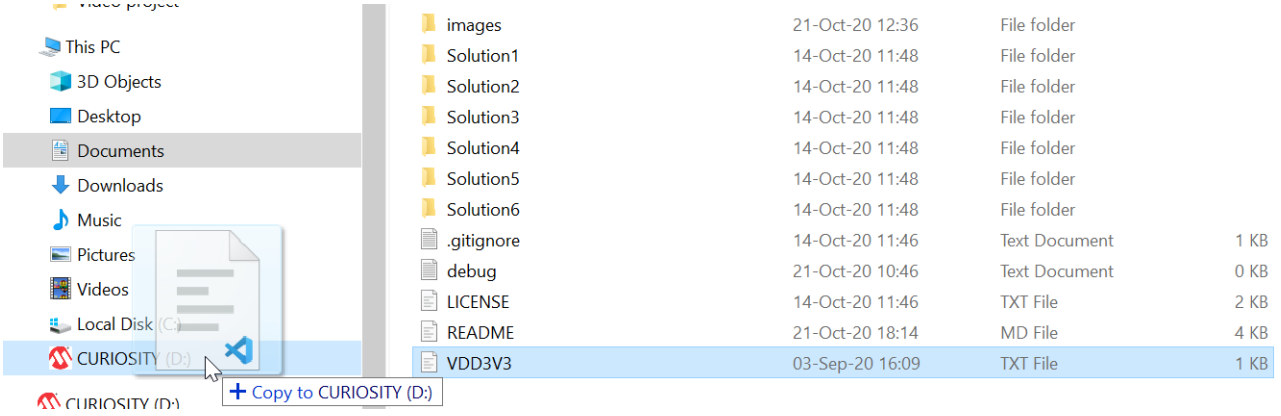
准备工作

作者：Martin Mostad 和 Martin Thomaz，Microchip Technology Inc.

- **硬件准备工作**
  - Microchip AVR128DB48 Curiosity Nano评估工具包 EV35L43A
  - Micro-USB 线缆（Type-A/Micro-B）
  - 采样速度为 100 MHz 的逻辑分析器或示波器（仅任务 2 需要）
- **软件准备工作**
  - MPLAB® X 5.40 或更高版本
  - MPLAB X Dx_DFP 版本 1.4.75
  - MPLAB 代码配置器版本 4.0.2（仅任务 4 需要）
  - MPLAB 数据可视化器（单机版）版本 1.1.793 或更高版本
  - MPLAB Mindi™模拟仿真器
- **其他**
  - MPLAB X 项目可从以下位置获取：
    github.com/microchip-pic-avr-examples/avr128db48-training-on-opamp-xoschf-mvio-mplab
  - AVR128DB48 Curiosity Nano 需要在 3.3V 电压下运行。为此，可将位于 GitHub 资源库中的 VDD3V3.txt 文件拖到 CURIOSITY 存储器上，如图 1 所示。

图 1. 将 AVR128DB48 Curiosity Nano 设置为 3.3V



- **预计完成时间：210 分钟**

目录

1. 简介

本培训文档包含多项任务，用于概述 AVR® DB 系列的各项新特性。每项任务都会先进行基本介绍，然后通过 MPLAB *数据可视化器*对功能进行验证。每项任务都提供了预配置代码，其中大部分代码已完成，只需要针对具体任务配置必要的模块即可。每项任务都提供了解决方案。

全新推出的 AVR DB 系列单片机有三项特性将在本培训中重点进行介绍。这三项特性分别为新的时钟控制器（CLKCTRL）、模拟信号调理（OPAMP）和多电压 I/O（Multi-Voltage I/O，MVIO）。

新的 CLKCTRL 具有一个名为高频晶振（XOSCHF）的新时钟源，以及时钟故障检测（Clock Failure Detection，CFD）功能。XOSCHF 允许使用频率最高达 32 MHz 的外部晶振或外部时钟信号，可以用作主时钟（CLK_MAIN）、实时计数器（Real-Time Counter，RTC）和 12 位 D 型定时器/计数器（TCDn）的时钟源。CFD 功能可用于检测时钟源的输出是否停止，并且可以将主时钟切换到不同的时钟源。

OPAMP 外设具有最多三个内部运算放大器（运放），其中每个运放都配有梯形电阻网络。梯形电阻网络可用于构建各种经典的运放配置。运放的主要用途是提前对将要采集的模拟信号进行调理，或者在控制应用中提供所需的输出驱动。

MVIO 允许一部分 I/O 引脚由名为 V_{DDIO2} 的其他 I/O 电压域供电。其余的 I/O 引脚在 V_{DD} 电压下运行，当与在不同目标电压下运行的其他器件通信时，可能不需要外接电平转换器。

本培训涵盖以下主题：

- 任务 1：CLKCTRL——外部高频振荡器（XOSCHF）和时钟故障检测（CFD）
- 任务 2：CLKCTRL——使用 PLL 的高频 TCD
- 任务 3：OPAMP——电压跟随器
- 任务 4：OPAMP——仪表放大器
- 任务 5：MVIO——OPAMP 作为 VDDIO2 的稳压电源
- 任务 6：MVIO——VDDIO2 故障检测

2. 图标密钥标识符

本文档中使用以下图标来标识任务的不同部分，从而使内容更加简洁明了。

信息：提供特定主题相关的上下文信息。

提示：强调有用的提示和技巧。

Todo：强调需要完成的目标。

结果：强调某个任务步骤的预期结果。

指示重要信息。

执行：强调必要时需在目标器件之外执行的操作。

3. 任务 1：CLKCTRL——外部高频振荡器（XOSCHF）和时钟故障检测（CFD）

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以使用外部高频振荡器（XOSCHF）作为主时钟源。时钟故障检测功能将配置为检测主时钟的故障。如果主时钟发生故障，则将产生不可屏蔽中断（Non-Maskable Interrupt，NMI）。

按下 AVR128DB48 Curiosity Nano 上的开关（SW0）时，软件将触发故障。

当检测到故障时，AVR128DB48 Curiosity Nano 将使 LED 闪烁，而闪烁频率取决于主时钟，这样便可通过闪烁频率的变化呈现主时钟源切换回默认内部源的过程。

这项任务的起始点为 MPLAB X 项目 *任务 1.X* 中的*Assignment1.X*。

- **目标**
  - 配置主时钟以使用 XOSCHF 作为其时钟源
  - 将 CFD 配置为在主时钟发生故障时产生 NMI
  - 使用 CFD 内置的测试功能触发主时钟故障

3.1. 配置主时钟以使用 XOSCHF

操作：

- 编辑 `CLOCK_XOSCHF_crystal_init()` 函数，让主时钟使用 XOSCHF 作为时钟源

信息：

在时钟控制器（CLKCTRL）外设中，几乎所有寄存器都受配置更改保护（Configuration Change Protection，CCP）影响。这意味着要写入这些寄存器，必须先将特定密钥写入 CPU.CCP 寄存器，然后在 4 条 CPU 指令内对受保护位进行写访问。为了确保在写入受保护寄存器时满足这一标准，应使用 `ccp_write_io()` 函数（可在 *avr/cpufunc.h* 文件中找到）。该函数接受两个参数：*uint8_t* 指针和要写入寄存器的值。*io.h* 文件提供的寄存器宏不是 *uint8_t* 指针，因此需要进行强制类型转换。使用该函数时可以参考以下有用的模板：

```
ccp_write_io((uint8_t *) &<register>, new_value)
```

1. 针对 XOSCHF 进行以下配置：使用外部晶振、启动时间为 4000 个周期、频率范围为 16 MHz、始终运行且使能 XOSCHF。将以下代码添加到 `CLOCK_XOSCHF_crystal_init()` 中即可完成上述所有配置：

```
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                  | CLKCTRL_CSUTHF_4K_gc
                  | CLKCTRL_FRQRANGE_16M_gc
                  | CLKCTRL_SELHF_CRYSTAL_gc
                  | CLKCTRL_ENABLE_bm);
```

信息：频率范围设置决定晶振支持的最大频率。频率范围越大，电流消耗越高。

信息：将 XOSCHF 设置为始终运行是为了确保该时钟源在使用之前能够达到稳定状态。后续可以考虑关闭该设置以节省功耗。

2. 通过添加以下代码来等待外部高频晶振达到稳定状态：

```
while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }
```

3. 通过编写以下代码将主时钟源设置为 XOSCHF 并使能时钟输出引脚：

```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc
                | CLKCTRL_CLKOUT_bm);
```

信息：通过使能时钟输出引脚，可以在 CLKCOUT（PA7）引脚上观察主时钟频率。

4. 通过轮询主时钟状态（CLKCTRL.MCLKSTATUS）寄存器中的主时钟振荡器变化（SOSC）位域，等待主时钟切换成功：

```
while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }
```

5. 为了节省功耗，可清零在待机模式下运行（RUNSTDBY）位，这样振荡器在不需要的时候就不会运行：

```
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

结果：通过添加上述全部代码以配置 XOSCHF 后， `CLOCK_XOSCHF_crystal_init()`函数将如下所示：

```
void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator
     * with frequency range 16MHz and 4K cycles start-up time
     */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    //PORTA.DIRSET = PIN7_bm;
    /* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin
*/
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc
                | CLKCTRL_CLKOUT_bm);

    /* Wait for system oscillator changing to complete */
    while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* Clear RUNSTDBY for power save when not in use */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);


    /* Change complete and the main clock is 16 MHz */
}
```

信息：主函数将如下所示，其中包含对 CLOCK_XOSCHF_crystal_init()函数的调用。

```
int main(void)
{
        CLOCK_XOSCHF_crystal_init();
        CLOCK_CFD_CLKMAIN_init();
        LED0_init();
        SW0_init();

        /* Enable global interrupts */
        sei();

        /* Replace with your application code */
        while(1)
        {
            LED0_toggle();
            _delay_ms(200);
    }
}
```

3.2. 将 CFD 配置为检测主时钟的故障

操作：

- 编辑 `CLOCK_CFD_CLKMAIN_init()` 函数，将 CFD 设置为监视主时钟并在主时钟发生故障时产生 NMI
- 设置 NMI 处理程序
- 使用软件测试创建主时钟故障

1. 使能 CFD 并将主时钟设置为时钟源，具体方法为将以下代码：

```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_CLKMAIN_gc
                 | CLKCTRL_CFDEN_bm);
```

添加到 `CLOCK_CFD_CLKMAIN_init()`。

2. 通过编写以下代码允许 CFD 中断并将终端类型设置为 NMI：

```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_INTTYPE_bm
                 | CLKCTRL_CFD_bm);
```

信息：CFD 可以产生正常中断或不可屏蔽中断，具体取决于主时钟中断控制（CLKCTRL.MCLKINTCTRL）寄存器中的中断类型（INTTYPE）位的设置。NMI 的优点是，即使单片机处于中断状态或禁止全局中断，也会触发该中断。这样可以确保立刻执行中断，从而立刻处理错误。若要退出 NMI，只能通过器件复位来实现。

3. 在 `ISR(NMI_vect)` 中，添加以下代码以检查 CFD 是否触发了 NMI 并在需要时使 LED 闪烁：

```
if(CLKCTRL.MCLKINTFLAGS & CLKCTRL_CFD_bm)
    {
    /* This interrupt will trigger if the source for the main clock fails
     * and the CFD is able to switch to a different working clock.
     * In this case that means XOSCHF has failed and is replaced by OSCHF.
     * The main clock is therefore reduced to 4 MHz.
     */

    /* Toggle the LED forever */
    while(1)
    {
        LED0_toggle();
        _delay_ms(200); // 200 ms calculated from 16 MHz == 800 ms
    }

    }
    else
    {
        /* A different NMI has been triggered */
    }
```

信息：所有的 NMI 都只共用一个中断向量，因此要确定是哪个 NMI 触发了中断，需要检查相应的中断标志。

4. 在 `ISR(PORTB_PORT_vect)` 中，添加以下代码以在按下 SW0 时触发时钟故障：

```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL.MCLKCTRLC | CLKCTRL_CFDTST_bm);
```

信息：当主时钟控制 C（MCLKCTRL）寄存器中的时钟故障检测测试（CFDTST）位置 1 时，硬件会模拟时钟故障来测试 CFD 功能。

5. 验证解决方案/项目是否成功编译，具体方法为从 MPLAB X 的顶部菜单栏中选择*Build Main Project*（编译主项目），或者按下 *F11* 键。
6. 刷写器件，具体方法为从 MPLAB X 的顶部菜单栏中选择*Make and Program Device Main Project*（编译并编程器件主项目）。

结果：按下 SW0 后，LED0 将以较低的频率开始闪烁。这是因为主时钟源切换到了启动时钟源，当检测到时钟故障时，将向主时钟控制 B（CLKCTRL.MCLKCTRLB）寄存器写入复位值。创建 `delay_ms(200)` 所需的时钟节拍数是基于 16 MHz 的 XOSCHF 时钟频率计算的，因此当时钟源切换到 4 MHz 内部时钟时，`delay_ms` 将变为原先的 4 倍。

信息：启动时钟源由振荡器配置（OSCCFG）熔丝中的时钟选择（CLKSEL）位域决定。

WARNING 由于 CFD 会向 MCLKCTRLB 寄存器中写入其默认值，因此在用户再次设置之前，CLKOUT 引脚上的时钟将无法使用。

4. 任务 2：CLKCTRL——使用 PLL 的高频 TCD

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以使用 TCD 和 PLL 作为时钟源来生成高分辨率的 PWM 信息。

PLL 将提高 OSCHF 的时钟频率。

PLL 的输出将用作 TCD 的时钟源，从而生成高分频率的 PWM 信号。

这项任务的起始点为 MPLAB X 项目*Assignment2.X*。

- **目标**
  - 配置 PLL 以使用 OSCHF 作为时钟源
  - 配置 TCD 以使用 PLL 作为时钟源并输出 PWM 信号
  - 使用逻辑分析器或示波器对 PWM 信号进行可视化

4.1. 配置 TCD 以使用 PLL 作为时钟源

操作：

- 编辑 `CLOCK_OSCHF_crystal_PLL_init()`，将 PLL 配置为使用 OSCHF 作为时钟源并将其频率提高至原先的 3 倍
- 编辑 `TIMER_TCD0_init()`，让 TCD 使用 PLL 作为时钟源并生成 PWM 信号
- 使用逻辑分析器绘制 PWM 信号

1. 编辑 `CLOCK_OSCHF_crystal_PLL_init()`，通过添加以下代码将 OSCHF 频率设置为 16 MHz：

```
ccp_write_io((uint8_t *)&CLKCTRL.OSCHFCTRLA, CLKCTRL_FREQSEL_16M_gc);
```

信息：PLL 的最小输入频率为 16 MHz，最大输出频率为 48 MHz。

2. 通过将以下代码添加到 `CLOCK_OSCHF_crystal_PLL_init()` 中，以将 PLL 倍频因子设置为 3x：

```
ccp_write_io((uint8_t *) &CLKCTRL.PLLCTRLA, CLKCTRL_MULFAC_3x_gc);
```

信息：PLL 有两个输入时钟源：OSCHF 和 XOSCHF。默认时钟源为 OSCHF，而 XOSCHF 可通过将 PLL 控制 A（PLLCTRLA）寄存器中的选择 PLL 的时钟源（SOURCE）位来配置。

3. 编辑 `TIMER_TCD0_init()`，通过添加以下代码将 PLL 设置为时钟源且不使用预分频器：

```
TCD0.CTRLA = TCD_CLKSEL_PLL_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;
```

4. 通过编写以下代码将 TCD 设置为一种斜坡 PWM 模式：

```
TCD0.CTRLB = TCD_WGMODE_ONERAMP_gc;
```

5. 通过添加以下代码在引脚上提供 PWM 信号：

```
/*Set TCD pins as output*/
PORTA.DIRSET = PIN4_bm | PIN5_bm;
ccp_write_io((uint8_t *) &TCD0.FAULTCTRL, TCD_CMPAEN_bm | TCD_CMPBEN_bm);
```

6. 使能 TCD，具体方法为先确保状态（TCDn.STATUS）寄存器中的使能就绪（ENRDY）位置 1，然后将控制 A（TCDn.CTRLA）寄存器中的 ENABLE 位置 1：

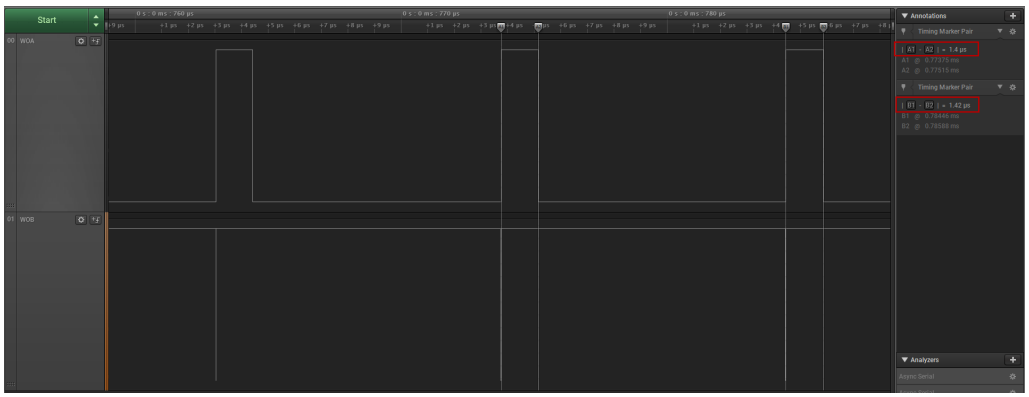
```
/* Wait for synchronization */
while(!(TCD0.STATUS & TCD_ENRDY_bm))
{
    ;
}
/* Enable TCD0 */
TCD0.CTRLA |= TCD_ENABLE_bm;
```

信息：除了已设置的配置之外，还需将 TCD 设置为在每个周期后将 WOA 的占空比递增一位。关于实现方法，请参见 `ISR(TCD0_OVF_vect)`。

7. 验证解决方案/项目是否成功编译，具体方法为从 MPLAB X 的顶部菜单栏中选择*Build Main Project*，或者按下 *F11* 键。
8. 刷写器件，具体方法为从 MPLAB X 的顶部菜单栏中选择*Make and Program Device Main Project*。
9. 使用逻辑分析器绘制 PWM 信号，其中 WOA 的输出位于 PA4 上，而 WOB 的输出位于 PA5 上。

结果：PLL 与 TCD 组合在一起来生成超高分辨率的 PWM 信号。在图 4-1 中，可以看到占空比在每个周期后增加了约 20 ns，从 1.4 µs 增加到 1.42 µs，呈现出了 PWM 信号的高分辨率。

图 4-1. 任务 2：结果



5. 任务 3：OPAMP——电压跟随器

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以使用模拟信号调理（OPAMP）外设中的运放 OP0 作为电压跟随器。

将数模转换器（Digital-to-Analog Converter，DAC）配置为生成正弦信号，以用作运放的输入。

将模数转换器（Analog-to-Digital Converter，ADC）配置为对运放的输出进行采样。

运放的输入和输出均采用数据流协议传输到 MPLAB 数据可视化器。

这项任务的起始点为 MPLAB X 项目*Assignment3.X*。

- **目标**
  - 了解在 OPAMP 外设中配置运放实例所需的步骤
  - 将 OP0 配置为电压跟随器
  - 使用 MPLAB 数据可视化器绘制 OP0 的输入和输出

5.1. 将 OP0 配置为电压跟随器

操作：

- 编辑 `OPAMP0_init()`，将 OP0 配置为电压跟随器
- 在 MPLAB 数据可视化器中绘制 OP0 的输入和输出

1. 编辑 `OPAMP0_init()`，通过编写以下代码设置 OPAMP 外设的时基：

```
OPAMP.TIMEBASE = OPAMP_TIMEBASE_US;
```

信息：出于内部计时目的，时基（OPAMP.TIMBASE）寄存器需要最大数量的 CLK_PER 周期才能实现不低于 1 µs 的计时间隔。用于确定向 TIMBASE 中写入哪个数字的规则如下：确定等于 1 µs 的 CLK_PER 周期的数量。如果是整数，则减 1。如果不是整数，则向下舍入。以下宏可用于计算 TIMEBASE 寄存器的值：

```
#define OPAMP_TIMEBASE_US      (ceil(F_CPU /1e6)-1)
```

2. 通过编写以下代码将输出模式设置为正常并将 OP0 的始终开启（ALWAYSON）位置 1：

```
OPAMP.OP0CTRLA = OPAMP_OP0CTRLA_OUTMODE_NORMAL_gc | OPAMP_ALWAYSON_bm;
```

信息：OPAMP 外设中的每个运放实例都可以单独配置输出模式，具体包括关闭和正常两种模式。在关闭模式下，运放的输出驱动器处于关闭状态，但可通过 DRIVE 事件改写。在正常模式下，驱动器始终开启。

信息：OPAMP 外设中的每个运放实例都可以单独开启或关闭。具体实现方法为将 ALWAYSON 位置 1/清零，或者使用 ENABLEn/DISABLEn 事件。如果使用事件来控制运放，必须将 ALWAYSON 位清零。

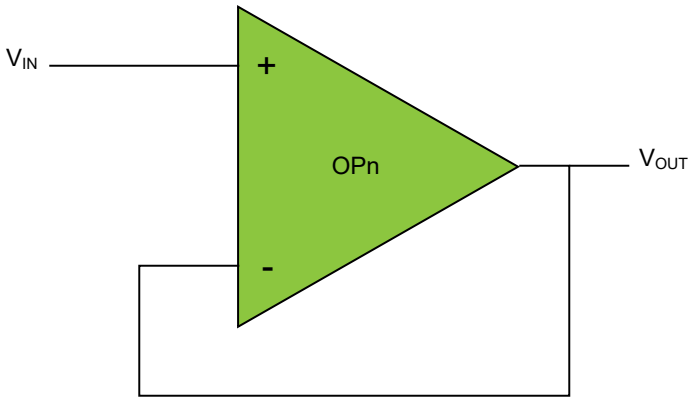
WARNING 必须将 OPnCTRA 寄存器中的事件使能（EVENTEN）位置 1，才能使用事件来控制运放

3. 通过编写以下代码将 OP0 配置为电压跟随器并使用 DAC 作为输入：

```
OPAMP.OP0INMUX = OPAMP_OP0INMUX_MUXNEG_OUT_gc | OPAMP_OP0INMUX_MUXPOS_DAC_gc;

/* Configure the Op Amp n Resistor Wiper Multiplexer */
OPAMP.OP0RESMUX =  OPAMP_OP0RESMUX_MUXBOT_OFF_gc | OPAMP_OP0RESMUX_MUXWIP_WIP0_gc |
                   OPAMP_OP0RESMUX_MUXTOP_OFF_gc;
```

信息：

图 5-1. 电压跟随器配置中的运放



要实现电压跟随器配置，需将运放的输出直接连接到反相输入。为此，可将运放 n 输入多路开关（OPnMUX）寄存器中的反相输入多路开关（MUXNEG）位域设置为 OUT。通过将 OPnMUX 寄存器中的正输入多路开关（MUXPOS）位域设置为 DAC，DAC 的输出与 OP0 的正输入可以通过内部通道相连接。由于不需要在电压跟随器中使用梯形电阻网络，因此可以将运算放大器 n 梯形电阻网络多路开关（OPnRESMUX）寄存器中的所有位域都设置为其默认值。

4. 通过编写以下代码设置稳定时间：

```
OPAMP.OP0SETTLE = OPAMP_MAX_SETTLE;
```

信息：运放 n 稳定定时器（OPnSETTLE）寄存器中的值是运放的输出达到稳定状态所需的微秒数。该时间在很大程度上取决于应用。如果未知，建议将其设置为最大值 0x7F。内部定时器将结合使用该值与 TIMEBASE 寄存器中的值，以确定何时产生 READYn 事件并将 OPnSTATUS 寄存器中的 SETTLED 标志置 1。

5. 通过编写以下代码使能 OPAMP 外设：

```
OPAMP.CTRLA = OPAMP_ENABLE_bm;
```

6. 通过检查运放 n 状态（OPnSTATUS）寄存器中的运放已稳定（SETTLED）位域，等待运放稳定后再退出初始化：

```
while (!(OPAMP.OP0STATUS & OPAMP_SETTLED_bm))
    {
        ;
    }
```

7. 验证解决方案/项目是否成功编译，具体方法为从 MPLAB X 的顶部菜单栏中选择*Build Main Project*，或者按下 *F11* 键。
8. 刷写器件，具体方法为从 MPLAB X 的顶部菜单栏中选择*Make and Program Device Main Project*。

结果：器件现已完成刷写，并准备好开始向 MPLAB *数据可视化器*传输数据。

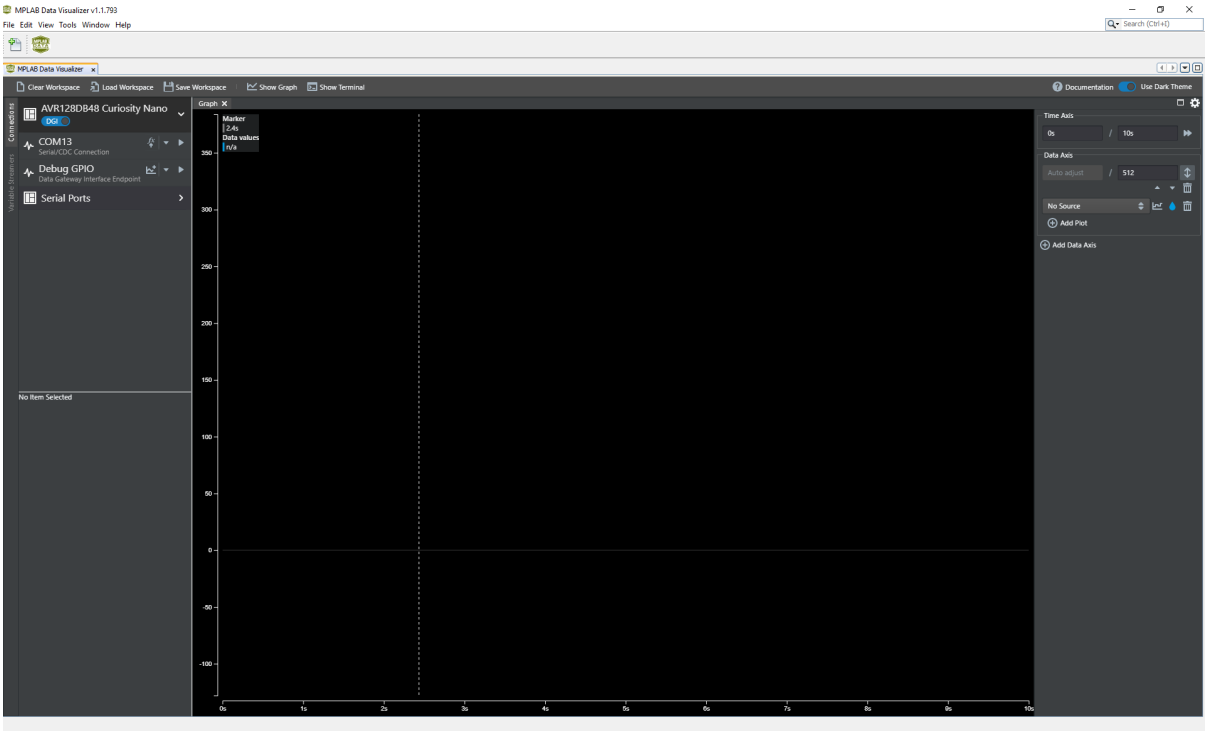
5.2. 在 MPLAB 数据可视化器中绘图

MPLAB *数据可视化器*是一款用于对正在运行的嵌入式目标的数据进行处理和可视化的程序。该程序既可作为 MPLAB X IDE 插件访问，也可作为独立程序访问。在这项任务中，将配置*数据可视化器*以对通过 USART 接收的 OP0 的输入和输出进行绘图。该配置将通过保存的工作区来完成，其中将介绍关于如何显示数据的基础知识。如需了解如何设置自己的工作区，可单击 MPLAB *数据可视化器*中的 *Documentation*（文档）按钮获取一份详细的指南。

操作：配置 MPLAB *数据可视化器*以对接收的 OP0 输入和输出采样进行绘图。

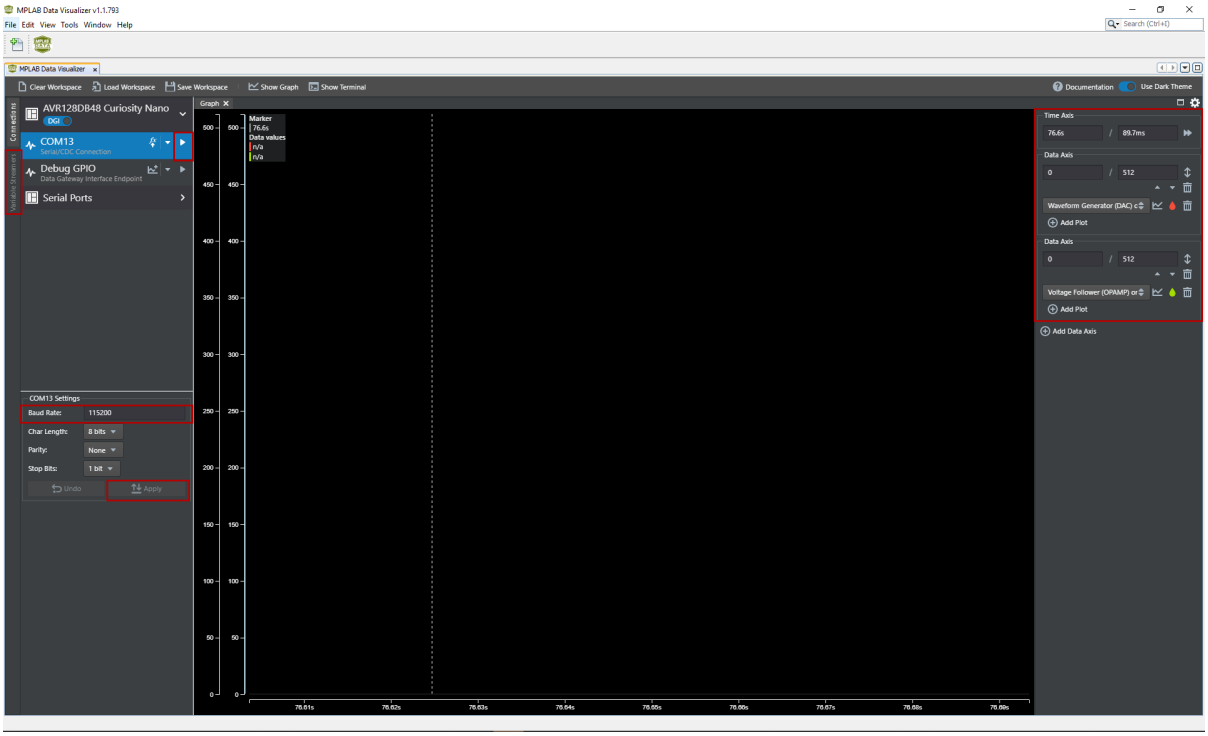
1. 打开程序，然后插入一个已刷写的器件。确保用于 USART 通信的 COM 端口尚未使用。启动画面将类似于图 5-2 所示。

图 5-2. 任务 3：MPLAB 数据可视化器启动页面



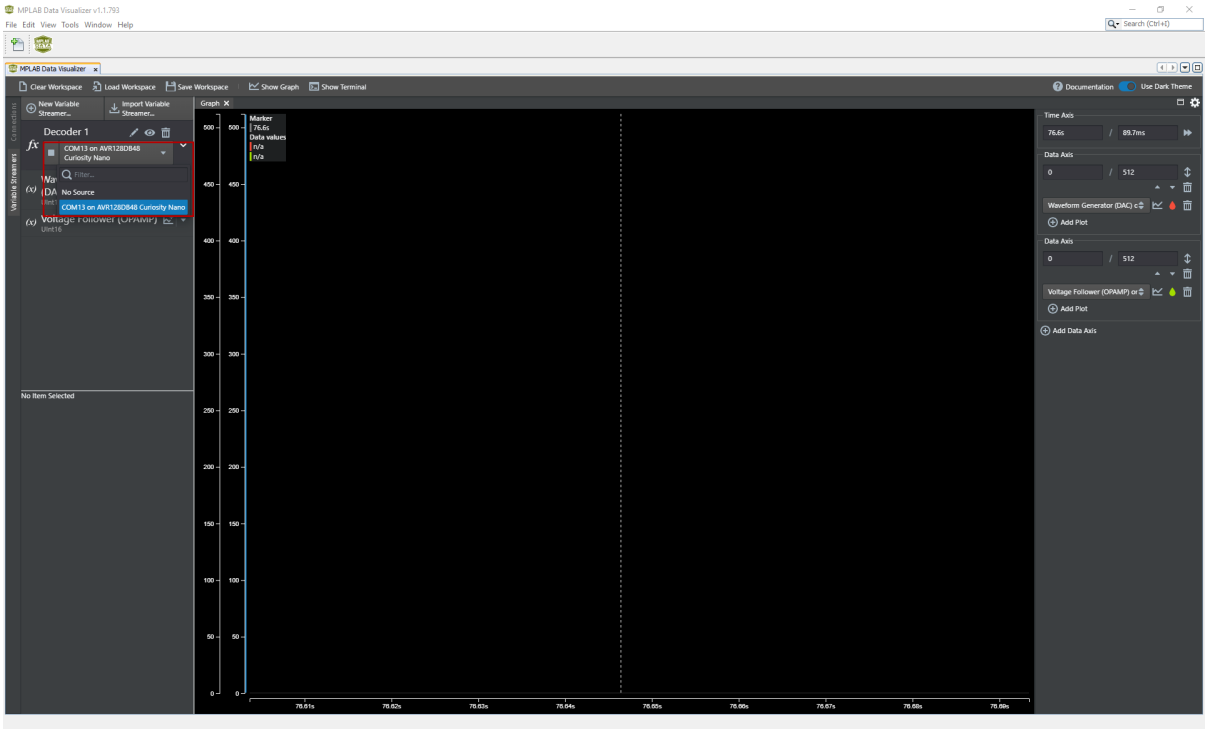
2. 加载工作区。按下 *Load Workspace*（加载工作区）按钮，然后添加名为 *Assignment3.json* 的工作区文件。本培训中的所有工作区均可在 `DataVisualizer` 中找到。图表中会出现两个轴。这两个轴可在右侧面板中进行配置，如图 5-3 所示。

图 5-3. 任务 3：数据可视化器中加载的工作区



3. 在左侧面板上选择 COM 端口（如图 5-3 所示）以绘制数据。确保 *Baud Rate*（波特率）为 *115200*，然后按下 *Apply*（应用）按钮以设置波特率。按下 COM 字段旁边的 *Play*（播放）按钮（如图 5-3 所示）。按下 *Variable Streamers*（变量流）按钮，以将解码器连接到 USART 数据流。将 COM 端口设置为解码器 1 的输入，如图 5-4 所示。

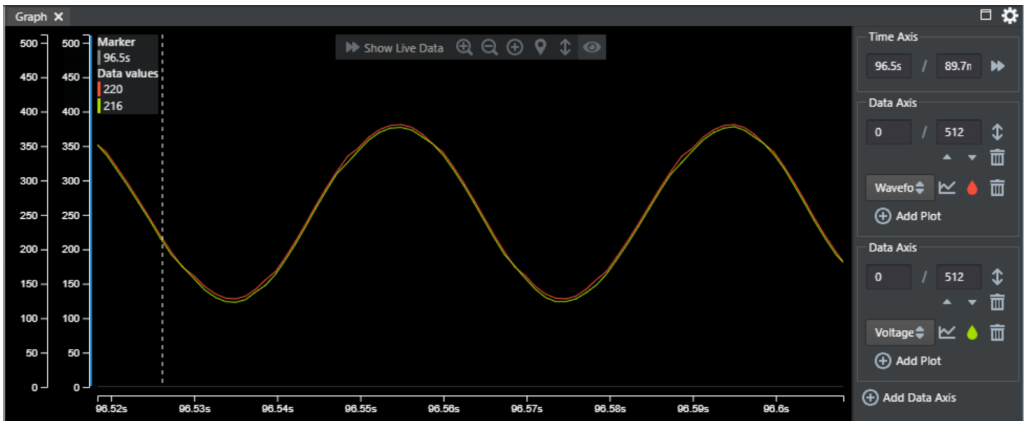
图 5-4. 任务 3：变量流



4. 按下 *Show Live Data*（显示实时数据）以开始绘制来自器件的实时数据。

结果：通过 USART 传输到 PC 的数据绘制在 MPLAB *数据可视化器*中，如图 5-5 所示。红色波形是 DAC 的输出，绿色波形是运放的输出。正如预期的那样，我们可以看到运放的输出紧跟运放的输入。

图 5-5. 任务 3：结果



信息：DAC 输出一个具有 256 mV_{pp} 和 256 mV 直流失调的 50 Hz 正弦波。启动时，该正弦波在 sine_wave_table_init()函数中预先计算，DAC使用 TCB0 ISR (SINE_WAVE_TIMER_vect)之后的值更新。

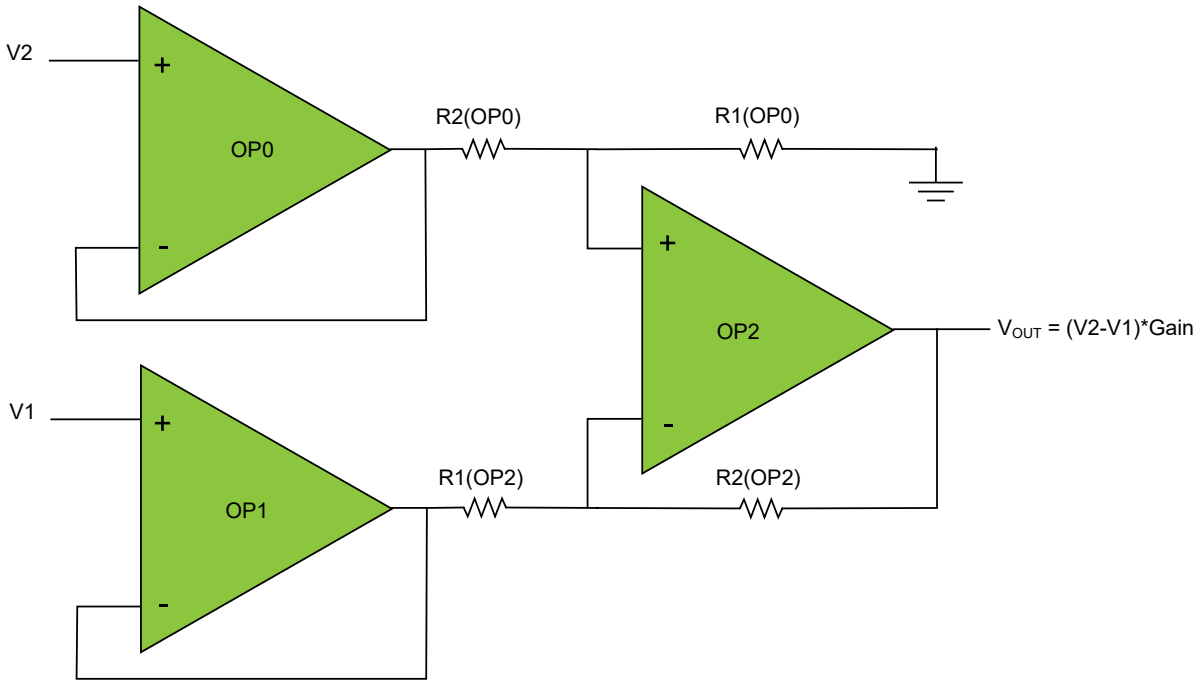
```
void sine_wave_table_init(void)
{
    for(uint16_t i = 0; i < SINE_WAVE_STEPS; i++)
    {
        sine_wave[i] = SINE_DC_OFFSET + SINE_AMPLITUDE *
sin(i * M_2PI / SINE_WAVE_STEPS);
    }
}
```

```
ISR(SINE_WAVE_TIMER_vect) {
    volatile static uint16_t sine_wave_index = 0

    data_stream.dacVal = sine_wave[sine_wave_index];
    DAC0_setVal(data_stream.dacVal);
    sine_wave_index++;
    sine_wave_index = sine_wave_index % SINE_WAVE_STEPS;

    /* Clear interrupt flag */
    SINE_WAVE_TIMER.INTFLAGS = TCB_CAPT_bm;
}
```

6. 任务 4：OPAMP——仪表放大器

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以将模拟信号调理（OPAMP）外设中的所有运算放大器组合构建一个仪表放大器，如下面的图 6-1 所示。

图 6-1. 任务 4：仪表放大器



该配置将通过 MCC 来完成。

将数模转换器（DAC）配置为生成正弦信号，以用作仪表放大器的输入之一。第二个输入将是 V_{DD} 除以 2。

将模数转换器（ADC）配置为对仪表放大器的输出进行采样。运放的输入和输出均采用数据流协议传输到 MPLAB *数据可视化器*。

该电路还将使用包含 AVR DB 运放模型的预制原理图进行仿真。该原理图可在 MCC 中找到。

这项任务的起始点为 MPLAB X 项目*Assigement4.X*。

- **目标**
  - 了解如何在 MCC 中配置运放
  - 使用 MCC 将 OPAMP 外设配置为仪表放大器
  - 使用 MPLAB 数据可视化器绘制仪表放大器的输入和输出
  - 在 Mindi 中使用包含 AVR DB 运放模型的预制原理图仿真仪表放大器

6.1. 将 OPAMP 配置为仪表放大器

操作：

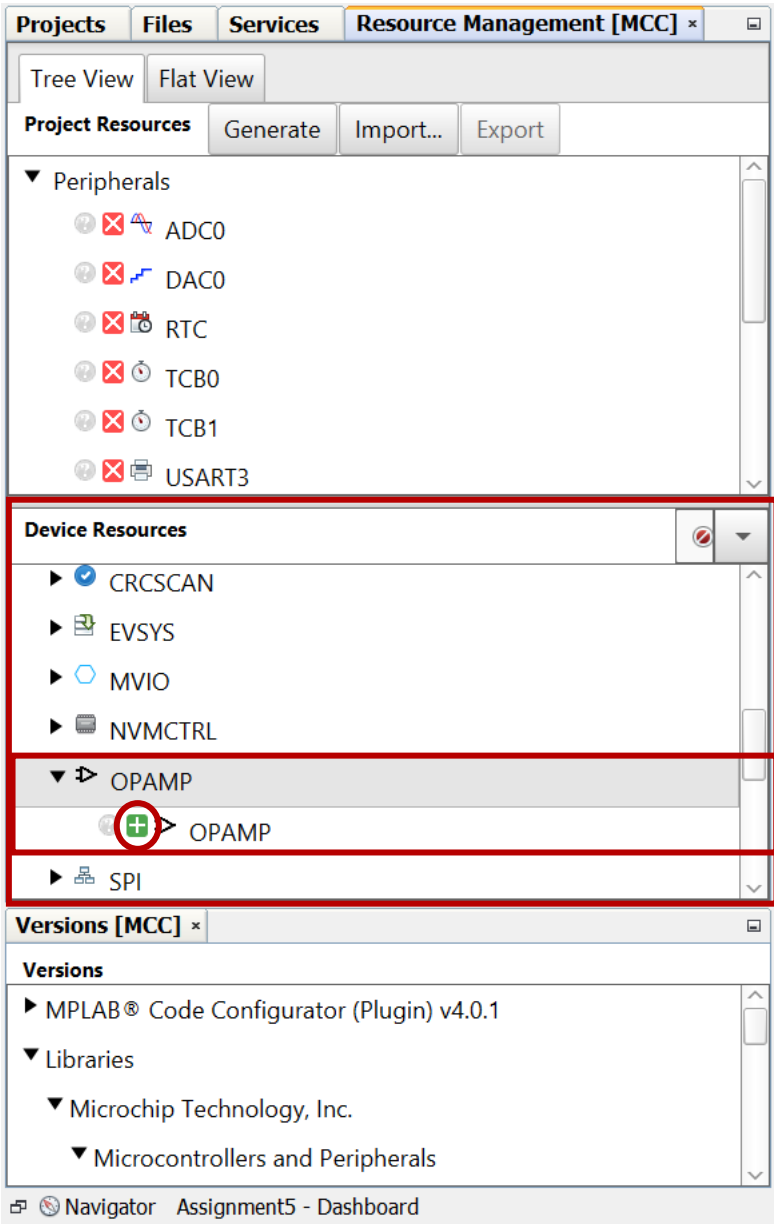
- 编辑 MCC 项目，将 OPAMP 配置为仪表放大器
- 在 MPLAB 数据可视化器中绘制仪表放大器的输入和输出

1. 按下工具栏中的 *MPLAB® Code Configurator v4: open/close*（MPLAB 代码配置器 v4：打开/关闭）按钮，以便能够编辑 MCC 项目。 图 6-2 显示了按钮的位置。

图 6-2. 任务 4：MCC 按钮

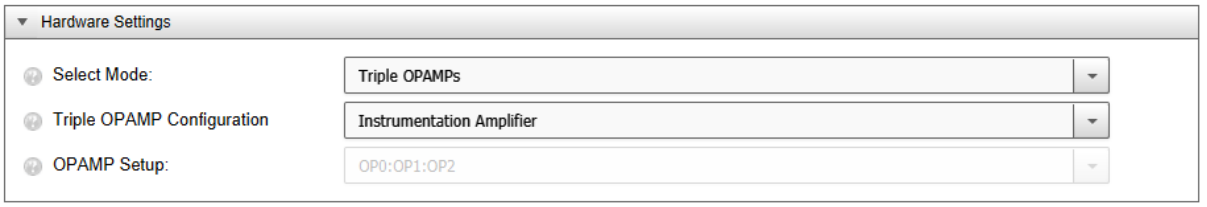
2. 在 Device Resources（器件资源）窗口中，向下滚动到 OPAMP 外设，按下箭头展开，然后按下绿色加号按钮将 OPAMP 外设添加到项目中，如图 6-3 所示。

图 6-3. 任务 4：添加 OPAMP



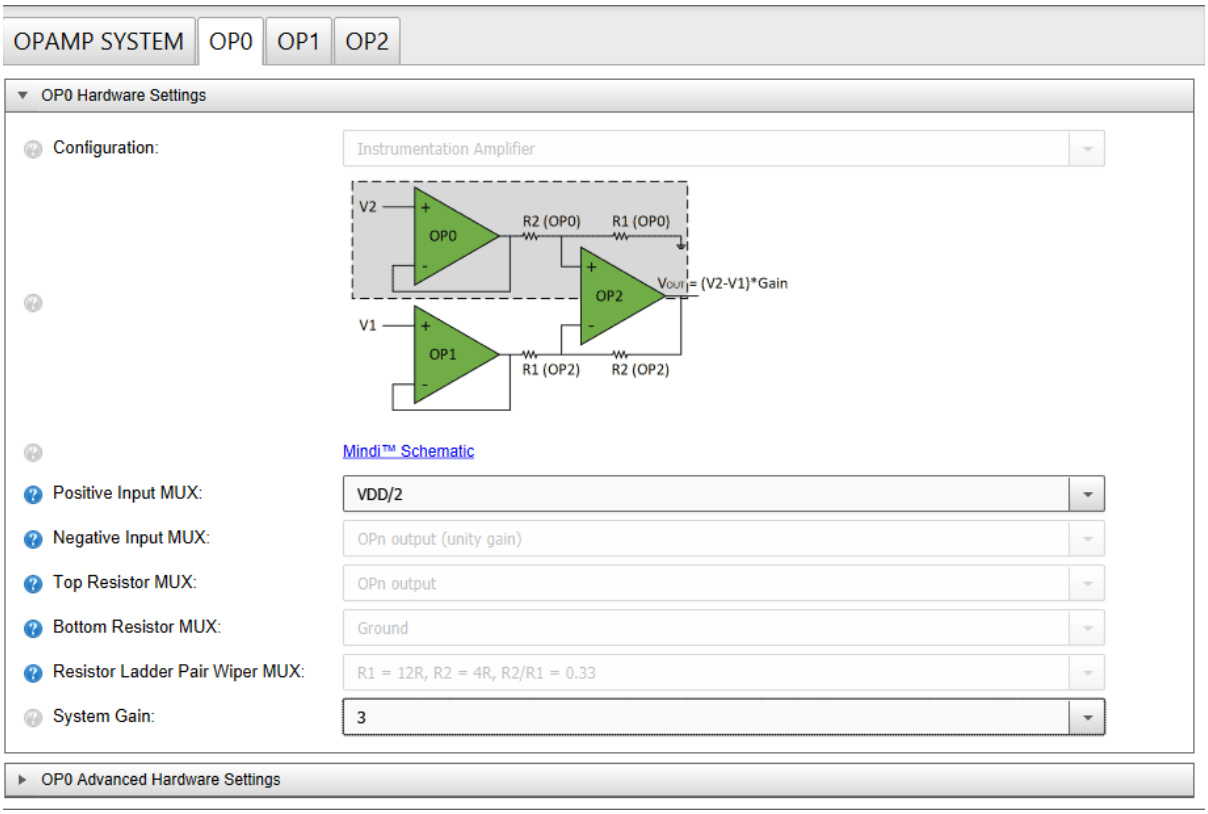
3. 在 OPAMP Hardware Settings（OPAMP 硬件设置）窗口中，将 Select Mode（选择模式）设置为 Triple OPAMPs（三 OPAMP），并将 Triple OPAMP Configuration（三 OPAMP 配置）设置为 Instrumentation Amplifier（仪表放大器），如图 6-4 所示。

图 6-4. 任务 4：硬件设置



4. 转到 OP0 选项卡，将 Positive Input MUX（同相输入多路开关）设置为 VDD/2，并将 System Gain（系统增益）设置为 3，如图 6-5 所示。

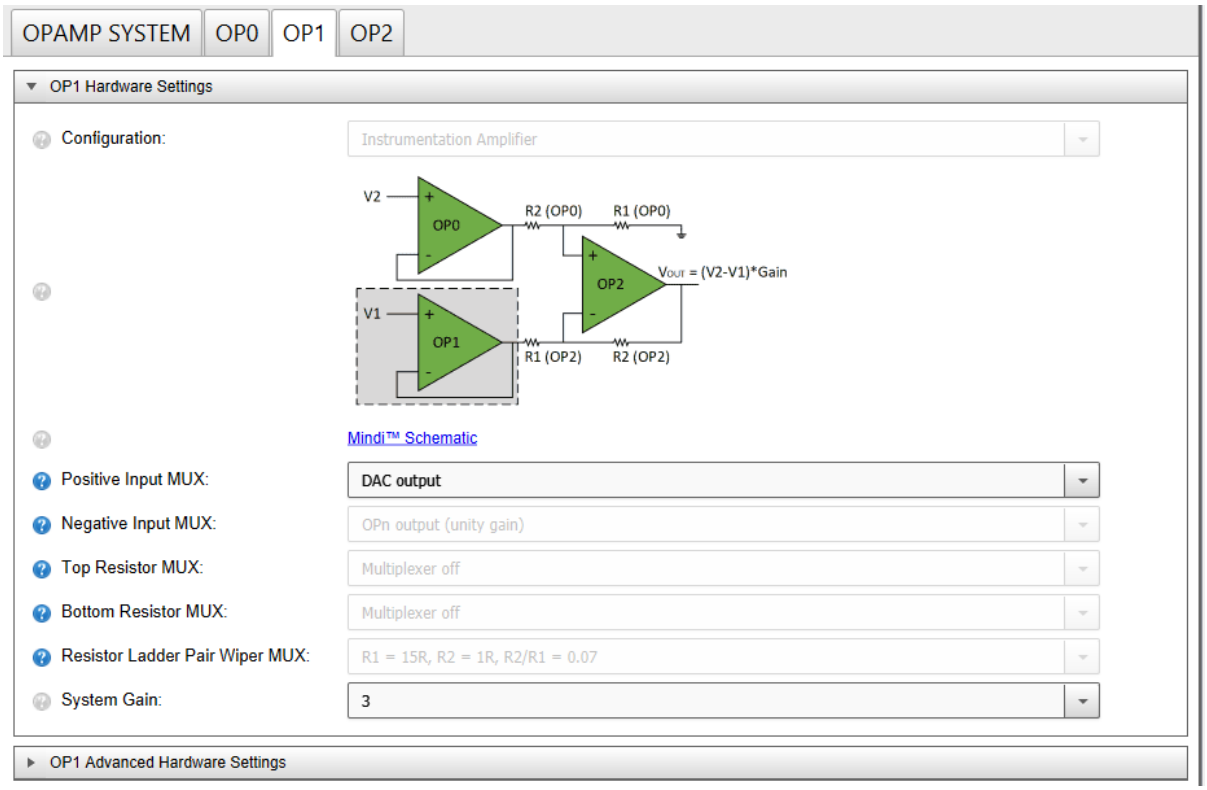
图 6-5. 任务 4：OP0



信息： OP0 和 OP2 的梯形电阻网络的设置组合用于控制仪表放大器的增益。为了便于使用，OP0 和 OP2 的梯形电阻网络由系统增益设置控制。系统增益设置由全部三个运放选项卡共享。

5. 转到 OP1 选项卡，将 Positive Input MUX 设置为 DAC output（DAC 输出），如图 6-6 所示。

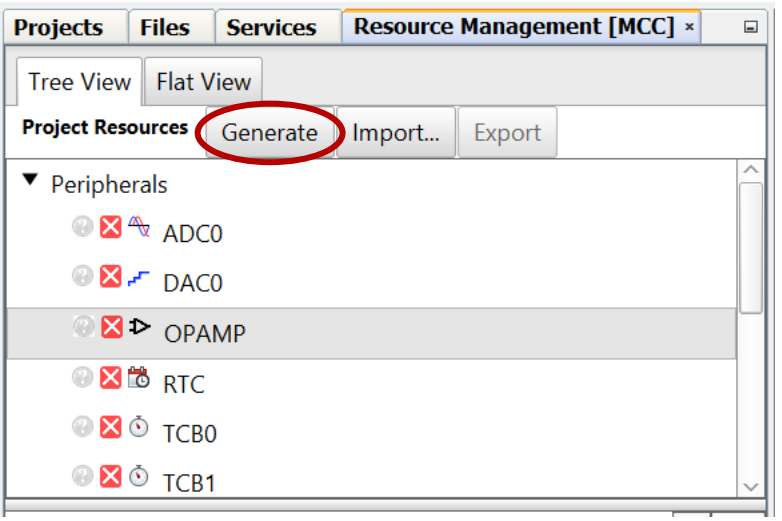
图 6-6. 任务 4：OP1



信息：OP2 没有可自定义的选项，因为它直接与其他运放相连，其梯形电阻网络设置由系统增益控制。

6. 按下 *Generate*（生成）按钮，重新生成添加了 OPAMP 外设的项目代码。*Generate* 按钮在 Project Resources 附近，如图 6-7 所示。

图 6-7. 任务 4：生成



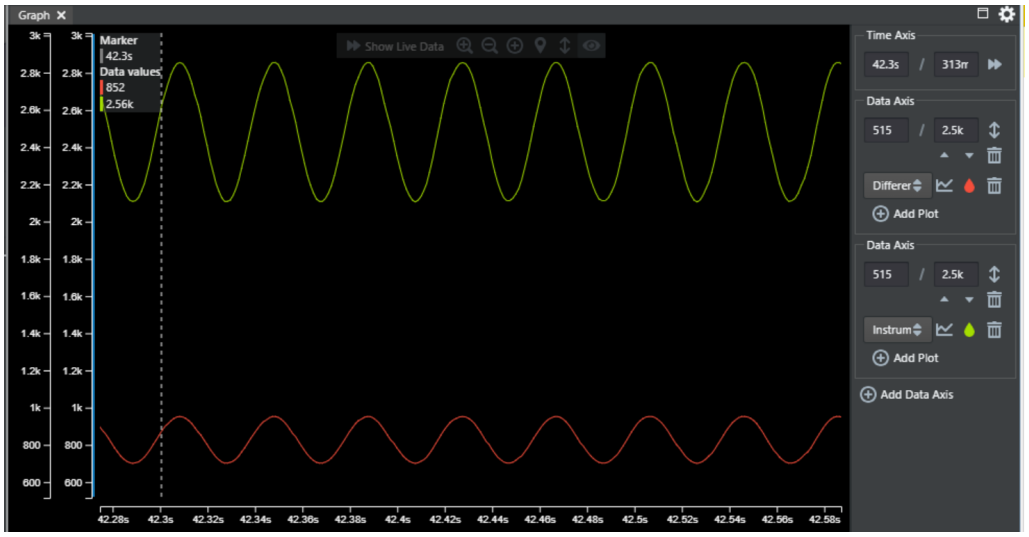
7. 验证项目是否成功编译，具体方法为按下 *Production→Build Project*（生产 → 编译项目）或按下 *F11*。

8. 按下 *Make and Program Device Main Project*以刷写器件。
9. 使用 MPLAB *数据可视化器*加载工作区 *Assignment4.json* 来绘制 DAC 输出与运放输出之间的关系图。

信息：DAC输出一个25 Hz的正弦波，峰峰值为256 mV，直流偏置为856 mV。

结果：在 MPLAB *数据可视化器*中绘制仪表放大器的输入和输出，如图 6-8 所示。红色波形是仪表放大器的差分输入，绿色波形是仪表放大器的输出。正如预期的那样，输出的大小是输入的三倍左右。

图 6-8. 任务 4：MPLAB *数据可视化器*



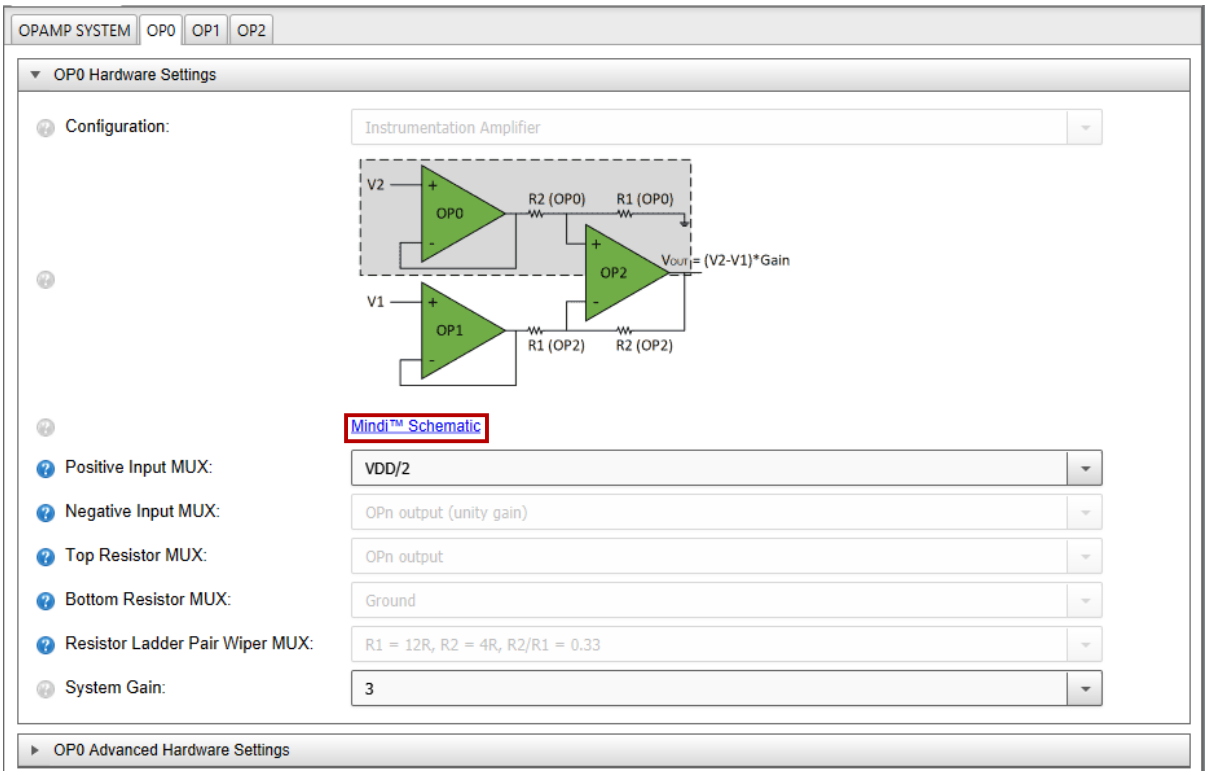
6.2. 在 Mindi 中仿真仪表放大器

操作：

- 下载包含 AVR DB 运放模型的预制仪表放大器仿真文件
- 运行仿真

1. 转到任何一个 OPn 选项卡，然后按下 Mindi Schematic 链接以进入仪表放大器的 GitHub 资源库。图 6-9 给出了链接的位置。

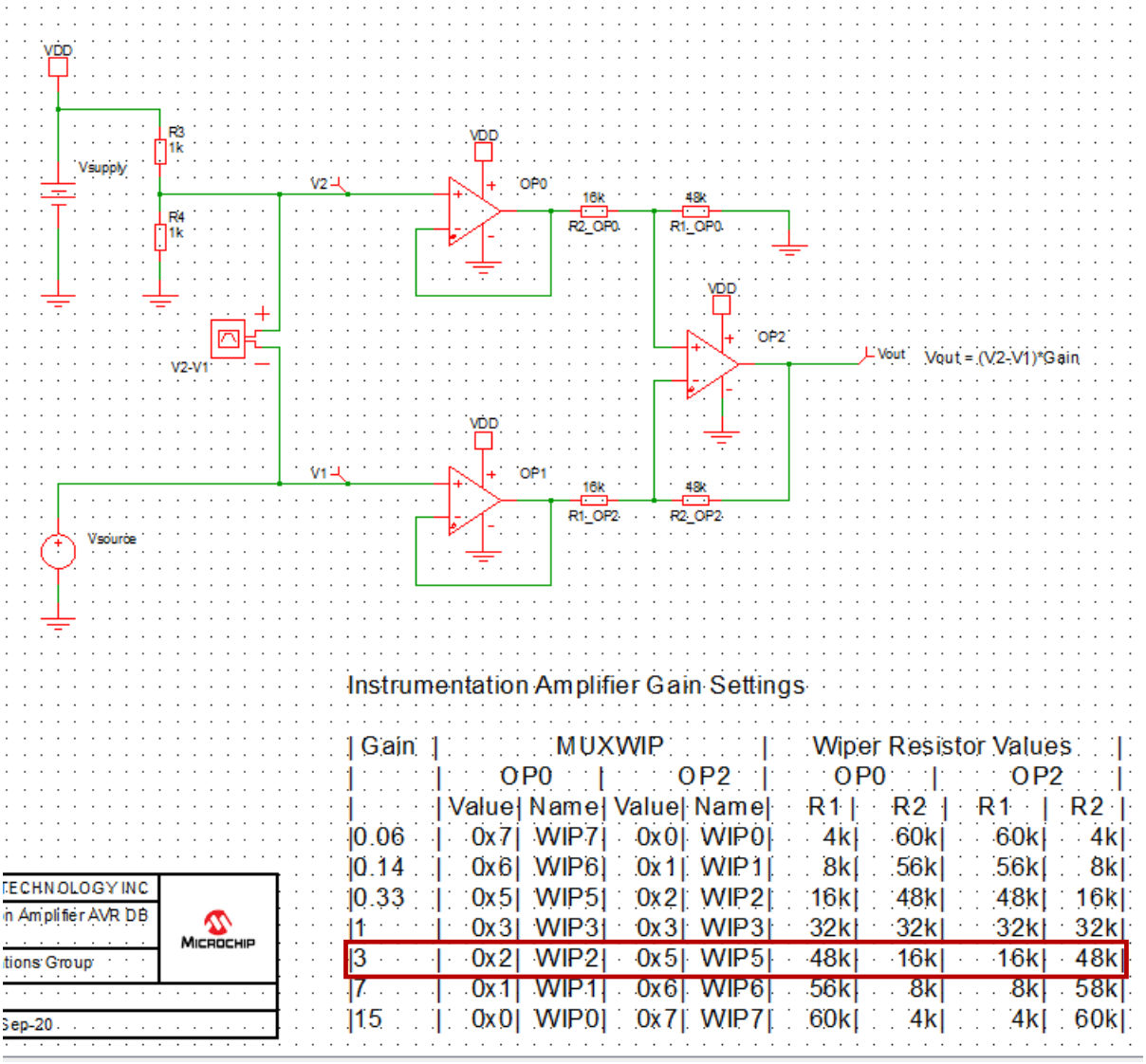
图 6-9. 任务 4：Mindi™链接



信息：Atmel START 中也有一个类似于 MCC 中给出的链接。

2. 在 GitHub 页面上，按下 *Releases*（发布）按钮，然后按下最新版本的 *Source Code (zip)*（源代码（zip））来下载原理图。
3. 选择一个目标位置对文件夹进行解压。
4. 打开 MPLAB Mindi，按下 *File→Open*（文件 → 打开）或 *Ctrl+o*，然后在 zip 文件的解压位置找到文件 *Instrumentation_Amplifier*。
5. 确保 OP0 和 OP2 的电阻值对于 3 倍增益来说是正确的，具体方法为查看图 6-10 所示的原理图下方的仪表放大器增益设置表。

图 6-10. 任务 4：仪表放大器增益

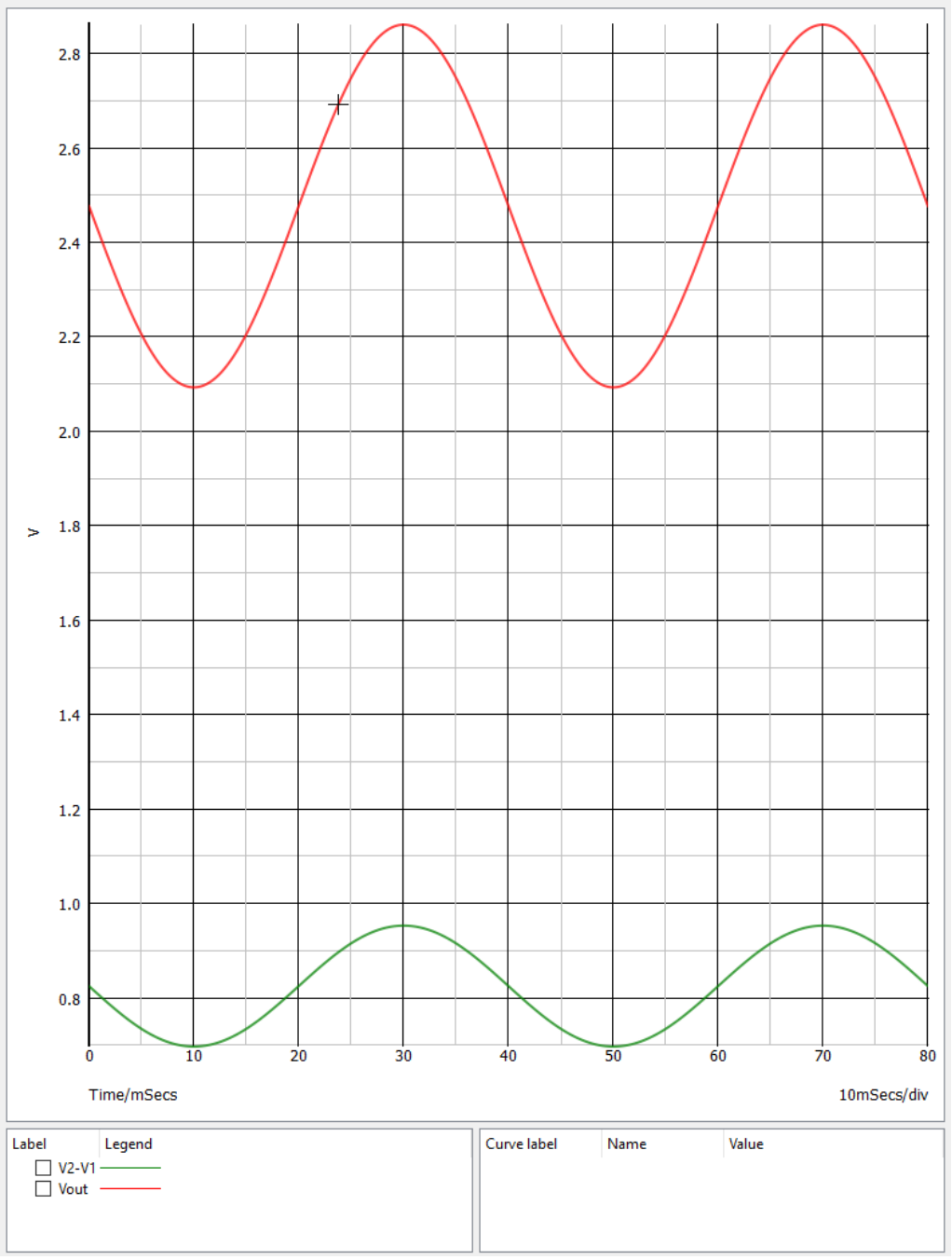


Gain	MUXWIP				Wiper Resistor Values			
	OP0		OP2		OP0		OP2	
	Value	Name	Value	Name	R1	R2	R1	R2
0.06	0x7	WIP7	0x0	WIP0	4k	60k	60k	4k
0.14	0x6	WIP6	0x1	WIP1	8k	56k	56k	8k
0.33	0x5	WIP5	0x2	WIP2	16k	48k	48k	16k
1	0x3	WIP3	0x3	WIP3	32k	32k	32k	32k
3	0x2	WIP2	0x5	WIP5	48k	16k	16k	48k
7	0x1	WIP1	0x6	WIP6	56k	8k	8k	58k
15	0x0	WIP0	0x7	WIP7	60k	4k	4k	60k

6. 按下 *Run Schematic*（运行原理图）按钮进行仿真。

结果：在 MPLAB Mindi 中绘制仪表放大器的输入和输出，如图 6-11 所示。红色波形是仪表放大器的输出，绿色波形是仪表放大器的差分输入。正如预期的那样，输出的大小是输入的三倍，绘图与 MPLAB *数据可视化器*中的绘图非常类似。

图 6-11. 任务 5：Mindi 结果



7. 任务 5：MVIO——OPAMP 作为 VDDIO2 的稳压电源

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以使用两个电压域 V_{DD} 和 V_{DDIO2}。V_{DD} 由 CNANO 稳压器供电，而 V_{DDIO2} 由 OP0 供电，OP0 配置为电压跟随器并使用 DAC 作为输入。之所以不直接使用 DAC 为 V_{DDIO2} 供电，是因为与运放相比，DAC 能提供的电流非常小。

将数模转换器（DAC）配置为生成 OP0 的输入电压。因此，DAC 的输出将是 V_{DDIO2} 的电压。

将模数转换器（ADC）配置为对 V_{DDIO2} 除以 10 之后的电压进行采样。测得的 V_{DDIO2} 采用数据流协议传输到 MPLAB 数据可视化器。

这项任务的起始点为 MPLAB X 项目*Assignment4.X*。

- **目标**
  - 了解 AVR DB 如何使用两个不同的电压域
  - 使用 ADC 通过内部连接来测量 V_{DDIO2}
  - 使用 MPLAB 数据可视化器绘制测得的 V_{DDIO2}

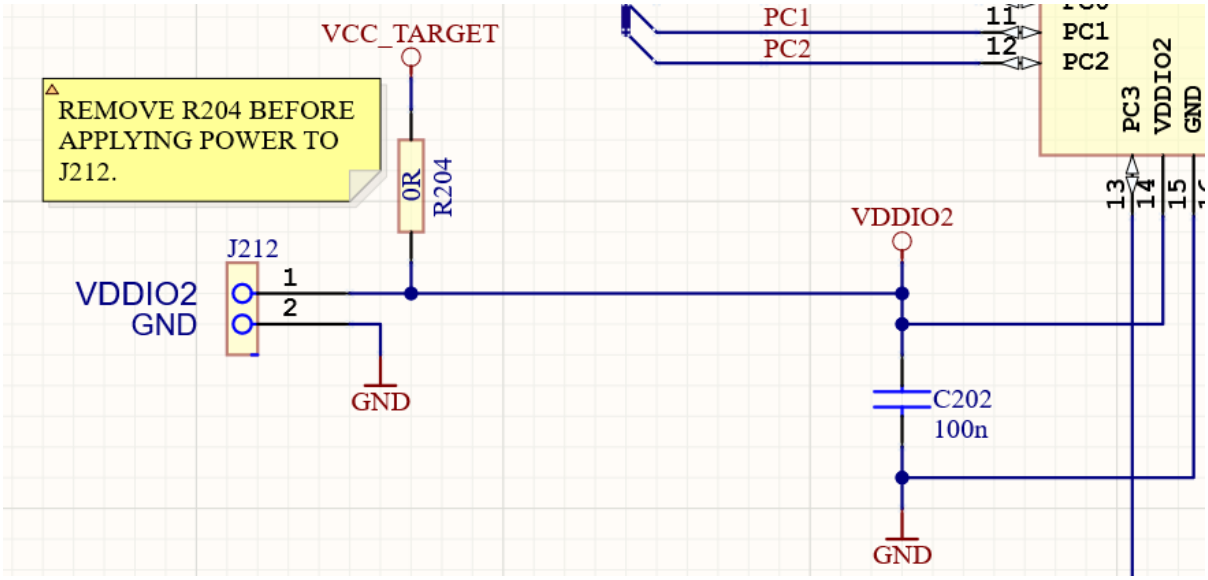
7.1. MVIO 硬件设置

操作：

- 修改 AVR128DB48 Curiosity Nano 以使用两个电压域
- 设置 AVR128DB48 Curiosity Nano 熔丝以使用两个电压域

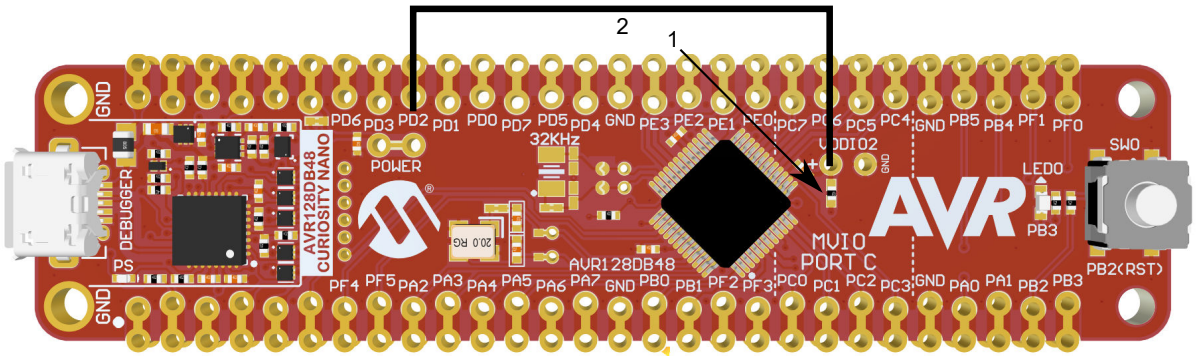
1. 移除 VDDIO2 标签正下方的电阻，以断开 VDDIO2 与 VDD 的连接。该电阻是图 7-1 中的电阻 R204，在图 7-2 中由箭头指出。

图 7-1. 任务 6：VDDIO2 原理图



2. 使用导线连接 PD2 与 VDDIO2+，如图 7-2 所示。

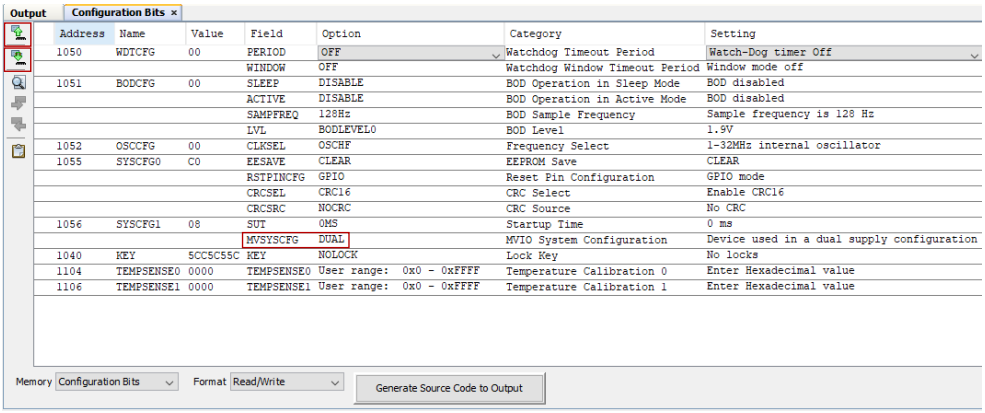
图 7-2. 任务 5：Curiosity Nano AVR128DB48



3. 将系统配置 1（SYSCFG）熔丝中的 MVSYSCFG 位域设置为 DUAL。

信息：要在 MPLAB X 中编程熔丝，请转到 *Production→Set Configuration Bits*（生产 → 设置配置位）以打开 Configuration Bits（配置位）窗口，如图 7-3 所示。按下 *Read Configuration Bits*（读取配置位）按钮以读回器件上当前的熔丝设置。将 MVSYSCFG 设置为 DUAL，然后按下 *Program Configuration Bits*（编程配置位）以将当前熔丝设置烧写到器件上。

图 7-3. 任务 5：MPLAB X 熔丝编程



结果：AVR128DB48 Curiosity Nano CNANO 已准备好使用两个电源域运行。

7.2. 使用 ADC 测量 VDDIO2

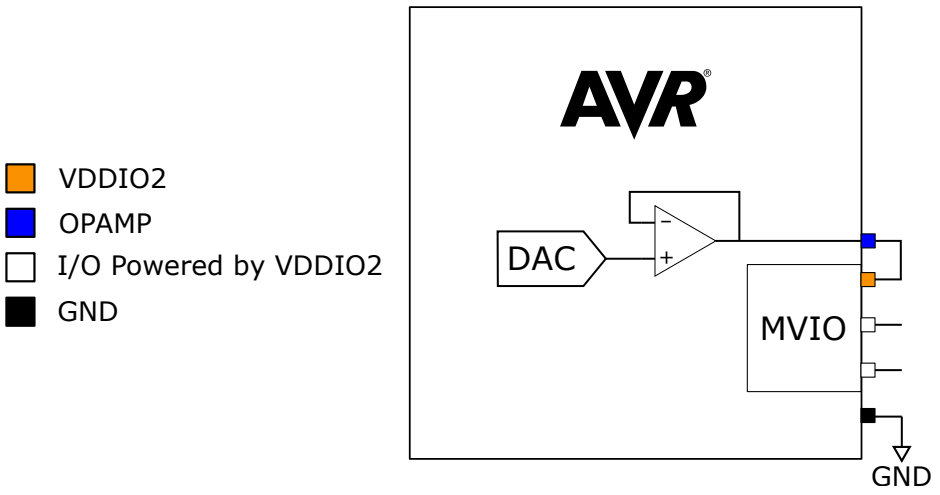
操作：

- 了解如何组合使用运放和 DAC 来构建 V_{DDIO2} 的电源
- 编辑 `adc0_init()`，将 ADC 配置为测量 V_{DDIO2}
- 使用 MPLAB *数据可视化器*绘制测得的 V_{DDIO2} 值

1. 研究 `op_amp_ini()` 函数以了解如何将 OP0 配置为电压跟随器并使用 DAC 作为输入。
2. 研究 `dac_init()` 函数以了解如何输出名为 `VDDIO2`（在 `dac.h` 中定义）的恒定电压。

信息：对 DAC 和运放应用这项任务中之前所作的硬件修改后，运放将成为 V_{DDIO2} 的电源。如图 7-4 所示。

图 7-4. 任务 5：运放作为电源



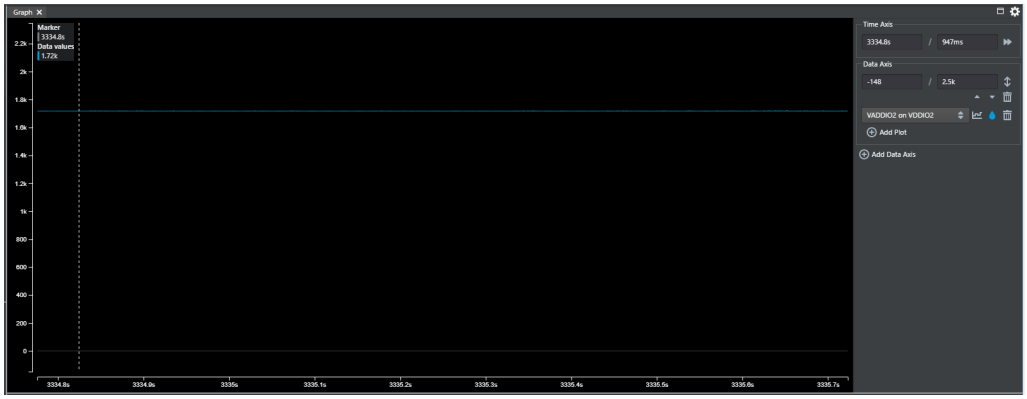
3. 在 `adc0_init()` 函数中，通过编写以下代码将 VDDIO2DIV10 设置为 ADC 的输入：

```
ADC0.MUXPOS = ADC_MUXPOS_VDDIO2DIV10_gc;
```

4. 验证解决方案/项目是否成功编译，具体方法为从 MPLAB X 的顶部菜单栏中选择*Build Main Project*，或者按下 *F11* 键。
5. 刷写器件，具体方法为从 MPLAB X 的顶部菜单栏中选择*Make and Program Device Main Project*。
6. 在 MPLAB *数据可视化器*中加载工作区 *Assignment6.json* 来绘制测得的 ADC 读数。

结果：MPLAB *数据可视化器*中将绘制 ADC 的测量结果，如图 7-5所示。请注意，单位是毫伏。

图 7-5. 任务 5：结果



8. 任务 6：MVIO——VDDIO2 故障检测

在这项任务中，将配置 AVR128DB48 Curiosity Nano 以使用两个电压域 V_{DD} 和 V_{DDIO2}。V_{DD} 由 CNANO 稳压器供电，而 V_{DDIO2} 由 OP0 供电，OP0 配置为电压跟随器并使用 DAC 作为输入。
将数模转换器（DAC）配置为生成 OP0 的输入电压。因此，DAC 的输出将是 V_{DDIO2} 的电压。
将模数转换器（ADC）配置为对 V_{DDIO2} 除以 10 之后的电压进行采样。测得的 V_{DDIO2} 采用数据流协议传输到 MPLAB 数据可视化器。
按下 SW0 时，DAC 的输出将减小，此时会检测到 VDDIO2 线路发生故障，进而触发 VDDIO2 中断。
这项任务的起始点为 MPLAB X 项目*Assignment6.X*。

- **目标**
  - 通过将电源电压降低到其规范值以下，向 V_{DDIO2} 引入一个错误
  - 了解 AVR DB 如何能够检测到 V_{DDIO2} 低于其最低要求的情况
  - 使用 ADC 通过内部连接来测量 V_{DDIO2}
  - 使用 MPLAB 数据可视化器绘制测得的 V_{DDIO2}

8.1. 设置 VDDIO2 故障检测

操作：
- 编辑 `mvio_init()` 以允许 VDDIO2 状态变化中断
- 编辑 `ISR(MVIO_MVIO_vect)` 以处理 VDDIO2 状态变化中断
- 使用 MPLAB *数据可视化器*绘制测得的 V_{DDIO2} 值

这项任务假定已完成任务 6 中所述的硬件设置步骤。如果尚未完成，可以在 MVIO 硬件设置中找到分步说明。

信息：DAC、ADC 和 OPAMP 外设的配置方式均与任务 6 中相同。有关设置说明，请参见使用 ADC 测量 VDDIO2。

1. 允许 VDDIO2 状态变化中断，具体方法为编辑 `mvio_init()` 以包含以下代码：

```
MVIO.INTCTRL = MVIO_VDDIO2IE_bm;
```

2. 通过清零中断标志和切换 LED0 来处理中断，具体方法为将以下代码添加到 `ISR(MVIO_MVIO_vect)` 中：

```
MVIO.INTFLAGS = MVIO_VDDIO2IF_bm;
LED0_toggle();
```

3. 了解如何通过降低运放输出在 SW0 被按下时注入 V_{DDIO2} 故障，具体可查看 `gpio.c` 中的 `ISR(PORTB_PORT_vect)`。

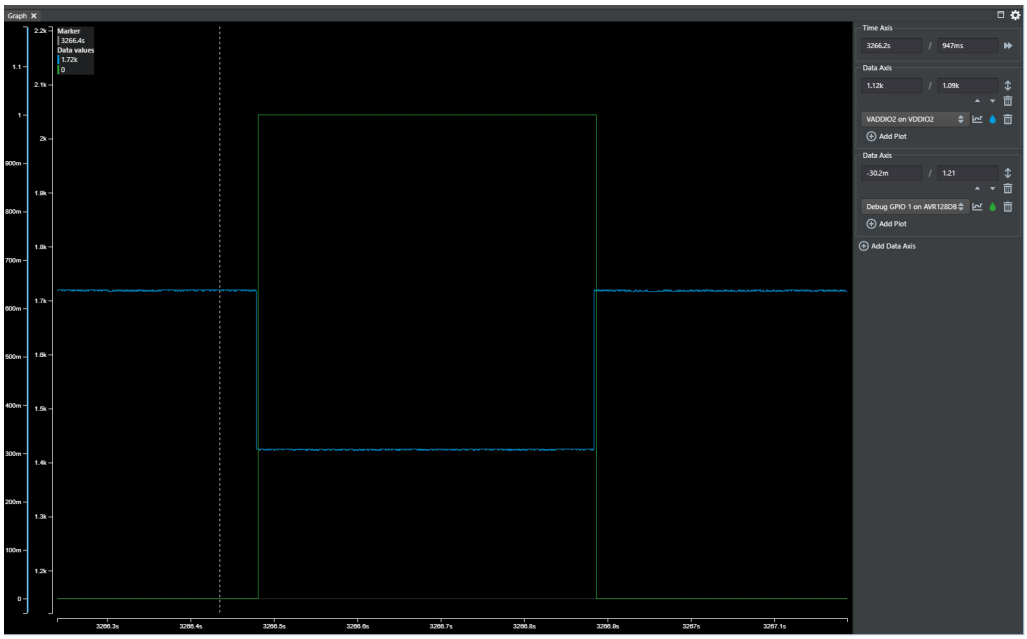
信息：当 V_{DDIO2} 低于可接受的范围时，VDDIO2 状态位会变为低电平，进而触发 VDDIO2 状态变化中断。当 V_{DDIO2} 低于可接受的范围时，PORTC 上的所有引脚都为三态。当 V_{DDIO2} 再次升高时，将再次装载 PORTC 寄存器的值。

4. 验证解决方案/项目是否成功编译，具体方法为从 MPLAB X 的顶部菜单栏中选择*Build Main Project*，或者按下 *F11* 键。
5. 刷写器件，具体方法为从 MPLAB X 的顶部菜单栏中选择*Make and Program Device Main Project*。
6. 在 MPLAB *数据可视化器*中加载工作区 Assignment7.json 来绘制测得的 ADC 读数和 LED0 的状态。

信息：LED0 为低电平有效，因此 MPLAB *数据可视化器*中的绘图将在 LED 点亮时显示 `0`，在 LED 熄灭时显示 `1`。

结果：MPLAB *数据可视化器*中将绘制 ADC 测量结果和 SW0 状态，如图 8-1 所示。可以看出，按下 SW0 后，只要 V_{DDIO2} 低于可接受的值，就会触发 VDDIO2 状态变化中断，同时 LED0 线路会变成高电平。再次按下 SW0 时，如果 V_{DDIO2} 恢复至可接受的范围内，则 LED0 会变回低电平。蓝线是 V_{DDIO2} 读数，绿线是 LED0。

图 8-1. 任务 6：结果



9. 版本历史

文档版本	日期	备注
A	2020 年 11 月	文档初始版本

Microchip信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN：979-8-3371-2418-6

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在按照操作规范正常使用的情况下，Microchip 系列产品非常安全。
- Microchip 重视并积极保护其知识产权。任何试图破坏 Microchip 产品代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》并予以严禁。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。