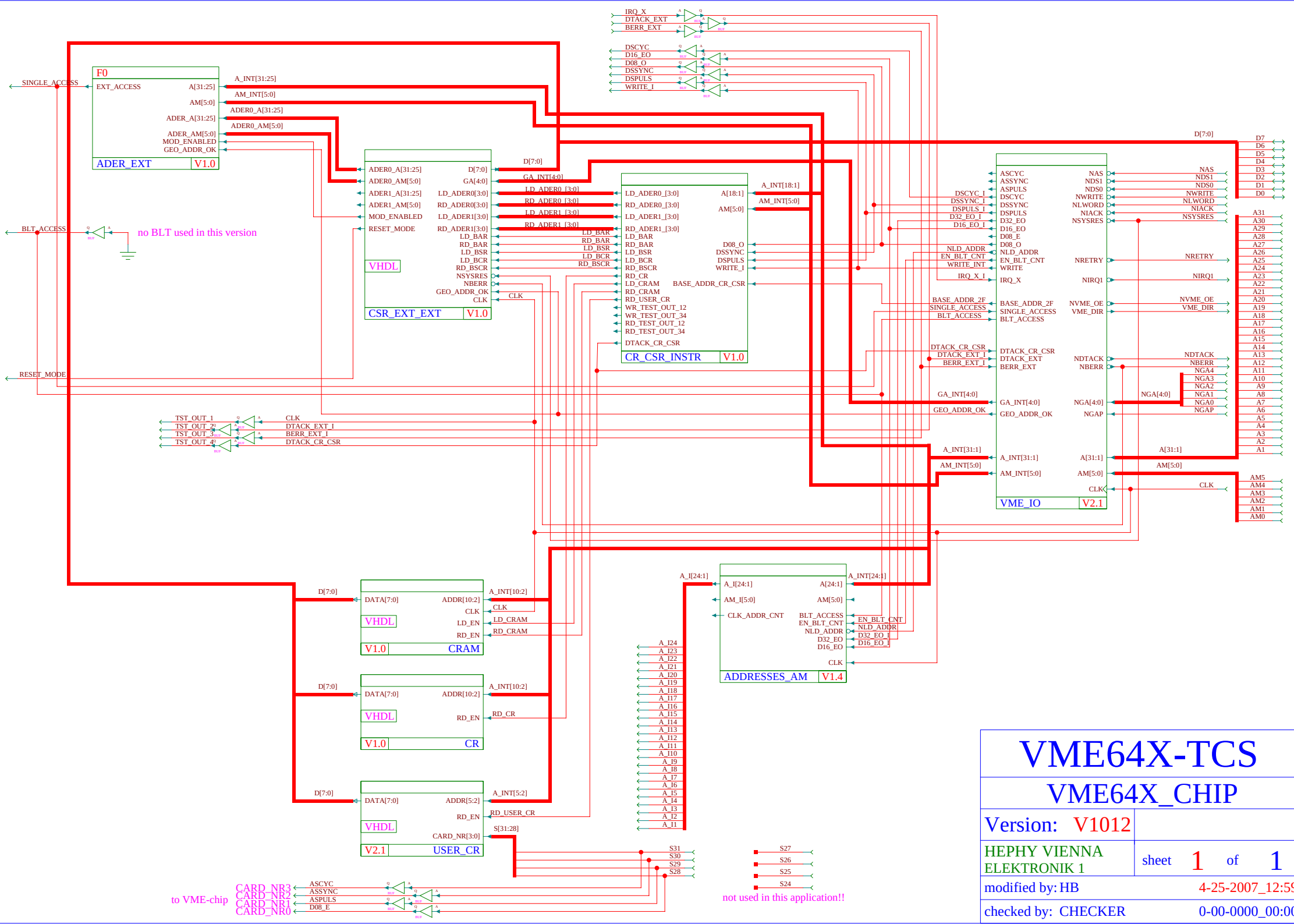
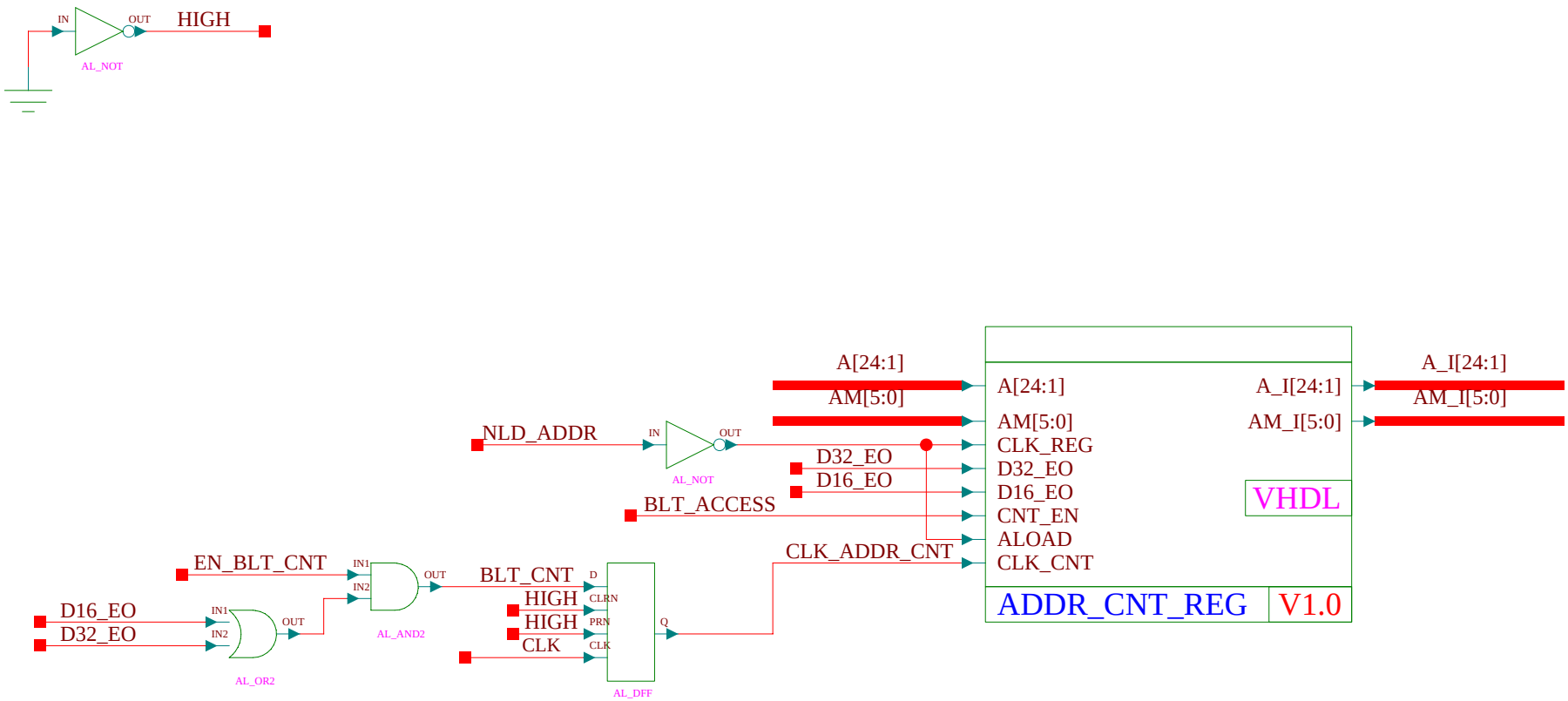

F0
EXT_ACCESS
A[31:25]
AM[5:0]
ADER_A[31:25]
ADER_AM[5:0]
MOD_ENABLED
GEO_ADDR_OK
ADER_EXT
V1.0
SINGLE_ACCESS
A_INT[31:25]
AM_INT[5:0]
ADER0_A[31:25]
ADER0_AM[5:0]
BLT_ACCESS
no BLT used in this version
RESET_MODE
ADER0_A[31:25]
ADER0_AM[5:0]
ADER1_A[31:25]
ADER1_AM[5:0]
MOD_ENABLED
RESET_MODE
D[7:0]
GA[4:0]
LD_ADER0[3:0]
RD_ADER0[3:0]
LD_ADER1[3:0]
RD_ADER1[3:0]
LD_BAR
RD_BAR
LD_BSR
LD_BCR
RD_BSCR
NSYSRES
NBERR
GEO_ADDR_OK
CLK
VHDL
CSR_EXT_EXT
V1.0
GA_INT[4:0]
LD_ADER0_[3:0]
RD_ADER0_[3:0]
LD_ADER1_[3:0]
RD_ADER1_[3:0]
CLK
A[18:1]
AM[5:0]
D08_O
DSSYNC
DSPULS
WRITE_I
BASE_ADDR_CR_CSR
RD_CR
LD_CRAM
RD_CRAM
RD_USER_CR
WR_TEST_OUT_12
WR_TEST_OUT_34
RD_TEST_OUT_12
RD_TEST_OUT_34
DTACK_CR_CSR
CR_CSR_INSTR
V1.0
A_INT[18:1]
AM_INT[5:0]
IRQ_X
DTACK_EXT
BERR_EXT
DSCYC
D16_EO
D08_O
DSSYNC
DSPULS
WRITE_I
TST_OUT_1
TST_OUT_2
TST_OUT_3
TST_OUT_4
CLK
DTACK_EXT_I
BERR_EXT_I
DTACK_CR_CSR
ASCYC
ASSYNC
ASPULS
DSCYC
DSSYNC
DSPULS
D32_EO
D16_EO
D08_E
D08_O
NLD_ADDR
EN_BLT_CNT
WRITE
IRQ_X
BASE_ADDR_2F
SINGLE_ACCESS
BLT_ACCESS
DTACK_CR_CSR
DTACK_EXT
BERR_EXT
GA_INT[4:0]
GEO_ADDR_OK
A_INT[31:1]
AM_INT[5:0]
CLK
DSCYC_I
DSSYNC_I
DSPULS_I
D32_EO_I
D16_EO_I
IRQ_X_I
NAS
NDS1
NDS0
NWRITE
NLWORD
NIACK
NSYSRES
NRETRY
NIRQ1
NVME_OE
VME_DIR
NDTACK
NBERR
NGA4
NGA3
NGA2
NGA1
NGA0
NGAP
NGA[4:0]
A[31:1]
AM[5:0]
CLK
VME_IO
V2.1
D[7:0]
D7
D6
D5
D4
D3
D2
D1
D0
A31
A30
A29
A28
A27
A26
A25
A24
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
AM5
AM4
AM3
AM2
AM1
AM0
DATA[7:0]
ADDR[10:2]
CLK
LD_EN
RD_EN
VHDL
V1.0
CRAM
D[7:0]
A_INT[10:2]
CLK
LD_CRAM
RD_CRAM
ADDR[10:2]
RD_EN
VHDL
V1.0
CR
D[7:0]
A_INT[10:2]
RD_CR
DATA[7:0]
ADDR[5:2]
RD_EN
CARD_NR[3:0]
VHDL
V2.1
USER_CR
D[7:0]
A_INT[5:2]
RD_USER_CR
S[31:28]
A_I[24:1]
AM_I[5:0]
CLK_ADDR_CNT
A[24:1]
AM[5:0]
BLT_ACCESS
EN_BLT_CNT
NLD_ADDR
D32_EO
D16_EO
CLK
ADDRESSES_AM
V1.4
A_INT[24:1]
A_I24
A_I23
A_I22
A_I21
A_I20
A_I19
A_I18
A_I17
A_I16
A_I15
A_I14
A_I13
A_I12
A_I11
A_I10
A_I9
A_I8
A_I7
A_I6
A_I5
A_I4
A_I3
A_I2
A_I1
S31
S30
S29
S28
S27
S26
S25
S24
not used in this application!!
to VME-chip
CARD_NR3
CARD_NR2
CARD_NR1
CARD_NR0
ASCYC
ASSYNC
ASPULS
D08_E
VME64X-TCS
VME64X_CHIP
Version: V1012
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 1
modified by: HB
4-25-2007_12:59
checked by: CHECKER
0-00-0000_00:00



VME64X-CHIP

ADDRESSES_AM

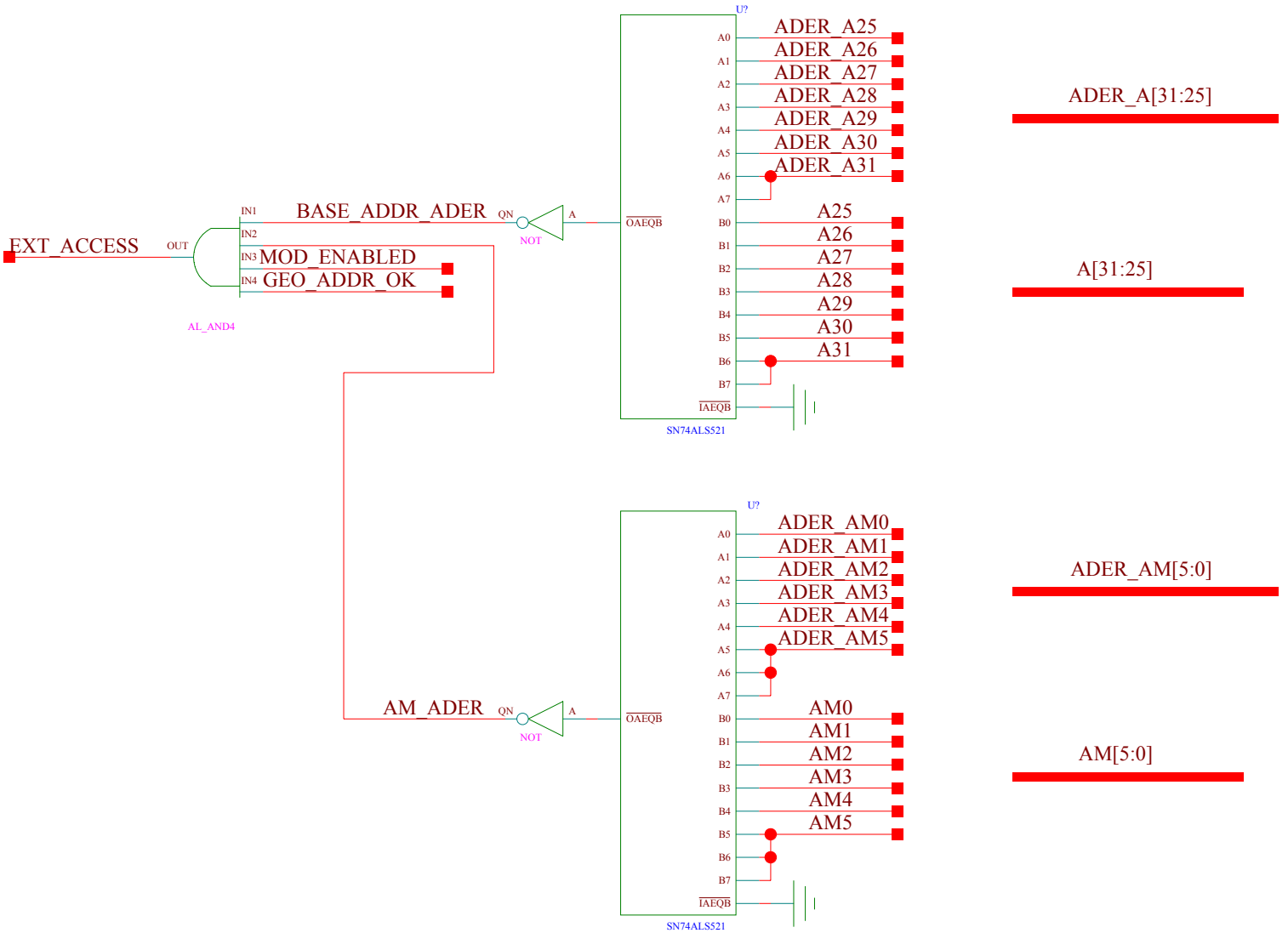
Version: V1.4

HEPHY VIENNA
ELEKTRONIK 1

sheet 1 of 1

modified by: HB 3-9-2007_15:49

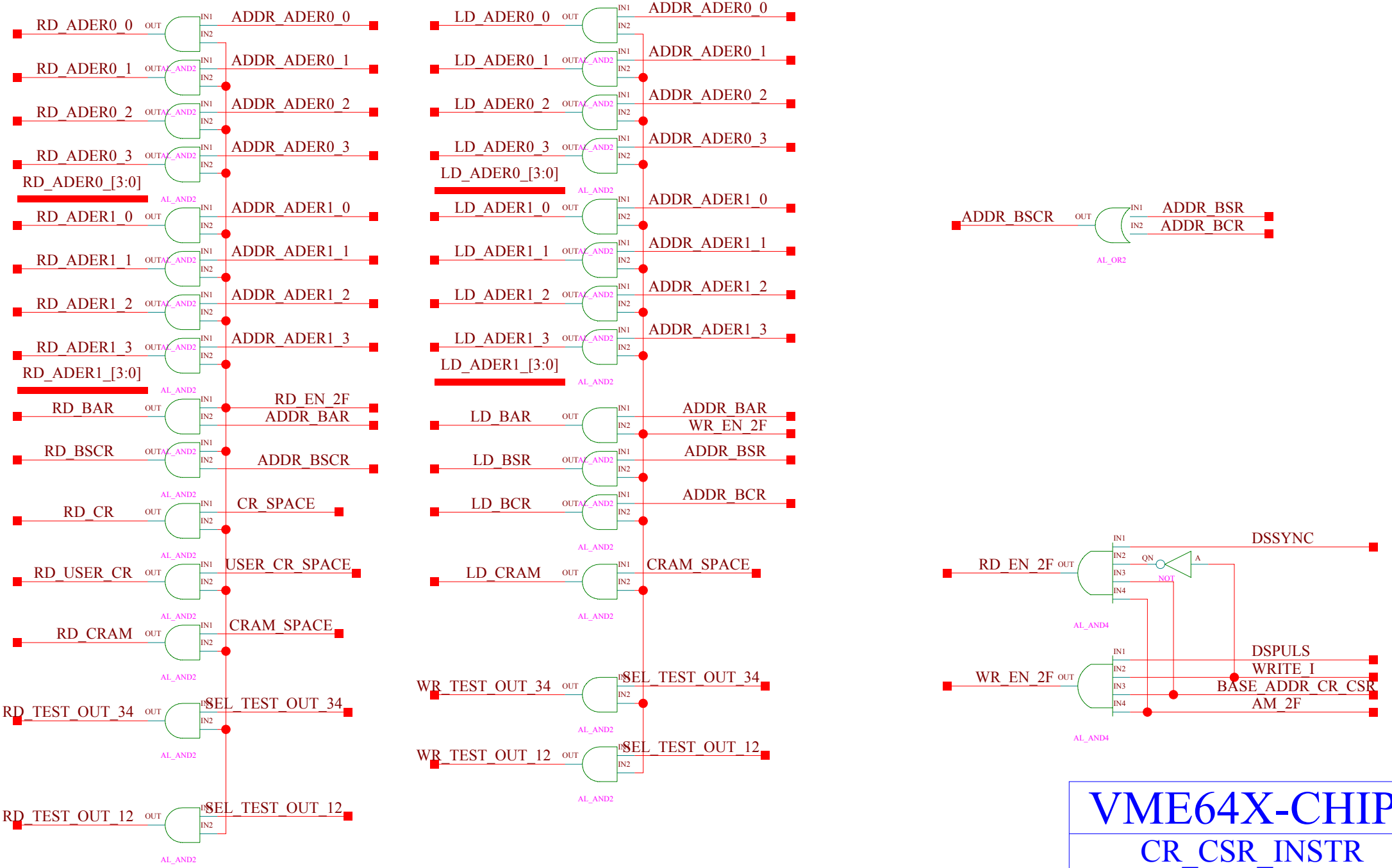
checked by: CHECKER 0-00-0000_00:00



VME64X-CHIP

ADER_EXT

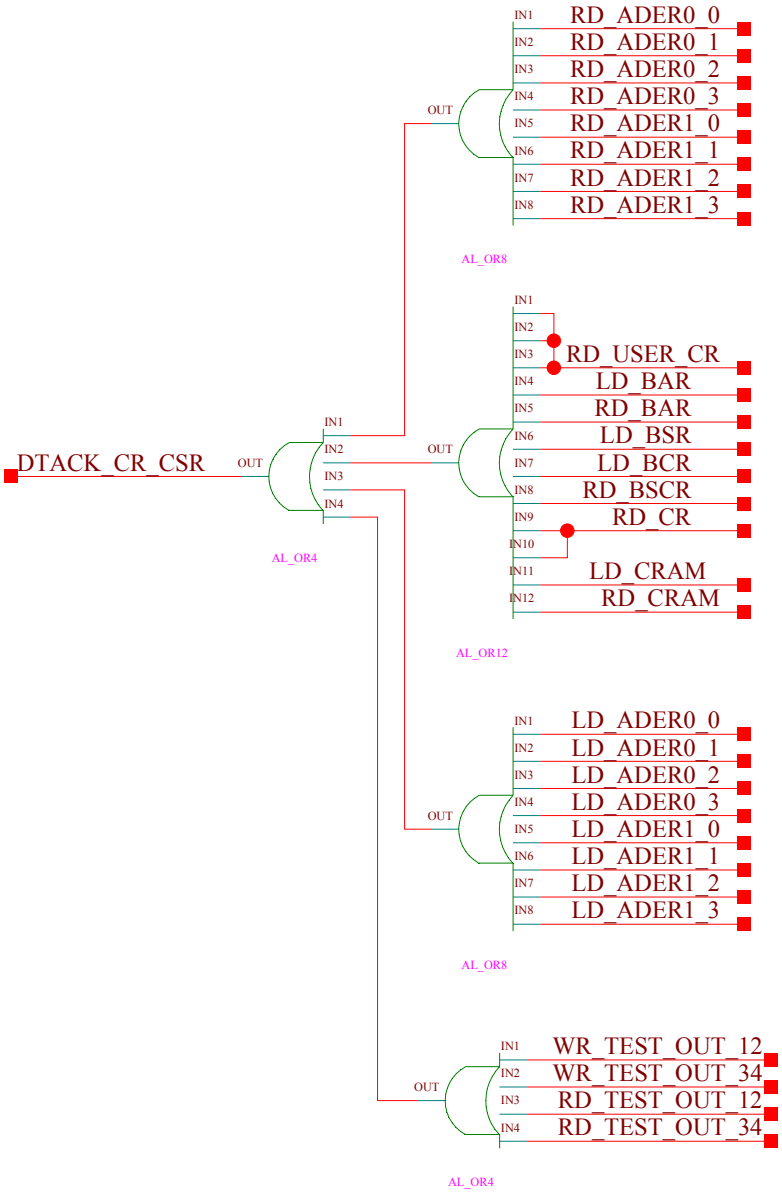
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	8-29-2005_9:32
checked by: CHECKER	0-00-0000_00:00



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

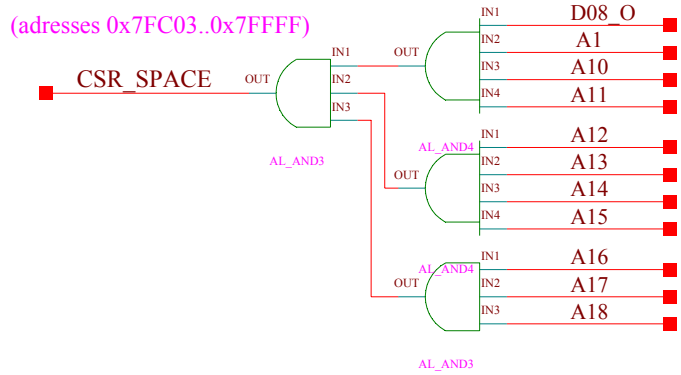


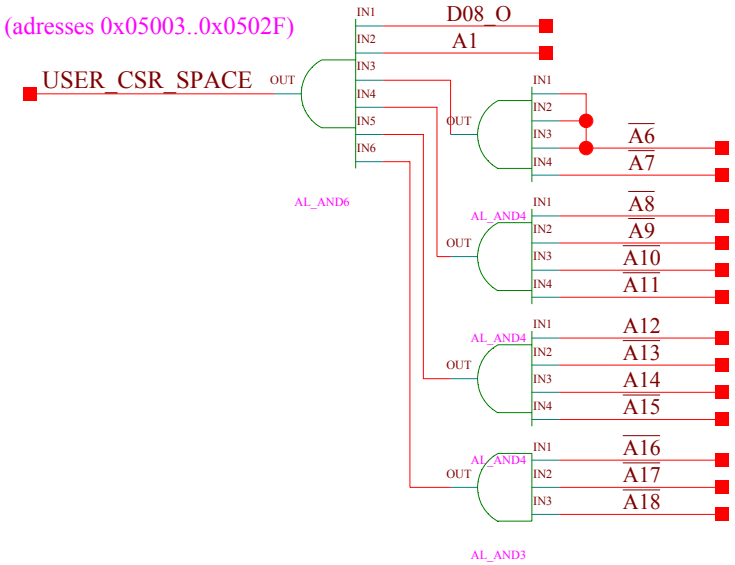
no BERR for CR/CSR access!!

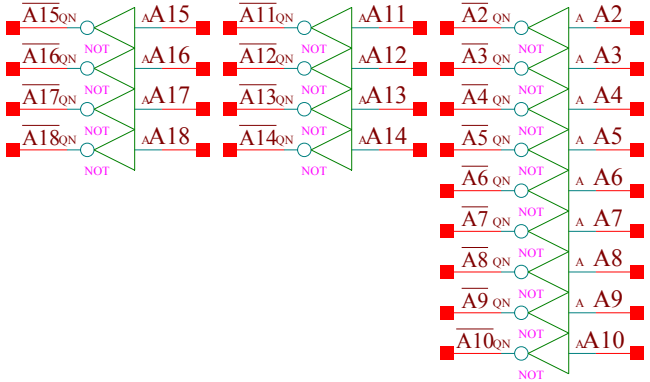
VME64X-CHIP

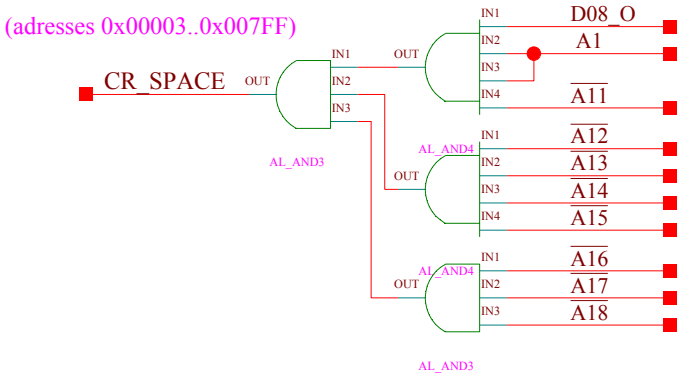
CR_CSR_INSTR

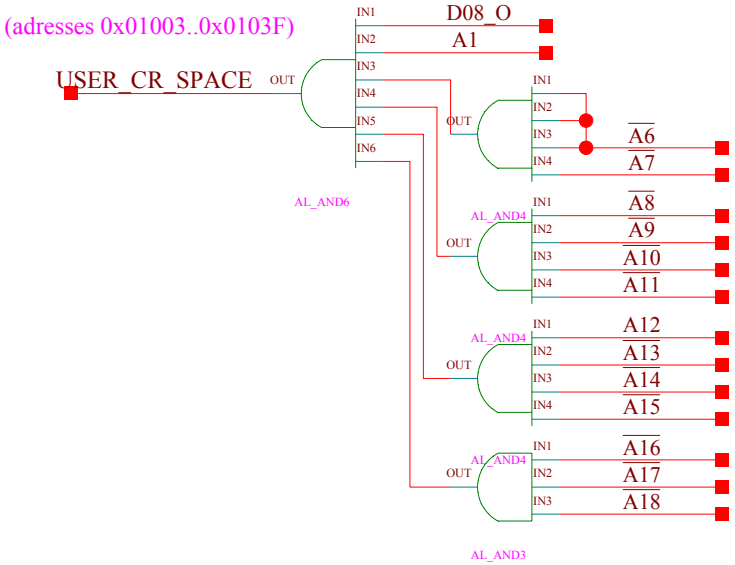
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00

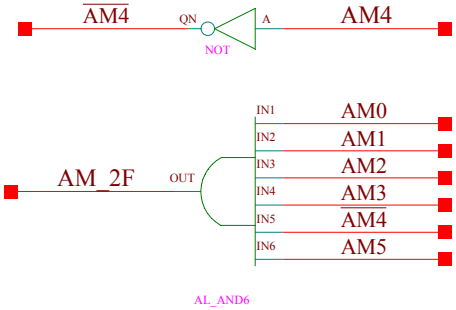






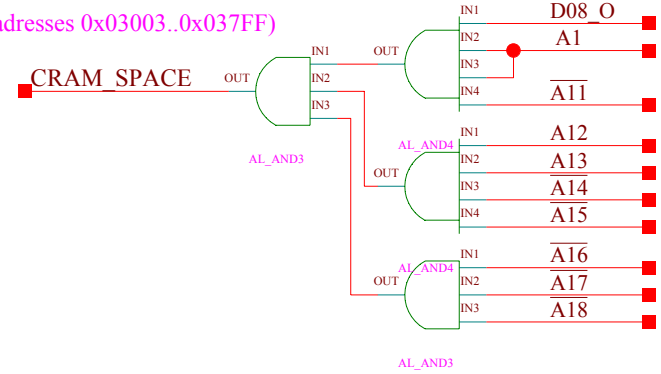






A[18:1]

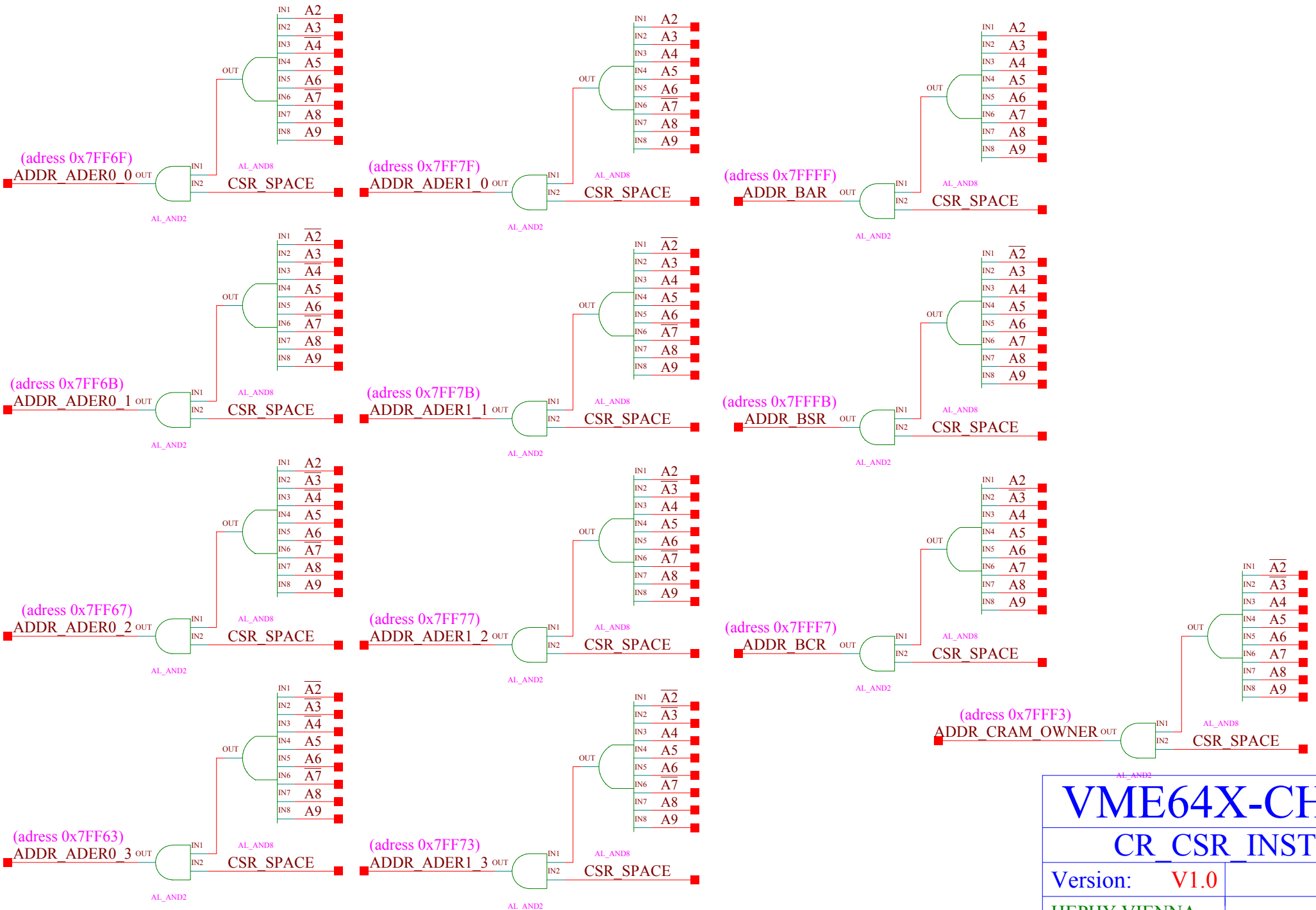
AM[5:0]



VME64X-CHIP

CR_CSR_INSTR

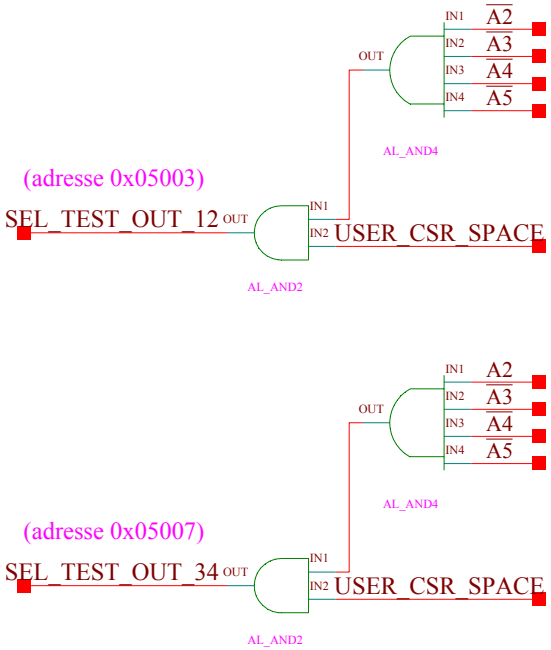
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP

CR_CSR_INSTR

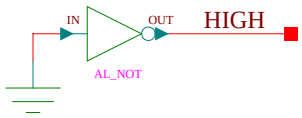
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 4 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00

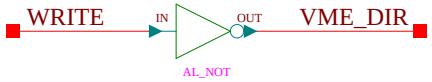


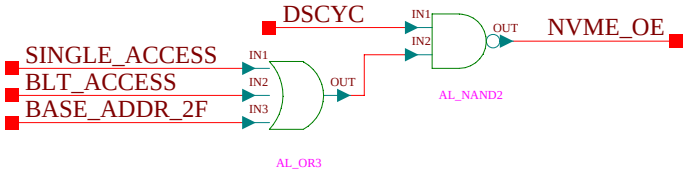
VME64X-CHIP

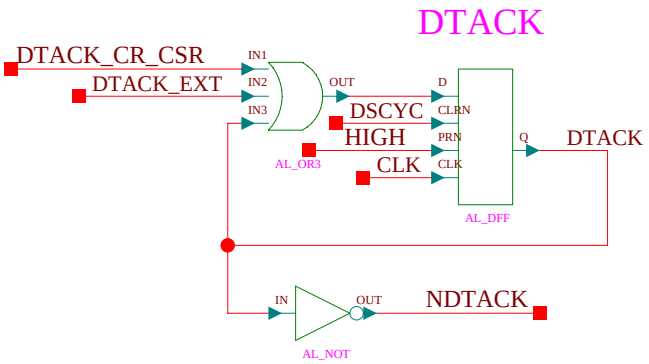
CR_CSR_INSTR

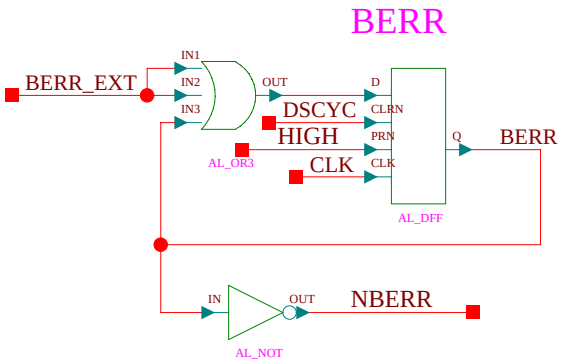
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 5 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00







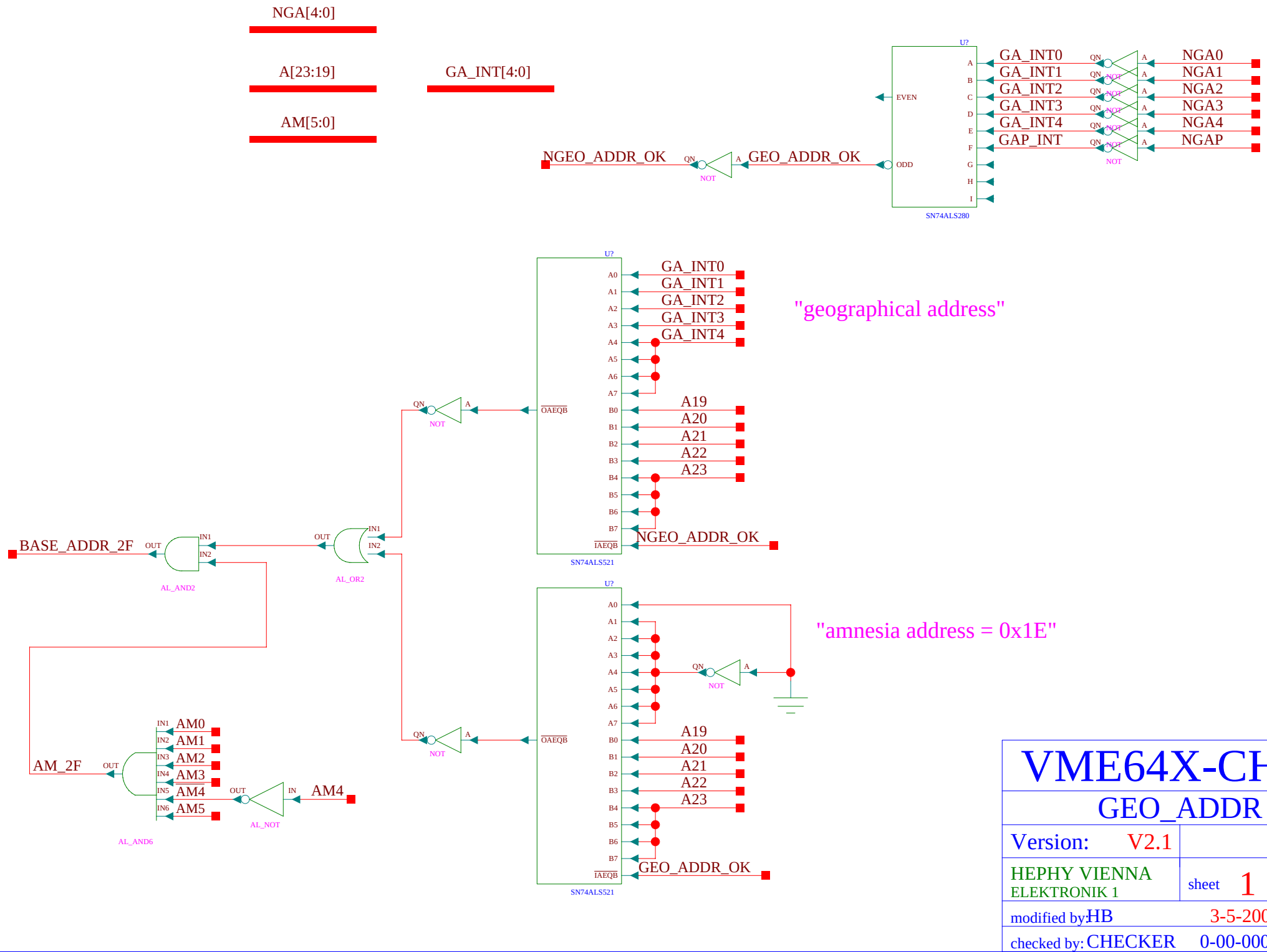




VME64X-CHIP

DTACK_BERR

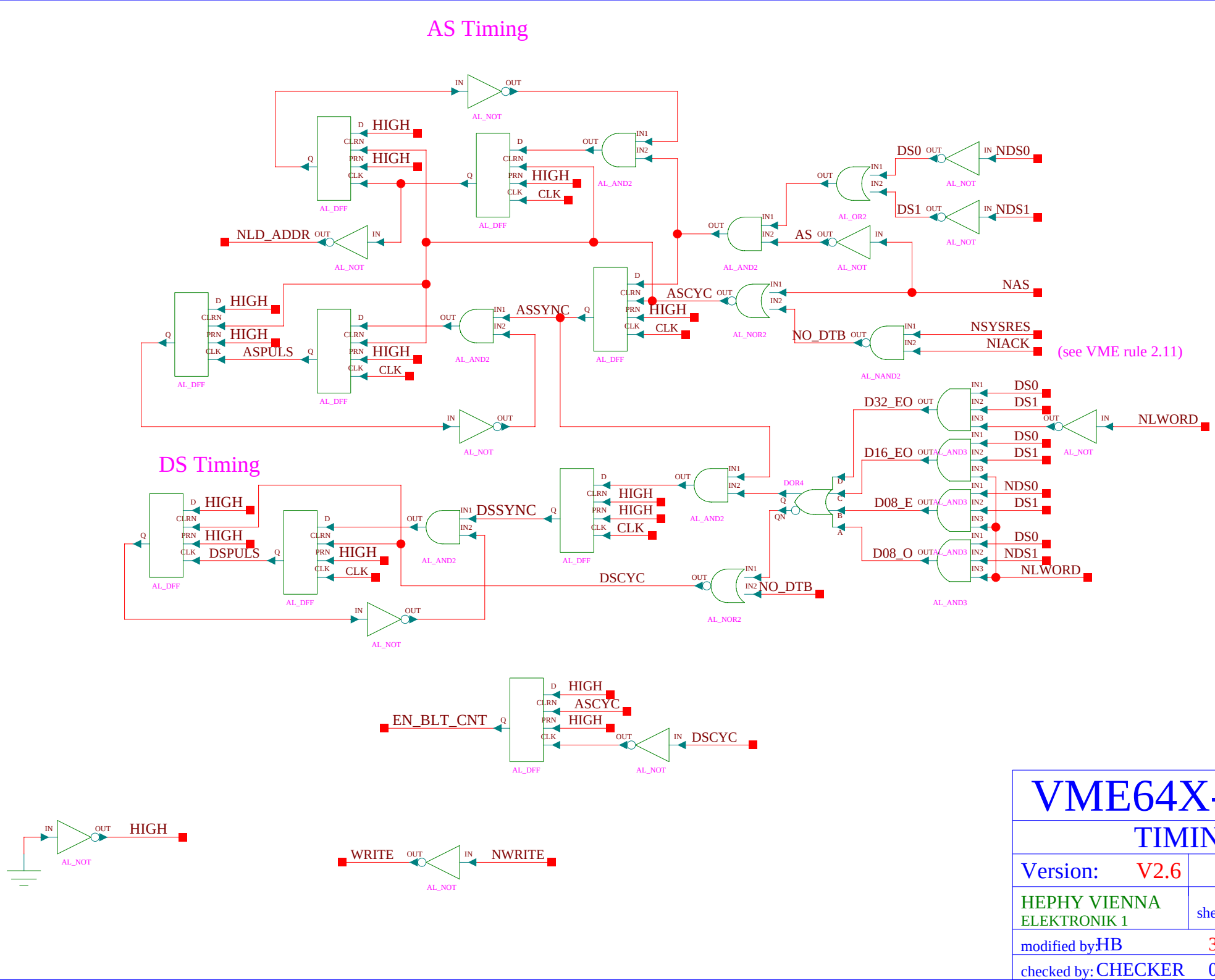
Version: V1.1	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	3-13-2007_14:47
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP

GEO_ADDR

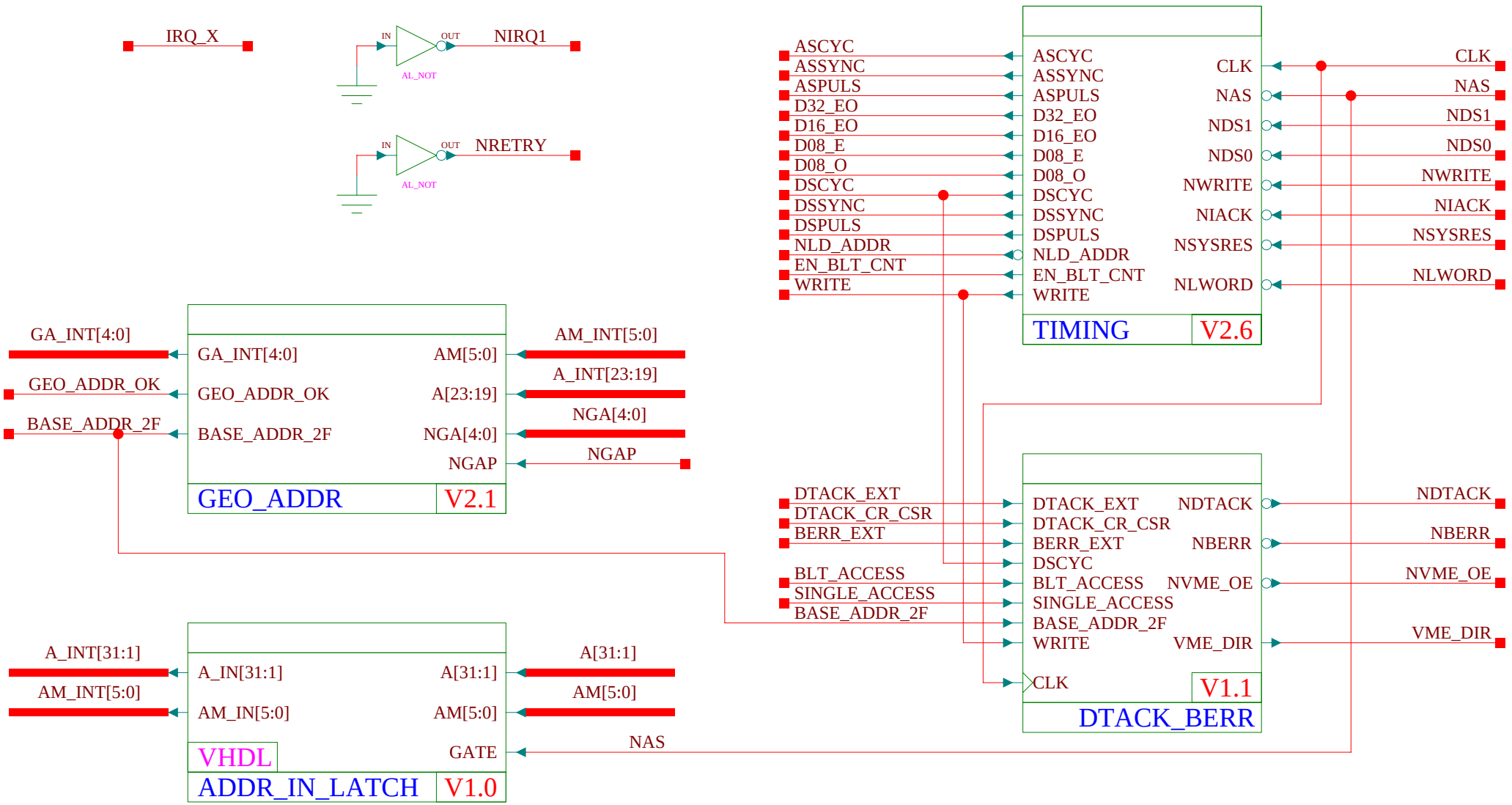
Version: V2.1	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	3-5-2007_13:00
checked by: CHECKER	0-00-0000_00:00



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;
```

```
------------------------------------------------------------------
--                                                              --
-- 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 DOWNTO 1);
 ader0_am <= ader0_0_out(7 DOWNTO 2);

 ader1_a <= ader1_3_out(7 DOWNTO 1);
 ader1_am <= ader1_0_out(7 DOWNTO 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 DOWNTO 3) <= ga(4 DOWNTO 0);
   bar_in(2 DOWNTO 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;
```

```
-- ********************************************************************************
-- Specify values for addresses of User Configuration ROM for VME64x of TCS-9u-cards
-- Only every forth address is used. D08_O = 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;
```