

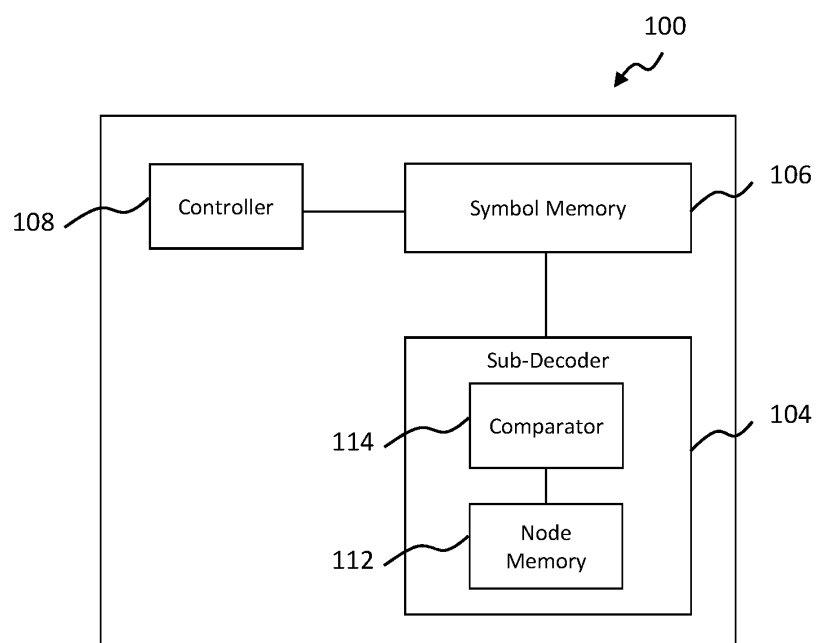


FIG. 1

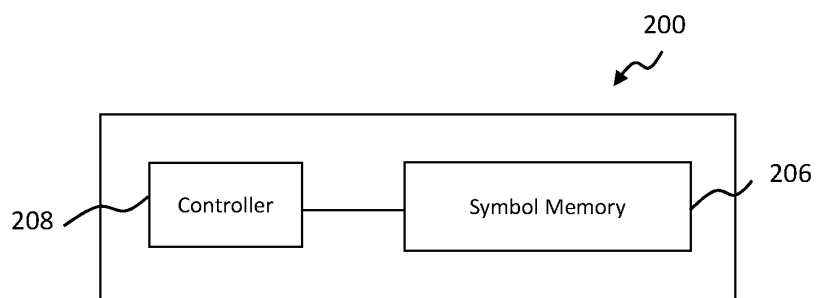


FIG. 2

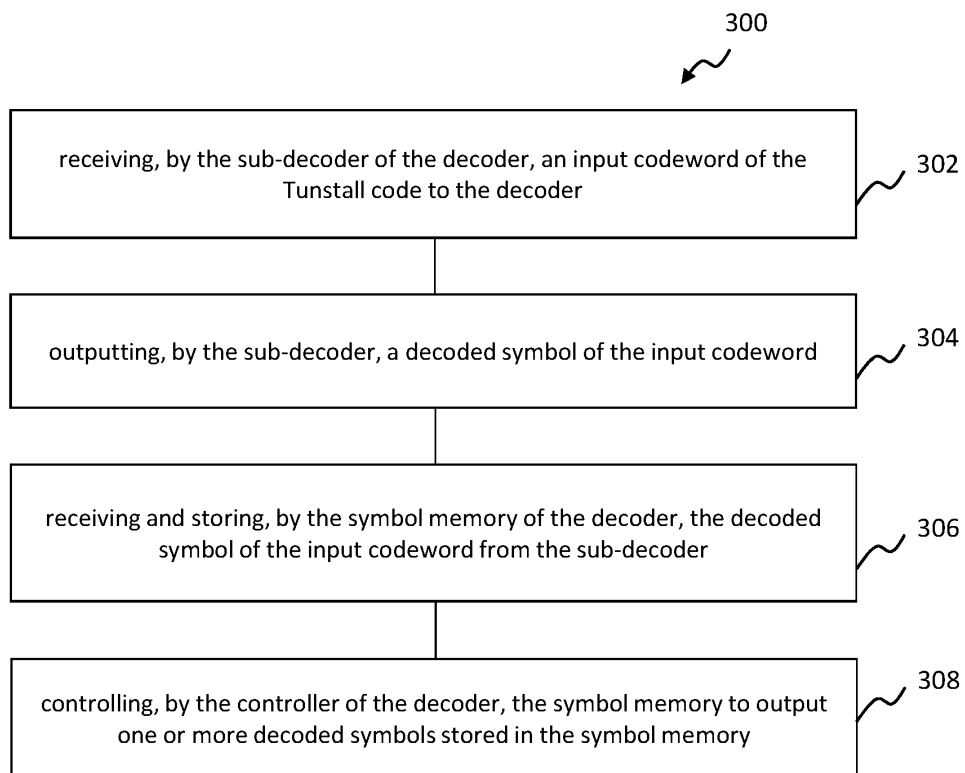


FIG. 3

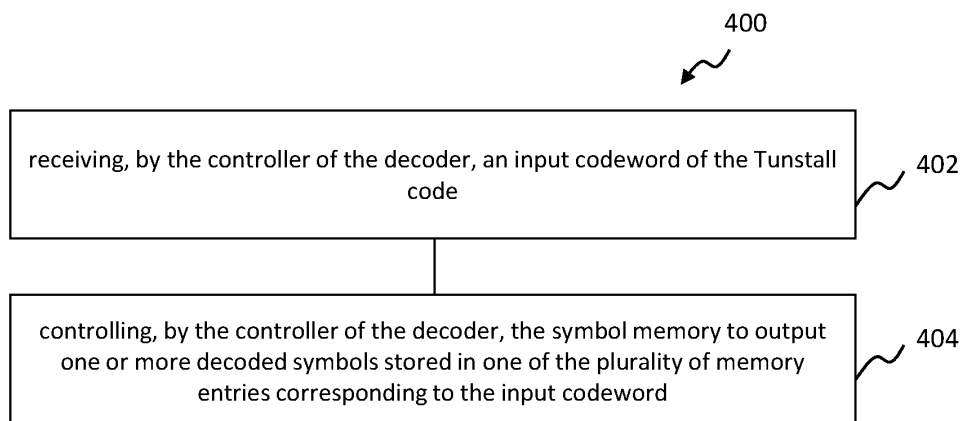


FIG. 4

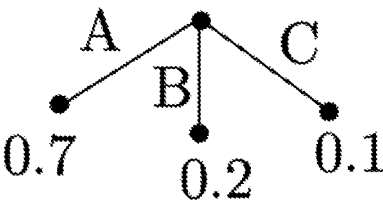


FIG. 5A

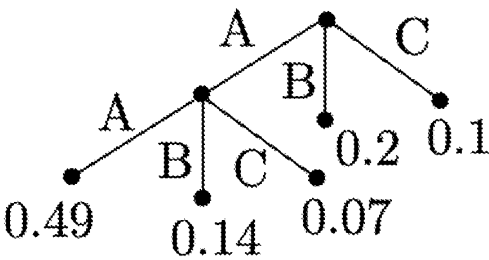


FIG. 5B

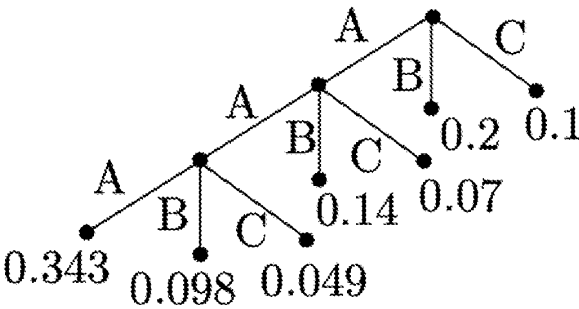


FIG. 5C

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

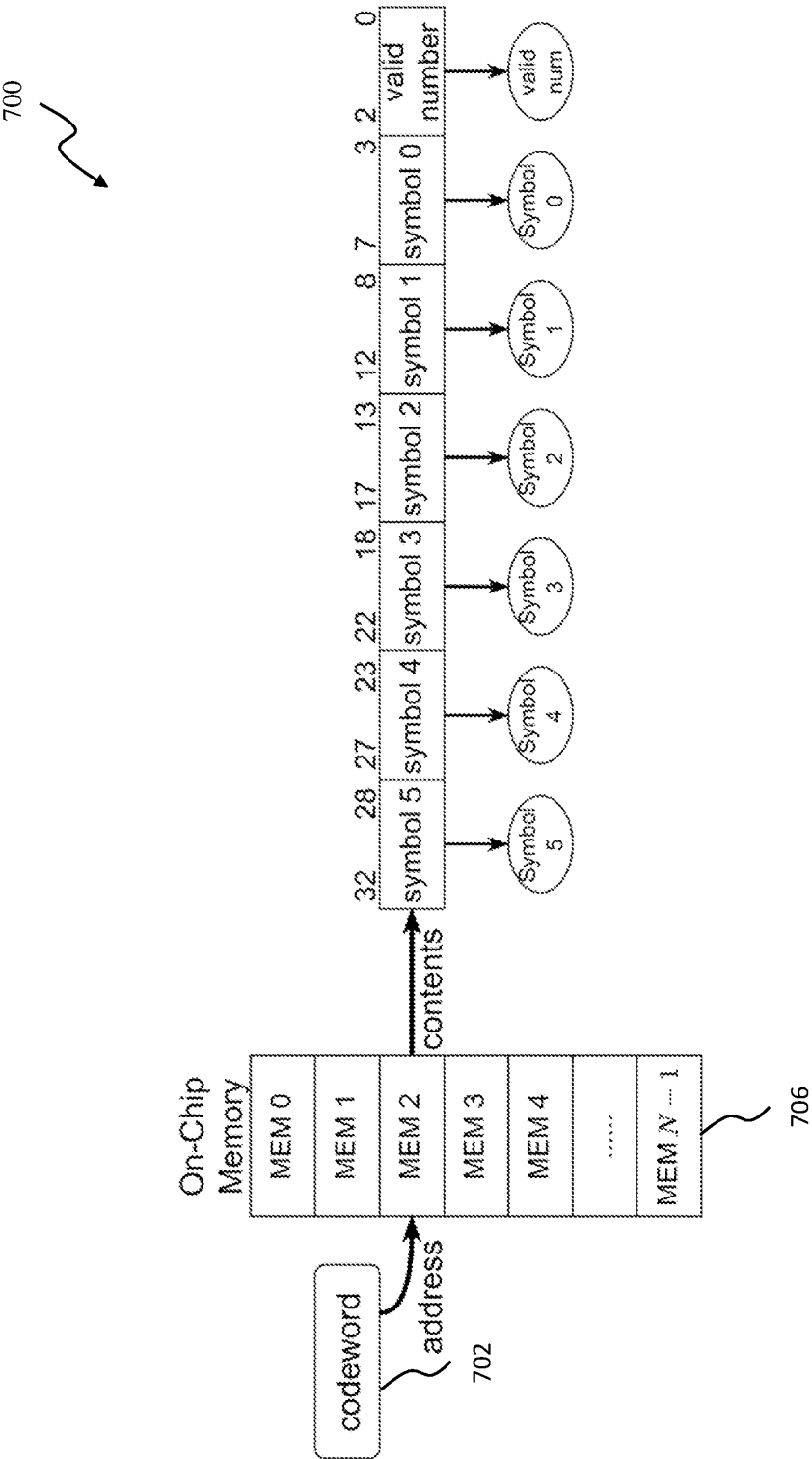


FIG. 7

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   |  $dictionary\ value \leftarrow b[c];$   
3   | foreach  $symbol$  in  $dictionary\ value$  do  
4   |   | Output the decoded symbol;  
5   | end  
6 end
```

FIG. 8

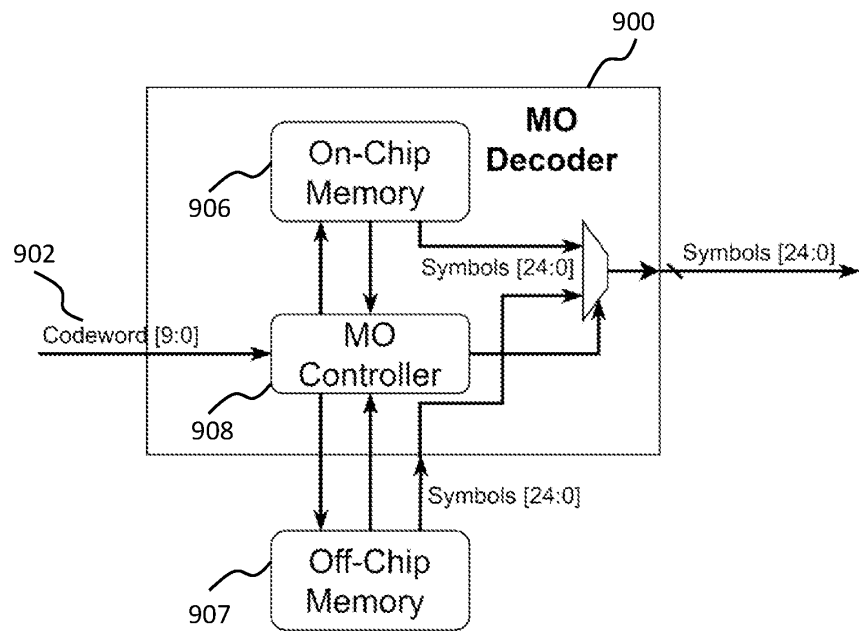


FIG. 9

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  $\leftarrow$  emptydictionary;
3   UpdatedCodebook  $\leftarrow$  emptydictionary;
4   // count the frequency
5   for codeIdx  $\leftarrow$  0 to numberOfcodewords do
6     if codeIdx  $\in$  keys(D) then
7       | D[codeIdx]  $\leftarrow$  D[codeIdx] + 1;
8     else
9       | D[codeIdx]  $\leftarrow$  1;
10    end
11  end
12  // sort the counter array
13  Sort(values(D));
14  // remapping the codewords
15  for i  $\leftarrow$  0 to size of keys(D) do
16    | UpdatedCodebook[i]  $\leftarrow$  oldCodebook[keys(D)[i]];
17  end
18 end

```

FIG. 11A

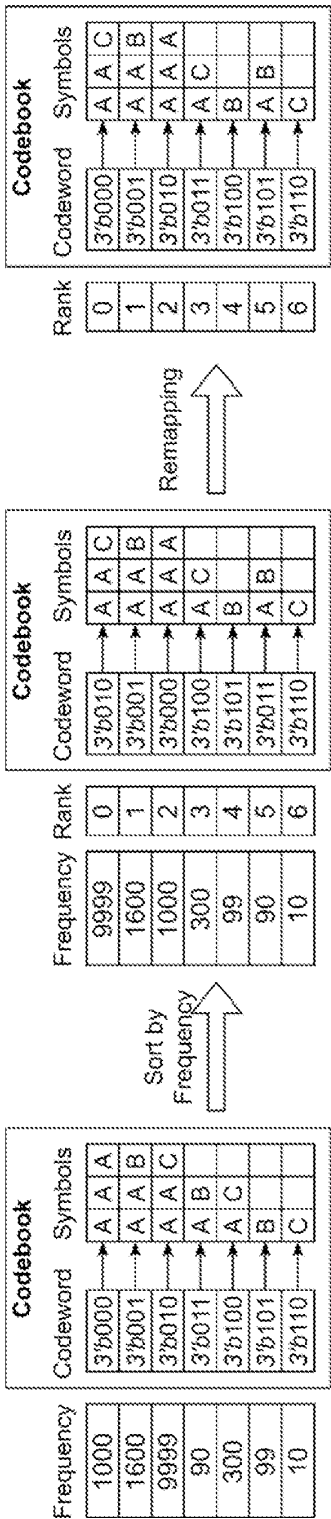


FIG. 11B

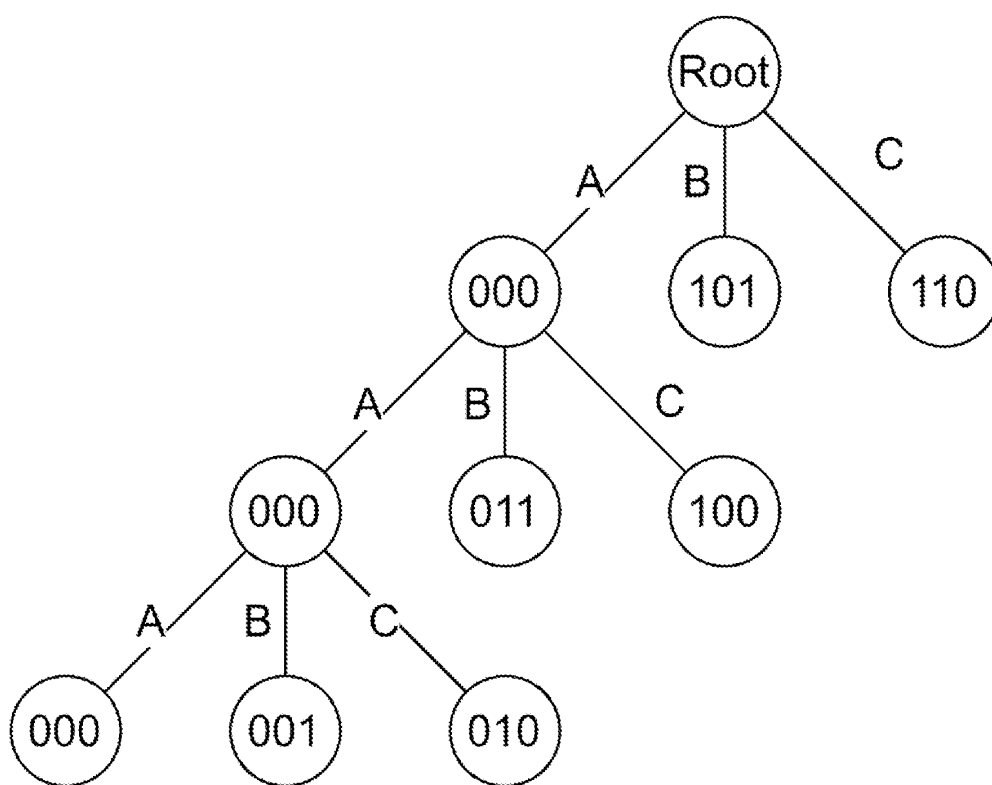


FIG. 12

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

```

1 Function FindRank ( $c, nc$ ) is
2    $compResult \leftarrow \emptyset$ ;
3   for  $node\ idx \leftarrow 0$  to  $number\ of\ nodes$  do
4     if  $c \geq nc[node\ idx]$  then
5        $compResult[node\ idx] \leftarrow 1$ ;
6     else
7        $compResult[node\ idx] \leftarrow 0$ ;
8     end
9   end
10   $onehot \leftarrow compResult \oplus [0, compResult[: -1]]$ ;
11   $//\ convert\ the\ onehot\ to\ a\ integer$ 
12   $r \leftarrow OHToInt(onehot)$ ;
13 end

```

FIG. 13

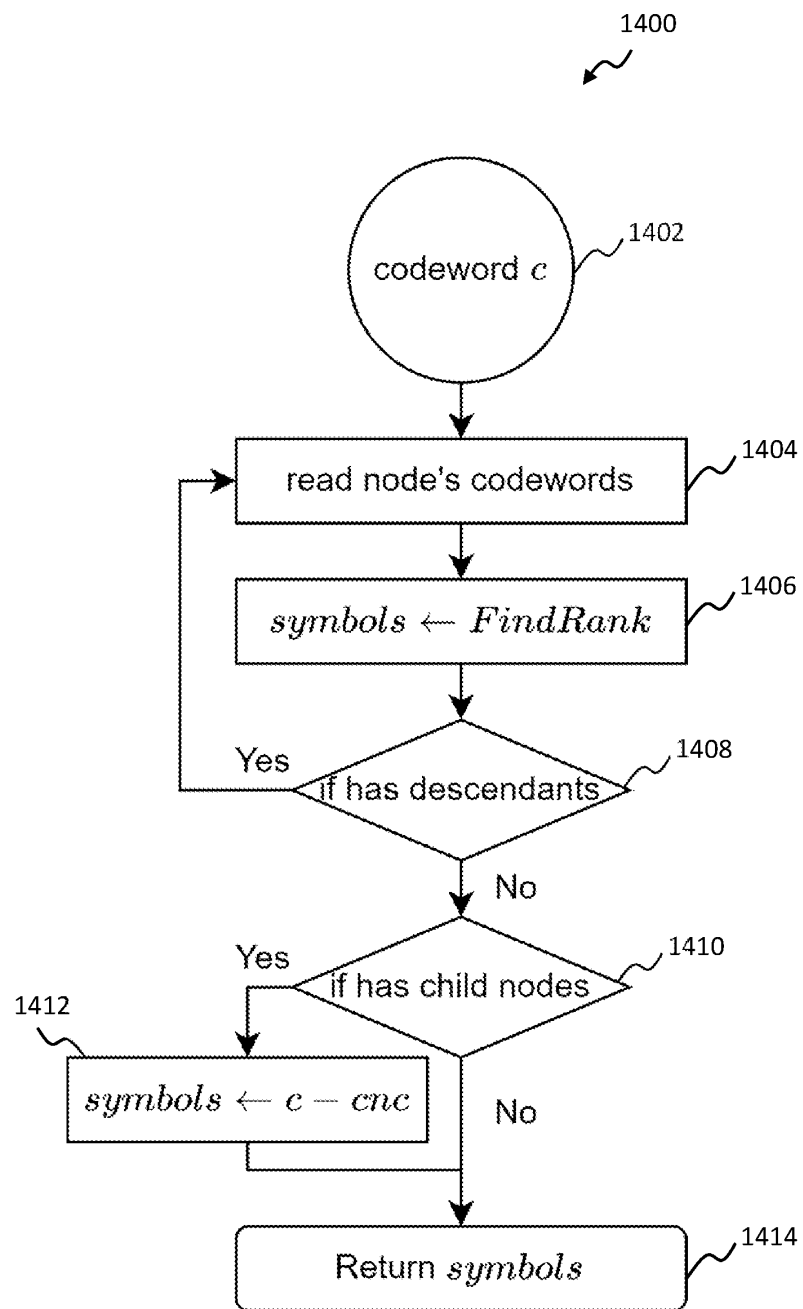


FIG. 14

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 *symbols*

```

1 Function Decoding(c, d, a) is
2   symbols  $\leftarrow \emptyset$ ;
3   nc, ns, na, ng  $\leftarrow n[d][a]$ ;
4   curSyb  $\leftarrow \text{FindRank}(c, nc)$ ;
5   if current node has descendants then
6     a  $\leftarrow na[curSyb]$ ;
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  $\leftarrow nc[curSyb]$ ;
12      symbols  $\leftarrow c - cnc$ 
13    else
14      current codeword has been decompressed
15    end
16  end
17 end

```

FIG. 15

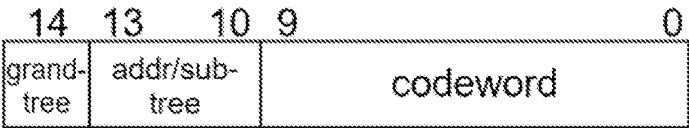


FIG. 16

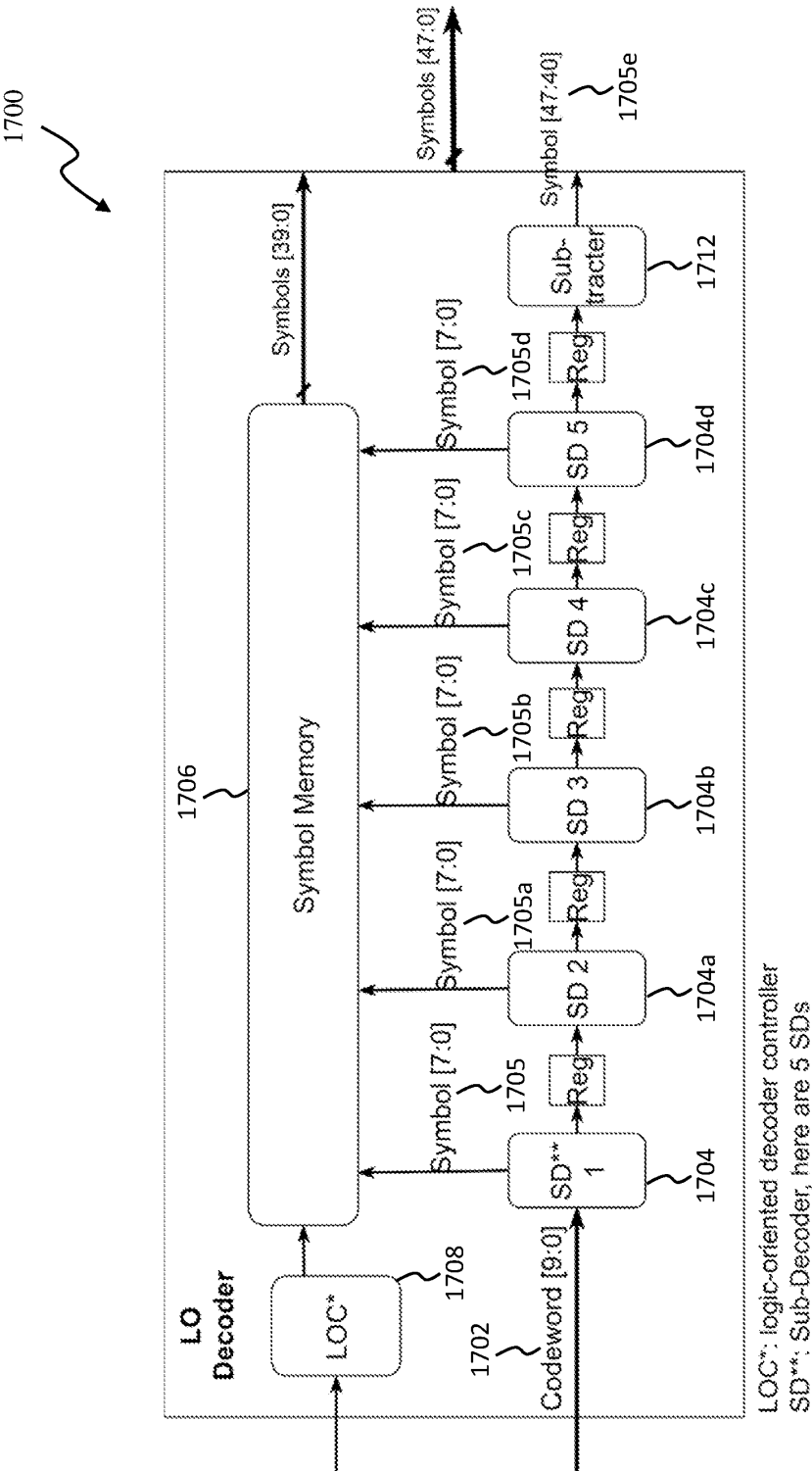


FIG. 17

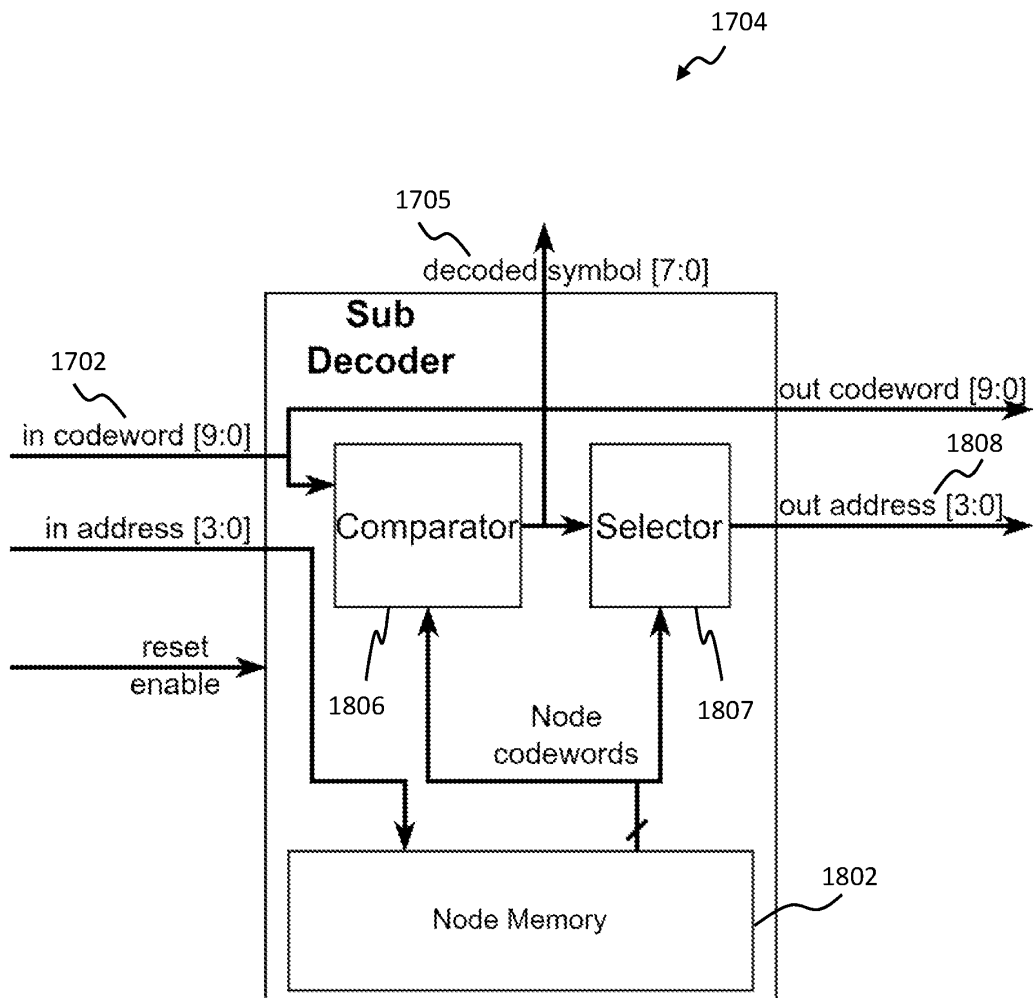


FIG. 18

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 Entries 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 Entries 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	543	33
	LUT RAM	2,585	1,342	2,797	1,236
	BRAM	2	1	2	1
	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

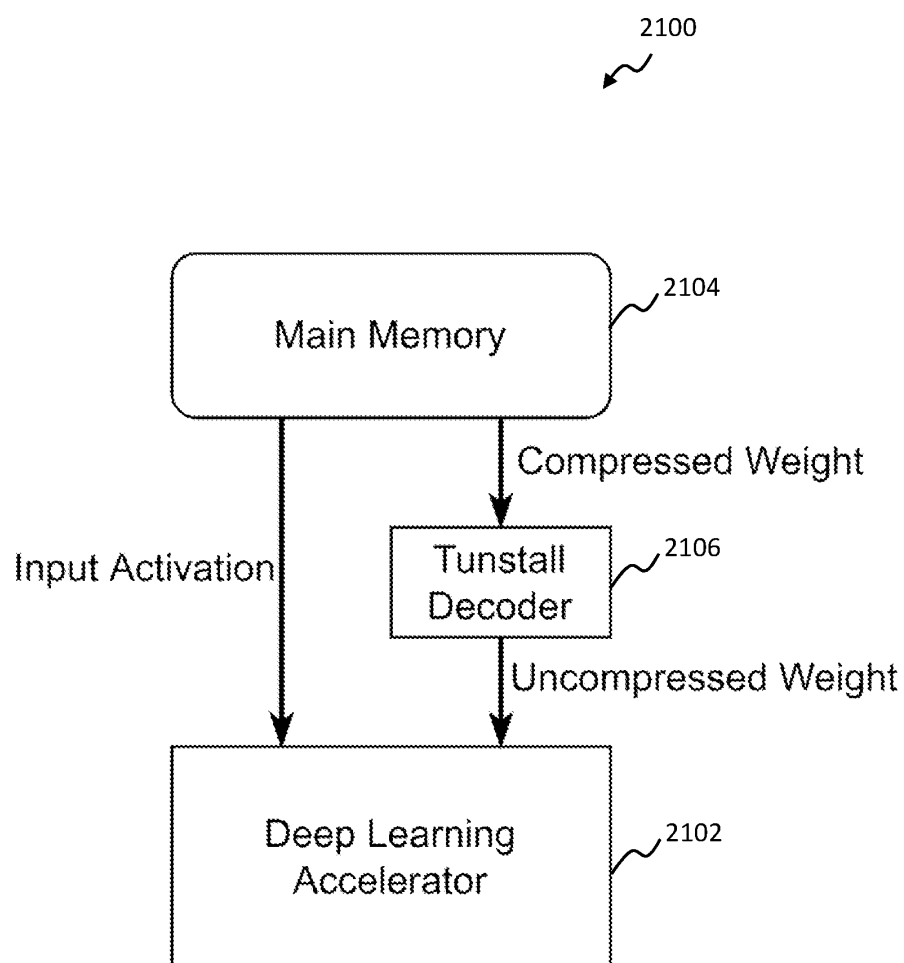


FIG. 21

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 codeword)			Estimated Power Consumption (pJ)
		<i>maximum</i>	<i>average</i>	<i>minimum</i>	
ResNet-50	<i>full precision (32-bit)</i>	11,180,732.0	2,108,203.7	26,188.6	1,707,645,000.0
	<i>Quantized 8-bit</i>	2,795,183.0	527,050.9	5,047.0	426,911,250.0
	<i>Quantized 4-bit</i>	1,397,591.5	263,525.5	2,523.5	213,455,625.0
	<i>Quantized 4-bit with Tunstall LO decoder</i>	585,622.7	113,640.9	4,733.8	46,893,720.0
	<i>Quantized 4-bit with Tunstall MO decoder</i>	571,026.7	119,114.3	16,942.6	16,266,349.2
	<i>Quantized 4-bit with Huffman coding</i>	521,362.3	108,522.5	3,520.9	17,412,582.0
MobileNet-v2	<i>full precision (32-bit)</i>	5,756,796.0	270,116.0	1,124.0	214,742,250.0
	<i>Quantized 8-bit</i>	1,439,199.0	67,529.0	281.0	53,640,355.0
	<i>Quantized 4-bit</i>	719,599.5	33,764.5	140.5	26,842,770.0
	<i>Quantized 4-bit with Tunstall LO decoder</i>	260,755.8	17,936.5	490.0	7,146,345.2
	<i>Quantized 4-bit with Tunstall MO decoder</i>	265,611.3	23,877.6	6,546.0	3,902,160.6
	<i>Quantized 4-bit with Huffman coding</i>	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		
		<i>maximum</i>	<i>average</i>	<i>minimum</i>
ResNet-50	<i>off-chip DRAM</i>	8,385,549	1,581,153	15,141
	<i>on-chip MRAM</i>	2,795,183	527,051	5,047
	<i>Quantized 4-bit with Tunstall LO decoder</i>	1,084,854	159,833	4,146
	<i>Quantized 4-bit with Tunstall MO decoder</i>	424,494	85,568	4,442
	<i>Quantized 4-bit with Huffman coding</i>	4,170,894	868,124	27,753
MobileNet-v2	<i>off-chip DRAM</i>	4,317,597	202,587	843
	<i>on-chip MRAM</i>	1,439,199	67,529	281
	<i>Quantized 4-bit with Tunstall LO decoder</i>	226,658	14,537	218
	<i>Quantized 4-bit with Tunstall MO decoder</i>	208,397	15,036	1,216
	<i>Quantized 4-bit with Huffman coding</i>	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

31 30 29										20 19										10 9										0									
1	1	0	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	1	0					
1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	0	0	1	0	0	0	1	1					

FIG. 25A

31 30		25				20				15				10				5				0			
1	1	0	1	0	1	1	1	0	1	1	0	1	1	0	0	1	1	1	1	1	0	1	1	1	
1	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0

FIG. 25B

TABLE VII
TUNSTALL AND HUFFMAN SOFTWARE DECODING TIME (MICROSECOND)

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

FIG. 26