hi :P

I'm not restating the problem

For part two everything is the same as in part one except we need count all zeros.

Un-rolling the mod operator makes this super easy, since we are already inadvertently counting how many times we pass zero

where are we doing this?

```
    let pos_sel = List.map ~f:(of_int_trunc ~width:16) [100; 200; 300; 400;
500; 600; 700; 800; 900; 1000]
              |> List.map ~f:(fun num -> din >=: num)
              |> Signal.concat_msb in
```

This 10 bit signal tells us how many times we "pass zero" while performing the mod

1000000000 = passed zero once
1100000000 = passed zero twice
in general, number of leading ones -> number of times passed zero

similarly, for the negative case:

000000000 -> passed once
100000000 -> passed twice
in general, leading ones + 1 = number of times passed zero

easy or best practice

I have a bad feeling this will make my critical path even worse, but it's cool so I'm using it.

literally three lines of code:

```
...
let pos_times_passed_zero = leading_ones pos_sel in
let neg_times_passed_zero = leading_ones neg_sel in
let times_passed_zero =
    mux2 din_is_pos pos_times_passed_zero (neg_times_passed_zero +:
of_int_trunc     ~width:4 1) in
...
```

next we need the zero_counter_reg to use this value (in addition to the zero flag it already uses)

```
...
let next_couter_reg = (counter_reg_out +: (uresize ~width:16 dial_reg_zero)
+: (uresize ~width:16 mod_out.times_passed_zero)) in
...
let cr = Signal.reg spec ~enable:valid next_couter_reg in
```

simply enable the counting register whenever there is valid data and define a new signal `acc = acc + zero_flag + times_passed_zero`

this built first try :)

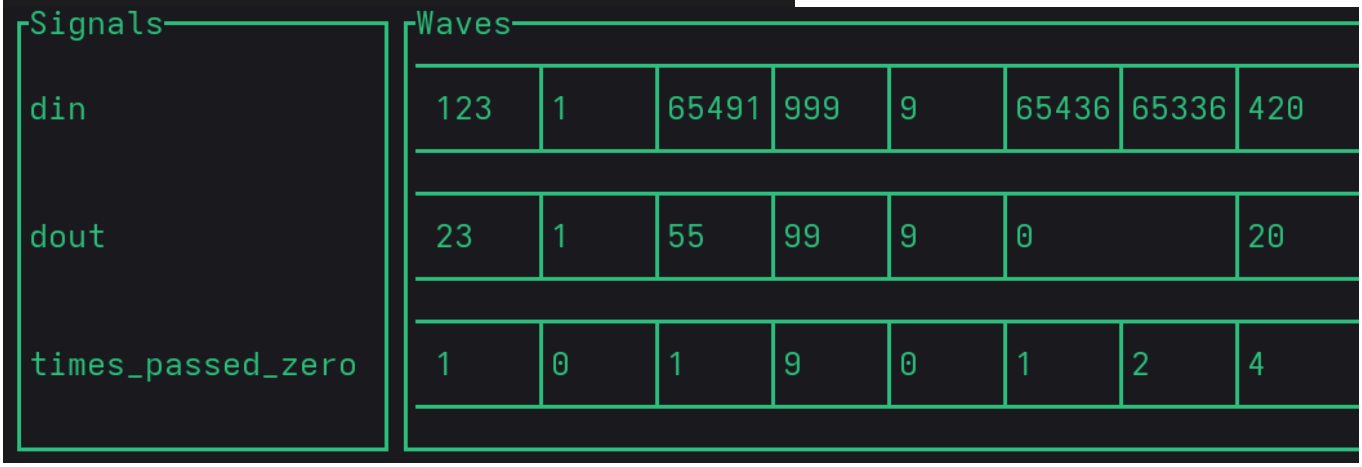
Running with my actual puzzle input:

```
 ;;
@@ -4612,7 +4612,7 @@ let%expect_test "AND gate with printed waveforms" =

   [%expect
     {|
-    0      | 1084   |
+    0      | 7567   |
```

Wrong :(but I would have been surprised if it worked first try.

Lets look at the tb for the mod operator specifically

Index	Input	Notes
0	123	
1	1	
2	-45	
3	999	
4	9	
5	-100	
6	-200	
7	420	



This is doing what appears to me to be the correct behavior, I need to break out the old ***compare to software and spot the difference***

what did I find?

Here are the first few outputs of the count register from the vcd file

```
0
1
2
4
5
6
7
8
9
11
```

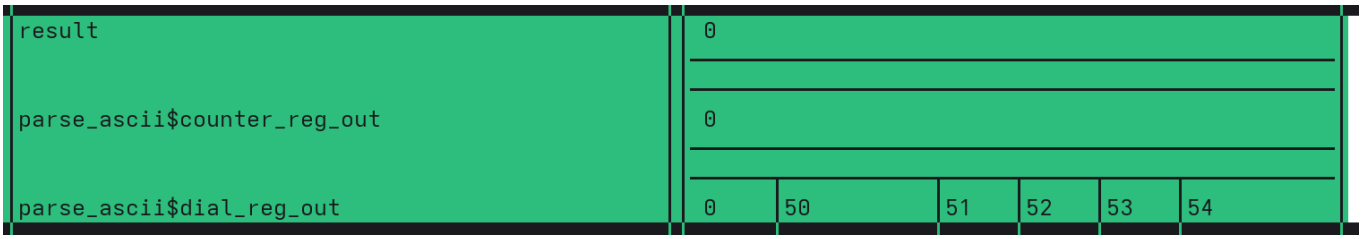
```
12
13
14
15
16
17
18
19
20
21
23
24
25
26
27
28
29
31
32
33
35
36
37
38
39
40
41
43
44
45
46
47
49
50
51
52
53
54
61
```

note that the number is always increasing! Never staying the same, which means there is probably something wrong with my `next_count_reg`

Maybe I can spot this issue with an expect test?

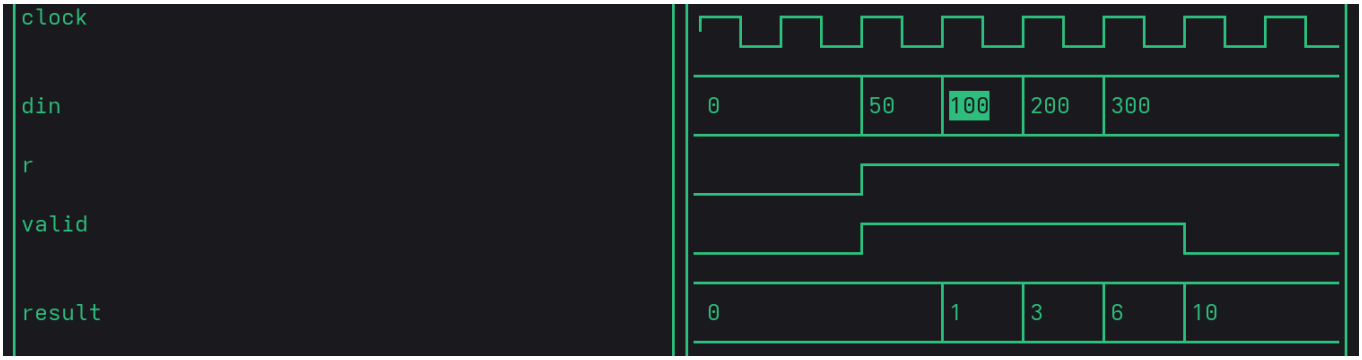
R1, R1, R1, R1...

when i simulate 50 -> 51 -> 52 -> 53 -> 54



:(now it looks like it's working what the heck

R50, R100, R200, R300...



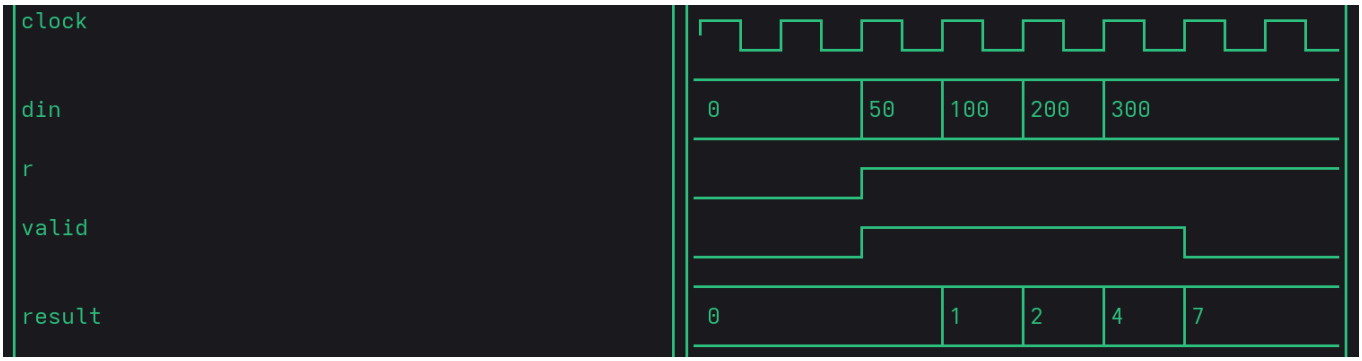
found it :(

current state = 50, input = R50, we count one zero
current state = 0, input = R100, we count 2 zeros (we should only count one)
current state =0, input = R200, we count 3 zeros (we should only count two)

soooooo, the zero output flag is annoyingly double counting

the obvious solution is to remove the zero flag from my solution but I think there might be some cases where it doesn't double count

instead of intelligently searching for a case when this happens, I'll just remove it and see if something else breaks



the simple case works now but I just thought of this case:

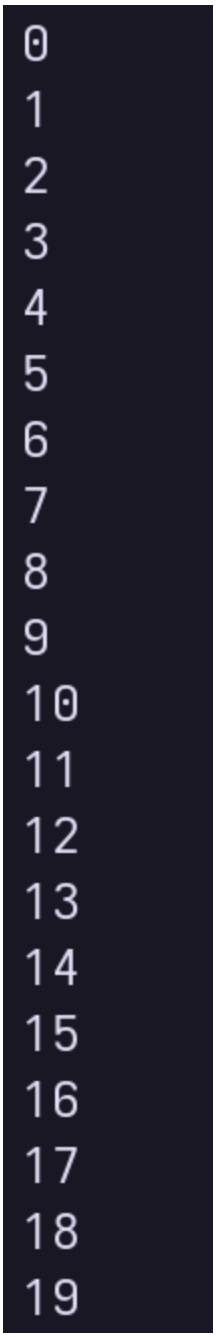
current state = 1, input = L1, output = 0 (we don't count because no mod wrap around happend)

```
solution:

if the result is zero don't accidentaly double count but we also can't just
remove the offending signal, it is needed when we land on zero without a
wrap-around

Any hopes of a high clock speed are going down the toilet :(
```

it looks like its working so lets try the actual puzzle input again

`still over counting :(`

lets see what it's doing:



I have successfully made something that counts (never learned how)

Since that implies some kind of mistake on one of the first few inputs, lets consider those inputs: spoiler, I did and they work fine

```
the vcd lib I was using only reports when a value changes, it doesn't report
the value at regular intervals. I had to write another scipt that tracked
the clock and the most recent value of the signal I cared about to extract
them

TLDR: my debugger was kinda lying to me sorta
```

but now that I fixed it I can compare sim values to software values:

```
diff signal_per_cycle.txt ../../please_delete_me/output.txt > diff.txt
```

some of the diff:

```
12c12
< 2
---
> 3
35a36
> 8
39c40
< 12
---
> 11
41a43,44
> 13
> 14
```

that 2 vs 3 difference is pretty early on we can totally look at that

I've spent several hours staring at this bug, but I have no way of debugging it because gtkwave refuses to show me intermediate signal values, even if I mark them with `%hw`.

I refuse to add random outputs to my design like some sort of cave person and won't work on anything else until I'm able to view intermediate values in a waveform outside the terminal

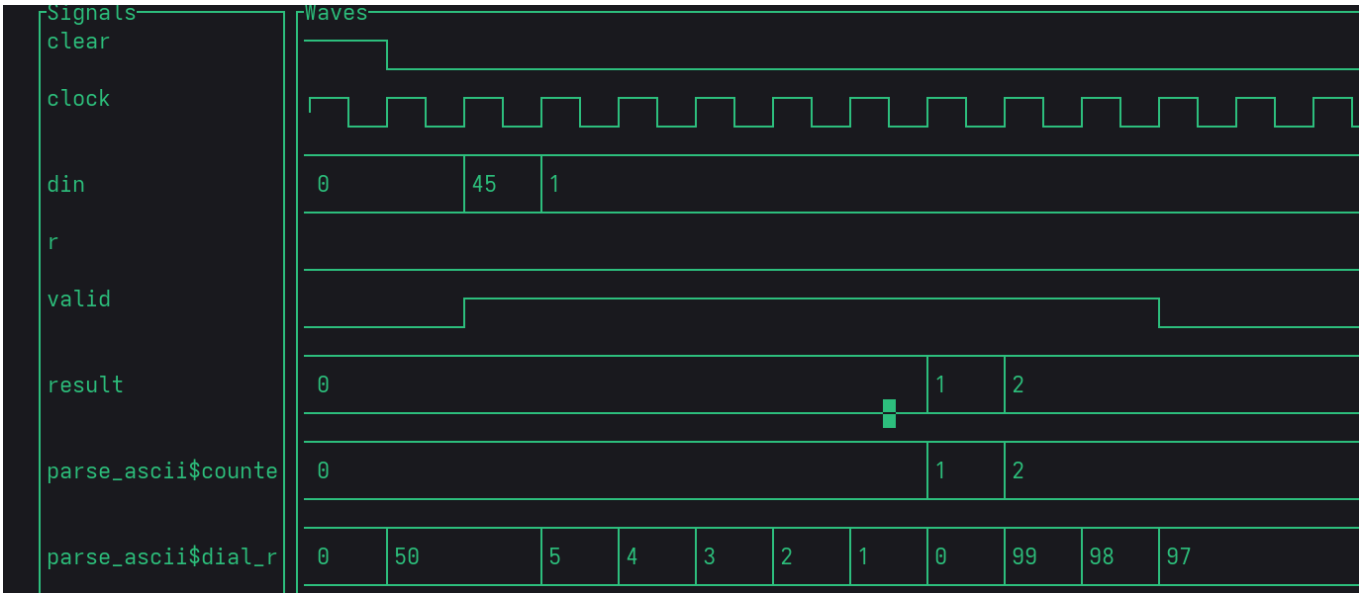
debugging this any other way would be a hacky work around (very angry at this if you can't tell)

```
after sleeping on it, gtkwave will happily display the signals but won't
display them in the hierarchy view literally what the heck :(
```

some bugs i found

for one, some of my signals were out of sync

for two:



current state 0, input L1 increments the counter :(

TLDR my `times_passed_zero` signal is wrong

I spent a lot of undocumented time on bugs and have decided the method I'm using is cursed! I will instead be using the mod_100 component to determine how much to increment the counter reg by

I need to determine how many times we passed zero given

- value to mod (sort of encodes current position and current input)
- mod result (encodes if we landed on a zero or not)

next next day and I'm in edge case hell :(

current bug:

current_pos = 73, input = L 473, count increments by 4 but it should by 5 ;(

after much debugging

I used my dad as a rubber ducky (ultimate debugging strat and managed to get it working)

I had it half solved and had to spot some symmetry to get it working

honestly this was much harder than I thought with all the weird edge cases

If i had to do this again I would have pipelined the counting of times passed zero, I only need the mod operator to take one cycle, not computing the times passed zero (and since it has so many edge cases, computing times passed zero is 100% the critical path)

I'm just happy I have something working, lets try day 2!