

Introduction

This document introduces AVR® LA microcontrollers (MCUs) and demonstrates how to develop applications using the hardware and software tools supported by Microchip. It helps readers get started with the AVR LA MCU family by covering device selection, toolchain configuration, and providing a step-by-step guide for creating example applications using the Curiosity Nano platform and various software development tools.

For development, this guide outlines the essential tools and resources required to establish a development environment using MPLAB® tools for Visual Studio Code® (VS Code®) or the MPLAB X Integrated Development Environment (IDE). It also indicates where to find code examples that use AVR LA MCUs. The AVR32LA32 microcontroller is used as a reference for this demonstration.

Refer to the device-specific data sheet for additional information about the differences between the devices within their respective families.

Table of Contents

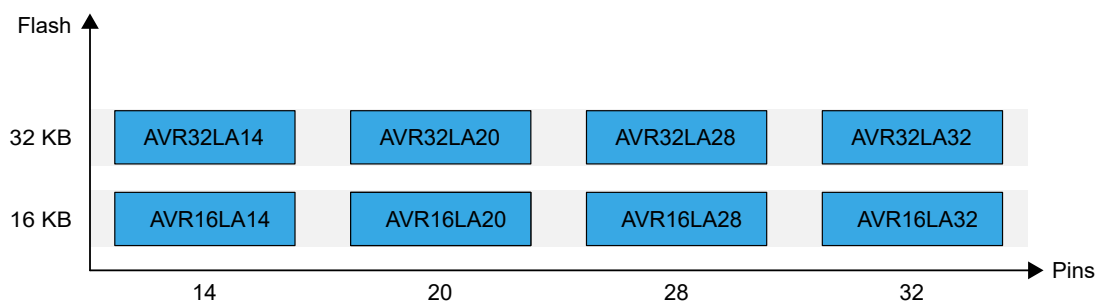
Introduction	1
1. Relevant Devices	3
2. Get the Device Data Sheet	4
3. Relevant Documents.....	5
4. Tools and Software.....	6
4.1. Get the Evaluation Kit.....	6
4.2. MPLAB® Tools for VS Code®	6
4.3. Get Device Support for VS Code®	7
4.4. Get MPLAB® X IDE	7
4.5. Get Device Support for MPLAB® X IDE.....	7
4.6. Configuring Compilers	7
4.7. Get Device Support for MPLAB® Code Configurator (MCC).....	7
4.8. Get Code Examples From MPLAB® Discover.....	7
5. Getting Started With VS Code®	9
5.1. Create the First Application on the AVR® LA MCU.....	9
5.2. Configure the Application Using MCC Melody	15
5.3. Generate the Code.....	21
5.4. Add the Application Logic	22
5.5. Build and Program the Application.....	23
5.6. Observe the Output	25
6. Getting Started With MPLAB® X IDE	28
6.1. Create the First Application on the AVR® LA MCU.....	28
6.2. Configure the Application Using MCC Melody	32
6.3. Generate the Code.....	38
6.4. Add the Application Logic	38
6.5. Build and Program the Application.....	40
6.6. Observe the Output	40
7. What's Next.....	42
7.1. Software.....	42
7.2. Firmware.....	42
7.3. Evaluation Kits.....	42
7.4. Recommended Programming/Debugging Tools.....	43
8. References.....	47
9. Revision History.....	48
Microchip Information.....	49
Trademarks.....	49
Legal Notice.....	49
Microchip Devices Code Protection Feature.....	49
Product Page Links.....	50

1. Relevant Devices

This section provides a list of devices relevant for this demonstration. The figure below illustrates various devices within this family, detailing their pin-count variants and memory sizes:

- Vertical migration upward is possible without code modification, as these devices are pin-compatible and provide the same features
- Horizontal migration to the left reduces the pin count, which in turn limits the features available

Figure 1-1. Relevant Devices



2. Get the Device Data Sheet

Product Pages

- AVR32LA32: www.microchip.com/wwwproducts/en/AVR32LA32
- AVR32LA28: www.microchip.com/wwwproducts/en/AVR32LA28
- AVR32LA20: www.microchip.com/wwwproducts/en/AVR32LA20
- AVR32LA14: www.microchip.com/wwwproducts/en/AVR32LA14
- AVR16LA32: www.microchip.com/wwwproducts/en/AVR16LA32
- AVR16LA28: www.microchip.com/wwwproducts/en/AVR16LA28
- AVR16LA20: www.microchip.com/wwwproducts/en/AVR16LA20
- AVR16LA14: www.microchip.com/wwwproducts/en/AVR16LA14

Documents

- [AVR® LA Product Brief](#)
- [AVR32LA14/20/28/32 Data Sheet](#)
- [AVR32LA14/20/28/32 Silicon Errata and Data Sheet Clarifications](#)
- [AVR16LA14/20/28/32 Data Sheet](#)
- [AVR16LA14/20/28/32 Silicon Errata and Data Sheet Clarifications](#)

The documentation for the AVR family is divided into two document types:

- Data sheet (includes device descriptions, the number of peripherals, pinout information, and electrical characteristics)
- Errata (includes known issues and errata for the device)

3. Relevant Documents

All relevant documents can be found under the documentation tab on the product page.

Below is a list of documents relevant to the AVR[®] LA family of microcontrollers.

Table 3-1. Application Notes and Technical Briefs

Servicing Peripheral Interrupts While Writing Flash
USART in One-Wire Mode
Interfacing AVR[®] Microcontrollers With Serial Memories
ADC Oversampling With tinyAVR[®] 0- and 1-series, and megaAVR[®] 0-series

Table 3-2. Code Examples

Curiosity Nano Out-Of-Box
Getting Started With the AVR32LA32 Microcontroller Using the Curiosity Nano and MCC Melody
Low Power Touch Using AVR32LA32 Microcontroller With QT7 Xpro

4. Tools and Software

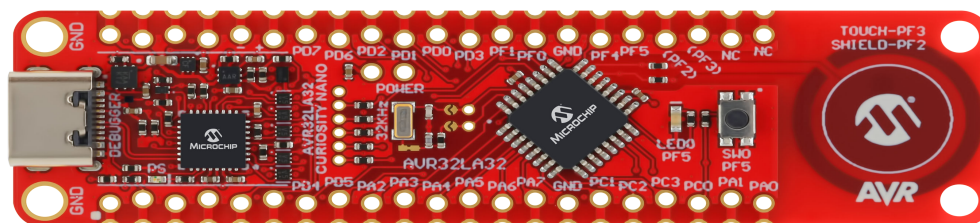
VS Code® and MPLAB® X IDE, with the appropriate extensions, can be used as development environments for working with the AVR® LA family, both utilizing the XC8 compiler.

4.1. Get the Evaluation Kit

4.1.1. Curiosity Nano Evaluation Kit

The Curiosity Nano is a compact hardware platform for rapid prototyping and hands-on evaluation of Microchip microcontrollers. It features an on-board debugger and breaks out the MCU pins to an edge connector, enabling easy access for testing and development.

Figure 4-1. AVR32LA32 Curiosity Nano Evaluation Kit



For additional information, refer to the [AVR32LA32 Curiosity Nano User Guide](#).

Note: The silkscreen designator for the on-board SW0 is incorrect. The correct designator for SW0 is PC0. Refer to the *Hardware Revision History and Known Issues* section of the [AVR32LA32 Curiosity Nano User Guide](#).

AVR32LA32 Curiosity Nano Key Features

- AVR32LA32 Microcontroller
- One Yellow User LED
- One Mechanical Switch
- One Capacitive Touch Button
- 32.768 kHz Crystal
- USB Powered
- On-Board Debugger
 - Board identification in MPLAB® X IDE and VS Code®
 - One green power and status LED
 - Programming and debugging
 - Virtual COM port (CDC)
 - One debug GPIO channel (DGI GPIO)

4.2. MPLAB® Tools for VS Code®

The MPLAB Tools for VS Code are available for download from the official [MPLAB Tools](#) website or directly through the [VS Code Marketplace](#). Both platforms offer a complete list of MPLAB extensions, which can be installed individually or collectively as part of the MPLAB Extensions Pack. Installation is performed through the VS Code Extensions interface. A step-by-step guide for getting started with the MCC plugin in VS Code is available on the official website, including links to the [Extensions Developer Help](#) and the [MPLAB Tools for VS Code video playlist](#).

MPLAB® Plug-ins

To enhance the functionality of MPLAB, users are encouraged to install the following additional plug-ins:

- Clangd for MPLAB: Provides support for Clang-based compilation and advanced code analysis
- MPLAB Code Configurator (MCC) for Microchip devices: Enables graphical configuration of peripherals and code generation for supported devices
- Toolchain Support for MPLAB: Integrates the XC8, XC16 and XC32 compilers for optimized code generation for Microchip devices

4.3. Get Device Support for VS Code®

The MPLAB® extension handles pack installation for AVR® LA devices, so users do not need to install any packs explicitly.

4.4. Get MPLAB® X IDE

MPLAB X can be used as an IDE for developing and debugging firmware for the AVR® LA family. For device support, refer to the [Get Device Support for MPLAB X IDE](#) section.

Web page: [MPLAB X IDE](#)

4.5. Get Device Support for MPLAB® X IDE

Use the MPLAB Pack Manager, found under *Tools > Packs*, to add support for new devices in MPLAB X.

For the AVR® LA family, update to the latest version by following these steps.

1. Click **Check for Updates**.
2. Select the latest available version of the AVR LA series (Microchip AVR Lx Series Device Support).
3. Click **Install**.

For offline installers, go to packs.download.microchip.com. Double-click the installer file and follow the instructions to install the package. Any open MPLAB X window must be closed for the installation to take effect.

4.6. Configuring Compilers

For AVR® LA devices, which are 8-bit microcontrollers, use the XC8 compiler.

Web page: [XC8 Compilers](#)

4.7. Get Device Support for MPLAB® Code Configurator (MCC)

MCC Melody

MCC Melody can be utilized within MPLAB X IDE as a graphical programming interface that generates peripheral and library code for the AVR® LA family. For a step-by-step guide to setting up the necessary tools, including MPLAB X IDE, the XC8 Compiler, and MCC, refer to this video: [How to Set up the Tools Required to Get Started with MCC](#).

Web page: [MPLAB Code Configurator](#)

4.8. Get Code Examples From MPLAB® Discover

MPLAB Discover bundles available resources from multiple sources into a single portal, including code examples, data sheets, evaluation boards, videos, and blogs.

Web page: [MPLAB Discover](#)

Figure 4-2. MPLAB® Discover**Code Examples**

Code examples for devices in the AVR® LA family can be found by searching for the device name (for example, AVR32LA32) in the search bar.

Users can download code examples as `.zip` files directly from MPLAB Discover. Repositories ending with "mplab-mcc" can be opened in MPLAB X.

When a code example is hosted on GitHub, MPLAB Discover provides an "Open with GitHub" link. Download the example code from there or use the Git tool on the PC to create a local clone of the repository.

5. Getting Started With VS Code®

Prerequisites

- Install the following software tools for this demonstration:
 - VS Code
 - VS Code Plug-ins:
 - MPLAB (v1.2.5 or later)
 - Toolchain Support for MPLAB (XC8 v3.10 or later)
 - MPLAB® Code Configurator (MCC) for Microchip devices (v5.7.0 or later)
- Hardware tool used for this demonstration:
 - AVR32LA32 Curiosity Nano evaluation kit (EV24D63A)
- Connect the AVR32LA32 Curiosity Nano board to the PC via the on-board USB connector, which is connected to the embedded debugger. The kit is powered through the USB connection, and the embedded debugger enables debugging and programming via USB.

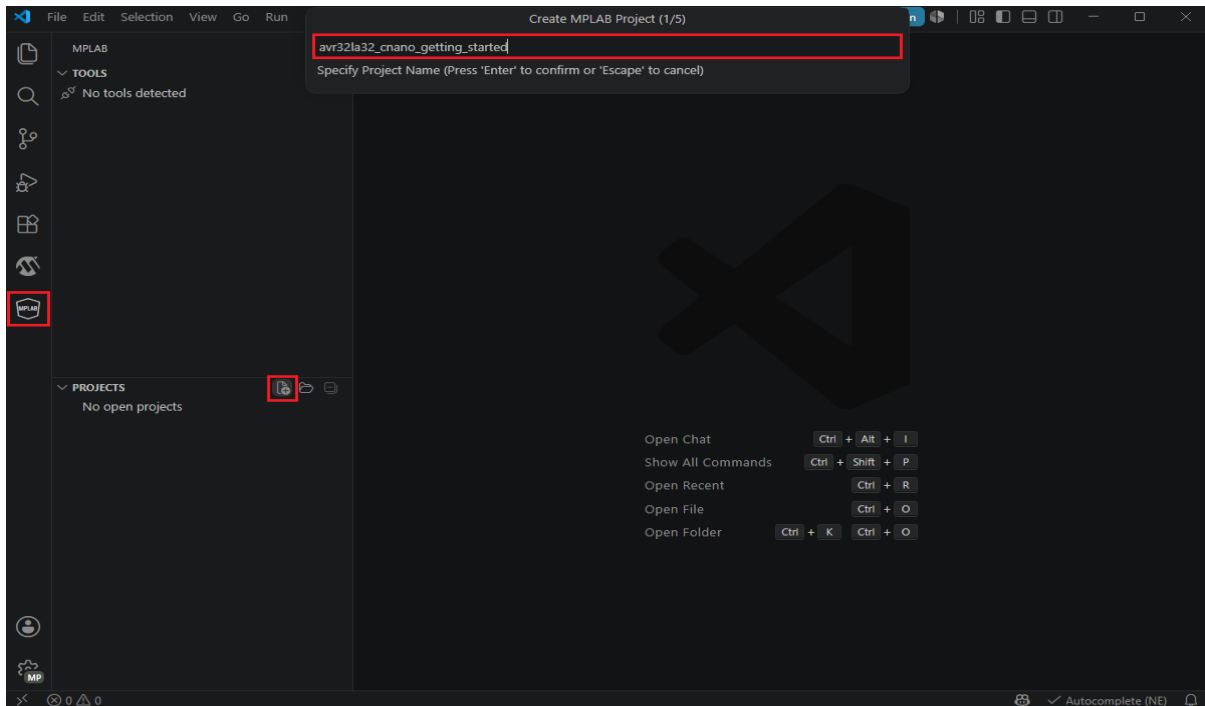
Note: If not already installed, VS Code can be downloaded from the official Visual Studio Code website and installed by following the standard setup procedure. Once installed, the MPLAB Tools for VS Code can be downloaded from the official MPLAB Tools website or directly from the VS Code Marketplace. Both resources offer a complete list of MPLAB extensions, which can be installed individually or collectively as part of the MPLAB Extensions Pack. Installation is performed through the VS Code Extensions interface. Additionally, tutorials are available on the official website, including links to the Extensions Developer Help and the MPLAB Tools for VS Code video playlist. Refer to the [Tools and Software](#) section for instructions on setting up the basic VS Code environment.

5.1. Create the First Application on the AVR® LA MCU

To create a VS Code®-based project, follow the steps below:

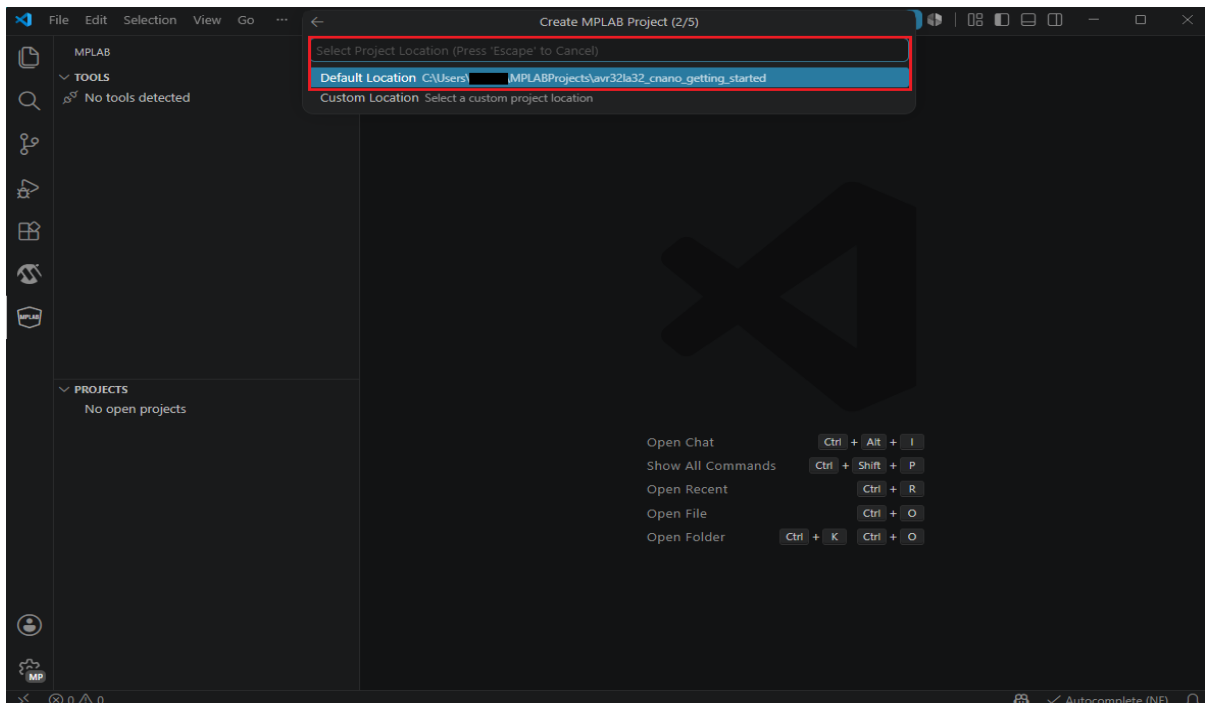
1. Launch VS Code.
2. On the left navigation bar, click the **MPLAB®** plug-in.
3. Under the Projects panel on the left, click New Project to create a new MPLAB project.
4. Enter the project title in the search window and press <Enter>.

Figure 5-1. Create a New Project in MPLAB® VS Code®



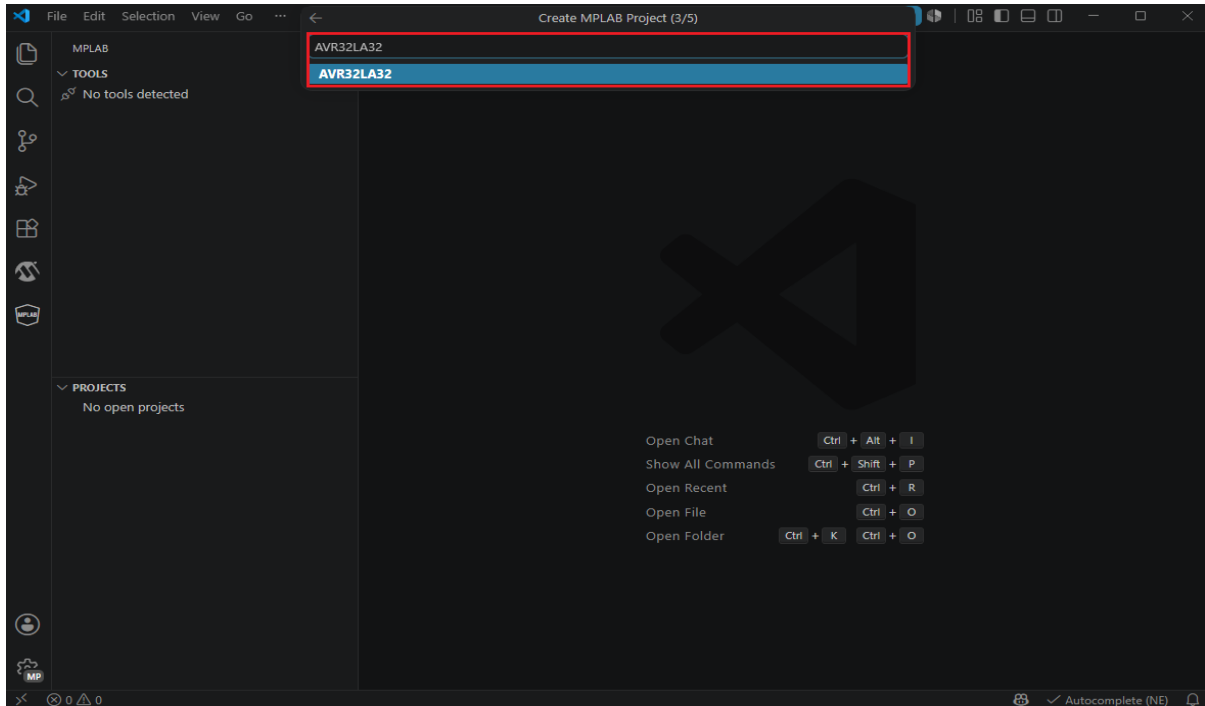
5. Select the location to use for creating the new project. Press <Enter> to proceed. In this example, the default location is selected.

Figure 5-2. Select the Project Location



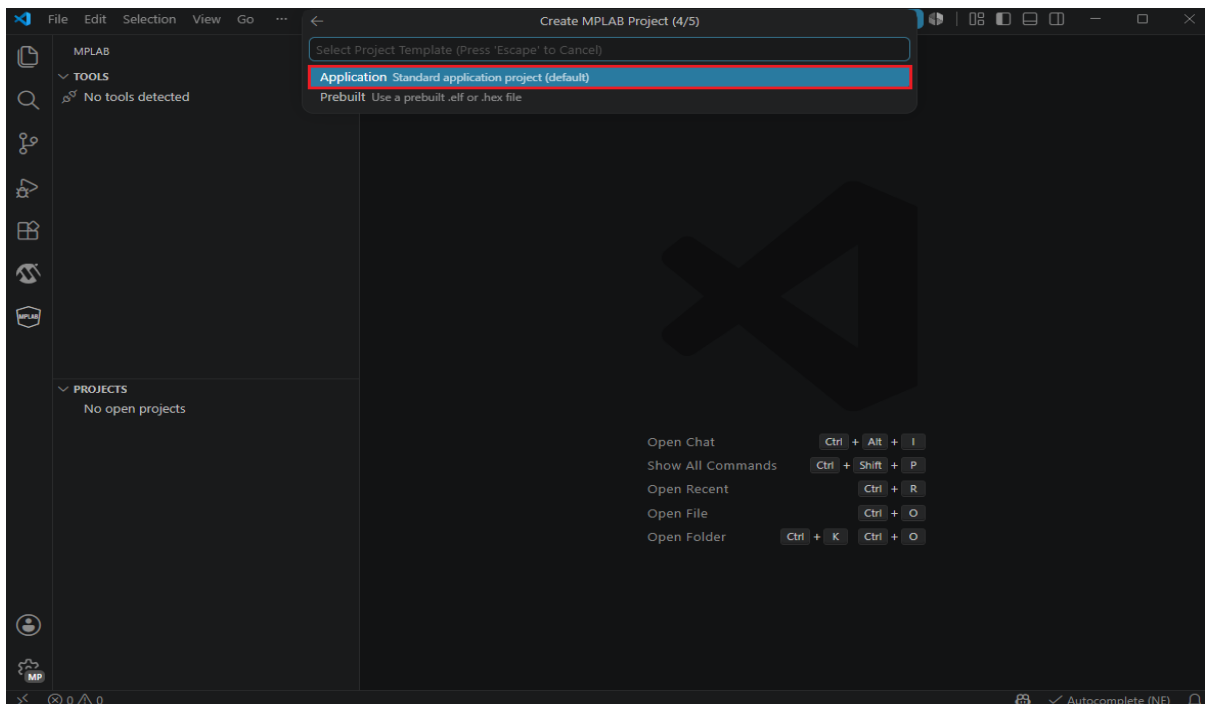
6. Select AVR32LA32 as the device and press <Enter>.

Figure 5-3. Select the Device



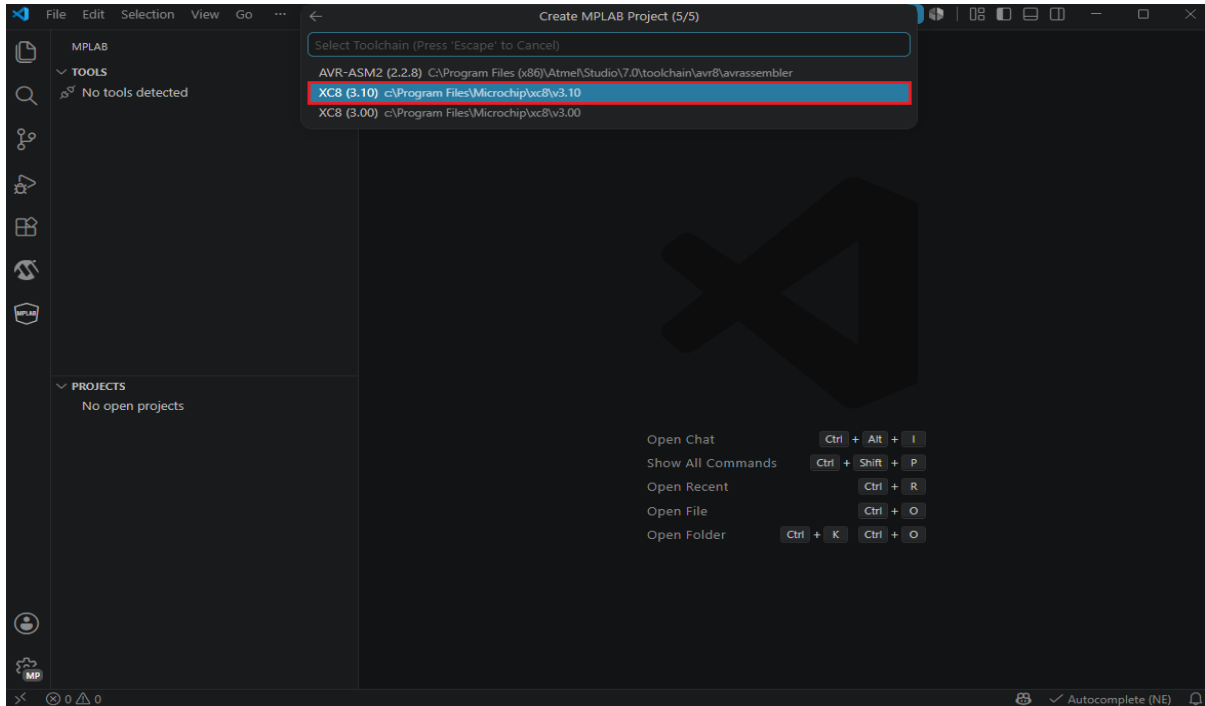
7. Select the project template as Application and press <Enter>.

Figure 5-4. Select the Project Template



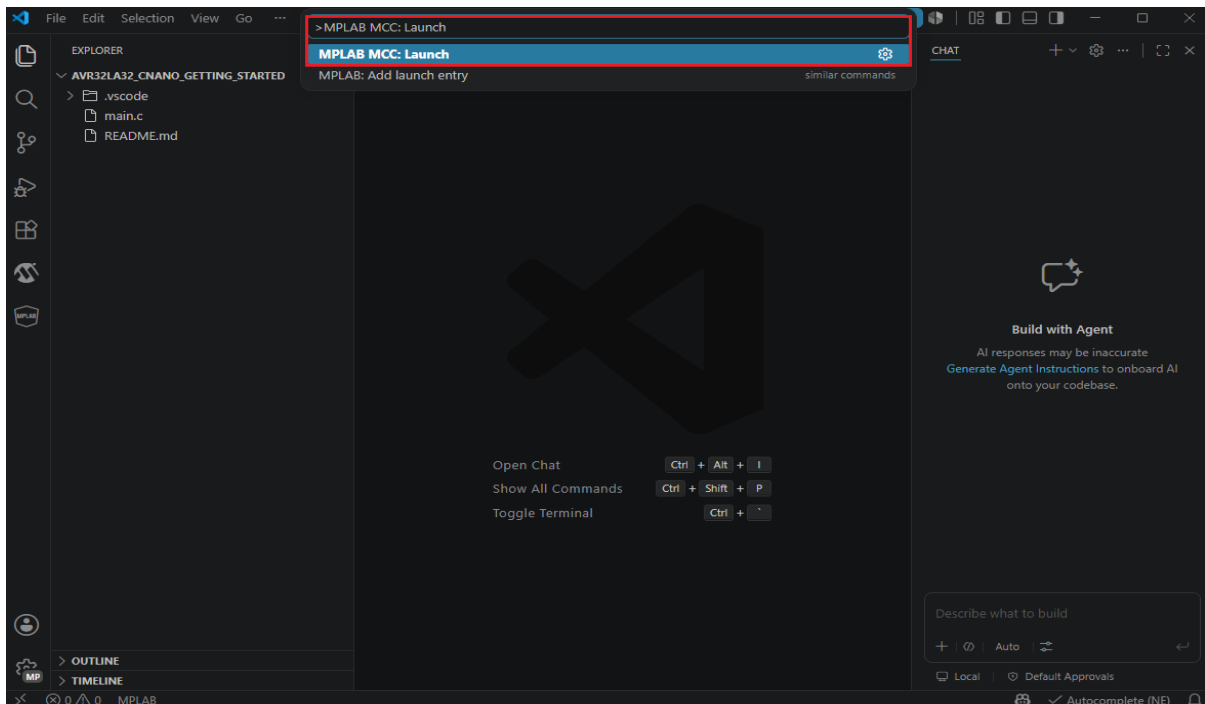
8. Select the latest compiler version (XC8 v3.10) and press <Enter>.

Figure 5-5. Select the Compiler



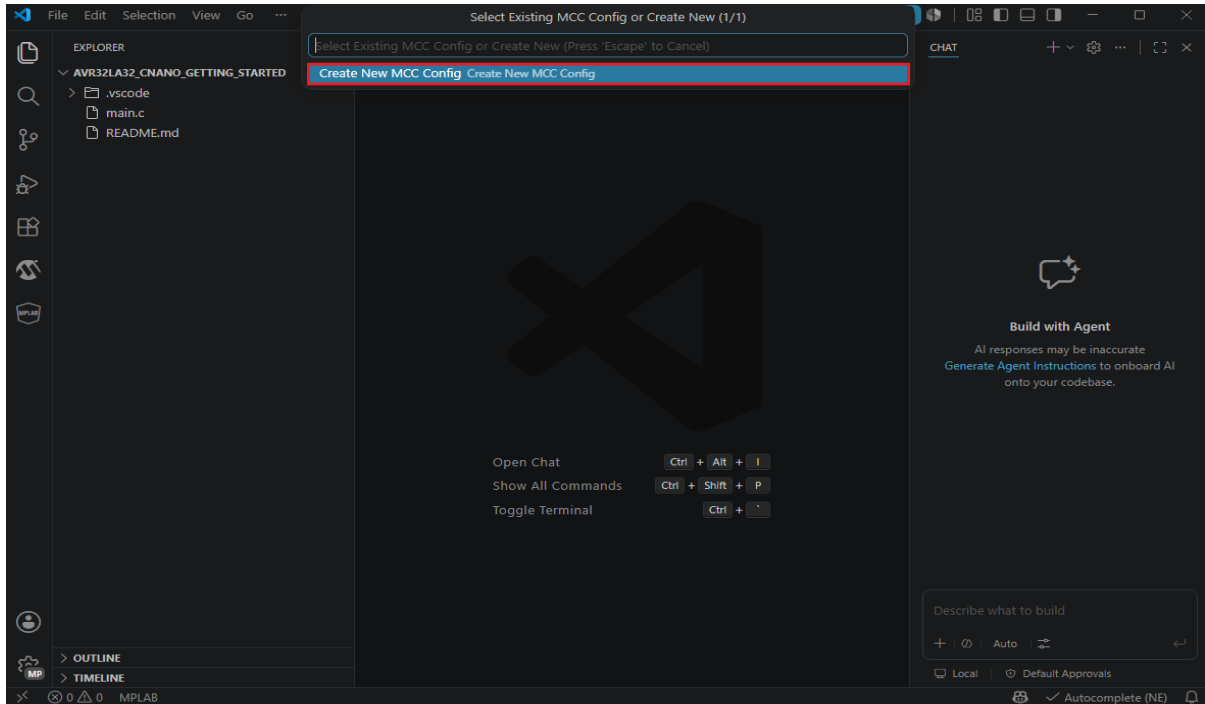
9. To launch the MCC window, type `>MPLAB MCC: Launch` in the Command Palette.

Figure 5-6. Launch the MCC Window



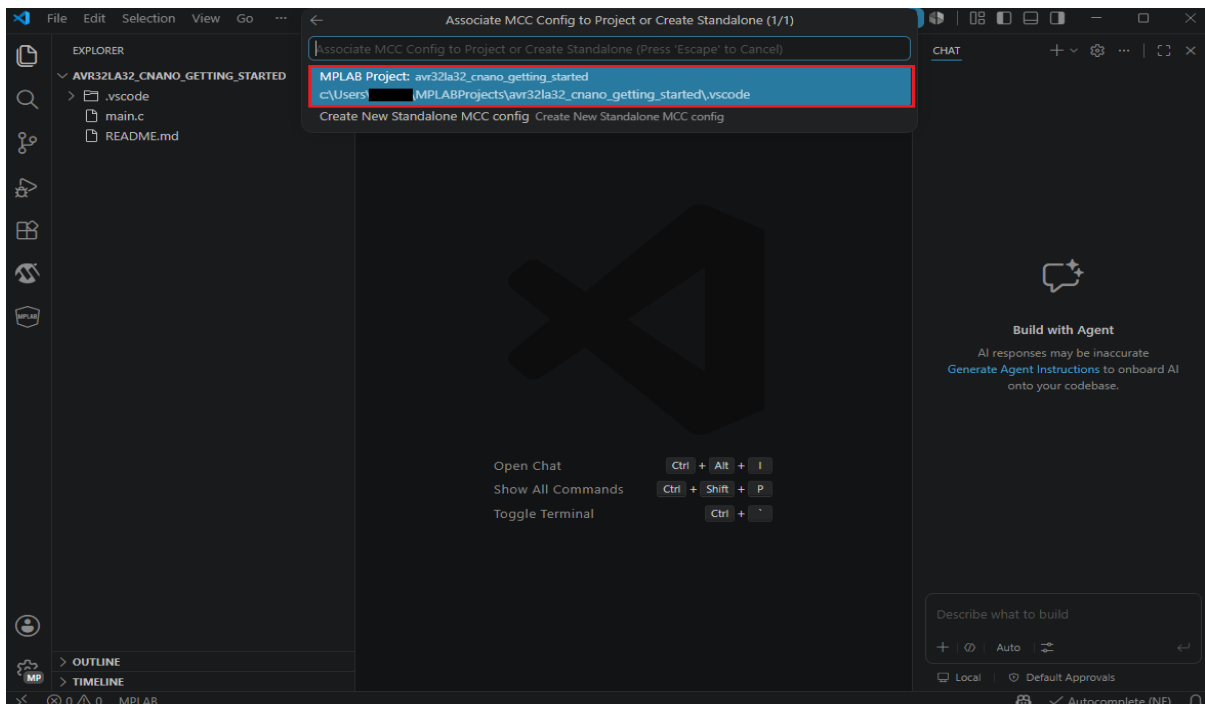
10. Select the Create New MCC Config option to create a new MCC configuration for the project.

Figure 5-7. New MCC Config



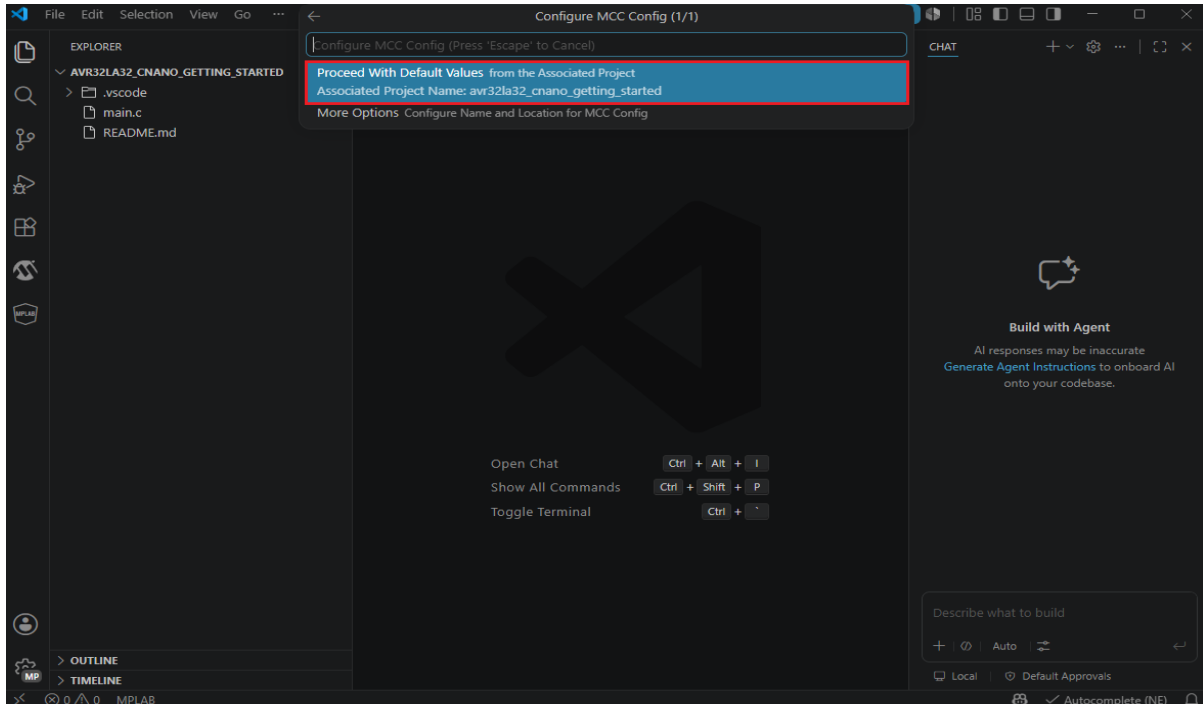
11. Associate the newly created MPLAB project with the new MCC configuration.

Figure 5-8. Associate the MPLAB® Project With the MCC Configuration



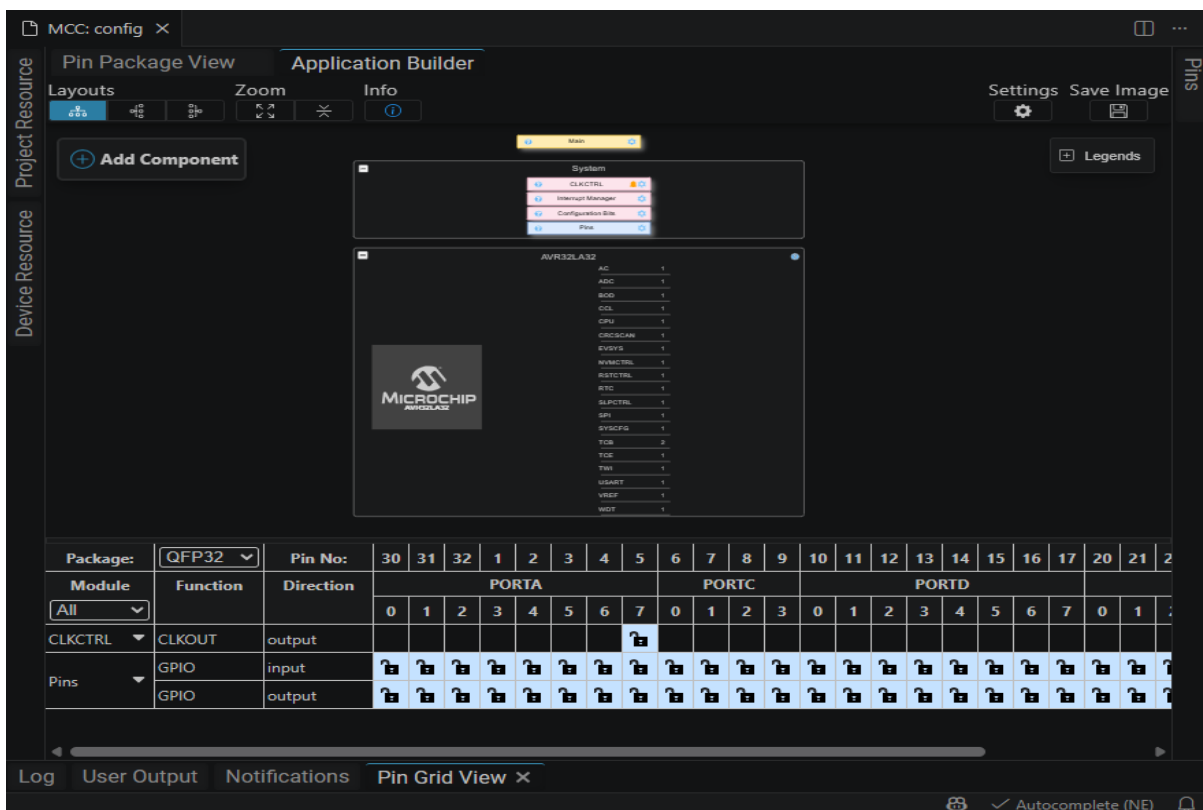
12. Configure the MCC settings to "Proceed With Default Values".

Figure 5-9. MCC Configuration With Default Values



13. The MCC plug-in will launch a new window, as shown in the following figure.

Figure 5-10. MPLAB® Code Configurator Window

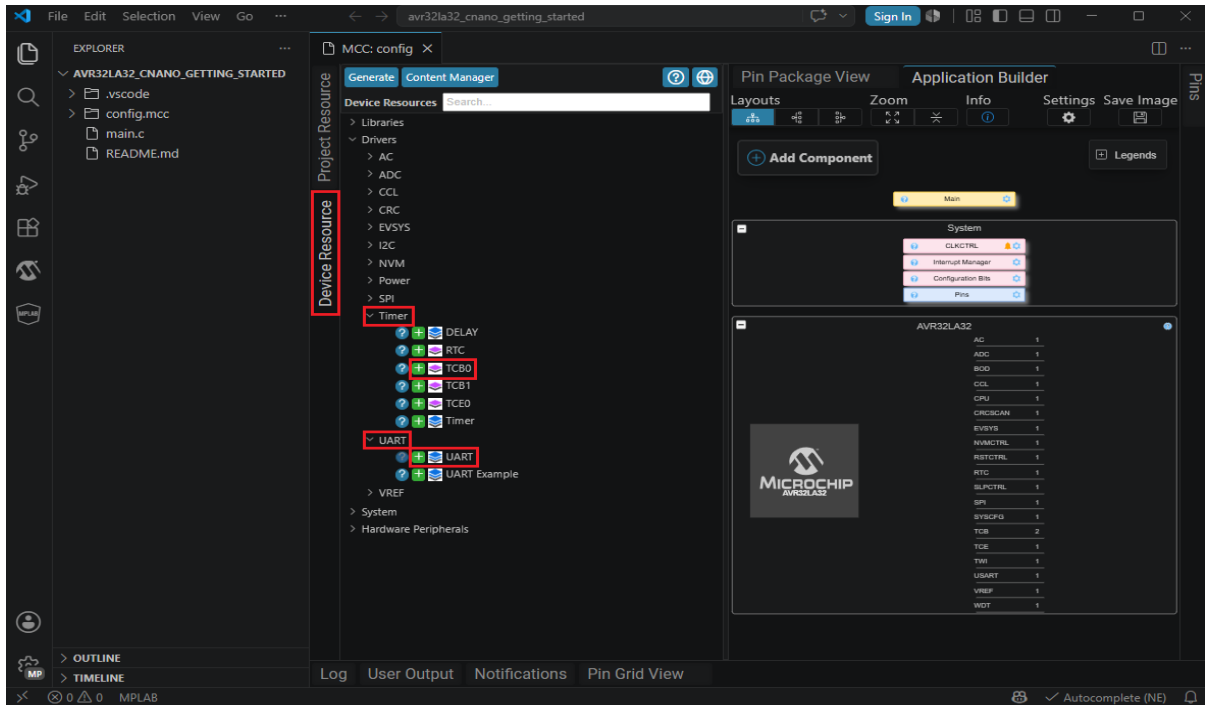


5.2. Configure the Application Using MCC Melody

Follow the steps below to add and configure peripherals using MCC Melody:

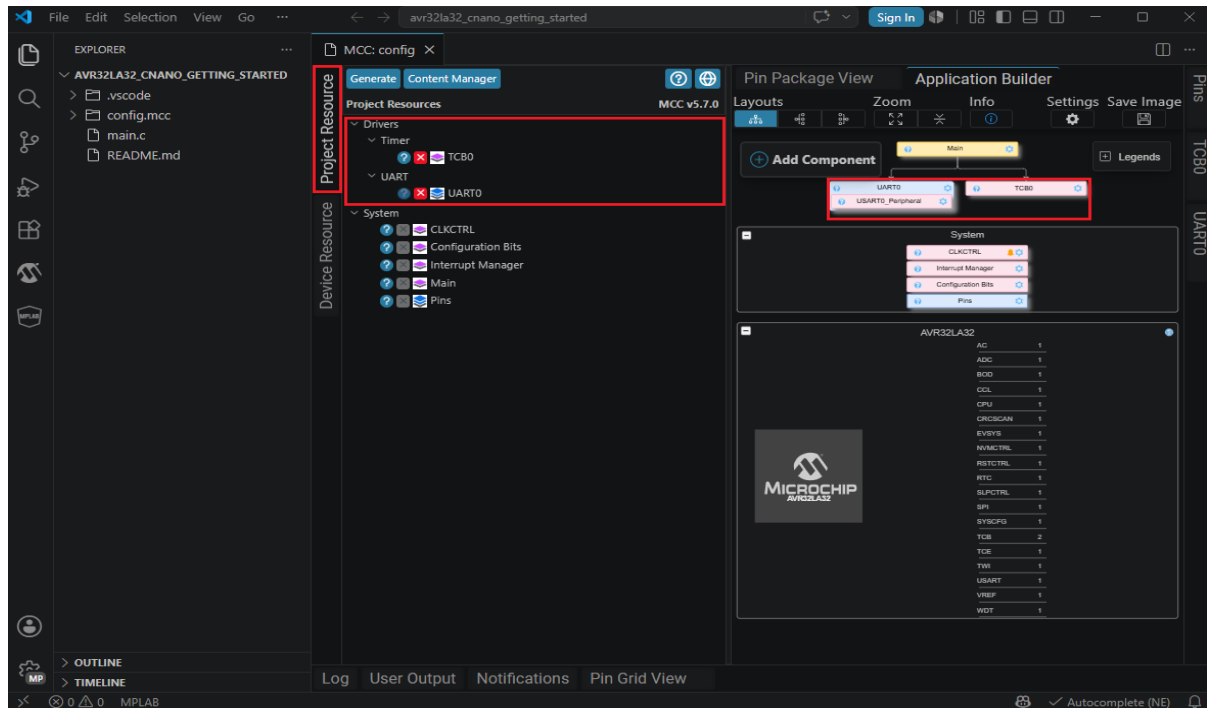
1. Click the **Device Resources** tab to view the drivers and libraries available for the device. Expand the Drivers drop-down and configure the following:
 - Expand the Timer drop-down and add TCB0
 - Expand the UART drop-down and add UART

Figure 5-11. MPLAB® Code Configurator—Device Resources



2. The peripherals added to the project will be visible in the **Project Resources** tab and the Application Builder view.

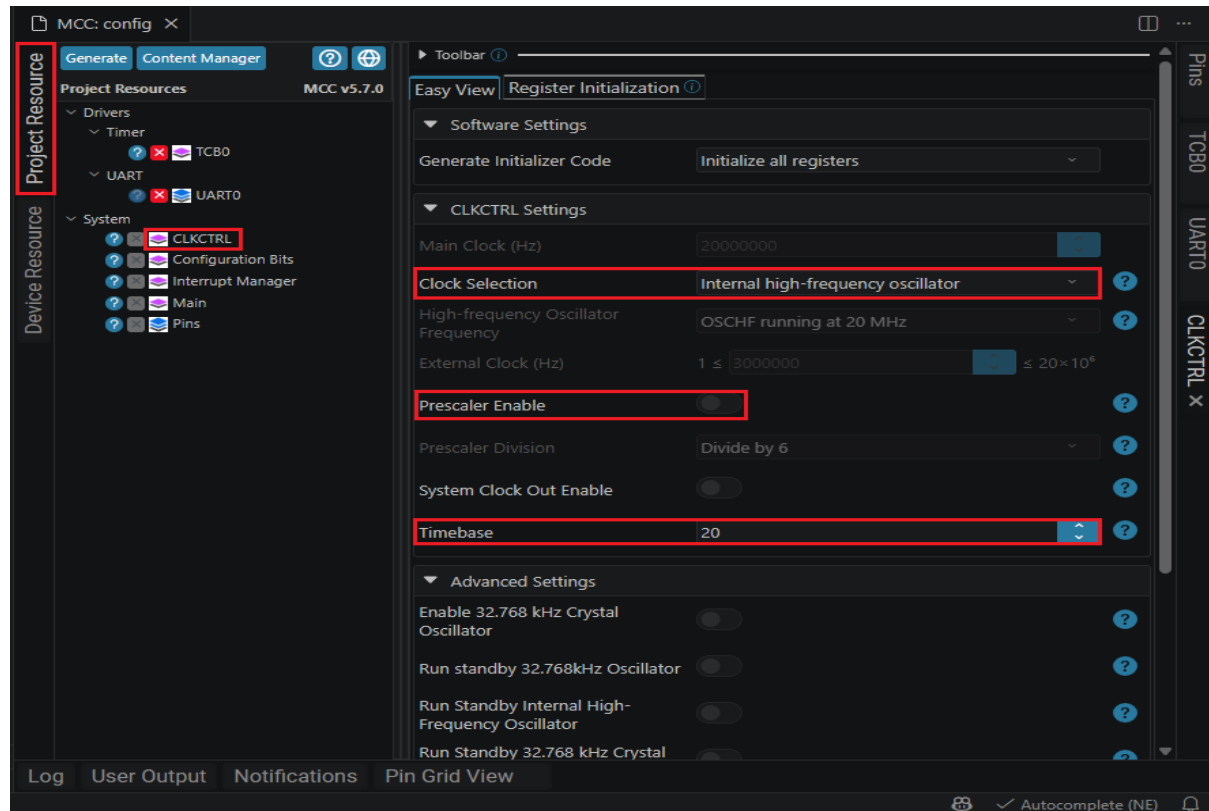
Figure 5-12. MPLAB® Code Configurator—Project Resources



In the following steps, configure the required peripherals.

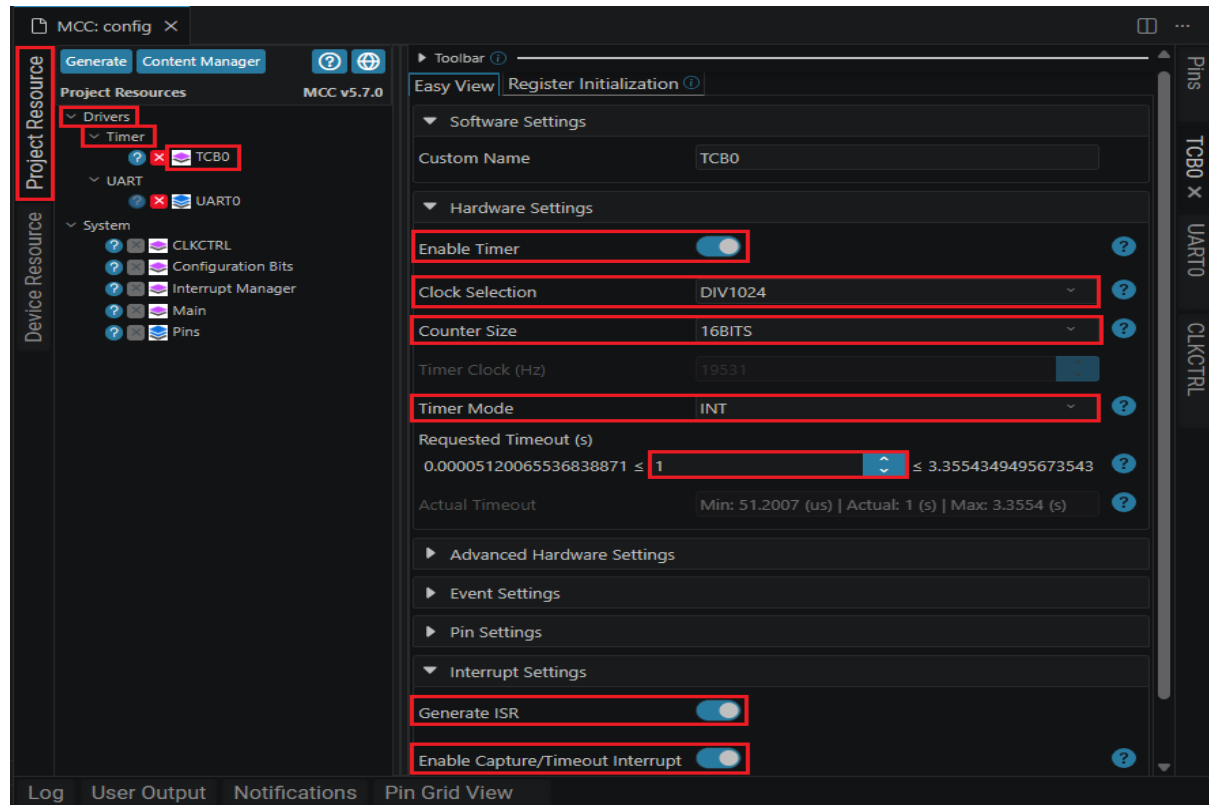
3. Configure the clock with the settings listed below by navigating to Project Resources>System>CLKCTRL:
 - Clock Selection: Internal high-frequency oscillator
 - Toggle off Prescaler Enable
 - Write Timebase to 20

Figure 5-13. MPLAB® Code Configurator—Clock Configuration



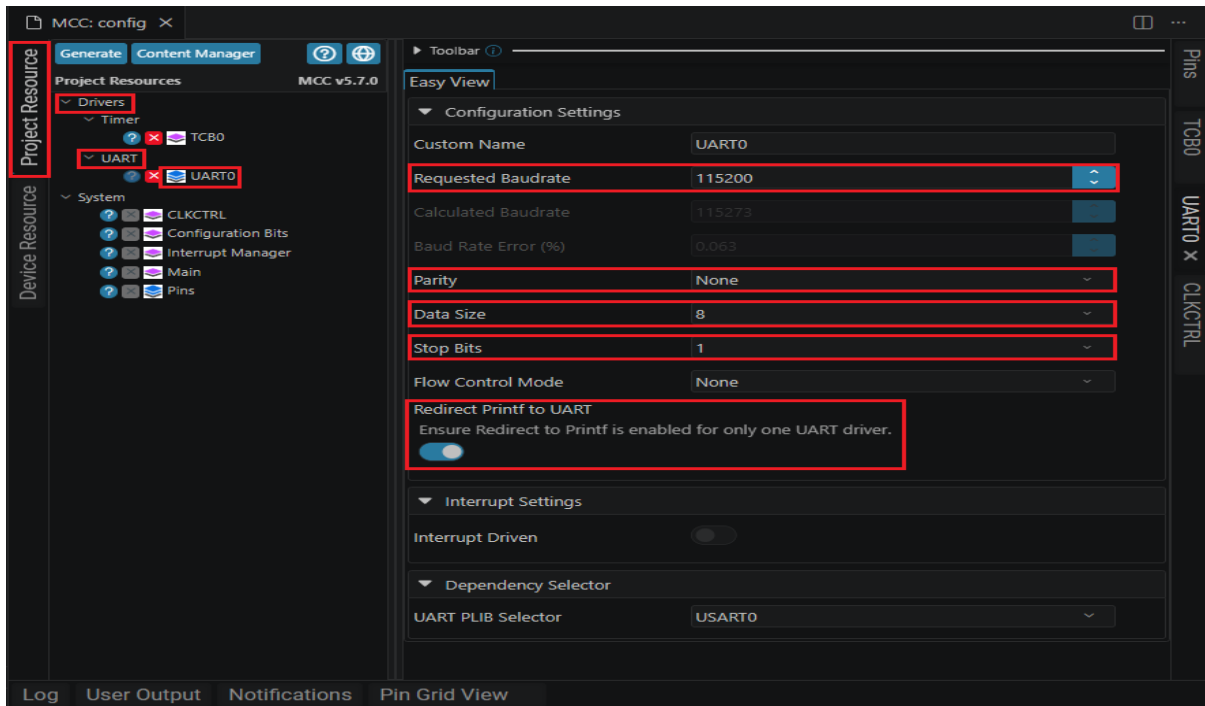
4. Configure TCB0 with the settings listed below by navigating to *Project Resources > Drivers > Timer > TCB0*:
 - Toggle on Enable Timer
 - Clock Selection: DIV1024
 - Counter Size: 16BITS
 - Timer Mode: INT
 - Requested Timeout (s): 1
 - Toggle on Generate ISR
 - Toggle on Enable Capture/Timeout Interrupt

Figure 5-14. MPLAB® Code Configurator—TCB0 Configuration



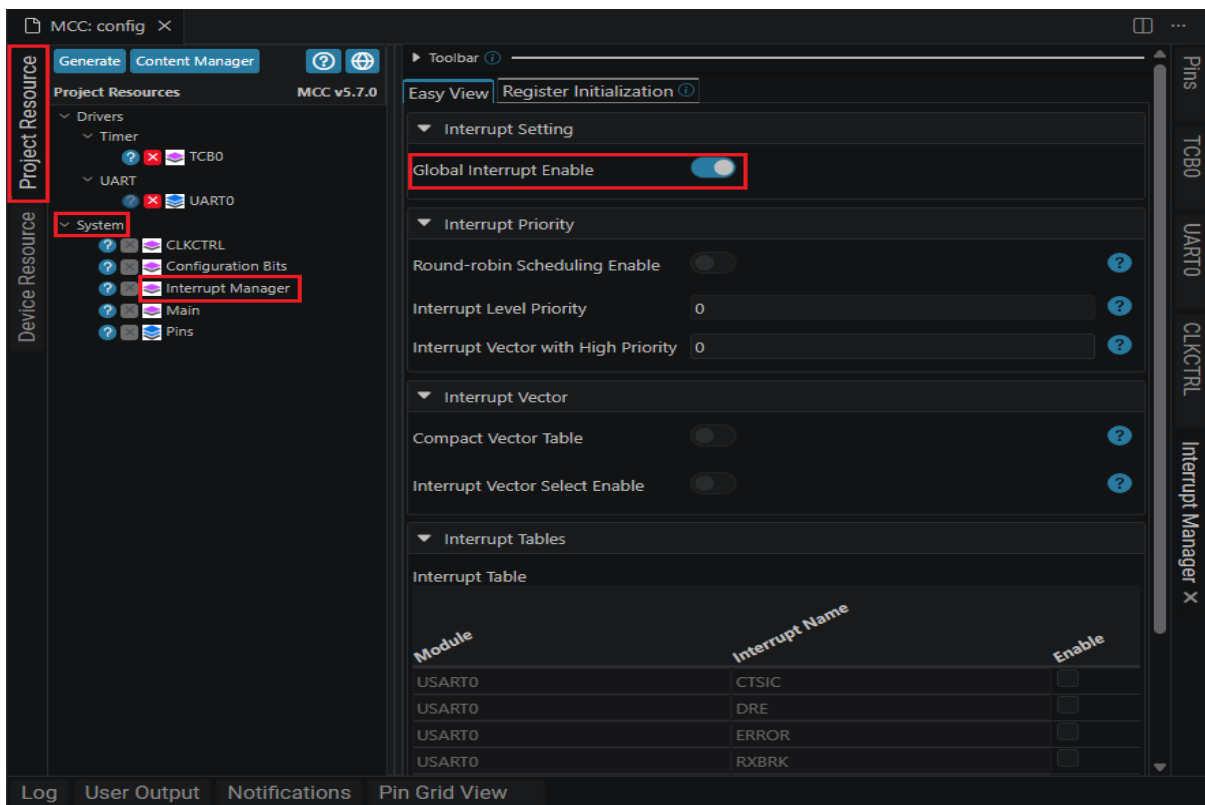
5. Configure UART0 with the settings listed below by navigating to *Project Resources > Drivers > UART > UART0*:
 - Requested Baudrate: 115200
 - Parity: None
 - Data size: 8 bits
 - Stop bits: 1 bit
 - Toggle on Redirect Printf to UART

Figure 5-15. MPLAB® Code Configurator—UART Configuration



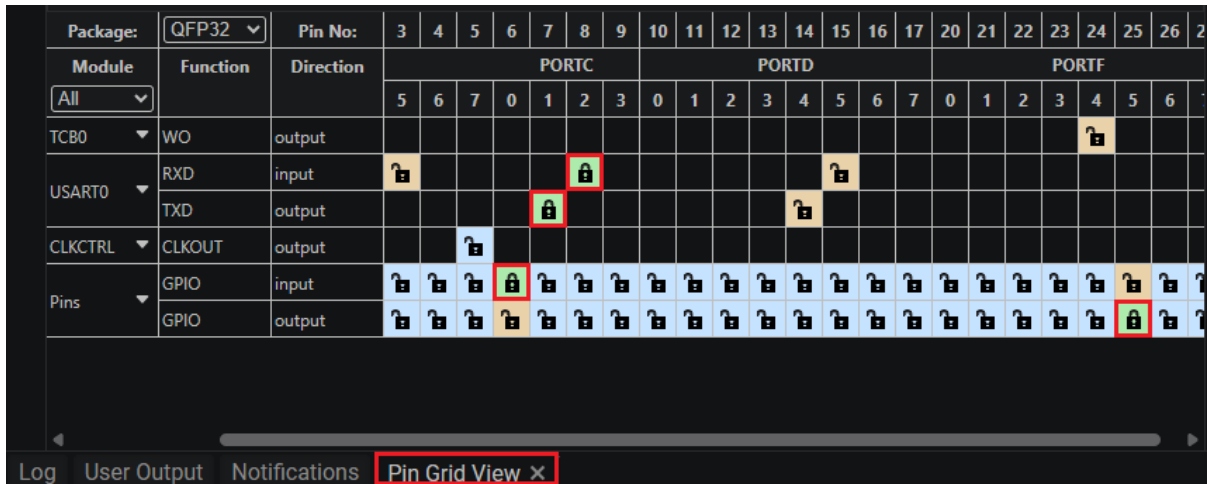
6. Configure the Interrupt Controller by navigating to *Project Resources > System > Interrupt Manager*:
 - Toggle on Global Interrupt Enable

Figure 5-16. MPLAB® Code Configurator—Interrupt Controller Configuration



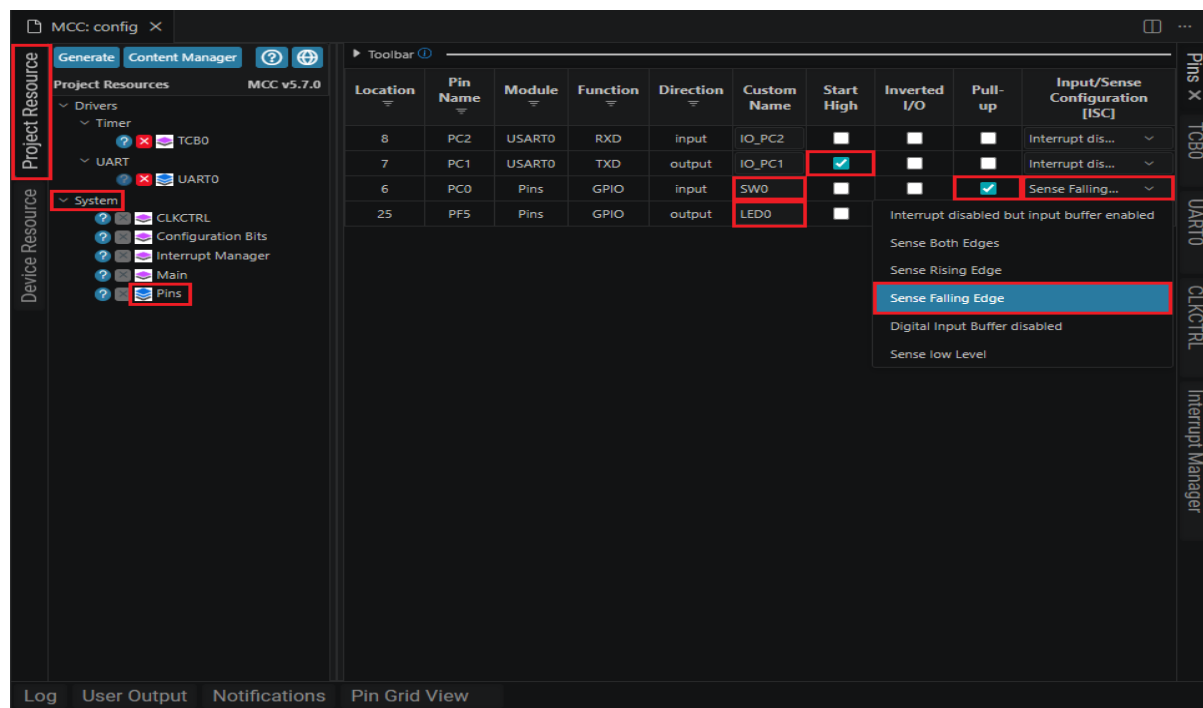
7. Click the **Pin Grid View** tab and configure the pins as shown below:
 - USART0: RXD—PC2, TXD—PC1
 - Pins: GPIO Input—PC0, GPIO Output—PF5

Figure 5-17. MPLAB® Code Configurator—Pin Grid View



8. Configure the selected pins by going to *Project Resources > System > Pins*:
 - PC0:
 - Custom Name: SW0
 - Enable Pull-Up
 - ISC configuration: Sense Falling Edge
 - PF5:
 - Custom Name: LED0
 - PC1:
 - Enable Start High

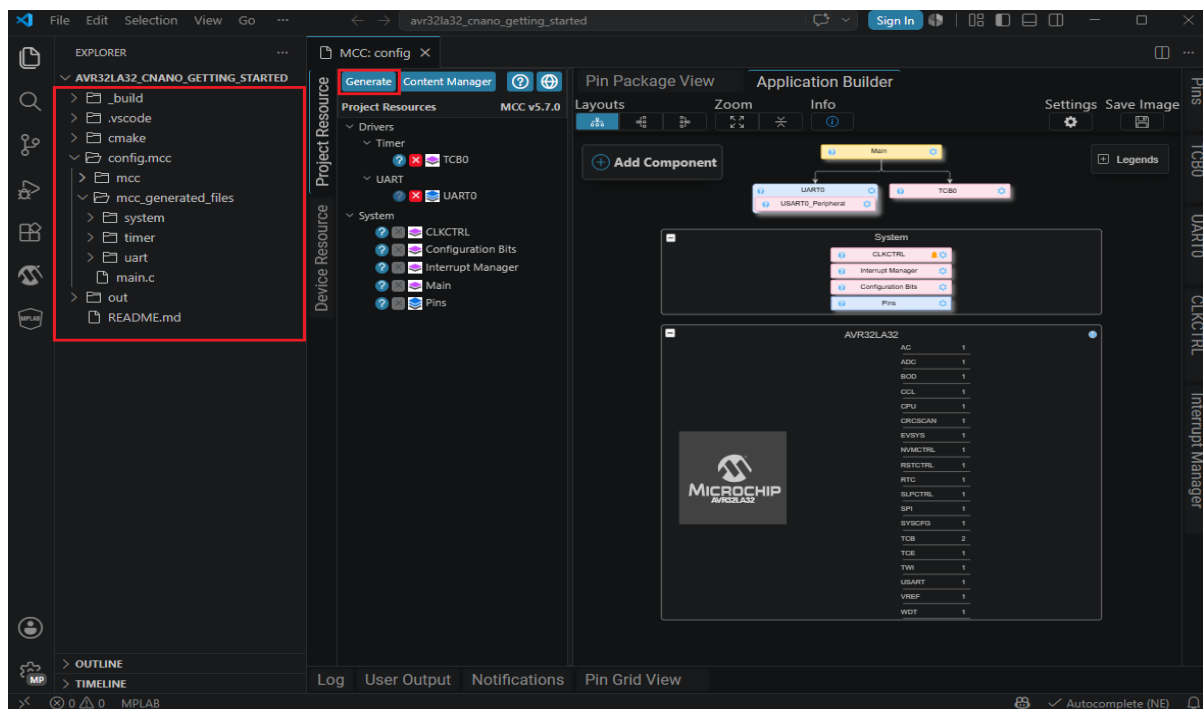
Figure 5-18. MPLAB® Code Configurator—Pins Configuration



5.3. Generate the Code

After configuring the peripherals as shown in the following figure, click the **Generate** button in the top-left corner of the **MCC** tab.

Figure 5-19. MPLAB® Code Configurator—Generate the Configuration



5.4. Add the Application Logic

To develop and run the application, open the `main.c` file of the project and perform the following steps:

1. Include the following files in the `main.c` file:

```
#include <util/delay.h>
```

2. Add the following definitions, which specify the value to be loaded into the timer's CCMP register to generate the compare interrupt.

```
// Value to load into the TCB0 CCMP register for the 1s timer interrupt; CCMP = (F_CPU /
// Prescaler) * Desired Time
#define TIMER_PERIOD_1S (19530U)
// Value to load into the TCB0 CCMP register for the 200 ms timer interrupt; CCMP =
(F_CPU / Prescaler) * Desired Time
#define TIMER_PERIOD_200MS (3906U)
// Value to clear the TCB0 counter
#define TIMER_COUNTER_CLEAR (0x00)
```

3. Add the following global variables, to track the state of the LED blink interval and the switch press.

```
static bool blinkIntervalState = false;
static volatile bool isSwitchPressed = false;
```

4. Add the callback functions for the timer compare and switch press-detection interrupts.

```
void DetectSwitchPress(void)
{
    // Debounce delay
    delay_ms(20);
    if (!SW0_GetValue())
    {
        isSwitchPressed = true;
    }
}

void ToggleLED(void)
{
    LED0_Toggle();
}
```

5. Add the following functions, which are used to alternate the LED blinking interval.

The `TCB0_Reset()` function clears the timer counter and Interrupt registers so that the timer starts from a known state when alternating the timer compare values.

The `VaryLEDBlinkTime()` function varies the LED blink interval by writing the corresponding value to the CCMP register of TCB0 for each switch press, depending on the previous LED blink interval state, and enables the timer interrupt.

```
static void TCB0_Reset(void)
{
    TCB0_ClearCaptInterruptFlag();
    TCB0_Write(TIMER_COUNTER_CLEAR);
    TCB0_DisableCaptInterrupt();
}

static void VaryLEDBlinkTime(void)
{
    LED0_SetLow();
    // Toggles the state of the LED blinking speed
    blinkIntervalState = !blinkIntervalState;
    // Disables the TCB0 interrupt to prevent ISR execution during the CCMP update and
    // resets the timer counter
    TCB0_Reset();
    // Updates the timer compare value based on the current blink interval state
    if (blinkIntervalState == false)
    {
        TCB0_CaptureCompareWrite(TIMER_PERIOD_1S);
        printf("LED blink interval changed to 2s\r\n");
    }
}
```

```

    }
    else
    {
        TCB0_CaptureCompareWrite(TIMER_PERIOD_200MS);
        printf("LED blink interval changed to 400 ms\r\n");
    }
    TCB0_EnableCaptInterrupt();
}

```

6. Add the main function to initialize the peripherals, set up callback functions, and enable global interrupts. The function continuously monitors the switch input and alternates the blinking interval whenever the switch is pressed.

```

int main(void)
{
    SYSTEM_Initialize();

    // Sets the interrupt handler for the timer interrupt. LED0 toggles when the timer
    // interrupt occurs.
    TCB0_CaptureCallbackRegister(ToggleLED);

    // Sets the interrupt handler for the switch press
    SW0_SetInterruptHandler(DetectSwitchPress);

    printf("*****PRINTING LED BLINK INTERVAL TIME*****\r\n");
    printf("LED blink interval is 2s \r\n");
    while (1)
    {
        if (isSwitchPressed)
        {
            VaryLEDBlinkTime();
            isSwitchPressed = false;
        }
    }

    return 0;
}

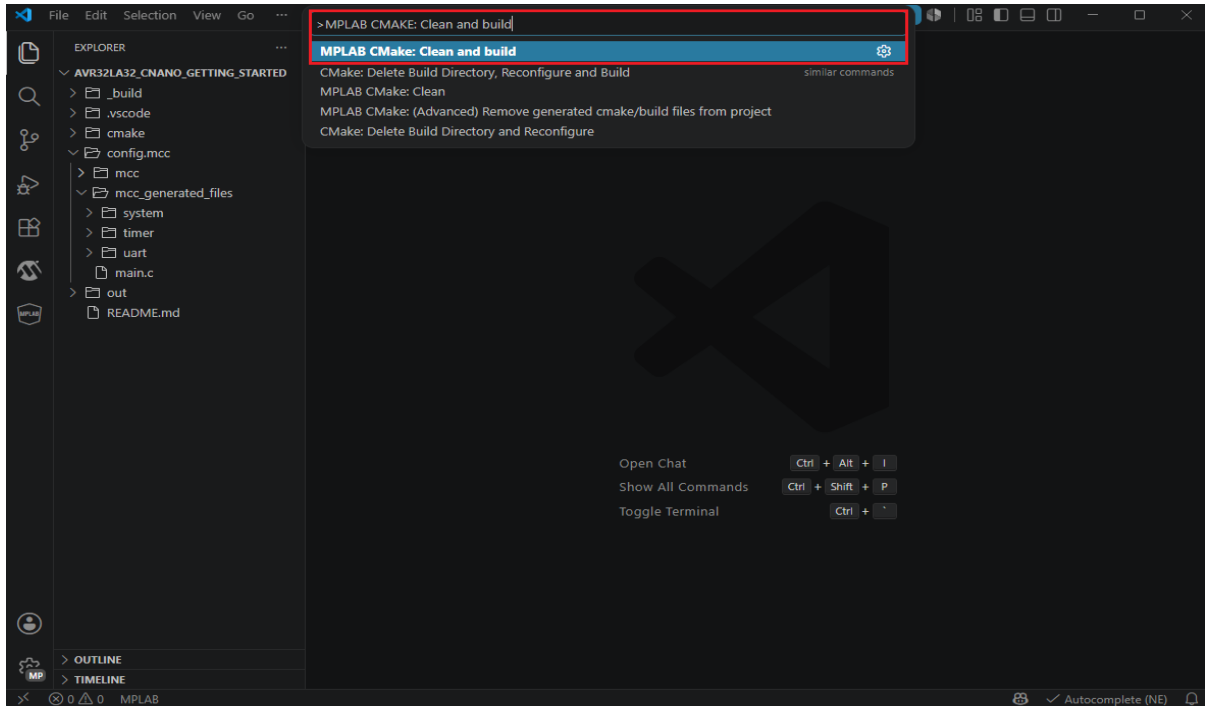
```

5.5. Build and Program the Application

To build and program the application, follow these steps:

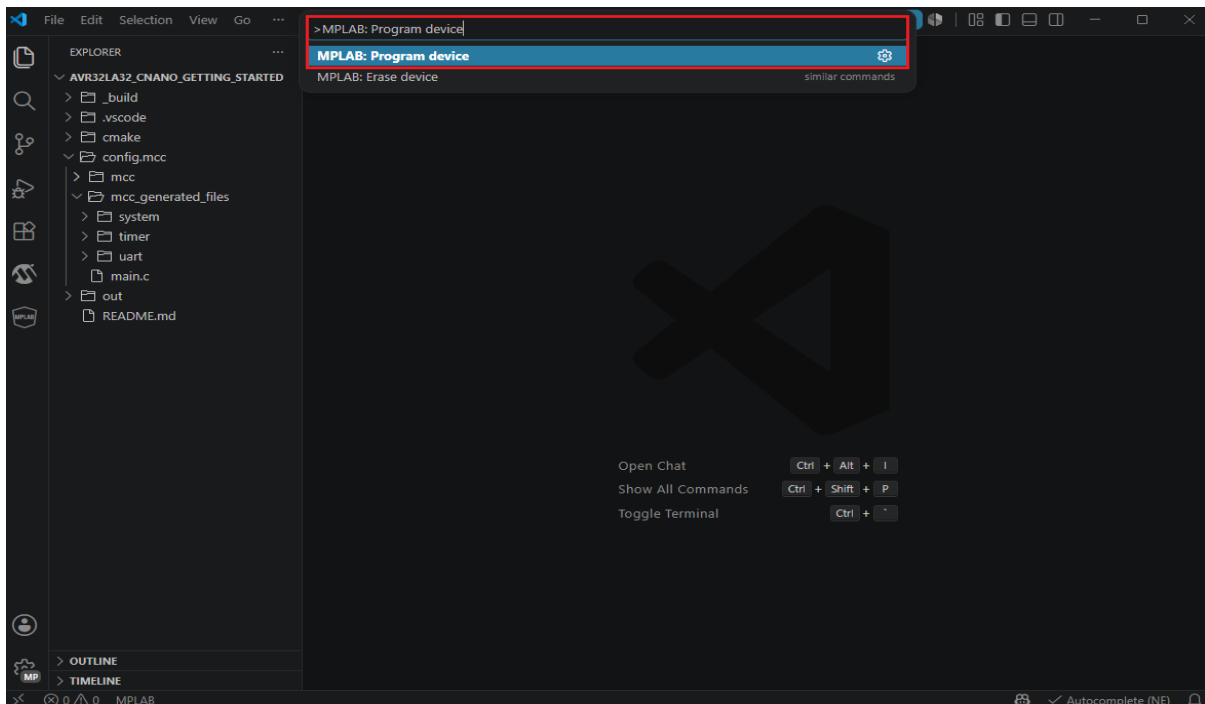
1. The AVR32LA32 Curiosity Nano evaluation kit supports programming and debugging using an on-board debugger. Connect the USB Type-C® cable to the AVR32LA32 Curiosity Nano evaluation kit to power and debug the Curiosity Nano board.
2. Clean and build the project by typing `>MPLAB CMAKE: Clean and build` in the Command Palette.

Figure 5-20. Clean and Build the Project



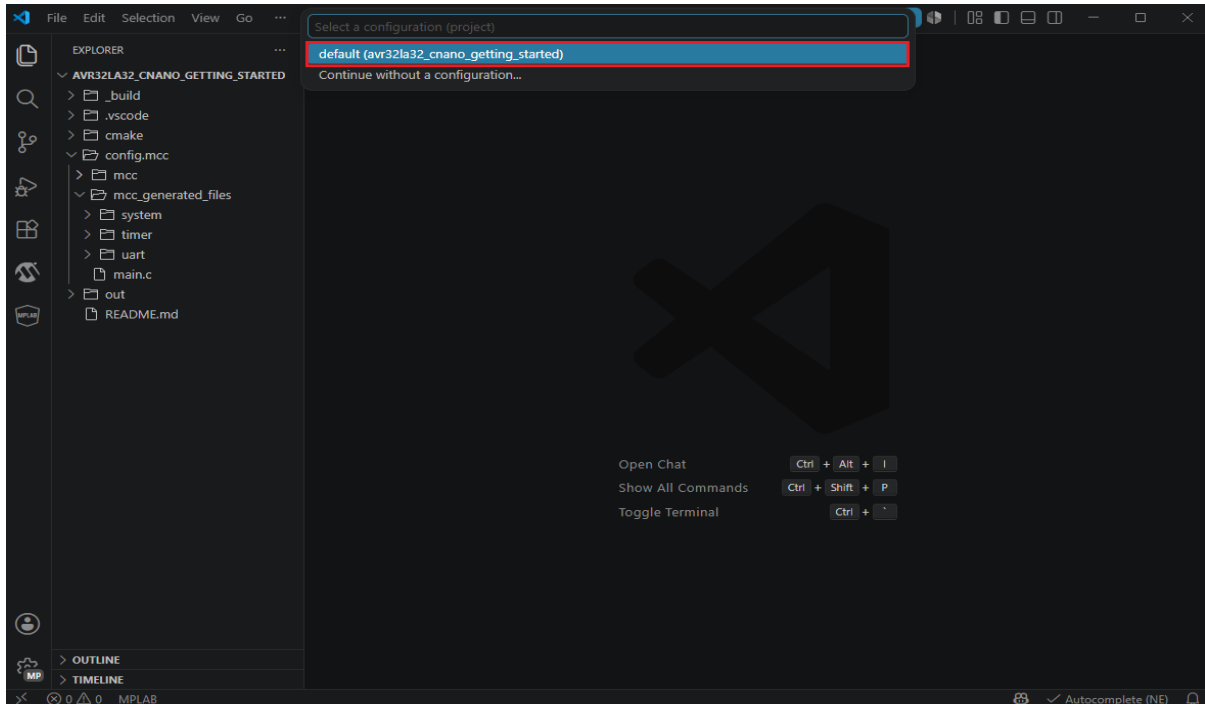
3. Program the device by typing `>MPLAB: Program device` in the Command Palette.

Figure 5-21. Program the Device



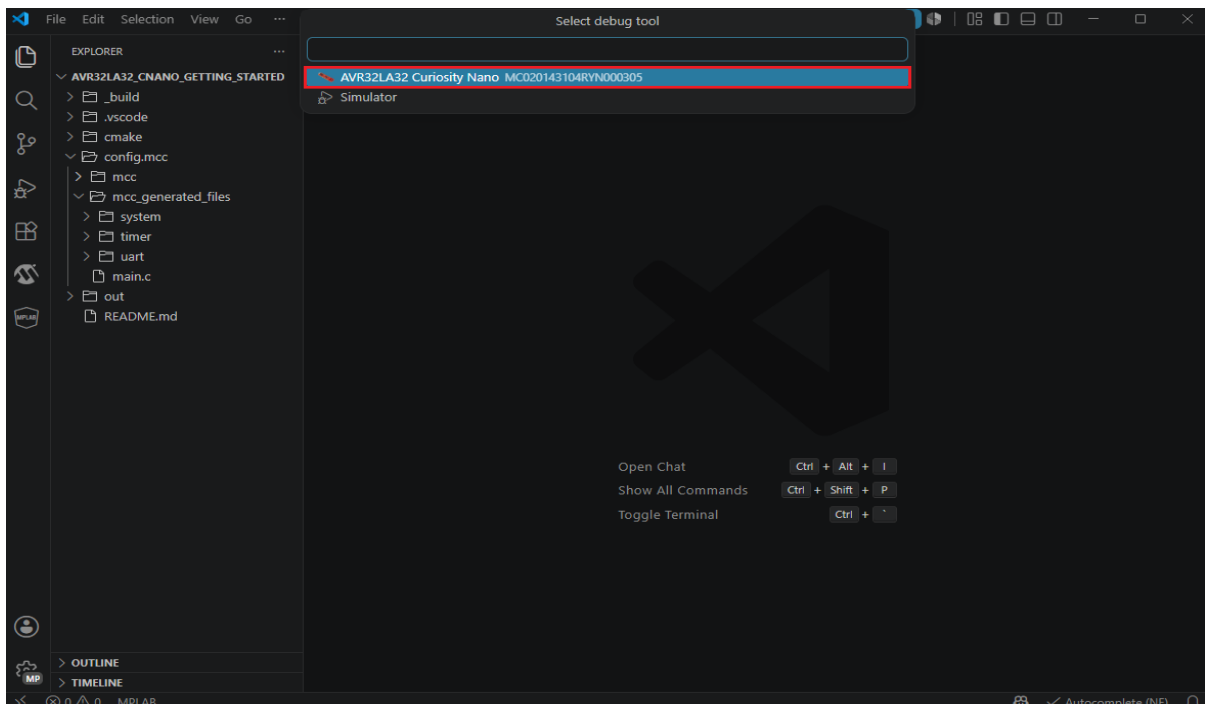
4. Select the project to program onto the device.

Figure 5-22. Select the Project



5. Select the evaluation kit to use for programming.

Figure 5-23. Select the Evaluation Kit

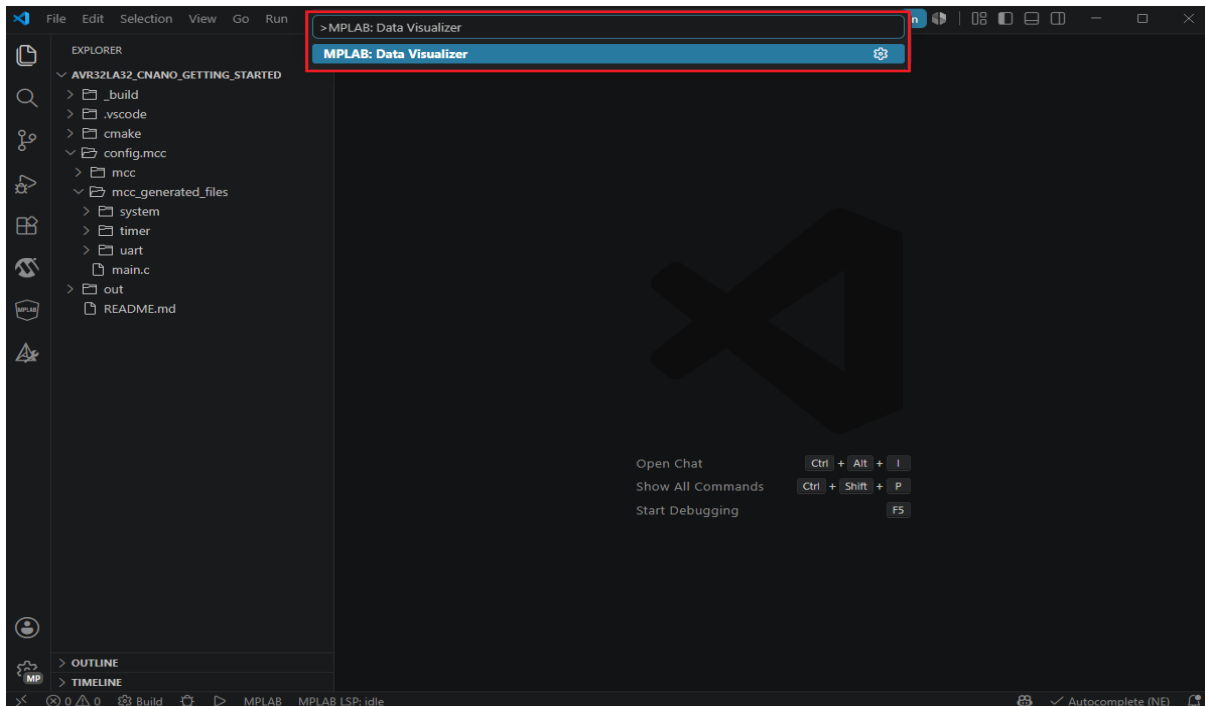


5.6. Observe the Output

To observe the output on the board and in the serial terminal, follow these steps:

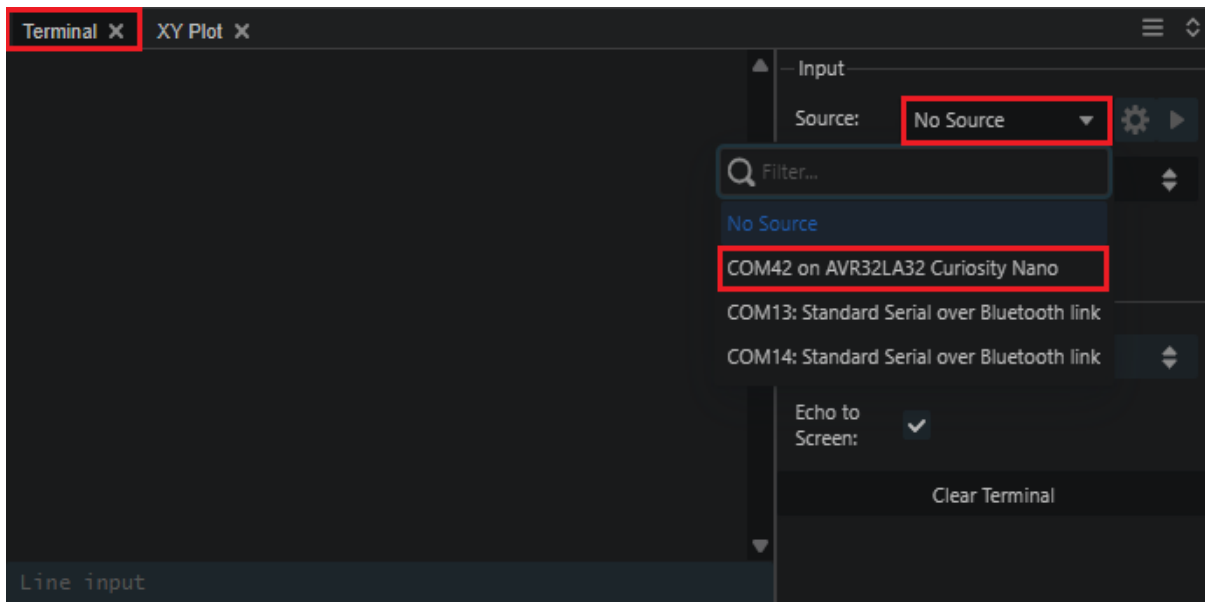
1. To launch MPLAB® Data Visualizer, type `>MPLAB: Data Visualizer` in the Command Palette.

Figure 5-24. Launch MPLAB® Data Visualizer



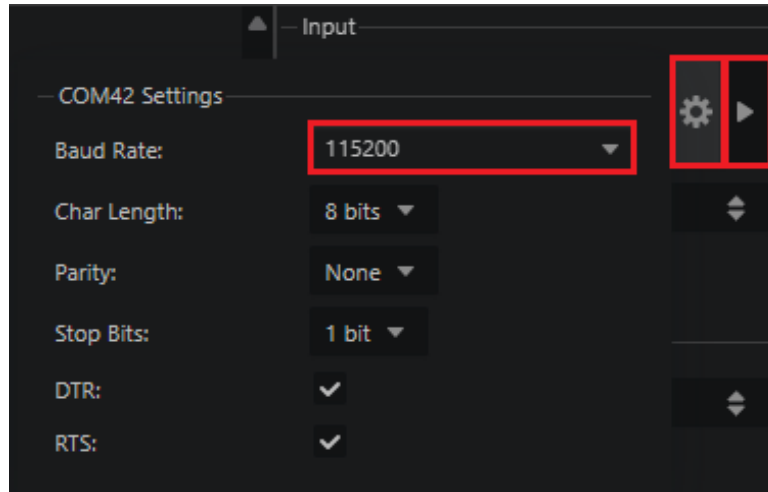
2. Select the COM port corresponding to the Curiosity Nano from the Source drop-down menu in the terminal window.

Figure 5-25. Serial Communication Port Selection



3. Set the baud rate to 115200 by clicking the settings icon next to the COM port.
4. Click the play icon to view the UART Communication in the terminal window.

Figure 5-26. Setting the Baud Rate



5. The on-board LED of the AVR32LA32 Curiosity Nano evaluation kit toggles at a default interval of two seconds. The LED blink interval is displayed on the serial terminal.
 - Press the SW switch on the AVR32LA32 Curiosity Nano evaluation kit to change the blink interval to 400 ms
 - Each press of the on-board switch (SW0) on the Curiosity Nano evaluation kit alternates the blink interval between 400 ms and two seconds

Figure 5-27. Output Observed on Terminal

```
*****PRINTING LED BLINK INTERVAL TIME*****
LED blink interval is 2s
LED blink interval changed to 400 ms
LED blink interval changed to 2s
LED blink interval changed to 400 ms
LED blink interval changed to 2s
LED blink interval changed to 400 ms
```

Note: The application discussed above is implemented and can be accessed on [MPLAB® Discover](#).

6. Getting Started With MPLAB® X IDE

Prerequisites

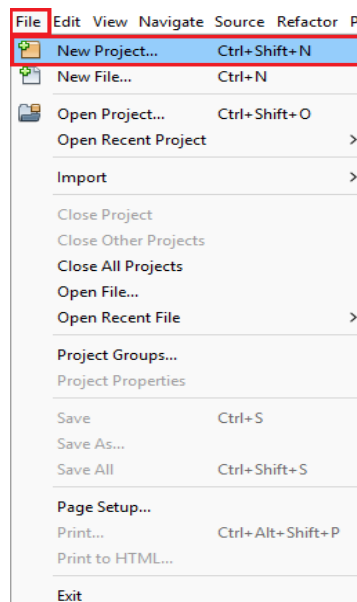
- Install the following software tools for this demonstration:
 - MPLAB X IDE (v6.30 or later)
 - Toolchain Support for MPLAB (XC8 v3.10 or later)
 - MPLAB®Code Configurator (MCC) for Microchip devices (v5.7.0 or later)
- Hardware tool used for this demonstration:
 - AVR32LA32 Curiosity Nano evaluation kit (EV24D63A)
- Connect the AVR32LA32 Curiosity Nano board to the PC via the on-board USB connector, which is connected to the embedded debugger. The kit is powered through the USB connection, and the embedded debugger enables debugging and programming via USB.

6.1. Create the First Application on the AVR® LA MCU

To create an MCC Melody-based project, follow these steps:

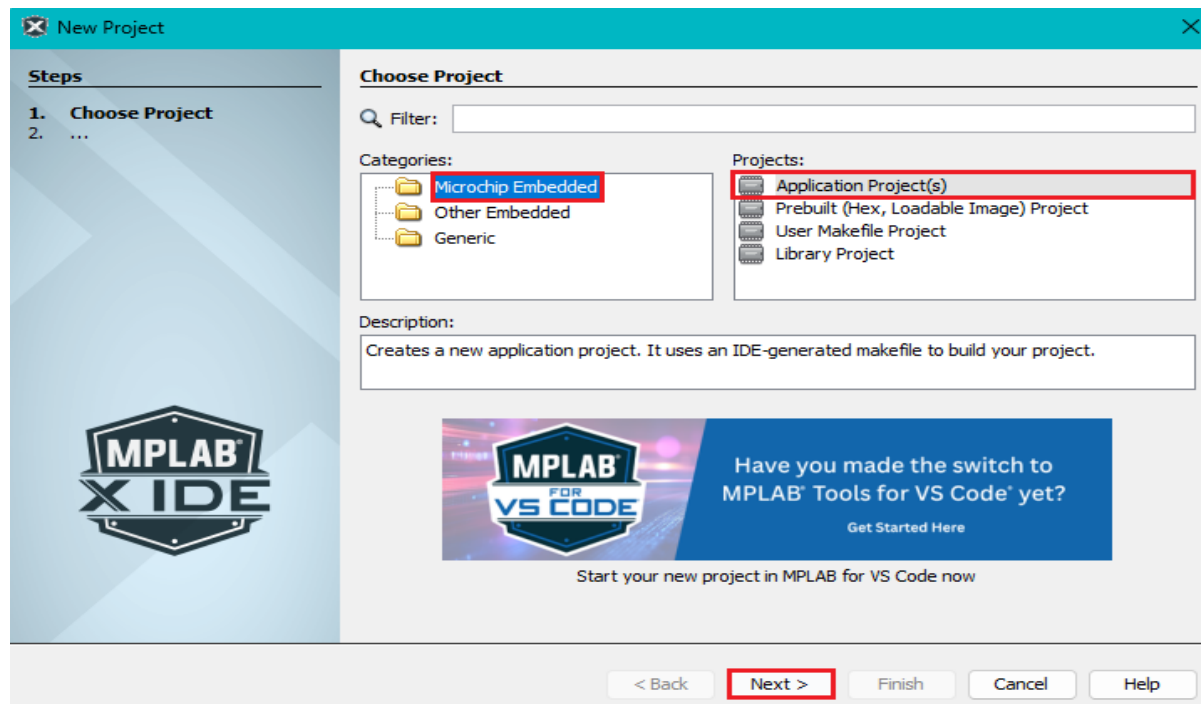
1. Launch MPLAB® X IDE.
2. From the File menu, click New Project, or click the “New Project” icon. The New Project window will open.

Figure 6-1. Create a New Project in MPLAB® X IDE



3. In the Choose Project step, select the following:
 - Categories: Microchip Embedded
 - Projects: Application Project(s)

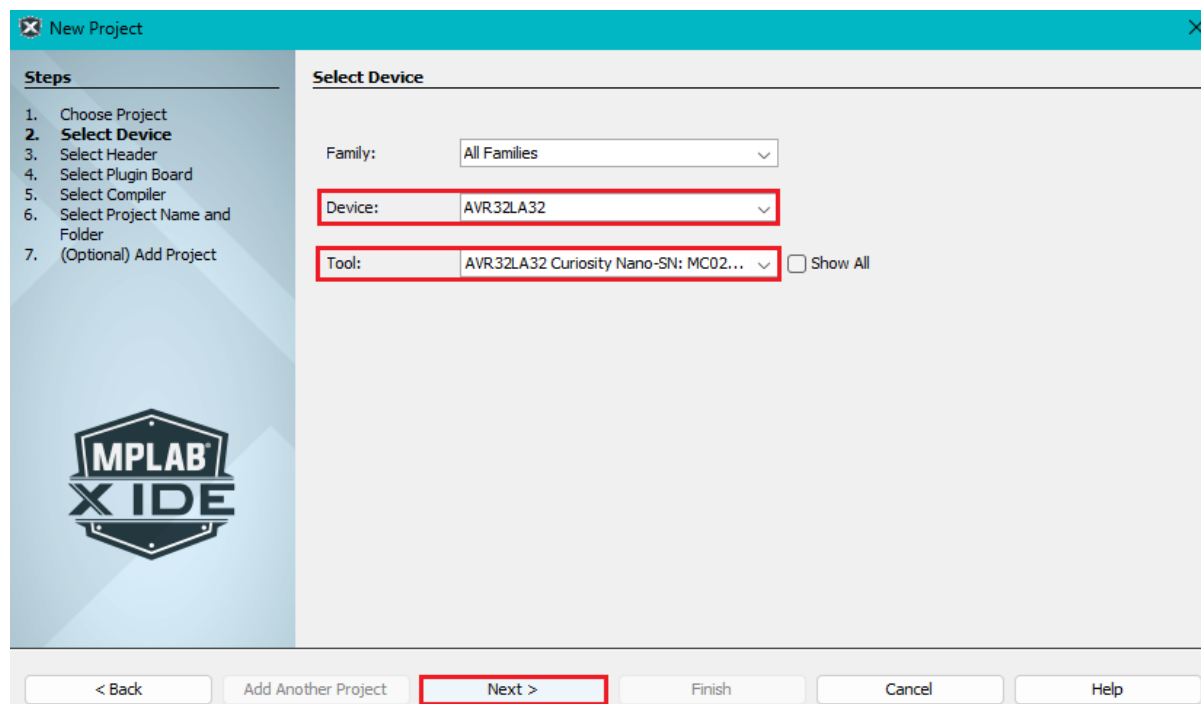
Click **Next**

Figure 6-2. New Project Window, Project Selection

4. In the Select Device step, select the following:

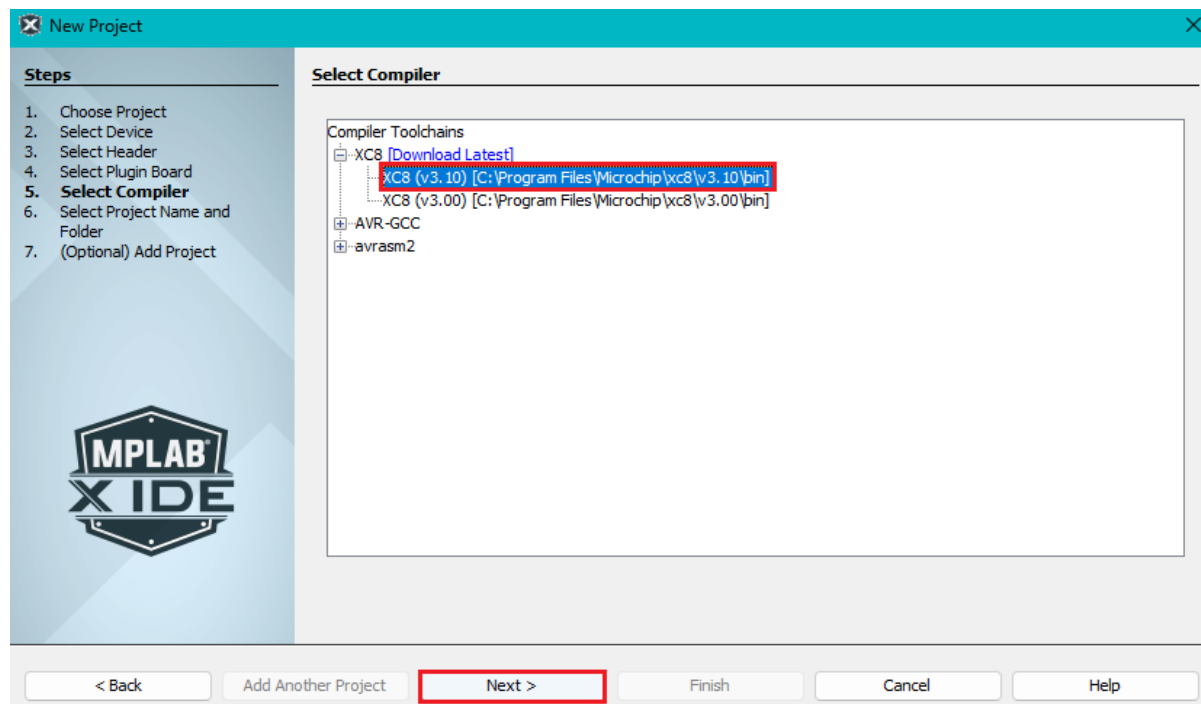
- Device: AVR32LA32
- Tool: AVR32LA32 Curiosity Nano

Click **Next**

Figure 6-3. Device and Tool Selection

5. In the Select Compiler step, expand the XC8 list of options, then select the compiler (for example, XC8 v3.10). Click **Next**.

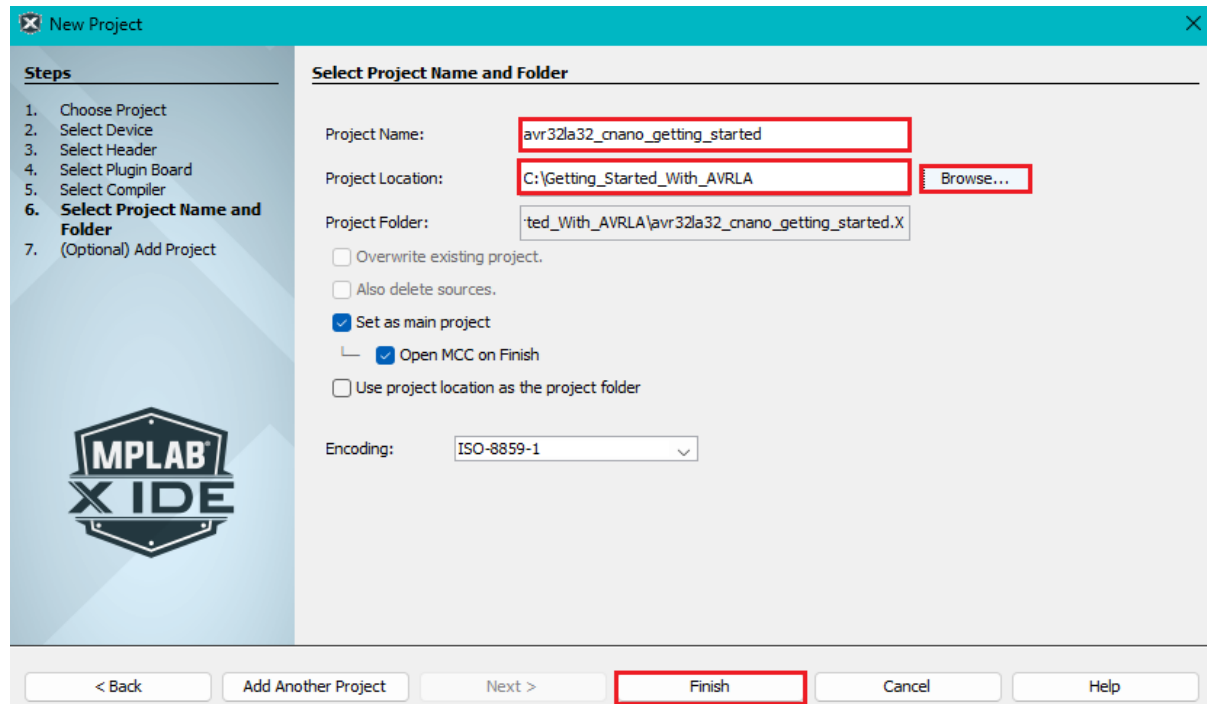
Figure 6-4. Compiler Selection



6. In the Select Project Name and Folder step, enter the following:
 - Project Name: avr321a32 _cnano_ getting_started
 - Project Location: C:\GettingStarted_With_AVRLA
 - The project location can be specified either by entering the path in the field or by clicking the **Browse** button next to the field

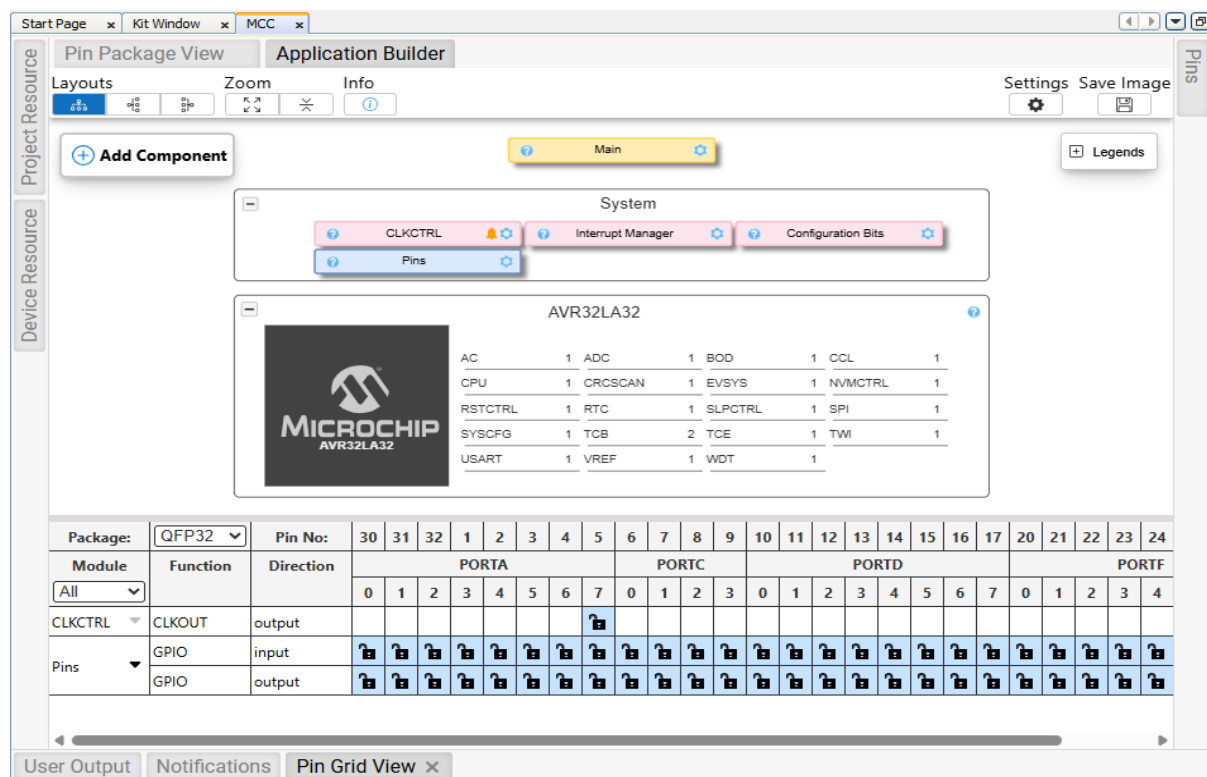
Click **Finish**

Figure 6-5. Project Name and Project Location



- The project is created, and the MCC plug-in launches in a new window.

Figure 6-6. MPLAB® Code Configurator Window

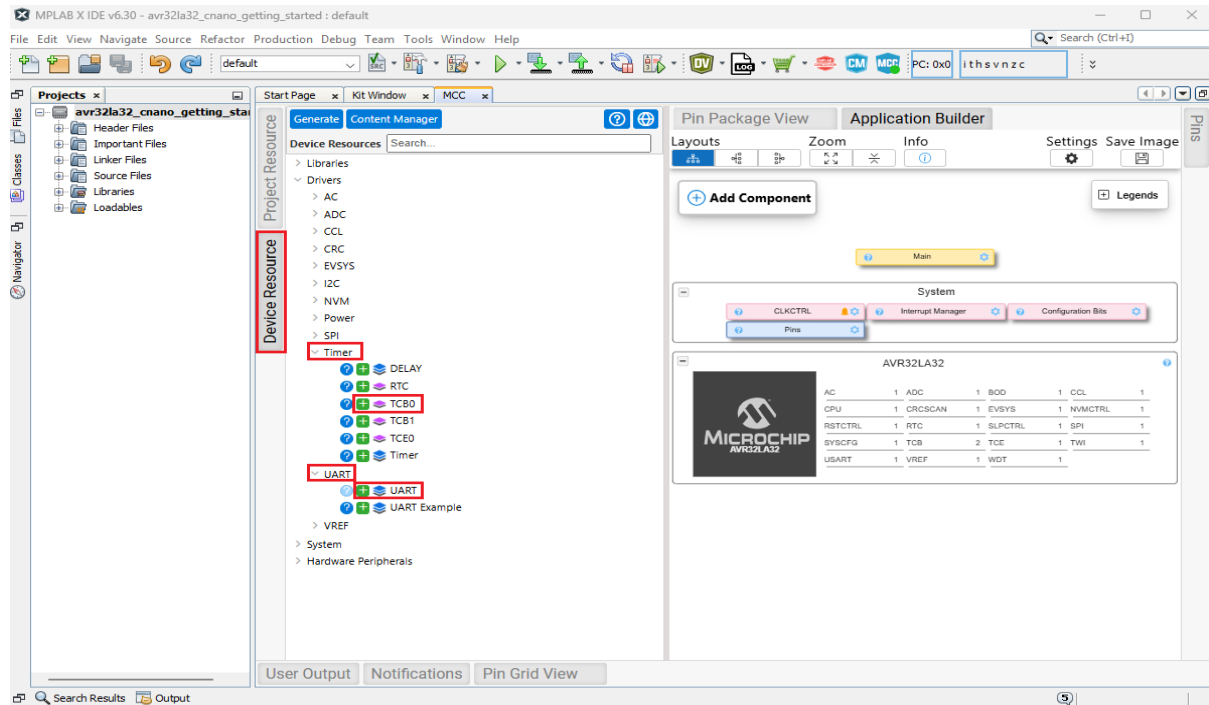


6.2. Configure the Application Using MCC Melody

Follow the steps below to add and configure peripherals using MCC Melody:

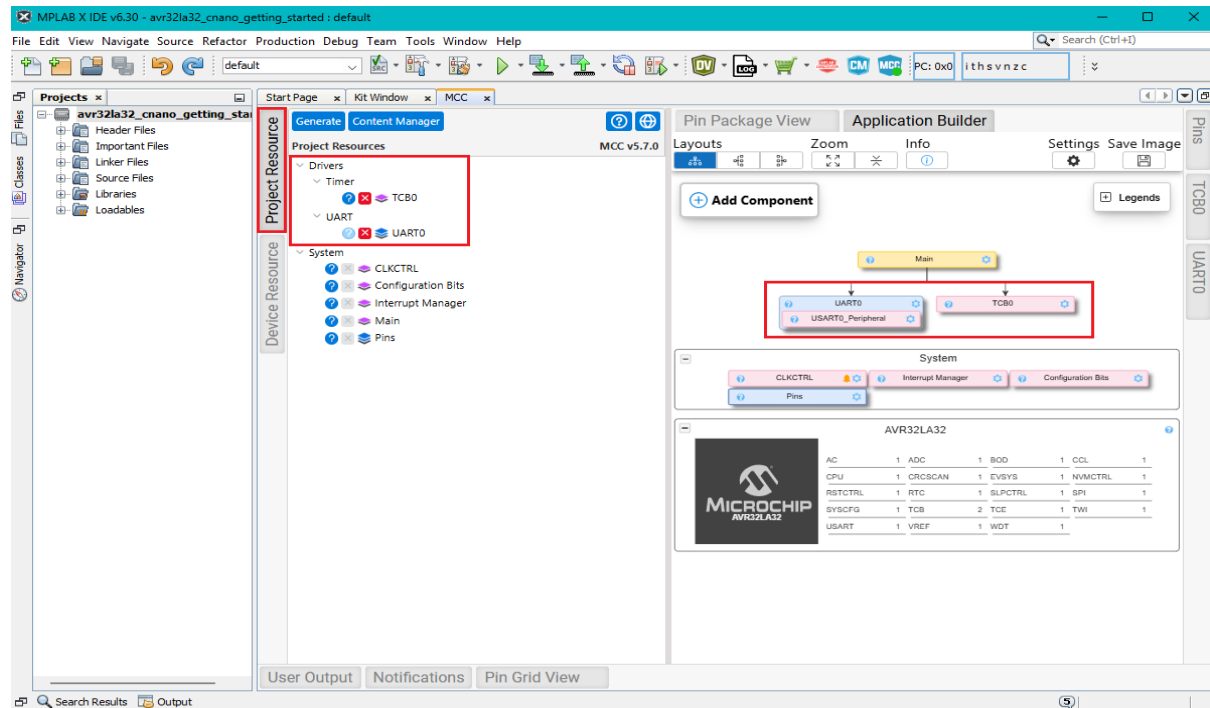
1. Click the **Device Resources** tab to view the drivers and libraries available for the device. Expand the Drivers drop-down and configure the following:
 - Expand the Timer drop-down and add TCB0
 - Expand the UART drop-down and add UART

Figure 6-7. MPLAB® Code Configurator—Device Resources



2. The peripherals added to the project will be visible in the **Project Resources** tab and the Application Builder view.

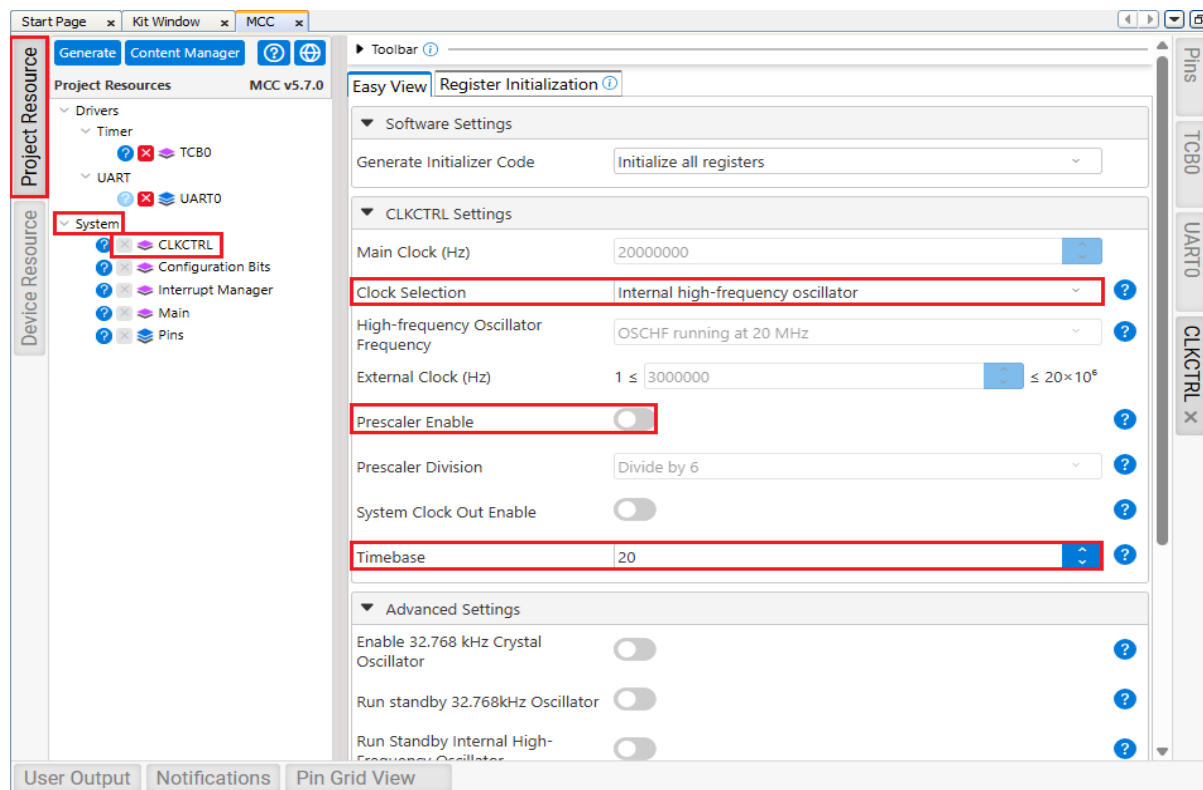
Figure 6-8. MPLAB® Code Configurator—Project Resources



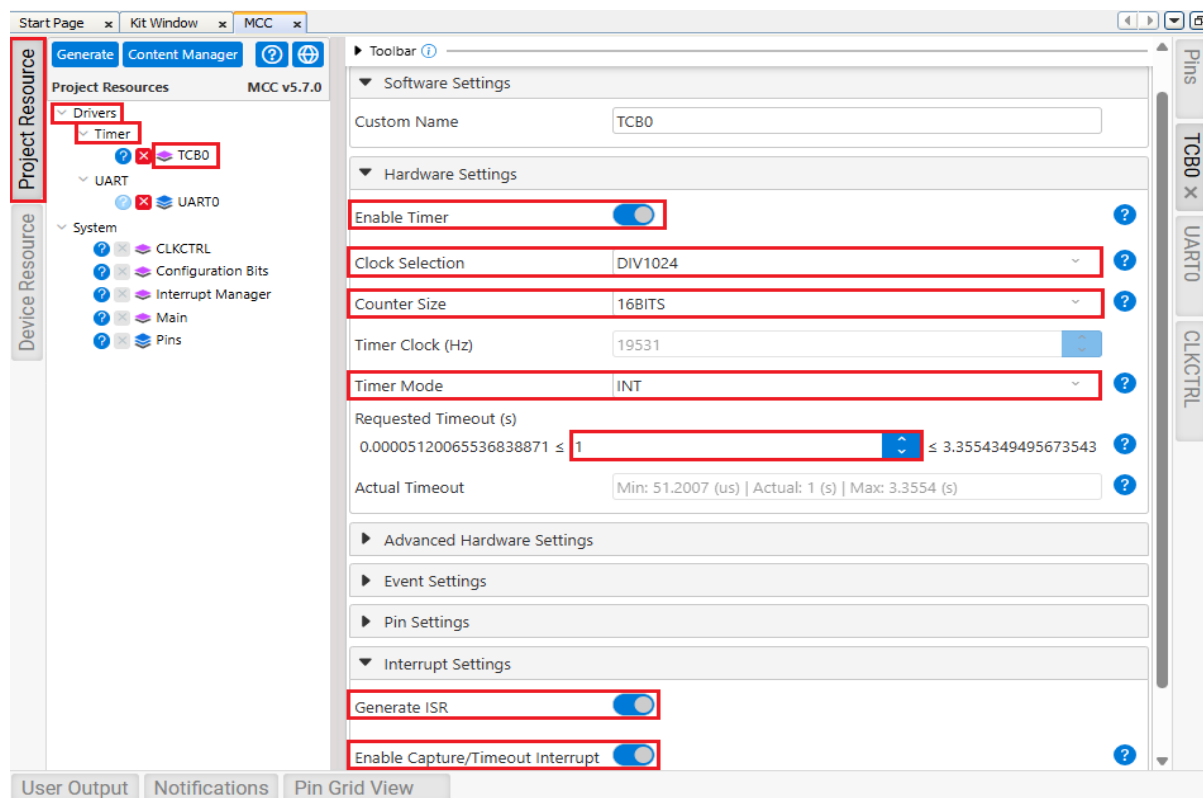
In the following steps, configure the required peripherals.

3. Configure the clock with the settings listed below by navigating to Project Resources > System > CLKCTRL:
 - Clock Selection: Internal high-frequency oscillator
 - Toggle off Prescaler Enable
 - Write Timebase to 20

Figure 6-9. MPLAB® Code Configurator—Clock Configuration

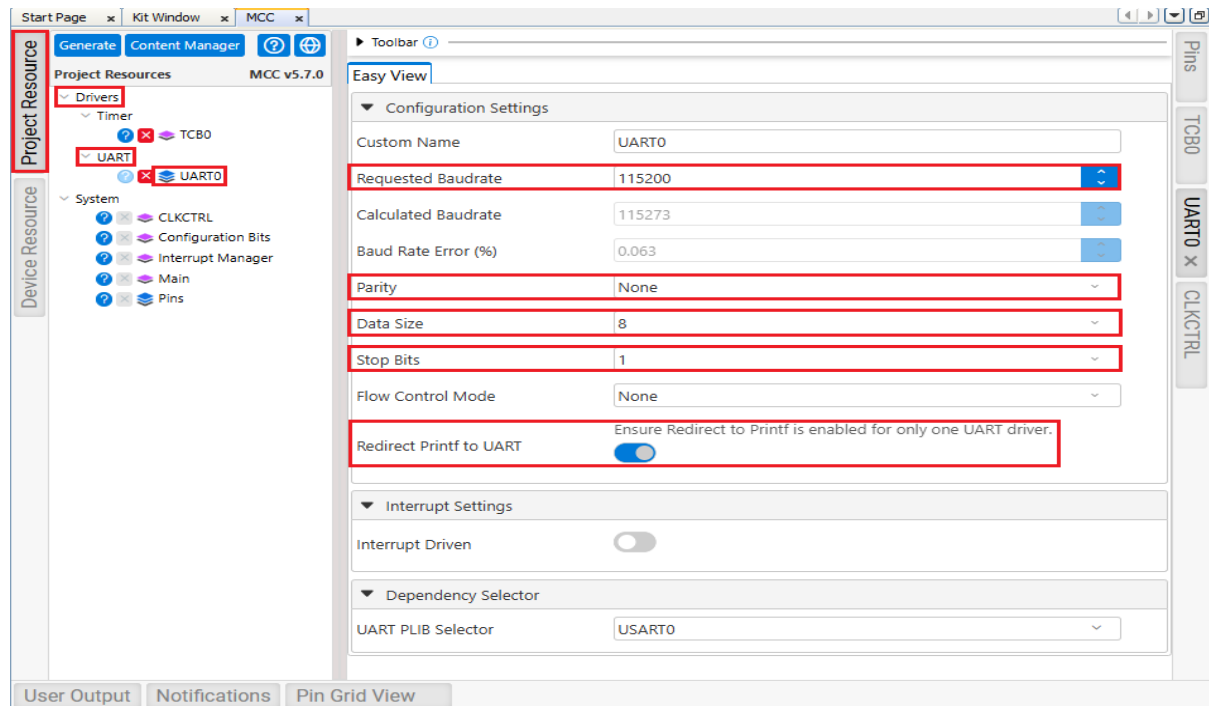


4. Configure TCB0 with the settings listed below by navigating to *Project Resources > Drivers > Timer > TCB0*:
 - Toggle on Enable Timer
 - Clock Selection: DIV1024
 - Counter Size: 16BITS
 - Timer Mode: INT
 - Requested Timeout (s): 1
 - Toggle on Generate ISR
 - Toggle on Enable Capture/Timeout Interrupt

Figure 6-10. MPLAB® Code Configurator—TCB0 Configuration

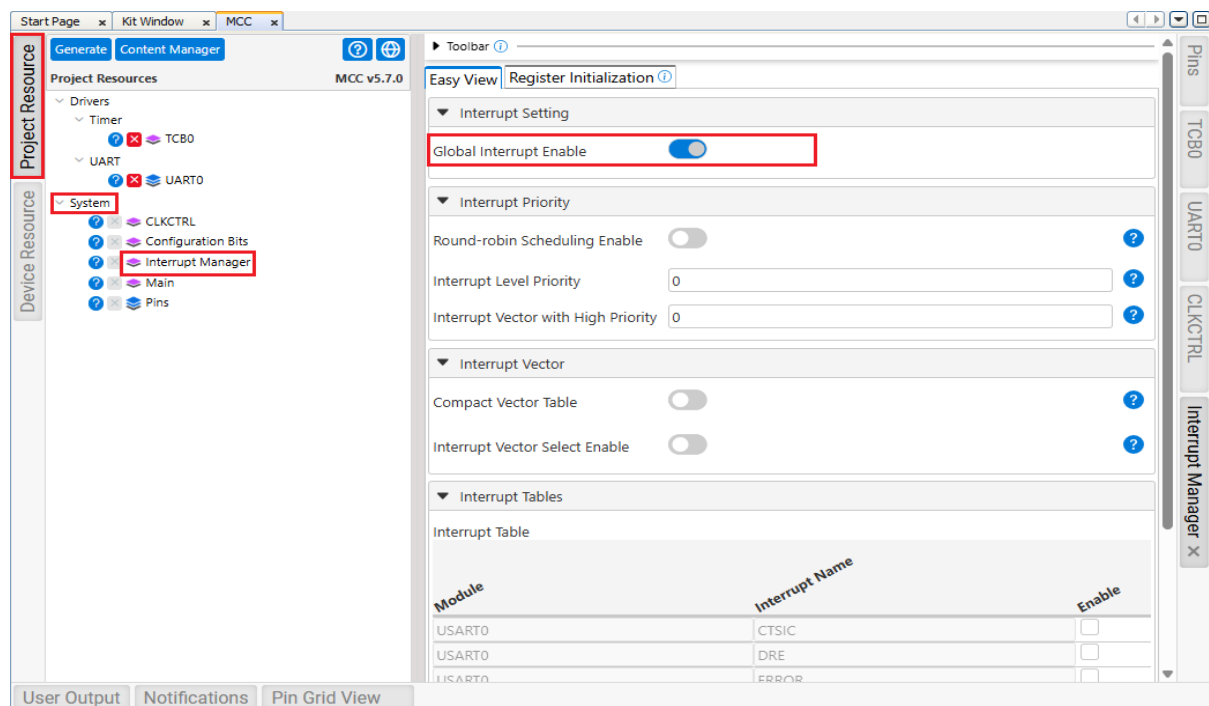
5. Configure UART0 with the settings listed below by navigating to *Project Resources > Drivers > UART > UART0*:
 - Requested Baudrate: 115200
 - Parity: None
 - Data size: 8 bits
 - Stop bits: 1 bit
 - Toggle on Redirect Printf to UART

Figure 6-11. MPLAB® Code Configurator—UART Configuration



- Configure the Interrupt Controller by navigating to *Project Resources > System > Interrupt Manager*:
 - Toggle on Global Interrupt Enable

Figure 6-12. MPLAB® Code Configurator—Interrupt Controller Configuration



- Click the **Pin Grid View** tab and configure the pins as shown below:
 - USART0: RXD—PC2, TXD—PC1
 - Pins: GPIO Input—PC0, GPIO Output—PF5

Figure 6-13. MPLAB® Code Configurator—Pin Grid View

Package:	QFP32	Pin No:	30	31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27
Module	Function	Direction	PORTA							PORTC							PORTD							PORTF						
All			0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
TCB0	WO	output																												
USART0	RXD	input																												
	TXD	output																												
CLKCTRL	CLKOUT	output																												
Pins	GPIO	input																												
	GPIO	output																												

8. Configure the selected pins by navigating to *Project Resources > System > Pins*.
 - PC0:
 - Custom Name: SW0
 - Enable Pull-Up
 - ISC configuration: Sense Falling Edge
 - PF5:
 - Custom Name: LED0
 - PC1:
 - Enable Start High

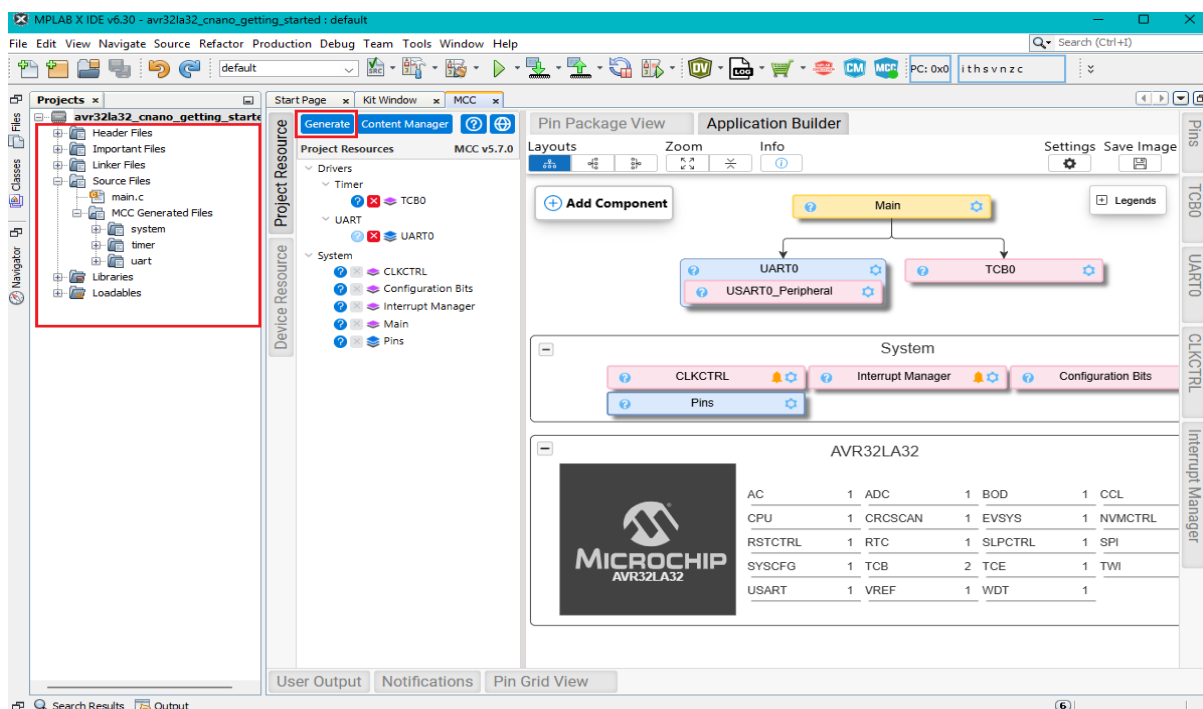
Figure 6-14. MPLAB® Code Configurator—Pins Configuration

Start Page x Kit Window x MCC x										Pins x TCB0 USART0 CLKCTRL Interrupt Manager									
Project Resource										Pin x									
Project Resources MCC v5.7.0										Location Pin Name Module Function Direction Custom Name Start High Inverted I/O Pull-up Input/Sense Configuration [ISC]									
Drivers										8 PC2 USART0 RXD input IO_PC2 ☐ ☐ ☐ Interrupt dis...									
Timer										7 PC1 USART0 TXD output IO_PC1 ☒ ☐ ☐ Interrupt dis...									
UART										6 PC0 Pins GPIO input SW0 ☐ ☐ ☒ Sense Falling...									
System										25 PF5 Pins GPIO output LED0 ☐ Interrupt disabled but input buffer enabled									
CLKCTRL										Sense Both Edges									
Configuration Bits										Sense Rising Edge									
Interrupt Manager										Sense Falling Edge									
Main										Digital Input Buffer disabled									
Pins										Sense low Level									
User Output Notifications Pin Grid View																			

6.3. Generate the Code

After configuring the peripherals as shown in the following figure, click the **Generate** button in the top-left corner of the **MCC** tab.

Figure 6-15. MPLAB® Code Configurator—Generate the Configuration



6.4. Add the Application Logic

To develop and run the application, open the project's `main.c` file and perform the following steps:

1. Include the following files in the `main.c` file:

```
#include <util/delay.h>
```

2. Add the following definitions, which specify the value to be loaded into the timer's CCMP register to generate the compare interrupt.

```
// Value to load into the TCB0 CCMP register for the 1s timer interrupt; CCMP = (F_CPU /
// Prescaler) * Desired_Time
#define TIMER_PERIOD_1S (19530U)
// Value to load into the TCB0 CCMP register for the 200 ms timer interrupt; CCMP =
// (F_CPU / Prescaler) * Desired_Time
#define TIMER_PERIOD_200MS (3906U)
// Value to clear the TCB0 counter
#define TIMER_COUNTER_CLEAR (0x00)
```

3. Add the following global variables, to track the state of the LED blink interval and the switch press.

```
static bool blinkIntervalState = false;
static volatile bool isSwitchPressed = false;
```

4. Add the callback functions for the timer compare and switch press-detection interrupts.

```
void DetectSwitchPress(void)
{
    // Debounce delay
    delay_ms(20);
    if (! (SW0_GetValue()))
    {
```

```

        isSwitchPressed = true;
    }

}

void ToggleLED(void)
{
    LED0_Toggle();
}

```

5. Add the following functions, which are used to alternate the LED blinking interval.

The `TCB0_Reset()` function clears the timer counter and interrupt registers so that the timer starts from a known state when alternating the timer compare values.

The `VaryLEDBlinkTime()` function varies the LED blink interval by writing the corresponding value to the CCMP register of TCB0 for each switch press, depending on the previous LED blink interval state, and enables the timer interrupt.

```

static void TCB0_Reset(void)
{
    TCB0_ClearCaptInterruptFlag();
    TCB0_Write(TIMER_COUNTER_CLEAR);
    TCB0_DisableCaptInterrupt();
}

static void VaryLEDBlinkTime(void)
{
    LED0_SetLow();
    // Toggles the state of the LED blinking speed
    blinkIntervalState = !blinkIntervalState;
    // Disables the TCB0 interrupt to prevent ISR execution during the CCMP update and
    // resets the timer counter
    TCB0_Reset();
    // Updates the timer compare value based on the current blink interval state
    if (blinkIntervalState == false)
    {
        TCB0_CaptureCompareWrite(TIMER_PERIOD_1S);
        printf("LED blink interval changed to 2s\r\n");
    }
    else
    {
        TCB0_CaptureCompareWrite(TIMER_PERIOD_200MS);
        printf("LED blink interval changed to 400 ms\r\n");
    }
    TCB0_EnableCaptInterrupt();
}

```

6. Add the main function to initialize the peripherals, set up callback functions, and enable global interrupts. The function continuously monitors the switch input and alternates the blinking interval whenever the switch is pressed.

```

int main(void)
{
    SYSTEM_Initialize();

    // Sets the interrupt handler for the timer interrupt. LED0 toggles when the timer
    // interrupt occurs.
    TCB0_CaptureCallbackRegister(ToggleLED);

    // Sets the interrupt handler for the switch press
    SW0_SetInterruptHandler(DetectSwitchPress);

    printf("*****PRINTING LED BLINK INTERVAL TIME*****\r\n");
    printf("LED blink interval is 2s \r\n");
    while (1)
    {
        if (isSwitchPressed)
        {
            VaryLEDBlinkTime();
            isSwitchPressed = false;
        }
    }
}

```

```
    return 0;
}
```

6.5. Build and Program the Application

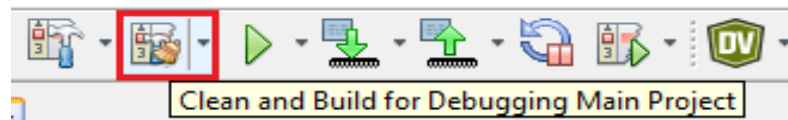
Follow these steps to build and program the application:

1. The AVR32LA32 Curiosity Nano evaluation kit supports programming and debugging using an on-board debugger. Connect the USB Type-C® cable to the AVR32LA32 Curiosity Nano evaluation kit to power and debug the Curiosity Nano board.
2. Set `avr32la_cnano_getting_started` as the main project by right-clicking the project and selecting Set as Main Project. Then, go to Project Properties and select the latest compiler version (v.3.10 or later).

Note: Disregard Steps 1 and 2 if they have already been completed.

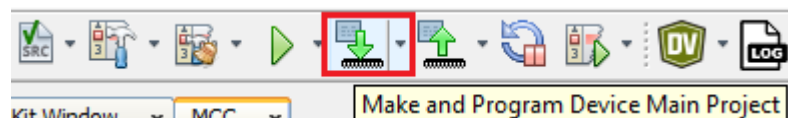
3. Clean and build the project by clicking the icon highlighted in the figure below.

Figure 6-16. Clean and Build the Main Project



4. Program the application by clicking the highlighted icon.

Figure 6-17. Make and Program Device

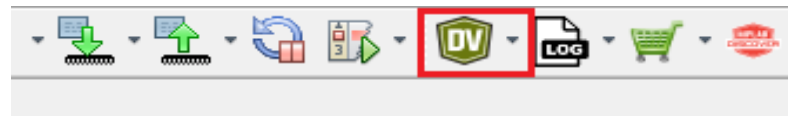


6.6. Observe the Output

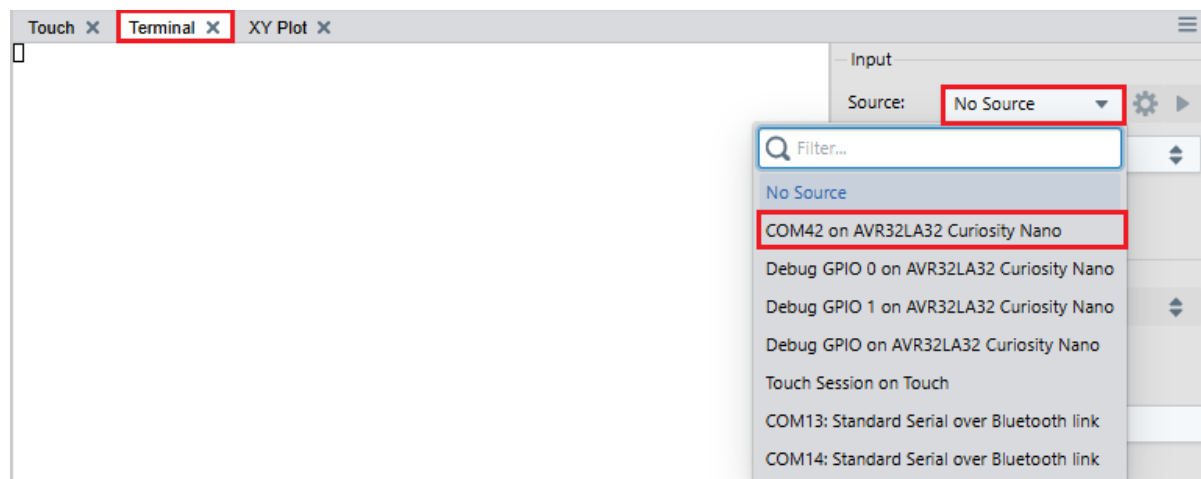
To observe the output on the board and in the serial terminal, follow these steps:

1. To launch MPLAB® Data Visualizer, click the icon shown in the below image.

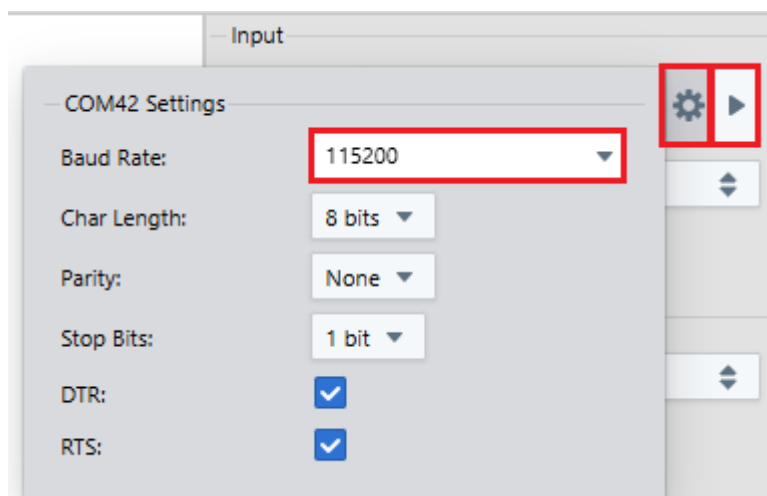
Figure 6-18. Launch MPLAB Data Visualizer



2. Select the COM port corresponding to the Curiosity Nano from the Source drop-down menu in the terminal window.

Figure 6-19. Serial Communication Port Selection

3. Set the baud rate to 115200 by clicking the settings icon next to the COM port.
4. Click the play icon to view the UART Communication in the terminal window.

Figure 6-20. Setting the Baud Rate

5. The on-board LED of the AVR32LA32 Curiosity Nano evaluation kit toggles at a default interval of two seconds. The LED blink interval is displayed in the serial terminal.
 - Press the SW switch on the AVR32LA32 Curiosity Nano evaluation kit to change the blink interval to 400 ms
 - Each press of the on-board switch (SW0) of the Curiosity Nano evaluation kit alternates the blink interval between 400 ms and two seconds

Figure 6-21. Output Observed on Terminal

```
*****PRINTING LED BLINK INTERVAL TIME*****
LED blink interval is 2s
LED blink interval changed to 400 ms
LED blink interval changed to 2s
LED blink interval changed to 400 ms
LED blink interval changed to 2s
LED blink interval changed to 400 ms
```

Note: The application discussed above is implemented and available on [MPLAB® Discover](#).

7. What's Next

From the previous sections, you have learned where to find information about Microchip products and how to get started by downloading the required software and obtaining the necessary hardware tools. For additional details about related AVR® LA products and the IDE, refer to the links provided below.

7.1. Software

- [VS Code® with extensions](#)
- [MPLAB® X IDE](#)
- [MPLAB® Code Configurator](#)
- [XC8 Compiler](#)

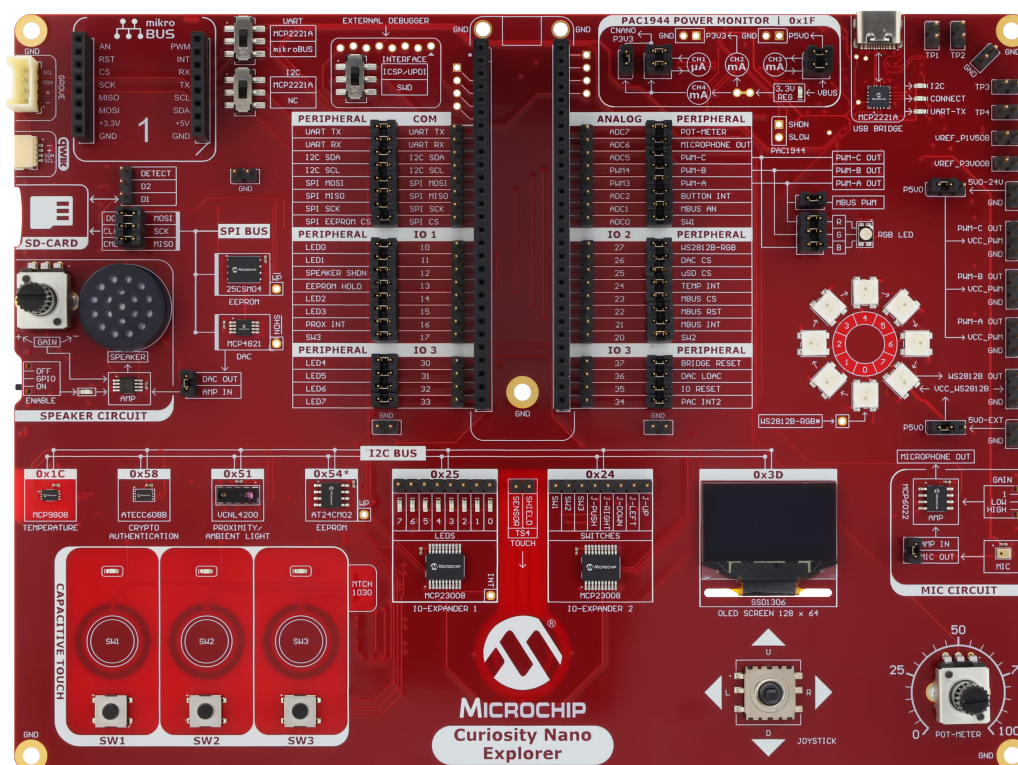
7.2. Firmware

- [MPLAB® Discover Examples](#)
- [GitHub Examples](#)

7.3. Evaluation Kits

Microchip Curiosity Nano Explorer Evaluation Kit

The Microchip Curiosity Nano Explorer lives up to its name by providing extensive on-board features that allow users to explore and experiment with the microcontroller peripherals of their Curiosity Nano development board. It also serves as an ideal platform for becoming familiar with Microchip's software offerings, including [MPLAB® Code Configurator \(MCC\)](#).



Web Page: <http://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/EV58G97A>

Get the Kit: <http://www.microchipdirect.com/dev-tools/EV58G97A>

For additional information, refer to the [Curiosity Nano Explorer User Guide](#).

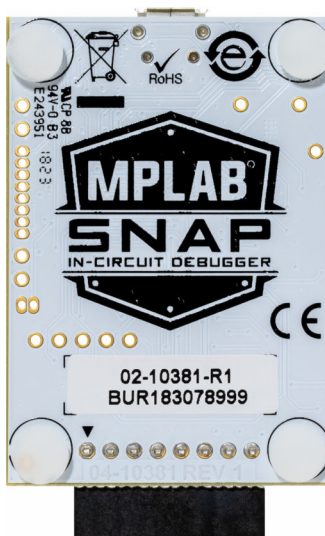
7.4. Recommended Programming/Debugging Tools

Microchip offers a range of programmers, debuggers and extensions to support all device architectures. All solutions, including in-circuit debuggers and programmers, are USB-powered and fully integrated into their respective environments. The MPLAB® Snap In-Circuit Debugger and MPLAB® PICKit® 5 In-Circuit Debuggers are economical choices for basic debugging functionality. Choosing a programmer or debugger depends on the type of development work being performed. The MPLAB® Snap is a low-cost, entry-level programmer/debugger; while the PICKit® 5 is a midrange option with more features than the Snap. The MPLAB® ICD 5 In-Circuit Debugger provides advanced debugging and hardware features suitable for most users and is intended for engineers developing commercial or industrial products.

MPLAB® SNAP In-Circuit Debugger

- [Product Page](#)
- [Buy Here](#)
- [User Guide](#)
- [Microchip Developer Help](#)

Figure 7-1. MPLAB® SNAP In-Circuit Debugger



MPLAB® PICKit® 5 In-Circuit Debugger

- [Product Page](#)
- [Buy Here](#)
- [User Guide](#)
- [Microchip Developer Help](#)

Figure 7-2. PICKit[®] 5 In-Circuit Debugger



MPLAB[®] ICD 5 In-Circuit Debugger

- [Product Page](#)
- [Buy Here](#)
- [User Guide](#)
- [Microchip Developer Help](#)

Figure 7-3. ICD 5 In-Circuit Debugger



ATMEL-ICE Debugger

- [Product Page](#)
- [Buy Here](#)
- [User Guide](#)
- [Microchip Developer Help](#)

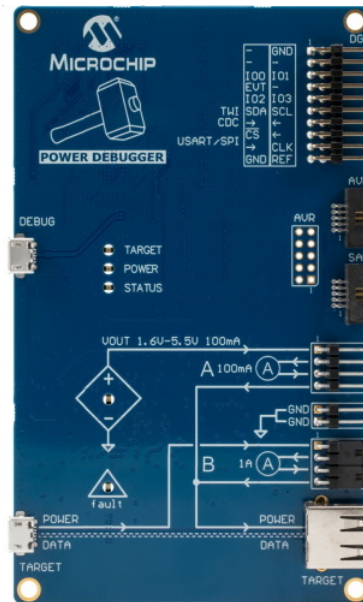
Figure 7-4. ATMEL ICE



Power Debugger (ATPOWERDEBUGGER)

- [Product Page](#)
- [Buy Here](#)
- [User Guide](#)
- [Microchip Developer Help](#)

Figure 7-5. Power Debugger



8. References

- AVR32LA32 Curiosity Nano Hardware User Guide ([DS50004120](#))
- For additional information on MCC Melody, refer to the [Microchip website](#)
- Developer Help: [Get Started for MPLAB® X IDE Users New to Visual Studio Code \(VS Code®\)](#)
- Microchip University Course: [Introduction to MPLAB® X IDE](#)
- Microchip University Course: [Overview of the MPLAB® Code Configurator \(MCC\) Content Manager \(CM\)](#)
- Microchip University Course: [MPLAB® Data Visualizer](#) (for Curiosity Nano)
- Additional online technical documentation concerning various products: [onlinedocs.microchip.com](#)
- Microchip Technical Support: [support.microchip.com/s/](#)
- For more information on various applications, refer to [AVR® LA on MPLAB® Discover](#)
- For the example application, refer to the [Getting Started with the AVR32LA32 Microcontroller Using the Curiosity Nano and MCC Melody](#)

9. Revision History

Revision	Date	Description
A	08/2026	Initial revision

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>.

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

[AVR16LA14](#), [AVR16LA20](#), [AVR16LA28](#), [AVR16LA32](#), [AVR32LA14](#), [AVR32LA20](#), [AVR32LA28](#), [AVR32LA32](#)