

(10) **Patent No.:** US 7,792,117 B1
(45) **Date of Patent:** Sep. 7, 2010

2008/0002716	A1	1/2008	Wiley et al.
2008/0005156	A1	1/2008	Edwards et al.
2008/0049626	A1	2/2008	Bugenhagen et al.
2008/0049628	A1	2/2008	Bugenhagen
2008/0049630	A1	2/2008	Kozisek et al.

(73) Assignee: **Xilinx, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 416 days.

(21) Appl. No.: 11/799,966

(22) Filed: **May 3, 2007**

(51) **Int. Cl.**
H04L 12/28 (2006.01)

(52) **U.S. Cl.** 370/392; 370/528; 709/223

(58) **Field of Classification Search** 370/229,
370/230, 231, 235, 252, 253, 392, 401, 528;
709/200, 223, 224, 225, 226, 227, 228, 229,
709/239, 231, 232, 233, 234, 235, 238

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,500,796	A	3/1996	Black, Jr.	
6,477,683	B1	11/2002	Killian et al.	
6,665,725	B1	12/2003	Dietz et al.	
6,687,247	B1	2/2004	Wilford et al.	
6,771,646	B1	8/2004	Sarkissian et al.	
6,794,896	B1	9/2004	Brebner	
7,047,303	B2*	5/2006	Lingafelt et al.	709/229
7,398,502	B1	7/2008	Kulkarni et al.	
7,552,042	B1	6/2009	Brebner et al.	
2002/0176378	A1	11/2002	Hamilton et al.	
2003/0177187	A1	9/2003	Levine et al.	
2005/0226238	A1	10/2005	Hoskote et al.	
2008/0002576	A1	1/2008	Bugenhagen et al.	
2008/0002670	A1	1/2008	Bugenhagen et al.	
2008/0002676	A1	1/2008	Wiley et al.	

(Continued)

OTHER PUBLICATIONS

A Performance model of Partial packet discard and early packet discard schemes in ATM switches by Sun etl al. Mar. 2000.*

(Continued)

Primary Examiner—Jason E Mattis

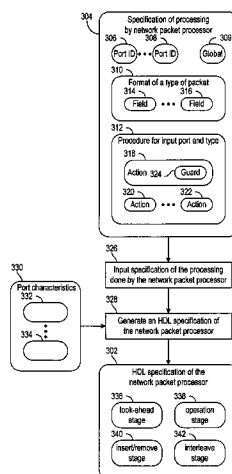
Assistant Examiner—Dady Chery

(74) *Attorney, Agent, or Firm*—LeRoy D. Maunu; Lois D. Cartier

(57) **ABSTRACT**

A method is provided for simulating a processor of network packets. A specification is input for the processor. The specification includes actions specifying a modification of the network packets by the processor. Each action includes a guard condition that enables and disables the action. First and second values of certain fields are determined for each action. The guard condition enables and disables the action respectively for the first and second values of the fields. The network packets are generated. For each field included in the guard conditions, a value of the field is selected for each generated network packet from the values of the field within the first and second values for the actions. The specification of the processor is translated into a simulator of the processor. The modification of the network packets is simulated in the simulator. A result of the modification is displayed on a user interface.

18 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

2008/0049769	A1	2/2008	Bugenhagen
2008/0049927	A1	2/2008	Wiley et al.
2008/0052387	A1	2/2008	Heinz et al.
2008/0052393	A1	2/2008	McNaughton et al.
2008/0052394	A1	2/2008	Bugenhagen et al.
2008/0133518	A1	6/2008	Kapoor et al.
2008/0162390	A1	7/2008	Kapoor et al.

OTHER PUBLICATIONS

U.S. Appl. No. 11/799,897 by Brebner et al., filed May 3, 2007 by Xilinx, Inc.

U.S. Appl. No. 11/799,953 by James-Roxby et al., filed May 3, 2007 by Xilinx, Inc.

U.S. Appl. No. 11/799,860 by Keller et al., filed May 3, 2007 by Xilinx, Inc.

U.S. Appl. No. 11/799,898 by Keller et al., filed May 3, 2007 by Xilinx, Inc.

U.S. Appl. No. 10/769,330, filed Jan. 30, 2004, James-Roxby et al.

U.S. Appl. No. 11/818,722, filed Jun. 14, 2007, Brebner et al.

U.S. Appl. No. 11/818,788, filed Jun. 14, 2007, Attig et al.

U.S. Appl. No. 11/818,792, filed Jun. 14, 2007, Brebner.

U.S. Appl. No. 11/818,811, filed Jun. 14, 2007, Attig et al.

Attig, Michael et al., "Systemic Characterization of Programmable Packet Processing Pipelines," *Proc. of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2006, pp. 195-204, IEEE, Piscataway, New Jersey, USA.

Kountouris, Apostolos et al., "A Method for the Generation of HDL Code at the RTL Level from High-Level Formal Specification Language," *Proc. of the 40th Midwest Symposium on Circuits & Systems*, Aug. 3, 1997, pp. 1095-1098, IEEE, Piscataway, New Jersey, USA.

Shiro, Takeshi et al., "A Processor Generation Method from Instruction Behavior Description Based on Specification of Pipeline Stages and Functional Units," *Proc. Of the 2007 Asia and South Pacific Design Automation Conference*, Jan. 23, 2007, pp. 286-291, IEEE Computer Society, Washington, DC, USA.

* cited by examiner

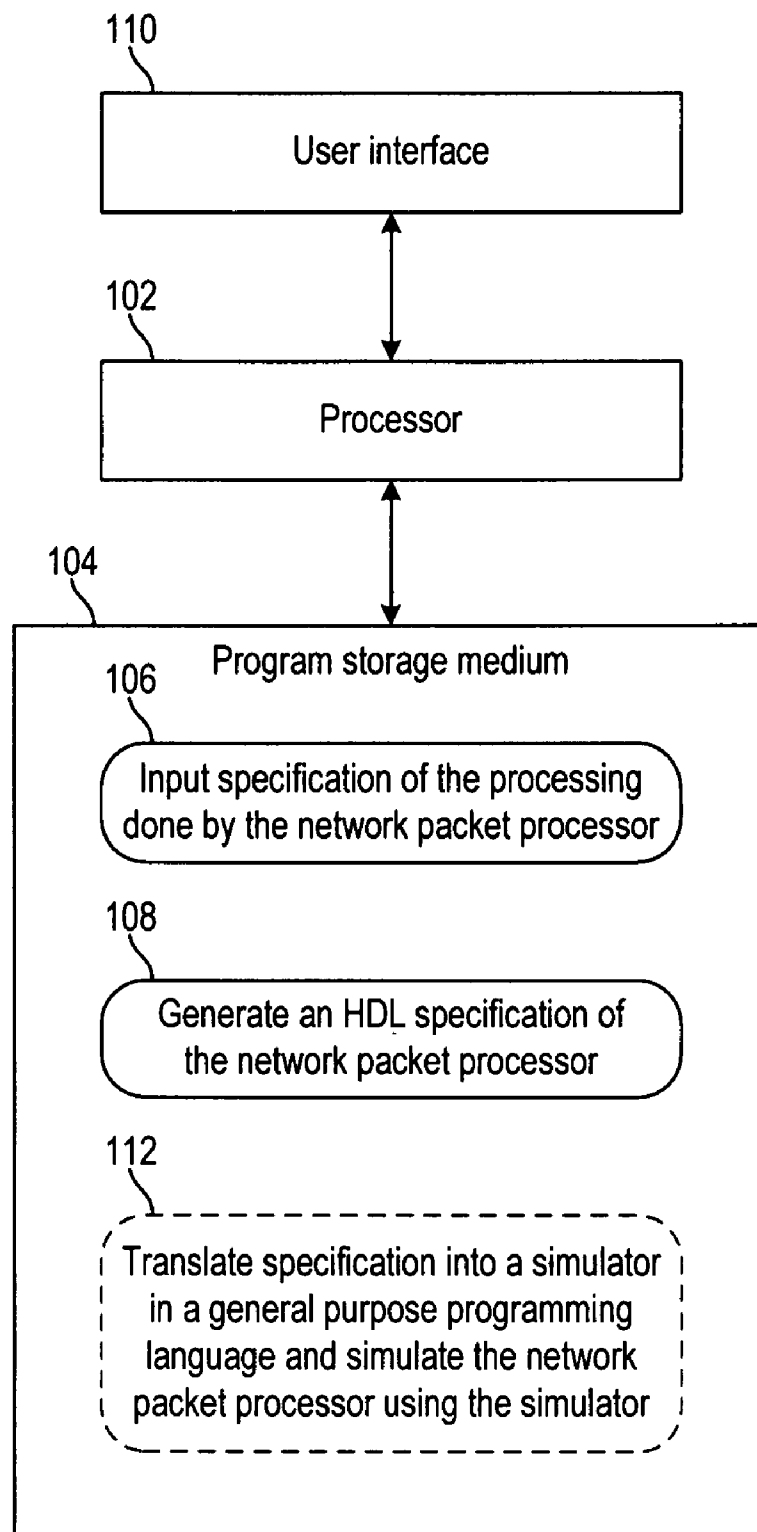


FIG. 1

```

202 { element IPOptionizer {
      // Ports
204 { input packetin : Packet;
      output packetout : Packet;
      output memoryreq : MemoryAccess;
      // Ethernet MAC + LLC + SNAP header
      format header = ( pad : 16,
                        source : 48,
                        destination : 48,
                        length : 16,
                        LLC : 24,
                        SNAP : 40
                        );
      // IP encapsulated in Ethernet packet
      format IPpacket = ( Ethernet : header,
                          version : 4,
                          hlen : 4,
                          TOS : 8,
                          length : 16,
                          id : 16,
206 { flags : 3,
                          offset : 13,
                          TTL : 8,
                          protocol : 8,
                          checksum : 16,
                          source : 32,
                          destination : 32,
                          payload : *
                          );
      // Memory data format – some sort of IP options field
      format optionfield = ( flag : 1,
                             class : 2,
                             number : 5,
                             length : 16,
                             data : 104
                             );
207 { // Global variable (initialization not shown)
      var default : 8;
      // Packet processing example
      handle IPpacket on packetin {
210 { var memorydata : optionfield;
212 { set TTL = TTL - 1;
213 { set checksum = default;
214 { read memorydata from memoryreq [ id ];
216 { [ version == 4 ] insert memorydata after destination;
218 { forward on packetout;
      }
      }
    }
  }

```

FIG. 2

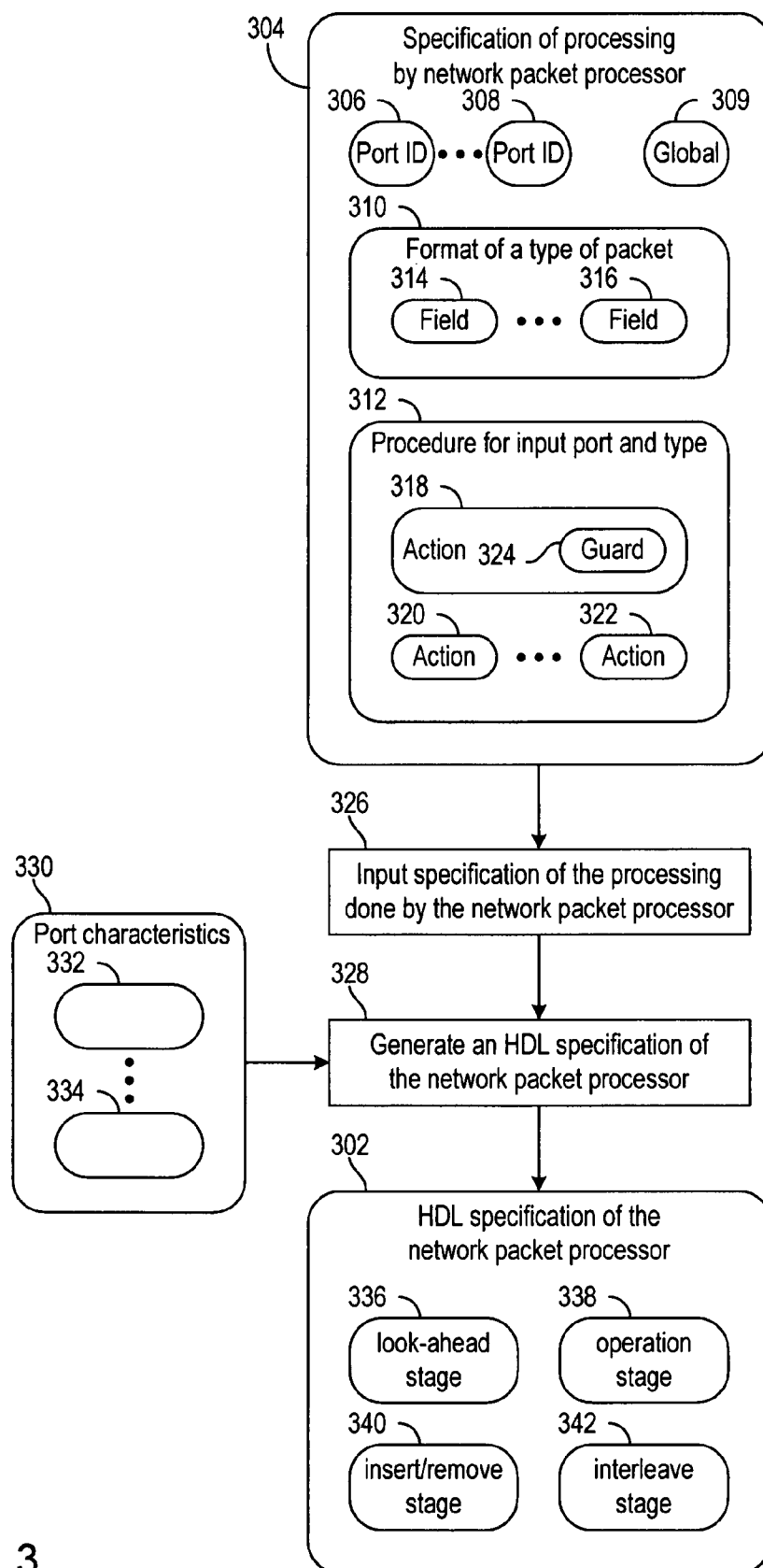


FIG. 3

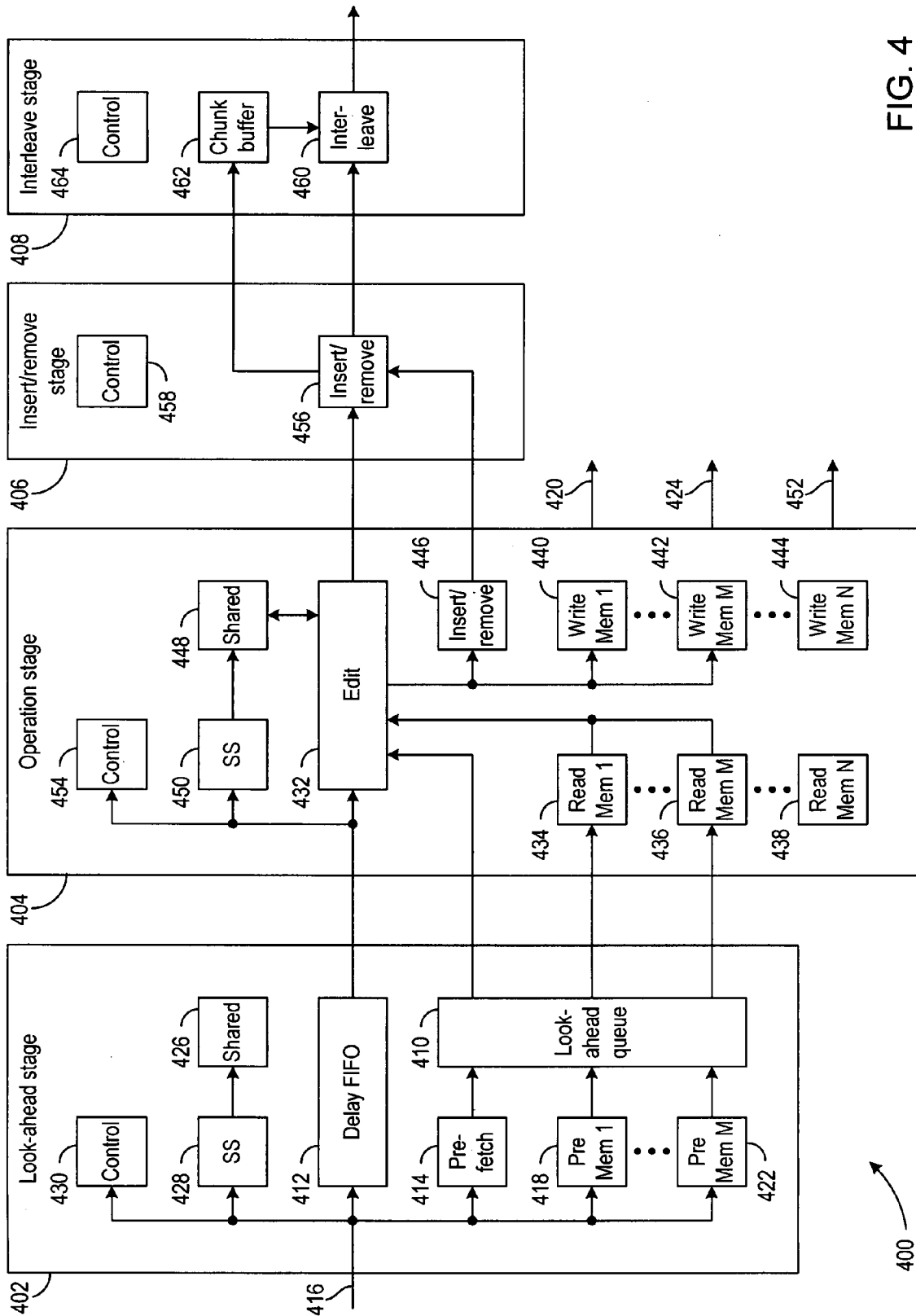


FIG. 4

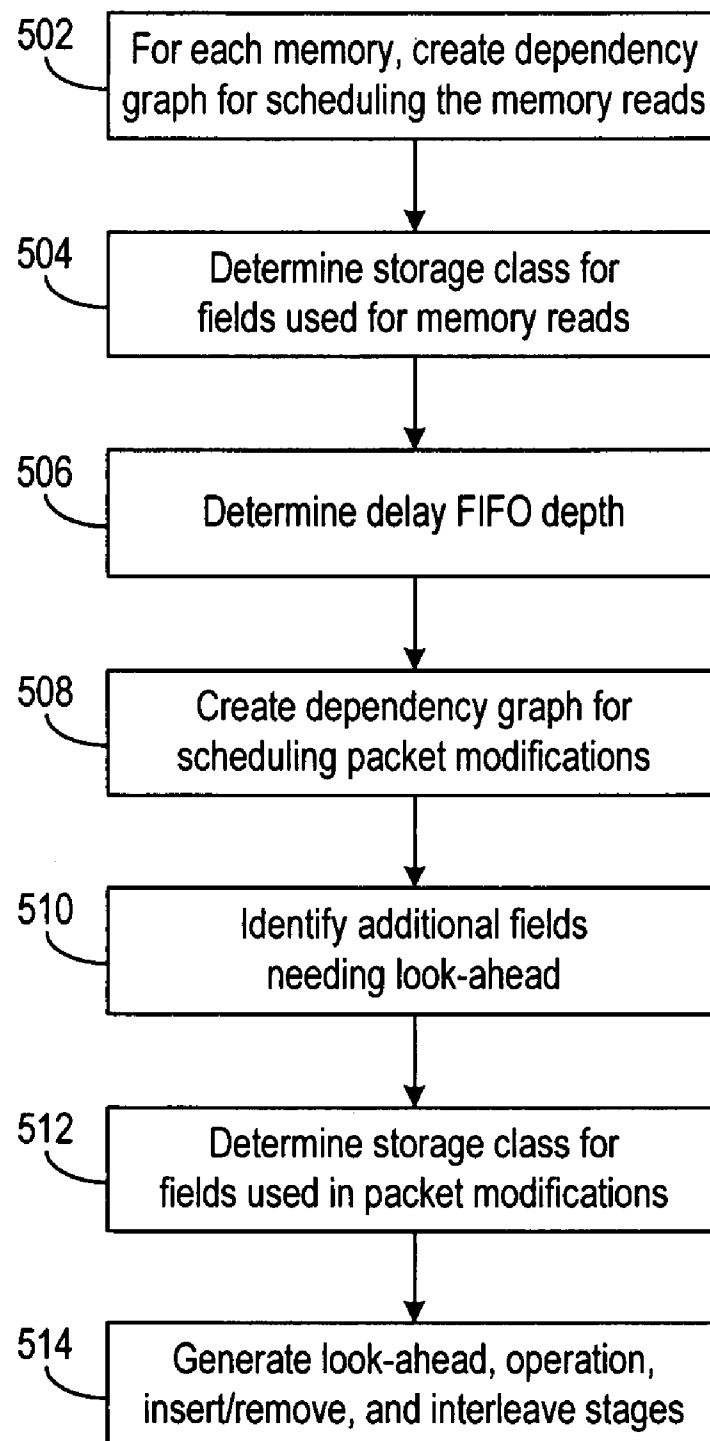


FIG. 5

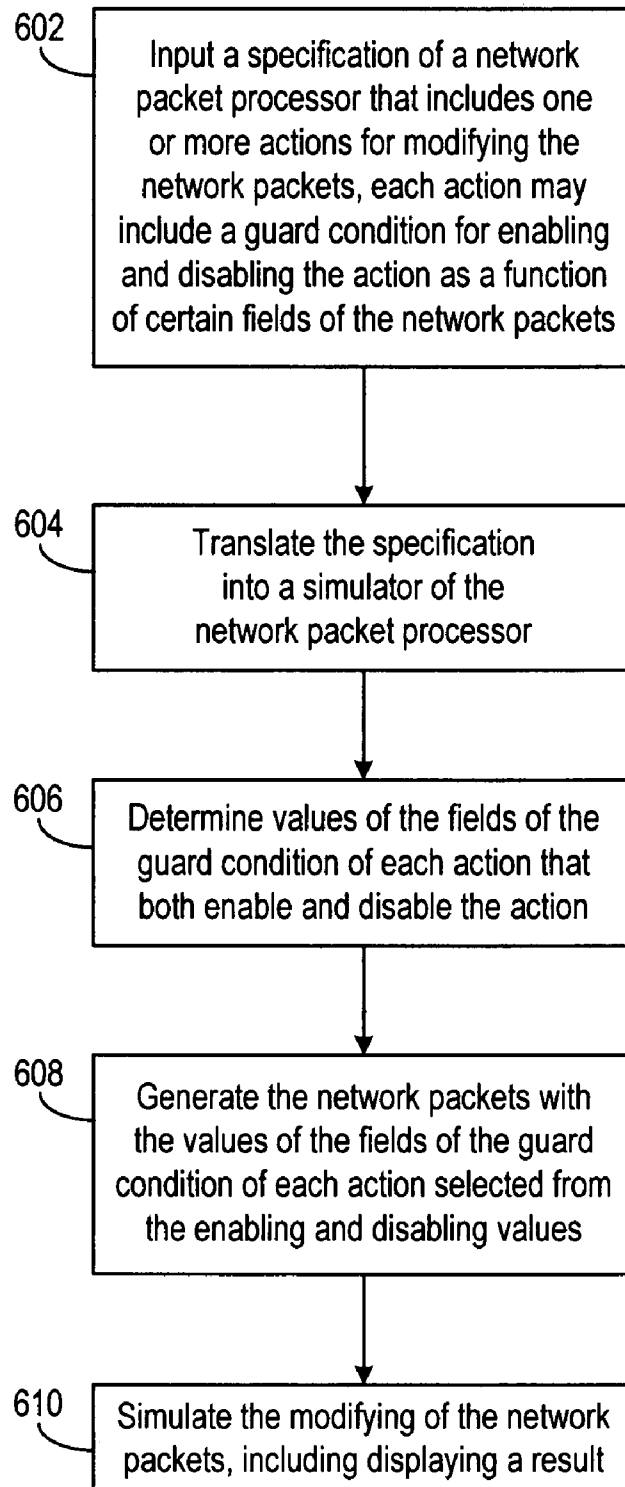


FIG. 6

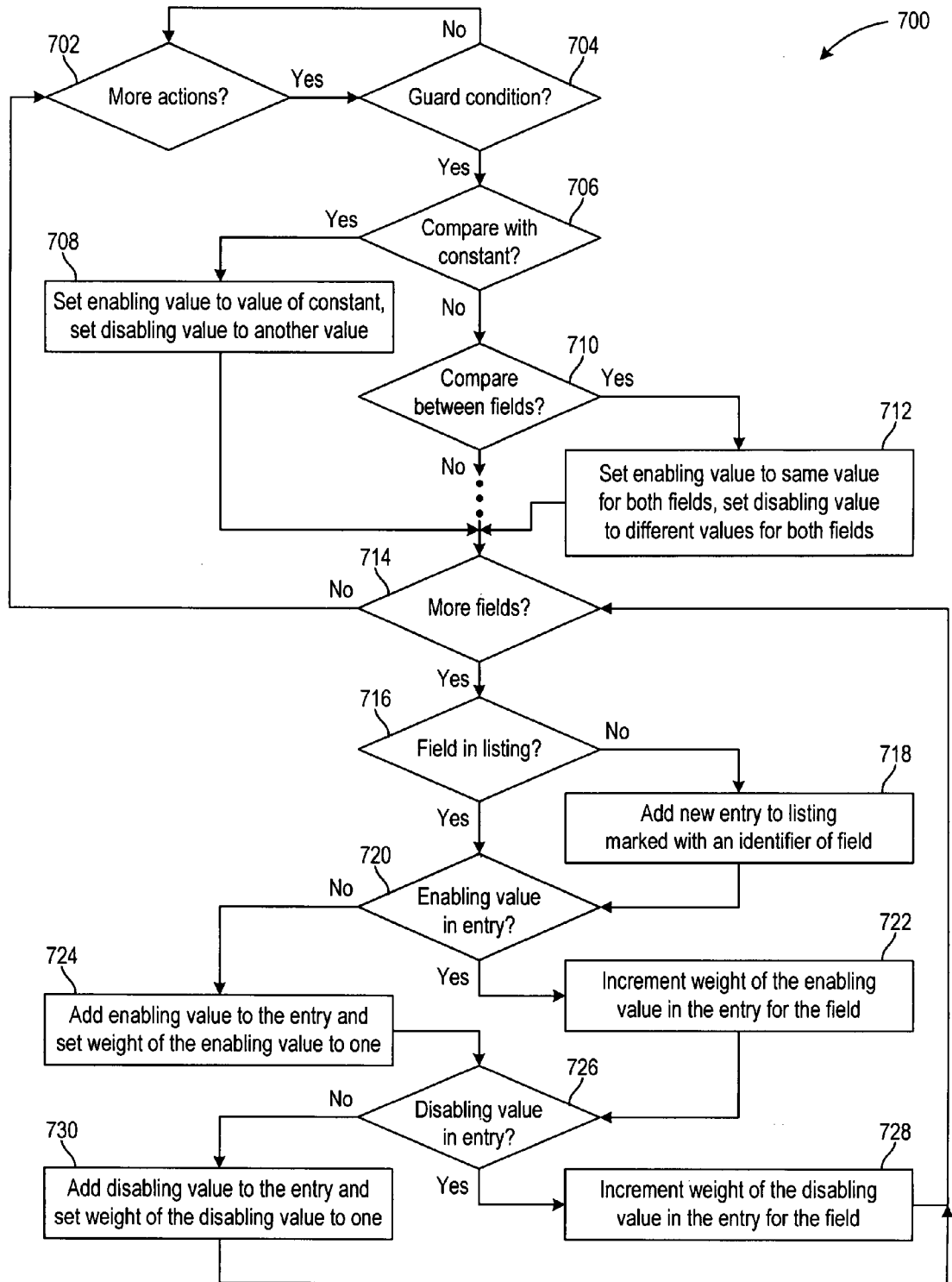


FIG. 7

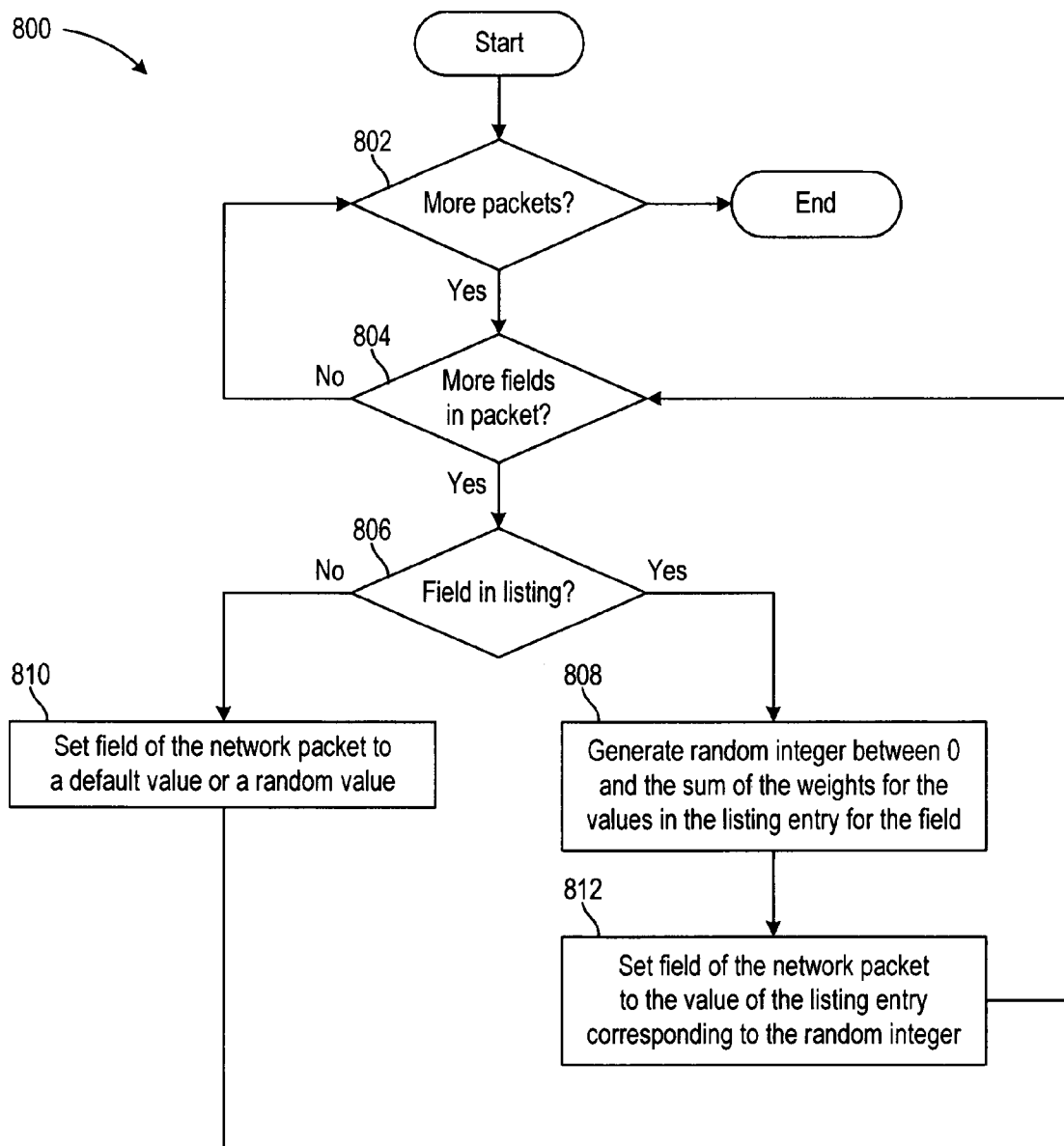


FIG. 8

1

METHOD FOR SIMULATING A PROCESSOR OF NETWORK PACKETS

FIELD OF THE INVENTION

The present invention generally relates to processors of network packets, and more particularly to generation of a network packet processor.

BACKGROUND

A network packet processor may input a stream of network packets, manipulate the contents of the network packets, and output another stream of modified network packets. The manipulations may implement a protocol for processing network packets. For example, the network packet processor may implement a protocol layer of a communication protocol, and for a high-level packet received from a higher protocol layer and delivered to a lower protocol layer for eventual transmission on the communication media, the manipulations may encapsulate the high-level packet within a low-level packet of the lower protocol layer.

A protocol designer may develop a specification of the processing of network packets by a network packet processor. A hardware designer, such as an integrated circuit designer, may create a hardware implementation of a network packet processor that complies with the requirements specified by the protocol designer. Thus, development of a network packet processor implemented in hardware may require the specialized knowledge and skills of at least two separate fields of expertise. Because multiple fields of expertise may be required during development of a hardware implementation of a network packet processor, development of a network packet processor may be time consuming and expensive.

Rapid development of the protocol or implementation of a network packet processor may improve the timeliness and/or quality of the network packet processor. For example, if a hardware implementation can be generated quickly given a specification of the processing of network packets, deficiencies discovered in an initial implementation may be addressed by modifying the specification and generating an improved implementation. Additional iterations of improvement may be possible if the hardware implementation can be generated quickly from a protocol specification.

The present invention may address one or more of the above issues.

SUMMARY

Various embodiments of the invention provide a method for simulating a processor of network packets. A specification in a textual language is input for the processor of the network packets. The specification includes actions specifying a modification of the network packets by the processor. Each action includes a respective guard condition that enables and disables the action in response to certain fields of a type of the network packets. The specification of the processor is translated from the textual language to a simulator of the processor in a general-purpose programming language. A first value and a second value of certain fields are determined for each action. The respective guard condition enables the action for the first value of the fields and disables the action for the second value of the fields. The network packets are generated. For each field included in the guard conditions of the actions, a value of the field is selected for each generated network packet from the values of the field within the first and second values for the

2

actions. The modification of the network packets is simulated in the simulator. A result of the modification is displayed on a user interface.

BRIEF DESCRIPTION OF THE DRAWINGS

Various aspects and advantages of the invention will become apparent upon review of the following detailed description and upon reference to the drawings in which:

FIG. 1 is a block diagram of a system for generating a hardware description language specification of a network packet processor in accordance with various embodiments of the invention;

FIG. 2 is a listing of an example textual language specification of a network packet processor in accordance with various embodiments of the invention;

FIG. 3 is a data flow diagram of a process of generating a hardware description language specification of a network packet processor in accordance with various embodiments of the invention;

FIG. 4 is a block diagram of a network packet processor that is generated in accordance with various embodiments of the invention;

FIG. 5 is a flow diagram of a process for generating a pipeline for processing network packets in accordance with various embodiments of the invention;

FIG. 6 is a flow diagram of a process for simulating a processor of network packets in accordance with various embodiments of the invention;

FIG. 7 is a flow diagram of a process for determining values of certain fields in a type of network packets in accordance with various embodiments of the invention; and

FIG. 8 is a flow diagram of a process for generating the network packets in accordance with various embodiments of the invention.

DETAILED DESCRIPTION

Various embodiments of the invention permit a network packet processor to be implemented from a specification of the required processing of network packets by the network packet processor. The implementation of the network packet processor may be a hardware implementation in a programmable logic device (PLD) that may be quickly generated from the specification of the processing protocol of the network packet processor. The generated hardware implementation may have low latency and may efficiently utilize the resources of the PLD.

FIG. 1 is a block diagram of a system for generating a hardware description language (HDL) specification of a network packet processor in accordance with various embodiments of the invention. A processor 102 reads and executes instructions from a program storage medium 104, and the execution of the instructions causes processor 102 to generate the HDL specification of the network packet processor. Software modules 106 and 108 may include the instructions of the program storage medium 104.

Execution of the instructions of software module 106 may cause processor 102 to input a specification of the processing to be performed by the network packet processor. In one embodiment, the network packet processor may receive network packets and transmit a modification of the received network packets. The specification of the processing by the network packet processor may be a protocol specification of the modifications to be performed by the network packet processor. For example, the network packet processor may implement a protocol layer of a communication protocol, and

for a high-level packet received from a higher protocol layer and delivered to a lower protocol layer for eventual transmission on the communication media, the specification may describe the encapsulation of the high-level packet within a low-level packet of the lower protocol layer. The encapsulation specified by the protocol specification may include insertion of a header with a particular format before the beginning of the high-level packet. The header may be generated from information in the incoming high-level packet and from information that is stored within the network packet processor.

Execution of the instructions of software module **106** may cause processor **102** to input the specification from the user interface **110**. In one embodiment, the user interface **110** may permit directly inputting a textual language specification of the processing expected from the network packet processor. In another embodiment, the textual language specification may be input from a data file stored in program storage medium **104** or another storage medium of the system.

Execution of the instructions of software module **108** may cause processor **102** to generate an HDL specification of the network packet processor from the textual language specification input by processor **102** during execution of the instructions of software module **106**. The network packet processor generated using software module **108** may implement the manipulations of the textual language specification.

Execution of the instructions of the optional software module **112** may cause processor **102** to translate the specification of the network packet processor into a simulator in a general purpose programming language and to simulate the network packet processor using the simulator.

FIG. **2** is a listing of an example textual language specification of a network packet processor in accordance with various embodiments of the invention. Various embodiments of the invention may translate the example textual language specification into a network packet processor that processes network packets as specified. The specification includes a declaration **202** of the name the network packet processor, a declaration **204** of the ports of the network packet processor, a declaration **206** of formats of types of network packets and other data structures, a declaration **207** of a global variable, and a specification **208** of a handler procedure for processing the network packets by the network packet processor.

As specified in ports declaration **204**, the network packet processor has an input port with identifier "packetin" for receiving network packets, an output port with identifier "packetout" for transmitting network packets, and a memory port with identifier "memoryreq" for accessing an external memory. As specified in the handler specification **208**, the network packets received at the input port are modified and the modified network packets are transmitted from the output port. Certain of the received network packets are modified using data that is read from an external memory using the memory port. It will be appreciated that a network packet may also be modified using state data that is stored within the network packet processor.

The handler specification **208** specifies processing of network packets that have type "IPpacket" and are received at the input port "packetin." The handler specification **208** includes a declaration **210** of a local variable and specifications of five actions, including set action **212**, set action **213**, memory read action **214**, insert action **216**, and forwarding action **218**. For each network packet of type "IPpacket" that is received at the input port "packetin," set action **212** decrements the "TTL" field of the network packet. Set action **213** sets the value of the "checksum" field to a value of the global variable of declaration **207**. Memory read action **214** reads a memory at the address given by the "id" field of the network packet. The

insert action **216** includes a guard condition within the square brackets that enables the insert action **216** when the "version" field of the network packet has a value of 4, and otherwise the insert action **216** is disabled. If the insert action **216** is enabled by the guard condition, the insert action **216** inserts data from the memory read action **214** after the "destination" field of the network packet. Forwarding action **218** transmits the modified network packet at the output port "packetout."

The value of the global variable of declaration **207** could be updated by a handler procedure during the processing of one network packet, and the updated value could be used during the processing of another network packet by the handler procedure. Thus, the global variable of declaration **207** may provide state data that is preserved between network packets. In contrast, the local variable of declaration **210** may be temporary data that should be reinitialized for each network packet that is processed.

The textual language for specifying the processing of a network packet handler may closely correspond with the techniques that a designer uses to specify a communication protocol. A designer of communication protocol may develop the textual language specification without having the specialized knowledge required to develop an integrated circuit, and various embodiments of the invention may translate the textual language specification of a communication protocol into an efficient implementation of the communication protocol in a network packet processor. Thus, various embodiments of the invention permit a designer of a communication protocol to develop an integrated circuit for efficiently implementing the communication protocol even though the designer does not have the skills for designing an integrated circuit.

FIG. **3** is a data flow diagram of a process of generating a hardware description language (HDL) specification **302** of a network packet processor in accordance with various embodiments of the invention. The HDL specification **302** may be generated from a textual language specification **304** of the processing performed by the network packet processor.

The textual language specification **304** of the network packet processor may include port identifiers **306** and **308**, global variables **309**, a format **310** or formats for one or more types of network packets and possibly other data structures, and a procedure **312** for each input port and each type of network packet that is expected at the input port of the network packet processor. The port identifiers **306** and **308** may include respective identifiers for the input, output, and any memory ports of the network packet processor. The global variables **309** may provide state data that is preserved between network packets. A format **310** includes the fields **314** and **316** of a type of network packet, and a field **314** or **316** may be declared in the format **310** as having a specified name and a specified size, such as a number of data bits of the field. The fields **314** and **316** may be specified in format **310** in an order corresponding to an order of the network packet manager receiving or transmitting the fields in the type of network packets.

A procedure **312** may include actions **318**, **320**, and **322** for manipulating network packets of a particular type that are received at a particular input port. Examples for actions **318**, **320**, and **322** include a set action for setting a value of a field **314** as a function of global variables **309** and/or fields **314** and/or **316**, an insert action for data insertion at a field **314** as a function of global variables **309** and/or fields **314** and/or **316**, a remove action for data removal at field **314** as a function of global variables **309** and/or fields **314** and/or **316**, an update action for updating one of the global variables **309** as a function of the global variables **309** and/or fields **314** and/or **316**, a forward action for transmitting modified network

packets from an output port of the network packet processor, a memory read action for reading a value from a memory, and a memory write action for writing a value to the memory.

An action **318** may include a guard condition **324** for enabling and disabling the action **318**, with the action **318** performed in response to the guard condition **324** being satisfied and the action **318** being bypassed in response to the guard condition **324** not being satisfied. Another action **320** might not include a guard condition and the action **320** may always be performed on each network packet that is processed using procedure **312**. Examples for guard condition **324** include a check of a value of a field **314** of a type of the network packets, a comparison between two fields **314** and **316** of a type of the network packets, and a comparison between a field **314** and one of the global variables **309**.

At step **326**, the specification **304**, which specifies the processing performed by the network packet processor, is input. At step **328**, the HDL specification **302** of the network packet processor is generated. To generate the HDL specification **302** of the network packet processor, a specification **330** may be input that specifies the characteristics **332** and **334** of the ports having identifiers **306** and **308**. Example characteristics include a width for each of the ports and a signaling protocol for each of the ports.

The generated HDL specification **302** may include a look-ahead stage **336**, an operation stage **338**, an insert/remove stage **340**, and an interleave stage **342**. The look-ahead stage **336** may synchronize among the fields **314** and **316** that are accessed by the actions **318**, **320**, and **322** of the procedure **312**. The look-ahead stage **336** may make certain of fields **314** and **316** concurrently available for processing according to the actions **318**, **320**, and **322** of the procedure **312**. The operation stage **338** may implement the actions **318**, **320**, and **322** that are not insert or remove actions. For each set action, the operation stage may set one of fields **314** and **316** according to the set action. The operation stage **338** may evaluate a guard condition **324** for an action **318**, and bypass performing the action **318** when the guard condition **324** is not satisfied. The insert/remove stage **340** may perform data insertion and/or data removal at one or more of fields **314** and **316** according to the actions **318**, **320**, and **322** that are insert or remove actions. The data insertion or data removal may be bypassed for an insert or remove action with a guard condition that is not satisfied. The interleave stage **342** may ensure that the modified network packet follows rules for interleaving network packets.

A sequence of network packets may be received at an input port of the network packet processor, with each network packet including a series of data chunks. The data chunks for the network packets may be serially received in an interleaved manner. For example, network packet A may include data chunks A[0], A[1], and A[2], and network packet B may include data chunks B[0], B[1], and B[2], which are serially received at an input port of the network packet processor in the interleaved order A[0], B[0], A[1], B[1], A[2], B[2]. For a network packet processor with an insert/remove stage **340** that modifies network packet A by removing a portion of data chunk A[1], to output full data chunks the network packet processor may need to fill in modified data chunk A[1] with data from subsequent chunks of network packet A. Similarly, for a network packet processor with an insert/remove stage **340** that modifies network packet B by inserting data into chunk B[1], to output proper data chunks the network packet processor may need to move data from modified data chunk B[1] to subsequent chunks of network packet B.

In one embodiment, a data chunk is the data width of the input and output ports of a network packet processor. The

interleaving rules may permit interleaving of network packets between each data transfer across the width of a port of the network packet processor. After a data insertion or data removal with a size that is not a multiple of the data chunk, the interleave stage **342** may pad the modified network packet with enough null data to make a full data chunk at the end of the network packet. It will be appreciated that the interleave stage **342** may be useful for generating complete data chunks across the full width of the input and output ports even though data chunks for different network packets are never interleaved.

In another embodiment, a data chunk is a burst of a fixed or variable number of data transfers across the width of a port of the network packet processor, and interleaving of network packets may occur between the bursts of the data chunks. For a variable number of data transfers, the interleaving rules may specify an upper limit on the number of data transfers within a burst. After a data insertion that increases the size of a network packet beyond the fixed number of data transfers allowed for a burst or beyond the upper limit on a variable number of data transfers for a burst, the interleave stage **342** may divide the modified network packet into multiple bursts to satisfy the interleaving rules.

It will be appreciated that the port characteristics **330** may specify both the width of the ports and the burst lengths for the ports. In addition, port characteristics **330** may specify whether network packets are received in an interleaved manner by the network packet processor.

FIG. 4 is a block diagram of a network packet processor that is generated in accordance with various embodiments of the invention. The network packet processor is a pipeline **400** that includes a look-ahead stage **402**, an operation stage **404**, an insert/remove stage **406**, and an interleave stage **408**. Stages **402**, **404**, **406**, and **408** of the pipeline **400** are configured by a textual language specification of the processing by the network packet processor.

The textual language specification may specify one or more actions for modifying the fields of a network packet as a function of the fields of the network packet and global variables corresponding to stored state **428** and **450** within the network packet processor. For example, a set action may modify a field of a network packet as a function of another field of this network packet, and these two fields may be received at different times during a serial reception of the data chunks of the network packet. Each modification of a field may be completed upon receiving the field providing the field is modified as a function of stored state and fields that are already available from the network packet. In contrast, a modification of a field might not be possible upon receiving the field when the field is modified as a function of a field or fields that are later received. The look-ahead stage **402** may effectively advance the availability of each field that is used to modify an earlier-received field, such that the operation stage **404** may perform the modification for each set action upon receiving the data chunk that includes the field that is modified by the set action.

To effectively advance the availability of certain fields, the look-ahead stage **402** may promptly provide these fields from look-ahead queue **410**, while the look-ahead stage **402** may provide other fields with delayed availability from delay FIFO **412**. Various fields may be concurrently available from the look-ahead queue **410** and the delay FIFO **412**. The delay FIFO **412** may have a depth equaling the maximum amount of relative advancement that is needed. If no fields need advanced availability for a particular textual language specification, the look-ahead stage **402** may be omitted from the pipeline **400**.

Certain fields of a network packet that need advanced availability may be selected by pre-fetch block **414** and stored in look-ahead queue **410** during reception of the network packet at the input port on line **416**. Because these fields are promptly made available to the operation stage **404** from look-ahead queue **410**, these fields are effectively advanced relative to fields that are later made available to the operation stage **404** from delay FIFO **412**.

The network packet processor may access a memory or memories during the processing of a network packet. For example, an address for a memory read action may be formed from a field of the network packet and a set action may modify the same field of the network packet using the value read from the memory at the address. The set action may include a guard condition that enables modifying the same field with a modification value, and either the guard condition or the modification value of the set action may be dependent on the value read from the memory. If a delay interval is required for reading the memory, the modification of the field might not be immediately possible upon receiving the field. The delay interval may be a latency of the memory and/or a time interval needed to form wide data by reading narrow data multiple times from the memory. The look-ahead stage **402** may also effectively advance the availability of certain fields needed for reading from one or more memories.

For the memory read actions that need advanced availability and that access the memory connected to the memory port on line **420**, memory pre-fetch block **418** may capture any fields needed to generate any addresses for these memory read actions. Memory pre-fetch block **418** may also capture the fields needed to generate any guard conditions for these memory read actions.

Similarly, memory pre-fetch block **422** may capture the fields needed to determine any addresses and any guard conditions for the memory read actions that need advanced availability for the memory connected to the memory port on line **424**. When advanced availability is needed for a value read from a memory for a memory read action, the amount of advanced availability may contribute to determining the depth of the delay FIFO **412**.

The stored state block **428** may include state data for certain of the global variables of the specification of the processing of the network packet processor. Some of the global variables may be used in actions needing advanced availability, and the stored state block **428** of look-ahead stage **402** may store the state data for these global variables, while the stored state block **450** of operation stage **404** may store the state data for the remaining global variables.

Certain fields of a network packet may be referenced by multiple actions of the textual language description. For example, a specific field may be referenced by two set actions, which modify respective fields that are in different data chunks and are consequently received by the pipeline **400** at different times. Thus, certain fields may need to be available at multiple time steps and to eliminate duplicated storage these fields may be stored in a register within shared block **426**. Shared block **426** may make these fields and/or certain temporary variables available to one or more of blocks **414**, **418**, and **422**.

The control block **430** may control the sampling of certain fields of a network packet by the blocks **428**, **426**, **414**, **418**, and **422** as specified by the certain actions of the textual language specification.

The operation stage **404** may implement the actions of the textual language specification excluding the insert and remove actions. The edit block **432** may perform the set actions on each network packet as the data chunks of the

network packet are received from the delay FIFO **412** of the look-ahead stage **402**. The edit block **432** may perform the set actions using fields with advanced availability from the look-ahead queue **410** of the look-ahead stage **402** and memory read results from the memory read blocks **434**, **436**, and **438**.

The edit block **432** may determine information that is provided to the memory write blocks **440**, **442**, and **444**, and to the insert/remove block **446**. Certain fields and/or temporary variables may be shared between the shared block **448** and the blocks **432**, **434**, **436**, **438**, **440**, **442**, **444**, and **446**. The edit block **432** may update the values of any temporary variables in the shared block **448**.

Because the memory connected to the memory port on line **420** may have a memory read action that produces a read result needing advanced availability, the memory read block **434** may receive certain fields with advanced availability from the look-ahead queue **410**. These fields may be used to determine any address or any guard conditions for this memory read action. Any address or guard condition for this memory read action may also be determined from other fields that do not need advanced availability and state data from stored state **450**.

The memory read block **434** may implement all of the memory read actions that target the memory connected to the memory port on line **420**. Depending upon whether or not the read result needs advanced availability for a memory read action accessing the memory port on line **420**, the memory read block **434** may generate an address or guard condition for the memory read action using information from the look-ahead queue **410**, or the memory read block **434** may generate an address or guard condition using information from the shared block **448**.

The memory read block **438** may implement all of the memory read actions that target the memory connected to the memory port on line **452**, and none of these memory read actions may need advanced availability of the read result. Because the memory read block **438** does not need to obtain advanced information from the look-ahead queue **410**, no connection is shown between the look-ahead queue **410** and the memory read block **438**. For example, the memory connected to the memory port on line **452** may be a FIFO or configuration register that does not require an address and hence advanced availability is not needed for the result read from the memory because the memory may be read at any time.

It will be appreciated that the actual number of memory read blocks **434**, **436**, and **438** for a particular textual language specification may depend on the number of memories accessed by the memory read actions of the textual language specification. In addition, the connections between the look-ahead queue **410** and the memory read blocks **434**, **436**, and **438** may depend on whether advanced availability is needed for the memory reach actions of the textual language specification.

For each memory write action of the textual language specification, the edit block **432** may provide the write value along with an optional address and optional guard condition to the write memory blocks **440**, **442**, and **444**. Write memory blocks **440**, **442**, and **444** may implement the memory write actions that write to the respective memories connected to memory ports on lines **420**, **424**, and **452**.

For each insert action and each remove action of the textual language specification, the insert/remove block **446** may determine the position of the data insertion or data removal and the amount of data to be inserted or removed. The insert/remove block **446** may also determine the value of any guard condition for enabling and disabling the data insertion or data

removal. For data insertion, the insert/remove block 446 may determine the insertion data. The insert/remove block 446 may obtain information for determining the position and amount of the data insertion or data removal and the insertion data from the edit block 432 and/or the shared block 448.

The control block 454 may manage the operation of the blocks 432, 434, 436, 438, 440, 442, 444, 446, and 448 of the operation stage 404.

The insert/remove stage 406 implements the insert actions and remove actions of the textual language specification. The insert/remove stage may be omitted from the pipeline 400 for a specific textual language specification that does not include any insert or remove actions. The insertion/removal block 456 performs the data insertion of any insert actions and the data removal of any remove actions according to information received from the insert/remove block 446 of the operation stage 404. The control block 458 controls the operation of the insertion/removal block 456. The insertion/removal block 456 may serially receive potentially modified data chunks of a network packet from the edit block 432 of the operation stage 404. While a data insertion or data removal is not in progress, the data chunks received by the insertion/removal block 456 are the data chunks output by the insertion/removal block 456.

Each insert action and each remove action of a textual language specification may optionally include a guard condition. An action that does not include a guard condition is always enabled, and an action with a guard condition is enabled when the guard condition is determined to have an enabling value.

For a remove action that is enabled, upon receiving the data chunk of the network packet that includes the position of the data removal, the insertion/removal block 456 may output a partial data chunk that includes the portion of the data chunk before the position of the data removal. The insertion/removal block 456 may suspend outputting data chunks until the specified amount of data is removed and the insertion/removal block 456 may subsequently resume outputting data chunks of the network packet. The first data chunk output on resuming output may be a partial data chunk. In one embodiment, multiples of a byte may be inserted and removed and the insertion/removal block 456 outputs a number of bytes in parallel for each word of the burst of a data chunk. Each byte that is output may have a corresponding data valid signal to indicate whether the byte is included in a full or partial data chunk.

For an insert action that is enabled, upon receiving the data chunk of the network packet that includes the position of the data insertion, the insertion/removal block 456 may output a full data chunk and then switch to outputting the insertion data. The final data chunk of the insertion data may be a partial data chunk and the insertion/removal block 456 may subsequently resume outputting full data chunks of the network packet. During the data insertion, the insert/remove stage 406 may stall the operation stage 404 and the look-ahead stage 402 to prevent data loss.

The interleave stage 408 may ensure that the modified network packet follows rules for interleaving network packets. The interleave stage 408 may eliminate the gaps in the valid data from any partial data chunks created by the insert/remove stage 406. The interleave stage 408 may divide a network packet into multiple bursts to satisfy the interleaving rules. The interleave stage 408 may be omitted for a textual language specification that only inserts and removes data that is always a multiple of the size of the data chunk. The interleave stage 408 may be eliminated along with the insert/

remove stage 406 for a textual language specification that does not include any insert or remove actions.

The interleave block 460 may generate each data chunk that is output from the pipeline 400 by selecting from either the chunk buffer 462 or the insertion/removal block 456 of the insert/remove stage 406. The chunk buffer 462 may contain a portion of the preceding network packet. The control block 464 may be a state machine that controls the operation of the chunk buffer 462 and the interleave block 460.

FIG. 5 is a flow diagram of a process for generating a pipeline for processing network packets in accordance with various embodiments of the invention. The processing of network packets by the pipeline may be specified in a textual language specification, and this specification may be analyzed to generate a specification of the pipeline in a hardware description language. Synthesis tools may generate a hardware implementation of the pipeline from the generated specification. For example, ISE Foundation available from Xilinx, Inc. may be used to create an implementation of the pipeline in a programmable logic device.

At step 502, the specification of the processing of the pipeline may be analyzed to determine the number of memories accessed by the pipeline. For each memory, a dependency graph may be created for scheduling the memory reads that access the memory. Each memory read may require an optional address and an optional guard condition, and each memory read may provide a result value. The addresses and guard conditions may be calculated from certain fields of a network packet and a node may be added to the dependency graph for each of these fields and for each read result from the memory. These nodes are marked with an availability time relative to a start of the network packet. An availability time for a field may be readily calculated from a position of the field within the network packet and a data width of an input of the pipeline for receiving the network packet. For a read result that is dependent on a set of fields for calculating the address and guard condition, the read result may be marked with an availability time that is the sum of the latency for the memory read and the latest availability time of the fields in the set. It will be appreciated that for a memory read without an address and without a guard condition, such as a read from a configuration memory, the result of the memory read may be marked as available at time zero corresponding to the beginning of the network packet.

The optional address and optional guard condition of each memory read may also be dependent on a local variable that is further dependent on certain fields of a network packet, global variables, or the results of other memory reads. Additional nodes may similarly be added and marked in the dependency graph for these indirect dependencies.

Once each dependency graph is resolved to assign an availability time to each node in the graph, each memory read is scheduled to provide the read result at the availability time of the read result. For a memory having a non-zero read latency, the memory read may be issued in advance to provide the read result at the required time. Because a result from a memory read may be used to determine the address or guard condition of another memory read, perhaps indirectly via a local variable, iteration may be required to completely resolve the dependency graph.

At step 504, a storage class may be determined for each field used to generate the address and guard conditions of the memory reads. For a field with an availability time that matches the issue time of every memory read that uses the field, the storage class may be a wire class indicating a direct connection. For a field with an availability time in advance of the issue time of every memory read that used the field, the

11

storage class may be a register class. Otherwise, the storage class may be a register with a bypass path.

At step **506**, a depth of the delay FIFO of the look-ahead stage may be determined. The delay FIFO may receive a stream of network packets from an input of the pipeline and the delay FIFO may output the stream of network packets after a fixed time delay, which is set by the depth of the delay FIFO.

In one embodiment, the depth may be selected to have the delay FIFO output the start of a network packet at a time when the results are available for all of the memory reads. Thus, the depth of the delay FIFO may equal the largest value of the availability times of the read results from the dependency graph or graphs. The delay FIFO may be regarded as effectively advancing the memory read results to ensure availability at time zero when the start of the network packet is output from the delay FIFO.

In another embodiment, a usage time and a difference between the usage time and the availability time is determined for each memory read result and the depth of the delay FIFO may equal the largest difference for the memory read results that have a usage time before the availability time.

At step **508**, a dependency graph may be created for scheduling the modifications of a network packet. A network packet may be modified by changing a field of the network packet, inserting data into a network packet, or removing data from a network packet. Various fields of the network packet may be used to determine, for example, a modification value for changing a field of a network packet or a packet-dependent length of data to be inserted or removed at a field of the network packet. Certain fields of the network packet may also be used to determine a guard condition for enabling or disabling a field modification or a data insertion or deletion.

A node may be added to the dependency graph for each of these fields and these nodes may be marked with an availability time of the receiving the field relative to a start of the network packet. Additional nodes may be added for the calculated modifications and these additional nodes may be marked with corresponding availability times. From the dependency graph, the modifications of the network packet may be scheduled at the availability times determined for the guard conditions of the modifications, the modification values, the variable locations of data insertion or removal, the variable lengths of data insertion or removal, and the insertion data values.

The FIFO depth selected at step **506** may provide inadequate advancement of some of the fields used for packet modifications. At step **510**, these additional fields needing look-ahead may be identified, and the FIFO depth determined at step **506** may be increased and consequently the storage classes determined at step **504** may be adjusted. At step **512**, the storage classes are determined for the fields used for packet modifications.

At step **514**, a specification of the look-ahead, operation, insert/remove, and interleave stages may be generated in a hardware description language.

FIG. **6** is a flow diagram of a process for simulating a processor of network packets in accordance with various embodiments of the invention. Network packets may be generated to test the processor of network packets during simulation. The network packets may be generated by selecting the values of the fields of the network packet.

At step **602**, a specification is input of the network packet processor. The specification includes one or more actions for modifying the network packets. Certain of the actions may include a guard condition for enabling and disabling the action. An action with a guard condition may be enabled or

12

disabled as a function of certain fields of the network packets. At step **604**, the specification is translated into a simulator of the network packet processor.

At step **606**, values are determined for certain fields of the network packets. Field values are determined that enable and disable the actions that have a guard condition. For example, a guard condition of an action may enable the action when two fields have equal values in a network packet. The same random value may be selected for both of the fields to enable the action and differing values may be randomly selected to disable the action.

At step **608**, the network packets are generated. For each network packet, values are selected for the fields that are referenced by the guard conditions of the actions. Each network packet may be generated by selecting the values of these fields from the field values generated at step **606**. For fields that are not referenced by the guard conditions of the actions, the field may be assigned a random value or a default value.

At step **610**, the network packet processor is simulated using the simulator generated at step **604**. The network packets generated at step **608** are supplied to the simulation of the network packet processor. During simulation, the network packets may be modified according to the specification of the processing of the network packet processor. The simulation may display the modified network packets or another predicted response of the network packet processor. An action having a guard condition may be enabled for some of the network packets, and the action may be disabled for the remaining network packets. Thus, the behavior of the network packet processor may be simulated for network packets that enable and disable the actions that have guard conditions.

The simulation also simulates the modification of the network packets for any actions that do not have a guard condition. Actions without a guard condition are effectively enabled for every network packet.

FIG. **7** is a flow diagram of a process **700** for determining values of certain fields in a type of network packets in accordance with various embodiments of the invention. Field values are determined that enable and disable the actions with guard conditions.

Process **700** starts at decision **702**. Process **700** ends at decision **702** when all actions are processed. If there an action to be processed, process **700** proceeds from decision **702** to decision **704**. Decision **704** checks whether the current action has a guard condition. If the current action does not have a guard condition then process **700** returns to decision **702** to determine whether there is another action that needs processing. Process **700** proceeds to decision **706** if the current action has a guard condition.

Decision **706** checks the format of the guard condition of the current action. If the guard condition is a comparison between a constant and one field of the type of network packets, then process **700** proceeds to step **708**. Otherwise, process **700** proceeds to decision **710** to determine whether the guard condition matches another format.

At step **708**, field values are determined that enable and disable the current action. The value for enabling the current action sets the field of the comparison to the constant of the comparison, and the value for disabling the current action sets the field of the comparison to a value other than the constant of the comparison. The value for disabling the current action may be a randomly selected value that differs from the constant of the comparison.

Decision **710** checks whether the guard condition of the current action matches a comparison between two fields of the type of the network packets. If the guard condition is a comparison between two fields, process **700** proceeds to step

13

712. Otherwise, process 700 continues checking the guard condition against additional formats (not shown). It will be appreciated that a guard condition for an action may have many possible formats, including complex expressions referencing many fields of the type of network packets. After the format of the guard condition is recognized and corresponding enabling and disabling values are determined for the guard condition, process 700 proceeds to decision 714.

At step 712, values for the fields of the type of network packets are determined that enable and disable the current action for the guard condition that compares two fields. The value for enabling the current action provides the same value for both of the fields of the comparison. The value for disabling the current actions sets the two fields of the comparison to differing values. Thus, the enabling and disabling values may each include values for two of the fields of the type of the network packets. The field values for the enabling and disabling values may be selected randomly.

Decision 714 checks whether all of the fields are processed for the enabling and disabling values of the current action. If either the enabling value and/or the disabling value of the current action provides a value for a field that is not yet processed, process 700 proceeds to decision 716. Otherwise, process 700 returns to decision 702 to process the next action.

Decision 716 checks whether the current field of the enabling and/or disabling value is already included in a listing of field values. If the current field is not included in the listing, process 700 proceeds to step 718. If the current field is included in the listing, process 700 proceeds to decision 720. At step 718, a new entry is created in the listing for the current field and this field may be marked with an identifier of the field.

Decision 720 checks whether the enabling value for the current field is already included in the entry of the listing for the current field. If so, process 700 proceeds to step 722 and otherwise process 700 proceeds to step 724. At step 724, the value of the current field within the enabling value is added to the listing entry for the current field, and a weight for the value is initialized to a weight of one. At step 722, the weight for the existing value of the current field is incremented.

It will be appreciated that a field that is included in an enabling value might not be included in a disabling value, or vice versa. For example, a guard condition may enable an action when two fields of the type of network packets are both equal a value of one. The enabling value may set both the first and second fields to a value of one. However, the disabling value may set the first field to a value of zero without specifying any value for the second field. Thus, if the current field does not have a value within the enabling value, process 700 might proceed directly from decision 720 to decision 726.

Decision 726 checks whether the disabling value for the current field is already included in the entry of the listing for the current field. If so, process 700 proceeds to step 728 and otherwise process 700 proceeds to step 730. At step 730, the value of the current field within the disabling value is added to the listing entry for the current field, and a weight for the value is initialized to a weight of one. At step 728, the weight for the existing value of the current field is incremented.

FIG. 8 is a flow diagram of a process 800 for generating the network packets in accordance with various embodiments of the invention. A sequence of network packets may be generated with the fields of the network packets set to enable and disable the actions of a specification of the processing by a processor of network packets.

Decision 802 checks whether more network packets should be generated. To generate more network packets, process 800 proceeds to decision 804 to generate the next net-

14

work packet. Decision 804 checks whether the network packet includes more fields that should be set to a value. If every field has been set to a value in the network packet, process 800 returns to decision 802 to check whether to generate another network packet. Otherwise, process 800 proceeds to decision 806.

Decision 806 checks whether the current field of the current network packet has a corresponding entry in a listing of values for certain fields of the type of network packet. This listing may be generated by the process 700 of FIG. 7. If the field is included in the listing, then process 800 proceeds to step 808; otherwise, process 800 proceeds to step 810.

For a field having an entry in the listing, the entry may include multiple possible values for the field and a weight for each value. The weights may specify a percentage of the time the corresponding value may be selected for the field of the network packet. The sum of the weights may equal one hundred percent or another value. To select one of the values in the entry at step 808, a random number may be generated between 0 and the sum of the weights. At step 812, the field of the network packet may be set to the value that corresponds to the random number. The value of this field may tend to enable or disable an action of the specification of the processing of the network packet processor.

At step 810, a value is selected for a field that might not be referenced by any action of the specification of the processing of the network packet processor. To define the value for the field of the network packet, the field may be set to a default value or a random value.

Those skilled in the art will appreciate that various alternative computing arrangements, including one or more processors and a memory arrangement configured with program code, would be suitable for hosting the processes and data structures of the different embodiments of the present invention. In addition, the processes may be provided via a variety of computer-readable media or delivery channels such as magnetic or optical disks or tapes, electronic storage devices, or as application services over a network.

While particular embodiments of the present invention have been shown and described, it will be obvious to those skilled in the art that changes and modifications may be made without departing from this invention in its broader aspects and, therefore, the appended claims are to encompass within their scope all such changes and modifications as fall within the true spirit and scope of this invention.

The invention claimed is:

1. A processor-based method for simulating a processor of network packets, comprising:

inputting a specification in a textual language of the processor of the network packets, the specification including a plurality of actions specifying a modification of the network packets by the processor, wherein each of at least one of the actions includes a respective guard condition, the respective guard condition for enabling and disabling the action in response to at least one of a plurality of fields of a type of the network packets;

translating the specification of the processor from the textual language to a simulator of the processor in a general-purpose programming language;

for each of the at least one action, determining a first and second value of the at least one field of the respective guard condition of the action, wherein the respective guard condition enables the action in response to the first value of the at least one field and the respective guard condition disables the action in response to the second value of the at least one field;

15

generating each of the network packets of the type including, for each field that is included in the at least one field of the respective guard condition of the at least one action, selecting a value of the field in the network packet from a plurality of values of the field within the first and second values of the at least one field of the respective guard condition of the at least one action; and simulating the modification of the network packets in the simulator, including displaying a result of the modification on a user interface, wherein the generating of each of the network packets of the type comprises:

generating a listing including a respective entry for each field that is included in the at least one field of the respective guard condition of the at least one action, wherein the respective entry for the field includes the plurality of values that includes each value of the field within the first and second values of the at least one field of the respective guard condition of each of the at least one action; and

generating a value for each field in the network packet, wherein, in response to the listing including the respective entry for the field, the value of the field is a selected one of the plurality of values in the respective entry.

2. The processor-based method of claim 1, wherein the actions of the specification include at least one action without a respective guard condition for enabling and disabling the action.

3. The processor-based method of claim 1, wherein the determining of the first and second values of the at least one field of the respective guard condition of the action includes, for the respective guard condition of the action that compares a constant and the at least one field that is one of the fields of the type of the network packets, selecting the constant for the first value of the at least one field of the respective guard condition of the action and selecting a value other than the constant for the second value of the at least one field of the respective guard condition of the action.

4. The processor-based method of claim 1, wherein the determining of the first and second values of the at least one field of the respective guard condition of the action includes, for the respective guard condition of the action that compares a first and second field of the type of the network packets, selecting a third value of both the first and second fields as the first value of the at least one field, and selecting a fourth value of the first field and a fifth value of the second field as the second value of the at least one field, the fourth and fifth values being different.

5. The processor-based method of claim 1, wherein the generating of the listing includes generating the listing including the respective entry for the field that further includes an identifier of the field.

6. The processor-based method of claim 1, wherein the generating of the value for each field in the network packet includes, in response to the listing not including a respective entry for the field, randomly selecting the value of the field in the network packet.

7. The processor-based method of claim 1, wherein the generating of the value for each field in the network packet includes, in response to the listing not including a respective entry for the field, selecting a default value for the value of the field in the network packet.

8. The processor-based method of claim 1, wherein the generating of the value for each field in the network packet includes, in response to the listing including the respective entry for the field, randomly selecting the selected value from the plurality of values in the respective entry.

16

9. The processor-based method of claim 8, wherein the generating of the listing further includes generating the listing including the respective entry for the field that further includes respective weights of the plurality of values in the respective entry.

10. The processor-based method of claim 9, wherein the randomly selecting of the selected value from the plurality of values in the respective entry includes randomly selecting the selected value from the plurality of values in the respective entry with a likelihood in proportion to the respective weights of the plurality of values.

11. The processor-based method of claim 10, wherein the generating of the listing further includes setting the respective weight of each of the plurality of values of the respective entry for each field in the listing to a number of times that the value occurs for the field in the first and second values of the in the at least one field of the respective guard condition of the at least one action.

12. The processor-based method of claim 1, wherein displaying the result of the modification on the user interface includes displaying the modification of the network packets on the user interface.

13. A non-transitory program storage medium, comprising:

a processor-readable device configured with instructions for simulating a processor of network packets, wherein execution of the instructions by a general-purpose computer causes the general-purpose computer to perform operations including,

inputting a specification in a textual language of the processor of the network packets, the specification including a plurality of actions specifying a modification of the network packets by the processor, wherein each of at least one of the actions includes a respective guard condition, the respective guard condition for enabling and disabling the action in response to at least one of a plurality of fields of a type of the network packets;

translating the specification of the processor from the textual language to a simulator of the processor in a general-purpose programming language;

for each of the at least one action, determining a first and second value of the at least one field of the respective guard condition of the action, wherein the respective guard condition enables the action in response to the first value of the at least one field and the respective guard condition disables the action in response to the second value of the at least one field;

generating each of the network packets of the type including, for each field that is included in the at least one field of the respective guard condition of the at least one action, selecting a value of the field in the network packet from a plurality of values of the field within the first and second values of the at least one field of the respective guard condition of the at least one action; and simulating the modification of the network packets in the simulator, including displaying a result of the modification on a user interface,

wherein the generating of each of the network packets of the type comprises:

generating a listing including a respective entry for each field that is included in the at least one field of the respective guard condition of the at least one action, wherein the respective entry for the field includes the plurality of values that includes each value of the field within the first and second values of the at least one field of the respective guard condition of each of the at least one action; and

17

generating a value for each field in the network packet, wherein, in response to the listing including the respective entry for the field, the value of the field is a selected one of the plurality of values in the respective entry.

14. The program storage medium of claim 13, wherein the determining of the first and second values of the at least one field of the respective guard condition of the action includes, for the respective guard condition of the action that compares a constant and the at least one field that is one of the fields of the type of the network packets, selecting the constant for the first value of the at least one field of the respective guard condition of the action and selecting a value other than the constant for the second value of the at least one field of the respective guard condition of the action.

15. The program storage medium of claim 13, wherein the determining of the first and second values of the at least one field of the respective guard condition of the action includes, for the respective guard condition of the action that compares a first and second field of the type of the network packets, selecting a third value of both the first and second fields as the first value of the at least one field, and selecting a fourth value of the first field and a fifth value of second field as the second value of the at least one field, the fourth and fifth values being different.

16. The program storage medium of claim 13, wherein the generating of the value for each field in the network packet includes, in response to the listing not including a respective entry for the field, randomly selecting the value of the field in the network packet.

17. The program storage medium of claim 13, wherein the generating of the value for each field in the network packet includes, in response to the listing including the respective entry for the field, randomly selecting the selected value from the plurality of values in the respective entry.

18. A system for simulating a processor of network packets, comprising:

means for inputting a specification in a textual language of the processor of the network packets, the specification including a plurality of actions specifying a modification of the network packets by the processor, wherein each of at least one of the actions includes a respective guard

18

condition, the respective guard condition for enabling and disabling the action in response to at least one of a plurality of fields of a type of the network packets;

means for translating the specification of the processor from the textual language to a simulator of the processor in a general-purpose programming language;

for each of the at least one action, means for determining a first and second value of the at least one field of the respective guard condition of the action, wherein the respective guard condition enables the action in response to the first value of the at least one field and the respective guard condition disables the action in response to the second value of the at least one field;

means for generating each of the network packets of the type including, for each field that is included in the at least one field of the respective guard condition of the at least one action, means for selecting a value of the field in the network packet from a plurality of values of the field within the first and second values of the at least one field of the respective guard condition of the at least one action; and

means for simulating the modification of the network packets in the simulator, including means for displaying a result of the modification on a user interface,

wherein the means for generating of each of the network packets of the type comprises:

means for generating a listing including a respective entry for each field that is included in the at least one field of the respective guard condition of the at least one action, wherein the respective entry for the field includes the plurality of values that includes each value of the field within the first and second values of the at least one field of the respective guard condition of each of the at least one action; and

means for generating a value for each field in the network packet, wherein, in response to the listing including the respective entry for the field, the value of the field is a selected one of the plurality of values in the respective entry.

* * * * *