Table of Contents

1. Laboratory on the Internet of Things - Advanced IoT Hardware - on-site 2
1.1. Hardware 2
1.1.1. ESP32 WROVER DEVKIT 2
1.1.1.1. Built-in integrated LCD (TFT) display 3
1.1.1.2. Configuring VREL - platformio.ini 4
1.1.2. M5 Stack ESP32 Core Ink DEVKIT 5
1.1.2.1. Built-in integrated e-Paper display 5
1.1.2.2. Configuring VREL - platformio.ini 6
1.1.3. A special note on PWM in ESP32 6
1.2. Laboratory scenarios 6
1.2.1. Task 1 7
1.2.2. Task 2 7
1.2.3. Task 3 7
1.2.4. Task 4 8
1.2.5. Report 8
1.3. Resources 8

1. Laboratory on the Internet of Things - Advanced IoT Hardware - on-site

Laboratory guide for IoT AIH students on ESP32 graphical display programming:

- ESP32 WROVER DEVKIT module - that includes (among others) dot-matrix LCD TFT display
- M5 Stack CORE Ink DEVKIT Module - treat contains (among others) dot-matrix e-paper (e-ink) display

1.1. Hardware

* ESP32 WROVER DEVKIT with LCD TFT module * ESP32 PICO D4 M5 Stack with e-Paper (e-Ink) module

Below we present hardware details on those two modules:

1.1.1. ESP32 WROVER DEVKIT

This board is double-sided; on one side, you will find an LCD screen (so-called **back**) on the other side (**front**) there are GPIOs connectors you connect sensors and actuators to, TF (microSD) card slot with SPI reader and an RGB LED (among others).
Please note you can directly observe only the **back** side of the module via the VREL node's camera.

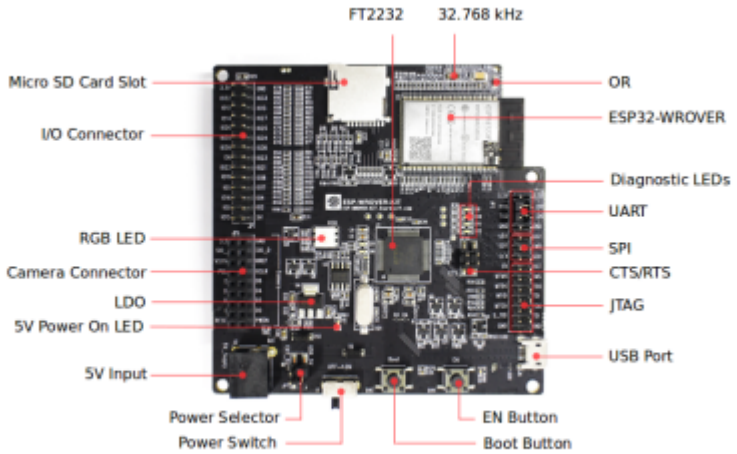
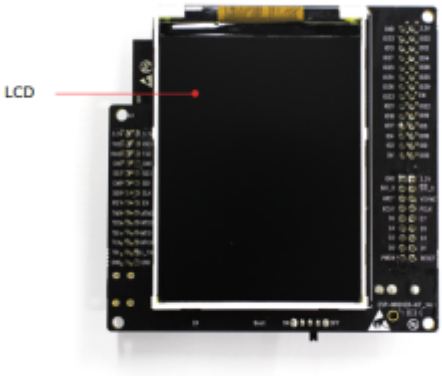


Fig. 1: ESP32 WROVER Dev Kit front side [1)]

Fig. 2: ESP32 WROVER Dev Kit back side [2)]



ESP32 provides a number of GPIOs to the developers, but some of them are hard-bound to their features as connected on the development board.

Some of the GPIOs are dedicated to implementing a limited number of features, while some are dedicated to using a particular protocol on this development board. As this development board contains some peripherals (RGB LED, LCD, TF card reader), do not use GPIOs that are controlling them.

The following list presents GPIOs used with the integrated hardware:

Tab. 1: ESP32 WROVER DEVKIT GPIO and features

No	Feature	ESP32 GPIO Pin	Notes
1	JTAG	GPIO12-15	
2	Camera	GPIO0, GPIO4-5, GPIO18-19, GPIO21-23, GPIO25-27, GPIO34-36, GPIO39	If you do not use the camera, feel free to use those GPIOs unless they're not shared with another feature.
3	RGB LED	GPIO0, GPIO2, GPIO4	
4	MicroSD card reader	GPIO2, GPIO4, GPIO12-15, GPIO21	
5	LCD	GPIO5, GPIO18-19, GPIO21-23, GPIO25	

Note - GPIOs 34, 35, 36 and 39 do not have internal pull-up nor pull-down resistors

1.1.1.1. Built-in integrated LCD (TFT) display

The built-in display is ILI9341 TFT display, 320×240 pixels (240×320 actually), RGB, 262k colours. Although it is possible to program it raw, we will use a ready software library to control it. This TFT display uses 16-bit per pixel memory organisation, frequently referenced as 565 (RGB, R-5-bits, G-6-bits, B-5-bits).

1.1.1.2. Configuring VREL - platformio.ini

Correct configuration is necessary to build and deploy your code.
For ESP32 WROVER DEVKIT, a good starting point is the following content for the `platformio.ini` file:

```
[env:d1_mini]
platform = espressif32
board = esp-wrover-kit
framework = arduino
lib_ldf_mode = deep+
lib_deps =
   adafruit/Adafruit GFX Library
   Wire
   espressif/WROVER KIT LCD @ ^1.0.3
```

1.1.1.2.1. Practice: handling LCD display

Write Hello World on the LCD screen.
Using an LCD and a camera, you can observe it through, are your only interface to the devkit. Therefore, please get familiar with handling LCD screens and give it a short practice, as presented below.

Type textual “Hello World” on the screen. Then, you're welcome to decorate it, change colours, size, orientation, etc.

To use LCD, you must include libraries. The beginning of your `.cpp` source code file should then look more-less following way:

```
...
#include <Arduino.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <WROVER_KIT_LCD.h>
...
```

Then define a controller class that manages your LCD:

```
...
WROVER_KIT_LCD tft;
...
```

Remember to initiate the LCD controller class:

```
...
  tft.begin();
...
```

Some useful functions to handle LCD operations are presented below (but not limited to):

```
...
  tft.fillScreen(WROVER_BLACK); //Clear screen
  tft.setCursor(0, 0); //Set upper left corner
  tft.setTextColor(WROVER_WHITE);  //Set text colour
  tft.setTextSize(2); //Set font size
  tft.println("Hello World!"); // Print the text
...
```

1.1.2. M5 Stack ESP32 Core Ink DEVKIT

This M5 Stack module is an integrated ESP32 microcontroller (PICO D4 version, full specification) is on the manufacturer's page. This module, even if compact in size, also contains expansion ports, an internal battery powering RTC (BM8563) when no external power is present, 3 physical buttons: reset, power and programmable, a green LED and a passive buzzer.

ESP32 provides a number of GPIOs to the developers, but some of them are hard-bound to their features as connected on the development board.



Following GPIOS are exposed: G25, G26, G36, G23, G34, G18, G21, G22, G14, G13



Some of the GPIOs are dedicated to implementing a limited number of features, while some are dedicated to using a particular protocol on this development board. This development board contains some peripherals (RGB LED, LCD, TF card reader) that do not use GPIOs that are controlling them.

Tab. 2: M5 Stack Core Ink Development Kit features and connections

No	Feature	ESP32 GPIO Pin	Notes
1	EPaper (e-Ink) SPI display controller	GPIO4, GPIO0, GPIO15, GPIO9, GPIO18, GPIO23	In order: EPaper BUSY, ePaper RST, EPaper DC, EPaper CS, EPaper SCLK, EPaper MOSI
2	Multifunction button, a physical button, led and buzzer	GPIO37, GPIO38, GPIO39, GPIO5, GPIO10, GPIO2, GPIO12	In order: Left (up) multifunction, Button press, Right (down) multifunction, Physical button, LED, Buzzer, Power control MOS
4	Internal I2C for BM8563 (RTC)	GPIO21, GPIO22	SDA, SCL, module I2C address is 0x51)
5	External port (I2C)	GPIO32, GPIO33	software-defined M-BUS
6	ESP32 Serial (via USB to serial chip, CP2104)	GPIO1, GPIO3	RXD, TXD

1.1.2.1. Built-in integrated e-Paper display

The e-Paper (e-Ink) display is a bi-coloured, 200×200 pixels with an SPI interface module GDEW0154M09. It has an integrated SPI controller.

1.1.2.2. Configuring VREL - platformio.ini

Correct configuration is necessary to build and deploy your code.
For M5 Stack Core Ink Development Kit, a good starting point is the following content for the `platformio.ini` file:

```
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32
framework = arduino
lib_deps =
    m5stack/M5Core-Ink@^0.0.7
```

1.1.2.2.1. Practice: handling e-Paper display

Write Hello World on the e-Paper screen.

1.1.3. A special note on PWM in ESP32

PWM is frequently used to control analogue-style, efficient voltage on the GPIO pin. Instead of using a resistance driver, PWM uses pulses to change adequate power delivered to the actuator. It applies to motors but also to LEDs, bulbs, heaters, and indirectly to the servos (but that works actually another way, see lectures).
The classical `analogWrite` method known as Arduino (Uno, Mega) and ESP8266 do not work here. ESP32 has sixteen (0 to 15) PWM channels (controllers) that can be freely bound to any of the regular GPIOs.
Moreover, ESP32 can use various “resolutions” of the PWM: from 8-bit (0..255) to 15bit (0..32767), while regular Arduino uses only 8-bit one.

To use PWM in ESP32, one must perform the following steps:

- Configure GPIO pin as OUTPUT
- Initiate PWM controller with fixing PWM frequency and resolution
- Bind the controller to the GPIO pin
- Write to the controller (not to the PIN!) providing a duty cycle related to the resolution selected above - every call persistently sets the PWM duty cycle until there is the next call to the function setting duty cycle.

1.2. Laboratory scenarios

In general, you will program using the Arduino framework.
As presented during the lecture, you will also use FreeRTOS functions to handle multicore parallel programming to perform tasks, as demonstrated during the lecture.



ESP32 framework uses C++ optimiser while compiling the source code. It is common



that C++ traces unused code and removes/optimises it, to minimise CPU effort. As your tasks to be completed, involve integer and floating-point performance evaluation, it is essential to use the result somewhere, i.e. do a dummy display of the final value on the screen. Otherwise, the C++ compiler may remove this section of the code, and you won't be able to obtain reasonable results, because your code may be already trunk and not executed at all!



FreeRTOS, once the task is created and scheduled for execution, executes the task infinitely! It means, your function implementing the task will be executed again and again. If you intended to run a looped task, that is pretty OK. If you intended to run the one-off task, then you need explicitly call:

```
...
vTaskDelete(NULL);
...
```

as a last line of your function implementing the task.

1.2.1. Task 1

Evaluate single-core performance for integer computation. Perform two experiments: with task bound to the APP core and separately to the PRO core. First, observe if there is a difference in measurements. Then, propose an algorithm that can generate a complexity of integer computation observable and measurable. Perform at least 10 measurements for each experiment. Consider using parts of code for the Dhrystone benchmark https://en.wikipedia.org/wiki/Dhrystone.

1.2.2. Task 2

Evaluate multicore performance for integer computation. Perform single-core computation, running at least 4 times (any even number works here) your algorithm. Then execute in parallel (2×2). Observe if there is a difference in measurements. Propose an algorithm that can generate a complexity of integer computation observable and measurable. Perform at least 20 measurements for each experiment (10 serial + 10 parallel). Consider using parts of code for the Dhrystone benchmark https://en.wikipedia.org/wiki/Dhrystone.

1.2.3. Task 3

Evaluate single-core performance for floating-point computation. Perform two experiments: with task bound to the APP core and separately to the PRO core. First, observe if there is a difference in measurements. Then, propose an algorithm that can generate a complex-enough floating points computation that is observable and measurable. Perform at least 10 measurements for each experiment. Consider using FP parts of code for Whetstone benchmark https://en.wikipedia.org/wiki/Whetstone_(benchmark).

1.2.4. Task 4

Evaluate multicore performance for floating-point computation. First, perform single-core computation, running your algorithm at least 4 times (any even number works here) your algorithm. Then execute in parallel (i.e. 2×2). Finally, propose an algorithm that can generate a complex-enough floating-point computation that is observable and measurable. Perform at least 20 measurements for each experiment (10 serial + 10 parallel). Consider using FP parts of code for Whetstone benchmark https://en.wikipedia.org/wiki/Whetstone_(benchmark).

1.2.5. Report

Include algorithm definition and measurements, along with statistical analysis of the results (at least average and standard deviation) and a summary.

1.3. Resources

1. ESP32 WROVER kit starting page
2. Hardware definition (pinouts, gpios, etc.)
3. ESP-WROVER-KIT V4.1 schematic
4. https://github.com/LilyGO/TTGO-Electronic-Badge
5. https://github.com/lewisxhe/TTGO-EPaper-Series
6. https://www.waveshare.com/wiki/2.9inch_e-Paper_Module
7. https://github.com/lewisxhe/GxEPD
8. On-line bitmap (image) to memory content (cpp) converter
9. FreeRTOS - multitasking with ESP32 explained

[1)] , [2)]
https://docs.espressif.com/projects/esp-idf/en/latest/hw-reference/get-started-wrover-kit.html

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:aih:lab:gfxonsite?rev=1668547644

Last update: **2022/11/15 22:27**