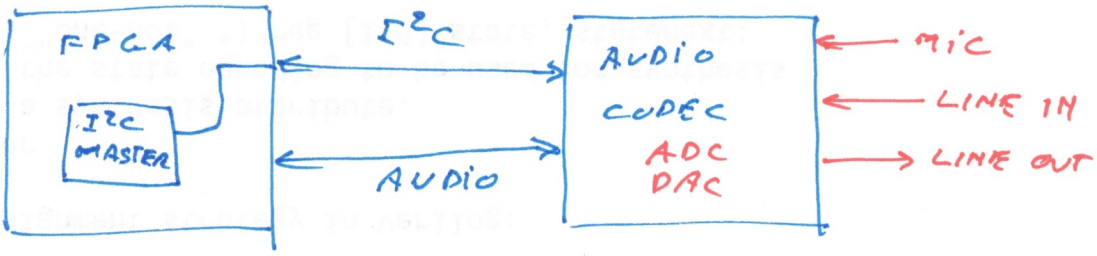
DE1 SOC



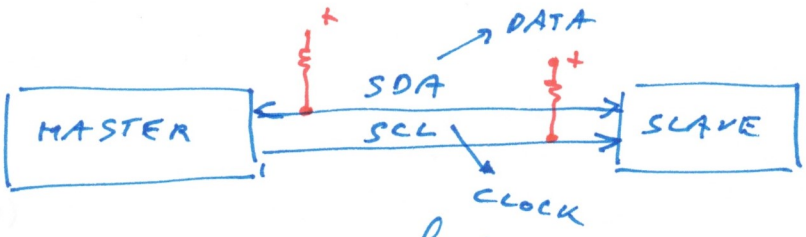
HOMEWORK 2 - SIMULATION I²C

HOMEWORK 3 - PROGRAM CODEC WAVE FORM GENERATION

HOMEWORK 4 - DIGITAL AUDIO SIGNAL GENERATION

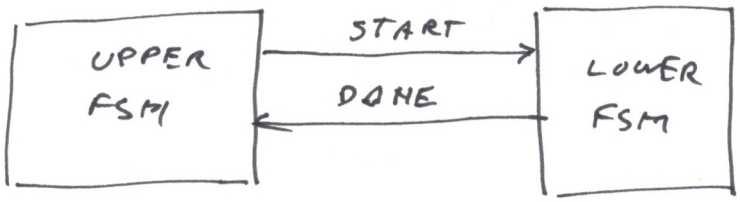
HOMEWORK 5 - RECORDING / PLAY BACK

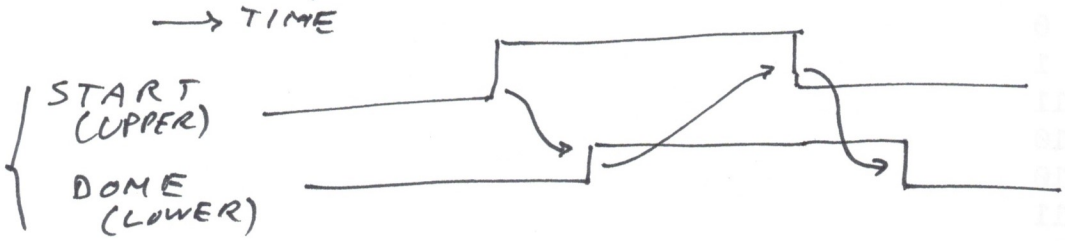
I²C

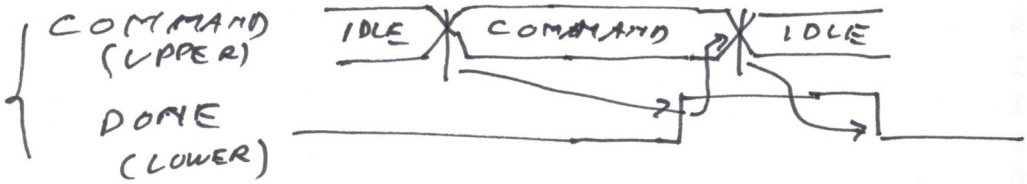


100 kps

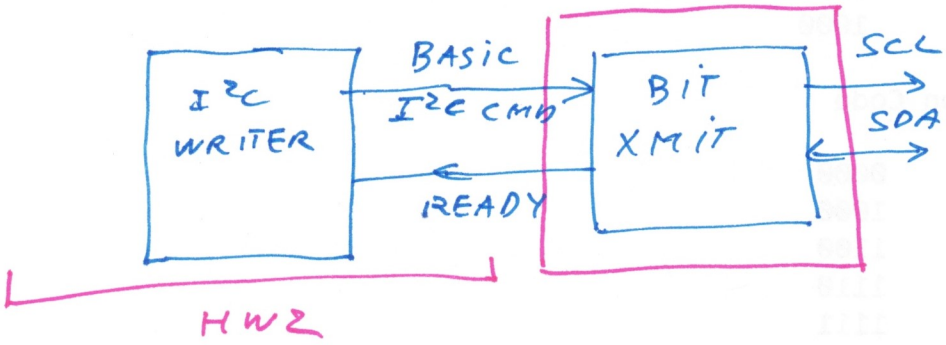
FSM HIERARCHY



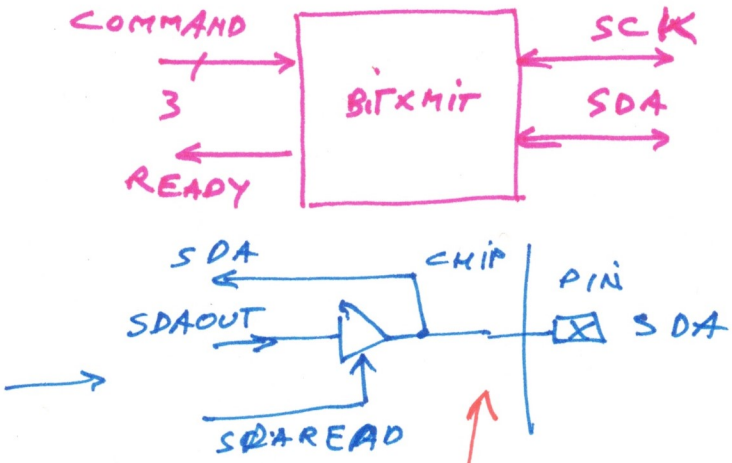




APPLIED TO I²C MASTER



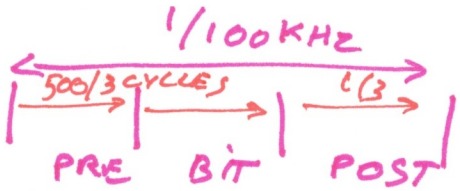
```
`include "bitxmit.h"

module bitxmit(
               input        clk,
               input        reset,

               // commands
               input [2:0]  command,

               // control response
               output       ready,

               // i2c pins
               output       sck,
               inout        sda
               );

  // 500 clock cycles -> 100 kbps for 50MHz clock
  parameter CYCLESPERBIT = 500;

  localparam S0 = 0, PRE = 1, BIT = 2, POST = 3, END = 4;

  reg [2:0]                 state, statenext;
  reg [11:0]                cnt, cntnext;

  reg                       sckout, sckoutnext;
  reg                       sdaout, sdaoutnext;
  reg                       sdaread, sdareadnext;

  assign sck = sckout;
  assign sda = sdaread ? 8'bz : sdaout;
  assign ready = (state == END) ? 1'b1 : 1'b0;

  // clock logic
  always @(posedge clk)
    if (reset)
      begin
         state   <= S0;
         cnt     <= 12'b0;
         sckout  <= 1'b1;
         sdaout  <= 1'b1;
         sdaread <= 1'b1;
      end
    else
      begin
         state   <= statenext;
         cnt     <= cntnext;
         sckout  <= sckoutnext;
```

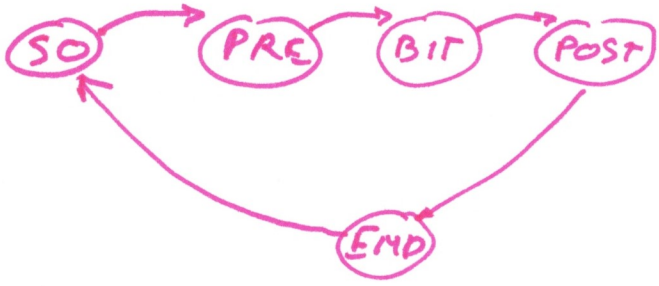


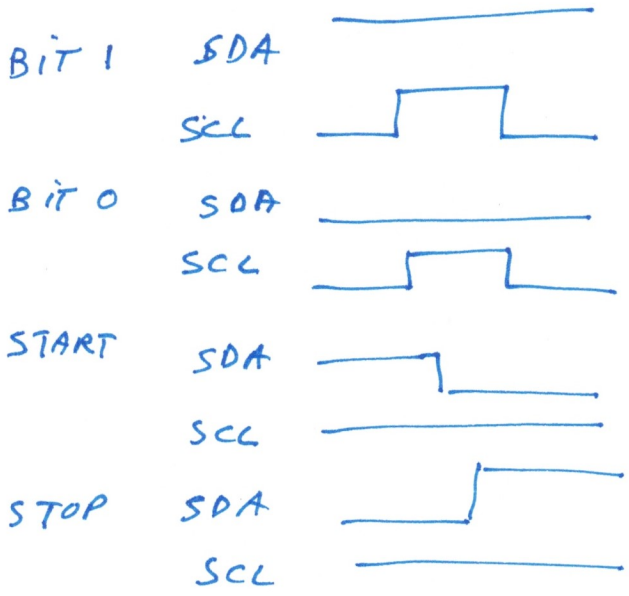
READ

WRITE

```
        sdaout   <= sdaoutnext;
        sdaread <= sdareadnext;
      end

  // state transition logic
  always @(*)
    begin
       statenext = state;
       cntnext    = cnt;

       case (state)

         S0: begin
            cntnext    = CYCLESPERBIT;
            if (command != `CMDWAIT)
              statenext = PRE;
         end

         PRE: begin
            cntnext = cnt - 12'b1;
            if (cnt == CYCLESPERBIT * 2 / 3)
              statenext = BIT;
         end

         BIT: begin
            cntnext = cnt - 12'b1;
            if (cnt == CYCLESPERBIT / 3)
              statenext = POST;
         end

         POST: begin
            cntnext = cnt - 12'b1;
            if (cnt == 0)
              statenext = END;
         end

       END: begin
          cntnext    = CYCLESPERBIT;
          if (command == `CMDWAIT)
            statenext = S0;
       end

         default:
           statenext = S0;

       endcase
    end

  // output decoding
  always @(*)
    begin
       sdareadnext    = sdaread;
```







```
sdaoutnext     = sdaout;
sckoutnext     = sckout;

case (command)

  `CMDIDLE: begin
     sdareadnext = 1'b1;
     sdaoutnext  = 1'b1;
     sckoutnext  = 1'b1;
  end

  `CMDWAIT: begin
  end

  `CMDBIT0: begin
     sdareadnext = (state == POST) ? 1'b1 : 1'b0;
     sdaoutnext  = 1'b0;
     sckoutnext  = (state == PRE)  ? 1'b0 :
                   (state == BIT)  ? 1'b1 :
                   (state == POST) ? 1'b0 :
                   sckout ;
  end

  `CMDBIT1: begin
     sdareadnext = (state == POST) ? 1'b1 : 1'b0;
     sdaoutnext  = 1'b1;
     sckoutnext  = (state == PRE)  ? 1'b0 :
                   (state == BIT)  ? 1'b1 :
                   (state == POST) ? 1'b0 :
                   sckout ;
  end

  `CMDSTART: begin
     sdareadnext = 1'b0;
     sdaoutnext  = (state == PRE)  ? 1'b1 :
                   (state == BIT)  ? 1'b0 :
                   (state == POST) ? 1'b0 :
                   sdaout;
     sckoutnext  = 1'b1;
  end

  `CMDSTOP: begin
     sdareadnext = 1'b0;
     sdaoutnext  = (state == PRE)  ? 1'b0 :
                   (state == BIT)  ? 1'b1 :
                   (state == POST) ? 1'b1 :
                   sdaout ;
     sckoutnext  = 1'b1;
  end

  `CMDRBIT: begin
     sdareadnext = 1'b1;
```

```
            sdaoutnext  = 1'b1;
            sckoutnext  = (state == PRE)  ? 1'b0 :
                          (state == BIT)  ? 1'b1 :
                          (state == POST) ? 1'b0 :
                          sckout;
        end

      endcase
    end


endmodule
```