Online-Journals.org

iJOE International Journal of Online and Biomedical Engineering

iJOE | eISSN: 2626-8493 | Vol. 21 No. 4 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijoe.v21i04.54297

PAPER

RHL-Butterfly: A Scalable IoT-Based Digital Twinning Platform for Embedded Systems and Remote Laboratories

Matthew Guo[1], Zhiyun Zhang[1], Pablo Orduña[2], Rania Hussein[1](✉)

[1]University of Washington, Seattle, WA, USA

[2]LabsLand, San Francisco, CA, USA

rhussein@uw.edu

ABSTRACT

The RHL-Butterfly is a digital twin of a breadboard designed for use with Field Programmable Gate Arrays (FPGAs) and microcontrollers in educational remote engineering laboratories and engineering education, aimed at providing greater flexibility and fostering equity in learning opportunities. This study introduces a novel server-side approach for simulating breadboard circuits using microcontroller backends, enhancing computational efficiency, and offering open-source transparency. The system interfaces the digital twin with a network of physical, remote FPGAs and microcontrollers and introduces a scalable communication protocol system that converts the graphical breadboard layout into a one-dimensional (1D) string representation for network communication. This custom protocol facilitates seamless integration with various digital twinning applications, ensuring a balance between virtualized interfaces and physical hardware and expanding the scalability and accessibility of engineering labs. Survey responses and feedback from students using the system in their course studies have been encouraging. Students particularly appreciate the equitable nature of the system and its similarity to a traditional physical breadboard.

KEYWORDS

educational remote laboratory, digital twin, virtual breadboard, embedded systems

1 INTRODUCTION

1.1 Remote laboratories

Virtualizing laboratory equipment has garnered significant attention and implementation in the past decade [1]–[5]. Recent research has indicated a preference for simulated and remote laboratory instruction, citing advantages such as cost savings, increased safety during potentially dangerous electrical experiments, and improved accessibility for learners with disabilities [6]. Additionally, these approaches have

Guo, M., Zhang, Z., Orduña, P., Hussein, R. (2025). RHL-Butterfly: A Scalable IoT-Based Digital Twinning Platform for Embedded Systems and Remote Laboratories. *International Journal of Online and Biomedical Engineering (iJOE)*, 21(4), pp. 4–28. https://doi.org/10.3991/ijoe.v21i04.54297

Article submitted 2024-10-07. Revision uploaded 2025-01-12. Final acceptance 2025-01-20.

shown no substantial disparity in educational outcomes for students [7], [8]. A study conducted at the University of Washington [9], [10] found that curricula incorporating remote laboratory hardware resulted in better analytical learning outcomes for students compared to those using traditional physical laboratory hardware. Similarly, another study examined student perspectives on the use of remote laboratories for homework assignments and revealed positive feedback regarding their convenience, ease of use, and cost-effectiveness [11], [12].

The educational system should offer tools and options that enable remote learning, thereby expanding students' access to education. Providing greater opportunities for asynchronous learning is crucial to accommodate students facing circumstances that prevent them from attending in-person classes throughout an academic term [13]. A study [14] revealed that 80% of students with prior remote learning experience expressed a desire to see some form of online instruction in their future courses. In response to this demand, the Remote Hub Lab [15] has been established as a dedicated platform offering a broad range of remote education technologies that serve as valuable learning tools whenever they are needed within the educational curriculum.

1.2 Existing digital twins of breadboard solutions

The transition to online and remote learning, accompanied by necessary curriculum adjustments, presented numerous challenges [16]. In engineering programs, which emphasized experiential learning through hands-on activities, teaching methods had to adapt while maintaining the "learning by doing" philosophy inherent in traditional in-person curricula. One widely adopted approach involved providing students with take-home kits containing essential laboratory equipment, such as breadboards and circuit components, throughout the academic term. To showcase their proficiency in breadboard circuitry, students used their cameras to record videos demonstrating the assembly of their hardware. However, a survey conducted at the end of the semester assessed student satisfaction with the remote course offerings and revealed significant dissatisfaction with the inadequate platform for seeking assistance with breadboard debugging. Many students resorted to sharing pictures and live Zoom feeds of their breadboard prototyping experiences [17]. While the utilization of take-home laboratory kits continues to uphold the "learning by doing" approach, it is crucial to acknowledge the challenges associated with the debugging process, primarily resulting from the lack of appropriate remote laboratory tools that effectively interface with real-world hardware.

Currently, there are several digital twins of breadboards available in the market and in literature. A notable example is the Java Digital Breadboard Simulator, introduced in 2002 as a thesis project at a university's computer science department. It features an open-source library and offers comprehensive simulations for circuits, logic, and functions. By balancing efficiency and realism, it enables students to explore complex circuitry and hone their skills. Furthermore, it integrates seamlessly with virtual microcontrollers and microprocessors, enabling students to explore complex circuitry and preparing them for practical applications [18]. Tinkercad [19], a free virtual breadboard simulator, is widely used in electrical engineering courses for Arduino programming and hardware interfaces. It features a digital twin of a breadboard that interfaces with a virtual Arduino, allowing students to learn basics

without physical components [20]. Like the Java Digital Breadboard Simulator, Tinkercad operates without physical hardware.

Current digital twins of breadboards and take-home lab kits offer varied perspectives on remote prototyping but lack real-world hardware integration, limiting the ability to maintain the authentic responses of physical components while simultaneously offering students easier access, sharing, and debugging options for their breadboard designs. LabsLand, spun off from the WebLab-Deusto research group, identified this gap and the need for a technical framework, business initiatives, and open-source contributions for establishing a comprehensive and collaborative remote laboratory hardware infrastructure across multiple universities globally [21]. Partnering with LabsLand, RHL-Butterfly aims to develop a new digital twin of a breadboard that seamlessly interfaces with real hardware, transforming it from a simulation tool to a powerful platform for virtualizing tangible experiences.

1.3 Previous digital twins of breadboards iterations

In the Fall of 2020, a junior-level digital logic design course at the University of Washington underwent a transition to an online format and explored the potential of remote laboratories [9]. The first assignment in the course focused on Field Programmable Gate Array (FPGA) design and aimed to review the concept of finite state machines (FSM), which students had previously learned in prerequisite courses. It also introduced the use of onboard general-purpose input/output (GPIO) and utilized an initial version of the breadboard digital twin interface to simulate circuit construction and the connection of GPIOs to students' SystemVerilog implementations of FSMs [22]. For this purpose, the digital twin utilized the VISIR breadboard libraries and connected to real FPGA DE1-SoC hardware through the LabsLand platform [23]–[25]. Following the initial offering of the digital twin of the breadboard interface, students were invited to participate in an anonymous online survey to assess the usability and practicality of the system. The survey revealed positive feedback on the user interface (UI) design and ease of use but identified areas for improvement. The previous iteration had fixed switches and light-emitting diodes (LEDs), limiting scalability for additional courses, curriculum, and design projects. Plans were made to enhance the interface with digital logic ICs and flexible component placement, expanding its capabilities for remote laboratories.

Figure 1 shows a preliminary, work-in-progress prototype of a revamped digital twin of a breadboard, initially introduced in [26], that boosts enhanced customization options aimed at enriching student learning experiences. This updated breadboard empowers students to place a diverse range of components, including LEDs, switches, and logic gates, according to their preferences. Notably, unlike its predecessor, this new iteration does not impose a fixed limit on the quantity of each component type, allowing for greater flexibility based on the intricacy and functionality of students' projects. This paper further elaborates on a fully functional, integrated curriculum leveraging a digital twin of a breadboard capable of interfacing with actual hardware targets. The system has been offered in the Design of Digital Circuits course at the University of Washington over two academic terms, during which we gathered and evaluated student feedback through anonymous surveys, assessing their experiences, learning outcomes, and accessibility.

Providing continued support to integrate and offer remote educational hardware into engineering curricula promotes greater flexibility in teaching and learning and creates additional opportunities to expand the equitable access of STEM education.

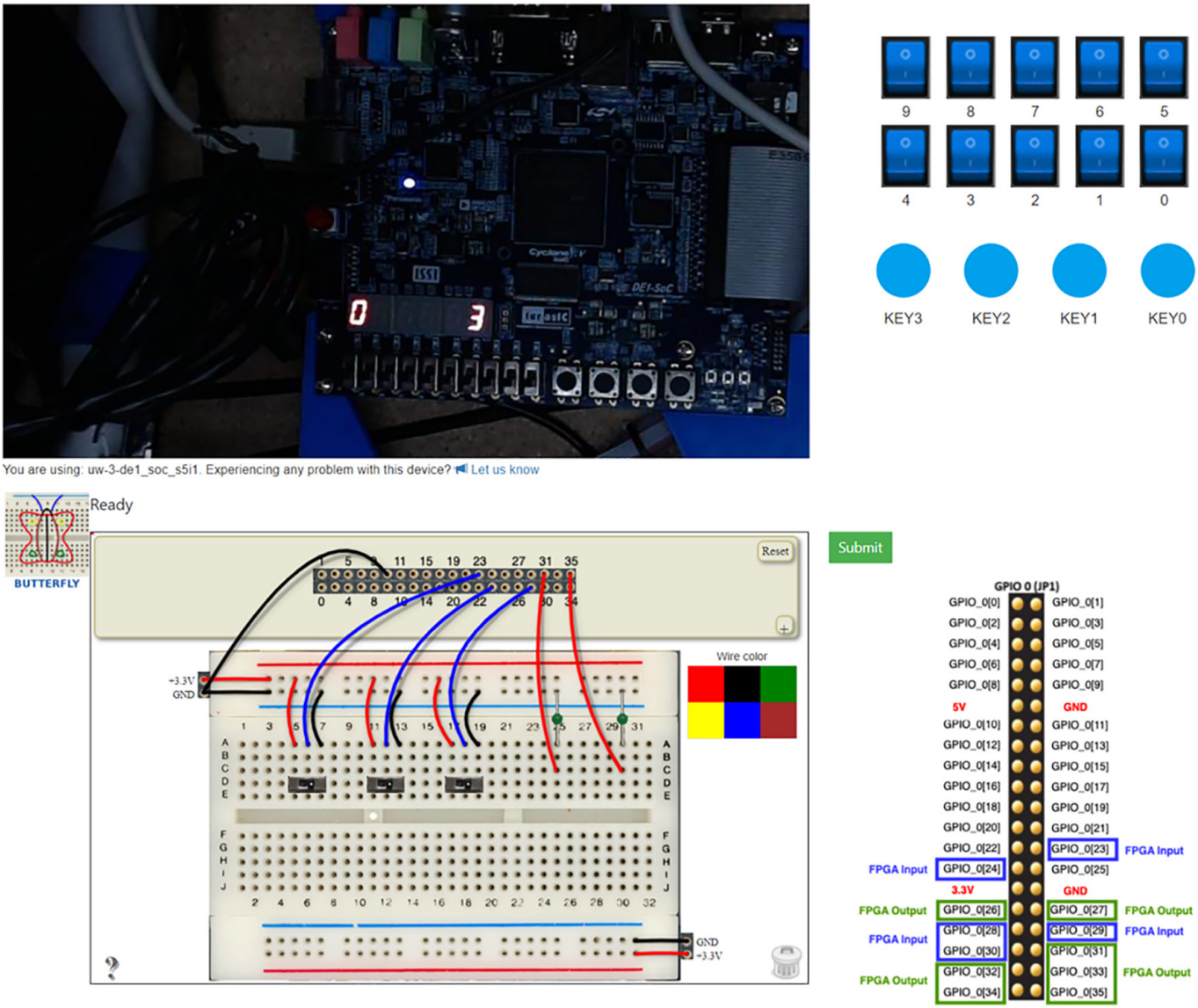

Fig. 1. The student interface of RHL-Butterfly with a real DE1_SoC (top) and the digital twin of a breadboard (bottom)

2 MATERIALS AND METHODS

The RHL-Butterfly facilitates seamless communication between a digital twin of a breadboard and physical target hardware. The digital twin needs to be both electrically accurate and have fast communication with the hardware. This blend ensures that the product design is valid and offers a viable remote substitute for traditional laboratory breadboards. These key metrics underpin the user experience in remote laboratories, as discrepancies with real-world hardware can distract users during experimentation, ultimately diminishing the effectiveness of remote labs and users' learning outcomes.

The virtual breadboard experience and system-level communication are structured into three distinct phases: a frontend phase, crafted in JavaScript, offering users an interactive interface for adding and manipulating components. The Raspberry Pi Pico backend phase processes the digital logic states derived from the user's breadboard design. Lastly, the target hardware phase involves the microcontroller or FPGA that students interface with the digital twin of a breadboard for their coursework. As depicted in Figure 2, the target hardware's response is captured via livestream and transmitted back to the frontend through WILSP [27], ensuring a remote laboratory experience with minimal latency.

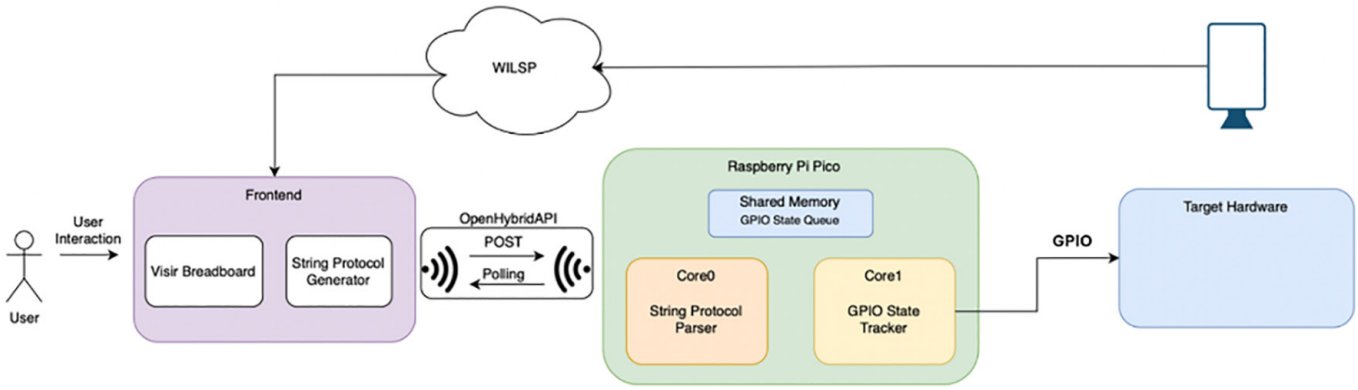

Fig. 2. Top-level block diagram of RHL-Butterfly virtualized breadboard infrastructure

2.1 RHL-Butterfly frontend components

The digital twin of a breadboard builds upon the VISIR HTML5 Client that allows wiring and measuring electronic circuits through a virtual platform [28]–[30]. Users of this platform are presented with a virtual image of a solderless breadboard with the ability to draw wires freely on the breadboard through a clickable mouse drag.

The VISIR virtual breadboard framework supports six different wire colors, allowing users to organize their circuits in a way that maximizes clarity and readability for them. Upon selecting a wire color, the mouse cursor transforms into a drawing tool, enabling users to trace a wire from a starting point to an endpoint while maintaining the mouse button pressed. Releasing the button finalizes the wire's placement, securing it in position on the breadboard.

RHL-Butterfly supports a range of fundamental components frequently encountered in introductory to intermediate digital logic and embedded systems courses, including single input logic gates (e.g., NOT gates), dual input logic gates (e.g., AND, OR, XOR gates), dual-state switches, and light-emitting diodes (LEDs).

1. Single input logic gates: A NOT gate is the only basic logic gate component with a single input, with the basic functionality of inverting incoming digital signals. Typical Dual Inline Package (DIP) NOT gate Integrated Circuits (ICs) purchased from online vendors and electrical component stores consist of multiple NOT gates. An example is the 7404 NOT Gate Package, consisting of 14 pins and six unique NOT gates, as shown in Figure 3. The input supplied voltage pin, or Vcc, is typically labeled as pin 14, while the associated electrical ground (GND) pin is typically pin number 7. Pins 1, 3, 5, 9, 11, and 13 are associated inputs to pins 2, 4, 6, 8, 10, and 12, respectively.

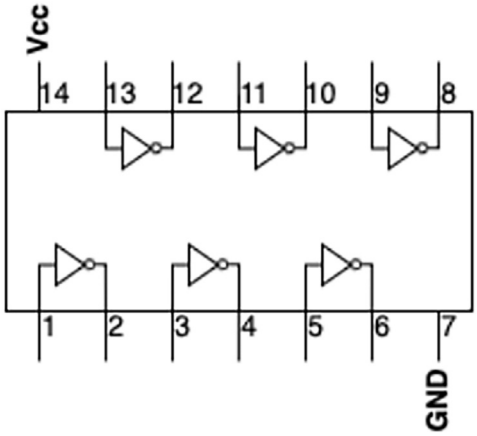

Fig. 3. NOT gate 14-pin 7404 topology

The digital twin equivalent of the 7404 NOT gate component was created using an image of the unique 14-pin IC, labeled with a "NOT" on top for easy identification by users, as depicted in Figure 4.

Fig. 4. Digital twin representation of a NOT gate

The NOT gate is handled through a JavaScript object, nested within a Breadboard object, which is the object container for the overall virtual breadboard. To ensure the functionality that users can click and drag the NOT gate component anywhere on the breadboard digital twin, mimicking the freedom of integrated circuit placement found in a physical breadboard, the NOT gate JavaScript object stores the current left position (this._leftPosition) and top position (this._topPosition) pixel values of each created NOT gate. Whenever the user drags the component to a different location, a setPinLocation() function is automatically called to update the stored pixel values.

2. Dual input logic gates: Aside from a NOT gate, the other three fundamental logic gate building blocks are dual inputs, requiring two inputs to compute the corresponding output.

   Similar to that of a NOT gate IC, a 14-pin dual input logic gate DIP package typically uses pin 14 for Vcc and pin 7 for GND. The rest of the pins, organized in groups of three, allow for four logic gates, each with two inputs and one output.

   Despite differences in logic gate functionality, a 7408 Quad 2-Input AND gate, a 7432 Quad 2-Input OR gate, and a 7486 Quad 2-Input XOR gate have the same associated pin assignments: input 1 is associated with pins 1, 4, 10, and 13; input 2 is associated with 2, 5, 9, and 12; and the outputs are associated with pins 3, 6, 8, and 11, respectively, as shown in Figure 5. Figure 6 shows the digital twins of AND, OR, and XOR gates.

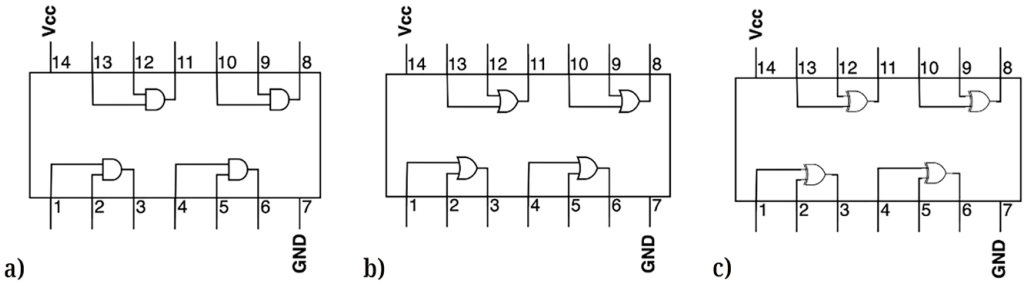

Fig. 5. a) AND gate 14-pin 7408 topology (left), b) OR gate 14-pin 7432 topology (middle), and c) XOR gate 14-pin 7486 topology (right)

Fig. 6. Digital twin representations of logic gates: a) AND gate (left), b) OR gate (middle), and c) XOR gate (right)

Given their shared pin assignments, the functionalities of JavaScript objects representing the dual input logic gates are similar and are inherited from a broader QuadDualInputGate object. To enable drag-and-drop placement on a virtual breadboard, the QuadDualInputGate object maintains internal variables for the current left and top pixel positions, which are updated via a prototype function named setPinLocation() whenever the component is moved.

3. Virtual switch: A Single Pole, Double Throw (SPDT) toggle switch controls the flow of electricity through a circuit by physically opening or closing a connection. Typically, one end is connected to a power source (Vcc), the other end to ground (GND), and the middle pin to the rest of the circuit. Toggling the switch connects the middle pin either to Vcc, representing a digital HIGH, or to GND, representing a digital LOW.

In a digital twin of a breadboard, users cannot physically manipulate an SPDT switch. Instead, a JavaScript switch component simulates its state using two images: one for the ON position and another for the OFF position, as shown in Figure 7. When a user clicks on the virtual switch, the JavaScript component immediately changes the image, thus mimicking the behavior of mechanically moving the switch state.

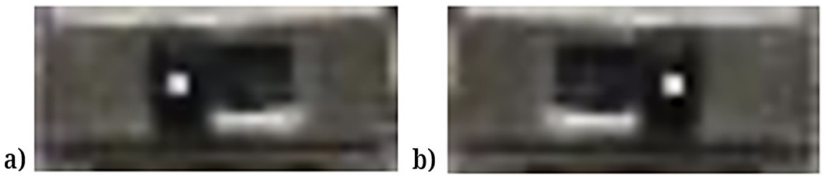

Fig. 7. a) Digital twin representation of an SPDT switch in the left position (left) and b) in the right position (right)

As with the other previously mentioned components, to ensure that users can drag and drop each switch digital twin to their desired location on the virtual breadboard, the virtual switch JavaScript object stores the left and top pixel value positions internally. These values are updated whenever the switch is moved.

However, unlike the logic gate components, the output pin digital logic state is computed in the client frontend, rather than through a microcontroller on the backend. Since the JavaScript must flip the virtual switch's image during a mouse-click event, the digital state is computed as part of this interaction.

4. Digital twin of an LED: A digital twin of an LED, like a virtual switch, simulates state changes with illuminated and unilluminated images. These images and LED functionality are stored in a JavaScript object. Unlike fixed-orientation switches and logic gates, virtual LEDs can be rotated 90 degrees to fit designs, as shown in Figure 8, enhancing the customizability and usability of the virtual breadboard.

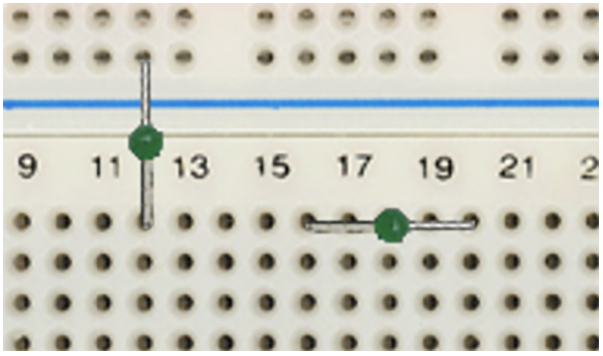

Fig. 8. Virtual LED placed virtually and horizontally

The virtual LED JavaScript object incorporates draggability by storing the top and left pixel positions of the image. When dragged, these positions are updated to reflect the new location. To support both vertical and horizontal orientations, the object includes a Boolean property, this._isVertical, which indicates the current orientation. Depending on this value, the top and left position updates differently.

2.2 RHL-Butterfly frontend protocol construction

The RHL-Butterfly offloads the digital twin of the breadboard state computation to a backend microcontroller to mimic the real-time behavior of an embedded system and save computation power on the client frontend. To process the current breadboard state and the necessary information for the backend microcontroller, a custom string protocol of that breadboard design is constructed and passed to a Raspberry Pi Pico microcontroller through a POST request method to the microcontroller server.

This custom string protocol is a minimalistic way of describing the user-designed circuit in as small of a string length as possible. This section describes the different elements of the virtual breadboard and its associated string representation.

1. GPIO header pins: The virtual breadboard can interface with a target microcontroller hardware using header pins that act as the GPIO of that target microcontroller. To simplify the protocol construction, certain GPIO pins are designated as virtual inputs (target hardware outputs), while others are designated as virtual outputs (target hardware inputs).

To provide information on whether a GPIO is connected, the protocol for a GPIO element on the virtual breadboard is a

$$\text{“}g\text{”} + \{gpio\_number\} \tag{1}$$

However, because the goal of the digital twin of the breadboard is to interface with as many different microcontrollers and Field Programmable Gate Arrays (FPGAs) as possible, each with different GPIO header sizes and pin mappings, the {*gpio_number*} is normalized to indicate not the exact header pin number, but rather in relation to the first GPIO input or the first GPIO output. For example, if a target microcontroller GPIO contains 40 distinct pins, with the supported input pins starting at pin 23 and the next supported input pin at pin 24, rather than constructing the intuitive but less scalable “g23” and “g24,” the protocol is instead constructed to be “g00” and “g01.” An example is given in Table 1.

Table 1. GPIO header protocol construction

Protocol Construction	Example
g{gpio_number}	g00

2. Power plane: Vcc and GND are two supported power planes of the virtual breadboard. If part of a design is connected to a power plane, the protocol is constructed as

$$\text{“}L\text{”} + \{logic\_level\} \tag{2}$$

where the logic level is either True (T) or False (F), as shown in Table 2.

Table 2. Power plane protocol construction

Protocol Construction	Example
L{logic_level}	Vcc: LT
	GND: LF

3. Switches: Digital twins of switches are designed similarly as a power plane. Because the frontend JavaScript keeps track of the virtual switch ON and OFF position images to mimic a mechanical switch when the user clicks on the switch image, the state of the virtual switch is already computed; thus, only the already computed state can be sent to the backend microcontroller.
The string protocol constructed is a

$$\text{“}S\text{”} + \{logic\_level\} \tag{3}$$

where the logic level is either True (T) or False (F), as shown in Table 3.

Table 3. Switch protocol construction

Protocol Construction	Example
S{logic_level}	HIGH: ST
	LOW: SF

4. LEDs: The digital twins of LEDs on the breadboard can be connected to output GPIOs only. The backend microcontroller computes the digital logic state of all

virtual LEDs on the frontend UI and sends information to either display an illuminated or an unilluminated LED image. The client frontend constructs a string protocol to provide information on what the LED is connected to, so the microcontroller can then properly compute the LED state.

When an LED is connected to a circuit on the virtual breadboard, the protocol constructed is

$$\text{“}d\text{”} + \{led\_number\} \tag{4}$$

Table 4 shows a string representation of LED 0.

Table 4. LED protocol construction

Protocol Construction	Example
d{led_number}	d0

5. Logic gates: Different logic gates contain different inputs and outputs. For a single input NOT gate, the string protocol constructed is

$$\text{“}n\text{”} + \{input\} + \{output\} \tag{5}$$

where the input is a GPIO header protocol, a power plane protocol, or a virtual switch protocol, and the output is a GPIO header protocol or a virtual LED protocol. While a 7404 NOT gate integrated circuit contains many logic gates in the 14-pin package, the information as to which exact logic gate number in the package is of little relevant information for the backend microcontroller and, as such, is abstracted out, as shown in Table 5.

Table 5. Single input logic gate protocol construction

Component	Protocol Construction	Example
NOT gate	n{input}{output}	nSTg01

For dual input logic gates (i.e., AND gates, OR gates, and XOR gates), the protocol constructed is a

$$\{gate\_identifier\} + \{input1\} + \{input2\} + \{output\} \tag{6}$$

Table 6 provides examples of string representations of AND, OR, and XOR gates.

Table 6. Dual input logic gate protocol construction

Component	Protocol Construction	Example
AND gate	a{input1}{input2}{output}	aSTLTg01
OR gate	o{input1}{input2}{output}	oSTLTd0
XOR gate	x{input1}{input2}{output}	xSFLFg02

6. No logic gates: A valid breadboard design for RHL-Butterfly does not have to contain a logic gate. For example, a wire can be constructed that simply connects a virtual switch to a virtual LED. When this is the case, the protocol constructed is

$$\text{“}y\text{”} + \{input\} + \{output\} \tag{7}$$

When the backend microcontroller receives a protocol containing a "y," it will immediately forward the logic state of the input to the output, as depicted in Table 7.

Table 7. No logic gate protocol construction

Protocol Construction	Example
y{input}{output}	ySTd0

7. Buffers: Occasionally, a wire will be drawn from an output of a logic gate to an input of another gate. This is labeled as a buffer wire, as shown in Figure 9.

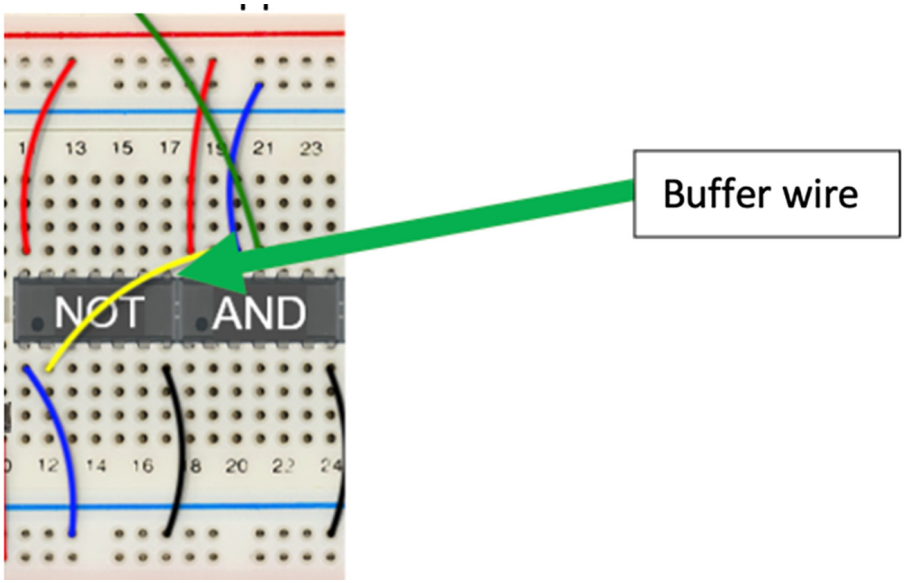

Fig. 9. Digital twin of a breadboard buffer wire

When the protocol string is constructed, the frontend does not know the current digital state of the buffer wire. The backend microcontroller handles the computation of this electrical node when computing the digital state of the entire breadboard circuit. The frontend interprets that there is a need for a buffer wire and will construct the string protocol as

$$\text{“}b\text{”} + \{buffer\_number\} \tag{8}$$

Table 8 shows the string representation of Buffer 0.

Table 8. Buffer protocol construction

Protocol Construction	Example
b{buffer_number}	b0

8. Multiple outputs: Valid breadboard designs can have multiple output wirings originating from a single output source. For instance, the output of a NOT gate can be distributed to multiple GPIO input pins. In such scenarios, the client frontend separates each output with a comma ",", as shown in Table 9.

Table 9. Multiple output protocol construction

Protocol Construction	Example
{output1},{output2},...	nSTg00,g01

9. Chaining strings together: To construct the string protocol that details the entire current breadboard state, the JavaScript sweeps through all user-drawn wires (2 endpoints each). In a valid design, all virtual inputs must be connected to virtual outputs. Breadboard inputs and outputs are listed in Table 10.

Table 10. Virtual breadboard inputs and outputs

Breadboard Inputs	Breadboard Outputs
Input GPIO pin	Output GPIO pin
Logic Gate Input	Logic Gate Output
LED	Vcc/GND
Buffer Wire	Buffer Wire
	Switch

For each wire, the frontend verifies both endpoints: one must be connected to an input and the other to an output. If this condition is met, the string protocol construction proceeds for microcontroller transmission. Any wire not adhering to this rule triggers an electrical error prompt, instructing the user to correct the design.

The breadboard-inspired GUI seamlessly updates its user-interaction state upon each mouse click, mirroring the intuitive experience of prototyping with physical breadboards. When power is supplied to a physical breadboard design, electrical current immediately flows through the elements when a component is placed in a valid location.

To maintain a safe and methodical breadboard prototyping approach, users are first encouraged to design and validate their circuit before virtually "powering" it. The breadboard frontend enforces this by requiring users to press a "submit" button on the webpage, signifying readiness for electrical testing. Upon successful validation and button press, the generated string protocol is sent asynchronously to the Raspberry Pi Pico microcontroller for digital processing.

2.3 Breadboard digital twin backend

The Raspberry Pi Pico, shown in Figure 10, serves as the core interpretation microcontroller for breadboard computations, directly translating GPIO data to target hardware. Its selection stems from its high clock speed (up to 133 MHz, standard at 125 MHz), multi-core capability for parallel processing, and cost-effectiveness [31]. The PicoSDK's extensive libraries facilitate experimentation, scalability, and future development of microcontroller peripherals [31]. Additionally, the Pico integrates seamlessly with the Raspberry Pi ecosystem, enabling web-based network communication.

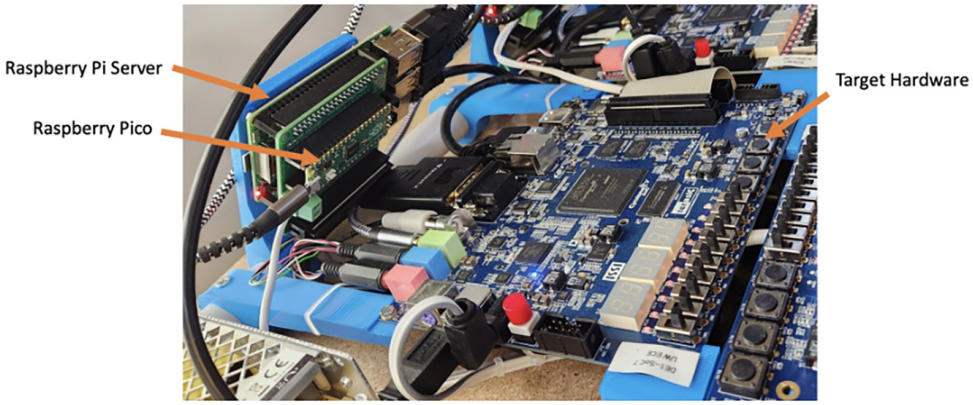

Fig. 10. RHL-Butterfly hardware perspective using, in this case, a LabsLand FPGA remote laboratory for DE1-SoC FPGA development board

The Raspberry Pi Pico features a dual-core processor, enabling multiple thread execution, as Figure 11 illustrates. The default program runs on Core0. In the virtual breadboard backend, Core0 interprets the string protocol sent by the JavaScript frontend and computes the states of the GPIOs to be updated on the target hardware. For example, an output GPIO may need to change from a logic HIGH to a logic LOW state. This change is stored in a C++ vector array and sent to a Raspberry Pi Pico queue.

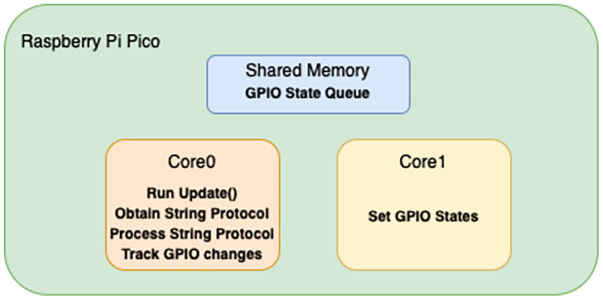

Fig. 11. Raspberry Pi Pico Core0 and Core1 functionality

Core0 and Core1 share the same memory space, allowing Core1 access to the queue updated by Core0. Core1 uses the C++ vector array to communicate the changes to the target hardware, updating the GPIO state. This division of work between the cores reduces the load on Core0, ensuring more efficient processing by sharing tasks between the two cores.

1. Protocol interpretation: The Raspberry Pi Pico uses shared memory to track input GPIO states, output GPIO states, buffers, and virtual LED states. For scalability and future expandability, the JavaScript string protocol does not use specific GPIO pin numbers but implements a counter based on the first available input or output GPIO. This counter system also tracks buffers and LED states, making it advantageous to use an array or vector. The counter from the JavaScript frontend protocol acts as the index for the array or vector, simplifying random access to different array addresses when changing GPIO states.

   The JavaScript frontend client does not need to know the GPIO states after the Raspberry Pi Pico finishes its computation, as this information is unnecessary for processing changes in the frontend user interface. Instead, the GPIO state information is sent directly to the target hardware to update the physical GPIO state. Similarly, neither the JavaScript frontend nor the target hardware needs to know

the intermediate buffer states, which are only used to aid in computing the GPIO and virtual LED states. These buffer states are stored in a vector in shared memory and are not passed elsewhere. However, the JavaScript frontend does need to know the LED states to update the digital twins of the LED images to reflect their state accurately. Figure 12 gives a top block diagram of the role of Raspberry Pico.

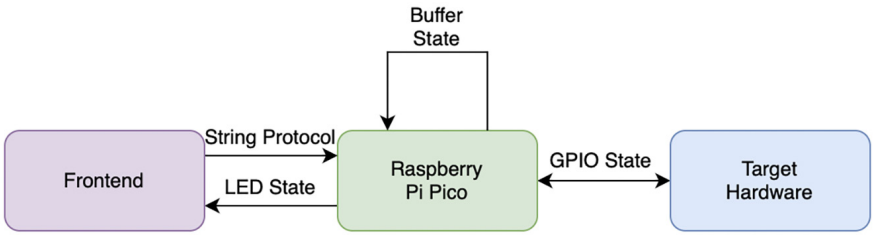

Fig. 12. Raspberry Pi Pico information transfer

The update() function in Core0 initiates the interpretation of the JavaScript string protocol. If no changes are made in the frontend (i.e., the JavaScript string protocol remains unchanged), Core0 will process and interpret the last received protocol. Error checking for the breadboard is handled exclusively in the JavaScript frontend, ensuring that only valid protocols are sent to the Raspberry Pi Pico. Therefore, the C++ firmware on the Raspberry Pi Pico assumes any received JavaScript string protocol to be a valid breadboard state.

The update() function sweeps through the characters in the JavaScript string protocol. Depending on the characters encountered, the code determines the number of inputs and outputs within each protocol section. For instance, if the protocol describes a NOT gate, the code handles one input; for an AND gate, it manages two inputs accordingly. The string protocol character and number of inputs and outputs of various logic gates are listed in Table 11.

Table 11. String protocol interpretation for logic gates

String Protocol Character	Breadboard Associated Description	Number of Inputs	Number of Outputs
“n”	NOT gate	1	1
“a”	AND gate	2	1
“o”	OR gate	2	1
“x”	XOR gate	2	1
“y”	No gate	1	1

The C++ code parses subsequent characters within each input and output section, adjusting based on the characters encountered. The string representation and number of inputs of various components are displayed in Table 12.

Table 12. String protocol interpretation for breadboard inputs and outputs

String Protocol Character	Breadboard Associated Description	Number of Inputs
“L”	Power or Ground Plane	1
“S”	Switch	1
“g”	GPIO Header	2
“b”	Buffer Wire	1
“d”	LED	1

If an "L" is read, the code reads the next character, which will either be a T or an F, to indicate a True or a False. The C++ code will store that Boolean value in a Boolean variable. If a "S" is read, the C++ code behavior is very similar to that of a power or GND plane; the code reads the next character, which will either be a T or an F, to indicate a True or a False. The C++ code will store that Boolean value in a Boolean variable. If a "g" is read, the C++ code will look at the next two characters that indicate the index for the GPIO vector array. The C++ code typecasts those next two characters into an integer and uses that as the index value to read or store new GPIO state Boolean values. If a "b" is read, the C++ code will look at the next value that indicates the index for the buffer vector array. Like the GPIO vector array, the C++ code typecasts the character into an integer and uses that as the index value to be read or store new buffer state Boolean values. If a "d" is read, the code behavior is again similar to that of a buffer; the C++ code will look at the next character value, which describes the index value of the vector array, to either read or store new virtual LED Boolean states. Depending on the gate that was read prior (NOT gate, AND gate, etc.), the C++ code will then compute the output state based on what the input variable(s) are. The code will then store them in the proper output location, based on the string protocol.

If the C++ detects a change in the digital twin of the LED states, which are captured and stored in the LED state vector, the C++ will then concatenate each LED state into a string to be sent back to the JavaScript frontend as a sim2web polling message: "led{led_number}={led_state}&led{led_number}={led_state}...," as shown in Table 13.

Table 13. String protocol construction for LED states

Protocol Construction	Example
"led{led_number}={led_state}&led{led_number}={led_state}..."	"led0=1&led1=0&led2=0&led3=0&led4=0"

When the JavaScript receives this sim2web message, the JavaScript deconstructs the LED state. Any LED state that is a logic HIGH will change the virtual LED image to show one that is illuminated.

2. Backend protocol scalability: A primary goal of the RHL-Butterfly is to scale its implementation to support a variety of target hardware by using OpenHybridAPI[1], which is a part of the REDTAIL (NSF #2336745) project. In this project, more simulations and virtual devices will be built and integrated into our remote laboratory. While the initial supported hardware is the Intel DE1-SoC FPGA, the backend Raspberry Pi Pico microcontroller's protocol interpretation is designed for seamless expansion to accommodate other target hardware, such as STM32 microcontrollers.

The DE1-SoC FPGA features two GPIO ports (GPIO_0 and GPIO_1) with expansion headers totaling 40 pins, as Figure 13 shows, allowing peripherals to be connected to the board.

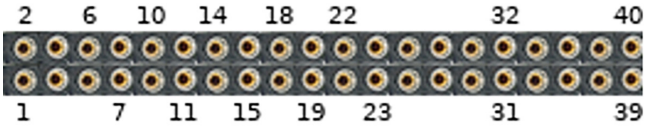

Fig. 13. Virtual breadboard 40-pin GPIO header

1 https://github.com/labsland/openhybridapi

The C++ code parses subsequent characters within each input and output to determine how many to process next.

Different FPGAs and microcontrollers used in target hardware offer varying numbers of GPIO pins on their expansion headers, affecting the availability of input and output GPIO pins for interfacing with virtual peripherals like the virtual breadboard.

Despite these hardware differences, the underlying C++ functionality remains consistent: it processes the same string protocol from the JavaScript frontend to manage GPIO states, buffer states, and virtual LED states using C++ vectors. This uniform approach enables expandability within the C++ codebase across various target hardware configurations.

The C++ update() function subsides within a Breadboard class called ButterflySimulation to allow for code reusability on different projects, as Figure 14 illustrates. A secondary class, DE1SoC_BufferflySimulation, inherits the ButterflySimulation parent class. The child class contains solely two functions: a getNumberOfSimulationInputs, which returns the number of available GPIO inputs based on that target hardware, and a getNumberOfSimulationOutputs, which returns the number of available GPIO outputs based on that target hardware. All other functions are inherited from the parent class. The number of simulation inputs and simulation outputs dictate the vector array for the output GPIO tracker and input GPIO tracker, which maps to the target hardware physical GPIO. As more target hardware becomes supported with the digital twin of the breadboard, additional child classes will be added that inherit BufferflySimulation, each returning a different number from getNumberOfSimulationInputs and getNumberOfSimulationOutputs.

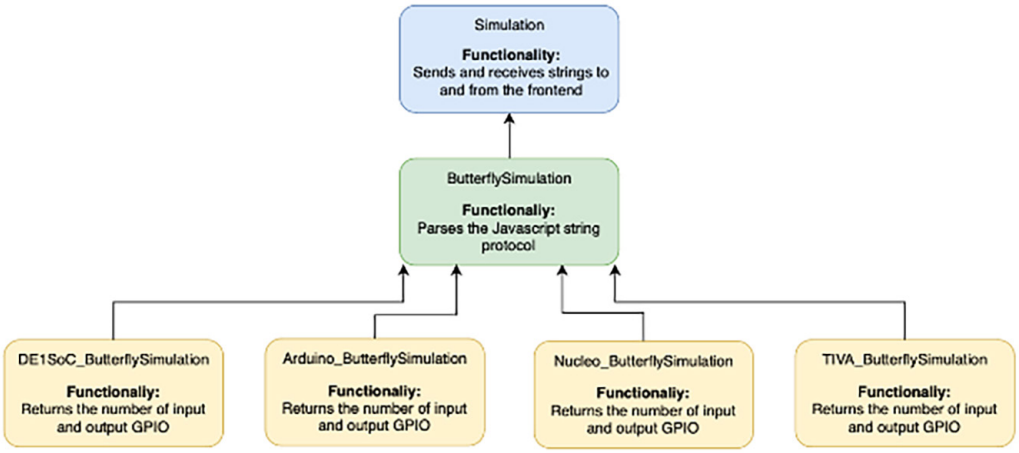

Fig. 14. Raspberry Pi Pico class inheritance

3 RESULTS

3.1 Example

Figure 15 depicts a digital circuit example utilizing two digital twins of SPDT switches to control the input logic levels of various logic gates. An OR gate has one input connected to a switch and the other input connected to GND. The OR gate's output serves as input to an AND gate, where the second input is connected to another switch. The output of the AND gate then connects to an input of a NOT gate, whose output feeds into another NOT gate. Finally, the output of the second NOT gate is linked to both a virtual LED and an output general-purpose input/output (GPIO).

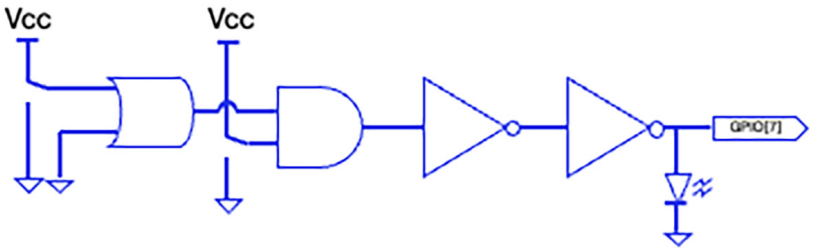

Fig. 15. Example digital circuit

After the circuit is constructed on the RHL-Butterfly breadboard, as shown in Figure 16, the JavaScript string protocol becomes "nb1b2;nb2g00,d0;ab0STb1; oSTLFb0;\n."

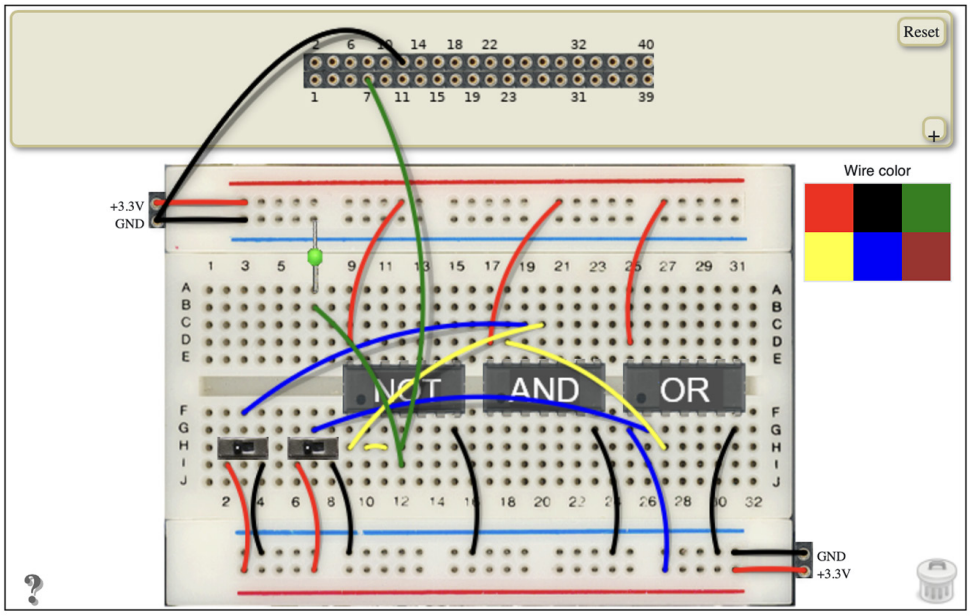

Fig. 16. Virtual breadboard construction for the example digital circuit

In this circuit, three intermediate buffers are employed. During each update call, the Raspberry Pi Pico computes and stores the buffer states in a vector array in shared memory, initially set to 0. In subsequent iterations, these stored buffer states are used to update and recompute each buffer's digital logic states, ensuring they reflect the current circuit configuration, as depicted in Table 14.

Table 14. Buffer, LED, and GPIO digital states for multiple update iterations

	Iter 1	Iter 2	Iter 3	Iter 4	Iter 5	Iter 6	Iter 7
GPIO 7	0	0	0	1	1	1	1
LED 0	0	0	0	1	1	1	1
Buffer 0	0	1	1	1	1	1	1
Buffer 1	0	0	1	1	1	1	1
Buffer 2	0	1	1	0	0	0	0
Buffer 3	0	0	0	0	0	0	0
Buffer 4	0	0	0	0	0	0	0

As the buffer states, GPIO outputs, and LED outputs remain volatile, the computation exhibits characteristics similar to metastability in real-world hardware. Eventually, after the 4th iteration of the update() call, both the buffer states and outputs stabilize into their final states for this circuit, as Figure 17 illustrates.

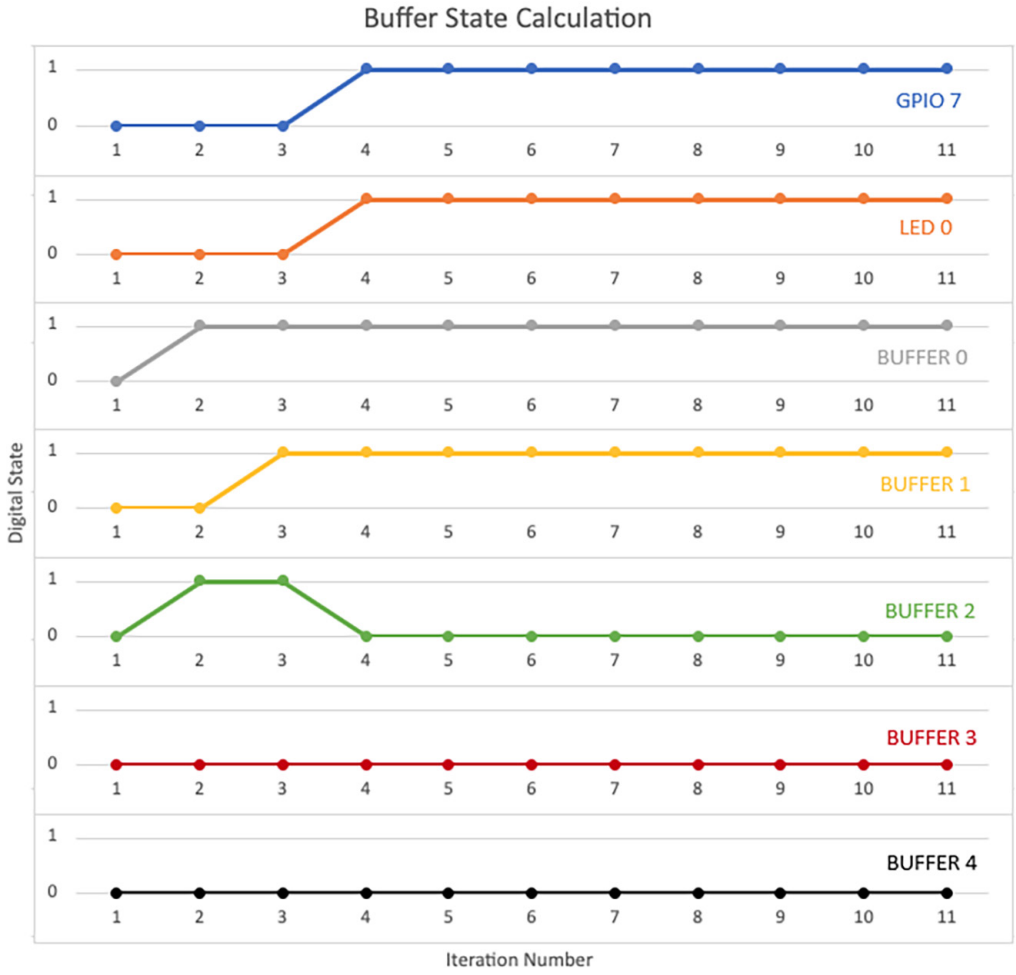

Fig. 17. Graphical representation of the digital states for multiple update iterations

3.2 Offering and usability survey

1. RHL-Butterfly deployment method: RHL-Butterfly was offered in two academic terms in a Design of Digital Circuits course at the University of Washington in 2023. Students used the new generation of breadboard digital twin to construct designs featuring Single Pole Double Throw (SPDT) breadboard switches and LEDs to interface with the GPIO of the DE1-SoC for their laboratory assignment. The objective of the laboratory assignment was to review students' knowledge on Mealy and Moore Finite State Machines (FSMs) by asking them to design a system that detects the entering and exiting of a car in a parking lot.
2. Survey construction: The goal of the survey was to evaluate student perspectives on the usability and viability of the digital twin of a breadboard learning

experience over a traditional physical breadboard. The survey was constructed using a combination of a Likert scale and long free response questions highlighting the overall virtual breadboard usability, remote nature, the interface to the target hardware's GPIO, and the usability of individual components.

With Likert scale questions, the questions were phrased in a positive way and had answer choices on a 5-point scale, where 1 indicated that the students strongly disagreed, a 2 indicated that they disagreed, a 3 indicated they felt neutral, a 4 indicated they agreed, and a 5 indicated that the students strongly agreed with the question statement.

3.3 Survey results

Following the completion of the course lab assignment involving using the digital twin of the breadboard, students were invited to participate in an anonymous survey to share their experiences with RHL-Butterfly for an exempt human research study. 110 unique responses were collected from 147 distinct groups in the two academic terms, yielding a response rate of approximately 75%. Throughout the bar graph figures below, blue bars represent responses obtained from Academic Term 1, while orange bars represent responses from Academic Term 2.

1. Overall experience: At the outset, students were requested to assess the intuitiveness of the breadboard digital twin on a 5-point Likert scale, with a rating of 5 indicating a highly intuitive experience and a rating of 1 reflecting a highly unintuitive experience. The responses in Figure 18a are right-skewed, showing that most students agreed with the statement that the digital twin of a breadboard was intuitive to use.

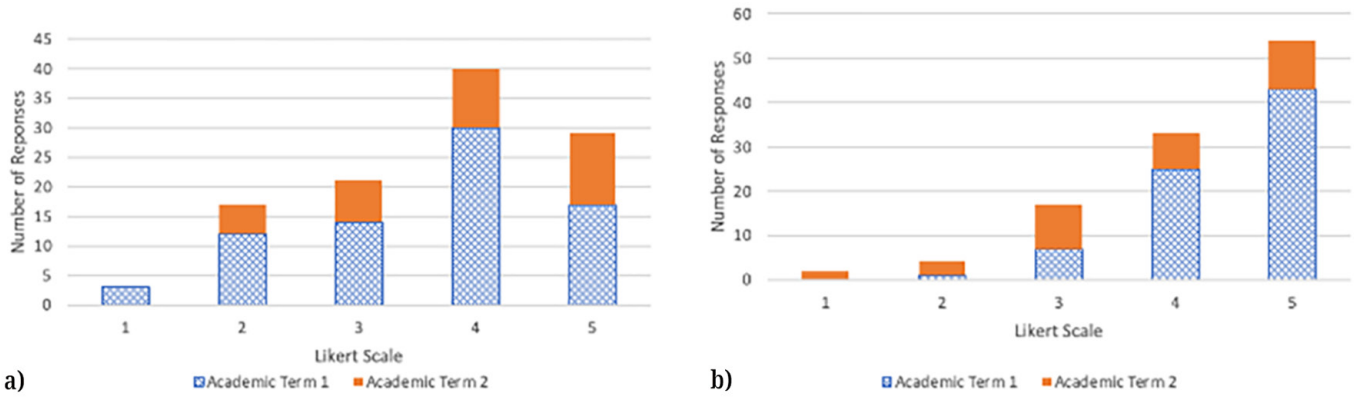

Fig. 18. Student agreement that a) the digital twin of the breadboard was intuitive to use (left), and b) the digital twin of the breadboard felt like that of a traditional physical breadboard (right)

The next question added a quantitative metric to show the similarities between a physical breadboard and the provided digital one, where students were asked to rank on a 5-point Likert scale how similar the breadboard digital twin experience was to their prior breadboard experience. The responses shown in Figure 18b show that many students agreed that their experience with using the digital twin was very similar to that of their prior experience with physical breadboards.

2. GPIO interface: The main goal of using the digital twin of a breadboard is to help students learn about interfacing with the DE1-SoC GPIO to connect external components and circuits to their FPGA designs. This section checks if the virtual GPIO interface with physical DE1-SoC provided a similar experience to that of a physical GPIO interface and if the digital twin of the breadboard implementation of that interface had aided in students' accessing the GPIO through their SystemVerilog implementation.

   Students were initially asked to rate whether interfacing with the DE1-SoC GPIO on the breadboard digital twin was like that of a physical breadboard and physical FPGA on a 5-point Likert scale, in Figure 19a, to which many students agreed, with the results skewed to the right. Students were then asked to rate on a 5-point Likert scale whether using the virtual GPIO had helped in their understanding of how to interface with DE1-SoC GPIOs, both through hardware connections and through their SystemVerilog implementation, indicated in Figure 19b. Most students responded with a 4 and 5, indicating they believed the breadboard digital twin had helped them understand interfacing with GPIOs.

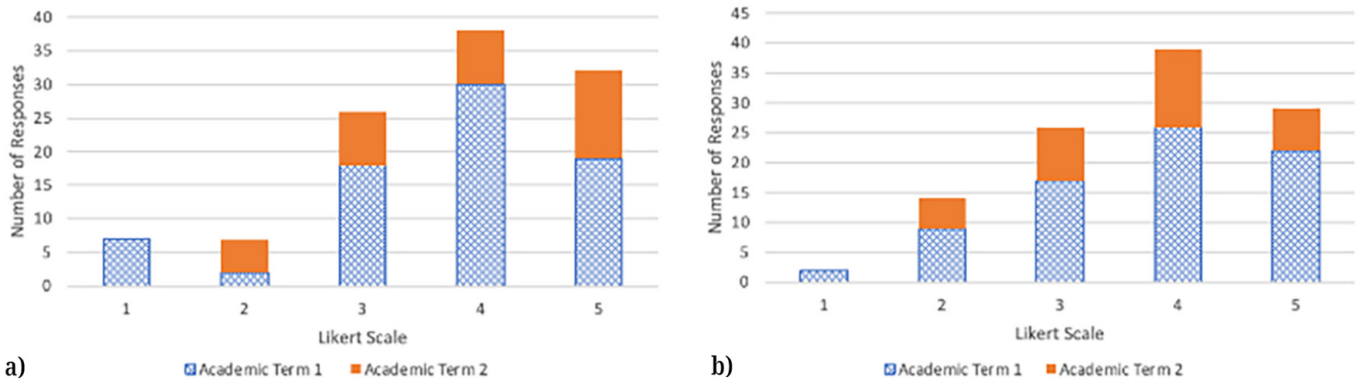

Fig. 19. Students' agreement to a) the digital twin of the breadboard helped in their understanding of interfacing with the FPGA GPIO (left), and b) the interfacing nature of the digital twin of the breadboard felt similar to that of a traditional physical GPIO interface (right)

3. Equitable nature: The next three questions address the equitable access and the virtual nature of the remote laboratory experience, specifically regarding the breadboard digital twin.

   The remote aspect survey theme was only asked in Academic Term 1, due to differences in teaching styles for the two different terms. The first term instruction offered a larger amount of location flexibility for students with available Zoom office hours from teaching assistants. In Academic Term 2, where most activities were conducted on-site or in-person, the emphasis on remote access was diminished, as most students did not utilize remote access technologies extensively. In such cases, including questions about remote access in the survey may yield limited insight or responses.

   The first question within this survey category asked students where they spent most of their time completing their laboratory assignment. More than 76% of students responded that they had spent the most amount of time at home, as shown in Figure 20.

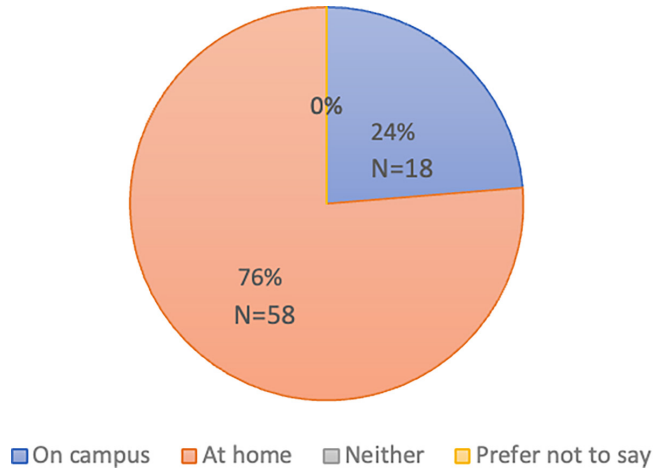

Fig. 20. Students' responses to where they spent the most time working on their lab assignment

Additionally, when asked to rate the convenience of accessing the digital twin of the breadboard online and allowed location flexibility, the majority of students responded with a "5," indicating that the location flexibility was strongly convenient, as shown in Figure 21a.

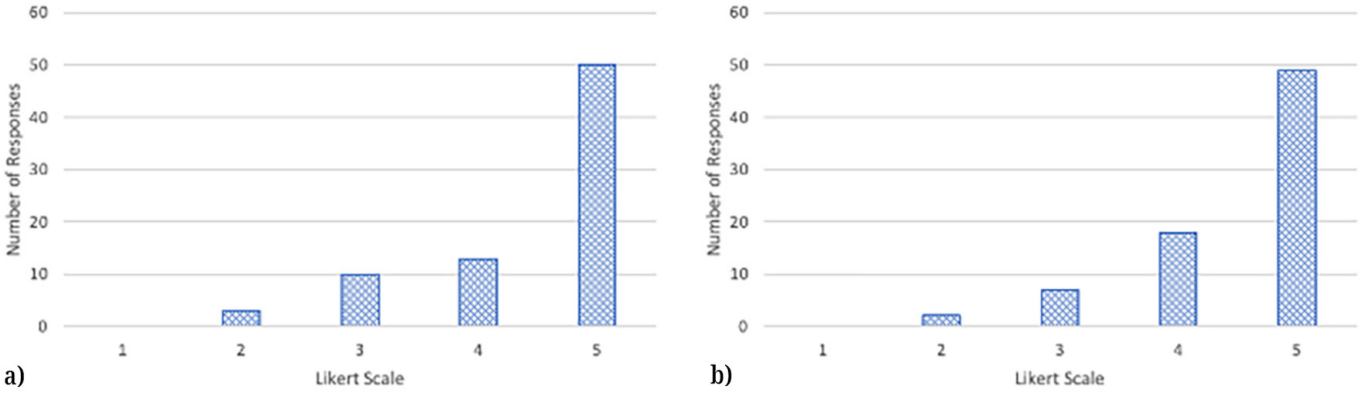

Fig. 21. Students' agreement to a) the virtual breadboard's location flexibility (left) and b) the virtual breadboard's time flexibility (right)

A similar question was then asked about the convenience of accessing the virtual breadboard at any time that fits their desired homework schedule, shown in Figure 21b. Again, the majority of students responded with a "5," indicating that the time flexibility was strongly convenient. Overall, students responded very positively to the notion of greater flexibility through remote laboratories and remote learning for breadboard digital twin experimentation.

4. Free response themes: Near the end of the survey, students were asked to share other helpful aspects of the digital twin of the breadboard as well as suggestions for future improvements in a free-response format.

The thematic analysis of student survey free-response questions revealed several common themes regarding what students found helpful in their experience with RHL-Butterfly. Firstly, students appreciated clear and informative error messages provided by the system when they made mistakes while using the digital twin. This feature helped them identify and rectify their errors, enhancing their learning process. Secondly, students appreciated the equitable nature of RHL-Butterfly. Students expressed satisfaction with the platform's ability to provide

an equal learning environment for all, irrespective of their physical resources or backgrounds. Lastly, the overall design of the breadboard was another significant theme identified. Students commended the user-friendly interface, intuitive layout, and ease of navigation. The thoughtful design of the digital twin of the breadboard contributed to a positive user experience, enabling students to focus on learning and experimentation.

Inferred themes from student suggestions on future improvements involve UI refinements, including a way to lock components, which can reduce the number of unintentional component drags. Additional suggestions for improvements involve creating more usable ways to enhance breadboard features, including shareability features for saving and loading XML files of breadboard designs for future use and support of analog circuit components.

4 CONCLUSION

Integrating remote laboratories and tools into the educational curriculum enhances equitable access to STEM education. The RHL-Butterfly brings a new approach to designing digital twins of breadboards by introducing an open-source, scalable custom network communication protocol, providing a practical solution for embedded systems and engineering education. This digital twin interfaces with a cluster of physical FPGAs and ARM microcontroller hardware by converting the 2D mapping of the breadboard state into a 1D string representing the electrical nodes and GPIO configuration. The hardware responds in real-time, with GPIO states captured via a livestream and displayed back to the frontend client. This approach allows RHL-Butterfly to integrate breadboard digital twins with real laboratory hardware, enabling users to see real-world and real-time responses to their designs. This system allows students to interact with physical hardware in a remote environment, resulting in positive student satisfaction and feedback.

This study addresses current limitations of equitable engineering education access due to varying student situations, including financial costs, seasonal illnesses, and disabilities, promoting STEM participation among underprivileged students. By bridging the gap between education and technology, RHL-Butterfly has become an integral part of the remote laboratory experience within the Remote Hub Lab [15, 32]. It plays a pivotal role in integrating Digital Logic Design curriculum into both college and pre-college STEM courses through the Remote Hub Lab's RHL-BEADLE project [33].

5 ACKNOWLEDGEMENTS

This study is supported by the National Science Foundation's Division of Undergraduate Education under Grant No. 2336745.

6 REFERENCES

[1] R. A. Abumalloh *et al.*, "The impact of coronavirus pandemic (COVID-19) on education: The role of virtual and remote laboratories in education," *Technology in Society*, vol. 67, p. 101728, 2021. https://doi.org/10.1016/j.techsoc.2021.101728

[2] S. Ray and S. Srivastava, "Virtualization of science education: A lesson from the COVID-19 pandemic," *J. Proteins Proteom.*, vol. 11, pp. 77–80, 2020. https://doi.org/10.1007/s42485-020-00038-7

[3] R. Mulya, K. Krismadinata, N. Jalinus, and H. Effendi, "Practical work of digital systems course based on virtual laboratory," *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 17, no. 8, pp. 22–37, 2021. https://doi.org/10.3991/ijoe.v17i08.23359

[4] L. Rodriguez-Gil, J. García-Zubia, P. Orduña, and D. López-de-Ipiña, "Towards new multi-platform hybrid online laboratory models," *IEEE Transactions on Learning Technologies*, vol. 10, no. 3, pp. 318–330, 2017. https://doi.org/10.1109/TLT.2016.2591953

[5] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, I. Angulo, and D. López-de-Ipiña, "Graphic technologies for virtual, remote and hybrid laboratories: WebLab-FPGA hybrid lab," in *2014 11th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, Porto, Portugal, 2014, pp. 163–166. https://doi.org/10.1109/REV.2014.6784245

[6] R. Heradio, L. de la Torre, D. Galan, F. J. Cabrerizo, E. Herrera-Viedma, and S. Dormido, "Virtual and remote labs in education: A bibliometric analysis," *Computers & Education*, vol. 98, pp. 14–38, 2016. https://doi.org/10.1016/j.compedu.2016.03.010

[7] A. K. Mohammed, H. M. El Zoghby, and M. M. Elmesalawy, "Remote controlled laboratory experiments for engineering education in the post-COVID-19 era: Concept and example," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt, 2020, pp. 629–634. https://doi.org/10.1109/NILES50944.2020.9257888

[8] M. Ogot, G. Elliott, and N. Glumac, "An assessment of in-person and remotely operated laboratories," *Journal of Engineering Education*, vol. 92, no. 1, pp. 57–64, 2003. https://doi.org/10.1002/j.2168-9830.2003.tb00738.x

[9] R. Hussein and D. Wilson, "Remote versus in-hand hardware laboratory in digital circuits courses," in *American Society for Engineering Education ASEE Conference, Electrical and Computer Engineering Division*, 2021, pp. 26–29.

[10] S. Li and R. Hussein, "Effectiveness of remote laboratories in promoting simulation and verification tools," in *Open Science in Engineering, REV 2023*, in Lecture Notes in Networks and Systems, M. E. Auer, R. Langmann, and T. Tsiatsos, Eds., Springer, Cham, vol. 763, 2023, pp. 87–95. https://doi.org/10.1007/978-3-031-42467-0_8

[11] F. Atienza and R. Hussein, "Student perspectives on remote hardware labs and equitable access in a post-pandemic era," in *2022 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, 2022, pp. 1–8. https://doi.org/10.1109/FIE56618.2022.9962440

[12] M. Inonan, A. Paul, D. May, and R. Hussein, "RHLab: Digital inequalities and equitable access in remote laboratories," *American Society for Engineering Education ASEE Conference*, 2023.

[13] Y. Zhao and J. Watterston, "The changes we need: Education post COVID-19," *J. Educ. Change*, vol. 22, pp. 3–12, 2021. https://doi.org/10.1007/s10833-021-09417-3

[14] M. Zheng, D. Bender, and C. Lyon, "Online learning during COVID-19 produced equivalent or better student course performance as compared with pre-pandemic: Empirical evidence from a school-wide comparative study," *BMC Med. Educ.*, vol. 21, 2021. https://doi.org/10.1186/s12909-021-02909-z

[15] R. Hussein *et al.*, "Remote Hub Lab – RHL: Broadly accessible technologies for education and telehealth," in *Open Science in Engineering, REV 2023,* in Lecture Notes in Networks and Systems, M. E. Auer, R. Langmann, and T. Tsiatsos, Eds., Springer, Cham, vol. 763, 2024, pp. 73–85. https://doi.org/10.1007/978-3-031-42467-0_7

[16] C. Hodges, S. Moore, B. Lockee, T. Trust, and A. Bond, "The difference between emergency remote teaching and online learning," *EDUCAUSE REVIEW*, 2020. [Online]. Available: https://er.educause.edu/articles/2020/3/the-difference-between-emergency-remote-teaching-and-online-learning

[17] W.-S. Soh, "Experiential learning through remote electrical engineering labs during the COVID-19 pandemic," in *2021 IEEE International Conference on Engineering, Technology & Education (TALE)*, Wuhan, Hubei Province, China, 2021, pp. 1–5. https://doi.org/10.1109/TALE52509.2021.9678756

[18] N. Glass, "Java digital breadboard simulator: A simulator for educational electronics environment," 2002. [Online]. Available: https://muchlas.ee.uad.ac.id/downloads/nicholas%20glass-jbreadboard.pdf

[19] Autodesk, "All you need is 'what if...'," Tinkercad. [Online]. Available: https://www.tinkercad.com/

[20] P. La Rocca, F. Riggi, and C. Pinto, "Remote teaching Arduino by means of an online simulator," *Physics Education*, vol. 55, no. 6, 2020. https://doi.org/10.1088/1361-6552/abaa21

[21] P. Orduña, L. Rodriguez-Gil, J. Garcia-Zubia, I. Angulo, U. Hernandez, and E. Azcuenaga, "LabsLand: A sharing economy platform to promote educational remote laboratories maintainability, sustainability and adoption," in *2016 IEEE Frontiers in Education Conference (FIE)*, Erie, PA, USA, 2016, pp. 1–6. https://doi.org/10.1109/FIE.2016.7757579

[22] S. Li, H. Wang, L. Rodriguez-Gil, P. Orduña, and R. Hussein, "FPGA meets breadboard: Integrating a virtual breadboard with real FPGA boards for remote access in digital design courses," in *Online Engineering and Society 4.0, REV 2021,* in Lecture Notes in Networks and Systems, M. E. Auer, K. R. Bhimavaram, and X. G. Yue, Eds., Springer, Cham, vol. 298, 2021, pp. 144–151. https://doi.org/10.1007/978-3-030-82529-4_15

[23] D. May, B. Reeves, M. Trudgen, and A. Alweshah, "The remote laboratory VISIR – Introducing online laboratory equipment in electrical engineering classes," in *2020 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, 2020, pp. 1–9. https://doi.org/10.1109/FIE44824.2020.9274121

[24] I. Gustavsson, J. Zackrisson, L. Håkansson, I. Claesson, and T. Lagö, "The VISIR project – an open-source software initiative for distributed online laboratories," *REV Conference 2007*, 2007.

[25] I. Evangelista *et al.*, "Science education at high school: A VISIR remote lab implementation," in *2017 4th Experiment@International Conference (exp.at'17)*, Faro, Portugal, 2017, pp. 13–17. https://doi.org/10.1109/EXPAT.2017.7984378

[26] M. Guo, R. Hussein, and P. Orduña, "RHL-Butterfly: A scalable IoT-Based breadboard prototype for embedded systems laboratories," in *2022 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, 2022, pp. 1–4. https://doi.org/10.1109/FIE56618.2022.9962495

[27] I. Gustavsson *et al.*, "Lab sessions in VISIR laboratories," in *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, Madrid, Spain, 2016, pp. 350–352. https://doi.org/10.1109/REV.2016.7444499

[28] F. García-Loro *et al.*, "Remote laboratory VISIR: Recent advances, initiatives, federation, limitations and future," in *2021 IEEE Global Engineering Education Conference (EDUCON)*, Vienna, Austria, 2021, pp. 1754–1757. https://doi.org/10.1109/EDUCON46332.2021.9453961

[29] L. Rodríguez-Gil, J. García-Zubia, P. Orduña, and D. Lopez-de-Ipiña, "An open and scalable web-based interactive live-streaming architecture: The WILSP platform," *IEEE Access*, vol. 5, pp. 9842–9856, 2017. https://doi.org/10.1109/ACCESS.2017.2710328

[30] M. Tawfik *et al.*, "Virtual instrument systems in reality (VISIR) for remote wiring and measurement of electronic circuits on breadboard," *IEEE Transactions on Learning Technologies*, vol. 6, no. 1, pp. 60–72, 2013. https://doi.org/10.1109/TLT.2012.20

[31] C. Bell, "Introducing the Raspberry Pi Pico," *In: Beginning MicroPython with the Raspberry Pi Pico*, *Maker Innovations Series*, Apress, Berkeley, CA, pp. 1–42, 2022. https://doi.org/10.1007/978-1-4842-8135-2_1

[32] R. Hussein, M. Guo, P. Amarante, L. Rodriguez-Gil, and P. Orduña, "Digital twinning and remote engineering for immersive embedded systems education," in *2023 IEEE Frontiers in Education Conference (FIE)*, College Station, TX, USA, 2023, pp. 1–8. https://doi.org/10.1109/FIE58773.2023.10343436

[33] R. Hussein, R. Maloney, P. Orduna, J. Ander Beroz, and L. Rodriguez-Gil, "RHL-BEADLE: Bringing equitable access to digital logic design in engineering education," *American Society for Engineering Education ASEE Conference*, 2023.

7 AUTHORS

Matthew Guo is a former Master's student and researcher in the Remote Hub Lab (RHLab), where his research focused on embedded systems and IoT solutions to enhance equitable access in engineering education. The work presented in this paper is based on his Master's thesis, which was recognized with the Distinguished Thesis Award from the University of Washington in 2023.

Zhiyun Zhang is a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Washington. He is a member of the Remote Hub Lab (RHLab), where his research focuses on digital twinning, Software-Defined Radios (SDRs), and embedded systems. His current work explores the application of digital twin technology to enhance the functionality and accessibility of remote laboratories.

Dr. Pablo Orduña is co-founder and CEO of LabsLand, a global network of remote laboratories. He has participated in multiple US (NSF), EU (H2020, FP7, ERASMUS+), and industry projects in the field of remote labs, and is co-author of over 150 peer-reviewed articles, most of them in this field.

Dr. Rania Hussein is an Associate Teaching Professor in the Department of Electrical & Computer Engineering at the University of Washington and the Founder and Director of the Remote Hub Lab (RHLab). Her research focuses on embedded systems, remote experimentation, and digital twinning for engineering education and healthcare. She is the Principal Investigator (PI) of the work presented in this paper. Dr. Hussein has received multiple awards and honors for her contributions to engineering education, research, and mentorship. Her leadership in developing innovative and accessible engineering education solutions has been recognized by national and international academic and professional organizations (E-mail: rhussein@uw.edu).