(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 November 2009 (26.11.2009)

PCT

(10) International Publication Number
WO 2009/141677 A2

(51) International Patent Classification: Not classified

(21) International Application Number:
PCT/IB2008/003697

(22) International Filing Date:
30 June 2008 (30.06.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data-
12/124,225 21 May 2008 (21.05.2008) US

(71) Applicant: TPACK [DK/DK]; Hoerkaer 12a, DK-2730 Herlev (DK).

(72) Inventor: **LAURSEN, SOEREN**; Skolelodden 14, DK-3450 Allerod (DK).

(74) Agent: **TADAYON, BIJAN;** Maxva 1lueip Liie, 1 Ail-bermyrtle Rd., Potomac, MD 20854 (US).

(81) Designated States *(unless otherwise indicated, for every Mnd of national protection available):* AE, AG, AL, AM, AO, AT, AU, AZ, BB, BG, BH, BR, BW, BY, CA, CH, CN, CO, CR, CU, DE, DK, DO, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, **IL,** IN, IS, **JP,** KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, **TJ,** TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

– *without international search report and t0 be republished upon receip<fthat report (Rule 48.2(g))*

(54) Title: SYSTEM AND METHOD FOR APPLICATION OF HASH FUNCTION IN TELECOMMUNICATION AND NETWORKING

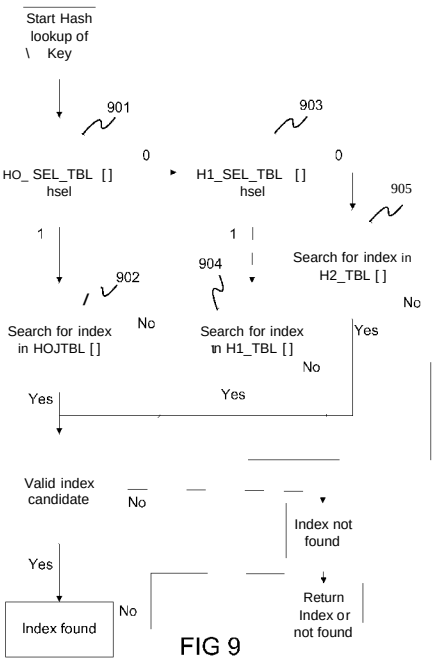

FIG 9

(57) Abstract: A novel hashing function and hashing collision resolution method are introduced that combine multiple known hashing resolution methods to achieve a very low collision probability that is specifically useful in lookup of long keys, such as (for example) the VLAN and MAC lookup in Ethernet switches. However, the system and method introduced here can be used in any networking and telecommunication systems.

WO 2009/141677 A2

System and Method for Application of Hash Function in Telecommunication and Networking

BACKGROUND OF THE INVENTION

Hashing is a well-known method in the computer industry, by which large number values (called Keys and denoted by "K") are compressed (Hashed) to smaller number values (called hashed numbers and denoted by "h"), in order to make it practical to use it as index for lookup tables. Using the key "K" directly as a lookup table index, without Hashing, requires a very large memory of size 2^n (2 to the power n, where n is the size of K in number of bits), which may have very little data scattered in it. Hashing will basically optimize the memory size requirements when large number values are to be used for indexing.

A well-known problem in Hashing is called "Collision". Collision happens when two or more Keys (K) are hashed to the same small Hashed value (h). Collision is possible because the mapping from the Keys to Hashed values is not a 1: 1 relationship, rather, it is an N: 1 relationship, in which many (N) Keys can map the same Hashed value (h).

There are also numerous Hashing functions that are used to convert Keys (K) to Hashed values (h). The ideal Hashing functions are those that are more random and have less correlation between Keys (K) and Hashed values (h). There are also numerous methods in the industry to resolve hashing collisions, which include, but are not limited to:

- Chaining
- Overflow areas
- Re-hashing
- Using neighboring slots (linear probing)
- Quadratic probing
- Random probing.

This invention uses unique Hashing functions and Hashing collision mechanisms that are novel. The main purpose of this invention is, for example, for Ethernet MAC address lookup and double VLAN tag lookups, but it can be used for other lookup purposes, as well.

SUMMARY OF THE INVENTION

This invention consists of unique hashing functions, as well as a novel hashing collision resolution method. The collision resolution method consists of populating the hash lookup tables, while doing the collision resolution.

The invention uses multiple hashing functions (example 3), to create multiple hashing tables. Starting for the first hash function, we do not do the lookup (i.e. do not have to read, to speed up the process), because a flag tells us if we have to go to the next hash function.

For the first function overflow, we try the overflow flag, and if that flag is zero, then we go to the next hash function, and repeat this process again and again.

Each hashing table entry consists of multiple index values (example 3). In order to select the candidate index, a matching of a 2-bit extract of the Key is done against the

stored extracts corresponding to each of those indexes. The candidate index goes through a final check, by doing a reverse lookup from Index to Key on a table that has stored the key.

Given 3 distinct keys, it is always possible to choose 2 bit positions that result in 3 distinct extracts. So, in each entry in H0_TBL, H1_TBL etc., the selected two-bit positions are stored, as are the resulting extracts, one for each candidate index. This allows the correct candidate index, and thus, the correct candidate record, to be selected, without having to actually inspect the keys in all 3 records.

BRIEF DESCRIPTION OF THE DRAWINGS

Figures 1-7 show the sequence of hash table and hash select table fill-outs for a VLAN and MAC lookup example.

Figure 8 shows an example of relationship between number of mappings, number of index entries, and the probability of collision.

Figure 9 is a data flow diagram of a hash lookup procedure.

Figure 10 shows the format of a hash table entry.

Figure 11 is a flow chart that shows the process of selecting the candidate index among 3 indices from a hash table entry.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

This invention comprises of two sets of functions/methods:

- Hashing functions
- Hashing collision resolution method

Hashing function: There are 6 hashing functions used in this invention. The hash Key (K) is of length "N". When a hash key has a shorter length than "N", then it is extended to length N by appending enough zeros to it. Two actual implementations are described below:

Implementation 1 (Double VLAN tag lookup in Ethernet): In this implementation, the hash key contains 37 non-zero bits that are zero, extended to 54 bits, when used in the hash function calculations. The bits in the key are:

- key [53:37] = AUO
- key [36:0] = rxVirtPort
- key [23:12] = VlanId
- key [11:0] = 0 or inner VLAN ID depending on lookup type

The first sets of hash functions for this invention are:

- VLAN_HO (key) = (key [44:30] RoL 10) ^ (key [29: 15] RoL 10) ^ key [14:0]
- VLAN_H1 (key) = (key [44:30] RoL 12) ^ (key [29: 15] RoL 11) ^ key [14:0]
- VLAN_H2 (key) = (key[53:36] RoL 0) ^ (key[35:18] RoL 0) ^ key [17:0]

In the above expressions, bit extraction from an expression is shown with [left : right]. E.g. if expl is 5 then expl [2:1] is 2. The symbol "^" above represents XOR function.

The symbol "RoL" represents the Rotate-Left (RoL) function. A Rotate-Left (RoL) function does a bit-wise rotate left for a number of bits. The result is thereby a

value with the same number of bits. E.g. if (0x1234 RoL 4) = 0x2341, and (0x1234 RoL 12) = 0x4123.

Implementation 2 (L2 MAC address lookup in Ethernet): In this implementation the hash key contains 61 non-zero bits that are zero extended to 72 bits, when used in the hash function calculations. The bits in the Key are:

- Key [71:61] = AUO
- Key [60:48] = Vsi
- Key [47:0] = MAC address, either SMAC or DMAC

The hash functions are:

- L2_H0 (Key) = (Key [63:48] RoL 7) ^ (Key[47:32] RoL 0) ^ (Key[31:16] RoL 0) ^ Key[15:0]
- L2_H1 (Key) = (Key [63:48] RoL 10) ^ (Key [47:32] RoL 2) ^ (Key [31:16] RoL 1) ^ Key [15:0]
- L2_H2 (Key) = (Key [71:54] RoL 0) ^ (Key[53:36] RoL 0) ^ (Key[35:18] RoL 0) ^ Key[17:0]

Hashing collision resolution method: Any hashing would naturally map many keys to the same hashed number. This is called collision. This invention uses a unique method for collision resolution, which consists of combining 3 methods, namely Chaining, Overflowing and re-hashing.

The collision resolution method consists of populating the hash lookup tables, while doing the collision resolution.

The invention uses multiple hashing functions (example 3), to create multiple hashing tables. Starting for the first hash function, we do not do the lookup (i.e. do not have to read, to speed up the process), because a flag tells us if we have to go to the next hash function.

For the first function overflow, we try the overflow flag, and if that flag is zero, then we go to the next hash function, and repeat this process as needed. The second stage is to do a lookup of the keys, based on the state built in the 1st stage.

Each hashing table entry consists of multiple index values (example 3). In order to select the candidate index, a matching of a 2-bit extract of the Key is done against the stored extracts corresponding to each of those indexes. The candidate index goes through a final check, by doing a reverse lookup from Index to Key on a table that has stored the key.

The purpose of the hash lookup is to find a unique index for a specific key. The mapping between key and index is configured in the main hash table called HO_TBL [ ] using the main hashing function called HO (key). Each entry in the hash tables can hold three index values (corresponding to 3 key values that hash to the same hash number), and additional information is provided in the entry in order to pick one of the index values based on the key. This part of the algorithm is known as "chaining" method. When more than 3 keys map to the same hash value, a second hash table called H1_TB[ ] is used that has similar structure to the main hash table, in which there are also 3 index entries for each hash value, but use a different hashing function called H1 (key). Note that the 3 corresponding index entries from previous table are remapped to 3 new entries in the new table. Similarly when more than 3 keys map to the same hash value in the 2nd

table, a 3rd hash table called H2_TB[ ] is used that has similar structure to the main hash table, in which there are also 3 index entries for each hash value, but use a different hashing function called H2(key). This part of the algorithm is a combination of "overflowing" and "re-hashing" methods. This hierarchy can continue to as many tables as desired. However, statistical calculations show that with 3 hashing tables, the possibility of collision can be reduced to a negligible value.

An example to illustrate the populating the hashing tables are illustrated in Figures 1 to 7. There are 3 hash tables in this example called HO_TBL [ ], Hl_TBL [ ] and H2_TBL [ ], as well as 2 hash selection tables called HO_SEL_TBL [ ] and Hl_SEL_TBL [ ] corresponding to hash tables HO_TBL [ ] and Hl_TBL [ ] respectively.

The hash tables HO_TBL [ ], Hl_TBL [ ] and H2_TBL [ ] hold the actual Key to Index mappings data, while the select tables, HO_SEL_TBL [ ] and Hl_SEL_TBL [ ], indicate if the corresponding hash table is to be used, or if the relation is to be found in the next hash table. There is no select table for hash table H2_TBL [ ], since H2_TBL[] is the last table. The hash selection table has one bit entry for every row of its corresponding hash table. A value of "1" indicates that the corresponding row is valid and a value of "0" indicates that the lookup must be done on the next hash table.

Seven key-to-index relations are inserted, starting with no relations in the tables. As illustrated in Figure 1, the first relation is inserted into entry 2 of HO_TBL [ ], determined by hash function HO(key), and takes the first of three free positions. As illustrated in Figure 2, the next relation is inserted into entry 4, where it also takes the first free positions. As shown in Figure 3, the third relation hits entry 2 again, taking the second position. As shown in Figure 4, then another relation hits entry 2 again, whereby

the last free position for entry 2 is taken. When the last relation hits entry 2 again, then there are no free positions in hash table HO_TBL [ ] entry. So as shown in Figure 5, all four relations that hit the same entry in HO_TBL [ ] through hash function HO (key) are redistributed to hash table H1_TBL [ ] using hash function H1 (key). The select table HO_SEL_TBL [ ] is then configured not to select hash table HO_TBL [ ] for entry 2 (this is done by setting the corresponding bit in HO_SEL_TBL [ ] to zero), since hash table H1_TBL [ ] is used, instead. Figure 6 shows that only entry 2 of hash table HO_TBL [ ] is affected by the congestion, so new relations can still be inserted in hash table HO_TBL [ ], as long as they do not hit entry 2. Figure 7 shows the final relation insertion that hits entry 2 of hash table HO_TBL [ ], and since the HO_SEL_TBL [ ] is zero for entry 2, it is inserted in hash table H1_TBL [ ].

The example shows that congestion in a hash table is handled by redistributing the entries in that table to the next hash table. Due to using different hash functions, the new distribution is likely to resolve the congestion. If, however, congestion in hash table H2_TBL [ ] occurs, then there is no resolution since there is no next level. In this case, it is not possible to insert the new relation. To limit the probability of such critical congestions, there are more hash table entries than the maximum number of key to index relations. The result is that the risk of critical congestion is very low. Figure 8 shows the probability of collision for double VLAN lookup, using the 3 hash tables. Both simulations and practical uses have shown that the probability of collisions is sufficiently low.

Assuming there are N hash tables and N-I select tables as explained above, in order to do a lookup of a certain key, first the key is hashed using first hash function and

the hash value is used to do a lookup on the first hash select table. If the corresponding select bit is "1", the candidate index should be searched in the first hash table, otherwise, the index is not in the first hash table and a lookup must be done on the second hash select and hash tables, and this process continues until a candidate index is found.

Figure 9 explains the collision resolution method via an example that uses 3 hash tables, with 3 different hash functions and 2 hash select tables. When lookup of a key is desired, the first step is to compute the hash value using the first hash function H O (key). Then, this hash value is used as index to lookup the first hash select table H O_SEL-TBL [ ], as shown in Box 901. If the corresponding select bit is "1", this means that H O_TBL[J entry has not overflowed and that a candidate index must be sought here as shown in Box 902, otherwise, the lookup must be done on the next select table H1_SEL_TBL [ ], as per Box 903. Similarly, if the corresponding select bit is "1", this means that H O_TBL[J entry has not overflowed and that a candidate index must be sought here as shown in Box 904, otherwise, the lookup must be done on the next hash table H2_TBL [ ] as per Box 905. (Since hash table H2_TBL [ ] is the last hash table, there is no need for a corresponding hash select table.)

Each entry in the hash tables can hold three index values, and additional information is provided in the entry in order to pick one of the index values, based on the key.

The hash tables, H O_TBL [ ], H1_TBL [ ] and H2_TBL [ ] each contain three index values per entry. The Key is used to differentiate the index values, and select only one of the indices. The format of a hash table entry is shown in Figure 10, where the

number of bits in the index depends on the lookup type. The procedure to pickup one of the 3 indexes stored in every hash table entry is as following.

As can be seen from figure 10, there are 3 indexes in each row, and each index has a companion 2-bit Extract. The 2-bit extract is generated using the Extract Generator. We can choose the two bits from anywhere in the key. Figure 11 shows the comparison between the Key extract and the 3 stored extracts.

In Figure 11, Hn_TBL [ ].extract θ, Hn_TBL [ ].extractl, and Hn_TBL [ ].extract2 are extracts of the keys for the different key to index relations. So, if keyO relates to .indexO, keyl relates to indexl and key2 relates to .index2, then:

- .extract θ = keyO[.extractGen[l l:6]] & key0[.extractGen[5:0]]
- .extract 1 = keyl [.extractGen[l 1:6]] & keyl[.extractGen[5:0]]
- .extract2 = key2[.extractGen[l l:6]] & key2[.extractGen[5:0]]

Hn_TBL [ ].extractGen must be selected so that the key extract values are different for valid index values. Such a value of Hn_TBL [ ].extractGen can always be selected, since all three key values for the index values are different.

Note that the symbol "&" denotes concatenation.

If a match is found, it is called the "candidate index", since a key for which no index is associated can also return a candidate index. The candidate index is therefore finally qualified, to ensure that the key for the candidate index matches the original key. The final qualification of candidate index is done by a reverse lookup from Index to Key is of a table that holds the Keys related to each Index. The reverse lookup ensures that the correct index is found.

Any variations of the above teaching are also intended to be covered by this patent application.

CLAIMS

1. A method of collision prevention or minimization for application of hash function in data transfer, data manipulation, telecommunication, verification, encryption, compression, or networking, said method comprising:

using multiple hash functions to create multiple hash tables;

hashing a key with a first hash function;

selecting an entry in a first table;

using an overflow-flag in said entry in said first table, to determine if a candidate index is sought in said entry in said first table, or if a second hash function and an entry in a second table are employed;

wherein said first table uses said first hash function; and

wherein said second table uses said second hash function.

2. A method as recited in claim 1, wherein said second hash function selects said entry in said second table, and uses an overflow-flag in said entry in said second table, to determine if a candidate index is sought in said entry in said second table, or if a third hash function and an entry in a third table are employed.

3. A method as recited in claim 2, wherein said third hash function selects said entry in said third table, and wherein a candidate index is sought in said entry in said third table.

4. A method as recited in claim 1, wherein said entry in said first table entry holds up to 3 indices, and where an extract of keys, one per said candidate index, stored with said candidate index, is used to determine which one, if any, of said up to 3 indices is said candidate index.

5. A method as recited in claim 1, wherein each stored index identifies a record, holding at least an associated key, wherein said associated key is compared with a key originally sought, to determine if a match has been found.
6. A method as recited in claim 1, wherein said method uses a quadratic probing algorithm.
7. A method as recited in claim 1, wherein said method uses a random probing algorithm.
8. A method as recited in claim 1, wherein said method further comprising the step of doing a reverse lookup from index to a first key on a table which stores said first key.
9. A method as recited in claim 1, wherein said method uses a combination of chaining, overflowing, and re-hashing algorithms.
10. A method as recited in claim 1, wherein said method is applied to network switches.
11. A method as recited in claim 1, wherein the probability of a collision is below a predetermined value.
12. A method as recited in claim 1, wherein said method uses a hash function with a small correlation value between keys and hashed values.
13. A method as recited in claim 1, wherein said method uses an XOR function.
14. A method as recited in claim 1, wherein said method uses a rotate function.
15. A method as recited in claim 1, wherein said method uses a concatenation function.
16. A method as recited in claim 1, wherein said method uses a reverse lookup table.

17. A method as recited in claim 1, wherein said method uses a variable or dynamic mapping technique.

18. A method as recited in claim 1, wherein said method uses a fixed mapping technique.

19. A method as recited in claim 1, wherein said method uses a hierarchical mapping technique.

20. A method as recited in claim 1, wherein said method uses a security mechanism to store said multiple hash functions.

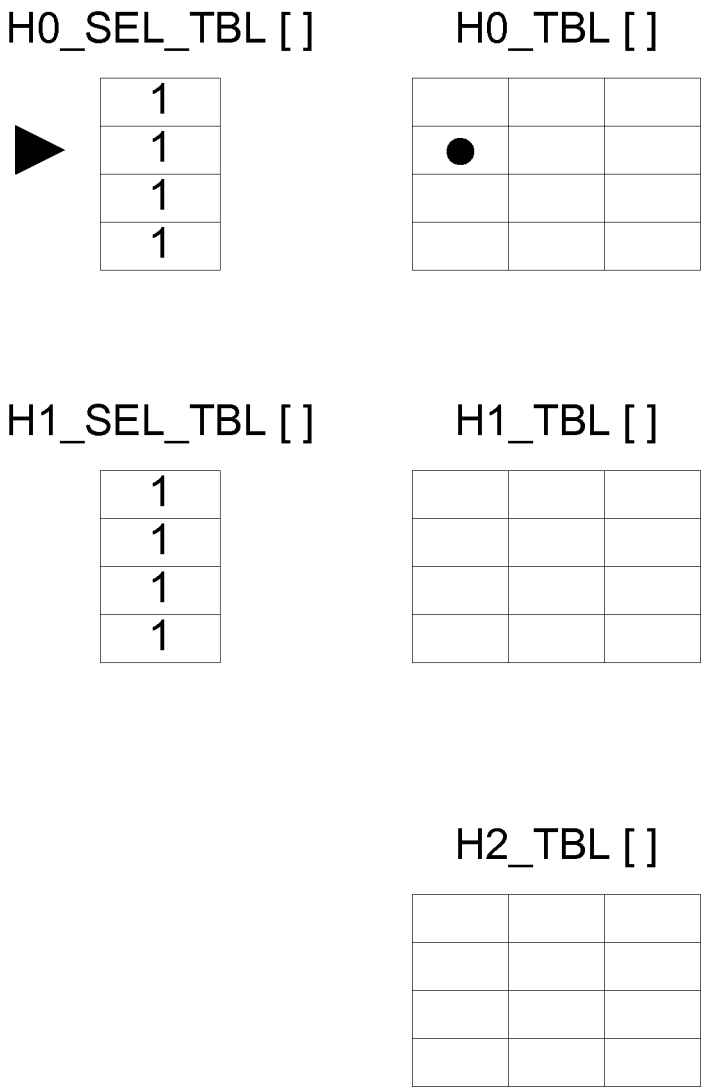

FIG 1

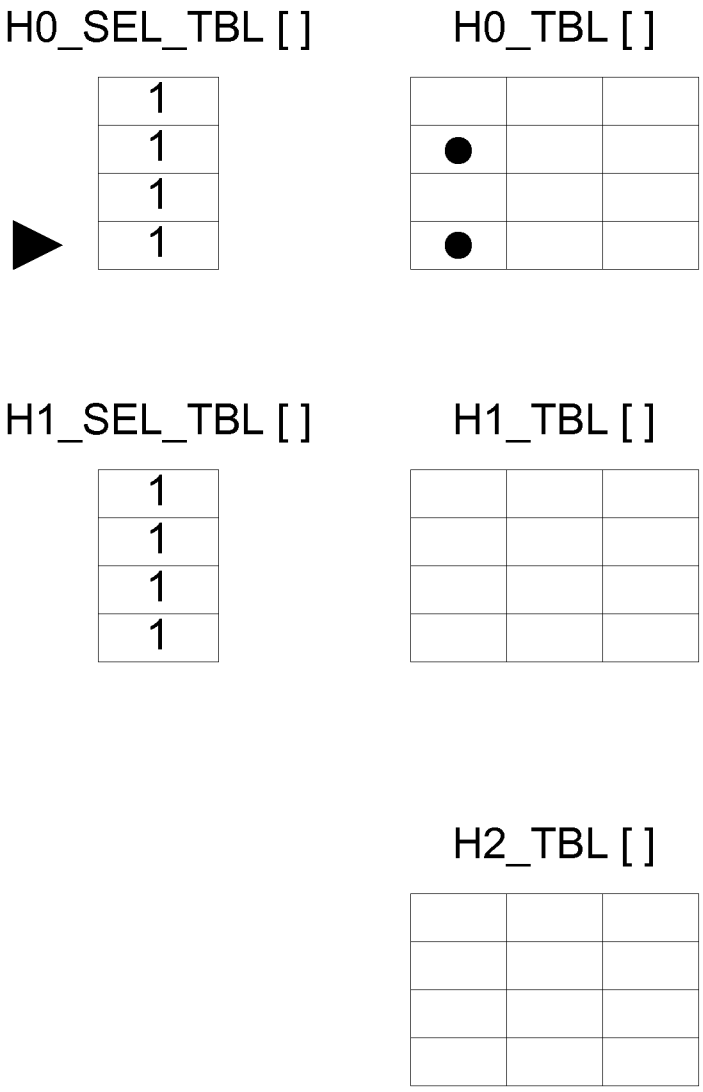

FIG 2

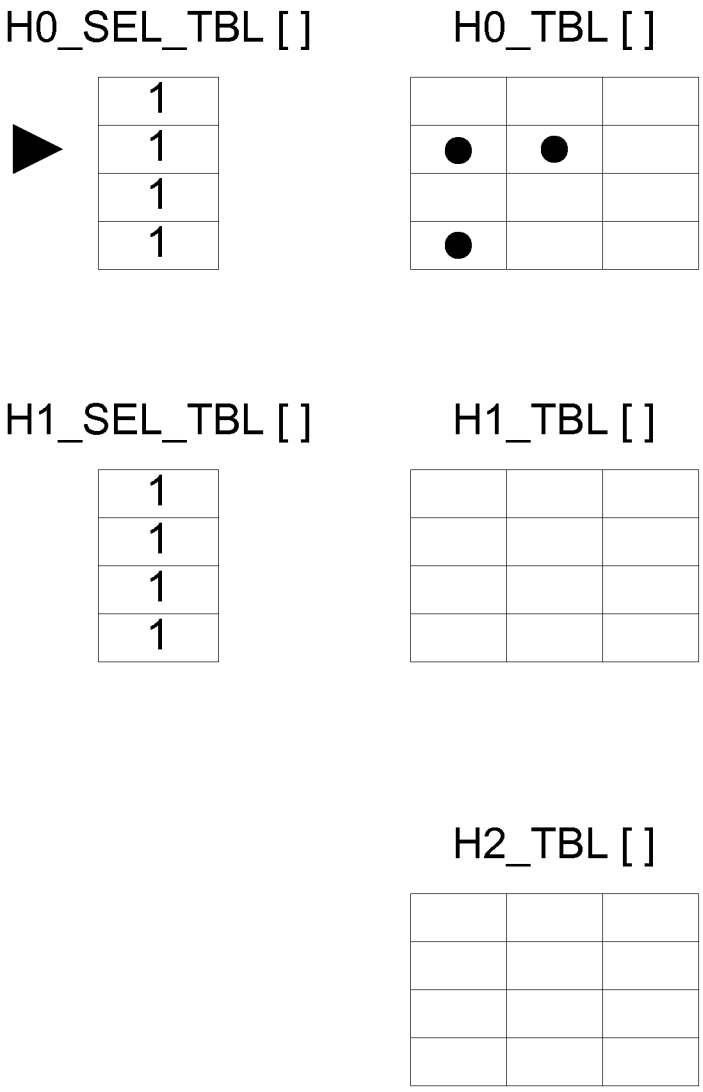

FIG 3

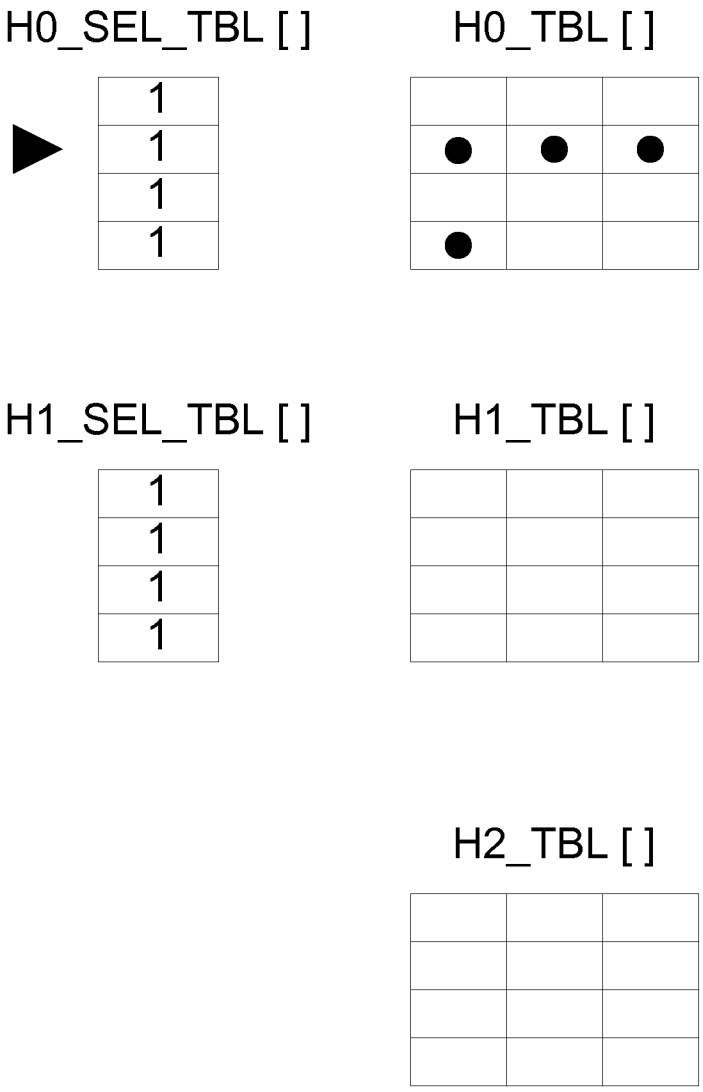

FIG 4

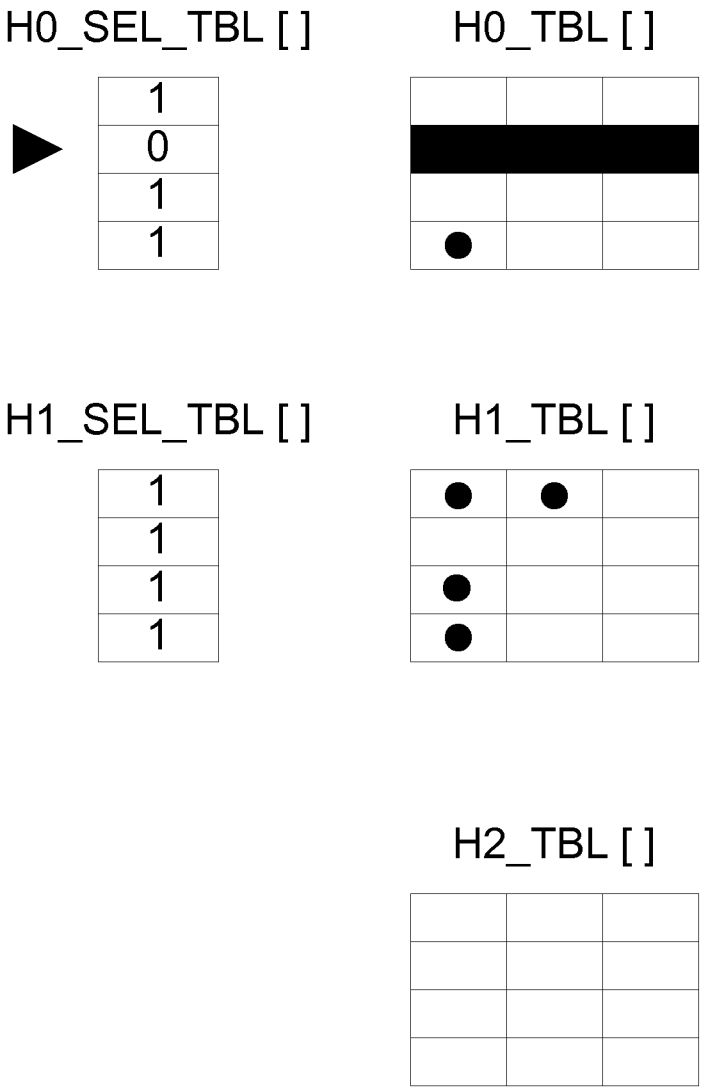

FIG 5

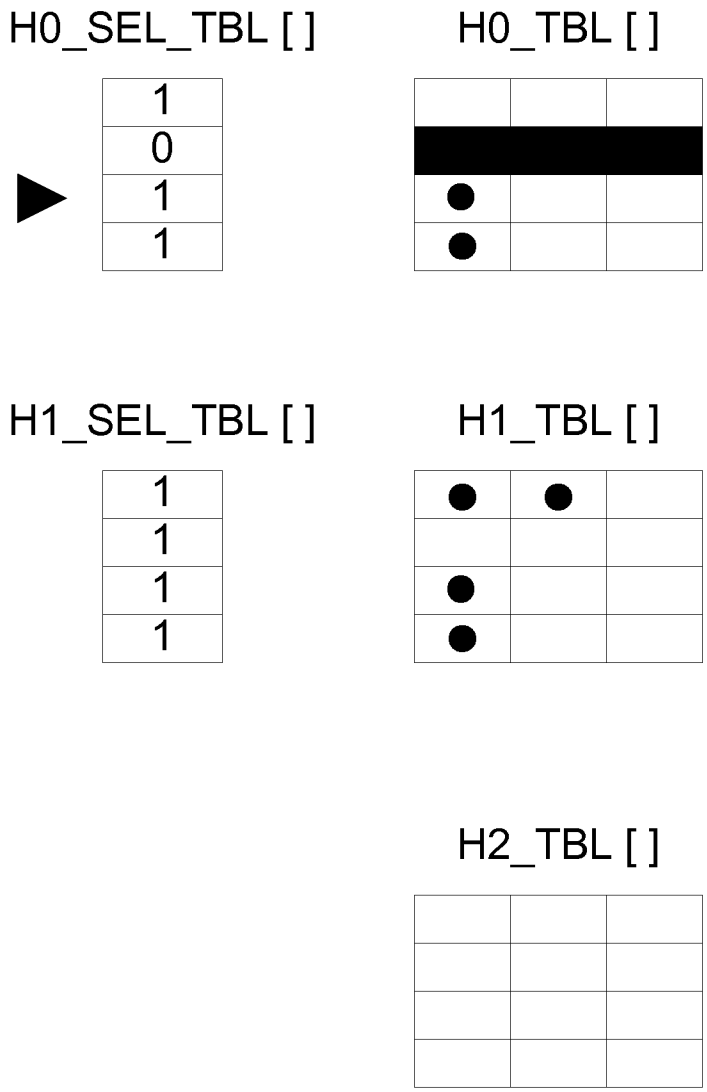

FIG 6

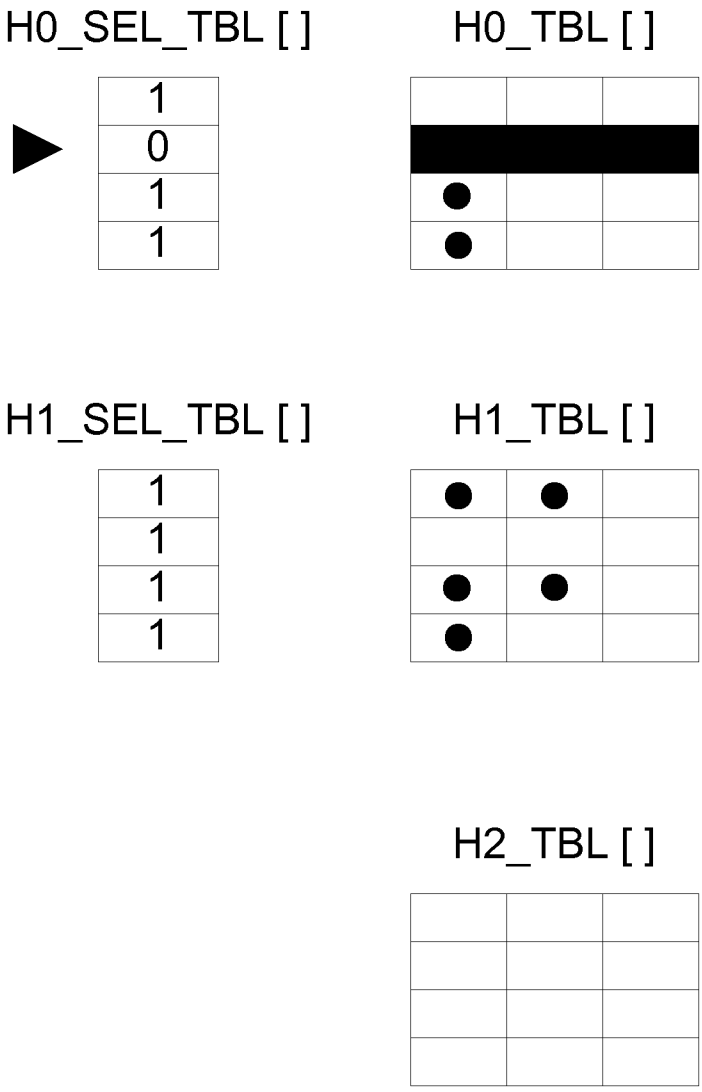

FIG 7

Key Number	Double VLAN lookup
No. of Mappings/Relations	64K
No. of Index entries	960K
Probability of collision at 100% relations	< 2.9 * 10 ^ -7
Probability of collision at 80% relations	< 5.0 * 10 ^ -11

FIG 8

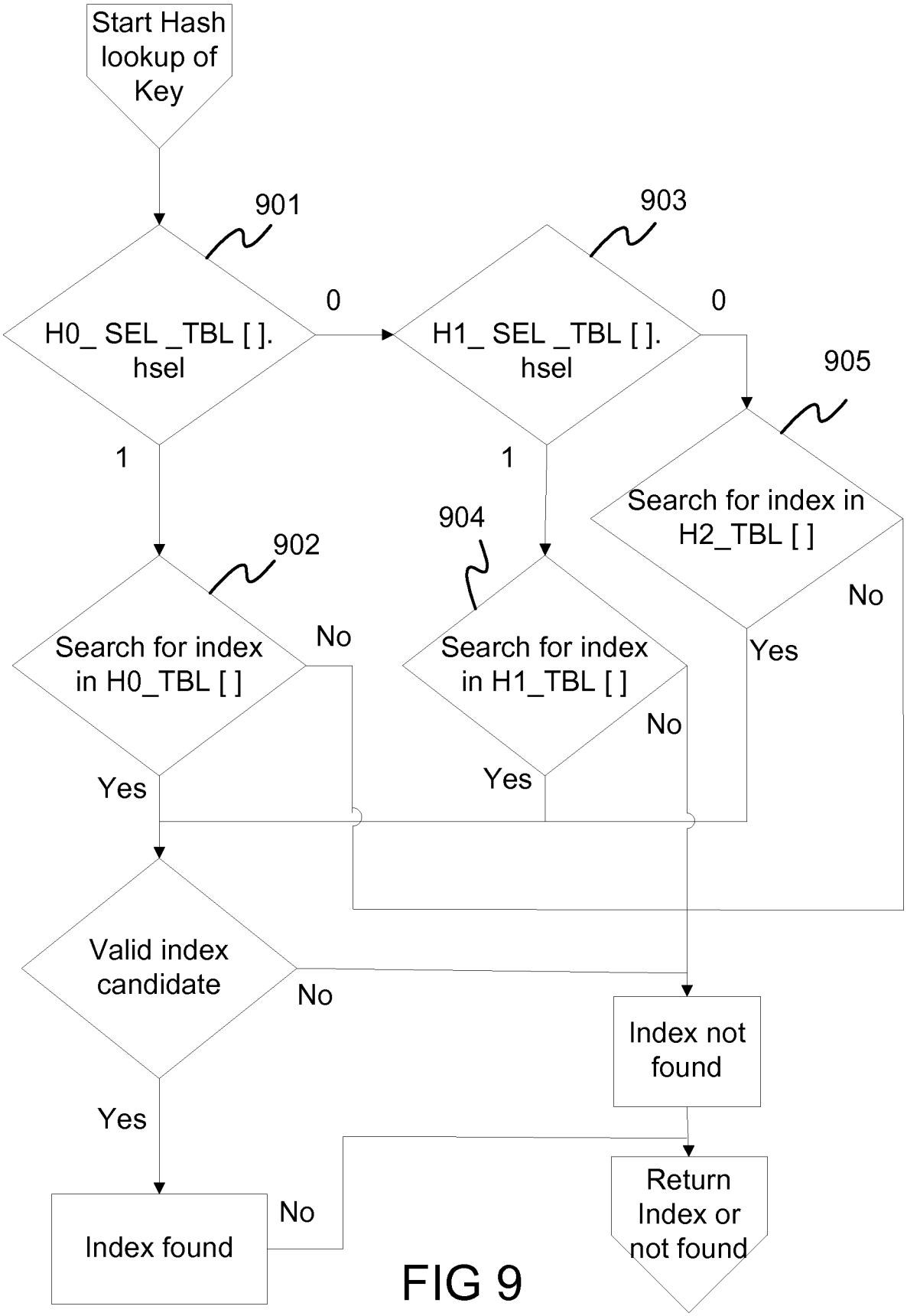
Start Hash lookup of Key
901
H0_ SEL _TBL [ ]. hsel
0
903
H1_ SEL _TBL [ ]. hsel
0
905
Search for index in H2_TBL [ ]
No
Yes
1
1
902
Search for index in H0_TBL [ ]
No
Yes
904
Search for index in H1_TBL [ ]
No
Yes
Valid index candidate
No
Yes
Index not found
Index found
No
Return Index or not found

FIG 9

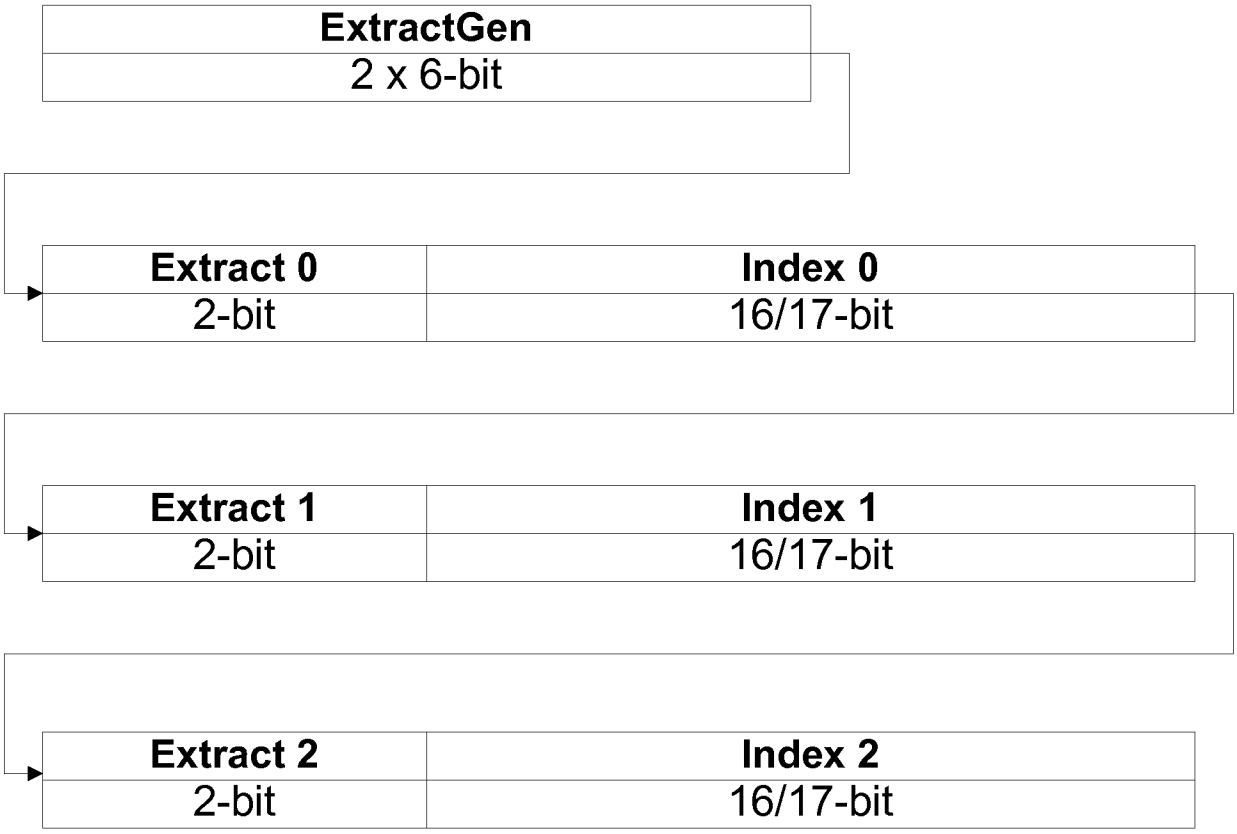
ExtractGen
2 x 6-bit
Extract 0
2-bit
Index 0
16/17-bit
Extract 1
2-bit
Index 1
16/17-bit
Extract 2
2-bit
Index 2
16/17-bit

FIG 10

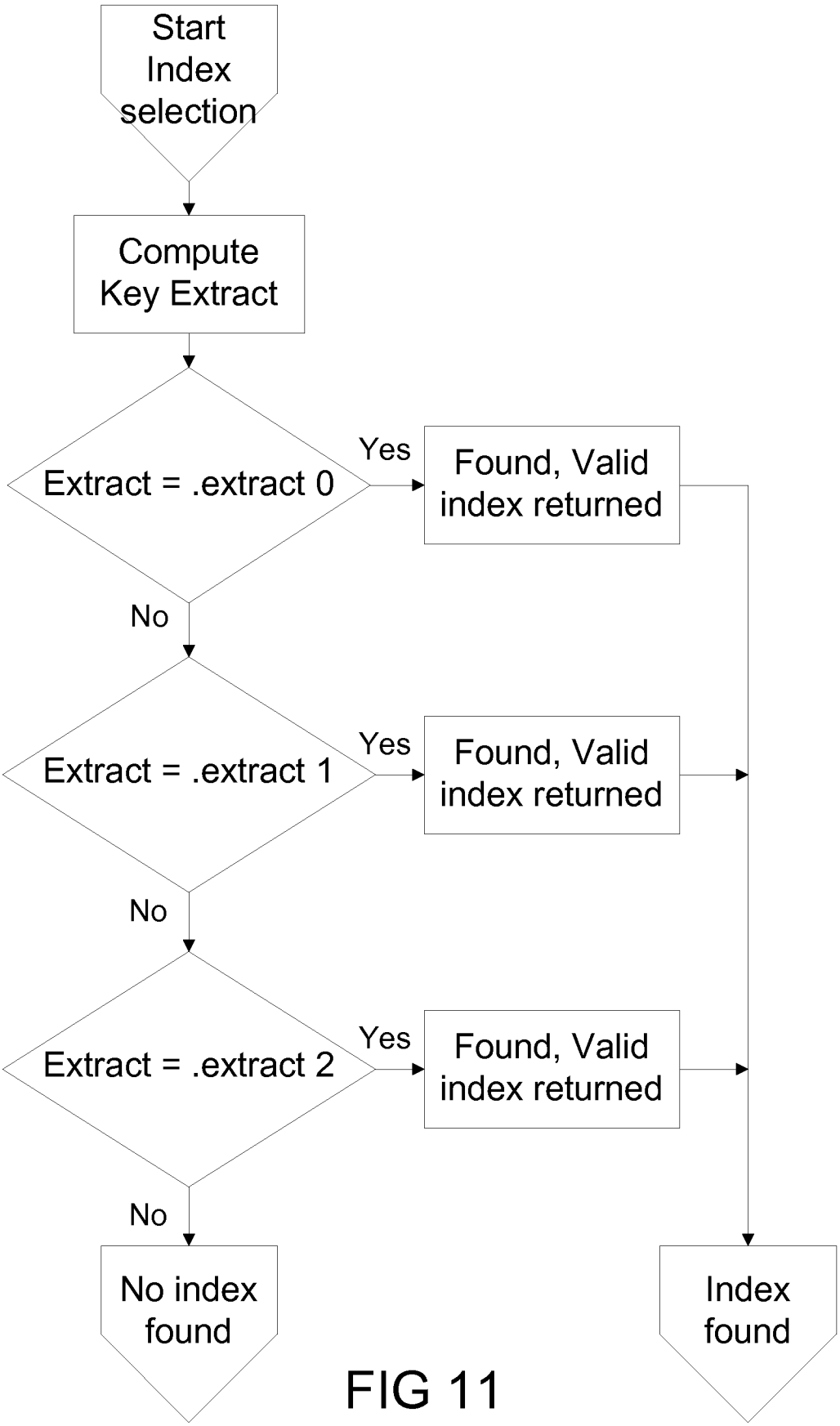
Start Index selection
Compute Key Extract
Extract = .extract 0
Yes
Found, Valid index returned
No
Extract = .extract 1
Yes
Found, Valid index returned
No
Extract = .extract 2
Yes
Found, Valid index returned
No
No index found
Index found

FIG 11