FPGA Implementation of JPEG Compression

ES204 - Digital Systems

Professor - Joycee M. Mekie

TEAM NAME: Discrete Cosine Sabha

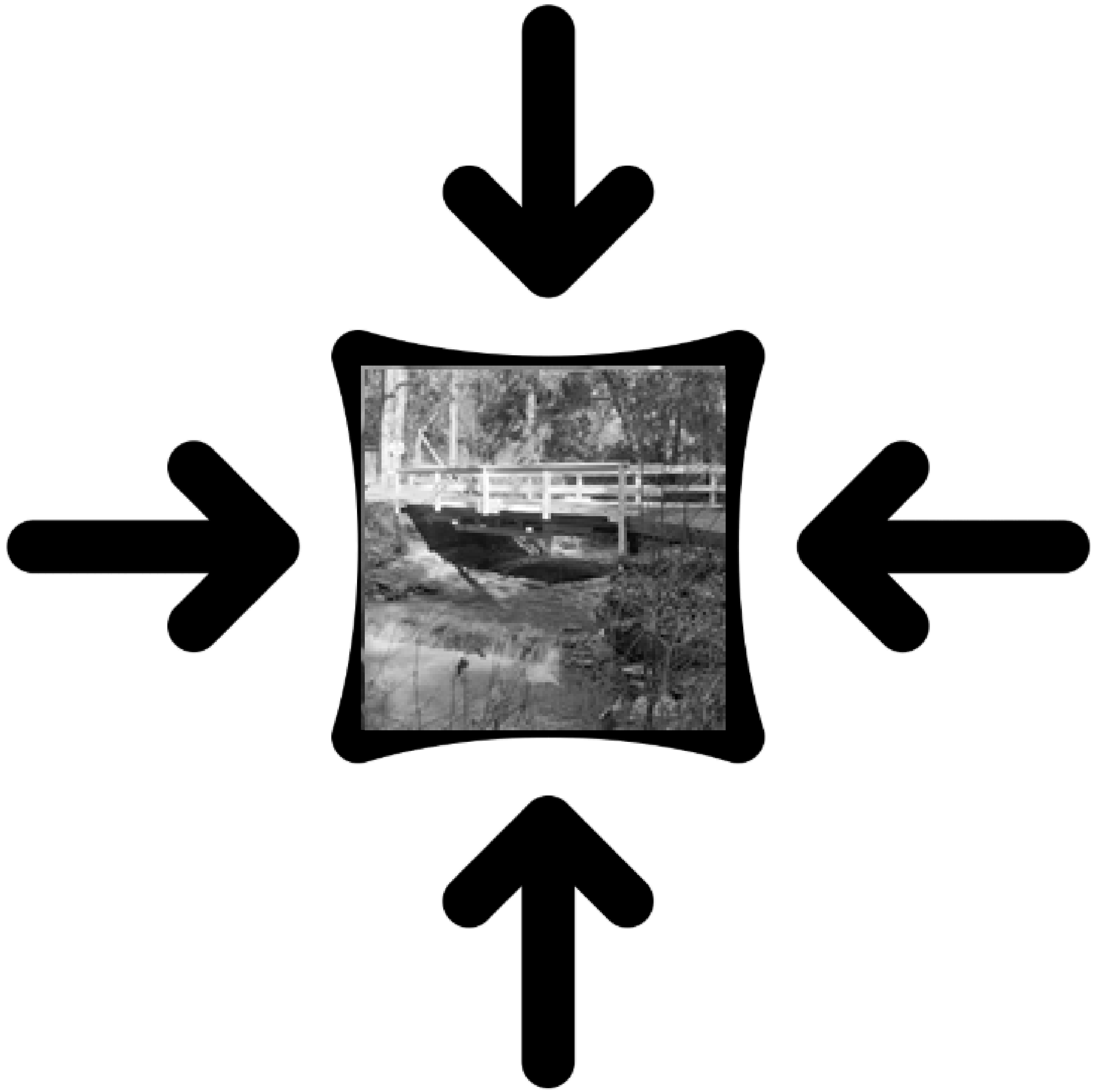
Jaskirat Singh Maskeen (23110146)

Nishchay Bhutoria (23110222)

Romit Mohane (23110279)

Soham Gaonkar (23110314)

Indian Institue of Technology Gandhinagar
Palaj, Gujarat - 382355



Source: https://github.com/Reckadon/JPEG-Compression/

Problem Statement

Given a 128x128, 8-bit image, compress it using JPEG compression.

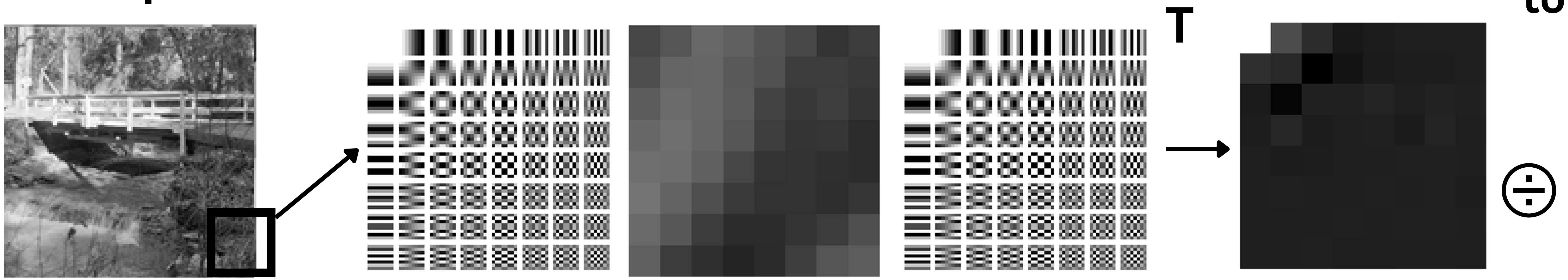
What is JPEG Compression?

Our eyes are bad at perceiving color. They are better at seeing changes in brightness. Also, they are bad at seeing high-frequency changes in texture. JPEG Compression algorithm takes advantage of these facts to store images in a fraction of the original file size.

How to extract the high-frequency components?

Discrete Cosine Transform (DCT !!)

Our Pipeline:



top kxk coefficients

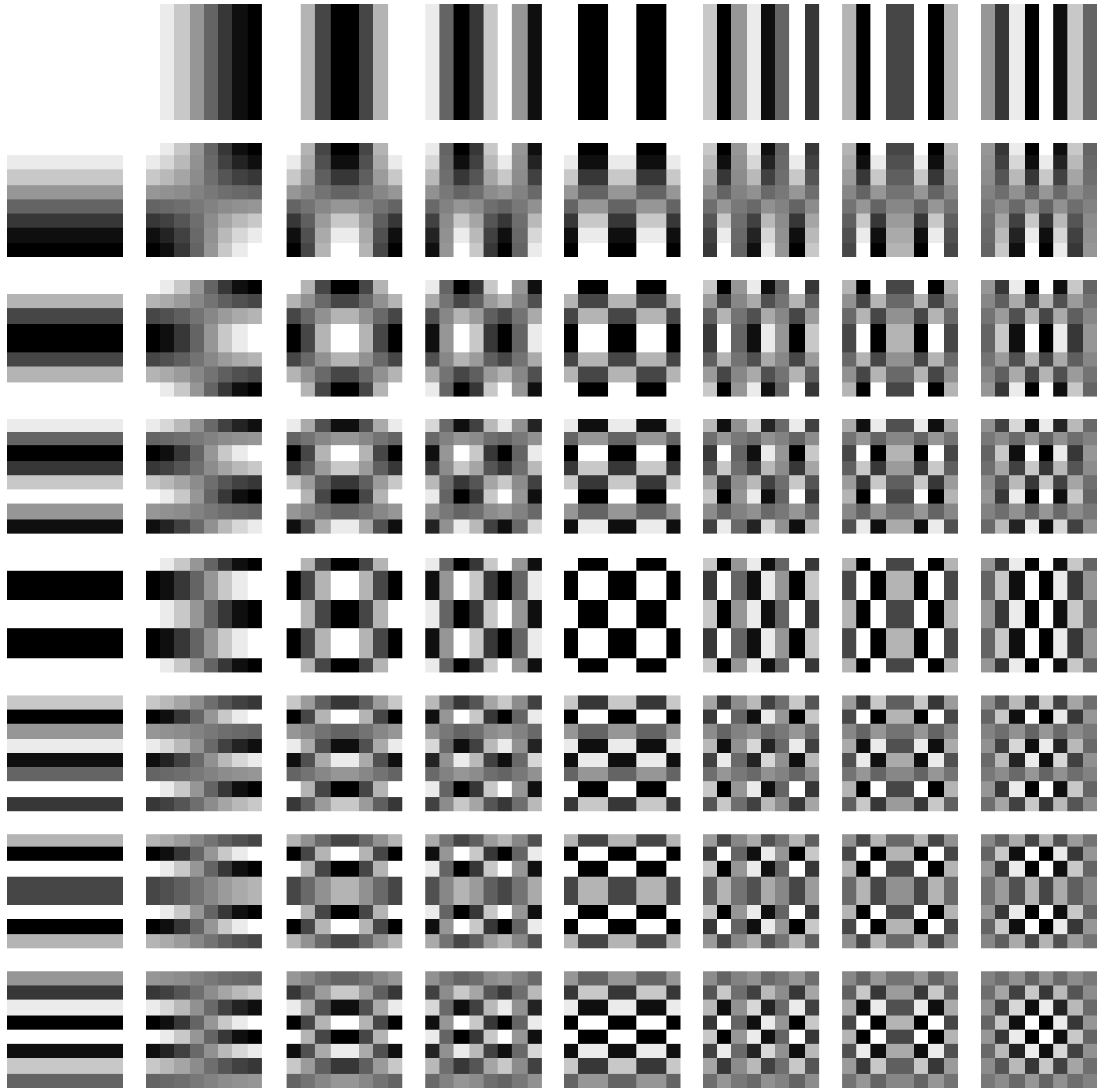
OR

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

What is DCT ?

Discrete Cosine Transform extracts the high-frequency components.
Or **DCT = Separate What's Important from What's Not**
For every 8x8 block in the image, we compute its DCT coefficients. We get a resultant 8x8 matrix.

The entries in this matrix represent the amount of respective images (see figure) in the original image. The figure shows the 64 basis vectors of any 8x8 image.



Adding the images on the right weighted by the computed coefficients gives us the original 8x8 block

Where does the compression happen ?

The compression happens when we divide the DCT coefficients obtained by a standard quantization table.

For example:

DCT Coefficients								Quantization Table								Quantized Coefficients							
586	119	38	-18	-8	-2	0	0	16	11	10	16	24	40	51	61	37	11	4	-1	0	0	0	0
41	23	-82	-31	-10	-5	-5	0	12	12	14	19	26	58	60	55	3	2	-6	-2	0	0	0	0
-9	-68	5	5	10	-2	7	3	14	13	16	24	40	57	69	56	-1	-5	0	0	0	0	0	0
-5	21	-8	1	3	-9	11	-2	14	17	22	29	51	87	80	62	0	1	0	0	0	0	0	0
0	-9	-5	1	-2	6	5	0	18	22	37	56	68	109	103	77	0	0	0	0	0	0	0	0
1	2	0	-2	3	-1	2	-5	24	35	55	64	81	104	113	92	0	0	0	0	0	0	0	0
0	-2	-1	2	1	6	3	-1	49	64	78	87	103	121	120	101	0	0	0	0	0	0	0	0
-1	0	3	-3	-2	0	1	-1	72	92	95	98	112	100	103	99	0	0	0	0	0	0	0	0

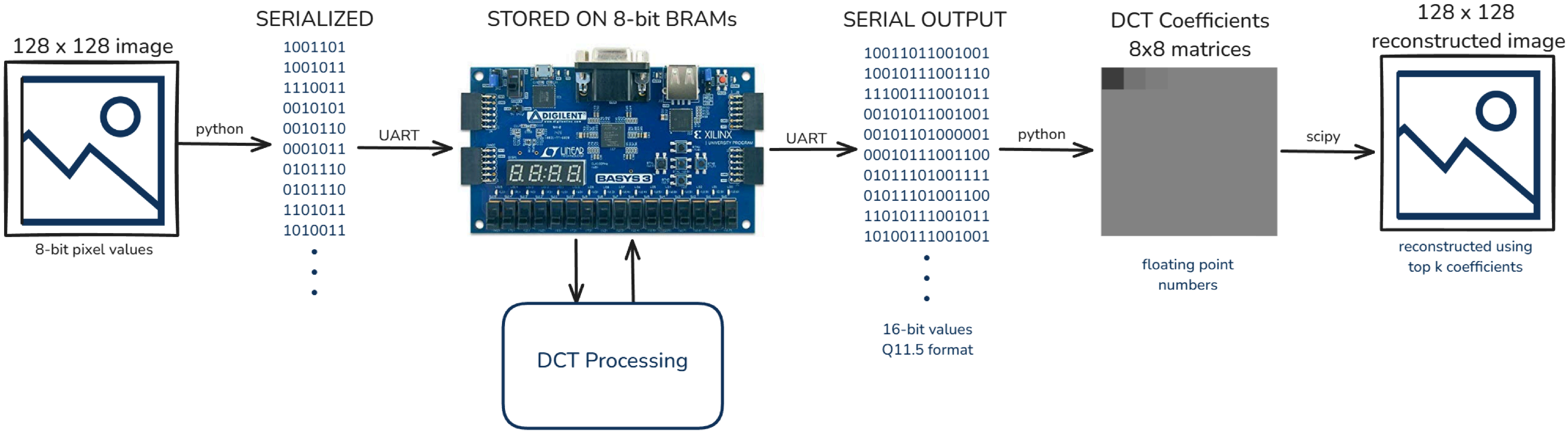
As seen in the figure, the high frequency DCT coefficients (bottom right) become zero.

Another approach for compression ?

Choose the top k x k elements and set others to zero. In this way only the prominent coefficients (dc and low frequency) are retained, while others are set to zero. This is a harsher quantization. This results in visually worse images.

```
              DCT Coefficients                   |              Top kxk (k=5)
-------------------------------------------------------------------------------------------------
 586  119   38  -18   -8   -2    0    0 |  586  119   38  -18   -8    0    0    0
  41   23  -82  -31  -10   -5   -5    0 |   41   23  -82  -31  -10    0    0    0
  -9  -68    5    5   10   -2    7    3 |   -9  -68    5    5   10    0    0    0
  -5   21   -8    1    3   -9   11   -2 |   -5   21   -8    1    3    0    0    0
   0   -9   -5    1   -2    6    5    0 |    0   -9   -5    1   -2    0    0    0
   1    2    0   -2    3   -1    2   -5 |    0    0    0    0    0    0    0    0
   0   -2   -1    2    1    6    3   -1 |    0    0    0    0    0    0    0    0
  -1    0    3   -3   -2    0    1   -1 |    0    0    0    0    0    0    0    0
```

Block Diagram of Our Approach



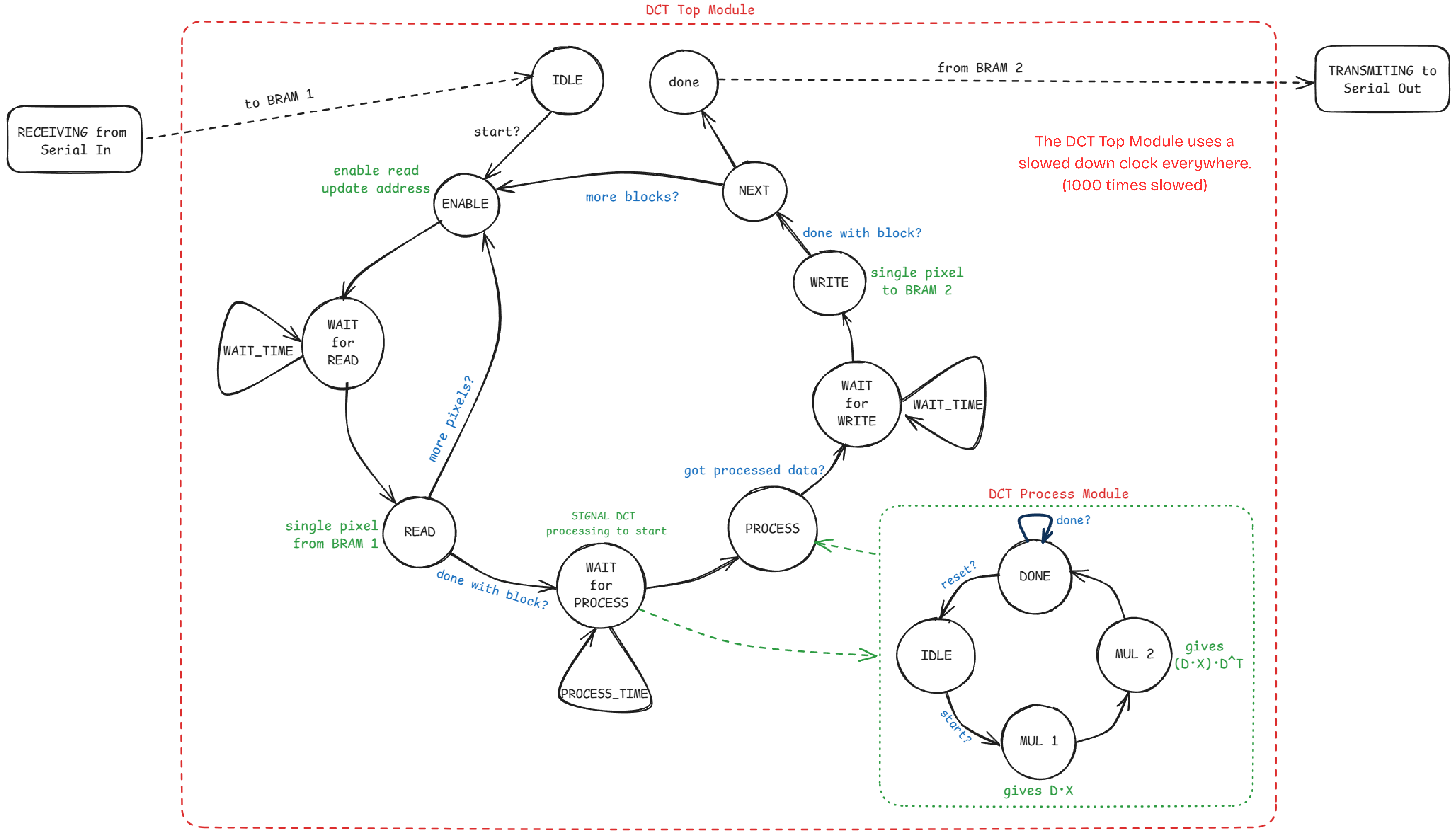
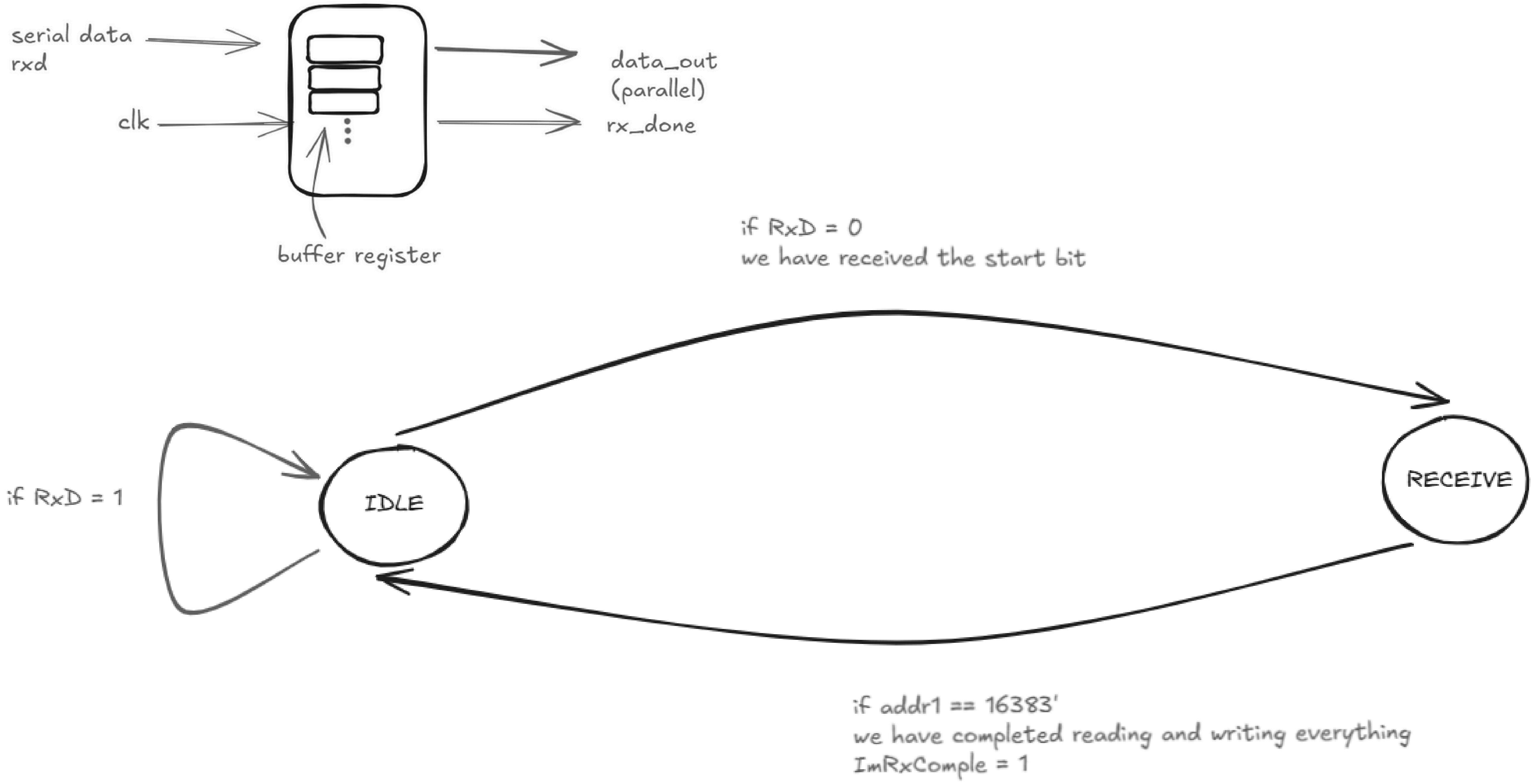

DCT Top Module
RECEIVING from Serial In
to BRAM 1
IDLE
done
from BRAM 2
TRANSMITING to Serial Out
start?
The DCT Top Module uses a slowed down clock everywhere. (1000 times slowed)
enable read
update address
ENABLE
more blocks?
NEXT
done with block?
WRITE
single pixel to BRAM 2
WAIT for READ
WAIT_TIME
more pixels?
WAIT for WRITE
WAIT_TIME
got processed data?
single pixel from BRAM 1
READ
SIGNAL DCT processing to start
PROCESS
DCT Process Module
done?
done with block?
WAIT for PROCESS
DONE
reset?
PROCESS_TIME
IDLE
MUL 2
gives (D·X)·D^T
start?
MUL 1
gives D·X

Image Receiving :

The imrx.v module receives serial data (UART) and stores it in memory. It uses oversampling for better accuracy and stores each received byte at a new memory address.



Baud Rate Counter
counter <= counter + 1;
if (counter >= div_counter - 1) counter <= 0;
→Generates ticks at 4x the baud rate (for oversampling).

Sampling
samplecounter <= samplecounter + 1;
if (samplecounter == mid_sample - 1) shift <= 1;
→ Data is Sampled at bit midpoint

Bit Shifting
if (shift) rxshiftreg <= {RxD, rxshiftreg[9:1]};
→ When the midpoint of a bit is reached, it shifts in the RxD value into a 10-bit shift register (rxshiftreg)

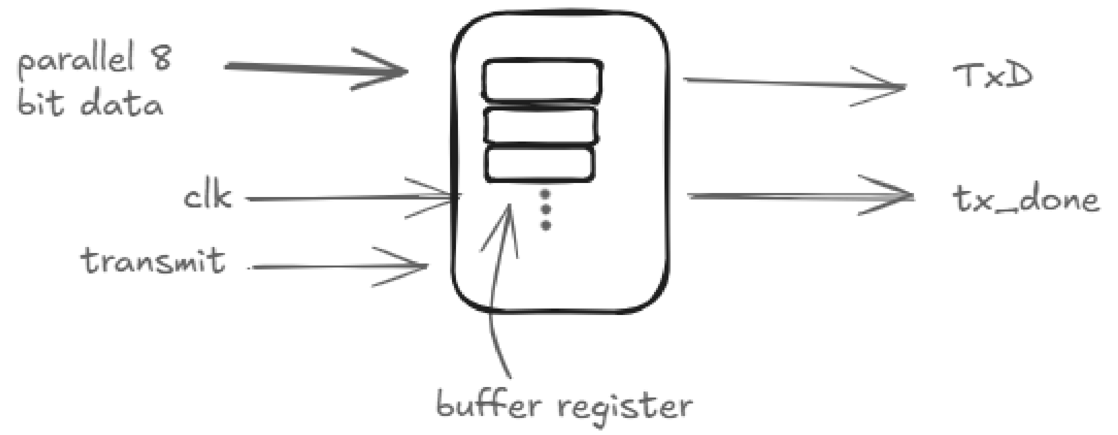
Bit Counting
if (bitcounter == div_bit - 1)
→ 10 bits received (start + 8 data + stop)

Data Storage
din <= rxshiftreg[8:1];
wea <= 1;
addr1 <= addr1 + 1;
→ Save byte to memory

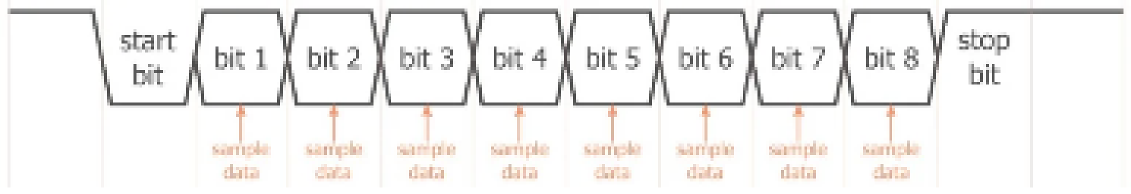
Reception Complete
ImRxComplete <= (addr1 >= 16383);
→ All data received

Image Transmitting :

UART transmitter module (imtx.v) that reads bytes from memory (dout_tx),
and sends them over a serial output (TxD) using UART protocol (start bit, 8 data bits, stop bit = 10 bits total).



timing diagram :



load a byte from BRAM
load in rightshiftreg = {1'b1, data, 1'b0}
startbit : 1
data
stopbit : 0

Every 10416 clock cycles:
TxD = rightshiftreg[0] → output 1 bit
rightshiftreg = rightshiftreg >> 1
bitcounter += 1

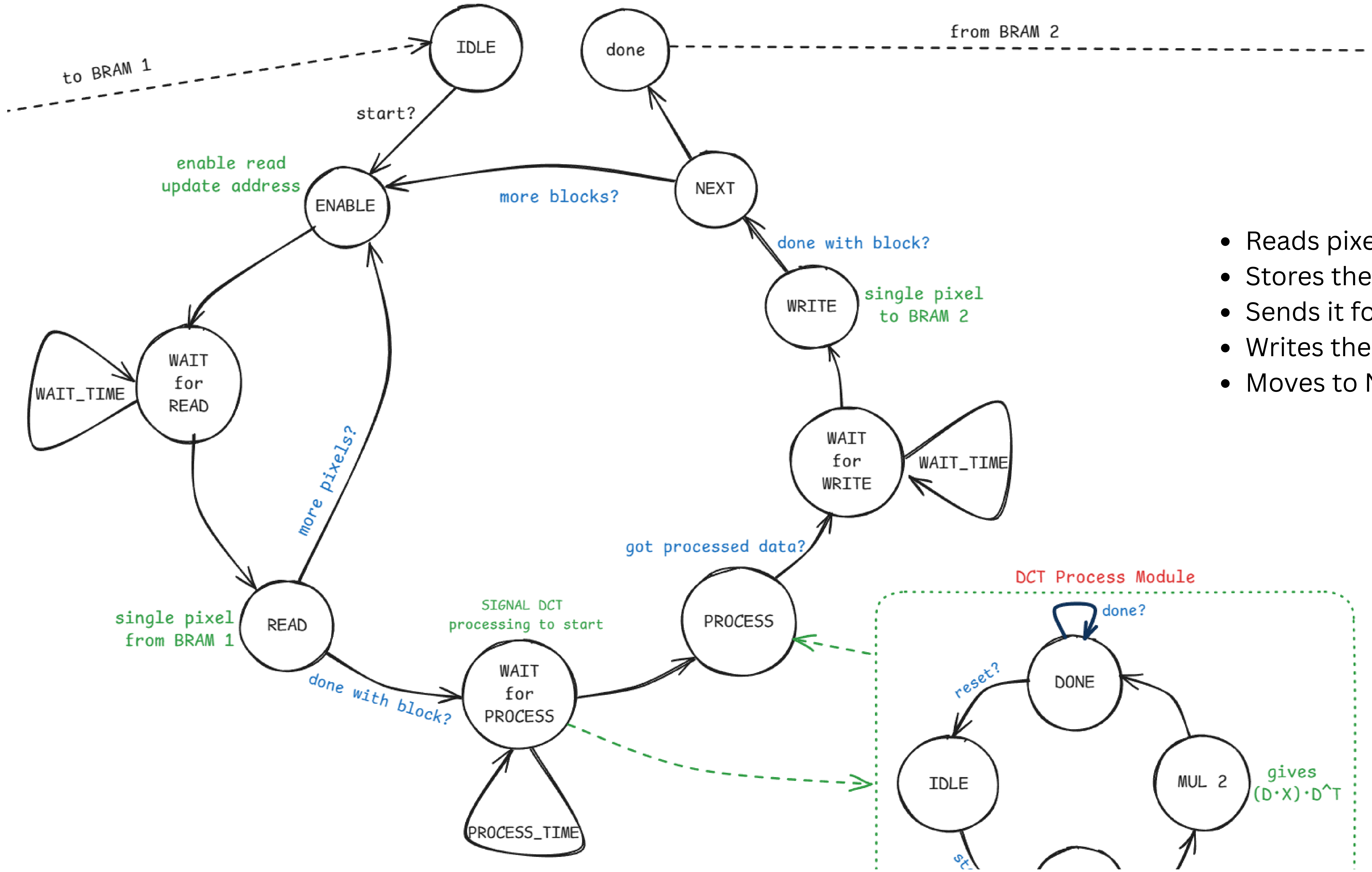
If bitcounter == 10:
nextstate = 0 (go to idle)
clear = 1 → reset bitcounter

inc_addr = 1 → go to next byte

If not yet done:
shift = 1 to keep shifting
TxD = current bit

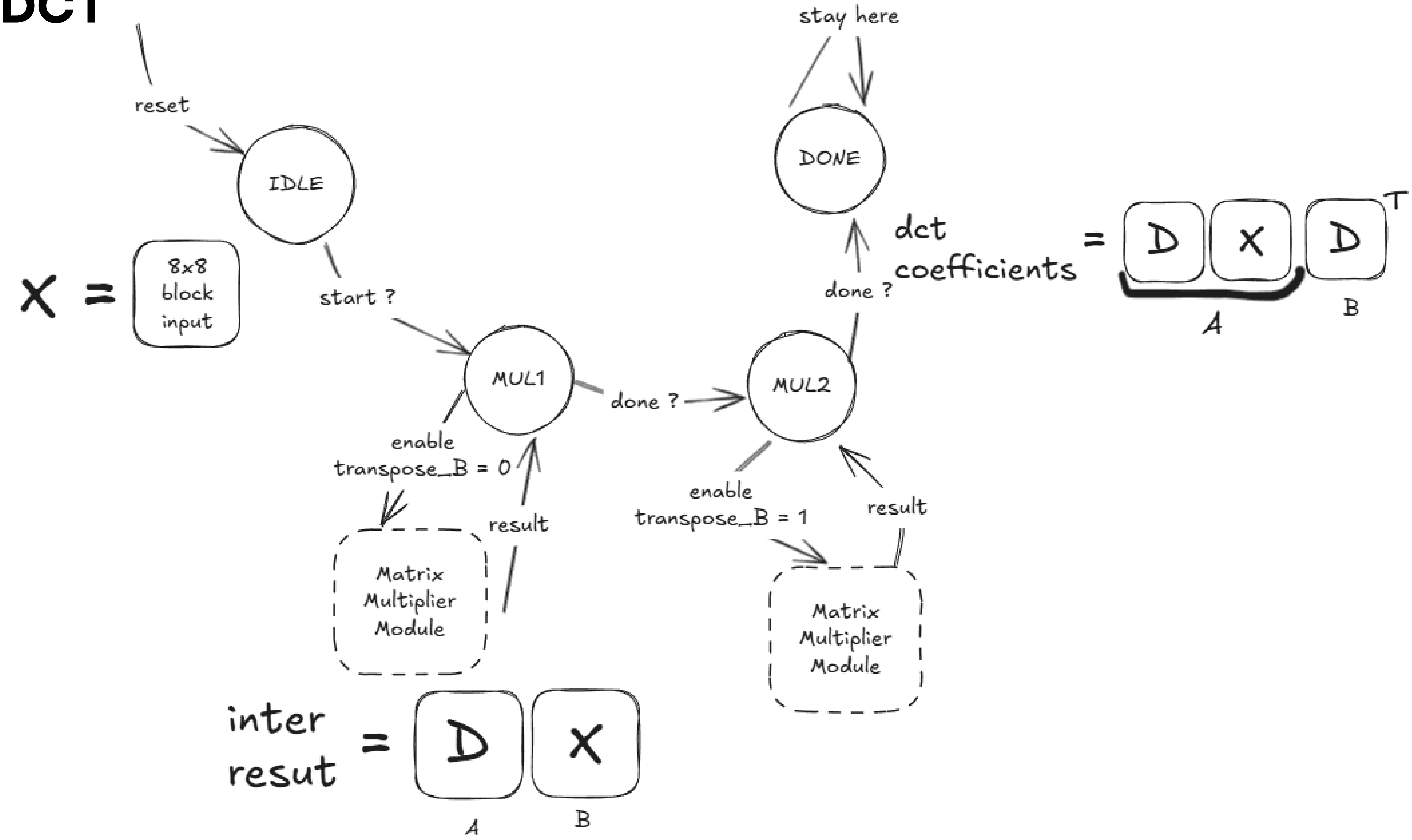
tx_done = 1
TxD =1

DCT Top

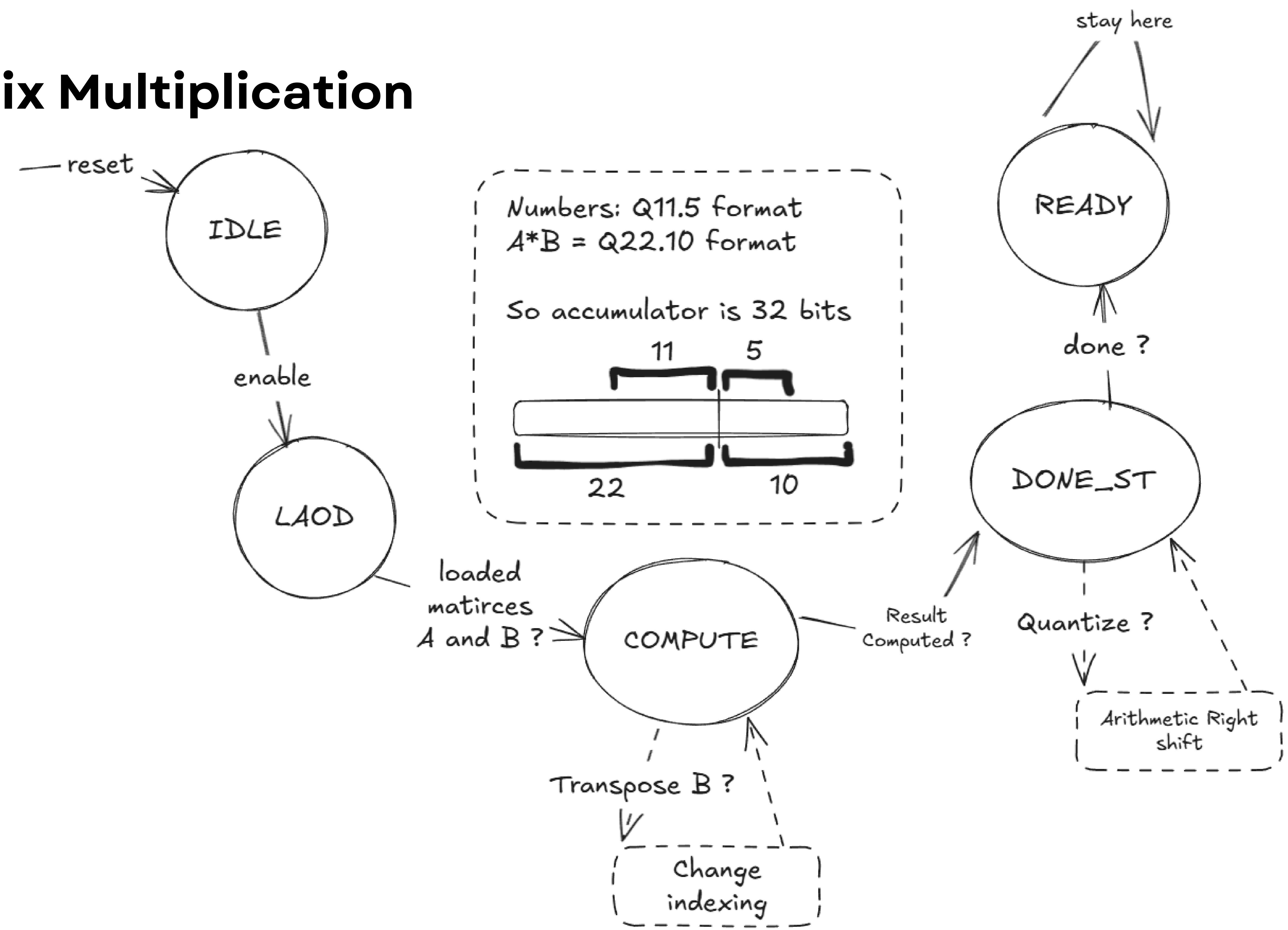


- Reads pixel wise data from BRAM
- Stores the 8x8 block in a register
- Sends it for processing
- Writes the processed output to BRAM 2
- Moves to Next Block

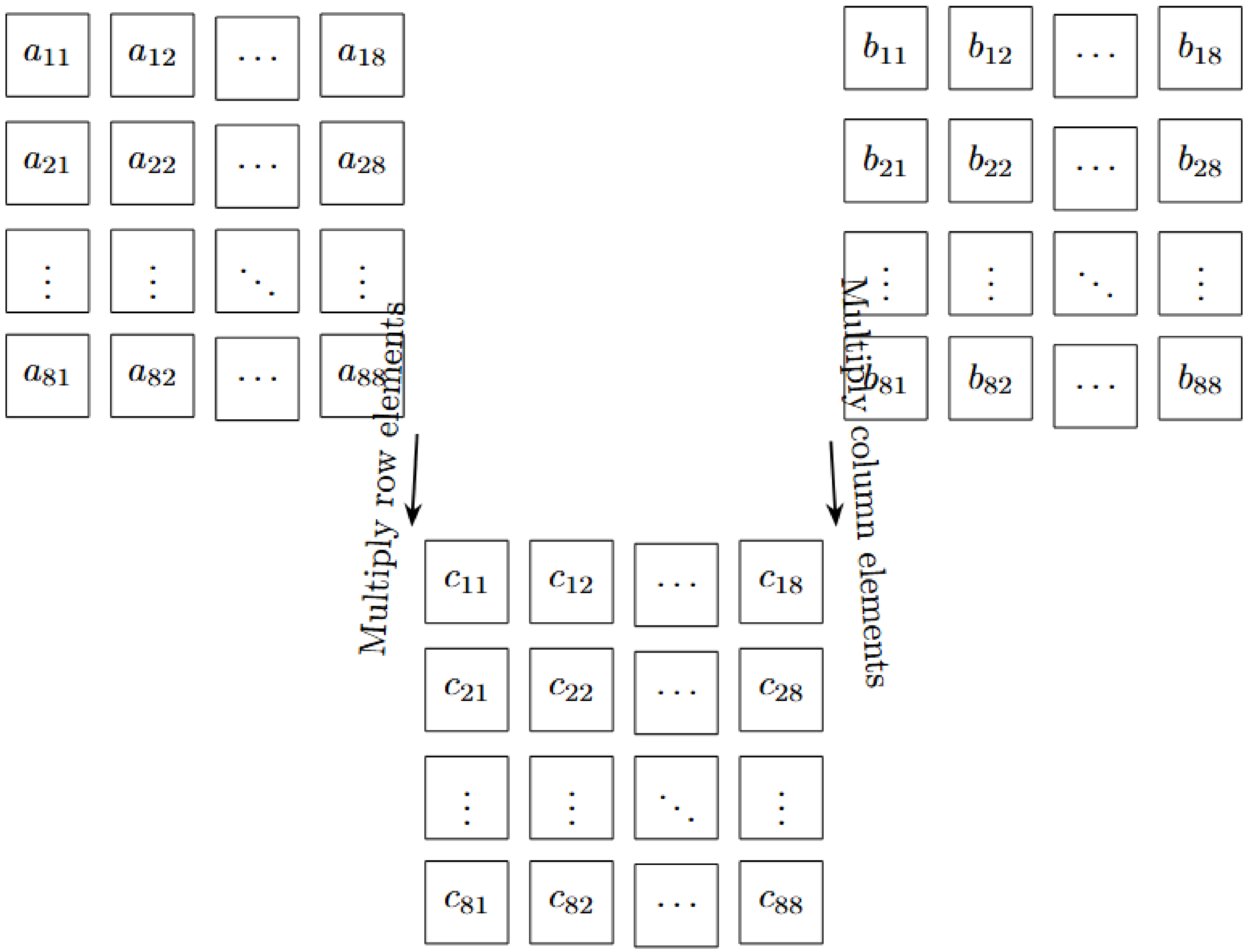
DoDCT


reset
IDLE
X =
8x8
block
input
start ?
MUL1
enable
transpose_B = 0
Matrix
Multiplier
Module
result
inter
resut
=
D
X
A
B
done ?
MUL2
enable
transpose_B = 1
Matrix
Multiplier
Module
result
done ?
DONE
stay here
dct
coefficients
=
D
X
D
T
A
B

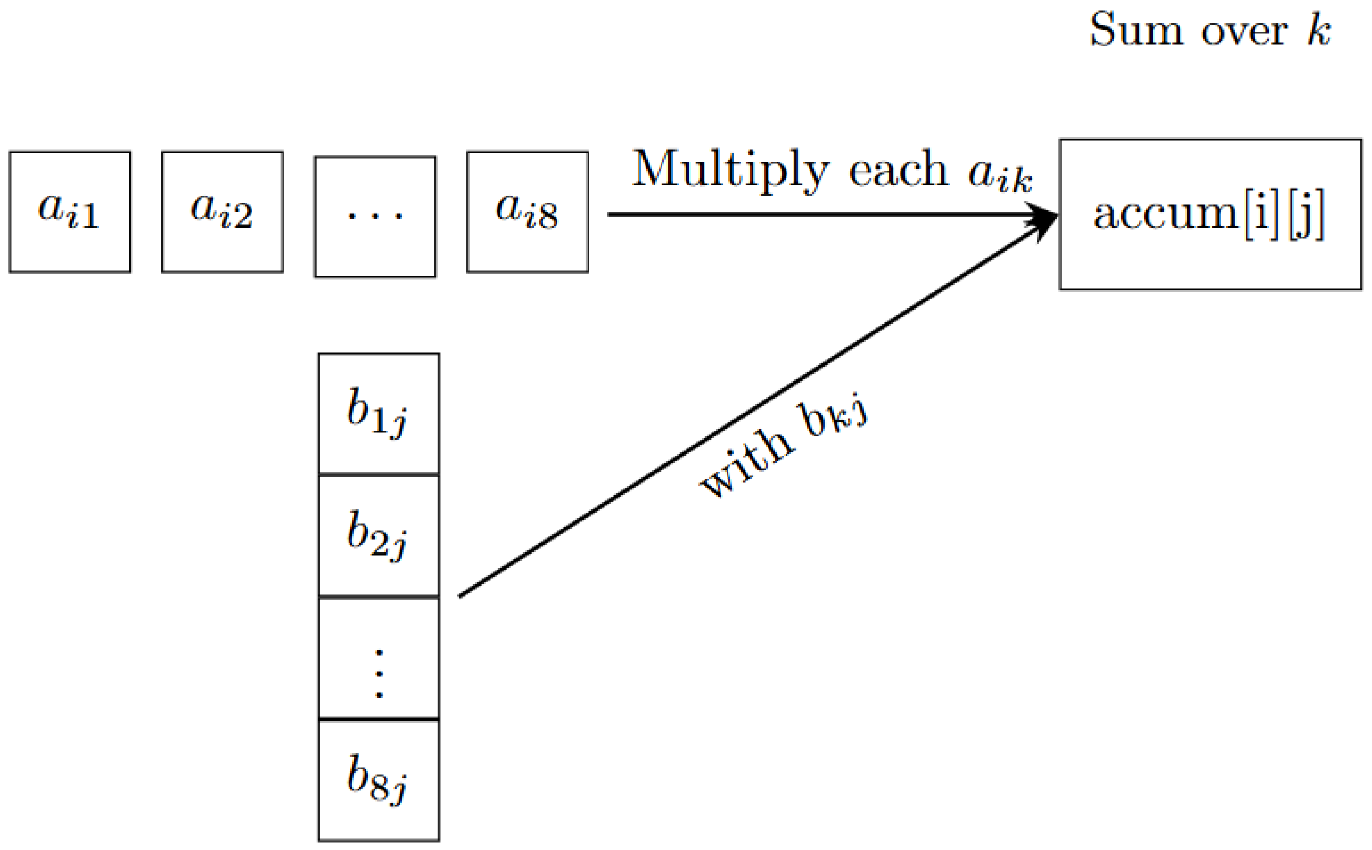
Matrix Multiplication



Matrix Multiplication



Note: Multiplying two Q11.5 numbers yields a Q22.10 result.



Dot products are accumulated over k

We use DSP slices for multiplication
For the entire module slow clock is used

Quantization

Q =

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quantization Matrix

16	8	8	16	32	32	64	64
16	16	16	16	32	64	64	64
16	16	16	32	32	64	64	64
16	16	16	32	64	64	64	64
16	16	32	64	64	128	128	64
32	32	64	64	64	128	128	128
64	64	64	64	128	128	128	128
64	128	128	128	128	128	128	128

Rounded to nearest powers of 2

log base 2

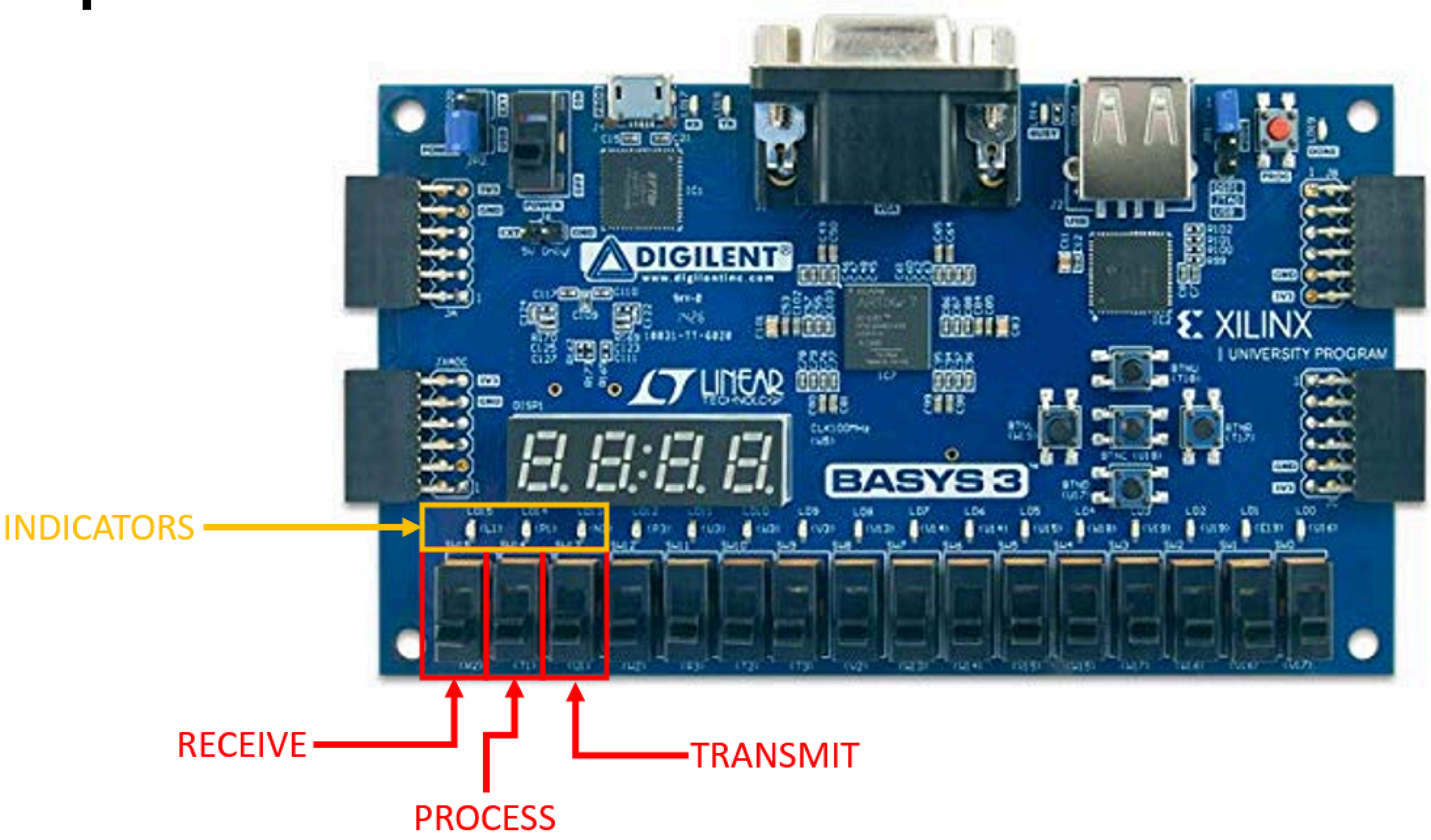
Number of Right Shifts Required

S =

4	3	3	4	5	5	6	6
4	4	4	4	5	6	6	6
4	4	4	5	5	6	6	6
4	4	4	5	6	6	6	6
4	4	5	6	6	7	7	6
5	5	6	6	6	7	7	7
6	6	6	6	7	7	7	7
6	7	7	7	7	7	7	7

D[i][j] = D[i][j] >>> S[i][j]

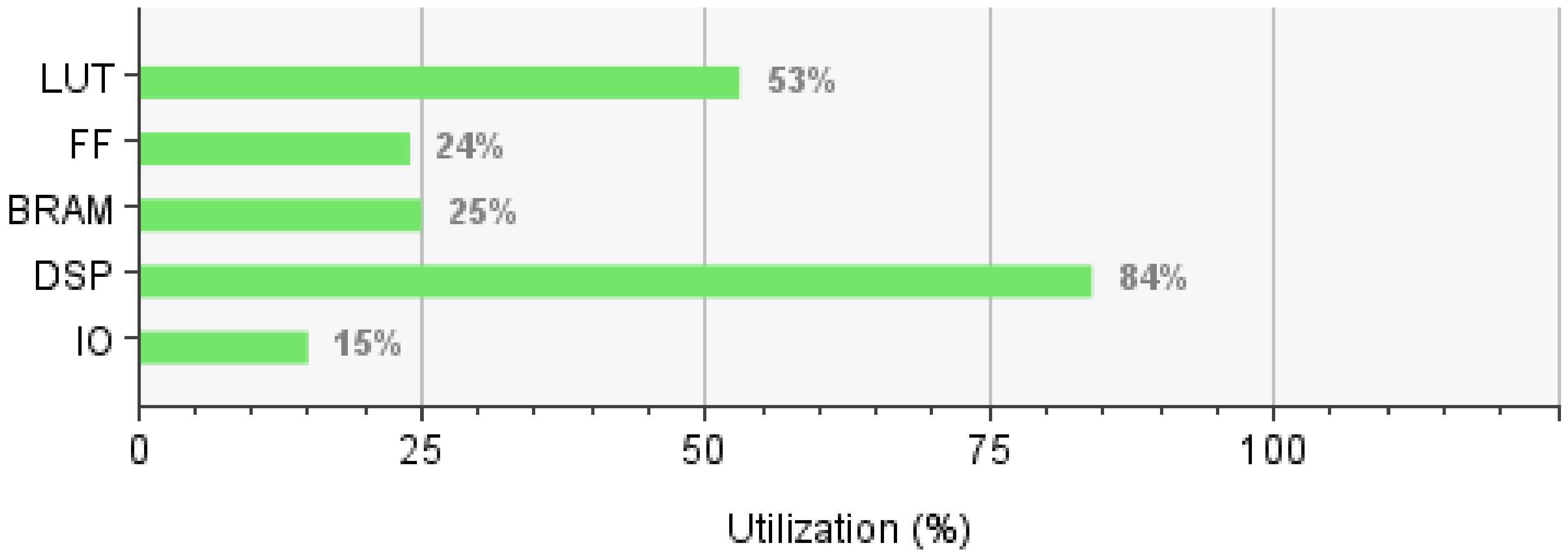
FPGA Implementation:



Hardware Utilization Report

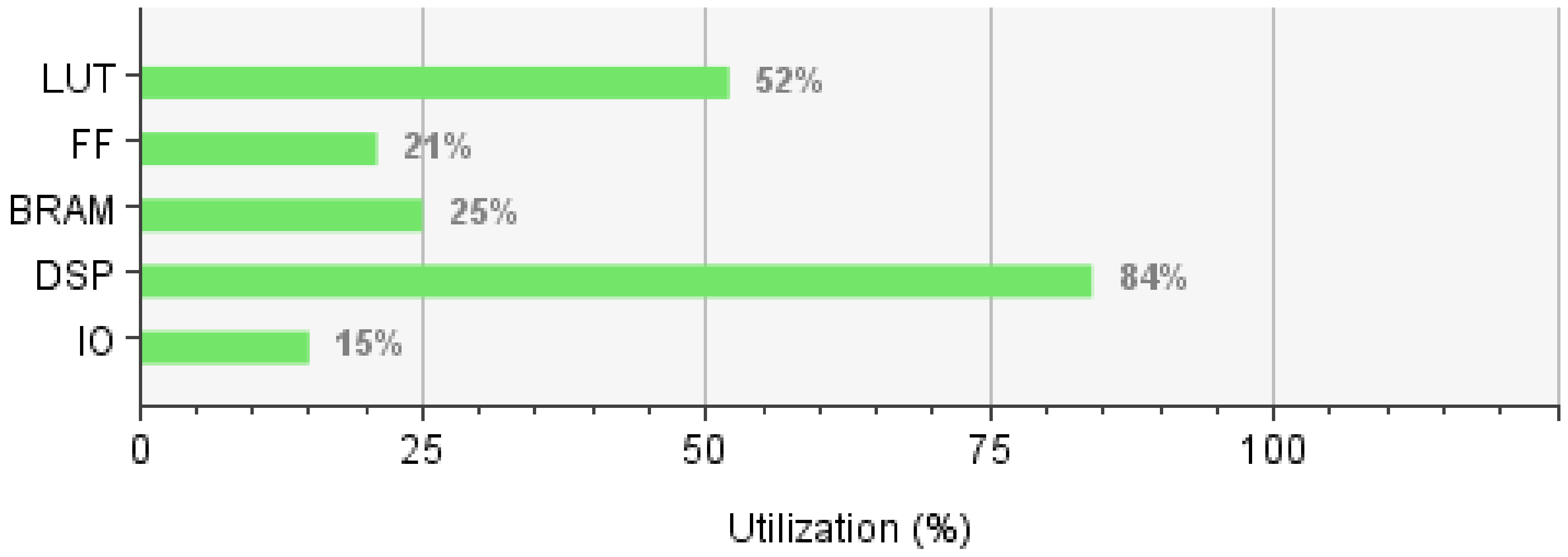
Quantizer Off

Resource	Utilization	Available	Utilization %
LUT	11080	20800	53.27
FF	9849	41600	23.68
BRAM	12.50	50	25.00
DSP	76	90	84.44
IO	16	106	15.09



Quantizer On

Resource	Utilization	Available	Utilization %
LUT	10845	20800	52.14
FF	8790	41600	21.13
BRAM	12.50	50	25.00
DSP	76	90	84.44
IO	16	106	15.09



Demonstration (No quantization)

Input Image

Output Reconstructed Image

PSNR = 40.55 dB

Demonstration (Top kxk) for k = 3, 4

Input Image

k=3

k=4

Output Reconstructed Image

PSNR = 28.9 dB PSNR = 31.8 dB

Demonstration (Powers of 2 quantization table)

Input Image

Output Reconstructed Image

PSNR = 14.6 dB

Compression Ratio = (output bits / input bits)

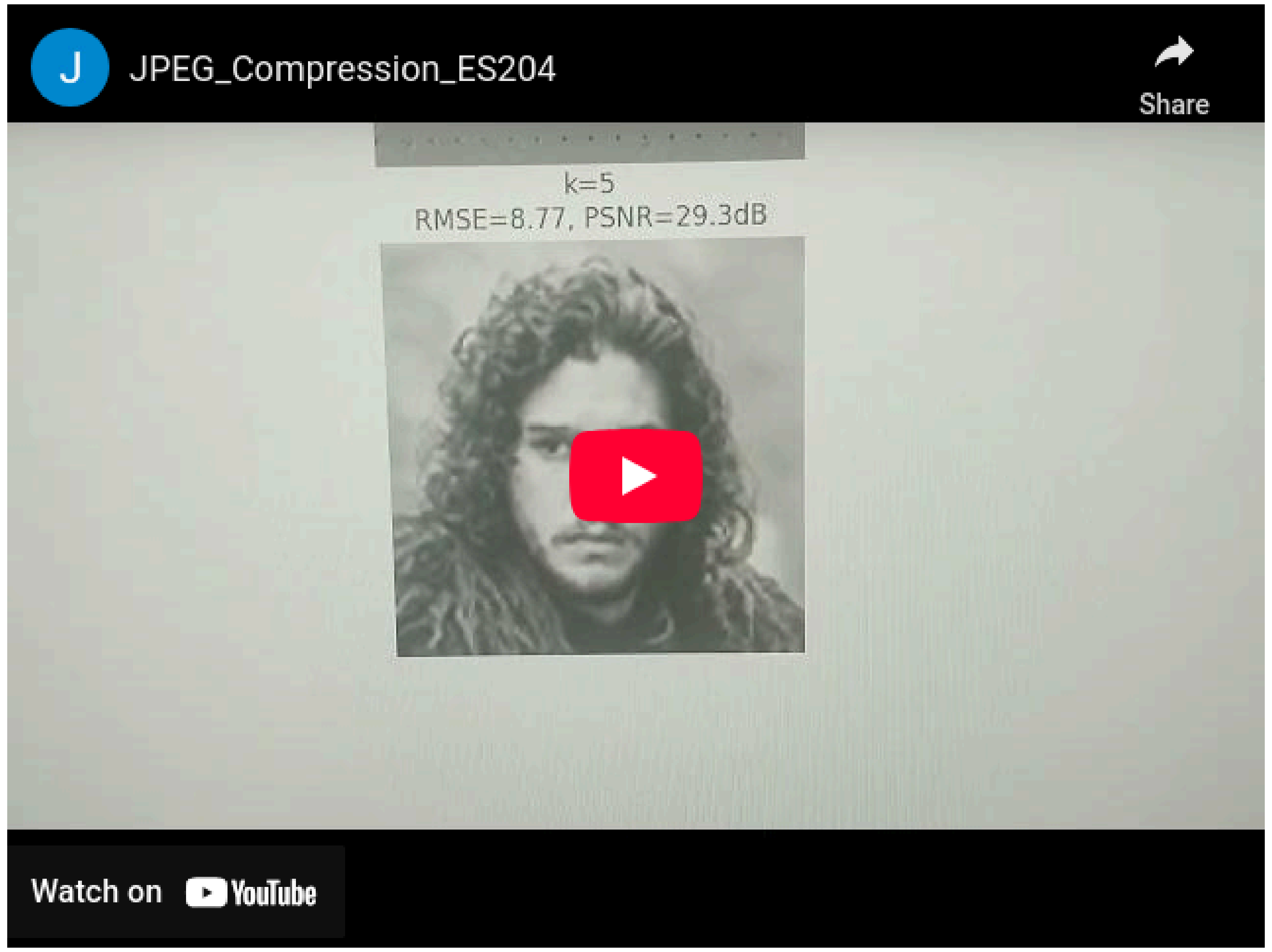
Input: 8-bit 128 x 128 image

No quantization:
Output 16 bit 128x128 coefficients → compression ratio = 2 (not a compression)

Top kxk
Output 16 bit k x k x 256 coefficients → compression ratio = k x k / 32
For k = 4, we have a compression ratio of 0.5

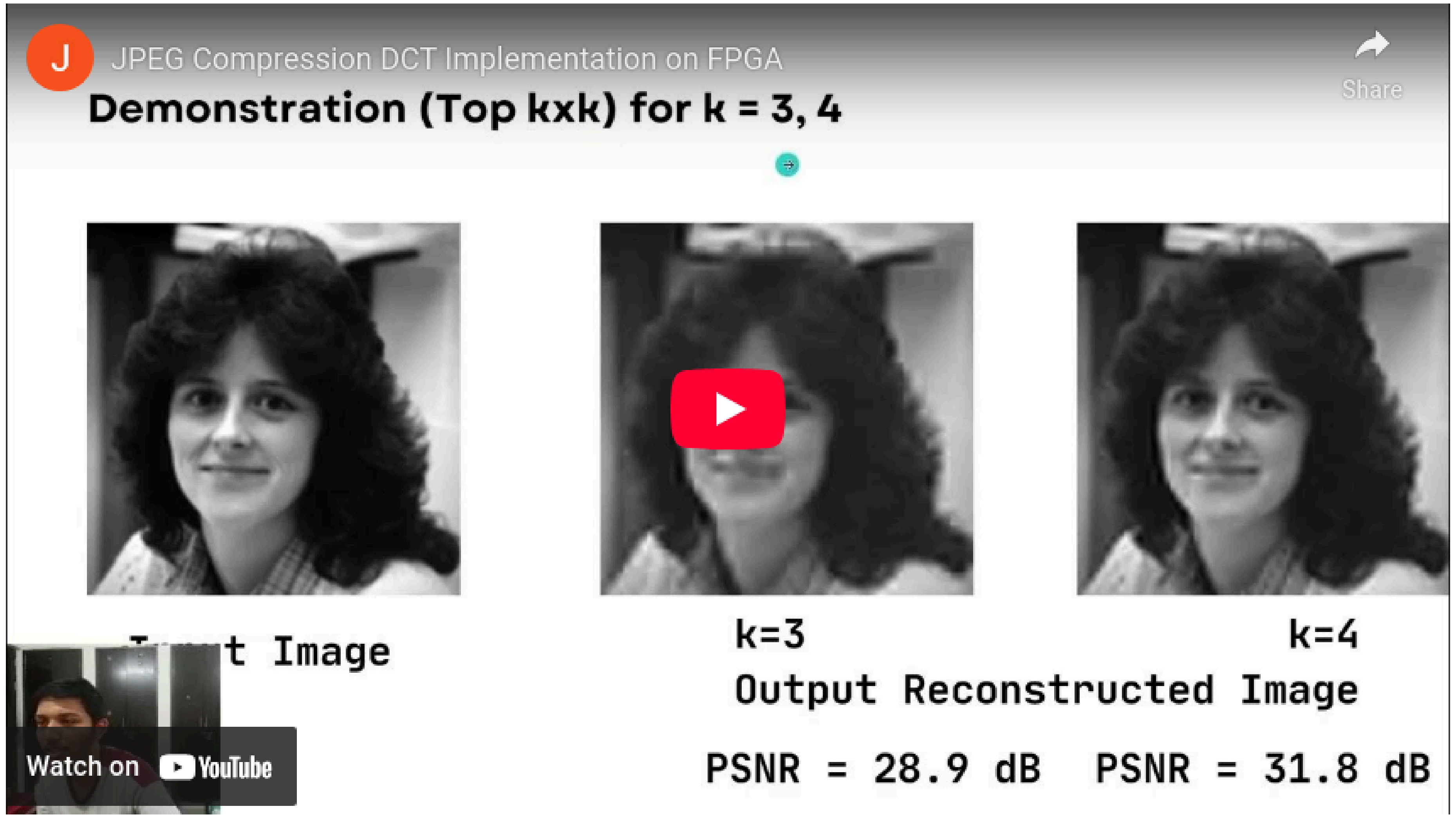
Quantization in powers of 2
Output 16 bits approx 25 x 256 coefficients → compression ratio = 0.78

Demonstration Video



https://youtu.be/vg_AMidRVCo

Presentation Video



https://youtu.be/_Ano39_kA1w

THANK YOU

Discrete Cosine Sabha

Jaskirat Singh Maskeen (23110146)

Nishchay Bhutoria (23110222)

Romit Mohane (23110279)

Soham Gaonkar (23110314)

Indian Institute of Technology Gandhinagar
Palaj, Gujarat - 382355