Software Components
AVR Cores: Pure Javascript, C based, simavr
HTML5 UI
Dynamically loaded Port UIs
SPI device emulators
ELF file format interpreter
Native Android components
Dropbox plugin

Build Requirements
Current Android SDK
Emscripten
Linux build-essential

System Design
The overall goal of the project is to enable quality embedded device simulation in a BYOD world. To achieve this goal, an HTML5/Javascript application was originally chosen as the final product given the large ecosystem of simple, unmodified cross-platform support. Over time and given the ability to directly port C applications to the web using Emscripten the choice has been made to target both Javascript and native applications written at least partially in C. UIs are now to be developed in whatever language provides both the best native experience and interoperability on a C based target. To this end Android uses a C backend and a Java based UI, and if supported in the future iOS would have an ObjC/Swift/Cocoa UI. The web based application remains to quickly enable future devices and cover the widest range of devices possible. Noteworthy examples include Chromebooks and Firefox OS devices. For testing purposes, headless versions can be run on a desktop using Node.js, or command line. I do not feel a pressing need to support PCs, as great AVR simulation options are already available.

All of the Software Components listed above are intended to be modified separately with stable interfaces allowing cross communication. The cores are intended to be shared amongst all targeted devices and simulated platforms whereas other components only opportunistically. In the currently form, it is not uncommon for native targets to have elements built in a native UI language, C, and Javascript. UIs can be modified and SPI device drivers added to allow new devices to be simulated. For instance, both the Esplora simulator and the Arduboy simulator use the 32u4 core, but have different software components to represent their screen and port configurations.

Three cores exist with the same interfaces. Overtime I would like to narrow that down to 1 of the 2 C based cores. The pure Javascript, Closure based core has been a great prototype, but is not production material. Each of the C implementations has weaknesses and strengths. The simavr variant works best and is well supported, thus it has taken over as the primary core deployed with the project. The other C based core, avrcore, is a ground up build of my own design. It is only partially complete which is its biggest downside. If completed, it would be a single source file, super portable core that could be optimized and software licensed as needed to best fit the market.

What Works

All of the debugging functionality demonstrated in the Hackaday 2015 video works in the original Javascript based core. This should all be easily ported to one of the more performant cores at an appropriate time. Deciding on and pruning the remaining cores needs to be done before this integration work. Speeds of up to 8Mhz can be reached in a modern desktop browser, 5Mhz is more likely on Android. The best documentation of the struggles and triumphs of the development process can be found in the Gamebuino forum topic regarding RetroMicro, attached below.

Future Goals

It should be obvious that there will be effort to fill the gaps in the What Works section. Another interesting development would be a provision for sound. I plan to continue focusing on flexibility by adding support for more small production devices. I would like to reach 16Mhz to cover a broader set of embedded devices accurately.

https://youtu.be/Tb0fG4G0uQ4
https://github.com/blakewford
https://hackaday.io/project/6873-boardmicro
http://gamebuino.com/forum/viewtopic.php?f=12&t=1381