EE314 Digital Electronics Laboratory Project Final Report:

Design of a Quality of Services Based Queuing

Ahmet Can YULAF
Department of Electrical & Electronics Engineering
Middle East Technical University
Ankara, Turkey
can.yulaf@metu.edu.tr

Ihsan Atakan ADALI
Department of Electrical & Electronics Engineering
Middle East Technical University
Ankara, Turkey
atakan.adali@metu.edu.tr

Mehmet ARSLAN
Department of Electrical & Electronics Engineering
Middle East Technical University
Ankara, Turkey
mehmet.arslan_02@metu.edu.tr

***Abstract*—This report aims to illustrate the design of simple quality of services based queuing. This report consists of design analysis of the project, simulation results, expectations and comparisons between theoretical and practical results.**

I. Introduction

In computer environment, too much data is intended to transfer from one point to another. However, all of the data cannot be transferred simultaneously. Some of them may not be transferred at all depending on the capability of the system. Therefore, an algorithm, which puts the transfer operation in an order, is required. This project focuses on the implementation of this simple quality of services based queuing algorithm on FPGA.

The overall system consists of three parts, namely taking input, sorting the data to be read and displaying the result of this operation on the screen by driving a VGA. There will be a main screen that allows users to trace the rate of data which is given as input, data read, and the data lost. Besides, main screen will enable users to trace which datum is read or lost from which source.

II. Design & Simulation

A. Inputs & Outputs

Table II shows the inputs, outputs and inter-mediate variables of the overall system.

B. Taking Inputs

The system has three buttons to take an allowed input, namely start, button0, button1 buttons. The system demands 4 bit binary input from button0 and button1, when start button is pressed and released. To prevent noise, the system will not allow 4-bit binary input unless the start button is not pressed more than 3 clock cycles. However, 15 clock cycles is a very tiny period for a human-being, in the range of nano seconds. As this manner prevents system from noisy inputs and disturbance, pressing on start button too long is not required.

The start and reset buttons are always high. If start button is pressed, when reset button is high (not pressed), it will turn to low (logical 0). After start button is released, it will be 1 again, which initializes upcoming inputs to the system. After that, using button1 and button0, the desired 4-bit input can be given to the system. In addition, 3 clock cycle rule holds also for button1 and button0.For instance, a user wants to give "1011" as an input to the system. Then, he/she should press start button not necessarily too long. After that, he/she should press button1, button0, button1, button1 respectively.

The start button allows input after returning high. Therefore, the system will not allow inputs when it is low. Moreover, after initializing the upcoming input, the system will take only 4-bit. In other words, if input buttons are pressed more than 4, the rest from 4-bit are ignored. Fig. 1 shows the how the allowed input is given to the system.

C. Queuing Algorithm

After taking inputs and storing them on buffers, reading operation is needed. However, there are some specifications, which will lead reading process accordingly. Those are reliability, and latency. It is desired that buffer1 has the lowest latency and buffer4 has the highest reliability. In Table I, it can be seen the weights of the data (created manually) stored in the buffer, which determines the precedence of data reading process. According to the Table I, if all the buffers have 6 data, the oldest data in the 4th buffer will be read.

To store data in the buffers without changing the priority when reading operation occurs, shifting operation is needed. That is, when reading occurs, read data(the oldest data in the buffer) will be deleted and the rest of the data will be shifted to the reading direction. Shifting operation is needed also for dropping operation. That is, if an input, whose data will be stored in a buffer, whose storage is already full, is given, the oldest data is deleted and the remaining data is shifted to the reading direction. Fig. 2 illustrates how the queuing algorithm works.

TABLE I. Weight Table of the Data Stored in the Buffers

Counter 1,2,3,4	Buffer 1	Buffer 2	Buffer 3	Buffer 4
0	0	0	0	0
1	9	6	3	1
2	13	8	5	2
3	15	12	7	4
4	16	14	11	10
5	17	18	19	22
6	19	21	23	24

TABLE II. INPUTS & OUTPUTS & INTERMEDIATE VARIABLES OF THE OVERALL SYSTEM (I: INPUT O: OUTPUT IV: INTER-MEDIATE VARIABLE)

Name	I/O/*IV*	Definition	Name	I/O/*IV*	Definition
clk	I	It is a 50 MHz internal clock signal of the FPGA. All the remaining inputs and outputs are latched on the positive edge of the "clk" signal.	[1:0]buffer4[5:0]	IV	The inputs"10xx" is stored in this variable. The data is shifted when a new input is given.
reset	I	The system is restarted when it is pressed and released more than 3 clock cycles.	clkk	O	It is a 25 MHz clock signal connected to the VGA. This signal is created by dividing "clk" signal.
start	I	button0 and button1 are activated, when it is pressed and released more than 3 clock cycles.	o_hsync[9:0]	O	It is the horizontal axis coordinate of the pixels of 640x480 screen. For instance a pixel having coordinates P(x,395) will be adressed with o_hsync ="0110001011"
button0	I	It enables system to take one "0" bit input, when it is pressed and released more than 3 clock cycles.	o_vsync[9:0]	O	It is the vertical axis coordinate of the pixels of 640x480 screen.. For instance a pixel having coordinates P(x,228) will be adressed with o_hsync ="0011100100"
button1	I	It enables system to take one "0" bit input, when it is pressed and released more than three clock cycles.	r[7:0]	O	It is the brightness of red colour of the adressed RGB LED. The most brightest option is "11111111", and the lowest option is "00000000".
[1:0]buffer1[5:0]	IV	The data given as "00xx" is stored in this variable. The data is shifted when a new input is given.	g[7:0]	O	It is brightness of green colour of the adressed RGB LED. The most brightest option is "11111111", and the lowest option is "00000000".
[1:0]buffer2[5:0]	IV	The data given as "01xx" is stored in this variable. The data is shifted when a new input is given.	b[7:0]	O	It is brightness of blue colour of the adressed RGB LED. The most brightest option is "11111111", and the lowest option is "00000000"...
[1:0]buffer3[5:0]	IV	The inputs"10xx" is stored in this variable. The data is shifted when a new input is given.	ct1(2,3,4)[2:0]	IV	It counts the number of data present in the bufffers . (e.g. ct2 counts the number of data in buffer2)
ct_dropped_buffer1(2,3,4)[9:0]	IV	It counts the number of data dropped from buffers (e.g. ct_dropped_buffer2 counts the number of data dropped from buffer2)	ct_transmitted_buffer1(2,3,4)[9:0]	IV	It counts the number of data dropped from buffers (e.g. ct_transmitted_buffer1 counts the number of data dropped from buffer2)
ct_received_buffer1(2,3,4)[9:0]	IV	It counts the number of data received buffers (e.g. ct_received_buffer2 counts the number of data received from buffer2)	time_to_read	IV	It is "read" signal, which initializes read operaiton will be done in the next clock cycle.

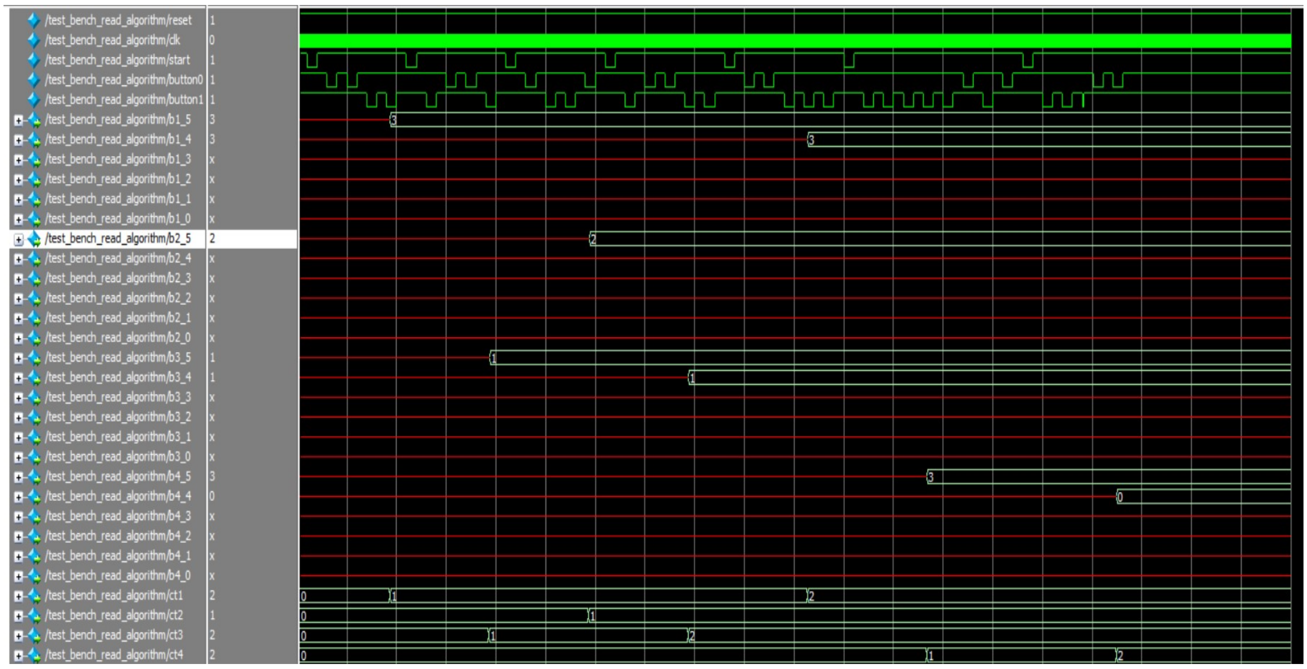

Figure 1: Input combinations and data placement waveform

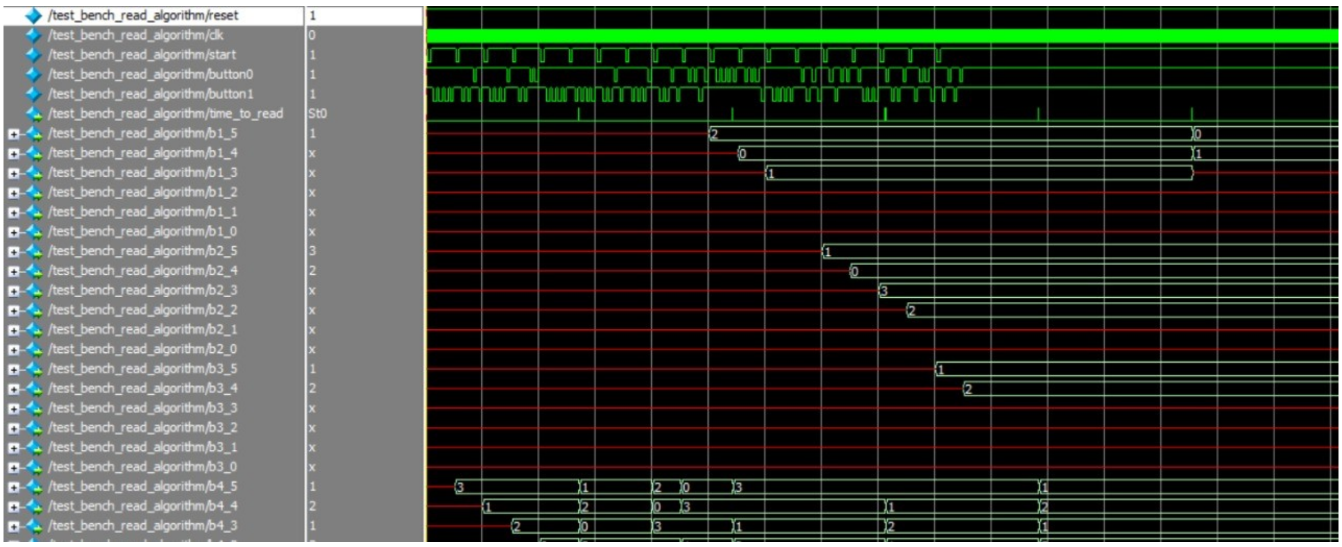
/test_bench_read_algorithm/reset
/test_bench_read_algorithm/clk
/test_bench_read_algorithm/start
/test_bench_read_algorithm/button0
/test_bench_read_algorithm/button1
/test_bench_read_algorithm/time_to_read
/test_bench_read_algorithm/b1_5
/test_bench_read_algorithm/b1_4
/test_bench_read_algorithm/b1_3
/test_bench_read_algorithm/b1_2
/test_bench_read_algorithm/b1_1
/test_bench_read_algorithm/b1_0
/test_bench_read_algorithm/b2_5
/test_bench_read_algorithm/b2_4
/test_bench_read_algorithm/b2_3
/test_bench_read_algorithm/b2_2
/test_bench_read_algorithm/b2_1
/test_bench_read_algorithm/b2_0
/test_bench_read_algorithm/b3_5
/test_bench_read_algorithm/b3_4
/test_bench_read_algorithm/b3_3
/test_bench_read_algorithm/b3_2
/test_bench_read_algorithm/b3_1
/test_bench_read_algorithm/b3_0
/test_bench_read_algorithm/b4_5
/test_bench_read_algorithm/b4_4
/test_bench_read_algorithm/b4_3

Figure 2: The reading and dropping operation illustration waveform

In general, queuing sub-system takes the data stored in the buffers and counter variables as input. Then, it processes variables and the data, and gives coordinates of pixels and the colours of RGB as output.

D. Driving VGA

After queuing process, the system has coordinates of pixels and colours. Then, displaying process is required. It will achieved via VGA port of FPGA. VGA takes coordinate of pixels (o_hsync[9:0], o_vsync[9:0]),clock signal (clkk) and colours of RGB's (r[7:0], b[7:0], g[7:0]) as input and prints the overall output, which is the main screen. clkk signal should have a frequency of 800*525*60 = $2.52*10^7$ Hertz because every bit of screen should printed 60 times in a second. However, the clkk is chosen 25 MHz as its effect is negligible. The 25 MHz clkk signal is achieved by dividing clk signal, which is 50 MHz internal clock signal. However, the desired main screen has more specifications. That is it will print the data stored in the buffers and transmission rates, which means main screen will contain numbers and texts. Moreover, buffers in the main screen will have different colours: red for buffer 1, blue for buffer 2, yellow for buffer 3, and green for buffer 4.

To achieve the desired main screen all texts and counters should be coded. 10 numbers, which are inspired by 7-segment display HEX, and 4 letters are coded. Besides, for colours other than red, green, and blue, combination of them are used. For instance, red, green, and blue signals are high for white, or they are low for black. Fig. 3 and Fig. 4 show the output of the overall system, which is the main screen.

III. Comparison Between Simulation and Experimental Results

All results are expected. There is no discrepancy between simulations and experimental results. What is desired, is achieved.

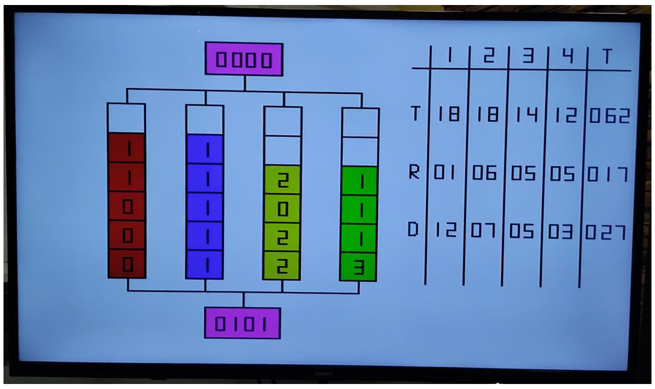

Figure 3: The main screen, the output of the overall project

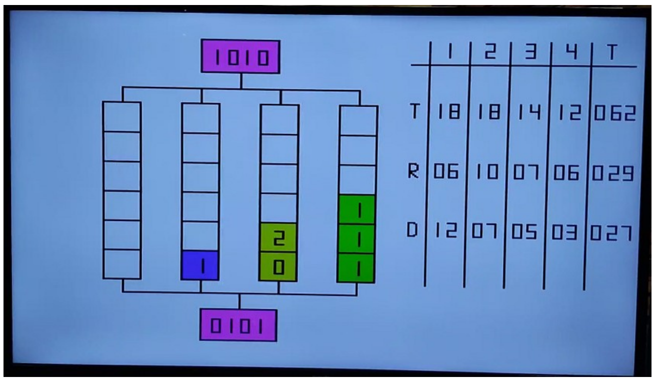

Figure 4: The main screen, the output of the overall project

IV. Conclusion

This document was created to illustrate how an QoS based queuing algorithm can be developed by using FPGA. It has been mentioned that which steps should be passed, namely taking inputs, processing them via queuing algorithm, and driving VGA to show the data stored and data flow ratings. At the end, it is observed that there is no discrepancy between theoretical results and experimental results. Simple Quality of Services based Queuing algorithm works properly.