COMPENG 3DQ5: Digital Systems Design Take-Home Exercise 5
10/28/2021
Thursday - Group 55
William Siddeley 400245905
Mohamed Al-Asfar 400262489

This exercise involved modifying the experiment 1 program when accounting for changes in the memory layout of the RGB data. We firstly made a state table, so that we could track which states are responsible for loading and displaying the data that is read from memory. We then created registers to hold the constant values of the memory locations where the green and blue values are (see appendix). We also created 3 other registers, which were even, SRAM_address_red and SRAM_address_prev (see appendix). Using the aforementioned registers, we changed the lead in (DELAY) states, so that instead of incrementing the SRAM_address variable by 1, we added the offset of the green or blue value we wanted to SRAM_address_prev, so that our lead in states retrieve the first part of the RGB data, (R0R1, G0G2, B0B2, G1G3 and B1B3). Once we come out of the lead states, we then move through the main pixel fetching state loop. We assemble the VGA red green and blue variables from the values we read previously. We use VGA_red_buf to hold R1 as it will get overwritten by R2R3 on the next cycle of states. We also set SRAM_address to read the next green or blue value we need (using the even register to determine if we need an odd or even memory address), as well as increment SRAM_address_red, since it has a different method of memory traversal than green or blue. As the states are cycled through, the VGA red, green and blue values are retrieved and used to create the .ppm file that is required.

In terms of register use for this lab, we estimate that our register usage is high, as we have many registers used to keep track of the addresses as well as pixel data. When factoring all the registers used for address tracking and pixel data, as well as registers used to hold the SRAM read and write data, we estimated our register count to be about 187. This is close to the Quartus estimate of 191 for the top level exercise. The total number according to Quartus is 323, which is broken up between the top level exercise, VGA controller and SRAM controller.

When inspecting the Timing Analyzer to find the critical path, we found that all the slack values are positive. This means that there are no timing violations with our design, and that our design has a margin of time where data is expected before it arrives. Our critical path is a delay of 11.168ns, which is split between 3.076ns of clock delay and 8.092ns of data delay from node SRAM_address[9] to node SRAM_address[7].

Appendix:

Module (Instance)	Register Name	Bits	Description
Top level exercise	even	1	1 bit value that is used to track

			when the value of SRAM_address_red is even (1) or odd (0).
Top level exercise	VGA_red_buf	8	Register used to hold the value of a red channel of 1 pixel. This buffer is needed as VGA_sram_data[2] gets overwritten, so the red value is stored here to fill the next pixel where it is needed.
Top level exercise	SRAM_even_G_offset	18	Register used to hold the constant value 18'h38400, which corresponds to where the even G values begin in memory.
Top level exercise	SRAM_even_B_offset	18	Register used to hold the constant value 18'h57600, which corresponds to where the even B values begin in memory.
Top level exercise	SRAM_odd_G_offset	18	Register used to hold the constant value 18'h76800, which corresponds to where the odd G values begin in memory.
Top level exercise	SRAM_odd_B_offset	18	Register used to hold the constant value 18'h96000, which corresponds to where the odd B values begin in memory.
Top level exercise	SRAM_address_red	18	Register used to hold the current SRAM address where the red pixels are being read from. Since the red segment of the SRAM has a different memory traversal method than the green and blue channels, this register is needed to keep track of which memory address we are on.
Top level exercise	SRAM_address_prev	18	Register used to hold the previous SRAM_address value. Since we rapidly switch SRAM_address from offset to offset, this register is needed to hold the previous value.