Reason for the new version of spi2dac and spi2adc

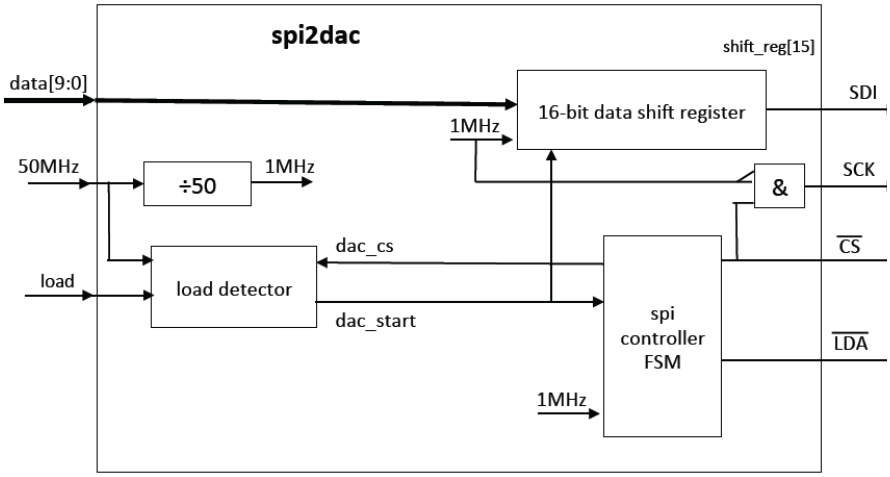
Peter Cheung, 8 December 2016

Some students discovered that their Verilog design using the spi2dac.v module sometimes give errors in the DAC output. For example, in one case, the sine wave generated is correct most of the time, but occasionally would have a pulse going to 0 (for one cycle) and then recovers afterwards. However, I found out that most students do not have any problem, and their design works perfectly for all parts of VERI. This apparent "random" error is due to a poor practice on my part in designing the spi2dac.v and spi2adc.v modules. These have now been replaced by a **new version** (version 3.0), which you can download from the experiment webpage.

This document explains the problem of my previous version, and how I have fixed the problem with this new version (version 3). YOU DO NOT NEED TO READ or understand this document in order to get your design working. I provide this explanation for those who wish to know more. Those who don't want to know the detail, you can just download the new version from the experiment webpage and used it by adding the new files (i.e. spi2dac_v3.v and spi2adc_v3.v) into your project, replacing the original version.

What did I do that's wrong?

The original design of spi2dac (as explained in Lecture 12 slide 8 onwards) contains THREE state machines: 1) the **÷50 module** to generate an internal 1MHz symmetrical clock signal; 2) the **load detector FSM** to detect the rising edge of the load signal; 3) the **spi controller FSM** which controls the generation of the serial data **SDI** and the serial clock **SCK**. It also has a shift register to shift data out.



Both the **clock divider FSM** and **the load detector FSM** are triggered by the 50MHz clock. However, I used the derived 1MHz clock signal to trigger the **spi controller** and to clock the 16-bit parallel to **serial shift register**. These are where the problem may lie.

Consider the block diagram of spi2dac.v above. The **dac_start** signal may change at the same time (or close to) the rising edge of the 1MHz clock signal. This could then cause a setup or hold time violation in the **spi controller** FSM circuit or the shift register.

Why does this only happen occasionally and not all the time? The Quartus software placement and routing algorithms use non-deterministic optimisation methods (e.g. simulated annealing). As a result, each time you run the Quartus software, the system may create a different physical design for the FPGA. The logic would be the

same; the actual locations of different ALMs and registers may be different. This could result in different propagation delays for the signal paths.

How to fix this?

The fix is very simple. I should have followed my own rule – the entire design should be **synchronous** to the 50MHz system clock (CLOCK_50). Therefore I did three things in the new design: 1) the **÷50 module** now produce two outputs, **clk_1MHz** which is the symmetrical 1MHz clock gated to produce **DAC_SCK** signal, and **tick** which is a 1MHz pulse lasting for only a single cycle of the 50MHz clock (and will be used as an enable signal in other state machines); 2) I make sure that the **spi controller FSM** is synchronized to the 50MHz, but state transition only occurs when **tick** is high; 3) the shift register only shift the data once every 1MHz clock cycle (using the signal **tick** again).

Shown here is the Verilog of the new spi2dac (file is: spi2dac_v3.v). I have highlighted the changes/additions to the old version.

..... Verilog code same as before...

```
   // --- internal 1MHz symmetical clock generator -----
   reg         clk_1MHz;   // 1Mhz clock derived from 50MHz
   reg [4:0]   ctr;        // internal counter
   reg         tick;       // 1MHz clock tick (1 cycle of 20ns every 1 usec)

   parameter   TC = 5'd24;  // Terminal count - change this for diff clk freq
   initial begin
      clk_1MHz = 0;        // don't need to reset - don't care if it is 1 or 0 to start
      ctr = 5'b0;          //  ... Initialise when FPGA is configured
      tick = 1'b0;
   end

   always @ (posedge sysclk)
     if (ctr==0) begin
        ctr <= TC;
        if (clk_1MHz==1'b0)
            tick <= 1'b1;
        clk_1MHz <= ~clk_1MHz; // toggle the output clock for squarewave
      end
     else begin
        ctr <= ctr - 1'b1;
        tick <= 1'b0;
      end
   // ---- end internal 1MHz symmetical clock generator ----------
   // .... with 17 states (idle, and S1-S16
   // .... for the 16 cycles each sending 1-bit to dac)
   reg [4:0]   state;

   initial  begin
      state = 5'b0; dac_ld = 1'b0; dac_cs = 1'b1;
      end

   always @(posedge sysclk)  // FSM state transition
      if (tick==1'b1)
      case (state)
         5'd0: if (dac_start == 1'b1)     // waiting to start
                  state <= state + 1'b1;
               else
                  state <= 5'b0;
         5'd17:   state <= 5'd0;  // go back to idle state
         default: state <= state + 1'b1;  // default go to next state
      endcase

   always @ (*)   begin       // FSM output
      dac_cs = 1'b0;  dac_ld = 1'b1;
      case (state)
         5'd0:    dac_cs = 1'b1;
         5'd17:   begin dac_cs = 1'b1; dac_ld = 1'b0; end
         default: begin dac_cs = 1'b0; dac_ld = 1'b1; end
         endcase
      end //always
   // --------- END of spi controller FSM

   // shift register for output data
   reg [15:0] shift_reg;
   initial  begin
      shift_reg = 16'b0;
      end

   always @(posedge sysclk)
      if((dac_start==1'b1)&&(dac_cs==1'b1)&&(tick==1'b1))       // parall
         shift_reg <= {cmd,data_in,2'b00};
      else if (tick==1'b1)                                     // . . els
         shift_reg <= {shift_reg[14:0],1'b0};

   // Assign outputs to drive SPI interface to DAC
         assign dac_sck = !clk_1MHz&!dac_cs;
         assign dac_sdi = shift_reg[15];
endmodule
```