ARM-Embedded-Path

Hardwaretests for STMF103CxT6

Pavel Pys

October 26, 2025

Overview

- Motivation
- Configuration
- Manual

Why Chip Verification Matters

Identifying Counterfeit STM32F103 Microcontrollers

Focus: STM32F103C6T6A & STM32F103C8T6

The Reality

Counterfeit Chips in the Supply Chain

The Problem is Real and Growing:

- ERAI reported **786** cases in **2023** and **1,055** in **2024** (highest level since 2015) [ERAI 2023/2024]
- "Blue Pill" boards with STM32F103 are particularly affected by clones/drop-in replacements
- Occurrences also on major marketplaces (AliExpress, eBay, Amazon)
- Even reputable distributors can unintentionally supply non-original parts

Common Scenarios:

- Remarking: *C6* sold as *C8* (lower density higher margin)
- Wafer rejects, refurbished parts
- Pin-compatible clones with different behavior (GD32, APM32, CH32, CS32)
- Completely different chips with counterfeit markings

Bottom Line: Never trust markings alone always verify!

Impact on Development
Why Should You Care?

1. **Debugging Nightmares:**
   - Intermittent bugs that make no sense
   - Code works on one board, fails on another
   - Hours wasted chasing phantom issues
   - Features that simply don't work (DMA, USB, ADC channels)
2. **Product Reliability:**
   - Field failures that are impossible to reproduce
   - Temperature/voltage instability
   - Unexpected resets or crashes
   - Warranty claims and customer dissatisfaction
3. **Time and Money:**
   - Wasted development hours debugging non-genuine chips
   - Production delays when issues discovered late
   - Cost of rework and board replacements

Example Scenario
Flash Size Verification

Common Issue: Tooling reports 64/128 KB, actual usable size differs

Verify via Flash Size Register (authoritative): Real-World Blue Pill Observations (Community):

- Tools/programmers sometimes report 128 KB, Flash Size Register shows 64 KB
- Clones (e.g., CS32) with different IDCODEs/USB behavior reported
- Writing beyond actual size causes corruption/crashes

STM32F103C6 vs C8

Understanding the Variants

Feature	C6T6A	C8T6
Flash Memory	32 KB	64 KB
SRAM	10 KB	20 KB
DBGMCU DEV_ID	**0x412** (Low density)	**0x410** (Medium density)
Density	Low	Medium
Price (typical)	Lower	Higher

Why C6C8 Remarking?

- Price difference, same 48-pin LQFP package, identical marking area

Common Drop-in Alternatives:

- **GD32F103** (GigaDevice), **APM32F103** (Geehy), **CH32F103** (WCH), **CS32F103**
- Pin-compatible but *not* bit-identical; timing/peripherals may differ

Failure Modes

Common Issues with Non-Genuine/Clone Chips

Missing/Deviating Features (Observations):

- DMA channels unreliable
- Fewer/different number of ADC channels
- USB enumeration/function differs (community reports, e.g., CS32)
- CAN/RTC oscillator issues

Electrical Characteristics:

- Higher power consumption, different limits
- Temperature/power supply sensitivity
- Poorer EMC/EMI performance

Timing/Performance:

- HSI accuracy/clock drift
- Flash wait states required at high clock frequencies
- Interrupt latencies/peripheral timing differ

Business Impact
Example Cost Calculation

Example Scenario (Assumptions):

- Small production batch: 100 units
- Non-genuine chips cause 20% field failure rate
- Cost per RMA: $50 (shipping, diagnosis, replacement)
- Engineering debugging time: 40 hours @ $75/hr

Example Cost Breakdown:

- 20 RMA returns @ $50 = $1,000
- Engineering time = $3,000
- **Total example loss: $4,000**
- Chip cost "savings": $100

Additional Hidden Costs:

- Lost customer confidence
- Damage to brand reputation
- Additional QA processes needed
- Production delays

Legal & Safety
Beyond Just Technical Issues

Regulatory Compliance (Examples):

- **Aerospace: AS9100D** (8.1.4) requires counterfeit prevention processes; **SAE AS5553** specifies measures
- **Automotive: ISO 26262** requires comprehensive *traceability* and controlled processes ($\Rightarrow$ non-traceable parts endanger the safety case)
- **Medical:** EU-**MDR**/IVDR with UDI traceability requirements
- **Industrial:** CE/UL require documented supply chain

Safety-Critical Applications: Medical devices, automotive, industrial safety, aviation

Legal Exposure:

- Product liability for damages caused by non-compliant parts
- Contract/compliance violations (traceability requirements)

Our Approach
Multi-Layer Verification Strategy

Why Multiple Tests? Single tests can be misleading; layering increases security

1. **DBGMCU Device ID (0xE0042000):**
   - DEV_ID **0x410** (Medium), **0x412** (Low) - quick plausibility check
   - Clones may provide same DEV_ID; **REV_ID** often differs (e.g., GD32)
2. **Flash Size Register (0x1FFFF7E0):**
   - Authoritative size in kB, more forgery-resistant than tool heuristics
3. **Unique ID (0x1FFFF7E8):**
   - 96-bit UID for serial number/batch analysis
4. **Peripheral Testing:**
   - Verify specific peripheral behavior (USB, ADC timing, etc.)
   - Compare against known genuine chip characteristics

Functional Verification
Testing Actual Hardware

4. DMA Functional Test:

- Verifies actual peripheral functionality (Mem $\leftrightarrow$ Periph)
- Catches missing/deviating DMA implementations

5. Additional Quick Tests:

- ADC channels and accuracy
- Timer/interrupt timing
- USB enumeration (if used)
- Clock accuracy (HSI/HSE drift)

Best Practice:

- Combine ID checks *plus* functional tests
- Document/archive results (traceability)
- Use conservative timings (derivatives may have different default clocks)

Today's Workshop
Practical Chip Verification

Hands-On Exercises:

1. Read `DBGMCU->IDCODE` (**0xE0042000**) DEV_ID/REV_ID
2. Verify flash size via `0x1FFFF7E0`
3. Read UID from `0x1FFFF7E8`
4. DMA functionality test

Deliverable: Test firmware, result documentation, decision tree (OK/Reject)

Prevention Strategy
How to Avoid Counterfeits

Procurement (Best Practice):

- Prefer **authorized** distributors: Digi-Key, Mouser, Farnell/element14, ST eStore
- Verify authorization on ST website; request CoC/traceability documentation
- Be cautious with prices significantly below market rate

Red Flags:

- No manufacturer packaging/trays
- Lack of traceability documentation
- Unusual fonts/markings; "pulls/refurbs" for new designs

Nevertheless: Establish QA verification before production & as part of bring-up process

Configuration
Goal & Setup

Preparation Goal:

- ST-LINK + STM32CubeProgrammer: Check *Connect* and **Device ID**
- UART output of test firmware (DEV_ID/FlashSize/UID)
- Small **DMA** test (USART1 $\rightarrow$ DMA1)

Required:

- Blue Pill (STM32F103**C6**T6A), ST-LINK/V2(/V3), 3.3V UART-TTL adapter
- USB cable, 4 Dupont wires for SWD, 3 wires for UART

Premise: *Configuration* now; *implementation/code* in next section.

Configuration
SWD ST-LINK Wiring

Pins (Blue Pill / STM32F103):

- **SWDIO** $\rightarrow$ **PA13**, **SWCLK** $\rightarrow$ **PA14**
- **GND** $\rightarrow$ GND, **3V3** $\rightarrow$ 3V3 (reference)

Notes:

- Set interface to **SWD** in tool (not JTAG)
- Common ground
- Power board with 3.3V (from ST-LINK or external)

Configuration
UART 3.3V TTL (USART1)

Pins:

- **PA9 = USART1_TX** $\rightarrow$ to **RX** of TTL adapter
- **PA10 = USART1_RX** $\rightarrow$ to **TX** of TTL adapter
- **GND $\rightarrow$ GND**; do not use 5V TTL

Terminal recommendation: 115200 8N1, enable CR/LF

Configuration
STM32CubeProgrammer Connect

Procedure:

1. *Interface* = **ST-LINK**, *Frequency* moderate (e.g., 400-1800 kHz)
2. Click **Connect** → Device info: *Device ID*, *Revision*, *Flash size*
3. Optional: View *OB* (Option Bytes) only, do *not* modify

Goal of this phase: Visual verification that ST-LINK and chip communicate correctly (DEV_ID plausible).

Configuration
Electronic Signature Addresses

In Firmware (for next section):

- **DBGMCU_IDCODE** @ `0xE0042000` → *DEV_ID*/*REV_ID*
- **Flash Size** @ `0x1FFFF7E0` → Size in KByte (e.g., 32/64)
- **UID (96-bit)** starting @ `0x1FFFF7E8` → Serial/Batch analysis

Practice: DEV_ID is a quick plausibility check; **Flash Size** is authoritative.

Configuration
Clocks & Reset Assumptions

Sufficient for configuration:

- **HSI 8MHz** (reset default), no PLL
- **RCC**: activate only required peripheral clocks:
  - **APB2**: GPIOA, AFIO, USART1
  - **AHB**: DMA1 (for DMA test)
- **Boot0**=0, **Boot1**=0 (Flash boot)

Configuration
USART1 Configuration Prerequisites

RCC: Enable **GPIOA**, **AFIO**, **USART1**

Pins:

- **PA9** as **AF Push-Pull** (TX), **PA10** as **Input Floating** (RX)
- Optional Remap: **USART1_REMAP**=0 (Default: TX/PA9, RX/PA10)

USART1 Basic Parameters:

- 115200 baud @ 8MHz HSI (baud rate calculated later), **8N1**
- Set **UE/TE/RE** (*USART_CR1*)

Configuration
DMA Goals & Mapping (USART1)

Goal: CPU-offloaded transfer, e.g., send string via DMA to USART1

RCC: Enable **DMA1**

Channel Assignment (F1):

- **USART1_TX** $\rightarrow$ **DMA1 Channel 4**
- **USART1_RX** $\rightarrow$ **DMA1 Channel 5**

USART1: Set **CR3.DMAT/DMAR** (TX/RX via DMA)

Configuration
DMA Configuration Checklist

TX (DMA1 Ch4):

- **DIR** = Memory→Peripheral
- **PADDR** = &USART1→DR, **MADDR** = data buffer
- **MINC**=1, **PSIZE/MSIZE**=8 bit, **TCIE** optional

RX (DMA1 Ch5):

- **DIR** = Peripheral→Memory
- **CIRC** optional for continuous reception

Start: Configure channel → set **EN**; monitor **USART1.TXE/TC** flags.

Configuration
Typical Pitfalls

- **Levels**: UART adapter must be **3.3V TTL** (not 5V)
- **SWD**: Wrong interface or no 3.3V reference → no connect
- **AFIO Remap**: USART1_REMAP accidentally set → pins no longer match
- **DMA**: Wrong channel (not Ch4/Ch5) or writing *value* instead of address to **PADDR**
- **Clock**: Forgot RCC enable for GPIO/USART/DMA

Ready for "Implementation": Code snippets next (DEV_ID/FlashSize/UID → UART, then DMA-TX).

A. Hardware & Tools

What you need & how to connect

Boards/Tools

- Blue Pill **STM32F103C6T6A** (Low-Density, 32 KB Flash)
- ST-LINK/V2 (or V3) for SWD (Debug/Flash)
- USB-UART (3.3V TTL) for console output

Wiring (Minimum)

- *SWD*: SWDIO → PA13, SWCLK → PA14, 3V3, GND
- *USART1*: PA9 (TX)→ RX(USB-TTL),
  PA10 (RX)← TX(USB-TTL), GND

Software

- STM32CubeIDE (includes CubeMX)
- STM32CubeProgrammer (ST-LINK drivers/flasher)
- (Optional) IDE Serial Terminal Plugin ("TM Terminal")

B. New Project in STM32CubeIDE
Project Setup (CubeMX integrated)

Setup

1. *File* → *New* → *STM32 Project* ⇒ MCU: `STM32F103C6Tx`
2. Assign project name, Toolchain = STM32CubeIDE

Why CubeIDE?

- Peripheral configuration & code generator (`.ioc`)
- Integrated debugging via ST-LINK
- Easy to add Eclipse plugins (Terminal)

C. CubeMX Configuration (.ioc)
Minimum for ID/UID/USART-DMA Demo

System

- *SYS*: **Debug = Serial Wire** (SWD enabled)
- *Clock*: HSI 8 MHz (sufficient for start), later optionally HSE→PLL (72 MHz)

USART1 (PA9/PA10)

- Mode: Asynchronous, 115200 8N1, Oversampling=16, no HW flow control
- *DMA Settings*: USART1_TX → DMA1 Ch4, USART1_RX → DMA1 Ch5

Registers we will read

- `DBGMCU_IDCODE @ 0xE0042000` (DEV_ID, REV_ID)
- `FLASH_SIZE @ 0x1FFFF7E0` (Size in kB)
- `UID[95:0] @ 0x1FFFF7E8` (3×32 Bit)

D. Flashing & Debugging with ST-LINK
From Build to Running Firmware

Connection

1. ST-LINK to SWDIO/PA13, SWCLK/PA14, 3V3, GND
2. In CubeIDE: *Debug As* $\rightarrow$ *STM32 MCU Application*

CubeProgrammer (optionally visible)

- Device Connect $\Rightarrow$ *Device Information*: DEV_ID, Flash Size
- ST-LINK drivers installed? (Windows: STSW-LINK009)

Typical Pitfalls

- BOOT0 must be 0 (Flash start), common ground (GND) not forgotten
- Reduce SWD frequency with long cables

E. Display UART Output

Console Setup

In CubeIDE (Serial Terminal)

1. (If not present) *Help* → *Install New Software* ⇒ "TM Terminal"
2. *Window* → *Show View* → *Terminal* ⇒ *Serial*
3. Correct COM port, 115200, 8N1, no flow control

Alternatives

- `screen`, PuTTY, `minicom`, etc.

Interpretation
What do DEV_ID, FlashSize, UID, BRR mean?

Example (your logs)

- `DBGMCU_IDCODE = 0x10006412` ⇒ **DEV_ID 0x412** (Low-Density F10x), **REV_ID 0x1000**
- `Flash size = 32 KB` ⇒ matches **STM32F103C6**
- `UID` (96-bit) readable ⇒ plausible serial identifier
- `PCLK2 = 8 MHz, BRR = 0x0045, Over = 16` ⇒ **115200 baud** correct

Classification

- **0x412** is the Low-Density device ID (F101/F102/F103 with 16-32 KB)
- **Flash Size Register** provides actual size in kB
- **UID** fixed at 0x1FFFF7E8 (3×32 Bit)
- **BRR** (with 16x oversampling): Baud $= \frac{f_{\text{PCLK}}}{16 \cdot (\text{Mant} + \text{Frac}/16)}$