International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 14, Number 3 (2021), pp. 271-277

Use of Embedded Systems on Software Simulation Environments Applied to Virtual Education

Fernando Martínez Santa[1], Fredy H. Martínez S.[2] and Edwar Jacinto Gómez[3]

Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá D.C, Colombia.

[1]*ORCID:* 0000-0003-2895-3084 [2]*ORCID*: 0000-0002-7258-3909
[3]*ORCID:* 0000-0003-4038-8137

Abstract

This paper describes the implementation of a simulation platform for embedded systems running a Python language interpreter. This platform uses Proteus-ISIS® electronics simulator, a reduced and efficient implementation of Python language for microcontrollers named MicroPython, and an implementation of an embedded systems development board based on an ARM architecture microcontroller. This simulation platform is proposed in order to be implemented instead of real laboratories for Electrical Technology and Engineering students in the *Microcontrollers Architecture* subject at Universidad Distrital in Bogotá Colombia. The process to join these software tools and components as a complete simulation platform is described in this paper step by step. At the end some proposed simulation laboratories are described and the results of implementing this approach instead of the regular one are shown and analyzed.

Keywords: Simulation Environment, Virtual Education, Embedded Systems, Python, MicroPython, Microcontroller.

1. INTRODUCTION

Software simulation environments have been priceless support tools for education and training in engineering and other multiple areas, allowing the students to do practices without the necessity of having real tools and instruments. These simulation software have been used in areas such as physics [1], [2], building and engineering design [3], [4], automotive [5], [6], energy efficiency [7], [8], software engineering and gamification [9]–[12], among others, and always they have demonstrated their applicability and efficiency as support tools in education and research. In 2020 due to the world health emergency produced by the COVID-19 virus most of countries decided to restrict the free movement of people in the cities and the agglomerations, which affected the normal way of teaching in schools and universities, obligating to adopt a new approach of on-line or virtual education in most of the education institutions. In some of the undergraduate programs such as engineering most of the subjects require to do laboratory practices, that is why the software simulation environments turned from support tools to a real necessity for completing the students learning nowadays. In electric and electronic engineering, there are several simulation software which works really well for electric analog circuits and basic digital circuits but simulating complex digital devices such as microcontrollers or FPGAs requires specialized software [13]–[15] even more if it is wanted to simulate complex embedded systems or development boards [16]–[18]. In the embedded systems area, several researches about simulation implementations have been done about topics such as networking [19], system-level modeling [20], co-simulation [21] and learning [22], [23]. Nowadays, the embedded systems are used for many different applications, mainly in IoT (Internet of Things) implementations, which require increasingly more memory and speed features, due to that the embedded systems have been highly improved. Those improvements allow using on them operative systems such as Android or Linux and/or interpreted languages or virtual machines, doing more difficult their software simulation. One of the most popular language used in data science is Python due to it has advantages in the development time and abstraction than compiled ones as C/C++, but at the same time it requires a lot of memory and a working high speed because it has to be interpreted. Some Python implementations on embedded systems has been done such as Zerynth® and MicroPython [24] this last one is an open source implementation, that was the reason why it was selected for the work shown in this paper. On the other hand, one of the most used simulation software for microcontrollers is Proteus ISIS® [25], which allows simulating complex embedded systems such as Raspberry Pi® among others. The main objective of the work reported in this paper is to implement a simulation environment based on Proteus ISIS® software, which is able to simulate embedded systems running MicroPython interpreter, focused on the teaching of electric and electronic engineering specially on the subjects of microcontrollers and advanced embedded systems.

2. METHODOLOGY

The order of this work is presented as follows. First, a brief description of the MicroPython project is presented as an introduction to the firmware to be simulated, including how to compile it and download it to a microcontroller. The next section shows the description and features of the microcontroller-based development board to be used as embedded system for the simulation. After that, the step-by-step process to make the process work, including the source code download, the setup and compiling process, the firmware

download and the simulation itself, all of this focused on Windows® operative system. Finally, the simulation samples obtained are shown as result of the implemented simulation architecture and its reliability of being used as educational supporting tool.

3. MICROPYTHON FIRMWARE

MicroPython is an open-source project that pretends to make it possible to run Python scripts on microcontrollers restricted in memory size, this implements a subset of Python 3 language including some of its standard libraries [26]. This efficient implementation is focused on microcontrollers and/or constrained environments, but including advance features such as: REPL (Read Evaluate Print Loop) prompt, list comprehension, exception handling, adaptable precision integers, garbage collection, among others. Table 1 shows some of the microcontrollers supported for MicroPython, indicating the processor features including the FPU (Float Point Unit).

Table 1. Summary list of the microcontrollers supported by MicroPython.

Brand (Family)	Reference	ROM [kB]	RAM [kB]	CPU speed [MHz]	FPU
ST Cortex-M7	STM32F722IEK	512	256	216	Yes
ST Cortex-M4	STM32F405RG	1024	192	168	Yes
ST Cortex-M4	STM32F401CD	384	96	84	Yes
ST Cortex-M0	STM32F091RC	256	32	48	No
Microchip dsPIC	33FJ256GP506	256	16	80	No
Microchip Cortex-M0+	ATSAMD21G18	256	32	48	No
TI Cortex-M4	CC3200	32 - 2048	256	80	No
Espressif LX106	ESP8266	512 - 4096	96	80 - 160	No
Espressif LX6	ESP32	448 - 16384	520	160 - 240	Yes

Additionally, MicroPython can run on JavaScript and Unix platforms, which makes it possible to run it on online emulators such as Unicorn (http://micropython.org/unicorn/) and other Linux-based embedded systems like Raspberry Pi® respectively. On the other hand, the source code of MicroPython firmware (interpreter and REPL) is implemented entirely in C language, and it is designed to be compiled basically by GNU GCC compiler (in some cases it works along with another specific compiler), that is why the preferred operative system to do this cross-compilation process is Linux or another Unix-based one. However, it is possible to compile MicroPython on a Windows® platform using some different tools or emulators such as: MinGW, Cygwin and (WSL) Windows Subsystem for Linux (only for Windows® 10) among others.

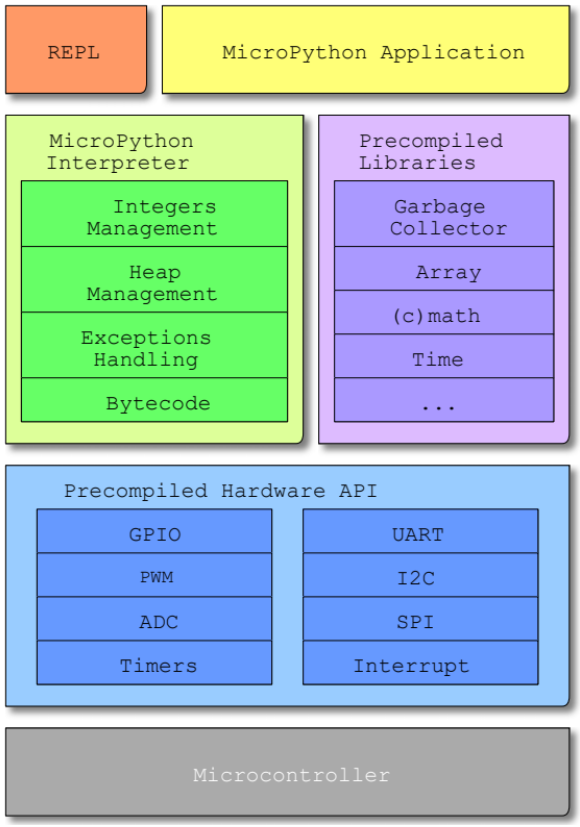

Figure 1. Layer diagram of a typical MicroPython working.

3.1 MicroPython Interpreter

As well as Python, MicroPython needs to be executed by an interpreter which reads the source code file line by line and executes the correspondent commands. This interpreter is written in C language and is compiled for a specific microcontroller. Once loaded, the interpreter is able to run Python commands one by one through the REPL (Read Evaluate Print Loop) which uses a default serial port to communicate with the user. Likewise, the interpreter is able to execute a complete Python source code file, in this case it is necessary to create a file named *main.py* in the MicroPython file system, once restarted the microcontroller this file will be executed. Along with the MicroPython interpreted, the firmware includes the precompiled hardware API for having access to the microcontroller peripherals and some precompiled common-use libraries, those ones are compiled into MicroPython Bytecode files *.mpy* which can be executed also by the interpreted but run faster than *.py* files. All of the firmware scheme of MicroPython interpreter is summarized in

the figure 1 as a layer diagram where the user only can access to the top layer, either the *REPL* or the *MicroPython Application*.

3.2 MicroPython Compiling

When MicroPython is wanted to be used on a real microcontroller or embedded system, only it is necessary to download the correspondent precompiled firmware from the official web page and load it in the microcontroller program memory. On the other hand, when MicroPython is wanted to be simulated, generally it is necessary to download and compile its source code, in order to adapt it to the available hardware in the simulation software. As previously said, MicroPython compiling needs a Unix-based platform to be executed, due to it uses programs such as *make* and *gcc*. Likewise, the installing of the specific toolchain for the microcontroller to be used is necessary. For Windows® 10 users the use of WSL (Windows Subsystem for Linux) is highly recommended because it simulates a complete Linux Kernel. Other alternative are Cygwin and MinGW, which need to install make and gcc on them for working with MicroPython.

After installing the Linux or Unix emulator and the microcontroller toolchain, it is necessary to download the source code, for this is highly recommended to clone the code repository via *git*. Then just get into the specific port directory (MicroPython version for a specific microcontroller) and run the command *make*. After than a *.hex* or *.bin* file is generated ready for being load to the microcontroller program memory.

4. SOFTWARE AND HARDWARE SELECTION

For the proposed simulation platform Proteus-ISIS® was selected as electronics simulation software, due to its large amount of processors, devices and peripherals that can be simulated on it, including some complete embedded systems such as Arduino® and Raspberry Pi®. On the other hand, a microcontroller of the brand ST® was selected, specifically the STM32F401RE with 512kB of ROM, 96kB of RAM and 84MHz of processor frequency. The main purpose is simulating an embedded system similar to the NUCLEO-F401RE development board. Figure 2 shows the diagram of the board implementation on the simulation software, including the serial interface (Virtual Terminal) for the REPL. Likewise, the algorithm 1 show the steps to compile Micropython for this board.

Algorithm 1. MicroPython compiling for the NUCLEO-F401RE board.

```
cd <MicroPython_dir>/ports/stm32
make BOARD=NUCLEO_F401RE
```

After compiled, a new directory is created in the current one, it is named *build-<board_name>*, inside it the compiled file *firmware.hex* is created. Then only it is necessary to load the *.hex* file as *Program File* in Proteus-ISIS® software and start the simulation. Figure 3 shows the result of loading only the basic firmware to the simulation, in this case as there is no *main.py* file the interpreter proceeds to run the REPL. In the example, the user manually defines two variables and operates them.

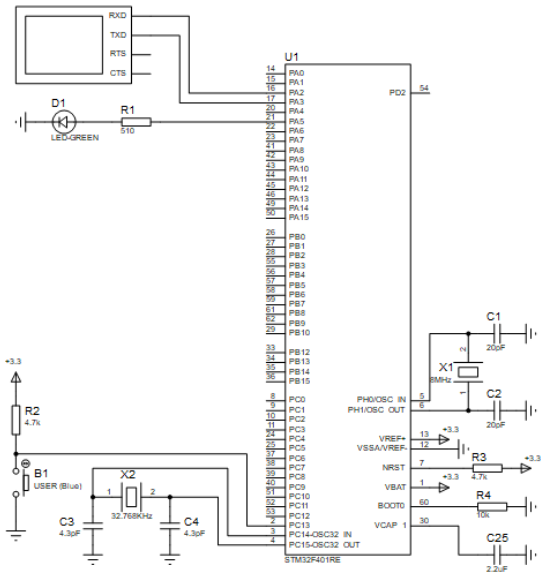

Figure 2. Implementation of the NUCLEO-F401RE board on Proteus-ISIS®.

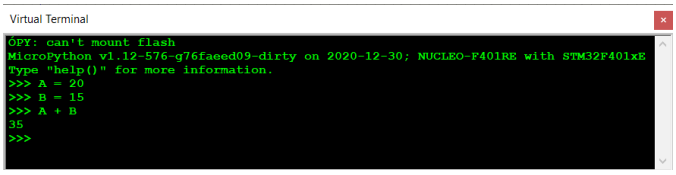

Figure 3. Execution of the MicroPython REPL on the Proteus-ISIS® Virtual Terminal.

Once MicroPython is working on the simulation, next step is load a complete Python script and run it. For this process there are two different ways, the first one is loading the source code file in the MicroPython file system via serial port (in Proteus-ISIS® using the COMPIM component and one additional virtual serial port software), and the second one is including it directly in the compiled firmware. For this case, the second way was selected in order to avoid using additional software. Including the source code inside the firmware requires to "freeze" it, then it is necessary to modify in the firmware source code the file *manifest.py* which is located in *<MicroPython_dir>\ports\stm32\boards* and include the main.py directory as shown in the algorithm 2. If any other source code file or module is wanted to be included to the project, the same

procedure has to be followed for each. After that, the firmware has to be compiled again following the commands shown in the algorithm 1.

Algorithm 2. Freezing modules or source code by modifying the *manifest.py* file.

```
include("$(MPY_DIR)/extmod/uasyncio/manifest.py")
#original frozen modules
freeze("$(MPY_DIR)/drivers/dht", "dht.py")
freeze("$(MPY_DIR)/drivers/display", ("lcd160cr.py", "lcd160cr_tes
freeze("$(MPY_DIR)/drivers/onewire", "onewire.py")

#User frozen modules including the main.py
#"libraries" and "scripts" directories are user-defined
freeze("$(MPY_DIR)/ports/stm32/scripts", "main.py")
freeze("$(MPY_DIR)/ports/stm32/scripts", "functions.py")
freeze("$(MPY_DIR)/libraries/micropython-tm1637-master", "tm1637.p
freeze("$(MPY_DIR)/libraries/micropython-charlcd-master", "LCD.py"
```

5. RESULTS

Six different sample laboratories are developed in order to be performed on the proposed simulation MicroPython platform, which are listed as follows:

- Printing messages on the Virtual Terminal.
- 4-LED sequence commanded by a switch.
- Printing of the converted value by the ADC.
- LEDs Indication of the state of an analog input.
- Integer counter with four 7-segment displays using a TM1637 driver.
- Printing messages on a 16x2 LCD using a PCF8574A driver.

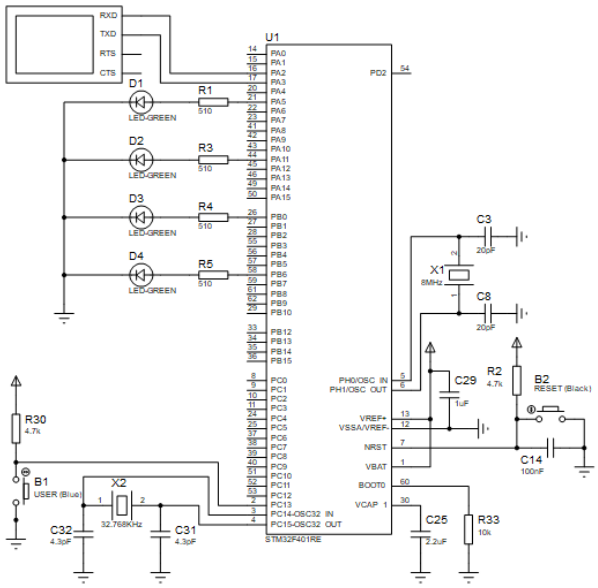

Figure 4. Sample laboratory 2, 4-LED sequence commanded by a switch.

Algorithm 3. MicroPython source code for the sample laboratory 2.

```
from machine import Pin
from time import sleep_ms

led1 = Pin('PA5',Pin.OUT) #outputs declaration
led2 = Pin('PA11',Pin.OUT)
led3 = Pin('PB0',Pin.OUT)
led4 = Pin('PB6',Pin.OUT)
pulsador = Pin('PC13',Pin.IN)   #input declaration

counter = 0

while True:
    #check the input switch
    if pulsador.value() == True:
        print('Released')
        counter -= 1
        if counter < 0:
            counter = 4
    else:
        print('Pulsed')
        counter += 1
        if counter > 4:    #if greater than 4
            counter = 0    #restart it
    #turn on the LEDs according to "counter"
    if counter == 0:
        led1.off(); led2.off(); led3.off(); led4.off()
    elif counter == 1:
        led1.off(); led2.off(); led3.off(); led4.on()
    elif counter == 2:
        led1.off(); led2.off(); led3.on(); led4.on()
    elif counter == 3:
        led1.off(); led2.on(); led3.on(); led4.on()
    elif counter == 4:
        led1.on(); led2.on(); led3.on(); led4.on()
    sleep_ms(500) #delay
```

The figures 4, 5 and 6 show the correspondent implementation diagrams for the laboratories 2,4 and 5 respectively. Likewise the algorithms 3, 4 and 5 shows the source code proposed for each of the same laboratories.

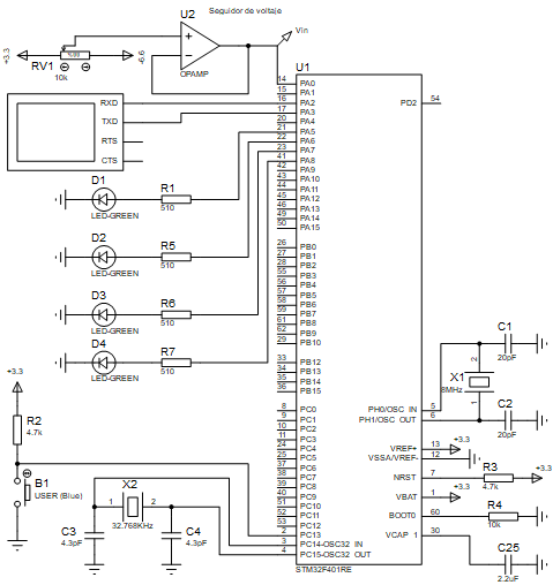

Figure 5. Sample laboratory 4, LEDs Indication of the state of an analog input.

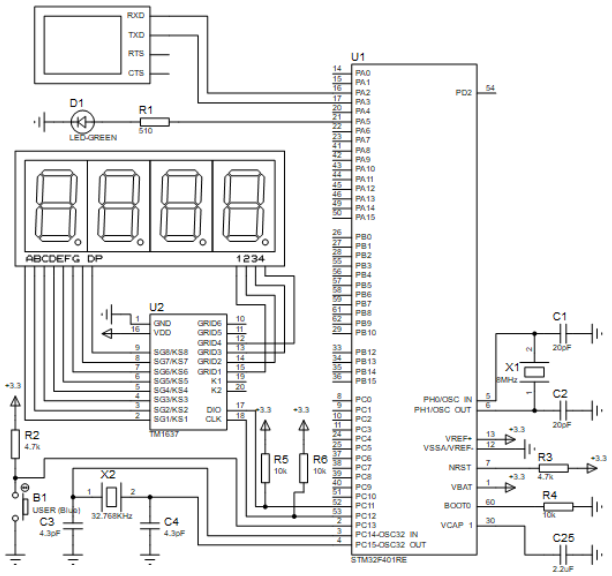

Figure 6. Sample laboratory 5, Integer counter with four 7-segment displays using a TM1637 driver.

Algorithm 4. MicroPython source code for the sample laboratory 4.

```
from pyb import ADC,Pin
from time import sleep_ms

adc = ADC(Pin('PA0'))   #define the analog input
led1 = Pin('PA5',Pin.OUT)  #outputs declaration
led2 = Pin('PA6',Pin.OUT)
led3 = Pin('PA7',Pin.OUT)
led4 = Pin('PA8',Pin.OUT)

while True:
    pot = adc.read() #read de 12-bit ADC (0-4095)
    voltage = (pot / 413.63)

    print('ADC = ',voltage)    #print the voltage

    if pot <= 819:
        led1.off(); led2.off(); led3.off(); led4.off()
    elif pot <= 819*2:  #1638
        led1.off(); led2.off(); led3.off(); led4.on()
    elif pot <= 819*3:  #2457
        led1.off(); led2.off(); led3.on(); led4.on()
    elif pot <= 819*4:  #3276
        led1.off(); led2.on(); led3.on(); led4.on()
    else:               #4095
        led1.on(); led2.on(); led3.on(); led4.on()

    sleep_ms(2000)
```

Algorithm 5. MicroPython source code for the sample laboratory 5

```
from tm1637 import TM1637
from machine import Pin

#driver tm1637 4 displays of 7 segments
tm = TM1637(clk=Pin('PC12'), dio=Pin('PC11'))


while True:
    for n in range(0,2000,127):
        print(n)        #terminal
        tm.integer(n)   #displays
```

6. CONCLUSION

The proposed MicroPython simulation platform and the designed laboratories were implemented during the first term of 2020 in the subject of *Microcontrollers Architecture* at Technology Faculty of Universidad Distrital Francisco José de Caldas in Bogotá, Colombia, specifically on the Electrical Technology and Engineering undergraduate programs. This proposal was done in order to give this subject totally virtually, without the necessity of using real laboratories, all of this due to the current world health emergency produced by the COVID-19 virus.

In general, the implementation of the simulations was successful but some of them run not in real-time, producing delays in the simulation and in some cases bugs. Almost half of the students of the *Microcontrollers Architecture* subject could run the simulations and obtain successful results, but the rest realized in at least one of the proposed laboratories that issues and bugs were presented and in some cases the simulator locked. The cause of this problem is that each student worked on its own personal computer, having too different features among them. Being the Universidad Distrital a public university most of the students presents a difficult social and economical situation which makes it hard

that they can acquire the adequate computer equipment required to carry out software simulations and computing in engineering.

ACKNOWLEDGMENTS

This work was supported by the Universidad Distrital Francisco José de Caldas, specifically by the Technology Faculty. The authors thank all of the students involved in the test and evaluation of the simulation platform, and the people of the research group ARMOS for their support in the development of this work.

REFERENCES

[1] M. Cápay and N. Klimová, "Engage Your Students via Physical Computing!," in *2019 IEEE Global Engineering Education Conference (EDUCON)*, 2019, pp. 1216–1223.

[2] L. M. Molías, J. M. C. Ranilla, and M. G. Cervera, "Pre-service Physical Education Teachers' self-management ability: a training experience in 3D simulation environments," *Retos nuevas tendencias en Educ. física, Deport. y recreación*, no. 32, pp. 30–34, 2017.

[3] P. Rajagopalan, J. P. C. Wong, and M. M. Andamon, "Building performance simulation in the built environment education: Experience from teaching two disciplines," in *Proceedings of the 50th International Conference of the Architectural Science Association*, 2016, pp. 359–368.

[4] C. Xie, C. Schimpf, J. Chao, S. Nourian, and J. Massicotte, "Learning and teaching engineering design through modeling and simulation on a CAD platform," *Comput. Appl. Eng. Educ.*, vol. 26, no. 4, pp. 824–840, 2018.

[5] E. Cioroaica, F. Pudlitz, I. Gerostathopoulos, and T. Kuhn, "Simulation methods and tools for collaborative embedded systems: with focus on the automotive smart ecosystems," *SICS Software-Intensive Cyber-Physical Syst.*, vol. 34, no. 4, pp. 213–223, 2019.

[6] F. Oszwald, P. Obergfell, M. Traub, and J. Becker, "Using Simulation Techniques within the Design of a Reconfigurable Architecture for Fail-Operational Real-Time Automotive Embedded Systems," in *2018 IEEE International Systems Engineering Symposium (ISSE)*, 2018, pp. 1–3.

[7] L. Bogdanov, "Statement-level energy simulation in embedded systems using GCC," in *2016 XXV International Scientific Conference Electronics (ET)*, 2016, pp. 1–4.

[8] P. Haririan, "DVFS and Its Architectural Simulation Models for Improving Energy Efficiency of Complex Embedded Systems in Early Design Phase," *Computers*, vol. 9, no. 1, p. 2, 2020.

[9] O. Chernikova, N. Heitzmann, M. Stadler, D. Holzberger, T. Seidel, and F. Fischer, "Simulation-based learning in higher education: A meta-analysis," *Rev. Educ. Res.*, vol. 90, no. 4, pp. 499–541, 2020.

[10] M. Kosa, M. Yilmaz, R. O'Connor, and P. Clarke, "Software engineering education and games: a systematic literature review," *J. Univers. Comput. Sci.*, vol. 22, no. 12, pp. 1558–1574, 2016.

[11] T. A. Vakaliuk, V. V Kontsedailo, D. S. Antoniuk, O. V Korotun, I. S. Mintii, and A. V Pikilnyak, "Using game simulator Software Inc in the Software Engineering education," *arXiv Prepr. arXiv2012.01127*, 2020.

[12] D. Vlachopoulos and A. Makri, "The effect of games and simulations on higher education: a systematic literature review," *Int. J. Educ. Technol. High. Educ.*, vol. 14, no. 1, p. 22, 2017.

[13] D. E. Bolanakis, "A Survey of Research in Microcontroller Education," *IEEE Rev. Iberoam. Tecnol. del Aprendiz.*, vol. 14, no. 2, pp. 50–57, 2019.

[14] K. Mondal and A. Elias-Medina, "Introducing Secure Design by Scripting in an Undergraduate Microcontroller Based Design Course," in *Journal of The Colloquium for Information Systems Security Education*, 2020, vol. 7, no. 1, p. 6.

[15] S. Varoumas, B. Pesin, B. Vaugon, and E. Chailloux, "Programming microcontrollers through high-level abstractions," in *Proceedings of the 12th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*, 2020, pp. 5–14.

[16] M. Ben Ayed, Y. Ben Salah, and M. Abid, "Conceptual/functional Co-simulation technique for embedded systems," in *2019 International Conference*

on Computer and Information Sciences (ICCIS), 2019, pp. 1–5.

[17] J. C. Kirchhof, E. Kusmenko, J. Meurice, and B. Rumpe, "Simulation of Model Execution for Embedded Systems," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 331–338.

[18] J. Kraft, "RTSSim-a simulation framework for complex embedded systems," *Tech. Rep.*, 2009.

[19] F. Fummi, D. Quaglia, and F. Stefanni, "A SystemC-based framework for modeling and simulation of networked embedded systems," in *2008 Forum on Specification, Verification and Design Languages*, 2008, pp. 49–54.

[20] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP J. Embed. Syst.*, vol. 2007, no. 1, p. 82123, 2007.

[21] J. S. Fitzgerald, P. G. Larsen, K. G. Pierce, and M. H. G. Verhoef, "A formal approach to collaborative modelling and co-simulation for embedded systems," *Math. Struct. Comput. Sci.*, vol. 23, no. 4, pp. 726–750, 2013.

[22] L. A. Ajao, J. Agajo, J. G. Kolo, M. A. Adegboye, and Y. Yusuf, "Learning of embedded system design, simulation and implementation: A technical approach," *Am. J. Embed. Syst. Appl.*, vol. 3, no. 3, pp. 35–42, 2016.

[23] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Learning-based response time analysis in real-time embedded systems: A simulation-based approach," in *2018 IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies (SQUADE)*, 2018, pp. 21–24.

[24] M. Khamphroo, N. Kwankeo, K. Kaemarungsi, and K. Fukawa, "MicroPython-based educational mobile robot for computer coding learning," in *2017 8th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)*, 2017, pp. 1–6.

[25] A. S. B. F. Rahman and A. R. B. A. Razak, "Proteus based simulation of a charge controller," in *2010 IEEE International Conference on Power and Energy*, 2010, pp. 539–542.

[26] D. George, "MicroPython," *George Robotics Limited*, 2018. [Online]. Available: https://micropython.org/.