SYNCHRO Timer

U S E R M A N U A L

Manual Reference: UM_SYNCHRO-Timer_0226_revA1

Bellamare, LLC
www.bellamare-us.com
San Diego, California - USA

info@bellamare-us.com
Tel: +1 (858) 578-8108

Table of Content

1. Introduction

1.1 Purpose of SYNCHRO Timer

The SYNCHRO Timer is a low-power autonomous scheduling module designed to eliminate wasted energy during oceanographic and subsea deployments.

Many instruments consume power continuously, even when data acquisition is not required. SYNCHRO Timer addresses this inefficiency by physically disconnecting the main power path outside of predefined operating windows. By powering instruments only when they are needed, SYNCHRO Timer significantly extends deployment duration without increasing battery size or system complexity.

Simple, reliable, and autonomous, SYNCHRO Timer enables longer missions while preserving valuable onboard energy resources.

1.2 How SYNCHRO Timer Works (Concept)

SYNCHRO Timer operates as an intelligent inline power switch placed between an external battery pack and a subsea instrument.

Internally, the system combines:

- A low-power microcontroller (Seeed Studio XIAO nRF52840)
- A precision Real-Time Clock (DS3231)
- A MOSFET-based power switching stage

The Real-Time Clock maintains accurate timekeeping. The microcontroller uses this time reference to determine when power should be enabled or disabled. At scheduled intervals, the microcontroller activates the MOSFET, closing the main power path and allowing current to flow to the connected instrument. When the programmed on-duration expires, the MOSFET opens the circuit, completely isolating the instrument from the battery supply.

This architecture ensures:

- Precise timing control
- Extremely low standby consumption
- Reliable operation over long deployments

SYNCHRO Timer's internal AAA batteries power only the control electronics and switching circuitry. The connected instrument is powered exclusively by the external battery pack.

1.3 System Overview

SYNCHRO Timer consists of a pressure-rated cylindrical housing with integrated power connectors and an internal electronics assembly mounted to the removable end-cap.

Key external features include:

- Power input connector (MCBH2M)
- Power output connector (MCBH2F)
- Acrylic sight glass for LED status indication
- Removable threaded end-cap with dual O-ring seals

Fig. 1: Overall Product View

The enclosure is available in:

- Hard-anodized 6061-T6 aluminum for deep-water applications
- Acetal for shallower deployments

The system is configured via a USB-C connection to the internal microcontroller using the Arduino IDE. Once configured and armed, SYNCHRO Timer operates fully autonomously without external communication.

1.4 Intended Use

SYNCHRO Timer is designed for:

- Oceanographic instrument scheduling
- Intermittent sensor activation
- Battery-powered subsea systems
- Long-duration autonomous deployments

The device is intended to be installed inline between a battery pack and a compatible DC-powered instrument within its specified voltage and depth ratings.

2. Technical Specifications

2.1 Mechanical

Housing Options:

Deep-Water Version
- Material: Hard-anodized 6061-T6 aluminum
- Maximum Depth Rating: 4,500 meters
- Weight: 2.5 lb (in air), 1.25 lb (in water)

Shallow-Water Version
- Material: Acetal
- Maximum Depth Rating: 500 meters
- Weight: 1.5 lb (in air), 0.25 lb (in water)

Dimensions
- Length: 7.5 in
- Outer Diameter: 2.5 in

Sealing
- Dual O-ring design
- O-rings: 2 × #134 Buna-N (end-cap bore)

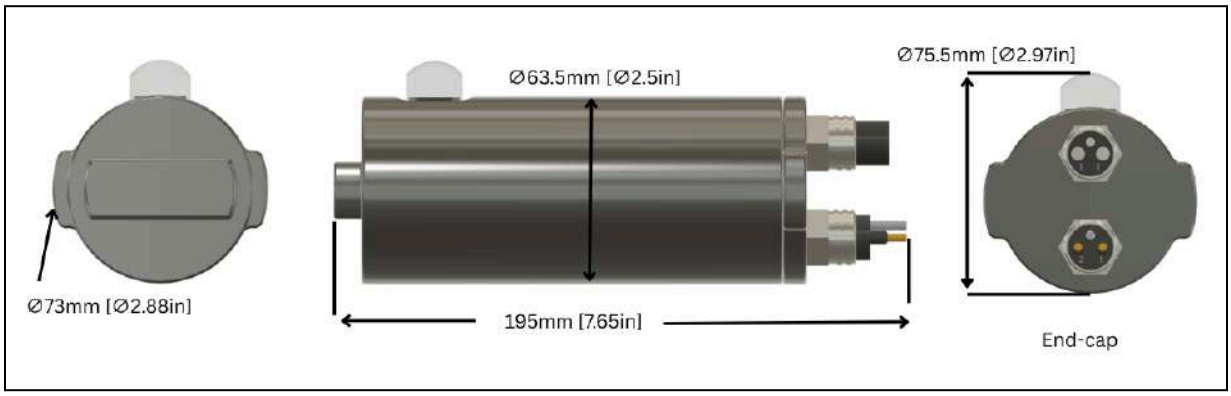


Fig. 2: SYNCHRO Timer Mechanical Dimensions

2.2 Electrical

Input Voltage Range
- 6-28 VDC
- Maximum current: 10 A

Control Power Source
- 3 × AAA batteries (internal)

RTC Backup
- 1 × CR1220 coin-cell battery

Power Switching
- MOSFET-based solid-state switching
- Full disconnection of output power when OFF

Control Electronics
- Microcontroller: Seeed Studio XIAO nRF52840
- Real-Time Clock: DS3231

2.3 Power Autonomy

Autonomy values below reflect conservative estimates for cold-water deployment conditions (0–5 °C).

Duty Cycle	Cold Alkaline AAA	Cold Lithium AAA
2 hours per day	~3–4 weeks	~6–7 weeks
1 hour per day	~5–6 weeks	~10 weeks
10 min per hour	~2 weeks	~3–4 weeks

Recommended battery types:
- Alkaline: Duracell Coppertop or Energizer Max
- Lithium: Energizer Ultimate Lithium AAA

2.4 Interfaces

Power Input
- MCBH2M (2-pin male)

Power Output
- MCBH2F (2-pin female)

Configuration Interface
- Internal USB-C serial port

Status Indication
- Green and Red LEDs (visible through acrylic sight glass)

2.5 Software Interface

- Configuration via Arduino IDE
- Serial-based interactive mission setup
- Internal flash storage for mission parameters

3. Hardware Description

3.1 Mechanical Construction

SYNCHRO Timer consists of a cylindrical pressure-rated housing with a removable threaded end-cap. The internal electronics assembly is mounted directly to the dry side of the end-cap.

The enclosure is available in two configurations:

- Hard-anodized 6061-T6 aluminum for deep-water applications
- Acetal for shallow-water deployments

The sealing system uses dual #134 Buna-N O-rings installed in the end-cap bores. Proper cleaning and lubrication of these O-rings are essential to maintain pressure integrity.

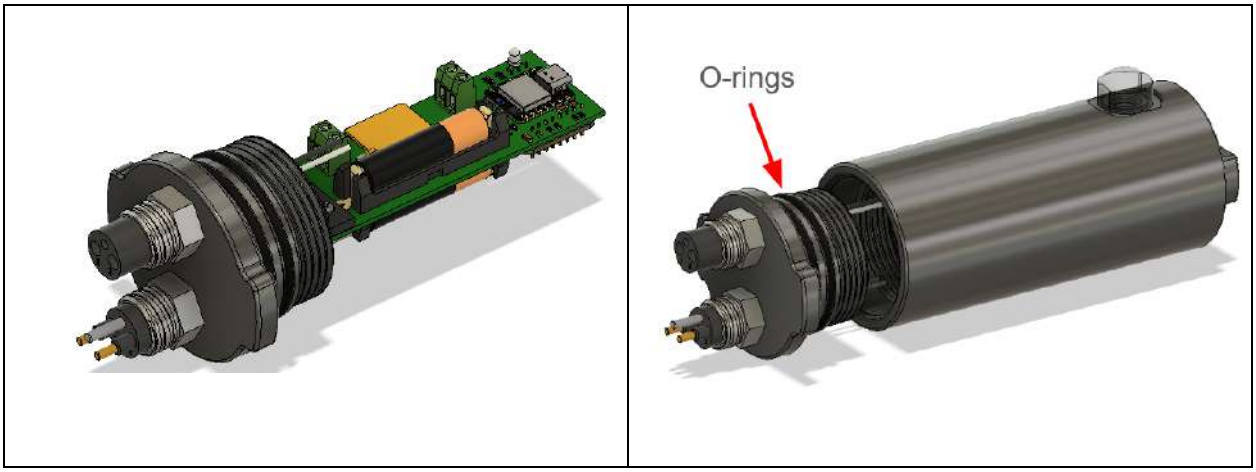


Fig. 3: End-Cap Assembly with Electronics and O-rings

3.1.1 End-Cap & Access

The end-cap integrates:

- Power input connector (MCBH2M)
- Power output connector (MCBH2F)
- Internal electronics board

To access the electronics, unthread the end-cap by turning it counter-clockwise. Grip tabs are integrated into the cap to assist removal. A wrench flat at the base of the cylinder accommodates a ¾” wrench if a strong grip of the housing is required during opening.

For deployment, the end-cap must be tightened by hand only. Ensure the cap is fully seated and flush with the housing before use.

Fig. 4: End-Cap Grip Tabs and Wrench Interface

3.2 Electrical Architecture

SYNCHRO Timer is built around a low-power control architecture designed to minimize standby consumption while providing precise timing control.

The main components include:

- Seeed Studio XIAO nRF52840 microcontroller
- DS3231 Real-Time Clock (RTC)
- MOSFET-based power switching stage
- Dual status LEDs

3.2.1 Power Switching

The MOSFET functions as a solid-state switch placed inline between the external battery pack and the connected instrument.

- When activated, the MOSFET closes the main power path, allowing current to flow to the instrument.
- When deactivated, it opens the circuit, fully isolating the instrument from the battery supply. This physical disconnection prevents idle drain and extends mission duration.

3.2.2 Timing and Control

The DS3231 Real-Time Clock maintains continuous timekeeping. The microcontroller uses this time reference to:

- Determine scheduled activation times
- Enable the MOSFET for the programmed duration
- Disable power at the end of each cycle

The RTC can generate hardware alarms that wake the microcontroller from low-power sleep, ensuring precise scheduling with minimal standby current.

3.2.3 Status Indication

Two internal LEDs provide system feedback:

- Green LED: Power-on and active states
- Red LED: Power-off and configuration events

The LEDs are visible through an acrylic sight glass. During deployment, LED activity is brief and minimized to conserve power.

Fig. 5: LED Sight Glass

3.3 Battery System

SYNCHRO Timer uses two independent battery systems, each serving a different function.

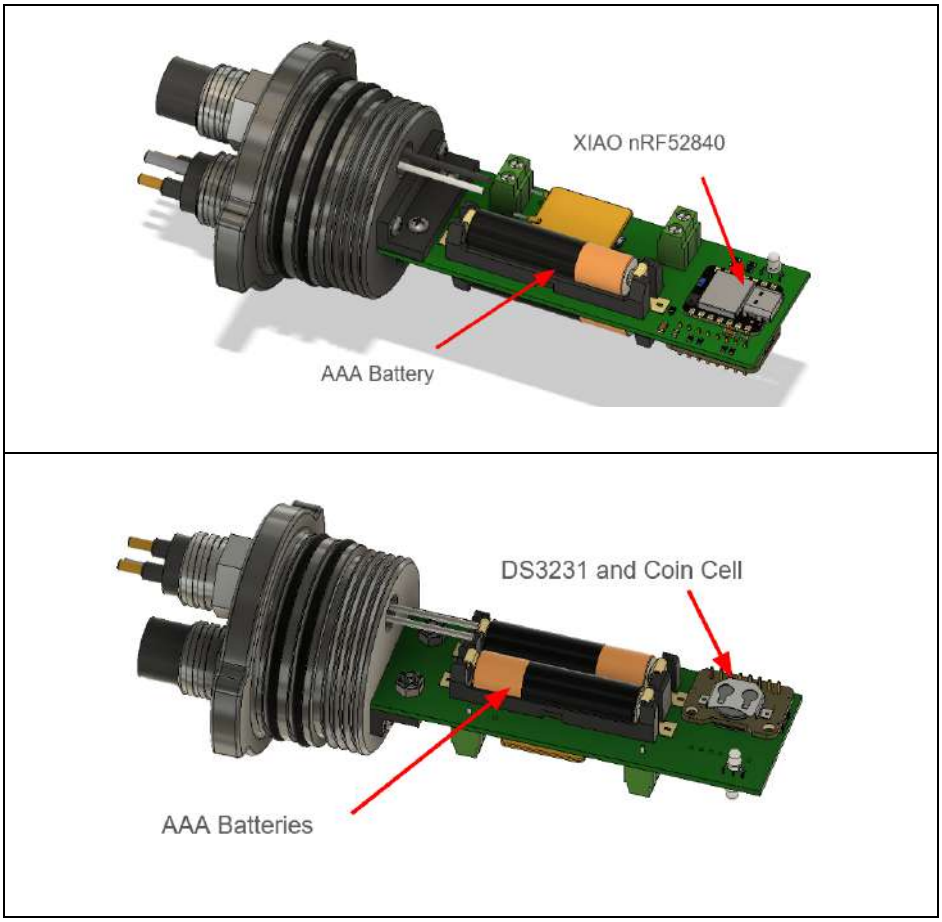


Fig. 6: Internal Battery Configuration

3.3.1 AAA Battery Pack (Control Power)

Three AAA batteries provide primary operating power for:

- Microcontroller
- RTC (when main power is present)
- MOSFET gate drive circuitry
- LED indicators

These batteries do not power the connected instrument. The instrument is powered exclusively by the external battery pack connected to the input/output connectors.

To fully stop SYNCHRO Timer's operation, all three AAA batteries must be removed.

3.3.2 Coin-Cell Battery (RTC Backup)

A CR1220 coin-cell battery powers the RTC backup supply only.

During normal operation, the RTC is powered from the main AAA supply.
If the AAA batteries are removed or depleted, the RTC automatically switches to the coin-cell backup source to preserve date and time.

The coin-cell does not power the microcontroller or any other subsystem.

Although capable of multi-year operation, the coin-cell should be replaced annually or prior to any critical deployment to ensure reliable time retention.

4. Installation & Configuration

4.1 Opening the Unit

- Ensure the system is powered OFF.
- Unscrew the end-cap by turning it counter-clockwise.
- Carefully remove the end-cap to expose the internal electronics assembly.

Grip tabs are integrated into the end-cap to assist removal. A ¾” wrench may be used to stabilize the housing if required.

4.2 Connecting to the Microcontroller

- Connect a USB-C cable to the Seeed Studio XIAO nRF52840 inside the enclosure.
- Connect the other end to your computer.
- The microcontroller will power from the USB connection.

Note (Setup Only):
During configuration, the microcontroller may be powered directly from USB. The internal AAA batteries are not required for setup and may be removed to simplify restart and reprogramming.

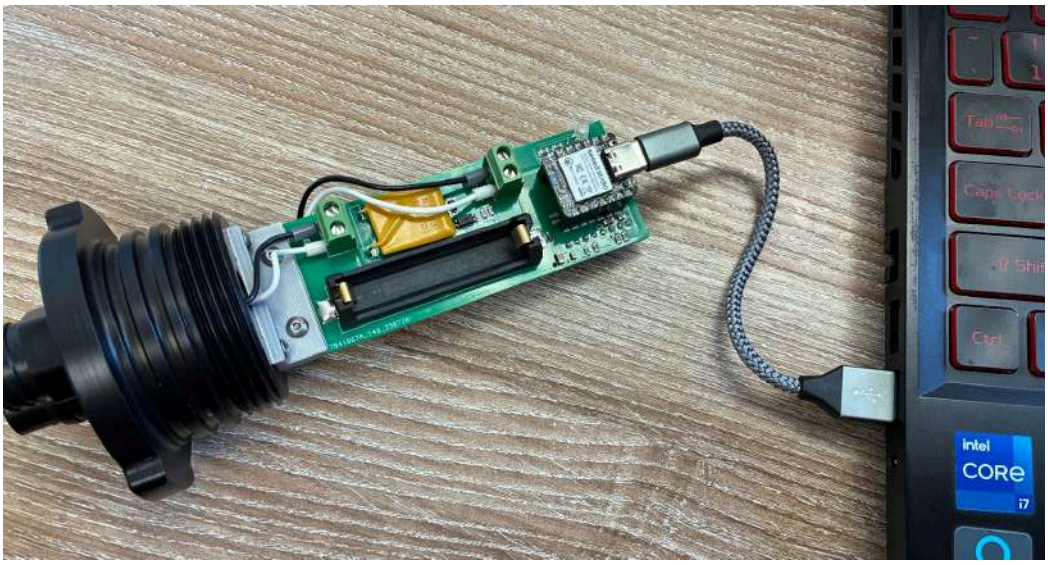


Fig. 7: USB-C Connection to XIAO nRF52840

4.3 Arduino IDE Setup (Optional – Firmware Modification Only)

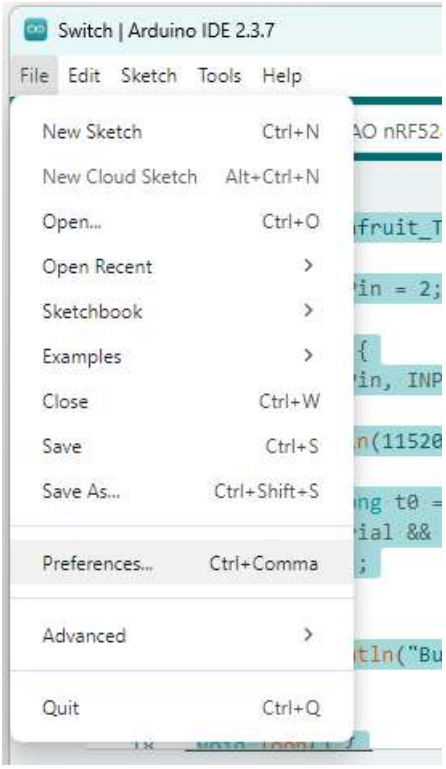
The SYNCHRO Timer is delivered with preloaded firmware and does not require the Arduino IDE for normal operation.
The following steps are required only if you intend to modify, recompile, or upload firmware.

4.3.1 Install Arduino IDE

- Download and install the Arduino IDE from:
  https://www.arduino.cc/en/software/

4.3.2 Install Seeed nRF52840 Board Package

- Open Arduino IDE.
- Go to File → Preferences
- In “Additional Boards Manager URLs” paste:
  https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json
- Click OK.
- Go to Tools → Board → Boards Manager
- Search for: Seeed nRF52
- Install: Seeed XIAO nRF52840



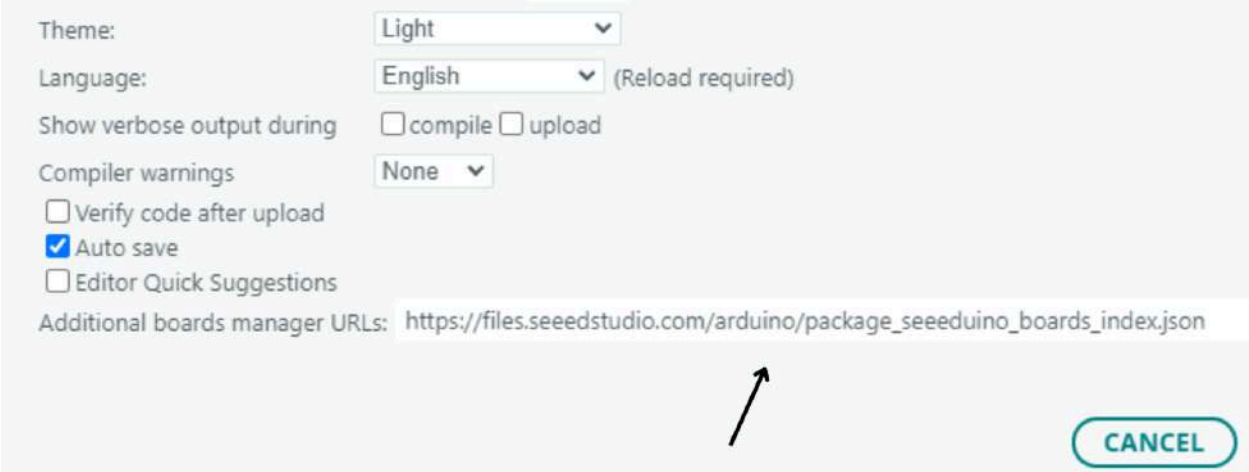


Fig. 8: Installing Board Package

4.3.3 Required Libraries

If compiling firmware, install:

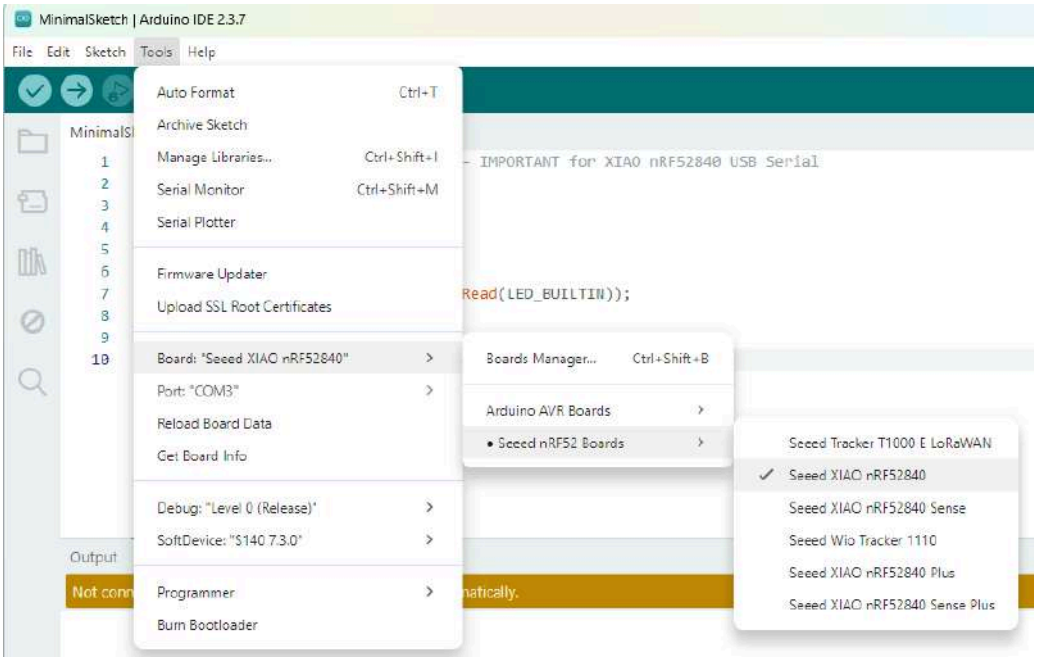
- Seeed Studio nRF52 board package
- Adafruit RTClib
- Adafruit LittleFS / InternalFileSystem

Failure to install these dependencies will result in compilation errors.

4.4 Selecting Board and Port

- Go to Tools → Board
  Select: Seeed XIAO nRF52840

- Go to Tools → Port
  Select the connected COM port (Windows) or USB device (Mac/Linux).



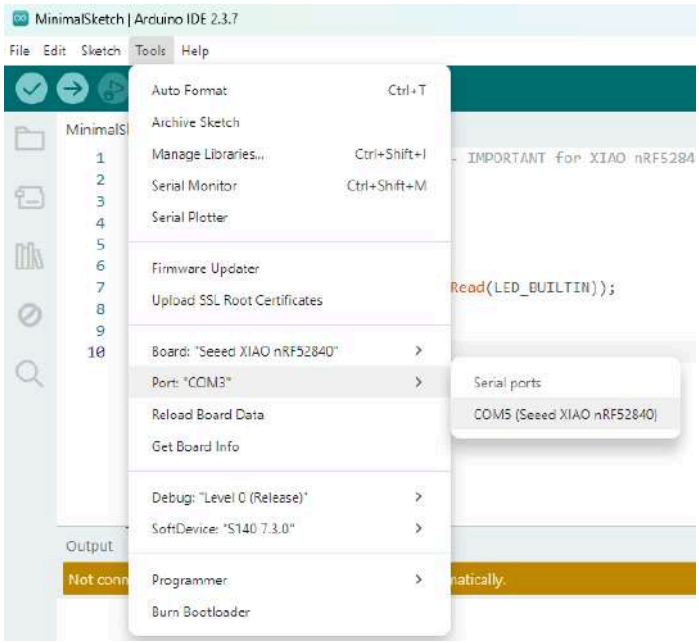


Fig. 9: Selecting Board and Port

4.5 Serial Configuration Procedure

- Open Tools → Serial Monitor
- Set baud rate to 115200
- If no message appears, press the reset button on the microcontroller.

Pressing reset is safe and restarts the firmware.

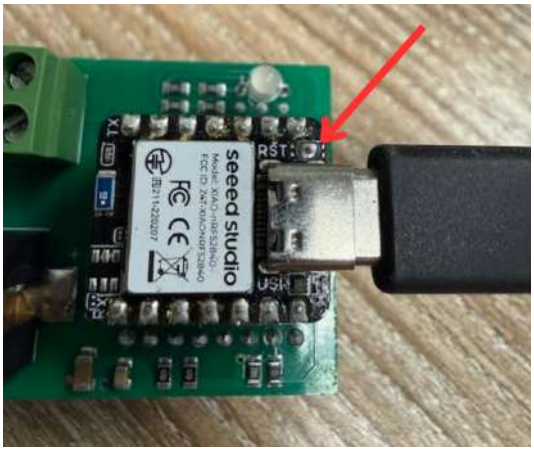


Fig. 10: Serial Monitor and Reset Button

4.6 Setting the RTC (Optional)

At startup, the system prompts:

```
Power OFF (pin driven LOW)

Would you like to change RTC? (yes/no)
```

- Enter no to keep the current RTC time.
- Enter yes to manually set date and time.

If “yes” is selected:
Enter time in the format: YYYY-MM-DD HH:MM:SS

4.7 Configuring Mission Timing

After RTC confirmation, the system displays current settings:

- Start Time
- On Duration
- Interval
- Time until first cycle

You will be prompted: `Use these settings? (yes/no)`

```
Loaded settings from flash.

=== SYNCRO SYSTEM CONFIGURATION ===

RTC Time: 2026-02-11T13:07:13

Start Time: 13:10:0

On Duration: 1h 0m 0s

Interval: 5h 0m 0s

First cycle starts in: 0h 2m 47s

-------------------------------------

Use these settings? (yes/no)
```

If you enter no, you will enter configuration mode.
You will be prompted to enter:

- `Start Time [HH:MM:SS]`
- `On Duration [HH:MM:SS]`
- `Interval [HH:MM:SS]`

After entry, the system summarizes the new configuration and again prompts:
`Use these settings? (yes/no)`

4.8 Saving to Flash Memory

After accepting settings, you will be asked:

```
Save these settings to flash for next boot? (yes/no)
```

- Selecting yes stores parameters in non-volatile memory.
- Selecting no applies settings for the current session only.

Once confirmed, the system arms the mission and displays:

```
Saved to flash.

Next RTC alarm: 2026-02-11T13:25:00

Mission armed.
```

4.9 LED Behavior During Operation

SYNCHRO Timer uses two internal LEDs for feedback:
During configuration:

- Green LED blinks to confirm accepted inputs.
- Red LED blinks during configuration changes.

When armed and awaiting start:

- Green LED blinks once every 5 seconds.

During operation:

- Green LED blinks briefly when power turns ON.
- Red LED blinks briefly when power turns OFF.

LED activity is intentionally brief to minimize power consumption during deployment.

5. Deployment Procedure

5.1 Settings Verification

Before closing SYNCHRO Timer, verify the following:

- Mission timing parameters have been correctly configured.
- The RTC clock displays the correct date and time.
- AAA batteries are installed and fresh.
- The CR1220 coin-cell battery is installed.

5.2 O-Ring Inspection and Preparation

Proper O-ring preparation is critical to maintaining pressure integrity.

- Remove the O-rings from the end-cap grooves.
- Inspect for cuts, deformation, or contamination.
- Clean the O-rings.
- Apply a light, even coat of appropriate O-ring grease.
- Ensure the O-ring grooves are clean and free of debris.
- Inspect the interior bore of the cylinder for dirt, hair, or foreign particles.
- Apply a very thin film of O-ring grease to the cylinder sealing surface.

  Do not over-grease. Excess lubricant may attract debris.

5.3 Closing the Enclosure

- Carefully insert the end-cap into the cylinder.
- Thread the end-cap clockwise.
- Tighten by hand only.
- Ensure the end-cap is fully seated and flush against the cylinder.
- Confirm there is no visible gap between cap and housing.

  A wrench is not required for final tightening.

5.4 Connecting External Power and Instrument

- Mount SYNCHRO Timer securely to the instrumented payload.
- Connect the external battery pack to the Power Input (MCBH2M) connector.
- Connect the instrument to the Power Output (MCBH2F) connector.
- Ensure cables are routed to avoid mechanical stress.

5.5 Functional Test Prior to Deployment

Before field deployment:

- Connect the external battery pack.
- Observe LED behavior.
- Confirm scheduled activation and deactivation occur at the programmed times.
- Verify the instrument powers ON and OFF as expected.

A full functional test under realistic conditions is strongly recommended before subsea deployment.

6. Maintenance & Corrosion Prevention

6.1 Routine Surface Protection

For long-term or repeated deployments:

- Apply a light protective coating of **LPS-3** to all exposed metallic surfaces.
- Wipe off excess product after application.
- Avoid applying lubricant to O-rings or sealing faces unless specifically intended for that purpose.

  Protective coating reduces oxidation and galvanic corrosion in saltwater environments.

6.2 Post-Recovery Procedure

After each recovery from seawater:

- Rinse the entire enclosure thoroughly with fresh water to remove salt deposits.
- Clean with fresh water and mild dish soap if necessary.
- Allow the unit to dry completely.
- Reapply a light protective coating of LPS-3 before storage.

  Salt deposits left on aluminum surfaces accelerate corrosion and pitting.

6.3 Inspection and Corrosion Treatment

If corrosion or pitting is observed:

- Remove surface oxidation using a soft wire brush.
- Clean the affected area.
- Temporarily seal the area with heavy grease if immediate repair is not possible.
- For long-term protection, apply epoxy coating.

  J-B Weld is recommended for ease of use and adhesion.

6.4 O-Ring Replacement

Replace O-rings if any of the following are observed:

- Cuts, cracking or abrasions
- Permanent deformation
- Flattening

  Only use specified replacement O-rings (#134 Buna-N).

7. Included Items & Accessories

7.1 Included Items

Each SYNCHRO Timer is delivered in a protective case and includes the following components:

Hardware:	• 3 × AAA batteries (control power) • 1 × CR1220 coin-cell battery (RTC backup) • 1 × USB-C cable (3 ft), • 2 x #134 Buna-N replacement O-rings
Dummy Plugs*	• 1 x Input POWER connector plug • 1 x Output POWER connector plug

***** Dummy plugs must be installed whenever connectors are not in use to prevent corrosion

7.2 Optional Accessories

Optional accessories may be available depending on deployment requirements, including:

- Replacement microcontroller and/or RTC
- Custom cable assemblies

7.3 Support Services

Bellamare provides technical support and service options, including:

- Firmware updates
- Field Support
- Replacement parts
- Deployment consultation

For assistance, contact:

Bellamare, LLC
San Diego, California – USA
www.bellamare-us.com
info@bellamare-us.com
Tel: +1 (858) 578-8108

8. Legal Notices and Warranty

8.1 General Disclaimer

This manual is provided “as is” without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability or fitness for a particular purpose.

Bellamare reserves the right to modify the product, firmware, specifications, or documentation without prior notice.

8.2 Safety Warnings

Failure to follow proper operating procedures may result in equipment damage, data loss, or personal injury.

- Do not operate SYNCHRO Timer outside of its specified depth rating.
- Always remove power before opening the enclosure.
- Use only specified connectors, cables, and voltage ranges.
- Ensure O-rings are clean and properly lubricated prior to deployment.
- Only trained personnel should perform internal modifications or repairs.

8.3 Electrical Responsibility

Improper handling of power connections may result in damage to the device or connected instruments.

Bellamare assumes no responsibility for damage resulting from:

- Incorrect wiring
- Use outside specified voltage limits
- Reverse polarity
- Unauthorized modification

Users are responsible for verifying compatibility with external battery systems and instruments.

8.4 Warranty

SYNCHRO Timer is covered by a limited one-year warranty against defects in materials and workmanship under normal use.

This warranty does not cover:

- Damage caused by misuse or improper installation
- Unauthorized modifications
- Corrosion resulting from inadequate maintenance
- Normal wear associated with marine environments

For warranty service or technical support, contact Bellamare.

8.5 Intellectual Property

The SYNCHRO Timer hardware and firmware are proprietary systems developed by Bellamare, LLC.

Unauthorized duplication, reverse engineering, or distribution of firmware or hardware designs is prohibited.

8.6 Data Responsibility

Bellamare is not liable for loss of data, missed acquisition events, or mission failures resulting from the use or misuse of this product.

Users are solely responsible for verifying correct configuration and performing functional tests prior to deployment.

Appendix 1: Arduino IDE SYNCHRO Timer Sketch

```
// === SYNCHRO Timer (XIAO nRF52840 / Seeed nRF52) ===
// Reliable polarity-safe power switching + interactive configuration + optional
flash save

#include <Wire.h>
#include "RTClib.h"
#include <Adafruit_LittleFS.h>
#include <InternalFileSystem.h>
using namespace Adafruit_LittleFS_Namespace;

// ----------------- PINS -----------------
const int MOSFET_PIN  = 0;
const int ledGreen    = 9;
const int ledRed      = 10;
const int rtcAlarmPin = 3;

// ----------------- FLAGS -----------------
const bool ACTIVE_LOW      = false;
const bool DEPLOYMENT_MODE = true;

// ----------------- RTC -----------------
RTC_DS3231 rtc;

// ----------------- SETTINGS -----------------
struct Settings {
  int startH = 12, startM = 45, startS = 0;
  int onH    = 0,  onM    = 6,  onS    = 0;
  int intH   = 0,  intM   = 2,  intS   = 0;
};
Settings cfg;

const char* CFG_FILE = "/syncro_cfg.txt";
bool fsOK = false;

// ----------------- STATE -----------------
volatile bool rtcAlarmTriggered = false;
bool initialized = false;
bool waitingForFirstCycle = true;
unsigned long lastPreStartBlink = 0;
bool isFirstCycle = true;

// ==================== Power control ====================
void setPower(bool on) {
  int level = on ? HIGH : LOW;
  if (ACTIVE_LOW) level = (level == HIGH) ? LOW : HIGH;
  digitalWrite(MOSFET_PIN, level);
```

```
  Serial.print("Power "); Serial.print(on ? "ON" : "OFF");
  Serial.print(" (pin driven "); Serial.print(level == HIGH ? "HIGH" : "LOW");
Serial.println(")");
}

// ===================== LEDs =====================
void blinkColor(int pin, int duration_ms) {
  digitalWrite(ledGreen, LOW);
  digitalWrite(ledRed, LOW);
  digitalWrite(pin, HIGH);
  delay(duration_ms);
  digitalWrite(pin, LOW);
}

// ===================== RTC interrupt =====================
void onRTCAlarm() {
  rtcAlarmTriggered = true;
}

// ===================== Serial helpers =====================
String readLineBlocking() {
  String s = "";
  while (s.length() == 0) {
    while (!Serial.available()) delay(10);
    s = Serial.readStringUntil('\n');
    s.trim();
  }
  while (Serial.available()) Serial.read();
  return s;
}

// ---------------- Reliable yes/no prompt ----------------
String getYesNo(const String &prompt) {
  String response = "";
  while (true) {
    while (Serial.available()) Serial.read();
    Serial.println(prompt);
    while (!Serial.available()) delay(10);
    response = Serial.readStringUntil('\n');
    response.trim();
    if (response.equalsIgnoreCase("yes") || response.equalsIgnoreCase("no"))
      return response;
    Serial.println("Please type 'yes' or 'no'.");
  }
}

// ===================== Parsing =====================
bool parseHMS(const String& in, int &h, int &m, int &s, bool isTimeOfDay) {
  int c1 = in.indexOf(':'); int c2 = in.indexOf(':', c1+1);
  if (c1 < 0 || c2 < 0) return false;
  h = in.substring(0, c1).toInt();
  m = in.substring(c1+1, c2).toInt();
  s = in.substring(c2+1).toInt();
```

```
  if (isTimeOfDay && (h < 0 || h > 23)) return false;
  if (!isTimeOfDay && h < 0) return false;
  if (m < 0 || m > 59 || s < 0 || s > 59) return false;
  return true;
}

bool parseDateTime(const String& in, DateTime& dt) {
  if (in.length() < 19) return false;
  int year  = in.substring(0,4).toInt();
  int month = in.substring(5,7).toInt();
  int day   = in.substring(8,10).toInt();
  int hour  = in.substring(11,13).toInt();
  int min   = in.substring(14,16).toInt();
  int sec   = in.substring(17,19).toInt();
  if (year < 2020 || month < 1 || month > 12 || day < 1 || day > 31) return false;
  if (hour < 0 || hour > 23 || min < 0 || min > 59 || sec < 0 || sec > 59) return
false;
  dt = DateTime(year, month, day, hour, min, sec);
  return true;
}

String two(int v){ return (v < 10) ? (String("0") + v) : String(v); }

bool validateSettings(const Settings& x) {
  if (x.startH < 0 || x.startH > 23 || x.startM < 0 || x.startM > 59 || x.startS < 0
|| x.startS > 59) return false;
  if (x.onH < 0 || x.onM < 0 || x.onS < 0 || x.onM > 59 || x.onS > 59) return false;
  if (x.intH < 0 || x.intM < 0 || x.intS < 0 || x.intM > 59 || x.intS > 59) return
false;

  unsigned long onSec  = (unsigned long)x.onH  * 3600UL + (unsigned long)x.onM  *
60UL + (unsigned long)x.onS;
  unsigned long intSec = (unsigned long)x.intH * 3600UL + (unsigned long)x.intM *
60UL + (unsigned long)x.intS;
  if (onSec == 0 || intSec == 0) return false;

  return true;
}

// ==================== Flash ====================
bool fsBeginOnce() {
  if (fsOK) return true;
  fsOK = InternalFS.begin();
  if (!fsOK) Serial.println("WARNING: InternalFS.begin() failed.");
  return fsOK;
}

bool loadSettings() {
  if (!fsBeginOnce()) return false;
  File f = InternalFS.open(CFG_FILE, FILE_O_READ);
  if (!f) return false;

  String s = "";
```

```
  while (f.available()) s += (char)f.read();
  f.close();
  s.trim();
  if (s.length() == 0) return false;

  int p1 = s.indexOf('\n');
  int p2 = s.indexOf('\n', p1+1);
  if (p1 < 0 || p2 < 0) return false;

  String l1 = s.substring(0, p1);
  String l2 = s.substring(p1+1, p2);
  String l3 = s.substring(p2+1);

  auto parseLine = [&](const String& line, const char* key, int &h, int &m, int
&sec, bool isTimeOfDay) -> bool {
    String k = String(key) + "=";
    if (!line.startsWith(k)) return false;
    String t = line.substring(k.length());
    return parseHMS(t, h, m, sec, isTimeOfDay);
  };

  Settings tmp = cfg;
  if (!parseLine(l1, "start",    tmp.startH, tmp.startM, tmp.startS, true))  return
false;
  if (!parseLine(l2, "on",       tmp.onH,    tmp.onM,    tmp.onS,    false)) return
false;
  if (!parseLine(l3, "interval", tmp.intH,   tmp.intM,   tmp.intS,   false)) return
false;
  if (!validateSettings(tmp)) return false;

  cfg = tmp;
  return true;
}

bool saveSettings() {
  if (!fsBeginOnce()) return false;
  if (!validateSettings(cfg)) return false;

  InternalFS.remove(CFG_FILE);
  File f = InternalFS.open(CFG_FILE, FILE_O_WRITE);
  if (!f) return false;

  String out = "start=" + two(cfg.startH) + ":" + two(cfg.startM) + ":" +
two(cfg.startS) + "\n";
  out +=       "on=" + two(cfg.onH) + ":" + two(cfg.onM) + ":" + two(cfg.onS) +
"\n";
  out +=       "interval=" + two(cfg.intH) + ":" + two(cfg.intM) + ":" +
two(cfg.intS) + "\n";
  f.write(out.c_str(), out.length());
  f.close();
  return true;
}
```

```
// ==================== RTC SETUP ====================
void maybeSetRTC() {
  String ans = getYesNo("Would you like to change RTC? (yes/no)");
  if (ans.equalsIgnoreCase("no")) return;

  Serial.println("Enter current date and time [YYYY-MM-DD HH:MM:SS]:");
  while (true) {
    String s = readLineBlocking();
    DateTime dt;
    if (parseDateTime(s, dt)) {
      rtc.adjust(dt);
      Serial.print("RTC set to: ");
      Serial.println(dt.timestamp());
      return;
    }
    Serial.println("Invalid format. Try again [YYYY-MM-DD HH:MM:SS]:");
  }
}

// ==================== CONFIGURATION ====================
void printConfiguration() {
  DateTime now = rtc.now();
  DateTime firstCycle(now.year(), now.month(), now.day(), cfg.startH, cfg.startM,
cfg.startS);
  if (firstCycle <= now) firstCycle = firstCycle + TimeSpan(1,0,0,0);
  TimeSpan delta = firstCycle - now;

  Serial.println("=== SYNCRO SYSTEM CONFIGURATION ===");
  Serial.print("RTC Time: "); Serial.println(now.timestamp());
  Serial.print("Start Time: "); Serial.print(cfg.startH); Serial.print(":");
Serial.print(cfg.startM); Serial.print(":"); Serial.println(cfg.startS);
  Serial.print("On Duration: "); Serial.print(cfg.onH); Serial.print("h ");
Serial.print(cfg.onM); Serial.print("m "); Serial.print(cfg.onS);
Serial.println("s");
  Serial.print("Interval: "); Serial.print(cfg.intH); Serial.print("h ");
Serial.print(cfg.intM); Serial.print("m "); Serial.print(cfg.intS);
Serial.println("s");
  Serial.print("First cycle starts in: ");
  Serial.print(delta.days()*24 + delta.hours()); Serial.print("h ");
  Serial.print(delta.minutes()%60); Serial.print("m ");
  Serial.print(delta.seconds()%60); Serial.println("s");
  Serial.println("-------------------------------------");
}

// ==================== SETTINGS PROMPTS ====================
// NOTE: Prompts are printed on their own lines (Serial.println),
// so they do not appear "next to each other" in Serial Monitor.
void promptForSettings() {
  Settings tmp = cfg;
  Serial.println("\nEnter new settings (format HH:MM:SS).");

  while (true) {
    Serial.println("Start Time [HH:MM:SS]:");
```

```
    String st = readLineBlocking();
    int h, m, s;
    if (parseHMS(st, h, m, s, true)) { tmp.startH = h; tmp.startM = m; tmp.startS =
s; break; }
    Serial.println("Invalid format. Try again.");
  }

  while (true) {
    Serial.println("On Duration [HH:MM:SS]:");
    String on = readLineBlocking();
    int h, m, s;
    if (parseHMS(on, h, m, s, false)) { tmp.onH = h; tmp.onM = m; tmp.onS = s;
break; }
    Serial.println("Invalid format. Try again.");
  }

  while (true) {
    Serial.println("Interval [HH:MM:SS]:");
    String iv = readLineBlocking();
    int h, m, s;
    if (parseHMS(iv, h, m, s, false)) { tmp.intH = h; tmp.intM = m; tmp.intS = s;
break; }
    Serial.println("Invalid format. Try again.");
  }

  if (!validateSettings(tmp)) { Serial.println("ERROR: Invalid settings."); return;
}
  cfg = tmp;
  Serial.println("\nNew settings captured.");
}

// ==================== CONFIRMATION ====================
// Simplified flow (as requested):
// 1) Ask: "Use these settings?"
//    - no  -> edit settings -> show summary -> repeat
//    - yes -> ask save-to-flash -> then return (mission arms after this)
void waitForConfirmation() {
  while (true) {
    String response = getYesNo("Use these settings? (yes/no)");

    if (response.equalsIgnoreCase("yes")) {
      blinkColor(ledGreen, 250);
      blinkColor(ledGreen, 250);

      response = getYesNo("Save these settings to flash for next boot? (yes/no)");
      if (response.equalsIgnoreCase("yes")) {
        if (saveSettings()) Serial.println("Saved to flash.");
        else Serial.println("FAILED to save settings.");
      } else {
        Serial.println("Not saving (revert to defaults or previous saved settings
next boot).");
      }
      return; // done: mission arms next
```

```
    }

    // "no" means: edit settings
    Serial.println("Entering configuration mode...");
    blinkColor(ledRed, 250);
    promptForSettings();
    printConfiguration();
  }
}

// ==================== ALARMS ====================
void scheduleNextAlarm() {
  DateTime now = rtc.now(), nextEvent;

  if (isFirstCycle) {
    nextEvent = DateTime(now.year(), now.month(), now.day(), cfg.startH, cfg.startM,
cfg.startS);
    if (nextEvent <= now) nextEvent = nextEvent + TimeSpan(1,0,0,0);
  } else {
    nextEvent = now + TimeSpan(0, cfg.intH, cfg.intM, cfg.intS);
  }

  rtc.setAlarm1(nextEvent, DS3231_A1_Date);
  rtc.clearAlarm(1);
  Serial.print("Next RTC alarm: "); Serial.println(nextEvent.timestamp());
}

// ==================== SETUP / LOOP ====================
void setup() {
  pinMode(MOSFET_PIN, OUTPUT);
  pinMode(ledGreen, OUTPUT);
  pinMode(ledRed, OUTPUT);
  pinMode(rtcAlarmPin, INPUT_PULLUP);

  Serial.begin(115200);
  delay(1500);

  blinkColor(ledGreen, 250); blinkColor(ledRed, 250);
  blinkColor(ledGreen, 250); blinkColor(ledRed, 250);

  setPower(false);
  Wire.begin();

  if (!rtc.begin()) {
    Serial.println("RTC not found!");
    for (int i = 0; i < 5; i++) { blinkColor(ledRed, 200); delay(100); }
  }

  rtc.clearAlarm(1);
  rtc.disableAlarm(1);
  rtc.writeSqwPinMode(DS3231_OFF);

  maybeSetRTC();  // handles prompt internally
```

```
  if (loadSettings()) Serial.println("Loaded settings from flash.");
  else Serial.println("No saved settings; using defaults.");

  printConfiguration();
  waitForConfirmation();  // simplified behavior

  scheduleNextAlarm();
  attachInterrupt(digitalPinToInterrupt(rtcAlarmPin), onRTCAlarm, FALLING);

  initialized = true;
  waitingForFirstCycle = true;
  isFirstCycle = true;

  Serial.println("Mission armed.");
}

void loop() {
  if (!initialized) return;

  if (rtcAlarmTriggered) {
    rtcAlarmTriggered = false;
    rtc.clearAlarm(1);

    DateTime now = rtc.now();
    DateTime expectedStart(now.year(), now.month(), now.day(), cfg.startH,
cfg.startM, cfg.startS);
    if (isFirstCycle && now < expectedStart) {
      Serial.println("RTC alarm early, ignored.");
      return;
    }

    isFirstCycle = false;
    waitingForFirstCycle = false;

    setPower(true);
    Serial.print("Power ON at: "); Serial.println(now.timestamp());
    blinkColor(ledGreen, 250);

    unsigned long onMs =
      (unsigned long)cfg.onH * 3600UL * 1000UL +
      (unsigned long)cfg.onM * 60UL * 1000UL +
      (unsigned long)cfg.onS * 1000UL;

    delay(onMs);

    setPower(false);
    Serial.println("Power OFF.");
    blinkColor(ledRed, 250);
    delay(100);

    scheduleNextAlarm();
  }
```

```
  if (waitingForFirstCycle && millis() - lastPreStartBlink >= 5000) {
    blinkColor(ledGreen, 200);
    lastPreStartBlink = millis();
  }

  if (DEPLOYMENT_MODE) __WFI();
  else delay(10);
}
```

9. Notes