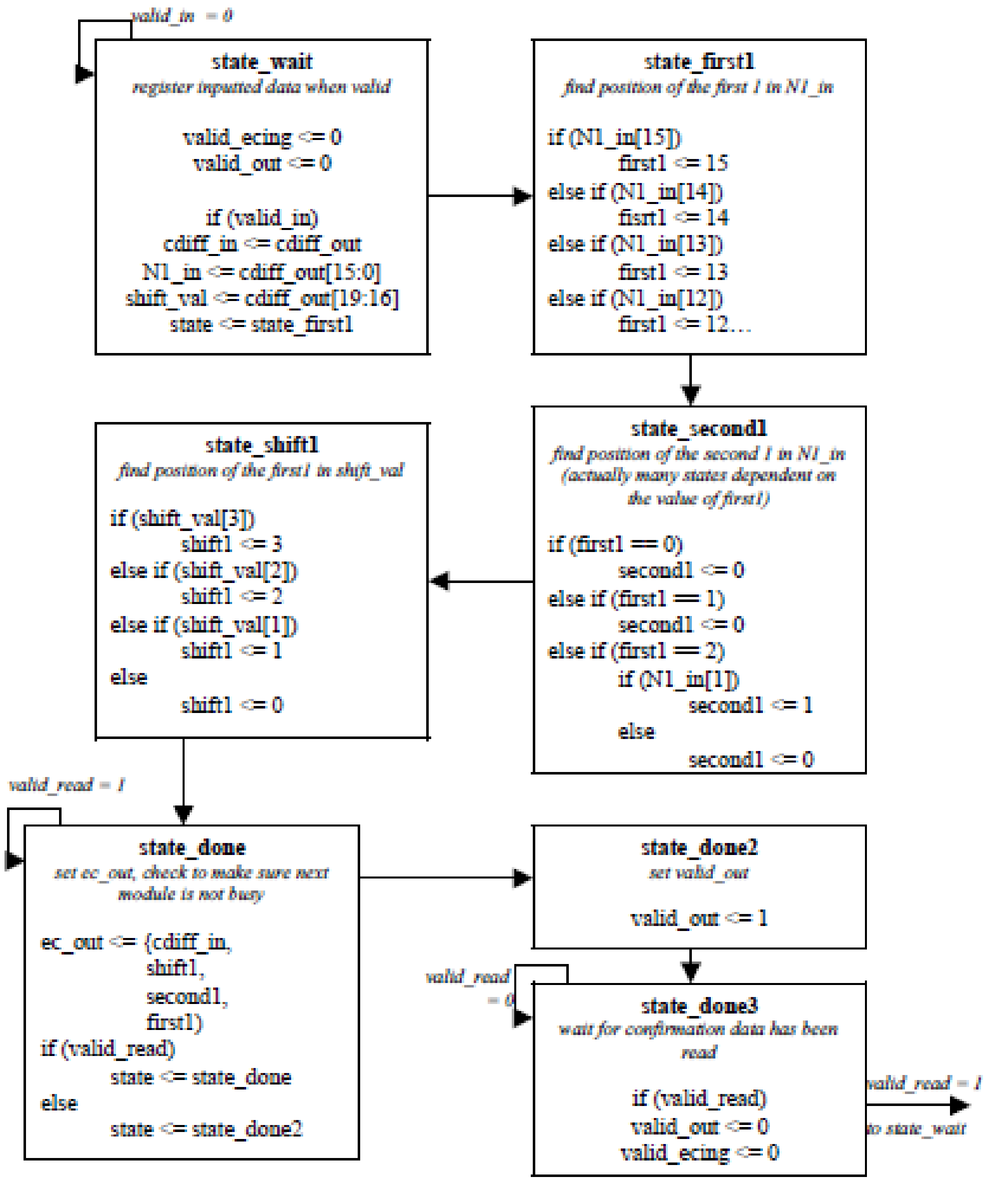
Appendix to Group1Report.pdf

ERROR CORRECTION INPUT



```
state_wait
register inputted data when valid

valid_ecing <= 0
valid_out <= 0

if (valid_in)
cdiff_in <= cdiff_out
N1_in <= cdiff_out[15:0]
shift_val <= cdiff_out[19:16]
state <= state_first1
```

```
state_first1
find position of the first 1 in N1_in

if (N1_in[15])
        first1 <= 15
else if (N1_in[14])
        fisrt1 <= 14
else if (N1_in[13])
        first1 <= 13
else if (N1_in[12])
        first1 <= 12…
```

```
state_second1
find position of the second 1 in N1_in
(actually many states dependent on
the value of first1)

if (first1 == 0)
        second1 <= 0
else if (first1 == 1)
        second1 <= 0
else if (first1 == 2)
        if (N1_in[1])
                second1 <= 1
        else
                second1 <= 0
```

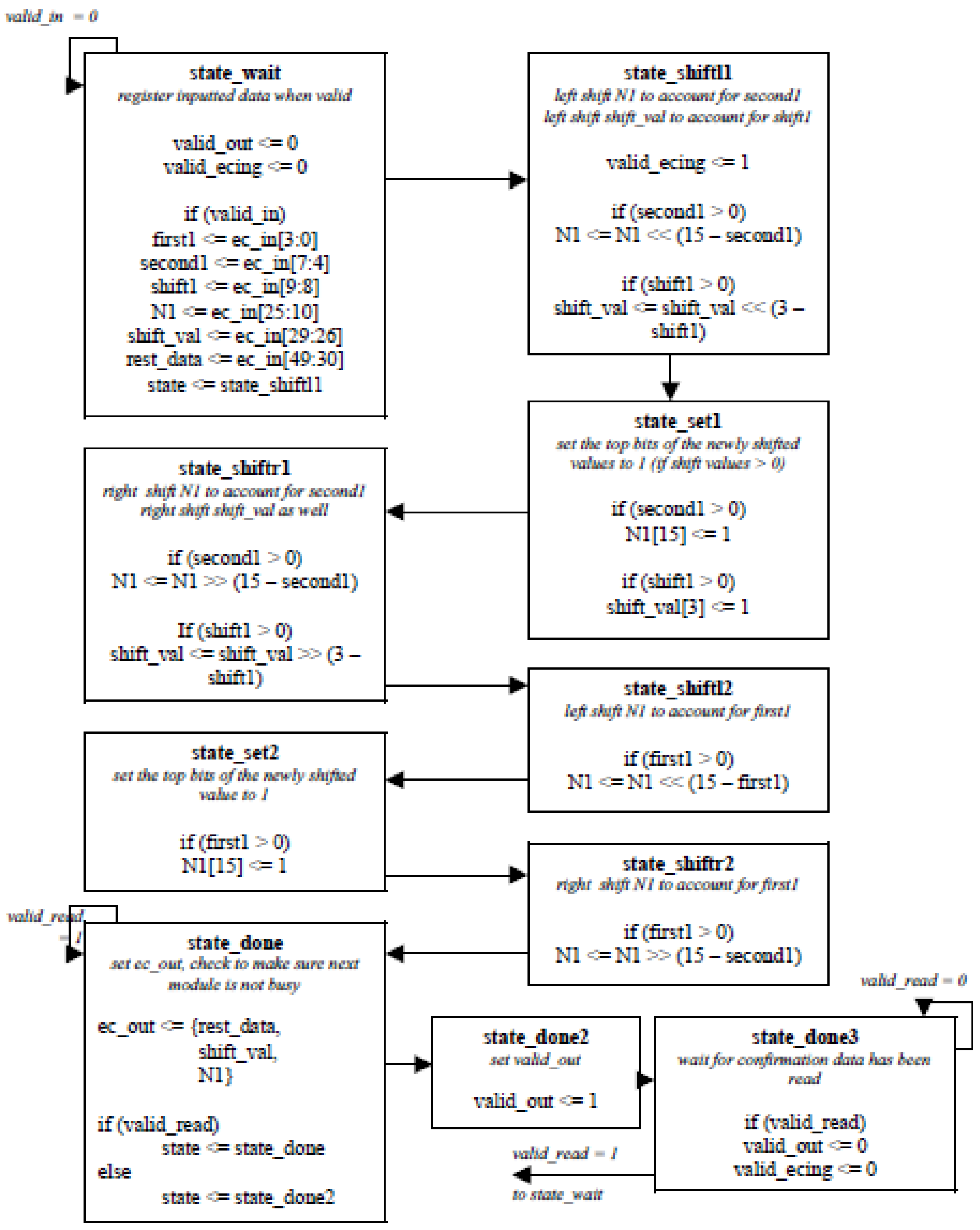
```
state_shift1
find position of the first1 in shift_val

if (shift_val[3])
        shift1 <= 3
else if (shift_val[2])
        shift1 <= 2
else if (shift_val[1])
        shift1 <= 1
else
        shift1 <= 0
```

```
state_done
set ec_out, check to make sure next
module is not busy

ec_out <= {cdiff_in,
           shift1,
           second1,
           first1}
if (valid_read)
        state <= state_done
else
        state <= state_done2
```

```
state_done2
set valid_out

valid_out <= 1
```

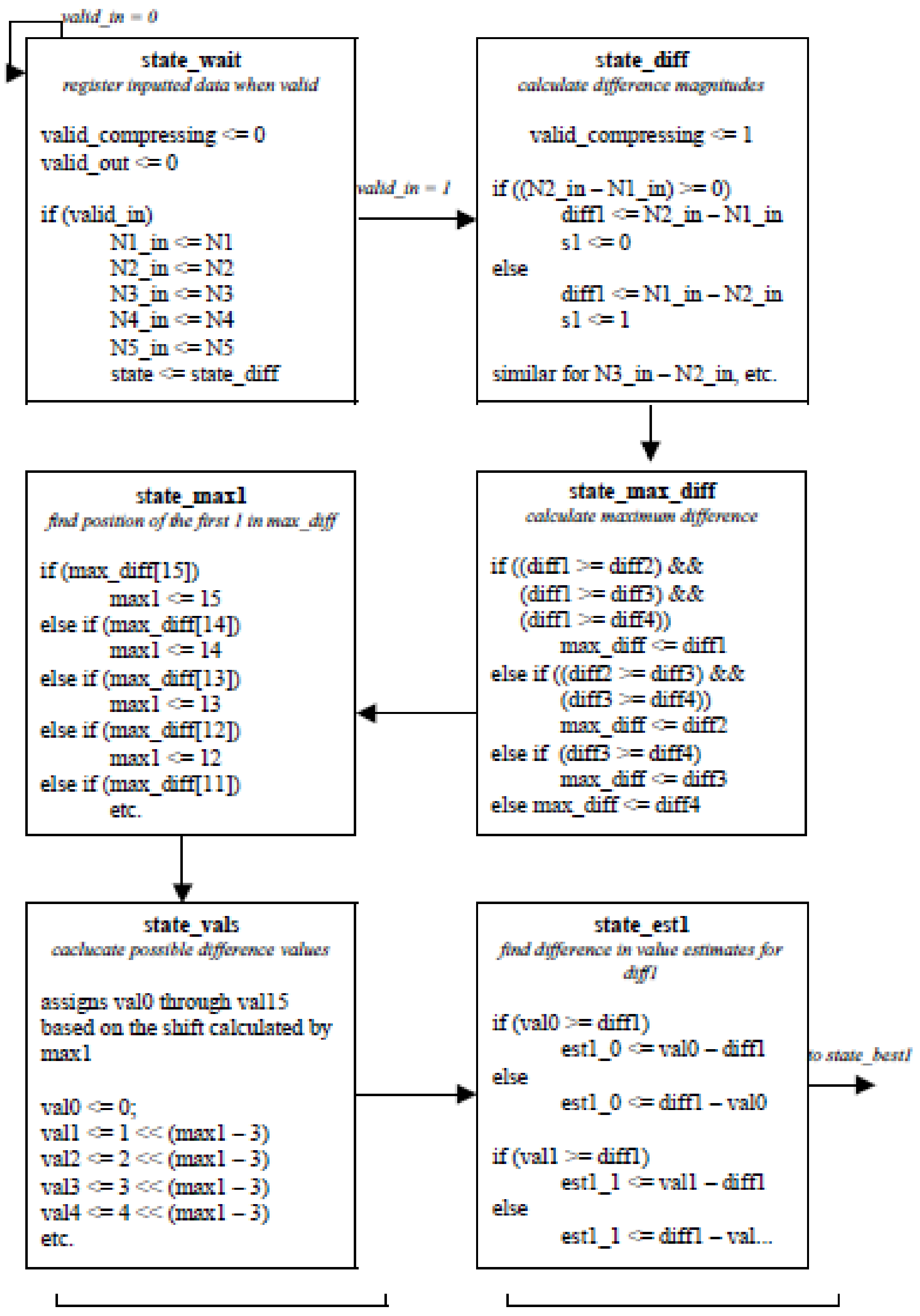
```
state_done3
wait for confirmation data has been
read

if (valid_read)
valid_out <= 0
valid_ecing <= 0
```

ERROR_CORRECTION_OUTPUT


valid_in = 0
state_wait
register inputted data when valid
state_shiftl1
left shift N1 to account for second1
left shift shift_val to account for shift1
state_set1
set the top bits of the newly shifted values to 1 (if shift values > 0)
state_shiftr1
right shift N1 to account for second1
right shift shift_val as well
state_shiftl2
left shift N1 to account for first1
state_set2
set the top bits of the newly shifted value to 1
state_shiftr2
right shift N1 to account for first1
valid_read = 1
state_done
set ec_out, check to make sure next module is not busy
state_done2
set valid_out
state_done3
wait for confirmation data has been read
valid_read = 0
valid_read = 1
to state_wait

```
state_wait:
valid_out <= 0
valid_ecing <= 0

if (valid_in)
first1 <= ec_in[3:0]
second1 <= ec_in[7:4]
shift1 <= ec_in[9:8]
N1 <= ec_in[25:10]
shift_val <= ec_in[29:26]
rest_data <= ec_in[49:30]
state <= state_shiftl1

state_shiftl1:
valid_ecing <= 1

if (second1 > 0)
N1 <= N1 << (15 – second1)

if (shift1 > 0)
shift_val <= shift_val << (3 –
shift1)

state_set1:
if (second1 > 0)
N1[15] <= 1

if (shift1 > 0)
shift_val[3] <= 1

state_shiftr1:
if (second1 > 0)
N1 <= N1 >> (15 – second1)

If (shift1 > 0)
shift_val <= shift_val >> (3 –
shift1)

state_shiftl2:
if (first1 > 0)
N1 <= N1 << (15 – first1)

state_set2:
if (first1 > 0)
N1[15] <= 1

state_shiftr2:
if (first1 > 0)
N1 <= N1 >> (15 – second1)

state_done:
ec_out <= {rest_data,
           shift_val,
           N1}

if (valid_read)
        state <= state_done
else
        state <= state_done2

state_done2:
valid_out <= 1

state_done3:
if (valid_read)
valid_out <= 0
valid_ecing <= 0
```

COMPRESSION



state_wait
register inputted data when valid

```
valid_compressing <= 0
valid_out <= 0

if (valid_in)
          N1_in <= N1
          N2_in <= N2
          N3_in <= N3
          N4_in <= N4
          N5_in <= N5
          state <= state_diff
```

state_diff
calculate difference magnitudes

```
    valid_compressing <= 1

if ((N2_in – N1_in) >= 0)
          diff1 <= N2_in – N1_in
          s1 <= 0
else
          diff1 <= N1_in – N2_in
          s1 <= 1

similar for N3_in – N2_in, etc.
```

state_max_diff
calculate maximum difference

```
if ((diff1 >= diff2) &&
    (diff1 >= diff3) &&
    (diff1 >= diff4))
          max_diff <= diff1
else if ((diff2 >= diff3) &&
          (diff3 >= diff4))
          max_diff <= diff2
else if  (diff3 >= diff4)
          max_diff <= diff3
else max_diff <= diff4
```

state_max1
find position of the first 1 in max_diff

```
if (max_diff[15])
          max1 <= 15
else if (max_diff[14])
          max1 <= 14
else if (max_diff[13])
          max1 <= 13
else if (max_diff[12])
          max1 <= 12
else if (max_diff[11])
          etc.
```

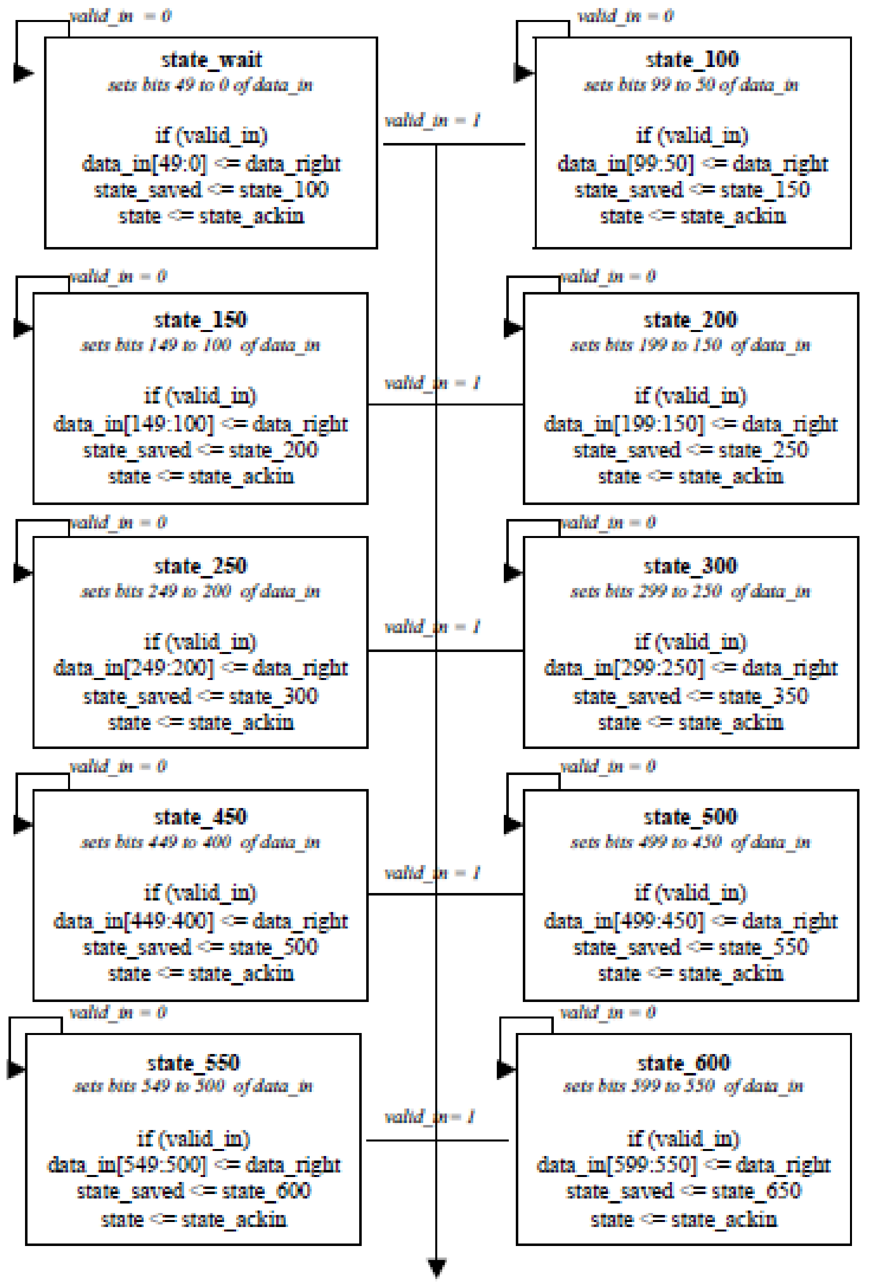
state_vals
caclucate possible difference values

```
assigns val0 through val15
based on the shift calculated by
max1

val0 <= 0;
val1 <= 1 << (max1 – 3)
val2 <= 2 << (max1 – 3)
val3 <= 3 << (max1 – 3)
val4 <= 4 << (max1 – 3)
etc.
```

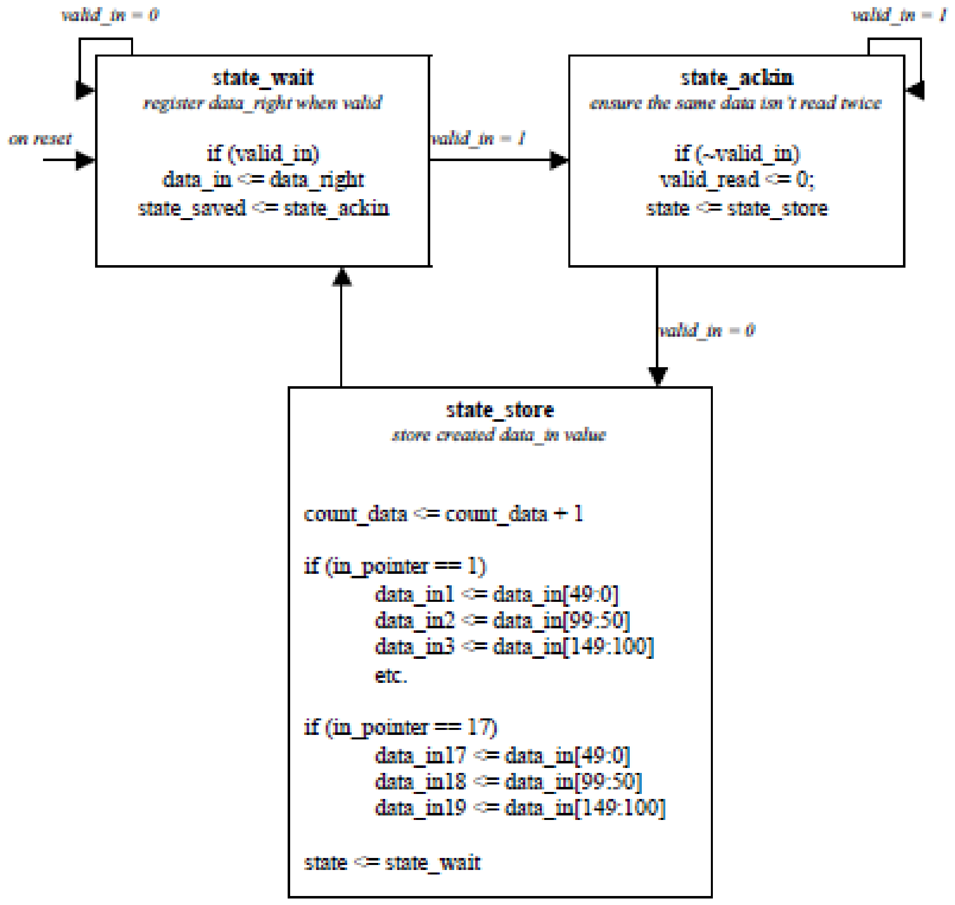
state_est1
find difference in value estimates for diff1

```
if (val0 >= diff1)
          est1_0 <= val0 – diff1
else
          est1_0 <= diff1 – val0

if (val1 >= diff1)
          est1_1 <= val1 – diff1
else
          est1_1 <= diff1 – val...
```

BUFFER_ADITI



BUFFER_NIVEDITA

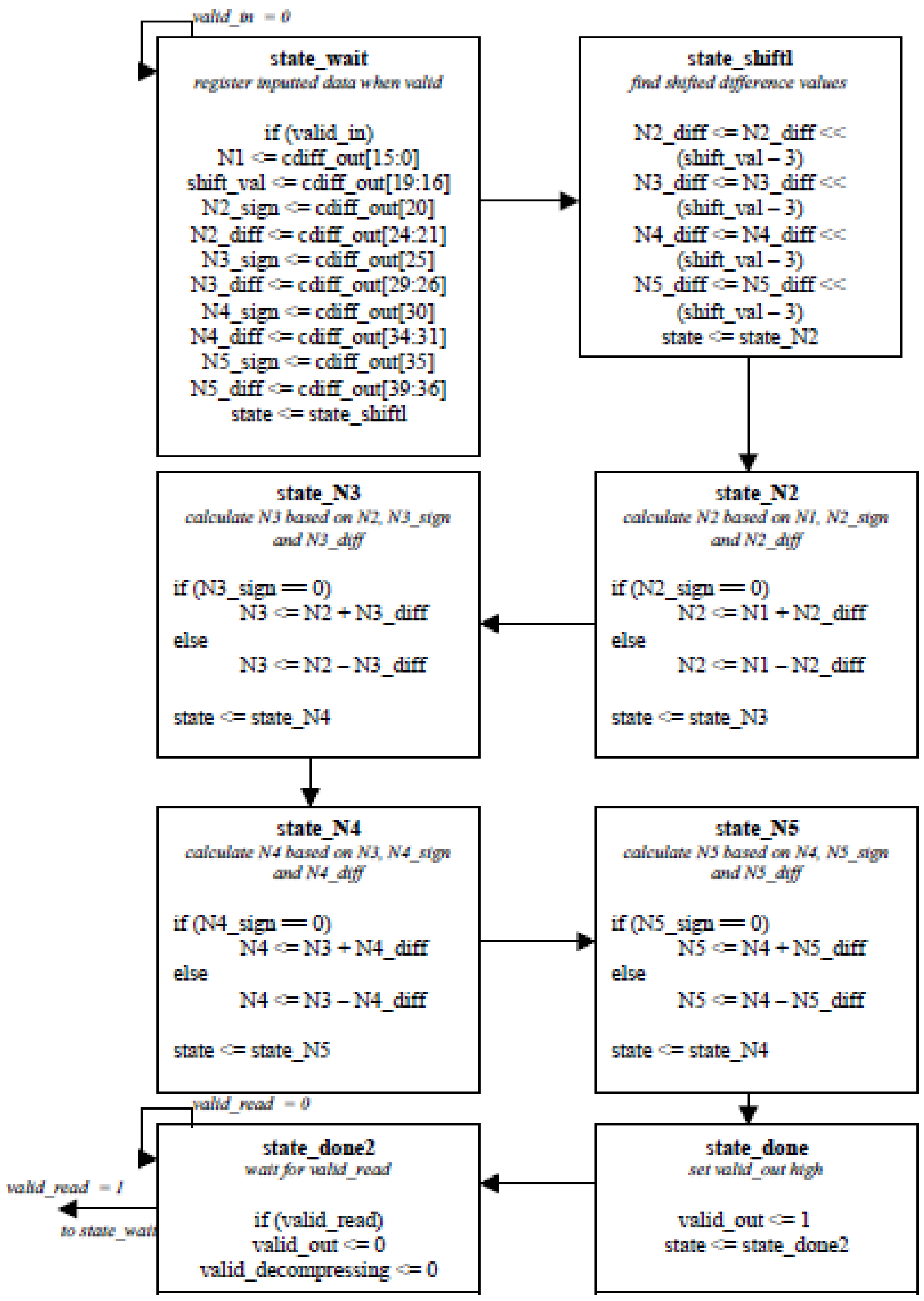

valid_in = 0
valid_in = 1
on reset
valid_in = 1
valid_in = 0
state_wait
register data_right when valid
state_ackin
ensure the same data isn't read twice
state_store
store created data_in value

```
if (valid_in)
data_in <= data_right
state_saved <= state_ackin
```

```
if (~valid_in)
valid_read <= 0;
state <= state_store
```

```
count_data <= count_data + 1

if (in_pointer == 1)
        data_in1 <= data_in[49:0]
        data_in2 <= data_in[99:50]
        data_in3 <= data_in[149:100]
        etc.

if (in_pointer == 17)
        data_in17 <= data_in[49:0]
        data_in18 <= data_in[99:50]
        data_in19 <= data_in[149:100]

state <= state_wait
```

Decompression



```
`timescale 1ns / 1ps

// Module: decompression

// Inputs:

        // clock: internal clock of the system
```

```
     // reset: zeros the outputs, places fsm in the wait state
      // valid_in: when high, registers cdiff_out.  Note that once valid_in
      //            becomes low the values of cdiff_out can be changed without affecting
      //            the system.
      // cdiff_out: compressed input to be decompressed, a 40-bit number representing five
      //             signed 16-bit numbers
// Outputs:
      // N1: signed 16-bit integer
      // N2: signed 16-bit integer
      // N3: signed 16-bit integer
      // N4: signed 16-bit integer
      // N5: signed 16-bit integer
      // valid_out: high when the final_decompression module is done decompressing, when
      //             the output is valid and readable.
      // valid_decompressing: high when the final_decompression module is decompressing data
      //                       while this is high, it will not register any new data in
      //                       even if valid_in becomes high.
// Notes:
      // 1.) decompression is just a decompression module.  It will not account for
      //     any added error checking mechanisms.
      // 2.) once it has taken in a cdiff_out value it will not take in any more values,
      //     until it has either a.) finished decompressing and recieved confirmation that
      //     the output has been read or b.) been reset.
      // 3.) here are a couple test cases to run to verify correct operation:
      //
      //     input: cdiff_out = 40'b0000_0_0000_0_0000_0_0000_0_0000_0000000000000001
      //     expected output: N1 = 1, N2 = 1, N3 = 1, N4 = 1, N5 = 1
   //
   //     input: cdiff_out = 40'b0100_0_1001_1_0010_0_0101_0_0100_1111111111111011
      //     expected output: N1 = -5, N2 = 5, N3 = 9, N4 = -9, N5 = -1
module decompression (clock, reset, valid_in, cdiff_out,
                           N1, N2, N3, N4, N5, valid_out, valid_decompressing);
    input  clock;
    input  reset;
       input  valid_in;
    input  [39:0] cdiff_out;
    output [15:0] N1;
    output [15:0] N2;
    output [15:0] N3;
    output [15:0] N4;
    output [15:0] N5;
    output valid_out;
       output valid_decompressing;
      // registered values of the inputs
      reg [15:0] N1_in;
      reg [ 3:0] shift_val;
      reg [15:0] N2_diff;
      reg [15:0] N3_diff;
      reg [15:0] N4_diff;
      reg [15:0] N5_diff;
      reg N2_sign;

      reg N3_sign;

      reg N4_sign;

      reg N5_sign;

      // module outputs

      reg signed [15:0] N1;

      reg signed [15:0] N2;

      reg signed [15:0] N3;

      reg signed [15:0] N4;

      reg signed [15:0] N5;

      reg valid_out;

      reg valid_decompressing;

      // internal variable to keep

      // track of the state of the fsm

      reg [3:0] state;

      // fsm states

      parameter state_wait   = 0;

      parameter state_shiftl = 1;

      parameter state_N2     = 2;

      parameter state_N3     = 3;

      parameter state_N4     = 4;

      parameter state_N5     = 5;

      parameter state_done   = 6;

      parameter state_done2  = 7;

      always @ (posedge clock)

            if (reset)

            // zero all outputs during a reset
            // after a reset, the fsm will always return

            // to state_wait.

                  begin

                  N1 <= 0;

                  N2 <= 0;

                  N3 <= 0;

                  N4 <= 0;

                  N5 <= 0;

                  valid_out           <= 0;

                  valid_decompressing <= 0;

                  state               <= state_wait;

                  end

            else

            case(state)

                  state_wait:

                  // not decompressing anything, waiting for

                  // valid_in to become high at which point

                  // the inputs are registered and the state

                  // transitions to state_shiftl.

                        begin

                        valid_out <= 0;

                        valid_decompressing <= 0;

                        if (valid_in)

                              begin

                              N1        <= cdiff_out[15:0];

                              shift_val <= cdiff_out[19:16];

                              N2_sign   <= cdiff_out[20];
                              N2_diff   <= cdiff_out[24:21];
                              N3_sign   <= cdiff_out[25];
                              N3_diff   <= cdiff_out[29:26];
                              N4_sign   <= cdiff_out[30];
                              N4_diff   <= cdiff_out[34:31];
                              N5_sign   <= cdiff_out[35];
                              N5_diff   <= cdiff_out[39:36];
                              state     <= state_shiftl;
                              end
                        else
                              state <= state_wait;
                        end
                  state_shiftl:
                        // find the shifted values of N2_diff, N3_diff, etc.
                        // this is done by left shifting them by (shift_val - 3).
                        // if shift_val <= 2, the difference values do not
                        // need to be shifted.
                        begin
                        valid_decompressing <= 1;
                        if (shift_val > 2)
                              begin
                              N2_diff <= N2_diff << (shift_val - 3);
                              N3_diff <= N3_diff << (shift_val - 3);
                              N4_diff <= N4_diff << (shift_val - 3);
                              N5_diff <= N5_diff << (shift_val - 3);
                              end
                        else
                              begin
                              // no shifting necessary in this case.
                              end
                        state <= state_N2;
                        end
                  state_N2:
                        // calculate N2 based on N1, N2_sign
                        // and the new N2_diff value that was
                        // determined in the previous state.
                        if (N2_sign == 0)
                              begin
                              N2 <= N1 + N2_diff;
                              state <= state_N3;
                              end
                        else
                              begin
                              N2 <= N1 - N2_diff;
                              state <= state_N3;
                              end
                  state_N3:
                        // calculate N3 based on N2, N3_sign
                        // and N3_diff
                        if (N3_sign == 0)
                              begin
                              N3 <= N2 + N3_diff;
                              state <= state_N4;
                              end
```

```
        else
              begin
              N3 <= N2 - N3_diff;
              state <= state_N4;
              end
  state_N4:
        // calculate N4 based on N3, N4_sign
        // and N4_diff
        if (N4_sign == 0)
              begin
              N4 <= N3 + N4_diff;
              state <= state_N5;
              end
        else
              begin
              N4 <= N3 - N4_diff;
              state <= state_N5;
              end
  state_N5:
        // calculate N5 based on N4, N5_sign
        // and N5_diff
        if (N5_sign == 0)
              begin
              N5 <= N4 + N5_diff;
              state <= state_done;
              end
        else
```

```
                              begin

                              N5 <= N4 - N5_diff;

                              state <= state_done;

                              end

                  state_done:

                        // set the valid_out signal high

                        begin

                        valid_out <= 1;

                        state <= state_wait; // state <= state_done2;

                        end

                  state_done2:

                        state <= state_wait;

            endcase

endmodule
```

```
timescale 1ns / 1ps
//Module: compression

// Inputs:

      // clock: internal clock of the system

      // reset: zeros the outputs, places fsm in the wait state

      // valid_in: when high, registers N1, N2, N3, N4, N5.  Note that once valid_in

      //           becomes low the values of the N's can be changed without affecting

      //           the system.

      // valid_read: input from the module following this.  Will go high when the

      //             next module has read cdiff_out.  In this way the final_compression

      //             module will know it is okay to move on to compressing new data.

      // N1: signed 16 bit integer
      // N2: signed 16 bit integer

      // N3: signed 16 bit integer

      // N4: signed 16 bit integer

      // N5: signed 16 bit integer

// Outputs:

      // valid_compressing: high when the final_compression module is compressing data

      //                    while this is high, it will not register any new data in

      //                    even if valid_in becomes high.

      // valid_out: high when the final_compression module is done compressing, when

      //            the output is valid and readable.

      // cdiff_out: 14 bit output of the system.  Arranged as follows:

      //            cdiff[15:0]  = N1

      //            cdiff[19:16] = shift value, the amount by which the following

      //                           differences are to be shifted + 3.

      //            cdiff[20]    = sign of the first difference (N2 - N1)

      //            cdiff[24:21] = magnitude of the first difference

      //            cdiff[25]    = sign of the second difference (N3 - N2)

      //            cdiff[29:26] = magnitude of the second difference

      //            cdiff[30]    = sign of the third difference (N4 - N3)

      //            cdiff[34:31] = magnitude of the third difference

      //            cdiff[35]    = sign of the fourth difference (N5 - N4)

      //            cdiff[39:36] = magnitude of the fourth difference

// Notes:

      // 1.) compression is just a compression module.  It provides no error

      //     correction scheme for wireless errors.

      // 2.) once it has taken in N1, N2, etc. it will not take in any more values,

      //     until it has either a.) finished compressing and recieved confirmation that
      //     the output has been read or b.) been reset.

      // 3.) here are a couple test cases to run to verify correct operation:

      //

      //     input: N1 = 1, N2 = 1, N3 = 1, N4 = 1, N5 = 1

      //     expected output: 0000_0_0000_0_0000_0_0000_0_0000_0000000000000001

   //

   //     input: N1 = -5, N2 = 5, N3 = 10, N4 = -10, N5 = 0

      //     expected output: 0100_0_1001_1_0010_0_0101_0_0100_1111111111111011

module compression(clock, reset, valid_in, valid_read, N1, N2, N3, N4, N5,

                         valid_compressing, valid_out, cdiff_out);

    input  clock;

    input  reset;

    input  valid_in;

    input  valid_read;

    input  signed [15:0] N1;

    input  signed [15:0] N2;

    input  signed [15:0] N3;

    input  signed [15:0] N4;

    input  signed [15:0] N5;

       output valid_compressing;

    output valid_out;

    output [39:0] cdiff_out;


       // registered values of the inputs

       reg signed [15:0] N1_in;

       reg signed [15:0] N2_in;

       reg signed [15:0] N3_in;
       reg signed [15:0] N4_in;

       reg signed [15:0] N5_in;


       // estimated values of the inputs

       // they are calculated as the respective difference

       // values are calculated.

       // i.e N2_est = N1 + diff1 or N1 - diff1

       reg signed [15:0] N2_est;

       reg signed [15:0] N3_est;

       reg signed [15:0] N4_est;


       // difference values, given as magnitudes.

       // diff1 = mag of diff between N2 and N1

       // diff2 = mag of diff between N3 and N2_est

       // diff3 = mag of diff between N4 and N3_est

       // diff4 = mag of diff between N5 and N4_est

       // Note that the estimated values are used for finding

       // differences to try to prevent compounding errors.

       reg [15:0] diff1;

       reg [15:0] diff2;

       reg [15:0] diff3;

       reg [15:0] diff4;


       // Sign values, the value of the sign of the

       // associated difference magnitude.

       // 0 corresponds to positive.

       // 1 corresponds to negative.
```

```
reg s1;

reg s2;

reg s3;

reg s4;


// max_diff = maximum difference value

// max1 = position of the first 1 (base 0 counting)

// in the number max_diff.

reg [15:0] max_diff;

reg [3:0]  max1;


// the 16 possible values that can be coded for

// given the value of max1

//  i.e. val1 == 0001 << (max1 - 3) or just 0001 if max1 < 2

//        val2 == 0010 << (max1 - 3) or just 0010 if max1 < 2

//        etc.

// note that val0 will always be equal to zero

reg [15:0] val0;

reg [15:0] val1;

reg [15:0] val2;

reg [15:0] val3;

reg [15:0] val4;

reg [15:0] val5;

reg [15:0] val6;

reg [15:0] val7;

reg [15:0] val8;

reg [15:0] val9;
reg [15:0] val10;

reg [15:0] val11;

reg [15:0] val12;

reg [15:0] val13;

reg [15:0] val14;

reg [15:0] val15;

// magnitudes of the difference between the val

// estimates and diff1

reg [15:0] est1_0;

reg [15:0] est1_1;

reg [15:0] est1_2;

reg [15:0] est1_3;

reg [15:0] est1_4;

reg [15:0] est1_5;

reg [15:0] est1_6;

reg [15:0] est1_7;

reg [15:0] est1_8;

reg [15:0] est1_9;

reg [15:0] est1_10;

reg [15:0] est1_11;

reg [15:0] est1_12;

reg [15:0] est1_13;

reg [15:0] est1_14;

reg [15:0] est1_15;

// magnitudes of the difference between the val

// estimates and diff2

reg [15:0] est2_0;
reg [15:0] est2_1;

reg [15:0] est2_2;

reg [15:0] est2_3;

reg [15:0] est2_4;

reg [15:0] est2_5;

reg [15:0] est2_6;

reg [15:0] est2_7;

reg [15:0] est2_8;

reg [15:0] est2_9;

reg [15:0] est2_10;

reg [15:0] est2_11;

reg [15:0] est2_12;

reg [15:0] est2_13;

reg [15:0] est2_14;

reg [15:0] est2_15;

// magnitudes of the difference between the val

// estimates and diff3

reg [15:0] est3_0;

reg [15:0] est3_1;

reg [15:0] est3_2;

reg [15:0] est3_3;

reg [15:0] est3_4;

reg [15:0] est3_5;

reg [15:0] est3_6;

reg [15:0] est3_7;

reg [15:0] est3_8;

reg [15:0] est3_9;
reg [15:0] est3_10;

reg [15:0] est3_11;

reg [15:0] est3_12;

reg [15:0] est3_13;

reg [15:0] est3_14;

reg [15:0] est3_15;

// magnitudes of the difference between the val

// estimates and diff4

reg [15:0] est4_0;

reg [15:0] est4_1;

reg [15:0] est4_2;

reg [15:0] est4_3;

reg [15:0] est4_4;

reg [15:0] est4_5;

reg [15:0] est4_6;

reg [15:0] est4_7;

reg [15:0] est4_8;

reg [15:0] est4_9;

reg [15:0] est4_10;

reg [15:0] est4_11;

reg [15:0] est4_12;

reg [15:0] est4_13;

reg [15:0] est4_14;

reg [15:0] est4_15;


// the best estimate of diff1, diff2, diff3,

// and diff4 respectivly.  Note that these
// are four bit values that represent the

// given number shifted to the left by (max1 - 3)

reg [3:0] best1;

reg [3:0] best2;

reg [3:0] best3;

reg [3:0] best4;

// system outputs

reg[39:0] cdiff_out;

reg valid_out;

reg valid_compressing;


// internal variable to track the state

// of the fsm

reg [4:0] state;


// enumeration of the states of the internal fsm.

parameter state_wait        = 0;

parameter state_diff        = 1;

parameter state_max_diff    = 2;

parameter state_max1        = 3;

parameter state_vals        = 4;

parameter state_est1        = 5;

parameter state_est2        = 6;

parameter state_est3        = 7;

parameter state_est4        = 8;

parameter state_best1       = 9;

parameter state_best2       = 10;
```

```
       parameter state_best3       = 11;

       parameter state_best4       = 12;

       parameter state_recal_diff2 = 13;

       parameter state_recal_diff3 = 14;

       parameter state_recal_diff4 = 15;

       parameter state_N2_est      = 16;

       parameter state_N3_est      = 17;

       parameter state_N4_est      = 18;

       parameter state_done        = 19;

       parameter state_done2       = 20;

       parameter state_done3       = 21;


always @ (posedge clock)

      if (reset)

            // zero all outputs during a reset

            // after a reset, the fsm will always return

            // to state_wait.

            begin

            cdiff_out         <= 0;

            valid_out         <= 0;

            valid_compressing <= 0;

            state             <= state_wait;

            end

      else

            case(state)

            state_wait:

                  // not compressing anything, waiting for
                  // valid_in to become high at which point
                  // the inputs are registered.
                  begin
                  valid_compressing <= 0;
                  valid_out         <= 0;
                  if (valid_in)
                        begin
                        N1_in <= N1;
                        N2_in <= N2;
                        N3_in <= N3;
                        N4_in <= N4;
                        N5_in <= N5;
                        state <= state_diff;
                        end
                  else
                        state <= state_wait;
                  end
            state_diff:
                  // assign initial differences and signs.
                  // Note that the differences are initial
                  // as they may be overridden later.  They
                  // are overridden to take into account the
                  // current values of N2_est, N3_est, etc.
                  // The actual compression process starts
                  // here so valid_compressing is set high.
                  begin
                  valid_compressing <= 1;
```

```
// We want the diff values to represent
// magnitudes, thus the if statements here
// ensure that the diff values are always
// positive.
if ((N2_in - N1_in) >= 0)
      begin
      diff1 <= N2_in - N1_in;
      s1    <= 0;
      end
else
      begin
      diff1 <= N1_in - N2_in;
      s1    <= 1;
      end
if ((N3_in - N2_in) >= 0)
      begin
      diff2 <= N3_in - N2_in;
      s2    <= 0;
      end
else
      begin
      diff2 <= N2_in - N3_in;
      s2    <= 1;
      end
if ((N4_in - N3_in) >= 0)
      begin
      diff3 <= N4_in - N3_in;
```

```
            s3    <= 0;

            end

        else

            begin

            diff3 <= N3_in - N4_in;

            s3    <= 1;

            end

        if ((N5_in - N4_in) >= 0)

            begin

            diff4 <= N5_in - N4_in;

            s4    <= 0;

            end

        else

            begin

            diff4 <= N4_in - N5_in;

            s4    <= 1;

            end

        state  <= state_max_diff;

        end

    state_max_diff:

        // the maximum difference is found

        begin

        if ((diff1 >= diff2) && (diff1 >= diff3) && (diff1 >= diff4))

            max_diff <= diff1;

        else if ((diff2 >= diff3) && (diff2 >= diff4))

            max_diff <= diff2;

        else if (diff3 >= diff4)
```

```
            max_diff <= diff3;
      else
            max_diff <= diff4;
      state <= state_max1;
      end
state_max1:
      // find the position of the first 1 in max_diff
      // this represents 3 greater than the shift value
      begin
            if (max_diff[15])
                  max1 <= 15;
            else if (max_diff[14])
                  max1 <= 14;
            else if (max_diff[13])
                  max1 <= 13;
            else if (max_diff[12])
                  max1 <= 12;
            else if (max_diff[11])
                  max1 <= 11;
            else if (max_diff[10])
                  max1 <= 10;
            else if (max_diff[9])
                  max1 <= 9;
            else if (max_diff[8])
                  max1 <= 8;
            else if (max_diff[7])
                  max1 <= 7;
            else if (max_diff[6])
                  max1 <= 6;
            else if (max_diff[5])
                  max1 <= 5;
            else if (max_diff[4])
                  max1 <= 4;
            else if (max_diff[3])
                  max1 <= 3;
            else if (max_diff[2])
                  max1 <= 2;
            else if (max_diff[1])
                  max1 <= 1;
            else
                  max1 <= 0;
            state <= state_vals;
            end
state_vals:
            // calculate the possible difference values
            // these are simply the integers 0 through 15
            // shifted by the appropriate value as determined
            // by max1.
            begin
            if (max1 > 2)
                  begin
                  val0  <= 0;
                  val1  <= 1  << (max1-3);
                  val2  <= 2  << (max1-3);
                  val3  <= 3  << (max1-3);

                  val4  <= 4  << (max1-3);

                  val5  <= 5  << (max1-3);

                  val6  <= 6  << (max1-3);

                  val7  <= 7  << (max1-3);

                  val8  <= 8  << (max1-3);

                  val9  <= 9  << (max1-3);

                  val10 <= 10 << (max1-3);

                  val11 <= 11 << (max1-3);

                  val12 <= 12 << (max1-3);

                  val13 <= 13 << (max1-3);

                  val14 <= 14 << (max1-3);

                  val15 <= 15 << (max1-3);

                  end

            else

                  // if max1 <= 2, no shift is needed.

                  begin

                  val0  <= 0;  // 0

                  val1  <= 1;  // 1

                  val2  <= 2;  // 10

                  val3  <= 3;  // 11

                  val4  <= 4;  // 100

                  val5  <= 5;  // 101

                  val6  <= 6;  // 110

                  val7  <= 7;  // 111

                  val8  <= 8;  // 1000

                  val9  <= 9;  // 1001
```

```
                        val10 <= 10; // 1010

                        val11 <= 11; // 1011

                        val12 <= 12; // 1100

                        val13 <= 13; // 1101

                        val14 <= 14; // 1110

                        val15 <= 15; // 1111

                        end

                state <= state_est1;

                end

        state_est1:

                // find the difference in value estimates

                // for diff1

                // i.e. take each val and find the difference

                // between it and diff1.  Note that these

                // are unsigned quantities and only represent

                // the magnitude of the difference.

                begin

                if (val0 >= diff1)

                        est1_0 <= val0 - diff1;

                        else

                        est1_0 <= diff1 - val0;

                if (val1 >= diff1)

                        est1_1 <= val1 - diff1;

                        else

                        est1_1 <= diff1 - val1;

                if (val2 >= diff1)

                        est1_2 <= val2 - diff1;
```

```
      else

      est1_2 <= diff1 - val2;

if (val3 >= diff1)

      est1_3 <= val3 - diff1;

      else

      est1_3 <= diff1 - val3;

if (val4 >= diff1)

      est1_4 <= val4 - diff1;

      else

      est1_4 <= diff1 - val4;

if (val5 >= diff1)

      est1_5 <= val5 - diff1;

      else

      est1_5 <= diff1 - val5;

if (val6 >= diff1)

      est1_6 <= val6 - diff1;

      else

      est1_6 <= diff1 - val6;

if (val7 >= diff1)

      est1_7 <= val7 - diff1;

      else

      est1_7 <= diff1 - val7;

if (val8 >= diff1)

      est1_8 <= val8 - diff1;

      else

      est1_8 <= diff1 - val8;

if (val9 >= diff1)
```

```
      est1_9 <= val9 - diff1;

      else

      est1_9 <= diff1 - val9;

if (val10 >= diff1)

      est1_10 <= val10 - diff1;

      else

      est1_10 <= diff1 - val10;

if (val11 >= diff1)

      est1_11 <= val11 - diff1;

      else

      est1_11 <= diff1 - val11;

if (val12 >= diff1)

      est1_12 <= val12 - diff1;

      else

      est1_12 <= diff1 - val12;

if (val13 >= diff1)

      est1_13 <= val13 - diff1;

      else

      est1_13 <= diff1 - val13;

if (val14 >= diff1)

      est1_14 <= val14 - diff1;

      else

      est1_14 <= diff1 - val14;

if (val15 >= diff1)

      est1_15 <= val15 - diff1;

      else

      est1_15 <= diff1 - val15;
```

```
            state <= state_best1;
            end
        state_best1:
            // find the best estimate of N2 - N1
            // this corresponds to finding the smallest
            // est1 value.
            begin
            if ((est1_0 <= est1_1)  && (est1_0 <= est1_2)  && (est1_0 <= est1_3)  &&
                (est1_0 <= est1_4)  && (est1_0 <= est1_5)  && (est1_0 <= est1_6)  &&
                  (est1_0 <= est1_7)  && (est1_0 <= est1_8)  && (est1_0 <= est1_9)  &&
                  (est1_0 <= est1_10) && (est1_0 <= est1_11) && (est1_0 <= est1_12) &&
                  (est1_0 <= est1_13) && (est1_0 <= est1_14) && (est1_0 <= est1_15))
                best1 <= 0;
            else if ((est1_1 <= est1_2)  && (est1_1 <= est1_3)  && (est1_1 <= est1_4)  &&
                   (est1_1 <= est1_5)  && (est1_1 <= est1_6)  && (est1_1 <= est1_7)  &&
                         (est1_1 <= est1_8)  && (est1_1 <= est1_9)  && (est1_1 <=
est1_10) &&
                         (est1_1 <= est1_11) && (est1_1 <= est1_12) && (est1_1 <=
est1_13) &&
                         (est1_1 <= est1_14) && (est1_1 <= est1_15))
                best1 <= 1;
            else if ((est1_2 <= est1_3)  && (est1_2 <= est1_4)  && (est1_2 <= est1_5)  &&
                         (est1_2 <= est1_6)  && (est1_2 <= est1_7)  && (est1_2 <=
est1_8)  &&
                         (est1_2 <= est1_9)  && (est1_2 <= est1_10) && (est1_2 <=
est1_11) &&
                         (est1_2 <= est1_12) && (est1_2 <= est1_13) && (est1_2 <=
est1_14) &&
                         (est1_2 <= est1_15))
```

```
                        best1 <= 2;
                  else if ((est1_3 <= est1_4)  && (est1_3 <= est1_5)  && (est1_3 <= est1_6)  &&
                           (est1_3 <= est1_7)  && (est1_3 <= est1_8)  && (est1_3 <= est1_9)  &&
                                  (est1_3 <= est1_10) && (est1_3 <= est1_11) && (est1_3 <=
est1_12) &&
                                  (est1_3 <= est1_13) && (est1_3 <= est1_14) && (est1_3 <=
est1_15))
                        best1 <= 3;
                  else if ((est1_4 <= est1_5)  && (est1_4 <= est1_6)  && (est1_4 <= est1_7)  &&
                           (est1_4 <= est1_8)  && (est1_4 <= est1_9)  && (est1_4 <= est1_10) &&
                                  (est1_4 <= est1_11) && (est1_4 <= est1_12) && (est1_4 <=
est1_13) &&
                                  (est1_4 <= est1_14) && (est1_4 <= est1_15))
                        best1 <= 4;
                  else if ((est1_5 <= est1_6)  && (est1_5 <= est1_7)  && (est1_5 <= est1_8)  &&
                              (est1_5 <= est1_9)  && (est1_5 <= est1_10) && (est1_5 <= est1_11)
&&
                                  (est1_5 <= est1_12) && (est1_5 <= est1_13) && (est1_5 <=
est1_14) &&
                                  (est1_5 <= est1_15))
                        best1 <= 5;
                  else if ((est1_6 <= est1_7)  && (est1_6 <= est1_8)  && (est1_6 <= est1_9)  &&
                           (est1_6 <= est1_10) && (est1_6 <= est1_11) && (est1_6 <= est1_12) &&
                                  (est1_6 <= est1_13) && (est1_6 <= est1_14) && (est1_6 <=
est1_15))
                        best1 <= 6;
                  else if ((est1_7 <= est1_8)  && (est1_7 <= est1_9)  && (est1_7 <= est1_10) &&
                           (est1_7 <= est1_11) && (est1_7 <= est1_12) && (est1_7 <= est1_13) &&
                                  (est1_7 <= est1_14) && (est1_7 <= est1_15))
                        best1 <= 7;
```

```
            else if ((est1_8 <= est1_9)  && (est1_8 <= est1_10) && (est1_8 <= est1_11) &&
                     (est1_8 <= est1_12) && (est1_8 <= est1_13) && (est1_8 <= est1_14) &&
                              (est1_8 <= est1_15))
                  best1 <= 8;
            else if ((est1_9 <= est1_10) && (est1_9 <= est1_11) && (est1_9 <= est1_12) &&
                     (est1_9 <= est1_13) && (est1_9 <= est1_14) && (est1_9 <= est1_15))
                  best1 <= 9;
            else if ((est1_10 <= est1_11) && (est1_10 <= est1_12) && (est1_10 <= est1_13)
&&
                     (est1_10 <= est1_14) && (est1_10 <= est1_15))
                  best1 <= 10;
            else if ((est1_11 <= est1_12) && (est1_11 <= est1_13) && (est1_11 <= est1_14)
&&
                     (est1_11 <= est1_15))
                  best1 <= 11;
            else if ((est1_12 <= est1_13) && (est1_12 <= est1_14) && (est1_12 <= est1_15))
                  best1 <= 12;
            else if ((est1_13 <= est1_14) && (est1_13 <= est1_15))
                  best1 <= 13;
            else if (est1_14 <= est1_15)
                  best1 <= 14;
            else
                  best1 <= 15;
            state <= state_N2_est;
            end
      state_N2_est:
            // calculate the value of N2_est.  This is the value
            // of N2 encoded for by best1 and N1.
            begin
            if (max1 > 2)
                  // here best1 must be shifted to obtain the correct
                  // difference magnitude
                  if (s1 == 0)
                        N2_est <= N1_in + (best1 << (max1 - 3));
                  else
                        N2_est <= N1_in - (best1 << (max1 - 3));
            else
                  // here best1 need not be shifted
                  if (s1 == 0)
                        N2_est <= N1_in + best1;
                  else
                        N2_est <= N1_in - best1;
            state <= state_recal_diff2;
            end
      state_recal_diff2:
            // recalculate diff2 taking into account any error
            // introduced by N2_est.
            begin
                  if (N3_in > N2_est)
                        diff2 <= N3_in - N2_est;
                  else
                        diff2 <= N2_est - N3_in;
            state <= state_est2;
            end
      state_est2:
```

```
// find the difference in value estimates

// for diff2

begin

if (val0 >= diff2)

      est2_0 <= val0 - diff2;

      else

      est2_0 <= diff2 - val0;

if (val1 >= diff2)

      est2_1 <= val1 - diff2;

      else

      est2_1 <= diff2 - val1;

if (val2 >= diff2)

      est2_2 <= val2 - diff2;

      else

      est2_2 <= diff2 - val2;

if (val3 >= diff2)

      est2_3 <= val3 - diff2;

      else

      est2_3 <= diff2 - val3;

if (val4 >= diff2)

      est2_4 <= val4 - diff2;

      else

      est2_4 <= diff2 - val4;

if (val5 >= diff2)

      est2_5 <= val5 - diff2;

      else

      est2_5 <= diff2 - val5;
```

```
if (val6 >= diff2)

      est2_6 <= val6 - diff2;

      else

      est2_6 <= diff2 - val6;

if (val7 >= diff2)

      est2_7 <= val7 - diff2;

      else

      est2_7 <= diff2 - val7;

if (val8 >= diff2)

      est2_8 <= val8 - diff2;

      else

      est2_8 <= diff2 - val8;

if (val9 >= diff2)

      est2_9 <= val9 - diff2;

      else

      est2_9 <= diff2 - val9;

if (val10 >= diff2)

      est2_10 <= val10 - diff2;

      else

      est2_10 <= diff2 - val10;

if (val11 >= diff2)

      est2_11 <= val11 - diff2;

      else

      est2_11 <= diff2 - val11;

if (val12 >= diff2)

      est2_12 <= val12 - diff2;

      else
      est2_12 <= diff2 - val12;
if (val13 >= diff2)
      est2_13 <= val13 - diff2;
      else
      est2_13 <= diff2 - val13;
if (val14 >= diff2)
      est2_14 <= val14 - diff2;
      else
      est2_14 <= diff2 - val14;
if (val15 >= diff2)
      est2_15 <= val15 - diff2;
      else
      est2_15 <= diff2 - val15;
state <= state_best2;
end
state_best2:
// find the best estimate of N3 - N2_est
// this corresponds to finding the smallest
// est2 value.
begin
if ((est2_0 <= est2_1)  && (est2_0 <= est2_2)  && (est2_0 <= est2_3)  &&
    (est2_0 <= est2_4)  && (est2_0 <= est2_5)  && (est2_0 <= est2_6)  &&
       (est2_0 <= est2_7)  && (est2_0 <= est2_8)  && (est2_0 <= est2_9)  &&
       (est2_0 <= est2_10) && (est2_0 <= est2_11) && (est2_0 <= est2_12) &&
       (est2_0 <= est2_13) && (est2_0 <= est2_14) && (est2_0 <= est2_15))
      best2 <= 0;
else if ((est2_1 <= est2_2)  && (est2_1 <= est2_3)  && (est2_1 <= est2_4)  &&
                         (est2_1 <= est2_5)  && (est2_1 <= est2_6)  && (est2_1 <= est2_7)  &&
                                (est2_1 <= est2_8)  && (est2_1 <= est2_9)  && (est2_1 <=
est2_10) &&
                                (est2_1 <= est2_11) && (est2_1 <= est2_12) && (est2_1 <=
est2_13) &&
                                (est2_1 <= est2_14) && (est2_1 <= est2_15))
                      best2 <= 1;
                 else if ((est2_2 <= est2_3)  && (est2_2 <= est2_4)  && (est2_2 <= est2_5)  &&
                                (est2_2 <= est2_6)  && (est2_2 <= est2_7)  && (est2_2 <=
est2_8)  &&
                                (est2_2 <= est2_9)  && (est2_2 <= est2_10) && (est2_2 <=
est2_11) &&
                                (est2_2 <= est2_12) && (est2_2 <= est2_13) && (est2_2 <=
est2_14) &&
                                (est2_2 <= est2_15))
                      best2 <= 2;
                 else if ((est2_3 <= est2_4)  && (est2_3 <= est2_5)  && (est2_3 <= est2_6)  &&
                         (est2_3 <= est2_7)  && (est2_3 <= est2_8)  && (est2_3 <= est2_9)  &&
                                (est2_3 <= est2_10) && (est2_3 <= est2_11) && (est2_3 <=
est2_12) &&
                                (est2_3 <= est2_13) && (est2_3 <= est2_14) && (est2_3 <=
est2_15))
                      best2 <= 3;
                 else if ((est2_4 <= est2_5)  && (est2_4 <= est2_6)  && (est2_4 <= est2_7)  &&
                         (est2_4 <= est2_8)  && (est2_4 <= est2_9)  && (est2_4 <= est2_10) &&
                                (est2_4 <= est2_11) && (est2_4 <= est2_12) && (est2_4 <=
est2_13) &&
                                (est2_4 <= est2_14) && (est2_4 <= est2_15))
                      best2 <= 4;
                 else if ((est2_5 <= est2_6)  && (est2_5 <= est2_7)  && (est2_5 <= est2_8)  &&
                           (est2_5 <= est2_9)  && (est2_5 <= est2_10) && (est2_5 <= est2_11)
&&
                                    (est2_5 <= est2_12) && (est2_5 <= est2_13) && (est2_5 <=
est2_14) &&

                                    (est2_5 <= est2_15))

                        best2 <= 5;

                  else if ((est2_6 <= est2_7)  && (est2_6 <= est2_8)  && (est2_6 <= est2_9)  &&

                           (est2_6 <= est2_10) && (est2_6 <= est2_11) && (est2_6 <= est2_12) &&

                                    (est2_6 <= est2_13) && (est2_6 <= est2_14) && (est2_6 <=
est2_15))

                        best2 <= 6;

                  else if ((est2_7 <= est2_8)  && (est2_7 <= est2_9)  && (est2_7 <= est2_10) &&

                           (est2_7 <= est2_11) && (est2_7 <= est2_12) && (est2_7 <= est2_13) &&

                                    (est2_7 <= est2_14) && (est2_7 <= est2_15))

                        best2 <= 7;

                  else if ((est2_8 <= est2_9)  && (est2_8 <= est2_10) && (est2_8 <= est2_11) &&

                           (est2_8 <= est2_12) && (est2_8 <= est2_13) && (est2_8 <= est2_14) &&

                                    (est2_8 <= est2_15))

                        best2 <= 8;

                  else if ((est2_9 <= est2_10) && (est2_9 <= est2_11) && (est2_9 <= est2_12) &&

                           (est2_9 <= est2_13) && (est2_9 <= est2_14) && (est2_9 <= est2_15))

                        best2 <= 9;

                  else if ((est2_10 <= est2_11) && (est2_10 <= est2_12) && (est2_10 <= est2_13)
&&

                           (est2_10 <= est2_14) && (est2_10 <= est2_15))

                        best2 <= 10;

                  else if ((est2_11 <= est2_12) && (est2_11 <= est2_13) && (est2_11 <= est2_14)
&&

                           (est2_11 <= est2_15))

                        best2 <= 11;

                  else if ((est2_12 <= est2_13) && (est2_12 <= est2_14) && (est2_12 <= est2_15))
```

```
            best2 <= 12;
      else if ((est2_13 <= est2_14) && (est2_13 <= est2_15))
            best2 <= 13;
      else if (est2_14 <= est2_15)
            best2 <= 14;
      else
            best2 <= 15;
      state <= state_N3_est;
      end
state_N3_est:
      // calculate the value of N3_est.  This is the value
      // of N3 encoded for by best2, best1 and N1.
      begin
      if (max1 > 2)
            // here best2 must be shifted to obtain the correct
            // difference magnitude
            if (s2 == 0)
                  N3_est <= N2_est + (best2 << (max1 - 3));
            else
                  N3_est <= N2_est - (best2 << (max1 - 3));
      else
            // here best2 need not be shifted
            if (s2 == 0)
                  N3_est <= N2_est + best2;
            else
                  N3_est <= N2_est - best2;
      state <= state_recal_diff3;
```

```
            end

      state_recal_diff3:

            // recalculate diff3 taking into account any error

            // introduced by N3_est.

            begin

                  if (N4_in > N3_est)

                        diff3 <= N4_in - N3_est;

                  else

                        diff3 <= N3_est - N4_in;

            state <= state_est3;

            end

      state_est3:

            // find the difference in value estimates

            // for diff3

            begin

            if (val0 >= diff3)

                  est3_0 <= val0 - diff3;

                  else

                  est3_0 <= diff3 - val0;

            if (val1 >= diff3)

                  est3_1 <= val1 - diff3;

                  else

                  est3_1 <= diff3 - val1;

            if (val2 >= diff3)

                  est3_2 <= val2 - diff3;

                  else

                  est3_2 <= diff3 - val2;
```

```
if (val3 >= diff3)

      est3_3 <= val3 - diff3;

      else

      est3_3 <= diff3 - val3;

if (val4 >= diff3)

      est3_4 <= val4 - diff3;

      else

      est3_4 <= diff3 - val4;

if (val5 >= diff3)

      est3_5 <= val5 - diff3;

      else

      est3_5 <= diff3 - val5;

if (val6 >= diff3)

      est3_6 <= val6 - diff3;

      else

      est3_6 <= diff3 - val6;

if (val7 >= diff3)

      est3_7 <= val7 - diff3;

      else

      est3_7 <= diff3 - val7;

if (val8 >= diff3)

      est3_8 <= val8 - diff3;

      else

      est3_8 <= diff3 - val8;

if (val9 >= diff3)

      est3_9 <= val9 - diff3;

      else
      est3_9 <= diff3 - val9;

if (val10 >= diff3)

      est3_10 <= val10 - diff3;

      else

      est3_10 <= diff3 - val10;

if (val11 >= diff3)

      est3_11 <= val11 - diff3;

      else

      est3_11 <= diff3 - val11;

if (val12 >= diff3)

      est3_12 <= val12 - diff3;

      else

      est3_12 <= diff3 - val12;

if (val13 >= diff3)

      est3_13 <= val13 - diff3;

      else

      est3_13 <= diff3 - val13;

if (val14 >= diff3)

      est3_14 <= val14 - diff3;

      else

      est3_14 <= diff3 - val14;

if (val15 >= diff3)

      est3_15 <= val15 - diff3;

      else

      est3_15 <= diff3 - val15;

state <= state_best3;

end
```

```
                    state_best3:
                    // find the best estimate of N4 - N3_est
                    // this corresponds to finding the smallest
                    // est3 value.
                    begin
                    if ((est3_0 <= est3_1)  && (est3_0 <= est3_2)  && (est3_0 <= est3_3)  &&
                        (est3_0 <= est3_4)  && (est3_0 <= est3_5)  && (est3_0 <= est3_6)  &&
                           (est3_0 <= est3_7)  && (est3_0 <= est3_8)  && (est3_0 <= est3_9)  &&
                           (est3_0 <= est3_10) && (est3_0 <= est3_11) && (est3_0 <= est3_12) &&
                           (est3_0 <= est3_13) && (est3_0 <= est3_14) && (est3_0 <= est3_15))
                          best3 <= 0;
                    else if ((est3_1 <= est3_2)  && (est3_1 <= est3_3)  && (est3_1 <= est3_4)  &&
                             (est3_1 <= est3_5)  && (est3_1 <= est3_6)  && (est3_1 <= est3_7)  &&
                                      (est3_1 <= est3_8)  && (est3_1 <= est3_9)  && (est3_1 <=
est3_10) &&
                                      (est3_1 <= est3_11) && (est3_1 <= est3_12) && (est3_1 <=
est3_13) &&
                                      (est3_1 <= est3_14) && (est3_1 <= est3_15))
                          best3 <= 1;
                    else if ((est3_2 <= est3_3)  && (est3_2 <= est3_4)  && (est3_2 <= est3_5)  &&
                                      (est3_2 <= est3_6)  && (est3_2 <= est3_7)  && (est3_2 <=
est3_8)  &&
                                      (est3_2 <= est3_9)  && (est3_2 <= est3_10) && (est3_2 <=
est3_11) &&
                                      (est3_2 <= est3_12) && (est3_2 <= est3_13) && (est3_2 <=
est3_14) &&
                                      (est3_2 <= est3_15))
                          best3 <= 2;
                    else if ((est3_3 <= est3_4)  && (est3_3 <= est3_5)  && (est3_3 <= est3_6)  &&
                    (est3_3 <= est3_7)  && (est3_3 <= est3_8)  && (est3_3 <= est3_9)  &&
                              (est3_3 <= est3_10) && (est3_3 <= est3_11) && (est3_3 <=
est3_12) &&
                              (est3_3 <= est3_13) && (est3_3 <= est3_14) && (est3_3 <=
est3_15))
                best3 <= 3;
            else if ((est3_4 <= est3_5)  && (est3_4 <= est3_6)  && (est3_4 <= est3_7)  &&
                    (est3_4 <= est3_8)  && (est3_4 <= est3_9)  && (est3_4 <= est3_10) &&
                              (est3_4 <= est3_11) && (est3_4 <= est3_12) && (est3_4 <=
est3_13) &&
                              (est3_4 <= est3_14) && (est3_4 <= est3_15))
                best3 <= 4;
            else if ((est3_5 <= est3_6)  && (est3_5 <= est3_7)  && (est3_5 <= est3_8)  &&
                      (est3_5 <= est3_9)  && (est3_5 <= est3_10) && (est3_5 <= est3_11)
&&
                              (est3_5 <= est3_12) && (est3_5 <= est3_13) && (est3_5 <=
est3_14) &&
                              (est3_5 <= est3_15))
                best3 <= 5;
            else if ((est3_6 <= est3_7)  && (est3_6 <= est3_8)  && (est3_6 <= est3_9)  &&
                    (est3_6 <= est3_10) && (est3_6 <= est3_11) && (est3_6 <= est3_12) &&
                              (est3_6 <= est3_13) && (est3_6 <= est3_14) && (est3_6 <=
est3_15))
                best3 <= 6;
            else if ((est3_7 <= est3_8)  && (est3_7 <= est3_9)  && (est3_7 <= est3_10) &&
                    (est3_7 <= est3_11) && (est3_7 <= est3_12) && (est3_7 <= est3_13) &&
                              (est3_7 <= est3_14) && (est3_7 <= est3_15))
                best3 <= 7;
            else if ((est3_8 <= est3_9)  && (est3_8 <= est3_10) && (est3_8 <= est3_11) &&
                    (est3_8 <= est3_12) && (est3_8 <= est3_13) && (est3_8 <= est3_14) &&
                        (est3_8 <= est3_15))
                best3 <= 8;
            else if ((est3_9 <= est3_10) && (est3_9 <= est3_11) && (est3_9 <= est3_12) &&
                    (est3_9 <= est3_13) && (est3_9 <= est3_14) && (est3_9 <= est3_15))
                best3 <= 9;
            else if ((est3_10 <= est3_11) && (est3_10 <= est3_12) && (est3_10 <= est3_13)
&&
                    (est3_10 <= est3_14) && (est3_10 <= est3_15))
                best3 <= 10;
            else if ((est3_11 <= est3_12) && (est3_11 <= est3_13) && (est3_11 <= est3_14)
&&
                    (est3_11 <= est3_15))
                best3 <= 11;
            else if ((est3_12 <= est3_13) && (est3_12 <= est3_14) && (est3_12 <= est3_15))
                best3 <= 12;
            else if ((est3_13 <= est3_14) && (est3_13 <= est3_15))
                best3 <= 13;
            else if (est3_14 <= est3_15)
                best3 <= 14;
            else
                best3 <= 15;
            state <= state_N4_est;
            end
        state_N4_est:
            // calculate the value of N4_est.  This is the value
            // of N4 encoded for by best3, best2, best1 and N1.
            begin
            if (max1 > 2)
                // here best3 must be shifted to obtain the correct
                // difference magnitude
                if (s3 == 0)
                      N4_est <= N3_est + (best3 << (max1 - 3));
                else
                      N4_est <= N3_est - (best3 << (max1 - 3));
            else
                // here best1 need not be shifted
                if (s3 == 0)
                      N4_est <= N3_est + best3;
                else
                      N4_est <= N3_est - best3;
            state <= state_recal_diff4;
            end
        state_recal_diff4:
            // recalculate diff4 taking into account any error
            // introduced by N4_est.
            begin
                if (N5_in > N4_est)
                      diff4 <= N5_in - N4_est;
                else
                      diff4 <= N4_est - N5_in;
            state <= state_est4;
            end
        state_est4:
            // find the difference in value estimates
            // for diff4
```

```
begin

if (val0 >= diff4)

      est4_0 <= val0 - diff4;

      else

      est4_0 <= diff4 - val0;

if (val1 >= diff4)

      est4_1 <= val1 - diff4;

      else

      est4_1 <= diff4 - val1;

if (val2 >= diff4)

      est4_2 <= val2 - diff4;

      else

      est4_2 <= diff4 - val2;

if (val3 >= diff4)

      est4_3 <= val3 - diff4;

      else

      est4_3 <= diff4 - val3;

if (val4 >= diff4)

      est4_4 <= val4 - diff4;

      else

      est4_4 <= diff4 - val4;

if (val5 >= diff4)

      est4_5 <= val5 - diff4;

      else

      est4_5 <= diff4 - val5;

if (val6 >= diff4)

      est4_6 <= val6 - diff4;
      else
      est4_6 <= diff4 - val6;
if (val7 >= diff4)
      est4_7 <= val7 - diff4;
      else
      est4_7 <= diff4 - val7;
if (val8 >= diff4)
      est4_8 <= val8 - diff4;
      else
      est4_8 <= diff4 - val8;
if (val9 >= diff4)
      est4_9 <= val9 - diff4;
      else
      est4_9 <= diff4 - val9;
if (val10 >= diff4)
      est4_10 <= val10 - diff4;
      else
      est4_10 <= diff4 - val10;
if (val11 >= diff4)
      est4_11 <= val11 - diff4;
      else
      est4_11 <= diff4 - val11;
if (val12 >= diff4)
      est4_12 <= val12 - diff4;
      else
      est4_12 <= diff4 - val12;
if (val13 >= diff4)
```

```
      est4_13 <= val13 - diff4;

      else

      est4_13 <= diff4 - val13;

if (val14 >= diff4)

      est4_14 <= val14 - diff4;

      else

      est4_14 <= diff4 - val14;

if (val15 >= diff4)

      est4_15 <= val15 - diff4;

      else

      est4_15 <= diff4 - val15;

state <= state_best4;

end

state_best4:

// find the best estimate of N5 - N4_est

// this corresponds to finding the smallest

// est4 value.

begin

if ((est4_0 <= est4_1)  && (est4_0 <= est4_2)  && (est4_0 <= est4_3)  &&

    (est4_0 <= est4_4)  && (est4_0 <= est4_5)  && (est4_0 <= est4_6)  &&

       (est4_0 <= est4_7)  && (est4_0 <= est4_8)  && (est4_0 <= est4_9)  &&

       (est4_0 <= est4_10) && (est4_0 <= est4_11) && (est4_0 <= est4_12) &&

       (est4_0 <= est4_13) && (est4_0 <= est4_14) && (est4_0 <= est4_15))

      best4 <= 0;

else if ((est4_1 <= est4_2)  && (est4_1 <= est4_3)  && (est4_1 <= est4_4)  &&

         (est4_1 <= est4_5)  && (est4_1 <= est4_6)  && (est4_1 <= est4_7)  &&

                  (est4_1 <= est4_8)  && (est4_1 <= est4_9)  && (est4_1 <=
est4_10) &&
                                (est4_1 <= est4_11) && (est4_1 <= est4_12) && (est4_1 <=
est4_13) &&
                                (est4_1 <= est4_14) && (est4_1 <= est4_15))
                      best4 <= 1;
                else if ((est4_2 <= est4_3)  && (est4_2 <= est4_4)  && (est4_2 <= est4_5)  &&
                                (est4_2 <= est4_6)  && (est4_2 <= est4_7)  && (est4_2 <=
est4_8)  &&
                                (est4_2 <= est4_9)  && (est4_2 <= est4_10) && (est4_2 <=
est4_11) &&
                                (est4_2 <= est4_12) && (est4_2 <= est4_13) && (est4_2 <=
est4_14) &&
                                (est4_2 <= est4_15))
                      best4 <= 2;
                else if ((est4_3 <= est4_4)  && (est4_3 <= est4_5)  && (est4_3 <= est4_6)  &&
                         (est4_3 <= est4_7)  && (est4_3 <= est4_8)  && (est4_3 <= est4_9)  &&
                                (est4_3 <= est4_10) && (est4_3 <= est4_11) && (est4_3 <=
est4_12) &&
                                (est4_3 <= est4_13) && (est4_3 <= est4_14) && (est4_3 <=
est4_15))
                      best4 <= 3;
                else if ((est4_4 <= est4_5)  && (est4_4 <= est4_6)  && (est4_4 <= est4_7)  &&
                         (est4_4 <= est4_8)  && (est4_4 <= est4_9)  && (est4_4 <= est4_10) &&
                                (est4_4 <= est4_11) && (est4_4 <= est4_12) && (est4_4 <=
est4_13) &&
                                (est4_4 <= est4_14) && (est4_4 <= est4_15))
                      best4 <= 4;
                else if ((est4_5 <= est4_6)  && (est4_5 <= est4_7)  && (est4_5 <= est4_8)  &&
                            (est4_5 <= est4_9)  && (est4_5 <= est4_10) && (est4_5 <= est4_11)
&&
                                (est4_5 <= est4_12) && (est4_5 <= est4_13) && (est4_5 <=
est4_14) &&
                                (est4_5 <= est4_15))
                      best4 <= 5;
                else if ((est4_6 <= est4_7)  && (est4_6 <= est4_8)  && (est4_6 <= est4_9)  &&
                         (est4_6 <= est4_10) && (est4_6 <= est4_11) && (est4_6 <= est4_12) &&
                                (est4_6 <= est4_13) && (est4_6 <= est4_14) && (est4_6 <=
est4_15))
                      best4 <= 6;
                else if ((est4_7 <= est4_8)  && (est4_7 <= est4_9)  && (est4_7 <= est4_10) &&
                         (est4_7 <= est4_11) && (est4_7 <= est4_12) && (est4_7 <= est4_13) &&
                                (est4_7 <= est4_14) && (est4_7 <= est4_15))
                      best4 <= 7;
                else if ((est4_8 <= est4_9)  && (est4_8 <= est4_10) && (est4_8 <= est4_11) &&
                         (est4_8 <= est4_12) && (est4_8 <= est4_13) && (est4_8 <= est4_14) &&
                                (est4_8 <= est4_15))
                      best4 <= 8;
                else if ((est4_9 <= est4_10) && (est4_9 <= est4_11) && (est4_9 <= est4_12) &&
                         (est4_9 <= est4_13) && (est4_9 <= est4_14) && (est4_9 <= est4_15))
                      best4 <= 9;
                else if ((est4_10 <= est4_11) && (est4_10 <= est4_12) && (est4_10 <= est4_13)
&&
                         (est4_10 <= est4_14) && (est4_10 <= est4_15))
                      best4 <= 10;
                else if ((est4_11 <= est4_12) && (est4_11 <= est4_13) && (est4_11 <= est4_14)
&&
                         (est4_11 <= est4_15))
                      best4 <= 11;
                else if ((est4_12 <= est4_13) && (est4_12 <= est4_14) && (est4_12 <= est4_15))
                      best4 <= 12;
```

```
else if ((est4_13 <= est4_14) && (est4_13 <= est4_15))
      best4 <= 13;
else if (est4_14 <= est4_15)
      best4 <= 14;
else
      best4 <= 15;
state <= state_done;
end
state_done:
      // set the output, cdiff_out
      begin
      cdiff_out <= {best4, s4, best3, s3, best2, s2, best1, s1, max1, N1_in};
      // if the next module is currently reading something, don't set
      // valid_out high yet.  Wait until they are done.
      if (valid_read)
            state <= state_done;
      else
            state <= state_done2;
      end
/////////////////////////////////////////
/// IF YOU DON'T WANT TO WAIT FOR THE
/// VALID_READ SIGNAL TO BECOME HIGH,
/// CHANGE THE STATE TRANSITION HERE
/// TO BE:
/// state <= state_wait;
/////////////////////////////////////////
state_done2:
```

```
                        // set the valid_out signal high
                        begin
                        valid_out <= 1;
                        state <= state_done3;  // state <= state_wait;
                        end
                  state_done3:
                        // wait for valid_read to become high before
                        // trying to collect more input data
                              if (valid_read)
                                    begin
                                    valid_out <= 0;
                                    valid_compressing <= 0;
                                    state <= state_wait;
                                    end
                              else
                                    state <= state_done3;
                  endcase
endmodule
```

```
`timescale 1ns / 1ps
// Module: ec_in
// Inputs:
      // clock: internal clock of the system
      // reset: zeros the outputs, places fsm in the wait state
      // cdiff_out: 40 bit packet provided by the compression module
      // valid_read: input from the module following this.  Will go high when the
      //             next module has read ec_out.
      // valid_in: when high, signifies that the data present on cdiff_out is valid

// Outputs;

      // valid_out: when high, signifies that the data presend on ec_out is valid

      // valid_ecing: when high, signifies that the module is busy error correcting data

      // ec_out: 50 bit data packet produced by the error correction module, adds

      //         10 bits of data to the 40 bit packet provided by the compression

      //         module

module error_correction_input(clock, reset, cdiff_out, valid_read, valid_in, valid_out,
valid_ecing, ec_out);

    input  clock;

    input  reset;

    input  [39:0] cdiff_out;

    input  valid_in;

       input  valid_read;

       output valid_out;

    output valid_ecing;

    output [49:0] ec_out;

      // N1_in: registered value of N1, the first 16 bits of any cdiff_out packet

      // shift_val: registered value of the shift_val created by the compression module

      //            it is the 4 bits following N1 of any cdiff_out packet

      reg [15:0] N1_in;

      reg [ 3:0] shift_val;

      // first1: the placing of the first 1 in the number N1_in

      // second1: the placing of the second 1 in the number N1_in

      // shift1: the placing of the first 1 in the number shift_val

      reg [3:0] first1;

      reg [3:0] second1;
      reg [1:0] shift1;

      // state: internal variable to keep track of the state of the fsm

      reg [3:0] state;

      // module outputs

      reg valid_out;

      reg valid_ecing;

      reg [49:0] ec_out;

      reg [39:0] cdiff_in;

      // parameterization of fsm states

      parameter state_wait    = 0;

      parameter state_first1  = 1;

      parameter state_second1 = 2;

      parameter state_shift1  = 3;

      parameter state_done    = 4;

      parameter state_done2   = 5;

      parameter state_done3   = 6;

      always @ (posedge clock)

            if (reset)

            // zero all outputs during a reset

            // after a reset, the fsm will always return

            // to state_wait.

                  begin

                  ec_out      <= 0;

                  valid_out   <= 0;

                  valid_ecing <= 0;

                  state       <= state_wait;

                  end
```

```
else

case(state)

      state_wait:

            // valid_ecing and valid_out are set to zero

            // simply waiting for valid_in to become high

            // so error compression can begin

            //

            // when valid_in becomes high all needed values

            // are registered, including the packet itself

            // (cdiff_in), N1_in, and shift_val

            begin

                  valid_ecing <= 0;

                  valid_out   <= 0;

            if (valid_in)

                  begin

                  cdiff_in  <= cdiff_out;

                  N1_in     <= cdiff_out[15:0];

                  shift_val <= cdiff_out[19:16];

                  state     <= state_first1;

                  end

            else

                  state <= state_wait;

            end

      state_first1:

      // finds the position of the first 1 in N1_in

      // registers this as the value first1

      begin
```

```
valid_ecing <= 1;

if (N1_in[15])

      first1 <= 15;

else if (N1_in[14])

      first1 <= 14;

else if (N1_in[13])

      first1 <= 13;

else if (N1_in[12])

      first1 <= 12;

else if (N1_in[11])

      first1 <= 11;

else if (N1_in[10])

      first1 <= 10;

else if (N1_in[9])

      first1 <= 9;

else if (N1_in[8])

      first1 <= 8;

else if (N1_in[7])

      first1 <= 7;

else if (N1_in[6])

      first1 <= 6;

else if (N1_in[5])

      first1 <= 5;

else if (N1_in[4])

      first1 <= 4;

else if (N1_in[3])

      first1 <= 3;
      else if (N1_in[2])
            first1 <= 2;
      else if (N1_in[1])
            first1 <= 1;
      else
            first1 <= 0;
      state <= state_second1;
      end
state_second1:
      // finds the position of the second1 in N1
      // this is based on the value of first1, namely
      // the position of the second one must come
      // after the position of the first one.
      //
      // Hence, if first1 = 5, second1 must equal
      // either 4,3,2,1 or 0.
      begin
      if (first1 == 0)
            second1 <= 0;
      else if (first1 == 1)
            second1 <= 0;
      else if (first1 == 2)
            if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
      else if (first1 == 3)
            if (N1_in[2])
                  second1 <= 2;
            else if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
      else if (first1 == 4)
            if (N1_in[3])
                  second1 <= 3;
            else if (N1_in[2])
                  second1 <= 2;
            else if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
      else if (first1 == 5)
            if (N1_in[4])
                  second1 <= 4;
            else if (N1_in[3])
                  second1 <= 3;
            else if (N1_in[2])
                  second1 <= 2;
            else if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
      else if (first1 == 6)
```

```
            if (N1_in[5])
                  second1 <= 5;
            else if (N1_in[4])
                  second1 <= 4;
            else if (N1_in[3])
                  second1 <= 3;
            else if (N1_in[2])
                  second1 <= 2;
            else if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
      else if (first1 == 7)
            if (N1_in[6])
                  second1 <= 6;
            else if (N1_in[5])
                  second1 <= 5;
            else if (N1_in[4])
                  second1 <= 4;
            else if (N1_in[3])
                  second1 <= 3;
            else if (N1_in[2])
                  second1 <= 2;
            else if (N1_in[1])
                  second1 <= 1;
            else
                  second1 <= 0;
```

```
else if (first1 == 8)
      if (N1_in[7])
            second1 <= 7;
      else if (N1_in[6])
            second1 <= 6;
      else if (N1_in[5])
            second1 <= 5;
      else if (N1_in[4])
            second1 <= 4;
      else if (N1_in[3])
            second1 <= 3;
      else if (N1_in[2])
            second1 <= 2;
      else if (N1_in[1])
            second1 <= 1;
      else
            second1 <= 0;
else if (first1 == 9)
      if (N1_in[8])
            second1 <= 8;
      else if (N1_in[7])
            second1 <= 7;
      else if (N1_in[6])
            second1 <= 6;
      else if (N1_in[5])
            second1 <= 5;
      else if (N1_in[4])
```

```
            second1 <= 4;
        else if (N1_in[3])
            second1 <= 3;
        else if (N1_in[2])
            second1 <= 2;
        else if (N1_in[1])
            second1 <= 1;
        else
            second1 <= 0;
    else if (first1 == 10)
        if (N1_in[9])
            second1 <= 9;
        else if (N1_in[8])
            second1 <= 8;
        else if (N1_in[7])
            second1 <= 7;
        else if (N1_in[6])
            second1 <= 6;
        else if (N1_in[5])
            second1 <= 5;
        else if (N1_in[4])
            second1 <= 4;
        else if (N1_in[3])
            second1 <= 3;
        else if (N1_in[2])
            second1 <= 2;
        else if (N1_in[1])
```

```
            second1 <= 1;
      else
            second1 <= 0;
else if (first1 == 11)
      if (N1_in[10])
            second1 <= 10;
      else if (N1_in[9])
            second1 <= 9;
      else if (N1_in[8])
            second1 <= 8;
      else if (N1_in[7])
            second1 <= 7;
      else if (N1_in[6])
            second1 <= 6;
      else if (N1_in[5])
            second1 <= 5;
      else if (N1_in[4])
            second1 <= 4;
      else if (N1_in[3])
            second1 <= 3;
      else if (N1_in[2])
            second1 <= 2;
      else if (N1_in[1])
            second1 <= 1;
      else
            second1 <= 0;
else if (first1 == 12)
```

```
        if (N1_in[11])
              second1 <= 11;
        else if (N1_in[10])
              second1 <= 10;
        else if (N1_in[9])
              second1 <= 9;
        else if (N1_in[8])
              second1 <= 8;
        else if (N1_in[7])
              second1 <= 7;
        else if (N1_in[6])
              second1 <= 6;
        else if (N1_in[5])
              second1 <= 5;
        else if (N1_in[4])
              second1 <= 4;
        else if (N1_in[3])
              second1 <= 3;
        else if (N1_in[2])
              second1 <= 2;
        else if (N1_in[1])
              second1 <= 1;
        else
              second1 <= 0;
  else if (first1 == 13)
        if (N1_in[12])
              second1 <= 12;
        else if (N1_in[11])
              second1 <= 11;
        else if (N1_in[10])
              second1 <= 10;
        else if (N1_in[9])
              second1 <= 9;
        else if (N1_in[8])
              second1 <= 8;
        else if (N1_in[7])
              second1 <= 7;
        else if (N1_in[6])
              second1 <= 6;
        else if (N1_in[5])
              second1 <= 5;
        else if (N1_in[4])
              second1 <= 4;
        else if (N1_in[3])
              second1 <= 3;
        else if (N1_in[2])
              second1 <= 2;
        else if (N1_in[1])
              second1 <= 1;
        else
              second1 <= 0;
  else if (first1 == 14)
        if (N1_in[13])
              second1 <= 13;
        else if (N1_in[12])
              second1 <= 12;
        else if (N1_in[11])
              second1 <= 11;
        else if (N1_in[10])
              second1 <= 10;
        else if (N1_in[9])
              second1 <= 9;
        else if (N1_in[8])
              second1 <= 8;
        else if (N1_in[7])
              second1 <= 7;
        else if (N1_in[6])
              second1 <= 6;
        else if (N1_in[5])
              second1 <= 5;
        else if (N1_in[4])
              second1 <= 4;
        else if (N1_in[3])
              second1 <= 3;
        else if (N1_in[2])
              second1 <= 2;
        else if (N1_in[1])
              second1 <= 1;
        else
              second1 <= 0;
  else if (first1 == 15)
```

```
if (N1_in[14])
      second1 <= 14;
else if (N1_in[13])
      second1 <= 13;
else if (N1_in[12])
      second1 <= 12;
else if (N1_in[11])
      second1 <= 11;
else if (N1_in[10])
      second1 <= 10;
else if (N1_in[9])
      second1 <= 9;
else if (N1_in[8])
      second1 <= 8;
else if (N1_in[7])
      second1 <= 7;
else if (N1_in[6])
      second1 <= 6;
else if (N1_in[5])
      second1 <= 5;
else if (N1_in[4])
      second1 <= 4;
else if (N1_in[3])
      second1 <= 3;
else if (N1_in[2])
      second1 <= 2;
else if (N1_in[1])
```

```
                        second1 <= 1;
                    else
                        second1 <= 0;
                state <= state_shift1;
                end
        state_shift1:
            // finds the position of the first 1 in
            // shift val.  Registers this number
            // as the value shift1
            begin
            if (shift_val[3])
                shift1 <= 3;
            else if (shift_val[2])
                shift1 <= 2;
            else if (shift_val[1])
                shift1 <= 1;
            else
                shift1 <= 0;
            state <= state_done;
            end
        state_done:
            // sets the value of ec_out, waits for valid_read
            // to become low before transitioning on.  This
            // makes sure that the module following this is
            // not busy when valid_out is set high
            begin
            ec_out <= {cdiff_in, shift1, second1, first1};
            if (valid_read)
                  state <= state_done;
            else
                  state <= state_done2;
            end
/////////////////////////////////////////
/// IF YOU DON'T WANT TO WAIT FOR THE ///
/// VALID_READ SIGNAL TO BECOME HIGH, ///
/// CHANGE THE STATE TRANSITION HERE  ///
/// TO BE:                            ///
/// state <= state_wait;              ///
/////////////////////////////////////////
      state_done2:
            // sets valid_out high, transitions to the
            // next state
            begin
            valid_out <= 1;
            state     <= state_done3; // state_wait;
            end
      state_done3:
            // waits for the data to be read by the following
            // module.  Once the data has been read (ie. valid
            // read goes high) the fsm can transition back to
            // state_wait and wait for new data to arrives
            if (valid_read)
                  begin
                  valid_out   <= 0;
```

```
                        valid_ecing <= 0;

                        state <= state_wait;

                        end

                  else

                        state <= state_done3;

            endcase

endmodule
```

```
timescale 1ns / 1ps

// Module: error_correction_output

// Inputs:

      // clock: internal clock of the system

      // reset: zeros the outputs, places fsm in the wait state

      // ec_in: 50 bit packet provided by the buffer_nivedita module

      // valid_in: when high, signifies that the data present on ec_in is valid

      // valid_read: input from the module following this.  Will go high when the

      //             next module has read ec_out.


// Outputs;

      // valid_ecing: when high, signifies that the module is busy error correcting data

      // valid_out: when high, signifies that the data presend on ec_out is valid

      // ec_out: 40 bit data packet produced by the error correction module, takes off

      //         10 bits of data from the 50 bit packet provided by the compression

      //         module

      // request_data_out: when high, signifies that the module is requesting new data

      //                   from the buffer.  Happens before the fsm transitions back

      //                   into the wait state.

module error_correction_output(clock, reset, ec_in, valid_in, valid_read, valid_ecing,
valid_out, ec_out, request_data_out);
```

```
input  clock;

input  reset;

input  [49:0] ec_in;

input  valid_in;

input  valid_read;

   output valid_ecing;

output valid_out;

output [39:0] ec_out;

   output request_data_out;

  // Outputs

  reg valid_ecing;

  reg valid_out;

  reg [39:0] ec_out;

  reg request_data_out;

  // Registered inputs from ec_in

  //

  // first1: location of the first 1 in N1

  // second1: location of the second 1 in N1

  // shift1: location of the first 1 in shift_val

  reg [3:0] first1;

  reg [3:0] second1;

  reg [1:0] shift1;

  // N1: value of N1 taken from the ec_in data

  // shift_val: value of the shift_value taken from the ec_in data

  // rest_data: the rest of the ec_in data not including N1 or shift_val

  reg [15:0] N1;

  reg [3:0]  shift_val;
  reg [19:0] rest_data;

  // state: register to store the state of the fsm

  reg [4:0] state;

  // parameterization of fsm states

  parameter state_wait    = 0;

  parameter state_shiftl1 = 1;

  parameter state_set1    = 2;

  parameter state_shiftr1 = 3;

  parameter state_shiftl2 = 4;

  parameter state_set2    = 5;

  parameter state_shiftr2 = 6;

  parameter state_done    = 7;

  parameter state_done2   = 8;

  parameter state_done3   = 9;

  always @ (posedge clock)

        // on reset, zero all outputs and return the fsm to

        // state_wait

        if (reset)

              begin

              ec_out      <= 0;

              valid_out   <= 0;

              valid_ecing <= 0;

              state       <= state_wait;

              request_data_out <= 0;

              end

        else

        case(state)
```

```
state_wait:
      // send valid_out and valid_ecing to 0 as the output
      // is not valid and the module is not currently error
      // correcting anything.  Also request_data_out is set
      // high so that the buffer knows the module is
      // currently waiting for new data
      //
      // when valid_in becomes high register all dat values needed
      // from ec_in
      begin
      valid_out   <= 0;
      valid_ecing <= 0;
      request_data_out <= 1;
      if (valid_in)
            begin
            first1    <= ec_in[3:0];
            second1   <= ec_in[7:4];
            shift1    <= ec_in[9:8];
            N1        <= ec_in[25:10];
            shift_val <= ec_in[29:26];
            rest_data <= ec_in[49:30];
            state <= state_shiftl1;
            end
      else
            state <= state_wait;
      end
state_shiftl1:
```

```
// request_data_out can now be set low as we no longer need
// data.  Valid_ecing is set high as we are now error
// correcting a packet.
//
// here left shifts of the data values occur
begin
request_data_out <= 0;
valid_ecing <= 1;
if (second1 == 0)
      begin
      end
      // if the second1 is zero,
      // shift nothing
else
      // otherwise shift the data (note that this may
      // shift out the position of the first1.  This is
      // okay, however, as the first1 will be set later on
      N1 <= N1 << (15 - second1);
if (shift1 == 0)
      begin
      end
      // if the shift1 is zero,
      // shift nothing
else
      // similarly for shift_val, we shift out all
      // data that should be zero
      shift_val <= shift_val << (3 - shift1);
      state <= state_set1;
      end
state_set1:
      // we have shifted out all of the top zeros so
      // far.  Thus we know that the highest order
      // digit should be a one.  To ensure this, it
      // is set high, unless second1 or shift1 is zero.
      begin
      if (second1 == 0)
            begin
            end
      else
            N1[15] <= 1;
      if (shift1 == 0)
            begin
            end
      else
            shift_val[3] <= 1;
      state <= state_shiftr1;
      end
state_shiftr1:
      // Now that the position of two of the known
      // ones have been set we can shift the data
      // values back.
      begin
      if (second1 == 0)
            begin
            end

      else

            N1 <= N1 >> (15 - second1);

      if (shift1 == 0)

            begin

            end

      else

            shift_val <= shift_val >> (3 - shift1);

      state <= state_shiftl2;

      end

state_shiftl2:

      // Only one shift remains, the shift corresponding

      // to first1.

      // Start by shifting out the zeros before the first1.

      begin

      if (first1 == 0)

            begin

            end

      else

            N1 <= N1 << (15 - first1);

      state <= state_set2;

      end

state_set2:

      // Then set the value of the top bit as 1.

      begin

      if (first1 == 0)

            begin
            end
      else
            N1[15] <= 1;
      state <= state_shiftr2;
      end
state_shiftr2:
      // Then shift back to the right by the
      // same amount.
      begin
      if (first1 == 0)
            begin
            end
      else
      N1 <= N1 >> (15 - first1);
      state <= state_done;
      end
state_done:
      // sets the value of ec_out, waits for valid_read
      // to become low before transitioning on.  This
      // makes sure that the module following this is
      // not busy when valid_out is set high
      begin
      ec_out <= {rest_data, shift_val, N1};
      if (valid_read)
            state  <= state_done;
      else
            state <= state_done2;
            end

      state_done2:

            // sets valid_out high, transitions to the

            // next state

            begin

            valid_out <= 1;

            state     <= state_done3;

            request_data_out <= 1;

            end

      state_done3:

            // waits for the data to be read by the following

            // module.  Once the data has been read (ie. valid

            // read goes high) the fsm can transition back to

            // state_wait and wait for new data to arrives

            if (valid_read)

                  begin

                  valid_out   <= 0;

                  valid_ecing <= 0;

                  state       <= state_wait;

                  end

            else

                  state <= state_done3;

      endcase

endmodule
```

```
// MODULE: buffer_aditi

      // To create an 800 bit packet of data compatable with Nivedita's wireless

      // system.
// clock: internal clock of the entire system

// reset: zeros the outputs, puts the fsm in the wait state

// valid_in: signals that the inputted data (called 'data_right here) is valid

// done_transmit: when high, signals that the transmitter has taken in the outputted data

//                and will soon be ready for new data

// data_right: 50 bit data chunk (compressed & formatted for error correction) representing

//       5 signed 16 bit numbers from the right-sided data

// data_out: 800 bit data pack comprised of 8 left/right data pairs outputted to the

//           transmitter

// valid_out: when high, signifies that the data present on data_out is a valid packet that

//            is to be transmitted

// valid_read: outputted by the FIFO buffer signifying that the buffer has taken in data_right

//            (active high for one clock cycle)

// buffer_full: indicates that the FIFO buffer has been filled

module buffer_aditi(clock, reset, valid_in, done_transmit,

                    data_right, data_out, valid_out, valid_read, buffer_full);

input  clock;

input  reset;

input  valid_in;

input  done_transmit;

input  [49:0] data_right;

output [799:0] data_out;

output valid_out;

output valid_read;

output buffer_full;

// INTERNAL

//
```

```
// in_pointer: this FIFO can store up to 10 800 bit packets at a
//             time (hence there are slots 1 to 10).  The value
//             of the in_pointer dictates which slot new data
//             is to be written to.
// out_pointer: dictates wich slot data is to be taken from when
//              outputting new data
// count_data: represents how many data packets are currently being
//             stored in the system.  Does not include packets that
//             have already been outputted as these can be over-
//             written without consequence.
// state: 5 bit value representing the current state of the system
// state_saved: the FSM implimented here uses a sort of 'jump back'
//              system whereby it will enter a state and then jump
//              back to a previous state, the value of which can
//              vary.  The state_saved is the previous state that it
//              is to jump back to.
reg [4:0] in_pointer;
reg [4:0] out_pointer;
reg [9:0] count_data;
reg [5:0] state;
reg [4:0] state_saved;
// buffer_full: when high, signifies that all ten slots have been
//              written to with data that has not yet been outputted
//              Given to an LED, it will let the user know if there
//              is potential for error
//              Note that this can be set low again if the buffer
//              is emptied
// valid_out: the data on the output terminals is valid when this

//            signal is high

// prev_done_transmit: the value of the done_transmit signal on the

//                     previous clock cycle.  Useful in

//                     looking for the posedge of done_transmit

// valid_read: signifies that both data_right and data_left have been

//             registered. High for one clock cycle when this occurs.

reg buffer_full;

reg valid_out;

reg prev_done_transmit;

reg valid_read;

// Here the 10 data slots for the 800 bit packets are declared.

// Note that data_in is not a slot, but is rather a place to

// store data as it comes in in the 50 bit packets.

reg [799:0] data_out;

reg [799:0] data_in;

reg [799:0] data_in1;

reg [799:0] data_in2;

reg [799:0] data_in3;

reg [799:0] data_in4;

reg [799:0] data_in5;

reg [799:0] data_in6;

reg [799:0] data_in7;

reg [799:0] data_in8;

reg [799:0] data_in9;

reg [799:0] data_in10;

// Here the various states of the fsm are parameterized.
```

```
parameter state_wait   = 0;

parameter state_50     = 1;

parameter state_100    = 2;

parameter state_150    = 3;

parameter state_200    = 4;

parameter state_250    = 5;

parameter state_300    = 6;

parameter state_350    = 7;

parameter state_400    = 8;

parameter state_450    = 9;

parameter state_500    = 10;

parameter state_550    = 11;

parameter state_600    = 12;

parameter state_650    = 13;

parameter state_700    = 14;

parameter state_750    = 15;

parameter state_800    = 16;

parameter state_ackin  = 17;

parameter state_ackin2 = 18;

parameter state_store  = 19;

always @ (posedge clock)

      if (reset)

            // on a reset all outputs are zeroed.  The in and

            // outpointers are both set to one (meaning that

            // incoming data will be written to slot one and

            // no data will be outputted until at least one

            // data packet has been stored)
            // The FSM is placed in the wait state.

            // Additionally, count_data is set to zero meaning

            // that there is no valid data stored in the FIFO.

            // ... which is why all of the data slots are

            // zeroed.  (data_in1, data_in2, etc.)

            begin

            valid_out    <= 0;

            data_out     <= 0;

            in_pointer   <= 1;

            out_pointer  <= 1;

            count_data   <= 0;

            valid_read   <= 0;

            buffer_full  <= 0;

            state        <= state_wait;

            data_in1  <= 0;

            data_in2  <= 0;

            data_in3  <= 0;

            data_in4  <= 0;

            data_in5  <= 0;

            data_in6  <= 0;

            data_in7  <= 0;

            data_in8  <= 0;

            data_in9  <= 0;

            data_in10 <= 0;

            // initialize the prev_done_transmit reg as well.

            prev_done_transmit <= 0;

            end
```

```
else

      begin

      // Begin by registering the current done_transmit signal. (This change will

      // not take effect until the next clock cycle)

      prev_done_transmit <= done_transmit;

      // Now, check if there has been a 'posedge' of done_transmit or if valid_out

      // has been set to zero.  Either condition implies that there is an

      // impending request for new data to be outputted.

      //

      // Note that valid_out is only ever zero after a reset or if there was a

      // posedge of done_transmit and no data to output.

      if  (((prev_done_transmit == 0) && (done_transmit == 1)) || (valid_out == 0))

            // First check if there is any data in the system.  There is data if and

            // only if count_data is greater than 0.  If this is not the case, then

            // no data can yet be outputted, valid_out is set to zero indicating

            // a request for new outputted data (also, it indicates that the

            // outputted data has not been updated and as such is not yet valid)

            if (count_data <= 0)

                  valid_out <= 0;

            // If count_data > 0 there is data that can be outputted.  The

            // data that is actually outputted is determined by the value of

            // the out_pointer register.

            else

                  begin

                  if (out_pointer == 1)

                        data_out <= data_in1;

                  else if (out_pointer == 2)
                        data_out <= data_in2;
                  else if (out_pointer == 3)
                        data_out <= data_in3;
                  else if (out_pointer == 4)
                        data_out <= data_in4;
                  else if (out_pointer == 5)
                        data_out <= data_in5;
                  else if (out_pointer == 6)
                        data_out <= data_in6;
                  else if (out_pointer == 7)
                        data_out <= data_in7;
                  else if (out_pointer == 8)
                        data_out <= data_in8;
                  else if (out_pointer == 9)
                        data_out <= data_in9;
                  else
                        data_out <= data_in10;
                  // Now that the outputted data has been refreshed, valid_out
                  // can be set high again.  Also, one data value has been
                  // used up so count_data is now one less than its previous value.
                  valid_out  <= 1;
                  count_data <= count_data - 1;
                  // out_pointer also has to be updated after new data has been
                  // outputted.  However, there are only 10 data slots.  Thus
                  // out_pointer is incremented by one in all cases except when
                  // it is 10.  In this case it is set back to 1, (creating
                  // a cyclic effect).
```

```
            if (out_pointer >= 10)

                  out_pointer <= 1;

            else

                  out_pointer <= out_pointer + 1;

            end

// Every clock cycle checks to see if count_data >= 10.  This means

// that more data values have been stored than there are slots for.

// This can cause problems as yet un-outputted data packets have

// been overwritten.

if (count_data >= 10)

      buffer_full <= 1;

else

      buffer_full <= 0;

// This FSM is in charge of creating 800 bit packets from the inputted

// 50 bit data values provided to the system.

case(state)

      // Note that this fsm starts by first building up the

      // data_in variable.  This variable is then stored permanently

      // in one of the data slots.  It can then be overwritten

      // with new data again.

state_wait:

      // Waiting for both valid_right and valid_left to become

      // valid.

      if (valid_in)

            // When it becomes valid, register the data value.

            // Called 'right' for historical reasons...

            // This becomes the bottom 50 bits of the data packet.
            // saved_state is stored as state_50, this is the
            // state the FSM will return to after it is done
            // acknowledging the intake of data.
            begin
            data_in[49:0]  <= data_right;
            state_saved    <= state_100;
            state          <= state_ackin;
            end
      else
            state <= state_wait;
            // Wait for valid_in to become valid
            // Register data_right as bits 99 to 50.
state_100:
      if (valid_in)
            begin
            data_in[99:50] <= data_right;
            state          <= state_ackin;
            state_saved    <= state_150;
            end
      else
            state <= state_100;
            // Wait for valid_in to become valid
            // Register data_right as bits 149 to 100
state_150:
      if (valid_in)
            begin
            data_in[149:100] <= data_right;
            state           <= state_ackin;
            state_saved     <= state_200;
            end
      else
            state <= state_150;
            // Wait for valid_in to become valid
            // Register data_right as bits 199 to 150
state_200:
      if (valid_in)
            begin
            data_in[199:150] <= data_right;
            state           <= state_ackin;
            state_saved     <= state_250;
            end
      else
            state <= state_200;
            // Wait for valid_in to become valid
            // Register data_right as bits 249 to 200
state_250:
      if (valid_in)
            begin
            data_in[249:200] <= data_right;
            state           <= state_ackin;
            state_saved     <= state_300;
            end
      else
            state <= state_250;
            // Wait for valid_in to become valid

            // Register data_right as bits 299 to 250

state_300:

      if (valid_in)

            begin

            data_in[299:250] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_350;

            end

      else

            state <= state_300;

            // Wait for valid_in to become valid

            // Register data_right as bits 349 to 300

state_350:

      if (valid_in)

            begin

            data_in[349:300] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_400;

            end

      else

            state <= state_350;

            // Wait for valid_in to become valid

            // Register data_right as bits 399 to 350

state_400:

      if (valid_in)

            begin
            data_in[399:350] <= data_right;
            state            <= state_ackin;
            state_saved      <= state_450;
            end
      else
            state <= state_400;
            // Wait for valid_in to become valid
            // Register data_right as bits 449 to 400
state_450:
      if (valid_in)
            begin
            data_in[449:400] <= data_right;
            state            <= state_ackin;
            state_saved      <= state_500;
            end
      else
            state <= state_450;
            // Wait for valid_in to become valid
            // Register data_right as bits 499 to 450
state_500:
      if (valid_in)
            begin
            data_in[499:450] <= data_right;
            state            <= state_ackin;
            state_saved      <= state_550;
            end
      else
```

```
            state <= state_500;

      // Wait for valid_in to become valid

      // Register data_right as bits 549 to 500

state_550:

      if (valid_in)

            begin

            data_in[549:500] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_600;

            end

      else

            state <= state_550;

            // Wait for valid_in to become valid

            // Register data_right as bits 599 to 550

state_600:

      if (valid_in)

            begin

            data_in[599:550] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_650;

            end

      else

            state <= state_600;

            // Wait for valid_in to become valid

            // Register data_right as bits 649 to 600

state_650:

      if (valid_in)
```

```
            begin

            data_in[649:600] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_700;

            end

      else

            state <= state_650;

            // Wait for valid_in to become valid

            // Register data_right as bits 699 to 650

state_700:

      if (valid_in)

            begin

            data_in[699:650] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_750;

            end

      else

            state <= state_700;

            // Wait for valid_in to become valid

            // Register data_right as bits 749 to 700

state_750:

      if (valid_in)

            begin

            data_in[749:700] <= data_right;

            state            <= state_ackin;

            state_saved      <= state_800;

            end
      else
            state <= state_750;
            // Wait for valid_in to become valid
            // Register data_right as bits 799 to 750
state_800:
      if (valid_in)
            begin
            data_in[799:750] <= data_right;
            state            <= state_ackin;
            state_saved      <= state_store;
            end
      else
            state <= state_800;
state_ackin:
      // Acknowledge that the data has been
      // been read by setting valid_read to 1.
      // Transistion to the second acknowledge state.
      begin
      valid_read <= 1;
      state <= state_ackin2;
      end
state_ackin2:
      // Wait for valid_in to become
      // low before transitioning back to the saved state.
      // This ensures that on transitioning back, valid_in
      // is not still high from the previous
      // registered data.  Such a thing could potentially cause
      // the same values to be registered twice.

      if (valid_in)

            state <= state_ackin2;

      else

            begin

            valid_read <= 0;

            state <= state_saved;

            end

state_store:

      // Store the 800 bit data packet that was created

      // in the previous states.

      begin

      // First update the value of count_data.  As yet

      // there is no safety check implemented for making

      // sure the buffer does not overflow.  If this is

      // to be implimented it would be done so here

      // (i.e. if count_data > 10 don't store anything yet)

      // However, allowing for overflow makes sure that the

      // front end of the system, from the A/D, does not

      // get hung up waiting for the buffer to empty.

      count_data <= count_data + 1;

      // The position the data is stored in is dictated

      // by the in_pointer.

      if (in_pointer == 1)

            data_in1 <= data_in;

      else if (in_pointer == 2)

            data_in2 <= data_in;
```

```
else if (in_pointer == 3)

      data_in3 <= data_in;

else if (in_pointer == 4)

      data_in4 <= data_in;

else if (in_pointer == 5)

      data_in5 <= data_in;

else if (in_pointer == 6)

      data_in6 <= data_in;

else if (in_pointer == 7)

      data_in7 <= data_in;

else if (in_pointer == 8)

      data_in8 <= data_in;

else if (in_pointer == 9)

      data_in9 <= data_in;

else

      data_in10 <= data_in;

// After the data has been stored the value of in_pointer

// is updated.  Note that the in_pointer is circular

// in the same fashion as the out_pointer.  Namely, once

// it reaches a value of 10 it cycles back to 1.

if (in_pointer >= 10)

      in_pointer <= 1;

else

      in_pointer <= in_pointer + 1;

// After the data has been stored the FSM can transition

// back to the wait state and re-write the data_in

// registers to build a new packet.
```

```
                  state <= state_wait;

                  end

            endcase

            end

endmodule
```

```
`timescale 1ns / 1ps

// Module: buffer_nivedita

// clock: internal clock of the entire system

// reset: zeros the outputs, puts the fsm in the wait state

// valid_right: signals that the data representing the right-sided sound is valid

// valid_left: signals that the data representing the left-sided sound is valid

// request_data_out: when high, signals that the transmitter had taken in the outputted data

//                and will soon be ready for new data

// data_right: 50 bit data chunk (compressed & formatted for error correction) representing

//       5 signed 16 bit numbers from the right-sided data

// data_left: 50 bit data chunk (compressed & formatted for error correction) representing

//            5 signed 16 bit numbers from the left-sided data

// data_out: 800 bit data packed comprised of 8 left/right data pairs outputted to the

//           transmitter

// valid_out: when high, signifies that the data present on data_out is a valid packet that

//            is to be transmitted

// valid_read: outputted by the FIFO buffer signifying that the buffer has taken in both the

//             data_right and data_left data values (active high for one clock cycle)

// buffer_full: indicates that the FIFO buffer has been filled

module buffer_nivedita(clock, reset, valid_in, request_data_out,

                       data_right, data_out, valid_out, valid_read, buffer_full);

input  clock;
input  reset;
input  valid_in;
input  request_data_out;
input  [799:0] data_right;
output [49:0] data_out;
output valid_out;
output valid_read;
output buffer_full;
// INTERNAL
//
// in_pointer: this FIFO can store up to 10 800 bit packets at a
//             time (hence there are slots 1 to 10).  The value
//             of the in_pointer dictates which slot new data
//             is to be written to.
// out_pointer: dictates wich slot data is to be taken from when
//              outputting new data
// count_data: represents how many data packets are currently being
//             stored in the system.  Does not include packets that
//             have already been outputted as these can be over-
//             written without consequence.
// state: 5 bit value representing the current state of the system
// state_saved: the FSM implimented here uses a sort of 'jump back'
//              system whereby it will enter a state and then jump
//              back to a previous state, the value of which can
//              vary.  The state_saved is the previous state that it
//              is to jump back to.
reg [10:0] in_pointer;
reg [10:0] out_pointer;

reg [9:0] count_data;

reg [5:0] state;

reg [4:0] state_saved;

// buffer_full: when high, signifies that all ten slots have been

//              written to with data that has not yet been outputted

//              Outputted to an LED will let the user know if there

//              has been an error and why there has been an error

//              Note that this can be set low again if the buffer

//              is emptied

// valid_out: the data on the output terminals is valid when this

//            signal is high

// prev_request_data_out: the valud of the request_data_out signal on the

//                        previously determined clock cycle.  Useful in

//                        looking for the posedge of request_data_out

// valid_read: signifies that both data_right and data_left have been

//             registered. High for one clock cycle when this occurs.

reg buffer_full;

reg valid_out;

reg prev_request_data_out;

reg valid_read;

// Here the 10 data slots for the 800 bit packets are declared.

// Note that data_in is not a slot, but is rather a place to

// store data as it comes in in the 50 bit packets.

reg [49:0] data_out;

reg [799:0] data_in;

reg [49:0] data_in1;
reg [49:0] data_in2;

reg [49:0] data_in3;

reg [49:0] data_in4;

reg [49:0] data_in5;

reg [49:0] data_in6;

reg [49:0] data_in7;

reg [49:0] data_in8;

reg [49:0] data_in9;

reg [49:0] data_in10;

reg [49:0] data_in11;

reg [49:0] data_in12;

reg [49:0] data_in13;

reg [49:0] data_in14;

reg [49:0] data_in15;

reg [49:0] data_in16;

reg [49:0] data_in17;

reg [49:0] data_in18;

reg [49:0] data_in19;

reg [49:0] data_in20;

reg [49:0] data_in21;

reg [49:0] data_in22;

reg [49:0] data_in23;

reg [49:0] data_in24;

reg [49:0] data_in25;

reg [49:0] data_in26;

reg [49:0] data_in27;

reg [49:0] data_in28;
reg [49:0] data_in29;

reg [49:0] data_in30;

reg [49:0] data_in31;

reg [49:0] data_in32;

reg [49:0] data_in33;

reg [49:0] data_in34;

reg [49:0] data_in35;

reg [49:0] data_in36;

reg [49:0] data_in37;

reg [49:0] data_in38;

reg [49:0] data_in39;

reg [49:0] data_in40;

reg [49:0] data_in41;

reg [49:0] data_in42;

reg [49:0] data_in43;

reg [49:0] data_in44;

reg [49:0] data_in45;

reg [49:0] data_in46;

reg [49:0] data_in47;

reg [49:0] data_in48;

reg [49:0] data_in49;

reg [49:0] data_in50;

reg [49:0] data_in51;

reg [49:0] data_in52;

reg [49:0] data_in53;

reg [49:0] data_in54;

reg [49:0] data_in55;
reg [49:0] data_in56;

reg [49:0] data_in57;

reg [49:0] data_in58;

reg [49:0] data_in59;

reg [49:0] data_in60;

reg [49:0] data_in61;

reg [49:0] data_in62;

reg [49:0] data_in63;

reg [49:0] data_in64;

reg [49:0] data_in65;

reg [49:0] data_in66;

reg [49:0] data_in67;

reg [49:0] data_in68;

reg [49:0] data_in69;

reg [49:0] data_in70;

reg [49:0] data_in71;

reg [49:0] data_in72;

reg [49:0] data_in73;

reg [49:0] data_in74;

reg [49:0] data_in75;

reg [49:0] data_in76;

reg [49:0] data_in77;

reg [49:0] data_in78;

reg [49:0] data_in79;

reg [49:0] data_in80;

reg [49:0] data_in81;

reg [49:0] data_in82;
reg [49:0] data_in83;

reg [49:0] data_in84;

reg [49:0] data_in85;

reg [49:0] data_in86;

reg [49:0] data_in87;

reg [49:0] data_in88;

reg [49:0] data_in89;

reg [49:0] data_in90;

reg [49:0] data_in91;

reg [49:0] data_in92;

reg [49:0] data_in93;

reg [49:0] data_in94;

reg [49:0] data_in95;

reg [49:0] data_in96;

reg [49:0] data_in97;

reg [49:0] data_in98;

reg [49:0] data_in99;

reg [49:0] data_in100;

reg [49:0] data_in101;

reg [49:0] data_in102;

reg [49:0] data_in103;

reg [49:0] data_in104;

reg [49:0] data_in105;

reg [49:0] data_in106;

reg [49:0] data_in107;

reg [49:0] data_in108;

reg [49:0] data_in109;
reg [49:0] data_in110;

reg [49:0] data_in111;

reg [49:0] data_in112;

reg [49:0] data_in113;

reg [49:0] data_in114;

reg [49:0] data_in115;

reg [49:0] data_in116;

reg [49:0] data_in117;

reg [49:0] data_in118;

reg [49:0] data_in119;

reg [49:0] data_in120;

reg [49:0] data_in121;

reg [49:0] data_in122;

reg [49:0] data_in123;

reg [49:0] data_in124;

reg [49:0] data_in125;

reg [49:0] data_in126;

reg [49:0] data_in127;

reg [49:0] data_in128;

reg [49:0] data_in129;

reg [49:0] data_in130;

reg [49:0] data_in131;

reg [49:0] data_in132;

reg [49:0] data_in133;

reg [49:0] data_in134;

reg [49:0] data_in135;

reg [49:0] data_in136;
reg [49:0] data_in137;

reg [49:0] data_in138;

reg [49:0] data_in139;

reg [49:0] data_in140;

reg [49:0] data_in141;

reg [49:0] data_in142;

reg [49:0] data_in143;

reg [49:0] data_in144;

reg [49:0] data_in145;

reg [49:0] data_in146;

reg [49:0] data_in147;

reg [49:0] data_in148;

reg [49:0] data_in149;

reg [49:0] data_in150;

reg [49:0] data_in151;

reg [49:0] data_in152;

reg [49:0] data_in153;

reg [49:0] data_in154;

reg [49:0] data_in155;

reg [49:0] data_in156;

reg [49:0] data_in157;

reg [49:0] data_in158;

reg [49:0] data_in159;

reg [49:0] data_in160;

// Here the various states of the fsm are parameterized.

parameter state_wait   = 0;

parameter state_ackin  = 1;
parameter state_ackin2 = 2;

parameter state_store  = 3;

always @ (posedge clock)

      if (reset)

            // on a reset all outputs are zeroed.  The in and

            // outpointers are both set to one (meaning that

            // incoming data will be written to slot one and

            // no data will be outputted until at least one

            // data packet has been stored)

            // The FSM is placed in the wait state.

            // Additionally, count_data is set to zero meaning

            // that there is no valid data stored in the FIFO.

            // ... which is why all of the data slots are

            // zeroed.  (data_in1, data_in2, etc.)

            begin

            valid_out    <= 0;

            data_out     <= 0;

            in_pointer   <= 1;

            out_pointer  <= 1;

            count_data   <= 0;

            valid_read   <= 0;

            buffer_full  <= 0;

            state        <= state_wait;

            data_in1  <= 0;

            data_in2  <= 0;

            data_in3  <= 0;

            data_in4  <= 0;
            data_in5  <= 0;

            data_in6  <= 0;

            data_in7  <= 0;

            data_in8  <= 0;

            data_in9  <= 0;

            data_in10 <= 0;

            // initialize the prev_request_data_out reg as well.

            prev_request_data_out <= 0;

            end

      else

            begin

            // Begin by registering the current request_data_out signal. (This change will

            // not take effect until the next clock cycle)

            prev_request_data_out <= request_data_out;

            // Now, check if there has been a 'posedge' of request_data_out or if valid_out

            // has been set to zero.  Either condition implies that there is an

            // impending request for new data to be outputted.

            //

            // Note that valid_out is only ever zero after a reset or if there was a

            // posedge of request_data_out and no data to output.

            if  (((prev_request_data_out == 0) && (request_data_out == 1)) || (valid_out == 0))

                  // First check if there is any data in the system.  There is data if and

                  // only if count_data is greater than 0.  If this is not the case, then

                  // no data can yet be outputted, valid_out is set to zero indicating

                  // a request for new outputted data (also, it indicates that the

                  // outputted data has not been updated and as such is not yet valid)

                  if (count_data <= 0)
                        valid_out <= 0;
                  // If count_data > 0 there is data that can be outputted.  The
                  // data that is actually outputted is determined by the value of
                  // the out_pointer register.
                  else
                        begin
                        if (out_pointer == 1)
                              data_out <= data_in1;
                        else if (out_pointer == 2)
                              data_out <= data_in2;
                        else if (out_pointer == 3)
                              data_out <= data_in3;
                        else if (out_pointer == 4)
                              data_out <= data_in4;
                        else if (out_pointer == 5)
                              data_out <= data_in5;
                        else if (out_pointer == 6)
                              data_out <= data_in6;
                        else if (out_pointer == 7)
                              data_out <= data_in7;
                        else if (out_pointer == 8)
                              data_out <= data_in8;
                        else if (out_pointer == 9)
                              data_out <= data_in9;
                        else if (out_pointer == 10)
                              data_out <= data_in10;
                        else if (out_pointer <= 11)
                              data_out <= data_in11;

                        else if (out_pointer <= 12)

                              data_out <= data_in12;

                        else if (out_pointer <= 13)

                              data_out <= data_in13;

                        else if (out_pointer <= 14)

                              data_out <= data_in14;

                        else if (out_pointer <= 15)

                              data_out <= data_in15;

                        else if (out_pointer <= 16)

                              data_out <= data_in16;

                        else if (out_pointer == 17)

                              data_out <= data_in17;

                        else if (out_pointer == 18)

                              data_out <= data_in18;

                        else if (out_pointer == 19)

                              data_out <= data_in19;

                        else if (out_pointer == 20)

                              data_out <= data_in20;

                        else if (out_pointer == 21)

                              data_out <= data_in21;

                        else if (out_pointer == 22)

                              data_out <= data_in22;

                        else if (out_pointer == 23)

                              data_out <= data_in23;

                        else if (out_pointer == 24)

                              data_out <= data_in24;
```

```
else if (out_pointer == 25)
      data_out <= data_in25;
else if (out_pointer == 26)
      data_out <= data_in26;
else if (out_pointer <= 27)
      data_out <= data_in27;
else if (out_pointer <= 28)
      data_out <= data_in28;
else if (out_pointer <= 29)
      data_out <= data_in29;
else if (out_pointer <= 30)
      data_out <= data_in30;
else if (out_pointer <= 31)
      data_out <= data_in31;
else if (out_pointer <= 32)
      data_out <= data_in32;
else if (out_pointer == 33)
      data_out <= data_in33;
else if (out_pointer == 34)
      data_out <= data_in34;
else if (out_pointer == 35)
      data_out <= data_in35;
else if (out_pointer == 36)
      data_out <= data_in36;
else if (out_pointer == 37)
      data_out <= data_in37;
else if (out_pointer == 38)
```

```
      data_out <= data_in38;
else if (out_pointer == 39)
      data_out <= data_in39;
else if (out_pointer == 40)
      data_out <= data_in40;
else if (out_pointer == 41)
      data_out <= data_in41;
else if (out_pointer == 42)
      data_out <= data_in42;
else if (out_pointer <= 43)
      data_out <= data_in43;
else if (out_pointer <= 44)
      data_out <= data_in44;
else if (out_pointer <= 45)
      data_out <= data_in45;
else if (out_pointer <= 46)
      data_out <= data_in46;
else if (out_pointer <= 47)
      data_out <= data_in47;
else if (out_pointer <= 48)
      data_out <= data_in48;
else if (out_pointer == 49)
      data_out <= data_in49;
else if (out_pointer == 50)
      data_out <= data_in50;
else if (out_pointer == 51)
      data_out <= data_in51;
```

```
else if (out_pointer == 52)
        data_out <= data_in52;
else if (out_pointer == 53)
        data_out <= data_in53;
else if (out_pointer == 54)
        data_out <= data_in54;
else if (out_pointer == 55)
        data_out <= data_in55;
else if (out_pointer == 56)
        data_out <= data_in56;
else if (out_pointer == 57)
        data_out <= data_in57;
else if (out_pointer == 58)
        data_out <= data_in58;
else if (out_pointer <= 59)
        data_out <= data_in59;
else if (out_pointer <= 60)
        data_out <= data_in60;
else if (out_pointer <= 61)
        data_out <= data_in61;
else if (out_pointer <= 62)
        data_out <= data_in62;
else if (out_pointer <= 63)
        data_out <= data_in63;
else if (out_pointer <= 64)
        data_out <= data_in64;
else if (out_pointer == 65)
```

```
      data_out <= data_in65;
else if (out_pointer == 66)
      data_out <= data_in66;
else if (out_pointer == 67)
      data_out <= data_in67;
else if (out_pointer == 68)
      data_out <= data_in68;
else if (out_pointer == 69)
      data_out <= data_in69;
else if (out_pointer == 70)
      data_out <= data_in70;
else if (out_pointer == 71)
      data_out <= data_in71;
else if (out_pointer == 72)
      data_out <= data_in72;
else if (out_pointer == 73)
      data_out <= data_in73;
else if (out_pointer == 74)
      data_out <= data_in74;
else if (out_pointer <= 75)
      data_out <= data_in75;
else if (out_pointer <= 76)
      data_out <= data_in76;
else if (out_pointer <= 77)
      data_out <= data_in77;
else if (out_pointer <= 78)
      data_out <= data_in78;
```

```
else if (out_pointer <= 79)
        data_out <= data_in79;
else if (out_pointer <= 80)
        data_out <= data_in80;
else if (out_pointer == 81)
        data_out <= data_in81;
else if (out_pointer == 82)
        data_out <= data_in82;
else if (out_pointer == 83)
        data_out <= data_in83;
else if (out_pointer == 84)
        data_out <= data_in84;
else if (out_pointer == 85)
        data_out <= data_in85;
else if (out_pointer == 86)
        data_out <= data_in86;
else if (out_pointer == 87)
        data_out <= data_in87;
else if (out_pointer == 88)
        data_out <= data_in88;
else if (out_pointer == 89)
        data_out <= data_in89;
else if (out_pointer == 90)
        data_out <= data_in90;
else if (out_pointer <= 91)
        data_out <= data_in91;
else if (out_pointer <= 92)
```

```
      data_out <= data_in92;
else if (out_pointer <= 93)
      data_out <= data_in93;
else if (out_pointer <= 94)
      data_out <= data_in94;
else if (out_pointer <= 95)
      data_out <= data_in95;
else if (out_pointer <= 96)
      data_out <= data_in96;
else if (out_pointer == 97)
      data_out <= data_in97;
else if (out_pointer == 98)
      data_out <= data_in98;
else if (out_pointer == 99)
      data_out <= data_in99;
else if (out_pointer == 100)
      data_out <= data_in100;
else if (out_pointer == 101)
      data_out <= data_in101;
else if (out_pointer == 102)
      data_out <= data_in102;
else if (out_pointer == 103)
      data_out <= data_in103;
else if (out_pointer == 104)
      data_out <= data_in104;
else if (out_pointer == 105)
      data_out <= data_in105;
```

```
else if (out_pointer == 106)

      data_out <= data_in106;

else if (out_pointer <= 107)

      data_out <= data_in107;

else if (out_pointer <= 108)

      data_out <= data_in108;

else if (out_pointer <= 109)

      data_out <= data_in109;

else if (out_pointer <= 110)

      data_out <= data_in110;

else if (out_pointer <= 111)

      data_out <= data_in111;

else if (out_pointer <= 112)

      data_out <= data_in112;

else if (out_pointer == 113)

      data_out <= data_in113;

else if (out_pointer == 114)

      data_out <= data_in114;

else if (out_pointer == 115)

      data_out <= data_in115;

else if (out_pointer == 116)

      data_out <= data_in116;

else if (out_pointer == 117)

      data_out <= data_in117;

else if (out_pointer == 118)

      data_out <= data_in118;

else if (out_pointer == 119)
```

```
      data_out <= data_in119;
else if (out_pointer == 120)
      data_out <= data_in120;
else if (out_pointer == 121)
      data_out <= data_in121;
else if (out_pointer == 122)
      data_out <= data_in122;
else if (out_pointer <= 123)
      data_out <= data_in123;
else if (out_pointer <= 124)
      data_out <= data_in124;
else if (out_pointer <= 125)
      data_out <= data_in125;
else if (out_pointer <= 126)
      data_out <= data_in126;
else if (out_pointer <= 127)
      data_out <= data_in127;
else if (out_pointer <= 128)
      data_out <= data_in128;
else if (out_pointer == 129)
      data_out <= data_in129;
else if (out_pointer == 130)
      data_out <= data_in130;
else if (out_pointer == 131)
      data_out <= data_in131;
else if (out_pointer == 132)
      data_out <= data_in132;
```

```
else if (out_pointer == 133)
      data_out <= data_in133;
else if (out_pointer == 134)
      data_out <= data_in134;
else if (out_pointer == 135)
      data_out <= data_in135;
else if (out_pointer == 136)
      data_out <= data_in36;
else if (out_pointer == 137)
      data_out <= data_in137;
else if (out_pointer == 138)
      data_out <= data_in138;
else if (out_pointer == 139)
      data_out <= data_in139;
else if (out_pointer <= 140)
      data_out <= data_in140;
else if (out_pointer <= 141)
      data_out <= data_in141;
else if (out_pointer <= 142)
      data_out <= data_in142;
else if (out_pointer <= 143)
      data_out <= data_in143;
else if (out_pointer <= 144)
      data_out <= data_in144;
else if (out_pointer == 145)
      data_out <= data_in145;
else if (out_pointer == 146)
```

```
          data_out <= data_in146;
      else if (out_pointer == 147)
          data_out <= data_in147;
      else if (out_pointer == 148)
          data_out <= data_in148;
      else if (out_pointer == 149)
          data_out <= data_in149;
      else if (out_pointer == 150)
          data_out <= data_in150;
      else if (out_pointer == 151)
          data_out <= data_in151;
      else if (out_pointer == 152)
          data_out <= data_in152;
      else if (out_pointer == 153)
          data_out <= data_in153;
      else if (out_pointer == 154)
          data_out <= data_in154;
      else if (out_pointer <= 155)
          data_out <= data_in155;
      else if (out_pointer <= 156)
          data_out <= data_in156;
      else if (out_pointer <= 157)
          data_out <= data_in157;
      else if (out_pointer <= 158)
          data_out <= data_in158;
      else if (out_pointer <= 159)
          data_out <= data_in159;
```

```
            else
                  data_out <= data_in160;
            // Now that the outputted data has been refreshed, valid_out
            // can be set high again.  Also, one data value has been
            // used up so count_data is now one less than its previous value.
            valid_out <= 1;
            count_data   <= count_data - 1;
            // out_pointer also has to be updated after new data has been
            // outputted.  However, there are only 10 data slots.  Thus
            // out_pointer is incremented by one in all cases except when
            // it is 10.  In this case it is set back to 1, (creating
            // a cyclic effect).
            if (out_pointer >= 160)
                  out_pointer <= 1;
            else
                  out_pointer <= out_pointer + 1;
            end
// Every clock cycle check to see if count_data >= 10.  This means
// that more data values have been stored than there are slots for.
// This can cause problems as yet un-outputted data packets have
// been overwritten.
if (count_data >= 160)
      buffer_full <= 1;
else
      buffer_full <= 0;
// This FSM is in charge of creating packets from the inputted
// left and right data values provided to the system.
```

```
case(state)

      // Note that this state starts by first building up the

      // data_in variable.  This variable is then stored permanently

      // in one of the data slots.  It can then be overwritted

      // with new data again.

state_wait:

      // Waiting for both valid_right and valid_left to become

      // valid. (high)

      if (valid_in)

            begin

            data_in     <= data_right;

            state_saved <= state_store;

            state       <= state_ackin;

            end

      else

            state <= state_wait;


state_ackin:

      // Acknowledge that the left and write data have

      // been read by setting valid_read to 1.

      // Transistion to the second acknowledge state.

      begin

      valid_read <= 1;

      state      <= state_ackin2;

      end

state_ackin2:

      // Wait for valid_in to become
      // low before transitioning back to the saved state.
      // This ensures that on transitioning back, valid_right
      // and valid_left aren't still high from the previous
      // registered data.  Such a thing could potentially cause
      // the same values to be registered twice.
      if (valid_in)
            state <= state_ackin2;
      else
            begin
            valid_read <= 0;
            state <= state_saved;
            end
state_store:
      // Store the 10 bit data packet that was created
      // in the previous states.
      begin
      // First update the value of count_data.  As yet
      // there is no safety check implemented for making
      // sure the buffer does not overflow.  If this is
      // to be implimented it would be done so here
      // (i.e. if count_data > 10 don't store anything yet)
      // However, allowing for overflow makes sure that the
      // front end of the system, from the A/D, does not
      // get hung up waiting for the buffer to empty.
      count_data <= count_data + 16;
      // The position the datis is stored in is dictated
      // by the in_pointer.
```

```
if (in_pointer == 1)
      begin
      data_in1 <= data_in[49:0];
      data_in2 <= data_in[99:50];
      data_in3 <= data_in[149:100];
      data_in4 <= data_in[199:150];
      data_in5 <= data_in[249:200];
      data_in6 <= data_in[299:250];
      data_in7 <= data_in[349:300];
      data_in8 <= data_in[399:350];
      data_in9 <= data_in[449:400];
      data_in10 <= data_in[499:450];
      data_in11 <= data_in[549:500];
      data_in12 <= data_in[599:550];
      data_in13 <= data_in[649:600];
      data_in14 <= data_in[699:650];
      data_in15 <= data_in[749:700];
      data_in16 <= data_in[799:750];
      in_pointer <= 17;
      end
else if (in_pointer == 17)
      begin
      data_in17 <= data_in[49:0];
      data_in18 <= data_in[99:50];
      data_in19 <= data_in[149:100];
      data_in20 <= data_in[199:150];
      data_in21 <= data_in[249:200];
      data_in22 <= data_in[299:250];
      data_in23 <= data_in[349:300];
      data_in24 <= data_in[399:350];
      data_in25 <= data_in[449:400];
      data_in26 <= data_in[499:450];
      data_in27 <= data_in[549:500];
      data_in28 <= data_in[599:550];
      data_in29 <= data_in[649:600];
      data_in30 <= data_in[699:650];
      data_in31 <= data_in[749:700];
      data_in32 <= data_in[799:750];
      in_pointer <= 33;
      end
else if (in_pointer == 33)
      begin
      data_in33 <= data_in[49:0];
      data_in34 <= data_in[99:50];
      data_in35 <= data_in[149:100];
      data_in36 <= data_in[199:150];
      data_in37 <= data_in[249:200];
      data_in38 <= data_in[299:250];
      data_in39 <= data_in[349:300];
      data_in40 <= data_in[399:350];
      data_in41 <= data_in[449:400];
      data_in42 <= data_in[499:450];
      data_in43 <= data_in[549:500];
      data_in44 <= data_in[599:550];
      data_in45 <= data_in[649:600];

      data_in46 <= data_in[699:650];

      data_in47 <= data_in[749:700];

      data_in48 <= data_in[799:750];

      in_pointer <= 49;

      end

else if (in_pointer == 49)

      begin

      data_in49 <= data_in[49:0];

      data_in50 <= data_in[99:50];

      data_in51 <= data_in[149:100];

      data_in52 <= data_in[199:150];

      data_in53 <= data_in[249:200];

      data_in54 <= data_in[299:250];

      data_in55 <= data_in[349:300];

      data_in56 <= data_in[399:350];

      data_in57 <= data_in[449:400];

      data_in58 <= data_in[499:450];

      data_in59 <= data_in[549:500];

      data_in60 <= data_in[599:550];

      data_in61 <= data_in[649:600];

      data_in62 <= data_in[699:650];

      data_in63 <= data_in[749:700];

      data_in64 <= data_in[799:750];

      in_pointer <= 65;

      end

else if (in_pointer == 65)
```

```
      begin
      data_in65 <= data_in[49:0];
      data_in66 <= data_in[99:50];
      data_in67 <= data_in[149:100];
      data_in68 <= data_in[199:150];
      data_in69 <= data_in[249:200];
      data_in70 <= data_in[299:250];
      data_in71 <= data_in[349:300];
      data_in72 <= data_in[399:350];
      data_in73 <= data_in[449:400];
      data_in74 <= data_in[499:450];
      data_in75 <= data_in[549:500];
      data_in76 <= data_in[599:550];
      data_in77 <= data_in[649:600];
      data_in78 <= data_in[699:650];
      data_in79 <= data_in[749:700];
      data_in80 <= data_in[799:750];
      in_pointer <= 81;
      end
else if (in_pointer == 81)
      begin
      data_in81 <= data_in[49:0];
      data_in82 <= data_in[99:50];
      data_in83 <= data_in[149:100];
      data_in84 <= data_in[199:150];
      data_in85 <= data_in[249:200];
      data_in86 <= data_in[299:250];
      data_in87 <= data_in[349:300];
      data_in88 <= data_in[399:350];
      data_in89 <= data_in[449:400];
      data_in90 <= data_in[499:450];
      data_in91 <= data_in[549:500];
      data_in92 <= data_in[599:550];
      data_in93 <= data_in[649:600];
      data_in94 <= data_in[699:650];
      data_in95 <= data_in[749:700];
      data_in96 <= data_in[799:750];
      in_pointer <= 97;
      end
else if (in_pointer == 97)
      begin
      data_in97 <= data_in[49:0];
      data_in98 <= data_in[99:50];
      data_in99 <= data_in[149:100];
      data_in100 <= data_in[199:150];
      data_in101 <= data_in[249:200];
      data_in102 <= data_in[299:250];
      data_in103 <= data_in[349:300];
      data_in104 <= data_in[399:350];
      data_in105 <= data_in[449:400];
      data_in106 <= data_in[499:450];
      data_in107 <= data_in[549:500];
      data_in108 <= data_in[599:550];
      data_in109 <= data_in[649:600];
      data_in110 <= data_in[699:650];
      data_in111 <= data_in[749:700];
      data_in112 <= data_in[799:750];
      in_pointer <= 113;
      end
else if (in_pointer == 113)
      begin
      data_in113 <= data_in[49:0];
      data_in114 <= data_in[99:50];
      data_in115 <= data_in[149:100];
      data_in116 <= data_in[199:150];
      data_in117 <= data_in[249:200];
      data_in118 <= data_in[299:250];
      data_in119 <= data_in[349:300];
      data_in120 <= data_in[399:350];
      data_in121 <= data_in[449:400];
      data_in122 <= data_in[499:450];
      data_in123 <= data_in[549:500];
      data_in124 <= data_in[599:550];
      data_in125 <= data_in[649:600];
      data_in126 <= data_in[699:650];
      data_in127 <= data_in[749:700];
      data_in128 <= data_in[799:750];
      in_pointer <= 129;
      end
else if (in_pointer == 129)
      begin
```

```
        data_in129 <= data_in[49:0];

        data_in130 <= data_in[99:50];

        data_in131 <= data_in[149:100];

        data_in132 <= data_in[199:150];

        data_in133 <= data_in[249:200];

        data_in134 <= data_in[299:250];

        data_in135 <= data_in[349:300];

        data_in136 <= data_in[399:350];

        data_in137 <= data_in[449:400];

        data_in138 <= data_in[499:450];

        data_in139 <= data_in[549:500];

        data_in140 <= data_in[599:550];

        data_in141 <= data_in[649:600];

        data_in142 <= data_in[699:650];

        data_in143 <= data_in[749:700];

        data_in144 <= data_in[799:750];

        in_pointer <= 145;

        end

else

        begin

        data_in145 <= data_in[49:0];

        data_in146 <= data_in[99:50];

        data_in147 <= data_in[149:100];

        data_in148 <= data_in[199:150];

        data_in149 <= data_in[249:200];

        data_in150 <= data_in[299:250];

        data_in151 <= data_in[349:300];
```

```
                    data_in152 <= data_in[399:350];

                    data_in153 <= data_in[449:400];

                    data_in154 <= data_in[499:450];

                    data_in155 <= data_in[549:500];

                    data_in156 <= data_in[599:550];

                    data_in157 <= data_in[649:600];

                    data_in158 <= data_in[699:650];

                    data_in159 <= data_in[749:700];

                    data_in160 <= data_in[799:750];

                    in_pointer <= 1;

                    end

              // After the data has been stored the FSM can transition

              // back to the wait state and re-write the data_in

              // registers to build a new packet.

              state <= state_wait;

              end

        endcase

        end

endmodule
```

```
//////////CONFIGURATION FSM

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Nivedita Chandrasekaran
// Create Date:    16:27:37 04/29/2007
// Module Name:    ConfigureFSM
// Description: This module configures the CC2420 wireless chip. It contains an FSM that
//1. Performs a hard pin reset on the chip
//2. Strobes the command strobe on the chip that turns on the CC2420 oscillator - IMPORTANT
//3. Writes data to 5 16-bit configuration registers
//4. Writes a Personal Area Network ID (PANID) to CC2420 RAM
```

```
//5. Writes a Short Address (a device ID) to CC2420 RAM
//It also takes in a:
//CHANNEL - frequency channel on which the wireless chip will send packets, 8 bits
//PANID - 16 bit number that defines the Personal Area Network ID. With this, chips with
//automatic address recognition can only send and receive packets from other chips with the
//same PANID.
//SHORTADDR - The device address of the chip. A reciever w/automatic address recognition will
receive
//packets only from a transmitter with same PANID and if the destination address in the sent
packet
//matches the reciever's device address.
//and outputs:
//READPANID - Reads out PANID from CC2420 RAM.
//READSHORTADDR - Reads out SHORTADDR from CC2420 RAM.
//mem2_counter - a counter for the second memory write stage - as this increments, the FSM
passes
//through the second stage of writing the PANID to memory
//configured - IMPORTANT. This signal goes high when the configuration is complete and is held
high
//until the configuration module is reset. In higher level modules,this is used as an enable
signal to
//other modules (like transmit and receive) which perform chip operations.
//All names for configuration registers can be found in the CC2420 chip documentation.
////////////////////////////////////////////////////////////////////////////////
module ConfigureFSM(clock, reset, reset_chipn, CHANNEL, PANID, SHORTADDR,
                                                CSn, SI, SCLK, SO, begin_config, done_config,
state,
                                                READPANID, READSHORTADDR, mem2_counter,
configured);
    input clock; //module clocked at 10MHz.
         input reset; //system reset
         input SO; //input from CC2420
         output mem2_counter;
         output reg configured;

         input begin_config; //enable signal to this FSM

         input[7:0] CHANNEL;    //8bit unsigned channel number, will construct frequency control
from this
         input[15:0] PANID;             //16bit unsigned Personal Area Network (PAN)
Identification number
         input[15:0] SHORTADDR; //16bit unsigned short address for Device Identification
         wire[15:0] PANID, SHORTADDR;
         wire[7:0] CHANNEL;

         output reset_chipn;    //hard reset, connected to CC2420's reset pin - ACTIVE LOW
         reg reset_chipn;
         output[3:0] state;

         output CSn;                    //chip enable (ACTIVE LOW)
         output SI;                     //chip input (1 bit)
         output SCLK;                   //chip clock, flipped version of this module's clock
         wire SCLK = ~clock; //want to make sure that SI straddles rising edge of SCLK
         reg CSn, SI;
    output done_config; //sends out the "configuration completed signal"
         reg done_config;

        //FSM states
        parameter RESET = 0;
```

```
	parameter STROBE = 1;
	parameter WAITLONG = 2;
	parameter REGWRITE = 3;
	parameter WAIT2 = 4;
	parameter MEM1WRITE = 5;
	parameter WAIT3 = 6;
	parameter MEM2WRITE = 7;
	parameter WAIT4 = 8;
	parameter ASKREAD1 = 9;
	parameter WAIT5 = 10;
	parameter ASKREAD2 = 11;
	parameter IDLE = 12;

	//state registers
	reg[3:0] state, next;

	//timers for each state
	reg[4:0] r_counter;
	reg[3:0] strobe_counter;
	reg[14:0] osc_counter;
	reg[6:0] reg_counter;
	reg[5:0] mem1_counter;
	reg[5:0] mem2_counter;
	reg[5:0] read1_counter;
	reg[5:0] read2_counter;
	//******Regs for storing configuration data of CC2420******
	//REGISTER WRITE VALUES
	//format for regs:
	//[24:16] = address of register to write
	//[15:0] = data to write to register
	//i.e. reg_SAMPLE = 24'h_addr_addr_data_data_data_data
	wire[23:0] reg_MDMCTRL0;
	wire[23:0] reg_MDMCTRL1;
	wire[23:0] reg_IOCFG0;
	wire[23:0] reg_SECCTRL0;
	assign reg_MDMCTRL0 = 24'h110AF2; //automatic address recog
	//assign reg_MDMCTRL0 = 24'h1102F2;	//NO automatic address recog
	assign reg_MDMCTRL1 = 24'h120500;
	assign reg_IOCFG0 = 24'h1C007F;
	assign reg_SECCTRL0  = 24'h1901C4;

	//wire[23:0] reg_FSCTRL; //value created on a reset

	//FSCTRL Data - constructing from CHANNEL
	 // f = (UINT16) (CHANNEL - 11); // Subtract the base channel
	 // f = f + (f << 2);			// Multiply with 5, which is the channel
spacing
	 // f = f + 357 + 0x4000;		// 357 is 2405-2048, 0x4000 is LOCK_THR = 1
	 //reg[15:0] FSCTRL_data; //value constructed on a reset
	 //reg_FSCTRL <= {8'h18, (((CHANNEL - 11) * 5) + 357 + 16'h4000)};


	//COMMAND STROBE REGISTER
	reg [7:0] STROBE_SXOCSN;
	//REG WRITE REGISTER
	reg [119:0] WRITEREG;
	//MEMWRITE REGISTER
	reg [31:0] MEMPANID;
	reg [31:0]MEMSHORTADDR;
```

```
        //registers containing memory read command strobes
        reg [15:0] MEMPANIDREAD;
        reg [15:0]MEMSHORTADDRREAD;
        //registers that will contain data read from memory
        output [15:0] READPANID;
        reg [15:0] READPANID;
        output [15:0] READSHORTADDR;
        reg [15:0] READSHORTADDR;


always @ (posedge clock)
begin
        //if enable high
        if(reset || begin_config)
        begin
                //if a reset, reconfigure the chip
                        //initialize counters
                        r_counter <= 0;
                        strobe_counter <= 0;
                        reg_counter <= 0;
                        osc_counter <= 0;
                        mem1_counter <= 0;
                        mem2_counter <= 0;
                        read1_counter <= 0;
                        read2_counter <= 0;
                        //initialize FSM
                        state <= RESET;
                        //initialize ouputs
                        done_config<=0;
                        read1_counter<=0;
                        read2_counter<=0;
                        CSn <= 1;
                        SI <= 0;
                        configured <= 0;
                        //initialize configuration registers
                        WRITEREG <= ({reg_MDMCTRL0[23:0], reg_MDMCTRL1[23:0],reg_IOCFG0[23:0],
reg_SECCTRL0[23:0],{8'h18, (((CHANNEL - 11) * 5) + 357 + 16'h4000)}});
                        //init PANID and SHORTADDR registers - these contain command to write to
memory AS WELL AS
                        //data to be written into memory address
                        //NOTE ORDER y
                        MEMPANID <= {1'b1, 1'b1, 8'h68, 6'b000000, PANID[7:0], PANID[15:8]};
                        MEMPANIDREAD <= {1'b1, 1'b1, 8'h68, 6'b100000};
                        //init SHORTADDR registers
                        MEMSHORTADDR <= {1'b1, 1'b1, 8'h6A, 6'b00000, SHORTADDR[7:0],
SHORTADDR[15:8]};
                        MEMSHORTADDRREAD <= {1'b1, 1'b1, 8'h6A, 6'b100000};
                        //init data storage regs that will be filled on a read to CC2420 RAM
                        READPANID <= 0;
                        READSHORTADDR <= 0;
                        //COMMAND STROBE REGISTER - Need to strobe oscillator to
                        //enable other CC2420 register accesses
                        STROBE_SXOCSN <= 8'h01;

        end
        else
                begin
                        //how to increment state counters
                        r_counter <=            ((state==RESET) ? (r_counter + 1) : r_counter);
                        osc_counter <= ((state==WAITLONG) ? (osc_counter + 1) : osc_counter);
                        strobe_counter <= ((state==STROBE) ? (strobe_counter + 1) :
strobe_counter);
                        reg_counter <=  ((state==REGWRITE) ? (reg_counter + 1) : reg_counter);
                        mem1_counter <=         ((state==MEM1WRITE) ? (mem1_counter + 1) :
mem1_counter);
                        mem2_counter <=         ((state==MEM2WRITE) ? (mem2_counter + 1) :
mem2_counter);
                        read1_counter <= ((state==ASKREAD1) ? (read1_counter + 1) :
read1_counter);
                        read2_counter <= ((state==ASKREAD2) ? (read2_counter + 1) :
read2_counter);
                        //CSn is tricky, since CC2420 takes in SI on rising edge of SCLK
                        //as a result, CSn is as follows:
                        CSn <=                          (!(
((state==STROBE)&&(strobe_counter<8)) ||
                                                                                    ((state==REGWRI
TE) && (reg_counter<120)) ||
(state==WAITLONG) ||
                                                                                    ((state==MEM1WR
ITE) && (mem1_counter<32))||
                                                                                    ((state==MEM2WR
ITE) && (mem2_counter<32)) ||
                                                                                    ((state==ASKREA
D1) && (read1_counter<32)) ||
                                                                                    ((state==ASKREA
D2) && (read2_counter<32))
                                                                           ));
                        //deals with shifting out bits in configuration registers
                        STROBE_SXOCSN <= ((state==STROBE) ? ({STROBE_SXOCSN[6:0], 1'b0}) :
STROBE_SXOCSN);
                        WRITEREG          <= ((state==REGWRITE) ? ({WRITEREG[118:0], 1'b0}) :
WRITEREG);
                        MEMPANID          <= ((state==MEM1WRITE) ? ({MEMPANID[30:0], 1'b0}) :
MEMPANID);
                        MEMSHORTADDR  <= ((state==MEM2WRITE) ? ({MEMSHORTADDR[30:0], 1'b0}) :
MEMSHORTADDR);
                        //shift out bits for read command in ASKREAD1 and ASKREAD2 states
                        MEMPANIDREAD  <= ((state==ASKREAD1) ? ({MEMPANIDREAD[14:0], 1'b0}) :
MEMPANIDREAD);
                        MEMSHORTADDRREAD  <= ((state==ASKREAD2) ? ({MEMSHORTADDRREAD[14:0],
1'b0}) : MEMSHORTADDRREAD);
                        //capture SO bits in data capture registers at appropriate times in read
states
                        READPANID               <= (((state==ASKREAD1)&&(read1_counter>15)) ?
({READPANID[15:0], SO}) : READPANID);
                        READSHORTADDR  <= (((state==ASKREAD2)&&(read2_counter>15)) ?
({READSHORTADDR[15:0], SO}) : READSHORTADDR);
        //Logic for SI values
        case (state)
                STROBE: begin
                                SI <= STROBE_SXOCSN[7];
                        end
                WAITLONG: begin
                        SI <= 0;
                end
                REGWRITE: begin
                                SI <= WRITEREG[119];
                        end
                MEM1WRITE: begin
                                SI <= MEMPANID[31];
                        end
                MEM2WRITE: begin
                                SI <= MEMSHORTADDR[31];
                        end
                ASKREAD1: begin
                                SI <= MEMPANIDREAD[15];
                        end
                ASKREAD2: begin
                                SI <= MEMSHORTADDRREAD[15];
                        end
                IDLE: begin
                                SI <= 0;
                                configured <= 1;
                        end
        endcase
                        //done_config high when last state reached
                        done_config <=  (state==IDLE);
                        //update state on every 10Mhz clock cycle
                        state <= next;
        end
end

always @ (r_counter or strobe_counter or reg_counter or mem1_counter or mem2_counter or
osc_counter or read1_counter or read2_counter or state)
begin
next = state;
case (state)
        //on a RESET state, pull reset to 0 and then to 1. Then, wait for
        //AT LEAST 1 us
        //Waiting period necessary for capacitors/hardware on CC2420 to properly
        //discharge.
        RESET: begin
                                reset_chipn = (!(r_counter==0));
                                if(r_counter < 17)
                                begin
                                        next = RESET;
                                end
                                else if(r_counter==17)
                                begin
                                        next = STROBE;  //once waiting period over, move to
STROBE state
                                end
                        end
        //In strobe state, pass 8bit command 8'h01
        //to CC2420. This strobes the SXOCSN register on the chip, which turns on the oscillator
        //in the chip. Without this oscillator on, no configuration registers can be written.
        STROBE: begin
                                if(strobe_counter < 8)
                                begin
                                        next = STROBE;
                                end
                                else if(strobe_counter == 8)
                                begin
                                        next = WAITLONG;
                                end
                        end
//WAIT FOR CRYSTAL OSCILLATOR TO TURN ON
```

```
        WAITLONG: begin
                                if(osc_counter < 20000)
                                begin
                                        next = WAITLONG;
                                end
                                else if(osc_counter == 20000)
                                begin
                                        next = REGWRITE;
                                end
                        end
        //Enter REGWRITE state. Passes 5 24-bit commands to configure 5 registers via SPI
interface.
        // 1. 24'h110AF2 - Turn on automatic packet acknowledgement
        // 2. 24'h120500 - Set the correlation threshold to 20
        // 3. 24'h1C007F - Set the FIFOP threshold to maximum --> WATCHOUT MAY NEED TO DECREASE
THIS
        // 4. 24'h1C01C4 - Turn off security enable
        // 5. 24'h18---- - Set the RF channel
        REGWRITE: begin
                                if(reg_counter<120)
                                begin
                                        next = REGWRITE;
                                end
                                else if((reg_counter == 120))
                                begin
                                        next = WAIT2;
                                end
                                end
        //Insert a WAIT state for safety's sake - pull CSn high for long enough so that
        //junky values of SI not pulled into SPI interface to CC2420
        WAIT2: begin
                        //next = MEM1WRITE;
                        next = MEM2WRITE;
                        end
        //Write the of the chip to CC2420 RAM
        //31-bit command serialized to CC2420
        MEM1WRITE: begin
                                if(mem1_counter < 32)
                                begin
                                        next = MEM1WRITE;
                                end
                                else if(mem1_counter == 32)
                                begin
                                        //next = WAIT3;
                                        next = WAIT4;
                                end
                        end
        //Insert a WAIT state for safety's sake - pull CSn high for long enough so that
        //junky values of SI not pulled into SPI interface to CC2420
        WAIT3: begin
                        //next = MEM2WRITE;
                        next = MEM1WRITE;
                        end
        //Write the SHORT ADDRESS (device ID) of the chip to CC2420 RAM
        //31-bit command serialized to CC2420
        MEM2WRITE: begin
                                if(mem2_counter < 32)
                                begin
                                        next = MEM2WRITE;
                                end
                                else if(mem2_counter == 32)
                                begin
                                        //next = WAIT4;
                                        next = WAIT3;
                                end
                        end
        //Insert a WAIT state for safety's sake - pull CSn high for long enough so that
        //junky values of SI not pulled into SPI interface to CC2420
        WAIT4: begin
                        //next = ASKREAD1;
                        next = ASKREAD2;
                        end
        //Do a check state - read from memory location containing PANID
        ASKREAD1: begin
                                if(read1_counter < 32)
                                begin
                                        next = ASKREAD1;
                                end
                                else if(read1_counter==32)
                                begin
                                        //next = WAIT5;
                                        next = IDLE;
                                end
                end
        //Insert a WAIT state for safety's sake - pull CSn high for long enough so that
        //junky values of SI not pulled into SPI interface to CC2420
        WAIT5: begin
                        //next = ASKREAD2;
                        next = ASKREAD1;
                        end
        //Do a check state - read from memory location containing SHORTADDR
        ASKREAD2: begin
                                if(read2_counter < 32)
                                begin
                                        next = ASKREAD2;
                                end
                                else if(read2_counter==32)
                                begin
                                        //next = IDLE;
                                        next = WAIT5;
                                end
                end
        //In IDLE state, configuration complete! done_config
        //gets pulled high. Once idle, will stay here forever
        //until next system reset.
        IDLE: begin
                        next = IDLE;
                        end
endcase
end

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Nivedita Chandrasekaran
// Create Date:    20:21:53 05/07/2007
// Module Name:    TransmitAgressiveFSM
// Description: This module is responsible for handling the transmission of 40-bit (or 5-byte)
packets of data.
// While this module is currently hardwired to handle the 5-byte packets of data, it can be
easily modified to
//handle larger amounts of data by simply (1) changing the allowed size of the PAYLOAD input and
(2) changing
//the counters in the WRITEPACKET stage. This module also waits for an acknowledgement(ACK)
frame from the receiver it
//targets. If the TransmitAggressiveFSM does not receive an ACK frame, the FSM ensures that it
will retransmit the packet
//until an ACK frame is received. If the ACK frame is received, TransmitAggressiveFSM will call
a minorFSM called CheckAck responsible
//for parsing ACK frames in order to make sure that the ACK frame
//This module does successfully transmit packets and retransmits packets when no ACK frame is
received. However, after debugging
//it is doubtful that the WRITEPACKET stage is correctly writing to the
//This module takes in inputs from the CC2420: FIFOP_PIN,SFD_PIN, SO
//It also takes in inputs from the user: LENGTH(length of payload + 11 bytes of overhead),
PANID, DESTADDR (receiver to transmit to)
//, MYADDR(shortaddress of transmitter), PAYLOAD (payload to transmit) and outputs SI, CSn,
SCLK, to chip.It also outputs the state it is in for
//debugging purposes.
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////
module TransmitAggressiveFSM(clock_in, reset_in, begin_trans, done_trans,
                        FIFOP_PIN, SFD_PIN, LENGTH, PANID, DESTADDR, MYADDR, PAYLOAD, SI, CSn,
SCLK, SO, state);
    input clock_in; //module clocked at 10MHz or less.
         wire clock_in;
         input reset_in; //system reset
         wire reset_in;
         input begin_trans; //enable signal to this FSM
         input FIFOP_PIN, SFD_PIN;// signals from CC2420 to modules
         input[6:0] LENGTH;
         wire[6:0] LENGTH;
         input[15:0] PANID;
         wire[15:0] PANID;
         input[15:0] DESTADDR;
         wire[15:0] DESTADDR;
         input[15:0] MYADDR;
         wire[15:0] MYADDR;
         input[39:0] PAYLOAD; //5 BYTES
         wire[39:0] PAYLOAD; //Packet payload
         input SO;                              //data from chip
         output CSn;                    //chip enable (ACTIVE LOW)
         output SI;                     //chip input (1 bit)
         output SCLK;                   //chip clock, flipped version of normal clock
                                                                  //want to make sure that SI
straddles rising edge of SCLK
         wire SO;

         //Transmit FSM's outputs to CC2420
         reg CSn_trans;                                 //chip enable (ACTIVE LOW)
         reg SI_trans;                                     //chip input (1 bit)
         wire SCLK_trans = ~clock_in;                      //chip clock, flipped version of this
module's clock
        //want to make sure that SI straddles rising edge of SCLK
    output done_trans; //sends out the "transmission completed signal"
         reg done_trans;

        //FSM states
        parameter IDLE = 0;
        parameter WAITBUSY = 1;
        parameter RECMODEON = 2;
        parameter STROBETX = 3;
        parameter WRITEPACKET = 4;
        parameter WAITACK = 5;
        parameter CHECKACK = 6;
        parameter RETRANS = 7;
        //state registers
        output [2:0] state;
        reg[2:0] state, next;


        //timers for each CC2420reg-writing state
        reg[5:0] rec_counter;
        reg[5:0] tx_counter;
        reg[7:0] write_counter;
        reg[12:0] ack_counter;
        reg[3:0] retx_counter;


        //TXFIFO FLUSH AND RECMODE ON REGISTER
        reg[15:0] STROBE_FLUSH_RECON;
        //STROBE STXON - turns on transmit mode REGARDLESS
        //of whether or not channel is clear
        reg[15:0] STROBE_STXON;
        //WRITE TO TXFIFO COMMAND
        //reg[7:0] WRITE_TXFIFO;
        //REG value = request ACK back
        reg[15:0] REQUEST_ACK;
        //construct packet to send
        reg[119:0] trans_packet;
        //REG value - request resend
        reg[7:0] STROBE_SReTXON;

        //reg to hold acknowledge on/off
        reg begin_ack, begin_ack_next;
        wire done_ack;
        //reg to hold DATA PACKET NUMBER
        reg[7:0] DATA_SEQ_NO;
        //output from Check_ACK module
        //reg SI_ack, SCLK_ack, CSn_ack, done_ack;
        wire SI_ack, SCLK_ack, CSn_ack;
        wire success;
        wire[6:0] LENGTH_ACK;
        wire[7:0] DATA_SEQ_NO_ACK, RSSI_ACK, CRC_BYTE_ACK;
        wire[15:0] FCF_ACK;
        Check_ACK checkACK (
                .reset(reset_in),
                .clock(clock_in),
                .begin_ACK(begin_ack),
                .done_ACK(done_ack),
                .success(success),
                .SI(SI_ack),
                .CSn(CSn_ack),
                .SO(SO),
                .SCLK(SCLK_ack),
                .DATA_SEQ_NO_IN(DATA_SEQ_NO),
                .LENGTH(LENGTH_ACK),
                .FCF(FCF_ACK),
                .DATA_SEQ_NO(DATA_SEQ_NO_ACK),
                .RSSI(RSSI_ACK),
                .CRC_BYTE(CRC_BYTE_ACK)
        );

        //declare when SI/CSn/SCLK outputs come from transFSM and when they come from
        //Check_Status module
        assign SI = begin_ack ? SI_ack : SI_trans;
        assign CSn = begin_ack ? CSn_ack : CSn_trans;
        assign SCLK = begin_ack ? SCLK_ack : SCLK_trans;


always @ (posedge clock_in)
begin
        //initialize counters
                if(reset_in)
                begin
                        rec_counter <= 0;
                        write_counter <= 0;
                        tx_counter <= 0;
                        ack_counter <= 0;
                        retx_counter <=0;
                        //initialize FSM
                        state <= WAITBUSY;
                        //initialize ouputs
                        done_trans<= 0;
                        CSn_trans <= 1;
                        SI_trans <= 0;
                        begin_ack <= 0;
                        //TXFIFO FLUSH AND RECMODE ON REGISTER
                        STROBE_FLUSH_RECON <= 16'h0903;
                        //STROBE STXON - turns on transmit mode REGARDLESS
                        //of whether or not channel is clear
                        STROBE_STXON <= 16'h043E;
                        STROBE_SReTXON <= 8'h04;
                        //WRITE TO TXFIFO COMMAND
                        //WRITE_TXFIFO <= 8'h3E;
                        //REG value = request ACK back
                        REQUEST_ACK <= 16'h8861;
                        //reset value of DATA_SEQ_NO
                        DATA_SEQ_NO <= 0;
                        //initialize packet - {Length, FrameControlSequence, SequenceNumber (set
to 0),
                        //                                                              Personal
Area Network ID, Destination Address,
                        //                                                              My
Address, Payload}
                        // Length includes everything after it AS WELL AS the 2 error check and
crc bytes
                        //For first-case testing purposes, the payload is currently 40 bits (5
bytes)long.
                        //In the actual project, this will be reset to 800 bits (100 bytes)
                        //LENGTH: (w/5bytes) is 16 - 7'b0010000
                        //LENGTH:(W/100bytes) is 111 - 7'b1101111
                        trans_packet <= {1'b0, LENGTH[6:0], REQUEST_ACK[15:0], 8'h00,
PANID[15:0], DESTADDR[15:0], MYADDR[15:0], PAYLOAD[39:0]};
                        //WATCHOUT
                end
                else if(begin_trans)
                begin
                        //how to increment state counters
                        rec_counter <=  ((state==RECMODEON) ? (rec_counter + 1) : 0);
                        tx_counter <=           ((state==STROBETX) ? (tx_counter + 1) : 0);
                        write_counter <=        ((state==WRITEPACKET) ? (write_counter + 1) :
0);
                        ack_counter <=  ((state==WAITACK) ? (ack_counter + 1) : 0);
                        retx_counter <=         ((state==RETRANS) ? (retx_counter + 1) : 0);
                        //CSn is tricky, since CC2420 takes in SI on rising edge of SCLK
                        CSn_trans <=                            (!(
((state==RECMODEON)&&(rec_counter<16)) ||

((state==STROBETX)&&(tx_counter<17)) ||

((state==WRITEPACKET)&&(write_counter<120)) ||

((state==RETRANS)&&(retx_counter<8))
                                                                ));
                        //DATA sequence number gets incremented at the end of every WRITEPACKET
state
                        //we will let it overflow
                        DATA_SEQ_NO <= (((state==WRITEPACKET) && (write_counter==120)) ?
(DATA_SEQ_NO + 1) : DATA_SEQ_NO);
                        //deals with shifting out bits in configuration registers
                        STROBE_FLUSH_RECON  <= ((state==RECMODEON) ? ({STROBE_FLUSH_RECON[14:0],
1'b0}) : (16'h0903));
                        STROBE_STXON              <= ((state==STROBETX) ? ({STROBE_STXON[14:0],
1'b0}) : (16'h043E));
                        trans_packet              <= ((state==WRITEPACKET) ?

({trans_packet[118:0], 1'b0}) :

({1'b0, LENGTH[6:0], REQUEST_ACK[15:0], 8'h00, PANID[15:0], DESTADDR[15:0], MYADDR[15:0],
PAYLOAD[39:0]}));
                        //Retransmit command gets refreshed every time RETRANS state is exited
out of
                        STROBE_SReTXON          <= ((state==RETRANS) ? ({STROBE_SReTXON[6:0],
1'b0}) : (8'h04));

                        //done_config high when last state reached
                        done_trans <=   (state==IDLE);
                        //update state on every 10Mhz clock cycle
                        state <= next;
                        begin_ack <=  begin_ack_next;
                end
                //logic for SI values
                case(state)
                        IDLE: begin
                                        SI_trans <= 0;
                                end
                        RECMODEON: begin
                                                SI_trans <= STROBE_FLUSH_RECON[15];
                                        end
                        STROBETX: begin
                                                SI_trans <= STROBE_STXON[15];
                                        end
                        WRITEPACKET: begin
                                                SI_trans <= trans_packet[119];
                                        end
                        RETRANS: begin
                                                SI_trans <= STROBE_SReTXON[7];
                                        end
                endcase
end

always @ (state or rec_counter or done_ack or FIFOP_PIN or SFD_PIN or
                        tx_counter or write_counter or ack_counter or retx_counter or
                        begin_ack or success or begin_trans)
begin
        begin_ack_next = begin_ack;
        next = state;
        case (state)
        //WAITBUSY is the initial state. This part of the transmit operation
        //checks to see whether a transmit operation is in progress - if one is in progress,
        //it will continue to WAITBUSY. Otherwise, the operation will move to the RECMODEON
state.
        WAITBUSY: begin
                                if((FIFOP_PIN==0) && (SFD_PIN==0)) //chip is not receiving
(FIFOP_PIN==0) and
                                begin
//is not writing to the TXFIFO (SFD_PIN==0)
                                        next = RECMODEON;
                                end
                                else
                                begin
                                        next = WAITBUSY; //chip is busy, keep waiting
                                end
                        end
        //RECMODEON strobes two command registers in succession. The first strobe
        //flushes the TXFIFO, and the second turns on the receive mode so that
        //the transmitter will be able to receive acknowledgement signals from the receiver.
        //Once this is done, the FSM moves to the STROBETX state.
        RECMODEON: begin
                                if(rec_counter < 16)
                                begin
                                        next = RECMODEON;
                                end
                                else if(rec_counter == 16)
                                begin
                                        next = STROBETX;
                                end
                        end
        //STROBETX strobes the STXON, which turns on transmit mode
        //and also strobes the WRITE_TXFIFO register, which will allow us to
        //write the packet to the TXFIFO in the WRITEPACKET stage
        STROBETX: begin
                                if(tx_counter < 16)
                                begin
                                        next = STROBETX;
                                end
                                else if(tx_counter == 16)
                                begin
                                        next = WRITEPACKET;
                                end
                        end
        //WRITEPACKET writes the packet to the CC2420 TXFIFO
        //after the packet has been constructed, the wireless chip
        //should send out the packet automatically once it hits the specified length.
        //After packet construction, the FSM will enter the WAITACK state.
        WRITEPACKET: begin
                                if(write_counter < 120) //WATCHOUT - 40BIT PAYLOAD HARDCODED
HERE
                                begin
// here, we loop through the trans_packet and write
                                        next = WRITEPACKET;             //each byte to the
TXFIFO
                                end
                                else if(write_counter == 120)
                                begin
                                        next = WAITACK; //done writing packet to TXFIFO, move to
WAITACK state
                                end
                        end
        //WAITACK is a wait state - the FSM sits here until
        //1. The waiting period for an acknowledge frame is up, in which case it moves to
RETRANS
        //2. The waiting period is not complete BUT an acknowledge has been received - then move
        //        move to CHECKACK
        WAITACK: begin          if((ack_counter<6760) && !FIFOP_PIN)
                                begin
                                        next = WAITACK;
                                end
                                else if((ack_counter<6760) && FIFOP_PIN)
                                begin
                                        next = CHECKACK;
                                        begin_ack_next = 1;
                                end
                                else if(ack_counter==6760)
                                begin
                                        next = RETRANS;
                                end
                        end
        //RETRANS simply asks the transmitter to retransmit the previous packet.
        //This is done by restrobing the CC2420 STXON strobe w/o writing again
        //to the TXFIFO.
        //When RETRANS is done, move to WAITACK state
        RETRANS: begin
                                if(retx_counter < 8)
                                begin
                                        next = RETRANS;
                                end
                                else if(retx_counter == 8)
                                begin
                                        next = WAITACK;
                                end
                        end
        //The CHECKACK state starts the Check_ACK module, which reads an ack frame
        //out of the RXFIFO of the transmitter and checks to see whether or not it is the
        //correct ack frame
        //1. If it is, success is HIGH, and we transition to an IDLE state
        //2. Otherwise, we move to the RETRANS state.
        CHECKACK: begin
                                if(done_ack==0) //ack not completed, keep begin_ack high
                                begin                                   //so that Check_ACK
module can continue running
                                        begin_ack_next = 1;
                                        next = CHECKACK;
                                end
                                else if((done_ack==1) && !(success)) //packet NOT successfully
transmitted
                                begin
                                        next = RETRANS; //move to RETRANS state
                                        begin_ack_next = 0; //turn off Check_ACK enable signal
                                end
                        else if((done_ack==1) && (success)) //packet successfully transmitted!
                                begin
                                        next = IDLE; //Move on to IDLE state
                                        begin_ack_next = 0;     //turn off Check_ACK enable
signal
                                end
                        end
        //In IDLE state, transmit complete! done_transmit
        //gets pulled high. Once transmitted, will stay here until
        //next begin_trans command
        IDLE: begin
                                if(!begin_trans)
                                begin
                                        next = IDLE;
                                end
                                else if(begin_trans)
                                begin
                                        next = WAITBUSY;
                                end
                        end
        endcase
end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Nivedita Chandraskaran
// Create Date:    23:18:00 05/06/2007
// Module Name:    Check_ACK
// Description: This module is responsible for accessing the CC2420 wireless chip's RXFIFO
//and checking if a received packet is a properly formatted acknowledge frame. If the
//packet is an acknowledge frame, this module returns a success=1 - If not, success=0.
//For debugging purposes, Check_ACK also returns the contents of the acknowledge frame.
//////////////////////////////////////////////////////////////////////////////////
module Check_ACK(reset, clock, begin_ACK, done_ACK, success, SI, CSn, SO, SCLK,
                                                DATA_SEQ_NO_IN, LENGTH, FCF, DATA_SEQ_NO, RSSI,
CRC_BYTE); //bottom outputs are acknowledge packet contents
    input begin_ACK;
         wire begin_ACK;
         input clock, reset;
    output done_ACK;
         reg done_ACK;
         reg done_ACK_next;
    output reg success;
    output SI;
         output SCLK;
    output CSn;
         reg CSn, SI;
    input SO;
         wire SCLK = ~clock; //want to make sure that SI straddles rising edge of SCLK
        input[7:0] DATA_SEQ_NO_IN;
        wire[7:0] DATA_SEQ_NO_IN;
        output[6:0] LENGTH;
        output[15:0] FCF;
        output[7:0] DATA_SEQ_NO;
        output[7:0] RSSI;
        output[7:0] CRC_BYTE;

        //FSM states
        parameter IDLE = 0;
        parameter CHECKACK = 1;
        //state registers
        reg state, next;

        //timers for each state
        reg[5:0] ack_counter;

        //COMMAND STROBE - RXFIFO - get access
        reg[7:0] STROBE_RXFIFO;

        //regs from ACK packet that need to be checked
        reg[6:0] LENGTH;
        reg[15:0] FCF;
        reg[7:0] DATA_SEQ_NO;
        reg[7:0] RSSI;
        reg[7:0] CRC_BYTE;

always@(posedge clock)
begin
        //if a reset, reconfigure the chip
                if(reset)
                begin
                        //initialize counters
                        ack_counter <= 0;
                        //initialize FSM
                        state <= IDLE;
                        //initialize ouputs
                        done_ACK <= 0;
                        success <= 0;
                        CSn <= 1;
                        SI <= 0;
                        //initialize all outputs to 0
                        LENGTH<=0;
                        FCF<=0;
                        DATA_SEQ_NO<=0;
                        RSSI<=0;
                        CRC_BYTE<=0;
                        //set config register that will let us read from RXFIFO
                        STROBE_RXFIFO <= 8'h7F;
                end
                else if (begin_ACK)
                begin
```

```
                        //how to increment state counters
                        ack_counter <= ((next==IDLE) ? 0 : ((state==CHECKACK) ? (ack_counter + 
1) : 0));
                        //CSn is tricky, since CC2420 takes in SI on rising edge of SCLK
                        //as a result, CSn is as follows:
                        CSn <= (!((state==CHECKACK)&&(ack_counter<56)));
                        //deals with shifting out bits in config register
                        STROBE_RXFIFO    <= ((state==CHECKACK) ? ({STROBE_RXFIFO[6:0], 1'b0}) : 
(8'h7F));

                        //capture SO bits in data capture registers at appropriate times in read 
state
                        //regs from ACK packet that need to be checked
                        LENGTH          <= 
(((state==CHECKACK)&&(ack_counter>7)&&(ack_counter<16)) ? 
                                                                                            ({LENGTH
[6:0], SO}) : LENGTH);
                        FCF                     <= 
(((state==CHECKACK)&&(ack_counter>15)&&(ack_counter<32)) ? 
                                                                                            ({FCF[15
:0], SO}) : FCF);
                        DATA_SEQ_NO <= 
(((state==CHECKACK)&&(ack_counter>31)&&(ack_counter<40)) ? 
                                                                                            ({DATA_S
EQ_NO[7:0], SO}) : DATA_SEQ_NO);
                        RSSI                    <= 
(((state==CHECKACK)&&(ack_counter>39)&&(ack_counter<48)) ? 
                                                                                            ({RSSI[7
:0], SO}) : RSSI);
                        CRC_BYTE        <= 
(((state==CHECKACK)&&(ack_counter>47)&&(ack_counter<56)) ? 
                                                                                            ({CRC_BY
TE[7:0], SO}) : CRC_BYTE);
                        //calculate result
                        success <= ((LENGTH==5) && (DATA_SEQ_NO_IN==DATA_SEQ_NO) && 
(FCF==16'h0002) && (CRC_BYTE[7]));

                        //update state on every 10Mhz clock cycle
                        state <= next;
                        done_ACK <= done_ACK_next;

                        //take care of SI logic for CHECKACK state
                        case(state)
                                CHECKACK:  begin
                                                SI <= STROBE_RXFIFO[7];
                                        end
                        endcase
        end
end

always @ (ack_counter or begin_ACK or state)
begin
case (state)
        //IDLE is simply a wait state the FSM sits in when it is not busy.
        IDLE: begin
                                //initialize packet outputs to 0
                                if(begin_ACK==1)
                                begin
                                        next = CHECKACK; //if check demanded, move to CHECKACK
```

```
state
                                        done_ACK_next = 0;
                                end
                                else
                                begin
                                        next = IDLE;    //if no status check demanded, stay in
IDLE
                                        done_ACK_next = 0;
                                end
                        end
        //CHECKACK is the operating state - here, the module strobes the RXFIFO command register
        //on the chip, and collects
        // 1. The length of the acknowledge packet --> proper length?
        // 2. The frame check field (FCF) --> correct FCF?
        // 3. The data sequence # --> equal to 0? (since packets always constructed with 0 as
txSeqNumber)
        // 4. FCS[7] --> CRC correct value?
        //When the operation is complete, the FSM puts out a high done_ACK and returns to
        //the IDLE state.
        CHECKACK: begin
                                if(ack_counter < 56)    //strobe RXFIFO to get access
                                begin
                                        next = CHECKACK;
                                        done_ACK_next = 0;
                                end
                                else if(ack_counter == 56) //done, move to IDLE state
                                begin
                                        next = IDLE;
                                        done_ACK_next = 1;
                                end
                        end
        endcase
end
endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Nivedita Chandrasekaran
// Create Date:    16:28:24 04/29/2007
// Module Name:    ReceiveFSM
// Description: This module is the control module for the wireless CC2420 receiver. It will
// configure the CC2420 chip for reception, wait for a packet to be received, and will then
// parse the packet and return a result such that a packet with a passing CRC will be labeled
// as such and so that a packet with a bad CRC will be labeled as such.
////////////////////////////////////////////////////////////////////////////////
module ReceiveFSM(clock, reset, FIFOP_PIN, FIFO_PIN, LENGTH, FCF,
                        PANID, DATA_SEQ_NO, DESTADDR, MYADDR, PAYLOAD, CRC_GOOD, RSSI,
                        CRC_BYTE, SI, CSn, SCLK, SO, begin_rec, done_rec, state);
         input clock, reset, begin_rec;
    output reg CRC_GOOD;
         input FIFOP_PIN, FIFO_PIN;
    output SI;
         output SCLK;
    output CSn;
         reg CSn, SI;
    input SO;
         wire SCLK = ~clock; //want to make sure that SI straddles rising edge of SCLK
         output done_rec;
         reg done_rec;
        output[7:0] LENGTH;
        output[15:0] FCF;
        output[7:0] DATA_SEQ_NO;
        output[15:0] PANID;
        output[15:0] DESTADDR;
        output[15:0] MYADDR;
        output[39:0] PAYLOAD; //HARDCODED WATCHOUT
        output[7:0] RSSI;
        output[7:0] CRC_BYTE;
        output[2:0] state;

        //FSM states for receive
        parameter RECON = 0;
        parameter WAIT = 1;
        parameter PARSEPACKET = 2;
        parameter FLUSH = 3;
        parameter DONE = 4;
        //state registers
        reg[2:0] state, next;

        //timers for each CC2420reg-writing state
        reg[4:0] recon_counter;
        reg[8:0] p_counter;
        reg[4:0] f_counter;

        //COMMAND STROBE - turn receive on, flush RXFIFO
        reg[15:0] STROBE_RXONCLR;
        //COMMAND STROBE - RX UNDERFLOW - flush RXFIFO twice!
        reg[15:0] STROBE_FLUSHRX;
        //COMMAND STROBE - RXFIFO - get access
        reg[7:0] STROBE_RXFIFO;

        //regs from RECEIVED packet that need to be checked
        reg[7:0] LENGTH;
        reg[15:0] FCF;
        reg[7:0] DATA_SEQ_NO;
        reg[15:0] PANID;
        reg[15:0] DESTADDR;
        reg[15:0] MYADDR;
        reg[39:0] PAYLOAD; //HARDCODED WATCHOUT
        reg[7:0] RSSI;
        reg[7:0] CRC_BYTE;

always@(posedge clock)
begin
        begin
        //if a reset, reconfigure the chip
                if(reset)
                begin
                        //initialize counters
                        recon_counter <= 0;
                        p_counter <= 0;
                        f_counter <= 0;
                        //initialize FSM
                        state <= RECON;
                        //initialize ouputs
                        CRC_GOOD <= 0;
                        CSn <= 1;
                        SI <= 0;
                        LENGTH <= 0;
                        FCF <= 0;
                        DATA_SEQ_NO <= 0;
                        PANID <= 0;
                        DESTADDR <= 0;
                        MYADDR <= 0;
                        PAYLOAD <= 0; //HARDCODED WATCHOUT
                        RSSI <= 0;
                        CRC_BYTE <= 0;
                        //initialize config registers
                        //COMMAND STROBE - turn receive on, flush RXFIFO
                        STROBE_RXONCLR <= 16'h0308;
                        //COMMAND STROBE - RX UNDERFLOW - flush RXFIFO twice!
                        STROBE_FLUSHRX <= 16'h0808;
                        //COMMAND STROBE - RXFIFO - get access
                        STROBE_RXFIFO <= 8'h7F;
                        done_rec <= (state==WAIT);
                end
                else if(begin_rec)
                begin
                        //how to increment state counters
                        recon_counter <= ((state==RECON) ? (recon_counter + 1) : 0);
                        p_counter <= ((state==PARSEPACKET) ? (p_counter + 1) : 0);
                        f_counter <= ((state==FLUSH) ? (f_counter + 1) : 0);
                        //CSn is tricky, since CC2420 takes in SI on rising edge of SCLK
                        //as a result, CSn is as follows:
                        CSn <=          (!( ((state==RECON)&&(recon_counter<16)) ||
                                                                   ((state==PARSEPACKET)&&(p_count
er<144)) || //WATCHOUT - HARDCODED HERE
                                                                   ((state==FLUSH)&&(f_counter<16)
)
                                                                 ));
                        //deals with shifting out bits in various command registers
                        STROBE_RXONCLR   <= ((state==RECON) ? ({STROBE_RXONCLR[14:0], 1'b0}) :
(16'h0308));
                        STROBE_FLUSHRX   <= ((state==FLUSH) ? ({STROBE_FLUSHRX[14:0], 1'b0}) :
(16'h0808));
                        STROBE_RXFIFO            <= ((state==PARSEPACKET) ?
({STROBE_RXFIFO[6:0], 1'b0}) : (8'h7F));

                        //capture SO bits in data capture registers at appropriate times in read
state
                        //regs from RECEIVED packet that need to be checked
                         LENGTH <= (((state==PARSEPACKET)&&(p_counter>7)&&(p_counter<16)) ?
                                                                                        ({LENGTH
[6:0], SO}) : LENGTH);
                    FCF           <= (((state==PARSEPACKET)&&(p_counter>15)&&(p_counter<32)) ?
                                                                                        ({FCF[14
:0], SO}) : FCF);
                         DATA_SEQ_NO <=
(((state==PARSEPACKET)&&(p_counter>31)&&(p_counter<40)) ?
                                                                                        ({DATA_S
EQ_NO[6:0], SO}) : DATA_SEQ_NO);
                    PANID                <=
(((state==PARSEPACKET)&&(p_counter>39)&&(p_counter<56)) ?
                                                                                        ({PANID[
14:0], SO}) : PANID);
                    DESTADDR     <= (((state==PARSEPACKET)&&(p_counter>55)&&(p_counter<72)) ?
                                                                                        ({DESTAD
DR[14:0], SO}) : DESTADDR);
                    MYADDR               <=
(((state==PARSEPACKET)&&(p_counter>71)&&(p_counter<88)) ?
                                                                                              ({MYADDR
[14:0], SO}) : MYADDR);
                    PAYLOAD      <= (((state==PARSEPACKET)&&(p_counter>87)&&(p_counter<128)) ?
                                                                                              ({PAYLOA
D[38:0], SO}) : PAYLOAD);
                         RSSI       <= (((state==PARSEPACKET)&&(p_counter>127)&&(p_counter<136))
?
                                                                                              ({RSSI[6
:0], SO}) : RSSI);
                         CRC_BYTE        <=
(((state==PARSEPACKET)&&(p_counter>135)&&(p_counter<144)) ?
                                                                                              ({CRC_BY
TE[6:0], SO}) : CRC_BYTE);

                        case(state)
                         RECON: begin
                                        SI <= STROBE_RXONCLR[15];
                                end
                         PARSEPACKET: begin
                                        SI <= STROBE_RXFIFO[7];
                                end
                         FLUSH: begin
                                        SI <= STROBE_FLUSHRX[15];
                                end
                        endcase
                        //CRC_GOOD updated on every clock cycle
                        CRC_GOOD <= ((state==DONE)&&(CRC_BYTE[7]));
                        //update state on every 10Mhz clock cycle
                        state <= next;
                end
        end
end

always @ (state or recon_counter or p_counter or FIFO_PIN or FIFOP_PIN or f_counter)
begin
        next = state;
        case (state)
                //on a reset, we enter a RECON state, which states that
                RECON: begin
                                if(recon_counter < 16)
                                begin
                                        next = RECON;
                                end
                                else if(recon_counter == 16)
                                begin
                                        next = WAIT;
                                end
                        end
                //In this state, we wait for an interrupt signalling that
                //a packet has been received
                WAIT: begin
                                //If the FIFOP_PIN is high (signalling that RXFIFO is filled
above threshold)
                                //and FIFO_PIN low (saying that no data in FIFO), then underflow
--> move
                                //to FLUSH state
                                if((FIFOP_PIN==1) && (FIFO_PIN==0))
                                begin
                                        next = FLUSH;
                                end
                                //In normal case of packet reception, move to PARSEPACKET state
                                else if((FIFOP_PIN==1) && (FIFO_PIN==1))
                                //else if((FIFO_PIN==1))
                                begin
                                        next = PARSEPACKET;
                                end
                                else
                                //otherwise, just wait for packet
                                begin
                                        next = WAIT;
                                end
                        end
                //PARSEPACKET state - Reads in a packet from the CC2420 wireless chip
                //and parses it to return all possible information about the packet.
                PARSEPACKET: begin
                                if(p_counter < 144)     //strobe RXFIFO to get access
                                begin
                                        next = PARSEPACKET;
                                end
                                else if(p_counter == 144) //done, move to IDLE state
                                begin
                                        next = DONE;
                                end
                        end
                //FLUSH state will be entered if RXFIFO underflow directed
                //Clears RXFIFO twice by inputting 2 command strobes to CC2420 FLUSHRX reg
                FLUSH: begin
                                if(f_counter < 16)
                                begin
                                        next = FLUSH;
                                end
                                else if(f_counter == 16)
                                begin
                                        next = WAIT;
                                end
                        end
                DONE: begin //can only get to this state after a full PARSEPACKET - serves as a
check for CRC_GOOD
                                next = WAIT;
                        end
        endcase
end
endmodule

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Engineer: Nivedita Chandrasekaran
// Create Date:    23:20:16 05/13/2007
// Module Name:    square10
// Description: This module generates a 6.75Mhz clock for the wireless FSMs.
////////////////////////////////////////////////////////////////////////////////
module square10(reset, input_clock, out_clock);
    input reset;
         input input_clock;
    output wire out_clock;
```

```
	reg[1:0] count;
	always @ (posedge input_clock)
	begin
		if(reset)
		begin
			count <= 0;
		end
		count <= (count+1);
	end

	assign out_clock = count[1];

endmodule

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Transmit Side of Group1 Final Project
//
//
// Created: March 15, 2007
// Author: Nathan Ickes, Wireless by Nivedita Chandrasekaran
//Description: This module is the top-level code for the wireless transmit side of the
// Group1 Final Project.
///////////////////////////////////////////////////////////////////////////////

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_synch, 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_mpbrdy;

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;
*/
   // Buttons, Switches, and Individual LEDs
 //  assign led = 8'hFF;

   // User I/Os
   assign user1 = 32'hZ;
//   assign user2 = 32'hZ;
//   assign user3 = 32'hZ;
  assign user4 = 32'hZ;

   // 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;

   // Logic Analyzer
   //assign analyzer1_data = 16'h0;
  // assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   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;

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

   //
   // Generate a 31.5MHz pixel clock from clock_27mhz
   //

   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // 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;

//*********************WIRELESS STARTS HERE*********************//

wire clock_27mhz;
//debounce reset signal
wire reset;
```

```
assign reset = ~button_enter;
reg Reset_sync;
reg q1;

        wire wireless_clock;
        //generates 6.75Mhz signal from 27Mhz signal for wirelessFSM clocks
        square10 clocker (
                .reset(~button0),
                .input_clock(clock_27mhz),
                .out_clock(wireless_clock)
        );



        //delay asynchronous original reset signal with two registers
        always @ (posedge wireless_clock)
        begin
                        q1 <= reset;
                        Reset_sync <= q1;
        end

wire done_config, done_trans, done_rec;
wire begin_config, begin_rec, begin_trans;
wire[1:0] state_MASTER;

wire SI, CSn, SCLK, reset_chipn, SO, FIFOP_PIN, SFD_PIN;
wire SI_cfg, CSn_cfg, SCLK_cfg;
wire SI_trans, CSn_trans, SCLK_trans;
wire[15:0] READPANID, READSHORTADDR;
wire configured;
//WIRELESS CONFIGURATION FSM HERE
//configured signal acts as enable to TransmitFSM
        ConfigureFSM configTEST (
                .clock(wireless_clock),
                .reset(Reset_sync),
                .reset_chipn(reset_chipn),
                .CHANNEL(11),
                .PANID(16'h6111),
                .SHORTADDR(16'h1234),
                .CSn(CSn_cfg),
                .SI(SI_cfg),
                .SCLK(SCLK_cfg),
                .SO(SO),
                .begin_config(0),
                .done_config(done_config),
                .state(state_cfg),
                .READPANID(READPANID),
                .READSHORTADDR(READSHORTADDR),
                .mem2_counter(counter),
                .configured(configured)
        );

//WIRELESS TRANSMIT FSM HERE
wire [2:0] state_trans;
        TransmitAggressiveFSM transmitFSM (
                .clock_in(wireless_clock),
                .reset_in(Reset_sync),
                .begin_trans(configured),
                .done_trans(done_trans),
                .FIFOP_PIN(FIFOP_PIN),
                .SFD_PIN(SFD_PIN),
                .LENGTH(7'b0010000),
                .PANID(16'h6111),
                .DESTADDR(16'h4567),
                .MYADDR(16'h1234),
                .PAYLOAD(40'h0123456789),
                .SI(SI_trans),
                .CSn(CSn_trans),
                .SCLK(SCLK_trans),
                .SO(SO),
                .state(state_trans)
        );

//for now, only have CFG and TRANS - switch SI/SO/SCLK between the two
//PROMISE: CSn held high in IDLE for cfg and trans, SI held low when not transmitting
assign SI = (SI_cfg || SI_trans);
assign SCLK = SCLK_cfg || SCLK_trans;
assign CSn = (CSn_cfg & CSn_trans);

        //USER 2 - wire connections to CC2420 on P4 port of CC2420DBK.
        assign user2[10:8] = state_trans[2:0];
        assign user2[7] = done_config;
        //input from CC2420
        assign FIFOP_PIN = user2[6];
        assign SFD_PIN = user2[5];
        assign SO = user2[4];
        assign user2[3] = SI;
        assign user2[2] = CSn;
        assign user2[1] = SCLK;
        assign user2[0] = reset_chipn;

        assign user2[30:11] = 20'hZ;
        assign user2[31] = 1'b1;
        //output


        assign led[7] = ~Reset_sync;
        assign led[6] = ~SFD_PIN;
        assign led[5] = ~FIFOP_PIN;
        assign led[4:2] = ~state_trans[2:0];
        assign led[1] = ~done_trans; //for now, should never go on since receiver not
functioning as expected
        assign led[0] = ~done_config;

        wire[79:0] dots1, dots2, dots3,dots4,dots5,dots6,dots7,dots8;

        //display to make sure that chip configured correctly
        dots DATA1(clock_27mhz, READPANID[3:0], dots1);
        dots DATA2(clock_27mhz, READPANID[7:4], dots2);
        dots DATA3(clock_27mhz, READPANID[11:8], dots3);
        dots DATA4(clock_27mhz, READPANID[15:12], dots4);
        dots DATA5(clock_27mhz, READSHORTADDR[3:0], dots5);
        dots DATA6(clock_27mhz, READSHORTADDR[7:4], dots6);
        dots DATA7(clock_27mhz, READSHORTADDR[11:8], dots7);
        dots DATA8(clock_27mhz, READSHORTADDR[15:12], dots8);

        display memdisplay(Reset_sync,
                                                        clock_27mhz,
                                                          disp_blank,
                                                          disp_clock,
                                                          disp_rs,
                                                          disp_ce_b,
                                                          disp_reset_b,
                                                          disp_data_out,
                                                          {dots2, dots1, dots4, dots3, dots6,
dots5, dots8, dots7});

   // Logic Analyzer
/*
 assign analyzer1_data = 16'h0;
  assign analyzer1_clock = 1'b1;
  assign analyzer3_data = 16'h0;
  assign analyzer3_clock = 1'b1;
*/
        assign analyzer1_data[15] = SI_trans;
        assign analyzer1_data[14] = SCLK_trans;
        assign analyzer1_data[13] = CSn_trans;
        assign analyzer1_data[12] = SI_cfg;
        assign analyzer1_data[11] = SCLK_cfg;
        assign analyzer1_data[10] = CSn_cfg;
        assign analyzer1_data[9] = configured;
        assign analyzer1_data[8:0] = 0;
        assign analyzer1_clock = clock_27mhz;

//      assign analyzer1_data[15:0] = {READSHORTADDR[7:0], READSHORTADDR[15:8]};
//      assign analyzer1_clock = clock_27mhz;
        assign analyzer3_data[15] = done_config;
        assign analyzer3_data[14] = done_trans;
        assign analyzer3_data[13] = FIFOP_PIN;
        assign analyzer3_data[12] = SFD_PIN;
        assign analyzer3_data[11] = SO;
        assign analyzer3_data[10] = SI;
        assign analyzer3_data[9] = CSn;
        assign analyzer3_data[8] = SCLK;
        assign analyzer3_data[7] = reset_chipn;
        assign analyzer3_data[6:4] = state_trans[2:0];
        assign analyzer3_data[3:0] = 0;
        assign analyzer3_clock = clock_27mhz;

endmodule

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Debounce/Synchronize module
//
//
// Use your system clock for the clock input to produce a synchronous,
// debounced output
//
///////////////////////////////////////////////////////////////////////////////

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

////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Number to bitmap decoder
//
//
// Author: Yun Wu, Nathan Ickes
// Date: March 8, 2006
//
// This module converts a 4-bit input to a 80-dot (2 digit) bitmap representing
// the numbers ' 0' through '15'.
//
////////////////////////////////////////////////////////////////////////////////

module dots(clk, num, dots);
   input clk;
   input [3:0] num;
   output [79:0] dots;

   reg [79:0] dots;
   always @ (posedge clk)
     case (num)
       4'd15: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '15'
                       40'b00100111_01000101_01000101_01000101_00111001};
       4'd14: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '14'
                       40'b00011000_00010100_00010010_01111111_00010000};
       4'd13: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '13'
                       40'b00100010_01000001_01001001_01001001_00110110};
       4'd12: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '12'
                       40'b01100010_01010001_01001001_01001001_01000110};
       4'd11: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '11'
                       40'b00000000_01000010_01111111_01000000_00000000};
       4'd10: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '10'
                       40'b00111110_01010001_01001001_01000101_00111110};
       4'd09: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 9'
                       40'b00000110_01001001_01001001_00101001_00011110};
       4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 8'
                       40'b00110110_01001001_01001001_01001001_00110110};
       4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 7'
```

```
                    40'b00000001_01110001_00001001_00000101_00000011};
       4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 6'
                    40'b00111100_01001010_01001001_01001001_00110000};
       4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 5'
                    40'b00100111_01000101_01000101_01000101_00111001};
       4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 4'
                    40'b00011000_00010100_00010010_01111111_00010000};
       4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 3'
                    40'b00100010_01000001_01001001_01001001_00110110};
       4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 2'
                    40'b01100010_01010001_01001001_01001001_01000110};
       4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 1'
                    40'b00000000_01000010_01111111_01000000_00000000};
       4'd00: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 0'
                    40'b00111110_01010001_01001001_01000101_00111110};
       // No default case, becase every case is already accounted for.
     endcase

        endmodule


///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// Change history
//
// 2007-02-09: Fixed register-select race condition. (Thanks to Chris
//             Buenrostro for finding this bug and David Wentzloff for
//             implementing the fix.)
// 2005-05-09: Made <dots> input registered, and converted the 640-input MUX
//             to a 640-bit shift register.
//
///////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out, dots);

   input reset; // Active high
   input clock_27mhz;
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
          disp_reset_b;
   input [639:0] dots;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
        if (reset)
          begin
             count = 0;
             clock = 0;
          end
        else if (count == 26)
          begin
             clock = ~clock;
             count = 5'h00;
          end
        else
          count = count+1;
     end

   always @(posedge clock_27mhz)
     if (reset)
       reset_count <= 100;
     else
       reset_count <= (reset_count==0) ? 0 : reset_count-1;

   assign dreset = (reset_count != 0);

   assign disp_clock = ~clock;

   //
   // Display State Machine
   //

   reg [7:0] state;
   reg [9:0] dot_index;
   reg [31:0] control;
   reg [639:0] ldots;

   assign disp_blank = 1'b0; // low = not blanked

   always @(posedge clock)
     if (dreset)
       begin
          state <= 0;
          dot_index <= 0;
          control <= 32'h7F7F7F7F;
       end
     else
       casex (state)
         8'h00:
           begin
              // Reset displays
              disp_data_out <= 1'b0;
              disp_rs <= 1'b0; // 0 = dot register
              disp_ce_b <= 1'b1;
              disp_reset_b <= 1'b0;
              dot_index <= 0;
              state <= state+1;
           end

         8'h01:
           begin
              // End reset
              disp_reset_b <= 1'b1;
              state <= state+1;
           end

         8'h02:
           begin
              // Initialize dot register
              disp_ce_b <= 1'b0;
              disp_data_out <= 1'b0; // dot_index[0];
              if (dot_index == 639)
                state <= state+1;
              else
                dot_index <= dot_index+1;
           end

         8'h03:
           begin
              // Latch dot data
              disp_rs <= 1'b1; // Select the control register
              disp_ce_b <= 1'b1;
              dot_index <= 31;
              state <= state+1;
           end

         8'h04:
           begin
              // Setup the control register
              disp_ce_b <= 1'b0;
              disp_data_out <= control[31];
              control <= {control[30:0], 1'b0};
              if (dot_index == 0)
                state <= state+1;
              else
                dot_index <= dot_index-1;
           end

         8'h05:
           begin
              // Latch the control register data
              disp_rs <= 1'b0; // Select the dot register
              disp_ce_b <= 1'b1;
              dot_index <= 639;
              ldots <= dots;
              state <= state+1;
           end

         8'h06:
           begin
              // Load the user's dot data into the dot register
              disp_ce_b <= 1'b0;
              disp_data_out <= ldots[639];
              ldots <= ldots<<1;
              if (dot_index == 0)
                state <= 5;
```

```
              else
                dot_index <= dot_index-1;
           end
       endcase

endmodule




///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module for Receive Side of Group1 Final Project
//
//
// Created: March 15, 2007
// Author: Nathan Ickes, Wireless by Nivedita Chandrasekaran
//Description: This module is the top-level code for the wireless receive side of the
// Group1 Final Project.
///////////////////////////////////////////////////////////////////////////////

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_synch, 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_mpbrdy;

   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;
*/
   // Buttons, Switches, and Individual LEDs
 //  assign led = 8'hFF;

   // User I/Os
   assign user1 = 32'hZ;
//   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
  assign user4 = 32'hZ;

   // 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;
   // Logic Analyzer
   //assign analyzer1_data = 16'h0;
  // assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   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;

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

   //
   // Generate a 31.5MHz pixel clock from clock_27mhz
   //

   wire pclk, pixel_clock;
   DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(pclk));
   // synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 6
   // synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 7
   // synthesis attribute CLK_FEEDBACK of pixel_clock_dcm  is "NONE"
   BUFG pixel_clock_buf (.I(pclk), .O(pixel_clock));

   //
   // 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;

//*********************WIRELESS STARTS HERE*********************//

wire clock_27mhz;
//debounce reset signal
wire reset;
assign reset = ~button_enter;
reg Reset_sync;
reg q1;

        wire wireless_clock;
        //generates 6.75Mhz signal from 27Mhz signal
        square10 clocker (
                .reset(~button0),
                .input_clock(clock_27mhz),
                .out_clock(wireless_clock)
        );



        //delay asynchronous original reset signal with two registers
        always @ (posedge wireless_clock)
        begin
                        q1 <= reset;
                        Reset_sync <= q1;
        end

wire done_config, done_trans, done_rec;
wire begin_config, begin_rec, begin_trans;
wire[1:0] state_MASTER;

wire SI, CSn, SCLK, reset_chipn, SO, FIFOP_PIN, FIFO_PIN, SFD_PIN;
wire SI_cfg, CSn_cfg, SCLK_cfg;
wire SI_rec, CSn_rec, SCLK_rec;
wire[15:0] READPANID, READSHORTADDR;
wire configured;
//WIRELESS CONFIGURATION FSM HERE
        ConfigureFSM configTEST (
                .clock(wireless_clock),
                .reset(Reset_sync),
                .reset_chipn(reset_chipn),
                .CHANNEL(11),
                .PANID(16'h6111),
                .SHORTADDR(16'h4567),
                .CSn(CSn_cfg),
                .SI(SI_cfg),
                .SCLK(SCLK_cfg),
                .SO(SO),
                .begin_config(0),
                .done_config(done_config),
                .state(state_cfg),
                .READPANID(READPANID),
                .READSHORTADDR(READSHORTADDR),
                .mem2_counter(counter),
                .configured(configured)
        );

wire[15:0] FCF;
wire[7:0] LENGTH, PANID, DESTADDR, DATA_SEQ_NO, MYADDR, RSSI, CRC_BYTE;
wire[39:0] PAYLOAD;
wire CRC_GOOD;
//WIRELESS RECEIVE FSM HERE
wire [2:0] state_rec;
        ReceiveFSM receiveFSM (
                .clock(wireless_clock),
                .reset(Reset_sync),
                .FIFOP_PIN(FIFOP_PIN),
                .FIFO_PIN(FIFO_PIN),
                .LENGTH(LENGTH),
                .FCF(FCF),
                .PANID(PANID),
                .DATA_SEQ_NO(DATA_SEQ_NO),
                .DESTADDR(DESTADDR),
                .MYADDR(MYADDR),
                .PAYLOAD(PAYLOAD),
                .CRC_GOOD(CRC_GOOD),
                .RSSI(RSSI),
                .CRC_BYTE(CRC_BYTE),
                .SI(SI_rec),
                .CSn(CSn_rec),
                .SCLK(SCLK_rec),
                .SO(SO),
                .begin_rec(configured),
                .done_rec(done_rec),
                .state(state_rec)
        );

reg[39:0] PAYLOAD_disp;
always @ (posedge wireless_clock)
begin
        PAYLOAD_disp <= (CRC_GOOD ? (PAYLOAD) : (PAYLOAD_disp));
end

//for now, only have CFG and TRANS - switch SI/SO/SCLK between the two
//PROMISE: CSn held high in IDLE for cfg and trans, SI held low when not transmitting
assign SI = (SI_cfg || SI_rec);
assign SCLK = SCLK_cfg || SCLK_rec;
assign CSn = (CSn_cfg & CSn_rec);

        //USER 2
        assign user2[11:9] = state_rec[2:0];
        assign user2[8] = done_config;
        //inputs from CC2420
        assign FIFO_PIN = user2[7];
        assign FIFOP_PIN = user2[6];
        assign SFD_PIN = user2[5];
        assign SO = user2[4];
        assign user2[3] = SI;
        assign user2[2] = CSn;
        assign user2[1] = SCLK;
        assign user2[0] = reset_chipn;

        assign user2[30:12] = 19'hZ;
        assign user2[31] = 1'b1;
        //output


        assign led[7] = ~FIFO_PIN;
        assign led[6] = ~SFD_PIN;
        assign led[5] = ~FIFOP_PIN;
        assign led[4:2] = ~state_rec[2:0];
        assign led[1] = ~done_rec; //for now, should always go on
        assign led[0] = ~CRC_GOOD;

        wire[79:0] dots1, dots2, dots3,dots4,dots5,dots6,dots7,dots8;

        /*dots DATA1(clock_27mhz, PAYLOAD_disp[3:0], dots1);
        dots DATA2(clock_27mhz, PAYLOAD_disp[7:4], dots2);
        dots DATA3(clock_27mhz, PAYLOAD_disp[11:8], dots3);
        dots DATA4(clock_27mhz, PAYLOAD_disp[15:12], dots4);
        */
        dots DATA1(clock_27mhz, READSHORTADDR[3:0], dots1);
        dots DATA2(clock_27mhz, READSHORTADDR[7:4], dots2);
        dots DATA3(clock_27mhz, READSHORTADDR[11:8], dots3);
        dots DATA4(clock_27mhz, READSHORTADDR[15:12], dots4);

        dots DATA5(clock_27mhz, PAYLOAD_disp[19:16], dots5);
        dots DATA6(clock_27mhz, PAYLOAD_disp[23:20], dots6);
        dots DATA7(clock_27mhz, PAYLOAD_disp[27:24], dots7);
        dots DATA8(clock_27mhz, PAYLOAD_disp[31:28], dots8);

        display memdisplay(Reset_sync,
                                                            clock_27mhz,
                                                            disp_blank,
                                                            disp_clock,
                                                            disp_rs,
                                                            disp_ce_b,
                                                            disp_reset_b,
                                                            disp_data_out,
                                                            {dots2, dots1, dots4, dots3, dots6,
dots5, dots8, dots7});

     // Logic Analyzer
/*
 assign analyzer1_data = 16'h0;
  assign analyzer1_clock = 1'b1;
  assign analyzer3_data = 16'h0;
  assign analyzer3_clock = 1'b1;
*/
        assign analyzer1_data[15] = SI_rec;
        assign analyzer1_data[14] = SCLK_rec;
        assign analyzer1_data[13] = CSn_rec;
        assign analyzer1_data[12] = SI_cfg;
        assign analyzer1_data[11] = SCLK_cfg;
        assign analyzer1_data[10] = CSn_cfg;
        assign analyzer1_data[9] = configured;
        assign analyzer1_data[8:0] = 0;
        assign analyzer1_clock = clock_27mhz;

//      assign analyzer1_data[15:0] = {READSHORTADDR[7:0], READSHORTADDR[15:8]};
//      assign analyzer1_clock = clock_27mhz;
        assign analyzer3_data[15] = done_config;
        assign analyzer3_data[14] = done_rec;
        assign analyzer3_data[13] = FIFOP_PIN;
        assign analyzer3_data[3] = FIFO_PIN;
        assign analyzer3_data[12] = SFD_PIN;
        assign analyzer3_data[11] = SO;
        assign analyzer3_data[10] = SI;
        assign analyzer3_data[9] = CSn;
        assign analyzer3_data[8] = SCLK;
        assign analyzer3_data[7] = reset_chipn;
        assign analyzer3_data[6:4] = state_rec[2:0];
        assign analyzer3_data[2:0] = 0;
        assign analyzer3_clock = clock_27mhz;

endmodule

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

```
//
// 6.111 FPGA Labkit -- Debounce/Synchronize module
//
//
// Use your system clock for the clock input to produce a synchronous,
// debounced output
//
///////////////////////////////////////////////////////////////////////////////

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

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Number to bitmap decoder
//
//
// Author: Yun Wu, Nathan Ickes
// Date: March 8, 2006
//
// This module converts a 4-bit input to a 80-dot (2 digit) bitmap representing
// the numbers ' 0' through '15'.
//
///////////////////////////////////////////////////////////////////////////////

module dots(clk, num, dots);
   input clk;
   input [3:0] num;
   output [79:0] dots;

   reg [79:0] dots;
   always @ (posedge clk)
     case (num)
       4'd15: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '15'
                       40'b00100111_01000101_01000101_01000101_00111001};
```

```
      4'd14: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '14'
                      40'b00011000_00010100_00010010_01111111_00010000};
      4'd13: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '13'
                      40'b00100010_01000001_01001001_01001001_00110110};
      4'd12: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '12'
                      40'b01100010_01010001_01001001_01001001_01000110};
      4'd11: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '11'
                      40'b00000000_01000010_01111111_01000000_00000000};
      4'd10: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '10'
                      40'b00111110_01010001_01001001_01000101_00111110};
      4'd09: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 9'
                      40'b00000110_01001001_01001001_00101001_00011110};
      4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 8'
                      40'b00110110_01001001_01001001_01001001_00110110};
      4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 7'
                      40'b00000001_01110001_00001001_00000101_00000011};
      4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 6'
                      40'b00111100_01001010_01001001_01001001_00110000};
      4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 5'
                      40'b00100111_01000101_01000101_01000101_00111001};
      4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 4'
                      40'b00011000_00010100_00010010_01111111_00010000};
      4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 3'
                      40'b00100010_01000001_01001001_01001001_00110110};
      4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 2'
                      40'b01100010_01010001_01001001_01001001_01000110};
      4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 1'
                      40'b00000000_01000010_01111111_01000000_00000000};
      4'd00: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 0'
                      40'b00111110_01010001_01001001_01000101_00111110};
      // No default case, becase every case is already accounted for.
    endcase

      endmodule


///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// Change history
//
// 2007-02-09: Fixed register-select race condition. (Thanks to Chris
//             Buenrostro for finding this bug and David Wentzloff for
//             implementing the fix.)
// 2005-05-09: Made <dots> input registered, and converted the 640-input MUX
//             to a 640-bit shift register.
//
///////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b,
                disp_reset_b, disp_data_out, dots);
```

```
input reset; // Active high
input clock_27mhz;
output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
       disp_reset_b;
input [639:0] dots;

reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

//
// Display Clock
//
// Generate a 500kHz clock for driving the displays.
//

reg [4:0] count;
reg [7:0] reset_count;
reg clock;
wire dreset;

always @(posedge clock_27mhz)
  begin
     if (reset)
       begin
          count = 0;
          clock = 0;
       end
     else if (count == 26)
       begin
          clock = ~clock;
          count = 5'h00;
       end
     else
       count = count+1;
  end

always @(posedge clock_27mhz)
  if (reset)
    reset_count <= 100;
  else
    reset_count <= (reset_count==0) ? 0 : reset_count-1;

assign dreset = (reset_count != 0);

assign disp_clock = ~clock;

//
// Display State Machine
//

reg [7:0] state;
reg [9:0] dot_index;
reg [31:0] control;
reg [639:0] ldots;

assign disp_blank = 1'b0; // low = not blanked

always @(posedge clock)
  if (dreset)
    begin
```

```
      state <= 0;
      dot_index <= 0;
      control <= 32'h7F7F7F7F;
   end
 else
   casex (state)
     8'h00:
       begin
          // Reset displays
          disp_data_out <= 1'b0;
          disp_rs <= 1'b0; // 0 = dot register
          disp_ce_b <= 1'b1;
          disp_reset_b <= 1'b0;
          dot_index <= 0;
          state <= state+1;
       end

     8'h01:
       begin
          // End reset
          disp_reset_b <= 1'b1;
          state <= state+1;
       end

     8'h02:
       begin
          // Initialize dot register
          disp_ce_b <= 1'b0;
          disp_data_out <= 1'b0; // dot_index[0];
          if (dot_index == 639)
            state <= state+1;
          else
            dot_index <= dot_index+1;
       end

     8'h03:
       begin
          // Latch dot data
          disp_rs <= 1'b1; // Select the control register
          disp_ce_b <= 1'b1;
          dot_index <= 31;
          state <= state+1;
       end

     8'h04:
       begin
          // Setup the control register
          disp_ce_b <= 1'b0;
          disp_data_out <= control[31];
          control <= {control[30:0], 1'b0};
          if (dot_index == 0)
            state <= state+1;
          else
            dot_index <= dot_index-1;
       end

     8'h05:
       begin
          // Latch the control register data
```

```
              disp_rs <= 1'b0; // Select the dot register
              disp_ce_b <= 1'b1;
              dot_index <= 639;
              ldots <= dots;
              state <= state+1;
           end

         8'h06:
           begin
              // Load the user's dot data into the dot register
              disp_ce_b <= 1'b0;
              disp_data_out <= ldots[639];
              ldots <= ldots<<1;
              if (dot_index == 0)
                state <= 5;
              else
                dot_index <= dot_index-1;
           end
       endcase

endmodule
```

```
`timescale 1ns / 1ps
// Analog to digital conversion module. Contains the chunk and sample modules and generates the
serial
// and sample clocks
//

module adconv(clk, reset, busy, sdata, chunk_ready_left, convst, notcs, sclk, lout1, lout2,
lout3,
                   lout4, lout5, double);

                   input clk, reset, busy, sdata, double;
                   output [15:0] lout1, lout2, lout3, lout4, lout5;
                   output chunk_ready_left, convst, notcs, sclk;

                   //wire sclk, sampletick;
                   wire [31:0] word;
                   wire word_ready, sample;
                   wire sclk_tick, samptick;

                   addivder div(clk, sampletick, sclk, double);
                   divider divider(clk, samptick, sclk_tick, double);

                   chunkfsm chunk(clk, reset, chunk_ready_left, sampletick, word, word_ready,
                                                              busy, sample, convst, lout1,
lout2, lout3, lout4, lout5);

                   samplefsm samp(clk, sclk_tick, sample, busy, sdata, notcs, word, word_ready);
endmodule


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

//generate the required sampling and serial clocks

module divider(clk, sampletick, sclk, double);
```

```
input clk, double;
          output sampletick, sclk;
          wire [24:0] ratio;
          assign ratio = double? 25'd5 : 25'd10; // a switch determines the sampling rate - 45
or 90 kHz

          ticker divsamp(sclk, sampletick, 25'd60);
          //ticker divsamp(clk, sampletick, 25'd120); // for testing
          // 45kHz sample tick

          ticker divsclk(clk, sclk, ratio);  // five or ten times slower, depending on double.
          //ticker divsclk(clk, sclk, 25'd2);  // for testing
          // 2.7-MHz serial clock. 60 times faster than the sample tick
endmodule

module addivder(clk, sampletick, sclk, double);

          input clk, double;
          output sampletick, sclk;
          wire [24:0] ratio;
          assign ratio = double? 25'd5 : 25'd10; // a switch determines the sampling rate - 45
or 90 kHz

          ticker divsamp(sclk, sampletick, 25'd60);

          adticker divsclk(clk, sclk, ratio);
endmodule

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

//  outputs a tick one clk cycle long once every "div" clk cycles
// by dividing the clk signal by the value div.

module ticker (clk, tick, div);

          input clk;
          input [24:0] div;
          output tick;
          reg [24:0] count;

          initial count = 0;

          always @ (posedge clk) begin
                    if (count < div-1) count <= count + 1;
                    else count <= 0 ;
          end

          assign tick = (count == div-1)? 1:0;


endmodule

// gives a 50 percent duty cycle for the same functionality as ticker.
module adticker(clk, tick, div);
          input clk;
          input [24:0] div;
          output tick;
          reg [24:0] count;
          initial count = 0;

          always @ (posedge clk) begin
                    if (count < div-1) count <= count + 1;
                    else count <=0;
          end

          assign tick = (count < div[24:1])? 0:1;
endmodule


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

//
// clocks in the data from the analog to digital converter
// on the sclk serial clock. Responds to the "sample" signal
// telling it when it's ok to sample
//
// outputs a 32 bit word and a word-ready pulse indicating that the
// sample is ready

module samplefsm(clk, sclk, sample, busy, dout, notcs, word, word_ready);

          input clk, sclk, sample, busy, dout;
   output [31:0] word;
          output notcs, word_ready;
          reg notcs;
          reg [31:0] word;
          reg [5:0] i;
          initial word = 32'd0;

          always @ (posedge sclk) begin
                    if (sample == 0) begin
                    word[31] <= 0;
                    i <= 6'd52;
                    notcs <=1;
                    end

                    else if(i > 6'd32) begin // wait till conversion has finished
                              word[31] <= 0;
                              i <= i-1;
                              notcs<=1;
                    end

                    else if(i > 6'd0) begin // clock in the data into word ={left[15:0]
right[15:0]}
                              word[i-1] <=dout;
                              i<= i-1;
                              notcs<=0;
                    end

                    else begin
                              i<= i;
                              notcs<=1;
                    end
          end
          assign word_ready = (i == 0)? 1:0; //goes high one clock cycle * after* data is
```

```
clocked in, so data

          // is valid on the positive edge and can be put into a stack
endmodule

`timescale 1ns / 1ps
// The chunk FSM
// dispatches the sample FSM
// Is in charge of sending chunks to the compresser
// Starts conversions
// Convention: Left bits arrive first in word then Right bits


module chunkfsm(clk, reset, chunk_ready_left, sampletick, word, word_ready, busy, sample,
convst, lout1, lout2, lout3,
                    lout4, lout5);

          input clk, reset, sampletick, busy, word_ready;
          input [31:0] word;
          output sample, convst, chunk_ready_left;
          output [15:0] lout1, lout2, lout3, lout4, lout5; // to the left compresser
          wire [15:0] left;
          wire crl_int, clkdiv;

          assign convst = ~sampletick; // active high
          // the specification for the minimum time that convst can be low is met

          assign sample = convst;
          assign left = word[15:0]; // the left sample

          chunkstack lstack(reset, word_ready, left, crl_int, lout1, lout2, lout3, lout4,
lout5);

          ticker t(clk, clkdiv, 25'd8);
          leveltopulse ltpl(crl_int, clkdiv, chunk_ready_left);

endmodule

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

`timescale 1ns / 1ps

//
// the stack for the chunk finite state machine
// stores samples from the sampling module and outputs
// chunks of five samples each
//

module chunkstack(reset, inclk, in, chunk_ready, outone, outtwo, outthree, outfour, outfive);
          input reset,inclk;
          input [15:0] in;
          output [15:0] outone, outtwo, outthree, outfour, outfive;
          output chunk_ready; // to signal to the compressor that data is coming. Data
definitely valid on negative edge.
          reg [2:0] stacknum;

          always @ (posedge inclk or posedge reset) begin
                    if(reset) stacknum <=0;
                    else begin
```

```
                              if(stacknum == 3'd4) stacknum <=0;
                              else stacknum <= stacknum + 1;
                    end
          end

          assign first = (stacknum == 3'd0 );
          assign second = (stacknum == 3'd1);
          assign third = (stacknum == 3'd2 );
          assign fourth =(stacknum == 3'd3 );
          assign fifth  =(stacknum == 3'd4 );
          assign chunk_ready = first;

          // the five stacks. Store up samples in order, and then output them on the next cycle
          stack sta1( reset, first, in, first, outone);
          stack sta2( reset, second, in, first, outtwo);
          stack sta3( reset, third, in, first, outthree);
          stack sta4( reset, fourth, in, first, outfour);
          stack sta5( reset, fifth, in, first, outfive);



endmodule


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

// The stack module - a simple FIFO data structure
// with an in pointer and an out pointer that follow each-other
// Has two separate signals that control pushing and popping
// called inclk and outclk.

module stack(reset, inclk, in, outclk, out);
          input reset, inclk, outclk;
          input [15:0] in;
          output [15:0] out;
          reg [15:0] out;
          reg [15:0] thestack [4:0] ; // array of  values, each 16 btis wide.
          reg [7:0] inptr; // index of the next empty location in the stack
          reg [7:0] outptr; // index of the next output
          reg [7:0] inptr_int;
          reg [7:0] outptr_int;
          reg [16:0] stack_in;

          initial begin
                    inptr <= 0;
                    outptr <= 0;
          end


          always @ (posedge inclk) begin // "in" must be valid on the positive edge
                    if(reset) begin
                    thestack[inptr] <= 16'd0;
                    inptr <= 8'd0;
                    end

                    else if(inptr == 5) begin
                              thestack[inptr] <= in;
                              inptr <= 0;
                    end

                    else begin
                              thestack[inptr] <= in;
                              inptr <= inptr + 1;
                    end
          end // always


          always @ (posedge outclk) begin
                    if(reset) begin
                              out <= 16'd0;
                              outptr <=0;
                    end

                    else if (outptr == inptr) begin // stack is empty
                              out <= 0;
                              outptr <= inptr;
                    end

                    else if (outptr == 5) begin
                              out <= thestack[outptr];
                              outptr <= 0;
                    end

                    else begin
                              out <= thestack[outptr];
                              outptr <= outptr + 1;
                    end
          end// always

endmodule

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

// The digital to analog interface.
// Takes in 5 16-bit numbers, stores them and send them out in order in serial, MSB first.
// The AD5063 expects an 8-bit header of zeroes, so this is tacked on as well
//

module dastack(clk, reset, data_valid, din1, din2, din3, din4, din5, d_out, sync, sclk, double);
          input clk, reset, data_valid, double;
          input [15:0] din1, din2, din3, din4, din5; // from decompresser
          output d_out, sync, sclk; // to AD5063
          reg d_out;
          reg [2:0] count; // keeps track of which sample to push out of the stacks.
          reg [5:0] i ; // for the serializer
          wire [15:0] sample1, sample2, sample3, sample4, sample5;
          wire [23:0] sample; // will be serialized

          divider div(clk, sync, sclk, double);

          always @ (posedge sync or posedge reset) begin
                    if (reset) count <= 0;
                    else if (count == 3'd4) count <=0;
                    else count <= count + 1;
          end
```

```
         // the signals that determine which stack will supply data to 'sample'.
         assign outclk1 = (count == 3'd0);
         assign outclk2 = (count == 3'd1);
         assign outclk3 = (count == 3'd2);
         assign outclk4 = (count == 3'd3);
         assign outclk5 = (count == 3'd4);

         stack stack1(reset, data_valid, din1, outclk1, sample1);
         stack stack2(reset, data_valid, din2, outclk2, sample2);
         stack stack3(reset, data_valid, din3, outclk3, sample3);
         stack stack4(reset, data_valid, din4, outclk4, sample4);
         stack stack5(reset, data_valid, din5, outclk5, sample5);

         assign sample[15:0] = (count == 3'd0)? sample1 :
                                                                               ((count == 3'd1)?
sample2 :
                                                                               ((count ==3'd2)?
sample3 :
                                                                               ((count == 3'd3)?
sample4 : sample5)));
         assign sample[23:16] = 8'b0;

         always @ (posedge sclk or posedge sync) begin
                  if (sync) i <= 24;
                  else if (i > 0) begin
                           d_out <= sample[i-1];
                           i<= i - 1;
                  end
                  else begin
                           d_out <= 0;
                           i <=0;
                  end
         end


endmodule
```