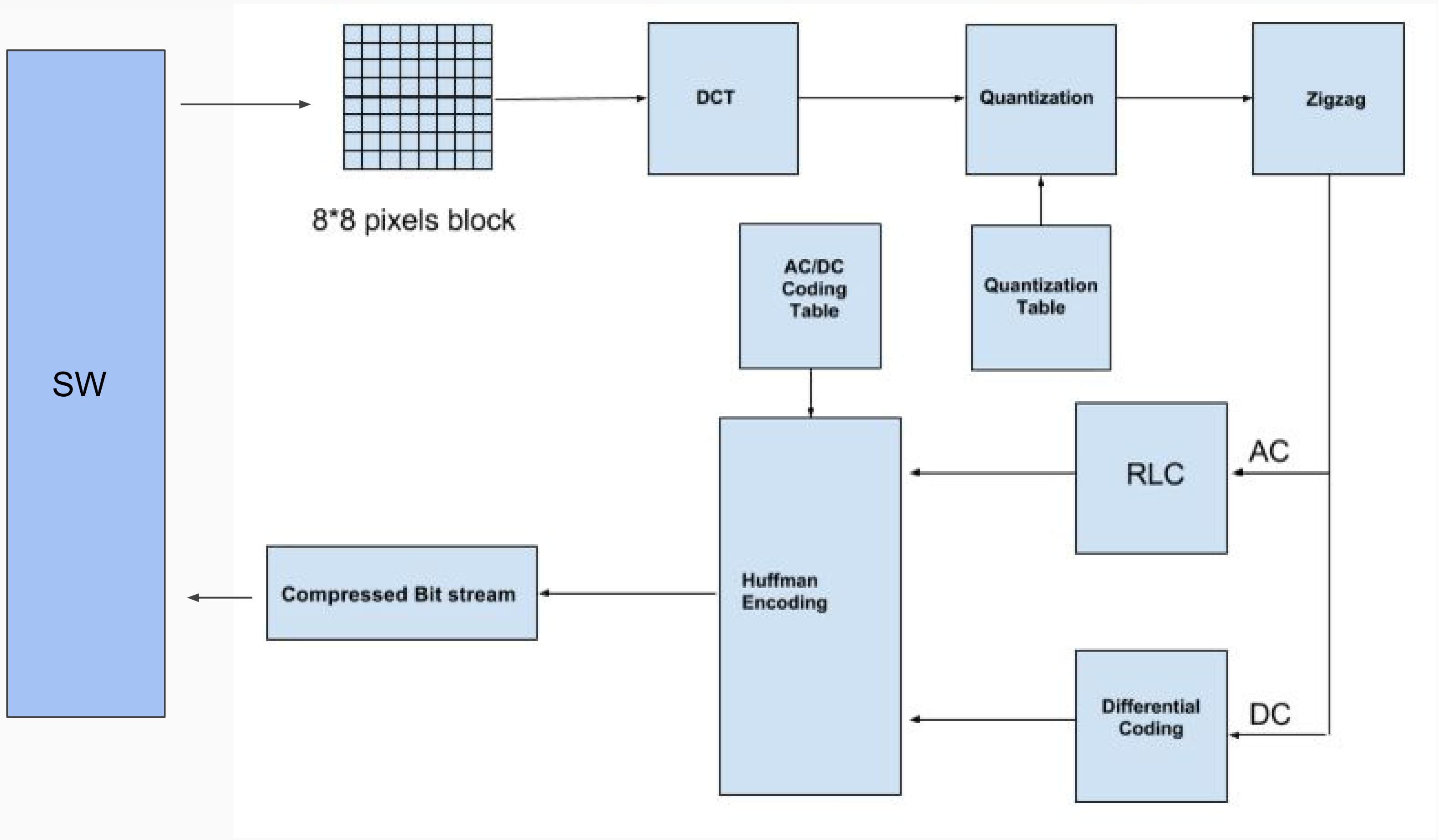
FPGA JPEG Image Compression Accelerator

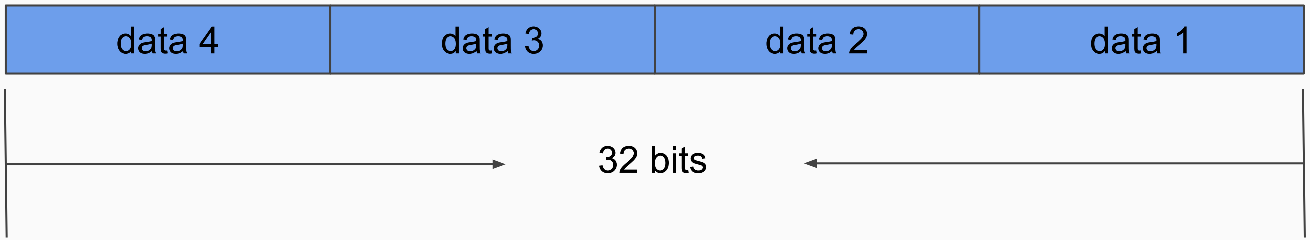
EECS 4840
Yuxiang Chen, Xinyi Chang, Song Wang, Nan Zhao
Electrical Engineering, Columbia University
Prof. Stephen A. Edwards

JPEG Image Compression



Software to FPGA

- 64 pixels x 8 bits/pixel = 256 bits
- 256 bits / 32 bits/stream = 16 stream
- Input = buffer[i] + buffer[i+1] << 8 + buffer[i+2] << 16 + buffer[i+3] << 24
- Decode the 32 bits data in HW
- HW waits for 16 write states, then go to the next state - computing DCT



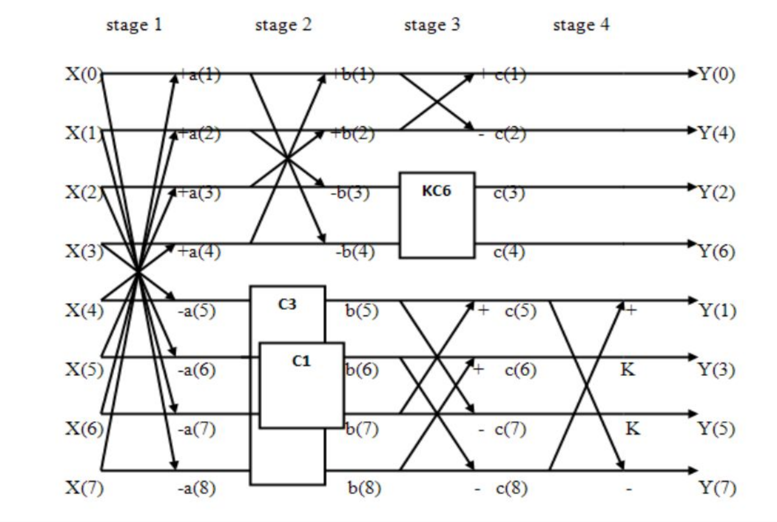
DCT with Loeffler Algorithm

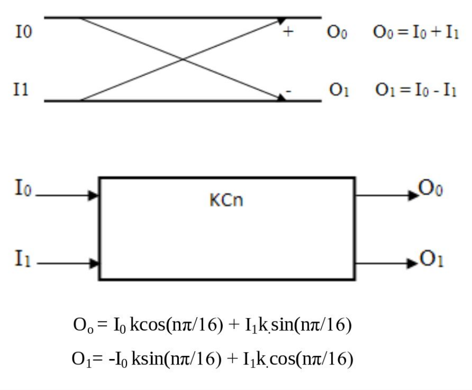
$$y(k) = w(k)\sum_{n=1}^{N} x(n)\cos\left(\frac{\pi}{2N}(2n-1)(k-1)\right), \quad k = 1, 2, \dots, N,$$

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 1, \\ \sqrt{\frac{2}{N}}, & 2 \le k \le N, \end{cases}$$

Loeffler Algorithm

- Number of multiplications reach the theoretical low limit.
- 4 Stages
- MultAddSub Blocks





[1]M. Jridi and A. Alfalou,

Canonical signed digit (CSD) representation

$$y(k) = w(k)\sum_{n=1}^{N} x(n)\cos\left(\frac{\pi}{2N}(2n-1)(k-1)\right), \quad k = 1, 2, \ldots, N,$$

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}}, & k = 1, \\ \sqrt{\frac{2}{N}}, & 2 \le k \le N, \end{cases}$$

CSD

- Signed representation containing the fewest number of nonzero bits
- Effective way to carry out constant multiplier for DCT.
- Number of additions and subtractions will be minimized.
- Identified common elements in CSD constant coefficients and shared required resource

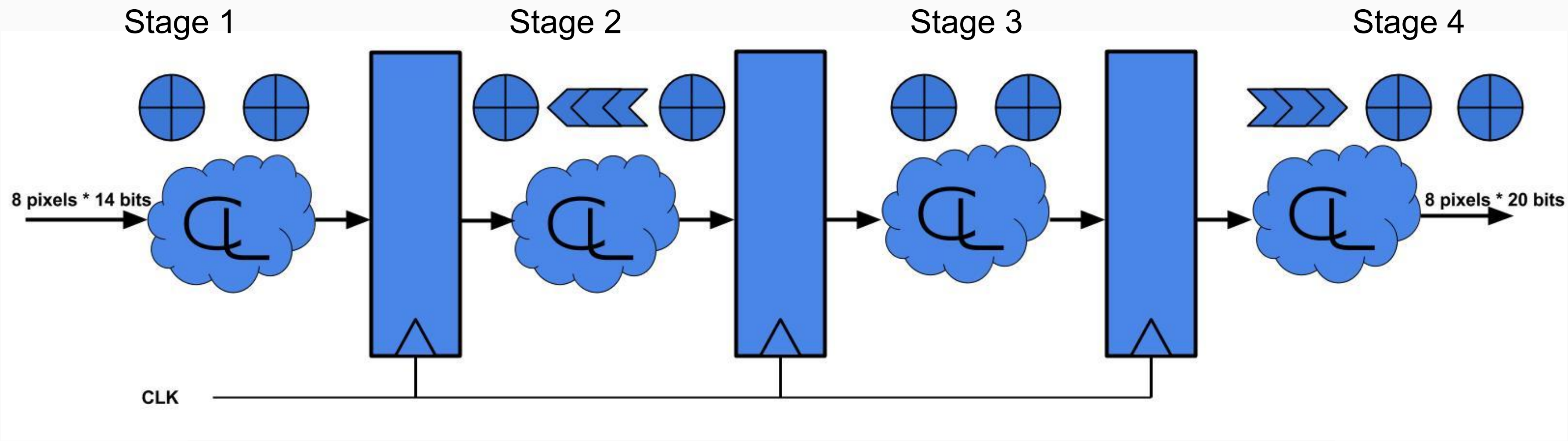
X = 2^a ± 2^b ± 2^c ±…..

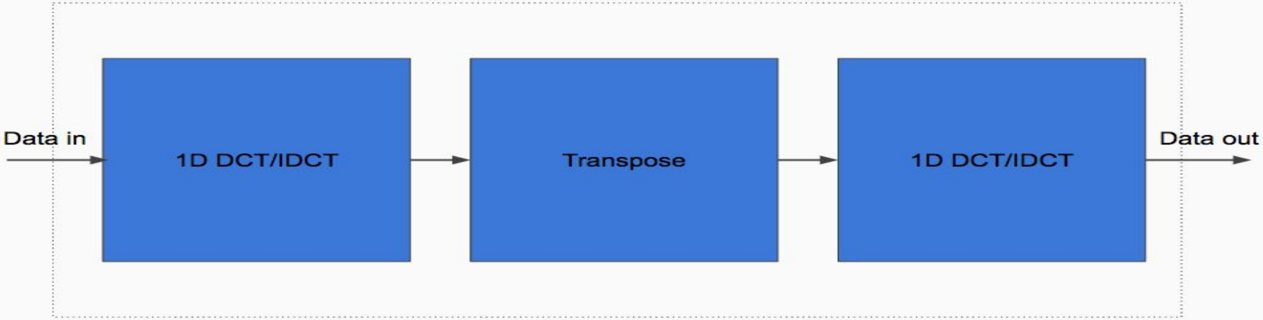
TABLE I
8-POINT DCT FIXED COEFFICIENT REPRESENTATION

Real value	Decimal	Natural binary	Partial products	CSD	Partial products
$cos\frac{3\pi}{16}$	106	01101010	4	+0-0+0+0	4
$sin\frac{3\pi}{16}$	71	01000111	4	0+00+00-	3
$cos\frac{\pi}{16}$	126	01111110	6	+00000-0	2
$sin\frac{\pi}{16}$	25	00011001	3	00+0-00+	3
$cos\frac{6\pi}{16}$	49	00110001	3	0+0-000+	3
$sin\frac{6\pi}{16}$	118	01110110	5	+000-0-0	3
$\sqrt{(2)}$	181	10110101	5	+0-0-0+0+	5
Total Partial products		30		23	

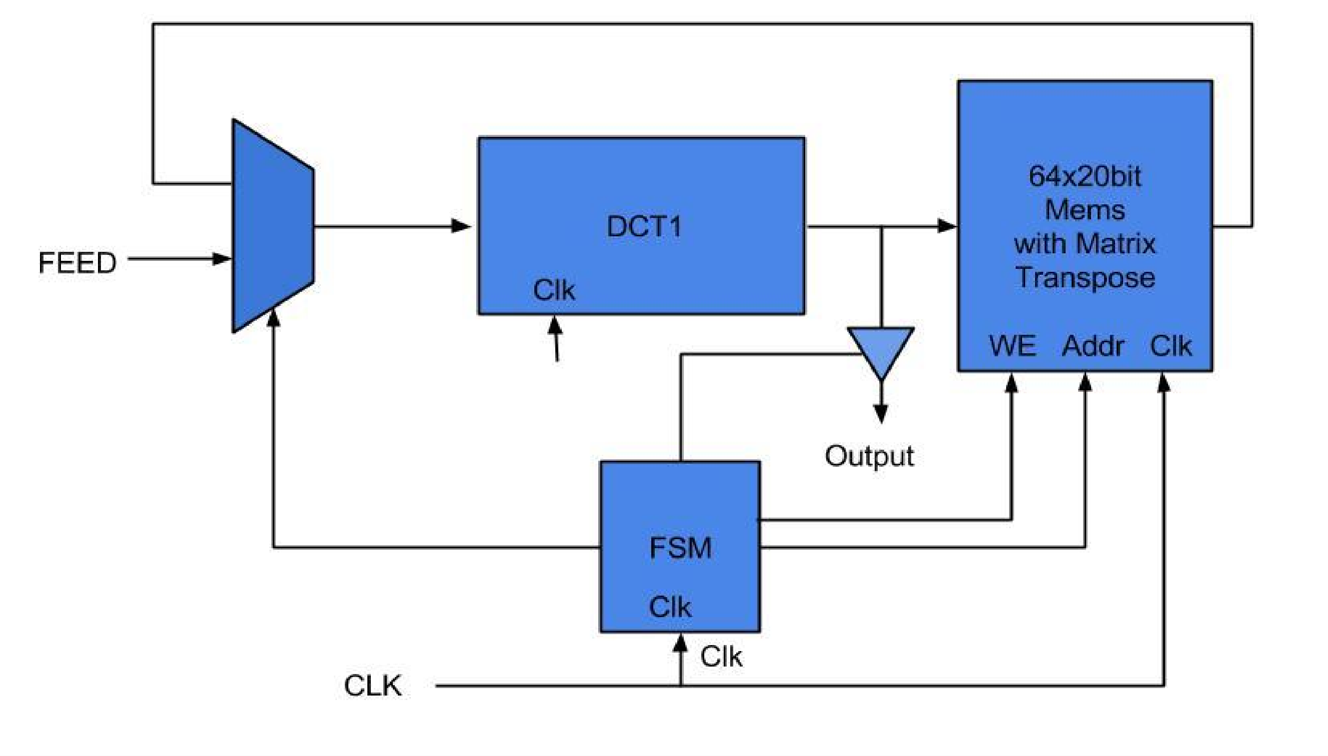
[1]M. Jridi and A. Alfalou,

RTL Block Diagram for DCT-1 and DCT-2





RTL Block Diagram for DCT-2



Quantization

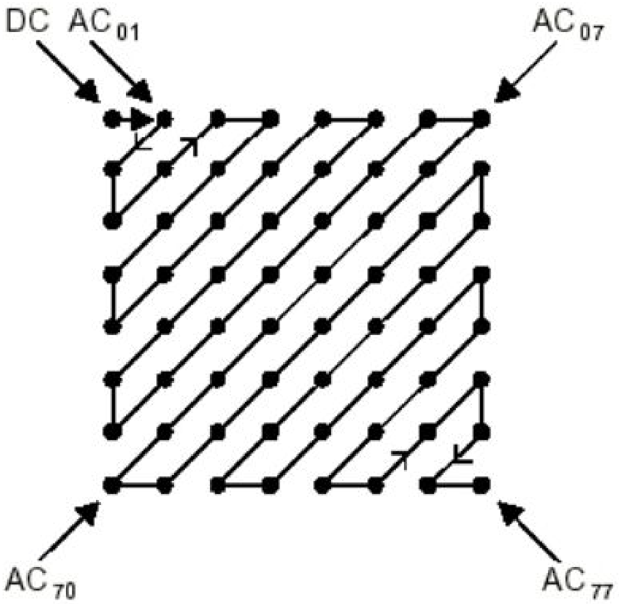
Table 2: Modified Normalization Matrix For Hardware Simplification

16	16	16	16	32	64	64	64
16	16	16	16	32	64	64	64
16	16	16	32	32	64	64	64
16	16	32	32	32	64	64	64
32	32	32	64	128	128	128	128
64	64	64	64	128	128	128	128
128	128	128	128	128	128	128	128
128	128	128	128	128	128	128	128

- The step where we actually throw away data.
- Reduce most of the less important high frequency DCT coefficients to zero,
- Lower numbers in the upper left direction and large numbers in the lower right direction

Zigzag

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63



Zigzag Scan Order

- Obtain the one-dimensional vectors with a lot of consecutive zeroes

RLC (Run Length Encoding)

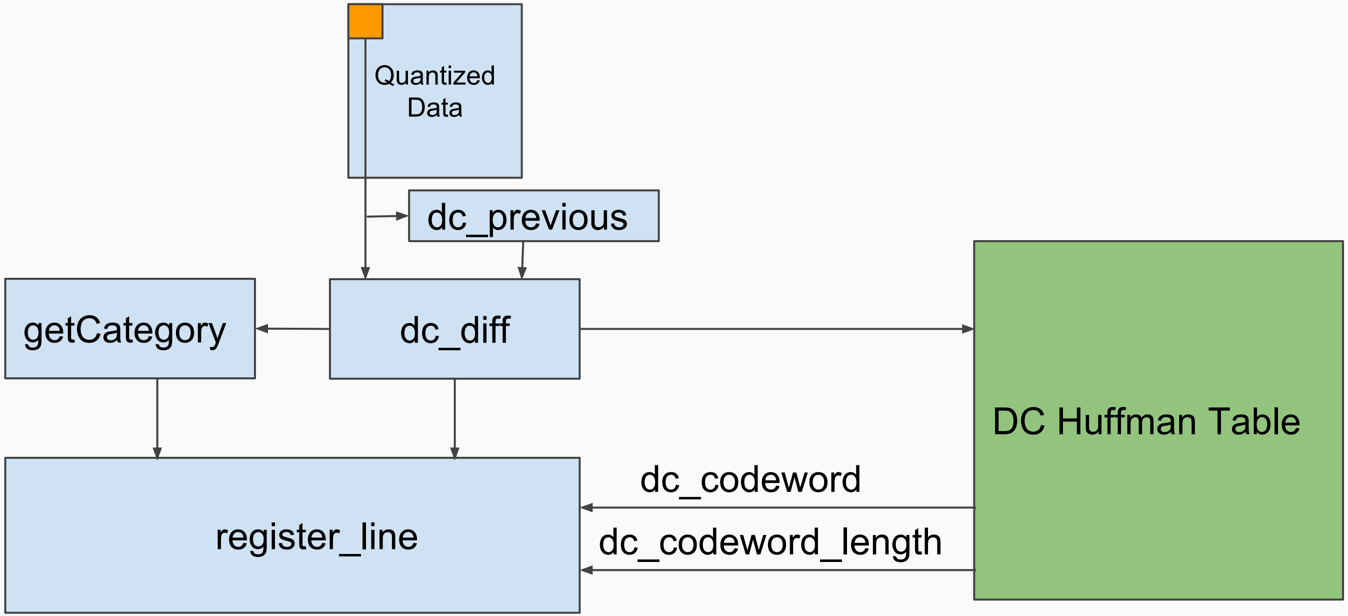
Run	Category	Bit Value	……	Run	Category	Bit Value	EOB

- Run represents the number of previous consecutive zeros.
- Category represents the bit value length of non-zero value.
- End with EOB when last bits are 0..

DC Huffman Encoding

Algorithm:

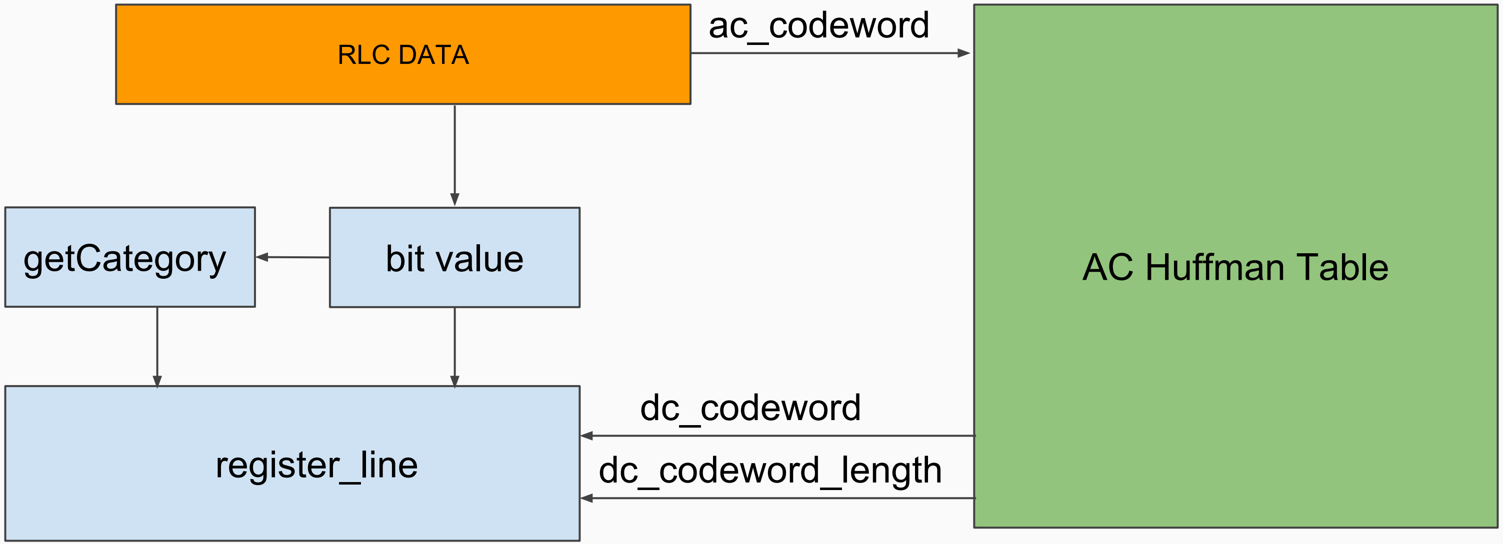
- ➢ dc_diff = dc_current -dc_previous
- ➢ dc_diff_length = **getCategory**(dc_diff)
- ➢ dc_codeword = **dc_lookup_table**(dc_diff)
- ➢ register_line = register_line + (ac_codeword << category) + dc_diff



AC Huffman Encoding

Algorithm:

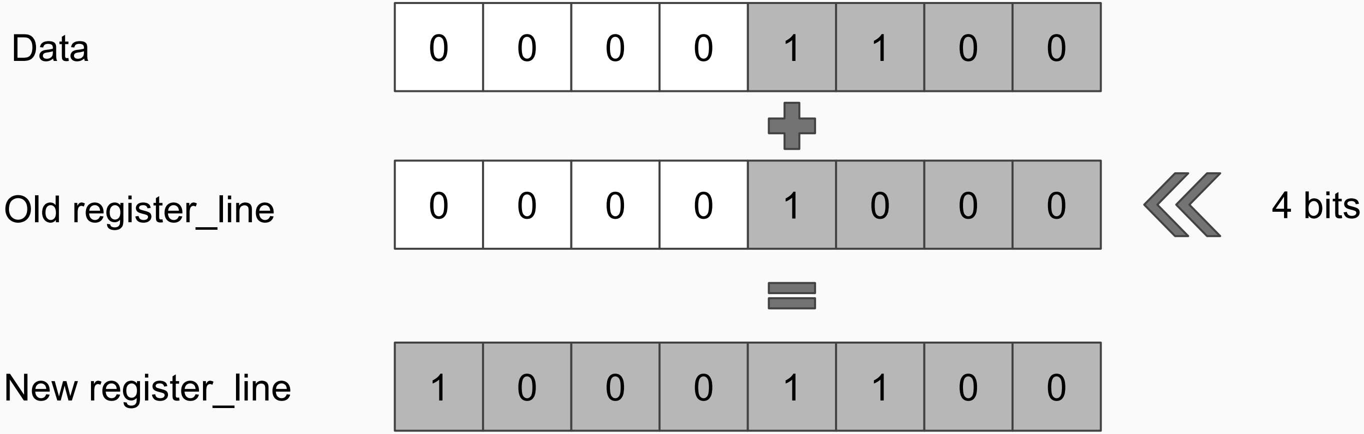
- ➢ ac_diff_length = **getCategory**(bit_value)
- ➢ ac_codeword = **ac_lookup_table**()
- ➢ register_line = register_line + (ac_codeword << category) + bit_value



Bit Stream Compression

Algorithm:

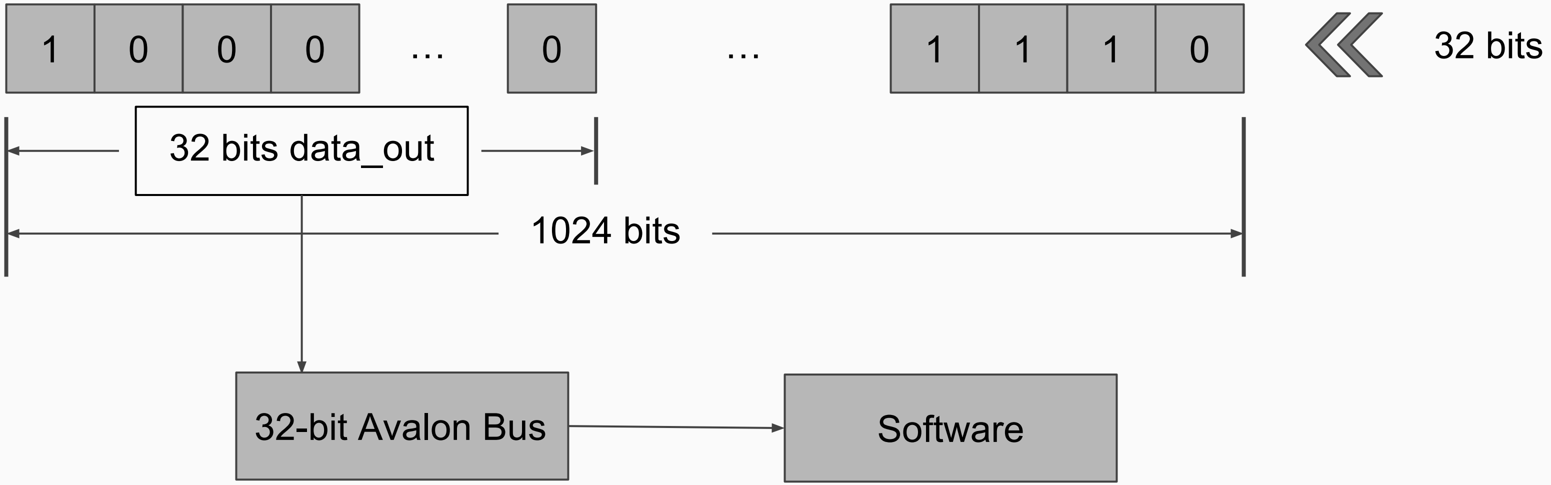
- ➢ Initialized a 1024-bit length register_line,
- ➢ While (there is data):
  - ■ register_line = (register_line << data_length) + data;
  - ■ total_line_size += data_length



Compressed Data to Software

Algorithm:

- ➢ register_line << register_length,
- ➢ do:
  - ■ data_back = register_line[1023:991]
  - ■ Register_line << 32 bits
- ➢ while(data != 0)



Result

1. DCT input:

```
Start
Input:Block:1
100, 0, 0, 100, 0, 77, 0, 88,
100, 0, 0, 0, 0, 0, 0, 0,
100, 0, 0, 0, 0, 0, 0, 0,
100, 200, 0, 0, 0, 0, 0, 32,
100, 0, 0, 0, 0, 0, 0, 0,
100, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 100,
```

2. DCT output:

input =

-986	102	133	55	90	15	30	-37
75	52	15	44	31	35	17	-37
-3	-119	-44	-44	40	23	51	-25
12	-8	-28	27	30	53	35	-14
82	-15	29	-38	6	-48	-14	-59
43	16	-7	-18	-35	-40	-36	-48
18	-23	6	4	36	24	31	-5
-45	-47	-41	10	24	51	40	20

3.Quantization output:

output =

-61	6	8	3	2	0	0	0
4	3	0	2	0	0	0	0
0	-7	-2	-1	1	0	0	0
0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

4. Zigzag output:

```
Columns 1 through 22
 -61  6  4  0  3  8  3  0  -7  0  2  0  -2  2  2  0  0  -1  0  0  0  0
Columns 23 through 44
 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Columns 45 through 64
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

5. RLC output:

DC: -61 -> (14) ->	111000001
AC: (0, 6)->(0,3,6)->(100,110)->38	100110
(0, 4)->(0,3,4)->(100,100)->36	100100
(1, 3)->(1,2,3)->(11011,11)->111	1101111
(0, 8)->(0,4,8)->(1011,1000)->184	10111000
(0, 3)->(0,2,3)->(01,11)->7	0111
(1,-7)->(1,3,000)->(1111001,000)->968	11110010000
(1,2)->(1,2,2)->(11011,10)->110	1101110
(1,-2)->(1,2,2)->(11011,01)->109	1101101
(0,2)->(0,2,2)->(01,10)->6	0110
(0,2)->(0,2,2)->(01,10)->6	0110
(2,-1)->(2,1,0)->(11100,0)->56	111000
(7,1)->(7,1,1)->(11111010,1)->501	111110101
(0,0)->(1010)->10	1010

6. Bitstream output:

```
Start
Input:Block:1
100, 0, 0, 100, 0, 77, 0, 88,
100, 0, 0, 0, 0, 0, 0, 0,
100, 0, 0, 0, 0, 0, 0, 0,
100, 200, 0, 0, 0, 0, 0, 32,
100, 0, 0, 0, 0, 0, 0, 0,
100, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 100,

Output:
11100000010100110100100110111110111000001111111001000011011101101101011001101110001111101011010
```

Reference

[1] M. Jridi, A. Alfalou, "A low-power, high-speed DCT architecture for image compression: Principle and implementation," 18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC), 2010, pp. 304-309.

[2] Y. H. Chen , T. Y. Chang and C. Y. Li , "Highthroughput DA-based DCT with high accuracy error-compensated adder tree" , IEEE Trans. VeryLarge Scale Integr. (VLSI) Syst. , vol. 19 , no. 4 , pp.709 -714 , 2011

[3] V. Gupta, D. Mohapatra, A. Raghunathan and K. Roy , "Low-power digital signal processing using approximate adders" , IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. , vol. 32 , no. 1 , pp.124 -137 , 2013

[4] P. Kulkarni, P. Gupta, and M. D. Ercegovac, “Trading accuracy for power in a multiplier architecture,” J. Low Power Electron., vol. 7, no. 4, pp.490–501, 2011.

[5] Y. V. Ivanov and C. J. Bleakley, “Real-time h.264 video encoding in software with fast mode decision and dynamic complexity control,” ACM Trans. Multimedia Comput. Commun. Applicat., vol. 6, pp. 5:1–5:21, Feb. 2010.

[6] Wei-Yi We, National Taiwan University, “An Introduction to Image Compression”