Lab 3: ADC - DAC

Quang Hai Nguyen

Revision 0.99, 07.01.2020

SAMD21 workshop

Revision History

Revision, Date	Editor	Subject (major changes)
Revision 0.99, 07.01.2020	Quang Hai Nguyen	Preliminary

Table of Contents

List of Figures

List of Icon Identifiers

Table 1: Icon Identifiers List

ⓘ	Extra information about a topic
✎	Task needs to be done
⚠	Important information or a warning
✓	Result expected to see

Prerequisite

Hardware

- Laptop or PC with Windows 7 or Windows 10
- A SAMD21-Xplained-Pro evaluation board (provided during the workshop)
- Extension boards, LEDs, Potential meter, buttons and jumper wires (provided during the workshop)
- A micro USB cable (provided during the workshop)
- Breadboard (provided during the workshop)
- **Internet access without proxy or firewall**

Software

- Terminal program (e.g. TeraTerm)
- Atmel Studio 7 (link)

Labs description

Description

This lab will show you how to configure Analog to Digital Converter (ADC) and the Digital to Analog converter (DAC). SAMD21 MCU uses an ADC to read the voltage value from a potential meter and create that voltage level again with a DAC.

Assignment

1. Setup the ADC and the DAC with Atmel Start
2. Writing the application

⚠ Some of the basic configuration steps are skipped in this lab so make sure you have already started lab 1 to learn the basic steps before continuing with this lab.

Lab procedure

Setup ADC and DAC with Atmel Start

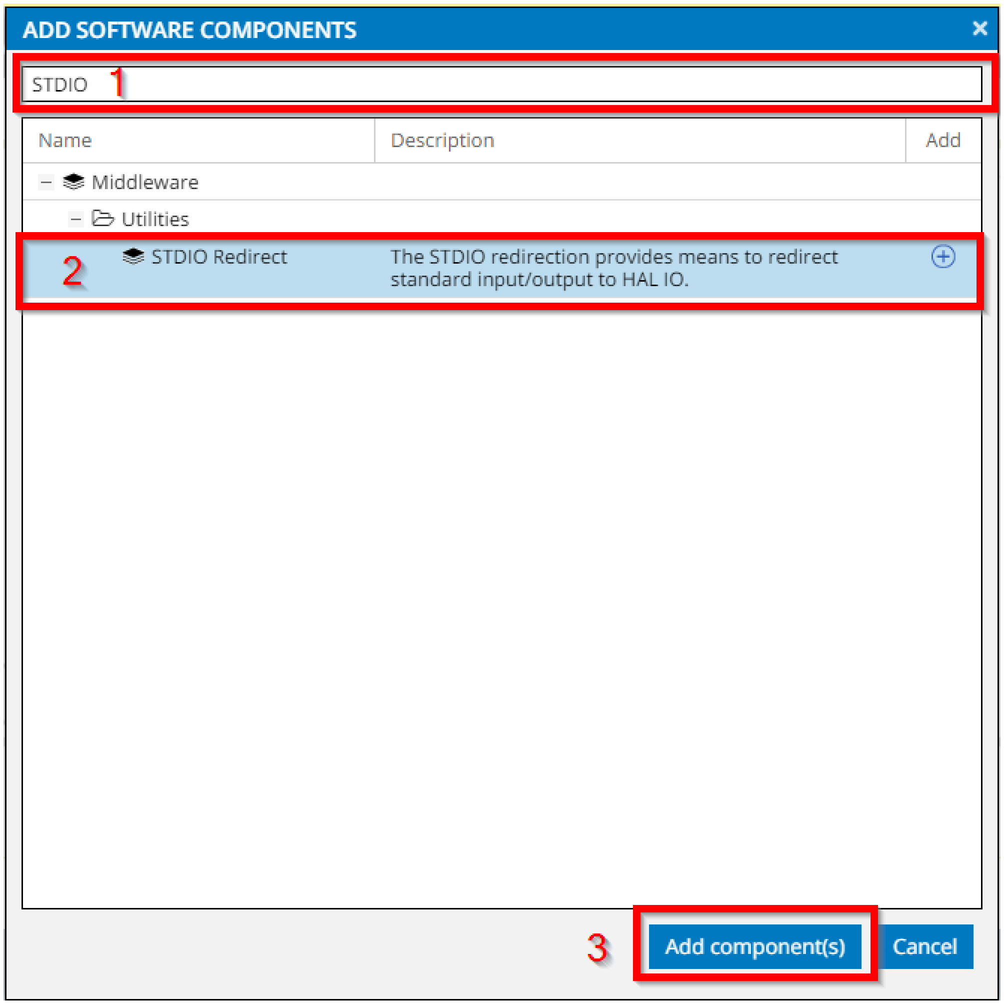
In Atmel Studio, create a new Atmel Start project.

When it is prompted, choose the correct device for your project. In this case, it is SAM D21 Xplained Pro.

Give your project a name. For my case, I name it SAMD21_Lab3.

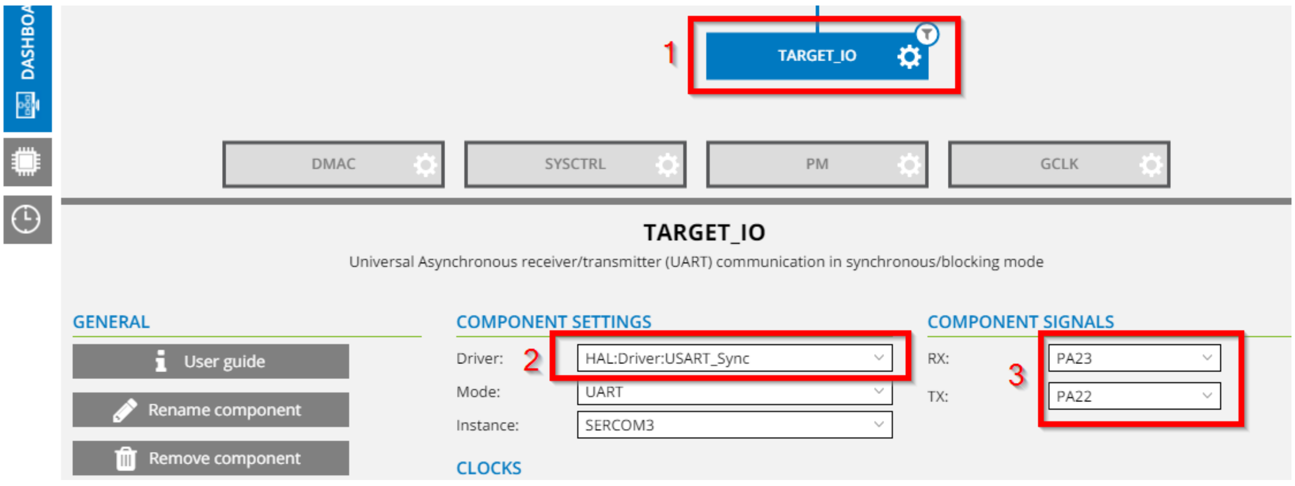
First, we will set up the serial communication so that we can print the debug message on the terminal.

Click “Add software component”. In the prompted window, enter “STDIO” into the filter text box and select “STDIO Redirect”



The STDIO redirection provides means to redirect standard input/output to HAL IO.

Click “TARGET_IO” and set the correct pin for the UART bus. From the “Board user guide”, we know that pin PA22 and PA23 are routed to output pins of USB Virtual Com. Therefore, they are configured as follows:



TARGET_IO

Universal Asynchronous receiver/transmitter (UART) communication in synchronous/blocking mode

GENERAL
- User guide
- Rename component
- Remove component

COMPONENT SETTINGS

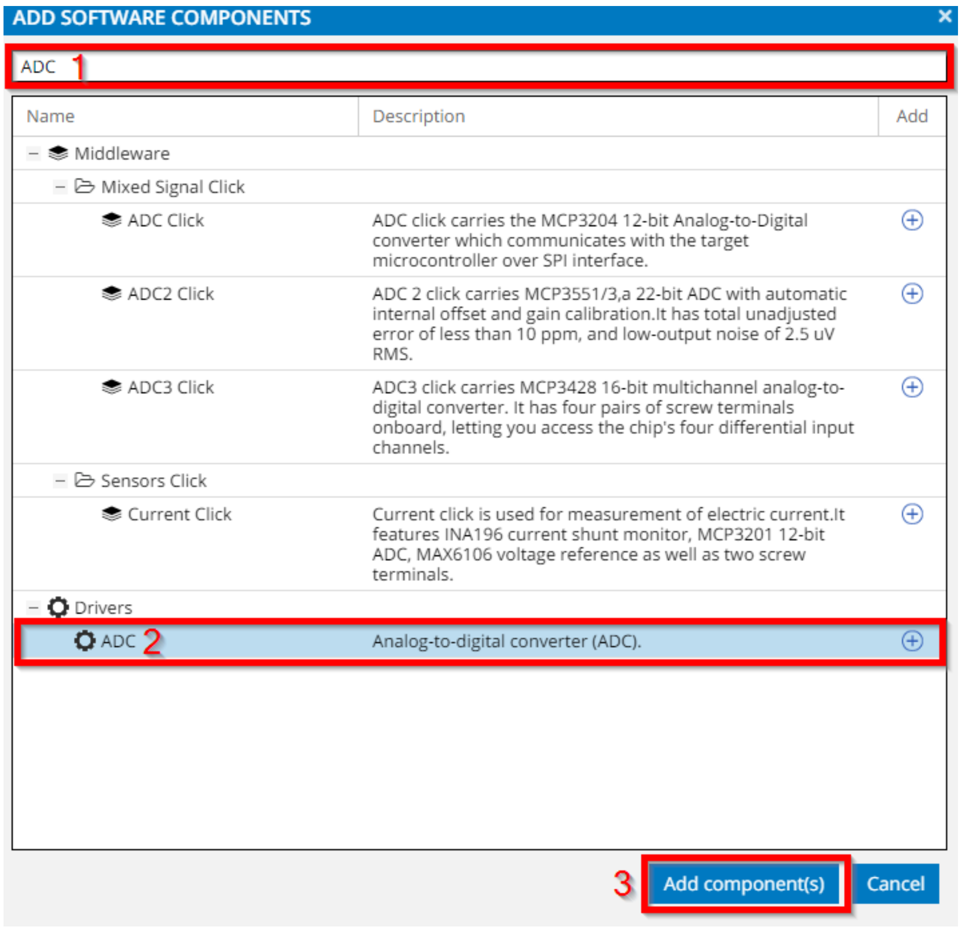
Driver:	2 HAL:Driver:USART_Sync
Mode:	UART
Instance:	SERCOM3

CLOCKS

COMPONENT SIGNALS

RX:	3 PA23
TX:	PA22

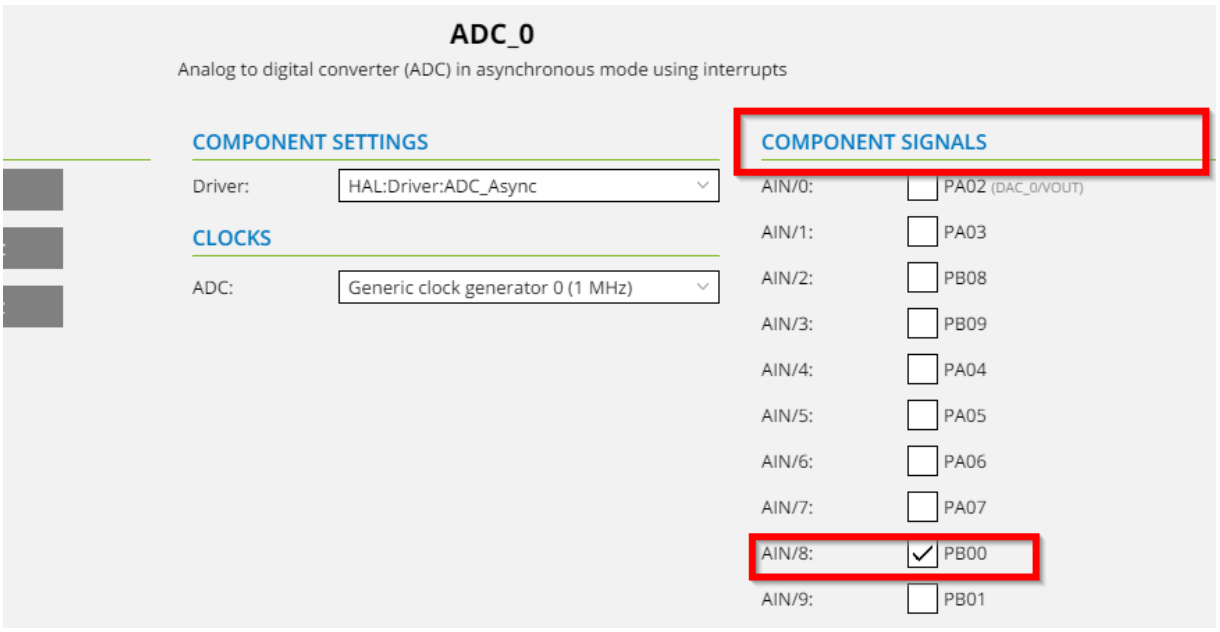
Next step we will add an ADC component to the project. Press “Add software component”, enter “ADC” in the filter text box, choose “ADC” in the Drivers category, and click add component.


ADD SOFTWARE COMPONENTS

ADC 1

Name	Description	Add
− Middleware		
− Mixed Signal Click		
ADC Click	ADC click carries the MCP3204 12-bit Analog-to-Digital converter which communicates with the target microcontroller over SPI interface.	⊕
ADC2 Click	ADC 2 click carries MCP3551/3,a 22-bit ADC with automatic internal offset and gain calibration.It has total unadjusted error of less than 10 ppm, and low-output noise of 2.5 uV RMS.	⊕
ADC3 Click	ADC3 click carries MCP3428 16-bit multichannel analog-to-digital converter. It has four pairs of screw terminals onboard, letting you access the chip's four differential input channels.	⊕
− Sensors Click		
Current Click	Current click is used for measurement of electric current.It features INA196 current shunt monitor, MCP3201 12-bit ADC, MAX6106 voltage reference as well as two screw terminals.	⊕
− Drivers		
ADC 2	Analog-to-digital converter (ADC).	⊕

In the ADC configuration component, we want to have an ADC input at pint PB00. Therefore we activate this pin in “COMPONENT SIGNALS”


ADC_0

Analog to digital converter (ADC) in asynchronous mode using interrupts

COMPONENT SETTINGS

Driver: HAL:Driver:ADC_Async

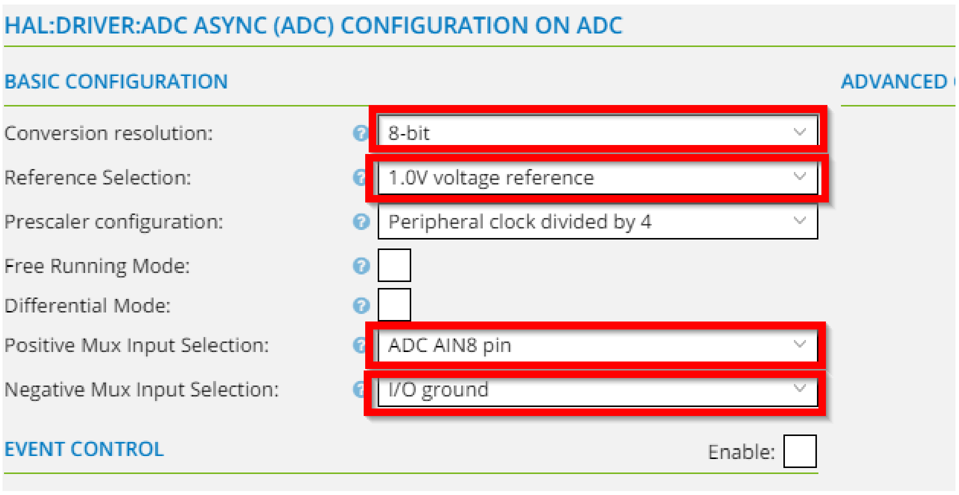
CLOCKS

ADC: Generic clock generator 0 (1 MHz)

COMPONENT SIGNALS

AIN/0:	☐	PA02 (DAC_0/VOUT)
AIN/1:	☐	PA03
AIN/2:	☐	PB08
AIN/3:	☐	PB09
AIN/4:	☐	PA04
AIN/5:	☐	PA05
AIN/6:	☐	PA06
AIN/7:	☐	PA07
AIN/8:	✓	PB00
AIN/9:	☐	PB01

After that, we want the ADC resolution is 8bit, the reference voltage is 1.0V, the positive input is PB00 and the negative input is I/O ground. Therefore, set up the configuration as follows:

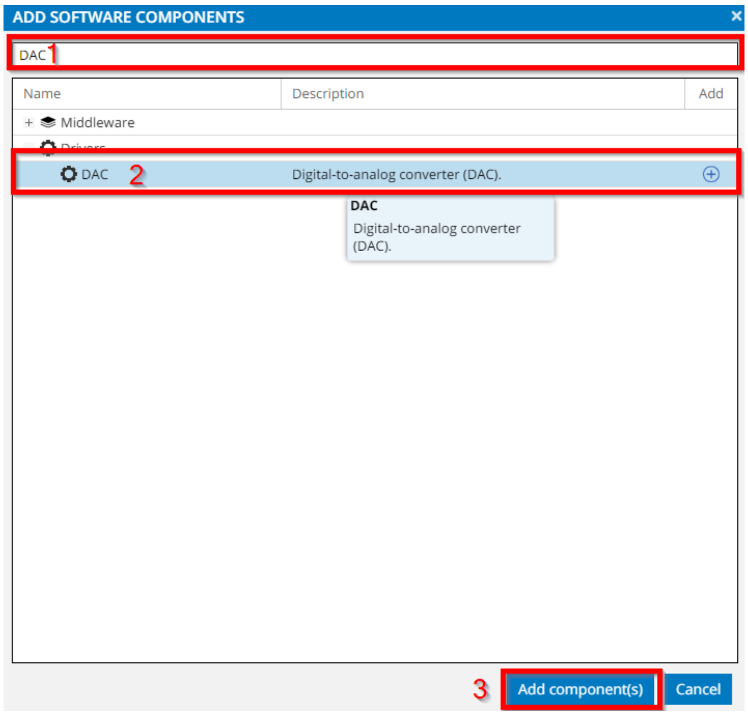

HAL:DRIVER:ADC ASYNC (ADC) CONFIGURATION ON ADC

BASIC CONFIGURATION

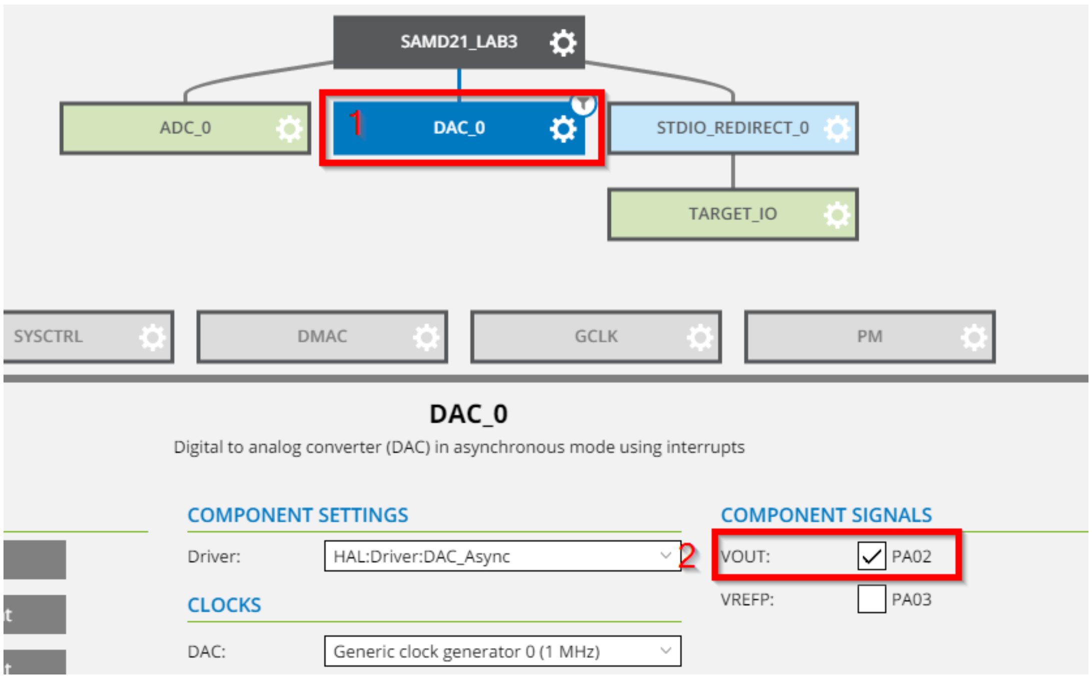
Conversion resolution:	8-bit
Reference Selection:	1.0V voltage reference
Prescaler configuration:	Peripheral clock divided by 4
Free Running Mode:	☐
Differential Mode:	☐
Positive Mux Input Selection:	ADC AIN8 pin
Negative Mux Input Selection:	I/O ground

EVENT CONTROL Enable: ☐

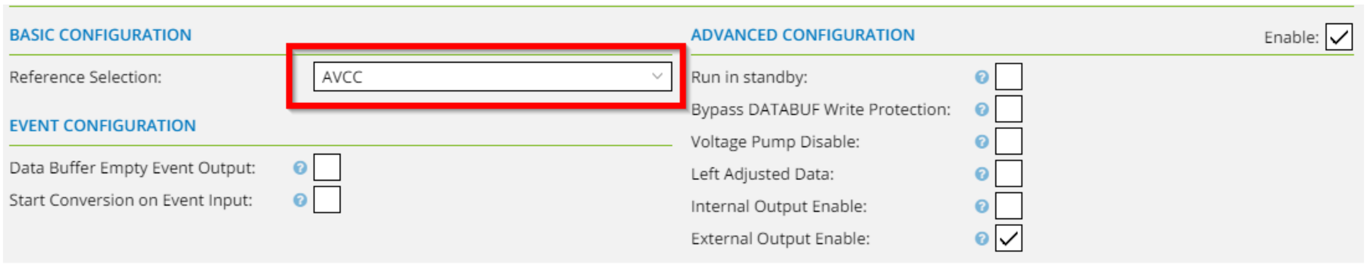
We add a DAC component by pressing “Add software component”, enter “DAC” in the filter text box and choose “DAC” under Drivers category



Click the DAC component and activate PA02 as out output pin.



We also must set the voltage reference for the DAC to 3.3V, so that our output ranges from 0.0V to 3.3V.


BASIC CONFIGURATION

Reference Selection: AVCC

EVENT CONFIGURATION

Data Buffer Empty Event Output: ☐

Start Conversion on Event Input: ☐

ADVANCED CONFIGURATION Enable: ✓

Run in standby: ☐

Bypass DATABUF Write Protection: ☐

Voltage Pump Disable: ☐

Left Adjusted Data: ☐

Internal Output Enable: ☐

External Output Enable: ✓

We have finished the configuration. Generate the code.

Writing the application

Compile the code to check if the code generation process causes no errors.

If errors happen, please start Atmel Studio and re-generate the project again.

We will check how to use the ADC by visiting the file "hal/include/hal_adc_async.h". The four functions that are interesting for us are: adc_async_enable_channel, adc_async_register_callback, adc_async_read_channel, and adc_async_start_conversion.

- The function adc_async_enable_channel enables the ADC feature on a channel.
- The function adc_async_register_callback allows us to register a callback function for an interrupt event, e.g. ADC conversion completed, ADC monitor or Error.
- The function adc_async_read_channel returns the converted value from a channel.
- The function adc_async_start_conversion starts the conversion.

The same process is applied for the DAC. We have a look at the file "hal/include/hal_dac_async.h" to understand how to use the DAC. For the DAC case, we need only two functions: dac_async_enable_channel and dac_async_write.

- The function dac_async_enable_channel enables DAC channel.
- The function dac_async_write sets the DAC output pin to a value.

With the information above, we go back to the "main.c" file to start developing the application. Before the main() function, we add the following snippet to initialize some variables and the callback function for the ADC conversion complete event.

You can also check file "examples/driver_examples.c" for the driver implementation

```
bool conversion_complete = false;
uint8_t adc_buffer = 0;
uint16_t dac_buffer = 0;

void ADC_ConversionComplete(const struct adc_async_descriptor *const descr, const uint8_t channel)
{
	conversion_complete = true;
}
```

In the main function, we will initialize the ADC and DAC peripherals.

```
printf("SAMD21_Lab3\n");

adc_async_register_callback(&ADC_0, 0, ADC_ASYNC_CONVERT_CB, ADC_ConversionComplete);
adc_async_enable_channel(&ADC_0, 0);
adc_async_start_conversion(&ADC_0);

dac_async_enable_channel(&DAC_0, 0);
dac_async_write(&DAC_0, 0, &dac_buffer, 1);
```

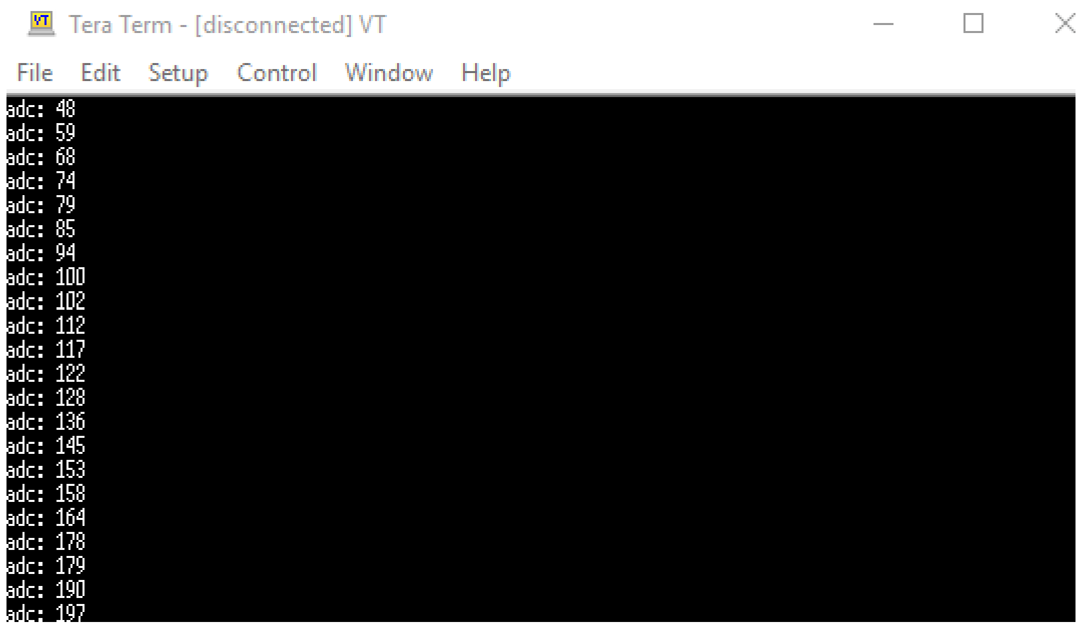
In the while(1) loop, we poll the "conversion_complete", read the ADC value and clear the flag. After that, we convert the ADC value into DAC value and set it to DAC output. We delay for a while before starting a new conversion.

```
while (1) {
	if(conversion_complete == true){
		adc_async_read_channel(&ADC_0,0,&adc_buffer,1);
		conversion_complete = false;
	}
	dac_buffer = (uint16_t)((1023 * adc_buffer)/255);
	printf("adc: %d\n",adc_buffer);
	dac_async_write(&DAC_0, 0, &dac_buffer, 1);
	delay_ms(100);
	adc_async_start_conversion(&ADC_0);
}
```

We are using the ADC in the one-shot mode so we must start a new conversion every time.

Compile the code, connect the hardware to the evaluation board and run the application.

You can see the value of the ADC on the terminal as follows:


Tera Term - [disconnected] VT
File Edit Setup Control Window Help

```
adc: 48
adc: 59
adc: 68
adc: 74
adc: 79
adc: 85
adc: 94
adc: 100
adc: 102
adc: 112
adc: 117
adc: 122
adc: 128
adc: 136
adc: 145
adc: 153
adc: 158
adc: 164
adc: 178
adc: 179
adc: 190
adc: 197
```

Connect an oscilloscope to pin PA02 to see the result of the DAC.

Congratulation! You have finished lab 3.

Contact information

Quang Hai Nguyen, B. Eng.
Field Application Engineer

P +49 6102 50308228
M +49 1511 6242003
quanghai.nguyen@arrow.com

Arrow Central Europe GmbH
Frankfurter Straße, 211
63263 Neu-Isenburg

THE END