

DMA Write to Local - Transfers with HK - Messsysteme's PCI-Proto Lab/PLX

Manfred Ruebel,
University of the Federal Armed Forces Hamburg,
Germany

August 29, 2001

Abstract

A small state machine to be used with the PCI-Proto Lab/PLX is described. It can be implemented in the onboard EPLD to test DMA write transfers (PCI to local) under various conditions. The source code and the state diagram are included to enable users to modify the machine according to their application. The code together with the documentation can be used by anybody without any restrictions.

Contents

1 Purpose of the sample state machine	2
2 Environment	2
3 State Diagram	3
4 Using the State Machine	5
5 VHDL Code of EPLD	7
6 VHDL Code of Test Bench	7
7 Simulation Results	7

1 Purpose of the sample state machine

The sample state machine is primarily used to get familiar with the signals and registers of PLX's PCI 9054 chip. The PCI-Proto Lab/PLX board of HK - Messsysteme, Berlin, Germany, uses mode C (nonmultiplexed) of the PCI 9054. Because the direct slave example of PLX's Mon 2000 (Revision 3.1) doesn't run under Windows NT (our operating system on PCs), only DMA transfers are initiated. There is no digital signal source on the board, so only DMA writes to local are tested. Everything is only tested for 32 bit transfers.

The state machine assumes DMA transfers for $LA[3..2] = 01$; this implies a local fifo with one local address¹. It is controlled by an internal transfer counter (`tr_cnt`) and the PCI 9054 signals. The following signal flows can be observed by means of a logic state analyzer²:

1. transfers with and without PCI 9054 initiated wait cycles,
2. transfers with intermediate new address cycles initiated by PCI 9054 (`blast#`-signal followed by new address cycle)
3. transfers with a locally requested new address cycle (`bterm# = low`),
4. signal flow with `ready#`-signal deasserted by state machine for one or 2 `1clk` cycles, with and without simultaneously asserted `wait#` by PCI 9054,
5. DMA burst abort by assertion of `dmpafeot#` with and without simultaneously asserted `wait#` by PCI 9054,
6. local bus request by assertion of `brequi`,
7. DMA transfer with burst not enabled.

Note: The primary purpose of the code example is to see local bus signals, therefore the on board input and output latches are discarded.

2 Environment

- The PCI 9054 is programmed by PLX Mon 2000, version 3.1.
- The code was developed using Modeltech VHDL compiler and simulation environment.
- Synthesis was done using Exemplar's Leonardo Spectrum Level 2.
- Fitting and download into EPLD M4A3-64-32 was done using Lattice's Starter Kit.
- Signal flow was observed by a logic state analyzer which was connected to nearly all local signals.

¹The decision for a DMA transfer is made each time `ads#` is asserted. If you forget to set the constant address mode bit in the `DMAMODE` register or if you set a local address with another bit pattern at `LA[3..2]`, the state machine assumes a non DMA transfer!

²To see all variations, several data acquisitions with different register programming of the PCI 9054 are necessary.

3 State Diagram

Important remark: Because I like to see asserted Signals as '1', all lowactive signals are inverted and renamed (i.e. blast# is declared as pin blastn and, after inverting, named blast which is used in the VHDL code and the state diagram).

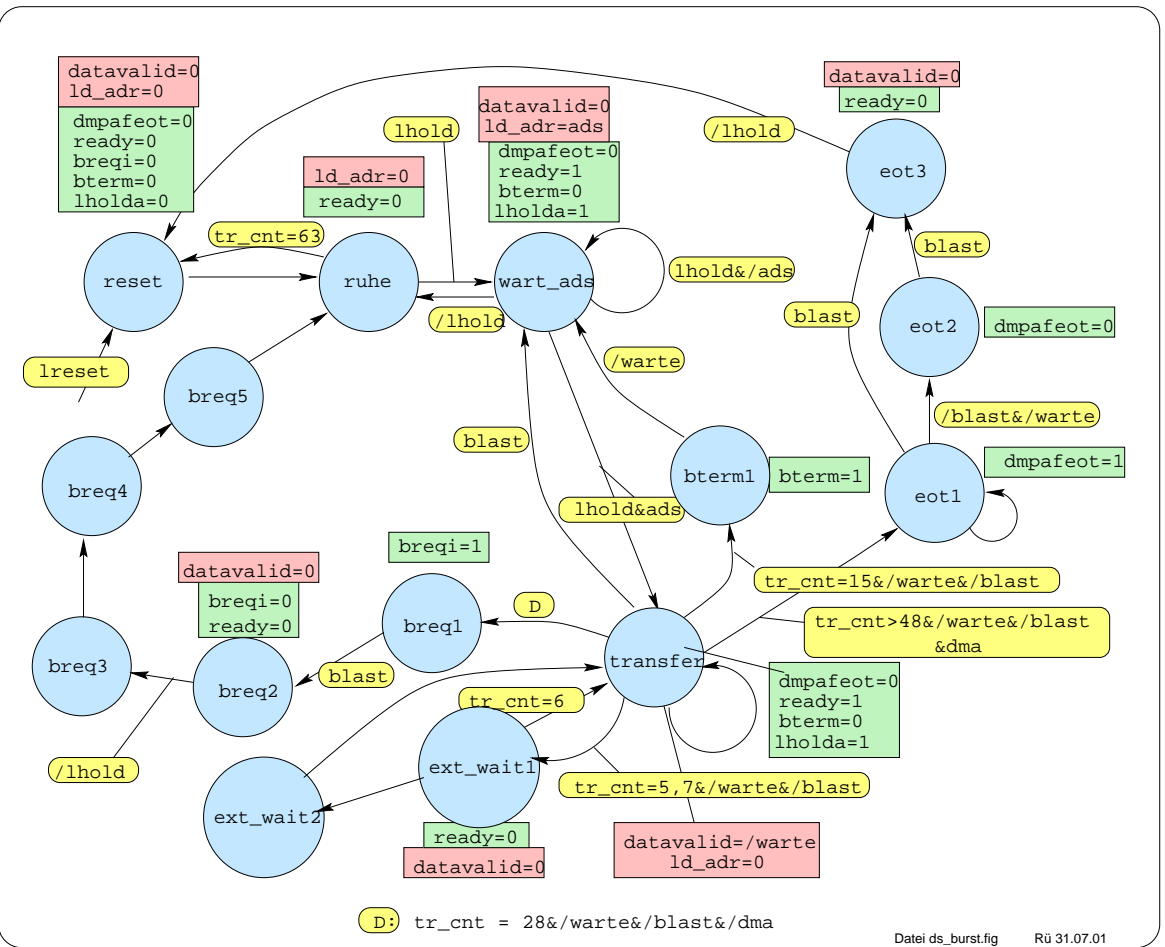


Figure 1: diagram of the sample state machine; output values (mealy (red) or moore/register type (green)) are only shown, if they change by a transition to the state, see VHDL code!

A transfer counter (`tr_cnt`), which controls some of the branches in the state machine, is implemented in the EPLD. At various counter values different actions are taken to show different signal dependencies on the local bus.

Description of the states (see Fig. 1):

reset The state machine is asynchronously reset to the `reset` state by the `lreset#`-Signal. Some internal and external control signals are set to their initial values. Additionally, the internal transfer counter (`tr_cnt`) is resetted. Because the different branches in the state machine are controlled by the actual value of `tr_cnt`, it is important to reset the counter each time before a new transfer is initiated.

Transitions to the `reset` state are therefore done in the following cases:

1. Applying `lreset#` as indicated above; you need to push the **Reset Board** button of the PLX Mon 2000 menu; **be aware, that this may change some bits in the PLX 9054, check each time those bits you manually changed!**
2. If a DMA transfer is pending, the state machine will abort it at `tr_cnt = 49`; transition to `reset` state is made to generate initial control signals without an external reset by `lreset#`.
3. If transfer is pending, which is interpreted as non DMA transfer (due to the selected `LA[3..2]` or if you forgot to select *hold local address constant* and a subsequent address cycle was executed with `LA[3..2] ≠ 01`), `tr_cnt` stops counting at a value of 63. After the PCI 9054 deasserts `lhold` the state machine returns to state `reset` for one `lclk` cycle.

Important: Do not set the `DMASIZ` register to values significantly greater than $4 * 64 = 256 = 0x100$ bytes, because the counter might have some arbitrary value if `lhold` is intermediately deasserted by the PCI 9054 due to other activities on the PCI bus of your computer system after `tr_cnt` has reached the uppermost value 63. A subsequent transfer may not show the same diagram on the logic state analyzer in this case. (Remember: The example is only for test purposes, it is not a final solution!!)

ruhe Waiting for `lhold` to be set by PCI 9054. In this special state machine `ruhe` and `reset` must be separate states: The transfer counter must not be resetted in state `ruhe`, because `ruhe` is also an intermediate state after the PCI 9054 has released the local bus following a `brequi` assertion.

wart_ads (wait for address strobe) This state is reached after a local bus request by the PCI 9054 (`lhold = 1`), after burst termination by the PCI 9054 (`blast# = low`), after a locally requested burst termination (`bterm# = low`), and after a local bus request (`brequi = 1`). The internal register `dma` is set, if the address `LA[3..2] = "01"` is valid during asserted `ads#`.

transfer In this state most of the transfers are done. Wait cycles asserted by the PCI 9054 `wait#` signal set internal signal `datavalid` to '0'. `Datavalid` is only used here to enable / stop the transfer counter, in normal systems

it must be used to inhibit data clocking, for example it can be used as a write enable signal for an on board synchronous fifo.

ext_wait1, ext_wait2 If the transfer counter has the value 5 or 7, neither **wait#** nor **blast#** are asserted, then **ready#** is asserted for one lclk cycle (**tr_cnt** = 5) or 2 lclk cycles (**tr_cnt** = 7), resp..

bterm1 Burst terminate state. Transition to this state is made if a burst is pending, the transfer counter has the value 15 and **wait#** is not asserted. If single transfers are done (burst enable bit in PCI 9054 not set) **blast#** is asserted with each data. **Blast** has priority, so the transition to **bterm1** is not possible in this case.

Bterm1 is hold, when PCI 9054 independently asserts **wait#** together with the transition to state **bterm1**. In this case in normal systems the actual valid data must be captured simultaneously with the transition to state **wart_ads**, controlled by **datavalid** = $\overline{\text{warte}}$. The PCI 9054 accepts **bterm#** if any burst is pending (i. e. not only DMA bursts), therefore the transition to **bterm1** doesn't depend from the register **dma**.

eot1, eot2, eot3 States to request an abort of the current DMA transfer. Transition to **eot1** is made if a DMA transfer has been recognized (register **dma** = 1), **wait#** is not asserted and **tr_cnt** > 48. Because the PCI 9054 might independently assert **wait#** with the transition to **eot1**, valid data must be transferred in **eot1** and **eot2**; with the last transfer in state **eot2** **blast#** is asserted by the PCI 9054. **Dmpaf/eot#** is set for one lclk cycle, it is extended for each wait cycle inserted by PCI 9054. **It may be not necessary** to keep **dmpaf/eot#** as short as possible and state **eot2** could be removed (comparable to signal **breqi** in the **breq** branch of this state machine). If **blast#** is asserted independently with the transition to state **eot1**, state **eot2** is not taken. Signals **blast#** and **wait#** are not asserted simultaneously!

breq1 The first state in the branch for a local bus request. Transition to this state is made if a DMA transfer is pending, and **blast#** and **wait#** are not asserted. In this state additional transfers are executed until the PCI 9054 deasserts **lhold**.

breq2 .. breq5 four lclk cycles replacing local bus activities for which the request is made; here nothing is done locally. Local bus requests by PCI9054 (**lhold** = 1) must not be served until local bus activities are finished.

4 Using the State Machine

1. Program the EPLD by means of an external programmer and insert it into the socket or program it via the download cable (JEDEC file by HK Messsysteme, Berlin ore generated by synthesizing the VHDL code).
2. If you want to abort a DMA transfer, a connection between pin 4 of the EPLD and signal **DMPAF/EOT#** (at connector JP48 of the board) must be soldered. All other pins are selected as hard wired on the board.

3. Start PLX Mon 2000.
4. Push the **Reset Board** button of the PLX Mon 2000 menu.
5. Type **vars** (variables) in the command line section of the PLX Mon 2000 menu. One of the values you get is the physical address of the host buffer. **Important: This physical address may change with the next boot process of the operating system!**
6. Menu **LCR** (Local Configuration Register) → **Mode DMA arbitration** button → set **Local Bus BREQ Enable**.
7. Menu **Memory** → select **DMA buffer** → **Memory Fill** → select **Fill with sequential values** → set **Increment** to 1 → set **Size in bytes** to 200 → **Fill Memory**.
 Setting the increment value to 1 allows you to see the number of the individual data transfer on the data bus (LD[31..0]). But be aware that **tr_cnt** represents the number of completed transfers, so its actual value will be the value on the data lines minus one!
8. Set the trigger menu of your logic state analyzer (may be **ads# = low**).
9. Menu **DMA**
 - **DMA Channel 0 Mode Details** → set **Enable Ready# Input**, **Enable Bterm# input**, **Enable bursting** and **Enable EOT input pin** and **Keep Local Address constant**. Attention: If you don't set a pin to be used as input you may not see the signal! Example: If you don't set **Enable EOT input pin**, the output of the PCI 9054 overrides the **dmpaf/eot#** output of the EPLD.
 - Set the **PCI Address** to the physical address you got by typing **vars** (above).
 - Set **Local Address** to 00000004; this is a byte address and will set **LA[3..2] = "01"** to be recognized as a DMA transfer address.
 - Set **Transfer Size** to 100 (i. e. 64 transfers with 32 bit each).
 - Set **Data Transfer Enable**
 - Press **Start Transfer**
10. Play with different options of the register settings of the PCI 9054! Examples: insert wait cycles of PCI 9054, disable bursting, set other local addresses, not long word matching byte addresses, local latency timer, local pause timer, etc.

5 VHDL Code of EPLD

```
-----
file ds_burst.vhd
-----
--
-- The code describes a simple state machine to test various modes of --
-- HK's development board PCI-Proto Lab/PLX with PLX's PCI9054 chip. --
-- Details of the state machine see separate description --
-- Users are free to modify the code for their own application --
-- without any restrictions --
-- Use of the code or part of the code are on the responsibility of --
-- the user only, no warranty is made for anything --
--
-- Author: Manfred Ruebel, Universitaet der Bundeswehr Hamburg, --
-- D-22039 Hamburg, email address Manfred.Ruebel@unibw-hamburg.de --
--
-- environment: modeltech VHDL, leonardo level 2 synthesis, lattice starter --
-- and download software --
-- pin constraints are made after importing --
-- the synthesized edif netlist --
-----

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ds_burst IS
PORT(adsn, -- address strobe, active low,
-- dacn0, -- DMA0 acknowledge, low active
-- breqo, -- burst request output (9054 output!)
lclk, -- local clock
lhold, -- local hold, active high
lresetn, -- local reset, active low
blastn, -- burst last, low active
lwrdn, -- local write/read, read --> low
waitn
: IN STD_LOGIC;
lben
: IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- local byte enable,
-- low active
la
: IN STD_LOGIC_VECTOR (3 DOWNTO 2); -- local address

brequi, -- burst request input (9054 input)
-- dreqn0, -- DMA0 request, low active
dmpafeotn, -- end of transfer used here (9054 input)
btermn, -- burst terminate, active low
lholda, -- local hold acknowledge, high active
readyn
: OUT STD_LOGIC;
ole
: BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0); -- output latch
-- enable, active high
ilen
: BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0) -- output latch
);

END ds_burst;

ARCHITECTURE fsm OF ds_burst IS

TYPE zustaende IS (reset,
ruhe, -- waiting for new activities
wart_ads, -- wait for address strobe
transfer, -- data transfers
ext_wait1, -- generate "not ready"
ext_wait2, -- additional "not ready" state if tr_cnt = 7
```

```

bterm1,    -- set burst terminate, keep state if PCI9054
           -- has wait-signal set, actual data on LD must
           -- be transferred
eot1,      -- force end of transfer if DMA is active, keep
           -- state if PCI9054 has wait-signal set, actual
           -- data on LD must be transferred
eot2,      -- remove EOT#
eot3,      -- wait for DMAC deasserting lhold
breq1,     -- force bus request if DMA is active,
           -- transfer pending data and wait for lhold
           -- to get deasserted
breq2,     -- deassert breq1 and readyn
breq3, breq4, breq5); -- 3 arbitrary additional states

SIGNAL zustand, folgezustand : zustaende;
SIGNAL ads, blast, lreset, warte, ready,
       bterm, dmpafeot,
       dma : STD_LOGIC;      -- dma = 1 if DMA is active
SIGNAL lbe : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL dma_addr: STD_LOGIC ; -- '1' if DMA-Address at LA
SIGNAL tr_cnt : INTEGER RANGE 0 TO 63; -- different branches
           -- of the state machine are controlled by
           -- actual value of this counter
SIGNAL datavalid, ld_adr : STD_LOGIC := '0';

BEGIN
invert_lowactive_inputs: -- only because I like signals to have
                        -- value '1' if they are asserted
PROCESS (adsn, blastn, lben, lresetn, waitn)
BEGIN
ads <= NOT adsn; blast <= NOT blastn;
lreset <= NOT lresetn; warte <= NOT waitn;
FOR i IN 0 TO 3 LOOP lbe(i) <= NOT lben (i); END LOOP;
END PROCESS invert_lowactive_inputs;

dma_addr <= '1' WHEN (la = "01") ELSE '0';

nextstatedecoder:
PROCESS (ads, tr_cnt, blast, lhold, dma, warte, zustand)
BEGIN
CASE zustand IS
WHEN reset => folgezustand <= ruhe;
WHEN ruhe =>
IF tr_cnt = 63 THEN folgezustand <= reset ;
ELSIF (lhold = '1') THEN folgezustand <= wart_ads;
ELSE folgezustand <= ruhe;
END IF;

WHEN wart_ads => -- wait for ads or /lhold
IF (lhold = '0') THEN folgezustand <= ruhe;
ELSIF (ads = '0') THEN folgezustand <= wart_ads;
ELSE folgezustand <= transfer;
END IF;

WHEN transfer => -- data transfers, dependencies of
                -- tr_cnt values are
                -- arbitrarily selected
IF (blast = '1') THEN folgezustand <= wart_ads;
ELSIF warte = '1' THEN folgezustand <= transfer;
ELSIF ((tr_cnt = 5) OR (tr_cnt = 7))
THEN folgezustand <= ext_wait1;
ELSIF (tr_cnt = 15) THEN folgezustand <= bterm1;
ELSIF (tr_cnt = 28) AND (dma = '1') THEN
folgezustand <= breq1; -- only if dma is active
ELSIF (tr_cnt > 48) AND dma = '1'
THEN folgezustand <= eot1; -- abort current dma

```

```

        ELSE folgezustand <= transfer;
    END IF;

    WHEN ext_wait1 => -- deassign ready, forcing PCI 9054 to wait
        IF tr_cnt = 6 THEN folgezustand <= transfer;
        ELSE folgezustand <= ext_wait2;
        END IF;

    WHEN ext_wait2 => -- ready deassigned for a second lclk period
        folgezustand <= transfer;

    WHEN bterm1 => IF warte = '0' THEN folgezustand <= wart_ads;
        -- generate burst terminate
        ELSE folgezustand <= bterm1; -- bterm not accepted
    END IF; -- by PCI 9054 during self generated wait cycle
        -- bterm forces new address strobe,
        -- does not terminate whole transfer
        -- pending write LD-data must be accepted!!

    WHEN eot1 => IF blast = '1' THEN folgezustand <= eot3;
        ELSIF warte = '0' THEN folgezustand <= eot2;
        ELSE folgezustand <= eot1; -- eot not accepted
    END IF; -- during wait cycle
        -- eot only to abort dma transfers,
        -- not for non dma transfers

    WHEN eot2 => IF blast = '1' THEN folgezustand <= eot3;
        ELSE folgezustand <= eot2; -- keep state if
    END IF; -- wait cycle pending
        -- blastn may be set

    WHEN eot3 => IF lhold = '1' THEN folgezustand <= eot3; -- wait for
        -- PCI 9054 releasing local bus
        ELSE folgezustand <= reset; -- transition to reset
        -- burst counter here, otherwise reset and ruhe
        -- may be one state
    END IF;

    WHEN breq1 => IF blast = '0' THEN folgezustand <= breq1; -- wait for
        -- local bus request to be accepted by PCI9054
        ELSE folgezustand <= breq2; END IF;

    WHEN breq2 => IF lhold = '1' THEN folgezustand <= breq2;
        ELSE folgezustand <= breq3; END IF;

    WHEN breq3 => folgezustand <= breq4;

    WHEN breq4 => folgezustand <= breq5;

    WHEN breq5 => folgezustand <= ruhe ; -- local bus activities may
        -- follow, lhold requests must not be answered
        -- until local activities are finished
    END CASE;
END PROCESS nextstatedecoder;

transitions: PROCESS (lclk, lreset, folgezustand)
BEGIN
    IF lreset = '1' THEN -- asynchronous reset
        zustand <= reset;
    ELSIF lclk'EVENT AND lclk = '1' THEN -- rising clock edge
        zustand <= folgezustand;
    END IF;
END PROCESS transitions;

moore_and_ffs: PROCESS (ads, blast, folgezustand, dma_addr, lbe, lclk,
    lhold, dma, lreset, lwrtn, warte)
    VARIABLE dat_val : STD_LOGIC := '0'; -- latch- or clockenable

```

-- for I/O-latches on HK-board

```
BEGIN
IF lreset = '1' THEN -- asynchronous reset (active low)
  ready <= '0'; bterm <= '0'; lholda <= '0'; dmpafeot <= '0';
  dma <= '0'; brequi <= '0';
  ilen <= (OTHERS => '1'); ole <= (OTHERS => '0');
ELSIF lclk'EVENT AND lclk = '1' THEN -- rising clock edge
CASE folgezustand IS
  WHEN reset | ruhe => ready <= '0'; lholda <= '0'; bterm <= '0';
    dat_val := '0'; dmpafeot <= '0'; dma <= '0';
    brequi <= '0';
  WHEN wart_ads => ready <= '1'; lholda <= '1'; bterm <= '0';
    dat_val := '0'; dmpafeot <= '0'; brequi <= '0';
    dma <= '0';
  WHEN transfer => ready <= '1'; lholda <= '1'; bterm <= '0';
    dat_val := NOT warte; dmpafeot <= '0';
    brequi <= '0';
    IF (dma_addr = '1' AND ads = '1') THEN dma <= '1';
      ELSE dma <= dma;
    END IF; -- DMA transfers are characterized by
      -- accesses to LA 0 or 4 (Byteaddress!, FIFOs,
      -- local address increment deselected)
  WHEN ext_wait1 | ext_wait2 => ready <= '0'; lholda <= '1';
    bterm <= '0'; dat_val := '0'; dmpafeot <= '0';
    dma <= dma; brequi <= '0';
  WHEN bterm1 => ready <= '1'; lholda <= '1'; bterm <= '1';
    dat_val := NOT warte; dmpafeot <= '0';
    dma <= dma; brequi <= '0';
  WHEN eot1 => ready <= '1'; lholda <= '1'; bterm <= '0';
    dat_val := NOT warte; dmpafeot <= '1';
    dma <= dma; brequi <= '0';
  WHEN eot2 => ready <= '1'; lholda <= '1'; bterm <= '0';
    dat_val := NOT warte; dmpafeot <= '0';
    dma <= dma; brequi <= '0';
  WHEN eot3 => ready <= '0'; lholda <= '1'; bterm <= '0';
    dat_val := '0'; dmpafeot <= '0'; brequi <= '0';
    dma <= '0';
  WHEN breq1 => dat_val := NOT warte; dmpafeot <= '0'; bterm <= '0';
    brequi <= '1'; lholda <= '1'; dma <= dma; ready <= '1';
  WHEN breq2 => dat_val := '0'; dmpafeot <= '0'; bterm <= '0';
    ready <= '0'; brequi <= '0'; dma <= '0'; lholda <= '1';
  WHEN breq3 | breq4 | breq5
    => dat_val := '0'; dmpafeot <= '0'; bterm <= '0';
    ready <= '0'; lholda <= '0'; brequi <= '0';
    dma <= '0';
END CASE;
FOR i IN 0 TO 3 LOOP ole(i) -- generate output latch enable signals
  <= dat_val AND lbe(i) AND lwrnd; END LOOP;
FOR i IN 0 TO 3 LOOP ilen(i) -- generate input latch enable signals
  <= NOT (dat_val AND lbe(i) AND NOT lwrnd); END LOOP;
END IF;
END PROCESS moore_and_ffs;

mealy: PROCESS (zustand, warte, ads, blast)
BEGIN
CASE zustand IS
  WHEN reset | ruhe => ld_adr <= '0'; datavalid <= '0';
  WHEN wart_ads => ld_adr <= ads; datavalid <= '0';
  WHEN transfer | bterm1 | eot1 | eot2 | breq1
    => datavalid <= NOT warte; ld_adr <= '0';
  WHEN OTHERS
    => ld_adr <= '0'; datavalid <= '0';
END CASE;
END PROCESS mealy;

transfer_counter: PROCESS (lclk, zustand, datavalid) -- transfer counter to
```

```

-- generate various control signals to PCI 9054
-- at arbitrary selected counter values

BEGIN
  IF zustand = reset THEN tr_cnt <= 0;
  ELSIF lclk'EVENT AND lclk = '1' THEN
    IF datavalid = '1' THEN
      IF (tr_cnt < 63) THEN
        tr_cnt <= tr_cnt + 1;
      ELSE tr_cnt <= tr_cnt;
      END IF;
    END IF;
  END IF;
END PROCESS transfer_counter;

-- invert to low active outputs, same reason as for inputs, see above
--
readyn <= NOT ready;
btermn <= NOT bterm;
dmpafeotn <= NOT dmpafeot;

END fsm;--

```

6 VHDL Code of Test Bench

```
-----
file tb_ds_burst.vhd
-----
--
-- Testbench to ds_burst.vhd
-- the test bench doesn't cover all variations that are possible with
-- the HK Messsystems board including the EPLD which is programmed
-- with the circuitry defined by ds_burst.vhd
--
-- Author: Manfred Ruebel, University of the Armed Forces Hamburg,
-- Germany, Email address Manfred.Ruebel@UniBw-Hamburg.de
--
-- The code may be used by anybody without any restrictions
-- Using the code is to the responsibility of the user, no
-- warranty is made for anything
-----

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tb_ds_burst IS

END tb_ds_burst;

ARCHITECTURE rtl OF tb_ds_burst IS

COMPONENT ds_burst
  PORT (adsn,          -- address strobe, active low,
        -- dacn0,     -- DMA0 acknowledge, low active
        -- breqo,     -- burst request output (9054 output!)
        lclk,         -- local clock
        lhold,       -- local hold, active high
        lresetn,     -- local reset, active low
        blastn,     -- burst last, low active
        lwrdsn,     -- local write/read, read --> low
        waitn        -- wait, low active
        : IN STD_LOGIC;
        lben         : IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- local byte enable,
        -- low active
        la           : IN STD_LOGIC_VECTOR (3 DOWNTO 2);-- local address

        brequi,     -- burst request input (9054 input)
        -- dreqn0,    -- DMA0 request, low active
        dmpafeotn,  -- end of transfer used here (9054 input)
        btermn,     -- burst terminate, active low
        lholda,     -- local hold acknowledge, high active
        readyn      -- ready, low active
        : OUT STD_LOGIC;
        ole         : BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0); -- output latch
        -- enable, active high
        ilen        : BUFFER STD_LOGIC_VECTOR (3 DOWNTO 0) -- output latch
        );
        -- enable, active low
END COMPONENT ds_burst;

CONSTANT period : TIME := 50 ns ;

SIGNAL lhold      : STD_LOGIC := '0'; -- inputs to EPLD
SIGNAL lresetn, lwrdsn, adsn, blastn, lclk, waitn: STD_LOGIC := '1';
SIGNAL lben       : STD_LOGIC_VECTOR (3 DOWNTO 0) := (OTHERS => '1'); -- dto.
```

```

SIGNAL la          : STD_LOGIC_VECTOR (3 DOWNT0 2) := (OTHERS => '0');

SIGNAL lholda, readyn, btermn, dmpafeotn,
       brequi      : STD_LOGIC; -- outputs of EPLD
SIGNAL ole, ilen   : STD_LOGIC_VECTOR (3 DOWNT0 0); -- dto.

```

```
BEGIN
```

```

 uut: ds_burst
   PORT MAP (adsn => adsn, lclk => lclk, lhold => lhold, lresetn => lresetn,
            blastn => blastn, lwrdsn => lwrdsn, waitn => waitn, lben => lben,
            la => la, brequi => brequi, dmpafeotn => dmpafeotn,
            btermn => btermn, lholda => lholda, readyn => readyn,
            ole => ole, ilen => ilen);

```

```
takt: lclk <= NOT lclk AFTER period / 2 ;
```

```
stimuli: PROCESS
```

```
BEGIN
```

```

 WAIT FOR 4 ns ; -- external variables asynchronous
 lresetn <= '0' ; WAIT FOR 2*period ; -- reset of EPLD
 lresetn <= '1' ; WAIT FOR 2*period ;

 lhold <= '1' ; WAIT FOR period ;
 la (3 DOWNT0 2) <= "01" ;
 lben(3 DOWNT0 0) <= "0001" ; WAIT FOR period ; -- not aligned
 adsn <= '0' ; -- Address Strobe
 WAIT FOR period ;
 blastn <= '0' ; adsn <= '1' ; WAIT FOR period ; -- 1. Transfer
 blastn <= '1' ; -- burst broken, bus hold
 WAIT FOR 3 * period ;
 la (3 DOWNT0 2) <= "01" ; -- new address phase
 lben(3 DOWNT0 0) <= "0000" ; WAIT FOR period ; -- aligned
 adsn <= '0' ; WAIT FOR period ; -- Address Strobe
 adsn <= '1' ; waitn <= '0' ; WAIT FOR period ; -- before transfer
 waitn <= '1' ; WAIT FOR period ; -- data transfer 2
 waitn <= '0' ; WAIT FOR period ; -- during data transfer
 waitn <= '1' ;
 WAIT FOR 3*period ; -- data transfers 3 to 5
 waitn <= '0' ; WAIT FOR period ;-- during data transfer
 waitn <= '1' ;
 WAIT FOR 12*period ; -- 12 transfers executed, fsm waiting
 -- for new address strobe after bterm
 adsn <= '0' ; WAIT FOR period ;
 adsn <= '1' ; WAIT FOR 3*period; -- now 15 transfers completed,
 waitn <= '0' ; WAIT FOR period ; -- inserting wait cycle
 waitn <= '1' ; -- problems
 WAIT FOR 6*period; -- now 15 transfers completed,
 -- fsm in state wart_ads
 la (3 DOWNT0 2) <= "01" ;
 adsn <= '0' ; WAIT FOR period ;
 la (3 DOWNT0 2) <= "00" ;
 adsn <= '1' ; WAIT FOR 15*period ; -- 29 transfers completed,
 -- brequi asserted
 blastn<= '0' ; WAIT FOR period ; -- 9054 asserts blastn
 blastn<= '1' ; WAIT FOR period ;
 lhold <= '0' ; WAIT FOR 5*period ; -- 9054 deasserts lhold
 -- local bus activities
 lhold <= '1' ; WAIT FOR 2*period ; -- 9054 asserts lhold
 la (3 DOWNT0 2) <= "01" ;
 adsn <= '0' ; WAIT FOR period ;
 adsn <= '1' ; WAIT FOR 20*period ; -- xx transfers completed,
 -- 51 transfers completed,

```

```

-- fsm has signaled eot to abort the
-- dma transfer, waiting in state eot2
-- for lhold to get removed by PCI 9054

blastn<= '0' ; WAIT FOR period ;
blastn<= '1' ;
lhold <= '0'; WAIT FOR 20*period ;      -- bus released by PCI 9054
WAIT;
END PROCESS stimuli;

END rtl; --

```

7 Simulation Results

See the following pages! Not all variations are covered by the test bench!

Remember: The PLX 9054 is not simulated, it's output signals are defined as stimulus signals in the testbench (tbdma2.vhd). The stimuli represent my momentary understanding of the data book!

