**Multiplier**

The multiplier that we decided to use is a Wallace-tree multiplier, which is an efficient hardware implementation of a binary multiplier, consisting of three main steps:

1. Multiply every bit of one operand with each bit of the other
2. Reduce the partial products with full and half adders until the partial products are two bits wide (as shown in figure 1)
3. Add the two remaining numbers via a fast adder like a carry lookahead adder or Kogge-Stone adder

Most of the work done is in step two, as step one only requires a 32x32 array of and gates to multiply every bit of both operands. An example of the partial product reduction during step 2 is shown below in figure 1, although it is only for an 8x8 multiplier

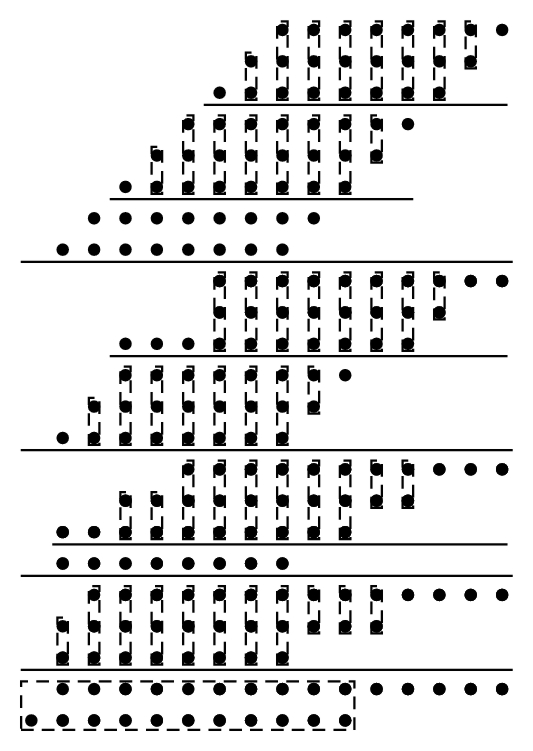


Figure 1: Partial product reduction of an 8x8 Wallace tree multiplier

In the image above, the dotted lines represent either full adders (dotted lines encompassing 3 dots) or half adders (dotted lines encompassing 2 dots). The wide lines divide different stages, while the thin lines divide groups within a stage. By reducing each partial product to be only two bits wide (as shown in the final stage), we can feed those two numbers into an adder (represented by the dotted lines in the final stage). The adders within each stage can work in parallel, allowing for much faster multiplication than in a typical add-shift multiplier, which requires shifting one operand by one bit each cycle. Currently, the implementation of the multiplier executes all stages during one cycle, however this will likely not remain the case for checkpoint 4, as executing all the stages during one cycle requires us to significantly decrease the clock frequency. Experimenting with the number of stages per cycle should yield an optimal number such that we can execute the multiplication in the fewest number of cycles while still hitting our 100 MHz target,

The divider we decided to implement is a simple shift subtract divider, which takes 32 cycles to complete. Because of this, a signal must be added to stall the rest of the stages for 32 cycles while the division is taking place.

Testing on both the multiplier and divider was done both through handwritten assembly code to ensure its functionality with forwarding and the rest of the pipeline, as well as a testbench that fed operands directly into both the multiplier and divider in order to ensure that they could handle a wide range of operands, both positive and negative.