Imperial College
London

Lecture 14
(Supplementary)

spi2dac.v Explained

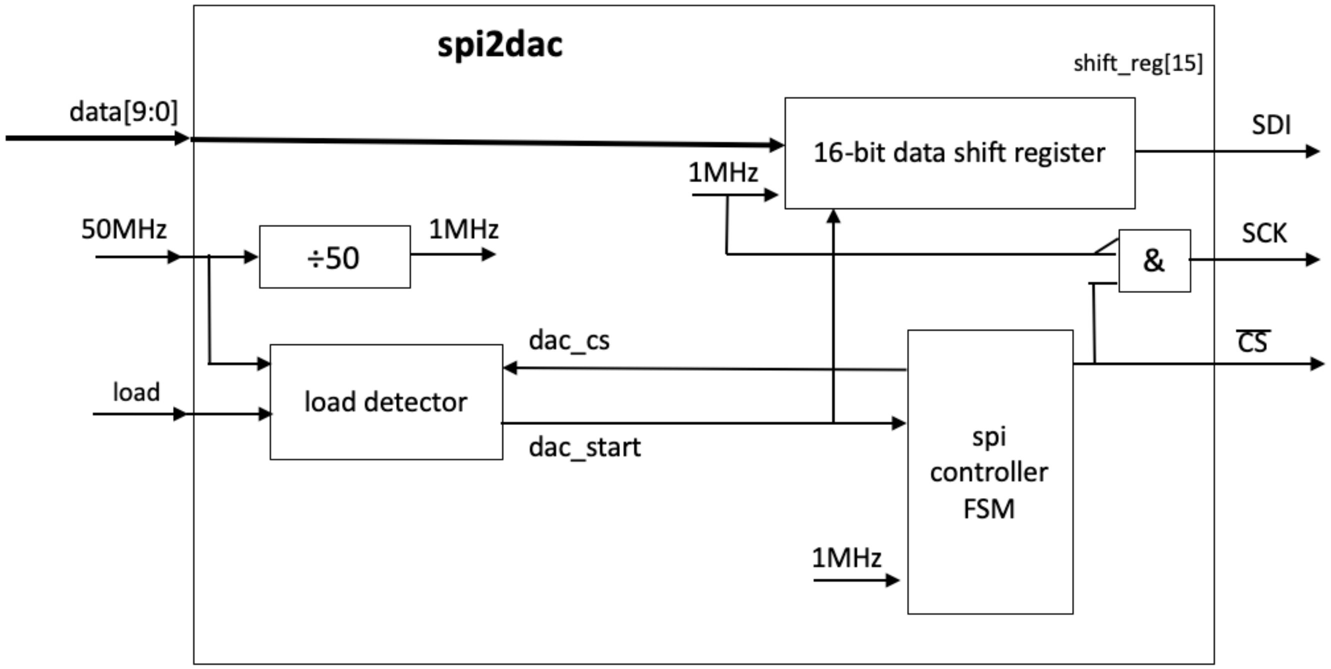
Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/E2_CAS/
E-mail: p.cheung@imperial.ac.uk

Spi2dac.v design overview

- The components inside spi2dac are:

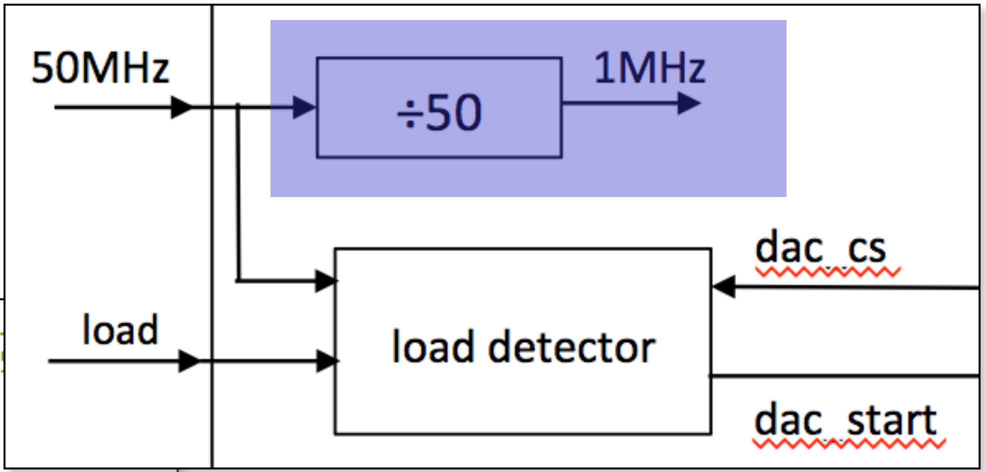
1. Clock divider
2. Load detector to detect load pulse
3. FSM to control the spi interface
4. Parallel to serial shift register to shift OUT the command and data to the DAC
5. Various gates e.g. inverters and AND gates



- 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

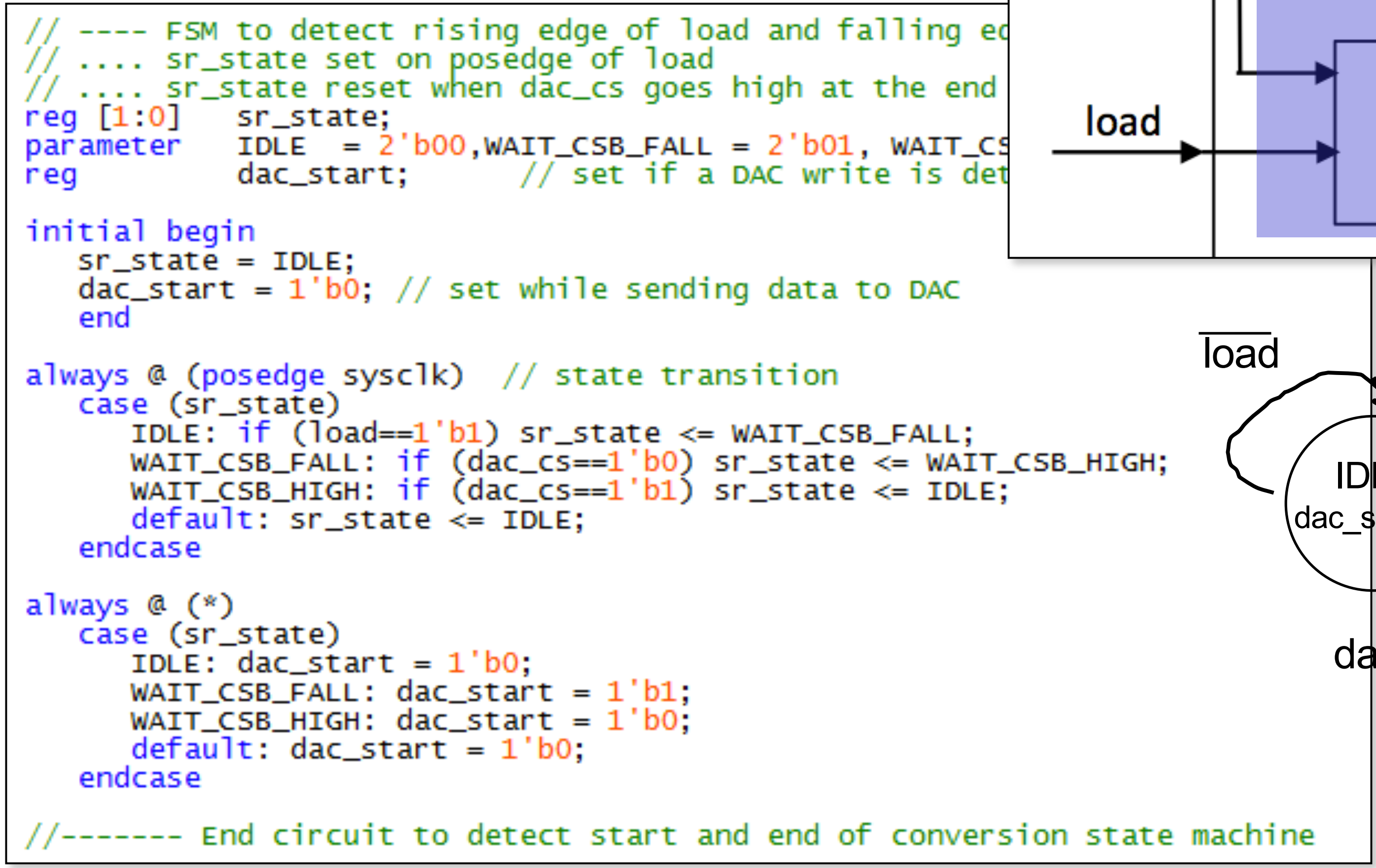
The 1MHz clock generator

```
parameter    BUF=1'b1;       // 0:no buffer, 1:Vref buffered
parameter    GA_N=1'b1;      // 0:gain = 2x, 1:gain = 1x
parameter    SHDN_N=1'b1;    // 0:power down, 1:dac active

wire [3:0] cmd = {1'b0,BUF,GA_N,SHDN_N};  // wire to VDD or GND
```

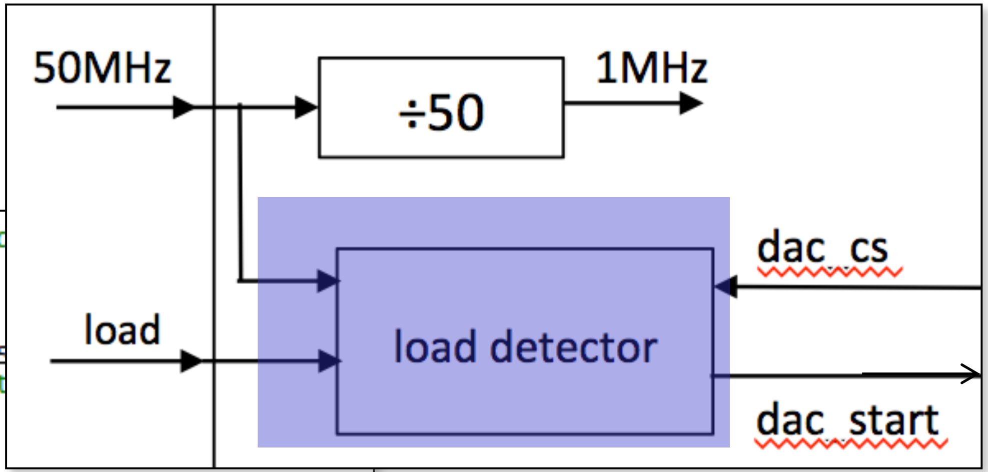


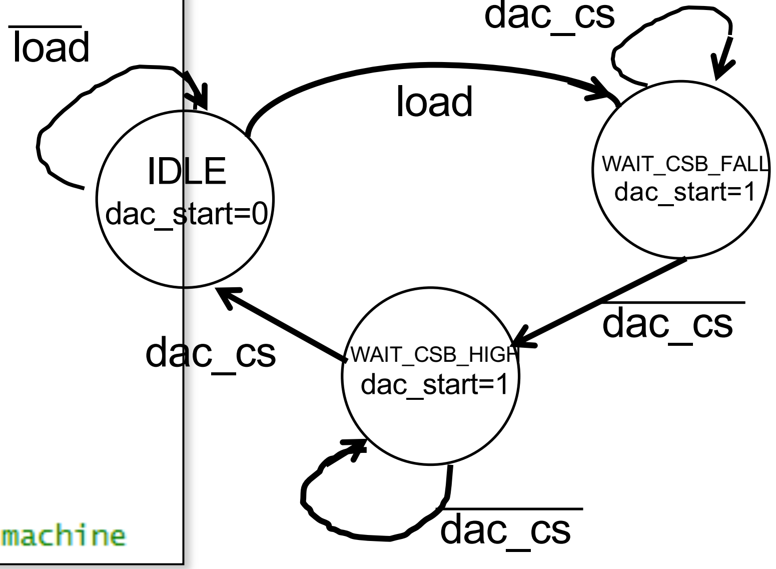
```
// --- internal 1MHz symmetical clock generator ---
reg         clk_1MHz;   // 1Mhz clock derived from
reg [4:0]   ctr;        // internal counter

parameter   TC = 5'd24;  // Terminal count - change
initial begin
   clk_1MHz = 0;        // don't need to reset - don't care i
   ctr = 5'b0;          //  ... Initialise when FPGA is confi
end

always @ (posedge sysclk)
  if (ctr==0) begin
     ctr <= TC;
     clk_1MHz <= ~clk_1MHz; // toggle the output clock for sc
   end
  else
     ctr <= ctr - 1'b1;
// ---- end internal 1MHz symmetical clock generator --------
```

The load pulse detector



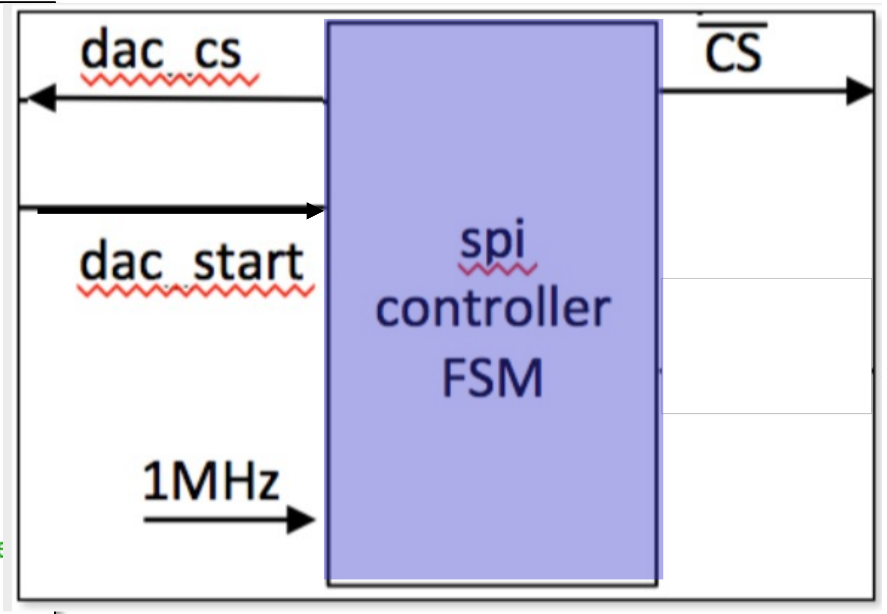
```
// ---- FSM to detect rising edge of load and falling ec
// .... sr_state set on posedge of load
// .... sr_state reset when dac_cs goes high at the end
reg [1:0]   sr_state;
parameter   IDLE  = 2'b00,WAIT_CSB_FALL = 2'b01, WAIT_CS
reg         dac_start;       // set if a DAC write is det

initial begin
   sr_state = IDLE;
   dac_start = 1'b0; // set while sending data to DAC
   end

always @ (posedge sysclk)  // state transition
   case (sr_state)
      IDLE: if (load==1'b1) sr_state <= WAIT_CSB_FALL;
      WAIT_CSB_FALL: if (dac_cs==1'b0) sr_state <= WAIT_CSB_HIGH;
      WAIT_CSB_HIGH: if (dac_cs==1'b1) sr_state <= IDLE;
      default: sr_state <= IDLE;
   endcase

always @ (*)
   case (sr_state)
      IDLE: dac_start = 1'b0;
      WAIT_CSB_FALL: dac_start = 1'b1;
      WAIT_CSB_HIGH: dac_start = 1'b0;
      default: dac_start = 1'b0;
   endcase

//------- End circuit to detect start and end of conversion state machine
```

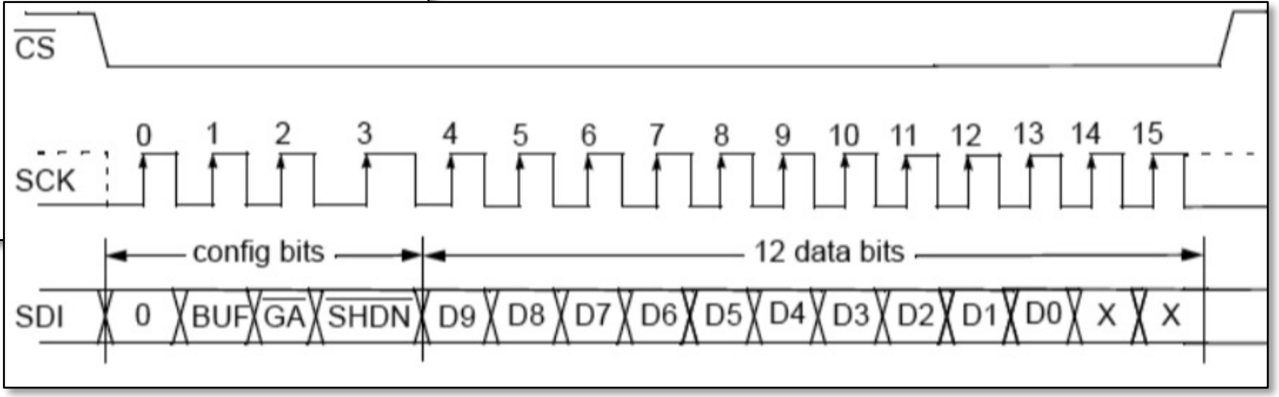




The SPI Controller FSM

```verilog
//------- spi controller FSM
// .... 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_cs = 1'b1;
   end

always @(posedge clk_1MHz)  // FSM state transition
   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;
   case (state)
      5'd0:    dac_cs = 1'b1;
      5'd17:   dac_cs = 1'b1;
      default: dac_cs = 1'b0;
      endcase
   end //always
// --------- END of spi controller FSM
```

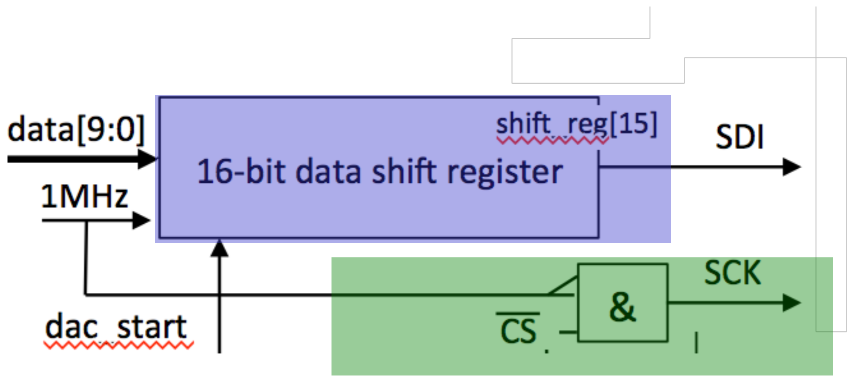




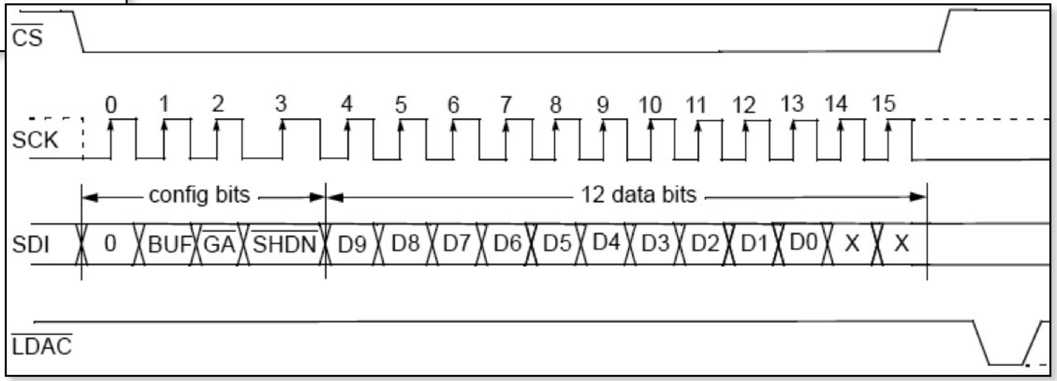
The data shift register

```
parameter   BUF=1'b1;       // 0:no buffer, 1:Vref buffered
parameter   GA_N=1'b1;      // 0:gain = 2x, 1:gain = 1x
parameter   SHDN_N=1'b1;    // 0:power down, 1:dac active

wire [3:0] cmd = {1'b0,BUF,GA_N,SHDN_N};  // wire to VDD or GND
```

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

always @(posedge clk_1MHz)
   if((dac_start==1'b1)&&(dac_cs==1'b1))        //
      shift_reg <= {cmd,data_in,2'b00};
   else                                         //
      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];
```



data[9:0]
1MHz
dac_start
shift_reg[15]
16-bit data shift register
SDI
SCK
&
$\overline{CS}$



$\overline{CS}$
SCK
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
config bits
12 data bits
SDI
0
BUF
GA
SHDN
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
X
X
$\overline{LDAC}$