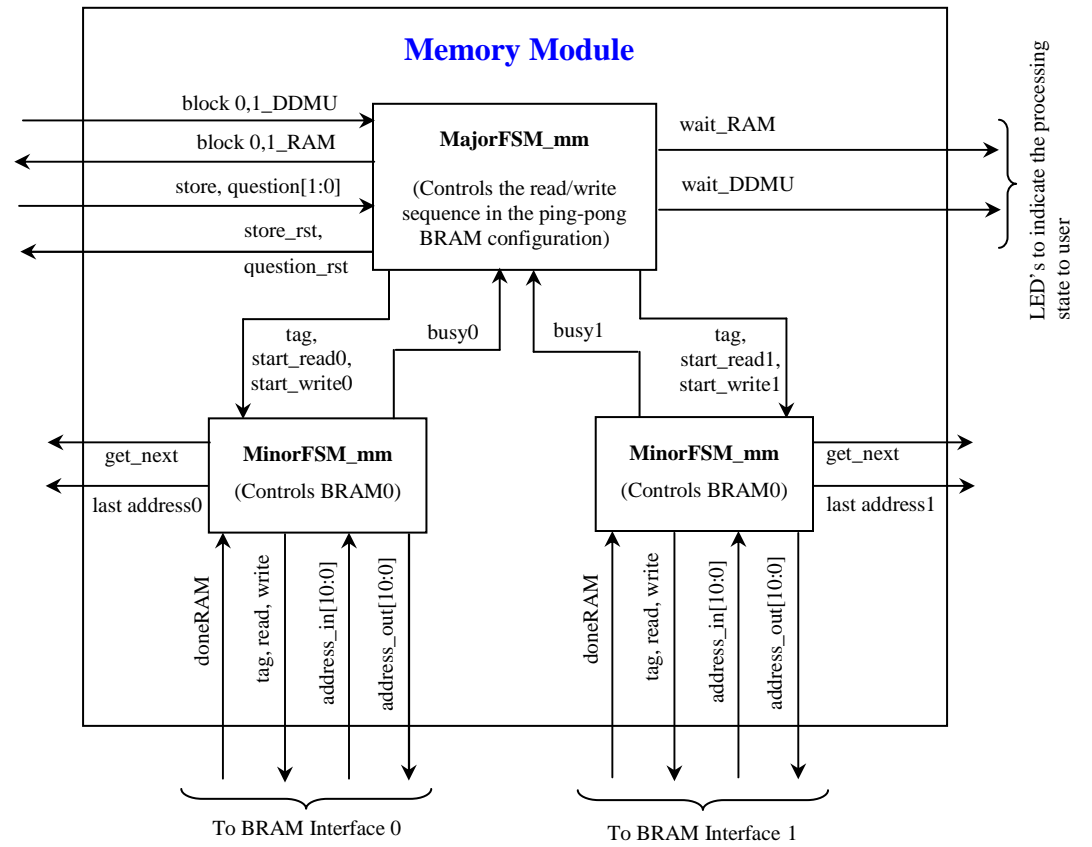
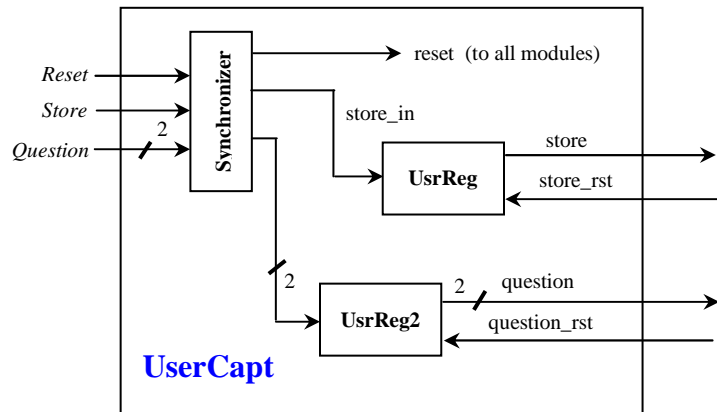
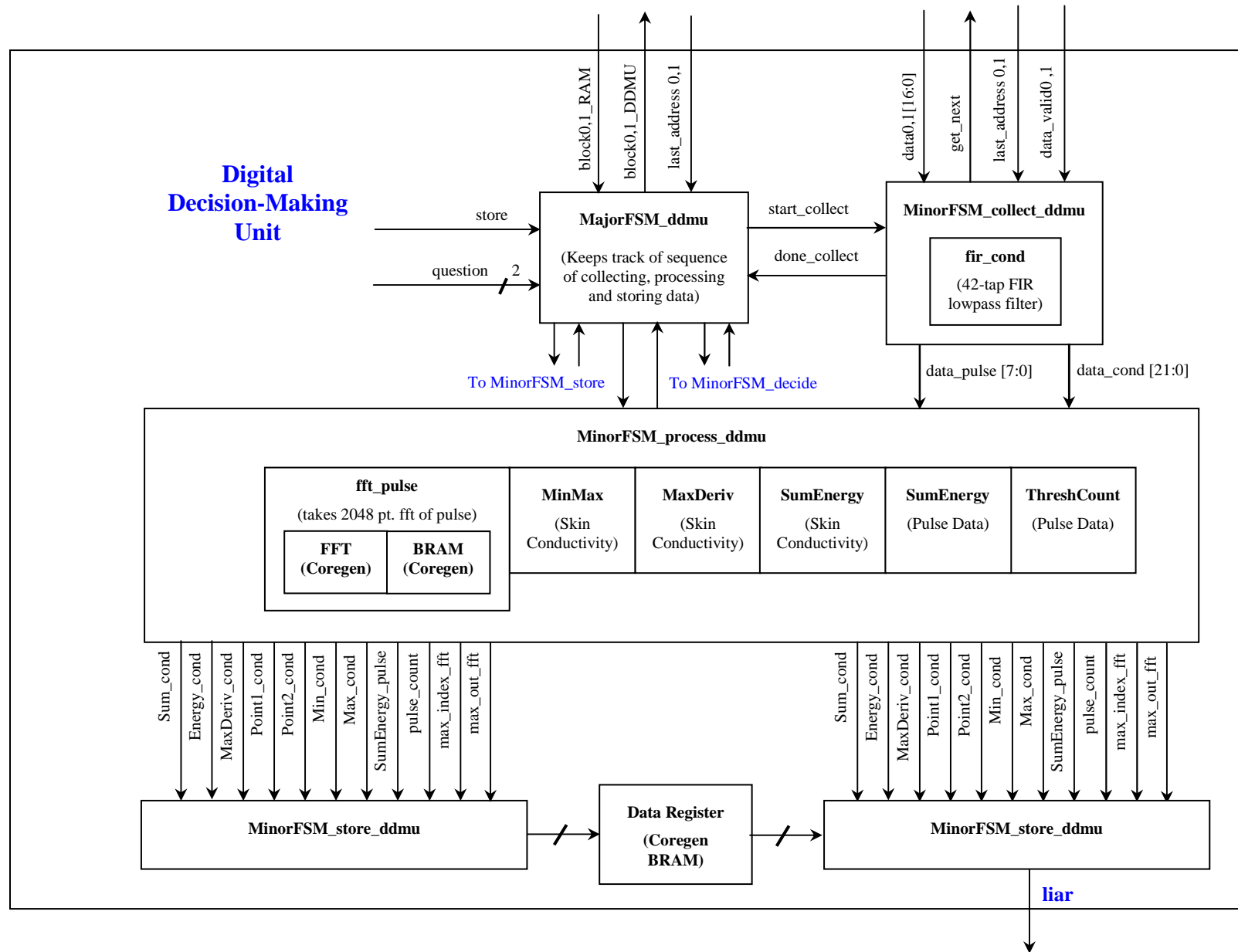


PolyVideo.v





\*\*\*\*\*

CODE BY CHRISTOPHER BUENROSTRO FOR ANALOG SENSORS

\*\*\*\*\*

`timescale 1ns / 1ps

```
module Recorder
(
// Global inputs
    input          global_clock,
    input          global_reset,
// Local inputs from user
    input          record_request,
    input          extra_bit_value,
    input          data_clear,
// Local inputs from devices
    input          ADC_busy_from_ADC_1,
    input          ADC_busy_from_ADC_2,
    input          RAM_busy_from_RAM,
    input [7:0]    ADC_1_data,
    input [7:0]    ADC_2_data,
// outputs to user
    output [10:0]  next_write_address,
    output        recorder_busy,
// outputs to devices
    output        ADC_request_to_device,
    output        RAM_write_request_to_device,
    output [16:0]  RAM_write_data_to_device,
    output        full,
    output [3:0]   state_out
);

reg        record_request_reg;
reg        extra_bit_value_reg;
reg        ADC_busy_from_ADC_1_reg;
reg        ADC_busy_from_ADC_2_reg;
reg        RAM_busy_from_RAM_reg;
reg [7:0]   ADC_1_data_reg;
reg [7:0]   ADC_2_data_reg;
reg [10:0]  next_write_address_reg;
reg        full_reg;
reg        data_clear_reg;

reg [3:0]   state, next;

parameter RESET          = 0;
parameter IDLE           = 1;
parameter ADC_DATA       = 2;
parameter RAM_WRITE      = 3;
parameter INC_ADDRESS    = 4;
parameter FULL           = 5;
parameter RESET_ADDRESS  = 6;

parameter address_max = 1499;
```

```

always @ (posedge global_clock)
begin
    if (global_reset)
        state <= RESET;
    else if (data_clear_reg)
//*****
        state <= RESET_ADDRESS;
//*****
    else
        state <= next;
end

always @ (posedge global_clock)
begin
    if ( (state == RESET) || (state == RESET_ADDRESS) )
        next_write_address_reg <= 0;
    else if (state == INC_ADDRESS)
        next_write_address_reg <= next_write_address_reg + 1;
end

always @ (posedge global_clock)
begin
    record_request_reg <= record_request;
    extra_bit_value_reg <= extra_bit_value;
    ADC_busy_from_ADC_1_reg <= ADC_busy_from_ADC_1;
    ADC_busy_from_ADC_2_reg <= ADC_busy_from_ADC_2;
    RAM_busy_from_RAM_reg <= RAM_busy_from_RAM;
    data_clear_reg <= data_clear;

    if (state == ADC_DATA)
    begin
        ADC_1_data_reg <= ADC_1_data;
        ADC_2_data_reg <= ADC_2_data;
    end
end

always @ (posedge global_clock)
begin
    if (next_write_address == address_max)
        full_reg <= 1;
    else
        full_reg <= 0;
end

always @ (state)
begin

    case(state)
        RESET:    next = IDLE;
        IDLE:
            begin
                if (full_reg)
                    next = FULL;
                else if (record_request_reg)
                    next = ADC_DATA;
            end
    endcase
end

```

```

        else
            next = IDLE;
        end
    ADC_DATA:
        begin
            if (ADC_busy_from_ADC_1_reg || ADC_busy_from_ADC_2_reg)
                next = ADC_DATA;
            else
                next = RAM_WRITE;
            end
        RAM_WRITE:
            begin
                if (RAM_busy_from_RAM_reg)
                    next = RAM_WRITE;
                else
                    next = INC_ADDRESS;
                end
            INC_ADDRESS:
                next = IDLE;
            FULL:
                begin
                    if (data_clear_reg)
                        next = RESET_ADDRESS;
                    else
                        next = FULL;
                    end
                RESET_ADDRESS:
                    next = IDLE;
                default;;
            endcase
        end

    assign recorder_busy          = (state == IDLE)          ? 0 : 1;
    assign ADC_request_to_device  = (state == ADC_DATA)      ? 1 : 0;
    assign RAM_write_request_to_device = (state == RAM_WRITE) ? 1 : 0;
    assign RAM_write_data_to_device = {extra_bit_value_reg, ADC_1_data_reg,
    ADC_2_data_reg};
    assign next_write_address     = next_write_address_reg;
    assign full                   = (state == FULL)          ? 1 : 0;

    assign state_out              = state;

endmodule

```

```

timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
//
// Company:
// Engineer: Chris Buenrostro
//
// Create Date: 12:38:25 05/04/06
// Design Name:
// Module Name: ADC_Interface
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// This module serves as the actual device interface for the ADC
// chips.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////
//
module ADC_Interface
(
// Global inputs
input global_clock,
input global_reset,
// Local inputs
input ADC_clock,
input ADC_request,
input ADC_intr_bar,
input [7:0] ADC_data_bus_in,
// outputs
output ADC_cs_bar,
output ADC_rd_bar,
output ADC_wr_bar,
output [7:0] ADC_data_bus_out,
output ADC_busy
);
/////////////////////////////////////////////////////////////////
//
//
//
// 1. DEFINITIONS
//
/////////////////////////////////////////////////////////////////
//
// Notes:
// 1)
//
/////////////////////////////////////////////////////////////////
//
// A. PARAMETERS
//
// STATE DEFINITIONS
//
parameter RESET = 0;
parameter INITIAL_ASSERT = 1;
parameter INITIAL_WAIT = 2;
parameter INITIAL_DE_ASSERT = 3;
parameter IDLE = 4;

```

```

        parameter CONV_WAIT    = 5;
        parameter READ_RD_ASSERT = 6;
        parameter READ_WAIT    = 7;
        parameter READ_RD_DE_ASSERT = 8;
        //
//      TIMING DEFINITIONS
        //
parameter wait_period    = 135135; // gives 5 hertz
        //
//      B.  REGISTERS
        //
        reg      ADC_rd_bar_int;
        reg [7:0] ADC_data_bus_int;
        reg [4:0] state, next;
        //
        reg [30:0] wait_counter;
        reg      ADC_request_int;
        //
//      C.  WIRES
        //
//
//      D.  INSTANTIATIONS
        //
//
////////////////////////////////////
//
////////////////////////////////////
        //
        //
        //
//      2.  BEHAVIORAL MODELING
        //
////////////////////////////////////
        //
        //      Notes:
        //      1)
        //
////////////////////////////////////
        //
//      STATE TRANSITION CALL
        //
        always @ (posedge global_clock)
        begin
            if (global_reset)
                state <= RESET;
            else
                state <= next;
            end
        //
        //      LATCH DATA
        //
        always @ (posedge global_clock)
        begin
            if (state == RESET)
                ADC_data_bus_int <= 0;
            else if (state == READ_RD_DE_ASSERT )
                ADC_data_bus_int <= ADC_data_bus_in;

            end
        //
//      SET CONTROL VARS
        //
        always @ (posedge global_clock)
        begin

```



```

//      ADC_intr_bar_int      <= ADC_intr_bar;
ADC_request_int      <= ADC_request;

        case (state)
            RESET:
ADC_rd_bar_int <= 1;
            INITIAL_ASSERT:
ADC_rd_bar_int <= 0;
            INITIAL_DE_ASSERT:
ADC_rd_bar_int <= 1;
            READ_RD_ASSERT:
ADC_rd_bar_int <= 0;
            READ_RD_DE_ASSERT:
ADC_rd_bar_int <= 1;
            default;;
        endcase
    end

//
//      WAIT COUNTERS
//
always @ (posedge global_clock)
    begin
        case (state)
            INITIAL_WAIT:
wait_counter <= wait_counter + 1;
            CONV_WAIT:
wait_counter <= wait_counter + 1;
            READ_WAIT:
wait_counter <= wait_counter + 1;
            default: wait_counter <= 1;
        endcase
    end

//
//      TRANSITION STEERING
//
always @ (state)
    begin
        case (state)
            RESET:
next = INITIAL_ASSERT;
            INITIAL_ASSERT:
next = INITIAL_WAIT;
            INITIAL_WAIT:
begin
if (wait_counter >= wait_period)
    next = INITIAL_DE_ASSERT;
            else
next = INITIAL_WAIT;
        end
            INITIAL_DE_ASSERT:
next = IDLE;
            IDLE:
begin
if (ADC_request_int)
next = CONV_WAIT;//WAIT_INTR;
            else
next = IDLE;
        end
//      WAIT_INTR:
//      begin
//      if (ADC_intr_bar_int == 0)
//      next = CONV_WAIT;
//      else
//      next = WAIT_INTR;

```

```
//
    CONV_WAIT:
        begin
            if (wait_counter >= wait_period)
                next = READ_RD_ASSERT;
            else
                next = CONV_WAIT;
            end
        end
        READ_RD_ASSERT:
            next = READ_WAIT;
        READ_WAIT:
            begin
                if (wait_counter >= wait_period)
                    next = READ_RD_DE_ASSERT;
                else
                    next = READ_WAIT;
                end
            end
        READ_RD_DE_ASSERT:
            next = IDLE;
            default;;
        endcase
    end

//
//
//
////////////////////////////////////
//
////////////////////////////////////
//
//
//
//      3. DATA-FLOW MODELING
//
////////////////////////////////////
//
//      Notes:
//          1)
//
////////////////////////////////////
//
//
assign ADC_cs_bar      = 1'b0;
assign ADC_rd_bar      = ADC_rd_bar_int;
assign ADC_wr_bar      = ADC_rd_bar;
assign ADC_data_bus_out = ADC_data_bus_int;
assign ADC_busy        = (state == IDLE) ? 0 : 1;
//
//
////////////////////////////////////
//
endmodule
```

```
`timescale 1ns / 1ps
```

```
module RAM_interface
(
// Inputs
    input      global_clock,
    input      global_reset,
// Inputs from User
    input      [16:0] ram_data_user,
    input      [10:0] ram_address_user,
    input      read_request,
    input      write_request,
// Inputs from RAM
    input      rfd,
    input      rdy,
    input      [16:0] return_data_RAM,
// Outputs to user
    output [16:0] ram_data_to_user,
    output      rfd_user,
    output      rdy_user,
// Outputs to RAM
    output [16:0] ram_data_RAM,
    output [10:0] ram_address_RAM,
    output      en,
    output      nd,
    output      we,

    output      RAM_busy,

    output      data_valid

);
//
////////////////////////////////////
//

// Registers
//
// Inputs
//
    reg      [16:0] ram_data_user_reg;
    reg      [10:0] ram_address_user_reg;
    reg      read_request_reg;
    reg      write_request_reg;
//
    reg      rfd_reg;
    reg      rdy_reg;
    reg      [16:0] return_data_RAM_reg;
//
// State_Machine
//
    reg      [2:0] state;
    reg      [2:0] next;
//
// Module Specific
```

```

//
    parameter RESET      = 0;
    parameter IDLE       = 1;
    parameter READ_ADDRESS = 2;
    parameter READ_DATA  = 3;
    parameter WRITE      = 4;
    parameter READ_VALID  = 5;
//
// Synchronized Component
//
always @ (posedge global_clock)
begin
    if (global_reset)
        state <= RESET;
    else
        state <= next;
end
//
always @ (posedge global_clock)
begin
    ram_data_user_reg    <= ram_data_user;
    ram_address_user_reg <= ram_address_user;
    read_request_reg     <= read_request;
    write_request_reg    <= write_request;
    rfd_reg             <= rfd;
    rdy_reg             <= rdy;
    return_data_RAM_reg <= return_data_RAM;
//
    if (global_reset)
    begin
        ram_data_user_reg    <= 0;
        ram_address_user_reg <= 0;
        read_request_reg     <= 0;
        write_request_reg    <= 0;
        rfd_reg             <= 0;
        rdy_reg             <= 0;
        return_data_RAM_reg <= 0;
    end
end
//
// Combinational always block
//
always @ (state)
begin
    case(state)
        RESET:
            next = IDLE;
        IDLE:
            begin
                if      (read_request_reg)
                    next = READ_ADDRESS;
                else if (write_request_reg)
                    next = WRITE;
                else
                    next = IDLE;
            end
        READ_ADDRESS:

```

```

        next = READ_DATA;
    READ_DATA:
        next = READ_VALID;
    READ_VALID:
        begin
            if      (read_request_reg)
                next = READ_DATA;
            else if (write_request_reg)
                next = WRITE;
            else
                next = IDLE;
            end
        end
    WRITE:
        begin
            if      (read_request_reg)
                next = READ_ADDRESS;
            else if (write_request_reg)
                next = WRITE;
            else
                next = IDLE;
            end
        end
    default:
        next = next;
    endcase
end

assign ram_data_to_user
    = (state == READ_DATA) ? return_data_RAM_reg :
      ram_data_to_user;
assign rfd_user      = rfd;
assign rdy_user      = rdy;
assign ram_data_RAM   = ram_data_user_reg;
assign ram_address_RAM = ram_address_user_reg;
assign en             = 1;
assign nd             = 1;
assign we = (state == WRITE) ? 1 : 0;
assign RAM_busy = (state == IDLE) ? 0 : 1;
assign data_valid = (state == READ_VALID) ? 1 : 0;

endmodule

```

\*\*\*\*\*

CODE BY ARCHANA VENKATARAMAN FOR DIGITAL CONTROL UNIT

\*\*\*\*\*

The code contains the following modules in the order listed below:

- [1] **Connect\_All** - Final module which instantiates and connects all other modules in the digital control unit
- [2] **DDMU\_top** - Top module which instantiates and connects all other modules in the Digital Decision-Making Unit (analyzes skin conductivity data and makes binary T/F decision)
- [3] **MajorFSM\_ddmu** - Keeps track of the data collection, processing, storage and/or decision sequence once sensor data has been obtained
- [4] **MinorFSM\_collect\_ddmu** - Obtains the raw sensor data from RAM, separates Conductivity and Pulse data, and pre-processes the Skin Conductivity
- [5] **fir\_cond** - 42-tap finite impulse response lowpass filter for the Skin Conductivity data
- [6] **MinorFSM\_process\_ddmu** - Instantiates and controls all the processing modules (which compute data statistics)
- [7] **fft\_pulse** - Takes a 2048 point FFT of the pulse data and determines the index with the maximum frequency content
- [8] **MinMax** - Computes the Minimum and Maximum value of the stream of Skin Conductivity data
- [9] **MaxDeriv** - Computes the maximum first different (magnitude-wise) of the Skin Conductivity data
- [10] **SumEnergy\_cond** - Computes the Sum and Energy (sum of absolute value) of the Skin Conductivity data
- [11] **SumEnergy\_pulse** - Computes the Sum of the Pulse data
- [12] **ThreshCount** - Computes the number of heart beats that occur during the sampling interval
- [13] **MinorFSM\_store** - Stores the computed statistics of Control questions in Data Register RAM (as one long memory row)
- [14] **MinorFSM\_decide** - Compares the computed statistics of Relevant questions with the statistics stored in Data Register (Control questions) to make T/F decision
- [15] **top\_FSM** - Instantiates the Major and Minor FSMs for the Memory Module
- [16] **MajorFSM\_mm** - Controls the sequence of reads and writes in the ping-pong BRAM configuration
- [17] **MinorFSM\_mm** - Given a start signal by the MajorFSM based on the processing state. It then sends the appropriate control signals to the RAM interface (for either a read or a write)
- [18] **UserCapt** - Captures the user-input reset, store, and question type and routes them to the other two blocks
- [19] **Synchronizer** - Instantiates "debounce" modules to debounce and to synchronize the user signals
- [20] **debounce, debounce2** - Code given by 6.111 staff to debounce 1-bit signals and modified code to debounce the 2-bit question type input
- [21] **UsrReg, UsrReg2** - Captures and registers the user signals until reset (similar to the Walk Register module in Lab 2)

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 12, 2006
// File: Connect_All.v
// Description: Module which connects the UserCapt, Memory Module, and DDMU

module Connect_All (clk, reset_in, question_in, store_in, done_RAM0,
    done_RAM1, data_valid0, data_valid1, address_in0,
    address_in1, data0, data1,

    reset, store, question, question_rst, store_rst,
    last_address0, read_out0, write_out0, tag_out0, address_out0,
    reset_interface0, busy0,
    last_address1, read_out1, write_out1, tag_out1, address_out1,
    reset_interface1, busy1,
    block0_RAM, block1_RAM, wait_RAM, wait_DDMU,
    block0_DDMU, block1_DDMU, get_next, liar,
    state_major_mm, next_major_mm, state_minor0_mm,
    next_minor0_mm,
    state_minor1_mm, next_minor1_mm, start_read0, start_read1,
    start_write0, start_write1, delay1, delay2,

    state_major_ddmu, next_major_ddmu, state_minor_coll,
    next_minor_coll, address_ddmu, comp, dout_cond, dout_pulse,

    start_coll, done_collect, proc_data1, data_ready,
    start_proc, done_proc, start_store, done_store, start_decide,
    done_decide, state_minor_proc, next_minor_proc,
    state_minor_store,
    next_minor_store, state_minor_decide, next_minor_decide,
    busy1_ddmu, busy2_ddmu, busy3_ddmu, busy4_ddmu, busy5_ddmu,
    busy6_ddmu, rdy_ddmu, rfd_ddmu, nd_ddmu, done0, reset_fir,
    index, block,

    Min_cond, Max_cond, Point1_cond,
    Point2_cond, MaxDeriv_cond, Sum_cond, Energy_cond,
    SumEnergy_pulse,
    pulse_count, max_index, max_out);

// Inputs and Outputs
input clk, reset_in, store_in, done_RAM0, done_RAM1, data_valid0, data_valid1;
input [1:0] question_in;
input [10:0] address_in0, address_in1;
input [16:0] data0, data1;

output reset, store;
output question;
output question_rst, store_rst;
output last_address0, read_out0, write_out0, tag_out0, reset_interface0,
busy0;
output last_address1, read_out1, write_out1, tag_out1, reset_interface1,
busy1;
output block0_RAM, block1_RAM;
output [10:0] address_out0, address_out1;
output wait_RAM, wait_DDMU;
output [2:0] state_major_mm, next_major_mm;
output [1:0] state_minor0_mm, next_minor0_mm, state_minor1_mm, next_minor1_mm;
output start_read0, start_read1, start_write0, start_write1;
output delay1, delay2;
output block0_DDMU, block1_DDMU, get_next, liar;
output [2:0] state_major_ddmu, next_major_ddmu;
output [1:0] state_minor_coll, next_minor_coll;
output [4:0] address_ddmu;

```

```

output [10:0] comp;
output signed [21:0] dout_cond;
output [7:0] dout_pulse;
output start_coll, done_collect, proc_data1, data_ready;
output start_proc, done_proc, start_store, done_store, start_decide,
done_decide;
output state_minor_proc, next_minor_proc, state_minor_store, next_minor_store;
output [1:0] state_minor_decide, next_minor_decide;
output busy1_ddmu, busy2_ddmu, busy3_ddmu, busy4_ddmu, busy5_ddmu, busy6_ddmu;
output rdy_ddmu, rfd_ddmu, nd_ddmu;
output done0, reset_fir, block;
output [10:0] index;

output signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
output signed [22:0] MaxDeriv_cond;
output signed [34:0] Sum_cond, Energy_cond;
output [19:0] SumEnergy_pulse;
output [11:0] pulse_count;
output signed [28:0] max_out;
output [10:0] max_index;

```

```

// Module Instantiations

```

```

wire reset, store;
wire [1:0] question;
wire question_rst, store_rst;

```

```

UserCapt1 user1 (
    .clk(clk),
    .reset(reset_in),
    .question_in(question_in),
    .store_in(store_in),
    .question_rst(question_rst),
    .store_rst(store_rst),
    .question_out(question),
    .store_out(store),
    .reset_sync(reset));

```

```

wire last_address0, read_out0, write_out0, tag_out0, reset_interface0, busy0;
wire last_address1, read_out1, write_out1, tag_out1, reset_interface1, busy1;
wire block0_RAM, block1_RAM;
wire [10:0] address_out0, address_out1;
wire wait_RAM, wait_DDMU;
wire [2:0] state_major_mm, next_major_mm;
wire [1:0] state_minor0_mm, next_minor0_mm, state_minor1_mm, next_minor1_mm;
wire start_read0, start_read1, start_write0, start_write1;
wire delay1, delay2;

```

```

top_FSM MemMod (
    .clk(clk),
    .reset(reset),
    .store(store),
    .question(question),
    .get_next(get_next),
    .done_RAM0(done_RAM0),
    .done_RAM1(done_RAM1),
    .block0_DDMU(block0_DDMU),
    .block1_DDMU(block1_DDMU),
    .last_address0(last_address0),
    .read_out0(read_out0),
    .write_out0(write_out0),
    .tag_out0(tag_out0),

```



```

        .address_out0(address_out0),
        .block0_RAM(block0_RAM),
        .block1_RAM(block1_RAM),
        .last_address1(last_address1),
        .read_out1(read_out1),
        .write_out1(write_out1),
        .tag_out1(tag_out1),
        .address_out1(address_out1),
        .wait_RAM(wait_RAM),
        .wait_DDMU(wait_DDMU),
        .store_rst(store_rst),
        .question_rst(question_rst),
        .address_in0(address_in0),
        .address_in1(address_in1),
        .reset_interface0(reset_interface0),
        .reset_interface1(reset_interface1),
        .state_major_mm(state_major_mm),
        .next_major_mm(next_major_mm),
        .state_minor0_mm(state_minor0_mm),
        .next_minor0_mm(next_minor0_mm),
        .busy0(busy0),
        .state_minor1_mm(state_minor1_mm),
        .next_minor1_mm(next_minor1_mm),
        .busyl(busyl),
        .start_read0(start_read0),
        .start_write0(start_write0),
        .start_read1(start_read1),
        .start_writel(start_writel),
        .delay1(delay1),
        .delay2(delay2));

wire block0_DDMU, block1_DDMU, get_next, liar;
wire [2:0] state_major_ddmu, next_major_ddmu;
wire [1:0] state_minor_coll, next_minor_coll;
wire [4:0] address_ddmu;
wire [10:0] comp;
wire signed [21:0] dout_cond;
wire [7:0] dout_pulse;
wire start_coll, done_collect, proc_data1, data_ready;
wire start_proc, done_proc, start_store, done_store, start_decide,
done_decide;
wire state_minor_proc, next_minor_proc, state_minor_store, next_minor_store;
wire [1:0] state_minor_decide, next_minor_decide;
wire busyl_ddmu, busy2_ddmu, busy3_ddmu, busy4_ddmu, busy5_ddmu, busy6_ddmu;
wire rdy_ddmu, rfd_ddmu, nd_ddmu;
wire done0, reset_fir, block;
wire [10:0] index;
wire signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
wire signed [22:0] MaxDeriv_cond;
wire signed [34:0] Sum_cond, Energy_cond;
wire [19:0] SumEnergy_pulse;
wire [11:0] pulse_count;
wire signed [28:0] max_out;
wire [10:0] max_index;

DDMU_top ddmu (
    .clk(clk),
    .reset(reset),
    .block0_RAM(block0_RAM),
    .block1_RAM(block1_RAM),
    .block0_DDMU(block0_DDMU),
    .block1_DDMU(block1_DDMU),
    .last_address0(last_address0),

```

```

.last_address1(last_address1),
.store(store),
.question(question),
.data0(data0),
.data1(data1),
.data_valid0(data_valid0),
.data_valid1(data_valid1),
.state_major(state_major_ddmu),
.next_major(next_major_ddmu),
.address(address_ddmu),
.liar(liar),
.comp(comp),
.get_next(get_next),
.dout_cond(dout_cond),
.dout_pulse(dout_pulse),
.state_minor1(state_minor_coll),
.next_minor1(next_minor_coll),
.start_coll(start_coll),
.done_collect(done_collect),
.proc_data1(proc_data1),
.data_ready(data_ready),
.start_proc(start_proc),
.done_proc(done_proc),
.start_store(start_store),
.done_store(done_store),
.start_decide(start_decide),
.done_decide(done_decide),
.state_minor2(state_minor_proc),
.next_minor2(next_minor_proc),
.state_minor3(state_minor_store),
.next_minor3(next_minor_store),
.state_minor4(state_minor_decide),
.next_minor4(next_minor_decide),
.busy1(busy1_ddmu),
.busy2(busy2_ddmu),
.busy3(busy3_ddmu),
.busy4(busy4_ddmu),
.busy5(busy5_ddmu),
.busy6(busy6_ddmu),
.rdy(rdy_ddmu),
.rfd(rfd_ddmu),
.nd(nd_ddmu),
.done0(done0),
.reset_fir(reset_fir),
.index(index),
.block(block),
.Min_cond(Min_cond),
.Max_cond(Max_cond),
.Point1_cond(Point1_cond),
.Point2_cond(Point2_cond),
.MaxDeriv_cond(MaxDeriv_cond),
.Sum_cond(Sum_cond),
.Energy_cond(Energy_cond),
.SumEnergy_pulse(SumEnergy_pulse),
.pulse_count(pulse_count),
.max_out(max_out),
.max_index(max_index));

```

```

endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 12, 2006
// File: DDMU_top.v
// Description: Top module which combines all Major and Minor FSMs

module DDMU_top (clk, reset, block0_RAM, block1_RAM, block0_DDMU, block1_DDMU,
    last_address0, last_address1, store, question, data0, data1,
    data_valid0, data_valid1, state_major, next_major, address,
    liar,
    comp, get_next, dout_cond, dout_pulse, state_minor1,
    next_minor1,

    start_coll, done_collect, proc_data1, data_ready, start_proc,
    done_proc, start_store, done_store, start_decide, done_decide,
    state_minor2, next_minor2, state_minor3, next_minor3,
    state_minor4, next_minor4, busy1, busy2, busy3, busy4, busy5,
    busy6, rdy, rfd, nd, done0, reset_fir, index, block,

    Min_cond, Max_cond, Point1_cond,
    Point2_cond, MaxDeriv_cond, Sum_cond, Energy_cond,
    SumEnergy_pulse,
    pulse_count, max_index, max_out);

// Inputs and Outputs
input clk, reset, block0_RAM, block1_RAM, last_address0, last_address1, store;
input data_valid0, data_valid1;
input [1:0] question;
input [16:0] data0, data1;

//output rfd_mem, rdy_mem;
output block0_DDMU, block1_DDMU, get_next;
output start_coll, done_collect, start_proc, done_proc, start_store,
done_store;
output start_decide, done_decide;
output proc_data1, data_ready;
output signed [21:0] dout_cond;
output [7:0] dout_pulse;
output [2:0] state_major, next_major;
output [1:0] state_minor1, next_minor1;
output state_minor2, next_minor2, state_minor3, next_minor3;
output signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
output signed [22:0] MaxDeriv_cond;
output signed [34:0] Sum_cond, Energy_cond;
output [19:0] SumEnergy_pulse;
output [11:0] pulse_count;
output [4:0] address;
output signed [28:0] max_out;
output [10:0] max_index;
output [1:0] state_minor4, next_minor4;
output liar;
output [10:0] comp;
output busy1, busy2, busy3, busy4, busy5, busy6;
output rdy, rfd, nd;
output done0, reset_fir, block;
output [10:0] index;

// Module Instantiations
wire start_coll, start_proc, start_store, start_decide;
wire done_collect, done_proc, done_store, done_decide;
wire proc_data1, tag, rdy, rfd, nd, get_next, data_ready;

```

```

wire signed [21:0] dout_cond;
wire [7:0] dout_pulse;
wire [2:0] state_major, next_major;
wire [1:0] state_minor1, next_minor1;
wire state_minor2, next_minor2, state_minor3, next_minor3;
wire signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
wire signed [22:0] MaxDeriv_cond;
wire signed [34:0] Sum_cond, Energy_cond;
wire [19:0] SumEnergy_pulse;
wire [11:0] pulse_count;
wire [4:0] address, address_store, address_decide;
wire nd1, nd2, nd_mem, we_mem, rdy_mem, rfd_mem;
wire signed [27:0] xreal_max, ximag_max;
wire signed [28:0] max_out;
wire [10:0] max_index;
wire signed [253:0] d_mem_in, d_mem_out;
wire [1:0] state_minor4, next_minor4;
wire liar;
wire reset_reg;
wire signed [8:0] din_cond;
wire [10:0] comp;
wire busy1, busy2, busy3, busy4, busy5, busy6;
wire done0, reset_fir, block;
wire [10:0] index;

assign nd_mem = (nd1 | nd2);
assign address = ((state_major == 5) ? (address_store) : (address_decide));

```

```

MajorFSM_ddmu major (.clk(clk),
    .reset(reset),
    .block0_RAM(block0_RAM),
    .block1_RAM(block1_RAM),
    .block0_DDMU(block0_DDMU),
    .block1_DDMU(block1_DDMU),
    .last_address0(last_address0),
    .proc_data1(proc_data1),
    .done_collect(done_collect),
    .done_proc(done_proc),
    .start_coll(start_coll),
    .start_proc(start_proc),
    .start_store(start_store),
    .start_decide(start_decide),
    .done_store(done_store),
    .done_decide(done_decide),
    .store(store),
    .question(question),
    .state(state_major),
    .next(next_major),
    .tag_type(tag));

```

```

MinorFSM_collect_ddmu minor1 (.clk(clk),
    .reset(reset),
    .last_address1(last_address1),
    .data0(data0),
    .data1(data1),
    .data_valid0(data_valid0),
    .data_valid1(data_valid1),
    .start(start_coll),
    .proc_data1(proc_data1),
    .dout_cond(dout_cond),
    .dout_pulse(dout_pulse),
    .done_collect(done_collect),
    .data_ready(data_ready),

```

```

.get_next(get_next),
.state(state_minor1),
.next(next_minor1),
.rdy(rdy),
.ready_for_data(rfd),
.new_data(nd),
.din_cond(din_cond),
.last_address0(last_address0),
.done0(done0),
.reset_fir(reset_fir),
.block(block));

```

```

MinorFSM_process_ddmu2 minor2 (.clk(clk),
    .reset(reset),
    .start_proc(start_proc),
    .din_cond(dout_cond),
    .din_pulse(dout_pulse),
    .done_collect(done_collect),
    .data_ready(data_ready),
    .Min_cond(Min_cond),
    .Max_cond(Max_cond),
    .MaxDeriv_cond(MaxDeriv_cond),
    .Point1_cond(Point1_cond),
    .Point2_cond(Point2_cond),
    .Sum_cond(Sum_cond),
    .Energy_cond(Energy_cond),
    .SumEnergy_pulse(SumEnergy_pulse),
    .pulse_count(pulse_count),
    .done_proc(done_proc),
    .state(state_minor2),
    .next(next_minor2),
    .xreal_max(xreal_max),
    .ximag_max(ximag_max),
    .max_index(max_index),
    .max_out(max_out),
    .reset_reg(reset_reg),
    .busy1(busy1),
    .busy2(busy2),
    .busy3(busy3),
    .busy4(busy4),
    .busy5(busy5),
    .busy6(busy6),
    .index(index));

```

```

MinorFSM_store_ddmu minor3 (.clk(clk),
    .reset(reset),
    .start(start_store),
    .Sum_cond(Sum_cond),
    .Energy_cond(Energy_cond),
    .MaxDeriv_cond(MaxDeriv_cond),
    .Point1_cond(Point1_cond),
    .Point2_cond(Point2_cond),
    .Min_cond(Min_cond),
    .Max_cond(Max_cond),
    .SumEnergy_pulse(SumEnergy_pulse),
    .pulse_count(pulse_count),
    .done_store(done_store),
    .address(address_store),
    .d_mem(d_mem_in),
    .nd(nd1),
    .rfd(rfd_mem),
    .we(we_mem),
    .tag(tag),
    .state(state_minor3),

```

```

        .next(next_minor3),
        .xreal_max(xreal_max),
        .ximag_max(ximag_max),
        .max_out(max_out),
        .max_index(max_index));

    bram_all_final data_reg (.addr(address),
        .clk(clk),
        .din(d_mem_in),
        .dout(d_mem_out),
        .nd(nd_mem),
        .we(we_mem),
        .rdy(rdy_mem),
        .rfd(rfd_mem));

    MinorFSM_decide_ddmu minor4 (.clk(clk),
        .reset(reset),
        .start(start_decide),
        .address_in(address_store),
        .Sum_cond(Sum_cond),
        .Energy_cond(Energy_cond),
        .MaxDeriv_cond(MaxDeriv_cond),
        .Point1_cond(Point1_cond),
        .Point2_cond(Point2_cond),
        .Max_cond(Max_cond),
        .Min_cond(Min_cond),
        .SumEnergy_pulse(SumEnergy_pulse),
        .pulse_count(pulse_count),
        .max_out(max_out),
        .max_index(max_index),
        .done(done_decide),
        .liar(liar),
        .d_mem(d_mem_out),
        .nd(nd2),
        .rdy(rdy_mem),
        .address_out(address_decide),
        .state(state_minor4),
        .next(next_minor4),
        .comp(comp));

endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 1, 2006
// File: MajorFSM_ddmu.v
// Description: Major FSM for the Digital Decision-Making Unit

module MajorFSM_ddmu (clk, reset, block0_RAM, block1_RAM, block0_DDMU,
                     block1_DDMU,
                     last_address0, proc_data1, done_collect, done_proc,
                     start_coll, start_proc, start_store, start_decide,
                     done_store,
                     done_decide, store, question, state, next, tag_type);

// Inputs and Outputs
input clk, reset, block0_RAM, block1_RAM, last_address0, store;
input done_collect, done_proc, done_store, done_decide;
input [1:0] question;

output start_coll, start_proc, start_store, start_decide, block0_DDMU;
output block1_DDMU, proc_data1;
reg start_coll, start_proc, start_store, start_decide, block0_DDMU;
reg block1_DDMU, proc_data1;
output [2:0] state, next;
output tag_type;

// Internal Signals
reg [2:0] state, next;
reg tag_type, switch_block, set_switch, reset_switch;
reg start_coll_int, start_proc_int, start_store_int, start_dec_int;
reg set_block0, set_block1, reset_block0, reset_block1;

// Parameters
parameter IDLE = 0;
parameter WAITING = 1;
parameter GET_DATA0 = 2;
parameter GET_DATA1 = 3;
parameter PROCESS = 4;
parameter STORE = 5;
parameter DECIDE = 6;

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            start_coll <= 0;
            start_proc <= 0;
            start_decide <= 0;
            start_store <= 0;
            block0_DDMU <= 0;
            block1_DDMU <= 0;
            switch_block <= 0;
            proc_data1 <= 0;
            tag_type <= 0;
        end
    else
        begin

```

```

state <= next;
start_coll <= start_coll_int;
start_proc <= start_proc_int;
start_decide <= start_dec_int;
start_store <= start_store_int;

if (store && (question == 1) && (state == IDLE))
    tag_type <= 0;
else if (store && (question == 2) && (state == IDLE))
    tag_type <= 1;
else
    tag_type <= tag_type;

if (switch_block && ~block1_RAM)
    proc_data1 <= 1;
else
    proc_data1 <= 0;

if (set_block0)
    block0_DDMU <= 1;
else if (reset_block0)
    block0_DDMU <= 0;
else
    block0_DDMU <= block0_DDMU;

if (set_block1)
    block1_DDMU <= 1;
else if (reset_block1)
    block1_DDMU <= 0;
else
    block1_DDMU <= block1_DDMU;

if (set_switch)
    switch_block <= 1;
else if (reset_switch)
    switch_block <= 0;
else
    switch_block <= switch_block;

end

// Combinational Block to set Next State and Internal Variables
always @ (done_collect or done_proc or done_store or done_decide or state or
store or
    question or block0_RAM or last_address0 or switch_block or
    block1_RAM or tag_type)
begin

    start_coll_int = 0;
    start_proc_int = 0;
    start_store_int = 0;
    start_dec_int = 0;
    set_block0 = 0;
    set_block1 = 0;
    reset_block0 = 0;
    reset_block1 = 0;
    set_switch = 0;
    reset_switch = 0;

    case (state)

        IDLE:

```





```

        default:
            endcase
        end
    endmodule
endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 1, 2006
// File: MinorFSM_collect_ddmu.v
// Description: Minor FSM to collect and pre-process data for the DDMU
//             Skin Conductivity will be filtered - should be signed output
//             Pulse is not filtered - just 8 bits unsigned

module MinorFSM_collect_ddmu (clk, reset, last_address1, data0, data1,
                             data_valid0,
                             data_valid1, start, proc_data1, dout_cond, dout_pulse,
                             done_collect, data_ready, get_next, state, next, rdy,
                             ready_for_data, new_data, din_cond, reset_fir,
                             last_address0, done0, block);

// Inputs and Outputs
input clk, reset, last_address1, data_valid0, data_valid1, last_address0;
input start, proc_data1;
input [16:0] data0, data1;

output signed [21:0] dout_cond;
output signed [7:0] dout_pulse;
output signed [8:0] din_cond;
reg signed [21:0] dout_cond;
reg signed [7:0] dout_pulse;
output done_collect, data_ready, get_next;
reg done_collect, data_ready, get_next;
output [1:0] state, next;
output rdy, ready_for_data, new_data;
output reset_fir;
output done0, block;

// Internal Signals
reg [1:0] state, next;
reg block, set_block, reset_block;
reg last, set_last, reset_last;
reg sample_data;
reg done_collect_int, data_ready_int, get_next_int, done_collect2;
reg done0, set_done0, reset_done0;
reg done_int, set_done_int, reset_done_int;

reg set_nd;
wire new_data, rdy, ready_for_data, reset_fir;
wire signed [21:0] dout_cond_int;
wire signed [8:0] din_cond;

// Parameters
parameter IDLE = 0;
parameter WAIT_RAM = 1;
parameter WAIT_RDY = 2;

// Module Instantiation

assign din_cond = (block ? {1'b0, data1[15:8]} : {1'b0, data0[15:8]});
assign reset_fir = (reset | done_collect | done_collect2);
assign new_data = set_nd;

fir_cond fir (.CLK(clk),
              .RESET(reset_fir),
              .ND(new_data),

```

```

        .DIN(din_cond),
        .RDY(rdy),
        .RFD(ready_for_data),
        .DOUT(dout_cond_int));

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            dout_cond <= 0;
            dout_pulse <= 0;
            done_collect <= 0;
            data_ready <= 0;
            get_next <= 0;
            block <= 0;
            last <= 0;
            done_collect2 <= 0;
            done_int <= 0;
            done0 <= 0;
        end
    else
        begin
            state <= next;

            if (set_done0)
                done0 <= 1;
            else if (reset_done0)
                done0 <= 0;
            else
                done0 <= done0;

            if (set_done_int)
                done_int <= 1;
            else if (reset_done_int)
                done_int <= 0;
            else
                done_int <= done_int;

            if (sample_data)
                begin
                    dout_cond <= dout_cond_int;
                end
            else
                begin
                    dout_cond <= dout_cond;
                end

            if (block && data_valid1)
                dout_pulse <= data1[7:0];
            else if (~block && data_valid0)
                dout_pulse <= data0[7:0];
            else
                dout_pulse <= dout_pulse;

            done_collect2 <= done_collect_int;
            done_collect <= done_collect2;
            data_ready <= data_ready_int;
            get_next <= get_next_int;

            if (set_block)

```

```

        block <= 1;
    else if (reset_block)
        block <= 0;
    else
        block <= block;

    if (set_last)
        last <= 1;
    else if (reset_last)
        last <= 0;
    else
        last <= last;
end

// Combinational Block to set Next State and Internal Variables
always @ (state or last_address1 or data0 or data1 or data_valid0 or
data_valid1 or
start or proc_data1 or block or ready_for_data or rdy or last or
done0
or last_address0 or done_int)
begin

    done_collect_int = 0;
    data_ready_int = 0;
    get_next_int = 0;
    reset_block = 0;
    reset_last = 0;
    sample_data = 0;
    set_nd = 0;
    reset_done0 = 0;
    reset_done_int = 0;

    if (last_address0 && (block == 0))
        begin
            set_done0 = 1;
            set_done_int = 1;
        end
    else
        begin
            set_done0 = 0;
            set_done_int = 0;
        end

    if (proc_data1)
        set_block = 1;
    else
        set_block = 0;

    if (last_address1 && (block == 1))
        set_last = 1;
    else
        set_last = 0;

    case (state)

        IDLE:
            if (start)
                begin
                    next = WAIT_RAM;
                    get_next_int = 1;
                end
            else
                begin

```

```

        next = IDLE;
    end

WAIT_RAM:
    if (block == 0)
        begin
            if (data_valid0 && ready_for_data)
                begin
                    if (done0)
                        begin
                            if (done_int)
                                begin
                                    data_ready_int = 1;
                                    reset_done_int = 1;
                                end
                            next = WAIT_RAM;
                        end
                    else
                        begin
                            next = WAIT_RDY;
                            set_nd = 1;
                        end
                    end
                end
            else
                next = WAIT_RAM;
            end
        end
    else
        begin
            if (done0)
                begin
                    reset_done0 = 1;
                    get_next_int = 1;
                    next = WAIT_RAM;
                end
            if (data_valid1 && ready_for_data)
                begin
                    next = WAIT_RDY;
                    set_nd = 1;
                end
            else
                next = WAIT_RAM;
            end
        end
    end

WAIT_RDY:
    if (rdy)
        begin
            if (last)
                begin
                    next = IDLE;
                    done_collect_int = 1;
                    data_ready_int = 1;
                    reset_block = 1;
                    reset_last = 1;
                    sample_data = 1;
                end
            else
                begin
                    next = WAIT_RAM;

                    if (~done0)
                        get_next_int = 1;

                    sample_data = 1;
                    data_ready_int = 1;
                end
            end
        end
    end

```

```

        end
    else
        next = WAIT_RDY;
    default:
        next = IDLE;
    endcase
end
endmodule
```

```
// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 13, 2006
// File: fir_cond.v
// Description: FIR to lowpass filter the skin conductivity data
```

```
module fir_cond (CLK, RESET, ND, DIN, RDY, RFD, DOUT);
```

```
// Inputs and Outputs
input CLK, RESET, ND;
input signed [8:0] DIN;

output RDY, RFD;
output signed [21:0] DOUT;
reg RDY, RFD;
reg signed [21:0] DOUT;
```

```
// Internal Signals
reg signed [8:0] x[41:0];
reg state, next;
reg [21:0] dout_int;
reg set_output, set_data;
reg second, set_second, reset_second;
reg set_rdy, reset_rdy, set_rfd, reset_rfd;
```

```
// Parameters
parameter signed a0 = 1;
parameter signed a1 = 0;
parameter signed a2 = -1;
parameter signed a3 = -1;
parameter signed a4 = 1;
parameter signed a5 = 2;
parameter signed a6 = 0;
parameter signed a7 = -2;
parameter signed a8 = -1;
parameter signed a9 = 2;
parameter signed a10 = 3;
parameter signed a11 = -1;
parameter signed a12 = -4;
parameter signed a13 = -1;
parameter signed a14 = 4;
parameter signed a15 = 4;
parameter signed a16 = -3;
parameter signed a17 = -9;
parameter signed a18 = -1;
parameter signed a19 = 19;
parameter signed a20 = 38;
parameter signed a21 = 38;
parameter signed a22 = 19;
parameter signed a23 = -1;
parameter signed a24 = -9;
parameter signed a25 = -3;
parameter signed a26 = 4;
parameter signed a27 = 4;
parameter signed a28 = -1;
parameter signed a29 = -4;
parameter signed a30 = -1;
parameter signed a31 = 3;
parameter signed a32 = 2;
parameter signed a33 = -1;
parameter signed a34 = -2;
```



```

parameter signed a35 = 0;
parameter signed a36 = 2;
parameter signed a37 = 1;
parameter signed a38 = -1;
parameter signed a39 = -1;
parameter signed a40 = 0;
parameter signed a41 = 1;

parameter IDLE = 0;
parameter COMPUTE = 1;

// Sequential Block to set State and Internal Variables
always @ (posedge CLK)
begin
    if (RESET)
    begin
        state <= IDLE;
        DOUT <= 0;
        RFD <= 1;
        RDY <= 0;
        x[0] <= 0;
        x[1] <= 0;
        x[2] <= 0;
        x[3] <= 0;
        x[4] <= 0;
        x[5] <= 0;
        x[6] <= 0;
        x[7] <= 0;
        x[8] <= 0;
        x[9] <= 0;
        x[10] <= 0;
        x[11] <= 0;
        x[12] <= 0;
        x[13] <= 0;
        x[14] <= 0;
        x[15] <= 0;
        x[16] <= 0;
        x[17] <= 0;
        x[18] <= 0;
        x[19] <= 0;
        x[20] <= 0;
        x[21] <= 0;
        x[22] <= 0;
        x[23] <= 0;
        x[24] <= 0;
        x[25] <= 0;
        x[26] <= 0;
        x[27] <= 0;
        x[28] <= 0;
        x[29] <= 0;
        x[30] <= 0;
        x[31] <= 0;
        x[32] <= 0;
        x[33] <= 0;
        x[34] <= 0;
        x[35] <= 0;
        x[36] <= 0;
        x[37] <= 0;
        x[38] <= 0;
        x[39] <= 0;
        x[40] <= 0;
        x[41] <= 0;
    end
end

```

```

else
begin
    state <= next;

    if (set_rfd)
        RFD <= 1;
    else if (reset_rfd)
        RFD <= 0;
    else
        RFD <= RFD;

    if (set_rdy)
        RDY <= 1;
    else if (reset_rdy)
        RDY <= 0;
    else
        RDY <= RDY;

    if (set_output)
        DOUT <= dout_int;
    else
        DOUT <= DOUT;

    if (set_data)
    begin
        x[0] <= DIN;
        x[1] <= x[0];
        x[2] <= x[1];
        x[3] <= x[2];
        x[4] <= x[3];
        x[5] <= x[4];
        x[6] <= x[5];
        x[7] <= x[6];
        x[8] <= x[7];
        x[9] <= x[8];
        x[10] <= x[9];
        x[11] <= x[10];
        x[12] <= x[11];
        x[13] <= x[12];
        x[14] <= x[13];
        x[15] <= x[14];
        x[16] <= x[15];
        x[17] <= x[16];
        x[18] <= x[17];
        x[19] <= x[18];
        x[20] <= x[19];
        x[21] <= x[20];
        x[22] <= x[21];
        x[23] <= x[22];
        x[24] <= x[23];
        x[25] <= x[24];
        x[26] <= x[25];
        x[27] <= x[26];
        x[28] <= x[27];
        x[29] <= x[28];
        x[30] <= x[29];
        x[31] <= x[30];
        x[32] <= x[31];
        x[33] <= x[32];
        x[34] <= x[33];
        x[35] <= x[34];
        x[36] <= x[35];
        x[37] <= x[36];
    end
end

```

```

        x[38] <= x[37];
        x[39] <= x[38];
        x[40] <= x[39];
        x[41] <= x[40];
    end
end
end

```

```

// Combinational Block to set Next and Internal Variables
always @ (state or ND or second)
begin

```

```

    set_second = 0;
    reset_second = 0;
    set_output = 0;
    set_data = 0;
    set_rdy = 0;
    reset_rdy = 0;
    set_rfd = 0;
    reset_rfd = 0;
    dout_int = 0;

```

```

    case (state)

```

```

        IDLE:
        if (ND)
            begin
                set_data = 1;
                next = COMPUTE;
                reset_rfd = 1;
                reset_rdy = 1;
            end
        else
            begin
                next = IDLE;
            end

```

```

        COMPUTE:
        begin
            dout_int = a0*(x[0] + x[41]) +
                a1*(x[1] + x[40]) +
                a2*(x[2] + x[39]) +
                a3*(x[3] + x[38]) +
                a4*(x[4] + x[37]) +
                a5*(x[5] + x[36]) +
                a6*(x[6] + x[35]) +
                a7*(x[7] + x[34]) +
                a8*(x[8] + x[33]) +
                a9*(x[9] + x[32]) +
                a10*(x[10] + x[31]) +
                a11*(x[11] + x[30]) +
                a12*(x[12] + x[29]) +
                a13*(x[13] + x[28]) +
                a14*(x[14] + x[27]) +
                a15*(x[15] + x[26]) +
                a16*(x[16] + x[25]) +
                a17*(x[17] + x[24]) +
                a18*(x[18] + x[23]) +
                a19*(x[19] + x[22]) +
                a20*(x[20] + x[21]);

```

```

            set_output = 1;
            set_rdy = 1;

```

```
        set_rfd = 1;
        next = IDLE;
    end

    default:
        begin
            next = IDLE;
            reset_rdy = 1;
            set_rfd = 1;
        end
    endcase
end
endmodule
```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 5, 2006
// File: MinorFSM_process_ddmu2.v
// Description: Minor FSM to process data once it has been filtered

module MinorFSM_process_ddmu2 (clk, reset, start_proc, din_cond, din_pulse,
                                done_collect, data_ready, Min_cond, Max_cond,
                                MaxDeriv_cond, Point1_cond, Point2_cond, Sum_cond,
                                Energy_cond, SumEnergy_pulse, pulse_count,
                                done_proc, state, next, xreal_max, ximag_max,
                                max_out, max_index, reset_reg, index,
                                busy1, busy2, busy3, busy4, busy5, busy6);

// Inputs and Outputs

input clk, reset, start_proc, done_collect, data_ready;
input [21:0] din_cond;
input [7:0] din_pulse;

output signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
output signed [22:0] MaxDeriv_cond;
output signed [34:0] Sum_cond, Energy_cond;
output [19:0] SumEnergy_pulse;
output [11:0] pulse_count;
output state, next, done_proc;
output signed [27:0] xreal_max, ximag_max;
output signed [28:0] max_out;
output [10:0] max_index;
output reset_reg;
output [10:0] index;

reg signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
reg signed [22:0] MaxDeriv_cond;
reg signed [34:0] Sum_cond, Energy_cond;
reg [19:0] SumEnergy_pulse;
reg [11:0] pulse_count;
reg state, next, done_proc;
reg signed [27:0] xreal_max, ximag_max;
reg signed [28:0] max_out;
reg [10:0] max_index;
output busy1, busy2, busy3, busy4, busy5, busy6;

// Internal Signals
reg done_proc_int, set_output, reset_reg, reset_reg_int;

wire [21:0] Min_cond_int, Max_cond_int, Point1_cond_int, Point2_cond_int;
wire [22:0] MaxDeriv_cond_int;
wire [34:0] Sum_cond_int, Energy_cond_int;
wire [19:0] SumEnergy_pulse_int;
wire [11:0] pulse_count_int;
wire signed [27:0] xreal_max_int, ximag_max_int;
wire signed [28:0] max_out_int;
wire [10:0] max_index_int;

// Parameters
parameter IDLE = 0;
parameter PROCESS = 1;

// Module Instantiations

```

```

wire busy1, busy2, busy3, busy4, busy5, busy6;
wire statel, next1, state2, next2, state3, next3, state4, next4;
wire [1:0] state5, next5;
wire [2:0] state6, next6;
wire start_fft, done_fft;
wire [10:0] index;

```

```

MinMax proc1 (.clk(clk),
              .reset(reset),
              .din_cond(din_cond),
              .done_collect(done_collect),
              .data_ready(data_ready),
              .start(start_proc),
              .busy(busy1),
              .Minimum(Min_cond_int),
              .Maximum(Max_cond_int),
              .reset_minmax(reset_reg),
              .state(statel),
              .next(next1));

```

```

MaxDeriv proc2 (.clk(clk),
               .reset(reset),
               .din_cond(din_cond),
               .done_collect(done_collect),
               .data_ready(data_ready),
               .start(start_proc),
               .busy(busy2),
               .Max_Derivative(MaxDeriv_cond_int),
               .Point1(Point1_cond_int),
               .Point2(Point2_cond_int),
               .state(state2),
               .next(next2),
               .reset_md(reset_reg));

```

```

SumEnergy_cond proc3 (.clk(clk),
                     .reset(reset),
                     .din_cond(din_cond),
                     .done_collect(done_collect),
                     .data_ready(data_ready),
                     .start(start_proc),
                     .busy(busy3),
                     .Sum(Sum_cond_int),
                     .Energy(Energy_cond_int),
                     .state(state3),
                     .next(next3),
                     .reset_se(reset_reg));

```

```

SumEnergy_pulse proc4 (.clk(clk),
                      .reset(reset),
                      .din_pulse(din_pulse),
                      .done_collect(done_collect),
                      .data_ready(data_ready),
                      .start(start_proc),
                      .busy(busy4),
                      .SumEnergy(SumEnergy_pulse_int),
                      .state(state4),
                      .next(next4),
                      .reset_se(reset_reg));

```

```

ThreshCount2 proc5 (.clk(clk),
                   .reset(reset),
                   .data_ready(data_ready),
                   .done_collect(done_collect),

```

```

        .start(start_proc),
        .din_pulse(din_pulse),
        .pulse_count(pulse_count_int),
        .busy(busy5),
        .state(state5),
        .next(next5),
        .reset_tc(reset_reg));

fft_pulse4 fftcomp (.clk(clk),
        .reset(reset),
        .start(start_proc),
        .data_ready(data_ready),
        .reset_fft(reset_reg),
        .din_pulse(din_pulse),
        .busy(busy6),
        .xreal_max(xreal_max_int),
        .ximag_max(ximag_max_int),
        .max_index(max_index_int),
        .state(state6),
        .next(next6),
        .start_fft(start_fft),
        .done_fft(done_fft),
        .max_out(max_out_int),
        .done_collect(done_collect),
        .index(index));

```

```

// Sequential Block to set State and Registers
always @ (posedge clk)

```

```

    if (reset)
        begin
            state <= IDLE;
            Min_cond <= 0;
            Max_cond <= 0;
            Point1_cond <= 0;
            Point2_cond <= 0;
            MaxDeriv_cond <= 0;
            Sum_cond <= 0;
            Energy_cond <= 0;
            SumEnergy_pulse <= 0;
            pulse_count <= 0;
            xreal_max <= 0;
            ximag_max <= 0;
            max_out <= 0;
            max_index <= 0;
            done_proc <= 0;
            reset_reg <= 0;
        end

    else
        begin
            state <= next;
            done_proc <= done_proc_int;
            reset_reg <= reset_reg_int;

            if (set_output)
                begin
                    Min_cond <= Min_cond_int;
                    Max_cond <= Max_cond_int;
                    Point1_cond <= Point1_cond_int;
                    Point2_cond <= Point2_cond_int;
                    MaxDeriv_cond <= MaxDeriv_cond_int;
                    Sum_cond <= Sum_cond_int;
                end
        end

```

```

        Energy_cond <= Energy_cond_int;
        SumEnergy_pulse <= SumEnergy_pulse_int;
        pulse_count <= pulse_count_int;
        xreal_max <= xreal_max_int;
        ximag_max <= ximag_max_int;
        max_out <= max_out_int;
        max_index <= max_index_int;
    end
else
    begin
        Min_cond <= Min_cond;
        Max_cond <= Max_cond;
        Point1_cond <= Point1_cond;
        Point2_cond <= Point2_cond;
        MaxDeriv_cond <= MaxDeriv_cond;
        Sum_cond <= Sum_cond;
        Energy_cond <= Energy_cond;
        SumEnergy_pulse <= SumEnergy_pulse;
        pulse_count <= pulse_count;
        xreal_max <= xreal_max;
        ximag_max <= ximag_max;
        max_out <= max_out;
        max_index <= max_index;
    end
end

// Combinational Block to set Next State and Internal Variables
always @ (start_proc or state or busy1 or busy2 or busy3 or busy4 or busy5 or
busy6)
begin
    done_proc_int = 0;
    reset_reg_int = 0;
    set_output = 0;

    case (state)
        IDLE:
            if (start_proc)
                next = PROCESS;
            else
                begin
                    next = IDLE;
                    reset_reg_int = 1;
                end
            PROCESS:
                if (busy1 || busy2 || busy3 || busy4 || busy5 || busy6)
                    next =
PROCESS;
                else
                    begin
                        next = IDLE;
                        done_proc_int = 1;
                        set_output = 1;
                    end
                    default:
                        next = IDLE;
                    endcase
    end
endmodule

```



```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 8, 2006
// File: fft_pulse4.v
// Description: Module to calculate the Fast Fourier Transform of the pulse //
data and output the maximum index

module fft_pulse4 (clk, reset, start, data_ready, reset_fft, din_pulse,
                  busy, xreal_max, ximag_max, max_index, state, next,
                  start_fft, done_fft, max_out, done_collect, index);

// Inputs and Outputs
input clk, reset, start, data_ready, reset_fft, done_collect;
input [7:0] din_pulse;

output busy, start_fft, done_fft;
output [2:0] state, next;
reg busy;

output signed [27:0] xreal_max, ximag_max;
output signed [28:0] max_out;
output [10:0] max_index;
reg signed [27:0] xreal_max, ximag_max;
reg [10:0] max_index, max_index_int;
reg signed [28:0] max_out;

output [10:0] index;

// Internal Signals
reg [2:0] state, next;
reg [10:0] index;
reg inc_index, reset_index;
reg finished, set_finished, reset_finished, inc, set_inc, reset_inc;
reg set_max;

// Parameters
parameter IDLE = 0;
parameter COLLECT_DATA = 1;
parameter FFT = 2;
parameter WAIT_FFT = 3;
parameter PROCESS = 4;

parameter MAX_INDEX = 2047;

// Module Instantiation (memory)
wire we_mem, nd_mem, rfd_mem, rdy_mem;
wire [7:0] din_mem, dout_mem;
wire [10:0] address_mem;

reg [7:0] din_mem_int;
reg set_mem;
reg nd, nd_int, we, we_int;

assign din_mem = din_mem_int;
assign nd_mem = nd;
assign address_mem = index;
assign we_mem = we;

bram_fft2 bram (.addr(address_mem),
               .clk(clk),

```

```

        .din(din_mem),
        .dout(dout_mem),
        .nd(nd_mem),
        .rfd(rfd_mem),
        .rdy(rdy_mem),
        .we(we_mem));

// Module Instantiation (fft)
wire start_fft, rfd_fft, dv_fft, edone_fft, done_fft;
wire signed [27:0] xreal_fft, ximag_fft;

wire [15:0] dfft_in;
wire [10:0] index_in, index_out;

reg start_fft_int, start_fft_reg;
reg signed [27:0] xreal_int, ximag_int, temp1, temp2, xreal_abs, ximag_abs;

reg [7:0] dfft_dell, dfft_int, dfft;
assign dfft_in = {8'd0, dfft};
assign start_fft = start_fft_reg;

fft_2048 fft (.xn_re(dfft_in),
             .xn_im(16'd0),
             .start(start_fft),
             .fwd_inv(1'b1),
             .fwd_inv_we(1'b0),
             .sclr(reset),
             .clk(clk),
             .xk_re(xreal_fft),
             .xk_im(ximag_fft),
             .xn_index(index_in),
             .xk_index(index_out),
             .rfd(rfd_fft),
             .dv(dv_fft),
             .edone(edone_fft),
             .done(done_fft));

// Combinational Block to set absolute value
always @ (xreal_fft or ximag_fft)
begin
    if (xreal_fft < 0)
        begin
            temp1 = ~xreal_fft + 1;
            xreal_abs = temp1;
        end
    else
        xreal_abs = xreal_fft;

    if (ximag_fft < 0)
        begin
            temp2 = ~ximag_fft + 1;
            ximag_abs = temp2;
        end
    else
        ximag_abs = ximag_fft;
end

// Sequential Block to set State and Registers
always @ (posedge clk)
begin

```

```

if (reset)
begin
    state <= IDLE;
    index <= 0;
    dfft_dell <= 0;
    dfft <= 0;
    start_fft_reg <= 0;
    din_mem_int <= 0;
    finished <= 0;
    inc <= 0;
    max_index <= 0;
    xreal_max <= 0;
    ximag_max <= 0;
end

else
begin
    state <= next;
    start_fft_reg <= start_fft_int;
    nd <= nd_int;
    we <= we_int;
    dfft_dell <= dfft_int;
    dfft <= dfft_dell;

    if (set_finished)
        finished <= 1;
    else if (reset_finished)
        finished <= 0;
    else
        finished <= finished;

    if (set_inc)
        inc <= 1;
    else if (reset_inc)
        inc <= 0;
    else
        inc <= inc;

    if (inc_index)
        index <= index + 1;
    else if (reset_index)
        index <= 0;
    else
        index <= index;

    if (set_mem)
        din_mem_int <= din_pulse;
    else
        din_mem_int <= din_mem_int;

    if (set_max)
    begin
        xreal_max <= xreal_int;
        ximag_max <= ximag_int;
        max_out <= xreal_abs + ximag_abs;
        max_index <= max_index_int;
    end
    else if (reset_fft)
    begin
        xreal_max <= 0;
        ximag_max <= 0;
        max_out <= 0;
        max_index <= 0;
    end
end

```

```

        else
            begin
                xreal_max <= xreal_max;
                ximag_max <= ximag_max;
                max_out <= max_out;
                max_index <= max_index;
            end
        end
    end

end

// Combinational Block to set Next state and Internal Variables
always @ (start or state or index or data_ready or finished or inc
        or dout_mem or done_fft or index_out or xreal_abs or
        ximag_abs or max_out or xreal_fft or ximag_fft or done_collect)
begin

    start_fft_int = 0;
    nd_int = 0;
    we_int = 0;
    inc_index = 0;
    reset_index = 0;
    set_mem = 0;
    set_finished = 0;
    reset_finished = 0;
    set_inc = 0;
    reset_inc = 0;
    dfft_int = 0;
    xreal_int = 0;
    ximag_int = 0;
    max_index_int = 0;
    set_max = 0;

    case (state)

        IDLE:
            if (start)
                begin
                    next = COLLECT_DATA;
                    busy = 1;
                end
            else
                begin
                    next = IDLE;
                    busy = 0;
                end
            end

        COLLECT_DATA:
            begin
                busy = 1;

                if (data_ready)
                    begin
                        nd_int = 1;

                        if (index >= MAX_INDEX)
                            begin
                                we_int = 1;
                                next = COLLECT_DATA;
                                set_finished = 1;
                                set_mem = 1;
                            end
                        else
                            begin

```

```

        next = COLLECT_DATA;
        set_inc = 1;
        we_int = 1;
        set_mem = 1;
    end
end

else
begin
    if (finished)
    begin
        next = FFT;
        reset_index = 1;
        reset_finished = 1;
        start_fft_int = 1;
    end
    else if (inc)
    begin
        next = COLLECT_DATA;
        reset_inc = 1;
        inc_index = 1;
    end
    else
        next = COLLECT_DATA;
    end
end

FFT:
begin
    busy = 1;
    if (inc)
    begin
        dfft_int = dout_mem;
        if (index >= MAX_INDEX)
        begin
            next = WAIT_FFT;
            reset_index = 1;
        end
        else
        begin
            next = FFT;
            inc_index = 1;
        end
    end
    else
    begin
        next = FFT;
        set_inc = 1;
    end
end

WAIT_FFT:
begin
    busy = 1;

    if (inc)
    begin
        dfft_int = dout_mem;
        reset_inc = 1;
    end

    if (done_fft)
        next = PROCESS;
    else
        next = WAIT_FFT;
end

```

```

        end

PROCESS:
    begin
        busy = 1;

        if (index_out <= MAX_INDEX)
            begin
                if ((xreal_abs + ximag_abs) > max_out)
                    begin
                        if (index_out != 0)
                            set_max = 1;
                        xreal_int = xreal_fft;
                        ximag_int = ximag_fft;
                        max_index_int = index_out;
                    end
                end

                if (index_out < MAX_INDEX)
                    next = PROCESS;
                else
                    next = IDLE;
                end
            end
        default:
            begin
                next = IDLE;
                busy = 0;
            end
        endcase
    end
endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 1, 2006
// File: MaxDeriv.v
// Description: Module to calculate the maximum derivative (magnitude)

module MaxDeriv (clk, reset, din_cond, done_collect, data_ready,
                 start, busy, Max_Derivative, Point1, Point2,
                 state, next, reset_md);

// Inputs and Outputs
input clk, reset, done_collect, data_ready, start, reset_md;
input signed [21:0] din_cond;

output busy;
reg busy;
output signed [22:0] Max_Derivative;
reg signed [22:0] Max_Derivative;

output signed [21:0] Point1, Point2;
reg signed [21:0] Point1, Point2;
output state, next;

// Internal Signals
reg state, next, set, set_prev;
reg signed [21:0] Point1_int, Point2_int, Previous;
reg signed [22:0] Max_Der_int;

// Parameters
parameter IDLE = 0;
parameter PROCESS = 1;

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            Max_Derivative <= 0;
            Point1 <= 0;
            Point2 <= 0;
            Previous <= 0;
        end
    else
        begin
            state <= next;
            if (set_prev)
                Previous <= din_cond;
            else
                Previous <= Previous;

            if (set)
                begin
                    Max_Derivative <= Max_Der_int;
                    Point1 <= Point1_int;
                    Point2 <= Point2_int;
                end
            else if (reset_md)
                begin

```

```

        Max_Derivative <= 0;
        Point1 <= 0;
        Point2 <= 0;
    end
else
    begin
        Max_Derivative <= Max_Derivative;
        Point1 <= Point1;
        Point2 <= Point2;
    end
end
end

// Combinational Block to set Next State and Internal Variables
always @ (state or data_ready or done_collect or start or din_cond or
Previous or Max_Derivative)
begin

    set = 0;
    set_prev = 0;
    Point1_int = Previous;
    Point2_int = din_cond;
    Max_Der_int = Max_Derivative;

    case (state)

        IDLE:
            if (start)
                begin
                    next = PROCESS;
                    busy = 1;
                end
            else
                begin
                    next = IDLE;
                    busy = 0;
                end

        PROCESS:
            begin
                busy = 1;

                if (data_ready)
                    begin
                        set_prev = 1;

                        if (din_cond > Previous)
                            begin
                                if ((din_cond - Previous) > Max_Derivative)
                                    begin
                                        Max_Der_int = din_cond - Previous;
                                        set = 1;
                                    end
                                end
                            end
                        else
                            begin
                                if ((Previous - din_cond) > Max_Derivative)
                                    begin
                                        Max_Der_int = Previous - din_cond;
                                        set = 1;
                                    end
                                end
                            end
                        end
                    end

                end

            end

        end
    end
end

```



```
        if (done_collect)
            next = IDLE;
        else
            next = PROCESS;
        end
    default:
        begin
            next = IDLE;
            busy = 0;
        end
    endcase
end
endmodule
```

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 1, 2006
// File: MinMax.v
// Description: Module to calculate the maximum and minimum data value

module MinMax (clk, reset, din_cond, done_collect, data_ready, start,
               busy, Minimum, Maximum, reset_minmax, state, next);

// Inputs and Outputs
input clk, reset, done_collect, data_ready, start, reset_minmax;
input signed [21:0] din_cond;

output busy, state, next;
reg busy;
output signed [21:0] Minimum, Maximum;
reg signed [21:0] Minimum, Maximum;

// Internal Signals
reg state, next, set_min, set_max;
reg signed [21:0] Min_int, Max_int;

// Parameters
parameter IDLE = 0;
parameter PROCESS = 1;

// Sequential Block to set State and Registers
always @ (posedge clk)
    if (reset)
        begin
            state <= IDLE;
            Minimum <= 0;
            Maximum <= 0;
        end
    else
        begin
            state <= next;

            if (set_min)
                Minimum <= Min_int;
            else if (reset_minmax)
                Minimum <= 0;
            else
                Minimum <= Minimum;

            if (set_max)
                Maximum <= Max_int;
            else if (reset_minmax)
                Maximum <= 0;
            else
                Maximum <= Maximum;
        end
    end

// Combinational Block to set Next State and Internal Variables
always @ (state or data_ready or done_collect or start or din_cond or
         Minimum or Maximum)
    begin

```

```

set_min = 0;
set_max = 0;
Min_int = din_cond;
Max_int = din_cond;

case (state)
  IDLE:
    if (start)
      begin
        next = PROCESS;
        busy = 1;
      end
    else
      begin
        next = IDLE;
        busy = 0;
      end
    end

  PROCESS:
    begin
      busy = 1;
      if (data_ready)
        begin
          if (din_cond < Minimum)
            set_min = 1;

            if (din_cond > Maximum)
              set_max = 1;
            end
          end

          if (done_collect)
            next = IDLE;
          else
            next = PROCESS;
          end
        end

      default:
        begin
          next = IDLE;
          busy = 0;
        end
      endcase
    end
  endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project
// Date: May 1, 2006
// File: SumEnergy_cond.v
// Description: Module to calculate the sum and energy (sum of absolute
values)
//             for skin data. Skin Conductivity has been filtered, so have
//             both positive and negative values

module SumEnergy_cond (clk, reset, din_cond, done_collect, data_ready, start,
                      busy, Sum, Energy, state, next, reset_se);

// Inputs and Outputs
input clk, reset, done_collect, data_ready, start, reset_se;
input signed [21:0] din_cond;

output busy;
reg busy;
output signed [34:0] Sum;
reg signed [34:0] Sum;
output signed [34:0] Energy;
reg signed [34:0] Energy;
output state, next;

// Internal Signals
reg state, next;
reg set_sum, set_energy, inc_count;
reg [9:0] num_count;
reg [20:0] din_abs, temp;

// Parameters
parameter IDLE = 0;
parameter PROCESS = 1;

// Combinational Block to set absolute value
always @ (din_cond)
    if (din_cond < 0)
        begin
            temp = ~din_cond + 1;
            din_abs = temp[20:0];
        end
    else
        din_abs = din_cond[20:0];

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            Energy <= 0;
            Sum <= 0;
            num_count <= 0;
        end
    else
        begin
            state <= next;

```

```

    if (set_energy)
        Energy <= Energy + din_abs;
    else if (reset_se)
        Energy <= 0;
    else
        Energy <= Energy;

    if (set_sum)
        Sum <= Sum + din_cond;
    else if (reset_se)
        Sum <= 0;
    else
        Sum <= Sum;

    if (inc_count)
        num_count <= num_count + 1;
    else if (reset_se)
        num_count <= 0;
    else
        num_count <= num_count;
end

// Combinational Block to set Next State and Internal Variables
always @ (state or data_ready or done_collect or start or
        din_cond or num_count)
begin

    set_sum = 0;
    set_energy = 0;
    inc_count = 0;

    case (state)
        IDLE:
            if (start)
                begin
                    next = PROCESS;
                    busy = 1;
                end
            else
                begin
                    next = IDLE;
                    busy = 0;
                end
        PROCESS:
            begin
                busy = 1;

                if (data_ready)
                    begin
                        busy = 1;
                        set_energy = 1;
                        set_sum = 1;
                        inc_count = 1;
                    end

                if (done_collect)
                    next = IDLE;
                else
                    next = PROCESS;
            end
    end

```

```
        default:
            begin
                next = IDLE;
                busy = 1;
            end
        endcase
    end
endmodule
```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 1, 2006
// File: SumEnergy_pulse.v
// Description: Module to calculate the sum and energy (sum of absolute
values)
//             for pulse data. Since pulse is unsigned, do not need to worry
//             about adding positive and negative numbers
//             (i.e. Energy = Sum, so only need one of the signals)

module SumEnergy_pulse (clk, reset, din_pulse, done_collect, data_ready,
                        start, busy, SumEnergy, state, next, reset_se);

// Inputs and Outputs
input clk, reset, done_collect, data_ready, start, reset_se;
input [7:0] din_pulse;

output busy;
reg busy;
output [19:0] SumEnergy;
reg [19:0] SumEnergy;
output state, next;

// Internal Signals
reg state, next;
reg set_energy;

// Parameters
parameter IDLE = 0;
parameter PROCESS = 1;

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            SumEnergy <= 0;
        end
    else
        begin
            state <= next;

            if (set_energy)
                SumEnergy <= SumEnergy + din_pulse;
            else if (reset_se)
                SumEnergy <= 0;
            else
                SumEnergy <= SumEnergy;
        end
    end

// Combinational Block to set Next State and Internal Variables
always @ (state or data_ready or done_collect or start or din_pulse)
    begin

        set_energy = 0;

        case (state)

```

```

IDLE:
  if (start)
    begin
      next = PROCESS;
      busy = 1;
    end
  else
    begin
      next = IDLE;
      busy = 0;
    end

PROCESS:
  begin
    if (data_ready)
      begin
        busy = 1;
        set_energy = 1;
      end
    if (done_collect)
      next = IDLE;
    else
      next = PROCESS;
    end

default:
  begin
    next = IDLE;
    busy = 1;
  end
endcase
end
endmodule

```



```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 2, 2006
// File: ThreshCount2.v
// Description: Sets threshold and calculates the number of pulses and
//              number of samples to get 1/rate

module ThreshCount2 (clk, reset, data_ready, done_collect, start, din_pulse,
                    pulse_count, busy, state, next, reset_tc);

// Inputs and Outputs
input clk, reset, data_ready, done_collect, start, reset_tc;
input [7:0] din_pulse;

output [11:0] pulse_count;
output busy;
reg [11:0] pulse_count;
reg busy;
output [1:0] state, next;

// Internal Signals
reg [1:0] state, next;
reg inc_sample, inc_pulse, pulse;
reg [11:0] num_count;

// Parameters
parameter IDLE = 0;
parameter WAIT = 1;
parameter PULSE_HIGH = 2;

parameter THRESHOLD = 128;

// Combinational Block to determine whether past threshold
always @ (din_pulse)
    pulse = (din_pulse >= THRESHOLD);

// Sequential Block to set State and Registers
always @ (posedge clk)
    if (reset)
        begin
            state <= IDLE;
            num_count <= 0;
            pulse_count <= 0;
        end
    else
        begin
            state <= next;

            if (inc_sample)
                num_count <= num_count+1;
            else if (reset_tc)
                num_count <= 0;
            else
                num_count <= num_count;

            if (inc_pulse)
                pulse_count <= pulse_count+1;
        end
    end
endmodule

```

```

        else if (reset_tc)
            pulse_count <= 0;
        else
            pulse_count <= pulse_count;
    end

// Combinational Block to set Next State and Internal Variables
always @ (state or pulse or data_ready or done_collect or start)
begin

    inc_sample = 0;
    inc_pulse = 0;

    case (state)
        IDLE:
            if (start)
                begin
                    busy = 1;
                    next = WAIT;
                end
            else
                begin
                    busy = 0;
                    next = IDLE;
                end
            end

        WAIT:
            begin
                busy = 1;
                if (data_ready)
                    begin
                        inc_sample = 1;

                        if (pulse)
                            inc_pulse = 1;
                        else
                            inc_pulse = 0;
                    end
                if (done_collect)
                    next = IDLE;
                else if (pulse)
                    next = PULSE_HIGH;
                else
                    next = WAIT;
            end

        PULSE_HIGH:
            begin
                busy = 1;
                if (data_ready)
                    begin
                        inc_sample = 1;
                    end

                if (done_collect)
                    next = IDLE;
                else if (~pulse)
                    next = WAIT;
                else
                    next = PULSE_HIGH;
            end
    end
end

```

```

        default:
            begin
                next = IDLE;
                busy = 0;
            end
        endcase
    end
endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 12, 2006
// File: MinorFSM_store_ddmu.v
// Description: Minor FSM to store the control data
//             Note: since possibility of storing both types of data, build
in extra
//             bit to all the BRAMs for a data tag

```

```

module MinorFSM_store_ddmu (clk, reset, start, Sum_cond, Energy_cond,
MaxDeriv_cond,
                        Point1_cond, Point2_cond, Min_cond, Max_cond,
                        SumEnergy_pulse, pulse_count, xreal_max, ximag_max,
                        max_out, max_index, done_store, address, d_mem,
                        nd, rfd, we, tag, state, next);

```

```

// Inputs and Outputs
input clk, reset, start, tag;
input signed [34:0] Sum_cond;
input signed [34:0] Energy_cond;
input signed [22:0] MaxDeriv_cond;
input signed [21:0] Point1_cond, Point2_cond, Min_cond, Max_cond;
input [19:0] SumEnergy_pulse;
input [11:0] pulse_count;
input signed [27:0] xreal_max, ximag_max;
input signed [28:0] max_out;
input [10:0] max_index;
input rfd;

```

```

output [4:0] address;
output [253:0] d_mem;
output done_store, nd, we, state, next;
reg [4:0] address;
reg [253:0] d_mem;
reg done_store, nd, we;
reg delay, set_delay, reset_delay;

```

```

// Internal Signals
reg state, next;
reg inc_address;
reg we_int, nd_int, set_data, done_store_int;

```

```

// Parameters
parameter IDLE = 0;
parameter STORE = 1;

parameter MAX_ADDRESS = 31;

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
    begin
        state <= IDLE;
        address <= 0;
        we <= 0;
        nd <= 0;
        d_mem <= 0;
        done_store <= 0;
        delay <= 0;
    end

    else
    begin
        state <= next;
        done_store <= done_store_int;

        if (set_delay)
            delay <= 1;
        else if (reset_delay)
            delay <= 0;
        else
            delay <= delay;

        if (inc_address)
        begin
            if (address >= MAX_ADDRESS)
                address <= 0;
            else
                address <= address + 1;
        end
        else
            address <= address;

        if (set_data)
            d_mem <= {tag, Sum_cond, Energy_cond, MaxDeriv_cond,
                    Point1_cond, Point2_cond, Min_cond, Max_cond,
                    SumEnergy_pulse, pulse_count, max_out, max_index};
        else
            d_mem <= d_mem;

        we <= we_int;
        nd <= nd_int;
    end

// Combinational Block to set Next State and Internal Variables
always @ (state or start or rfd or delay)
    begin

        inc_address = 0;
        nd_int = 0;
        we_int = 0;
        set_data = 0;
    end

```

```

done_store_int = 0;
set_delay = 0;
reset_delay = 0;

case (state)

    IDLE:
        if (start)
            begin
                next = STORE;
                set_data = 1;
            end
        else
            next = IDLE;

    STORE:
        if (rfd)
            begin
                if (delay)
                    begin
                        inc_address = 1;
                        next = IDLE;
                        done_store_int = 1;
                        reset_delay = 1;
                    end
                else
                    begin
                        nd_int = 1;
                        we_int = 1;
                        next = STORE;
                        set_delay = 1;
                    end
                end
            end
        else
            next = STORE;

    default:
        next = IDLE;
endcase
end

endmodule

```

```
// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: May 1, 2006
// File: MinorFSM_decide_ddmu.v
// Description: Minor FSM to decide whether subject is lying or telling the
truth
```

```
module MinorFSM_decide_ddmu (clk, reset, start, address_in, Sum_cond,
Energy_cond,
                                MaxDeriv_cond, Point1_cond, Point2_cond, Max_cond,
                                Min_cond, SumEnergy_pulse, pulse_count, max_out,
                                max_index, done, liar, d_mem, nd, rdy, address_out,
                                state, next, comp);
```

```
// Inputs and Outputs
input clk, reset, start, rdy;
input signed [34:0] Sum_cond;
input signed [34:0] Energy_cond;
input signed [22:0] MaxDeriv_cond;
input signed [21:0] Point1_cond, Point2_cond, Min_cond, Max_cond;
input [19:0] SumEnergy_pulse;
input [11:0] pulse_count;
input signed [28:0] max_out;
input [10:0] max_index;
input [253:0] d_mem;
input [4:0] address_in;
```

```
output [1:0] state, next;
output [4:0] address_out;
output [10:0] comp;
output done, nd, liar;
reg [4:0] address_out;
reg done, done_int, nd, liar;
```

```
// Internal Signals
reg [1:0] state, next;
reg sample_data, nd_int, set_address, set_liar, reset_liar;
reg [10:0] comp, comp_int;
reg set_comp;
```

```
reg signed [34:0] Sum_cond_control;
reg signed [34:0] Energy_cond_control;
reg signed [22:0] MaxDeriv_cond_control;
reg signed [21:0] Point1_cond_control, Point2_cond_control;
reg signed [21:0] Min_cond_control, Max_cond_control;
reg [19:0] SumEnergy_pulse_control;
reg [11:0] pulse_count_control;
reg signed [28:0] max_out_control;
reg [10:0] max_index_control;
reg [3:0] temp;
```

```
// Parameters
parameter IDLE = 0;
parameter GET_DATA = 1;
parameter COMPARE = 2;
parameter DECIDE = 3;

parameter THRESH_SUM_COND = 0;
parameter THRESH_ENERGY_COND = 0;
parameter THRESH_MAXDERIV_COND = 0;
```

```

parameter THRESH_POINT1_COND = 0;
parameter THRESH_POINT2_COND = 0;
parameter THRESH_MAX_COND = 0;
parameter THRESH_MIN_COND = 0;
parameter THRESH_SUMENERGY_PULSE = 0;
parameter THRESH_PULSE_COUNT = 0;
parameter THRESH_MAX_OUT_FFT = 0;
parameter THRESH_MAX_INDEX_FFT = 0;

parameter DECISION_THRESHOLD = 5;

// Sequential Block to set State and Registers
always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            address_out <= 0;
            nd <= 0;
            liar <= 0;
            comp <= 0;
            done <= 0;
        end
    else
        begin
            state <= next;
            nd <= nd_int;
            done <= done_int;

            if (set_liar)
                liar <= 1;
            else if (reset_liar)
                liar <= 0;
            else
                liar <= liar;

            if (set_address)
                begin
                    if (address_in == 0)
                        address_out <= 0;
                    else
                        address_out <= address_in - 1;
                end

            if (sample_data)
                begin
                    Sum_cond_control <= d_mem[252:218];
                    Energy_cond_control <= d_mem[217:183];
                    MaxDeriv_cond_control <= d_mem[182:160];
                    Point1_cond_control <= d_mem[159:138];
                    Point2_cond_control <= d_mem[137:116];
                    Min_cond_control <= d_mem[115:94];
                    Max_cond_control <= d_mem[93:72];
                    SumEnergy_pulse_control <= d_mem[71:52];
                    pulse_count_control <= d_mem[51:40];
                    max_out_control <= d_mem[39:11];
                    max_index_control <= d_mem[10:0];
                end

            if (set_comp)
                comp <= comp_int;
            else

```

```

        comp <= comp;

    end

// Combinational block to set Next State and Internal Variables
always @ (state or rdy or Sum_cond_control or Sum_cond or Energy_cond_control
or
    Energy_cond or MaxDeriv_cond_control or MaxDeriv_cond or
    Point1_cond_control or Point1_cond or Point2_cond_control or
    Point2_cond or Max_cond_control or Max_cond or Min_cond_control or
    Min_cond or SumEnergy_pulse or SumEnergy_pulse_control or
    pulse_count_control or pulse_count or max_out_control or max_out or
    max_index_control or max_index or comp or start)
begin

    nd_int = 0;
    set_liar = 0;
    set_address = 0;
    sample_data = 0;
    comp_int = 0;
    reset_liar = 0;
    set_comp = 0;
    done_int = 0;

    case (state)

        IDLE:
            if (start)
                begin
                    set_address = 1;
//                    nd_int = 1;
                    next = GET_DATA;
                    reset_liar = 1;
                end
            else
                begin
                    next = IDLE;
                end

        GET_DATA:
            if (rdy)
                begin
                    sample_data = 1;
                    next = COMPARE;
                end
            else
                begin
                    next = GET_DATA;
                    nd_int = 1;
                end

        COMPARE:
            begin

                if ((Sum_cond_control - THRESH_SUM_COND) >= Sum_cond)
                    comp_int[0] = 0;
                else
                    comp_int[0] = 1;

                if ((Energy_cond_control - THRESH_ENERGY_COND) >= Energy_cond)
                    comp_int[1] = 0;
                else
                    comp_int[1] = 1;
            end
        endcase
    end
end

```



```

        if ((MaxDeriv_cond_control - THRESH_MAXDERIV_COND) >=
MaxDeriv_cond)
            comp_int[2] = 0;
        else
            comp_int[2] = 1;

        if ((Point1_cond_control - THRESH_POINT1_COND) >= Point1_cond)
            comp_int[3] = 0;
        else
            comp_int[3] = 1;

        if ((Point2_cond_control - THRESH_POINT2_COND) >= Point2_cond)
            comp_int[4] = 0;
        else
            comp_int[4] = 1;

        if ((Max_cond_control - THRESH_MAX_COND) >= Max_cond)
            comp_int[5] = 0;
        else
            comp_int[5] = 1;

        if ((Min_cond_control - THRESH_MIN_COND) >= Min_cond)
            comp_int[6] = 0;
        else
            comp_int[6] = 1;

        if (SumEnergy_pulse_control < THRESH_SUMENERGY_PULSE)
            comp_int[7] = 1;
        else if ((SumEnergy_pulse_control - THRESH_SUMENERGY_PULSE) >=
SumEnergy_pulse)
            comp_int[7] = 0;
        else
            comp_int[7] = 1;

        if (pulse_count_control < THRESH_PULSE_COUNT)
            comp_int[8] = 1;
        else if ((pulse_count_control - THRESH_PULSE_COUNT) >=
pulse_count)
            comp_int[8] = 0;
        else
            comp_int[8] = 1;

        if ((max_out_control - THRESH_MAX_OUT_FFT) >= max_out)
            comp_int[9] = 0;
        else
            comp_int[9] = 1;

        if (max_index_control < THRESH_MAX_INDEX_FFT)
            comp_int[10] = 1;
        else if ((max_index_control - THRESH_MAX_INDEX_FFT) >= max_index)
            comp_int[10] = 0;
        else
            comp_int[10] = 1;

        next = DECIDE;
        set_comp = 1;
    end

DECIDE:
begin
    temp = comp[0] + comp[1] + comp[2] + comp[3] + comp[4] + comp[5] +
        comp[6] + comp[7] + comp[8] + comp[9] + comp[10];

```

```
        if (temp > DECISION_THRESHOLD)
            set_liar = 1;
            next = IDLE;
            done_int = 1;
        end

    default:
        begin
            next = IDLE;
        end
    endcase
end
endmodule
```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: April 24, 2006
// File: top_FSM.v
// Description: Top module which joins all FSMs

module top_FSM (clk, reset, store, question, get_next, done_RAM0, done_RAM1,
               block0_DDMU, block1_DDMU, last_address0, read_out0,
               write_out0, tag_out0, address_out0, block0_RAM, block1_RAM,
               last_address1, read_out1, write_out1, tag_out1, address_out1,
               wait_RAM, wait_DDMU, store_rst, question_rst, address_in0,
               address_in1, reset_interface0, reset_interface1,
               state_major_mm,
               next_major_mm, state_minor0_mm, next_minor0_mm, busy0,
               state_minor1_mm, next_minor1_mm, busy1, start_read0,
               start_write0,
               start_read1, start_writel, delay1, delay2);

// Inputs and Outputs
input clk, reset, store, get_next, done_RAM0, done_RAM1;
input [1:0] question;
input block0_DDMU, block1_DDMU;
input [10:0] address_in0, address_in1;

output last_address0, read_out0, write_out0, tag_out0;
output last_address1, read_out1, write_out1, tag_out1;
output [10:0] address_out0, address_out1;
output block0_RAM, block1_RAM, wait_RAM, wait_DDMU;
output store_rst, question_rst;
output reset_interface0, reset_interface1;
output [1:0] state_minor0_mm, next_minor0_mm;
output [1:0] state_minor1_mm, next_minor1_mm;
output [2:0] state_major_mm, next_major_mm;
output busy0, busy1;
output start_read0, start_write0;
output start_read1, start_writel;
output delay1, delay2;

// Internal Signals
wire start_read0, start_write0, tag_out0, busy0;
wire start_read1, start_writel, tag_out1, busy1;
wire tag;

wire last_address0, last_address1;
wire [1:0] state_minor0_mm, next_minor0_mm;
wire [1:0] state_minor1_mm, next_minor1_mm;
wire [2:0] state_major_mm, next_major_mm;
wire reset_interface0, reset_interface1;
wire [10:0] address_out0, address_out1;
wire delay1, delay2;

MajorFSM_mm major (
    .clk(clk),
    .reset(reset),
    .store(store),
    .question(question),
    .store_rst(store_rst),
    .question_rst(question_rst),
    .block0_DDMU(block0_DDMU),
    .block1_DDMU(block1_DDMU),
    .block0_RAM(block0_RAM),
    .block1_RAM(block1_RAM),

```

```

        .read0(start_read0),
        .write0(start_write0),
        .read1(start_read1),
        .write1(start_write1),
        .busy0(busy0),
        .busy1(busy1),
        .tag(tag),
        .wait_RAM(wait_RAM),
        .wait_DDMU(wait_DDMU),
        .state(state_major_mm),
        .next(next_major_mm),
        .delay1(delay1),
        .delay2(delay2));

MinorFSM_mm minor0 (
    .clk(clk),
    .reset(reset),
    .start_read(start_read0),
    .start_write(start_write0),
    .tag(tag),
    .get_next(get_next),
    .done_RAM(done_RAM0),
    .busy(busy0),
    .last_address(last_address0),
    .read(read_out0),
    .write(write_out0),
    .address_in(address_in0),
    .address_out(address_out0),
    .state(state_minor0_mm),
    .next(next_minor0_mm),
    .tag_reg(tag_out0),
    .reset_interface(reset_interface0));

MinorFSM_mm minor1 (
    .clk(clk),
    .reset(reset),
    .start_read(start_read1),
    .start_write(start_write1),
    .tag(tag),
    .get_next(get_next),
    .done_RAM(done_RAM1),
    .busy(busy1),
    .last_address(last_address1),
    .read(read_out1),
    .write(write_out1),
    .address_in(address_in1),
    .address_out(address_out1),
    .state(state_minor1_mm),
    .next(next_minor1_mm),
    .tag_reg(tag_out1),
    .reset_interface(reset_interface1));

endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: April 24, 2006
// File: MajorFSM_mm.v
// Description: Major FSM for the Memory Module

module MajorFSM_mm (clk, reset, store, question, store_rst, question_rst,
                    block0_DDMU, block1_DDMU, block0_RAM, block1_RAM,
                    read0, write0, read1, writel, busy0, busy1, tag,
                    wait_RAM, wait_DDMU, state, next, delay1, delay2);

// Inputs and Outputs
input clk, reset, store;
input block0_DDMU, block1_DDMU, busy0, busy1;
input [1:0] question;

output store_rst, question_rst, block0_RAM, block1_RAM;
output read0, write0, read1, writel, tag;
reg store_rst, question_rst, block0_RAM, block1_RAM;
reg read0, write0, read1, writel, tag;
output wait_RAM, wait_DDMU;
reg wait_RAM, wait_DDMU;
output [2:0] state, next;
reg [2:0] state, next;
output delay1, delay2;

// Internal Signals and Registers

reg store_rst_int, question_rst_int, tag_int;
reg read0_int, write0_int, read1_int, writel_int;
reg set_b0_RAM, reset_b0_RAM, set_b1_RAM, reset_b1_RAM;
reg delay1, delay2, set_delay1, set_delay2, reset_delay;

// Parameters

parameter IDLE = 0;
parameter WRITE0 = 1;
parameter WRITE1_READ0 = 2;
parameter READ1 = 3;
parameter READ1_WRITE0 = 4;

// Sequential Block to set State and Registers

always @ (posedge clk)

    if (reset)
        begin
            state <= IDLE;
            store_rst <= 0;
            question_rst <= 0;
            block0_RAM <= 0;
            block1_RAM <= 0;
            read0 <= 0;
            read1 <= 0;
            write0 <= 0;
            writel <= 0;
            tag <= 1;
            delay1 <= 0;
            delay2 <= 0;
        end

```

```

        end

    else
        begin
            state <= next;
            store_rst <= store_rst_int;
            question_rst <= question_rst_int;

            if (set_delay1)
                delay1 <= 1;
            else if (reset_delay)
                delay1 <= 0;
            else
                delay1 <= delay1;

            if (set_delay2)
                delay2 <= 1;
            else if (reset_delay)
                delay2 <= 0;
            else
                delay2 <= delay2;

            if (set_b0_RAM)
                block0_RAM <= 1;
            else if (reset_b0_RAM)
                block0_RAM <= 0;
            else
                block0_RAM <= block0_RAM;

            if (set_b1_RAM)
                block1_RAM <= 1;
            else if (reset_b1_RAM)
                block1_RAM <= 0;
            else
                block1_RAM <= block1_RAM;

            read0 <= read0_int;
            read1 <= read1_int;
            write0 <= write0_int;
            writel <= writel_int;
            tag <= tag_int;
        end
    end
end

```

// Combinational Block to set Next State and Internal Variables

```

always @ (busy0 or busy1 or store or question or block0_DDMU or
        block1_DDMU or state or delay1 or delay2)
    begin
        read0_int = 0;
        read1_int = 0;
        write0_int = 0;
        writel_int = 0;
        set_b0_RAM = 0;
        reset_b0_RAM = 0;
        set_b1_RAM = 0;
        reset_b1_RAM = 0;
        store_rst_int = 0;
        question_rst_int = 0;
        set_delay1 = 0;
        set_delay2 = 0;
        reset_delay = 0;
    end

```

```

if (question == 1)
    tag_int = 0;
else
    tag_int = 1;

case (state)

    IDLE:
        if (store && ((question == 1) || (question == 2)) && ~block0_DDMU)
            begin
                next = WRITE0;
                set_b0_RAM = 1;
                write0_int = 1;
            end

        else next = IDLE;

    WRITE0:
        if (busy0 || block1_DDMU) next = WRITE0;
        else
            begin
                next = WRITE1_READ0;
                read0_int = 1;
                writel_int = 1;
                reset_b0_RAM = 1;
                set_b1_RAM = 1;
            end

    WRITE1_READ0:
        if ((busy0 || busy1) && (~delay1) && (~delay2))          next =
WRITE1_READ0;
        else
            begin
                if (delay2)
                    begin
                        next = READ1;
                        reset_delay = 1;
                    end
                else if (delay1)
                    begin
                        next = WRITE1_READ0;
                        set_delay2 = 1;
                    end
                else
                    begin
                        next = WRITE1_READ0;

                        store_rst_int = 1;
                        question_rst_int = 1;
                        read1_int = 1;
                        reset_b1_RAM = 1;
                        set_delay1 = 1;
                    end
            end

    READ1:
        if (store && ((question == 1) || (question == 2)) && ~block0_DDMU)
            begin
                next = READ1_WRITE0;
                write0_int = 1;
                set_b0_RAM = 1;
            end

```

```

        else if (~busy1) next = IDLE;
        else
            next = READ1;

    READ1_WRITE0:
        if (busy0 || busy1 || block1_DDMU) next = READ1_WRITE0;
        else
            begin
                next = WRITE1_READ0;

                read0_int = 1;
                write1_int = 1;
                reset_b0_RAM = 1;
                set_b1_RAM = 1;
            end

        default:    next = IDLE;
    endcase
end

// Combinational Block to set output LEDs

always @ (block0_DDMU or block1_DDMU or block0_RAM or block1_RAM or state)
    begin
        wait_DDMU = (block0_DDMU || block1_DDMU);
        wait_RAM = (block0_RAM || block1_RAM);
    end

endmodule

```



```
// Author: Archana Venkataraman
// Assignment: Final Project
// Date: April 24, 2006
// File: MinorFSM_mm.v
// Description: Minor FSM for the Memory Module (need 1 for each block RAM)
```

```
module MinorFSM_mm (clk, reset, start_read, start_write, tag, get_next,
done_RAM,
                    busy, last_address, read, write, address_in, address_out,
                    state, next, tag_reg, reset_interface);
```

```
// Inputs and Outputs
```

```
input clk, reset, start_read, start_write, tag, get_next, done_RAM;
input [10:0] address_in;
```

```
output busy, last_address, read, write;
output [10:0] address_out;
output [1:0] state, next;
output tag_reg, reset_interface;
```

```
reg busy, last_address, read, write;
reg [10:0] address_out;
reg [1:0] state, next;
reg reset_interface;
```

```
// Internal Signals and Registers
```

```
reg inc_address, reset_address, tag_reg, set_tag;
reg read_int, write_int, last_address_int;
reg set_rst, reset_rst;
reg second, set_second, reset_second;
```

```
// Parameters
```

```
parameter IDLE = 0;
parameter WRITE = 1;
parameter READ = 2;
```

```
parameter MAX_ADDRESS = 1499;
//parameter MAX_ADDRESS = 7;
```

```
// Sequential Block to set State and Registers
```

```
always @ (posedge clk)
```

```
    if (reset)
    begin
        state <= IDLE;
        last_address <= 0;
        address_out <= 0;
        read <= 0;
        write <= 0;
        tag_reg <= tag;
        reset_interface <= 0;
        second <= 0;
    end

    else
    begin
        state <= next;
```

```

last_address <= last_address_int;

if (set_rst)
    reset_interface <= 1;
else if (reset_rst)
    reset_interface <= 0;
else
    reset_interface <= reset_interface;

if (set_second)
    second <= 1;
else if (reset_second)
    second <= 0;
else
    second <= second;

if (inc_address)
    address_out <= address_out + 1;
else if (reset_address)
    address_out <= 0;
else
    address_out <= address_out;

if (set_tag)
    tag_reg <= tag;
else
    tag_reg <= tag_reg;

read <= read_int;
write <= write_int;
end

```

// Combinational Block to set Next State and Internal Variables

```

always @ (start_read or start_write or get_next or tag or done_RAM or state
    or address_in or address_out or second)

```

```

begin

```

```

    last_address_int = 0;
    inc_address = 0;
    reset_address = 0;
    set_tag = 0;
    read_int = 0;
    write_int = 0;
    set_second = 0;
    reset_second = 0;
    set_rst = 0;
    reset_rst = 0;

```

```

case (state)

```

```

    IDLE:

```

```

        if (start_write)
            begin
                next = WRITE;
                set_tag = 1;
                write_int = 1;
                busy = 1;
                set_tag = 1;
                reset_rst = 1;
            end

```

```

        else if (start_read)
            begin

```

```

        next = READ;
        busy = 1;
        reset_rst = 1;
    end
else
    begin
        next = IDLE;
        busy = 0;
    end

WRITE:
    if (done_RAM || (address_in >= MAX_ADDRESS))
    begin
        next = IDLE;
        busy = 1;
        set_rst = 1;
    end
else
    begin
        next = WRITE;
        write_int = 1;
        busy = 1;
    end

READ:
    if (address_out > MAX_ADDRESS)
    begin
        next = IDLE;
        reset_address = 1;
        busy = 1;
    end
    else if (get_next)
    begin
        next = READ;
        read_int = 1;
        busy = 1;
        set_second = 1;

        if (address_out == MAX_ADDRESS)
            last_address_int = 1;
        end
    else
    begin
        if (second)
        begin
            inc_address = 1;
            reset_second = 1;
        end

        next = READ;
        busy = 1;
    end

default:
    begin
        next = IDLE;
        busy = 0;
    end

endcase
end
endmodule

```

```

// Author: Archana Venkataraman
// Assignment: Final Project - UserCapture
// Date: April 21, 2006
// File: UsrReg2.v
// Description: Synchronizes user-input signals

module UserCapt1 (clk, reset, question_in, store_in, question_rst,
                  store_rst, question_out, store_out, reset_sync);

input clk, reset, store_in;
input [1:0] question_in;
input question_rst, store_rst;

output store_out, reset_sync;
output [1:0] question_out;

wire store_sync, reset_sync;
wire [1:0] question_sync;

Synchronizer1 sync1 (.clk(clk),
                    .reset(reset),
                    .question(question_in),
                    .store(store_in),
                    .reset_sync(reset_sync),
                    .question_sync(question_sync),
                    .store_sync(store_sync));

UsrReg2 reg_quest (.clk(clk),
                  .reset(reset_sync),
                  .reg_reset(question_rst),
                  .in(question_sync),
                  .out(question_out),
                  .store(store_sync));

UsrReg reg_store (.clk(clk),
                  .reset(reset_sync),
                  .reg_reset(store_rst),
                  .in(store_sync),
                  .out(store_out));

endmodule

```



```

// Author: 6.111 Staff
// Assignment: Final Project - UserCapture
// Date: April 21, 2006
// File: debounce.v
// Description: Produces synchronous, debounced output for 2-bit input

module debounce (reset, clock, noisy, clean);
    parameter DELAY = 270000;    // .01 sec with a 27Mhz clock
    input reset, clock, noisy;
    output clean;

    reg [18:0] count;
    reg new, clean;

    always @(posedge clock)
        if (reset)
            begin
                count <= 0;
                new <= noisy;
                clean <= noisy;
            end
        else if (noisy != new)
            begin
                new <= noisy;
                count <= 0;
            end
        else if (count == DELAY)
            clean <= new;
        else
            count <= count+1;

endmodule

```

```

// Author: 6.111 Staff - modified by Archana Venkataraman
// Assignment: Final Project - UserCapture
// Date: April 21, 2006
// File: debounce2.v
// Description: Produces synchronous, debounced output for 2-bit input

module debounce2 (reset, clock, noisy, clean);
    parameter DELAY = 270000;    // .01 sec with a 27Mhz clock
    input reset, clock;
    input [1:0] noisy;
    output [1:0] clean;

    reg [18:0] count;
    reg [1:0] new, clean;

    always @(posedge clock)

        if (reset)
            begin
                count <= 0;
                new <= noisy;
                clean <= noisy;
            end

        else if (noisy != new)
            begin
                new <= noisy;
                count <= 0;
            end

        else if (count == DELAY)
            clean <= new;

        else
            count <= count+1;
endmodule

```

```
// Author: Archana Venkataraman
// Assignment: Final Project - UserCapture
// Date: April 21, 2006
// File: UsrReg.v
// Description: Registers user-input signal until used

module UsrReg (clk, reset, reg_reset, in, out);

input clk, reset, reg_reset, in;

output out;
reg out;

always @ (posedge clk)
begin
    if(reset || reg_reset)
        out <= 0;
    else if (in)
        out <= in;
    else
        out <= out;
end

endmodule
```



```
// Author: Archana Venkataraman
// Assignment: Final Project - UserCapture
// Date: April 21, 2006
// File: UsrReg2.v
// Description: Registers 2-bit user-input signal until used

module UsrReg2 (clk, reset, reg_reset, in, out, store);

input clk, reset, reg_reset, store;
input [1:0] in;

output [1:0] out;
reg [1:0] out;

always @ (posedge clk)
begin
    if(reset || reg_reset)
        out <= 0;
    else if (store)
        out <= in;
    else
        out <= out;
end

endmodule
```

```
// Author: Archana Venkataraman
// Assignment: Final Project - MemModule
// Date: April 24, 2006
// File: test_Connect_All3.v
// Description: Testbench for module which joins all FSMs
```

```
`timescale 1ns / 1ps
```

```
module test_Connect_All3_v;
```

```
    // Inputs
```

```
    reg clk;
    reg reset_in;
    reg [1:0] question_in;
    reg store_in;
    reg done_RAM0;
    reg done_RAM1;
    reg data_valid0;
    reg data_valid1;
    reg [10:0] address_in0;
    reg [10:0] address_in1;
    reg [16:0] data0;
    reg [16:0] data1;
```

```
    // Outputs
```

```
    wire reset;
    wire store;
    wire [1:0] question;
    wire question_rst;
    wire store_rst;
    wire last_address0;
    wire read_out0;
    wire write_out0;
    wire tag_out0;
    wire [10:0] address_out0;
    wire reset_interface0;
    wire busy0;
    wire last_address1;
    wire read_out1;
    wire write_out1;
    wire tag_out1;
    wire [10:0] address_out1;
    wire reset_interface1;
    wire busy1;
    wire block0_RAM;
    wire block1_RAM;
    wire wait_RAM;
    wire wait_DDMU;
    wire block0_DDMU;
    wire block1_DDMU;
    wire get_next;
    wire liar;
    wire [2:0] state_major_mm;
    wire [2:0] next_major_mm;
    wire [1:0] state_minor0_mm;
    wire [1:0] next_minor0_mm;
    wire [1:0] state_minor1_mm;
    wire [1:0] next_minor1_mm;
    wire start_read0;
    wire start_read1;
    wire start_write0;
    wire start_writel;
    wire delay1;
```

```

wire delay2;
wire [2:0] state_major_ddmu;
wire [2:0] next_major_ddmu;
wire [1:0] state_minor_coll;
wire [1:0] next_minor_coll;
wire [4:0] address_ddmu;
wire [10:0] comp;
wire [21:0] dout_cond;
wire [7:0] dout_pulse;
wire start_coll;
wire done_collect;
wire proc_data1;
wire data_ready;
wire start_proc;
wire done_proc;
wire start_store;
wire done_store;
wire start_decide;
wire done_decide;
wire state_minor_proc;
wire next_minor_proc;
wire state_minor_store;
wire next_minor_store;
wire [1:0] state_minor_decide;
wire [1:0] next_minor_decide;
wire rdy_ddmu, rfd_ddmu, nd_ddmu;
wire done0;
wire busy1_ddmu, busy2_ddmu, busy3_ddmu, busy4_ddmu, busy5_ddmu,
busy6_ddmu;

```

```

// Instantiate the Unit Under Test (UUT)

```

```

Connect_All uut (
    .clk(clk),
    .reset_in(reset_in),
    .question_in(question_in),
    .store_in(store_in),
    .done_RAM0(done_RAM0),
    .done_RAM1(done_RAM1),
    .data_valid0(data_valid0),
    .data_valid1(data_valid1),
    .address_in0(address_in0),
    .address_in1(address_in1),
    .data0(data0),
    .data1(data1),
    .reset(reset),
    .store(store),
    .question(question),
    .question_rst(question_rst),
    .store_rst(store_rst),
    .last_address0(last_address0),
    .read_out0(read_out0),
    .write_out0(write_out0),
    .tag_out0(tag_out0),
    .address_out0(address_out0),
    .reset_interface0(reset_interface0),
    .busy0(busy0),
    .last_address1(last_address1),
    .read_out1(read_out1),
    .write_out1(write_out1),
    .tag_out1(tag_out1),
    .address_out1(address_out1),
    .reset_interface1(reset_interface1),
    .busy1(busy1),

```

```

.block0_RAM(block0_RAM),
.block1_RAM(block1_RAM),
.wait_RAM(wait_RAM),
.wait_DDMU(wait_DDMU),
.block0_DDMU(block0_DDMU),
.block1_DDMU(block1_DDMU),
.get_next(get_next),
.liar(liar),
.state_major_mm(state_major_mm),
.next_major_mm(next_major_mm),
.state_minor0_mm(state_minor0_mm),
.next_minor0_mm(next_minor0_mm),
.state_minor1_mm(state_minor1_mm),
.next_minor1_mm(next_minor1_mm),
.start_read0(start_read0),
.start_read1(start_read1),
.start_write0(start_write0),
.start_writel(start_writel),
.delay1(delay1),
.delay2(delay2),
.state_major_ddmu(state_major_ddmu),
.next_major_ddmu(next_major_ddmu),
.state_minor_coll(state_minor_coll),
.next_minor_coll(next_minor_coll),
.address_ddmu(address_ddmu),
.comp(comp),
.dout_cond(dout_cond),
.dout_pulse(dout_pulse),
.start_coll(start_coll),
.done_collect(done_collect),
.proc_data1(proc_data1),
.data_ready(data_ready),
.start_proc(start_proc),
.done_proc(done_proc),
.start_store(start_store),
.done_store(done_store),
.start_decide(start_decide),
.done_decide(done_decide),
.state_minor_proc(state_minor_proc),
.next_minor_proc(next_minor_proc),
.state_minor_store(state_minor_store),
.next_minor_store(next_minor_store),
.state_minor_decide(state_minor_decide),
.next_minor_decide(next_minor_decide),
.rdy_ddmu(rdy_ddmu),
.rfd_ddmu(rfd_ddmu),
.nd_ddmu(nd_ddmu),
.busy1_ddmu(busy1_ddmu),
.busy2_ddmu(busy2_ddmu),
.busy3_ddmu(busy3_ddmu),
.busy4_ddmu(busy4_ddmu),
.busy5_ddmu(busy5_ddmu),
.busy6_ddmu(busy6_ddmu),
.done0(done0));

always
    #37      clk = ~clk;

initial begin
    // Initialize Inputs
    clk = 0;
    reset_in = 1;
    question_in = 1;

```

[illegible]

```
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;

#444;
address_in1 = 1500;
#74;
address_in1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;

#1600;
store_in = 1;
#148;
store_in = 0;
#222;
address_in0 = 1500;
#74;
address_in0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
```

```
#74;
data_valid0 = 1;
#444;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
address_in1 = 1500;
#74;
address_in1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
```

```
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#1600;
store_in = 1;
#148;
store_in = 0;
#222;
address_in0 = 1500;
#74;
address_in0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
```



```
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
data_valid0 = 1;
#74;
data_valid0 = 0;
#444;
address_in1 = 1500;
#74;
address_in1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
```

```
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;
#444;
data_valid1 = 1;
#74;
data_valid1 = 0;

end

endmodule
```

```

////////////////////////////////////
/
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Lab 4 (Spring 2006)
//
//
// Created: March 13, 2006
// Author: Nathan Ickes
//
////////////////////////////////////
/

module labkit (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
               ac97_bit_clock,

               vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
               vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
               vga_out_vsync,

               tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
               tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
               tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

               tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
               tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
               tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
               tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

               ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
               ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

               ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
               ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

               clock_feedback_out, clock_feedback_in,

               flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
               flash_reset_b, flash_sts, flash_byte_b,

               rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

               mouse_clock, mouse_data, keyboard_clock, keyboard_data,

               clock_27mhz, clock1, clock2,

               disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
               disp_reset_b, disp_data_in,

               button0, button1, button2, button3, button_enter, button_right,
               button_left, button_down, button_up,

               switch,

               led,

               user1, user2, user3, user4,

               daughtercard,

               systemace_data, systemace_address, systemace_ce_b,
               systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

               analyzer1_data, analyzer1_clock,
               analyzer2_data, analyzer2_clock,

```

```

        analyzer3_data, analyzer3_clock,
        analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synth, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
       vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
       tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
       tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
       tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
       tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0] systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbdrdy;

```

```

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
    analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////
//
// I/O Assignments
//

////////////////////////////////////

// Audio Input and Output
assign beep = 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;

```

```

assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
/*
// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;
*/
// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;

////////////////////////////////////
/
//
// Lab 4 Components
//
////////////////////////////////////

wire pclk, pixel_clock;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(slowclk));
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 4
// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 2
// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is "NONE"
BUFG pixel_clock_buf (.I(slowclk), .O(clock_out));

//
// VGA output signals
//

// Inverting the clock to the DAC provides half a clock period for signals
// to propagate from the FPGA to the DAC.
assign vga_out_pixel_clock = ~pixel_clock;

// The composite sync signal is used to encode sync data in the green
// channel analog voltage for older monitors. It does not need to be
// implemented for the monitors in the 6.111 lab, and can be left at 1'b1.
assign vga_out_sync_b = 1'b1;

// The following assignments should be deleted and replaced with your own
// code to implement the Pong game.
assign vga_out_red = 8'h0;
assign vga_out_green = 8'h0;
assign vga_out_blue = 8'h0;
assign vga_out_blank_b = 1'b1;

```

```

assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;

//
*****
***
//
// Code for Final Project - Archana
//
//
*****
***

    // Buttons, Switches, and Individual LEDs

    // User I/Os
    assign user1[30:0] = 31'hZ;
    assign user2 = 32'hZ;
// assign user3 = 32'hZ;
// assign user4 = 32'hZ;

    // Logic Analyzer
// assign analyzer1_data = 16'h0;
    assign analyzer1_clock = 1'b1;
// assign analyzer2_data[15:13] = 3'd0;
    assign analyzer2_clock = 1'b1;
// assign analyzer3_data = 16'h0;
    assign analyzer3_clock = 1'b1;
// assign analyzer4_data = 16'h0;
// assign analyzer4_clock = 1'b1;

//*****
*****
// CONNECT_ALL

    // Inputs
    wire reset_in;
    wire [1:0] question_in;
    wire store_in;
    wire done_RAM0;
    wire done_RAM1;
    wire data_valid0;
    wire data_valid1;
    wire [10:0] address_in0;
    wire [10:0] address_in1;
    wire [16:0] data0;
    wire [16:0] data1;

    // Outputs
    wire reset;
    wire store;
    wire [1:0] question;
    wire question_rst;
    wire store_rst;
    wire last_address0;
    wire read_out0;
    wire write_out0;
    wire tag_out0;
    wire [10:0] address_out0;
    wire reset_interface0;
    wire busy0;

```

```

wire last_address1;
wire read_out1;
wire write_out1;
wire tag_out1;
wire [10:0] address_out1;
wire reset_interfacel;
wire busy1;
wire block0_RAM;
wire block1_RAM;
wire wait_RAM;
wire wait_DDMU;
wire block0_DDMU;
wire block1_DDMU;
wire get_next;
wire liar;
wire [2:0] state_major_mm;
wire [2:0] next_major_mm;
wire [1:0] state_minor0_mm;
wire [1:0] next_minor0_mm;
wire [1:0] state_minor1_mm;
wire [1:0] next_minor1_mm;
wire start_read0;
wire start_read1;
wire start_write0;
wire start_writel;
wire delay1;
wire delay2;
wire [2:0] state_major_ddmu;
wire [2:0] next_major_ddmu;
wire [1:0] state_minor_coll;
wire [1:0] next_minor_coll;
wire [4:0] address_ddmu;
wire [10:0] comp;
wire [21:0] dout_cond;
wire [7:0] dout_pulse;
wire start_coll;
wire done_collect;
wire proc_data1;
wire data_ready;
wire start_proc;
wire done_proc;
wire start_store;
wire done_store;
wire start_decide;
wire done_decide;
wire state_minor_proc;
wire next_minor_proc;
wire state_minor_store;
wire next_minor_store;
wire [1:0] state_minor_decide;
wire [1:0] next_minor_decide;
wire busy1_ddmu, busy2_ddmu, busy3_ddmu, busy4_ddmu, busy5_ddmu,
busy6_ddmu;
wire done0, reset_fir, block;
wire rdy_ddmu, rfd_ddmu, nd_ddmu;
wire [10:0] index_fft;

wire signed [21:0] Min_cond, Max_cond, Point1_cond, Point2_cond;
wire signed [22:0] MaxDeriv_cond;
wire signed [34:0] Sum_cond, Energy_cond;
wire [19:0] SumEnergy_pulse;
wire [11:0] pulse_count;
wire signed [28:0] max_out;
wire [10:0] max_index;

```



```

Connect_All uut (
    .clk(clock_out),
    .reset_in(reset_in),
    .question_in(question_in),
    .store_in(store_in),
    .done_RAM0(done_RAM0),
    .done_RAM1(done_RAM1),
    .data_valid0(data_valid0),
    .data_valid1(data_valid1),
    .address_in0(address_in0),
    .address_in1(address_in1),
    .data0(data0),
    .data1(data1),
    .reset(reset),
    .store(store),
    .question(question),
    .question_rst(question_rst),
    .store_rst(store_rst),
    .last_address0(last_address0),
    .read_out0(read_out0),
    .write_out0(write_out0),
    .tag_out0(tag_out0),
    .address_out0(address_out0),
    .reset_interface0(reset_interface0),
    .busy0(busy0),
    .last_address1(last_address1),
    .read_out1(read_out1),
    .write_out1(write_out1),
    .tag_out1(tag_out1),
    .address_out1(address_out1),
    .reset_interface1(reset_interface1),
    .busy1(busy1),
    .block0_RAM(block0_RAM),
    .block1_RAM(block1_RAM),
    .wait_RAM(wait_RAM),
    .wait_DDMU(wait_DDMU),
    .block0_DDMU(block0_DDMU),
    .block1_DDMU(block1_DDMU),
    .get_next(get_next),
    .liar(liar),
    .state_major_mm(state_major_mm),
    .next_major_mm(next_major_mm),
    .state_minor0_mm(state_minor0_mm),
    .next_minor0_mm(next_minor0_mm),
    .state_minor1_mm(state_minor1_mm),
    .next_minor1_mm(next_minor1_mm),
    .start_read0(start_read0),
    .start_read1(start_read1),
    .start_write0(start_write0),
    .start_writel(start_writel),
    .delay1(delay1),
    .delay2(delay2),
    .state_major_ddmu(state_major_ddmu),
    .next_major_ddmu(next_major_ddmu),
    .state_minor_coll(state_minor_coll),
    .next_minor_coll(next_minor_coll),
    .address_ddmu(address_ddmu),
    .comp(comp),
    .dout_cond(dout_cond),
    .dout_pulse(dout_pulse),
    .start_coll(start_coll),
    .done_collect(done_collect),

```

```

.proc_data1(proc_data1),
.data_ready(data_ready),
.start_proc(start_proc),
.done_proc(done_proc),
.start_store(start_store),
.done_store(done_store),
.start_decide(start_decide),
.done_decide(done_decide),
.state_minor_proc(state_minor_proc),
.next_minor_proc(next_minor_proc),
.state_minor_store(state_minor_store),
.next_minor_store(next_minor_store),
.state_minor_decide(state_minor_decide),
.next_minor_decide(next_minor_decide),
.busy1_ddmu(busy1_ddmu),
.busy2_ddmu(busy2_ddmu),
.busy3_ddmu(busy3_ddmu),
.busy4_ddmu(busy4_ddmu),
.busy5_ddmu(busy5_ddmu),
.busy6_ddmu(busy6_ddmu),
.done0(done0),
.reset_fir(reset_fir),
.rdy_ddmu(rdy_ddmu),
.rfd_ddmu(rfd_ddmu),
.nd_ddmu(nd_ddmu),
.index(index_fft),
.block(block),
.Min_cond(Min_cond),
.Max_cond(Max_cond),
.Point1_cond(Point1_cond),
.Point2_cond(Point2_cond),
.MaxDeriv_cond(MaxDeriv_cond),
.Sum_cond(Sum_cond),
.Energy_cond(Energy_cond),
.SumEnergy_pulse(SumEnergy_pulse),
.pulse_count(pulse_count),
.max_out(max_out),
.max_index(max_index));

//*****
*****

//*****
*****
// Simulate Data

wire [7:0] pulse;
wire [7:0] conductivity;
wire [7:0] conductivity_change;

randomizer conduct_random (.clk(clock_out) , .reset(reset),
                           .value(conductivity_change));

conductivity_sim2 cd_sim (.clk(clock_out), .reset(reset),
                          .value(conductivity),.random (conductivity_change));

pulse_sim2 p_sim (.clk(clock_out) ,.reset(reset),
                  .pulse(pulse), .random(conductivity_change));

//*****
*****

```

```

//*****
// TESTING ASSIGNMENTS

    assign clk = clock_out;
    assign reset_in = ~button0;
    assign question_in = switch[1:0];
    assign store_in = ~button1;

// assign done_RAM0 = 1'b0;
// assign done_RAM1 = 1'b0;
// assign address_in0 = {7'd0, switch[6], 3'd0};
// assign address_in1 = {7'd0, switch[7], 3'd0};

// assign data0 = {1'b0, conductivity, pulse};
// assign data1 = {1'b0, conductivity, pulse};
// assign data0 = 0;
// assign data1 = 0;

//debounce_test deb_in1 (.reset(reset), .clock(clock_out), .noisy(~button2),
.pulse_out(data_valid0));
//debounce_test deb_in2 (.reset(reset), .clock(clock_out), .noisy(~button3),
.pulse_out(data_valid1));

    assign led[0] = ~block0_RAM;
    assign led[1] = ~block1_RAM;
    assign led[2] = ~wait_RAM;
    assign led[3] = ~wait_DDMU;
    assign led[4] = ~block0_DDMU;
    assign led[5] = ~block1_DDMU;
    assign led[6] = ~done0;
    assign led[7] = ~busy6_ddmu;

// Logic Analyzer Assignments

    assign analyzer4_clock = clock_out;

    assign analyzer4_data[0] = busy1_ddmu;
    assign analyzer4_data[1] = busy2_ddmu;
    assign analyzer4_data[2] = busy5_ddmu;
    assign analyzer4_data[3] = busy6_ddmu;
    assign analyzer4_data[14:4] = comp;

    assign analyzer4_data[15] = liar;

// assign analyzer4_data[8] = start_coll;
// assign analyzer4_data[9] = done_collect;
// assign analyzer4_data[10] = done_proc;
// assign analyzer4_data[11] = get_next;
// assign analyzer4_data[15:12] = address_out1[3:0];

    assign analyzer2_data[0] = start_coll;
    assign analyzer2_data[1] = start_proc;
    assign analyzer2_data[2] = start_store;
    assign analyzer2_data[3] = start_decide;

    assign analyzer2_data[4] = get_next;
    assign analyzer2_data[5] = done_collect;
    assign analyzer2_data[6] = data_valid0;
    assign analyzer2_data[7] = data_valid1;

    assign analyzer2_data[8] = data_ready;
    assign analyzer2_data[9] = reset_interface0;
    assign analyzer2_data[10] = rfd_ddmu;

```

```

assign analyzer2_data[11] = done0;

assign analyzer2_data[13:12] = state_minor_coll;
assign analyzer2_data[14] = block;
assign analyzer2_data[15] = proc_data1;

assign analyzer1_data[15:0] = dout_cond[15:0];
assign analyzer3_data[15:0] = data0[15:0];

// Output Assignments
//
// led[0] -> block0_RAM;
// led[1] -> block1_RAM;
// led[2] -> wait_RAM
// led[3] -> wait_DDMU
// led[4] -> ~block0_DDMU;
// led[5] -> ~block1_DDMU;
//
// button2 -> data_valid0;
// button3 -> data_valid1;
//
// char0 -> state_major_mm
// char1 -> state_minor0_mm
// char2 -> state_minor1_mm
// char3 -> state_major_ddmu
// char4 -> state_minor_coll
// char5 -> state_minor_proc
// char6 -> state_minor_store
// char7 -> state_minor_decide

//*****

//*****

// WIRE DEFINITIONS

// Recorder

        wire      ADC_request_0;
        wire      recorder0_busy;
        wire      full_0;

//      wire [3:0]      state_out;

        wire      ADC_request_1;
        wire      recorder1_busy;
        wire      full_1;

// RAM Interface

// RAM 0

        wire [16:0] ram0_data_user;
        wire [10:0] ram0_address_user;
        wire      read_request_0;
        wire      write_request_0;
        wire      ram0_rfd;
        wire      ram0_rdy;

```

```

        wire [16:0] ram0_return_data_RAM;
        wire [16:0] ram0_data_to_user;
        wire      rfd_user_0;
        wire      rdy_user_0;
        wire [16:0] ram0_data_RAM;
        wire [10:0] ram0_addr;
        wire      ram0_en;
        wire      ram0_nd;
        wire      ram0_we;
        wire      RAM_busy_0;
//      wire      data_valid0;

//      RAM 1

        wire [16:0] raml_data_user;
        wire [10:0] raml_address_user;
        wire      read_request_1;
        wire      write_request_1;
        wire      raml_rfd;
        wire      raml_rdy;
        wire [16:0] raml_return_data_RAM;
        wire [16:0] raml_data_to_user;
        wire      rfd_user_1;
        wire      rdy_user_1;
        wire [16:0] raml_data_RAM;
        wire [10:0] raml_addr;
        wire      raml_en;
        wire      raml_nd;
        wire      raml_we;
        wire      RAM_busy_1;
//      wire      data_valid1;

//      ADC 1
        wire      ADC_clock;
        wire      ADC_intr_bar_1;
        wire [7:0] ADC_data_bus_1;
        wire      ADC_cs_bar_1;
        wire      ADC_rd_bar_1;
        wire      ADC_wr_bar_1;
        wire [7:0] ADC_data_bus_out_1;
        wire      ADC_busy_1;

//      ADC 2
        wire      ADC_intr_bar_2;
        wire [7:0] ADC_data_bus_2;
        wire      ADC_cs_bar_2;
        wire      ADC_rd_bar_2;
        wire      ADC_wr_bar_2;
        wire [7:0] ADC_data_bus_out_2;
        wire      ADC_busy_2;

//*****

//*****

// RAM INTERFACES

        RAM_interface ram_interface_block_0
        (
//      Inputs
            .global_clock(clock_out),
            .global_reset(global_reset),

```

```

//      Inputs from User
        .ram_data_user(ram0_data_user),
        .ram_address_user(address_out0),    // set read address
        .read_request(read_out0),          // tell when to read
        .write_request(write_request_0),
//      Inputs from RAM
        .rfd(ram0_rfd),
        .rdy(ram0_rdy),
        .return_data_RAM(ram0_return_data_RAM),
//      Outputs to user
        .ram_data_to_user(ram0_data_to_user),    // returned data
        .rfd_user(rfd_user_0),
        .rdy_user(rdy_user_0),
//      Outputs to RAM
        .ram_data_RAM(ram0_data_RAM),
        .ram_address_RAM(ram0_addr),
        .en(ram0_en),
        .nd(ram0_nd),
        .we(ram0_we),

        .RAM_busy(RAM_busy_0),
        .data_valid(data_valid0)
    );

```

```

bram ram_block_0
(
    .addr(ram0_addr),
    .clk(clock_27mhz),
    .din(ram0_data_RAM),
    .dout(ram0_return_data_RAM),
    .en(ram0_en),
    .nd(ram0_nd),
    .rfd(ram0_rfd),
    .rdy(ram0_rdy),
    .we(ram0_we)
);

```

```

RAM_interface ram_interface_block_1
(
//      Inputs
        .global_clock(clock_out),
        .global_reset(global_reset),
//      Inputs from User
        .ram_data_user(ram1_data_user),
        .ram_address_user(address_out1),
        .read_request(read_out1),
        .write_request(write_request_1),
//      Inputs from RAM
        .rfd(ram1_rfd),
        .rdy(ram1_rdy),
        .return_data_RAM(ram1_return_data_RAM),
//      Outputs to user
        .ram_data_to_user(ram1_data_to_user),
        .rfd_user(rfd_user_1),
        .rdy_user(rdy_user_1),
//      Outputs to RAM
        .ram_data_RAM(ram1_data_RAM),
        .ram_address_RAM(ram1_addr),
        .en(ram1_en),
        .nd(ram1_nd),
        .we(ram1_we),

```

```

        .RAM_busy(RAM_busy_1),
        .data_valid(data_valid1)
    );

    bram ram_block_1
    (
        .addr(ram1_addr),
        .clk(clock_out),
        .din(ram1_data_RAM),
        .dout(ram1_return_data_RAM),
        .en(ram1_en),
        .nd(ram1_nd),
        .rfd(ram1_rfd),
        .rdy(ram1_rdy),
        .we(ram1_we)
    );

//*****

//*****

// RECORDERS

    Recorder rec0
    (
        // Global inputs
        .global_clock(clock_out),
        .global_reset(global_reset),
        // Local inputs from user
        .record_request(write_out0),
        .extra_bit_value(tag_out0), // type of question
        .data_clear(reset_interface0), // Clears data when bank full
(set by user)
        // Local inputs from devices
        .ADC_busy_from_ADC_1(ADC_busy_1),
        .ADC_busy_from_ADC_2(ADC_busy_2),
        .RAM_busy_from_RAM(RAM_busy_0),
        .ADC_1_data(ADC_data_bus_1), //[7:0]
        .ADC_2_data(ADC_data_bus_2), //[7:0]
        // outputs to user
        .next_write_address(ram0_address_user), // 11 bit array...
determines next address
        .recorder_busy(recorder0_busy),
        // outputs to devices
        .ADC_request_to_device(ADC_request_0),
        .RAM_write_request_to_device(write_request_0),
        .RAM_write_data_to_device(ram0_data_user), //[16:0]
        .full(full_0) // determines when bank full
//
        .state_out(state_out)
    );

    Recorder rec1
    (
        // Global inputs
        .global_clock(clock_out),
        .global_reset(global_reset),
        // Local inputs from user
        .record_request(write_out1),
        .extra_bit_value(tag_out1),
        .data_clear(reset_interfacel),
        // Local inputs from devices
        .ADC_busy_from_ADC_1(ADC_busy_1),

```

```

        .ADC_busy_from_ADC_2(ADC_busy_2),
        .RAM_busy_from_RAM(RAM_busy_1),
        .ADC_1_data(ADC_data_bus_1),    //[7:0]
        .ADC_2_data(ADC_data_bus_2),    //[7:0]
//    outputs to user
        .next_write_address(ram1_address_user),    //[10:0]
        .recorder_busy(recorder1_busy),
//    outputs to devices
        .ADC_request_to_device(ADC_request_1),
        .RAM_write_request_to_device(write_request_1),
        .RAM_write_data_to_device(ram1_data_user), //[16:0]
        .full(full_1)
//    .state_out(state_out)
    );

//*****

//*****

// ADC

//    ADC 1

    ADC_Interface adc_inter_1
    (
//    Global inputs
        .global_clock(clock_out),
        .global_reset(global_reset),
//    Local inputs
        .ADC_clock(ADC_clock),
        .ADC_request( (ADC_request_0 | ADC_request_1) ),
        .ADC_intr_bar(ADC_intr_bar_1),
        .ADC_data_bus_in(ADC_data_bus_1),
//    outputs
        .ADC_cs_bar(ADC_cs_bar_1),
        .ADC_rd_bar(ADC_rd_bar_1),
        .ADC_wr_bar(ADC_wr_bar_1),
        .ADC_data_bus_out(ADC_data_bus_out_1),
        .ADC_busy(ADC_busy_1)
    );

//    ADC 2

    ADC_Interface adc_inter_2
    (
//    Global inputs
        .global_clock(clock_out),
        .global_reset(global_reset),
//    Local inputs
        .ADC_clock(ADC_clock),
        .ADC_request( (ADC_request_0 | ADC_request_1) ),
        .ADC_intr_bar(ADC_intr_bar_2),
        .ADC_data_bus_in(ADC_data_bus_2),
//    outputs
        .ADC_cs_bar(ADC_cs_bar_2),
        .ADC_rd_bar(ADC_rd_bar_2),
        .ADC_wr_bar(ADC_wr_bar_2),
        .ADC_data_bus_out(ADC_data_bus_out_2),
        .ADC_busy(ADC_busy_2)
    );

```



```

//*****

//*****

// WIRE ASSIGNMENTS

// ADC 1

    assign ADC_intr_bar_1 = user3[11];
    assign ADC_data_bus_1[7:0] = (ADC_rd_bar_1 == 0) ? user3[7:0] :
    ADC_data_bus_1[7:0];
    //ADC_intr_bar is high z for input

    assign user3 = {20'hz, 1'bz, ADC_wr_bar_1, ADC_rd_bar_1, ADC_cs_bar_1,
    8'hz};

// ADC 2

    assign ADC_intr_bar_2 = user4[11];
    assign ADC_data_bus_2[7:0] = (ADC_rd_bar_2 == 0) ? user4[7:0] :
    ADC_data_bus_2[7:0];
    //ADC_intr_bar is high z for input

    assign user4 = {20'hz, 1'bz, ADC_wr_bar_2, ADC_rd_bar_2, ADC_cs_bar_2,
    8'hz};

assign global_reset = reset;
assign data0 = ram0_data_to_user;
assign data1 = ram1_data_to_user;
assign done_RAM0 = full_0;
assign done_RAM1 = full_1;
assign address_in0 = ram0_address_user;
assign address_in1 = ram1_address_user;
assign user1[31] = liar;

//*****

//*****

//
// Display Definitions
//
//*****

    wire [639:0]my_dots;
    wire [7:0]    char_0;
    wire         ascii_0;
    wire [7:0]    char_1;
    wire         ascii_1;
    wire [7:0]    char_2;
    wire         ascii_2;
    wire [7:0]    char_3;
    wire         ascii_3;
    wire [7:0]    char_4;
    wire         ascii_4;
    wire [7:0]    char_5;

```

```

wire      ascii_5;
wire [7:0] char_6;
wire      ascii_6;
wire [7:0] char_7;
wire      ascii_7;
wire [7:0] char_8;
wire      ascii_8;
wire [7:0] char_9;
wire      ascii_9;
wire [7:0] char_10;
wire      ascii_10;
wire [7:0] char_11;
wire      ascii_11;
wire [7:0] char_12;
wire      ascii_12;
wire [7:0] char_13;
wire      ascii_13;
wire [7:0] char_14;
wire      ascii_14;
wire [7:0] char_15;
wire      ascii_15;

assign char_0    = {1'b0, state_major_mm};
assign ascii_0 = 1'b0;
assign char_1    = {2'b0, state_minor0_mm};
assign ascii_1 = 1'b0;
assign char_2    = {2'b0, state_minor1_mm};
assign ascii_2 = 1'b0;
assign char_3    = {1'b0, state_major_ddmu};
assign ascii_3 = 1'b0;

//
assign char_4    = {2'b0, state_minor_coll};
assign ascii_4 = 1'b0;
assign char_5    = {3'b0, state_minor_proc};
assign ascii_5 = 1'b0;
assign char_6    = {3'b0, state_minor_store};
assign ascii_6 = 1'b0;
assign char_7    = {2'b0, state_minor_decide};
assign ascii_7 = 1'b0;

//
assign char_8    = address_in0[3:0];
assign ascii_8 = 1'b0;
assign char_9    = address_in1[3:0];
assign ascii_9 = 1'b0;
assign char_10   = user3[3:0];
assign ascii_10 = 1'b0;
assign char_11   = user3[7:4];
assign ascii_11 = 1'b0;

//
assign char_12   = user4[3:0];
assign ascii_12 = 1'b0;
assign char_13   = user4[7:4];
assign ascii_13 = 1'b0;
assign char_14   = {3'b0, store};
assign ascii_14 = 1'b0;
assign char_15   = {2'b0, question};
assign ascii_15 = 1'b0;

Dot_Matrix D_Mat(
    system_reset, clock_out,
    char_0, ascii_0, char_1, ascii_1, char_2, ascii_2, char_3, ascii_3,

```

```

        char_4,ascii_4,char_5,ascii_5,char_6,ascii_6,char_7,ascii_7,char_8,ascii_8,
        char_9,ascii_9,char_10,ascii_10,char_11,ascii_11,char_12,ascii_12,char_13,a
        scii_13,
            char_14,ascii_14,char_15,ascii_15,my_dots);

    alphanumeric_displays disp
    (
//      Inputs
        .global_clock(clock_out),
        .manual_reset(global_reset),
        .disp_test(1'b0),//butt1),
//      Outputs
        .disp_blank(disp_blank),
        .disp_clock(disp_clock),
        .disp_rs(disp_rs),
        .disp_ce_b(disp_ce_b),
        .disp_reset_b(disp_reset_b),
        .disp_data_out(disp_data_out),
//      Input
        .dots(my_dots)
    );
endmodule

```