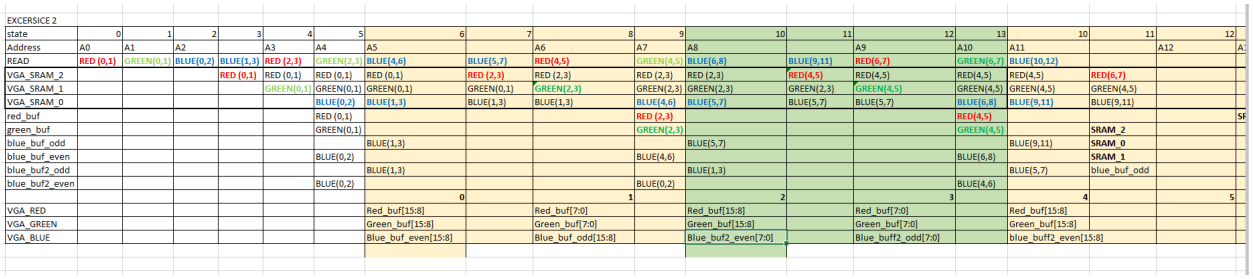
Lab 5
COMPENG 3DQ5

October 25, 2022

Muaz Akhtar (400249273)
Abdelmoniem Hassan (400248003)

In our implementation, we first designed a state table based on an extra read due to the blue values being split into even and odd sections.



EXCERSICE 2																		
state	0	1	2	3	4	5	6	7	8	9	10	11	12	13	10	11	12	
Address	A0	A1	A2		A3	A4	A5		A6	A7	A8		A9	A10	A11		A12	A
READ	RED (0,1)	GREEN(0,1)	BLUE(0,2)	BLUE(1,3)	RED (2,3)	GREEN(2,3)	BLUE(4,6)	BLUE(5,7)	RED(4,5)	GREEN(4,5)	BLUE(6,8)	BLUE(9,11)	RED(6,7)	GREEN(6,7)	BLUE(10,12)			
VGA_SRAM_2				RED (0,1)	RED (0,1)	RED (0,1)	RED (0,1)	RED (2,3)	RED (2,3)	RED (2,3)	RED (2,3)	RED(4,5)	RED(4,5)	RED(4,5)	RED(4,5)	RED(6,7)		
VGA_SRAM_1					GREEN(0,1)	GREEN(0,1)	GREEN(0,1)	GREEN(0,1)	GREEN(2,3)	GREEN(2,3)	GREEN(2,3)	GREEN(2,3)	GREEN(4,5)	GREEN(4,5)	GREEN(4,5)	GREEN(4,5)		
VGA_SRAM_0						BLUE(0,2)	BLUE(1,3)	BLUE(1,3)	BLUE(1,3)	BLUE(4,6)	BLUE(5,7)	BLUE(5,7)	BLUE(5,7)	BLUE(6,8)	BLUE(9,11)	BLUE(9,11)		
red_buf						RED (0,1)				RED (2,3)				RED(4,5)				SF
green_buf						GREEN(0,1)				GREEN(2,3)				GREEN(4,5)		SRAM_2		
blue_buf_odd							BLUE(1,3)				BLUE(5,7)				BLUE(9,11)	SRAM_0		
blue_buf_even						BLUE(0,2)				BLUE(4,6)				BLUE(6,8)		SRAM_1		
blue_buf2_odd							BLUE(1,3)				BLUE(1,3)				BLUE(5,7)	blue_buf_odd		
blue_buf2_even						BLUE(0,2)				BLUE(0,2)				BLUE(4,6)				
							0		1		2		3		4		5	
VGA_RED							Red_buf[15:8]		Red_buf[7:0]		Red_buf[15:8]		Red_buf[7:0]		Red_buf[15:8]			
VGA_GREEN							Green_buf[15:8]		Green_buf[7:0]		Green_buf[15:8]		Green_buf[7:0]		Green_buf[15:8]			
VGA_BLUE							Blue_buf_even[15:8]		Blue_buf_odd[15:8]		Blue_buf2_even[7:0]		Blue_buff2_odd[7:0]		blue_buff2_even[15:8]			

Multiple buffers were used to retain old values as after 3 clock cycles, one of the new Red, Green, Blue even or Blue odd values were being overwritten as they were now available. After writing out a few iterations, we noticed a pattern and implemented our design based on our state table above. We noticed also that for every pair of values for Red and Green that was read from the SRAM, the index was doubled that for blue (since blue is split between even and odd). To counter this, in our design when passing in the address when it was the time to read Blue values (odd or even), we divided the data counter value by 2 (right shift by 1) and added the respective offsets for odd and even starting memory location (BLUE_ODD_START_ADDRESS and BLUE_EVEN_START_ADDRESS). Finally with the help of our buffers, we were able to see a constant pattern for sending the VGA Red, Green and Blue signals and implemented that in our design based on our state table.

After reviewing the compilation report, it was noted that a total of 335 logic registers were used in the design from the report, 203 of which came directly from the experiment file. Only registers that are used with non blocking assignment are counted in the register count within quartus. After manually counting all the registers used in the design (FSM and the combinational logic block we implemented) that were used with non blocking assignment within the code, the number of registers counted was a total of 203, exactly the same as the value from the compilation report. The registers counted to get the value were the following: blue_buffer_even, blue_buffer_even_2, blue_buffer_odd ,blue_buffer_odd_2, green_buffer, red_buffer, first_in_cycle, VGA_red, VGA_green, VGA_blue and VGA_sram_data[2:0].

After viewing the timing analyzer in quartus, it was noted that the critical path was about 8.401 ns based on the worst case timing paths in the timing analyzer. This path originates from the VGA_Controller module from the Vcont node to the data_counter[17] node in the experiment 1 module. Based on our design structure, Vcont and Hcount in the VGA_controller module maps to the pixel_Y_pos and pixel_X_pos registers respectively. In the S_WAIT_NEW_PIXEL_ROW state in our design, it is quite clear that this is where the critical path occurs as there are 3 MUX's (3 if statements) that are checked and the last of which (lowest priority one (worst case one)) is

where data_counter gets reset to 0, which means that the MSB (bit 17) would get reset last and hence why that is the critical path of our design.

Appendix

Module	Register Name	Bits	Description
Experiment 1	data_counter	18	Used to track index of reading for each block of memory (R, G and B)
Experiment 1	data_counter	18	Used to track index of reading for each block of memory (R, G and B)
Experiment 1	VGA_sram_data	16	Buffer register - holds the Red, Green and Blue values read from the SRAM
Experiment 1	Red_buffer	16	Buffer register to hold previous Red value
Experiment 1	Green_buffer	16	Buffer register to hold previous Green value
Experiment 1	Blue_buffer_even	16	Buffer register to store previous Blue even value
Experiment 1	Blue_buffer_even_2	16	Copy of Blue_buffer_even to retain value as it will get overwritten by the next write
Experiment 1	Blue_buffer_odd	16	Buffer register to store previous Blue odd value

Experiment 1	Blue_buffer_odd_2	16	Copy of Blue_buffer_odd to retain value as it will get overwritten by the next write
Experiment 1	first_in_cycle	1	Flag to indicate if this is first fetch 0 cycle based on our state table
Experiment 1	SRAM_address	18	Holds the address to read from the SRAM
Experiment 1	SRAM_read_data	16	Contains the value read from the SRAM (2 Bytes)