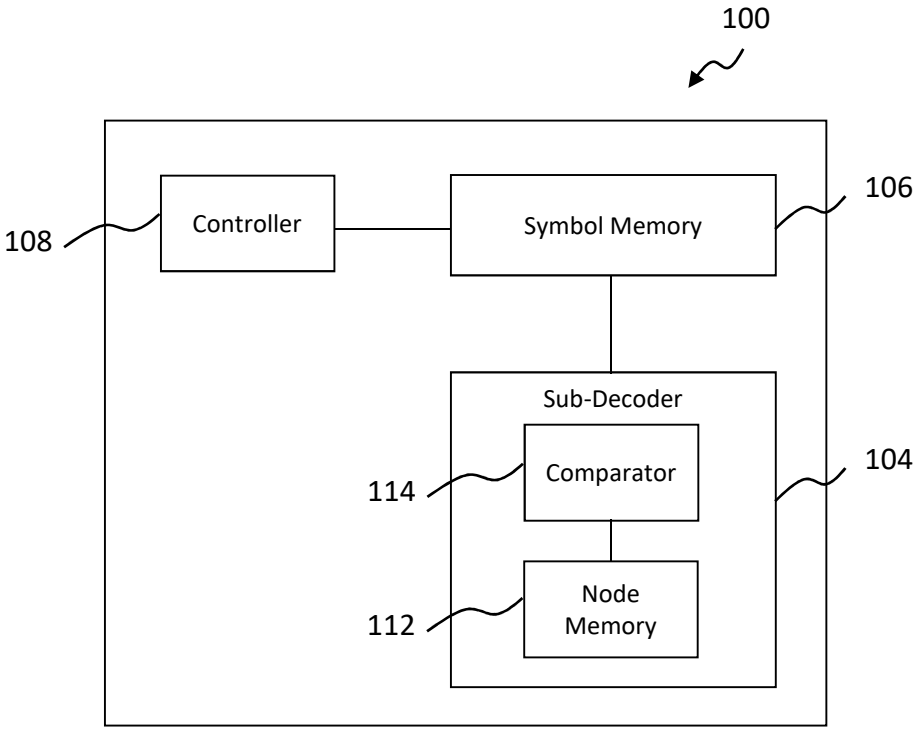

100
Controller
Symbol Memory
106
108
Sub-Decoder
Comparator
114
104
Node
Memory
112

FIG. 1

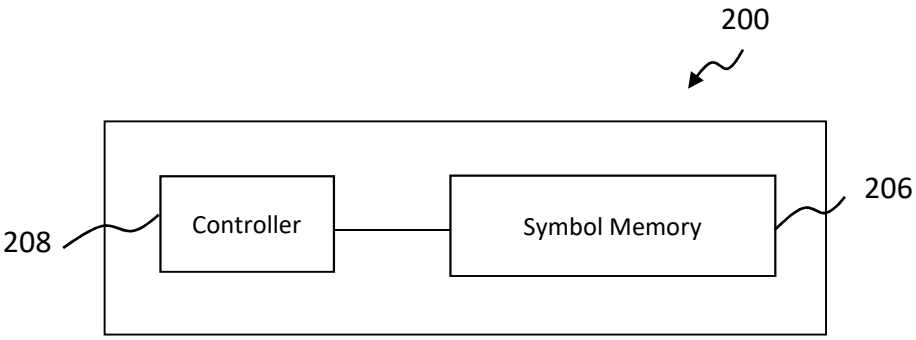

200
Controller
Symbol Memory
206
208

FIG. 2

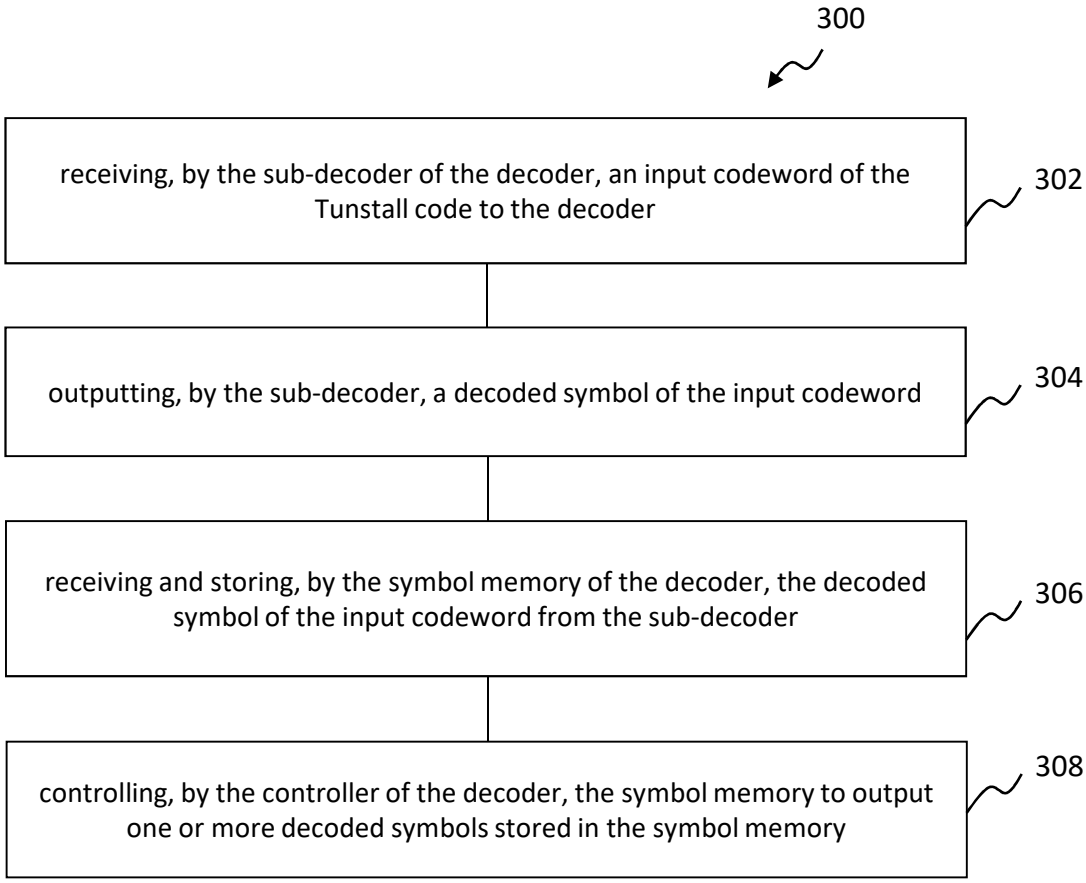

300
receiving, by the sub-decoder of the decoder, an input codeword of the Tunstall code to the decoder
302
outputting, by the sub-decoder, a decoded symbol of the input codeword
304
receiving and storing, by the symbol memory of the decoder, the decoded symbol of the input codeword from the sub-decoder
306
controlling, by the controller of the decoder, the symbol memory to output one or more decoded symbols stored in the symbol memory
308

FIG. 3

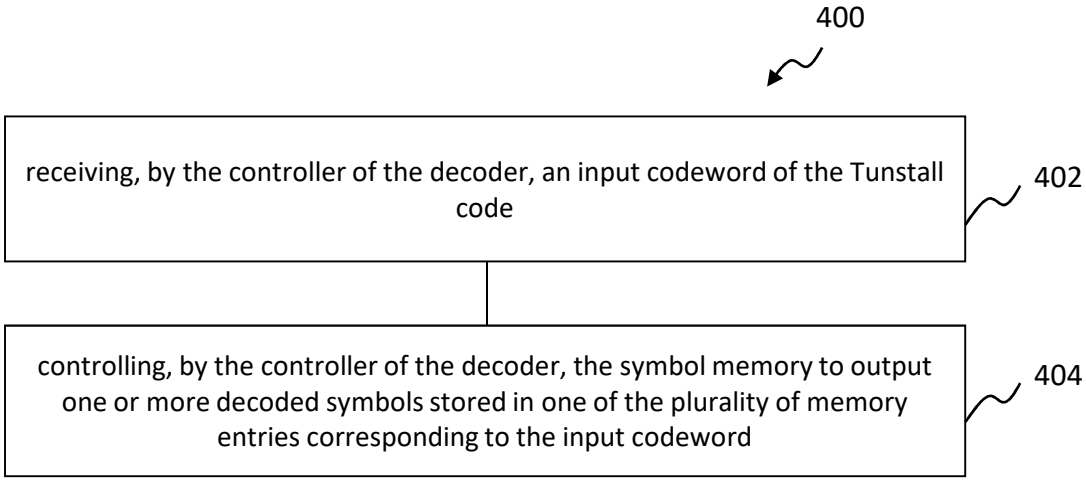

400
receiving, by the controller of the decoder, an input codeword of the Tunstall code
402
controlling, by the controller of the decoder, the symbol memory to output one or more decoded symbols stored in one of the plurality of memory entries corresponding to the input codeword
404

FIG. 4

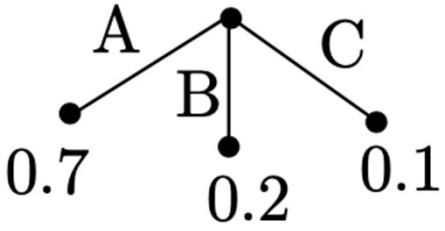


FIG. 5A

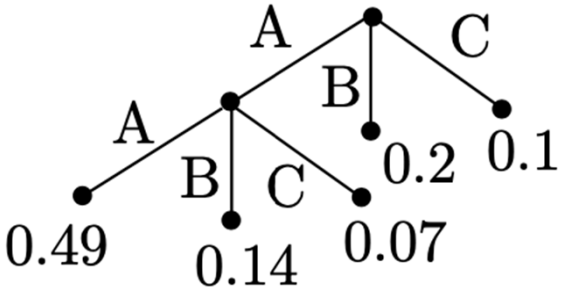


FIG. 5B

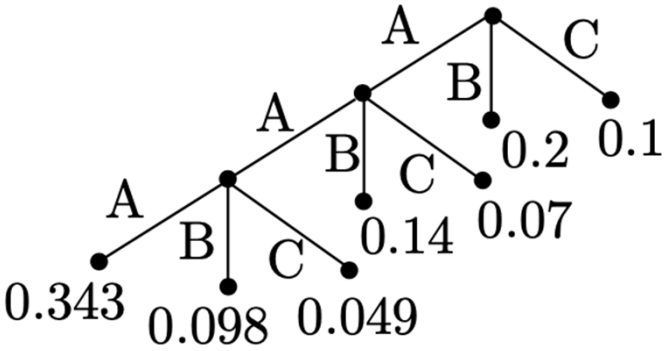


FIG. 5C

TABLE I
THE CODEBOOK OF THE EXAMPLE TUNSTALL TREE

Codeword	**000**	**001**	**010**	**011**	**100**	**101**	**110**
Symbols	*AAA*	*AAB*	*AAC*	*AB*	*AC*	*B*	*C*

FIG. 6

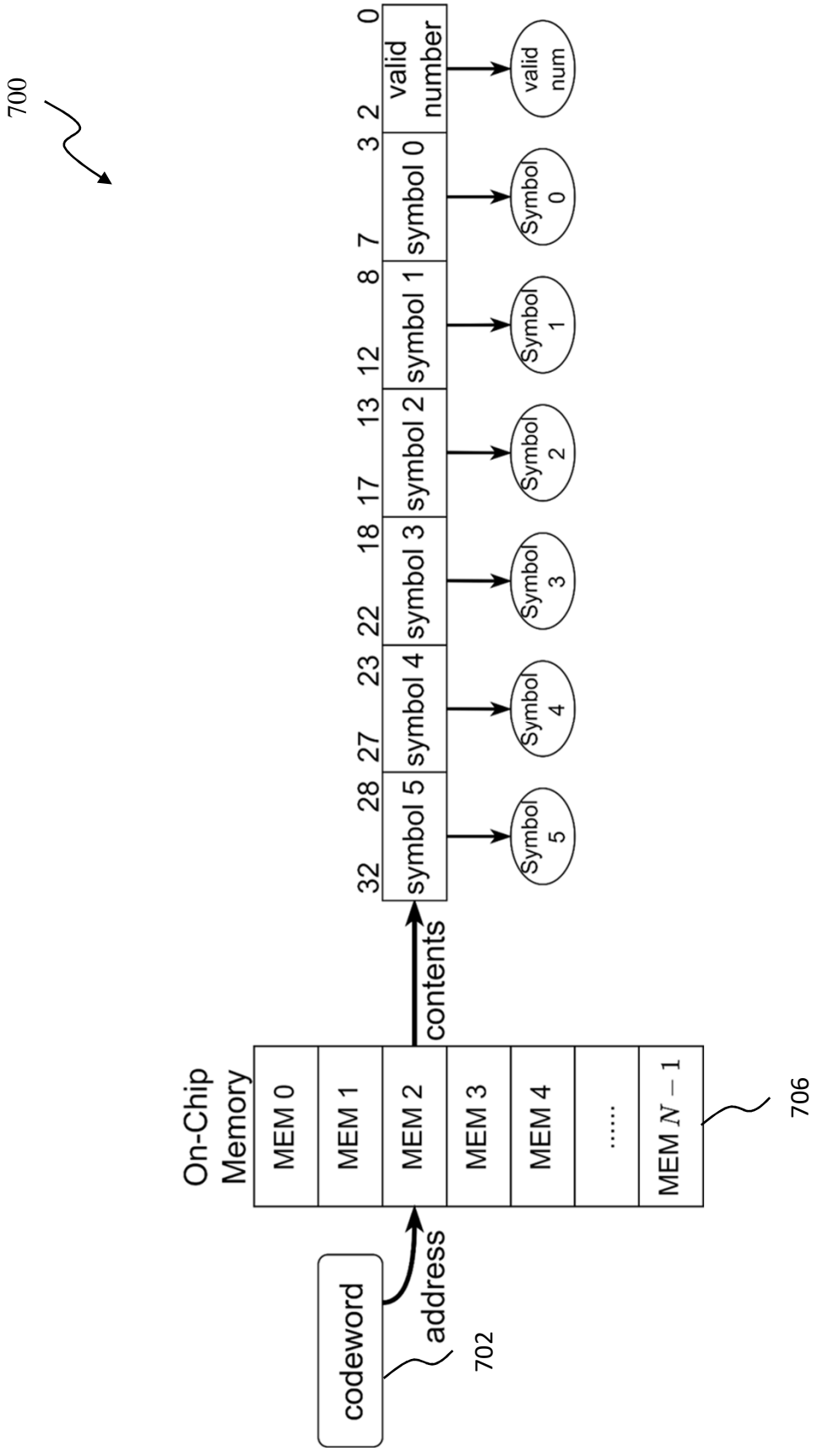

700
codeword
address
702
On-Chip Memory
MEM 0
MEM 1
MEM 2
MEM 3
MEM 4
.....
MEM $N-1$
706
contents
32
28
27
23
22
18
17
13
12
8
7
3
2
0
symbol 5
symbol 4
symbol 3
symbol 2
symbol 1
symbol 0
valid number
Symbol 5
Symbol 4
Symbol 3
Symbol 2
Symbol 1
Symbol 0
valid num

FIG. 7

Algorithm 1: The procedure of dictionary-based Tunstall decoding

input : A codeword under decoding c,
the codebook b
output: A sequence of decoded $symbols$

```
1 Function DicDecoding(b, c) is
2 |   dictionary value ← b[c];
3 |   foreach symbol in dictionary value do
4 |   |   Output the decoded symbol;
5 |   end
6 end
```

FIG. 8

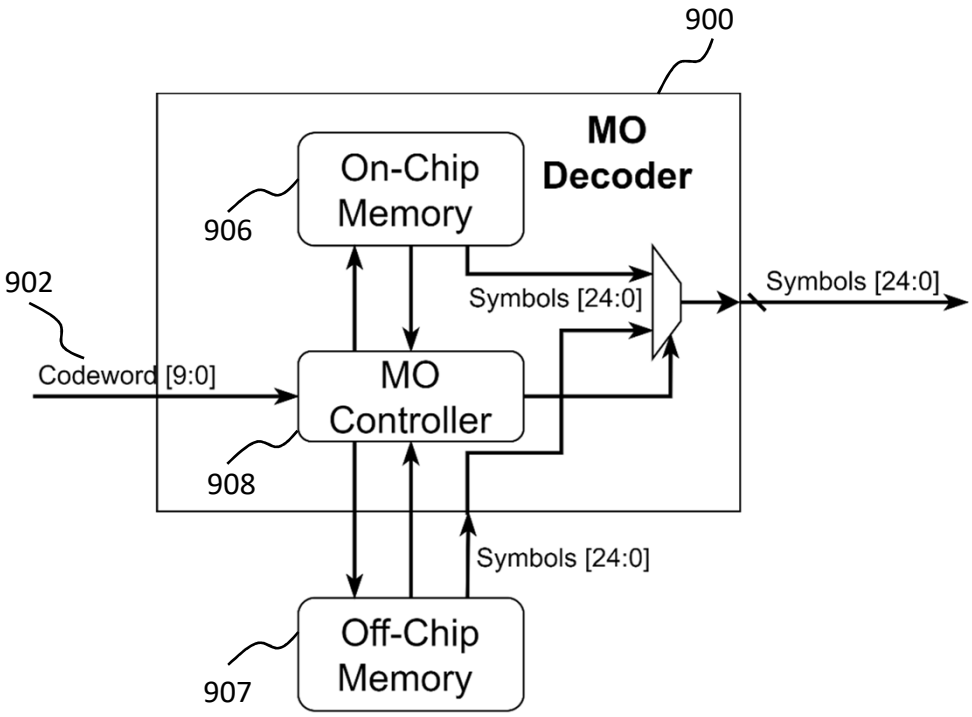


FIG. 9

32 28	27 23	22 18	17 13	12 8	7 3	2 0
symbol 5	symbol 4	symbol 3	symbol 2	symbol 1	symbol 0	valid number
					symbol 0	$3'b001$
				symbol 1	symbol 0	$3'b010$
			symbol 2	symbol 1	symbol 0	$3'b011$
		symbol 3	symbol 2	symbol 1	symbol 0	$3'b100$
symbol 5	symbol 4	symbol 3	symbol 2	symbol 1	symbol 0	$3'b101$

FIG. 10

Algorithm 2: Update the codewords

```
1  Function Updating(oldCodebook) is
2      D ← emptydictionary;
3      UpdatedCodebook ← emptydictionary;
4      // count the frequency
5      for codeIdx ← 0 to numberofcodewords do
6          if codeIdx ∈ keys(D) then
7              D[codeIdx] ← D[codeIdx] +1;
8          else
9              D[codeIdx] ← 1;
10         end
11     end
12     // sort the counter array
13     Sort(values(D));
14     // remapping the codewords
15     for i ← 0 to size of keys(D) do
16         UpdatedCodebook[i] ← oldCodebook[keys(D)[i]];
17     end
18 end
```

FIG. 11A

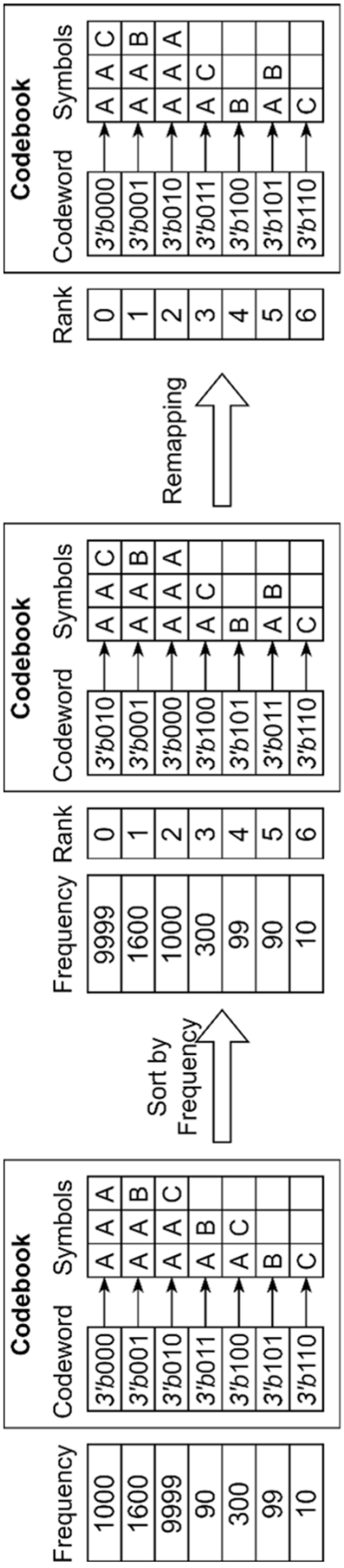

Frequency
1000
1600
9999
90
300
99
10
Codebook
Codeword
Symbols
3'b000 A A A
3'b001 A A B
3'b010 A A C
3'b011 A B
3'b100 A C
3'b101 B
3'b110 C
Sort by Frequency
Frequency
9999
1600
1000
300
99
90
10
Rank
0
1
2
3
4
5
6
Codebook
Codeword
Symbols
3'b010 A A C
3'b001 A A B
3'b000 A A A
3'b100 A C
3'b101 B
3'b011 A B
3'b110 C
Remapping
Rank
0
1
2
3
4
5
6
Codebook
Codeword
Symbols
3'b000 A A C
3'b001 A A B
3'b010 A A A
3'b011 A C
3'b100 B
3'b101 A B
3'b110 C

FIG. 11B

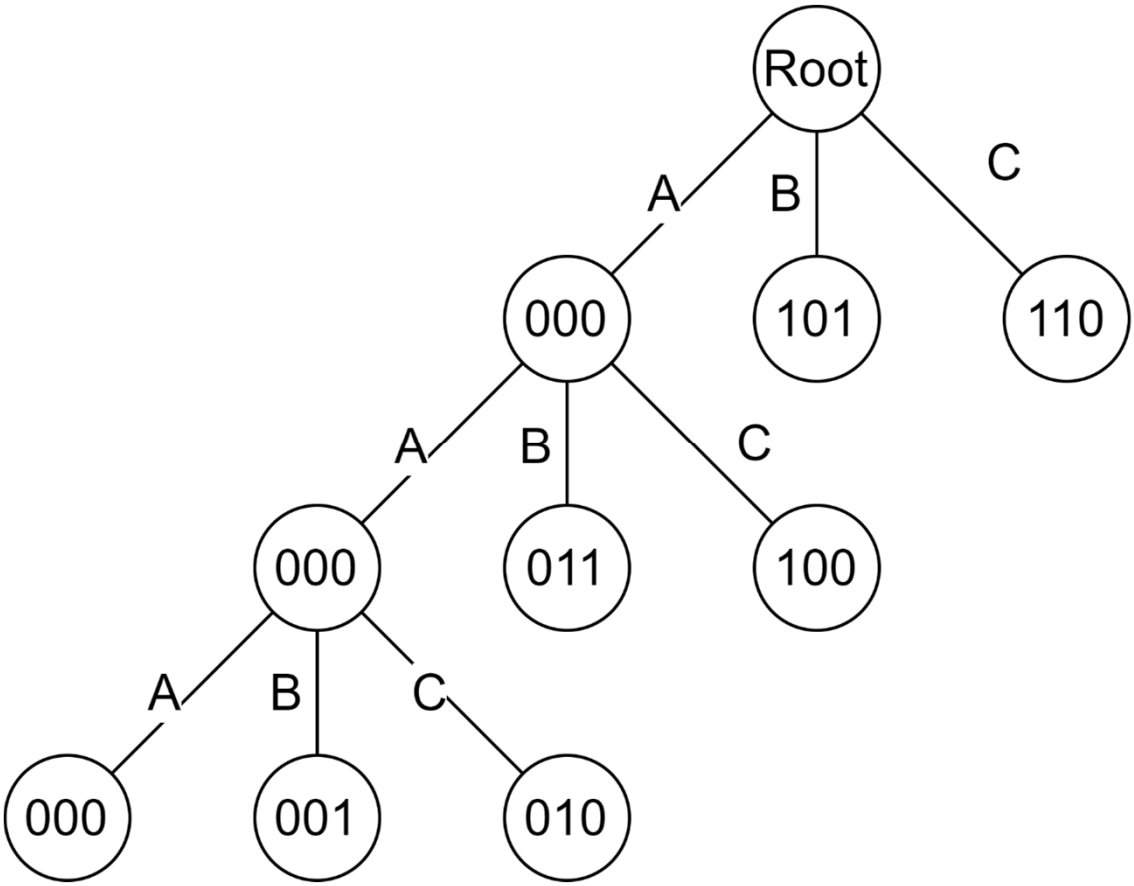


FIG. 12

Algorithm 3: Find the rank of the codeword

input : codeword under decoding c,
codewords of the nodes nc
output: the rank of this codeword r

Function `FindRank(`c, nc`)` **is**
  $compResult \leftarrow \emptyset$;
  for $node\ idx \leftarrow 0$ **to** $number\ of\ nodes$ **do**
    if $c \geq nc[node\ idx]$ **then**
      $compResult[node\ idx] \leftarrow 1$;
    else
      $compResult[node\ idx] \leftarrow 0$;
    end
  end
  $onehot \leftarrow compResult \oplus [0, compResult[: -1]]$;
  `// convert the onehot to a integer`
  $r \leftarrow$ `OHToInt(`$onehot$`)`;
end

FIG. 13

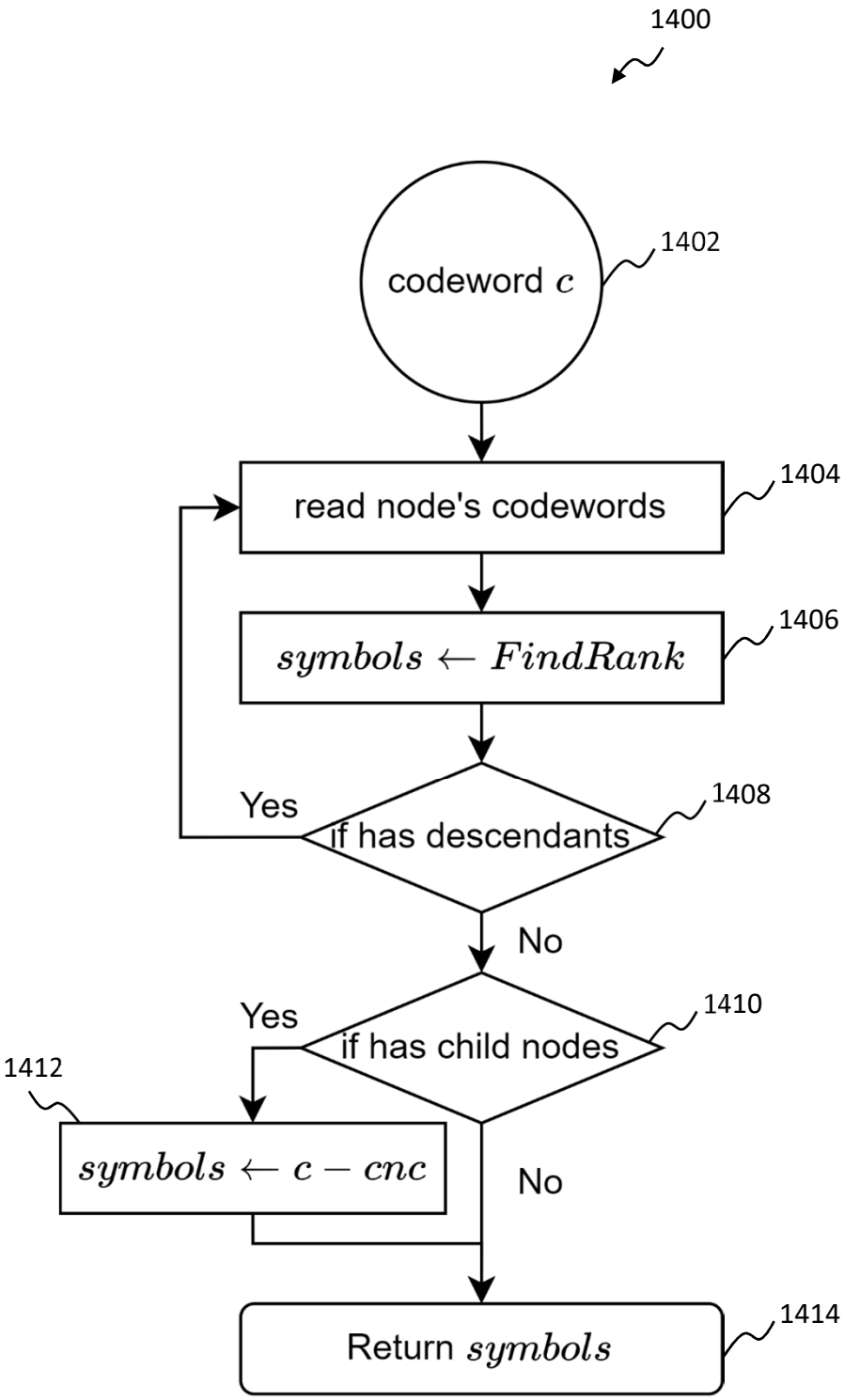

1400
codeword c
1402
read node's codewords
1404
$symbols \leftarrow FindRank$
1406
Yes
if has descendants
1408
No
Yes
if has child nodes
1410
1412
$symbols \leftarrow c - cnc$
No
Return $symbols$
1414

FIG. 14

Algorithm 4: decoding the symbols along the tunstall tree. nc: a array of the codewords of the nodes; ns: a array of Boolean values of whether the nodes are inner nodes; na: a array of node addresses; ng: a array of Boolean values of whether the nodes have descendants

input : codeword under decoding c,
depth of current nodes in the Tunstall tree d,
address of the memory a

output: A sequence of uncompressed $symbol$s

```
1  Function Decoding(c, d, a) is
2      symbols ← ∅;
3      nc, ns, na, ng ← n[d][a];
4      curSyb ← FindRank(c, nc);
5      if current node has descendants then
6          a ← na[curSyb];
7          Decoding the symbol using the function
            Decoding(c, d + 1, a);
8      else
9          if current node has child nodes then
10             // cnc: current node's
                   codeword
11             cnc ← nc[curSyb];
12             symbols ← c − cnc
13         else
14             current codeword has been decompressed
15         end
16     end
17 end
```

FIG. 15

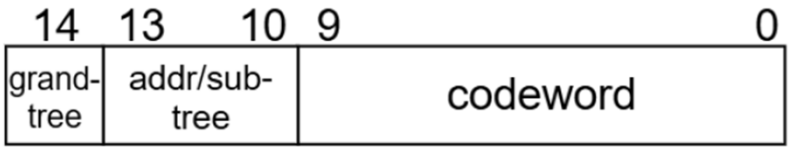


FIG. 16

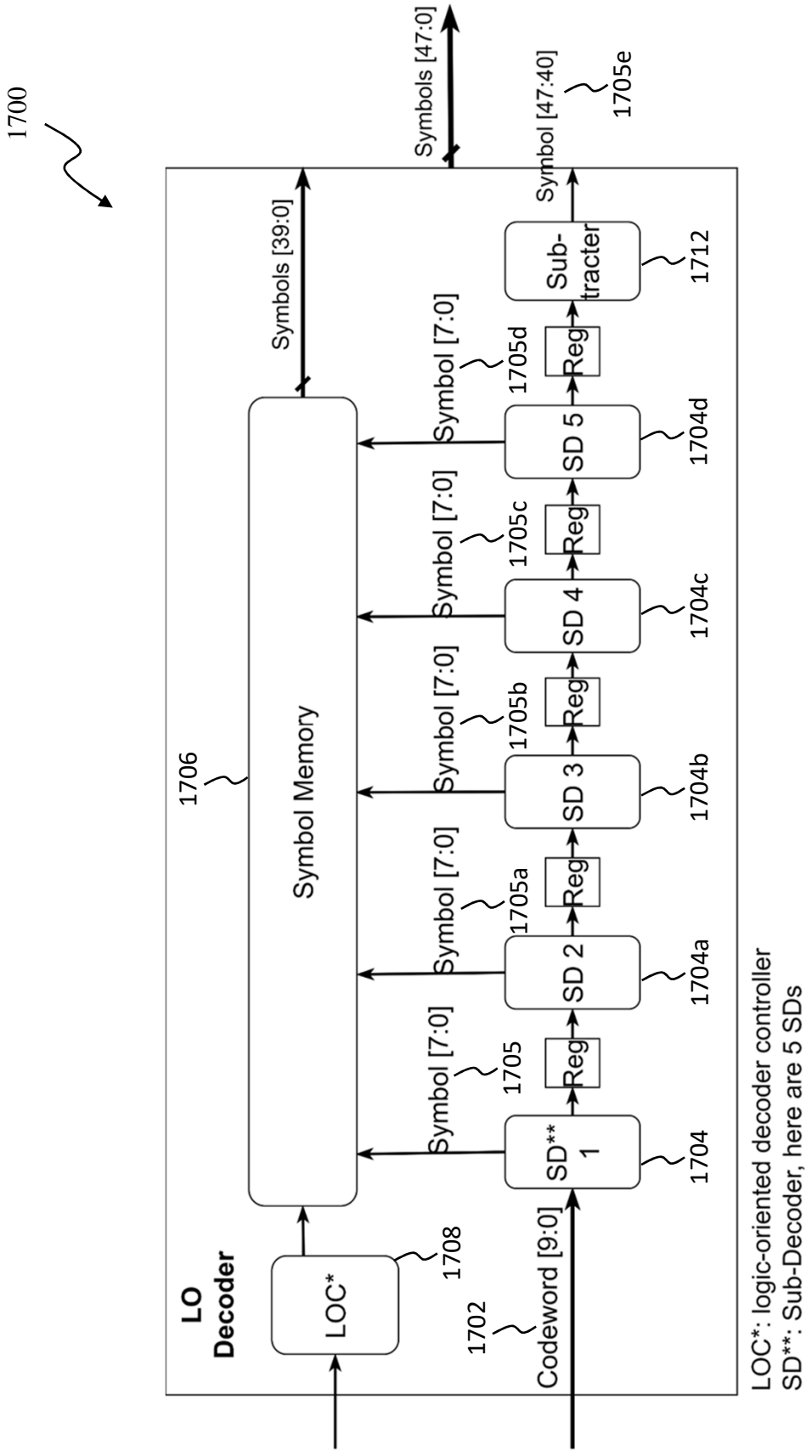

1700
LO Decoder
LOC*
1708
1702
Codeword [9:0]
Symbols [39:0]
1706
Symbol Memory
Symbols [47:0]
Symbol [47:40]
1705e
SD** 1
1704
Symbol [7:0]
1705
Reg
SD 2
1704a
1705a
SD 3
1704b
1705b
SD 4
1704c
1705c
SD 5
1704d
1705d
Sub-tracter
1712
LOC*: logic-oriented decoder controller
SD**: Sub-Decoder, here are 5 SDs

FIG. 17

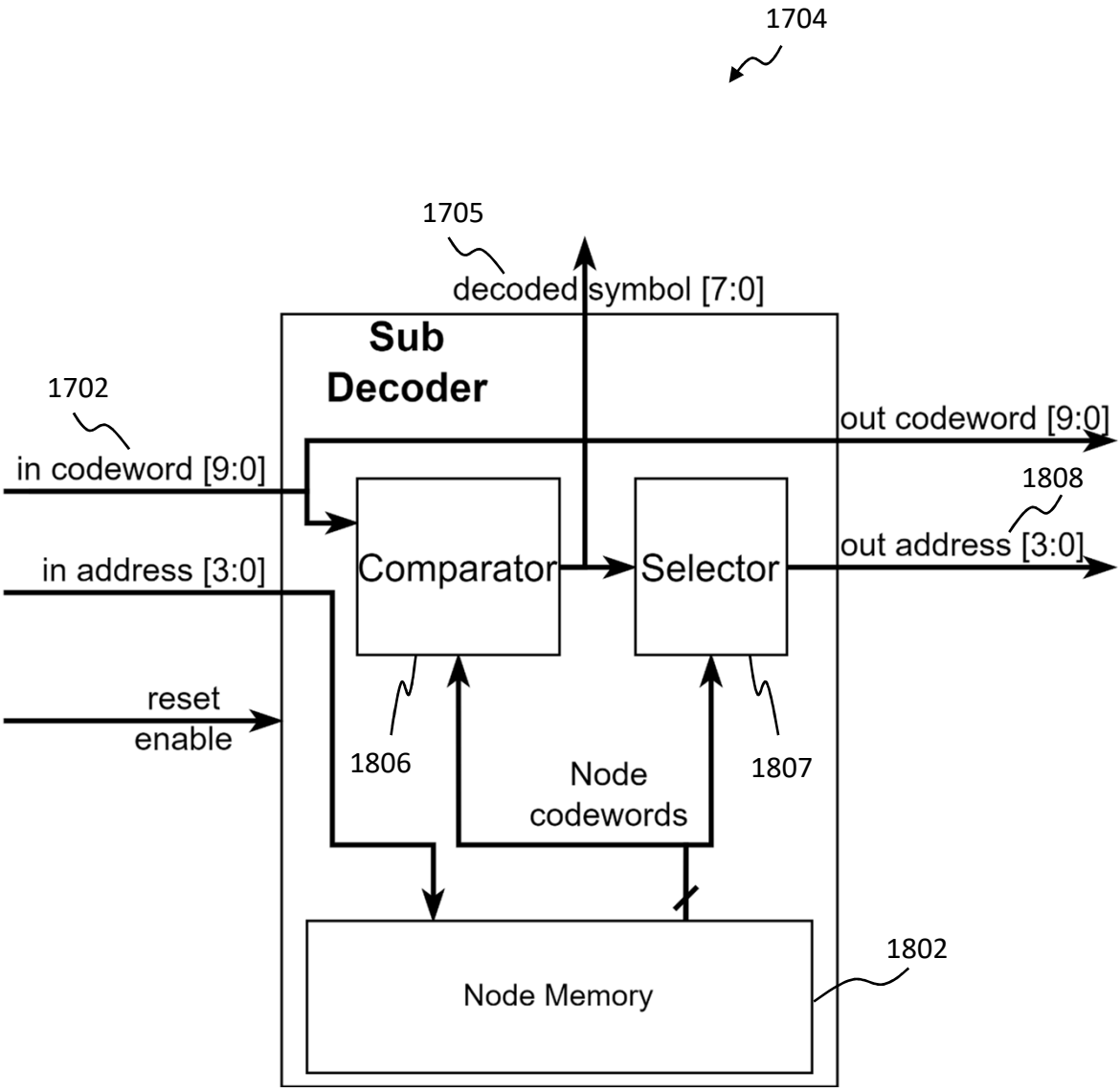

1704
1705
decoded symbol [7:0]
Sub
Decoder
1702
in codeword [9:0]
out codeword [9:0]
1808
in address [3:0]
Comparator
Selector
out address [3:0]
reset
enable
1806
Node
codewords
1807
Node Memory
1802

FIG. 18

TABLE II
THE RESOURCES UTILIZATION AND DECODING CYCLES OF MO DECODERS WITH DIFFERENT MEMORY ENTRIES AND MEMORY TYPE

Resources and Decoding Cycle		**Memory-Oriented Decoder Entires and Memory Type**			
		MO 1024 RAM	***MO 1024 SRAM***	***MO 64 RAM***	***MO 64 SRAM***
Resources*	***LUT***	1,601	192	267	197
	FF	1	13	13	20
	LUT RAM	1,184	0	76	0
	BRAM	0	2	0	1
	BUFG	1	1	1	1
Cycle	***ResNet-50***	4,565,527	10,764,540	9,131,054	12,212,460
	MobileNet-v2	743,007	1,486,014	1,848,581	2,035,657

*The RTLs are generated by the Chisel codes, and synthesis by Vivado 2020.1 without optimization.

FIG. 19

TABLE III
THE RESOURCES UTILIZATION AND DECODING CYCLES OF LO DECODERS WITH DIFFERENT NUMBER OF SUB-DECODERS AND MEMORY BANKS

Resources and Decoding Cycle		**Logic-Oriented Decoder Entires and Memory Type**			
		LO 8 SD* 27 banks	***LO 1 SD 27 banks***	***LO 8 SD 10 banks***	***LO 1 SD 10 banks***
Resources**	***LUT***	6,322	2,140	4,942	1,771
	FF	3,201	33	544	33
	LUT RAM	2,585	1,442	2,797	1,236
	BUFG	2	1	2	1
Cycle	***ResNet-50***	8,621,733	26,099,734	8,711,905	26,184,537
	MobileNet-v2	1,081,004	3,621,028	1,117,003	3,655,032

* Sub-Decoder
**The RTLs are generated by the Chisel codes, and synthesis by Vivado 2020.1 without optimization.

FIG. 20

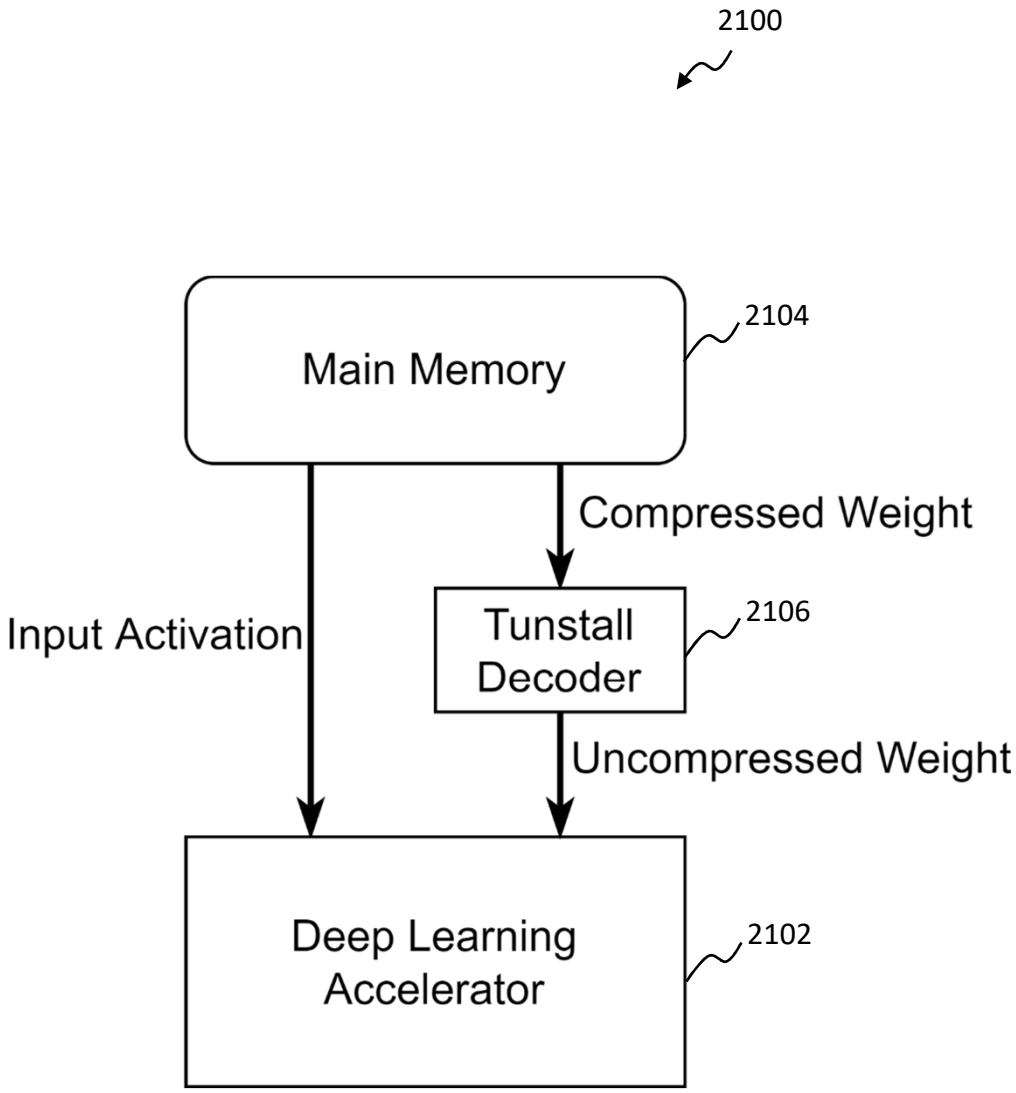

2100
Main Memory
2104
Compressed Weight
Input Activation
Tunstall Decoder
2106
Uncompressed Weight
Deep Learning Accelerator
2102

FIG. 21

TABLE IV
THE CAPACITY OF THE MAIN MEMORY AND ESTIMATED MEMORY ACCESS POWER IN DIFFERENT METHODS

CNN Architectures	Methods	Memory Capacity Requirements (Bytes, including the codewbook)			Estimated Power Consumption (pJ)
		maximum	*average*	*mimimum*	
ResNet-50	*full precision (32-bit)*	11,180,732.0	2,108,203.7	20,188.0	1,707,645,000.0
	Quantized 8-bit	2,795,183.0	527,050.9	5,047.0	426,911,250.0
	Quantized 4-bit	1,397,591.5	263,525.5	2,523.5	213,455,625.0
	Quantized 4-bit with Tunstall LO decoder	565,622.7	113,640.9	4,733.8	46,693,720.0
	Quantized 4-bit with Tunstall MO decoder	571,026.7	119,114.5	10,942.6	16,266,349.2
	Quantized 4-bit with Huffman coding	521,362.3	108,522.5	3,520.9	17,412,582.0
MobileNet-v2	*full precision (32-bit)*	5,756,796.0	270,116.0	1,124.0	214,742,220.0
	Quantized 8-bit	1,439,199.0	67,529.0	281.0	53,640,555.0
	Quantized 4-bit	719,599.5	33,764.5	140.5	26,842,770.0
	Quantized 4-bit with Tunstall LO decoder	260,755.8	17,936.5	490.0	7,146,345.2
	Quantized 4-bit with Tunstall MO decoder	265,611.3	23,877.6	6,540.0	3,002,169.6
	Quantized 4-bit with Huffman coding	263,962.3	17,549.4	268.4	2,576,045.4

FIG. 22

TABLE V
MEMORY ACCESS CYCLES IN DIFFERENT METHODS

CNN Architectures	Methods	Weights Read Clock Cycle		
		maximum	*average*	*minimum*
ResNet-50	*off-chip DRAM*	8,385,549	1,581,153	15,141
	on-chip MRAM	2,795,183	527,051	5,047
	Quantized 4-bit with Tunstall LO decoder	1,084,854	159,833	4,146
	Quantized 4-bit with Tunstall MO decoder	424,494	85,568	4,442
	Quantized 4-bit with Huffman coding	4,170,894	868,124	27,753
MobileNet-v2	*off-chip DRAM*	4,317,597	202,587	843
	on-chip MRAM	1,439,199	67,529	281
	Quantized 4-bit with Tunstall LO decoder	226,658	14,537	218
	Quantized 4-bit with Tunstall MO decoder	208,397	15,036	1,216
	Quantized 4-bit with Huffman coding	2,111,695	140,333	1,847

FIG. 23

TABLE VI
TUNSTALL DECODER AND HUFFMAN DECODER PERFORMANCE OF RESNET-50

Hardware Decoder	Resources Utilization (# LUT)	Decoding Cycles	Memory Capacity (Byte)	Memory Access Power (pJ)
7-stage 24 memory banks LO Tunstall decoder	6,322	8,653,482	6,136,611.08	46,693,720.0
1024 entries MO Tunstall decoder	1,601	4,565,527	6,432,181.83	16,266,349.2
256 entries Huffman decoder	5,021	28,460,750	5,860,216.50	17,412,582.0

FIG. 24

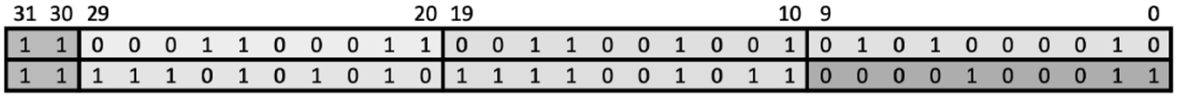


FIG. 25A

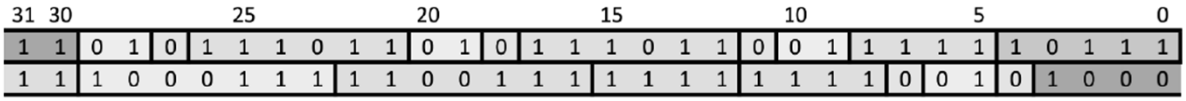


FIG. 25B

TABLE VII
TUNSTALL AND HUFFMAN SOFTWARE DECODING TIME (MICROSECOND)

CNN Architectures	Coding Methods	Software Decoding Time (microseconds)		
		maximum	*average*	*minimum*
ResNet-50	***Tunstall Coding***	121,999	23,073.59	541
	Huffman Coding	178,217	35,786.87	732
MobileNet-v2	***Tunstall Coding***	71,046	3,606.79	302
	Huffman Coding	105,512	5,578.81	75

FIG. 26