Object Oriented Modeling of Electronic Circuits – Spring 1402-03
Computer Assignment 4

Selection Sorter Circuit

Amir Hosein Samoudi – 810100108

SSC Design:

In this assignment we are going to design a sorter circuit that will sort 16-bit unsigned integers in a memory and sort them in the ascending order.

The data-path and control-unit that we designed is shown in the figure 1.

Data-Path:

Data-path is consisted of two counters (Counter1 and Counter2). Counter1 is used to address the sorted and unsorted parts and Counter2 is used to address the data which we want to compare it with the current smallest number.

We start the process by a complete pulse on “start” signal. At each stage the last sorted address of the memory is in Counter1 and Counter2 gets incremented in each step and we the memory data corresponding to that address with the smallest value that we founded so far (which is stored at MinReg). If we find a smaller number than the current MinReg value (we perform this by using a comparator) and if it is smaller, we store the new value in the MinRegister and the corresponding address of that data in MinAddrReg. We repeat this until we reach the end of the memory(Counter2 is set to 255) and then we swap the smallest value that we found with the first data of the unsorted part. We repeat this process until the sorted part reach to the end of the file. In that case we issue a complete pulse on the “done” signal.

A wrapper is sat next to the SSC and after initializing the memory it issues a complete pulse on “start” to begin the process.(for simplicity in figure 1 we have shown the Memory beside the data-path)

Controller:

As we can see in figure 1 the controller is consisted of 14 states. The role of each state is explained below:

S0, S1, S2: This states are used for handshaking with the wrapper and make sure we start the process with complete pulse of the start.

S3: Load the Counter2 in this state with the current value of Counter1 plus one.

S4: Load the first data of the unsorted part of the memory.

S5: Load the next data in the unsorted part.

S6: Compare the last value stored in DataReg with the current smallest value (in MinReg)

S7: In case the new loaded data is smaller, we store it in the MinReg as the smallest value so far and also store its address in MinAddrReg.

S9: enable the count of Counter2. Also Check if we have reach the end of the current iteration. If not we go back to S5 and load the next number.

S10 (Update0): After finishing each iteration we should swap the smallest value with the first unsorted value. In this stage we read the first unsorted value from memory and store it in DataReg(so we don't overwrite its value)

S11(Update1): We write the smallest value in the unsorted part as the last number in the sorted part.

S12(Update2): Now we write the value in the DataReg in the address of the smallest value that we found and start a new iteration (states S10, S11, S12 infact swap two memory data)

S13: After completing the process we issue a complete pulse on "start".

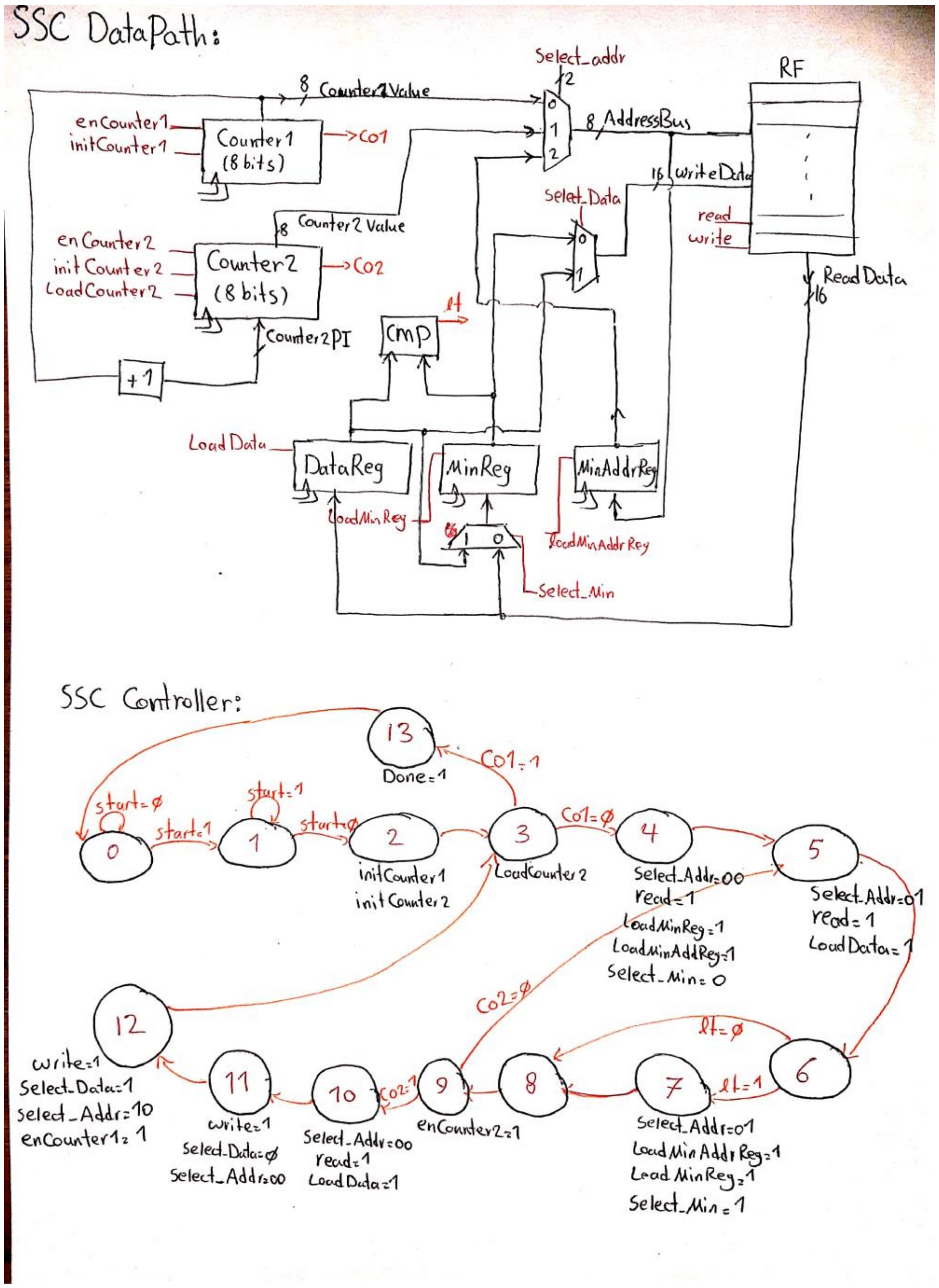
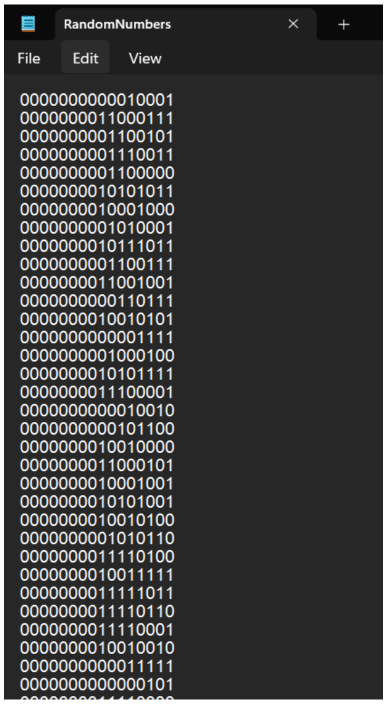


Figure 1- Data-path and Control-unit of SSC

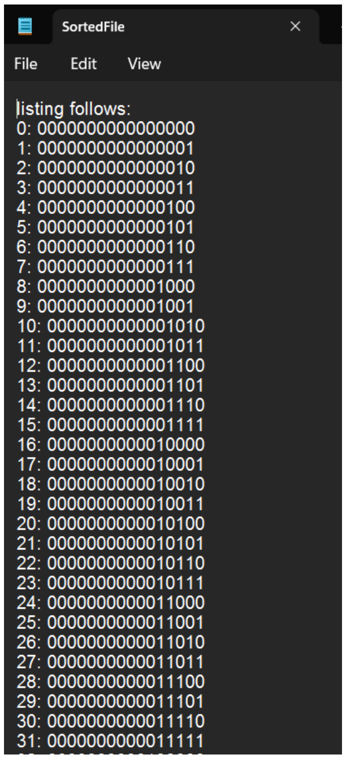
C++ RTL description:

After writing the C++ RTL description of this circuit we test the circuit by generating a random set of numbers (from 0 to 255).



```
0000000000010001
0000000011000111
0000000001100101
0000000001110011
0000000001100000
0000000010101011
0000000010001000
0000000001010001
0000000010111011
0000000001100111
0000000011001001
0000000000110111
0000000010010101
0000000000001111
0000000001000100
0000000010101111
0000000011100001
0000000000010010
0000000000101100
0000000010010000
0000000011000101
0000000010001001
0000000010101001
0000000010010100
0000000001010110
0000000011110100
0000000010011111
0000000011111011
0000000011110110
0000000011110001
0000000010010010
0000000000011111
0000000000000101
```

Figure 2- unsorted file



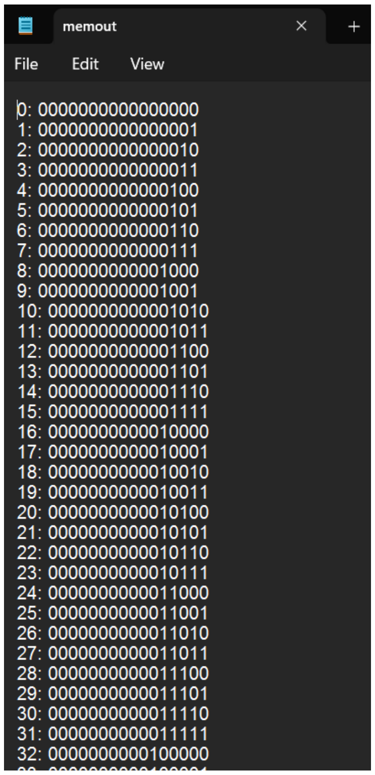
```
listing follows:
0: 0000000000000000
1: 0000000000000001
2: 0000000000000010
3: 0000000000000011
4: 0000000000000100
5: 0000000000000101
6: 0000000000000110
7: 0000000000000111
8: 0000000000001000
9: 0000000000001001
10: 0000000000001010
11: 0000000000001011
12: 0000000000001100
13: 0000000000001101
14: 0000000000001110
15: 0000000000001111
16: 0000000000010000
17: 0000000000010001
18: 0000000000010010
19: 0000000000010011
20: 0000000000010100
21: 0000000000010101
22: 0000000000010110
23: 0000000000010111
24: 0000000000011000
25: 0000000000011001
26: 0000000000011010
27: 0000000000011011
28: 0000000000011100
29: 0000000000011101
30: 0000000000011110
31: 0000000000011111
```

Figure 3- sorted file

As we can see in the figure 3 which is part of the result, it gets sorted correctly.

SystemC RTL description:

We have written the SystemC description of the SSC and tested it for the same set of inputs.



```
0: 0000000000000000
1: 0000000000000001
2: 0000000000000010
3: 0000000000000011
4: 0000000000000100
5: 0000000000000101
6: 0000000000000110
7: 0000000000000111
8: 0000000000001000
9: 0000000000001001
10: 0000000000001010
11: 0000000000001011
12: 0000000000001100
13: 0000000000001101
14: 0000000000001110
15: 0000000000001111
16: 0000000000010000
17: 0000000000010001
18: 0000000000010010
19: 0000000000010011
20: 0000000000010100
21: 0000000000010101
22: 0000000000010110
23: 0000000000010111
24: 0000000000011000
25: 0000000000011001
26: 0000000000011010
27: 0000000000011011
28: 0000000000011100
29: 0000000000011101
30: 0000000000011110
31: 0000000000011111
32: 0000000000100000
```

Figure 4- sorted file(SystemC)

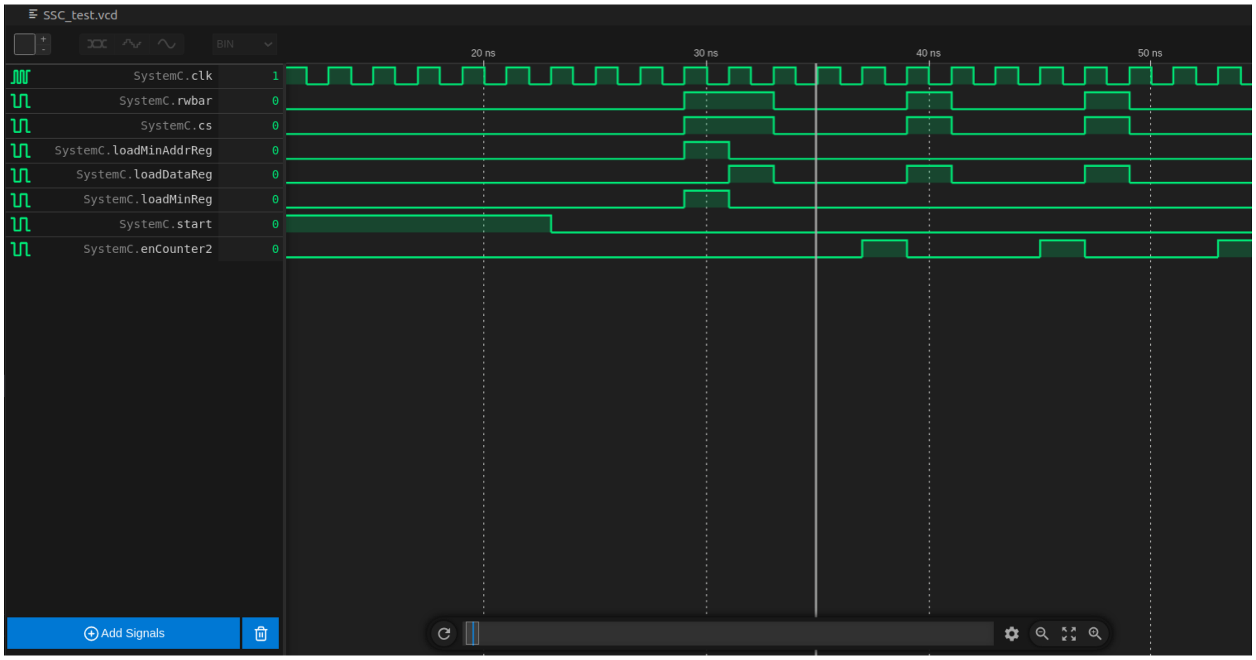


Figure 5- Control Signals in SystemC

In figure 5 we can see some of the control signals that are set in specific states.