

Article

KidBright: An Open-Source Embedded Programming Platform with a Dedicated Software Framework in Support of Ecosystems for Learning to Code

Surapol Tan-a-ram , Anuchit Leelayuttho , Somphong Kittipiyakul, Wuthipong Pornsukjantra, Thundluck Sereevoravitgul, Apichart Intarapanich, Saowaluck Kaewkamnerd and Dusadee Treeumnuk * 

National Electronics and Computer Technology Center (NECTEC), National Science and Technology Development Agency (NSTDA), Pathum Thani 12120, Thailand

* Correspondence: dusadee.treeumnuk@nectec.or.th



Citation: Tan-a-ram, S.; Leelayuttho, A.; Kittipiyakul, S.; Pornsukjantra, W.; Sereevoravitgul, T.; Intarapanich, A.; Kaewkamnerd, S.; Treeumnuk, D. KidBright: An Open-Source Embedded Programming Platform with a Dedicated Software Framework in Support of Ecosystems for Learning to Code. *Sustainability* **2022**, *14*, 14528. <https://doi.org/10.3390/su142114528>

Academic Editor: Luis Hernández-Callejo

Received: 9 September 2022

Accepted: 31 October 2022

Published: 4 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Abstract: The concept of coding at school has enabled educators and parents around the globe to become interested in coding as, these days, coding is regarded as a gateway to computational thinking for children. However, coding education in Thailand appears to lag behind that of many other countries due to the lack of accessible coding learning resources suitable for students as well as the limited number of teachers with coding experience. Regarding these issues, we have developed an open-source hardware-based coding platform named KidBright, based on these requirements: 1. making coding simple for novice learners through the use of graphical blocks with Thai and English support; 2. inspiring young students to develop creative embedded applications with minimal effort; and 3. providing sustainable support for coding education. KidBright is proposed as a coding learning tool that can motivate children to learn to code and develop embedded system projects using its block-based coding environment, the KidBright IDE, in conjunction with its embedded device, the KidBright board. KidBright is distinguished from other embedded programming platforms due to the deployment of a dedicated software framework as the backend of the KidBright IDE. In this article, we introduce KidBright and present the design, architecture, and demonstrations of the software framework, the key roles of which are to conceal low-level hardware issues from learners and to enable makers to develop new command blocks and hardware peripherals in support of KidBright. We claim that, with this particular design, KidBright can help support coding education sustainably. In particular, we present how a small research team introduces coding to a large number of learners who have little or no coding experience, resulting in an impact on coding education in the country.

Keywords: coding; coding education; block-based coding; embedded programming; embedded systems; microcontroller; software framework

1. Introduction

In the 21st century, digital technology has become part of human life. Currently, the use of digital technology is quickly growing, in expectation of improving quality of life, while the development sector is still searching for new innovators. In preparation for human resources, coding in school has emerged as a new educational concept that introduces computer coding to children. In particular, the fundamental coding concept has been introduced in early childhood education [1–5], and coding has been regarded as another language that teaches children not only computer programming but also how to think logically and systematically [6,7]. The concept of introductory coding has been gaining a lot of attention across the globe, and coding has been adopted as an important subject in elementary school curricula in many countries [8,9], especially those in North America and Europe [3,10].

Coding used to be taught only in technology schools such as engineering and computer science institutions. Today, learning to code is ubiquitous, and various options for coding

environments have been made available to learners. However, due to the fact that children's interests are based on fun rather than knowledge, suitable coding tools must be able to engage them [11,12]. One promising approach is to use block-based coding environments where learners can simply develop code by visually assembling graphical drag-and-drop instruction blocks. Unlike text-based coding, this coding approach allows children to focus on logical flows rather than complicated syntax, which is prone to human errors. For example, Scratch and Code.org offer block-based coding environments in which learners can simply create games, interactive stories, and animations by means of visual instruction blocks [13,14].

In particular, learning to code in conjunction with computing hardware allows children to build embedded system projects based on their creativity. It is well known that Arduino and Raspberry Pi are computing hardware platforms that help to reduce the amount of work needed to develop embedded systems due the large amount of support from their worldwide communities. However, learning to code using them is still not simple enough, as their use is concerned with low-level hardware and text-based programming languages, which are too much for young beginners. Much simpler examples are Lego Mindstorms [15] and micro:bit [16] which offer educational toys that can be equipped with hardware devices and programmed to perform embedded applications through their block-based coding environments. In this direction, introducing coding to children through the use of embedded devices can significantly benefit them beyond fundamental coding as they have the opportunity to apply coding to real-world implementations of embedded systems. Thus, children can play as well as experience embedded programming, leading to reinforced skills in science, technology, engineering, and mathematics (STEM) [17,18].

Coding education in Thailand is far behind that of many countries due to the lack of accessible coding tools and learning resources suitable for Thai students, who are significantly hindered by the fact that English is their second language, and an insufficient number of teachers who can teach coding. Motivated by these matters, we have developed a coding learning platform named KidBright, based on the following requirements: (1) coding simplification for beginners through a block-based coding environment with Thai and English support; (2) a methodology that eliminates problematic embedded programming issues for inexperienced learners and allows the development of practical embedded systems with minimal effort; (3) sustainable usage supported by expandable device connectivity and an approach for the development of external instruction blocks.

Specifically, the KidBright is an open-source embedded programming platform that includes three components, namely, the KidBright board, the KidBright Integrated Development Environment (KidBright IDE), and the KidBright simulator (KidBright has also been acknowledged as a brand as well as an ecosystem, representing various educational tools created by the KidBright team). The KidBright board is a compact microcontroller-based device equipped with electronic components such as an LED matrix display, environmental sensors, switches, a buzzer, and so on. A KidBright board can be programmed to perform dedicated tasks through the use of the KidBright IDE, which provides a friendly block-based coding environment based on Google Blockly [19]. Thus, children can write code on the KidBright IDE by means of drag-and-drop instruction blocks. Furthermore, with the KidBright simulator, the learner can compose block-based code in the same way as with the KidBright IDE, except that the result corresponding to the code simulated is virtually displayed on the simulator. Both the KidBright IDE and the KidBright simulator support Thai and English languages, which can be simply switched by the learner. With the KidBright, children can learn not only coding but also STEM in support of today's technology-driven economy. The KidBright can inspire children to create unlimited ideas for STEM projects by deploying the on-board components and/or external ones. At this point, it is noted that the KidBright supports well-known wired and wireless interfaces such as I/Os, I2C, SPI, UART, Bluetooth, WiFi, and IoT. In particular, with the IoT feature, children can program their KidBright boards to sense and collect data from anywhere in the world.

In embedded programming, learners do not only learn how to code but also how to control hardware peripherals with code. Therefore, the KidBright IDE must provide access to a variety of hardware through its block-based coding environment. We have developed a software framework called the KidBright Engine as the backend of the KidBright IDE, which invisibly handles hardware interfaces in support of learners such that they can focus on coding flows rather than low-level hardware issues, e.g., bit-level configurations, device read and write, memory management, latency control, and so on. In particular, the KidBright Engine is designed to support expandable device connectivity and the development of new instruction blocks, both of which are key features that support learning-to-code ecosystems where learners and developers can support each other with more hardware peripherals along with new instruction blocks and learning resources provided on their own. Despite the fact that some other embedded coding platforms provide these features, those provided by the KidBright IDE rely on the KidBright Engine, of which the design and architecture are different, as presented and discussed in this manuscript.

In this article, our contributions are as follows: we present the use and design of the KidBright, which is made available as an open-source platform for learning embedded programming. Our software, schematic diagrams, and supporting resources can be found at the KidBright website, www.kid-bright.org (accessed on 29 August 2022). In support of developers, our source code and development information are provided on GitLab, <https://gitlab.com/kidbright> (accessed on 29 August 2022). Although the KidBright shares the concept of using graphical blocks to ease coding with other coding tool companions, it has dedicated features according to the design requirements. We explain in detail the designed architecture of the KidBright's software framework and hardware interfacing paradigm, which help to reduce learners' learning workloads as well as to grow ecosystems for learning embedded programming. We demonstrate the effectiveness of the proposed design in the context of software and hardware through empirical investigations. To promote and sustain coding education, we also contribute to a strategy for introducing coding to a large number of learners who have little or no coding knowledge, allowing the use of the KidBright for coding education to continue to grow on its own.

The rest of the article is organized as follows: Section 2 reviews related works focusing on block-based coding environments for embedded programming. In Section 3, the KidBright and its components are introduced. Section 4 presents the KidBright's design and architecture in the context of hardware and software that promote and sustain coding education. In Section 5, demonstrations of the proposed software framework in supporting embedded programming and sustainability are presented. Section 6 describes the dissemination strategy and how it helps introduce coding nationwide. Section 7 discusses the methods and results with respect to the software framework and the dissemination strategy. Finally, Section 8 concludes all the work presented.

2. Related Work

In the past few decades, it can be said that computer programming has been reserved as a job for specialists, because the limited options of programming languages and development environments defined a huge barrier to the interest of people outside of the field. Computer science today is more advanced than it was in the past, and people around the world realize how important computer coding is to their children in this digital world. High-level programming languages such as C, Java, Python, and so on are options for expert developers and learners as well. However, text-based programming languages are not simple enough for young children, whose attention relies on fun. Therefore, coding learning tools that can both educate and entertain kids have been developed. This section presents a succinct review of coding tools that can motivate children to learn to code and inspire the development of the KidBright as well.

To simplify coding, most coding learning tools include graphic-based environments in the form of instruction blocks that help accelerate the learning process. In this approach, the learner does not need to write code by typing text instructions but by visually assem-

bling command blocks together. In particular, this approach prevents syntax errors as mismatched blocks cannot be joined together [11]. With all these beneficial features, block-based coding environments are integrated into many coding software tools. As reviewed in [11], prior forms of the block-based coding approach can be seen in BridgeTalk [20] and LogoBlocks [21], designed to help novices learn the concept of coding. Aiming to introduce coding to children, the development of block-based environments continues to grow in numbers. Similar to Scratch and Code.org, introduced in Section 1, Snap! [22], Alice [23], and Pencil Code [24] also devote their block-based coding approaches to coding education. These platforms can inspire learners to learn to code through the creation of easy-to-build applications such as music, art, games, stories, and animations.

Furthermore, Blockly Games [25], developed by Google, provides a set of educational games in the form of block-based coding environments from which children can learn basic coding lessons in preparation for text-based programming languages. The use of block-based coding for STEM education can be seen in many studies, such as teaching robot programming to K–12 students [26], introductory machine learning [27], and mathematical thinking [28]. Block-based coding environments developed for advanced studies are Block4DS [29] and SEAL-Blockly [30], which are used for learning data structures and wireless sensor network programming, respectively. The concept of coding without code using graphical blocks is also applied to other areas besides education, such as MIT App Inventor [31] for mobile application development, Codejig [32] for web-based application development, and Tinkercad [33] for 3D design. Block-based coding is also applied in robot control in studies such as [34,35].

In embedded programming where coding is for developing embedded applications, numerous block-based environments are offered for introducing embedded programming and quick development of embedded systems. As Arduino (<https://www.arduino.cc/> (accessed on 31 August 2022)) is a very popular embedded platform, several block-based coding environments are offered as alternatives for beginner coders in place of the Arduino IDE, which provides only a text-based coding environment. Some examples of them are given in Table 1.

Table 1. Block-based coding environments for the Arduino (accessed on 31 August 2022).

Block-Based Coding Environment	Link	Usage
Snap4Arduino	https://snap4arduino.rocks/	Web app/installer
Visualino	http://www.visualino.net/	Executable file
ArduinoBlocks	http://www.arduinoblocks.com/	Web app
mBlock	https://mblock.makeblock.com/en-us/	Web app/installer
S4A	http://s4a.cat/	Installer
Blockly@rduino	https://technologiescollege.github.io/Blockly-at-rduino/	Web app
Otto Blockly	https://www.ottodiy.com/software	Installer
BlocklyDuino	https://blocklyduino.github.io/BlocklyDuino/blockly/apps/blocklyduino/	Web app
ElectroBlocks	https://electroblocks.org/	Web app
ArduBlock	http://blog.ardublock.com/	Add-on
miniBloq	http://blog.minibloq.org/p/download.html	Installer
ArduBlockly	https://ardublockly.embeddedlog.com/demo/index.html#	Web app

Furthermore, as listed in Table 2, there are other block-based coding environments that support a variety of off-the-shelf embedded devices. It should be noted that the coding environments shown in Tables 1 and 2 can be accessed through various ways, such as via a software installer, a web application, an executable file, or an add-on provided on the respective websites indicated in the tables. More information regarding the use of those coding environments can also be found on their websites.

Table 2. Examples of block-based coding software tools for programming off-the-shelf embedded hardware platforms (accessed on 31 August 2022).

Block-Based Coding Software	Embedded Device	Link	Usage
Raspblocks	Raspberry Pi	http://www.raspblocks.com/	Web app
Wylidrin Studio	Raspberry Pi, Arduino	https://wylidrin.studio/	Web app/installer
EduBlocks	micro:bit, Raspberry Pi	https://app.edublocks.org/	Web app
RoverCode	micro:bit, micro:bit-based robotic toys	https://rovercode.com/	Web app
Microsoft MakeCode (or MakeCode)	micro:bit, Lego Mindstorms	https://www.microsoft.com/en-us/makecode	Web app/installer
Scratch	micro:bit, Lego Wedo/Boost	https://scratch.mit.edu/projects/editor/	Web app/installer
Open Roberta	Lego Mindstorms/NXT/WeDo, micro:bit, Arduino, mBot	https://lab.open-roberta.org/	Web app
MicroBlocks	micro:bit, Raspberry Pi Pico, M5Stack, Collioep, ESP32/ESP8266	https://microblocks.fun/get-started	Web app/installer
UIFlow	M5Stack serries	https://flow.m5stack.com/	Web app

In particular, the development of block-based embedded programming platforms was presented in the following articles. In [36], DSCBlocks was proposed as an open-source platform for learning to program digital signal controllers, dsPIC 33FJ128GP804 and dsPIC 33FJ128MC802. Using Google Blockly, this platform provides instruction blocks for configuring and controlling the controllers' pins, ports, and peripherals. A block-based program can be written by means of drag-and-drop coding and translated into C code in real-time. In [37], the development of a web-based, block-based coding platform for embedded systems was presented. This platform deploys MakeCode, which mainly provides a block-based coding environment and a code compiler, and CODAL (CODAL (<https://lancaster-university.github.io/codal/> (accessed on 31 August 2022)) is now used as a support C++ coding environment for MakeCode), which provides libraries for device drivers and scheduler mechanisms. The platform has been tested with micro:bit, Adafruit Circuit Playground Express, and Arduino boards. In [38], a web-based coding tool named BIPES was introduced. BIPES's block-based coding environment is based on Google Blockly's JavaScript library. In learning embedded programming with BIPES, the learner constructs a block-based program, which is then translated to Python code and sent to the target embedded device through USB or network connectivity. In this way, the target device must have a Python interpreter. BIPES does not compile code and program the device but lets the device interpret the Python code so as to run the program.

Moreover, it is worth mentioning the micro:bit (<https://microbit.org/> (accessed on 30 October 2022)) and its coding tools, as it is the most similar coding learning platform to the KidBright in many ways. The micro:bit or the BBC micro:bit is a pocket-sized, programmable computing board, which can inspire children to learn embedded programming. The micro:bit has been developed by the Micro:bit Educational Foundation, a non-profit organization in the UK. Technically, the micro:bit's latest version is equipped with a number of peripherals, such as 25 red LEDs, two programmable push buttons, an accelerometer, a magnetometer, a temperature sensor, a microphone, a speaker, and so on. The micro:bit can also interface with other devices through its I/Os, I2C, SPI, Bluetooth, and dedicated radio. Coding and programming the micro:bit can simply be accomplished through various options such as MakeCode, MakeCode Apps, Scratch, Swift Playgrounds, Python, and so on in support of diverse technology, e.g., computer platforms (PCs, tablets, and mobile phones); operating systems (Microsoft Windows, iOS, macOS, and Android); and development environments (block-based and text-based) [39]. With the on-board components provided, learners can build plenty of easy-to-learn projects. For instance, using the on-board accelerometer, the micro:bit can be programmed to be an electronic die which can display random numbers when it is shaken.

All of the reviewed work shows that block-based coding is a promising approach to introducing coding to learners who are new to computer programming. The embedded programming software tools mentioned provide their block-based coding environments as an entry point to lower the barrier to learning embedded programming. Their coding environments rely on one of the following software technologies:

- Google Blockly [19] is an open-source library for building visual block-based environments. Many coding tools, including BIPES, DSCBlocks, Wyliodrin Studio, Visualino, Blockly@rduino, and BlocklyDuino, deploy Google Blockly;
- Scratch Blocks [40] is actually based on Google Blockly. It provides a framework for developing graphical coding blocks. Scratch, mBlock, and S4A are example applications of Scratch Blocks;
- Microsoft Programming Experience Toolkit (PXT) [41] is a software framework for Microsoft MakeCode;
- Snap [42] is a graphical block-based programming language derived from Scratch. Snap4Arduino is a block-based coding tool that relies on Snap.

It should be noted that embedded programming basically includes coding statements (loops, decisions, math operations, and so on) and hardware access (bit-level configurations, device setup, data acquisition, output control, and so on), both of which can be made into the form of graphical instruction blocks, as can be seen from those software tools. Besides the deployment of the block-based coding approach, what those software tools have in common is that they convert block-based code for a target device into a programming file format, such as binary firmware, machine code, or programming language syntax such as C or Python. Generally, instruction blocks for fundamental coding statements can run at the CPU's speed, whereas most instruction blocks for hardware access cannot, especially those that access slow peripheral devices. In this way, direct hardware access by instruction blocks creates issues in embedded programming. For example, an instruction block reading a slow-operated sensor would result in substantial latency that affects all the blocks connected below and the overall user program flow as well.

What makes the KidBright different is the implementation of the KidBright Engine, a dedicated software framework, which provides a device manager to access hardware in support of instruction blocks, such that instruction blocks do not directly access hardware, instead using memory updated by the device manager. Thus, instruction blocks that access time-consuming hardware will never freeze the user's program as they perform in a non-blocking, asynchronous fashion. Furthermore, based on this principle, the KidBright Engine also provides an approach for the development of external instruction blocks. Although there are block-based coding software platforms that allow developers to build external instruction blocks for them, such as MakeCode, Scratch, Wyliodrin Studio, MicroBlocks, and BIPES, their approaches still allow direct hardware access by instruction blocks built under the rules of the block technologies they deploy. For example, MakeCode's architecture includes CODAL in combination with fibers for handling hardware in a concurrent, even-driven fashion. However, MakeCode still allows code implementations of instruction blocks to directly access hardware with the CODAL runtime, where latency can be an issue (demonstrations are provided in Section 5).

Furthermore, motivated by the fact that none of those coding software platforms and their learning resources support Thai, the proposed software framework is designed to provide both Thai and English language support. Makerasia, one of the KidBright's partners, provides a software tool called KB-IDE (<https://kb32ft.makerasia.com/getstart/kbide>) (accessed on 31 August 2022) for coding education with the official and modified versions of the KidBright board and the Arduino. Although KB-IDE offers a block-based coding environment for programming the KidBright board, it does not support Thai and does not have the KidBright Engine. Thus, our software framework enables developers to create command blocks that support Thai and English and helps handle low-level hardware issues to support inexperienced learners who understand Thai and/or English. Altogether,

these benefits help support the KidBright community in learning to program embedded systems worldwide.

3. Coding with KidBright

It would be hard to attract young children to become interested in coding with traditional textual coding environments. Coding with the KidBright is a promising option that can amuse children through experiencing coding as well as tangible activities created by themselves. Basically, coding with the KidBright involves two components, namely, the KidBright board and the KidBright IDE. From imagination to realization, one example is that a KidBright board is programmed to water plants by detecting a specified humidity level as well as a predefined light intensity. To control a KidBright board to function as expected, the learner can simply create a program on the KidBright IDE by just visually dragging and dropping a set of relevant command blocks. After that, the KidBright IDE compiles the command blocks and uploads the resulting file to the KidBright board through a USB cable. Without a KidBright board, learning to code with the KidBright can be performed through the use of the KidBright simulator. More details on the KidBright board, KidBright IDE, and KidBright simulator are described in the following sections.

3.1. KidBright Board

The KidBright board is an open-source, small-sized computer device powered by a 32-bit microcontroller and a set of on-board peripheral components. Coding and programming the KidBright board can simply proceed through the use of the KidBright IDE, allowing students to create unlimited ideas using not only the on-board components provided but also external ones. The KidBright board supports various communication interfaces such as I2C, SPI, UART, Bluetooth, WiFi, and the internet. Students are able to have their KidBright boards function as Internet of Things (IoT) devices, so they would enjoy sending and collecting data anywhere.

The KidBright board has been continuously developed since 2015. Thus far, a number of versions have already been released. The evolution of the KidBright board is divided into generations 1 and 2, based on the deployed microcontrollers ESP8266 and ESP32, respectively. All the versions in the second generation have the same size, which is 50 mm by 90 mm. As KidBright is open-source, diverse versions of KidBright boards with minor differences have been launched by our partners. Nevertheless, in this paper, we focus on version 1.5, which has recently been officially released. Figure 1 shows both the top and bottom sides of the KidBright board, the components of which are as follows: (1) temperature sensor, (2) 16 × 8 LED display, (3) light sensor, (4) wireless connection indicators, (5) USB type C port, (6) buzzer, (7) I²C port, (8) USB type A port, (9) servo power terminal, (10) two servo connectors, (11) reset switch, (12) push button switch 2, (13) push button switch 1, (14) input/output connectors, (15) digital output port, (16) input port, (17) 3.3 V power source and ground (GND), (18) auxiliary input/output connector, (19) ESP32-WROOM microcontroller, (20) real-time clock battery holder, and (21) real-time clock chip.

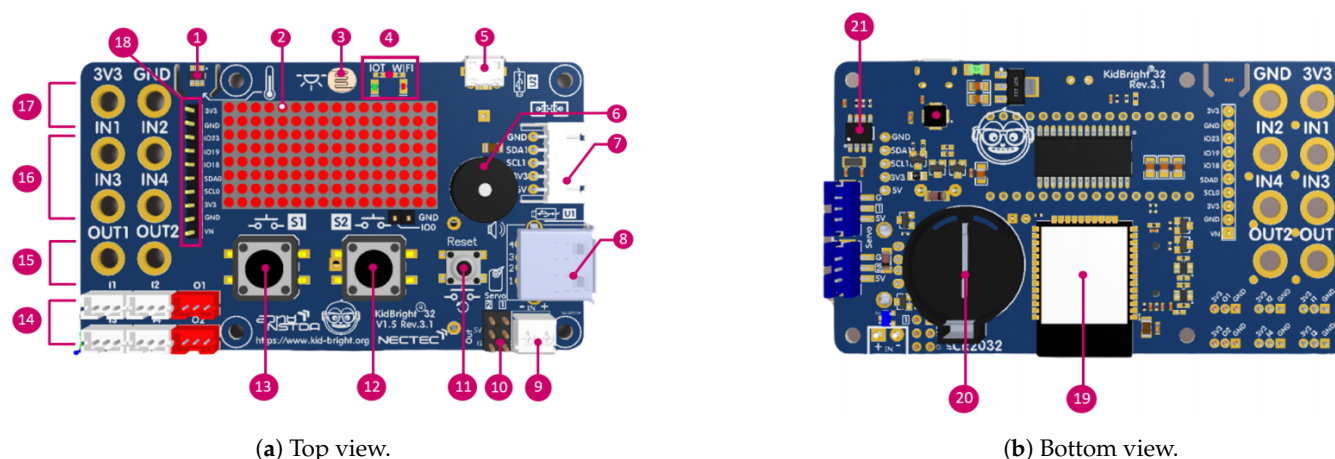


Figure 1. KidBright board V1.5.

Regardless of extra devices, the KidBright board alone is sufficient for beginners, covering basic coding lessons available on the KidBright website. With the 16×8 LED dot matrix, students can display a number, a text message, or an image, which can be made still or moved. A student may use the temperature and/or light sensors for environmental detection and the buzzer to play notes or even music. A KidBright board can control a USB-powered fan or light bulb through its USB port. Students can also make use of one or two programmable switches for enabling dedicated application tasks. The KidBright board's full schematic diagram can be found on the KidBright Download page [43], and its key features are summarized below:

- A powerful 32-bit microcontroller, ESP32-WROOM-32, runs at 80 MHz.
- A 16×8 dot-matrix display for displaying a static or dynamic message, number, or image.
- Two general-purpose push-buttons as application inputs.
- A buzzer that can generate sounds for alarms, melodies, music, and indications.
- A real-time clock and its battery backup that keeps track of the current time and date.
- A temperature sensor and a light sensor for sensing the ambient temperature and light intensity, respectively.
- Support for Bluetooth, WiFi, and IoT.
- Support for external devices through I/Os, I2C, SPI, and UART interfaces.

Beyond the need to learn how to code, advanced learners might be curious about developing challenging applications to which extra sensors and/or actuators are needed to connect. With our dedicated design, the KidBright board has an expansion paradigm called the KidBright Chain. The KidBright Chain includes the so-called I/O, I2C, and SPI interfaces. More details on the KidBright Chain are presented in Section 4.

3.2. KidBright IDE

The concept of coding with the KidBright is to let students learn and play at the same time. A student can learn to code through the use of the KidBright IDE in conjunction with a KidBright board. The KidBright IDE is an open-source, all-in-one software tool designed for coding and programming the KidBright board. The latest and previous versions, which support both Windows and Mac operating systems, are available on the KidBright Download page [43]. As previously introduced, the coding language of the KidBright is block-based, wherein users can construct a program by means of drag-and-drop instruction blocks. Figure 2 shows the aspects of the KidBright IDE, which mainly includes the dedicated code space along with its tool buttons and command tabs.

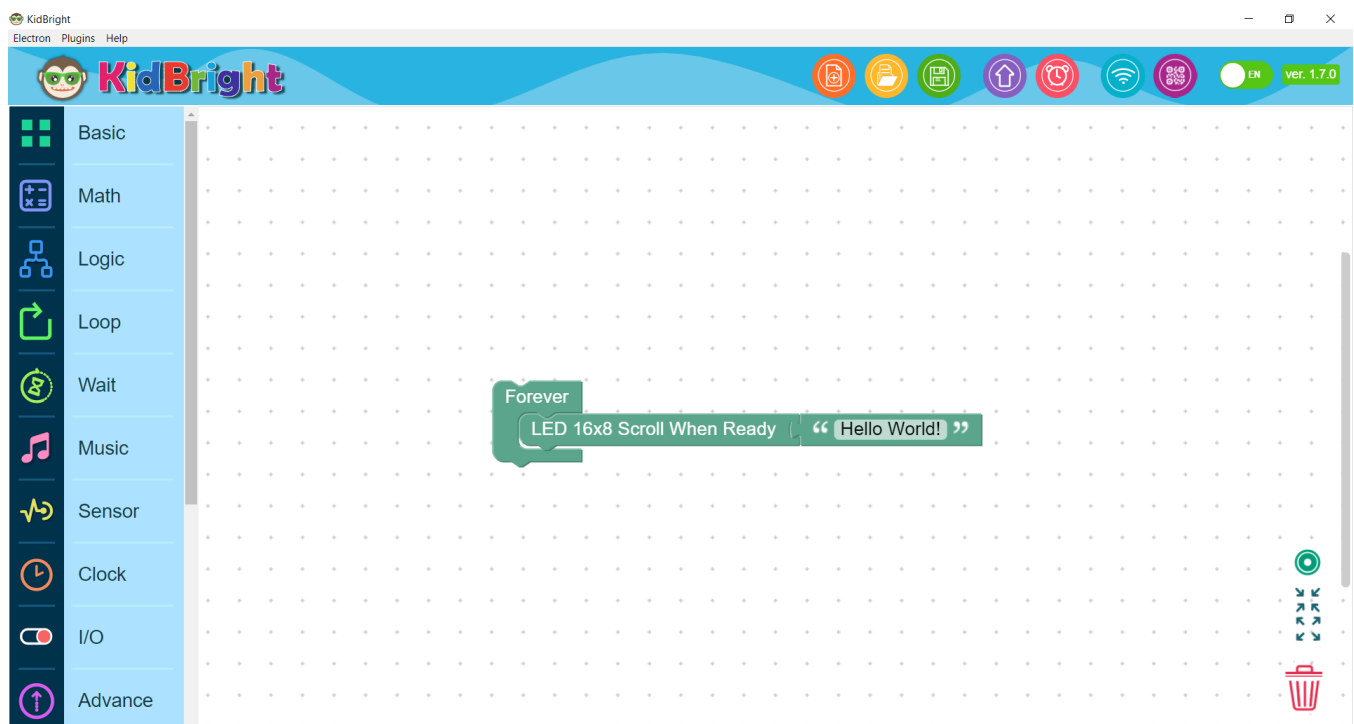


Figure 2. KidBright IDE V1.7.

With the tool buttons, learners can create, open, and save their code spaces and program their KidBright boards as well. The KidBright command blocks are grouped into command tabs based on their similarities in terms of functions, such as Basic, Math, Logic, Loop, Wait, Music, and so on. Therefore, the learner can simply find preferred instruction blocks from the command tabs and then drag and drop them to form a jigsaw-like program. Furthermore, for advanced learners, the KidBright instruction blocks provided might not satisfy their requirements, e.g., reading an external sensor, controlling an actuator, interfacing an OLED display, and so on. Nevertheless, the KidBright IDE is designed to support imported instruction blocks as well. Learners are allowed to install and uninstall additional instruction blocks anytime through the Plugins tab of the KidBright IDE.

It should be noted that all official KidBright boards are not pre-programmed by factories. In other words, each KidBright board is built without firmware in the first place, as the KidBright IDE will be responsible for the firmware when used by the learner. This direction supports our open-source concept that allows the KidBright board to not rely on any specific firmware. Once the KidBright IDE compiles a block-based program, the respective firmware is generated and uploaded to a KidBright board's flash memory. Specifically, the KidBright firmware generated includes not only the learner's block-based program but also device and system management. In particular, the whole firmware is built under our software framework called the KidBright Engine, which is the most significant component of the KidBright IDE. The KidBright Engine, as the backend of the KidBright IDE, plays an important role in hiding complex hardware settings and operations from learners. Thus, it enables learners to enjoy coding without concerns about unnecessary hardware and software issues. The KidBright Engine is explained more in Section 4.

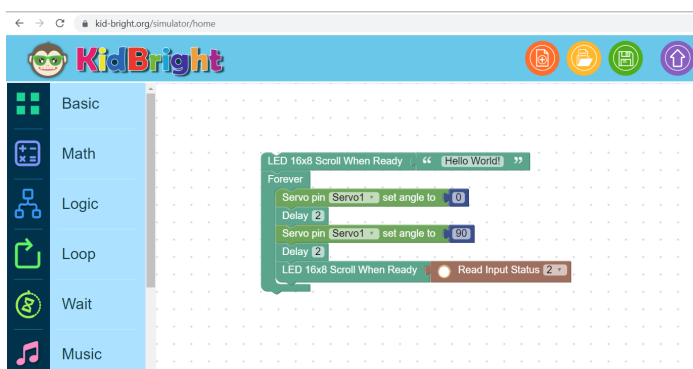
3.3. KidBright Simulator

Our goal is to expand coding education with the KidBright as widely as possible. Therefore, the KidBright IDE is allowed to be used free of charge and the KidBright board is designed to be low-cost and open-source. However, it appears that not every student can afford a KidBright board, and the worldwide availability of KidBright boards requires more time. To achieve our goal, we provide the KidBright simulator, which

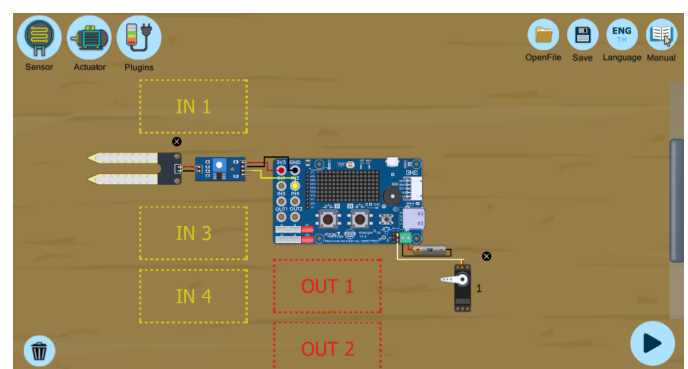
allows any learner to code, program, and run the virtual KidBright board without the need for a real one. The KidBright simulator is a web-based software tool available on <https://www.kid-bright.org/simulator/home> (accessed on 25 August 2022), developed and promoted in parallel with the KidBright board and the KidBright IDE. Everyone is welcome to experience it without registration and fees.

As can be seen in Figure 3a, the KidBright simulator's coding environment looks similar to that of the KidBright IDE, except that a few tool buttons specified for the real KidBright board are made invisible. With the same command tabs, the KidBright simulator allows students to learn basic KidBright coding lessons with the virtual KidBright board, as illustrated in Figure 3b. Specifically, the virtual board can function similar to a real KidBright board, except that operation results are graphically displayed on the simulator only. It should be noted that the use of the simulator is also useful to those who have KidBright boards, as the simulator can be used for testing code before real implementation as well. Currently, the KidBright simulator supports a number of virtual devices, listed as follows:

- Sensor category: moisture, force, PIR, IR, line tracker, gas, sound, ball switch, shock, tilt, object avoidance, and touch sensors.
- Actuator category: USB flash light, USB fan, water pump, DC motor, LED, and laser diode.
- Plugin category: servo motor, Line Notify, and Bluetooth.



(a) Coding environment.



(b) Virtual KidBright board.

Figure 3. KidBright Simulator.

Through the simulation, these virtual devices can be controlled by the virtual KidBright board with respect to a KidBright user program. It should also be noted that the simulator is designed to support both virtual built-in and plug-in devices. From the list above, we note that the servo motor, Line Notify, and Bluetooth are plugged in, whereas the others are built in. However, the current version of the simulator is not ready to support virtual plugins developed by makers.

4. Design of the KidBright

The KidBright is designed to be as simple as possible in terms of usage, such that students can spend their time enjoying coding rather than learning the tools. However, simplification for learners comes at the expense of sophisticated designs on the development side, as there are a lot of hidden elements handling the frontend. To accommodate an ecosystem, the design concept of the KidBright is based not only on simplicity but also on sustainability, which means that, besides the KidBright design team, the KidBright would be continuously maintained by its community. In this direction, therefore, the KidBright is open for makers to build new command blocks in the form of plugins for the KidBright IDE as well as to build alternative versions of the KidBright board. We believe that this concept is the key to success in broadening the use of the KidBright for coding education. Thus,

this section is dedicated to the design of the KidBright in the context of its firmware and its hardware extension paradigm, which significantly support the concept of sustainability.

4.1. Firmware

Block-based coding has been widely adopted for introducing coding to children, as it can help to simplify coding by its nature. However, success in the development of hardware-based coding learning tools does not rely solely on the use of block-based coding. One more significant key is the management of all resources and peripherals in favor of learners. It should be pointed out that beginners could focus only on their coding process, without any concerns regarding low-level software and hardware issues, e.g., handling inputs/outputs, accessing devices, scheduling tasks, and so on. Handling those kinds of things in embedded systems is far more complicated for inexperienced learners. To eliminate those difficulties, peripheral management should be involved. One approach is to install an operating system (OS) on the microcontroller. However, high-end OSs require large installation storage as well as CPU resources beyond the limitations of most microcontrollers, including the ESP32 on the KidBright board. Another possible solution is to implement a device manager that handles peripheral devices and enhances data throughput by allowing participating components to perform direct data transfer with memory.

From this point of view, in order for the KidBright to manage all behind-the-scenes activities, the KidBright Engine introduced in Section 3 is integrated into the KidBright IDE. We recall that the KidBright Engine is a dedicated software framework designed to handle the learner's block-based program in combination with its device and system management in order to generate the respective firmware at compiling time. The KidBright Engine works in such a way that learners do not need to know what is behind the KidBright command blocks in order to get their KidBright board to function. To be more clear, the KidBright Engine is structured as shown in Figure 4.

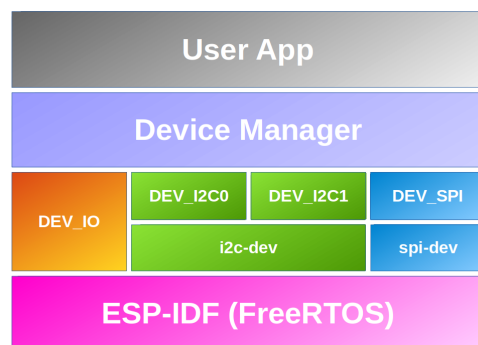


Figure 4. The KidBright Engine's structure.

As can be seen from the figure, under the user application (a KidBright block-based program), the KidBright Engine has its device manager running on the ESP-IDF FreeRTOS [44], a well-known lightweight real-time operating system for microcontrollers, which handles concurrent operations and shared resources. Specifically, the device manager is responsible for device registration, device state machines, resource locking, device status, and maintenance in support of expandable device connectivity and enhanced data throughput with predictable latency.

From Section 3, recall that the KidBright board has the KidBright Chain, which provides I/O, I2C, and SPI interfaces. To be more specific, the KidBright Chain is referred to as the group of the I/O chain, I2C chain, and SPI chain, corresponding to the I/O, I2C, and SPI interfaces, respectively, as shown in Figure 5. In the context of the KidBright Engine, these peripheral interfaces are handled by the device manager and the respective device drivers, namely, DEV_IO (for I/O interfacing), DEV_I2C0 and DEV_I2C1 (for two ports of I2C interfacing), and DEV_SPI (for SPI interfacing). For the I2C and SPI drivers, each can support connections up to 64 devices. In respect of the four device drivers, the device

manager creates four tasks, each of which is described by a finite-state machine with three main states, running on the ESP-IDF Free RTOS, as illustrated in Figure 6.

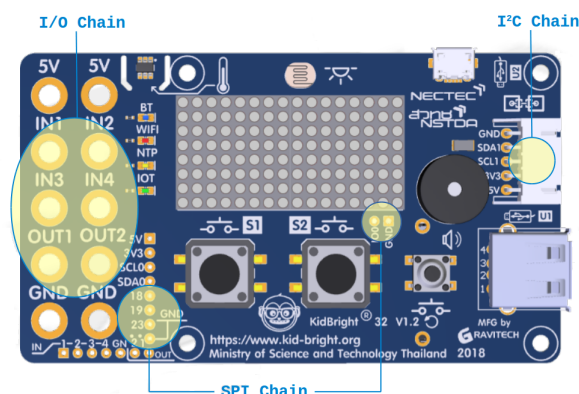


Figure 5. The KidBright Chain on the KidBright board.

vDevListTask State Machine

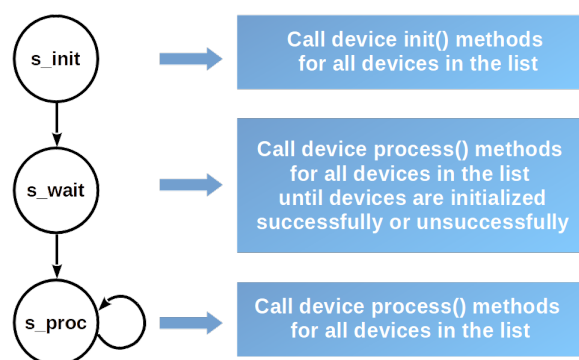


Figure 6. A state machine describing a device driver.

Firstly, in the initialization state (s_{init}), initialization methods of all devices in the registration list are called in order for the devices to initialize their software and hardware settings. However, the amount of time required for initializing different devices is basically not equal. Therefore, the state machine jumps to the wait state (s_{state}) and waits until all the devices are marked as enabled or disabled, corresponding to successful or unsuccessful initialization, respectively. Then, the state machine is in the process state (s_{proc}), where all enabled devices are periodically scheduled to acquire and transfer data to the device manager.

Specifically, the device manager saves all acquired data received from devices into memory, which stores both input and output data, e.g., data read from sensors, data sent to the LED matrix display, and so on. In order to mitigate the latency introduced by sensors in data transfer, this approach allows instruction blocks to perform indirect access to memory instead of direct access to devices. In addition to this, we note that the priority of the device manager's tasks is assigned to be higher than that of all the instruction blocks' tasks. With this principle, the device manager can help improve the throughput of data transfer by handling low-level hardware activities in support of the instruction blocks.

To date, we note that the KidBright Engine also plays one more important role in allowing developers to create new command blocks in the form of KidBright plugins through the device manager. The approach for the plugin development is presented in Section 4.3.

4.2. Device Expansion

To enhance the KidBright board's capabilities, learners are recommended to interface their KidBright board with external devices in addition to the on-board peripherals. As introduced earlier, the KidBright board can connect to the outside world via its I/O, I2C, and SPI chains, altogether referred to as the KidBright Chain. In particular, both the I2C and SPI chains support expandable device connectivity up to 64 device channels. According to this specification, the I2C and SPI channel expanders are designed to increase the capacities of the I2C and SPI chains. In addition to this, we note that the I2C and SPI channel expanders also allow sensors with the same device addresses to coexist without conflicts and help translate different voltages of participating devices to be compatible with those of the chains. All these features of the expanders are summarized in Figure 7. KidBright instruction blocks that access I2C and SPI peripheral devices have a drop-down property that provides device channel options. In the context of software, what is responsible for the channel switching mechanism is the KidBright Engine's device manager. It should be noted that this expansion paradigm can empower the KidBright board to be scalable in implementing large applications, for example, an irrigation system having numerous sensors mounted around a greenhouse.

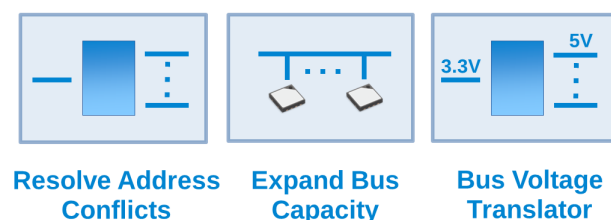


Figure 7. Features of the I2C and SPI channel expanders.

Technically, a KidBright board can connect up to eight I2C and four SPI channel expanders. An I2C channel expander, which deploys a TCA9548A chip, and a SPI channel expander, which deploys a MCP23S17 chip, can support up to 8 I2C and 16 SPI channels, respectively. With this design, therefore, a KidBright board can support up to 64 I2C and 64 SPI channels. Scenarios of a KidBright board's connectivity to multiple I2C and SPI devices through the device channel expanders are illustrated in Figure 8. The circuit diagrams of the I2C and SPI channel expanders can be found at <https://gitlab.com/kidbright/kbide/-/tree/master/docs/schematics> (accessed on 25 August 2022).

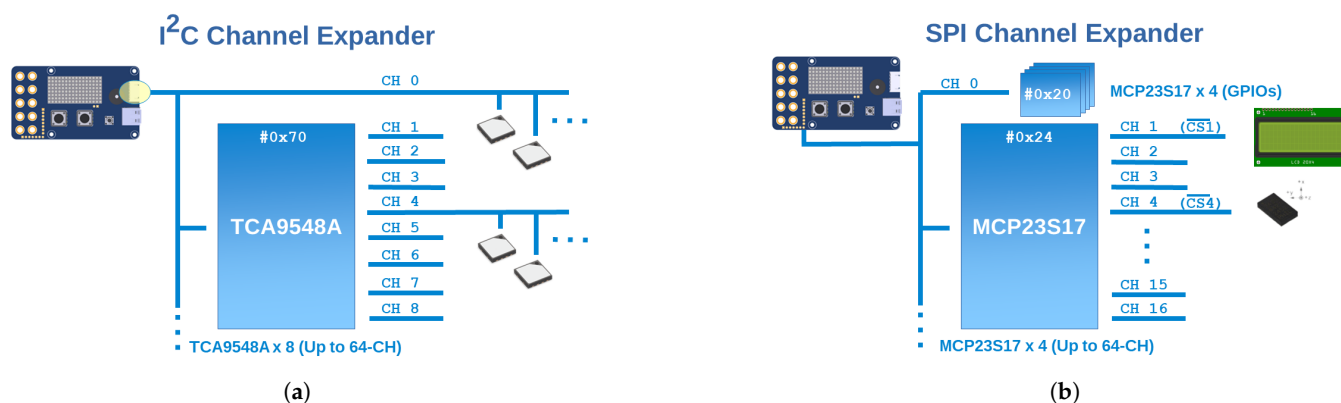


Figure 8. KidBright device expansion. (a) Multiple I2C channel expanders. (b) Multiple SPI channel expanders.

We note that once a new device unknown to the KidBright IDE is connected to a KidBright board through either the I2C chain or the SPI chain, the user has to install a plugin that supports the device on the KidBright IDE; thus, the user can use command blocks provided by that plugin to access the new device. We also note that this requirement

is also applied to the case of the IO chain. A number of plugins supporting various devices and functions are provided on the KidBright website where developers are also allowed to share their plugins with the community.

4.3. Plugin Development

Recall that one important contribution of our work is to enable developers to create new instruction blocks supporting peripheral devices not provided by our team so as to let learners and developers support each other. One of the KidBright Engine's functions is to provide a plugin development approach for new instruction blocks. In this section, we describe, by example, how to develop a plugin. The software installation requirements are as below:

- Node.js (versions 8.XX are recommended)
- Python
- Pyserial

The KidBright IDE's source code can be installed by following these steps:

```
> git clone https://gitlab.com/kidbright/kbide—recursive (accessed on 25 August 2022)
> cd kbide
> npm run build
```

During development, the developer must run *npm* by using the command line *npm start* and then open a web browser, the url of which is specified to <http://localhost:8000> in order to run the KidBright IDE for testing the developing plugin. To explain how to develop a plugin, a simple plugin that provides instruction blocks to start and stop blinking an on-board LED is illustrated as an example. This plugin is named *Blink* and is located under its plugin group named *Examples*. By design, the KidBright IDE's source tree provides a separate folder named *Plugins* for storing all the plugins. That is, under *Plugins*, there are groups of plugins, each of which stores one or more plugins in the form of file folders. With this structure, we note that a plugin group and its plugin(s) can be easily uninstalled by physically removing its folder (to provide comfort to the user, a plugin can be installed and uninstalled through the Plugin tab of the KidBright IDE). Figure 9a,b shows groups of plugins before and after adding the group plugin *Examples* and Figure 9c shows the instruction blocks of the *Blink* plugin, which includes the *start and stop blinking LED* blocks. Following the KidBright Engine framework, *Blink* plugin's directory tree is organized as shown in Figure 9d, the folders and files of which are explained as follows:

- *examples.json* is located in the *Examples* folder and is used to provide the group plugin's name in both Thai and English for the KidBright IDE (the current version of the KidBright IDE supports English and Thai languages). It should be noted that the file's name must be the same as the plugin group's plugin name.
- The *Blink* folder as the plugin folder is to store the *Blink* plugin's files, which are *blocks.js*, *generators.js*, and the source code: *blink.c* and *blink.h*. In addition, the *Blink* folder also includes a folder named *docs* for providing documents regarding the plugin, useful for users and developers. However, the *docs* folder is optional.
- The *msg* folder stores two files, *en.js* and *th.js*, which provide text messages, namely titles, tooltips, and supporting URLs for the plugin's instruction blocks in English (*en.js*) and Thai (*th.js*).
- *blocks.js* is used to define the aspects and behaviors of the *Blink* plugin's instruction blocks, deploying Google Blockly's library.
- *generators.js* is used to map the instruction blocks to the corresponding C++ methods implemented in the *Blink* plugin's source code so as to control the KidBright board to operate according to the functions of the instruction blocks.

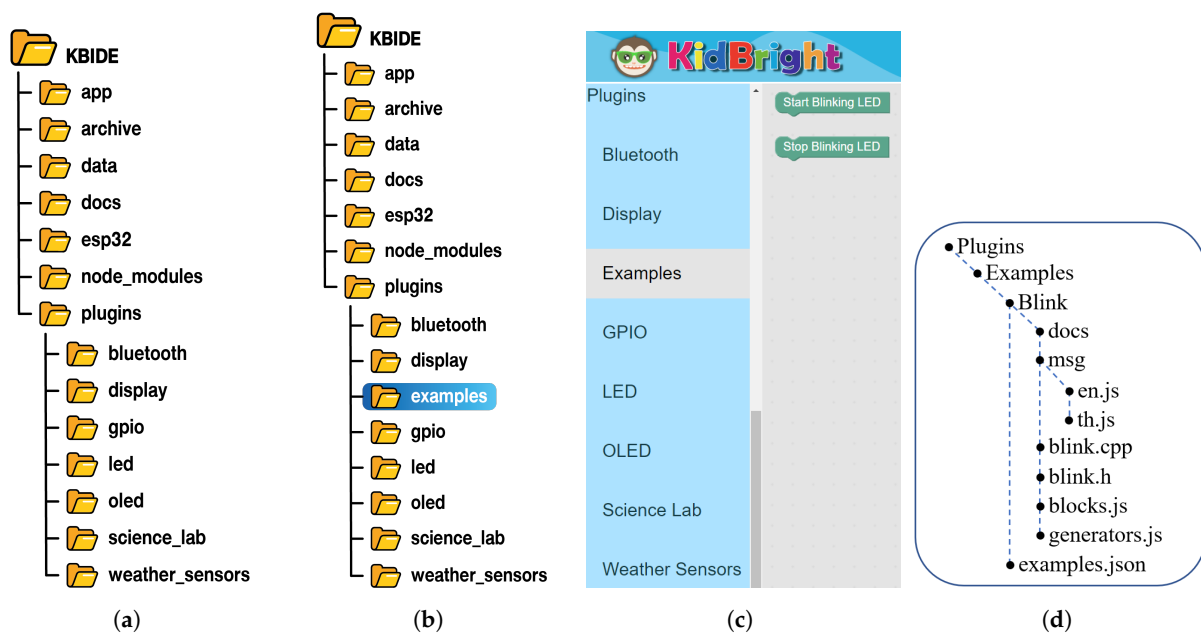


Figure 9. An example of KidBright plugin development. (a) Before adding examples. (b) After adding examples. (c) The new instruction blocks. (d) Directory tree of the *Blink* plugin.

Figure 10 shows the scripts of *examples.json*, *en.js*, *th.js*, *blocks.js*, and *generators.js*.



Figure 10. Scripts for the *Blink* plugin (support for English and Thai is provided through the messages in *examples.json* (English and Thai), *en.js* (English), and *th.js* (Thai)).

- *blink.cpp* and *blink.h* make up the C++ implementation of the plugin, of which the structure complies with the KidBright Engine framework. Recall that one significant key design of the KidBright Engine is the use of the device manager to handle all peripherals' activities behind instruction blocks. That is, instruction blocks are not

allowed to access peripheral devices directly, accessing the devices through the device manager instead. In this paradigm, the device manager handles device access by means of scheduling plugins' state machines in a non-blocking, time-slot fashion. As latency is a significant issue in embedded programming, with this design, one obvious benefit is that devices that take time to get initialized, read, and/or written will never freeze the user program's flow due to their excessive latency. Demonstrations of the device manager in resolving latency are presented in Section 5.

As introduced earlier, the device manager provides four device drivers, namely DEV_IO, DEV_I2C0, DEV_I2C1, and DEV_SPI for the I/O, I2C channel 0, I2C channel 1, and SPI interfaces, respectively. Because the *Blink* plugin uses an I/O to control the LED, DEV_IO is deployed which means that the *Blink* plugin's device class will be registered in the I/O device list and then processed in the device list task of DEV_IO (in the context of software programming, a device class is defined as a representation of a device). Specifically, a plugin's device class mainly includes three methods: *init()*, which serves as an entry point of the plugin where concerning hardware, variables, and so on are initialized; *process()*, which provides the plugin's state machine; and *control methods* that serve the plugin's instruction blocks in accessing dedicated hardware. The *Blink* plugin's device class is defined in *blink.h*, shown in Figure 11, whereas the respective code is in *blink.cpp*, which is shown in Figure 12. Following the framework, the *Blink* plugin's *init()* is to configure the I/O that controls the LED as well as initialize the control flags and the start state of the state machine. The *process()* is to be called by the device manager through the DEV_IO device list task such that its state machine is scheduled to run concurrently. The *start()* and *stop()* control methods are to start and stop the blinking of the LED in response to the *start* and *stop blinking LED* blocks, respectively.

```

blink.h

#ifndef __BLINK_H__
#define __BLINK_H__

#include "driver.h"
#include "device.h"
#include "driver/gpio.h"

class BLINK : public Device {
private:
    enum {
        s_detect, s_blink
    } state;
    int blink_status;
    bool blink_flag;

public:
    TickType_t tickcnt;
    // constructor
    BLINK(void);
    // override
    void init(void);
    void process(Driver *drv);
    int prop_count(void);
    bool prop_name(int index, char *name);
    bool prop_unit(int index, char *unit);
    bool prop_attr(int index, char *attr);
    bool prop_read(int index, char *value);
    bool prop_write(int index, char *value);
    // method
    void start(void);
    void stop(void);
};

#endif

```

Figure 11. *blink.h*.

```

blink.cpp

#include "esp_system.h"
#include "kidbright32.h"
#include "blink.h"

BLINK::BLINK(void) {
    //
}

void BLINK::init(void) {
    gpio_config_t io_conf;

    // outputs init
    io_conf.intr_type = GPIO_INTR_DISABLE;           // disable interrupt
    io_conf.mode = GPIO_MODE_OUTPUT;                // set as output mode
    io_conf.pin_bit_mask = (1ULL << BT_LED_GPIO);   // pin bit mask
    io_conf.pull_down_en = GPIO_PULLDOWN_DISABLE;    // disable pull-down mode
    io_conf.pull_up_en = GPIO_PULLUP_DISABLE;        // disable pull-up mode
    blink_status = 1;
    gpio_set_level(BT_LED_GPIO, blink_status);       // active low
    gpio_config(&io_conf);

    blink_flag = false;
    state = s_detect;
}

void BLINK::process(Driver *drv) {
    switch (state) {
        case s_detect:
            // clear error flag
            error = false;
            // set initialized flag
            initialized = true;

            // go to blink state
            state = s_blink;
            break;

        case s_blink:
            if (blink_flag) {
                if (is_tickcnt_elapsed(tickcnt, 500)) {
                    blink_status ^= 0x01;           // toggle status
                    gpio_set_level(BT_LED_GPIO, blink_status);
                    // get current tickcnt
                    tickcnt = get_tickcnt();
                }
            }
            break;
    }
}

void BLINK::start(void) {
    blink_status = 0;
    gpio_set_level(BT_LED_GPIO, blink_status);       // active low
    // get current tickcnt
    tickcnt = get_tickcnt();
    blink_flag = true;
}

void BLINK::stop(void) {
    blink_flag = false;
    blink_status = 1;
    gpio_set_level(BT_LED_GPIO, blink_status); // active low
}

```

Figure 12. C++ code implementation of *Blink* plugin.

Although driving the LED does not introduce latency, the blinking period does. Thus, the use of an instruction block that directly blinks the LED with delays can potentially block other instruction blocks involved in a user program. With the proposed framework, the instruction blocks do not directly control the blinking of the LED but indirectly make it through the provided C++ methods while the blinking is handled by the state machine running in the background.

Thus far, although DEV_IO is raised as an example, this plugin development framework is also applied to the other device drivers to which any other peripheral devices belong. We note that all built-in KidBright instruction blocks for accessing the on-board hardware peripherals have also been developed with the framework, and their source code, as well as that of other plugins, is available for learning and use on <https://gitlab.com/kidbright/kbide> (accessed on 25 August 2022).

5. Demonstrations

In introducing coding, the KidBright does not only make use of the block-based coding approach but also the proposed software framework, which plays an important role in helping learners to learn embedded programming with little effort as well as in building ecosystems where learners and developers can support each other. We note that there is no doubt that block-based coding can inspire learners to learn to code, as block-based coding has become a promising approach for introductory coding. Thus, rather than evaluating the KidBright coding environment in terms of user interface satisfaction, we investigate the KidBright Engine in the context of support for embedded programming and sustainability.

5.1. Latency

Latency is one big issue in embedded systems where fast and slow devices might coexist, and the embedded software must cooperate with all of them to work smoothly. To illustrate the effectiveness of the KidBright Engine in resolving this issue, two empirical examinations were performed: one for expandable device connectivity and the other one for responsiveness. We conducted these two tests using the KidBright board and the

KidBright IDE in comparison to the micro:bit and MakeCode. We note that the most similar embedded programming platform to the KidBright is the use of micro:bit and MakeCode. With all due respect, we believe in different designs for different purposes.

In the first experiment, we investigate hardware latency through I2C bus occupancy. It is noted that I2C's protocol and physical bus characteristics are latency themselves, making it useful for our observations. We used a KidBright board and a micro:bit, each of which has a SHT21 temperature and humidity sensor connected to its I2C port in order for us to investigate the I2C bus occupancy in conjunction with the number of SHT21 temperature read blocks. The SHT21 temperature read block of the KidBright is the *SHT21 Temperature* block, while that of the micro:bit is the *read temperature* block. The KidBright and micro:bit's block-based programs with different numbers of temperature read blocks are shown in Figure 13. Some captured waveforms of the micro:bit and KidBright's I2C frames with different numbers of SHT21 temperature read blocks are illustrated in Figure 14.

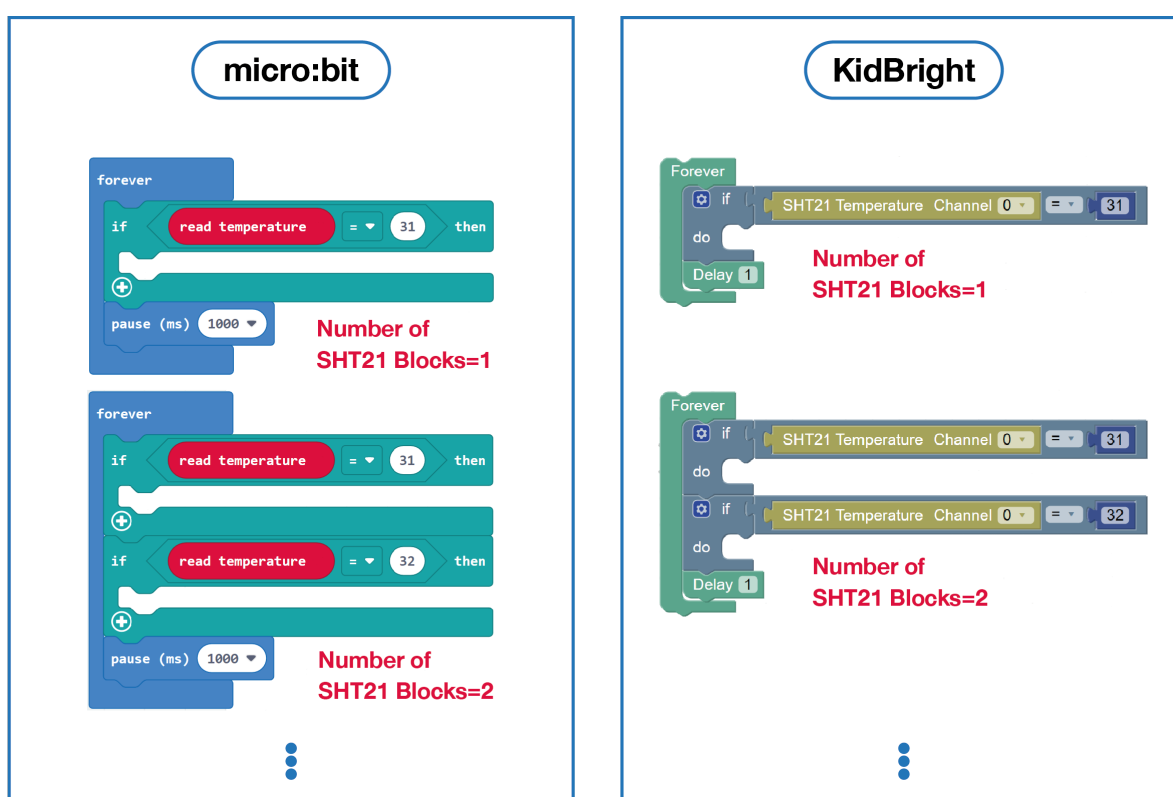
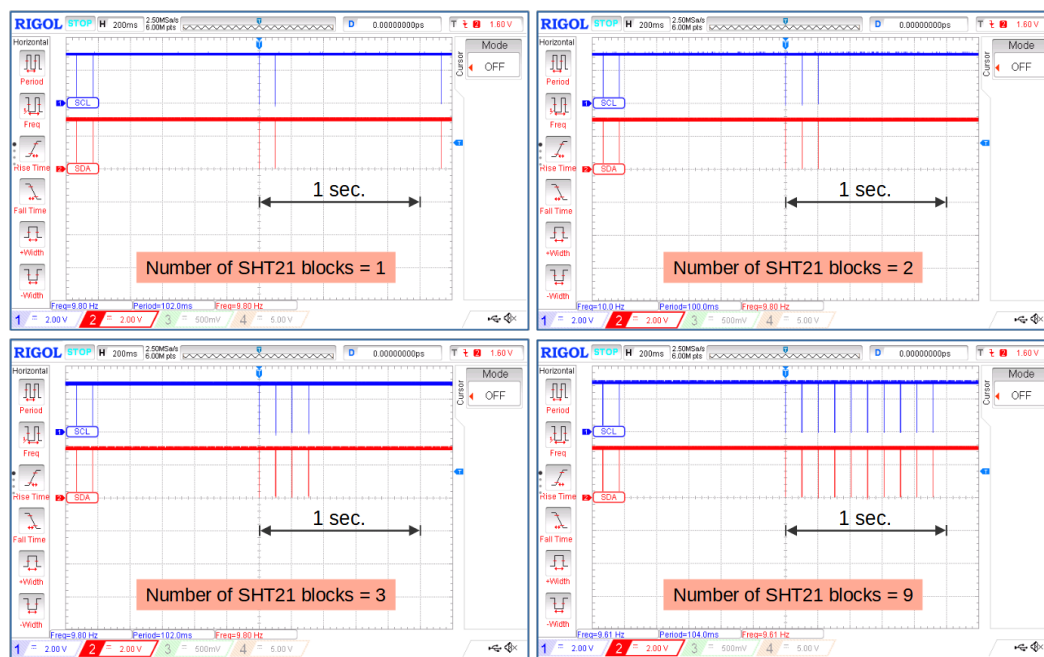
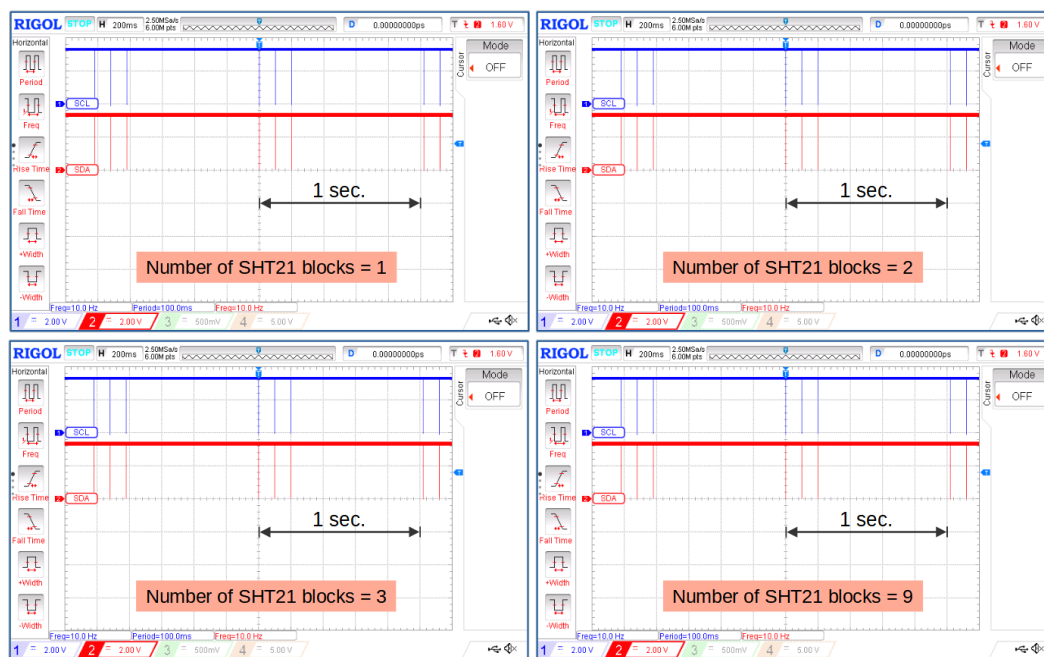


Figure 13. Block-based programs for the I2C bus occupancy investigation.



(a) micro:bit



(b) KidBright

Figure 14. Captured waveforms obtained from the I2C bus occupancy tests.

Figure 15 plots the results obtained from the experiment, where the horizontal axis represents the number of temperature read blocks ranging from one to nine, while the vertical axis represents the corresponding number of I2C frames within a given window period of 1 second. As can be observed in the figure, for the micro:bit, the number of I2C frames linearly increases as more micro:bit temperature read blocks are used, whereas, for the KidBright, the number of I2C frames stays constant at three regardless of how many KidBright temperature read blocks it includes, even though its single SHT21 temperature read block requires one I2C frame more than that of the micro:bit. The results imply that the micro:bit SHT block directly accesses the sensor while the KidBright does not. In particular, the results prove that the device manager under the KidBright Engine helps manage the

hardware in support of the instruction blocks such that the number of temperature read instruction blocks does not influence the I2C bus occupancy, unlike the micro:bit.

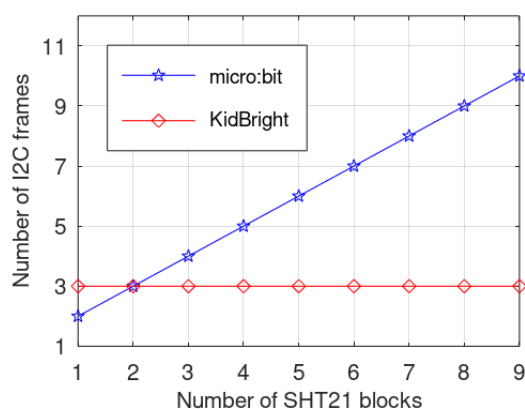


Figure 15. The number of I2C frames within 1 second vs. the number of SHT21 instruction blocks.

It is noted that occupancy comes at a cost, e.g., repeatedly reading a SHT21 in a big loop, resulting in blocking other instruction blocks, and then the whole user program cannot function practically. The device manager provides this benefit to inexperienced learners in developing their embedded programs efficiently without having to worry about this problematic hardware issue.

In the second scenario, responsiveness as a consequent result of latency is investigated, where we examine how the device manager handles a slow device while serving another device that is required to be promptly responsive. For this test, we use a KidBright board and a micro: bit in comparison again, and their block programs are shown in Figure 16. Each board is programmed to show “Hello World!” on its LED display as well as to read its push button in order to make a beep once the button is pressed. With the character length limit on their LED matrices, the whole text must be scrolled to display. While scrolling the text, the responsiveness is examined by means of manually pressing their push buttons. The results in Table 3 show that the micro:bit is responsive only when the whole message has completely been displayed, whereas the KidBright’s push button is responsive all the time, not affected by the message being scrolled. We note that the result obtained from the micro:bit is a classic embedded system issue that most learners would experience, and the KidBright’s device manager helps keep this issue away from them so that they do not perceive its existence.

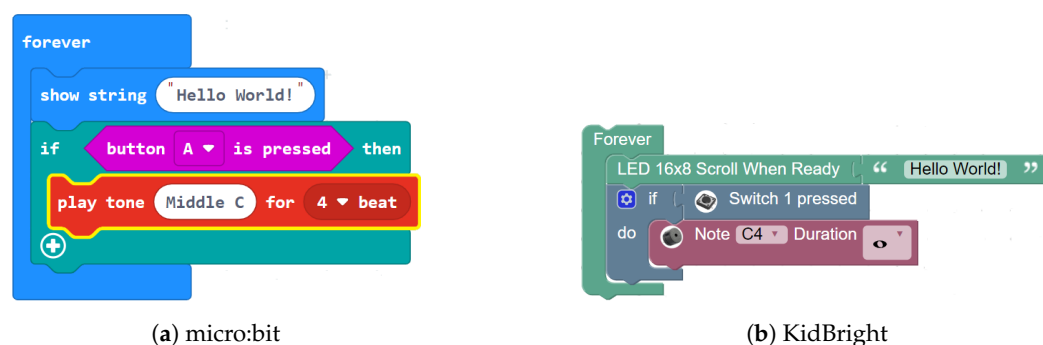


Figure 16. Block-based programs for the responsiveness investigation.

Table 3. Button switch responsiveness.

Board	Button Responsiveness	
	Display: Scrolling	Display: Complete
micro:bit	No	Yes
KidBright	Yes	Yes

5.2. DIY KidBright Board

We believe that the use of the KidBright should not be limited by the availability of the KidBright board. Therefore, we test the use of the KidBright coding environment with an off-the-shelf ESP32-WROOM breakout board, interfacing with an LED matrix display module, two LEDs, and two push button switches. All the components are connected based on the KidBright board's schematic diagram, which is available on the KidBright website. The hardware and the program for this test are shown in Figure 17. In order for the KidBright IDE to upload the program to the hardware, the USB VID and PID of the ESP32 breakout board must be added into *config.json*, which can be found in the KidBright IDE's directory.

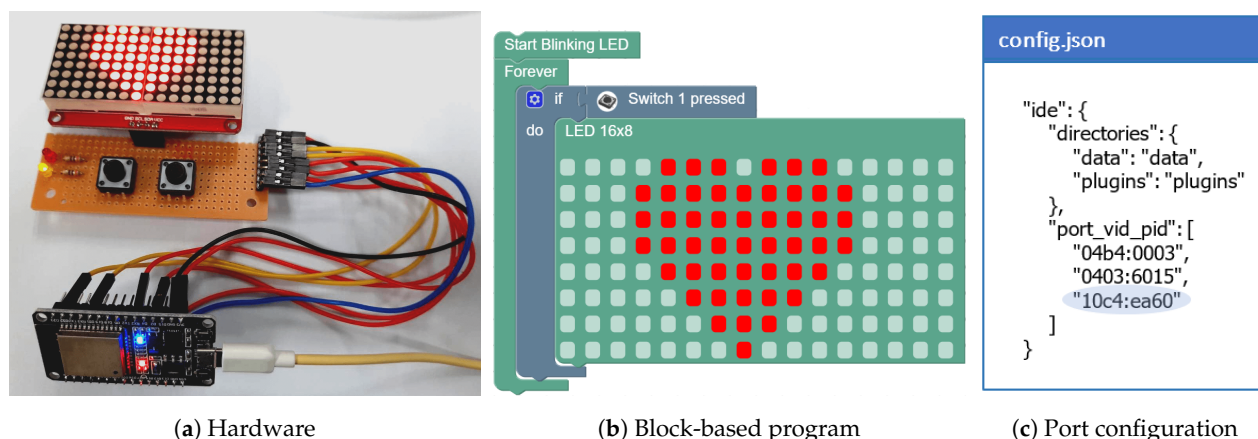
**Figure 17.** A DIY KidBright board.

Figure 17c shows the added USB VID and PID, which are 10C4 and EA60, respectively. As can be seen in Figure 17b, the program's function is to blink the yellow LED and display a heart image once button 1 is pressed. The obtained result shown in Figure 17a is exactly what the program is meant for. This shows that a KidBright board can be built by anyone as long as the ESP32-WROOM microcontroller is deployed, and the KidBright software framework supports our open-source and sustainability goals.

6. Strategy for Pervasive Dissemination

The KidBright team initiated the KidBright project around mid-2015, and by the end of the year, the first prototype of the KidBright board was successfully developed. About half a year later, in 2016, the initial version of the KidBright board along with its block-based coding software tool (not the KidBright IDE) was officially announced. However, its distribution did not take place at that time. Earlier than expected, by the time the KidBright team received funding from the Bangkok Bank Public Company in the same year, 500 KidBright boards were manufactured and delivered to seven high schools in order to demonstrate and examine the pilot use of the KidBright with their students. The KidBright team took into account feedback received from the schools and then spent an entire year revising the KidBright board as well as developing the KidBright IDE. Eventually, in 2018, the KidBright board version 1.0 was released as the start of the second generation of the KidBright board development. Subsequently, the preliminary version is no longer supported and is regarded as the first and final version of the first generation, as it is

rather different than the new version in many ways. In the same year, following version 1.0, versions 1.1 and 1.2 were also released. However, all of them are now regarded as prototypes. Once the team officially released the KidBright board V1.3, aka KidBright32, and the KidBright IDE V1.3 on 8 June 2018, distribution of the KidBright board officially began under a megaproject called “Coding at School”.

The Coding at School project was funded by the Ministry of Higher Education, Science, Research and Innovation (MHESI) on a mission to introduce coding associated with STEM to students across the country. From 2018 to 2020, the KidBright team distributed 200,000 KidBright boards of version 1.3 to more than 3200 schools at no charge. In the meantime, many training courses and activities were carried out in complement to the mission. Because the numbers of students enrolled in elementary and high schools throughout Thailand are considerably large, as shown in Table 4 [45], and the coding discipline is very new to Thai students, especially those in elementary schools, it would be difficult for us, as a small team of only eight members, to introduce the KidBright to most of them successfully without a good strategy.

Table 4. Thailand’s student populations from 2017 to 2021.

Grade		Number of Students				
		2017	2018	2019	2020	2021
Elementary School	1	791,482	834,732	829,016	767,738	752,896
	2	783,369	770,269	812,400	812,034	761,245
	3	784,342	778,044	764,701	806,418	812,750
	4	796,062	782,902	775,822	762,444	809,246
	5	798,223	794,052	779,868	772,889	765,037
	6	797,298	793,883	788,400	774,747	773,245
	Total	4,750,776	4,753,882	4,750,207	4,696,270	4,674,419
High School	7	809,520	793,310	790,042	783,188	771,500
	8	767,098	778,473	765,621	763,284	767,317
	9	740,735	732,626	742,172	733,900	743,122
	10	674,127	679,130	679,933	708,606	707,595
	11	613,997	596,203	601,783	611,161	652,013
	12	621,636	614,613	596,437	609,347	631,923
	Total	4,227,113	4,194,355	4,175,988	4,209,486	4,273,470

From this point of view, we first focused on schools that have the potential to adopt and help disseminate the KidBright, and we also arranged a training program called “Train the Trainers” aiming to train teachers as well as individuals who volunteer to be KidBright trainers. As Thailand is administratively divided into northern, northeastern, eastern, southern, western, and central regions, a considerable number of volunteers from all the different regions were recruited and trained by the KidBright team. Subsequently, they have helped conduct coding with the KidBright in local schools around their areas. In addition, we also have six KidBright supporting centers organized by six national universities located in northern, northeastern, eastern, western, southern, and central Thailand where regional KidBright activities take place. In support of learners as well as trainers, we also make use of social media that include our websites, YouTube, Facebook, and GitLab, so as to share our resources such as software tools, books, tutorial video clips, updates, bulletins, source code, and so on.

Besides the Train the Trainers, we also organize on-site and on-line workshops where students can learn the KidBright in conjunction with heuristic activities in our accommodated environments. One remarkable workshop is the Tuk Tuk Kid, in which students

learn how to build a robotic three-wheeler that represents Thailand's most recognizable transportation form known as the Tuk Tuk. In this workshop, students are motivated to learn to code using the KidBright and are also assigned to build their robotic Tuk Tuks, each of which is equipped with a KidBright board and a line detector. In order to complete the assignment, students also have to program their KidBright boards so that their Tuk Tuks can move automatically along a winding track, similar to an automated guided vehicle (AGV). Given a challenge, students must apply either a fuzzy logic or PID control technique to their code so as to get their little robots to move more accurately. Some snapshots are shown in Figure 18 and a tutorial video clip for this assignment is available on <https://www.youtube.com/watch?v=Ms5mcPbLGHM> (accessed on 28 August 2022). From this workshop, we note that students learn not only coding but also how coding can benefit them.



Figure 18. The robotic three-wheeler workshop and contest.

To motivate students to experience IoT technology through coding, we arrange workshops that demonstrate how the KidBright board enables environmental awareness with its IoT capability. One IoT-based workshop practical to real scenarios in Thailand is the dust monitoring device implementation, where we teach students how to develop a dust monitoring device by deploying a KidBright board and a particle sensor. By connecting the KidBright board to the internet, the device can sense and remotely report concentration values of PM2.5 and PM10 particles in the air where the sensor is installed, so that the air quality can be evaluated and monitored anywhere. In addition to this activity, we also offer learners a more advanced workshop from which they can learn the development of an IoT weather station that consists of a KidBright board and a set of environmental sensors. Empowered by the KidBright's IoT feature, learners can distantly aggregate basic weather data from their weather stations that are capable of measuring the air temperature, light intensity, humidity, wind speed and direction, and raindrop level.

Some activity photos from these two workshops are shown in Figures 19 and 20a. As a result of this workshop, there are more than 200 KidBright weather stations to date, distributed nationwide, as shown in Figure 20b. It is apparent that these two workshops provide learners with the idea that the KidBright board can be more than a coding learning tool.



Figure 19. A dust monitoring device workshop.



Figure 20. Development of the KidBright weather station. (a) A workshop. (b) Distribution of the weather stations in Thailand.

One of our remarkable workshops shows that the use of the KidBright is not limiting for learners with disabilities. In this workshop, more than 130 students with disabilities from 27 schools participated in our particular coding activities that could inspire them to create ideas of how coding and the KidBright board can be utilized for themselves. We always believe that the KidBright's learners have the potential to be inventors as well as innovators. In order to provide motivation, we therefore organize contests where they can present their inventions and learn from other developers as well. Since 2018, more than 900 KidBright-based projects developed by students as well as teachers have been admitted into our contests. For instance, a student team showed that a simple, inexpensive electronic piano can be built using just a KidBright board and pieces of paper. One simple but creative project developed by another student team was an automatic pet feeder that was designed specifically for the civet, whose eating habits are particular. Impressively, a teacher team proposed a teaching tool that can illustrate how gravity influences weight by deploying a KidBright board and an inertial measurement unit. Thus far, with our efforts, those workshops and contests introduced are occasionally ongoing, and a wide range of miscellaneous workshops and events not aforementioned are also carried on in support of the dissemination of the KidBright and its coding approach.

As of 2018, we have organized an annual conference series called the KidBright Developer Conference (KDC), in which presentations, forums, contests, and showcases are included in support of the KidBright. In particular, this conference gives learners, teachers, makers, and the KidBright team the opportunity to meet and share knowledge so as to grow the KidBright community. Moreover, a smaller version of the KDC is also invited to be part of two annual conferences every year, namely, the National Electronics and Computer Technology Center Annual Conference and Exhibition (NECTEC-ACE) and the National Science and Technology Development Agency Annual Conference (NAC), where novel research and development in science and engineering are presented to the public.

We note that these three conferences are significant stages for the KidBright, where many people in academia and business sectors can step in and get to know the KidBright through discussions with the KidBright team.

Initially, the primary goal of the KidBright is to support Thailand's coding education due to the lack of accessible coding learning resources suitable for the Thai people. However, around one year after the official release of the KidBright board version 1.3, the KidBright has been introduced to neighboring countries. KidBright ASEAN (ASEAN stands for the Association of Southeast Asian Nations, which includes 10 members, namely, Brunei, Cambodia, Indonesia, Laos, Malaysia, Myanmar, the Philippines, Singapore, Thailand, and Vietnam) has been introduced to countries such as Laos, Myanmar, and Cambodia through training workshops, as the first step of our road map to global coding education. With cooperation, to date, the KidBright handbooks and tutorial videos have been translated into Burmese and Lao, in addition to Thai and English. Consequently, coding using the KidBright has been gaining more attention in Southeast Asia.

Thus far, the big picture of how we disseminate the KidBright is visualized in Figure 21. From the figure, in addition to all the endeavors previously presented, one more important part of the whole plan is the partnership. Our partners include academic institutions, makers, and companies that support the KidBright in several forms. For example, we cooperate with universities and makers that help support the KidBright's regional training and activities, and also with academic professors and makers who design community versions of the KidBright boards, such as versions 1.4 and 1.5. There are maker clubs and companies that offer training courses and/or develop hardware peripherals along with plugins for the KidBright, such as Makerasia (<https://makerasia.com/> (accessed on 28 August 2022)), Khon Kaen Maker Club (<https://th-th.facebook.com/KhonKaenMakerClub/> (accessed on 28 August 2022)), Phuket Maker Club (https://web.facebook.com/groups/608886809255215/?_rdc=1&_rdr (accessed on 28 August 2022)), INEX (<https://inex.co.th/home/> (accessed on 28 August 2022)), and Gravitech (<https://www.gravitechthai.com/> (accessed on 28 August 2022)). To support the growing demand for the KidBright board, mass production of the official versions and alternative versions is run by manufacturing companies under our guidance. In addition to all of these, we also collaborate with the Institute for the Promotion of Teaching Science and Technology (IPST), a government agency under the Ministry of Education (MOE), to develop seven KidBright-based supplementary curricula for teaching coding to students in grades 4 to 10 at school.

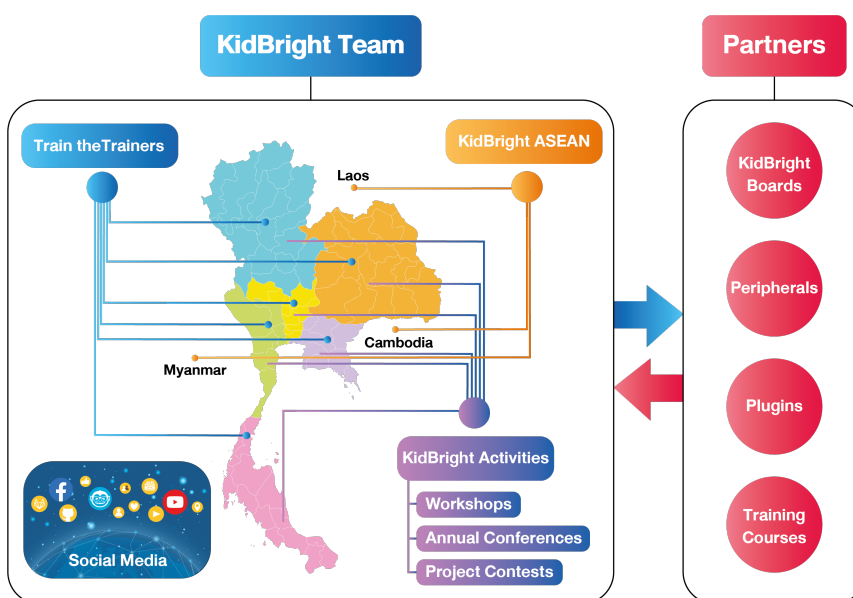


Figure 21. Dissemination plan for the KidBright.

Looking back on what we have done so far, Table 5 summarizes the estimated outcomes of the dissemination. We have trained over 8500 trainers, and the number of schools and the number of students joining the KidBright have exceeded 3700 schools and 400,000 students, respectively. The use of the KidBright simulator is increasing beyond 60,000 users (although learners are likely to use the simulator more than once, this count is based on the number of unique IP addresses that have accessed the simulator, regardless of how many times each IP address has accessed it) and an estimated number of the KidBright boards distributed across Thailand is greater than 220,000 boards. More than 900 school projects based on the KidBright have been involved in our activities. Compared to the micro:bit, the most similar platform to the KidBright (discussed in Section 5), some specifications and outcomes of both platforms are summarized in comparison in Table 6 (the information for the micro:bit is taken from <https://microbit.org/> (accessed on 30 October 2022) and [16]). The table shows that the KidBright outcomes lag far behind the micro:bit outcomes. We have realized that all these results have not yet met our targets and it is difficult to evaluate how far the KidBright has pushed Thailand's coding education. However, the KidBright is the first coding learning platform in the country that has inspired a huge number of children and teachers to become interested in coding through numerous activities. With the supplementary curricula using the KidBright, coding is also introduced in schools systematically with the recognition of the Thai government through the MHESI and the MOE. The KidBright has triggered coding education across the country.

Table 5. Estimated outcomes of the KidBright's dissemination from 2018 to 2021.

Outcome	Estimated Number
Number of trainers	>8500
Number of schools joining the KidBright	>3700
Number of students learning the KidBright	>400,000
Number of the KidBright simulator's users	>60,000
Number of KidBright boards	>220,000
Number of KidBright-based projects	>900

Table 6. Comparison between KidBright and micro:bit.

Consideration	KidBright	micro:bit
Board size	90.0 mm × 50.0 mm	51.6 mm × 42.0 mm
Coding environment	block-based (KidBright IDE)	block-based (MakeCode, Scratch), text-based (Python, Swift Playgrounds)
Support of external instruction block development	Yes	Yes
Support of expandable device connectivity	Yes	Yes
Initial distribution of official boards (year)	2018	2016
Number of users	>400,000	>25,000,000 *
Number of boards distributed	>220,000	>4,000,000 **
Learning community	Thailand, Myanmar, and Laos	>60 countries *
Partner	Local	Worldwide *

* See <https://microbit.org/about/#top> (accessed on 30 October 2022). ** This number is an estimate presented in [16] in 2020, whereas the most recent number is expected to be much higher.

7. Discussion

It has been addressed that the use of block-based coding environments can encourage young students to learn to code. However, for embedded programming, there is much more information for learners to learn and the use of a block-based coding approach alone is not sufficient. Therefore, the KidBright Engine is designed to bridge this gap. Recall that the KidBright Engine is a key component of the KidBright, as it is the software framework that helps ease embedded programming in conjunction with providing support for the sustainability concept. Therefore, instead of evaluating the KidBright block-based coding environment in terms of user satisfaction, we investigate the proposed software

framework in the context of supporting embedded programming and sustainability. The empirical investigations in Section 5 demonstrate the capability of the KidBright Engine in easing embedded programming and also supporting the implementation of affordable DIY hardware devices to be used with the KidBright IDE.

It is known that latency is a significant issue in embedded systems that imposes difficulty on inexperienced learners in developing their embedded systems practically. In the first scenario, we comparatively investigated the KidBright's and the micro:bit's expandable device connectivity through the I2C bus, which is more likely to introduce substantial latency than other hardware interfaces. The results show that the KidBright's command blocks that access the SHT21 temperature sensor introduce a constant value of latency regardless of the number of them used, whereas the micro:bit's SHT21 instruction blocks introduce more latency proportional to the number of them. This is because the KidBright instruction blocks do not access the hardware directly, instead using memory supported by the KidBright Engine's device manager. In the same way, instruction blocks that do not have hardware support from a backend have to access hardware directly. As a consequence, this embedded system issue is left for learners to handle on their own.

In the second scenario, we investigated the responsiveness of a push button switch in the presence of a slow-operated LED matrix display. The investigation was carried out using a KidBright board and a micro:bit with identical block-based programs and hardware setups. The results show that while their displays are scrolling the text message, the KidBright board's button switch is responsive, whereas the micro:bit's is not. Again, this is what we can expect from the KidBright in resolving latency, similar to the case of I2C expandable device connectivity. We note that responsiveness is a classic issue in embedded programming and in order to resolve it for the micro:bit, learners are recommended to code by separating the program into two parts: one for the LED display and the other one is for the button, based on the concurrency, event-driven principle introduced in [46]. For beginners, this principle would be difficult for them to understand properly, even though they can follow MakeCode's example code in order to resolve this responsiveness issue. Furthermore, with the support of the KidBright Engine, it is inferred that without loss of generality, coding using any other KidBright instruction blocks that access hardware peripherals guarantees predictable and minimal latency because all of the behind-the-scenes hardware activities are intrinsically put into the background in a concurrent fashion through the device manager.

Regarding all the results obtained from the KidBright investigations, it might be argued that learners will lack knowledge of real embedded programming issues, as the KidBright Engine hides them in such a way that learners do not even recognize their existence. On the other hand, for beginners with no experience in coding and embedded systems, we claim that our design can potentially help engage and motivate them to take the first step into embedded programming. Thus, the KidBright can help prepare them for more advanced learning in the next step and keep them from giving up too soon. Furthermore, the third scenario demonstrates how the KidBright Engine supports a low-cost KidBright board built from off-the-shelf hardware components. We point out that the KidBright board is based on the ESP32-WROOM microcontroller, and it is not necessary for a DIY KidBright board to have all of the hardware components such as the official KidBright board in order to work with the KidBright IDE. The KidBright Engine's architecture allows any ESP32-WROOM breakout board to be used with the KidBright coding environment through the hardware configuration file, and we note that this is also part of our contributions to supporting ecosystems for learning to program embedded systems. It should be known that modified versions of the official KidBright board are also allowed, as long as the microcontroller and its pinout match the schematic diagram provided on the KidBright website.

Our primary goal is to support coding education in Thailand. To achieve it, the hardest part is not the development of the KidBright but the introduction of coding with KidBright to students as well as teachers throughout the country. At this point, we should note that we encountered no difficulties during the development; rather, we have found greater

difficult during dissemination. We had a difficult time finding support for funding to begin our first initial distribution. As mentioned in the first paragraph of Section 6, we received an amount of funding sufficient for distributing only 500 KidBright boards. However, with this funding, we also had the opportunity to revise our prototypes and then introduce the first official version of the KidBright board along with the KidBright IDE. Consequently, we received a larger amount of funding to distribute 200,000 KidBright boards across the country. Then, we encountered another difficulty regarding this mission, which was how we, as a group of 8 people, could distribute 200,000 KidBright boards and introduce coding to a huge number of students and teachers. This difficulty then turned us to setting up a dissemination plan. Section 6 describes our dissemination strategy which includes practical activities carried out over a few years. We found that the Train the Trainers project helps us increase the number of trainers who can teach coding to their students in their local areas, and all the workshops and miscellaneous activities can potentially attract students' attention to coding. However, we always realize that, as a small team, we cannot keep all the activities provided by us going on for a long time. We expect to see a growing learning community to which people who are interested in KidBright can contribute. Thus, in addition to trainers, learners, and motivational events, partners are also important. In order to get them involved, what we have planned from the start is to make the KidBright open at no cost, including all software tools, circuit diagrams, learning resources, and the software framework along with source code. In particular, our software framework is open in such a way that people other than us can contribute to the community by developing hardware peripherals and plugins. With all the efforts, the KidBright's current status has shown that its use is now growing on its own.

8. Conclusions

In this paper, we addressed how coding is important to children and how block-based coding can be a promising approach to teaching children to code. In particular, we identified the barriers to Thailand's coding education, which include the lack of accessible, easy-to-understand coding learning resources as well as the limited number of teachers with sufficient experience in coding. We have developed the KidBright as a potential open-source coding learning platform that can motivate children to learn to code and develop embedded systems. Specifically, the KidBright consists of the KidBright board as the dedicated computing hardware, the KidBright IDE for coding and programming the KidBright board, and the KidBright simulator for learning to code without the KidBright board. The KidBright supports both Thai and English.

We have described our software framework, called the KidBright Engine, as the backend of the KidBright IDE, which eliminates unnecessary software and hardware issues in support of learners with no embedded programming experience. We claim that the KidBright can help broaden learning to code through the KidBright Engine's device manager and the KidBright board's device expansion paradigm called the KidBright Chain, because it allows other people besides the KidBright team to develop new command blocks and hardware peripherals in support of the KidBright. The KidBright plugin development approach has also been introduced, by example, so that makers can help contribute to the KidBright community. Furthermore, we have demonstrated the potential of the KidBright Engine through three empirical investigations. The results have shown that the KidBright Engine can resolve hardware latency in support of expandable device connectivity and the existence of slow devices that require time to operate. The results have also shown that the KidBright Engine allows the use of inexpensive DIY KidBright boards built from off-the-shelf hardware components to work with the KidBright IDE.

Finally, we have presented how the KidBright team, which has only eight members, introduced coding using the KidBright to a large number of students and teachers across Thailand. With our well-planned strategy, we united trainers, learners, and partners to help disseminate the KidBright coding approach through many activities, mainly including workshops, contests, and conferences. In addition, the KidBright's resources, such as

software tools, circuit diagrams, books, tutorial video clips, source code, and so on, are made available through social media for free. We believe that this direction supports the concept of sustainability, which grows the KidBright as an ecosystem, leading to pervasive coding education. We also claim that the KidBright has officially triggered the country's coding education, as it is the first coding learning platform that potentially encourages a large number of students and teachers to learn to code systematically. Although the outcomes obtained have not yet reached our targets, the KidBright community has shown us that the use of the KidBright has been growing on its own, leading to widespread coding education in return.

To increase the use of the KidBright for coding education, in our future work we will introduce how to develop DIY KidBright boards, as we realize that the expense and the availability of a KidBright board are obstacles to introducing embedded programming to many learners. In this direction, learners would be able to build affordable KidBright boards on their own or with the support of their KidBright communities. According to the strategy plan, we will also cooperate with more partners and intensively further the use of social media to help introduce the KidBright. Moreover, we have planned to introduce the KidBright to global coding education, and to create opportunities to cooperate with many more partners outside Thailand and broaden the KidBright community.

Author Contributions: Conceptualization, S.T.-a.-r. and D.T.; methodology, A.L. and A.I.; software, A.L., S.K. (Somphong Kittipiyakul) and A.I.; validation, S.T.-a.-r., A.L., T.S. and D.T.; investigation, A.L. and D.T.; resources, W.P.; data curation, S.K. (Somphong Kittipiyakul), W.P. and T.S.; writing—original draft preparation, S.T.-a.-r., S.K. (Saowaluck Kaewkamnerd) and D.T.; writing—review and editing, D.T.; visualization, S.K. (Somphong Kittipiyakul), W.P. and T.S.; project administration, S.K. (Saowaluck Kaewkamnerd); funding acquisition, S.K. (Saowaluck Kaewkamnerd) and A.I. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Electronics and Computer Technology Center (NECTEC), and in part by the National Science and Technology Development Agency (NSTDA) through projects P2052225 and P2150643.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors are thankful to Karanpas Jaitha, Thitimaporn Sangkong, Oraya Kaewchang, and Parichat Soisungwan for their assistance in preparing supporting information.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

STEM	Science, Technology, Engineering, and Mathematics
IDE	Integrated Development Environment
VID	Vendor ID
PID	Product ID
DIY	Do-It-Yourself
MHESI	Ministry of Higher Education, Science, Research and Innovation (Thailand)
IPST	Institute for the Promotion of Teaching Science and Technology (Thailand)
MOE	Ministry of Education (Thailand)

References

1. Alghamdi, M.Y. Supporting The Learning of Computer Programming in An Early Years Education. Ph.D. Thesis, Liverpool John Moores University, Liverpool, UK, 2017.
2. Macrides, E.; Miliou, O.; Angeli, C. Programming in Early Childhood Education: A Systematic Review. *Int. J. Child Comput. Interact.* **2022**, *32*, 100396. [[CrossRef](#)]

3. Kaplancali, U.T.; Demirkol, Z. Teaching Coding to Children: A Methodology for Kids 5+. *Int. J. Elem. Educ.* **2017**, *6*, 32–37. [CrossRef]
4. Bers, M.U. Coding and Computational Thinking in Early Childhood: The Impact of ScratchJr in Europe. *Eur. J. STEM Educ.* **2018**, *3*, 8:1–8:13. [CrossRef]
5. Critten, V.; Hagon, H.; Messer, D. Can Pre-school Children Learn Programming and Coding Through Guided Play Activities? A Case Study in Computational Thinking. *Early Child. Educ. J.* **2022**, *50*, 969–981. [CrossRef]
6. Okal, G.; Yildirim, B.; Timur, S. The Effect of Coding Education on 5th, 6th and 7th Grade Students' Programming Self-Efficacy and Attitudes About Technology. *Educ. Policy Anal. Strateg. Res.* **2020**, *15*, 143–165. [CrossRef]
7. Fessakis, G.; Gouli, E.; Mavroudi, E. Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Comput. Educ.* **2013**, *63*, 87–97. [CrossRef]
8. Sentance, S.; Csizmadia, A. Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Educ. Inf. Technol.* **2017**, *22*, 469–495. [CrossRef]
9. Kjällander, S.; Mannila, L.; Åkerfeldt, A.; Heintz, F. Elementary Students' First Approach to Computational Thinking and Programming. *Educ. Sci.* **2021**, *11*, 80. [CrossRef]
10. Balanskat, A.; Engelhardt, K. *Computing Our Future: Computer Programming and Coding—Priorities, School Curricula and Initiatives across Europe*; European Schoolnet: Brussels, Belgium, 2015.
11. Weintrop, D.; Wilensky, U. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *Int. J. Child Comput. Interact.* **2018**, *17*, 83–92. [CrossRef]
12. Kiraly, S.; Balla, T.; Kusper, G. Code School: Learn to Code by Practicing. In Proceedings of the 11th Joint Conference on Mathematics and Computer Science, Eger, Hungary, 20–22 May 2016; pp. 136–143.
13. Maloney, J.; Resnick, M.; Rusk, N.; Silverman, B.; Eastmond, E. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 16. [CrossRef]
14. Popović, J.; Korolija, N.; Marković, Ž.; Bojić, D. Developing algorithmic skills of pupils in Serbian schools using code.org materials. In Proceedings of the 25th Telecommunication Forum (TELFOR 2017), Belgrade, Serbia, 21–22 November 2017; pp. 1–4.
15. Noor, F.H.; Mohamad, F.S.; Minoi, J.L. Learning Programming using Lego Mindstorms: Analysis of Learner Experiences. In Proceedings of the 2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC), Kuching, Malaysia, 1–3 December 2020; pp. 1–6.
16. Austin, J.; Baker, H.; Ball, T.; Devine, J.; Finney, J.; Halleux, P.D.; Hodges, S.; Moskal, M.; Stockdale, G. The BBC micro:bit—From the U.K. to the World. *Commun. ACM* **2020**, *63*, 62–69. [CrossRef]
17. Bicer, A.; Lee, Y.; Capraro, R.M.; Capraro, M.M.; Barroso, L.R.; Bevan, D.; Vela, K. Cracking the Code: The Effects of Using Microcontrollers to Code on Students' Interest in Computer and Electrical Engineering. In Proceedings of the 2018 IEEE Frontiers in Education Conference (FIE), San Jose, CA, USA, 3–6 October 2018; pp. 1–7.
18. Nite, S.B.; Bicer, A.; Currens, K.C.; Tejani, R. Increasing STEM Interest through Coding with Microcontrollers. In Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 21–24 October 2020; pp. 1–7.
19. Google. Introduction to Blockly. Available online: <https://developers.google.com/blockly/guides/overview> (accessed on 20 August 2022).
20. Bonar, J.G.; Liffick, B.W. *A Visual Programming Language for Novices*; Technical Report AIP-22; Carnegie Mellon University: Pittsburgh, PA, USA, 1987.
21. Begel, A. *LogoBlocks: A Graphical Programming Language for Interacting with the World*; MIT Media Laboratory: Cambridge, MA, USA, 1996.
22. Huisman, M.; Monti, R.E. Teaching Design by Contract using Snap! In Proceedings of the 2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG), Madrid, Spain, 24 May 2021; pp. 1–5.
23. Biju, S.M. Taking Advantage of Alice to Teach Programming Concepts. *E-Learn. Digit. Media* **2013**, *10*, 22–29. [CrossRef]
24. Bau, D.; Bau, A.; Dawson, M.; Pickens, C.S. Pencil code: Block code for a text world. In Proceedings of the 14th International Conference on Interaction Design and Children, Boston, MA, USA, 21–24 June 2015; pp. 445–448.
25. Google. Blockly Games. Available online: <https://blockly.games/> (accessed on 21 June 2022).
26. Yadagiri, R.G.; Krishnamoorthy, S.P.; Kapila, V. A Blocks-based Visual Environment to Teach Robot-Programming to K-12 Students. In Proceedings of the 122nd ASEE Annual conference & Exposition, Seattle, WA, USA, 14–17 June 2015; pp. 26.17.1–26.17.12.
27. Chung, C.J.; Shamir, L. Introducing Machine Learning with Scratch and Robots as a Pilot Program for K-12 Computer Science Education. *Int. J. Learn. Teach.* **2021**, *7*, 181–186. [CrossRef]
28. Miller, J. STEM education in the primary years to support mathematical thinking: Using coding to identify mathematical structures and patterns. *Springer ZDM Math. Educ.* **2019**, *51*, 915–927. [CrossRef]
29. Feijóo-García, P.G.; Wang, S.; Cai, J.; Polavarapu, N.; Gardner-McCune, C.; Ragan, E.D. Design and evaluation of a scaffolded block-based learning environment for hierarchical data structures. In Proceedings of the 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Memphis, TN, USA, 14–18 October 2019; pp. 145–149.
30. Judvaitis, J.; Elst1, A.; Selavo, L. Demo Abstract: SEAL-Blockly: Sensor Network Visual Programming Using a Web Browser. In Proceedings of the 10th European Conference on Wireless Sensor Networks (EWSN 2013), Ghent, Belgium, 13–15 February 2013; pp. 1–4.

31. Barkana, B.Z.; McDonough, W. AP Computer Science Principles: Designing the hour.ly App in MIT App Inventor. In Proceedings of the 2019 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 3 May 2019; pp. 1–6.
32. Codejig. Available online: <https://www.codejig.com/en/> (accessed on 22 August 2021).
33. Tinkercad. Available online: <https://www.tinkercad.com/learn/codeblocks> (accessed on 22 August 2021).
34. Weintrop, D.; Shepherd, D.C.; Francis, P.; Franklin, D. Blockly Goes to Work: Block-based Programming for Industrial Robots. In Proceedings of the 2017 IEEE Blocks and Beyond Workshop (B&B), Raleigh, NC, USA, 9–10 October 2017; pp. 29–36.
35. Mayr-Dorn, C.; Winterer, M.; Salomon, C.; Hohensinger, D.; Ramler, R. Considerations for using Block-Based Languages for Industrial Robot Programming—A Case Study. In Proceedings of the 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE), Madrid, Spain, 2 June 2021; pp. 5–12.
36. Ariza, J.Á. DSCBlocks: An Open-Source Platform for Learning Embedded Systems Based on Algorithm Visualizations and Digital Signal Controllers. *Electronics* **2019**, *8*, 228. [CrossRef]
37. Devine, J.; Finney, J.; de Halleux, P.; Moskal, M.; Ball, T.; Hodges, S. MakeCode and CODAL: Intuitive and efficient embedded systems programming for education. *J. Syst. Archit.* **2019**, *98*, 468–483. [CrossRef]
38. Junior, A.G.D.S.; Gonçalves, L.M.G.; Caurin, G.A.D.P.; Tamanaka, G.T.B.; Hernandes, A.C.; Aroca, R.V. BIPES: Block Based Integrated Platform for Embedded Systems. *IEEE Access* **2020**, *8*, 197955–197968. [CrossRef]
39. micro:bit. Let's Code. Available online: <https://microbit.org/code/> (accessed on 2 August 2021).
40. Scratch Blocks. Available online: <https://scratch.mit.edu/developers> (accessed on 20 August 2022).
41. Microsoft MakeCode. Available online: <https://github.com/Microsoft/pxt> (accessed on 20 August 2022).
42. Snap-Materials. Available online: <https://snap.berkeley.edu/materials> (accessed on 28 August 2022).
43. KidBright Download. Available online: <https://www.kid-bright.org/download-kidbright/> (accessed on 15 August 2021).
44. ESP-IDF FreeRTOS (SMP). Available online: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html#id1> (accessed on 23 August 2022).
45. NSO. Education Branch. Available online: <http://statbbi.nso.go.th/staticreport/page/sector/en/03.aspx> (accessed on 2 August 2021).
46. micro:bit. The micro:bit—A Reactive System. Available online: <https://makecode.microbit.org/device/reactive> (accessed on 23 August 2022).