Code Generation with VS Code NuCodeGen

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example demonstrates how to use the NuCodeGen extension in VS Code to generate M55M1 code.
BSP Version	M55M1_Series_BSP_CMSIS_V3.01.003
Hardware	NuMaker-X-M55M1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This document explains how to use NuCodeGen, a VS Code extension, to automatically generate source code using large language models (LLMs). This approach is designed to enhance development efficiency and reduce manual coding costs. The content first demonstrates NuCodeGen's basic operations through a UART example, followed by a full code-generation workflow illustrated with an I^2C example. Both examples include the chat files and the final generated code to clearly present each stage of the code-generation process.

This example utilizes VS Code as the development environment, integrated with the Nuvoton NuMicro Cortex-M Pack development toolset (which includes NuCodeGen). Prior to implementation, please ensure that the toolset is properly installed within VS Code.

While this document uses the NuMaker-X-M55M1 as an example, the workflows and methods covered in this document apply to the entire Nuvoton NuMicro family, such as the M0, M23, and M4 series.

More examples can be found in the NuCodeGen sample prompts.

1.1 Principle

NuCodeGen is an AI agent built on GitHub Copilot and provides the following core capabilities:

- **Natural Language to Code:** Users can describe their development requirements using natural language prompts, and NuCodeGen automatically generates compilable code, improving development efficiency.
- **Instantaneous Technical Retrieval:** NuCodeGen provides real-time answers to technical inquiries—such as peripheral IP overviews, API references, and usage instructions—eliminating the need for manual searches through standard driver files.
- **Automated Error Resolution:** NuCodeGen analyzes error messages generated during compilation to diagnose issues and directly implement code fixes, significantly streamlining the debugging process.
- **Integrated AI-Driven Development Loop:** By combining GitHub Copilot with NuMicro development tools, NuCodeGen enables a unified 'Build, Load, and Debug' workflow. Furthermore, it supports autonomous command execution via AI, establishing a fully closed-loop development environment.

By integrating code generation, technical inquiry, and error-handling capabilities, NuCodeGen significantly enhances the development efficiency of NuMicro projects while effectively minimizing manual intervention.

1.2 Extension Configuration

1.2.1 Installing the Extension

The Nuvoton NuMicro Cortex-M Pack development toolset (which includes NuCodeGen) must be installed and its resources loaded through VS Code's online mechanisms. If you are

operating in a restricted network environment (such as behind a corporate firewall or proxy), please ensure that network access permissions include at least the following core domains to maintain basic extension functionality:

- **marketplace.visualstudio.com:** Used to search for extensions and retrieve version information.
- ***.gallerycdn.vsassets.io:** Used to download extension installation packages (*.vsix*).
- ***.vscode-cdn.net:** Used to load VS Code–related CDN resources.

All connections must be made over HTTPS (TCP port 443).

Actual connectivity requirements may vary depending on the VS Code version or environment configuration. For more comprehensive network requirements, as well as proxy and firewall configuration details, please refer to the official VS Code documentation: VS Code Documentation – Network Setup.

1.2.2 Activating the Extension

By default, NuCodeGen is not automatically enabled in VS Code. It only takes effect in CMSIS-Csolution projects (for example, BSP projects). In the case of newly created projects, NuCodeGen-related features will not be enabled until the CMSIS-Csolution initialization process is completed.

1.3 Introduction to Supported BSP Structure

Users can download the BSP from OpenNuvoton and open the project in VS Code. For instructions on how to open the project, please refer to the Example Code Execution chapter. Note that it is important to follow the procedure described in the Example Code Execution chapter and open the *VSCode* folder that is at the same directory level as *main.c*.

NuCodeGen supports BSPs with the structure shown in Figure 1-1. At runtime, the system automatically performs an upward search from the current project directory to locate Standard Driver Library files within the *Library* folder. Therefore, when any project belonging to a BSP is opened, the system will automatically use the *Library* folder within the BSP directory.

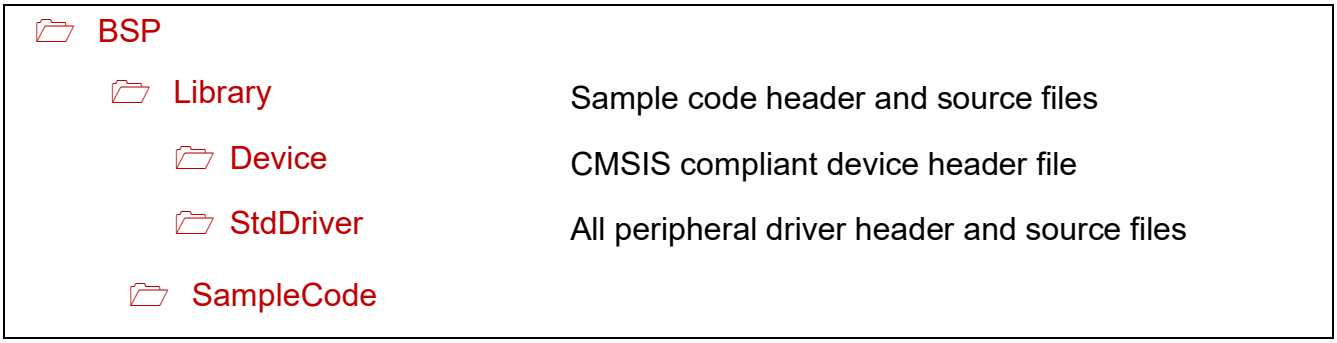

Figure 1-1 BSP Structure Supported by NuCodeGen

Since NuCodeGen operates within the local BSP environment, please download and update the BSP to the latest version from OpenNuvoton before using it if a new BSP version is

released.

1.4 Operation Overview

To use the NuCodeGen extension, select the custom agent “NuCodeGen” in the GitHub Copilot Chat interface, as shown in Figure 1-3.

In the GitHub Copilot Chat interface, please click the Agent icon, as shown in Figure 1-2, and select the custom agent “NuCodeGen” to use the NuCodeGen extension, as illustrated in Figure 1-3.

The examples in this chapter are demonstrated using the Claude Haiku 4.5 model.

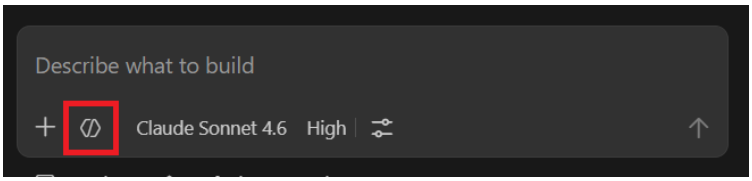

Figure 1-2 Agent Icon

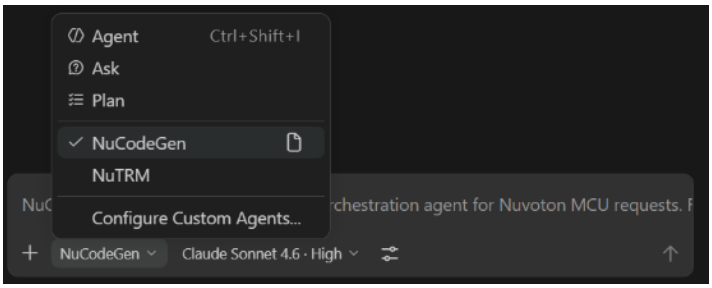

Figure 1-3 NuCodeGen Usage

1.4.1 Prompt Generation

The quality of the prompt is a critical determinant of the generated code's accuracy. If the prompt is too brief, important configuration details may be omitted, potentially leading to incorrect settings, runtime issues, and increased debugging time.

Therefore, NuCodeGen provides the command **/<peripheral_name>-settings**, as shown in Figure 1-4, which lists the configuration options related to a specific peripheral. When the user enters this command, NuCodeGen automatically generates a configuration list and example prompts for that peripheral based on the target microcontroller’s standard driver library, as illustrated in Figure 1-5 and Figure 1-6. This helps developers customize more complete and accurate prompts according to their requirements.

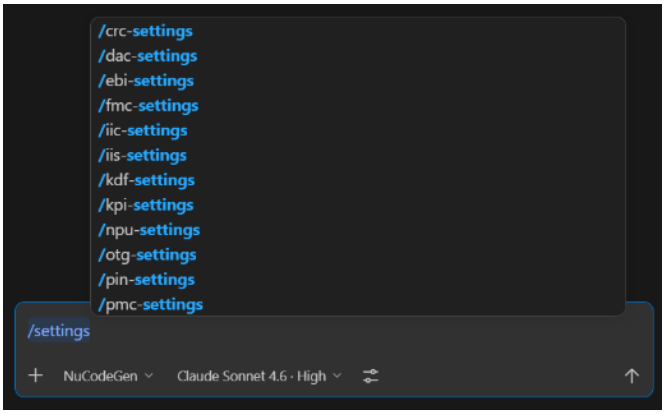

Figure 1-4 **/<peripheral_name>-settings** Command

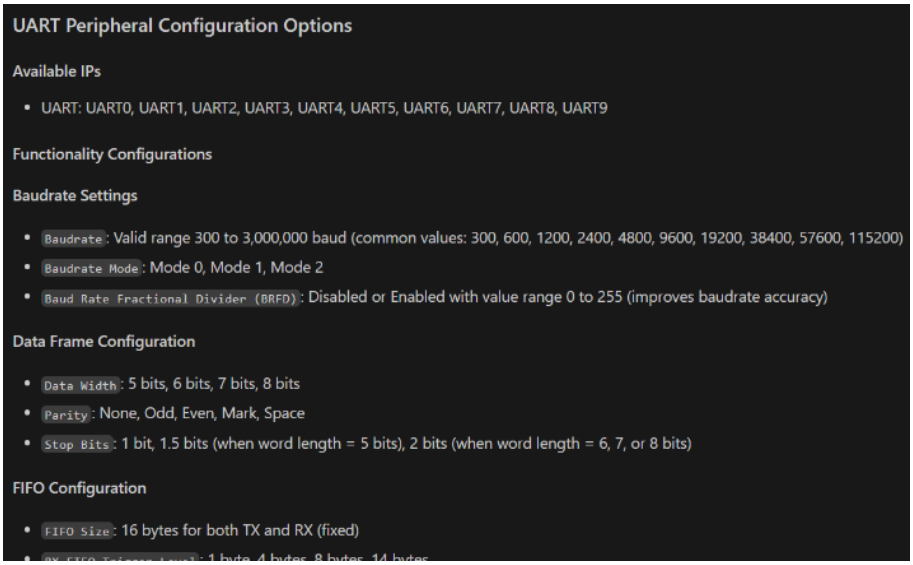

Figure 1-5 UART Configuration Options

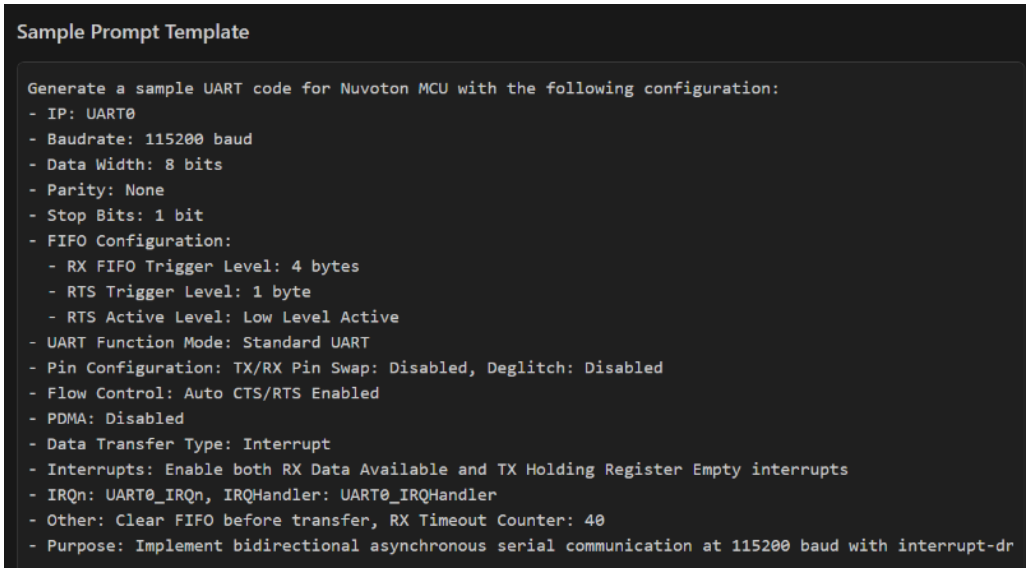

Figure 1-6 UART Example Prompt

1.4.2 Code Generation

Once the prompt is generated, it can be used to produce the corresponding source code. NuCodeGen provides a set of tools for the AI, allowing it to autonomously determine and select the appropriate tools to retrieve required information—such as function names or macro definitions—from the standard driver libraries (as described in the Introduction to Supported BSP Structure section). Based on the prompt content, the AI then generates the code. After code generation, NuCodeGen automatically performs compilation and applies necessary corrections according to the compilation results until the process succeeds.

Through the two steps described above, developers can iteratively generate the required code until all peripheral, clock, and pin configurations are finalized. Figure 1-7 demonstrates the programming result of a UART loopback test program generated using the Claude Haiku 4.5 model, while Figure 1-8 illustrates the complete code-generation workflow. In this example, the generated code is programmed directly after generation, without any additional code modifications.

For more detailed information and usage guidelines of NuCodeGen, see the NuCodeGen Overview section.

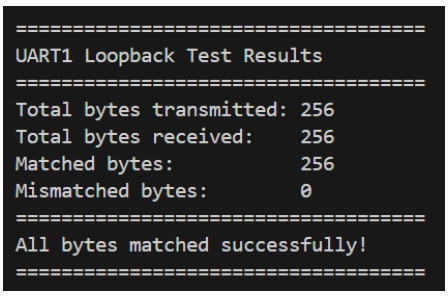

Figure 1-7 UART Loopback Test Programming Result

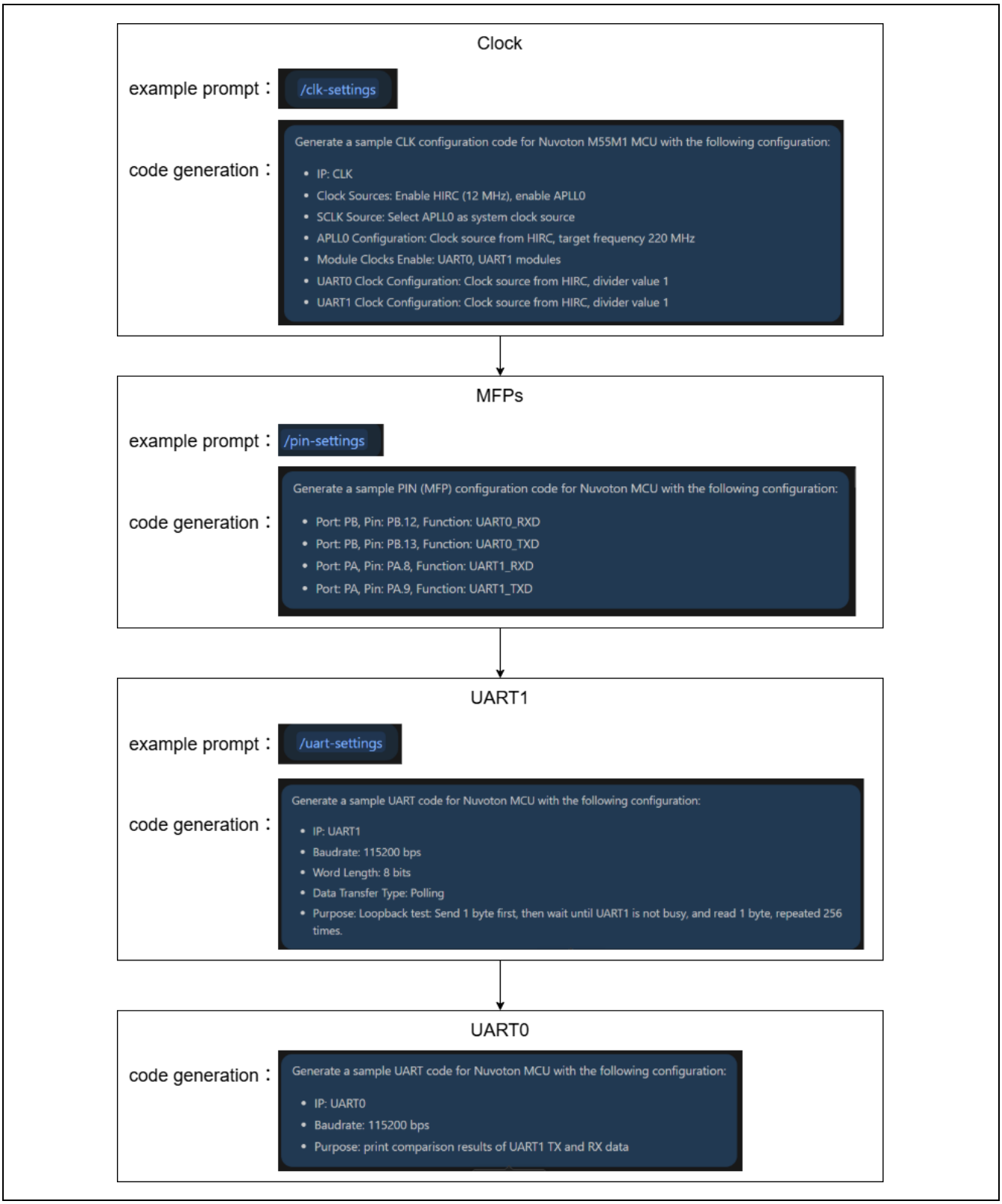

Figure 1-8 UART Loopback Test Code Generation Flow

1.5 Model Selection

The monthly quota for premium requests varies across GitHub Copilot plans; for instance, the Free plan currently includes 50 requests per month, while the Pro plan offers 300. Developers are advised to monitor their usage accordingly.

In the free plan, all supported models currently consume message credits at the same rate, as shown in Figure 1-9. For this reason, when generating peripheral, clock, and pin-related code, it is recommended to prioritize using Claude Haiku 4.5.

In addition, when sending requests using the NuCodeGen agent, premium requests are counted in the same way as those for built-in GitHub Copilot agents. Currently, each request consumes one premium request. If Copilot's counting mechanism is adjusted in the future, NuCodeGen will be affected accordingly.

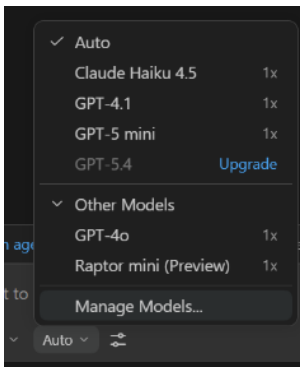

Figure 1-9 Models Supported in the Free Plan

For developers on paid GitHub Copilot plans, models should be selected based on task complexity. For basic configurations—such as peripheral clocks, pin setups, or simple UART0 debug messaging—lightweight models like Grok, Gemini 3 Flash, or Claude Haiku 4.5 are ideal. For standard peripheral function generation, models that premium request is x1 (e.g., Claude Sonnet 4.6, Gemini 3.1 Pro, or GPT 5.3 Cortex) are recommended. For highly complex features, use models with higher premium-request levels to ensure more accurate and precisely tailored code.

You can check the GitHub Copilot usage status in the VS Code status bar, as shown in Figure 1-10. If updated or higher-performance models are released in the future, they can be adopted based on specific requirements.

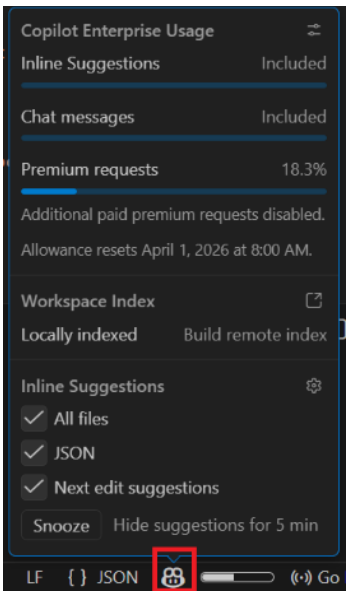

Figure 1-10 GitHub Copilot usage status

The pricing of GitHub Copilot plans, the number of available premium requests, and the set of models supported under each plan may be adjusted at any time. For the latest information, please refer to the official documentation: GitHub Copilot Plans & Pricing.

2. Code Description

This section uses an I²C example to illustrate the complete development workflow, along with the code generated at each prompt stage. The model used for this example is Claude Sonnet 4.6. In the example, I2C0 is configured as the master and I2C1 as the slave, implementing two tests.

The first test is for the master to transmit 128 bytes of data to the slave at address 0x16, and the second test is for the master to receive 128 bytes of data from the slave at address 0x16.

The entire code generation process is shown in Figure 2-1. It is recommended to configure the required clocks and MFPs in advance to avoid mismatches between the AI-generated settings and the intended clock or pin configuration when generating I²C or UART functional code.

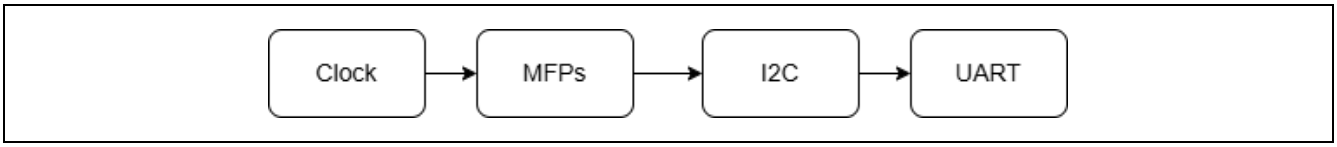

Figure 2-1 I²C Example Workflow

Finally, Figure 2-2 shows the result after directly programming the generated code, without any additional code modifications.

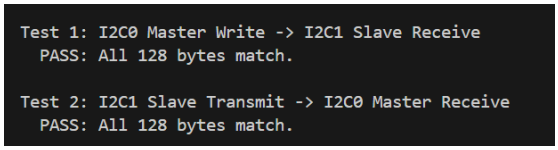

Figure 2-2 I²C Example Programming Result

2.1 Clock Code Generation

First, generate the system clock–related code based on the prompt shown in Figure 2-3.

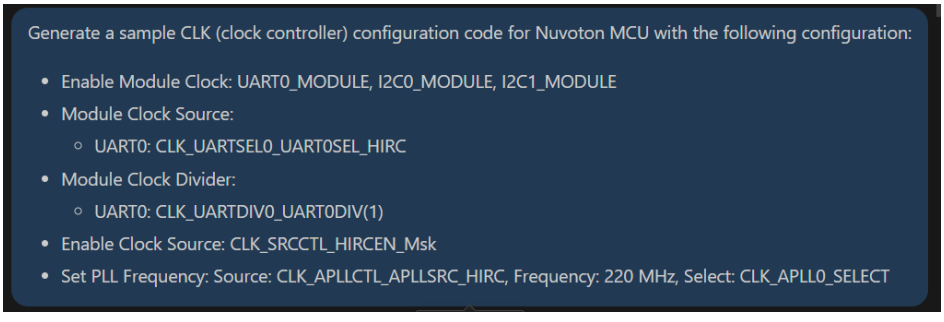

Figure 2-3 Peripheral Clock Configuration Prompt

```
void SYS_Init(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Enable HIRC clock source */
    CLK_EnableXtalRC(CLK_SRCCTL_HIRCEN_Msk);
    /* Wait for HIRC clock source stable */
    CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk);
    /* Set APLL0 clock source to HIRC and set frequency to 220 MHz */
    CLK_EnableAPLL(CLK_APLLCTL_APLLSRC_HIRC, 220000000UL, CLK_APLL0_SELECT);
    /* Wait for APLL0 clock source stable */
    CLK_WaitClockReady(CLK_STATUS_APLL0STB_Msk);
    /* Switch SCLK clock source to APLL0 */
    /* Power level and flash access cycle are set automatically */
    CLK_SetSCLK(CLK_SCLKSEL_SCLKSEL_APLL0);

    /* Update System Core Clock */
    SystemCoreClockUpdate();

    /* Set UART0 module clock source to HIRC and divider to 1 */
    CLK_SetModuleClock(UART0_MODULE, CLK_UARTSEL0_UART0SEL_HIRC,
CLK_UARTDIV0_UART0DIV(1));

    /* Enable UART0 module clock */
    CLK_EnableModuleClock(UART0_MODULE);

    /* Enable I2C0 module clock */
    CLK_EnableModuleClock(I2C0_MODULE);

    /* Enable I2C1 module clock */
    CLK_EnableModuleClock(I2C1_MODULE);

    /* Set PB.12 as UART0_RXD and PB.13 as UART0_TXD */
    /* Lock protected registers */
    SYS_LockReg();
}
```

2.2 Pin Code Generation

Next, configure the pins using the prompt shown in Figure 2-4.

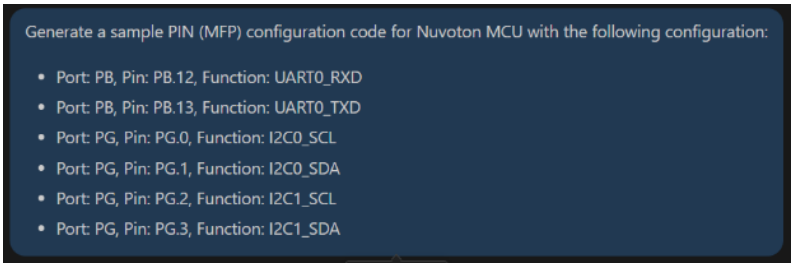

Figure 2-4 Pin Configuration Prompt

```
void SYS_Init(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();
    /* Set PB.12 as UART0_RXD and PB.13 as UART0_TXD */
    SYS->GPB_MFP3 = (SYS->GPB_MFP3 & ~(SYS_GPB_MFP3_PB12MFP_Msk |
SYS_GPB_MFP3_PB13MFP_Msk)) |
                    (SYS_GPB_MFP3_PB12MFP_UART0_RXD | SYS_GPB_MFP3_PB13MFP_UART0_TXD);

    /* Set PG.0 as I2C0_SCL, PG.1 as I2C0_SDA, PG.2 as I2C1_SCL, PG.3 as I2C1_SDA */
    SYS->GPG_MFP0 = (SYS->GPG_MFP0 & ~(SYS_GPG_MFP0_PG0MFP_Msk | SYS_GPG_MFP0_PG1MFP_Msk |
                                       SYS_GPG_MFP0_PG2MFP_Msk |
SYS_GPG_MFP0_PG3MFP_Msk)) |
                    (SYS_GPG_MFP0_PG0MFP_I2C0_SCL | SYS_GPG_MFP0_PG1MFP_I2C0_SDA |
                     SYS_GPG_MFP0_PG2MFP_I2C1_SCL | SYS_GPG_MFP0_PG3MFP_I2C1_SDA);

    /* Lock protected registers */
    SYS_LockReg();
}
```

2.3 I²C Code Generation

First, the I²C-related code is generated according to the prompt shown in Figure 2-5, covering the necessary configurations for I2C0 and I2C1, along with their corresponding function implementations.

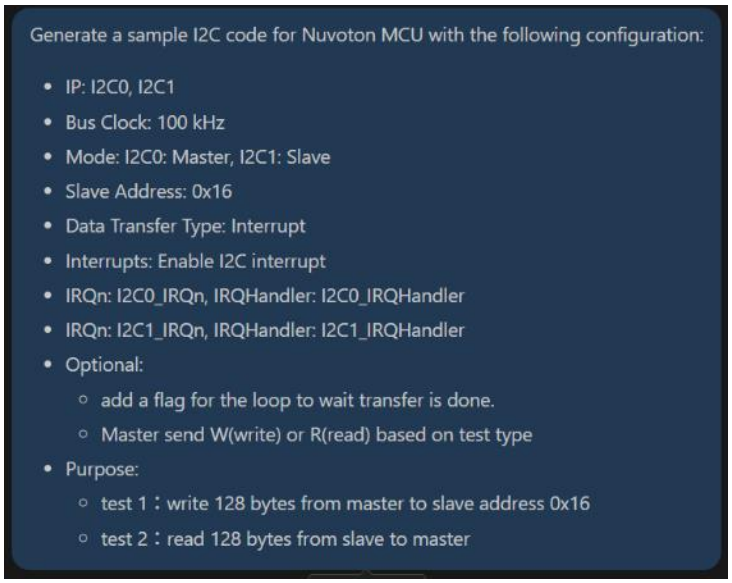

Figure 2-5 I²C Prompt

The code generated from this prompt includes:

- The macro definitions and variables used in the code.

```
/*---------------------------------------------------------------------------------------------*/
/* Defines                                                                                     */
/*---------------------------------------------------------------------------------------------*/
#define TRANSFER_SIZE   128
```

```
#define SLAVE_ADDRESS   0x16

/*---------------------------------------------------------------------------------------------*/
/* Global variables (volatile: shared between main and ISR)                                    */
/*---------------------------------------------------------------------------------------------*/
/* Test type: 0 = Master Write / Slave Receive, 1 = Master Read / Slave Transmit */
volatile uint8_t  g_u8TestType = 0;

/* Master TX/RX buffers and transfer state */
volatile uint8_t  g_au8MasterTxBuf[TRANSFER_SIZE];
volatile uint8_t  g_au8MasterRxBuf[TRANSFER_SIZE];
volatile uint32_t g_u32MasterDataLen = 0;
volatile uint8_t  g_u8MasterEndFlag  = 0;  /* Set by ISR when master transfer completes */

/* Slave TX/RX buffers and transfer state */
volatile uint8_t  g_au8SlaveTxBuf[TRANSFER_SIZE];
volatile uint8_t  g_au8SlaveRxBuf[TRANSFER_SIZE];
volatile uint32_t g_u32SlaveDataLen  = 0;
volatile uint8_t  g_u8SlaveEndFlag   = 0;  /* Set by ISR when slave transfer completes */
```

- I²C initialization

```
/*---------------------------------------------------------------------------------------------*/
/* I2C Initialization: I2C0 (Master) and I2C1 (Slave at address 0x16)                          */
/*---------------------------------------------------------------------------------------------*/
void I2C_Init(void)
{
    /* Open I2C0 with 100 kHz bus clock (Master) */
    I2C_Open(I2C0, 100000);

    /* Open I2C1 with 100 kHz bus clock (Slave) */
    I2C_Open(I2C1, 100000);

    /* Configure I2C1 slave address as 0x16, GC mode disabled */
    I2C_SetSlaveAddr(I2C1, 0, SLAVE_ADDRESS, I2C_GCMODE_DISABLE);

    /* Enable I2C0 interrupt and register NVIC */
    I2C_EnableInt(I2C0);
    NVIC_EnableIRQ(I2C0_IRQn);

    /* Enable I2C1 interrupt and register NVIC */
    I2C_EnableInt(I2C1);
    NVIC_EnableIRQ(I2C1_IRQn);

    /* Enter I2C1 (Slave) into non-addressed slave mode, ready to respond */
    I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
}
```

- The I²C interrupt handlers execute the appropriate actions for each master/slave state and update the required control signals.

```
/*---------------------------------------------------------------------------------------------*/
/* I2C0 IRQ Handler (Master)                                                                   */
/*---------------------------------------------------------------------------------------------*/
void I2C0_IRQHandler(void)
{
    uint32_t u32Status = I2C_GET_STATUS(I2C0);

    if (g_u8TestType == 0)
    {
```

```
        /* Test 1: Master Write 128 bytes to slave */
        switch (u32Status)
        {
            case 0x08:  /* START transmitted */
                I2C_SET_DATA(I2C0, (uint8_t)((SLAVE_ADDRESS << 1) | 0x00));  /* SLA+W */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            case 0x18:  /* SLA+W transmitted, ACK received */
                I2C_SET_DATA(I2C0, g_au8MasterTxBuf[g_u32MasterDataLen++]);
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            case 0x28:  /* Data byte transmitted, ACK received */
                if (g_u32MasterDataLen < TRANSFER_SIZE)
                {
                    I2C_SET_DATA(I2C0, g_au8MasterTxBuf[g_u32MasterDataLen++]);
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                }
                else
                {
                    /* All bytes sent, generate STOP */
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                    g_u8MasterEndFlag = 1;
                }
                break;

            case 0x20:  /* SLA+W transmitted, NACK received */
            case 0x30:  /* Data byte transmitted, NACK received */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                g_u8MasterEndFlag = 1;
                break;

            case 0x38:  /* Arbitration lost */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            default:
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                break;
        }
    }
    else
    {
        /* Test 2: Master Read 128 bytes from slave */
        switch (u32Status)
        {
            case 0x08:  /* START transmitted */
                I2C_SET_DATA(I2C0, (uint8_t)((SLAVE_ADDRESS << 1) | 0x01));  /* SLA+R */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            case 0x40:  /* SLA+R transmitted, ACK received */
                /* Set AA=1 to return ACK for the incoming bytes (more than 1 byte) */
                if (TRANSFER_SIZE > 1)
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI_AA);
```

```
                else
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            case 0x50:  /* Data byte received, ACK returned */
                g_au8MasterRxBuf[g_u32MasterDataLen++] = (uint8_t)I2C_GET_DATA(I2C0);
                /* Return NACK for the last byte (index TRANSFER_SIZE-1) */
                if (g_u32MasterDataLen < (TRANSFER_SIZE - 1))
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI_AA);
                else
                    I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);  /* Next byte will be NACK'd */
                break;

            case 0x58:  /* Data byte received, NACK returned (last byte) */
                g_au8MasterRxBuf[g_u32MasterDataLen++] = (uint8_t)I2C_GET_DATA(I2C0);
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                g_u8MasterEndFlag = 1;
                break;

            case 0x48:  /* SLA+R transmitted, NACK received */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                g_u8MasterEndFlag = 1;
                break;

            case 0x38:  /* Arbitration lost */
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_SI);
                break;

            default:
                I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STO_SI);
                break;
        }
    }

    /* M55M1: Read status register again to force write-clear propagation through the
interconnect */
    u32Status = I2C_GET_STATUS(I2C0);
}

/*---------------------------------------------------------------------------------------*/
/* I2C1 IRQ Handler (Slave)                                                              */
/*---------------------------------------------------------------------------------------*/
void I2C1_IRQHandler(void)
{
    uint32_t u32Status = I2C_GET_STATUS(I2C1);

    switch (u32Status)
    {
        case 0x60:  /* Own SLA+W received, ACK returned */
            g_u32SlaveDataLen = 0;
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            break;

        case 0x80:  /* Data received, ACK returned */
            g_au8SlaveRxBuf[g_u32SlaveDataLen++] = (uint8_t)I2C_GET_DATA(I2C1);
            if (g_u32SlaveDataLen < TRANSFER_SIZE)
                I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
```

```
            else
                I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI);
            break;

        case 0xA0:  /* STOP or Repeated START received */
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            if (g_u8TestType == 0)
                g_u8SlaveEndFlag = 1;  /* Test 1: slave receive complete */
            break;

        case 0xA8:  /* Own SLA+R received, ACK returned */
            g_u32SlaveDataLen = 0;
            I2C_SET_DATA(I2C1, g_au8SlaveTxBuf[g_u32SlaveDataLen++]);
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            break;

        case 0xB8:  /* Data transmitted, ACK received (master wants more) */
            if (g_u32SlaveDataLen < TRANSFER_SIZE)
            {
                I2C_SET_DATA(I2C1, g_au8SlaveTxBuf[g_u32SlaveDataLen++]);
                I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            }
            else
            {
                /* All bytes loaded; master controls end via NACK */
                I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            }
            break;

        case 0xC0:  /* Data transmitted, NACK received (master done reading) */
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            g_u8SlaveEndFlag = 1;
            break;

        case 0xC8:  /* Last data transmitted (AA=0), ACK received */
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            g_u8SlaveEndFlag = 1;
            break;

        default:
            I2C_SET_CONTROL_REG(I2C1, I2C_CTL_SI_AA);
            break;
    }

    /* M55M1: Read status register again to force write-clear propagation through the
interconnect */
    u32Status = I2C_GET_STATUS(I2C1);
}
```

- Two test cases

```
/* Initialize I2C0 (Master) and I2C1 (Slave) */
    I2C_Init();
/*---------------------------------------------------------------------*/
/* Test 1: I2C0 Master writes 128 bytes to I2C1 Slave (addr 0x16)      */
/*---------------------------------------------------------------------*/
/* Prepare Master TX buffer with incrementing data */
for (i = 0; i < TRANSFER_SIZE; i++)
```

```
    g_au8MasterTxBuf[i] = (uint8_t)i;

/* Reset transfer state for Test 1 */
g_u8TestType       = 0;
g_u32MasterDataLen = 0;
g_u32SlaveDataLen  = 0;
g_u8MasterEndFlag  = 0;
g_u8SlaveEndFlag   = 0;

/* Master sends START to initiate write transfer */
I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STA);

/* Wait for Master to complete */
while (g_u8MasterEndFlag == 0);
/* Wait for Slave to complete */
while (g_u8SlaveEndFlag  == 0);

/*---------------------------------------------------------------------------*/
/* Test 2: I2C0 Master reads 128 bytes from I2C1 Slave (addr 0x16)           */
/*---------------------------------------------------------------------------*/
/* Prepare Slave TX buffer with offset data */
for (i = 0; i < TRANSFER_SIZE; i++)
    g_au8SlaveTxBuf[i] = (uint8_t)(i + 0x80);

/* Reset transfer state for Test 2 */
g_u8TestType       = 1;
g_u32MasterDataLen = 0;
g_u32SlaveDataLen  = 0;
g_u8MasterEndFlag  = 0;
g_u8SlaveEndFlag   = 0;

/* Master sends START to initiate read transfer */
I2C_SET_CONTROL_REG(I2C0, I2C_CTL_STA);

/* Wait for Master to complete */
while (g_u8MasterEndFlag == 0);
/* Wait for Slave to complete */
while (g_u8SlaveEndFlag  == 0);
```

2.4 UART Code Generation

Finally, enable UART0 to output the execution results. The corresponding prompt is shown in Figure 2-6.

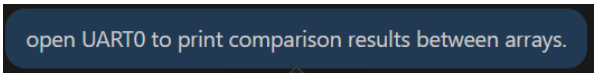

Figure 2-6 UART Prompt

```
int main(void)
{
    uint32_t i;

    /* Open UART0 at 115200 bps for standard I/O retargeting (printf) */
    UART_Open(UART0, 115200);

    /*---------------------------------------------------------------------------*/
```

```
    /* Print comparison results via UART0                                     */
    /*------------------------------------------------------------------------*/
    /* Test 1: g_au8MasterTxBuf vs g_au8SlaveRxBuf */
    printf("\r\nTest 1: I2C0 Master Write -> I2C1 Slave Receive\r\n");
    for (i = 0; i < TRANSFER_SIZE; i++)
    {
        if (g_au8MasterTxBuf[i] != g_au8SlaveRxBuf[i])
        {
            printf("  FAIL at byte[%u]: MasterTx=0x%02X, SlaveRx=0x%02X\r\n",
                   (unsigned int)i, g_au8MasterTxBuf[i], g_au8SlaveRxBuf[i]);
            break;
        }
    }
    if (i == TRANSFER_SIZE)
        printf("  PASS: All %d bytes match.\r\n", TRANSFER_SIZE);

    /* Test 2: g_au8SlaveTxBuf vs g_au8MasterRxBuf */
    printf("\r\nTest 2: I2C1 Slave Transmit -> I2C0 Master Receive\r\n");
    for (i = 0; i < TRANSFER_SIZE; i++)
    {
        if (g_au8SlaveTxBuf[i] != g_au8MasterRxBuf[i])
        {
            printf("  FAIL at byte[%u]: SlaveTx=0x%02X, MasterRx=0x%02X\r\n",
                   (unsigned int)i, g_au8SlaveTxBuf[i], g_au8MasterRxBuf[i]);
            break;
        }
    }
    if (i == TRANSFER_SIZE)
        printf("  PASS: All %d bytes match.\r\n", TRANSFER_SIZE);

    while (1);
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
- M55M1_Series_BSP_CMSIS_V3.01.003
- IDE version
- VS Code 1.110.0

3.2 Hardware Requirements

- Circuit components
- NuMaker-X-M55M1
- Pin Connect
- Connect the I2C0 SCL (PG.0) pin to the I2C1 SCL (PG.2).
- Connect the I2C0 SDA (PG.1) pin to the I2C1 SDA (PG.3).
- Connect the Nu-Link2-Me USB to the PC to display the execution results of the example code.

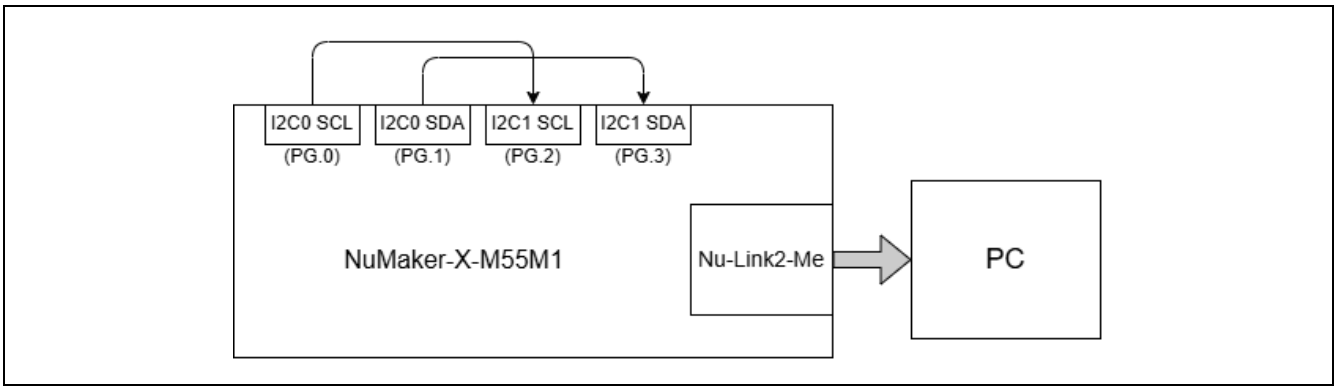

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_Code_Generation_with_VSCode_NuCodeGen_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
ChatJson	Chat histories for the example code
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Chat JSON Importing

1. In VS Code, press **Ctrl + Shift + P**.
2. Search for “Chat: Import Chat”, as shown in Figure 5-1, and select the JSON file you want to import.

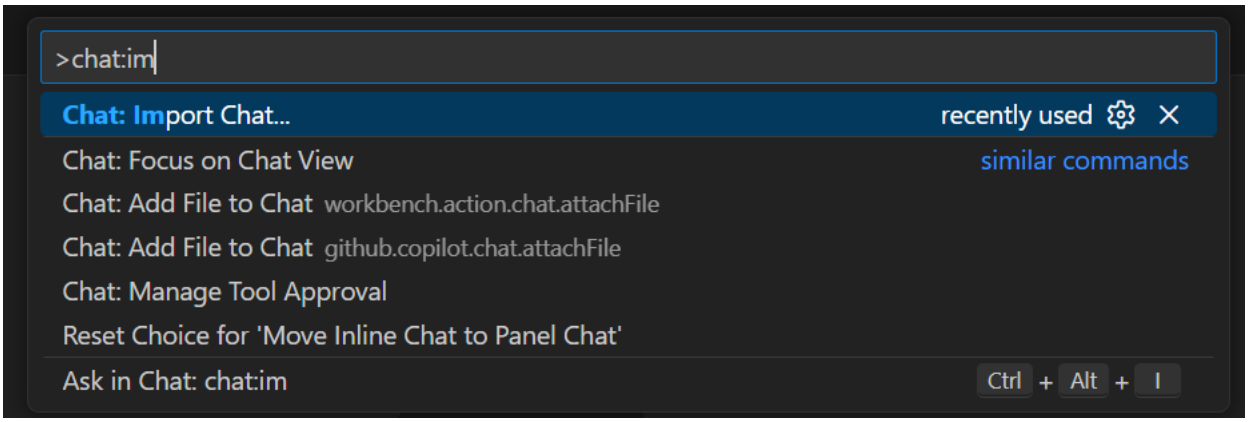

Figure 5-1 Import JSON File

6. Example Code Execution

1. Based on the Directory Information section, open the *VSCode* folder located in the *ExampleCode* directory within VS Code.
2. Click the CMSIS icon in the VS Code Activity Bar to open the CMSIS view.
    - Build
    - Download
    - Start/Stop debug session
3. Enter debug mode.
    - Run

7. Revision History

Date	Revision	Description
2026.03.04	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.