Lecture 12
Serial-Peripheral Interface (SPI)

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London
URL: www.ee.imperial.ac.uk/pcheung/teaching/ee2_digital/
E-mail: p.cheung@imperial.ac.uk

Lecture Objectives

- Learn about the DAC used in the Analogue I/O card
- Serial Peripheral Interface used by the DAC – how it works?
- Explore in details the Verilog design of the SPI interface module
- Examine the ADC used in the Analogue I/O card
I also provide a purpose-built ADC/DAC board to support the lab experiment. This analogue I/O board is only needed for Part 3 and 4 of VERI. However, I will now be examining the digital serial interface for these converter chips.

This shows the block diagram of the analogue I/O card used in the VERI experiment. It consists of a DAC (MCP4911) and an ADC (MCP3002), both using Serial Peripheral Interface (SPI). The DAC output is buffered by a unity gain opamp connected to the right channel of a stereo jack socket. The ADC has two input channels, one from a potentiometer providing a dc voltage (CH0) and another from the 3.5mm jack socket (CH1).

Finally, there is a 2nd order low-pass active filter, the input of which is driven directly from a digital output pin of the Cyclone FPGA. This is intended to provide filtering of a pulse-width modulated DAC output from the FPGA.
The DAC used with the I/O card is 10-bit, and it uses the Serial Peripheral interface. Its functional block diagram is shown here. The SPI interface has four signals, which should be drive by either the microcontroller or the FPGA. The DAC itself uses a resistor string architecture (i.e. just a bunch of 1024 series resistors of identical values). It has a selectable gain of 1X or 2X.

To send a value to the DAC to output (i.e. produce the analogue output Vout), a 16-bit value is sent to the DAC chip in a serial manner. The Chip Select (SC) signal going low indicate that this is the start of the data. This establishes the beginning of the data frame. First data bit (bit 15) is always 0. Bit 14 determines whether the reference voltage (Vref) is buffered or not buffered (via an internal opamp). For our design, Vref is around 3.3V.

Bit 13 determines the gain of the DAC (x1 or x2). Bit 12 is set to 1 if you are using the DAC, and set to 0 if you want to shutdown the device to conserve power.

Bit 11 to 2 contains the 10-bit data D(9:0) to convert into analogue voltage Vout, MSB first. Bit 1 and 0 are don’t cares.

The LDAC (low active) signal can be connected to ground or used a low active strobe signal to transfer the data to the DAC register (i.e. tell the DAC to update Vout). If LDAC is low, DAC update happens on rising edge of CS_bar.
Interfacing the FPGA to the DAC and ADC

- Overview of the DAC/ADC
- DAC is DC coupled (no capacitor in signal path)
- ADC is AC coupled (why?)
- Interface circuit to DAC: spi2dac.v
- Interface circuit to ADC: spi2adc.v

Important points to note:
- DAC and ADC function are NOT done within Cyclone V FPGA
- Conversion from/to analogue signals are done with 2 8-pin chips on Add-on card
- Why do we need serial-parallel interface circuits? To fit everything within 8-pin package
- A single serial clock is used for both ADC and DAC – set at 1MHz
- This is different from the system clock of 50MHz (fixed within DE1)
- Chip-select is low only when sending serial data to DAC chip on SDI pin
- LDA is low only when all 10-bit data sent and DAC to be loaded with new value

This is a simplified diagram showing how the Cyclone V FPGA is interfaced to the two data converters. There are two ADC channels and in our experiment, we are mostly using channel 1 via the 3.5mm jack socket. You will be supplying speech signals from the desktop computer.

There is one DAC which drives both the small speaker and, much better, drives the ear-phone. (Please bring the ear-phone to the lab.)

The interface between the FPGA chip and the converters is through the SPI bus. You are given the Verilog design for these two interface modules: spi2dac.v and spi2adc.v. In the rest of this lecture, I will be going through the design of the spi2dac module.

spi2dac design overview

- The components inside spi2dac are:
  1. Clock divider
  2. Load detector to detect load pulse
  3. FSM to control the spi interface
  4. Parallel to serial shift register to shift OUT the command and data to the DAC
  5. Various gates e.g. inverters and AND gates

- Note that the Verilog code is designed to match the block diagram shown here
- It consists of TWO state machines, a counter and a shift register

In order to use the DAC, you must include the interface module "spi2dac" in your design. This module has a schematic shown above. It takes two inputs (in addition to the 50MHz clock signal): data[9:0] is the 10-bit digital data to be converted by the DAC, and a load signal which is a high pulse to trigger the spi2dac module to send the 10-bit data to the DAC.

The internal working of spi2dac can be divided into 4 main modules. The divide-by-50 module is straightforward – it produces a 1MHz clock for the finite state machine, and is gated through the AND gate to generate the serial clock signal (at 1MHz).

The load detector module handles the load command and produces control signals to the SPI state machine and the shift register.

The shift register sends the control bits and the 10-bit data serially to the SDI output.

The spi controller FSM is the main control module designed as a state machine.

We will now consider each sub-module individually.
This is a straight forward clock divider. The Terminal Count (TC) is set to 24. Divide by 50 is done by toggling the output (clk_1MHz) after 25 clock cycles. Note that I generally prefer to use a down-counter instead of an up-counter. The counter (ctr) is set to 24, it then counts to zero. Output is toggled and the counter (ctr) is reset to the initial value of 24 again.

We have TWO signals to detect: the load pulse and the dac_cs signal.

Starting in the IDLE state, when load signal is asserted, we start the DAC cycle by entering the WAIT_CSb_FALL state. In this state, dac_start is asserted, and we wait for DAC_CS to go low from the SPI controller circuit. In this condition, the DAC is in the middle of accepting a new data for conversion. We go to state WAIT_CSb_HIGH TO wait for the conversion to be completed, which is indicated by DAC_CS going high. When that happens, we return to the IDLE state waiting for another 10-bit data to be loaded.
The controlling FSM controller is actually simpler than it first appears.

We need a FSM to have 18 states. State 0 is the idle state, waiting for a new data to be sent to the DAC. Here DAC_CS (which is low active) is ‘1’ and we wait for the dac_start to be asserted.

The default value of dac_cs and dac_id are specified first. By default we always go to the next state, i.e. state value goes up by 1.

Once the state machine moves to state 1, it just go through to state 16, which corresponds to cycle 0 to 15 in the timing diagram here. At the end of state 16, we de-assert dac_cs (i.e. go high), assert dac_id (low) and go back to the IDLE state.

Finally, the data and clock output is specified here. SDI is driven through a parallel in, serial out shift register.

We use a number of useful tricks here:
1. cmd is a 4-bit value defining the first four bits of the SDI data values. We use symbolic variable names to make the code easy to read.
2. Shift_reg <= (cmd, data_in, 2'b00) - parallel load the 16-bit value into the shift register.
3. Shift_reg <= (shift_reg[14:0], 1'b0) - perform left shift

The SDI is taken from the MSB of the shift register. The serial clock is dac_cs (low active) ANDed with the inverter version of the clock (making the rising edge of the SCK signal in the middle of the data bit).
Part 3 (ex10 & 11) - Testing DAC, SPI and PWM

- Use the 10 slider switches to set data value to converter to analogue voltage
- Continuously loading the switch value to DAC at 10KHz rate
- You need: clk,tick_16, pwm and spi2dac

In the Lab experiment, you will test the spi2dac.v module both with the simulator and on the hardware (with a scope) by inspecting the output signals on the test pins (located at the top of the I/O board).

Ex 10 and 11 are simple, but will give you confidence that the interface module works.

module pwm (clk, data_in, load, pwm_out);
    input clk;
    input [15:0] data_in;
    input load;
    output pwm_out;

    reg [15:0] cr; // counter register
    reg [15:0] count; // internal 16-bit count
    reg pwm_out;

always @(posedge clk)
    if (load == 1'b1) do <= data_in;

initial count <= 1'b0;
always @(posedge clk) begin
    count <= count + 1'b1;
    if (count >= 1'b1):
        pwm_out <= 1'b0;
    else
        pwm_out <= 1'b1;
end

endmodule

ex 11 - PWM DAC in Verilog (Lecture 9, slide 15)

Here is the same slide as that found in Lecture 9. Just a reminder for you on how the PWM module works. It is very simple, but very effective. You should compare the DAC output and PWM output, and see that the two methods are equally effective in producing an analogue voltage.