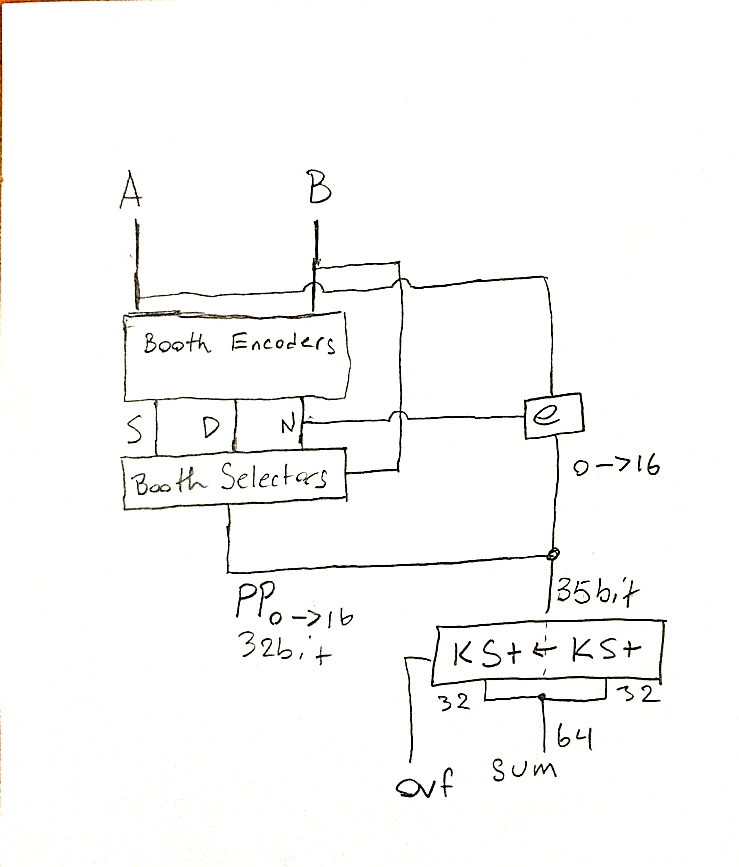
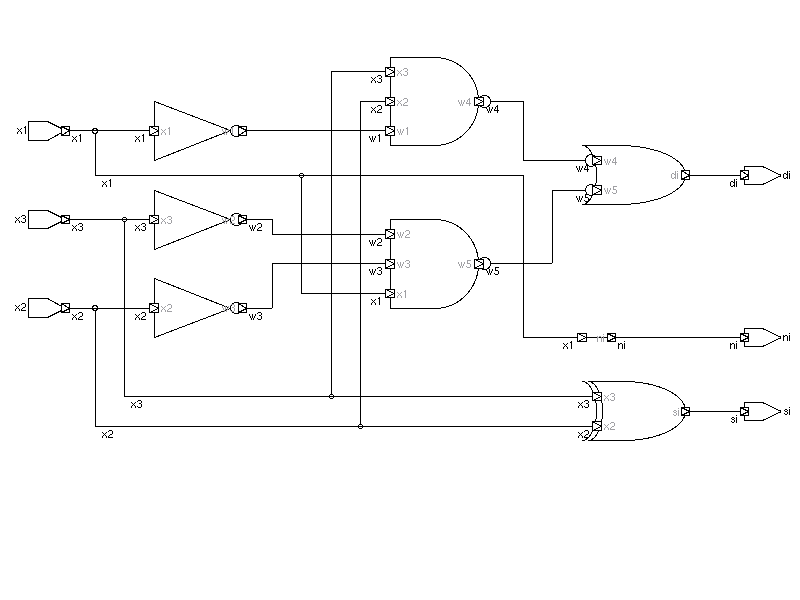
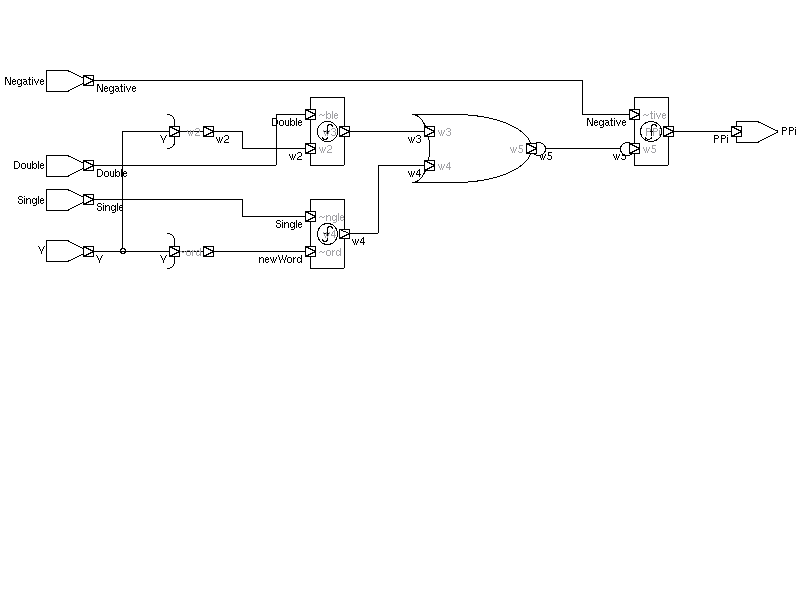
The multiplier module we designed was originally intended to perform signed multiplication. Ultimately the module is only capable of unsigned multiplication, due to time constraints we were unable to successfully debug the module. Similarly, our original intent was to use a compressor tree to sum the partial products of the multiplier. This was also scrapped in favor of a simpler, but much larger, ripple CSA sum using out Kogg-Stone adders. It was constructed as a 32- bit radix 4 booth encoding multiplier. A top-level schematic is as follows:

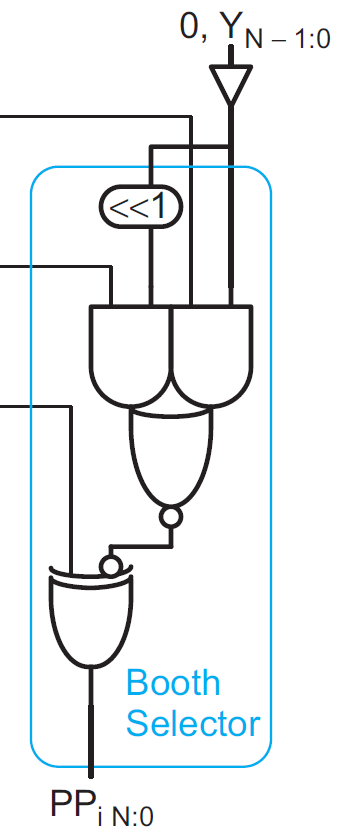
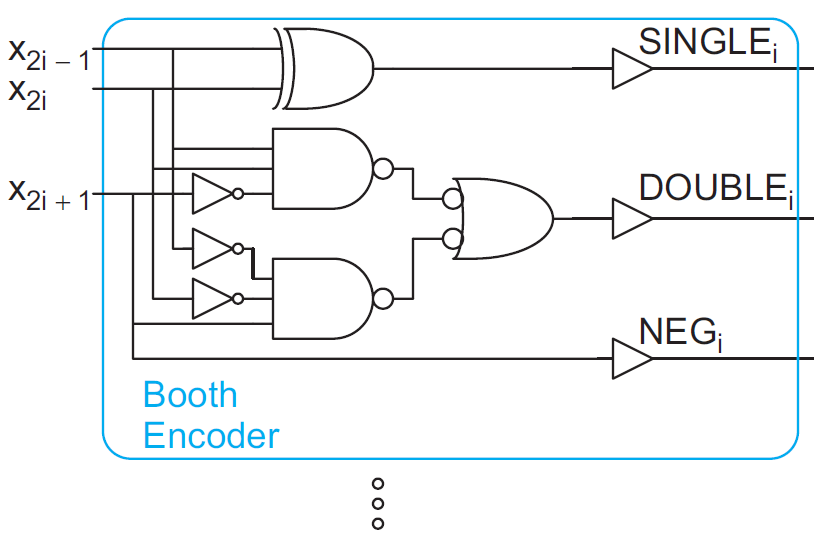


As is plain to see, the e-bits used in signed multiplication are generated despite functionality not being present. The Booth encoders generate control signals based on the multiplier value that allow the Booth selector to output the final partial products. The selectors and encoders were implemented as individual modules, generated iteratively. Their schematics follow.

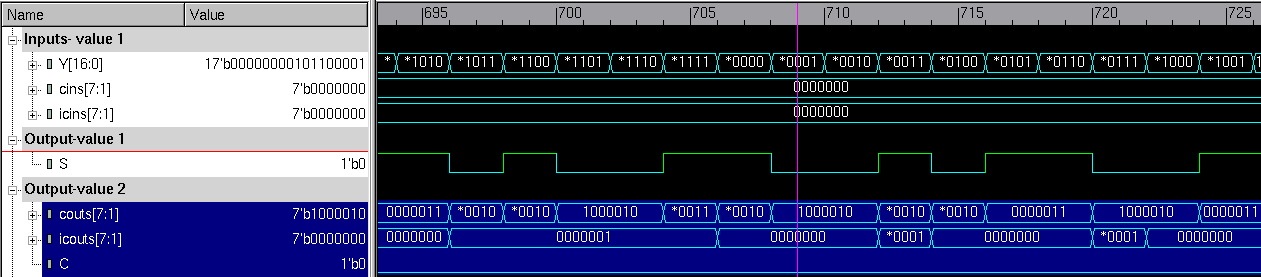




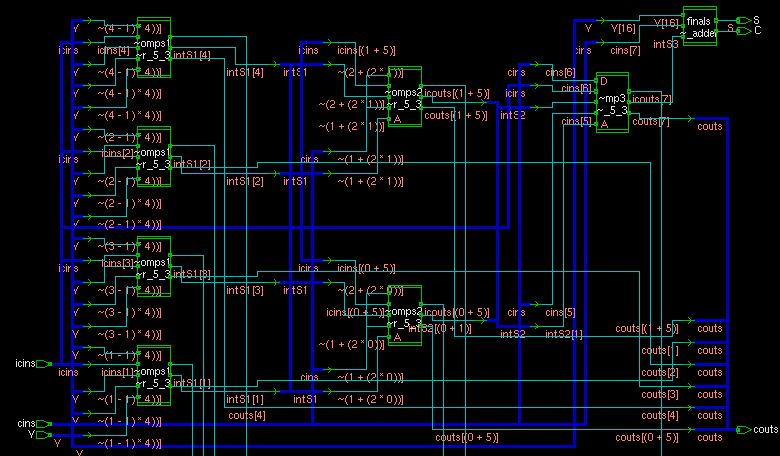
Comparing the generated modules to the book’s diagrams, they clearly have an identical structure:



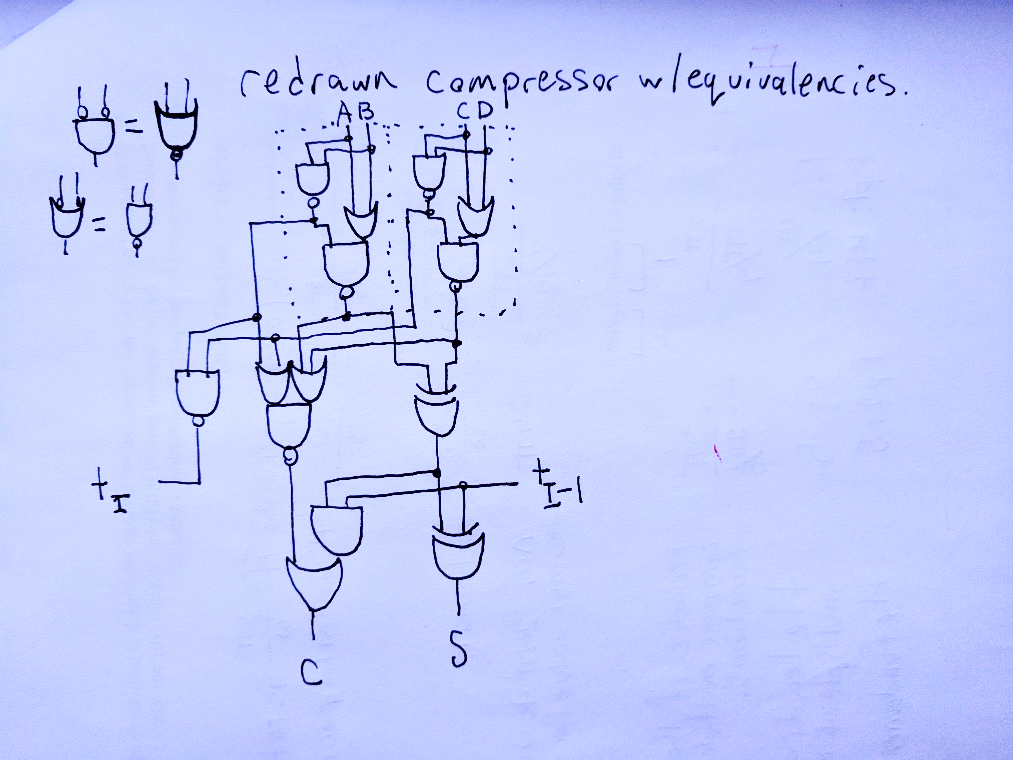
The extended partial products generated from the Booth selectors’ outputs and the e bits are extended to 64 bits, zero-filled in the appropriate locations to preserve the magnitude of the data. The 64-bit signals were originally going to be processed by a set of uniform 5-3 compressor trees, in order to save space and delay compared to the Kogg-Stone ripple network. The “overflow” output from the KS adders pair set will not ever be set, and as such is not utilized in the top level module. The compressor tree was implemented, tested, and verified, but we were unable to utilize it within the multiplier itself successfully. Note in the compressor tree waveform that the sum of the weights of the outputs is equal to the sum of the weights of the inputs in all cases.



See figure X for a schematic of the 5:3 compressor, and figure Y for the compressor tree.



(tree)



(single)

The compressor circuit implemented differs from the book’s implementation. The implemented circuit uses fewer logic gates, avoiding many inversions necessary to implement the circuit depicted in the book. Undoubtedly, with more experience coding SystemVerilog, the book’s implementation would be more efficient, as it takes advantage of alternating parity to reduce the number/size of transistors synthesized. See the appendix for any and all multiplier module and submodule waveforms and data.