Bull. Nov. Comp. Center, Comp. Science, 47 (2023), 23-31

Creating a prototype of an IoT development web platform

Alexander Mordvinov

Abstract. Nowadays, the creation of custom electronic devices of varying degrees of complexity is available for a wide audience. The applications vary from simple devices such as clocks and weather stations to automated homes and manufacturing control systems. In the last decade, the Arduino platform [1] has significantly simplified the development and, as a result, has lowered the entry barrier. However, this platform lacks some basic functions such as task management and state synchronization, hence, has opportunities for improvements. This work aims to offer a new platform and open the field of custom electronics for a broader audience.

Keywords: IoT, Electronics, Microcontrollers, Development platform, Web, Arduino, ESP.

Introduction

Over the past decade, microcontroller development has become more accessible. The Arduino platform has greatly simplified their programming, thereby has opened the way for enthusiasts to enter the field. It has become possible to create useful and functional projects without having to deep dive into the work of programmer hardware and kernel loaders, without studying the technical documentation of microcontrollers and working with its control registers.

However, user ambitions are growing, projects are becoming more complex, and sometimes amateur developments are superior to similar products in price and quality. Some of the modern development boards contain WiFi and Bluetooth modules[2][3], which opens up the possibility of wireless communication and moves the devices into the Internet of Things (IoT) area with a broad field of applications[4]. The increasing complexity of projects makes it clear that by today the Arduino platform is becoming insufficient.

The Arduino code contains verbose, not always intuitive constructions that are repeated from project to project, often copied by users from examples without proper understanding. Moreover, such complexity of code is almost always unrelated to the problem being solved. However, a significant part of the applied tasks of the Internet of Things can be solved by a more straightforward, reactive or event-driven approach. In other words, a solution can be described by reactions to external or internal events of the system, and task scheduling. Working with a peripheral is also not always an intuitive process - the lack of standardization of libraries complicates the initialization and usage of software components.

The goal of this work is to create a prototype of an IoT development platform, which abstracts out all the details that are not essential for solving problems, such as the selection of libraries, their configuration and connection, initialization of components, task scheduling and state synchronization. Such a platform will simplify the development approach and attract a larger audience to the field.

1. Existing platforms overview

There are existing solutions aiming for the same goal of simplifying and standardizing IoT development. Xod.io [5] is a platform that provides a graphical development language for microcontrollers and a web editor. The language is based on blocks, which can be either electronic components or logical blocks. The connection of these blocks determines the functionality of the project. As elegant as it might seem for primitive projects, the platform, however, fails to scale. As the size of a diagram increases, it becomes difficult to read, edit, and debug. An example is shown in Figure 1.

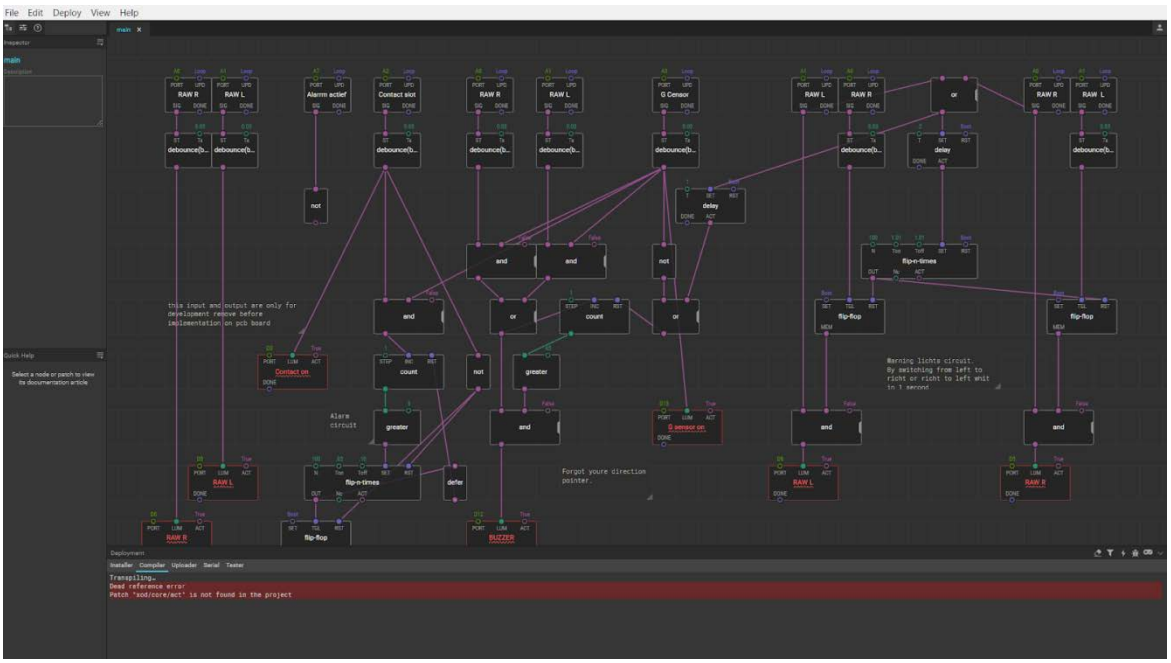

Figure 1. Xod.io project example

Thus, the Xod.io project, thanks to its graphical programming language, is an excellent tool for getting started with electronics development, but quickly becomes unusable in practice.

The other platform aiming for the simplification of development is ESPHome[6]. Its projects are based on configuration files of the YAML format.

```
sensor:
  - platform: dht
    pin: 16
    model: DHT22
    temperature:
      name: "Example Temperature"
    humidity:
      name: "Example Humidity"
    update_interval: 10s
```

Figure 2. Creating of the DHT22 component on the YAML language

Such an approach, combined with the platforms rich open component library, has shown to be effective in practice - lots of real projects can be found on the Internet. However, it has limitations for advanced use cases. The only way for a user to execute custom code is to place it in the configuration file as string. So the code and configuration are mixed together in a text file, hence the project becomes hard to maintain. Moreover, the platform is deeply embedded in another project – HomeAssistant [7], so its applications are limited to home automation and requires users to install and maintain additional infrastructure.

2. Platform overview

2.1. Platform primitives and visual configurator

The Arduino platform involves writing programs in the Arduino C language - this approach is imperative and verbose. The solutions reviewed earlier, offer their own methods - a graphical language and configuration files - such approaches quickly run into the limitations they generate.

This work proposes a symbiosis of configuration and source code. Configurators were implemented as visual forms with a set of validated fields. Entities generated from forms were named *system primitives*. The business logic, in its turn, is specified in the code editor in which these primitives can be used.

Three such primitives were defined:

- **Components** - Peripherals. Electronic components such as buttons, temperature sensors, displays, etc.
- **Tasks** - blocks of code that run at a certain time interval.
- **State** - global application state variables available in program code.

The platform is implemented as a web application, which requires no infrastructure setup by the user. For a prototype phase, it was decided to limit the supported MCU (microcontroller unit) to the ESP family by Espressif Systems, which is gaining popularity in the field, due to its low cost, on-board wifi and BT interfaces.

2.2. Project editor

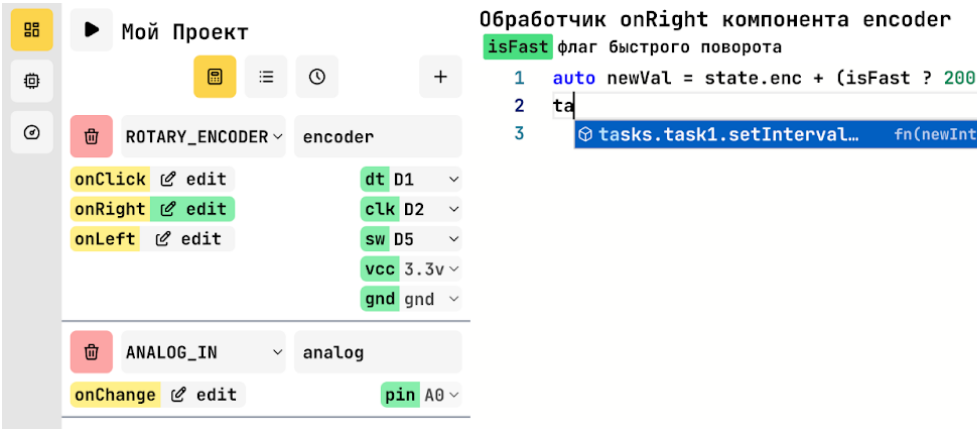

Figure 3. Project editor

The project editor (Figure 3) is designed as a separate screen of the platform interface. It is divided into two parts. On the right is a full-fledged code editor with syntax highlighting and autocompletion support. On the left there is a visual editor for system primitives, divided into three tabs - *Components*, *Global State* and *Tasks*. Let us look at each of them.

2.3. Components

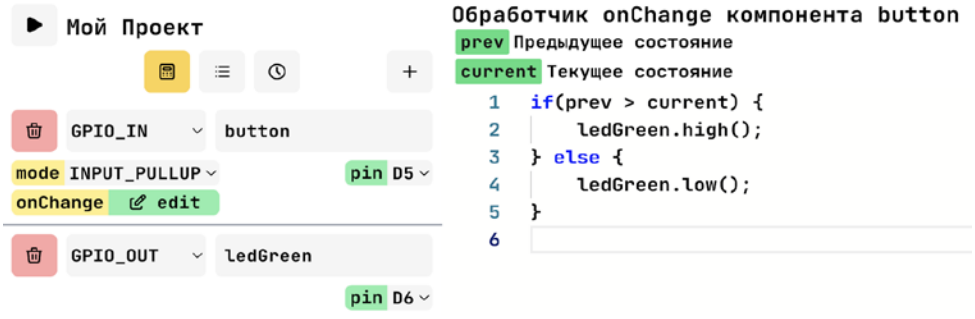

Figure 4. Components initialization and event handler editing

The MCU involves connecting electronic modules to it. To work with them, the system proposes the *Components* primitive. Practically, they can be divided into two types:

- components that generate an event (button, high-speed sensors (for example, motion detection sensor)) and
- components executing the command (LED, display, relay).

Both types are initialized visually, using the visual editor. Each is given a name, pinout (description of connection to the board) and, if necessary, initialization parameters. The first type of components supports event handlers that are defined by the user in the code editor. The second type of component declares a set of methods for working with it - these methods are available for invoking in the code editor. In Figure 4, the state change handler for a component named *button* uses the *high()* and *low()* methods of the *ledGreen* component.

Thus, the platform reduces the task of connecting a peripheral to specifying the pinout and necessary parameters, so the work with components is unified and tightly linked to the user interface, thereby minimizing the potential errors in initialization and usage are minimized.

2.4. Tasks

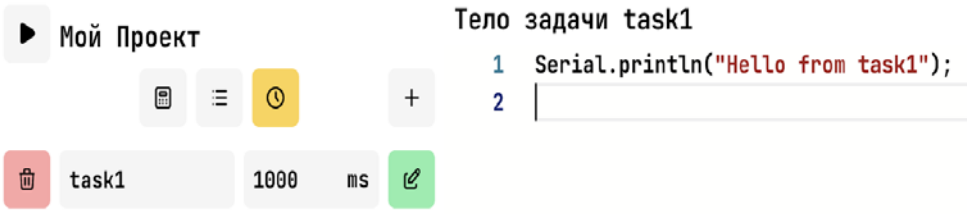

Figure 5. Adding task *task1* and its body editing

In almost every project there is a need to run a code at certain intervals. This platform proposes to move such code into a separate primitive called *Tasks*. Tasks are initialized using the forms on the left side of the editor – the user specifies its name and execution interval in milliseconds. The task body is defined by the user in the code editor on the right. A task has methods to control it (start/stop, change the interval), which are available for calling both in component handlers and in task bodies (including in the task itself). The platform takes care of all the work on task scheduling, so users do not have to do it.

2.5. Global state

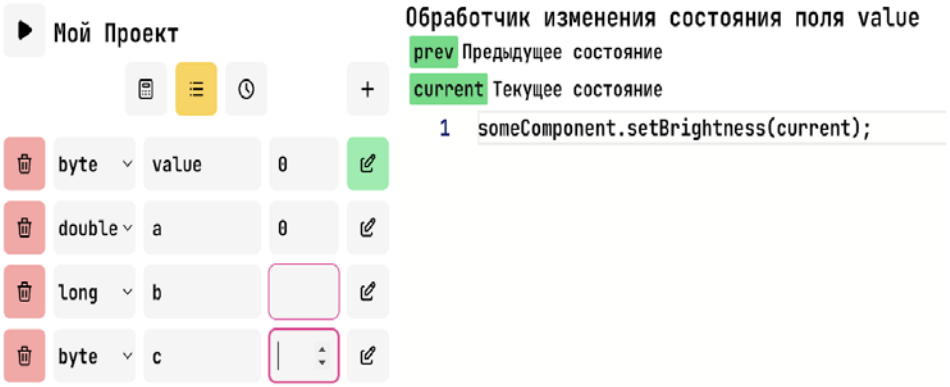

Figure 6. Working with the global state

The *Global State* primitive is a set of fields - variables of the standard C++ language type (double, int, char, etc.). Visually, the user specifies the type, name, and initial value. Similar to the event handlers of the Component primitive, each state field can be assigned a code to run on value change. As can be seen from Figure 6, the system values *prev* and *current* are available in the handler code - the previous and current states of the field.

These fields are available in the code of any handler and the body of any task. To access the value of a field, one should use the *state.fieldName* construction; and *state.setFieldName(newValue)* to change the value.

The state is synchronized between a group of microcontrollers and the UI. For example, a change in the field state initiated by a task on the microcontroller A could, if necessary, be processed by the microcontroller B. The system manages all the jobs of initialization, connection setup, synchronization and handlers invoking.

2.5. Boards and widgets

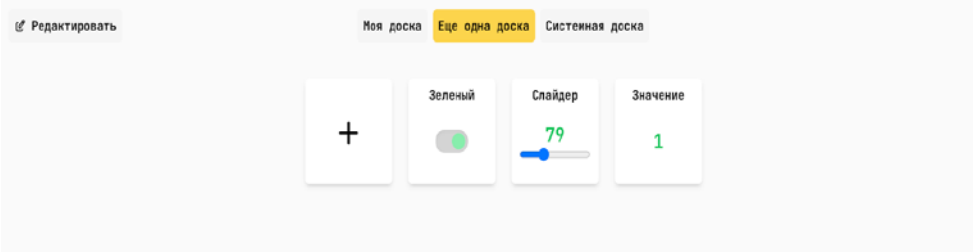

Figure 7. Board example with widgets of different type

The global state of the system is available to all its participants, including the user interface. A widget mechanism has been introduced to display and modify the state. A widget is a UI card directly linked to a previously created global state field. Currently, the platform provides the following three types of widgets:

- **Value**. The field value is displayed as it is, providing no way to modify it.
- **Switch**. The field value is displayed as a switch button. It is on for any non-zero value and off for the zero value. Pressing the switch initiates a change to the opposite value.
- **Slider**. A slider input displays the current value of the field. Moving the slider initiates a state field change.

The field type determines the types of widgets available for it. For example, a slider widget would not make sense for bool value. Widgets are created and edited on a special UI page called “Boards”. A board is a group of widgets with a given name. The user can create an arbitrary number of them and add an arbitrary number of widgets to each. In addition to user boards, the platform provides a special one - the *system board* (Figure 8). It automatically creates the widgets of type Value for each field of the global system state and is not editable.

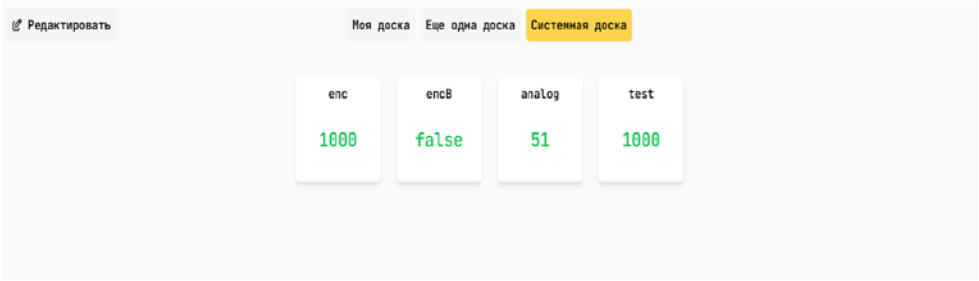

Figure 8. System board

2.6. Network management

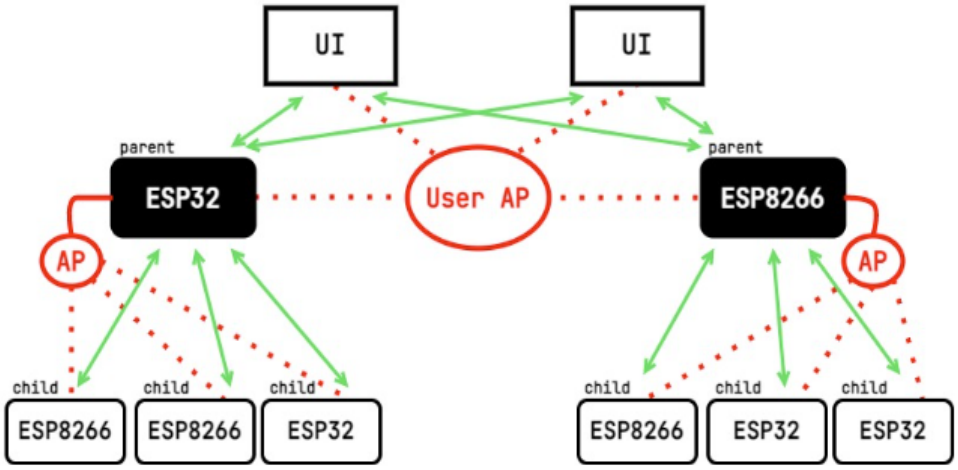

Figure 9. Overall diagram of a microcontroller system built with the platform

The platform involves working with several microcontrollers simultaneously. As described earlier, the global state primitive is synchronized between all system participants and can be controlled from the platform's UI through a widget mechanism. Hence the need for network management arises.

The ESP family MCUs support WiFi. The MCU can both connect to an access point (AP) and open its own AP. Since the global state must be synchronized within the network of microcontrollers, it was decided to give state orchestration to the “single source of the truth”. For this purpose, the MCUs were divided into two roles.

Parent:

- connects, if possible, to the WiFi access point specified in the firmware;
- launches its own WiFi access point;
- starts WebSocketServer and receives state change notifications from other participants;
- when the state changes, it sends a broadcast message to other MCUs and the user interface.

Child:

- connects to the parent's access point;
- starts the WebSocketClient and connects to the parent's WebSocketServer;
- listens to broadcasted *state change* messages and processes them by modifying its local copy.

Thus, the parent and the children connected to it, form a group of modules. Any update to the global state of the group passes through a request to the parent MCU. A positive

side effect of this solution is offloading of the user network. In Figure 9, a network of eight modules adds only two clients on the user's WiFi access point.

3. Conclusion

As a result of this work, a prototype of an IoT development web platform was implemented. The key principle of the platform is to abstract out three primary aspects of IoT development as language primitives. These primitives are specified visually through the user interface. Thus, the mechanisms for installing and connecting libraries, initializing components, synchronizing the state and scheduling tasks do not require any effort from the user. Moreover, the platform, through a widget mechanism, provides a built-in UI constructor. All these features combined, allow the user to begin solving the real-world problems immediately, without diving into language or hardware specific features, which fulfills the goal of the work.

References

[1] Arduino platform - https://www.arduino.cc/reference/en/

[2] D. Singh, A. Sandhu, A. Thakur, and N. Priyank, “An overview of IoT hardware development platforms,” International Journal on Emerging Technologies, 11 (5) 155–163 (2020).

[3] Singh KJ, Kapoor DS (2017) Create Your Own Internet of Things: A survey of IoT platforms. IEEE Consum. Electron. Mag. 6(2):57–68. https://doi.org/10.1109/MCE.2016.2640718.

[4] P. Suresh, J.V. Daniel, V. Parthasarathy, R.H. Aswathy, A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment, Int. Conf. Sci. Eng. Manag. Res. (ICSEMR) 2014 (2014) 1–8, http://doi.org/10.1109/ICSEMR.2014.7043637.

[5] XOD.io - https://xod.io/.

[6] ESPHome - https://esphome.io/.

[7] HomeAssistant - https://www.home-assistant.io/.