Inhaltsverzeichnis: PSB-9U-info

PSB-9U-card:

Beschreibung PSB-9U-card

Schaltpläne PSB-9U-card

Front PSB-9U-card

Topview PSB-9U-card

Bottomview PSB-9U-card

Beschreibung jumper und switches

VME64X-chip:

Beschreibung VME64X-chip (Version V100C)

Schaltpläne VME64X-chip (Version V100C)

MIF-Dateien

VME-PSB-chip:

Beschreibung VME-PSB-chip (Version V1007)

Schaltpläne VME-PSB-chip (Version V1007)

PSB
Pipelined Synchronising Buffer Module

9U-Version

H. Bergauer, K. Kastner, M. Padrta, A. Taurok

Aug-05

Version 1.1
with
PSB chip V0005

1 PSB Module 9U description

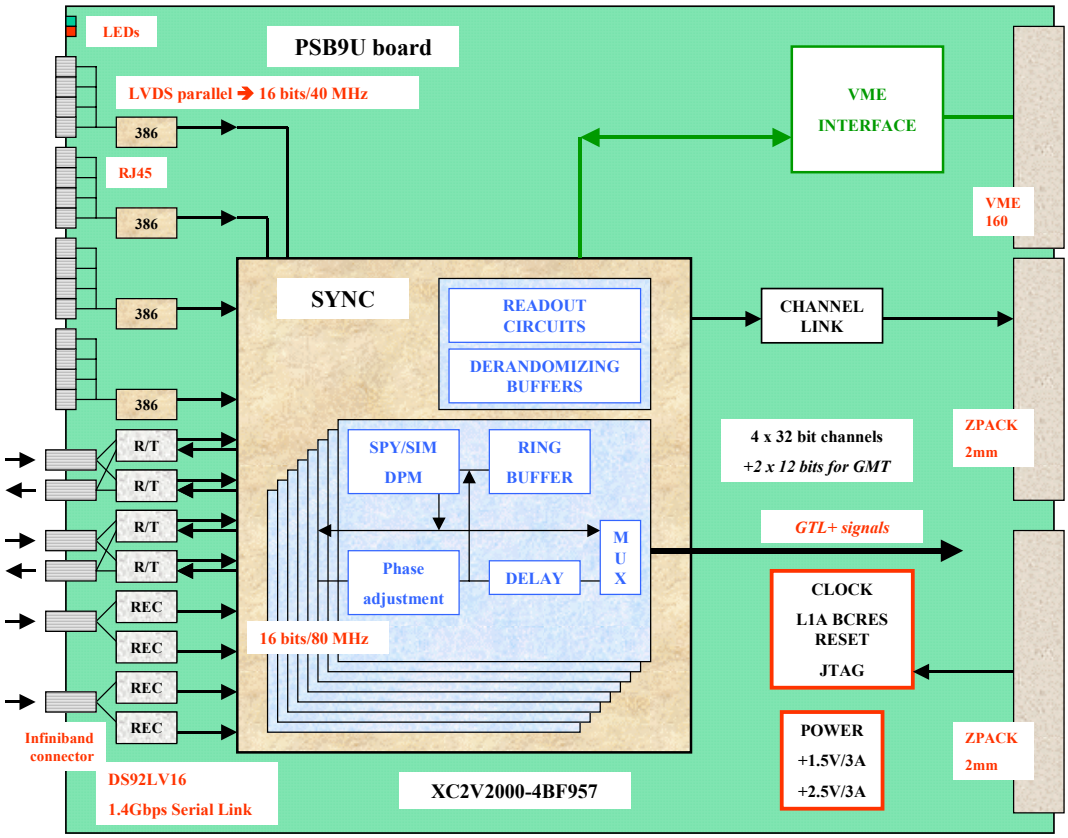


Figure 1 : PSB Overview

The PSB9U board receives 1.4 Gbps serial data from the Calorimeter trigger system, converts the serial bits into 16 bit words, synchronizes the words to the local clock, applies a programmable delay and sends the trigger objects as 80 MHz GTLp signals via the back-plane to the destination boards. Therefore each data channel contains the data bits of two time-multiplexed trigger objects.
The PSB9U boards are used by the Global Trigger as well as by the Global Muon Trigger.
Four Infiniband connectors forward the serial data to 8 DS92LV16 interface chips. After the serial to parallel conversion the 80 MHz data streams enter two Synchronization chips (=SYNC) where all the synchronization and monitoring logic is implemented.
The Synchronization Chip contains the over-sampling circuit that samples each word 4 times that means every 12.5/4 = 3.01 ns. The sample furthest from the data switching time is connected to the following programmable delay. After the delay the trigger words go through a multiplexer and then as terminated GTLp signals to the back-plane. The multiplexer circuit allows sending test data instead of trigger data to the back-plane.

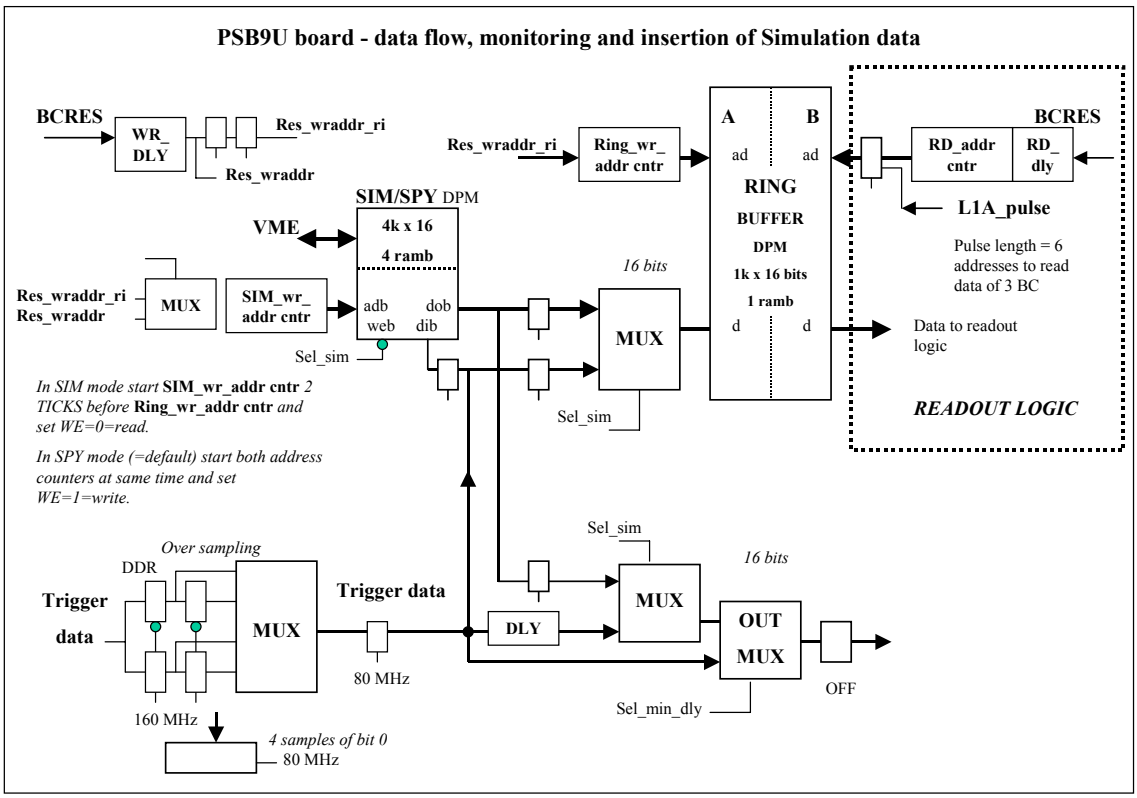


Figure 2 PSB9U data flow and monitoring scheme

For each 16-bit data stream a Ring Buffer memory monitors the trigger data. The memory runs in dual port mode. At one side the delayed trigger data are written continuously into the memory using addresses generated by a counter. The common Bunch Crossing Reset (BCRES) signal clears the counter synchronously to the clock thereby locking the addresses to the LHC orbit. After the end of the memory the write procedure continues at the first address overwriting old data. At the other side of the memory the DAQ-readout circuit applies a read address to get the data of the bunch crossing that has generated a L1A trigger signal. The read address is also supplied by a counter running synchronously to the LHC orbit. But a delayed BRES signal clears the read counter later than the write counter to compensate the trigger latency, that is the period between the writing time and a read access due to a L1A signal generated by trigger data of this bunch crossing. The length of the Ring Buffer memory (256 words) far exceeds the trigger latency.

Each (pair of) 16-bit channel(s) is locked separately to the LHC clock and to the orbit to compensate different cable delays.

The SYNC chip contains a second set of memories to load test or simulation data and to send them either once or repeatedly instead of trigger data to the Ring-Buffer and via the back-plane to the GTL respectively to the GMT board.

The readout circuits and a Derandomizing Buffer are either implemented in the SYNC chip or in a dedicated ROP chip if more than one SYNC chip per board is required.

1.1 Parallel LVDS input data

The PSB9U accepts also up to 64 bits of parallel trigger data alternatively to data from one Infiniband cable. The 40 MHz parallel input bits are accepted as LVDS signals received by 16 RJ45 connectors, each accepting 4 signals. The PSB board expects negative logic for

differential signals. **Trigger bit =1 is expected with a negative voltage difference.** The SYNC chip inverts the values back to the true input levels before transmission.

1.1.1 Synchronization of 40 MHz parallel trigger data

As the precise arrival time of the data bits is unknown the SYNC chip first samples the input bits 4 times per bunch-crossing period (~25 ns) to find the switching point of the input data. Phase selection and delay adjustment are done separately for each 4-bit group to consider time skew between cables and link chips. The SYNC chip takes the sample furthest away from the switching time, delays it for a programmable period, multiplexes the data into a 80 MHz data stream and sends the data as GTL+ signals over the back-plane to the logic board (GTL).
The SYNC chip also writes the input data into a RING BUFFER and in addition into a SPY memory for monitoring. The Ring Buffer runs continuously overwriting old data. If a Level-1 Accept (L1A) signal arrives at the PSB board, data are moved from the Ring Buffer into a Derandomizing Memory to be transferred later by the Readout Processor (ROP) to the readout board (GTFE).

1.1.1.1 Totem Trigger bits

Up to 16 bits are booked be the TOTEM Trigger (May 2004).

1.1.1.2 Free trigger bits (48)

48 bits are free (May 2004).

1.1.2 Synchronization of Technical Trigger bits

One PSB module accepts 'Technical Trigger' signals, which might not arrive as 40 MHz pulses. Therefore an edge sensing circuit generates synchronously to the system clock signal a 25ns pulse if either a rising or falling edge has been detected.

2 Keywords

2.1 RESET Signals

POWER OFF and ON
To switch the GT-crate off is the last option to reset non-working Global Trigger electronics.

NSYSRES ➔ configuration of FPGAs

The common crate-reset signal NSYSRES starts the configuration procedure for all FPGAs except the VME64 chip. It pulls the NPROG net to a low voltage level forcing the VME and the PSB chip to reconfigure from Proms.

RESET_DCM_PSB

The VME chip sends **RESET_DCM_PSB** to resynchronize the clock in the PSB chip to the board CLK.

RESET_PSB and INACTIVE ➔STARTUP of PSB chip

The VME chip sends **RESET_PSB** to reset the STARTUP module inside the PSB chip and the common INACTIVE signal enables the IO-pins to switch from high-Z to active mode.
RESET_PSB ➔ GSR pin of STARTUP ➔ set initial default values of registers
INACTIVE ➔ GTS pin of STARTUP
The RESET_PSB should reload the initial default values into all registers.

L1RES, BCRES, L1A ➔ reset State Machines and Counters

The Trigger Control System sends via the backplane the signals L1RES, BCRES, L1A and Event Counter Reset to reset state machines and counters

- L1RES clears the event data in the FIFOs.
- The ROC state machine returns always to IDLE state and cannot be reset. It stays in IDLE mode when the FIFOs are empty or when READY =0 (defined by software).

.

2.2 BCRES signal

It is possible that the distributed BCRES signal does not arrive every LHC orbit. An internal BC-counter and an Orbit_Length register are used to generate an internal BCRES signal running in phase with the distributed signal. Every circuit uses the internal BCRES signal to remain locked to the LHC orbit.

2.3 Sync check for BC0 data

Spy logic stores the arrival time of the SYNC bits as defined in the interface note CMS-IN-02-069.pdf for each 16 bit word. The number can be read by VME. A difference to a default value will be flagged as an error.

2.4 Phase check with over-sampling bits

All four samples of bit 0 are spied and compared to each other to find the time when the input data change their state. Four phase counters are incremented to check the stability of trigger data. At the end of every LHC orbit the counter contents are saved to be read by VME and the counters cleared. The contents are used to decide which sample should be taken as reliable data input.

2.5 Private Monitoring (option)

The 8kwords long SIM memory can run also in SPY mode to store a complete LHC orbit. In spy-mode the SIM/SPY memory will be written in parallel with the Ring-Buffer but can be read by VME. It can be used to get a 'snap shot' of the input data.

2.6 Test modes

- Read-back registers and memories in the SYNC chip allow checking VME accesses.
- The SIM/SPY memory is used to send test- or simulation data to other boards.
- VME generated BCRES pulses and an on-board 40 MHz oscillator are used to run tests also in stand-alone mode.

2.7 SIM/SPY Memories and serial outputs

For each channel a 8kW SIM/SPY memory can be used either to spy input data or to send simulation data to the back-plane and for channels 0…3 also to the transmitter part of four DS92LV16 Serial Link Chips. Two output connectors are foreseen to send test data to channels 4 to 7 on the same board or to other PSB boards.

2.8 Configuration modes

2.8.1 PROM and JTAG

PROMS contain the default configuration that is loaded

- at start-up time or whenever
- a SYSRESET signal arrives from the VME or by a
- NPROG command sent by software.

The Proms are be loaded by JTAG.

- JTAG connectors are foreseen for the Parallel-CableIV interface that is used to download the configuration file from a Notebook-PC.
- JTAG via VME loads the PROMS via the emulated JTAG interface in the VME chip. A configuration program takes the configuration files and loads the data serially into the Proms.

The configuration options above require that the PSB chip has been set by solder-jumpers to MASTER MODE.

2.8.2 Other configuration options

For tests other configuration options are possible, but then the PSB chip has to be re-soldered to SLAVE mode. Using an optional modification on the PSB Mezzanine board it will/might also be possible to switch by a software command to Slave Mode.

2.9 Power

+5V and +3.3V are provided by the backplane. Linear voltage generators provide +2.5V and +1.5V for the FPGA chips.

2.10 Front Panel

240 mm = 4x60mm …4 4-fold Ethernet connectors
80 mm = 6 x 15 …6 Infiniband conn
7 mm LEMO connector
0mm = LEDs mounted above Infiniband connectors
= 327 mm

2.11 Sync IO-pins, ram-blocks

XC2V2000-4 BF957 mounted on MEZZ957 board
56 ramb; Mezz957: 641 io-pins connected to PSB9U board
565 IO-pins:
8x16 in+ 4x16 par_in + 4x16 out (DS92LV16; test)
+ (128+24)GTLp + 21 VREF+ 59 VME(32bit)
+ 29 ChLink + 16 RO-bus
+ 3(L1A,BCRES,L1RESET) +4 Status + 5Config + 20 VRN/VRP

50 RAM blocks:
8x4 Spy + 8 Ring + 8 Derand + (1Ring+1Derand for BCnr)
Other chip types: xc2v1500: 48 ramb; xc2v3000: 96 ramb; Mezz896:ca 600io
It might be required to use a XC2V3000-4 BF957 to implement all circuits.

3 VME chip PSB - V1000

4 PSB chip Adresses

4.1 Overview VME Addresses

A23-20: = 0001 …PSB chip

(A23-20: = 0002 …PSB chip memories is not used in present design)

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	Register space (1kx16 readback)										
0	0	0	0	0	0	0	0	1	Read only status registers										
0	0	1	0	0	0	Sim_Spy_memory 0 (8kx16)													
0	0	1	0	0	1	Sim_Spy_memory 1 (8kx16)													
0	0	1	0	1	0	Sim_Spy_memory 2 (8kx16)													
0	0	1	0	1	1	Sim_Spy_memory 3 (8kx16)													
0	0	1	1	0	0	Sim_Spy_memory 4 (8kx16)													
0	0	1	1	0	1	Sim_Spy_memory 5 (8kx16)													
0	0	1	1	1	0	Sim_Spy_memory 6 (8kx16)													
0	0	1	1	1	1	Sim_Spy_memory 7 (8kx16)													

4.2 Sim Spy Memories

Remark: In present chip design the memories are also in same address space as the registers.

BB12 0000 SIM_SPY_MEM0 w/r
BB12 4000 SIM_SPY_MEM1 w/r
BB12 8000 SIM_SPY_MEM2 w/r
BB12 C000 SIM_SPY_MEM3 w/r
BB13 0000 SIM_SPY_MEM4 w/r
BB13 4000 SIM_SPY_MEM5 w/r
BB13 8000 SIM_SPY_MEM6 w/r
BB13 C000 SIM_SPY_MEM7 w/r

4.3 Register overview

BB10 0000 CHAN_REG0 w/r
BB10 0002 CHAN_REG1 w/r
BB10 0004 CHAN_REG2 w/r
BB10 0006 CHAN_REG3 w/r
BB10 0008 CHAN_REG4 w/r
BB10 000A CHAN_REG5 w/r
BB10 000C CHAN_REG6 w/r
BB10 000E CHAN_REG7 w/r

***** Delay Registers for Serial Link channels*****

BB10 0010 CHAN_DELAY0 w/r
BB10 0012 CHAN_DELAY1 w/r
BB10 0014 CHAN_DELAY2 w/r
BB10 0016 CHAN_DELAY3 w/r

BB10 0018 CHAN_DELAY4 w/r
BB10 001A CHAN_DELAY5 w/r
BB10 001C CHAN_DELAY6 w/r
BB10 001E CHAN_DELAY7 w/r

***** Delay Registers for parallel LVDS data channels*****

BB10 0020 LVDS_DELAY0 w/r // bit 3-0
BB10 0022 LVDS_DELAY1 w/r // bit 7-4
BB10 0024 LVDS_DELAY2 w/r
BB10 0026 LVDS_DELAY3 w/r
BB10 0028 LVDS_DELAY4 w/r
BB10 002A LVDS_DELAY5 w/r
BB10 002C LVDS_DELAY6 w/r
BB10 002E LVDS_DELAY7 w/r
BB10 0030 LVDS_DELAY8 w/r
BB10 0032 LVDS_DELAY9 w/r
BB10 0034 LVDS_DELAY10 w/r
BB10 0036 LVDS_DELAY11 w/r
BB10 0038 LVDS_DELAY12 w/r
BB10 003A LVDS_DELAY13 w/r
BB10 003C LVDS_DELAY14 w/r
BB10 003E LVDS_DELAY15 w/r // bit 63-60

***** Setup Registers *****

BB10 0040 BOARD_ID w/r
BB10 0042 BCRES_DELAY w/r
BB10 0044 LATENCY_DELAY w/r
BB10 0046 ROP_SETUP w/r
BB10 0048 MAX_BC_NUMBER w/r //=orbit length -1
BB10 004A SEL_PHASE3100 w/r // select phases for LVDS data
BB10 004C SEL_PHASE6332 w/r // select phases for LVDS data
BB10 004E IDLE_ID_LOW w/r // idle identifier low part

BB10 0050 IDLE_ID_HIGH w/r // idle identifier high part
BB10 0052 TESTMASK0 w/r // tespoint 0 bits
BB10 0054 TESTMASK1 w/r // tespoint 1 bits
BB10 0056 TESTMASK2 w/r // tespoint 2 bits
BB10 0058 TESTMASK3 w/r // tespoint 3 bits
BB10 005A TESTMASK4 w/r // tespoint 4 bits
BB10 005C TESTMASK5 w/r // tespoint 5 bits
BB10 005E TESTMASK6 w/r // tespoint 6 bits

***** Write Only Command Pulses ******

BB10 0070 CMD_PULSE w/-

++++++++++++++++++ READ ONLY ADDRESSES ++++++++++++++++

***** Phase Counters for Serial Link channels ******

BB10 0800 PHASE_CNTR_A0 -/r // compares ph1-ph0 and ph0-pre3
BB10 0802 PHASE_CNTR_ A1 -/r
BB10 0804 PHASE_CNTR_A2 -/r
BB10 0806 PHASE_CNTR_A3 -/r
BB10 0808 PHASE_CNTR_A4 -/r
BB10 080A PHASE_CNTR_A5 -/r

BB10 080C	PHASE_CNTR_A6	-/r	
BB10 080E	PHASE_CNTR_A7	-/r	
BB10 0810	PHASE_CNTR_B0	-/r	compares ph3-ph2 and ph2-1
BB10 0812	PHASE_CNTR_B1	-/r	
BB10 0814	PHASE_CNTR_B2	-/r	
BB10 0816	PHASE_CNTR_B3	-/r	
BB10 0818	PHASE_CNTR_B4	-/r	
BB10 081A	PHASE_CNTR_B5	-/r	
BB10 081C	PHASE_CNTR_B6	-/r	
BB10 081E	PHASE_CNTR_B7	-/r	

***** Phase Counters for parallel LVDS data channels ******

BB10 0820	PHASE_CNTR_A0_3	// compares ph1-ph0 and ph0-pre3 of bits 0-3
BB10 0822	PHASE_CNTR_A4_7	// compares ph1-ph0 and ph0-pre3 of bits 4-7
BB10 0824	PHASE_CNTR_A8_11	
BB10 0826	PHASE_CNTR_A12_15	
BB10 0828	PHASE_CNTR_A16_19	
BB10 082A	PHASE_CNTR_A20_23	
BB10 082C	PHASE_CNTR_A24_27	
BB10 082E	PHASE_CNTR_A28_31	
BB10 0830	PHASE_CNTR_A32_35	
BB10 0832	PHASE_CNTR_A36_39	
BB10 0834	PHASE_CNTR_A40_43	
BB10 0836	PHASE_CNTR_A44_47	
BB10 0838	PHASE_CNTR_A48_51	
BB10 083A	PHASE_CNTR_A52_55	
BB10 083C	PHASE_CNTR_A56_59	
BB10 083E	PHASE_CNTR_A60_63	// compares ph1-ph0 and ph0-pre3 of bits 60_63
BB10 0840	PHASE_CNTR_B0_3	// compares ph3-ph2 and ph2-1 of bits 0-3
BB10 0842	PHASE_CNTR_B4_7	// compares ph3-ph2 and ph2-1 of bits 4-7
BB10 0844	PHASE_CNTR_B8_11	
BB10 0846	PHASE_CNTR_B12_15	
BB10 0848	PHASE_CNTR_B16_19	
BB10 084A	PHASE_CNTR_B20_23	
BB10 084C	PHASE_CNTR_B24_27	
BB10 084E	PHASE_CNTR_B28_31	
BB10 0850	PHASE_CNTR_B32_35	
BB10 0852	PHASE_CNTR_B36_39	
BB10 0854	PHASE_CNTR_B40_43	
BB10 0856	PHASE_CNTR_B44_47	
BB10 0858	PHASE_CNTR_B48_51	
BB10 085A	PHASE_CNTR_B52_55	
BB10 085C	PHASE_CNTR_B56_59	
BB10 085E	PHASE_CNTR_B60_63	// compares ph3-ph2 and ph2-1 of bits 60_63

***** Status registers ******

BB10 0860	PSB_STATUS	-/r
BB10 0862	ROP_STATUS	-/r
BB10 0864	CHIP_ID	-/r
BB10 0866	VERSION_NR	-/r
BB10 0868	CHIP_IDH	-/r

********** **END OF OVERVIEW** **********

4.4 8 Channel Registers

write & read		functions		
BB10 0000	CHAN_REG0	rx	tx	lvds
BB10 0002	CHAN_REG1	rx	tx	lvds
BB10 0004	CHAN_REG2	rx	tx	
BB10 0006	CHAN_REG3	rx	tx	
BB10 0008	CHAN_REG4	rx		
BB10 000A	CHAN_REG5	rx		
BB10 000C	CHAN_REG6	rx		
BB10 000E	CHAN_REG7	rx		

Channels 0,1: Parallel LVDS data can be received instead of the serial input data.
Channels 0,1,2,3: Simulation data can also be sent via the serial transmitter circuits to two front panel connectors.
Channels 4,5,6,7: receive serial data only.

If we select simulation mode (**sel_sim_mode=1)** for these channels then LVDS data are not transferred to the FDL board (Technical trigger bits) and not to the GTL board (TOTEM trigger bits).

bits 15-6 : free

bit 5: **en_trx_data**
=1: send data from SIM_SPY memory also via DS92LV16 Serial Link transmitter to the Frontpanel connector.

bit 4: **sel_contin_mode**
=1: The SIM_SPY memory runs continuously either sending simulation data or storing input data
=0: The SIM_SPY memory runs for one orbit only and stops afterwards.

bit 3: **sel_sim_mode**
=1: The SIM_SPY memory sends simulation data to the backplane and if enabled in channels 0-3 also to the DS92LV16 Serial Link transmitters.
=0: The SIM_SPY memory stores input data for monitoring tasks.
For channels 0 and 1 the simulation data will replace either serial or parallel trigger data.

bit 2: **sel_lvdsdata**
…is valid for channels 0 and 1 only. For channels 2…7 it has to be set =0 otherwise no data will be transferred.
=1: Send parallel LVDS data to the backplane (Technical Trigger data)
=0: Send data from the DS92LV16 Serial Link to the backplane (Calorimeter trigger data)

bit 1: **sel_phase(1)**

bit 0: **sel_phase(0)**
Select Phase in over-sampling circuit to forward the trigger input data from the serial link. Take the sample that is most far from the data switching time.
V0005: Only phases 0 and 2 can be selected because of timing problems in the chip.
00 = take phase 0, **10** = take phase 2, **01** = inhibit data, **11** = inhibit data

4.5 8 Delay Registers for Serial Link Channels

write & read

BB10 0010	CHAN_DELAY0
BB10 0012	CHAN_DELAY1
BB10 0014	CHAN_ DELAY2
BB10 0016	CHAN_DELAY3
BB10 0018	CHAN_ DELAY4
BB10 001A	CHAN_DELAY5
BB10 001C	CHAN_ DELAY6
BB10 001E	CHAN_DELAY7

4.5.1 Programming Guideline for DELAYS

15 - 12	11 - 8	7 - 4	3 - 0	
Delay C	Delay B	Delay A	Delay= 0…3	

Total Delay = Delay C + Delay B + Delay A + (0…3)

-- DELAY =0 ➔ 0000 0000 0000 0000
-- DELAY =1 ➔ 0000 0000 0000 0001
-- DELAY =2 ➔ 0000 0000 0000 0010
-- DELAY =3 ➔ 0000 0000 0000 0011
-- DELAY =C+B+A+3 ➔ CCCC BBBB AAAA 0011

-- For DELAY <4 the bits 15-4 have to be =0 !!
-- Bits 3,2 are always =0; are not decoded

Remark about tests with different delays:

The SRL16 works like a shift register with an output multiplexer that can switch each shift register bit to the output as selected by A3,2,1,0.
If we change from a short to a long delay then it is possible that the shifted signal appears a second time at the output. Therefore we have to wait until the signal has been moved out before applying the new longer delay. However in real life the delay will not be changed during a run.

4.6 16 Delay Registers for parallel LVDS data channels

BB10 0020	LVDS_DELAY0	w/r	// bit 3-0
BB10 0022	LVDS_DELAY1	w/r	// bit 7-4
BB10 0024	LVDS_DELAY2	w/r	
BB10 0026	LVDS_DELAY3	w/r	
BB10 0028	LVDS_DELAY4	w/r	
BB10 002A	LVDS_DELAY5	w/r	
BB10 002C	LVDS_DELAY6	w/r	
BB10 002E	LVDS_DELAY7	w/r	
BB10 0030	LVDS_DELAY8	w/r	
BB10 0032	LVDS_DELAY9	w/r	
BB10 0034	LVDS_DELAY10	w/r	
BB10 0036	LVDS_DELAY11	w/r	
BB10 0038	LVDS_DELAY12	w/r	
BB10 003A	LVDS_DELAY13	w/r	
BB10 003C	LVDS_DELAY14	w/r	

BB10 003E LVDS_DELAY15 w/r // bit 63-60

4.6.1 Programming Guideline for DELAYS

15 - 12	11 - 8	7 - 4	3 - 0	
Delay C	Delay B	Delay A	Delay= 0…3	

Total Delay = Delay C + Delay B + Delay A + (0…3)

-- DELAY =0 ➔ 0000 0000 0000 0000
-- DELAY =1 ➔ 0000 0000 0000 0001
-- DELAY =2 ➔ 0000 0000 0000 0010
-- DELAY =3 ➔ 0000 0000 0000 0011
-- DELAY =C+B+A+3 ➔ CCCC BBBB AAAA 0011

-- **For DELAY <4 the bits 15-4 have to be =0 !!**
-- Bits 3,2 are always =0; are not decoded

4.7 Board Identifier

BB10 0040 BOARD_ID **write & read**

16 bit word to identify the PSB board in the data record for CMS readout.

4.8 BCRES Delay

BB10 0042 BCRES_DELAY **write & read**

--

15 - 12	11 - 8	7 - 4	3 - 0	
Delay C	Delay B	Delay A	Delay= 0…3	

Total Delay = Delay C + Delay B + Delay A + (0…3)

-- DELAY =0 ➔ 0000 0000 0000 0000
-- DELAY =1 ➔ 0000 0000 0000 0001
-- DELAY =2 ➔ 0000 0000 0000 0010
-- DELAY =3 ➔ 0000 0000 0000 0011
-- DELAY =C+B+A+3 ➔ CCCC BBBB AAAA 0011

-- **For DELAY <4 the bits 15-4 have to be =0 !!**
-- Bits 3,2 are always =0; are not decoded

4.9 Latency Delay

BB10 0044 LATENCY_DELAY **write & read**

15 - 12	11 - 8	7 - 4	3 - 0	
Delay C	Delay B	Delay A	Delay= 0…3	

Total Delay = Delay C + Delay B + Delay A + (0…3)

-- DELAY =0 ➔ 0000 0000 0000 0000
-- DELAY =1 ➔ 0000 0000 0000 0001
-- DELAY =2 ➔ 0000 0000 0000 0010

-- DELAY =3 ➔ 0000 0000 0000 0011
-- DELAY =C+B+A+3 ➔ CCCC BBBB AAAA 0011

-- For DELAY <4 the bits 15-4 have to be =0 !!
-- Bits 3,2 are always =0; are not decoded

4.10 ROP Setup register

BB10 0046 ROP_SETUP **write & read**

Bit 15 – 4 are not used

Bit 3:**en_robus** =0 (default)

=1 enable the Bgo commands as the TIM board sends via the ROBUS

=0 the PSB uses the encoded command (L1Res, Bcres, L1A) signals sent by the TIM board.

Bit 2: **five_bx_event** =0 (default)

=1: A readout record contains data from 5 bunch crossings around the triggering bx (-2, -1, 0, +1, +2).

=0: A readout record contains data from 3 bunch crossings around the triggering bx (-1, 0, +1)

Bit 1 and bit 0: **PSB_MODE**

= 0 0 PSB is **DISCONNECTED** from readout (=default)

= 0 1 PSB is **BUSY** with other tasks and cannot receive any L1A for the time being. But the ROP, all counters and registers are correct to continue the data taking run.

= 1 0 PSB is **READY** and waits for the Bgo command 'RUN' to receive L1A and to send events to the GTFE board. For tests the 'RUN' command can also be simulated by a vme cmd pulse.

= 1 1 PSB sends **BAD CODE**…should never be set except for a test

4.11 MAX_BC_NUMBER

BB10 0048 MAX_BC_NUMBER w/r //=orbit length -1
Default value = 3563 dec = DEB hex
The number is used by a comparator to generate an internal bunch counter reset signal.
If it does not agree with external BCRES signal from the TIM board then a BC_ERROR flag will be set.
A BC_ERROR appears always with the first external BCRES and when sending a BCRes_vme signal. It has to be cleared by the command pulse 'Res_BC_error'.
Remark: If MAX_BC_NUMBER = 0 then no BCRES will be generated inside the chip and the power consumption increases by 1-2 A.

4.12 SEL_PHASES for LVDS bits 63-00

BB10 004A SEL_PHASE3100 w/r // select phases for LVDS data
BB10 004C SEL_PHASE6332 w/r // select phases for LVDS data

SEL_PHASE3100	**15,14**	**13,12**	**11,10**	**9, 8**	**7, 6**	**5, 4**	**3, 2**	**1, 0**
Selects phases for LVDS bits:	31-28	27-24	23-20	19-16	15-12	11 - 8	7 - 4	3 - 0
SEL_PHASE6332	**15,14**	**13,12**	**11,10**	**9, 8**	**7, 6**	**5, 4**	**3, 2**	**1, 0**

Selects phases for LVDS bits:	63-60	59-56	55-52	51-48	47-44	43-40	39-36	35-32

00 ➔ selects phase sample 0
01 ➔ selects phase sample 1
10 ➔ selects phase sample 2
11 ➔ selects phase sample 3

4.13 Idle Identifier low

BB10 004E IDLE_IDL **write & read**

IDLE_IDL(15:0) defines the bits 15-0 that are sent between data records over the Channel Links to the GTFE readout board.

4.14 Idle Identifier high

BB10 0050 IDLE_IDH **write & read**

IDLE_IDH(11:0) defines the bits 27-16 that are sent between data records over the Channel Links to the GTFE readout board.

IDLE_IDH(15:12) =B"0000" are not used.

4.15 TESTPOINTS

BB10 0052	TESTMASK0	w/r	// tespoint 0 bits
BB10 0054	TESTMASK1	w/r	// tespoint 1 bits
BB10 0056	TESTMASK2	w/r	// tespoint 2 bits
BB10 0058	TESTMASK3	w/r	// tespoint 3 bits
BB10 005A	TESTMASK4	w/r	// tespoint 4 bits
BB10 005C	TESTMASK5	w/r	// tespoint 5 bits
BB10 005E	TESTMASK6	w/r	// tespoint 6 bits

TESTMASK0…6 select the signals that should be connected to the test points to be monitored with an oscilloscope. If more than one signal per test point are selected then the signals are merged with an OR-function.

Changed for V0005:

bits	TESTMASK 0	TESTMASK 1	TESTMASK 2	TESTMASK 3
15	Clk40	Clk80	'0'	clr =/locked
14	BCRes_int	bc_error	bcres_dlyed	bcres_dlyed1
13	Res_Evnr	Res_Orbitnr_i	L1Res_int	run_rop
12	Run_next_orbit(0)	en_spy_0	we_spy_0	chout0(15)
11	L1Res_vme	L1A_int	L1A_int	L1A_int
10	psb_status(0)	psb_status(1)	psb_status(2)	psb_status(3)
9	'0'	'0'	vme_wr	vme_en_spy
8	daq_data(24)	daq_data(25)	daq_data(26)	daq_data(27)
7	write_fifo	read_fifo	store_fifo_data	sclr_fifo
6	clr_ring_rdaddr	clr_ring_wradr	'0'	inc_event_nr
5	rop_status(11)	rop_status(12)	rop_status(14)	rop_status(15)
4	rop_status(7)	rop_status(8)	rop_status(9)	rop_status(10)
3	inc_phas4(3)	inc_phas5(3)	inc_lvd0sph(3)	stat_reg0(3)
2	inc_phas4(2)	inc_phas5(2)	inc_lvd0sph(2)	stat_reg0(2)
1	inc_phas4(1)	inc_phas5(1)	inc_lvd0sph(1)	stat_reg0(1)
0	inc_phas4(0)	inc_phas5(0)	inc_lvd0sph(0)	stat_reg0(0)

See ROP_STATUS bits: 15= roc_is_idle, 14=run_rop, 13=0, 12=out_of_sync, 11=error, 10=warning, 9=full_fifo, 8-0 = empty fifos.
ROP internal signals:

write_fifo
read_fifo, sclr_fifo
store_fifo_data
inc_event_nr
clr_ring_rdaddr, clr_ring_wradr

bits	TESTMASK 4	TESTMASK 5	TESTMASK6
15	'0'	vme_en	vme_en
14	'0'	dtack	vme_wr
13	'0'	vme_we_spy(0)	vme_rd_spy(0)
12	'0'	rd_chan_reg(0)	wr_chan_reg(0)
11	'0'	rd_chan_delay(0)	wr_chan_delay(0)
10	'0'	rd_lvds_delay(0)	wr_lvds_delay(0)
9	'0'	rd_setup_reg(0)	wr_setup_reg(0)
8	'0'	rd_setup_reg1(0)	wr_setup_reg1(0)
7	'0'	rd_stat_regs(0)	w_cmd_pulse
6	'0'	rd_phase_cntb (0)	rd_phase_cnta (0)
5	'0'	rd_phase_a6332 (0)	rd_phase_a3100 (0)
4	'0'	rd_phase_b6332 (0)	rd_phase_b3100 (0)
3	'0'	Start_rop_vme	Stop_rop_vme
2	'0'	ResOrbnr_vme	reset_error_flag
1	'0'	ResEvnr_vme	res_bc_error
0	'0'	BCRes_vme	run_next_orbit

4.16 Command Pulses

BB10 0070 PSB_CMD_PULSE **write only**
A data bit =1 generates a spurious pulse to start any action in the PSB chip.
Bit 15: reset_error_flag
Bit14: stop_rop_vme // stop ROP state machine = 'stop run'
Bit13: start_rop_vme // start ROP state machine making and sending events
Bit12: Res_BC_error // reset BC error after startup and after BCRes_vme
Bit 11: Res_Evnr_vme // reset Event Number Counter per software
Bit 10: Res_Orbitnr_vme // reset Orbit Number Counter per software (not used)
Bit 9: BCRes_vme // BCRES per software (for test only)
Bit 8: L1Res_vme // simulate a L1Res (Resync) pulse

// in chip design: 'run_next_orbit(7:0)' = start sim_spy7…0 at next orbit
Bit 7: start sim_spy7 at next orbit //Start Sim_Spy memory at begin of next orbit
Bit 6: start sim_spy6 at next orbit
Bit 5: start sim_spy5 at next orbit
Bit 4: start sim_spy4 at next orbit
Bit 3: start sim_spy3 at next orbit
Bit 2: start sim_spy2 at next orbit
Bit 1: start sim_spy1 at next orbit
Bit 0: start sim_spy0 at next orbit

4.17 Phase Counters for Serial data

read only 32 8bit-counters

BB10 0800	PHASE_CNTR_A0	// compares ph1-ph0 and ph0-pre3 of chann 0
BB10 0802	PHASE_CNTR_ A1	// compares ph1-ph0 and ph0-pre3 of chann 1
BB10 0804	PHASE_CNTR_A2	
BB10 0806	PHASE_CNTR_A3	
BB10 0808	PHASE_CNTR_A4	
BB10 080A	PHASE_CNTR_A5	
BB10 080C	PHASE_CNTR_A6	
BB10 080E	PHASE_CNTR_A7	
BB10 0810	PHASE_CNTR_B0	// compares ph3-ph2 and ph2-1 of chann 0
BB10 0812	PHASE_CNTR_B1	// compares ph3-ph2 and ph2-1 of chann 1
BB10 0814	PHASE_CNTR_B2	
BB10 0816	PHASE_CNTR_B3	
BB10 0818	PHASE_CNTR_B4	
BB10 081A	PHASE_CNTR_B5	
BB10 081C	PHASE_CNTR_B6	
BB10 081E	PHASE_CNTR_B7	// compares ph3-ph2 and ph2-1 of chann 7

15 - 8	7 - 0	
Phase Counter 10	Phase Counter 0p3	PHASE_CNTR_Ax
Phase Counter 32	Phase Counter 21	PHASE_CNTR_Bx

x = 0…7 = Channel number

If the incoming data bit switches between two consecutive samples then a 8 bit Phase Counter will be incremented. If a phase counter becomes 'FF' then counting stops, showing an overflow. Reading of phase counters also clears their content.
Phase Counter A checks between sample pre3 and 0.
Phase Counter B checks between sample 0 and 1.
Phase Counter C checks between sample 1 and 2.
Phase Counter D checks between sample 2 and 3.
/pre3 = sample 3 of preceding 12.5 ns tick

4.18 Phase Counters for parallel LVDS data

read only 32 words for 64 8bit-counters

BB10 0820	PHASE_CNTR_A0_3	// compares ph1-ph0 and ph0-pre3 of bits 0-3
BB10 0822	PHASE_CNTR_A4_7	// compares ph1-ph0 and ph0-pre3 of bits 4-7
BB10 0824	PHASE_CNTR_A8_11	
BB10 0826	PHASE_CNTR_A12_15	
BB10 0828	PHASE_CNTR_A16_19	
BB10 082A	PHASE_CNTR_A20_23	
BB10 082C	PHASE_CNTR_A24_27	
BB10 082E	PHASE_CNTR_A28_31	
BB10 0830	PHASE_CNTR_A32_35	
BB10 0832	PHASE_CNTR_A36_39	
BB10 0834	PHASE_CNTR_A40_43	
BB10 0836	PHASE_CNTR_A44_47	
BB10 0838	PHASE_CNTR_A48_51	
BB10 083A	PHASE_CNTR_A52_55	
BB10 083C	PHASE_CNTR_A56_59	
BB10 083E	PHASE_CNTR_A60_63	// compares ph1-ph0 and ph0-pre3 of bits 60_63

BB10 0840 PHASE_CNTR_B0_3 // compares ph3-ph2 and ph2-1 of bits 0-3
BB10 0842 PHASE_CNTR_B4_7 // compares ph3-ph2 and ph2-1 of bits 4-7
BB10 0844 PHASE_CNTR_B8_11
BB10 0846 PHASE_CNTR_B12_15
BB10 0848 PHASE_CNTR_B16_19
BB10 084A PHASE_CNTR_B20_23
BB10 084C PHASE_CNTR_B24_27
BB10 084E PHASE_CNTR_B28_31
BB10 0850 PHASE_CNTR_B32_35
BB10 0852 PHASE_CNTR_B36_39
BB10 0854 PHASE_CNTR_B40_43
BB10 0856 PHASE_CNTR_B44_47
BB10 0858 PHASE_CNTR_B48_51
BB10 085A PHASE_CNTR_B52_55
BB10 085C PHASE_CNTR_B56_59
BB10 085E PHASE_CNTR_B60_63 // compares ph3-ph2 and ph2-1 of bits 60_63

4.19 PSB Status register

BB10 0860 PSB_STATUS **read only**
Bit 15- 5 unused
Bit 4: BC_error

- =1 if the external BCRES signal and the BC-counter disagree. The length of the orbit is defined by the MAX_BC_NUMBER content.
- BC_error appears always after the initial power-up or DCM(clock) reset, the first external BCRES or after a BCRES_vme command.
- If BC_error becomes =1 during normal run then there are serious hardware problems, due to instable electronics.
- It has to be cleared by the **command pulse 'Res_BC_error'** before starting a run.
- The **'Res_BC_error'** pulse has to be sent at least *1 orbit after having loaded a new value* into the MAX_BC_NUMBER register.

Bit 3 -0 : encoded 4 bit status sent via FDL to the TCS Trigger Control board.

Bit3 Ready	Bit2 Busy	Bit1 Out_of_Sync	Bit0 Warning	Status of PSB
0	0	0	0	Disconnected
0	0	0	1	Warning
0	0	1	0	Out_of_Sync error
0	0	1	1	----
0	1	0	0	Busy
0	1	0	1	---
0	1	1	0	---
0	1	1	1	---
1	0	0	0	Ready
1	0	0	1	---
1	0	1	0	---
1	0	1	1	---
1	1	0	0	Error (not used by PSB)

1	1	0	1	---
1	1	1	0	---
1	1	1	1	Disconnected

Encoded Status of PSB board

The table agrees with the TCS Note.

4.20 ROP Status register

BB10 0862 ROP_STATUS **read only**

Bit 15: roc_is_idle // The ROC Readout Controller state machine is in idle mode
Bit 14: run_flag // 1= ROC is running when software sets the
// ROP SETUP register = B"….10" = PSB READY=1
// 0= either software or Bgo command has stopped
// the ROC readout controller to extract and send events
Bit 13: 0
Bit 12: out_of_sync // empty bits of readout FIFOs did not appear at same time
Bit 11: error // currently not implemented
Bit 10: warning // more than 75% of the readout FIFO has been filled
Bit 9: full_fifo // readout FIFOs are full ➔ error!!
Bit 8: empty(8) // empty bit of readout FIFO 8 (BC number)
Bit 7: empty(7) // empty bit of readout FIFO for channel 7
Bit 6: empty(6)
Bit 5: empty(5)
Bit 4: empty(4)
Bit 3: empty(3)
Bit 2: empty(2)
Bit 1: empty(1)
Bit 0: empty(0) // empty bit of readout FIFO for channel 0

4.21 CHIP Identifier

BB10 0864 CHIP_ID **read only**

The 16 bit identifier is defined in the VHDL code for the PSB chip and cannot be changed by software.

CHIPID = 8131 8= PSB board, 1=cardnr, 3 = PSB chip, 1=chipnr (only one psb chip per board)

4.22 Version Number

BB10 0866 VERSION_NR **read only**

The 16 bit identifier is defined in the VHDL code for the PSB chip and cannot be changed by software.

VERSION_NR = 0001….and higher

4.23 CHIP Identifier H

BB10 0868 CHIP_IDH **read only**

The 16 bit identifier is defined in the VHDL code for the PSB chip and cannot be changed by software.

CHIPIDH = 0001 1= Global Trigger crate

5 PSB logic functions

5.1 Data Format of Channel Link

The table is defined for 5 bx per event. For normal events the record contains the parts for bx-1, bx+0, bx+1 only.

Normal record length=24 x 3 +1 = 73
Debug record length = 24 x 5 +1 = 121
Transfer time: 73 x 25 ns = 1800 ns resp. 121x 25= 3025 ns per event for 40 MHz Channel Link.

27-24	23-20	19-16	15-12	11-8	7-4	3-0	Name	Comment	Example
I	I	I	I	I	I	I	IDLE	Between records	555AAAA
A	0	0	e	e	e	e	HEADER A	EVNr(15:0)	A000001
B	0	0	0	0	e	e	HEADER B	EVNr(23:16)	0000000
C	0	0	bx-2	b	b	b	HEADER C	Bx_in_ev/bx of fifo	C00E017
D	0	0	n	n	n	n	HEADER_D	Board identifier	D00ABCD
1	0	0	d	d	d	d	A_data ch0 of bx-2		
1	0	0	d	d	d	d	A_data ch1 of bx-2		
1	0	0	d	d	d	d	A_data ch2 of bx-2		
1	0	0	d	d	d	d	A_data ch3 of bx-2		
1	0	0	d	d	d	d	A_data ch4 of bx-2		
1	0	0	d	d	d	d	A_data ch5 of bx-2		
1	0	0	d	d	d	d	A_data ch6 of bx-2		
1	0	0	d	d	d	d	A_data ch7 of bx-2		
1	0	0	d	d	d	d	B_data ch0 of bx-2		
1	0	0	d	d	d	d	B_data ch1 of bx-2		
1	0	0	d	d	d	d	B_data ch2 of bx-2		
1	0	0	d	d	d	d	B_data ch3 of bx-2		
1	0	0	d	d	d	d	B_data ch4 of bx-2		
1	0	0	d	d	d	d	B_data ch5 of bx-2		
1	0	0	d	d	d	d	B_data ch6 of bx-2		
1	0	0	d	d	d	d	B_data ch7 of bx-2		
E	0	0	*000b*	b	b	b	End of bx-2	Ring addr of B_data	E000018
E	0	0	0	0	0	0	End of bx-2		E000000
E	0	0	0	0	0	0	End of bx-2		E000000
E	0	0	0	0	0	0	End of bx-2		E000000
A	0	0	e	e	e	e	HEADER A	EVNr(15:0)	A000001
B	0	0	0	0	e	e	HEADER B	EVNr(23:16)	0000000
C	0	0	bx-1	b	b	b	HEADER C	Bx_in_ev/bx of fifo	C00F019
D	0	0	n	n	n	n	HEADER_D	Board identifier	D00ABCD
1	0	0	d	d	d	d	A_data ch0 of bx-1		
1	0	0	d	d	d	d	A_data ch1 of bx-1		
1	0	0	d	d	d	d	A_data ch2 of bx-1		
1	0	0	d	d	d	d	A_data ch3 of bx-1		
1	0	0	d	d	d	d	A_data ch4 of bx-1		
1	0	0	d	d	d	d	A_data ch5 of bx-1		
1	0	0	d	d	d	d	A_data ch6 of bx-1		
1	0	0	d	d	d	d	A_data ch7 of bx-1		
1	0	0	d	d	d	d	B_data ch0 of bx-1		
1	0	0	d	d	d	d	B_data ch1 of bx-1		
1	0	0	d	d	d	d	B_data ch2 of bx-1		
1	0	0	d	d	d	d	B_data ch3 of bx-1		
1	0	0	d	d	d	d	B_data ch4 of bx-1		
1	0	0	d	d	d	d	B_data ch5 of bx-1		
1	0	0	d	d	d	d	B_data ch6 of bx-1		
1	0	0	d	d	d	d	B_data ch7 of bx-1		
E	0	0	*000b*	b	b	b	End of bx-1	Ring addr of B_data	E00001A
E	0	0	0	0	0	0	End of bx-1		E000000
E	0	0	0	0	0	0	End of bx-1		E000000
E	0	0	0	0	0	0	End of bx-1		E000000
A	0	0	e	e	e	e	HEADER A	EVNr(15:0)	A000001
B	0	0	0	0	e	e	HEADER B	EVNr(23:16)	0000000

C	0	0	bx+0	b	b	b	HEADER C	Bx_in_ev/bx of fifo	C00001B
D	0	0	n	n	n	n	HEADER_D	Board identifier	D00ABCD
1	0	0	d	d	d	d	A_data ch0 of bx+0		
1	0	0	d	d	d	d	A_data ch1 of bx+0		
1	0	0	d	d	d	d	A_data ch2 of bx+0		
1	0	0	d	d	d	d	A_data ch3 of bx+0		
1	0	0	d	d	d	d	A_data ch4 of bx+0		
1	0	0	d	d	d	d	A_data ch5 of bx+0		
1	0	0	d	d	d	d	A_data ch6 of bx+0		
1	0	0	d	d	d	d	A_data ch7 of bx+0		
1	0	0	d	d	d	d	B_data ch0 of bx+0		
1	0	0	d	d	d	d	B_data ch1 of bx+0		
1	0	0	d	d	d	d	B_data ch2 of bx+0		
1	0	0	d	d	d	d	B_data ch3 of bx+0		
1	0	0	d	d	d	d	B_data ch4 of bx+0		
1	0	0	d	d	d	d	B_data ch5 of bx+0		
1	0	0	d	d	d	d	B_data ch6 of bx+0		
1	0	0	d	d	d	d	B_data ch7 of bx+0		
E	0	0	*000b*	b	b	b	End of bx	Ring addr of B_data	E00001C
E	0	0	0	0	0	0	End of bx		E000000
E	0	0	0	0	0	0	End of bx		E000000
E	0	0	0	0	0	0	End of bx		E000000
A	0	0	e	e	e	e	HEADER A	EVNr(15:0)	A000001
B	0	0	0	0	e	e	HEADER B	EVNr(23:16)	0000000
C	0	0	bx+1	b	b	b	HEADER C	Bx_in_ev/bx of fifo	C00101D
D	0	0	n	n	n	n	HEADER_D	Board identifier	D00ABCD
1	0	0	d	d	d	d	A_data ch0 of bx+1		
1	0	0	d	d	d	d	A_data ch1 of bx+1		
1	0	0	d	d	d	d	A_data ch2 of bx+1		
1	0	0	d	d	d	d	A_data ch3 of bx+1		
1	0	0	d	d	d	d	A_data ch4 of bx+1		
1	0	0	d	d	d	d	A_data ch5 of bx+1		
1	0	0	d	d	d	d	A_data ch6 of bx+1		
1	0	0	d	d	d	d	A_data ch7 of bx+1		
1	0	0	d	d	d	d	B_data ch0 of bx+1		
1	0	0	d	d	d	d	B_data ch1 of bx+1		
1	0	0	d	d	d	d	B_data ch2 of bx+1		
1	0	0	d	d	d	d	B_data ch3 of bx+1		
1	0	0	d	d	d	d	B_data ch4 of bx+1		
1	0	0	d	d	d	d	B_data ch5 of bx+1		
1	0	0	d	d	d	d	B_data ch6 of bx+1		
1	0	0	d	d	d	d	B_data ch7 of bx+1		
E	0	0	*000b*	b	b	b	End of bx+1	Ring addr of B_data	E00001E
E	0	0	0	0	0	0	End of bx+1		E000000
E	0	0	0	0	0	0	End of bx+1		E000000
E	0	0	0	0	0	0	End of bx+1		E000000
A	0	0	e	e	e	e	HEADER A	EVNr(15:0)	A000001
B	0	0	0	0	e	e	HEADER B	EVNr(23:16)	0000000
C	0	0	bx+2	b	b	b	HEADER C	Bx_in_ev/bx of fifo	C00201F
D	0	0	n	n	n	n	HEADER_D	Board identifier	D00ABCD
1	0	0	d	d	d	d	A_data ch0 of bx+2		
1	0	0	d	d	d	d	A_data ch1 of bx+2		
1	0	0	d	d	d	d	A_data ch2 of bx+2		
1	0	0	d	d	d	d	A_data ch3 of bx+2		
1	0	0	d	d	d	d	A_data ch4 of bx+2		
1	0	0	d	d	d	d	A_data ch5 of bx+2		
1	0	0	d	d	d	d	A_data ch6 of bx+2		
1	0	0	d	d	d	d	A_data ch7 of bx+2		
1	0	0	d	d	d	d	B_data ch0 of bx+2		

1	0	0	d	d	d	d	B_data ch1 of bx+2		
1	0	0	d	d	d	d	B_data ch2 of bx+2		
1	0	0	d	d	d	d	B_data ch3 of bx+2		
1	0	0	d	d	d	d	B_data ch4 of bx+2		
1	0	0	d	d	d	d	B_data ch5 of bx+2		
1	0	0	d	d	d	d	B_data ch6 of bx+2		
1	0	0	d	d	d	d	B_data ch7 of bx+2		
E	0	0	*000b*	b	b	b	End of bx+2	Ring addr of B_data	E000020
E	0	0	0	0	0	0	End of bx+2		E000000
E	0	0	0	0	0	0	End of bx+2		E000000
E	0	0	0	0	0	0	End of bx+2		E000000
F	F	F	F	F	F	F	END of RECORD		FFFFFFF
I	I	I	I	I	I	I	IDLE	Between records	555AAAA

5.2 RESET LOGIC

L1Res:

- **Clears FIFOs** (derandomizing buffers)
  L1Res either from backplane or from VME clears the FIFO content so that the ROC Readout Controller stays in IDLE mode after having finished the current event.

5.3 ROP logic

- ROC State Machine cannot be reset, it returns always to IDLE state. ROC starts only when FIFO is not empty and PSB is READY.

TTCrx receives BGo commands and sends them via the ROBUS to the boards.

Bit #	Signal name	Internal action when high	Coarse delay value 1=bits<3:0> 2=bits<7:4>	Output syn-chronised with 1: Clock40Des1 2: Clock40Des2	Output pin name
0	Bunch counter reset	Resets internal bunch counter	1	1	BcntRes
1	Event counter reset	Resets internal event counter	1	1	EvCntRes
<5:2>	System message	-	1	1	Brcst<5:2>
<7:6>	User message	-	2	1 or 2	Brcst <7:6>

TIM CHIP V1004, V1005 from July 2004

The TIM board sends via the RO bus to all boards

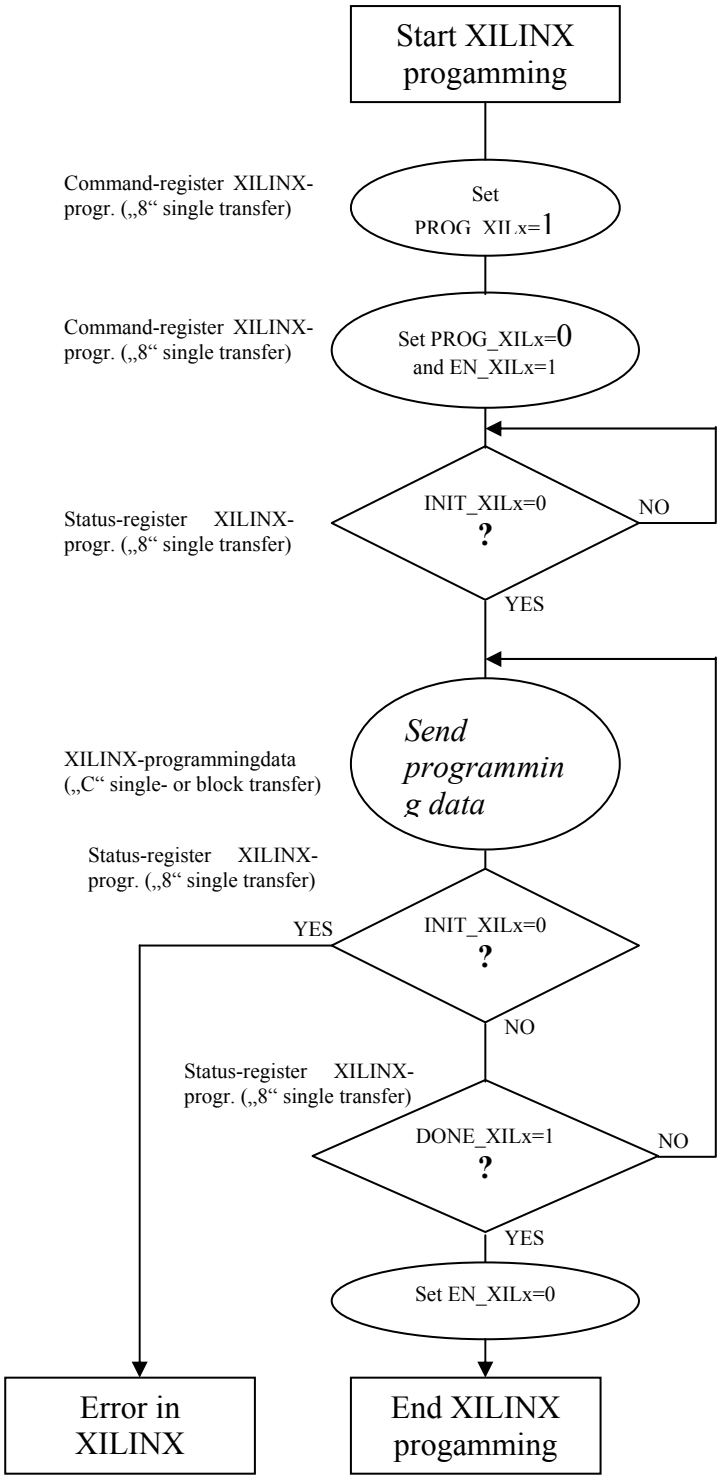
- BGo commands from TTCrx or VME on TIM
- Monitoring Request Identifier
- Test data that where loaded into the TIM register.

ROBUS	BGO cmds	Monitoring	Tests	
RDRQST	1	1	1	OR_STROBES
STROBE 2	0	0	1	TEST_STROBE
STROBE 1	0	1	0	MON_RQST_STROBE
STROBE 0	1	0	0	BGO_CMD_STROBE
BX 11	USER_MSG3	MON_RQST_ID 11	TEST 11	
BX 10	USER_MSG2	MON_RQST_ID 10	TEST 10	
BX 9	USER_MSG1	MON_RQST_ID 9	TEST 9	

BX 8	USER_MSG0	MON_RQST_ID 8	TEST 8	
BX 7	0	MON_RQST_ID 7	TEST 7	
BX 6	STOP_RUN	MON_RQST_ID 6	TEST 6	
BX 5	START_RUN	MON_RQST_ID 5	TEST 5	
BX 4	RES_ORBITNR	MON_RQST_ID 4	TEST 4	
BX 3	HARD_RES	MON_RQST_ID 3	TEST 3	
BX 2	PRIVATE_ORBIT	MON_RQST_ID 2	TEST 2	
BX 1	PRIVATE_GAP	MON_RQST_ID 1	TEST 1	
BX 0	TEST_ENABLE	MON_RQST_ID 0	TEST 0	

6 Flowchart of XILINX-programming

The XILINX chips on PSB-card are programmed via VME-instructions, this programming sequence has to happen every time after power up. There is no PROM on board for power-up-programming! The following instructions should be implemented in the control software.



Beginning of configuration in enabled XILINX chip(s).

Checking whether XILINX chip(s) is (are) ready for configuration.

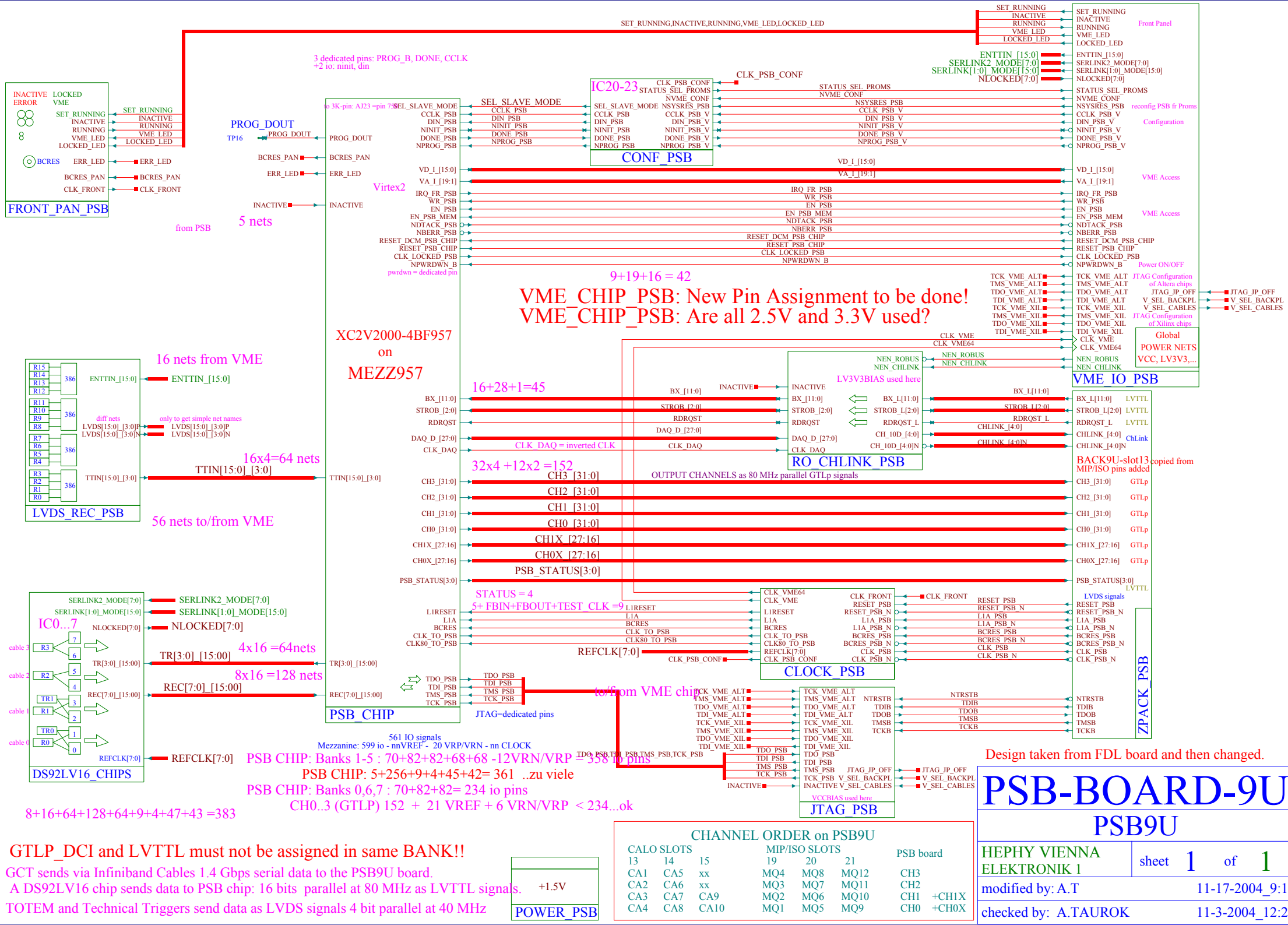
Sending programming data of XILINX chip(s) on D0. Programming data file comes from XILINX software. Setup-file has to implement these data and send it via single- or blocktransfer.

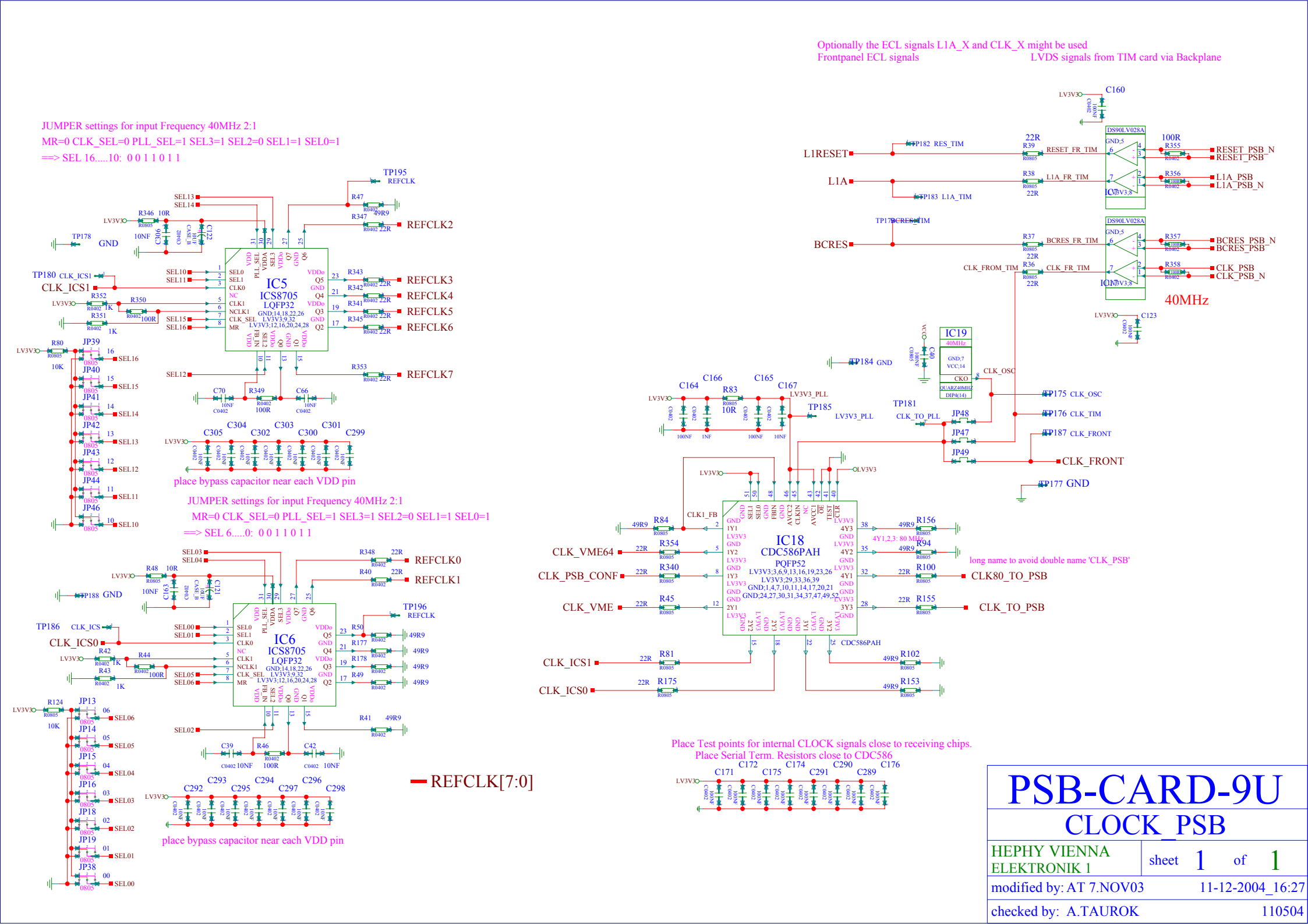
Checking errors during programming of XILINX chip(s).

Checking end of programming data frames of XILINX chip(s).

Ending programming sequence of XILINX chip(s).

After the XILINX programming sequence all other VME-accesses on card are possible, e. g. DPM-access or FIFO-access in XILINX chips and so on.

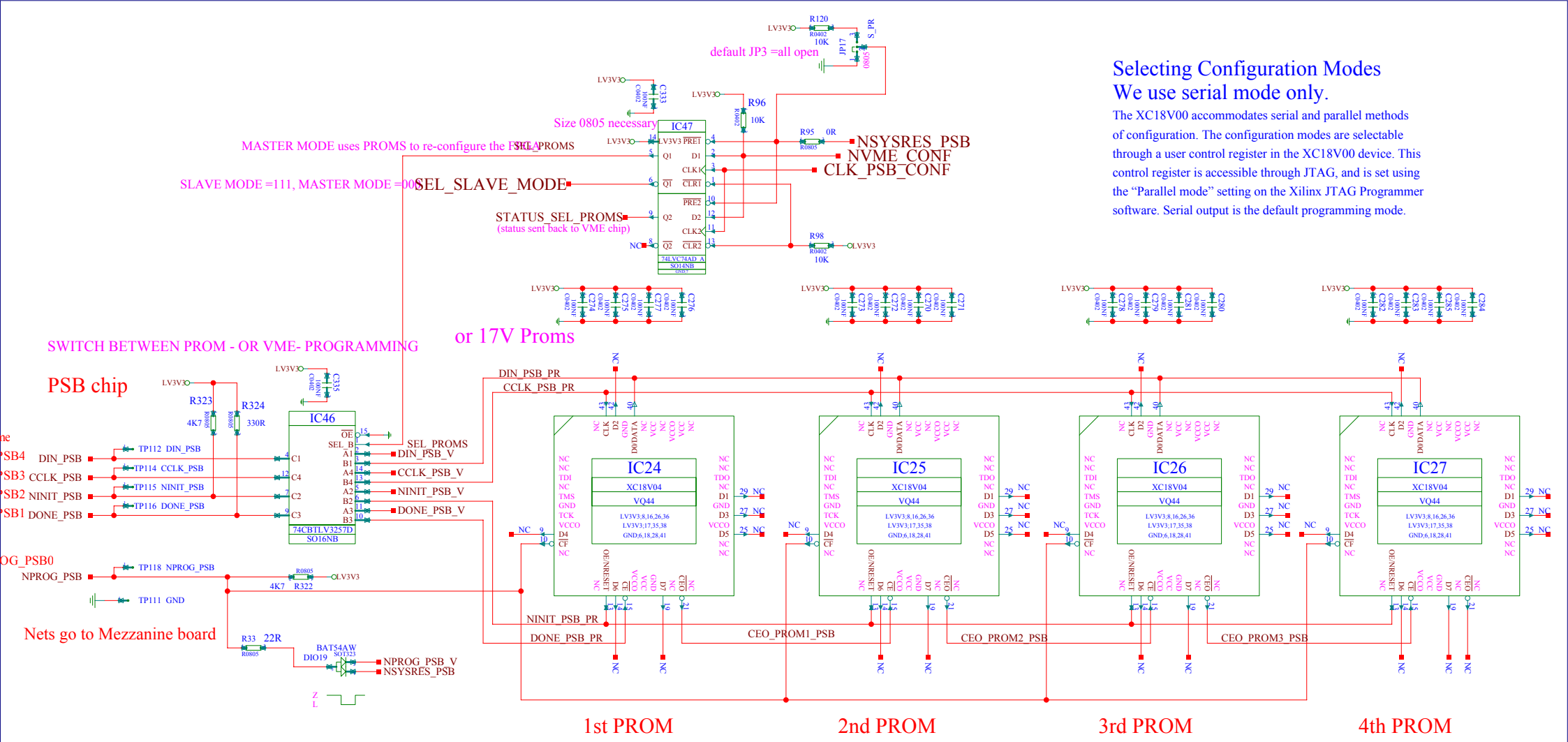

SET_RUNNING,INACTIVE,RUNNING,VME_LED,LOCKED_LED
Front Panel
3 dedicated pins: PROG_B, DONE, CCLK
+2 io: ninit, din
CLK_PSB_CONF
IC20-23
CONF_PSB
ENTTIN_[15:0]
SERLINK2_MODE[7:0]
SERLINK[1:0]_MODE[15:0]
NLOCKED[7:0]
reconfig PSB fr Proms
Configuration
INACTIVE LOCKED
ERROR VME
FRONT_PAN_PSB
PROG_DOUT
TP16
5 nets
from PSB
Virtex2
VME Access
Power ON/OFF
JTAG Configuration of Altera chips
JTAG Configuration of Xilinx chips
JTAG_JP_OFF
V_SEL_BACKPL
V_SEL_CABLES
9+19+16 = 42
VME_CHIP_PSB: New Pin Assignment to be done!
VME_CHIP_PSB: Are all 2.5V and 3.3V used?
XC2V2000-4BF957
on
MEZZ957
Global
POWER NETS
VCC, LV3V3,...
VME_IO_PSB
16 nets from VME
ENTTIN_[15:0]
LVDS_REC_PSB
diff nets
only to get simple net names
56 nets to/from VME
16+28+1=45
LV3V3BIAS used here
RO_CHLINK_PSB
16x4=64 nets
TTIN[15:0]_[3:0]
CLK_DAQ = inverted CLK
32x4 +12x2 =152
OUTPUT CHANNELS as 80 MHz parallel GTLp signals
BACK9U-slot13 copied from MIP/ISO pins added
CH3_[31:0]
CH2_[31:0]
CH1_[31:0]
CH0_[31:0]
CH1X_[27:16]
CH0X_[27:16]
PSB_STATUS[3:0]
STATUS = 4
5+ FBIN+FBOUT+TEST_CLK =9
CLOCK_PSB
REFCLK[7:0]
LVDS signals
LVTTL
GTL.p
ZPACK_PSB
IC0...7
DS92LV16_CHIPS
4x16 =64nets
8x16 =128 nets
TR[3:0]_[15:00]
REC[7:0]_[15:00]
REFCLK[7:0]
PSB_CHIP
TDO_PSB
TDI_PSB
TMS_PSB
TCK_PSB
JTAG=dedicated pins
to/from VME chip
JTAG_PSB
VCCBIAS used here
561 IO signals
Mezzanine: 599 io - nnVREF - 20 VRP/VRN - nn CLOCK
PSB CHIP: Banks 1-5 : 70+82+82+68+68 -12VRN/VRP = 358 io pins
PSB CHIP: 5+256+9+4+45+42= 361 ..zu viele
PSB CHIP: Banks 0,6,7 : 70+82+82= 234 io pins
CH0..3 (GTLP) 152 + 21 VREF + 6 VRN/VRP < 234...ok
8+16+64+128+64+9+4+47+43 =383
Design taken from FDL board and then changed.
PSB-BOARD-9U
PSB9U
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 1
modified by: A.T 11-17-2004_9:10
checked by: A.TAUROK 11-3-2004_12:26
GTLP_DCI and LVTTL must not be assigned in same BANK!!
GCT sends via Infiniband Cables 1.4 Gbps serial data to the PSB9U board.
A DS92LV16 chip sends data to PSB chip: 16 bits parallel at 80 MHz as LVTTL signals.
TOTEM and Technical Triggers send data as LVDS signals 4 bit parallel at 40 MHz
+1.5V
POWER_PSB
CHANNEL ORDER on PSB9U
CALO SLOTS
13 14 15
CA1 CA5 xx
CA2 CA6 xx
CA3 CA7 CA9
CA4 CA8 CA10
MIP/ISO SLOTS
19 20 21
MQ4 MQ8 MQ12
MQ3 MQ7 MQ11
MQ2 MQ6 MQ10
MQ1 MQ5 MQ9
PSB board
CH3
CH2
CH1 +CH1X
CH0 +CH0X



PSB-CARD-9U

CLOCK_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: AT 7.NOV03	11-12-2004_16:27
checked by: A.TAUROK	110504



Selecting Configuration Modes
We use serial mode only.

The XC18V00 accommodates serial and parallel methods of configuration. The configuration modes are selectable through a user control register in the XC18V00 device. This control register is accessible through JTAG, and is set using the “Parallel mode” setting on the Xilinx JTAG Programmer software. Serial output is the default programming mode.

Achtung: REFDES in CONF_PSB und JTAG mnssen nbereinstimmen!!!

JTAG pins of PROMs : See JTAG circuits

See PSB_CHIP schematic to find:
HSWAP_EN_PSB
NPWRDWN_B
DOUT_PSB
On Mezzanine board for PSB you find: M0,M1,M2

CLK: IN

/CF: OUT: OPEN DRAIN
/CF: Allows JTAG CONFIG instruction to initiate FPGA configuration without powering down FPGA. This is an open-drain output that is pulsed Low by the JTAG CONFIG command.

OE/RESET: BIDIR and OPEN DRAIN
OE/RESET: When Low, this input holds the address counter reset and the DATA output is in a Low while the PROM is reset. Polarity is NOT programmable.
high-impedance state. This is a bidirectional open-drain pin that is held

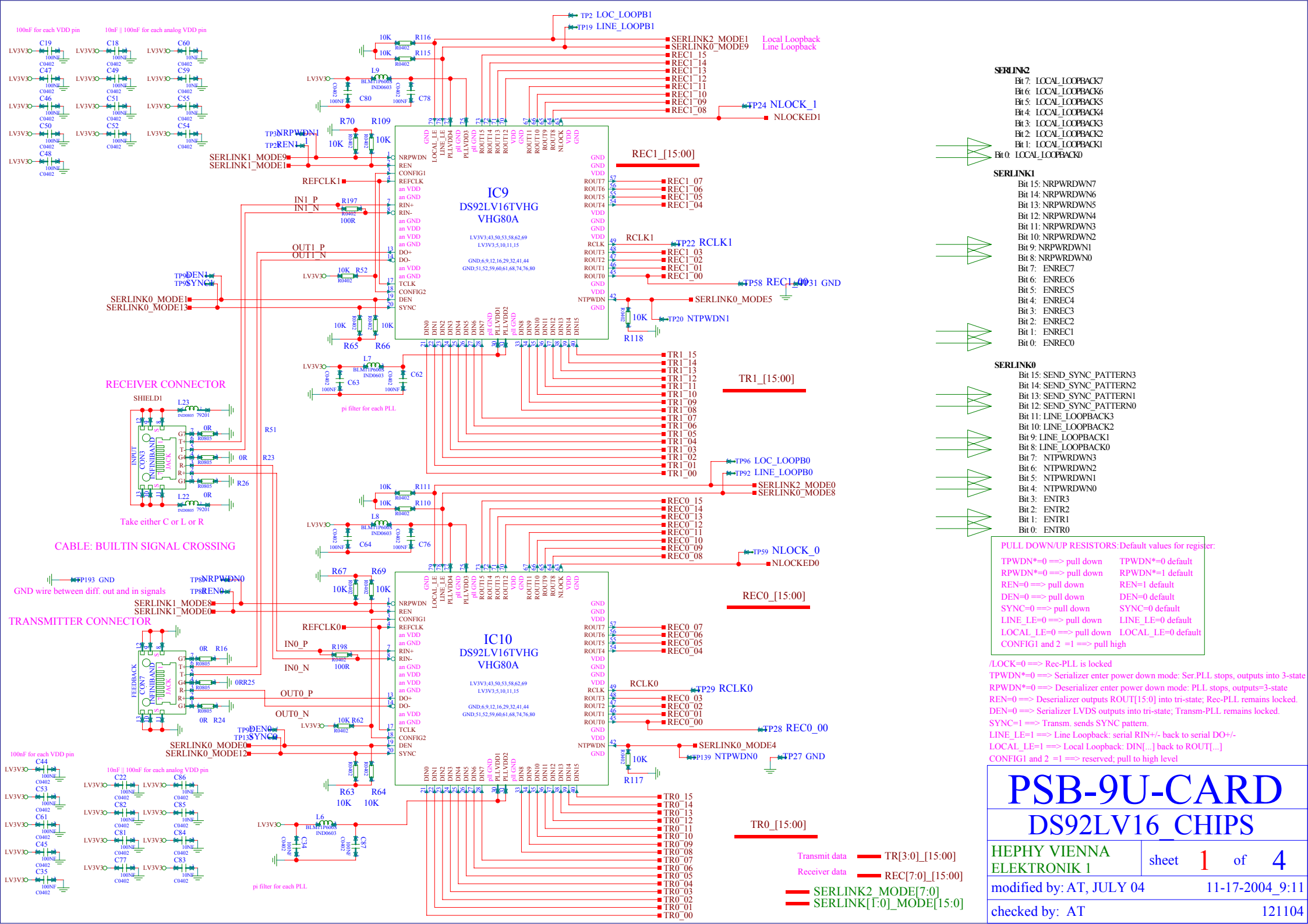
/CE: IN
/CE: When CE is High, the device is put into low-power standby mode, the address counter is reset, and the DATA pins are put in a high-impedance state.

/CEO: OUT
/CEO Chip Enable Output (CEO) is connected to the CE input of the next PROM in the chain. This output is Low when CE is Low and OE/RESET input is High, AND the internal address counter has been incremented beyond its Terminal Count (TC) value. CEO returns to High when OE/RESET goes Low or CE goes High.

PSB-CARD-9U

CONF_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: A.TAUROK	11-11-2004_18:49
checked by: A.TAUROK	11-11-2004_18:47



SERLINK2

Bit 7: LOCAL_LOOPBACK7
Bit 6: LOCAL_LOOPBACK6
Bit 5: LOCAL_LOOPBACK5
Bit 4: LOCAL_LOOPBACK4
Bit 3: LOCAL_LOOPBACK3
Bit 2: LOCAL_LOOPBACK2
Bit 1: LOCAL_LOOPBACK1
Bit 0: LOCAL_LOOPBACK0

SERLINK1

Bit 15: NRPWRDWN7
Bit 14: NRPWRDWN6
Bit 13: NRPWRDWN5
Bit 12: NRPWRDWN4
Bit 11: NRPWRDWN3
Bit 10: NRPWRDWN2
Bit 9: NRPWRDWN1
Bit 8: NRPWRDWN0
Bit 7: ENREC7
Bit 6: ENREC6
Bit 5: ENREC5
Bit 4: ENREC4
Bit 3: ENREC3
Bit 2: ENREC2
Bit 1: ENREC1
Bit 0: ENREC0

SERLINK0

Bit 15: SEND_SYNC_PATTERN3
Bit 14: SEND_SYNC_PATTERN2
Bit 13: SEND_SYNC_PATTERN1
Bit 12: SEND_SYNC_PATTERN0
Bit 11: LINE_LOOPBACK3
Bit 10: LINE_LOOPBACK2
Bit 9: LINE_LOOPBACK1
Bit 8: LINE_LOOPBACK0
Bit 7: NTPWRDWN3
Bit 6: NTPWRDWN2
Bit 5: NTPWRDWN1
Bit 4: NTPWRDWN0
Bit 3: ENTR3
Bit 2: ENTR2
Bit 1: ENTR1
Bit 0: ENTR0

PULL DOWN/UP RESISTORS: Default values for register:

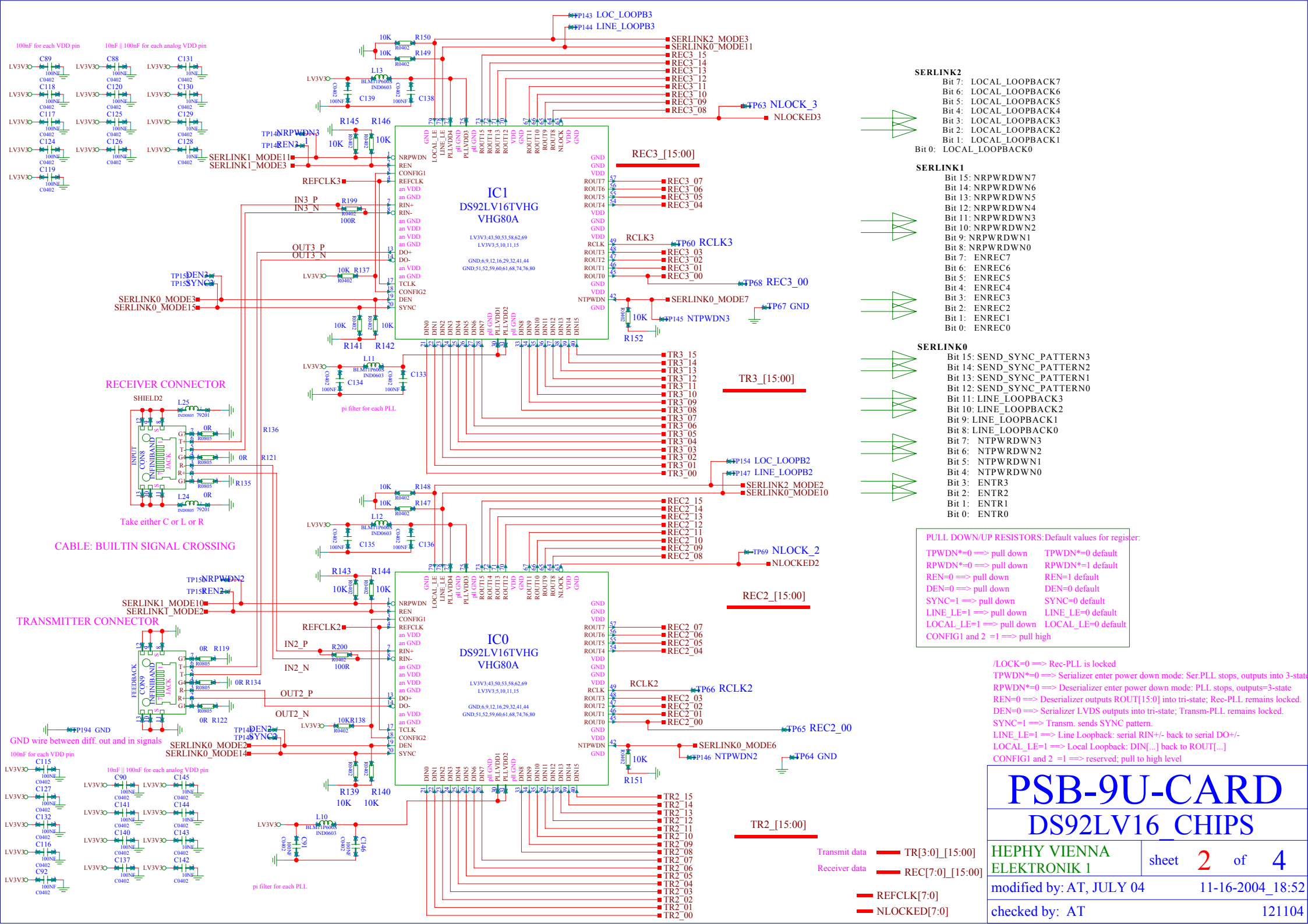
TPWDN*=0 ==> pull down	TPWDN*=0 default
RPWDN*=0 ==> pull down	RPWDN*=1 default
REN=0 ==> pull down	REN=1 default
DEN=0 ==> pull down	DEN=0 default
SYNC=0 ==> pull down	SYNC=0 default
LINE_LE=0 ==> pull down	LINE_LE=0 default
LOCAL_LE=0 ==> pull down	LOCAL_LE=0 default
CONFIG1 and 2 =1 ==> pull high	

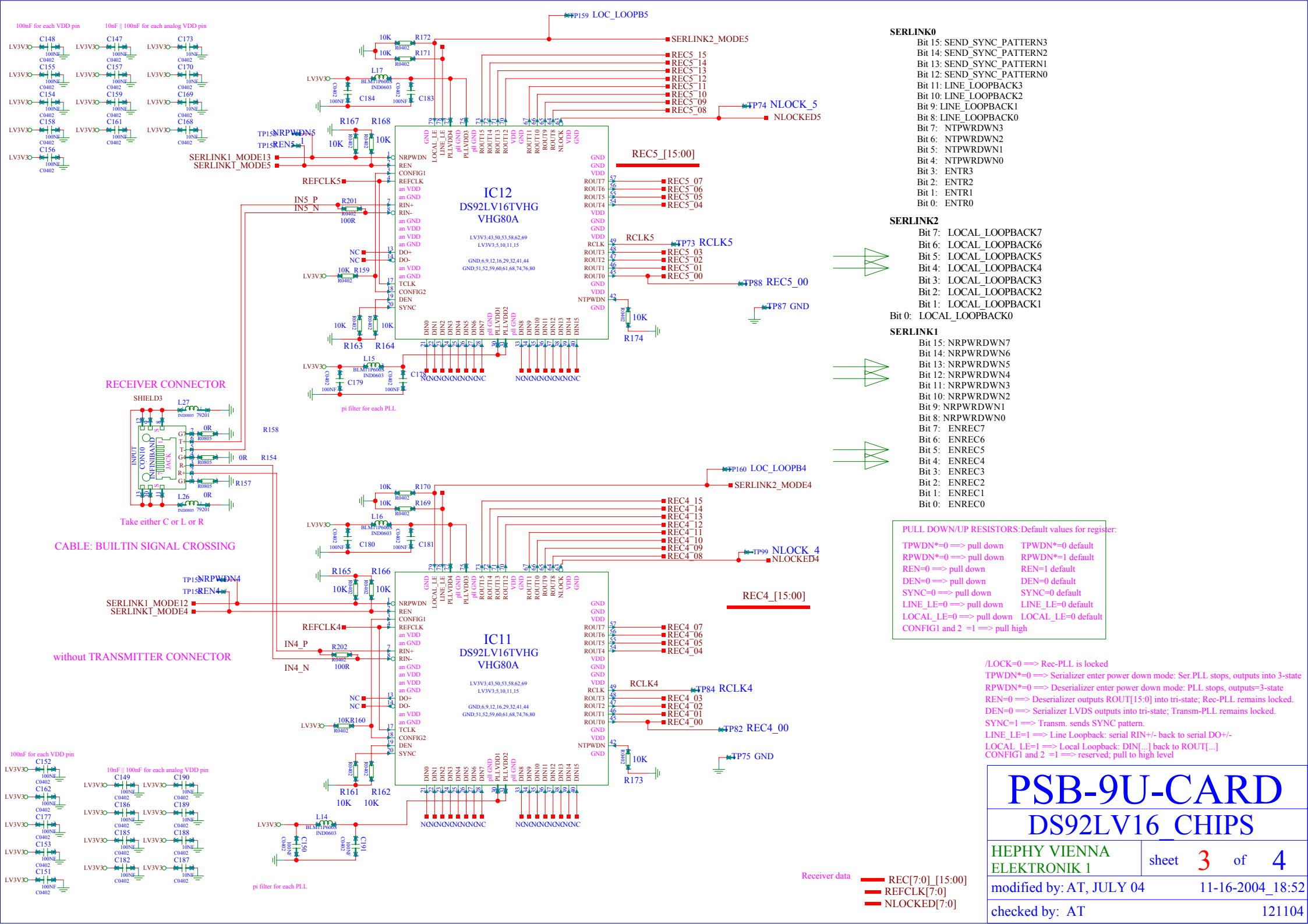
/LOCK=0 ==> Rec-PLL is locked
TPWDN*=0 ==> Serializer enter power down mode: Ser.PLL stops, outputs into 3-state
RPWDN*=0 ==> Deserializer enter power down mode: PLL stops, outputs=3-state
REN=0 ==> Deserializer outputs ROUT[15:0] into tri-state; Rec-PLL remains locked.
DEN=0 ==> Serializer LVDS outputs into tri-state; Transm-PLL remains locked.
SYNC=1 ==> Transm. sends SYNC pattern.
LINE_LE=1 ==> Line Loopback: serial RIN+/- back to serial DO+/-
LOCAL_LE=1 ==> Local Loopback: DIN[...] back to ROUT[...]
CONFIG1 and 2 =1 ==> reserved; pull to high level

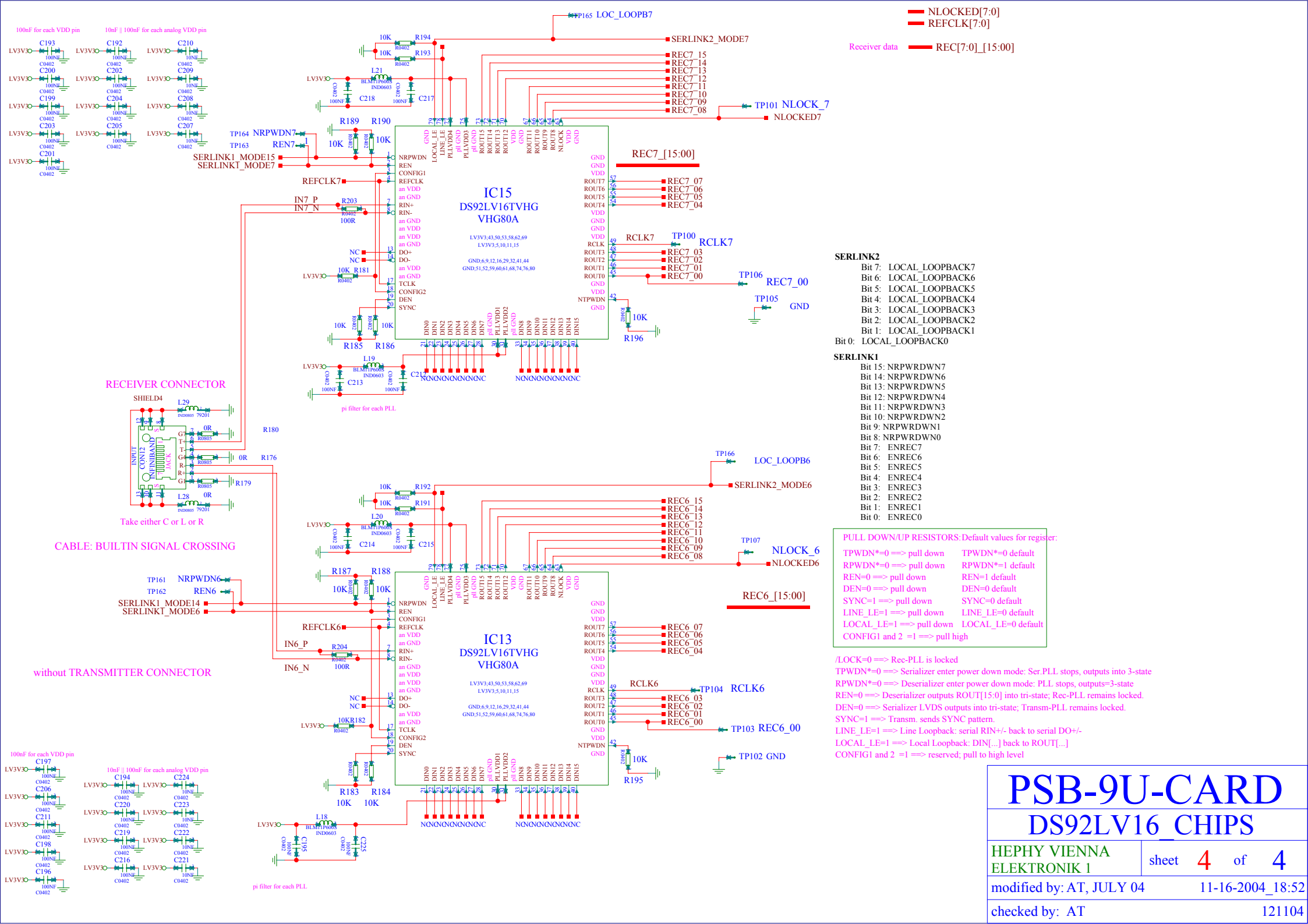
PSB-9U-CARD

DS92LV16_CHIPS

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 4
modified by: AT, JULY 04	11-17-2004_9:11
checked by: AT	121104


100nF for each VDD pin
10nF || 100nF for each analog VDD pin
IC1
DS92LV16TVHG
VHG80A
IC0
DS92LV16TVHG
VHG80A
REC3_[15:00]
TR3_[15:00]
REC2_[15:00]
TR2_[15:00]
RECEIVER CONNECTOR
TRANSMITTER CONNECTOR
CABLE: BUILTIN SIGNAL CROSSING
Take either C or L or R
GND wire between diff. out and in signals
pi filter for each PLL
SERLINK2
Bit 7: LOCAL_LOOPBACK7
Bit 6: LOCAL_LOOPBACK6
Bit 5: LOCAL_LOOPBACK5
Bit 4: LOCAL_LOOPBACK4
Bit 3: LOCAL_LOOPBACK3
Bit 2: LOCAL_LOOPBACK2
Bit 1: LOCAL_LOOPBACK1
Bit 0: LOCAL_LOOPBACK0
SERLINK1
Bit 15: NRPWRDWN7
Bit 14: NRPWRDWN6
Bit 13: NRPWRDWN5
Bit 12: NRPWRDWN4
Bit 11: NRPWRDWN3
Bit 10: NRPWRDWN2
Bit 9: NRPWRDWN1
Bit 8: NRPWRDWN0
Bit 7: ENREC7
Bit 6: ENREC6
Bit 5: ENREC5
Bit 4: ENREC4
Bit 3: ENREC3
Bit 2: ENREC2
Bit 1: ENREC1
Bit 0: ENREC0
SERLINK0
Bit 15: SEND_SYNC_PATTERN3
Bit 14: SEND_SYNC_PATTERN2
Bit 13: SEND_SYNC_PATTERN1
Bit 12: SEND_SYNC_PATTERN0
Bit 11: LINE_LOOPBACK3
Bit 10: LINE_LOOPBACK2
Bit 9: LINE_LOOPBACK1
Bit 8: LINE_LOOPBACK0
Bit 7: NTPWRDWN3
Bit 6: NTPWRDWN2
Bit 5: NTPWRDWN1
Bit 4: NTPWRDWN0
Bit 3: ENTR3
Bit 2: ENTR2
Bit 1: ENTR1
Bit 0: ENTR0
PULL DOWN/UP RESISTORS: Default values for register:
TPWDN*=0 ==> pull down
TPWDN*=0 default
RPWDN*=0 ==> pull down
RPWDN*=1 default
REN=0 ==> pull down
REN=1 default
DEN=0 ==> pull down
DEN=0 default
SYNC=1 ==> pull down
SYNC=0 default
LINE_LE=1 ==> pull down
LINE_LE=0 default
LOCAL_LE=1 ==> pull down
LOCAL_LE=0 default
CONFIG1 and 2 =1 ==> pull high
/LOCK=0 ==> Rec-PLL is locked
TPWDN*=0 ==> Serializer enter power down mode: Ser.PLL stops, outputs into 3-state
RPWDN*=0 ==> Deserializer enter power down mode: PLL stops, outputs=3-state
REN=0 ==> Deserializer outputs ROUT[15:0] into tri-state; Rec-PLL remains locked.
DEN=0 ==> Serializer LVDS outputs into tri-state; Transm-PLL remains locked.
SYNC=1 ==> Transm. sends SYNC pattern.
LINE_LE=1 ==> Line Loopback: serial RIN+/- back to serial DO+/-
LOCAL_LE=1 ==> Local Loopback: DIN[...] back to ROUT[...]
CONFIG1 and 2 =1 ==> reserved; pull to high level
Transmit data
Receiver data
TR[3:0]_[15:00]
REC[7:0]_[15:00]
REFCLK[7:0]
NLOCKED[7:0]
PSB-9U-CARD
DS92LV16_CHIPS
HEPHY VIENNA
ELEKTRONIK 1
sheet 2 of 4
modified by: AT, JULY 04
11-16-2004_18:52
checked by: AT
121104

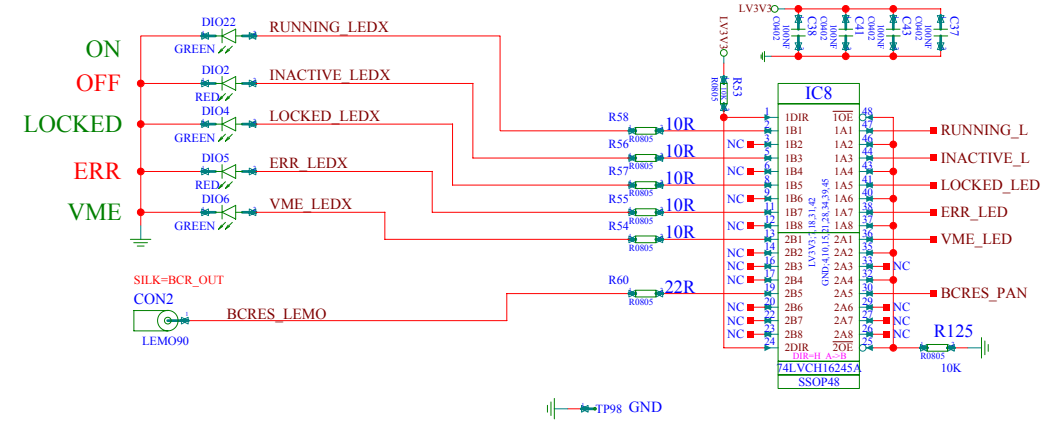

100nF for each VDD pin
10nF || 100nF for each analog VDD pin
IC12
DS92LV16TVHG
VHG80A
IC11
DS92LV16TVHG
VHG80A
REC5_[15:00]
REC4_[15:00]
RECEIVER CONNECTOR
Take either C or L or R
CABLE: BUILTIN SIGNAL CROSSING
without TRANSMITTER CONNECTOR
pi filter for each PLL
SERLINK0
Bit 15: SEND_SYNC_PATTERN3
Bit 14: SEND_SYNC_PATTERN2
Bit 13: SEND_SYNC_PATTERN1
Bit 12: SEND_SYNC_PATTERN0
Bit 11: LINE_LOOPBACK3
Bit 10: LINE_LOOPBACK2
Bit 9: LINE_LOOPBACK1
Bit 8: LINE_LOOPBACK0
Bit 7: NTPWRDWN3
Bit 6: NTPWRDWN2
Bit 5: NTPWRDWN1
Bit 4: NTPWRDWN0
Bit 3: ENTR3
Bit 2: ENTR2
Bit 1: ENTR1
Bit 0: ENTR0
SERLINK2
Bit 7: LOCAL_LOOPBACK7
Bit 6: LOCAL_LOOPBACK6
Bit 5: LOCAL_LOOPBACK5
Bit 4: LOCAL_LOOPBACK4
Bit 3: LOCAL_LOOPBACK3
Bit 2: LOCAL_LOOPBACK2
Bit 1: LOCAL_LOOPBACK1
Bit 0: LOCAL_LOOPBACK0
SERLINK1
Bit 15: NRPWRDWN7
Bit 14: NRPWRDWN6
Bit 13: NRPWRDWN5
Bit 12: NRPWRDWN4
Bit 11: NRPWRDWN3
Bit 10: NRPWRDWN2
Bit 9: NRPWRDWN1
Bit 8: NRPWRDWN0
Bit 7: ENREC7
Bit 6: ENREC6
Bit 5: ENREC5
Bit 4: ENREC4
Bit 3: ENREC3
Bit 2: ENREC2
Bit 1: ENREC1
Bit 0: ENREC0
PULL DOWN/UP RESISTORS:Default values for register:
TPWDN*=0 ==> pull down
TPWDN*=0 default
RPWDN*=0 ==> pull down
RPWDN*=1 default
REN=0 ==> pull down
REN=1 default
DEN=0 ==> pull down
DEN=0 default
SYNC=0 ==> pull down
SYNC=0 default
LINE_LE=0 ==> pull down
LINE_LE=0 default
LOCAL_LE=0 ==> pull down
LOCAL_LE=0 default
CONFIG1 and 2 =1 ==> pull high
/LOCK=0 ==> Rec-PLL is locked
TPWDN*=0 ==> Serializer enter power down mode: Ser.PLL stops, outputs into 3-state
RPWDN*=0 ==> Deserializer enter power down mode: PLL stops, outputs=3-state
REN=0 ==> Deserializer outputs ROUT[15:0] into tri-state; Rec-PLL remains locked.
DEN=0 ==> Serializer LVDS outputs into tri-state; Transm-PLL remains locked.
SYNC=1 ==> Transm. sends SYNC pattern.
LINE_LE=1 ==> Line Loopback: serial RIN+/- back to serial DO+/-
LOCAL_LE=1 ==> Local Loopback: DIN[...] back to ROUT[...]
CONFIG1 and 2 =1 ==> reserved; pull to high level
Receiver data
REC[7:0]_[15:00]
REFCLK[7:0]
NLOCKED[7:0]
PSB-9U-CARD
DS92LV16_CHIPS
HEPHY VIENNA
ELEKTRONIK 1
sheet 3 of 4
modified by: AT, JULY 04
11-16-2004_18:52
checked by: AT
121104


NLOCKED[7:0]
REFCLK[7:0]
Receiver data
REC[7:0]_[15:00]
IC15
DS92LV16TVHG
VHG80A
IC13
DS92LV16TVHG
VHG80A
REC7_[15:00]
REC6_[15:00]
RECEIVER CONNECTOR
Take either C or L or R
CABLE: BUILTIN SIGNAL CROSSING
without TRANSMITTER CONNECTOR
pi filter for each PLL
SERLINK2_MODE7
SERLINK2_MODE6
LOC_LOOPB7
LOC_LOOPB6
NLOCK_7
NLOCK_6
RCLK7
RCLK6
REC7_00
REC6_00
SERLINK1_MODE15
SERLINKT_MODE7
SERLINK1_MODE14
SERLINKT_MODE6
REFCLK7
REFCLK6
IN7_P
IN7_N
IN6_P
IN6_N
NRPWDN7
REN7
NRPWDN6
REN6
SERLINK2
Bit 7: LOCAL_LOOPBACK7
Bit 6: LOCAL_LOOPBACK6
Bit 5: LOCAL_LOOPBACK5
Bit 4: LOCAL_LOOPBACK4
Bit 3: LOCAL_LOOPBACK3
Bit 2: LOCAL_LOOPBACK2
Bit 1: LOCAL_LOOPBACK1
Bit 0: LOCAL_LOOPBACK0
SERLINK1
Bit 15: NRPWRDWN7
Bit 14: NRPWRDWN6
Bit 13: NRPWRDWN5
Bit 12: NRPWRDWN4
Bit 11: NRPWRDWN3
Bit 10: NRPWRDWN2
Bit 9: NRPWRDWN1
Bit 8: NRPWRDWN0
Bit 7: ENREC7
Bit 6: ENREC6
Bit 5: ENREC5
Bit 4: ENREC4
Bit 3: ENREC3
Bit 2: ENREC2
Bit 1: ENREC1
Bit 0: ENREC0
PULL DOWN/UP RESISTORS: Default values for register:
TPWDN*=0 ==> pull down
RPWDN*=0 ==> pull down
REN=0 ==> pull down
DEN=0 ==> pull down
SYNC=1 ==> pull down
LINE_LE=1 ==> pull down
LOCAL_LE=1 ==> pull down
CONFIG1 and 2 =1 ==> pull high
TPWDN*=0 default
RPWDN*=1 default
REN=1 default
DEN=0 default
SYNC=0 default
LINE_LE=0 default
LOCAL_LE=0 default
/LOCK=0 ==> Rec-PLL is locked
TPWDN*=0 ==> Serializer enter power down mode: Ser.PLL stops, outputs into 3-state
RPWDN*=0 ==> Deserializer enter power down mode: PLL stops, outputs=3-state
REN=0 ==> Deserializer outputs ROUT[15:0] into tri-state; Rec-PLL remains locked.
DEN=0 ==> Serializer LVDS outputs into tri-state; Transm-PLL remains locked.
SYNC=1 ==> Transm. sends SYNC pattern.
LINE_LE=1 ==> Line Loopback: serial RIN+/- back to serial DO+/-
LOCAL_LE=1 ==> Local Loopback: DIN[...] back to ROUT[...]
CONFIG1 and 2 =1 ==> reserved; pull to high level
PSB-9U-CARD
DS92LV16_CHIPS
HEPHY VIENNA
ELEKTRONIK 1
sheet 4 of 4
modified by: AT, JULY 04
11-16-2004_18:52
checked by: AT
121104

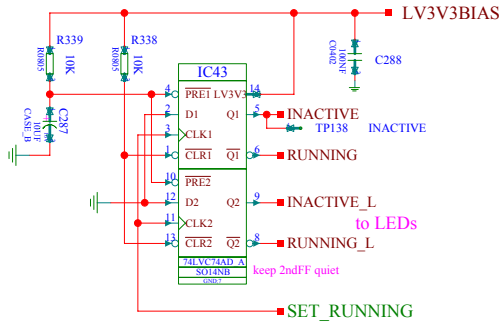
DISPLAY

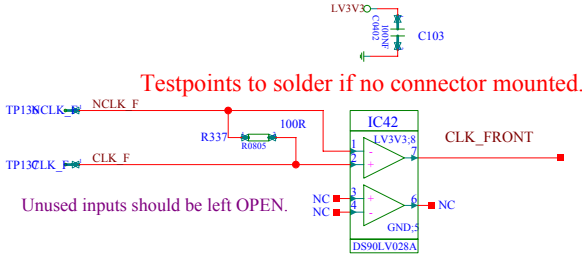
leds mounted above all connectors on frontpanel

Front Panel Text



INTERLOCK





Frontpanel

DR1 DRILL 2P8 DR2 DRILL 2P8 DR3 DRILL 2P8

Rail

X3 DRILL 2P8 X2 DRILL 2P8

LEDS

Front Panel cables

DIFF_CONN: CLK //in for test

LEMO: BCRES //out

4x4 RJ45-8

REC cable 3 TRANS cable 3
REC cable 2 TRANS cable 2
REC cable 1
REC cable 0

Differential Connector: JMP2 to solder 2 pins with 100mil distance

RS: Binder Subminiatur-Rundsteckverbinder d= 10mm der Serie 711 ; 719: 8mm; 712: 12mm;
RS: Subminiatur-Rundsteckverbinder d= 12mm der Serie 710 haben eine Bajonett-Verriegelung
TRIAD01 Serie von Thomas&Betts bei RS d=15mm bei 3 pins
MMCX Mikro-Miniatur-Koax-Steckverbinder 3-5 Euro bei RS, für RG178 cable;
DOPPEL LEMO COAX: EPY.00.250.NTN 14.3 hoch passt in VME, 8 mm breit; RS: 28 Euro

4x4 RJ45-8 Connectors ==> 24 cm
4 + 2 INFINIBAND CONNECTORS ==> 6x1=6 cm
Front Panel space for 9U: >300 mm

PSB-9U-CARD

FRONT_PAN_PSB

HEPHY VIENNA
ELEKTRONIK 1

sheet 1 of 1

modified by: A.TAUROK 11-15-2004 13:21

checked by: AT 11-12-2004 17:24

VREF is adjustable for other future download device
Set VREF =3.3V for Parallel CableIV

JTAG-chain-selection:

JTAG-cables-jumper = IN => MasterBlaster and Par.-cable IV selected
JTAG-cables-jumper = OUTand VME-chip not ready => Backplane selected
JTAG-cables-jumper = OUTand VME-chip ready => selection by VME

to/from sheet 2 (JTAG-chain for XILINX)

to/from sheet 2 (JTAG-chain for ALTERA)

to/from VME-chip (VME-JTAG for XILINX)

to/from VME-chip (VME-JTAG for ALTERA)

IC34 74LVC157AD SO16NB — SELECT_B, NSEL_CABLES
TMS_XIL, TCK_XIL, TDO_XIL, NC
TMS_PC, TMS_VME_XIL_BP_2, TCK_PC, TCK_VME_XIL_BP_2, TDO_PC, TDO_VME_XIL_BP_2

IC33 74LVC157AD SO16NB — SELECT_B, NSEL_BP
tms.., tck.., tdo, NC

IC44 74LVC157AD SO16NB — SELECT_B, NSEL_BP
tms, tck, tdo, NC

IC35 74LVC157AD SO16NB — SELECT_B, NSEL_CABLES
TMS_ALT, TCK_ALT, TDO_ALT, NC
TMS_MB, TMS_VME_ALT_BP_1, TCK_MB, TCK_VME_ALT_BP_1, TDO_MB, TDO_VME_ALT_BP_1

TDI_PAR_C
TDI_BACKP_2
TDI_VME_XIL
TDI_MB
TDI_BACKP_1
TDI_VME_ALT

CON4 PAR_CABLE_IV JTAG_CON_XIL — VREF, TMS, TCK, TDO, TDI, NC, NC
SPECIAL CONNECTOR to XILINX parallel cable IV
2 4 6 8 10 12 14
1 3 5 7 9 11 13
JTAG_CON_XIL TOP VIEW

CON5 MASTERBLAST JTAG_CON — TCK, GND, TDO, VCC, TMS, VIO, NC, NC, TDI, GND
SPECIAL CONNECTOR to ALTERA MasterBlaster
2 4 6 8 10
1 3 5 7 9
JTAG_CON TOP VIEW

LV3V3 — R97 0R — C65 100NF — JP37
JP3 SEL_VIO — LV3V3 / VCC
Run MasterBlaster either with 3.3 or 5V
JP1 VIO

IC45 — JTAG_JP_OFF (into VME chip), SILK=IN=>SEL_CABLE_JTAG, JP50, V_SEL_CABLES (from VME chip), V_SEL_BACKPL (from VME chip), X12 SEL_CAB, X13 SEL_BA
NSEL_CABLES, NSEL_BP
Emergency jumpers to configure via JTAG even if VME does not work

JTAG-address
X6 S0, X7 S1, X8 S2, X9 S3, X10 S4, X11 S5

IC28 SCANPSC110F SO28MB — TDI_BACKP_2, TCK_BP_2, TMS_BP_2, TDO_BP_2, TDI_BACKP_1, TCK_BP_1, TMS_BP_1, TDO_BP_1
NTRSTB_I, TMSB_I, TCKB_I, TDIB_I, TDOB_I

IC23 74CBT3245A SO20M3 — VCCBIAS, INACTIVE, NTRSTB, TMSB, TCKB, TDIB, TDOB, NC, NC, NC
FROM/TO BACKPLANE via bus-switches

TMS_VME_XIL, TCK_VME_XIL, TDO_VME_XIL, TDI_VME_XIL
TMS_VME_ALT, TCK_VME_ALT, TDO_VME_ALT, TDI_VME_ALT

LINKS=DSX;P:\GLOBALTRIGGER\LITERATURE\XILINX\CONFIGURATION\PARALLELCABLE4\PARALLELCABLEIV_DS097.PDF
LINKS=DSA;P:\GLOBALTRIGGER\LITERATURE\ALTERA\GENERAL\DATASHEETS\DSMASTER.PDF

PSB-CARD-9U

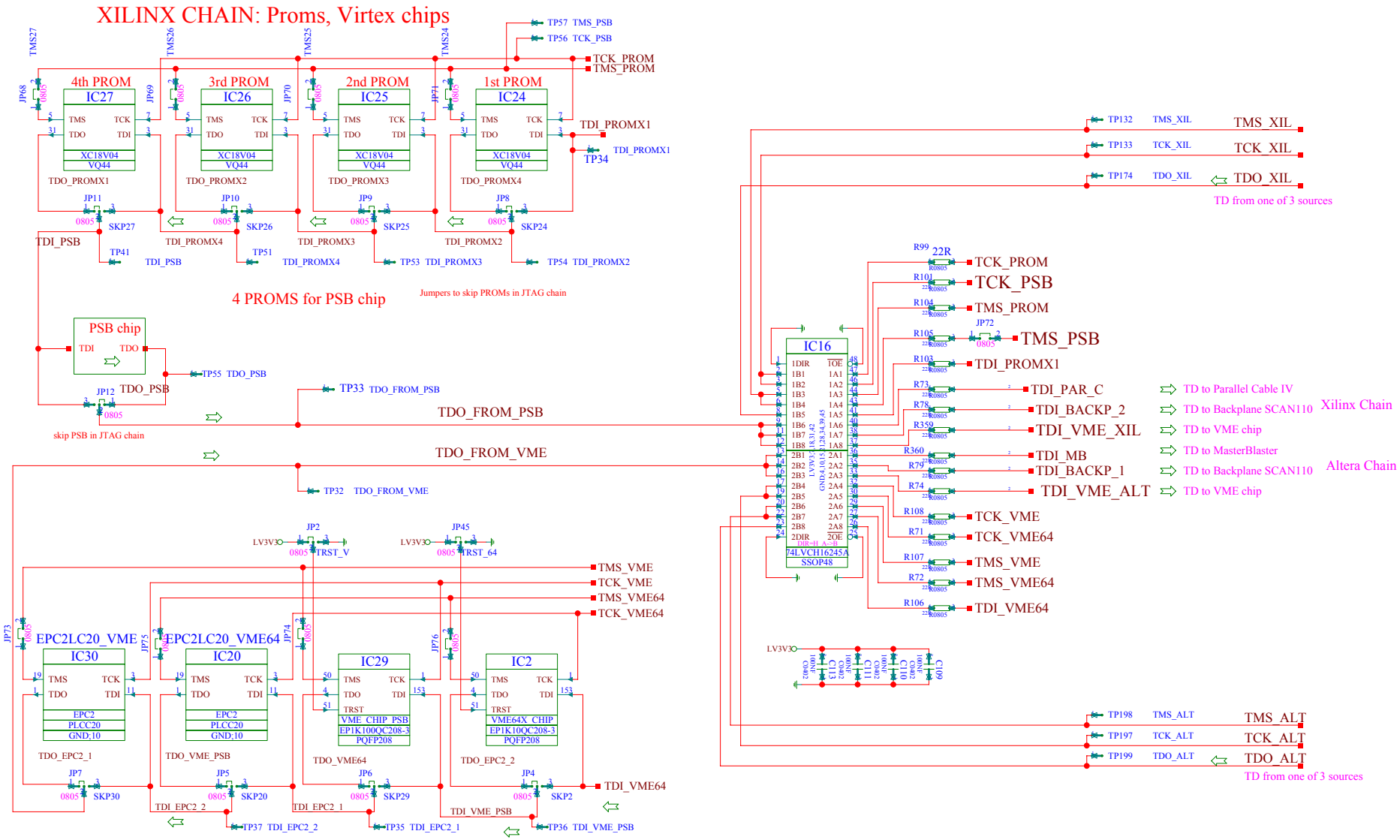
JTAG_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 2	
modified by: HB		11-17-2004_16:09
checked by: AT		211004

JTAG: Xilinx Config. options: VME, Par_CableIV, Bridge-backplane

JTAG: Altera Config. options: VME, Master Blaster, Bridge-backplane

The Virtex chips are mounted on mezzanine boards.
==>There is no JTAG hetero symbol for Virtex chips.



PSB-CARD-9U

JTAG_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 2
modified by: HB	11-11-2004_18:58
checked by: AT	111104

LVDS[3:0]_[3:0]P

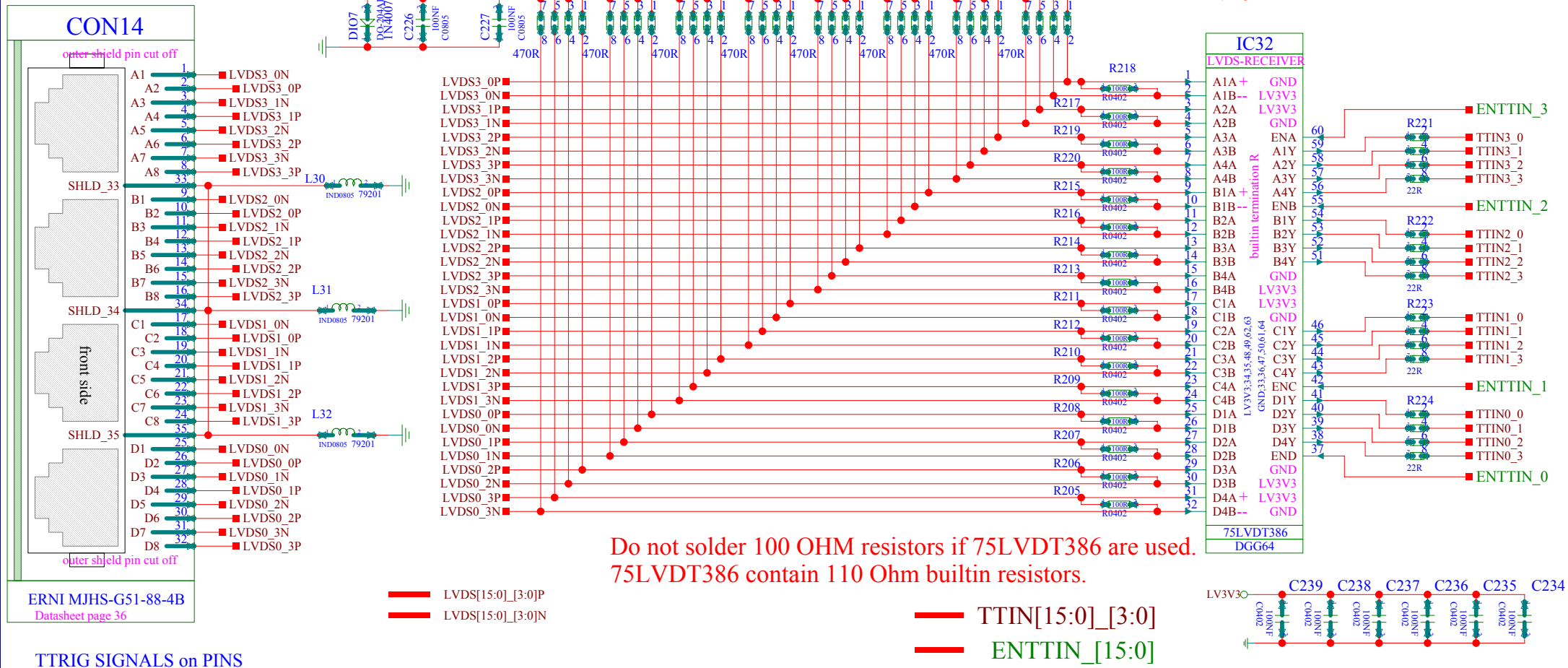
LVDS[3:0]_[3:0]N

BIAS NETWORK for LVDS

Pos. BIAS voltage about 1.8V
Neg. BIAS voltage about 0.7 V

STUBs to R as short as possible

VD< | 0.1 | <==> undefined
Open inputs ==> H
EN=0 ==> Z



Do not solder 100 OHM resistors if 75LVDT386 are used.
75LVDT386 contain 110 Ohm builtin resistors.

LVDS[15:0]_[3:0]P

LVDS[15:0]_[3:0]N

TTIN[15:0]_[3:0]

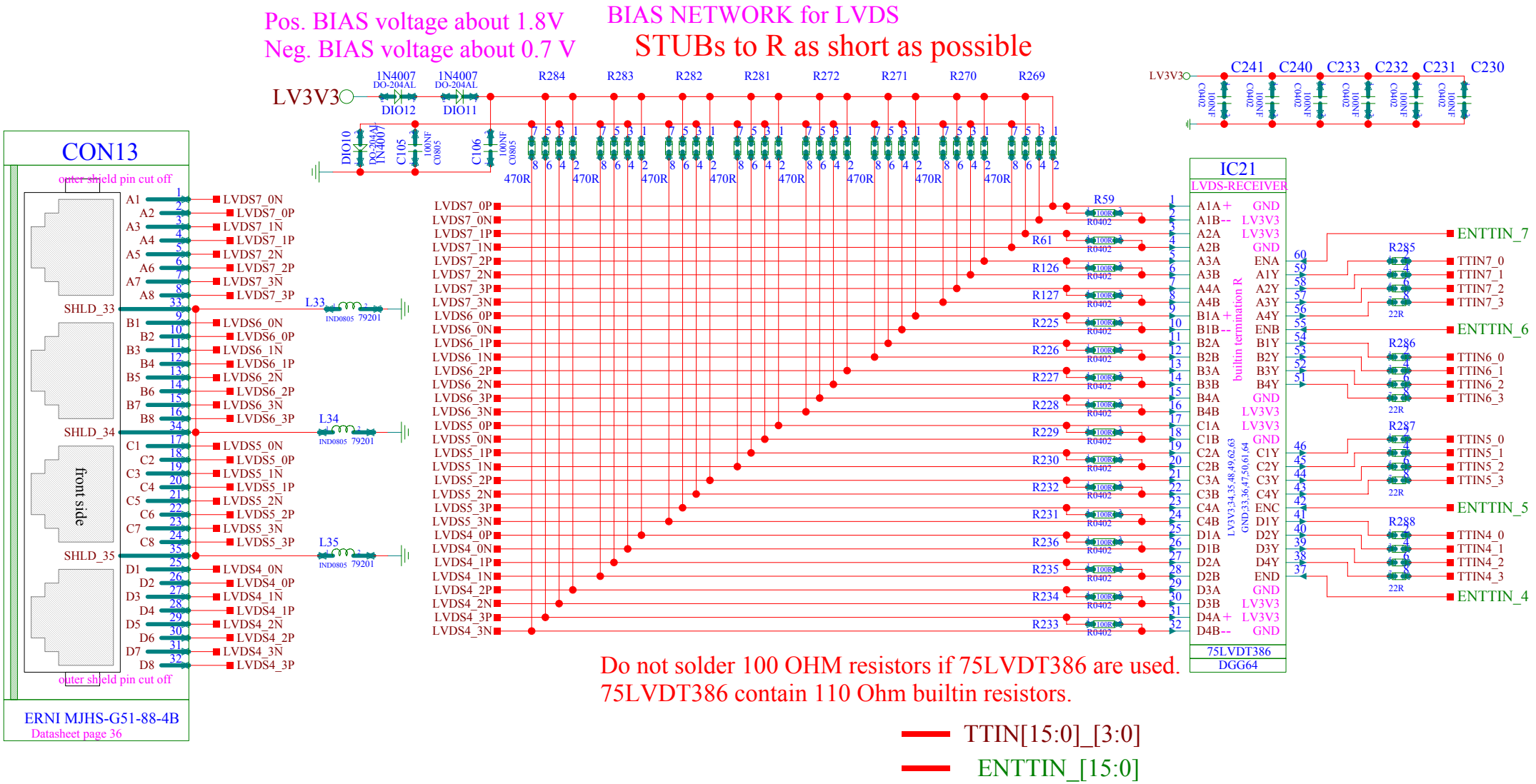
ENTTIN_[15:0]

TTRIG SIGNALS on PINS

PIN8 + BIT3
PIN7 - BIT3
PIN6 + BIT2
PIN5 - BIT2
PIN4 + BIT1
PIN3 - BIT1
PIN2 + BIT0
PIN1 - BIT0

LVDS387 drives 350mV -10dB of 20m cable (350mV+0.3162)=110 mV > 100 mV of LVDS386
Driver common mode voltages: LVDS387 = +1.3V; DS90LV047A...+1.2V
LVDS386 rec. accepts common mode voltage = +0.3...+2.3 V

PSB_BOARD-9U	
LVDS_REC_PSB	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 4
modified by: A.TAUROK	11-17-2004_14:52
checked by: CHECKER	0-00-0000_00:00

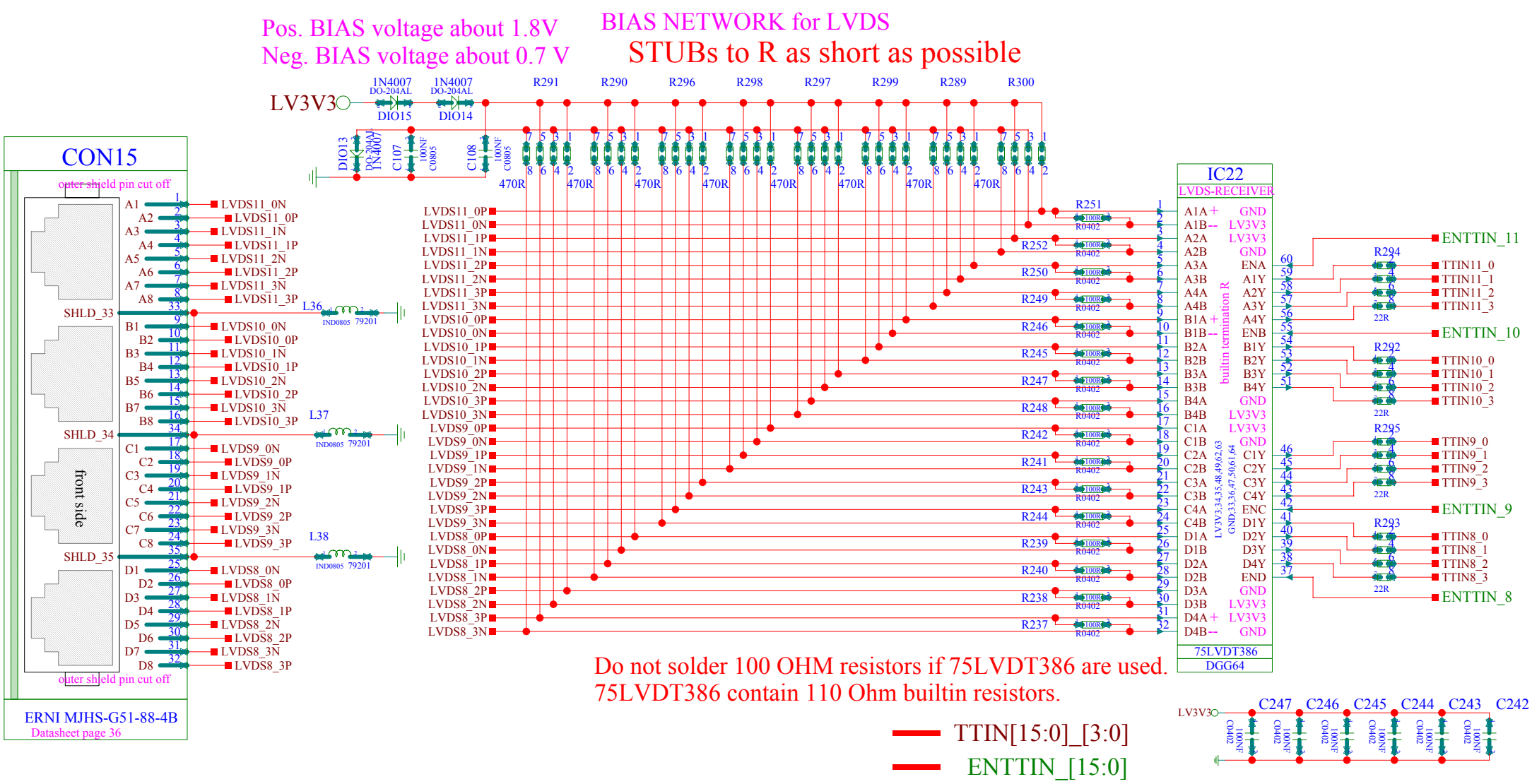


TTRIG SIGNALS on PINS

PIN8 + BIT3
PIN7 - BIT3
PIN6 + BIT2
PIN5 - BIT2
PIN4 + BIT1
PIN3 - BIT1
PIN2 + BIT0
PIN1 - BIT0

LVDS387 drives 350mV -10dB of 20m cable (350mV+0.3162)=110 mV > 100 mV of LVDS386
Driver common mode voltages: LVDS387 = +1.3V; DS90LV047A...+1.2V
LVDS386 rec. accepts common mode voltage = +0.3...+2.3 V

PSB_BOARD-9U	
LVDS_REC_PSB	
HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 4
modified by: A.TAUROK	11-12-2004_15:48
checked by: AT	121104



TTRIG SIGNALS on PINS

PIN8 + BIT3
PIN7 - BIT3
PIN6 + BIT2
PIN5 - BIT2
PIN4 + BIT1
PIN3 - BIT1
PIN2 + BIT0
PIN1 - BIT0

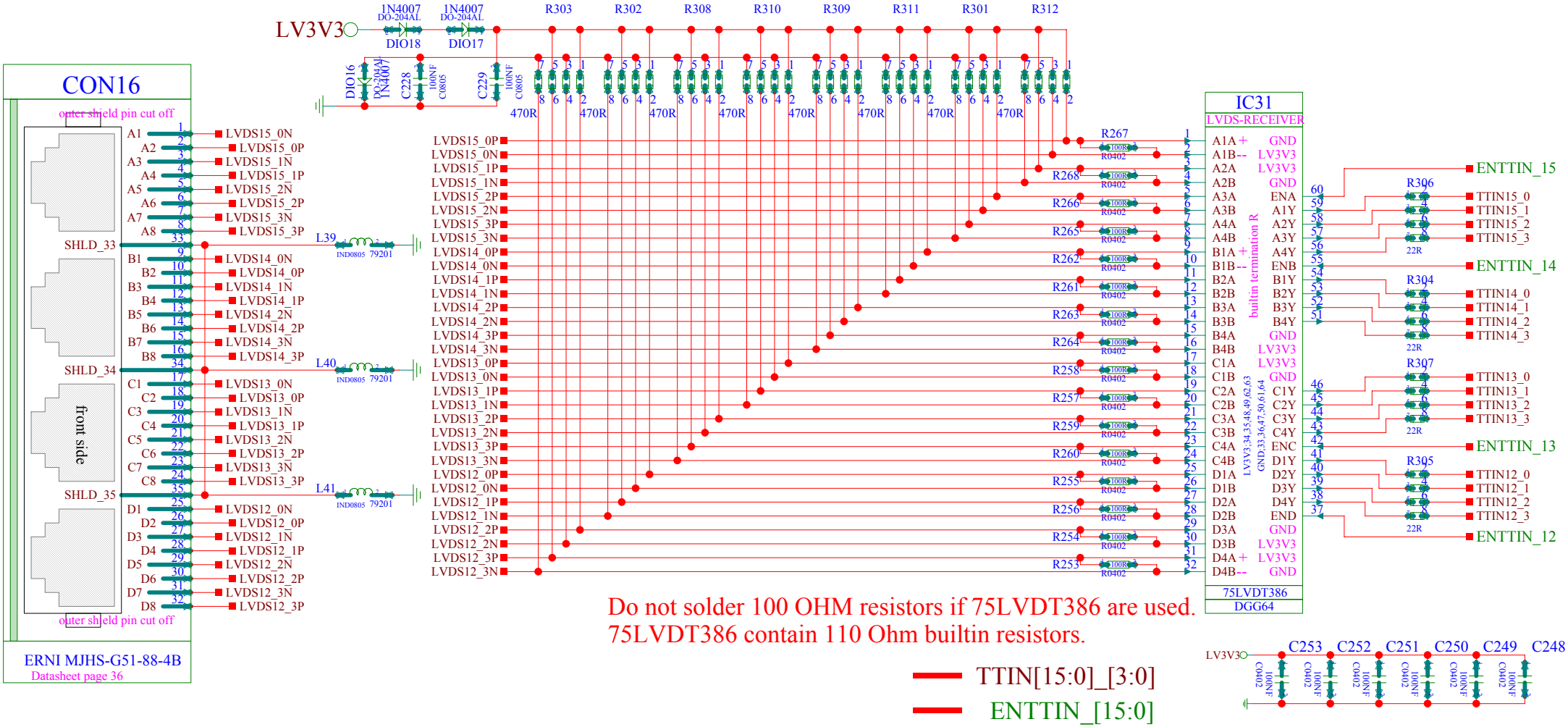
LVDS387 drives 350mV -10dB of 20m cable (350mV+0.3162)=110 mV > 100 mV of LVDS386
Driver common mode voltages: LVDS387 = +1.3V; DS90LV047A...+1.2V
LVDS386 rec. accepts common mode voltage = +0.3...+2.3 V

PSB_BOARD-9U	
LVDS_REC_PSB	
HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 4
modified by: A.TAUROK	11-12-2004_15:49
checked by: AT	121104

Pos. BIAS voltage about 1.8V
Neg. BIAS voltage about 0.7 V

BIAS NETWORK for LVDS

STUBs to R as short as possible



Do not solder 100 OHM resistors if 75LVDT386 are used.
75LVDT386 contain 110 Ohm builtin resistors.

TTIN[15:0]_[3:0]

ENTTIN_[15:0]

TTRIG SIGNALS on PINS

PIN8 + BIT3
PIN7 - BIT3
PIN6 + BIT2
PIN5 - BIT2
PIN4 + BIT1
PIN3 - BIT1
PIN2 + BIT0
PIN1 - BIT0

LVDS387 drives 350mV -10dB of 20m cable (350mV+0.3162)=110 mV > 100 mV of LVDS386
Driver common mode voltages: LVDS387 = +1.3V; DS90LV047A...+1.2V
LVDS386 rec. accepts common mode voltage = +0.3...+2.3 V

PSB_BOARD-9U

LVDS_REC_PSB

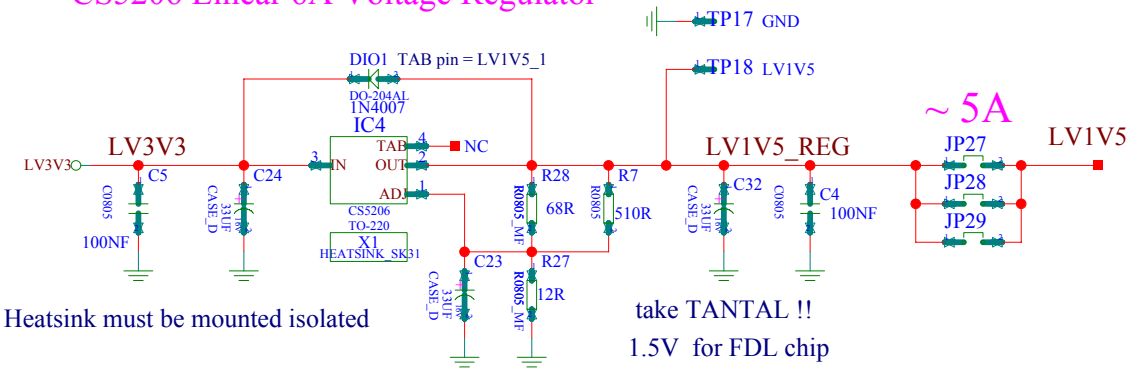
HEPHY VIENNA
ELEKTRONIK 1

sheet 4 of 4

modified by: A.TAUROK 11-12-2004_15:50

checked by: AT 121104

CS5206 Linear 6A Voltage Regulator



Heatsink must be mounted isolated

take TANTAL !!
1.5V for FDL chip

V_ref
(1.240...1.254..1.266)V/68.1 = 18.2...18.4...18.6 mA >10mA
250 mV /(18.2...18.6 mA) = 13.73...13.44 Ohm
Adjust voltage with R131 = 475....562 Ohm
I_adj=54 uA...can be neglected
all resistors for CS5206 in metalfilm

copied from FDL board

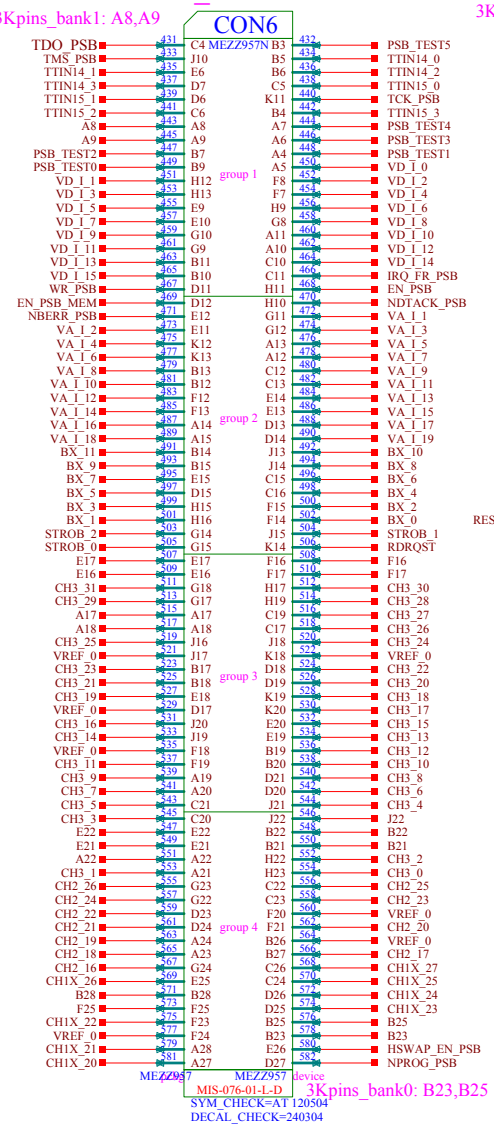
PSB-CARD-9U

POWER_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: H. BERGAUER	10-14-2004_9:55
checked by: AT+MP	110504

NORD MIS_076 connector

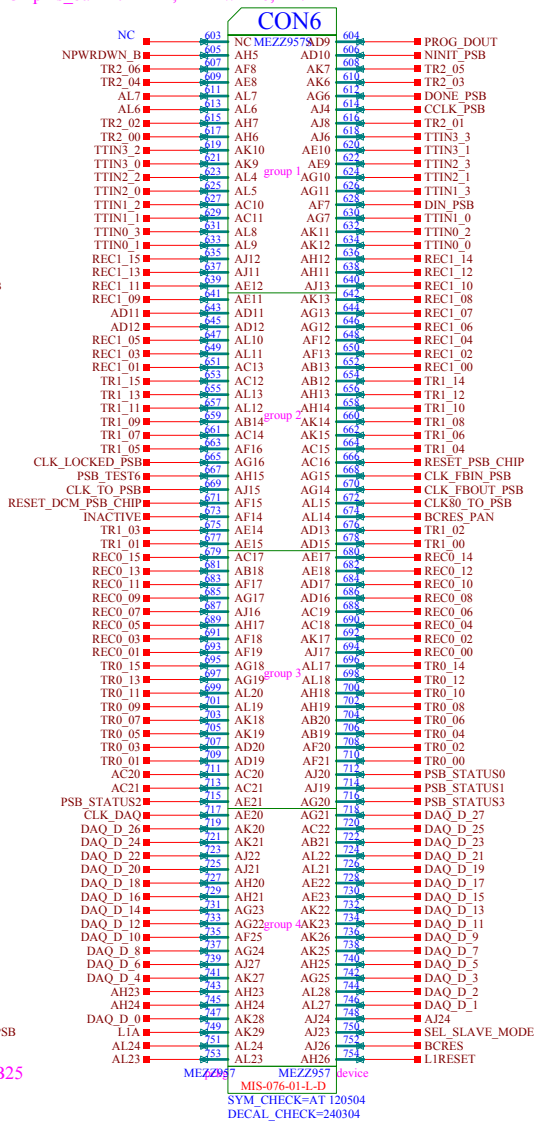
3Kpins_bank1: A8,A9



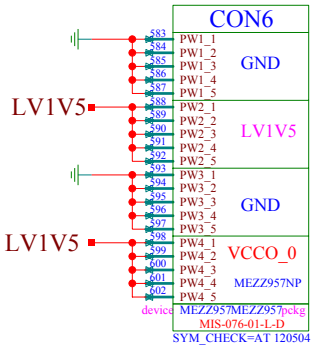
3Kpins_bank0: B23,B25

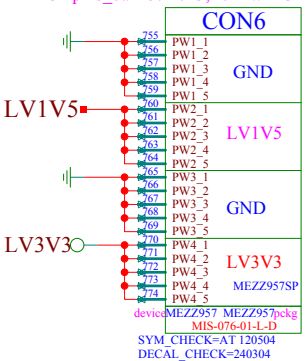
SOUTH MIS_076 connector

3Kpins_bank4: AD11,AD12 //AL6,AL7



3Kpins_bank5: AJ23,AJ24 // AC20,AC21 // AL23,AL24 // AH23,AH24





MEZZ957 TEMPLATE for 9U BOARDS

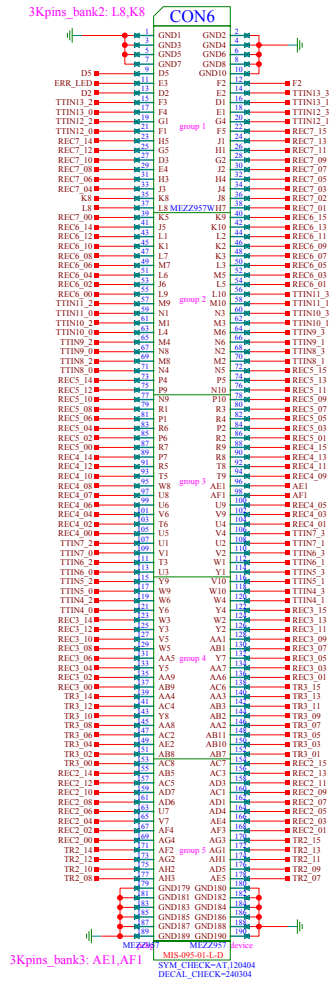
MEZZ957 TEMPLATE		
PSB_CHIP		
HEPHY VIENNA ELEKTRONIK 1	sheet 1	of 3
modified by: A.T 13.2.04	11-17-2004 9:16	
checked by: CHECKER	0-00-0000 00:00	

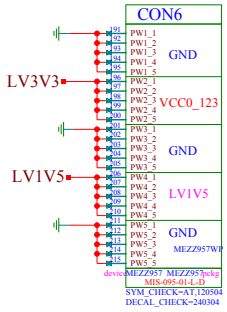
Default: Apply 3.3V to VCCO_0 (=Banks 0)

FDL,AUB,AUF: Apply 1.5V to VCCO_0 (=Banks 0) for GTL+ signals

APPLY always 3.3V to Banks 4,5

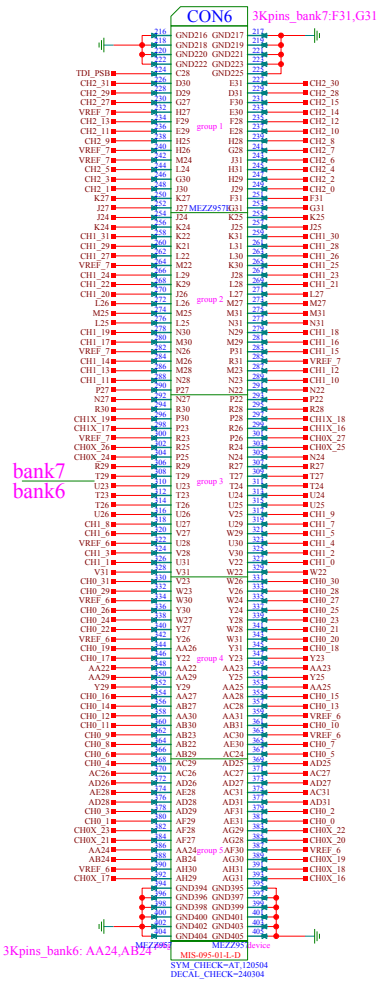
WEST MIS_095 connector

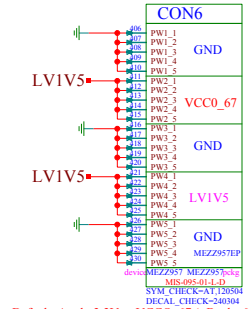




Default: Apply 3.3V to VCCO_123 (=Banks 1,2,3)

EAST MIS_095 connector





Default: Apply 3.3V to VCCO_67 (=Banks 6,7)
AUB,AUF,SRT: Apply 1.5V to VCCO_67 (=Banks 6&7) for GTL+ signals

MEZZ957 TEMPLATE for 9U BOARDS

MEZZ957 TEMPLATE

PSB_CHIP

HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 3
modified by: AT 13.2.04	11-17-2004_9:16
checked by: CHECKER	0-00-0000_00:00

1CLB = 4 slices (as 2x2) 1slice= F+G LUT, 2 FF, hor.cascade OR,

Device	Row x Col.	/Slices/	RAM kbits/	Multiplier/	RAMBlocks (Kbits)/	DCMs/	maxIO pads
XC2V1000	40x32	5120	160	40	40	8	432
XC2V1500	48x40	7680	240	48	48	8	528

XC2V1500: =15360 DFF XC2V2000-4BF957 xxx io
XC2V1500-5FF896 413 $ XC2V3000-4BF957 xxx io

Monitor RAMs: (192+16+68)/16 = 18RAMblocks RingBuf (DPM)
18RAMblocks RO-Buf (Fifo)
1RAMblocks RX-Buf
1RAMblocks L1Aqueue 1 RAMblock =18 kbit DPRAM

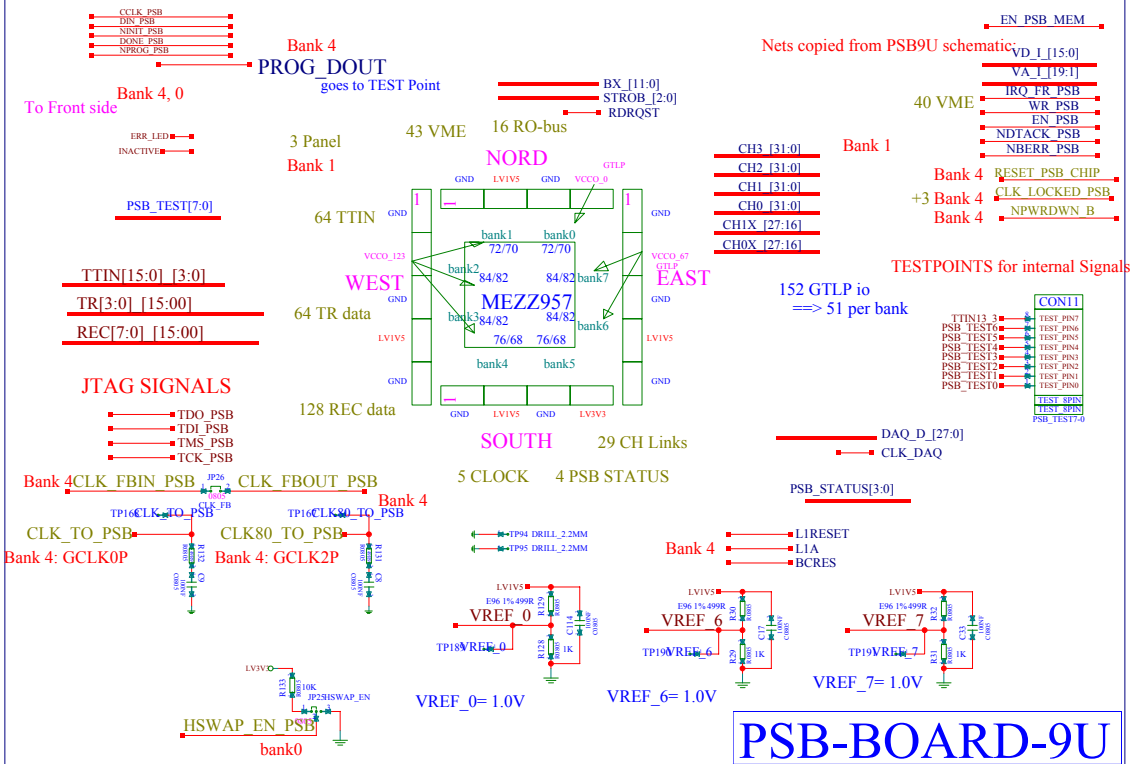
XC2V3000 benötigt 3 Proms XC18V04VQ44C
XC2V4000 benötigt 4 Proms XC18V04VQ44C
XC2V6000 benötigt 6 Proms XC18V04VQ44C

A.T: Ich hoffe dass XC2V2000 genügt.

VREF, CLK, Special PINS have to be on same pins on both MEZZ boards.

RESET Signale to PSB chip
RESET_DCM_PSB...resynchronizes CHIP to CLK
RESET_PSB...==> GSR to STARTUP...resets all registers
INACTIVE ==> GTS to STARTUP...releases IO pins
NSYSRES ==> reconfigures PSB chip from PROM

PSB CHIP: REFDES =CON6

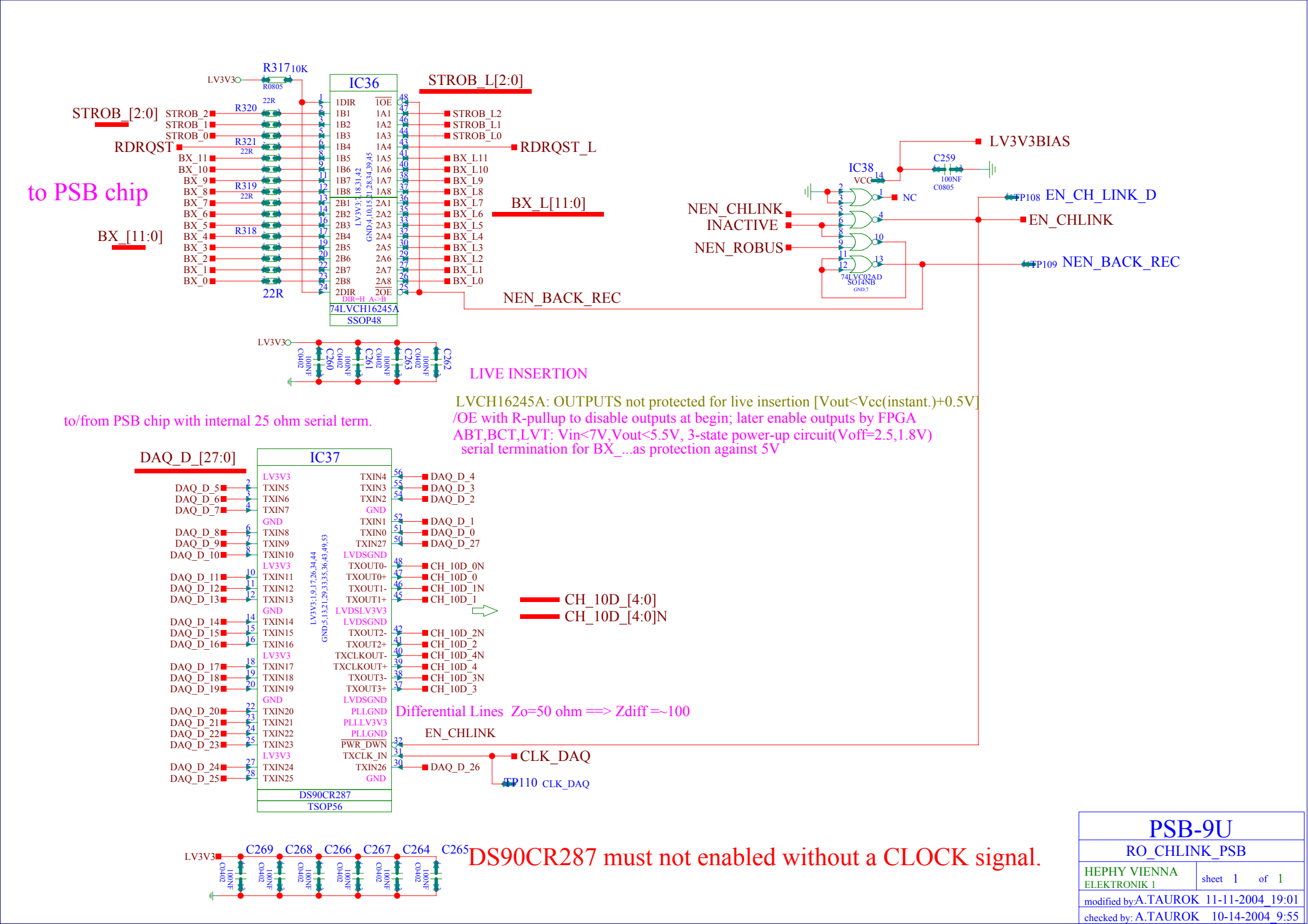


NPWRDWN_B is bidirectional pin!!
NPWRDWN_for each Virtex chip necessary!

PSB-BOARD-9U

PSB_CHIP

HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 3
modified by: AT	10-20-2004_15:08
checked by: CHECKER	0-00-0000_00:00

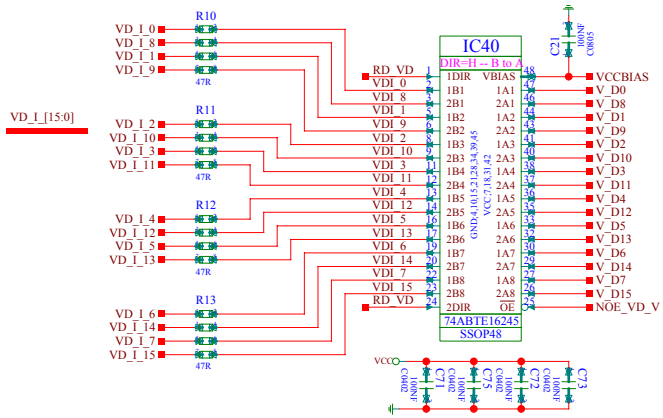


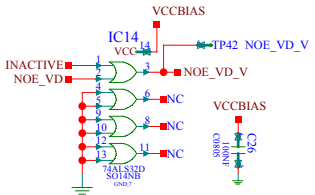
PSB-9U	
RO_CHLINK_PSB	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: A.TAUROK 11-11-2004_19:01	
checked by: A.TAUROK 10-14-2004_9:55	

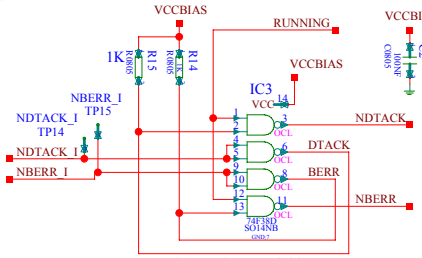
47 Ohm resistors protect the Virtex drivers against overvoltage spikes.

PSB9U must not be inserted into the 6U Backplane!!!

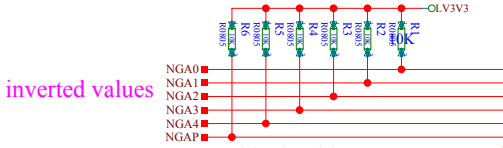
D1 = VCC on BACK6U
D1 = LV3V3bias on BACK9U







Keep NDTACK NBERR inactive while VME64X chip is unconfigured



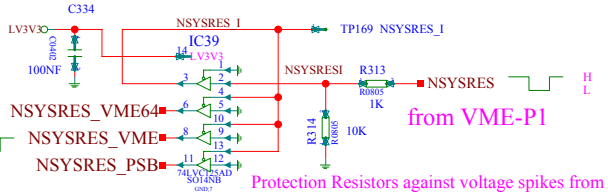
inverted values

Geographical Addresses

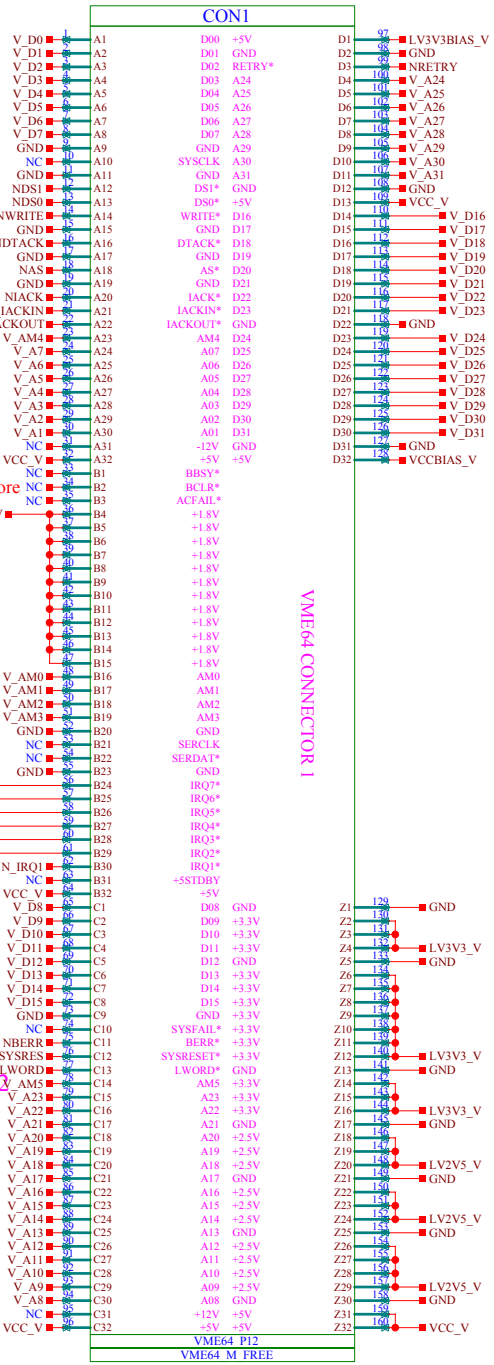
Parity bit: for odd parity

slot=1: GA=0 0001 => GAP=0
slot=2: GA=0 0010 => GAP=0
slot=3: GA=0 0011 => GAP=1
etc

See VME64x.pdf page 10 Table 3-2



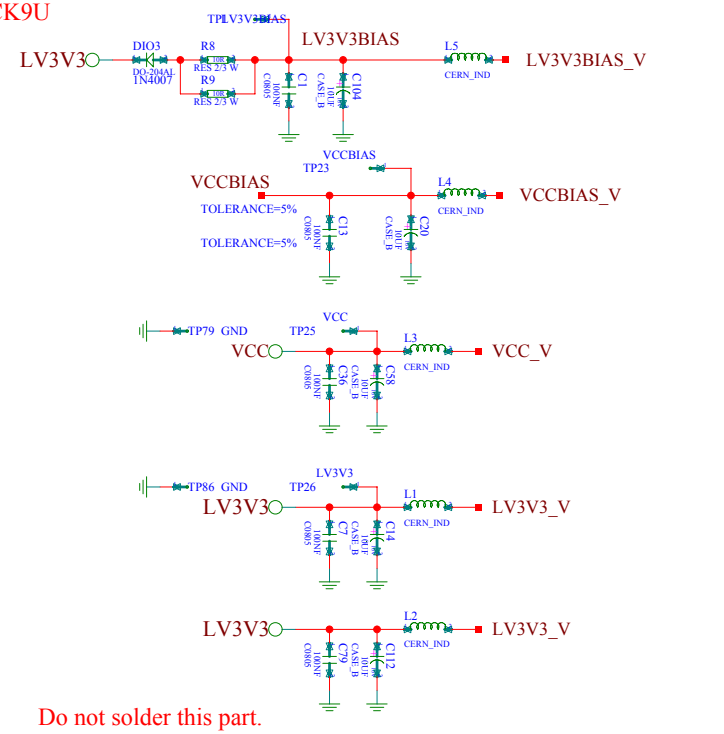
Protection Resistors against voltage spikes from backplane.



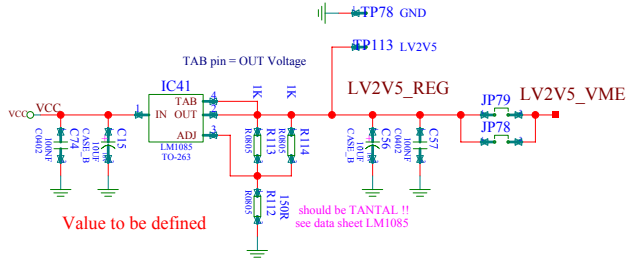
LV1V8_V not used anymore

LV2V5_V not used anymore

Stecker und Signale mit GTL und TIM schematic vergleichen



Unused 2.5V plane can be connected to 3.3V optionally on 9U backplane if GTL-6U is not used anymore in Crate



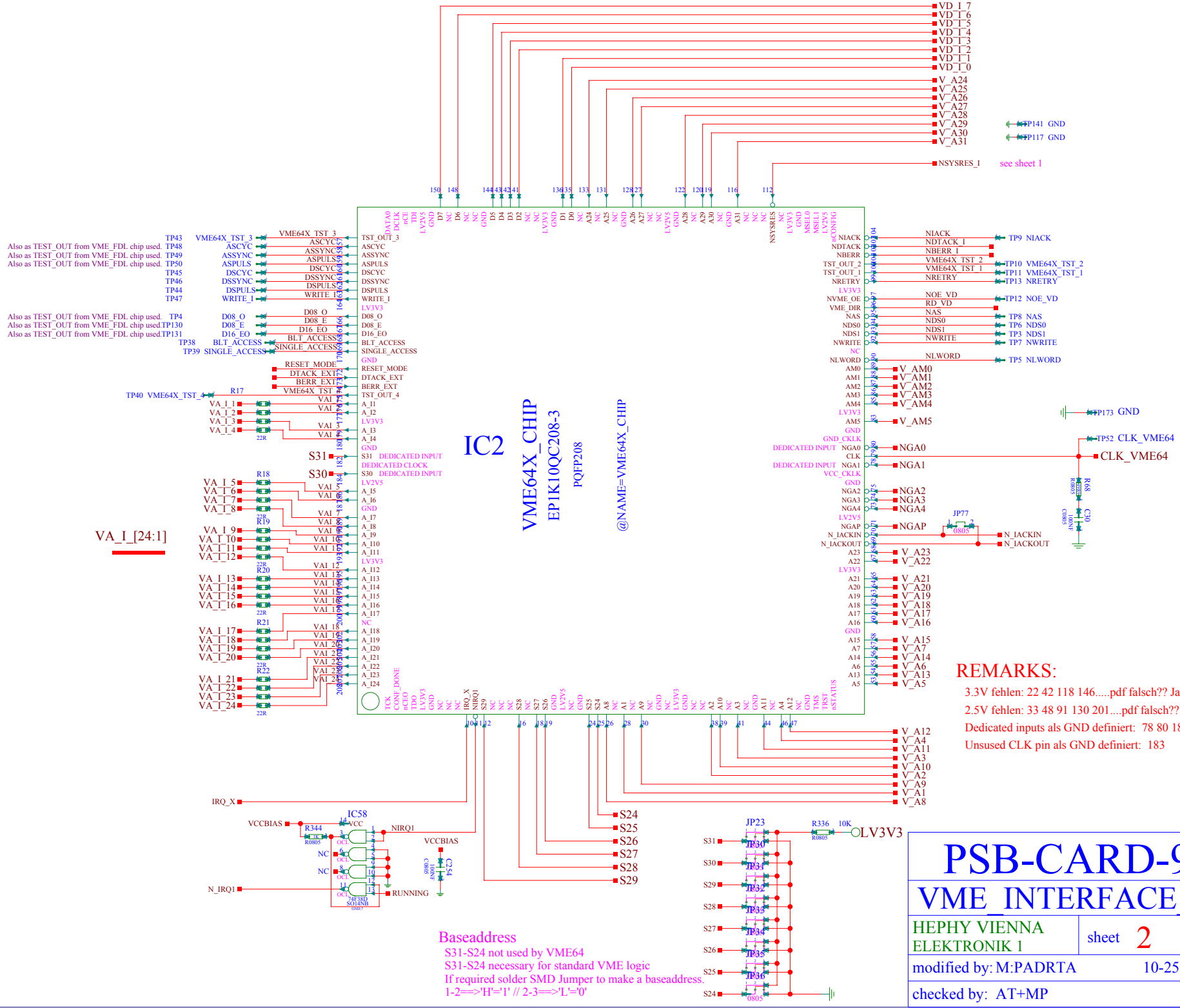
Do not use LV2V5 and LV1V8 from BACKPLANE-6U

LV2V5 and LV1V8 will not be connected on the Backplane-9U

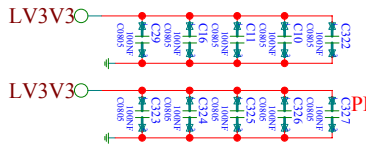
PSB-CARD-9U

VME_INTERFACE_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 3
modified by: H. BERGAUER	11-17-2004_16:59
checked by: A TAUROK	8-18-2003_16:53

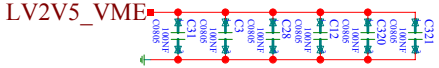

IC2
VME64X_CHIP
EP1K10QC208-3
PQFP208
@NAME=VME64X_CHIP
VA_I_[24:1]
VD_I_7
VD_I_6
VD_I_5
VD_I_4
VD_I_3
VD_I_2
VD_I_1
VD_I_0
V_A24
V_A25
V_A26
V_A27
V_A28
V_A29
V_A30
V_A31
NSYSRES_I
see sheet 1
TP141 GND
TP117 GND
VME64X_TST_3
ASCYC
ASSYNC
ASPULS
DSCYC
DSSYNC
DSPULS
WRITE_I
D08_O
D08_E
D16_EO
BLT_ACCESS
SINGLE_ACCESS
RESET_MODE
DTACK_EXT
BERR_EXT
VME64X_TST_4
TP43
TP48
TP49
TP50
TP45
TP46
TP44
TP47
TP4
TP130
TP131
TP38
TP39
TP40
R17
R18
R19
R20
R21
R22
22R
S31
S30
NIACK
NDTACK_I
NBERR_I
VME64X_TST_2
VME64X_TST_1
NRETRY
NOE_VD
RD_VD
NAS
NDS0
NDS1
NWRITE
NLWORD
TP9 NIACK
TP10 VME64X_TST_2
TP11 VME64X_TST_1
TP13 NRETRY
TP12 NOE_VD
TP8 NAS
TP6 NDS0
TP3 NDS1
TP7 NWRITE
TP5 NLWORD
V_AM0
V_AM1
V_AM2
V_AM3
V_AM4
V_AM5
TP173 GND
TP52 CLK_VME64
CLK_VME64
NGA0
NGA1
NGA2
NGA3
NGA4
NGAP
N_IACKIN
N_IACKOUT
JP77
0805
R68
C30
100NF
V_A23
V_A22
V_A21
V_A20
V_A19
V_A18
V_A17
V_A16
V_A15
V_A7
V_A14
V_A6
V_A13
V_A5
V_A12
V_A4
V_A11
V_A3
V_A10
V_A2
V_A9
V_A1
V_A8
IRQ_X
VCCBIAS
IC58
R344
NIRQ1
N_IRQ1
RUNNING
C254
S24
S25
S26
S27
S28
S29
JP23
R336
10K
LV3V3
TP30
TP31
TP32
TP33
TP34
TP35
TP36
REMARKS:
3.3V fehlen: 22 42 118 146.....pdf falsch?? Ja, pinfile ist Refere
2.5V fehlen: 33 48 91 130 201....pdf falsch?? Ja, pinfile ist Ref
Dedicated inputs als GND definiert: 78 80 182 184
Unsused CLK pin als GND definiert: 183
Baseaddress
S31-S24 not used by VME64
S31-S24 necessary for standard VME logic
If required solder SMD Jumper to make a baseaddress.
1-2==>'H'='1' // 2-3==>'L'='0'
PSB-CARD-9U
VME_INTERFACE_PSB
HEPHY VIENNA
ELEKTRONIK 1
sheet 2 of 3
modified by: M:PADRTA 10-25-2004_17:44
checked by: AT+MP 110504

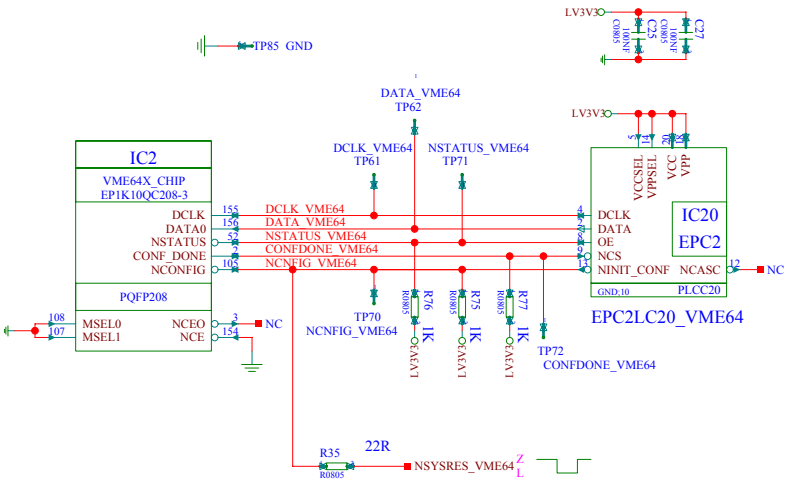
10 pins for 3.3V



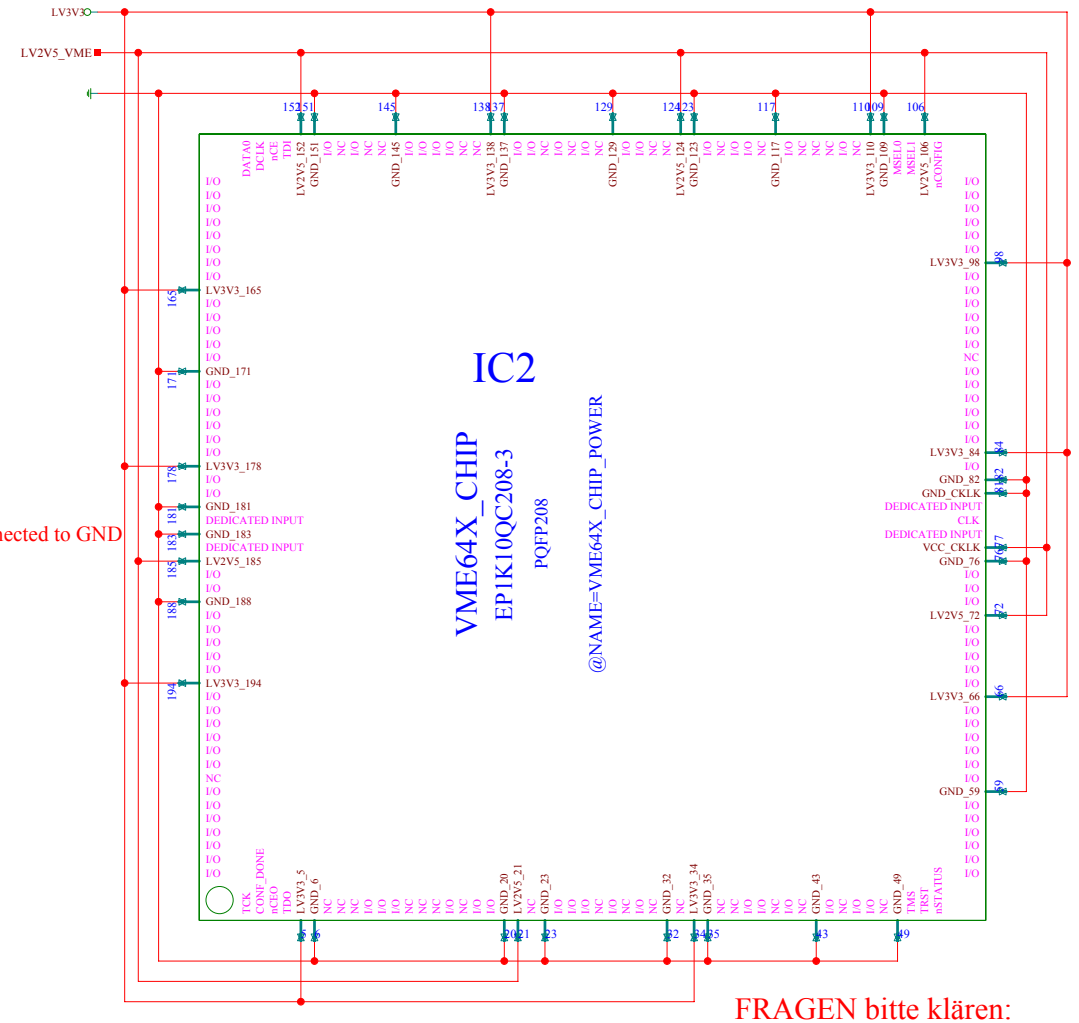
PIN183=dedicated CLOCK input connected to GND

6 pins for 2.5V





See configdevices.pdf:
Do not insert 1K resistor when internal pullup R are used in IC20: EPC2



This pin is the power or ground for the ClockLock and ClockBoost circuitry of a PLL. To ensure noise resistance the power and ground supply to the ClockLock and ClockBoost circuitry should be isolated from the power and ground to the rest of the device. If the PLL is not used, this power or ground pin should be connected to VCCINT or GNDINT, respectively.

FRAGEN bitte klären:

3.3V fehlen: 22 42 118 146.....pdf falsch?? Ja, pinfile ist Referenz, HB 200803

2.5V fehlen: 33 48 91 130 201....pdf falsch?? Ja, pinfile ist Referenz, HB 200803

Dedicated inputs als GND definiert: 78 80 182 184

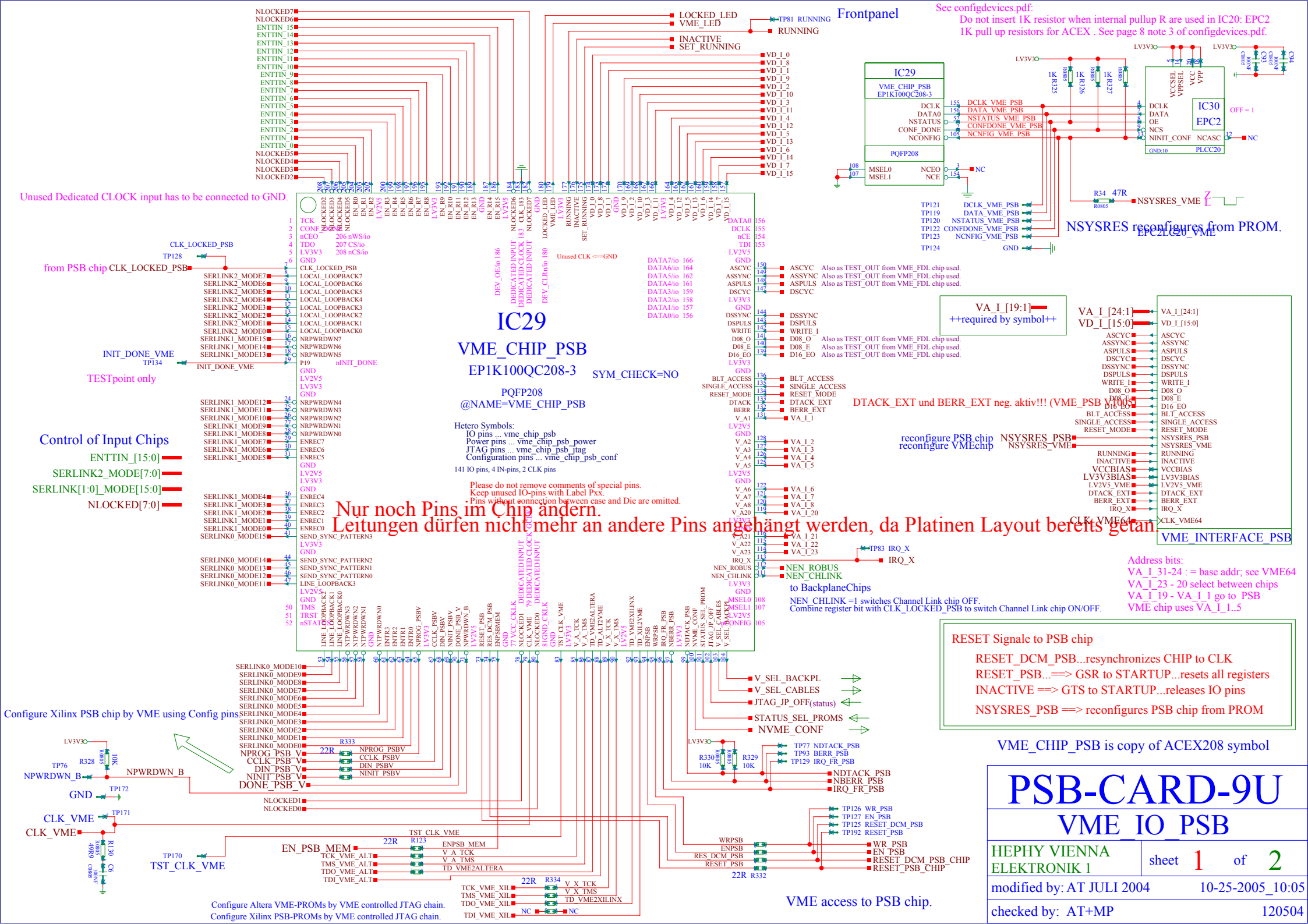
Unused CLK pin als GND definiert: 183

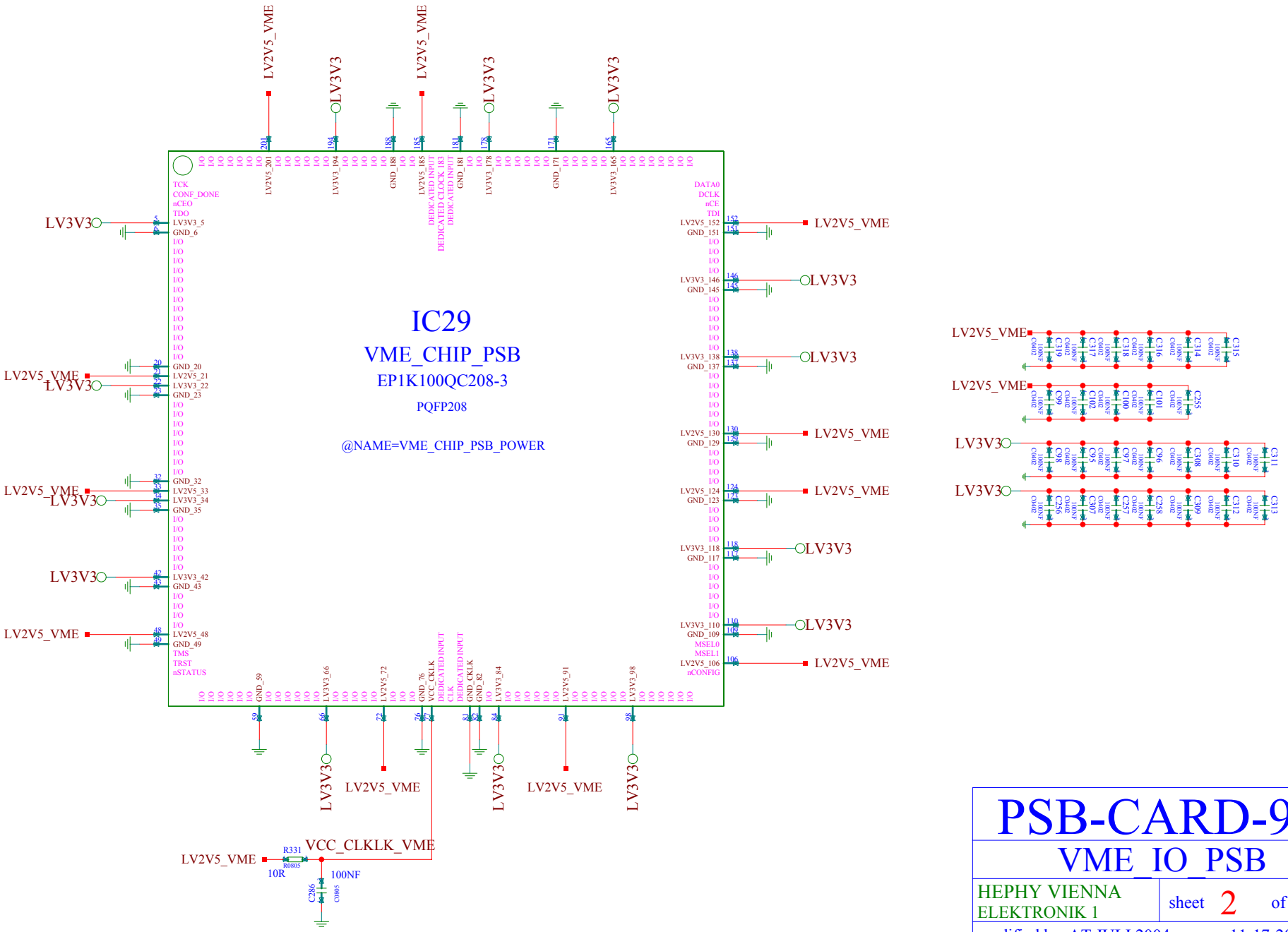
INIT_DONE used as I/O: 19

PSB-CARD-9U

VME_INTERFACE_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 3
modified by: H. BERGAUER	10-28-2004_17:15
checked by: CHECK_NAME	110504

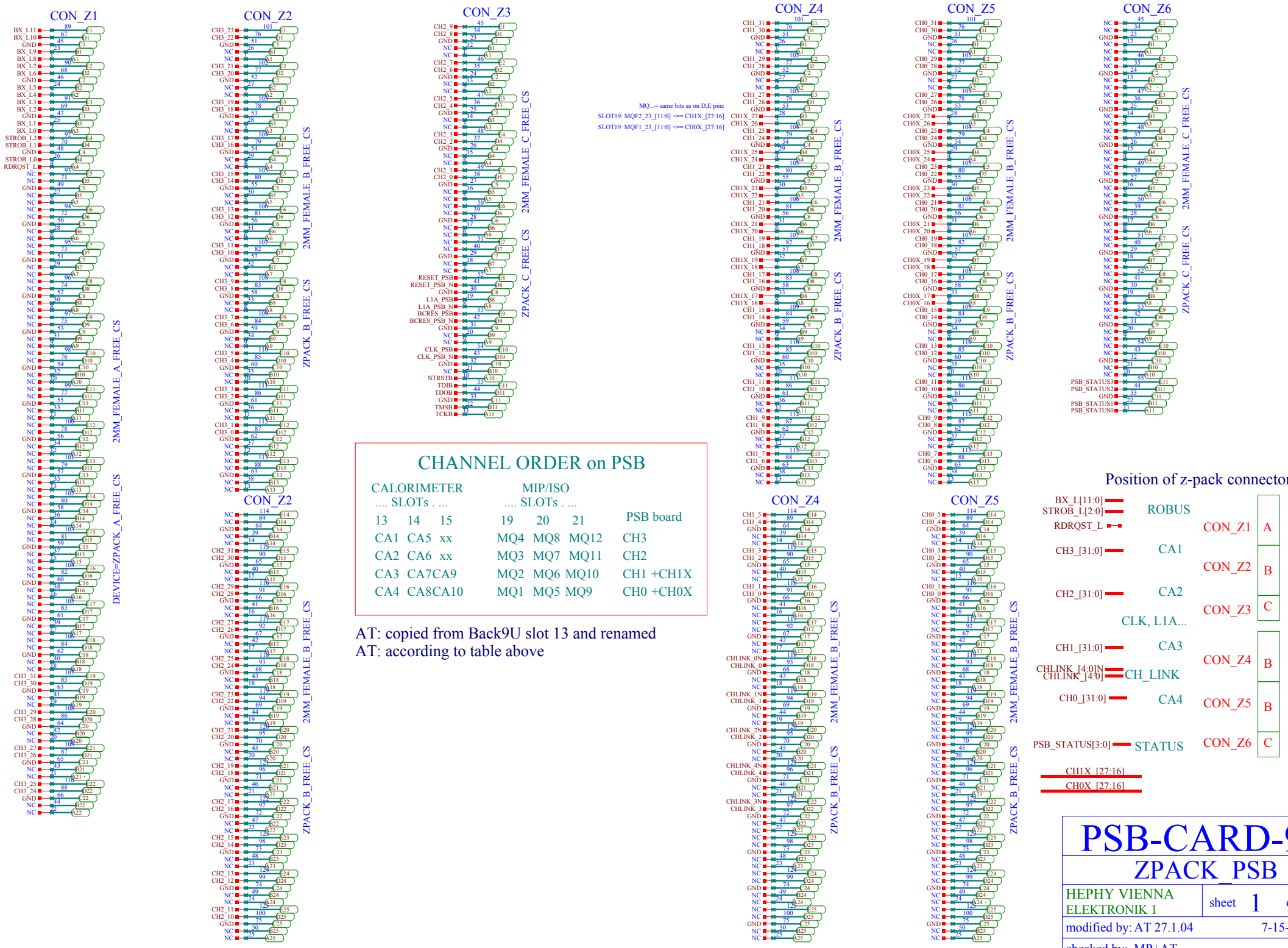

Frontpanel
See configdevices.pdf:
Do not insert 1K resistor when internal pullup R are used in IC20: EPC2
1K pull up resistors for ACEX . See page 8 note 3 of configdevices.pdf.
IC29
VME_CHIP_PSB
EP1K100QC208-3
PQFP208
IC30
EPC2
PLCC20
NSYSRES reconfigures from PROM.
EPC2LC20_VME
Unused Dedicated CLOCK input has to be connected to GND.
IC29
VME_CHIP_PSB
EP1K100QC208-3
PQFP208
@NAME=VME_CHIP_PSB
SYM_CHECK=NO
Hetero Symbols:
IO pins ... vme_chip_psb
Power pins ... vme_chip_psb_power
JTAG pins ... vme_chip_psb_jtag
Configuration pins ... vme_chip_psb_conf
141 IO pins, 4 IN-pins, 2 CLK pins
Please do not remove comments of special pins.
Keep unused IO-pins with Label Pxx.
Pins without connection between case and Die are omitted.
Unused CLK <==GND
DTACK_EXT und BERR_EXT neg. aktiv!!! (VME_PSB V1.06)
VA_I_[19:1]
++required by symbol++
VME_INTERFACE_PSB
Control of Input Chips
ENTTIN_[15:0]
SERLINK2_MODE[7:0]
SERLINK[1:0]_MODE[15:0]
NLOCKED[7:0]
Nur noch Pins im Chip ändern.
Leitungen dürfen nicht mehr an andere Pins angehängt werden, da Platinen Layout bereits getan.
reconfigure PSB chip
reconfigure VMEchip
NSYSRES_PSB
NSYSRES_VME
to BackplaneChips
NEN_CHLINK =1 switches Channel Link chip OFF.
Combine register bit with CLK_LOCKED_PSB to switch Channel Link chip ON/OFF.
Address bits:
VA_I_31-24 : = base addr; see VME64
VA_I_23 - 20 select between chips
VA_I_19 - VA_I_1 go to PSB
VME chip uses VA_I_1..5
RESET Signale to PSB chip
RESET_DCM_PSB...resynchronizes CHIP to CLK
RESET_PSB...==> GSR to STARTUP...resets all registers
INACTIVE ==> GTS to STARTUP...releases IO pins
NSYSRES_PSB ==> reconfigures PSB chip from PROM
VME_CHIP_PSB is copy of ACEX208 symbol
Configure Xilinx PSB chip by VME using Config pins
Configure Altera VME-PROMs by VME controlled JTAG chain.
Configure Xilinx PSB-PROMs by VME controlled JTAG chain.
VME access to PSB chip.
PSB-CARD-9U
VME_IO_PSB
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 2
modified by: AT JULI 2004
10-25-2005_10:05
checked by: AT+MP
120504



PSB-CARD-9U

VME_IO_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 2
modified by: AT JULI 2004	11-17-2004_17:42
checked by: AT+MP	120504



CHANNEL ORDER on PSB

CALORIMETER .... SLOTs			MIP/ISO .... SLOTs			PSB board
13	14	15	19	20	21	
CA1	CA5	xx	MQ4	MQ8	MQ12	CH3
CA2	CA6	xx	MQ3	MQ7	MQ11	CH2
CA3	CA7	CA9	MQ2	MQ6	MQ10	CH1 +CH1X
CA4	CA8	CA10	MQ1	MQ5	MQ9	CH0 +CH0X

AT: copied from Back9U slot 13 and renamed
AT: according to table above

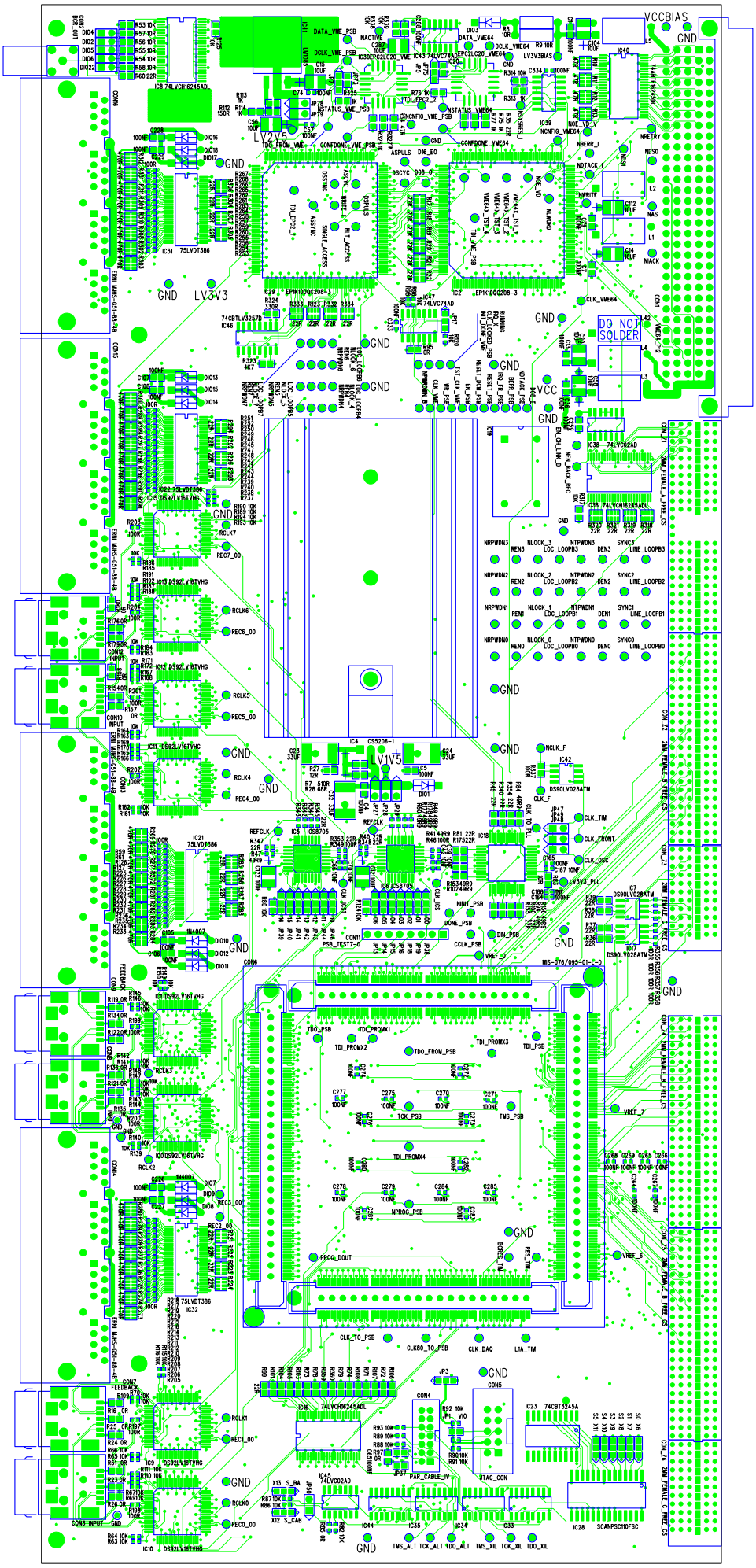
Position of z-pack connectors

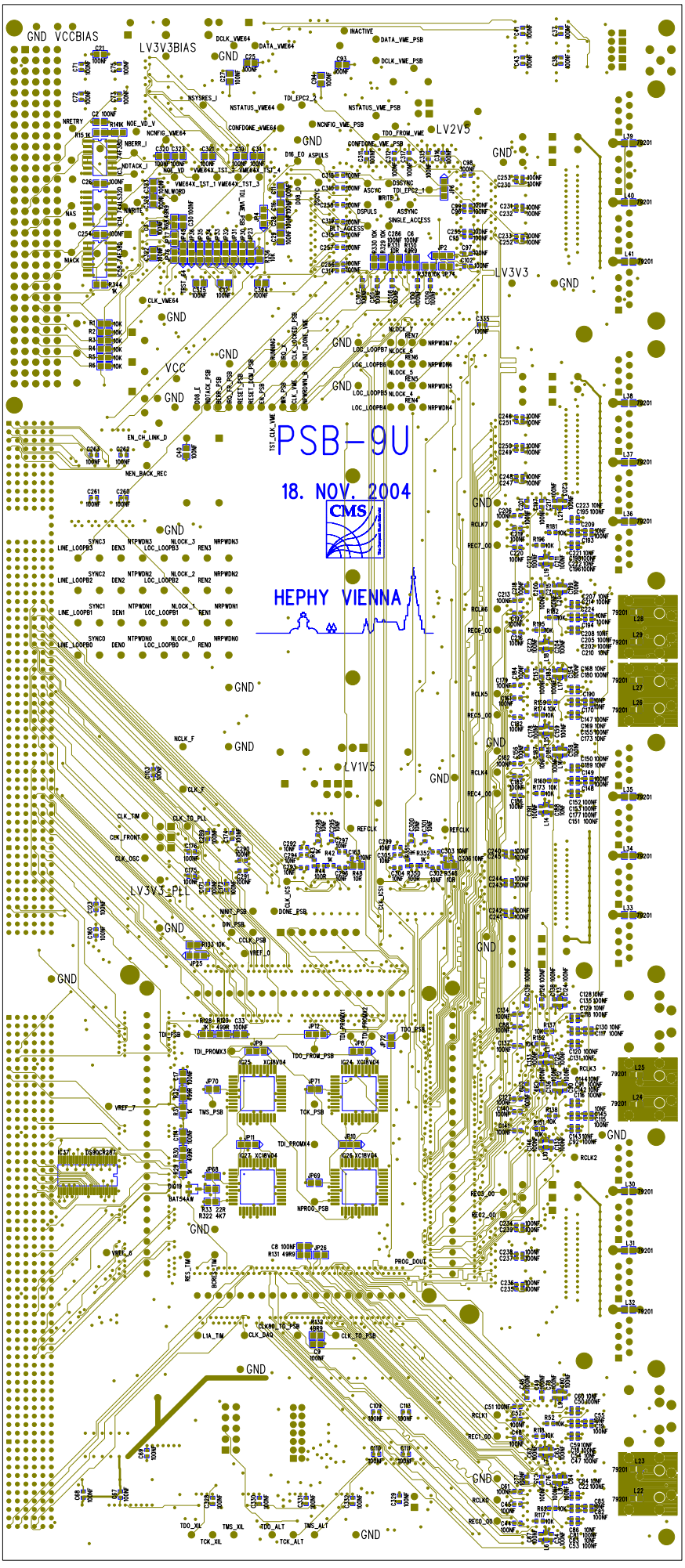
PSB-CARD-9U

ZPACK_PSB

HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: AT 27.1.04	7-15-2004_16:30
checked by: MP+AT	120504


BCRES
63 - 60
59 - 56
55 - 52
51 - 48
47 - 44
43 - 40
39 - 36
35 - 32
LOCK
IN 6-7
INACT
IN 4-5
ERROR
31 - 28
27 - 24
23 - 20
19 - 16
VME
OUT 2-3
RUN
IN 2-3
15 - 12
11 - 8
7 - 4
3 - 0
OUT 0-1
IN 0-1


VCCBIAS
GND
LV2V5
LV3V3
LV1V5
VCC
DO NOT SOLDER
DATA_VME_PSB
DCLK_VME_PSB
INACTIVE
NSTATUS_VME64
CONFDONE_VME64
NCONFIG_VME64
TDO_FROM_VME
TDO_FROM_PSB
TDI_PROMX1
TDI_PROMX2
TDI_PROMX3
TDI_PROMX4
TDI_PSB
TDO_PSB
TCK_PSB
TMS_PSB
NPROG_PSB
PROG_DOUT
DONE_PSB
CCLK_PSB
NINIT_PSB
DIN_PSB
VREF_10
VREF_7
VREF_6
CLK_TO_PSB
CLK80_TO_PSB
CLK_DAQ
LTA_TIM
CLK_TIM
CLK_FRONT
CLK_OSC
CLK_F
NCLK_F
MIS-076/095-01-E-D
JTAG_CON
PAR_CABLE_JV
SCANPSC110FSC
TMS_ALT
TCK_ALT
TDO_ALT
TMS_XIL
TCK_XIL
TDO_XIL


PSB-9U
18. NOV. 2004
CMS
HEPHY VIENNA
GND
VCCBIAS
LV3V3BIAS
LV2V5
LV3V3
LV1V5
VCC
LV3V3_PLL

Jumper and switches on PSB-9U-card

JP1 (top side)**:** selection of VIO of masterblaster
OFF ➔ no voltage on VIO.
ON ➔ LV3V3 or VCC on VIO (see JP3).

JP2 (bottom side)**:** TRST (JTAG) of VME-chip
1-2 (default)➔ solder R with 10K (LV3V3), TRST inactive.
2-3 ➔ do not solder.

JP3 (top side)**:** voltage selection for masterblaster
1-2 (default)➔ LV3V3.
2-3 ➔ VCC.

JP4 (bottom side)**:** VME64x-chip in JTAG-chain
1-2 ➔ VME64x-chip in JTAG-chain.
2-3 (default)➔ VME64x-chip **not** in JTAG-chain.

JP5 (top side)**:** PROM of VME64x-chip in JTAG-chain
1-2 ➔ PROM of VME64x-chip in JTAG-chain.
2-3 (default)➔ PROM of VME64x-chip **not** in JTAG-chain.

JP6 (bottom side)**:** VME-chip in JTAG-chain
1-2 ➔ VME-chip in JTAG-chain.
2-3 (default)➔ VME-chip **not** in JTAG-chain.

JP7 (top side)**:** PROM of VME-chip in JTAG-chain
1-2 ➔ PROM of VME-chip in JTAG-chain.
2-3 (default)➔ PROM of VME-chip **not** in JTAG-chain.

JP8 (bottom side)**:** 1st PROM of PSB-chip in JTAG-chain
1-2 ➔ in JTAG-chain.
2-3 (default)➔ **not** in JTAG-chain.

JP9 (bottom side)**:** 2nd PROM of PSB-chip in JTAG-chain
1-2 ➔ in JTAG-chain.
2-3 (default)➔ **not** in JTAG-chain.

JP10 (bottom side)**:** 3rd PROM of PSB-chip in JTAG-chain
1-2 ➔ in JTAG-chain.
2-3 (default)➔ **not** in JTAG-chain.

JP11 (bottom side)**:** 4th PROM of PSB-chip in JTAG-chain
1-2 ➔ in JTAG-chain.
2-3 (default)➔ **not** in JTAG-chain.

JP12 (bottom side)**:** PSB-chip in JTAG-chain
1-2 ➔ PSB-chip in JTAG-chain.
2-3 (default)➔ PSB-chip **not** in JTAG-chain.

JP13, JP14, JP15, JP16, JP18, JP19 and JP38 (top side)**:** jumper for SELxx input of PLL (IC6).

JP17 (top side)**:** jumper for "SEL_PROMS"
(default)➔ nothing inserted (NSYSRES_PSB works).

JP20 - JP22: not in design!!!

JP23, JP30, JP31, JP32, JP33, JP34, JP35 and JP36 (bottom side)**:** S31-S24 for base address, **not used** in VME64x-systems!!!
1-2 ➔ '1'.
2-3 ➔ '0'.

JP24: not in design!!!

JP25 (bottom side)**:** HSWAP_EN input of PSB-chip
1-2 ➔ HSWAP_EN=LV3V3, keeps jumper in OFF-position (I/O-pins in high-Z before configuration). In this position **configuration of PSB-chip** via **VME not possible**.
2-3 (default)➔ HSWAP_EN=GND, enables pull-up-Rs of all I/O-pins in PSB-chip before configuration. In this position **configuration of PSB-chip via VME possible**.

JP26 (bottom side)**:** jumper for "CLK_FBxxx"
(default)➔ not inserted.

JP27, JP28 and JP29 (top side)**:** jumper for "LV1V5"
(default)➔ solder-bridges after voltage-testing.

JP37 (top side)**:** jumper for "VREF" of Parallel-Cable IV
(default)➔ not inserted.

JP39, JP40, JP41, JP42, JP43, JP44 and JP46 (top side)**:** jumper for SELxx input of PLL (IC5).

JP45 (bottom side)**:** TRST (JTAG) of VME64x-chip
1-2 (default)➔ solder R with 10K (LV3V3), TRST inactive.
2-3 ➔ do not solder.

JP47, JP48 and JP49 (top side)**:** selection of CLK_TO_PLL
Only one jumper may be ON!
JP47 ON (default)➔ CLK_TIM.
JP48 ON ➔ CLK_OSC (from oscillator).
JP49 ON ➔ CLK_FRONT.

JP50 (top side)**:** jumper for "SEL_CABLE_JTAG"
OFF (default)➔ selection via VME.
ON ➔ MB and PC-IV selected for JTAG.

JP51 - JP67: not in design!!!

JP68, JP69, JP70, JP71 and JP72 (bottom side)**:** jumper for "TMS-signals" for PROMs and PSB-chip. These jumpers are set in the same way as JP8-JP12.
OFF ➔ PROM **not** in JTAG-chain.
ON (default)➔ PROM in JTAG-chain.
JP8: 1st PROM of PSB-chip
JP9: 2nd PROM of PSB-chip
JP10: 3rd PROM of PSB-chip
JP11: 4th PROM of PSB-chip
JP12: PSB-chip

JP73, JP74, JP75 and JP76 (both sides)**:** jumper for "TMS-signals" for PROMs and VME-chips. These jumpers are set in the same way as JP4-JP7.
OFF ➔ PROM **not** in JTAG-chain.
ON (default)➔ PROM in JTAG-chain.
JP4: VME64x-chip
JP5: PROM of VME64x-chip
JP6: VME-chip
JP7: PROM of VME-chip

JP77 (bottom side)**:** N_IACKIN/N_IACKOUT
ON ➔ always on, no interrupt.

JP78 and JP79 (top side)**:** jumper for "LV2V5_VME"
(default)➔ solder-bridges after voltage-testing.

X1 - X3: drill-holes!!!

X4 - X5: not in design!!!

X6 - X11 (top side)**:** jumper for JTAG-code from backplane for SCANPSC110
(default)➔ not used now.

X12 (top side)**:** connect V_SEL_CABLES from VME
1-2 (default)➔ connected.
2-3 ➔ GND connection.

X13 (top side)**:** connect V_SEL_BACKPL from VME
1-2 (default)➔ connected.
2-3 ➔ GND connection.

VME64X-CHIP
(Version 0x100C)

of
PSB-9U-card
(9U-Version)

H. Bergauer, K. Kastner, M. Padrta, A. Taurok

Jan-06

Version 0x100C

1 Introduction

The VME64x Interface for Global Trigger boards is made for a slave module without interrupt capabilities. It works in systems with backplanes supplying VME64x standard as well as in systems with VME/VME64 backplanes. The Interface will contain a VME64x-chip, a VME-CHIP-PSB, transceivers for VME-data, logic for "live-insertion", DTACK*- and BERR*-drivers and a special VME64x connector (P1/J1).

2 VME64x-chip

2.1 Versionshistory

- V1001: block-transfer via VME implemented (F1). (HB, 270504)
- V1003: 32-bit-transfer via VME implemented. (HB, 100804)
- V1004: same as V1001, but NLD_ADDR freezes addresses. (HB, 211004)
- V1005: same as V1003 (32-bit-transfer), but NLD_ADDR freezes addresses. (HB, 221104). **Do not use for PSB-card**.
- V1006: same as V1005 (32-bit-transfer), but with TEST_OUT-selection in USER_CSR. (HB, 291104). **Do not use for PSB-card**.
- V1007: same as V1004 (16-bit-transfer), but with TEST_OUT-selection in USER_CSR. (HB190805)
- V1008: same as V1007 (16-bit-transfer), but VIEWDRAW in local directory and working with vme_chips_lib. Changes in timing - TIMING_V2_1 used. (HB140905)
- V1009: based on V1008 (16-bit-transfer), but complete new fully synchronous design implemented. (HB041005)
- V100A: based on V1009 (16-bit-transfer), but DTACK_EXT and BERR_EXT from VME-CHIP-PSB are used as negative active signals now (because at power-up configuration of VME64X-CHIP is faster than configuration of VME-CHIP and therefore wrong DTACK and BERR signals are generated after configuration, which causes LEDs="on" of CAEN-controller). Card-number is on S27-S24 (CARD_NR[3..0]). (HB201005)
- V100B: based on V100A (16-bit-transfer), but INIT_DONE-feedback on pin 19 (S26 and pin 18 (S27) is implemented to have no wrong DTACK and BERR signals during init-phase after configuration-phase. Card-number is on S31-S28 (CARD_NR[3..0]) now. (HB241005)
- **V100C: based** on V100B of VME64x-chip, but AM=0x2F is combined with BASE_ADDR_CR_CSR to generate correct NVME_OE. Therefore geo_addr_v2_0 and vme_d16_v1_6 are used. (HB071205)

2.2 Hardware

The VME64x-chip is an Altera EP1K10QC208-3.

2.3 Firmware

```
serial-nr.:   PSBxx          (xx = CARD_NR [dec] jumpers S31-S28)
chip_id:      0x00018n11     (n = CARD_NR [hex] jumpers S31–S28)
version:      0x0000100C
```

2.4 References

See VME64-specification and VME64x-specification for definitions.

2.5 Features of the VME64x-chip (V100C)

- **User Configuration ROM** (USER_CR: address range 0x01003..0x01033, size is 13 bytes) for **"chip identifier" and "version"** of VME64x-chip (see 2.7.3) [0x01003-0x0101F] and for **"serial number"** of board (see 2.7.3) [0x01023-0x01033].
- **"Card number"** is part of **"chip identifier"** and is fix soldered by jumpers on the lines S31-S28 (CARD_NR[3..0]).
- **"Serial number"** is **PSBxx** (xx is the decimal expression of **"card number"** with leading zero - exception: PSB16 has "card number"=0).
- **User Command Status Register** (USER_CSR: address range 0x05003..0x0502F, size is 12 bytes) for the **"TEST_OUT-selection-registers"** is implemented (see USER_CSR space).
- **Function 0** (F0) – **D16** only, base-address at **A31-A25**, AM=**0x0D** and **0x09** (only **single** transfer).
- **Function 1** (F1) – **D16** only, base-address at **A31-A25**, AM=**0x0F** and **0xoB** (only **block** transfer).

2.6 Address spaces overview

AM: **0x2F**, access: **D08_O**
A23-A19: Geographic address (=VME slot number) or '11110'=amnesia address

A18-A00	**=>**	**Register-name**
0x00003 - 0x007FF	=>	512x8 bit Configuration ROM (read)
0x01003	=>	chip-id_3 (read)
0x01007	=>	chip-id_2 (read)
0x0100B	=>	chip-id_1 (read)
0x0100F	=>	chip-id_0 (read)
0x01013	=>	version_3 (read)
0x01017	=>	version_2 (read)
0x0101B	=>	version_1 (read)
0x0101F	=>	version_0 (read)
0x01023 - 0x01033	=>	5 bytes Serial Number [PSBxx] (read)
0x03003 - 0x037FF	=>	CRAM 512x8 bit RAM (not used!!) (read/write)
0x05003 - 0x05007	=>	TEST_OUT-selection in USER_CSR (read/write)
[0x7FC03 - 0x7FFFF	=>	Command/Status registers (read/write)]
0x7FF63	=>	ADER-F0_3 register (read/write)
0x7FF67	=>	ADER-F0_2 register (read/write)
0x7FF6B	=>	ADER-F0_1 register (read/write)
0x7FF6F	=>	ADER-F0_0 register (read/write)
0x7FF73	=>	ADER-F1_3 register (read/write)
0x7FF77	=>	ADER-F1_2 register (read/write)
0x7FF7B	=>	ADER-F1_1 register (read/write)
0x7FF7F	=>	ADER-F1_0 register (read/write)
0x7FFF7	=>	Bit Clear Register [BCR] (read/write)
0x7FFFB	=>	Bit Set Register [BSR] (read/write)
0x7FFFF	=>	BAR - Geographic address (read)

2.7 Parts of the VME64x-chip

2.7.1 Defined Configuration ROM (CR)

The definition of the CR is made in the VME64x-specification (10.2.1 The defined CR area, page 39 and Table 10-12, page 53).

- Checksum (0x03): see VME64-specification (Table 2-32, page 55)
  not calculated yet, to be done in cr.mif!!!
- Length of ROM (0x07..0x0F): see VME64-specification (Table 2-32, page 55)
  not calculated yet, to be done in cr.mif!!!
- Configuration ROM data access width (0x13): see VME64-specification (Table 2-32, page 55)
  0x81 => "Only use D08(O), every fourth byte".
- CSR data access width (0x17): see VME64-specification (Table 2-32, page 55)
  0x81 => "Only use D08(O), every fourth byte".
- CR/CSR space specification ID (0x1B): see VME64x-specification (Rule 10.3, page 39)
  0x02 => VME64x.
- Manufactor's ID (0x27..0x2F): see VME64-specification (Table 2-32, page 56)
  0x00.
- Board ID (0x33..0x3F): see VME64-specification (Table 2-32, page 56)
  not fixed yet, has to be defined for all boards of the GT-system!!
- Revision ID (0x43..0x4F): see VME64-specification (Table 2-32, page 56)
  not fixed yet, has to be defined for all boards of the GT-system!!
- Program ID (0x7F): see VME64-specification (Table 2-32, page 56)
  0x01 => "No program, ID ROM only".
- Offset to BEG_USER_CR (0x83..0x8B): see VME64x-specification (Table 10-12, page 53)
  0x01003 => used for chip_id- and version-register.
- Offset to END_USER_CR (0x8F..0x97): see VME64x-specification (Table 10-12, page 53)
  0x0101F => used for chip_id- and version-register.
- Offset to BEG_CRAM (0x9B..0xA3): see VME64x-specification (Table 10-12, page 53)
  0x03003 => used for future applications.
- Offset to END_CRAM (0xA7..0xAF): see VME64x-specification (Table 10-12, page 53)
  0x037FF => used for future applications.
- Offset to BEG_USER_CSR (0xB3..0xBB): see VME64x-specification (Table 10-12, page 53)
  0x05003 => used for TEST_OUT-selection register..
- Offset to END_USER_CSR (0xBF..0xC7): see VME64x-specification (Table 10-12, page 53)
  0x0502F.
- Offset to BEG_SN (0xCB..0xD3): see VME64x-specification (Table 10-12, page 53)
  0x01023 => part of USER_CR, contains the "serial number".
- Offset to END_SN (0xD7..0xDF): see VME64x-specification (Table 10-12, page 53)
  0x01033.
- Slave characteristics parameter (0xE3): see VME64x-specification (Table 10-1, page 40)
  0x00.
- Master characteristics parameter (0xEB): see VME64x-specification (Table 10-2, page 40)
  0x00.
- CRAM_ACCESS_WIDTH (0xFF): see VME64x-specification (Table 10-10, page 49)

0x81 => "Only use D08(O), every fourth byte".

- Function 0 and 1 DAWPR (0x103..0x107): see VME64x-specification (Table 10-3, page 42)
  0x83 => "Accepts D16 or D08(EO) cycles".
- Function 0 AMCAP (0x123..0x13F): see VME64x-specification (Table 10-5, page 44)
  0x0000 0000 0000 2200 => AM=0x0D and 0x09 "extended data access" - single access.
- Function 1 AMCAP (0x143..0x15F): see VME64x-specification (Table 10-5, page 44)
  0x0000 0000 0000 8800 => AM=0x0F and 0x0B "standard data access" - single access.
- Function 0 and 1 ADEM (0x623..0x63F): see VME64x-specification (Table 10-4, page 43)
  0xFE000000 => "mask bits 31-25=1".

2.7.2 Defined Control/Status Register (CSR)

The definition of the CSR is made in the VME64x-specification (10.2.2 The defined CSR area, page 45 and Table 10-13, page 55).

- **Base Address Register** (BAR) (0x7FFFF): see VME64x-specification (Table 10-13, page 55), set with geographical address or amnesia address.
- **Bit Set Register** (BSR) (0x7FFFB): see VME64x-specification (Table 10-13, page 55), for settting see Table 10-6, page 45.
- **Bit Clear Register** (BCR) (0x7FFF7): see VME64x-specification (Table 10-13, page 55), for settting see Table 10-7, page 46.
  BCR, BSR bits:
  Bit 7: EN/DIS RESET_MODE
  Bit 4: EN/DIS MODULE
  Bit 3: EN/DIS BERR FLAG
- **Function 1 ADER** (0x7FF73..0x7FF7F): see VME64x-specification (Table 10-13, page 55), used for address relocation with Function 1 ADEM and Function 1 AMCAP (see Table 10-8, page 47).
- **Function 0 ADER** (0x7FF63..0x7FF6F): see VME64x-specification (Table 10-13, page 55), used for address relocation with Function 0 ADEM and Function 0 AMCAP (see Table 10-8, page 47).

2.7.3 Chip_ID and version ROM space

A user configuration ROM is implemented for the "chip_ID" and "version" of the VME64x-chip of the board. It is located at the addresses 0x01003-0x0101F, size is 8 bytes, part of the USER_CR.

2.7.4 Serial Number ROM space

A user configuration ROM is implemented for the "serial number" of the board. It is located at the addresses 0x01023-0x01033, size is 5 bytes (PSBxx), part of the USER_CR.

2.7.5 USER_CSR space

A "user command status register (USER_CSR)" is implemented for the „TEST_OUT-selection-registers".

2.7.5.1 TEST_OUT registers

TEST_OUT-registers are used to select internal signals to a certain TEST_OUT-pin. There are four TEST_OUT-pins implemented, each pin can driven by 1 of 16 internal signals. So 4 bits are used for the code of the selection of internal signals. We have two registers, one for the selection of TST_OUT_1 and TST_OUT_2, the other for TST_OUT_3 and TST_OUT_4.

2.7.5.2 Test-signal-definition

Code for the selection of internal signals for TEST_OUT-pins:

sel_test_out_x[3:0]		**testsignal-name**
0000	➔	TST_CLK_VME
0001	➔	D08_O_I
0010	➔	D16_EO_I
0011	➔	LD_CNT
0100	➔	BLT_CNT
0101	➔	CNT_EN
0110	➔	DTACK_CR_CSR
0111	➔	RD_CR
1000	➔	ASCYC_I
1001	➔	ASSYNC_I
1010	➔	ASPULS_I
1011	➔	DTACK_EXT_I
1100	➔	BERR_EXT_I
1101	➔	D32_EO_I
1110	➔	D08_E_I
1111	➔	MODULE_ENABLED

2.7.5.3 Registerdefinition

0x05003 => sel_test_out_12 (write/read)

D7	D6	D5	D4	D3	D2	D1	D0
sel_test_out_2[3:0]				**sel_test_out_1[3:0]**			

0x05007 => sel_test_out_34 (write/read)

D7	D6	D5	D4	D3	D2	D1	D0
sel_test_out_4[3:0]				**sel_test_out_3[3:0]**			

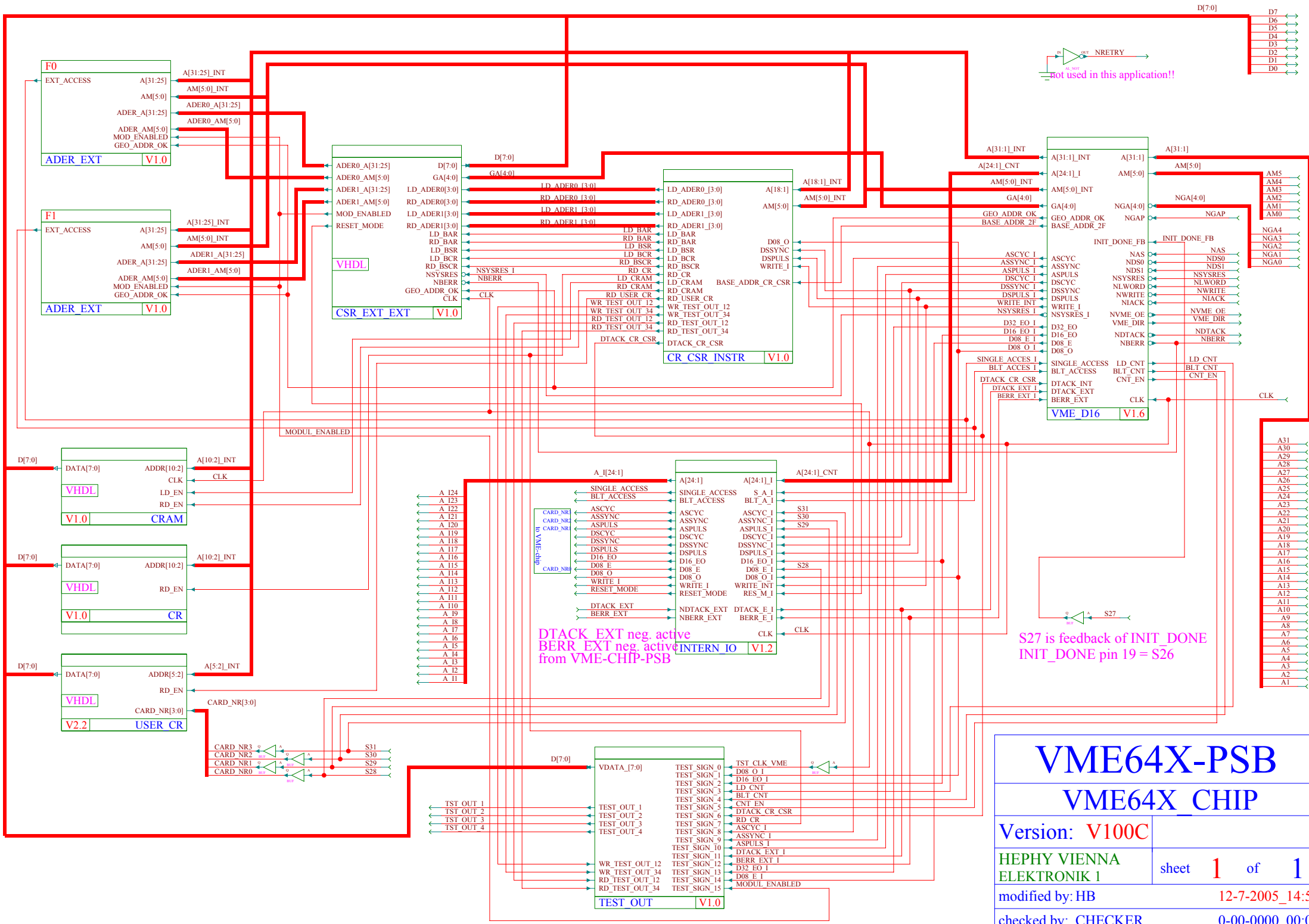
2.7.6 CRAM space

The configuration RAM (CRAM) is defined as a RAM for special purpose. The size is 512 bytes. The CRAM is located at addresses 0x03003..0x037FF.
The contains of the CRAM has to be defined!!!

3 Softwareguide for the VME64x Interface

3.1 Module enable

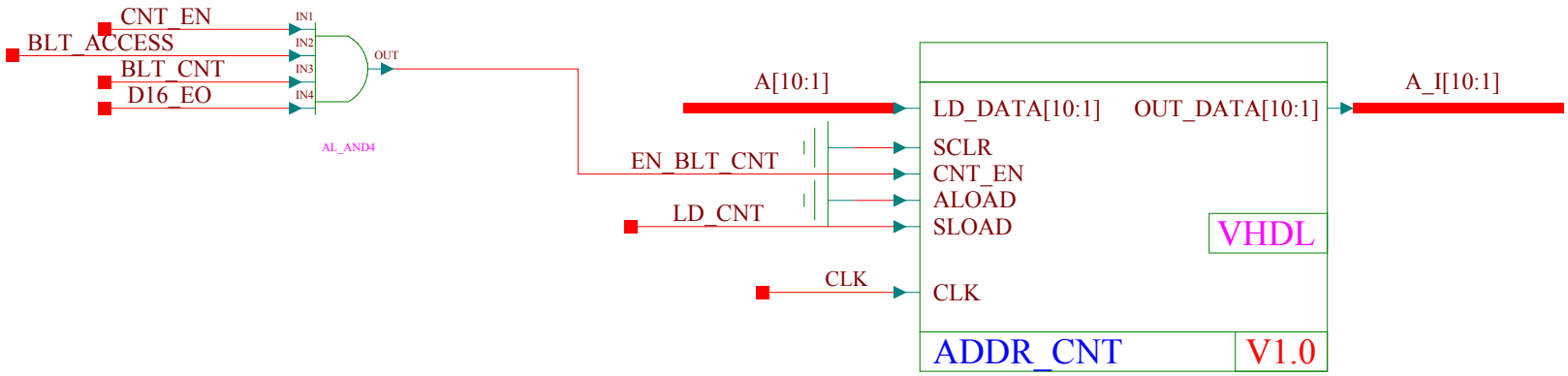
After power-up the module is disabled through the default value of the “ENABLE MODULE”-bit of the BitSet-register in the CommandStatusRegister (CSR) of VME64x.
To enable the module, one has to set bit 4 in the CSR, that means to write 0x10 to address 0x7FFFB with AM=0x2F.



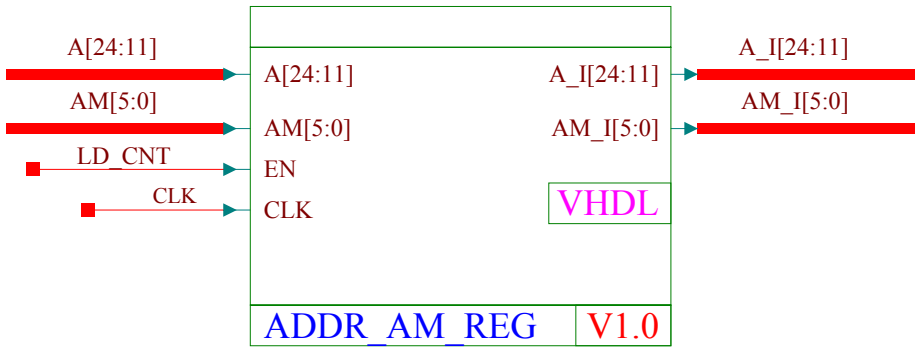
VME64X-PSB

VME64X_CHIP

Version: V100C	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	12-7-2005_14:55
checked by: CHECKER	0-00-0000_00:00



NLD_ADDR freezes addresses during VME cycle



A[24:1]

A_I[24:1]

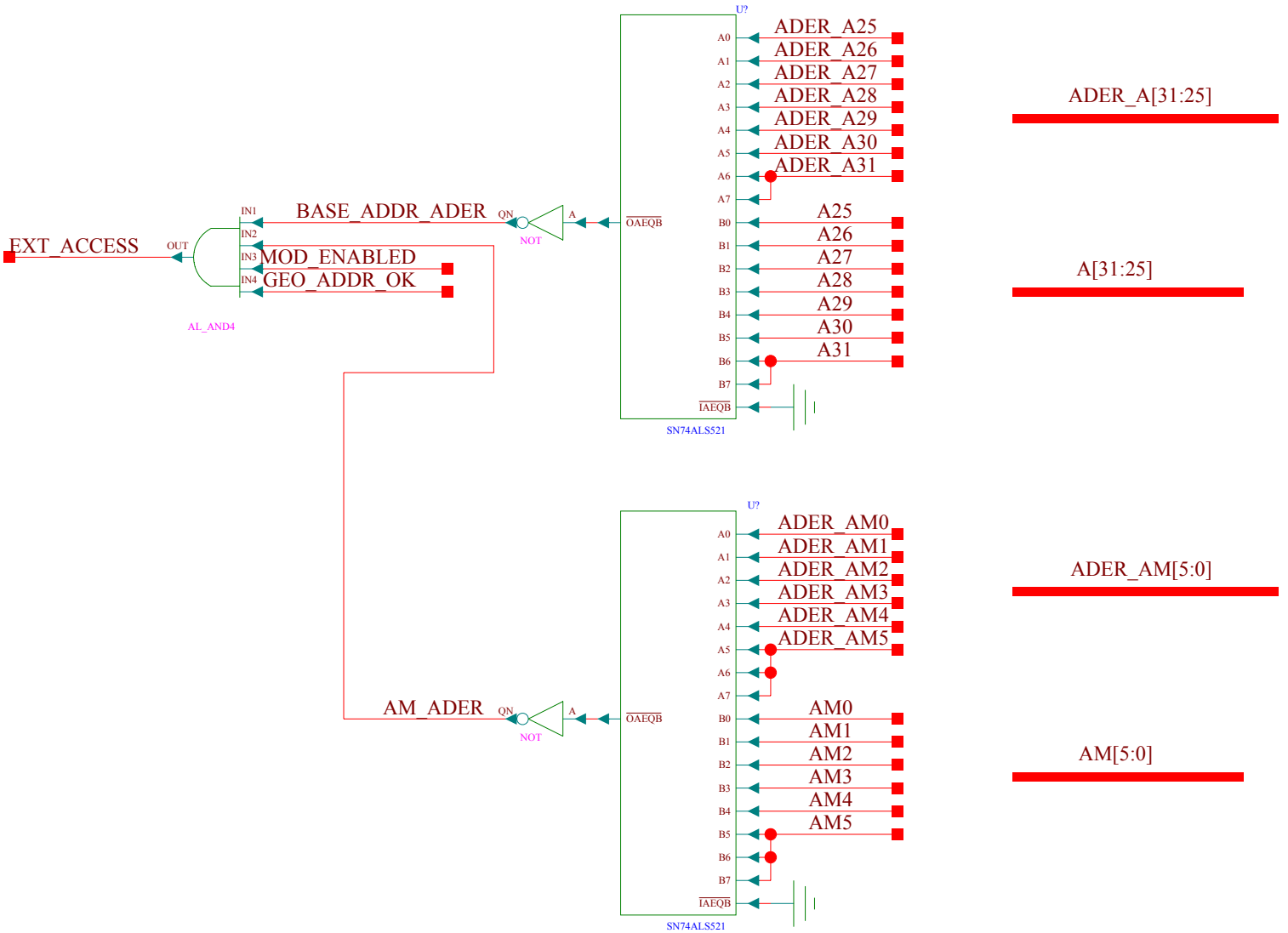
AM[5:0]

AM_I[5:0]

VME64X-CHIP

ADDRESSES_AM

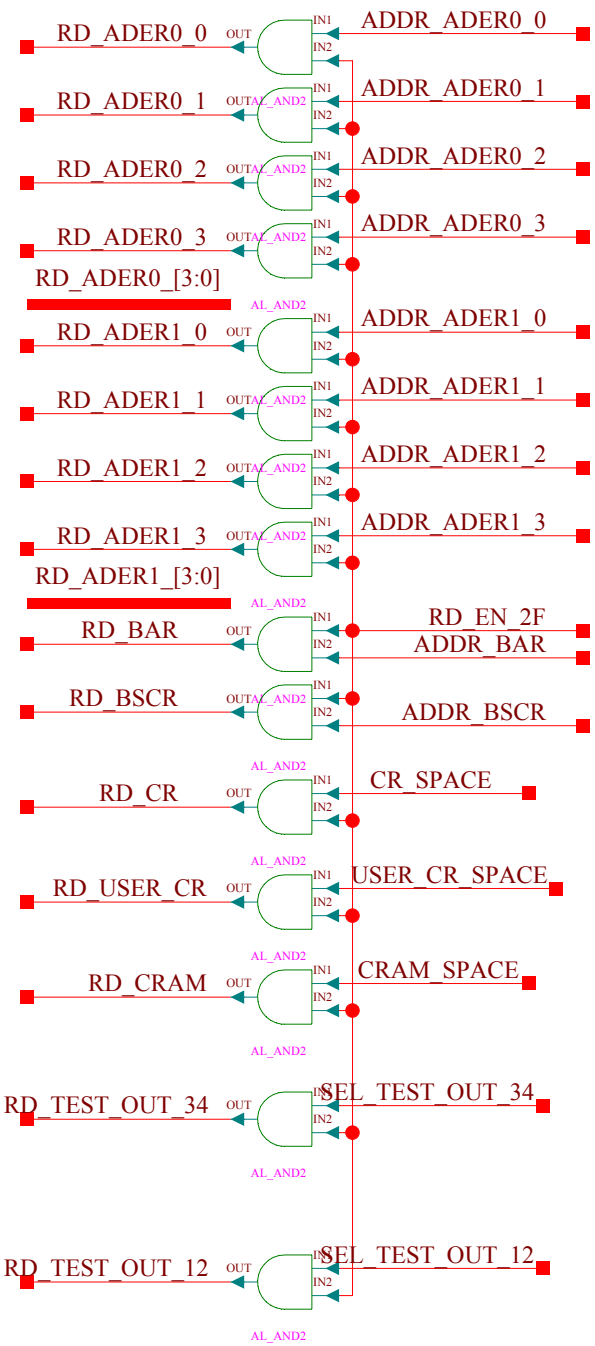
Version: V1.1	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	9-23-2005_10:45
checked by: CHECKER	0-00-0000_00:00

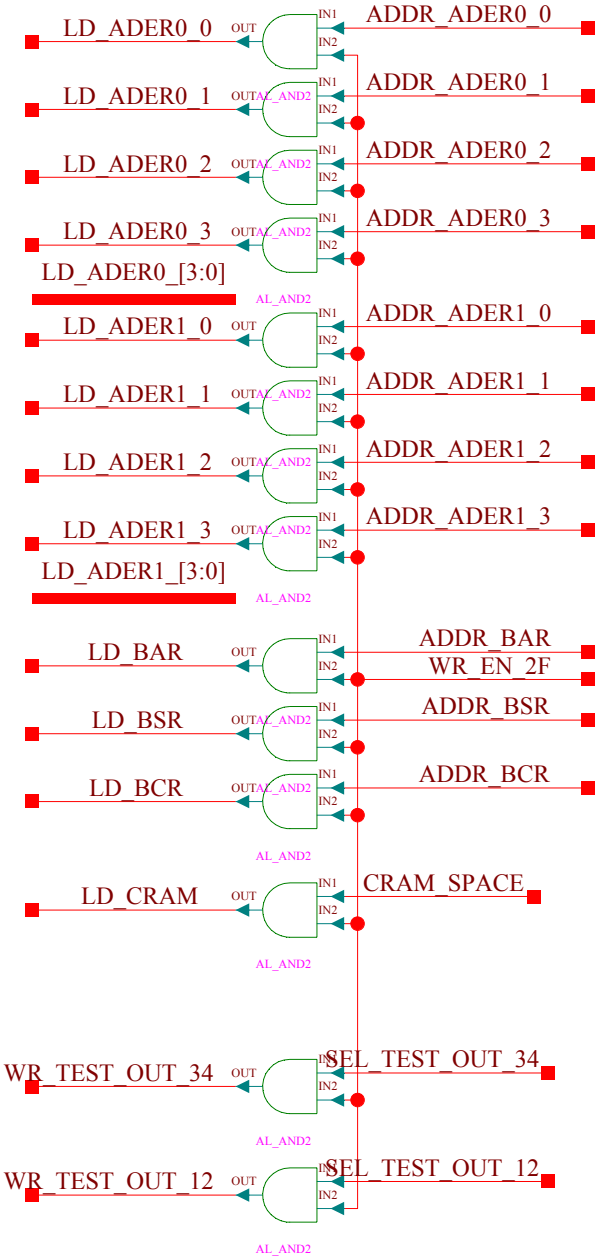


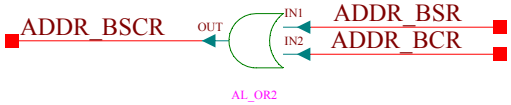
VME64X-CHIP

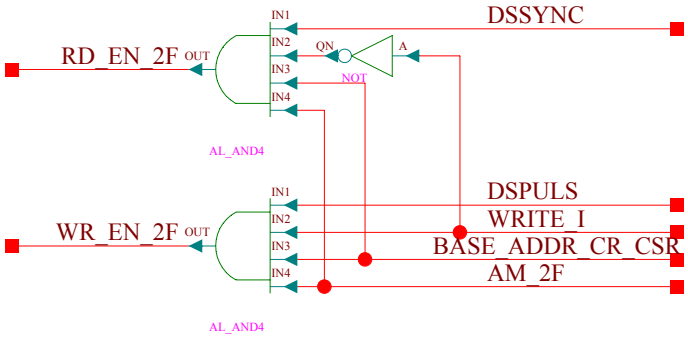
ADER_EXT

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





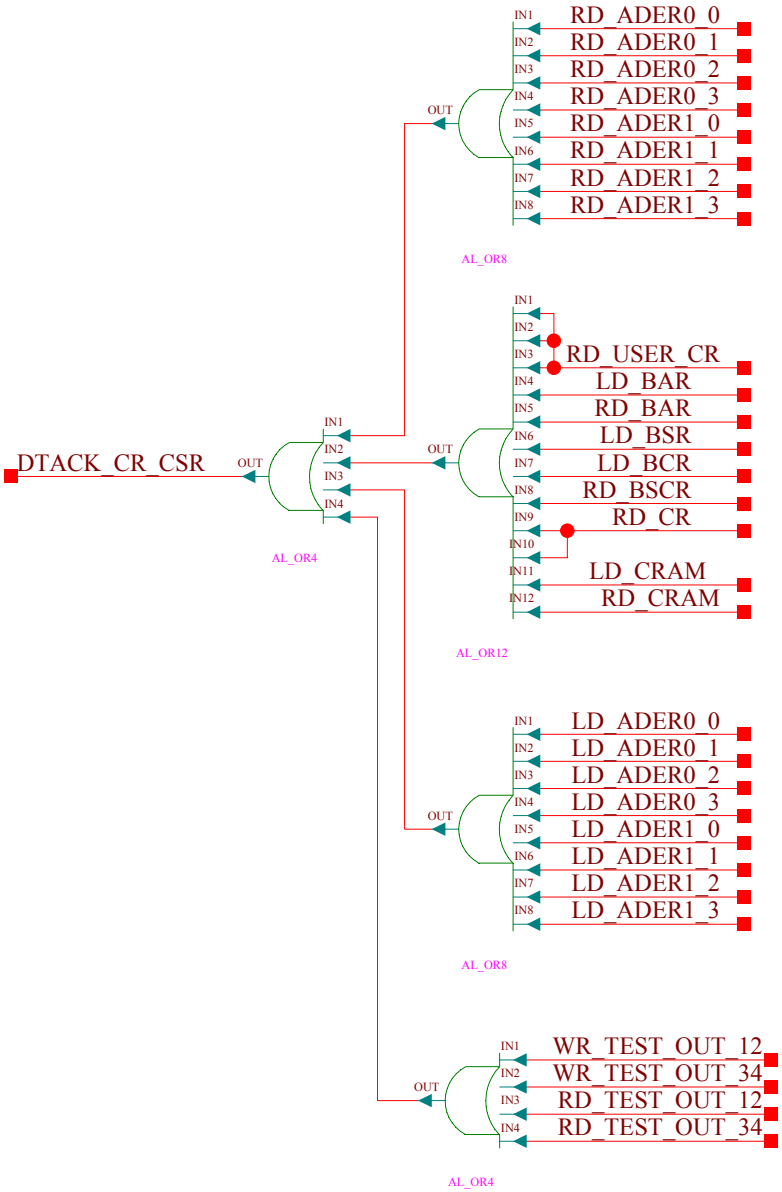




VME64X-CHIP

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

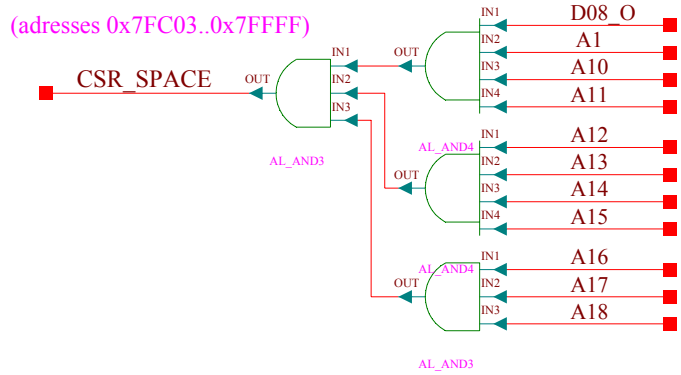


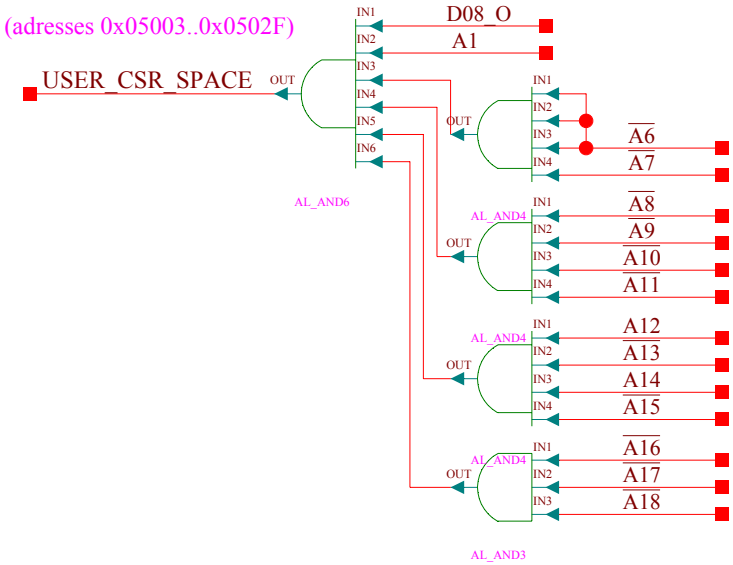
no BERR for CR/CSR access!!

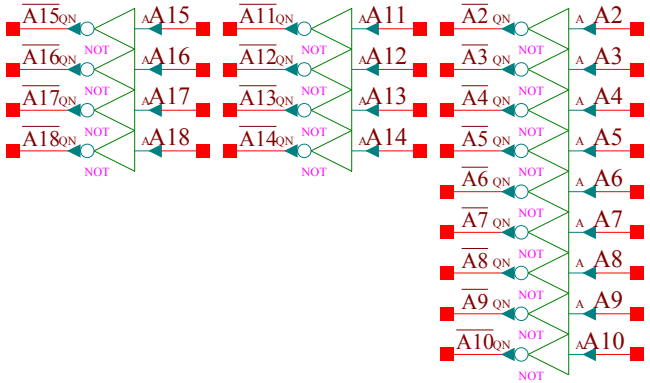
VME64X-CHIP

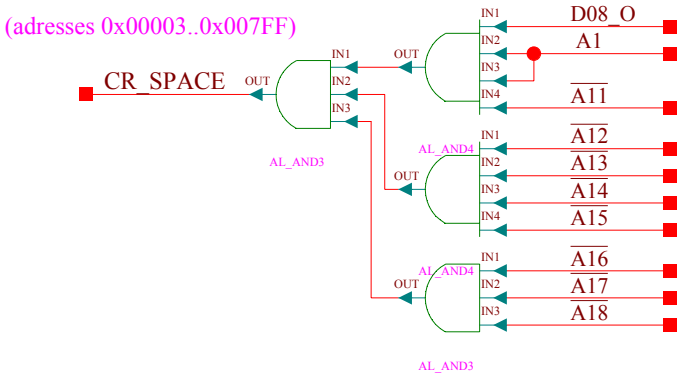
CR_CSR_INSTR

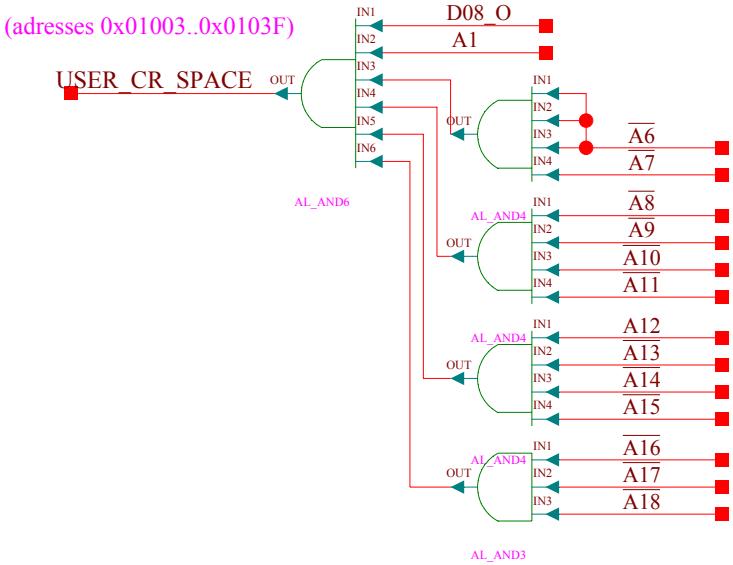
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00

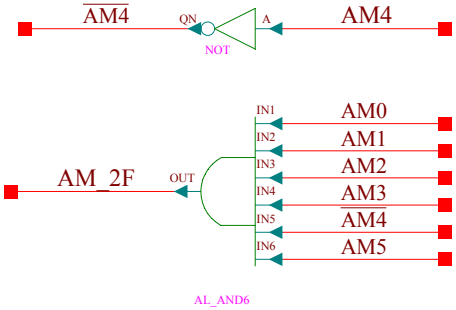


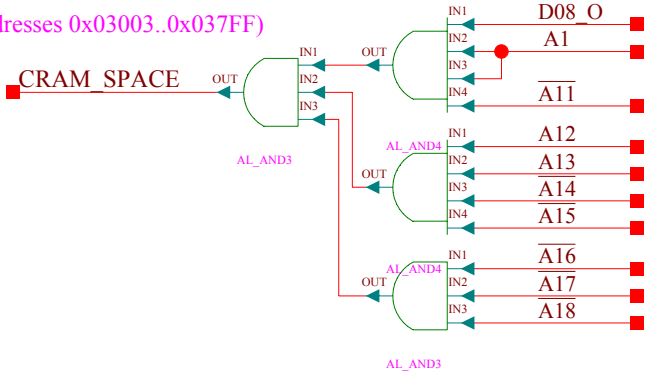












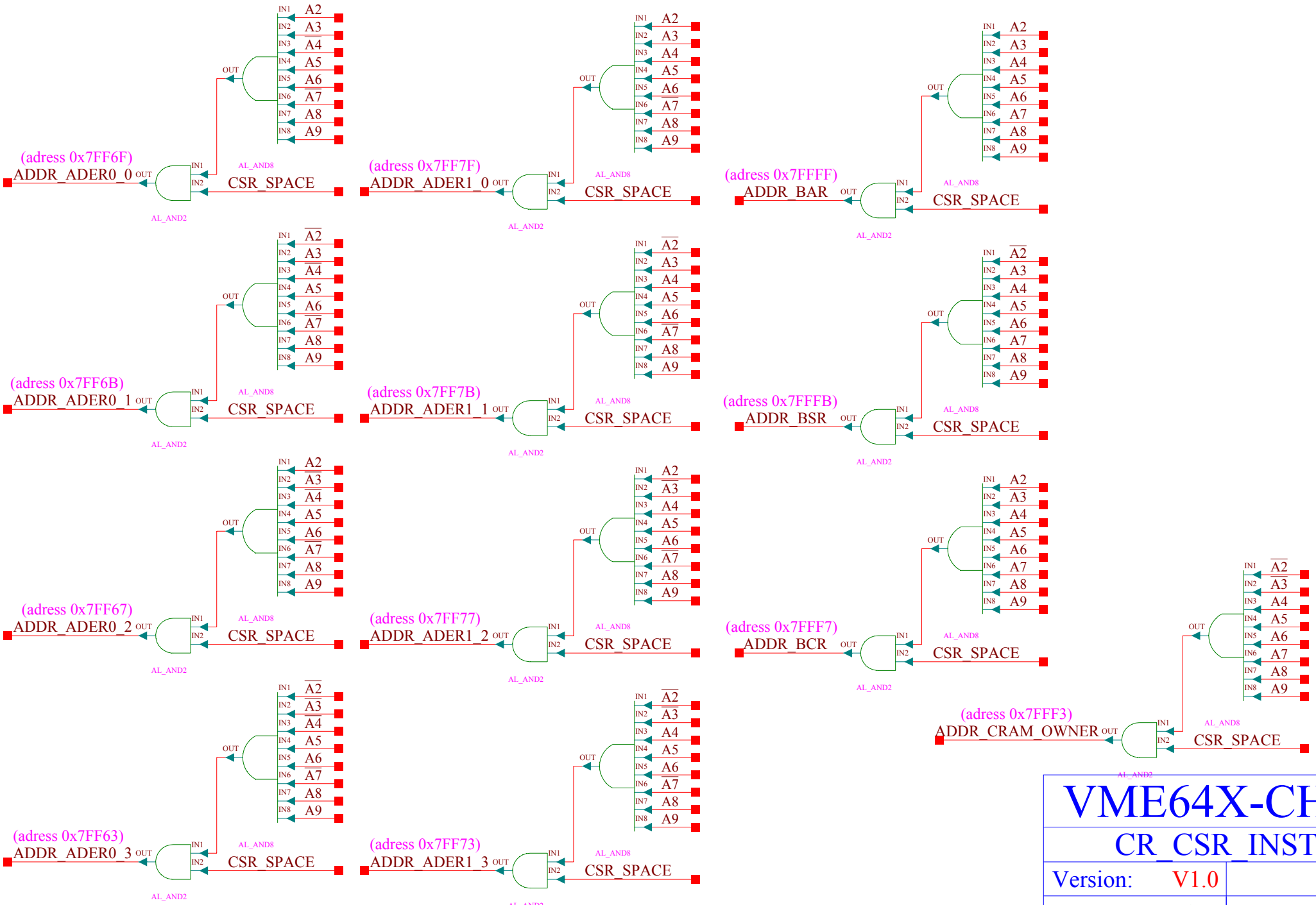
A[18:1]

AM[5:0]

VME64X-CHIP

CR_CSR_INSTR

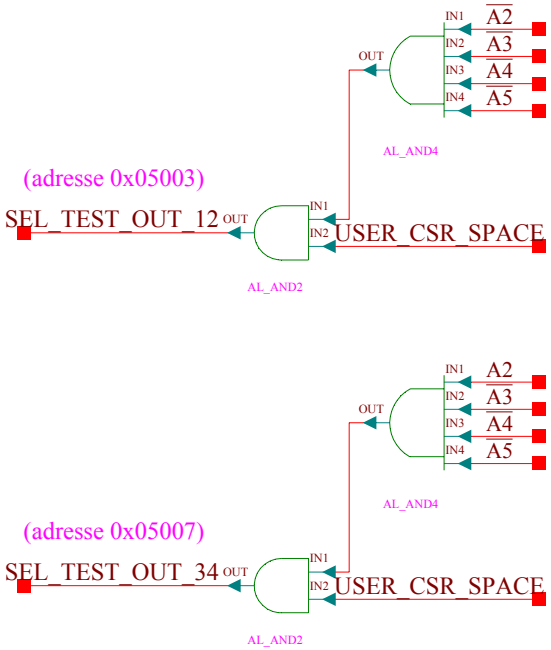
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 3 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP

CR_CSR_INSTR

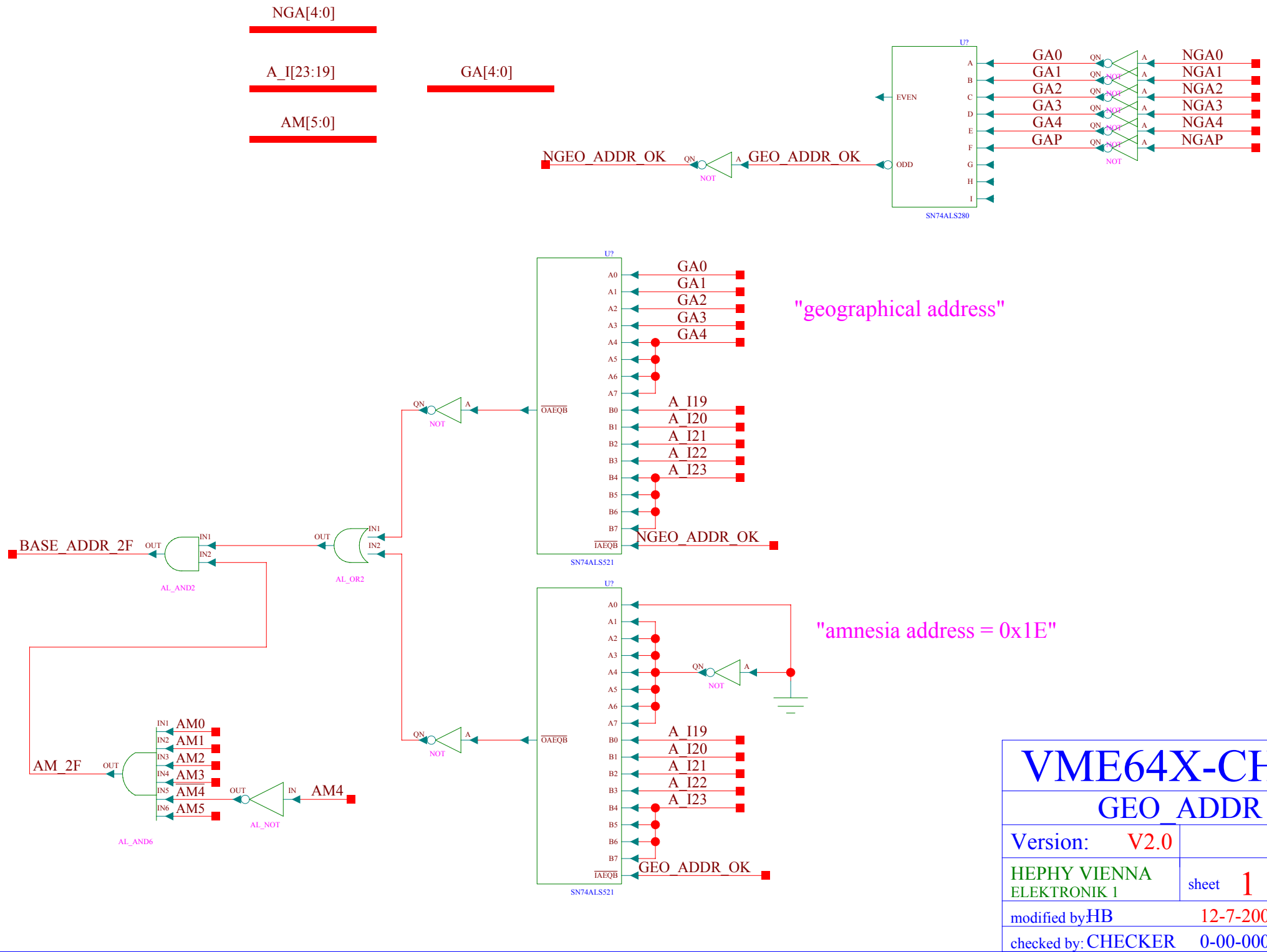
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 4 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00

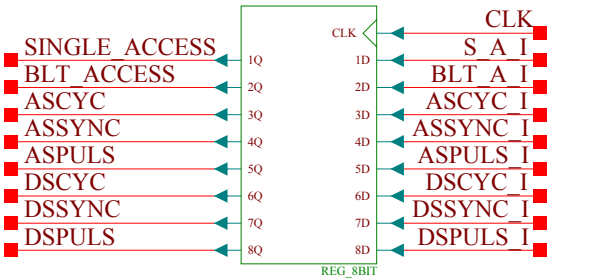


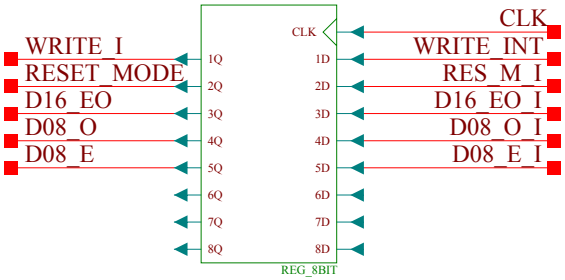
VME64X-CHIP

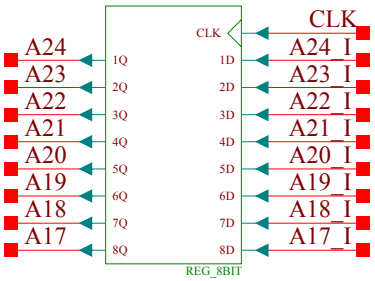
CR_CSR_INSTR

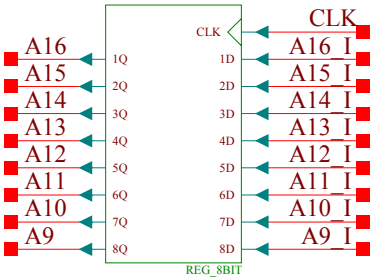
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 5 of 5
modified by: HB	8-26-2005_13:54
checked by: CHECKER	0-00-0000_00:00

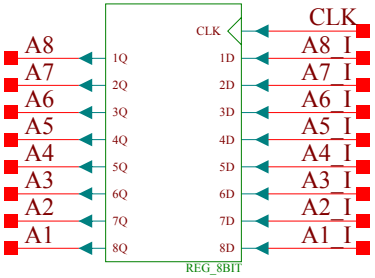

NGA[4:0]
A_I[23:19]
AM[5:0]
GA[4:0]
U?
EVEN
ODD
SN74ALS280
GA0
GA1
GA2
GA3
GA4
GAP
NGA0
NGA1
NGA2
NGA3
NGA4
NGAP
NGEO_ADDR_OK
GEO_ADDR_OK
NOT
"geographical address"
"amnesia address = 0x1E"
BASE_ADDR_2F
AL_AND2
AL_OR2
AM_2F
AL_AND6
AM0
AM1
AM2
AM3
AM4
AM5
AL_NOT
OAEQB
IAEQB
SN74ALS521
A_I19
A_I20
A_I21
A_I22
A_I23
VME64X-CHIP
GEO_ADDR
Version: V2.0
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 1
modified by: HB 12-7-2005_14:36
checked by: CHECKER 0-00-0000_00:00

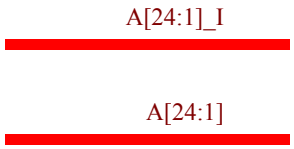


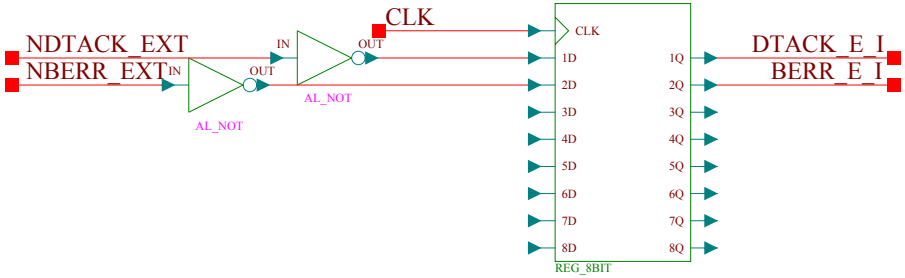








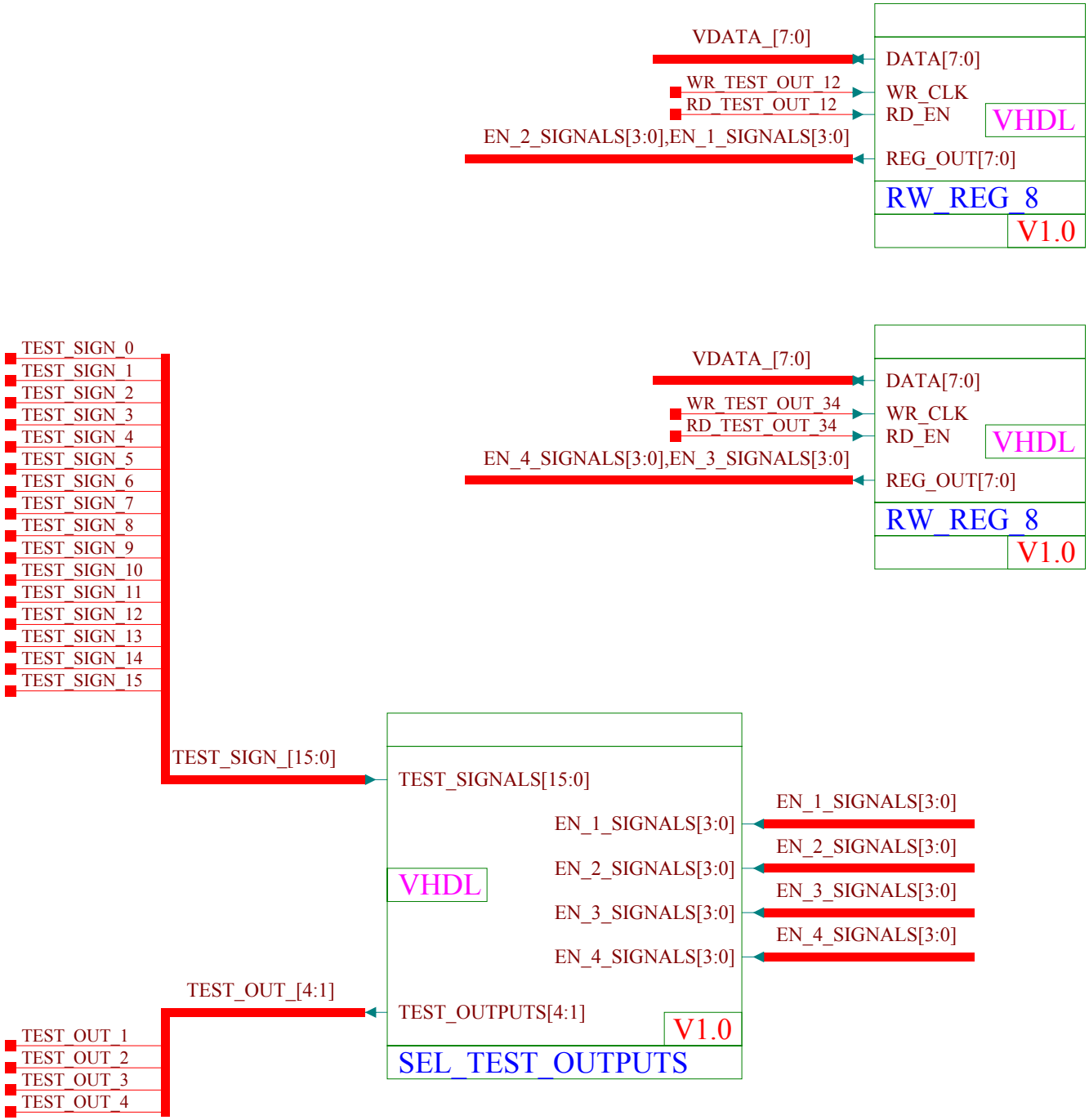




VME64X-CHIP

INTERN_IO

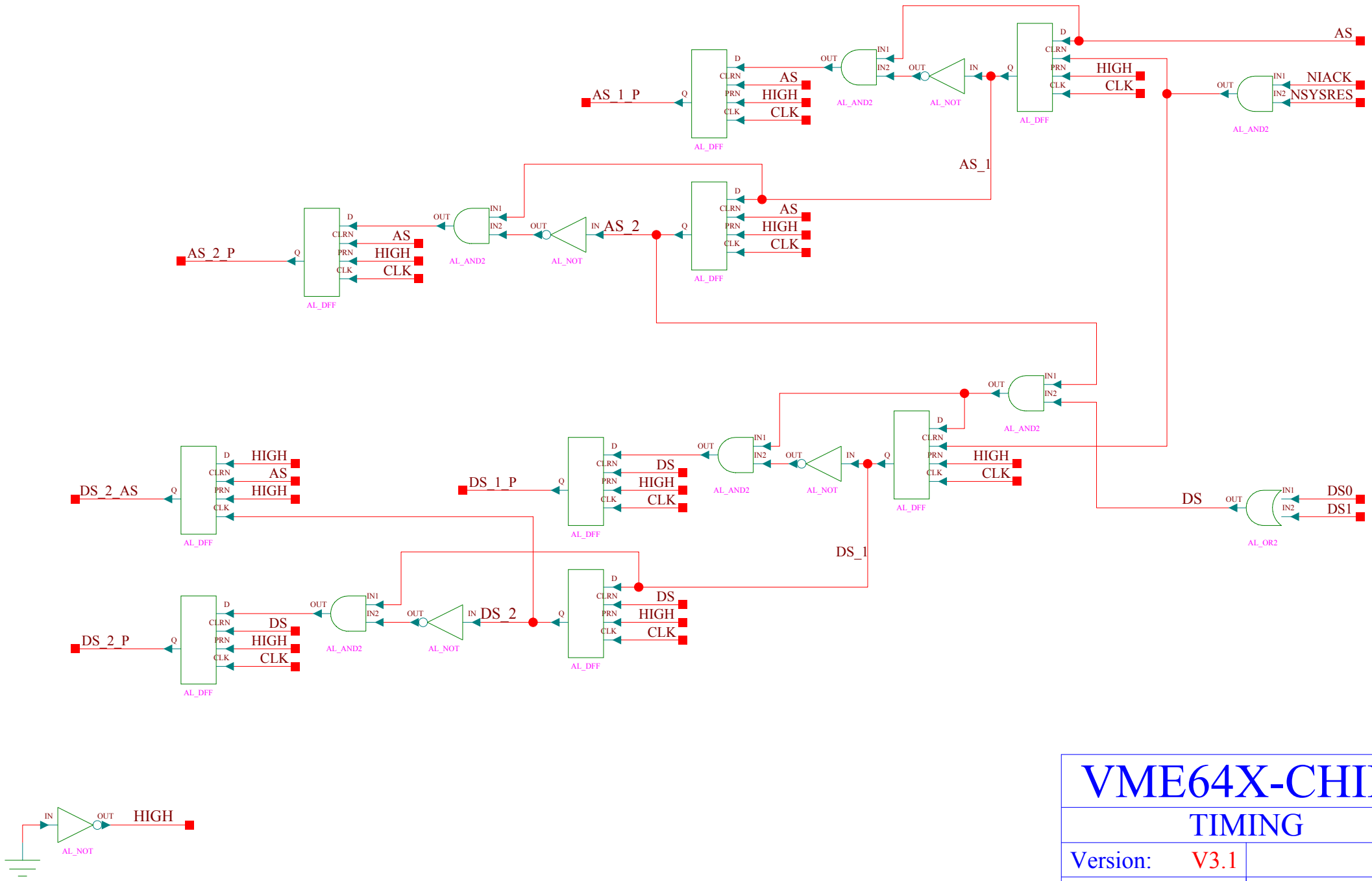
Version: V1.2	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	10-20-2005_9:17
checked by: CHECKER	0-00-0000_00:00



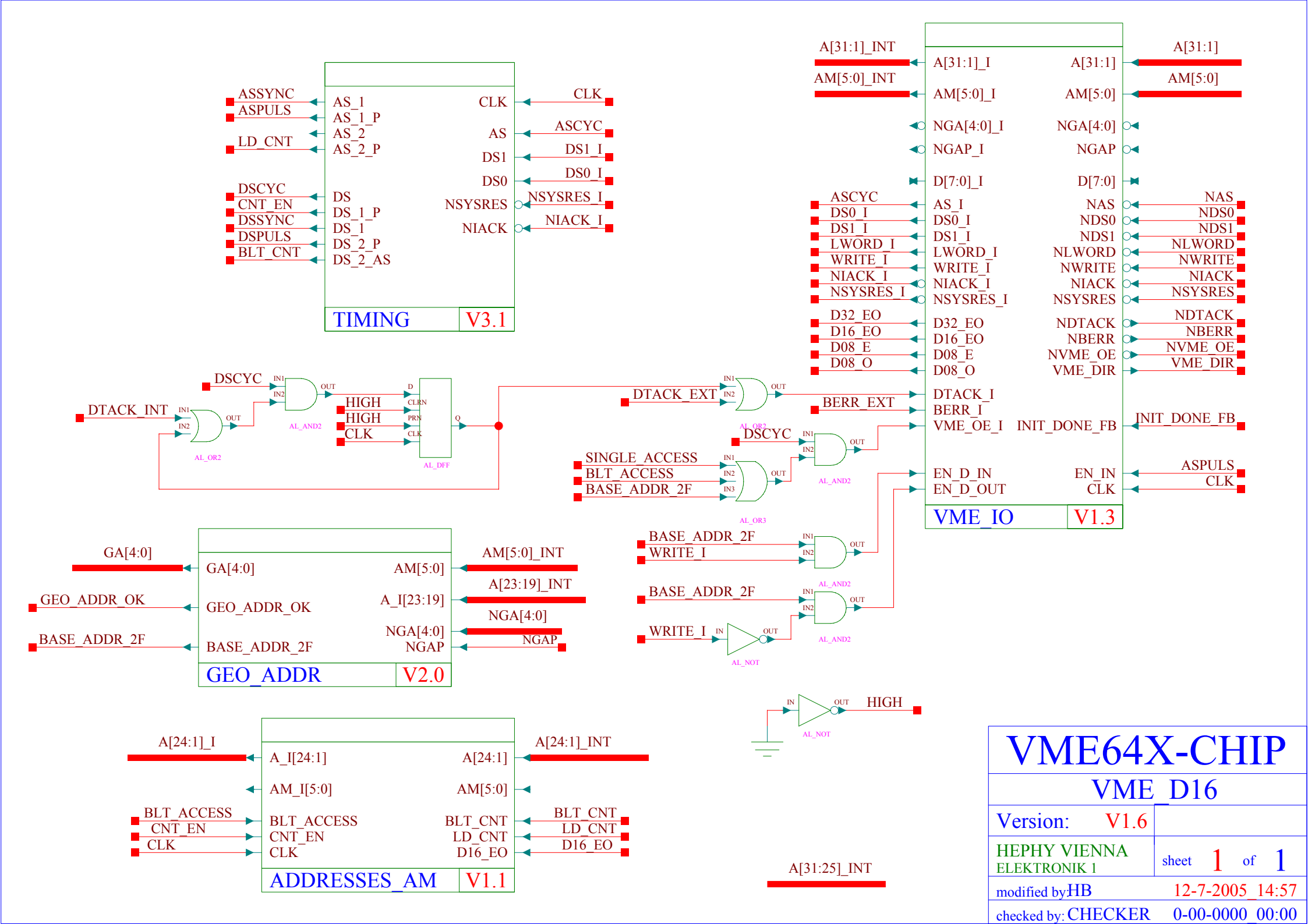
VME64X-CHIP

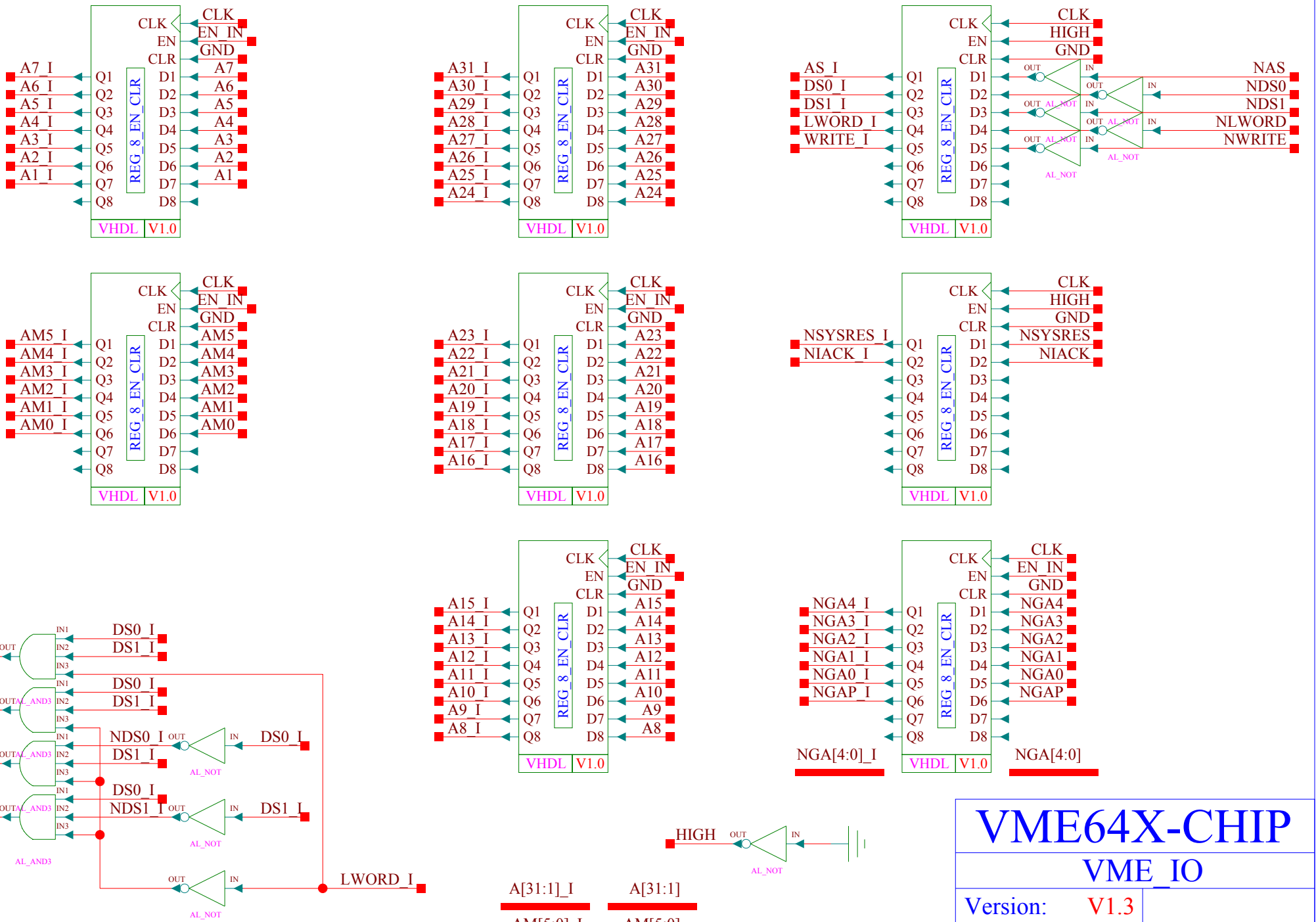
TEST_OUT

Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	8-29-2005_11:07
checked by: CHECKER	0-00-0000_00:00



VME64X-CHIP	
TIMING	
Version: V3.1	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	10-7-2005_14:18
checked by: CHECKER	0-00-0000_00:00

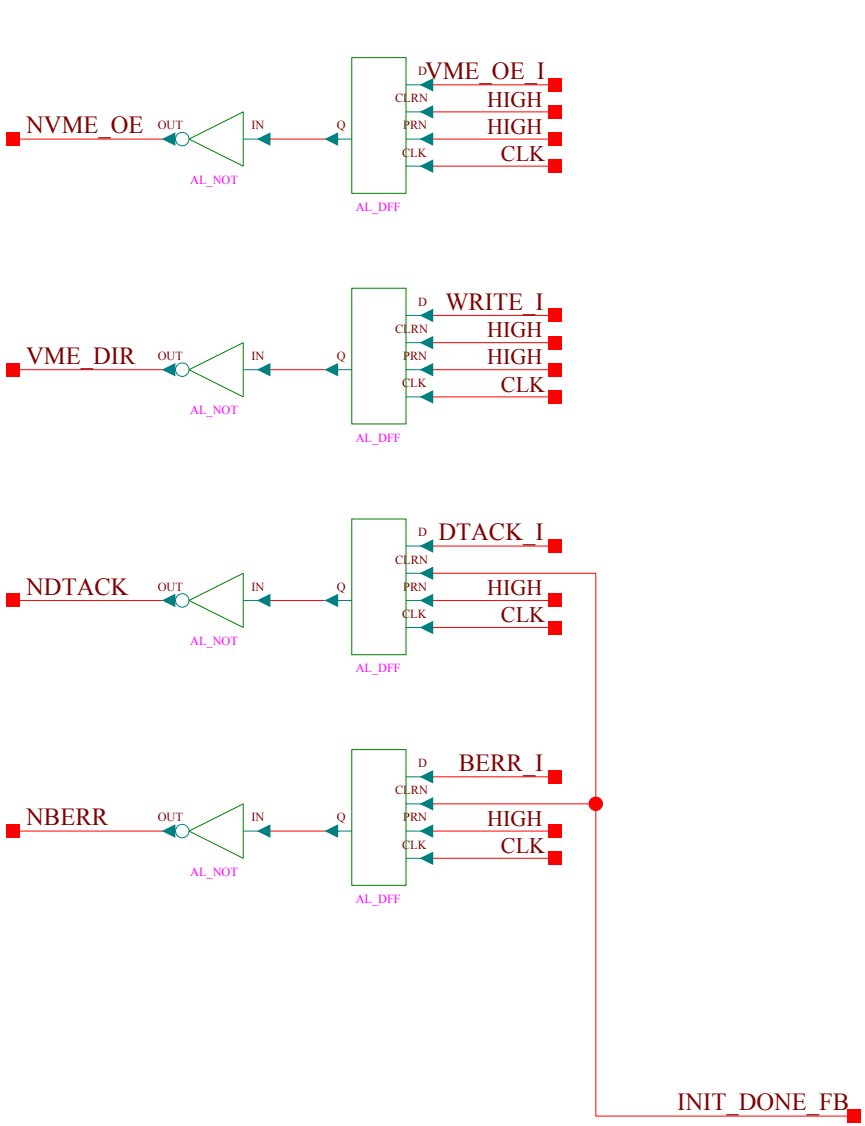

ASSYNC
ASPULS
LD_CNT
DSCYC
CNT_EN
DSSYNC
DSPULS
BLT_CNT
AS_1
AS_1_P
AS_2
AS_2_P
DS
DS_1_P
DS_1
DS_2_P
DS_2_AS
CLK
AS
DS1
DS0
NSYSRES
NIACK
CLK
ASCYC
DS1_I
DS0_I
NSYSRES_I
NIACK_I
TIMING
V3.1
A[31:1]_INT
AM[5:0]_INT
A[31:1]_I
AM[5:0]_I
NGA[4:0]_I
NGAP_I
D[7:0]_I
ASCYC
DS0_I
DS1_I
LWORD_I
WRITE_I
NIACK_I
NSYSRES_I
D32_EO
D16_EO
D08_E
D08_O
DTACK_I
BERR_I
VME_OE_I
EN_D_IN
EN_D_OUT
A[31:1]
AM[5:0]
NGA[4:0]
NGAP
AS_I
DS0_I
DS1_I
LWORD_I
WRITE_I
NIACK_I
NSYSRES_I
D32_EO
D16_EO
D08_E
D08_O
DTACK_I
BERR_I
VME_OE_I
EN_D_IN
EN_D_OUT
NAS
NDS0
NDS1
NLWORD
NWRITE
NIACK
NSYSRES
NDTACK
NBERR
NVME_OE
VME_DIR
INIT_DONE_FB
EN_IN
CLK
ASPULS
CLK
VME_IO
V1.3
DSCYC
DTACK_INT
HIGH
HIGH
CLK
AL_AND2
AL_OR2
AL_DFF
DTACK_EXT
BERR_EXT
AL_OR2
DSCYC
SINGLE_ACCESS
BLT_ACCESS
BASE_ADDR_2F
AL_AND2
AL_OR3
BASE_ADDR_2F
WRITE_I
AL_AND2
BASE_ADDR_2F
WRITE_I
AL_AND2
AL_NOT
GA[4:0]
GEO_ADDR_OK
BASE_ADDR_2F
GA[4:0]
GEO_ADDR_OK
BASE_ADDR_2F
AM[5:0]
A_I[23:19]
NGA[4:0]
NGAP
AM[5:0]_INT
A[23:19]_INT
NGA[4:0]
NGAP
GEO_ADDR
V2.0
HIGH
AL_NOT
A[24:1]_I
AM_I[5:0]
BLT_ACCESS
CNT_EN
CLK
A_I[24:1]
AM_I[5:0]
BLT_ACCESS
CNT_EN
CLK
A[24:1]
AM[5:0]
BLT_CNT
LD_CNT
D16_EO
A[24:1]_INT
BLT_CNT
LD_CNT
D16_EO
ADDRESSES_AM
V1.1
A[31:25]_INT
VME64X-CHIP
VME_D16
Version: V1.6
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 1
modified by: HB
12-7-2005_14:57
checked by: CHECKER
0-00-0000_00:00

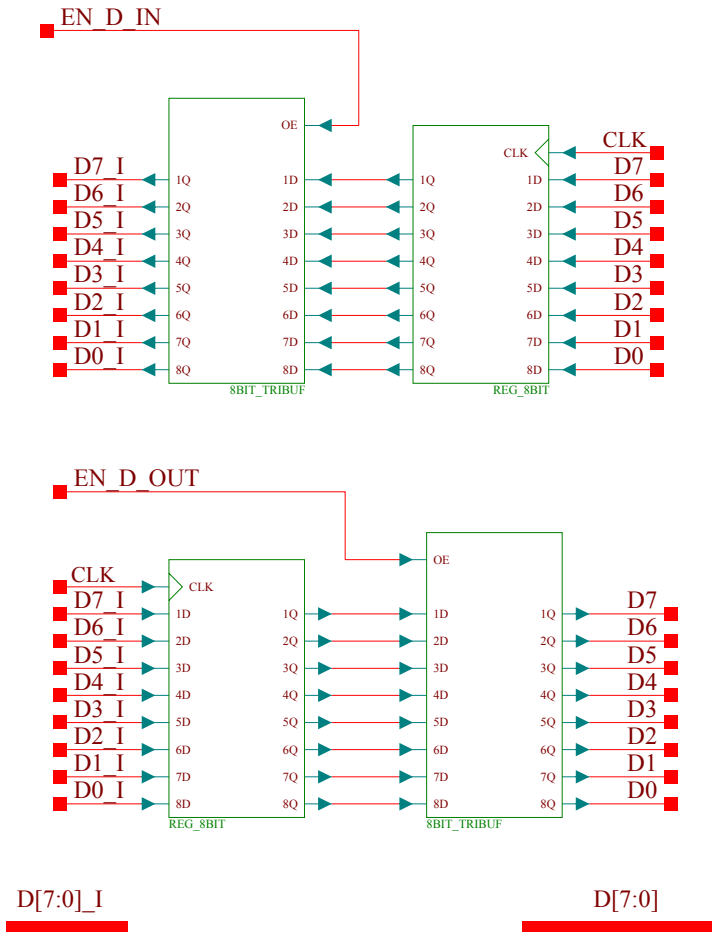


VME64X-CHIP

VME_IO

Version: V1.3	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 2
modified by: HB	10-21-2005_14:41
checked by: CHECKER	0-00-0000_00:00



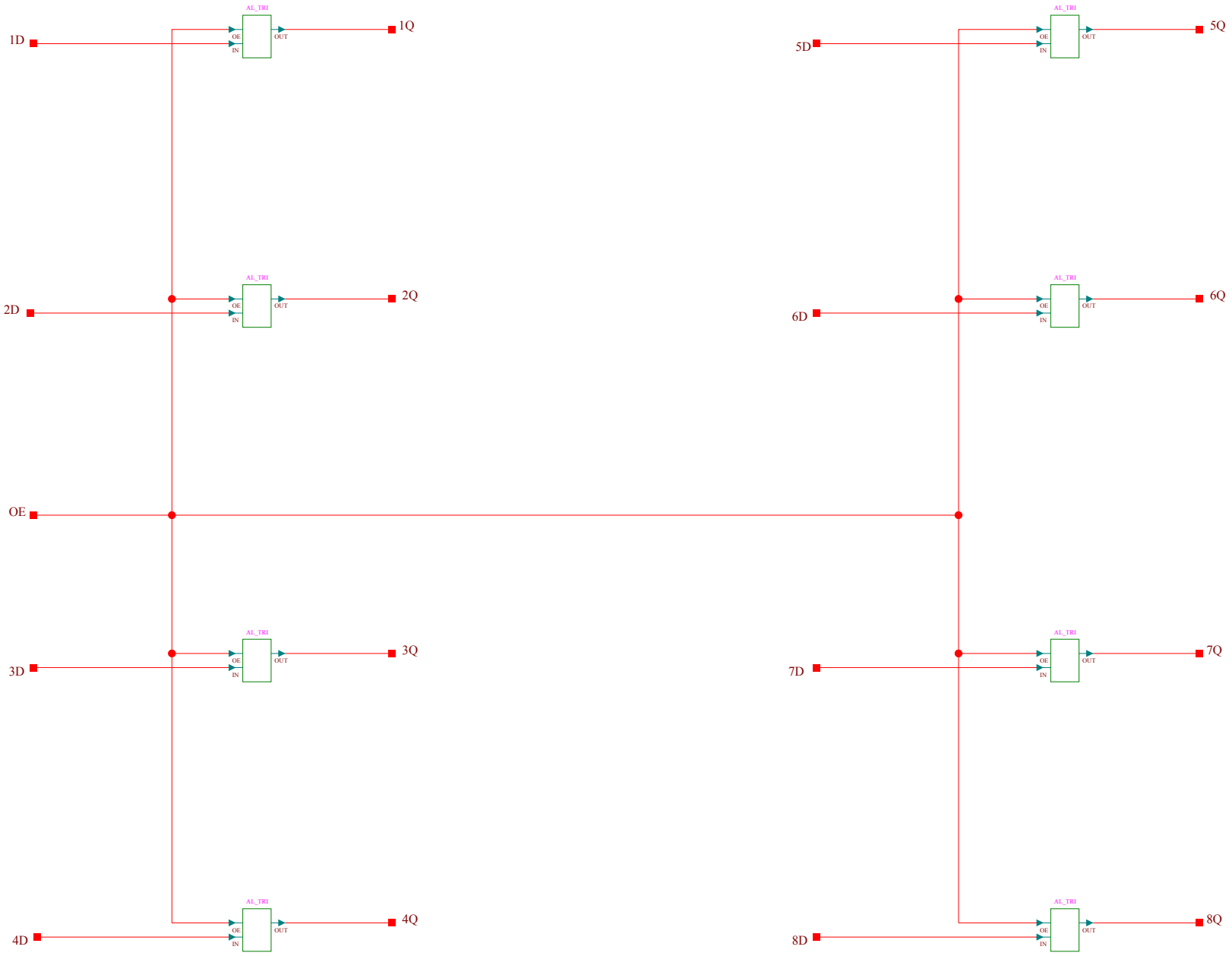


VME64X-CHIP

VME_IO

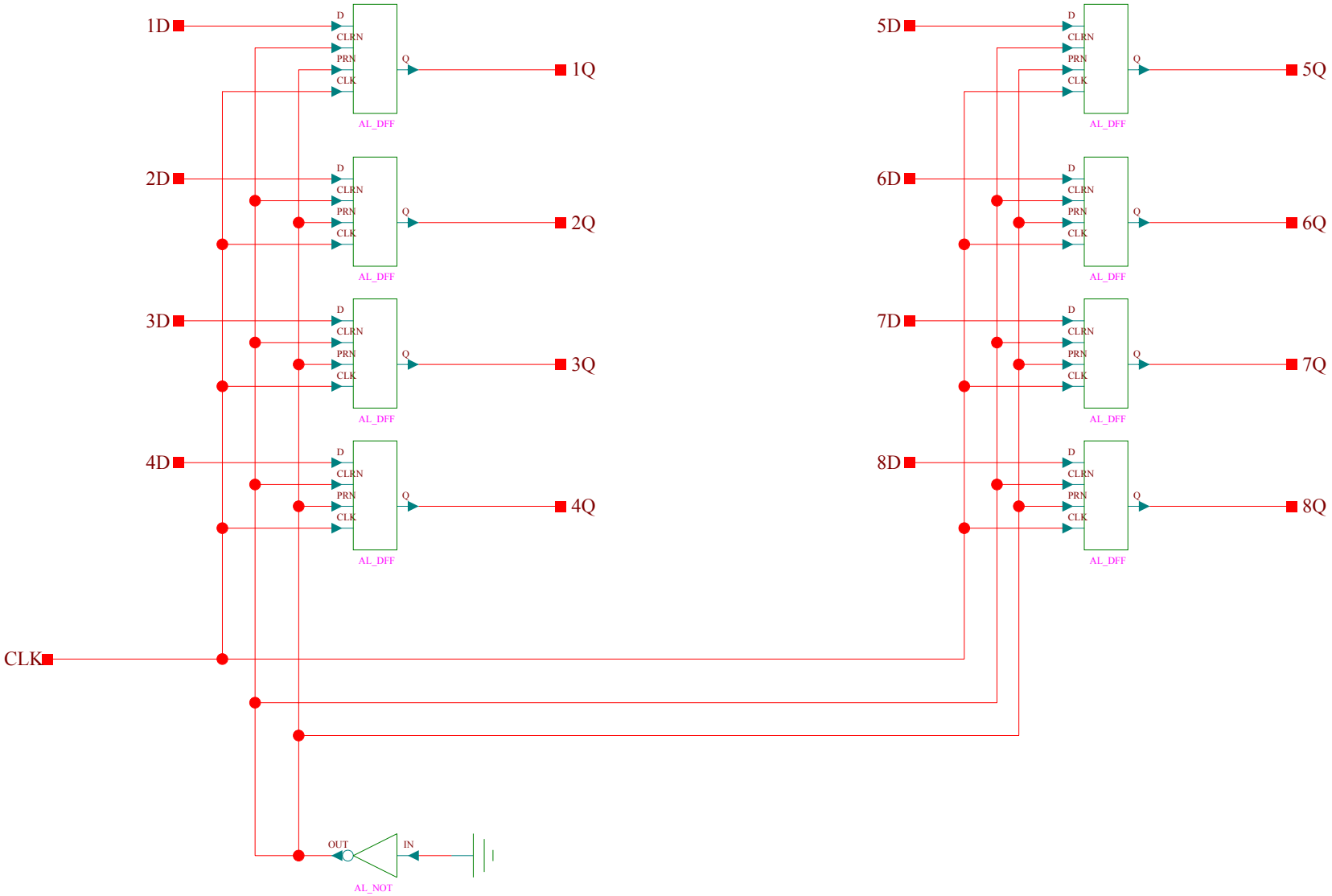
Version: V1.3	
HEPHY VIENNA ELEKTRONIK 1	sheet 2 of 2
modified by: HB	10-21-2005_14:41
checked by: CHECKER	0-00-0000_00:00

8bit_tribuf



reg_8bit

8-bit register generated by LAB3



```
------------------------------------------------------------------
--                                                              --
-- LOGIC CORE: GTL-module vme64x interface chip logic           --
-- MODULE NAME: addr_am_reg                                     --
-- INSTITUTION: Hephy Vienna                                    --
-- DESIGNER: H. Bergauer                                        --
--                                                              --
-- VERSION: V1.0                                                --
-- DATE: 08 2005                                                --
--                                                              --
-- FUNCTIONAL DESCRIPTION:                                      --
-- input register for addresses and address modifier            --
--                                                              --
------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY addr am reg IS
    PORT(
        clk : IN    STD_LOGIC;
        en  : IN    STD LOGIC;
        a   : IN    STD_LOGIC_VECTOR(24 DOWNTO 11);
        am  : IN    STD_LOGIC_VECTOR(5 DOWNTO 0);
        a i : OUT   STD LOGIC VECTOR(24 DOWNTO 11);
        am_i    : OUT   STD_LOGIC_VECTOR(5 DOWNTO 0));
END addr_am_reg;

ARCHITECTURE rtl OF addr_am_reg IS
BEGIN
 
-- input register for addresses 
    addr_reg: lpm_ff
  GENERIC MAP   (LPM_WIDTH => 14)
  PORT MAP  (data => a, 
   clock => clk,
   enable => en,
   q => a_i);

-- input register for address modifier  
    am_reg: lpm_ff
  GENERIC MAP   (LPM_WIDTH => 6)
  PORT MAP  (data => am, 
   clock => clk,
   enable => en,
   q => am_i);

END ARCHITECTURE rtl;
```

```
------------------------------------------------------------------
--                                                              --
-- LOGIC CORE: GTL-module vme64x interface chip logic           --
-- MODULE NAME: addr_cnt                                        --
-- INSTITUTION: Hephy Vienna                                    --
-- DESIGNER: H. Bergauer                                        --
--                                                              --
-- VERSION: V1.0                                                --
-- DATE: 08 2005                                                --
--                                                              --
-- FUNCTIONAL DESCRIPTION:                                      --
-- addresses counter for BLT                                    --
--                                                              --
------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY addr_cnt IS
    PORT(
        clk : IN    STD_LOGIC;
        cnt_en  : IN    STD_LOGIC;
        sclr    : IN    STD_LOGIC;
        aload   : IN    STD_LOGIC;
        sload   : IN    STD_LOGIC;
        ld_data : IN    STD_LOGIC_VECTOR(10 DOWNTO 1);
        out_data    : OUT   STD_LOGIC_VECTOR(10 DOWNTO 1));
END addr_cnt;

ARCHITECTURE rtl OF addr_cnt IS
BEGIN
-- default of updown is UP
-- this is an up-counter

inst_cnt: lpm_counter
   GENERIC MAP(LPM_WIDTH => 10,
      LPM_TYPE => "LPM_COUNTER")
   PORT MAP(data => ld_data,
        clock => clk,
        cnt_en => cnt_en,
        sclr => sclr,
        aload => aload,
        sload => sload,
        q => out_data);

END ARCHITECTURE rtl;
```

```
------------------------------------------------------------------
--                                                              --
-- 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;
```

```
------------------------------------------------------------
--                                                        --
-- LOGIC CORE: GT-logic                                   --
-- MODULE NAME: reg_8_en_clr                              --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--                                                        --
-- VERSION: V1.0                                          --
-- DATE: 09 2005                                          --
--                                                        --
-- FUNCTIONAL DESCRIPTION:                                --
-- register (8 bit) with enable and clr                   --
--                                                        --
------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY reg_8_en_clr IS
    PORT(
        d1  : IN    STD_LOGIC;
        d2  : IN    STD_LOGIC;
        d3  : IN    STD_LOGIC;
        d4  : IN    STD_LOGIC;
        d5  : IN    STD_LOGIC;
        d6  : IN    STD_LOGIC;
        d7  : IN    STD_LOGIC;
        d8  : IN    STD_LOGIC;
        clk : IN    STD_LOGIC;
        en  : IN    STD_LOGIC;
        clr : IN    STD_LOGIC;
        q1  : OUT   STD_LOGIC;
        q2  : OUT   STD_LOGIC;
        q3  : OUT   STD_LOGIC;
        q4  : OUT   STD_LOGIC;
        q5  : OUT   STD_LOGIC;
        q6  : OUT   STD_LOGIC;
        q7  : OUT   STD_LOGIC;
        q8  : OUT   STD_LOGIC);
END reg_8_en_clr;

ARCHITECTURE rtl OF reg_8_en_clr IS
    SIGNAL d : STD_LOGIC_VECTOR(8 DOWNTO 1);
    SIGNAL q : STD_LOGIC_VECTOR(8 DOWNTO 1);
BEGIN

d <= d8 & d7 & d6 & d5 & d4 & d3 & d2 & d1;
q1 <= q(1);
q2 <= q(2);
q3 <= q(3);
q4 <= q(4);
q5 <= q(5);
q6 <= q(6);
q7 <= q(7);
q8 <= q(8);

reg: lpm_ff
  GENERIC MAP(LPM_WIDTH => 8)
    PORT MAP(data => d, clock => clk, enable => en,
             aclr => clr, q => q);

END ARCHITECTURE rtl;
```

```
------------------------------------------------------------
--                                                        --
-- LOGIC CORE: GT-logic                                   --
-- MODULE NAME: rw_reg_8                                  --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--                                                        --
-- VERSION: V1.0                                          --
-- DATE: 08 2005                                          --
--                                                        --
-- FUNCTIONAL DESCRIPTION:                                --
-- read/write register (8 bit)                            --
--                                                        --
------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;
LIBRARY altera;
USE altera.maxplus2.ALL;

ENTITY rw_reg_8 IS
    PORT(
        data    : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        wr_clk  : IN    STD_LOGIC;
        rd_en   : IN    STD_LOGIC;
        reg_out : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0));
END rw_reg_8;

ARCHITECTURE rtl OF rw_reg_8 IS
    SIGNAL data_int : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

-- write register

write_reg: lpm_ff
  GENERIC MAP(LPM_WIDTH => 8)
    PORT MAP(data, wr_clk, q => data_int);

reg_out <= data_int;

-- read register (tri-state outputs to VME)
tri_stat_out:
FOR i IN 0 TO 7 GENERATE
 call_tri: tri
   PORT MAP(data_int(i), rd_en, data(i));
END GENERATE tri_stat_out;

END ARCHITECTURE rtl;
```

```
------------------------------------------------------------
--                                                        --
-- LOGIC CORE: GT logic                                   --
-- MODULE NAME: sel_test_outputs                          --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--                                                        --
-- VERSION: V1.0                                          --
-- DATE: 08 2005                                          --
--                                                        --
-- FUNCTIONAL DESCRIPTION:                                --
-- selection of 1 of 16 signals for                       --
-- 4 test_out-pins (scope)                                --
--                                                        --
------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY sel_test_outputs IS
    PORT(
--      test_signals_0      : IN    STD_LOGIC;
--      test_signals_1      : IN    STD_LOGIC;
--      test_signals_2      : IN    STD_LOGIC;
--      test_signals_3      : IN    STD_LOGIC;
--      test_signals_4      : IN    STD_LOGIC;
--      test_signals_5      : IN    STD_LOGIC;
--      test_signals_6      : IN    STD_LOGIC;
--      test_signals_7      : IN    STD_LOGIC;
--      test_signals_8      : IN    STD_LOGIC;
--      test_signals_9      : IN    STD_LOGIC;
--      test_signals_10     : IN    STD_LOGIC;
--      test_signals_11     : IN    STD_LOGIC;
--      test_signals_12     : IN    STD_LOGIC;
--      test_signals_13     : IN    STD_LOGIC;
--      test_signals_14     : IN    STD_LOGIC;
--      test_signals_15     : IN    STD_LOGIC;
        test_signals        : IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
        en_1_signals        : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        en_2_signals        : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        en_3_signals        : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        en_4_signals        : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        test_outputs        : OUT   STD_LOGIC_VECTOR(4 DOWNTO 1)
        );
END sel_test_outputs;

ARCHITECTURE rtl OF sel_test_outputs IS
--COMPONENT test_out_coded IS
--  PORT(
--      en_signals          : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
--      test_signals        : IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
--      test_out            : OUT   STD_LOGIC
--      );
--END COMPONENT test_out_coded;

--  SIGNAL test_signals_vec : STD_LOGIC_VECTOR(15 DOWNTO 0);

    TYPE en_signals_type IS ARRAY (4 DOWNTO 1)
        OF STD_LOGIC_VECTOR(3 DOWNTO 0);

    SIGNAL en_signals_arr : en_signals_type;
BEGIN
-- *****************************************************************
-- ERKLÄRUNG:
-- en_signals sind codiert, 4 bits aus VME-registers (2x8 bits für
-- 4 test_outputs mit je 16 test-signals), die angeben,
-- welches interne signal auf test-output gelegt wird.
-- test_signals ist der vector, der die internen signals enthält.
-- Korrespondierend mit dem value der en_signals wird das jeweilige
```

```
-- interne signal auf den test-output gelegt. Definition des internen
-- signals in vector notwendig!!!
-- ****************************************************************

en_signals_arr(4) <= en_4_signals;
en_signals_arr(3) <= en_3_signals;
en_signals_arr(2) <= en_2_signals;
en signals arr(1) <= en 1 signals;

--test_signals_vec <=
--  test signals 15 & test signals 14 & test signals 13 & test signals 12 &
--  test_signals_11 & test_signals_10 & test_signals_9 & test_signals_8 &
--  test_signals_7 & test_signals_6 & test_signals_5 & test_signals_4 &
--  test signals 3 & test signals 2 & test signals 1 & test signals 0;

loop_test_outputs:
    for i in 1 to 4 generate
        test_outputs(i) <= test_signals(CONV_INTEGER(en_signals_arr(i)));
    end generate loop_test_outputs;

--loop_test_outputs:
--for i in 1 to 4 generate
--  call test out: test out coded
--      PORT MAP(en_signals_arr(i), test_signals_vec, test_outputs(i));
--end generate loop_test_outputs;

END rtl;
```

```
------------------------------------------------------------
--                                                        --
-- LOGIC CORE: GTL-module vme64x interface chip logic     --
-- MODULE NAME: user_cr                                   --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--                                                        --
-- VERSION: V2.2                                          --
-- 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                     --
-- V2.2: CARD_NR from S24-S27 jumpers used for            --
--       serial number of PSB-cards too                   --
--                                                        --
------------------------------------------------------------
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_sel : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL addr_mem : std_logic;
    SIGNAL addr_card_nr : std_logic;
    SIGNAL addr_ser_nr_1 : std_logic;
    SIGNAL addr_ser_nr_2 : std_logic;
    SIGNAL data_mem : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_cnr : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_snr_1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_snr_2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_tri : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_cnr_tri : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_snr_1_tri : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL data_snr_2_tri : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

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

-- serial_nr in ASCII, first character !!! 0x30 + Card_nr !!!
WITH card_nr SELECT
data_snr_1 <=
        X"31" WHEN X"0",
        X"30" WHEN X"1",
        X"30" WHEN X"2",
        X"30" WHEN X"3",
```

```
        X"30" WHEN X"4",
        X"30" WHEN X"5",
        X"30" WHEN X"6",
        X"30" WHEN X"7",
        X"30" WHEN X"8",
        X"30" WHEN X"9",
        X"31" WHEN X"A",
        X"31" WHEN X"B",
        X"31" WHEN X"C",
        X"31" WHEN X"D",
        X"31" WHEN X"E",
        X"31" WHEN X"F",
        X"00" WHEN OTHERS;

WITH card_nr SELECT
data_snr_2 <=
        X"36" WHEN X"0",
        X"31" WHEN X"1",
        X"32" WHEN X"2",
        X"33" WHEN X"3",
        X"34" WHEN X"4",
        X"35" WHEN X"5",
        X"36" WHEN X"6",
        X"37" WHEN X"7",
        X"38" WHEN X"8",
        X"39" WHEN X"9",
        X"30" WHEN X"A",
        X"31" WHEN X"B",
        X"32" WHEN X"C",
        X"33" WHEN X"D",
        X"34" WHEN X"E",
        X"35" WHEN X"F",
        X"00" WHEN OTHERS;

WITH addr SELECT
addr_sel <=
        X"1" WHEN X"0",
        X"1" WHEN X"1",
        X"2" WHEN X"2",
        X"1" WHEN X"3",
        X"1" WHEN X"4",
        X"1" WHEN X"5",
        X"1" WHEN X"6",
        X"1" WHEN X"7",
        X"1" WHEN X"8",
        X"1" WHEN X"9",
        X"1" WHEN X"A",
        X"4" WHEN X"B",
        X"8" WHEN X"C",
        X"1" WHEN X"D",
        X"1" WHEN X"E",
        X"1" WHEN X"F",
        X"0" WHEN OTHERS;

-- addresses for card_nr and serial_nr
addr_mem <= addr_sel(0) AND rd_en;
addr_card_nr <= addr_sel(1) AND rd_en;
addr_ser_nr_1 <= addr_sel(2) AND rd_en;
addr_ser_nr_2 <= addr_sel(3) 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_WIDTHAD => 4,
   LPM_OUTDATA => "UNREGISTERED",
   LPM_ADDRESS_CONTROL => "UNREGISTERED",
   LPM_FILE => "user_cr.mif")
  PORT MAP  (address => addr,
   memenab => rd_en,
```

```
   q => data_mem);

-- mux for card_nr
call_mux_cnr: busmux
  GENERIC MAP   (WIDTH => 8)
  PORT MAP  (dataa => data_mem,
   datab => data_cnr,
   sel => addr card nr,
   result => data_cnr_tri);

tri data cnr:
FOR i IN 0 TO 7 GENERATE
 call_data_cnr: tri
  PORT MAP(data cnr tri(i),
      addr_card_nr,
      data(i));
END GENERATE tri data cnr;

-- mux for serial_nr first ASCII character
call mux snr 1: busmux
  GENERIC MAP   (WIDTH => 8)
  PORT MAP  (dataa => data_mem,
   datab => data snr 1,
   sel => addr_ser_nr_1,
   result => data_snr_1_tri);

tri_data_snr_1:
FOR i IN 0 TO 7 GENERATE
 call data snr 1: tri
  PORT MAP(data_snr_1_tri(i),
      addr_ser_nr_1,
      data(i));
END GENERATE tri_data_snr_1;

-- mux for serial_nr second ASCII character
call_mux_snr_2: busmux
  GENERIC MAP   (WIDTH => 8)
  PORT MAP  (dataa => data_mem,
   datab => data_snr_2,
   sel => addr_ser_nr_2,
   result => data_snr_2_tri);

tri_data_snr_2:
FOR i IN 0 TO 7 GENERATE
 call_data_snr_2: tri
  PORT MAP(data_snr_2_tri(i),
      addr_ser_nr_2,
      data(i));
END GENERATE tri_data_snr_2;

tri_data_mem:
FOR i IN 0 TO 7 GENERATE
 call_data_mem: tri
  PORT MAP(data_mem(i),
      addr_mem,
      data(i));
END GENERATE tri_data_mem;

END ARCHITECTURE rtl;
```

```
-- ********************************************************************************
-- Specify values for addresses of Configuration ROM for VME64x of PSB9U-cards
-- Only every forth address of CR table is used. D08_O = 1 and A01 = 1.

-- V100C:

-- CR/CSR space definition:

-- VME64x_CR range: 0x03-0xFFF (0x03-0x7FF in this mif-file defined)
-- USER_CR range for chip_id, version and SN: 0x001003-0x001033 (see user_cr.mif)
-- CRAM range for future applications (not defined yet): 0x003003-0x0037FF
-- USER_CSR range for TEST_OUT-selection register: 0x005003-0x00502F
-- VME64x_CSR range: 0x7FC00-0x7FFFF

-- FUNCTION definition:

-- Function 0: D16 only, A31-A25, AM=0x0D and 0x09 (single transfer)
-- Function 1: D16 only, A31-A25, AM=0x0F and 0x0B (block 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 = 0x001033
036          : 10; -- Offset to END_SN (byte 1), (0x10) [CR address = 0xDB]
037          : 33; -- 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          : 83; -- Function 1 DAWPR  (0x83, "D16 only (C. Schwick!)", 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          : 00; -- Function 1 AMCAP  (MSB, byte 7) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x143]□
051          : 00; -- Function 1 AMCAP  (byte 6) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x147]□
052          : 00; -- Function 1 AMCAP  (byte 5) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x14B]□
053          : 00; -- Function 1 AMCAP  (byte 4) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x14F]□
□
054          : 00; -- Function 1 AMCAP  (byte 3) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x153]□
055          : 00; -- Function 1 AMCAP  (byte 2) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x157]□
056          : 88; -- Function 1 AMCAP  (byte 1) (0x88, AM=0x0F and 0x0B, see 10.2.1.4.2 VME64x spec.) [CR address = 0x15B]□
057          : 00; -- Function 1 AMCAP  (LSB, byte 0) (0x00, see 10.2.1.4.2 VME64x spec.) [CR address = 0x15F]□
□
[058..187]   : 00;  -- not used! ("Function 2 - 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 = 0
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          : FE; -- Function 1 ADEM  (MSB, byte 3) (0xFE, "mask bits 31-25=1, 24=0", see Table 10-4 VME64x spec.) [CR address = 0
18D          : 00; -- Function 1 ADEM  (byte 2) (0x00, "mask bits 23-16=0", see Table 10-4 VME64x spec.) [CR address = 0x637]
18E          : 00; -- Function 1 ADEM  (byte 1) (0x00, "mask bits 15-8=0", see Table 10-4 VME64x spec.) [CR address = 0x63B]
18F          : 00; -- Function 1 ADEM  (LSB, byte 0) (0x00, "special bits=0", see Table 10-4 VME64x spec.) [CR address = 0x63F]

[190..1AA]   : 00; -- not used! ("Function 2 - 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]

[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;

```

```
-- ********************************************************************************
-- Specify values for addresses of User Configuration ROM for VME64x of PSB9U-cards
-- Only every forth address is used. D08_O = 1 and A01 = 1.
-- USER_CR range for chip_id, version and SN (VME64x): 0x001003-0x001037
-- chip id: 0x00018011 => 0 for card nr, card nr comes in hardware from jumper S31-S28!!!!
-- see user_cr.vhd V2.2 !!!
-- version: 0x0000100C
-- ********************************************************************************

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           : 80; -- 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           : 0C; -- version-register 0 [USER_CR address = 0x00101F]

8           : 50; -- serial number byte 7 (MSB), ASCII "P" [SN address = 0x001023]
9           : 53; -- serial number byte 6, ASCII "S" [SN address = 0x001027]
A           : 42; -- serial number byte 5, ASCII "B" [SN address = 0x00102B]
B           : 00; -- serial number byte 4 ASCII "from CARD_NR jumper S31-S28" [SN address = 0x00102F]

C           : 00; -- serial number byte 3 (LSB), ASCII "from CARD_NR jumper S31-S28" [SN address = 0x001033]

[D..F]      : 00; -- not used! [CR address = 0x001037-0x00103F]

END;
```

VME-CHIP
(Version 0x1007)

of
PSB-9U-card
(9U-Version)

H. Bergauer, K. Kastner, M. Padrta, A. Taurok

Jan-06

Version 0x1007

1 Introduction 3
2 VME chip of PSB-9U-card 3
2.1 Versionshistory 3
2.2 Hardware 3
2.3 Firmware 3
2.4 Features of the VME-CHIP-PSB (V1007) 3
2.5 VME access 4
2.5.1 Base address 4
2.5.2 AM and datatransfer 4
2.6 Chip selection on PSB-9U-card 4
2.7 VME-CHIP-PSB register 4
2.7.1 VME-CHIP-PSB address-table 4
2.7.2 Register for Programmable-chips-configuration 6
2.7.3 General pulse registers 7
2.7.4 General registers 8
2.7.5 Chip_ID and version registers 9
2.7.6 JTAG-registers 9
2.7.7 Serial link mode registers 10
2.7.8 EN_TTIN Register to enable Technical or Totem Trigger bits 10
2.7.9 Error-counter Register for serial-link-chips (SERLINK) 11
2.8 DTACK/BERR-generation 11

1 Introduction

The VME-CHIP-PSB works with the VME64x chip PSB as controller for the VME-bus of the PSB-9U-card. There are VME-registers on it as the Registers for Programmable-chips-configuration, General registers, Chip ID / version registers and JTAG registers. The VME-accesses to the PSB-chip are made via this chip too.

2 VME chip of PSB-9U-card

2.1 Versionshistory

- V1000: first design. Error at RESET_MODE. **DO NOT USE!!** (HB110805).
- V1001: based on V1000, but RESET_MODE error corrected. **DO NOT USE!!** (HB110805).
- V1002: based on V1001, but PSB/MEM access (ENPSB, ENPSBMEM) made with DSSYNC for read and write accesses. (HB120805).
- V1003: based on V1002, but write/read for general-register and configuration-register implemented. (HB160805).
- V1004: based on V1003, but lines ASCYC, ASSYNC, ASPULS and D08_E from VME64x-chip represent the CARD_NR[3:0] – from jumpers S27-S24 on board.
  DSSYNC with two clock delays (ADDR_DEC_PSB_V2_2 and output-FF) used for read/write of PSB-chip.
  [VIEWDRAW: library P:\Lab3Lib\..\vme_chip_lib used] (HB121005).
- V1005: based on V1004, but DTACK_EXT and BERR_EXT are used as negative active signals to VME64x-chip (because at power-up configuration of VME64X-CHIP is faster than configuration of VME-CHIP and therefore wrong DTACK and BERR signals are generated after configuration, which causes LEDs=”on” of CAEN-controller). (HB201005)
  Because of changes in the VME64x-chip version, the CARD_NR[3:0] comes from jumpers S31-S28 on board (HB071105).
- V1006: based on V1005, but error-counters for serial-link-chips implemented. (HB071105).
- **V1007:** based on V1006, but JTAG_CTRL V1.5 implemented, because of Quartus 5.1 and EN_JTAG is delayed to have proper setup-time for addresses, because of syncronous version of VME64x-chip. (HB050106).

2.2 Hardware

The VME-CHIP-PSB is an Altera EP1K100QC208-3.

2.3 Firmware

chip_id: `0x00018n21` `(n = CARD_NR from jumpers S31 – S28)`
version: `0x00001007`

2.4 Features of the VME-CHIP-PSB (V1007)

- Cardnumber from VME64x-chip (from jumpers S31-S28 on board).
- Register for chip_ID and version.
- Register for commands and status (see VME-CHIP-PSB address-table).
- Register for modes of serial-link-chips.
- Register to enable technical- and/or totem-trigger-bits.
- VME-access to PSB-chip with DSSYNC for read and write (two clock delays).
- “JTAG over VME” for configuration of FPGAs.
- Configuration of PSB-chip via VME (changes in hardware of mezz957 necessary!!!).

- DTACK and BERR generation from PSB-chip.
- Error-counters for serial-link-chips.

2.5 VME access

2.5.1 Base address

Base address of all GT-slaves is encoded on A31-A25 (A24 not used), because of address space of GTL-6U-card. See definition in VME64x-chip for PSB-9U.

2.5.2 AM and datatransfer

AM=0x0D and 0x09 „extended data access“ - for single access.
AM=0x0F and 0x0B „extended block transfer“ - for block transfer access.
D16 „word access” - for all accesses.
See definitions in VME64x-chip for PSB.

2.6 Chip selection on PSB-9U-card

With the VME addresses A23-A20 the chip selection is done on the PSB-9U-card.

A23	A22	A21	A20	Chip-name
0	0	0	0	VME-CHIP-PSB
0	0	0	1	PSB-chip-registers
0	0	1	0	PSB-chip-memories

2.7 VME-CHIP-PSB register

Registeraddresses:			
A31..A24	A23..A20	A19..A05	A06..A01,(00)
8 bits		13bits	6bits
Base address	0000	XXXX	Registers

2.7.1 VME-CHIP-PSB address-table

The address-table lists the address-offset which has to be combined with the base-address of the card.

A23-A00 => Register-name

Register for Programmable-chips-configuration:

`0x000000` => CMD_ENPROG-register (write/read)
`0x000002` => CMD_NPROG-register (write/read)
`0x000004` => CMD_INIT-register (write/read)
`0x000006` => STAT_INIT-register (read)
`0x000008` => STAT_DONE-register (read)
`0x00000A` => Configuration register PSB-chip (write)

General pulse registers:

`0x000010` => Command pulse register (write)
`0x000012` => Status pulse register (read)

General registers:

0x000014 => Command register (write/read)
0x000016 => Status register (read)

Chip ID and version registers:

0x000020 => chip_id_register_3 (read)
0x000022 => chip_id_register_2 (read)
0x000024 => chip_id_register_1 (read)
0x000026 => chip_id_register_0 (read)
0x000028 => version_register_3 (read)
0x00002A => version_register_2 (read)
0x00002C => version_register_1 (read)
0x00002E => version_register_0 (read)

JTAG registers:

0x000030 => tdo_register (write/read)
0x000032 => tdi_register (write/read)
0x000034 => tms0_register (write/read)
0x000036 => tms1_register (write/read)
0x000038 => cnt32_register (write/read)
0x00003A => mode0_register (write/read)
0x00003C => mode1_register (write/read)
0x00003E => mode2_register (write/read)

Serial link mode registers:

0x000040 => SERLINK0-register (write/read)
0x000042 => SERLINK1-register (write/read)
0x000044 => SERLINK2-register (write/read)
0x000046 => LOCKED-register (read)

EN_TTIN Register to enable Technical or Totem Trigger bits:

0x000050 => EN_TTIN -register (write/read)

Error-counter Register for serial-link-chips (SERLINK):

0x000060 => ERR_CNT_SERLINK_0-register (read)
0x000062 => ERR_CNT_SERLINK_1-register (read)
0x000064 => ERR_CNT_SERLINK_2-register (read)
0x000066 => ERR_CNT_SERLINK_3-register (read)
0x000068 => ERR_CNT_SERLINK_4-register (read)
0x00006A => ERR_CNT_SERLINK_5-register (read)
0x00006C => ERR_CNT_SERLINK_6-register (read)
0x00006E => ERR_CNT_SERLINK_7-register (read)

Access to/from PSB-chip:

0x1XXXXX => see PSB-chip-registers
0x2XXXXX => see PSB-chip-memories

2.7.2 Register for Programmable-chips-configuration

The PSB-chip (Virtex-II) is configurable by configuration device and by VMEbus instructions. The selection is made by jumpers. The register-definition for configuration by VMEbus shall be a standard. See P:\Lab3Lib\Altera\Lab3_altera\sch\xilinx_conf.

Register names	D7..D1	D0
CMD_ENPROG	-	ENPROG_PSB
CMD_NPROG	-	NPROG_PSB
CMD_INIT	-	INIT_PSB
STAT_INIT	-	INIT_PSB
STAT_DONE	-	DONE_PSB
CONF_PSB	-	configuration data

2.7.2.1 CMD_ENPROG-register

0x000000 => CMD_ENPROG-register (write/read)

Bit 0 of the CMD_ENPROG-register allows sending the configuration bits via VME-bus to the PSB-chip.

2.7.2.2 CMD_NPROG-register

0x000002 => CMD_NPROG-register (write/read)

Data-bit 0 = 1 of this register set the NPROG-signal of PSB-chip active. Then it should be reset to '0'. Then the PSB-chip enters into the configuration procedure. The FPGA either waits for configuration data (slave mode) sent via VME or starts to read configuration bits from a serial PROM (master mode).

2.7.2.3 CMD_INIT-register

0x000004 => CMD_INIT-register (write/read)

Data-bit 0 = 1 of this register set the NINIT-signal of PSB-chip active.

2.7.2.4 STAT_INIT-register

0x000006 => STAT_INIT-register (read)

Read the status of the NINIT-signal of PSB-chip (data-bit 0)

2.7.2.5 STAT_DONE-register

0x000008 => STAT_DONE-register (read)

Read the status of the DONE-signal of PSB-chip (data-bit 0). After a successful configuration the PSB-chip sets DONE = 1.

2.7.2.6 Configuration register PSB-chip

0x00000A => Configuration-register PSB-chip (write)

The register is used to load the configuration bits into the PSB-chip (Virtex-II).
A write access to this register generates a CCLK and sends the data-bit 0 as DIN-signal to the PSB-chip, if the CMD_ENPROG-register bit has been set before. The VME accesses are repeated until the last bit has been loaded into the PSB-chip.

2.7.3 General pulse registers

Register names	D3	D2	D1	D0
Command_Pulse _Reg	SET_RUNNING (pulse)	RESET_PSB (pulse*)	RES_DCM_ PSB (pulse)	PWRDWN_ PSB (pulse)
Status_Pulse_Reg	RUNNING	LOCKED_ LED	CLK_LOCKED _PSB	not used

*) also generated by RESET_MODE

Register names	D7	D6	D5	D4
Command_Pulse _Reg	not used	not used	not used	not used
Status_Pulse_Reg	not used	not used	not used	not used

2.7.3.1 Command pulse register

`0x000010` => Command-pulse-register (write)

D0: **PWRDWN_PSB = 1** sends a low active pulse to the PSB-chip setting it into power down mode. NPWRDWN_B is sent as an open drain signal from the VME-PSB-chip to the PSB-chip.
Remark from data sheet:
The power-down sequence enables a designer to set the device into a low-power, inactive state. The sequence is initiated by pulling the PWRDWN_B pin Low. To monitor power-down status, observe the PWRDWN_B pin. When asserted, power-down has completed. After a successful wake-up, the status pin de-asserts. While powered down, the only active pins are the PWRDWN_B and DONE. All inputs are off and all outputs are 3-stated. While in the POWERDOWN state, the Power On Reset (POR) circuit is still active, but it does not reset the device if V_{CCINT} , V_{CCO} , or V_{CCAUX} falls below its minimum value. The POR circuit waits until the PWRDWN_B pin is released before resetting the device. Also, the PROG_B pin is not sampled while the device is in the POWERDOWN state. The PROG_B pin becomes active when the PWRDWN_B pin is released. Therefore, the device cannot be reset while in the POWERDOWN state. The wake-up sequence is the reverse of the power-down sequence.

D1: **RES_DCM_PSB = 1** sends a high active pulse to the PSB-chip, to forces the DCM module to lock.

D2: **RESET_PSB = 1** sends a high active pulse to the PSB-chip for reset activities. (RESET_MODE is another source for RESET_PSB.)

D3: **SET_RUNNING = 1** sends a high active pulse to set board in RUNNING mode.

2.7.3.2 Status pulse register

`0x000012` => Status-pulse-register (read)

D0: not used.

D1: **CLK_LOCKED_PSB = 1** indicates, that the DCM module of the PSB chip are locked to the 40 MHz clock.
This status bit has to be checked immediately after the configuration of the PSB chip and before any other actions. If the chips do not lock then either the clock signal from the TIM board or the on-board oscillator are bad.

D2: **LOCKED_LED** is the status of an AND of all LOCKED-signals.
The LOCKED_LED signal will illuminate the front-panel LED only if all enabled Serial Receiver Chips are locked to the clock of incoming serial data and if the PSB Chip has locked to the system clock.

D3: **RUNNING = 1** board is active.

If RUNNING = 0, send a SET_RUNNING command via VME.

2.7.4 General registers

Register names	**D3**	**D2**	**D1**	**D0**
Command_Reg	V_SEL_ CABLES	VME_CONF	EN_ROBUS	EN_CHLINK
Status_Reg	not used	STATUS_SEL_ VME	not used	EN_CHLINK

Register names	**D7**	**D6**	**D5**	**D4**
Command_Reg	not used	not used	not used	V_SEL_ BACKPL
Status_Reg	not used	not used	JTAG_JUMPER	not used

2.7.4.1 Command register

`0x000014` => Command-register (write)

D0: **EN_CHLINK = 1** enables channel-link-chips, if CLK_LOCKED_PSB is active (signal NEN_CHLINK active).

D1: **EN_ROBUS = 1** enables ROBUS (signal NEN_ROBUS active).

D2: **VME_CONF = 1** enables configuration of PSB-chip via VME and switches external mux from PROM to VME.

D3: **V_SEL_CABLES = 1** switches JTAG-chains to cables (MasterBlaster and Parallel-Cable-IV).

D4: **V_SEL_BACKPL = 1** switches JTAG-chains to backplane connection via SCANPSC110 (if V_SEL_CABLES = 0).

Truthtable for D4 and D3:

D4	**D3**	
0	0	JTAG-chains via VME
X	1	JTAG-chains via cables
1	0	JTAG-chains via backplane

2.7.4.2 Status register

`0x000016` => Status-register (read)

D0: **EN_CHLINK** is the inverted status of signal NEN_CHLINK.

D1: not used.

D2: **STATUS_SEL_VME = 1** indicates, that configuration of PSB-chip via VME is selected.
For configuration of PSB-chip via VME, set VME_CONF = 1 in the Command-register.

D3: not used.

D4: not used.

D5: **JTAG_JUMPER = 1** indicates, that SEL_CABLE_JTAG-jumper (JP50) is inserted. Therefore JTAG-chains are connected to cables (MasterBlaster and Parallel-Cable-IV).
For changing the sources of JTAG-chains, remove the jumper and make the selection with V_SEL_CABLES and V_SEL_BACKPL in the Command-register.

2.7.5 Chip_ID and version registers

2.7.5.1 Definitions

Chip_id_register and version_register have fixed values in the hardware. These registers have read access only.
The versions `0x00000000 - 0x00000FFF` are used for tests.
The versions `0x00001000 - 0xFFFFFFFF` are used for runs in CMS.

2.7.5.2 Settings

chip_id: `0x00018x21`
`(x=CARD_NR comes from jumpers S27-S24 on board)`
version: `0x00001004`

2.7.5.3 Chip_ID and version registers addresses

`0x000020` => chip_id_register_3 (read)

D7	D6	D5	D4	D3	D2	D1	D0
chip_ID [31..24]							

`0x000022` => chip_id_register_2 (read)

D7	D6	D5	D4	D3	D2	D1	D0
chip_ID [23..16]							

`0x000024` => chip_id_register_1 (read)

D7	D6	D5	D4	D3	D2	D1	D0
chip_ID [15..08]							

`0x000026` => chip_id_register_0 (read)

D7	D6	D5	D4	D3	D2	D1	D0
chip_ID [07..00]							

`0x000028` => version_register_3 (read)

D7	D6	D5	D4	D3	D2	D1	D0
version [31..24]							

`0x00002A` => version_register_2 (read)

D7	D6	D5	D4	D3	D2	D1	D0
version [23..16]							

`0x00002C` => version_register_1 (read)

D7	D6	D5	D4	D3	D2	D1	D0
version [15..08]							

`0x00002E` => version_register_0 (read)

D7	D6	D5	D4	D3	D2	D1	D0
version [07..00]							

2.7.6 JTAG-registers

2.7.6.1 Definitions

JTAG registers are used to control JTAG-chains via VME-bus.

For details see JTAGController.vhd from Hannes Sakulin.

2.7.7 Serial link mode registers

Register names	**D15..D12**	**D11..D8**	**D7..D4**	**D3..D0**
SERLINK0	SEND_SYNC_ PATTERN [3..0]	LINE_ LOOPBACK [3..0]	TPWRDWN [3..0]	ENTR [3..0]
SERLINK1	RPWRDWN [7..0]		ENREC [7..0]	
SERLINK2	not used		LOCAL_LOOPBACK [7..0]	
LOCKED	not used		LOCKED [7..0]	

Remarks:

23.805 AT corrected in table above NTPWRDWN to TPWRDWN, NRPWRDWN to RPWRDWN.

Index [7..0] means DS92LV16 chip number = channel number.

SEND_SYNC_PATTERN = 1 => transmitter sends SYNC patterns so that a receiver can synchronize to the incoming data stream.
LINE_LOOPBACK = 1 => the serial received data are returned via the serial transmission line.
TPWRDWN = 1 => powers down the transmitter part of the chip (signal NTPWRDWN=0).
ENTR = 1 => enables the transmitter circuits of the chip.
LOCAL_LOOPBACK = 1 => returns parallel Transmit-data to parallel Receiver lines.
RPWRDWN = 1 => powers down the receiver part of the chip (signal NRPWRDWN=0).
ENREC = 1 => enables the receiver circuits of the DS92LV16 chip.

`0x000040` => SERLINK0-register (write/read)
`0x000042` => SERLINK1-register (write/read)
`0x000044` => SERLINK2-register (write/read)
`0x000046` => LOCKED-register (read)

2.7.8 EN_TTIN Register to enable Technical or Totem Trigger bits

Enables the LVDS receivers for Technical Trigger resp. Totem Trigger signals.
Disabled Receivers send bits 1111 = 'F'.
See also registers in PSB9U chip to switch between Parallel LVDS and Serial input channels.

Register names	**D15..D0**
EN_TTIN	EN_R[15..0]

EN_Rxx = 1 => enables parallel LVDS receivers for Parallel cable xx (0 = default).

`0x000050` => EN_TTIN-register (write/read)

2.7.9 Error-counter Register for serial-link-chips (SERLINK)

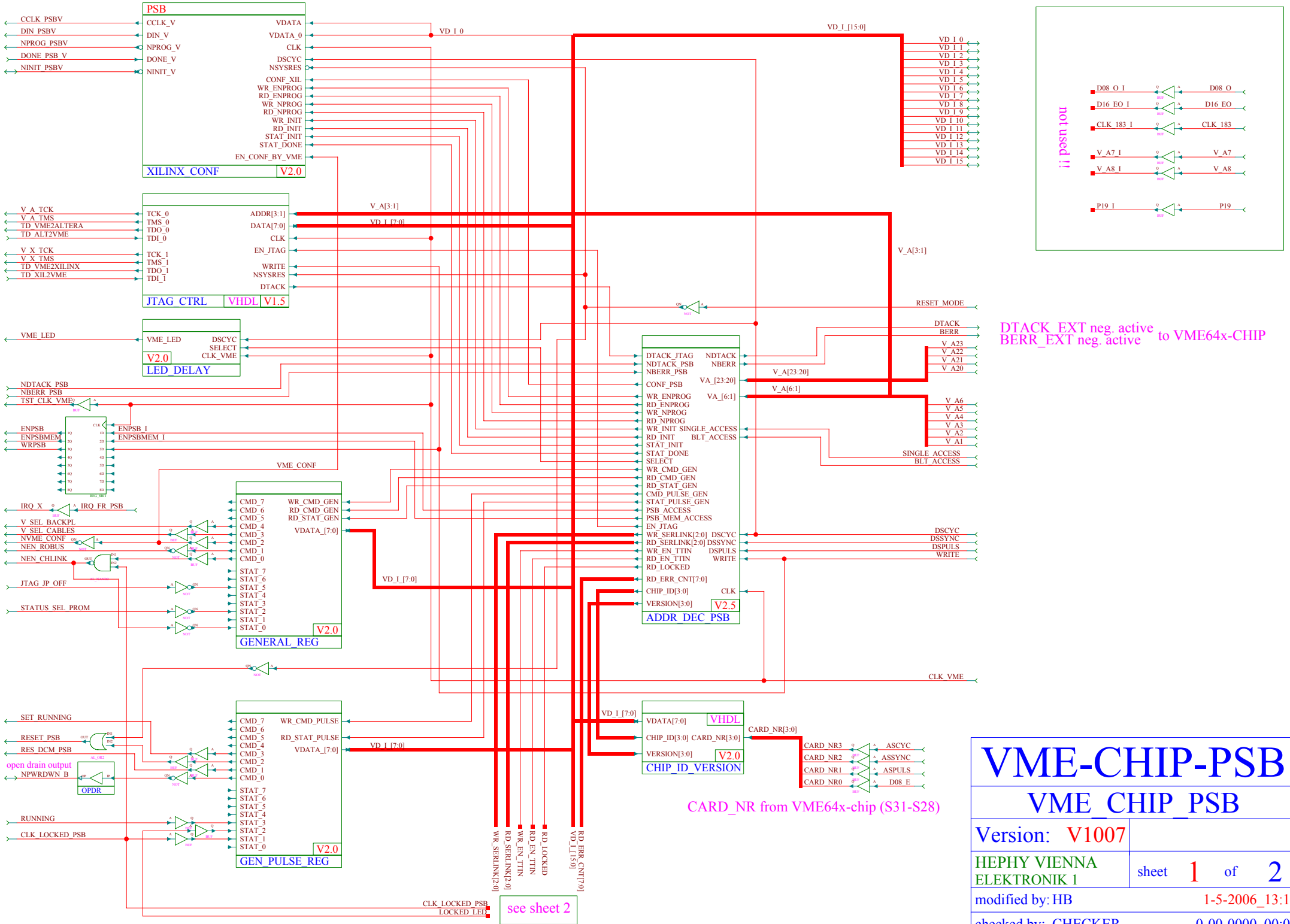
The Error-counter Register for serial-link-chips contains the number of missed LOCKED-signals of the serial-link-chip. If there is an overflow in the counter, the value remains at 0xFF. Reading the register causes a clear on the counter.

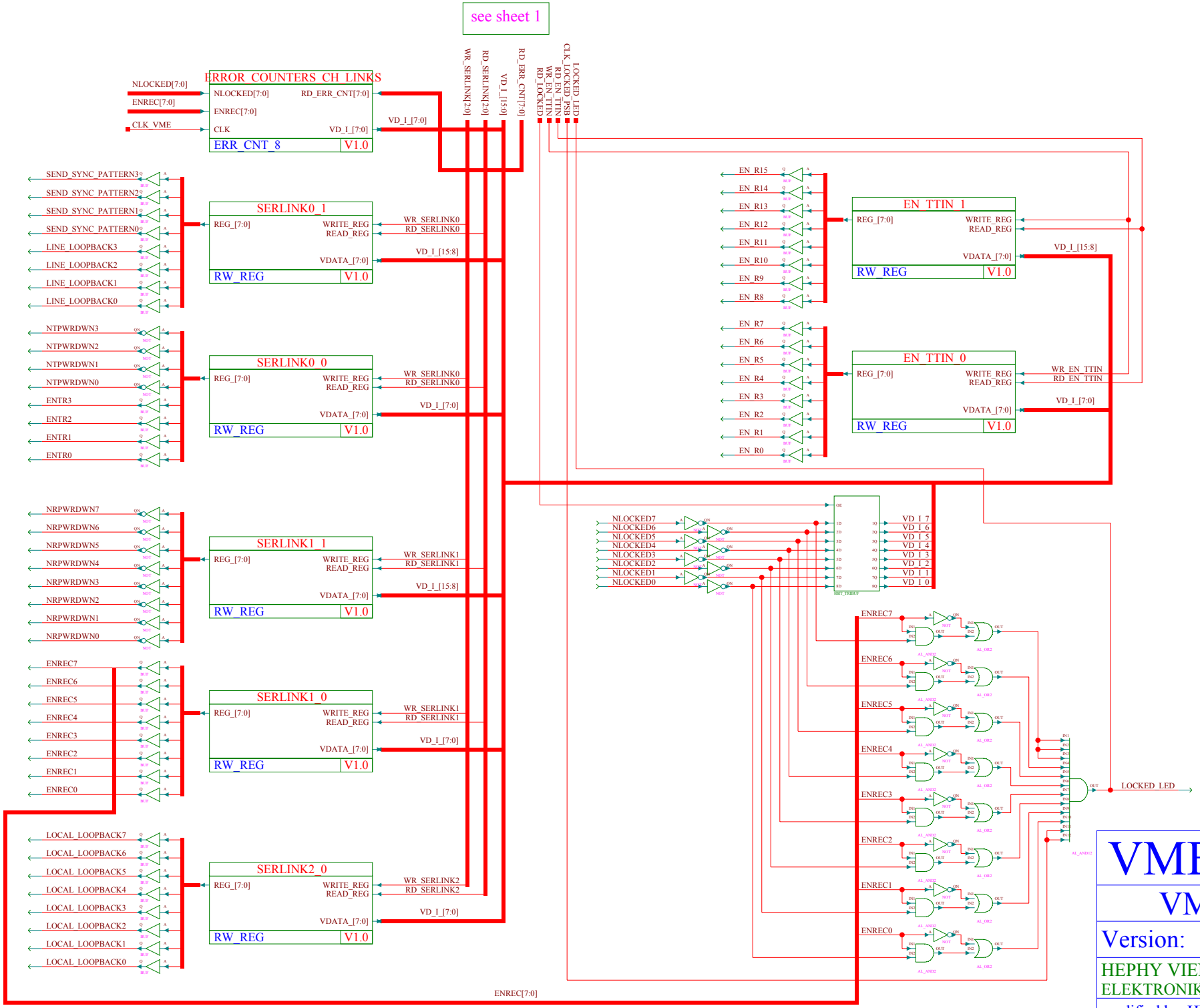
Register names	**D7..D0**
ERR_CNT_SERLINK_x	Errors caused by missing LOCKED-signal of serial-link-chip

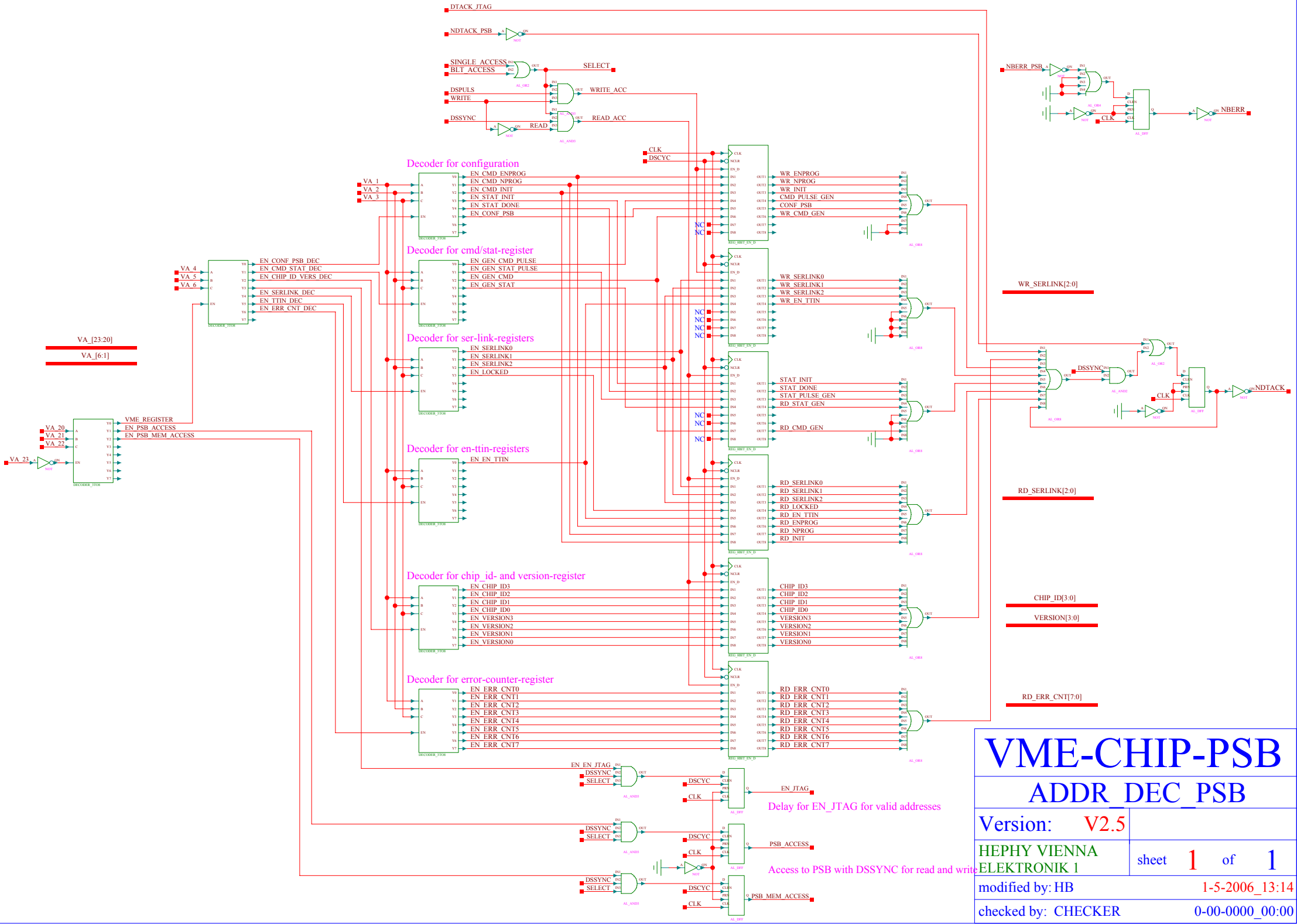
0x000060 => ERR_CNT_SERLINK_0-register (read)
0x000062 => ERR_CNT_SERLINK_1-register (read)
0x000064 => ERR_CNT_SERLINK_2-register (read)
0x000066 => ERR_CNT_SERLINK_3-register (read)
0x000068 => ERR_CNT_SERLINK_4-register (read)
0x00006A => ERR_CNT_SERLINK_5-register (read)
0x00006C => ERR_CNT_SERLINK_6-register (read)
0x00006E => ERR_CNT_SERLINK_7-register (read)

2.8 DTACK/BERR-generation

Writing to writeable registers and reading from readable registers generates a DTACK signal. Access to/from PSB-chip generates a DTACK and a BERR signal.


PSB
CCLK_PSBV
DIN_PSBV
NPROG_PSBV
DONE_PSB_V
NINIT_PSBV
CCLK_V
DIN_V
NPROG_V
DONE_V
NINIT_V
VDATA
VDATA_0
CLK
DSCYC
NSYSRES
CONF_XIL
WR_ENPROG
RD_ENPROG
WR_NPROG
RD_NPROG
WR_INIT
RD_INIT
STAT_INIT
STAT_DONE
EN_CONF_BY_VME
XILINX_CONF
V2.0
VD_I_0
VD_I_[15:0]
VD_I_0
VD_I_1
VD_I_2
VD_I_3
VD_I_4
VD_I_5
VD_I_6
VD_I_7
VD_I_8
VD_I_9
VD_I_10
VD_I_11
VD_I_12
VD_I_13
VD_I_14
VD_I_15
not used !!
D08_O_I
D16_EO_I
CLK_183_I
V_A7_I
V_A8_I
P19_I
D08_O
D16_EO
CLK_183
V_A7
V_A8
P19
V_A_TCK
V_A_TMS
TD_VME2ALTERA
TD_ALT2VME
V_X_TCK
V_X_TMS
TD_VME2XILINX
TD_XIL2VME
TCK_0
TMS_0
TDO_0
TDI_0
TCK_1
TMS_1
TDO_1
TDI_1
ADDR[3:1]
DATA[7:0]
CLK
EN_JTAG
WRITE
NSYSRES
DTACK
JTAG_CTRL
VHDL
V1.5
V_A[3:1]
VD_I_[7:0]
V_A[3:1]
VME_LED
VME_LED
DSCYC
SELECT
CLK_VME
V2.0
LED_DELAY
RESET_MODE
DTACK
BERR
DTACK_EXT neg. active
BERR_EXT neg. active
to VME64x-CHIP
NDTACK_PSB
NBERR_PSB
TST_CLK_VME
ENPSB
ENPSBMEM
WRPSB
ENPSB_I
ENPSBMEM_I
DTACK_JTAG
NDTACK_PSB
NBERR_PSB
CONF_PSB
WR_ENPROG
RD_ENPROG
WR_NPROG
RD_NPROG
WR_INIT
RD_INIT
STAT_INIT
STAT_DONE
SELECT
WR_CMD_GEN
RD_CMD_GEN
RD_STAT_GEN
CMD_PULSE_GEN
STAT_PULSE_GEN
PSB_ACCESS
PSB_MEM_ACCESS
EN_JTAG
WR_SERLINK[2:0]
RD_SERLINK[2:0]
WR_EN_TTIN
RD_EN_TTIN
RD_LOCKED
RD_ERR_CNT[7:0]
CHIP_ID[3:0]
VERSION[3:0]
NDTACK
NBERR
VA_[23:20]
VA_[6:1]
SINGLE_ACCESS
BLT_ACCESS
DSCYC
DSSYNC
DSPULS
WRITE
CLK
V2.5
ADDR_DEC_PSB
V_A23
V_A22
V_A21
V_A20
V_A6
V_A5
V_A4
V_A3
V_A2
V_A1
V_A[23:20]
V_A[6:1]
VME_CONF
IRQ_X
IRQ_FR_PSB
V_SEL_BACKPL
V_SEL_CABLES
NVME_CONF
NEN_ROBUS
NEN_CHLINK
JTAG_JP_OFF
STATUS_SEL_PROM
CMD_7
CMD_6
CMD_5
CMD_4
CMD_3
CMD_2
CMD_1
CMD_0
STAT_7
STAT_6
STAT_5
STAT_4
STAT_3
STAT_2
STAT_1
STAT_0
WR_CMD_GEN
RD_CMD_GEN
RD_STAT_GEN
VDATA_[7:0]
GENERAL_REG
V2.0
VD_I_[7:0]
CLK_VME
SET_RUNNING
RESET_PSB
RES_DCM_PSB
open drain output
NPWRDWN_B
OPDR
RUNNING
CLK_LOCKED_PSB
WR_CMD_PULSE
RD_STAT_PULSE
VDATA_[7:0]
GEN_PULSE_REG
V2.0
VD_I_[7:0]
VDATA[7:0]
CHIP_ID[3:0]
CARD_NR[3:0]
VERSION[3:0]
VHDL
V2.0
CHIP_ID_VERSION
CARD_NR[3:0]
CARD_NR3
CARD_NR2
CARD_NR1
CARD_NR0
ASCYC
ASSYNC
ASPULS
D08_E
CARD_NR from VME64x-chip (S31-S28)
WR_SERLINK[2:0]
RD_SERLINK[2:0]
WR_EN_TTIN
RD_EN_TTIN
RD_LOCKED
RD_ERR_CNT[7:0]
VD_I_[15:0]
CLK_LOCKED_PSB
LOCKED_LED
see sheet 2
VME-CHIP-PSB
VME_CHIP_PSB
Version: V1007
HEPHY VIENNA
ELEKTRONIK 1
sheet 1 of 2
modified by: HB
1-5-2006_13:14
checked by: CHECKER
0-00-0000_00:00

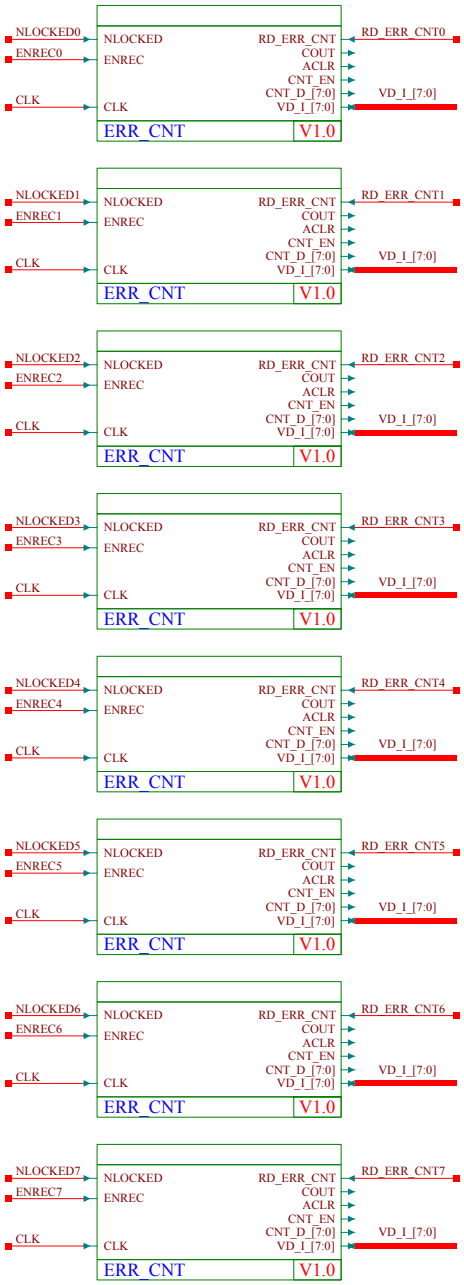

see sheet 1
ERROR_COUNTERS_CH_LINKS
NLOCKED[7:0]
ENREC[7:0]
CLK_VME
NLOCKED[7:0]
ENREC[7:0]
CLK
RD_ERR_CNT[7:0]
VD_I_[7:0]
ERR_CNT_8
V1.0
WR_SERLINK[2:0]
RD_SERLINK[2:0]
VD_I_[15:0]
RD_ERR_CNT[7:0]
RD_LOCKED
WR_EN_TTIN
RD_EN_TTIN
CLK_LOCKED_PSB
LOCKED_LED
SEND_SYNC_PATTERN3
SEND_SYNC_PATTERN2
SEND_SYNC_PATTERN1
SEND_SYNC_PATTERN0
LINE_LOOPBACK3
LINE_LOOPBACK2
LINE_LOOPBACK1
LINE_LOOPBACK0
SERLINK0_1
REG_[7:0]
WRITE_REG
READ_REG
VDATA_[7:0]
RW_REG
V1.0
WR_SERLINK0
RD_SERLINK0
VD_I_[15:8]
NTPWRDWN3
NTPWRDWN2
NTPWRDWN1
NTPWRDWN0
ENTR3
ENTR2
ENTR1
ENTR0
SERLINK0_0
WR_SERLINK0
RD_SERLINK0
VD_I_[7:0]
NRPWRDWN7
NRPWRDWN6
NRPWRDWN5
NRPWRDWN4
NRPWRDWN3
NRPWRDWN2
NRPWRDWN1
NRPWRDWN0
SERLINK1_1
WR_SERLINK1
RD_SERLINK1
VD_I_[15:8]
ENREC7
ENREC6
ENREC5
ENREC4
ENREC3
ENREC2
ENREC1
ENREC0
SERLINK1_0
WR_SERLINK1
RD_SERLINK1
VD_I_[7:0]
LOCAL_LOOPBACK7
LOCAL_LOOPBACK6
LOCAL_LOOPBACK5
LOCAL_LOOPBACK4
LOCAL_LOOPBACK3
LOCAL_LOOPBACK2
LOCAL_LOOPBACK1
LOCAL_LOOPBACK0
SERLINK2_0
WR_SERLINK2
RD_SERLINK2
VD_I_[7:0]
ENREC[7:0]
EN_R15
EN_R14
EN_R13
EN_R12
EN_R11
EN_R10
EN_R9
EN_R8
EN_TTIN_1
REG_[7:0]
WRITE_REG
READ_REG
VDATA_[7:0]
RW_REG
V1.0
VD_I_[15:8]
EN_R7
EN_R6
EN_R5
EN_R4
EN_R3
EN_R2
EN_R1
EN_R0
EN_TTIN_0
WR_EN_TTIN
RD_EN_TTIN
VD_I_[7:0]
NLOCKED7
NLOCKED6
NLOCKED5
NLOCKED4
NLOCKED3
NLOCKED2
NLOCKED1
NLOCKED0
VD_I_7
VD_I_6
VD_I_5
VD_I_4
VD_I_3
VD_I_2
VD_I_1
VD_I_0
LOCKED_LED
VME-CHIP-PSB
VME_CHIP_PSB
Version: V1007
HEPHY VIENNA
ELEKTRONIK 1
sheet 2 of 2
modified by: HB
12-19-2005_13:53
checked by: CHECKER
0-00-0000_00:00



VME-CHIP-PSB

ADDR_DEC_PSB

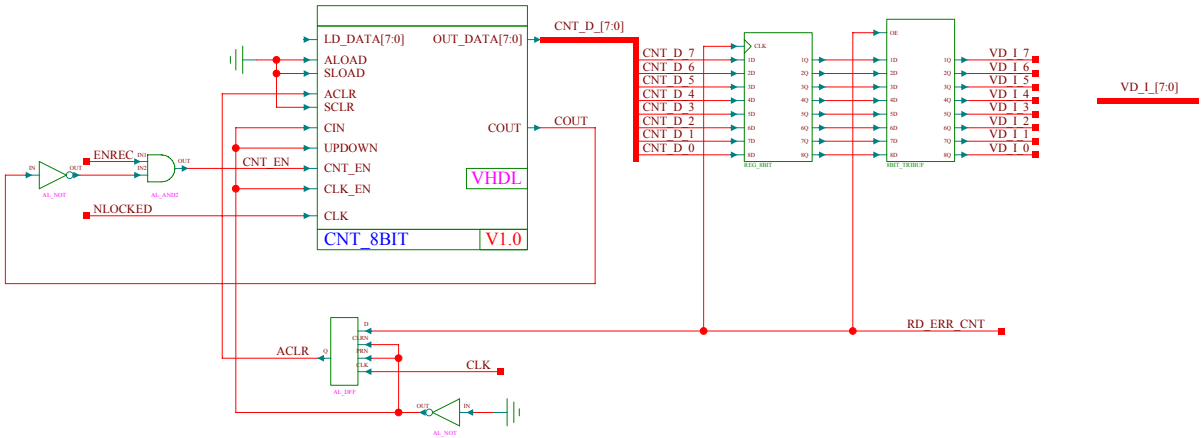
Version: V2.5	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	1-5-2006_13:14
checked by: CHECKER	0-00-0000_00:00



VME-CHIP-PSB

ERR_CNT_8

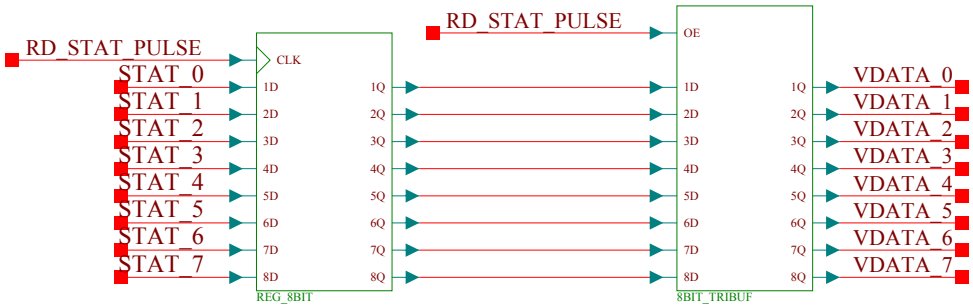
Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	11-7-2005_14:22
checked by: CHECKER	0-00-0000_00:00



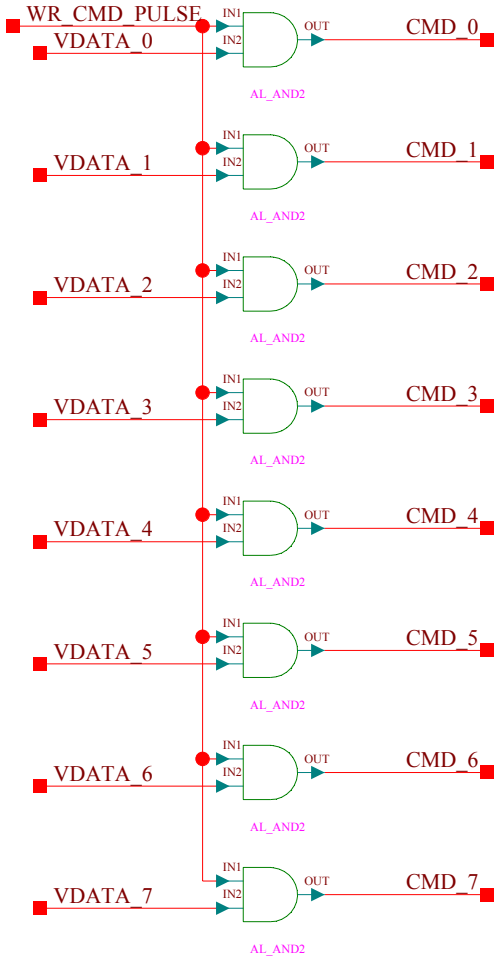
VME-CHIP-PSB

ERR_CNT

Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	11-7-2005_14:44
checked by: CHECKER	0-00-0000_00:00



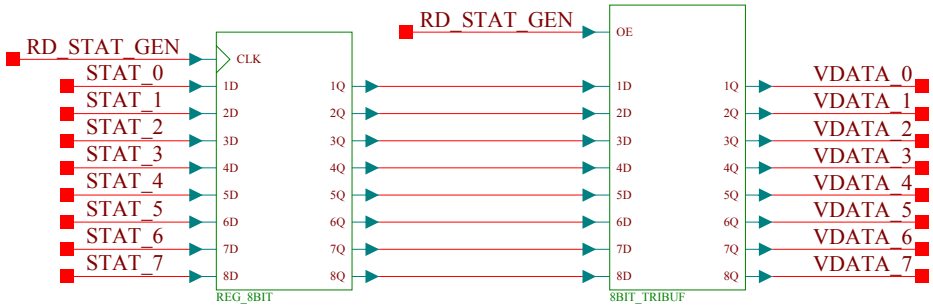
VDATA_[7:0]

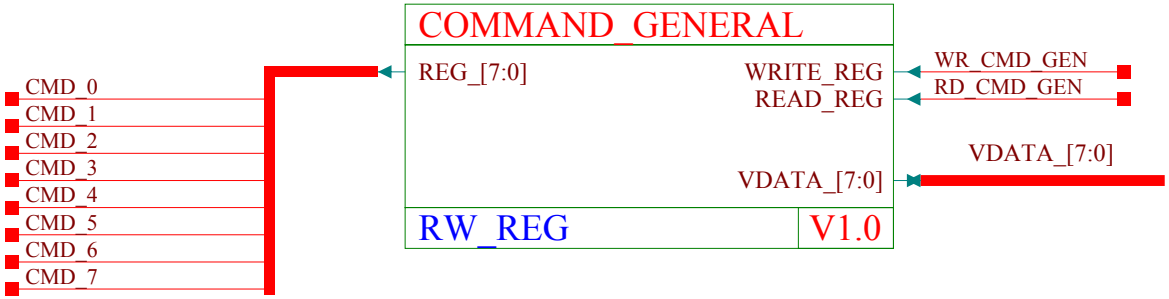


VME-CHIP

GEN_PULSE_REG

Version: V2.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	9-1-2005_10:29
checked by: CHECKER	0-00-0000_00:00



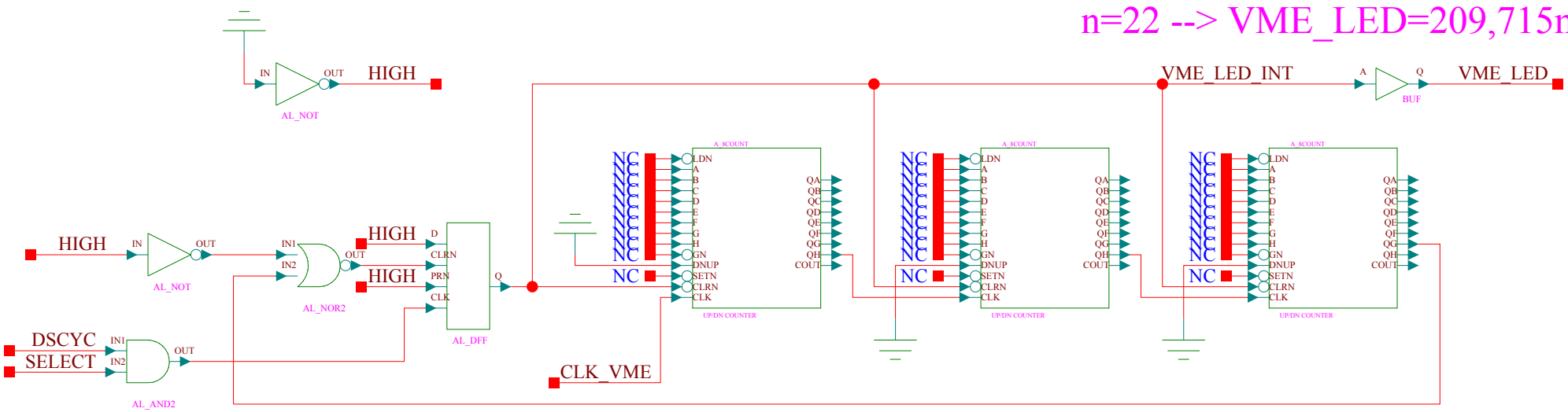


VME-CHIP

GENERAL_REG

Version: V2.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	9-1-2005_10:29
checked by: CHECKER	0-00-0000_00:00

n=22 --> VME_LED=209,715ms



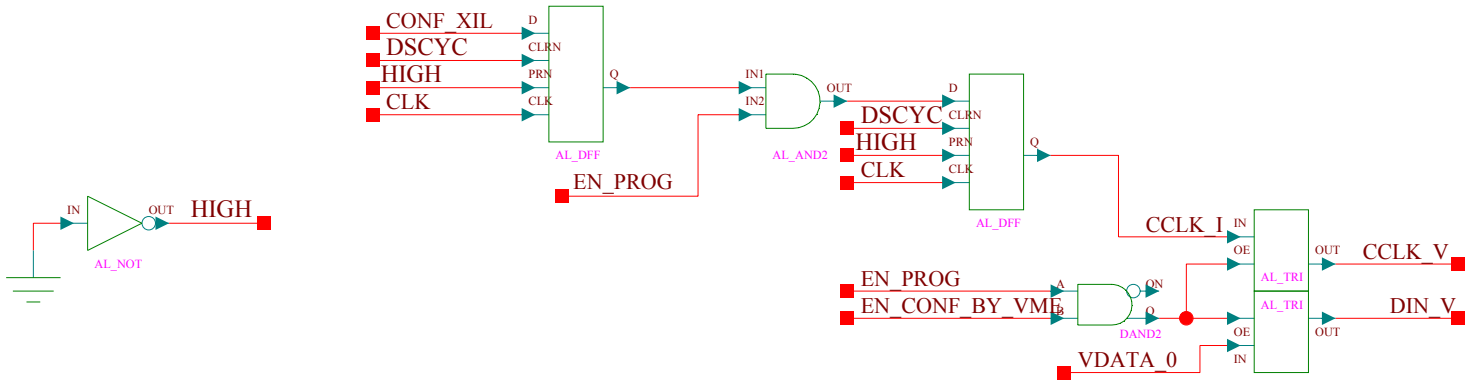
Formel: (2**(n+1)-1)*25ns (25ns=CLK_VME)

VME-CHIP

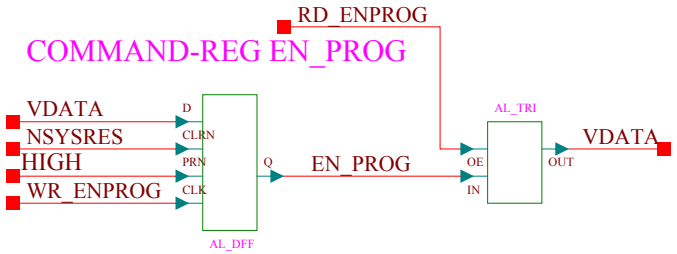
LED_DELAY

Version: V2.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	9-1-2005_10:32
checked by: CHECKER	0-00-0000_00:00

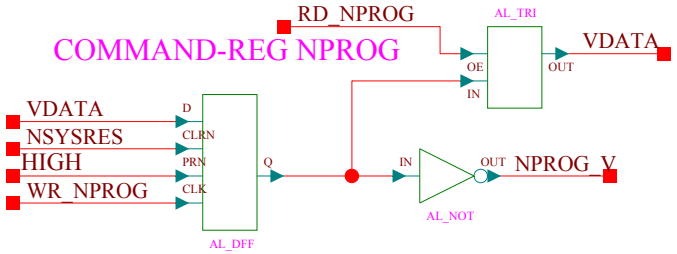
CONFIGURATION OF XILINX CHIP (CCLK, DIN)



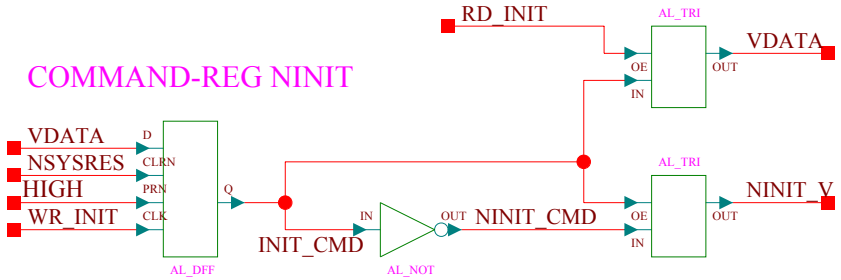
COMMAND-REG EN_PROG



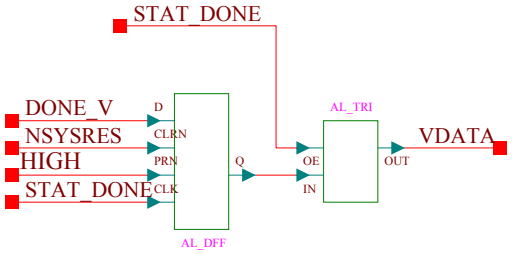
COMMAND-REG NPROG



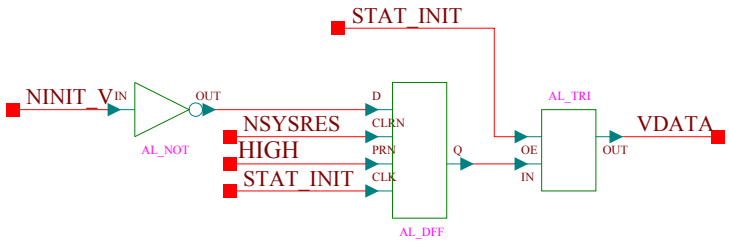
COMMAND-REG NINIT



STATUS-REG DONE



STATUS-REG NINIT

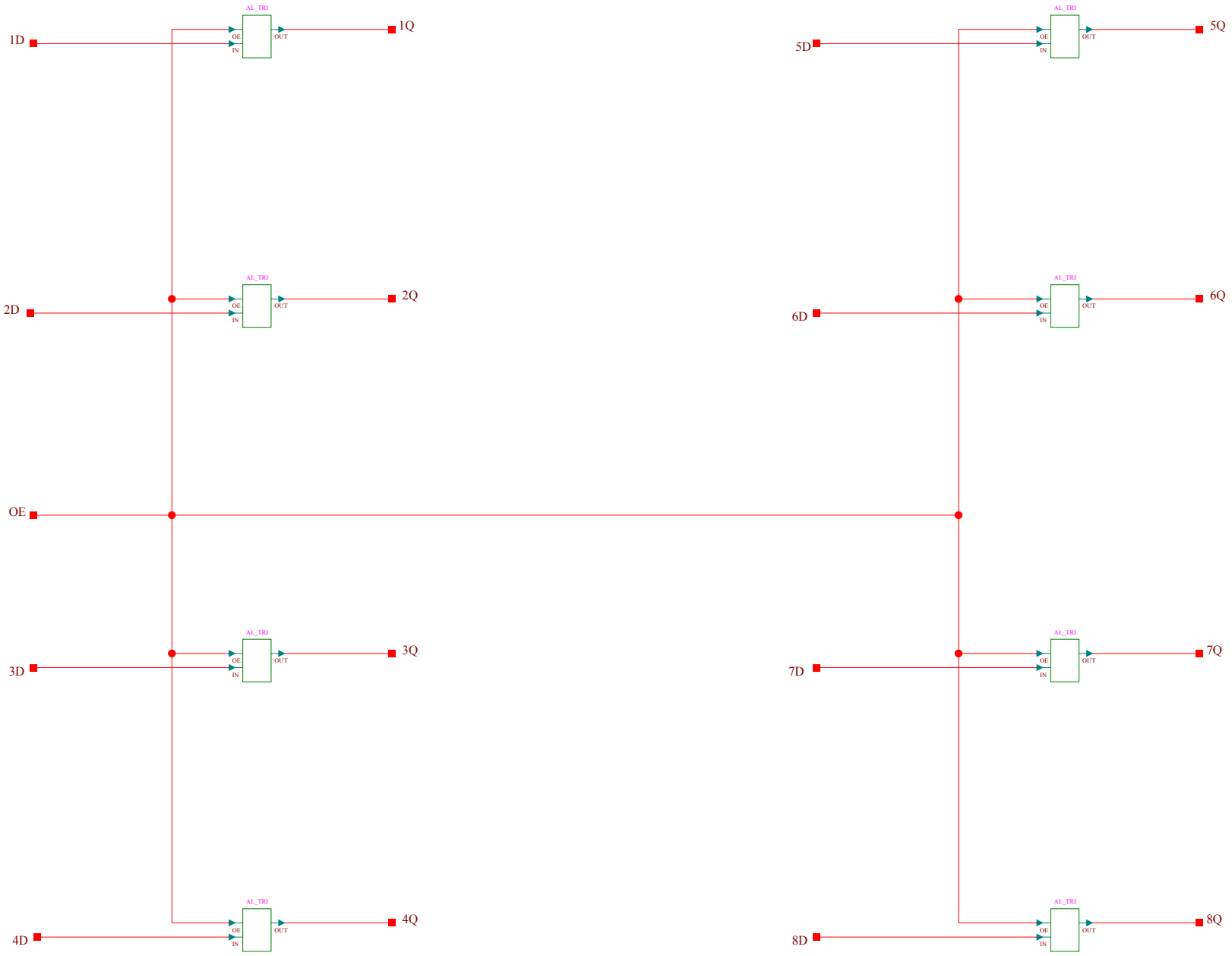


VME-CHIP

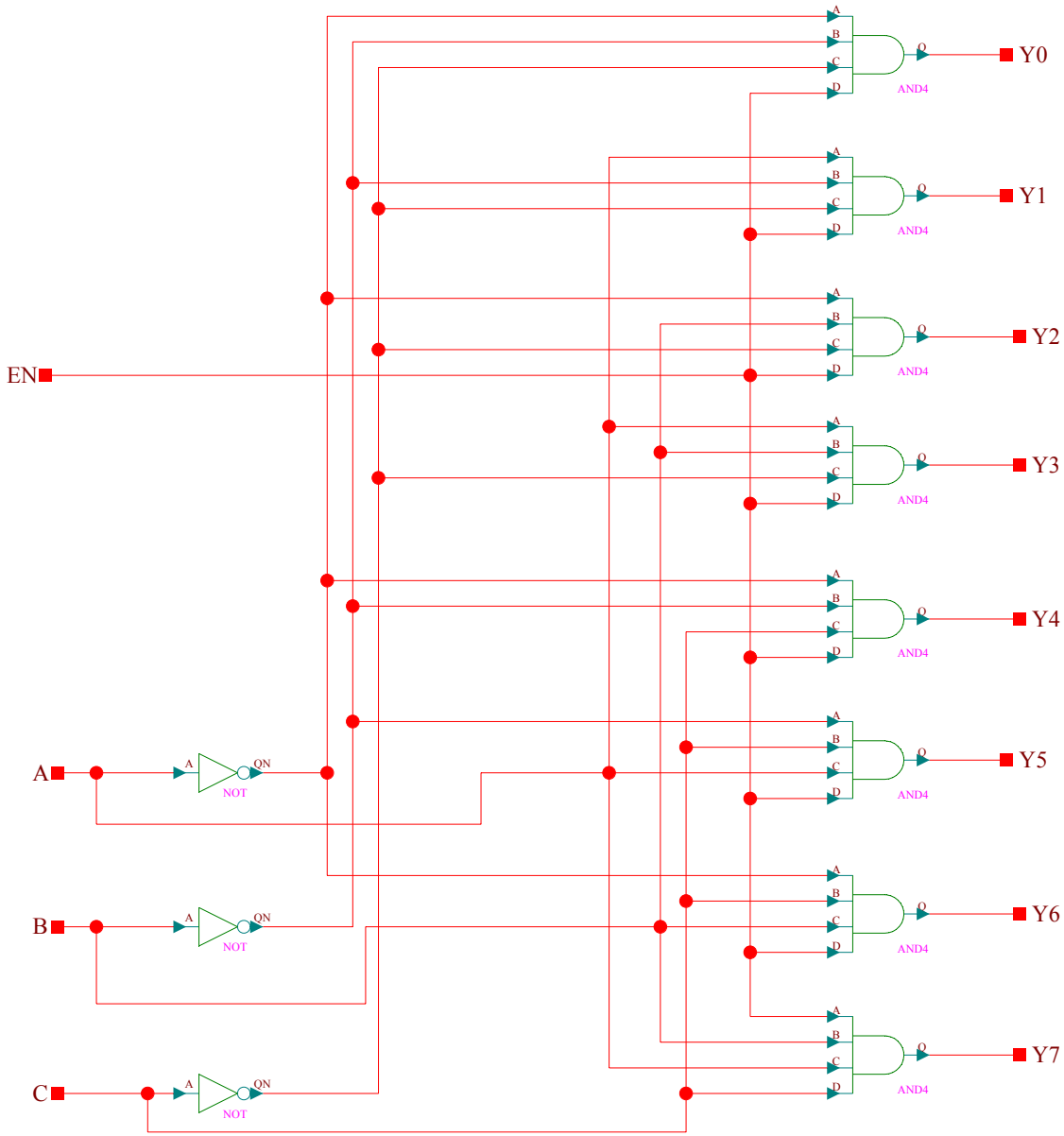
XILINX_CONF

Version: V2.0	Configuration
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	8-26-2005_14:23
checked by: CHECKER	0-00-0000_00:00

8bit_tribuf

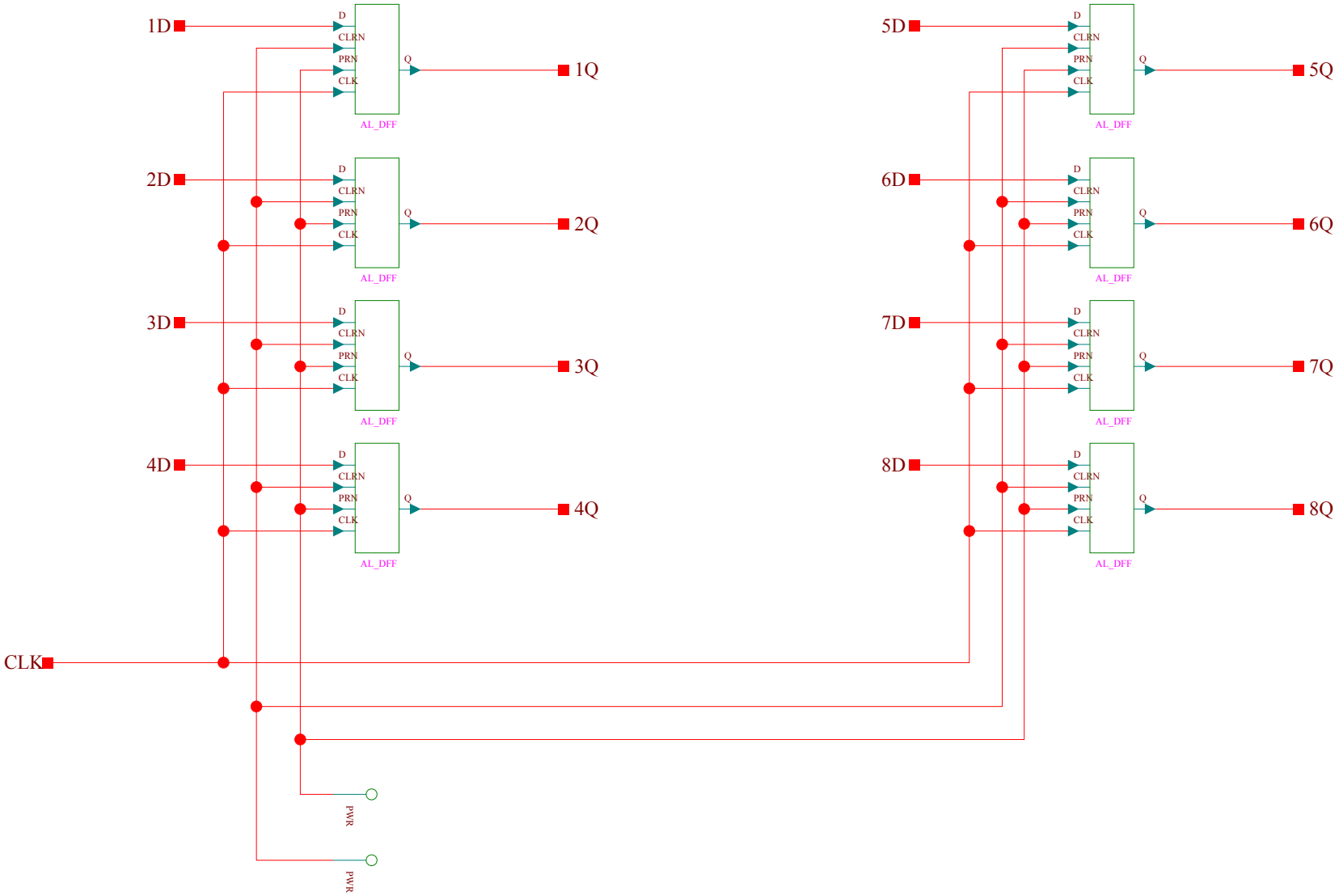


decoder_3to8

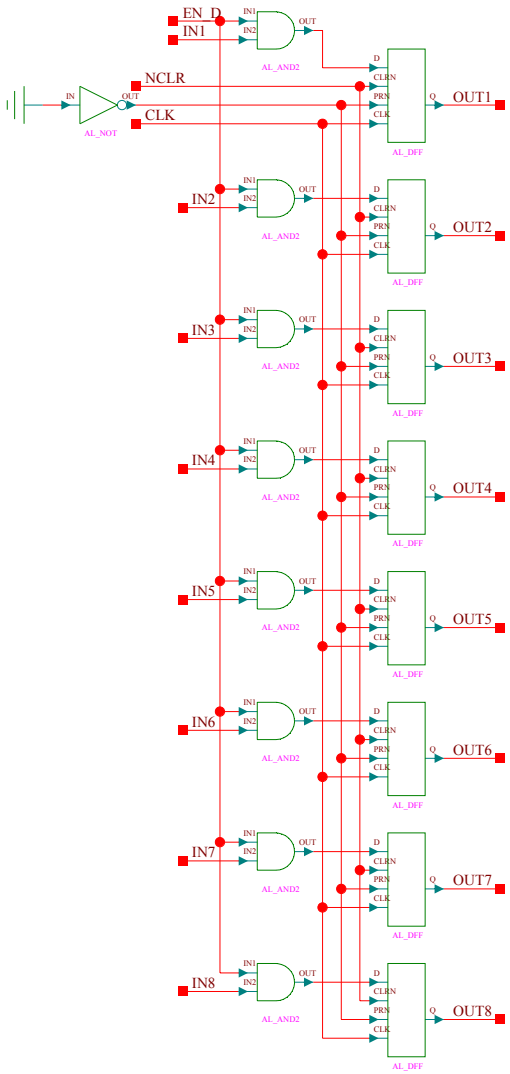


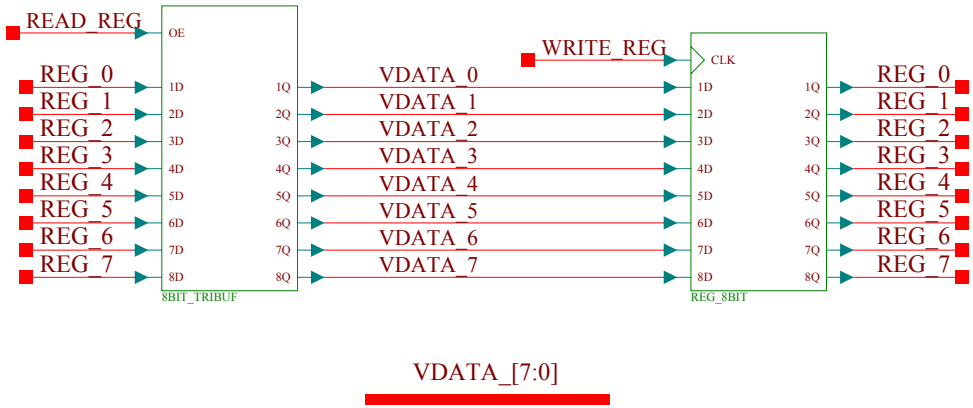
reg_8bit

8-bit register generated by LAB3



reg_8bit_en_d


EN_D
IN1
NCLR
CLK
IN2
IN3
IN4
IN5
IN6
IN7
IN8
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
AL_AND2
AL_DFF
AL_NOT



VME-CHIP

RW_REG

Version: V1.0	
HEPHY VIENNA ELEKTRONIK 1	sheet 1 of 1
modified by: HB	9-9-2005_14:49
checked by: CHECKER	0-00-0000_00:00

```
------------------------------------------------------------
--                                                        --
-- LOGIC CORE: vme-chip logic                             --
-- MODULE NAME: chip_id_version                           --
-- INSTITUTION: Hephy Vienna                              --
-- DESIGNER: H. Bergauer                                  --
--                                                        --
-- VERSION: V2.0                                          --
-- DATE: 08 2005                                          --
--                                                        --
-- FUNCTIONAL DESCRIPTION:                                --
-- chip_id and version register (read only)               --
--                                                        --
------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2.ALL;

USE work.constant pkg.ALL;

ENTITY chip_id_version IS
    PORT(
        VDATA       : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        CARD_NR     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        CHIP_ID     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        VERSION     : IN    STD_LOGIC_VECTOR(3 DOWNTO 0));
END chip_id_version;

ARCHITECTURE rtl OF chip_id_version IS

--  CONSTANT for card_name and version defined in working-dir\vhdl !!!

    SIGNAL chip_id_value : std_logic_vector(31 downto 0);
    CONSTANT cms_gt: STD_LOGIC_VECTOR(15 DOWNTO 0) := X"0001"; -- fixed code for GT-system = 0x0001
--  card_nr will be delivered from VME64x-chip, HB250805
    CONSTANT chip_name: STD_LOGIC_VECTOR(3 DOWNTO 0) := X"2"; -- fixed code for chip_name = 0x2
    CONSTANT chip_nr: STD_LOGIC_VECTOR(3 DOWNTO 0) := X"1"; -- fixed code for chip_nr = 0x1

BEGIN
chip_id_value <= cms_gt & card_name & card_nr & chip_name & chip_nr;


-- chip_id and version register (read only)
tri_chip_id_3:
FOR i IN 0 TO 7 GENERATE
```

```
    call_chip_id_3: tri
   PORT MAP(chip_id_value(i+24),
      chip_id(3),
      vdata(i));
END GENERATE tri chip id 3;

tri_chip_id_2:
FOR i IN 0 TO 7 GENERATE
    call_chip_id_2: tri
   PORT MAP(chip_id_value(i+16),
      chip id(2),
      vdata(i));
END GENERATE tri_chip_id_2;

tri_chip_id_1:
FOR i IN 0 TO 7 GENERATE
    call chip id 1: tri
   PORT MAP(chip_id_value(i+8),
      chip_id(1),
      vdata(i));
END GENERATE tri_chip_id_1;

tri_chip_id_0:
FOR i IN 0 TO 7 GENERATE
    call_chip_id_0: tri
   PORT MAP(chip_id_value(i),
      chip_id(0),
      vdata(i));
END GENERATE tri_chip_id_0;

tri_version_3:
FOR i IN 0 TO 7 GENERATE
    call_version_3: tri
   PORT MAP(version_value(i+24),
      version(3),
      vdata(i));
END GENERATE tri_version_3;

tri_version_2:
FOR i IN 0 TO 7 GENERATE
    call_version_2: tri
   PORT MAP(version_value(i+16),
      version(2),
      vdata(i));
END GENERATE tri_version_2;
```

```
tri_version_1:
FOR i IN 0 TO 7 GENERATE
    call_version_1: tri
   PORT MAP(version_value(i+8),
      version(1),
      vdata(i));
END GENERATE tri_version_1;

tri_version_0:
FOR i IN 0 TO 7 GENERATE
    call version 0: tri
   PORT MAP(version_value(i),
      version(0),
      vdata(i));
END GENERATE tri_version_0;

END ARCHITECTURE rtl;
```

```
--------------------------------------------------------------------------------
-- Title      : Top of JTAG Controller for vme-chip of GT-boards
-- Project    :
--------------------------------------------------------------------------------
-- File       : jtag_ctrl.vhd
-- Author     : H. Bergauer
-- Company    :
--------------------------------------------------------------------------------
-- Description: top module for VIEWDRAW of JTAG Controller
--------------------------------------------------------------------------------
-- $Date: 2006/01/05 08:54:17 $
-- $Revision: 1.5 $
--------------------------------------------------------------------------------

library IEEE;
library work;

use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

LIBRARY altera;
USE altera.maxplus2.ALL;

--------------------------------------------------------------------------------
-- Entity Declaration
--------------------------------------------------------------------------------

entity jtag_ctrl is
  port (
    addr       : in    std_logic_vector(3 downto 1);
    data       : inout    std_logic_vector(7 downto 0);
    clk        : in    std_logic;
    en_jtag    : in    std_logic;
--    dssync     : in    std_logic;
    write      : in    std_logic;
    nsysres    : in    std_logic;
    dtack      : out   std_logic;
    tck_0      : out   std_logic;
    tck_1      : out   std_logic;
    tms_0      : out   std_logic;
    tms_1      : out   std_logic;
    tdo_0      : out   std_logic;
    tdo_1      : out   std_logic;
    tdi_0      : in    std_logic;
    tdi_1      : in    std_logic
);
end;

--------------------------------------------------------------------------------
-- Architecture declaration
--------------------------------------------------------------------------------

architecture rtl of jtag_ctrl is

--  address parameters for JTAGController
  constant base_address : integer := 0;
  constant address_increment : integer := 2;
  constant addr_high : integer := 3;
  constant addr_low : integer := 1;

component JTAGController is
  generic (
    base_address      : integer := 0;   -- base address (in bytes)
    address_increment : integer := 2;   -- increment (2 for word access,
                                        -- 4 for long word access)
    addr_high : integer := 3;           -- upper index of vme_addr vector
    addr_low  : integer := 1            -- lower index of vme_addr vector
    );
```

```
  port (
    -- JTAG port
    -- to be connected directly to I/O pins of chip
    oTck  : out std_logic;
    oTms0 : out std_logic;
    oTms1 : out std_logic;
    oTdo  : out std_logic;
    iTdi0 : in  std_logic;
    iTdi1 : in  std_logic;

    -- VME port
    vme_addr       : in    std_logic_vector(addr_high downto addr_low);
                                                    
    vme_data       : in    std_logic_vector(15 downto 0);  -- has to be at least 8 bit
    vme_en         : in    std_logic;   -- should not be a pulse when writing
    vme_wr         : in    std_logic;   -- state. must remain for at least two
                                        -- clocks after enable goes to 0
    vme_dtack      : out   std_logic;         -- not inverted

    vme_data_out   : out   std_logic_vector(15 downto 0);
    vme_en_out     : out   std_logic;
        
    -- Clock and control
    clk            : in    std_logic;
    reset          : in    std_logic);  -- asynchronous reset, active high
end component;

--  signal vme_en_jtag : std_logic;
  signal en_tri : std_logic;
  signal tck : std_logic;
  signal tdo : std_logic;
  signal data_jtag : std_logic_vector(15 downto 0);
  signal data_in : std_logic_vector(15 downto 0);
  signal data_out : std_logic_vector(15 downto 0);
  signal sysres : std_logic;

begin
-- verändert HB191205 wegen Quartus 5.1 Fehlermeldung

en_tri <= en_jtag AND NOT write;
--data <= data_out(7 downto 0);
data_in <= X"00" & data(7 downto 0);
tck_0 <= tck;
tck_1 <= tck;
tdo_0 <= tdo;
tdo_1 <= tdo;
sysres <= NOT nsysres;

inst_jtag: JTAGController
  generic map(base_address, address_increment, addr_high, addr_low)
  port map(
    oTck => tck,
    oTms0 => tms_0,
    oTms1 => tms_1,
    oTdo => tdo,
    iTdi0 => tdi_0,
    iTdi1 => tdi_1,
    vme_addr => addr,
    vme_data => data_in,
--    vme_en => vme_en_jtag,
    vme_en => en_jtag,
    vme_wr => write,
    vme_dtack => dtack,
    vme_data_out => data_jtag,
    clk => clk,
    reset => sysres
);

```

```
tri_loop:
FOR i IN 0 TO 7 GENERATE
  inst_tri: tri
  PORT MAP(data_jtag(i), en_tri, data(i));
END GENERATE tri_loop;

end rtl;
```

```
--------------------------------------------------------------------------------
-- Title      : JTAG Controller
-- Project    :
--------------------------------------------------------------------------------
-- File       : JTAGController.vhd
-- Author     : SAKULIN Hannes  <hsakulin@dsy-srv3.cern.ch>
-- Company    :
--------------------------------------------------------------------------------
-- Description: Simplified version of a ScanPSC100 JTAG controller
--------------------------------------------------------------------------------
-- $Date: 2004/10/25 12:54:54 $
-- $Revision: 1.10 $
--------------------------------------------------------------------------------
-- Revision-Description:
-- generic parameter for address width of vme_addr implemented (HB)
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- based on:

-- File name:    Controller_soc2.vhd %
-- Title:        VHDL Controller Chip for the PHTF Soc2
-- Author:       JEJr. %
-- This VHDL Modul of the Board´s VME Controller
-- Version | Author | Mod. Date | Change |
-----------|--------|-----------|      ------------------------------------------------
-- 1 | JEJr | 31.03.2004 | First design (derived from ETTF_contr_jxx) |
-----------|--------|-----------|      ------------------------------------------------

library IEEE;
library work;

use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
USE IEEE.STD_LOGIC_UNSIGNED.ALL; -- for +- operators
------------------------------------------------------------------------------
-- Entity Declaration
------------------------------------------------------------------------------


entity JTAGController is
  generic (
    base_address      : integer := 0;   -- base address (in bytes)
    address_increment : integer := 2;   -- increment (2 for word access,
                                        -- 4 for long word access)
    addr_high : integer := 3;           -- upper index of vme_addr vector
    addr_low  : integer := 1            -- lower index of vme_addr vector
    );

  port (

    -- JTAG port
    -- to be connected directly to I/O pins of chip
    oTck  : out std_logic;
    oTms0 : out std_logic;
    oTms1 : out std_logic;
    oTdo  : out std_logic;
    iTdi0 : in  std_logic;
    iTdi1 : in  std_logic;

    -- VME port
    vme_addr       : in    std_logic_vector(addr_high downto addr_low);

    vme_data       : in    std_logic_vector(15 downto 0);  -- has to be at least 8 bit
    vme_en         : in    std_logic;   -- should not be a pulse when writing
    vme_wr         : in    std_logic;   -- state. must remain for at least two
                                        -- clocks after enable goes to 0
    vme_dtack      : out   std_logic;         -- not inverted

```

```
    vme_data_out   : out   std_logic_vector(15 downto 0);
    vme_en_out     : out   std_logic;
        
    -- Clock and control
    clk            : in    std_logic;
    reset          : in    std_logic);  -- asynchronous reset, active high
  
end;

-------------------------------------------------------------------------------
-- VME signals:
-- 
-- the unit assumes that data and address are valid for the whole time
-- that vme en is active
--
-- the unit further assumes that vme_en is a synchronous signal (no
-- asynchronous clear) as the falling edge of vme en is used to trigger actions.
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Architecture declaration
-------------------------------------------------------------------------------

architecture behavioral of JTAGController is

  -- need the following attributes to force Synplify to use
  -- input and output flip-flops
  -- (currently does not work for TDI, TMS)
  attribute syn_useioff                : boolean;
  attribute syn_useioff of behavioral : architecture is true;

-- JTAG Registers

  signal mode0_reg : std_logic_vector( 7 downto 0);
  signal mode1_reg : std_logic_vector( 7 downto 0);
  signal mode2_reg : std_logic_vector( 7 downto 0);
  signal cnt32_reg : unsigned (31 downto 0);
  signal tms0_reg  : std_logic_vector( 7 downto 0);
  signal tms1_reg  : std_logic_vector( 7 downto 0);
  signal tdo_reg   : std_logic_vector( 7 downto 0);
  signal tdi_reg   : std_logic_vector( 7 downto 0);


-- JTAG Register Enable Signals

  signal ena       : std_logic;
  signal mode0_ena : std_logic;
  signal mode1_ena : std_logic;
  signal mode2_ena : std_logic;
  signal cnt32_ena : std_logic;
  signal tms0_ena  : std_logic;
  signal tms1_ena  : std_logic;
  signal tdo_ena   : std_logic;
  signal tdi_ena   : std_logic;

  signal tdi_ena_d  : std_logic;
  signal tdi_ena_d2 : std_logic;

  signal tdi_stat : std_logic;

-- JTAG status signals
  signal cnt_loaded     : std_logic;
  signal tdo_empty      : std_logic;
  signal tms0_empty     : std_logic;
  signal tms1_empty     : std_logic;
  signal tdi_full       : std_logic;
  signal cnt_loaded_del : std_logic;
  signal tdo_empty_del  : std_logic;
```

```
  signal tms0_empty_del : std_logic;
  signal tms1_empty_del : std_logic;
  signal tdi_full_del   : std_logic;

  signal cnt_pointer : unsigned ( 1 downto 0);
  signal dtack_ff    : std_logic;

  signal tms0 a exit : std logic;
  signal tms1_a_exit : std_logic;

--DTACK flip-flops

  signal tdo_rdy_ff : std_logic;
  signal tdi rdy ff : std logic;
  signal reg_rdy_ff        : std_logic;
  signal reg_rdy_ff_del    : std_logic;
  signal ds del sig        : std logic;


-- JTAG Registers

  signal jtag_ck_cnt     : unsigned (2 downto 0);
  signal jtag ck         : std logic;
  signal jtag_ck_pulse   : std_logic;

  signal shift enable : std logic;


  signal tms01 sig : std logic;  -- Memorize last TMS action


  signal cnt001 : std logic;


  signal tdo_enable   : std_logic;
  signal tdi_enable   : std_logic;
  signal cnt32_enable : std_logic;
  signal tms0_enable  : std_logic;
  signal tms1_enable  : std_logic;
  signal auto_tms_h   : std_logic;
  signal int_reset    : std_logic;
  signal int_dtack    : std_logic;

  signal cnt_stat      : std_logic;
  signal cnt_stat_del  : std_logic;
  signal tms0_stat     : std_logic;
  signal tms0_stat_del : std_logic;
  signal tms1_stat     : std_logic;
  signal tms1_stat_del : std_logic;
  signal tdo_stat      : std_logic;
  signal tdo_stat_del  : std_logic;

-- JTAG Outputs internal name

  signal stms0 : std_logic;
  signal stms1 : std_logic;
  signal stdo  : std_logic;
  signal stdi0 : std_logic;
  signal stdi1 : std_logic;


  function addr_match (
    constant vme_addr : std_logic_vector;
    constant address  : integer)        -- in bytes
    return boolean is

    variable my_addr_vec : std_logic_vector(vme_addr'high downto 0);
  begin  -- process vme_addr_decode
    my_addr_vec := std_logic_vector( TO_UNSIGNED ( address, vme_addr'high+1 ) );
```

```
    return my_addr_vec(addr_high downto addr_low) = vme_addr(addr_high downto addr_low);
  end;

  signal vme_en_d  : std_logic;

  signal nreset  : std_logic;
  signal vme_cycend_p : std_logic;
  
begin  -- Main

  nreset <= not reset;

  ---------------------------------------------------------------------------
  -- Address Decode
  ---------------------------------------------------------------------------
  

  tdo_ena   <= vme_en when addr_match(vme_addr, base_address)    else '0';
  tdi_ena   <= vme_en when addr_match(vme_addr, base_address+ address_increment)  else '0'
  tms0_ena  <= vme_en when addr_match(vme_addr, base_address+ address_increment*2) else '0
  tms1_ena  <= vme_en when addr_match(vme_addr, base_address+ address_increment*3) else '0
  cnt32_ena <= vme_en when addr_match(vme_addr, base_address+ address_increment*4) else '0
  mode0_ena <= vme_en when addr_match(vme_addr, base_address+ address_increment*5) else '0
  mode1_ena <= vme_en when addr_match(vme_addr, base_address+ address_increment*6) else '0
  mode2_ena <= vme_en when addr_match(vme_addr, base_address+ address_increment*7) else '0

  ena <= mode0_ena
               or mode1_ena
               or mode2_ena
               or ( tdi_ena and tdi_full )
               or cnt32_ena
               or tdo_ena
               or tms0_ena
               or tms1_ena;

  ---------------------------------------------------------------------------
  -- Synchronization
  ---------------------------------------------------------------------------
  
  Synchro : process (clk, nreset)

  begin
    if nreset = '0' then
      vme_cycend_p <= '0';
      vme_en_d <= '0';
    elsif clk'event and clk='1' then
      vme_en_d <= vme_en;
      
      if (( vme_en = '0' ) and ( vme_en_d = '1' )) then
        vme_cycend_p <= '1';
      else
        vme_cycend_p <= '0';
      end if;
    end if;
  end process Synchro;


  ---------------------------------------------------------------------------
  -- Write registers
  ---------------------------------------------------------------------------

  write_reg: process (clk, reset) 
  begin  -- process write_reg
    if reset = '1' then
      mode0_reg              <= "00100000";  --HS
      mode1_reg              <= (others => '0');
      mode2_reg (1 downto 0) <= (others => '0');
    elsif clk'event and clk = '1' then  -- rising clock edge
      if int_reset = '1' then
```

```
        mode0_reg              <= "00100000";  --HS
        mode1_reg              <= (others => '0');
        -- do not reset mode2 as it is being written to
      end if;
      if (vme_wr = '1') then
        if (mode0_ena = '1' and vme_en_d = '0') then mode0_reg <= vme_data(7 downto 0); er
        if (mode1_ena = '1' and vme_en_d = '0') then mode1_reg <= vme_data(7 downto 0); er
        if (mode2_ena = '1' and vme_en_d = '0') then mode2_reg(1 downto 0) <= vme_data(1 d
      end if;
    end if;
  end process write_reg;

-- mode registers are complete
  ------------------------------------------------------------------------------
  -- Connect status signals to registers
  ------------------------------------------------------------------------------

  tdo_enable   <= mode0_reg (7);
  tdi_enable   <= mode0_reg (6);
  cnt32_enable <= mode0_reg (5);
  tms0_enable  <= mode0_reg (4);
  tms1_enable  <= mode0_reg (3);
  auto_tms_h   <= mode0_reg (1);
  -- mode0, bit 0: loopback not implemented.

   
  int_reset <= mode2_reg (1);
  
  

  -- mode2, bit 0: single step not implemented
  -- mode2, bit 2: update status not implemented (status is always updated)
  -- mode2, bit 3: continuous update not implemented (status is always updated)
  
  mode2_reg (7) <= tdo_empty;
  mode2_reg (6) <= tdi_full;
  mode2_reg (5) <= not cnt_loaded;
  mode2_reg (4) <= tms0_empty;
  mode2_reg (3) <= tms1_empty;
  mode2_reg (2) <= shift_enable;  --????


  -- TBD: jtag mode 2 reg reset bit has to return to 0 after reset

  -- mode2, bit0: Single step CNT32 not implemented.

  ------------------------------------------------------------------------------
  -- Read registers
  ------------------------------------------------------------------------------

  vme_data_out <=      ( "00000000" & mode0_reg ) when mode0_ena = '1'
                  else ( "00000000" & mode1_reg ) when mode1_ena = '1'
                  else ( "00000000" & mode2_reg ) when mode2_ena = '1'
                  else ( "00000000" & tdi_reg )   when (( tdi_ena = '1' ) and ( tdi_full =
                  else ( "00000000" & std_logic_vector ( cnt32_reg ( 7 downto 0 )))   when
                  else ( "00000000" & std_logic_vector ( cnt32_reg ( 15 downto 8 )))  when
                  else ( "00000000" & std_logic_vector ( cnt32_reg ( 23 downto 16 ))) when
                  else ( "00000000" & std_logic_vector ( cnt32_reg ( 31 downto 24 ))) when
                  else (others => '0');
  
  vme_en_out <= ena;
 
  ------------------------------------------------------------------------------
  -- DTACK generation
  ------------------------------------------------------------------------------

  -- we do need a special dtack generation:
  --   dtack is delayed if registers are not ready when
  --   reading from tdi or writing to tms0/1, tdo and cnt32.
```

```
  -- HS : do not yet understand ds_synch story .
  Dtack_Synchro : process (clk, nreset)
  begin
    if nreset = '0' then
      reg_rdy_ff   <= '0';
--      ds_del_sig   <= '0';
    elsif clk'event and clk='1' then
--       if ds_synch = "00" then
--         ds_del_sig <= '0';
--       elsif ( not (ds_synch = "00" ) and valid_address = '1' ) then
--         ds_del_sig <= '1';
--       end if;
--       if ds_synch = "00" then
--         reg_rdy_ff   <= '0';
--       elsif ds_del_sig = '1' then
      if ena = '0' then
        reg_rdy_ff   <= '0';
      else
        if ( tdi_ena or tdo_ena or tms0_ena or tms1_ena or cnt32_ena) = '1' then
          reg_rdy_ff <= dtack_ff;
        else
          reg_rdy_ff <= '1';
        end if;
      end if;
    end if;
  end process Dtack_Synchro;


  Dtack_del_proc : process ( clk, nreset)
  begin
    if ( nreset = '0' ) then
      reg_rdy_ff_del <= '0';
--      dtack_del      <= '0';

    elsif clk'event and clk='1' then
      reg_rdy_ff_del <= reg_rdy_ff;
--      dtack_del      <= int_dtack;
    end if;
  end process Dtack_del_proc;

  int_dtack <= reg_rdy_ff_del and ena;

  vme_dtack <= int_dtack;


  -- generate dtack only when register is ready to be read/written

  -- HS: deadlocks: two successive writes to cnt without


  Jtag_dtack : process ( clk, nreset)
  begin
    if ( nreset = '0' ) then
      dtack_ff <= '0';
    elsif clk'event and clk='1'  then
      if ena = '1' then

        if (( cnt32_ena = '1' ) and ( vme_wr = '1' ) and ( cnt_loaded = '0' )) then
          dtack_ff <= '1';
        elsif (( cnt32_ena = '1' ) and ( vme_wr = '0' )) then
```

```
          dtack_ff <= '1';
        end if;

        if (( tdo_ena = '1' ) and ( vme_wr = '1' ) and ( tdo_empty = '1' )) then
          dtack_ff <= '1';
        elsif (( tdo_ena = '1' ) and ( vme_wr = '0' )) then
          dtack_ff <= '1';
        end if;

        if (( tms0_ena = '1' ) and ( vme_wr = '1' ) and ( tms0_empty = '1' )) then
          dtack ff <= '1';
        elsif (( tms0_ena = '1' ) and ( vme_wr = '0' )) then
          dtack_ff <= '1';
        end if;

        if (( tms1_ena = '1' ) and ( vme_wr = '1' ) and ( tms1_empty = '1' )) then
          dtack ff <= '1';
        elsif (( tms1_ena = '1' ) and ( vme_wr = '0' )) then
          dtack_ff <= '1';
        end if;

        if (( tdi_ena = '1' ) and ( vme_wr = '0' ) and ( tdi_full = '1' )) then
          dtack ff <= '1';
        elsif (( tdi_ena = '1' ) and ( vme_wr = '1' )) then
          dtack_ff <= '1';
        end if;

      else
        dtack ff   <= '0';
      end if;


    end if;
  end process Jtag_dtack;


  ----------------------------------
  -- Generate JTAG Clock 40 MHz / 4 = 10 MHz
  ----------------------------------
  JTAG_Clock : process (clk, nreset)

  begin

    if ( nreset = '0' ) then

      jtag_ck_cnt     <= (others => '0');
      jtag_ck         <= '0';
      jtag_ck_pulse   <= '0';


    elsif clk'event and clk='1' then

      if ( shift_enable = '1' ) then
        jtag_ck_cnt <= jtag_ck_cnt + 1;
      end if;

      if ( shift_enable = '0' ) then
        jtag_ck_cnt <= (others => '0');
      end if;

      jtag_ck <= jtag_ck_cnt (2);

      -- pulse on rising JTAG clock
      if (( jtag_ck_cnt (2) = '1' ) and ( jtag_ck = '0' )) then
        jtag_ck_pulse <= '1';
```

```
      else
        jtag_ck_pulse <= '0';
      end if;

    end if;
  end process JTAG_Clock;

  ---------------------------------------------------------------------------
  -- memorize last TMS operation
  ---------------------------------------------------------------------------
  
  TMSmem : process (clk, nreset)

  begin
    if ( nreset = '0' ) then
      tms01_sig   <= '0';
    elsif clk'event and clk='1' then
      if tms0_enable = '1' then
        tms01_sig <= '0';
      elsif tms1 enable = '1' then
        tms01_sig <= '1';
      end if;
    end if;
  end process TMSmem;


  cnt001 <= '1' when ( cnt32_reg = "00000000000000000000000000000001" ) else '0';

  ---------------------------------------------------------------------------
  -- 32 bit counter
  ---------------------------------------------------------------------------
  
  Counter : process (clk, nreset)

  begin

    if ( nreset = '0' ) then

      cnt32_reg      <= (others => '0');
      cnt_pointer    <= (others => '0');
      cnt_stat       <= '0';
      cnt_stat_del   <= '0';
      cnt_loaded     <= '0';
      cnt_loaded_del <= '0';
      tms0_a_exit           <= '0';
      tms1_a_exit           <= '0';

    elsif clk'event and clk='1' then
      if int_reset = '1' then
        cnt32_reg      <= (others => '0');
        cnt_pointer    <= (others => '0');
        cnt_stat       <= '0';
        cnt_stat_del   <= '0';
        cnt_loaded     <= '0';
        cnt_loaded_del <= '0';
        tms0_a_exit           <= '0';
        tms1_a_exit           <= '0';
      end if;

      --
      -- Load counter
      --
      if (( vme_wr = '1' ) and ( cnt32_ena = '1' ) and ( cnt_loaded = '0' )) then
        cnt_stat <= '1';
        case cnt_pointer is
          when "00"   => cnt32_reg ( 7 downto 0 )   <= unsigned ( vme_data ( 7 downto 0 ))
          when "01"   => cnt32_reg ( 15 downto 8 )  <= unsigned ( vme_data ( 7 downto 0 ))
          when "10"   => cnt32_reg ( 23 downto 16 ) <= unsigned ( vme_data ( 7 downto 0 ))
          when "11"   => cnt32_reg ( 31 downto 24 ) <= unsigned ( vme_data ( 7 downto 0 ))
```

```
          when others => cnt32_reg ( 7 downto 0 )   <= unsigned ( vme_data ( 7 downto 0 ))
        end case;
      end if;

      --
      -- Increment pointer at end of VME cycle
      --
      if (( vme cycend p = '1' ) and ( cnt stat = '1' )) then
        cnt_pointer      <= cnt_pointer + 1;
        cnt_stat     <= '0';
        if cnt pointer = "11" then
          cnt_loaded_del <= '1';
        end if;
      end if;

      cnt_loaded <= cnt_loaded_del;

      if cnt_loaded = '1' then
        cnt_pointer <= (others => '0');
      end if;

      --
      -- count down
      --
      if (( jtag_ck_pulse = '1' ) and ( shift_enable = '1' )) then
        cnt32 reg        <= cnt32 reg - 1;
        if cnt001 = '1' then
          cnt_loaded_del <= '0';
        end if;
      end if;

      --
      -- auto TMS high
      -- 
      if (( auto_tms_h = '1') and (cnt001 = '1') and (cnt_loaded = '1' )) then
        if tms01_sig = '0' then
          tms0_a_exit <= '1';
        else
          tms1_a_exit <= '1';
        end if;
      end if;

      if ( cnt32_reg =  "00000000000000000000000000000000" ) then
        tms0_a_exit <= '0';
        tms1_a_exit <= '0';
      end if;

    end if;
  end process Counter;


  ------------------------------------------------------------------------------
  -- TMS 0
  ------------------------------------------------------------------------------
  tms0 : process (clk, nreset)

    variable tms0_bytcnt : unsigned ( 3 downto 0 );

  begin

    if ( nreset = '0' ) then

      tms0_reg       <= (others => '0');
      tms0_empty     <= '1';
      tms0_empty_del <= '1';
      tms0_bytcnt := (others    => '0');
      tms0_stat      <= '0';
      tms0_stat_del  <= '0';

```

```
    elsif clk'event and clk='1' then
      if int_reset = '1' then
        tms0_reg       <= (others => '0');
        tms0_empty     <= '1';
        tms0_empty_del <= '1';
        tms0_bytcnt := (others    => '0');
        tms0_stat      <= '0';
        tms0 stat del  <= '0';
      end if;

      tms0 empty <= tms0 empty del;

      -- load TMS0 register
      if ( vme wr and tms0 ena ) = '1' and tms0 empty = '1' then
        tms0_reg      <= vme_data ( 7 downto 0);
        tms0_stat_del <= '1';
      end if;

      tms0_stat <= tms0_stat_del;

      if (( tms0_stat = '1' ) and ( vme_cycend_p = '1' )) then
        tms0_bytcnt := "1000";
        tms0 empty del <= '0';
        tms0_stat_del  <= '0';
      end if;

      if (( jtag_ck_pulse = '1' ) and ( shift_enable = '1' ) and ( tms0_enable = '1' )) th
        tms0_reg <= '0' & tms0_reg ( 7 downto 1);

        tms0_bytcnt := tms0_bytcnt - 1;

        if tms0 bytcnt = "0000" then
          tms0_empty_del <= '1';
        end if;
      end if;
      if cnt_loaded_del = '0' then
        tms0_empty_del   <= '1';
      end if;
    end if;
  end process tms0;

  stms0 <= tms0_reg (0) or tms0_a_exit;


  ------------------------------------------------------------------------------
  -- TMS 1
  ------------------------------------------------------------------------------
  tms1 : process (clk, nreset)

    variable tms1_bytcnt : unsigned ( 3 downto 0 );

  begin

    if ( nreset = '0' ) then

      tms1_reg       <= (others => '0');
      tms1_empty     <= '1';
      tms1_empty_del <= '1';
      tms1_bytcnt := (others    => '0');
      tms1_stat      <= '0';
      tms1_stat_del  <= '0';

    elsif clk'event and clk='1' then
      if int_reset = '1' then
        tms1_reg       <= (others => '0');
        tms1_empty     <= '1';
        tms1_empty_del <= '1';
        tms1_bytcnt := (others    => '0');
        tms1_stat      <= '0';
```

```
        tms1_stat_del  <= '0';
      end if;

      tms1_empty <= tms1_empty_del;

      if ( vme_wr and tms1_ena ) = '1' and  tms1_empty = '1' then
        tms1_reg      <= vme_data ( 7 downto 0);
        tms1 stat del <= '1';
      end if;

      tms1 stat <= tms1 stat del;

      if (( tms1_stat = '1' ) and ( vme_cycend_p = '1' )) then
        tms1 bytcnt := "1000";
        tms1_empty_del <= '0';
        tms1_stat_del  <= '0';
      end if;
      if (( jtag_ck_pulse = '1' ) and ( shift_enable = '1' ) and ( tms1_enable = '1' )) th
        tms1_reg <= '0' & tms1_reg ( 7 downto 1);

        tms1_bytcnt := tms1_bytcnt - 1;

        if tms1 bytcnt = "0000" then
          tms1_empty_del <= '1';
        end if;
      end if;
      if cnt_loaded = '0' then
        tms1_empty_del   <= '1';
      end if;
    end if;
  end process tms1;

  stms1 <= tms1_reg (0) or tms1_a_exit;

  ------------------------------------------------------------------------------
  -- TDO
  ------------------------------------------------------------------------------
  tdo : process (clk, nreset)

    variable tdo_bytcnt : unsigned ( 3 downto 0 );

  begin

    if ( nreset = '0' ) then

      tdo_reg       <= (others => '0');
      tdo_empty     <= '1';
      tdo_empty_del <= '1';
      tdo_bytcnt := (others    => '0');
      tdo_stat      <= '0';
      tdo_stat_del  <= '0';

    elsif clk'event and clk='1' then
      if int_reset = '1' then
        tdo_reg       <= (others => '0');
        tdo_empty_del <= '1';
        tdo_bytcnt := (others    => '0');
        tdo_stat      <= '0';
        tdo_stat_del  <= '0';
      end if;

      tdo_empty <= tdo_empty_del;

      if ( vme_wr and tdo_ena ) = '1' and tdo_empty = '1' then
        tdo_reg      <= vme_data ( 7 downto 0);
        tdo_stat_del <= '1';
      end if;

      tdo_stat <= tdo_stat_del;
```

```
      if tdo_stat = '1' and vme_cycend_p = '1' then
        tdo_bytcnt := "1000";
        tdo_empty_del <= '0';
        tdo_stat_del  <= '0';
      end if;

      if (( jtag ck pulse = '1' ) and ( shift enable = '1' ) and ( tdo enable = '1')) then
        tdo_reg <= '0' & tdo_reg ( 7 downto 1);

        tdo bytcnt := tdo bytcnt - 1;

        if tdo_bytcnt = "0000" then
          tdo empty del <= '1';
        end if;
      end if;

      if cnt_loaded = '0' then
        tdo_empty_del <= '1';
      end if;
    end if;
  end process tdo;

  stdo <= tdo_reg (0);


  ------------------------------------------------------------------------------
  -- TDI
  ------------------------------------------------------------------------------
  tdi : process (clk, nreset)

    variable tdi bytcnt : unsigned ( 3 downto 0 );

  begin

    if ( nreset = '0' ) then

      tdi_reg      <= (others => '0');
      tdi_full_del <= '0';
      tdi_full     <= '0';
      tdi_bytcnt := "1000";
  --    tdi_ena_d  <= '0';
  --    tdi_ena_d2 <= '0';
      tdi_stat <= '0';
    elsif clk'event and clk='1' then
      if int_reset = '1' then
        tdi_reg  <= (others => '0');
        tdi_full_del <= '0';
        tdi_full <= '0';
        tdi_bytcnt := "1000";
        tdi_stat <= '0';
      end if;

      -- delay tdi_ena so that it is still valid during vme_cycend_p
--      tdi_ena_d  <= tdi_ena;
--      tdi_ena_d2 <= tdi_ena_d;
      -- vme_wr stays on bus longer, do not need to delay

      -- set stat if there is a read on the TDI register
      if (( vme_wr = '0' ) and ( tdi_ena = '1' ) and ( tdi_full = '1' )) then
        tdi_stat <= '1';
      end if;

      -- if stat is set, clear full on end of cycle
      if ( (tdi_stat = '1') and (vme_cycend_p = '1')) then
        tdi_stat <= '0';
        tdi_full_del <= '0';
        tdi_bytcnt := "1000";
      end if;
```

```
      tdi_full <= tdi_full_del;  -- Delay full signal by one clock
      if (( jtag_ck_pulse = '1' ) and ( shift_enable = '1' ) and ( tdi_enable = '1' )) the
        tdi_reg ( 6 downto 0 ) <= tdi_reg ( 7 downto 1 );
        if tms01_sig = '0' then
          tdi_reg ( 7 ) <= stdi0;
        else
          tdi_reg ( 7 ) <= stdi1;
        end if;
        tdi_bytcnt := tdi_bytcnt - 1;
        -- HS: condition unnecessary? would not get here if shift_enable was 0
        if (( tdi_bytcnt = "0000" ) or (shift_enable = '0')) then
          tdi_full_del <= '1';
        end if;
      end if;
      -- stop shifting if at last JTAG clock pulse
      if (( jtag_ck_pulse = '1' ) and ( cnt001 = '1' ) and ( tdi_enable = '1' ))then
        tdi_full_del   <= '1';
      end if;
    end if;
-- HS: why different stop conditions? cnt_loaded, cnt_loaded_del, cnt001
  end process tdi;
  ------------------------------------------------------------------------------
  -- Shift Enable
  ------------------------------------------------------------------------------
  -- HS: shift enable, when counter loaded _AND_ one of the following:
  --     TMS0 enabled and not empty
  --     TMS1 enabled and not empty
  --     TD0 enabled and not empty _AND_ TDI enabled and not full
  -- only works if TDO and TDI are enabled and if TDI is actually read out
  shift_en_proc : process (clk, nreset)
  begin
    if ( nreset = '0' ) then
      shift_enable <= '0';
    elsif clk'event and clk='1' then
      if cnt_loaded = '1' then
        if (( tms0_enable = '1') and ( tms0_empty = '0' )) then
          shift_enable <= '1';
        elsif (( tms1_enable = '1') and ( tms1_empty = '0' )) then
          shift_enable <= '1';
        elsif (( tdo_enable = '1') and ( tdo_empty = '0' )
               and ( tdi_enable = '1') and ( tdi_full = '0' )) then
          shift_enable <= '1';
        else
          shift_enable <= '0';
```

```
        end if;
      else
        shift_enable   <= '0';

      end if;
    end if;
  end process shift_en_proc;

  ------------------------------------------------------------------------------
  -- Test clock generation
  ------------------------------------------------------------------------------

  -- HS: why invert the clock ??
  -- internal operations shpuld happen on falling edge, external ones on rising
  -- edge.

  tclk generation : process (clk, nreset)

  begin

    if ( nreset = '0' ) then

      oTck <= '0';

    elsif clk'event and clk='1' then
      if shift enable = '1' then
        oTck <= not jtag_ck;
      else
        oTck <= '0';
      end if;
    end if;

  end process tclk_generation;



-------------------------

  oTdo   <= stdo;
  oTms0  <= stms0;
  oTms1  <= stms1;
  stdi0 <= iTdi0;
  stdi1 <= iTdi1;                    -- Separate TDI inputs (as long no tri-state at ETTF_F


end behavioral;
```

```
------------------------------------------------------------------
--                                                              --
-- LOGIC CORE: GTL-module vme chip logic                        --
-- MODULE NAME: cnt_8bit                                        --
-- INSTITUTION: Hephy Vienna                                    --
-- DESIGNER: H. Bergauer                                        --
--                                                              --
-- VERSION: V1.0                                                --
-- DATE: 11 2005                                                --
--                                                              --
-- FUNCTIONAL DESCRIPTION:                                      --
-- 8-bit counter                                                --
--                                                              --
------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY cnt_8bit IS
    PORT(
        ld_data : IN    STD_LOGIC_VECTOR(7 DOWNTO 0);
        clk     : IN    STD_LOGIC;
        clk_en  : IN    STD_LOGIC;
        cnt_en  : IN    STD_LOGIC;
        updown  : IN    STD_LOGIC;
        cin     : IN    STD_LOGIC;
        aclr    : IN    STD_LOGIC;
        sclr    : IN    STD_LOGIC;
        aload   : IN    STD_LOGIC;
        sload   : IN    STD_LOGIC;
        cout    : OUT   STD_LOGIC;
        out_data    : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0));
END cnt_8bit;

ARCHITECTURE rtl OF cnt_8bit IS
BEGIN

inst_cnt: lpm_counter
   GENERIC MAP(LPM_WIDTH => 8,
      LPM_TYPE => "LPM_COUNTER")
   PORT MAP(data => ld_data,
        clock => clk,
        clk_en => clk_en,
        cnt_en => cnt_en,
        updown => updown,
        cin => cin,
        aclr => aclr,
        sclr => sclr,
        aload => aload,
        sload => sload,
        cout => cout,
        q => out_data);

END ARCHITECTURE rtl;
```