DEflow: A Hardware Design Platform for Teaching Digital Electronics

Marco Selvatici

Problem

First year students are expected to design digital circuits using **Quartus**: a hardware design platform that is **too complex** and adds unnecessary cognitive load. Furthermore, the application is only accessible via a paid licence and cannot run on all operating systems. Students generally SSH into college servers in order to use it remotely. A survey among university students showed that:

- **88%** of the students get confused by Quartus' convoluted menus system.
- **63%** of the students struggle to understand the error messages provided by Quartus.
- **75%** of the students agreed that a simpler application would improve their learning experience.

Requirements Capture

This project aims to create a hardware design application that is more suitable than Quartus for first year digital electronics teaching. Therefore the following features have been outlined:

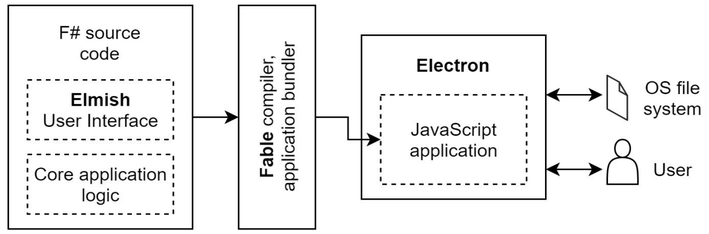
Essential features:

- Circuit diagram editor.
- Ability to reuse previous designs.
- Support for buses.
- Diagram validation and simulation tools.

Desirable features:

- Open-source and cross-platform.
- Easily extensible.
- Intuitive and responsive user interface.
- Highly informative error messages.

Technology Stack: Fable-Elmish-Electron

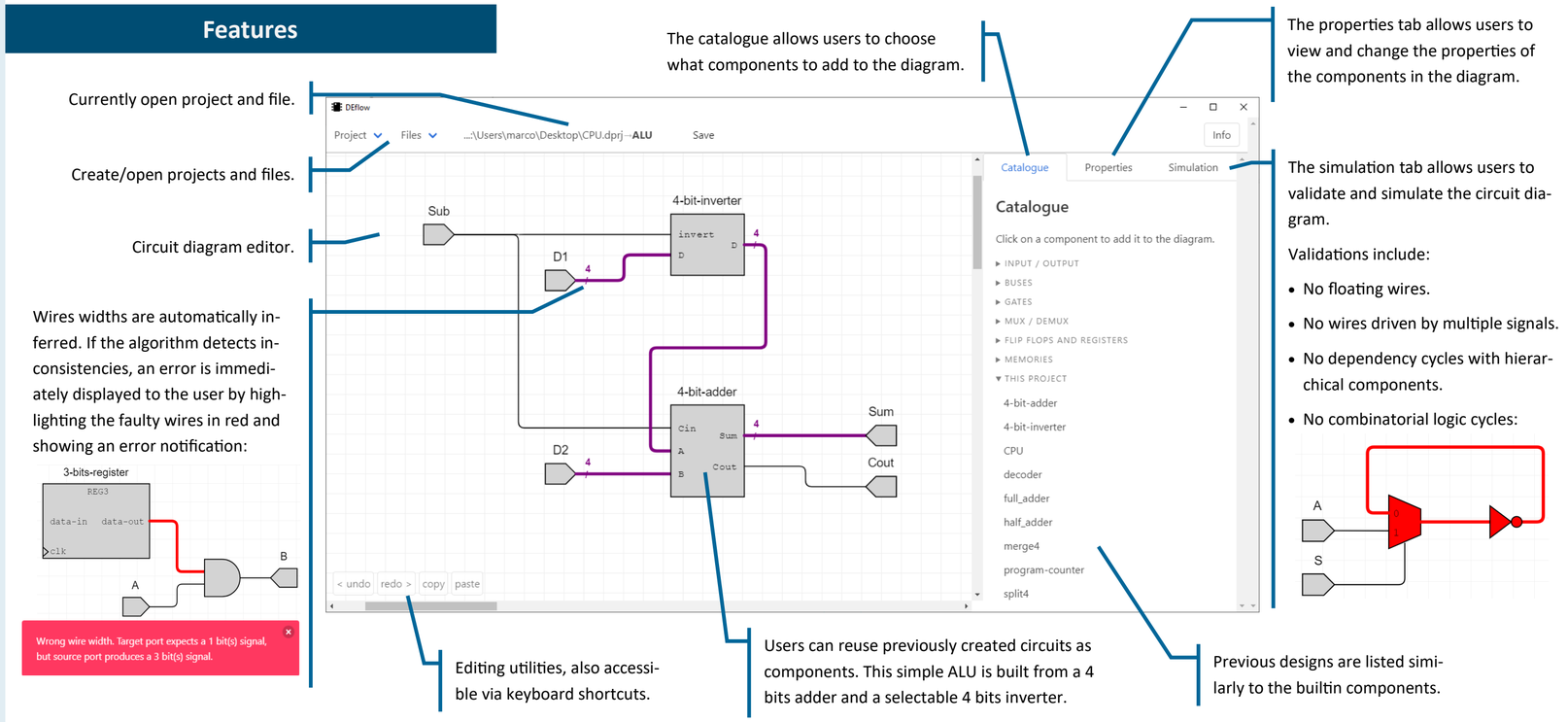


Web Ecosystem: it is cross-platform, commonly known and increasingly adopted. **Electron** allows developers to deploy cross-platform desktop applications developed with HTML, CSS and JavaScript. Electron uses Node.js to access OS APIs like the file system ones.

Functional Programming in F#: F# has a powerful type checker that provides strong code correctness guarantees. Furthermore, functional code is easy to test and has no unexpected side-effects. **Fable** allows developers to compile F# code to JavaScript, which can be run on Electron.

Functional Reactive Programming: this is a declarative programming paradigm used to design highly responsive and maintainable UIs. The F# **Elmish** library implements this with the Model-View-Update (MVU) architecture. MVU allows developers to split the interface design into a series of independent views, which makes maintaining and extending the application easier.

Features



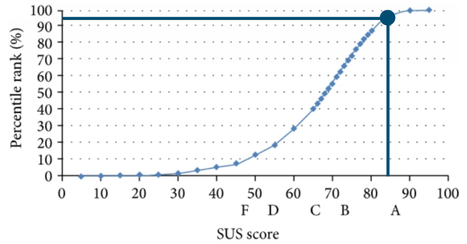
Evaluation

DEflow has been evaluated by collecting user feedback among university students, by comparing it with alternative software and by using it to design a RISC CPU.

Intuitiveness

Users were required to complete 5 tasks, while being given **minimal guidance**. In the majority of cases, first-time-users were able to complete tasks in **less than five minutes**.

Task	Less than 5 minutes	Less than 10 minutes	Less than 15 minutes	More than 15 minutes
AND gate	100	0	0	0
half-adder	88.9	11.1	0	0
full-adder	55.6	33.3	0	11.1
ROM	66.7	33.3	0	0
2 bits register	77.8	11.1	11.1	0



System Usability Scale (SUS)

SUS is an industry standard to measure usability of digital products. A score of 68 is average, a score above 80 is excellent.

Users answers yielded a SUS score of **84.57** for DEflow.

Performance

Latencies for all DEflow algorithms are orders of magnitude below the threshold of human perception. Hence, the application always feels **highly responsive**.

Algorithm or group of algorithms	Mean Latency (ms)	Standard Deviation (ms)
Wires width inference	2.2	0.7
Diagram validations	3.6	1.1
Feed combinatorial signal	5.6	0.3
Feed clock tick	29.2	6.8
Cycle detection precalculation	2.5	0.9
Cycle detection	4.6	1.9

Error Messages Quality

100% of the users agreed that DEflow's error messages **guided them well** towards the resolution of their issues.

Scalability

DEflow was successfully employed to design, validate and simulate a **RISC CPU** in less than an hour.

Code Extensibility

High Degree of Modularisation: the codebase is modularised to ensure changes in a module do not affect the others.

Descriptive Data Types: type definitions precisely map the data domain being modelled and allow the F# type checker to spot most bugs.

Clear Modules API: module behaviours are exposed via a set of simple and clear APIs.

Functional Style: the application is mostly written in F#. Functional Reactive Programming allows developers to extend the benefits of functional programming to user interface design.

Conclusion

The extremely positive results obtained in the evaluation indicate that the application is a better alternative than Quartus in the context of first year digital electronics teaching. Future works include the introduction of a waveform generator and the addition of components via drag & drop.