

VME64X-TCS

VME64X_CHIP

Version: **V1012**

HEPHY VIENNA
ELEKTRONIK I

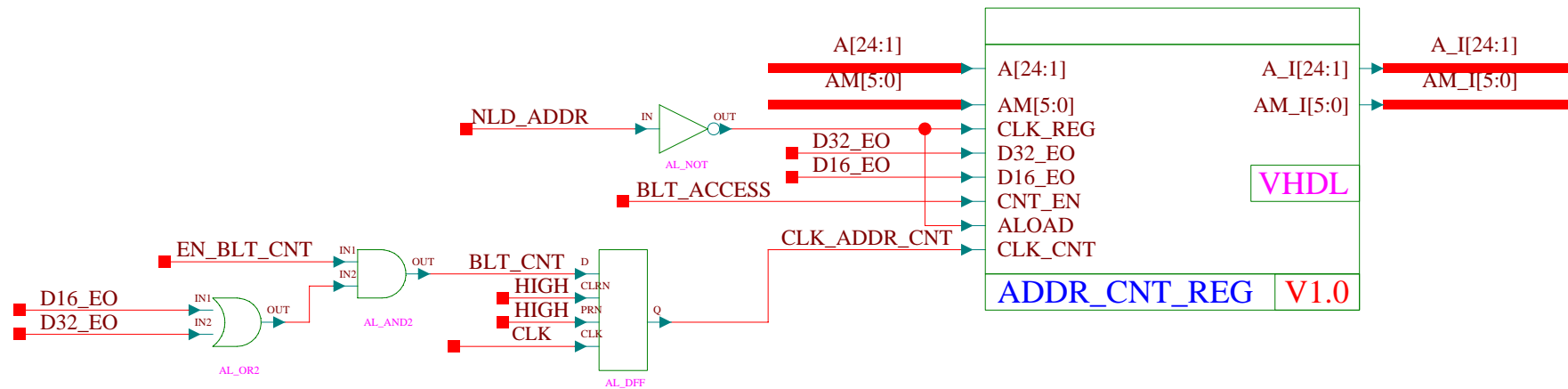
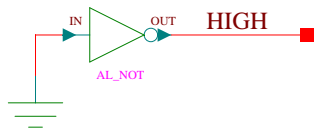
sheet **1** of **1**

modified by: HB

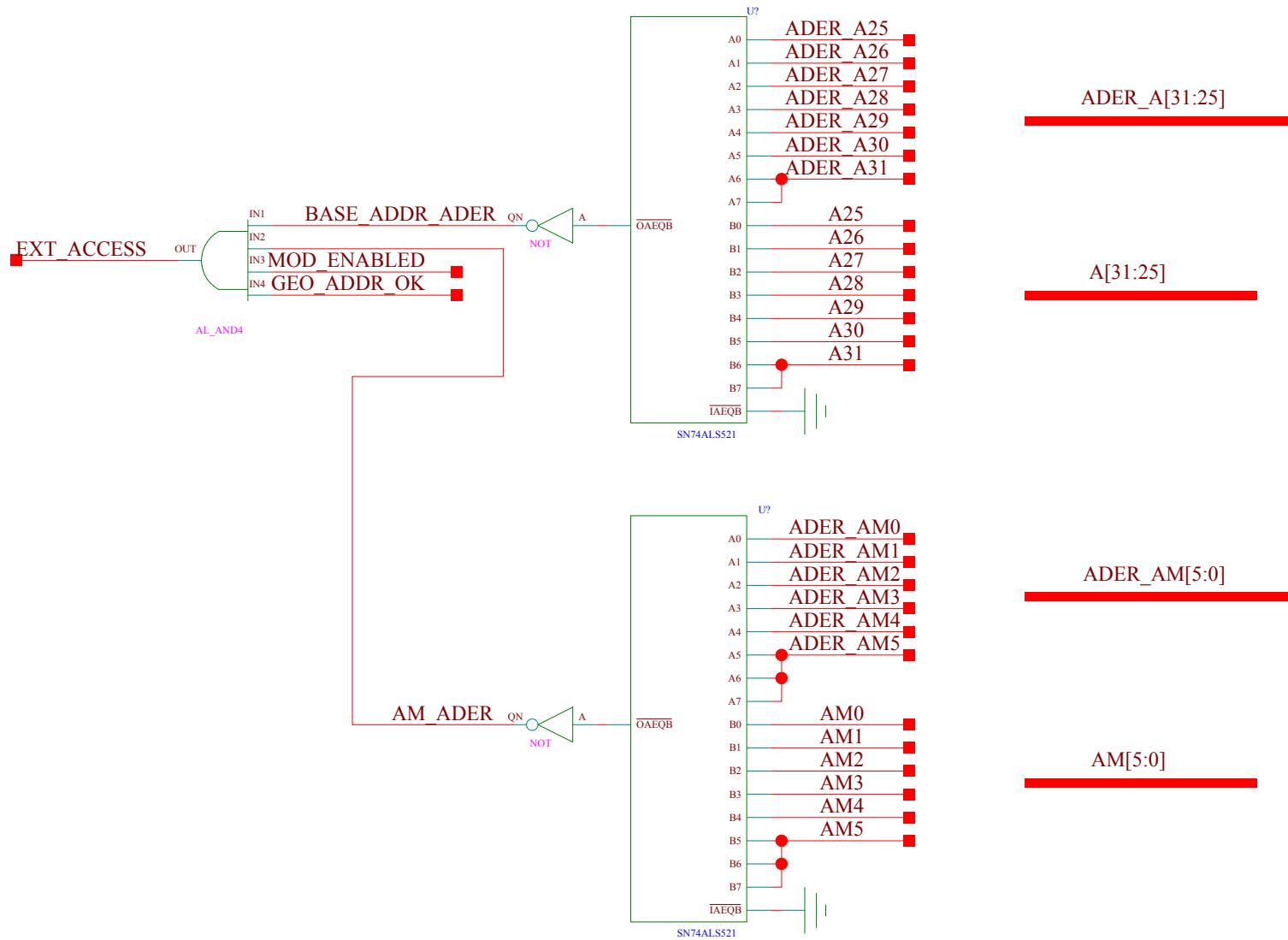
4-25-2007_12:59

checked by: CHECKER

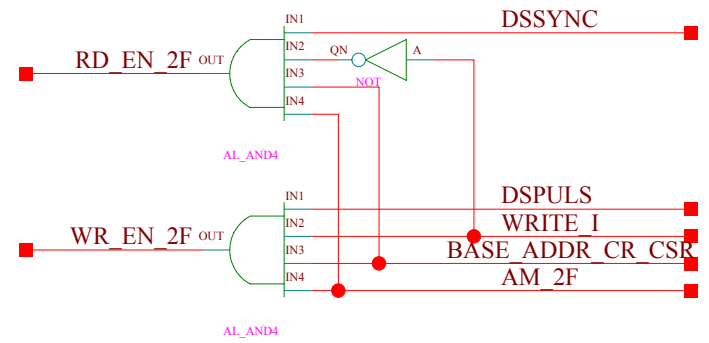
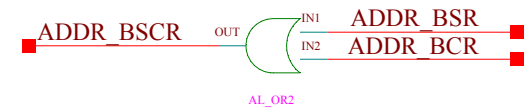
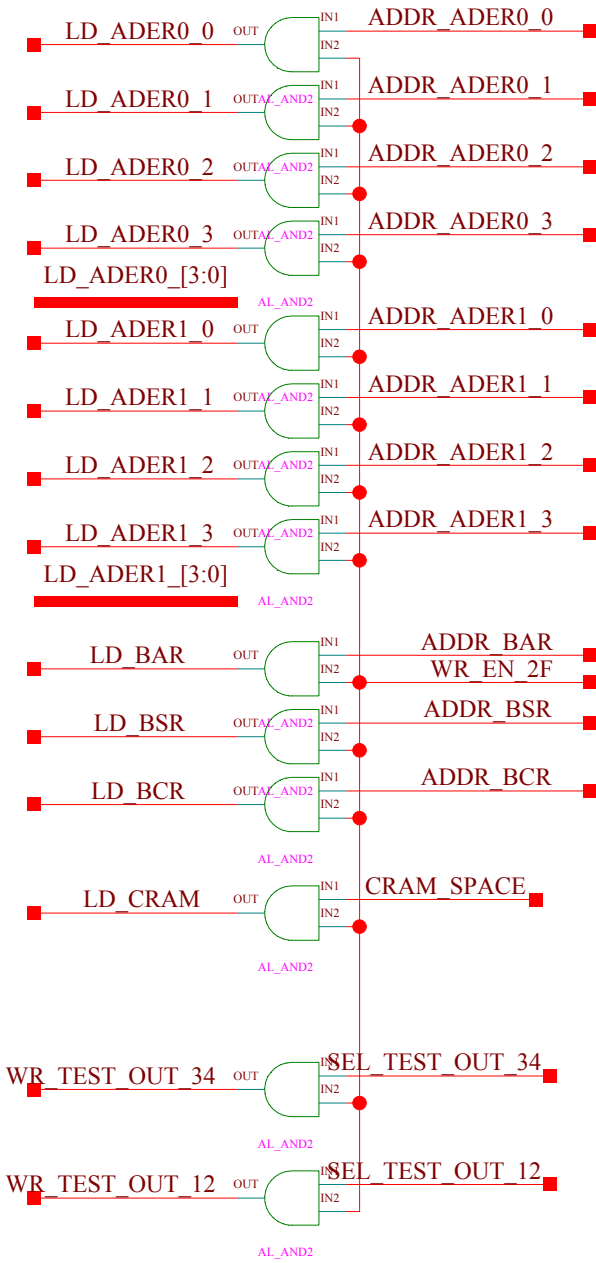
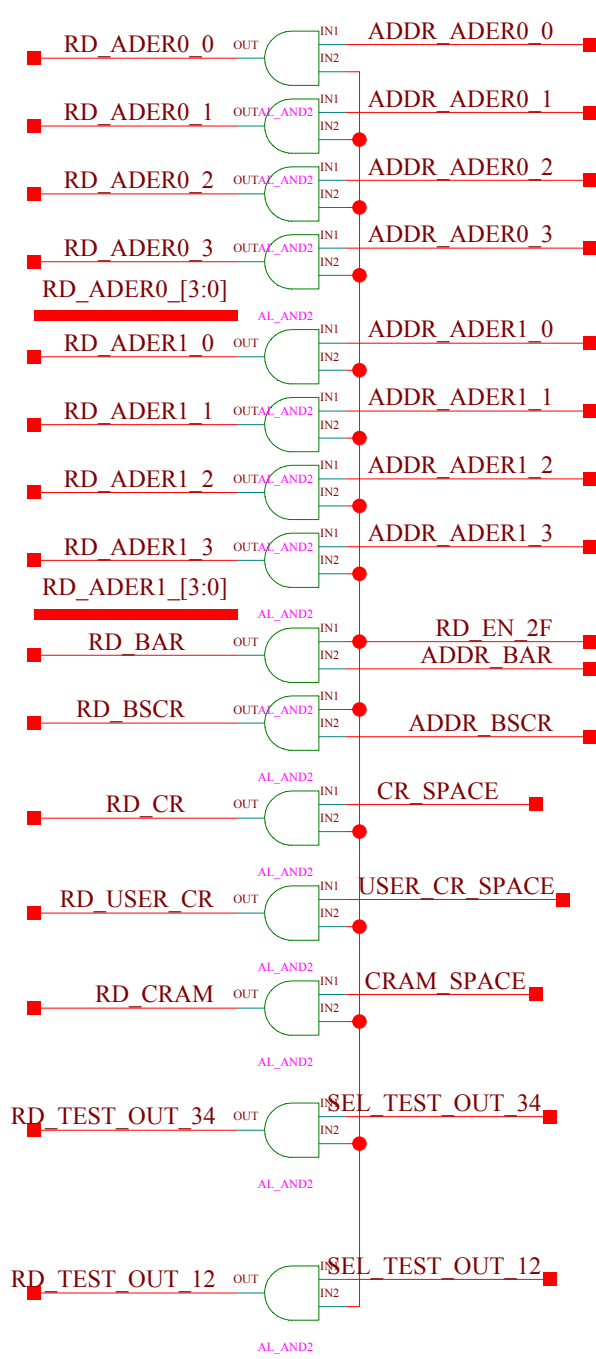
0-00-0000_00:00



VME64X-CHIP	
ADDRESSES_AM	
Version: V1.4	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	3-9-2007_15:49
checked by: CHECKER	0-00-0000_00:00



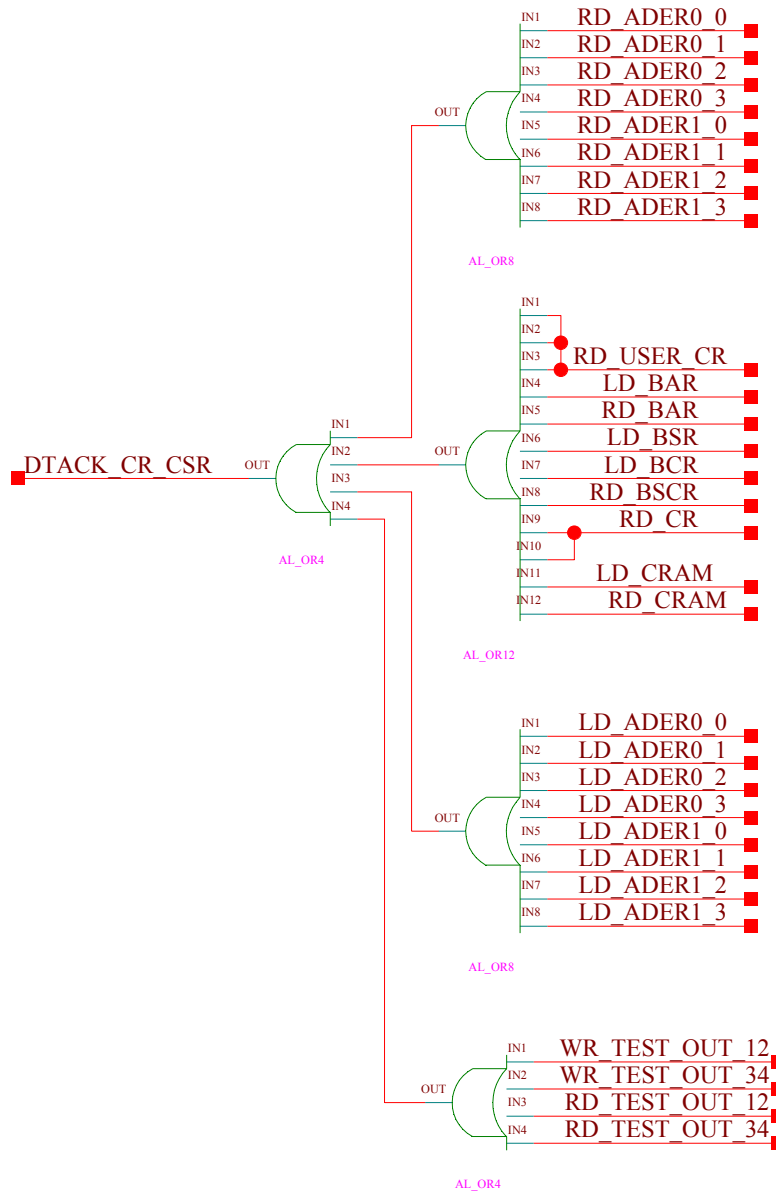
VME64X-CHIP	
ADER_EXT	
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	8-29-2005_9:32
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP

CR_CSR_INSTR

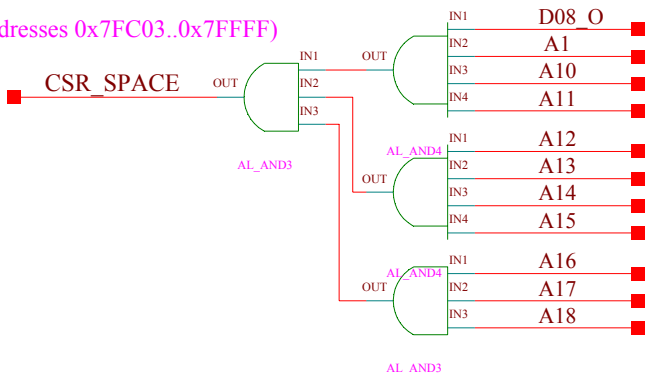
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00



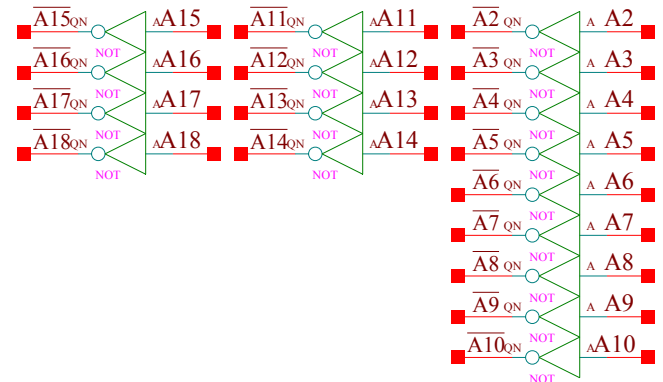
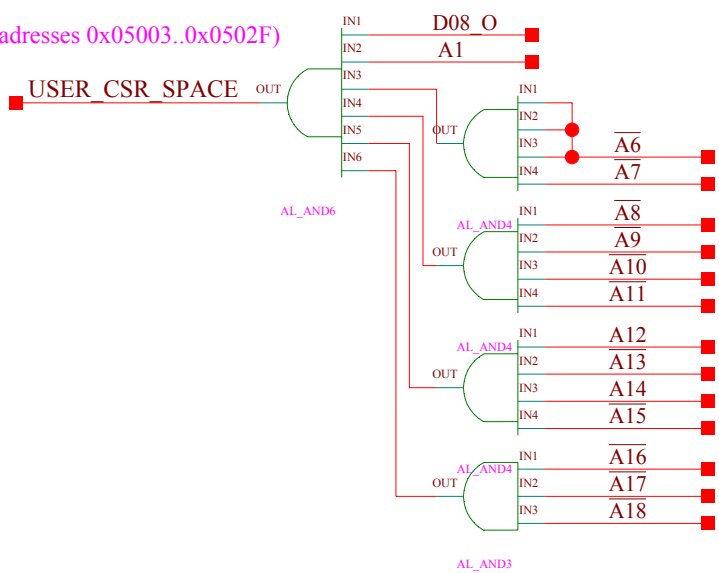
no BERR for CR/CSR access!!

<h1>VME64X-CHIP</h1>	
<h2>CR CSR INSTR</h2>	
Version:	V1.0
HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000 00:00

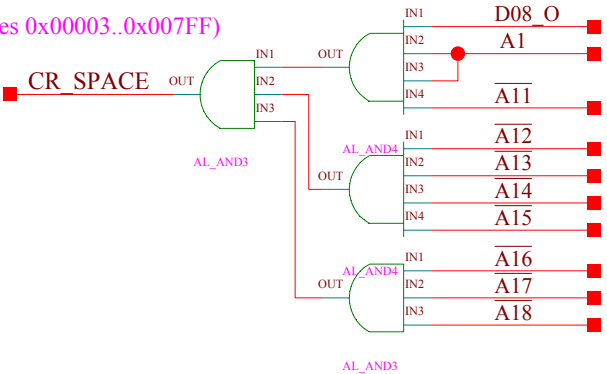
(addresses 0x7FC03..0x7FFF)



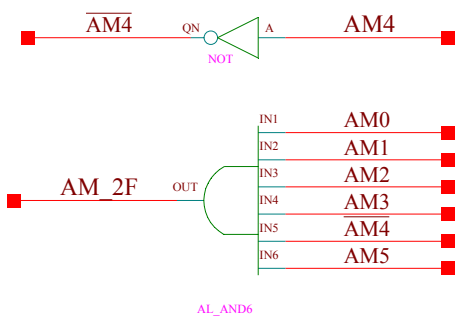
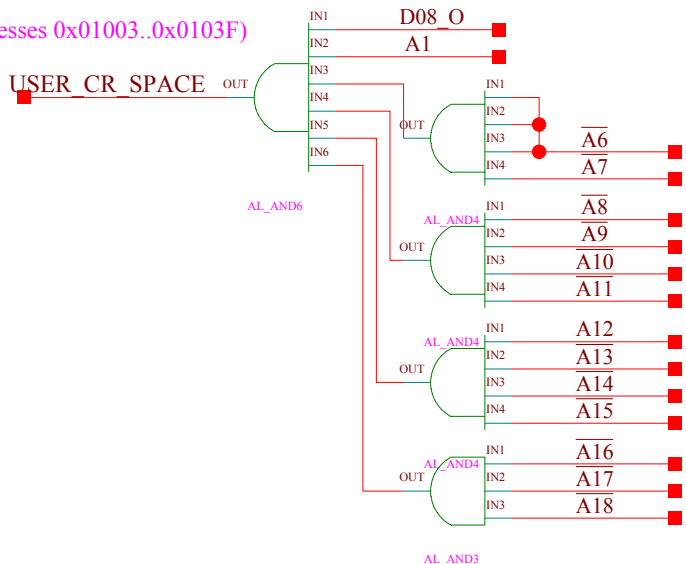
(addresses 0x05003..0x0502F)



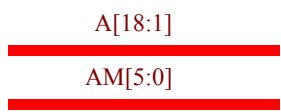
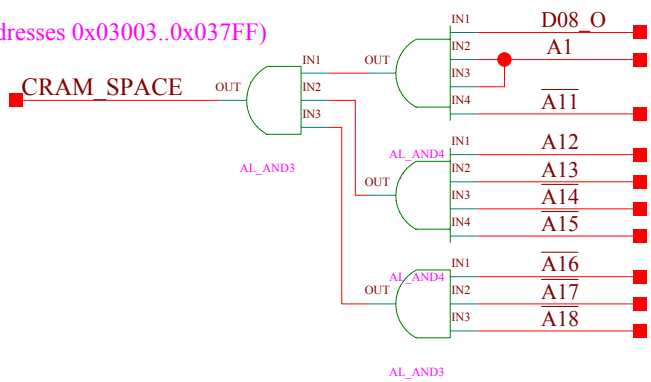
(addresses 0x00003..0x007FF)



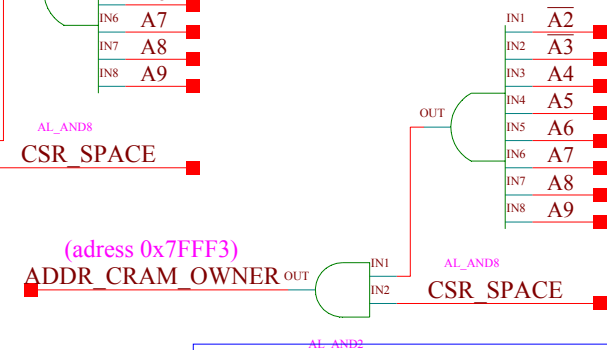
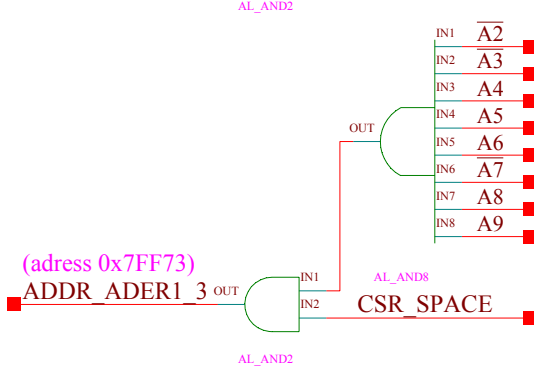
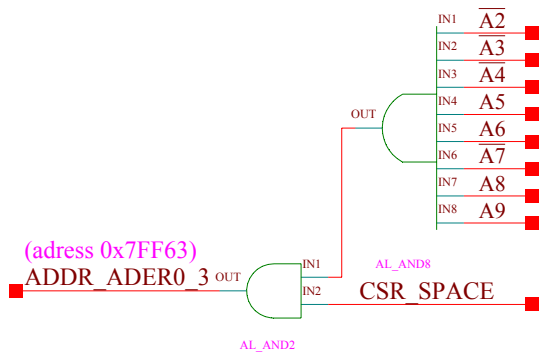
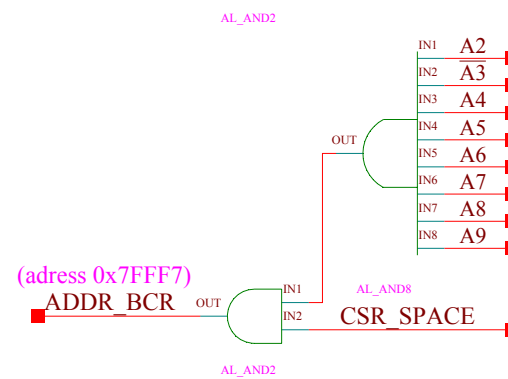
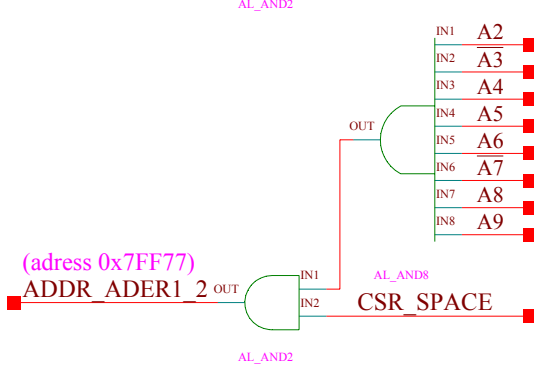
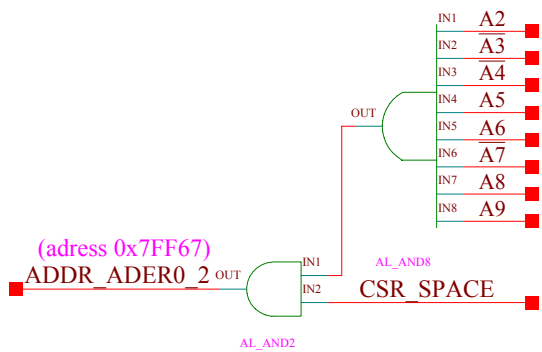
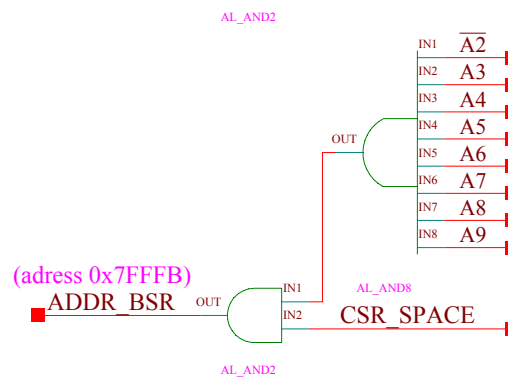
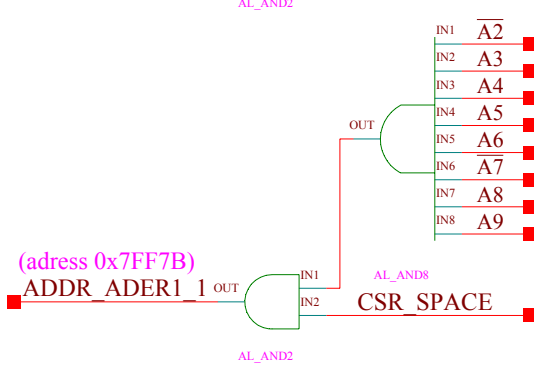
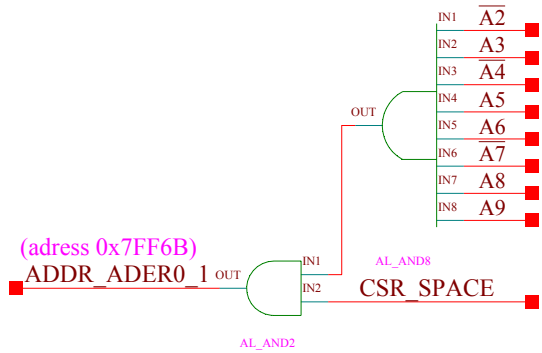
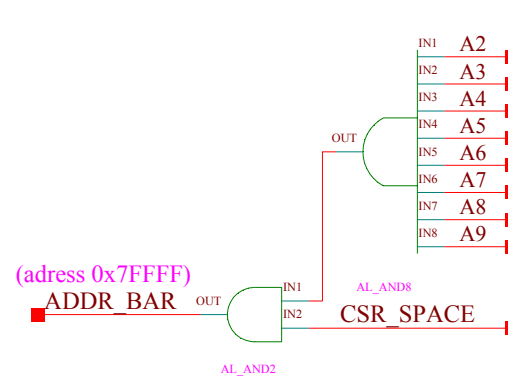
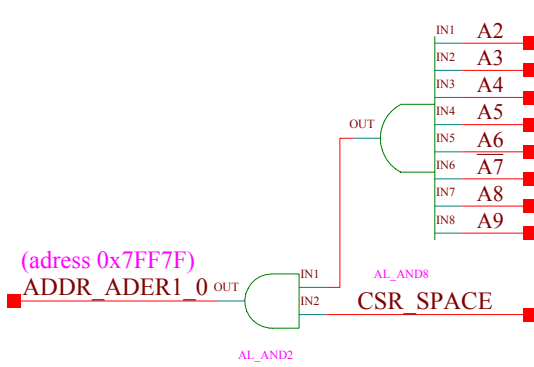
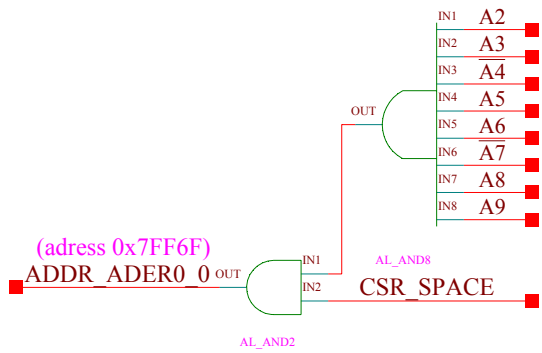
(addresses 0x01003..0x0103F)



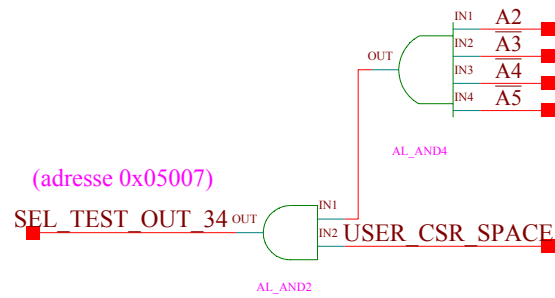
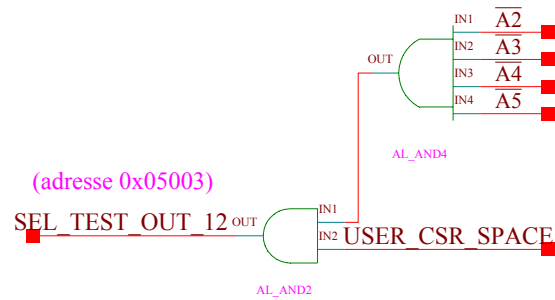
(addresses 0x03003..0x037FF)



VME64X-CHIP	
CR CSR INSTR	
Version:	V1.0
HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 5
modified by: HB	8-26-2005 13:54
checked by: CHECKER	0-00-0000 00:00



VME64X-CHIP	
CR CSR INSTR	
Version:	V1.0
HEPHY VIENNA ELEKTRONIK 1	sheet 4 of 5
modified by: HB	8-26-2005 13:54
checked by: CHECKER	0-00-0000 00:00



VME64X-CHIP

CR_CSR_INSTR

Version: **V1.0**

HEPHY VIENNA
ELEKTRONIK 1

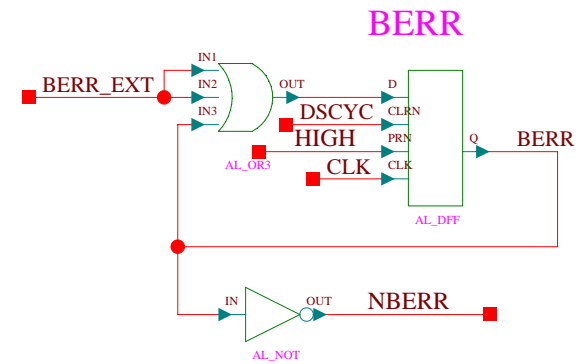
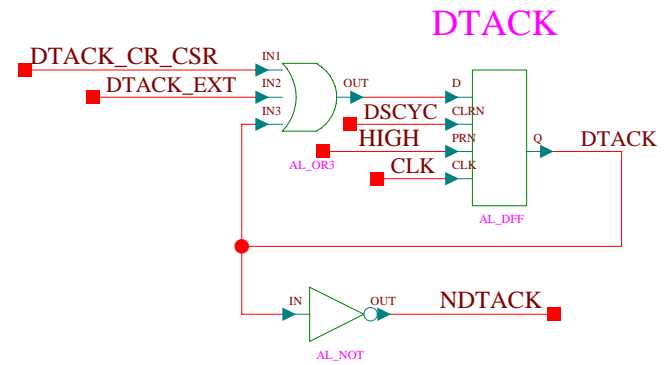
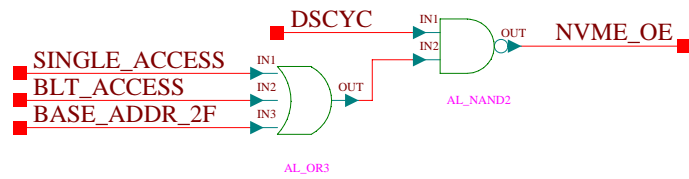
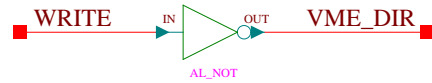
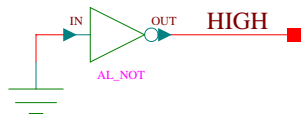
sheet **5** of **5**

modified by: HB

8-26-2005 13:54

checked by: CHECKER

0-00-0000 00:00



VME64X-CHIP	
DTACK_BERR	
Version: V1.1	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	3-13-2007_14:47
checked by: CHECKER	0-00-0000_00:00

NGA[4:0]

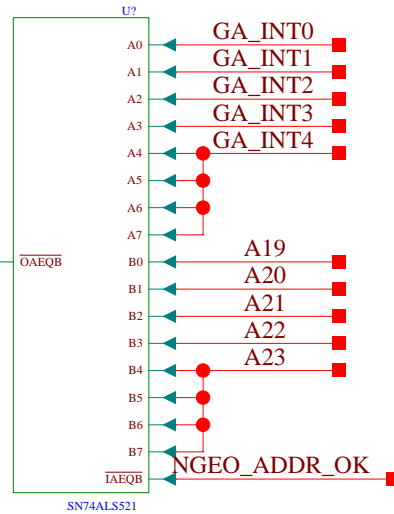
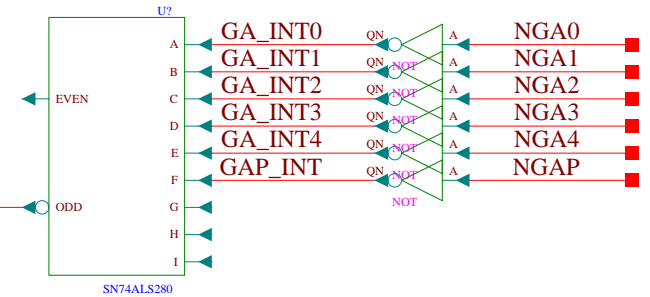
A[23:19]

AM[5:0]

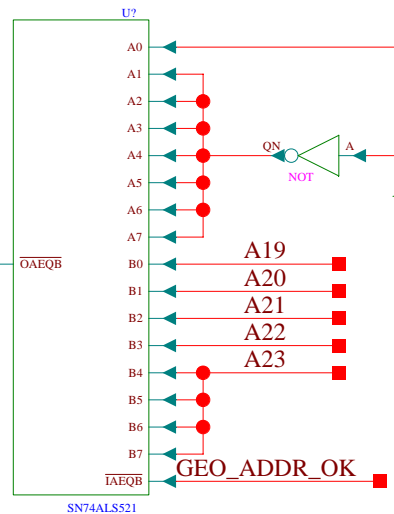
GA_INT[4:0]

NGEO_ADDR_OK

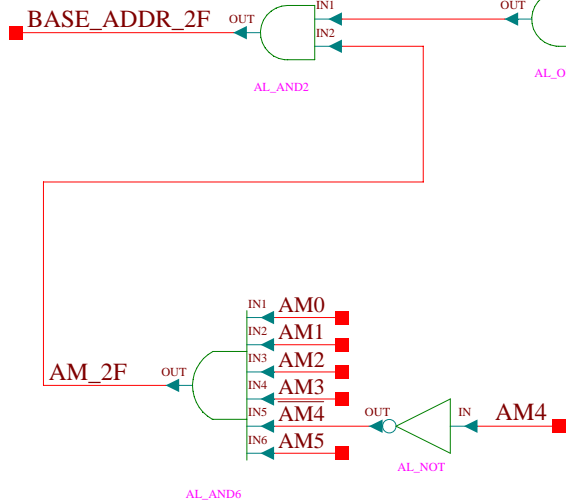
GEO_ADDR_OK



"geographical address"



"amnesia address = 0x1E"



VME64X-CHIP

GEO_ADDR

Version: **V2.1**

HEPHY VIENNA
ELEKTRONIK 1

sheet **1** of **1**

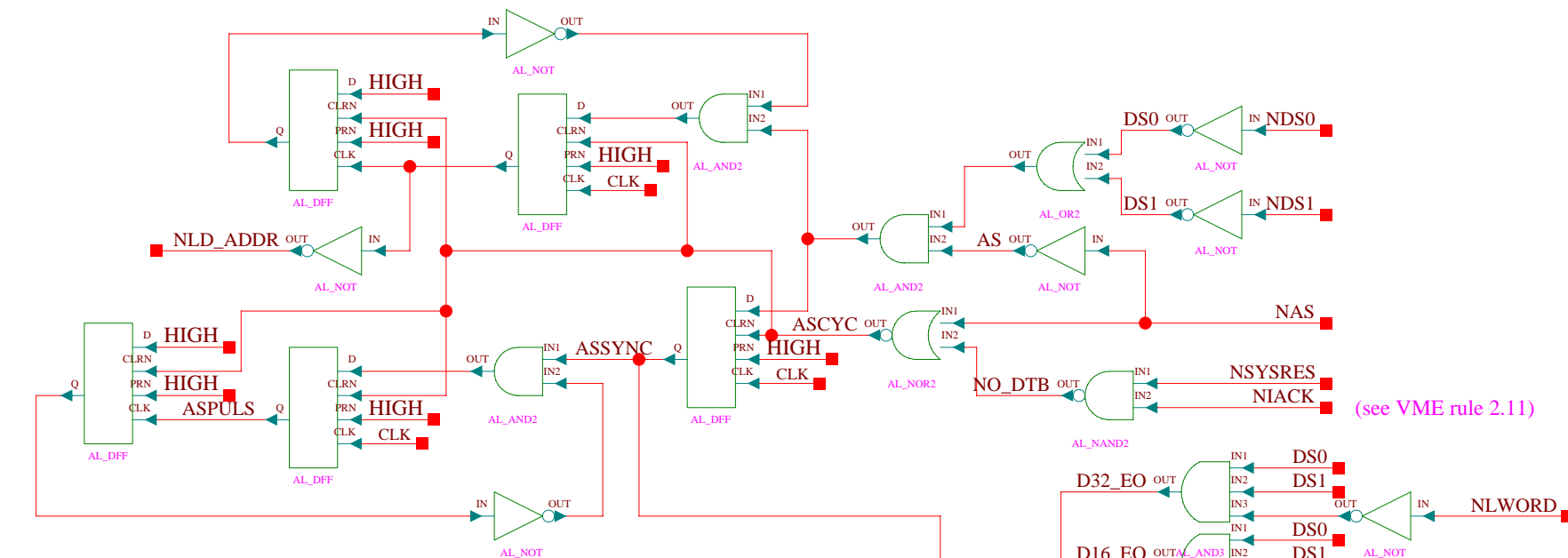
modified by **HB**

3-5-2007_13:00

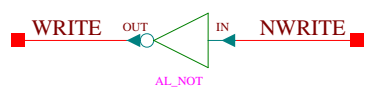
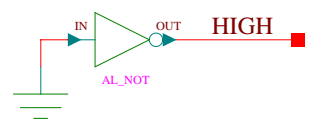
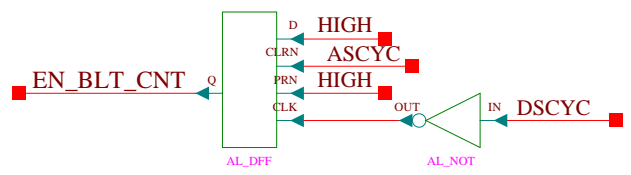
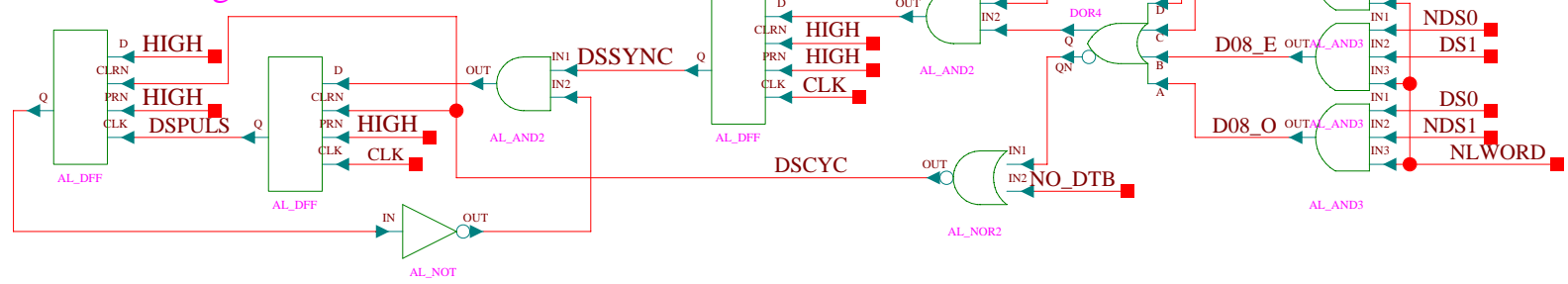
checked by: **CHECKER**

0-00-0000_00:00

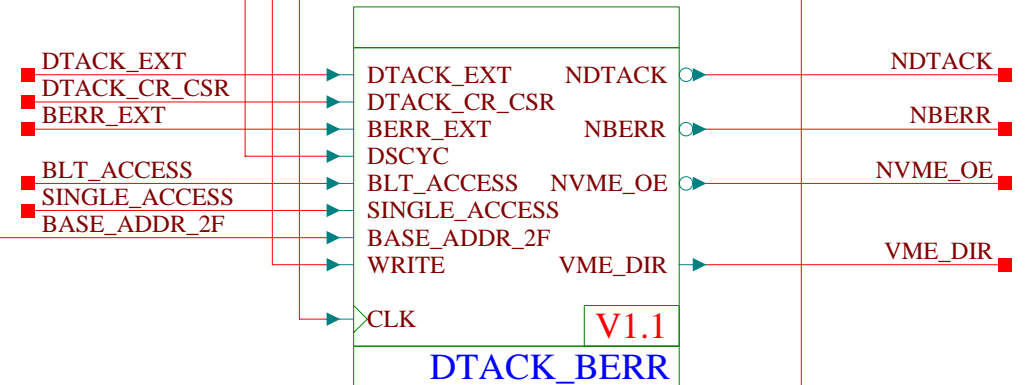
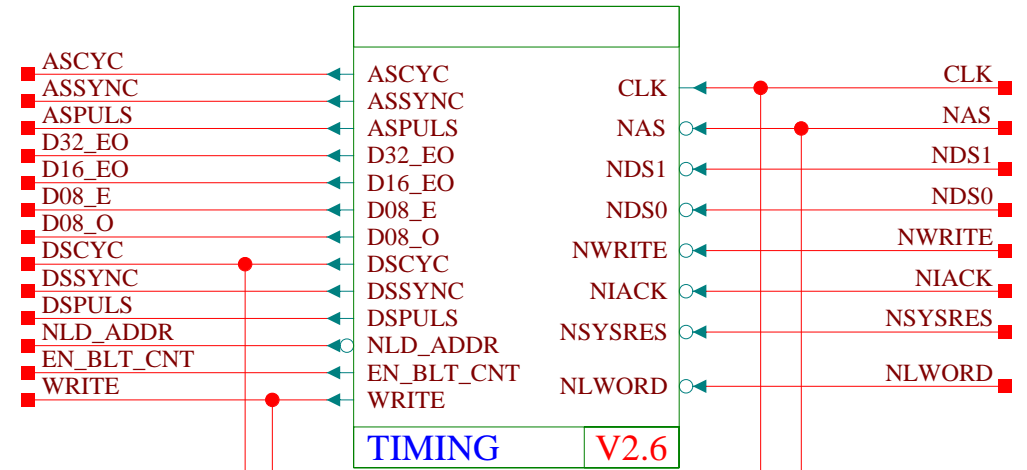
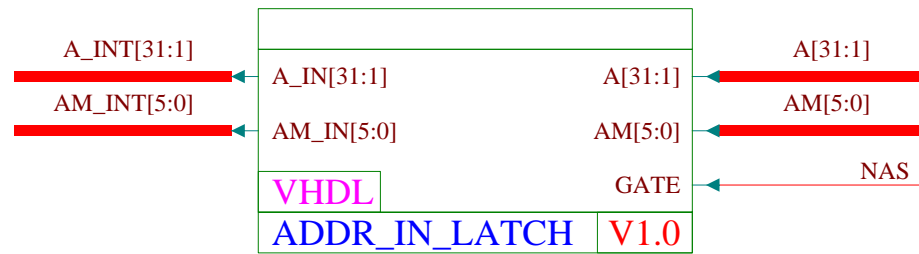
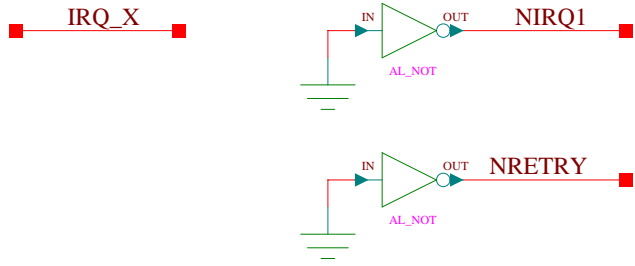
AS Timing



DS Timing



VME64X-CHIP	
TIMING	
Version: V2.6	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by HB	3-13-2007_14:41
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP	
VME_IO	
Version:	V2.1
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	3-28-2007_14:43
checked by: CHECKER	0-00-0000_00:00

```
-----  
--  
-- LOGIC CORE: GTL-module vme64x interface chip logic  
-- MODULE NAME: cr  
-- INSTITUTION: Hephy Vienna  
-- DESIGNER: H. Bergauer  
--  
-- VERSION: V1.0  
-- DATE: 08 2005  
--  
-- FUNCTIONAL DESCRIPTION:  
-- configuration ROM for VME64x  
-- range: 0x03 - 0x7FF  
--  
-----  
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
LIBRARY lpm;  
USE lpm.lpm_components.ALL;  
  
ENTITY cr IS  
    PORT(  
        addr      : IN      STD_LOGIC_VECTOR(10 DOWNTO 2);  
        rd_en     : IN      STD_LOGIC;  
        data      : INOUT   STD_LOGIC_VECTOR(7 DOWNTO 0));  
END cr;  
  
ARCHITECTURE rtl OF cr IS  
BEGIN  
  
    -- Configuration ROM für VME64x mit 512x8 bits  
    cr_rom:lpm rom  
    GENERIC MAP (LPM_WIDTH => 8,  
        LPM_WIDTHAD => 9,  
        LPM_OUTDATA => "UNREGISTERED",  
        LPM_ADDRESS_CONTROL => "UNREGISTERED",  
        LPM_FILE => "cr.mif")  
    PORT MAP (address => addr,   
        memenab => rd_en,  
        q => data);  
  
END ARCHITECTURE rtl;
```

```

-- *****
-- Specify values for addresses of Configuration ROM for VME64x of TCS9U-cards
-- Only every forth address of CR table is used. D08_0 = 1 and A01 = 1.

-- V1012:

-- CR/CSR space definition:

-- VME64x CR range: 0x03-0xFFFF (0x03-0x7FF in this mif-file defined)
-- USER_CR range for chip id, version and SN: 0x001003-0x00103F (see user cr.mif)
-- CRAM_range for future applications (not defined yet): 0x003003-0x0037FF
-- [USER CSR range for TEST OUT-selection register: 0x005003-0x00502F - not used] only fixed TEST_OUTs (HB120407)
-- VME64x_CSR range: 0x7FC00-0x7FFFF

-- FUNCTION definition:

-- Function 0: D16 only, A31-A25, AM=0x0D and 0x09 (single transfer)
-- *****

DEPTH = 512;      % Memory depth and width are required      %
WIDTH = 8;        % Enter a decimal number                    %

ADDRESS_RADIX = HEX;      % Address and value radixes are required      %
DATA_RADIX = HEX;        % Enter BIN, DEC, HEX, OCT, or UNS; unless      %
                        % otherwise specified, radixes = HEX          %

-- used address-bits
-- A10 | A09 A08 A07 A06 | A05 A04 A03 A02

CONTENT
BEGIN
000      : 00; -- checksum [CR address = 0x03] -- to be defined!!
001      : 00; -- length of ROM (MSB, byte 2), not defined [CR address = 0x07] -- to be defined!!
002      : 00; -- length of ROM (byte 1), not defined [CR address = 0x0B]
003      : 00; -- length of ROM (LSB, byte 0), not defined [CR address = 0x0F]

004      : 81; -- CR data access width (0x81=D08(O)) [CR address = 0x13]
005      : 81; -- CSR data access width (0x81=D08(O)) [CR address = 0x17]
006      : 02; -- CR/CSR space specification ID (0x02=VME64x) [CR address = 0x13]
007      : 43; -- 0x43 (ASCII "C") [CR address = 0x1F]

008      : 52; -- 0x52 (ASCII "R") [CR address = 0x23]
009      : 00; -- Manufactor's ID (MSB, byte 2) (0x00=no IEEE code) [CR address = 0x27]
00A      : 00; -- Manufactor's ID (byte 1) (0x00=no IEEE code) [CR address = 0x2B]
00B      : 00; -- Manufactor's ID (LSB, byte 0) (0x00=no IEEE code) [CR address = 0x2F]

00C      : A0; -- Board ID (MSB, byte 3) (0xA0=example) [CR address = 0x33]
00D      : 12; -- Board ID (byte 2) (0x12=example) [CR address = 0x37]

```

```

00E      : 34; -- Board ID (byte 1) (0x34=example) [CR address = 0x3B]
00F      : 56; -- Board ID (LSB, byte 0) (0x56=example) [CR address = 0x3F]

010      : B9; -- Revision ID (MSB, byte 3) (0xB9=example) [CR address = 0x43]
011      : 87; -- Revision ID (byte 2) (0x87=example) [CR address = 0x47]
012      : 65; -- Revision ID (byte 1) (0x65=example) [CR address = 0x4B]
013      : 43; -- Revision ID (LSB, byte 0) (0x43=example) [CR address = 0x4F]

[014..01E] : 00; -- not used! ("Pointer to null ..." and "RESERVED") [CR address = 0x53..0x7B]

01F      : 01; -- Programm ID (0x01=no program, ID code only) [CR address = 0x7F]

020      : 00; -- Offset to BEG_USER_CR (MSB, byte 2), (0x00) [CR address = 0x83] -- BEG_USER_CR = 0x001003
021      : 10; -- Offset to BEG_USER_CR (byte 1), (0x10) [CR address = 0x87]
022      : 03; -- Offset to BEG_USER_CR (LSB, byte 0), (0x03) [CR address = 0x8B]
023      : 00; -- Offset to END_USER_CR (MSB, byte 2), (0x00) [CR address = 0x8F] -- END_USER_CR = 0x00101F

024      : 10; -- Offset to END_USER_CR (byte 1), (0x10) [CR address = 0x93]
025      : 1F; -- Offset to END_USER_CR (LSB, byte 0), (0x1F) [CR address = 0x97]
026      : 00; -- Offset to BEG_CRAM (MSB, byte 2), (0x00) [CR address = 0x9B] -- BEG_CRAM = 0x003003
027      : 30; -- Offset to BEG_CRAM (byte 1), (0x30) [CR address = 0x9F]

028      : 03; -- Offset to BEG_CRAM (LSB, byte 0), (0x03) [CR address = 0xA3]
029      : 00; -- Offset to END_CRAM (MSB, byte 2), (0x00) [CR address = 0xA7] -- END_CRAM = 0x0037FF
02A      : 37; -- Offset to END_CRAM (byte 1), (0x37) [CR address = 0xAB]
02B      : FF; -- Offset to END_CRAM (LSB, byte 0), (0xFF) [CR address = 0xAF]

02C      : 00; -- Offset to BEG_USER_CSR (MSB, byte 2), (0x00) [CR address = 0xB3] -- BEG_USER_CSR = 0x005003
02D      : 50; -- Offset to BEG_USER_CSR (byte 1), (0x50) [CR address = 0xB7]
02E      : 03; -- Offset to BEG_USER_CSR (LSB, byte 0), (0x03) [CR address = 0xBB]
02F      : 00; -- Offset to END_USER_CSR (MSB, byte 2), (0x00) [CR address = 0xBF] -- END_USER_CSR = 0x00502F

030      : 50; -- Offset to END_USER_CSR (byte 1), (0x50) [CR address = 0xC3]
031      : 2F; -- Offset to END_USER_CSR (LSB, byte 0), (0x2F) [CR address = 0xC7]
032      : 00; -- Offset to BEG_SN (MSB, byte 2), (0x00) [CR address = 0xCB] -- BEG_SN = 0x001023
033      : 10; -- Offset to BEG_SN (byte 1), (0x10) [CR address = 0xCF]

034      : 23; -- Offset to BEG_SN (LSB, byte 0), (0x23) [CR address = 0xD3]
035      : 00; -- Offset to END_SN (MSB, byte 2), (0x00) [CR address = 0xD7] -- END_SN = 0x00103F
036      : 10; -- Offset to END_SN (byte 1), (0x10) [CR address = 0xDB]
037      : 3F; -- Offset to END_SN (LSB, byte 0), (0x2F) [CR address = 0xDF]

038      : 00; -- Slave characteristics parameter (0x00, see Table 10-1 VME64x spec.) [CR address = 0xE3]
039      : 00; -- User defined slave characteristics, not used! [CR address = 0xE7]
03A      : 00; -- Master characteristics, not used! [CR address = 0xEB]
03B      : 00; -- User defined master characteristics, not used! [CR address = 0xEF]

03C      : 00; -- Interrupt handler capabilities, not used! [CR address = 0xF3]

```

```

03D      : 00; -- Interrupter capabilities, not used! [CR address = 0xF7]
03E      : 00; -- Reserved, not used! [CR address = 0xFB]
03F      : 00; -- CRAM_ACCESS_WIDTH (0x81=D08(O)) [CR address = 0xFF]

040      : 83; -- Function 0 DAWPR (0x83, "D16 only (C. Schwick!)", see Table 10-3 VME64x spec.) [CR address
= 0x103]
041      : 00; -- Function 1 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x107]
042      : 00; -- Function 2 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x10B]
043      : 00; -- Function 3 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x10F]

044      : 00; -- Function 4 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x113]
045      : 00; -- Function 5 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x117]
046      : 00; -- Function 6 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x11B]
047      : 00; -- Function 7 DAWPR (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR address
= 0x11F]

048      : 00; -- Function 0 AMCAP (MSB, byte 7) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x123]
049      : 00; -- Function 0 AMCAP (byte 6) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x127]
04A      : 00; -- Function 0 AMCAP (byte 5) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x12B]
04B      : 00; -- Function 0 AMCAP (byte 4) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x12F]

04C      : 00; -- Function 0 AMCAP (byte 3) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x133]
04D      : 00; -- Function 0 AMCAP (byte 2) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x137]
04E      : 22; -- Function 0 AMCAP (byte 1) (0x22, AM=0x0D and 0x09, see 10.2.1.4.2 VME64x spec.) [CR address
= 0x13B]
04F      : 00; -- Function 0 AMCAP (LSB, byte 0) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x13F]

[050..187] : 00; -- not used! ("Function 1 - 7 AMCAP" and "Function 0 - 7 XAMCAP") [CR address = 0x163..0x61F]

188      : FE; -- Function 0 ADEM (MSB, byte 3) (0xFE, "mask bits 31-25=1, 24=0", see Table 10-4 VME64x
spec.) [CR address = 0x623]
189      : 00; -- Function 0 ADEM (byte 2) (0x00, "mask bits 23-16=0", see Table 10-4 VME64x spec.) [CR
address = 0x627]
18A      : 00; -- Function 0 ADEM (byte 1) (0x00, "mask bits 15-8=0", see Table 10-4 VME64x spec.) [CR
address = 0x62B]
18B      : 00; -- Function 0 ADEM (LSB, byte 0) (0x00, "special bits=0", see Table 10-4 VME64x spec.) [CR
address = 0x62F]

[18C..1AA] : 00; -- not used! ("Function 1 - 7 ADEM" and "reserved, read as zero") [CR address = 0x643..0x6AB]

1AB      : 00; -- Master data access width (0x00, "feature not implemented", see Table 10-3 VME64x spec.) [CR
address = 0x6AF]

```

cr.mif

25.04.2007

```
[1AC..1D3]      : 00; -- not used! ("Master AMCAP" and "Master XAMCAP") [CR address = 0x6B3..0x74F]  
[1D4..1FF]      : 00; -- not used! ("RESERVED") [CR address = 0x753..0x7FF]
```

END;

```
-----□
--
-- LOGIC CORE: GTL-module vme64x interface chip logic  --□
-- MODULE NAME: cram  --□
-- INSTITUTION: Hephy Vienna  --□
-- DESIGNER: H. Bergauer  --□
--  --□
-- VERSION: V1.0  --□
-- DATE: 08 2005  --□
--  --□
-- FUNCTIONAL DESCRIPTION:  --□
-- configuration RAM 512x8 for VME64x  --□
-- range: 0x03003 - 0x037FF  --□
--  --□
-----□

LIBRARY ieee;□
USE ieee.std_logic_1164.ALL;□
LIBRARY lpm;□
USE lpm.lpm_components.ALL;□
□
ENTITY cram IS□
    PORT(□
        addr      : IN      STD_LOGIC_VECTOR(10 DOWNTO 2);□
        clk       : IN      STD_LOGIC;□
        ld_en     : IN      STD_LOGIC;□
        rd_en     : IN      STD_LOGIC;□
        data      : INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0));□
END cram;□
□
ARCHITECTURE rtl OF cram IS□
BEGIN□
□
-- Configuration RAM für VME64x mit 512x8 bits  □
    config_ram: lpm_ram_io□
GENERIC MAP (LPM_WIDTH => 8,□
    LPM_WIDTHAD => 9,□
        LPM_INDATA => "REGISTERED",□
        LPM_OUTDATA => "UNREGISTERED",□
        LPM_ADDRESS_CONTROL => "REGISTERED")□
PORT MAP (address => addr, □
    inclock => clk,□
    we => ld_en,□
    outenab => rd_en,□
    dio => data);□
□
END ARCHITECTURE rtl;□
```

```

-----
--
-- LOGIC CORE: GTL-module vme64x interface chip logic
-- MODULE NAME: csr_ext_ext
-- INSTITUTION: Hephy Vienna
-- DESIGNER: H. Bergauer
--
-- VERSION: V1.0
-- DATE: 08 2005
--
-- FUNCTIONAL DESCRIPTION:
-- control/status register
-- range: 0x7FC00 - 0x7FFFF
-- F0 => extended access A31-A25
-- F1 => extended access A31-A25
--
-----

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

```

```

ENTITY csr_ext_ext IS
    PORT(
        clk : IN STD_LOGIC;
        d : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        ga : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        nsysres : IN STD_LOGIC;
        nberr : IN STD_LOGIC;
        geo_addr_ok : IN STD_LOGIC;
        ld_ader0 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        ld_ader1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        rd_ader0 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        rd_ader1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        ld_bar : IN STD_LOGIC;
        rd_bar : IN STD_LOGIC;
        ld_bsr : IN STD_LOGIC;
        ld_bcr : IN STD_LOGIC;
        rd_bscr : IN STD_LOGIC;
        reset_mode : INOUT STD_LOGIC;
        mod_enabled : INOUT STD_LOGIC;
        ader0_a : OUT STD_LOGIC_VECTOR(31 DOWNTO 25);
        ader1_a : OUT STD_LOGIC_VECTOR(31 DOWNTO 25);
        ader0_am : OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
        ader1_am : OUT STD_LOGIC_VECTOR(5 DOWNTO 0));
END csr_ext_ext;

```

```

ARCHITECTURE rtl OF csr_ext_ext IS
    CONSTANT amnesia_addr: STD_LOGIC_VECTOR(4 DOWNTO 0) := "11110";
    SIGNAL aclr: STD_LOGIC;
    SIGNAL ader0_3_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader0_2_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader0_1_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader0_0_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader1_3_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader1_2_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader1_1_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ader1_0_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL bsr_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL bcr_out: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL bscr_in: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL bar_in: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL set_res_mode: STD_LOGIC;
    SIGNAL dis_res_mode: STD_LOGIC;
    SIGNAL en_module: STD_LOGIC;
    SIGNAL dis_module: STD_LOGIC;
    SIGNAL set_berr_flag: STD_LOGIC;

```

```

        SIGNAL clr_berr_flag: STD_LOGIC;
        SIGNAL berr_flag: STD_LOGIC;
BEGIN
    aclr <= NOT nsysres;
    --
    -- base-address A31-A25
    ader0_a <= ader0_3_out(7 DOWNT0 1);
    ader0_am <= ader0_0_out(7 DOWNT0 2);
    --
    ader1_a <= ader1_3_out(7 DOWNT0 1);
    ader1_am <= ader1_0_out(7 DOWNT0 2);
    --
    -- *****
    -- load ader0_3 register
    ader0_3_load: lpm_ff
    GENERIC MAP (LPM_WIDTH => 8)
    PORT MAP (data => d,
             clock => clk,
             enable => ld_ader0(3),
             aclr => aclr,
             q => ader0_3_out);
    --
    -- load ader0_2 register
    ader0_2_load: lpm_ff
    GENERIC MAP (LPM_WIDTH => 8)
    PORT MAP (data => d,
             clock => clk,
             enable => ld_ader0(2),
             aclr => aclr,
             q => ader0_2_out);
    --
    -- load ader0_1 register
    ader0_1_load: lpm_ff
    GENERIC MAP (LPM_WIDTH => 8)
    PORT MAP (data => d,
             clock => clk,
             enable => ld_ader0(1),
             aclr => aclr,
             q => ader0_1_out);
    --
    -- load ader0_0 register
    ader0_0_load: lpm_ff
    GENERIC MAP (LPM_WIDTH => 8)
    PORT MAP (data => d,
             clock => clk,
             enable => ld_ader0(0),
             aclr => aclr,
             q => ader0_0_out);
    --
    -- read ader0_3 register
    ader0_3_read:
    FOR i IN 0 TO 7 GENERATE
        tri_ader0_3: tri
        PORT MAP(ader0_3_out(i),
                rd_ader0(3),
                d(i));
    END GENERATE ader0_3_read;
    --
    -- read ader0_2 register
    ader0_2_read:
    FOR i IN 0 TO 7 GENERATE
        tri_ader0_2: tri
        PORT MAP(ader0_2_out(i),
                rd_ader0(2),
                d(i));
    END GENERATE ader0_2_read;
    --
    -- read ader0_1 register
    ader0_1_read:

```

```

FOR i IN 0 TO 7 GENERATE
  tri_ader0_1: tri
  PORT MAP(ader0_1_out(i),
    rd_ader0(1),
    d(i));
END GENERATE ader0_1_read;
-- read ader0 0 register
ader0_0_read:
FOR i IN 0 TO 7 GENERATE
  tri_ader0_0: tri
  PORT MAP(ader0_0_out(i),
    rd_ader0(0),
    d(i));
END GENERATE ader0_0_read;
-- *****
-- load ader1_3 register
ader1_3_load: lpm_ff
GENERIC MAP (LPM_WIDTH => 8)
PORT MAP (data => d,
  clock => clk,
  enable => ld_ader1(3),
  aclr => aclr,
  q => ader1_3_out);
-- load ader1_2 register
ader1_2_load: lpm_ff
GENERIC MAP (LPM_WIDTH => 8)
PORT MAP (data => d,
  clock => clk,
  enable => ld_ader1(2),
  aclr => aclr,
  q => ader1_2_out);
-- load ader1_1 register
ader1_1_load: lpm_ff
GENERIC MAP (LPM_WIDTH => 8)
PORT MAP (data => d,
  clock => clk,
  enable => ld_ader1(1),
  aclr => aclr,
  q => ader1_1_out);
-- load ader1_0 register
ader1_0_load: lpm_ff
GENERIC MAP (LPM_WIDTH => 8)
PORT MAP (data => d,
  clock => clk,
  enable => ld_ader1(0),
  aclr => aclr,
  q => ader1_0_out);
-- read ader1_3 register
ader1_3_read:
FOR i IN 0 TO 7 GENERATE
  tri_ader1_3: tri
  PORT MAP(ader1_3_out(i),
    rd_ader1(3),
    d(i));
END GENERATE ader1_3_read;
-- read ader1_2 register
ader1_2_read:
FOR i IN 0 TO 7 GENERATE
  tri_ader1_2: tri
  PORT MAP(ader1_2_out(i),
    rd_ader1(2),
    d(i));

```

```

END GENERATE ader1_2_read;
-- read ader1_1 register
  ader1_1_read:
  FOR i IN 0 TO 7 GENERATE
    tri_ader1_1: tri
    PORT MAP(ader1_1_out(i),
      rd ader1(1),
      d(i));
  END GENERATE ader1_1_read;
-- read ader1_0 register
  ader1_0_read:
  FOR i IN 0 TO 7 GENERATE
    tri_ader1_0: tri
    PORT MAP(ader1_0_out(i),
      rd ader1(0),
      d(i));
  END GENERATE ader1_0_read;
-- *****
-- load bit set register
  bsr_load: lpm_ff
  GENERIC MAP (LPM_WIDTH => 8)
  PORT MAP (data => d,
    clock => clk,
    enable => ld_bsr,
    aclr => aclr,
    q => bsr_out);
  set_res_mode <= bsr_out(7);
  en_module <= bsr_out(4);
  set_berr_flag <= bsr_out(3);
-- load bit clear register
  bcr_load: lpm_ff
  GENERIC MAP (LPM_WIDTH => 8)
  PORT MAP (data => d,
    clock => clk,
    enable => ld_bcr,
    aclr => aclr,
    q => bcr_out);
  dis_res_mode <= bcr_out(7);
  dis_module <= bcr_out(4);
  clr_berr_flag <= bcr_out(3);
-- *****
-- setting and clearing bits of BSR and BCR
PROCESS (set_res_mode, dis_res_mode, en_module, dis_module, set_berr_flag, clr_berr_flag,
BEGIN
  IF set_res_mode='1' THEN
    reset_mode <= '1';
  ELSIF (set_res_mode='0' AND dis_res_mode='1') OR nsysres='0' THEN
    reset_mode <= '0';
  END IF;
  IF en_module='1' OR nsysres='0' THEN
    mod_enabled <= '1';
  ELSIF (en_module='0' AND dis_module='1' AND nsysres='1') THEN
    mod_enabled <= '0';
  END IF;
  IF en_module='1' THEN
    mod_enabled <= '1';
  ELSIF (en_module='0' AND dis_module='1') OR nsysres='0' THEN
    mod_enabled <= '0';
  END IF;
-- set berr_flag if a BERR is generated on board or set_berr_flag='1'??

```

```

-- clear berr_flag if a SYSRES is generated or clr_berr_flag='1' (and set_berr_flag='0')
-- see VME64x-specification Rule 10.16
[]
    IF set_berr_flag='1' OR nberr='0' THEN[]
        berr_flag <= '1';[]
    ELSIF (set_berr_flag='0' AND clr_berr_flag='1') OR nsysres='0' THEN[]
        berr_flag <= '0';[]
    END IF;[]
END PROCESS;[]
[]
-- *****
bscr_in <= reset_mode & '0' & '0' & mod_enabled & berr_flag & '0' & '0' & '0'; []
[]
-- read bit set/clear register
bscr_read:[]
FOR i IN 0 TO 7 GENERATE[]
    tri_bscr: tri[]
    PORT MAP(bscr_in(i),[]
        rd_bscr,[]
        d(i));[]
END GENERATE bscr_read;[]
[]
-- *****
-- setting BAR with GA or amnesia address
[]
PROCESS (geo_addr_ok, ga)[]
BEGIN[]
    IF geo_addr_ok = '1' THEN[]
        bar_in(7 DOWNT0 3) <= ga(4 DOWNT0 0); []
        bar_in(2 DOWNT0 0) <= "000"; []
    ELSE[]
        bar_in <= amnesia_addr & '0' & '0' & '0'; []
    END IF;[]
END PROCESS;[]
[]
-- read base address register
bar_read:[]
FOR i IN 0 TO 7 GENERATE[]
    tri_bar: tri[]
    PORT MAP(bar_in(i),[]
        rd_bar,[]
        d(i));[]
END GENERATE bar_read;[]
[]
END ARCHITECTURE rtl;[]

```

```

-----
--
-- LOGIC CORE: GTL-module vme64x interface chip logic
-- MODULE NAME: user_cr
-- INSTITUTION: Hephy Vienna
-- DESIGNER: H. Bergauer
--
--
-- VERSION: V2.1
-- DATE: 08 2005
--
-- FUNCTIONAL DESCRIPTION:
-- ROM for chip id and version (each 4 bytes)
-- and serial number (VME64x)
-- range: 0x01003 - 0x0103F
--
-- REVISION:
-- V2.1: CARD_NR from S24-S27 jumpers
-----

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm components.ALL;
LIBRARY altera;
USE altera.maxplus2.ALL;

ENTITY user_cr IS
    PORT(
        addr      : IN      STD_LOGIC_VECTOR(5 DOWNTO 2);
        card_nr   : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);
        rd_en     : IN      STD_LOGIC;
        data      : INOUT   STD_LOGIC_VECTOR(7 DOWNTO 0));
END user_cr;

ARCHITECTURE rtl OF user_cr IS
    SIGNAL addr_card_nr : std_logic;
    SIGNAL data_mem : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_int : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_tri : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

    addr_card_nr <= NOT addr(5) AND NOT addr(4) AND addr(3) AND NOT addr(2) AND rd_en;

    -- USER CR for chip id, version and SN with 16x8 bits
    call user_cr: lpm rom
        GENERIC MAP (LPM_WIDTH => 8,
            LPM_WIDTHHAD => 4,
            LPM_OUTDATA => "UNREGISTERED",
            LPM_ADDRESS_CONTROL => "UNREGISTERED",
            LPM_FILE => "user_cr.mif")
        PORT MAP (address => addr,
            memenab => rd_en,
            q => data_mem);

    -- card nr
    data_int(7) <= data_mem(7);
    data_int(6) <= data_mem(6);
    data_int(5) <= data_mem(5);
    data_int(4) <= data_mem(4);
    data_int(3) <= card_nr(3);
    data_int(2) <= card_nr(2);
    data_int(1) <= card_nr(1);
    data_int(0) <= card_nr(0);

    -- mux for card_nr
    call mux: busmux
        GENERIC MAP (WIDTH => 8)
        PORT MAP (dataa => data_mem,
            datab => data_int,
            sel => addr_card_nr,
            result => data_tri);

```

```
tri_data:
FOR i IN 0 TO 7 GENERATE
  call_data mem: tri
  PORT MAP(data_tri(i),
    rd en,
    data(i));
END GENERATE tri_data;

END ARCHITECTURE rtl;
```

```

-- *****
-- Specify values for addresses of User Configuration ROM for VME64x of TCS-9u-cards
-- Only every forth address is used. D08 0 = 1 and A01 = 1.
-- USER CR range for chip_id, version and SN (VME64x): 0x001003-0x00102B
-- chip id: 0x00015011 => 0 for card_nr, card_nr comes in hardware from S31-S28!!!!
-- see user_cr.vhd V2.1 !!!
-- version: 0x00001012
-- *****

DEPTH = 16;      % Memory depth and width are required      %
WIDTH = 8;       % Enter a decimal number                    %

ADDRESS_RADIX = HEX;      % Address and value radixes are required      %
DATA_RADIX = HEX;        % Enter BIN, DEC, HEX, OCT, or UNS; unless      %
                        % otherwise specified, radixes = HEX      %

-- used address-bits
-- A05 A04 A03 A02

CONTENT
BEGIN
0      : 00; -- chip_id-register 3 (MSB) [USER_CR address = 0x001003]
1      : 01; -- chip id-register 2 [USER CR address = 0x001007]
2      : 50; -- chip id-register 1 [USER CR address = 0x00100B]
3      : 11; -- chip_id-register 0 [USER_CR address = 0x00100F]

4      : 00; -- version-register 3 (MSB) [USER CR address = 0x001013]
5      : 00; -- version-register 2 [USER CR address = 0x001017]
6      : 10; -- version-register 1 [USER_CR address = 0x00101B]
7      : 12; -- version-register 0 [USER_CR address = 0x00101F]

8      : 54; -- serial number byte 3 (MSB), ASCII "T" [SN address = 0x001023]
9      : 43; -- serial number byte 2, ASCII "C" [SN address = 0x001027]
A      : 53; -- serial number byte 1, ASCII "S" [SN address = 0x00102B]

[B..F] : 00; -- not used! [CR address = 0x00102F-0x00103F]

END;
```

```

-----
--
-- LOGIC CORE: GT-module vme64x interface chip logic      --
-- MODULE NAME: addr cnt reg                               --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--
-- VERSION: V1.0                                          --
-- DATE: 03 2007                                          --
--
-- FUNCTIONAL DESCRIPTION:                                --
-- addresses counter for BLT and register for ADDR and AM --
--
-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY addr_cnt_reg IS
    PORT(
        clk_cnt : IN    STD_LOGIC;
        cnt_en  : IN    STD_LOGIC;
        aload   : IN    STD_LOGIC;
        d16_eo  : IN    STD_LOGIC;
        d32_eo  : IN    STD_LOGIC;
        clk_reg : IN    STD_LOGIC;
        a       : IN    STD_LOGIC_VECTOR(24 DOWNTO 1);
        am      : IN    STD_LOGIC_VECTOR(5 DOWNTO 0);
        a_i     : OUT   STD_LOGIC_VECTOR(24 DOWNTO 1);
        am_i    : OUT   STD_LOGIC_VECTOR(5 DOWNTO 0));
END addr_cnt_reg;

ARCHITECTURE rtl OF addr_cnt_reg IS
    SIGNAL a_reg_int : std_logic_vector(24 downto 1);
    SIGNAL a_cnt_d16 : std_logic_vector(10 downto 1);
    SIGNAL a_cnt_d32 : std_logic_vector(11 downto 2);
    SIGNAL blt_sel   : std_logic_vector(2 downto 0);
    SIGNAL d_reg_in  : std_logic_vector(1 downto 0);
    SIGNAL d_reg     : std_logic_vector(1 downto 0);
    SIGNAL naload    : std_logic;
BEGIN
    -- register for freezing d16_eo and d32_eo with aload
    naload <= NOT aload;

    d_reg_in <= d16_eo & d32_eo;

    inst_d_reg: lpm_ff

```

```

    GENERIC MAP (LPM WIDTH => 2)
    PORT MAP (data => d_reg_in,
             clock => naload,
             q => d_reg);

blt_sel <= cnt_en & d_reg;

-- register for address modifier
inst am_reg: lpm ff
    GENERIC MAP (LPM WIDTH => 6)
    PORT MAP (data => am,
             clock => clk_reg,
             q => am_i);

-- register for addresses
inst addr_reg: lpm ff
    GENERIC MAP (LPM WIDTH => 24)
    PORT MAP (data => a,
             clock => clk_reg,
             q => a_reg_int);

inst cnt_d16: lpm_counter
    GENERIC MAP(LPM WIDTH => 10,
               LPM TYPE => "LPM_COUNTER")
    PORT MAP(data => a(10 DOWNTO 1),
             clock => clk_cnt,
             cnt_en => cnt_en,
             aload => aload,
             q => a_cnt_d16);

inst cnt_d32: lpm_counter
    GENERIC MAP(LPM WIDTH => 10,
               LPM TYPE => "LPM_COUNTER")
    PORT MAP(data => a(11 DOWNTO 2),
             clock => clk_cnt,
             cnt_en => cnt_en,
             aload => aload,
             q => a_cnt_d32);

WITH blt_sel SELECT
    a_i <=
        a_reg_int(24 DOWNTO 1)           WHEN "000",
        a_reg_int(24 DOWNTO 11) & a_cnt_d16 WHEN "110",
        a_reg_int(24 DOWNTO 12) & a_cnt_d32 & '0' WHEN "101",
        a_reg_int(24 DOWNTO 1)           WHEN OTHERS;

END ARCHITECTURE rtl;

```

```

-----
--
-- LOGIC CORE: GTL-module vme64x interface chip logic      --
-- MODULE NAME: addr in latch                               --
-- INSTITUTION: Hephy Vienna                               --
-- DESIGNER: H. Bergauer                                   --
--
-- VERSION: V1.0                                           --
-- DATE: 02 2007                                           --
--
-- FUNCTIONAL DESCRIPTION:                                  --
-- input latch for addresses and address modifier          --
--
-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY addr_in_latch IS
    PORT(
        gate    : IN    STD_LOGIC;
        a       : IN    STD_LOGIC_VECTOR(31 DOWNTO 1);
        am      : IN    STD_LOGIC_VECTOR(5 DOWNTO 0);
        a_in    : OUT   STD_LOGIC_VECTOR(31 DOWNTO 1);
        am_in   : OUT   STD_LOGIC_VECTOR(5 DOWNTO 0));
END addr_in_latch;

ARCHITECTURE rtl OF addr_in_latch IS
BEGIN

-- input register for addresses
    addr_latch: lpm_latch
    GENERIC MAP    (LPM_WIDTH => 31)
    PORT MAP      (data => a,
        gate => gate,
        q => a_in);

-- input register for address modifier
    am_latch: lpm_latch
    GENERIC MAP    (LPM_WIDTH => 6)
    PORT MAP      (data => am,
        gate => gate,
        q => am_in);

END ARCHITECTURE rtl;

```