MCP3903 Arduino Library

Precise and Efficient ADC Interface for MCP3903 (6-Channel Simultaneous Sampling ADC)

This Arduino library provides a comprehensive interface for interacting with the **MCP3903**, a high-performance analog front-end featuring six 24-bit delta-sigma ADC channels with simultaneous sampling capability. This library enables configuration, reading, and real-time monitoring of voltage and current data from MCP3903 across all six channels.

Features

- Supports **6-channel simultaneous sampling**
- Supports both **24-bit and 16-bit modes**
- Easily configurable **Oversampling Ratio (OSR)** and **Prescaler**
- Enables/Disables **Internal Voltage Reference**
- SPI-based communication abstraction
- **Individual channel gain** and **phase shift configuration**
- Functions for **reading 1, 2, or all 6 ADC channels**
- Register-level **read and write functions**

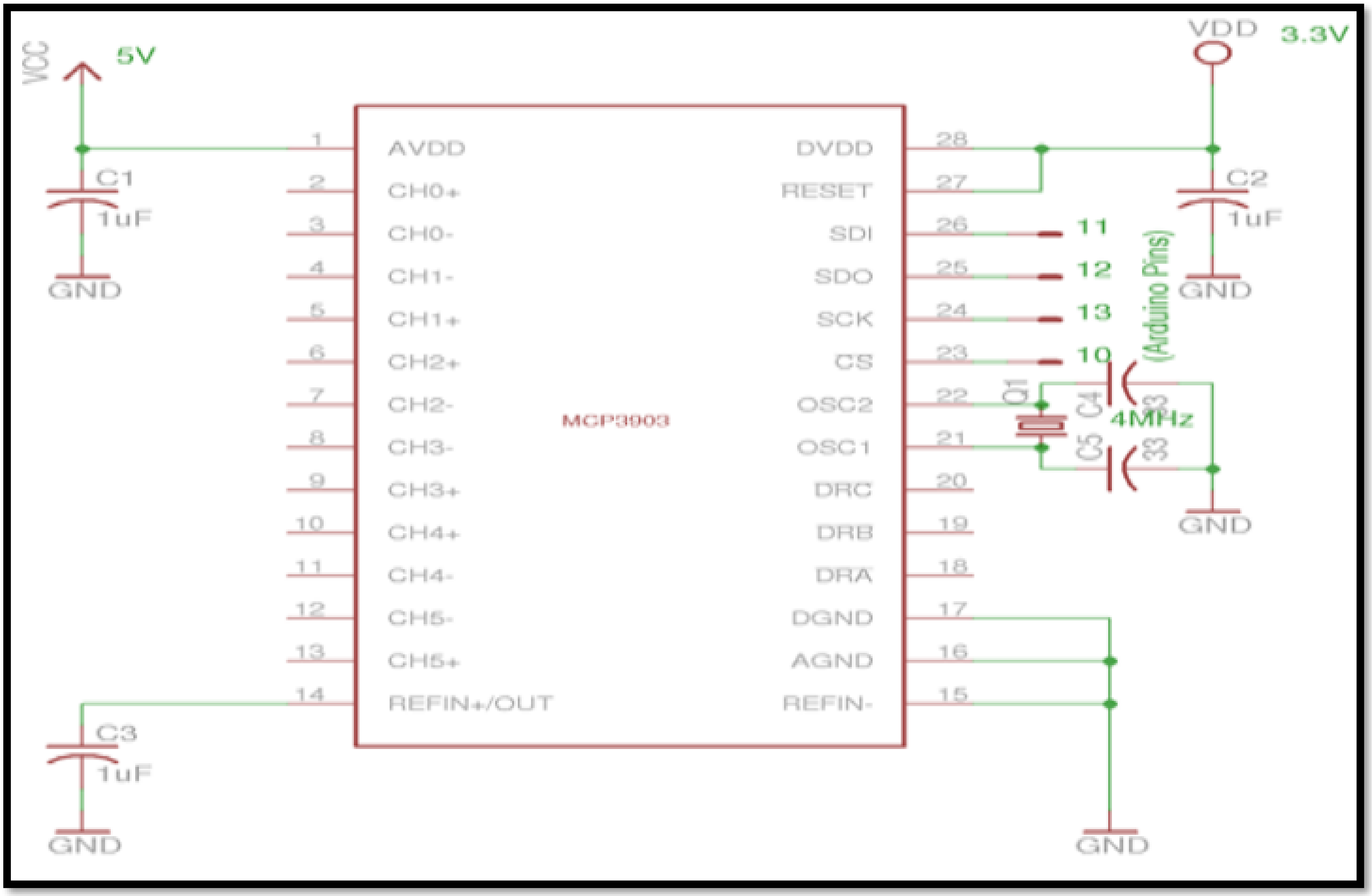
Hardware Requirements

- **MCP3903 ADC IC**
- Compatible microcontroller (e.g., ESP32, Arduino Mega)
- SPI Communication (MOSI, MISO, SCK, CS)
- Optional: Crystal oscillator for MCP3903 (e.g., 4 MHz or 8.192 MHz for boost mode)

Pin Configuration (Default for ESP32)

MCP3903 Pin	ESP32 Pin
MOSI	23
MISO	19
SCK	18
CS	5

Modify MCP3903.h if you're using different pins.



Installation

1. Clone or download this repository.
2. Place the folder in your Arduino `libraries` directory.
3. Include it in your sketch:

```
#include <MCP3903.h>
```

Basic Usage Example

```
#include <MCP3903.h>

MCP3903 adc;

void setup() {
  Serial.begin(115200);

  adc.reset();  // Reset all settings to default

  adc.init_config(MCP3903::o64, MCP3903::p1, 0, 0, 1, 1, 1, 1, 1, 1, 1);
  adc.init_status(MCP3903::group, 24, 0, MCP3903::lag, MCP3903::lag, MCP3903::lag);

  adc.Gain(0, MCP3903::g1);  // Set Gain for Channel 0
  adc.phase('a', 0);         // No phase shift
}

void loop() {
  double val;
  val = adc.readADC(0);  // Read Channel 0
  Serial.println(val);
  delay(100);
}
```

API Reference

reset()

Resets MCP3903 to default settings.

init_config(osr, ps, E_vref, E_clk, dither, ch0, ch1, ch2, ch3, ch4, ch5)

Configure sampling and hardware behavior.

Param	Description
osr	Oversampling ratio (o32, o64, o128, o256)
ps	Prescaler (p1, p2, p4, p8)
E_vref	0: Enable internal VREF, 1: Disable
E_clk	0: Use external crystal, 1: Use clock mode
dither	1: Enable dithering, 0: Disable
chX	0: Disable channel X, 1: Enable channel X

init_status(loop, width, link, DRA, DRB, DRC)

Configure output data formatting and data-ready behavior.

Param	Description
loop	Read looping mode (no, group, type, all)
width	16 or 24-bit output
link	1: Link DR outputs, 0: No link
DRA/B/C	DR output configuration (lag, first, second, both)

readADC(channel)

Read data from a specific ADC channel (0 to 5).
Automatically detects bit mode (16 or 24).

read2ADC(ch1, ch2, result1, result2)

Simultaneously reads two channels.

```
double r1, r2;
adc.read2ADC(0, 1, r1, r2);
```

read6ADC(...)

Reads all 6 channels simultaneously.

```
double a, b, c, d, e, f;
adc.read6ADC(a, b, c, d, e, f);
```

readRegister(byte reg)

Read raw 24-bit data from a specific register.

writeRegister(byte reg, unsigned long data)

Write raw 24-bit data to a register.

Gain(channel, gain)

Apply gain to a specific channel.

```
adc.Gain(0, MCP3903::g8);  // 8x gain on CH0
```

phase(group, phase)

Apply phase correction between two grouped channels.

```
adc.phase('a', 10);  // Group CH0 & CH1, CH0 lags by 10 units
```

Data Rate Calculation (Example)

Assume:

- Crystal = 4 MHz
- Prescaler = 1
- OSR = 32

Calculated Clocks:

- **Analog Clock (AMCLK)** = 4 MHz / 1 = 4 MHz
- **Digital Clock (DMCLK)** = AMCLK / 4 = 1 MHz
- **Data Output Rate (fD)** = DMCLK / OSR = 31.25 kHz
