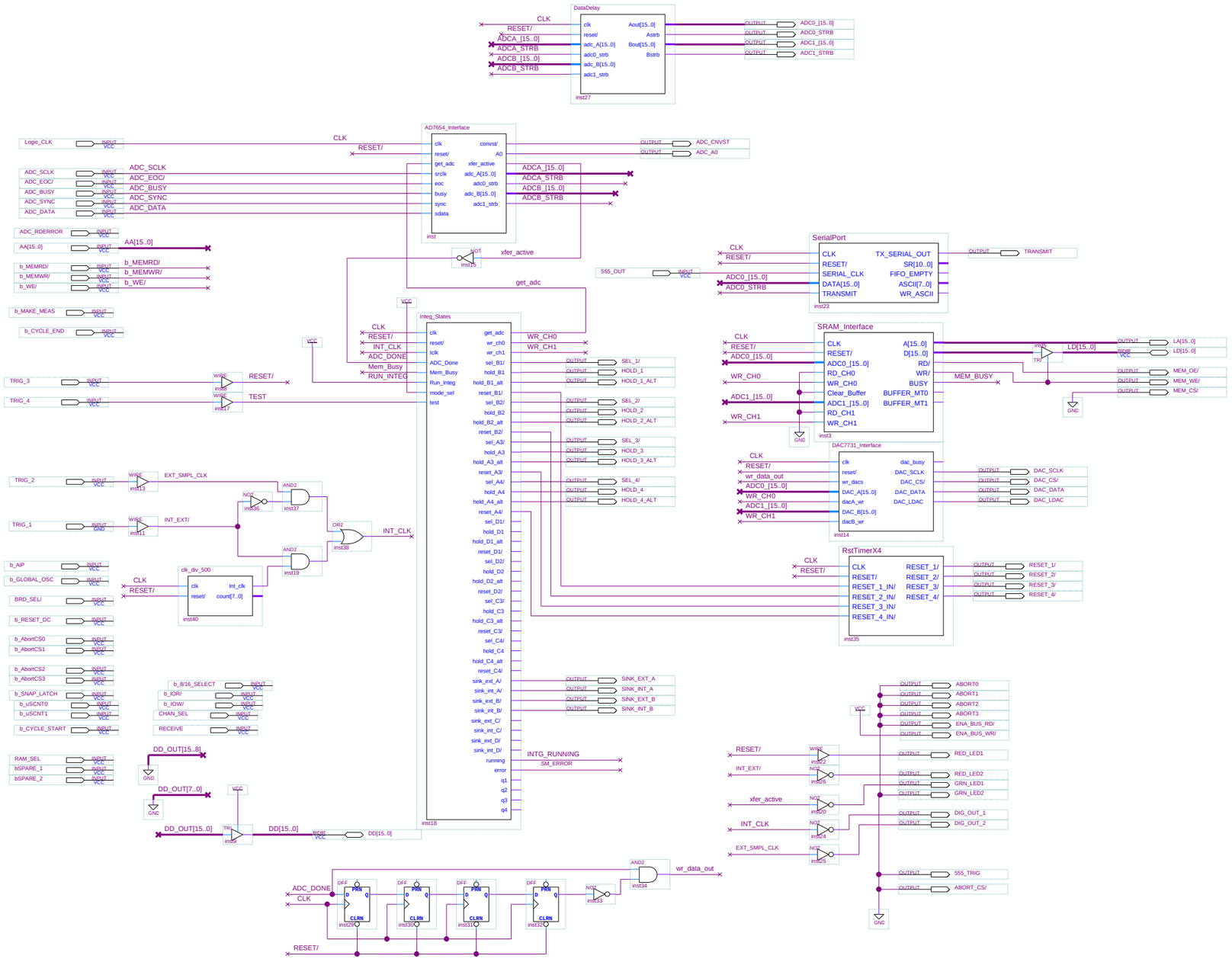

DataDelay
AD7654_Interface
SerialPort
Integ_States
SRAM_Interface
DAC7731_Interface
RstTimerX4
clk_div_500
INTG_RUNNING
SM_ERROR
xfer_active
get_adc
INT_CLK
EXT_SMPL_CLK
wr_data_out
ADC_DONE
RESET/
CLK
ADC_SCLK
ADC_EOC/
ADC_BUSY
ADC_SYNC
ADC_DATA
ADC_RDERROR
AA[15..0]
b_MEMRD/
b_MEMWR/
b_WE/
b_MAKE_MEAS
b_CYCLE_END
TRIG_3
TRIG_4
TRIG_2
TRIG_1
b_AIP
b_GLOBAL_OSC
BRD_SEL/
b_RESET_DC
b_AbortCS0
b_AbortCS1
b_AbortCS2
b_AbortCS3
b_SNAP_LATCH
b_uSCNT0
b_uSCNT1
b_CYCLE_START
RAM_SEL
bSPARE_1
bSPARE_2
b_8/16_SELECT
b_IOR/
b_IOW/
CHAN_SEL
RECEIVE
DD_OUT[15..8]
DD_OUT[7..0]
DD_OUT[15..0]
DD[15..0]
ADC0_[15..0]
ADC0_STRB
ADC1_[15..0]
ADC1_STRB
ADC_CNVST
ADC_A0
TRANSMIT
555_OUT
LA[15..0]
LD[15..0]
MEM_OE/
MEM_WE/
MEM_CS/
MEM_BUSY
DAC_SCLK
DAC_CS/
DAC_DATA
DAC_LDAC
RESET_1/
RESET_2/
RESET_3/
RESET_4/
ABORT0
ABORT1
ABORT2
ABORT3
ENA_BUS_RD/
ENA_BUS_WR/
RED_LED1
RED_LED2
GRN_LED1
GRN_LED2
DIG_OUT_1
DIG_OUT_2
555_TRIG
ABORT_CS/
SEL_1/
HOLD_1
HOLD_1_ALT
SEL_2/
HOLD_2
HOLD_2_ALT
SEL_3/
HOLD_3
HOLD_3_ALT
SEL_4/
HOLD_4
HOLD_4_ALT
SINK_EXT_A
SINK_INT_A
SINK_EXT_B
SINK_INT_B
WR_CH0
WR_CH1
ADC_DONE
Mem_Busy
RUN_INTEG
INT_CLK
EXT_SMPL_CLK
INT_EXT/
xfer_active
GND
VCC

```
SUBDESIGN Integ_States
(
   clk, reset/, Iclk, ADC_Done, Mem_Busy, Run_Integ : INPUT;
   mode_sel : INPUT;  % high = default (ext switch -> LM, int switch -> Test) %
   test : INPUT;

   get_adc, wr_ch0, wr_ch1 : OUTPUT;
   sel_B1/, hold_B1, hold_B1_alt, reset_B1/: OUTPUT;
   sel_B2/, hold_B2, hold_B2_alt, reset_B2/: OUTPUT;
   sel_A3/, hold_A3, hold_A3_alt, reset_A3/: OUTPUT;
   sel_A4/, hold_A4, hold_A4_alt, reset_A4/: OUTPUT;
   sel_D1/, hold_D1, hold_D1_alt, reset_D1/: OUTPUT;
   sel_D2/, hold_D2, hold_D2_alt, reset_D2/: OUTPUT;
   sel_C3/, hold_C3, hold_C3_alt, reset_C3/: OUTPUT;
   sel_C4/, hold_C4, hold_C4_alt, reset_C4/: OUTPUT;

   sink_ext_A/, sink_int_A/, sink_ext_B/, sink_int_B/ : OUTPUT;
   sink_ext_C/, sink_int_C/, sink_ext_D/, sink_int_D/ : OUTPUT;
   running, error : OUTPUT;
    q1,q2,q3,q4 : OUTPUT;
)

VARIABLE
sm :   MACHINE
          OF BITS (q1, q2, q3, q4)
          WITH STATES (
            idle    = B"0000",
            s1_set  = B"0001",
            s1_strt = B"0010",
            s1_conv = B"0011",
            s1_wr0  = B"0100",
            s1_wait = B"0101",
            s1_wr1  = B"0110",
            s1_done = B"0111",
            s2_set  = B"1000",
            s2_strt = B"1001",
            s2_conv = B"1010",
            s2_wr0  = B"1011",
            s2_wait = B"1100",
            s2_wr1  = B"1101",
            s2_done = B"1110",
            s1_err  = B"1111"
            );

    s1,s2 : NODE; % Select Signals (active high) %
    h1,h2 : NODE; % Hold Signals (active high) / !Integrate (active low) %
    r1,r2 : NODE; % Reset Signals (active high) %
    clamp : NODE; % Drain input when nothing else is switched on %

    sample[1..0] : dff; %used to create a pulse from the rising edge of Iclk %
    smple : NODE;

BEGIN

    sample[].clk  = clk;
    sample[].clrn = reset/;
    sample[0].d = Iclk;
    sample[1].d = sample[0].q;

    smple = Iclk & !sample[1].q;

    sm.clk = clk;
    sm.reset = !reset/;

```

```
 clamp = h1 & h2;
 IF (sm != idle) THEN running = VCC; ELSE running = GND; END IF;

TABLE
%  current                    current                  next                          current                      %
%  state                      inputs                   state                         outputs                      %

   sm      ,  smple ,  ADC_Done, Mem_Busy, Run_Integ =>   sm    , get_adc, wr_ch0 , wr_ch1 , s1, s2, h1, h2, r1, r2, error ;

   idle    ,    x   ,     x    ,    x    ,    0      => idle    ,    0   ,    0   ,    0   , 1 , 0 , 1 , 1 , 1 , 1 ,   0   ;
   idle    ,    x   ,     x    ,    x    ,    1      => s1_set  ,    0   ,    0   ,    0   , 1 , 0 , 1 , 1 , 1 , 1 ,   0   ;

   s1_set  ,    x   ,     x    ,    x    ,    x      => s1_strt ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;

   s1_strt ,    x   ,     1    ,    x    ,    x      => s1_strt ,    1   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_strt ,    x   ,     0    ,    x    ,    x      => s1_conv ,    1   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
                      %    ^--- ADC_DONE is the inverse of the ADC_BUSY line %

   s1_conv ,    x   ,     0    ,    x    ,    x      => s1_conv ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_conv ,    x   ,     1    ,    1    ,    x      => s1_conv ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_conv ,    x   ,     1    ,    0    ,    x      => s1_wr0  ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;

   s1_wr0  ,    x   ,     x    ,    0    ,    x      => s1_wr0  ,    0   ,    1   ,    0   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;
   s1_wr0  ,    x   ,     x    ,    1    ,    x      => s1_wait ,    0   ,    1   ,    0   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;

   s1_wait ,    x   ,     x    ,    1    ,    x      => s1_wait ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;
   s1_wait ,    x   ,     x    ,    0    ,    x      => s1_wr1  ,    0   ,    0   ,    0   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;

   s1_wr1  ,    x   ,     x    ,    0    ,    x      => s1_wr1  ,    0   ,    0   ,    1   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;
   s1_wr1  ,    x   ,     x    ,    1    ,    x      => s1_done ,    0   ,    0   ,    1   , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;

   s1_done ,    x   ,     x    ,    x    ,    0      => idle    ,    0   ,    0   ,    0   , 0 , 1 , 1 , 0 , 0 , 0 ,   0   ;
   s1_done ,    0   ,     x    ,    x    ,    1      => s1_done ,    0   ,    0   ,    0   , 0 , 1 , 1 , 0 , 0 , 0 ,   0   ;
   s1_done ,    1   ,     x    ,    x    ,    1      => s2_set  ,    0   ,    0   ,    0   , 0 , 1 , 1 , 0 , 0 , 0 ,   0   ;

   s2_set  ,    x   ,     x    ,    x    ,    x      => s2_strt ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;

   s2_strt ,    x   ,     1    ,    x    ,    x      => s2_strt ,    1   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_strt ,    x   ,     0    ,    x    ,    x      => s2_conv ,    1   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
                      %    ^--- ADC_DONE is the inverse of the ADC_BUSY line %

   s2_conv ,    x   ,     0    ,    x    ,    x      => s2_conv ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_conv ,    x   ,     1    ,    1    ,    x      => s2_conv ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_conv ,    x   ,     1    ,    0    ,    x      => s2_wr0  ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;

   s2_wr0  ,    x   ,     x    ,    0    ,    x      => s2_wr0  ,    0   ,    1   ,    0   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;
   s2_wr0  ,    x   ,     x    ,    1    ,    x      => s2_wait ,    0   ,    1   ,    0   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;

   s2_wait ,    x   ,     x    ,    1    ,    x      => s2_wait ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;
   s2_wait ,    x   ,     x    ,    0    ,    x      => s2_wr1  ,    0   ,    0   ,    0   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;

   s2_wr1  ,    x   ,     x    ,    0    ,    x      => s2_wr1  ,    0   ,    0   ,    1   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;
   s2_wr1  ,    x   ,     x    ,    1    ,    x      => s2_done ,    0   ,    0   ,    1   , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;

   s2_done ,    x   ,     x    ,    x    ,    0      => idle    ,    0   ,    0   ,    0   , 1 , 0 , 0 , 1 , 0 , 0 ,   0   ;
   s2_done ,    0   ,     x    ,    x    ,    1      => s2_done ,    0   ,    0   ,    0   , 1 , 0 , 0 , 1 , 0 , 0 ,   0   ;
   s2_done ,    1   ,     x    ,    x    ,    1      => s1_set  ,    0   ,    0   ,    0   , 1 , 0 , 0 , 1 , 0 , 0 ,   0   ;



END TABLE;

 sel_B1/ = !s1;
 sel_B2/ = !s2;
```

```
    sel_A3/ = !s1;
    sel_A4/ = !s2;
    sel_D1/ = !s1;
    sel_D2/ = !s2;
    sel_C3/ = !s1;
    sel_C4/ = !s2;

    reset_B1/ = !r1;
    reset_B2/ = !r2;
    reset_A3/ = !r1;
    reset_A4/ = !r2;
    reset_D1/ = !r1;
    reset_D2/ = !r2;
    reset_C3/ = !r1;
    reset_C4/ = !r2;

    IF ( mode_sel == VCC) THEN
        % Internal Switches %
        hold_B1 = h1 # !test;
        hold_B2 = h2 # !test;
        hold_A3 = h1 # !test;
        hold_A4 = h2 # !test;
        hold_D1 = h1 # !test;
        hold_D2 = h2 # !test;
        hold_C3 = h1 # !test;
        hold_C4 = h2 # !test;
        % External Switches %
        hold_B1_alt = h1 # test;
        hold_B2_alt = h2 # test;
        hold_A3_alt = h1 # test;
        hold_A4_alt = h2 # test;
        hold_D1_alt = h1 # test;
        hold_D2_alt = h2 # test;
        hold_C3_alt = h1 # test;
        hold_C4_alt = h2 # test;
        % Clamping Switches %
        sink_ext_A/ = !( test # clamp );
        sink_ext_B/ = !( test # clamp );
        sink_ext_C/ = !( test # clamp );
        sink_ext_D/ = !( test # clamp );
        sink_int_A/ = !(!test # clamp );
        sink_int_B/ = !(!test # clamp );
        sink_int_C/ = !(!test # clamp );
        sink_int_D/ = !(!test # clamp );
    ELSE
        % Internal Switches %
        hold_B1 = h1 # test;
        hold_B2 = h2 # test;
        hold_A3 = h1 # test;
        hold_A4 = h2 # test;
        hold_D1 = h1 # test;
        hold_D2 = h2 # test;
        hold_C3 = h1 # test;
        hold_C4 = h2 # test;
        % External Switches %
        hold_B1_alt = h1 # !test;
        hold_B2_alt = h2 # !test;
        hold_A3_alt = h1 # !test;
        hold_A4_alt = h2 # !test;
        hold_D1_alt = h1 # !test;
        hold_D2_alt = h2 # !test;
        hold_C3_alt = h1 # !test;
        hold_C4_alt = h2 # !test;
        % Clamping Switches %
```

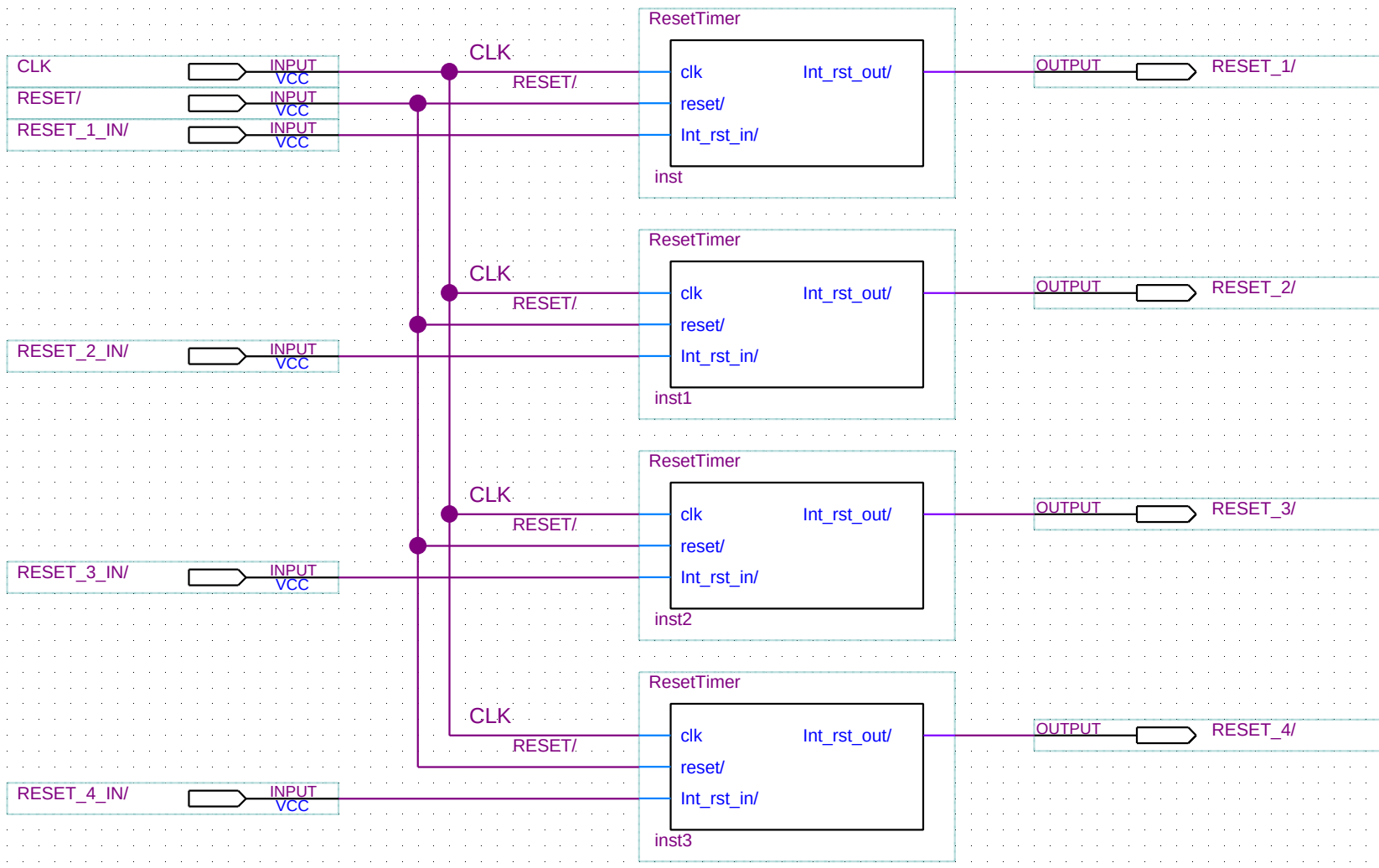
```
        sink_ext_A/ = !(!test # clamp );
        sink_ext_B/ = !(!test # clamp );
        sink_ext_C/ = !(!test # clamp );
        sink_ext_D/ = !(!test # clamp );
        sink_int_A/ = !( test # clamp );
        sink_int_B/ = !( test # clamp );
        sink_int_C/ = !( test # clamp );
        sink_int_D/ = !( test # clamp );
    END IF;

END;

```

```
SUBDESIGN DataDelay
(
   clk, reset/         :INPUT;
   adc_A[15..0], adc0_strb, adc_B[15..0], adc1_strb : INPUT;

   Aout[15..0], Astrb, Bout[15..0], Bstrb : OUTPUT;
)

VARIABLE
Aout[15..0]   : DFFE;
Bout[15..0]   : DFFE;

delay[8..0]    : DFFE;
run            : SRFF;
strobe/         : NODE;
done           : NODE;
BEGIN

    Aout[].clk = clk;
    Aout[].clrn = reset/;
    Aout[].d = adc_A[];
    Aout[].ena = adc0_strb;


    Bout[].clk = clk;
    Bout[].clrn = reset/;
    Bout[].d = adc_B[];
    Bout[].ena = adc1_strb;

    run.clk = clk;
    run.clrn = reset/;
    run.s = adc0_strb # adc1_strb;
    run.r = done;

    delay[].clk = clk;
    delay[].clrn = run.q;
    delay[].ena  = run.q;
    delay[].d = delay[].q + 1;

    IF((delay[] >= 250)&(delay[] < 253)) THEN strobe/ = GND;
    ELSE strobe/ = VCC;
    END IF;

    IF((delay[] >= 253)) THEN done = VCC; ELSE done = GND;
    END IF;

    Astrb = strobe/;
    Bstrb = strobe/;

END;
```

```
SUBDESIGN clk_div_500
(
   clk, reset/: INPUT;
   Int_clk, count[7..0] : OUTPUT;
)

% Clock expected to be (25 MHz / 500) %

VARIABLE
    count[7..0]    : DFF;
    load           : DFF;
    toggle         : TFF;
    tgl            : NODE;

BEGIN

%=================================================================%
    count[].clk = clk;

    load.clk = clk;
    load.d   =  tgl;

    toggle.t   = VCC;
    toggle.clk = load.q;
    toggle.clrn = reset/;

    IF load.q THEN
      count[].d = B"11111000";
    ELSE
      count[].d = count[].q - 1;
    END IF;

    IF (count[].q == B"00000000") THEN tgl = VCC; ELSE tgl = GND; END IF;

    Int_clk = toggle.q;
END;
```

```
SUBDESIGN AD7654_Interface
(
   clk, reset/, get_adc         :INPUT;
   srclk, eoc, busy, sync, sdata :INPUT;

   convst/, A0, xfer_active : OUTPUT;
   adc_A[15..0], adc0_strb, adc_B[15..0], adc1_strb : OUTPUT;
)

VARIABLE
sr[31..0]      : DFFE;
dataA[15..0]   : DFFE;
dataB[15..0]   : DFFE;

delay[4..0]    : DFFE;
done           : DFFE;
strb           : DFFE;

BEGIN

    % Delay the sample due to ADC Input RC filter %
    delay[].clk = clk;
    delay[].clrn = get_adc; % reset clock when not active %
    delay[].d = delay[].q + 1;


    IF(delay[] > 25) THEN
        delay[].ena = GND; % not counting %
        convst/ = GND; %active now %
    ELSE
        delay[].ena = VCC; % still counting %
        convst/ = VCC; %not active yet%
    END IF;

    % Input Serial Data Shift Register %
    sr[].clk = srclk;
    sr[].clrn = reset/;
    sr[].ena  = sync;
    sr[0].d = sdata;
    sr[31..1].d = sr[30..0].q;

    % Buffer Register for the A channel Data %
    dataA[15..0].clk  = !sync;
    dataA[15..0].clrn = reset/;
    dataA[15..0].ena  = VCC;
    dataA[15..0].d    = sr[31..16].q;
    adc_A[15..0]      = dataA[15..0].q;

    % Buffer Register for the B channel Data %
    dataB[15..0].clk  = !sync;
    dataB[15..0].clrn = reset/;
    dataB[15..0].ena  = VCC;
    dataB[15..0].d    = sr[15..0].q;
    adc_B[15..0]      = dataB[15..0].q;

    A0 = GND; % get adc data from INA1 and INB1 %

    done.clk  = clk;        % synchronize the asyncronous busy signal %
    done.clrn = reset/;
    done.ena  = VCC;
    done.d    = !busy;
    xfer_active = !done.q;

    strb.clk  = clk;
```

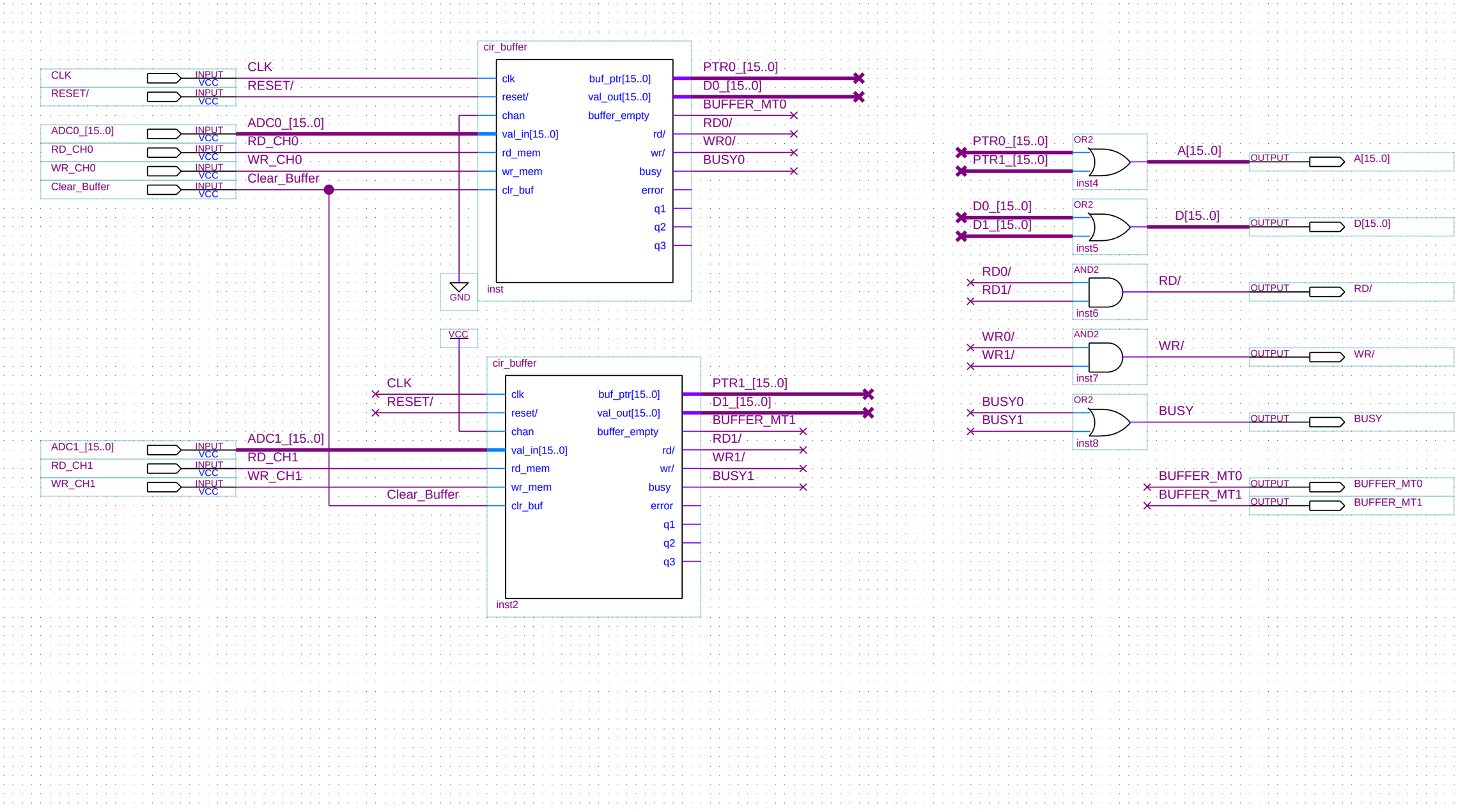
```
    strb.clrn = reset/;
    strb.ena  = VCC;
    strb.d    = done.q;

    adc0_strb = done.q & !strb.q;
    adc1_strb = done.q & !strb.q;

END;

```


CLK
INPUT
VCC
RESET/
INPUT
VCC
RESET_1_IN/
INPUT
VCC
CLK
RESET/
ResetTimer
clk
Int_rst_out/
reset/
Int_rst_in/
inst
OUTPUT
RESET_1/
CLK
RESET/
ResetTimer
clk
Int_rst_out/
reset/
Int_rst_in/
inst1
OUTPUT
RESET_2/
RESET_2_IN/
INPUT
VCC
CLK
RESET/
ResetTimer
clk
Int_rst_out/
reset/
Int_rst_in/
inst2
OUTPUT
RESET_3/
RESET_3_IN/
INPUT
VCC
CLK
RESET/
ResetTimer
clk
Int_rst_out/
reset/
Int_rst_in/
inst3
OUTPUT
RESET_4/
RESET_4_IN/
INPUT
VCC

```
SUBDESIGN ResetTimer
(
   clk, reset/  :INPUT;
   Int_rst_in/   : INPUT;

   Int_rst_out/ : OUTPUT;
)

VARIABLE
delay[7..0]    : DFFE;
run            : SRFF;
done           : NODE;
BEGIN

    run.clk = clk;
    run.clrn = reset/;
    run.s = !Int_rst_in/;
    run.r = done;

    delay[].clk = clk;
    delay[].clrn = run.q;
    delay[].ena  = run.q;
    delay[].d = delay[].q + 1;

    IF((delay[] >= 2)&(delay[] < 252)) THEN Int_rst_out/ = GND;
    ELSE Int_rst_out/ = VCC;
    END IF;

    IF((delay[] >= 252)) THEN done = VCC; ELSE done = GND;
    END IF;


END;
```

```
SUBDESIGN DAC7731_Interface
(
   clk, reset/, wr_dacs         :INPUT;
   DAC_A[15..0], dacA_wr, DAC_B[15..0], dacB_wr : INPUT;

   dac_busy : OUTPUT;
   DAC_SCLK, DAC_CS/, DAC_DATA, DAC_LDAC : OUTPUT;
    sregclk, q1,q2,sr_done, sr_cnt[4..0] : OUTPUT;
)

VARIABLE
   sm :   MACHINE
          OF BITS (q1, q2)
          WITH STATES (
            idle   = B"00",
            ld_sr  = B"01",
            clk_sr = B"11",
            ld_dac = B"10"
            );


clk/2          : TFF;

sr[31..0]      : DFFE;
dataA[15..0]   : DFFE;
dataB[15..0]   : DFFE;

sr_cnt[4..0]   : DFFE;
sync_cs        : DFFE;
strt_ld        : NODE;

BEGIN
    % Buffer Register for the A channel Data %
    dataA[15..0].clk  = clk;
    dataA[15..0].clrn = reset/;
    dataA[15..0].ena  = dacA_wr;
    dataA[15..0].d    = DAC_A[15..0];

    % Buffer Register for the B channel Data %
    dataB[15..0].clk  = clk;
    dataB[15..0].clrn = reset/;
    dataB[15..0].ena  = dacB_wr;
    dataB[15..0].d    = DAC_B[15..0];

    % 80 ns clk for DAC data clock %
    clk/2.clk   = clk;
    clk/2.clrn  = reset/;
    clk/2.t     = VCC;

    sregclk  = clk/2.q;
    DAC_SCLK = !clk/2.q;



    % Syncronize the DAC_CS/ to the DAC_SCLK %
%   sync_cs.clk  = DAC_SCLK;
    sync_cs.clrn = reset/;
    sync_cs.d    = strt_ld;

    DAC_CS/ = !sync_cs.q;
%
    % Input Serial Data Shift Register %
    sr[].clk  = sregclk;
    sr[].clrn = reset/;
```

```
65      sr[].ena  = !DAC_CS/ # wr_dacs;
66
67      if (wr_dacs) THEN sr[15..0].d = dataA[15..0].q; sr[31..16].d = dataB[15..0].q;
68      ELSE              sr[0].d = GND; sr[31..1].d = sr[30..0].q;
69      END IF;
70
71      DAC_DATA = sr[31].q;
72
73      % Counter to count the 32 clocks for loading the DACs %
74      sr_cnt[].clk  = sregclk;
75      sr_cnt[].clrn = reset/;
76
77      IF(sm == ld_sr) THEN sr_cnt[].d = 31;
78      ELSE                 sr_cnt[].d = sr_cnt[].q - 1;
79      END IF;
80
81      IF((sm == clk_sr)#(sm == ld_sr)) THEN sr_cnt[].ena = VCC;
82      ELSE                                  sr_cnt[].ena = GND;
83      END IF;
84
85      IF(sr_cnt[] > 0) THEN sr_done = GND;
86      ELSE                  sr_done = VCC;
87      END IF;
88
89      % state machine to control the DAC write sequence %
90      sm.clk = sregclk;
91      sm.reset = !reset/;
92
93         TABLE
94      % current         current        next                 current                  %
95      % state           inputs         state                outputs                  %
96
97        sm      , wr_dacs, sr_done  =>   sm    , dac_busy, DAC_CS/, DAC_LDAC;
98
99        idle    ,   0    ,   x      => idle    ,    0    ,    1   ,    0    ;
100       idle    ,   1    ,   x      => ld_sr   ,    0    ,    1   ,    0    ;
101
102       ld_sr   ,   x    ,   1      => ld_sr   ,    1    ,    1   ,    0    ;
103       ld_sr   ,   x    ,   0      => clk_sr  ,    1    ,    1   ,    0    ;
104
105       clk_sr  ,   x    ,   0      => clk_sr  ,    1    ,    0   ,    0    ;
106       clk_sr  ,   x    ,   1      => ld_dac  ,    1    ,    0   ,    0    ;
107
108       ld_dac  ,   x    ,   x      => idle    ,    1    ,    0   ,    1    ;
109
110
111     END TABLE;
112
113
114 END;
115
```


CLK
RESET/
ADC0_[15..0]
RD_CH0
WR_CH0
Clear_Buffer
INPUT
VCC
cir_buffer
clk
reset/
chan
val_in[15..0]
rd_mem
wr_mem
clr_buf
buf_ptr[15..0]
val_out[15..0]
buffer_empty
rd/
wr/
busy
error
q1
q2
q3
inst
PTR0_[15..0]
D0_[15..0]
BUFFER_MT0
RD0/
WR0/
BUSY0
GND
VCC
ADC1_[15..0]
RD_CH1
WR_CH1
inst2
PTR1_[15..0]
D1_[15..0]
BUFFER_MT1
RD1/
WR1/
BUSY1
OR2
inst4
A[15..0]
inst5
D[15..0]
AND2
inst6
RD/
inst7
WR/
inst8
BUSY
OUTPUT
A[15..0]
D[15..0]
RD/
WR/
BUSY
BUFFER_MT0
BUFFER_MT1

```
SUBDESIGN cir_buffer
(
   clk, reset/, chan, val_in[15..0], rd_mem, wr_mem, clr_buf : INPUT;

   buf_ptr[15..0], val_out[15..0]       : OUTPUT;
   buffer_empty, rd/, wr/, busy : OUTPUT;
   error, q1,q2,q3 :OUTPUT;

)

VARIABLE
   sm :   MACHINE
          OF BITS (q1, q2, q3)
          WITH STATES (
            idle   = B"000",
            set_rd = B"001",
            read   = B"011",
            write  = B"010",
            wr_done= B"110",
            rd_done= B"111",
            err1   = B"100",
            err2   = B"101"
            );


% Circular Buffer Variables %
 in_ptr[14..0]   : DFFE;
 nx_ptr[14..0]   : DFFE;  % the nx_ptr helps detect wrapping of other pointers %
 out_ptr[14..0]  : DFFE;

 ptr_reset/ : NODE;
 incr_inp, incr_outp : NODE;

 wrap  : NODE;

BEGIN
    sm.clk = clk;
    sm.reset = !reset/;

   TABLE
    % current     current        next                 current                          %
    % state       inputs         state                outputs                          %

      sm      , wr_mem , rd_mem   =>   sm    , incr_inp, incr_outp, rd/, wr/,  busy  ,  error ;

      idle    ,   0    ,   0      => idle    ,    0    ,    0     , 1  , 1  ,    0   ,    0   ;
      idle    ,   1    ,   x      => write   ,    0    ,    0     , 1  , 1  ,    0   ,    0   ;
      idle    ,   0    ,   1      => set_rd  ,    0    ,    0     , 1  , 1  ,    0   ,    0   ;

      set_rd  ,   x    ,   x      => read    ,    0    ,    0     , 0  , 1  ,    1   ,    0   ;

      read    ,   x    ,   x      => rd_done ,    0    ,    1     , 0  , 1  ,    1   ,    0   ;

      write   ,   x    ,   x      => wr_done ,    1    ,    0     , 1  , 0  ,    1   ,    0   ;

      rd_done ,   x    ,   1      => rd_done ,    0    ,    0     , 1  , 1  ,    1   ,    0   ;
      rd_done ,   x    ,   0      => idle    ,    0    ,    0     , 1  , 1  ,    1   ,    0   ;

      wr_done ,   1    ,   x      => wr_done ,    0    ,    0     , 1  , 1  ,    1   ,    0   ;
      wr_done ,   0    ,   x      => idle    ,    0    ,    0     , 1  , 1  ,    1   ,    0   ;

      err1    ,   x    ,   x      => idle    ,    0    ,    0     , 1  , 1  ,    1   ,    1   ;
      err2    ,   x    ,   x      => idle    ,    0    ,    0     , 1  , 1  ,    1   ,    1   ;
```

```
65    END TABLE;
66
67
68     %=================================================================%
69     IF    (!wr/)  THEN buf_ptr[14..0] = in_ptr[14..0].q ; buf_ptr[15] = chan;
70     ELSIF (!rd/)  THEN buf_ptr[14..0] = out_ptr[14..0].q; buf_ptr[15] = chan;
71     ELSE              buf_ptr[] = GND;
72     END IF;
73
74     val_out[] = val_in[] & (!wr/); % val_out[] = GND if wr_mem/ is not active %
75
76     %=================================================================%
77     % Circular Buffer Pointers ==================================== %
78     ptr_reset/ = !(!reset/ # clr_buf);
79
80     % Input Pointer channel %
81     in_ptr[].clk  = clk;
82     in_ptr[].clrn = ptr_reset/;
83     in_ptr[].ena  = incr_inp;
84
85     in_ptr[].d = in_ptr[].q +1;
86     %-----------------------------------------%
87     % next In Pointer channel %
88     nx_ptr[].clk       = clk;
89     nx_ptr[14..1].clrn = ptr_reset/;
90     nx_ptr[0].prn      = ptr_reset/;
91     nx_ptr[].ena       = incr_inp;
92
93     nx_ptr[].d = nx_ptr[].q + 1;
94
95     IF(nx_ptr[] == out_ptr[]) THEN % The nx_ptr warns of wrapping in_ptr past out_ptr %
96         wrap = VCC; % ACTIVE %    % A flag is set to move the out pointer with the in %
97     ELSE
98         wrap = GND; % NOT ACTIVE %
99     END IF;
100    %-----------------------------------------%
101    % Output Pointer channel %
102    out_ptr[].clk  = clk;
103    out_ptr[].clrn = ptr_reset/;
104    out_ptr[].ena  = (incr_outp # (incr_inp & wrap));  % two ways to advance this counter %
105
106    IF (out_ptr[] == in_ptr[]) THEN
107          out_ptr[].d = out_ptr[].q;
108          buffer_empty = VCC;
109    ELSE
110          out_ptr[].d = out_ptr[].q +1;
111          buffer_empty = GND;
112    END IF;
113
114 END;
115
```