Digital Communication Circuits Laboratory

Lab. 2

Comparator and Sorter

I. Purpose

From this lab., we will learn how to find out the maximum or minimum of a sequence. Besides, we will learn the sorting algorithm to sort a sequence.

II. Principle

A. Comparator

In signal processing algorithm, sometimes we need a comparator to find out the maximum or minimum of two values or of a sequence. The comparison can be done serially or in parallel.

For serial comparison, one comparator is required and the element of the input sequence enters one by one. A D flip-flop is also required to hold the maximum or minimum value from the beginning to the current input. A new input is then compared with the content of the D flip-flop. If the input is larger than or equal to the value stored in the D flip-flop and a maximum of the sequence is desired, then the input will replace the content of the D flip-flop. On the other hand, if a minimum is desired, the content of the D flip-flop will be replaced only when the input is smaller than the old minimum value. An example of serial maximum searcher is shown in Fig. 1. The output of the comparator can be regarded as the sign bit of the difference of inputs A and B. Thus,

$$Comparator\ \ Output = \begin{cases} 1 & A < B \\ 0 & A \geq B \end{cases}. \tag{1}$$

Sometimes, the index corresponding to the maximum or minimum is required. In this case, the index will also be a part of the content to be stored in the D flip-flop.

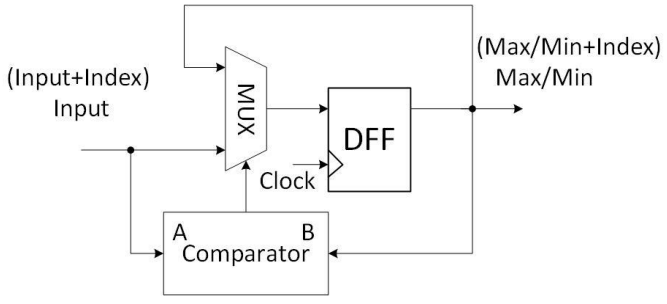


Fig. 1 Serial Comparator

For parallel comparison, usually we will use the comparator tree to find the minimum or the maximum. In this case, multiple inputs can be fed as shown in Fig. 2, where a comparator tree is used to 8 inputs to find out the maximum.

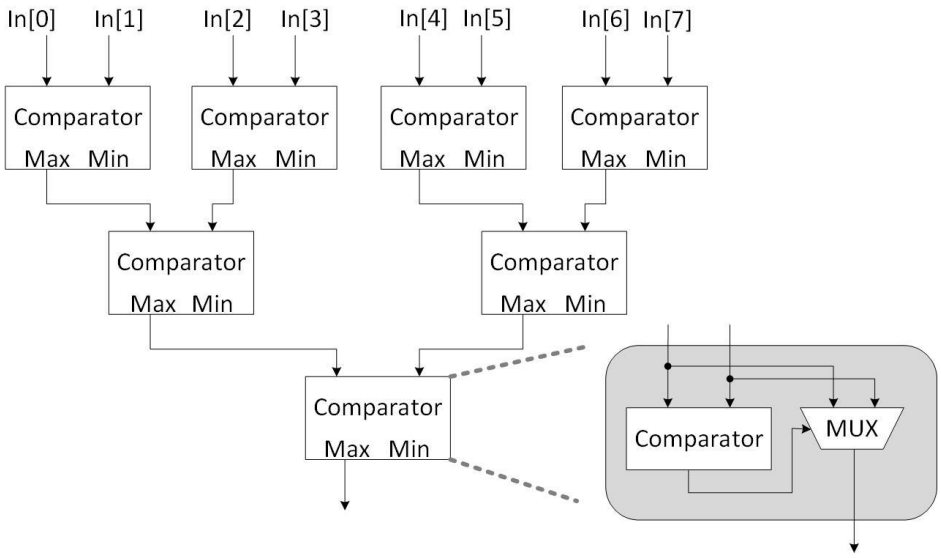


Fig. 2 Comparator tree for parallel comparison

B. Sorter

While the parallel or sequential comparator provides the maximum or minimum value of the whole sequence, the sorter can arrange the elements of a sequence in a descending or ascending order. There are many famous sorting algorithm, such as bubble sort, merge sort or heap sort. Their time complexities are also different. Of course, we would like to implement a sorter with good time complexity. Here, we consider the merge sort algorithm.

The merge sort is a recursive algorithm, which actually contains two phases, split and merge. In the split phase, the input sequence is partitioned into several groups due to the recursive call and the order in each group must be determined. Thus, the split phase is usually continued until one element in one group as shown in Fig. 3. Of course, the size of the smallest group still can be set by the designer.

In the merge phase, the candidates that are the maximum in the respective groups are compared and merged. If the candidate of one group is chosen as the desired one in the current run of the merge phase, that group must generate a new candidate for the comparison in the next run. When all the groups are merged successfully, the input sequence are arranged in the given order and the sorting is completed as shown in Fig. 3.

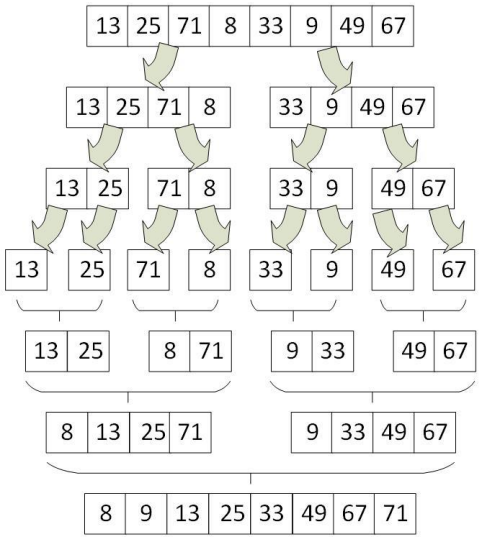


Fig. 3 Merge sort.

The sorting algorithm can be changed to select top M among N inputs. Assume that input sequence is partitioned into G groups. Each group contains N/G inputs arranged in order. When we proceed the candidate selection for M times, the top M elements are determined and the sorting operation is completed. Fig. 4 shows an example with $M = 3,\ N = 8,\ G = 4$. Practically, in the implementation we can use a pointer to point the target element for comparison in each group during the merge phase. When a new candidate is desired, the address of the pointer will be increased by 1 until the end of the group.

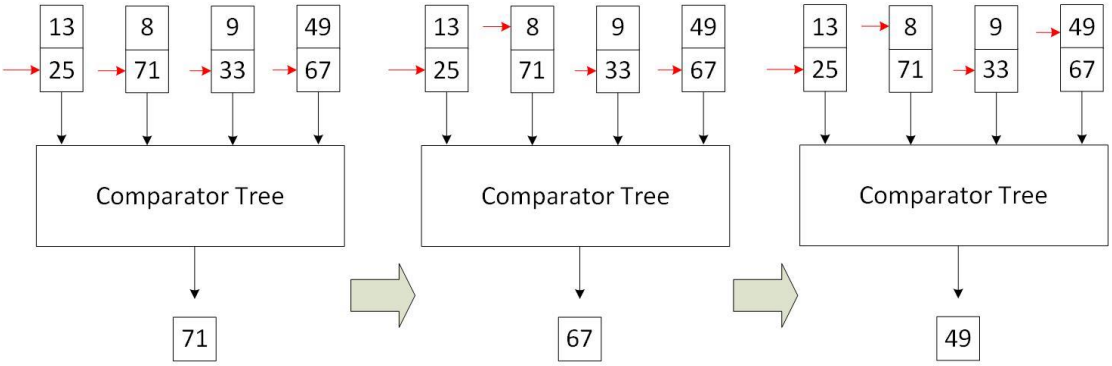


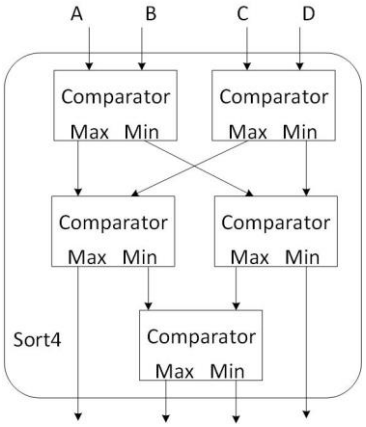
Fig. 4 Selection of top 3 among 8.

III. Procedures

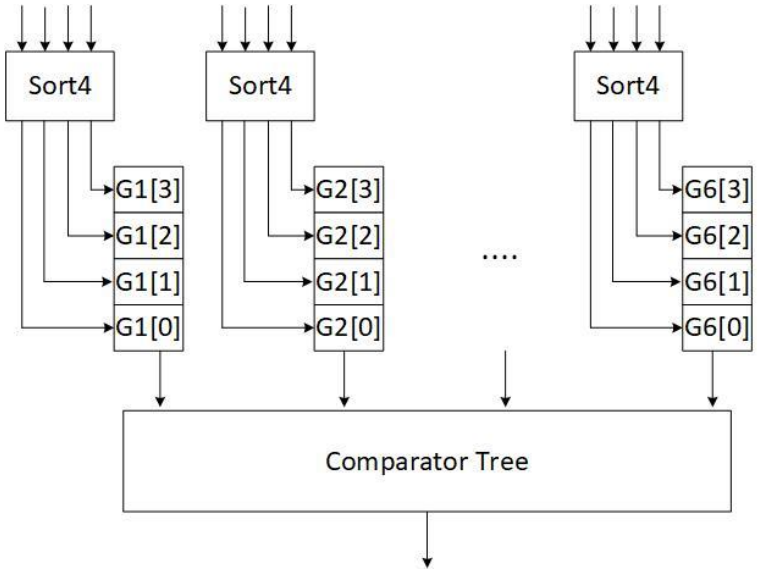
1. For odd-numbered students, please implement maximum searcher and for even-numbered students, please implement minimum searcher by the architecture as given in Fig. 1. Now, generate a random sequence with 24 elements

$[x_1\ x_2\ x_3 \ldots\ x_{24}]$ between -32 and 31. Feed your sequence together with the index counting from 1 to 24 to find out the maximum/minimum of your sequence and the associated index.

2. For odd-numbered students, please implement the parallel comparator for finding the maximum and for even-numbered students, please implement the parallel comparator for finding the minimum. Try to construct a function call/subroutine that generates the maximum or minimum of two inputs as show in the right bottom in Fig. 2. Then, use this function call/subroutine to complete the parallel comparator for 24 arbitrary inputs between -32 and 31. Also the index is desired.
3. Now, use the above sequence with 24 elements $[x_1\ x_2\ x_3 \ldots\ x_{24}]$ between -32 and 31. Given $M = 6,\ N = 24,\ G = 6$, now each group contains 4 elements. Construct the following block (Sort4) to sort four elements in one group. Use $[x_1\ x_2\ x_3\ x_4]$ as the input and check for the correctness of the function "**Sort4**"



4. Construct the following block (**SelectTop8**) to select top 8 among 24 input elements $[x_1\ x_2\ x_3 \ldots\ x_{24}]$. Please use Matlab command "sort" to verify your results. Note that both the values and the indexes are required at the output.



==========

5. Please use Verilog to implement the serial comparator. Now feed the 24 inputs one at each clock cycle. So, it takes at least 24 clock cycles. Observe the outputs. Note that the serial comparator must generate the maximum/minimum value and the associated index.
6. Please use Verilog to implement the parallel comparator. Feed the 24 inputs simultaneously in one clock cycle. Observe the intermediate outputs of every layer and the final outputs including the values and the indexes.
7. Please use Verilog to implement **Sort4**. Feed the 4 inputs $[x_1\ x_2\ x_3\ x_4]$ simultaneously in one clock cycle. Observe the outputs.
8. Please use Verilog to implement the block **SelectTop8**. Feed 24 parallel inputs $[x_1\ x_2\ x_3 \ldots\ x_{24}]$ simultaneously in one clock cycle. Observe the outputs. Note that it takes at least 6 clock cycles to generate the top 6 elements.
9. Synthesis your design in Q6 and Q8. Show the critical path. Note that if your design follows the comparator tree architecture. The critical path will contain the correct number of comparators.

IV. Results

1. Use command "stem" to depict $[x_1\ x_2\ x_3 \ldots\ x_{24}]$. Use marker to indicate the maximum/minimum in the figure. Show your execution results including index and value of your sequential comparator.
2. Show the intermediate outputs (value and index) of every layer and the final output (value and index) in the comparator tree, namely the outputs of all the max/min comparators, given the same inputs as in Q1.
3. Please compare the advantage and disadvantage of serial comparator and parallel comparator.
4. Print out your inputs and outputs of your function "**Sort4**".
5. Print out the 5 outputs of the block **SelectTop8** (value and index) among 24 input elements from your program and the results generated by the Matlab command "sort".

========== ==========

6. Upload your Verilog code the serial comparator. Show the Verilog behavior simulation results and post-route simulation results. (20%) Note that outputs must include values and indexes.
7. Upload your Verilog codes for the parallel comparator. Show the Verilog behavior simulation results and post-route simulation results of every layer. (20%) Note that outputs must include values and indexes.
8. Upload your Verilog codes for **Sort4**. Show the Verilog behavior simulation results

and post-route simulation results. (20%) Note that outputs must include values and indexes.

9. Upload your Verilog codes for the block **SelectTop8**. Show the Verilog behavior simulation results and post-route simulation results. (20%) Note that outputs must include values and indexes.
10. Show your timing report and draw the critical path in the schematic of the block **SelectTop8** from the max delay timing report. (20%)