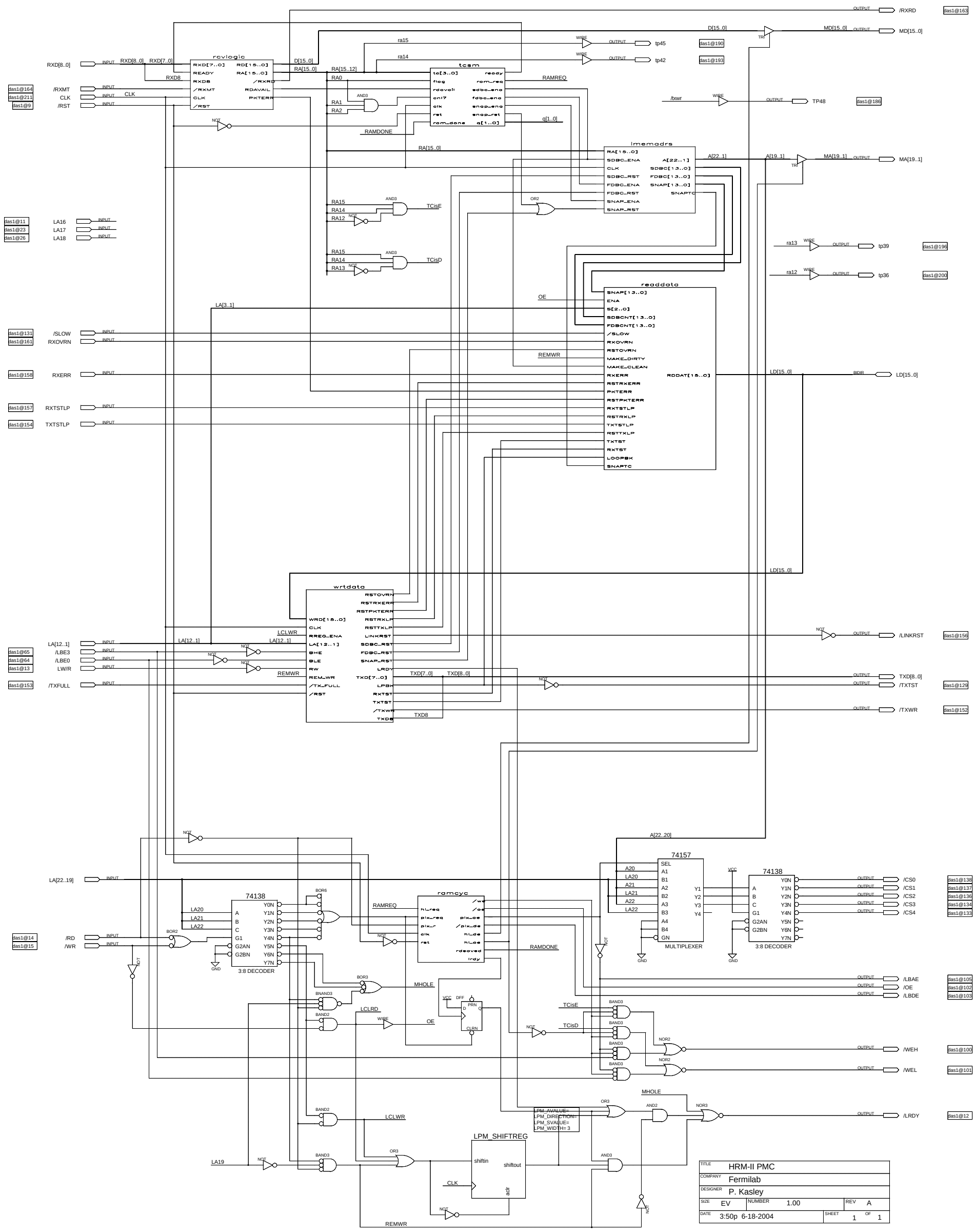
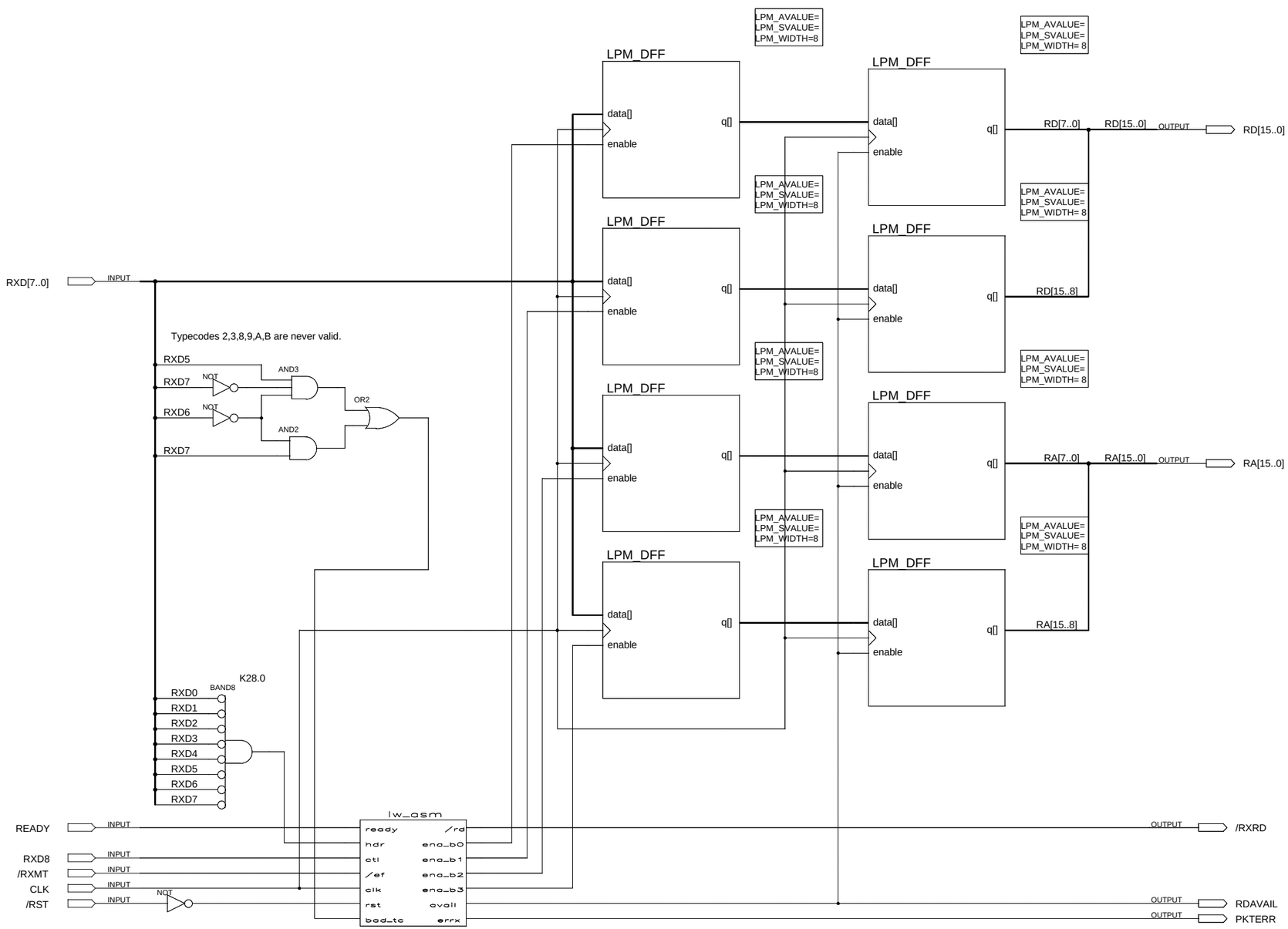

rcvlogic
tcsm
lmemadrs
readdata
wrtdata
ramcyc
74157
MULTIPLEXER
74138
3:8 DECODER
LPM_SHIFTREG
LPM_AVALUE=
LPM_DIRECTION=
LPM_SVALUE=
LPM_WIDTH= 3
TITLE HRM-II PMC
COMPANY Fermilab
DESIGNER P. Kasley
SIZE EV
NUMBER 1.00
REV A
DATE 3:50p 6-18-2004
SHEET 1 OF 1



Scan data from the receiver FIFO until a packet header is found.
When a header is found, load each of the next four bytes from the FIFO into registers.
When the registers are full, signal Received Data Available and wait for the packet to be "consumed".
After the packet has been consumed, scan for the next header.
If an error occurs, signal Packet Error, discard the current data and begin scanning for a header.

TITLE	RCVLOGIC.GDF		
COMPANY	Fermilab		
DESIGNER	P. Kasley		
SIZE D	NUMBER 1.00		REV A
DATE 11:07a 5-12-2003		SHEET 1	OF 1

```
%*****************************************************************************
DAS-PMC Hot Link Receiver Long Word Assembly
8 Nov 02 pak

Read the RX FIFO until a header is found. Retrieve the next four bytes from the
RX FIFO to a 32-bit register. Signal external logic that a frame is ready for
processing and wait for it to be accepted. Search for next header.

If header is found where data is expected, signal an error, discard the current
frame, and build a new frame. If control is found where data is expected, signal
error, discard the current frame, and search for a new header.
*****************************************************************************%

TITLE "Long Word Assembly";

SUBDESIGN lw_asm
(ready      : INPUT;        % Type Code controller ready to accept a frame. Deasserted by
AVAIL. %
 hdr        : INPUT;        % Decode of FIFO data indicates start of data frame %
 ctl        : INPUT;        % SC/D* (control symbol received) %
 /ef        : INPUT;        % RX FIFO Empty flag %
 clk        : INPUT;        % clock %
 rst        : INPUT;        % reset %
 bad_tc     : INPUT;        % Invalid type code %
 /rd        : OUTPUT;       % RX FIFO read enable %
 ena_b0     : OUTPUT;       % Enable byte 0 latch (D7:0) %
 ena_b1     : OUTPUT;       % Enable byte 1 latch (D15:8) %
 ena_b2     : OUTPUT;       % Enable byte 2 latch (D23:16) %
 ena_b3     : OUTPUT;       % Enable byte 3 latch (D31:24) %
 avail      : OUTPUT;       % New frame available to process. %
 errx       : OUTPUT;       % Error flag %
)

VARIABLE
lw_asm: MACHINE OF BITS (q[3..0])
    WITH STATES (   wait0=H"0",
                    read0=H"1",
                    wait1=H"9",
                    read1=H"3",
                    wait2=H"b",
                    read2=H"2",
                    wait3=H"a",
                    read3=H"6",
                    wait4=H"e",
                    read4=H"7",
                    wait5=H"f",
                    xfrrd0=H"5",
                    xfr=H"4",
                    void3=H"c",
                    void4=H"d",
                    err=H"8");

BEGIN

% State machine clock and reset%
lw_asm.clk=clk;
lw_asm.reset=rst;

% Outputs %

% Wrap the MT flag back tothe read input. Disable reading during these states. %
/rd = !/ef # err # void3 # void4 # wait5 # read4;

ena_b0=read1;
ena_b1=read2;
```

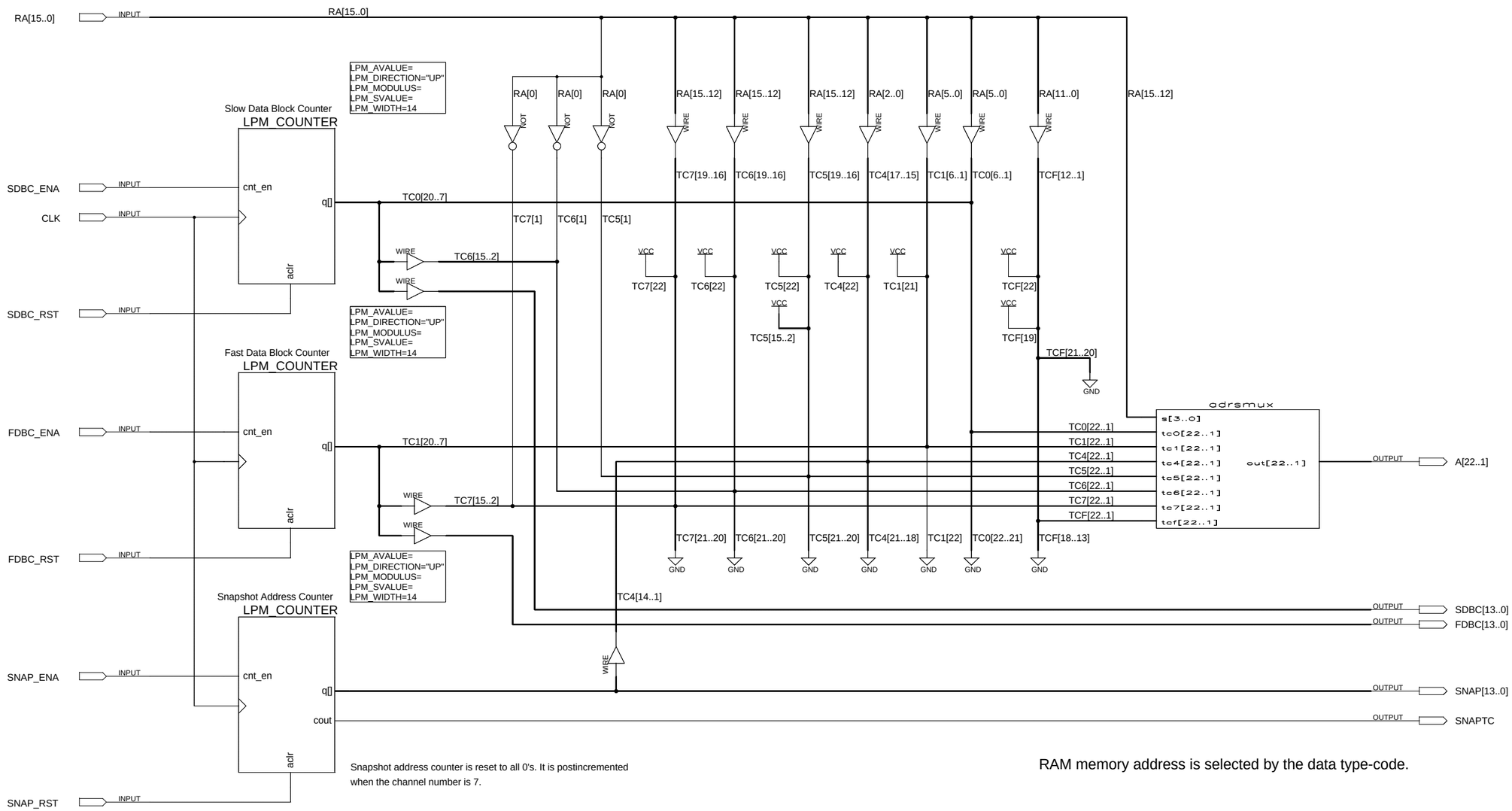
```
ena_b2=read3;
ena_b3=read4;
errx=err;
avail=xfr # xfrrd0;

% state transitions %
TABLE
    lw_asm,/ef,hdr,ctl,ready,bad_tc => lw_asm;

%   PS      /EF HDR CTL RDY BAD_TC  =>  NS %

%  WAIT0: idle %
    wait0,  0,  x,  x,  x,      x       =>  wait0; % Idle, waiting for data %
    wait0,  1,  x,  x,  x,      x       =>  read0; % FIFO Data %

%  READ0: pull from FIFO until frame header is found %
    read0,  0,  0,  x,  x,      x       =>  wait0; % Header not found, no further data %
    read0,  1,  0,  x,  x,      x       =>  read0; % Header not found, FIFO has data %
    read0,  0,  1,  1,  x,      x       =>  wait1; % Header found but no further data %
    read0,  1,  1,  1,  x,      x       =>  read1; % Header found, FIFO has data %

%  WAIT1: wait for LSB0 data %
    wait1,  0,  x,  x,  x,      x       =>  wait1; % Waiting for data %
    wait1,  1,  x,  x,  x,      x       =>  read1; % FIFO Data %

%  READ1: latch LSB0 data %
    read1,  0,  x,  x,  x,      x       =>  wait2; % Waiting further data %
    read1,  1,  x,  0,  x,      x       =>  read2; % FIFO has LSB1 %
    read1,  x,  x,  1,  x,      x       =>  err; % ERROR: Expected data, found control %

%  WAIT2: wait for LSB1 data %
    wait2,  0,  x,  x,  x,      x       =>  wait2; % Waiting for data %
    wait2,  1,  x,  x,  x,      x   =>  read2; % FIFO Data %

%  READ2: latch LSB1 data %
    read2,  0,  x,  x,  x,      x       =>  wait3; % Waiting further data %
    read2,  1,  x,  0,  x,      x       =>  read3; % FIFO has MSB0 %
    read2,  x,  x,  1,  x,      x       =>  err; % ERROR: Expected data, found control %

%  WAIT3: wait for MSB0 data %
    wait3,  0,  x,  x,  x,      x       =>  wait3; % Waiting for data %
    wait3,  1,  x,  x,  x,      x       =>  read3; % FIFO Data %

%  READ3: latch MSB0 data %
    read3,  0,  x,  x,  x,      x       =>  wait4; % Waiting further data %
    read3,  1,  x,  0,  x,      x       =>  read4; % FIFO has MSB1 %
    read3,  x,  x,  1,  x,      x       =>  err; % ERROR: Expected data, found control %

%  WAIT4: wait for MSB1 data %
    wait4,  0,  x,  x,  x,      x       =>  wait4; % Waiting for data %
    wait4,  1,  x,  x,  x,      x       =>  read4; % FIFO Data %

%   PS      /EF HDR CTL RDY BAD_TC  =>  NS %
%  READ4: latch MSB1 data %
    read4,  0,  x,  0,  1,      0       =>  xfr; % FIFO MT. Transfer frame to buffer and
idle %
    read4,  0,  x,  0,  0,      0       =>  wait5; % FIFO MT but memory busy. Wait. %
    read4,  1,  x,  0,  1,      0       =>  xfrrd0; % FIFO not MT and memory idle. Get to
work on next frame. %
    read4,  1,  x,  0,  0,      0       =>  wait5; % FIFO not MT but memory busy. Wait. %
    read4,  x,  x,  1,  x,      x       =>  err; % ERROR: Expected data, found control %
    read4,  x,  x,  0,  x,      1       =>  err; % ERROR: Bad typecode %

%  XFR: transfer from receiving distributor to holding register %
    xfr,    0,  x,  x,  x,      x       =>  wait0; % nothing in FIFO. Go to idle. %
    xfr,    1,  x,  x,  x,      x       =>  read0; % New data in FIFO. Go to search header %
```

```
%  XFRRD0: transfer from receiving distributor to holding register and
    start into next frame %
    xfrrd0,0,   0,  x,  x,      x       =>  wait0; % Header not found, no further data %
    xfrrd0,0,   1,  0,  x,      x       =>  wait0; % Header not found, no further data %
    xfrrd0,0,   1,  1,  x,      x       =>  wait1; % Header found but no further data %
    xfrrd0,1,   0,  x,  x,      x       =>  read0; % Header not found, FIFO has data %
    xfrrd0,1,   1,  0,  x,      x       =>  read0; % Header not found, FIFO has data %
    xfrrd0,1,   1,  1,  x,      x       =>  read1; % Header found, FIFO has data %

%  WAIT5: wait for frame to be stored or discard invalid frame %
    wait5,  x,  x,  x,  0,      x       =>  wait5;
    wait5,  x,  x,  x,  1,      x       =>  xfr;

%  ERR: dummy state to make error output %
    err,    x,  x,  x,  x,      x       =>  wait0;

%  VOID: illegal states %
    void3,  x,  x,  x,  x,      x       =>  wait0;
    void4,  x,  x,  x,  x,      x       =>  wait0;

END TABLE;

END;
```

```
%*****************************************************************************
DAS-PMC Type Code State Machine 21Jun01 pak

TC  Flag    Cnl     T1      T2
0   x       x       store   skip
1   x       x       store   skip
4   x       =7      store   snap++
4   x       !=7     store   nop
5   0       x       store   snap=0
5   1       x       store   skip
6   0       x       store   skip
6   1       x       sdbc++  store
7   0       x       store   skip
7   1       x       fdbc++  store
c   x       x       store   skip
d   x       x       store   skip
e   x       x       store   skip
f   x       x       store   skip
*****************************************************************************%

TITLE "Type Code State Machine";

SUBDESIGN tcsm
(tc[3..0]   : INPUT;        % Received type code %
 flag       : INPUT;        % MSW-LSW select (RA0) %
 rdavail    : INPUT;        % Longword available %
 cnl7       : INPUT;        % Snapshot channel=7 %
 clk        : INPUT;        % clock %
 rst        : INPUT;        % reset %
 ram_done   : INPUT;        % RAM cycle done %
 ready      : OUTPUT;       % ready for the next frame %
 ram_req    : OUTPUT;       % RAM cycle request %
 sdbc_ena   : OUTPUT;       % Slow data block counter enable %
 fdbc_ena   : OUTPUT;       % Fast data block counter enable %
 snap_ena   : OUTPUT;       % Snapshot address counter enable %
 snap_rst   : OUTPUT;       % Snapshot address counter reset %
 q[1..0]    : OUTPUT;
)

VARIABLE
tc_sm: MACHINE OF BITS (q[1..0])
    WITH STATES (   tidle=H"0",
                    t1=H"1",
                    t2=H"2",
                    t3=H"3");

BEGIN

% State machine clock and reset%
tc_sm.clk=clk;
tc_sm.reset=rst;

% Outputs %
ready = tidle;

CASE tc[3..0] IS
    when H"0" => ram_req = t1;
    when H"1" => ram_req = t1;
    when H"4" => snap_ena = t2 & cnl7;ram_req = t1;
    when H"5" => snap_rst = t2 & !flag;ram_req = t1;
    when H"6" => sdbc_ena = t1 & flag;ram_req = flag & t2 # !flag & t1;
    when H"7" => fdbc_ena = t1 & flag; ram_req = flag & t2 # !flag & t1;
    when H"c" => ram_req = t1;
    when H"d" => ram_req = t1;
    when H"e" => ram_req = t1;
```

```
    when H"f" => ram_req = t1;
END CASE;

% state transitions %
TABLE
    tc_sm,rdavail,tc[3..0],flag, ram_done => tc_sm;

%   PS      RDA TC      FLAG    DONE    =>  NS %

% TIDLE %
    tidle,  0,  x,      x,      x       =>  tidle;
    tidle,  1,  x,      x,      x       =>  t1;

% T1 %
    t1  ,   x,  H"0",   x,      0       =>  t1;
    t1  ,   x,  H"0",   x,      1       =>  tidle;
    t1  ,   x,  H"1",   x,      0       =>  t1;
    t1  ,   x,  H"1",   x,      1       =>  tidle;
    t1  ,   x,  H"4",   x,      0       =>  t1;
    t1  ,   x,  H"4",   x,      1       =>  t2;
    t1  ,   x,  H"5",   x,      0       =>  t1;
    t1  ,   x,  H"5",   0,      1       =>  t2;
    t1  ,   x,  H"5",   1,      1       =>  tidle;
    t1  ,   x,  H"6",   0,      0       =>  t1;
    t1  ,   x,  H"6",   0,      1       =>  tidle;
    t1  ,   x,  H"6",   1,      x       =>  t2;
    t1  ,   x,  H"7",   0,      0       =>  t1;
    t1  ,   x,  H"7",   0,      1       =>  tidle;
    t1  ,   x,  H"7",   1,      x       =>  t2;
    t1  ,   x,  H"c",   x,      0       =>  t1;
    t1  ,   x,  H"c",   x,      1       =>  tidle;
    t1  ,   x,  H"d",   x,      0       =>  t1;
    t1  ,   x,  H"d",   x,      1       =>  tidle;
    t1  ,   x,  H"e",   x,      0       =>  t1;
    t1  ,   x,  H"e",   x,      1       =>  tidle;
    t1  ,   x,  H"f",   x,      0       =>  t1;
    t1  ,   x,  H"f",   x,      1       =>  tidle;
    t1  ,   x,  H"2",   x,      x       =>  tidle;
    t1  ,   x,  H"3",   x,      x       =>  tidle;
    t1  ,   x,  H"8",   x,      x       =>  tidle;
    t1  ,   x,  H"9",   x,      x       =>  tidle;
    t1  ,   x,  H"a",   x,      x       =>  tidle;
    t1  ,   x,  H"b",   x,      x       =>  tidle;

% T2 %
    t2  ,   x,  H"4",   x,      x       =>  tidle;
    t2  ,   x,  H"6",   x,      0       =>  t2;
    t2  ,   x,  H"6",   x,      1       =>  tidle;
    t2  ,   x,  H"7",   x,      0       =>  t2;
    t2  ,   x,  H"7",   x,      1       =>  tidle;
    t2  ,   x,  H"0",   x,      x       =>  tidle;
    t2  ,   x,  H"1",   x,      x       =>  tidle;
    t2  ,   x,  H"2",   x,      x       =>  tidle;
    t2  ,   x,  H"3",   x,      x       =>  tidle;
    t2  ,   x,  H"5",   x,      x       =>  tidle;
    t2  ,   x,  H"8",   x,      x       =>  tidle;
    t2  ,   x,  H"9",   x,      x       =>  tidle;
    t2  ,   x,  H"a",   x,      x       =>  tidle;
    t2  ,   x,  H"b",   x,      x       =>  tidle;
    t2  ,   x,  H"c",   x,      x       =>  tidle;
    t2  ,   x,  H"d",   x,      x       =>  tidle;
    t2  ,   x,  H"e",   x,      x       =>  tidle;
    t2  ,   x,  H"f",   x,      x       =>  tidle;

% T3 %
    t3  ,   x,  x   ,   x,      x       =>  tidle;
```

```
END TABLE;

END;
```

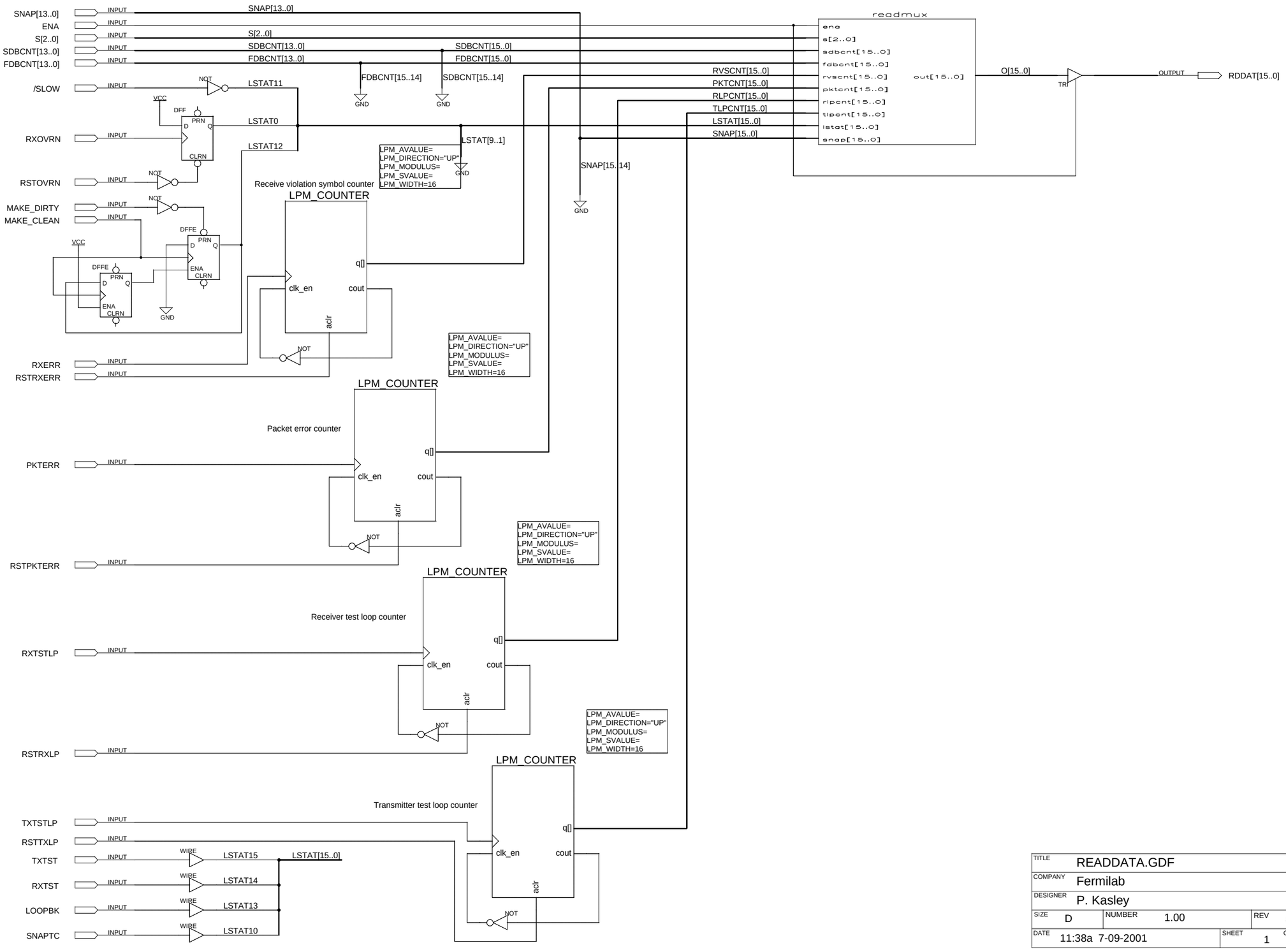


Snapshot address counter is reset to all 0's. It is postincremented when the channel number is 7.

RAM memory address is selected by the data type-code.

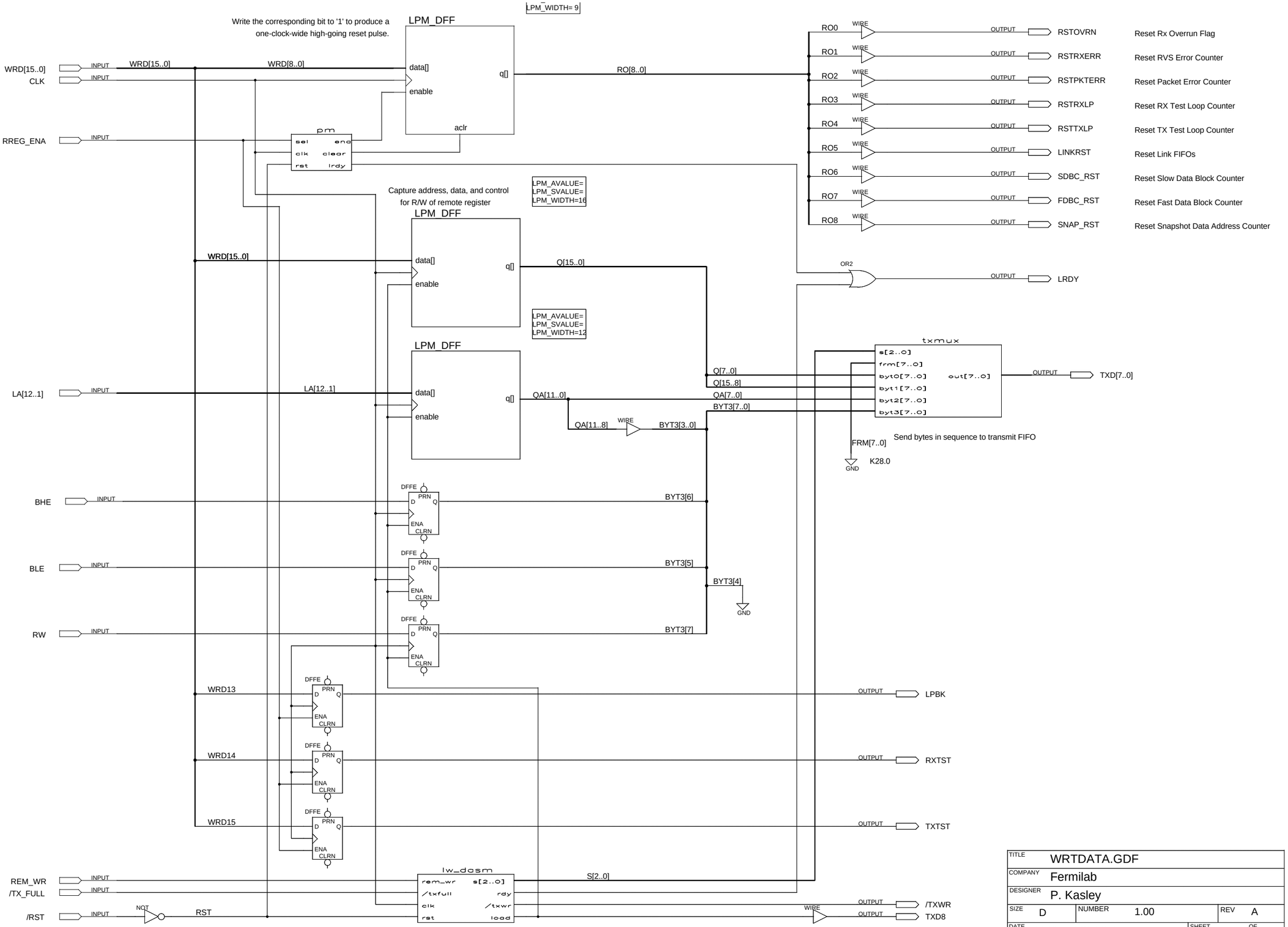
TITLE	LOCAL MEMORY ADDRESS GEN		
COMPANY	Fermilab		
DESIGNER	P. Kasleys		
SIZE D	NUMBER 1.00	REV A	
DATE 1:32p 8-05-2004	SHEET 1	OF 1	

```
%*****************************************************************************
DAS-PMC Hot Link Receiver Address Multiplexer 10 Aug 00 pak

24 bit wide by 6 port multiplexer. Four bits of received type code
directly drive the select inputs to route address sub-fields to the
RAM address bus.
*****************************************************************************%

TITLE "Receive Address Mux";

SUBDESIGN adrsmux
(s[3..0]    : INPUT;        % Select inputs %
 tc0[22..1] : INPUT;        % Type code 0 bus %
 tc1[22..1] : INPUT;        % Type code 1 bus %
 tc4[22..1] : INPUT;        % Type code 4 bus %
 tc5[22..1] : INPUT;        % Type code 5 bus %
 tc6[22..1] : INPUT;        % Type code 6 bus %
 tc7[22..1] : INPUT;        % Type code 7 bus %
 tcf[22..1] : INPUT;        % Type code C,D,E & F bus %
 out[22..1] : OUTPUT;       % output bus %
)

BEGIN

IF (s[]==0) THEN out[]=tc0[]; END IF;
IF (s[]==1) THEN out[]=tc1[]; END IF;
IF (s[]==4) THEN out[]=tc4[]; END IF;
IF (s[]==5) THEN out[]=tc5[]; END IF;
IF (s[]==6) THEN out[]=tc6[]; END IF;
IF (s[]==7) THEN out[]=tc7[]; END IF;
IF (s[]==7) THEN out[]=tc7[]; END IF;
IF (s[]==H"c") THEN out[]=tcf[]; END IF;
IF (s[]==H"d") THEN out[]=tcf[]; END IF;
IF (s[]==H"e") THEN out[]=tcf[]; END IF;
IF (s[]==H"f") THEN out[]=tcf[]; END IF;

END;
```



TITLE	READDATA.GDF		
COMPANY	Fermilab		
DESIGNER	P. Kasley		
SIZE D	NUMBER 1.00	REV A	
DATE 11:38a 7-09-2001		SHEET 1	OF 1

```
%*****************************************************************************
DAS-PMC Read Data Multiplexer 21Jun01 pak

16 bit wide by 7 port multiplexer routes data source to PLX local data bus.
*****************************************************************************%

TITLE "Read Data Mux";

SUBDESIGN readmux
(ena            : INPUT;        % Enable input %
 s[2..0]        : INPUT;        % Select inputs %
 sdbcnt[15..0]  : INPUT;        % Slow data block counter %
 fdbcnt[15..0]  : INPUT;        % Fast data block counter %
 rvscnt[15..0]  : INPUT;        % Code violation error counter %
 pktcnt[15..0]  : INPUT;        % Packet error counter %
 rlpcnt[15..0]  : INPUT;        % Receive test loop counter %
 tlpcnt[15..0]  : INPUT;        % Transmit test loop counter %
 lstat[15..0]   : INPUT;        % Link status register %
 snap[15..0]    : INPUT;        % Snapshot address counter %
 out[15..0]     : OUTPUT;       % Mux output %
)

BEGIN

IF s[]==0 & ena==1 THEN out[]=sdbcnt[]; END IF;
IF s[]==1 & ena==1 THEN out[]=fdbcnt[]; END IF;
IF s[]==2 & ena==1 THEN out[]=snap[]; END IF;
IF s[]==3 & ena==1 THEN out[]=rvscnt[]; END IF;
IF s[]==4 & ena==1 THEN out[]=pktcnt[]; END IF;
IF s[]==5 & ena==1 THEN out[]=rlpcnt[]; END IF;
IF s[]==6 & ena==1 THEN out[]=tlpcnt[]; END IF;
IF s[]==7 & ena==1 THEN out[]=lstat[]; END IF;

END;
```



TITLE	WRTDATA.GDF		
COMPANY	Fermilab		
DESIGNER	P. Kasley		
SIZE D	NUMBER 1.00	REV A	
DATE 11:32a 6-21-2001	SHEET 1	OF 1	

```
%*****************************************************************************
DAS-PMC Hot Link Receiver Long Word Dis-assembly
21 Jun 01 pak

Load the Hot Link transmtter with a sequence of 5 bytes after any PCI write of
a remote register.

While the framing byte is being loaded into the TX, capture the data and
address. Send four bytes. Signal LRDY and finish.
*****************************************************************************%

TITLE "Long Word Dis-assembly";

SUBDESIGN lw_dasm
(rem_wr     : INPUT;        % remote register write -> dump and run %
 /txfull    : INPUT;        % TX FIFO full flag %
 clk        : INPUT;        % clock %
 rst        : INPUT;        % reset %
 s[2..0]    : OUTPUT;       % Output mux select %
 rdy        : OUTPUT;       % hold-up to local bus sequencer %
 /txwr      : OUTPUT;       % TX FIFO write %
 load       : OUTPUT;       % Write data capture %
)

VARIABLE
lw_dasm: MACHINE OF BITS (q[2..0])
    WITH STATES (   idle=H"0",
                    frame=H"1",
                    byte0=H"3",
                    byte1=H"2",
                    byte2=H"6",
                    byte3=H"7",
                    lrdy=H"5",
                    unused=H"4"
                );

BEGIN
% State machine clock and reset %
lw_dasm.clk=clk;
lw_dasm.reset=rst;

% Outputs %
rdy = lrdy;
load = frame;
s[2] = q2;
s[1] = q1;
s[0] = q0;
/txwr = !((frame # byte0 # byte1 # byte2 # byte3) & /txfull);

% state transitions %
TABLE
    lw_dasm,rem_wr,/txfull => lw_dasm;

%   PS      REM_WR  /TXFULL =>  NS %

% IDLE %
    idle,   0,      x       =>  idle;
    idle,   1,      x       =>  frame;

% FRAME - %
    frame,  x,      0       => frame;
    frame,  x,      1       => byte0;

% BYTE0 %
    byte0,  x,      0       => byte0;
    byte0,  x,      1       => byte1;
```

```
% BYTE1 %
    byte1,  x,      0       => byte1;
    byte1,  x,      1       => byte2;

% BYTE2 %
    byte2,  x,      0       => byte2;
    byte2,  x,      1       => byte3;

% BYTE3 %
    byte3,  x,      0       => byte3;
    byte3,  x,      1       => lrdy;    % Rem wr: set LRDY & wait for end-of-cyc %

% LRDY %
    lrdy,   1,      x       => lrdy;
    lrdy,   0,      x       => idle;

% UNUSED %
    unused, x,      x       => idle;

END TABLE;

END;
```

```
%*****************************************************************************
DAS-PMC Hot Link Transmitter Multiplexer 10 Aug 00 pak

8 bit wide by 5 port multiplexer. Sequentially select each port to load the Hot-
Link transmit FIFO
*****************************************************************************%

TITLE "Transmit Mux";

SUBDESIGN txmux
(s[2..0]    : INPUT;        % Select inputs %
 frm[7..0]  : INPUT;
 byt0[7..0] : INPUT;
 byt1[7..0] : INPUT;
 byt2[7..0] : INPUT;
 byt3[7..0] : INPUT;
 out[7..0]  : OUTPUT;
)

BEGIN

IF s[]==1 THEN out[]=frm[]; END IF;
IF s[]==3 THEN out[]=byt0[]; END IF;
IF s[]==2 THEN out[]=byt1[]; END IF;
IF s[]==6 THEN out[]=byt2[]; END IF;
IF s[]==7 THEN out[]=byt3[]; END IF;

END;
```

```
%*****************************************************************************
DAS-PMC Reset Register Pulse Maker
31 Aug 00 pak
*****************************************************************************%

TITLE "Reset Register Pulse Maker";

SUBDESIGN pm
(sel        : INPUT;        % reset register select %
 clk        : INPUT;        % clock %
 rst        : INPUT;        % reset %
 ena        : OUTPUT;       % lpm_dff enable %
 clear      : OUTPUT;       % lpm_dff reset %
 lrdy       : OUTPUT;       % finished %
)

VARIABLE
pm: MACHINE OF BITS (zz[1..0])
    WITH STATES (   idle=H"0",
                    load=H"1",
                    rr=H"3",
                    wait=H"2"
                );

BEGIN
% State machine clock and reset%
pm.clk=clk;
pm.reset=rst;

% Outputs %
lrdy = wait;
ena = load;
clear = wait;

% state transitions %
TABLE
    pm, sel => pm;

%   PS      SEL     =>  NS %
    idle,   0       =>  idle;
    idle,   1       =>  load;

    load,   x       =>  rr;

    rr,     x       =>  wait;

    wait,   1       =>  wait;
    wait,   0       =>  idle;
 END TABLE;

END;
```

```
%*****************************************************************************
DAS-PMC RAM Cycle Control
23 Aug 00 pak

Rather than use the PLX arbiter, the RAM is dual-ported. This avoids a deadlock
on the PLX bus in which the PLX is stopped waiting for a return packet from the
remote chassis while the Hot-Link is stopped waiting for the PLX bus to dispose
of a data packet. It also avoids a design problem in the PLX whereby a bus
cycle in progress gets trashed if the PLX decides to preempt:

The 9050 spec states in several places, "If the PCI 9050-1 needs the
local bus before the external master has finished, LHOLDA is negated
(preempt condition).

1. External  Bus Master ("XBM") asserts LHOLD and waits for LHOLDA from
the 9050
2. 9050 tristates its outputs and asserts LHOLDA. This also tristates
the external driver on CS[3:0] and RD (per design note rev 1.1, Jan
1999).
3. XBM sees LHOLDA and asserts its buses and controls, including CS[3:0]
and RD. It continues to assert LHOLD.
4. 9050 decides it wants the local bus and removes LHOLDA while XBM is
still busy and has continued to assert LHOLD.
5. External driver is now enabled and trashes XBM's bus cycle.

Monitor the PLX bus and the Hot-Link bus for cycles directed to RAM. Give
priority to the Hot-Link, but do not interrupt a cycle already granted.

RDAV and Type_Code3=0 is the Hot-Link request. RDSAVED tells the recv logic
that the current word has been stored and a new word may be pulled from the
receive FIFO.

CS and (RD + WR) is the PLX request. LRDY is the PLX end-of-cycle signal.

4-bit Gray Seq: 01326754cdfeab980
*****************************************************************************%

TITLE "RAM Cycle";

SUBDESIGN ramcyc
(hl_req     : INPUT;        % Hot-Link RAM grant %
 plx_req    : INPUT;        % PLX RAM grant %
 plx_r      : INPUT;        % PLX read (Hot-Link is always write) %
 clk        : INPUT;
 rst        : INPUT;
 /we        : OUTPUT;       % RAM Write Enable (WR#) %
 /oe        : OUTPUT;       % RAM Output Enable (RD#) %
 plx_ae     : OUTPUT;       % RAM address enable %
 /plx_de    : OUTPUT;       % PLX data enable %
 hl_de      : OUTPUT;       % Hot-Link data enable %
 hl_ae      : OUTPUT;       % Hot-Link address enable %
 rdsaved    : OUTPUT;       % End-of-Hot_Link-cycle %
 lrdy       : OUTPUT;       % End-of-PLX-cycle %
)

VARIABLE
ramcyc: MACHINE OF BITS (q[3..0])
    WITH STATES (plx_idle = H"0",
                 hl_idle = H"8",
                 plx_ra = H"1",
                 plx_rb = H"3",
                 plx_rc = H"2",
                 plx_rd = H"6",
                 plx_wa = H"7",
                 plx_wb = H"5",
```

```
                    plx_wc = H"4",
                    plx_wd = H"C",
                    hl_a = H"E",
                    hl_b = H"A",
                    hl_c = H"B",
                    hl_d = H"9"
                );

BEGIN

% State machine clock and reset%
ramcyc.clk=clk;
ramcyc.reset=rst;

% Outputs %
plx_ae = !(q3 & !q2 # q3 & q1 & !q0);   % hl_idle # hl_a # hl_b # hl_c # hl_d %
/we = !(!q3 & q2 & !q1 # q3 & !q2 & q1); % plx_wb # plx_wc # hl_b # hl_c %
/oe = !(!q3 & q1 & plx_req);
/plx_de = !(plx_rb # plx_rc # plx_rd # plx_wb # plx_wc # plx_wd);
hl_de = hl_b # hl_c # hl_d;
hl_ae = !plx_ae;
rdsaved = hl_b;
lrdy = plx_rd # plx_wc # plx_wd;

% state transitions %
TABLE
    ramcyc,plx_req,hl_req,plx_r => ramcyc;

%   PS      PLXREQ  HLREQ   PLX_R   =>  NS %

% RAM IDLE %
    plx_idle,   0,      0,      x   => plx_idle;    % Parked at PLX %
    plx_idle,   1,      0,      0   => plx_wb;      % PLX write %
    plx_idle,   1,      0,      1   => plx_rb;      % PLX read %
    plx_idle,   x,      1,      x   => hl_a;        % Hot-Link priority %

    hl_idle,    0,      0,      x   => hl_idle;     % Parked at HL %
    hl_idle,    1,      0,      0   => plx_wa;      % PLX write %
    hl_idle,    1,      0,      1   => plx_ra;      % PLX read %
    hl_idle,    x,      1,      x   => hl_b;        % Hot-Link priority %

% PLX Read Cycle %

    plx_ra,     x,      x,      x   => plx_rb;      % Switch RAM inputs %
    plx_rb,     x,      x,      x   => plx_rc;      % Assert OE %
    plx_rc,     x,      x,      x   => plx_rd;      % Hold OE, assert RDY %
    plx_rd,     1,      x,      x   => plx_rd;      % Wait for PLXREQ to go away %
    plx_rd,     0,      x,      x   => plx_idle;    % Done %

% PLX Write Cycle %

    plx_wa,     x,      x,      x   => plx_wb;      % Switch RAM inputs %
    plx_wb,     x,      x,      x   => plx_wc;      % Assert WE %
    plx_wc,     x,      x,      x   => plx_wd;      % Finish WE, assert RDY %
    plx_wd,     1,      x,      x   => plx_wd;      % Wait for PLXREQ to go away %
    plx_wd,     0,      x,      x   => plx_idle;    % Done %

% Hot-Link Write Cycle %

    hl_a,       x,      x,      x   => hl_b;        % Switch RAM inputs %
    hl_b,       x,      x,      x   => hl_c;        % Assert WE %
    hl_c,       x,      x,      x   => hl_d;        % Finish WE, assert RDSAVED %
    hl_d,       0,      0,      x   => hl_idle;     % Done %
    hl_d,       x,      1,      x   => hl_b;        % Start into new cycle %
    hl_d,       1,      0,      1   => plx_ra;      % PLX read waiting %
    hl_d,       1,      0,      0   => plx_wa;      % PLX write waiting %
```

```
END TABLE;

END;
```