EE314 Digital Circuits Laboratory Project Report

Mehmet Eren Altuncu, Ramazan Cem Çıtak, Ahmet Genç

Electrical-Electronics Engineering Department, Middle East Technical University
Ankara, TÜRKİYE

altuncu.eren@metu.edu.tr

cem.citak@metu.edu.tr

genc.ahmet@metu.edu.tr

***Abstract*— This paper describes the theoretical and experimental results of our project of FPGA Implementation of Simple Quality of Service based Queuing.**

***Keywords*— Quality of Service, Queueing Algorithm, FPGA, Verilog HDL, VGA Interface**

I. INTRODUCTION

This document is the final report for the EE314 Digital Circuits Laboratory project that we are expected to implement a Simple Quality of Service based Queuing on FPGA board. In this project, at first, we give a binary input of 4-bit sequence to the system using two buttons for binary 0 and 1 as well as a start button before sending the input data sequence. Then, according to its two most significant bits which are header bits, the system chooses one of the four buffers, each of which has 6 registers. Also, the two least significant bits is used to fill the chosen buffer in decimal form. After that, for each buffer, the oldest 4-bit input sequence is read first in a first-in-first-out manner, which forms the queueing algorithm. In addition, some of the input data are lost due to the exceeding capacity of the buffer, which are represented by dropped data packets. In the end, it is supposed to show buffers with different colors having available entered data values as well as the number of transmitted, received and dropped data sequences, or packets on a screen using a VGA interface.

II. BACKGROUND INFORMATION AND REQUIREMENTS FOR THE PROJECT

Firstly, we did research about Quality of Service and the use of VGA by a FPGA board using Verilog language in Quartus II 13.1 program. As an FPGA, we used Altera DE1-Soc FPGA Board.

A. Quality of Service

It keeps the capacity of a network system in optimum capacity and desired conditions. In this project, there are two requirements for the Quality of Service System: Latency Requirement and Reliability Requirement. The latency precedence for the buffers is given as

Buffer 1 > Buffer 2 > Buffer 3 > Buffer 4

which means that Buffer 1 will be read first and will have lowest delay in the system.

On the other hand, the reliability precedence for the buffers is given as

Buffer 4 > Buffer 3 > Buffer 2 > Buffer 1

which means that the number of dropped data packets will be lowest in Buffer 4.

B. The Queueing Algorithm and a Working System as a Whole

According to the Latency and Reliability Requirements, a queueing algorithm is formed for the system. These requirements are satisfied while taking the inputs as well as reading the data packets at every 3 seconds from the buffers.

Our simple Quality of Service as a whole system has several steps and can be represented by a state diagram as shown in Figure 1.

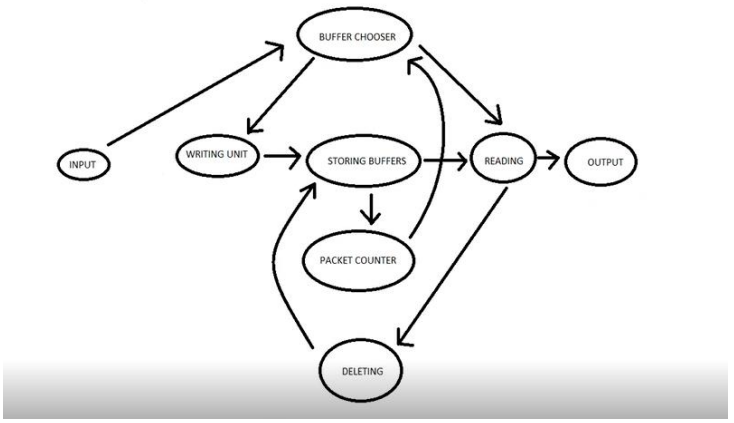

Figure 1. The state diagram for the system, which makes easier to understand the algorithm used in the project

Firstly, we considered how we can give input to the system, and we concluded that the input should not be related to clock signal. In other words, taking input process should be independent of time. This is the first main point. After that, according to the input, the system has to choose a buffer that one of the data packets will be written. After choosing buffer, writing process should start. The system should write input into buffer which is chosen, and this data should be stored in this buffer. After storing, the system has to read data. It should determine the numbers of the packet in buffers, and according to this number, it should choose correct buffer. It has to consider latency and reliability requirements to choose correct buffer and then system should read data in every 3 seconds.

After reading, it should delete data which is read, and system should show this data as output.

C. VGA in FPGA using Verilog HDL

We used a basic reference code for our VGA interface for the project. The Smiley Face code formed the skeleton of our VGA interface code. [1] Then, we drew our buffering system screen for the VGA interface on a website that shows each pixel in our drawings. [2] This made fairly easier the determination of the values of the horizontal and vertical borders in our buffers and numbers on the 640x480 screen. Adjusting the colors in the VGA interface by Verilog code is done by using three colors: Red, Green, and Blue.

As in sample code of Buffer 1 in Figure 2, an '`if`' statement is required for the determination of the borders of a buffer. In this `if` statement, `hcount` and `vcount` as well as `hcoun` and `vcoun` gives us the scanning information, or synchronization of horizontal and vertical axes on the screen. By doing so, we achieve every pixel on a 640x480 screen and adjust the colors as desired. The synchronization code for horizontal and vertical axes is given in Figure 2.

```
always @(posedge clk25MHz)  // horizontal counter
   begin
      if (new_hcount < 799)
         new_hcount <= new_hcount + 1;  // horizontal counter (including off-screen horizontal 160 pixels) total of 800 pixels
      else
         new_hcount <= 0;
   end  // always

always @ (posedge clk25MHz)  // vertical counter
   begin
      if (new_hcount == 799)  // only counts up 1 count after horizontal finishes 800 counts
         begin
            if (vcount < 525)  // vertical counter (including off-screen vertical 45 pixels) total of 525 pixels
               vcount <= vcount + 1;
            else
               vcount <= 0;
         end  // if (counter_x...
   end  // always
// end counter and sync generation
```

Figure 2. The code for synchronization of horizontal and vertical axes on our VGA interface

Since the pixel clock for the 640x480 resolution screen needs to be 25 MHz, we used a frequency divider to divide the 50 MHz FPGA clock by 2. [3] The frequency divider code is given in Figure 3.

```
   always @ (posedge clk)
begin
counter = counter+1;
if (counter ==1)
begin
clk25MHz=~clk25MHz;
counter=0;
end
end
```

Figure 3. Frequency Divider code in Verilog HDL

In addition, inside of each register in a buffer is filled with a certain color. Buffer 1 registers are Red, Buffer 2 registers are Blue, Buffer 3 registers are Yellow, and Buffer 4 registers are Green. The color determination is done by three colors of VGA. In Figure 4, the code below the if statement with `red`, `blue`, and `green` gives green color in a register for Buffer 4 if the assignment operators <= have `8'hff`, `8'h00`, and `8'h00`, respectively for red, blue, and green.

```
else if (hcount < 260 && hcount > 220 && vcount < 350 && vcount > 310 && Buffer4[0] == 1)begin
hcoun = hcount -180;
vcoun = vcount +0;
  if(Buffer4[1] == 0 && Buffer4[2] == 0)begin
     if(((vcoun == 330 || vcoun== 326) && (hcoun == 59 || hcoun == 60))||
        ((vcoun==327 || vcoun==328 || vcoun == 329) && (hcoun==58 || hcoun == 61)))begin
        green <= 8'h00;
        blue <= 8'h00;
        red <= 8'h00;
     end
     else
     begin
        green <= 8'hff;
        blue <= 8'h00;
        red <= 8'h00;
     end
  end
```

Figure 4. The code for writing 0 in Buffer 4 and filling inside of one of its registers with green

The other numbers on our VGA interface are shown in a similar way as in the Figure 5, which is the code for showing the read data of 3 in Buffer 1 on the right side of the screen, where the counters are.

```
else if(read_data1 % 10 == 3 &&((hcount == 414 && vcount <247 && vcount > 239)||
((vcount == 240|| vcount == 243 || vcount==246)&& hcount<415 && hcount > 409)))
begin
   green <= 8'h00;
         blue <= 8'h00;
         red <= 8'h00;
end
```

Figure 5. The code for showing 3 on the received data part of the counters on the right side of our screen

III. Implementation of the Project and Demonstration Results

Our design consists of two main parts, which are writing and reading operations of the system.

A. Writing 4-Bit Data Sequence

We mentioned about our general solution. Now, we can mention detailly. 50MHz frequency was used for FPGA. PIN_AF14 was assigned for this frequency. [3] We used this clock for first always. One array which has four bits for input signal was assigned. We get input data using start, one, and zero buttons. We pressed firstly start button and then give input using one and zero buttons. This process is not related to clock. In other words, process can continue until input array get data which has four bits. We used if and else configurations to make independent input from clock. When we press start button, our variable is equal to one. And then, we assigned input data which is four bits with using if and else conditions. If one is equal to one, system write one into array. If zero is equal to one, this means zero button is pressed, system writes zero into array. As we expected. After taking input, system has to choose buffer which data is written in as we mentioned before. This part is related to most significant bits of input. If most significant bits are 00 system is choosing buffer 1. If they are 01, system is choosing buffer 2. If they are 10, system is choosing buffer 3. Finally, if they are 11, system is choosing buffer 4. We used `if` and `else` conditions to choose buffer.

After choosing operations, data which is two least significant bits is written in buffers which is chosen. We used again if and else conditions for this part. As you expected, if least significant bits are 00, it is writing 0.
If they are 01, it is writing 1. If they are 10, it is writing 2. If they are 11, it is writing 3. If buffers are not empty, system is firstly shifting 3 bits and then input is written into buffers 1 and 2 bits. Also, we create buffers with 18 bits. Each packet of buffers has 3 bits. Most significant bit of 3 bits determine if buffer packet is empty or not. In other words, if most significant bit of 3 bits is equal to 1, this means packet of buffer is not empty. If most significant bit of 3 bits is equal to 0, this means packet of buffer is empty. These configurations help us in reading data. We will see this in reading part.

B. Reading 4-Bit Data Sequence

After writing operations, reading part can start. In this part, we considered latency and reliability conditions. The desired condition is the lowest delay is desired for Buffer 1. For example, if some data are available at Buffer 1 and Buffer 2, Buffer 1 should be read first. This condition is related to latency. Second desired condition is reliability. This means that the lowest packet drop rate is desired for Buffer 4. When we considered these two conditions, we wrote code according to this. In our code, we used `if` and `else` conditions.
Firstly, we considered if Buffer 4 is full or not. If it is full read data in this buffer. If Buffer 4 is not full, system should look if Buffer 3 is full or not and if it is full, system should read data from Buffer 3. If it is not full, system should look if Buffer 2 is full or not and if it is full, system should read data from Buffer 2. If it is not full, system should look if Buffer 1 is full or not and if it is full, system should read data from Buffer 1. If all buffer is not full, system should read data firstly in Buffer 1, secondly in Buffer 2, thirdly in Buffer 3, and lastly in Buffer 4. When we do this process, data which is from Buffer 1 is read fastest. Also, there is a loss data. Our system can read data in every 3 seconds as we mentioned before. Therefore, when a buffer is full and we give inputs in less than 3 seconds, loss data occurs. System cannot read this data because it is delated due to writing new input. We can easily see this reading operations which are latency and reliability with following Figure 6, Figure 7, Figure 8, Figure 9, and Figure 10. When we look Figure 6, we expect that system will read and delete 2 which is 1110 from Buffer 4. 11 indicates Buffer 4, 10 indicates the value 2. When we look Figure 7, output is 1110 as we expected. Reliability is satisfied as we can see.

Figure 6. First step of example

After that, when we look Figure 7, we expect that system will read and delete 2 which is 0110 from Buffer 2. 01 indicates Buffer 2, 10 indicates the value 2.

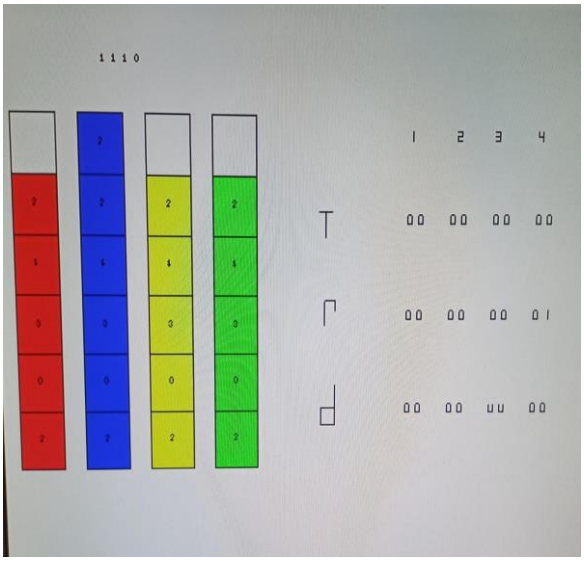

Figure 7. Second step of example

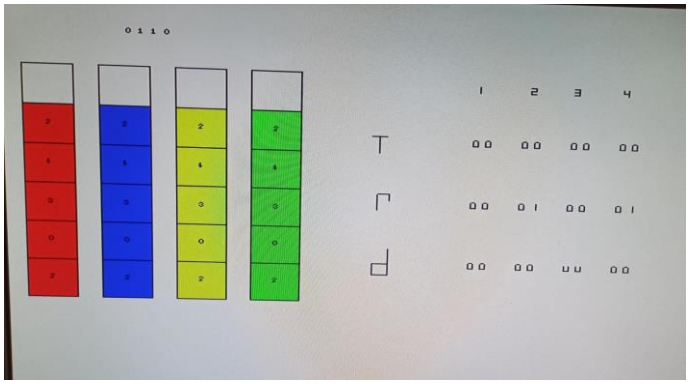

Figure 8. Third step of example

When we look Figure 8, output is 0110 as we expected. Again, reliability is satisfied as we can see.

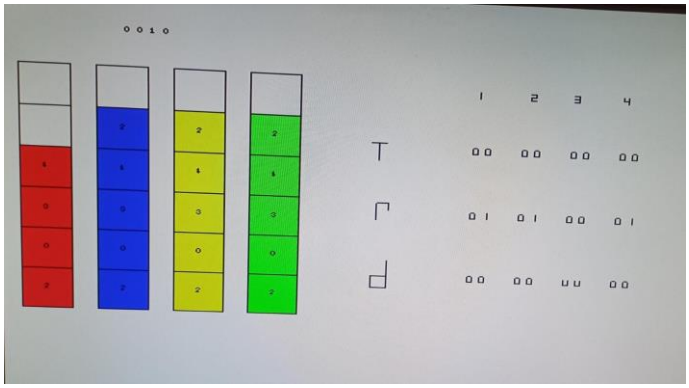

Figure 9. Fourth step of example

After that, when we look Figure 9, we expect that system will read and delete 2 which is 0010 from Buffer 1. Here, 00 indicates Buffer 1, and 10 indicates the value 2.

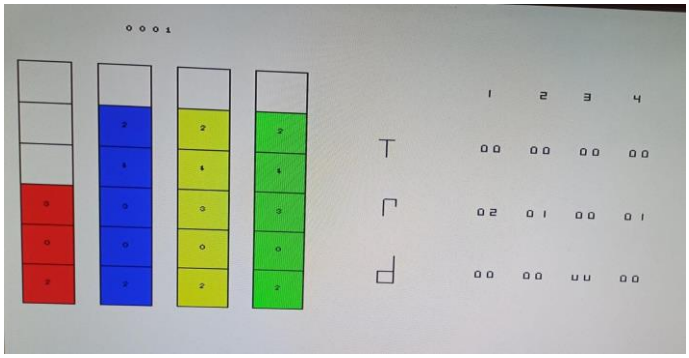

Figure 10. Fifth step of example

When we look Figure 10, output is 0010 as we expected. In this time, latency is satisfied as we can see. Finally, when we look Figure 10, we expect that system will read and delete 1 which is 0001 from Buffer 1, where 00 indicates Buffer 1, and 01 indicates the value 1. When we look figure 6, output is 0001 as we expected.

Again, latency is satisfied as we can see. In addition, as we can see number of transmitted, received and dropped data in right side of figures. While calculating them, we also used if else configurations. After that we pressed them into screen with VGA.

IV. Conclusion

In this project, we successfully implemented a Simple Quality of Service based Queuing in FPGA. Firstly, we wrote a Verilog HDL code that uses the correct first-in-first-out queueing algorithm, then we used a VGA interface via Verilog HDL, which is the part that we learned how to create and use a VGA interface. Finally, we implemented our code on FPGA board by using the buttons on FPGA board and a VGA-in monitor.

We used our knowledge of EE348 course as well as practicing our EE314 laboratory skills such as writing Verilog HDL code. We observed some discrepancies in the simulation results of our code and the VGA interface implementation of the project, which is an expected result, and we decreased these undesired results and flaws as much as possible. As a result, we obtained a proper working system that satisfies the desired results.

References

[1] Dom. “VGA image driver (make a face) on an Intel FPGA,” *YouTube*, Nov. 9, 2020 [Video file]. Available: https://www.youtube.com/watch?v=mR-eo7a4n5Q [Accessed: July 5, 2022].

[2] https://www.pixilart.com/ [Accessed: July 5, 2022].

[3] *DE1-SoC User Manual*, Terasic Technologies Inc., June 11, 2014. Accessed: July 5, 2022. [Online]. Available: http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/DE1-SoC_User_manual.pdf