# AVR® DB Family

# Training Getting Started with AVR® DB OPAMP, XOSHF and MVIO - Atmel Studio

## Prerequisites

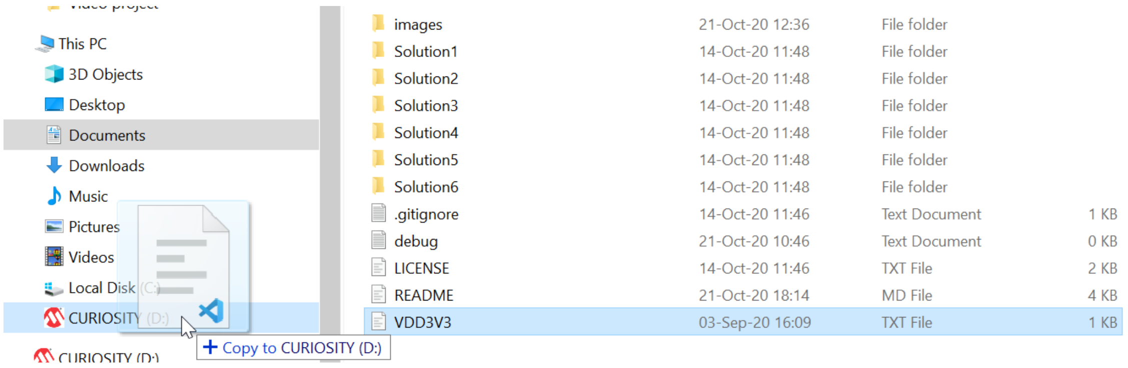
Authors: Martin Mostad, Martin Thomaz, Microchip Technology Inc.

- **Hardware Prerequisites**
  - The Microchip AVR128DB48 Curiosity Nano Curiosity Nano Evaluation Kit EV35L43A
  - Micro-USB cable (Type-A/Micro-B)
  - Logic Analyzer or oscilloscope with 100 MHz sampling speed (only needed for Assignment 2)
- **Software Prerequisites**
  - Atmel Studio 7.0.2397 or later
  - Atmel Studio Dx_DFP version 1.4.308 or above
  - Atmel Start version 1.8.457 or later (only needed for Assignment 4)
  - MPLAB® Data Visualizer Stand-alone version 1.1.793 or above
  - MPLAB® Mindi™ Analog Simulator
- **Other**
  - The Atmel Studio projects are available at:

    View Code Example on GitHub
    Click to browse repository

  - The AVR128DB48 Curiosity Nano needs to be running at 3.3V. This can be achieved by dragging the VDD3V3.txt file located in the GitHub repository over to the CURIOSITY memory device, as shown in Figure 1.

**Figure 1. Setting AVR128DB48 Curiosity Nano to 3.3V**



| | | | |
|---|---|---|---|
| images | 21-Oct-20 12:36 | File folder | |
| Solution1 | 14-Oct-20 11:48 | File folder | |
| Solution2 | 14-Oct-20 11:48 | File folder | |
| Solution3 | 14-Oct-20 11:48 | File folder | |
| Solution4 | 14-Oct-20 11:48 | File folder | |
| Solution5 | 14-Oct-20 11:48 | File folder | |
| Solution6 | 14-Oct-20 11:48 | File folder | |
| .gitignore | 14-Oct-20 11:46 | Text Document | 1 KB |
| debug | 21-Oct-20 10:46 | Text Document | 0 KB |
| LICENSE | 14-Oct-20 11:46 | TXT File | 2 KB |
| README | 21-Oct-20 18:14 | MD File | 4 KB |
| VDD3V3 | 03-Sep-20 16:09 | TXT File | 1 KB |

- **Estimated Completion Time: 210 minutes**

## Table of Contents

# 1. Introduction

This training document contains assignments that give a general introduction to the new features of the AVR® DB Family. Each assignment starts with a basic introduction, then the functionality is verified through the MPLAB® *Data Visualizer*. Every assignment is provided with a preconfigured code, where most of the code is already in place, and only essential modules for the specific assignment need to be configured. Each assignment comes with a solution.

The new AVR® DB Family of microcontrollers comes with three features that will be highlighted in this training. The features are a new Clock Controller (CLKCTRL), Analog Signal Conditioning (OPAMP), and Multi-Voltage I/O (MVIO).

The new CLKCTRL has a new clock source called the High Frequency Crystal Oscillator (XOSCHF) and Clock Failure Detection (CFD). The XOSCHF enables the use of an external crystal or an external clock signal up to 32 MHz. This can be used as a clock source for the Main Clock (CLK_MAIN), the Real-Time Counter (RTC), and the 12-bit Timer/Counter Type D (TCDn). The CFD feature can be used to detect if the output from a clock source stops and can switch the Main Clock to a different clock source.

The OPAMP peripheral features up to three internal operational amplifiers (op amps) as well as a resistor ladder for each op amp. The resistor ladder can be used to create a variety of classical op amp configurations. The main purpose of op amps is to condition the analog signals before an acquisition or to provide the necessary output drive in control applications.

The MVIO allows a subset of the I/O pins to be powered by a different I/O voltage domain called $V_{DDIO2}$. The rest of the I/O pins runs on $V_{DD}$, which may remove the need for external level shifters when communicating with other devices running on a different target voltage.

This training covers the following topics:

- Assignment 1: CLKCTRL - External High-Frequency Oscillator (XOSCHF) and Clock Failure Detection (CFD)
- Assignment 2: CLKCTRL - High Frequency TCD using PLL
- Assignment 3: OPAMP - Voltage Follower
- Assignment 4: OPAMP - Non-Inverting PGA
- Assignment 5: MVIO - OPAMP as a Regulated Power Supply for VDDIO2
- Assignment 6: MVIO - VDDIO2 Failure Detection

## 2. Icon Key Identifiers

The following icons are used in this document to identify different assignment sections and to reduce complexity.

**Info:** Delivers contextual information about a specific topic.

**Tip:** Highlights useful tips and techniques.

**To do:** Highlights objectives to be completed.

**Result:** Highlights the expected result of an assignment step.

⚠WARNING Indicates important information.

**Execute:** Highlights actions to be executed out of the target when necessary.

# 3. Assignment 1: CLKCTRL - External High-Frequency Oscillator (XOSCHF) and Clock Failure Detection (CFD)

In this assignment, the AVR128DB48 Curiosity Nano will be configured to use the external high-frequency oscillator (XOSCHF) as the main clock source.

The clock failure detection will be configured to detect failures on the main clock. A failure on the main clock will result in a Non-Maskable Interrupt (NMI).

A failure will be triggered by the software when the switch (SW0) on the AVR128DB48 Curiosity Nano is pressed.

The AVR128DB48 Curiosity Nano will blink an LED with a frequency dependent on the main clock to show how the main clock source will be switched back to the default internal source when an error is detected.

The starting point for this assignment is the Atmel Studio Solution *Assignment1* found in *Assignment1*.

- **Objectives**
  - Configure the main clock yo use XOSCHF as its clock source
  - Configure the CFD to generate an NMI on the failure of the main clock
  - Trigger a failure on the main clock using the built-in test functionality of the CFD

## 3.1 Configure the Main Clock to Use XOSCHF

**To do:**
- Edit the `CLOCK_XOSCHF_crystal_init()` function so that the main clock uses the XOSCHF as a clock source

**Info:**
Almost all the registers in the Clock Controller (CLKCTRL) peripheral are under Configuration Change Protection (CCP), which means that for write to this register, a certain key must first be written to the CPU.CCP register, followed by a write to the protected bits within four CPU instructions. To make sure this criterion is met when writing to the protected registers, the function `ccp_write_io()` is used, which can be found in the *avr/cpufunc.h* file. The function takes two arguments: A *uint8_t* pointer and the values to be written to the register. The register macros provided by the *io.h* file are not *uint8_t* pointers, so typecasting is therefore needed. A useful template showing how to use the function is:

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

1. Configure the XOSCHF to use an external crystal, have a start-up time of four thousand cycles, a frequency range of 16 MHz, to always run, and enable XOSCHF. All this can be done by adding the following to `CLOCK_XOSCHF_crystal_init()`:

```
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);
```

**Info:** The frequency range setting determines the maximum frequency for the crystal that is supported. The larger the range, the higher the current consumption.

**Info:** The XOSCHF is set to always run so we can verify that the clock source is stable before being used. This setting will later be turned off to save power.

2. Wait until the external high-frequency crystal is stable by adding:

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

3. Set the main clock source to be XOSCHF and enable the clock output pin by writing:

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

**Info:** By enabling the clock out pin, it is possible to observe the main clock frequency on the CLKCOUT (PA7) pin.

4. Wait for the main clock to switch successfully by polling the Main Clock Oscillator Change (SOSC) bit field in Main Clock Status (CLKCTRL.MCLKSTATUS) register:

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

5. To save power, clear the Run Standby (RUNSTDBY) bit, so the oscillator is not running when it is not needed:

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

**Result:** After adding all the code to configure XOSCHF, the `CLOCK_XOSCHF_crystal_init()` function will look like this:

```
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 */
}
```

**Info:** The main function will be as follows, containing a call to the CLOCK_XOSCHF_crystal_init() function.

```
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 Configure the CFD to Detect Failures on the Main Clock

**To do:**
- Edit the `CLOCK_CFD_CLKMAIN_init()` function so that the CFD is set up to monitor the main clock and generates an NMI upon clock failure
- Set up the NMI handler
- Create a main clock failure using the software test

1. Enable the CFD and set the main clock as the source by adding:
```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_CLKMAIN_gc
                 | CLKCTRL_CFDEN_bm);
```
to `CLOCK_CFD_CLKMAIN_init()`.
2. Enable the CFD interrupt and set the interrupt type to NMI by writing:
```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_INTTYPE_bm
                 | CLKCTRL_CFD_bm);
```

**Info:** The CFD can create a normal interrupt or non-maskable interrupt depending on the setting in the Interrupt Type (INTTYPE) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) register. The advantage with an NMI is that the interrupt will trigger even if the microcontroller is in an interrupt or the global interrupt is disabled. This ensures that the interrupt will be executed straight away, and the error can be handled straight away. The only way to exit an NMI is by a device reset.

3. In the `ISR(NMI_vect)`, add the code to check if a CFD triggered the NMI and to blink the LED:
```
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 */
    }
```

**Info:** There is only one interrupt vector for all the NMIs, so to be sure which NMI caused the interrupt, check the respective interrupt flags.

4. In the `ISR(PORTB_PORT_vect)`, add the code to trigger a clock failure when SW0 is pressed:
```
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL.MCLKCTRLC | CLKCTRL_CFDTST_bm);
```

**Info:** When setting the Clock Failure Detection Test (CFDTST) bit in the Main Clock Control C (MCLKCTRL) register, hardware simulates a clock failure to test the CFD feature.

5. Verify that the solution/project builds by selecting the *Build* → *Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
6. Flash the device by selecting the *Debug* → *Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.

**Result:** After pressing SW0, the LED0 will start blinking with a lower frequency. This because the main clock source is switched to the start-up clock source, and the Main Clock Control B (CLKCTRL.MCLKCTRLB) register will be written to the reset value when a clock failure is detected. The number of clock ticks needed to create the `delay_ms(200)` is calculated using the XOSCHF clock frequency of 16 MHz, resulting in the `delay_ms` being four times as slow when the clock source is switched to the 4 MHz internal clock.

**Info:** The start-up clock source is decided by the Clock Select (CLKSEL) bit field in the Oscillator Configuration (OSCCFG) fuse.

⚠WARNING As the MCLKCTRLB register is written to its default values by the CFD, the clock will not be available on the CLKOUT pin until it is set by the user again.

# 4. Assignment 2: CLKCTRL - High Frequency TCD using PLL

In this assignment, the AVR128DB48 Curiosity Nano will be configured to create a high-resolution PWM signal using TCD with the PLL as a clock source.

The PLL will increase the clock frequency of OSCHF.

The output from the PLL will be used as a clock source for TCD, which will create a high-resolution PWM signal.

The starting point for this assignment is the Atmel Studio Solution *Assignment2* found in *Assignment2*.

- **Objectives**
  - Configure the PLL to use OSCHF as a clock source
  - Configure the TCD to use the PLL as a clock source and output a PWM signal
  - Visualize the PWM signal using a logic analyzer or an oscilloscope

## 4.1 Configure TCD to Use the PLL as a Clock Source

**To do:**

- Edit the `CLOCK_OSCHF_crystal_PLL_init()` to configure the PLL to use OSCHF as a source and multiply it by 3x
- Edit the `TIMER_TCD0_init()` so the TCD uses the PLL as a clock source and creates a PWM signal
- Plot the PWM signal using a logic analyzer

1. Edit `CLOCK_OSCHF_crystal_PLL_init()` so it sets the OSCHF frequency to 16 MHz by adding:

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

**Info:** The minimum input frequency to the PLL is 16 MHz, and it has a maximum output of 48 MHz.

2. Set the PLL multiplication factor to 3x by adding the following to `CLOCK_OSCHF_crystal_PLL_init()`:

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

**Info:** The PLL has two input sources: The OSCHF and XOSCHF. The default source is OSCHF, but the XOSCHF can be configured by setting the Select Source for PLL (SOURCE) bit in the PLL Control A (PLLCTRLA) register.

3. Edit `TIMER_TCD0_init()` to include the following to set the PLL as the clock source with no prescaler:

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

4. Set the TCD to one ramp PWM mode by writing:

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

5. Make the PWM signal available on the pins by adding:

```
/*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. Enable the TCD by first ensuring that the Enable Ready (ENRDY) bit in the Status (TCDn.STATUS) register is set to ‘`1`’ and then setting the ENABLE bit in Control A (TCDn.CTRLA) register:

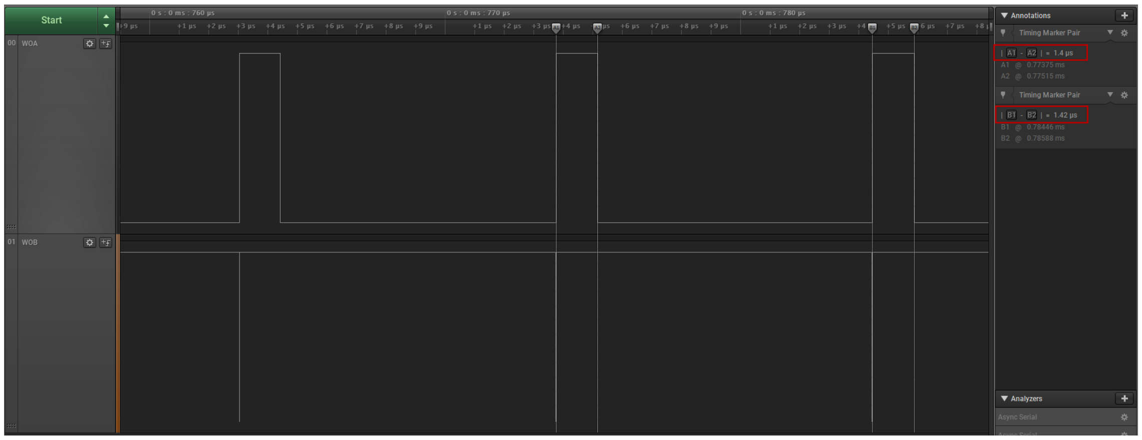
```
/* Wait for synchronization */
while(!(TCD0.STATUS & TCD_ENRDY_bm))
{
    ;
}
/* Enable TCD0 */
TCD0.CTRLA |= TCD_ENABLE_bm;
```

**Info:** In addition to the configurations that have been set, the TCD is set to increment the duty cycle of WOA by one bit after every period. See `ISR(TCD0_OVF_vect)` how this is implemented.

7. Verify that the solution/project builds by selecting the *Build → Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
8. Flash the device by selecting the *Debug → Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.
9. Plot the PWM signal using a Logic Analyzer the output for WOA is available on PA4, while the output for WOB is available on PA5.

**Result:** The PLL and TCD are combined to create a very high-resolution PWM signal. In Figure 4-1, you can seen that the duty cycle is increased by about 20 ns from 1.4 µs to 1.42 µs after each period showing the high-resolution of the PWM signal.

**Figure 4-1. Assignment 2: Result**



| A1 - A2 | = 1.4 µs
A1 @ 0.77375 ms
A2 @ 0.77515 ms

| B1 - B2 | = 1.42 µs
B1 @ 0.78446 ms
B2 @ 0.78588 ms

# 5. Assignment 3: OPAMP - Voltage Follower

In this assignment, the AVR128DB48 Curiosity Nano will be configured to use the op amp OP0 in the Analog Signal Conditioning (OPAMP) peripheral as a voltage follower.

The Digital-to-Analog Converter (DAC) is configured to create a sinusoidal signal, which will be used as an input to the op amp.

The Analog-to-Digital Converter (ADC) is configured to sample the output of the op amp.

Both the input and output of the op amp are streamed over to the MPLAB® Data Visualizer using the data streamer protocol.

The starting point for this assignment is the Atmel Studio Solution *Assignment3* found in *Assignment3*.

- **Objectives**
  - Understand the necessary steps to configure an op amp instance in the OPAMP peripheral
  - Configure OP0 as a voltage follower
  - Plot the input and output from OP0 using the MPLAB® Data Visualizer

## 5.1 Configure OP0 as a Voltage Follower

**To do:**
- Edit the `OPAMP0_init()` so it configures OP0 to be a voltage follower
- Plot the input and output of OP0 in the MPLAB® Data Visualizer

1. Edit `OPAMP0_init()` to set the time base for the OPAMP peripheral by writing:

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

**Info:** For internal timing purposes, the Timebase (OPAMP.TIMBASE) register needs the maximum number of CLK_PER cycles to achieve a timing interval equal to or larger than 1 µs. The rule to determine what number to write into the TIMBASE is: Determine the number of CLK_PER cycles equal to 1 µs. If this is an integer, subtract one. If not, round it down. The following macro can be used to calculate the value for the TIMEBASE register:

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

2. Set the output mode to normal and the Always On (ALWAYSON) bit for OP0 by writing:

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

**Info:** Each op amp instance in the OPAMP peripheral has an independent output mode. The two modes are off and normal. In the off mode, the output driver for the op amp is off but can be overwritten by the DRIVE event. In the normal mode, the driver is always on.

**Info:** Each op amp instance in the OPAMP peripheral can be individually turned on or off. This can be achieved either by setting/clearing the ALWAYSON bit or by the ENABLEn/DISABLEn events. If events are used to control the op amp, the ALWAYSON bit has to be cleared.

⚠WARNING The Event Enable (EVENTEN) bit must be set in the OPnCTRA register to control the op amp using events

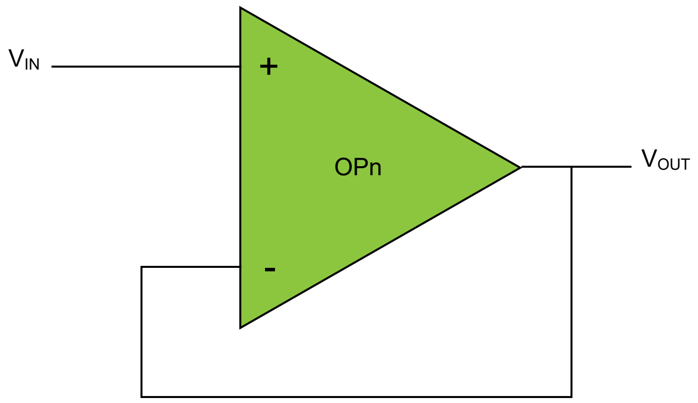
3. Configure OP0 to be a voltage follower with the DAC as an input by writing:

```
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;
```

**Info:**

**Figure 5-1. Op Amp in Voltage Follower Configuration**



To achieve a voltage follower configuration, the output of the op amp needs to be connected directly to the negative input. This is done by setting the Multiplexer for Negative Input (MUXNEG) bit field in the Op Amp n Input Multiplexer (OPnMUX) register to OUT. The output of the DAC and the positive input to OP0 are connected through an internal channel by setting the Multiplexer for Positive Input (MUXPOS) bit field in the OPnMUX register to DAC. There is no need to use the resistor ladder in the voltage follower, so all the bit fields in the Op Amp n Resistor Ladder Multiplexer (OPnRESMUX) register can be set to their default values.

4. Set the settling time by writing:

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

**Info:** The value in the Op Amp n Settle Timer (OPnSETTLE) register is the number of microseconds needed for the output of the op amp to settle. This time is highly application dependent. If it is not known, it is recommended to set it to the max value `0x7F`. This value, together with the value in the TIMEBASE register, is used by an internal timer to determine when to generate the READYn event and set the SETTLED flag in the OPnSTATUS register.

5. Enable the OPAMP peripheral by writing:

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

6. Wait for the op amp to settle before leaving the initialization by checking the Op Amp has Settled (SETTLED) bit field in the Op Amp n Status (OPnSTATUS) register:

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

7. Verify that the solution/project builds by selecting the *Build → Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
8. Flash the device by selecting the *Debug → Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.

**Result:** The device is now flashed and ready to start streaming data to the MPLAB® *Data Visualizer*.

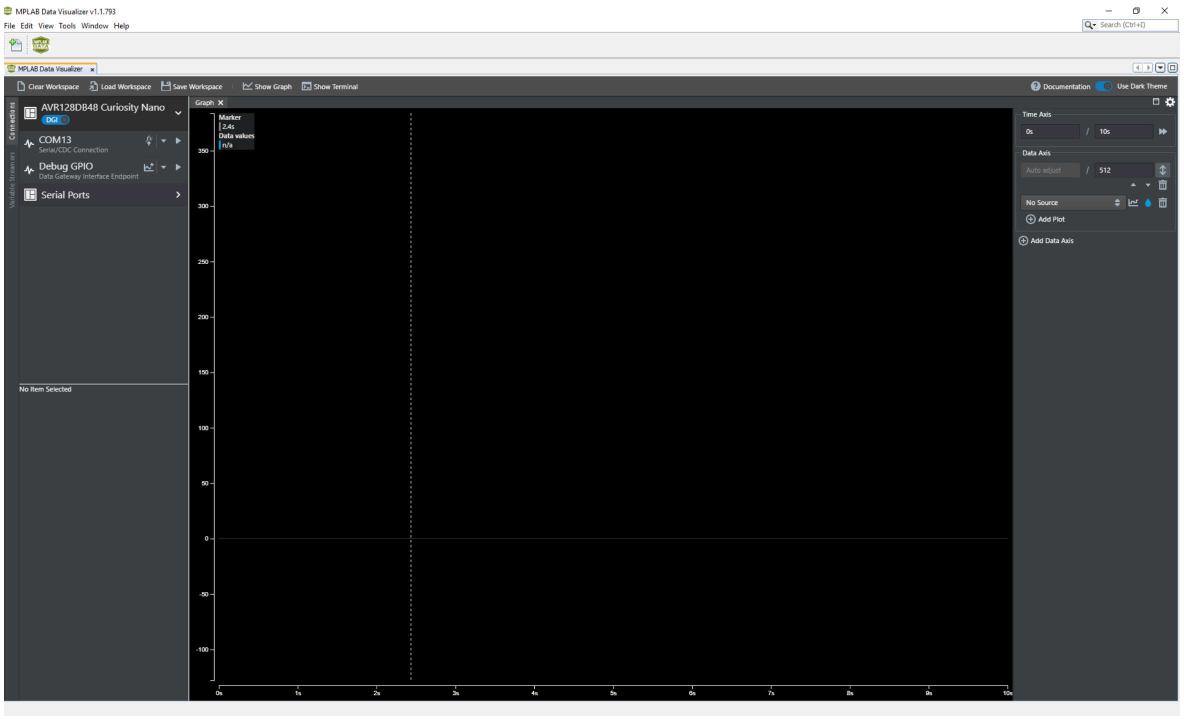
## 5.2 Plot Graph In MPLAB® Data Visualizer

The MPLAB® *Data Visualizer* is a program used to process and visualize data from a running embedded target. The program may be accessed as an MPLAB X IDE plugin or a stand-alone program. In this assignment, the *Data Visualizer* will be configured to graph the input and output of OP0 received over USART. The configuration is done through a saved workspace, and the basics of how to display the data are explained. To find out how to set up your own workspace, a detailed guide can be found by clicking on the *Documentation* button in MPLAB *Data Visualizer*.

**To do:** Configure MPLAB *Data Visualizer* to graph received OP0 input and output samples.

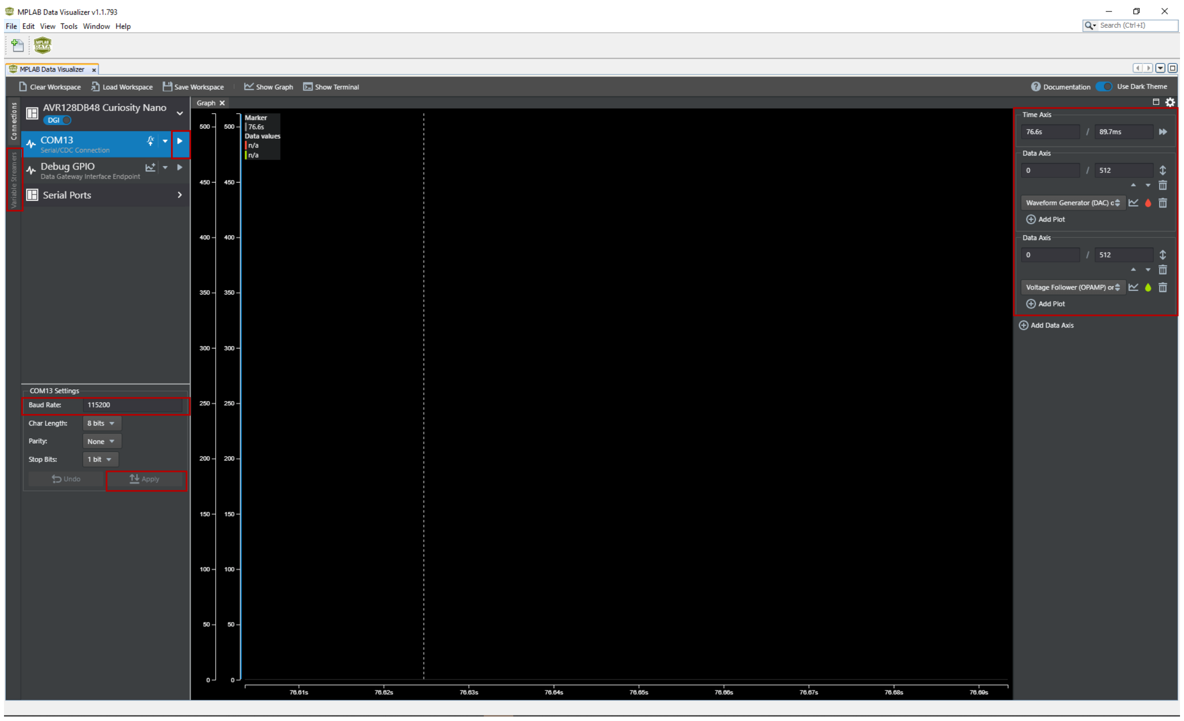
1. Open the program and plug in an already flashed device. Make sure the COM-port used for USART communication is not already in use. The start screen will be similar to Figure 5-2.

**Figure 5-2. Assignment 3: MPLAB® Data Visualizer Starting Page**



2. Load the workspace. Press the *Load Workspace* button and add the workspace-file called *Assignment3.json*. All the workspaces for this training can be found in `DataVisualizer`. Two axes appear in the graph. These can be configured on the panel on the right-hand side, as illustrated in Figure 5-3.

**Figure 5-3. Assignment 3: Data Visualizer Loaded Work Space**



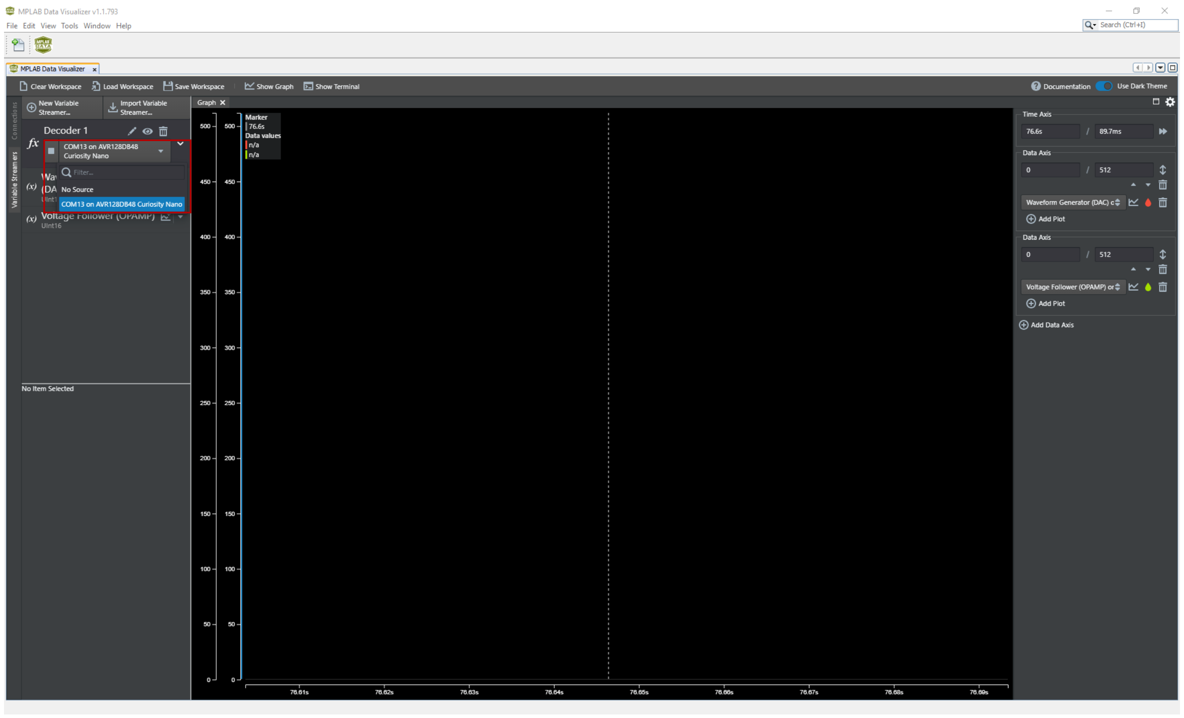
COM13 Settings
Baud Rate: 115200
Char Length: 8 bits
Parity: None
Stop Bits: 1 bit

Time Axis: 76.6s / 89.7ms
Data Axis: 0 / 512 — Waveform Generator (DAC) c
Data Axis: 0 / 512 — Voltage Follower (OPAMP) or

3. Choose the COM-port on the left-hand side panel seen in Figure 5-3 to plot the data. Make sure the *Baud Rate* is *115200* and press the *Apply* button to set the baud rate. Press the *Play* button next to the COM field

shown in Figure 5-3. Press the *Variable Streamers* button to connect the decoder to the USART data stream. Set the COM port as input to Decoder 1, as shown in Figure 5-4.

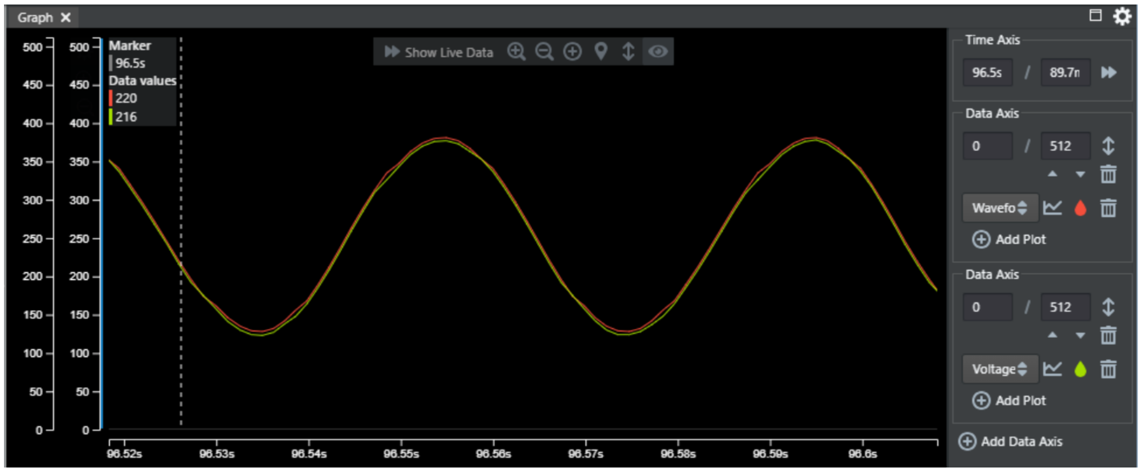
**Figure 5-4. Assignment 3: Variable Streamers**



4. Press *Show Live Data* to start plotting live data from the device.

**Result:** The data streamed to the PC over USART are plotted in the MPLAB® *Data Visualizer*, as shown in Figure 5-5. The red waveform is the output from the DAC, while the green is the output from the op amp. As expected, we can see that the output of the op amp closely follows the input to the op amp.

**Figure 5-5. Assignment 3: Result**



**Info:** The DAC outputs a 50 Hz sinusoidal wave with 256 $mV_{pp}$ and 256 mV DC offset. The sinusoidal wave is pre-calculated at startup in the sine_wave_table_init() function, and DAC updated with values upon TCB0 ISR (SINE_WAVE_TIMER_vect).

```
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);
    }
}
```

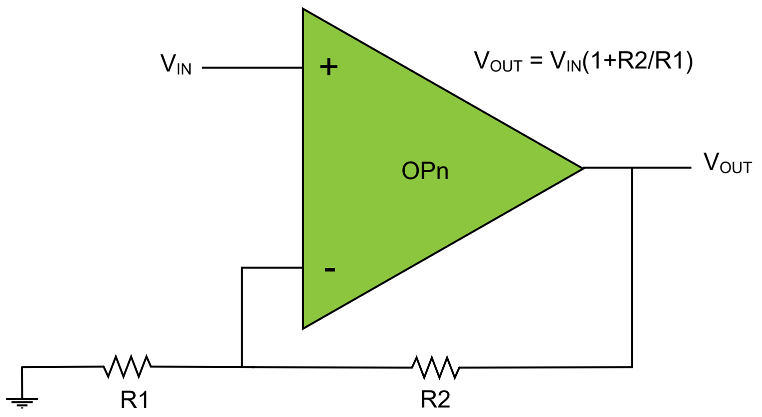
```
ISR(SINE_WAVE_TIMER_vect) {
    volatile static uint16_t sine_wave_index = 0

    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. Assignment 4: OPAMP - Non-Inverting PGA

In this assignment, the AVR128DB48 Curiosity Nano will be configured to use the op amp OP0 in the Analog Signal Conditioning (OPAMP) peripheral as a non-inverting programmable gain amplifier, shown in the block diagram below.



The configuration is done through Atmel START.

The Digital-to-Analog Converter (DAC) is configured to create a sinusoidal signal, which will be used as an input to the op amp.

The Analog-to-Digital Converter (ADC) is configured to sample the output of the op amp. Both the input and output of the op amp is streamed over to the MPLAB® Data Visualizer using the data streamer protocol.

The circuit will also be simulated using MPLAB® Mindi™ Analog Simulator and a pre-made schematic containing a full model of the AVR DB op amp. The schematic is available at github.com/microchip-pic-avr-examples/avrdb-opamp-mindi-non-inverting-pga.

**Info:** The schematic file (*.wxsch), uses the encrypted model (*.lb) so as to not count towards the node simulation limit in Mindi™. If desired an unencrypted version of the model (*.txt) can be found in the same location, for use with a full version of simulation tools.

The starting point for this assignment is the Atmel Studio Solution *Assignment4* found in *Assignment4*.

- **Objectives**
  – Simulate the non-inverting PGA using the pre-made schematic containing a model of the AVR DB op amp
  – Understand how to configure an op amp in Atmel START
  – Configure OP0 instance of the OPAMP, as non-inverting PGA using Atmel START
  – Plot the input and output from OP0 using the MPLAB® Data Visualizer

## 6.1 Simulating Non-Inverting PGA in Mindi™

**To do:**
- Download the pre-made non-inverting PGA simulation file containing the AVR DB op amp model
- Run the simulation

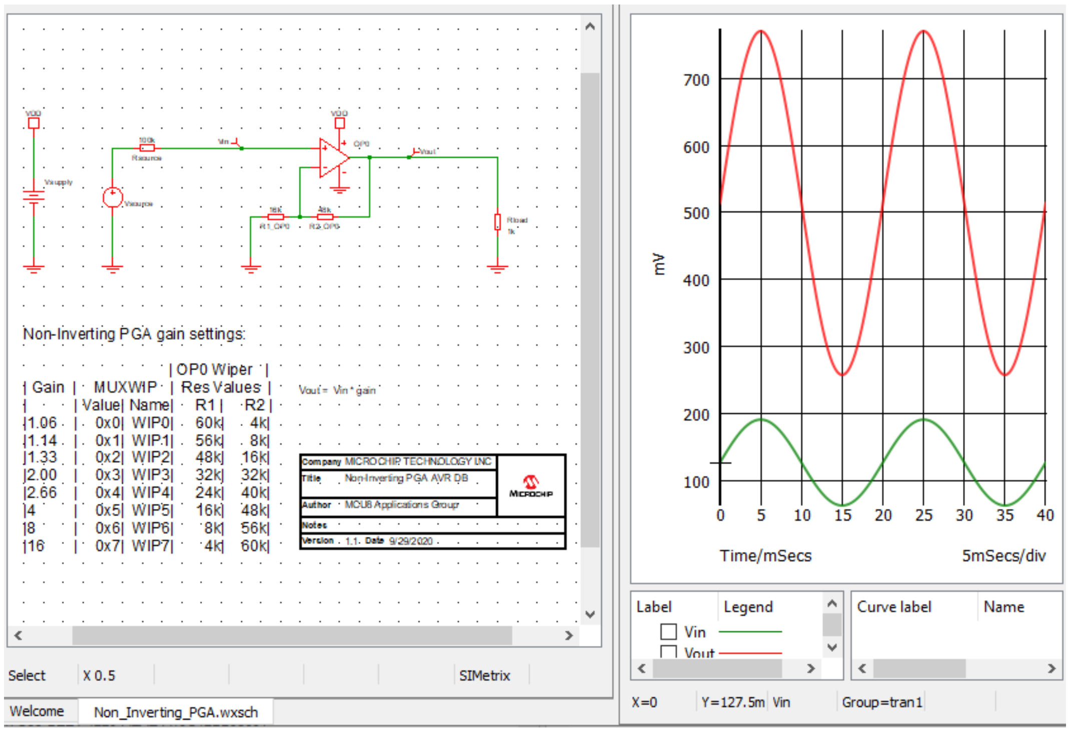
1. Go to github.com/microchip-pic-avr-examples/avrdb-opamp-mindi-non-inverting-pga/releases and press the Source code (zip) link to download the simulation.

**Info:** All the pre-made simulations can be found at github.com/search?l=&q=org%3Amicrochip-pic-avr-examples+avrdb+in%3Aname+extension%3Awxsch&type=repositories.

2. Unzip the folder to your location of choice.
3. Open MPLAB® Mindi™, press *File→Open* or *Ctrl+o*, and locate the file *Non_Inverting_PGA.wxsch* in the location where the zip file was extracted.
4. Simulate by pressing the *Run Schematic* button.

**Result:** If everything worked correctly, the plot of the simulation, as shown in Figure 6-1, should come up.

**Figure 6-1. Assignment 4: MPLAB® Mindi™ Plot**



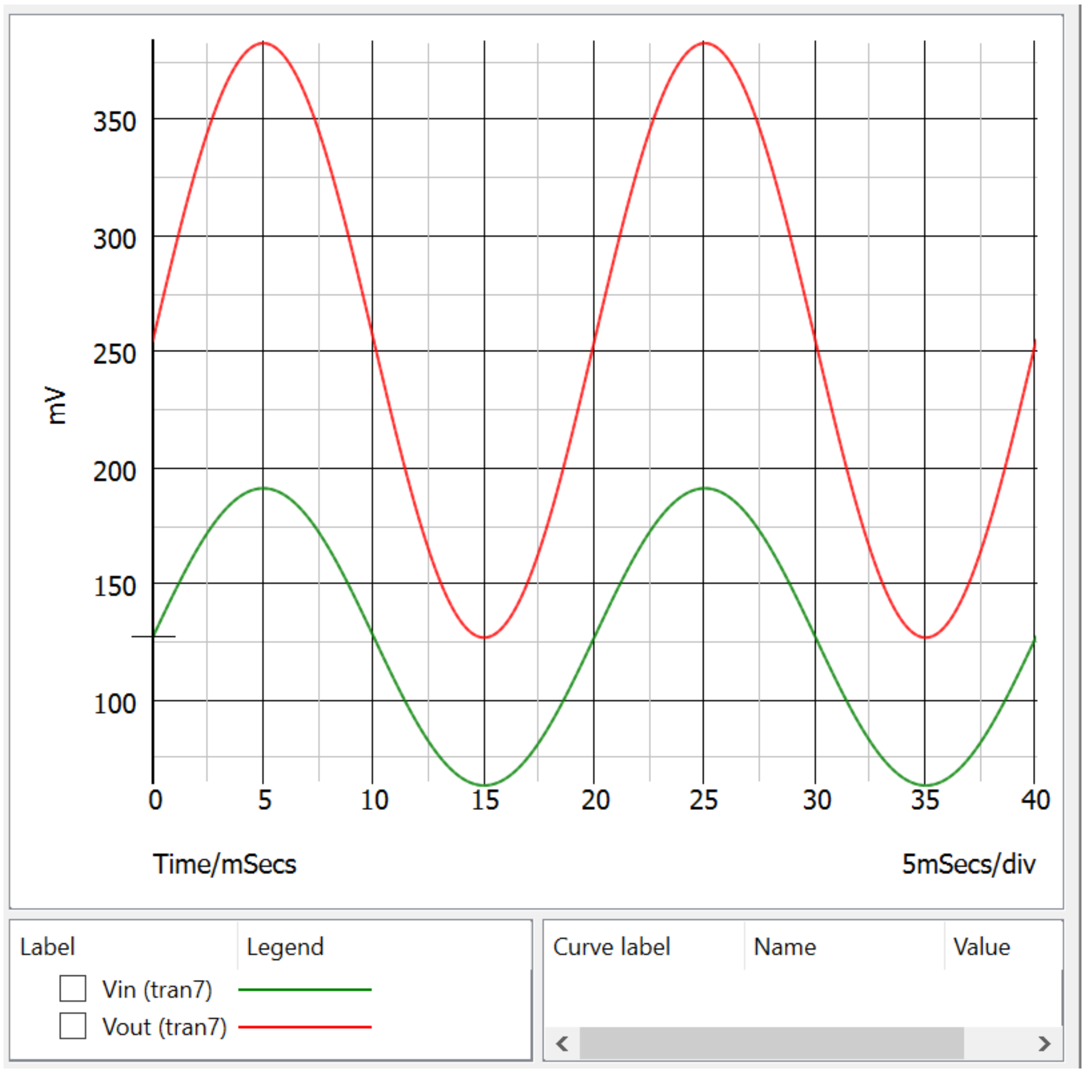
Non-Inverting PGA gain settings:

| Gain | MUXWIP Value | MUXWIP Name | OP0 Wiper Res Values R1 | OP0 Wiper Res Values R2 |
|---|---|---|---|---|
| 1.06 | 0x0 | WIP0 | 60k | 4k |
| 1.14 | 0x1 | WIP1 | 56k | 8k |
| 1.33 | 0x2 | WIP2 | 48k | 16k |
| 2.00 | 0x3 | WIP3 | 32k | 32k |
| 2.66 | 0x4 | WIP4 | 24k | 40k |
| 4 | 0x5 | WIP5 | 16k | 48k |
| 8 | 0x6 | WIP6 | 8k | 56k |
| 16 | 0x7 | WIP7 | 4k | 60k |

5. Change the resistors R1_OP0 and R2_OP0 to the once corresponding to WIP3. This will set the gain to 2x.
6. Simulate with the new resistor values by pressing *Run Schematic*.

**Result:** The plot in MPLAB® Mindi™ should look similar to the one in Figure 6-2, where the red waveform is the output from the op amp, and the green is the input. As expected, the output is twice as large as the input.

**Figure 6-2. Assignment 4: Mindi™ Result**



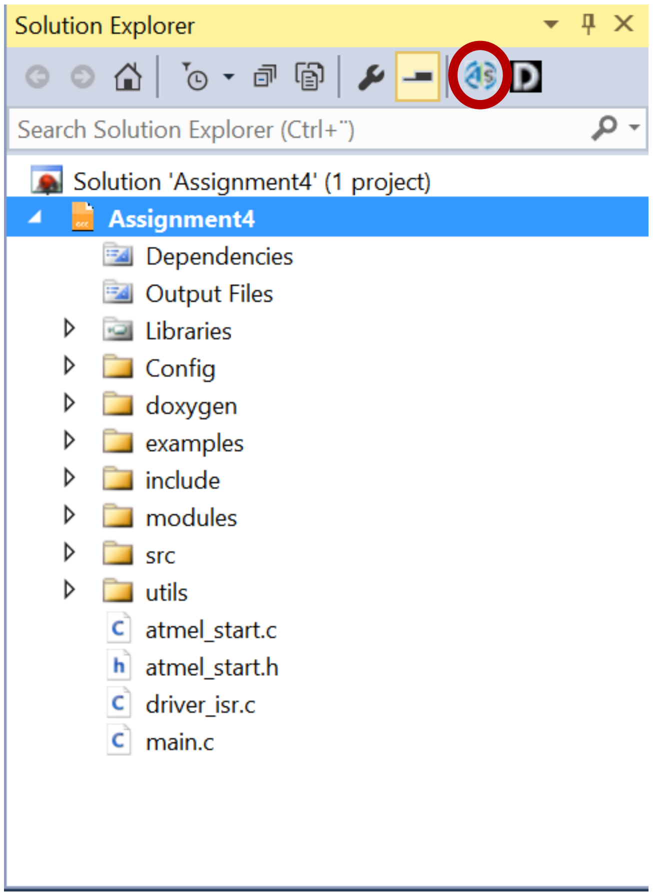
## 6.2 Configure OP0 as a Non-Inverting PGA Using Atmel START

**To do:**

- Edit the Atmel START project, so OP0 is configured as a non-inverting PGA
- Plot the input and output of OP0 in the MPLAB® Data Visualizer

1. Press the *Reconfigure Atmel Start Project* button marked with a red circle in Figure 6-3 to reconfigure the project.

**Figure 6-3. Assignment 4: Reconfigure Atmel START**

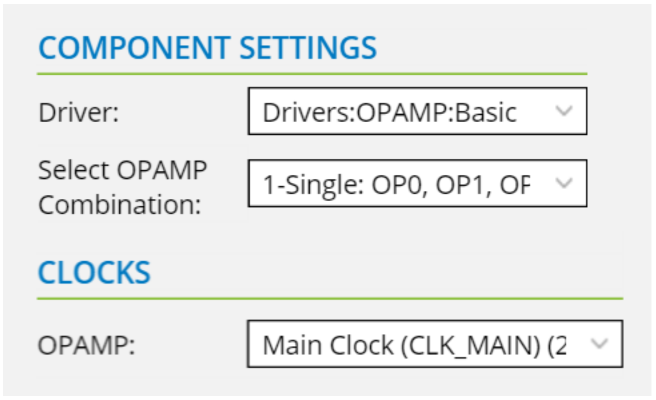

Solution Explorer

Search Solution Explorer (Ctrl+¨)

- Solution 'Assignment4' (1 project)
  - Assignment4
    - Dependencies
    - Output Files
    - Libraries
    - Config
    - doxygen
    - examples
    - include
    - modules
    - src
    - utils
    - atmel_start.c
    - atmel_start.h
    - driver_isr.c
    - main.c

2. Press the *Add software component* button, search for op amp, press the plus button and the *Add component(s)* to add the op amp driver to the project.
3. Press the *OPERATIONAL_AMPLIFIER_0* to be able to configure the OPAMP peripheral.
4. Make sure the COMPONENT SETTINGS and CLOCK is as shown in Figure 6-4.

**Figure 6-4. Assignment 4: Component Settings**


COMPONENT SETTINGS

Driver: Drivers:OPAMP:Basic

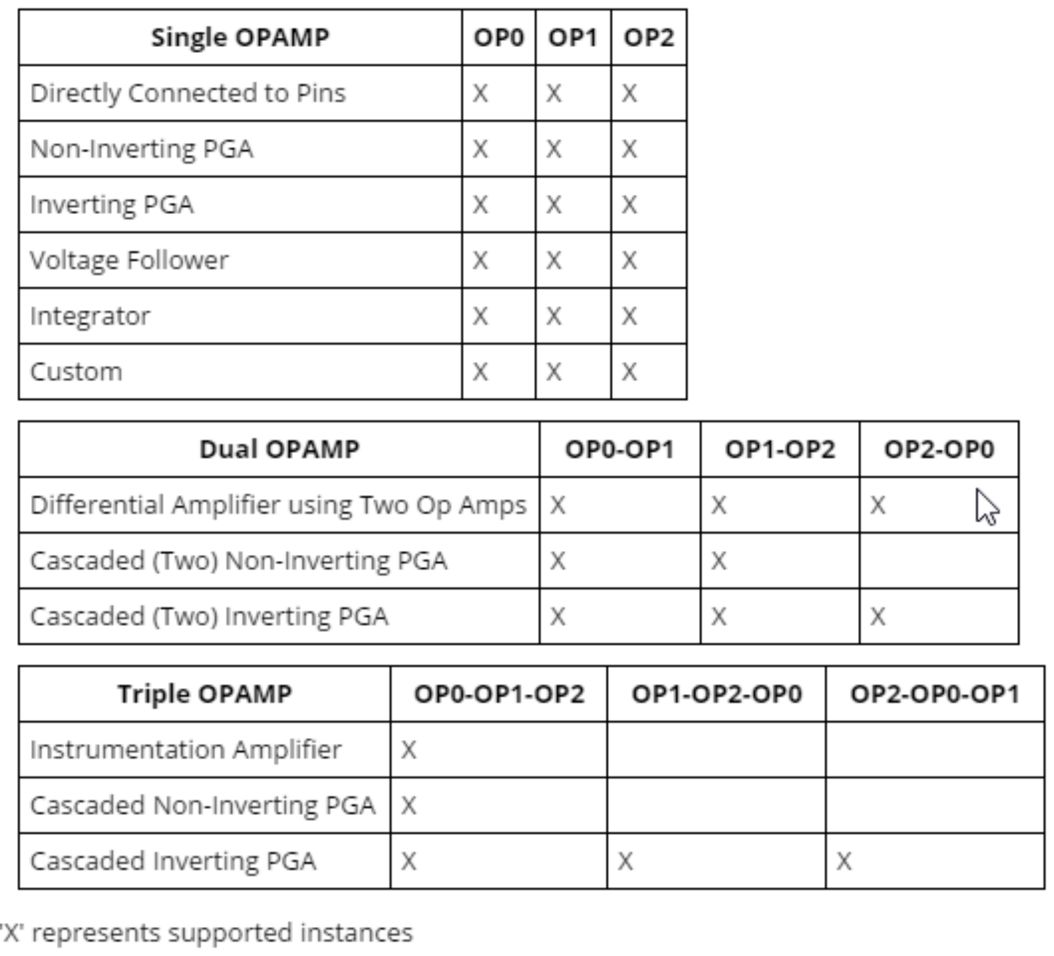
Select OPAMP Combination: 1-Single: OP0, OP1, OF

CLOCKS

OPAMP: Main Clock (CLK_MAIN) (2

**Info:** The Select OPAMP combination determines if the op amps are configured individually or combined to create multi op amp configurations. Not every configuration is available for every combination of op amps. See the START driver user guide for an overview. It is also provided below.

**Figure 6-5. Assignment 4: OPAMP User Guid**



| Single OPAMP | OP0 | OP1 | OP2 |
|---|---|---|---|
| Directly Connected to Pins | X | X | X |
| Non-Inverting PGA | X | X | X |
| Inverting PGA | X | X | X |
| Voltage Follower | X | X | X |
| Integrator | X | X | X |
| Custom | X | X | X |

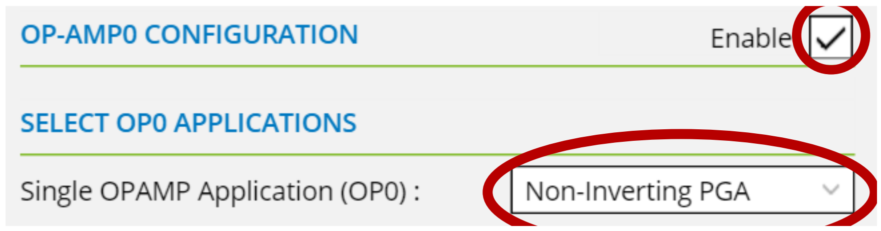
| Dual OPAMP | OP0-OP1 | OP1-OP2 | OP2-OP0 |
|---|---|---|---|
| Differential Amplifier using Two Op Amps | X | X | X |
| Cascaded (Two) Non-Inverting PGA | X | X | |
| Cascaded (Two) Inverting PGA | X | X | X |

| Triple OPAMP | OP0-OP1-OP2 | OP1-OP2-OP0 | OP2-OP0-OP1 |
|---|---|---|---|
| Instrumentation Amplifier | X | | |
| Cascaded Non-Inverting PGA | X | | |
| Cascaded Inverting PGA | X | X | X |

'X' represents supported instances

5. Make sure that OP0 is enabled by checking the OP-AMP0 CONFIGURATION Enable. Checkmark and configure the op amp as a non-inverting PGA by selecting Non-Inverting PGA in the Single OPAMP Application (OP0) drop-down menu as shown in Figure 6-6.

**Figure 6-6. Assignment 4: Single OPAMP Applications**



OP-AMP0 CONFIGURATION

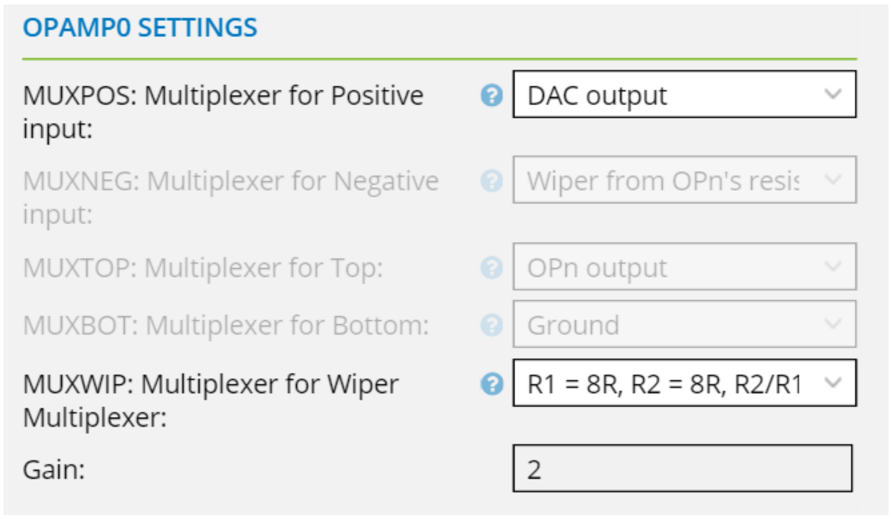
Enable ✓

SELECT OP0 APPLICATIONS

Single OPAMP Application (OP0) : Non-Inverting PGA

6. Set the OPAMP0 SETTINGS, as shown in Figure 6-7. This will set the DAC as an input to the op amp, and the non-inverting PGA will have a gain of 2.

**Figure 6-7. Assignment 4: OPAMP0 SETTINGS**



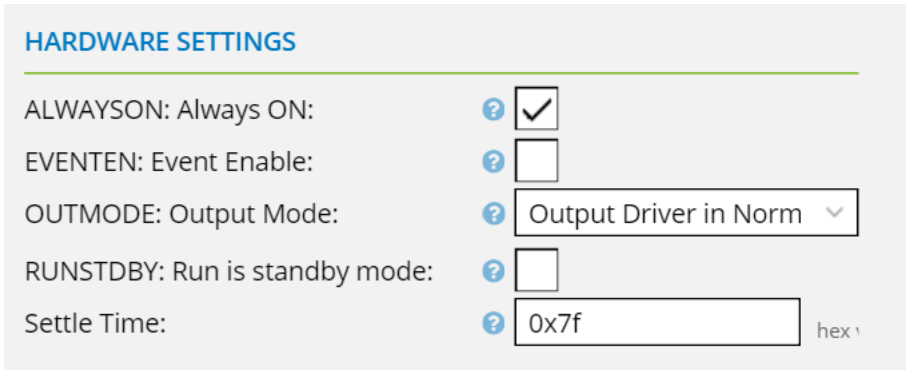
OPAMP0 SETTINGS

| | |
|---|---|
| MUXPOS: Multiplexer for Positive input: | DAC output |
| MUXNEG: Multiplexer for Negative input: | Wiper from OPn's resis |
| MUXTOP: Multiplexer for Top: | OPn output |
| MUXBOT: Multiplexer for Bottom: | Ground |
| MUXWIP: Multiplexer for Wiper Multiplexer: | R1 = 8R, R2 = 8R, R2/R1 |
| Gain: | 2 |

**Info:** The grayed out options are options that cannot change as that will break the chosen configurations, but they are still visible to inform the user what kind of settings are needed for the configuration.

7. Set the HARDWARE SETTINGS to what is shown in Figure 6-8. It should be the default values.

**Figure 6-8. Assignment 4: Hardware Settings**



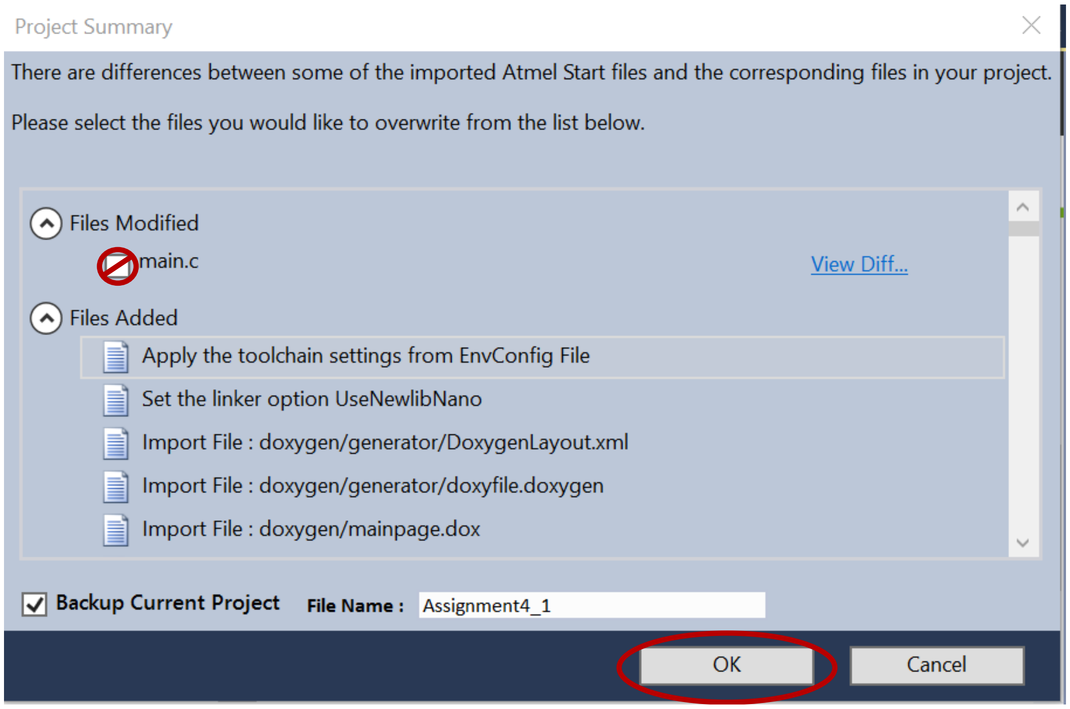
HARDWARE SETTINGS

| | |
|---|---|
| ALWAYSON: Always ON: | ✓ |
| EVENTEN: Event Enable: | ☐ |
| OUTMODE: Output Mode: | Output Driver in Norm |
| RUNSTDBY: Run is standby mode: | ☐ |
| Settle Time: | 0x7f hex |

8. Press *GENERATE PROJECT* to regenerate the start project with the newly added op amp settings.

**⚠WARNING** There will pop up a window called Project Summary shown in Figure 6-9. Press okay and do not check the checkmark next to main.c. If this is done, the project custom changes in main.c will be overwritten and lost, and the project will not work.

**Figure 6-9. Assignment 4: Project Summary**



Project Summary

There are differences between some of the imported Atmel Start files and the corresponding files in your project.

Please select the files you would like to overwrite from the list below.

Files Modified
- main.c — View Diff...

Files Added
- Apply the toolchain settings from EnvConfig File
- Set the linker option UseNewlibNano
- Import File : doxygen/generator/DoxygenLayout.xml
- Import File : doxygen/generator/doxyfile.doxygen
- Import File : doxygen/mainpage.dox

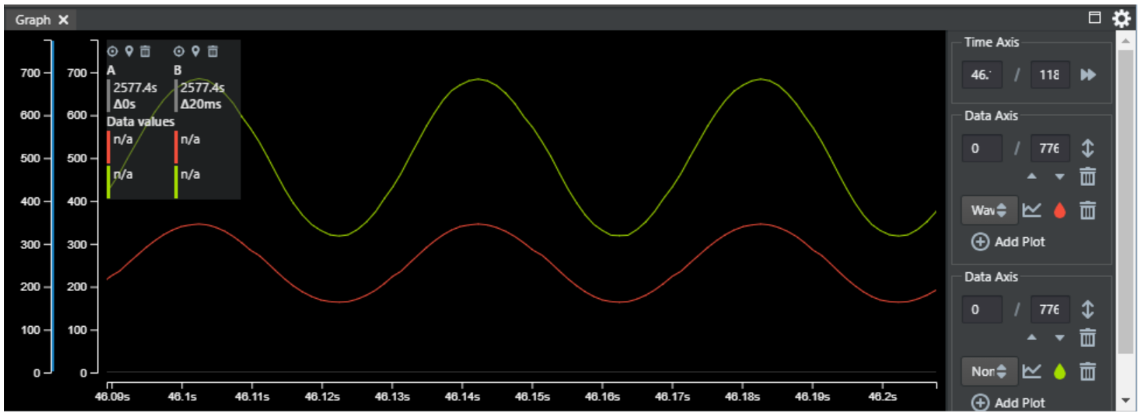
[✓] Backup Current Project File Name : Assignment4_1

9. Verify that the solution/project builds by selecting the *Build* → *Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
10. Flash the device by selecting the *Debug* → *Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.
11. Plot the DAC output vs. the op amp output using the MPLAB® *Data Visualizer* by loading the workspace *Assignment4.json*.

**Info:** The DAC outputs a 50 Hz sinusoidal wave with 128 $mV_{pp}$ and a DC offset of 128 mV.

**Result:** The plot in the MPLAB® *Data Visualizer* should look similar to the one in Figure 6-10. The red waveform is the output from the DAC, while the green is the output from the op amp. As expected, the output from the op amp is two times as large as the output from the DAC. This also matches what we saw in the simulations earlier in the assignment.

**Figure 6-10. Assignment 4: Result**



# 7. Assignment 5: MVIO - OPAMP as a Regulated Power Supply for VDDIO2

In this assignment, the AVR128DB48 Curiosity Nano will be configured to use two voltage domains $V_{DD}$ and $V_{DDIO2}$. $V_{DD}$ is supplied by the CNANO voltage regulator, while $V_{DDIO2}$ is supplied by OP0, which is configured as a voltage follower with the DAC as an input. The reason the DAC is not used directly to supplying $V_{DDIO2}$ is that the DAC can source very little current compared to the op amp.

The Digital-to-Analog Converter (DAC) is configured to create the input voltage to OP0. Therefore, the output from the DAC will be the voltage level for $V_{DDIO2}$.

The Analog-to-Digital Converter (ADC) is configured to sample $V_{DDIO2}$ divided by 10. The measured $V_{DDIO2}$ is streamed over to the MPLAB® Data Visualizer using the data streamer protocol.

The starting point for this assignment is the Atmel Studio Solution *Assignment5* found in *Assignment5*.

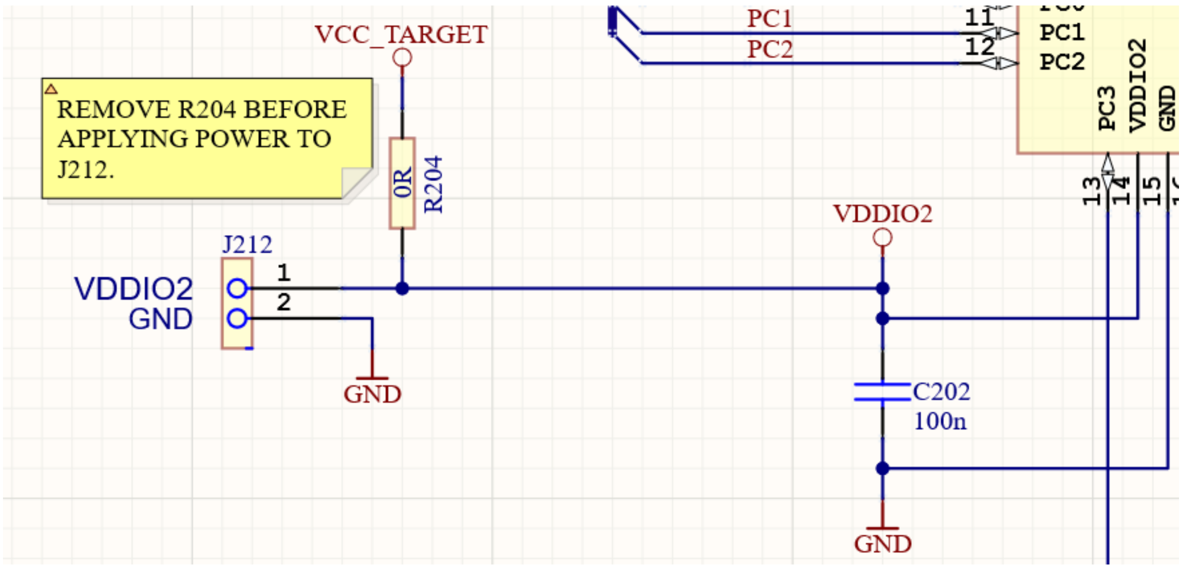
- **Objectives**
  – Understand how the AVR DB can work with two different voltage domains
  – Use the ADC to measure $V_{DDIO2}$ using the internal connection
  – Plot the measured $V_{DDIO2}$ using the MPLAB® Data Visualizer

## 7.1 MVIO Hardware Setup

**To do:**
- Modify the AVR128DB48 Curiosity Nano to work with two voltage domains
- Set the AVR128DB48 Curiosity Nano fuses so it operates with two voltage domains

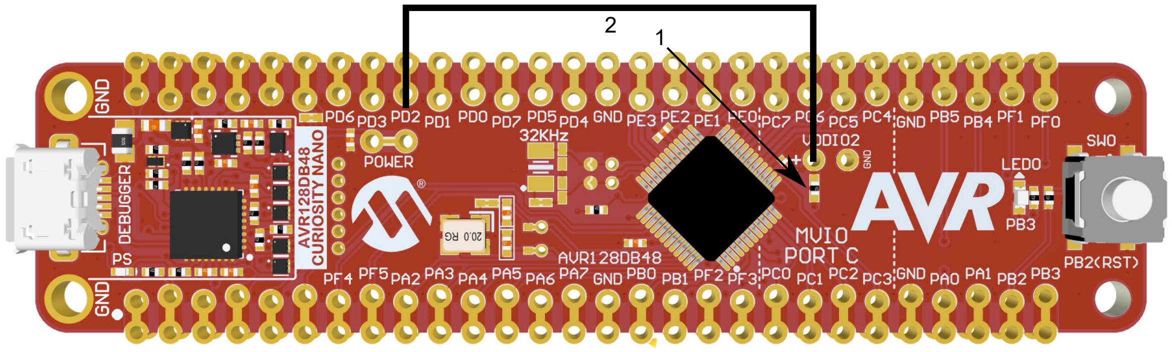
1. Remove the resistor right below the VDDIO2 label to disconnect VDDIO2 from VDD. This is the resistor R204 in Figure 7-1, pointed to by the arrow in Figure 7-2.

**Figure 7-1. Assignment 6: VDDIO2 Schematic**



2. Connect a wire between PD2 and VDDIO2+, as shown in Figure 7-2.

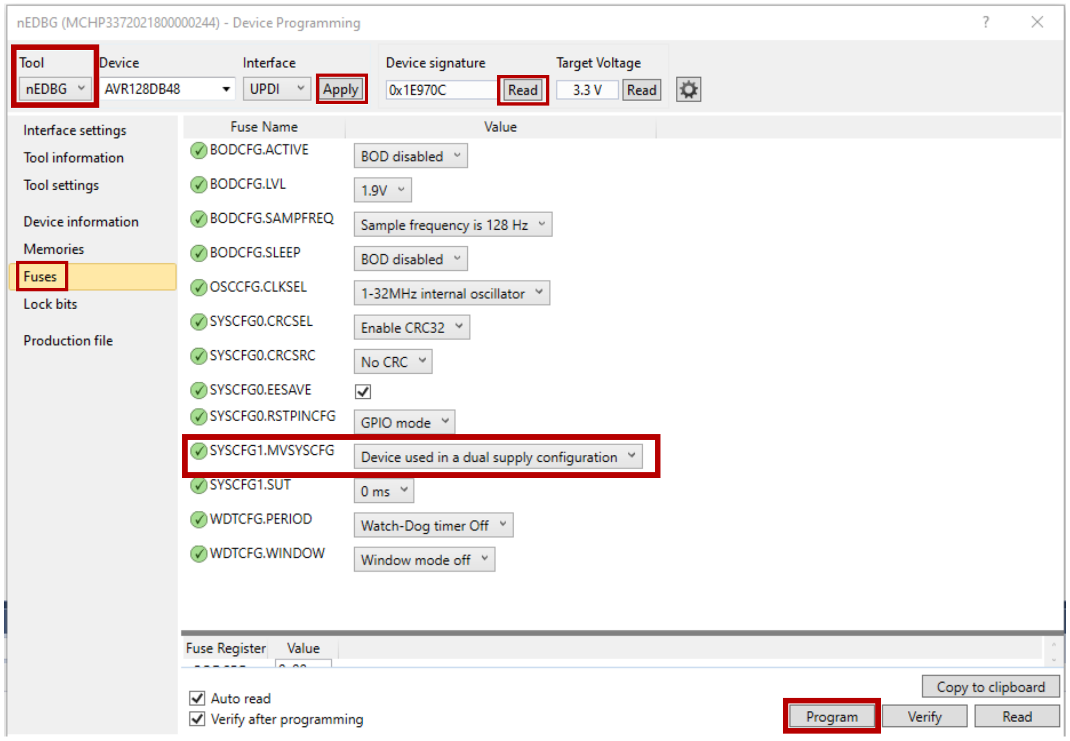
**Figure 7-2. Assignment 5: Curiosity Nano AVR128DB48**


2
1
GND
DEBUGGER
PS
AVR128DB48 CURIOSITY NANO
POWER
32KHz
VDDIO2
MVIO
PORT C
AVR128DB48
LED0
SW0
PB3
PB2(RST)
PD6 PD3 PD2 PD1 PD0 PD7 PD5 PD4 GND PE3 PE2 PE1 PE0 PC7 PC6 PC5 PC4 GND PB5 PB4 PF1 PF0
PF4 PF5 PA2 PA3 PA4 PA5 PA6 PA7 GND PB0 PB1 PF2 PF3 PC0 PC1 PC2 PC3 GND PA0 PA1 PB2 PB3

3. Set the MVSYSCFG bit field in the System Configuration 1 (SYSCFG) fuse to DUAL.

**Info:** To program fuses in Atmel Studio, press *Tools→Device Programming* to get the Device Programming window. Under Tools, select the nEDBG, then press *Apply* followed by *Read*. Press *Fuses* and set SYSCFG1.MVSYSCFG to "Device used in a dual supply configuration." The window should then look like in Figure 7-3. Press *Program* to program the current fuse settings onto the device.

**Figure 7-3. Assignment 5: Atmel Studio Fuse Programing**



| Fuse Name | Value |
|---|---|
| BODCFG.ACTIVE | BOD disabled |
| BODCFG.LVL | 1.9V |
| BODCFG.SAMPFREQ | Sample frequency is 128 Hz |
| BODCFG.SLEEP | BOD disabled |
| OSCCFG.CLKSEL | 1-32MHz internal oscillator |
| SYSCFG0.CRCSEL | Enable CRC32 |
| SYSCFG0.CRCSRC | No CRC |
| SYSCFG0.EESAVE | ✓ |
| SYSCFG0.RSTPINCFG | GPIO mode |
| SYSCFG1.MVSYSCFG | Device used in a dual supply configuration |
| SYSCFG1.SUT | 0 ms |
| WDTCFG.PERIOD | Watch-Dog timer Off |
| WDTCFG.WINDOW | Window mode off |

**Result:** The AVR128DB48 Curiosity Nano CNANO is ready to run with two power domains.

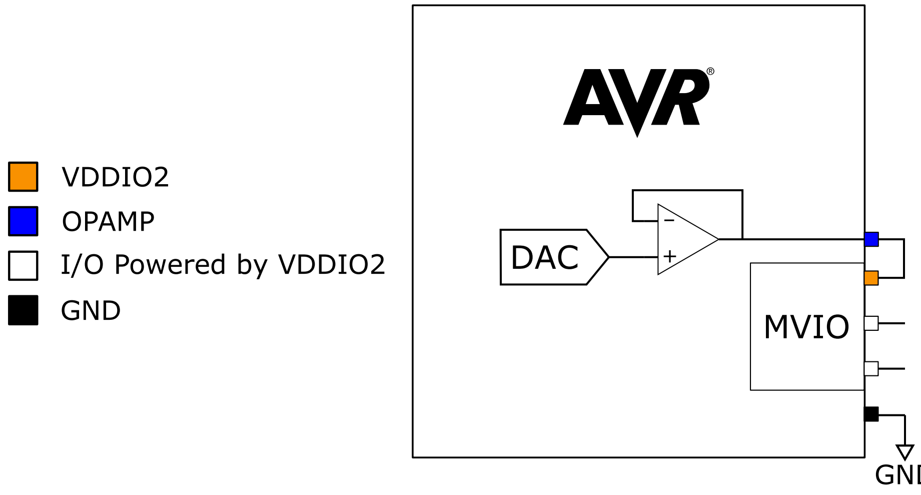
## 7.2 Measuring VDDIO2 Using the ADC

**To do:**
- Understand how the op amp and DAC combine to create a power supply for $V_{DDIO2}$
- Edit `adc0_init()` so it configures the ADC to measure $V_{DDIO2}$
- Plot the measured $V_{DDIO2}$ values in MPLAB® *Data Visualizer*

1. Study the `op_amp_ini()` function to see how OP0 is configured as a voltage follower with the DAC as an input.
2. Study the `dac_init()` function to see how it outputs a constant voltage called `VDDIO2`, which is defined in `dac.h`.

**Info:** Combining the DAC and op amp with the hardware modifications used earlier in this assignment, the op amp will be a power supply for $V_{DDIO2}$. This is illustrated in Figure 7-4.

**Figure 7-4. Assignment 5: Op Amp as a Power Supply**



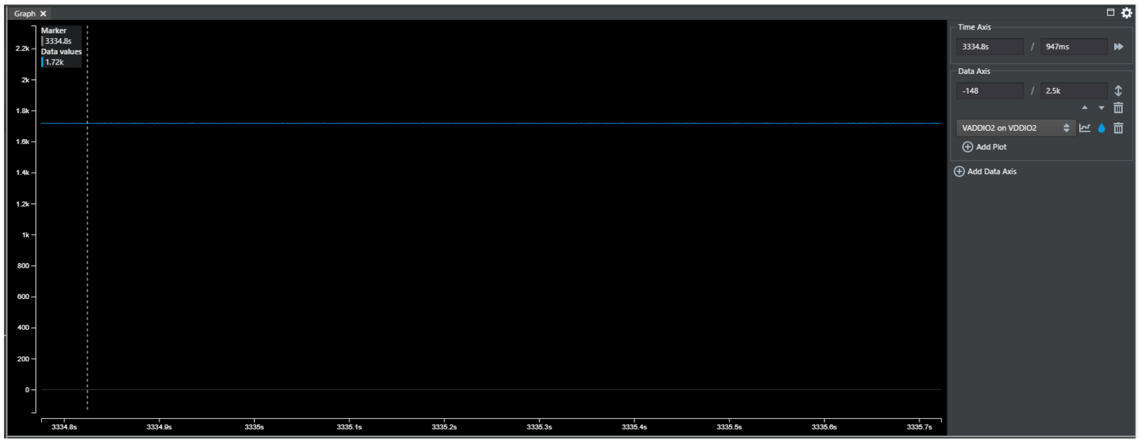
3. In the `adc0_init()` function, set VDDIO2DIV10 as an input to the ADC by writing:

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

4. Verify that the solution/project builds by selecting the *Build* → *Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
5. Flash the device by selecting the *Debug* → *Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.
6. Plot the measured ADC reading in the MPLAB® *Data Visualizer* by loading the workspace *Assignment6.json*.

**Result:** The ADC measurements are plotted in MPLAB® *Data Visualizer*, as shown in Figure 7-5. Note that the unit is millivolts.

**Figure 7-5. Assignment 5: Result**



Time Axis: 3334.8s / 947ms
Data Axis: -148 / 2.5k
VADDIO2 on VDDIO2

# 8. Assignment 6: MVIO - VDDIO2 Failure Detection

In this assignment, the AVR128DB48 Curiosity Nano will be configured to use two voltage domains $V_{DD}$ and $V_{DDIO2}$. $V_{DD}$ is supplied by the CNANO voltage regulator, while $V_{DDIO2}$ is supplied by OP0, which is configured as a voltage follower with the DAC as an input.

The Digital-to-Analog Converter (DAC) is configured to create the input voltage to OP0. The output from the DAC will therefore be the voltage level for $V_{DDIO2}$.

The Analog-to-Digital Converter (ADC) is configured to sample $V_{DDIO2}$ divided by 10. The measured $V_{DDIO2}$ is streamed over to the MPLAB® Data Visualizer using the data streamer protocol.

When SW0 is pressed, the output from the DAC will be reduced, resulting in a VDDIO2 interrupt detecting the failure on the VDDIO2 line.

The starting point for this assignment is the Atmel Studio Solution *Assignment6* found in *Assignment6*.

- **Objectives**
  – Introduce an error to $V_{DDIO2}$ by lowering the supply voltage bellow it's specifications
  – Understand how the AVR DB can detect that $V_{DDIO2}$ falls bellow its minimum requirement
  – Use the ADC to measure $V_{DDIO2}$ using the internal connection
  – Plot the measured $V_{DDIO2}$ using the MPLAB® Data Visualizer

## 8.1 Set Up VDDIO2 Failure Detection

**To do:**
- Edit `mvio_init()` so it enables the VDDIO2 status change interrupt
- Edit `ISR(MVIO_MVIO_vect)` to handle the VDDIO2 status change interrupt
- Plot the measured $V_{DDIO2}$ values in MPLAB® *Data Visualizer*

This assignment assumes the hardware setup steps described in Assignment 6 have been completed. If they have not been completed, the step-by-step instructions can be found in 7.1 MVIO Hardware Setup.

**Info:** The DAC, ADC and OPAMP peripheral are all configured in the same way as it was done for assignment 6. The description of the setup can be found in 7.2 Measuring VDDIO2 Using the ADC.

1. Enable the VDDIO2 status change interrupt by editing `mvio_init()` to include:
```
  MVIO.INTCTRL = MVIO_VDDIO2IE_bm;
```
2. Handle the interrupt by clearing the interrupt flag and toggling LED0 by adding the following to `ISR(MVIO_MVIO_vect)`:
```
MVIO.INTFLAGS = MVIO_VDDIO2IF_bm;
LED0_toggle();
```
3. Understand how the $V_{DDIO2}$ fault is injected when SW0 is pressed by lowering the op amp output by looking at the `ISR(PORTB_PORT_vect)` found in `gpio.c`.

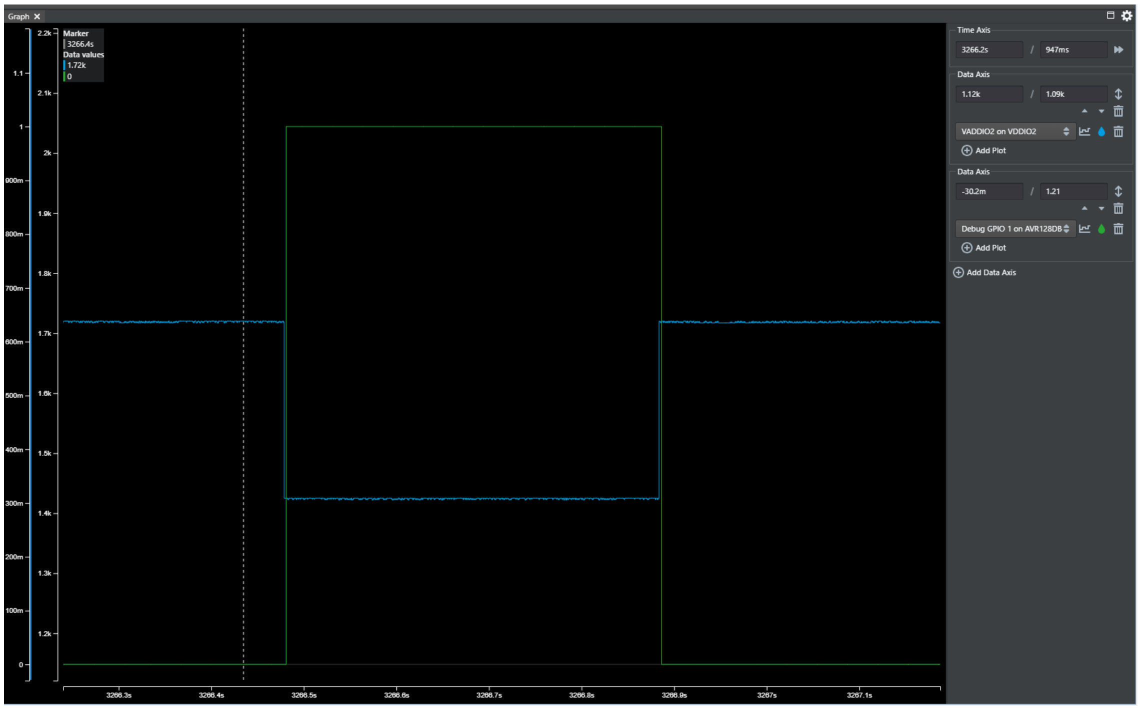
**Info:** When $V_{DDIO2}$ goes below the acceptable range, the VDDIO2 Status bit goes low, triggering the VDDIO2 status change interrupt. When $V_{DDIO2}$ is below the acceptable range, all the pins on PORTC are tri-stated. When $V_{DDIO2}$ goes back up again, the values from the PORTC registers are loaded again.

4. Verify that the solution/project builds by selecting the *Build* → *Build Solution* from the top menu bare in Atmel Studio or by pressing the *F7* key.
5. Flash the device by selecting the *Debug* → *Start without debugging* from the top menu bar in Atmel Studio or by pressing the *Ctrl+Alt+F5* keys.
6. Plot the measured ADC reading and the status of LED0 in MPLAB® *Data Visualizer* by loading the workspace Assignment7.json.

**Info:** LED0 is active low, so the plot in MPLAB® *Data Visualizer* will show '`0`' when the led is lighting and '`1`' when it is off.

**Result:** The ADC measurements and SW0 states are plotted in MPLAB® *Data Visualizer*, as shown in Figure 8-1. It can be seen that as soon as SW0 is pressed and $V_{DDIO2}$ goes below the acceptable value, the VDDIO2 status change interrupt triggers, and the LED0 line goes high. When SW0 is pressed again and $V_{DDIO2}$ goes back into the acceptable range, the LED0 goes back low. The blue line is the $V_{DDIO2}$ reading, while the green line is LED0.

**Figure 8-1. Assignment 6: Result**



## 9. Revision History

| Doc. Rev. | Date | Comments |
|---|---|---|
| A | 11/2020 | Initial document release |

## The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

## Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

## Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

## Microchip Devices Code Protection Feature

## Legal Notice

## Trademarks

ISBN: 978-1-5224-7066-3

## Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

# Worldwide Sales and Service

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

**Atlanta**
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

**Austin, TX**
Tel: 512-257-3370

**Boston**
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

**Chicago**
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

**Dallas**
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

**Detroit**
Novi, MI
Tel: 248-848-4000

**Houston, TX**
Tel: 281-894-5983

**Indianapolis**
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

**Raleigh, NC**
Tel: 919-844-7510

**New York, NY**
Tel: 631-435-6000

**San Jose, CA**
Tel: 408-735-9110
Tel: 408-436-4270

**Canada - Toronto**
Tel: 905-695-1980
Fax: 905-695-2078

## ASIA/PACIFIC

**Australia - Sydney**
Tel: 61-2-9868-6733

**China - Beijing**
Tel: 86-10-8569-7000

**China - Chengdu**
Tel: 86-28-8665-5511

**China - Chongqing**
Tel: 86-23-8980-9588

**China - Dongguan**
Tel: 86-769-8702-9880

**China - Guangzhou**
Tel: 86-20-8755-8029

**China - Hangzhou**
Tel: 86-571-8792-8115

**China - Hong Kong SAR**
Tel: 852-2943-5100

**China - Nanjing**
Tel: 86-25-8473-2460

**China - Qingdao**
Tel: 86-532-8502-7355

**China - Shanghai**
Tel: 86-21-3326-8000

**China - Shenyang**
Tel: 86-24-2334-2829

**China - Shenzhen**
Tel: 86-755-8864-2200

**China - Suzhou**
Tel: 86-186-6233-1526

**China - Wuhan**
Tel: 86-27-5980-5300

**China - Xian**
Tel: 86-29-8833-7252

**China - Xiamen**
Tel: 86-592-2388138

**China - Zhuhai**
Tel: 86-756-3210040

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-3090-4444

**India - New Delhi**
Tel: 91-11-4160-8631

**India - Pune**
Tel: 91-20-4121-0141

**Japan - Osaka**
Tel: 81-6-6152-7160

**Japan - Tokyo**
Tel: 81-3-6880- 3770

**Korea - Daegu**
Tel: 82-53-744-4301

**Korea - Seoul**
Tel: 82-2-554-7200

**Malaysia - Kuala Lumpur**
Tel: 60-3-7651-7906

**Malaysia - Penang**
Tel: 60-4-227-8870

**Philippines - Manila**
Tel: 63-2-634-9065

**Singapore**
Tel: 65-6334-8870

**Taiwan - Hsin Chu**
Tel: 886-3-577-8366

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830

**Taiwan - Taipei**
Tel: 886-2-2508-8600

**Thailand - Bangkok**
Tel: 66-2-694-1351

**Vietnam - Ho Chi Minh**
Tel: 84-28-5448-2100

## EUROPE

**Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

**Denmark - Copenhagen**
Tel: 45-4485-5910
Fax: 45-4485-2829

**Finland - Espoo**
Tel: 358-9-4520-820

**France - Paris**
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

**Germany - Garching**
Tel: 49-8931-9700

**Germany - Haan**
Tel: 49-2129-3766400

**Germany - Heilbronn**
Tel: 49-7131-72400

**Germany - Karlsruhe**
Tel: 49-721-625370

**Germany - Munich**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Germany - Rosenheim**
Tel: 49-8031-354-560

**Israel - Ra'anana**
Tel: 972-9-744-7705

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Italy - Padova**
Tel: 39-049-7625286

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**Norway - Trondheim**
Tel: 47-72884388

**Poland - Warsaw**
Tel: 48-22-3325737

**Romania - Bucharest**
Tel: 40-21-407-87-50

**Spain - Madrid**
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

**Sweden - Gothenberg**
Tel: 46-31-704-60-40

**Sweden - Stockholm**
Tel: 46-8-5090-4654

**UK - Wokingham**
Tel: 44-118-921-5800
Fax: 44-118-921-5820