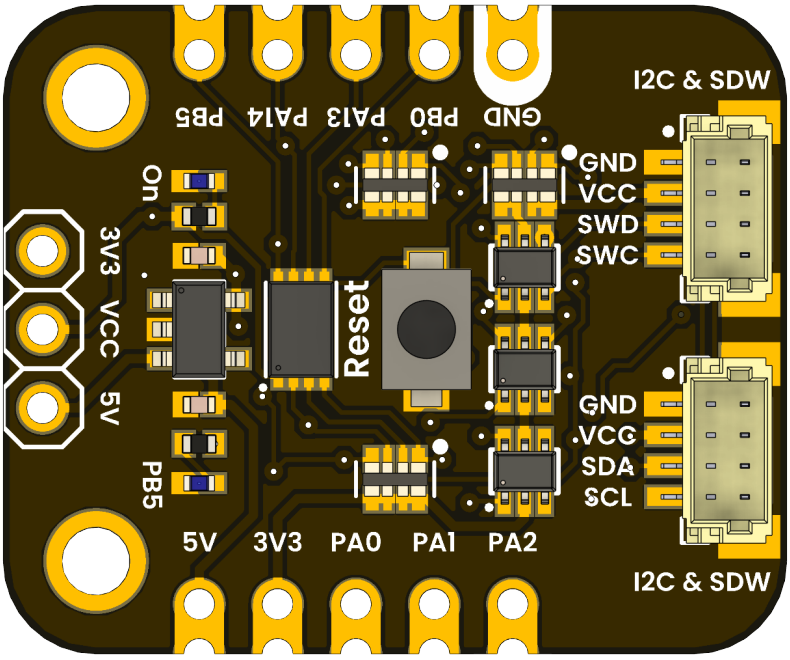


DevLab Development Board User Guide and Technical Reference

Release 0.0.1

Department of Research, Innovation, and Development

Feb 03, 2026

Contents

The DevLab Development Board based on the PY32F003L24D6TR microcontroller is designed for rapid prototyping, embedded systems education, IoT experimentation, and wearable devices. This board combines flexible power options, modern connectivity, and accessible interfaces to accelerate your hardware development. The microcontroller features a 32-bit ARM Cortex-M0 core, up to 24 MHz clock speed, 16KB Flash memory, and 2KB SRAM, making it suitable for a wide range of applications. With built-in peripherals like SPI, I2C, UART, and a 12-bit ADC, the board supports diverse project requirements.

CHAPTER
ONE

TERMS, ACKNOWLEDGMENTS, AND LICENSES

1.1 Terms and Conditions

By using, modifying, or distributing the documentation, firmware, or hardware designs included in this repository, you agree to the following terms:

- All materials are provided **"as-is"**, without warranty of any kind.
- The authors and contributors shall not be held responsible for **any damages**, data loss, or legal issues arising from the use of these materials.
- Usage is intended for **educational, development, prototyping**, and other lawful purposes.
- When redistributing or reusing any part of this project, you must **retain attribution** and comply with the corresponding license terms of each component.

1.2 MIT License

Copyright (c) 2025 UNIT-Electronics-MX

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER
TWO

INTRODUCTION

The DevLab Development Board, powered by the PY32F003L24D6TR microcontroller, provides an efficient and compact platform for rapid prototyping, embedded systems education, IoT experimentation, and wearable technology development. Its design emphasizes flexibility and simplicity, integrating essential features that allow developers to focus on innovation rather than complex setups. The board operates around a 32-bit ARM® Cortex®-M0 core running up to 24 MHz, with 16 KB of Flash and 2 KB of SRAM, offering an ideal balance between processing performance and memory efficiency. This configuration enables smooth execution of control algorithms, communication routines, and sensor data processing across a wide range of embedded applications.

To complement its processing capability, the DevLab board includes a rich set of onboard peripherals and hardware conveniences. Interfaces such as SPI, I^2C, and UART simplify communication with external modules and sensors, while the 12-bit ADC and dual comparators support precise analog signal acquisition. The integrated 5 V $\rightarrow$ 3.3 V voltage regulator, 4-pin JST 1.0 mm connector for programming and I^2C communication, reset button, and status LED on PB5 streamline both debugging and prototyping. Furthermore, the board maintains level-shifter compatibility between 3.3 V and 5 V logic, ensuring seamless operation with a broad ecosystem of peripherals. Together, these features make the DevLab PY32F003 board a practical and reliable tool for students, makers, and engineers aiming to explore low-power, high-efficiency embedded systems.

2.1 Key Features

- **ARM® Cortex®-M0 Core**: 32-bit RISC processor running up to 24 MHz, optimized for real-time control and low-power operation.
- **Compact and Efficient Design**: Designed for embedded and IoT applications requiring small form factor and low energy consumption.
- **Wide Operating Voltage**: Fully functional from 1.7 V to 5.5 V, suitable for battery or USB-powered systems.
- **Onboard Voltage Regulator**: Integrated 5 V $\rightarrow$ 3.3 V LDO regulator, ensuring stable MCU operation from USB or external 5 V supply.
- **Communication Interface**: 4-pin JST 1.0 mm connector for programming and I^2C communication, compatible with standard DevLab and Qwiic wiring schemes.
- **User Interface Components**:
  - Reset Button: Allows manual restart of the microcontroller.
  - Status LED (PB5): Blinks for power or firmware activity indication.
- **Analog and Timing Peripherals**:
  - 12-bit ADC (4 external + 2 internal channels)
  - 2 analog comparators
  - 1 advanced and 4 general-purpose 16-bit timers
  - Low-power timer, SysTick, and dual watchdogs
- **Communication Peripherals**: 1× SPI, 1× I^2C, 1× USART — supporting up to 1 MHz I^2C, 18 MHz SPI, and LIN/IrDA UART modes.
- **Direct Memory Access (DMA)**: 3 independent channels for high-speed peripheral data handling.
- **Real-Time Clock (RTC)**: Built-in calendar and alarm support with low-power retention.
- **GPIO and Expansion**: Up to 7 multipurpose I/O pins with alternate function mapping.
- **Level Shifter Compatibility**: Fully supports logic level interfacing between 3.3 V and 5 V systems, ensuring cross-platform signal safety.
- **Operating Environment**: -40 °C to +85 °C industrial temperature range.
- **DevLab format compatibility**.

The DevLab format is a standardized hardware layout that exposes the majority of the microcontroller or integrated

circuit functional pins, including connections for external sensors, converters, and other peripheral devices. It provides direct access to I²C, SPI, UART, and additional communication interfaces for flexible hardware interaction.

2.2 Hardware Electrics characteristics

Table 2.1: Electrical characteristics

Parameter	Description	Min	Typ	Max	Unit
Vin	Input voltage to power on the module	2.5	—	5.5	V
Ivcc	Current flowing into VCC pin of microcontroller	—	—	100	mA
I3v3*	Maximum current of 3V3 regulator	—	—	600	mA
Iio	Maximum Input/Output current of IOs	—	—	20	mA
f_ext	User external clock frequency	—	8	32	MHz
Vih	Input high level voltage	0.7V	—	—	V
Vil	Input low level voltage	—	—	0.3V	V
Vol	Output low level voltage	—	—	0.4	V
Voh	Output high level voltage	VCC 0.4	—	—	V

2.3 Software Support

- **Arduino IDE**
- **Py32f0xx-hal/ VS Code**: https://github.com/UNIT-Electronics-MX/py32f0xx-hal — Supports development with automated builds, serial monitoring, and advanced debugging projects.

2.3.1 Py32f0xx-hal

Hardware Abstraction Layer (HAL) for PY32F0xx microcontrollers in Rust

What is this?

This crate provides a Hardware Abstraction Layer (HAL) for the PY32F0xx family of microcontrollers from Puya Semiconductor. These are low-cost ARM Cortex-M0+ based MCUs that offer an excellent alternative to STM32F0xx series, perfect for:

2.4 Applications

- **Consumer Electronics**: Smart home devices, small appliances, and battery-powered gadgets.
- **Industrial Control**: Sensor interfaces, automation modules, power monitoring, and fan controllers.
- **IoT Devices**: Environmental data loggers, wireless bridge modules, smart meters, and wearable submodules.
- **Lighting Systems**: RGB/NeoPixel LED controllers, PWM dimmers, and sound-reactive lighting.
- **Education & Prototyping**: Training kits, MCU programming practice, and portable debugging tools.
- **Interface Control**: I²C/SPI bridges, input devices, touch keys, and DAC-based tone generators.

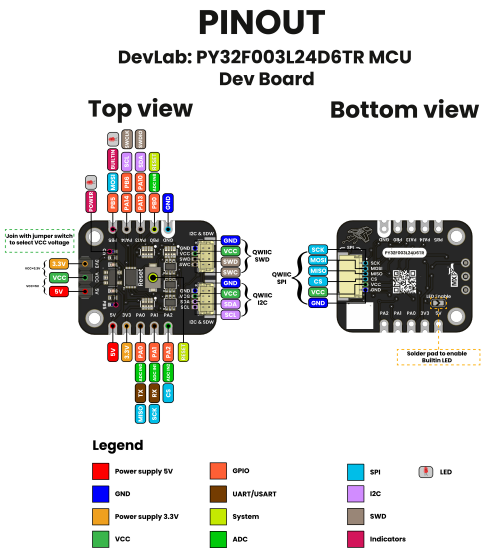
PY32F0xx Hardware Abstraction Layer (HAL) documentation

CHAPTER
THREE

HARDWARE PINOUT

This section describes the pin mapping of the DevLab Development Board based on the PY32F003L24T6 microcontroller.

3.1 Pinout Diagram



The image above shows the physical pinout of the DevLab board and its relationship with the PY32F003L24T6 MCU pins.

3.2 Board Pin Description

Table 3.1: DevLab Pinout Overview

Pin Label	Description
VCC	Power input
GND	Ground
PA0	USART2_TX / SPI1_MISO
PA1	USART2_RX / SPI1_SCK
PA2	ADC_IN2 / SPI_CS
PB0 / PF2	GPIO / NRST
PB5	Built-in LED / GPIO / SPI_MOSI
PA13	SWDIO / I2C_SDA
PA14	SWCLK / I2C_SCL
PB6	GPIO / I2C_SCL (alternate)
PA10	GPIO / I2C_SDA (alternate)

3.3 Connectivity Interfaces

Table 3.2: Available Interfaces

Interface	Description
I2C	JST 1.0 mm QWIIC connector (Power + I2C)
SPI	JST 1.0 mm connector (Power + SPI)
GPIO	Two 4-pin headers for general-purpose I/O
SWD	Dedicated programming and debugging interface

CHAPTER
FOUR

MULTIPLEXING FUNCTION MAPPING

4.1 GPIO Alternate Function Registers

Pin multiplexing is controlled by the GPIO Alternate Function Registers (AFR). Each pin uses 4 bits to select the alternate function.

4.1.1 AFR Bit Mapping

- Bits 0–3 : Pin 0
- Bits 4–7 : Pin 1
- Bits 8–11 : Pin 2
- Bits 12–15 : Pin 3
- Bits 16–19 : Pin 4
- Bits 20–23 : Pin 5
- Bits 24–27 : Pin 6
- Bits 28–31 : Pin 7

AF values:

- 0 → GPIO (default)
- 1 → AF1
- …
- 15 → AF15

4.2 Configuration Example

Configure PA0 as USART2_TX (AF9):

```
GPIOA->AFR = (GPIOA->AFR & ~(0xF << 0)) |␣
↪(9 << 0);
```

4.3 Design Considerations

- PA13 / PA14 should remain reserved for SWD during development.
- I2C is recommended on PB6 / PA10 for applications.
- The onboard LED on PB5 shares SPI_MOSI.

4.4 Configuration Example

Configure PA0 as USART2_TX (AF9):

```
GPIOA->AFR = (GPIOA->AFR & ~(0xF << 0)) |␣
↪(9 << 0);
```

4.5 Design Considerations

- PA13 / PA14 should remain reserved for SWD during development.
- I2C is recommended on PB6 / PA10 for applications.
- The onboard LED on PB5 shares SPI_MOSI.

4.6 PA0 – Alternate Function Mapping

AF	Function
AF1	USART1_CTS
AF4	USART2_CTS
AF7	COMP1_OUT
AF9	USART2_TX
AF10	SPI1_MISO
AF13	TIM1_CH3
AF14	TIM1_CH1N
AF15	IR_OUT

4.7 PA1 – Alternate Function Mapping

AF	Function
AF0	SPI1_SCK
AF1	USART1_RTS
AF4	USART2_RTS
AF7	EVENTOUT
AF9	USART2_RX
AF10	SPI1_MOSI
AF13	TIM1_CH4
AF14	TIM1_CH2N
AF15	MCO

4.8 PA2 – Alternate Function Mapping

AF	Function
AF0	SPI1_MOSI
AF1	USART1_TX
AF4	USART2_TX
AF7	COMP2_OUT
AF10	SPI1_SCK
AF12	I2C_SDA
AF13	TIM3_CH1

4.9 PB0 – Alternate Function Mapping

AF	Function
AF0	SPI1_NSS
AF1	TIM3_CH3
AF2	TIM1_CH2N
AF5	EVENTOUT
AF7	COMP1_OUT

4.10 PB5 – Alternate Function Mapping

AF	Function
AF0	SPI1_MOSI
AF1	TIM3_CH2
AF2	TIM16_BKIN
AF3	USART1_CK
AF4	USART2_CK
AF5	LPTIM_IN1
AF7	COMP1_OUT

4.11 PA13 – Alternate Function Mapping

AF	Function
AF0	SWDIO
AF1	IR_OUT
AF7	EVENTOUT
AF9	USART1_RX
AF10	SPI1_MISO
AF13	TIM1_CH2
AF15	MCO

4.12 PA14 – Alternate Function Mapping

AF	Function
AF0	SWCLK
AF1	USART1_TX
AF4	USART2_TX
AF7	EVENTOUT
AF15	MCO

4.13 PB6 – Alternate Function Mapping

AF	Function
AF0	USART1_TX
AF1	TIM1_CH3
AF2	TIM16_CH1N
AF4	USART2_TX
AF5	LPTIM_ETR
AF6	I2C_SCL
AF7	EVENTOUT

4.14 PA10 – Alternate Function Mapping

AF	Function
AF1	USART1_RX
AF2	TIM1_CH3
AF4	USART2_RX
AF6	I2C_SDA
AF7	EVENTOUT
AF9	USART1_TX
AF10	SPI1_NSS
AF12	I2C_SCL

CHAPTER
FIVE

OPENOCD AND PYOCD OVERVIEW

5.1 Introduction

5.1.1 OpenOCD (Open On-Chip Debugger)

OpenOCD is an open-source software tool that facilitates debugging, in-system programming, and boundary-scan testing of embedded devices. It supports a diverse range of microcontrollers including STM32, RP2040, and PY32. Through interfaces such as JTAG or SWD, OpenOCD provides a command-line interface for effective interaction with the target devices.

5.1.2 PyOCD: A Python Interface to OpenOCD

PyOCD simplifies the use of OpenOCD by providing a Python-based high-level interface. It offers a Python API that abstracts the complexities of the OpenOCD command-line interface, making it easier to program and debug microcontrollers. PyOCD is user-friendly, flexible, and designed to be extensible for various microcontroller interactions.

Tool	Functionality
OpenOCD	Provides a command-line interface for debugging and programming microcontrollers.
PyOCD	Offers a Python API for seamless interaction with microcontrollers via OpenOCD.

5.2 ARM Cortex-M Debug Capabilities

5.2.1 Debugging Interface of ARM Cortex-M

The ARM Cortex-M architecture includes a comprehensive debug interface that allows external tools to access the microcontroller core for debugging and programming tasks. This interface includes a suite of registers and commands that empower debuggers to halt the core, access memory, and manage the execution of programs.

5.2.2 Connection Through Debug Pins

Access to the debug interface is facilitated through specific pins on the microcontroller, such as SWDIO, SWCLK, and nRESET. These are linked to a debug adapter like the ST-Link or CMSIS-DAP, which forms the physical bridge between the host computer and the target device.

5.3 CMSIS-DAP: A Standardized Debug Adapter

5.3.1 Overview of CMSIS-DAP

CMSIS-DAP (Cortex Microcontroller Software Interface Standard - Debug Access Port) serves as a standardized debug adapter for ARM Cortex-M devices. It is endorsed by OpenOCD and provides a cost-effective solution compared to proprietary debug adapters.

5.4 SWD Communication Protocols

5.4.1 Signals and Operations

The SWDIO (Serial Wire Debug Input/Output) and SWCLK (Serial Wire Clock) signals are integral for communication over the SWD interface. SWDIO is a bidirectional line used for transmitting debug commands and data, while SWCLK is a clock signal that ensures synchronized data transfer between the host and the target.

5.4.2 Advantages of SWD Over JTAG

Utilizing a two-wire connection, the SWD interface minimizes the pin count required compared to the traditional JTAG interface. This reduction is particularly beneficial for microcontrollers where pin availability is limited.

CHAPTER
SIX

RUST HAL FOR PY32F0XX MICROCONTROLLERS

Rust HAL is developed to provide a safe and efficient interface for programming PY32F0xx microcontrollers using the Rust programming language. It abstracts the low-level hardware details, allowing developers to write high-level code while still having access to the underlying hardware features.

- Rust HAL for PY32F0xx microcontrollers
- Documentation

6.1 Why Rust HAL?

- **Safety**: Rust's ownership model and type system help prevent common programming errors, such as null pointer dereferencing and data races, making embedded development more reliable.
- **Performance**: Rust is designed for performance, allowing developers to write low-level code that runs efficiently on resource-constrained microcontrollers.
- **Abstraction**: The HAL provides a high-level interface to interact with the microcontroller's peripherals, making it easier to develop applications without needing to manage hardware registers directly.
- **Community Support**: The Rust embedded ecosystem is growing, with active community contributing to libraries, tools, and documentation.

6.2 Linux Support

The Rust HAL for PY32F0xx microcontrollers is fully supported on Linux platforms. Developers can leverage the rich set of development tools available on Linux, including Cargo (Rust's package manager and build system), GDB for debugging, and various flashing tools compatible with PY32F0xx microcontrollers.

6.3 Getting Started

To get started with Rust HAL for PY32F0xx microcontrollers, follow these steps:

1. **Set Up Rust Toolchain**: Install Rust and the necessary toolchains for embedded development.
2. **Clone the Repository**: Clone the Rust HAL repository from GitHub.

```
git clone git@github.com:UNIT-Electronics-
↪MX/py32f0xx-hal.git
```

3. **Build and Flash: Use Cargo to build your project and flash it to the** PY32F0xx microcontroller using a compatible programmer.
4. **Explore Examples**: Check out the example projects provided in the repository to understand how to use various peripherals and features of the microcontroller.

CHAPTER
SEVEN

SUPPORTED BASIC ARDUINO FUNCTIONS

Packages the basic functions of the Arduino IDE, allowing you to program the DevLab Development Board using the familiar Arduino environment. This includes digital and analog I/O operations, PWM control, serial communication, and more.

```
https://raw.githubusercontent.com/UNIT-
↪Electronics-MX/unit_electronics_py32_
↪arduino_package/main/package_unit_
↪electronics_py32_index.json
```

Learn how to program the **DevLab: PY32F003L24D6TR** development board using **Arduino IDE** and the **CH552 Multi-Protocol Programmer** in **CMSIS-DAP mode**.

- Rapid prototyping
- Embedded systems education
- Wearable device development

These resources are sufficient for a wide range of control, signal acquisition, and communication applications.

7.1 Materials Required

To program the **DevLab: PY32F003L24D6TR** using Arduino IDE, you will need:

- USB Type-C cable
- JST 1.0 mm cable (4-pin)
- Arduino IDE installed on your computer
- DevLab: PY32F003L24D6TR board
- DevLab: CH552 Multi-Protocol Programmer

7.2 Required Installations

This tutorial uses the **pyOCD Debugger**, therefore the following must be installed:

- Python (installed on your system)
- pyOCD (Python package)

7.3 CMSIS-DAP Mode

The programmer must operate in **CMSIS-DAP mode**.

The following repository contains all available firmware images for the Multi-Protocol Programmer. For this tutorial, you only need:

```
cmsis_dap.bin
```

7.3.1 Firmware – CMSIS-DAP

Warning: Before connecting the programmer, make sure the device is powered with **+5 V**. Use the onboard voltage selector switch to choose the correct supply level.

7.4 Flashing the Programmer

Use **WCHISPStudio** to flash the file `cmsis_dap.bin` to the Multi-Protocol Programmer.

Wait until the update process is complete before using the programmer in **CMSIS-DAP mode**.

Refer to the official tutorial for more details:

```
WCHISPStudio Tutorial
```

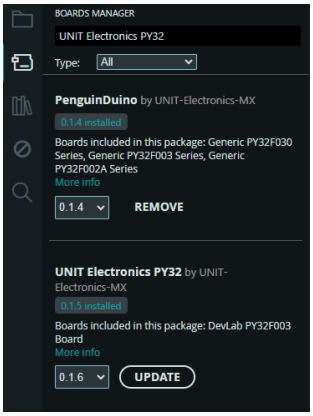
7.5 Additional Boards Manager (Arduino IDE)

1. Open **Arduino IDE**
2. Go to **File > Preferences**
3. In **Additional Boards Manager URLs**, paste the following URL:

```
https://raw.githubusercontent.com/UNIT-Electronics-MX/unit_electronics_py32_arduino_package/main/package_unit_electronics_py32_index.json
```

7.6 Board Manager

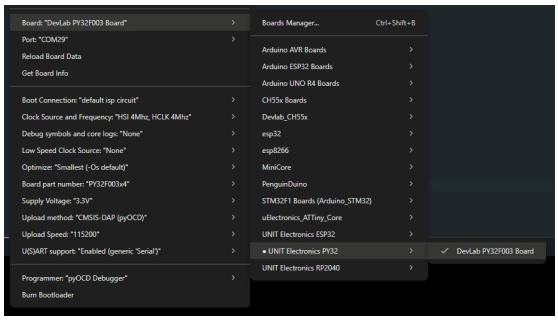
1. Open **Boards Manager**
2. Search for **"UNIT Electronics PY32"**
3. Install the package



7.7 Selecting the Board

In Arduino IDE, select:

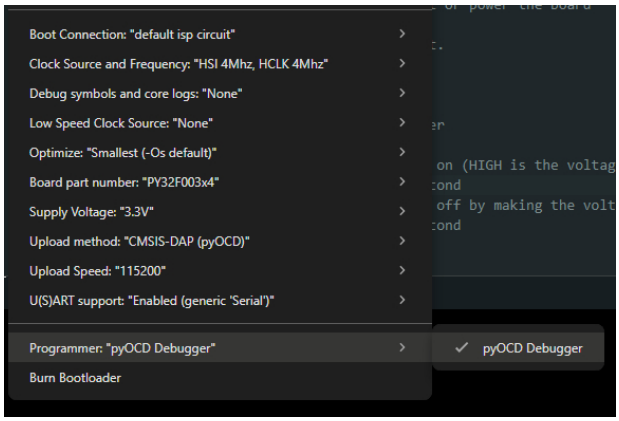
```
Tools > Board > UNIT DevLab: PY32F003
```



7.8 Programmer Selection

Select the programmer:

```
Tools > Programmer > pyOCD Debugger
```



7.9 Connections

With the Multi-Protocol Programmer operating in **CMSIS-DAP mode** and the DevLab board already selected in **Arduino IDE**, follow the steps below.

7.9.1 Step-by-step connection

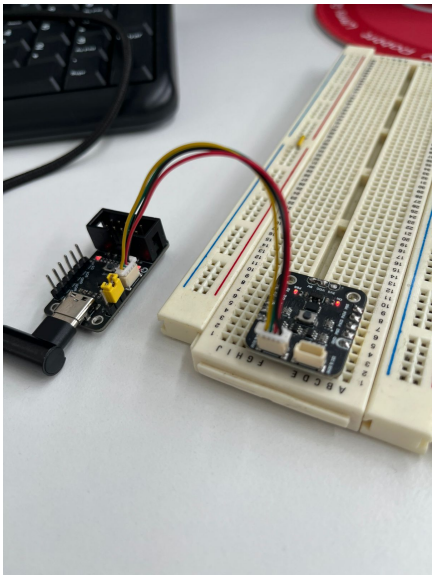
1. Connect the **CH552 Multi-Protocol Programmer** to your computer using a USB cable.
2. Using a **JST 1.0 mm (4-pin) cable**, connect the programmer to the **DevLab: PY32F003L24D6TR** board.
3. Verify that the SWD signals are correctly aligned between both devices.

7.9.2 Required pin alignment

The following signals **must match exactly** between the programmer and the board:

- **VCC** – Target power
- **GND** – Ground reference
- **SWDIO** – SWD data line
- **SWCLK** – SWD clock line

7.9.3 Connection reference



7.10 Example Code (Blink)

Compile and upload the following example to blink the onboard LED.

Note: It is not necessary to select a COM port, since programming is performed via **CMSIS-DAP**.

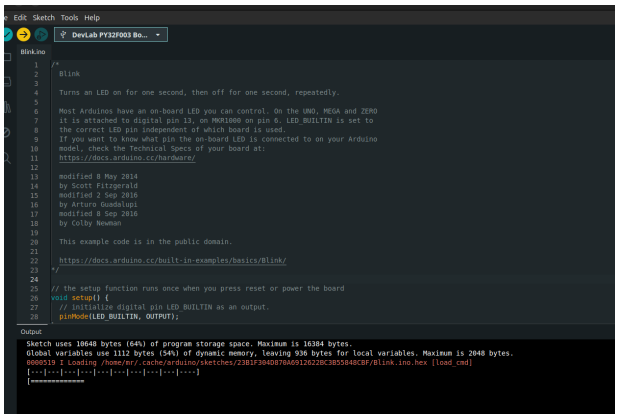
```
void setup() {
  // Initialize digital pin LED_BUILTIN as␣
↪an output
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // LED␣
↪ON
```

(continues on next page)

(continued from previous page)

```
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);   // LED␣
↪OFF
  delay(500);
}
```

Once the connections are completed and verified, the **DevLab: PY32F003L24D6TR** is correctly connected to the **CH552 Multi-Protocol Programmer** and ready to be programmed using **Arduino IDE** via **CMSIS-DAP**.

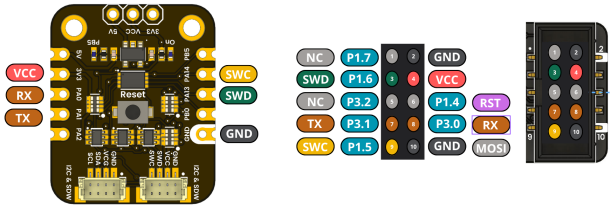


7.10.1 Flashing Firmware operation

CHAPTER
EIGHT

SERIAL COMMUNICATION (UART)

The DevLab PY32F003L24D6TR Development Board offers robust serial communication capabilities, making it ideal for applications requiring reliable data exchange. The board features multiple serial interfaces, including USART, which supports various communication protocols. With a maximum baud rate of up to 115200 bps, the USART interface ensures efficient data transmission for a wide range of applications.



8.1 Example Usage

Here is a simple example of how to set up and use the USART interface on the DevLab PY32F003L24D6TR Development Board using the Arduino IDE:

```
void setup() {
// Initialize serial communication
// PA0 = TX, PA1 = RX automatically␣
↪configured
Serial.begin(115200);

// Startup message
Serial.println("=== PY32F003 Serial Echo␣
↪===");
Serial.println("Type something and press␣
↪Enter");
}

void loop() {
// If there is data available in the␣
↪serial buffer
if (Serial.available()) {
    // Read and echo the character
    char c = Serial.read();
    Serial.write(c);
```

(continues on next page)

(continued from previous page)

```
}
}
```

8.1.1 Overview of Serial Communication

The DevLab PY32F003L24D6TR Development Board offers versatile serial communication through its USART interfaces, supporting UART protocols for efficient data transmission. Key features include:

- Configurable baud rates up to 115200 bps.
- Standard pin assignments, such as PA0 for TX and PA1 for RX on USART2.
- Multiple USART peripherals for flexible connectivity.

These capabilities make the board suitable for a broad range of applications requiring reliable serial data exchange.

CHAPTER
NINE

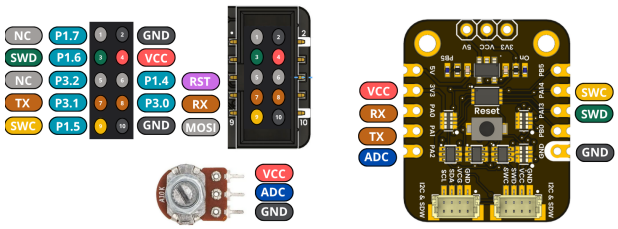
ANALOG TO DIGITAL CONVERTER (ADC)

The PY32F003L24D6TR microcontroller integrated into the DevLab Development Board features a 12-bit Analog to Digital Converter (ADC) that provides high-resolution analog signal measurement capabilities. The ADC supports multiple input channels, allowing users to connect various analog sensors and devices for data acquisition.

9.1 Overview

The PY32F003 microcontroller features a 12-bit Analog-to-Digital Converter (ADC) at the hardware level, providing a theoretical range of 0–4095.

However, when using the Arduino core, the function `analogRead()` is limited to 10-bit resolution by default, returning values in the range 0–1023. This behavior is intentional and follows the traditional Arduino API for compatibility with legacy boards (e.g., AVR-based Arduino UNO).



9.1.1 Default Behavior

```
int value = analogRead(PA2);
// Range: 0 - 1023 (10-bit)
```

Key characteristics:

- **Default ADC resolution:** 10 bits
- **Maximum value:** 1023
- **Hardware ADC resolution (internal):** 12 bits

9.1.2 Changing ADC Resolution

If supported by the Arduino core, the ADC resolution can be modified using:

```
analogReadResolution(12);
```

Example:

```
void setup() {
  Serial.begin(115200);
  analogReadResolution(12);
}

void loop() {
  int value = analogRead(PA2);
  Serial.println(value);  // Range: 0 -␣
↪4095
}
```

Warning: Some Arduino cores for PY32 devices may ignore `analogReadResolution()` and always return 10-bit values.

9.1.3 Compatibility Note

If `analogReadResolution(12)` is not implemented in the core:

- `analogRead()` will always return 10-bit values
- The ADC hardware still operates at 12 bits internally
- The result is scaled or truncated by the core

A simple approximation can be done manually:

```
int adc10 = analogRead(PA2);
int adc12 = adc10 << 2;  // Approximate 12-
↪bit value
```

9.1.4 Summary

Item	Value
Hardware ADC	12-bit
Arduino analogRead() default	10-bit
Output range (default)	0–1023
Output range (12-bit)	0–4095
Recommended for precision	Use HAL / low-level ADC

9.1.5 Recommendation

For applications requiring full ADC precision, direct ADC access via HAL or low-level drivers is recommended instead of the Arduino abstraction layer.

9.2 Example Usage

9.2.1 Complete ADC Example

This example demonstrates ADC reading, LED control, and serial communication:

```
/* ================================
 *  PY32F003 - ADC + LED + Serial
 * ================================
 * ADC  : PA2
 * LED  : PB5
 * UART : PA0 (TX) / PA1 (RX)
 */

int sensorPin = PA2;   // ADC input
int ledPin    = PB5;   // LED output
int sensorValue = 0;

void setup() {
  // GPIO
  pinMode(ledPin, OUTPUT);
  analogReadResolution(12);

  // Serial (PA0 = TX, PA1 = RX)
  Serial.begin(115200);
  delay(100);

  Serial.println("=== PY32F003 ADC +␣
↪Serial + LED ===");
```

(continues on next page)

(continued from previous page)

```
  Serial.println("ADC on PA2 | LED on PB5
↪");
  Serial.println("Type something for echo..
↪.");
}

void loop() {
  /* ===== ADC READ ===== */
  sensorValue = analogRead(sensorPin);  //␣
↪0-4095 (12-bit)

  /* ===== SERIAL OUTPUT ===== */
  Serial.print("ADC Value: ");
  Serial.println(sensorValue);

  /* ===== LED BLINK BASED ON ADC ===== */
  digitalWrite(ledPin, HIGH);
  delay(sensorValue / 4);   // scale delay␣
↪to be reasonable
  digitalWrite(ledPin, LOW);
  delay(sensorValue / 4);

  /* ===== SERIAL ECHO ===== */
  while (Serial.available()) {
    char c = Serial.read();
    Serial.write(c);   // echo back
  }
}
```

9.2.2 Pin Configuration

Function	Pin
ADC Input	PA2
LED Output	PB5
UART TX	PA0
UART RX	PA1

Note: Make sure to connect your analog sensor to PA2 with appropriate voltage levels (0-3.3V).

CHAPTER
TEN

HOW TO GENERATE AN ERROR REPORT

This guide explains how to generate an error report using GitHub repositories.

10.1 Steps to Create an Error Report

1. **Access the GitHub Repository**
   Navigate to the GitHub repository where the project is hosted.
2. **Open the Issues Tab**
   Click on the "Issues" tab located in the repository menu.
3. **Create a New Issue**
   - Click the "New Issue" button.
   - Provide a clear and concise title for the issue.
   - Add a detailed description, including relevant information such as:
     - Steps to reproduce the error.
     - Expected and actual results.
     - Any related logs, screenshots, or files.
4. **Submit the Issue**
   Once the form is complete, click the "Submit" button.

10.2 Review and Follow-Up

The development team or maintainers will review the issue and take appropriate action to address it.