Microchip Nano Debugger Manual

Nano Debugger

Product Page Links

Introduction

The Microchip Nano Debugger is a cost-effective programmer/debugger solution used in entry-level development kits made by Microchip and various partners. The most notable of these is the Curiosity Nano platform.

The Nano Debugger provides a unique balance of simplicity and versatility, which is needed in space- and cost-constrained development boards and tools.

About This Document

This Nano Debugger Manual is a generic user guide for the Nano Debugger, which is found on various Microchip and partner development boards. If you are using a debugger or development board that utilizes the Nano Debugger, then this document complements or replaces the debugger-specific sections of that product's user guide.

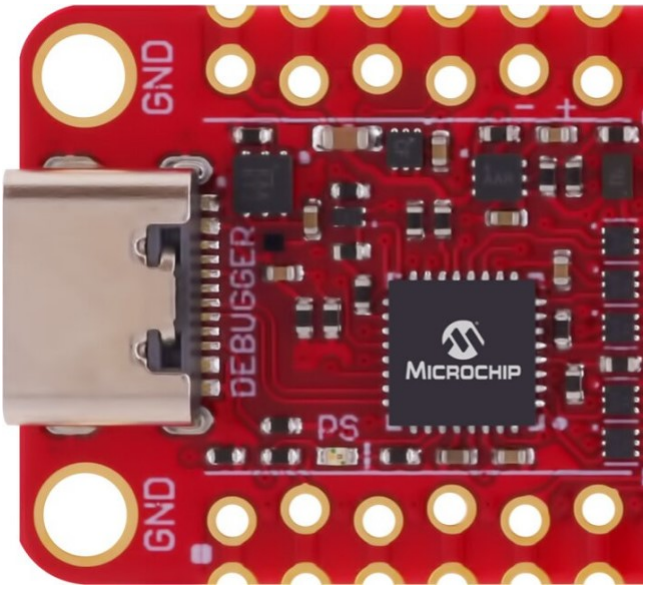


Table of Contents

1. Use Cases

The Nano Debugger can be used in many ways. Here are some examples.

Curiosity Nano MCU Boards

Curiosity Nano MCU boards are entry-level development boards that provide a uniform way to evaluate new silicon products from Microchip. The board pinout is standardized and mapped to the device pinout in a logical, systematic manner. This enables the use of a set of base boards to provide peripherals and features across all MCU boards.

All Curiosity Nano kits include a Nano Debugger.

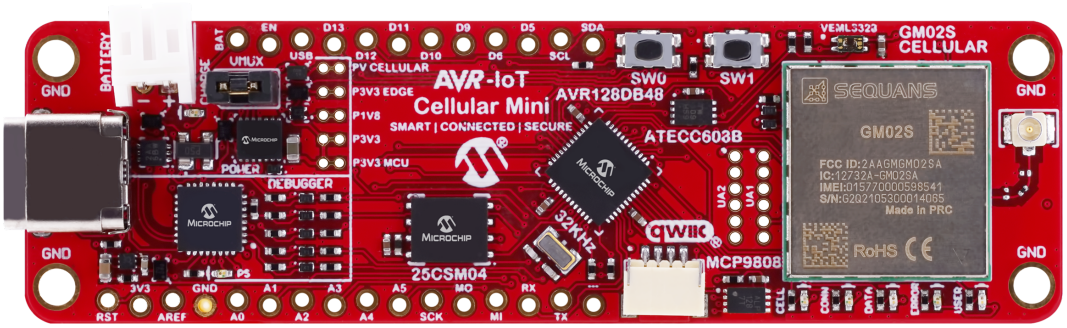
Figure 1-1. PIC32CM PL10 Curioisty Nano

Custom Development Kit

The Nano Debugger can be used on any development kit with an ARM® Cortex or modern AVR® microcontroller.

An example is the AVR-IoT Cellular Mini

Figure 1-2. AVR-IoT Cellular Mini



Tip: You can make your own development kit using the Nano Debugger by following the Nano Debugger Integration Guide.

Making a Standalone Debugger

When mounted on a development kit, the Nano Debugger provides both debugger and board control functionality. In a standalone implementation, the Nano Debugger no longer has full control of its environment and has no knowledge of the target device because it is not permanently mounted to the board. In this configuration, the Nano Debugger is a cost-effective debugger with an optional Virtual Serial Port bridge.

Tip: You can make your own standalone debugger using the Nano Debugger by following the Nano Debugger Integration Guide Nano Debugger Integration Guide

2. Capabilities

2.1. Features

Key Features of the Nano Debugger:

- Programming and Debugging of ARM Cortex Devices via the Industry-Standard CMSIS-DAP Interface
- Programming and Debugging of AVR and Selected PIC16, PIC18, PIC24 and dsPIC33 Devices via CMSIS-DAP Vendor Extensions
- Standard Virtual Serial Port (CDC)
- Mass Storage Implementation for Reading Kit Information and Drag-and-Drop Programming of Selected Device Families
- Data Gateway Interface for Simple Logic Analysis FunctionsBoard Control Functionality
- Open Protocol Implementation Supported in Many IDEs and Ecosystems
- Firmware Available for Integration into Custom Hardware Solutions Using a SAMD21 MCU

2.2. Target Devices

Programming and Debugging of ARM Cortex Devices

The Microchip Nano Debugger implements a standard CMSIS-DAP interface as specified by ARM. This means that it is inherently capable of programming and debugging any ARM Cortex-based device.

The CMSIS-DAP implementation is version 1, which uses a HID interface to communicate with the host PC.

SWO trace is not yet supported.

Tip: In a future firmware update, the Nano Debugger will support CMSIS-DAP version 2 using the bulk/vendor interface. For legacy IDE support (e.g., Microchip Studio), an older firmware must be used.

Table 2-1. ARM SWD Pinout

Target Signal	Purpose	Debugger Pin
SWDIO	Serial Wire Data	DBG0
SWCLK	Serial Wire Clock	DBG1
$\overline{\text{RESET}}$	Target reset	DBG3

Programming and Debugging of AVR Devices

The Microchip Nano Debugger can be used with any AVR device that has the Unified Programming and Debugging Interface (UPDI). This interface is a single-wire, asynchronous, UART-based protocol (v1), with some implementations requiring $\overline{\text{RESET}}$ as an additional signal (UPDI header version 2).

NOTICE Older AVR devices that use JTAG, debugWIRE, PDI, TPI, ISP, HVSP, or HVPP are not supported by the Nano Debugger.

Programming non-volatile memories using the UPDI interface is documented in the data sheet for the AVR device in question, while the debugging interface is not publicly distributed.

The AVR programming and debugging commands are implemented using Vendor Commands on the CMSIS-DAP specification. The command-set used for this protocol is documented in the EDBG-based Tools Protocols document and implemented in code in pyedbglib.

Tip: The Nano Debugger is an evolutionary extension of the EDBG and largely follows the same protocols, with some refinements and extensions.

Table 2-2. AVR UPDI Pinout

Target Signal	Purpose	Debugger Pin
UPDI	Programming and Debugging	DBG0
$\overline{\text{RESET}}$	Target reset (where applicable)	DBG3

Programming and Debugging of PIC® Devices

The Microchip Nano Debugger can be used with only a selection of the devices listed below. These devices all have the Microchip ICSP physical interface, but they differ logically between device families. This includes:

- PIC16 devices
- PIC18 devices
- PIC24 devices
- dsPIC33 devices

The Nano Debugger includes a scripting engine very similar to the one used for programming and debugging PIC devices with the MPLAB® PICkit™ 5 (and similar) tools. Python scripts are included in the scripts folder of the Device Family Pack for supported devices. The Python stack converts programming algorithms into bytecode for the scripting engine, which is passed to the Nano Debugger using the same Vendor Commands on the CMSIS-DAP interface.

Important: The PIC programming interface is documented in the device programming specifications, but debugging protocols are not published. The scripting language used for programming and debugging is proprietary and not published.

Table 2-3. PIC ICSP Pinout

Target Signal	Purpose	Debugger Pin
ICSPDAT	Programming data	DBG0
ICSPCLK	Programming clock	DBG1
$\overline{\text{MCLR}}$	Master clear (reset)	DBG3

3. Nano Debugger USB Interface

The Nano Debugger presents itself to the host computer as a composite USB device with 4 interfaces:

- CMSIS-DAP - an industry standard open specification provided by ARM for programming and debugging Cortex MCUs
- Virtual Serial Port (CDC) - a standard serial port interface
- Mass Storage Class device (MSC) - a standard removable storage 'disk'
- Data Gateway Interface - Microchip proprietary interface for streaming data to the host computer

Remember: Keep the debugger's firmware up to date. Firmware upgrades are automatic when using a current Microchip IDE. Alternatively, check for updated Tool Packs for **PKOB nano support** and use pydebuggerupgrade to manually upgrade the firmware.

4. CMSIS-DAP Debug Interface

The Nano Debugger is an implementation of the CMSIS-DAP debugger interface standard specified by ARM and will appear as a Human Interface Device (HID) on the host computer's USB subsystem.

HID devices typically don't need specific operating system drivers since they are widely used for mouse and keyboard applications. This also means that they may not obviously appear as Microchip devices in your system configuration or device manager.

Tip: For more information on device drivers, see section USB Drivers.

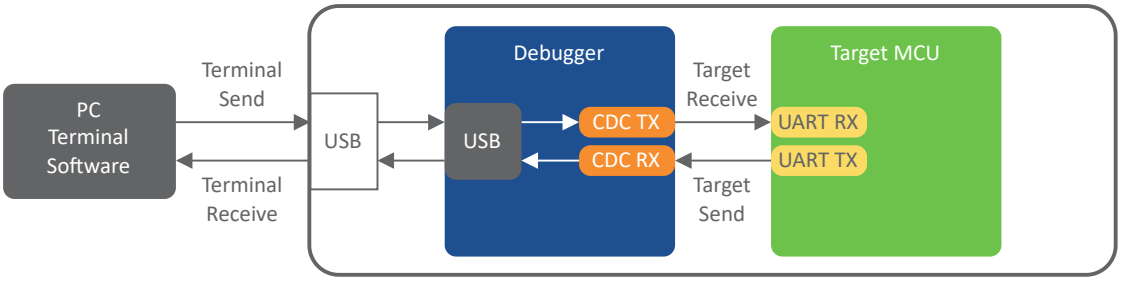
5. Virtual Serial Port (CDC)

The virtual serial port (CDC) is a general purpose serial bridge between a host PC and a target device.

5.1. Overview

The Nano Debugger implements a composite USB device with a standard Communications Device Class (CDC) interface, which appears on the host as a virtual serial port. Use the CDC to stream arbitrary data between the host computer and the target in both directions: All characters sent through the virtual serial port on the host computer will be transmitted as UART on the debugger's CDC TX pin. The UART characters captured on the debugger's CDC RX pin will be returned to the host computer through the virtual serial port.

Figure 5-1. CDC Connection



Info: The debugger's CDC TX pin is connected to a UART RX pin on the target for receiving characters from the host computer, as shown in the figure above. Similarly, the debugger's CDC RX pin is connected to a UART TX pin on the target for transmitting characters to the host computer.

5.2. Operating System Support

On Windows® machines, the CDC will enumerate as *Curiosity Virtual COM Port* and appear in the Ports section of the Windows Device Manager. The COM port number can also be found there.

Info: On older Windows systems, the CDC requires a USB driver. The MPLAB X IDE installation includes this driver.

On Linux® machines, the CDC will enumerate and appear as `/dev/ttyACM#`.

Info: tty* devices belong to the "dialout" group in Linux, so it may be necessary to become a member of that group to have permission to access the CDC.

On Mac® machines, the CDC will enumerate and appear as `/dev/tty.usbmodem#`. Depending on the terminal program used, it will appear in the available list of modems as `usbmodem#`.

5.3. Limitations

Not all UART features are implemented in the Nano Debugger CDC. The constraints are outlined here:

- **Baud rate**: Must be in the range of 1200 bps to 500 kbps. Any baud rate outside this range will be set to the closest limit without warning. The baud rate can be changed on the fly.
- **Character format**: Only 8-bit characters are supported.
- **Parity**: Can be odd, even or none.
- **Hardware flow control**: Not supported.
- **Stop bits**: One or two bits are supported.

5.4. Signaling

During USB enumeration, the host OS will start the communication and data pipes of the CDC interface. At this point, it is possible to set and read back the baud rate and other UART parameters of the CDC, but sending and receiving data will not be enabled.

The terminal must assert the DTR signal when it connects to the host. As this is a virtual control signal implemented on the USB interface, it is not physically present on the board. Asserting the DTR signal from the host will indicate to the Nano Debugger that a CDC session is active. The debugger will enable its level shifters (if available) and start the CDC data send and receive mechanisms.

Deasserting DTR in debugger firmware version 1.20 or earlier has the following behavior:

- The debugger UART receiver is disabled, and no more data will be transferred to the host computer
- The debugger UART transmitter will continue to send queued data ready for transfer, but no new data is accepted from the host computer
- Level shifters (if available) are not disabled, and the debugger CDC TX line remains driven

Deasserting DTR in debugger firmware version 1.21 or later has the following behavior:

- The debugger UART receiver is disabled, and no further data will be transferred to the host computer
- The debugger UART transmitter will continue to send queued data that are ready for transfer, but no new data will be accepted from the host computer
- Once the ongoing transmission is complete, level shifters (if available) are disabled, and the debugger CDC TX line will become high-impedance

Remember: Set up the terminal emulator to assert the DTR signal. Without this signal, the Nano Debugger will not send or receive data through its UART.

Tip: The Nano Debugger's CDC TX pin will not be driven until the CDC interface is enabled by the host computer. Also, there are no external pull-up resistors on the CDC lines connecting the debugger and the target, meaning the lines are floating during power-up. The target device may enable the internal pull-up resistor on the pin connected to the debugger's CDC TX pin to avoid glitches resulting in unpredictable behavior like framing errors.

5.5. Advanced Use

CDC Override Mode

In ordinary operation, the Nano Debugger is a UART bridge between the host and the device. However, in certain use cases, the Nano Debugger can override the basic Operating mode and use the CDC TX and RX pins for other purposes.

Dropping a text file into the Nano Debugger's mass storage drive can send characters out of the debugger's CDC TX pin. The filename and extension are trivial, but the text file will start with the characters:

```
CMD:SEND_UART=
```

Debugger firmware version 1.20 or earlier has the following limitations:

- The maximum message length is 50 characters – all remaining data in the frame are ignored
- The default baud rate used in this mode is 9600 bps, but if the CDC is already active or configured, the previously used baud rate still applies

Debugger firmware version 1.21 and later has the following limitations/features:

- The maximum message length will vary depending on the MSC/SCSI layer timeouts on the host computer and/or operating system. A single SCSI frame of 512 bytes (498 characters of payload) is ensured, and files up to 4 KB will work on most systems. The transfer will be completed on the first NULL character encountered in the file.
- The baud rate used is always 9600 bps for the default command:

  ```
  CMD:SEND_UART=
  ```

- Do not use the CDC Override mode simultaneously with data transfer over the CDC/terminal. If a CDC terminal session is active when receiving a file via the CDC Override mode, it will be suspended for the duration of the operation and resumed once complete.
- Additional commands are supported with explicit baud rates:

  ```
  CMD:SEND_9600=
  ```

  ```
  CMD:SEND_115200=
  ```

  ```
  CMD:SEND_460800=
  ```

USB-Level Framing Considerations

Sending data from the host to the CDC can be done byte-wise or in blocks, chunked into 64-byte USB frames. Each frame will be queued for transfer to the debugger's CDC TX pin. Sending a small amount of data per frame can be inefficient, particularly at low baud rates, as the Nano Debugger buffers frames but not bytes. A maximum of four 64-byte frames can be active at any time. The Nano Debugger will throttle the incoming frames accordingly. Sending full 64-byte frames containing data is the most efficient method.

When receiving data on the debugger's CDC RX pin, the Nano Debugger will queue up the incoming bytes into 64-byte frames, which are sent to the USB queue for transmission to the host when they are full. Incomplete frames are also pushed to the USB queue at approximately 100 ms intervals, triggered by USB start-of-frame tokens. Up to eight 64-byte frames can be active at any time.

An overrun will occur if the host (or the software running) fails to receive data fast enough. When this happens, the last-filled buffer frame recycles instead of being sent to the USB queue, and a complete data frame will be lost. To prevent this, the user must ensure that the CDC data pipe is continuously read, or alternatively reduce the incoming data rate.

Sending Break Characters

The host can send a UART break character to the device using the CDC, which can be usable for resetting a receiver state-machine or signaling an exception condition from the host to the application running on the device.

A break character is a sequence of at least 11 zero bits transmitted from the host to the device.

Not all UART receivers have support for detecting a break, but a correctly-formed break character usually triggers a framing error on the receiver.

Sending a break character using the debugger's CDC has the following limitations:

- A break must NOT be sent simultaneously with the use of CDC Override mode (drag-and-drop). Both these functions are temporary states and must be used independently.
- Sending a break will cause any data being sent to be lost. Be sure to wait a sufficient amount of time to allow all characters in the transmission buffer to be sent (see above section) before sending the break, which is also in line with expected break character usage. For example, reset a receiver state-machine after a timeout occurs waiting for returning data to the host.
- The CDC specification allows for debugger-timed breaks of up to 65534 ms in duration to be requested. For simplicity, the debugger will limit the break duration to a maximum of 11 bit-durations at its minimum supported baud rate.
- The CDC specification allows for indefinite host-timed breaks. In this case, it is the responsibility of the terminal application or user to release the break state.

Note: Sending break characters is available in debugger firmware version 1.24 and later.

6. Mass Storage Device

The Nano Debugger includes a simple Mass Storage Device implementation, which is accessible for read/write operations via the host operating system to which it is connected.

It provides:

- Read access to basic text and HTML files for detailed kit information and support
- Write access for programming Intel® HEX and UF2 formatted files into the target device's memory
- Write access for simple text files for utility purposes

Note: Support for the UF2 format is available in debugger firmware version 1.31 or later.

6.1. Mass Storage Device Implementation

The Nano Debugger implements a highly optimized variant of the FAT12 file system with several limitations, partly due to the nature of FAT12 itself and optimizations made to fulfill its purpose for its embedded application.

The Curiosity Nano USB device is USB Chapter 9-compliant as a mass storage device but does not, in any way, fulfill the expectations of a general purpose mass storage device. This behavior is intentional.

When using the Windows operating system, the Nano Debugger enumerates as a Curiosity Nano USB Device found in the disk drives section of the device manager. The CURIOSITY drive appears in the file manager and claims the following available drive letter in the system.

The CURIOSITY drive contains approximately 1 MB of free space and does not reflect the target device's Flash size. When programming an Intel HEX or UF2 file, the binary data are encoded in ASCII with metadata providing significant overhead. Therefore, 1 MB is an arbitrarily chosen value for the disk size.

It is not possible to format the CURIOSITY drive. When programming a file to the target, the filename may appear in the disk directory listing; however, this is merely the operating system's view of the directory, which in reality has not been updated. It is not possible to read out the file contents. Removing and replugging the board will return the file system to its original state, but the target will still contain the previously programmed application.

Copy a text file starting with "`CMD:ERASE`" onto the disk to erase the target device.

By default, the CURIOSITY drive contains several read-only files used for generating icons, reporting status, and linking to further information:

- `AUTORUN.ICO` – icon file for the Microchip logo
- `AUTORUN.INF` – system file required for Windows Explorer to show the icon file
- `CLICK-ME.HTM` – redirect to a kit-specific web demo application
- `KIT-INFO.HTM` – redirect to the development board website
- `KIT-INFO.TXT` – a text file containing details about the board's debugger firmware version, board name, USB serial number, device, and drag-and-drop support
- `PUBKEY.TXT` – a text file containing the public key for data encryption
- `STATUS.TXT` – a text file containing the programming status of the board

Info: The Nano Debugger dynamically updates `STATUS.TXT`. However, the contents may not reflect the correct status because the operating system may cache the file.

Tip: The presence of the CLICK-ME and PUBKEY files depends on the debugger board configuration.

6.2. Fuse/Configuration Bytes/Words

Fuse Bytes (AVR® MCU Targets)

When performing drag-and-drop programming, the Nano Debugger masks out any fuse bits that attempt to disable Unified Program and Debug Interface (UPDI), meaning that the UPDI pin cannot be used in its Reset or GPIO modes. Selecting one of the alternative functions of the UPDI pin will render the device inaccessible without using an external debugger capable of high-voltage UPDI activation.

Configuration Bytes/Words (PIC® MCU Targets)

When performing drag-and-drop programming, the debugger masks out any configuration bytes or words that attempt to disable the ICSP interface, or other One-Time Programmable (OTP) configurations.

6.3. Limitations of Drag-and-Drop Programming

ARM Cortex Devices

Drag-and-drop programming is not supported on ARM Cortex devices. Use one of the supported IDEs or command-line utilities for programming.

Lock Bits

Lock bits included in the hex file will be ignored when using drag-and-drop programming. To program lock bits, use one of the supported IDEs.

Enabling CRC Check in Fuses

It is not advisable to enable the CRC check fuses in the target device when using drag-and-drop programming because a subsequent chip erase (which does not affect fuse bits) will cause a CRC mismatch, and the application will fail to boot. A chip erase must be performed using a supported IDE, which automatically clears the CRC fuses after erasing to recover the target from this state.

6.4. Special Commands

Several utility commands are supported by copying text files to the mass storage disk. The filename or extension is irrelevant – the command handler reacts to content only.

Table 6-1. Special File Commands

Command Content	Description
`CMD:ERASE`	Executes a target chip erase
`CMD:SEND_UART=`	Sends a string of characters to the CDC UART. See Advanced Use.
`CMD:SEND_9600=` `CMD:SEND_115200=` `CMD:SEND_460800=`	Sends a string of characters to the CDC UART at the specified baud rate. Note that only the baud rates explicitly specified here are supported. See Advanced Use (Debugger firmware v1.25 or newer.)
`CMD:RESET`	Resets the target device by entering Programming mode and then exiting Programming mode immediately afterwards. The exact timing may vary depending on the programming interface of the target device. (Debugger firmware v1.25 or newer.)

Table 6-1. Special File Commands (continued)

Command Content	Description
CMD:POWERTOGGLE	Powers down the target and restores it after a 100 ms delay. If external power is provided, this command has no effect. (Debugger firmware v1.25 or newer.)
CMD:0V	Powers down the target device by disabling the target supply regulator. If external power is provided, this has no effect. (Debugger firmware v1.25 or newer.)
CMD:1V8	Sets the target voltage to 1.8V. If using external power, this has no effect. (Debugger firmware v1.25 or newer.)
CMD:3V3	Sets the target voltage to 3.3V. If using external power, this has no effect. (Debugger firmware v1.25 or newer.)
CMD:5V0	Sets the target voltage to 5.0V. If using external power, this command has no effect. (Debugger firmware v1.25 or newer.)

Info: The content sent to the emulated mass storage disk disk triggers the commands listed in the table above and does not provide feedback in the case of either success or failure.

6.5. Drag-and-Drop Programming Using UF2 Format

UF2 Format for Drag-and-Drop

Drag-and-Drop programming provides a simple mechanism for programming the non-volatile memories of the microcontroller on a development kit. This is typically done using the Intel® HEX format, which includes the necessary addresses and segmentation information as part of the format. Intel HEX files contain the memory contents encoded as ASCII characters, which must be parsed in strictly sequential order. This means that the host operating system must send the content in the correct sequence. This is not always the case for all variants of all operating systems. The UF2 format was developed by Microsoft as a means to allow memory to be transferred out of sequence. This is achieved by enforcing a fixed block size throughout the entire data transfer path, thereby preventing partial writes.

More information on the UF2 format is available on GitHub.

Generating a UF2 file

The result of a project compilation procedure is typically an Intel hex file, which can be converted into a UF2 file using a post-build step.

The pymcuprog package distributed on pypi.org includes a function to convert an Intel hex file to a UF2 file in version 3.17 or later.

Tip: This procedure requires the installation of a recent release of Python 3 and the pymcuprog package in that environment, as well as ensuring that the Python scripts folder is included in the system or user path.

An Intel HEX file can be converted into a UF2 file using the pymcuprog command-line interface:

```
pymcuprog makeuf2 -f app.hex --uf2file app.uf2
```

This process can be streamlined by adding a post-build step in MPLAB X IDE. In the **Project Properties** configuration dialog, select the **Building** tab and check **Execute this line after build**. Add the command to the edit box:

```
pymcuprog makeuf2 -f ${ImagePath} --uf2file ${ImageDir}\${ProjectName}.X.${IMAGE_TYPE}.uf2
```

Each time the application is built, the produced Intel hex file will automatically be converted to a UF2 file with the same filename but with a .uf2 file extension.

7. Data Gateway Interface (DGI)

Data Gateway Interface (DGI) is a USB interface that transports raw and timestamped data between the Nano Debugger and host computer-based visualization tools. MPLAB Data Visualizer is used on the host computer to display any debug GPIO data. It is available as a plug-in for MPLAB X IDE or as a stand-alone application that can be used in parallel with MPLAB® X IDE.

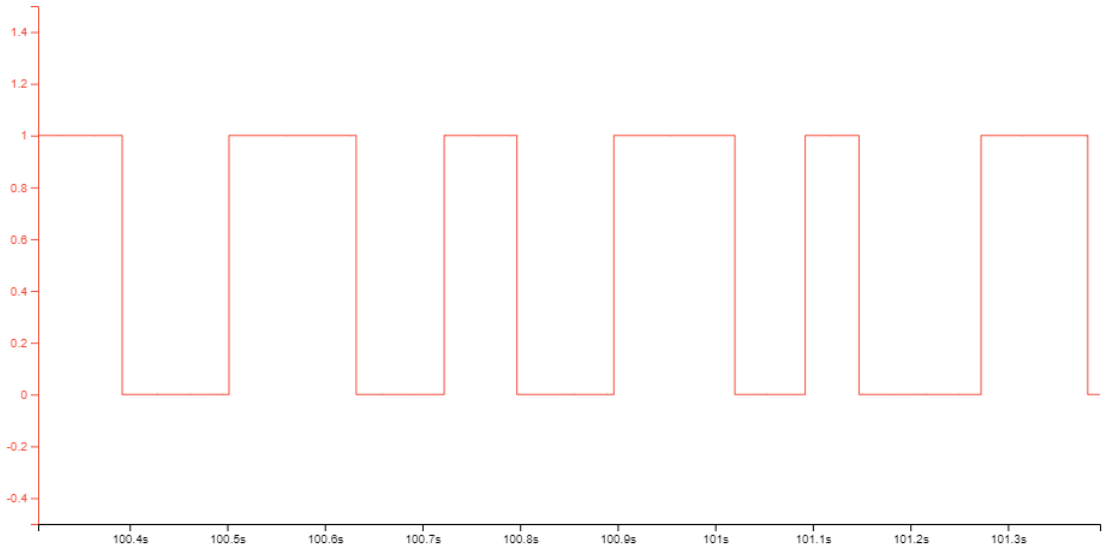
Tip: The number of GPIO channels can vary depending on the device mounted on the kit. AVR devices have two channels, while all other device types have a single channel.

7.1. Debug GPIO

Debug GPIO channels are timestamped digital signal lines that connect the target application to a host computer visualization application. They are typically used to plot low-frequency events on a time axis, such as when specific application state transitions occur.

The figure below shows the monitoring of the digital state of a mechanical switch connected to a debug GPIO in MPLAB Data Visualizer.

Figure 7-1. Monitoring Debug GPIO with MPLAB Data Visualizer



Debug GPIO channels are timestamped, so the resolution of DGI GPIO events is determined by the DGI Timestamp module resolution.

Important: Although capturing higher-frequency signal bursts is possible, the frequency range in which the debug GPIO can be used is up to about 2 kHz. Attempting to capture signals above this frequency will result in data saturation and overflow, which may cause the DGI session to abort.

7.2. Timestamping

When captured by the debugger, DGI sources are timestamped. The timestamp counter implemented in the Curiosity Nano debugger increments at a frequency of 2 MHz, providing a timestamp resolution of half a microsecond.

8. Board Controller Functions

8.1. Voltage Monitor

The Nano Debugger uses an ADC channel to monitor the voltage on the target device. Reading the target voltage is controlled by the TVR (Target Voltage Readout) field in the board configuration - if this feature bit is cleared, then voltage monitoring is disabled.

The voltage monitor is a background task running in the Nano Debugger firmware that samples the ADC at regular intervals.

When used as an on-board debugger, the board designer sets the VMIN field in the board configuration, which specifies the minimum operating voltage value. Below this voltage, the Voltage Monitor registers that the target voltage is off. Handling of attempted programming and debugging operations in the off-state depends on the target device type.

The current operating voltage can be read out using a Microchip IDE or pymcuprog:

```
pymcuprog getvoltage
```

To read the set-point for the supply voltage, use:

```
pymcuprog getsupplyvoltage
```

USB Voltage

The Nano Debugger can also read the voltage on the USB VBUS line, which is its power source. This is primarily a convenience feature.

The USB voltage can be read using a Microchip IDE or pymcuprog:

```
pymcuprog getusbvoltage
```

8.2. Voltage Control

The Nano Debugger can supply the target device with its own voltage source. This is optional for both on-board and standalone debugger configurations.

To change the operating voltage, use one of the Microchip IDEs or pymcuprog:

```
pymcuprog --setsupplyvoltage
```

Tip: There is an easy option to adjust the target voltage by copying a drag-and-drop command text file to the board, which supports a set of commonly used target voltages. See section Special Commands for further details.

The mechanism used by the Nano Debugger for controlling target voltage depends upon the board configuration in place.

Table 8-1. Board configuration for voltage control

Field	Description	Notes
ANALOG_FEATURES: TVS	Target voltage set	Indicates that the target voltage can be controlled
VMIN	Minimum operating voltage	Setting the voltage below this value will fail
VMAX	Maximum operating voltage	Setting the voltage above this value will fail
VTG	Default operating voltage	Voltage used when no voltage is set
VREG	Voltage regulator configuration	Indicates which regulator is mounted on the board

Voltage Regulator Configuration 0 - None

No voltage control is available when this configuration is used.

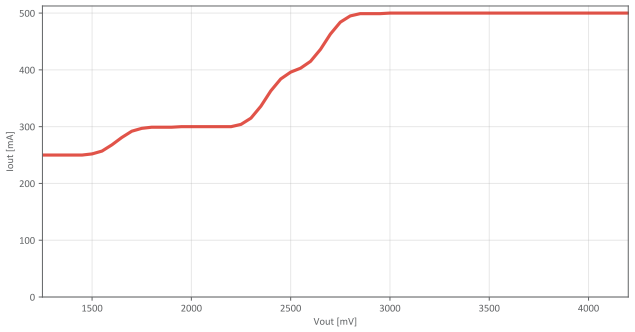
Voltage Regulator Configuration 1 - Curiosity Nano
Note: This configuration is used on Curiosity Nano boards (adjustable voltage boards).

When the VREG board configuration is set to 0x01, the Nano Debugger uses a DAC to regulate the voltage on the target device by manipulating the voltage on the feedback pin of a low drop-out regulator. The Nano Debugger monitors the output of this regulator as described in Voltage Monitor, and regulates the voltage accordingly.

The target voltage regulator is a MIC5353 variable output LDO. The Nano Debugger can adjust the voltage output supplied to the board's target section by manipulating the MIC5353's feedback voltage. The hardware implementation is limited to an approximate voltage range of 1.7V to 5.1V.

MIC5353 supports a maximum current load of 500 mA. It is an LDO regulator in a small package, placed on a small printed circuit board (PCB), and can reach thermal shutdown at loads lower than 500 mA. The maximum current load depends on the input voltage, the selected output voltage, and the ambient temperature. The figure below shows the safe operating area for the regulator, with an input voltage of 5.1V and an ambient temperature of 23°C.

Figure 8-1. Target Regulator Safe Operation Area



NOTICE The profile shown here was measured on a Curiosity Nano board and may vary depending on the PCB.

The voltage output of the target regulator is continuously monitored (measured) by the Nano Debugger. An error condition is flagged if a deviation of 100 mV above or below the expected voltage is detected. In this case, the voltage regulator will be switched off.

NOTICE The error detection will also handle deviations in an externally applied voltage. For more information, see Voltage Off Pin (VOFF).

8.3. Voltage Off Pin (VOFF)

NOTICE The VOFF pin is only relevant when an adjustable regulator is used.

Instead of using the on-board target regulator, an external voltage can be used to power the target device. When the Voltage Off (VOFF) pin is shorted to the ground (GND) pin, the Nano Debugger firmware disables the target regulator, making it safe to apply an external voltage to the VTG pin.

The VOFF pin is used to allow a Curiosity Nano base board to provide power to a mounted Curiosity Nano kit. Voltage is applied via the VTG pin; otherwise, the two power supplies would be in contention. The base board pulls the VOFF pin low when it is connected.

The VOFF pin can be tied low or released at any time, and this change will be detected by a pin-change interrupt to the Nano Debugger, which will then control the target voltage regulator accordingly.

WARNING Applying an external voltage to the VTG pin without shorting VOFF to GND may cause permanent damage to the board.

WARNING Do not apply any voltage to the VOFF pin. Let the pin float to enable the power supply.

WARNING Applying a higher voltage than the maximum operating voltage of the target device may cause permanent damage to the board.

Info: If the Nano Debugger shuts off the target regulator, it will begin blinking the status LED rapidly, to indicate an error condition. Once the error is resolved the Nano Debugger will switch on the target regulator and stop blinking the status LED.

Programming, debugging, and data streaming are still possible with an external power supply. The USB cable powers the debugger and signal level shifters. Both regulators, the debugger, and the level shifters are powered down when the USB cable is removed.

Info: In addition to the power consumed by the target device, approximately 100 µA will be drawn from any external power source to power the on-board level shifters and voltage monitor circuitry when a USB cable is plugged into the DEBUG connector on the board. When the USB cable is unplugged, some current is used to supply the level shifter's voltage pins, with a worst-case current consumption of approximately 5 µA. Typical values may be as low as 100 nA.

8.4. Voltage Control Errors

This section summarizes the most common issues that can arise with the power supply.

Target Voltage Shuts Down

Not reaching the set target voltage can occur if the target section draws too much current at a given voltage, causing the thermal shutdown safety feature of the MIC5353 regulator to activate. To avoid this, reduce the current load of the target section.

Target Voltage Setting is Not Reached

The USB input voltage (specified to be 4.4-5.25V) limits the maximum output voltage of the MIC5353 regulator at a given voltage setting and current consumption. If a higher output voltage is needed, use a USB power source with a higher input voltage or an external voltage supply on the VTG pin.

Target Voltage is Different From Setting

An externally applied voltage to the VTG pin without setting the VOFF pin low can cause this. If the target voltage fluctuates by more than 100 mV above or below the voltage setting, the Nano Debugger will detect it, and the internal voltage regulator will shut down. To fix this issue, remove the applied voltage from the VTG pin, and the Nano Debugger will enable the voltage regulator when the new condition is detected. Note that the PS LED will blink rapidly if the target voltage is below 100 mV of the setting but will remain on if it is more than 100 mV above it.

No, or Very Low Target Voltage and PS LED is Blinking Rapidly

A full or partial short circuit can cause this and is a specific instance of the issue described above. Remove the short circuit, and the Nano Debugger will re-enable the target voltage regulator.

No Target Voltage and PS LED is Lit 1

This situation occurs if the target voltage is set to 0.0V. To fix this, set the target voltage to a value within the specified voltage range for the target device.

No Target Voltage and PS LED is Lit 2

This situation can occur when power jumper(s) are cut and the target voltage regulator is set to a value within the specified voltage range for the target device. To fix this, solder a wire or bridge between the pads or, if a pin header is mounted, add a jumper.

VBUS Output Voltage is Low or Not Present

If the VBUS output voltage is low or missing, the most likely reason is a high current drain on VBUS, which has caused the current limiter to trip and cut off VBUS completely. To fix this issue, reduce the current consumption on the VBUS pin.

8.5. ID System

The ID line on the Nano Debugger supports a mechanism used to identify base boards and extensions that are connected to the Xplained Pro (XPRO) extension header on those base boards.

Note: Only Microchip Xplained Pro extensions and Curiosity Nano base boards feature the identification system.

During power-up, the Nano Debugger scans the ID line for both base boards and extensions and presents this information to the IDE upon request.

Note: An extension or base board connected after powering up the Nano Debugger will not be detected.

Tip: For more information on the ID system, see the Xplained Pro Hardware Development Kit (HDK).

9. Debugger Configuration

The Nano Debugger uses a portion of its internal non-volatile memory to store information about the hardware it interacts with. This content is typically programmed during manufacturing but can also be modified by end users. This configuration is preserved when debugger firmware is updated, and is thus an important mechanism to prevent specific customizations from being made in the firmware itself.

Debugger configuration includes two parts:

- Board configuration describes the board on which the Nano Debugger is mounted and is **mandatory**
- Device configuration describes the device that is permanently mounted to a kit and is **optional**

Tip: If no device configuration is needed, a dummy configuration is used. This indicates to the Nano Debugger that drag-and-drop programming is not available.

9.1. Board Configuration

The board configuration (or board-config) of the Nano Debugger resides in a section of non-volatile memory. The board configuration format is specified in XML, with various offsets in this section allocated to different fields and bits. The API to the host computer is simply a block read/write operation with no knowledge of the format. The meaning of the fields is defined by the Python files (for the host computer) and header files (for the Nano Debugger) generated from the configuration XML itself. New fields can be added without changing the API by allocating unused space in the XML and regenerating the Python and header files. The XML schema uses semantic versioning (semver) for compatability.

Table 9-1. Board Configuration 1.14

Register	Size	Description	What its for/Default Value
DEVNAME	Up to 32 ASCII characters	Device name	Specify the exact device name here if a device is permanently mounted on the board. If the target device is not mounted on the board this field must be blank.
KITNAME	Up to 60 ASCII characters	Kit name	Give your kit a nice name here
MNFRNAME	Up to 60 ASCII characters	Manufacturer name	Specify who you are as a vendor
SERNUM	20 characters	USB serial number	A specific USB serial number can be set here. It is recommended to use exactly 20 upper-case alphanumeric characters and no spaces. Setting the "automatic serial number" bit will override this at enumeration time, but the original value will remain in the configuration in flash memory.
REDIRECT	4 characters	Redirect ID	See note below

Table 9-1. Board Configuration 1.14 (continued)

Register	Size	Description	What its for/Default Value
DATE	8 digits	Manufacturing date	Optionally specify the manufacturing date of the kit in format YYYYMMDD
INSTANCE	1 byte	Content revision	This is a simple version field, and can be incremented for each change made to the configuration
TARGET_DEBUG_INTERFACES	4 bytes	Set of programing and debugging interfaces supported	Set the bits in this bitfield to indicate which physical interface(s) are supported in this implementation. Relevant bits are: • Bit 1: SWD for ARM devices • Bit 8: UPDI for AVR devices • Bit 10: SWO for ARM devices • Bit 11: ICSP for PIC and dsPIC® devices
TARGET_DEBUG_FEATURES	1 byte	Set of programming and debugging features implemented	Set the bits in this bitfield to enable individual features: Relevant bits are: • Bit 0: SINGLE_DEVICE - set this bit if the device is permanently mounted to the board • Bit 1: PROG_ENABLED - set this bit to enable programming • Bit 2: DEBUG_ENABLED - set this bit to enable debugging • Bit 3: FUSE_CONFIG_PROTECTION - set this bit to enable protection of fuses and configuration bytes. A set of protection masks must be provided in the device configuration for this to work. • Bit 4: AUTO_SERIAL_NUMBER - set this bit to automatically generate a USB serial number
DGI_INTERFACES	4 bytes	Set of DGI interfaces supported	Set the bits in this bitfield to indicate to the debugger which DGI interfaces are connected. Relevant bits are: • Bit 0: GPIO

Table 9-1. Board Configuration 1.14 (continued)

Register	Size	Description	What its for/Default Value
DGI_GPIO_MAP	1 byte	Map of DGI GPIO channels	Set the bits in this bitfield to indicate to the debugger which DGI GPIO lines are connected. Relevant bits are: • Bit 0: DGI_GPIO0 • Bit 1: DGI_GPIO1
ANALOG_FEATURES	4 bytes	Analog features	Set the bits in this bitfield to indicate to the debugger which analog implementation is present on the board. Relevant bits are: • Bit 0: TVS - the debugger can control the target voltage • Bit 1: TVR - the debugger can measure the target voltage • Bit 2: LVC - level shifters are present between debugger and the target
VMIN	1 byte	Minimum voltage	For adjustable-voltage boards, specify the minimum voltage in deciVolts
VMAX	1 byte	Maximum voltage	For adjustable-voltage boards, specify the maximum voltage in deciVolts
VTG	1 byte	Default voltage	For adjustable-voltage boards, specify the default voltage in deciVolts
VREG	1 byte	Voltage regulator	Specify the voltage regulation circuit implemented Values are: • 0x00 - not adjustable • 0x01 - adjustable regulator configuration 1 controlled by the Nano Debugger
ID_CHANNELS	1 byte	ID channels	Specify which ID channels are implemented • Bit 0: ID channel 1
CONFIG_FORMAT_MAJOR CONFIG_FORMAT_MINOR CONFIG_FORMAT_BUILD	4 bytes	The version of the configuration specification being used. If new fields are added, this version is updated and this document will also be updated.	(latest)
HARDWARE_MOD	1 byte	Specify hardware modifications which affect the firmware	0x00
USB_ENUMERATION	1 byte	USB enumeration	0x03
MSD_SETTINGS	1 byte	Mass storage content settings	0x00
CLICK_ME_TYPE	1 byte	Click-me type configuration	0xFF

Table 9-1. Board Configuration 1.14 (continued)

Register	Size	Description	What its for/Default Value
I2C_ADDR_0, I2C_ADDR_1	1 byte	I2C address, alternative I2C address	0x00
LABS	1 byte	Activate experimental firmware features	0x00
KEY1_TYPE	1 byte	Key type: • 0: None • 1: 48-bit MAC • 2: 64-bit MAC • 3: UUID • 4: UID • 5: PUBKEY	IoT kits
KEY2_TYPE	1 byte	Key type: • 0: None • 1: 48-bit MAC • 2: 64-bit MAC • 4: UID	IoT kits
KEY3_TYPE	1 byte		IoT kits
KEY4_TYPE	1 byte		IoT kits
KEY1_LEN	1 byte	Up to 64 bytes	Length of content of KEY1
KEY2_LEN	1 byte	Up to 32 bytes	Length of content of KEY2
KEY3_LEN	1 byte	Up to 16 bytes	Length of content of KEY3
KEY4_LEN	1 byte	Up to 16 bytes	Length of content of KEY4
KEY1	64 bytes	Key 1 value	-
KEY2	32 bytes	Key 2 value	-
KEY3	16 bytes	Key 3 value	-
KEY4	16 bytes	Key 4 value	-

Note: The REDIRECT_ID field is used to populate the URL on the USB Mass Storage disk, redirecting via kits.microchip.com to a product-specific web page. Contact Microchip support or edbg@microchip.com if you want to customize a Nano Debugger for your product.

9.2. Device Configuration

The device configuration of the Nano Debugger provides information about the device that is permanently connected to the debugger. Its main function is to support drag-and-drop programming, which needs to be self-contained (no IDE is involved). The device configuration can be empty (unprogrammed), but it is recommended to program a null or dummy device configuration in cases where drag-and-drop programming is not supported (e.g., ARM target devices) or when a target device is not mounted (e.g., a standalone debugger scenario).

Device configuration has only a single instance, and is not intended to be modified by the user.

The images in the Nano Debugger firmware zip include the null device configuration.

The pydebuggerconfig tool available on pypi.org can be used to write or replace the device config on a Nano Debugger.

Device Configuration - Advanced Information

Important: This information is for reference only. The device configuration is not intended to be user-serviced.

The device configuration is a 3 kB section in non-volatile memory on the Nano Debugger that is preserved during debugger upgrades.

Table 9-2. Header Structure

Offset	Type	Field	Description
0	uint8	major version	Major version of device-specific configuration specification
1	uint8	minor version	Minor version
2	uint16	build number	Build number of specification
4	uint16	content length	Number of bytes of actual content (after header)
6	uint16	content checksum	Checksum over content (not currently validated)
8	uint8	instance	Version of this config instance. Sequentially increasing.
9	7 bytes	reserved	For future use

Table 9-3. Version 1 Encoding

Offset	Type	Field	Description
16	uint8 (enum)	programming interface type	0x00: None 0x01: AVR UPDI by EDBG API 0x02: reserved 0x03: PIC by GEN4 binary code 0x04: PIC by GEN4 primitives 0x05-0xFF: reserved
17	uint8 (enum)	device sub-variant	For PIC devices: • 0x00: PIC16 • 0x01: PIC18 • 0x02: PIC24/dsPIC • 0x03-0xFF: reserved
18	14 bytes	reserved	For future use

After these headers, a lookup table of relative offsets of binary objects (BLOB) is encoded

Table 9-4. BLOB Table Encoding

Offset	Type	Field	Description
32	uint8	'L'	List token
33	uint8	size	Number of items
34	uint16	offset	Offset of item 1
...	...	...	...
34+2n	uint16	offset	Offset of item n

Each BLOB is encoded

Table 9-5. BLOB Header Encoding

Offset	Type	Field	Description
0	uint8	Structure type	• 'S': binary object from GEN4 script • 'D': device context (as used for AVR)
1	uint8	ID	eg: script ID
2	uint16	size	data length in bytes
...		data	binary data

Table 9-6. GEN4 Script Binary Types

ID	Function
0	Binary code for enter programming mode
1	Binary code for get device ID
2	Binary code for erase chip
3	Binary code for program flash memory
4	Binary code for program config words
5	Binary code for exit programming mode
6	Binary code for read flash memory

Device Context (parametric device data)

The Device Context for AVR devices is described in the Embedded Debugger-Based Tools Protocol User's Guide and in pymcuprog.

Table 9-7. Device Context for AVR devices

Offset	Size	Description
0x00	uint16	Base address for Program Flash memory
0x02	uint8	Flash page bytes
0x03	uint8	EEPROM page bytes
0x04	uint16	Address of NVMCTRL module
0x06	uint16	Address of OCD module
0x08	uint16	reserved
0x0A	uint16	reserved
0x0C	uint16	reserved
0x0E	uint16	reserved
0x10	uint16	Maximum frequency of PDI pin
0x12	uint32	Flash size in bytes
0x16	uint16	EEPROM size in bytes
0x18	uint16	User Row size in bytes
0x1A	uint8	Fuse memory size in bytes
0x1B	uint8	Offset of SYSCFG0 or PINCFG fuse within fuse area
0x1C	uint8	AND mask to apply to SYSCFG/PINCFG value when writing
0x1D	uint8	OR mask to apply to SYSCFG/PINCFG value when writing
0x1E	uint8	AND mask to apply to SYSCFG/PINCFG value after erase
0x1F	uint8	OR mask to apply to SYSCFG/PINCFG value after erase
0x20	uint16	Base address of EEPROM memory
0x22	uint16	Base address of User Row memory
0x24	uint16	Base address of Signature Row memory
0x26	uint16	Base address of Fuse memory
0x28	uint16	Base addrses of Lockbits memory
0x2A	uint16	Device ID
0x2C	uint8	MSB of base address of Program Flash memory (extends offset 0x00)
0x2D	uint8	MSB of flash page size in bytes (extends offset 0x02)
0x2E	uint8	Addressing mode of UPDI: • 0x00: use 16-bit addressing • 0x01: use 24-bit addressing
0x2F	uint8	High-voltage implementation
0x30	uint16	Base address of Boot Row

Table 9-7. Device Context for AVR devices (continued)

Offset	Size	Description
0x32	uint16	Boot Row size in bytes
0x34	uint16	Fuse protection mask. One bit per fuse byte (from LSB): • 0: write allowed • 1: protected

The Device Context for PIC devices is described here:

Table 9-8. Device Context for PIC Devices

Offset	Size	Description
0x00	uint32	Base address for PFM (Program Flash Memory)
0x04	uint32	Base address for EEPROM memory
0x08	uint32	Base address for User Row memory
0x0C	uint32	Base address for Configuration memory
0x10	uint32	Flash size in bytes
0x14	uint16	EEPROM size in bytes
0x16	uint16	User Row size in bytes
0x18	uint16	Configuration memory size in bytes
0x1A	uint16	Flash write block size in bytes
0x1C	uint16	EEPROM write block size in bytes
0x1E	uint8	User Row write block size in bytes
0x1F	uint8	Configuration memory write block size in bytes
0x20	uint16	Device ID
0x22	uint32	Configuration memory protection mask. One bit per configuration location (from LSB): • 0: write allowed • 1: protected

9.3. Modifying the Debugger Configuration

The board configuration and device configuration can be changed by the end user using the pydebuggerconfig utility.

For more general information about using pydebuggerconfig, see the information on pypi.org.

Use-Case: Altering the Configuration for Device Protection

When used in conjunction with AVR devices that are permanently mounted to a kit, the Nano Debugger includes a protection mechanism intended to prevent the MCU from becoming unrecoverable, especially when using drag-and-drop programming. Disabling the programming and debugging interface or permanently locking memories are features of some MCUs that are not conducive to evaluation on a development kit platform.

The protection mechanism works by intercepting write operations to the relevant fuses or configuration bits and conditionally masking the addresses and values written.

Info: The protection mechanism is intended to prevent making *accidental* irreversible changes. Users intentionally making irreversible changes do so at their own risk.

Info: The available feature-set and corresponding prevention mechanisms depend on the device. Check the data sheet for further information.

The protection mechanism can be disabled for users who want to have the full experience of features that require irreversible changes. Doing so will result in permanent changes.

pydebuggerconfig can be used to tweak many aspects of the debugger, including configuration protection. This procedure requires the installation of a recent release of Python 3 and the pydebuggerconfig package in that environment.

Step 1: Current Status

Determine whether protection is currently enabled by executing:

```
pydebuggerconfig read
```

Next, check for the section:

```
Register TARGET_DEBUG_FEATURES:   0x0F (15)  # Program/debug features
            bit 0, SINGLE_DEVICE: 1 # Single-device
            bit 1, PROG_ENABLED: 1 # Programming
            bit 2, DEBUG_ENABLED: 1 # Debug
            bit 3, FUSE_PROTECTION: 1 # Fuse protection
```

A FUSE_PROTECTION value of '1' indicates that protection is in place.

Step 2: Modifying the Protection Setting

Calculate the desired setting for TARGET_DEBUG_FEATURES by setting FUSE_PROTECTION to '0'. Replace the current value in the TARGET_DEBUG_FEATURES register by executing, for example:

```
pydebuggerconfig replace -r TARGET_DEBUG_FEATURES=0x07
```

To verify the new value, repeat the process from step 1 and then toggle the power on the kit. All protection mechanisms are now disabled.

Step 3: Restoring Protection

Protection can be re-enabled either by repeating step 2 with the initial value or using the factory-restore function by executing:

```
pydebuggerconfig restore
```

Important: The restore function will only restore any *kit* configuration settings that have been altered since manufacturing, by restoring a copy of the *kit* configuration settings stored internally on the debugger. This operation does not affect the MCU and will not undo any irreversible changes made to its configuration.

10. Tools and IDEs

The Nano Debugger is supported by multiple front-end tools published by Microchip, our partners and the open-source community.

10.1. Partner Ecosystems

The Nano Debugger is supported by various partner ecosystems and IDEs. The CMSIS-DAP standard debug implementation it is based on effectively provides support for any ARM Cortex device. The AVR protocol specification is public, and various IDEs that support AVR devices also support the Nano Debugger (This is the same protocol as used in JTAGICE3, Atmel-ICE, PowerDebugger, and EDBG.)

Partner integration includes, but is not limited to:

- IAR Embedded Workbench
- Keil Microvision
- CodeVisionAVR
- OpenOCD
- pyOCD
- AVRDUDE
- PlatformIO
- Bloom
- probe-rs

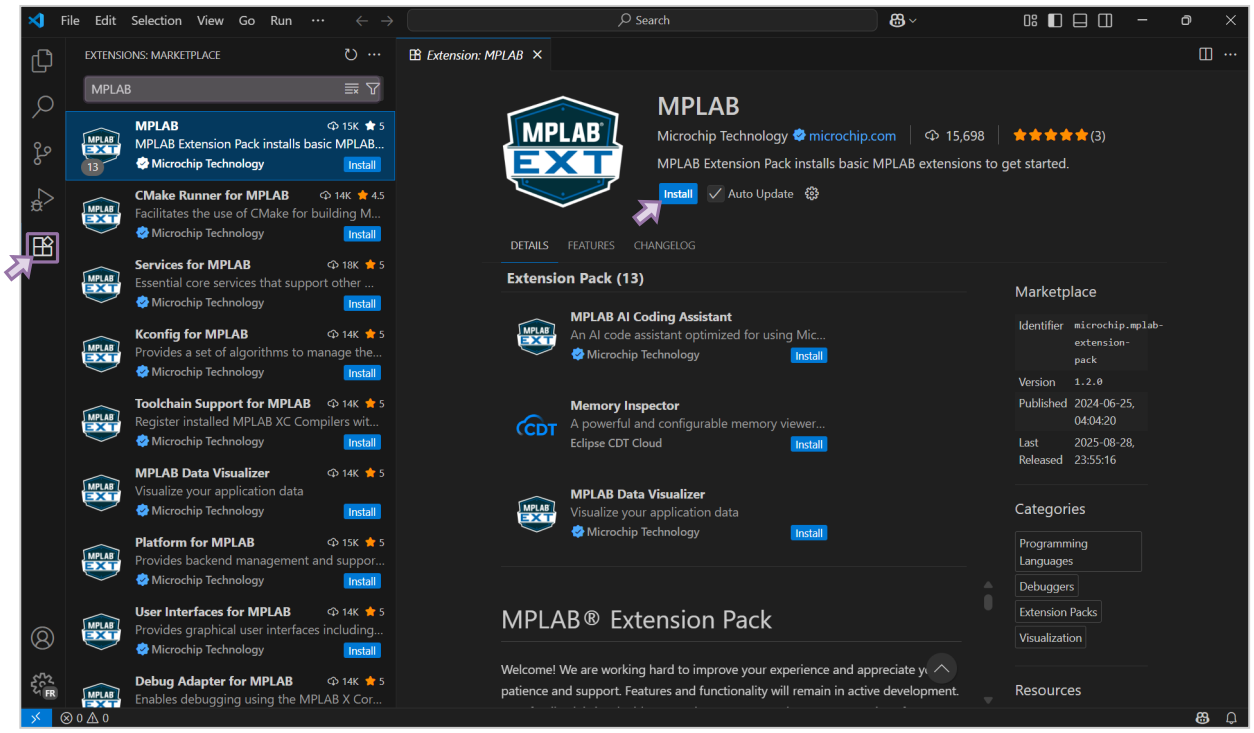
Tip: For up-to-date information about our partners, see Development Tool Partners.

10.2. MPLAB® Tools for VS Code

The Nano Debugger is supported in the VS Code development environment. As a standard CMSIS-DAP debugger, it supports generic debug adapters for ARM Cortex devices and frameworks built using these mechanisms. In addition, it is supported by MPLAB Tools for VS Code.

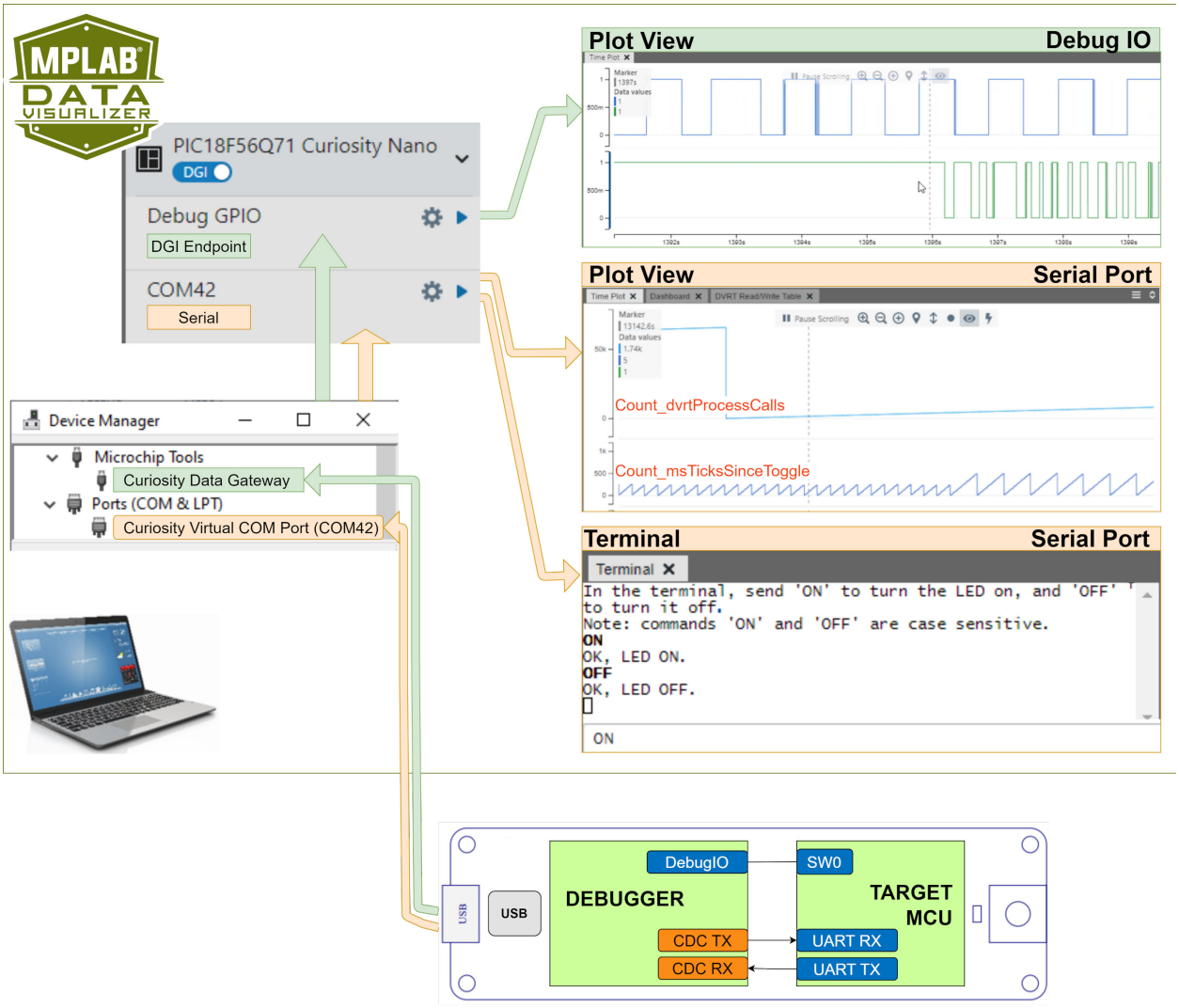
The MPLAB Tools for VS Code can be downloaded from the MPLAB Extensions official website or from the VS Code Marketplace. Each resource provides a complete list of MPLAB extensions that can be installed individually or together as part of the MPLAB Extensions Pack. The installation is performed through the VS Code Extensions. Tutorials are available on the official website with links to the Extensions Developer Help and the MPLAB Extensions for VS Code Video playlist.

Figure 10-1. MPLAB Extensions within the VS Code Extensions Marketplace



10.3. Using MPLAB Data Visualizer

The Nano Debugger, via its USB/serial bridge, facilitates a connection between a UART on the target MCU and your computer's COM port. For example, you can use this to connect to MPLAB Data Visualizer or other terminal programs.



Tip: References for the example Data Visualizer Plot- and Terminal-views above:

1. Plot View - DebugIO: DebugIO Hello World (Microchip University).
2. Plot View - Serial Port: MCC Melody Use Case Data Visualizer Run Time Use Case 1.
3. Terminal - Serial Port: MCC Melody UART Driver: LED Control Commands.

10.4. Kit Window View

Microchip IDEs have a mechanism for detecting Microchip tools connected via USB and presenting useful information in a Kit Window. This includes, for example, an image of the kit in question, ordering information for the kit and/or the device mounted on it, schematics, and links to example projects.

This view is rendered in each IDE with a different look and feel, but the data is deployed on a web server. Updates to the database are not synchronised with any IDE release cycle.

Some of the data presented in the Kit Window is read from the connected kit and not retrieved from the database. This data is stored in the Board Configuration of the Nano Debugger, and can thus be modified by kit developers (partners) as well as by end-users.

Note: The Kit Window is a mechanism developed by Microchip. A subset of this kit information is accessible using the standard CMSIS-DAP API commands on partner IDEs and tools.

For an example of the Kit Window view, see MPLAB X.

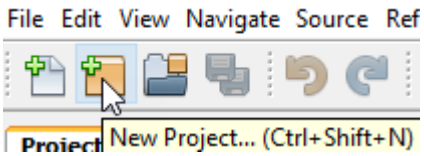
10.5. MPLAB X

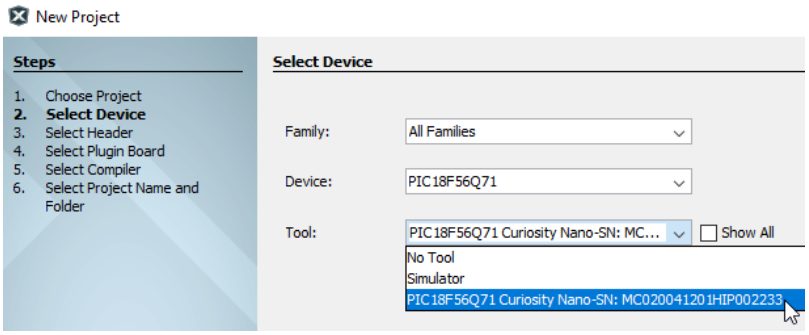
The Nano Debugger is supported in MPLAB X IDE and IPE.

Tip: Microchip has released MPLAB Tools for VS Code, which is a valuable update for MPLAB X users!

When a Nano Debugger is connected to a host PC via USB, if MPLAB X IDE is open, a Kit Window is opened with several key links presented.

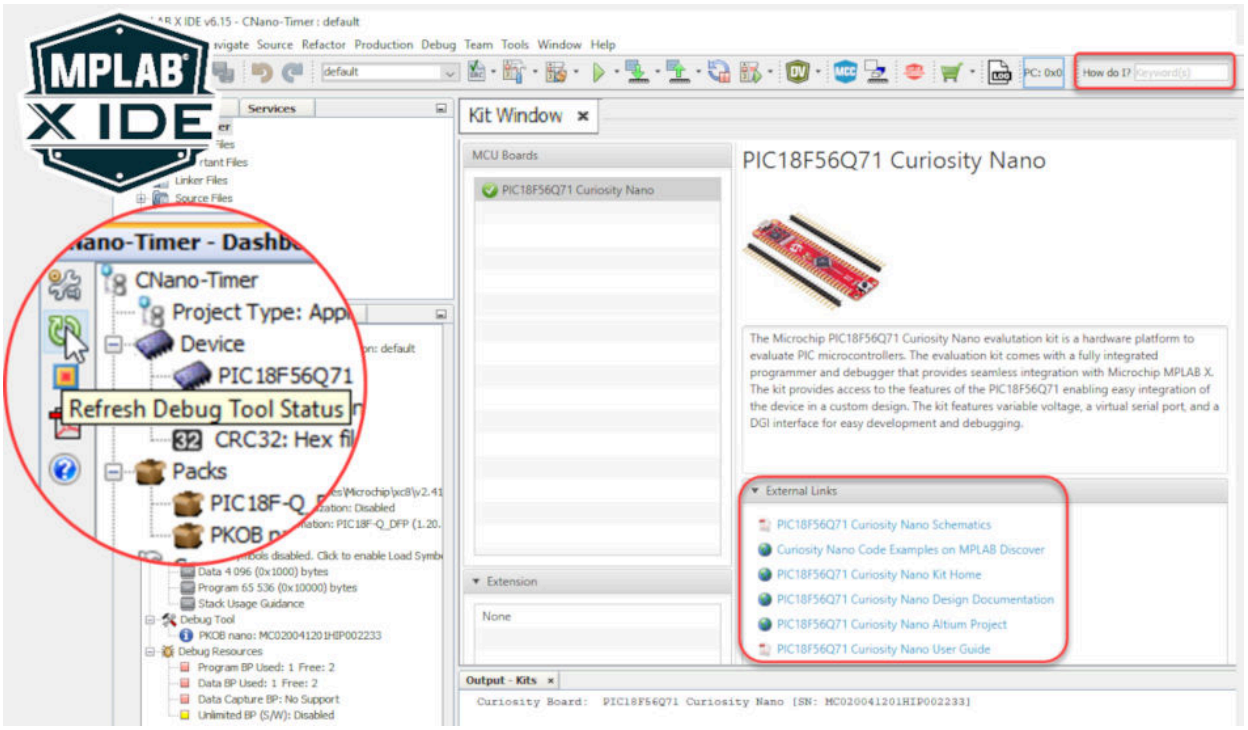
When creating a new project using a development board that includes a Nano Debugger (for example, a Curiosity Nano kit), the part number will be read from the connected board. Nano Debuggers in a standalone configuration will not have this information so the part number must be entered manually.





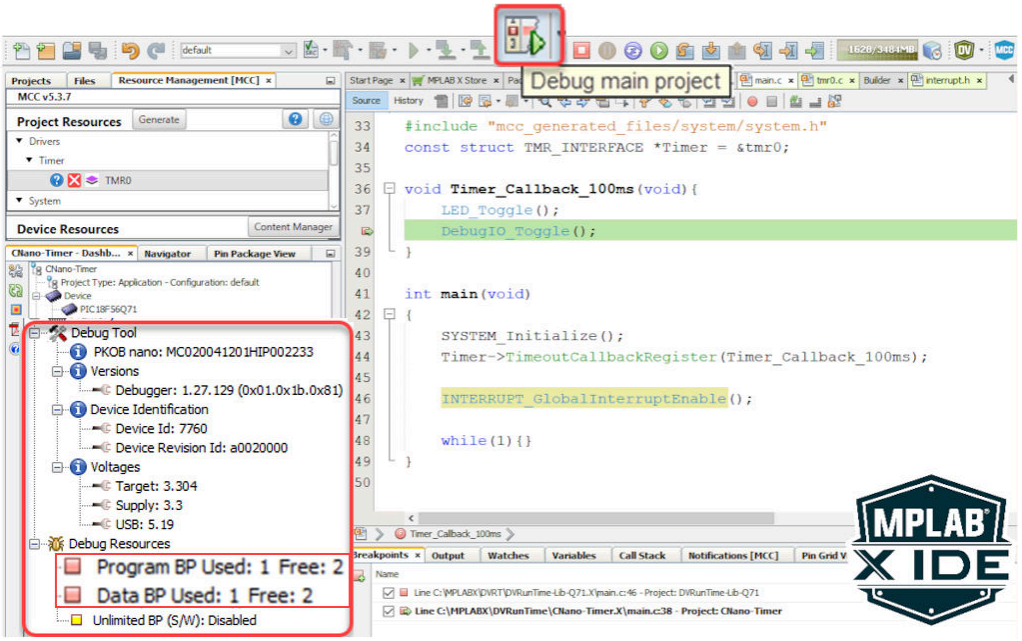
Note: Be sure to use the latest Tool Pack and Device Family Pack, especially when starting a new project.

As shown in the image below, additional information about your Curiosity Nano can be seen in the "Debug Tool" window once you click "Refresh Debug Tool Status".



Tip:

- If closed, you can reopen the Kit Window in MPLAB® X IDE through the menu bar: Window > Kit Window
- The 'How do I?' search bar often provides excellent results if you're new to MPLAB X IDE
- 'Debug main project" will start a debug session



Tip:

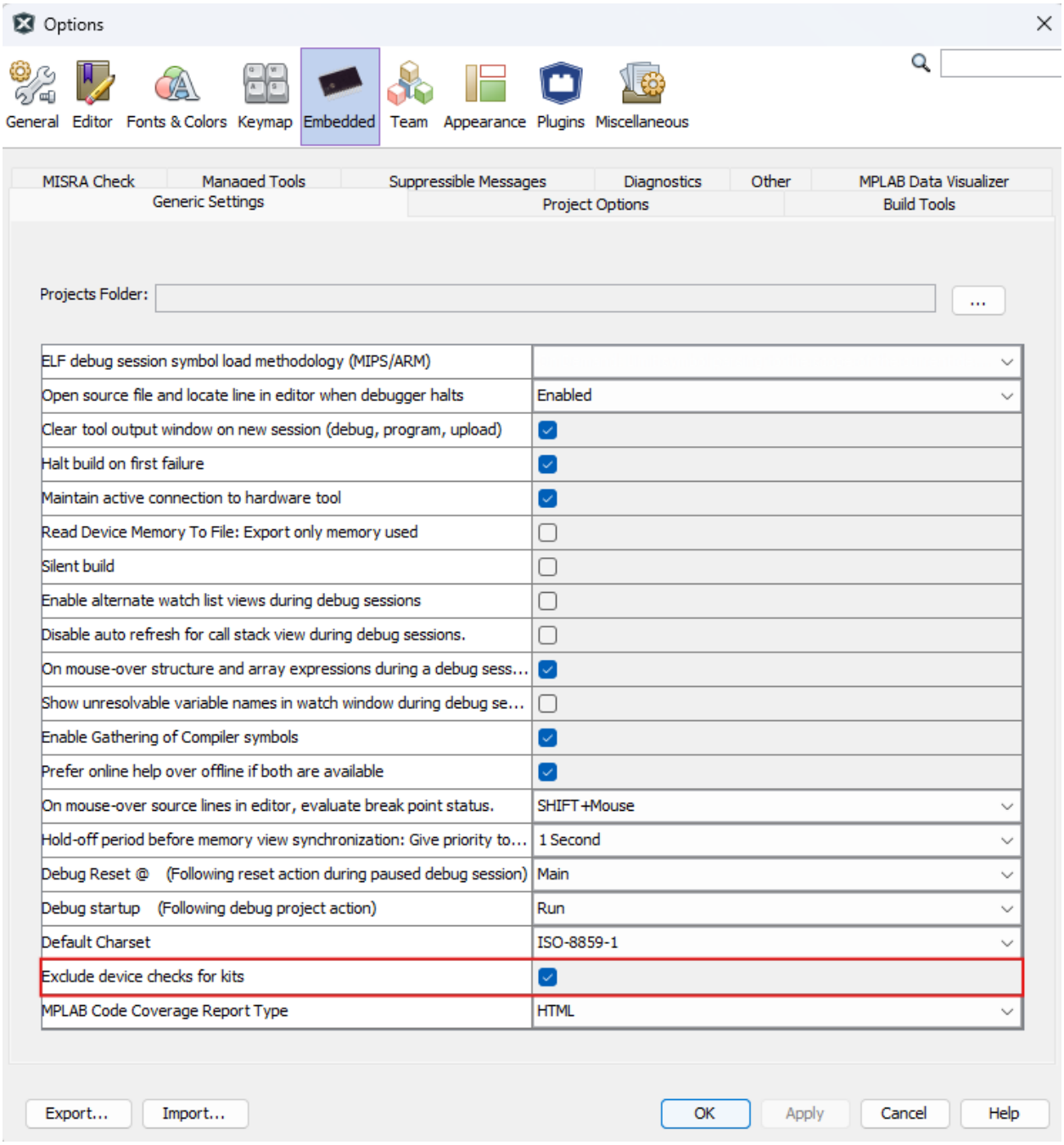
- After clicking on 'Refresh Debug Tool Status,' you can see information such as MCU Target Voltage
- What are Program- and Data-Break Points? MPLAB X IDE Advanced Debugging - Breakpoints Demo

Using a Nano Debugger kit with External Microcontrollers

A Nano Debugger on a development kit can easily be modified to operate with a microcontroller not originally mounted on that kit.
Note: See the User Guide for that particular kit when performing this procedure with a Curiosity Nano kit.

To program and debug a microcontroller other than the one mounted on the board, configure Microchip MPLAB X IDE to allow independent selection of devices and programming interfaces.

1. Navigate to Tools > Options through the menu at the top of the application.
2. Select the Embedded > Generic Settings category in the options window.
3. Check the **Exclude device checks for kits** option.



Note: Microchip MPLAB X IDE allows any microcontroller and interface to be selected when the **Exclude device checks for kits** setting is **checked** - also microcontrollers and interfaces not supported by the Nano Debugger.

10.6. Microchip Studio

The Nano Debugger is supported in Microchip Studio when used with devices that are supported by Microchip Studio and distributed through the Pack Server. This includes many AVR devices with a UPDI interface and some SAM devices.

Microchip Studio is "mature" and not recommended for new designs.

Important: The Nano Debugger USB interface may change in the future, which could render the firmware unusable in Microchip Studio.

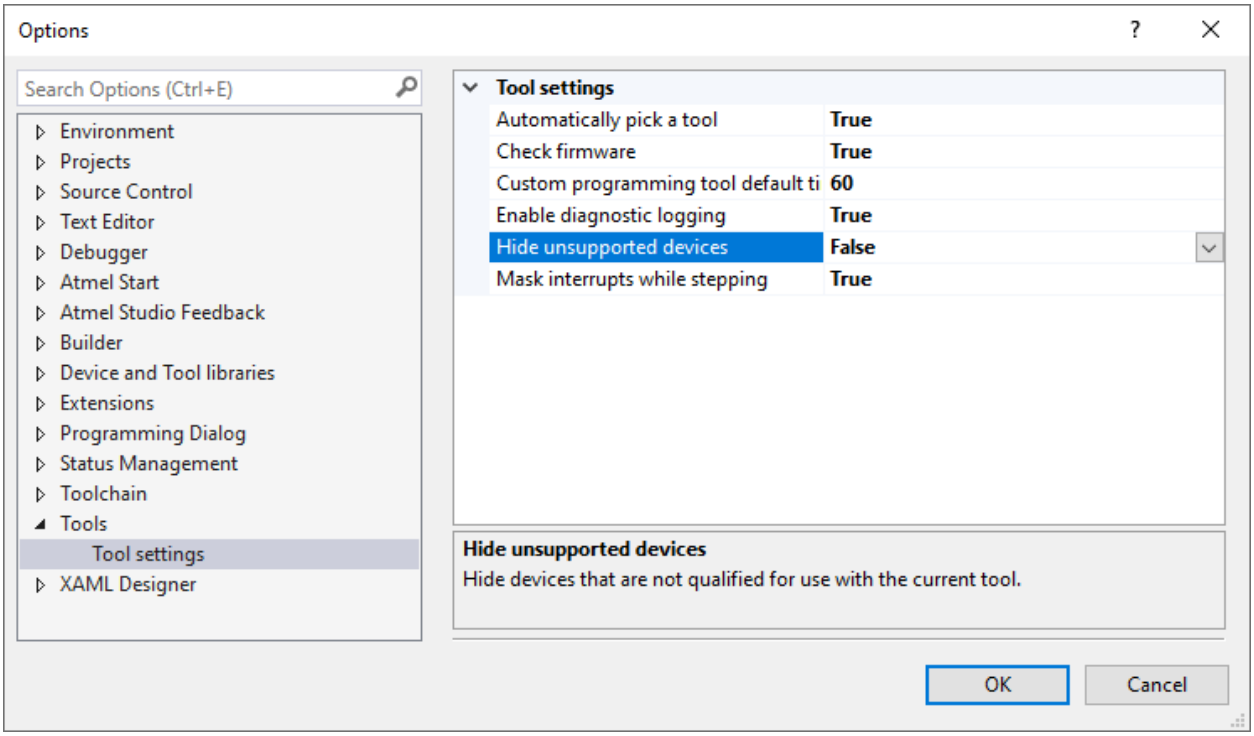
Tip: Microchip has released MPLAB Tools for VS Code, which is a valuable update for Microchip Studio users!

Using a Nano Debugger kit with External Microcontrollers

A Nano Debugger on a development kit can easily be modified to operate with a microcontroller not originally mounted on that kit.
Note: See the User Guide for that particular kit when performing this procedure with a Curiosity Nano kit.

To program and debug a different microcontroller than the one mounted on the board, configure Microchip Studio to allow an independent selection of devices and programming interfaces.

1. Navigate to Tools > Options through the menu at the top of the application.
2. Select the Tools > Tool settings category in the options window.
3. Set the **Hide unsupported devices** option to **False**.



Note: Microchip Studio allows any microcontroller and interface to be selected when the **Hide unsupported devices** setting is set to **False** - also microcontrollers and interfaces not supported by the on-board debugger.

10.7. Using Other Hardware Tools with a Nano Debugger Kit

When mounted as an on-board debugger on a development kit, the Nano Debugger allows for external tools to be connected. Such tools could offer additional features or better performance.

The Nano Debugger will not drive the debugger signal lines without instruction from an IDE.

Tip: For information on using an external debugger with a Nano Debugger kit read that kit's User Guide.

Important: When using a programmer capable of high-voltage programming, be sure to disconnect the Nano Debugger from the high-voltage line. High voltage can permanently damage the Nano Debugger.

10.8. USB Drivers

When a board with the Nano Debugger is connected to the computer for the first time, the operating system will install the driver software. The drivers for the Nano Debugger are included with MPLAB Tools for VS Code, MPLAB X IDE, and Microchip Studio.

11. Python Tools

A number of Python-based utilities are published on pypi.org that are useful when working with the Nano Debugger.

11.1. pydebuggerupgrade

pydebuggerupgrade is a CLI and library for upgrading the firmware on the Nano Debugger.

Why you Might use it

Use pydebuggerupgrade to keep your Nano Debugger firmware up to date.

Example

```
$ pydebuggerupgrade latest
Upgrading nedbg (ATML3203081800012252) to 'latest'
Upgrade to firmware version '1.33.76' successful
```

Table 11-1. In a Nutshell

Property	Value
Download link	pypi.org/project/pydebuggerupgrade/
Source code	-
Documentation	-
Command line interface	Yes
Library	Yes
Supported kit types	Nano Debugger based
Supported device types	All

11.2. pykitinfo

pykitinfo is a lightweight Python utility for reading out useful information regarding connected development kits and debuggers, including the Nano Debugger.

Why you Might use it

The pykitinfo CLI is especially useful when you have multiple Nano Debugger kits connected to a computer and need a quick lookup of which kit is mapped to which Virtual Serial Port.

Example

```
$ pykitinfo
Looking for Microchip kits...
Compatible kits detected: 1
Kit ATML3203081800012252: 'AVR-IoT WG' (ATmega4808) on COM163
```

Table 11-2. In a Nutshell

Property	Value
Download link	pypi.org/project/pykitinfo/
Source code	github.com/microchip-pic-avr-tools/pykitinfo
Documentation	microchip-pic-avr-tools.github.io/pykitinfo/
Command line interface	Yes
Library	Yes
Supported kit types	Various
Supported device types	All

11.3. pyedbglib

pyedbglib is a library that provides access to the protocols and sub-protocols embedded in the CMSIS-DAP interface of the Nano Debugger.

Why you Might use it

pyedbglib might be useful when building your own tools or utilities.

Example (Python library)

```
"""
Example usage of pyedbglib to read debugger firmware version and target voltage
"""
from pyedbglib.hidtransport.hidtransportfactory import hid_transport
from pyedbglib.protocols.housekeepingprotocol import Jtagice3HousekeepingProtocol
from pyedbglib import __version__ as pyedbglib_version

# Report library version
print("pyedbglib version {}".format(pyedbglib_version))

# Make a connection using HID transport
transport = hid_transport()
transport.connect()

# Create a housekeeper
housekeeper = Jtagice3HousekeepingProtocol(transport)
housekeeper.start_session()

# Read out debugger firmware version
major = housekeeper.get_byte(Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONTEXT_CONFIG,
                             Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONFIG_FWREV_MAJ)
minor = housekeeper.get_byte(Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONTEXT_CONFIG,
                             Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONFIG_FWREV_MIN)
build = housekeeper.get_le16(Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONTEXT_CONFIG,
                             Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONFIG_BUILD)
print ("Debugger firmware is version {}.{}.{}".format(major, minor,build))

# Read out target voltage
target_voltage =
housekeeper.get_le16(Jtagice3HousekeepingProtocol.HOUSEKEEPING_CONTEXT_ANALOG,
                                    Jtagice3HousekeepingProtocol.HOUSEKEEPING_ANALOG_VTREF)
print ("Target voltage is {:.02f}V".format(target_voltage/1000.0))

# Tear down
housekeeper.end_session()
transport.disconnect()
```

Table 11-3. In a Nutshell

Property	Value
Download link	pypi.org/project/pyedbglib/
Source code	github.com/microchip-pic-avr-tools/pyedbglib
Documentation	microchip-pic-avr-tools.github.io/pyedbglib/
Command line interface	No
Library	Yes
Supported kit types	Various
Supported device types	All

11.4. pymcuprog

pymcuprog is a lightweight Python utility for erasing, reading and writing non-volatile memories on devices connected to the Nano Debugger and other Microchip standalone debuggers.

Why you Might use it

The pymcuprog CLI is especially useful when you want to check connectivity to a device ("ping"), or perform simple read and write operations from the command line.

Example

```
$ pymcuprog ping
Connecting to anything possible
Connected to nEDBG CMSIS-DAP from Microchip (serial number ATML3203081800012252)
Debugger firmware version 1.21.37
Debugger hardware revision 0
Device mounted: 'atmega4808'
No device specified.  Using on-board target (atmega4808)
Pinging device...
Ping response: 1E9650
Done.
```

Table 11-4. In a Nutshell

Property	Value
Download link	pypi.org/project/pymcuprog/
Source code	github.com/microchip-pic-avr-tools/pymcuprog
Documentation	microchip-pic-avr-tools.github.io/pymcuprog/
Command line interface	Yes
Library	Yes
Supported kit types	• Curiosity Nano • Standalone CMSIS-DAP v1 debuggers
Supported device types	• AVR devices • PIC devices with Curiosity Nano boards • Limited support of select SAMD devices

11.5. pydebuggerconfig

pydebuggerconfig is a utility for configuring the Nano Debugger.

Why you Might use it

Use pydebuggerconfig when creating custom hardware with the Nano Debugger.

Example

```
$ pydebuggerconfig write -b board-configs/ATmega4809-CNANO.xml
```

Table 11-5. In a Nutshell

Property	Value
Download link	pypi.org/project/pydebuggerconfig/
Source code	-
Documentation	-
Command line interface	Yes
Library	Yes
Supported kit types	Nano Debugger
Supported device types	All

11.6. pycmsisdapswitcher

pycmsisdapswitcher is a utility for switching firmware on certain Microchip debuggers and kits.

Important: pycmsisdapswitcher does NOT work for the Nano Debugger - use pydebuggerupgrade

Why you Might use it

Use pycmsisdapswitcher to switch your PICkit™ 4, PICkit™ 5, PICkit™ Basic, Snap, or PKOB4 into CMSIS-DAP v2 mode.

Example

```
$ pycmsisdapswitcher  --appsource path_to_appfile
```

Table 11-6. In a Nutshell

Property	Value
Download link	pypi.org/project/pycmsisdapswitcher/
Source code	-
Documentation	-
Command line interface	Yes
Library	Yes
Supported kit types	PKOB4
Supported device types	All

11.7. pykitcommander

pykitcommander manages interactions with applications running on kits that use the Nano Debugger.

Why you Might use it

Use pykitcommander when you want to load an application onto an MCU and communicate with it using the Virtual Serial Port.

Table 11-7. In a Nutshell

Property	Value
Download link	pypi.org/project/pykitcommander/
Source code	github.com/microchip-pic-avr-tools/pykitcommander
Documentation	microchip-pic-avr-tools.github.io/pykitcommander/
Command line interface	No
Library	Yes
Supported kit types	Various
Supported device types	All

12. Pinout Reference

Table 12-1. Nano Debugger SAMD21 Pinout

32 QFN pin	Name	Description
1	CDC_TX	CDC UART transmit
2	CDC_RX	CDC UART receive
3	Power supply adjust	Analog output for power supply adjustment
4	VTG_ADC	Analog input for target voltage measurement
5	DBG0_RX	Receive signal for debug interface data
6	Reserved	Do not connect
7	DBG0_TX	Transmit signal for debug interface data
8	DBG1	Debug interface clock
9	VDDANA	Analog voltage supply for SAMD21
10	GNDANA	Analog ground for SAMD21
11	DBG2_CTRL	Level shifter direction for DBG2 (DGI GPIO)
12	DBG2	Data for DBG2 (DGI GPIO)
13	Power supply enable	Enable signal for power supply switch
14	VBUS_ADC	Analog input for USB VBUS measurement
15	DBG0_CTRL	Level shifter direction for DBG0 (debug data)
16	DBG1_CTRL	Level shifter direction for DBG1 (debug clock)
17	DBG3 $\overline{\text{RESET}}$ $\overline{\text{MCLR}}$	Open-drain control for DBG3 ($\overline{\text{RESET}}$,$\overline{\text{MCLR}}$)
18	VTG_EN	Enable switch for target voltage
19	CDC_RX_CTRL	Level shifter direction for CDC RX
20	VOFF	External voltage detection
21	CDC_TX_CTRL	Level shifter direction for CDC TX
22	ID_SYS	ID system
23	USB_DM	USB data
24	USB_DP	USB data
25	BOOT	Force bootloader entry
26	$\overline{\text{RESET}}$	Nano Debugger reset
27	LED	Status LED
28	GND	Ground
29	VDDCORE	SAMD21 core decoupling
30	VDDIN	SAMD21 VDD
31	SWCLK	SWCLK for debugging the Nano Debugger
32	SWDIO	SWDIO for debugging the Nano Debugger

13. Nano Debugger Firmware

13.1. Firmware Packs

The Nano Debugger firmware is not open source. Intel HEX files are included in the Tool Packs published by Microchip on the Microchip Packs Repository under **PKOB nano support**.

Inside each Tool Pack, you will find a *firmware* subfolder containing nedbg_fw.zip. The file *avrtools_fw.xml* provides information about the HEX file *nedbg.hex*.

Standalone Upgrade

Use pydebuggerupgrade to update the firmware on your Nano Debugger.

Tip: use the command

```
pydebuggerupgrade latest
```

to automatically get the latest released firmware

Tip: use the command

```
pydebuggerupgrade <x.y.z>
```

to automatically download and install the firmware in pack version x.y.z

IDE Upgrade

Use the Pack Manager in MPLAB X and MPLAB Tools for VS Code to download the latest Nano Debugger firmware pack. The firmware is updated automatically when the IDE connects to the kit.
Note: Be sure to select either a specific Tool Pack or "latest" in the project properties.

Legacy Upgraders

Microchip Studio only includes bundled firmware, which is no longer updated. Use a standalone upgrader or atfw.exe to update the firmware on the Nano Debugger.
Note: Microchip Studio may automatically downgrade your Nano Debugger to the latest bundled firmware. To prevent this behavior, delete or rename the firmware zip file inside the Microchip Studio Program Files folder.

Use the atfw.exe from Microchip Studio:

Tip: use the command

```
atfw.exe -a <zipfile>
```

to upgrade the firmware using a zip archive extracted from a Tool Pack

13.2. Revision History

Note: The revision is specified in decimal in this table, but some IDEs and tools report it in hexadecimal.

Table 13-1. Revision history of the Nano Debugger firmware

Revision	Tool Pack	Release date	Reference	Notable fixes and features
1.34	1.18.x	December 2025	FW5G-1580	• SWD timing improvements
1.33	1.17.969	September 2025	FW5G-1459	• dsPIC33A support and fixes • DAP_SWJ_Pins timing fix
1.32	1.16.876	March 2025	FW5G-1362	• AVR-SD UPDI handshake
1.31	1.14.751	April 2024	FW5G-1323	• added UF2 parser
1.30	1.13.715	October 2023	FW5G-1306	• hotfix for previous 1.29 hex file issue
1.29	1.12.711	October 2023	FW5G-1250	• fix for AVR NVM P:4 • added support for AVR BOOTROW • added PDID protection filter
1.27	1.11.554	December 2022	FW5G-1192	• added AVR NVM P:4 • added support for dsPIC33
1.25	1.10.488	May 2022	FW5G-1133	• added support for CDC BREAK
1.23	1.9.446	December 2021	FW5G-1086	• added AVR EA support • fix to AVR fuse protection
1.22	1.7.295	March 2021	FW5G-880	• multiple CDC fixes
1.21	1.5.210	September 2020	FW5G-744	• added MSC to CDC bridge
1.20	1.3.164	June 2020	FW5G-708	• SAM-IoT related fixes
1.17		March 2020	FW5G-620	• added SAM-IoT
1.16	1.0.33	March 2020	FW5G-516	Initial public tool pack release
pre 1.16	-	2018-2019	FW5G-485	Internal releases and bundles with Studio/MPLAB

Tip: The reference can be used when contacting Microchip support

14. Document revision history

Table 14-1. Document Revisions

Doc. Rev	Date	Comments
A	12/2025	Initial document release

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-2428-5

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Product Page Links

ATmega1608, ATmega1609, ATmega3208, ATmega3209, ATmega4808, ATmega4809, ATmega808, ATmega809, ATSAM4E16E, ATSAM4LC4C, ATSAM4LC8C, ATSAM4N16C, ATSAM4SD32C, ATSAMC20E15A, ATSAMC20E16A, ATSAMC20E17A, ATSAMC20E18A, ATSAMC20G15A, ATSAMC20G16A, ATSAMC20G17A, ATSAMC20G18A, ATSAMC20J15A, ATSAMC20J16A, ATSAMC20J17A, ATSAMC20J18A, ATSAMC20N17A, ATSAMC20N18A, ATSAMC21E15A, ATSAMC21E16A, ATSAMC21E17A, ATSAMC21E18A, ATSAMC21G15A, ATSAMC21G16A, ATSAMC21G17A, ATSAMC21G18A, ATSAMC21J15A, ATSAMC21J16A, ATSAMC21J17A, ATSAMC21J18A, ATSAMC21N17A, ATSAMC21N18A, ATSAMCA1E17A, ATSAMD20E14, ATSAMD20E15, ATSAMD20E16, ATSAMD20E17, ATSAMD20E18, ATSAMD20G14, ATSAMD20G15, ATSAMD20G16, ATSAMD20G17, ATSAMD20G18, ATSAMD20J14, ATSAMD20J15, ATSAMD20J16, ATSAMD20J17, ATSAMD20J18, ATSAMD21E15, ATSAMD21E15L, ATSAMD21E16, ATSAMD21E16L, ATSAMD21E17, ATSAMD21E17L, ATSAMD21E18, ATSAMD21G15, ATSAMD21G16, ATSAMD21G16L, ATSAMD21G17, ATSAMD21G17L, ATSAMD21G18, ATSAMD21J15, ATSAMD21J16, ATSAMD21J17, ATSAMD21J18, ATSAMD51G18A, ATSAMD51G19A, ATSAMD51J18A, ATSAMD51J19A, ATSAMD51J20A, ATSAMD51N19A, ATSAMD51N20A, ATSAMD51P19A, ATSAMD51P20A, ATSAMDA1E14B, ATSAMDA1E15B, ATSAMDA1E16B, ATSAMDA1G14B, ATSAMDA1G15B, ATSAMDA1G16B, ATSAMDA1J14B, ATSAMDA1J15B, ATSAMDA1J16B, ATSAME51G18A, ATSAME51G19A, ATSAME51J18A, ATSAME51J19A, ATSAME51J20A, ATSAME51N19A, ATSAME51N20A, ATSAME53J18A, ATSAME53J19A, ATSAME53J20A, ATSAME53N19A, ATSAME53N20A, ATSAME54N19A, ATSAME54N20A, ATSAME54P19A, ATSAME54P20A, ATSAME70J19, ATSAME70J20, ATSAME70J21, ATSAME70N19, ATSAME70N20, ATSAME70N21, ATSAME70Q19, ATSAME70Q20, ATSAME70Q21, ATSAMG53, ATSAMHA0E14A-B, ATSAMHA0E15A-B, ATSAMHA0E16A-B, ATSAMHA0G14A-B, ATSAMHA0G15A-B, ATSAMHA0G16A-B, ATSAMHA0G17A-B, ATSAMHA1E14A-B, ATSAMHA1E15A-B, ATSAMHA1E16A-B, ATSAMHA1G14A, ATSAMHA1G14A-B, ATSAMHA1G15A, ATSAMHA1G15A-B, ATSAMHA1G16A, ATSAMHA1G16A-B, ATSAMHA1G17A-B, ATSAML10D14A, ATSAML10D15A, ATSAML10D16A, ATSAML10E14A, ATSAML10E15A, ATSAML10E16A, ATSAML11D14A, ATSAML11D15A, ATSAML11D16A, ATSAML11E14A, ATSAML11E15A, ATSAML11E16A, ATSAML21E15B, ATSAML21E16B, ATSAML21E17B, ATSAML21E18B, ATSAML21G16B, ATSAML21G17B, ATSAML21G18B, ATSAML21J16B, ATSAML21J17B, ATSAML21J18B, ATSAML22G16A, ATSAML22G17A, ATSAML22G18A, ATSAML22J16A, ATSAML22J17A, ATSAML22J18A, ATSAML22N16A, ATSAML22N17A, ATSAML22N18A, ATSAMS70J19, ATSAMS70J20, ATSAMS70J21, ATSAMS70N19, ATSAMS70N20, ATSAMS70N21, ATSAMS70Q19, ATSAMS70Q20, ATSAMS70Q21, ATSAMV70J19, ATSAMV70J20, ATSAMV70N19, ATSAMV70N20, ATSAMV70Q19, ATSAMV70Q20, ATSAMV71J19, ATSAMV71J20, ATSAMV71J21, ATSAMV71N19, ATSAMV71N20, ATSAMV71N21, ATSAMV71Q19, ATSAMV71Q20, ATSAMV71Q21, ATtiny1604, ATtiny1606, ATtiny1607, ATtiny1614, ATtiny1616, ATtiny1617, ATtiny1624, ATtiny1626, ATtiny1627, ATtiny202, ATtiny204, ATtiny212, ATtiny214, ATtiny3216, ATtiny3217, ATtiny3224, ATtiny3226, ATtiny3227, ATtiny402, ATtiny404, ATtiny406, ATtiny412, ATtiny414, ATtiny416, ATtiny417, ATtiny424, ATtiny426, ATtiny427, ATtiny804, ATtiny806, ATtiny807, ATtiny814, ATtiny816, ATtiny817, ATtiny824, ATtiny826, ATtiny827, ATtiny828, ATtiny861, AV32DU20, AV32DU28, AVR128DA28, AVR128DA28S, AVR128DA32, AVR128DA32S, AVR128DA48, AVR128DA48S, AVR128DA64, AVR128DA64S, AVR128DB28, AVR128DB32, AVR128DB48, AVR128DB64, AVR16DD14, AVR16DD20, AVR16DD28, AVR16DD32, AVR16DU14, AVR16DU20, AVR16DU28, AVR16DU32, AVR16EA28, AVR16EA32, AVR16EA48, AVR16EB14, AVR16EB20, AVR16EB28, AVR16EB32, AVR16LA14, AVR16LA20, AVR16LA28, AVR16LA32, AVR32DA28, AVR32DA28S, AVR32DA32, AVR32DA32S, AVR32DA48, AVR32DA48S, AVR32DB28, AVR32DB32, AVR32DB48, AVR32DD14, AVR32DD20, AVR32DD28, AVR32DD32, AVR32DU14, AVR32DU20, AVR32DU28, AVR32DU32, AVR32EA28, AVR32EA32, AVR32EA48, AVR32EB14, AVR32EB20, AVR32EB28, AVR32EB32, AVR32EC28, AVR32EC32, AVR32EC48, AVR32LA14, AVR32LA20, AVR32LA28, AVR32LA32, AVR32SD20, AVR32SD28, AVR32SD32, AVR64DA28, AVR64DA28S, AVR64DA32, AVR64DA32S, AVR64DA48, AVR64DA48S, AVR64DA64, AVR64DA64S, AVR64DB28, AVR64DB32, AVR64DB48, AVR64DB64, AVR64DD14, AVR64DD20, AVR64DD28, AVR64DD32, AVR64DU28, AVR64DU32, AVR64EA28, AVR64EA32, AVR64EA48, AVR64EC28, AVR64EC32,

AVR64EC48, AVR64SD28, AVR64SD32, AVR64SD48, PIC16F13145, PIC16F13276, PIC16F15244, PIC16F15276, PIC16F17146, PIC16F18076, PIC16F18446, PIC18F16Q20, PIC18F16Q40, PIC18F16Q41, PIC18F47K42, PIC18F47Q10, PIC18F56Q24, PIC18F56Q35, PIC18F56Q71, PIC18F57Q43, PIC18F57Q84, PIC32CK0512GC00064, PIC32CK0512GC00100, PIC32CK0512GC01064, PIC32CK0512GC01100, PIC32CK0512SG00064, PIC32CK0512SG00100, PIC32CK0512SG01064, PIC32CK0512SG01100, PIC32CK1025GC00064, PIC32CK1025GC00100, PIC32CK1025GC01064, PIC32CK1025GC01100, PIC32CK1025GC01144, PIC32CK1025SG00064, PIC32CK1025SG00100, PIC32CK1025SG01064, PIC32CK1025SG01100, PIC32CK1025SG01144, PIC32CK2051GC00064, PIC32CK2051GC00100, PIC32CK2051GC00144, PIC32CK2051GC01064, PIC32CK2051GC01100, PIC32CK2051GC01144, PIC32CK2051SG00064, PIC32CK2051SG00100, PIC32CK2051SG00144, PIC32CK2051SG01064, PIC32CK2051SG01100, PIC32CK2051SG01144, PIC32CM1216JH01032, PIC32CM1216MC00032, PIC32CM1216MC00048, PIC32CM1216PL10028, PIC32CM1216PL10032, PIC32CM1216PL10048, PIC32CM1216PL10064, PIC32CM1602GV00032, PIC32CM1602GV00048, PIC32CM1602GV00064, PIC32CM1602PL10020, PIC32CM1602PL10028, PIC32CM1602PL10032, PIC32CM2532JH00032, PIC32CM2532JH00048, PIC32CM2532JH00064, PIC32CM2532JH00100, PIC32CM2532JH01032, PIC32CM2532JH01048, PIC32CM2532JH01064, PIC32CM2532JH01100, PIC32CM2532LE00048, PIC32CM2532LE00064, PIC32CM2532LE00100, PIC32CM2532LS00048, PIC32CM2532LS00064, PIC32CM2532LS00100, PIC32CM2532LS60048, PIC32CM2532LS60064, PIC32CM2532LS60100, PIC32CM3204GV00032, PIC32CM3204GV00048, PIC32CM3204GV00064, PIC32CM3204JH00032, PIC32CM3204JH00048, PIC32CM3204JH00064, PIC32CM3204PL10020, PIC32CM3204PL10028, PIC32CM3204PL10032, PIC32CM5112GC00048, PIC32CM5112GC00064, PIC32CM5112GC00100, PIC32CM5112SG00048, PIC32CM5112SG00064, PIC32CM5112SG00100, PIC32CM5164JH00032, PIC32CM5164JH00048, PIC32CM5164JH00064, PIC32CM5164JH00100, PIC32CM5164JH01032, PIC32CM5164JH01048, PIC32CM5164JH01064, PIC32CM5164JH01100, PIC32CM5164LE00048, PIC32CM5164LE00064, PIC32CM5164LE00100, PIC32CM5164LS00048, PIC32CM5164LS00064, PIC32CM5164LS00100, PIC32CM5164LS60048, PIC32CM5164LS60064, PIC32CM5164LS60100, PIC32CM6408JH00032, PIC32CM6408JH00048, PIC32CM6408JH00064, PIC32CM6408MC00032, PIC32CM6408MC00048, PIC32CM6408PL10028, PIC32CM6408PL10032, PIC32CM6408PL10048, PIC32CM6408PL10064, PIC32CX1012BZ24032, PIC32CX1012BZ25048, PIC32CX1025MTC, PIC32CX1025MTG, PIC32CX1025MTSH, PIC32CX1025SG41100, PIC32CX1025SG41128, PIC32CX1025SG60100, PIC32CX1025SG60128, PIC32CX1025SG61100, PIC32CX1025SG61128, PIC32CX2051BZ62132, PIC32CX2051MTC, PIC32CX2051MTC128, PIC32CX2051MTG, PIC32CX2051MTSH, PIC32CX5109BZ31032, PIC32CX5109BZ31048, PIC32CX5109BZ36032, PIC32CX5109BZ36048, PIC32CX5112MTC, PIC32CX5112MTG, PIC32CX5112MTSH, PIC32CX-BZ2, PIC32CX-BZ3, PIC32CX-BZ6, PIC32CZ2051CA70064, PIC32CZ2051CA70100, PIC32CZ2051CA70144, PIC32CZ2051CA80100, PIC32CZ2051CA80144, PIC32CZ2051CA80176, PIC32CZ2051CA80208, PIC32CZ2051CA90100, PIC32CZ2051CA90144, PIC32CZ2051CA90176, PIC32CZ2051CA90208, PIC32CZ2051MC70064, PIC32CZ2051MC70100, PIC32CZ2051MC70144, PIC32CZ4010CA80100, PIC32CZ4010CA80144, PIC32CZ4010CA80176, PIC32CZ4010CA80208, PIC32CZ4010CA90100, PIC32CZ4010CA90144, PIC32CZ4010CA90176, PIC32CZ8110CA80100, PIC32CZ8110CA80144, PIC32CZ8110CA80176, PIC32CZ8110CA80208, PIC32CZ8110CA90100, PIC32CZ8110CA90144, PIC32CZ8110CA90176, PIC32CZ8110CA90208