AN14509

How to Use SmartMDA to Implement MDIO Slave Interface on MCX MCU

Rev. 1.0 — 29 November 2024 Application note

Document information

Information	Content
Keywords	AN14509, MDIO, SmartDMA, MCX N947
Abstract	This application note describes the use of SmartDMA to implement the MDIO slave interface on MCX series MCUs.

1 Introduction

This application note describes the use of SmartDMA to implement the MDIO slave interface on MCX series MCUs.

It includes the introduction of MDIO interface, features and API routines, and a demo. In MCX N947, there is a co-processor, called by SmartDMA, which can be used to implement MDIO slave interface.

Performance:

The SmartDMA can use the system clock (150 MHz) of the MCU as its clock source.

1. For example, to generate a 2 kHz PWM wave, the resolution per cycle can reach 16 bits.
2. If generating a 250 kHz PWM wave with a period of 40 microseconds, the resolution can reach 600 points (40 microseconds divided by the inverse of 150 MHz). it is about 9-bit resolution for full range tuning.

2 MDIO interface

Management Data Input/Output (MDIO) is the serial bus protocol defined in the IEEE 802.3 standard for Ethernet for the Media Independent Interface (MII). MII connects Media Access Control (MAC) devices to Ethernet physical layer (PHY) circuits. The MDIO bus has two signal lines: Management Data Clock (MDC) and Management Data Input/Output (MDIO). MDIO was originally defined in Clause 22 of IEEE 802.3.

The following is an introduction to the relevant timing.

2.1 Management frame structure

Frames transmitted on the MII Management Interface have the frame structure, as shown in Table 1. The order of bit transmission must be from left to right.

Table 1. Management frame format

	Management frame fields							
	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
READ	1...1	01	10	AAAAA	RRRRR	Z0	DDDDDD DDDDDD DDDD	Z
WRITE	1...1	01	01	AAAAA	RRRRR	10	DDDDDD DDDDDD DDDD	Z

2.1.1 PRE (preamble)

The IDLE condition on MDIO is a high-impedance state. All the three state drivers must be disabled and the pull-up resistor of the PHY pulls the MDIO line to a logic one. At the beginning of each transaction, the station management entity sends a sequence of 32 contiguous logic one bits on MDIO with 32 corresponding cycles on MDC to provide the PHY with a pattern that it can use to establish synchronization. A PHY shall observe a sequence of 32 contiguous one bit on MDIO with 32 corresponding cycles on MDC before it responds to any transaction.

If the STA determines that every PHY connected to the MDIO signal is able to accept management frames that are not preceded by the preamble pattern, then the STA may suppress the generation of the preamble pattern, and may initiate management frames with the ST (Start of Frame) pattern.

2.1.2 ST (start of frame)

The start of frame is indicated by a <01> pattern. This pattern assures transitions from the default logic one line state to zero and back to one.

2.1.3 OP (operation code)

The operation code for a read transaction is <10>, while the operation code for a write transaction is <01>.

2.1.4 PHYAD (PHY address)

The PHY Address is five bits, allowing 32 unique PHY addresses. The first PHY address bit transmitted and received is the MSB of the address. A PHY that is connected to the station management entity via the mechanical interface defined in 22.6 responds to transactions addressed to PHY Address zero <00000>. A station management entity that is attached to multiple PHYs must have prior knowledge of the appropriate PHY Address for each PHY.

2.1.5 REGAD (register address)

The operation code for a read transaction is <10>, while the operation code for a write transaction is <01>. The register address is five bits, allowing 32 individual registers to be addressed within each PHY. The first register address bit transmitted and received is the MSB of the address.

2.1.6 TA (turnaround)

The start of frame is indicated by a <01> pattern. This pattern assures transitions from the default logic one line state to zero and back to one. The turnaround time is a 2-bit time spacing between the Register Address field and the Data field of a management frame to avoid contention during a read transaction. For a read transaction, both the STA and the PHY remain in a high-impedance state for the first bit time of the turnaround. The PHY drives a zero bit during the second bit time of the turnaround of a read transaction. During a write transaction, the STA drives a one bit for the first bit time of the turnaround and a zero bit for the second bit time of the turnaround.

2.1.7 DATA (data)

The data field is 16 bits. The first data bit transmitted and received must be bit 15 of the register being addressed.

2.2 Clause 45

To meet the growing needs of 10 Gigabit Ethernet devices, clause 45 of the 802.3ae specification is introduced. Clause 45 added support for low voltage devices down to 1.2 V and extended the frame format to provide access to many more devices and registers.

Table 2 describes the timing difference.

Table 2. Time difference

Clause	ST (Start of Frame)	OP Code	16-bit ADDRESS/DATA
Clause 22	`0b01` for Clause 22	`0b01`: Write `0b10`: Read	Write: Write Data Read: Read Data
Clause 45	`0b00` for Clause 45	`0b00`: RW Address `0b01`: Write `0b11`: Read	Address: Reg Address Write: Write Data Read: Read Data

Table 2. Time difference...*continued*

Clause	ST (Start of Frame)	OP Code	16-bit ADDRESS/DATA
		`0b10`: Read Increment	Read: Inc Read Data

Figure 1 shows the frame structure (OP is the RW address).

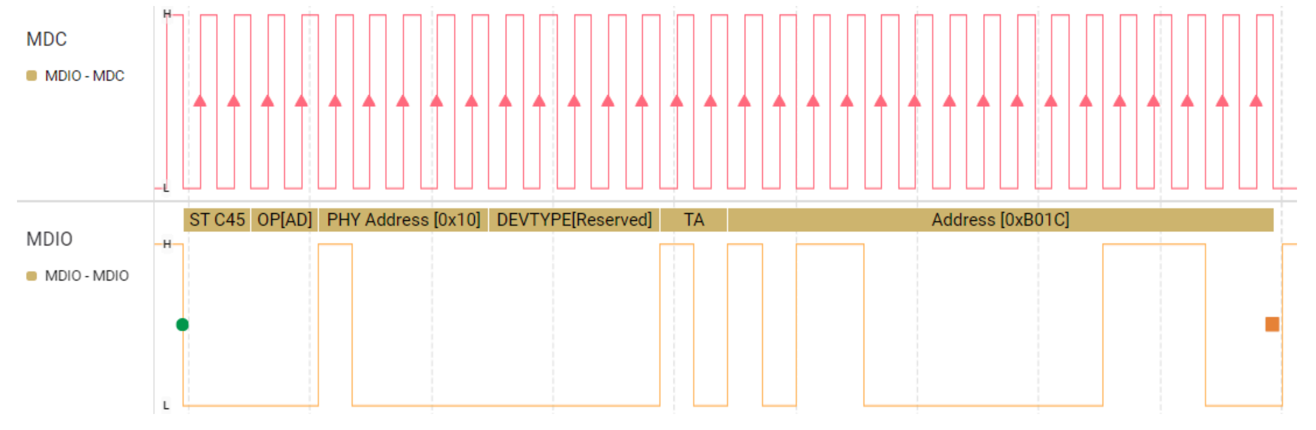


Figure 1. Frame structure

3 SmartDMA for MDIO

SmartDMA is a coprocessor unit within the MCX MCU that can execute a reduced instruction set. It can access GPIO in a single cycle and can receive GPIO input signals as trigger sources. When the MDC clock signal rises, SmartDMA can synchronously capture the value of the MDIO data signal. When the clock signal falls, SmartDMA can transmit the signal level of the MDIO data. Additionally, SmartDMA can set an internal timeout signal to prevent bus hang-ups.

3.1 SmartDMA configuration

SmartDMA, like other peripherals, also has functions, such as reset, clock, and interrupt. Enable the SmartDMA clock in the `SMARTDMA_InitWithoutFirmware()` function.

To use SmartDMA functions more friendly, this application keeps SmartDMA code encapsulated into an array, providing some API functions directly for the user to call.

The SmartDMA code is required to run in the SRAMX at address `0x4000000` when it is packaged. Therefore, before running the SmartDMA code, the user must first transfer the array to the SRAMX at address `0x4000000` using the function `SMARTDMA_InstallFirmware()`. SmartDMA has an interrupt function, and a callback function is executed when an interrupt occurs. Users can install the callback function using the function `SMARTDMA_InstallCallback()`. Users can also enable SmartDMA interrupts and set interrupt priority using the function `EnableIRQWithPriority()`. The function `SMARTDMA_Boot()` is the startup function for SmartDMA, and parameters can be passed to this function as smartdmaParam.

3.2 SmartDMA parameter settings

The parameters consist of two parts: the stack used for SmartDMA operation and the MDIO register settings. Table 3 shows the settings for the MDIO registers.

The settings of the MDIO registers are for the following purposes:

- To receive information on the bus, include operation code, PHY address, DEV address, and data and memory address.

- Users can set the MDIO PHY address and DEV address, and can enable related interrupts, such as frame completion interrupts.

Table 3. Settings for MDIO registers

Offset	Register	Function	Offset	Register	Function
0x0	RXOPCODE	Operation code received	0x40	MEM0ADDR	Memory 0 address
0x4	RXPHYADD	PHY address received	0x44	MEM1ADDR	Memory 1 address
0x8	RXDEVADD	DEV address received	0x48	MEM2ADDR	Memory 2 address
0xc	RXDAT	Data received	0x4c	MEM3ADDR	Memory 3 address
0x10	RXMEMADD	Memory address received	0x50	MEM4ADDR	Memory 4 address
0x14	ADDINC	Address increased	0x54	MEM0SIZE	Memory 0 size
0x18	SETPHYADD	PHY address to be set	0x58	MEM1SIZE	Memory 1 size
0x1c	SETDEVADD	DEV address to be set	0x5c	MEM2SIZE	Memory 2 size
0x20	STA	Status	0x60	MEM3SIZE	Memory 3 size
0x24	INTEN	Interrupt enabled	0x64	MEM4SIZE	Memory 4 size
0x28	TIMERADDR	Timeout timer	0x68	MEM0ZONE	Memory 0 zone
0x2c	MDIODEBUG	Debug buffer	0x6c	MEM1ZONE	Memory 1 zone
0x30	RESERVED	Reserved	0x70	MEM2ZONE	Memory 2 zone
0x34	RESERVED	Reserved	0x74	MEM3ZONE	Memory 3 zone
0x38	RESERVED	Reserved	0x78	MEM4ZONE	Memory 4 zone
0x3c	RESERVED	Reserved	0x7c	RESERVED	Reserved

3.3 Block diagram

Figure 2 shows the block diagram of MDIO implemented on the MCXN947. SmartDMA runs the code in SRAMX, operates the GPIO, and sends the MDIO data from RAM to the master. If it is a read operation from the master, SmartDMA can send out the data.

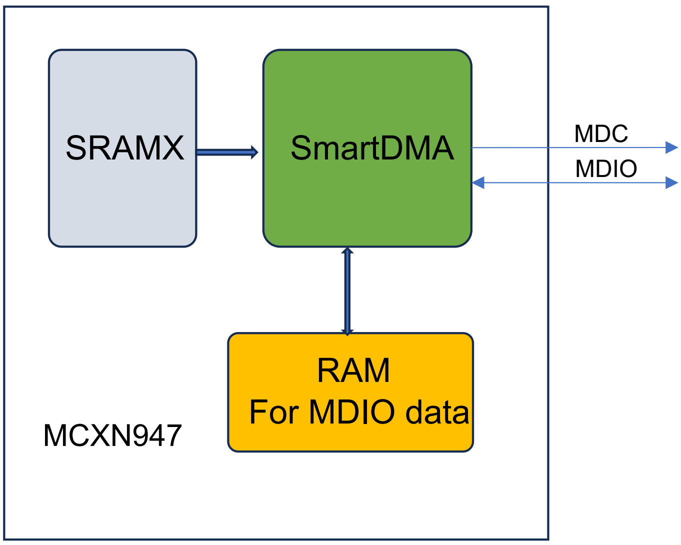


Figure 2. Block diagram

3.4 Features

In this application, SmartDMA implements the functionality of an MDIO slave. It has the following features:

- Users can configure the PHY address, device address, and memory module definitions.
- Users can receive the operation code, PHY address, and device address of the current frame.
- Users can enable frame completion interrupts.
- The slave can receive MDC clock data up to 4 MHz.
- The entire data reception and transmission do not require the involvement of the Arm core.

4 Demo

In this application, two FRDM-MCXN947 boards are used to implement MDIO communication.

4.1 MDIO master code

The Ethernet peripheral of MCX N947 has MDIO master functionality. Users can demonstrate the MDIO master function through the example named “txrx_rxpoll” in the corresponding SDK. The example path is *SDK_2_16_000_FRDM-MCXN947\boards\frdmmcxn947\driver_examples\enet\txrx_rxpoll*.

The important code routines are as follows:

```
static void MDIO_Init(void)
{
    (void)CLOCK_EnableClock(s_enetClock[ENET_GetInstance(EXAMPLE_ENET_BASE)]);
    EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS = ENET_MAC_MDIO_ADDRESS_CR(0);
}

static status_t MDIO_Write(uint8_t phyAddr, uint8_t devAddr,uint16_t regAddra,
 uint16_t data)
{
    uint32_t reg = EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS &
 ENET_MAC_MDIO_ADDRESS_CR_MASK;

    /* Build MII write command. */
    EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS =
        reg | ENET_MAC_MDIO_ADDRESS_GOC_0(1) | ENET_MAC_MDIO_ADDRESS_PA(phyAddr)
 | ENET_MAC_MDIO_ADDRESS_RDA(devAddr)| ENET_MAC_MDIO_ADDRESS_C45E(1);
    EXAMPLE_ENET_BASE->MAC_MDIO_DATA =  (regAddra << 16) | data;
    EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS |= ENET_MAC_MDIO_ADDRESS_GB_MASK;


    while (((EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS &
 ENET_MAC_MDIO_ADDRESS_GB_MASK) != 0U))
    {
    }

}

static status_t MDIO_Read(uint8_t phyAddr, uint8_t devAddr, uint16_t regAddr,
 uint16_t *pData)
{
    uint32_t reg = EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS &
 ENET_MAC_MDIO_ADDRESS_CR_MASK;

    /* Build MII read command. */
```

```
    EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS = reg | ENET_MAC_MDIO_ADDRESS_GOC_0(1) |
ENET_MAC_MDIO_ADDRESS_GOC_1(1)  |
                            ENET_MAC_MDIO_ADDRESS_PA(phyAddr)  |
ENET_MAC_MDIO_ADDRESS_RDA(devAddr)| ENET_MAC_MDIO_ADDRESS_C45E(1);
    EXAMPLE_ENET_BASE->MAC_MDIO_DATA =  (regAddr << 16);

    EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS |= ENET_MAC_MDIO_ADDRESS_GB_MASK;
    while (((EXAMPLE_ENET_BASE->MAC_MDIO_ADDRESS &
ENET_MAC_MDIO_ADDRESS_GB_MASK) != 0U))
    {
    }

    *pData = (EXAMPLE_ENET_BASE->MAC_MDIO_DATA & ENET_MAC_MDIO_DATA_GD_MASK);

}
```

The operations code is as below:

```
MDIO_Init();
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
  MDIO_Write(0x10, 0x20, 0x8000+i, i);
  SDK_DelayAtLeastUs(10, SystemCoreClock);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
  MDIO_Write(0x10, 0x20, 0x9000+i, i+0x100);
  SDK_DelayAtLeastUs(10, SystemCoreClock);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
  MDIO_Write(0x10, 0x20, 0xa000+i, i+0x200);
  SDK_DelayAtLeastUs(10, SystemCoreClock);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
  MDIO_Write(0x10, 0x20, 0xb000+i, i+0x300);
  SDK_DelayAtLeastUs(10, SystemCoreClock);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
        MDIO_Read(0x10, 0x20,  0x8000+i, &g_rec_data);
        PRINTF("addr:0x%4x,RxD:0x%4x.\r\n",0x8000+i,g_rec_data);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
        MDIO_Read(0x10, 0x20,  0x9000+i, &g_rec_data);
        PRINTF("addr:0x%4x,RxD:0x%4x.\r\n",0x9000+i,g_rec_data);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
        MDIO_Read(0x10, 0x20,  0xa000+i, &g_rec_data);
        PRINTF("addr:0x%4x,RxD:0x%4x.\r\n",0xa000+i,g_rec_data);
    }
    for(uint32_t i = 0; i < 8*4; i = i+4)
    {
        MDIO_Read(0x10, 0x20,  0xb000+i, &g_rec_data);
        PRINTF("addr:0x%4x,RxD:0x%4x.\r\n",0xb000+i,g_rec_data);
    }
```

4.2 MDIO slave code

For the MDIO slave, the main functions are implemented by SmartDMA. Users primarily provides register configuration parameters, correctly initiates SmartDMA, and handles pin initialization and interrupt processing.

The important routines are as below:

```
    INPUTMUX_Init(INPUTMUX0);
     /* CTIMER4_CH3 is selected for SMARTDMA arch B 0 */
     INPUTMUX_AttachSignal(INPUTMUX0, 0U, kINPUTMUX_Ctimer4M3ToSmartDma);

     PRINTF("MCXN947 SmartDMA MDIO Demo.\r\n");
     memset((void *)&g_mdio_registers, 0, sizeof(g_mdio_registers));
     memset((void *)g_mdio_mem0, 0x0, sizeof(g_mdio_mem0));
     memset((void *)g_mdio_mem1, 0x0, sizeof(g_mdio_mem1));
     memset((void *)g_mdio_mem2, 0x0, sizeof(g_mdio_mem2));
     memset((void *)g_mdio_mem3, 0x0, sizeof(g_mdio_mem3));
     g_mdio_registers.SETPHYADD = 0x10;
     g_mdio_registers.SETDEVADD = 0x0;
     g_mdio_registers.TIMERADDR  =(uint32_t)&CTIMER4_PERIPHERAL->TCR;
     g_mdio_registers.MDIODEBUG = (uint32_t)g_mdio_debug;
     g_mdio_registers.MEM0ADDR =(uint32_t)g_mdio_mem0;
     g_mdio_registers.MEM0SIZE = MEM0_SIZE;
     g_mdio_registers.MEM0ZONE = MEM0_ZONE;
     g_mdio_registers.MEM1ADDR =(uint32_t)g_mdio_mem1;
     g_mdio_registers.MEM1SIZE = MEM1_SIZE;
     g_mdio_registers.MEM1ZONE = MEM1_ZONE;
     g_mdio_registers.MEM2ADDR =(uint32_t)g_mdio_mem2;
     g_mdio_registers.MEM2SIZE = MEM2_SIZE;
     g_mdio_registers.MEM2ZONE = MEM2_ZONE;
     g_mdio_registers.MEM3ADDR =(uint32_t)g_mdio_mem3;
     g_mdio_registers.MEM3SIZE = MEM3_SIZE;
     g_mdio_registers.MEM3ZONE = MEM3_ZONE;
     g_mdio_registers.INTEN   = (1<<0);
     PRINTF("g_mdio_registers.MEM0ADDR:0x%8x\r\n", g_mdio_registers.MEM0ADDR);
     PRINTF("g_mdio_registers.MEM1ADDR:0x%8x\r\n", g_mdio_registers.MEM1ADDR);
     PRINTF("g_mdio_registers.MEM2ADDR:0x%8x\r\n", g_mdio_registers.MEM2ADDR);
     PRINTF("g_mdio_registers.MEM3ADDR:0x%8x\r\n", g_mdio_registers.MEM3ADDR);

     /* Initialize components */
     SMARTDMA_InitWithoutFirmware();
     SMARTDMA_InstallFirmware(SMARTDMA_MDIO_MEM_ADDR,s_smartdmaMDIOFirmware,
  SMARTDMA_MDIO_FIRMWARE_SIZE);
     SMARTDMA_InstallCallback((smartdma_callback_t)SMARTDMA_Callbck, NULL);
     EnableIRQWithPriority(SMARTDMA_IRQn, 3);
     smartdmaParam.smartdma_stack = (uint32_t*)g_samrtdma_stack;
     smartdmaParam.p_registers_base_address = (uint32_t *)&g_mdio_registers;
     SMARTDMA_Boot(kSmartDMA_MDIO_Slave, &smartdmaParam, 0x2);
```

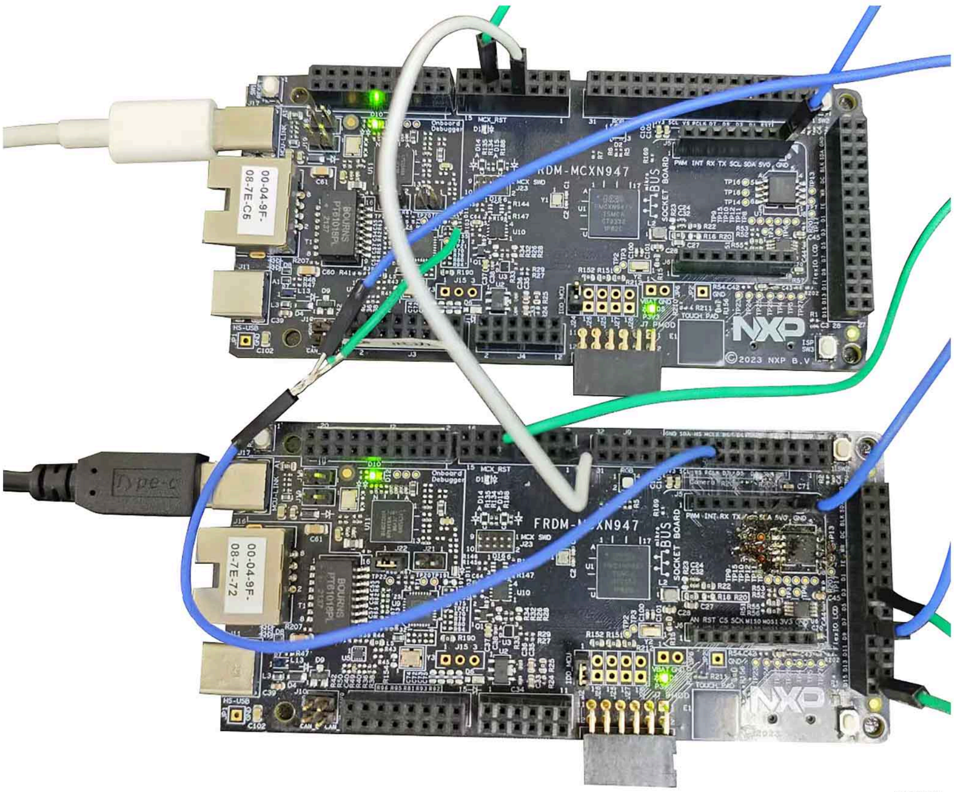
4.3 Hardware preparation

This application requires two FRDM-MCXN947 boards to implement MDIO communication, with one acting as the MDIO master and the other as the MDIO slave. The two boards are connected through two pins and ground. Table 4 describes the hardware connections.

Table 4. Hardware connection

Function	Position on MDIO slave FRDM-MCXN947 boards	Position on MDIO slave FRDM-MCXN947 boards
MDIO	J1-1 (P3_16)	J1-1 (P3_16)
MDC	Pad 1 of R191 (remove R191) (P1_20)	J8-10 (P1_0)
GND	J5-8 (GND)	J5-8 (GND)

Figure 3 is a physical diagram of the hardware connections. The logical device can capture the MDIO waveform and analyze the data format.



aaa-058353

Figure 3. Physical diagram of the hardware connection

Board: FRDM-MCXN947

Logic device: Saleae logic pro16

1. Connect the logic device to the personal computer with USB cable and the logical device to the MDIO signal.
2. Connect FRDM-MCXN947 boards to the personal computer with a USB type-c cable.
3. Connect the signal pins of the two FRDM-MCXN947 boards.

4.4 Software preparation

To modify SDK example `txrx_rxpoll` with MDIO master code as below, download the master firmware into the master board. Unzip the attached MDIO slave software project and open it with MCUXpresso IDE.

1. Import the project in IDE.
2. Build the project code.

3. Download the firmware.

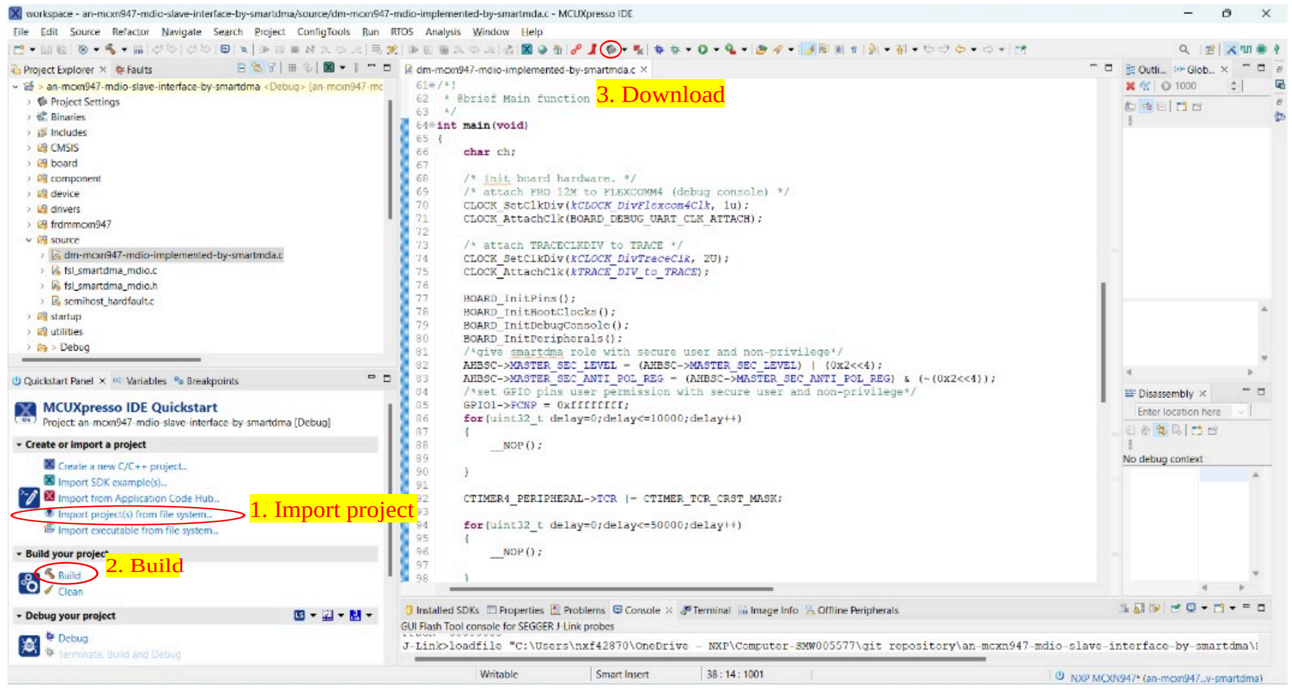


Figure 4. Steps to prepare software

4.5 Result

1. Open the PC host software connected to the serial port of the MDIO master board and then reset the boards.
2. Open the logic device host software tool.
3. Reset the demo boards: first the MDIO slave board and then the MDIO master board.
4. You can see the serial log printed by the MDIO master board as follows:
   - addr:0x8000,RxD:0x 0.
   - addr:0x8004,RxD:0x 4.
   - addr:0x8008,RxD:0x 8.
   - addr:0x800c,RxD:0x c.
   - addr:0x8010,RxD:0x c.
   - addr:0x8014,RxD:0x c.
   - addr:0x8018,RxD:0x c.
   - addr:0x801c,RxD:0x 1c.
   - addr:0x9000,RxD:0x 1c.
   - addr:0x9004,RxD:0x 1c.
   - addr:0x9008,RxD:0x 108.
   - addr:0x900c,RxD:0x 10c.
   - addr:0x9010,RxD:0x 110.
   - addr:0x9014,RxD:0x 114.
   - addr:0x9018,RxD:0x 118.
   - addr:0x901c,RxD:0x 11c.
   - addr:0xa000,RxD:0x 200.
   - addr:0xa004,RxD:0x 204.

- addr:0xa008,RxD:0x 208.
- addr:0xa00c,RxD:0x 20c.
- addr:0xa010,RxD:0x 210.
- addr:0xa014,RxD:0x 214.
- addr:0xa018,RxD:0x 218.
- addr:0xa01c,RxD:0x 21c.
- addr:0xb000,RxD:0x 300.
- addr:0xb004,RxD:0x 304.
- addr:0xb008,RxD:0x 308.
- addr:0xb00c,RxD:0x 30c.
- addr:0xb010,RxD:0x 310.
- addr:0xb014,RxD:0x 314.
- addr:0xb018,RxD:0x 318.
- addr:0xb01c,RxD:0x 31c.

5. The MDIO waveform can be observed on the logic device host software tool as follows:

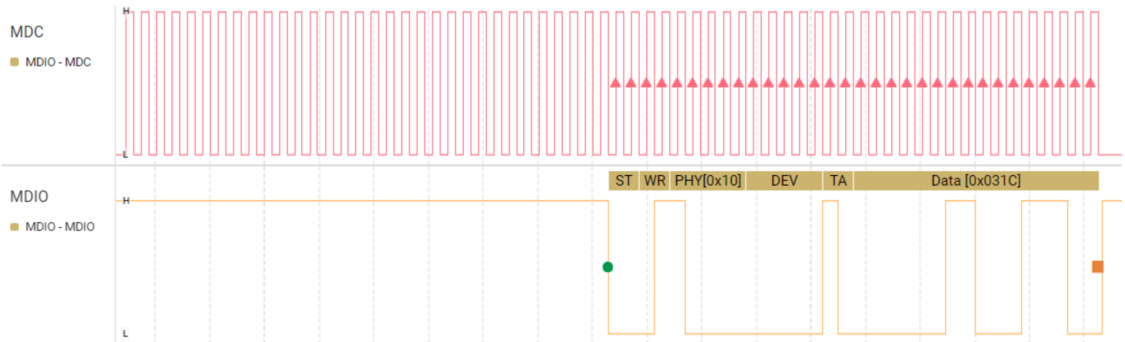


Figure 5. MDIO waveform

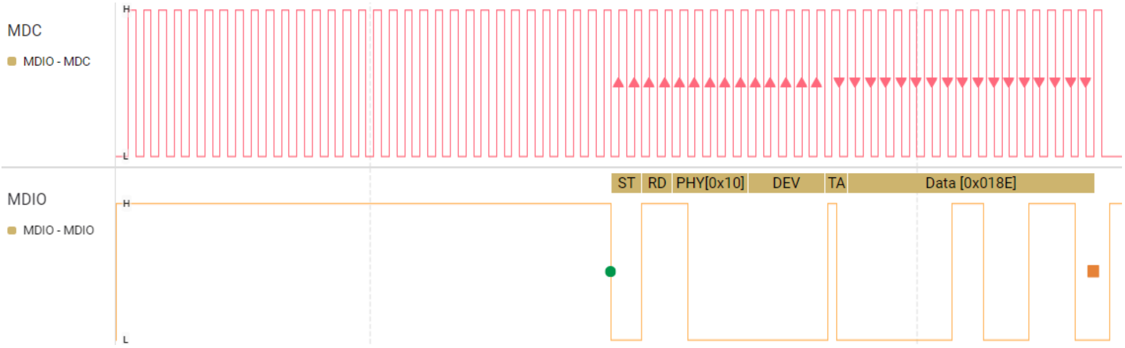


Figure 6. MDIO waveform

5 Reference document

- IEEE Standard for Ethernet (IEEE Std 802.3™-2018)
- CFP MSA Management Interface Specification v2.0
- CFP MSA Hardware Specification v1.4

6 Note about the source code in the document.

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 5 summarizes the revisions to this document.

Table 5. Revision history

Document ID	Release date	Description
AN14509 v.1.0	29 November 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Quick reference data — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents