BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 416 – Microprocessor and Embedded Systems Laboratory
January 2023

Experiment 05

Arm Cortex M: Analog Interfacing

Evaluation Form:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated to and verified by your lab instructor *before the end* of your scheduled lab period.

STEP	DESCRIPTION	MAX	SCORE
1	Blinking LED with Potentiometer using ADC	10	
2	Adjusting LED brightness using potentiometer and ADC	20	
3	Displaying the sine wave using DAC	20	
4	Generating a smoother version of the sine wave using a simple low pass filter	15	
5	Answering all questions in the lab sheet	35	
	TOTAL	100	

Signature of Evaluator: _____

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. ***You will not receive credit for this lab experiment unless this statement is signed in the presence of your lab instructor.***

"In signing this statement, I hereby certify that the work on this experiment is my own and that I have not copied the work of any other student (past or present) while completing this experiment. I understand that if I fail to honor this agreement, I will receive a score of ZERO for this experiment and be subject to possible disciplinary action."

Last Name (Printed):_____ **Lab Group:**_____ **Date:** _____

E-mail:_____**@**eee.buet.ac.bd **Signature:** _____

Contents

1 Introduction

1.1 Lab overview

At the end of this lab, you will be able to:

- Understand basic ADC concepts (resolution, successive-approximation)
- Map a GPIO pin to an ADC input
- Use the output of the ADC to control an output device (LED, Motor etc.)
- Understand basic concepts of DAC conversions
- Use the output of the DAC to visualise different waveforms
- Use the concept of filter design to make smooth continuous waveforms from DAC output

2 Pre-lab Study

Before attempting this lab, please do the following:

1. Make sure you have completed tasks for Experiment 03 and 04
2. Study:
   - Read [Zhu] Textbook **Chapter 20 Analog to Digital Conversion (ADC)**
   - Read [Zhu] Textbook **Chapter 21 Digital to Analog Conversion (DAC)**
   - Watch YouTube Tutorials on ADC (https://www.youtube.com/watch?v=h0CGtr4SC9s)
   - Watch YouTube Tutorials on DAC (https://www.youtube.com/watch?v=Pc1aFloxSMw&t=417s)

3 Analog to Digital Conversion (ADC)

Audio signals, analog sensor input waveforms, or input waveforms from certain types of input devices (sliders, rotary encoders, etc.) are all examples of analog signals which a microcontroller may need to process. To operate on these signals, a mechanism is needed to convert these analog signals into the digital domain; this process is known as analog-to-digital conversion.

In analog-to-digital conversion, an analog signal is read by a circuit known as an analog-to digital converter, or ADC. The ADC takes an analog signal as input and outputs a quantized digital signal which is directly proportional to the analog input. In an n-bit analog-to-digital converter, the voltage range of the digital logic family is divided equally into 2^n levels. A simple 3-bit (8-level) analog-to-digital quantization scheme is shown in figure 1, below:

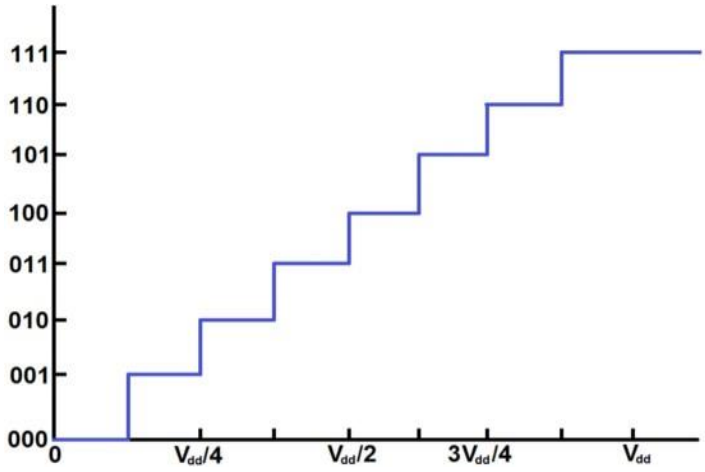


Figure 1: A 3-bit ADC quantization scheme.

3.1 Successive Approximation ADC

Successive Approximation is a broadly used method of converting an analog input to digital output. It has three main components: (a) successive approximation register (SAR), (b) comparator, and (c) control unit. See the figure below.

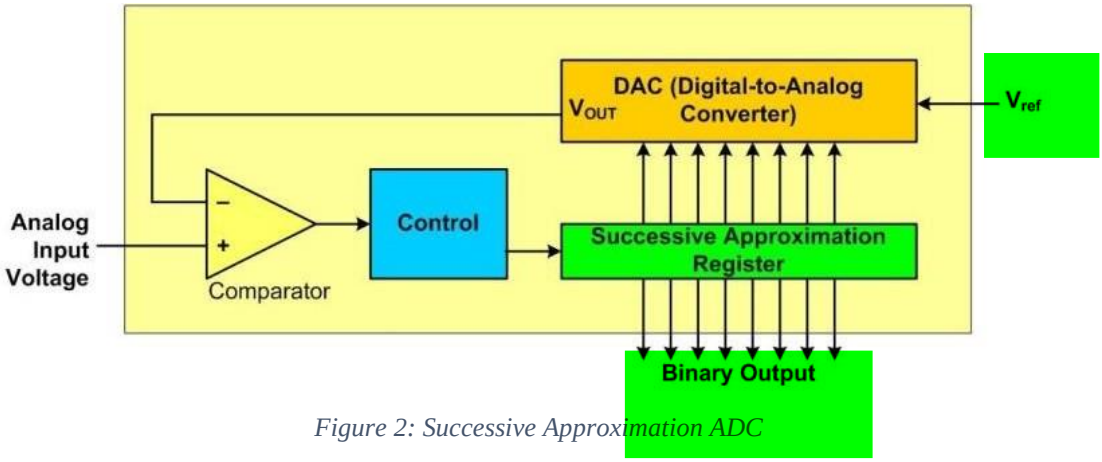


Figure 2: Successive Approximation ADC

The successive approximation register is loaded with only the most significant bit set at the start. An internal digital-to-analog converter converts the value of SAR to an analog voltage which is used to compare to the input voltage. If the input voltage is higher, the bit is kept. If the voltage is lower, the bit is cleared. The next bit is tried and the DAC and compare are exercised. This process is repeated for all bits of the SAR.

3.2 Programming the ADC

The STM32F446 Arm chip has three on-chip ADC modules. It can support up to 16 external analog input channels and three internal sources using multiplexing. The ADC can support a resolution of up to 12 bits. In the reference manual, the ADCs are designated as ADC1 (master), ADC2 (slave) and ADC3 (slave). We will use ADC1 in this experiment, the master ADC. In order to program the ADC1, we first need to know more about the different registers associated with it.

a) Programming the input pin in analog mode:

A GPIO pin can only be connected to a pre-defined input channel of an ADC module. For our purposes, we will use pin PA1 which is internally connected to one of the input channels of ADC1. To use PA1 as an analog input, we use the GPIO_MODER register as in previous experiments and set the corresponding bits (MODER1[1:0]).

GPIO port mode register (GPIOx_MODER) (x = A..H)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Complete the following statement to program pin PA1 in analog mode.

GPIO->MODER |=_____/*PA1 analog*/

b) Enabling the clock:

Like other peripheral modules, the ADC needs a clock to drive the conversion. The ADC clock is enabled by bits of RCC_APB2ENR (RCC APB2 peripheral clock enable register) register in the Reset and Control (RCC) section of the microcontroller as demonstrated in the following.

RCC APB2 peripheral clock enable register (RCC_APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SAI2 EN	SAI1 EN	Res.	Res.	Res.	TIM11 EN	TIM10 EN	TIM9 EN
								rw	rw				rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG EN	SPI4 EN	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Res.	Res.	USART6 EN	USART1 EN	Res.	Res.	TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw			rw	rw			rw	rw

Bit 8 ADC1EN: ADC1 clock enable

This bit is set and cleared by software. **0: ADC1 clock disabled, 1: ADC1 clock enabled**

Complete the given statement to enable the clock to ADC1.

```
RCC->APB2ENR |=_____/*enable ADC1 clock*/
```

c) Setting the sequence of ADC sampling:

ADC regular sequence registers decide the sequence of sampling from ADC components connected to the microcontroller. Say, for example, we have multiple analog devices such as a temperature sensor, a humidity sensor, and a moisture sensor connected to a microcontroller. We can set up a sequence where the temperature sensor is sampled first, then the humidity sensor, and then the moisture sensor. Even if there is only a single analog device connected to a microcontroller board, then we still use the ADC regular sequence register, specifying a single device to be sequenced.

Let us look at the regular sequence registers first:

ADC regular sequence register 1 (ADC_SQR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	L[3:0]				SQ16[4:1]			
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16_0	SQ15[4:0]					SQ14[4:0]					SQ13[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC regular sequence register 2 (ADC_SQR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SQ12[4:0]					SQ11[4:0]					SQ10[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10_0	SQ9[4:0]						SQ8[4:0]					SQ7[4:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC regular sequence register 3 (ADC_SQR3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SQ6[4:0]					SQ5[4:0]					SQ4[4:1]			
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4_0	SQ3[4:0]						SQ2[4:0]					SQ1[4:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Within the regular sequence registers, we specify the ADC channels we are using. For example, let's say that we want to sample from the analog device connected to channel 8, then sample from the analog device connected to channel 12, and then sample from the analog device connected to channel 9.

In this case, we want the first ADC sampling to come from channel 8; thus, for SQ1, we specify 1000, which is the binary for 8. Next, we want to sample from channel 12; thus, for SQ2, we specify 1100, which is the binary for 12. Next, we want to sample from channel 9; thus, for SQ3, we specify 1001, which is the binary for 9. And this can be continued for up to 16 conversions or sequences.

You can see this in the 3 registers as the sequences go from SQ1 to SQ16. If you are working with a single ADC component connected to a single channel, then you would just specify the ADC channel used in SQ1 and not have any values for any other registers. So, if you are using ADC1 as the first (the only) sequence, we set bit 0 to 1.

d) Setting the conversion sequence length:

The next thing we need to do is specify the sequence length (how many elements we are doing ADC sampling from). This is done through the ADC regular sequence register 1.

Bits 20 to 23 control the regular channel sequence length. This determines the total number of conversions in the regular channel conversion sequence. So, if we are sampling from 3 devices, the regular channel sequence length would be 3. Therefore, we set bits 20 to 23 to 0011. If we are sampling only from a single device, then the regular channel sequence length would be 1.

Complete the given statement to program the regular sequence registers such that the conversion starts at channel 1 with a conversion sequence length of 1:

ADC1->SQR3 =_____/*conversion sequence starts at channel 1*/

ADC1->SQR1 =_____/*conversion sequence length 1*/

e) Enabling/Powering on the ADC:

The ADC is powered on by setting the ADON bit in the ADC_CR2 (Control register 2) register. When the ADON bit is set for the first time, it wakes up the ADC from the Power-down mode. The following shows the control register 2 of the ADC.

ADC control register 2 (ADC_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	SWSTART	EXTEN		EXTSEL[3:0]				Res.	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ALIGN	EOCS	DDS	DMA	Res.	Res.	Res.	Res.	Res.	Res.	CONT	ADON
				rw	rw	rw	rw							rw	rw

Initially, the ADC is turned off by resetting the bit. Then, we program the regular sequence registers and finally, we turn on the ADC.

Complete the given statement to turn on the ADC.

ADC1->CR2 =_____/*enable the ADC*/

f) Triggering a conversion:

There are two ways to start an Analog-to-Digital conversion in STM32F4xx: (a) manual, and (b) automatic. The manual trigger is done by writing a 1 to the SWSTART (Start conversion of regular channels) in ADC_CR2 register (Refer to the ADC_CR2).

A free-running mode could be used by setting the CONT (Continuous conversion) bit in ADC_CR1 register. This bit is set and cleared by software. If set, conversion takes place continuously at the specified input channel. A trigger would no longer be required to start the conversion. It will start automatically at the end of the previous conversion.

Complete the given statement to trigger a conversion:

ADC1->CR2 =_____/*start a conversion*/

g) Waiting for a conversion to complete:

The end-of-conversion (EOC) is indicated by the EOC flag bit in ADC_SR (ADC Status Register) register. The following shows the ADC status register:

ADC status register (ADC_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OVR	STRT	JSTRT	JEOC	EOC	AWD
										rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

***This EOC bit is set by ADC itself at the end of each conversion of a channel. It is cleared by software or by reading the ADC_DR (ADC Data) register.**

Write a statement such that the code execution will be stuck at the statement until an end of a conversion is reached.

```
_____ /*ADC_SR bit 1 is set when end of conversion is reached*/
```

h) Read conversion result:

The result of the ADC conversion is stored in the ADC_DR (ADC Data Register). As soon as the EOC flag goes high, the ADC_DR needs to be read to avoid any overrun error condition. The following shows the ADC data register:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Complete the following statement to store the result of the ADC conversion to the ADC data register:

```
uint32_t result;

result =_____/*read ADC result*/
```

The value stored in the ADC_DR register ranges from 0 to 4095 (12^{th} power of 2 due to a 12-bit resolution). We can use this value for further processing.

After following the steps mentioned above, we are now ready to perform a simple experiment with the ADC by using it to turn an LED on under certain conditions.

3.3 Turning on an LED using Potentiometer

A potentiometer, informally a pot, is a three-terminal variable resistor. It uses a sliding contact and works as an adjustable voltage divider. When two outer terminals are connected to V_{cc} and the ground respectively,

the center terminal generates a voltage that varies from 0 to V_{cc} depending on the position of the sliding contact. In the following sections, we use the internal voltage reference, which is 5V.

In the following example, we measure the input voltage adjusted by a potentiometer. If the input voltage V_{input} is larger than ½ of V_{cc}, then we turn on an LED.

Example:

Circuit Diagram:

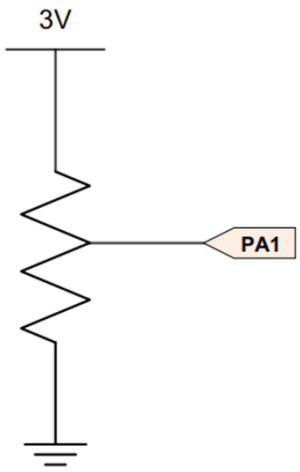


Fig: Measurement of voltage from a potentiometer-based divider at pin **PA1**.

We will make a simple system using the ADC in the STM32 board. The code provided inside the folder "ADC project" in ba*se_code.zip* reads the voltage read by the ADC; turns on the on-board green LED (LD2, connected to PA5) if the voltage recorded at the ADC pin is greater than 50% of the reference voltage. If less than the mentioned voltage, the LED is turned off.

After constructing the circuit diagram, we write the code following the steps described in the previous section. It is important to note that since our ADC has a resolution of 12 bits and gives us a result ranging from 0 (Input voltage = 0) and 4095 (Input voltage = V_{cc}), then we set the threshold for turning on the LED to 2048 (V_{cc}/2).

Lab Exercise:

Using the same circuit diagram, write a code such that the brightness of the on-board green LED (LD2, connected to PA5) can be controlled by turning the potentiometer.

4 Digital to Analog Conversion (DAC)

The digital-to-analog converter (DAC) is a device used to convert digital signals to analog signals. There are two methods of making a DAC: (a) binary weighted DAC, and R/2R ladder. Most integrated circuit DACs use the R/2R method since it can achieve a much higher degree of precision. The first criterion for selecting a DAC is its resolution, which is a function of the number of bits of the digital input. The most common DACs come in 8, 10, and 12 bits of resolution. The number of digital input bits decides the resolution of the DAC since the number of analog output levels is equal to 2^n, where n is the number of digital input bits. Therefore, an 8-bit DAC provides 256 discrete voltage (or current) levels of output. See the figure below.

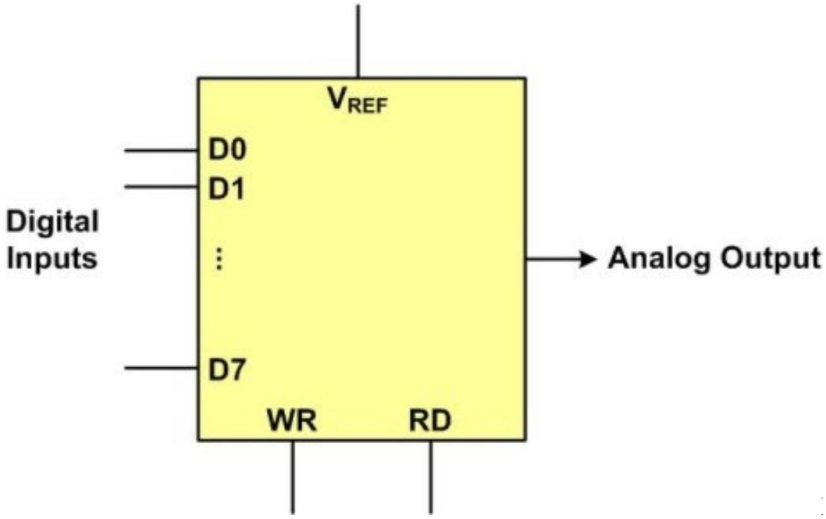


Figure 3: Block Diagram of DAC

4.1 R-2R Ladder DAC

The R-2R layout is a straightforward arrangement that comprises of parallel and series resistors connected to an operational amplifier. Depending on the polarity of the output voltage from the DAC, an operational amplifier can be employed in either inverting or non-inverting mode. Along with the entire network, R-2R ladder resistors operate as voltage dividers, with the output voltage determined by the input voltages.

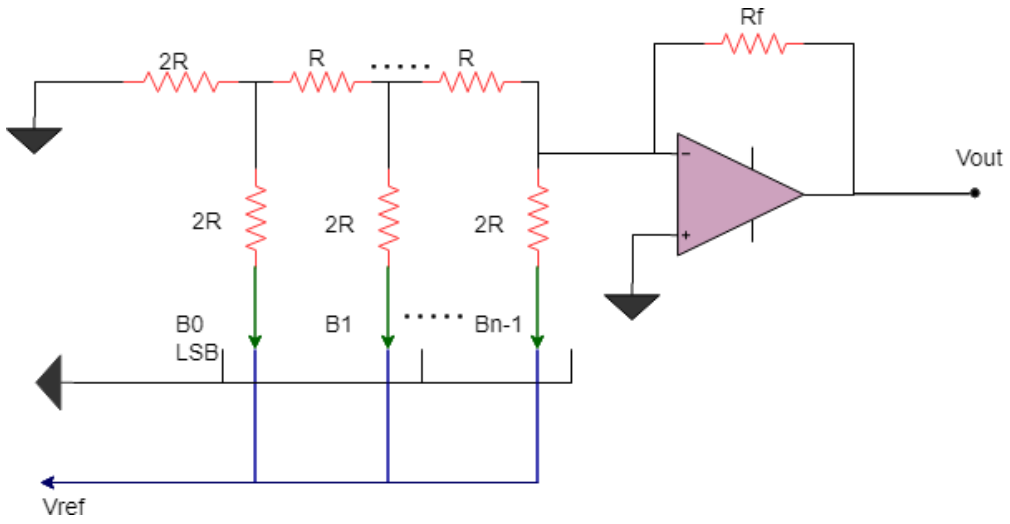


Figure 4: R-2R Ladder DAC[Ref]

The ladder configuration is made up of two resistors: a base resistor R, and a second resistor that is twice the amount of the base resistor. This feature helps in maintaining an accurate output analog signal while avoiding the need of a large variety of resistor values. Binary switches connected to V_{ref} and GND give digital inputs for binary 1, and 0, respectively.

Ref : https://microcontrollerslab.com/r-2r-ladder-dac-digital-to-analog-converter-working-examples-circuits/

4.2 *Programming the DAC*:

Many of the STM32F4xx Arm chips have on-chip digital-to-analog converters (DACs). The STM32F446RE comes with two on-chip DACs. They can be configured for independent or simultaneous conversions. They are designated as DAC1 and DAC2. Each one has its own analog output pin.

In STM32F446RE, the analog output pins are PA4 and PA5 for channel 1 and channel 2, respectively. On-chip DAC has a 12-bit resolution, but it can also be configured as an 8-bit DAC too. That means digital data loaded into the DAC register can be 8-bit or 12-bit. We will be using the DAC1 for this experiment. So, let us now look at how we can program the DAC1 to take a digital input and generate an analog output using the different registers associated with it:

i) Programming the input pin in analog mode:

We first program the pin PA4 in analog mode using the same GPIOA->MODER register as we have previously done for ADC.

Complete the following statement to program pin PA4 in analog mode.

GPIO->MODER |=_____/*PA4 analog*/

j) Enabling the clock:

Same as the ADC, the DAC also needs a clock to drive the conversion. The ADC clock is enabled by bits of RCC_APB1ENR (RCC APB2 peripheral clock enable register) register in the Reset and Control (RCC) section of the microcontroller as demonstrated in the following:

RCC APB1 peripheral clock enable register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DAC EN	PWR EN	CEC EN	CAN2 EN	CAN1 EN	FMPI2C1 EN	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	SPDIFRX EN
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Res.	Res.	WWDG EN	Res.	Res.	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 29 DAC1EN: DAC1 clock enable

This bit is set and cleared by software. **0: DAC1 clock disabled, 1: DAC1 clock enabled**

Complete the following statement to enable the clock to ADC1.

RCC->APB1ENR |=_____/*enable DAC1 clock*/

k) Enabling/Powering on the DAC:

Similar to the ADC, the DAC1 is powered on by setting the EN1 bit in the DAC_CR1 (Control register 1) register. It is important to note that the DAC has separate enable bits for the two channels. The following shows the control register 1 in the DAC.

DAC control register 1 (DAC_CR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Complete the following statement to turn on the DAC.

DAC1->CR1 = _____/*enable the DAC*/

l) Loading digital data into DAC:

To make programming easier, the STM32 Arm uses many data registers. The registers are called DAC Data Holding Registers (DAC_DHR). Writing new data into DAC holding register starts a conversion. Besides holding register, we also have DAC Data Output registers (DAC_DOR). We can only write to DAC_DHR register.

According to STM32 reference manual: "**Data stored in the DAC_DHRx register are automatically transferred to the DAC_DORx register after one clock cycle**." Again, according to reference manual "**The Digital inputs are converted to output voltages on a linear conversion between 0 and VREF+**". The analog output voltages on each DAC channel pin are determined by the following equation:

$$DAC_{output} = V_{ref} \times \frac{DAC\_DOR}{4096}; V_{ref} = 3.3V$$

It must be emphasized that "**The DAC_DORx cannot be written directly and any data transfer to the DAC channelx must be performed by loading the DAC_DHRx register**". The following shows the DAC1 data holding register and DAC1 data output register.

DAC channel1 12-bit right-aligned data holding register (DAC_DHR12R1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DACC1DHR[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

DAC channel1 data output register (DAC_DOR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	DACC2DOR[11:0]											
				r	r	r	r	r	r	r	r	r	r	r	r

Question: Suppose a value of 2555 has been loaded into the DAC_DHR12R1, what value do we get as analog output?

Now that we have an understanding of how to program a DAC, we will now use our knowledge to generate a sinusoidal analog waveform from digital input.

4.3 *Generating a Sinusoid*:

To generate a sine wave, we first need a set of discrete values that represent the voltage-level of the sine wave across its entire period, and then write the corresponding voltages at the DAC output pin. The value of the sine wave can vary between -1.0 to +1.0. We will need to translate this range to 0-3.3V and represent each volage level by an integer between 0 and 4095.

Fill up the following table showing the angles, the sine values, the voltage levels, and the corresponding integer values representing the voltage for each angle with 30-degree increments. Assume that full scale output for DAC is V_{ref} (3.3V). These values will be saved in the code in the form of a look-up table.

Here, $V_{out} = \frac{3.3V}{2}(1 + \sin\theta)$

Angle, θ	Sin θ	V_{out} (Voltage Mag.)	Values sent to DAC
0			
30			
60			
90			
120			
150			
180			
210			
240			
360			

Now, the first task is to program the DAC as discussed in the previous section. Since we are loading discrete integer values into the DAC, we will get discrete amplitudes of the sinusoidal waveform as output. Of course, we can hold the output at that discrete value for as long as want by adding a delay before the next conversion takes place.

Code snippet:

```
void delayUs(int n)
{
int i;
for (; n > 0; n--)
for (i = 0; i < 3; i++) ;
}
```

For example, let us set a delay of 10us using the code snippet provided above. Now, if we observe the output of PA4 at the oscilloscope, it should show a waveform as follows:

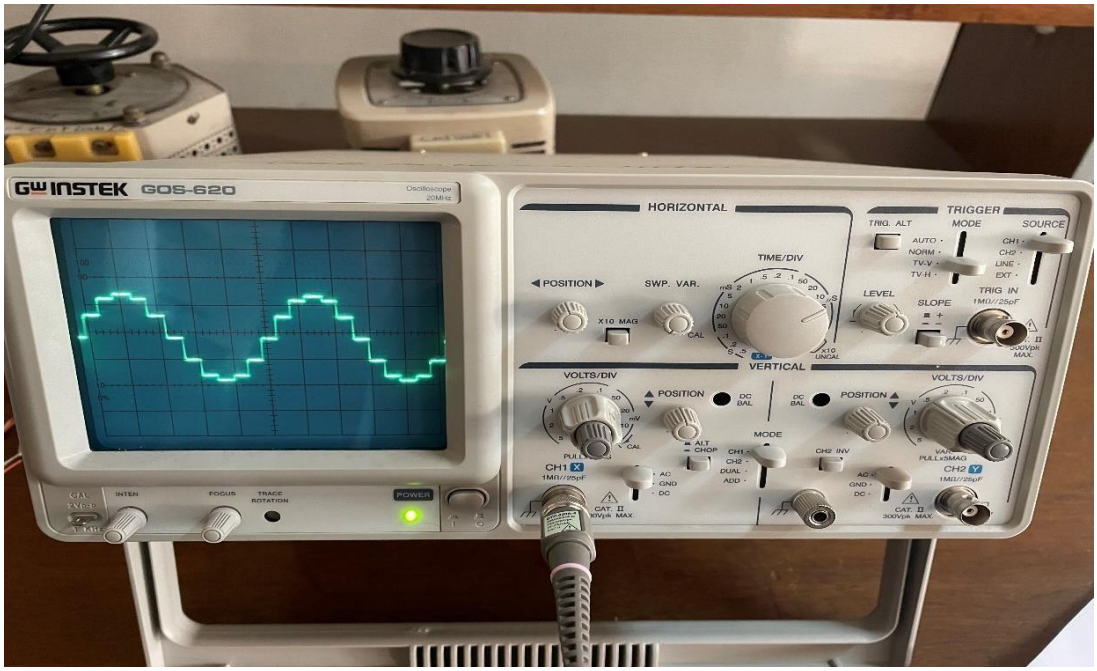


Figure 5: Waveform of a quantized sinusoid

As expected, the output is continuous in time, but discrete in amplitude. The different voltage levels correspond to the different integer values loaded into the DAC1. The time for which the output stays constant is determined by the delay before the next conversion.

Lab Exercise:

a) Determine the fundamental frequency of the observed sine wave. Devise a method to change this frequency of the apparent sinusoid by modifying the code.
b) Construct a simple low-pass filter circuit to obtain a smoother sine wave from the staircase-like sine wave (DAC output). Use resistors/capacitors/inductors as required.

Additional Exercise:

Update the given DAC code that will generate the following analog signals: (a) triangular wave, (b) saw-tooth wave, (c) a full-wave rectified sine wave, (d) Random noise.

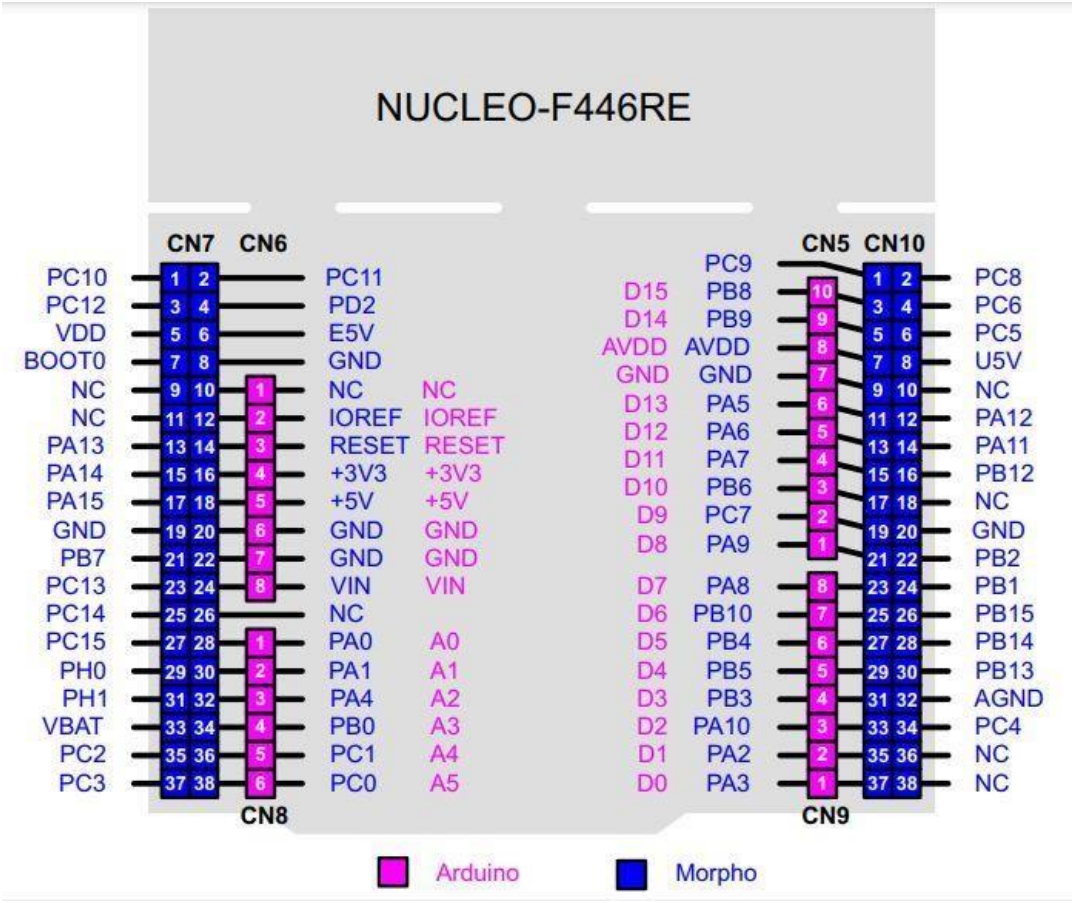
5 Code Printout

After this page, please attach printout of the main codes for each section. Use monospace fonts (Such as ***Noto Mono***) for your codes. Use font size 8. Paste them in Microsoft Word. Give appropriate heading for each problem (Such as **Lab Exercise of Section 3**, etc)

6 Appendix

6.1 Appendix A: Pin Connections on Nucleo-64 board

Board Component	Microcontroller Pin	Comment
Green LED	PA 5	SB42 closed and SB29 open by default
Blue user button	PC 13	Pulled up externally
Black reset button	NRST	Connect to ground to reset
ST-Link UART TX	PA 2	STLK_TX
ST-Link UART RX	PA 3	STLK_RX
ST-Link SWO/TDO	PB 3	Trace output pin/Test Data Out pin
ST-Link SWDIO/TMS	PA 13	Data I/O pin/Test Mode State pin
ST-Link SWDCLK/TCK	PA 14	Clock pin/Test Clock pin

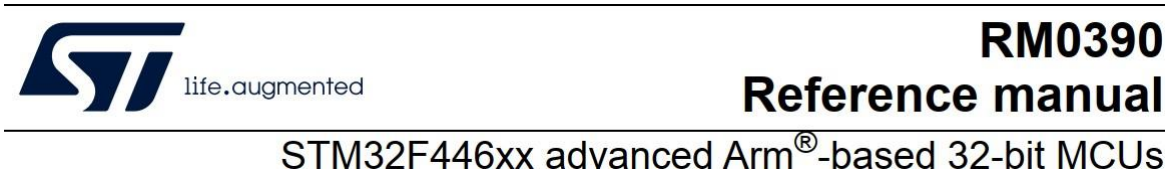


6.2 Appendix B: Acknowledgement and References

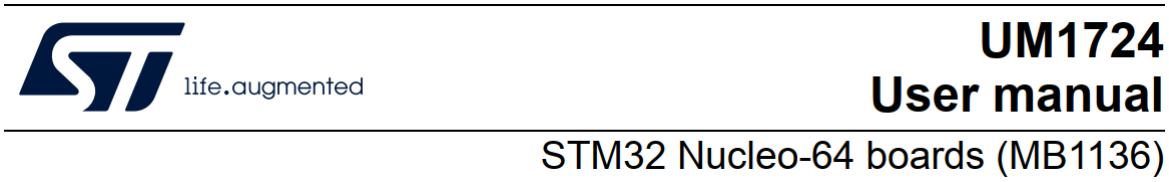
The labsheet is prepared by **Shahed Ahmed**, **Shafin Bin Hamid,** and **Ramit Dutta,** Dept of EEE, BUET, on 08/08/2022 and edited by **Md. Jawad Ul Islam** on 21/07/2023 according to Jan 2023 term.

The labsheet is based on Lab Materials provided as Instructor Supplement of "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, Third Edition." By Dr. Yifeng Zhu and, "STM32 Arm Programming for Embedded Systems" by Muhammad Ali Mazidi, Shujen Chen and Eshragh Ghaemi.

The following documents are strongly recommended as reference:

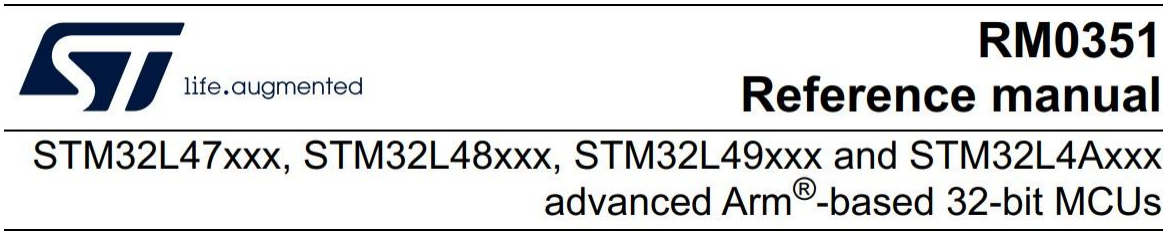


https://www.st.com/resource/en/reference_manual/rm0390-stm32f446xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf



https://www.st.com/resource/en/user_manual/dm00105823-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf

The following Document is recommended for Yifeng Zhu's book literature



https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf