291 mW**

Given that the switch implemented by the researchers of the present study supports the features of the OpenFlow v 1.3. As observed in figure 17, it is faster than other versions, but it is at the same level in terms of the hardware volume and the consumed power.

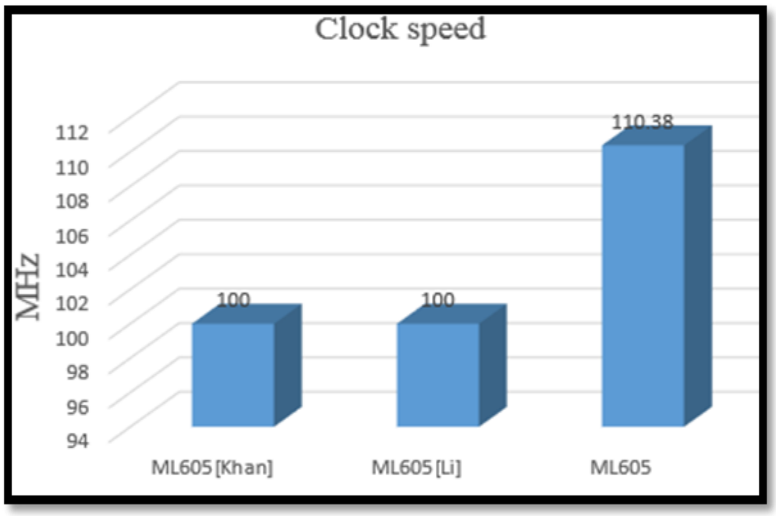


Figure 17: comparing speed of the implemented architectures

6. Conclusion

The SDN is a new architecture for a revolution in the network allowing researchers and developers to test their ideas and implement new protocols via the OpenFlow. The present study introduced the SDN architecture and its components such as the OpenFlow v 1.3 Switch. Implementation of the OpenFlow Switch was conducted on the hardware FPGA platform with re-programmability in the network. The data plane and control plane on the FPGA Xilnix Virtex6 was implemented via the ISE 14.7 design suite. Besides implementing, stimulation of the architecture of the OpenFlow Switch was studied for investigation of its performance accuracy. In implementing the switch, in addition to supporting features of the OpenFlow v 1.3, the possibility of development, high flexibility, and low volume of consumed hardware for processing the pipeline in the Flow Table, the increase in the speed, and support of 40 different headers in the parser are considered in order that they can meet users' increasing needs.

References

[1] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper,* vol. 2, pp. 2.6-6.1, 2012.

[2] J. Nunez-Martinez, J. Baranda, and J. Mangues-Bafalluy, "A service-based model for the hybrid software defined wireless mesh backhaul of small cells," in *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015, pp. 390-393.

[3] L. Y. Ong, "ONF SDN Architecture and Standards for Transport Networks," in *Optical Fiber Communication Conference*, 2017, p. M2H.1 .

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* vol. 103, pp. 14-76, 2015.

[5] H. Hata, "A study of requirements for SDN switch platform," in *Intelligent Signal Processing and Communications Systems (ISPACS), 2013 International Symposium on*, 2013, pp. 79-84.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue,* vol. 11, p. 20, 2013.

[7] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks,* vol. 72, pp. 74-98, 2014.

[8] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, 2013, pp. 1-7.

[9] A. Brunstrom, K.-J. Grinnemo, and J. Taheri, "SDN/NFV-based Mobile Packet Core Network Architectures: A Survey," *IEEE Communications Surveys & Tutorials,* 2017.

[10] NEC, OpenFlow Feature Guide (IP8800/S3640), May 2010. http://support. necam.com/kbtools/sdocs.cfm?id=fcbdcb3e-45fa-4ec4-9311-215bd9ab9f81.

[11] ONF, “OpenFlow Switch Specification,” Dec. 2011. http://goo.gl/tKo6r.

[12] ONF, “OpenFlow Switch Specification,” June 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[13] "Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," April 13, 2012"..

[14] K. Shahmir Shourmasti, "Stochastic switching using openflow," Institutt for telematikk, 2013.

[15] A .Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *ACM SIGCOMM Computer Communication Review*, 2013, pp. 7-12.

[16] J. W. Shin, H. Y. Lee, W. J. Lee, and M. Y. Chung, "Access control with ONOS controller in the SDN based WLAN testbed," in *Ubiquitous and Future Networks (ICUFN), 2016 Eighth International Conference on*, 2016, pp. 656-660.

[17] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice ",*Computer Networks,* vol. 121, pp. 100-111, 2017.

[18] "Open Networking Foundation, "OpenFlow Switch Specification," September 6, 2012"..

[19] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems,* 2008, pp. 1-9.

[20] M. Wielgosz, M. Panggabean, J. Wang, and L. A. Rønningen, "An FPGA-based platform for a network architecture with delay guarantee," *Journal of Circuits, Systems and Computers*, vol. 22, p. 1350045, 2013.

[21] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM Computer Communication Review*, 2013, pp. 99-110.

[22] A. Khan and N. Dave, "Enabling hardware exploration in software-defined networking: A flexible, portable OpenFlow switch," in *Field-Programmable Custom Computing Machines (FCCM* 2013 ,(*IEEE 21st Annual International Symposium on*, 2013, pp. 145-148.

[23] T. Liu, "Implementing Open flow switch using FPGA based platform," Institutt for telematikk, 2014.

[24] Xilinx, LogiCORE IP FIFO Generator v9.2 Product Guide, July 2012. http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v9_2/pg057-fifo generator.pdf.," 2012.

[25] Xilinx, LogiCORE IP Block Memory Generator v7.3 Product Guide, December 2012. http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v7_3/pg058-blk-mem-gen.pdf.," 2013.

[26] Xilinx, ISE Design Suite 14: Release Notes, Installation, and Licensing.

[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69-74, 2008.

[28] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE communications surveys & tutorials*, vol. 16, pp. 493-512, 2014.

[29] T .Mizrahi and Y. Moses, "Software defined networks: It's about time," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, 2016, pp. 1-9.

[30] U. Lamping and E. Warnicke, "Wireshark user's guide," *Interface*, vol.2004 ,4 .

[31] Xilinx, ML605 Hardware User Guide, 1.2.1 ed., January 2010. http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf..

[32] Xilinx, Virtex-6 Family Overview, January 2012. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.