Co-funded by the European Union

SREE platform - demonstrator, pilot, prototype

Deliverable D3.2

Volodymyr Kazymyr
CPNU

Version 1.1
29.11.2024

Co-funded by the European Union

Record of changes

Version #	Status of document	Date	Authors	Comments
0.1	Draft version	15.11.2024	V. Kazymyr	Draft version
1.0	Full version	24.11.2024	V. Kazymyr	Consolidated version
1.1	Final version	29.11.2024	V. Kazymyr	QA version

Quality assurance

No.	Date	Version	Approved by
1	29.11.24	1.1	Robert Madalin Chivu

CONTENTS

1 Summary of the Report

In our time, when the countries are under the epidemic and wars, it is very important for educational institutions to provide their students with the opportunity to fully continue their work remotely. All specialties, the study of which requires the presence of certain physical equipment, encountered problems, as teachers need to find ways to provide students with all the necessary equipment for study. Those specialties that study hardware development, in particular digital circuitry using programmable logic integrated circuits and microprocessors, encountered such problem. Therefore, the question of creating such an application that will allow students to perform laboratory work online, for which they only need access to the Internet, becomes crucial.
The main objective of the project is to address the regional and specific priorities and neds of Moldavian and Ukrainian partners and to support the further reformation of Ukrainian and Moldavian HEIs according to the CBHE call priorities and ET2020 strategy of European Union.

Development of SREE within the DIGITRANS project is one of the project priorities, which is related to the tasks of WP3. ***The aim of this Report is to demonstrate a pilot of the Sharing Remote Experiment Environment (SREE) platform and to describe the prototype of the platform.***

Objectives of SREE:

- Developing the Sharing Remote Experiment Environment (SREE) platform for on-line laboratory works with physical equipment of remote laboratories for learning and teaching practical topics in computer and electronic engineering.
- Integrating SREE with Sharing Modelling and Simulation Environment (SMSE) that affords virtual laboratories based on open software kernels using Jupiter Notebooks, for resulting acquisition and piloting of Digital Learning Ecosystem (DLE).
- Creating methodology of implementing and sharing remote applications of the HEIs laboratories' equipment and software tools for distance usage in framework of DLE based on application of ICT tools.

Component Limitation: purchased hardware for two laboratories, open-source software.

2 SREE prototype overview

2.1 Architecture

The general idea is to provide laboratory work using FPGA and MCU boards remotely, namely to test the operation of digital circuits implemented on FPGA or MCU, but the user only needs a PC with Internet access for this. The SREE architecture designed for this, is shown in Figure 1.

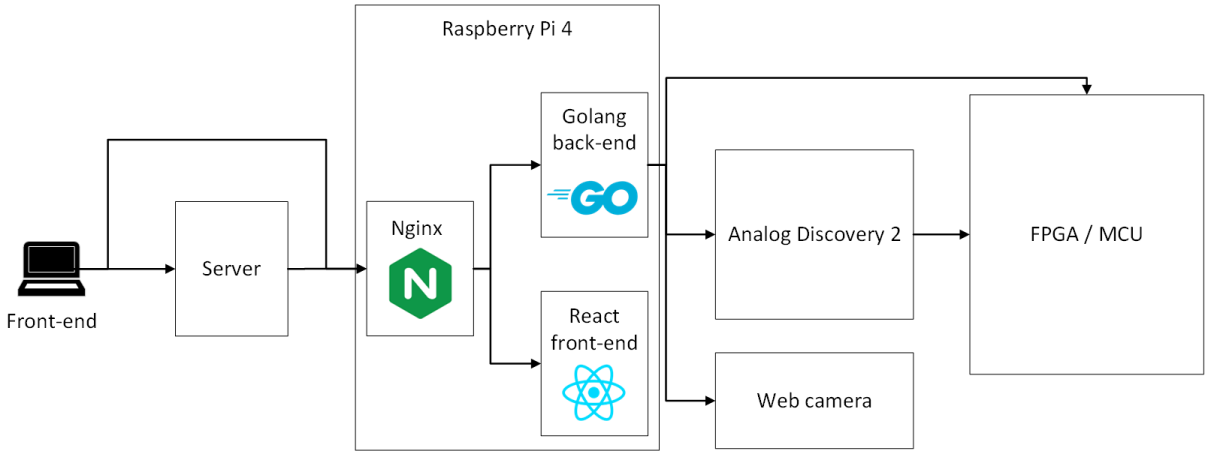


Figure 1 Architecture of SREE base platform

The main components of SREE include:

- Front-end: Client PC with web interface.
- Server that is responsible for user and sessions management.
- Raspberry Pi that is responsible for the laboratory work workflow and interaction with the hardware.
- Analog Discovery 2 digital oscilloscope used for buttons emulation and measuring signals on the board.
- Web camera that is processed by the back-end and is used to show the Client the real-time video of the hardware.

The Raspberry Pi runs Nginx, Golang back-end and React front-end.
The Nginx is used in the system as a webserver for the client-side (front-end) software serving and as a reverse-proxy for the back-end to create a single point of entry for easier network and CORS configuration.
The Raspberry Pi software is accessed directly by the Client during the laboratory work session. At the same time, it's accessible by the Server for a configuration and management purposes.
The Golang back-end implements the laboratory work session, serves the front-end requests and controls the hardware: it flashes MCU/FPGA and writes/reads data to/from Analog Discovery 2.

2.2 Component overview

2.2.1. Raspberry Pi back-end component

2.2.1.1. Back-end programming language choice

Since the main task of the backend is to transfer signals from the client to the Raspberry Pi, only converting one signal format to another, a technology that excels at handling concurrent operations and has efficient I/O processing is best suited for this task.
Another important criterion for choosing a technology is its popularity, as it is identical to the quality of technology support, the number of libraries and frameworks, documentation and ready-made solutions. An additional advantage of popular technologies is a large user base, that is a community, which allows to reduce the number of errors, and therefore to increase the speed of development, thanks to the use of time-tested solutions.
Important issue is the "cost" of the chosen technology. Go programs are compiled to machine code, which provides excellent performance and efficient resource utilization compared to interpreted languages. Unlike languages that require significant server resources due to heavy runtime environments, Go's lightweight runtime and garbage collector ensure minimal overhead.
The safety of the technology is also important. Go was designed with security in mind from the ground up, featuring built-in memory safety, garbage collection, and strong type checking. These features help prevent common security vulnerabilities like buffer overflows and memory leaks that plague languages like C and C++.

Go's strong typing system and clear interface definitions allow for straightforward API contracts between services. Go's ability to generate API documentation and interface specifications makes it easy to maintain consistency across different parts of the system, even in a polyglot environment.
Go fully satisfies the above criteria:

- Go is an open-source programming language developed by Google, specifically designed for building scalable and maintainable backend systems.
- Go implements built-in concurrency through goroutines and channels, which provides an elegant and efficient way to handle multiple connections and I/O operations without the complexity of manual thread management.
- Go's standard library includes robust networking capabilities and a high-performance HTTP server, making it ideal for building APIs and network services. The language's garbage collector is optimized for low-latency applications, ensuring consistent performance without unexpected pauses.
- Go is compilation model produces single static binaries that include all dependencies, making deployment simple and eliminating runtime dependency issues, which is particularly beneficial for deployment on embedded systems like Raspberry Pi.

2.2.1.2. Back-end programming framework choice

When developing a backend service in Go, the choice of web framework significantly impacts development speed, maintainability, and performance. While Go's standard library provides a robust net/http package, using a framework can substantially reduce boilerplate code and provide additional useful features without sacrificing Go's inherent performance benefits.
The primary criteria for selecting a web framework include:

- Performance characteristics, especially under concurrent load
- Community size and active maintenance
- Learning curve and documentation quality
- Built-in middleware ecosystem

- API design flexibility.

Gin Framework satisfies these requirements and provides several distinct advantages:

- Gin is built on top of “httprouter” module, which provides extremely fast HTTP request routing through the use of a radix tree structure, making it up to 40 times faster than Go's default multiplexer
- The framework is widely adopted in the Go ecosystem, being one of the most starred Go web frameworks on GitHub (over 70k stars), which ensures long-term maintenance and a rich ecosystem of third-party middleware
- Gin provides a clean, intuitive API that follows Go's philosophy of simplicity and explicitness, making it easy for both beginners and experienced developers to work with
- The framework includes built-in support for common web development tasks.

While alternatives like Echo, Fiber, and Chi were also considered, Gin was chosen for several specific reasons:

- Its middleware system is simple yet powerful, allowing easy integration of custom processing logic for authentication, logging, and error handling
- The framework's support for binding request data to Go structs reduces the amount of manual parsing code needed
- Gin's error management approach aligns well with Go's error handling patterns while providing additional convenience methods
- The framework's minimal memory footprint and zero allocation router make it particularly suitable for running on resource-constrained systems like Raspberry Pi

For the signal processing backend, Gin's routing capabilities and middleware system are particularly valuable as they allow us to:

- Easily set up different endpoints for various signal types
- Implement request validation to ensure signal format correctness
- Add authentication and authorization layers
- Handle WebSocket connections for real-time signal transmission (if needed)
- Implement rate limiting and request throttling when needed

2.2.2. Raspberry Pi client software component

2.2.2.1. Client software programming framework choice

Alongside with a server component that is responsible for sending responses back to user, additional component that allows generating requests is also necessary. Client in web developing sphere is also known as frontend, while server part is commonly called backend. The main tasks of the frontend are producing requests and providing human machine interface as an intermediate component between user input and server data. Generally, program code is stored on the server. This approach helps us simplify project structure and automatically install all updates for all possible users (it is because web applications can be accessed on any device via the browser: from portable smartphones to large PCs). Similar to server, there are several frameworks for client software that can be used in the process of web application development.

It is important to note that in this system frontend part delivers to end user single-page web application, where all input and output components can be accessed through only one web page.

Creating new large-scale applications with the use of programming or markup language as the only one component is a time-consuming process in a modern world. That is why frameworks were invented. Framework is a program platform which defines project structure, its file subsystem and is built based on a programming language. For instance, Spring is a Java-based framework for simplifying creation and testing of corporate applications.
There are many frameworks for writing effective and attractive single-page applications: React, Angular, Vue.js, Backbone.js, Mithril etc. However, from this variety of program platforms only three of them are commonly used: React, Angular and Vue.js.
React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library that aims to make building user interfaces based on components more "seamless". It is maintained by Meta company and a community of individual developers and companies. This framework can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js. Because React is only concerned with the user interface and rendering components to the DOM, React applications often rely on libraries for routing and other client-side functionality. A key advantage of React is that it only re-renders those parts of the page that have changed, avoiding unnecessary re-rendering of unchanged DOM elements. That is why it can be very powerful for applications with dynamic components generating.
Another one single-page framework is Angular. Angular (also referred to as "Angular 2+") is a TypeScript-based free and open-source web application framework, developed by Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. The Angular ecosystem consists of a diverse group of over 1.7 million developers, library authors, and content creators. According to the Stack Overflow Developer Survey, Angular is one of the most commonly used web frameworks. Main features include component-based architecture, data binding, dependency injection, server-side rendering, routing and use of directives.
Vue.js (also referred to as Vue) is an open-source model–view–viewmodel frontend JavaScript framework for building user interfaces and single-page applications. It was created by Evan You and is maintained by him and the rest of the active core team members. Vue.js features an incrementally adaptable architecture that focuses on declarative rendering and component composition. The core library is focused on the view layer only. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages. Vue.js allows for extending HTML with HTML attributes called directives. The directives offer functionality to HTML-applications, and come as either built-in or user defined directives. This framework also features a reactivity system that uses plain JavaScript objects and optimized re-rendering. Each component keeps track of its reactive dependencies during its render, so the system knows precisely when to re-render, and which components to re-render.
As we stated before, our web application includes different input and output components that are generated dynamically based on conditions. Furthermore, it is important to note that we should use solutions which are regularly maintained by developers. For these cases the most appropriate choice is the React framework.

2.3 SREE Prototype Hardware catalogue

- Server, routing equipment – 1 set.
- Network equipment - 1 set.
- Single-board computers (Raspberry Pi 4) – 1 pcs.

- Development boards for digital design (FPGA boards) – 1 pcs;
- Development boards for microcontroller systems (MCU boards) – 1 pcs;
- Webcam – 1 pcs;
- Laptop – 1 pcs;
- APC Smart – 1 pcs.

2.4 SREE Prototype Software catalogue

Free software for programming of MCU/FPGA boards:

- Web browser with Internet connection;
- STM32CubeIDE for the MCU (accessible by link https://www.st.com/en/development-tools/stm32cubeide.html);
- Intel Quartus Prime Lite for the FPGA (accessible by link https://www.intel.com/content/www/us/en/software-kit/660907/intel-quartus-prime-lite-edition-design-software-version-20-1-1-for-windows.html).

2.5 Prototype performance/Capacity

- 1 prototype work place
- 1 GB disk space for user

2.6 Prototype Security and Safety

Communications should have a secure connection. Need SSL certificates for HTTPS.

3 Software specification

3.1 Back-end software specification

The back-end project contains of the “main.go” file with all the used modules imports and initializations and has a “internal” folder which contains multiple subfolders for each used module.
The implemented modules are:

- Analog Discovery
- Camera
- Config
- FPGA
- STM32 Flash

The Analog Discovery module implements various features including setting output voltages for buttons emulation.
The Camera module creates an “/stream” endpoint that produces a stream of video frames used by the front-end.
The Config module is responsible for parsing the .env file content for the project configuration: setting paths, used pin numbers, etc.
The FPGA and STM32 Flash models are used for FPGA and STM32 flashing respectively.
It’s possible to flash the FPGA via Raspberry Pi’s GPIO directly without the USB Blaster using a created by us software that is a fork of a urjtag tool changed to support the new GPIO drivers using a C programming language.

The STM32 is flashed using “stm32flash” tool and is set to bootloader mode using GPIO controlled by the “pinctrl” tool.
The project also contains the docker-compose.yml file together with the nginx.conf file both used for the Nginx setup as a docker container.
The main.go content with all the initializations and endpoints creations is represented below:

```
var outputPins = []int{0, 1, 2, 3}
var outputChannels = []int{0, 1}
var wavegenFunctions = []string{"sine", "rampup", "triangle", "pulse"}
func main() {
	r := gin.Default()
	cfg, err := config.LoadConfig()
	if err != nil {
		log.Fatalf("Error loading config:", err)
	}
	cam, err := camera.NewWebcamServer("/dev/video0")
	if err != nil {
		log.Fatal(err)
	}
	defer cam.Close()
	device, err := analogdiscovery.CreateDevice()
	if err != nil {
		log.Fatalf("Error creating Analog Discovery device: %v", err)
	}
	for _, outputPin := range outputPins {
	device.SetPinMode(outputPin, true)
	}

	r.POST("/api/firmware/fpga", handleFirmware(*cfg, true))
	r.POST("/api/firmware/mcu", handleFirmware(*cfg, false))
	r.POST("/api/write-pin", handleWritePin(device))
	r.POST("/api/wavegen/write-channel", handleWavegenEnableChannel(device))
	r.POST("/api/wavegen/write-function", handleWavegenFunctionSet(device))
	r.POST("/api/wavegen/write-amplitude", handleWavegenAmplitudeSet(device))
	r.POST("/api/wavegen/write-frequency", handleWavegenFrequencySet(device))
	//r.POST("/api/wavegen/duty-cycle")
	r.POST("/api/wavegen/write-config", handleWavegenRun(device))
	r.Any("/api/stream", cam.ServeHTTP)
	log.Fatal(r.Run(":" + cfg.PORT))
}
func handleFirmware(cfg config.Config, isFPGA bool) func(c *gin.Context) {
	return func(c *gin.Context) {
	// Limit the size of the request body
	c.Request.Body = http.MaxBytesReader(c.Writer, c.Request.Body, maxUploadSize)
	if err := c.Request.ParseMultipartForm(maxUploadSize); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "File too large"})
		return
	}
	// Get the file from the request
	file, err := c.FormFile("file")
	if err != nil {
```

```
			c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
			return
		}
		postfix := "MCU"
			if isFPGA {
			postfix = "FPGA"
		}
		fp := filepath.Join(uploadPath, postfix)
		c.SaveUploadedFile(file, fp)
		fmt.Println("Firmware file uploaded:", file.Filename, " to ", fp, " for ",
		postfix)
		if isFPGA {
			fmt.Println("Flashing FPGA")
			device := fpga.CreateFPGA(cfg.TDI, cfg.TDO, cfg.TCK, cfg.TMS)
			err = device.Flash(fp)
		} else {
			fmt.Println("Flashing STM32")
			err = stm32flash.Flash(fp, cfg.RESET_PIN, cfg.BOOT0_PIN)
		}
		if err != nil {
			fmt.Println("Error flashing device:", err)
			c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
			return
		}
		c.JSON(http.StatusOK, gin.H{"message": "Firmware flashed successfully"})
		}
}
type WritePinRequest struct {
		Pin   int `json:"pin"`
		State int `json:"state"`
}
func handleWritePin(device *analogdiscovery.AnalogDiscoveryDevice) func(c
*gin.Context) {
		return func(c *gin.Context) {
		var pinReq WritePinRequest
		decoder := json.NewDecoder(c.Request.Body)
		if err := decoder.Decode(&pinReq); err != nil {
			c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid request body"})
			return
		}
		if !isPinAllowed(pinReq.Pin) {
			c.JSON(http.StatusBadRequest, gin.H{"error": fmt.Sprintf("Invalid pin,
			only %v are allowed", outputPins)})
			return
		}
}
```

Back-end repository is located on the link https://github.com/vtinkerer/mcu-and-fpga-remote-lab and FPGA firmware to programming of FPGA is located on the link https://github.com/vtinkerer/Raspberry-Pi-Flash-FPGA-using-GPIO.

3.2 Front-end software specification

Front-end software is written with the use of web development technologies: HTML, CSS, JavaScript, and frameworks: Bootstrap (CSS-based framework) and React (JavaScript-based framework).
The main source code is stored inside /src project directory. Each element in it is either a level-down directory or React component (file that ends with .js extension).
The entire structure of /src directory consists of these elements:

- /img (directory that stores all images which are non-Bootstrap icons or non-HTML-elements)
- /styles (directory where all CSS-styles are defined)
- /utils (directory with the only one file called get-url-for-request.js, which is responsible for convenient obtaining of HTTP-request URL)
- App.js (main component, which is represented with web page containing all tools)
- AppCountdown.js (special component that works as an experiment countdown)
- CameraView.js (component which contains a frame where real-time video from laboratory is being shown)
- DigitalInputs.js (component with digital inputs that can work in one of three modes: ‘Button’, ‘Switch’, ‘Gen’)
- DigitalInputsButton.js (component that is rendered when ‘Button’ mode of Digital Inputs is chosen)
- DigitalInputsSwitch.js (component that is rendered when ‘Switch’ mode of Digital Inputs is chosen)
- DigitalInputsGen.js (component that is rendered when ‘Gen’ mode of Digital Inputs is chosen)
- FunctionalGenerator.js (includes FunctionalGeneratorChannel.js)
- FunctionalGeneratorChannel.js (component for device that allows students to generate different types of signals (sine, ramp up, triangle and pulse) with a preferred amplitude, frequency and duty cycle values)
- Index.js (entry point for the whole web application)
- Instructions.js (set of instructions for the laboratory work)
- ProgramDevice.js (component that allows users to program FPGA or MCU (read more about it in 3.3 “Firmware specification”))
- Scope.js (component for Oscilloscope)
- ScopeChart.js (component for Oscilloscope chart, where users can investigate captured signals).

Typical React component in /src directory contains function definition (JSX markup), import statements, hooks for action handlers and export statement (this allows component to be used inside another one). Moreover, some of them include transferring specific data to the child components (for instance, DigitalInputs passes ‘channel’ property value to its child component (either DigitalInputsGen, DigitalInputsSwitch or DigitalInputsButton). The same can be said about Scope and ScopeChart, FucntionalGenerator and FunctionalGeneratorChannel.

In a /styles directory all CSS-styles for respective React components are placed. It is important to note that there are no special CSS-files for subcomponents (for example, all styles of ScopeChart.js, which is a subcomponent of Scope.js, are stored inside Scope.css). Also keep in mind the usage of Bootstrap framework, so that custom-defined CSS-styles in most cases are sets of additional classes and properties with the purpose to modify basic Bootstrap components (buttons, markup, columns etc.).
In order to dynamically display some elements of a React component functions with a render prefix are implemented. It means that if condition of some variable or hook is met, then one block will be shown, and, if not, then another one will be displayed. This also allows us to dynamically define colours of texts (for instance, if text means success of performed operation, then make it look like green, if it means failure, then make this text red).
The FunctionalGeneratorChannel, CameraView and DigitalInputsButton components currently include HTTP-requests to the server. In the future similar requests are going to be added to DigitalInputsSwitch, DigitalInputsGen and Scope. After receiving JSON-response from server components are rendered dynamically too.
Communication between server and client is being handled thanks to JSON-requests and JSON-responses. However, there is one exception from this rule for ProgramDevice component. There a FormData object is used in order to upload firmware for MCU or FPGA board.
Front-end repository is located on the link https://github.com/BogdanVeligorskyi/mcu-and-fpga-remote-lab-frontend.

4 User interface definition

4.1. Main window of SREE web-interface

The main window of web-interface is shown in Figure 2.

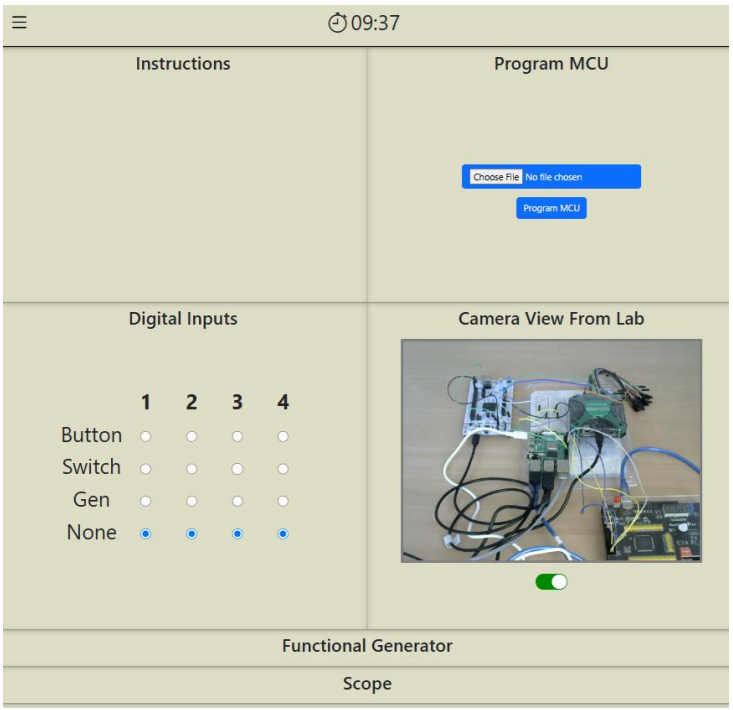


Figure 2. The main window of the web-interface

In this implementation Program MCU and Program FPGA features are combined inside one web page and programmable device can be chosen by checking its checkbox in submenu (Figure 3) that is called by click on button ☰.

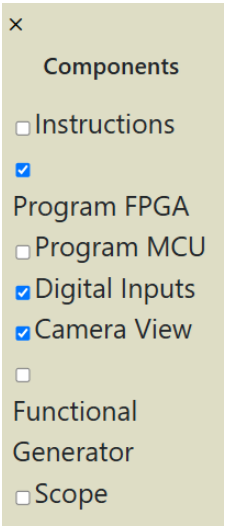


Figure 3. The submenu of main window

Also, from the submenu it is possible to switch ON/OFF the deference frames of main window to show the control panels. The timer 08:16 on upper line of main window show time that is leave in current lab session on the work place.

4.2. Digital inputs

An example of web-based interface to set up of input signals is shown in Figure 4.

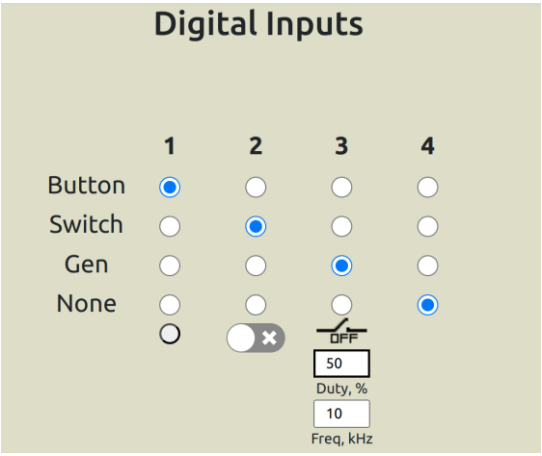


Figure 4. Web-based interface to set up of input signals

Digital input interface consists of 4 elements, each of them can be configured as one of three input functions: “Button”, “Switch”, “Generator”, or as “None”, if it is not used.
The function “Button” provides user following functionality: when grey button is pressed, then pin is set to logical “1”, and when it is released, then pin is set to logical “0”.
In contrast with the “Button”, the function “Switch” has another principle of the operation. In order to set logical “1”, student needs to press switch below the channel radio buttons and

move it to the right side. After that student can set the logical “0” by moving pressed switch to the left side.
The function “Generator”, when it is selected, provide for any out of 4 digital inputs the functionality of the simple functional generator with PWM-signal. User can on it or off by means of the “on/off” switch, as well as set a duty cycle (in the range from 0 to 100 %) and frequency (from 1 kHz to 30 kHz).

4.3. Functional signal generator

Web-interface for the functional signal generator, that can be used for the producing of several most commonly used types of signals, is shown in Figure 5.

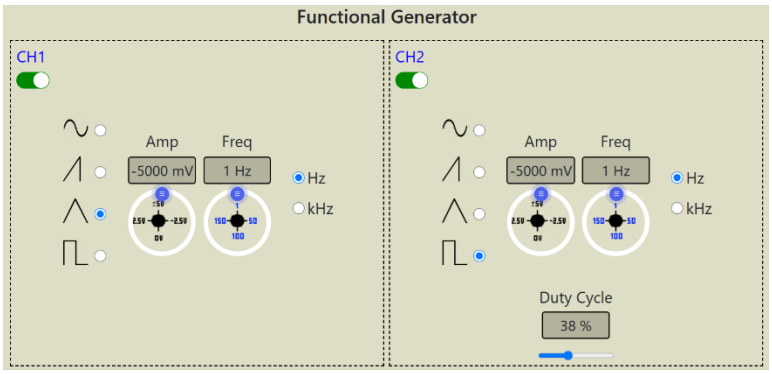


Figure 5. Web interface of the functional signal generator

User can set one of the following types of the signal: sine. ramp up, triangle and pulse. The ranges of the signal parameters are set by utilizing such input elements, as knobs on the round sliders. Parameters are amplitude, frequency and duty cycle.

4.4. Oscilloscope

An Oscilloscope will be connected which enables opportunity to view and examine captured signals. Its web interface is shown in Figure 6.

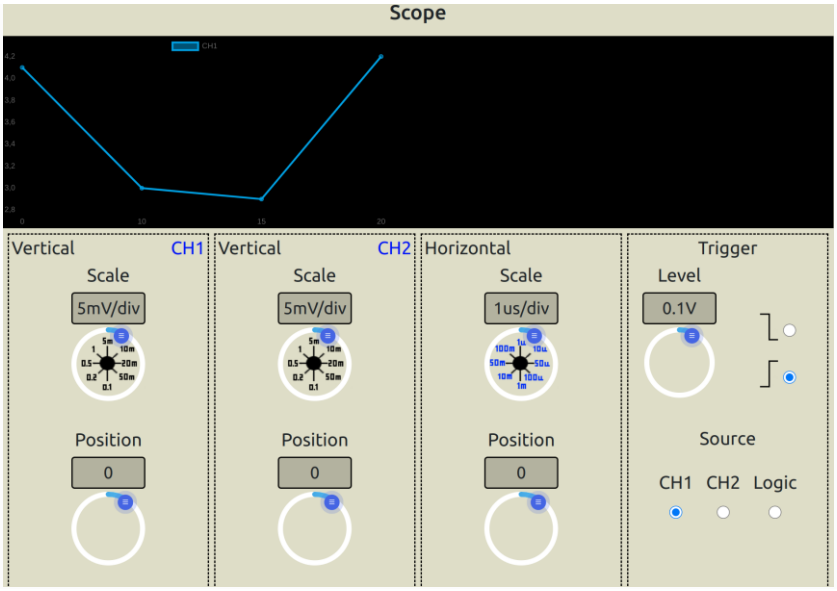


Figure 6. Web interface of Scope component

Student can change vertical (Y) and horizontal (X) scales of virtual oscilloscope and choose appropriate position by moving knobs on round sliders. Furthermore, in order to gain real experience, users have option to select trigger level value, trigger source option and trigger fail/rise mode.

4.5. Web-camera

Web-camera (or Camera View From Lab) demostrates video from laboratory where all the equipment and hardware is connected. Thanks to web-camera student can see how devices and LEDs react to the user input. For instance, when student presses button in Digital Inputs, a LED lights on. This brings more interactivity to the remote laboratories experiment.
However, there is one more element that needs to be explained. That is on/off switch. By default its value is set to off, which means that camera is switched off, and instead of real-time video student sees black block. When camera is switched on, then video starts demonstrating with approximately 1-second delay. Web-camera frame is shown in Figure 7.

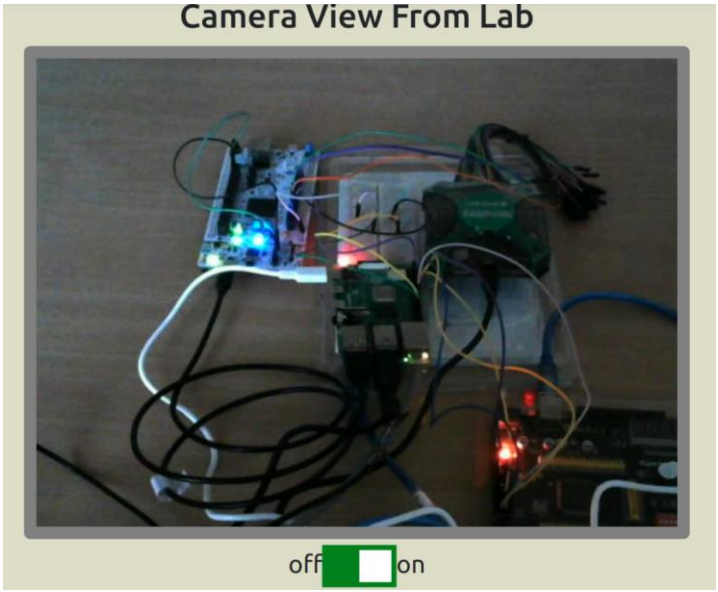
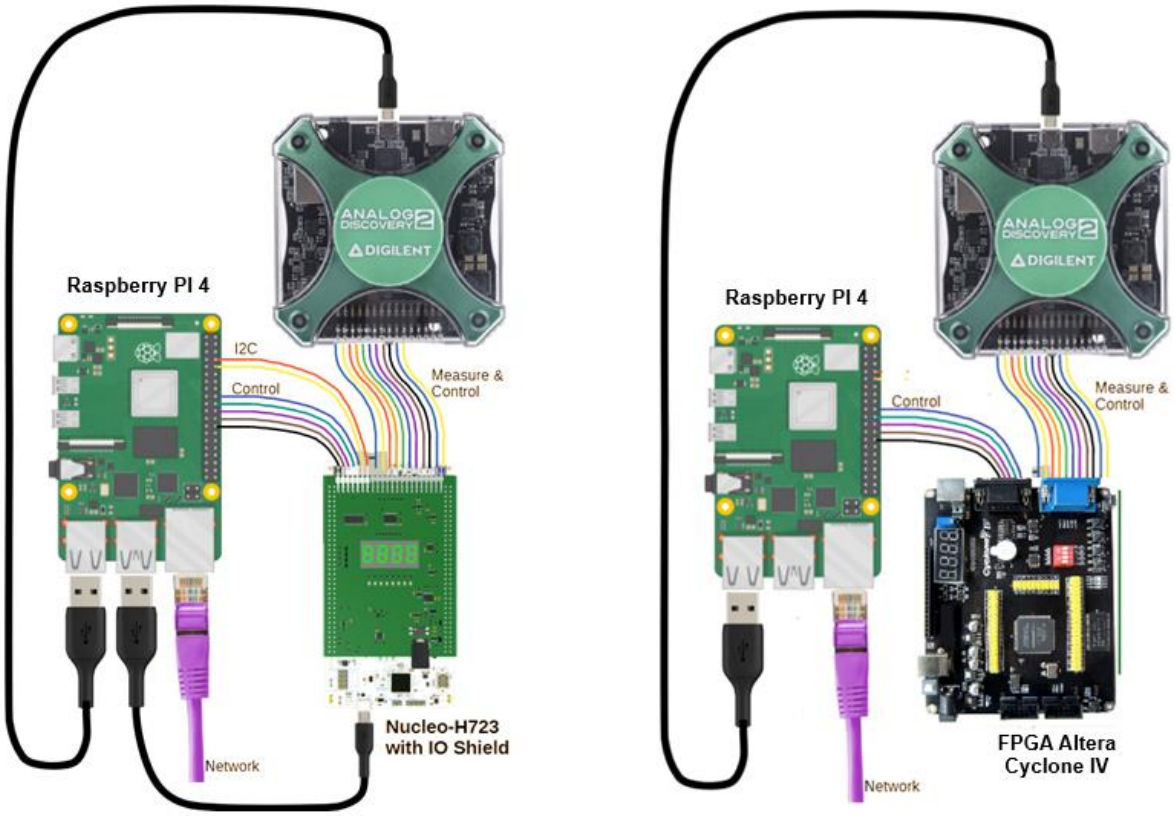


Figure 7. Web-camera frome

5 Prototype deployment

5.1. Hardware deployment

The structures of hardware deployment for two variant of prototype realization is shown in Figure 8. Deference is in absent of USB and I2C connections between Raspberry PI 4 and FPGA plate.



a) MCU variant b) FPGA variant

Figure 8. MCU/FPGA prototype hardware structure

5.2. Software deployment

The back-end is deployed as a simple Golang process started by the PM2 software so it can be easily restarted on reboot together with a PM2's Logs module for a convenient logs manipulation and log files rotation.
The Nginx is deployed as a docker container for a simplicity purpose and as a future-proof decision for multiple Nginx instances on the board if needed.
The front-end is built on the host-machine by Node.js and served as a .js, .css and .html files by the Nginx directly from the build folder, so each build process causes Nginx to serve the new files immediately.
The reworked by us urjtag version is in the home directory of the Raspberry Pi and accessed by the back-end as a child process.
The whole system is located on the Raspberry Pi's SD card, which image can be easily duplicated when needed to increase the number of Raspberry Pi instances.

6 Pilot demonstration user manual

6.1 Preparation and uploading of MCU firmware

Considering that the SREE platform is intended for lab exercises, the student needs to have special software (IDE) for generating firmware. This firmware will be uploaded remotely to the MCU board connected to the SREE via a web interface, as shown in Figure 7. To do this, the student needs to perform the following steps.

1. Write the program in C language in STM32CubeIDE software according to the technical task, for example, the LED blinking effect. When the button is pressed all the discrete LEDs on the Nucleo-H723 IO Shield are blinking and when the button is not pressed all the LEDs are switched on constantly. To do this create the project in the STM32CubeIDE for the STM32H723ZG MCU with the next settings of the clocks (see Figures 9 and 10) and GPIOs (see Figures 11 and 12). Then add the following portion of C code to the endless loop in the main function (see Figure 13):

```
// While Button 1 is pressed (high-level signal) blink the LEDs
while (HAL_GPIO_ReadPin(Button1_GPIO_Port, Button1_Pin))
{
 // Toggle the state of the LED1, LED2, LED3, and LED4 connected to the GPIO
       // port F pins of the MCU
 HAL_GPIO_TogglePin(GPIOF, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
 // Toggle the state of the LED5, LED6, LED7, and LED8 connected to the GPIO
       // port B pins of the MCU
 HAL_GPIO_TogglePin(GPIOB, LED5_Pin | LED6_Pin | LED7_Pin | LED8_Pin);
 // Delay 500 ms
 HAL_Delay(500);
}
// When Button 1 is not pressed anymore check the state of the LEDs (one LED is  //
enough because all the LEDs are blinking simultaneously and therefore have the
// same state)
if (HAL_GPIO_ReadPin(GPIOF, LED1_Pin) == GPIO_PIN_RESET)
{
 // If the LEDs are off, toggle the state of the LEDs
 HAL_GPIO_TogglePin(GPIOF, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
 HAL_GPIO_TogglePin(GPIOB, LED5_Pin | LED6_Pin | LED7_Pin | LED8_Pin);
}
```

2. Generate the hex file, which will be uploaded remotely to MCU board. For this purpose the option "*Convert to Intel Hex file (-O ihex)*" in the **Project>>Properties>>C/C++ Build>>Settings>>MCU/MPU Post build outputs** should be set. Finally the project compilation and linking must be done by the command **Project>>Build Project**. The generated hex file can be found in the project tree's **Debug** or **Release** folder (depending on the active build configuration).
3. Upload the hex file through the web-interface to MCU board by choosing the file in your computer and clicking "*Program MCU*" (see Figure 14).
4. Interact with the hardware (MCU/FPGA boards) remotely via the web interface of SREE, for example by pushing button 1 (see Figure 4), or by using switch 1. The responses of the MCU board to the input signal can be monitored by the web-camera (LED indicators, displays on the MCU board IO Shield).

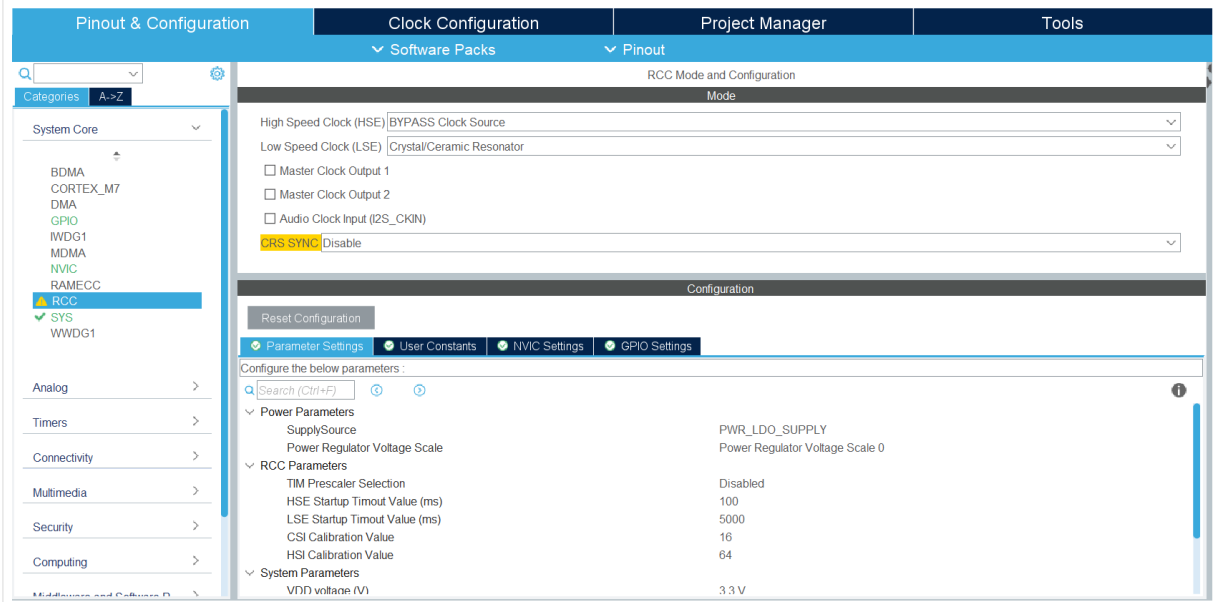


Figure 9. MCU RCC settings

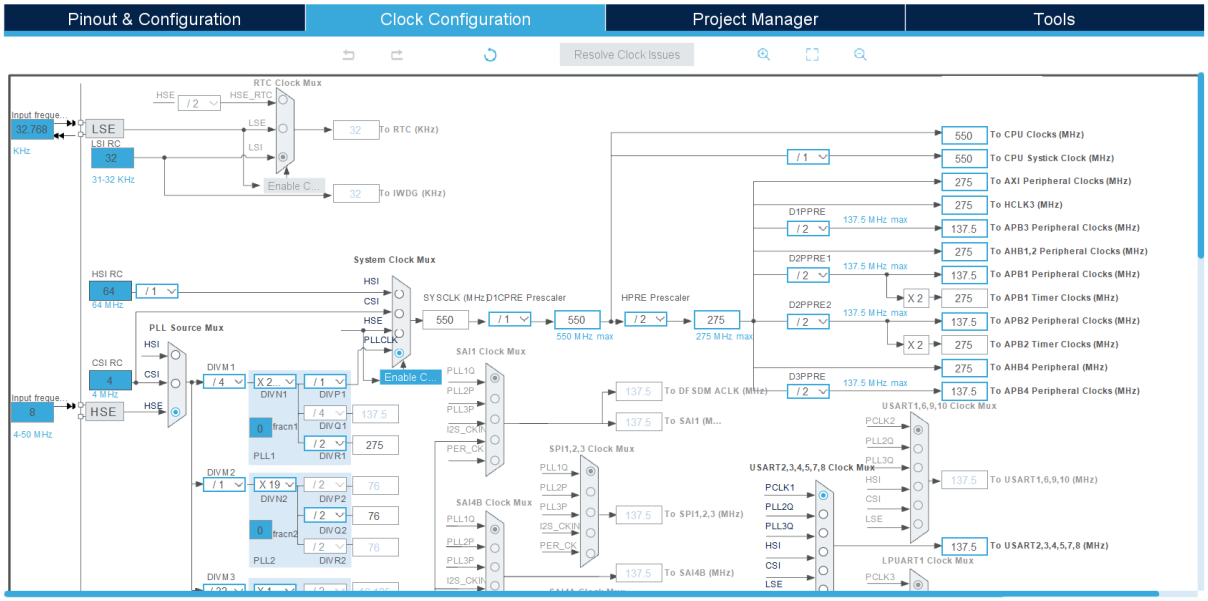


Figure 10. MCU Clock configuration

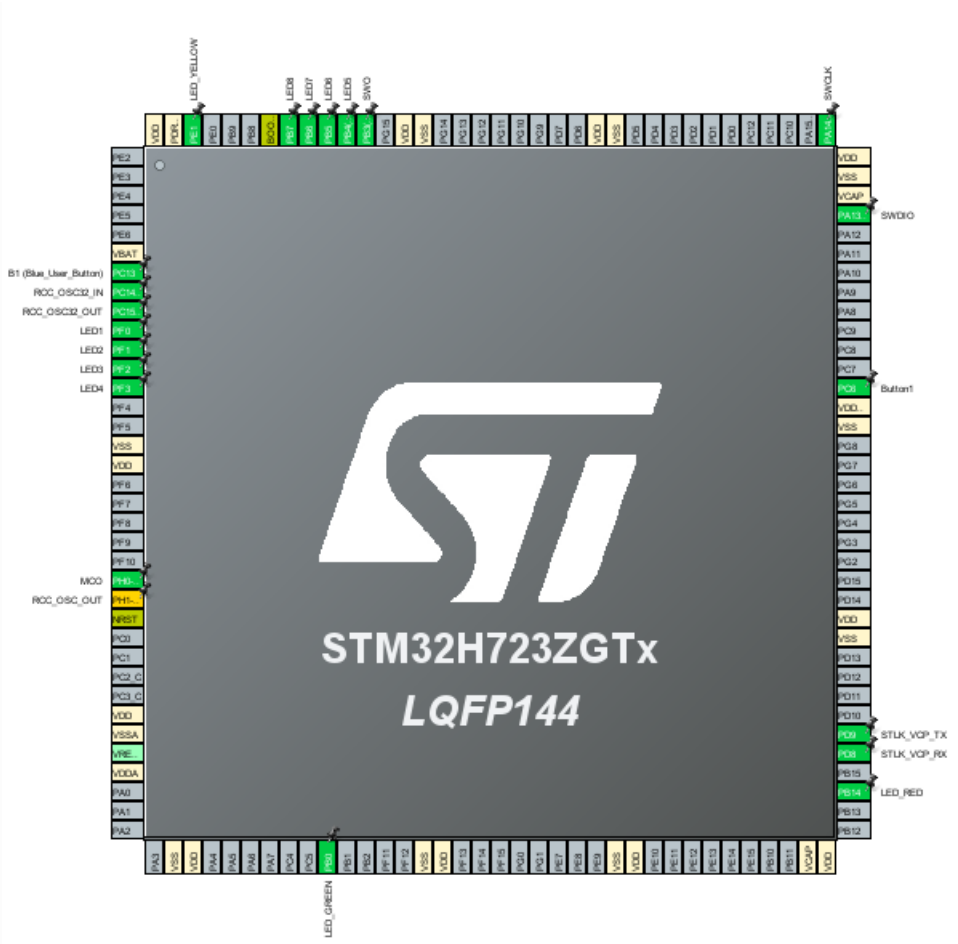


Figure 11. MCU GPIO select

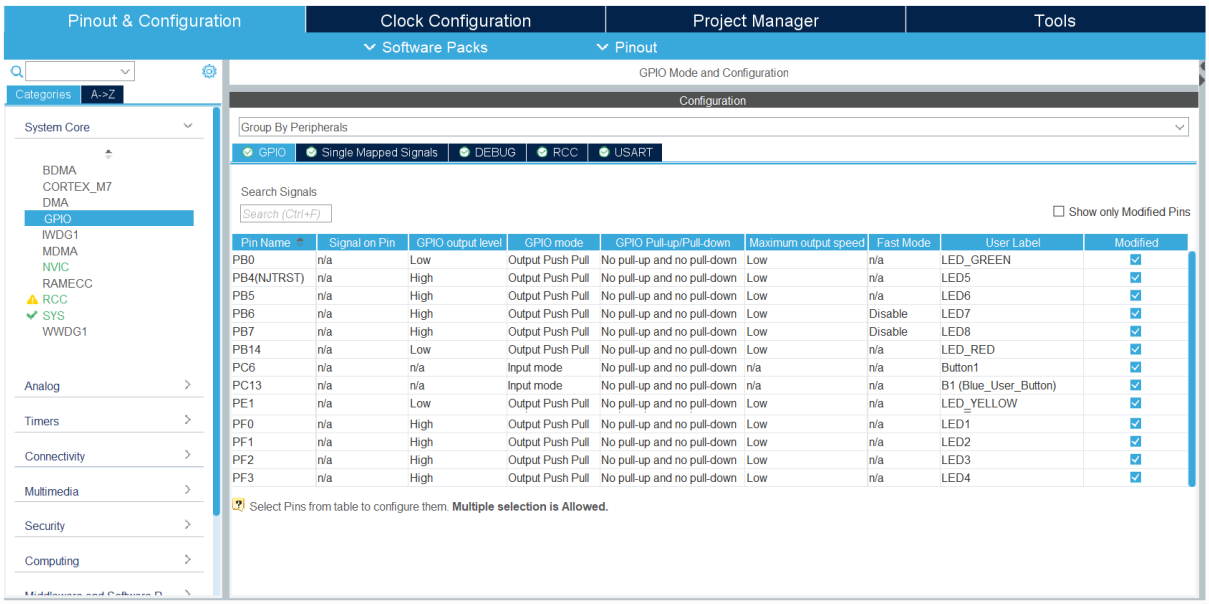


Figure 12. MCU GPIO settings

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    // While Button 1 is pressed (high-level signal) blink the LEDs
    while (HAL_GPIO_ReadPin(Button1_GPIO_Port, Button1_Pin))
      {
        // Toggle the state of the LED1, LED2, LED3, and LED4 connected to the GPIO port F pins of the MCU
        HAL_GPIO_TogglePin(GPIOF, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
        // Toggle the state of the LED5, LED6, LED7, and LED8 connected to the GPIO port B pins of the MCU
        HAL_GPIO_TogglePin(GPIOB, LED5_Pin | LED6_Pin | LED7_Pin | LED8_Pin);
        // Delay 500 ms
        HAL_Delay(500);
      }
    // When Button 1 is not pressed anymore check the state of the LEDs (one LED is enough because all the
    // LEDs are blinking simultaneously and therefore have the same state)
    if (HAL_GPIO_ReadPin(GPIOF, LED1_Pin) == GPIO_PIN_RESET)
      {
        // If the LEDs are off, toggle the state of the LEDs
        HAL_GPIO_TogglePin(GPIOF, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
        HAL_GPIO_TogglePin(GPIOB, LED5_Pin | LED6_Pin | LED7_Pin | LED8_Pin);
      }
  /* USER CODE END WHILE */
```

Figure 13. Blinking LEDs program infinite loop

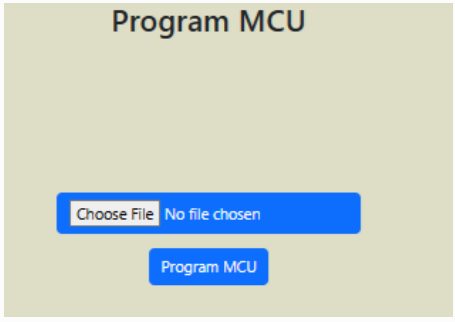


Figure 14. MCU remote programming dialogue in the SREE web-interface

6.2 Preparation and uploading of FPGA firmware

Generation of a firmware file for an FPGA development board in Intel Quartus Prime Lite software is carried out in several stages and requires project configuration. The following will provide step-by-step instructions for setting up the project, taking into account safety requirements and the need to use peripheral modules of the development board.

The Intel Quartus Prime software organizes and manages the elements of your design within a project. The project encapsulates information about your design files, hierarchy, libraries, constraints, and project settings. This chapter describes the basics of working with Intel Quartus Prime software projects, including initial project setup, viewing project information, adding design files and constraints, and exporting compilation results.

Note! It is highly recommended to use a new folder for each Quartus Prime project.

Click File -> New Project Wizard to quickly setup and open a new project, Figure 15.

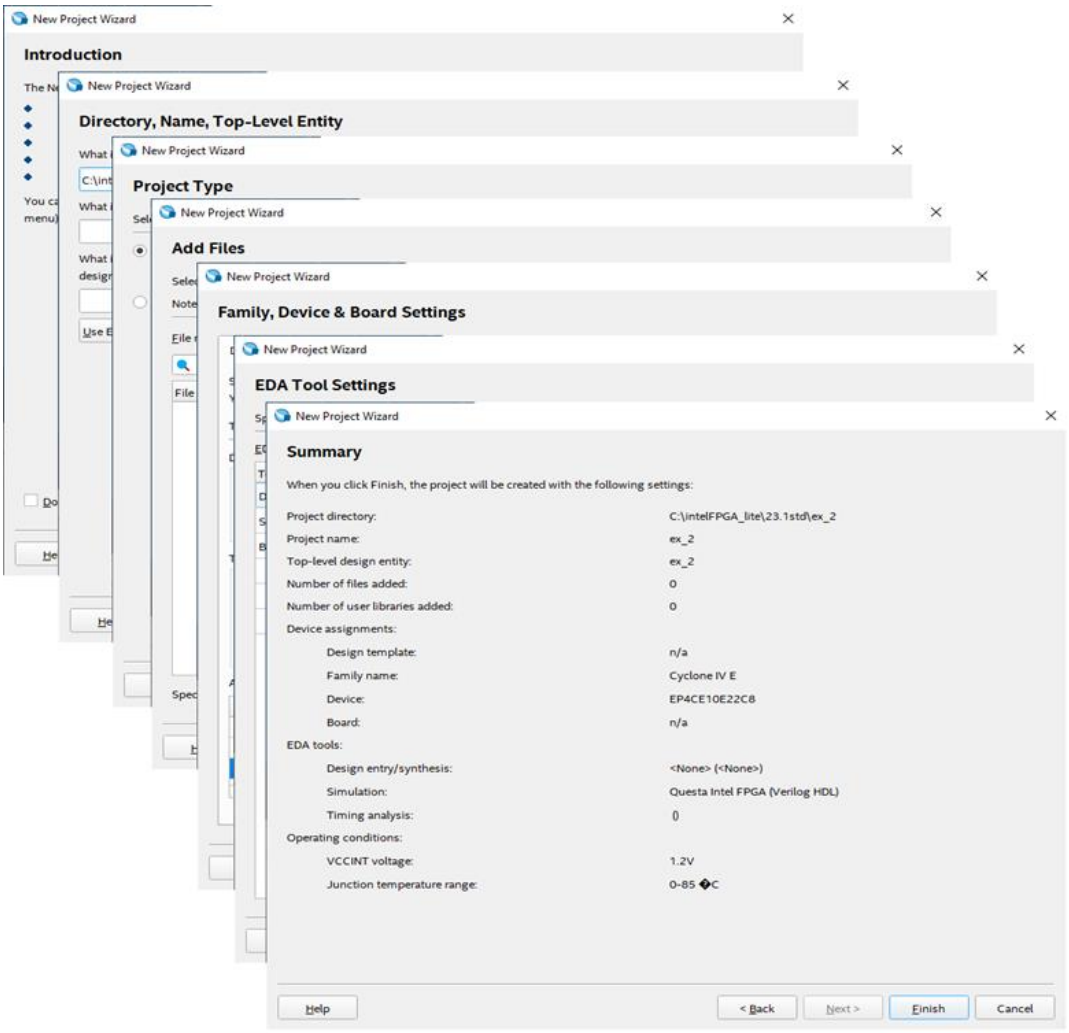


Figure 15 – New project wizard

The most critical step in new project generation is **Family, Device & board settings**. At this window user must correctly chose FPGA model presented in laboratory. In this case it is Altera Cyclone IV **EP4CE10E228N**. If precisely **EP4CE10E228N** is absent in Quartus Prime device library for Cyclone IV, user may choose **EP4CE10E228** (see Figure 16).

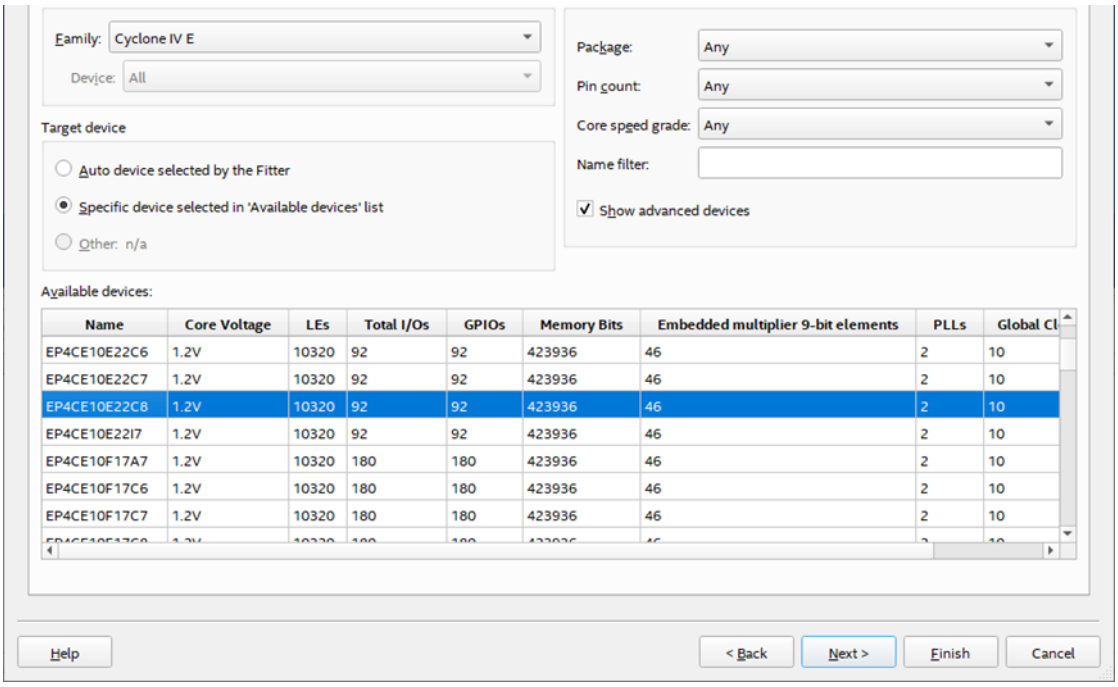


Figure 16 – Choose the correct device

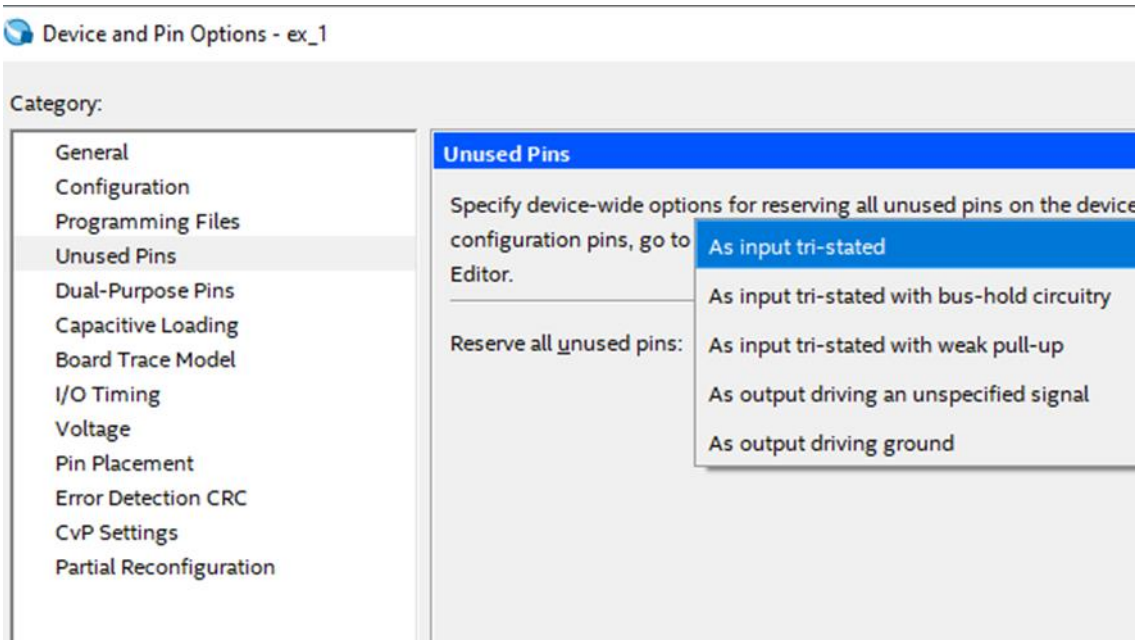


Figure 17 – Unused pins safety setting

There are only four user programmed buttons and four LEDs on the development board physically. The connection to FPGA pin schematic is demonstrated in Figure 18. Active logic level for LED light is LOW (“0”) and LOW (“0”) for pressed buttons.
The pin numbers for LEDs, buttons and 7-Segment LED display are presented in table 1.

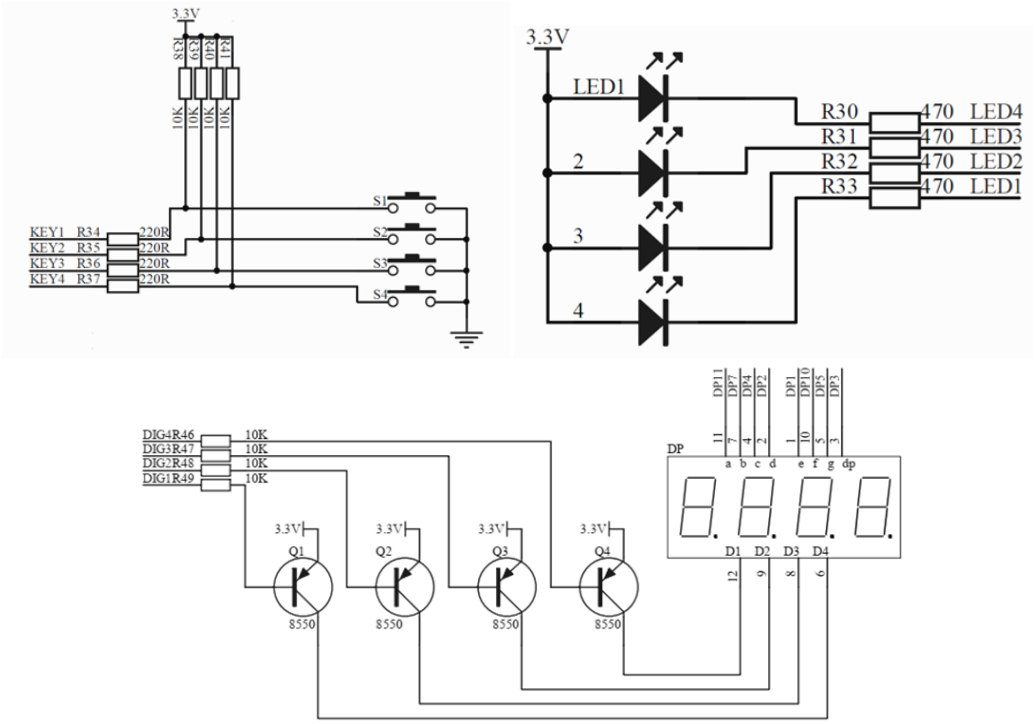


Figure 18 – Schematic connection of LEDs, buttons7-Segment LED display

Table 1 – Pin numbers and functions

Name	Pin number	I/O type	Logic level
KEY1	88	input	3.3V TTL
KEY2	89	input	3.3V TTL
KEY3	90	input	3.3V TTL
KEY4	91	input	3.3V TTL
LED1	87	output	3.3V TTL
LED2	86	output	3.3V TTL
LED3	85	output	3.3V TTL
LED4	84	output	3.3V TTL
DIG1	133	output	3.3V TTL
DIG2	135	output	3.3V TTL
DIG3	136	output	3.3V TTL
DIG4	137	output	3.3V TTL
SEG0	128	output	3.3V TTL
SEG1	121	output	3.3V TTL
SEG2	125	output	3.3V TTL
SEG3	129	output	3.3V TTL
SEG4	132	output	3.3V TTL
SEG5	126	output	3.3V TTL
SEG6	124	output	3.3V TTL
SEG7	127	output	3.3V TTL

To connect all necessary pins in your project design to development board pins use pin planner in Quartus Prime tools menu (Figure 19). Choose the correct pin location for each signal in your design. After that all pin numbers would be presented in block diagram schematic file.

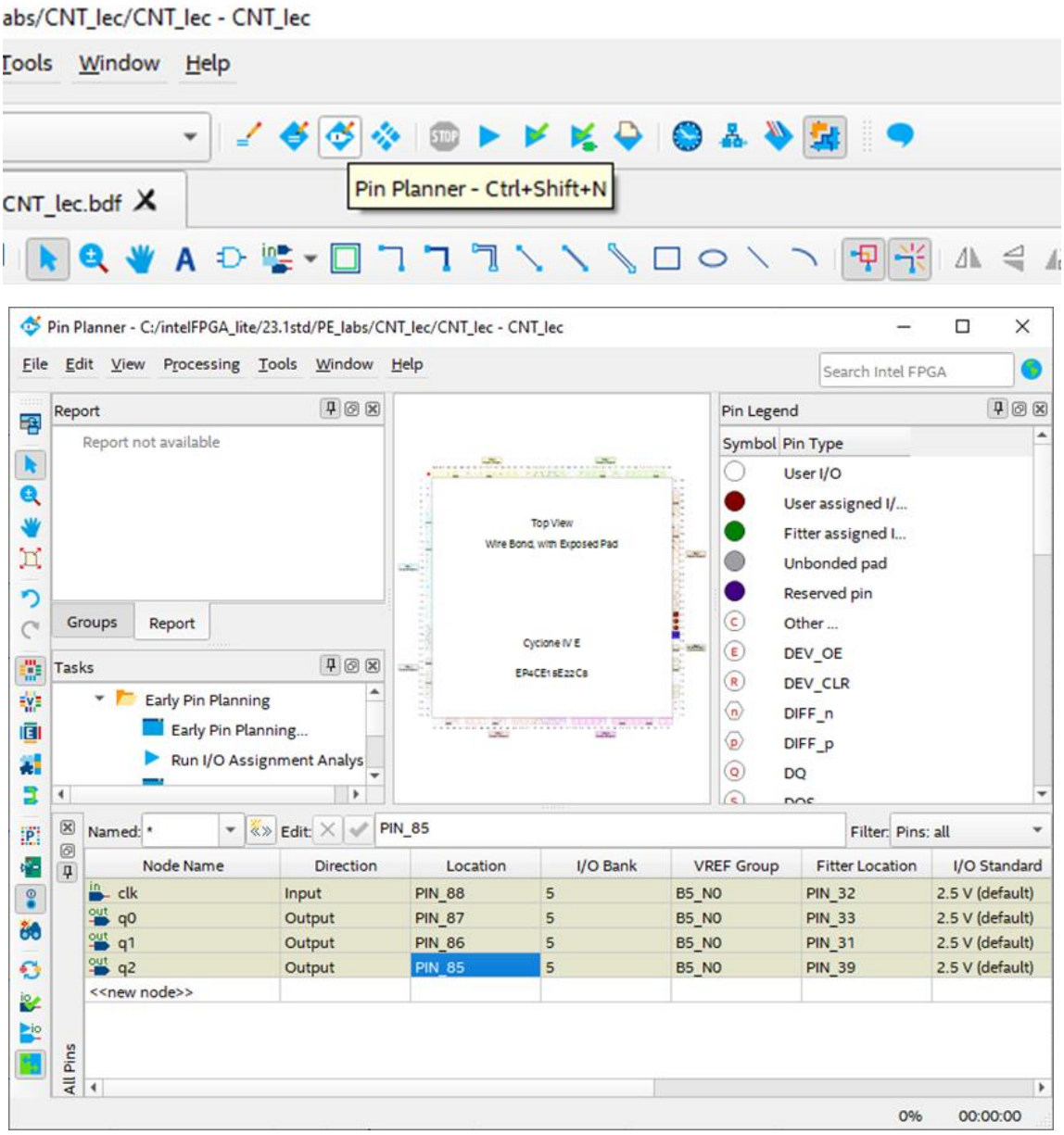


Figure 19 – Pin connection in Pin planner

Quartus Prime software always generate SRAM Object File *.sof firmware file after compilation of the project. It also generates Partial SRAM Object File *.psof or a Programmer Object File *.pof depending on the device family.

To load the firmware file to the development board user must find *.sof file at the project folder project_folder\output_files\%project_name%.sof and upload it to the web page.

6.3 Launching pilot demonstration

- To launch the pilot demonstration run the prototype from web-browser by link http://195.69.76.135:8082/;
- In submenu switch ON Instruction, Program FPGA or MCU, Digital Inputs, Camera View, Functional Generator and Scope flags;
- Chose a file with program for MCU (mcu.hex) or FPGA (fpga.svf) respectively from https://github.com/vtinkerer/mcu-and-fpga-remote-lab/commit/7bb0c1a5aa9bd62da9ce04ff903873301b7cbea6
- Follow the Instruction in main window that is shown in Figure 20.

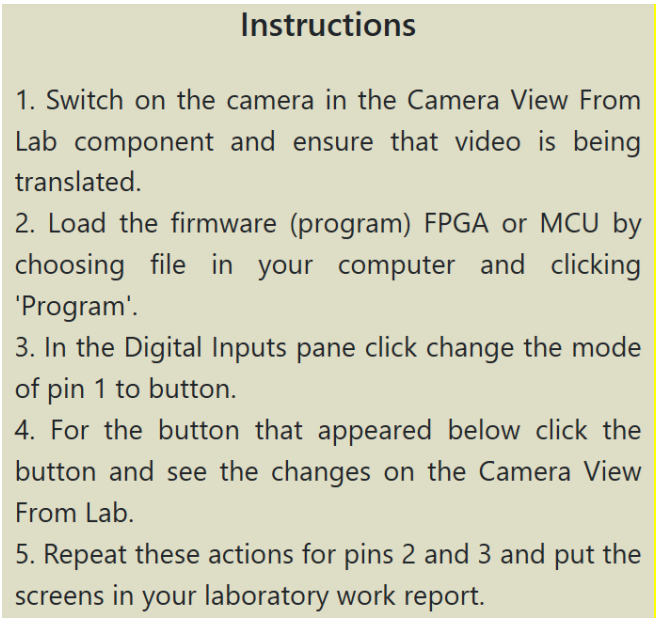


Figure 20 – Pilot demonstration Instruction

Note: This D 3.2 Pilot doesn't allow training lab lessons but demonstrates only a working SREE prototype and shows planned possibilities for online experiments with real equipment in remote mode. Extended possibilities will be implemented in next D3.3, D3.4 and D3.5 deliverables.