EE290C CORDIC Lab 2

Vighnesh Iyer

October 3, 2018

1 Type Generic CORDIC

I genericified my CORDIC and found that at least **27 stages** were required for the `DspReal` version to achieve 2 LSB precision.

I think my usage of typeclasses was excessive, but I just kept adding what was needed to get the code to compile. Is there a way to alias a conjunction of typeclasses like

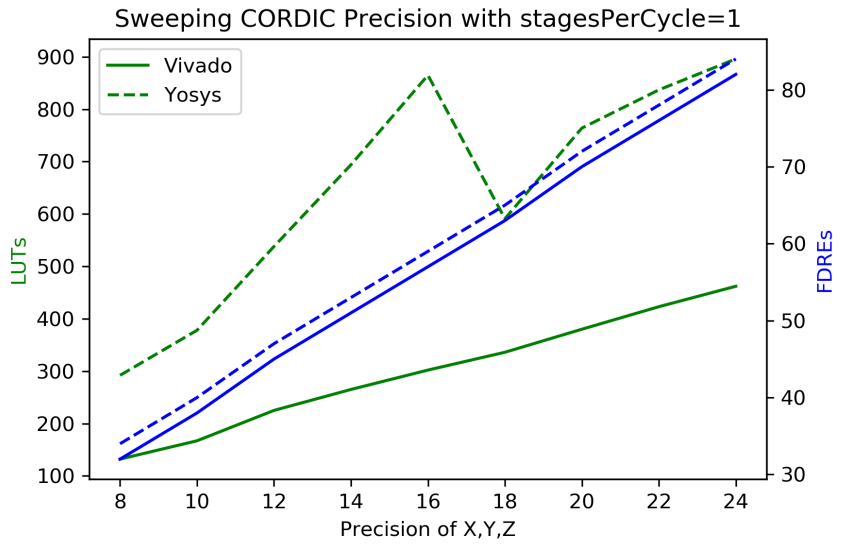
```
typeclass CORDICNumber = Data : Ring : BinaryRepresentation : ConvertableTo : Order
```

2 Synthesis

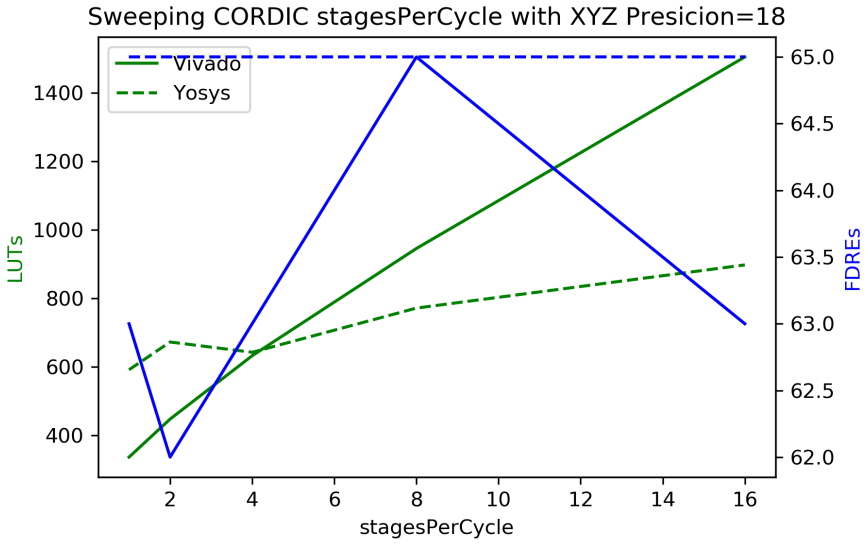
I modified the `runhammer.sh` script to sweep precision from 8 to 24 in steps of 2 with fixed `stagesPerCycle` of 1 and to sweep `stagesPerCycle` within [1, 2, 4, 8, 16] for fixed precision of 18 (which yields 16 stages). `runhammer.sh` was also modified to run a local Vivado Webpack edition (so I'm synthesizing for a Kintex Ultrascale+ part).

The number of DSPs was always zero, so I'm going to compare the LUT and FDRE utilization versus the module parameters.

I decided to try out `yosys` Xilinx synthesis just for fun.



Yosys appears to be decent at logic optimization (unsurprisingly since it uses Berkeley ABC just as Vivado does for this purpose). But it appears to be a lot worse in LUT packing, at least for this small design. LUT packing is likely something Xilinx has sunk a lot of time into. Yosys uses ABC for LUT packing, but Vivado likely uses something custom.



Here we can see again that Vivado is slighly better at register trimming as it seems to be able to optimize away what I think is the result of my setting the width of my `currentStage` register to `log2Ceil(nStages) + 1`. I think Vivado is able to recode my state machine and extract the savings from this width reduction and knowing that `currentStage` is always incremented by a power of 2 (reducing its number of possible states).

Yosys appears to perform favorably in LUT packing here, probably due to the increased amount

of logic being more amenable to ABC's packing algorithm.

3 Rocket-Chip Integration

I filled in the template provided to integrate the CORDIC block as a diplomatic TL module. I tested the rotation mode across the same 16 angles I tested in Scala, and dumped the output to `fa18-by/verisim/rotation.out`. A reference output was generated by the `CORDIC.ipynb` notebook and is in `fa18-by/verisim/rotation.gold`. Running `diff --color rotation.out rotation.gold` shows the RTL simulation matches the reference within 0-2 LSBs.

The same files were generated for vectoring mode tests in C (`vectoring.gold` and `vectoring.out`). When it comes to matching $z_{out} = \tan^{-} 1(y)$ all the results are within 1-2 LSBs. But, when looking at the expected value of $x_{out} = \sqrt{x_0^2 + y_0^2}$, the values in the range $y_0 \in [-0.5, 0.5]$ all are within 2 LSBs, but the extreme y_0 of -1 and 1 give large errors. We never tested this in Chisel simulation, and I think this is caused by overflow with large y_0.