Simple Software Simulator for Teaching Embedded Programming

https://doi.org/10.3991/ijep.v12i6.28193

Jan Dolinay(✉), Petr Dostalek, Vladimir Vašek
Tomas Bata University in Zlin, Zlín, Czech Republic
dolinay@utb.cz

Abstract—This article presents simple software simulator of a microcontroller evaluation board FRDM-KL25Z. The simulator was developed to make it possible to teach our embedded systems course online during the COVID-19 pandemic. It is in principle a software library that handles function calls and register access from student's program and displays the outputs in a console window of a standard desktop application. It does not require any special hardware or software tools except an IDE capable of building C++ applications for the desktop computer. It can be easily modified for different microcontrollers and thus can be useful if existing lessons need to be switched from in-person to distance learning at a short notice.

Keywords—microcontroller, simulator, Kinetis, embedded programming

1 Introduction

The COVID-19 pandemic forced great number of students and teachers to rethink the way they learn and teach, as they needed to switch to distance learning to be able to continue the educational process in partial or complete isolation [1]. This brought many challenges. For example, teachers found that the conventional model based on teacher explaining the materials is not effective in online teaching [2] and the courses need to be adapted [3]. Online evaluation of the students can present significant problems as well [4]. Both teachers and students also had to learn how to use new technologies [5].

Practical courses were especially affected by the unprepared shift to distance learning as these courses include application or practical work which is not available in online environment [6]. Embedded programming courses are good example of such practical courses. They are traditionally taught using specific hardware – an evaluation board with a microcontroller (MCU). Such a board is normally provided in the labs together with other devices needed for the course; the students do not have the hardware at home.

This was also our case when faced with the requirement to organize our courses of embedded systems programming online during the winter semester of 2020. In the labs we use FRDM-KL25Z evaluation board attached to a custom-made expansion board with many peripherals, such as LCD display, switches, potentiometer, rotation encoder, temperature, and humidity sensor, etc. [7].

There are just enough boards for the 12 workplaces in our lab. Approximately 60 students take turns on the equipment every week during the semester. Although in the summer of 2020 it looked that the course will be organized as usual, perhaps with the lectures in online mode, as the pandemic situation rapidly worsened at the beginning of the semester, it was clear that also the labs will have to be taught online. We came up with the following options:

1. Use remote access to the lab.
2. Use hardware which each student has (student buys or school buys or lends).
3. Use simulation (student needs just a computer/laptop to do the exercises).

As the title of the article suggests we decided for the third option. However, this is not to say that this option is be the best; rather it was the only one that was feasible given the limited time. Let us therefore briefly discuss all the above-mentioned options.

1.1 Remote access to the lab

In general, enabling remote access to computers in school labs is not a problem and it is often used as a replacement of physical labs for various reasons besides the pandemics [8]. Remote access was used at our faculty in several courses that require specific software (which the students cannot install at their computers because of license, hardware, or other restrictions). However, for embedded programming course physical access to the evaluation board with microcontroller is needed for user interaction, such as pressing a switch or rotating a knob. One solution to this problem is described in [9]. In the lab the MCU kit is connected to a computer running standard IDE (Keil microVision in this case) and the remote access to this computer is realized via remote access application. This allows students to upload and debug their programs. The physical access to the board is realized using web camera and data acquisition board that generates physical signals equivalent to a button press etc. The authors created virtual instruments in LabVIEW to allow the students control various peripherals of their MCU board using graphical interface. The drawback of this solution is that it is based on relatively expensive software. In our situation, it would not be possible to implement this kind of system in the limited time available even if the necessary investment would be granted.

1.2 Providing students with the hardware

If each student had their own development board, they could work with the programs during online lessons in a similar way as if physically present in the labs. As an added benefit they could work on the assigned tasks at any time that suits them; not being limited by the busy schedule of the lab. This benefit applies to course with physical presence in the lab as well, which makes it an interesting option – the investment required to buy the board for every student would not be used just in this pandemic situation, but the boards could be lent to new students in the following semesters.

However, this option was not practical for us because we use custom-made expansion board of which we only have 12 pieces, while there are some 60 students taking the course. Leaving aside the option of rotating limited number of boards between all the students on e.g., a weekly basis, which would be logistically very complicated and not possible for students who live far from the university or even in other countries, there are two options left for fulfilment of the goal that each student has access to the hardware: either the students buy the boards, or the school buys the boards for them. Which option is more suitable will depend on many factors, for example, in some countries students may be reluctant to spend their money on the hardware, it may be too expensive for them, or it may take too much time for all of them to obtain the boards. It may be easier that the school buys and distributes the hardware, as described in [10], if this is not complicated by administrative requirements on buying any equipment (such as selection procedures that take several months) or tight budget of many schools.

As already mentioned, we would not be able to provide students with the custom-made expansion board containing all the peripherals, but it would be possible to use just the core of our platform – the FRDM-KL25Z board that is factory-made and available in specialized shops. However, this would require considerable changes to the course contents because many example programs are based on the hardware of the custom-made expansion board. Using different hardware that is cheaper and more readily available, such as Arduino boards, would require complete change of the course materials.

Therefore, we eliminated this option even though in general, this is an interesting alternative, especially if the course uses affordable factory-made hardware.

1.3 Using simulation

The third option is not to use the hardware at all and teach with simulator. Most teachers of embedded programming (including us) will probably argue that contact with the real hardware is essential part of each embedded systems course. As [11] states, the students need to work with teaching kits in order to assimilate the basics of programming microprocessor systems. On the other hand, simulations and other technology can add value to science study [12]. There are reports on platforms that use simulation, such as interactive simulator-based approach described in [13] or software simulation packages for 8051 and PIC microcontrollers [14]. In [15] the authors present Arduino simulator which allows the users to access virtual Arduino board implemented on a web server from the Arduino IDE in the same way as if they used the physical board. An interesting system which proposes using complete toolchain based on simplified MCU design, which include simulator but can also be implemented on real hardware is presented in [16]. Simulators are also often used in online embedded systems courses which are now offered in large number [17], although some courses also employ hardware kits [18]. The research on effectiveness of computer simulation as an instructional method does not provide uniform answers. For example, meta-analysis [19] from 40 reports found significant positive differences in achievements of students who used computer simulated experiments as compared to students who used more traditional learning activities, but no significant differences in retention or student attitudes toward the subject. Based on the findings the author recommends using simulated experiments and simulations to enhance students' learning, especially in cases where the use of traditional laboratory activities is expensive, dangerous, or impractical. A recent study

[20] which compared experimental and simulation approaches for the same learning content showed that simulation performs similarly to the experimental method.

In our opinion, simulator can be a helpful tool to improve the quality of learning, but at some point, the real hardware should be used. Nevertheless, given the options mentioned above we concluded that simulation was the only option, at least in short term, before we can implement necessary changes to the course. To make this option useful the simulator should fulfill at least these criteria:

- Use the Kinetis MCU so that the course materials would not need to be excessively updated.
- Be freely available and easy to use because there is no time to obtain licenses, etc.

Unfortunately, there seems to be no simulator support for the NXP Kinetis MCUs. There are several simulators for the ARM CPU (Kinetis MCU uses ARM core), such as [21], but these simulate only the ARM core, not the peripherals. As the peripherals of an ARM-based MCUs are manufacturer-specific, it would be the MCU manufacturer who would need to implement simulation support for the peripherals. Our course (and probably most similar courses) is focused on working with peripherals, such as timers or ADC and therefore using the ARM core simulator would not be sufficient.

2 Simple source level simulator

We decided that implementing a very simple simulator would be the best solution given the situation – we needed working environment within few weeks.

Typical MCU simulator simulates the instructions of the target CPU, reading the binary code produced by the build tools. It would not be possible for us to implement such a simulator in the time available. Therefore, we decided for an option that can be described as a source-level simulator, or in another words, a port of the MCU code to a desktop computer. This means that students will write their program for the simulated MCU in a standard IDE as a standard console application for the operating system running on their computer. The advantage of this approach is that it can be used on Windows, Linux, or Mac operating systems because it uses standard C/C++ features and there is no simulator binary that the students would need to run.

In practice, the students open or create C/C++ console application project, add the simulator to the project as a module (header and source files) and then they can paste the code used for the MCU into their source file and build it. When they run this application, it shows the status of switches, LEDs, and an LCD display in the console window. A switch press on the evaluation board can be simulated by pressing keyboard keys.

2.1 Usage of the simulator

In the in-person labs we use Kinetis Design Studio IDE (KDS). In this IDE the students are able to open example projects or create their own projects with the help of several peripheral drivers, e.g., for an LCD display or for serial communication.

In simulator-based lessons a different IDE must be used because the KDS is not able to build programs for the desktop computer. This is not a problem, as both the instructors and students already have experience with C/C++ development for desktop computers from previous courses. We provided support materials for using the simulator in CodeBlocks and Qt Creator IDEs in the form of preconfigured example projects that the students could just open and run. If a new project is created, it is necessary to add the source files of the simulator to the project. To simplify this process, we concentrated the code into just two source files: simul_core.cpp and simul_kl25z.cpp. The "core" file contains general code that is not specific to the MCU but rather to the simulated I/O – it contains the code to display state of LEDs, virtual LCD display and processing of the keyboard input. The kl25z file implements the code of the simulated peripherals and drivers for our microcontroller.

The students can open the KDS project, copy the code into their simulator-enabled project in CodeBlocks or Qt Creator IDE, build and run it without any further change.

2.2 Simulator implementation

When implementing the simulator there were two main situations to handle.

- Simulating calls to driver functions.
- Simulating peripheral control using registers.

Simulating driver calls is straightforward. We just need to provide the implementation of the MCU function modified for the console application used in the simulator. For example, consider LCD display driver function lcd_print that outputs texts to the LCD display on the evaluation board. In the simulator this should output the text to a console window. So, the simulation code is little more than just using the printf standard C function to output the text.

However, the focus of our course is on teaching how to control MCU peripherals using registers. This is the universal knowledge that students can apply in their career, no matter which MCU they work with. Simulating this proved more difficult. In the students program the access to a peripheral register is equivalent to an assignment to a variable, for example:

```
ADC0->SC1 = 0x05;
```

This code writes 5 to register SC1 of the ADC0 peripheral. Note that the peripheral registers are defined as a C language structure in the header files for our MCU. Thus, speaking in term of C language, the above code accesses member variable SC1 of a structure representing the ADC registers and this structure is referenced through a pointer named ADC0.

The problem in simulating peripherals is that we often need to perform an action upon the assignment operation in user program. For example, when user program writes to certain register, this should start simulated ADC conversion. Yet in the program there is no function call that we could intercept and re-implement as in the above-mentioned example of LCD driver function.

Therefore, a mechanism is needed to perform an action in the simulated peripheral when the user program assigns a value to the register of the peripheral.

One option to do this could be to monitor the variable in a background thread for changes, but apart from solvable problems with access to the variable from multiple threads it would hardly be possible to deliver realistic behavior. Consider a program that contains two consequent writes to two registers. In real peripheral the response to the first write is immediate, but in the simulator many lines of the simulated program could be executed before the background thread would be scheduled and could update the state of the peripheral. The results of this approach would be unsatisfactory.

Better solution is to handle the modification directly – to have simulator code executed upon write to the simulated register. While we are not aware of a simple way to implement this in C language, it is relatively easy to do in C++ with operator overloading technique. For this reason, we decided to use C++ language for the simulator implementation even though the original MCU programs are written in C. Choosing C++ does not present any problem; the students just need to create their projects for the simulator as a C++ (instead of C) console application. The source code of the simulated MCU program is not affected; it can be in C.

2.3 Implementing register access

Each peripheral register is represented by a class that overrides the required operators so that simulator code can be executed when the user program writes to this register. Besides the assignment operator (=), the short-hand versions of OR with assignment (|=) and AND with assignment (&=) which are commonly used in MCU code are also implemented.

A user-defined callback functions are called when these operators are invoked. We can define what these callback functions do depending on the peripheral. For example, the ADC peripheral is a class which contains several registers (member variables) of the register type. The callback functions are defined so that when the user program writes to any of the ADC registers, appropriate action is performed.

3 Peripherals available in the simulator

We implemented the following peripherals.

- Digital I/O (LEDs and push buttons) – as a driver.
- ADC – with register access.
- LCD display – as a driver.
- Timer TPM0 – with register access and simulated overflow interrupt.

3.1 Digital I/O

Digital output for LEDs and input for switches are implemented using driver functions. There are functions pinWrite and pinRead, similar to those available in the Arduino framework. These functions are included in a driver which we use in the in-person labs before learning how to control the pins directly using GPIO registers. The simulator version allows controlling the RGB LED that is contained in the FRDM-KL25Z board and three extra LEDs available in our expansion board. The state of each LED is shown in the console window of the simulated program.

There is also support for reading four push buttons which are available in our expansion board. In the simulated version these switches are controlled by pressing a key on the computer keyboard.

3.2 Analog to digital converter

The simulated ADC is greatly simplified but it allows controlling some properties of the peripheral by registers, such as configuring the resolution, starting conversion, and waiting for the conversion to complete. There is only one simulated channel available, other channels always return 0. In the simulated channel the value changes with each conversion – a next value from a predefined set is returned. It would be easy to provide a mechanism for reading the values from a configuration file, but for our purpose simple approach with several hard-coded values was sufficient.

3.3 LCD display

LCD display with four alphanumerical lines which is included in our expansion board is also supported in the simulator. In the real labs the student use driver with functions like lcd_puts (print a string) to interact with the display. The same interface is implemented for the simulated program. The output is simply printed to designated area of the console window which represents the LCD display, see Figure 1.

3.4 Timer with overflow interrupt

The simulator also supports one of the timers available in the KL25Z MCU, the TPM0 timer. In the real MCU there are two more instances of the same timer, TPM1 and TPM2. It would be quite easy to simulate these timers as well, but one timer was sufficient for our purpose.

The simulated timer properly sets the overflow flag, so it is possible to use it in programs which polls the flag to wait for certain time. It can also call a function (a simulated interrupt service routine) when the counter overflows. The interrupt service routine is defined in the same way as in the real MCU project, using weak symbol declaration. It is then called from a separate thread to simulate asynchronous execution.

4 Use case

We will illustrate the use of the simulator on an example program that uses ADC to read potentiometer input and shows the level of the input as a bar graph with three LEDs. The program repeatedly reads analog channel 11 and turns on appropriate number of LEDs depending on the obtained value from the ADC.

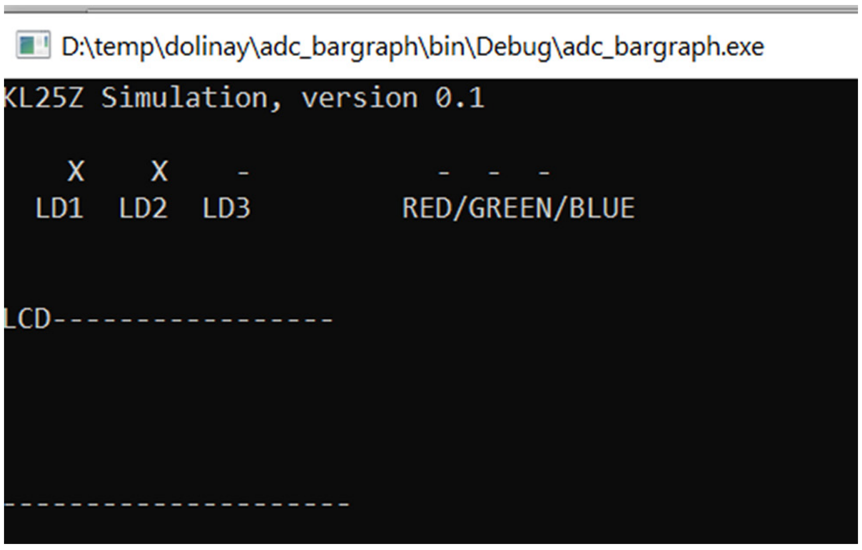
The code relevant to ADC is as follows:

```
// Initialize the ADC
ADCInit();
// Enable clock for the input port we will use
SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;
// Set the pin function to ADC
PORTC->PCR[2] = PORT_PCR_MUX(0);
while (1) {
 // Start conversion on channel 11 where our pot is
 ADC0->SC1[0] = ADC_SC1_ADCH(11);
 // Wait for the conversion to complete
 while ( (ADC0->SC1[0] & ADC_SC1_COCO_MASK) == 0 )
  ;
 // Store the result
 uint16_t result = ADC0->R[0];
…
```

In the in-person labs the student can open example project in KDS IDE, build it and run it. To use it with the simulator the student needs to copy the code of the main.c file from the KDS project and paste it into the main.cpp file of their simulator project in the C/C++ IDE, e.g., CodeBlocks.

To make it even easier for the students, example projects can be provided for the target C/C++ IDE with the code already set.

Then the student builds the project and runs it as any other console application project in the IDE. The output looks as shown in Figure 1.


D:\temp\dolinay\adc_bargraph\bin\Debug\adc_bargraph.exe

```
KL25Z Simulation, version 0.1

   X    X    -           -  -  -
 LD1  LD2  LD3         RED/GREEN/BLUE


LCD-----------------




--------------------
```

Fig. 1. The output of the ADC example program

Figure 1 shows the console window of the example program. LEDs LD1 and LD2 are lit, LED3 is off. This changes over time as the simulate ADC result changes from 0 to maximum value.

To illustrate how the simulator works let us analyze what happens in the background for several lines of the program.

When the program executes this line

```
ADC0->SC1[0] = ADC_SC1_ADCH(11);
```

It sets the status and control register SC1 of the ADC to certain values including selecting channel 11 as the input for the ADC.

In the simulator the ADC class contains definition of the SC1 registers as follows:

```
Property<uint32_t> SC1[2];
```

The Property class is the class mentioned above with overloaded assignment operators. Upon assignment a callback function is called that validates the pin configuration (and reports error if the input pin is not properly configured) and then it sets internal flag indicating that the conversion is in progress. The relevant part of the callback function code is shown below:

```
// For our pin - channel 11 check the pin config,
// for other channels just
// skip test and simulate that conversion completed.
// This is needed for calibration etc.
int channel = (data & ADC_SC1_ADCH_MASK) >>
ADC_SC1_ADCH_SHIFT;
if ( channel == 11 ) {
   if ( ADC0->IsPinConfigValid() )
      ADC0->mConversionStarted = true;
} else {
     // conversion must be started for any other
     // channel, also for calibration
     ADC0->mConversionStarted = true;
}
```

Then the code calls internal function UpdateData that first checks if the ADC clock is enabled and outputs an error if not.

```
// test if clock is enabled
if ( (SIM->SCGC6 & SIM_SCGC6_ADC0_MASK) == 0 ) {
   printf("BSOD - Default ISR :) \nADC - clock not
   enabled!\n");
   while (1) ;
}
```

If the ADC is configured properly, it sets new value into the simulated results register R,

```
R[0].data = adc_values[mCurrentDataIndex++] << shift;
```

and sets the conversion complete flag:

```
SC1[0].data |= ADC_SC1_COCO_MASK;    // set coco flag
```

When the simulated program waits for the conversion to complete, it reads the value of the SC1[0] member variable of the ADC class. Since the COCO flag is set, the program continues to read the result like this:

```
uint16_t result = ADC0->R[0];
```

The code just obtains the value from the R member variable of the ADC class. This variable contains one of the predefined conversion results.

4.1 Implementation overview

Besides the code that is executed from the simulated program directly, by callback mechanism attached to the assignment operators, the simulator contains three threads that run independently of the simulated program. There is a thread that updates the display (console output) – it prints the contents of global buffers to the console. These buffers are updated by the code that simulates LCD and digital outputs (LED).

Second thread processes keyboard input and translates it to the state of the simulated digital inputs. This is then processed by the part of the digital I/O driver that simulates input switches.

Lastly, there is a thread that simulates the timer. It repeatedly executes a function where the hardware of the timer is simulated; it increments the counter and if an overflow occurs, it sets the appropriate flags. If interrupt is enabled, it also calls the simulated ISR.

4.2 Practical experience

We used the simulator in our course during the winter semester 2020. The students started in the labs but after 4 weeks we had to switch to online classes due to worsening pandemic situation. The projects developed by students during the semester included a bicycle light with simulated LEDs flashing in different modes or simple program that reads ADC values and turns on different number of LEDs based on the values (a bar graph). As mentioned later, towards the end of the semester we switched to TinkerCad simulator for Arduino which was also used for the final student's project.

At the end of the semester students answered short survey about the course. In total, 54 students finished the course and completed the survey. The survey questions focused on general experience with the course, such as whether the lectures and labs were clear, or the quality of learning materials was adequate. Nevertheless, several students mentioned positive experience with the simulator in their comments. They appreciated the ability to see the output of their program, even though it was just in a console window.

There were no negative notes regarding the simulator. However, further research is needed to evaluate the benefits and disadvantages of using this simple simulator, preferably in comparison with classes using the hardware kit during the same semester.

We are aware that this solution cannot provide adequate replacement of the hardware kit for complex tasks such as using timers to generate pulse-width modulated signal. Also, the choice of peripherals is very limited. Nevertheless, it served its purpose and provided us with some time to prepare lessons in an alternative environment. For the last few weeks of the semester, we also used TinkerCad online environment with Arduino Uno board, using programs that access the peripherals of the MCU at registry level. Registry access is supported by TinkerCad simulator to some extent; it is even possible to set up a timer to generate PWM signal of desired frequency and check the results in virtual oscilloscope. However, in our experience not all the hardware features work in the simulator. There were cases where certain code worked on a real board but in the simulator it did not.

5 Conclusion

The COVID-19 pandemic brought dramatic changes to education. Schools were shut and courses had to be organized remotely. Practical courses, such as embedded systems programming, were among the most affected by this unprepared shift to distance learning as the equipment used in these courses cannot be easily made available online. Alternative ways of teaching had to be used. In this article we described a simulator which we implemented to quickly replace physical hardware kit for our embedded systems course.

The main advantage of our solution is that it can be used quickly without the need to change existing course materials. Students are able to use the same example programs as in the in-person labs, only in different development environment. The output of the program is displayed in a console window. Another advantage is that the simulator can provide more debugging information than the real hardware, which is useful especially in the distance learning settings, where the instructor is not immediately available to assist with problems in the program. Thus, the simulator makes the learning process smoother and less frustrating for the students.

From implementation point of view the advantage is that the simulator code can be easily modified for other platforms and hardware boards, so it can be quickly used also in courses based on different hardware.

Nevertheless, this simulator should be considered an emergency option for the specific situation where an embedded systems course must be switched from traditional to online form at a short notice, rather than a full featured replacement for hardware kits or professional simulation tools. Its main limitation is that it offers only a small subset of the peripherals available in the real hardware. Namely, ADC and timer are implemented at the registry level, allowing students to practice low-level control of these peripherals. GPIO and LCD display functions are implemented using high-level driver functions.

From an educator's point of view, using the simulator was a useful experience. Although not completely new to this idea, we were not aware of certain benefits. For example, one of the common programming errors with an ARM MCU is that the students do not enable the clock signal for a given peripheral in their code. When the program attempts to access registers of this peripheral, the MCU generates an exception, and the program stops in a default exception handler. This is a rather confusing situation for the students, as they find their program stopped in the debugger in unknown assembly code. In the simulator, it is easy to output a message such as “ADC clock is not initialized”, which immediately points to the problem.

With the experience brought by the COVID-19 pandemic, we believe it will be beneficial to employ some kind of simulator in our embedded systems course even in normal situation where students are physically present in the labs. Although the hands-on experience with real hardware is irreplaceable, simulator can improve the learning in several ways, such as the above-mentioned extended error reporting or the obvious advantage that students can experiment with their programs outside the labs.

6 References

[1] Ž. Bojović, P. D. Bojović, D. Vujošević and J. Šuh, “Education in Times of Crisis: Rapid Transition to Distance Learning,” *Comput Appl Eng Educ*., vol. 28, pp. 1467–1489, 2020. https://doi.org/10.1002/cae.22318

[2] P. Jana, N. Nurchasanah and S. Fatih ‘Adna, “E-Learning During Pandemic Covid-19 Era: Drill Versus Conventional Models,” *International Journal of Engineering Pedagogy*, vol. 11, no. 3, pp. 54–70, 2021. https://doi.org/10.3991/ijep.v11i3.16505

[3] N. Muhammad and S. Srinivasan, “Online Education During a Pandemic – Adaptation and Impact on Student Learning,” *International Journal of Engineering Pedagogy*, vol. 11, no. 3, pp. 71–83, 2021. https://doi.org/10.3991/ijep.v11i3.20449

[4] D. Idnani, A. Kubadia, Y. Jain and C. P. Prathamesh, “Experience of Conducting Online Test During COVID-19 Lockdown: A Case Study of NMIMS University,” *International Journal of Engineering Pedagogy*, vol. 11, no. 1, pp. 49–63, 2021. https://doi.org/10.3991/ijep.v11i1.15215

[5] L. N. Fewella, L. M. Khodeir and A. H. Swidan, “Impact of Integrated E-learning: Traditional Approach to Teaching Engineering Perspective Courses,” *International Journal of Engineering Pedagogy*, vol. 11, no. 2, pp. 82–101, 2021. https://doi.org/10.3991/ijep.v11i2.17777

[6] I. A. Elhaty, T. Elhadary, R. Elgamil and H. Kilic, “Teaching University Practical Courses Online during COVID-19 Crisis: A Challenge for Elearning,” *J. Crit. Rev.*, vol. 7, no. 8, pp. 1–10, 2020.

[7] J. Dolinay, P. Dostálek and V. Vašek, “ARM-Based Microcontroller Platform for Teaching Microcontroller Programming,” *International Journal of Education and Information Technologies*, vol. 10, pp. 113–119, 2016.

[8] J. Ma and J. V. Nickerson, “Hands-On Simulated and Remote Laboratories: A Comparative Literature Review,” *ACM Comput. Surv*., vol. 38, no. 3, pp. 7, 2006. https://doi.org/10.1145/1132960.1132961

[9] M. Gilibert, J. Picazo, M. Auer, A. Pester, J. Cusidó and J. A. Ortega, “80C537 Microcontroller Remote Lab for E-Learning Teaching,” *International Journal of Online Engineering*, vol. 2, no. 4, pp. 1–3, 2006.

[10] N. Alamatsaz and A. Ihlefeld, “Teaching Electronic Circuit Fundamentals via Remote Laboratory Curriculum,” *Biomedical Engineering Education*, vol. 1, no. 1, pp. 105–108, 2021. https://doi.org/10.1007/s43683-020-00008-x

[11] M. Hedley and S. Barrie, "An Undergraduate Microcontroller Systems Laboratory," *IEEE Trans. on Education*, pp. 345–353, 1998. https://doi.org/10.1109/TE.1998.787371
[12] L. Newton, and L. Rogers, "Thinking Frameworks for Planning ICT in Science Lessons," *School Science Review*, pp. 113–120, 2003.
[13] S. Tang, "An Interactive Simulator-Based Pedagogical (ISP) Approach for Teaching Microcontrollers in Engineering Programs," *Advances in Engineering Education*, pp. 1–18, 2014.
[14] N. Swain, "Teaching Microcontrollers through Simulation," in *2011 ASEE Annual Conference & Exposition*, Vancouver, BC, Canada, 2011, pp. 1–14.
[15] F. Paulo, J. S. Gonçalves, A. Coelho and J. Durães, "An Arduino Simulator in Classroom – a Case Study," in *First International Computer Programming Education Conference (ICPEC 2020),* Online, 2020.
[16] S. Sirowy, D. Sheldon, T. Givargis and F. Vahid, "Virtual Microcontrollers," *ACM SIGBED Review*, vol. 6, no. 1, pp. 1–8, 2009. https://doi.org/10.1145/1534480.1534486
[17] M. Koenig and R. Rasch, "Digital Teaching an Embedded Systems Course by Using Simulators," *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*, 2021, pp. 1–7. https://doi.org/10.1109/WCAE53984.2021.9707146
[18] J. W. Valvano, R. Yerraballi, and C. Fulton, 2016. Teaching Embedded Systems in a MOOC Format. *2016 ASEE Annual Conference & Exposition*, June 28, 2016, New Orleans, USA.
[19] J. V. LeJeune, A Meta-Analysis of Outcomes from the Use of Computer-Simulated Experiments in Science Education, *Dissertation*, Texas A&M University, 2002.
[20] A. Zendler, H. Greiner, The Effect of Two Instructional Methods on Learning Outcome in Chemistry Education: The Experiment Method and Computer Simulation, *Education for Chemical Engineers*, vol. 30, pp. 9–19, 2020. https://doi.org/10.1016/j.ece.2019.09.001
[21] Crossware (2020). ARM Simulator [Online]. Available: https://www.crossware.com/arm/simulator

7 Authors

Jan Dolinay, Tomas Bata University in Zlin, Zlín, Czech Republic. E-mail: dolinay@utb.cz

Petr Dostalek, Tomas Bata University in Zlin, Zlín, Czech Republic.

Vladimir Vašek, Tomas Bata University in Zlin, Zlín, Czech Republic.

Article submitted 2021-11-10. Resubmitted 2022-04-19. Final acceptance 2022-11-07. Final version published as submitted by the authors.

Imprint

iJEP – International Journal of Engineering Pedagogy
Online issue: http://www.i-jep.org

Indexing
International Journal of Engineering Pedagogy (iJEP) is indexed in Elsevier Scopus, Clarivate Analytics ESCI, EBSCO, DOAJ and DBLP.

Publication Frequency
Bi-monthly (January, March, May, July, September, November)

ISSN
2192-4880

Publisher
International Society of Engineering Education (IGIP)
Europastrasse 4
A-9524 Villach
Austria