US009270517B1

(12) **United States Patent**
Attig et al.

(10) **Patent No.: US 9,270,517 B1**
(45) **Date of Patent: Feb. 23, 2016**

(54) **TUPLE CONSTRUCTION FROM DATA PACKETS**

(71) Applicant: **Xilinx, Inc.**, San Jose, CA (US)

(72) Inventors: **Michael E. Attig**, Sunnyvale, CA (US); **Gordon J. Brebner**, San Jose, CA (US)

(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 248 days.

(21) Appl. No.: **13/789,331**

(22) Filed: **Mar. 7, 2013**

(51) **Int. Cl.**

H04J 3/00	(2006.01)
H04L 29/06	(2006.01)
H04L 12/851	(2013.01)
H04W 28/06	(2009.01)

(52) **U.S. Cl.**
CPC ***H04L 29/0653*** (2013.01); ***H04L 69/22*** (2013.01); *H04L 47/2441* (2013.01); *H04L 47/2483* (2013.01); *H04L 69/04* (2013.01); *H04L 69/166* (2013.01); *H04W 28/06* (2013.01); *H04W 28/065* (2013.01)

(58) **Field of Classification Search**
CPC H04L 69/04; H04L 69/166; H04L 69/22; H04L 47/2441; H04L 47/2483; H04L 29/0653; H04W 28/06; H04W 28/065
USPC 370/473, 474, 476
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,778,530	B1 *	8/2004	Greene	370/389
7,100,078	B1 *	8/2006	Pass	714/18
8,358,653	B1	1/2013	Attig et al.	
8,385,340	B1	2/2013	Attig et al.	
8,443,102	B1	5/2013	Attig et al.	
8,625,438	B1	1/2014	Attig	
2003/0046429	A1 *	3/2003	Sonksen	709/246

OTHER PUBLICATIONS

"400 Gb/s Programmable Packet Parsing on a single FPGA", Michael Attig and Gordon Brebner, 2011, provided in IDS.*
Attig M. et al., "400 Gb/s Programmable Packet Parsing on a Single FPGA", 2011 Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, (ANCS '11), Oct. 3-4, 2011, pp. 12-23., Brookleyn, NY, US.

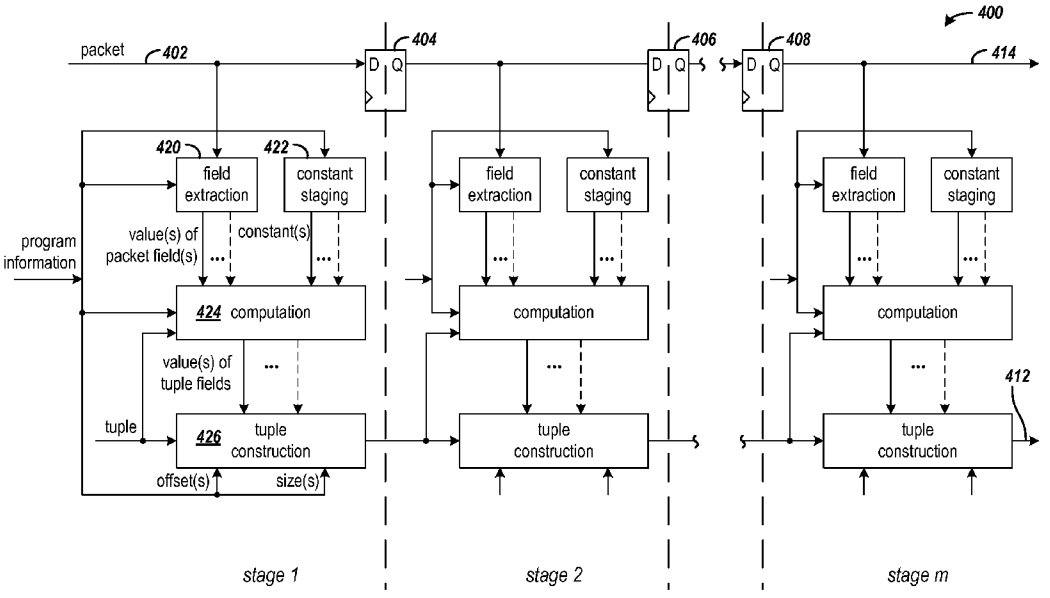
* cited by examiner

Primary Examiner — Peter Cheng
(74) *Attorney, Agent, or Firm* — LeRoy D. Maunu

(57) **ABSTRACT**

In one approach for processing a data packet, in at least one stage of a plurality of stages of a pipeline circuit, a respective packet field value is extracted from the data packet. In each stage of the plurality of stages, a respective tuple field value is inserted into a respective tuple register of the stage at a respective offset. The respective tuple field value in the at least one stage is based on the respective packet field value. In each stage of the plurality of stages except a last one of the stages, the contents of the respective tuple register of the stage are provided as input to a next one of the stages.

18 Claims, 6 Drawing Sheets



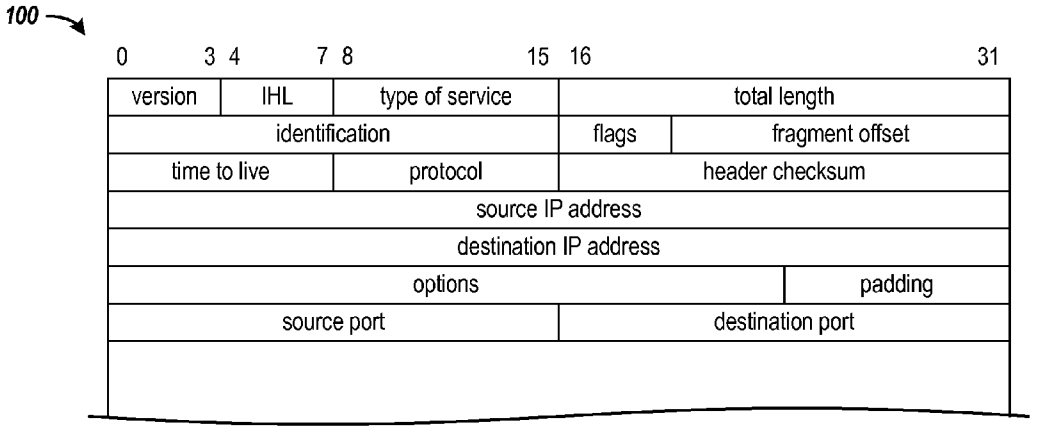


FIG. 1

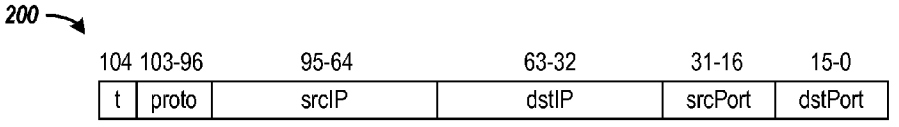


FIG. 2

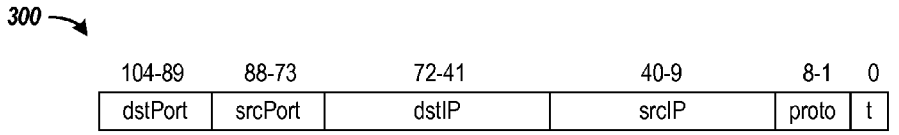


FIG. 3

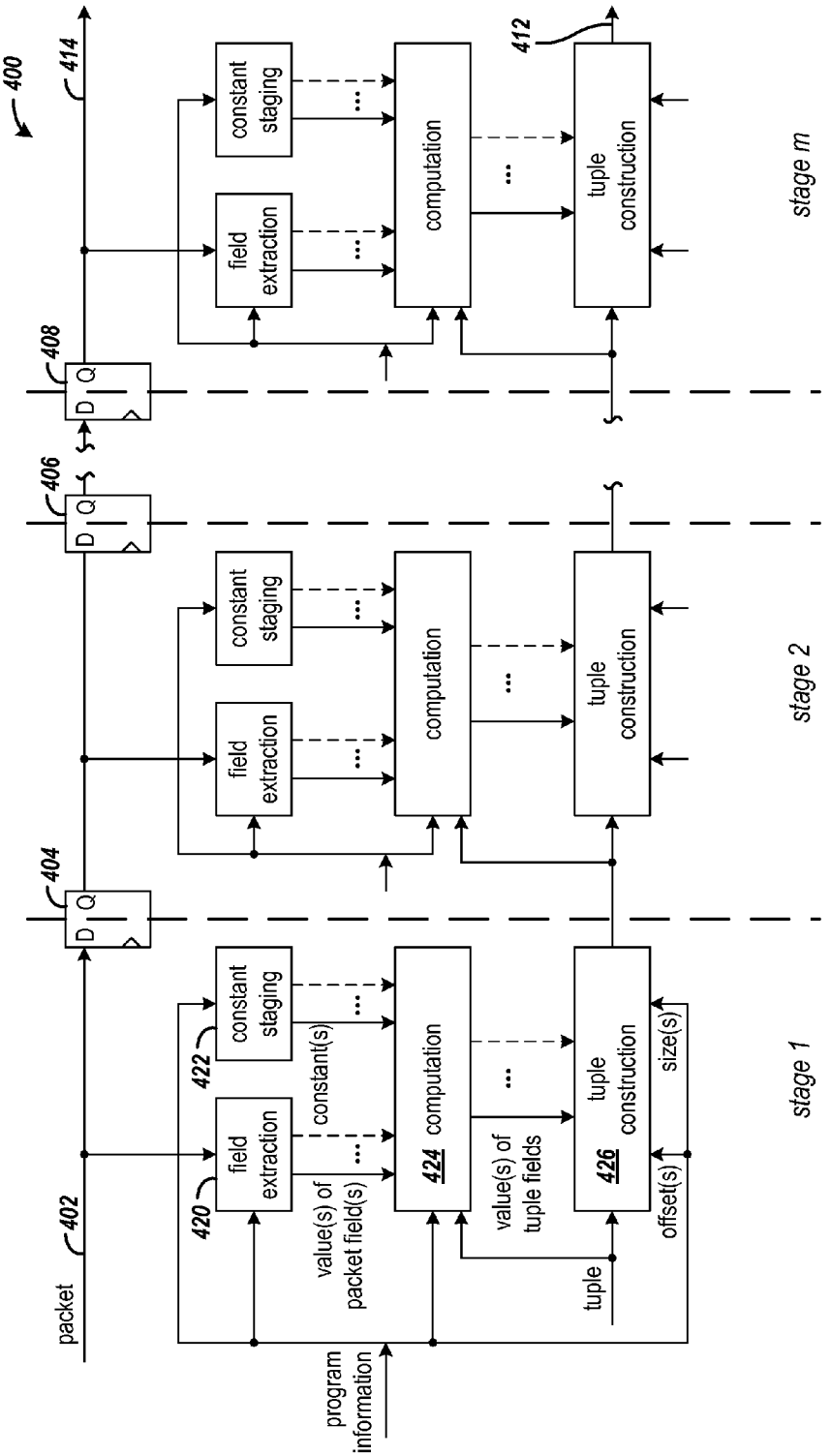

400
414
412
408
406
404
402
packet
program information
field extraction
constant staging
computation
tuple construction
420
422
424
426
constant(s)
value(s) of packet field(s)
value(s) of tuple fields
tuple
offset(s)
size(s)
stage 1
stage 2
stage m
D
Q

FIG. 4

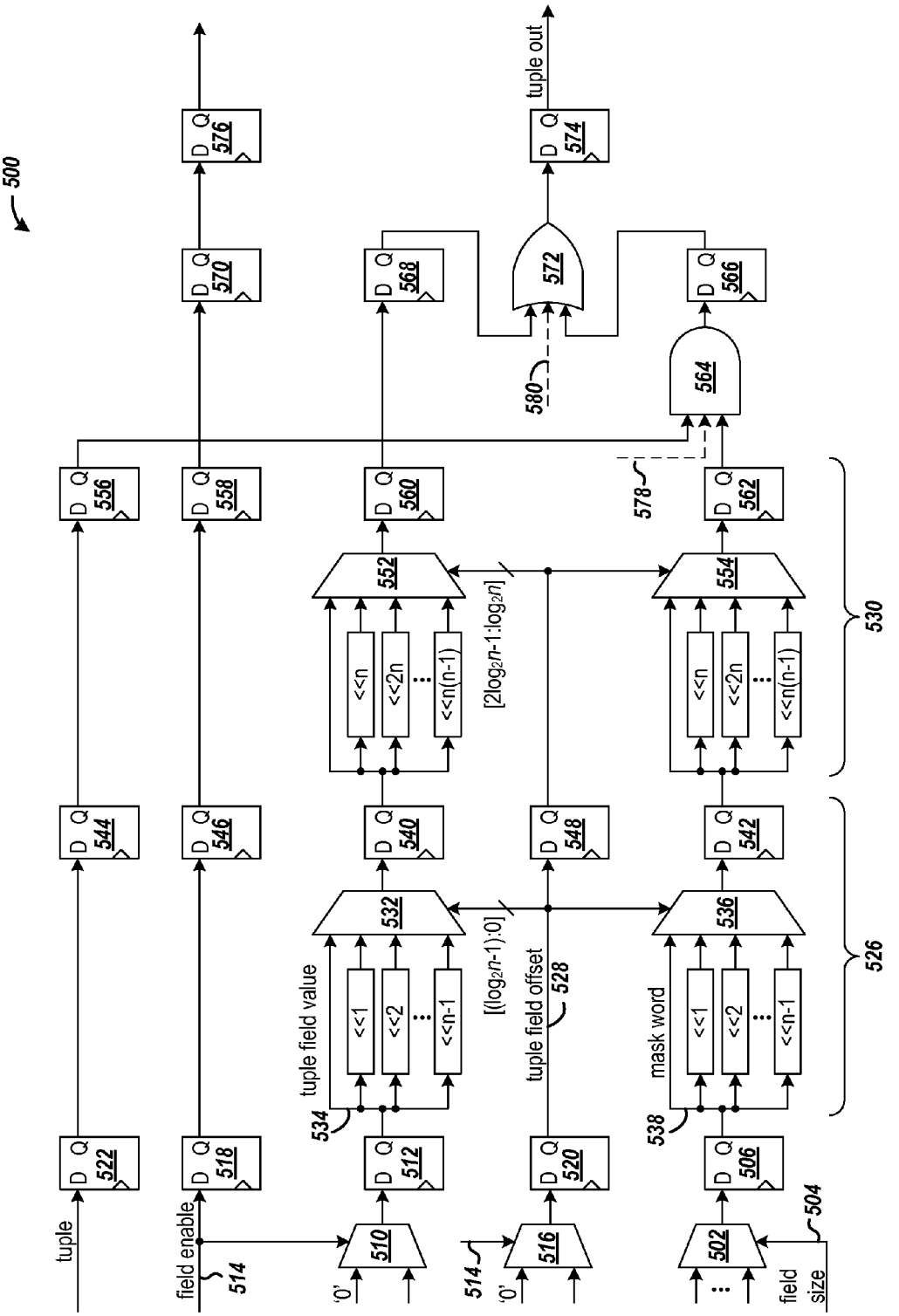

500
tuple
field enable
514
field size
504
'0'
'0'
tuple field value
tuple field offset
528
mask word
534
538
[(log2n-1):0]
[2log2n-1:log2n]
<<1
<<2
<<n-1
<<n
<<2n
<<n(n-1)
tuple out
580
578
526
530
502
506
510
512
516
518
520
522
532
536
540
542
544
546
548
552
554
556
558
560
562
564
566
568
570
572
574
576

FIG. 5

Step	ipType	proto	srcIP	dstIP	srcPort	dstPort
0	0	0x00	0x00000000	0x00000000	0xffff	0x8888
1	0	0x00	0x00000000	0x00000000	0xffff	0x8888
2	0	0x00	0x00000000	0x00000000	0xffff	0x8888
3	0	**0x00**	0x00000000	0x00000000	**0x0000**	0x8888
4	0	**0x06**	0x00000000	0x00000000	**0x0032**	0x8888
5	**0**	**0x06**	**0x00000000**	**0x00000000**	**0x0032**	**0x8888**

FIG. 6

Step	Field 1	Field 1 Size	Field 1 Offset	Field 1 Mask
0	0x0000 0000 0000 0000 0000 0000 0006	8	96	0x0000 0000 0000 0000 0000 0000 0000
1	0x0000 0000 0000 0000 0000 0000 0006	8	96	**0xFFFF FFFF FFFF FFFF FFFF FFFF FF00**
2	**0x0006 0000 0000 0000 0000 0000 0000**	8	96	**0xFF00 FFFF FFFF FFFF FFFF FFFF FFFF**
3	0x0006 0000 0000 0000 0000 0000 0000	8	96	0xFF00 FFFF FFFF FFFF FFFF FFFF FFFF
4	0x0006 0000 0000 0000 0000 0000 0000	8	96	0xFF00 FFFF FFFF FFFF FFFF FFFF FFFF
5	0x0006 0000 0000 0000 0000 0000 0000	8	96	0xFF00 FFFF FFFF FFFF FFFF FFFF FFFF

FIG. 7

Step	Field 2	Field 2 Size	Field 2 Offset	Field 2 Mask
0	0x0000 0000 0000 0000 0000 0000 0032	16	16	0x0000 0000 0000 0000 0000 0000 0000
1	0x0000 0000 0000 0000 0000 0000 0032	16	16	**0xFFFF FFFF FFFF FFFF FFFF FFFF 0000**
2	**0x0000 0000 0000 0000 0000 0032 0000**	16	16	**0xFFFF FFFF FFFF FFFF FFFF 0000 FFFF**
3	0x0000 0000 0000 0000 0000 0032 0000	16	16	0xFFFF FFFF FFFF FFFF FFFF 0000 FFFF
4	0x0000 0000 0000 0000 0000 0032 0000	16	16	0xFFFF FFFF FFFF FFFF FFFF 0000 FFFF
5	0x0000 0000 0000 0000 0000 0032 0000	16	16	0xFFFF FFFF FFFF FFFF FFFF 0000 FFFF

FIG. 8

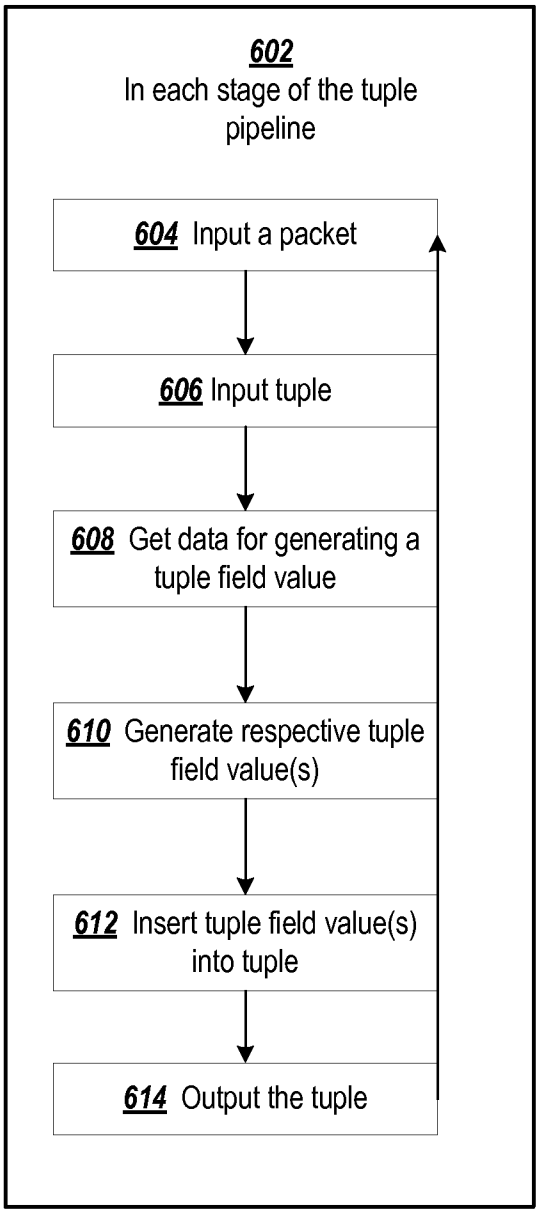

602
In each stage of the tuple pipeline
604 Input a packet
606 Input tuple
608 Get data for generating a tuple field value
610 Generate respective tuple field value(s)
612 Insert tuple field value(s) into tuple
614 Output the tuple

FIG. 9

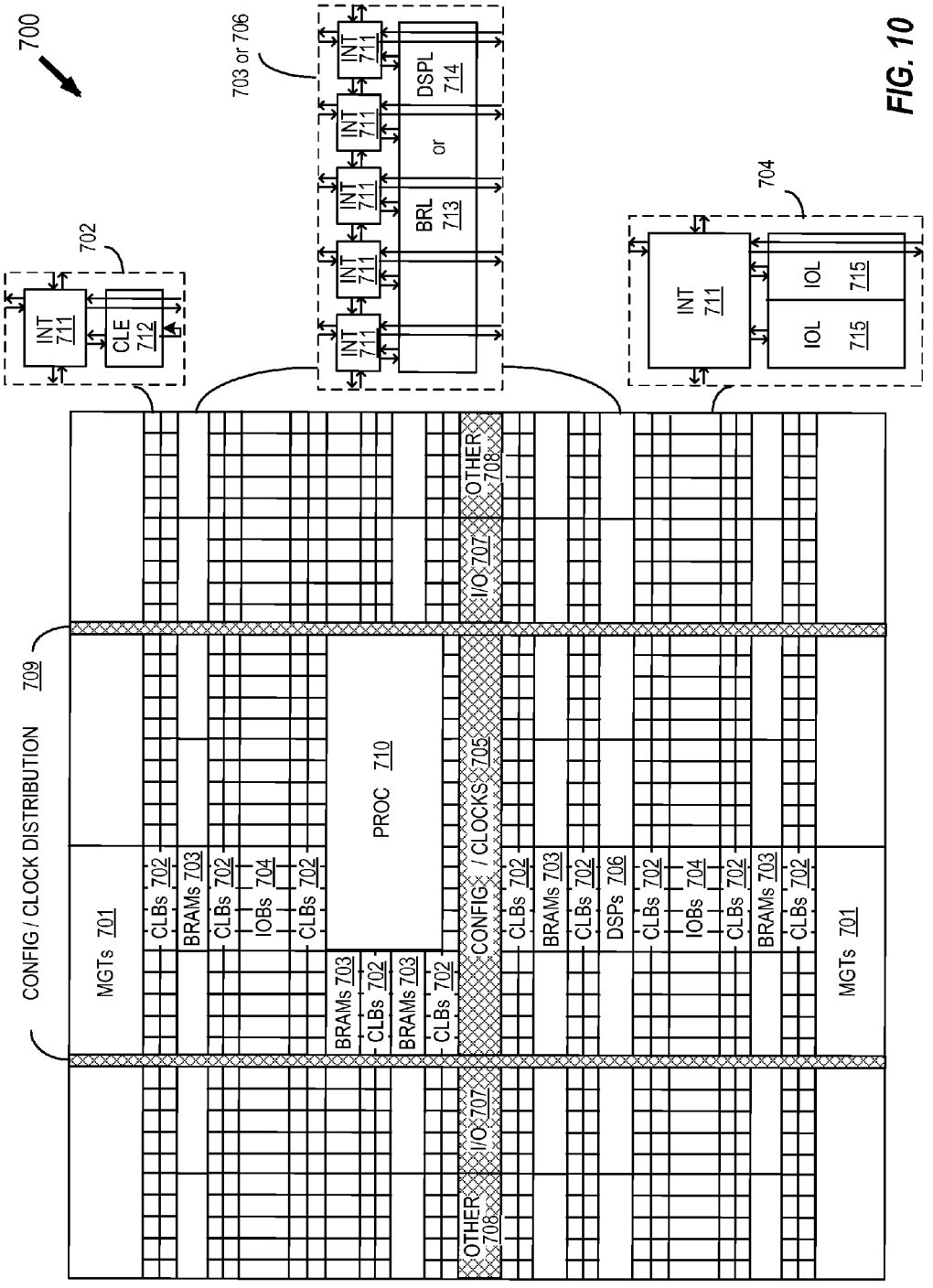

700
702
INT 711
CLE 712
703 or 706
INT 711
BRL 713
or
DSPL 714
704
IOL 715
CONFIG / CLOCK DISTRIBUTION 709
MGTs 701
CLBs 702
BRAMs 703
IOBs 704
DSPs 706
PROC 710
CONFIG / CLOCKS 705
I/O 707
OTHER 708

FIG. 10

TUPLE CONSTRUCTION FROM DATA PACKETS

FIELD OF THE INVENTION

The disclosure generally relates to packet processing and building tuples from the packets.

BACKGROUND

In some implementations, a network packet processor inputs a stream of network packets, manipulates the contents of the network packets, and outputs 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 common task in processing packets is to form a compact data tuple based on certain fields of a packet. The data tuple makes processing of the assembled data convenient. For example, in a packet classification task, certain address fields and/or type fields are extracted from a packet and then used together as a lookup key to determine the class of the packet. The particular fields and positions of the fields in the packet may vary depending on processing functions and protocols.

The data rate at which packets are transmitted presents challenges for processing the packets at a rate sufficient to keep pace with the data transmission rate. In packet processing applications, packets are streamed word-wise, for example using words that are 512-bits wide and achieving a 100 Gbps data rate. Each packet may be comprised of multiple 512-bit words. The fields of a packet that are used in constructing a tuple are generally located in different areas of the packet. Thus, the fields of a packet will be available at different discrete times. The times at which the fields become available is not necessarily static since packet structures can vary from packet to packet, such as with variable field sizes.

SUMMARY

A method for processing a data packet includes, in at least one stage of a plurality of stages of a pipeline circuit, extracting a respective packet field value from the data packet. In each stage of the plurality of stages, a respective tuple field value is inserted into a respective tuple register of the stage at a respective offset. The respective tuple field value in the at least one stage is based on the respective packet field value. In each stage of the plurality of stages except a last one of the stages, the contents of the respective tuple register of the stage are provided as input to a next one of the stages.

A packet processing circuit includes a plurality of pipeline stages. Each stage includes a field extraction circuit and a tuple construction circuit. The field extraction circuit is configured to receive a data packet and is configurable to extract none or a plurality of packet field values from the data packet. The tuple construction circuit is coupled to receive an input tuple and each packet field value from the field extraction circuit. The tuple construction circuit is configured to insert a respective tuple field value into the input tuple at a respective offset and output a tuple having the inserted respective tuple field value. The respective tuple field value is based on the at least one packet field value.

Other aspects and features will be recognized from consideration of the Detailed Description and Claims, which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

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

FIG. **1** shows the content and format of an IP packet;

FIG. **2** shows a first example tuple formed from fields extracted from the IP packet of FIG. **1**;

FIG. **3** shows a second example tuple formed from fields extracted from the IP packet of FIG. **1**;

FIG. **4** shows a circuit diagram of a pipeline circuit having m stages for extracting fields from an input packet and assembling the extracted fields into a tuple;

FIG. **5** shows an example implementation of a tuple construction circuit that inserts a field value into a tuple in a stage of the pipeline circuit of FIG. **4**;

FIG. **6** shows the steps of a tuple under construction;

FIG. **7** shows the construction of a first tuple field and the mask for the first tuple field;

FIG. **8** shows the construction of a second tuple field and the mask for the second tuple field;

FIG. **9** is a flowchart of an example process for constructing a tuple; and

FIG. **10** shows an example programmable integrated circuit (IC) on which the circuitry described herein may be implemented.

DETAILED DESCRIPTION OF THE DRAWINGS

To achieve a suitable level of performance and flexibility, it may be desirable to aggregate field values of packets into tuples at a high data rate. In addition, it may be desirable to programmably select fields from data packets and formats of the tuples. In one approach, a method of processing a data packet includes, in at least one stage of multiple stages of a pipeline circuit, extracting a respective packet field value from the multiple fields of the data packet. In each of the stages, a respective tuple field value is inserted into a respective tuple register of the stage at a respective offset. In at least one stage in which the value of a field is extracted, the respective tuple field value is based on the respective packet field value. Depending on application requirements, the tuple field value may also be based on one or more constants or one or more input tuple field values. In each stage except the last stage, the contents of the respective tuple register of the stage are provided as input to a next one of the stages. With the pipelined approach, a tuple can be produced from an input stream of data packets in every cycle. With parallel circuitry, multiple tuples could be generated.

FIGS. **1**, **2**, and **3** illustrate an example packet and two alternative tuples generated from data in the packet. Though the example is based on an Internet Protocol (IP) packet, it will be appreciated that the approaches described herein may be applied to a variety of different packet protocols.

FIG. **1** shows the content and format of an IP packet **100**. The first six 32-bit words are the header of the packet. The source port and destination port fields are part of the payload of the packet. Additional data in the payload portion and the trailer are not illustrated.

FIG. **2** shows a first example tuple **200** formed from 6 fields extracted from the IP packet of FIG. **1**. The t field is a type field the value of which is computed based on fields that

appear before an IPv4 or IPv6 header. For example, the value of the field may be based on a field called 'type' in the Ethernet header as:

```
If (Ethernet.type==0x800) //IPv4 type code
Tuple.t=0
Else if (Ethernet.type==0x86dd) //IPv6 type code
Tuple.t=1
Else
Tuple.t=0; //default case
```

The proto field in the tuple **200** corresponds to the protocol field in the IP header, the srcIP field in the tuple is the source IP address field from the packet **100**, the dstIP field in the tuple is the destination IP address field from the packet, the srcPort field is the source port field from the packet, and the dstPort field is the destination port field from the packet.

FIG. **3** shows a second example tuple **300** formed from the 6 fields extracted from the IP packet of FIG. **1**. The order of the fields in tuple **300** is different from the order of the fields in tuple **200**. The particular tuple structure depends on the requirements for high-speed processing of the tuple. Although the example tuples **200** and **300** show fields of the packet being copied from the packet to the designated positions in the tuples, the tuple field values may be computed using arithmetic or logic functions of combinations of the field values, local variables, constants, and/or tuple field values from a previous stage of the pipeline.

A tuple aggregation circuit is provided to construct tuples in a pipelined fashion. FIG. **4** shows a circuit diagram of a pipeline circuit **400** having m stages for extracting fields from an input packet and assembling the extracted fields into a tuple. Each stage inserts one or more tuple field value into the tuple. A full packet is input on line **402** and is registered between each of the stages in packet registers **404**, **406**, and **408**. Once the processing of a packet is complete in one stage, that packet is passed to the next stage for processing and a new packet is input. Thus, the stages are processing different packets concurrently, or different words of the same packet. The completed tuple for a packet is output on line **412** as the corresponding packet is output on line **414**.

Each stage of the pipeline circuit **400** includes a field extraction circuit **420**, a constant staging circuit **422**, a computation circuit **424**, and a tuple construction circuit **426**. Programmed control information is input to the circuit elements for controlling each circuit element. The programmed control information indicates which fields to extract from the packet, any constants to be used, the computation to be performed, and offsets and sizes of the tuple field values in the tuple. The programmed control information may be provided via a microprogramming control store (not shown), for example.

The field extraction circuit **420** is controllable to extract one or more fields from the input packet. For each field to be extracted by the field extraction circuit, the programmed control information indicates an offset of the field in the packet and a size of the field. For a tuple field value that is not based on a packet field, the input program information indicates to the field extraction circuit to not extract any fields from the packet. Further disclosure of a field extraction circuit is found in the co-pending patent application having Ser. No. 13/229,083, entitled, "CIRCUIT AND METHOD FOR EXTRACTING FIELDS FROM PACKETS, by Michael Attig, and assigned to Xilinx, Inc.; the entire contents of this co-pending application are incorporated by reference into this application. The extracted value(s) of the field(s) of the packet are output by the field extraction circuit and input to the computation circuit **424**.

The constant staging circuit **422** stages constant values for input to the computation circuit **424**. The programmed control information input to the constant staging circuit indicates which constant value, if any, is to be provided to the computation circuit. Depending on application requirements, multiple constant values may be provided to the computation circuit. The programmed control information input to the constant staging circuit may provide the constant values, or alternatively, reference constant values stored within the constant staging circuit. The time at which the constant value(s) is provided as input to the computation circuit coincides with the provision of the field value(s) as input to the computation circuit.

The computation circuit **424** computes the value of the tuple field to be inserted into the tuple based on registered packet field values, registered constant values, and/or a registered input tuple. The computation circuit may be an arithmetic logic unit that performs arithmetic and/or logic functions on designated operands. The operation(s) to be performed may be provided to the computation circuit as executable instructions. The instructions also indicate which registered values are the operands. A no-operation-type instruction may be used to indicate to the computation circuit that a registered value is to be output without changing its value. The computation circuit may provide values for multiple tuple fields depending on application requirements.

The tuple construction circuit **426** inserts the tuple field value(s) from the computation circuit **424** into the proper location(s) in the in-process tuple (the tuple being constructed). The offset(s) provided in the programmed control information indicates the proper location(s) of the tuple field value(s). The size(s) provided in the programmed control information indicates the number of bits occupied by the tuple field value(s). Once the tuple field value(s) is inserted in the tuple, the tuple and packet are forwarded to the next stage in the pipeline. Since packets are streamed word-wise, a tuple does not necessarily have to wait until the entire packet has been received to proceed to the next stage. Rather a tuple may be forwarded to the next stage once the word of the packet having the last needed packet field has been extracted and processed to create the tuple field value. If no field is extracted from an input packet to create any tuple field value, the tuple may be forwarded to the next stage at the same time the first packet word is forwarded to the next stage.

FIG. **5** shows an example implementation of a tuple construction circuit **500** that inserts a field value into a tuple in a stage of the pipeline circuit of FIG. **4**. The tuple construction circuit performs four main tasks. The first task is to create a mask of the size needed for the tuple field value to be inserted. For example, if the tuple field value is 16 bits, then a 16-bit mask is generated. Next the tuple field value and the mask are shifted to align with the proper position in the tuple. In the third task, the mask is applied to the tuple to clear the appropriate bits in the tuple for the tuple field value. The fourth task is to insert the tuple field value into the tuple.

The data path including elements **502**, **506**, **536**, **542**, **554**, **562**, **564**, and **566** may be viewed as a mask circuit within the tuple construction circuit, and the elements **510**, **512**, **532**, **540**, **552**, **560**, **568**, and **572** may be viewed as a tuple insertion circuit within the tuple construction circuit.

The proper size mask is created by selecting a mask word with multiplexer **502** from mask words having mask sizes that correspond to the different possible sizes of tuple fields. In an example implementation, the mask bits are logic 0 bits and are right aligned in a mask word having logic 1 bits in all other positions. For example, for a tuple field of size 8 bits, the rightmost 8 bits of the mask word selected by and output from

multiplexer **502** are logic 0 bits, and all other bits of the selected mask word are logic 1. The tuple field size signal **504** selects the proper mask word, and the selected mask word is stored in register **506**.

In parallel with the selection of the mask word, the tuple field value is input via multiplexer **510** and register **512**. Also, the field enable signal **514** provides the selection of the tuple field value via multiplexer **510** and the field offset via multiplexer **516**. The state of the field enable signal is stored in register **518**, and the field offset is stored in register **520**. The tuple being constructed is input to register **522** also in parallel with selection of the mask word.

The mask word and the tuple field value are shifted in two stages. In stage **526**, the tuple field value and the mask word are left shifted by a number of bits indicated by the low-order bits of the field offset **528**, and in stage **530** the output of the first shift stage is shifted by a number of bits indicated by the high-order bits of the field offset. In stage **526**, multiplexer **532** selects from inputs in which the tuple field value has been left shifted by 0 to n−1 bits. The notation "<<x" in the diagram indicates a circuit that left shifts the input by x bits. The input tuple field value **534** occupies the low-order (right-most) bits of the input word, and the other bits are logic 0. Logic 0 values are shifted in as the tuple field value is left shifted. The mask in the mask word is also left shifted, and multiplexer **536** selects the mask word that was shifted by the same number of bits as the tuple field value. The mask occupies the low-order bits in the input mask word **538**, and the other bits are logic 1. Logic 1 bits are shifted in as the mask is left shifted.

The low-order bits of the field offset are used to control the selections by multiplexers **532** and **536**. For selecting from words that have been left shifted from 0 to n−1 bits, bits **0** through $\log_2$ n−1 of the field offset are used.

The selected tuple field value is stored in register **540**, and the selected mask word is stored in register **542**. The tuple, field enable signal, and field offset are forwarded to registers **544**, **546**, and **548**, respectively, to maintain proper timing within the pipeline and allow the next tuple and tuple field value to be processed.

In stage **530**, the tuple field value and the mask are left shifted by a number of bits specified by the high-order bits of the field offset. In stage **530**, multiplexer **552** selects from inputs in which the tuple field value has been left shifted by 0, n, 2n, . . . n(n−1) bits, and multiplexer **554** selects from inputs in which the mask has been left shifted by 0, n, 2n, . . . n(n−1) bits. For the tuple field value, logic 0 bits are shifted in, and for the mask word, logic 1 bits are shifted in. For selecting from words that have been left shifted from 0, n, 2n, . . . n(n−1) bits, bits $\log_2$n through 2 $\log_2$ n−1 of the field offset are used. The tuple, field enable signal, selected tuple field value, and selected mask word are stored in registers **556**, **558**, **560**, and **562**, respectively.

The tuple from register **556** and the mask word from register **562** are input to AND circuit **564**, which clears the bits in the tuple for the tuple field value to be inserted. The output is stored in register **566**, and in parallel, the tuple field value from register **560** is stored in register **568**, and the field enable signal is stored in register **570**. The tuple with the cleared bits from register **566** and the tuple field value from register **568** are input to OR circuit **572**, which outputs the tuple with the tuple field value inserted at the proper offset in the tuple. The tuple is stored in register **574**, and in parallel, the field enable signal is forwarded for storage in register **576**. The tuple is then ready for the next stage (if any) of the pipeline circuit **400** of FIG. **4**. The field enable signal indicates availability of the tuple having the tuple field value inserted.

Multiple tuple fields may be inserted into a tuple in parallel in an example implementation. For each tuple field value to be inserted, the circuitry for shifting the tuple field value and constructing and shifting a mask would be replicated. The dashed line **578** input to AND circuit **564** represents the mask word having the shifted mask for the additional tuple field value. The dashed line **580** input to OR circuit **572** represents the additional shifted tuple field value.

FIGS. **6**, **7**, and **8** show an example in which two tuple field values are inserted into a tuple. FIG. **6** shows the tuple under construction in steps **0**-**5**; FIG. **7** shows the construction of tuple field **1** and the mask for tuple field **1**; and FIG. **8** shows the construction of tuple field **2** and the mask for tuple field **2**. In Step **0**, the initial input tuple is specified as having srcPort set to 0xFFFF and dstPort set to 0x8888, while all other tuple fields are initialized to 0. There are two tuple field values to insert into the tuple. The first tuple field value (field **1**) to be inserted is the value 0x06, which is 8 bits, and is to be placed at offset 96 (from the least significant bit position) in the tuple. The second tuple field value (field **2**) to be inserted is the value 0x0032, which is 16 bits, and is to be placed at offset 16. Thus, the first tuple field value is to be inserted as the proto tuple field, and the second tuple field value is to be inserted as the srcPort tuple field.

In Step **1**, the masks for fields **1** and **2** are constructed. This involves creating a mask of 0xFFFF FFFF FFFF FFFF FFFF FFFF FF00 for field **1** and a mask of 0xFFFF FFFF FFFF FFFF FFFF FFFF 0000 for field **2**. Note that the first mask clears 8 bits while the second mask clears 16 bits.

In Step **2**, the fields and masks are aligned to the appropriate position in the tuple being constructed by using the appropriate offset for the input field. The aligned field and mask values for field **1** are 0x0006 0000 0000 0000 0000 0000 0000 and 0xFF00 FFFF FFFF FFFF FFFF FFFF FFFF, respectively. The aligned field and mask values for field **2** are 0x0000 0000 0000 0000 0000 0032 0000 and 0xFFFF FFFF FFFF FFFF FFFF 0000 FFFF, respectively.

In Step **3** the masks are applied to the input tuple. This results in a change to the value held in the srcPort from 0xFFFF to 0x0000. There is no change to the proto field, because it was already at 0x00.

In Step **4** the new fields are inserted. This results in a value of 0x06 for the proto field and 0x0032 for the srcPort field.

In Step **5** the result is output. The final tuple value is 0x006 0000 0000 0000 0000 0032 8888.

FIG. **9** is a flowchart of an example process for constructing a tuple. The processing of blocks **604**-**614** is performed in each stage of the tuple pipeline as indicated by block **602**. At block **604** a packet is input. Depending on the application and particular tuple to be constructed, the packet may contain data to use in generating a tuple field value. A tuple is input at block **606**. The tuple that is input depends on the stage of the pipeline and on the application. For example, for the first stage of the pipeline, the input tuple may be a tuple with initialized values or a tuple input from elsewhere in a packet processing system. For other stages, the input tuple is the tuple from the previous stage of the tuple construction pipeline.

At block **608**, the data for generating a tuple field value is obtained. As described above, the data may be one or more fields extracted from the input packet, one or more constant values, or one or more input tuple field values. The tuple field value is generated at block **610**. The tuple field value may be an arithmetic or logic function of one or more packet field values, one or more constants, and/or one or more input tuple field values.

The tuple field value is inserted into the tuple at block **612**, and the tuple is output at block **614**. For stages other than the final stage, the tuple is output for processing by the next stage in the tuple construction pipeline, and for the final stage, the tuple is output from the pipeline.

FIG. **10** shows an example programmable integrated circuit (IC) on which the circuitry described herein may be implemented. The programmable IC of FIG. **10** is an FPGA. FPGAs can include several different types of programmable logic blocks in the array. For example, FIG. **10** illustrates an FPGA architecture (**700**) that includes a large number of different programmable tiles including multi-gigabit transceivers (MGTs **701**), configurable logic blocks (CLBs **702**), random access memory blocks (BRAMs **703**), input/output blocks (IOBs **704**), configuration and clocking logic (CONFIG/CLOCKS **705**), digital signal processing blocks (DSPs **706**), specialized input/output blocks (I/O **707**), for example, e.g., clock ports, and other programmable logic **708** such as digital clock managers, analog-to-digital converters, system monitoring logic, and so forth. Some FPGAs also include dedicated processor blocks (PROC **710**) and internal and external reconfiguration ports (not shown).

In some FPGAs, each programmable tile includes a programmable interconnect element (INT **711**) having standardized connections to and from a corresponding interconnect element in each adjacent tile. Therefore, the programmable interconnect elements taken together implement the programmable interconnect structure for the illustrated FPGA. The programmable interconnect element INT **711** also includes the connections to and from the programmable logic element within the same tile, as shown by the examples included at the top of FIG. **10**.

For example, a CLB **702** can include a configurable logic element CLE **712** that can be programmed to implement user logic plus a single programmable interconnect element INT **711**. A BRAM **703** can include a BRAM logic element (BRL **713**) in addition to one or more programmable interconnect elements. Typically, the number of interconnect elements included in a tile depends on the width of the tile. In the pictured FPGA, a BRAM tile has the same width as five CLBs, but other numbers (e.g., four) can also be used. A DSP tile **706** can include a DSP logic element (DSPL **714**) in addition to an appropriate number of programmable interconnect elements. An **10**B **704** can include, for example, two instances of an input/output logic element (IOL **715**) in addition to one instance of the programmable interconnect element INT **711**. As will be clear to those of skill in the art, the actual I/O pads connected, for example, to the I/O logic element **715** are manufactured using metal layered above the various illustrated logic blocks, and typically are not confined to the area of the input/output logic element **715**.

In the pictured FPGA, a horizontal area near the center of the die (shown shaded in FIG. **10**) is used for configuration, clock, and other control logic. Vertical areas **709** extending from this horizontal area are used to distribute the clocks and configuration signals across the breadth of the FPGA.

Some FPGAs utilizing the architecture illustrated in FIG. **10** include additional logic blocks that disrupt the regular row structure making up a large part of the FPGA. The additional logic blocks can be programmable blocks and/or dedicated logic. For example, the processor block PROC **710** shown in FIG. **10** spans several rows of CLBs and BRAMs.

Note that FIG. **10** is intended to illustrate only an exemplary FPGA architecture. The numbers of logic blocks in a row, the relative heights of the rows, the number and order of rows, the types of logic blocks included in the rows, the relative sizes of the logic blocks, and the interconnect/logic implementations included at the top of FIG. **10** are purely exemplary. For example, in an actual FPGA more than one adjacent row of CLBs is typically included wherever the CLBs appear, to facilitate the efficient implementation of user logic.

The methods and circuits are thought to be applicable to a variety of systems for constructing tuples. Other aspects and features will be apparent to those skilled in the art from consideration of the specification. The processes and circuits may be implemented as one or more processors configured to execute software, as an application specific integrated circuit (ASIC), or as a logic on a programmable logic device. It is intended that the described features and aspects be considered as examples only, with a true scope of the invention being indicated by the following claims.

What is claimed is:

1. A method of processing a data packet, comprising:

in at least one stage of a plurality of stages of a pipeline circuit, extracting a respective packet field value from the data packet;

in each stage of the plurality of stages:

inputting an in-process tuple into a respective tuple register;

inputting a respective programmable offset value;

creating in a mask register, a mask word having a subset of bits equal in number to a number of bits of a respective tuple field value and positioned in the mask word in response to the respective programmable offset value;

clearing by a first circuit, bits of the in-process tuple in the respective tuple register using the subset of bits in the mask word in the mask register;

inserting by a second circuit, the respective tuple field value based on the respective packet field value into the respective tuple register of the stage by replacing the cleared bits of the respective in-process tuple with the tuple field value; and

in each stage of the plurality of stages except a last one of the stages, providing contents of the respective tuple register of the stage as input to a next one of the stages.

2. The method of claim **1**, further comprising, computing the respective tuple field value in the at least one stage as a function of the respective packet field value.

3. The method of claim **2**, further comprising:

inputting a respective set of one or more constants to the at least one stage; and

computing the respective tuple field value in the at least one stage as a function of the respective packet field value and the respective set of one or more constants.

4. The method of claim **1**, wherein:

the extracting of the respective packet field value from the data packet in the at least one stage includes, extracting a respective set that includes two or more packet field values from the data packet; and

the inserting of the respective tuple field value into a respective tuple register in the at least one stage includes inserting the respective tuple field value based on the respective set of two or more packet field values.

5. The method of claim **1**, further comprising, in at least one stage of the plurality of stages, inserting two tuple field values into the respective tuple register in parallel.

6. The method of claim **1**, further comprising, computing the respective tuple field value in the at least one stage as a function of the respective packet field value and at least one tuple field value of the input from a previous one of the plurality of stages.

7. The method of claim 1, further comprising:

inputting a respective set of one or more constants to the at least one stage; and

computing the respective tuple field value in the at least one stage as a function of the respective packet field value, at least one tuple field value of the input from a previous one of the plurality of stages, and the respective set of one or more constants.

8. The method of claim 1, further comprising inputting a respective programmable field size indicative of a number of bits of the respective tuple field value.

9. The method of claim 1, wherein the creating the mask word includes:

selecting a mask word having the subset of bits in right-most bits of the mask word and storing the selected mask word in the mask register; and

shifting bits of the mask word a number of positions indicated by the programmable offset value.

10. A packet processing circuit, comprising:

a plurality of pipeline stages, each stage including:

a field extraction circuit configured to receive a data packet and configurable to extract none or a plurality of packet field values from the data packet; and

a tuple construction circuit coupled to receive an input tuple, a respective programmable offset value, and each packet field value from the field extraction circuit, the tuple construction circuit configured to insert a respective tuple field value based on the received packet field values into the input tuple at a respective offset and output a tuple having the inserted respective tuple field value;

wherein each tuple construction circuit includes:

a first circuit configured to:

create a mask word in a mask register having a subset of bits equal in number to a number of bits of the respective tuple field value and positioned in the mask word in response to the respective programmable offset value, and

clear bits of the input tuple using the subset of bits in the mask word; and

a second circuit configured to replace the cleared bits of the input tuple with the respective tuple field value.

11. The circuit of claim 10, wherein each stage further comprises a computation circuit coupled to the field extraction circuit, the computation circuit configured to compute the respective tuple field value as a function of the packet field values.

12. The circuit of claim 11, wherein each stage further comprises:

a constant staging circuit coupled to the computation circuit, the constant staging circuit configured to input a respective set of one or more constants;

wherein the computation circuit is configured to compute the respective tuple field value as a function of the packet field values and the respective set of one or more constants.

13. The circuit of claim 10, wherein:

the field extraction circuit is further configured to extract a respective set that includes two or more packet field values from the data packet; and

the tuple construction circuit is further configured to insert the respective tuple field value based on the respective set of two or more packet field values.

14. The circuit of claim 10, wherein the tuple construction circuit in at least one stage of the plurality of stages is further configured to insert two tuple field values into the input tuple in parallel.

15. The circuit of claim 10, wherein each stage further comprises a computation circuit coupled to the field extraction circuit, the computation circuit configured to compute the respective tuple field value as a function of the packet field values and at least one tuple field value of the input tuple.

16. The circuit of claim 10, further comprising:

a computation circuit coupled to the field extraction circuit; and

a constant staging circuit coupled to the computation circuit, the constant staging circuit configured to input a respective set of one or more constants;

wherein the computation circuit is configured to compute the respective tuple field value as a function of the packet field values, at least one tuple field value of the input tuple, and the respective set of one or more constants.

17. The circuit of claim 10, wherein each tuple construction circuit is responsive to a respective programmable tuple field size indicative of a number of bits of the respective tuple field value.

18. The packet processing circuit of claim 10, wherein the mask circuit is further configured to:

select a mask word having the subset of bits in right-most bits of the mask word;

store the selected mask word in the mask register; and

shift bits of the mask word a number of positions indicated by the programmable offset value.

* * * * *