3DQ5 Lab 5 Report

Jack Wawrychuk – 400145293

Minhaj Shah – 400119266

This exercise involved implementing the state machine that displays the image from experiment 1 while account for a series of memory layout changes. Different memory segments were used to store different values for R, G and the odd and even portions of B.

We used different variables to keep track of the different colour address needed which are described as follows: *Red_address* was used for red, *Green_address* was used for Green, and *Blue_address_E* and *Blue_address_O* were used for the even and odd blue locations respectively. Each time the red and green addresses where buffered, they were also incremented. This ensured that for the next time the red or green pixel data was needed, the address was already in the right location. A register called *VGA_red_buff* was used to buffer the lower byte of the red pixel data, as the memory location is incremented before the second half is red. If no additional buffering were to occur, then the data for the second location would be overwritten by the next address's data, e.g. pixel data 1 would be overwritten by pixel data 3.

For the blue pixel data, both *blue_address_E* and *blue_address_O* were incremented only every second cycle, with the first of those cycles buffering the first half of the SRAM data, and the second cycle buffering the second half of the SRAM data, e.g. on the first cycle pixel data 0 and 1 would be buffered, and on the second cycle pixel data 2 and 3 would be buffered. This is implemented using two variables called *blue_delay_E* and *blue_delay_O*, which were both initialized to 0. On the first cycle, the SRAM address is not incremented, and instead the respective *blue_delay* is set to 1. On the next cycle, when *blue_delay* is now 1, the address is incremented and *blue_delay* is reset to 0. This ensures that there are two cycles for each blue address. When the blue pixel data is buffered, if blue delay is 1, then we know that this is our first cycle on this memory address, so the first half of each of the even and odd locations is buffered, e.g. data for pixel 0 and 1. If blue delay is 0, then we know that this is our second cycle on this memory address, so we read the second half, e.g. 2 and 3.

At the beginning of each new row, the green and red address variables need to be corrected to ensure that the starting location for each row is correct. At the end of each row, it was found that the red and green addresses were 2 locations ahead of where they should have been. Since the red address is buffered while the corrections occur, it only needed to be decreased by one, while the green address was decreased by 2. At the end of each row, both *blue_delay* variables are high, meaning that the first half of each of the even and odd blue addresses have been buffered. In the states leading up to the common case, this would cause an extra increment to the address to occur, causing data mismatch. To fix this, both of *blue_delay_O* and *blue_delay_E* were flipped at the time as the corrections to the red and green addresses took place. This ensured no extra increment was made to the blue addresses in the states leading up to the common case.