



Article

Simulation Oriented Layer of Embedded Software Architecture for Rapid Development of Custom Embedded Systems Virtual Simulators Used in Didactics

Andrzej Radecki  and Tomasz Rybicki * 

Institute of Automatic Control, Lodz University of Technology, 90-537 Lodz, Poland; andrzej.radecki@p.lodz.pl

* Correspondence: tomasz.rybicki@p.lodz.pl

Abstract: The application of the proposed Simulation Oriented Layer in the embedded-software architecture is shown in this paper. The SOL's purpose is to deliver only limited and highly desirable microprocessor-system functionality to the Application Layer, which would be implemented in a virtual simulator without requiring its complex development. It was used in two virtual simulators of embedded systems, as presented in the article. Each virtual simulator covers one customized embedded system (RPILAB and TMSLAB) used for didactical purposes. On each embedded platform, a different method of system-functionality simulation was shown. Presented virtual simulators can run recompiled (for the virtual-simulator platform) programs in a seamless process, giving real-like experiences for programmers, who can verify and test their high-level solutions. Being accurately chosen, taken for the simulation because of essential and limited functionality, and used in the Application Layer allowed for the rapid design of the virtual simulators. Unit- and functional-test results using RPILAB- and TMSLAB-embedded systems and their virtual simulators are shown in this paper. Both simulators of real RPILAB and TMSLAB platforms are used with success in the didactical process, at the Institute of Automatic Control in Lodz University of Technology, since the COVID-19 pandemic.

Keywords: virtual simulator; embedded-system virtual platform; embedded-system software architecture; software layers architecture



Citation: Radecki, A.; Rybicki, T. Simulation Oriented Layer of Embedded Software Architecture for Rapid Development of Custom Embedded Systems Virtual Simulators Used in Didactics. *Appl. Sci.* **2022**, *12*, 6322. <https://doi.org/10.3390/app12136322>

Academic Editor: Luigi Pomante

Received: 18 May 2022

Accepted: 18 June 2022

Published: 21 June 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/>).

1. Introduction

Embedded-system design nowadays plays a key role in the development of industry, science and technology, and many areas of the economy [1,2]. Embedded systems are special-purpose computer systems that integrate a hardware component (a microprocessor or microcontroller with the necessary peripheral environment) with software, to perform a specific function in a larger electronic or mechanical system [3]. Over the past several years, there has been a great deal of research work focused on creating and improving various approaches and tools for teaching as well as supporting the process of designing and simulating embedded systems. The following can be distinguished: an integrated approach to teaching of embedded systems [4], the use of the Linux system [5], a motivation-driven approach [6], a teaching process with graphical-system-design methodologies [7], application-oriented teaching [8], teaching based on outcomes-based education [9], and many others [10–14].

An important part of the development of the tools to support the design of embedded systems are the various types of simulation tools, virtual platforms, and virtual labs [15–18]. Virtual platforms of embedded systems are the key to software's modern and rapid development process [19–21]. The programming process, using those platforms, can start in the early-project stage, in parallel to hardware development [22], so in the very early stage of an embedded project. There are many solutions in the field of virtual platforms [23]. They can be oriented on precise system emulation [24,25] or try to imitate the system behavior of

embedded systems through their simulation [26]. Virtual simulators can imitate a whole system [27,28] or can reproduce behavior at the hardware level [29]. A mixed approach is also common [30], to prepare an efficient virtual environment for embedded-system programming [18].

Software modeling for embedded systems using such virtual platforms can reduce the time to market and allow for performing extensive unit tests [31] that could be troublesome in real-device [32] design. For didactic purposes in the time of the COVID-19 pandemic, such virtual platforms are invaluable at any teaching level [33–35].

There are many solutions in the area of virtual platforms and simulators [19–21]. To name a few, we have hardware emulators, such as QEMU or IDEs, with hardware simulation support (e.g., Keil μ Vision (Richardson, TX, USA), Microchip Studio (Chandler, AZ, USA)). There are system-level simulators that can reproduce the behavior of different software architectures (GFX simulator (New York, NY, USA), Qt simulator (Espoo, Finland)) or a whole embedded operating system such as FreeRTOS.

In case of developing a highly customized embedded system, with a so-called in-house system (which is a common situation) and a very specific hardware setup, we cannot expect that a simulator will be available. Both the software and the virtual hardware platform can be unavailable for either emulation or simulation purposes.

In our previous paper [36], a simple and effective algorithm for the precise embedded-system peripherals' visualization used in virtual simulators was presented.

The true motivation for this paper as well as for the accompanying work was the activity of our ambitious students. With every new class of microprocessor techniques or embedded-software-development labs, based on TMSLAB or RPILAB systems, the same question arises: are virtual simulators of the embedded systems available? The paper tries to answer that question and satisfy the needs of students by giving them the tools for developing programs without direct hardware. The first ideas and main concepts were created some time before the COVID-19 pandemic, but the real need for virtual simulators arose during the lockdown, to allow students continue their labs.

In this article, two custom virtual-simulator solutions for highly customized embedded systems are presented. Such simulation software can be easily developed on the assumption of the serious simplification of the embedded-system hardware and software. Such a simplification would consider only the most desirable, limited functionality and allow for easy and rapid implementation of its behavior in the virtual simulator. Such a layer will lay directly below the Application Layer [37] of the typical embedded-system software architecture [37].

This work is divided into the following sections. In Section 2, two didactic modules used for the teaching of embedded systems are shown. In Section 3, the common software architecture, along with what is proposed in the article as an extension for embedded-system programming, is shown. In Sections 4 and 5, the results, unit and functional tests, and discussion are presented, while Section 6 concludes the paper.

2. Materials

2.1. Didactic Modules for Embedded-System-Programming Teaching

At the Lodz University of Technology, in the Institute of Automatic Control [38], a few custom and dedicated microprocessor-embedded systems are used for didactic purposes. They are as simple as a legacy family of Intel MCS51s [39,40] and as complex as modern microcontrollers from Texas Instruments (TMS320F28379D in TMSLAB version 2), STMicroelectronics (STM32F429 in STMLAB module), or Broadcom (SOC BCM2836). All those modules were put together on a common didactical panel (Figure 1). The Institute of Automatic Control has 10 separate panels, like the one in Figure 1, allowing 10 student groups (mostly made up of two students) to work simultaneously in microprocessor techniques and embedded-systems software classes.

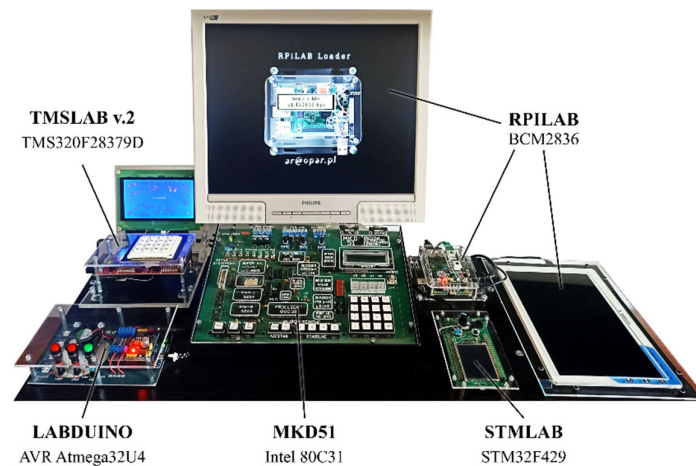


Figure 1. Laboratory stand for embedded microprocessor-system teaching at IAC of LUT [38].

The microprocessor systems shown in Figure 1 [41] are programmed by students during stationary courses (projects, laboratories), in order to become familiar with various programming methods, from Bare-Metal techniques [42] through simple operating systems, such as FreeRTOS, to reach Embedded Linux. There was also a strong need to make all the systems available for students not only during class time but outside of the laboratory too. To achieve this necessity, the virtual simulators of didactical systems were designed. They not only gave almost unlimited access time for practice with the modules but also were indispensable in the time of remote classes during the COVID-19 pandemic [33–35]. The microprocessor systems are programmed with high-level languages (C and C++) and also with the use of an assembler. For the two modules shown, TMSLAB and RPILAB (Figure 1), which were programmed with the use of Bare-Metal techniques [42], the virtual simulators are presented in this article.

2.2. TMSLAB Module

The TMSLAB module was designed mainly to implement discrete-time-control methods and mathematical models of dynamic objects. It was developed in two versions. Both implementations are used to perform control of a real object or to realize a discrete-time simulation of closed-loop-control systems, with the use of the TMS320C2000 MCU family. The TMSLAB in its second version (Figure 2a) is based on the TMS320F28379D MCU, which is a very advanced modern unit, with two CPUs and two CLAs (Control Law Accelerators—auxiliary control coprocessors). The first implementation of the TMSLAB was based on the TMS320F2812 (Figure 2b), which was designed for industrial-control applications but is obsolete these days. Both of the didactical modules were based on development boards: LaunchXL-F28379D and the eZdspF2812 Starter Kit. For each of them, a dedicated extension board was designed—an Interface Module for TMSLAB v.2 (Figure 2c) and an ACX Board for TMSLAB v.1 (Figure 2d). Both extensions allow to connect a matrix keyboard and an LCD as parts of the user-interface panel.

The most time-consuming programming tasks, on the TMSLAB module, are related to the so-called front-end [43], which is focusing on a complex Graphical User Interface (GUI) [44]. The back-end [45], which concerns object-control strategies and modeling algorithms, also requires a lot of programming work. Those two main software features should be simulated, giving more time to work without access to the hardware.

There is no commercial emulator or virtual simulator of the TMSLAB didactical module. This is obvious, since the TMSLAB is a custom microprocessor system. Such a simulator would be an incredibly complex piece of software, so its production time and costs would be incomparably higher than the TMSLAB module itself. Alternatively, a virtual simulator of TMSLAB can also be very simple and rapid in development, if it simulates only limited functionality—the most important in the programming process. Based on this assumption, the TMSLAB simulator should deliver only the components

needed for front-end programming using Bare-Metal techniques [42]. Properly designed back-end algorithms can easily reach multiplatform capability, so they do not need any special simulation mechanism.

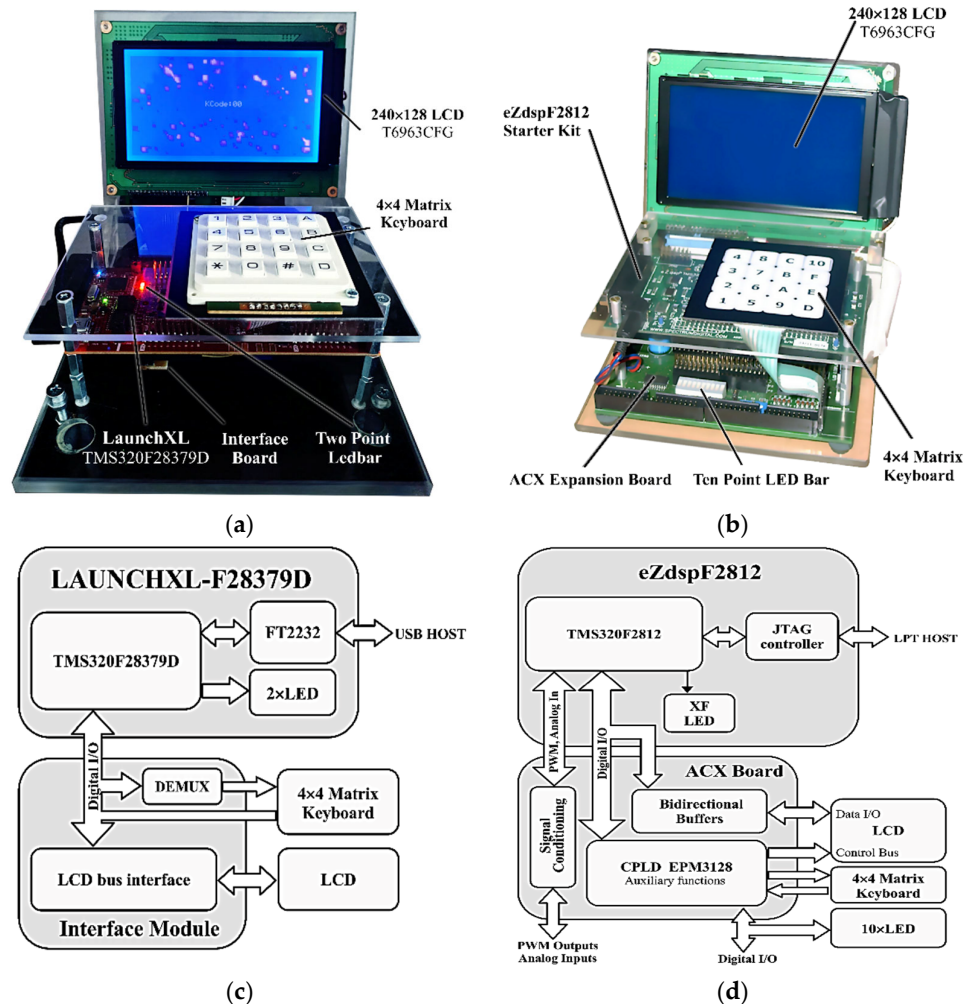


Figure 2. TMSLAB didactical module in two developed versions: (a) second version (the current one) based on TMS320F28379D MCU; (b) first version based on TMS320F2812 MCU; (c,d) their internal structure-block diagrams, respectively.

2.3. RPILAB Module

The second laboratory module is based on Raspberry Pi (version 2), a very popular microcomputer. Its main purpose, in the designated didactic process, is to be an embedded system for the entertainment-class device, which will be programmed with two opposite techniques. One is a classic Bare Metal and the second uses Embedded Linux [5] components. A designed RPILAB module was equipped with a UART communication extension and input controllers connected through a USB interface (Figure 3).

In the case of RPILAB Bare-Metal programming for an entertainment application, the user inputs and LCD output are the key peripherals. For virtual-simulation purposes of the RPILAB, this functionality is the primary need and its implementation through a simple API is easily reachable.

2.4. IDEs Used for Laboratory-Modules Programming

All programming tasks for both the TMSLAB and RPILAB modules use an Eclipse-CDT-derived Integrated Development Environment (IDE). This is a very popular IDE and most MCU vendors have adopted it to their needs. For the real TMSLAB device, the

Code Composer Studio IDE (CCS) is used. It adds some vital modules and views to the standard Eclipse and uses its standard configuration files. Switching the project nature and compiler-specific settings from CCS to Eclipse CDT (for virtual-simulation needs) requires only replacing the two configuration files, named *project* and *cproject*.

To form a common-programming platform for the RPILAB module and both of the virtual simulators, an Eclipse CDT was set with two different toolchains. The first one was for the real module with cross compilation to ARMv7-M architecture, and the second one was for virtual simulators with native compilation on an Intel x86 platform based on the MINGW project. A prepared IDE (Figure 4) was used to seamlessly develop software for all three platforms (real RPILAB and the two virtual simulators) with toolchain settings written in the Eclipse-configuration file.

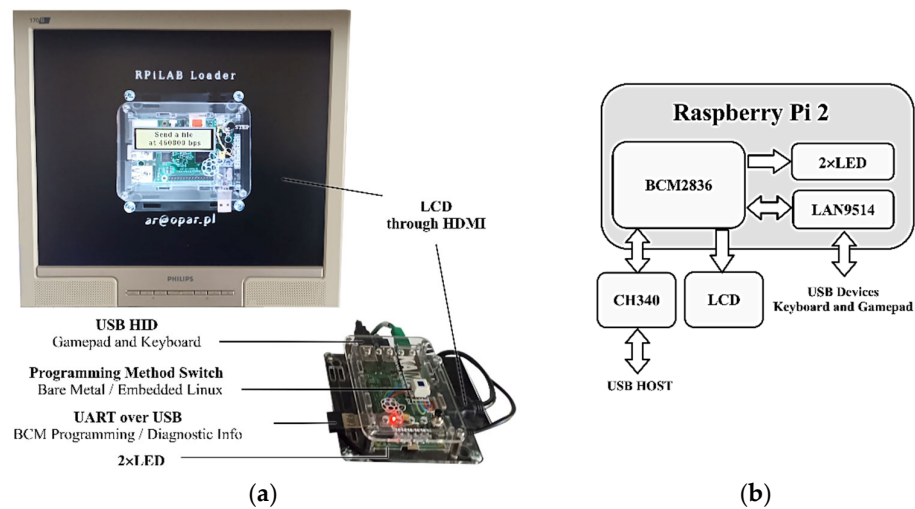


Figure 3. RPILAB module: (a) main components; (b) internal-structure block diagram.

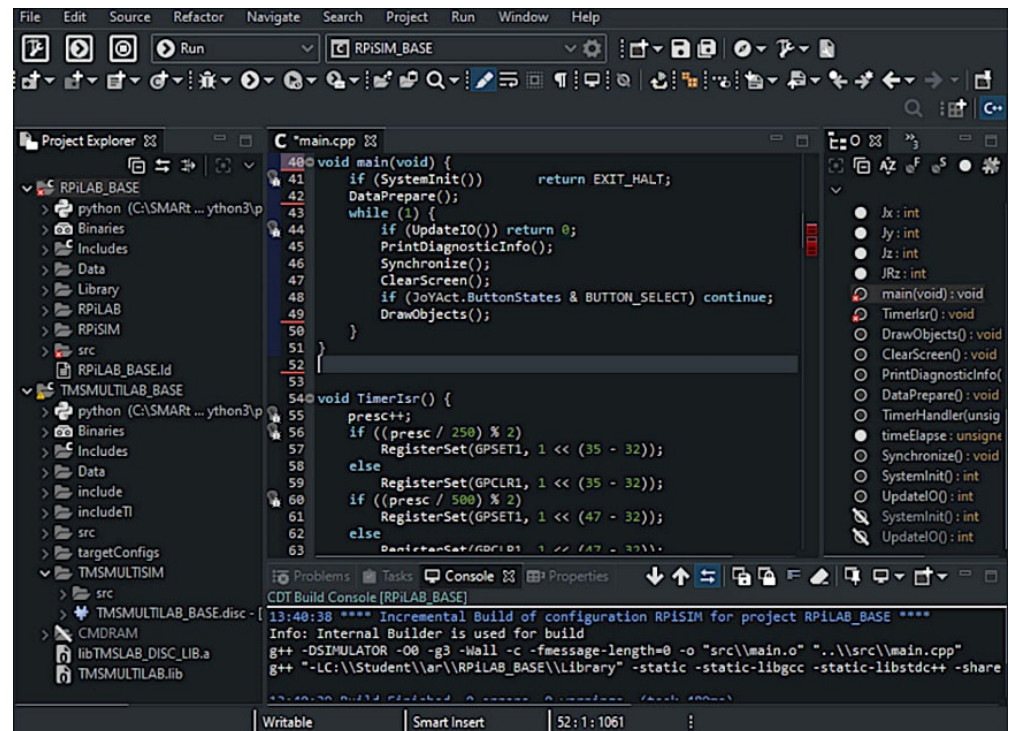


Figure 4. The Eclipse CDT IDE [46] used for programming real RPILAB and virtual simulators of RPILAB and TMSLAB.

The real didactical modules require programming of microprocessor-system memories. For TMSLAB, this is covered by Code Composer Studio, which uploads the whole output program, in ELF or COFF format, directly to the MCU memories. In the case of the real RPILAB module, a dedicated RPILAB Loader (Figure 5) was developed for downloading the program to a Raspberry Pi with 2 RAM memory, through the UART interface. The RPILAB Loader covers another functionality, by being the UART monitor, so it is used during RPILAB-program execution to print diagnostic information.

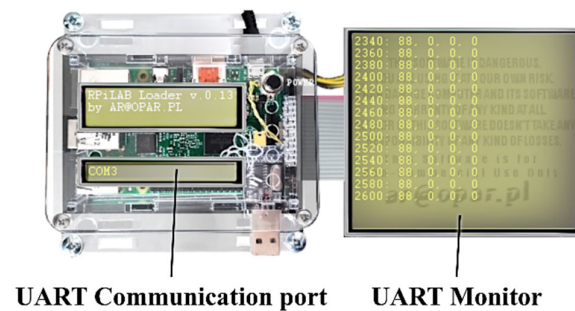


Figure 5. The RPILAB Loader—a dedicated programmer and monitor using UART communication.

3. Methods

3.1. Common Software Architecture for Embedded Systems Programming

There are many concepts of software development. They can consider different aspects of hardware platforms, operating systems, or specific frameworks. For embedded systems, few dedicated operating systems [47] and frameworks were designed [48]. Most of them use a typical software-layer architecture [1], which is shown in Figure 6. According to it, a prepared embedded application can run successfully on real or precisely emulated hardware platform without any additional efforts. For embedded-system-software development, those layers often overlap or omit each other to gain access to the critical hardware components or other layer resources.

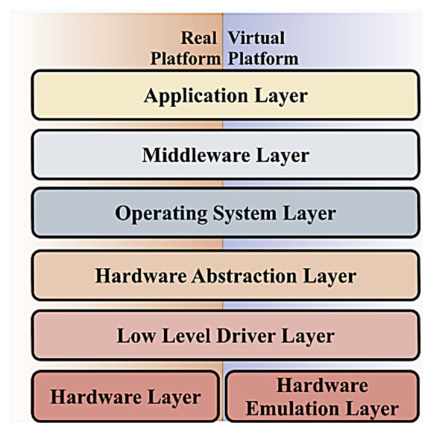


Figure 6. Typical architecture of embedded-system-software layers.

3.2. Software-Architecture Layer for Limited-System-Functionality Simulation

Virtual simulators are unique in many aspects. They do not try to emulate hardware on its low-level structure but rather imitate whole embedded-system functionality [2] using the available resources of the platform for which they were developed. The proposed virtual-simulator concept assumes that this functionality will be the most desirable and time-consuming during embedded-software development and is easy to implement for reaching rapid development of the simulator. This functionality can be further easily delivered to the Application Layer by introducing an additional layer that was called the Simulation-Oriented Layer (SOL), and shown in Figure 7.

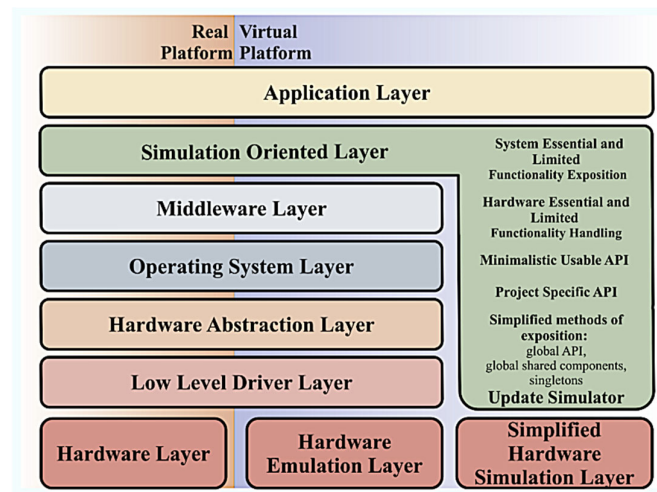


Figure 7. Proposed Simulation-Oriented Layer as a top-level API.

The SOL's purpose is to deliver to the Application Layer only limited and highly desirable microprocessor-system functionality, which would be implemented in a virtual simulator without requiring its complex development. The SOL for a virtual simulator is a direct connection between an embedded application and the simulation mechanisms. In case of running application in real hardware, the SOL will use the standard layers' API for the embedded systems. This approach is in opposite to the actual existing software layers, which are focused to be versatile and expose as large a functionality as possible [30].

There are numerous versatile components and layers that are close-coupled with hardware (e.g., HAL, UEFI, BIOS, Device Tree, or drivers). There are whole frameworks for embedded systems such as CMSIS. All of those solutions have their place or define a complex system-software architecture. Their API is not suitable for rapid virtual-simulator development. It is obvious because they are focused on a maximum of hardware efficiency and capabilities. For a simplified virtual simulator, a simplified API, suitable for rapid simulator development, is needed. This missing role will fulfill the SOL.

A designed custom-embedded system, through its architecture and software, limits itself intentionally in many aspects. It plays its role inside the device to meet all of its requirements: not less and not more, just as is needed. For example, its user interface uses limited peripherals (e.g., LCD, keyboard), communication is based only on required interfaces (e.g., UART), and program-execution timing and memory footprints need to fit chosen MCU capabilities. This leads to the conclusion that the virtual simulator for custom embedded systems needs to implement only this limited functionality, and SOL is for delivering it to the Application Layer. Such an assumption dramatically reduces virtual-simulator complexity and gives the chance to develop it rapidly.

3.3. Simulation-Oriented Layer API Concepts

As proposed in the article concept, most of the SOL API components are implemented directly in a virtual simulator. In case of a building project for a real device, the SOL uses an existing software stack available for real hardware. The SOL needs to be a multiplatform layer and allow the embedded software to be portable for the virtual simulator in as easy a way as possible. It should deliver to the Application Layer only the common parts for both real and virtual solutions. The SOL API can implement its functionality through one of the typical methods:

- global shared data, accessible across a virtual simulator, embedded-software API, and embedded application;
- global functions of system functionality—a simple API to get access to simulated hardware functionality (e.g., *SetLed*, *GetKeyCode*, etc.);

- global handlers—functions of specialized purpose (e.g., Interrupt Subroutine Handler) that will be called by lower embedded-system layers and the virtual simulator;
- singletons for hardware handling—composition of both the above concepts, in terms of the C++ language, to gather resources in a single instance of a hardware-handling class;
- global simulator update function—a function that is an entry point to the virtual simulator where all its data will be processed; such a function could be also called a hidden one, inside of a global SOL API.

All the above API components will have the same definition for real and simulated platforms but with different implementations. They will behave like typical multiplatform-programming resources. For a real device, its implementation will require to add some extra intermediate code in the SOL, but for a virtual simulator it will be only a pointer to a function or data defined inside the virtual simulator software.

3.4. Simulation-Oriented Layer for TMSLAB and RPILAB Programmed Using Bare-Metal Techniques

Based on the above considerations and the internal hardware structure of the TMSLAB, its virtual simulator for rapid development should simulate only three user peripherals: LCD, KEYBOARD and LEDBAR. All of them could have support in the shape of three singletons with the same names. Since SOL needs to be consistent with both platforms (real and simulated), support for hardware initialization and CPU Timer 2 Handler is also needed. The Timer is used to implement algorithms that require constant time period of execution. In case of TMSLAB didactical software-project development, it is required either for automatic-control algorithms or for the modeling dynamics of control-object-algorithm implementations. All SOL API components of the TMSLAB programming with Bare-Metal techniques are depicted in Figure 8.

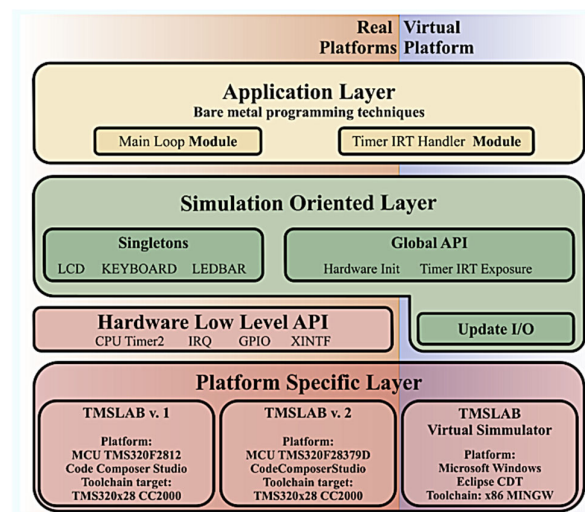


Figure 8. Software-architecture layers for application development using Bare-Metal techniques on TMSLAB module and its virtual simulator.

The Timer Interrupt Handler is simulated through fixed function name that application should define according to its prototype:

```
void Timer2Isr(void);
```

The timer resolution used for the simulation of periodic subroutines on Microsoft Windows is one millisecond, with the minimum possible value also being one millisecond. The timer usage in the simulator, and its resolution is not directly related to the hardware timer. Its main purpose is to deliver the simulation of a constant period-interrupt subroutine called the interrupt subsystem. The timer IRQ period is controlled in the embedded software (for both the real and virtual-simulator platforms) by a global variable and cannot

be changed during the execution of the application (it is used in the software-initialization part only).

An entry point to the virtual simulator is given as a pointer to the UpdateIO function, with a prototype:

```
extern int (*UpdateIO)(void);
```

Calling this function will update the whole virtual-simulator content and the data structures that can be used by the embedded application. When the user will force-close a virtual simulator, this function returns a non-zero value and the embedded software should exit. This is a non-standard behavior because a typical embedded application will never end.

The SOL proposed for RPILAB module is a little different and is presented in Figure 9.

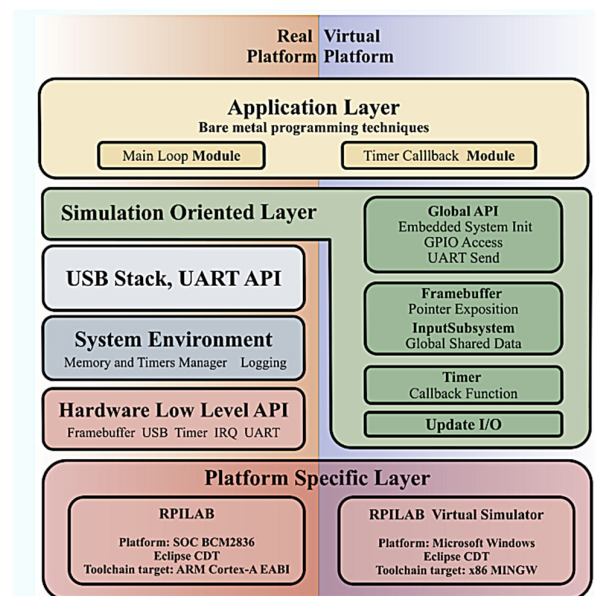


Figure 9. Software-architecture layers for application developing using Bare-Metal techniques on RPILAB module and its virtual simulator.

The input/output subsystem of user-interface peripherals is based on the global shared data that cover video-framebuffer functionality and feed global data structures with pressed keys. It has an API for simplified GPIO and UART utilization. Periodic-procedure calling, which is important in embedded systems, is like the TMSLAB case and is based on a global handler. In this case, a handler lacks an interrupt implementation, since the underlying software layers for a real device require to register this procedure as a call-back.

3.5. Virtual Simulator as a Rapidly Developed Stand-Alone Application

There are a few possibilities to develop a virtual simulator for the SOL API implementation. One could include the GNU Debugger (GDB)-server implementation that would be under the full control of the IDE running the GDB client. This is a typical way for how emulators work. Developing a virtual simulator using this approach would be complex and time-consuming task. The simpler and more rapid way to develop a virtual-simulator approach would be to assume that the embedded software is compiled as a dynamically linked library and runs directly inside a virtual simulator. This would behave like a plugin system with an entry point for the main function from the embedded software (Figure 10). This approach requires to implement only few simulator components related to the SOL that should be simple and reduced according to its definition.

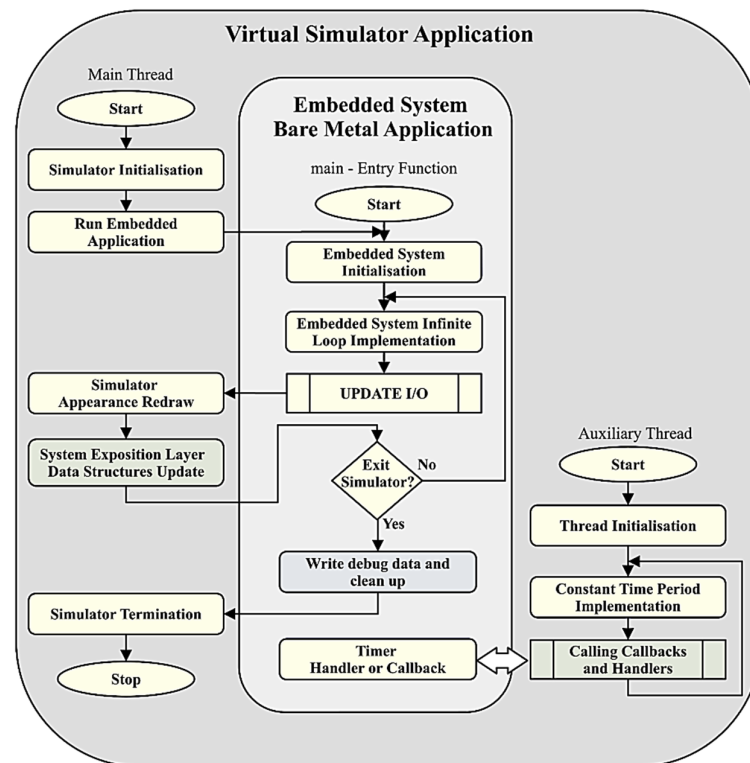


Figure 10. Rapidly developed stand-alone virtual-simulator application architecture.

4. Results

4.1. Virtual-Simulator Implementation on Microsoft Windows

Microsoft Windows is assumed to be the most user-friendly operating system. To reach desirable popularity among students, the choice for the virtual-simulator platform could not be different. The Win32 API delivers simple and efficient low-level functions to draw bitmaps with transparency directly on the application canvas. This feature was used to prepare real-looking user interfaces of the simulators, which were basically composed from real-device photographs.

Virtual-simulator software architecture implements an embedded system-timer functionality using Windows' high-resolution timers. This can lead to some issues, since Microsoft Windows is not the RTOS and will not guarantee to meet the timing requirements.

From the didactic point of view, the features of the microprocessor system related to the User Interface are the most attractive for students. The simulation of the devices, such as buttons, keyboards, LEDs, or LCD displays, gives the feeling of working with a very precise and realistic virtual device. Our observations show that this causes a lot of enthusiasm in developing software projects created in simulation conditions and encourages students to take on activities outside the classroom.

This is the main reason for the limited hardware support of SOL, which is focused mainly on embedded-system User-Interface peripherals.

The ability to run embedded an application as a dynamically linked library is the key for the rapid development of virtual simulators. Both requirements (limited functionality and simple plugin as a dynamic library) can be easily reached in a Microsoft Windows user-oriented environment.

Due to using simulator architecture, as shown in the previous chapter, embedded-software source codes need to be compiled using toolchains for the same platform as the simulator. This can lead to some issues that were considered in the discussion chapter.

4.2. TMSLAB Virtual Simulator

As a result of the proposed SOL-implementation methods and the simulator-application architecture, a TMSLAB virtual simulator was developed for Microsoft Windows platform. It imitates the behavior of the two layers of the LCD (graphical and textual), matrix keyboard, and 10-points LED bar, through the implementation of three singletons named: LCD, KEYBOARD, and LEDBAR. The SOL for the LCD additionally contains two memory areas that act as buffers for one-bit-depth graphics and ASCII-coded text layers. All features of the SOL for the TMSLAB were used in the Demo Application and are shown in Figure 11.

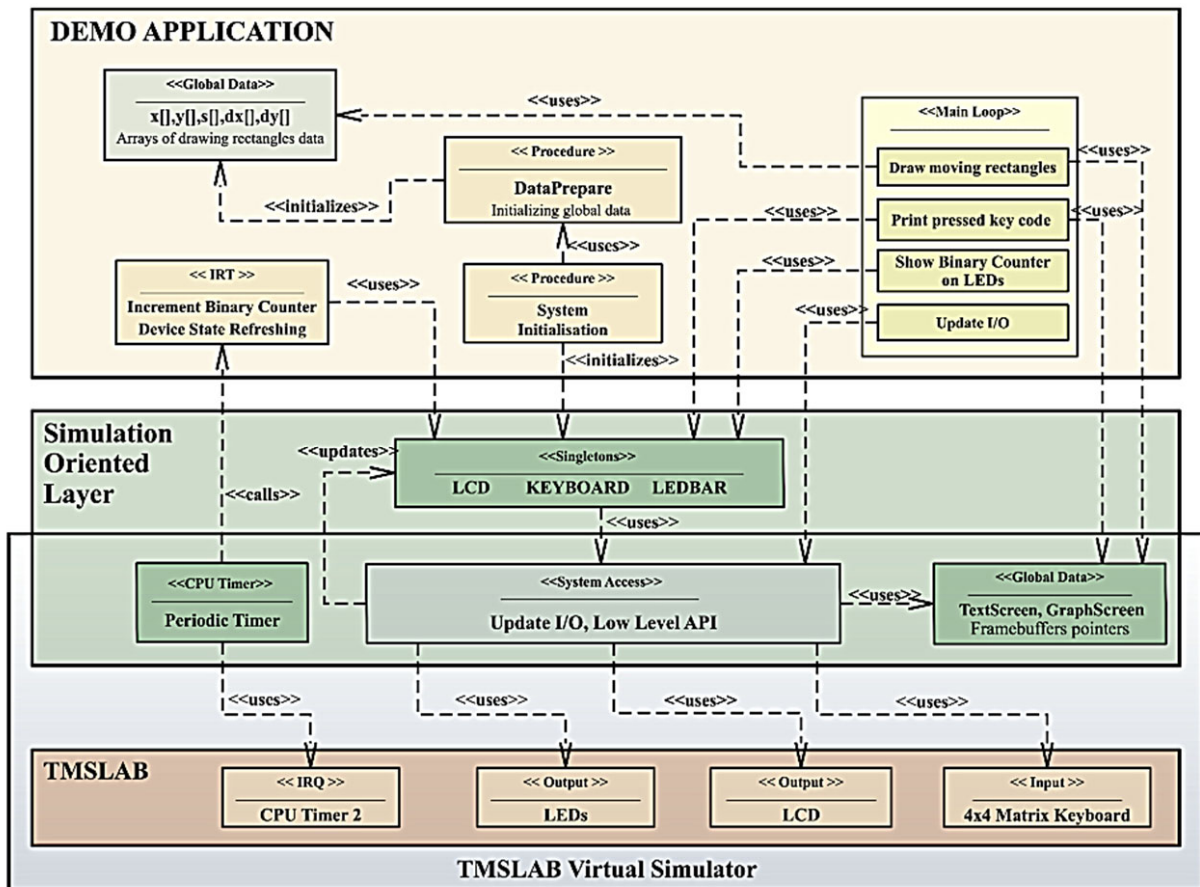


Figure 11. Demo-software-architecture diagram for real TMSLAB and its virtual simulator.

The Demo Application draws objects in a graphical layer, prints string in the text layer, reads input from keyboard, and controls LEDs, all by utilizing the SOL layer. The squares visualized on the LCD move at different velocity vectors, with constant component controlled by the keyboard, and are dynamically resized to perform a complex test of the whole embedded system. This application can be run both on a real target and in a virtual simulator (Figure 12).

The designed simulator application allows for running embedded applications by dropping them on the simulator area or through the application’s command line. Its user interface introduces a few graphical buttons, allowing for switching to full screen, exiting the simulator, or performing a simplified stepping-run mode on every SOL UpdateIO function calling. The visual look of the virtual simulator is exactly the same as for the real TMSLAB designed in version 1. It might seem that this is a minor detail, but it makes students feel that they work with a very precise virtual-hardware representation, so they use it with great enthusiasm.

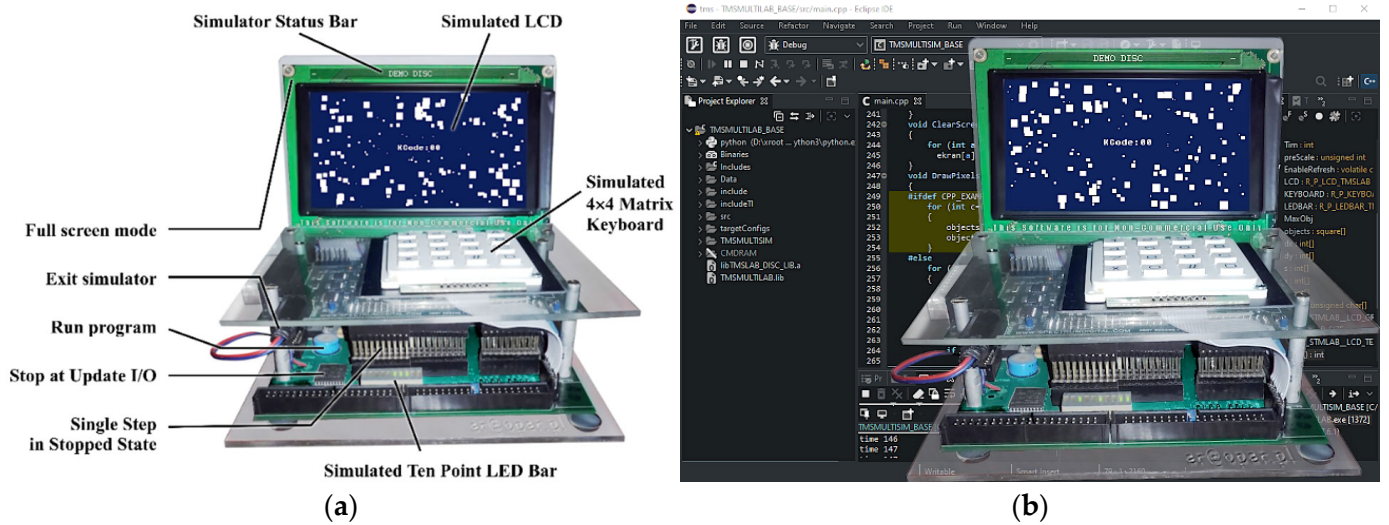


Figure 12. Virtual simulator of the TMSLAB didactical module running a demo program: (a) simulator components and GUI; (b) simulator running on Microsoft Windows platform under Eclipse IDE control.

4.3. RPILAB Virtual Simulator

In a similar way as for TMSLAB, the RPILAB virtual simulator was developed for the Microsoft Windows platform. It imitates the basic behavior of Video Framebuffer, Keyboard, Gamepad, UART, and two LEDs. The functionality is accessible through the global shared data and a dedicated API in the SOL. All those features were used in the Demo Application and are shown in Figure 13.

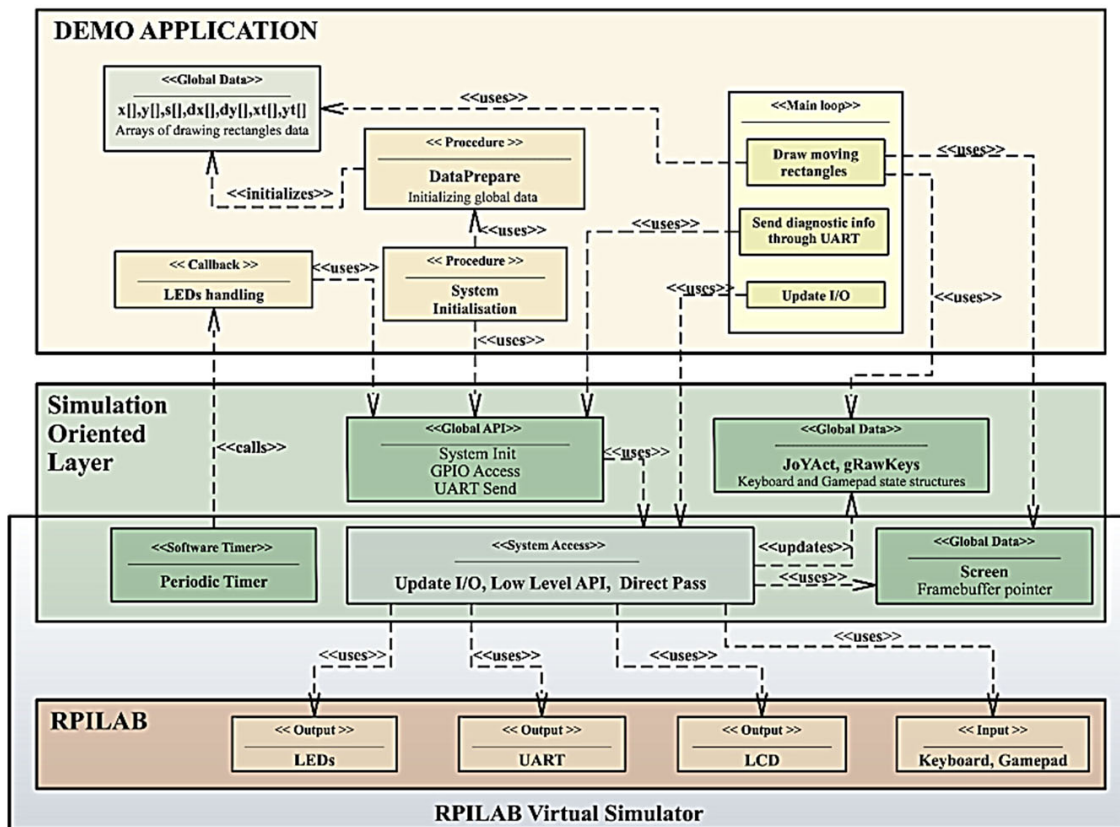


Figure 13. Demo-software-architecture diagram for real RPILAB and its virtual simulator.

The RPI virtual simulator running the Demo Application is shown in Figure 14. This application draws tens of thousands of small squares that move at different velocity vectors, with a few components controlled by the keyboard and gamepad. They can be accelerated and rotated, plus their starting point can be moved. The overall dynamical effect is used to perform a functional system-level test. The internal architecture of the RPILAB virtual simulator and its user interface is almost the same as for the TMSLAB. This allowed for reusing most of the TMSLAB virtual-simulator source code. An additional functionality for the RPILAB module was required, due to sending the debug information through the UART interface. The SOL in this case includes one API function to send a whole string using UART. For simulation purposes, its implementation just prints the string in a dedicated virtual-simulator window, as shown in Figure 14, as a “UART Monitor”.

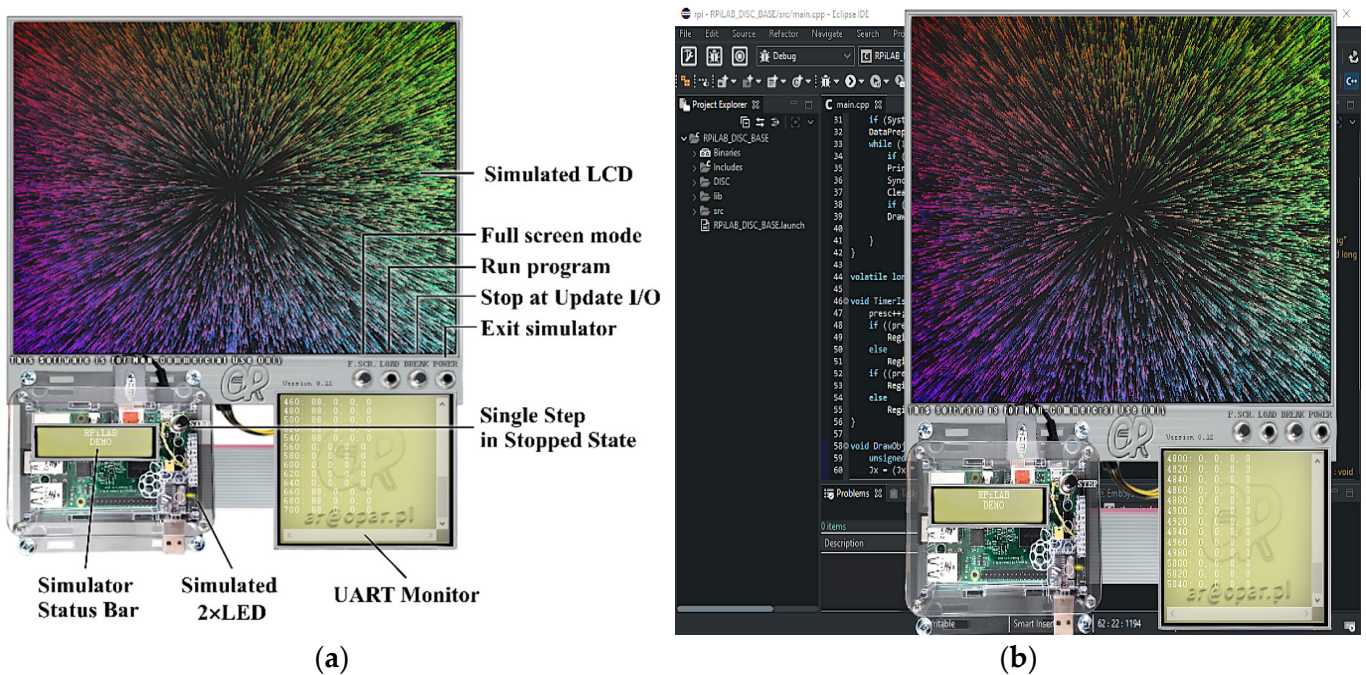


Figure 14. Virtual simulator of the RPILAB didactical module running demo program: (a) simulator components and GUI; (b) simulator running on Microsoft Windows platform under Eclipse IDE control.

4.4. Simulators Tests Results

The two developed virtual simulators are running under the Microsoft Windows OS with embedded software compiled for Intel x86 architecture. For this reason, a comparison of performance, memory footprints, or timing relations of both platforms would be unreasonable and unnecessary, bearing in mind the purpose of the presented simulators (progressing development of embedded-application functionality, without access to a real device). The simulators were tested according to the system functionality implemented in the SOL. The sample test results for the TMSLAB module are shown in Figure 15a–d (for memory-visualization-unit tests) and in Figure 15e,f (for functional tests based on the demonstration program shown in Figure 11).

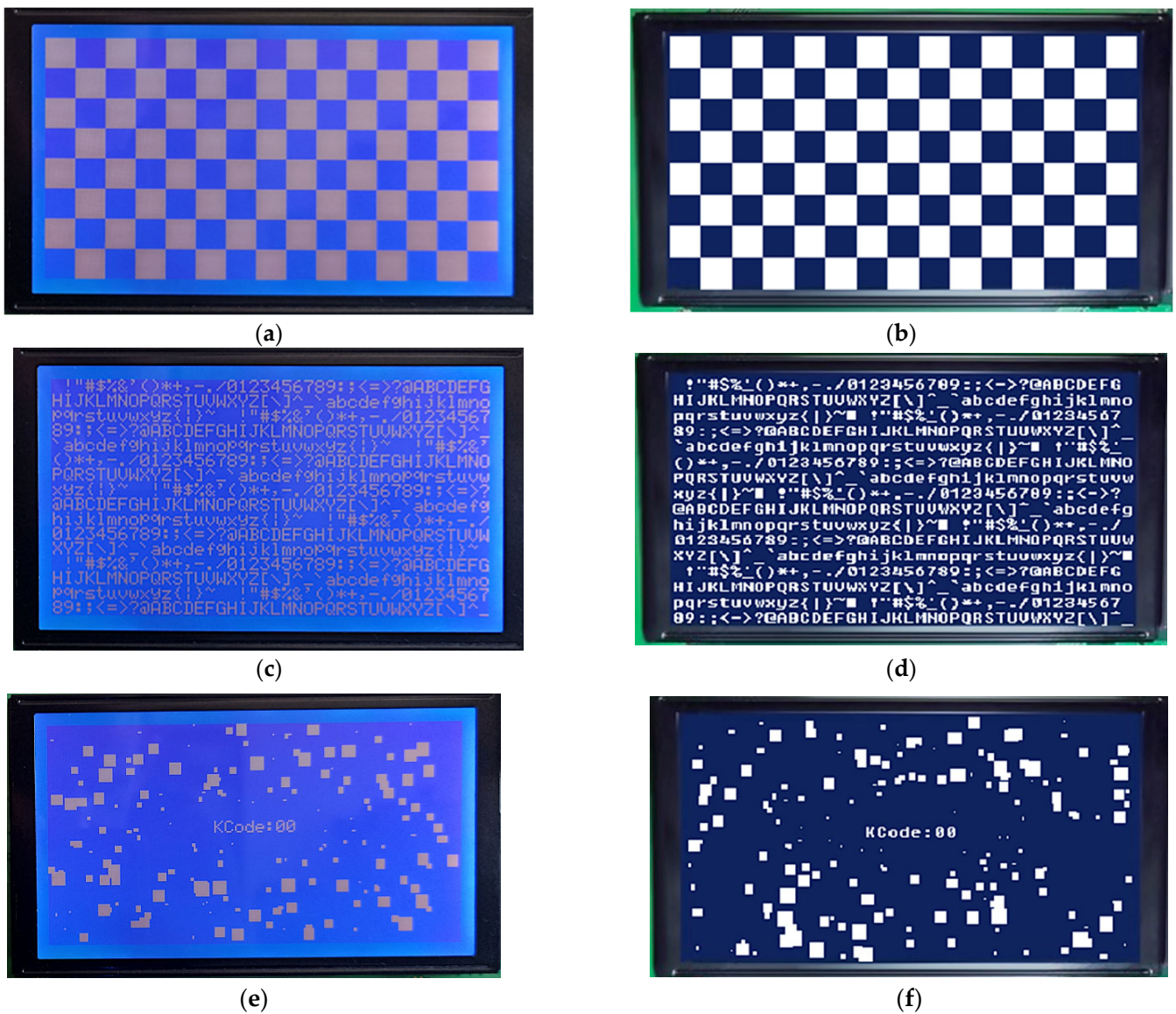


Figure 15. Results of unit tests (a–d) and functional tests (e,f) performed using TMSLAB real device (a,c,e) and its virtual simulator (b,d,f): (a) photograph of real TMSLAB graphic-layer test; (b) screenshot of virtual-simulator graphic-layer test; (c) photograph of real TMSLAB textual-layer test; (d) screenshot of virtual-simulator textual-layer test; (e) photograph of real TMSLAB system-level test; (f) screenshot of virtual-simulator system-level test.

Similar tests were performed for the RPILAB module. In Figure 16a,b, the unit tests of the framebuffer are shown, and in Figure 16c,d the functional tests based on the demonstration program (Figure 13) are included.

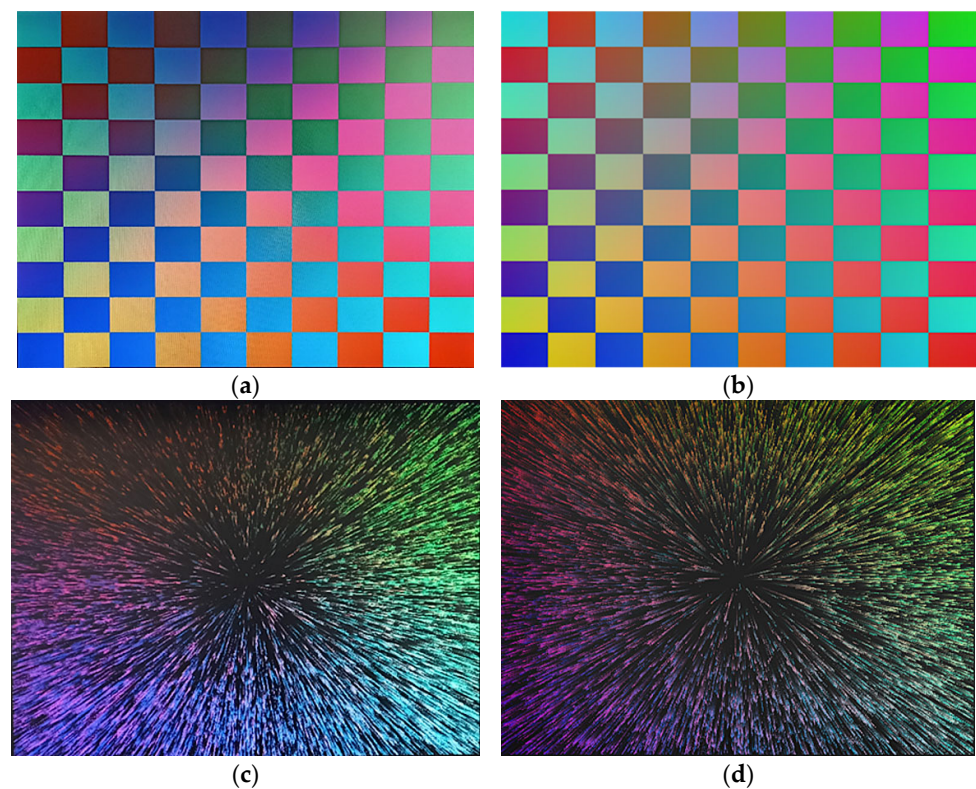


Figure 16. Results of unit tests (a,b) and functional tests (c,d) performed using RPILAB real device (a,c) and its virtual simulator (b,d): (a) photograph of real LCD during framebuffer test; (b) screenshot of virtual-simulator framebuffer test; (c) photograph of real LCD for functional test using demo application; (d) screenshot of virtual-simulator functional test using demo application.

5. Discussion

Proposed virtual-simulator architecture, with a limited but most desirable simulation capability and running an embedded application as a part of the virtual simulator, is a straight way to rapid development of virtual simulator for custom embedded system. The proposed solution enables remote work on most time-consuming embedded-software parts implementation. Prepared source codes for the simulator are ready to be compiled and run on a real embedded system platform. This is possible because the real and virtual platforms use the SOL, which is reduced and has made it easy to implement and simulate embedded-system components. Despite the high compatibility of a properly designed SOL with both platforms (real and virtual), there can be some issues that will be related to:

- basic data types that can vary for used platform compilers: a safe approach should consider using types defined in *stdint* header;
- memory layout, which is typically one continuous space for simulator and for embedded system can be fragmented among many blocks; this issue can be solved using a proper linker script;
- timings of peripherals operation and program execution in the target platform; this will be different on both platforms, so properly designed software cannot rely on this behavior directly in the Application Layer;
- a lack of common synchronization mechanism for both platforms, between the main program and timer handlers, so for simplicity and rapid development of the simulator, there is no synchronization provided, and to avoid inconsistent access to shared resources, they should be accessed atomically;
- a risk of using libraries on a virtual simulator platform that are not supported in the embedded system, which can be a serious mistake and force a redesign of software developed purely in the virtual simulator;

- Non-typical behavior of an embedded application running in a virtual simulator that is forced to close, so this should be covered by the software to make a clean exit from an application that will be unnecessary in a real device;
- using Microsoft Windows as a platform for a virtual simulator can lead to potential issues with constant time-period requirements for embedded-timer implementation; this is caused by the non-deterministic nature of the Microsoft Windows operation system.

Despite the limitations listed above the properly designed applications work well using the virtual simulators that were proven in Figures 15 and 16. The results of unit (static) and functional (dynamic) tests show high compatibility between the virtual and real embedded systems for TMSLAB and RPILAB cases using chosen peripherals such as an LCD, keyboard, led bar, gamepad, and video framebuffer.

To accelerate the process of application development for students, special demo programs were created. They are shown in Figures 12 and 14. The purpose of this programs is the functional demonstration of the usage of Simulated-Oriented Layer classes and limitations. Besides this, each laboratory class is accompanied by a lecture on which important theoretical issues and SOL limitations are discussed.

Taking into account the limitations listed above, we have to emphasize that the proposed approach in the paper was intentionally simplified.

6. Conclusions

The Simulation-Oriented Layer and its usage in stand-alone virtual simulators, to run embedded-system applications, was shown in this article. The disclosed internal architecture of virtual simulators, with a plugin-like system and essential SOL-functionality implementation, allowed for the rapid development of the simulators. The results shown include two didactical microprocessor-system virtual simulators, which were used successfully for embedded-software development using Bare-Metal techniques. During the COVID-19 pandemic, this allowed for an undisturbed didactical process.

The presented virtual simulators do not tend to be a full replacement for the hardware platforms. They present simple solutions, and their purpose is to simulate the most time-consuming parts of software development, giving students enough time to complete their tasks, even without access to physical devices. The proposed solution can have a limited application in product development intended for the market. The choice given to the developers (students) is: accept a simplified and limited simulator and be able to advance a software project (even in a limited way) without access to the hardware, or reject it and do not use any simulator because there is no virtual simulator for the particular embedded system.

It is believed that many readers would find the solution of SOL useful for their own custom embedded system, which could be virtualized in the same way.

Author Contributions: Conceptualization, A.R.; methodology, A.R.; software, A.R.; validation, A.R.; formal analysis, A.R. and T.R.; investigation, A.R.; resources, T.R.; data curation, A.R.; writing—original draft preparation, A.R.; writing—review and editing, T.R.; visualization, A.R.; supervision, A.R. and T.R.; funding acquisition, T.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. *Real-Time Embedded Systems*; MDPI: Basel, Switzerland, 2018; ISBN 978-3-03897-510-6.
2. De Micco, L.; Vargas, F.L.; Fierens, P.I. A Literature Review on Embedded Systems. *IEEE Lat. Am. Trans.* **2020**, *18*, 188–205. [[CrossRef](#)]
3. Edwards, S.; Lavagno, L.; Lee, E.A.; Sangiovanni-Vincentelli, A. Design of embedded systems: Formal models, validation, and synthesis. *Proc. IEEE* **1997**, *85*, 366–389. [[CrossRef](#)]
4. Morison, G.; Barrie, P. An integrated approach to teaching embedded systems early in the curriculum—Flip flops to PONG. In Proceedings of the EDERC 2012 5th European DSP Education and Research Conference, Amsterdam, The Netherlands, 13–14 September 2012; pp. 169–172. [[CrossRef](#)]
5. McLoughlin, I.; Aendenroomer, A. Linux as a teaching aid for embedded systems. In Proceedings of the 2007 International Conference on Parallel and Distributed Systems—ICPADS, Hsinchu, Taiwan, 5–7 December 2007; Volume 2. [[CrossRef](#)]
6. Hollstein, T.; Reinsalu, U.; Leier, M. Motivation-driven learning processes at the example of embedded systems. In Proceedings of the EWME 2014: The 10th European Workshop on Microelectronics Education, Tallinn, Estonia, 14–16 May 2014; pp. 3–6. [[CrossRef](#)]
7. Sharad, S.; Muecke, K. Teaching complete embedded systems design process with graphical system design methodologies. In Proceedings of the International Conference Parallel and Distributed Systems—ICPADS, Hsinchu, Taiwan, 5–7 December 2007; Volume 2, pp. 9–14. [[CrossRef](#)]
8. Yang, K.C.; Chang, Y.T.; Wu, C.M.; Huang, C.M.; Luo, H.H. Application-oriented teaching of embedded systems. In Proceedings of the 2011 IEEE International Conference on Microelectronic Systems Education (MSE), San Diego, CA, USA, 5–6 June 2011; pp. 118–121. [[CrossRef](#)]
9. Liang, R. Practical Teaching Reform of ARM Embedded Technology Based on OBE Concept for Applied Undergraduates. In Proceedings of the 2021 2nd International Conference on Computers, Information Processing and Advanced Education (CIPAE 2021), Ottawa, ON, Canada, 25–27 May 2021; pp. 90–92. [[CrossRef](#)]
10. Rankovska, V. Innovative Approaches in Teaching Embedded Systems. The Smart Home Project. In Proceedings of the 12th National Conference with International Participation (ELECTRONICA), Sofia, Bulgaria, 27–28 May 2021; pp. 22–25. [[CrossRef](#)]
11. Schuster, H.; Wenzl, M.; Zauner, M. A framework for teaching embedded multi-core programming. In Proceedings of the 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Suzhou, China, 8–10 July 2012; pp. 292–297. [[CrossRef](#)]
12. Ping, W. Research on the embedded system teaching. In Proceedings of the 2008 International Workshop on Geoscience and Remote Sensing (Ett and Grs), Shanghai, China, 21–22 December 2008; Volume 1, pp. 19–21. [[CrossRef](#)]
13. Fan, H.; Wu, X.; Ghannam, R.; Feng, Q.; Heidari, H.; Imran, M.A. Teaching Embedded Systems for Energy Harvesting Applications: A Comparison of Teaching Methods Adopted in UESTC and KTH. *IEEE Access* **2020**, *8*, 50780–50791. [[CrossRef](#)]
14. Ibrahim, I.; Ali, R.; Adam, M.Z.; Elfidel, N. Embedded systems teaching approaches & challenges. In Proceedings of the 2014 IEEE 6th Conference on Engineering Education, Kuala Lumpur, Malaysia, 9–10 December 2014; pp. 34–39. [[CrossRef](#)]
15. Werner, S.; Lauber, A.; Koedam, M.; Becker, J.; Sax, E.; Goossens, K. Cloud-based design and virtual prototyping environment for embedded systems. *Int. J. Online Eng.* **2016**, *12*, 52–60. [[CrossRef](#)]
16. Silva, E.T.; Barcelos, D.; Wagner, F.R.; Pereira, C.E. A virtual platform for multiprocessor real-time embedded systems. In Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems, Madrid, Spain, 23–25 September 2009; pp. 31–37. [[CrossRef](#)]
17. Ghosh, A.; Bershteyn, M.; Casley, R.; Chien, C.; Jain, A.; Lipsie, M.; Tarrodaychik, D.; Yamamoto, O. Hardware-software co-simulator for embedded system design and debugging. In Proceedings of the Asia and South Pacific Design Automation Conference ASP-DAC, Chiba, Japan, 29 August–1 September 1995; pp. 155–164. [[CrossRef](#)]
18. Cho, S.-Y. A virtual simulation package for Embedded System training and education. In Proceedings of the 2009 International Conference on Engineering Education (ICEED), Kuala Lumpur, Malaysia, 7–8 December 2009; pp. 72–76.
19. Scherp, A. Software development process model and methodology for virtual laboratories. *Appl. Inform. Proc.* **2002**. Available online: <http://ansgarscherp.net/publications/pdf/C01-Scherp-SoftwareDevelopmentProcessModelAndMethodologyForVirtualLaboratories.pdf> (accessed on 20 April 2022).
20. Beghi, A.; Marcuzzi, F.; Martin, P.; Tinazzi, F.; Zigliotto, M. Virtual prototyping of embedded control software in mechatronic systems: A case study. *Mechatronics* **2017**, *43*, 99–111. [[CrossRef](#)]
21. Xie, W.; Yang, X.; Li, F. A virtual laboratory platform and simulation software based on web. In Proceedings of the 2008 10th International Conference on Control, Automation, Robotics and Vision, Hanoi, Vietnam, 17–20 December 2008; pp. 1650–1654.
22. Engblom, J. On Hardware and Hardware Models for Embedded Real-Time Systems. In *IEEE Workshop on Real-Time Embedded Systems*; IEEE: New York, NY, USA, 2001. Available online: <https://www.engbloms.se/publications/engblom-wrtes2001.pdf> (accessed on 20 April 2022).
23. Han, A.H.; Hwang, Y.S.; An, Y.H.; Lee, S.J.; Chung, K.S. Virtual ARM platform for embedded system developers. In Proceedings of the ICALIP 2008—2008 International Conference on Audio, Language and Image Processing, Shanghai, China, 7–9 July 2008; pp. 586–592. [[CrossRef](#)]

24. Yuan, H.; Yao, Y.; He, P. An Emulation and Context Reconstruction Tool for Embedded High-Precision Positioning System. In Proceedings of the 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Daegu, Korea, 17–19 August 2016; p. 107.
25. Petrot, F.; Hommais, D.; Greiner, A. Cycle precise core based hardware/software system simulation with predictable event propagation. In Proceedings of the EUROMICRO 97, 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No.97TB100167), Budapest, Hungary, 1–4 September 1997; pp. 182–187.
26. Helali Moghadam, M.; Saadatmand, M.; Borg, M.; Bohlin, M.; Lisper, B. Learning-Based Response Time Analysis in Real-Time Embedded Systems: A Simulation-Based Approach. In Proceedings of the 2018 IEEE/ACM 1st International Workshop on Software Qualities and Their Dependencies (SQUADE), Gothenburg, Sweden, 27 May–3 June 2018; pp. 21–24.
27. Helmstetter, C.; Joloboff, V.; Xiao, H. SimSoC: A full system simulation software for embedded systems. In Proceedings of the 2009 IEEE International Workshop on Open-Source Software for Scientific Computation (OSSC), Guiyang, China, 18–20 September 2009; pp. 49–55. [CrossRef]
28. Ishikawa, M.; McCune, D.J.; Saikalis, G.; Oho, S. CPU Model-Based Hardware/Software Co-design, Co-simulation and Analysis Technology for Real-Time Embedded Control Systems. In Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07), Bellevue, WA, USA, 3–6 April 2007; pp. 3–11.
29. Moon, T.-Y.; Seo, S.-H.; Kim, J.-H.; Hwang, S.-H.; Jeon, J.W. Simulation with consideration of hardware characteristics and auto-generated code using matlab/simulink. In Proceedings of the 2007 International Conference on Control, Automation and Systems, Seoul, Korea, 17–20 October 2007; pp. 1494–1498.
30. Saboori, E.; Abdi, S. Hybrid Prototyping of Multicore Embedded Systems. In Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; pp. 1627–1630.
31. Muttenthaler, F.; Wilker, S.; Sauter, T. Lean automated hardware/software integration test strategy for embedded systems. In Proceedings of the 2021 22nd IEEE International Conference on Industrial Technology (ICIT), Valencia, Spain, 10–12 March 2021; pp. 783–788.
32. Nooshabadi, S.; Garside, J. Teaching embedded systems design—An international collaborative project. In Proceedings of the Frontiers in Education 35th Annual Conference, Indianapolis, IN, USA, 19–22 October 2005; Volume 2015, pp. 254–262. [CrossRef]
33. Rodriguez-Segura, L.; Zamora-Antuñano, M.A.; Rodriguez-Resendiz, J.; Paredes-García, W.J.; Altamirano-Corro, J.A.; Cruz-Pérez, M.Á. Teaching Challenges in COVID-19 Scenery: Teams Platform-Based Student Satisfaction Approach. *Sustainability* **2020**, *12*, 7514. [CrossRef]
34. Khant, S.; Patel, A. COVID19 remote engineering education: Learning of an embedded system with practical perspective. In Proceedings of the 2021 International Conference on Innovative Practices in Technology and Management (ICIPTM 2021), Noida, India, 17–19 February 2021; pp. 15–19. [CrossRef]
35. Shoufan, A. Active Distance Learning of Embedded Systems. *IEEE Access* **2021**, *9*, 41104–41122. [CrossRef]
36. Radecki, A.; Rybicki, T. An Accurate State Visualization of Multiplexed and PWM Fed Peripherals in the Virtual Simulators of Embedded Systems. *Appl. Sci.* **2022**, *12*, 3137. [CrossRef]
37. Binns, P.; Vestal, S. *Formalizing Software Architectures for Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 451–468, ISBN 978-3-540-42673-8.
38. Institute of Automatic Control, Lodz University of Technology. Available online: <https://www.automatyka.p.lodz.pl/?lang=en> (accessed on 19 February 2022).
39. Fei, J.; Quan, H.; Yuan, D. Analysis and Comparison of Two Different Implementations of MCS-51 Compatible Microcontrollers. In Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, Sydney, NSW, Australia, 12–14 December 2011; pp. 275–278.
40. Mroczek, H. *Microprocessor Technique*; Lodz University of Technology Press: Łódź, Poland, 2007; ISBN 978-83-7283-238-2. (In Polish)
41. Microprocessor Systems Laboratory. Available online: <http://ztchs.p.lodz.pl/index.php?www=SM> (accessed on 20 February 2022).
42. Lal, S. Bare-Metal Systems. In *Real World Multicore Embedded Systems*; Elsevier: Amsterdam, The Netherlands, 2013; pp. 517–560.
43. Tang, S.; Chan, W.C.; Vai, M.I.; Mak, P.U. A Front-end Platform of the Network-based Intelligent Home Healthcare Embedded System. In Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, San Francisco, CA, USA, 1–5 September 2004; Volume 4, pp. 3116–3119.
44. Burhan, I.; Othman, R.; Azman, A.A. Development of electro pneumatic trainer embedded with Programmable Integrated Circuit (PIC) and graphical user interface (GUI) for educational applications. In Proceedings of the 2016 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), Selangor, Malaysia, 22 October 2016; pp. 1–6.
45. Mehta, K.; Anand, R.S. Robust front-end and back-end processing for feature extraction for Hindi speech recognition. In Proceedings of the 2010 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 28–29 December 2010; pp. 1–4.
46. Eclipse CDT (C/C++ Development Tooling). Available online: <https://www.eclipse.org/cdt/> (accessed on 20 April 2022).
47. Raitahila, I. Software Architectures in Embedded Systems. Available online: https://www.cs.helsinki.fi/u/iivorait/Software_Architectures_in_Embedded_Systems.pdf (accessed on 20 April 2022).
48. Lee, J.; Park, G.; Shin, J.; Lee, J.; Sreenan, C.; Yoo, S. SoEasy: A Software Framework for Easy Hardware Control Programming for Diverse IoT Platforms. *Sensors* **2018**, *18*, 2162. [CrossRef] [PubMed]