# I/O

Arbejdsopgaverne blev fordelt tidligt i forløbet pga. naturlig interesse for at lege med VHDL og bestemte dele af FPGA’en. Denne fordeling fortsatte, og dette dokument omhandler I/O og den senere sammenfletning med CPU’en.

## FPGA

Første trin var at lege med 7 seg displayet på FPGA’en for at få en idé om hvordan den virker. Cyclone’s og Papilio’s 7 seg display fungerer forskelligt mht. pins og timing, så det er værd at notere at legen er lavet på Cyclone V.

Herefter koblede vi dipswitchene til displayet, så de styrede det der stod på displayet. Her var der problemer med timingen når to switches blev håndteret samtidigt.

Koden her for I/O var meget sammenflettet, dvs. display og input var ikke adskilt. Det gjorde det svært at se hvordan input fra CPU’en skulle komme til.

Debugging tog lang tid på FPGA’en, og der var ingen serial monitor. Det er svært at fejlfinde når man selv skal programmere f.eks. at LED’erne lyser ved fejl.

## ModelSim

For at lette fejlfinding (og også fordi vi er adskilt pga. corona og kun har 1 Cyclone V til rådighed) valgte vi at fortsætte i ModelSim. Dette gjorde det muligt at observere waveforms af alle de anvendte signaler. Herved var det ikke nødvendigt at programmere fejlmeddelelser. Det gjorde det også lettere at observere timingproblemer.

Arbejdet her bestod i at opdele input og output, så grænsefladerne var tydelige. Det skulle gøre det muligt at koble I/O til CPU’en på et senere tidspunkt og ikke kun læse fra input. Dette resulterede I, at vi ikke bare kunne sende enkelte cifre fra input til output, så vi var nødt til at konvertere fra input til binær og senere igen fra binær til output. Fra input til binær skulle vi ved hvert nyt taltryk bare gange det nuværende tal med 10 og lægge det nye ciffer til. Fra binær til output var vi nødt til at programmere en binær til BCD converter for at kunne vise de enkelte cifre på skærmen i decimal og ikke hex.

Vi kørte direkte videre med displaydriveren fra FPGA’en, idet den bare består af en MUX der tegner hvert enkelt ciffer ud fra en nibble (i henhold til manualen 3.3). På Cyclone V, som vi udviklede til, har hvert ciffer sine egne 7 pins. Dette er forskelligt fra Papilio’en, hvor alle cifrene deler 7 pins og det enkelte ciffer vælges med en selector.

Undervejs i arbejdet forsøgte vi at tage hensyn til at vores input-modul skal kunne give både tal og signaler videre til CPU’en for videre behandling. Outputtet skulle også kunne modtage fra enten input eller CPU afhængigt af den nuværende state.

Senere i processen blev det besluttet at systemet skulle kunne håndtere 2’s complement. På display skulle der derfor implementeres et minus-tegn, og det skulle kunne tjekke hvor minus-tegnet skal stå. Dette fungerede efter hensigten.

# Sammenfletning af CPU og I/O

De følgende trin beskriver arbejdet med sammenfletning af CPU og I/O (betegnet de to systemer). Det blev løbende undersøgt i ModelSim’s Waveform om de to systemer hver især fortsat kørte som ønsket, før vi gik videre til næste trin.

1. De to systemer var hver bygget op med egen testbench. De to testbenches skulle flettes sammen til én, så vi kunne bekræfte at de to systemer kunne køre uden problemer på samme testbench.
2. Grænsefladerne mellem de to systemer blev identificeret. De steder, hvor det var muligt at koble grænsefladerne til uden videre omkodning (f.eks. clock), blev dette gjort.
3. Det blev undersøgt hvordan grænsefladerne mellem de to systemer egentlig skulle fungere, da CPU’en på dette tidspunkt ikke havde opkoder til håndtering af I/O-funktioner. F.eks. pull af indtastede cifre og operationer fra input og push af resultat til output.
4. Der blev oprettet globale kommunikationsveje mellem de to systemer. Heriblandt er ActionJackson’s 6 bits, som gemmes i et 8 bit array og derfor kan gemmes uden videre i register eller RAM. De enkelte bits i ActionJackson fortæller hvilken operation der ønskes, om der er trykket facit, og om der er trykket clear. Disse bitkombinationer giver unikke tal, som CPU’en vil kunne teste logisk op imod.
5. Der blev oprettet 4 nye opkoder i CU’en og ALU’en. To af dem gemmer indtastede værdier på en given registerplads. En gemmer ActionJackson på en given registerplads. Den sidste sender indholdet af en registerplads til display.
6. På dette tidspunkt blev det vurderet at systemet var klar til at blive testet med programkoder, der skal håndtere behandling af data som er indhentet fra I/O. Der blev skrevet særskilte programkoder og testopstillinger for at debug-teste følgende:
   1. x + y = z
   2. x - y = z
   3. y - x = z (for at se om displayet kan håndtere negative resultater – det kan det)
   4. x \* y = z
   5. x / y = z (runder ned – det er fint på nuværende tidspunkt)
   6. x / 0 = z (giver 0 pga. CPU’ens programmering – skal muligvis give ERROR)
7. Der skrevet en programkode og en testopstilling, hvor de forskellige beregninger blev udført efter hinanden, både med og uden at der blev tastet Clear imellem dem. Det lykkedes at få systemet til kun at foretage beregninger efter der er tastet Clear (men selvfølgelig også lige når lommeregneren er tændt).

Per 22. april findes en fungerende lommeregnerkode. Denne er gemt i projektmappen på GitHub og ligger også i VHDL-programkodefilen.

I forbindelse med programkodeskrivning har vi også udviklet en assembler til denne specifikke CPU, der ligeledes findes i GitHub-mappen. Koderne der skal bruges til den findes i instruktionssæt-filen (Excel). Desuden findes der en MARK-funktion, som bruges til dynamisk at referere til steder i koden. Se evt. den gemte programkode nævnt i afsnittet ovenover.