3dq5- project report

Group 01: Zihao Zhou (Bill),Xikai Jack Xu

zhouz247@mcamster.ca   xu503@mcmaster.ca   November 25, 2024

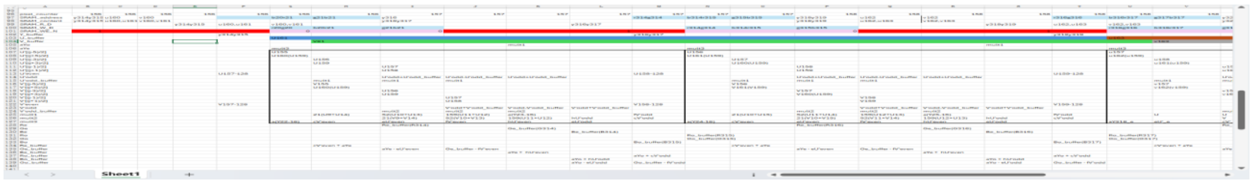
1. Introduction

This project aims to provide us with practical experience in FPGA embedded design by implementing hardware that compact-modifies the mic18 specification. It focuses on receiving compressed image data via a virtual UART interface, decoding it using a programmed circuit, and storing the output in SRAM for display on a VGA monitor. This involves key stages of image compression and decompression, such as color space conversion, signal conversion, quantization, and efficient data storage, all performed by hardware programming methods.

2. Implementation detailed

1) Upsampling and color space convention [Milestone 1]

1. Excel state table



2. The register table

Name of the registers	Size(bit)	Description
odd_even_cycle	1	In an even cycle read 1 Yaddress, Uaddress, Vaddress, odd for 1Yaddr only.
new_row	1	Every 160 coln(320pixel), toggle once for a new row, as re-entrance a new lead-in case.
Y, U, V, RGB_memlocation, Y, UV, RGB_offset	18*(7)	Direct link with the SRAM addressing line, mem_location is the base address and offset is the depth of the address.
U_Xm, U_Xp, V_Xm, V_Xp,	8*(12)	Used for storing the UV data, from the SRAM and used for computation, exchanged data through a certain logic.
Vp_even, Up_even, Vp_odd_buffer, Up_odd_buffer, Vp_odd, Up_odd	32*(6)	Accumulator for the result for the U' and V' result for the next colorspace conversion, for both odd and even V and U.
V_buffer,U_buffer, Y_buffer	8*(2) + 16	Temporary storage of the lower 8bit, ready for odd Y, U, and V, but for consistency, store but Ye and Yo.
aYo, aYe	32*(2)	For fast computing, any RGB calc shard with the same value of aYx, odd, and even pixel.
Re, Ge, Be, Ro, Go, Bo	8*(6)	Storage the after compute RGB value, ready to send to the SRAM (the RGB segment)
Re_buffer, Ge_buffer, Be_buffer, Ro_buffer, Go_buffer, Bo_buffer	32*(6)	Storage for the after-calculated RGB value before clipping.
pixel_counter	8	Use for counting coins, once it reaches 157 s reuse the U V edge value, detect it, and apply its algorithm.

Total register counting 760, Quartus summary 665.

3. Latency analysis

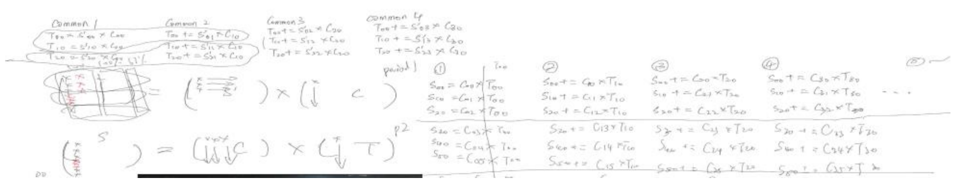
a) Multiplier common case utilization: 16/21 = 76.19%

b) General multiplier case utilization: $[(160 * 240) * 16]/[(11 + 160 * 7) * 240 * 3] = 75.45\%$

c) Latency = [271443] clk(50mhz) = 0.05429s

2) Inverse discrete cosine transforms

1. The memory layout



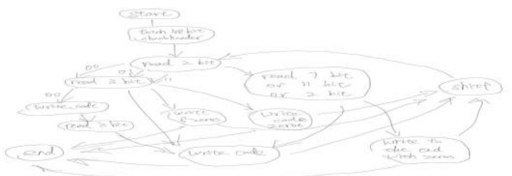
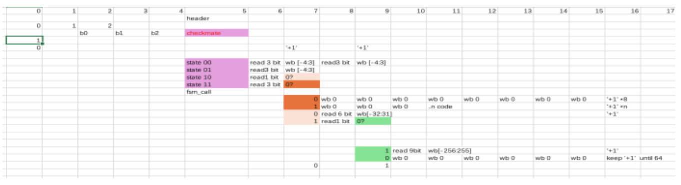
2. The register table Total count reg: 556, Quartus: 820

M2 Top FSM		
Name of the registers	Size(bit)	Description
CA_init_W, RA_init_W, exit_test_counter, Y_finished_W, exit_test_counter, write_S_finish, compute_S_start, write_S_start, CS_flag, FS_flag	10*(2) +20+1+1 +1+1+1+ 1+1	CA&& RA is one of the most complex operations designs inside of our program offset index, for Y and UV are different. But generally, the detect edge is 156@Y and 76@uV since the depth is half and the width the same so CA just adds 4 but RA raises its accumulation speed. Exit_test_counter was supposed to be a tester but now acts like an exit statement. Y_finished is for changing the accumulation speed of an RA, and the starter and flags are just for giving the running signal.
Compute T		
T0, T1, T2, T0_buffer, T2_buffer, T0_result_buffer, T1_result_buffer, Sp_buffer, C_buffer, C_buffer2,	32*(7) + 16*(3)	This Tx register accumulates the result that the T matrix calculation, T0, and T2 buffer is for storing the data from mult3 that must be used over the next clock cycle. Same for the T0 and T1 result buffer, the result will be separator into 2 stages of shifting to the output wire of the CT module only through 64bit total bandwidth, so give 2 regs for preparing the data sending. For the Sp buffer and both C buffers, storage of the least significant data position for the next calculation
T1_buffer_dp_offset, T2_buffer_dp_offset, Address_C_offset, Address_Sp_a_offset, Address_Sp_b_offset,	7*(5)	These are DP ram address offset, controlled by the main FSM, and combined with base address, shifting to the correct location for the containing data for computing S value.
operation_counter; C_counter, C_last_state; is_first_time, even_col, end_col, even_col_counter;	4 + 5 + 2	Is_first_time is used for preventing from writing the garbage data, C_counter is the detector of "6" periodic cases, used for switching the state between 1-5, an extremely complex design. End_col is an end_dector of the last column, ready for neo-row. Even_col_counter is to check for even and odd common cases, Which is used for addressing line shifting.
ComputeS		
//counter registers address_C_counter; row_counter; S0, S1, S2, S1_buffer, S2_buffer. is_first_time, block_counter; test_counter; sp_b_address,address_offset,	3 + 2 + 32* (5) + 1 + 2 + 5 + 7 + 3	Addressing the C counter for Ca and Cb's address, and C was dependent on the most significant bit to make an 8 16 24 order, the row counter is used for shifting 8 for each column to make a 0 8 1 9, etc order. S0 S1 and S2 as the accumulator, or in other words, keep adding the result through three different adders. Is_first_time flag is for preventing the first-time writing from garbage data, and the block counter is to track the last coln which only contains 67% multiplication. Test_counter is for debugging purposes and the sp_reg address is just for addressing from the sp_dpram, to fetch the

s_b_offset,address_s_b_counter;		data. Another very very important point for the row_counter is the selection for CX0, X1, X2 and X3 X4 X5 and X6, and X7 position for the calculation execution order.
Fetch_Extreme_Edition		
counter, first_time;	8 + 1	The counting system for all three addresses uses certain bit changes to allocate the address. First-time flag is to prevent the first-time writing garbage.

3. Milestone 2 latency: Cat image: [962627] clk = 0.019252s

3) Lossless decoding and dequantization

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
					header												
0	1	2															
		b0	b1	b2	checkmate												
1																	
0							'+1'		'+1'								
					state 00	read 3 bit	wb [-4:3]	read3 bit	wb [-4:3]								
					state 01	read3 bit	wb [-4:3]										
					state 10	read1 bit	0?										
					state 11	read 3 bit	0?										
					fsm_call												
							0	wb 0	wb 0	wb 0	wb 0	wb 0	wb 0	wb 0	wb 0	'+1' +8	
							1	wb 0	wb 0	wb 0	..n code					'+1' +n	
							0	read 6 bit	wb[-32:31]							'+1'	
							1	read1 bit	0?								
									1	read 9bit	wb[-256:255]					'+1'	
									0	wb 0	wb 0	wb 0	wb 0	wb 0	wb 0	keep '+1' until 64	
							0		1								

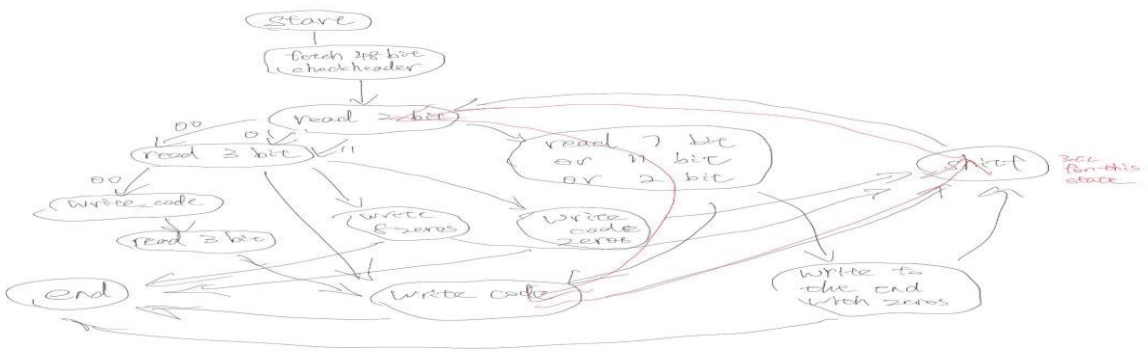
1.

2. The register table for the victory on the performance crown

Name of the registers	Size(bit)	Description
SRAM_new_base_address, SRAM_address_offset SRAM_address_offset	16	This is the container for the previous data location, after finishing one cycle of fetching from the SRAM, the base address will be refreshed and restored.
ZZ_counter	6	This is the Zig_zag address transition register, for just the Dp dram address. (Wired design)
first_choice	2	This is the selection that considers the first 2-bit read.
Q_number, check_header, first_time, DEAD_flag, BEEF_flag	1*(5)	They are the flags, for variant use for header detection (file format detection), Q_precision selection, prevent from input garbage data, and after the first decoding, it no longer enters to the leading statement but direct entrance to the header detection.
code	9	Code is the value before the 32bit signed extension that is ready to be sent
read_buffer	48	48-bit shifting buffer that is used for reloading operation, and bit shifting
read_bits header1, header2, write_counter, code, header_counter, counter;	16*(2) + 6 + 7 +9 +2 + 7	These are basic operators that interact with the logic system, which does all the allocations of the task, the header includes the mic18 head segment, header_counter coun45r4ts the total amount of the header, and the left mount of zeros we need to write.
cmd	3	Used for Q shift

Total counter register: 152 Quartus: 207

3. Worst cases: with 13bit read, 16bit shift, and another 13bit read, as I/O limitation. Read for 2 cycles, shift, write 1



4) Resource usage and critical path

1. Resources usage: The project uses 4,569 logic elements (4% of the Cyclone IV E's capacity) and 2,062 registers, with the M1m1_unit and M2m2_unit consuming the most logic cells. Memory usage is minimal at 11,264 bits (<1%), and only 30 of 532 embedded multipliers (6%) are utilized, reflecting efficient resource usage.

2. For the register optimization, the major of it is m3, by swapping the zigzag mapping to addition arguthium, will save a lot of registers, also for the multiplier could be shared between the module, and also flags swapped by the multifunctional counter to reduce the register usage.
3. The Timing Analyzer shows critical paths between RAM blocks (`RAM_block1a`) and the `M2m2_unit` components (`CT: Compute_T` and `CS: Compute_S`) with data delays ranging from 15.3 ns to 15.8 ns. With a 50 MHz clock (20 ns period), the design meets timing but has limited slack (~4.2 ns). Minor clock skew (-0.4 to -0.5 ns) slightly reduces the margin. The critical paths involve data transfers and computation, suggesting potential optimization opportunities like pipelining or reducing logic depth.

3. Weekly activity and progress

Week	The tasks we have done
0	Jack: reading the experiment documents and learning the separate modules, create the first M1 register table. Bill: learning about debug strategy, especially about the modem techniques, create the first state table for milestone 1.
1	Jack: Implement the FSM for all determinants of milestone 1 and make a data-filled timetable, the lead-in statement, and the lead-out early-age design, making the module independent from the project. Sv. Bill: implementation of the input and output framework, upgrade the timetable, implementation on the register exchange process of the U' and V' reg, making the common case timetable and transit to operation table, debugging for VGA rendering problem.
2-3	Jack: Implement the write back and the Compute S process with operation timetable, make the exchange mux for the module switch and debug with MATLAB and hex editor. Bill: implementation of the Compute T and Fetch data, optimized the Mux system, also figured out the most important RA and CA addressing system, data structure, and so on, with enhanced skill on debugging with homemade input matrix and medium wave. We both debug bit by bit on the addressing line, but Bill carries the lead on seeking bugs :>
4	Jack: debugging milestone 2, debugging the WB process, and swapping the countering system to the flag system. Making the milestone 3 through synchronized design, shifting order, block feed, and so on. Manually gave the input test bench and result, lost the speed competition, and helped debug bills' code. Bill: debugging milestone 2, enhanced the Compute T structure by reducing the S' dram access and reducing its complexity, finding the greatest number of bugs, and finishing integrating the code. Making milestone 3 through register shifting design, block synchronized feed, shifting pointer, and so on, win the competition on the speed. Giving the optimization on the code to match 85% of efficiency, is a great, great achievement. Bill by just observers the communication serial indicator displays to figure out the location of uart shifting offset is not 0, but instead of 76800 which is crazy!

4. Conclusion

In this lab, we learned the problems encountered in FPGA development step by step. We first made extensive use of the wave.do file as a monitor to ensure that the first phase of the debug process went smoothly. By drawing a multithreaded state table, we were able to very smoothly implement milestones 1 and 2 which interact with the multiply dsp and require strict structure, milestone 3 which implements asynchronous SRAM operation by drawing fork tree, and then we learned to split module file design program, This solution simplifies debugging because it can be tested in modules, and then you learned how to use the always_comb structure to limit the number of DSP multiplications used. This uses always_comb's synchronization feature, then we learn to refine the idea of debugging by comparing Modism's wave with other tools such as Matlab and Testbench, and finally by asking questions in the lab. We learned that reg [non_const -: depth] supports floating values, and in general we learned a lot.

5. References: in the reference folder