Baby Monitoring System

- To monitor Vitals of a baby like temperature, heartbeat and motion from resp. Sensors
- The ckt processes the incoming sensor data, temporality stores it and automatically generates alerts if any abnormal reading are detected.
- i/ps :- ① clk ② rst
  ③ hb_data, temp_data, motion_data
  ④ hb_valid, temp_valid, motion_valid
  → (indicate sensor are ready to Read)
  → 8-bit sensor data
- o/ps :- ① alert_data :- latest value from memory that caused alert
  ② hb_alert, temp_alert, motion_alert
  (binary flag showing abnormal hb or temp or motion) ⇒ any one
- System Architecture
  ① Sensor interface module
  ⇒ Each 3 sensor is connected to its own interface
  - checks if valid data or not (valid = 1/0)
  - Transfers data fwd (data_out)
  - Generates a trigger signal when new signal arrives (trig_write)

=> so sensor module look like

```
module sensor_interface (
    input wire [7:0] sensor_data;
    input valid, clk,
    output reg trig_write;
    output reg [7:0] data out );
  always @(posedge clk) begin
    if (valid) begin      // valid = 1 ✓
      trig_write <= 1'b1;    // trig = 1
      data_out <= sensor_data;
    end
    else begin
      trig_write <= 1'b0;    // valid = 0, trig = 0
    end
  end
endmodule
```

(2) central SRAM memory (8 x 8)

=> Acts as a temporary data storage for all sensor.

=> contain 8 memory locatⁿ (each 8 bit wide)

=> stores latest sensor reading whenever a valid bit is received

=> Address ptr (addr) increments after every write opⁿ, allowing sequencial storage

=> when no write happens, it outputs the last stored value through (alert_data)

→ SRAM behaves as small logging memory

```
module sram_8x8 (
    input clk,
    input we,          // write-enable (control signal)
    input [7:0] data_in,
    input [2:0] address,
    output reg [7:0] data_out);

    reg [8:0] mem [8:0];
```

WE = 1
⇒ data_in stored in memory
WE = 0
⇒ memory is o/p to data_out

* { mem width — mem depth

```
    integer i,
    initial begin .
        for (i=0 ; i<8 ; i=i+1)
            mem[i] = 8'h00;       // hex 00 assign to mem
            data_out = 8'h00;     // mem is o/p as 00
    end                           or data_out is
                                  intialised
    always @(posedge clk) begin
        if (we) begin
            mem[address] <= data_in;
            data_out <= data_in;
        end
        else begin
            data_out <= mem[address]
        end
    end
endmodule
```

⇒ So, when we=1, data_in is stored in memory as well as comes as data_out
when we=0, memory comes as data_out

③ Alert Generation unit

① Heartbeat $\geq$ 120 bpm
② Temperature $\geq$ 100 °F
③ Motion $\geq$ 1

when conditⁿ met, corresponding alert flag is set (1) otherwise reset (0).

Verilog code :-

```
module baby-monitor-system (
  { input clk, rst,
  { input [7:0] hb_data, temp-data, motion_data;
  { input hb_valid, temp-valid, motion_valid;
  { output [7:0] alert_data,
  { output hb_alert, temp-alert, motion alert);

  { parameter hb_th = 8'd120;   // heartbeat threshold
  {     "     temp-th = 8'd100;
  {     "     motion_th = 8'd1;

for sensor interface
  { wire we-hb, we-temp, we-motion;   // trig.
  { wire [7:0] hb_out, temp-out, motion-out;
  { wire [7:0] sram_out;
  { reg  [2:0] addr

for SRAM
  { wire we-any;        // if any sensor alert
  { wire [8:0] data-to-write; // priority

alert
  { reg hb_alert_r, temp-alert_r, motion_alert_r;
    // for alert data
```

```
sensor_interface hb_sensor (.sensor_data(hb_data),
        .valid(hb_valid), .clk(clk), .trig-wrt(we_hb),
            .data_out(hb-out));
Sensor_interface temp-sensor (.sensor_data(temp-data),
        .valid(temp-valid), .clk(clk), .trig-wrt(we_temp),
            .data-out(temp-out));
Sensor_interface motion-sensor(.sensor-data(motion_data),
        .valid(motion_valid), .clk(clk); trigwrt(we_motion)
            .data-out(motion-out));
```

//module sensor-interface is called and values are assigned using Named Associatⁿ ⇒ .clk(clk)

```
assign we_any = we_hb | we_temp | we_motion;
assign data_to-write = we_hb ? hb_out :
            we_temp ? temp-out : motion-out;
```

nested ternary (? : ? : ---)

// we_any → if any sensor is true we_any = 1

// data_to-write → sets priority i.e if hb is true then hb-out, if not then if temp is true then temp-out, if not then motion-out.

```
3_sram_8x8 sram(.clk(clk), .data-in(data-to-wrt),
        .we(we_any), .address(addr),
        .data-out(sram-out));
```

// very crucial for SRAM assign.

```
always @(posedge clk or posedge rst) begin
  if(rst)
      addr<=0;
  else if (we_any)
      addr <= addr+1;   //addr increment
end

always @(posedge clk or posedge rst) begin
   if (rst)        // default or rst to 0
     hb_alert_r <=0; temp_alert_r <=0
     motion_alert_r <=0;
   end
   else begin      // for alert msg or(1)
     if(hb_valid && hb_data>hb_th)
        hb_alert_r <=1;
     else
        hb_alert_r <=0;
     if (temp_valid && temp_data > temp_th)
        temp_alert_r<=1;
     else
        temp_alert_r <=0;
     if (motion_valid && motion_data>motion_th)
        motion_alert_r <=1;
     else
        motion_alert_r <=0;
    end
end
assign hb_alert = hb_alert_r;
assign temp_alert = temp_alert_r;
assign motion_alert = motion_alert_r;
assign alert_data = sram_out;
endmodule
```

(4) operation Sequence :-

1. On Reset ⇒ all memories & alert are clr
2. Each sensor sends data when ready (through *_valid signals)
3. The system then picks high priority i/p (Hb ⇒ temp ⇒ motion)
4. Readings is stored in SRAM and address counter ↑ (advances)
5. Threshold logic check if reading exceed safe limit
6. If yes → alert_o/p goes High (1)
7. stored alert_data reflects the value currently causing or being checked for alert