COE4DS4 – Lab #3 Exercise 1&2&3 Report

Group 22

Mengjia Li (400011362)
Nicole Wu (400089778)
lim14@mcmaster.ca
wuz78@mcmaster.ca

Feb 3rd, 2020

Exercise 1:

The main objective for this elevator exercise is to experiment, practice with pointer and structure usage in developing the processor and FPGA. In order to achieve the requirement, an extra switch interrupt function is created to add the floor request into a buffer and sort them from smallest to the largest. To handle the door open/close and elevator moving state change, an elevator file is created to initiate the counter interrupts from these two state functions and a *FindNext()* function is used to change the direction along with grabbing the desired next destination floor from the buffer. For push button 3, an exception handling is written within *Key3pressed()* interrupt. *reset_counter()* is called in the main while loop every time when checking if the elevator does not reach the destination floor.

Exercise 2:

In this exercise, capital letter cases are added within the switch statement, which is controlled by a *cap_lock* variable. In the interrupt function, we use the *make_code_flag* variable to identify and invert the *cap_lock* signal. This allows it to switch itself upon press and release. In *the read_PS2_data* function, we use a buffer to store the current data by comparing the buffer and the data itself. This way we can find if it is a break or make code, which would allow us to execute the correct *make_code_flag* and control the output.

Exercise 3:

In exercise 3, the random array is generated a char type with 135 elements. In the inner loop of the compacted bubble sorting, *array[i]*, *array[i+1]*, *array[i+2]* are read every loop. *Lead* and *end* represent the number of bits which means the first value consists of "*lead*" bits from *array[i]* and "*end*" bits from *array[i+1]*. At the end of each loop, *i* and *end* is incremented by 1. *Lead* is decreased by 1. When *lead* is equal to 1, the special case occurs. An extra array element *array[i+3]* is read to consist of a value with *array[i+2]*. After the special case, the *lead* is reset to 8 and the *end* is reset to be 1 to start another general case. 4 segments *sec_1* and *sec_2* consists of *var1*. *sec_3* and *sec_4* consists of *var2*. if *var1* is bigger than *var2*, the *var1* and *var2* are separated into 4 segments with a length indicated by *lead* and *end*. Then, they are written into the array elements. For the uncompacted sorting, the array is generated by short int. if the *array[i]* is bigger than the *array[i+1]*. swap function is applied to swap the elements stored in these two addresses.

Compare two types of sorting, in general, the uncompacted sorting takes a shorter time than compacted sorting. This is because there are more steps such as divide the elements into multiple segments, comparison, split the values into segments and write them into the arrays. There is a trade-off between time and space resources usage.

Compare the performances of three processors, the NIOS/e has the longest running time as the NIOS/f takes the shortest time to complete. This is because NIOS/f has more components for processing data at a higher speed.

	3A (uncompacted)	3A (compacted)	3B (uncompacted)	3B (compacted)	3C (uncompacted)	3C (compacted)
1	761359	7710100	219614	2083441	128004	1398662
2	718613	7704275	206286	2082701	121694	1398286
3	766811	7777535	221244	2102358	129481	1412008
4	744191	7818610	214224	2118345	125556	1423635
5	744655	7772595	214368	2100418	126200	1410761
6	748831	7824231	215664	2123774	126675	1426786
7	741233	7789193	213306	2110189	124541	1416709
8	773597	7732521	223350	2089341	130241	1402249
9	742915	7851479	213828	2136447	125126	1434401
10	723427	7722530	207780	2089243	121773	1402601
AVERAGE =	723427	7722530	207780	2089243	121773	1402601