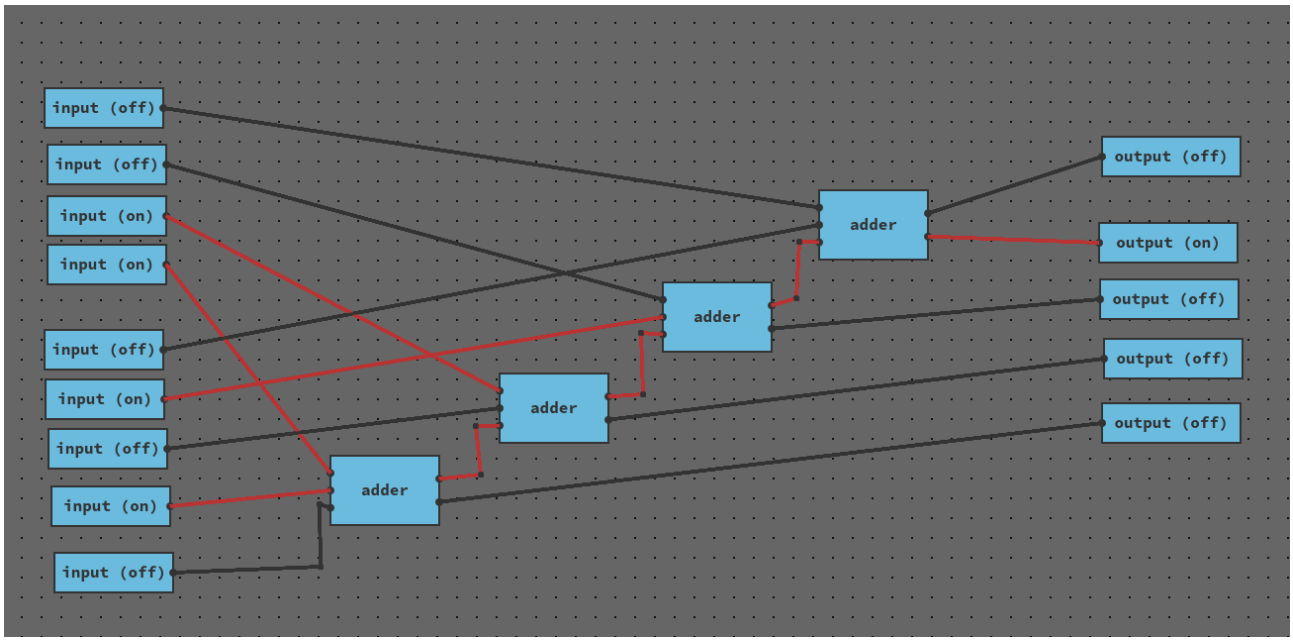
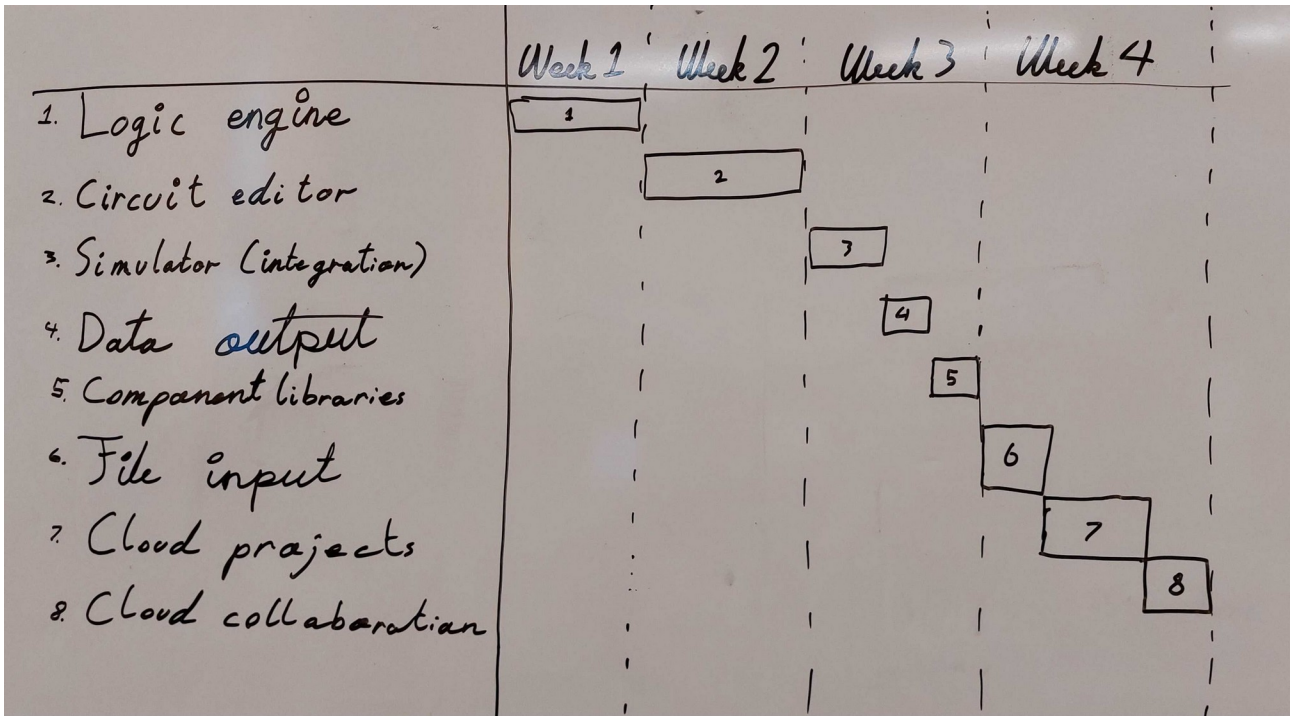
Svendeprøve Processrapport
Simon From Jakobsen
Juni 2026

NandSim - Process report



Initial estimation



I expect to spend the most time on the logic engine and the circuit editor. I have prioritized the logic engine and the circuit editor, because they are more important for fulfilling the minimum viable product. The other features depend on these. I have also placed what I deem are less important features at the end. In case I don’t get to them, the product will still fulfill the minimum viable production requirements.

Journal

Week 1

This week was primarily spent developing the case, the requirements, researching the topic, reading about the chosen technologies and setting up the project.

I tried implementing an experimental logic engine as a stand-alone project. This was to get more familiar with representing and simulating logic circuits.

I initialized the application project using Vite, React and TypeScript. I spent significant time tweaking the setup to be more how I wanted it.

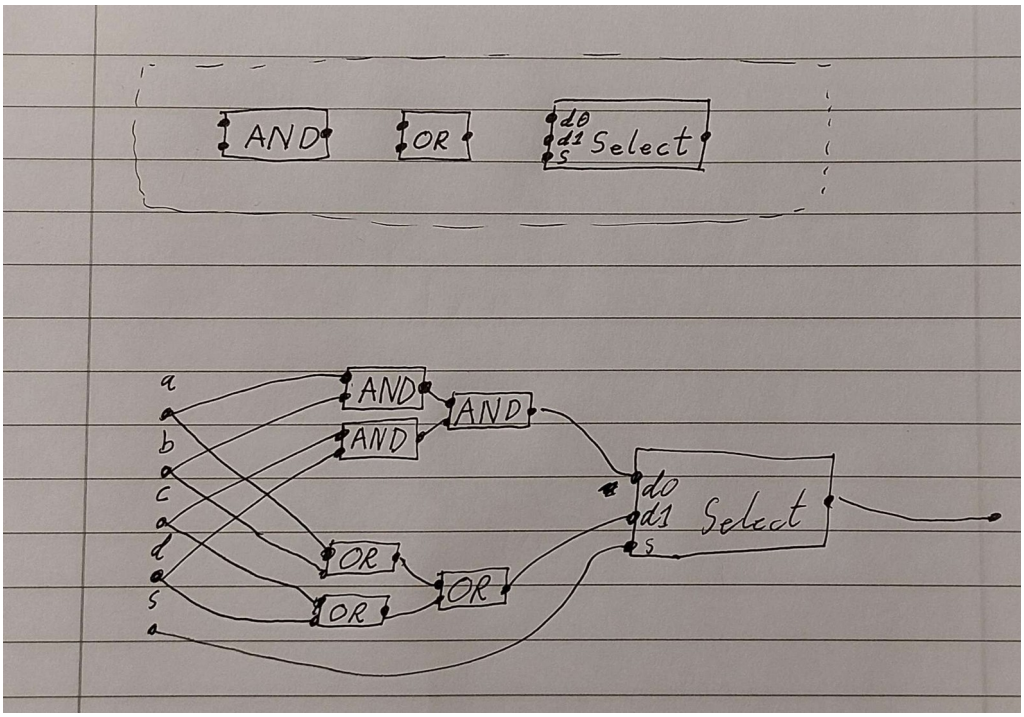
I added an Editor-component to the project, that was the beginning of the actual implementation of the application. I struggled at this stage with composing the project. I had little knowledge of the the application was going to develop. I added features and components anyway with the hopes that I would later be able to refactor it with more knowledge.

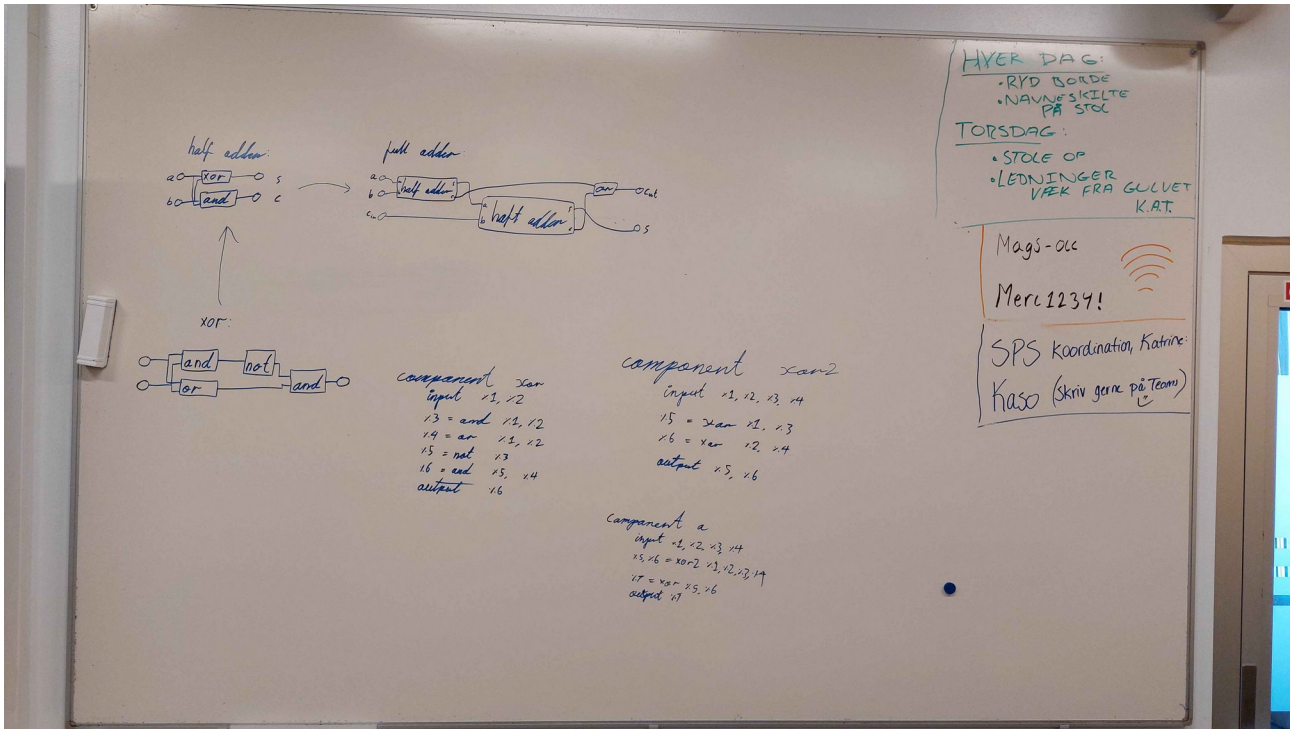
I implemented some of the first editor functionality. This was camera panning and grid drawing and box selection. I decided to try and use a state pattern for the editor.

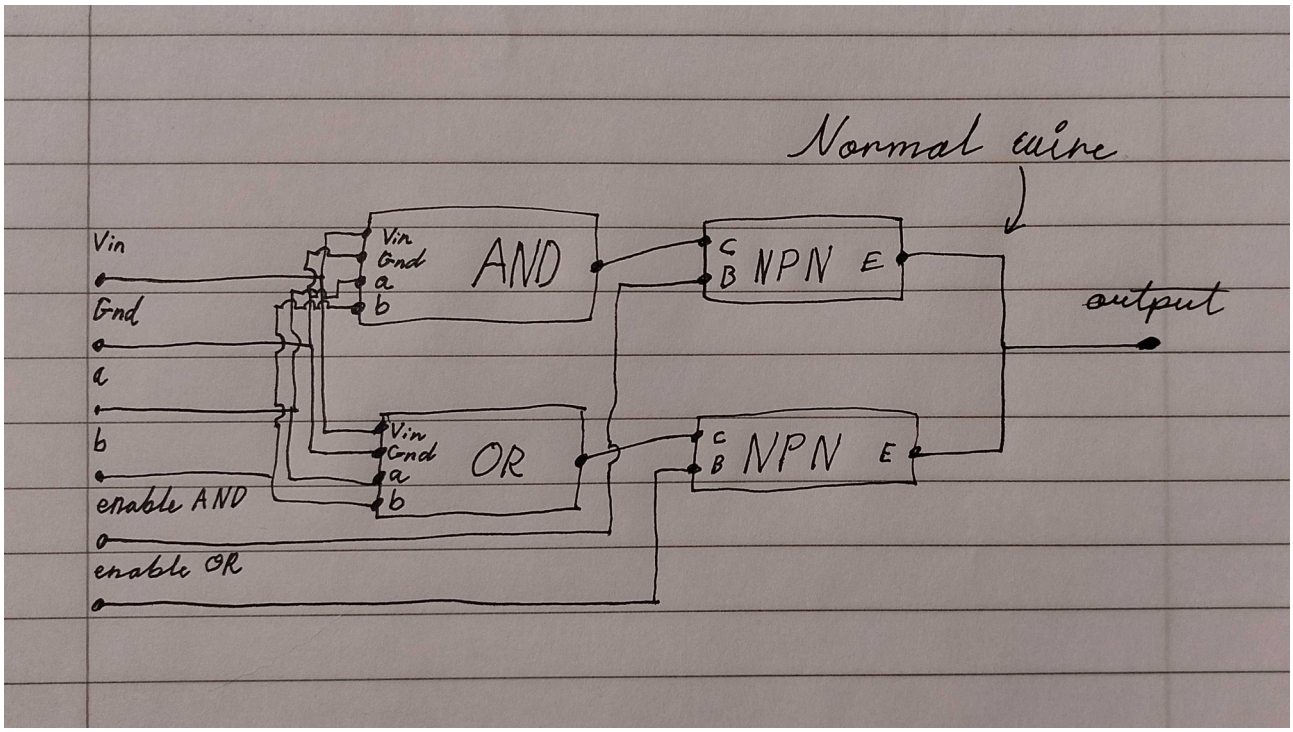
I added placing, so you could place components. I added component-placing as a new state to the editor. In this process I also experimented a bit with how the state pattern was implemented. I landed on an implementation that seemed to work well.

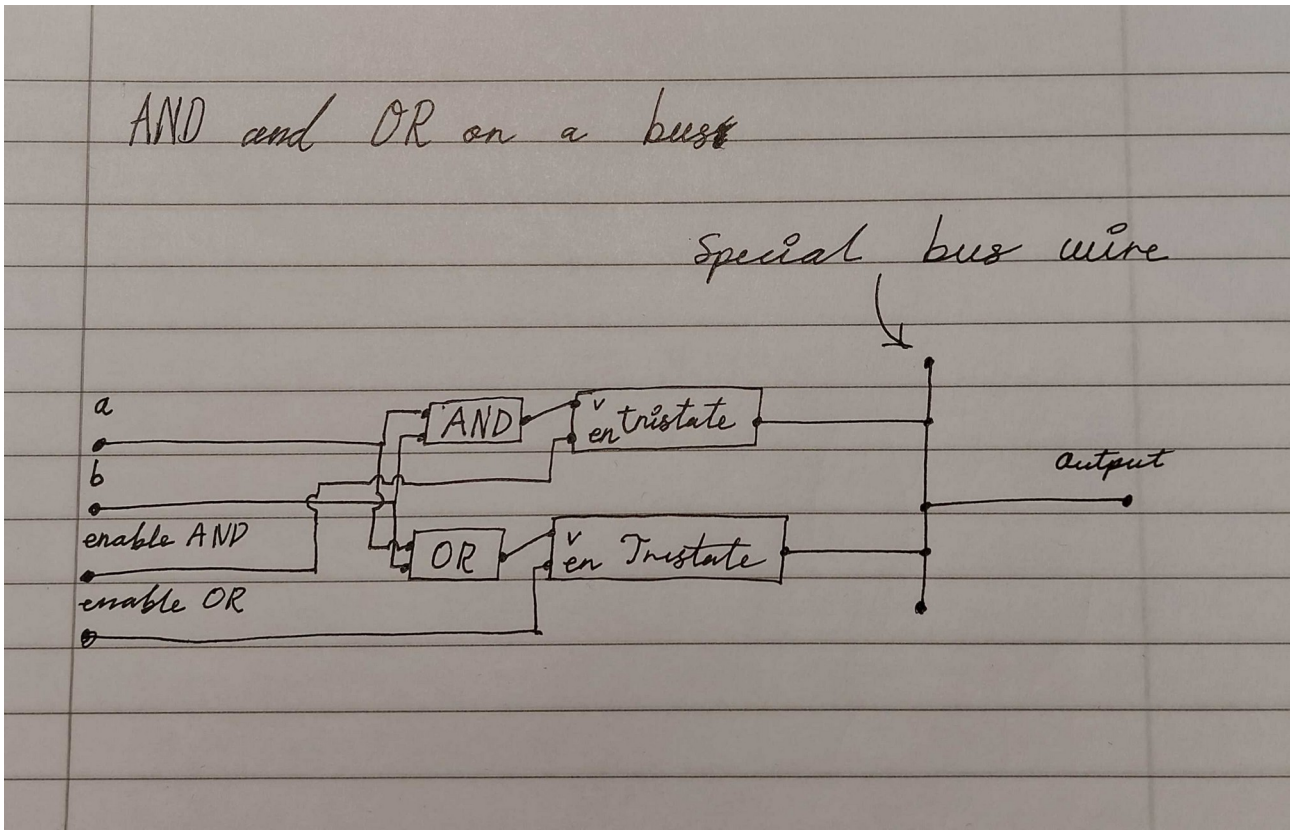
The last thing I added this week was component kinds and pin hovering.

I also spent time this week experimenting with different approached of representing and simulating circuits.


AND
OR
d0
d1
Select
s
a
b
c
d
s
AND
AND
AND
OR
OR
OR
d0
d1
Select
s


half adder
xor
and
full adder
half adder
half adder
or
xor
and
not
and
or
component xor
input x1, x2
x3 = and x1, x2
x4 = or x1, x2
x5 = not x3
x6 = and x5, x4
output x6
component xor2
input x1, x2, x3, x4
x5 = xor x1, x3
x6 = xor x2, x4
output x5, x6
component a
input x1, x2, x3, x4
x5, x6 = xor2 x1, x2, x3, x4
x7 = xor x5, x6
output x7
HVER DAG:
•RYD BORDE
•NAVNESKILTE PÅ STOL
TORSDAG:
•STOLE OP
•LEDNINGER VÆK FRA GULVET K.A.T.
Mags-occ
Merc1234!
SPS koordination, Katrine:
Kaso (skriv gerne på Teams)


Normal wire
Vin
Gnd
a
b
enable AND
enable OR
Vin
Gnd
a
b
AND
OR
C
B
NPN
E
output

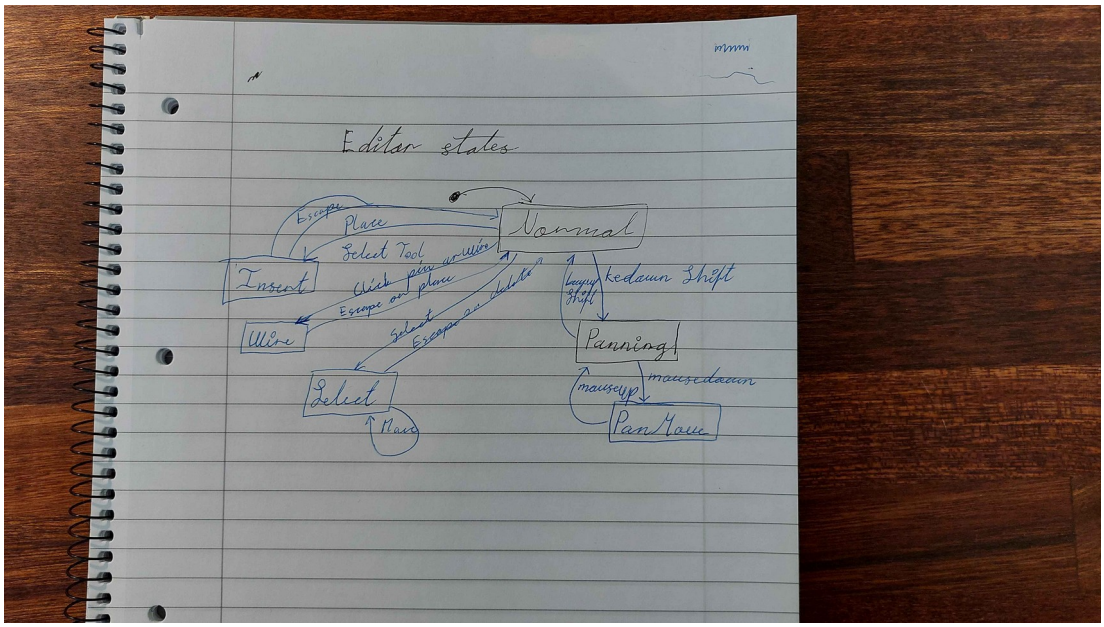

AND and OR on a bus
Special bus wire
a
b
enable AND
enable OR
AND
OR
V
en
tristate
Tristate
output

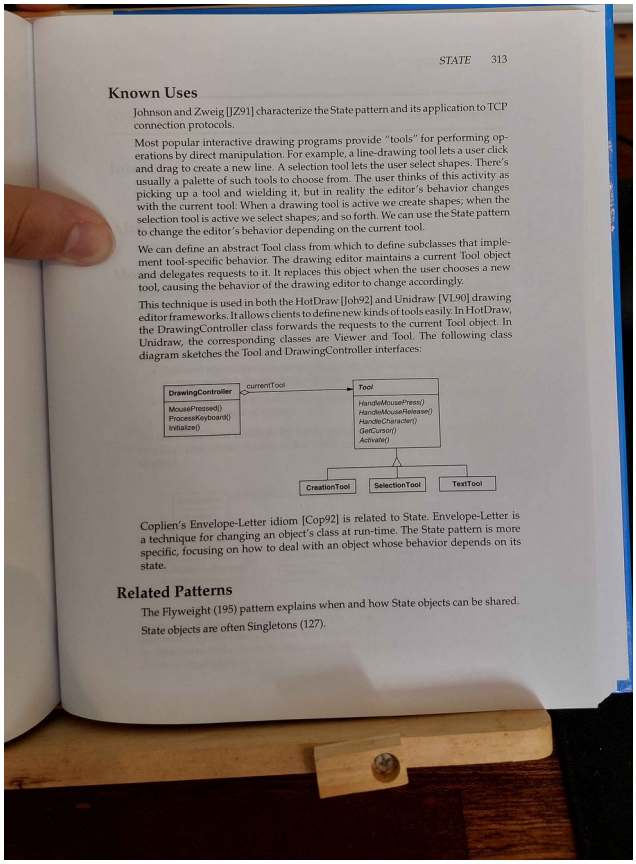
Week 2

The first part of the second week was spent refactoring the code from last week. Given my lack of knowledge of how the code was going to work, when I implemented it, I made some poor decisions. Primarily this involved extracting code from the central Editor class and extracting and centralizing all rendering code into a Renderer class.

After this refactor I spent a significant about of time researching and developing ideas of designing the implementation. Through this research is learned more about how to use the state pattern in the editor, and how I could use it without the code becoming messy.

The last part of the week was spent implementing selection and wiring of components. You could now place components, wire them together with wires and joint, selecting them, and move them around.





STATE 313

Known Uses

Johnson and Zweig [JZ91] characterize the State pattern and its application to TCP connection protocols.

Most popular interactive drawing programs provide "tools" for performing operations by direct manipulation. For example, a line-drawing tool lets a user click and drag to create a new line. A selection tool lets the user select shapes. There's usually a palette of such tools to choose from. The user thinks of this activity as picking up a tool and wielding it, but in reality the editor's behavior changes with the current tool: When a drawing tool is active we create shapes; when the selection tool is active we select shapes; and so forth. We can use the State pattern to change the editor's behavior depending on the current tool.

We can define an abstract Tool class from which to define subclasses that implement tool-specific behavior. The drawing editor maintains a current Tool object and delegates requests to it. It replaces this object when the user chooses a new tool, causing the behavior of the drawing editor to change accordingly.

This technique is used in both the HotDraw [Joh92] and Unidraw [VL90] drawing editor frameworks. It allows clients to define new kinds of tools easily. In HotDraw, the DrawingController class forwards the requests to the current Tool object. In Unidraw, the corresponding classes are Viewer and Tool. The following class diagram sketches the Tool and DrawingController interfaces:

Coplien's Envelope-Letter idiom [Cop92] is related to State. Envelope-Letter is a technique for changing an object's class at run-time. The State pattern is more specific, focusing on how to deal with an object whose behavior depends on its state.

Related Patterns

The Flyweight (195) pattern explains when and how State objects can be shared.

State objects are often Singletons (127).

Week 3

Week three was spent predominantly on activities unrelated to the project, hence there's not much to show.

I tweaked the wiring functionality, and changed the way wire connection were represented in the code. I also changed the code design on how the Board component interacts with mouse input. Before, there were no good ways of handling mouse input in a generic way. I change the design so that the board itself is responsible for checking the position viz-a-viz components, wires and joints. This improved the implementation around this significantly.

Week 4

At this point I had spent the first 3 weeks implementing what felt like half of the circuit editor functionality. I hadn't even started on the simulation part yet.

The first thing I did was put in a large effort into finishing the visual editor. The went without major problems, and the visual editor now worked to my satisfaction. There were still some minor bugs and glitches, but given the time pressure, I decided that it wasn't worth to spent time on.

I then implementing functionality to lower the circuit to IR. I assumed that this would take multiple days. Instead it took a couple of hours. I think it's because I have prior experience implementing compilers, and this is a very compiler-like problem to solve.

I then added some optimization passes, because the generated IR was insufficient, the way it was generated. This worked surprisingly well.

After having implemented the IR lowering and optimization, I implemented the simulator itself. Because of the structure of the IR, this was trivial to implement. After this, I had a simulator that worked in theory. But it wasn't hooked up to the visual editor.

I decided to introduce an event bus into the system. The purpose was to decouple some of the reactive components. This seems to have been a correct decision. Refactoring to use the event bus took some effort. But it allowed my to really easily add some more input functionality that was needed, specifically double clicking and mouse offset handling.

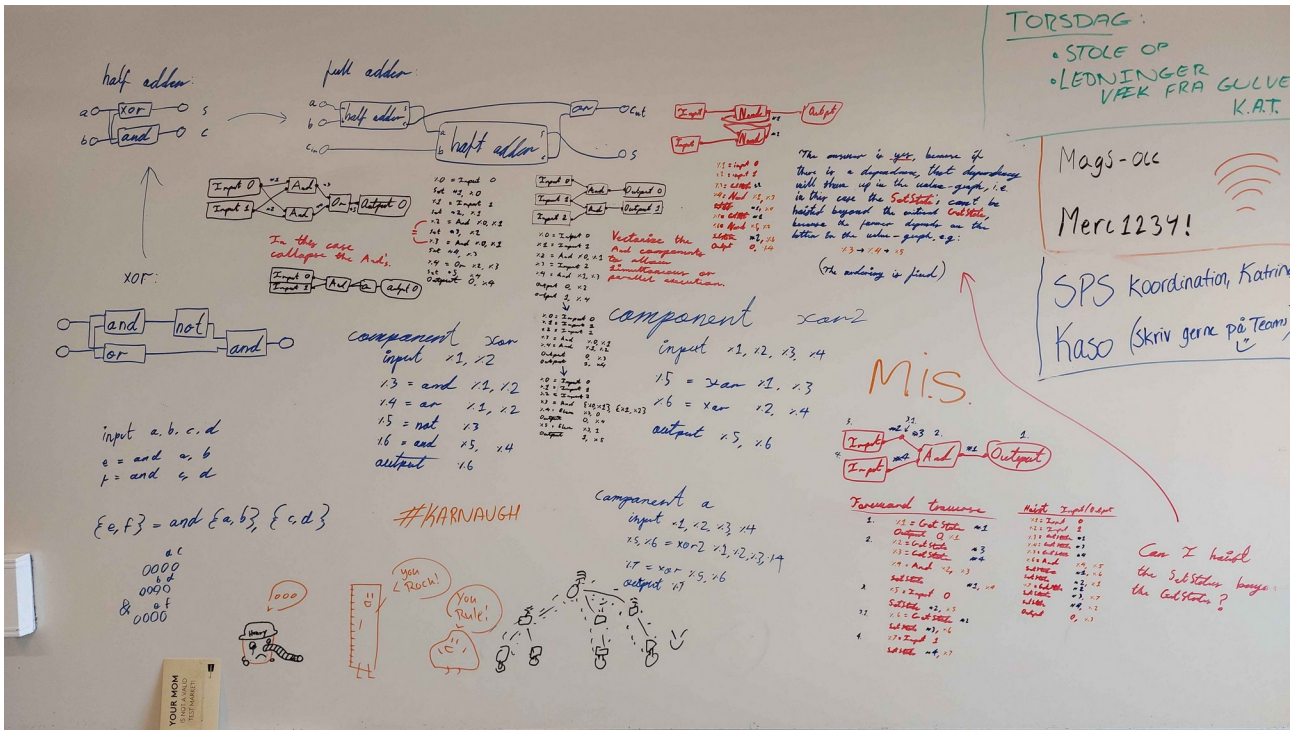
After this, I implemented so that the application wasn't just one circuit in an editor. Instead it was to be a project with multiple components and editors. I started by implementing the UI part, with tabs and tab control buttons.

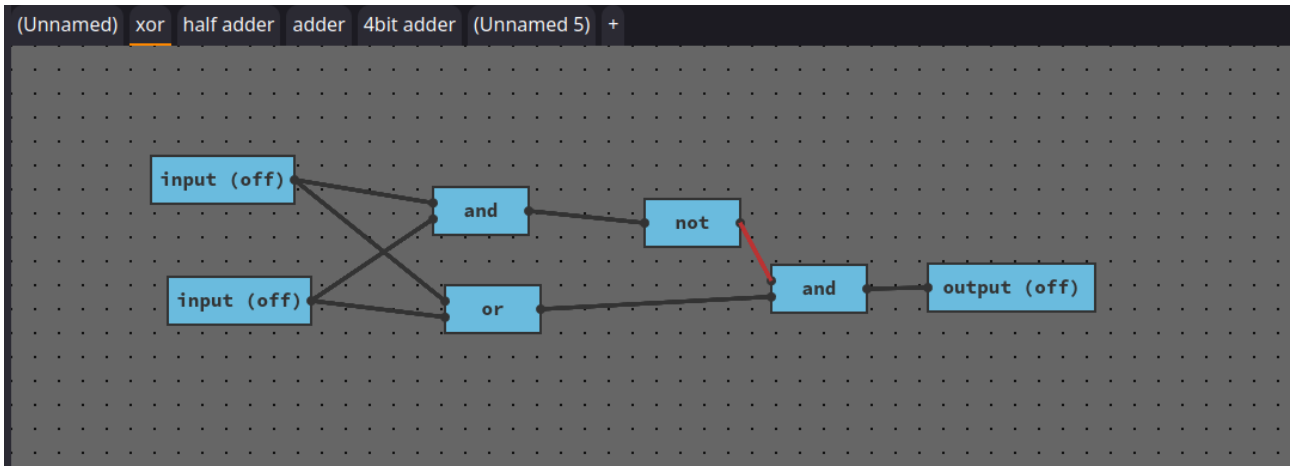
I then refactored the code a bit. One thing I did was merging Editor with EditerCx. Previously the states implementations of the Editor would get a reference to EditerCx, and EditerCx contained basically everything of Editor. I decided to merge them, as there were less of a point to them being separate, as I had originally thought.

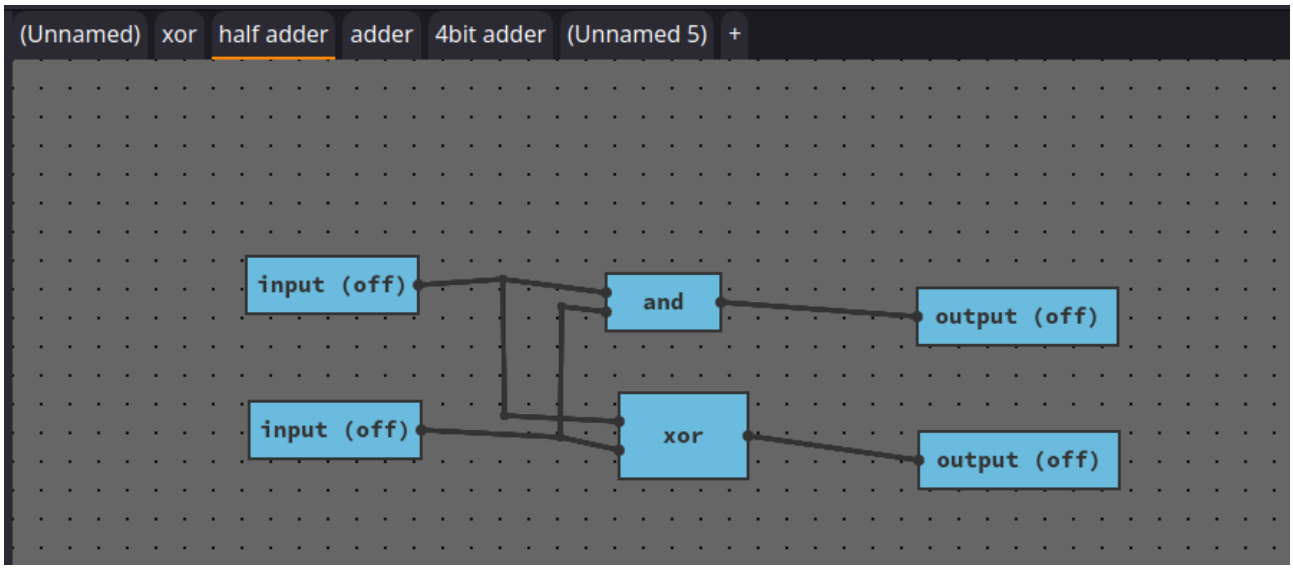
With the event bus refactor, some of the code structure also change. Because of this, I spent some time moving code around to be better organized.

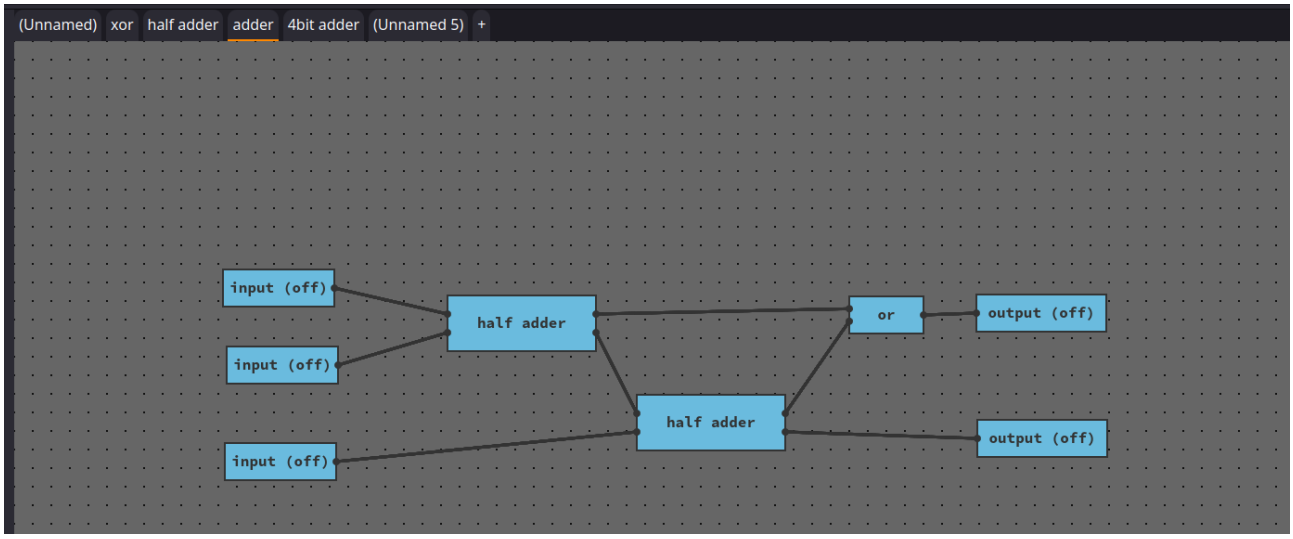
Next I implemented functionality so that you could save circuits as components and use them in other circuits. When I got this working, I made an example for demonstration of an XOR component, a half adder component, a full adder component and a 4bit adder. This worked, and could tell me that 3 + 5 = 8.

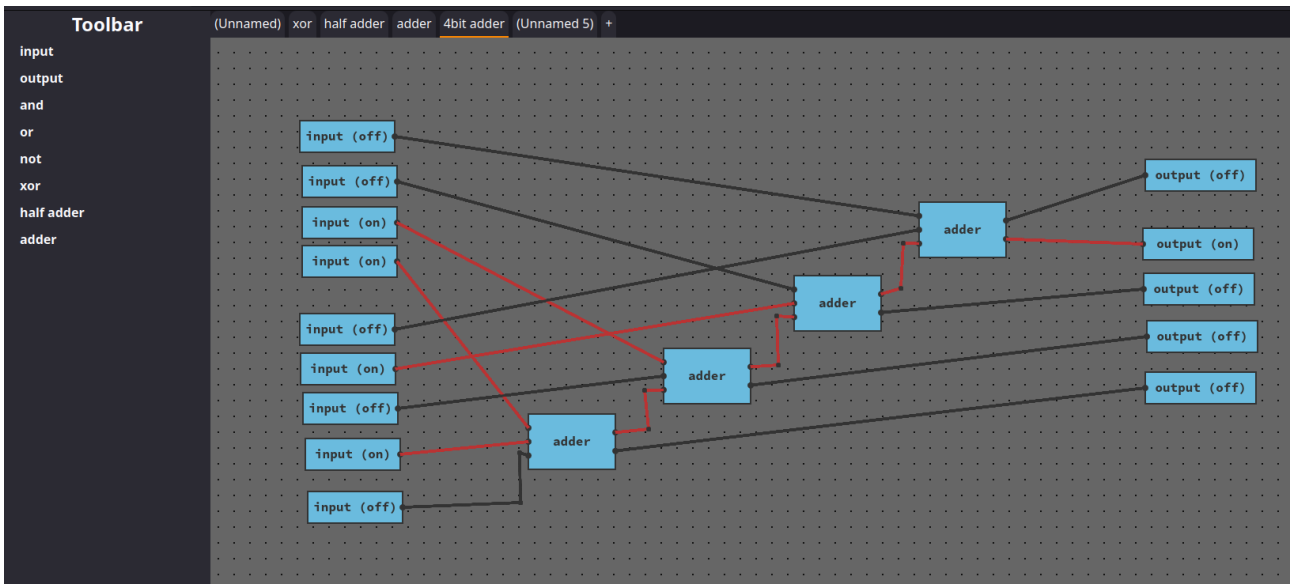
Beyond this, I spent the time writing reports and making finishing touches.



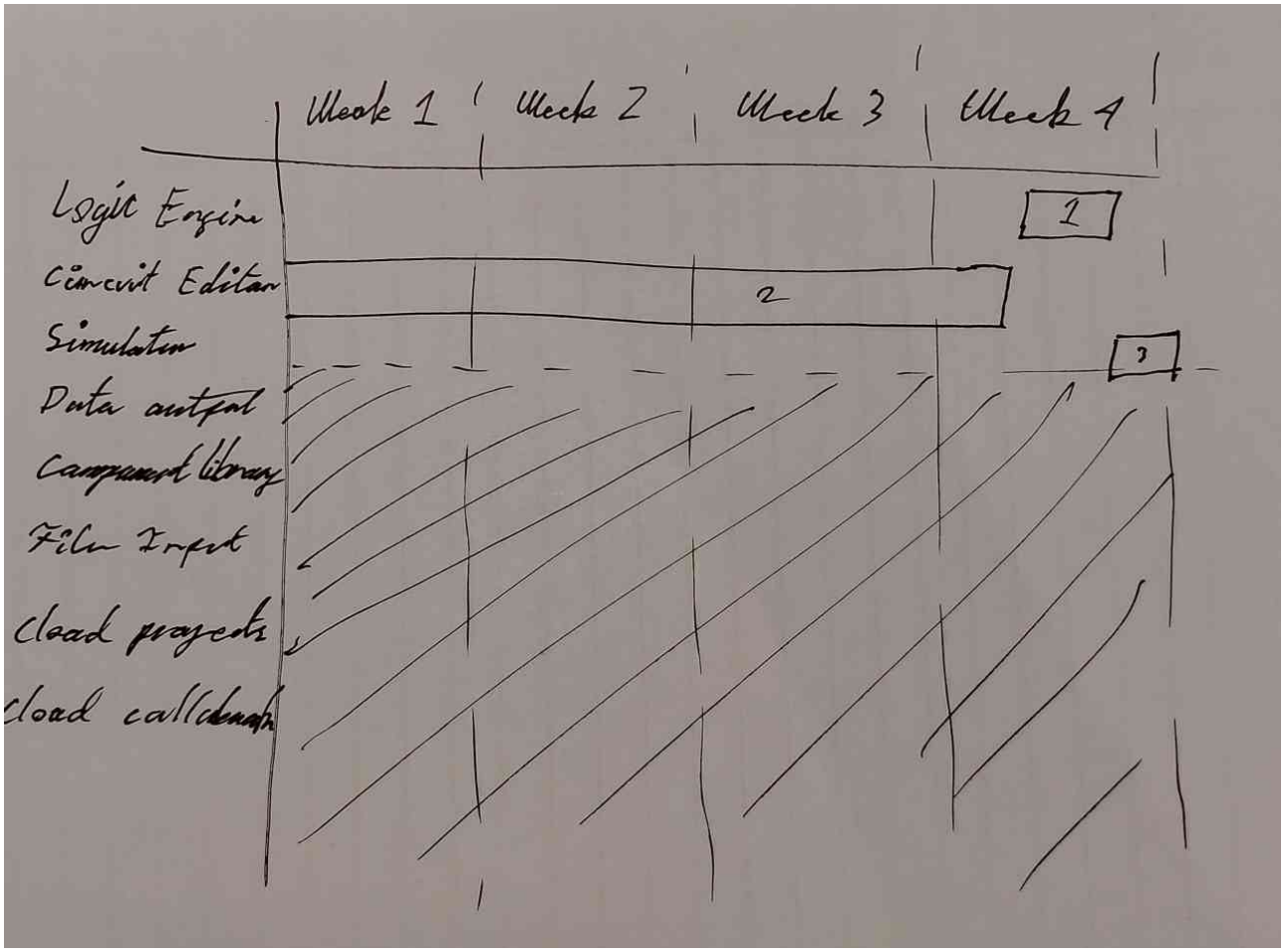



(Unnamed)
xor
half adder
adder
4bit adder
(Unnamed 5)
+
input (off)
input (off)
and
xor
output (off)
output (off)


(Unnamed)
xor
half adder
adder
4bit adder
(Unnamed 5)
+
input (off)
input (off)
input (off)
half adder
half adder
or
output (off)
output (off)


Toolbar
input
output
and
or
not
xor
half adder
adder
(Unnamed)
xor
half adder
adder
4bit adder
(Unnamed 5)
+
input (off)
input (off)
input (on)
input (on)
input (off)
input (on)
input (off)
input (on)
input (off)
adder
adder
adder
adder
output (off)
output (on)
output (off)
output (off)
output (off)

Realized plan



This is the realized plan of the project. As can be seen, I didn’t get to the lower half of the board. I only got to implement the core functionality the application.

Additionally, my estimation that simulation would take as much time as implementing the circuit building was completely wrong. The simulation part of the application took my barely one and a half days to implement. The circuit took almost the entirety of the 4 weeks to implement.

Granted, all days were as productive as the next. In week 2 and 3 I had a minor burnout of the project, due to two things. The first was struggling with the code design of the application. The more code I added, the more it felt like I was making the project worse overall. It required that I spent a couple of days working on a different project, before I developed a sufficient understanding to where I could continue development properly. The second reason was due to external stress factors, that I couldn’t control.

Conclusion

I have reached the end of development. The product works and fulfills the minimum viable product requirements according to the requirement specification. Given that the requirements align with the case description, this would mean that my product could generate value in some capacity for the customer.

My initial estimation was very flawed as can be seen in the realized plan. This was due to lack of experience developing exactly the type of application, that I set out to develop. As mentioned above, it turned out that implementing the simulation part was very easy, compared with implementing the circuit building editor.

I had originally tried to limit the requirements, such that the minimum viable product was easily attainable, and only the last one or two requirements would be out of reach. With this, there was also a failure of estimation. I only managed to finish the minimum viable product, no features to satisfy requirements beyond that.

What I think went well in the process what that I at all times managed to make progress despite it being a quite complex project. When I began struggling in week 2 and 3 with the design, I spent the time productive, using other projects to develop designs that I could apply in the implementation.

Another aspect that went well were my idea and design decision. Prior to starting the project, I had researched and experimented similar projects. This gave me an advantage, specifically in regard to knowing the direction to take the project.

I am personally satisfied with the product. Both with the time constraint in consideration, but also just generally. I like the way the product turned out. It I had more time and resources, I would like to keep developing it. I made many ideas for improvements.