```
TITLE "VME Address Decoder";

%*************************************************************************
    Author: Walter Knopf & Neal Wilcer
    7/22/98
    File Name: addrdec.tdf

    This portion of the chip handles the slave and interrupter vme functions.
    In addition, it controls whether the TSC , the CSRs, or
    the emulator ram is being accessed and the associated handshaking.

*************************************************************************%

INCLUDE "vmemap.inc";


SUBDESIGN AddrDec
(
    RFCLK               : INPUT;    % 53 MHz Clock              %
    Bsync               : INPUT;    % 53/7 Mhz clock %
    % Bus Interface Signals %
    VmeWr               : INPUT;    % VME Write                 %
    VmeAs               : INPUT;    % VME Address Strobe        %
    VmeDs[1..0]         : INPUT;    % VME Data Strobes          %
    VmeLword            : INPUT;    % VME Longword              %
    VmeA[23..1]         : INPUT;    % VME Address bus.          %
    VmeAm[5..0]         : INPUT;    % VME Address Modifiers     %
    VmeDtack            : OUTPUT;   % VME Data Acknowledge      %
    VmeBerr             : INPUT;    % VME Bus Error             %
    VmeIack             : INPUT;    % VME Interrupt Ackn.       %
    /VIackIn            :INPUT;     % Vme interrupt acknowledge input from
                                        IACKIN daisy chain %
    /VIackOut           :OUTPUT;    % Vme interrupt acknowledge input to
                                        IACKOUT daisy chain %
    Vreset              :INPUT;     % reset signal %
    % Address Select switch %
    AdSw[3..0]          : INPUT;
    TSCIrq              : INPUT;    % Interrupt request from the one of the status
                                            controllers %
    /VIRQ[7..1]         :OUTPUT;    % One of seven interrupt levels available on vme
                                        the user will select via register which one
                                        to use %
    IRQSel[2..0]        :INPUT;     %   3-bit value that translates to the vme IRQ level%

    % Local Select and Ack Signals %
    RegSel              : OUTPUT;   % Local Register Select     %
    RegAck              : INPUT;    % Local Register Acknowl.   %
    % Status Select and Ack Signals %
```

```
    /StatSel                : OUTPUT;   % Status Register Select    %
    StatAck             : INPUT;    % Status Register Acknowl.  %
    /StatusEnable       :OUTPUT;    % enable for the buffers that write/read the TSC modules %
    StatusDir           : OUTPUT;   % direction control for the buffers that write/read the TSC %
    /StatusDS           : OUTPUT;   % data strobe for the status registers %
    StatusClk           : OUTPUT;   % Copy of the bsync clock XORed with the StatSel line. If we're not
                                        trying to read the status modules from vme, this clock will latch
                                        status data on the TSC modules. %
    % Emulation Ram select and Ack Signals %
    RamSel              : OUTPUT;   % Emulation Ram Select      %
    RamAck              : INPUT;    % Emulation Ram Acknowl.    %
    % Decoded Read and Write Strobes %
    RdStb               : OUTPUT;   % Read Strobe               %
    WrStb               : OUTPUT;   % Write Strobe              %
    % The enable and direction controls for the vme data buffers %
    DataDir             : OUTPUT;   % Direction control for the vme read/write buffers %
    VmeDataEnable       :OUTPUT;
)
VARIABLE
        Amod                : NODE;     % Address Modifier          %
    Valid               : NODE;     % Valid VME Access          %
    Active              : DFF;      % Active Cycle              %
    Ds[1..0]            : DFF;      % Local Sync. Addr.Stb.     %
    AsFF                : DFF;      % Sync AS to RFCLK          %
    WrtFF               :DFF;       % used to latch the vme write signal %
    IRQFF               :DFF;       % D flop used to latch the IRQ signal from the TSC module %

BEGIN

DEFAULTS

    /VIrq[] = VCC;
END DEFAULTS;

    % Status readout logic %
    StatusClk = Bsync AND /StatSel; % Have the status clock latch the status data in the TSC module, but
                                        not if the status registers are selected from vme %
    !/StatusDs = !/StatSel AND (Ds[1..0] == 3) AND Valid & Active.q; % If the status registers are selected
                                                                       and the DS signals are active, send a
                                                                       Data strobe to the TSC modules %

    % Interrupt logic %
    IRQFF.clk = TSCIrq; % Use and interrupt from the TSC to clock %
    IRQFF.d = GND;      % If we get a clock, the irq output goes low %

                        % If the lower three bits of the vme address line are equivalent to our chose request level AND
the vme IACK
```

```
                            signal is active AND we've got the IACKIN signal present and we've got an interrupt request
active AND
                            the DS lines are low, then go ahead and clear (preset) the interrupt request %

    IRQFF.prn = Vreset
                OR
                !((VmeA[3..1] == IRQSel[2..0]) AND
                !VmeIack AND
                !/VIackIn AND
                (IRQFF.q == GND) AND
                (VmeDs[] == GND));

    % Choose the IRQ level using the value in the IRQSel register %
    IF TSCIrq THEN
        CASE IRQSel[] IS
            WHEN 1 =>
                /VIRQ1 = IRQFF.q;
            WHEN 2 =>
                /VIRQ2 = IRQFF.q;
            WHEN 3 =>
                /VIRQ3 = IRQFF.q;
            WHEN 4 =>
                /VIRQ4 = IRQFF.q;
            WHEN 5 =>
                /VIRQ5 = IRQFF.q;
            WHEN 6 =>
                /VIRQ6 = IRQFF.q;
            WHEN 7 =>
                /VIRQ7 = IRQFF.q;
        END CASE;
    END IF;

    % Conditions for passing the IACKIN signal along to the next slot %
    IF (!/VIackIn AND !VmeIack AND ((IRQFF.q == VCC) OR (VmeA[3..1] != IRQSel[2..0]))) THEN
        /VIackOut = GND;
    ELSE
        /VIackOut = VCC;
    END IF;





    Amod =  (VmeAm[5..0] == AmStdNonPrivData) OR
            (VmeAm[5..0] == AmStdNonPrivCode);
% OR
            (VmeAm[5..0] == AmStdSuperData) OR
            (VmeAm[5..0] == AmStdSuperCode); !!! commented out for simulation %
```

```
% Start valid cycle if conditions right %
Valid       = (VmeLword & Amod & (AdSw[3..0] == VmeA[23..20])
               & VmeIack & VmeBerr) OR (Active.q & VmeAs); % hold cycle %

% latch the  vme write signal to eliminate glitches on the rdstrb output %

WrtFF.clk = !rfclk;
WrtFF.d = VmeWr;
WrtFF.prn = Vreset;

% align AS with 53 MHz clock %
AsFF.clk    = !RFCLK;
AsFF.d      = !VmeAs;
AsFF.prn    = Vreset;

% If it's a vme write and this board is selected, enable the buffers to
        drive onto vme, otherwise set them to receive data. If this board
        isn't selected, tri-state the buffers %

!DataDir = !VmeWr OR (!DataDir AND !VmeDtack) ;
!VmeDataEnable = Valid AND (VmeDs[] == 0);
!StatusDir = !Datadir AND !/Statsel;
!/StatusEnable = !VmeDataEnable AND !/StatSel;


% Latch active cycle flag %
Active.clk  = AsFF.q;
Active.d    = Valid;
Active.clrn = Valid AND Vreset;

% Address range select lines %
RegSel = Active.q & (VmeA[19..1] <= RegHi) & (vmeds[] == 0);
!/StatSel = Active.q & (VmeA[19..1] >= StatLo) & (VmeA[19..1] <= StatHi);
RamSel = Active.q & (VmeA[19..1] >= RamLo) & (VmeA[19..1] <= RamHi) & (vmeds[] == 0);

% Write Strobe %
WrStb = !WrtFF.q & (Ds[1..0] == 3) & Valid & Active.q;
% Read Strobe %
RdStb = WrtFF.q & (Ds[1..0] == 3) & Valid & Active.q;

% Data Strobe Syncronization %
Ds[].clk    = !RFCLK;
Ds[].clrn   = Active.q AND Vreset;
Ds[0].d = (VmeDs[1..0] == 0) & Active.q;
Ds[1].d = (VmeDs[1..0] == 0) & Active.q;

% Send a Dtack if we're done doing a read/write or if we're done
```

```
        with an interrupt request %

    !VmeDtack = (Active.q AND % Standard read/write dtack %
                ((!/StatSel & StatAck) OR
                ( RegSel & RegAck) OR
                ( RamSel & RamAck)))
                OR          %Interrupt request Dtack %
                (((VmeA[3..1] == IRQSel[2..0]) AND
                !VmeIack AND
                !/VIackIn AND
                (IRQFF.q == GND) AND
                (VmeDs[] == GND))
                OR  %If the Dtack's active, hold it until DS lines go inactive %
                (!VmeDtack AND (VmeDs[] == GND)));

END;
```