SAM L21

AT03976: SAM L21 OPAMP as ADC Gain Amplifier

APPLICATION NOTE

Description

Atmel® SAM L21 includes three on-chip operational amplifiers. The Operational Amplifier Controller (OPAMP) module offers most common inverting and non-inverting programmable gain and hysteresis configurations. Each OPAMP is software configurable and can be used as a standalone general purpose operational amplifier. This application note introduces the configuration and usage of the OPAMP module, different built-in modes, and also presents an application example. In the application example, one of the operational amplifiers is configured as a non-inverting gain amplifier with output internally connected to ADC. With this setup, the OPAMP operates as a gain amplifier stage for ADC sampling.

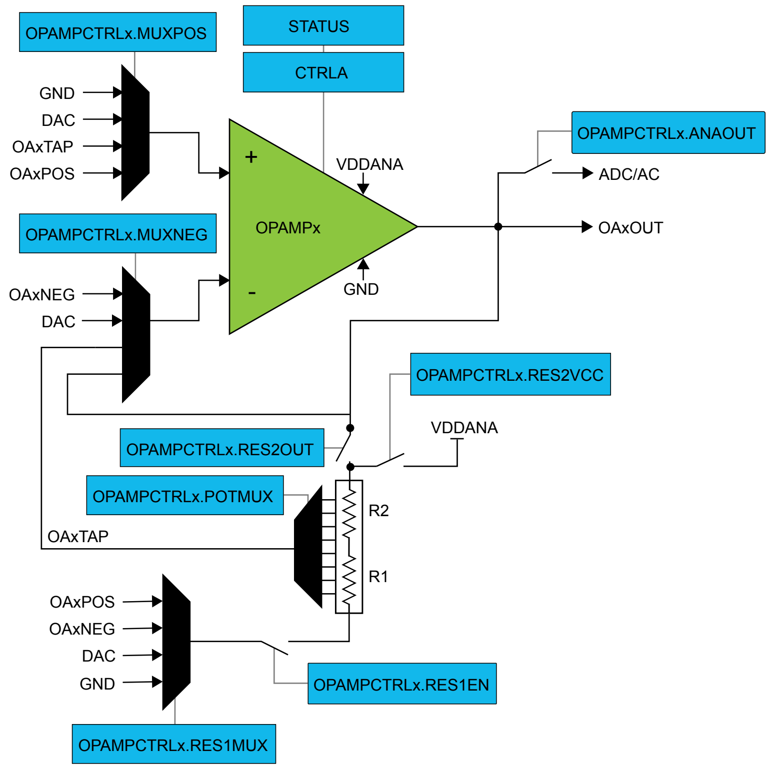


Table of Contents

1. Introduction

This application note introduces the Operational Amplifier Controller (OPAMP) module on SAM L21. The operational amplifiers can be software configured and combined to a number of different modes, as described in the datasheet. Before the OPAMP module is further investigated, a brief introduction to ideal operational amplifiers and the application example is presented.

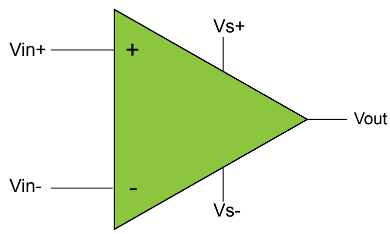
1.1. Operational Amplifiers

An operational amplifier (op-amp) is an electronic component designed for voltage gain amplification. Op-amps most often include a differential input (Vin±), inputs for power supply (Vs±), and a single output (Vout). For an ideal op-amp there are two laws determining its main properties:

1. The op-amp will attempt to keep the differential inputs the same (Vin+ = Vin-).
2. No current flows into or out of the differential inputs. For a non-ideal op-amp the low leakage current is due to the high impedance input stage composed by transistor gates.

Using the two laws, a number of configurations for different purposes can be obtained by creating circuits with op-amps and other components, often resistors. The most common configuations are available as built-in modes on SAM L21 as presented in Built-in Modes on page 5.

Figure 1-1 Ideal op-amp



1.2. Introduction to Application Example

In the code example, one of the OPAMPs is configured as a Non-Inverting Programmable Gain Amplifier (PGA) with output internally connected to the ADC module for signal sampling. The input reference signal is connected to the non-inverting (positive) input while the inverting (negative) input and the output is connected in a closed loop with an internal resistor ladder, as can be seen in Figure 2-3 Non-Inverting PGA on page 5. For a non-inverting PGA, the ideal amplified output is given by the equation: Vout = Vin (1 + R2/R1), which is derived using the two laws of an ideal op-amp.

The example is useful e.g. when sampling a low voltage measurement and there is a need for gain amplification. Recalling the laws for an ideal op-amp, no current flows into the inputs. This property, known as buffering, is valuable for minimizing the impact of the measurement on the actual voltage.

1.2.1. Prerequisites

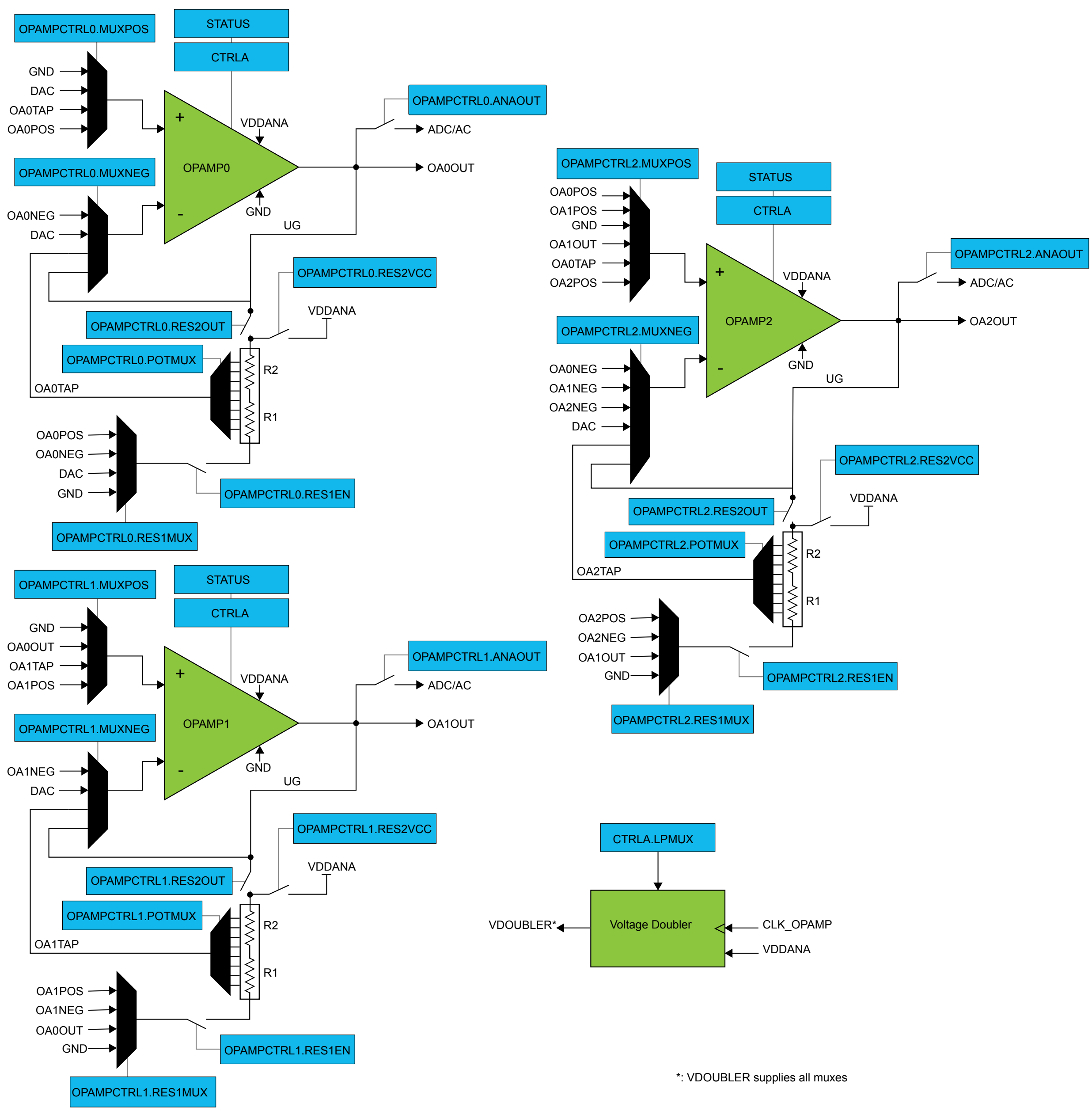
Running the example described in this application note requires:

- Atmel® Studio 6.2 Service Pack 2
- Atmel Software Framework 3.21.0 or higher
- SAM L21 Xplained PRO Evaluation Kit with USB cable

2. Module Overview

A block diagram of the OPAMP module is shown in Figure 2-1 OPAMP Block Diagram on page 4. Each individual operational amplifier is configured by its respective Operational Amplifier Control (OPAMPCTRLx) register.

Figure 2-1 OPAMP Block Diagram



2.1. Configuring the Operational Amplifiers

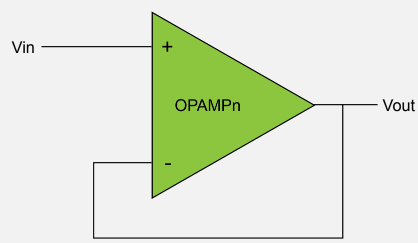
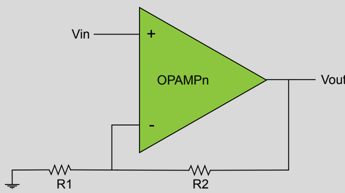
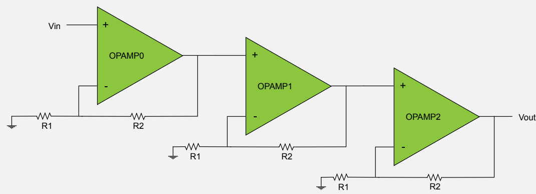
The OPAMP settings must be configured before the amplifier is enabled. Following is a brief description of significant bit fields and groups in the Operational Amplifier Control registers:

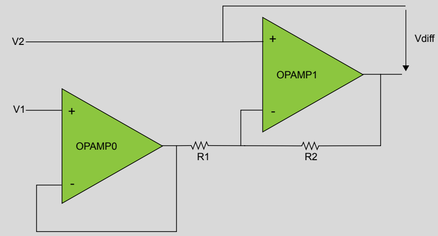
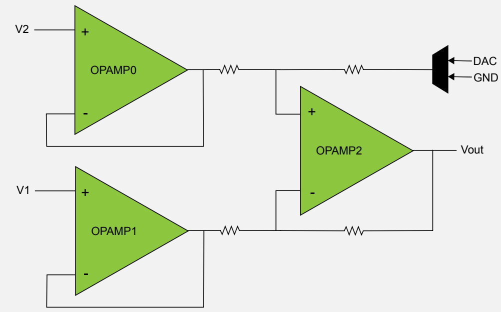
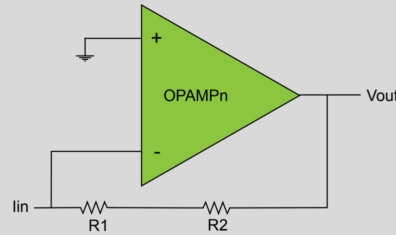
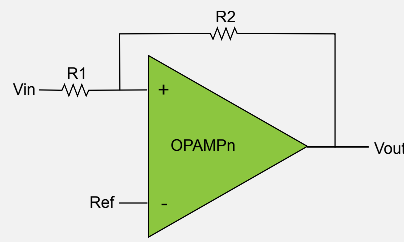
- Select the positive input in OPAMPCTRLx.MUXPOS.
- Select the negative input in OPAMPCTRLx.MUXNEG.
- Select RES1EN if resistor ladder is used.
- Select the input for the resistor ladder in OPAMPCTRLx.RES1MUX.
- Select the potentiometer selection of the resistor ladder in OPAMPCTRLx.POTMUX.
- Select the VCC input for the resistor ladder in OPAMPCTRLx.RES2VCC.
- Connect the operational amplifier output to the resistor ladder using OPAMPCTRLx.RES2OUT.
- Select the trade-off between speed and energy consumption in OPAMPCTRLx.BIAS.

2.2. Built-in Modes

The OPAMP module includes software configurable internal connections, enabling most common op-amp modes. The different modes are briefly introduced in Table 2-1 Built-in Modes on page 5. For more details related to implementation of the built-in modes, refer to the datasheet.

Table 2-1 Built-in Modes

Mode	Figure
Voltage Follower A unity gain amplifier. Vout = Vin	**Figure 2-2 Voltage Follower** 
Inverting/Non-Inverting PGA A software configurable Programmable Gain Amplifier (PGA) using the internal resistor ladder. Vout = Vin (1 + R2/R1)	**Figure 2-3 Non-Inverting PGA** 
Cascaded Inverting/Non-Inverting PGA The OPAMPs can be internally connected in cascade to provide a larger varity of gains. Vout = Vin (1 + R2/R1)(1 + R2/R1)(1 + R2/R1)	**Figure 2-4 Cascaded Non-Inverting PGA** 

Mode	Figure
Differential Amplifier A Voltage Follower and a non-inverting PGA connected in cascade. Vdiff = (V2 - V1) R2/R1	**Figure 2-5 Differential Amplifier** 
Instrumentation Amplifier A Differential Amplifier with input buffer amplifiers. Vout = (V2 - V1) Gain + Ref	**Figure 2-6 Instrumentation Amplifier** 
Transimpedance Amplifier For converting current to voltage. Vout = -Iin (R1 + R2)	**Figure 2-7 Transimpedance Amplifier** 
Programmable Hysteresis Inverting or Non-Inverting comparator with programmable hysteresis. Threshold = ±Vcc R2/R1 + Ref	**Figure 2-8 Non-Inverting Comparator with Programmable Hysteresis** 

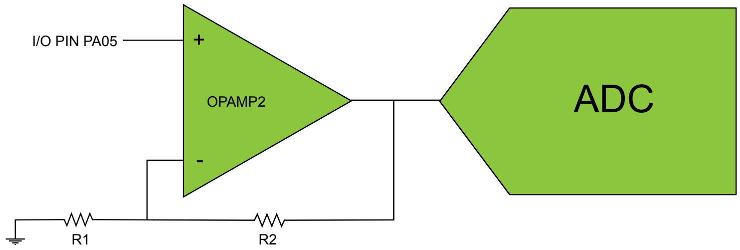
2.3. Low-Power Mode

For low-power applications, the OPAMP module contains settings for power saving. One of them is to tradeoff speed versus power efficiency to get the lowest power consumption. The speed setting is configured for each amplifier individually by the Bias Control field in the Operational Amplifier x Control register (OPAMPCTRLx.BIAS). The BIAS bits select the amount of bias current provided to the operational amplifiers. This will also affect the start-up time. Another low-power configuration is to disable the voltage doubler by setting the Low-Power Mux bit in the Control Register (CTRLA.LPMUX), if the supply voltage is guaranteed to be above 2.5V. Disabling the voltage doubler saves power and reduces the start-up time.

3. Example Overview

In the application example, OPAMP2 is configured as a non-inverting PGA. The amplified output is enabled for sampling with ADC. As can be seen in the datasheet, the OPAMP2 positive input can be multiplexed to I/O pin PA05, hence the input signal to be amplified must be connected to this pin. The pin must also be configured as an input. The ADC needs to be configured in order to use the OPAMP2 output as input.

Figure 3-1 OPAMP2 Configured as Non-inverting PGA with Output Connected to the ADC Module



In this example OPAMP2 is configured with R2/R1 = 1/3 and the internal connection to the ADC is enabled. When OPAMP2 is configured and enabled in the main loop, the ADC starts converting a predefined number of samples of the amplified signal. The ADC is configured in callback mode, implying that an interrupt occurs when the conversion is complete.

3.1. Peripherals

The following peripherals are used by this example:

- Operational Amplifier Controller (OPAMP)
- Analog-to-Digital Converter (ADC)

3.2. Configurations

3.2.1. Configure OPAMP2

1. Add the necessary modules and create a function `configure_opamp2()` in an empty Atmel Studio ASF Board Project for SAM L21 or within an existing SAM L21 project. Configure the OPAMP2 as a Non-Inverting PGA:
   - Inverting input connected to resistor ladder
   - Non-inverting input connected to reference signal
   - R1 enabled and connected to Ground (GND)
   - R2 connected to OPAMP2 output

   In addition, the resistor ladder must be configured to achieve a desired gain. In this example the potentiometer option R1 = 12R and R2 = 4R is selected. Since the output signal from the OPAMP2 will be used by the ADC, it must be enabled by software to be made available. The configuration function should follow the ASF standard by creating a configuration struct, loading default settings, applying necessary settings, and enabling the module.

```
/* Configure OPAMP2 and I/O PORT */
void configure_opamp2(void)
{
    /* Creates a new configuration structure for the OPAMP2. */
```

```
    struct opamp2_config conf;

    /* Initializes OPAMP module. */
    opamp_module_init();

    /* Settings and fill with the default settings. */
    opamp2_get_config_defaults(&conf);

    /* Set the the OPAMP2 in "Non-Inverted PGA" mode. */
    conf.negative_input           = OPAMP2_NEG_MUX_TAP2;
    conf.positive_input           = OPAMP2_POS_MUX_PIN2;
    conf.r1_connection            = OPAMP2_RES1_MUX_GND;
    conf.config_common.potentiometer_selection = OPAMP_POT_MUX_12R_4R;
    conf.config_common.r1_enable  = true;
    conf.config_common.r2_out     = true;
    conf.config_common.analog_out = true;
```

2. Further, the I/O pin must be configured to enable the input signal. As previously mentioned, PA05 corresponds to the positive input for OPAMP2:

```
/* Set up OA2POS pin as input. */
struct system_pinmux_config opamp2_input_pin_conf;
system_pinmux_get_config_defaults(&opamp2_input_pin_conf);
opamp2_input_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
opamp2_input_pin_conf.mux_position = OPAMP_INPUT_MUX;
system_pinmux_pin_set_config(OPAMP_INPUT_PIN, &opamp2_input_pin_conf);
```

Note: `OPAMP_INPUT_MUX` and `OPAMP_INPUT_PIN` is defined in conf_example.h which must be included.

3. Finally, apply the settings and enable OPAMP2:

```
/* Initialize and enable the OPAMP2 with the user settings. */
opamp2_set_config(&conf);
opamp_enable(OPAMP_2);

/* Wait for the output ready. */
while(!opamp_is_ready(OPAMP_2));
```

4. The resulting `configure_opamp2()` function should be like the following:

```
/* Configure OPAMP2 and I/O PORT */
void configure_opamp2(void)
{
    /* Creates a new configuration structure for the OPAMP2. */
    struct opamp2_config conf;

    /* Initializes OPAMP module. */
    opamp_module_init();

    /* Settings and fill with the default settings. */
    opamp2_get_config_defaults(&conf);

    /* Set the the OPAMP2 in "Non-Inverted PGA" mode. */
    conf.negative_input           = OPAMP2_NEG_MUX_TAP2;
    conf.positive_input           = OPAMP2_POS_MUX_PIN2;
    conf.r1_connection            = OPAMP2_RES1_MUX_GND;
    conf.config_common.potentiometer_selection = OPAMP_POT_MUX_12R_4R;
    conf.config_common.r1_enable  = true;
    conf.config_common.r2_out     = true;
    conf.config_common.analog_out = true;

    /* Set up OA2POS pin as input. */
```

```
    struct system_pinmux_config opamp2_input_pin_conf;
    system_pinmux_get_config_defaults(&opamp2_input_pin_conf);
    opamp2_input_pin_conf.direction    = SYSTEM_PINMUX_PIN_DIR_INPUT;
    opamp2_input_pin_conf.mux_position = OPAMP_INPUT_MUX;
    system_pinmux_pin_set_config(OPAMP_INPUT_PIN,
&opamp2_input_pin_conf);

    /* Initialize and enable the OPAMP2 with the user settings. */
    opamp2_set_config(&conf);
    opamp_enable(OPAMP_2);

    /* Wait for the output ready. */
    while(!opamp_is_ready(OPAMP_2));
}
```

3.2.2. Configure ADC

1. In this example, the ADC is configured in callback mode, meaning that an interrupt will be executed at a predefined event after starting the conversion. This is further emphasized when configuring the callback routine in the following section. Declare the following struct globally:

```
struct adc_module adc_instance;
```

2. Create a function `configure_adc()` in your project where you configure the ADC to sample the OPAMP2 output:

```
/* Configure ADC */
void configure_adc(void)
{
    /* Creates a new configuration structure for the ADC */
    struct adc_config config_adc;

    adc_get_config_defaults(&config_adc);

    /* Setup ADC with OPAMP2 output as ADC input */
    config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV8;
    config_adc.positive_input  = ADC_POSITIVE_INPUT_OPAMP2;
```

Note: The ADC clock prescaler must be set accordingly to the specifications of the ADC module and the speed of the clock source. The ADC clock speed should not exceed the maximum conversion frequency.

3. The ADC is initiated and enabled with the following functions:

```
/* Initialize and enable ADC */
adc_init(&adc_instance, ADC, &config_adc);
adc_enable(&adc_instance);
```

4. The resulting `configure_adc()` function shows the minimum number of configurations required for sampling the OPAMP2 output:

```
/* Configure ADC */
void configure_adc(void)
{
    /* Creates a new configuration structure for the ADC */
    struct adc_config config_adc;

    adc_get_config_defaults(&config_adc);

    /* Setup ADC with OPAMP2 output as ADC input */
    config_adc.clock_prescaler = ADC_CLOCK_PRESCALER_DIV8;
    config_adc.positive_input  = ADC_POSITIVE_INPUT_OPAMP2;
```

```
    /* Initialize and enable ADC */
    adc_init(&adc_instance, ADC, &config_adc);
    adc_enable(&adc_instance);
}
```

Note: For the sake of simplicity, the ADC is mostly configured with default settings. The accuracy of the sampling can be increased by applying built-in software selectable features, e.g. accumulate and divide samples for averaging and gain/offset correction.

3.3. Callback Functions

3.3.1. ADC Callback Function

When the ADC is configured in callback mode, a user-defined function is set to run when a desired ADC interrupt occurs. The predefined ADC conditions for excecuting an interrupt are: buffer received, window hit, and conversion error. In this example it is desirable to recieve an interrupt when the ADC conversion is complete, i.e. when the result buffer is recieved.

A basic rule of thumb is to minimize the excecution time of the callback function. In this example, this is ensured by signaling the application with a globally defined conversion complete-flag.

```
/* ADC Callback Function */
void adc_complete_callback(const struct adc_module *const module)
{
    /* Set ADC conversion ended flag */
    adc_read_done = true;
}
```

The callback function is enabled with the following function where the `ADC_CALLBACK_READ_BUFFER` parameter indicates that the callback function is to be associated with a complete conversion.

```
/* Enable ADC Callback Function */
void configure_adc_callbacks(void)
{
    adc_register_callback(&adc_instance,
            adc_complete_callback, ADC_CALLBACK_READ_BUFFER);
    adc_enable_callback(&adc_instance, ADC_CALLBACK_READ_BUFFER);
}
```

3.4. Main Routine

In the beginning of your code, a result buffer for the ADC conversion and the conversion complete-flag should be defined globally.

```
/* Buffer for ADC sample storage */
#define ADC_SAMPLES 128
uint16_t adc_result_buffer[ADC_SAMPLES];

/* ADC interrupt flag */
volatile bool adc_read_done = false;
```

In the main function, the OPAMP and ADC modules are initialized and enabled using the defined configuration functions. After running the configuration functions for the OPAMP and ADC modules, it remaines to enable global interrupts and start the first ADC sampling. The `adc_read_buffer_job()`

function handles the read buffer interrupts and excecutes ADC_SAMPLES number of samples before calling the user-defined callback routine.

```
/* Enable global interrupts */
system_interrupt_enable_global();

/* Start ADC conversion */
adc_read_buffer_job(&adc_instance, adc_result_buffer, ADC_SAMPLES);
```

After the ADC sampling is initiated, the processor is available to perform other tasks. The ADC will interrupt when the conversion is complete. In the following main function, the application enters a while loop and waits for the ADC callback.

```
/* Main function */
int main(void)
{
    system_init();

    /* Initialize OPAMP2 and ADC */
    configure_opamp2();
    configure_adc();
    configure_adc_callbacks();

    /* Enable global interrupts */
    system_interrupt_enable_global();

    /* Start ADC conversion */
    adc_read_buffer_job(&adc_instance, adc_result_buffer, ADC_SAMPLES);

    while (adc_read_done == false) {
        /* Wait for asynchronous ADC read to complete */
    }

    while(true) {
        /* Do nothing */
    }
}
```

Note: The amplified signal voltage can be calculated from the ADC result buffer values according to: `adc_result_buffer * V_REF / ADC_MAX_VALUE`, where `V_REF` is the sampling voltage reference (`ADC_REFERENCE_INTVCC1`) and `ADC_MAX_VALUE` is 0xFFF for a 12-bit conversion.

3.5. Summary

The SAM L21 operational amplifiers can be used for a number of purposes. In this material, we presented how to use the on-chip OPAMP module as non-inverting amplifier as well as to configure both OPAMP and ADC module, such that the output of OPAMP2 is sampled by the ADC.

4. Software License

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

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel® may not be used to endorse or promote products derived from this software without specific prior written permission.
4. This software may only be redistributed and used in connection with an Atmel® microcontroller product.

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

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**