CPEN313 project:

Bitonic Sorter

By:

Mark Hanna	A2110650
Jad Zaiter	A2111535
Ibrahim Chami	A2111681

Presented to:

Dr Rafic Ayoubi

A report for CPEN313: Embedded Systems

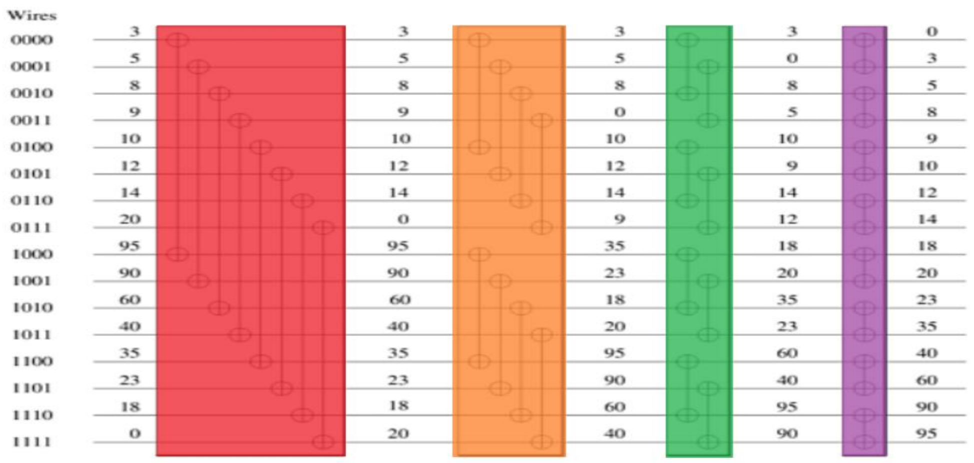
20–05–2023

Table of Contents

Contents

Objective:

Our objective is to design an efficient, low cost, and fast bitonic sorter. This sorter works by dividing the sequence of numbers and comparing them either in ascending or descending order. The time complexity O($\log(n)^2$) must be respected. All of this was achieved using registers, multiplexers, parallel comparators, State machines, counter, and simple gates.



Introduction

Embedded design involves electronic systems that are integrated into a larger product or system to be designed with specific functionality and limitations. The development of the microprocessor in the 1970s, which made it possible to combine CPU, memory, and I/O functions on a single chip, is credited with giving rise to embedded design. The original microprocessor, the Intel 4004, had 2,300 transistors and operated at a frequency of 740 kHz. Since then, cost and power consumption of microprocessors have decreased while performance and complexity have increased. Consumer electronics, automotive, industrial automation, medical equipment, and aerospace are just a few of the applications where embedded systems are used.

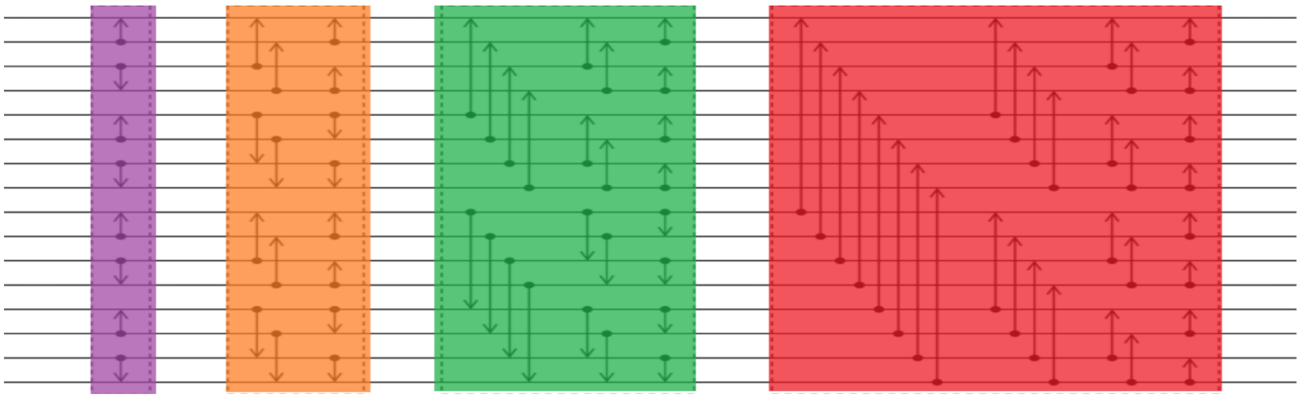
Sorting algorithms are used to arrange data in a specific order. Some commonly used sorting algorithms include Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Merge Sort. These algorithms have different characteristics and performance trade-offs, with Quick Sort and Merge Sort being the most efficient for large datasets.

In this project we will implement a specialized sorting algorithm called bitonic sort. It is utilized in applications for parallel computing and digital signal processing. By splitting an input array into a bitonic sequence and its reverse, it can recursively sort the data in the array. It is effectively implemented in hardware using parallel processors or digital circuits, with a worst-case time complexity of O(log2 n). Applications for bitonic sort can be found in machine learning, computer vision, and cryptography.

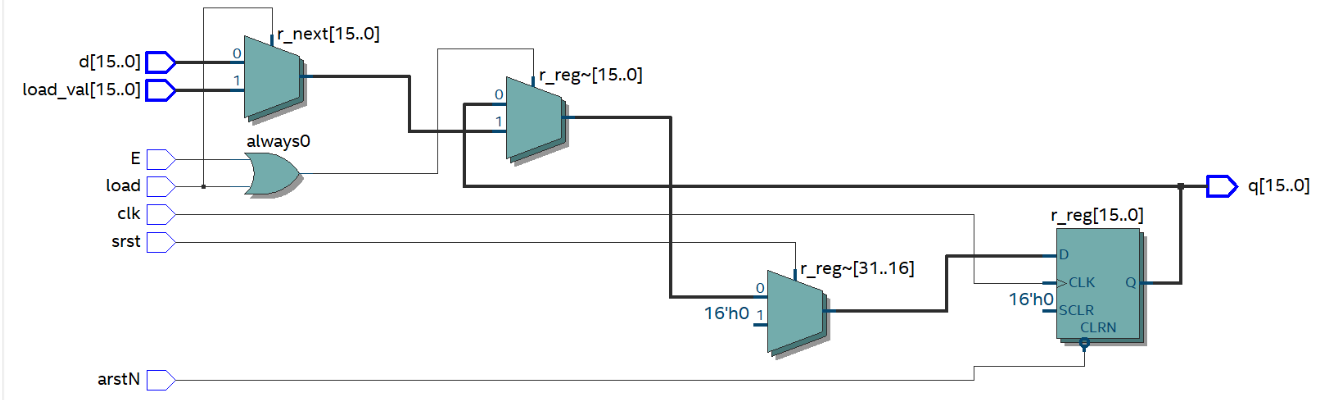
Description Of Project

The datapath:

This part will contain all the components and units designed that are capable of doing the sorting.



1-reg_count_1 and reg_count_16:



The reg_count_1 unit is a parametrized N bit register with an enable line, synchronous and asynchronous reset. It has 2 inputs d and load_val. The load_val is used to store in the beginning a number from sequence then the d input will be used to change the content of the register (when comparing and switching). The output is q.

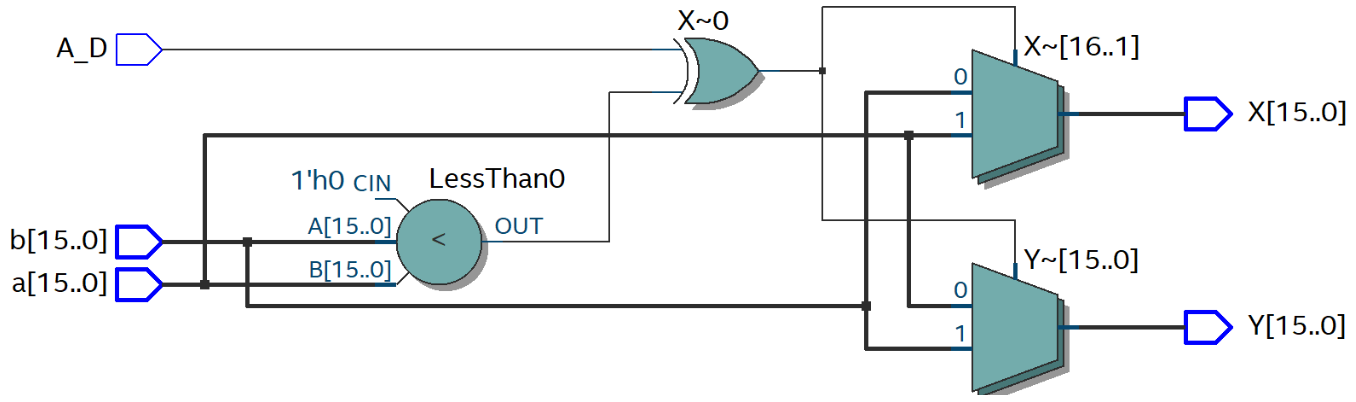
The reg_count_16 is 16 concatenated instances of reg_count_1. Each unit can hold an N bits wide value. They all share the same clock, enable line, synchronous and asynchronous reset. This component will be used to store the sequence of numbers and will contain the final result (sorted sequence).

1-level1:

The purpose of this unit is to route the outputs of the registers towards the 8 comparators. Since there are 4 different offsets we need 2 select lines S[1:0] to control the multiplexers. The route is done according to the table above.

	Step1		Step2		Step3		Step4	
Comparator	A	B	A	B	A	B	A	B
1	q0	q1	q0	q2	q0	q4	q0	q8
2	q2	q3	q1	q3	q1	q5	q1	q9
3	q4	q5	q4	q6	q2	q6	q2	q10
4	q6	q7	q5	q7	q3	q7	q3	q11
5	q8	q9	q8	q10	q8	q12	q4	q12
6	q10	q11	q9	q11	q9	q13	q5	q13
7	q12	q13	q12	q14	q10	q14	q6	q14
8	q14	q15	q13	q15	q11	q15	q7	q15

2-comp_and_switch and comp_and_switch_8:



The comp_and_switch unit has 2 inputs and 2 outputs. The way it works is the comparator compares 2 numbers and then using 2 multiplexers, one for each output, it switches these 2 values. The select line is the output of the comparator XOR A_D. The A_D bit if enabled means we are comparing and switching based on the ascending order otherwise we are comparing and switching in the descending order.

The comp_and_switch_8 is 8 concatenated instances of comp_and_switch. Each unit has a separate A_D bit. This component will be used to compare the values and switch them on its output.

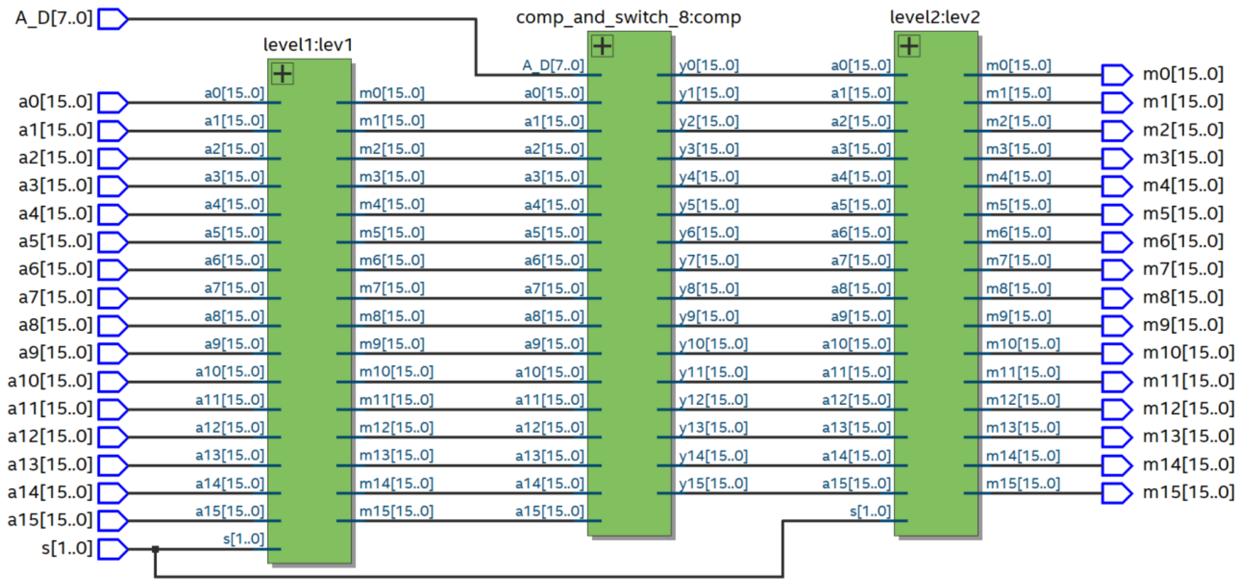
3-level2:

The purpose of this unit is to route the outputs of the 8 comparators towards the registers. Since there are 4 different offsets, we need 2 select lines S[1:0] to control the multiplexers. The route is done according to the table above. (Each comparator unit has 2 output, first unit A0 and A1, second A2 and A3 and so on)

	Input			
Register	Step 1	Step 2	Step 3	Step 4
d0	A0	A0	A0	A0
d1	A1	A2	A2	A2
d2	A2	A1	A4	A4
d3	A3	A3	A6	A6
d4	A4	A4	A1	A8
d5	A5	A6	A3	A10
d6	A6	A5	A5	A12
d7	A7	A7	A7	A14
d8	A8	A8	A8	A1
d9	A9	A10	A10	A3
d10	A10	A9	A12	A5
d11	A11	A11	A14	A7
d12	A12	A12	A9	A9
d13	A13	A14	A11	A11
d14	A14	A13	A13	A13
d15	A15	A15	A15	A15

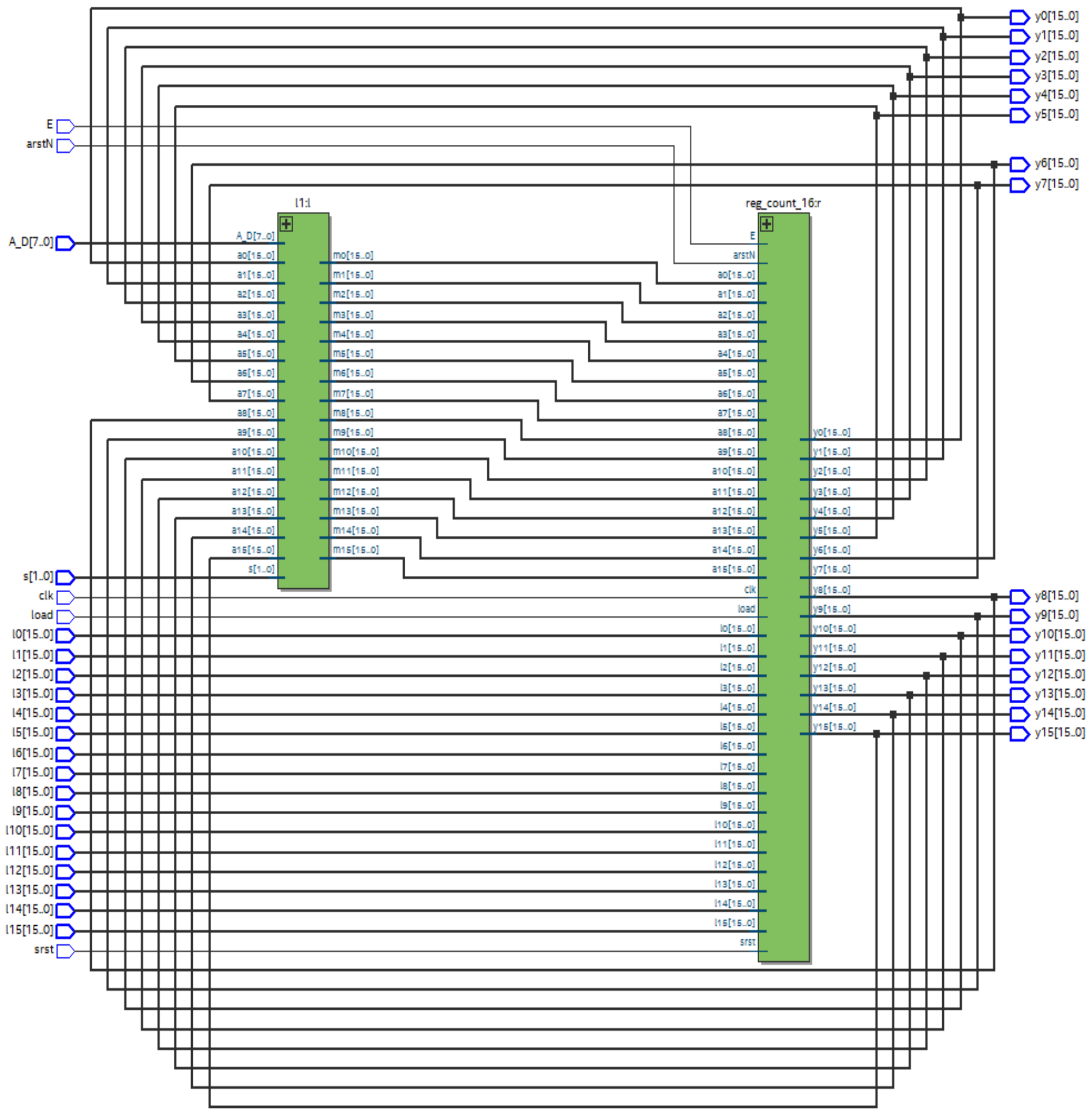
4-l1:

This unit connects the level1, comp_and_switch_8, and level2 units together. This component can be used to test the all the steps and check that the routing is working correctly.



5-reg_com_switch:

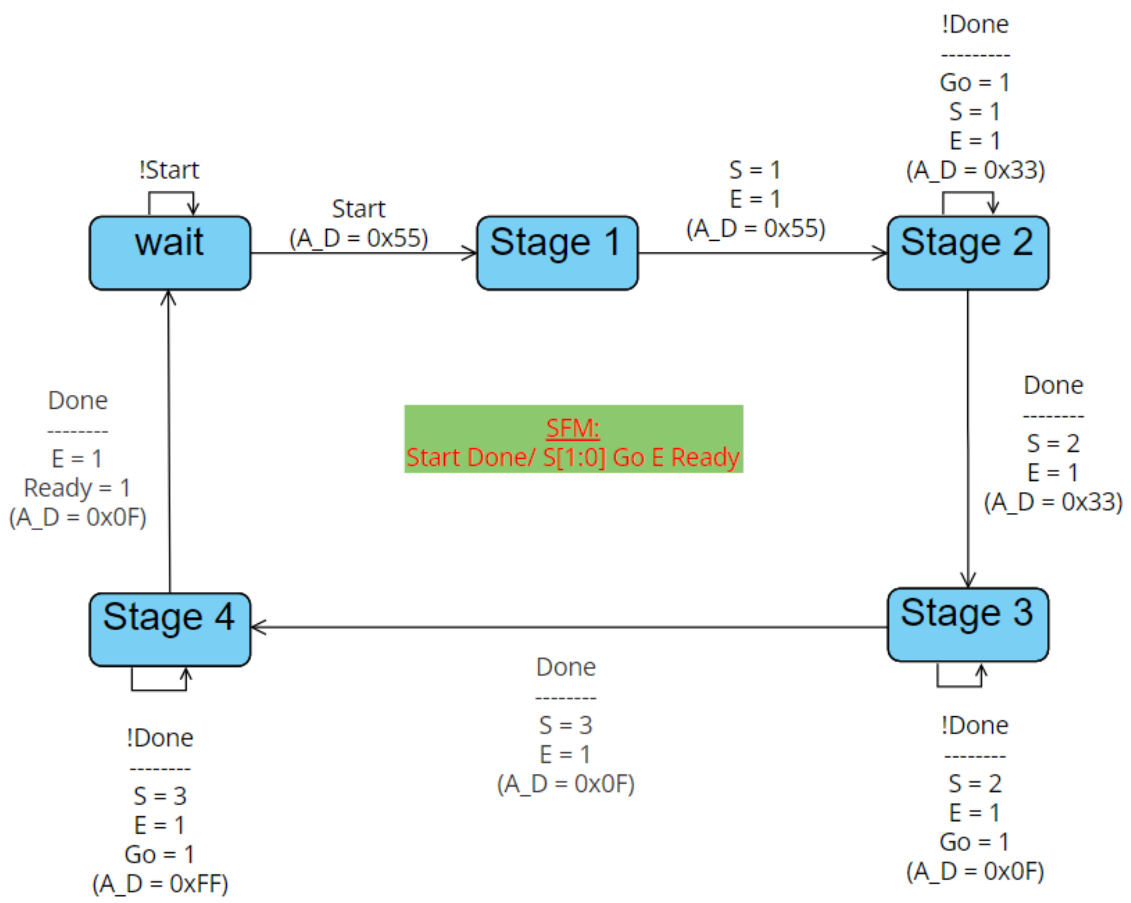
This unit connects the reg_count_16 and l1 units together. This component can be used to test the sorting and make sure that everything is correct. This unit misses a control unit to control the select lines, A_D bits, enable line of registers.



The control unit:

This part will contain all the components and units designed that are controlling the sorting to generate the result.

1-choose_step_SFM:



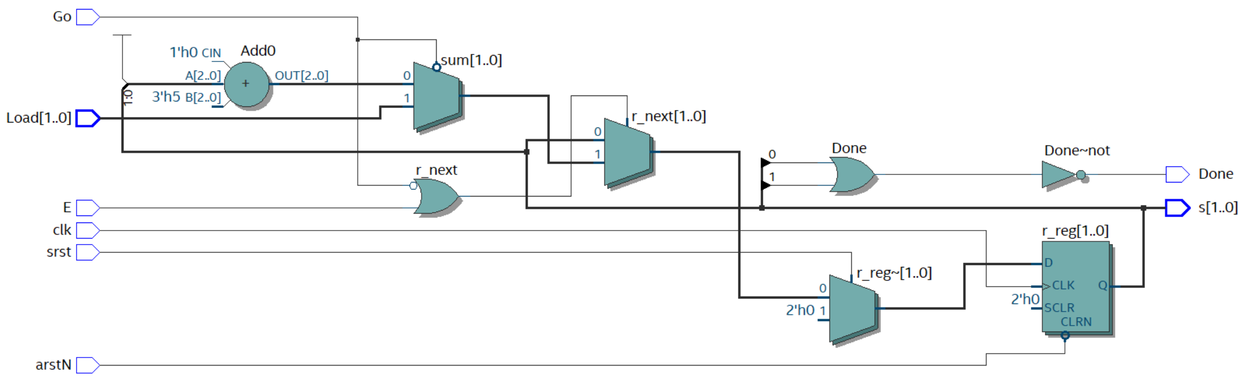
The purpose of this state machine is to control the datapath. This state machine provides the enable line to the registers, the select line S[1:0] (step) to the level1, choose_A_D, down_counter_enable and level2 unit. This state machine will also provide the Go signal to the down_counter_enable. To start the process of sorting the SFM waits for the Start signal and to know if each stage is done a signal Done from the counter is asserted. The state machine does not have an A_D 8 bit output but we are only displaying it so that the user can know the value of it in each state.

2-choose_A_D:

This component sets the A_D enable bit for each comparator in the comp_and_switch_8 unit. It takes as input the select line S[1:0] and as output outputs the A_D bits for each stage.

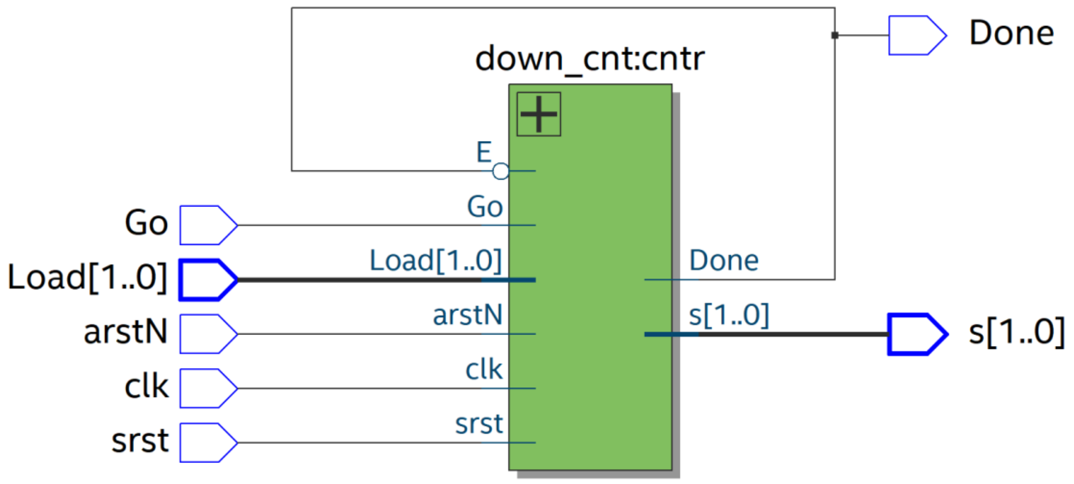
Offset	1	2	4	8	A:Ascending order
Direction of comparison	ADADADAD	AADDAADD	AAAADDDD	AAAAAAAA	D:Desceding order

3-down_cnt:



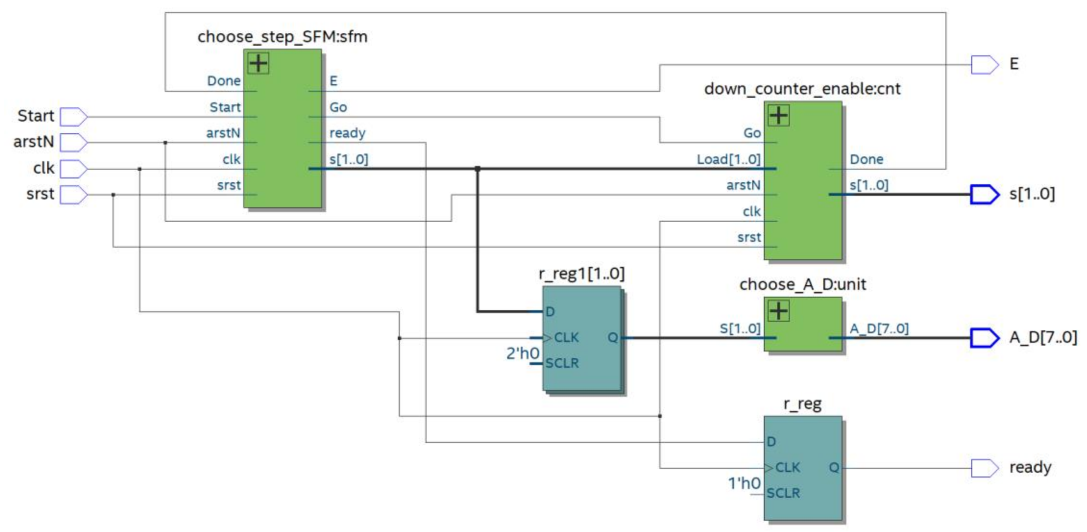
This component is controlled by the “choose_step_SFM”. Initially the FSM loads the desired lvl(S[1:0]) into the counter which will indicate the level we are currently in, using the “GO” signal. If the “GO” signal is low the value is loaded into the counter, then the “GO” signal transitions to high indicating the beginning of the operation. The output of the down_cnt is connected to the multiplexers above the comparators which will indicate what combination will enter the 2 inputs of the comparator. The down_cnt also outputs a done signal fed back to the “choose_step_SFM” indicating the end of all of the steps in the specified lvl.

4-down_counter_enable:



This component is an abstraction of the down counter. This component prioritizes loading a value over it enable state. The “Done” signal is inverted then fed back to the enable input so that the counter stops when 0b00 is reached and doesn’t roll back to 0b11 (Bubble is the same as an inverter).

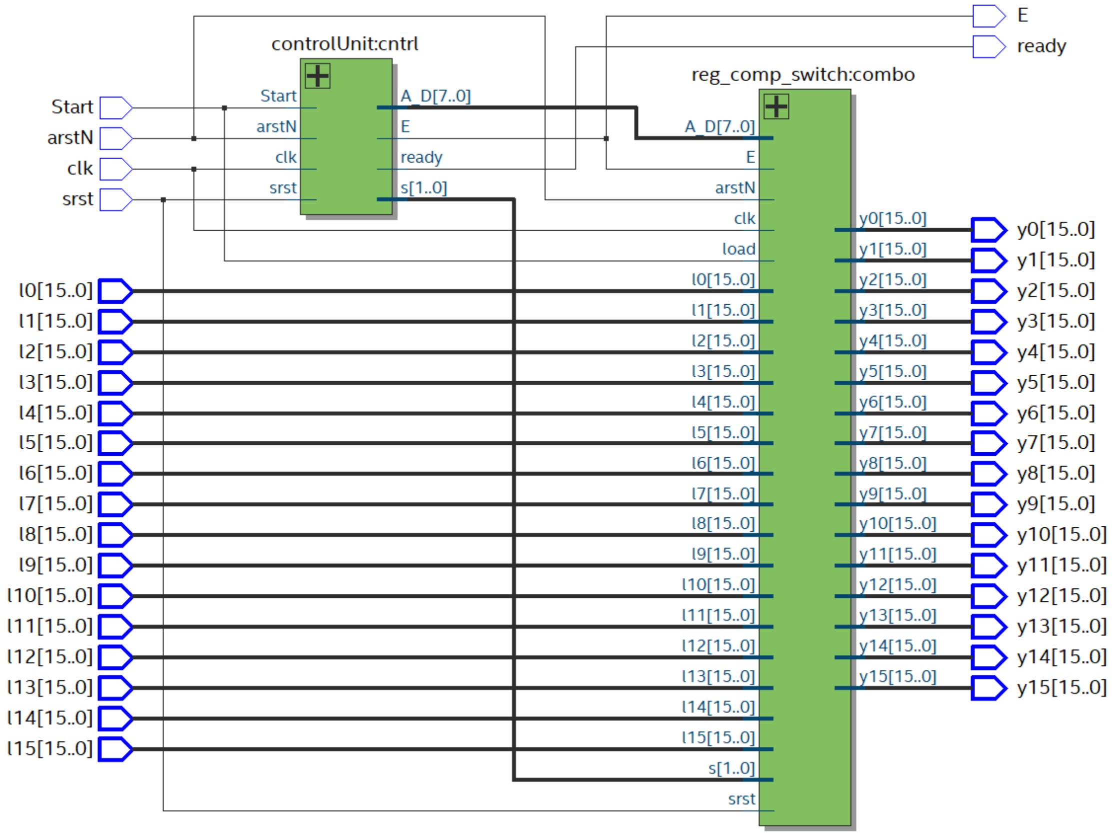
5-controlUnit:



The control unit connects both the choose_step_SFM, choose_A_D and down_counter_enable units together. This component controls the datapath and provides all the necessary control signals. Since on the transition we are always finalizing with the last step and loading the counter, so we want to hold the old value of s for one more cycle so that the choose_A_D unit keeps on outputting the same value for the current stage. The ready signal is also delayed 1 cycle since the sorting will finish after the transition.

The sorter:

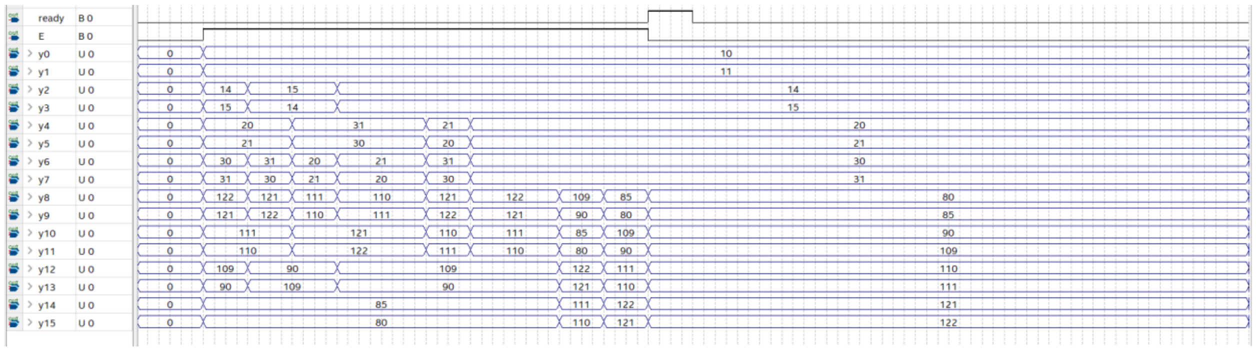
top_lvl:



The top level connects both the reg_com_switch and controlUnit units together. This component is the complete bitonic sorter that can be used as a black box.

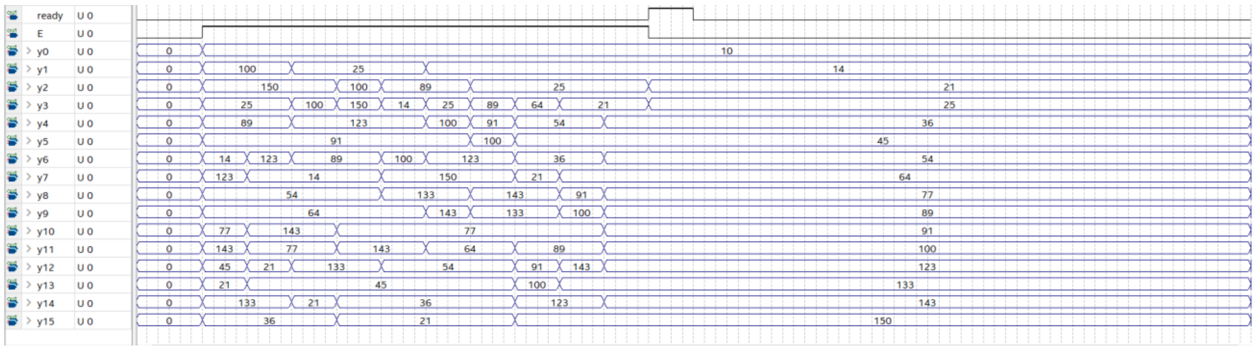
Simulations:

Simulation 1:



In the 1st simulation the sequence of numbers {10,11,14,15,20,21,30,31,122,121,111,110,109,90,85,80} is divided into two sequences one ascending {10,11,14,15,20,21,30,31} and the other descending {122,121,111,110,109,90,85,80}. We can clearly see that the output is sorted and correct.

Simulation 2:



In the 2nd simulation the sequence of numbers {10,100,150,25,89,91,14,123,54,64,77,143,45,21,133,36} is randomly divided into ascending and descending sequences. We can clearly see that the output is sorted and correct.

Conclusion:

The sorter designed in this project is an actual working sorter that can be used in the field. The sorter starts working when the Start button is enabled. Then it finishes stage by stage until it outputs the ready signal meaning the sequence is sorted. The efficiency and low cost make it accessible to all manufacturers to produce.

Future Work:

- The Bitonic sorter we designed could further be minimized and ran at higher clock speed if we implements serial comparison. All of the 16-bit wide register will be transformed each to a 16-bit shift-register and the comparison will be done bit by bit meaning that there will be 8 comparators 1-bit wide each. All of the 16-bit multiplexers will become 1-bit multiplexers. The comparison will be controlled through the use of a state machine at the output of each comparator. The state machine will be able to control the feedback mux(s) on the register and their enable lines. This approach was not implemented due to time constraints.
- The Design could also be implemented, and mass produced on VLSI chips to even reach higher speeds. This approach was not implemented due to financial constraints.