Microsoft Touch Develop and the BBC micro:bit

Thomas Ball*, Jonathan Protzenko*, Judith Bishop*, Michał Moskal*, Jonathan de Halleux*, Michael Braun*, Steve Hodges§, Clare Riley°

*Microsoft Research
One Microsoft Way
Redmond, WA, USA

§Microsoft Research
Station Road
Cambridge, UK

°Microsoft
Education Relations
London, UK

tball, protz, jbishop, micmo, jhalleux, v-braum, shodges, clarer @microsoft.com

ABSTRACT

The chance to influence the lives of a million children does not come often. Through a partnership between the BBC and several technology companies, a small instructional computing device called the BBC micro:bit will be given to a million children in the UK in 2016. Moreover, using the micro:bit will be part of the CS curriculum. We describe how Microsoft's Touch Develop programming platform works with the BBC micro:bit. We describe the design and architecture of the micro:bit and the software engineering hurdles that had to be overcome to ensure it was as accessible as possible to children and teachers. The combined hardware/software platform is evaluated and early anecdotal evidence is presented. A video about the micro:bit is available at http://aka.ms/bbcmicrobit.

CCS Concepts

• Hardware • Sensor devices and platforms • Applied computing • E-learning • Software and its engineering • Compilers

Keywords

K-12 education; BBC micro:bit; Touch Develop; devices; cloud.

1. INTRODUCTION

Computer scientists are continually looking at new ways to engage and retain the interest of young students in the K-12 years. Recently there have been several waves of new initiatives to engage children aged 8-13 (middle school) in computer science, for example: coding [6], computational thinking [10], games [11], robots [12] and storytelling [2] All of these are successful when led by dedicated and qualified teachers.

The challenge is scaling out an initiative to influence an entire country of students, or even globally. Two significant success at the coding level have been:

1. code.org which initially took up the challenge of getting the K-12 students to code using a variety of online tools, and subsequently has started training teachers in the USA [7].

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4205-6/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889179

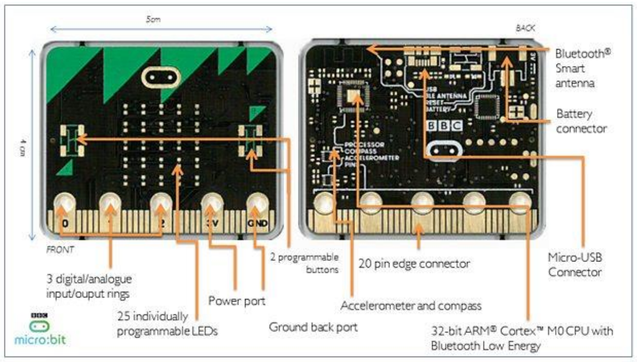


Figure 1 The front and rear of the BBC micro:bit device.

2. CAS (Computing at School) in the UK is an established community of mainly teachers who create curriculum for formal computer science courses nationally [3].

The Microsoft .NET Gadgeteer system provided evidence that students and children are enticed by activities where they can see, touch and change "the computer", in addition to seeing code on a screen [4]. In addition to .NET Gadgeteer, the growth of interest in Arduino, Raspberry Pi and other small computers has been considerable in the developer world. However, rolling these out in schools presents the two challenges of cost and training. Gadgets, internet of things, and the maker culture are all attractive goals, but they need to be at a cost that schools can meet with a low barrier for entry in terms of skills required by teachers.

If a device can be made inexpensive and robust enough to be distributed to millions of children, and if the accompanying software is engaging and intuitive, then there is a chance of making a leap in capturing the minds of an entire generation. At the same time, any such initiative for large scale roll out should acknowledge that children learn and grow up very fast, so that a progression of tools within the same basic platform is highly desirable.

Taking up this challenge in the UK, a multi-partner initiative led by the BBC is providing a million small programmable devices – BBC micro:bits – to UK middle schoolers in 2016. We describe the device, the Microsoft-developed programming environment, some of our design considerations, and early evaluation results. There is also an accompanying video[1].

2. THE BBC micro:bit

The BBC micro:bit[2] is a pocket-sized, codeable computing device, designed to allow children to get engaged and creative with technology. The BBC announced the micro:bit on July 7, 2015, teacher training started in August 2015, and devices were made available to schools from February 2016.

[1] http://aka.ms/bbcmicrobit

[2] https://www.microbit.co.uk

The micro:bit measures 4cm by 5cm and is designed to be fun and easy to use. It is visually appealing, and comes in four different colour variants. Simple programs can be coded in seconds – like lighting up its LEDs to display a pattern – with little prior knowledge of computing. The emphasis is on engagement, imagination and creativity.

The micro:bit is powered by an ARM Cortex-M0 processor[3] and has 256K non-volatile flash (for program and static data) and 16K volatile RAM (for stack, heap). Key features of the micro:bit include:

- **25 red LEDs** to scroll text, create games and invent digital stories.
- **Two programmable buttons** to provide input.
- **An on-board accelerometer motion detector** to detect forces acting on the device.
- **A built-in magnetometer (compass)** to sense direction.
- **Bluetooth Smart Technology** to connect to other micro:bits and devices such as phones and tablets.
- **A temperature sensor** for measuring ambient temperature.
- **Basic light-level sensing** via the LED display.

Through its hybrid expansion port, which combines a standard edge connector with three 4mm Input and Output (I/O) rings, the micro:bit can connect to other sensors, actuators and devices. It is intended as a companion rather than a competitor to devices such as Arduino, Galileo, Kano, littleBits and Raspberry Pi, acting as a spring-board to more complex learning.

3. PROGRAMMING THE micro:bit

To bring the BBC micro:bit to life, we developed an enhanced version of the Microsoft Touch Develop[4] web application and hosting service. A dedicated website (www.microbit.co.uk) allows users to choose from a range of online code editors available from most modern web browsers (no extra client-side software or device driver installation is needed, a huge benefit to school systems).

Microsoft supplied two languages/editors – Touch Develop, a semi-structured text-based language, and the Block Editor, a graphical coding language much like Scratch. The Touch Develop web app supports all the code editors built for the micro:bit, runs the micro:bit simulator, and compiles programs to ARM machine code. Advanced users can use C++ to directly program against the run-time system and interface with not-yet-developed hardware.

3.1 Languages, Editors, Compilers

We extended Touch Develop to support a *progression* of languages with accompanying browser-based editors. The Block Editor provides an introduction to structured programming via blocks that can be snapped together (Figure 2). Unlike other similar offerings [5] [9], Touch Develop's Block Editor is strongly-typed and the programs convert seamlessly to a next level of complexity in learning [8].

Touch Develop's classic mode shown in Figure 4 features a statically-typed scripting language with syntax-directed editor. The language subset contains: while and for loops; if-then-else conditional statement; functions; local and global variables; integer, boolean, string and image types; operations over values of the above types; user-defined event handlers and libraries.

Browser-based compilers from the Block Editor to Touch Develop and then to ARM assembly and machine code automate the transition from a visual language to a text-based language, and then to binary language of the ARM-based micro:bit. The first compiler allows a student to convert a Block Editor script into a Touch Develop script with a single press of a button (Figure 2).

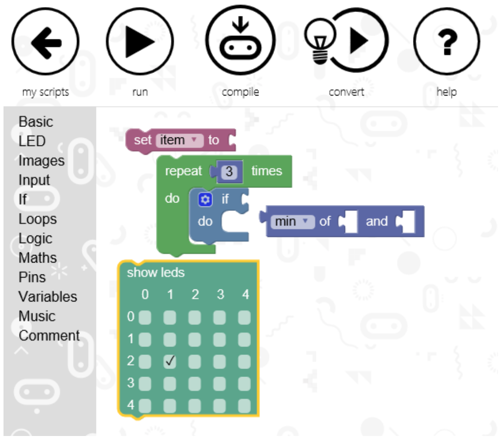


Figure 2. The Microsoft Block Editor for BBC micro:bit.

3.2 Compile and Flash

Figure 3 shows the flow of compilation for the micro:bit. When a student has her Block Editor or Touch Develop script (Step 1) ready, she can connect her micro:bit to a computer via a USB cable, whereupon it appears as a mounted drive. Compilation from Touch Develop to a micro:bit binary proceeds within the confines of the web browser. The student is prompted to save the ARM binary program to a file (Step 2) which she then simply drags to the micro:bit mounted drive. This flashes the micro:bit (Step 3) with the new program and starts running it.

In more detail, the Touch Develop script is first compiled to ARM Thumb assembly code, which is then translated into machine code. The machine code is then injected into a pre-compiled runtime file (which conveniently uses a text-based hex format). The runtime file comes with metadata specifying the addresses of runtime functions, which helps with linking. The in-browser assembler handles the entire Thumb instruction set even though the compiler only generates around 12 of these, which allows users to write inline assembly in their Touch Develop scripts if they so desire.

3.3 The Simulator

Before a student compiles her script for the micro:bit, she can run it using the Touch Develop micro:bit simulator, all within the web browser. The simulator supports the LED screen, buttons, compass, accelerometer, and digital I/O pins. To run a student's Touch Develop script in the web browser, Touch Develop compiles it into JavaScript, the scripting language built into all web browsers.

The working of the simulator can be seen in the following example of the Rock-Paper-Scissors game (Figure 4). The code in Touch Develop is shown on the left and a picture of the device on the right. In the simulator, either the accelerometer of the phone or tablet, or the mouse pointer of a desktop computer is substituted for the micro:bit accelerometer. The example also illustrates how succinct Touch Develop programs are. In particular, output to the LED display can be easily created on the simulator, as is shown in Figure 2 and illustrated in the fourth program line. It is not necessary to reference individual LEDs by number on a grid.

[3] http://www.arm.com/products/processors/cortex-m/cortex-m0.php

[4] http://www.touchdevelop.com

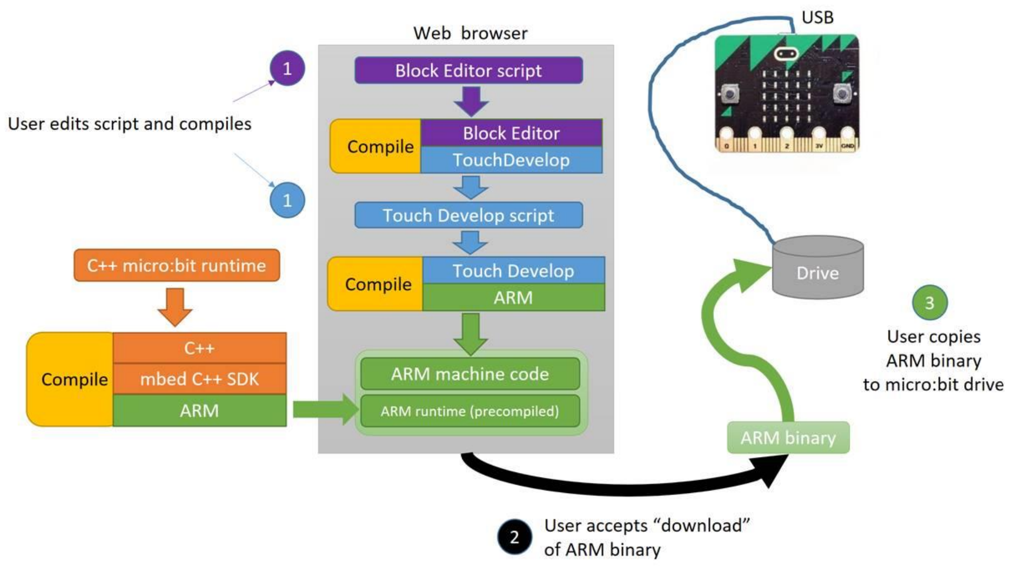


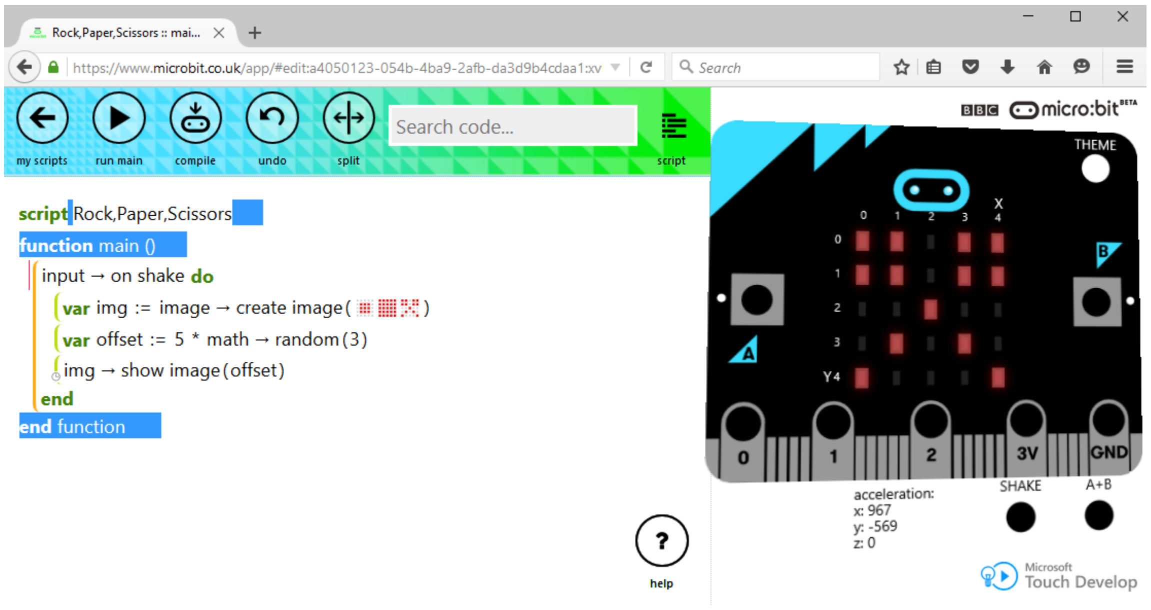
Figure 3 The compilation flow for the micro:bit

3.4 The Libraries

The micro:bit runtime and libraries are programmed in C++. These provide access to the hardware functions of the micro:bit, as well as a set of helper functions, such as displaying a number/image/string on the LED screen. The Touch Develop micro:bit library mirrors the functions of the C++ library. When a Touch Develop script is compiled, the calls to Touch Develop micro:bit functions are replaced with calls to the corresponding C++ functions. The C++ run-time system exposes hardware features and takes care of low-level tasks, such as:

- multiplexing the LEDs;
- managing an event bus for inter-thread communication;
- "de-bouncing" the buttons, that is, making sure only one "button pressed" event is generated per press;
- setting up a lightweight thread scheduler;
- abstracting away the specific communication protocols of the on-board accelerometer, thermometer and magnetometer.

Touch Develop is strongly typed, which is good because well-typed Touch Develop programs generate well-typed C++ programs, hence alleviating the need for a dynamic type mechanism. Given the underlying constraints and the small amount of memory, dynamic types would likely put a low limit on the size of programs supported.



```
script Rock,Paper,Scissors
function main ()
  input → on shake do
    var img := image → create image( )
    var offset := 5 * math → random(3)
    img → show image(offset)
  end
end function
```

```
acceleration:
x: 967
y: -569
z: 0
```

Figure 4 The micro:bit simulator

Touch Develop, in its traditional JavaScript code generation scheme, is garbage-collected. We did not implement a garbage collector for the micro:bit; rather, we rely on reference-counting. We claim that this is "good enough" for an embedded device; by the time users generate cyclic data structures, they will likely be advanced enough to switch to C++ and handle memory themselves.

Users can define records with fields in Touch Develop ("objects"); we map these to ref-counted C++ structs. Users can write event handlers that capture local variables; we use C++11's lambda-capture for that purpose.

4. DESIGN CONSIDERATIONS

There were many design considerations to ensure ease of use, hardware safety, privacy in the cloud, authorization and ultimately the roll-out of micro:bit devices at such a large scale. These topics will be covered in another paper.

Providing young students with an understandable programming model was a challenge, and it took us several iterations to evolve the final semantics. For instance, students can use a busy-loop, active-polling programming model that is easier to understand and reason about, but does not scale well to larger programs. The alternative is an evented, reactive-based programming model that relies on cooperative threading. Naturally, this latter model comes with its own set of problems, such as making sure students' code yields often enough and defining clear semantics for handling button events. We mitigate these difficulties using a variety of mechanisms, such as warnings in the simulator if a loop has been visited a great number of times.

5. EVALUATION

We have been collecting and evaluating data on Touch Develop since 2011 [1] based on over half a million users and over 150,000 scripts. The median size for scripts was found to be 24 lines, but up to 100 lines was common. The target audience of the micro:bit is children who will write short scripts, making heavy use of libraries to accomplish complex tasks. With the more customized libraries (as described in Section 3.4) we estimate that micro:bit users will create satisfying apps in less than 50 lines.

Our prime success metrics include ease of use and robustness, which we can only measure once the micro:bits are in the field. To date, we have trained 600 teachers in the UK prior to the launch and are working hard to incorporate the valuable feedback we received.

6. RELATED WORK

It is natural to compare the micro:bit to other device-based coding experiences such as Arduino and the Raspberry Pi. A variety of metrics such as processing capacity, supported I/O and programming environment are useful for this. The micro:bit has onboard sensors, buttons and LEDs to be used as a physical computing device.

The micro:bit features an ARM Cortex-M0 processor which is more powerful than the AVR in the standard Arduino, meaning that valid C++ programs can run unmodified on the micro:bit using the GCC toolchain. However, the micro:bit is less powerful than a Raspberry Pi, which is a complete computer that runs a fully-fledged operating system. The micro:bit runs one program at a time.

The software platform for the micro:bit has some similarities to Pocket Code [9] and App Inventor [5]. The advantages of Touch Develop are its progression for lifelong coding, as discussed in Section 3.1, and its platform independence. Considerable thought went into a seamless linking of the software to the hardware, providing a unified experience, which is essential for children and teachers who are new to coding.

7. ACKNOWLEDGMENTS

Nikolai Tillmann was the inspiration behind the Touch Develop system. A large number of individuals within the BBC micro:bit partnership worked tirelessly to bring the micro:bit to fruition and we are indebted to them all. In particular Joe Finney and James Devine at the University of Lancaster developed the runtime system for the micro:bit and Jonathan Austin from ARM enabled the micro:bit device programming experience.

8. REFERENCES

[1] Thomas Ball, Sebastian Burckhardt, Jonathan de Halleux, Michał Moskal, Jonathan Protzenko, and Nikolai Tillmann, Beyond Open Source: The TouchDevelop Cloud-based Integrated Development Environment, MOBILESoft, 83 - 93, 2015.

[2] Quinn Burke and Yasmin B. Kafai. 2012. The writers' workshop for youth programmers: digital storytelling with scratch in middle school classrooms. SIGCSE Technical Symposium, 433-438, 2012.

[3] CAS: http://www.computingatschool.org.uk/

[4] Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil, and Steven Johnston. 2013. .NET gadgeteer: a new platform for K-12 computer science education. SIGCSE Technical Symposium, 391-396, 2013.

[5] J. Liu, C.-H. Lin, P. Potter, E. P. Hasson, Z. D. Barnett, and M. Singleton, Going mobile with App Inventor for Android: a one-week computing workshop for K-12 teachers, SIGCSE Technical Symposium, 433–438, 2013.

[6] Orni Meerbaum-Salant , Michal Armoni , Mordechai (Moti) Ben-Ari, Learning computer science concepts with Scratch, Computer Science Education , Vol. 23, Iss. 3, pp239-264, 2013

[7] Hadi Partovi. 2015. A comprehensive effort to expand access and diversity in computer science. ACM Inroads 6, (3) 67-72, 2015

[8] J. Protzenko, Pushing Blocks All The Way To C++, In Blocks and Beyond Workshop, Atlanta, Georgia, 2015

[9] W. Slany, A mobile visual programming system for Android smartphones and tablets, VL/HCC, 265–266, 2012

[10] Amber Settle, Baker Franke, Ruth Hansen, Frances Spaltro, Cynthia Jurisson, Colin Rennert-May, and Brian Wildeman. 2012. Infusing computational thinking into the middle- and high-school curriculum. ITiCSE, 22-27, 2012

[11] Linda Werner, Shannon Campe, and Jill Denner. Children learning computer science concepts via Alice game-programming. SIGCSE Technical Symposium, 427-432, 2012.

[12] Teruya Yamanishi, Kazutomi Sugihara, Kazumasa Ohkuma and Katsuji Uosaki, Programming instruction using a micro robot as a teaching tool, Computer Applications and Engineering Education, 23, (1), 109–116, 2