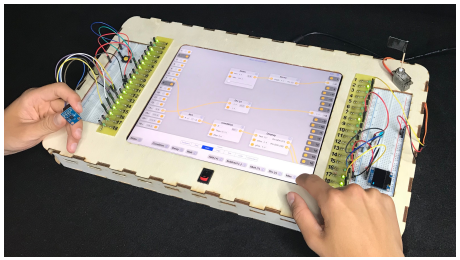


Flowboard equipped with various components

Figure 1: Flowboard's hardware frame combines a large iPad Pro with an Arduino board beneath it, and one breadboard left and one right of the iPad. The user develops her embedded code using Flowboard's visual flow-based programming app on the iPad. Electronic components for sensing (input, left) and actuation (output, right) are plugged into the breadboards and linked seamlessly to processing nodes on the screen via two duplicate rows of I/O pins. Parallel processes are easy to code: Here, a physical button controls what is written to a serial OLED display, a force sensor dims an LED, an on-screen slider controls a servo, and a 3-axis accelerometer is connected just to see its sensor values live.

Flowboard: A Visual Flow-Based Programming Environment for Embedded Coding

Anke Brocker, Simon Voelker, Tony Zelun Zhang, Mathis Müller, Jan Borchers
RWTH Aachen University
52056 Aachen, Germany
brocker@cs.rwth-aachen.de,voelker@cs.rwth-aachen.de,tony.zhang@.rwth-aachen.de,
mathis.mueller@.rwth-aachen.de,borchers@cs.rwth-aachen.de

ABSTRACT

With Maker-friendly environments like the Arduino IDE, embedded programming has become an important part of STEM education. But learning embedded programming is still hard, requiring both coding and basic electronics skills. To understand if a different programming paradigm can help, we developed *Flowboard,* which uses *Flow-Based Programming* (FBP) rather than the usual imperative programming paradigm. Instead of command sequences, learners assemble processing nodes into a graph through which signals and data flow. Flowboard consists of a visual flow-based editor on an iPad, a hardware frame integrating the iPad, an Arduino board and two breadboards next to the iPad, letting learners connect their visual graphs *seamlessly* to the input and output electronics. Graph edits take effect immediately, making Flowboard a *live coding* environment.

CHI'19 Extended Abstracts, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5971-9/19/05.
https://doi.org/10.1145/3290607.3313247

KEYWORDS

Embedded Development Environments; Visual Flow-Based Programming; Electronics; Young Learners

INTRODUCTION

Embedded development environments like the Arduino IDE have greatly lowered the threshold to get involved in the maker movement [7]. They enable makers to create interactive artifacts, and have become a main ingredient of STEM education [8]. However, learning embedded development is inherently challenging, requiring an understanding of (a) basic electronics, (b) coding, and (c) the connections of hardware and software of a project [6]. A number of research projects have tackled the first aspect, we set out to address the other two facets (b and c). The traditional *imperative programming* paradigm is widespread and the default in embedded development environments, including Arduino. Users need to type in their program as a sequence of programming statements, which requires knowing the language and is prone to syntax errors. *Block-based* programming environments like Scratch[1] replace textual source code with a graphical editor to assemble code from visual programming blocks.

We wanted to explore if the *flow-based programming* paradigm can provide a better starting point to learn embedded development. In FBP, programs are not sequences of commands, but a network of processing nodes through which data flows. Each node changes the data based on its parameters. This paradigm closely resembles electronic signal processing circuits. Unlike in imperative programming, parallel processes in one program are straightforward [3]. FBP has been used in commercial IDEs in many domains, from scientific experimentation (LabVIEW[2]) to interactive media systems (Max/MSP[3]). FBP-based embedded development environments such as Microflo[4] and XOD[5] have been available for a few years. TAC[1] lets user specify their circuit in a FBP graphical editor, then presents the user possible physical circuit designs and matching embedded textual source code. These systems are missing two aspects that motivated us to design and build our own hardware and software:

Liveness: We made use of a key opportunity of FBP for embedded development: program graphs can process incoming data and reflect changes directly. These live programs, like analog circuits, can respond immediately to incoming electronic signals, without an Edit-Compile-Run cycle.

Seamlessness: The points where electrical signals flow into and out of the program graph have a direct correspondence both in the visual FBP graph and, as I/O pins, in the hardware circuit, as *Flowboard* places these virtual and physical pins directly next to each other.

Liveness and seamlessness fit the visual FBP paradigm for embedded development naturally: A graph is easy to consider always-active, and the hardware/software boundary are simply particular edges in the graph. In imperative programming environments, it is less trivial to create liveness [4], and the interface to hardware is represented in statements that are spatially dispersed by the code, making it more challenging to create a seamless view [5].

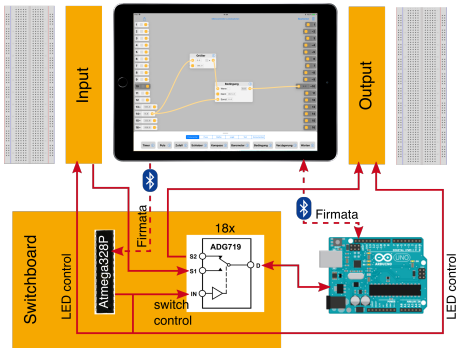
[1] https://scratch.mit.edu

[2] https://ni.com/labview

[3] https://cycling74.com/products/max

[4] http://microflo.org

[5] https://xod.io



Flowboard's system architecture

Figure 2: Flowboard system architecture. The Arduino pins are brought out to the input and output side of the Flowboard via the switchboard, which controls the connection to either side. The iPad runs the visual flow-based editor, and communicates with both the Arduino and the microcontroller on the switchboard via Bluetooth using the Firmata protocol.

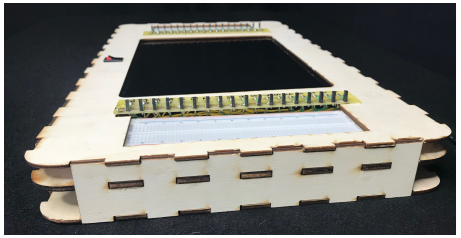


Figure 3: The Flowboard case contains several layers to house the iPad, the custom circuit board, the Arduino, and a 5V battery pack.

[6]https://github.com/firmata/protocol/blob/master/protocol.md

SYSTEM DESIGN AND IMPLEMENTATION

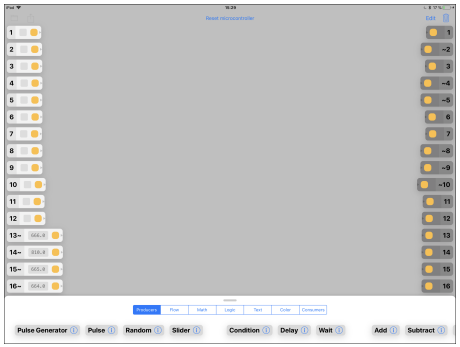
The user creates her program graph using a visual, flow-based multitouch editor on a 12.9" iPad Pro lying flat on the table. This allows for a system design with a seamless link between electronics and program graphs. Touch-based interfaces also support more natural interactions that can support learning [2]. Flowboard contains an Arduino Uno board and a custom printed circuit board that also holds the "switchboard": a second microcontroller and 18 electronic switches. The Flowboard app is a native iOS app. The iPad editor talks to the Arduino and the switchboard controller via Bluetooth. The Arduino is running our modified version of the Firmata[6] protocol. Firmata allows the iPad editor to set and read the Arduino pins through serial commands sent via Bluetooth. We extended the protocol to cover additional electronic components. With a real-time protocol like Firmata Flowboard is a live system as the iPad editor interprets the graph continuously, sending Firmata commands to the Arduino to achieve the appropriate behavior. We provide all files here: https://github.com/i10/Flowboard.

Flowboard Hardware

The user has access to all Arduino's I/O pins twice, once on each side of the iPad, except the two pins required used for Bluetooth. If a user connects to a virtual in- or output pin in the editor, it instantly instructs the switchboard controller that enusres that each Arduino pin is connected to either the input or output row. Pins are Always active and detect plugged in components automatically. The Flowboard, and also its breadboards, can be powered through a battery pack in the frame or via the Arduino USB connector. Below the screen, a hardware toggle switch lets the learner disconnect power from the breadboards to reduce the risk of short circuits while building them. The Flowboard case was lasercut out of plywood. Its bottom layer contains the custom circuit board, switchboard controller, cables, and the battery pack. The middle layer serves as platform for the breadboards and the iPad. The top layer contains the breadboards, the external pin row connectors, a power switch, and the iPad. The side walls hold all three layers together (Fig. 3).

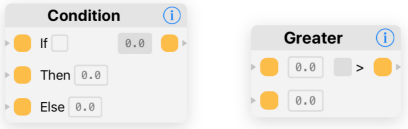
Visual Editor

The visual editor initially shows an empty canvas to place nodes on. The user drags nodes onto the canvas from the node menu, and connects them by drawing virtual wires between them. Both sides of the editor show virtual representations of the input resp. output pins, aligned with the hardware pins on each side of the iPad. A virtual output pin is greyed out if something is connected to the physical pin at the input side and vice versa. Active pins show a green LED and are not greyed out on screen. To maximize screen space for the program graph, the node menu on the bottom is only visible while adding nodes. Scrolling sideways through it shows all available nodes, with direct shortcuts



Flowboard's IDE

Figure 4: Flowboard's editor after launching. Virtual I/O pins are visible on the left and right. The bottom menu is only visible while the user picks a node to add to his program



The programming nodes

Figure 5: These two programming nodes ('Greater Than' and 'Condition') are examples of the nodes used on the iPad

for experienced users. The node menu includes nodes for basic mathematical and logical processing functions as well as nodes to work with more complex electronic components, such as servo motors.

First Evaluation

After designing Flowboard we gathered feedback from learners concerning layout and design of the Flowboard editor. We target programming beginners, that can have programming experiences with Scratch etc. With these, we improved its UI, such as highlighting matching inputs based on input type when connecting nodes. Furthermore, we tested the study design for exploring our research question if FBP would make embedded programming more accessible to learners. We found that a 1-on-1 tutorial with guided tasks is more suitable than letting the user read a handbook without guidance for the first steps in the new system. In the future, we would like to run a user study comparing our system to an imperative programming environment and evaluate our underlying research question.

CONCLUSION

Our *Flowboard* prototype uses FBP and is a live and seamless device for embedded development. It needs to be tested more in terms of suitability for learning embedded development. We hope to gather feedback concerning its design and functionality during the demonstration. With this feedback we plan to evaluate and improve Flowboard further as a device for learners.

REFERENCES

[1] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proc. UIST '17*. ACM, 331–342. https://doi.org/10.1145/3126594.3126637

[2] Eva Hornecker, Paul Marshall, Nick Sheep Dalton, and Yvonne Rogers. 2008. Collaboration and Interference: Awareness with Mice or Touch Input. In *Proc. CSCW '08*. ACM, 167–176. https://doi.org/10.1145/1460563.1460589

[3] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. 2004. Advances in Dataflow Programming Languages. *ACM Comput. Surv.* 36, 1 (March 2004), 1–34. https://doi.org/10.1145/1013208.1013209

[4] Jan P. Krämer, Joachim Kurz, Thorsten Karrer, and Jan. Borchers. 2014. How live coding affects developers' coding behavior. In *VL/HCC '14*. 5–8. https://doi.org/10.1109/VLHCC.2014.6883013

[5] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and Checking Behavior of Embedded Systems Across Hardware and Software. In *Proc. UIST'17*. ACM, 299–310. https://doi.org/10.1145/3126594.3126658

[6] Will McGrath, Jeremy Warner, Mitchell Karchemsky, Andrew Head, Daniel Drew, and Björn Hartmann. 2018. Wifröst: Bridging the Information Gap for Debugging of Networked Embedded Systems. In *Proc. UIST '18*. ACM, 447–455. https://doi.org/10.1145/3242587.3242668

[7] S. Papavlasopoulou, M. Giannakos, and Maria Letizia Jaccheri. 2012. Creative and Open Software Engineering Practices and Tools in Maker Community Projects. In *Proc. EICS '12*. ACM, 333–334. https://doi.org/10.1145/2305484.2305545

[8] S. Papavlasopoulou, M. Giannakos, and Maria Letizia Jaccheri. 2017. Empirical studies on the Maker Movement, a promising approach to learning: A literature review. *Entertainment Computing* 18 (2017), 57 – 78. https://doi.org/10.1016/j.entcom.2016.09.002