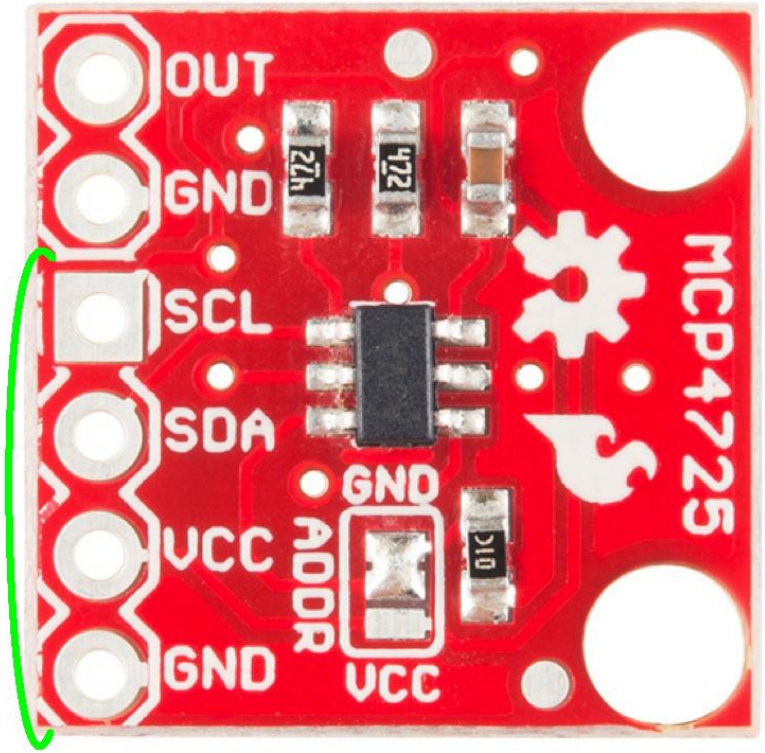
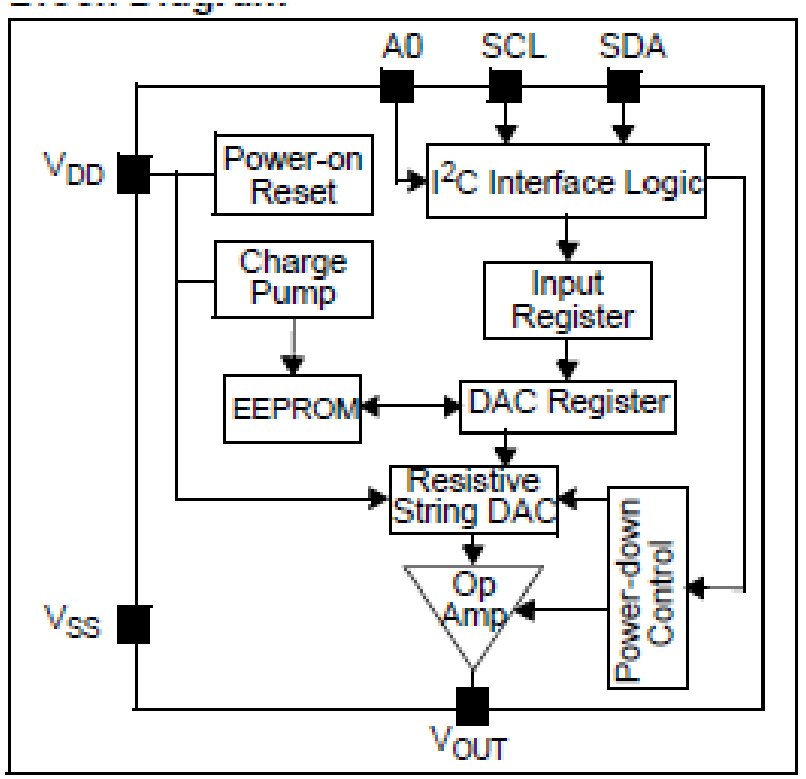
Digital Analog Converter

Dinesh Kumar

ISRO Satellite Center

Bangalore

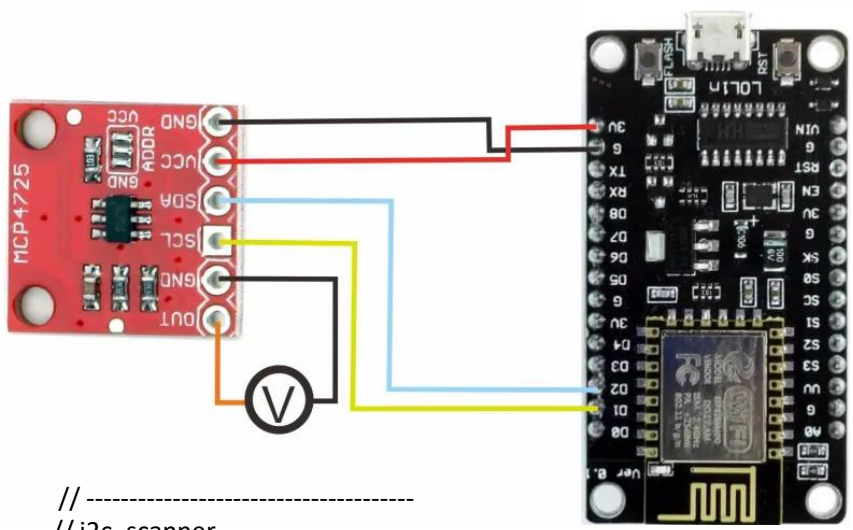




- 12-Bit Resolution
- On-Board Nonvolatile Memory (EEPROM)
- ±0.2 LSB DNL (typical)
- External A0 Address Pin
- Normal or Power-Down Mode
- Fast Settling Time: 6 µs (typical)
- External Voltage Reference (V_{DD})
- Rail-to-Rail Output
- Low Power Consumption
- Single-Supply Operation: 2.7V to 5.5V
- I^2C Interface:
  - Eight Available Addresses
  - Standard (100 kbps), Fast (400 kbps), and High-Speed (3.4 Mbps) Modes
- Small 6-Lead SOT-23 and DFN Package Options
- Extended Temperature Range: -40°C to +125°C

Applications

- Set Point or Offset Trimming
- Sensor Calibration
- Closed-Loop Servo Control
- Low Power Portable Instrumentation
- PC Peripherals
- Data Acquisition Systems




```
 // ----------------------------------------
// i2c_scanner
//
// Version 1
//    This program (or code that looks like it)
//    can be found in many places.
//    For example on the Arduino.cc forum.
//    The original author is not know.
// Version 2, Juni 2012, Using Arduino 1.0.1
//     Adapted to be as simple as possible by Arduino.cc user Krodal
// Version 3, Feb 26  2013
//    V3 by louarnold
// Version 4, March 3, 2013, Using Arduino 1.0.3
//    by Arduino.cc user Krodal.
//    Changes by louarnold removed.
//    Scanning addresses changed from 0...127 to 1...119,
//    according to the i2c scanner by Nick Gammon
//    http://www.gammon.com.au/forum/?id=10896
// Version 5, March 28, 2013
//    As version 4, but address scans now to 127.
//    A sensor seems to use address 120.
// Version 6, November 27, 2015.
//    Added waiting for the Leonardo serial communication.
//
//
// This sketch tests the standard 7-bit addresses
// Devices with higher bit address might not be seen properly.
//
```

Scan & find out Device address

```
#include <Wire.h>
void setup()
{
 Wire.begin();

 Serial.begin(9600);
 while (!Serial);            // Leonardo: wait for serial monitor
 Serial.println("\nI2C Scanner");
}
void loop()
{
 byte error, address;
 int nDevices;
Serial.println("Scanning...");
nDevices = 0;
 for(address = 1; address < 127; address++ )
 {
  // The i2c_scanner uses the return value of
  // the Write.endTransmisstion to see if
  // a device did acknowledge to the address.
  Wire.beginTransmission(address);
  error = Wire.endTransmission();

  if (error == 0)
  {
   Serial.print("I2C device found at address 0x");
   if (address<16)
    Serial.print("0");
   Serial.print(address,HEX);
   Serial.println("  !");

   nDevices++;
  }
  else if (error==4)
  {
   Serial.print("Unknown error at address 0x");
   if (address<16)
    Serial.print("0");
   Serial.println(address,HEX);
  }
 }
 if (nDevices == 0)
  Serial.println("No I2C devices found\n");
 else
  Serial.println("done\n");

 delay(500);          // wait 5 seconds for next scan
}
```

$$DAC = \frac{V \times 4096}{3.3}$$

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;

// Set this value to 9, 8, 7, 6 or 5 to adjust the resolution
#define DAC_RESOLUTION    (9);

void setup() {
  Serial.begin(9600);
  Serial.println("Hello!");
  dac.begin(0x60);
}

void loop() {
  // put your main code here, to run repeatedly:
  dac.setVoltage(1* 4096 / 3.3, false);
}
```

Generating sin wave using DAC

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#include <math.h>
// Create an MCP4725 object
Adafruit_MCP4725 dac;
// Define parameters for the sine wave
const float amplitude = 2047.5; // Half of the DAC's 12-bit range (0-4095)
const float frequency = 1.0;    // Frequency of the sine wave in Hz
const float samplingPeriod = 1.0 / frequency; // Sampling period in seconds
const int numSamples = 100;     // Number of samples per cycle

void setup() {
  // Initialize the DAC object
  dac.begin(0x62); // You might need to adjust the address based on your hardware setup

  // Initialize Serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Calculate and output the sine wave
  for (int i = 0; i < numSamples; i++) {
    float t = (float)i / numSamples;
    float angle = 2.0 * PI * frequency * t;
    int dacValue = amplitude * sin(angle) + amplitude;

    // Set the DAC output voltage
    dac.setVoltage(dacValue, false); // The second argument is "writeEEPROM," set to false for immediate output

    // Print the DAC value to the Serial Monitor for debugging (optional)
    Serial.println(dacValue);

    // Delay for a short time to control the sine wave frequency
    delayMicroseconds(samplingPeriod * 1000000);
  }
}
```

Generating saw tooth wave

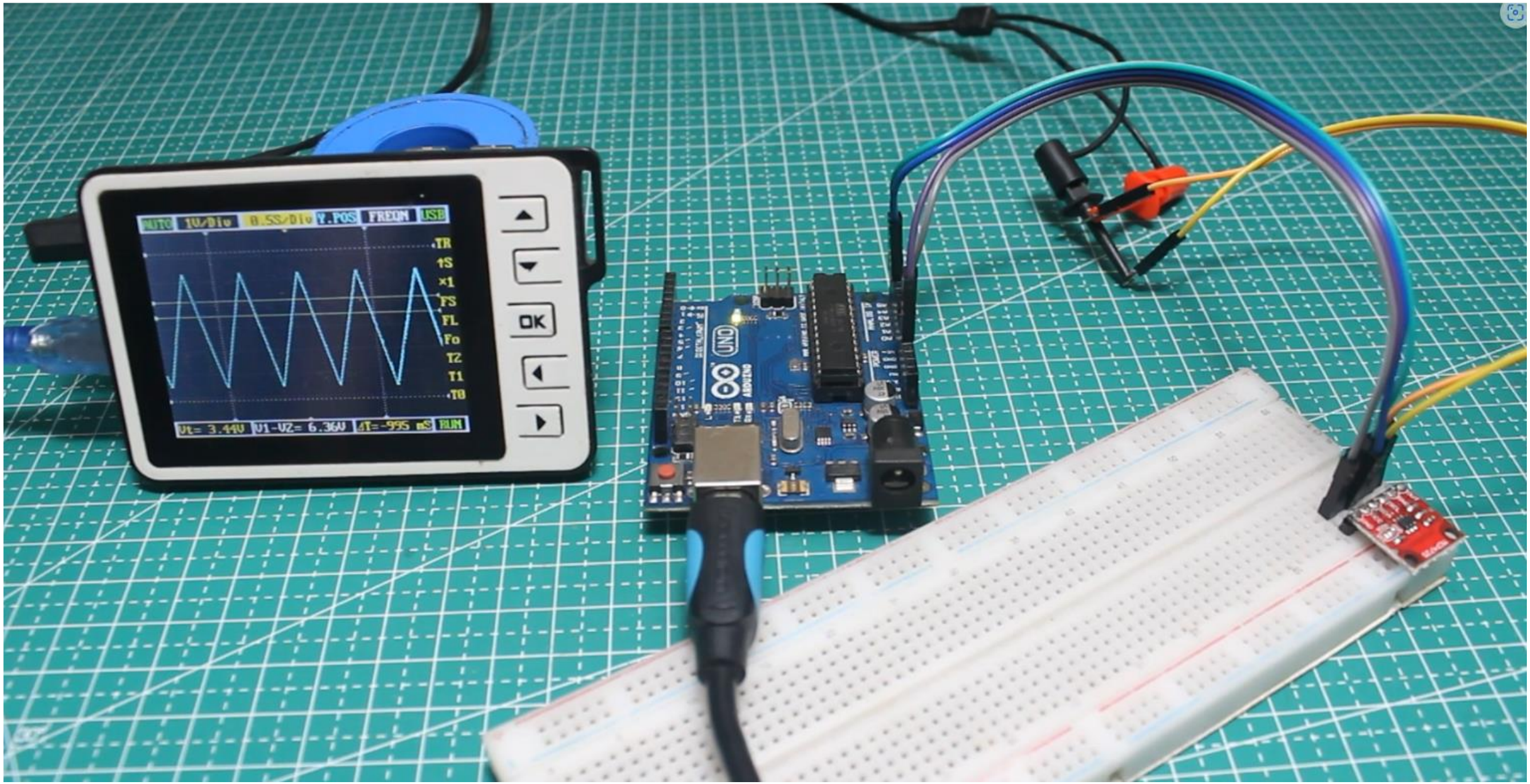
```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

// Create an MCP4725 object
Adafruit_MCP4725 dac;

// Define parameters for the sawtooth wave
const float amplitude = 4095.0; // Full range of the DAC (0-4095)
const float frequency = 1.0;   // Frequency of the sawtooth wave in Hz
const float samplingPeriod = 1.0 / frequency; // Sampling period in seconds
const int numSamples = 100;    // Number of samples per cycle

void setup() {
  // Initialize the DAC object
  dac.begin(0x60); // You might need to adjust the address based on your hardware setup

  // Initialize Serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Calculate and output the sawtooth wave
  for (int i = 0; i < numSamples; i++) {
    float t = (float)i / numSamples;
    int dacValue = (int)(amplitude * t);

    // Set the DAC output voltage
    dac.setVoltage(dacValue, false); // The second argument is "writeEEPROM," set to false for immediate output

    // Print the DAC value to the Serial Monitor for debugging (optional)
    Serial.println(dacValue);

    // Delay for a short time to control the sawtooth wave frequency
    delayMicroseconds(samplingPeriod * 1000000);
  }
}
```

Generating triangular wave

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

// Create an MCP4725 object
Adafruit_MCP4725 dac;

// Define parameters for the triangular wave
const float amplitude = 4095.0; // Full range of the DAC (0-4095)
const float frequency = 1.0;   // Frequency of the triangular wave in Hz
const float samplingPeriod = 1.0 / frequency; // Sampling period in seconds
const int numSamples = 100;    // Number of samples per cycle

void setup() {
  // Initialize the DAC object
  dac.begin(0x60); // You might need to adjust the address based on your hardware setup

  // Initialize Serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Calculate and output the triangular wave
  for (int i = 0; i < numSamples; i++) {
    float t = (float)i / numSamples;
    int dacValue;

    if (t < 0.5) {
      // Increasing phase
      dacValue = (int)(amplitude * t * 4.0); // Multiply by 4 to complete one cycle in half the time
    } else {
      // Decreasing phase
      dacValue = (int)(amplitude * (1.0 - (t - 0.5) * 4.0)); // Subtract from 1.0 to decrease linearly
    }

    // Set the DAC output voltage
    dac.setVoltage(dacValue, false); // The second argument is "writeEEPROM," set to false for immediate output

    // Print the DAC value to the Serial Monitor for debugging (optional)
    Serial.println(dacValue);

    // Delay for a short time to control the triangular wave frequency
    delayMicroseconds(samplingPeriod * 100000);
  }
}
```

Generating Square Wave

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

// Create an MCP4725 object
Adafruit_MCP4725 dac;

// Define parameters for the square wave
const int amplitude = 4095;  // Full range of the DAC (0-4095)
const float frequency = 1.0; // Frequency of the square wave in Hz
const float period = 1.0 / frequency; // Period of the square wave in seconds

void setup() {
  // Initialize the DAC object
  dac.begin(0x60); // You might need to adjust the address based on your hardware setup

  // Initialize Serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Set the DAC output voltage to the minimum value (0)
  dac.setVoltage(0, false); // The second argument is "writeEEPROM," set to false for immediate output

  // Wait for half of the period (low phase)
  delayMicroseconds(period * 500000);

  // Set the DAC output voltage to the maximum value (amplitude)
  dac.setVoltage(amplitude, false);

  // Wait for half of the period (high phase)
  delayMicroseconds(period * 500000);
}
```


AUTO
1V/Div
0.5S/Div
Y-POS
FREQM
USB
TR
1S
x1
FS
FL
Fo
T2
T1
T0
Vt= 3.44V
V1-V2= 6.36V
ΔT=-995 mS
RUN
OK
UNO
ARDUINO