Lab 8: State Machines and DAC Setting

UC Davis Physics 116B
Version: March 9, 2020

In this lab, you will be writing a module to set the dual channel, 12-bit DAC that has been inlcuded on the custom UCD breakout board, which is connected between the Alchitry-Au motherboard and the Alchitry-Io board that we have been using up until now. This allows both of these boards to be used simultaneously.

Required Files

You will need the following files, which can be downloaded from the Canvas site if they are not already on the computer.

- `FPGA Files/Alchitry-pins.txt`: pin definitions
- `FPGA Files/Achitry-UCD-pins.txt`: pin definitions for UCD breakout board
- `FPGA Files/Lab 8 Files/dac_set-template.v`: Template for your dac_set.v module
- `FPGA Files/Lab 8 Files/dac_set-testbench.v`: Test bench file for your dac_set.v module

Pre-lab Questions

You're using a 12-bit unipolar DAC with a 2.5V reference. What voltage changes should be associated with changing the most significant bit (MSB) and the least significant bit (LSB)?

Overview

The custom UCD Breakout Board contains a TI 7822 12-bit dual channel DAC. This DAC is controlled with 17 pins, which are defined in the `Achitry-UCD-pins.txt` file:

- **DB[11:0] (dac_data[11:0]):** 12-bit data bus. Can be used to read or write data to the DAC. In this lab, we will only be writing.
- **R/$\overline{\text{W}}$ (dac_rw):** read/write select. 0 for write, 1 for read. We will be tying it to 0.
- **$\overline{\text{A}}$/B (dac_ab):** A/B DAC select. 0 to write to DAC A, 1 to write to DAC B.

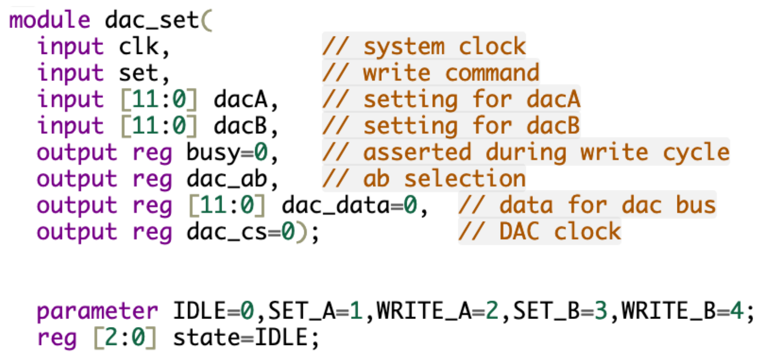
- **CS (dac_cs):** DAC chip select. When this transitions from 0 to 1, the data on DB[11:0] will be written to the DAC selected by $\overline{\text{A}}$/B.
- $\overline{\textbf{LDAC}}$ **(dac_ldac):** The DAC output will be set to the value loaded into the data bus when this pin goes to 0. We will be tying it permanently to 0.
- $\overline{\textbf{CLR}}$ **(dac_clr):** Asserting 0 on this pin will reset both DACs to 0. We will be tying it to 1 to prevent this from happening.

We have listed the manufacturer's name for the pin, with our corresponding signal name shown in parenthesis.

The DAC is unipolar with a 2.5V reference, so the output should be given by

$$\begin{aligned} V_{out} &= (2.5)\frac{data}{2^{12}} \\ &= (2.5)\frac{data}{4096} \end{aligned}$$

Lab Activities



```
module dac_set(
  input clk,              // system clock
  input set,              // write command
  input [11:0] dacA,      // setting for dacA
  input [11:0] dacB,      // setting for dacB
  output reg busy=0,      // asserted during write cycle
  output reg dac_ab,      // ab selection
  output reg [11:0] dac_data=0,  // data for dac bus
  output reg dac_cs=0);          // DAC clock


  parameter IDLE=0,SET_A=1,WRITE_A=2,SET_B=3,WRITE_B=4;
  reg [2:0] state=IDLE;
```

Figure 1: Header for dac_set.v module, including the setup for the state machine.

Write a module with the header shown in Figure 1, starting with the template that has been provided for you at the Canvas site. Since we'll be simulating this first, it's probably best to simply write it in EDA Playground to begin with.

The desired behavior is that in response to receiving a **set** signal, the code should generate the signals necessary to write the data in **dacA** and **dacB** to DAC A and B, respectively. It should also assert a **busy** bit will this action is in progress. To this end, it should generate the time line shown in Figure 2.

There are many ways to to this, but we will make this an exercise in using state machines. The state definitions are also shown in Figure 1, and their behavior is indicated in Figure 2.

We should be able to write to the DAC at the full clock speed of the FPGA, but that won't be the case with all DACs, so as an exercise, we will also declare a **state_clock** counter so that we will

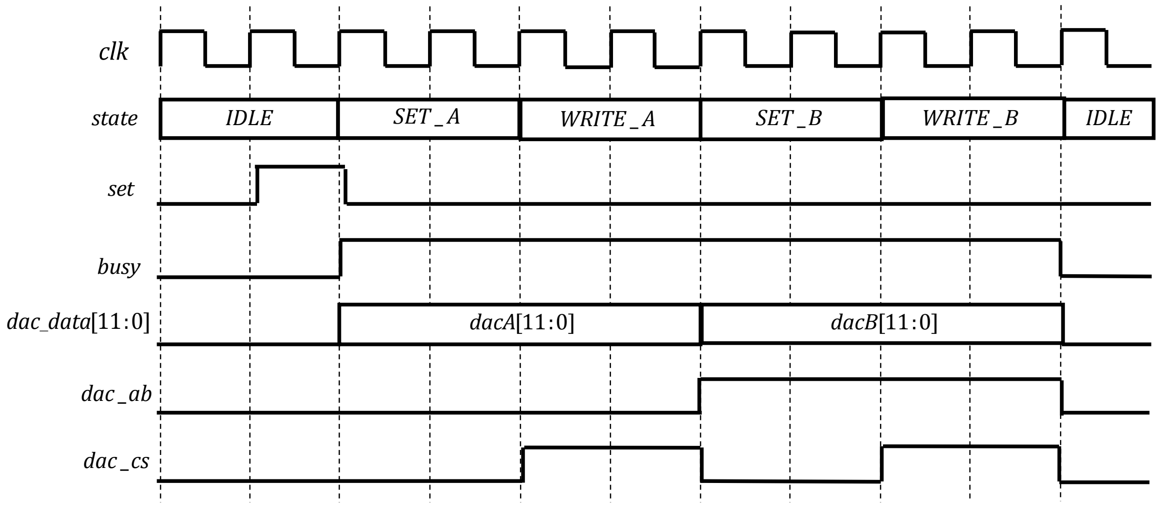


Figure 2: Time line for the dac_set module. In this case, the **HOLD** parameter has been set to 2. Also shown are the corresponding states.

remain in each state for a number of clock cycles defined by the parameter **HOLD**, which will initially be set to 2, as it is setup in the template.

The state behavior is defined as follows:

- **IDLE:** Set **dac_data**,**dac_cs**, and **dac_ab** to 0. If **set** is asserted, reset **state_clock** and set the next state to **SET_A**.
- **SET_A:** Set **dac_ab=0**, **dac_cs=0**, and **dac_data=dacA**. Increment **state_clock** and test if it's equal to **HOLD**. If it is, reset it and set the next state to **WRITE_A**.
- **WRITE_A:** Set **dac_cs=1**, to clock the data into DAC A. Increment **state_clock** and test if it's equal to **HOLD**. If it is, reset it and set the next state to **SET_B**.
- **SET_B:** Set **dac_ab=1**, **dac_cs=0**, and **dac_data=dacB**. Increment **state_clock** and test if it's equal to **HOLD**. If it is, reset it and set the next state to **WRITE_B**.
- **WRITE_B:** Set **dac_cs=1** to clock the data into DAC B. Increment **state_clock** and test if it's equal to **HOLD**. If it is, reset it and set the next state to **IDLE**.

The template includes a working example of the **IDLE** state, which you can use as a basis for defining the others.

Simulating the dac_set.v Module

After you have written your module in EDA Playground, simulate it using the test bench code provided. Verify that it generates the waveforms shown in Figure 3. You'll need to add, remove, and move traces to get exactly these to display. Once you get the module working, verify that setting **HOLD=4** will double the length of each state.

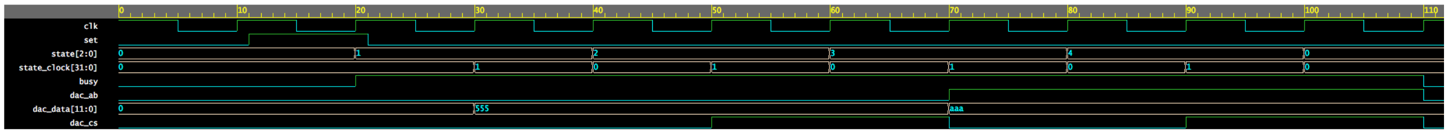


Figure 3: Time line simulation for the dac_set module.

Implementing and Testing Your Code

Transfer your ”dac_set” module to Alchitry labs and write an ”au_top” module to implement it on the board. In addition to the **clk** and **io_dip** inputs that you have used before, you will need to include outputs for all of the signals defined in the Overview section. The definitions for these pins can be found in the “Alchitry-UCD-pins.txt” file, and must be copied into your constraint file.

You should tie **dac_clr** to 1 to keep the chip from resetting, and both pins **dac_rw** and **dac_ldac** to 0, to indicate that we are writing data and want it to be promptly loaded to the DAC.

Instantiate your **dac_set** module. Tie **dac_data**, **dac_ab**, and **dac_cs** to the corresponding IO pins. Tie **dacA** to **io_dip[11:0]** and **dacB** to **io_dip[23:12]**. You can just tie the **set** input to 1, to cause it to continually write whatever is on the data bits. You can leave the **busy** output disconnected (connect it to an unconnected wire), since you’re not using it.

Finally, the lines on the breakout board ring quite a bit at high rates, so set **HOLD=10** to slow down the write cycle a bit.

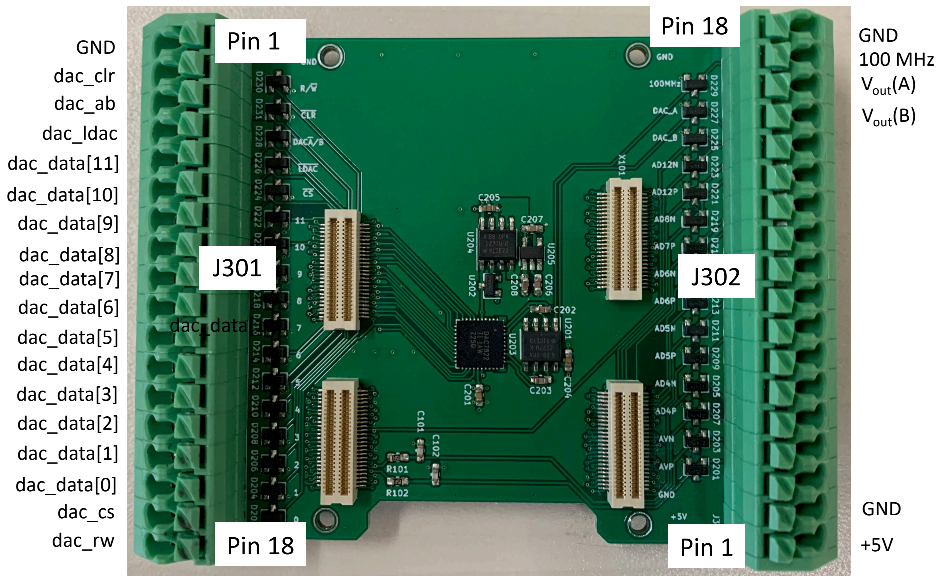


Figure 4: Pins on the UCD breakout board that are relevant to this lab. Note that the labels on the silk screen are NOT correct for J301.

Synthesize and download your code. You can test whether it’s working by using an oscilloscope with the breakout connectors. The pins associated with this lab are shown in Table 1, as well as Figure 4. These are spring-loaded connectors, which are released by pressing on the top. It’s

Table 1: Pinouts for the connectors on the UCD breakout board. Only the ones of relevance to this lab are listed.

pin	**J301**	**J302**
1	GND	+5V
2	dac_clr	GND
3	dac_ab	-
4	dac_ldac	-
5	dac_data[11]	-
6	dac_data[10]	-
7	dac_data[9]	-
8	dac_data[8]	-
9	dac_data[7]	-
10	dac_data[6]	-
11	dac_data[5]	-
12	dac_data[4]	-
13	dac_data[3]	-
14	dac_data[2]	-
15	dac_data[1]	$V_{out}(B)$
16	dac_data[0]	$V_{out}(A)$
17	dac_cs	100MHz
18	dac_rw	GND

usually easier to insert jumper wires and connecting the scope to those, rather than trying to use the scope directly in the connector. You can ground the probes to any of the indicated **GND** pins. One hint is to trigger on the **dac_ab** pin, which will transition high in the middle of each write cycle. Use the scope to verify that the **dac_cs** pin is behaving as expected.

Verify that you can set the voltages of DAC A and DAC B by setting **io_dip[11:0]** and **io_dip[23:12]**, respectively. Measure the voltage changes associated with the LSB and MSB, and verify that they match your predictions. Also verify that changing the setting of one DAC does not affect the other.

Lab Report

You report should include the answers to the prelab questions, the final version of your **dac_set** module and **au_top** module, a screen shot of you simulation results, and a scope trace showing that **dac_ab** and **dac_cs** are behaving as expected.