Quartus Assignment 1

Steven Van der Werf

Introduction

The purpose of this exercise is to take a 4-bit input and return the appropriate 1-bit response. This is to be built in Verilog HDL using the Quartus development platform. While not especially complex, this will serve as a reminder as to the functions of Quartus and the building of logical commands.

Design

We begin with a conversion of the logic involved into Verilog code, and a breakdown of the logic into checkable statements

F=A.B.(C+D) translates to Verilog as: F <= A & B & (C | D) (1)

G=(A+B).C.D translates as: G<=(A | B) & C & D (2)

These statements break down to:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **F** | **G** |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |

Figure 1: Table of results

Where all other combinations return 0 for F and G.

I originally attempted to code this as a set of case statements based on Figure 1 but this turned out to be very inefficient and somewhat buggy

The solution presented by (1) and (2) is more elegant and removes some probability of basic coding errors. The resulting code is a near direct translation of logic, but by default settings produced glitches in the vector waveform that appeared to be false positives [Figure 2].

This issue was caused by propagation delays, and solved by increasing the count state from 1ns to 50ns, as shown in Fig 3. The resulting waveform exactly matches expected results as detailed in Figure 1 above.

Discussion & Conclusion

Verilog is clearly a highly powerful and incredibly useful tool, which can be used to solve hardware issues in a wide variety of ways. In this case, simple logical statements are ideal, but case statements or a graphic construction could have provided adequate solutions.

It is essential to be aware of the accuracy of simulation, which in this project was initially mistaken for a coding error, which cost time and effort to ‘correct’. This is where careful iteration of code becomes essential, as is the habit of commenting – both to provide commentary and to deactivate lines of code rather than deleting them.

Of further interest is the range of Netlist Viewers, which offer component maps of hardware coded. Examples are included in the Appendix section, including RTL [Fig. 4] and Multiplex [Fig. 5]

The primary concern, however, has been the instability of Quartus itself. It is essential to have at least two HW PCs running the software at any given time, since at least one of them is likely to fail at any given time. This makes even coding an individual project an effective team effort. The code detailed in the Appendix section was written at home after Quartus failed to function at HW.

Appendix

module assess1(A,B,C,D, F, G); // define module, nodes

input A, B, C, D; // individual inputs

output F, G; // and outputs

reg F,G; // outputs must be held in memory, inputs are simply read

always @(A,B,C,D) // act on any change of input

begin // on any input, do the following calculations

F <= A & B & (C | D); // A and B and (C or D)

G <= (A | B) & C & D; // (A or B) and C and D

End // and display results. Repeat.

endmodule

---

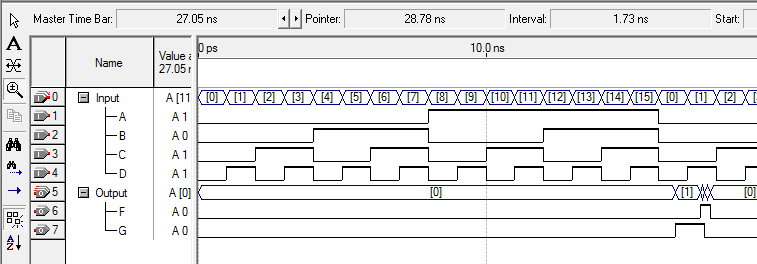


Figure 2: first waveform at 1ns with errors

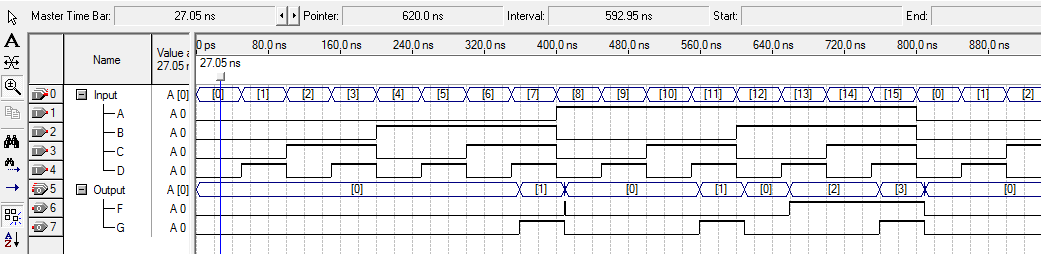


Figure 3: adjusted waveform at 50ns, no errors

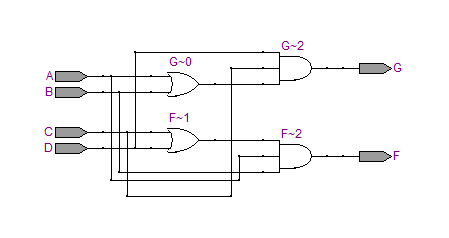


Figure 4: RTL view of the Verilog code

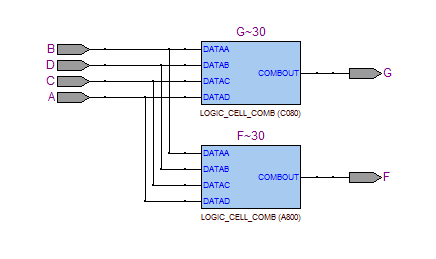


Figure 5: Mux view

As can be seen below, progress was hindered by Quartus repeatedly crashing, freezing and failing to launch.

