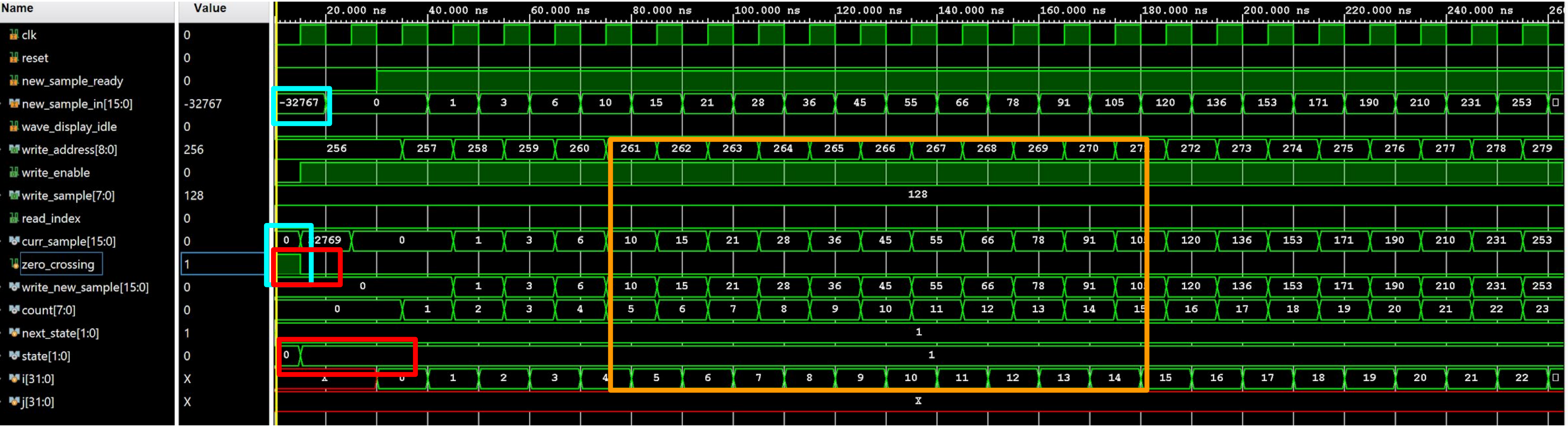
wave_capture



zero_crossing turns HIGH since curr_sample is positive new_sample_in is negative

state moves from STATE_ARM (0) to STATE_ACTIVE (1) due to zero_crossing being HIGH

while write_enable HIGH and in STATE_ACTIVE, count increments and write_new_sample updates

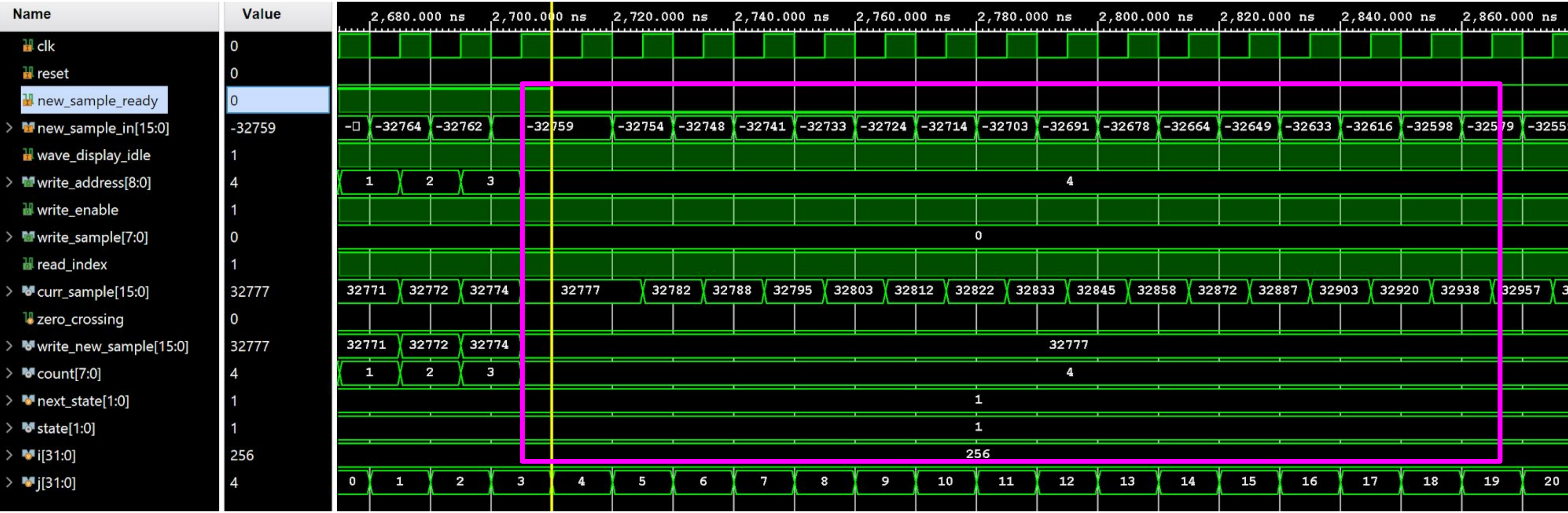
wave_capture

when count reaches 255 it resets to 0, state changes from STATE_ACTIVE to STATE_WAIT and write_enable turns off

when wave_display_idle goes HIGH state moves from STATE_WAIT to STATE_ARM

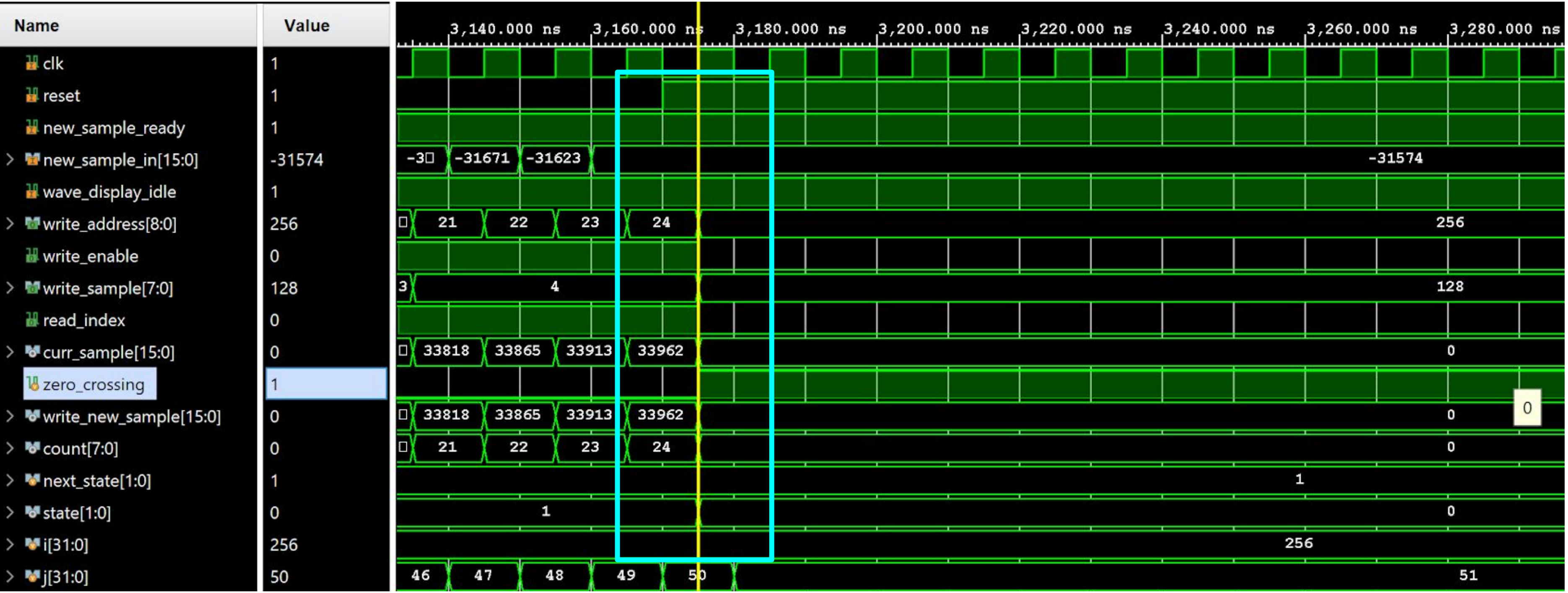
curr_sample and new_sample_in are opposite signs so once again it triggers zero_crossing to go HIGH and this causes state to go from STATE_ARM to STATE_ACTIVE

wave_capture



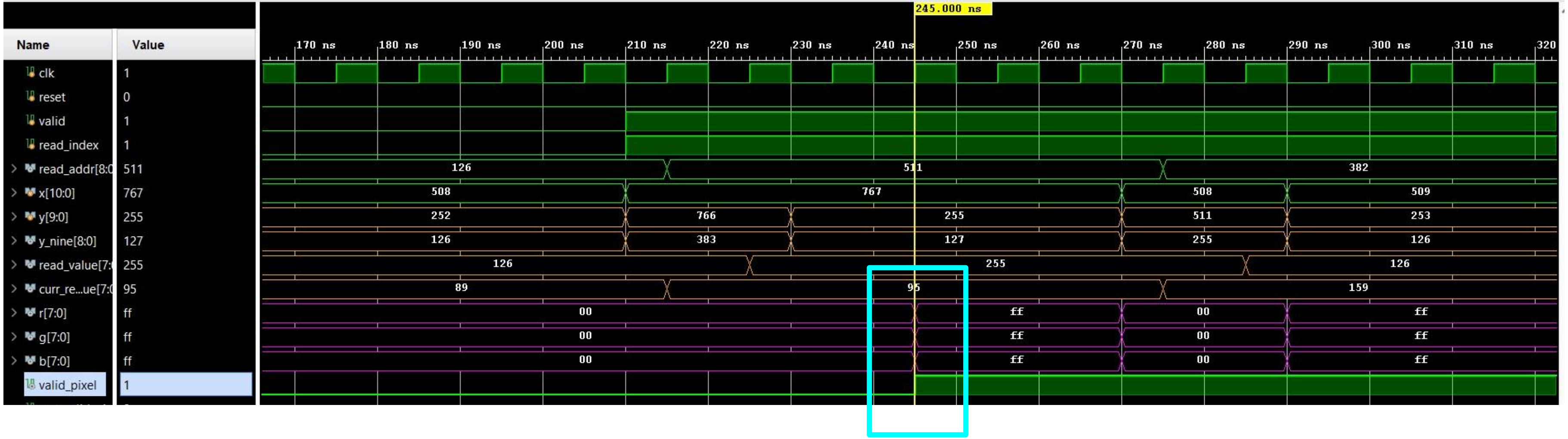
when new_sample_ready is off the counter stops incrementing and the write_new_sample etc stop updating

wave_capture



when reset is triggered, all the flip-flop values are set back at 0 and sets the counting in STATE_ACTIVE back to 0

wave_display

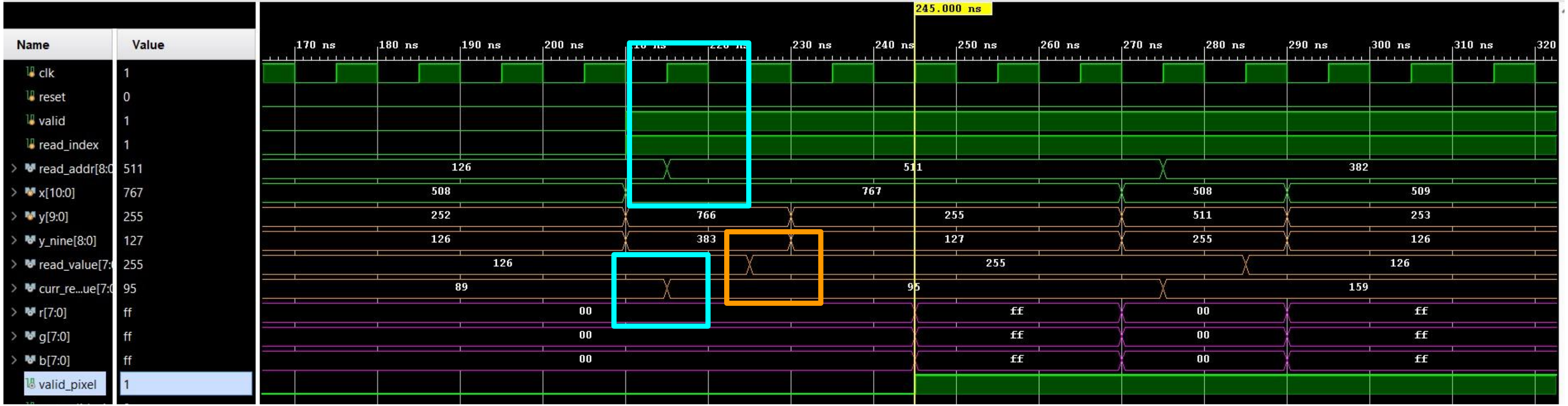


x: 00000000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 00100000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 01000000000, y: 1111111111 = valid_pixel: 0 -> TRUE
x: 00000000000, y: 0000000000 = valid_pixel: 0 -> TRUE
x: 01000000000, y: 0111111111 = valid_pixel: 1 -> TRUE
x: 00100011000, y: 0111111111 = valid_pixel: 1 -> TRUE

Valid_pixel goes high at the same time when rbg pixels equal white as expected.

Text to the right oututs TRUE when the our expected_valid_pixel matches valid_expel

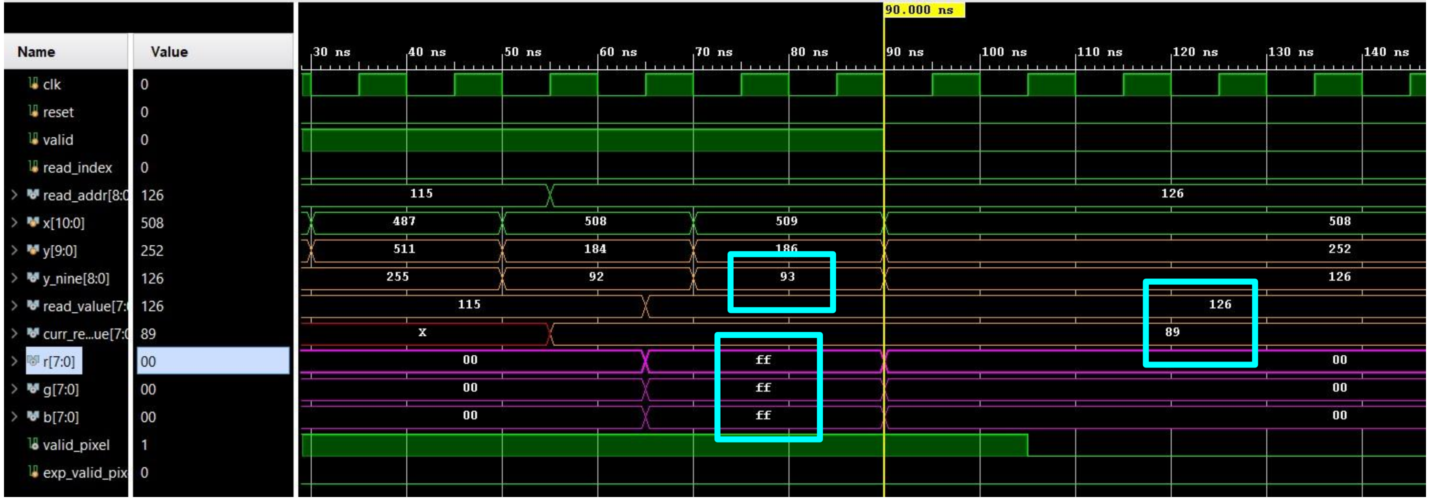
Wave_display continued



When the read_addr changes (in blue) the curr_read_value changes at the same time as expected since read_addr changing is the enable to the flipflop that pushes a new read_value

When read_addr changes, read_value changes one clock cycle later because the RAM takes one clock cycle as expected

Wave_display continued



This case shows when the wave is moving upward and RAM[x-1] < y < RAM[x] so the pixels should be white as expected.

RAM[x-1] = 89

RAM[x] = 126

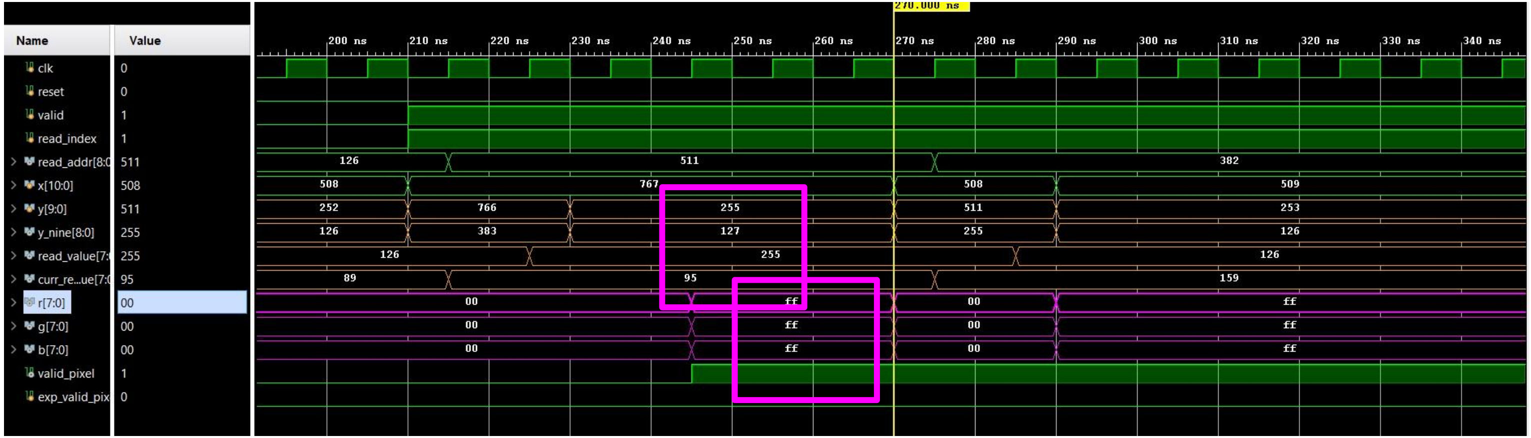
Y = 93

This case shows when wave is moving downward and RAM[x-1] > y > RAM[x] so pixels should be white as expected

RAM[x-1] = 255

RAM[x] = 95

Y = 127



Wave_display continued

x 00111100110, y 0111111110, valid 1, read_address 001110011, read_value 01110011, read_index 0, valid_pixel 1, r 00000000, g 00000000, b 00000000, read_value bounds [01011001, xxxxxxxx]
We can see here that the read_address is being calculated properly - take off the MSB and concatenate {read_index, {x[9], x[7:1]}}. The read value is the last 8 bits of read_address as expected. Even though the pixel is valid as the xquad is 01 and y MSB indicates top half of the screen, the read_address has not changed so the read_value bounds do not have an upper bound as expected.

x 00111100111, y 0111111111, valid 1, read_address 001110011, read_value 01110011, read_index 0, valid_pixel 1, r 00000000, g 00000000, b 00000000, read_value bounds [01011001, xxxxxxxx]
Here, we see that since the read_address is the same as above, the read_value does not get propagated through as expected so the pixels are still black.

x 00111111100, y 0010111000, valid 1, read_address 001111110, read_value 01111110, read_index 0, valid_pixel 1, r 11111111, g 11111111, b 11111111, read_value bounds [01011111, 01011001]
Here, the read_address finally changes, so the read_value propagates through. Since the y-coor falls in between the read_value bounds and the xquad is 01, we have white pixels and the pixel is valid.

x 00111111101, y 0010111010, valid 1, read_address 001111110, read_value 01111110, read_index 0, valid_pixel 1, r 11111111, g 11111111, b 11111111, read_value bounds [01011111, 01011001]
Same case as above, except the y coor changes however it is still within bounds so the pixels are still white.

x 00111111100, y 0011111100, valid 0, read_address 001111110, read_value 01111110, read_index 0, valid_pixel 0, r 00000000, g 00000000, b 00000000, read_value bounds [01011111, 01011001]
Here, we have set valid = 0 so even though the x and y coor are valid, the pixels are black and valid_pixel = 0 as expected.

x 01011111111, y 1011111110, valid 1, read_address 111111111, read_value 11111111, read_index 1, valid_pixel 0, r 00000000, g 00000000, b 00000000, read_value bounds [10011111, 01011111]
Here we set the read_index to be 1 so the read_address should have a 1 in its MSB as expected. However, the y coor is not valid because the MSB is not 0 so pixels are black as expected.

x 01011111111, y 0011111111, valid 1, read_address 111111111, read_value 11111111, read_index 1, valid_pixel 1, r 11111111, g 11111111, b 11111111, read_value bounds [10011111, 01011111]
Here, we adjust the y-coor so it is valid. The read_value bounds indicate that the wave is moving downward. However, since RAM[x-1] > y > RAM[x], is satisfied, the pixels are white as expected.

x 00111111100, y 0011111111, valid 1, read_address 111111111, read_value 11111111, read_index 1, valid_pixel 1, r 11111111, g 11111111, b 11111111, read_value bounds [10011111, 01011111]
Here, the address did not change so the read_value bounds should stay the same as expected

x 00111111101, y 0011111101, valid 1, read_address 101111110, read_value 01111110, read_index 1, valid_pixel 1, r 11111111, g 11111111, b 11111111, read_value bounds [01011111, 10011111]
Here, the y-coor is within the bounds and the wave is moving upward so the pixels should be white as expected. Also, the read_address changed so the read_value bounds changed as expeced.