Department of Electrical & Electronic Engineering

Imperial College London

2nd Year Laboratory

Experiment VERI: FPGA Design with Verilog (Part 3)

PART 3 – Analogue I/O and SPI serial Interface

1.0 The Add-on Card

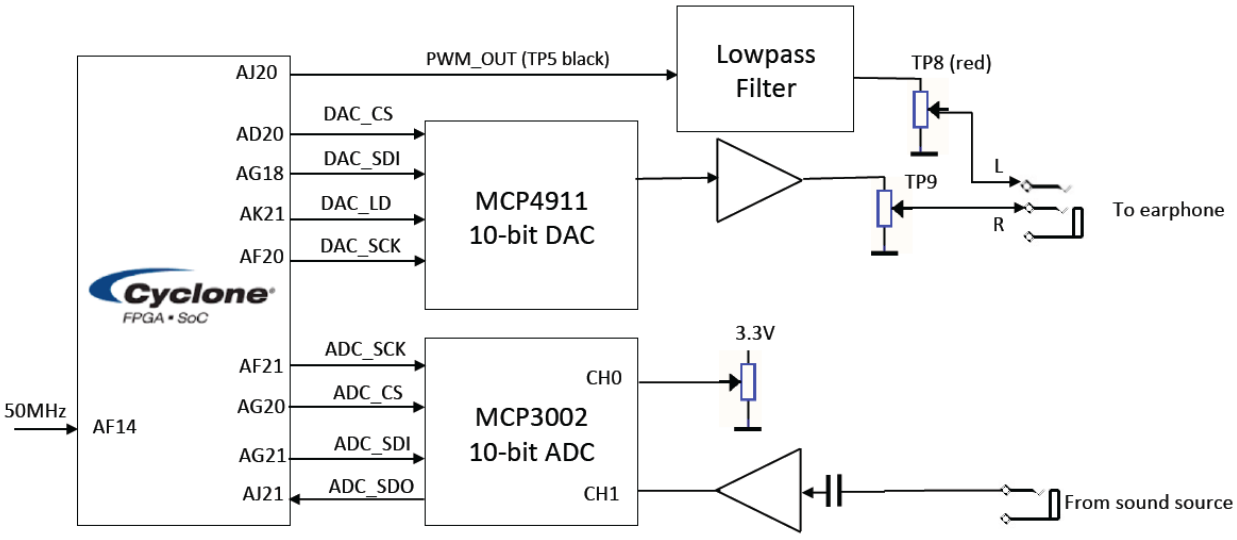
This part of the experiment introduces you to the add-on card to the DE1 board. The add-on card consists of a 10-bit ADC and a 10-bit DAC, a quad op-amp, sockets for earphone (analogue output) and sound source (analogue input), and a potentiometer. The overall block diagram of the add-on card is shown below. It should be plugged into the expansion socket furthest away from the edge of the DE1 board. Beware of the alignment between the plug and the socket. If the add-on board is inserted correctly, the green LED will light up when the DE1 board is turned ON.

You do not need to understand all the circuitry on this board in details. Nevertheless, the schematic diagram and a detail explanation on how this board works, together with all the datasheets of the components used, are provided on the Experiment webpage.

For this part of the experiment, you would need to bring your personal earphone, and from the Lab, get a 3.5mm lead and a digital voltmeter.

By the end of this part of the experiment, you will have:

- Understood and verified the operation of the **S**erial-to-**P**arallel **I**nterface (SPI) of the digital-to-analogue converter (DAC) MCP4911 using Modelsim;
- Tested the DAC and measured its output voltage range;
- Learned how to use a ROM (read-only memory) and a constant coefficient multiplier;
- Use the analogue-to-digital converter (ADC) MCP3002 to convert dc voltages;
- (Finally), designed a sinewave tone generator with variable frequency which is controlled with the slide switches, and the frequency value showed on the 7-segment displays in decimal format.



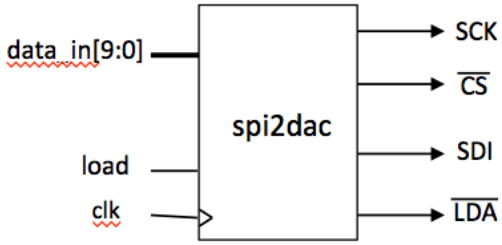
2.0 Experiment 10: Interface with the MCP4911 Digital-to-Analogue Converter

Step 1: Understanding Datasheet - Go to the Experiment website and download the datasheet for the MCP4911 DAC and the file **spi2dac.v**, which is a Verilog module that implements the SPI interface circuit to communicate with the DAC. Make sure that you understand from reading the datasheet:

- the purpose of each pin on the DAC (Section 3.0, page 17 of datasheet);
- how information is sent to the DAC through the serial data input (SDI) pin (Section 5.0, page 23-24);
- how to configure the DAC's internal function (page 25);
- DAC's timing specifications and timing diagram (pages 4 and 7).

There is no need for you to know how exactly the DAC works internally. However, you need to have sufficient appreciation of the serial interface in order to conduct this part of the experiment. Furthermore, don't worry if you don't fully understand the Verilog code in **spi2dac.v**. This will be explained in a Lecture.

Step 2: Timing diagram – The **spi2dac** module takes a 10-bit number in parallel (controlled through the load signal which must be high for at least 20ns) and generates the necessary serial signals to drive the MCP4911 DAC. Based on the information from the Datasheet, draw in your logbook the expected timing diagram of the SPI interface signals when a word 10'h23b is sent to the DAC.

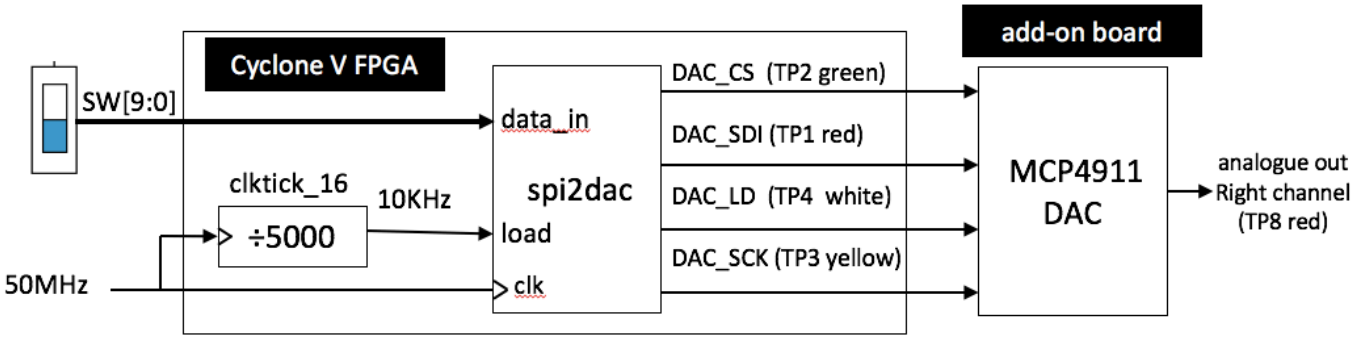


Step 3: Verify timing of spi2dac.v using Modelsim – The steps are:

1. Create a project ex10 and a top-level module ex10_top.v
2. Copy to the directory ex10 the file spi2dac.v downloaded from the webpage
3. Make this file top-level module (for now)
4. Click: … **> Process > Start > Analyze and Synthesise**
5. Start Modelsim (**Tools > Run Simulation Tools > RTL Simulation**)
6. Design a **do-file** as a testbench to exercise the input signals correctly
7. Run the do-file and match the waveform generated with your prediction

Step 4: Testing the DAC on DE1

In order to test the **spi2dac.v** module and verify that DAC works properly, create the top-level design **ex10_top.v** that implements the circuit shown in the following diagram.



The **data_in** value determined by the 10 switches (**SW[9:0]**) is loaded to the **spi2dac** module at a rate of 10k samples per second as governed by the **load** signal. The steps for this part are:

1. Download from the experiment website the file **spi2dac.v**.
2. Check that the **clktick_16.v** module that you used last week is in the **"mylib"** folder.
3. Create a top-level module **ex10_top**.v to connect all modules together as shown in the diagram.
4. Click: **Project > Add/Remove Files in Project** …, and select all the relevant files used here. This step is important – it allows you to select which modules to include in your design.
5. When **ex1__top**.v is the current file, click: **Project > Set as Top-Level Entity**. This is another useful step, which defines the top module, and all those module below this one, for compilation. With steps 4 and 5, you can move up or down the design hierarchy in a project for compilation.
6. Edit the **ex10_top.qsf** file to include **pin_assignment.txt**.
7. Compile and correct errors as necessary.

Once the design is compiled without error, download the bit-stream file to the DE1 board. Using the DVM feature of the scope, measure the DAC output voltage at TP8 for SW[9:0] = 0 and 10'h3ff. (The voltage range of the DAC output should be from 0V to 3.3V.)

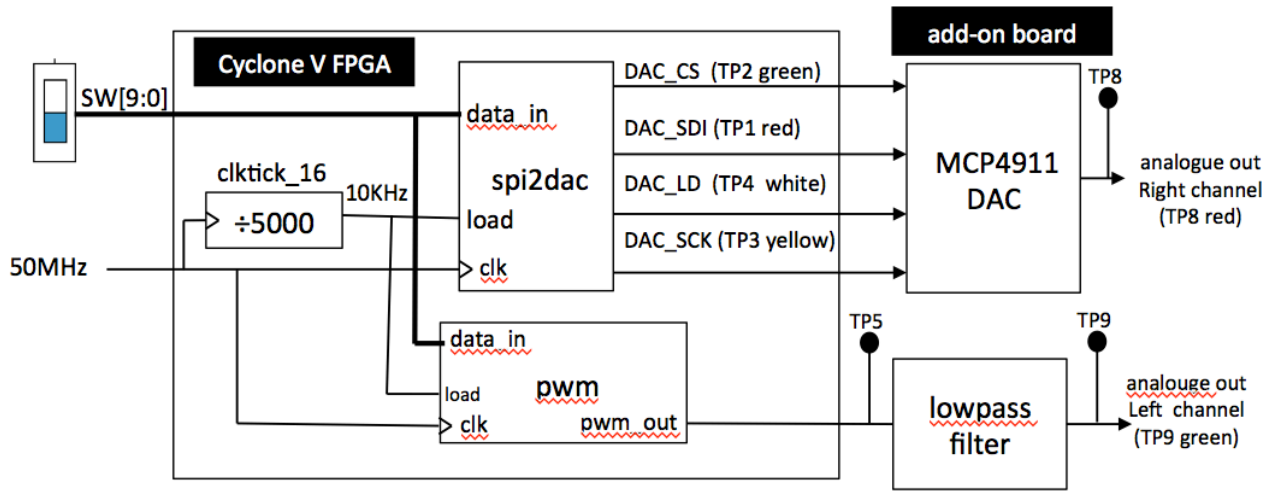
Step 5: Verify the signals on an oscilloscope

Confirm that the signals produced by the FPGA with the **spi2dac.v** module agree with those from Modelsim. Set SW[9:0] to 10'h23b and measure DAC_SCK (TP3) and DAC_SDI (TP1) using an oscilloscope. You may need to trigger the scope externally with the DAC_CS signal (TP2). Compare the waveforms to those predicted by Modelsim.

3.0 Experiment 11: D-to-A conversion using pulse-width modulation

Instead of using a DAC chip (and SPI serial interface to communicate with the chip), an alternative method to produce an analogue output from a digital number is to use pulse-width modulation (PWM). The Verilog code for a **pwm.v** module is given to you in Lecture 9 slide 15.

Create a design **ex11_top.v** according to the circuit shown below. Use the scope to examine the signals at TP5 and TP8. Compare the output voltage ranges at TP8 and TP9.



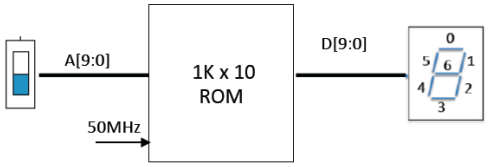
4.0 Experiment 12: Designing and testing a sinewave table in ROM

This part of the experiment leads you through the design of a 1K x 10 bit ROM, which stores a table of sine values suitable to drive our DAC. The relationship between the content of the ROM D[9:0] and its address A[9:0] is:

$$D[9:0] = \text{int}(511*\sin(A[9:0]*2*\pi/1024)+512) \qquad \text{for } 1023 \geq A[9:0] \geq 0$$

Since the DAC accepts an input range of 0 to 1023, we must add an offset of 512 in this equation. (This number representation is know as **off-set binary** code.)

Before generating the ROM in Quartus using the "**Memory Compiler**" tool, we need to first create a text file specifying the contents of the ROM. This can be done in different ways. Included on the Experiment webpage are: 1) a Python script to do this; 2) a Matlab script to do the same thing; 3) a memory initialization file **rom_data.mif** created by either method. Download these files and examine them.



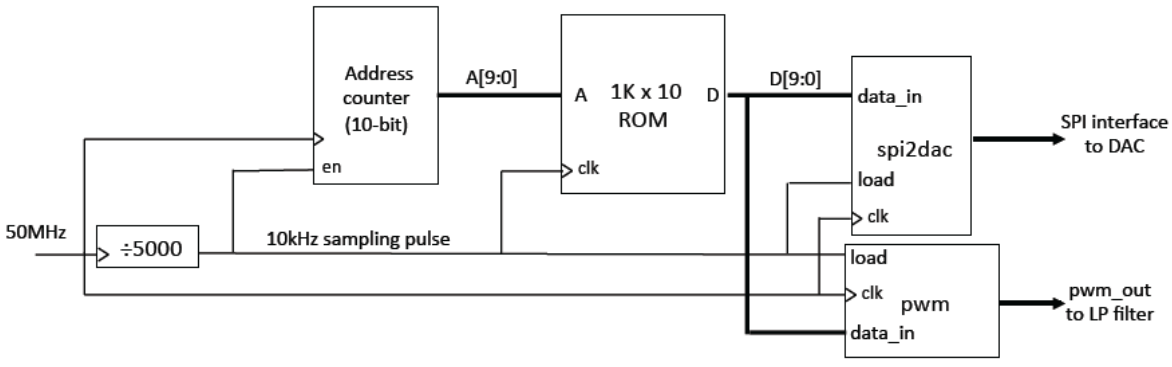

```
module ROM (
    address,
    clock,
    q);

    input   [9:0]  address;
    input    clock;
    output  [9:0]  q;
```

Click Tools > IP Catalog to bring up a tool which helps to create a 1-Port ROM. A catalog window will pop up. Select from the window **>Library >Basic Functions > Onchip Memory > ROM 1-Port.** Complete the on-screen form to create **ROM.v**.

To verify the ROM, create the design **ex12_top**.v, that uses the switches **SW[9:0]** to specify the address to the ROM, and display the contents stored at the specified location on the four 7-segment display. Once this is done and loaded onto the DE1, verify that the contents stored in the ROM matches those specified in the **rom_data.mif** file.

5.0 Experiment 13: A fixed frequency sinewave generator

Let us now replace the slide switches with a 10-bit binary counter and connect the ROM data output to **spi2dac** and **pwm** modules as shown in the figure below. Since the ROM contains one cycle of sinewave and the address to the ROM is incremented every cycle of the 10kHz clock, a perfect sinewave is produced at the left and right outputs of the 3.5mm jack socket.



Implement this circuit and verify that the signals produced by both the DAC and the PWM are as expected. What is the frequency of the sinewave?

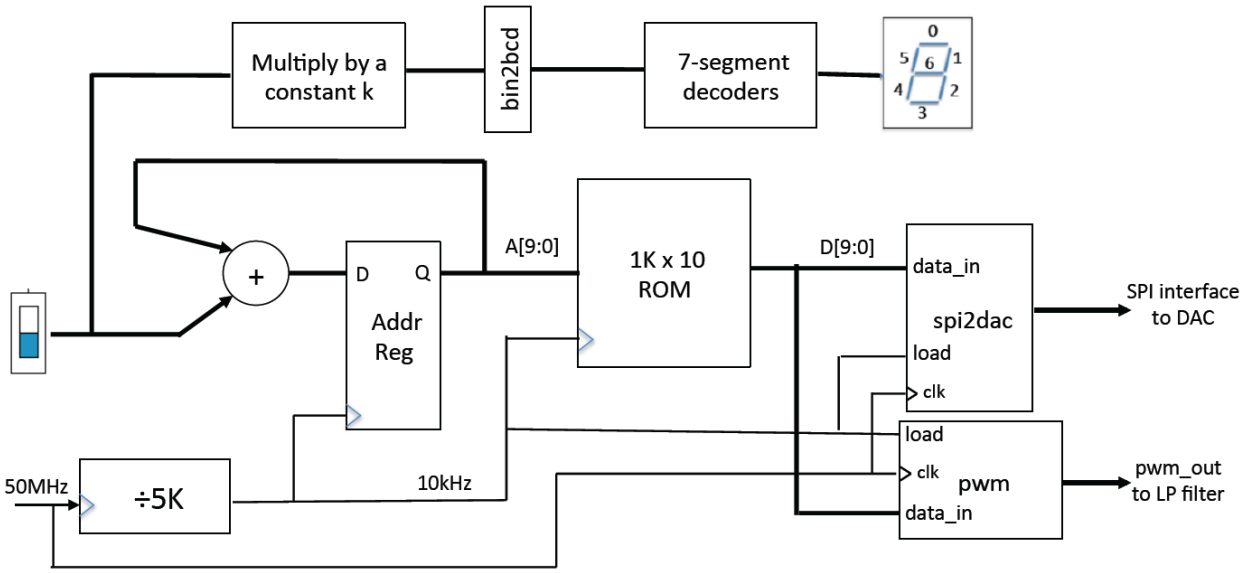
6.0 Experiment 14 (optional challenge): A variable sinewave generator

Combine everything together to produce a design **ex14_top.v**, which produces a variable frequency sinewave using table-lookup method. The sampling frequency is 10kHz, and the sine value is read from the ROM that is preloaded with one-cycle of a sinewave (i.e. the address of the ROM is the phase and the content is the sine value). On every sample period, advance the address (i.e. the phase) by an amount determined by SW[9:0]. Derive the relationship between the output signal frequency and the switch setting.

The overall block diagram is shown below. The switch setting is multiplied by a constant k to convert the phase increment SW[9:0] to frequency.

You can produce a 10-bit x 14-bit constant coefficient multiplier using the IP catalog tool. The 14-bit constant is 14'h2710 (which is 14'd10000). The product is a 24-bit number, and the frequency is the top 14-bits (Why?). The frequency can then be displayed on the 7-segment displays.

Produce a 439Hz sinewave, which is close to 440Hz, the frequency commonly found in tuning forks. Make sure that this is indeed correct (through listening or measuring with a frequency counter).



7. Experiment 15 (Optional Challenge): Using the A-to-D converter

In this experiment, you will learn to use A-to-D converter MCP3002 on the add-on board to convert analogue voltages to digital signals. Again, download from the webpage the **spi2adc.v** module, which is already written for you to use.

Instead of using the slide switches to control the frequency, use the A-to-D converter to convert the dc voltage of the potentiometer (which is between 0v and 3.3v) and use this converter value instead.

To help you know what you should aim for, the solutions for **ex14sol.sof** and **ex15sol.sof** are available to download.