Verilog Code

msg_module.v

```
`timescale 1ns / 1ps

module msg_module(

        input clk,
        input byte_sent,                // connects to tx_dly_rdy in UART module

        input msg_transmit,             // starts transmission of a number of bytes
        input [7:0] byte_1,             // connects to sens_id in top module
        input [7:0] byte_2,             // connects to v0-v5 in top module

        output reg [7:0] byte_N,        // connects to tx_byte in UART module
        output send_byte,               // connects to transmit in top module


        output msg_sent                 // used to create clr_flgN signals in top module


);




//////////////////////////////////////////////////////////////////////////////////
// START Five message state machine
//////////////////////////////////////////////////////////////////////////////////



parameter MSG_IDLE = 0;
parameter BYTE_1 = 1;
parameter BYTE_2 = 2;
parameter BYTE_3 = 3;
parameter BYTE_4 = 4;
parameter BYTE_5 = 5;



reg [7:0] tx_byte_reg;

wire [7:0] status = 8'h99;                      // Note on, channel 9
```

Verilog Code

```
//wire [7:0] data1 = 8'h00;                     // Key #0
//wire [7:0] data2_on = 8'h7F;          // max (fixed) velocity
wire [7:0] data2_off = 8'h00;           // velocity of 0 = Note off


reg [2:0] msg_state = MSG_IDLE;         // 8 possible states


reg [2:0] st_temp0; always@(posedge clk) st_temp0[2:0]=msg_state[2:0];

assign send_byte = ((msg_state==1 && st_temp0==0) || (msg_state==2 && st_temp0==1) ||
                                                    (msg_state==3 && st_temp0==2) ||
(msg_state==4 && st_temp0==3) ||
                                                    (msg_state==5 && st_temp0==4));

assign msg_sent = (msg_state==0 && st_temp0==5);


assign tx_byte = tx_byte_reg;


always @(posedge clk) begin

        // MIDI Note on/of message transmit state machine
        case (msg_state)

                MSG_IDLE: begin
                        if(msg_transmit)begin
                                byte_N = status;
                                msg_state = BYTE_1;             // begin sending 5 bytes if this
state machine is triggered
                        end
                end

                BYTE_1: begin
                        if (byte_sent) begin
                                byte_N = byte_1;                // store byte to be tx'ed
                                msg_state = BYTE_2;
                        end
                end

                BYTE_2: begin
                        if (byte_sent) begin
                                byte_N = byte_2;
```

Verilog Code

```
                                msg_state = BYTE_3;
                        end
                end

                BYTE_3: begin
                        if (byte_sent) begin
                                byte_N = byte_1;
                                msg_state = BYTE_4;
                        end
                end


                BYTE_4: begin
                        if (byte_sent) begin
                                byte_N = data2_off;
                                msg_state = BYTE_5;

                        end
                end

                BYTE_5: begin
                        if (byte_sent) begin
                                msg_state = MSG_IDLE;
                        end
                end


        endcase
end

////////////////////////////////////////////////////////////////////////////////////
// END Five message state machine
////////////////////////////////////////////////////////////////////////////////////



Endmodule


Peak_finder.v

`timescale 1ns / 1ps
```

Verilog Code

```
module msg_module(

        input clk,
        input byte_sent,                // connects to tx_dly_rdy in UART module

        input msg_transmit,             // starts transmission of a number of bytes
        input [7:0] byte_1,             // connects to sens_id in top module
        input [7:0] byte_2,             // connects to v0-v5 in top module

        output reg [7:0] byte_N,        // connects to tx_byte in UART module
        output send_byte,               // connects to transmit in top module


        output msg_sent                 // used to create clr_flgN signals in top module


);




////////////////////////////////////////////////////////////////////////////////////////
// START Five message state machine
////////////////////////////////////////////////////////////////////////////////////////



parameter MSG_IDLE = 0;
parameter BYTE_1 = 1;
parameter BYTE_2 = 2;
parameter BYTE_3 = 3;
parameter BYTE_4 = 4;
parameter BYTE_5 = 5;



reg [7:0] tx_byte_reg;

wire [7:0] status = 8'h99;                      // Note on, channel 9
//wire [7:0] data1 = 8'h00;                     // Key #0
//wire [7:0] data2_on = 8'h7F;          // max (fixed) velocity
wire [7:0] data2_off = 8'h00;           // velocity of 0 = Note off
```

Verilog Code

```
reg [2:0] msg_state = MSG_IDLE;          // 8 possible states


reg [2:0] st_temp0; always@(posedge clk) st_temp0[2:0]=msg_state[2:0];

assign send_byte = ((msg_state==1 && st_temp0==0) || (msg_state==2 && st_temp0==1) ||
                                                     (msg_state==3 && st_temp0==2) ||
(msg_state==4 && st_temp0==3) ||
                                                     (msg_state==5 && st_temp0==4));

assign msg_sent = (msg_state==0 && st_temp0==5);


assign tx_byte = tx_byte_reg;


always @(posedge clk) begin

        // MIDI Note on/of message transmit state machine
        case (msg_state)

                MSG_IDLE: begin
                                if(msg_transmit)begin
                                        byte_N = status;
                                        msg_state = BYTE_1;          // begin sending 5 bytes if this
state machine is triggered
                                end
                end

                BYTE_1: begin
                        if (byte_sent) begin
                                        byte_N = byte_1;             // store byte to be tx'ed
                                        msg_state = BYTE_2;
                        end
                end

                BYTE_2: begin
                        if (byte_sent) begin
                                        byte_N = byte_2;
                                        msg_state = BYTE_3;
                        end
                end
```

Verilog Code

```
                BYTE_3: begin
                        if (byte_sent) begin
                                        byte_N = byte_1;
                                        msg_state = BYTE_4;
                        end
                end


                BYTE_4: begin
                        if (byte_sent) begin
                                        byte_N = data2_off;
                                        msg_state = BYTE_5;

                        end
                end

                BYTE_5: begin
                        if (byte_sent) begin
                                msg_state = MSG_IDLE;
                        end
                end


        endcase
end

////////////////////////////////////////////////////////////////////////////////////
// END Five message state machine
////////////////////////////////////////////////////////////////////////////////////




Endmodule
```

Top.v

```
`timescale 1ns / 1ps

module top(

////////////////////////////////////////////////////////////////////////////////////////////
// Top module ports do not need to be declared if they are connected in a lower module instantiation
// Doing so will reserve a physical pin for the signal, which is not needed for internal signals
```

Verilog Code

```
////////////////////////////////////////////////////////////////////////////////////////////////////

        input clk,
        input IR_enable,
                                        // position switch

//      output clr0,clr1,clr2,clr3,clr4,clr5,

//      output reg [6:0] data1, data2,
        //output [3:0] led,
        output [1:0] mux_cntrl_bus,
        output [3:0] led,
        //output tx_wire,
        //output wr_wire,
//      output single_click,

// Port declaration common to all sub modules
        input clock,

// peak_finder port declarations
        input [7:0] sensor,
        input clear,
        output reg [7:0] velocity,
        output reg flag,

// ADC and MUX control port declarations
        output reg [6:0] s0,s1,s2,s3,s4,s5,s6,s7,                       // sensor data registers
        input [7:0] ADC0_data, ADC1_data,
        output wr, csn,
        output reg [1:0] mux_cntrl,

// PB_debouncer port declarations
        input PB,       // also listed here for the top module
        output PB_down,

// double click finder port declarations
        /*output single_click,                                          // tied to PB_down
        output double_click,*/

// MIDI message module port declarations
        input msg_transmit,             // starts transmission of a number of bytes
        input [7:0] byte_1,             // connects to sens_id in top module
        input [7:0] byte_2,             // connects to v0-v5 in top module
```

Verilog Code

```
        output [7:0] byte_N,      // connects to tx_byte in sub module, in of sub
                                                                    // must be declared here or will be
assumed as one bit

        output send_byte,               // connects to transmit in sub module, in of sub
        input byte_sent,                // connects to tx_dly_rdy in sub module, out of sub

        output msg_sent,                // used to create clr_flgN signals in top module

// UART port declarations
        input [7:0] tx_byte,
        input transmit,
        output tx,                                  // listed here for both modules
        output tx_rdy_dly


// low battery  module
        /*input battery_voltage,        // connects to battery in top module
        output low_bat_led*/            // connects to LED, flashes when battery is low

 );


wire [7:0] s0_w,s1_w,s2_w,s3_w,s4_w,s5_w,s6_w,s7_w;
wire [7:0] v0,v1,v2,v3,v4,v5,IR_signal,battery;                  // battery connects to battery_voltage
in sub module

wire clr0,clr1,clr2,clr3,clr4,clr5;


reg [7:0] data1, data2;

wire [7:0] data1_w = data1;
wire [7:0] data2_w = data2;

//output tx_wire,
//output wr_wire,
//wire single_click;

assign csn = 0;


 // Module instantiations
```

Verilog Code

```
 peak_finder pk_finder0 (

  .clock                    (clk),              // an input in top and sub modules
        .sensor             (s0_w),             // sensor = 7 bit input wire in sub
module
                                                                    // s0 =
7 bit output reg in top module
        .clear              (clr0),             // clear = input wire in sub module
                                                                    // clr0 =
output wire in top module
        .flag               (flg0),             // flag = output reg in sub
module
                                                                    // flg0 =
output wire in top module

        .single_click       (single_click),// for debugging


        .velocity           (v0)                // velocity = 7 bit output reg in
sub module
                                                                    // v0 =
7 bit output wire in top module
        );

peak_finder pk_finder1 (
  .clock                    (clk),
  .sensor                   (s1_w),
        .clear              (clr1),
        .flag               (flg1),
        .velocity           (v1)
        );

peak_finder pk_finder2 (
  .clock                    (clk),
  .sensor                   (s2_w),
        .clear              (clr2),
        .flag               (flg2),
        .velocity           (v2)
        );

peak_finder pk_finder3 (
  .clock                    (clk),
  .sensor                   (s3_w),
        .clear              (clr3),
```

Verilog Code

```
        .flag                   (flg3),
        .velocity               (v3)
        );

peak_finder pk_finder4 (
   .clock                   (clk),
   .sensor                  (s4_w),
        .clear              (clr4),
        .flag               (flg4),
        .velocity           (v4)
        );

peak_finder pk_finder5 (
   .clock                   (clk),
   .sensor                  (s5_w),
        .clear              (clr5),
        .flag                       (flg5),
        .velocity           (v5)
        );


wire [7:0] byte_N_w;

msg_module          msg_module(
        .clk                        (clk),
        .msg_transmit       (send_msg),
//      .msg_transmit       (single_click),
        .byte_N             (byte_N_w),             // output

        .byte_1             (data1_w),              // input
        .byte_2             (data2_w),              // input
        .send_byte          (send_byte),
        .byte_sent          (tx_rdy_dly),           // input
        .msg_sent           (msg_sent)

);

UART    UART(

        .clk                (clk),
        .tx_byte            (byte_N_w),

        .transmit           (send_byte),            // input
        .tx                 (tx),
```

Verilog Code

```
        .tx_rdy_dly             (tx_rdy_dly)

);


PB_debouncer  PB_debouncer(
        .clk                    (clk),
        .PB                     (PB),                           // into sub from pins
        .PB_down                (single_click)          // out of sub to top then into double click finder

);



ADC_MUX_control ADC_MUX_control(

        .clk                            (clk),
        .ADC0_data                      (ADC0_data),    // into sub from pins
        .ADC1_data                      (ADC1_data),    // into sub from pins
        .s0(s0_w), .s1(s1_w), .s2(s2_w), .s3(s3_w), .s4(s4_w), .s5(s5_w), .s6(IR_signal), .s7(battery),
        // out of sub to top then into peak finders
        .mux_cntrl                      (mux_cntrl_bus),                // out of sub to pins
        .wr                                     (wr)                                                    // out
of sub to pins
);


//-----------------------------------------------------------------------------------------------------------------
//Parse flags, when one is set sensor id and velocity info is sent and a signal is asserted to clear that flag
//-----------------------------------------------------------------------------------------------------------------


reg [1:0] buf_st = 0;
reg [1:0] buf_st_temp = 0; always@ (posedge clk) buf_st_temp <= buf_st;
reg [2:0] flag_index = 0;

parameter READ_N = 0;
parameter ASSIGN_N = 1;
parameter SEND_N = 2;
parameter INC = 3;

parameter IR_THRESH = 120;
//wire tble = (IR_signal > IR_THRESH);
wire tble = 0;
```

Verilog Code

```
wire [5:0] flags = {flg5,flg4,flg3,flg2,flg1,flg0};


/*reg [5:0] flag_reg;
wire [5:0] flags = flag_reg;*/

reg [3:0] led_r;
assign led = led_r;


// this could be a problem area

assign clr0 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==0));
assign clr1 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==1));
assign clr2 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==2));
assign clr3 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==3));
assign clr4 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==4));
assign clr5 = ((buf_st==ASSIGN_N) && (buf_st_temp==READ_N) && (flag_index==5));

assign send_msg = ((buf_st==SEND_N) && (buf_st_temp==ASSIGN_N));
//assign send_msg = ((buf_st==INC) && (buf_st_temp==SEND_N));




// For debugging
always@(posedge clk)begin
        if(single_click) begin
                led_r <= ~(led_r);
        end
end




always@ (posedge clk) begin

        case(buf_st)

                READ_N:begin
```

Verilog Code

```
                if(flags[flag_index]==0) buf_st <= INC;
                else buf_st <= ASSIGN_N;
        end

        ASSIGN_N:begin

        //buf_st <= SEND_N;

                if(IR_enable)begin

                        if(tble)
                                case(flag_index)
                                        0:begin
                                                data1 <= 6;                     // assign sensor
id
                                                data2 <=        v0;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        1:begin
                                                data1 <= 7;                     // assign sensor
id
                                                data2 <=        v1;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        2:begin
                                                data1 <= 8;                     // assign sensor
id
                                                data2 <=        v2;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        3:begin
                                                data1 <= 9;                     // assign sensor
id
                                                data2 <=        v3;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        4:begin
                                                data1 <=10;                     // assign sensor
id
```

Verilog Code

```
                                        data2 <=        v4;                     //
assign velocity
                                        buf_st <= SEND_N;
                                end
                                5:begin
                                        data1 <= 11;                    // assign sensor
id
                                        data2 <=        v5;                     //
assign velocity
                                        buf_st <= SEND_N;
                                end

                        endcase
                end

                else begin // !tble case
                        case(flag_index)
                                0:begin
                                        data1 <= 0;                     // assign sensor
id
                                        data2 <=        v0;                     //
assign velocity
                                        buf_st <= SEND_N;
                                end
                                1:begin
                                        data1 <= 1;                     // assign sensor
id
                                        data2 <=        v1;                     //
assign velocity
                                        buf_st <= SEND_N;
                                end
                                2:begin
                                        data1 <= 2;                     // assign sensor
id
                                        data2 <=        v2;                     //
assign velocity
                                        buf_st <= SEND_N;
                                end
                                3:begin
                                        data1 <= 3;                     // assign sensor
id
                                        data2 <=        v3;                     //
assign velocity
                                        buf_st <= SEND_N;
```

Verilog Code

```
                                        end
                                        4:begin
                                                data1 <= 4;                     // assign sensor
id
                                                data2 <=        v4;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        5:begin
                                                data1 <= 5;                     // assign sensor
id
                                                data2 <=        v5;                     //
assign velocity =  because !tble case
                                                buf_st <= SEND_N;
                                        end

                                endcase

                        end

                //end


                if(!IR_enable) begin
                                case(flag_index)
                                        0:begin
                                                data1 <= 0;                     // assign sensor
id
                                                data2 <=        v0;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        1:begin
                                                data1 <= 1;                     // assign sensor
id
                                                data2 <=        v1;                     //
assign velocity
                                                buf_st <= SEND_N;
                                        end
                                        2:begin
                                                data1 <= 2;                     // assign sensor
id
                                                data2 <=        v2;                     //
assign velocity
```

Verilog Code

```
                                        buf_st <= SEND_N;
                                    end
                                    3:begin
                                        data1 <= 3;                 // assign sensor
id
                                        data2 <=    v3;                 //
assign velocity
                                        buf_st <= SEND_N;
                                    end
                                    4:begin
                                        data1 <= 4;                 // assign sensor
id
                                        data2 <=    v4;                 //
assign velocity
                                        buf_st <= SEND_N;
                                    end
                                    5:begin
                                        data1 <= 5;                 // assign sensor
id
                                        data2 <=    v5;                 //
assign velocity
                                        buf_st <= SEND_N;
                                    end

                                endcase
                        end
                end

                SEND_N:begin
                        if(msg_sent) buf_st <= INC;
                        // buf_st <= INC;
                end

                INC:begin
                        flag_index <= flag_index + 1;
                        buf_st <= READ_N;
                end

        endcase
end
```

Verilog Code

```
endmodule
```

UART.v

```
`timescale 1ns / 1ps

module UART(

        input clk,
        input [7:0] tx_byte,                // tx data
        input transmit,             // starts transmission of tx_byte
        output tx,
        output tx_rdy_dly                       // connects to byte_sent

);

//////////////////////////////////////////////////////////////////////////////////
// START UART
//////////////////////////////////////////////////////////////////////////////////

parameter CLOCK_DIVIDE = 400; // clock rate (50Mhz) / (baud rate (9600) * 4), 108.5 for 115200
                                                                                // 400 = 31250


// States for the transmitting state machine.
// Constants - do not override.
parameter TX_IDLE = 0;
parameter TX_SENDING = 1;
parameter TX_DELAY_RESTART = 2;


reg [10:0] tx_clk_divider = CLOCK_DIVIDE;

reg tx_out = 1'b1;
reg [1:0] tx_state = TX_IDLE;
reg [5:0] tx_countdown;
reg [3:0] tx_bits_remaining;
```

Verilog Code

```
reg [7:0] tx_data;

assign tx = tx_out;
assign is_transmitting = tx_state != TX_IDLE;

wire rst = 0;


always @(posedge clk) begin
	if (rst) begin
		tx_state = TX_IDLE;
	end

	// The clk_divider counter counts down from
	// the CLOCK_DIVIDE constant. Whenever it
	// reaches 0, 1/16 of the bit period has elapsed.
  // Countdown timers for the receiving and transmitting
	// state machines are decremented.

	tx_clk_divider = tx_clk_divider - 1;
	if (!tx_clk_divider) begin
		tx_clk_divider = CLOCK_DIVIDE;
		tx_countdown = tx_countdown - 1;
	end



	// Transmit state machine
	case (tx_state)
		TX_IDLE: begin
			if (transmit) begin
				// If the transmit flag is raised in the idle
				// state, start transmitting the current content
				// of the tx_byte input.
				tx_data = tx_byte;
				// Send the initial, low pulse of 1 bit period
				// to signal the start, followed by the data
				tx_clk_divider = CLOCK_DIVIDE;
				tx_countdown = 4;
				tx_out = 0;
				tx_bits_remaining = 8;
				tx_state = TX_SENDING;
			end
		end
```

Verilog Code

```
                TX_SENDING: begin
                        if (!tx_countdown) begin
                                if (tx_bits_remaining) begin
                                        tx_bits_remaining = tx_bits_remaining - 1;
                                        tx_out = tx_data[0];
                                        tx_data = {1'b0, tx_data[7:1]};
                                        tx_countdown = 4;
                                        tx_state = TX_SENDING;
                                end else begin
                                        // Set delay to send out 2 stop bits.
                                        tx_out = 1;
                                        tx_countdown = 8;
                                        tx_state = TX_DELAY_RESTART;
                                end
                        end
                end
                TX_DELAY_RESTART: begin
                        // Wait until tx_countdown reaches the end before
                        // we send another transmission. This covers the
                        // "stop bit" delay.
                        tx_state = tx_countdown ? TX_DELAY_RESTART : TX_IDLE;
                end
        endcase
end

////////////////////////////////////////////////////////////////////////////////////////
// END UART
////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////
// START lengthen is_transmitting
////////////////////////////////////////////////////////////////////////////////////////

reg ext_tx_busy0; always@(posedge clk) ext_tx_busy0<= is_transmitting;
reg ext_tx_busy1; always@(posedge clk) ext_tx_busy1<= ext_tx_busy0;
reg ext_tx_busy2; always@(posedge clk) ext_tx_busy2<= ext_tx_busy1;
reg ext_tx_busy3; always@(posedge clk) ext_tx_busy3<= ext_tx_busy2;
reg ext_tx_busy4; always@(posedge clk) ext_tx_busy4<= ext_tx_busy3;
reg ext_tx_busy5; always@(posedge clk) ext_tx_busy5<= ext_tx_busy4;
reg ext_tx_busy6; always@(posedge clk) ext_tx_busy6<= ext_tx_busy5;
reg ext_tx_busy7; always@(posedge clk) ext_tx_busy7<= ext_tx_busy6;
reg ext_tx_busy8; always@(posedge clk) ext_tx_busy8<= ext_tx_busy7;
reg ext_tx_busy9; always@(posedge clk) ext_tx_busy9<= ext_tx_busy8;
reg ext_tx_busy10; always@(posedge clk) ext_tx_busy10<= ext_tx_busy9;
```

Verilog Code

```
reg ext_tx_busy11; always@(posedge clk) ext_tx_busy11<= ext_tx_busy10;

assign tx_rdy_dly = (ext_tx_busy11==1 && ext_tx_busy10==0);

//////////////////////////////////////////////////////////////////////////////////////
// END lengthen is_transmitting
//////////////////////////////////////////////////////////////////////////////////////


Endmodule


ADC_MUX_control.v

`timescale 1ns / 1ps

module ADC_MUX_control(

        input clk,
        input PB,
        input [7:0] ADC0_data, ADC1_data,
        output reg [7:0] s0,s1,s2,s3,s4,s5,s6,s7,
        output wr,
        output reg [1:0] mux_cntrl

   );



reg [6:0] cnv_cnt;          // counter to create WR' control signal
wire cnv_cnt_ovf = (cnv_cnt==95);  // counter overflows at 95
reg wr_reg = 1;  // initialize WR' signal to 1

assign wr = wr_reg;



always@(posedge clk)begin

        if(cnv_cnt_ovf) begin

                cnv_cnt<=0;                // if counter overflows, reset it
```

Verilog Code

```
                mux_cntrl<=mux_cntrl + 1;       // inc mux_cntrl
                //mux_cntrl <= 0;

                case (mux_cntrl)

                        0:      begin
                                s0 <= ADC0_data;        // take data at end of conversion and store it for
later use
                                s4 <= ADC1_data;
                        end
                        1:      begin
                                s1 <= ADC0_data;
                                s5 <= ADC1_data;
                        end
                        2:      begin
                                s2 <= ADC0_data;
                                s6 <= ADC1_data;
                        end
                        3:      begin
                                s3 <= ADC0_data;
                                s7 <= ADC1_data;
                        end

                endcase

        end
        else cnv_cnt <= cnv_cnt + 1;      // else keep incrementing counter with every 50MHz rising clock
edge

        if(cnv_cnt>25 && cnv_cnt<=55) wr_reg<=0;       // active low pulse of the WR' signal to start
conversion
        else wr_reg<=1;

end

//-----------------------------------------------------------------------------------------------------
//  END write mode standalone logic
//-----------------------------------------------------------------------------------------------------


endmodule
```

Verilog Code