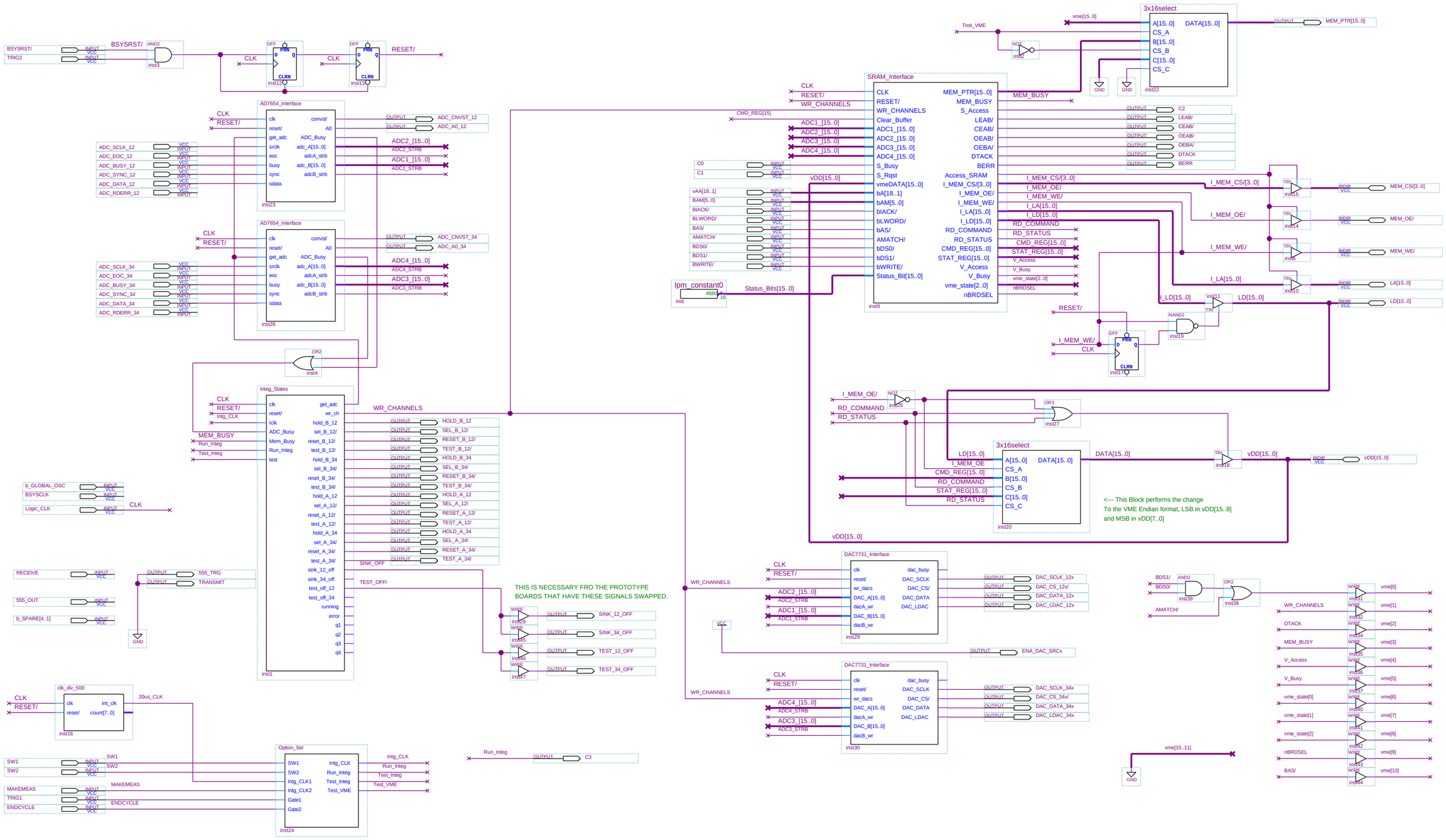
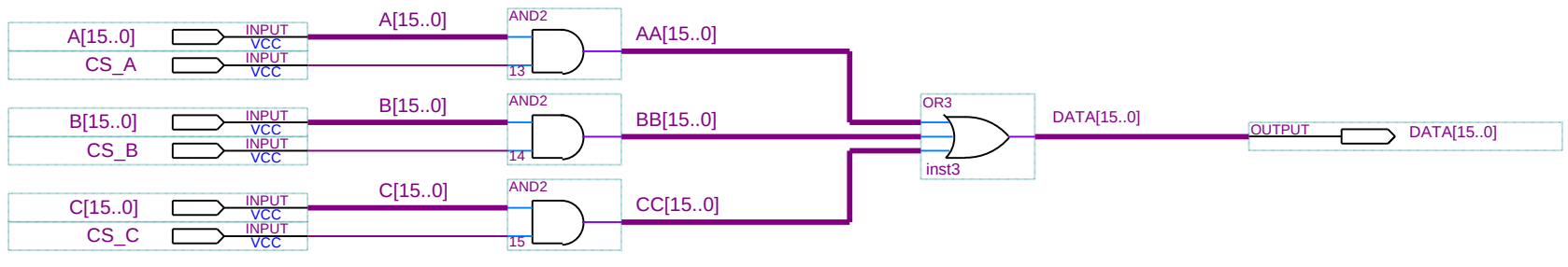

BSYSRST/
TRIG2
AND2
inst3
DFF
inst12
inst13
CLK
RESET/
AD7654_Interface
inst23
inst26
ADC_SCLK_12
ADC_EOC_12
ADC_BUSY_12
ADC_SYNC_12
ADC_DATA_12
ADC_RDERR_12
ADC_SCLK_34
ADC_EOC_34
ADC_BUSY_34
ADC_SYNC_34
ADC_DATA_34
ADC_RDERR_34
ADC_CNVST_12
ADC_A0_12
ADC_CNVST_34
ADC_A0_34
ADC2_[15..0]
ADC2_STRB
ADC1_[15..0]
ADC1_STRB
ADC4_[15..0]
ADC4_STRB
ADC3_[15..0]
ADC3_STRB
OR2
inst4
Integ_States
inst1
Intg_CLK
MEM_BUSY
Run_Integ
Test_Integ
WR_CHANNELS
SINK_OFF
TEST_OFF/
b_GLOBAL_OSC
BSYSCLK
Logic_CLK
RECEIVE
555_OUT
b_SPARE[4..1]
555_TRG
TRANSMIT
GND
THIS IS NECESSARY FRO THE PROTOTYPE
BOARDS THAT HAVE THESE SIGNALS SWAPPED.
SINK_12_OFF
SINK_34_OFF
TEST_12_OFF
TEST_34_OFF
clk_div_500
inst16
20us_CLK
SW1
SW2
MAKEMEAS
TRIG1
ENDCYCLE
Option_Sel
inst24
Test_VME
Run_Integ
C3
SRAM_Interface
inst9
Clear_Buffer
ADC1_[15..0]
ADC2_[15..0]
ADC3_[15..0]
ADC4_[15..0]
S_Busy
S_Rqst
vmeDATA[15..0]
bA[18..1]
bAM[5..0]
bIACK/
bLWORD/
bAS/
AMATCH/
bDS0/
bDS1/
bWRITE/
Status_Bit[15..0]
MEM_PTR[15..0]
MEM_BUSY
S_Access
LEAB/
CEAB/
OEAB/
OEBA/
DTACK
BERR
Access_SRAM
I_MEM_CS/[3..0]
I_MEM_OE/
I_MEM_WE/
I_LA[15..0]
I_LD[15..0]
RD_COMMAND
RD_STATUS
CMD_REG[15..0]
STAT_REG[15..0]
V_Access
V_Busy
vme_state[2..0]
nBRDSEL
lpm_constant0
Status_Bits[15..0]
vDD[15..0]
3x16select
inst22
inst20
A[15..0]
B[15..0]
C[15..0]
DATA[15..0]
CS_A
CS_B
CS_C
Test_VME
vme[15..0]
MEM_PTR[15..0]
C2
MEM_CS/[3..0]
MEM_OE/
MEM_WE/
LA[15..0]
LD[15..0]
NAND2
inst19
inst11
inst17
inst25
inst27
inst18
<--- This Block performs the change
To the VME Endian format, LSB in vDD[15..8]
and MSB in vDD[7..0]
DAC7731_Interface
inst29
inst30
DAC_SCLK_12x
DAC_CS_12x/
DAC_DATA_12x
DAC_LDAC_12x
ENA_DAC_SRCx
DAC_SCLK_34x
DAC_CS_34x/
DAC_DATA_34x
DAC_LDAC_34x
BDS1/
BDS0/
AMATCH/
AND2
inst39
OR2
inst38
vme[15..11]
vme[0]
vme[1]
vme[2]
vme[3]
vme[4]
vme[5]
vme[6]
vme[7]
vme[8]
vme[9]
vme[10]


A[15..0]
INPUT
VCC
CS_A
INPUT
VCC
B[15..0]
INPUT
VCC
CS_B
INPUT
VCC
C[15..0]
INPUT
VCC
CS_C
INPUT
VCC
A[15..0]
B[15..0]
C[15..0]
AND2
13
AND2
14
AND2
15
AA[15..0]
BB[15..0]
CC[15..0]
OR3
inst3
DATA[15..0]
OUTPUT
DATA[15..0]

```
SUBDESIGN mem_arbitration
(
   clk, reset/ : INPUT;
   I_Busy, I_Rqst, S_Busy, S_Rqst, V_Busy, V_Rqst : INPUT;

   I_Access, S_Access, V_Access : OUTPUT;
   q1,q2,q3 : OUTPUT;
)

VARIABLE
sm :   MACHINE
          OF BITS (q1, q2, q3)
          WITH STATES (
            idle     = B"000",
            I_setup  = B"001",
            S_setup  = B"010",
            V_setup  = B"011",
            done     = B"100",
            I_active = B"101",
            S_active = B"110",
            V_active = B"111"
            );

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

    % ==================================================================================%
    % Explanation of the Protocol                                                       %
    % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~                                                      %
    %  1) Logical devices wanting to read or write from the SRAM memory must set their  %
    %     Request Line (Rqst) active and wait for their Bus Access Line (Access) to     %
    %     become active before beginning their memory access.                           %
    %                                                                                   %
    %  2) Before releasing their Request Line, their Busy Line should be set active.    %
    %     Note that the Request Line may remain active until the memory access has been %
    %     completed.  Then the Request line must be released before or simultaneous with %
    %     the Busy Line going inactive.                                                 %
    %                                                                                   %
    %  3) No other logical device will be granted access to the memory while the device %
    %     that was granted access, holds its Busy Line active.                          %
    %                                                                                   %
    %  4) If a higher priority device sets its Request Line active while a lower priority %
    %     device is Busy, the Bus Access Line for the lower priority device will go     %
    %     inactive indicating that the lower priority device should suspend its memory  %
    %     access and set its Busy Line inactive.  The lower priority device should keep %
    %     its Request Line active if it had not completed its memory access.            %
    %                                                                                   %
    % Sequence of events for a logical devices memory access                            %
    % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                   %
    %  1) Set Request Line active.                                                       %
    %  2) Wait for Bus Access Line to go active.                                         %
    %  3) Set Busy Line active and do memory read and/or write.                          %
    %  4) If while busy accessing the memory the Bus Access Line goes inactive           %
    %     suspend further memory accesses, set the Busy line inactive, and leave        %
    %     the Request Line active if the memory access was not completed.               %
    %  5) When the Bus Access Line goes active once again set the Busy Line active       %
    %     again and continue the memory access.                                         %
    %  6) Once the memory access is complete set both the Busy Line and the Request     %
    %     Line inactive.                                                                %
    %                                                                                   %
    %   Note:  A devices memory access is considered to be any number of reads and/or   %
    %          writes.                                                                  %

```

```
65
66    TABLE
67    %  current                    current                          next                 current             %
68    %  state                      inputs                          state                 outputs             %
69
70       sm      ,  I_Busy, S_Busy, V_Busy, I_Rqst, S_Rqst, V_Rqst =>   sm    , I_Access, S_Access, V_Access   ;
71
72       idle    ,    x   ,   x   ,   x   ,   0   ,   0   ,   0    => idle    ,   0     ,   0     ,    0       ;
73       idle    ,    x   ,   x   ,   x   ,   0   ,   0   ,   1    => V_setup ,   0     ,   0     ,    0       ;
74       idle    ,    x   ,   x   ,   x   ,   0   ,   1   ,   x    => S_setup ,   0     ,   0     ,    0       ;
75       idle    ,    x   ,   x   ,   x   ,   1   ,   x   ,   x    => I_setup ,   0     ,   0     ,    0       ;
76
77       I_setup ,    0   ,   x   ,   x   ,   1   ,   x   ,   x    => I_setup ,   1     ,   0     ,    0       ;
78       I_setup ,    0   ,   x   ,   x   ,   0   ,   x   ,   x    => idle    ,   0     ,   0     ,    0       ;
79       I_setup ,    1   ,   x   ,   x   ,   x   ,   x   ,   x    => I_active,   1     ,   0     ,    0       ;
80
81       I_active,    1   ,   x   ,   x   ,   x   ,   x   ,   x    => I_active,   1     ,   0     ,    0       ;
82       I_active,    0   ,   x   ,   x   ,   0   ,   0   ,   0    => idle    ,   0     ,   0     ,    0       ;
83       I_active,    0   ,   x   ,   x   ,   0   ,   0   ,   1    => V_setup ,   0     ,   0     ,    0       ;
84       I_active,    0   ,   x   ,   x   ,   0   ,   1   ,   x    => S_setup ,   0     ,   0     ,    0       ;
85       I_active,    0   ,   x   ,   x   ,   1   ,   x   ,   x    => I_setup ,   0     ,   0     ,    0       ;
86
87       S_setup ,    x   ,   0   ,   x   ,   x   ,   1   ,   x    => S_setup ,   0     ,   1     ,    0       ;
88       S_setup ,    x   ,   0   ,   x   ,   x   ,   0   ,   x    => idle    ,   0     ,   0     ,    0       ;
89       S_setup ,    x   ,   1   ,   x   ,   x   ,   x   ,   x    => S_active,   0     ,   1     ,    0       ;
90
91       S_active,    x   ,   1   ,   x   ,   0   ,   x   ,   x    => S_active,   0     ,   1     ,    0       ;
92       S_active,    x   ,   1   ,   x   ,   1   ,   x   ,   x    => S_active,   0     ,   0     ,    0       ;
93       S_active,    x   ,   0   ,   x   ,   0   ,   0   ,   0    => idle    ,   0     ,   0     ,    0       ;
94       S_active,    x   ,   0   ,   x   ,   0   ,   0   ,   1    => V_setup ,   0     ,   0     ,    0       ;
95       S_active,    x   ,   0   ,   x   ,   0   ,   1   ,   x    => S_setup ,   0     ,   0     ,    0       ;
96       S_active,    x   ,   0   ,   x   ,   1   ,   x   ,   x    => I_setup ,   0     ,   0     ,    0       ;
97
98       V_setup ,    x   ,   x   ,   0   ,   x   ,   x   ,   1    => V_setup ,   0     ,   0     ,    1       ;
99       V_setup ,    x   ,   x   ,   0   ,   x   ,   x   ,   0    => idle    ,   0     ,   0     ,    0       ;
100      V_setup ,    x   ,   x   ,   1   ,   x   ,   x   ,   x    => V_active,   0     ,   0     ,    1       ;
101
102      V_active,    x   ,   x   ,   1   ,   0   ,   0   ,   x    => V_active,   0     ,   0     ,    1       ;
103      V_active,    x   ,   x   ,   1   ,   0   ,   1   ,   x    => V_active,   0     ,   0     ,    0       ;
104      V_active,    x   ,   x   ,   1   ,   1   ,   x   ,   x    => V_active,   0     ,   0     ,    0       ;
105      V_active,    x   ,   x   ,   0   ,   0   ,   0   ,   0    => idle    ,   0     ,   0     ,    0       ;
106      V_active,    x   ,   x   ,   0   ,   0   ,   0   ,   1    => V_setup ,   0     ,   0     ,    0       ;
107      V_active,    x   ,   x   ,   0   ,   0   ,   1   ,   x    => S_setup ,   0     ,   0     ,    0       ;
108      V_active,    x   ,   x   ,   0   ,   1   ,   x   ,   x    => I_setup ,   0     ,   0     ,    0       ;
109
110      done    ,    x   ,   x   ,   x   ,   x   ,   x   ,   x    => idle    ,   0     ,   0     ,    0       ;
111      % ^-- This state is currently unused %
112   END TABLE;
113
114 END;
115
```

```
CONSTANT SETUP_DELAY_COUNT = B"0101";

SUBDESIGN Intg_vme
(
    clk, reset/, A[18..1], V_Access : INPUT;
    AM[5..0], IACK/, LWORD/, AS/, AMATCH/ : INPUT;
    DS0/, DS1/, WRITE/  : INPUT;

    vmePTR[15..0], V_Busy, V_Rqst : OUTPUT;
    LEAB/, CEAB/, OEAB/, OEBA/ : OUTPUT;
    RD_GATE/, WR_GATE/, DTACK, BERR, STATE[2..0], nDS_SUM : OUTPUT;

      RD_CirBuf1,   RD_CirBuf2,   RD_CirBuf3,   RD_CirBuf4 : OUTPUT;
     RD_Command, WR_Command, RD_Status, WR_ENA : OUTPUT;
    vme_MEM_CS/1, vme_MEM_CS/2, vme_MEM_CS/3, vme_MEM_CS/4 : OUTPUT;

    RD_SRAM/, WR_SRAM/, vmeSRAM_ACCESS, nBRDSEL : OUTPUT;
)

VARIABLE
    nBRDSEL         : LCELL;
    BLOCK           : LCELL;

    sync_DS[1..0]   : DFF;
    nDS             : LCELL;
    sync_write      : DFF;
    vmeWRITE/       : LCELL;
    sync_AS/        : DFF;
    vmeAS/          : LCELL;

    VME_SM    : MACHINE
                OF BITS (STATE[2..0])
                WITH STATES (
                    IDLE           =B"000",
                    BRD_SELECTED   =B"001",
                    DATA_SETUP_DLY =B"101",
                    STROBE_WR      =B"011",
                    DTACKEN        =B"100",
                    XFER_DONE      =B"010",
                    BUS_ERROR      =B"110",
                    XSTATE1        =B"111" );
    CNT[3..0]   : DFFE;
    SETUP       : LCELL;
    BUSY        : LCELL;

    ALATCH[18..1] : DFFE;
    CHAN[1..0]    : NODE;

    r1  : DFF; % Helps synchronize combinatorial logic for the RD_GATE/ signal %


BEGIN
    % Synchronize the VME signals to the local logic clock %
    sync_DS[].clk  = clk;
    sync_DS[].prn  = reset/;
    sync_DS1.d = DS1/;
    sync_DS0.d = DS0/;

    nDS   = sync_DS1.q & sync_DS0.q;
    nDS_SUM = nDS;

    sync_write.clk  = clk;
    sync_write.prn  = reset/;
    sync_write.d = WRITE/;
```

```
65
66     vmeWRITE/ = sync_write.q;
67
68     sync_AS/.clk  = clk;
69     sync_AS/.prn  = reset/;
70     sync_AS/.d = AS/;
71
72     vmeAS/ = sync_AS/.q;
73
74     vmePTR[15..0] = A[16..1];
75
76 %==============================================================================%
77 % VME Handshaking Logic                                                        %
78
79     VME_SM.clk = clk;
80     VME_SM.reset = !reset/;
81
82     CNT[].clk = clk;
83     CNT[].clrn = !(IDLE#XFER_DONE);
84     CNT[].ena  = !SETUP;
85
86     IF DATA_SETUP_DLY THEN
87        CNT[].d = CNT[].q + 1;
88     ELSE
89        CNT[].d = CNT[].q;
90     END IF;
91
92     IF (CNT[].q == SETUP_DELAY_COUNT) THEN
93        SETUP = VCC;
94     ELSE
95        SETUP = GND;
96     END IF;
97     % STANDARD ADDRESS MODIFIER DECODE ============================================%
98     nBRDSEL = !(
99                 (IACK/ & LWORD/ & AM5 & AM4 & AM3 &  AM1 & !AM0 & !vmeAS/ & !AMATCH/)
100              # (IACK/ & LWORD/ & AM5 & AM4 & AM3 & !AM1 &  AM0 & !vmeAS/ & !AMATCH/)
101              # (IACK/ & LWORD/ & AM5 & AM4 & AM3 &  AM1 &  AM0 & !vmeAS/ & !AMATCH/));
102
103    BLOCK = !nBRDSEL & AM1 & AM0; % Block transfer indication %
104
105    % Memory bus arbitration variables ============================================%
106    BUSY = !V_Access;  % This says Hold-off, the memory is busy, VME doe not have access%
107    V_Rqst = (!nBRDSEL & !nDS);
108    V_Busy = (DATA_SETUP_DLY # STROBE_WR # DTACKEN);
109
110    % OUTPUTS ======================================================================%
111    DTACK = DTACKEN;
112    BERR  = BUS_ERROR;
113    WR_GATE/ = !DATA_SETUP_DLY;
114
115    r1.clk  = clk;
116    r1.prn = reset/;
117    r1.d    = !(DATA_SETUP_DLY # STROBE_WR # DTACKEN);
118    RD_GATE/ = !(DATA_SETUP_DLY # STROBE_WR # DTACKEN) & r1.q;
119
120 TABLE % VME STATE MACHINE ==========================================================%
121 reset/,nBRDSEL, BLOCK,   nDS, SETUP, BUSY, VME_SM => VME_SM;
122  % Reset %
123    0,     X,        X,     X,   X,     X,   IDLE => IDLE;
124  % IDLE  %
125    1,     1,        X,     X,   X,     X,   IDLE => IDLE;
126    1,     0,        X,     X,   X,     X,   IDLE => BRD_SELECTED;  % got a board select %
127  % BOARD SELECTED %
128    1,     1,        X,     X,   X,     X,   BRD_SELECTED => IDLE;  % lost the board select %
```

```
  1,     0,        X,     1,   X,     X,   BRD_SELECTED => BRD_SELECTED;  % wait for DS to go low %
  1,     0,        X,     0,   X,     1,   BRD_SELECTED => BRD_SELECTED;  % HOLDOFF IS ACTIVE %
  1,     0,        X,     0,   X,     0,   BRD_SELECTED => DATA_SETUP_DLY;  % received DS %
% DATA_SETUP_DLY %
  1,     X,        X,     1,   X,     X,   DATA_SETUP_DLY => BUS_ERROR;  % Lost the data strobe -> BUS ERR %
  1,     X,        X,     0,   0,     X,   DATA_SETUP_DLY => DATA_SETUP_DLY;  % wait for 200ns %
  1,     X,        X,     0,   1,     X,   DATA_SETUP_DLY => STROBE_WR;  % setup over %
% STROBE_WR %
  X,     X,        X,     X,   X,     X,   STROBE_WR => DTACKEN; % Enable writes for one clock %
% DTACK ENABLE %
  1,     X,        X,     0,   X,     X,   DTACKEN => DTACKEN;  % wait for DS to go inactive %
  1,     X,        X,     1,   X,     X,   DTACKEN => XFER_DONE;  % transfer is completed %
% TRANSFER COMPLETED %
  1,     0,        X,     1,   X,     X,   XFER_DONE => XFER_DONE;  % wait for nDS or AS/ to change %
  1,     0,        0,     X,   X,     X,   XFER_DONE => XFER_DONE;  % wait for nDS or AS/ to change %
  1,     0,        1,     0,   X,     1,   XFER_DONE => XFER_DONE; % HOLDOFF IS ACTIVE %
  1,     0,        1,     0,   X,     0,   XFER_DONE => DATA_SETUP_DLY;  % continue the block transfer %
  1,     1,        X,     X,   X,     X,   XFER_DONE => IDLE;  % all done -> IDLE %
% BUS ERROR %
  1,     X,        X,     0,   X,     X,   BUS_ERROR => IDLE;  % Go Idle %
% ILLEGAL STATES %
  1,     X,        X,     X,   X,     X,   XSTATE1 => IDLE;  % Go Idle %
END TABLE; %==============================================================================%

%==============================================================================%
% VME Address Decoding Logic                                                   %

    ALATCH[].clk = clk;
    ALATCH[].clrn = reset/;
    ALATCH[].d   = A[18..1];
    ALATCH[].ena = RD_GATE/;

    CHAN[1..0] = ALATCH[18..17].q;

    RD_CirBuf1 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 0) & (ALATCH[16..1].q == H"FFFF");
    RD_CirBuf2 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 1) & (ALATCH[16..1].q == H"FFFF");
    RD_CirBuf3 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 2) & (ALATCH[16..1].q == H"FFFF");
    RD_CirBuf4 = !RD_GATE/ & (vmeWRITE/ == VCC) & (CHAN[] == 3) & (ALATCH[16..1].q == H"FFFF");

    RD_Command  = !RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q == H"FFFE");
    WR_Command = (STROBE_WR & (vmeWRITE/ == GND) & (ALATCH[16..1].q == H"FFFE"));
    WR_ENA      =  STROBE_WR & (vmeWRITE/ == GND);

    RD_Status   = !RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q == H"FFFD");

    vmeSRAM_ACCESS = (!RD_GATE/ & (A[16..1] < H"FFF0"));

    RD_SRAM/ = !(!RD_GATE/ & (vmeWRITE/ == VCC) & (ALATCH[16..1].q < H"FFF0"));

    WR_SRAM/ = !(!WR_GATE/ & (vmeWRITE/ == GND) & (ALATCH[16..1].q < H"FFF0"));

    vme_MEM_CS/1 = !(!RD_GATE/ & (CHAN[] == 0) & (ALATCH[16..1].q < H"FFF0"));
    vme_MEM_CS/2 = !(!RD_GATE/ & (CHAN[] == 1) & (ALATCH[16..1].q < H"FFF0"));
    vme_MEM_CS/3 = !(!RD_GATE/ & (CHAN[] == 2) & (ALATCH[16..1].q < H"FFF0"));
    vme_MEM_CS/4 = !(!RD_GATE/ & (CHAN[] == 3) & (ALATCH[16..1].q < H"FFF0"));

    LEAB/ = !(!RD_GATE/ & (vmeWRITE/ == VCC));  % Doing a vme read %
    CEAB/ = LEAB/;
    OEAB/ = LEAB/;

    OEBA/ = !(!RD_GATE/ & (vmeWRITE/ == GND));  % Doing a vme write %


END;
```

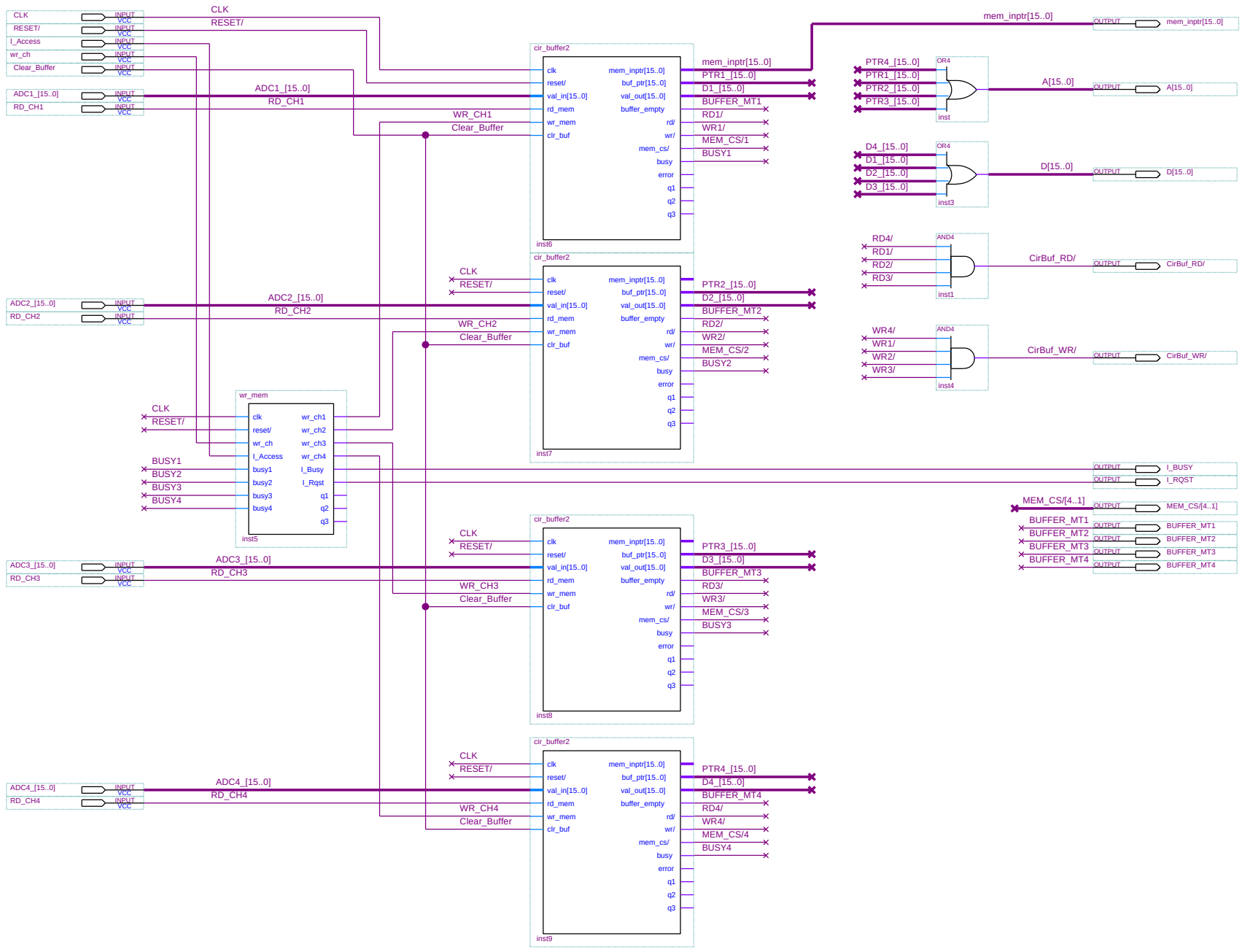
```
SUBDESIGN cir_buffer2
(
   clk, reset/, val_in[15..0], rd_mem, wr_mem, clr_buf : INPUT;

   mem_inptr[15..0], buf_ptr[15..0], val_out[15..0]    : OUTPUT;
   buffer_empty, rd/, wr/, mem_cs/, 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[15..0]   : DFFE;
 nx_ptr[15..0]   : DFFE;  % the nx_ptr helps detect wrapping of other pointers %
 out_ptr[15..0]  : DFFE;

 ptr_reset/ : NODE;
 incr_inp, incr_outp : NODE;

 wrap    : NODE;
 wr_ena/ : NODE; %used to gate the buffer pointer during a write %
 sync[6..1] : DFF; % used to avoid glitches from state machine combinatorial logic;%

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

    sync[].clk  = clk;
    sync[].clrn = reset/;

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

      sm      , wr_mem , rd_mem   =>   sm    ,  error ;

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

      set_rd  ,   x    ,   x      => read    ,    0   ;

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

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

      rd_done ,   x    ,   x      => idle    ,    0   ;

```

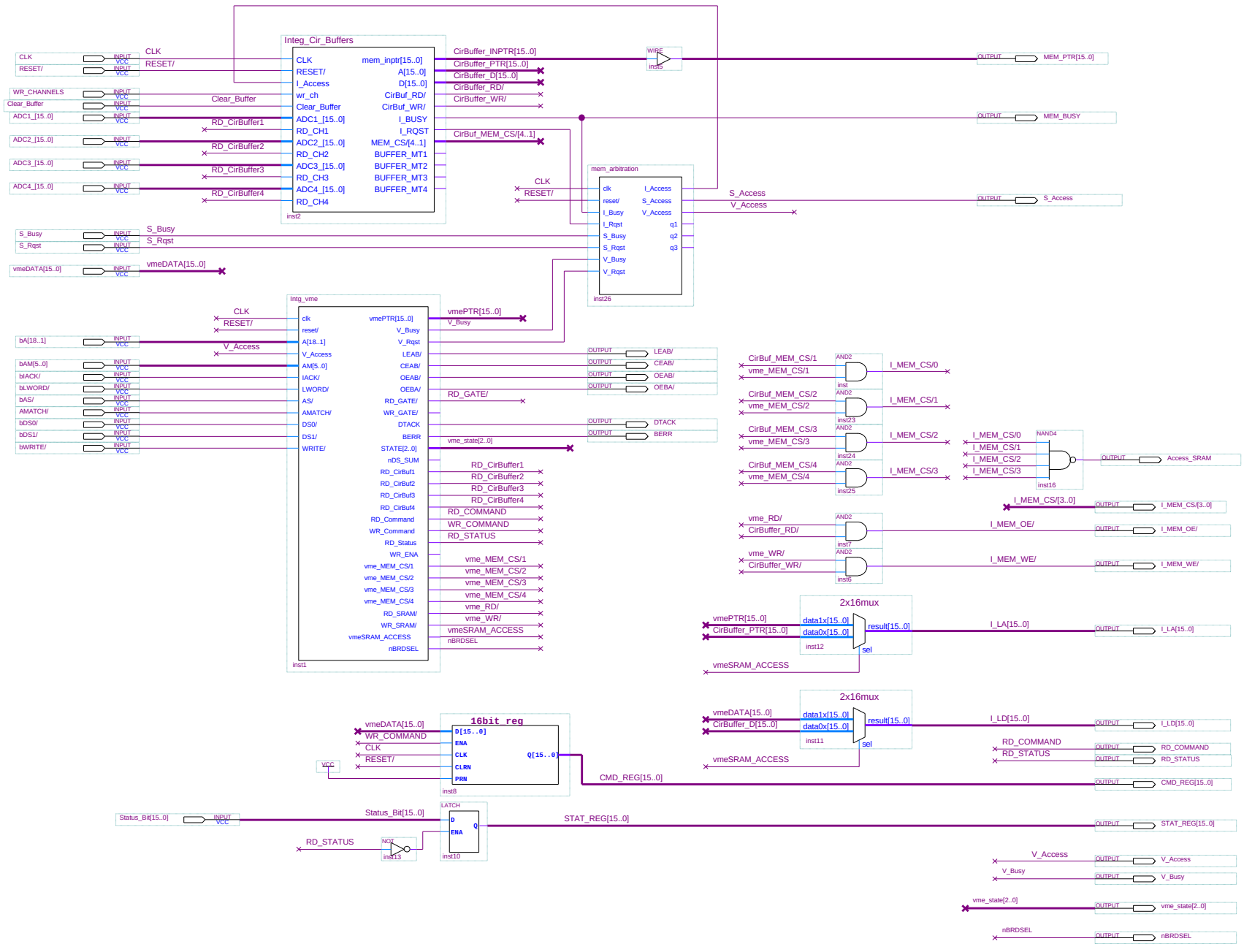
```
65        wr_done ,   x    ,   x      => idle    ,    0   ;
66
67        err1    ,   x    ,   x      => idle    ,    1   ;
68        err2    ,   x    ,   x      => idle    ,    1   ;
69
70     END TABLE;
71
72      sync1.d = (set_rd # read # rd_done);
73      sync2.d = (write);
74      sync3.d = (write # wr_done);
75      sync4.d = wr_done;
76      sync5.d = rd_done;
77      sync6.d = idle;
78
79      rd/     = !sync1.q;
80      wr/     = !sync2.q;
81      wr_ena/ = !sync3.q;
82      incr_inp  = sync4.q;
83      incr_outp = sync5.q;
84      busy      = !sync6.q;
85
86      mem_cs/ = !(!rd/ # !wr_ena/);
87
88      %=================================================================%
89      IF    ( !wr_ena/)  THEN buf_ptr[15..0] = in_ptr[15..0].q;
90      ELSIF (!rd/)       THEN buf_ptr[15..0] = out_ptr[15..0].q;
91      ELSE                    buf_ptr[] = GND;
92      END IF;
93
94      val_out[] = val_in[] & (!wr_ena/); % val_out[] = GND if wr_mem/ is not active %
95
96      mem_inptr[15..0] = in_ptr[15..0].q;
97      %=================================================================%
98      % Circular Buffer Pointers ===================================== %
99      ptr_reset/ = !(!reset/ # clr_buf);
100
101     % Input Pointer channel %
102     in_ptr[].clk  = clk;
103     in_ptr[].clrn = ptr_reset/;
104     in_ptr[].ena  = incr_inp;
105
106     in_ptr[].d = in_ptr[].q +1;
107     %----------------------------------------%
108     % next In Pointer channel %
109     nx_ptr[].clk       = clk;
110     nx_ptr[14..1].clrn = ptr_reset/;
111     nx_ptr[0].prn      = ptr_reset/;
112     nx_ptr[].ena       = incr_inp;
113
114     nx_ptr[].d = nx_ptr[].q + 1;
115
116     IF(nx_ptr[] == out_ptr[]) THEN % The nx_ptr warns of wrapping in_ptr past out_ptr %
117         wrap = VCC; % ACTIVE %    % A flag is set to move the out pointer with the in %
118     ELSE
119         wrap = GND; % NOT ACTIVE %
120     END IF;
121     %----------------------------------------%
122     % Output Pointer channel %
123     out_ptr[].clk  = clk;
124     out_ptr[].clrn = ptr_reset/;
125     out_ptr[].ena  = (incr_outp # (incr_inp & wrap));  % two ways to advance this counter %
126
127     IF (out_ptr[] == in_ptr[]) THEN
128           out_ptr[].d = out_ptr[].q;
```

```
129          buffer_empty = VCC;
130     ELSE
131          out_ptr[].d = out_ptr[].q +1;
132          buffer_empty = GND;
133     END IF;
134
135 END;
136
```

```
SUBDESIGN wr_mem
(
   clk, reset/, wr_ch, I_Access : INPUT;
   busy1, busy2, busy3, busy4 : INPUT;

   wr_ch1, wr_ch2, wr_ch3, wr_ch4, I_Busy, I_Rqst  : OUTPUT;
   q1,q2,q3 : OUTPUT;
)

VARIABLE
sm :   MACHINE
          OF BITS (q1, q2, q3, q4)
          WITH STATES (
            idle     = B"0000",
            mem_rqst = B"0001",
            wr_ch1s  = B"0010",
            wait1    = B"0011",
            wr_ch2s  = B"0100",
            wait2    = B"0101",
            wr_ch3s  = B"0110",
            wait3    = B"0111",
            wr_ch4s  = B"1000",
            wait4    = B"1001",
            err1     = B"1010",
            err2     = B"1011",
            err3     = B"1100",
            err4     = B"1101",
            err5     = B"1110",
            err6     = B"1111"
            );

    Sync_Busy : DFF; %Ensure the outgoing busy is resynced to the clock to avoid combinatorial loop issue%
    busy : NODE;

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

    Sync_Busy.clk  = clk;
    Sync_Busy.clrn = reset/;
    Sync_Busy.d    = busy;

    I_Busy = Sync_Busy.q;

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

      sm      ,  wr_ch , I_Access,  busy1, busy2, busy3, busy4 =>   sm    , busy , I_Rqst  ;

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

      mem_rqst,    x   ,    0    ,   x   ,   x  ,   x  ,   x   => mem_rqst,   0  ,   1     ;
      mem_rqst,    x   ,    1    ,   x   ,   x  ,   x  ,   x   => wr_ch1s ,   0  ,   1     ;

      wr_ch1s ,    x   ,    x    ,   0   ,   x  ,   x  ,   x   => wr_ch1s ,   1  ,   1     ;
      wr_ch1s ,    x   ,    x    ,   1   ,   x  ,   x  ,   x   => wait1   ,   1  ,   1     ;

      wait1   ,    x   ,    x    ,   1   ,   x  ,   x  ,   x   => wait1   ,   1  ,   0     ;
      wait1   ,    x   ,    x    ,   0   ,   x  ,   x  ,   x   => wr_ch2s ,   1  ,   0     ;

      wr_ch2s ,    x   ,    x    ,   x   ,   0  ,   x  ,   x   => wr_ch2s ,   1  ,   0     ;
      wr_ch2s ,    x   ,    x    ,   x   ,   1  ,   x  ,   x   => wait2   ,   1  ,   0     ;
```

```

      wait2   ,    x   ,    x    ,   x   ,   1  ,   x  ,   x   => wait2   ,   1  ,   0     ;
      wait2   ,    x   ,    x    ,   x   ,   0  ,   x  ,   x   => wr_ch3s ,   1  ,   0     ;

      wr_ch3s ,    x   ,    x    ,   x   ,   x  ,   0  ,   x   => wr_ch3s ,   1  ,   0     ;
      wr_ch3s ,    x   ,    x    ,   x   ,   x  ,   1  ,   x   => wait3   ,   1  ,   0     ;

      wait3   ,    x   ,    x    ,   x   ,   x  ,   1  ,   x   => wait3   ,   1  ,   0     ;
      wait3   ,    x   ,    x    ,   x   ,   x  ,   0  ,   x   => wr_ch4s ,   1  ,   0     ;

      wr_ch4s ,    x   ,    x    ,   x   ,   x  ,   x  ,   0   => wr_ch4s ,   1  ,   0     ;
      wr_ch4s ,    x   ,    x    ,   x   ,   x  ,   x  ,   1   => wait4   ,   1  ,   0     ;

      wait4   ,    x   ,    x    ,   x   ,   x  ,   x  ,   1   => wait4   ,   1  ,   0     ;
      wait4   ,    x   ,    x    ,   x   ,   x  ,   x  ,   0   => idle    ,   1  ,   0     ;

   END TABLE;

    wr_ch1 = wr_ch1s;
    wr_ch2 = wr_ch2s;
    wr_ch3 = wr_ch3s;
    wr_ch4 = wr_ch4s;

END;

```


CLK
RESET/
I_Access
wr_ch
Clear_Buffer
ADC1_[15..0]
RD_CH1
ADC2_[15..0]
RD_CH2
ADC3_[15..0]
RD_CH3
ADC4_[15..0]
RD_CH4
INPUT
VCC
wr_mem
clk
reset/
wr_ch
I_Access
busy1
busy2
busy3
busy4
wr_ch1
wr_ch2
wr_ch3
wr_ch4
I_Busy
I_Rqst
q1
q2
q3
inst5
BUSY1
BUSY2
BUSY3
BUSY4
cir_buffer2
val_in[15..0]
rd_mem
clr_buf
mem_inptr[15..0]
buf_ptr[15..0]
val_out[15..0]
buffer_empty
rd/
wr/
mem_cs/
busy
error
inst6
inst7
inst8
inst9
WR_CH1
WR_CH2
WR_CH3
WR_CH4
PTR1_[15..0]
D1_[15..0]
BUFFER_MT1
RD1/
WR1/
MEM_CS/1
BUSY1
PTR2_[15..0]
D2_[15..0]
BUFFER_MT2
RD2/
WR2/
MEM_CS/2
BUSY2
PTR3_[15..0]
D3_[15..0]
BUFFER_MT3
RD3/
WR3/
MEM_CS/3
BUSY3
PTR4_[15..0]
D4_[15..0]
BUFFER_MT4
RD4/
WR4/
MEM_CS/4
BUSY4
OR4
inst
inst3
AND4
inst1
inst4
A[15..0]
D[15..0]
CirBuf_RD/
CirBuf_WR/
OUTPUT
I_BUSY
I_RQST
MEM_CS/[4..1]
BUFFER_MT1
BUFFER_MT2
BUFFER_MT3
BUFFER_MT4


Integ_Cir_Buffers
CLK
RESET/
I_Access
wr_ch
Clear_Buffer
ADC1_[15..0]
RD_CH1
ADC2_[15..0]
RD_CH2
ADC3_[15..0]
RD_CH3
ADC4_[15..0]
RD_CH4
mem_inptr[15..0]
A[15..0]
D[15..0]
CirBuf_RD/
CirBuf_WR/
I_BUSY
I_RQST
MEM_CS/[4..1]
BUFFER_MT1
BUFFER_MT2
BUFFER_MT3
BUFFER_MT4
inst2
CirBuffer_INPTR[15..0]
CirBuffer_PTR[15..0]
CirBuffer_D[15..0]
CirBuffer_RD/
CirBuffer_WR/
CirBuf_MEM_CS/[4..1]
WR_CHANNELS
RD_CirBuffer1
RD_CirBuffer2
RD_CirBuffer3
RD_CirBuffer4
S_Busy
S_Rqst
vmeDATA[15..0]
mem_arbitration
reset/
V_Busy
V_Rqst
q1
q2
q3
inst26
S_Access
V_Access
MEM_PTR[15..0]
MEM_BUSY
Intg_vme
bA[18..1]
bAM[5..0]
bIACK/
bLWORD/
bAS/
AMATCH/
bDS0/
bDS1/
bWRITE/
vmePTR[15..0]
LEAB/
CEAB/
OEAB/
OEBA/
RD_GATE/
WR_GATE/
DTACK
BERR
vme_state[2..0]
STATE[2..0]
nDS_SUM
RD_CirBuf1
RD_CirBuf2
RD_CirBuf3
RD_CirBuf4
RD_Command
WR_Command
RD_Status
WR_ENA
vme_MEM_CS/1
vme_MEM_CS/2
vme_MEM_CS/3
vme_MEM_CS/4
RD_SRAM/
WR_SRAM/
vmeSRAM_ACCESS
nBRDSEL
RD_COMMAND
WR_COMMAND
RD_STATUS
inst1
CirBuf_MEM_CS/1
vme_MEM_CS/1
CirBuf_MEM_CS/2
vme_MEM_CS/2
CirBuf_MEM_CS/3
vme_MEM_CS/3
CirBuf_MEM_CS/4
vme_MEM_CS/4
I_MEM_CS/0
I_MEM_CS/1
I_MEM_CS/2
I_MEM_CS/3
NAND4
inst16
Access_SRAM
I_MEM_CS/[3..0]
vme_RD/
CirBuffer_RD/
vme_WR/
CirBuffer_WR/
I_MEM_OE/
I_MEM_WE/
2x16mux
data1x[15..0]
data0x[15..0]
result[15..0]
sel
inst12
I_LA[15..0]
inst11
I_LD[15..0]
16bit reg
D[15..0]
ENA
CLK
CLRN
PRN
Q[15..0]
inst8
VCC
CMD_REG[15..0]
Status_Bit[15..0]
LATCH
D
ENA
Q
inst10
NOT
inst13
STAT_REG[15..0]
V_Access
V_Busy
vme_state[2..0]
nBRDSEL

```
SUBDESIGN Option_Sel
(
    SW1, SW2                : INPUT;
    Intg_CLK1, Intg_CLK2, Gate1, Gate2 :INPUT;

    Intg_CLK, Run_Integ, Test_Integ, Test_VME : OUTPUT;
)

%VARIABLE%

BEGIN

    IF    (SW1 == 0 & SW2 == 0) THEN
        Intg_CLK  = Intg_CLK1;
        Run_Integ = VCC;
        Test_Integ= GND;
        Test_VME  = GND;
    ELSIF (SW1 == 1 & SW2 == 0) THEN
        Intg_CLK  = Intg_CLK1;
        Run_Integ = Gate1;
        Test_Integ= GND;
        Test_VME  = GND;
    ELSIF (SW1 == 0 & SW2 == 1) THEN
        Intg_CLK  = Intg_CLK1;
        Run_Integ = VCC;
        Test_Integ= GND;
        Test_VME  = VCC;
    ELSIF (SW1 == 1 & SW2 == 1) THEN
        Intg_CLK  = Intg_CLK1;
        Run_Integ = VCC;
        Test_Integ= VCC; % TEST Mode %
        Test_VME  = GND;
    END IF;

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 Integ_States
(
   clk, reset/, Iclk, ADC_Busy, Mem_Busy, Run_Integ : INPUT;
   test : INPUT;

   get_adc, wr_ch : OUTPUT;
   hold_B_12, sel_B_12/, reset_B_12/, test_B_12/ : OUTPUT;
   hold_B_34, sel_B_34/, reset_B_34/, test_B_34/ : OUTPUT;
   hold_A_12, sel_A_12/, reset_A_12/, test_A_12/ : OUTPUT;
   hold_A_34, sel_A_34/, reset_A_34/, test_A_34/ : OUTPUT;

   sink_12_off, sink_34_off, test_off_12, test_off_34 : 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_wrmem= B"0100",
            s1_done = B"0101",
            s2_set  = B"0110",
            s2_strt = B"0111",
            s2_conv = B"1000",
            s2_wrmem= B"1001",
            s2_done = B"1010",
            err1    = B"1011",
            err2    = B"1100",
            err3    = B"1101",
            err4    = B"1110",
            err5    = 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 %
    smp_latch    : srff; %used to latch the sample indication%
    smple   : NODE;
    clr_smp : NODE;  % clears the sample clock latch at an opportune time %


    delay[7..0]    : DFFE;
    r_latch[2..1]  : SRFF;
    do_reset       : NODE;
    done           : NODE;

BEGIN

    sample[].clk  = clk;        % This register produces a high pulse 2 clocks wide   %
    sample[].clrn = reset/;     % upon the rising edge of the Integrator sample clock %
    sample[0].d = Iclk;
    sample[1].d = sample[0].q;

    smp_latch.clk  = clk;
    smp_latch.clrn = reset/;
```

```
 smp_latch.s    = Iclk & !sample[1].q;
 smp_latch.r    = clr_smp;

 smple = smp_latch.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_Busy, Mem_Busy, Run_Integ =>   sm    , get_adc, wr_ch, clr_smp, 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   ,     0    ,    x    ,    x      => s1_strt ,    1   ,   0  ,   0    , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_strt ,    x   ,     1    ,    x    ,    x      => s1_conv ,    1   ,   0  ,   0    , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;

   s1_conv ,    x   ,     1    ,    x    ,    x      => s1_conv ,    0   ,   0  ,   1    , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_conv ,    x   ,     0    ,    1    ,    x      => s1_conv ,    0   ,   0  ,   1    , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;
   s1_conv ,    x   ,     0    ,    0    ,    x      => s1_wrmem,    0   ,   0  ,   1    , 1 , 0 , 1 , 0 , 0 , 0 ,   0   ;

   s1_wrmem,    x   ,     x    ,    0    ,    x      => s1_wrmem,    0   ,   1  ,   0    , 1 , 0 , 1 , 0 , 1 , 0 ,   0   ;
   s1_wrmem,    x   ,     x    ,    1    ,    x      => s1_done ,    0   ,   1  ,   0    , 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   ,     0    ,    x    ,    x      => s2_strt ,    1   ,   0  ,   0    , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_strt ,    x   ,     1    ,    x    ,    x      => s2_conv ,    1   ,   0  ,   0    , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;

   s2_conv ,    x   ,     1    ,    x    ,    x      => s2_conv ,    0   ,   0  ,   1    , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_conv ,    x   ,     0    ,    1    ,    x      => s2_conv ,    0   ,   0  ,   1    , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;
   s2_conv ,    x   ,     0    ,    0    ,    x      => s2_wrmem,    0   ,   0  ,   1    , 0 , 1 , 0 , 1 , 0 , 0 ,   0   ;

   s2_wrmem,    x   ,     x    ,    0    ,    x      => s2_wrmem,    0   ,   1  ,   0    , 0 , 1 , 0 , 1 , 0 , 1 ,   0   ;
   s2_wrmem,    x   ,     x    ,    1    ,    x      => s2_done ,    0   ,   1  ,   0    , 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;

 % Start of Reset pulse width control ++++++++++++++++++++++%
 r_latch[].clk  = clk;
 r_latch[].clrn = reset/;
 r_latch[].r    = done;
 r_latch1.s     = r1;
 r_latch2.s     = r2;

 do_reset = r_latch1.q # r_latch2.q;
```

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

    IF((delay[] >= 252)) THEN done = VCC; ELSE done = GND;
    END IF;
    % End Reset pulse width control %

    sel_B_12/ = !s1;
    sel_A_12/ = !s2;
    sel_B_34/ = !s1;
    sel_A_34/ = !s2;

    reset_B_12/ = !r_latch1.q;
    reset_A_12/ = !r_latch2.q;
    reset_B_34/ = !r_latch1.q;
    reset_A_34/ = !r_latch2.q;

    hold_B_12 = h1 & !test;
    hold_A_12 = h2 & !test;
    hold_B_34 = h1 & !test;
    hold_A_34 = h2 & !test;

    test_B_12/ = !(!h1 & test);
    test_A_12/ = !(!h2 & test);
    test_B_34/ = !(!h1 & test);
    test_A_34/ = !(!h2 & test);

    sink_12_off = !test & !clamp;
    sink_34_off = sink_12_off;

    test_off_12 = test & !clamp;
    test_off_34 = test_off_12;


END;
```

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

   convst/, A0, ADC_Busy : OUTPUT;
   adc_A[15..0], adcA_strb, adc_B[15..0], adcB_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;
    ADC_Busy = !done.q;

    strb.clk  = clk;
```

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

    adcA_strb = done.q & !strb.q;
    adcB_strb = done.q & !strb.q;

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;

 %  q1,q2,q3,count,sr_done, sr_cnt[5..0] : OUTPUT;%
)

VARIABLE
   sm :   MACHINE
          OF BITS (q1, q2, q3)
          WITH STATES (
            idle    = B"000",
            ld_sr1  = B"001",
            ld_sr2  = B"011",
            ready   = B"010",
            clk_sr1 = B"110",
            clk_sr2 = B"111",
            ld_dac  = B"101" );

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

sr_cnt[5..0]   : DFFE;
count          : NODE;
sr_done        : NODE;

% buffer the outputs of the state machine %
outbuf[3..0] : DFF;
DAC_SCLKx  : NODE;
DAC_CS/x   : NODE;
DAC_DATAx  : NODE;
DAC_LDACx  : 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[14..0].d    = DAC_A[15..1];  % Divide by 2 %
    dataA[15].d        = GND;

    % Buffer Register for the B channel Data %
    dataB[15..0].clk  = clk;
    dataB[15..0].clrn = reset/;
    dataB[15..0].ena  = dacB_wr;
    dataB[14..0].d    = DAC_B[15..1];  % Divide by 2 %
    dataB[15].d        = GND;

    % Input Serial Data Shift Register %
    sr[].clk  = clk;
    sr[].clrn = reset/;
    sr[].ena  = DAC_SCLK;

    if (ld_sr1 # ld_sr2) THEN sr[15..0].d = dataA[15..0].q; sr[31..16].d = dataB[15..0].q;
    ELSE                      sr[0].d = GND; sr[31..1].d = sr[30..0].q;
    END IF;

    DAC_DATAx = sr[31].q;
```

```
65
66      % Counter to count the 32 clocks for loading the DACs %
67      sr_cnt[].clk  = clk;
68      sr_cnt[].clrn = reset/ & !(sm == ready);
69      sr_cnt[].ena  = count;
70
71      sr_cnt[].d = sr_cnt[].q + 1;
72
73      IF(sr_cnt[] < 63) THEN sr_done = GND;
74      ELSE                   sr_done = VCC;
75      END IF;
76
77      % state machine to control the DAC write sequence %
78      sm.clk = clk;
79      sm.reset = !reset/;
80
81         TABLE
82      % current         current        next                current                  %
83      % state           inputs         state               outputs                  %
84
85        sm      , wr_dacs, sr_done  =>   sm    , dac_busy, DAC_CS/x, DAC_LDACx, DAC_SCLKx, count;
86
87        idle    ,   0    ,   x      => idle    ,    0    ,    1   ,    1    ,    0    ,    0 ;
88        idle    ,   1    ,   x      => ld_sr1  ,    0    ,    1   ,    1    ,    0    ,    0 ;
89
90        ld_sr1  ,   x    ,   x      => ld_sr2  ,    1    ,    1   ,    1    ,    1    ,    0 ;
91        ld_sr2  ,   x    ,   x      => ready   ,    1    ,    1   ,    1    ,    0    ,    0 ;
92        ready   ,   x    ,   x      => clk_sr1 ,    1    ,    1   ,    1    ,    0    ,    0 ;
93
94        clk_sr1 ,   x    ,   0      => clk_sr2 ,    1    ,    0   ,    1    ,    0    ,    1 ;
95        clk_sr1 ,   x    ,   1      => ld_dac  ,    1    ,    0   ,    1    ,    0    ,    1 ;
96
97        clk_sr2 ,   x    ,   0      => clk_sr1 ,    1    ,    0   ,    1    ,    1    ,    1 ;
98        clk_sr2 ,   x    ,   1      => ld_dac  ,    1    ,    0   ,    1    ,    1    ,    1 ;
99
100       ld_dac  ,   x    ,   x      => idle    ,    1    ,    1   ,    0    ,    1    ,    0 ;
101
102
103     END TABLE;
104
105
106      outbuf[].clk = clk;
107      outbuf[].clrn = reset/;
108      outbuf[0].d = DAC_SCLKx;
109      outbuf[1].d = DAC_CS/x;
110      outbuf[2].d = DAC_DATAx;
111      outbuf[3].d = DAC_LDACx;
112
113      DAC_SCLK = outbuf[0].q;
114      DAC_CS/  = outbuf[1].q;
115      DAC_DATA = outbuf[2].q;
116      DAC_LDAC = outbuf[3].q;
117
118 END;
119
```

Note: The column header names should not be changed if you wish to import this .csv file into the Quartus II software.

To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	SignalProbe Source
555_OUT	PIN_153	3	LVTTL	Dedicated Clock	CLK2/LVDSCLK2p		
555_TRG	PIN_178	3	LVTTL	Row I/O	LVDS36p		
ADC_A0_12	PIN_3	1	LVTTL	Row I/O	LVDS13p/CLKUSR		
ADC_A0_34	PIN_17	1	LVTTL	Row I/O	LVDS9n		
ADC_BUSY_12	PIN_238	2	LVTTL	Column I/O	LVDS16p		
ADC_BUSY_34	PIN_21	1	LVTTL	Row I/O	LVDS7n/DM0L		
ADC_CNVST_12	PIN_2	1	LVTTL	Row I/O	LVDS14n		
ADC_CNVST_34	PIN_16	1	LVTTL	Row I/O	LVDS9p		
ADC_DATA_12	PIN_240	2	LVTTL	Column I/O	LVDS15p/DEV_CLRn		
ADC_DATA_34	PIN_19	1	LVTTL	Row I/O	LVDS8n		
ADC_EOC_12	PIN_237	2	LVTTL	Column I/O	LVDS16n		
ADC_EOC_34	PIN_23	1	LVTTL	Row I/O	VREF1B1		
ADC_RDERR_12	PIN_1	1	LVTTL	Row I/O	LVDS14p/INIT_DONE		
ADC_RDERR_34	PIN_15	1	LVTTL	Row I/O	LVDS10n		
ADC_SCLK_12	PIN_236	2	LVTTL	Column I/O	LVDS17p/DQ0T7		
ADC_SCLK_34	PIN_24	1	LVTTL	Row I/O	nCSO		
ADC_SYNC_12	PIN_239	2	LVTTL	Column I/O	LVDS15n/DEV_OE		
ADC_SYNC_34	PIN_20	1	LVTTL	Row I/O	LVDS7p		
AMATCH/	PIN_177	3	LVTTL	Row I/O	LVDS36n		
BAS/	PIN_168	3	LVTTL	Row I/O	LVDS38n		
BDS0/	PIN_170	3	LVTTL	Row I/O	DPCLK4/DQS0R		
BDS1/	PIN_173	3	LVTTL	Row I/O	LVDS37n/DQ1R2		
BERR	PIN_167	3	LVTTL	Row I/O	LVDS39p		
BIACK/	PIN_165	3	LVTTL	Row I/O	LVDS40p		
BLWORD/	PIN_164	3	LVTTL	Row I/O	LVDS40n		
BSYSCLK	PIN_29	1	LVTTL	Dedicated Clock	CLK1/LVDSCLK1n		
BSYSRST/	PIN_174	3	LVTTL	Row I/O	LVDS37p/DQ1R1		
BWRITE/	PIN_169	3	LVTTL	Row I/O	LVDS38p/DQ1R3		
C0	PIN_61	4	LVTTL	Column I/O	LVDS71p		
C1	PIN_62	4	LVTTL	Column I/O	LVDS71n		
C2	PIN_63	4	LVTTL	Column I/O	LVDS70p		
C3	PIN_64	4	LVTTL	Column I/O	LVDS70n		
CEAB/	PIN_182	2	LVTTL	Column I/O	LVDS34p		
DAC_CS_12x/	PIN_5	1	LVTTL	Row I/O	VREF0B1		
DAC_CS_34x/	PIN_12	1	LVTTL	Row I/O	LVDS11p/DQ0L2		
DAC_DATA_12x	PIN_6	1	LVTTL	Row I/O			
DAC_DATA_34x	PIN_13	1	LVTTL	Row I/O	LVDS11n/DQ0L3		
DAC_LDAC_12x	PIN_7	1	LVTTL	Row I/O	LVDS12p/DQ0L0		
DAC_LDAC_34x	PIN_14	1	LVTTL	Row I/O	LVDS10p		
DAC_SCLK_12x	PIN_4	1	LVTTL	Row I/O	LVDS13n		
DAC_SCLK_34x	PIN_11	1	LVTTL	Row I/O	DPCLK1/DQS0L		
DTACK	PIN_166	3	LVTTL	Row I/O	LVDS39n		
ENA_DAC_SRCx	PIN_8	1	LVTTL	Row I/O	LVDS12n/DQ0L1		
ENDCYCLE	PIN_74	4	LVTTL	Column I/O	VREF2B4		
HOLD_A_12	PIN_224	2	LVTTL	Column I/O	LVDS20p		
HOLD_A_34	PIN_225	2	LVTTL	Column I/O	LVDS19n		
HOLD_B_12	PIN_208	2	LVTTL	Column I/O	VREF1B2		
HOLD_B_34	PIN_213	2	LVTTL	Column I/O			
LEAB/	PIN_183	2	LVTTL	Column I/O	LVDS33n		
Logic_CLK	PIN_28	1	LVTTL	Dedicated Clock	CLK0/LVDSCLK1p		
MAKEMEAS	PIN_73	4	LVTTL	Column I/O	DPCLK7/DQS1B		
MEM_OE/	PIN_84	4	LVTTL	Column I/O	LVDS63p		
MEM_WE/	PIN_108	4	LVTTL	Column I/O	DPCLK6/DQS0B		
OEAB/	PIN_181	2	LVTTL	Column I/O	LVDS34n		
OEBA/	PIN_179	3	LVTTL	Row I/O	LVDS35n		
RECEIVE	PIN_76	4	LVTTL	Column I/O	LVDS67n/DQ1B5		
RESET_A_12/	PIN_228	2	LVTTL	Column I/O	DPCLK2/DQS1T		
RESET_A_34/	PIN_233	2	LVTTL	Column I/O	LVDS18n/DQ0T4		
RESET_B_12/	PIN_216	2	LVTTL	Column I/O	LVDS23n		
RESET_B_34/	PIN_217	2	LVTTL	Column I/O	LVDS23p		
SEL_A_12/	PIN_226	2	LVTTL	Column I/O	LVDS19p		
SEL_A_34/	PIN_227	2	LVTTL	Column I/O	VREF2B2		
SEL_B_12/	PIN_214	2	LVTTL	Column I/O	LVDS24n		
SEL_B_34/	PIN_215	2	LVTTL	Column I/O	LVDS24p		
SINK_12_OFF	PIN_220	2	LVTTL	Column I/O	LVDS21n		
SINK_34_OFF	PIN_221	2	LVTTL	Column I/O	LVDS21p		
SW1	PIN_175	3	LVTTL	Row I/O	DQ1R0		
SW2	PIN_176	3	LVTTL	Row I/O	VREF0B3		
TEST_12_OFF	PIN_222	2	LVTTL	Column I/O			
TEST_34_OFF	PIN_223	2	LVTTL	Column I/O	LVDS20n		
TEST_A_12/	PIN_234	2	LVTTL	Column I/O	LVDS18p/DQ0T5		
TEST_A_34/	PIN_235	2	LVTTL	Column I/O	LVDS17n/DQ0T6		
TEST_B_12/	PIN_218	2	LVTTL	Column I/O	LVDS22n		
TEST_B_34/	PIN_219	2	LVTTL	Column I/O	LVDS22p		
TRANSMIT	PIN_75	4	LVTTL	Column I/O	LVDS67p		
TRIG1	PIN_41	1	LVTTL	Row I/O			
TRIG2	PIN_42	1	LVTTL	Row I/O	LVDS6p		
b_GLOBAL_OSC	PIN_152	3	LVTTL	Dedicated Clock	CLK3/LVDSCLK2n		
BAM[0]	PIN_158	3	LVTTL	Row I/O	LVDS43n/DM1R		
b_SPARE[1]	PIN_194	2	LVTTL	Column I/O	VREF0B2		
b_SPARE[2]	PIN_193	2	LVTTL	Column I/O	DPCLK3/DQS0T		
b_SPARE[3]	PIN_185	2	LVTTL	Column I/O	LVDS32n/DQ0T0		
b_SPARE[4]	PIN_184	2	LVTTL	Column I/O	LVDS33p		

BAM[1]	PIN_159	3	LVTTL	Row I/O	LVDS43p
BAM[2]	PIN_160	3	LVTTL	Row I/O	LVDS42n
BAM[3]	PIN_161	3	LVTTL	Row I/O	LVDS42p
BAM[4]	PIN_162	3	LVTTL	Row I/O	LVDS41n
BAM[5]	PIN_163	3	LVTTL	Row I/O	LVDS41p
LA[0]	PIN_86	4	LVTTL	Column I/O	LVDS62p
LA[1]	PIN_85	4	LVTTL	Column I/O	LVDS63n
LA[10]	PIN_117	4	LVTTL	Column I/O	LVDS53p
LA[11]	PIN_119	4	LVTTL	Column I/O	LVDS52p
LA[12]	PIN_120	4	LVTTL	Column I/O	LVDS52n
LA[13]	PIN_118	4	LVTTL	Column I/O	LVDS53n
LA[14]	PIN_116	4	LVTTL	Column I/O	LVDS54n/DQ1B0
LA[15]	PIN_114	4	LVTTL	Column I/O	LVDS55n/DQ1B2
LA[2]	PIN_83	4	LVTTL	Column I/O	LVDS64n
LA[3]	PIN_81	4	LVTTL	Column I/O	LVDS65n
LA[4]	PIN_79	4	LVTTL	Column I/O	
LA[5]	PIN_78	4	LVTTL	Column I/O	LVDS66n
LA[6]	PIN_80	4	LVTTL	Column I/O	LVDS65p
LA[7]	PIN_82	4	LVTTL	Column I/O	LVDS64p
LA[8]	PIN_113	4	LVTTL	Column I/O	LVDS55p/DQ1B3
LA[9]	PIN_115	4	LVTTL	Column I/O	LVDS54p/DQ1B1
LD[0]	PIN_93	4	LVTTL	Column I/O	VREF1B4
LD[1]	PIN_95	4	LVTTL	Column I/O	LVDS61n
LD[10]	PIN_102	4	LVTTL	Column I/O	LVDS57p
LD[11]	PIN_100	4	LVTTL	Column I/O	LVDS58p
LD[12]	PIN_98	4	LVTTL	Column I/O	LVDS59p
LD[13]	PIN_96	4	LVTTL	Column I/O	LVDS60p
LD[14]	PIN_94	4	LVTTL	Column I/O	LVDS61p/DM1B
LD[15]	PIN_88	4	LVTTL	Column I/O	
LD[2]	PIN_97	4	LVTTL	Column I/O	LVDS60n
LD[3]	PIN_99	4	LVTTL	Column I/O	LVDS59n
LD[4]	PIN_101	4	LVTTL	Column I/O	LVDS58n
LD[5]	PIN_103	4	LVTTL	Column I/O	LVDS57n
LD[6]	PIN_105	4	LVTTL	Column I/O	LVDS56n
LD[7]	PIN_107	4	LVTTL	Column I/O	VREF0B4
LD[8]	PIN_106	4	LVTTL	Column I/O	
LD[9]	PIN_104	4	LVTTL	Column I/O	LVDS56p
MEM_CS/[0]	PIN_65	4	LVTTL	Column I/O	LVDS69p
MEM_CS/[1]	PIN_66	4	LVTTL	Column I/O	LVDS69n
MEM_CS/[2]	PIN_67	4	LVTTL	Column I/O	LVDS68p/DQ1B7
MEM_CS/[3]	PIN_68	4	LVTTL	Column I/O	LVDS68n/DQ1B6
MEM_PTR[0]	PIN_60	1	LVTTL	Row I/O	LVDS0n
MEM_PTR[1]	PIN_59	1	LVTTL	Row I/O	LVDS0p
MEM_PTR[10]	PIN_48	1	LVTTL	Row I/O	LVDS3p/DQ0L4
MEM_PTR[11]	PIN_47	1	LVTTL	Row I/O	LVDS4n
MEM_PTR[12]	PIN_46	1	LVTTL	Row I/O	LVDS4p
MEM_PTR[13]	PIN_45	1	LVTTL	Row I/O	LVDS5n
MEM_PTR[14]	PIN_44	1	LVTTL	Row I/O	LVDS5p
MEM_PTR[15]	PIN_43	1	LVTTL	Row I/O	LVDS6n
MEM_PTR[2]	PIN_58	1	LVTTL	Row I/O	LVDS1n
MEM_PTR[3]	PIN_57	1	LVTTL	Row I/O	LVDS1p
MEM_PTR[4]	PIN_56	1	LVTTL	Row I/O	
MEM_PTR[5]	PIN_55	1	LVTTL	Row I/O	VREF2B1
MEM_PTR[6]	PIN_54	1	LVTTL	Row I/O	LVDS2n/DQ0L7
MEM_PTR[7]	PIN_53	1	LVTTL	Row I/O	LVDS2p/DQ0L6
MEM_PTR[8]	PIN_50	1	LVTTL	Row I/O	DPCLK0/DQS1L
MEM_PTR[9]	PIN_49	1	LVTTL	Row I/O	LVDS3n/DQ0L5
vAA[1]	PIN_122	3	LVTTL	Row I/O	LVDS51p
vAA[10]	PIN_133	3	LVTTL	Row I/O	LVDS48p/DQ1R4
vAA[11]	PIN_134	3	LVTTL	Row I/O	LVDS47n
vAA[12]	PIN_135	3	LVTTL	Row I/O	LVDS47p
vAA[13]	PIN_136	3	LVTTL	Row I/O	LVDS46n
vAA[14]	PIN_137	3	LVTTL	Row I/O	LVDS46p
vAA[15]	PIN_138	3	LVTTL	Row I/O	LVDS45n
vAA[16]	PIN_139	3	LVTTL	Row I/O	LVDS45p
vAA[17]	PIN_140	3	LVTTL	Row I/O	LVDS44n
vAA[18]	PIN_141	3	LVTTL	Row I/O	LVDS44p
vAA[2]	PIN_123	3	LVTTL	Row I/O	LVDS50n
vAA[3]	PIN_124	3	LVTTL	Row I/O	LVDS50p
vAA[4]	PIN_125	3	LVTTL	Row I/O	LVDS49n/DQ1R7
vAA[5]	PIN_126	3	LVTTL	Row I/O	LVDS49p
vAA[6]	PIN_127	3	LVTTL	Row I/O	VREF2B3
vAA[7]	PIN_128	3	LVTTL	Row I/O	DQ1R6
vAA[8]	PIN_131	3	LVTTL	Row I/O	DPCLK5/DQS1R
vAA[9]	PIN_132	3	LVTTL	Row I/O	LVDS48n/DQ1R5
vDD[0]	PIN_186	2	LVTTL	Column I/O	LVDS32p/DQ0T1
vDD[1]	PIN_187	2	LVTTL	Column I/O	LVDS31n/DQ0T2
vDD[10]	PIN_202	2	LVTTL	Column I/O	LVDS27n
vDD[11]	PIN_203	2	LVTTL	Column I/O	LVDS27p
vDD[12]	PIN_204	2	LVTTL	Column I/O	LVDS26n
vDD[13]	PIN_205	2	LVTTL	Column I/O	LVDS26p
vDD[14]	PIN_206	2	LVTTL	Column I/O	LVDS25n/DM0T
vDD[15]	PIN_207	2	LVTTL	Column I/O	LVDS25p
vDD[2]	PIN_188	2	LVTTL	Column I/O	LVDS31p/DQ0T3
vDD[3]	PIN_195	2	LVTTL	Column I/O	
vDD[4]	PIN_196	2	LVTTL	Column I/O	LVDS30n
vDD[5]	PIN_197	2	LVTTL	Column I/O	LVDS30p

vDD[6]	PIN_198	2	LVTTL	Column I/O	LVDS29n
vDD[7]	PIN_199	2	LVTTL	Column I/O	LVDS29p
vDD[8]	PIN_200	2	LVTTL	Column I/O	LVDS28n
vDD[9]	PIN_201	2	LVTTL	Column I/O	LVDS28p
	PIN_18	1	LVTTL	Row I/O	LVDS8p
	PIN_37	1	LVTTL	Row I/O	ASDO
	PIN_38	1	LVTTL	Row I/O	PLL1_OUTp
	PIN_39	1	LVTTL	Row I/O	PLL1_OUTn
	PIN_77	4	LVTTL	Column I/O	LVDS66p/DQ1B4
	PIN_87	4	LVTTL	Column I/O	LVDS62n
	PIN_121	3	LVTTL	Row I/O	LVDS51n
	PIN_143	3	LVTTL	Row I/O	PLL2_OUTn
	PIN_144	3	LVTTL	Row I/O	PLL2_OUTp
	PIN_156	3	LVTTL	Row I/O	VREF1B3
	PIN_180	3	LVTTL	Row I/O	LVDS35p
	PIN_9		LVTTL	VCCIO1	
	PIN_10		LVTTL	GND	
	PIN_22		LVTTL	VCCIO1	
	PIN_25	1	LVTTL	Dedicated Programming	DATA0
	PIN_26	1	LVTTL	Dedicated Programming	nCONFIG
	PIN_27		LVTTL	VCCA_PLL1	
	PIN_30		LVTTL	GNDA_PLL1	
	PIN_31		LVTTL	GNDG_PLL1	
	PIN_32	1	LVTTL	Dedicated Programming	nCEO
	PIN_33	1	LVTTL	Dedicated Programming	nCE
	PIN_34	1	LVTTL	Dedicated Programming	MSEL0
	PIN_35	1	LVTTL	Dedicated Programming	MSEL1
	PIN_36	1	LVTTL	Dedicated Programming	DCLK
	PIN_40		LVTTL	GND	
	PIN_51		LVTTL	VCCIO1	
	PIN_52		LVTTL	GND	
	PIN_69		LVTTL	GND	
	PIN_70		LVTTL	VCCIO4	
	PIN_71		LVTTL	GND	
	PIN_72		LVTTL	VCCINT	
	PIN_89		LVTTL	GND	
	PIN_90		LVTTL	VCCINT	
	PIN_91		LVTTL	GND	
	PIN_92		LVTTL	VCCIO4	
	PIN_109		LVTTL	GND	
	PIN_110		LVTTL	VCCINT	
	PIN_111		LVTTL	GND	
	PIN_112		LVTTL	VCCIO4	
	PIN_129		LVTTL	GND	
	PIN_130		LVTTL	VCCIO3	
	PIN_142		LVTTL	GND	
	PIN_145	3	LVTTL	Dedicated Programming	CONF_DONE
	PIN_146	3	LVTTL	Dedicated Programming	nSTATUS
	PIN_147	3	LVTTL	JTAG	TCK
	PIN_148	3	LVTTL	JTAG	TMS
	PIN_149	3	LVTTL	JTAG	TDO
	PIN_150		LVTTL	GNDG_PLL2	
	PIN_151		LVTTL	GNDA_PLL2	
	PIN_154		LVTTL	VCCA_PLL2	
	PIN_155	3	LVTTL	JTAG	TDI
	PIN_157		LVTTL	VCCIO3	
	PIN_171		LVTTL	GND	
	PIN_172		LVTTL	VCCIO3	
	PIN_189		LVTTL	VCCIO2	
	PIN_190		LVTTL	GND	
	PIN_191		LVTTL	VCCINT	
	PIN_192		LVTTL	GND	
	PIN_209		LVTTL	VCCIO2	
	PIN_210		LVTTL	GND	
	PIN_211		LVTTL	VCCINT	
	PIN_212		LVTTL	GND	
	PIN_229		LVTTL	VCCINT	
	PIN_230		LVTTL	GND	
	PIN_231		LVTTL	VCCIO2	
	PIN_232		LVTTL	GND	