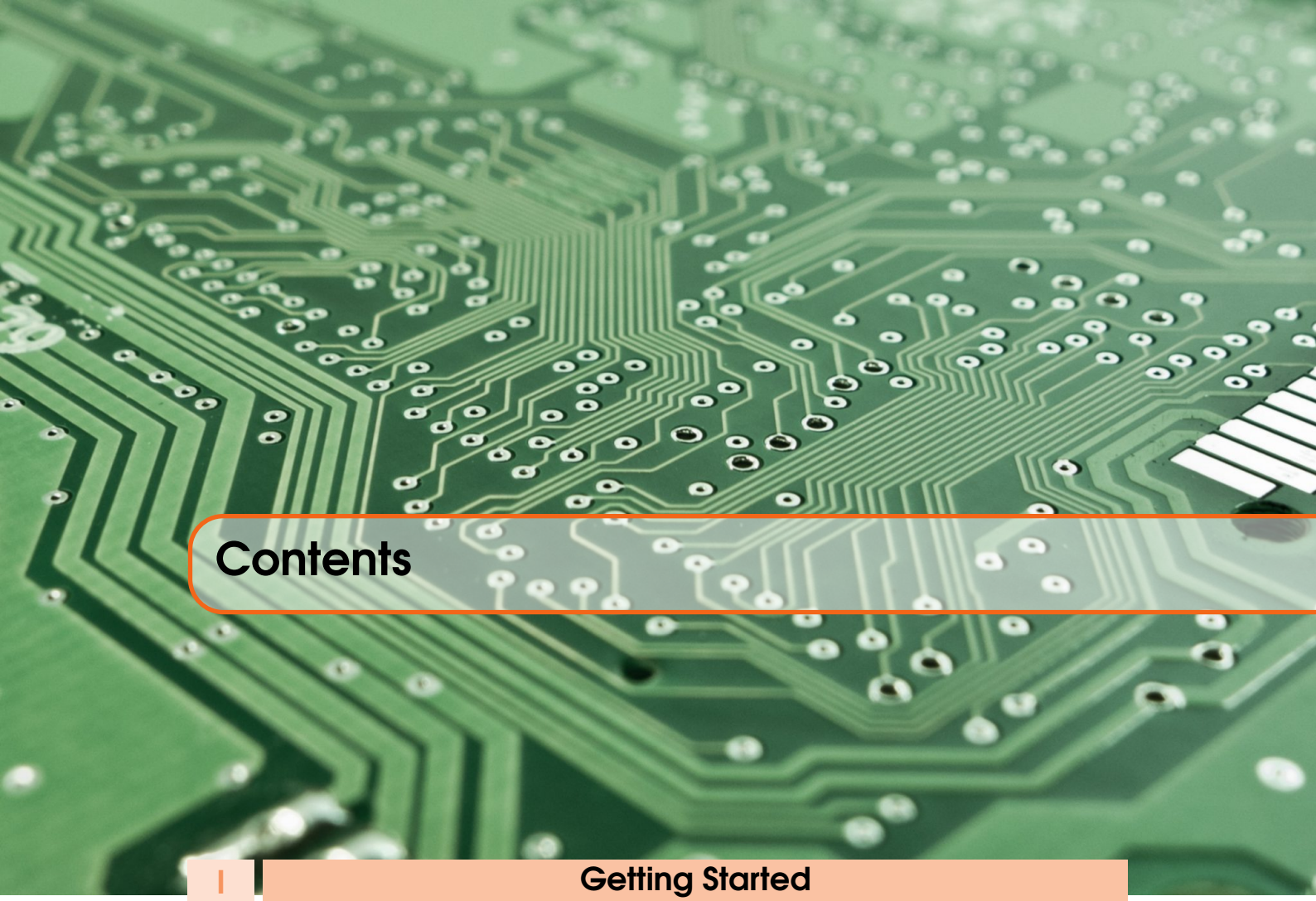
User Manual for LogicTerm 16

Next Generation Test & Debug Tool

Model: LT16M v1.1

PUBLISHED BY LOGICTERM INC.
LogicTerm is an innovative startup focused on affordable solutions for testing and debugging Embedded Systems. Our leading product, the LT16M, is a budget-friendly mixed-signal test and debug station, designed for flexibility and performance. http://www.LogicTerm.com

Draft, Jan 2025



Contents

Getting Started

1. Introduction

1.1 The Concept

Have you ever thought of a system that combines the deterministic performance of RTL languages with the flexibility of high-level scripting? The LT16M bridges this gap by offering a dual-level programming model: low-level digital pattern generation and high-level Python scripting for automation and analysis.

The LT16M, figure 1.1, is a comprehensive mixed-signal test and debug station designed for embedded systems engineers, educators, and hardware developers. It enables precise control over digital signals while allowing advanced data manipulation and visualization through Python.

Figure 1.1: LT16M box front panel

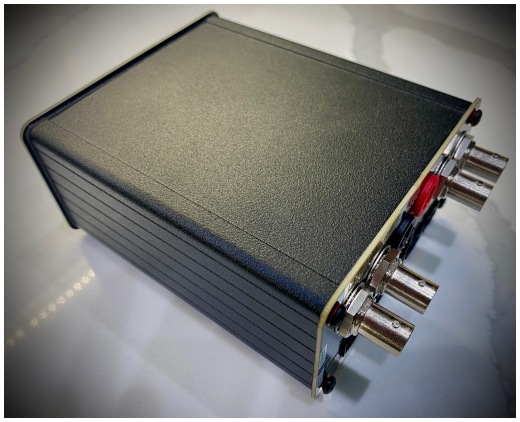


Figure 1.2: LT16M Box

1.2 Use Cases and Capabilities

The LT16M is ideal for:

- Debugging faulty circuits and integrated chips
- Prototyping digital and analog designs

- Mixed-signal waveform generation and capture
- Automated test sequencing and data visualization
- Educational labs and hands-on embedded systems training

1.3 System Overview

The LT16M system consists of:

- **LT16M Hardware Box** – Executes patterns, captures/generates analog signals, and manages power supplies
- **LTStudio IDE** – A Windows-based GUI for editing, compiling, and executing patterns and Python services
- **USB Interface** – Connects the PC to the LT16M for control and data exchange

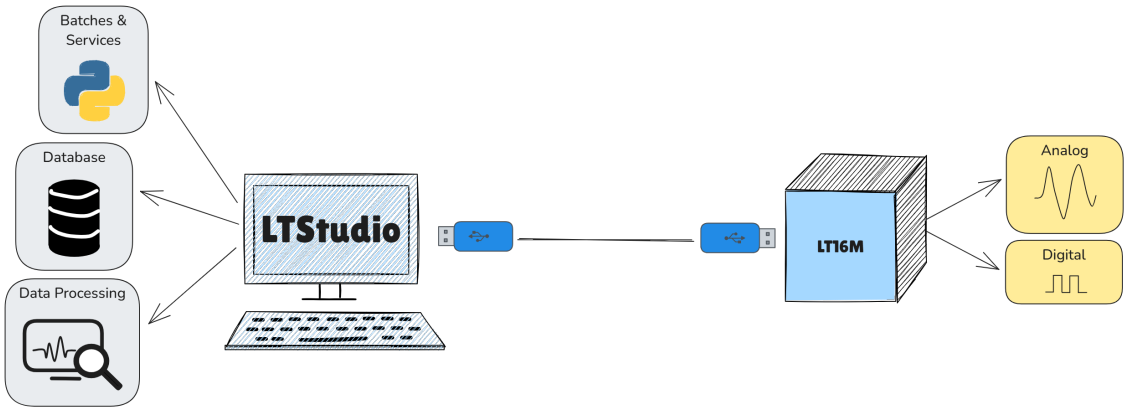
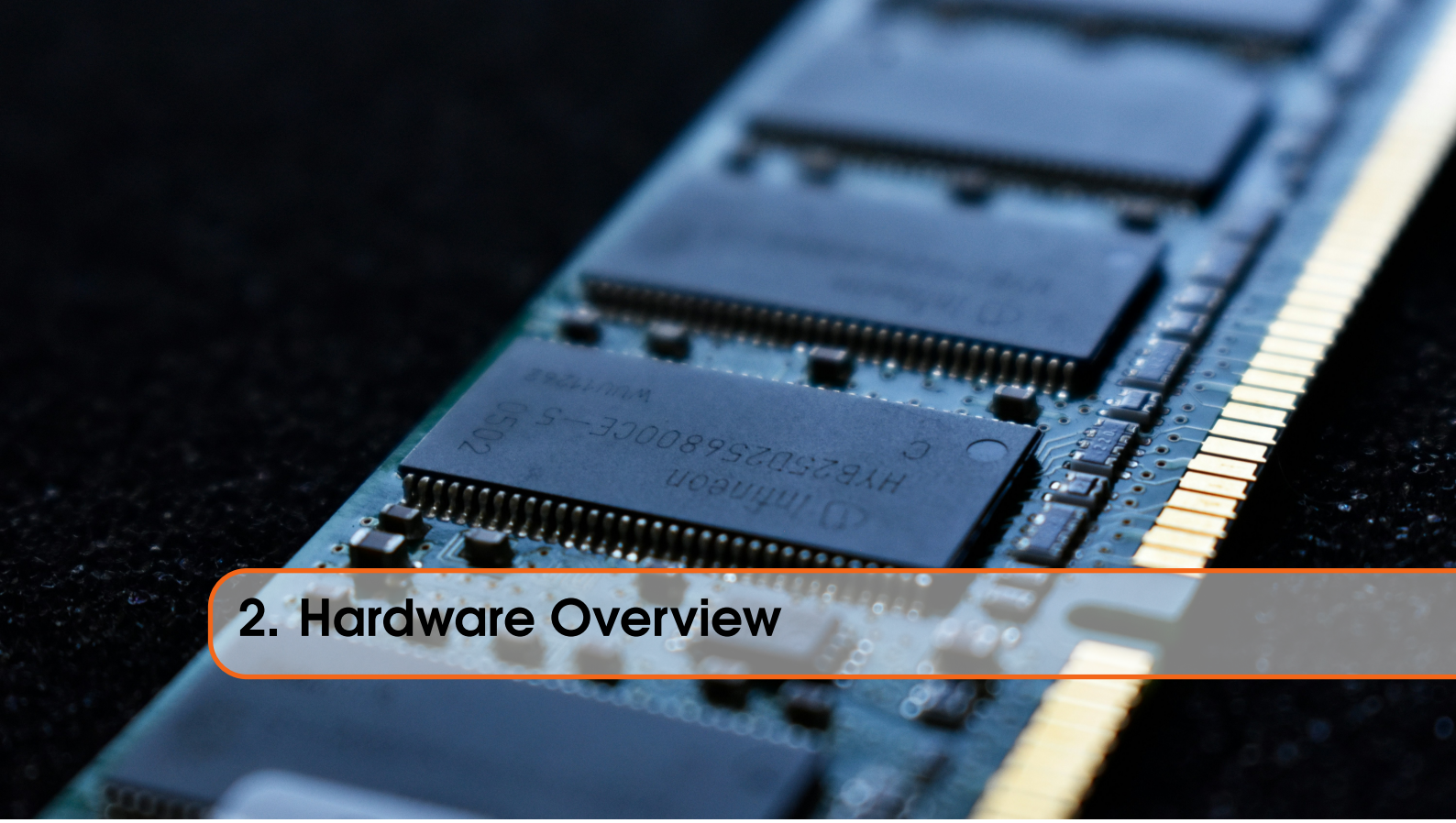


Figure 1.3: Top-level system architecture of LT16M

The LT16M integrates the capabilities of multiple instruments into one platform:

- Logic Analyzer
- Pattern Generator
- Oscilloscope
- Programmable Power Supply

This manual will guide you through setup, programming, data logging, and practical examples to help you make the most of LT16M.



2. Hardware Overview

2.1 LT16M Box

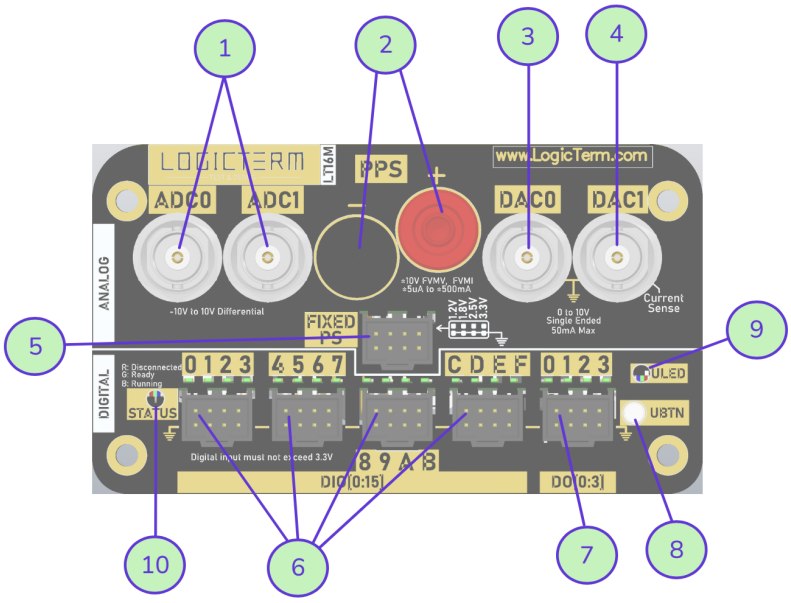


Figure 2.1: Front panel of LT16M

The LT16M hardware is a compact, mixed-signal test and debug station designed by LogicTerm. It integrates digital and analog I/O, power supplies, and user interface elements into a single enclosure.

- **16 bidirectional digital I/O (DIO)** pins with LED status indicators ⑥
- **4 digital drive-only (DO)** pins ⑦
- **2 high-speed ADCs** (±10V, 200 KS/s, 16-bit) ①
- **2 DACs** (0–10V, 1 MS/s, 16-bit) ③ ④
- **Programmable Power Supply (PPS)** with 7 current ranges and clamping ②

- **Fixed power rails:** 3.3V, 2.5V, 1.8V, 1.2V @ 200mA ⑤
- **User I/O:** RGB LED (ULED) ⑨ and push button (UBTN) ⑧
- **System status indicator** can be Red (Disconnected), Green (Ready), or Blue (Running) ⑩

2.2 Digital I/O Capabilities

- Fully programmable at 100 MS/s real-time sample rate for DIOs
- Fully programmable at 25 MHz update rate for DOs
- 1K-entry pattern buffer (16-bit wide)
- 4K sample capture buffer
- Selectable LVCMOS levels: 1.8V, 2.5V, 3.3V (via jumper, figure 2.2)
- 1mA source/sink per pin
- Integrated LEDs for all 20 digital pins

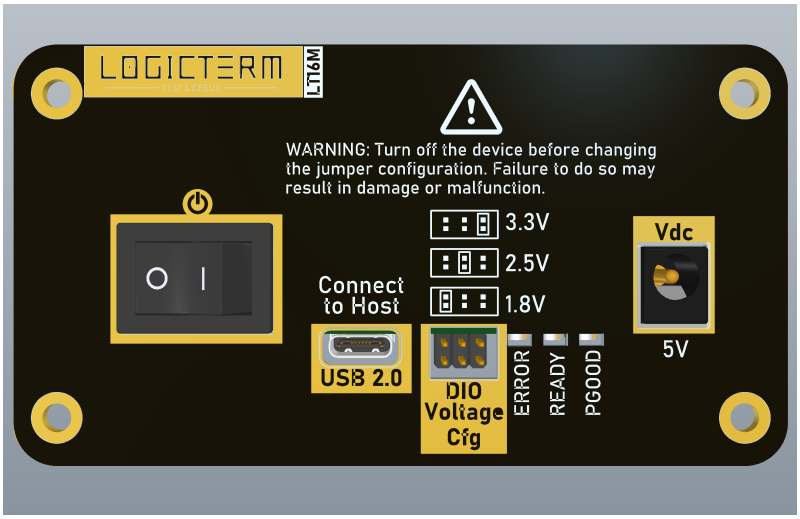


Figure 2.2: Back panel of LT16M showing jumper configuration

2.3 User I/O

- **UBTN:** User push button checked by pattern or service
- **ULED:** RGB LED controlled via pattern or service

2.4 Analog Capabilities

Analog-to-Digital Converters (ADC)

- Two differential channels
- ±10V input range
- 200 KS/s, 16-bit resolution
- 0.3 mV resolution, ±1% accuracy
- 128 dB CMRR, 94.2 dB typical SNR
- 1000 MΩ ‖ 3 pF input impedance

Digital-to-Analog Converters (DAC)

- Two single-ended channels
- 0–10V output range

- 1 MS/s update rate
- Max output current: 50 mA
- 16-bit resolution, 0.1 mA current resolution (DAC1 only) [1]

2.5 Power Supply

- **Fixed Power Rails:** 3.3V, 2.5V, 1.8V, 1.2V @ 200mA maximum current each

Programmable Power Supply (PPS)

- **Voltage Range:** three ranges covering ±10V
- **Current Ranges:** ±5 µA, ±25 µA, ±250 µA, ±2.5 mA, ±25 mA, ±250 mA, to +500/-250 mA (7 ranges)
- **Modes:** Force Voltage / Measure Voltage (FVMV), Force Voltage / Measure Current (FVMI)
- **Features:** Current clamping, programmable compliance

2.6 Safety Notes

Warning 1 Always power off the LT16M before changing jumper settings. Failure to do so may result in hardware damage or incorrect voltage levels. ■

[1] DAC1 output voltage can drop up to 5mV depending on the load.

3. Software Overview

3.1 LTStudio IDE

LTStudio is a Windows-based integrated development environment (IDE) designed to control the LT16M hardware. It allows users to:

- Compile and load digital patterns
- Execute Python services and batch scripts
- Visualize and export logged data
- Manage projects and object libraries

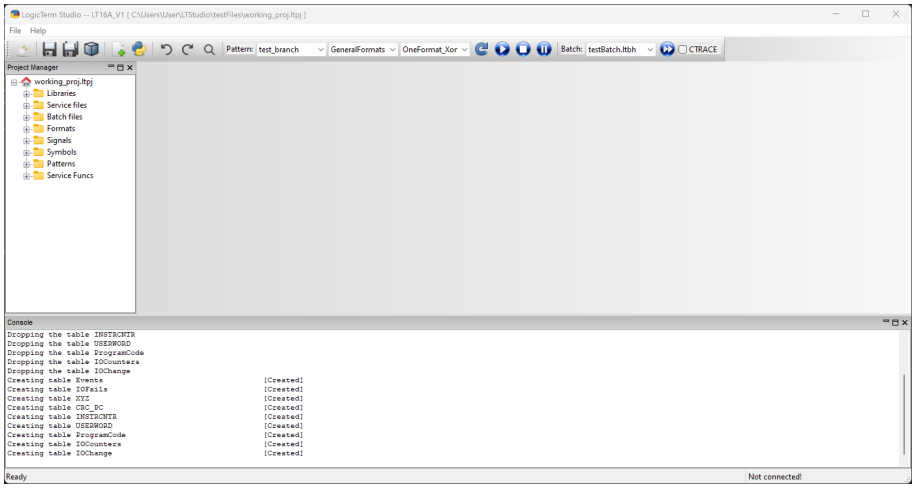


Figure 3.1: LTStudio startup screen

3.2 LTStudio Control Menu

1. Connect: connects LTStudio to the LT16M hardware box via a USB cable. LT16M Status LED changes color from Red to Green when connected.

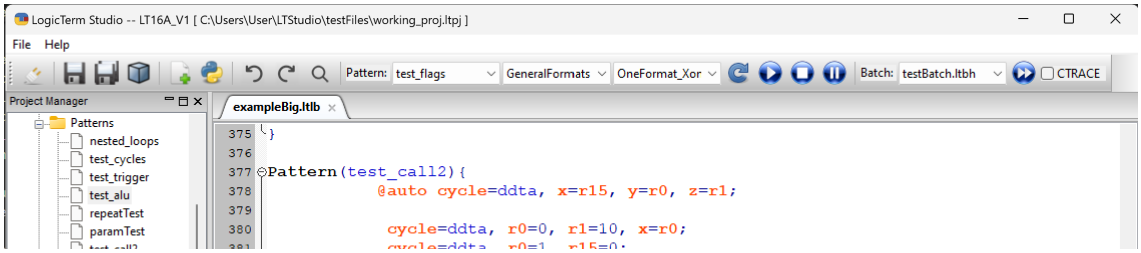


Figure 3.2: LTStudio control button bar

2. Save project: save all files into project and compile all.
3. Add object: add a new LT16M library or batch file to current project.
4. Add service: add a service file to current project.
5. Undo: undo last change
6. Redo: redo last change after it was undone
7. Find: search in current open tab
8. Pattern drop-down menu: choose one of the current compiled patterns.
9. Formats drop-down menu: choose one of the current compiled formats.
10. Signals drop-down menu: choose one of the current compiled signals.
11. Reset: reset digital pattern generator. It will bring the pattern generator to a known state.
12. Start pattern: start the current selected pattern/formats/signals.
13. Stop pattern: stop the running pattern.
14. Pause pattern: pause the running pattern.
15. Batch dropdown menu: choose one of the current batches.
16. Start batch: start the selected batch.

3.3 Supported File Types

LTStudio supports the following file extensions:

- `.ltlb` – Library files containing Formats, Signals, and Patterns
- `.ltpy` – Python service files for logging, plotting, and control
- `.ltbh` – Batch scripts for automated test sequences

3.4 Installation and USB Driver Setup

To connect LT16M to your PC, follow these steps:

1. Install LTStudio from `https://www.LogicTerm.com/download/LTStudio`
2. Install Zadig v2.9 or later from `https://zadig.akeo.ie`
3. Connect LT16M via USB-C to a USB 2.0 port
4. Power on the LT16M (Status LED turns red)
5. Open Zadig and select “List All Devices”
6. Choose `LT16D (Interface 1)` and install `libusb-win32`, as seen in Figure 3.3.

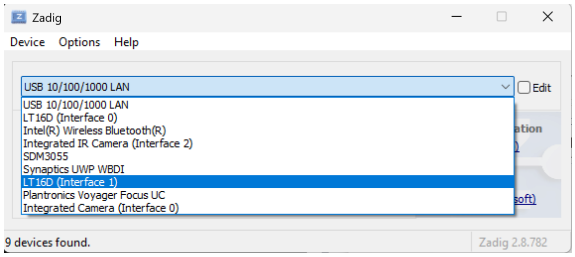


Figure 3.3: Zadig showing two interfaces for LT16M.

7. Disconnect and reconnect the USB cable
8. In Device Manager, update `LT16D (Interface 0)` to use `USB Serial Converter A`, it is different. For a successful setup, the device will appear in the device manager as Figure 3.4.
9. Launch LTStudio and press “Connect” — the Status LED should turn green

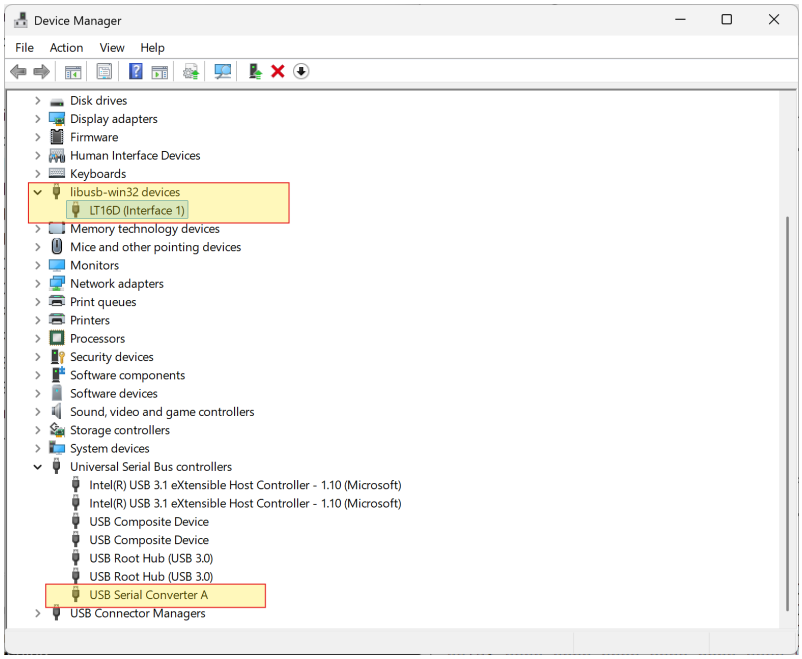


Figure 3.4: LT16M driver configuration in Device Manager

4. First-Time Setup

4.1 Connecting the LT16M to a PC

To set up the LT16M, first connect the supplied power adapter, then link the device to your PC using the included USB-C cable. Confirm that the unit is switched on, start LTStudio, and select "Connect." The LT16M status LED displays red when idle and changes to green after LTStudio establishes a successful connection.

> **Warning 2** Use only the supplied power adapter. Using other power supplies may damage the device or impair its performance. ■

4.2 Jumper Configuration

The LT16M supports three digital I/O voltage levels: 1.8V, 2.5V, and 3.3V. These are selected using a jumper on the back panel, refer to Figure 2.2.

> **Warning 3** Always power off the LT16M before changing jumper settings. Changing voltage levels while the device is powered may cause permanent damage. ■

4.3 Verifying USB and Driver Installation

After connecting the LT16M:

1. Open Device Manager on your PC, as seen in Figure 3.4.
2. Under `libusb-win32 devices`, confirm that `LT16D (Interface 1)` is listed.
3. Under `Universal Serial Bus controllers`, confirm that `USB Serial Converter A` is installed for Interface 0.

4.4 Using LTStudio: Running your first pattern!

After installing the drivers, follow these steps to run your first pattern:

1. Open LTStudio from the Start Menu or desktop shortcut. Figure 3.2 illustrates the toolbar buttons available for different functions.
2. Power on LT16M and connect USB-C cable.
3. Click the **Connect** button in the toolbar.
4. Verify that the LT16M status LED turns green, confirming a successful connection.
5. Navigate to `File` → `New Project`, name the project `HelloWorld`, and select the desired directory.
6. Create a new library via `File` → `New Library`, naming it `HelloWorld_lib`.
7. Go to `Utilities` → `Templates` → `Patterns` → `Hello World Full Example - Pattern`, then copy the template text.
8. Paste the template text into `HelloWorld_lib`, save the current project, and note that the new pattern, formats, and signals now appear in their respective drop-down menus.
9. Select `helloWorldPattern`, `helloWorldFormats`, and `HelloWorldSignals`, then click `Start Pattern`.
10. Observe that the pattern maps the 16-bit counter to the 16 DIOs, causing the DIO LEDs to blink at varying rates.
11. Confirm that the LT16M status LED turns blue, indicating the system is running.
12. To stop the pattern, either press `Stop Pattern` or hold the `UBTN`.

4.5 Using LTStudio: Running your first service!

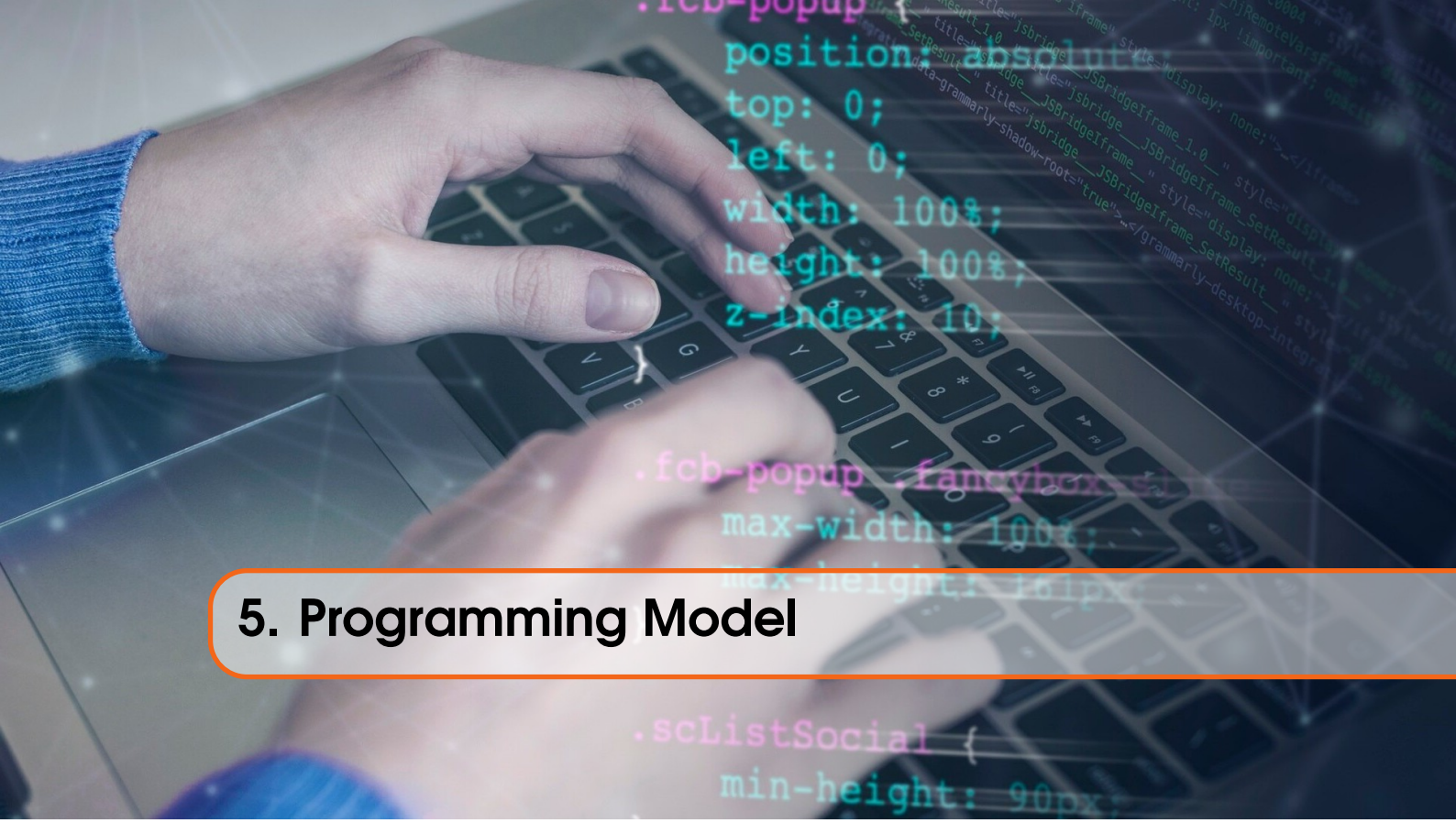
The simplest way to call a hardware function outside a pattern is to call it from the "Interactive Shell" in LTStudio. You can type `hw.setUSERLED(0xff, 0, 0)` and hit enter to execute. You should see the ULED in red. You may change the USERLED section 6.5.1. For creating a custom service and call it from a pattern, please follow these steps.

1. Make sure you are connected to the LT16M.
2. Create a new service file via `File` → `New Service`, naming it `HelloWorld_services`.
3. Go to `Utilities` → `Templates` → `Services` → `echo`, then copy the template text.
4. Paste the template text into `HelloWorld_services`, save the current project, and note that the new service now appears under services in project manager.
5. `Hello World Full Example - Service`
6. helloWorldPattern2 helloWorldFormats HelloWorldSignals helloWorldServices

4.6 Using LTStudio: Running your first batch!

You are now ready to explore patterns, signals, and services within LTStudio.

II Programming Architecture



5. Programming Model

5.1 Dual-Level Execution

The LT16M system is built around a dual-level programming model that combines deterministic digital control with flexible high-level scripting:

- **Level 1: Pattern Execution on LT16M** Patterns are compiled and executed directly on the LT16M hardware. These patterns define precise digital signal behavior using a custom instruction set. They are ideal for timing-critical operations and deterministic control.
- **Level 2: Python Services on Host PC** Python scripts run on the host PC via LTStudio. These services can extract logged data, visualize results, control execution flow, and automate batch operations. They provide a high-level interface for analysis and orchestration.

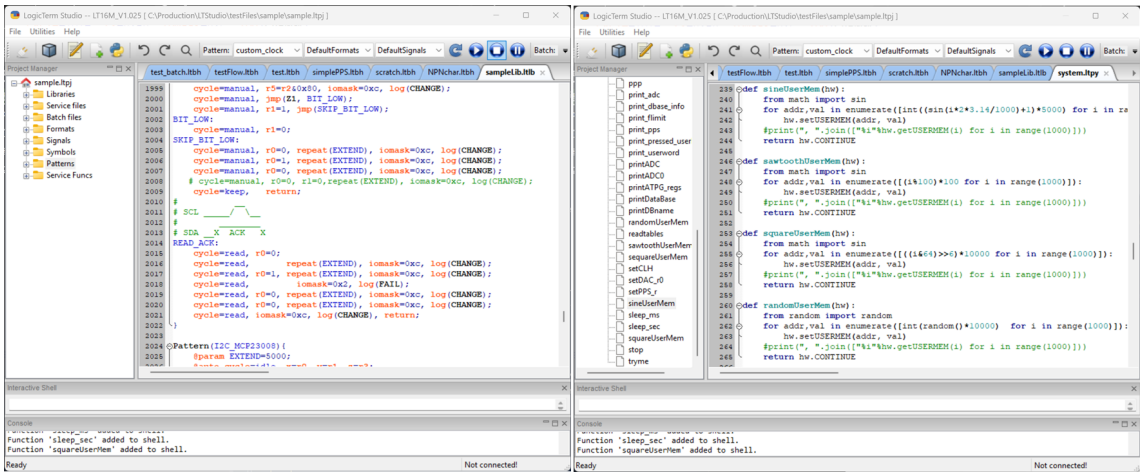


Figure 5.1: Dual-level programming model of LT16M. (Left) writing a pattern for digital signals and low-level conditioning. (Right) writing a service script for mixed signals interaction and high-level data manipulation.

5.2 Object Flow Architecture

The LT16M programming workflow is modular and object-oriented. Each test or debug session typically involves the following components:

- **Formats** – Define timing and signal structure across cycles
- **Signals** – Map physical DIO pins to logical pattern outputs
- **Patterns** – Generate digital waveforms and call services
- **Services** – Python functions for logging, plotting, and control with specific return
- **Batches** – Python scripts that automate execution of multiple objects

This architecture allows users to build reusable libraries, automate test sequences, and integrate digital and analog operations seamlessly.

6. Programming Objects

6.1 Overview

LT16M uses a modular object model to define, execute, and automate digital and analog tests. Each object type plays a specific role in the signal generation and data logging pipeline.

6.2 Formats

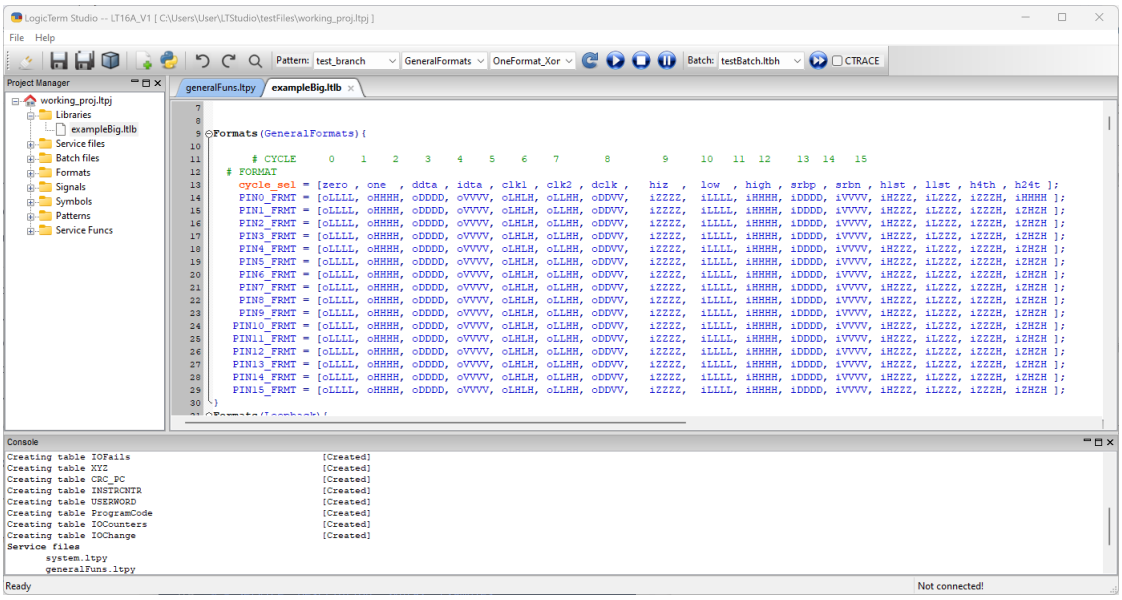


Figure 6.1: LTStudio Formats

Formats define the timing and structure of digital signals. Each format can contain up to 16 cycle configurations, and each cycle can specify output values, masks, and control flags.

- Up to 16 format definitions per pattern

7. Database Architecture

7.1 Overview

LT16M uses an in-memory SQLite database to log digital and analog data during pattern execution. This architecture enables fast access, structured grouping, and seamless export to external formats.

7.2 Database Structure

The database is composed of multiple tables that capture different aspects of execution:

- **Records** – Raw digital and analog samples
- **Groups** – Logical grouping of records by test phase or pattern
- **GroupsInfo** – Metadata for each group (name, timestamp, pattern ID)
- **Info** – Instruction info
- **IOChange** – Changes to pins
- **AnalogData** – Captured ADC samples with timestamps
- **IOCounters** – Per-pin counters for toggles, transitions, and activity
- **IOFails** – Pin-level failure logs

7.3 View Tables

LTStudio provides simplified views for quick access to grouped data, as seen in figure 7.2.

7.4 Exporting Data

Users can export logged data from LTStudio in multiple formats:

- **Excel (.xlsx)** – Tabular export for analysis and reporting
- **CSV (.csv)** – Lightweight format for scripting and automation
- **SQLite (.db)** – Full database export for advanced querying

8. Quick Start Examples

8.1 LED Blinking

This example demonstrates how to blink a single LED connected to a DIO pin. We will try using different ways to acheive that in order to explore the tool's capability and flexibility.

Setup

- Connect an LED (with series resistor) to `DIO[0]` or rely on the DIO integrated LED
- Set jumper to 3.3V logic level

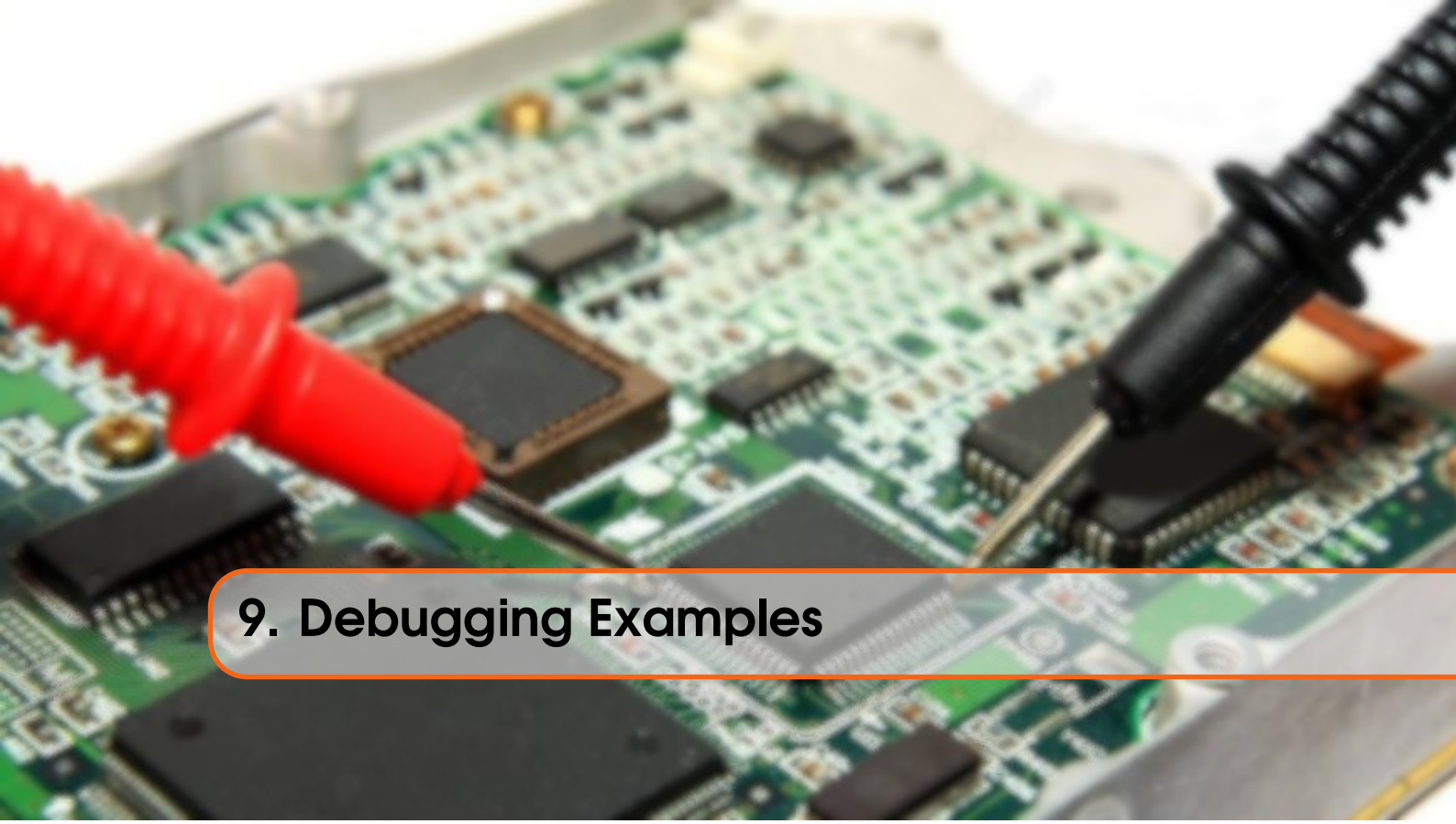
8.1.1 LED Blinking by changing the cycle

In this example, we define a single format with two cycles. One cycle, called `led_off`, is configured as an **o**utput with all four ticks set to **L**ow. On the other hand, the `led_high` cycle is set to **H**igh.

When an LED is connected to the pin `DIO[0]` of the LT16M, the Signals **mySig** define what data source should be used for this pin and what format.

When the pattern myTest starts executing, the first instruction uses the cycle `led_off` which sets the `DIO[0]` to 0 for all 4 ticks (40 ns). The second instruction uses the cycle `led_on` which sets the connected pin 0 to 1 for all 4 ticks (40ns). In addition, it jumps to the instruction at label `AGAIN`. In summary, this will cause the pattern to toggle infinitely. Unfortunately, the toggle at 12.5MHz is too fast for humans to see.

```
1  Formats(myFrmt){
2          # CYCLE          0       1
3      # FORMAT
4        cycle_sel  = [ led_off, led_on];
5        LED_FRMT   = [  oLLLL,  oHHHH ];
6      }
7
8  Signals(mySig){
9      A1    = dio(pin=0, map=0, format=LED_FRMT);
10     }
11
12 Pattern(led_blinking){
13     AGAIN: cycle=led_off;                # 1 Cycle ON  takes 40ns
```



9. Debugging Examples

9.1 Detecting Stuck-at Faults

This example checks whether a pin is stuck at logic high or low.

Setup

- Connect DIO2 to the node under test
- Ensure the node is driven externally

Pattern

```
Pattern(stuck_pin){
# TODO:
cycle=low;
cycle=low;
cycle=low;
cycle=low, service(stop_pattern(hw));
}
```

Expected Behavior

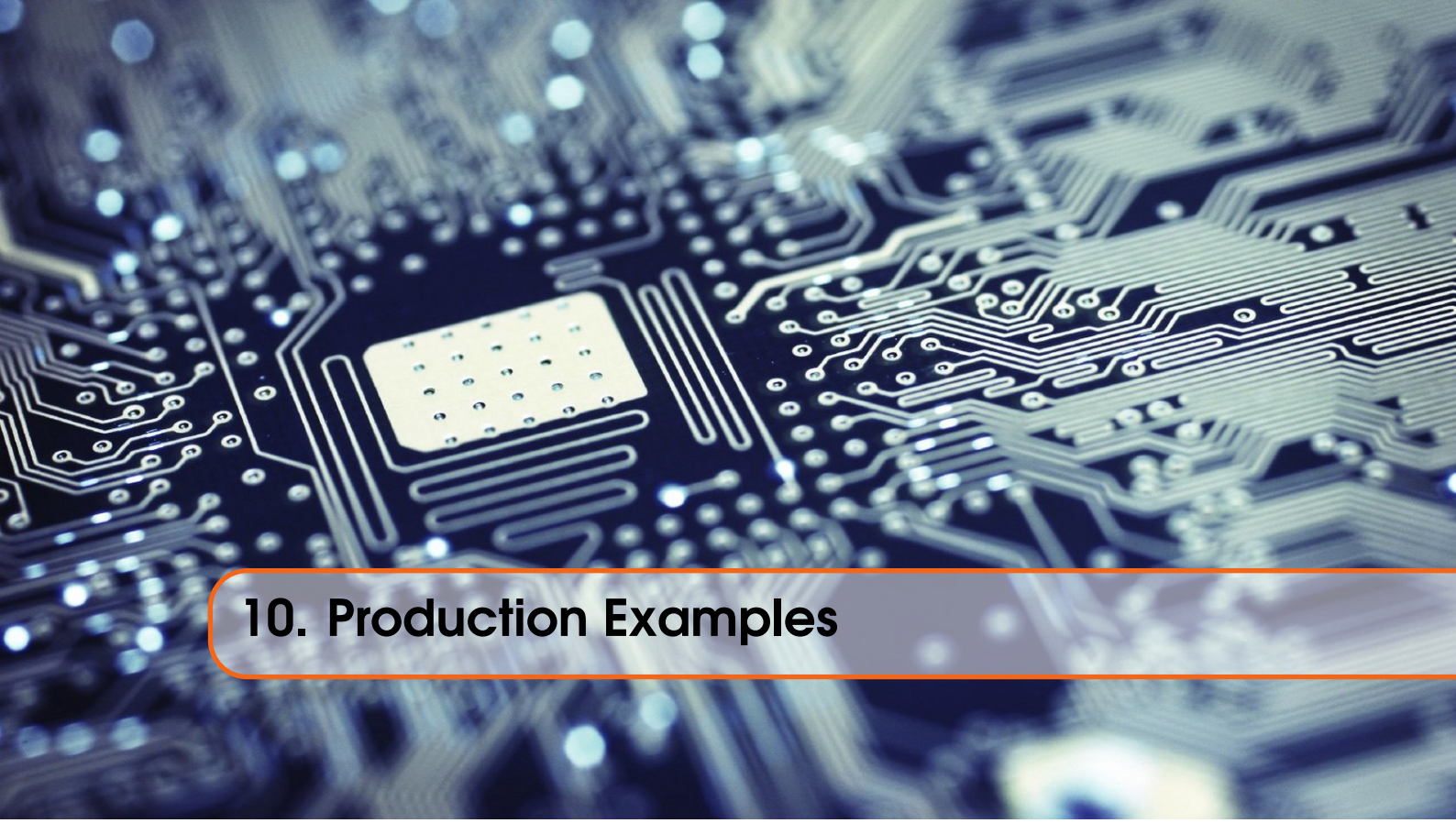
If DIO2 reads the same value repeatedly, the pattern jumps to ‘Fault‘ and calls a Python service to report the issue.

9.2 Open Circuit Detection

This example verifies whether a pin is floating or disconnected.

Pattern

```
Pattern(open_test){
# TODO:
cycle=low;
cycle=low;
```



10. Production Examples

10.1 Automated Test Sequence

This example runs a full test cycle using a batch script that loads patterns, logs results, and calls services.

Batch Script

```
run("PowerOnSelfTest");
wait(1);
run("FunctionalTest");
call("log_results");
call("export_to_excel");
```

Expected Behavior

Executes a power-on test, followed by a functional test, then logs and exports results.

10.2 Reusable Pattern with Parameters

This example uses a parameterized pattern to test multiple devices with different configurations.

Pattern

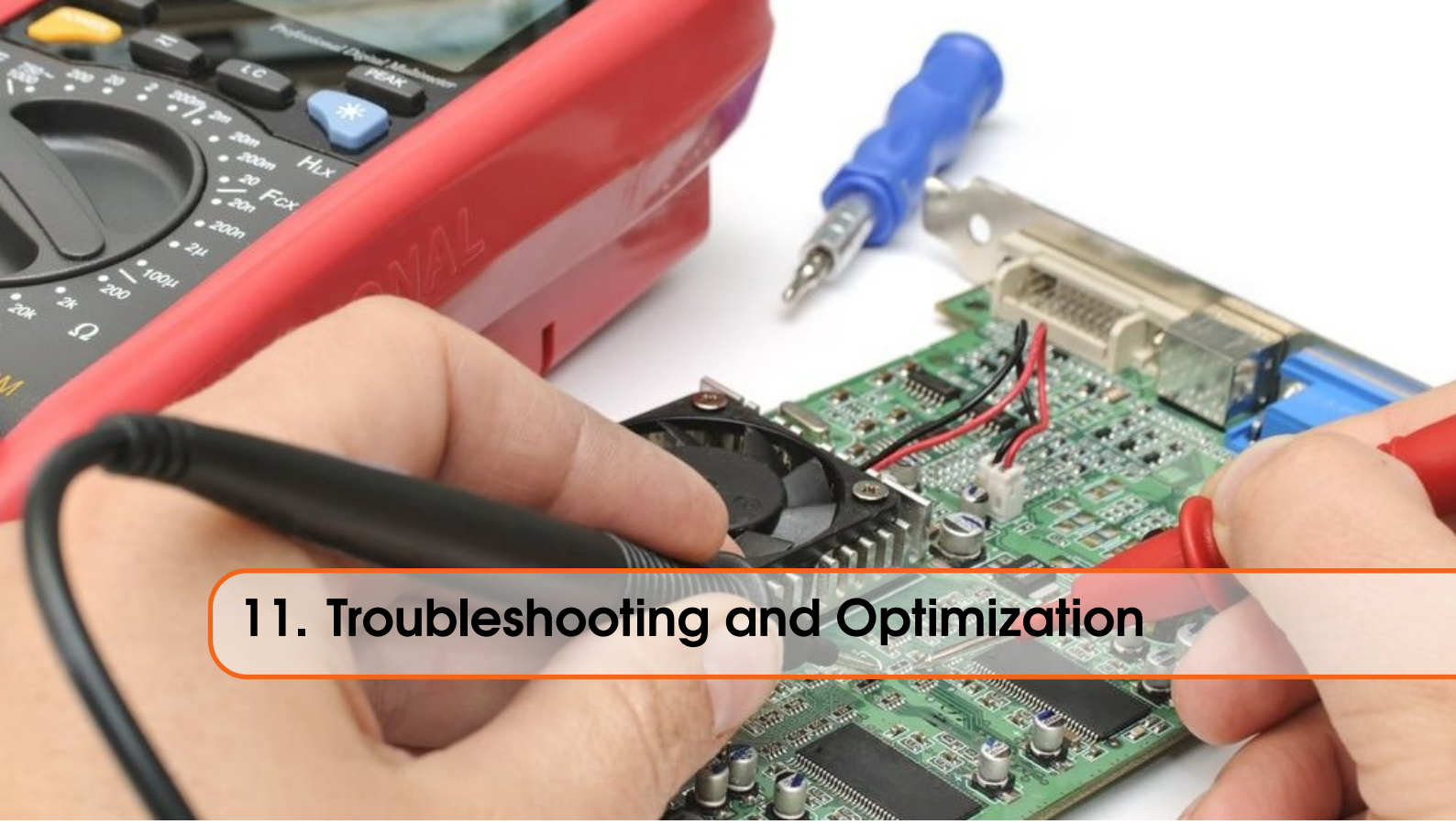
Batch Invocation

```
set("VSET", 2.5);
run("DeviceTest");

set("VSET", 3.3);
run("DeviceTest");
```

10.3 Logging and Exporting Results

This example shows how to log data and export it to Excel for reporting.



11. Troubleshooting and Optimization

11.1 Connection Issues

If LTStudio fails to connect to LT16M:

- Verify USB-C cable and port (use USB 2.0 if possible)
- Confirm power is on (status LED should be red or green)
- Check Device Manager for correct drivers:
  - Interface 1: `libusb-win32`
  - Interface 0: `USB Serial Converter A`
- Reinstall drivers using Zadig if needed
- Restart LTStudio and reconnect

11.2 Pattern Execution Failures

Common causes:

- Syntax errors in pattern code
- Missing signal or format definitions
- Invalid register usage or branching
- Unreachable service calls

Tips

- Use LTStudio's compiler messages to locate errors
- Test patterns incrementally
- Validate signal mappings before execution
- Verify with known patterns

11.3 Signal Integrity

To improve digital signal quality:

- Use short, shielded cables

12. Definitions

12.1 Object Types

- **Format** – Defines timing and signal behavior
- **Signal** – Maps logical bits to physical pins
- **Pattern** – Sequence of instructions executed on LT16M
- **Service** – Python function executed on host PC
- **Batch** – Python script automating multiple objects

12.2 Formats Syntax

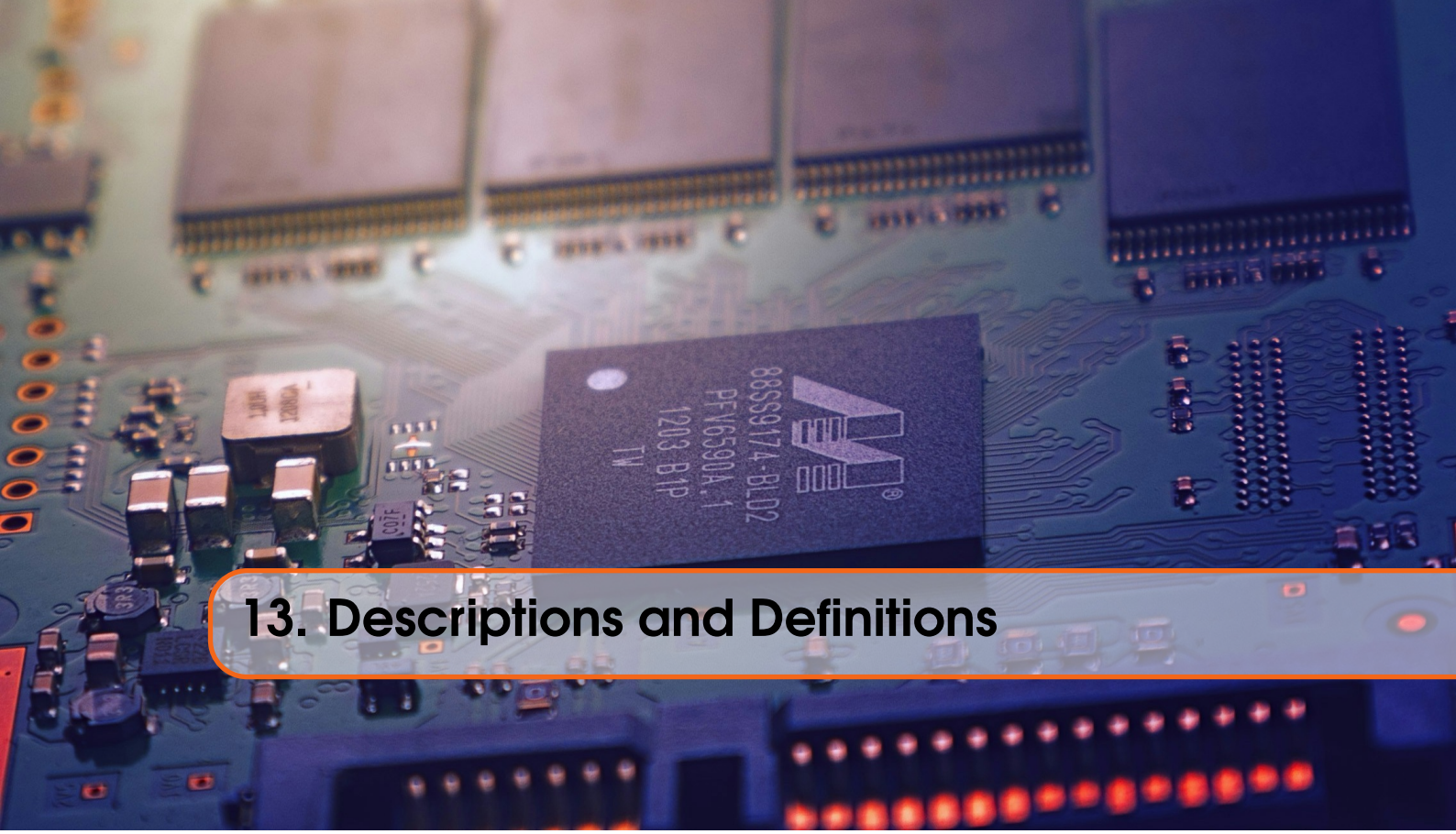
```
Formats Object Syntax:
======================
    FormatsObj    := Formats(<validName>){<format_header>; <formatSel_def; ...>}
    validName     := [a-zA-Z_][a-zA-Z0-9_]*
    list(<items>) := [<item0>, <item1>, ...]

    format_header := cycle_sel=list(<validNames>);
    formatSel_def := validName=list(<cycle_formats>);
    cycle_format  := <dir><tick><tick><tick><tick>
    dir           := [io]
    tick          := [LHDVKTMZ]

    Reserved Keywords:
         Formats cycle_sel
```

12.3 Signals Syntax

```
Signals Object Syntax:
======================
    SignalsObj    := Signals(<validName>){<[PinAssignment], ...>}
    validName     := [a-zA-Z_][a-zA-Z0-9_]*
    PinAssignment := PinLabel=pinType(pin=<PinNum>, map=<DataSource>, format=<FormatSel>);
    PinLabel      := validName
                  := Must be a valid variable name, e.g., SEL, CS1, WR_EN, _PIN1.
    pinType       := [dio]
```



13. Descriptions and Definitions

13.1 Descriptions and Definitions

ADC – Analog-to-digital converter
ATPG – Algorithmic Test Pattern Generator
DAC – Digital-to-analog converter
DIO – Digital input/output pin
DO – Digital Drive-only pin
LT16M – LogicTerm Mixed-signal 16-pin box
LT16D – LogicTerm analog carrier board
LT16A – LogicTerm digital base board
PPS – Programmable power supply
UBTN – User button
ULED – User RGB LED

13.2 Glossary

- **Pattern** – A sequence of instructions executed on LT16M
- **Format** – Defines signal behavior per cycle
- **Signal** – Maps logical bits to physical pins
- **Service** – Python function executed on host PC
- **Batch** – Python script that automates multiple objects
- **log** – Micro-instruction to capture data

Index

I

J

L

M

O

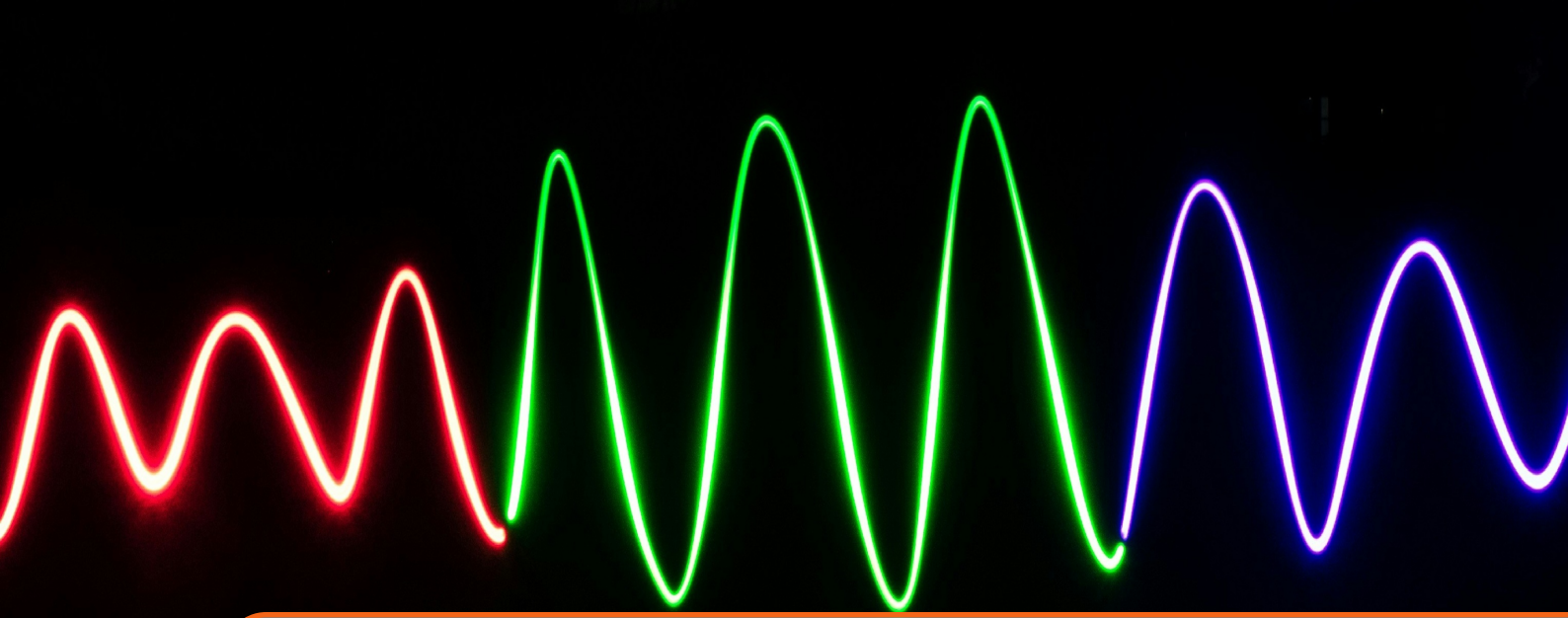
P

R

S

U

W



14. Version History

14.1 Version History

- v1.0 – Initial release with core features
- v1.1 – Added PPS current clamping and ADC enhancements
- v1.2 – Improved LTStudio UI and batch scripting support