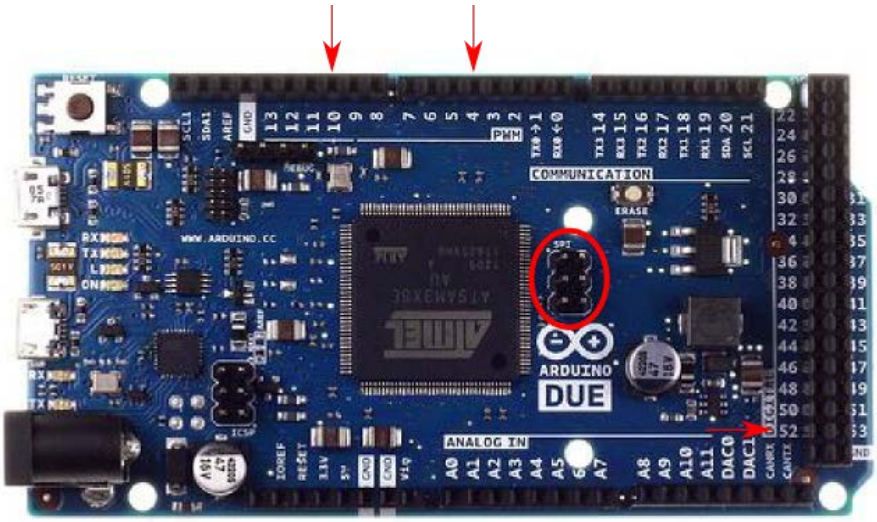
Arduino DUE + DAC MCP4922 (SPI)

v101

In this document it will described how to connect and let a Digital/Analog convert work with an Arduino DUE.

The big difference between and Arduino DUE and other Arduinos about SPI is that with the DUE is possible to address up to 3 different SPI devices at the same time, instead on the Arduino UNO for you can address just one SPI device at a time. The most of the libraries are using the pin 10 as CS pin but that is up to software developer. Some libraries allows the user also to specify the CS pin at the beginning.

On the DUE the pins (CS) with which it is possible to address SPI devices are: 10, 4, 52 (See picture below). The other SPI pins are available in middle of the board as shown in the following picture.

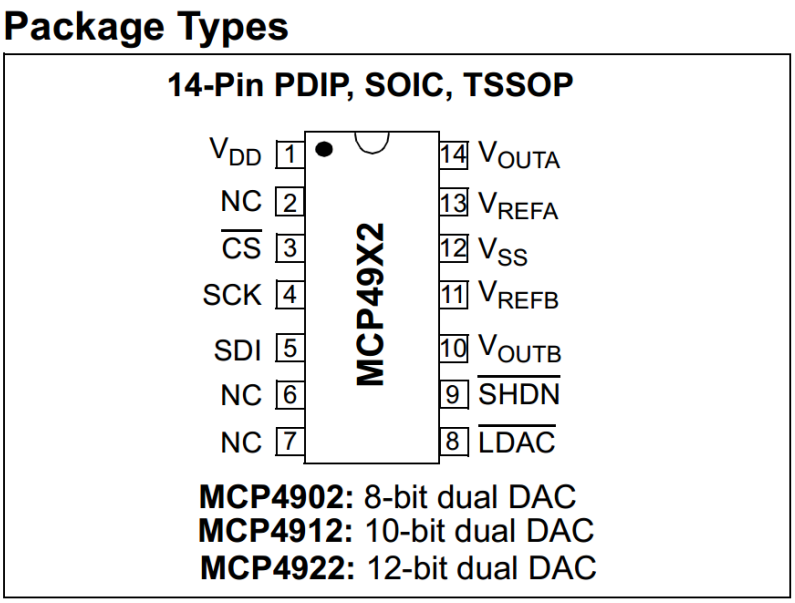


The following table display on which pins the SPI lines are broken out on the different Arduino boards:

Arduino Board	MOSI	MISO	SCK	SS (slave)	SS (master)
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	-
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	-
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-
Due	**ICSP-4**	**ICSP-1**	**ICSP-3**	**-**	**4, 10, 52**

Note that MISO, MOSI, and SCK are available in a consistent physical location on the ICSP header; this is useful, for example, in designing a shield that works on every board.

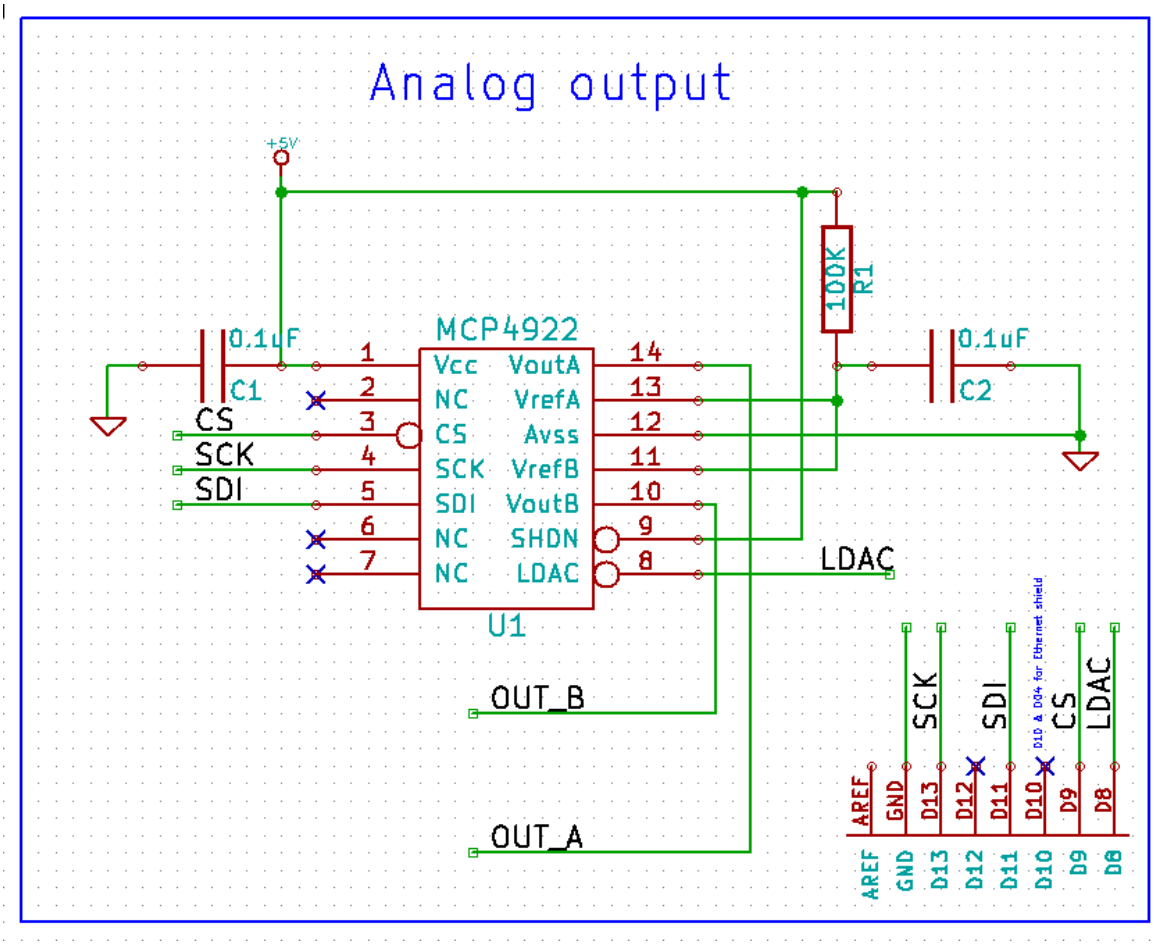
In the following picture is displayed a scheme of the pins of the DAC MCP4922.



So to let the DAC MCP4922 Digital Analog Converter work with the Arduino DUE the following connections can be made:

ARDUINO DUE	**MCP4922**
Pin 10	Pin 3: CS
Pin 7 (Optional): It is used only if you want to have the possibility to shut down the DAC	Pin 9: SHDN
SCK (from the SPI pins in the middle of the DUE board)	Pin 4: SCK
MOSI (from the SPI pins in the middle of the DUE board)	Pin 5: SDI (Serial Data In)

The other pins of the DAC are connected according to the following picture:



To test if the sketch that we will upload it is working we can use a digital multimeter. Of course this will work only if the frequency of the signal we want to generate is less than the maximum frequency we can observe with the multimeter. The multimeter will be connected with one end to the GND of the Arduino and the other one to the OUT_A of the previous picture.

A sketch test which can be uploaded to test our DAC + DUE connections is (you can copy it and paste directly in the Arduino IDE for a better readability):

```
/*
DAC _MCP4922 to DUE

This sketch allows you to interface the Arduino DUE with a DAC MCP4922.
The DAC needs a 16 bit word to work.
The first 4 bits are control bits and the remaining are bits which will contain
the value you want to send.
We have 12 bit available for our message so we will have a range of values between
0 and 4095.

  Connections
  ===================================================

  +5V              > 4922 pin 1
```

```
  DUE pin CS      > 4922 pin 3   (SS - slave select)
  DUE SCK         > 4922 pin 4   (SCK - clock)
  DUE MOSI        > 4922 pin 5   (MOSI - data out)
  Ground          > 4922 pin 8   (LDAC)
  +3.3V           > 4922 pin 11  (voltage ref DAC B)
  Ground          > 4922 pin 12
  +3.3v           > 4922 pin 13  (voltage ref DAC A)

  4922 pin 14 DAC A > 1k resistor > SignalOUT(you can put this signal in a
multimeter or oscilloscope to look at it)
  The other wire of the multimeter/oscilloscope should be at the same GND of the
Arduino

*/

#include <SPI.h> // Include the SPI Library
// I suggest to give a first look to the following link:
// http://arduino.cc/en/Reference/SPI

#define CS 52 // It can be 10,4, or 52 . Be careful if you use an Ethernet Shield

void setup() {
  // put your setup code here, to run once:
SPI.begin(CS);
SPI.setBitOrder(MSBFIRST);
pinMode(7,OUTPUT);
digitalWrite(7,HIGH);

/* Our DAC can operate at Mode 0,0 (which corresponds to mode 0 in the SPI
library) and in mode 1,1 (which corresponds to mode 3 in the SPI library) */

/* I don´t know why instead the DAC is working with the following instruction??
 Everything works also without this instruction. I don´t know why!
 it´s working just with modes 2 and 3. Maybe these mode 2 and 3 corresponds
 to the mode 0,0 and 1,1 to which the MCP4922 datasheet is referring.
 To understand what I´m talking about give a look to the following page:
 http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
 in the section "Mode Numbers".
*/
SPI.setDataMode(CS, 2);
/*
From the datasheet we understand that the maximum frequency at which the DAC
can operate is 20MHz. The Arduino DUE has a system clock of 84MHz so if we don´t
say
anything to the SPI library it won´t change the clock at which we will operate.
84 MHz is too big for the DAC, to change this we can use the function
SPI.setClockDivider(SS,divider)
This function set the clock, for the device on pin SS, to 84/divider. The divider
variable
can only be an integer. For this reason for example to have an operating clock for
our device of 1MHz we
```

```
will write: SPI.setClockDivider(SS,84);
to have an operating clock for our device of 2MHz we will write:
SPI.setClockDivider(SS,42); and so on.
By the way, seems that it works also without this instruction even if I have no
idea what happens internally
when you don´t specify the clock of the device. Maybe, I guess it goes up to the
maximum frequency of the DAC
but I´m not sure and for this reason I suggest to choose a frequency of operation
and fix it with the
instruction described before.
*/
SPI.setClockDivider(CS,21); // 4MHz
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(7,HIGH);
  // The entire message to send a 4095 number will be:
  // 0111 | 111111111111

  // The following is the value we want to send ;)
    int value=1230;
  // With V_REF of 3.3 V and VDD of 5V , 2470 corresponds to 4.00V
  //                                    1230 corresponds to 2.00V

  // Value parsing (the following 3 lines are needed just to convert an int
(32bit) in 2 bytes)
    if(value>=4095){
    value=4095;
    }

    byte msg2=(byte)value;
    byte msg1=(byte)(value>>8);
    msg1=msg1 | 0b01010000;
//  byte msg1=0b01010100; // 01111111
//  byte msg2=0b00000001; // 11111111

/* The DAC accepts only words of 16bit and here we are sending 2 bytes one after
another one. For more info about the SPI.transfer(…,…) function give a look to the
following page: http://arduino.cc/en/Reference/SPITransfer */
  SPI.transfer(CS,msg1,SPI_CONTINUE);
  SPI.transfer(CS,msg2,SPI_LAST);
  delay(4000);

// Uncomment the following 3 lines if you want to ShutDown every 4 seconds the DAC
//  digitalWrite(7,LOW);
//  delay(4000);11111010000
//  digitalWrite(7,HIGH);
}
```

Now what you will see on your multimeter depends from what is the V_{DD} and V_{REF} you chose for your circuit. As you can see from the datasheet, the V_{OUT} (the value you will observe on the multimeter) depends from the gain, and from the V_{REF}. In our case, a DAC MCP4922, $n = 12$.

EQUATION 4-1: ANALOG OUTPUT VOLTAGE (V_{OUT})

$$V_{OUT} = \frac{(V_{REF} \times D_n)}{2^n} G$$

Where:

V_{REF}	=	External voltage reference
D_n	=	DAC input code
G	=	Gain Selection
	=	2 for <$\overline{GA}$> bit = 0
	=	1 for <$\overline{GA}$> bit = 1
n	=	DAC Resolution
	=	8 for MCP4902
	=	10 for MCP4912
	=	12 for MCP4922

Moreover it depends also from the V_{DD} because the maximum value the V_{OUT} will be saturated at $V_{DD} - 0.04$. (Look at p. 4 of the Datasheet).

In the following picture is showed the picture, just to show you that I made really what I´m telling you: