Emulating MDIO via M252 PSIO

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses M252 PSIO to emulate MDIO waveform for controlling Ethernet PHY (IP101GR).
BSP Version	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.006
Hardware	NuMaker-M252SD V1.2

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 example code uses M252 PSIO (Programmable Serial I/O) to emulate serial management interface (MDC/MDIO) waveform for controlling Ethernet PHY. PSIO provides an easy way to receive and transmit various serial transmissions and control the pin output high or low at a specific time and how long the voltage level should be maintained by configuring the time slot, check point and the voltage level.

The MDIO interface is implemented by two signals, including MDC and MDIO. MDIO is the bi-directional data line and MDC is clock line.

Most of the NuMicro Family does not support MDIO master interface for PHY device configuration. This example code provides the solution that can be widely used in the NuMicro Family for accessing Ethernet PHY.

This example code requires additional external hardware modules, including an Ethernet PHY. The part number of Ethernet PHY is IP101GR made by IC PLUS Corporation.

1.1 Principle

1.1.1 MDIO Data Format

The MDIO data format is shown in Figure 1-1 and MDIO data field description is shown in Table 1-1.

Preamble	ST	OP	PHY_ADR	REG_ADR	TA	DATA
32-bit	2-bit	2-bit	5-bit	5-bit	2-bit	16-bit

Figure 1-1 MDIO Data Format

Field	Width	Description
Preamble	32-bit	32 continuous bits of value 1.
ST	2-bit	Start of frame is 2'01.
OP	2-bit	Op code. 01 = Write operation 10 = Read operation
PHY_ADR	5-bit	PHY address.
REG_ADR	5-bit	Register address.
TA	2-bit	Turnaround time.
DATA	16-bit	Any 16-bit value.

Table 1-1 MDIO Field Description

1.1.2 MDIO Timing Sequence

To use PSIO to control MDIO device, the user must first configure the slot, check point and voltage high level or low level stands for a specified period on each pin according to the expected timing.

PSIO0 CH3 (PC.2) pin is used to simulate MDIO input/output waveform and PSIO0 CH1(PC.4) pin is used to simulate MDC output waveform. In the Read operation, the program must configure the PSIO0 CH3 (PC.4) as input mode before reading data.

Figure 1-2 and Figure 1-3 show how to set the slot and operation about Write operation and Read operation, respectively.

Assuming the target MDC clock speed is 500 kHz, and the MDC timing calculation formula is as follows：

- PSIO clock source selected HIRC, HIRC divided by 6 = 8 MHz; and one slot cycle time 0.125 μs (= 1 / 8 MHz).
- Configure the slot counter as 8 and half-a-clock period of MDC = 0.125 μs x 8 (SLOT counter) = 1 μs.

Based on the above, PSIO0 CH1(PC.4) pin generates 500 kHz square wave and configures PSIO0 CH3(PC.2) pin as output buffer or input buffer.

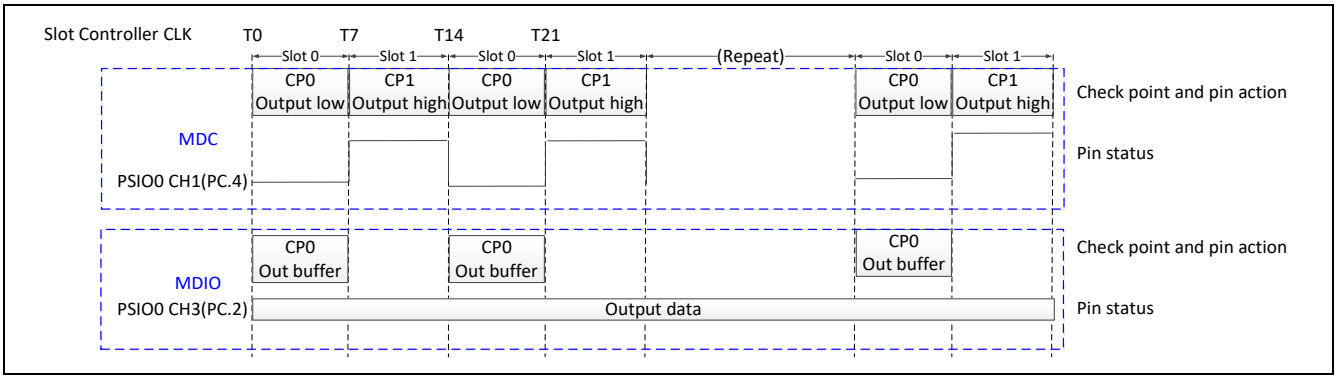

Figure 1-2 Write Operation

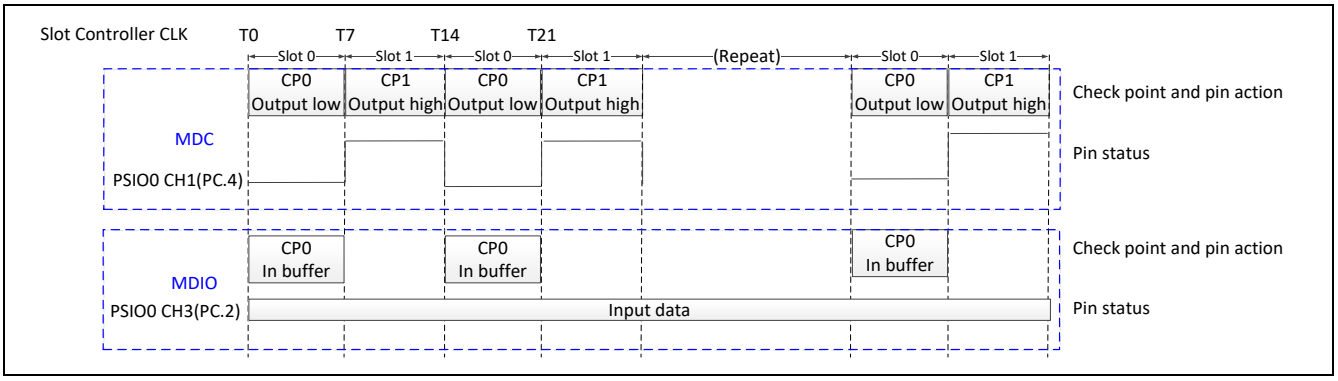

Figure 1-3 Read Operation

1.2 Demo Result

Start the computer terminal software, set UART baud rate to 115200 bps, no parity bit, data length is 8 bits, and the stop bit is 1 bit. After executing the program, the terminal displays the instructions.

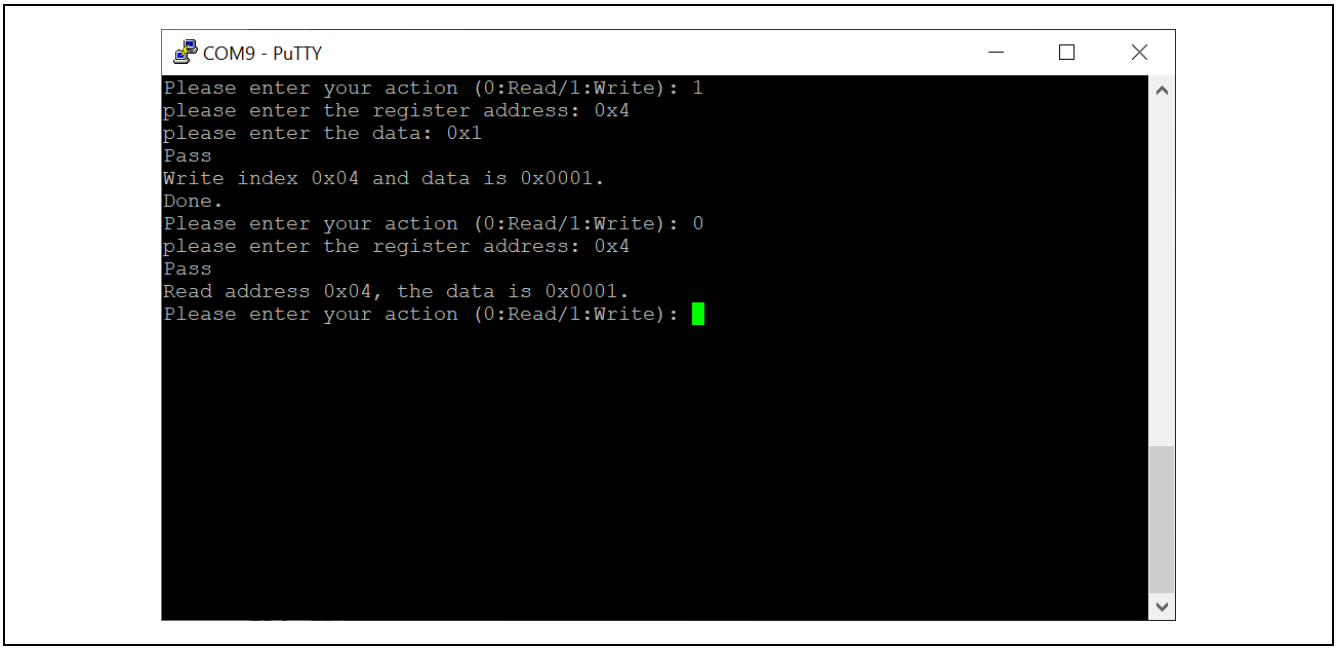

Figure 1-4 Screenshot of the Terminal

The logic analyzer display waveform of MDIO Write and Read operation are shown in Figure 1-5 and Figure 1-6, respectively, which shows the information of configuring register 0x04 as 0x0001 and performing a verify operation.

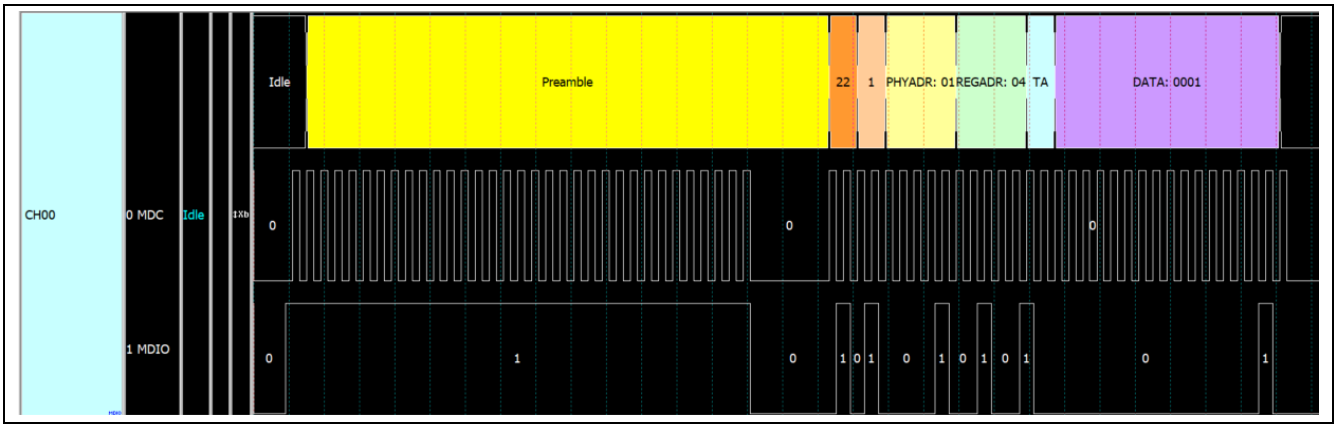

Figure 1-5 MDIO Write Results

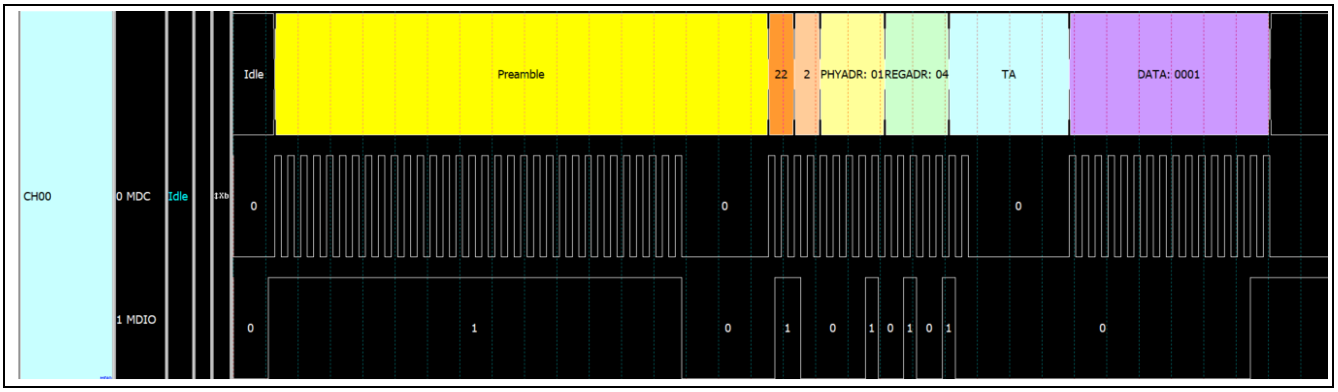

Figure 1-6 MDIO Read Results

2. Code Description

The Write operation and Read operation workflow is as follows and the code is located in *main.c*.

For the Write operation, the program:

- Sends the 32-bit preamble.
- Sends the variable consisting of 2-bit ST, 2-bit Op code, 5-bit PHY address, 5-bit register address, 2-bit TA and 16-bit data.

```
int main(void)
{
    …

    /* Prepare the MDIO package with 32-bit data, including 2-bit ST, 2-bit Op code, 5-bit
       PHY address, 5-bit Register address,  2-bit TA and 16-bit Data */
    u32TxData = (MDIO_ST_CAUSE_22 << MDIO_ST_OFFSET) | (MDIO_OP_W << MDIO_OP_OFFSET) |
                (MDIO_PHY_ADDR << MDIO_PHY_OFFSET) | (u32InputRegAdr << MDIO_REG_OFFSET) |
                (MDIO_TA << MDIO_TA_OFFSET) | u32InputData;

    /* Write Preamble field to Ethernet PHY */
    PSIO_MDIO_Write(&sConfig, &u32Preamble, 32);

    /* Write ST/Op code/PHY adress/TA fields to Ethernet PHY */
    PSIO_MDIO_Write(&sConfig, &u32TxData, 32);

    …
}
```

For the Read operation, the program:

- Sends the 32-bit preamble.
- Sends the variable consisting of 2-bit of ST, 2-bit of Op code, 5-bit of PHY address, 5-bit of register address and 2-bit of TA.
- Reads data from Ethernet PHY.

```
int main(void)
{
    …

    /* Prepare the MDIO package with 32-bit data, including 2-bit ST, 2-bit Op code, 5-bit
       PHY address, 5-bit Register address and 2-bit TA */
    u32TxData = (MDIO_ST_CAUSE_22 << MDIO_ST_OFFSET) | (MDIO_OP_R << MDIO_OP_OFFSET) |
    (MDIO_PHY_ADDR << MDIO_PHY_OFFSET) | (u32InputRegAdr << MDIO_REG_OFFSET) | (MDIO_TA <<
    MDIO_TA_OFFSET);
    u32TxData = u32TxData >> 16;

    /* Write Preamble field to Ethernet PHY */
    PSIO_MDIO_Write(&sConfig, &u32Preamble, 32);

    /* Write ST/Op code/PHY adress/TA fields to Ethernet PHY */
    PSIO_MDIO_Write(&sConfig, &u32TxData, 16);
```

```
    /* Read data from Ethernet PHY */
    PSIO_MDIO_Read(&sConfig, &u32InputData);
    …
}
```

The PSIO initialization, write operation and read operation are located in *MDIO_driver.c*.

Use PSIO_MDIO_Init() to set each check point operation and slot count on each slot .

```
void PSIO_MDIO_Init(S_PSIO_MDIO *psConfig)
{
    const S_PSIO_CP_CONFIG sMDCConfig =
    {

            /* Check Point0 */ /* Check Point1 */  /* Check Point2 */  /* Check Point3 */
/* Slot */   PSIO_SLOT0,        PSIO_SLOT1,         PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */ PSIO_OUT_LOW,      PSIO_OUT_HIGH,      PSIO_NO_ACTION,     PSIO_NO_ACTION,

            /* Check Point4 */ /* Check Point5 */  /* Check Point6 */  /* Check Point7 */
/* Slot */   PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */ PSIO_NO_ACTION,    PSIO_NO_ACTION,     PSIO_NO_ACTION,     PSIO_NO_ACTION
    };

    /* Enable MDC pin and setting general configuration */
    PSIO_SET_GENCTL(PSIO, psConfig->u8MDC, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_OUTPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);

    /* Set data order and MSB */
    PSIO_SET_ORDER(PSIO, psConfig->u8MDC, PSIO_MSB);
    PSIO_SET_ORDER(PSIO, psConfig->u8MDIO, PSIO_MSB);

    /* Set slot counter number */
    PSIO_SCSLOT_SET_SLOT(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT0, 8);
    PSIO_SCSLOT_SET_SLOT(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT1, 8);

    /* Set check point configuration */
    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8MDC, &sMDCConfig);
}
```

Use PSIO_MDIO_Write() to output waveform. Please refer to Figure 1-2 for time sequence.

```
void PSIO_MDIO_Write(S_PSIO_MDIO *psConfig, uint32_t *pu32Data, uint32_t bit_length)
{
    const S_PSIO_CP_CONFIG sDataOutConfig =
    {
            /* Check Point0 */ /* Check Point1 */  /* Check Point2 */  /* Check Point3 */
/* Slot */   PSIO_SLOT0,        PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */PSIO_OUT_BUFFER,    PSIO_NO_ACTION,     PSIO_NO_ACTION,     PSIO_NO_ACTION,

            /* Check Point4 */ /* Check Point5 */  /* Check Point6 */  /* Check Point7 */
/* Slot */   PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */ PSIO_NO_ACTION,    PSIO_NO_ACTION,     PSIO_NO_ACTION,     PSIO_NO_ACTION
    };

    /* Enable MDIO pin and setting general configuration */
```

```
    PSIO_SET_GENCTL(PSIO, psConfig->u8MDIO, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_OUTPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);

    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8MDIO, &sDataOutConfig);

    /* Loop slot0~slot1 */
    PSIO_SET_SCCTL(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT0, PSIO_SLOT1, bit_length - 1,
PSIO_REPEAT_DISABLE);

    /* Set output data width */
    PSIO_SET_WIDTH(PSIO, psConfig->u8MDIO, 0, bit_length);

    /* Set output data */
    PSIO_SET_OUTPUT_DATA(PSIO, psConfig->u8MDIO, *pu32Data);

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for slot controller is not busy */
    while (PSIO_GET_BUSY_FLAG(PSIO, psConfig->u8SlotCtrl));
}
```

Use PSIO_MDIO_Read() to read data from Ethernet PHY. Please refer to Figure 1-3 for time sequence.

```
void PSIO_MDIO_Read(S_PSIO_MDIO *psConfig, uint32_t *pu32Data)
{
    uint32_t u32Data;

    const S_PSIO_CP_CONFIG sDataInConfig =
    {
              /* Check Point0 */ /* Check Point1 */  /* Check Point2 */  /* Check Point3 */
/* Slot */   PSIO_SLOT0,        PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */ PSIO_IN_BUFFER,    PSIO_NO_ACTION,     PSIO_NO_ACTION,     PSIO_NO_ACTION,

              /* Check Point4 */ /* Check Point5 */  /* Check Point6 */  /* Check Point7 */
/* Slot */   PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,  PSIO_SLOT_DISABLE,
/* Action */ PSIO_NO_ACTION,    PSIO_NO_ACTION,     PSIO_NO_ACTION,     PSIO_NO_ACTION
    };

    /* Enable MDIO pin and setting general configuration */
    PSIO_SET_GENCTL(PSIO, psConfig->u8MDIO, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_INPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);

    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8MDIO, &sDataInConfig);

    /* Loop slot 0~slot 1 15 times */
    PSIO_SET_SCCTL(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT0, PSIO_SLOT1, 15,
PSIO_REPEAT_DISABLE);

    /* Set input data width as 16 */
    PSIO_SET_WIDTH(PSIO, psConfig->u8MDIO, 16, 0);

    /* Set input data depth as 1 */
    PSIO_SET_INPUT_DEPTH(PSIO, psConfig->u8MDIO, PSIO_DEPTH1);
```

```
    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for data buffer is full */
    while (!PSIO_GET_TRANSFER_STATUS(PSIO, PSIO_TRANSTS_INFULL0_Msk << (psConfig->u8MDIO *
4)));

    /* Read data from device */
    u32Data = PSIO_GET_INPUT_DATA(PSIO, psConfig->u8MDIO);
    *pu32Data = u32Data;
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.006
- IDE version
  - Keil uVersion 5.38

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M252SD V1.2
- Pin Connect
  - Connect the PSIO0_CH3 (PC.2) pin to the Ethernet PHY MDIO.
  - Connect the PSIO0_CH1 (PC.4) to Ethernet PHY MDC to control Ethernet PHY.

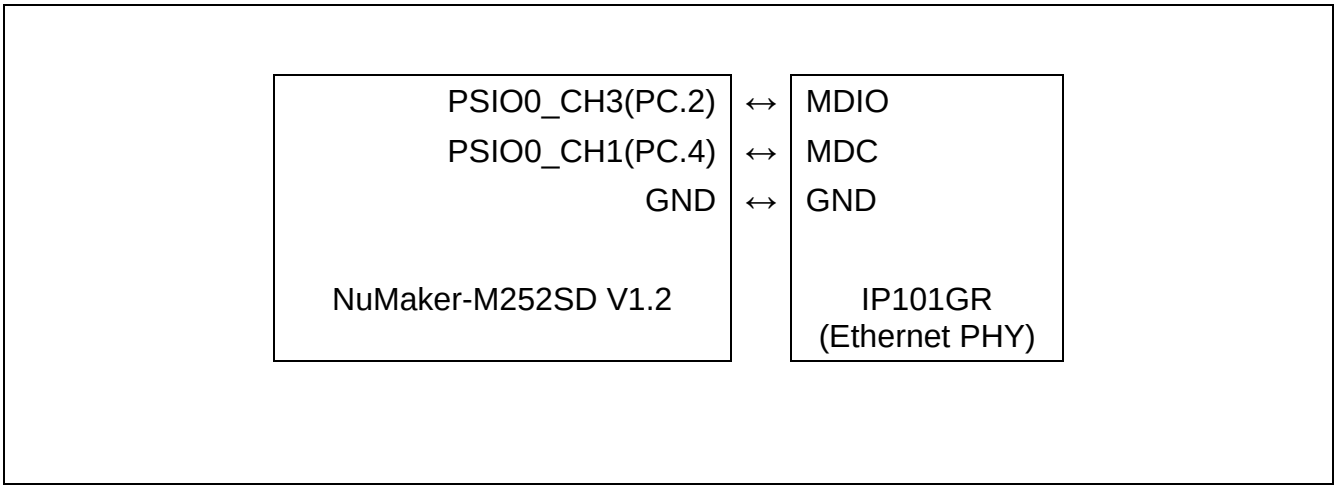

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

EC_M252_PSIO_Emulate_MDIO_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
SampleCode	
ExampleCode	
Project	Source file of example code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M252_PSIO_Emulate_MDIO.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2024.08.22	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.