Simplifying the Programming of Microcontroller-based Devices

Microsoft MakeCode and Lancester University Teams

Anonymous Author(s)

Abstract

Text of abstract ....

Keywords keyword1, keyword2, keyword3

1 Introduction

Microcontrollers, traditionally the workhorses of embedded systems, have become central to efforts in making and education. For example, the Arduino project, started in 2003, created a printed circuit board (the Uno) based on the 8-bit Atmel AVR microcontroller unit that makes most of the its I/O pins available via headers on the board. Hardware modules (shields) may be connected to the main board to extend its capability. The Arduino ecosystem, based on an open hardware design, has grown tremendously in the past 15 years, with the support of companies such as Adafruit Industries and Sparkfun Electronics, to name a few.

What has not changed much in this time is the way these boards are programmed. The C and C++ programming languages are the primary way to program microcontrollers. This is not a huge surprise, given the low-level nature of microcontroller programming, where direct access to the hardware is the order of the day. There generally is no operating system running on such boards, as they have very little RAM (2K for the Uno, for example) and lack memory protection hardware. What is more surprising about the Arduino platform is that:

- it encourages the use of polling by the end-user as the primary way to interact with sensors, which leads to monolithic sequential programs;
- its IDE lacks any code âĂIJintellisenseâĂİ or common interactive features of modern IDEs;
- it loads code onto the microcontroller using 1980s era bootloader technology.

As a result, it is not simple to get started with systems based on Arduino, of which there are many. On the other hand, on the web we find many excellent environments for introducing programming to beginners. Visual block-based editors such as Scratch and Blockly allow the creation of programs without the possibility of syntax errors. HTML and JavaScript allow a complete programming experience to be delivered as an interactive web application, including editing with intellisense, code execution and debugging. (While the Arduino IDE recently has been ported to the web, it lacks many of the above features and requires a web connection to a server which runs a C/C++ compile tool chain to compile user code.) The programming models associated with these environments are generally event-based, freeing the user from the tyranny of polling.

We present a new programming platform that bridges the gap between the worlds of the microcontroller and the web app. The goals of the platform are three-fold:

1. make it simple to program microcontrollers using an interactive web app that works when offline;
2. allow a userâĂŹs compiled program to be easily installed on a microcontroller;
3. support the addition of new of software/hardware components to a microcontroller.

The platform, MCCU, consists of three major components (MakeCode, CODAL and UF2), which we now describe. The MakeCode web app (see www.makecode.com) supports both visual block-based programming and text-based programming using TypeScript, a gradually-typed superset of JavaScript, with the ability to convert between the two representations. The web app supports in-browser execution, via a device simulator, and compilation to machine code, linking against the pre-compiled CODAL C++ runtime to produce a binary for execution on an microcontroller (either 8-bit AVR and 16-bit ARM Thumb instruction set). No C/C++ compiler is invoked for a compilation of user code. The result of compilation is a binary file that is âĂIJdownloadedâĂİ from the web app to the userâĂŹs computer. The USB flashing format (UF2) makes copying of the binary file to the device, mounted as removable flash drive, fast and reliable, across all major operating systems. Once the web app has been loaded, all the above functionality works offline (i.e., if the host machine loses its connection to the internet).

The main innovations of MCCU are:

- The design and implementation of MakeCode, which bridges the worlds of JavaScript and C++, enabling beginners to get started programming microcontrollers from any modern web browser and enabling hardware vendors to innovate and safely add new components to the mix.
- Static TypeScript, a statically-typed subset of TypeScript for fast execution on low-memory devices and

PL'17, January 01–03, 2017, New York, NY, USA
2017.

a simple model for linking against pre-compiled C++; Static TypeScript also can be used to write safe and performant device driver code.
- CODAL, the Component-oriented Device Abstraction Layer, maps each hardware component to one or more software components that communicate over a message bus and schedule event handlers to run non-preemptively on fibers.
- The USB Flashing Format (UF2), a file format designed for flashing microcontrollers over the Mass Storage Class (removable USB pen drive) protocol. This new file format greatly speeds the installation of user programs and is robust to difference in operating systems.

MCCU combines these innovations in programming languages, language runtime, and code loading to make a simple programming experience for the end user. Through its support for Static TypeScript and a foreign function interface to C++, MCCU makes it easy for hardware manufacturers to share their C++ components with a wider audience. All of MCCUâĂŹs components are open source under the MIT license, as detailed below.

MCCU targets can be seen at www.makecode.com, where the MakeCode web app for a variety of boards is available, including the micro:bit (a Nordic nRF51822 microcontroller with Cortex-M0 processor, 16K RAM), AdafruitâĂŹs Circuit Playground Express (CPX: an Atmel SAMD21 microcontroller with Cortex-M0 processor, 16K RAM), and the Arduino Uno (Uno: an Atmel ATmega328 microcontroller with AVR processor, 2K RAM).

We encourage the reader to choose a board and experiment with programming it, using the simulator to explore many of each boardâĂŹs features, to appreciate the qualitative aspects of MCCU: its simplicity and ease of use. In this paper, we will evaluate quantitative aspects of MCCU: compilation speed, code size, and runtime performance. We evaluate:

- the compile time of Static TypeScript compile/link of user code (to machine code) with respect to the GCC-based C/C++ toolchain, as well as the size of the resulting executable;
- the time to load code onto a microcontroller using UF2, compared to standard bootloaders;
- The performance of a set of small benchmarks, written in both Static TypeScript and C++, compiled with the MakeCode and GCC-based toolchains, as well as the performance of device drivers written in Static TypeScript compared to their C++ counterparts.

[evaluate with respect to the popular Arduino toolset, for boards with 8-bit (AVR) and 32-bit (Cortex-M0) microcontrollers. Summary of evaluation]

MCCU is open source on www.github.com. The MakeCode framework is at microsoft/pxt (PXT is previous codename of MakeCode). MakeCode targets for the three previously mentioned boards are at microsoft/pxt-microbit, microsoft/pxt-adafruit and microsoft/pxt-arduino-uno. The latter two targets make use of a common set of MakeCode libraries (packages) at microsoft/pxt-common-packages. [pxt-monaco, pxt-blockly]. Many other MakeCode packages, developed by Microsoft and hardware partners [details later]. A few examples: XYZ.

The rest of this paper is organized as follows. Section 2 presents the design and implementation of the MakeCode framework. Section 3 describes Static TypeScript Overview of the paper: MakeCode, Static TypeScript, CODAL, UF2, Evaluation, Related Work, Conclusion and Future Directions.

A Appendix

Text of appendix ...