**Current Measurement using ADC in STM32F407 Microcontroller**

**1. Aim:**

Measure the AC/DC current in a power circuit using a LEM LA-55P current sensor and a STM32F407 Discovery Board.

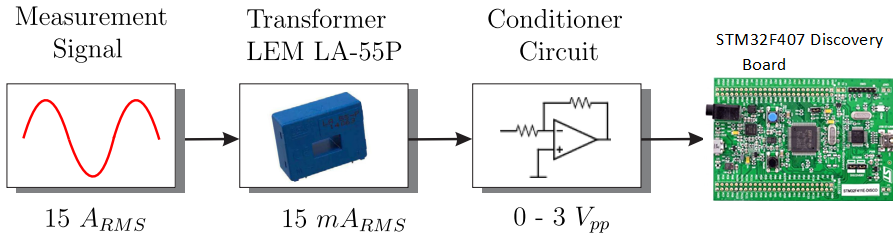
**2. Introduction:**

An ADC (Analog-To-Digital) converter is an electronic circuit that takes in an analog voltage as input and converts it into digital data, a value that represents the voltage level in binary code. The ADC samples the analog input whenever you trigger it to start conversion. And it performs a process called quantization so as to decide on the voltage level and its binary code that gets pushed in the output register.

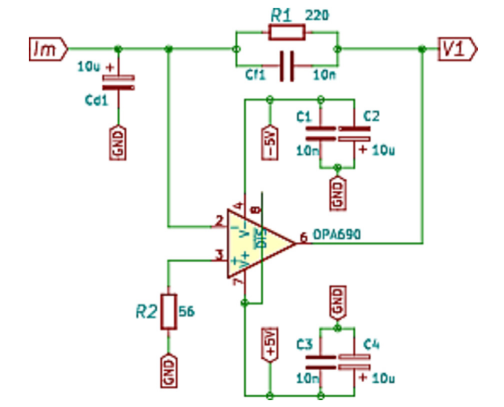
The ADC is one of the most expensive electronic components especially when it does have a high sampling rate and high resolution. Therefore, it’s a valuable resource in microcontrollers and different manufacturers provide us (the firmware engineers) with various features so as to make the best use of it. And the flexibility also to make a lot of decisions like sacrificing resolution in exchange for a higher resolution or having the ADC to trigger on an internal timer signal to periodically sample the analog channels, and much more as we’ll see in this tutorial.

**3. Hardware Circuit:**

The current measurement scheme and conditioning circuit consist of three stages, as shown in Figure. The first stage is responsible for obtaining a voltage of 3 Vpp proportional to the output of the current sensor LEM LA 55-P. For this reason, a trans-impedance amplifier is used, as shown in the figure below.



**Current Measurement System**



**Trans-impedance Amplifier Stage**

Resistor R1 sets the gain for this stage.

Subsequently, the next stage generates a reference of 1.5 V, used for an offset voltage. Finally, V1 is added with 1.5V to provide the offset. The final output voltage of signal conditioning circuit is 0-3V.

The firmware has been implemented on the proposed experiment using the following flowchart.

To measure the current, collect the samples for 1second or 8000 samples. Using those samples, calculate the Vrms value as given in the equation.

A picture containing font, text, white, design

Description automatically generated

Finally, to convert it into a current value Irms use the following equation.

Irms = Vrms x Scaling Factor

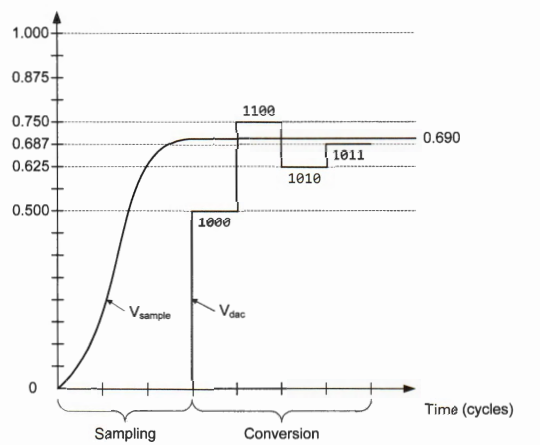
**4. ADC Registers Configuration:**

An analog-to-digital converter (ADC) produces a finite-precision signed or unsigned digital number to represent approximately the size of an analog voltage relative to a reference voltage. The reference voltage is a fixed voltage provided by the internal circuit of the microprocessor or by an external circuit connected to a pin of the microprocessor. It does not convert a voltage larger than the reference voltage.

There are three most popular ADC architectures: sigma-delta ADC for low-speed applications, successive-approximation (SAR) ADC for low-power applications, and pipelined ADC for high-speed applications.

Sigma-delta ADCs are mostly used in applications requiring low sampling rates, but high resolution, typically less than 100 kilo samples per second and 12 to 24- bit resolution, such as voice band and audio applications. They have been widely used in modern cell phones.

SAR ADCs are suitable for applications with low-power data acquisitions and moderate sampling rates, typically less than 5 million samples per second (MSPS). If the ADC has a resolution of n bits, the successive-approximation conversion takes n steps to complete. It plays a tradeoff between the resolution and sampling rate. A higher resolution usually reduces the ADC conversion rate. An example of four-bit successive-approximation (SAR) ADC. Suppose the input voltage is between 0 and 1 V.



The figure above shows an example of converting an input voltage of 0.690V into a 4-bit binary value by using SAR ADC. Suppose the range of the input voltage is between 0V and 1V.

After the input voltage is sampled, the SAR control logic starts the conversion by setting Vdac as O.5V (i.e., the ADC output is 0b1000) and comparing it with Vsample.

Since Vsample is larger than Vdac, the control logic then sets Vdac as 0.750V (i.e., the ADC output is 0b1100).

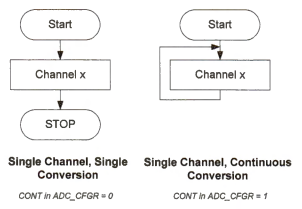
Because Vsample is smaller than Vdac, the control logic then sets Vdac as 0.625V.

The above process repeats, and the final output of the ADC is 0b1011. In this example, the conversion process takes four cycles.

In the single-end mode, if the ADC result has n bits, we can calculate the conversion result of an input voltage V as follows:

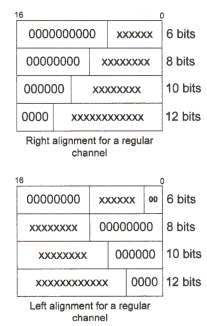
**4.1 ADC Conversion:**

The conversion operation can be set up to perform only once or repeatedly, depending on bit CONT bit in the ADC\_CFGR register. After each conversion, ADC result should be copied to a user buffer because ADC may overwrite the ADC data registers. An interrupt request or a DMA request can be triggered at the end of each conversion if enabled. Thus, to reduce the software overhead, we often use the ADC interrupt handler or the DMA controller to copy the ADC results to a user buffer.



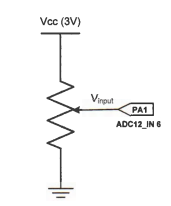
**4.2 ADC Data Alignment:**

Software can change the ADC resolution. The resolution can be either 12, 10, 8 or 6 bits, determined by the RES[1:0] bits in the ADC configuration register ADC\_CFGR. However, each ADC data register (DR) has 16 bits. Because ADC results have fewer bits than ADC data registers, alignment must be considered when a data result is stored in a data register. Figure 20-8 shows different data alignment formats. ADC output data registers can be either right-aligned or left-aligned.

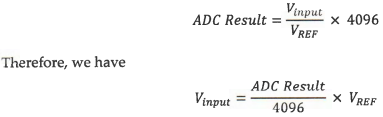


**4.3 Measuring the Input Voltage:**

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 VCC and the ground respectively, the center terminal generates a voltage that varies from 0 to VCC depending on the position of the sliding contact. In the following sections, we use the internal voltage reference, which is 3V. In this example, we measure the input voltage adjusted by a potentiometer. If the input voltage Vinput is larger than 1/2 of VCC, then we turn on an LED. An interesting application is that we use the potentiometer to control the brightness of an LED dynamically if a PWM controls the LED. The Vinput is used to adjust the duty cycle of the PWM output signal.



Suppose the ADC result has 12 bits and ADC is configured as single-ended. Then, we have the following conversion result:



**5. Program for Current Measurement using ADC:**

#include "stm32f4xx.h"

void ADC\_Init(void);

void ADC\_Enable(void);

void ADC\_Start(int);

float32\_t value[8500],value1[8500],sum,RMS,out,currentRMS;

uint8\_t ADC\_VAL[8500];

uint32\_t len=8500;

float32\_t step=1.5/128;

void ADC\_Init (void)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* STEPS TO FOLLOW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Enable ADC and GPIO clock

2. Set the prescalar in the Common Control Register (CCR)

3. Set the Resolution in the Control Register 1 (CR1)

4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

5. Set the Sampling Time for the channels in ADC\_SMPRx

6. Set the Regular channel sequence length in ADC\_SQR1

7. Set the Respective GPIO PINs in the Analog Mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//1. Enable ADC and GPIO clock

RCC->APB2ENR |= (1<<8); // enable ADC1 clock

RCC->AHB1ENR |= (1<<0); // enable GPIOA clock

RCC->CFGR |= 6<<13; // set APB2 = 2 MHz (16/8)

//2. Set the pre-scalar in the Common Control Register (CCR)

ADC->CCR |= 0<<16; // PCLK2 divide by 2

//3. Set the Resolution in the Control Register 1 (CR1)

//ADC1->CR1 &= ~(1<<8); // SCAN mode disabled, enable for multichannel use

ADC1->CR1 |= (2<<24); // 8 bit RES

//4. Set the Continuous Conversion, EOC, and Data Alignment in Control Reg 2 (CR2)

//ADC1->CR2 |= (1<<1); // enable continuous conversion mode

ADC1->CR2 |= (1<<10); // EOC after each conversion

ADC1->CR2 &= ~(1<<11); // Data Alignment RIGHT

//5. Set the Sampling Time for the channels

//ADC1->SMPR2 &= ~(1<<0); // Sampling time of 112 cycles for channel 0

ADC1->SMPR2|=(5<<0);

//6. Set the Regular channel sequence length in ADC\_SQR1

//ADC1->SQR1 &= ~(1<<20); // SQR1\_L =0 for 1 conversion

//7. Set the Respective GPIO PIN in the Analog Mode

GPIOA->MODER |= (3<<0); // analog mode for PA 0 (channel 0)

}

void msDelay(uint32\_t msTime)

{

/\* For loop takes 4 clock cycles to get executed. Clock frequency of stm32f407 by default is 16MHz

So, 16MHz/4=4MHz. If we want 1000ms delay, 4MHz/1000=4000, so we have to multiply by 4000 to get a delay of 1s

\*/

for(uint32\_t i=0;i<msTime\*3000;i++)

{

\_\_NOP();

}

}

int main ()

{

ADC\_Init ();

ADC1->CR2 |= 1<<0; // ADON =1 enable ADC1

uint32\_t delay = 10000;

while (delay--);//Wait sometime for ADC to start

//ADC1->CR2 |= (1<<30); // start the conversion

while (1)

{

for(int i=0;i<8500;i++)

{

ADC1->CR2 |= (1<<30); // start the conversion

ADC1->SR = 0; // clear the status register

//ADC1->CR2 |= (1<<30); // start the conversion

while (!(ADC1->SR & (1<<1))); // wait for EOC flag to set

ADC\_VAL[i]=ADC1->DR;

}

//((float)value[i]-128)\*step

//float step=1.5/128

for(int j=0;j<8500;j++)

{

value[j]=((float)ADC\_VAL[j]);

value1[j]=(((float32\_t)ADC\_VAL[j]\*0.0118)-1.506);

}

arm\_rms\_f32(value1,len,&RMS);

//sum=sum/8500;

//arm\_sqrt\_f32(sum,&out);

currentRMS=RMS\*1.658;

}

}

**6. Conclusion:**

In conclusion, current from the current sensor has been measured using the ADC in STM32F407.