Hardware, High-Level Synthesis, and Software Implementations of Array Sorting Algorithms

Bubble Sort, Insertion Sort and Radix Sort

Exam ID: 6001

Exam ID: 6007

Exam ID: 6008

A report presented for the course of Embedded Systems

Course Code: CS4110
Department of Science and Industry Systems
Faculty of Technology, Natural Sciences, and Maritime Sciences
University of South-Eastern Norway
Kongsberg, Norway
December 2022

Hardware, High-Level Synthesis, and Software Implementations of Array Sorting Algorithms

Bubble Sort, Insertion Sort, and Radix Sort

Exam ID: 6001
Embedded Systems
Course Code: CS4110
University of South-Eastern Norway

Exam ID: 6007
Embedded Systems
Course Code: CS4110
University of South-Eastern Norway

Exam ID: 6008
Embedded Systems
Course Code: CS4110
University of South-Eastern Norway

***Abstract*—In computer science, we have come across multiple array-sorting algorithms which help us to organize elements in specific orders. There are many ways to implement them, i.e. with programming languages like C, C++, or java, we can implement them on software. We can also implement them on hardware using VHDL. This report discusses 3 different array sorting algorithms and the opportunities of implementing them both in software and hardware.**

***Index Terms*—ASMD, FSMD, data_mux, add_mux, FSMD, Control path.**

I. Introduction

This report intends to summarize our project work done for the course Embedded Systems- Hardware/Software co-design. We were told to select any 3 sorting algorithms and implement them through hardware and software. We choose bubble sort, insertion sort, and radix sort as our sorting algorithms and for the software tools, we will be using Vivado and Vitis HLS. After the implementation, we will observe which part is efficient- the hardware or the software.

II. Array Sorting Algorithms

A. Bubble Sort

Bubble sort, one of the simplest sorting algorithms works by swapping adjacent elements repeatedly. It is easy to use because of its simplistic algorithm. However, due to its time complexity, it is ineffective against big amounts of data [3], [4].

B. Insertion Sort

Insertion sort works by continually moving an element from an unsorted array to a sorted portion during each iteration until the entire list is sorted. It is more efficient in practice than other quadratic algorithms like selection or bubble sort. It is very simple to implement and efficient for smaller data sets. [13]

C. Radix Sort

Radix sort, also known as bucket sort, is a non-comparative sorting algorithm. Based on the radix of the elements, this algorithm distributes them into different buckets. The sorting starts with the least significant digit of each element and moves toward the most significant digit [1], [2].
For the simplicity of this project, we have limited sorting of the array using only one-digit integers.

III. Hardware Implementation

A. VHDL

VHDL stands for "VHSIC (very high-speed integrated circuit) hardware description language". This a very complex language intended for modeling the behavior of digital systems at multiple abstraction levels. [6], [8] It has become very useful as it captures complex digital circuits for both simulation and synthesis. It also has the ability to capture performance specifications in a form of a test bench. This helps to verify the behavior of a circuit over time. [11]
For the hardware implementation in this project, we are going to use VHDL.

B. Xilinx Vivado

The software suite we use to implement our hardware design is Vivado which is produced by Xilinx. It also has an IP integrator that allows engineers to configure and integrate IP from the large Xilinx IP Library. [7] We chose Vivado because it provides unparalleled run time and memory utilization and has robust performance.
We use version 2022.2 in this project.

C. Vitis HLS

Vitis HLS is a synthesizing tool that can be tightly integrated with the Vivado design suite and it also synthesizes C or C++ functions into RTL code. Vivado IP can be developed using Vitis HLS which later can be integrated into the vivado design suite for hardware designs. By performing high-level synthesis, this tool complies with the hardware kernels for the Vitis tools. [9], [12]
We use version 2022.2 in this project.

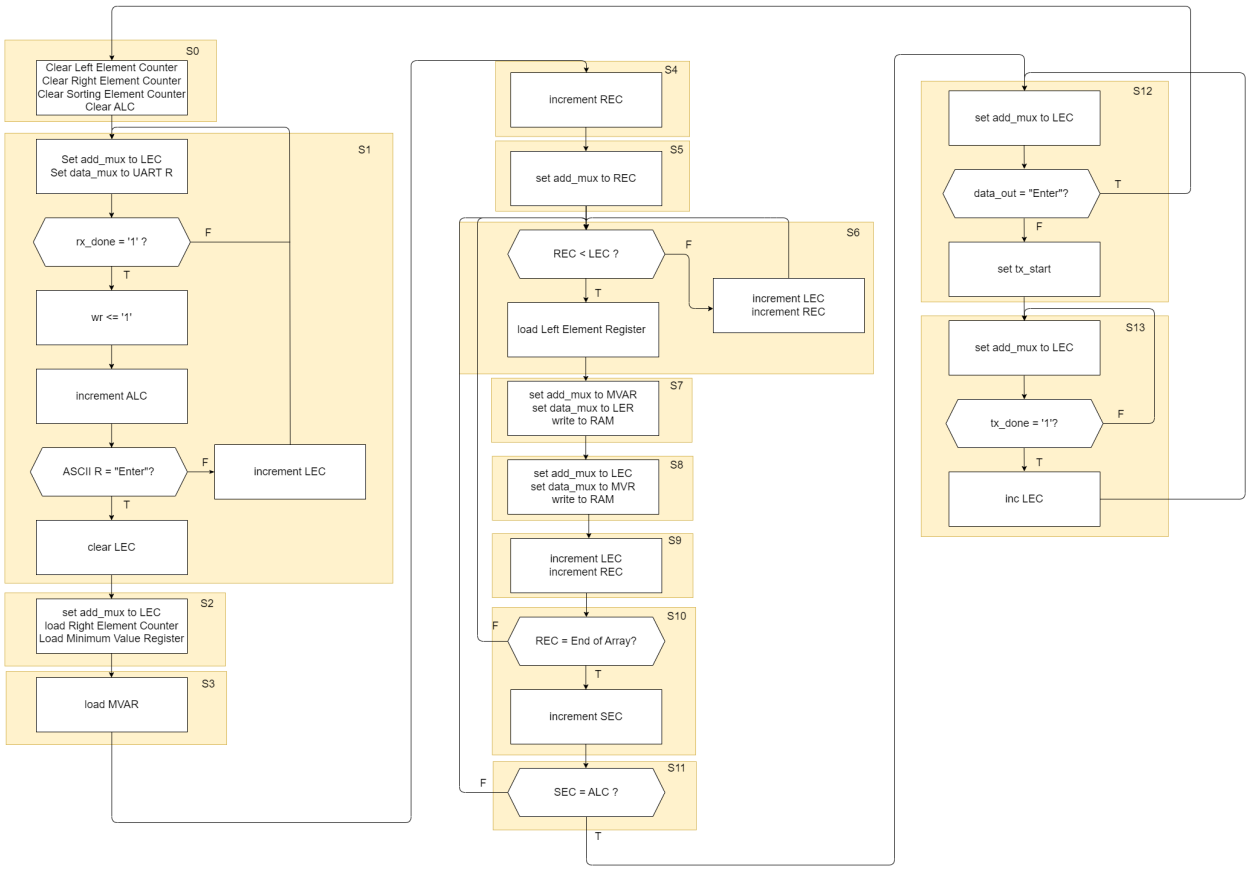

Fig. 1. ASMD of Bubble Sort

IV. Overview of Hardware Implementation

A. Overall Architecture

1) Top-Level Architecture: Each architecture is built upon small elements (i.e registers, counters, multiplexers) and glued back together in the top-level architecture. This has ensured flexibility while designing the components as well as given us the impression of how real world's components may work like.

2) RAM: Random Access Memory (RAM) has been used in each architecture for storing elements. It is connected to the multiplexers and UART Transmitter, as well as the control path.

3) Multiplexer: The multiplexer (also known as data selector) takes multiple signals as input and forwards a selected signal to the output line. The main benefit of using a multiplexer is that multiple input signals can share the same device. [10]

In this project we have used multiplexers in two forms, one as a data multiplexer and another as an address multiplexer.

4) UART: Universal Asynchronous Receiver and Transmitter (UART) is a hardware device for asynchronous serial communication. It can send parallel data through a serial line. A UART includes a transmitter and a receiver, where the transmitter is a shift register that loads data in parallel and then shifts it out bit by bit at a specific rate. The receiver shifts the data bit by bit and then reassembles the data. The serial line is '1' when it is idle. The transmission starts with a start bit '0', followed by data bits, and optional parity bit, and ends with stop bits '1'. [6], [8]

B. Architecture of Bubble Sort

In Bubble sort, we use four counters, one 8-bit RAM, three registers, one 3:8 address mux and one 2:4 data mux, a UART transmitter, a UART receiver, and a control path. In fig 2, we can see that the counters are:

- Left Element Counter (LEC)
- Right Element Counter (REC)
- Sorting Element Counter (SEC)
- Array Length Counter (ALC)

Also the registers are:

- Left Element Register (LER)
- Min Value Register (MVR)
- Min Value Address Register (MVAR)

At the very beginning, in S0 (State 0), we clear all the counters to set their value to zero. In S1 (State 1), we take the input from the user and store it in the RAM. For that, we set the address mux to LEC and data mux to UART R so that whatever input is given in the UART receiver it will be stored in the

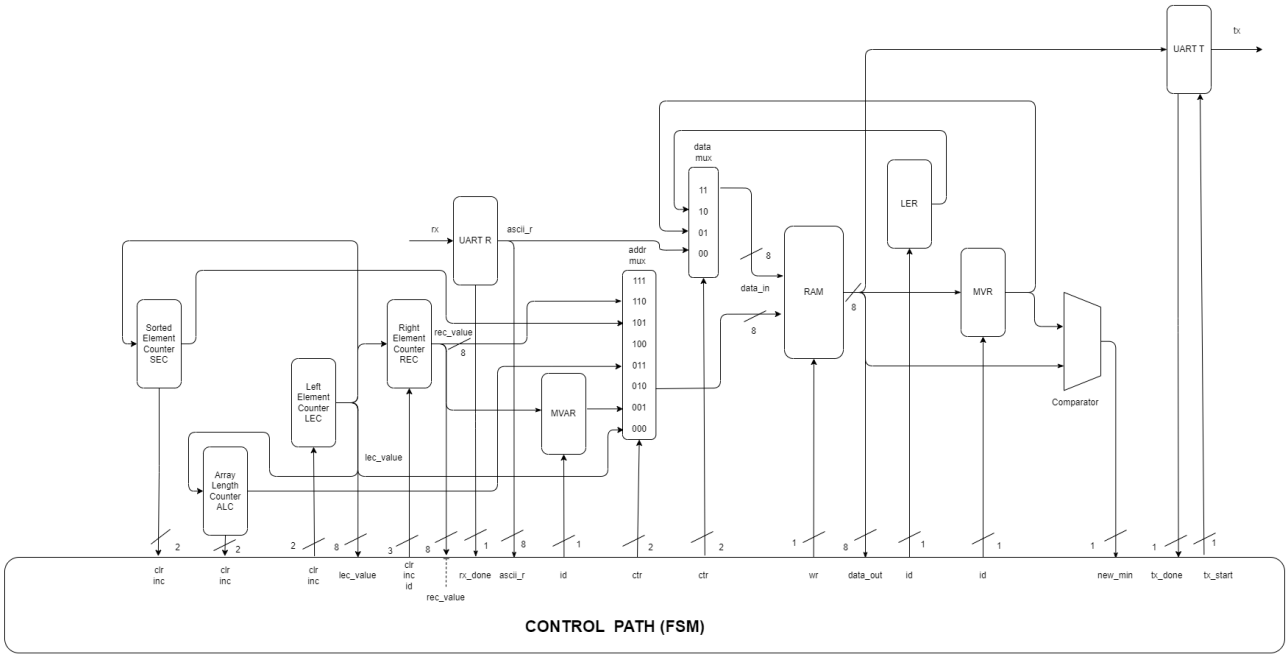

Fig. 2. FSMD of Bubble Sort

RAM and LEC will work as an address pointer in the RAM. We also set the wr (write) signal high so that RAM can accept the input. In S2 (state 2), and S3 (state 3), we set the LEC and load REC, MVR, and MVAR accordingly. We increment the value of REC in S4 (State 4) so that we can compare LEC with REC. We also update the address mux to REC in S5 (State 5). In S6 (State 6), we compare LEC with REC. If REC is smaller than LEC then we have found ourselves a new minimum, so we load it in the LER. Otherwise, we increment LEC along with REC and continue S6 which is the core algorithm for Bubble sort. If we have a new minimum in S6, then we have to swap the value. For doing that we use S7 (State 7) and S8 (State 8) where we use MVR to store the new minimum and then write it to RAM. Then in S9 (State 9), we increment LEC and REC to check the next input. In S10 (State 10) and S11 (State 11), we check whether REC reaches the end of the given array. If true then the search will again take place until the SEC is equal to ALC. Each time REC reaches the end of the array, SEC will increment and compare with ALC. After that, in S12 (State 12) and S13 (State 13), we print the sorted array. For that, we update the value of LEC until we get ASCII ”Enter” in the Ram output.

C. Architecture of Insertion Sort

The FSMD architecture of insertion sort algorithms consists of five counters, three registers, a 3:8 multiplexer for address mux, a 2:4 multiplexer for data mux, an 8-bit ram to store the unsorted and sorted array, a UART transmitter, a UART receiver, and control path. The counters are:

- Left Element Counter (LEC)
- Right Element Counter (REC)
- Sorting Element Counter (SEC)
- Array Length Counter (ALC)
- First Element Counter (FEC)

The registers that have been used are:

- Left Element Register (LER)
- Min Value Register (MVR)
- Min Value Address Register (MVAR)

We have divided the total work into several states. We can find in fig 3 that initially at S0 (State 0), we clear all the counters. Then we take the user input just like we did in the bubble sort in S1 (State 1). S2 (State 2) and S3 (State 3) are used to set LEC and load REC, MVR, and MVAR which is similar to Bubble sort. In S4 (State 4) we set SEC, FEC, and increment REC to compare with LEC. S5a (State 5a) is used to set the address mux to REC whereas, S5b (State 5b) is used for comparing REC with LEC. IF REC is smaller than LEC then we get a new minimum which needs to be written into the RAM. If we do not get a new minimum in S5b, we need to update the LEC and REC to check the next two values which happen in S6a (State 6a). In S6b (State 6b), if SEC is equal to the length of the array (ALC) then we print the sorted array. Otherwise, we continue searching in S5b. If we get a new minimum in S5b, we need to swap LEC with REC in S7 (State 7) and S8 (State 8), we do that by updating MVAR and LER with a new minimum value and writing it to the RAM at the LEC address in S7. In addition, we also update

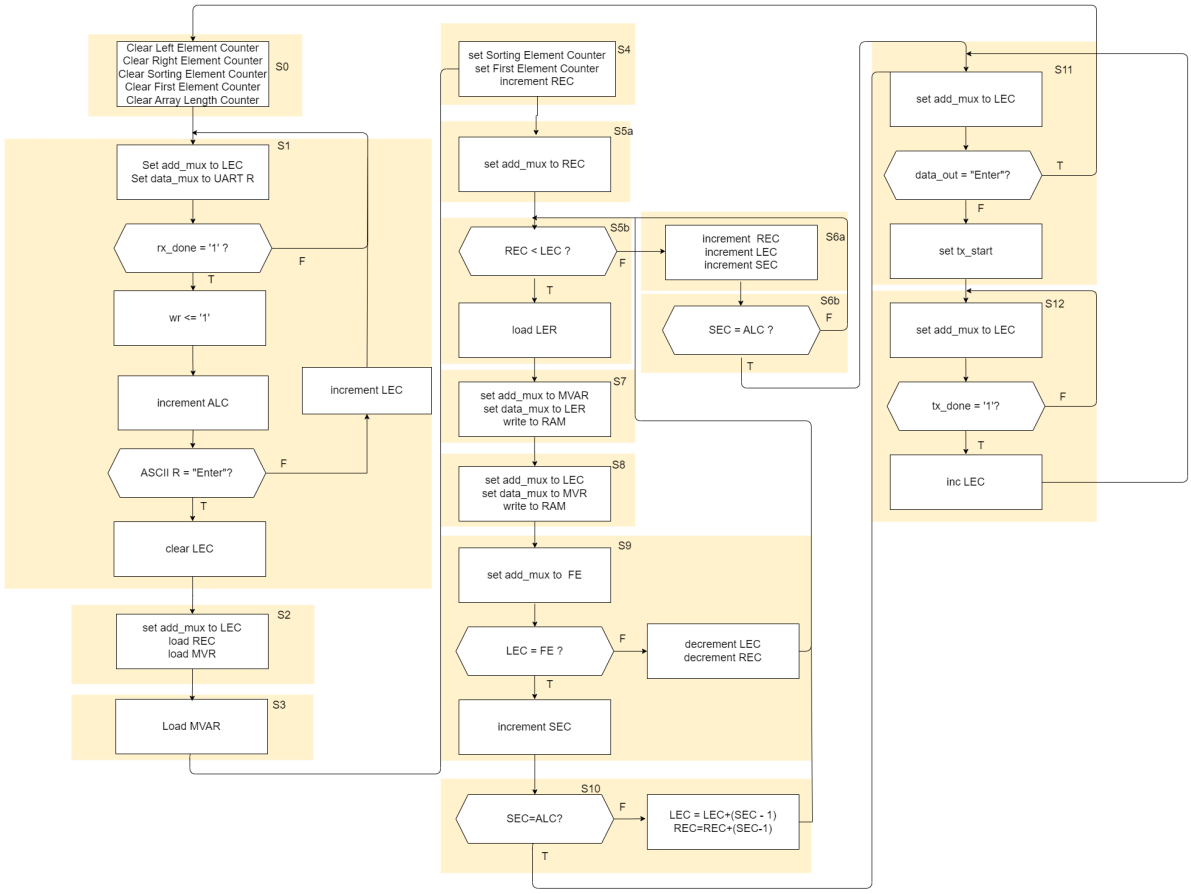

Fig. 3. ASMD of Insertion Sort

the LEC and MVR to write the new REC into the RAM in S8. In S9 (State 9), we check if the LEC reaches the FEC. If LEC reaches FEC then one element is sorted and we update SEC. If not, we decrement LEC and REC and continue our search in S5b. After that, we compare if SEC is equal to ALC. If true we print the sorted array through S11 and S12 just like bubble sort. If not, we update LEC and REC values and continue our search at S5b.

D. Architecture of Radix Sort

The FSMD architecture (Figure 6) of this sorting algorithm comprises multiple registers and counters, a 2:4 multiplexer as address multiplexer (add_mux), another 2:4 multiplexer as data multiplexer (data_mux), one UART receiver, one UART transmitter, an 8-bit RAM and a control path. Below are the counters that we have used:

- Left Element Counter (LEC)
- Sorting Element Counter (SEC)
- Temporary Element Counter (TEC)

The registers we have used to store values are:

- Increment Counting Register (ICR)
- Fixed Value Register (FVR)
- Current Value Register (CVR)

To begin with the procedure, in the S0 (state 0), we have sent clr (clear signal) to Left Element Counter (LEC) and Sorting Element Counter (SEC). S1 (state 1) is the part where we take the elements as input and store them in the RAM. Every time an element is stored in the RAM, all 3 counters LEC, SEC, and TEC are incremented. Once all the input has been stored, LEC is cleared again, and SEC and TEC are incremented. We end this state by setting the ASCII value of ”0” to the Increment Counting Register (ICR).
In S2 (state 2) we pass the value of ICR to the Fixed Value Register (FVR). The add_mux is pointed to LEC, which

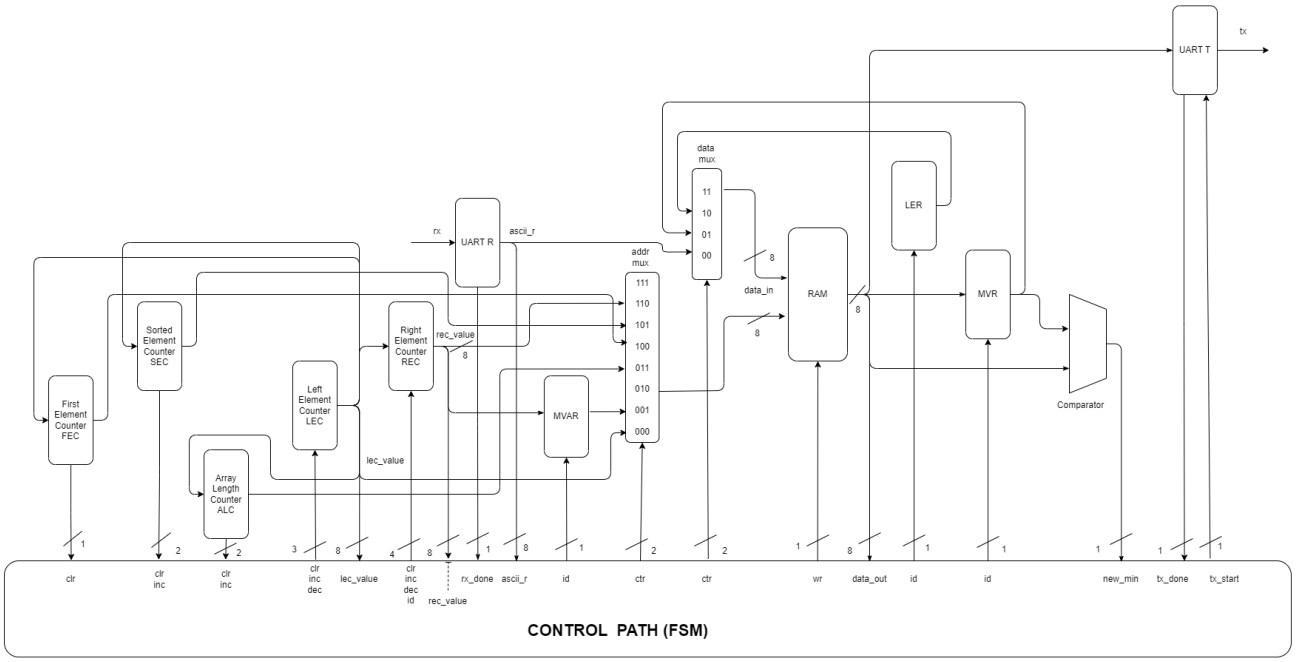

Fig. 4. FSMD of Insertion Sort

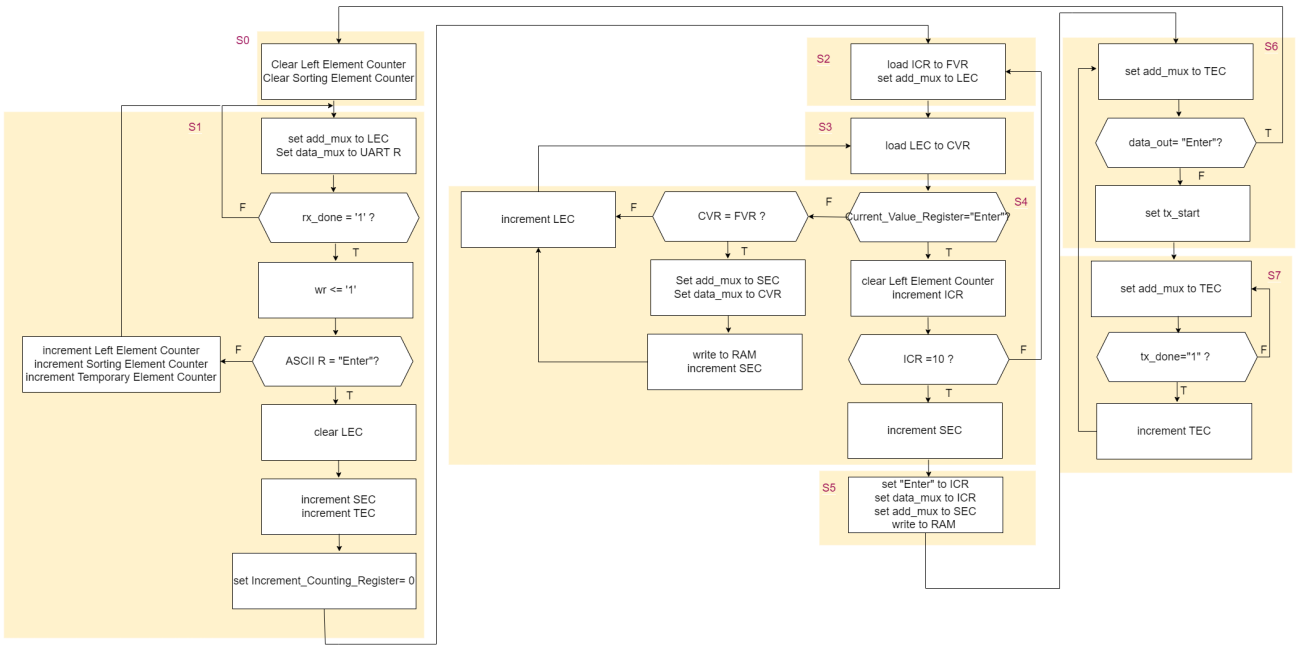

Fig. 5. FSMD of Radix Sort

indicates the first element of the array. In S3 (state 3), we pass the value indicated by LEC to the Current Value Register (CVR).

In S4 (state 4), we compare the value of the Fixed Value Register (FVR) and Current Value Register (CVR). As the value of FVR is now ”0”, and CVR holds the first value of the array, if they are equal, the value of CVR will be stored in RAM. If CVR and FVR are not equal, LEC will be

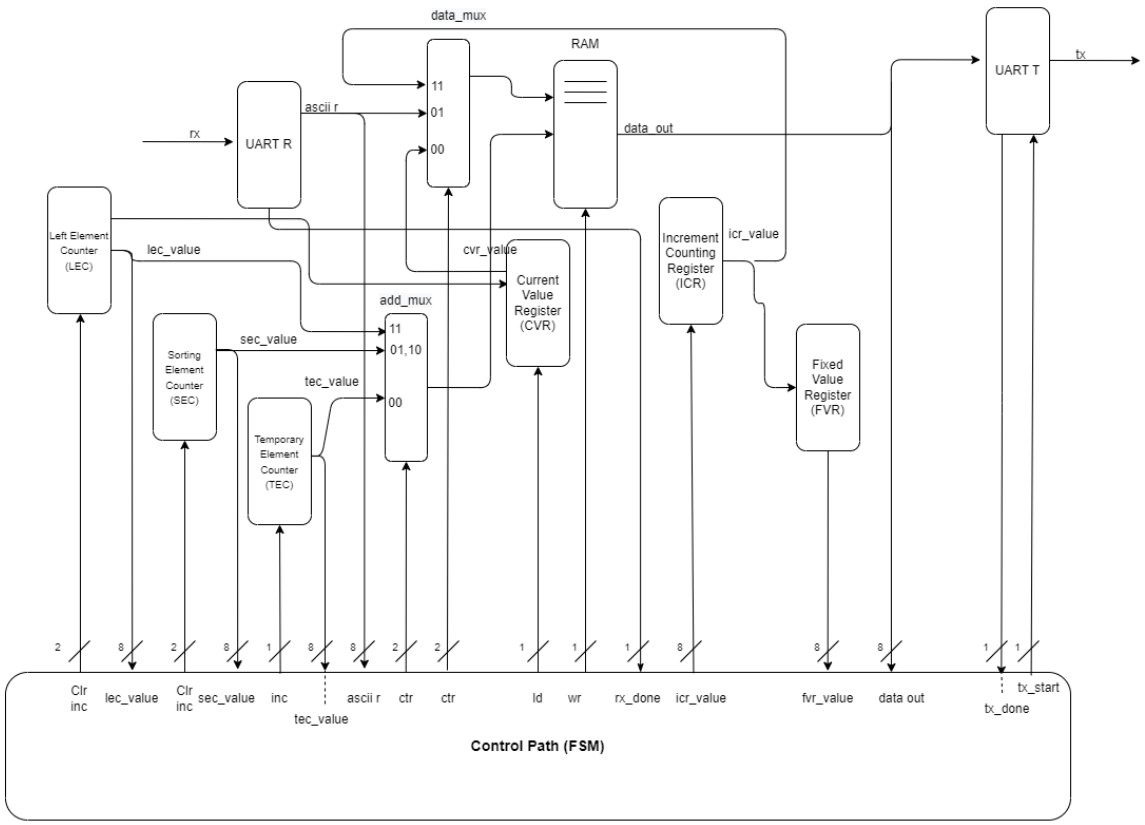

Fig. 6. FSMD of Radix Sort

incremented and it will be pointed at the second value of the array. As FVR still holds ”0”, this process will be continued until the values in the array are compared with FVR. Once we reach the end of the array, we are done with comparing all the values with ”0”. Now the LEC will be cleared again, pointing to the first value of the array, and ICR will be incremented (now it is 1). The value of ICR will be loaded into the Fixed Value Register (FVR). Again, the same steps will be followed and all the values of the array will be compared against ”1”. As we have considered only one-digit integers in the array for radix sort, we will increment ICR until it reaches 10. Once it reaches 10, our sorting is completed, we increment SEC (end of State 4) and store the ASCII value of ”Enter” in the RAM (State 5).
State 6 and 7 comprise the steps of transmitting the sorted array elements.

V. Software Implementation

The software implementation for insertion sort, bubble sort, and radix sort is done by using the C programming language. We take an unsorted array and the function will sort the array using the corresponding sorting algorithms functions. In our testbench file, we take an array which is an unsorted list and our goal to sort this array and return the sorted array result.

A. Insertion Sort:

We created a header file where we declare our insertion sort function. And in our insertionsort.c file we declare our insertionsort function. We take array and array size as function’s parameters. We start comparing the first two adjacent elements to find the minimum value and place it at the first of the index. We continue the loop until all the elements are not sorted. Then we need to synthesize and simulate our c code. After that, we need to export RTL where we get a zip file for vivdo to add the IP. Then generate the bitstream. We will follow the same procedure for the other two algorithms.

B. Bubble Sort:

We created a header file where we declare our bubble sort function. And in our bubble.c file we declare our functions. We take array and array size as function’s parameters. For the bubble sort algorithm, we also need a swap function when we get a maximum value in comparing two adjacent we swap it with the last index value.

C. Radix Sort:

For radix sort, we need to declare two functions. one for counting the values and placing them in another array and another function we need a for loop to find out the values

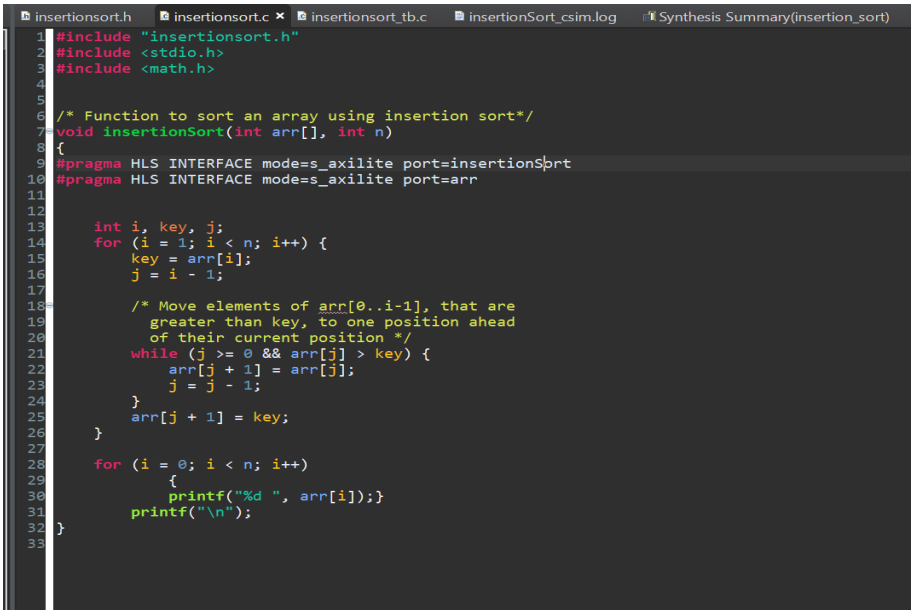

Fig. 7. Insertion sort c code

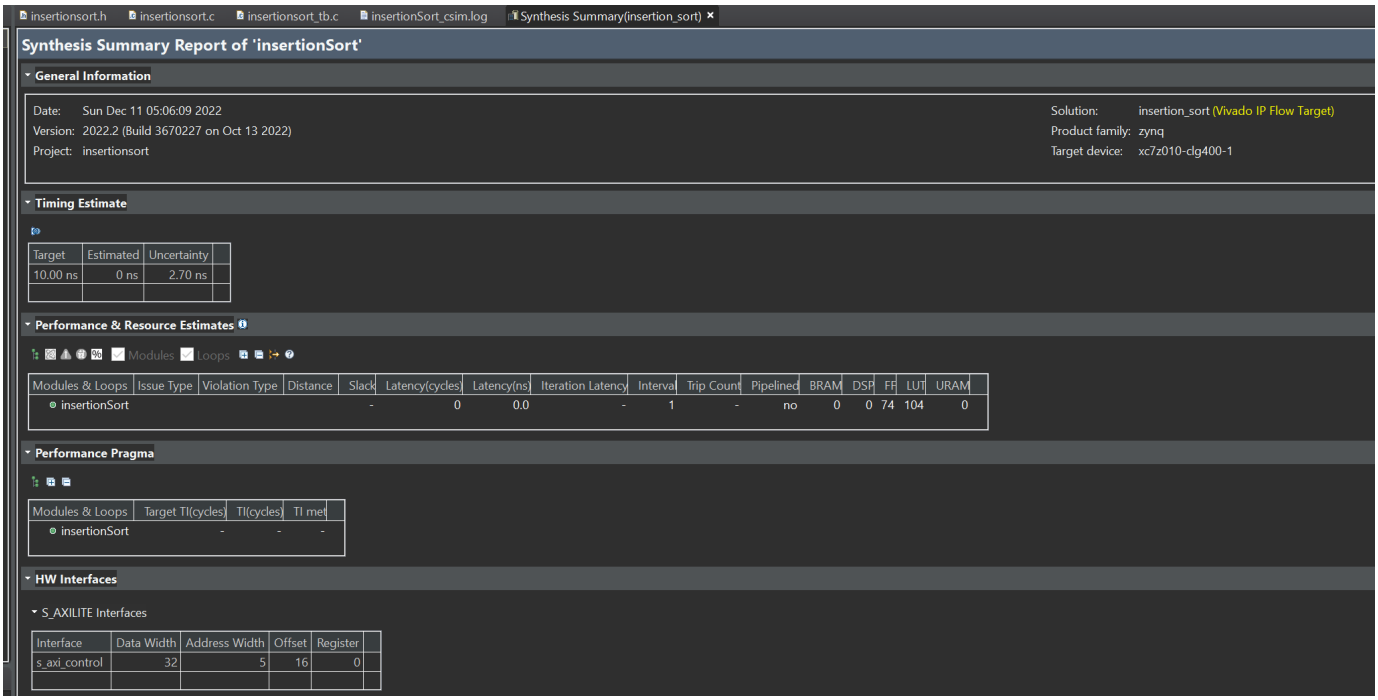

Fig. 8. insertion sort Synthesis

from 0 to 9. So we need to run the countSort function 10 times. Here we take a one-bit value only because in putty we will strike one button for declaring one number of the array.

VI. Result

A. Hardware Implementation

Basys3 board is used in terms of hardware implementation. Vivado serves as our IDE. For each sorting algorithm, we first create our FSMD (Fine State Machine Diagram). We use the algorithms to create our ASMD graphic. Following that, we build files in Vivado for each component, including RAM, registers, counters, UART R, UART T, and others. In accordance with our requirement of signals for each algorithm, we also build a control path file. Accordingly, we update the top file. Then we ran the synthesis, and it worked as expected as we can see in fig 7. However, when we attempt to run the simulation, it fails to provide any results because of problems in the code we placed in the control path file;

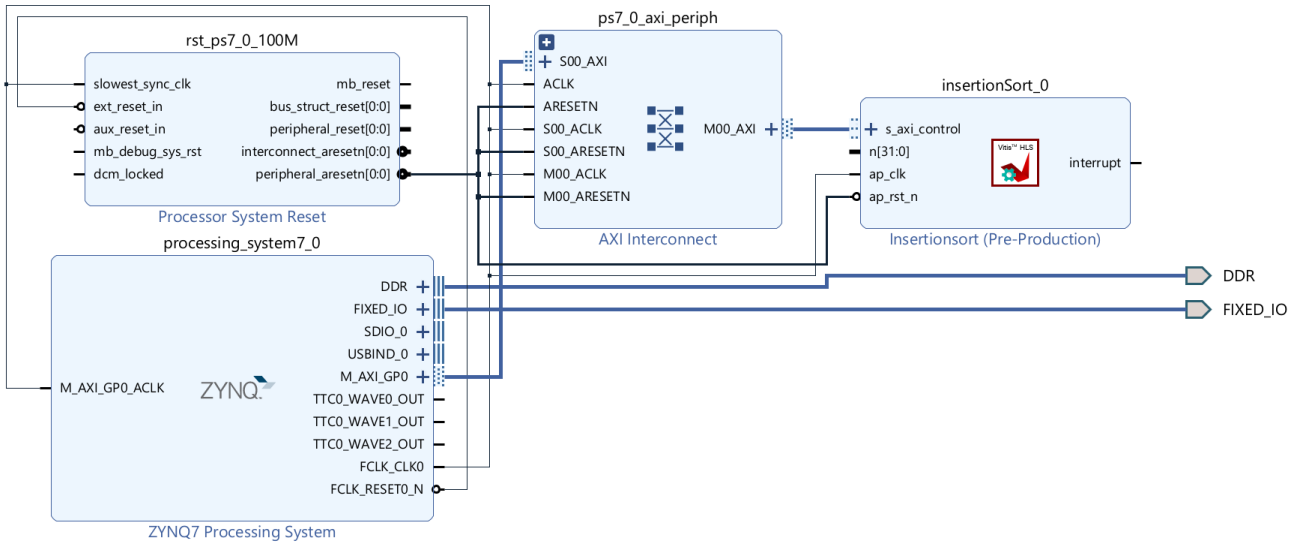

Fig. 9. insertion sort ip generator

bubble.c | bubble.h | bubble_tb.c | bubble_csim.log | Synthesis Summary(bubbleHLS)

```c
#include "bubble.h"
#include <stdio.h>
void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubble(int arr[], int n)
{
#pragma HLS INTERFACE mode=s_axilite port=bubble
#pragma HLS INTERFACE mode=s_axilite port=arr
#pragma HLS INTERFACE mode=s_axilite port=n

    int i, j;
    for (i = 0; i < n; i++){

        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++){
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
        }
    }
}
```

Fig. 10. Bubble c code

as a result, the bitstream generator is also inoperable. We are unable to produce the bitstream and debug the code due to time constraints.

B. Software Implementation

For software implementation, we use Vitis IDE and Zybo board. We create platform project. There we update the helloworld.c file. After building the project, we connect with the Zybo board. But we cannot see our desired result. Each step was successful even though we can see the sorted array result in the console. To generate the output in the zybo board we use inbyte() function and also we use while(1) loop. The inbyte() function takes one character input at a time and the loop continues while we press enter. But here we place the input unsorted values in the array. We find all the projects which take only one input at a time and execute the function while we do not press enter. In our software part, every step was successful despite the hardware connection part. We researched a lot

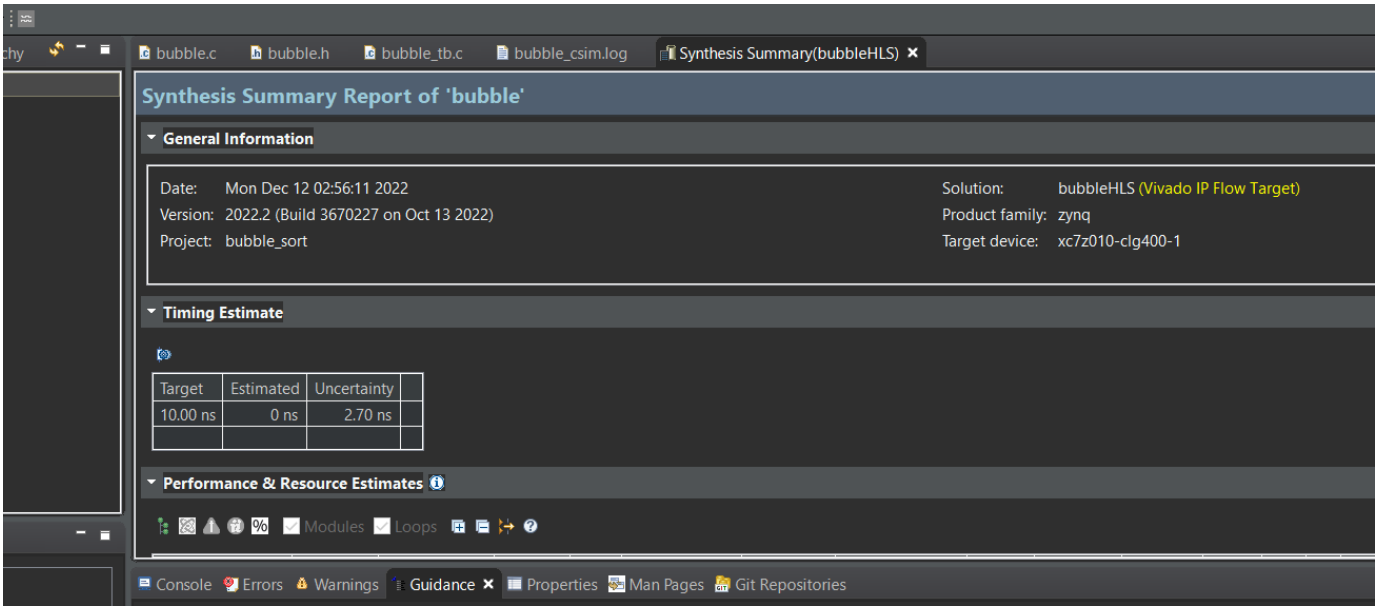

Fig. 11. Bubble sort Synthesis

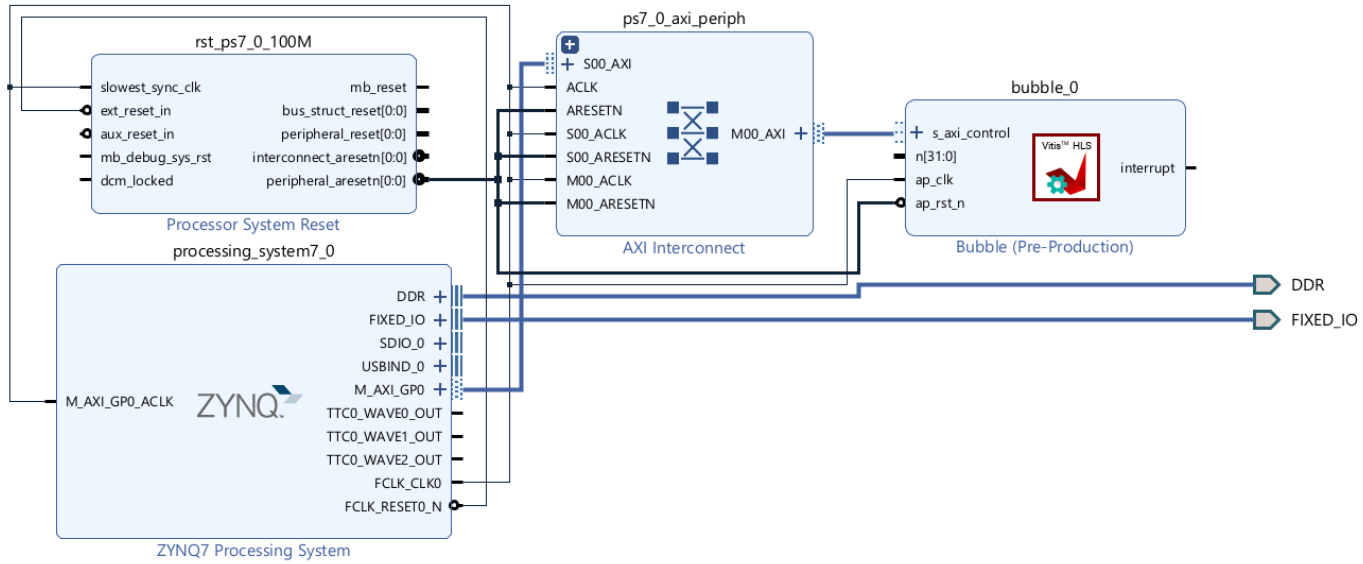

Fig. 12. Bubble sort ip generator

about how to convert the inbyte() function as an array type but due to the lack of resources, we could not get the desired output as zybo board.

VII. Comparison

A. Hardware and Software Implementation

It was much harder to build in hardware than in software since we need three sorting algorithms: Bubble sort, Insertion sort, and Radix sort. As we use the Zybo board to check the software implementation, we employ partial software implementation. We implement our hardware using a Basys3 board.

Recursion is frequently required in sorting algorithms, such as in merge sort and quick sort, both of which are challenging to implement on a hardware level. Therefore, we settle on three non-recursive algorithms. However, in terms of hardware implementation, it is still incredibly challenging. One of the main causes of this is that we are unable to confirm the accuracy of our FSMD architecture and ASMD chart. It takes a lot of effort to build the code and run the simulations necessary to detect any bugs in the control flow. Another factor is how quickly the Vivado IDE, which we use, provides hints to fix bugs. Even if we discover the error, we are unable to fix it because

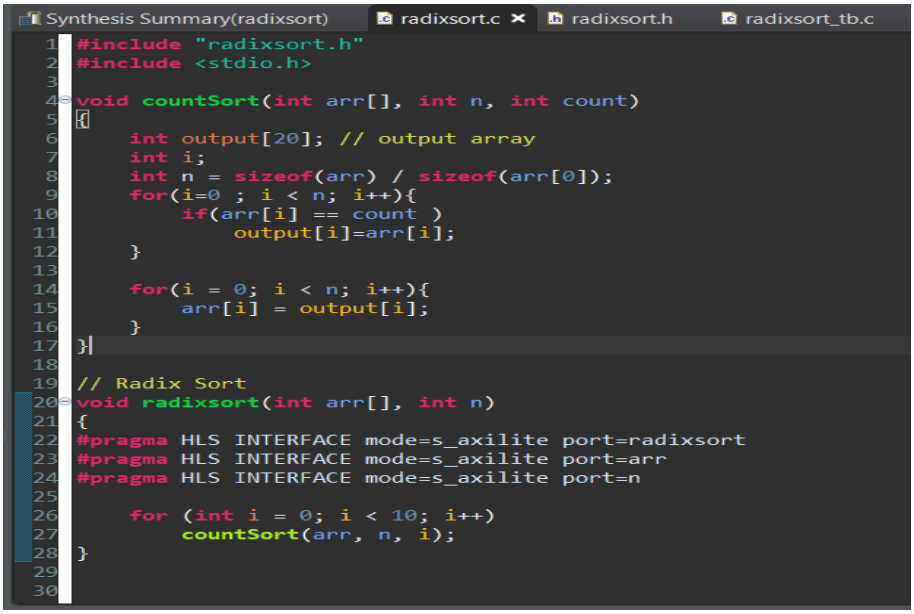

Fig. 13. Radix c code

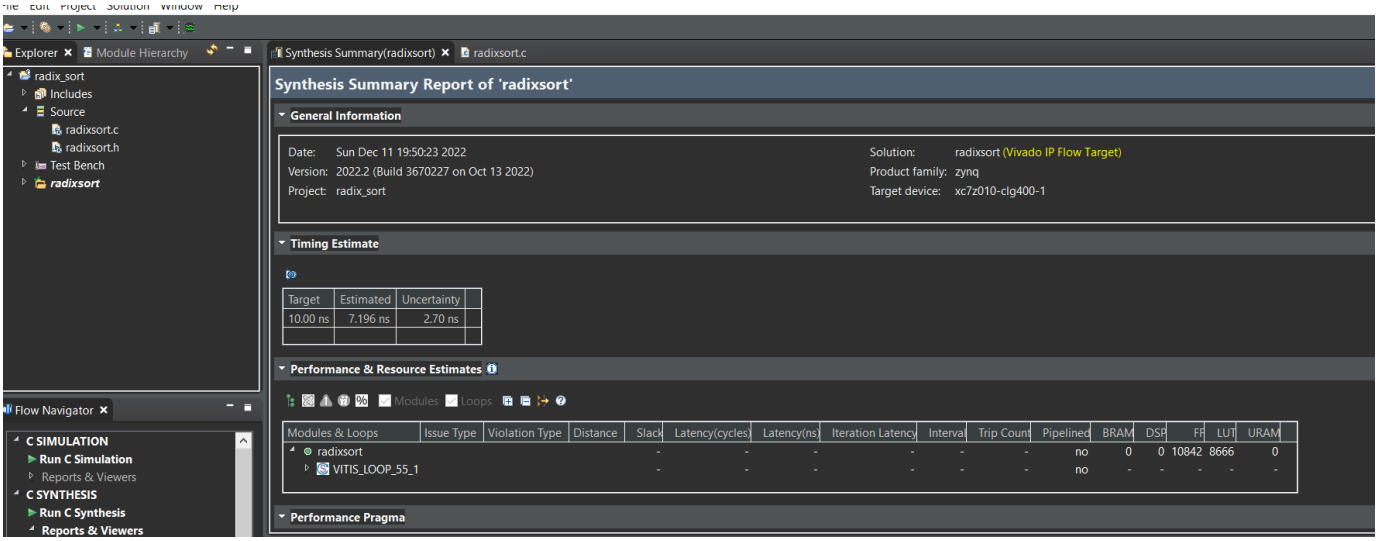

Fig. 14. radix synthesis

the compiler did not provide any hints. Finally, there are very few sources that are relevant to the project. Without any relevant sources, completing the project takes a long time.

We successfully complete the software portion of the software implementation, but we are unable to run it on the hardware due to some restrictions. However, compared to hardware implementation, it is simpler.

B. Time and Space Complexity

The most crucial calculations in any sorting algorithm are those involving time and spatial complexity. We can see from our FSMD design in Figures 1, 3, and 5 that Radix sort requires the fewest counters, Insertion sort requires the highest, and Bubble sort occupies the middle ground. However, all three algorithms require the same amount of RAM. Here is Table I which allows us to quickly compare the temporal complexities of the three algorithms. The space complexity of the algorithms is shown in Table II.

TABLE I
TIME COMPLEXITY

Algorithm	*Best*	*Average*	*Worst*
Bubble Sort	$\Omega(n^2)$	$\theta(n^2)$	O (n^2)
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	O (n^2)
Radix Sort	$\Omega(nk)$	$\theta(nk)$	O (nk)

For Bubble sort and Insertion sort,
n = total number of elements in the array

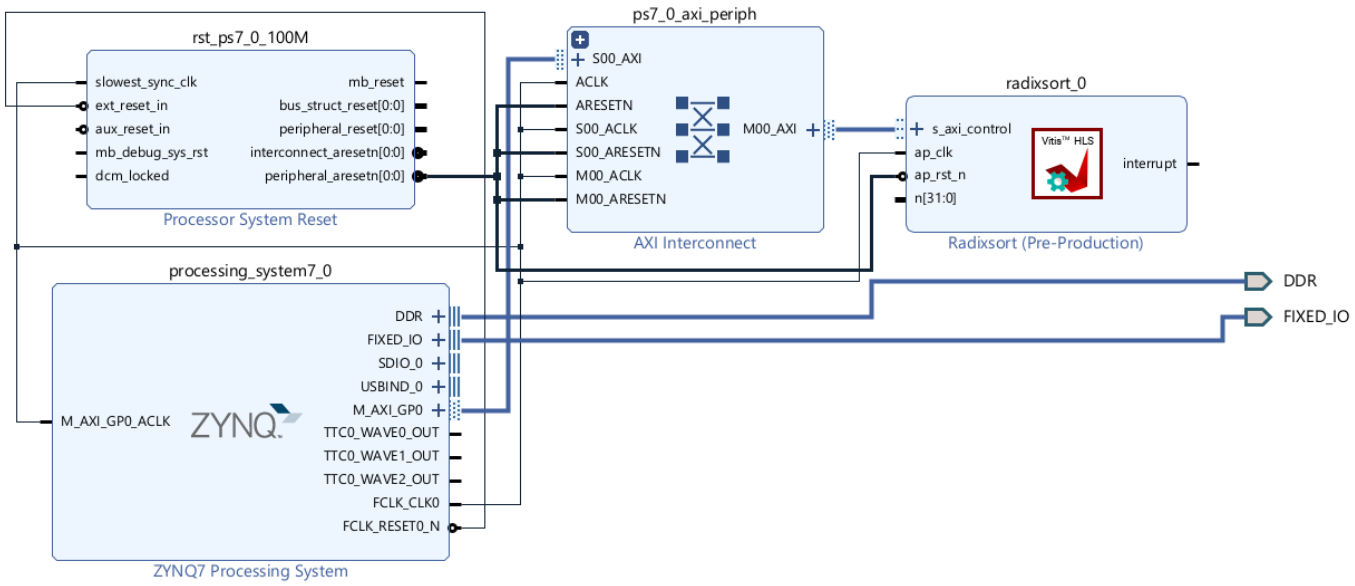

Fig. 15. radix sort ip generator

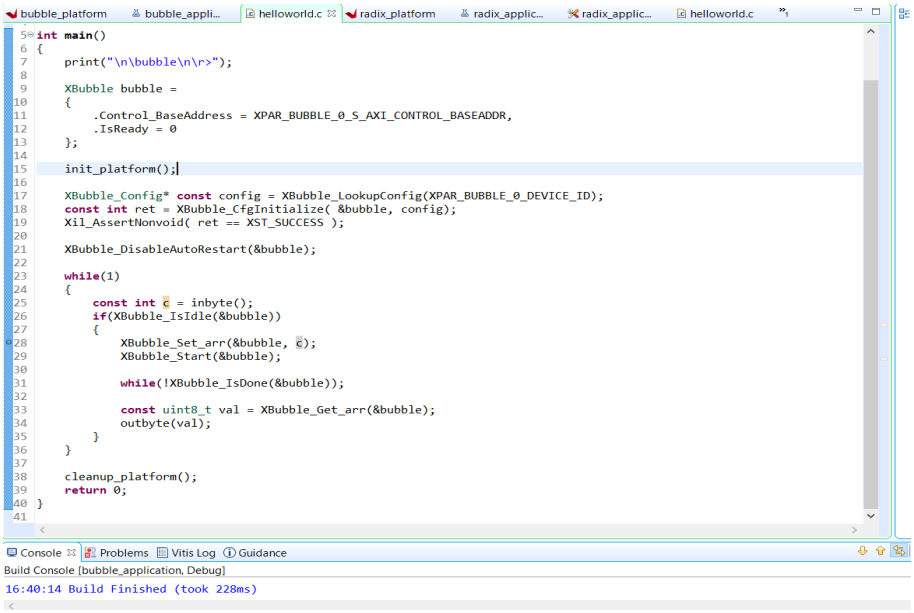

Fig. 16. Zybo board connection code

TABLE II
SPACE COMPLEXITY

Algorithm	*Worst*
Bubble Sort	o(1)
Insertion Sort	o(1)
Radix Sort	o(n+k)

For Radix sort,
n = total number of elements in the array
k = largest element of the input array

VIII. CONCLUSION

This project has been very critical in terms of VHDL coding and HLS implementation for three different algorithms. We had a great learning opportunity of getting started with VHDL coding, which is very new to us. Also, we got to understand how each element of the hardware can be modified using VHDL and the Vivado design suite.
Hardware and software are 2 important components of systems. With the help of co-design, the new technology space evolves that gets benefits from the synergy between these two components.

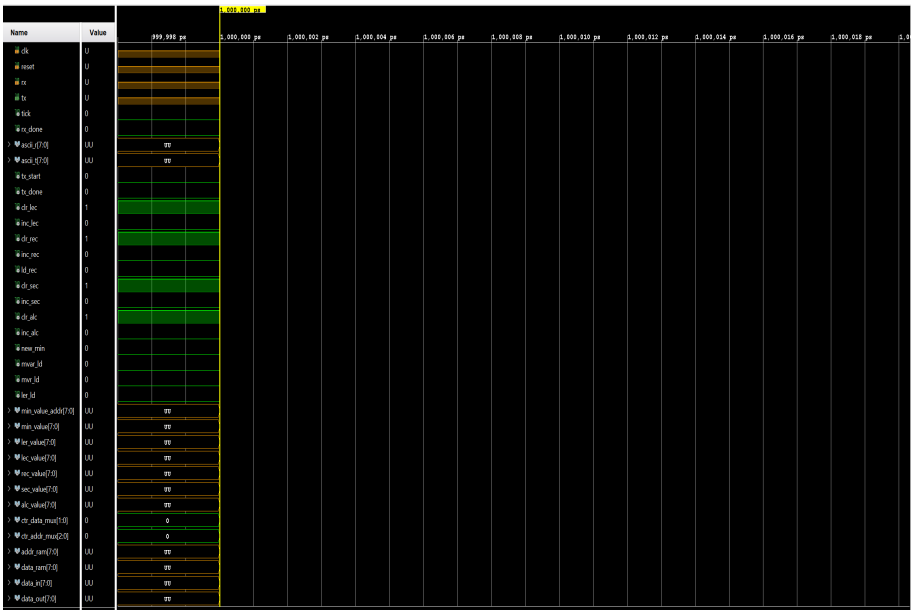

Fig. 17. Simulation

Within 3 weeks of time, it was hard to implement all the implementation of 3 algorithms. But we tried out best to put all our knowledge into the design solution for these sorting algorithms.

REFERENCES

[1] 'Radix sort', Wikipedia. Nov. 10, 2022. Accessed: Dec. 09, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Radix_sortoldid=1121168753

[2] 'Radix Sort - javatpoint'. https://www.javatpoint.com/radix-sort (accessed Dec. 09, 2022).

[3] 'Bubble Sort - javatpoint'. https://www.javatpoint.com/bubble-sort (accessed Dec. 09, 2022).

[4] 'Bubble sort', Wikipedia. Nov. 28, 2022. Accessed: Dec. 09, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Bubble_sort&oldid=1124348623

[5] 'VHDL', Wikipedia. Dec. 05, 2022. Accessed: Dec. 11, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=VHDLoldid=1125781404

[6] 'Universal asynchronous receiver-transmitter, Wikipedia. Dec. 03, 2022. Accessed: Dec. 11, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Universal_
asynchronous_receiver-transmitter&oldid=1125388023

[7] 'Xilinx Vivado', Wikipedia. Oct. 16, 2022. Accessed: Dec. 11, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Xilinx_Vivado&oldid
=1116440962

[8] P. Chu, "FPGA prototyping by VHDL examples", John Wiley Sons Inc, Hoboken, Second edition, 2017.

[9] 'Introduction to Vitis HLS • Vitis High-Level Synthesis User Guide (UG1399) • Reader • Documentation Portal'. https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS (accessed Dec. 11, 2022).

[10] 'Multiplexer', Wikipedia. Jun. 29, 2022. Accessed: Dec. 12, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Multiplexer&oldid
=1095698273

[11] 'An Introduction to VHDL'. http://www.uco.es/ ff1mumuj/h_intro.htm (accessed Dec. 12, 2022).

[12] 'Vitis HLS — Vitis™ Tutorials 2022.1 documentation'. https://xilinx.github.io/Vitis-Tutorials/2022-1/build/html/docs/Getting_Started/Vitis_HLS/Getting_Started_Vitis_HLS.html (accessed Dec. 12, 2022).

[13] 'Insertion sort', Wikipedia. Nov. 14, 2022. Accessed: Dec. 12, 2022. [Online].
Available:
https://en.wikipedia.org/w/index.php?title=Insertion_sortoldid=1121840955