Lisa Jacklin

12/11/2022

CS 301

Project technical post

Four-Bit Multiplier Circuit, From VHDL to Assembly

My goal with building a four bit multiplier circuit in assembly, was not only to understand how this circuit works, but also to see how VHDL compares to assembly, and in turn through the translation process, how VHDL compares to C++.

Diagram, schematic

Description automatically generated

*Figure 1: four bit multiplier circuit schematic.*

This circuit, consisting of multiples of full adder circuits, and additional subcircuits that combined, build this complex four bit machine, which in VHDL, could compute the multiplied total from the given bits, 0000 through 1111. Through VHDL, the product of these multiplied values were also printed out giving an answer ranging from zero, to 225, but in binary of course.

Diagram, engineering drawing

Description automatically generated

*Figure 2: subcircuits contained within the four-bit multiplier circuit. Each of the boxes displayed in the full multiplier circuit schematic contains one of these two subcircuits.*

Diagram

Description automatically generated

*Figure 3: Full adder circuit, and the smallest circuit form contained in the full four-bit multiplier. Note that each FA contained in the subcircuits, is a block of this full adder circuit.*

In converting the code to assembly as well as C++, the initial full adder circuits worked as expected, taking in any one bit value and giving the resulting single or double bit value based on if carrying a digit was necessary. When considering the completed subcircuits, these worked without flaw, each was easy to test, and worked well when connected to the full adders, but issues followed from here.

In VHDL, a function called *portmap*, is used to take components made in other files, and add them as a portion of the overall circuits *architecture*. This essentially works as a form of function call that, when called, requires the component that is being called, and the input and output values and destinations.

Text

Description automatically generated

These functions created more issues that I would have realized working through the code in VHDL. In converting to C++ or Assembly, I had a difficult time finding a suitable match that would allow the outputs to lineup in a similar fashion to those displayed in the portmaps. This also lead to the realization that the use of VHDL *signal* functions was another concern when it relates to connecting the circuits in the overall build.



These signal functions, essentially work the same way as a C++ vector, but depending on the logic setup, 8 downto 0, or 6 downto 2 in this case, changes the order in which the vectors are read. This was fairly easy to replicate in assembly and C++, while the connection between these vectors or signals, and the portmaps remained unsolved.

In the future, considerations towards creating templates that could handle the specified output and input vectors, based on the functions listed values, and perhaps even ways to sort a single vector so it could be utilized by all sections of the overall circuit, are potential options to fix the hang up on *portmap* conversion. These issues, although something I thought at the time would be a simple fix, became a bit more of a treacherous task to wrap my head around, while maintaining my limited understanding of how this circuit is built. Had I considered ways to implement this circuit without going about each gate individually, I believe this task would have been feasible in the time allotted, but for now, will remain code to contemplate and adjust.