EE314 Term Project Report Implementing a Quality of Service Network on FPGA

Çağrı Arı
Electrical and Electronics Eng. Dept.
Middle East Technical University
Ankara, Turkey
2303956

Semanur Avşar
Electrical and Electronics Eng. Dept.
Middle East Technical University
Ankara, Turkey
2442481

Beyza Akın
Electrical and Electronics Eng. Dept.
Middle East Technical University
Ankara, Turkey
2303881

***Abstract*—The amount of data that is transferred between users exponentially grows day by day, therefore the quality of service algorithms becomes more essential. In the scope of the digital circuits laboratory course, we have realized a very simple QoS network using FPGA, VGA interface, and Verilog HDL as the term project. In addition to aesthetic design requirements which are related to VGA generally, some important concepts in communication like reliability, and latency have also been considered. By designing appropriate queuing algorithms, the predefined latency and reliability requirements have been tried to be satisfied. Working on such a project and producing an end product made us realize the importance of the data queuing systems, their efficiency, and robustness.**

***Index Terms*—FPGA, Quality of Service, VGA, Verilog HDL, Logic Design**

I. Introduction

QUALITY of service (QoS) networks control the data traffic between the users. Such systems are important and necessary since the importance of different data is not the same. While the loss of some types of data is acceptable, the loss of other types may be a vital error. Therefore, the data traffic in any communication system should be controlled so that reliability and efficiency are increased. As the term project of the course EE314, we are required to design a basic QoS network. This network arranges the data traffic between two different nodes. There is one-way communication between the nodes. In other words, one of the nodes is the transmitter node, and the other is the receiver node. Moreover, there are four different channels through which the transmitter node can send data to the receiver node. The designed QoS network should also satisfy the predefined latency and reliability requirements of each channel and the overall reliability requirement of the system. We are expected to realize such a system using Verilog HDL, an FPGA board, and VGA interface. The user manual of the FPGA board we used can be found in [1]. In the coming parts of the report, the general problem and all predefined requirements are introduced. Furthermore, the design solutions corresponding to each requirement and general problem are explained in detail.

II. Problem Definition

There are three main parts of the system designed in this project. Every part has some special requirements. In this part of the report, those requirements are introduced. The terms channel and buffer have been considered synonyms throughout this report.

The first part of the system is receiving the input packets. The transmitter node, which can be called the user, should be able to send input packet at any time by using three buttons which are the start button, the logic high button, and the logic low button. The collected inputs are stored in four different channels. Each channel has six different memory boxes, and only one packet can be stored in each box. Whenever a channel is full, the oldest data is dropped and the new data is collected. In other words, the oldest information (packet) is lost. This is defined in the project as a packet drop.

The second part is reading the data from the channels. Every three seconds, a single data is read. The reading operation should be performed such that the requirements introduced at the following subsections are satisfied.

A. Latency Requirement

$$Channel1 > Channel2 > Channel3 > Channel4$$

The latency is the time delay between the reception time and transmission time of the data packet. Channel 1 should have the smallest latency, i.e, the fastest communication is desired for this channel. Moreover, the latency of each channel should be smaller compared to the higher-ordered channels.

B. Reliability requirement

$$Channel4 > Channel3 > Channel2 > Channel1$$

The reliability means the less number of packet drops here. Channel 4 should have the highest reliability, i.e, the number of packet drops from channel 4 should be minimized. Moreover, each channel should be more reliable compared to the smaller-ordered channels.

In addition to those two requirements, the general priority of the system should be minimizing the number of packet drops because the information loss is irreversible and the most undesired case for a communication network. To satisfy all the above-mentioned requirements, a queuing algorithm should be designed and implemented.

The final part of the project is creating a screen using the VGA interface which displays all required information. The number of received, transmitted, and dropped packets should be displayed on the screen as well as the momentary content of each channel. Also, the filled memory boxes of each channel should be colored appropriately. The colors corresponding to each channel are given in Table I.

TABLE I
CHANNEL COLORS

Channel Number	Color
1	Red
2	Blue
3	Yellow
4	Green

III. Module Operations

In this section, the module operations will be discussed. Block diagrams for each module will be shared and the relation between input and output ports will be explained. Finally, the overall module connections and their simulation results will be shared.

A. Input Collection Module

This module is responsible from collecting the 4 bit data sequence from the user. For this purpose, three push buttons of the FPGA board are used. We assigned one push button for start, input 1 and input 0 actions. The module should start collecting the data after the start button is pressed. Then, it should form the sequence according to the input 1 or 0 values. After the start button is pressed, only the first four data bits should be collected. Then start button should be waited to start forming another group of data. To realize this, we simply use a counter that initializes to 1 when the start button is pressed and gets increased by 1 every time an input is received. We only collect the input if the counter value is less than or equal to 4. Output of this unit is one four bit data sequence and a one bit output expressing whether the data is completed or not. One thing to pay attention is that the push buttons of the FPGA are active low, meaning their value is 0 whenever they are pressed. The algorithm for this module is given in Algorithm 1.

Algorithm 1 Algorithm for Input Collection Unit

```
if start button is pressed then
    counter=1
end if
if input button is pressed and counter==1,2,3,4 then
    $data\_output = \{data\_output, input\_data\}$
    $counter = counter + 1$
end if
if input button is pressed and counter==4 then
    $data\_completed = 1$
else
    $data\_completed = 0$
end if
```

One problem with using push buttons is that the inside clock of the FPGA is 50MHz, therefore, even one push coincides with multiple clock pulses. Hence any input push is realized multiple times. To prevent this from happening, we added a debouncing code that eliminates consecutive inputs from the push buttons. The idea is that if we count how many consecutive times a button is pressed, any value for the counter, say 2, happens only once for each consecutive input. Hence rather than using the information of whether the push button is pressed or not we use if the counter of that input has reached 2 or not in our remaining code.

The block diagram of this unit is given in Fig. 1

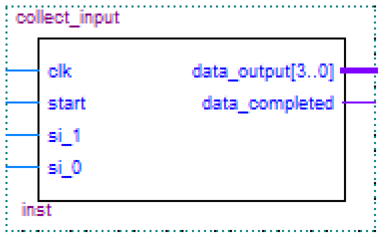


Fig. 1. Block diagram of the input collection module showing input and output ports

B. Buffer Distribution Module

This module is responsible from the buffer contents. For this purpose, one register for each buffer is used. Buffer registers are 18 bit long where every third bit is the data indicator bit and the following two bits are the data itself. If the data indicator bit is 0 that means there is not a data at that part of the buffer. The module should update the content of the buffers whenever a data is received, transmitted or dropped. These three operations will be examined one by one.

1) Data Receive: This operation uses the output of the input collection module. From this module, it receives the information of whether a data has been completed or not, and if ,there is, the corresponding data sequence. When a data is completed, its content should be added to the fist available slot of the buffer indicated by its first two bits. To realize this, first two bits of the data is checked and the buffer corresponding to that sequence is decided. Then, the first available slot is found using the data indicator bits. The data value is loaded to that part of the register and the corresponding data indicator bit is updated to 1 to show that there is a data there.

2) Data Drop: When a data is completed, if there is no available slot in the buffer i.e. all of the data indicators are 1 the oldest data is dropped and the remaining content is shifted by three bits. Also, the new data is written to the open slot with making its data indicator bit 1.

3) Data Transmit: This operation uses the output of the reading data module. From this input, it receives the information of whether there should be a data read or not, and if there should, which buffer it should be read from. If there should been a data read, it finds the corresponding buffer, shifts every entry of that buffer by three bits making the oldest data overwritten and makes the data indicator bit of the last slot 0. This can seem similar to the data drop operation with two differences:

1) It stores the read output data along with which buffer it is read from (similar to how the data was received in the first place) in a register called $packet\_read$ and passes this information to VGA module to print on the screen.
2) At the end of this operation there is at least one empty slot in the buffer. When we shift the entries of the buffer by three bits, we also shift the data indicator bit. Hence, we pass the information of whether that slot should be filled or not compatible with our goal. And by making the data indicator bit of the last slot 0 we make sure that it is empty.

Another task of this unit is to count the number of dropped, transmitted and received data packets for each buffer separately to be printed at the VGA screen. To achieve this, rather than counting the number itself, it creates a pulse for one clock cycle and passes this as an input to the VGA unit where it will be use to update the count numbers on the screen. To achieve this, it makes the corresponding counter pulse value one whenever it performs that action and whenever it does not, the value is immediately made back to 0. Such counter values are present for every buffer and every action making $4 * 3 = 12$ output ports for this task.

Also, this module provides the buffer registers as outputs to be used in the reading data and VGA modules. The block diagram summarizing the input and output ports is given in Fig. 2

C. Reading Data Module

This module shown in Fig. 3 is responsible for producing a 3-bit sequence as an input to the buffer distribution module. This 3-bit sequence determines the buffer from which the next packet will be read. The module has five possible states as shown in Fig. 4. The system is in state 0 initially and whenever all the channels are empty. The other states represent the channels. For example, when the algorithm determines that the next packet should be read from channel 3, the system goes into state 3.

The module is constructed according to a simple algorithm that we designed so that the QoS network satisfies the latency

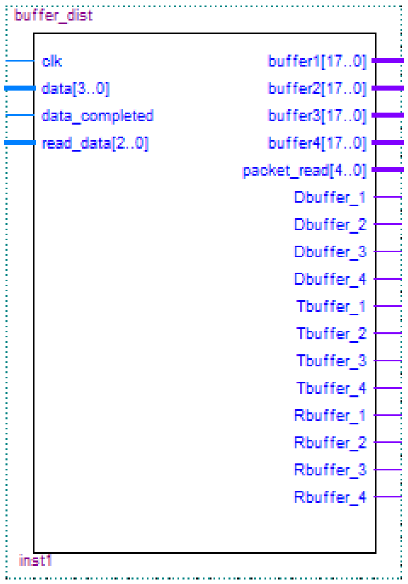


Fig. 2. Block diagram of the buffer distribution module showing input and output ports

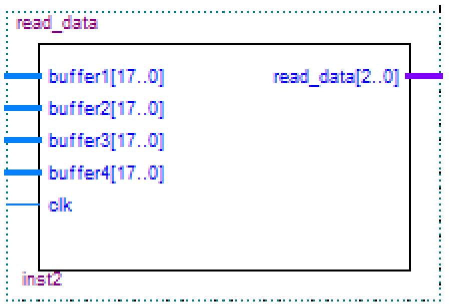


Fig. 3. Block diagram of the reading data module showing input and output ports

and reliability requirements. The algorithm has two major behaviors. The first behavior of the algorithm focuses on minimizing the number of packet drops. As a result, whenever a channel is full, the algorithm prioritizes that channel. However, as known, channel 4 should be the most reliable. To achieve that, the algorithm selects the highest-ordered channel whenever more than one channel is full. For example, if channels 2 and 4 are full simultaneously, the algorithm selects channel 4 to read data from. This feature makes the reliability increase from channel 1 to channel 4. The second major behavior of the algorithm focuses on latency. If no channel is full, the algorithm prioritizes the lowest-ordered channel so that the latency reduces from channel 4 to channel 1. This feature makes channel 1 possess the lowest latency. The algorithm determines and prioritizes the most convenient channel at each clock pulse. In other words, the system reevaluate the most convenient channel to read data at every clock pulse. Whenever a read-data clock which is once every 3 seconds hits, the data packet corresponding to the most recent state of the reading data module is read.

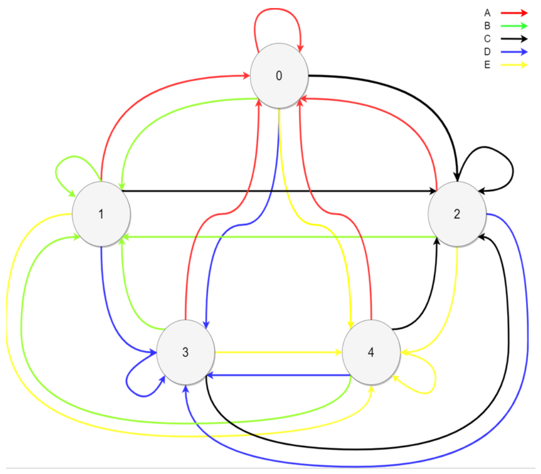
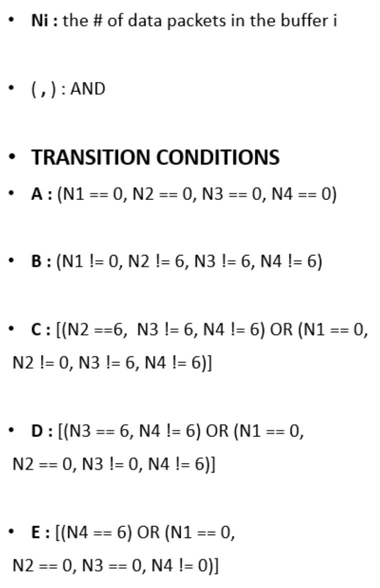


Fig. 4. State diagram of the reading data module

The module produces different outputs depending on the recent state. If the system is in state 0, which means there is no data packet to read, the first bit of the input sequence becomes logic low. This bit operates like an enable signal. When it is logic low, it means there will be no read operation. Otherwise, the module produces the 3-bit output sequence such that the first bit is logic high and the last two bits points out the channel from which the data will be read. For example, the output sequence is 110 if the system is in state 2, 101 is the system is in state 1, etc.



- **Ni :** the # of data packets in the buffer i
- (,) : AND
- **TRANSITION CONDITIONS**
- **A :** (N1 == 0, N2 == 0, N3 == 0, N4 == 0)
- **B :** (N1 != 0, N2 != 6, N3 != 6, N4 != 6)
- **C :** [(N2 ==6, N3 != 6, N4 != 6) OR (N1 == 0, N2 != 0, N3 != 6, N4 != 6)]
- **D :** [(N3 == 6, N4 != 6) OR (N1 == 0, N2 == 0, N3 != 0, N4 != 6)]
- **E :** [(N4 == 6) OR (N1 == 0, N2 == 0, N3 == 0, N4 != 0)]

Fig. 5. Transition conditions for the state diagram given in Figure 4

Fig. 5 shows the state transition conditions of Fig. 4. As can be seen, the algorithm uses the number of data packets that each channel has to make decisions.

D. VGA Module

Video graphics array (VGA) is commonly used for video transmission applications. The color values for the pixels on the screen with VGA technology are represented by three bits that correspond to RGB (Red, Green, and Blue). Also, VGA cables normally include five key pins that correspond to R, G, B, hsync, and vsync. In this project, we used 640x480 resolution frame which implies that there are 480 lines, and each one has 640 pixels. Each frame begins to display from the first line and continues from left to right. There are two synchronization signals that control the construction of a frame: hsync (horizontal synchronization) and vsync (vertical synchronization). Some pixels in hsync and vsync signals are used for display, while others generate blanking time determined by the front porch and back porch around sync pulse. Moreover, VGA requires a 25 MHz clock, whereas our FGPA board has an internal clock of 50 MHz. We used a frequency divider code to generate a 25 MHz clock. In order to learn more about VGA we examined the github project in [2].

We have designed the VGA Module of our project in 3 different sections that are static content, buffers, and counter values.

1) Static Contents: A considerable amount of pixels on the screen do not change in time. These pixels form the letters and the border lines of the buffers. In the VGA module, an if-block is used to control these pixels. The condition of the if-block is written so that whenever a pixel that is a part of a static entity is updated, the if-block runs and paints the pixel black. There are no additional structures like switch cases in this part of the code because these pixels are static. However, the if-block has been modified in the final version of the system such that the static entities change color every second.

The letters and numbers are constructed by considering the formats given in Fig. 6 and Fig. 7. We simply colored pixels in the screen using with these formats whenever necessary.

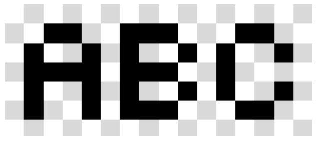


Fig. 6. The letter format

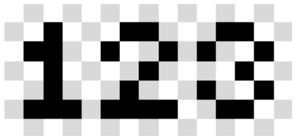


Fig. 7. The number format

2) Buffers: As shown in Fig. 8, the contents of all the buffers are input to the VGA module. While the screen is continuously updated, the buffer contents are also updated. In the code, there are sub-code blocks for each memory box of each buffer. As mentioned earlier, the buffers consist of 18 bits, and six of those bits are there to show whether each memory box is empty or full. When these indicator bits are logic low, the sub-code blocks paint the corresponding memory boxes white. However, if indicator bits are logic high, then the code runs the switch case structures inside the memory boxes' sub-code blocks. That switch case structures have four different cases depending on the data, which are either 0, 1, 2, or 3. The data is held in the remaining 12 bits of the buffers. Whenever an indicator bit is logic high, the corresponding case of the switch case structure runs, for example, the second case runs if the memory box carries the data '2'. Then, it draws the decimal equivalent of the data packet inside the memory box and paints the remaining pixels corresponding color.

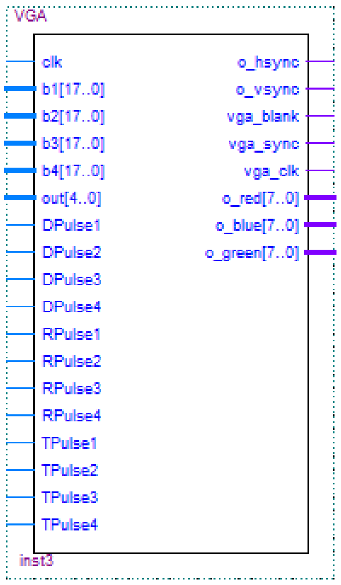


Fig. 8. Block diagram of the VGA module showing input and output ports

3) Counter Values: The number of received, transmitted, and dropped packets is counted in the VGA module so that the numbers can be displayed on the screen. To achieve that, we created 12 pulse signals in the buffer distribution module, and give them inputs to the VGA module as shown in Fig. 8. Any of these pulses trigger the corresponding number to increase. For example, TPulse2 goes high for a single clock whenever a packet is transmitted from channel 2. DPulse3 goes high for a single clock pulse whenever a packet drops from channel 3. There are 45 integers defined in the VGA module. Each integer corresponds to a single decimal digit of the packet counts. Each count information, the number of received, transmitted, and dropped packets, has three decimal digits, i.e, the range is from 0 to 999. Those pulse signals trigger the least significant decimal digit of the corresponding count to increase by 1. Whenever the least significant bit reaches 9, the next pulse pulls it to 0 and increases the middle decimal digit by 1. The same procedure is repeated for every decimal digit of every count operation. However, in the total number of received, transmitted, and dropped packet cases, there is a small difference. The decimal digits of these counts are triggered by all RPulse, TPulse, and DPulse signals. For example, the total number of dropped packets is triggered by any DPulse signal and increased by 1.

For every defined integer variable, there is a switch-case block in the VGA module. These blocks have 10 different cases corresponding to the value of the variables, which is limited between 0 and 9 in design. Each decimal digit of each count number has its own switch case and is written on the screen by that switch case. The structure is very simple. For example, if the value of the digit is 7, then the corresponding case of the switch case structure runs and writes 7 to the part of the screen dedicated to that digit.

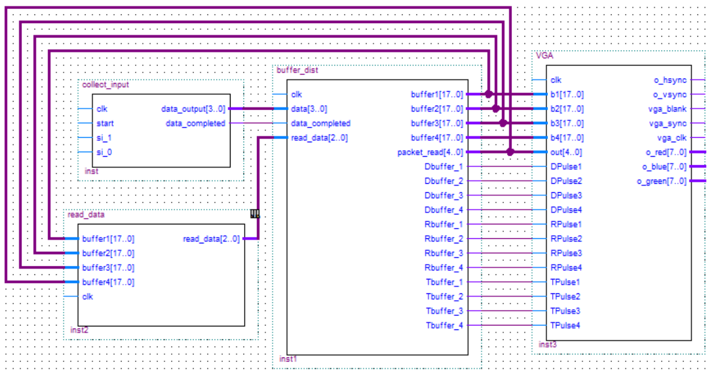


Fig. 9. Overall block diagram showing connections between the modules

IV. Overall Structure and Demonstration Photos

In this section, we share overall structure of our code and demonstration photos showing our VGA screen. Fig. 9 shows the connections between individual modules. Our overall structure has input ports for clock, push buttons and provides output ports for the VGA screen. Also, Fig. 10 shows the simulation photos of our first 3 modules, before integrating them with the VGA module. The simulation consists of a number of input trials, the corresponding actions taken from the read data module and the buffer content updates made by the buffer distribution module. Fig. 11 shows our final VGA screen. We have buffers on the left side of the screen along with read data at that moment. Right side of the screen shows number of received, transmitted and dropped data respectively.

V. Conclusion

In conclusion, implementation of a basic Quality of Service network on FPGA is discussed. Four modules designed for this purpose are explained. The overall relation between these modules and the final VGA screen is shared. The remarks of this project are listed below:

- Latency and reliability are two important Quality of Service parameters. Some packets may need highest timeliness whereas others may need lowest information lost. An algorithm that decides which data to prioritize when transmitting should be designed according to these specifications.

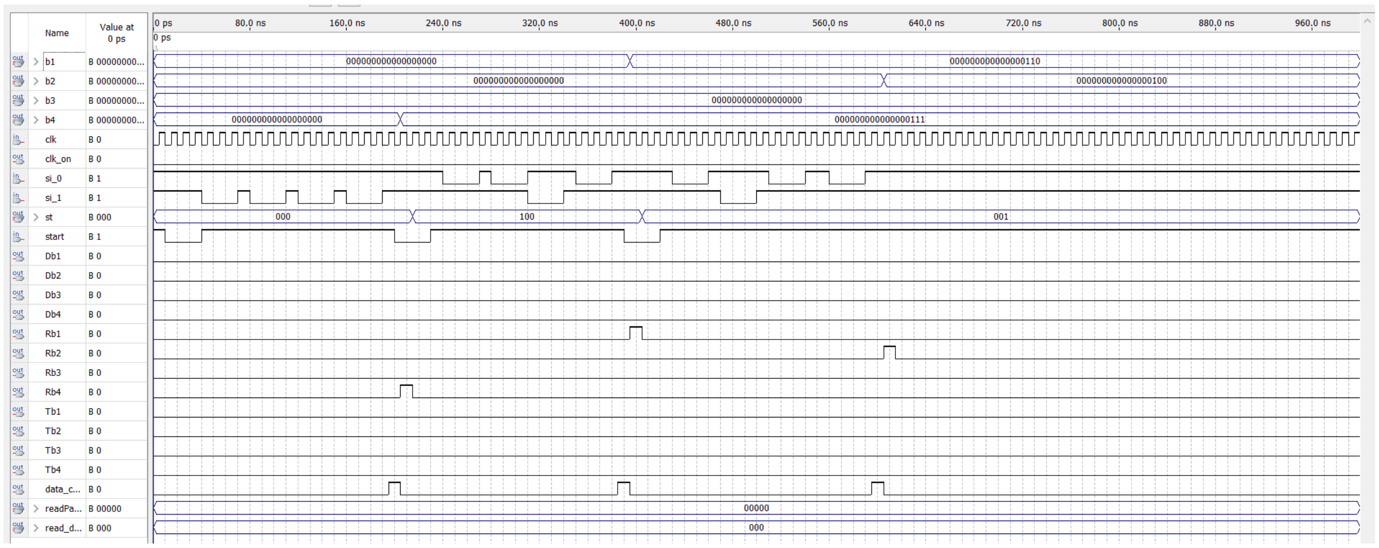


Fig. 10. Simulation results of the overall structure

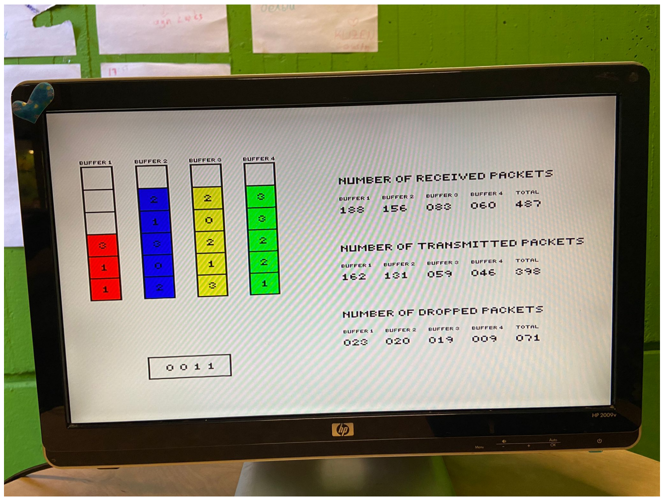

NUMBER OF RECEIVED PACKETS
NUMBER OF TRANSMITTED PACKETS
NUMBER OF DROPPED PACKETS
0 0 1 1

Fig. 11. Demonstration Photo

- When using the push buttons of the FPGA board, the debouncing problem has to be considered. Debouncing occurs due to the high clock rate of the FPGA. One can solve this problem using a counter for each input.
- When representing buffers as registers, one has to consider whether that part of the buffer is empty or not. Hence even if there are four possible data values for each slot of the buffers, 3 bit space is necessary for every data.
- VGA is a video transmission application where every bit is represented by an RGB value. Controlling an FPGA screen is tricky because one has to plan the overall image pixel by pixel.

REFERENCES

[1] Terasic Technologies(2014) De1-Soc: User Manual
[2] (n.d.). Retrieved from https://github.com/dominic-meads/Quartus-Projects