

Lab 7 - DAC (SPI) - Libraries

In this lab we will learn about the built in SPI (Serial Peripheral Interface) library for MPIDE.

Equipment Needed

Fubarino SD
MCP4902 DAC
Oscilloscope
3 Oscilloscope Probes

Part 1 - SPI

The first program we are going to look at sets up SPI communications in the setup() and then continuously transmits a single character in the loop(). Prior to transmitting data on an SPI bus, the chip select line for the chip we are trying to communicate with must be brought low. This is simply done using the digitalWrite().

The SPI library can be imported into our project using either the "Sketch->Include Library...->SPI" menu selection or by adding the line "#include <SPI.h>" at the top of our sketch.

Run this code

```
#include <SPI.h> //use the SPI library

uint8_t chip_select = 7;
void setup() {
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV64);
  pinMode(chip_select, OUTPUT);
}
void loop() {
  digitalWrite(chip_select, LOW);
  SPI.transfer(0xA3);
  digitalWrite(chip_select, HIGH);
}
```

Using pin 7 and the SCK and SDO line, trace out the timing on the oscilloscope. Do NOT draw the wave forms on the provided worksheet.

Analysis

```
#include <SPI.h> //use the SPI library
Use the built in SPI library
```

```
uint8_t chip_select = 7;
```

A chip select is what tells a particular chip it is the one you want to talk to. The chip select for SPI is Active low so only the chip with a low input should be listening.

```
SPI.begin();
```

Set up the microcontroller to use SPI.

```
SPI.setClockDivider(SPI_CLOCK_DIV64);
```

This changes the rate at which SPI data is sent. In this case the rate is set pretty slow (80MHz / 64) = 1.25MHz.

```
SPI.transfer(0xA3);
```

Transfer the value 0xA3 out the SPI port

Check off

Connect channel 1 and 2 of the oscilloscope probes to the SPI SCK and SPI SDO lines coming from the Fubarino SD. Connect the chip select line from the Fubarino SD to the trigger input on the oscilloscope. Adjust the oscilloscope time base (take the scope out of cal on the time axis) so that you get one bit of data per graticule on the oscilloscope screen. Convert the binary data shown on the oscilloscope screen to hexadecimal.

Change the value being transmitted to 0x66 and re-upload your program. Convert the binary data shown on the oscilloscope screen to hexadecimal.

Does the data on the scope screen match the expected output?

Is data transmitted lsb to msb or msb to lsb?

In what ways does SPI communications differ UART communications?

Call the instructor over for check off when you get it working.

Part 2 - DAC Functions

In this lab we will create some utility functions that allow us to easily use both channels of the MCP4902. In the process we will be hiding some of the complexity of talking to the chip allowing higher level thinking about the device as we develop more complex programs.

This code also utilizes the LDAC input of the MCP4902. This input controls the dual buffering of the chip which allows us to shift data into both channels while not affecting the analog output. Once data is completely shifted into the dac, the LDAC line is asserted and data is transferred from the shift registers to the output registers.

Load this code onto your board note the functions and globals for controlling the DAC

```
#include <SPI.h> //use the SPI library

uint8_t dac_cs = 7;
uint8_t dac_load = 8;

void setup() {
    dac_init();
    SPI.begin();
}

void loop() {
    dac_xy(0xC3, 0x55);
}

/* DAC Functions */
void dac_init(void) {
    pinMode(dac_cs, OUTPUT);
    pinMode(dac_load, OUTPUT);
}

void dac_MCP49xx (uint8_t channel, uint8_t value) {
    uint8_t output_1, output_2; // represent 1st and 2nd bytes of 16 bits
    digitalWrite(dac_cs, LOW);
    if (channel == 1) {
        output_1=0x80;
    }
    else {
        output_1=0x00;
    }
    output_1 |= 0x70;
    output_2 = 0x00;
    output_1 = output_1 | (value >> 4);
    output_2 = value & 0x0f;
    output_2 = output_2 << 4;
    SPI.transfer(output_1);
    SPI.transfer(output_2);
    digitalWrite(dac_cs, HIGH);
}

void dac_xy(uint8_t x, uint8_t y)
{
    digitalWrite(dac_load, HIGH);
    dac_MCP49xx(1, x);
    dac_MCP49xx(0, y);
    digitalWrite(dac_load, LOW);
}
```

Using the `dac_load` line as the trigger, trace the following signals onto the top form of the Oscilloscope Worksheet provided `dac_load`, `dac_cs`, `SDO`

Label each trace and label the graph.

Analysis

```
dac_init()
```

Sets up outputs used with the MCP4902 DAC

```
dac_MCP49xx(unsigned char channel, unsigned char value)
```

This function takes the input data and formats it for the DAC then transfers it out the SPI port.

```
dac_xy(uint8_t x, uint8_t y)
```

This function calls `dac_MCP49xx` twice to set both channels of the DAC. The `dac_load` pin causes the DAC to change both values at the same time


Check off

Connect channel 1 and 2 of the oscilloscope to chip select and SPI SDO. Connect the oscilloscope trigger to the DAC-LOAD line. Adjust the waveform so that you can see all 32 bits being transferred out the SPI.

Call the instructor over for check off.

Part 3 - DAC Code to Separate file

In this section we are going to turn the DAC code into a library that we can utilize in additional projects. This is similar to what we have done in previous labs such as with the serial receive functions.

Open **your parser project** in a new window and on the gray bar in the IDE  Click the arrow and select "New Tab"

This opens  Name for new file OK Cancel

Type dac.h and hit OK

Now you should have a new blank file in your project selectable with the gray bar

Copy the DAC functions and global variables from part 2 this file (dac.h) and add

Include the dac.h file at the top of your main project code below the SPI.h include:

```
#include <SPI.h> //use the SPI library
#include "dac.h"
```

Verify that the dac portion of the program still functions the same way as before.

Check off

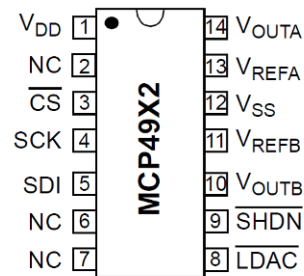
Call the instructor over for check off.

Part 4 - Wiring up the DAC

Power off the circuit and wire up the DAC.

Shown below is the pinout of the Microchip MCP4902 two channel 8-bit DAC. There is also a connection diagram showing how to connect the MCP4902 to the Fubarino with the pinouts specified in the code.

14-Pin PDIP, SOIC, TSSOP



MCP4902: 8-bit dual DAC
MCP4912: 10-bit dual DAC
MCP4922: 12-bit dual DAC

MCP4902 Pin Number	MCP4902 Pin Function	Connect to
1	V _{DD}	+3.3V
2	NC	NC
3	CS'	7
4	SCK	SCK 24
5	SDI	SDO 26
6	NC	NC
7	NC	NC
8	LDAC'	8
9	SHDN'	+3.3V
10	V _{OUTB}	Scope Ch 2
11	V _{REFB}	+3.3V
12	V _{SS}	GND
13	V _{REFA}	+3.3V
14	V _{OUTA}	Scope Ch 1

The MCP4902 is an 8-bit DAC can output put a voltage that is scaled between 0V and Vref, where Vref can be between 0V and Vdd. The chip Vdd can be between 2.7V and 5.5V. It is convenient for us to use 3.3V for Vdd since this is the voltage that our processor is running at.

Check Wiring

Call the instructor over for verification before powering the chip.

Add two global or static variables

```
uint8_t x = 0;  
uint8_t y = 0;
```

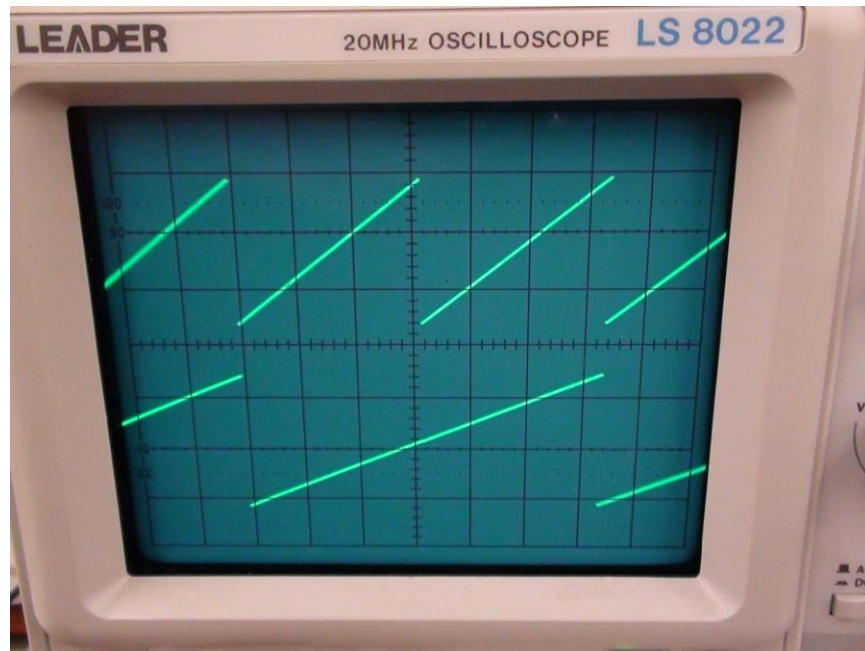
Remove this from your loop

```
dac_xy(0xC3, 0x55);
```

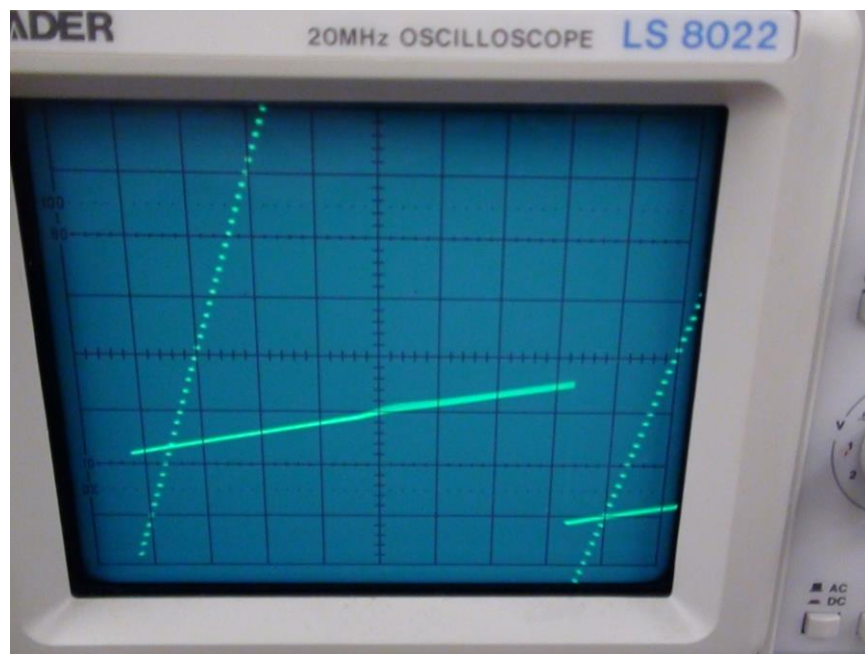
and make the loop function start like

```
void loop() {  
    dac_xy(x, y);  
    x++;  
    y+=2;  
}
```

Connect ch1 and ch2 to outputs A and B of the DAC and view the output on the scope. Your output should look something like this:



The voltage expanded lets you see the discrete steps of the DAC.



Label record the time per division and the volts per division for each channel.

Check off

Call the instructor over for check off. Be prepared to explain why one has twice the frequency of the other.


```

dac_xy(0xA4, 0x50);      dac_xy(0x50, 0x5C);      dac_xy(0x60, 0x50);      dac_xy(0x28, 0x44);
dac_xy(0xA8, 0x50);      dac_xy(0x1C, 0x58);      dac_xy(0x64, 0x50);      dac_xy(0x34, 0x44);
dac_xy(0xAC, 0x50);      dac_xy(0x20, 0x58);      dac_xy(0x68, 0x50);      dac_xy(0x4C, 0x44);
dac_xy(0x24, 0x60);      dac_xy(0x24, 0x58);      dac_xy(0x6C, 0x50);      dac_xy(0x50, 0x44);
dac_xy(0x30, 0x60);      dac_xy(0x28, 0x58);      dac_xy(0x2C, 0x4C);      dac_xy(0x24, 0x40);
dac_xy(0x34, 0x60);      dac_xy(0x2C, 0x58);      dac_xy(0x38, 0x4C);      dac_xy(0x30, 0x40);
dac_xy(0x38, 0x60);      dac_xy(0x38, 0x58);      dac_xy(0x58, 0x4C);      dac_xy(0x34, 0x40);
dac_xy(0x3C, 0x60);      dac_xy(0x54, 0x58);      dac_xy(0x1C, 0x48);      dac_xy(0x38, 0x40);
dac_xy(0x40, 0x60);      dac_xy(0x2C, 0x54);      dac_xy(0x20, 0x48);      dac_xy(0x3C, 0x40);
dac_xy(0x44, 0x60);      dac_xy(0x38, 0x54);      dac_xy(0x24, 0x48);      dac_xy(0x40, 0x40);
dac_xy(0x48, 0x60);      dac_xy(0x58, 0x54);      dac_xy(0x28, 0x48);      dac_xy(0x44, 0x40);
dac_xy(0x28, 0x5C);      dac_xy(0x30, 0x50);      dac_xy(0x2C, 0x48);      dac_xy(0x48, 0x40);
dac_xy(0x34, 0x5C);      dac_xy(0x3C, 0x50);      dac_xy(0x38, 0x48);      }
dac_xy(0x4C, 0x5C);      dac_xy(0x5C, 0x50);      dac_xy(0x54, 0x48);

```

Change the loop function so that it calls one of them when the PRG button (PIN_BTN1) is pressed and the other when it is not pressed.

Connect ch1 and ch2 to outputs A and B of the DAC and view the output on the scope in XY mode.

Analysis

The oscilloscope in XY mode uses the analog value of one input to control the X location of the dot and the other for the Y location.

Those two functions use the DAC to set analog values that move the dot to a series of locations that form a picture.

Check off

Call the instructor over for check off when you get it working.

Part 6 - Make own pattern

Use the worksheet or the excel sheet to generate a custom pattern and display it on the oscilloscope screen.

Here is an example of how to make a moving object. The parameters x and y are used to set the location of the object in the main loop:

```
void ball(uint8_t x, uint8_t y)
{
    dac_xy(0x04 + x, 0x10 + y);
    dac_xy(0x08 + x, 0x10 + y);
    dac_xy(0x0C + x, 0x10 + y);
    dac_xy(0x00 + x, 0x0C + y);
    dac_xy(0x10 + x, 0x0C + y);
    dac_xy(0x00 + x, 0x08 + y);
    dac_xy(0x10 + x, 0x08 + y);
    dac_xy(0x00 + x, 0x04 + y);
    dac_xy(0x10 + x, 0x04 + y);
    dac_xy(0x04 + x, 0x00 + y);
    dac_xy(0x08 + x, 0x00 + y);
    dac_xy(0x0C + x, 0x00 + y);
}
```

Check off

Call the instructor over for check off.

Part 7 - Parse integer from a string

Create a function that will convert a string to a uint8_t so that a command such as “dac 128” can be sent to the board to set the output voltage.

Hint: Converting a string like “321” to a number can easily be done by converting each character to its number by subtracting '0'. Then adding the first one to a variable that starts out at 0 (var =3) then multiplying that variable by 10 (var=30) and adding the next one (var=32) and so on until you get to the end of the number.

Check off

Call the instructor over for check off.

Part 8 - Parsing two integers from a string

Modify the command to take two parameters so that you can control which dac channel is being set. The new command would look like this “dac 0 128” to set DAC 0 to a value of 128 or “dac 1 0” to set DAC 1 to a value of zero.

Check off

Call the instructor over for check off.

Part 9 - Bitmap the Oscilloscope

EXTRA CREDIT

Create a function that uses an array to map the whole screen one bit for each dot.
Make at least 2 patterns and display them using your function.

```
void fillscreen(uint32_t inarray[2][64])
```

Check off

Call the instructor over to demonstrate when working.