M2A23 控制仪表指针

NuMicro® 32位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码使用 M2A23 来实现农机仪表盘指针的控制
BSP 版本	M2A23_Series_BSP_CMSIS_V3.00.001
开发平台	NuMaker-M2A23SG V0.1

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. 概述

本范例代码基于 M2A23 平台，实现对专用两相四线步进电机的精确控制，适用于农机仪表盘及各类仪器仪表的指针驱动场景。系统主要包含两个核心功能：

开机自动归零校准：系统启动时先定位机械零点以确立物理基准，随后校准至仪表盘零点，从而确保指针指示的准确性。

周期性往复运动：初始化完成后，控制指针在「零点」与「设定目标位置」之间进行周期性的往复摆动，以验证驱动效果。

1.1 原理

系统原理主要包含机械传动与电气驱动两个部分。

机械传动结构如图 1-1 所示，步进电机转子通过齿轮组驱动指针轴转动。

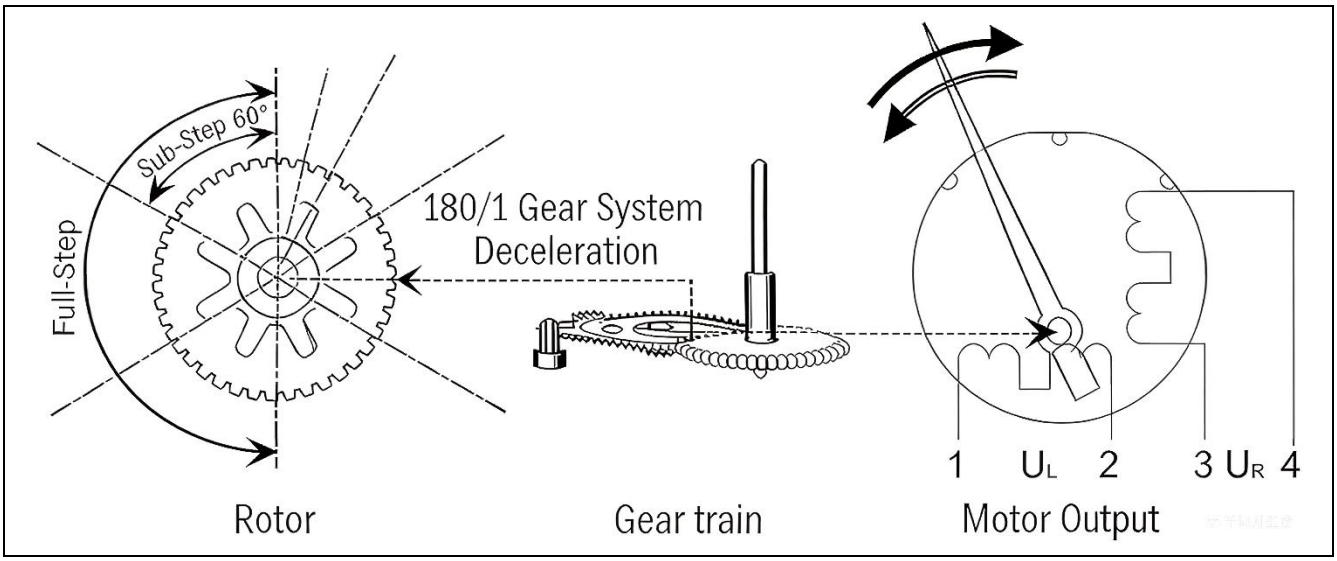

图 1-1 步进电机机械传动系统结构

- **减速比**：180:1。即转子旋转 180°，输出轴（指针）仅旋转 1°。高减速比设计有助于提高指针的定位精度和保持扭矩。
- **步进角度定义**：

  - 分步：转子转动 60°。

  - 全步：转子转动 180°。

电气驱动时序如图 1-2 所示，电机采用六步换相法驱动。通过改变 GPIO 的输出顺序，可控制电机的旋转方向。

正转序列：1 → 2 → 3 → 4 → 5 → 6

反转序列：6 → 5 → 4 → 3 → 2 → 1

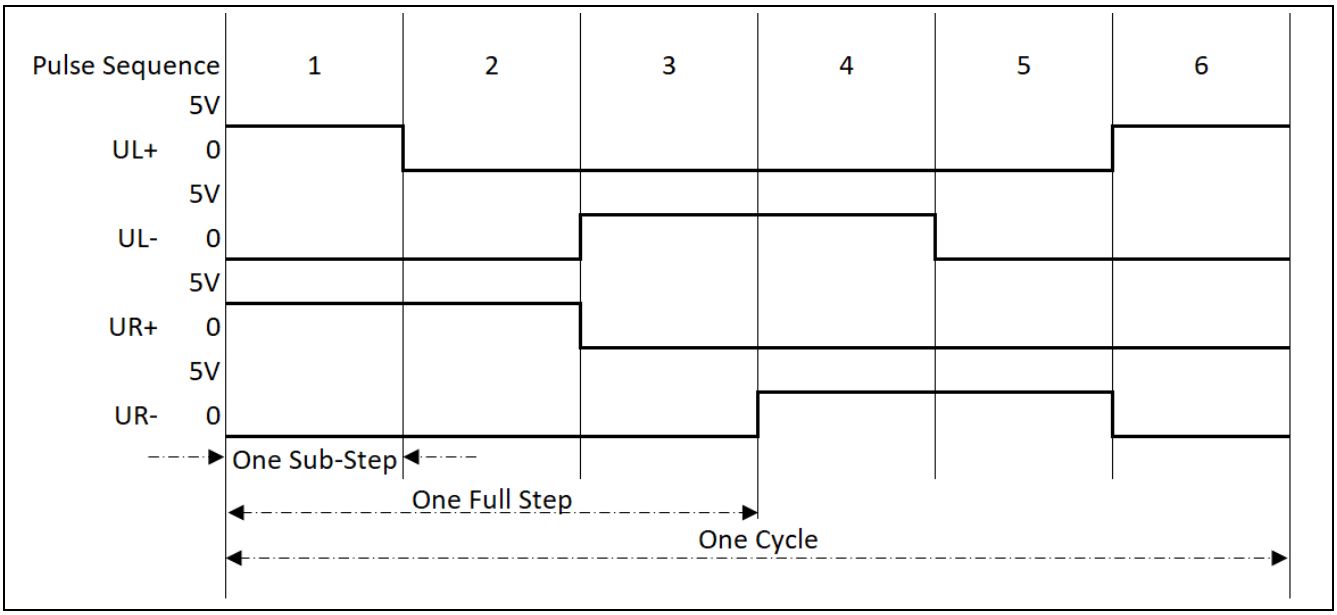

图 1-2 步进电机控制时序

本范例使用GPIO和TIMER协同工作，实现对步进电机的控制。

GPIO：连接电机的两相双极绕组，通过输出特定的电平序列驱动转子步进运动。

TIMER：提供精确的时间基准，用于控制步进脉冲的频率，从而调节电机的转动速度。

1.2 执行结果

PC 上打开串口工具，连接 UART0 用于打印调试信息，如图 1-3 所示。

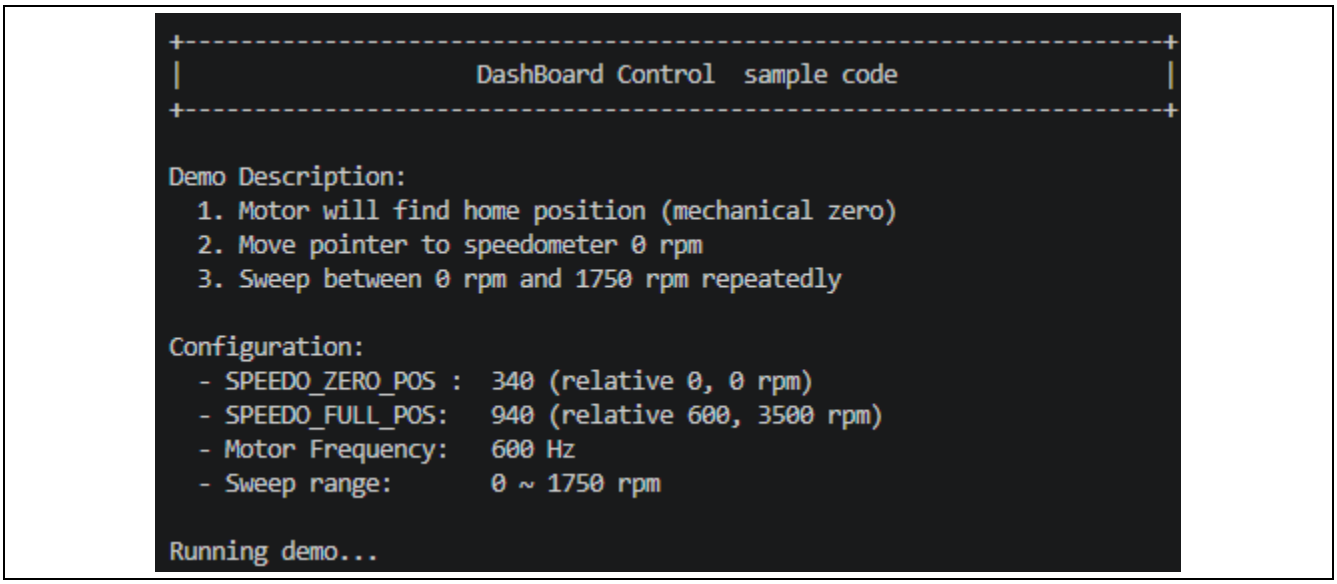
+--+
| DashBoard Control sample code |
+--+
Demo Description:
1. Motor will find home position (mechanical zero)
2. Move pointer to speedometer 0 rpm
3. Sweep between 0 rpm and 1750 rpm repeatedly
Configuration:
- SPEEDO_ZERO_POS : 340 (relative 0, 0 rpm)
- SPEEDO_FULL_POS: 940 (relative 600, 3500 rpm)
- Motor Frequency: 600 Hz
- Sweep range: 0 ~ 1750 rpm
Running demo...

图 1-3 执行结果

2. 代码介绍

代码位于 *main.c*。

宏定义

定义电机控制引脚、最大步数、速度表零点和终点、电机的转动速度等。

```
/*****************************************************************************
 * Motor Pin Definitions
 * GPIO pins used for stepper motor coil control
 *****************************************************************************/
#define UA_POS PB5    // Motor A phase positive (GPIO PB5)
#define UA_NEG PB4    // Motor A phase negative (GPIO PB4)
#define UB_POS PB3    // Motor B phase positive (GPIO PB3)
#define UB_NEG PB2    // Motor B phase negative (GPIO PB2)

/*****************************************************************************
 * Motor Position Definitions
 * All positions are absolute steps from mechanical zero point
 *****************************************************************************/
#define STEP_MAX 1080             // Maximum steps for homing (mechanical zero)
#define SPEEDO_ZERO_POS 340       // Speedometer zero position (absolute, 0 rpm reference)
#define SPEEDO_FULL_POS 940       // Speedometer full scale position (absolute, 3500 rpm)

/*****************************************************************************
 * Motor Control Configuration
 *****************************************************************************/
#define MOTOR_FREQ 600            // Timer interrupt frequency (Hz)
                                  // Controls motor stepping speed
                                  // 600 Hz = 600 steps/second max
#define FULL_STEP_COUNT 6         // Number of steps in one full electrical cycle

#define SPEED_MAX 3500            // Maximum speed on speedometer
                                  // Range: 0 ~ 3500 rpm
```

电机控制参数

六步换相时序、状态机、电机步数、指针位置、方向等。

```
const MotorStep full_step_sequence[] =
{
    {1, 0, 1, 0},  // Step 1: A +5V, B +5V
    {0, 0, 1, 0},  // Step 2: A 0V,  B +5V
    {0, 1, 0, 0},  // Step 3: A -5V, B 0V
    {0, 1, 0, 1},  // Step 4: A -5V, B +5V
    {0, 0, 0, 1},  // Step 5: A 0V,  B -5V
    {1, 0, 0, 0}   // Step 6: A +5V, B 0V
};

......

volatile uint8_t machine_state = MOTOR_STATE_IDLE;   // Current motor state
volatile uint32_t total_step = 0;                    // Remaining steps to move
```

```
volatile uint32_t delay = 0;                          // Software delay counter

......

volatile int16_t speedo_ptr_pos = 0;   // Speedometer pointer position (0 to 600)
volatile uint8_t direction = MOTOR_DIR_FORWARD;  // Current motor direction
```

系统初始化

使能 TIMER0 的时钟、设置电机控制引脚。

```
void SYS_Init(void)
{
    ......
    /* Enable TIMER module clock */
    CLK_EnableModuleClock(TMR0_MODULE);

    /* Select TIMER clock source */
    CLK_SetModuleClock(TMR0_MODULE, CLK_CLKSEL1_TMR0SEL_PCLK0, 0);

    ......

    /* Set multi-function pins for UART0 RXD and TXD */
    GPIO_SetMode(PB, BIT5, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT4, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT3, GPIO_MODE_OUTPUT);
    GPIO_SetMode(PB, BIT2, GPIO_MODE_OUTPUT);
}
```

电机初始化

初始化电机输出以及相关全局变量。

```
void Motor_Init(void)
{
    // Disable all motor coils (set all pins to LOW)
    UA_POS = 0;
    UA_NEG = 0;
    UB_POS = 0;
    UB_NEG = 0;

    // Reset step counter (no pending movement)
    total_step = 0;

    // Set default motor direction
    direction = MOTOR_DIR_FORWARD;

    // Set motor state to idle
    machine_state = MOTOR_STATE_IDLE;

    // Reset speedometer pointer position to zero (0 rpm)
    speedo_ptr_pos = 0;
}
```

电机单步驱动

执行步进电机单步控制，采用六步查表法驱动电机。

```
void Motor_Step(uint8_t direction)
{
    static uint32_t current_phase = 0;      // Internal phase state tracking
    static uint8_t last_direction = MOTOR_DIR_FORWARD;    // Track last direction
                                                          for smooth reversal

    // Handle direction change - adjust phase to avoid repetition
    if (direction != last_direction)
    {
        // Adjust phase index for smooth direction transition
        if (direction == MOTOR_DIR_FORWARD) // Switch to Forward
        {
            current_phase = (current_phase + 2) % FULL_STEP_COUNT;
        }
        else  // Switch to Reverse
        {
            current_phase = (current_phase + 4) % FULL_STEP_COUNT;
        }

        last_direction = direction;
    }

    // Directly assign values from full_step_sequence to GPIO pins
    UA_POS = full_step_sequence[current_phase].A1;
    UA_NEG = full_step_sequence[current_phase].A2;
    UB_POS = full_step_sequence[current_phase].B1;
    UB_NEG = full_step_sequence[current_phase].B2;

    // Update phase for next call (AFTER)
    if (direction == MOTOR_DIR_FORWARD) // Forward
    {
        current_phase = (current_phase + 1) % FULL_STEP_COUNT;
    }
    else  // Reverse
    {
        current_phase = (current_phase - 1 + FULL_STEP_COUNT) % FULL_STEP_COUNT;
    }
}
```

电机启停控制

电机的相位切换由定时器中断自动处理，这两个函数通过修改共享变量控制电机的启停，从而实现主程序与中断解耦。

```
void Motor_Start(uint8_t dir, uint32_t steps)
{
    if (steps == 0)
        return;  // No movement needed
```

```
    direction = dir;
    machine_state = (dir == MOTOR_DIR_FORWARD) ? MOTOR_STATE_FORWARD :
                                                 MOTOR_STATE_REVERSE;
    total_step = steps;
}
/**
 * @brief Stop motor immediately
 */
void Motor_Stop(void)
{
    total_step = 0;
    machine_state = MOTOR_STATE_IDLE;

    UA_POS = 0;
    UA_NEG = 0;
    UB_POS = 0;
    UB_NEG = 0;
}
```

速度表指针归零校准

为确保指针指示的准确性，系统在启动时必须执行自动归零校准流程。

- 寻找机械零点：电机反向旋转到机械零点。
- 校准速度表零点：从机械零点正向旋转至速度表零点。

```
void Speedo_ZeroCalibration(void)
{
    // Step 1: Reverse to mechanical zero point (Home Position)
    direction = MOTOR_DIR_REVERSE;
    total_step = STEP_MAX;

    while (total_step > 0); // Wait until homing complete (step counts down to 0)

    // Step 2: Forward to speedometer working zero position (0 RPM reference)
    direction = MOTOR_DIR_FORWARD;
    total_step = SPEEDO_ZERO_POS;

    while (total_step);

    // Step 3: Reset relative position counter to 0
    speedo_ptr_pos = 0;
}
```

速度表指针控制

将速度值映射为电机控制参数（方向和步数），实现速度表指针的精确位置控制。

```
uint8_t Speedo_SetSpeed(uint32_t speed)
{
    uint32_t target_position;
    int32_t steps_to_move;
```

```
    uint8_t dir;

    // Validate speed range
    if (speed > SPEED_MAX)
    {
        return 2;  // Invalid speed value
    }

    // Convert speed to speedometer pointer position
    // Linear mapping: speed 0~3500 → position 0~600
    if (speed > SPEED_MAX)
        speed = SPEED_MAX;

    target_position = (speed * (SPEEDO_FULL_POS - SPEEDO_ZERO_POS)) / SPEED_MAX;

    // Validate position range
    if (target_position < 0 || target_position > (SPEEDO_FULL_POS - SPEEDO_ZERO_POS))
    {
        return 2;  // Invalid position
    }

    // Calculate steps needed to reach target speed
    if (target_position > speedo_ptr_pos)
    {
        steps_to_move = target_position - speedo_ptr_pos;
        dir = MOTOR_DIR_FORWARD;   // Move pointer toward higher speed
    }
    else if (target_position < speedo_ptr_pos)
    {
        steps_to_move = speedo_ptr_pos - target_position;
        dir = MOTOR_DIR_REVERSE;   // Move pointer toward lower speed
    }
    else
    {
        return 1;  // Already at target speed
    }

    // Start motor movement to update speedometer
    Motor_Start(dir, steps_to_move);

    return 0;  // Success
}
```

定时器初始化

初始化并启动定时器 TIMER0，为电机控制和软件延时提供精确的时间基准。频率由宏定义 MOTOR_FREQ 决定。

```
void TIMER0_Init(void)
{
    // Configure Timer0 in periodic mode
    // Interrupt frequency: MOTOR_FREQ Hz (600 Hz = 600 steps/second)
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, MOTOR_FREQ);
```

```
    // Enable Timer0 interrupt
    TIMER_EnableInt(TIMER0);

    // Enable NVIC interrupt for TMR0
    NVIC_EnableIRQ(TMR0_IRQn);

    // Start Timer0 counting
    TIMER_Start(TIMER0);
}
```

定时器中断服务程序

处理电机步进、位置更新以及软件延时计数。

```
void TMR0_IRQHandler(void)
{
    // Check Timer0 interrupt flag
    if (TIMER_GetIntFlag(TIMER0) == 1)
    {
        /* Clear Timer0 time-out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);

        // Execute motor step if remaining steps exist
        if (total_step > 0)
        {
            // Execute one motor step
            Motor_Step(direction);

            // Decrement remaining step count
            total_step--;

            // Update speedometer pointer position
            if (machine_state == MOTOR_STATE_FORWARD)
                speedo_ptr_pos++;      // Forward: pointer moves toward higher speed
            else if (machine_state == MOTOR_STATE_REVERSE)
                speedo_ptr_pos--;      // Reverse: pointer moves toward lower speed
        }
        else
        {
            // Steps complete, set motor state to idle
            machine_state = MOTOR_STATE_IDLE;
        }

        // Decrement delay counter (if delay in progress)
        if (delay > 0)
            delay--;
    }
}
```

主程序

依次执行硬件初始化、电机归零校准，并进入无限循环以演示速度表指针的往复运动。

```
int32_t main(void)
```

```
{
    ......

    // Initialize motor variables and pins
    Motor_Init();

    // Initialize Timer0 for motor stepping control
    TIMER0_Init();

    // Perform speedometer zero calibration (find mechanical zero, then move to 0 rpm)
    Speedo_ZeroCalibration();

    // Main loop - continuously sweep speedometer pointer
    while (1)
    {
        // Short delay at current position (1 second)
        Delay(MOTOR_FREQ * 1);

        // Wait for any pending movement to complete
        while (Motor_IsBusy());

        // Set speedometer to mid speed 1750 rpm
        Speedo_SetSpeed(SPEED_MAX/2);

        // Wait for movement to complete
        while (Motor_IsBusy());

        // Short delay at mid speed position (1 second)
        Delay(MOTOR_FREQ * 1);

        // Wait for any pending movement to complete
        while (Motor_IsBusy());

        // Set speedometer to zero (0 rpm)
        Speedo_SetSpeed(0);

        // Wait for movement to complete
        while (Motor_IsBusy());
    }
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本

  - M2A23_Series_BSP_CMSIS_V3.00.001

- IDE 版本

  - Keil uVision 5.43
  - VS Code 1.115.3

3.2 硬件需求

- 电路组件

  - NuMaker-M2A23SG V0.1

- 线路示意图

  - 将 MCU 的 PB.2 / PB.3 / PB.5 / PB.4 分别连接到步进电机的 1 / 2 / 3 / 4 脚。
  - 将 UART0 TX （PB.13）引脚通过 UART 转 USB 工具连接到 PC，以显示范例代码的执行结果。

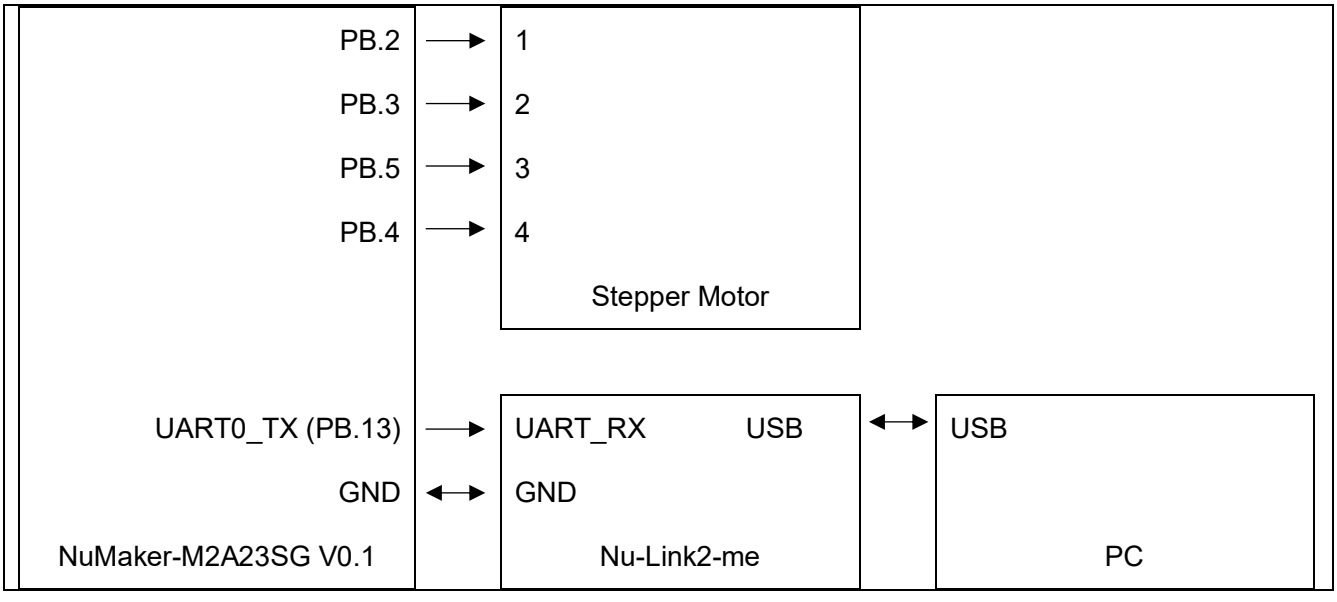

图 3-1 线路示意图

4. 目录信息

目录结构如下图所示。

EC_M2A23_DashBoard_Control_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

图 4-1 目录信息

5. 范例程序执行

5.1 Keil 平台

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 M2A23_DashBoard_Control.uvprojx。
2. 进入编译模式界面
   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式
3. 进入仿真模式界面
   - 执行代码

5.2 VS Code 平台

1. 选择 File
2. 选择 Open Folder
3. 选择 ExampleCode 路径中的 VSCode 文件夹
4. 选择 CMSIS
   - 单击「Build solution」编译
   - 单击「Load & Run application」下载代码至内存
5. 进入仿真模式界面
   - 单击「Load & Debug application」执行代码

6. 修订纪录

Date	Revision	Description
2026.04.15	1.00	初始发布。

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.