By- ANIKET GUPTA

ENTRY NO.- 2019CS10327

COL215

ASSIGNMENT 3: IMAGE FILTER

DESIGN OVERVIEW

In this assignment, I have completed a circuit model that filters image stored in a memory. It supports two type of filtering: smoothening and sharpening.

It supports filtering of only QQVGA (120 x 160 pixels). Now since the filtering take place by calculating over square blocks of size 3*3, the final filtered image has size (118 x 158 pixels).

NOTE: I have assumed the pixel values are stored in the memory row-wise, i.e. the first 160 memory locations store the pixel values of the first row of the image, the second 160 locations store that of second row and so on. Another, assumption I have made is that the signal to start or stop the process is controlled by a push button, whereas the type of filtering required (sharpening or smoothening) is controlled by some external signal. The value of this system at the beginning of the process decides whether sharpening or smoothening will take place.

External INPUTS for the overall structure: Since we are not concerned with how the image data and coefficients are stored in the memory, there are only 4 input signals: PUSH, RESET, CLOCK and FILTER_TYPE. RESET signal initialises all the internal signals and outputs to initial value. PUSH signal helps the user to start a process. Pressing the push button while filtering is going on has no effect. Once the filtering is over, the system will wait till the user presses the push button again. CLOCK signal synchronises the process of different components. FILTER_TYPE helps to decide the type of filter that the user requires (smoothening or sharpening).

OUTPUTS for the overall structure: It has three output signals: START, BUSY and DONE. DONE helps the user to know it the filtering is completed. BUSY tells if a filtering process is already going on. START signal let the user to know if he has already pressed push button or not.

part_d COMPONENT: This component controls the working of overall design. When the user presses push button to start filtering, it initialises all the signals. For each pixel of filtered image, it takes 12 clock cycles to complete the calculations and store it at required position. During the first cycle, it updates the row and column index for which the pixel is to be calculated. It stores the row and column signals that helps to know the pixel position for which the calculation is being done. I have used the row and column index of the top left cell in any (3 x 3) block to recognise the block. During the next 9 cycles, it gives addresses to the ram and rom memory components to get the required pixel value and coefficient. Now, the ram and rom gives the data only after the clock edge, so the required data comes only after a cycle when the address was given. part_d component takes this data, convert them into 18 bit data and then send this to MAC. The MAC also performs the calculation only at clock edge, so part_d gives the ‘0’ control signal (initialising sum) after one cycle when it has provided the address of the first cell. So for each pixel position there is a time gap of two cycles when the address was provided and when the calculation corresponding to that address is over. Thus, for each pixel position of filtered image, it takes [1(update row and column index for filtered image) + 9(give the address of the 9 pixels and coefficients required for each pixel of filtered image) + 1(to finally write that pixel value for the filtered image)] = 12 cycles. After every 12 cycles it updates the row and column index for the filtered image. It continues this process until all the required pixels are calculated and stored. After the filtering is over, it waits for the user to press the push button again to finish the process.

NOTE: RAM_64Kx8, ROM_32x9, MAC components were already provided. RAM_64Kx8 stores the initial image and also allows to store the filtered image. It allows to read and write based on the read_enable and write_enable signals. ROM_32x9 stores the coefficients required for filtering and allows to read the data based on the read_enable signal. MAC performs the overall calculation required for each required pixel. It initialises the sum on getting '0' control signal and keeps on adding to previous sum at each clock edge on getting '1' control signal.

Image_filter COMPONENT: This component just creates one instance of each component defined and establish the connections required between the different components.

CALCULATION OF EACH PIXEL VALUE OF FILTERED IMAGE:

For calculating each pixel value of final image, calculation over corresponding 3*3 block in original image must be made. I have used 0-based indexing, thus topmost left pixel is at index (0,0). To calculate the pixel at (ro,co) in the filtered image, we have to consider the pixel in 3*3 square block (in original image) with topmost left pixel at index (ro,co) and the bottommost right pixel at index(ro+2,co+2). The address of a pixel at index (ro,co) in the RAM is 160*ro+co, since the data was stored row-wise. Also during calculation, along with pixel value at index (ro+i,co+j) (where i and j belongs to {0,1,2}) we require coefficient at index (i,j) in the coefficient matrix. We update the address sent to ram and rom accordingly during each cycle. And thus it takes total 9 cycles for calculation of each bit.

MEANING OF IMPORTANT SIGNALS USED

data_from_mac: it is the output coming from MAC after multiplication and additions.

Data_from_ram: it is the data coming from RAM from the address provided.

Data_from_rom: it is the data coming from ROM from the address provided.

Data1_to_mac: data given to the first data input terminal of mac.

Data2_to_mac: data given to the second data input terminal of mac.

Data_to_ram: it is the data given to ram during writing.

Read_s_ram: controls the read_enable input of RAM.

Read_s_rom: controls the read_enable output of ROM.

Write_s_ram: controls the write_enable input to RAM.

Control_mac: input signal to MAC to initialise the calculation.

Address_ram: address value provided to RAM.

Address_rom: address value provided to ROM.