1. Multiply 2 2's compl. #s.
   4-bit \times 4-bit $[-8, +7]$ 
   \[ \Rightarrow \text{8-bit product} \]
   
   -8 \times -8 \text{ requires 12-bit product!}

\[ \begin{array}{c}
\text{Ex: Fractional } \\
0.4 \times 0.4 \\
1.3 \times 1.3
\end{array} \]

2. Often
   
   \[ [-1, +1] \cos(\theta) \]
   
   Q: how to encode \( \cos(\theta) \)?

   Perform \( x \times (+1) \) with a max
   
   Encode input \( A \) in 1.\text{xxx} format

\[ \begin{array}{c}
1.\text{xxx} \quad \text{[-1, 0.999]} \\
2.\text{xxx} \quad \text{[-2, +1.19]} \\
\end{array} \]
A = 0.99 + j0.99

|A| = 1.4

θ = 45°

A * 1eθ = 0 + j1.4  can not fit in [-1, +0.999]

Solution 1: grow output by 1-bit

Solution 2: guarantee A is inside the unit circle

|A| < 1.0, |A| ≠ 1.0

Solution 3: scale 1eθ  ⇒ \[
\frac{1}{1.4}\]

But it grows a bit if we don’t do something special
Same as 2) for complex number:

Two critical cases:

1) Multiply by $[1, 0, 0, 0]$
2) $u = [0, 1, 0, 1]$

Solutions:

1) Encode in 2. x
2) Use mixer to bypass complex multiplication for 2 cases

Like to do:

Soln 1)

4 mults
$n \times 8$
$(n=8, 04)$

Soln 2)

4 mults
$n \times 9$
$(n=8, 72)$

$z = (0.1)(1,0)$
Control HW

- Typ, small HW
- Substantial amount of verilog
- 
  - Care little about area
  - " " " performance
- Care about clarity in code
- " " bugs

Design process

- Think
- Determine all registers
- Think
- Draw diagrams
- Think
- Write verilog
- Test it
- Optimize it
FSMs contain
1) N.S. Logic
2) State register

Generating Complex Functions

Ex: cos, sin, tan
    tan^{-1}
    log
    e^x

analog or RF conversion values

1) High-precision numerical calculations

Ex: CORDIC
Exs: Taylor series, polynomial

Multi-cycle
1-2 bits per cycle

Can regain throughput with parallel implementations
Still have latency

2) Lookup Table