GRETA
Technical description

Martin Åberg
`martin@fripost.org`

April 1, 2014

Contents

Chapter 1

Introduction

This document is a technical description of the GRETA expansion board for Amiga 500. The intended audience is any person who wants to learn more about how to develop hardware for the Amiga computer in general, and especially the one who wants to develop the GRETA expansion further.

All schematics, PCB layout and VHDL code described in this document are released under the GNU General Public License, version 3. Gerber files are readily available and can be sent to a PCB manufacturer for production. Refer to the schematic files for bill of materials (BOM).

As of January 2014, some features are still to be implemented in VHDL as well as corresponding software device drivers on the Amiga.

Feel free to contact the original author at `martin@fripost.org`. Also feel free and welcome to develop this project further under the current license and report your progress to the author. Cheerful support and cooperation will be provided. :-)

1.1 Background

The Amiga 500 computer, based on the Amiga platform, became popular in the late eighties for combining low-cost, a powerful CPU (Motorola MC68000), advanced multimedia functionality and a well-designed operating system. The Amiga platform was designed to be a basis for business, creative computing, video production, as well as pure game machines of its times. Hardware expandability and modularity was designed into the architecture from start. The auto-configuration protocol, AUTOCONFIG, is an example of this (more on this later).

A set of on-board custom hardware chips were responsible for DMA-powered multimedia functionality including graphics, hardware sprites, memory copying, four channels of digital audio, and more.

Amiga 500 came as standard with memory configurations of 512KiB or 1024KiB. This memory resides on a memory bus which is shared between the main CPU and the Amiga custom chips. When the custom chips are heavily utilized, only a fraction of the memory bus time is available for the CPU to use.

For more information on the Amiga architecture, see [1].

1.2 GRETA Features

GRETA expansion board is a hardware expansion which connects to the expansion bus on the Amiga 500 computer and adds the following features to the system[1].

8 MiB Fast RAM
The RAM is always available to the main CPU. There are no wait states involved so throughput is 3.4 MiB/s for a PAL Amiga 500. This also holds when the on-board peripherals described below are used. It is installed automatically in the system at boot by using the AUTOCONFIG protocol.

micro SD
A uSD flash memory controller is used for non-volatile storage. All accesses are performed by DMA to the 8 MiB Fast RAM. AUTOCONFIG together with custom device drivers loaded from the GRETA board is responsible for allowing booting from uSD memory.

Ethernet controller
A custom network controller that allows for 10/100MBit Ethernet networking. The network controller performs DMA to the 8 MiB Fast RAM.

External I/O
An external I/O pin header allows for use for dirverse use. For example a fast UART can be implemented. Another option is to use it as a debug port implementing the GDB Remote Serial protocol [2].

1.3 Expandability of GRETA

The GRETA expansion board can in principle be the basis of any peripheral suitable for the expansion bus of Amiga 500. For example a graphics or SATA adapter could be implemented by fitting the appropriate controller chip which is directed by the FPGA chip.

Also, with minor modification, the board can be made to fit the ZORRO II bus in Amiga 2000/3000/4000. It could in principle be achieved by modifying the physical connection and rerouting the AUTOCONFIG signals as appropriate.

1.4 Hardware overview

Figure 1.1 illustrates the hardware components constituting the GRETA expansion board. For more details, appendix A contains the electronic schematics the GRETA expansion board. The schematic files are also the hardware documentation. Everything needed to reproduce the hardware is included.

The PCB itself is routed on four layers with the following stack-up from top to bottom: Signal, ground, 3.3V, signal. Great care has been taken to achieve good signal integrity properties. For example, no bus signals or fast switching

[1]As of January 2014, only Fast RAM with AUTOCONFIG is implemented in HDL. Preparation for the other peripherals is done in terms of AUTOCONFIG and DMA access to the SDRAM. All the necessary hardware components are there aswell.

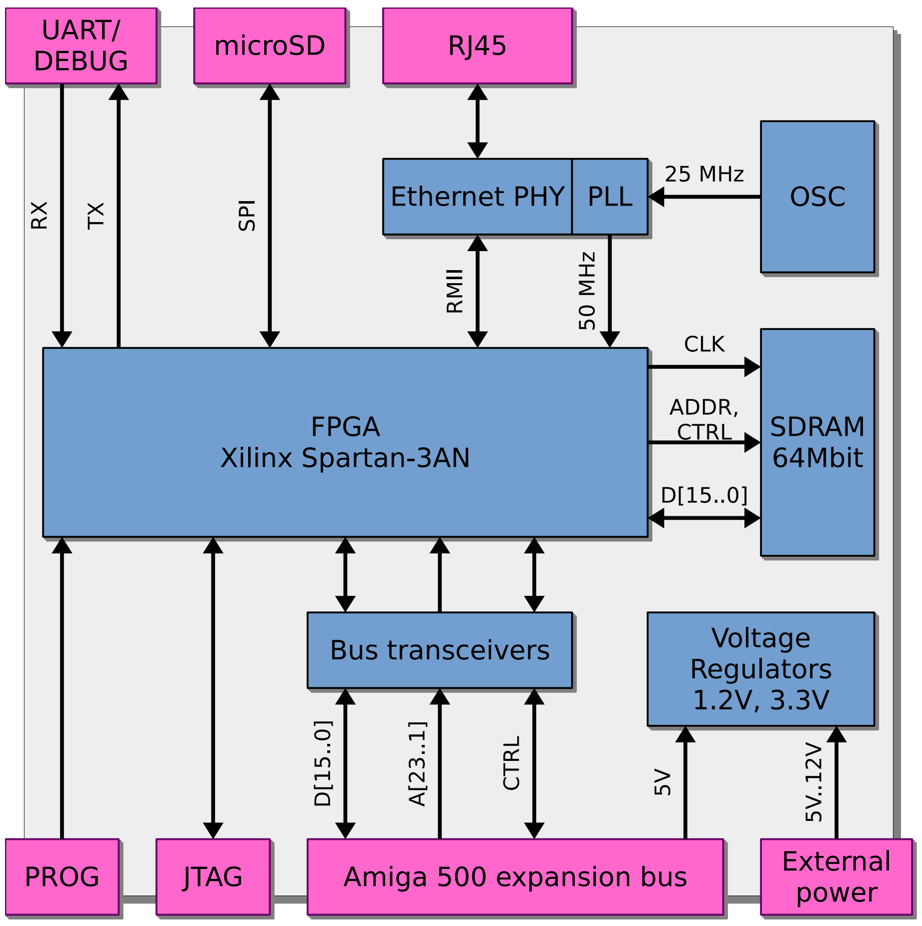


Figure 1.1: Hardware block diagram of the GRETA expansion board. Blue boxes indicate on-board devices. Pink boxes are external connectors.

signals are routed through vias. Routing is tight; only two pins on the FPGA are unconnected.

All components, except for external connectors, are surface mounted. 0603 package is chosen for all resistors and most capacitors.

Figure 1.2 shows the PCB with components mounted and figure 1.3 the expansion board connected to the host computer. Figures 1.4 and 1.5 are images rendered from the Gerber files supplied to the PCB manufacturer.

1.4.1 Notes on the hardware design

The Amiga computer uses 5V TTL signals througout while the FPGA and the other GRETA components are rated at 3.3V and thus some signal level conversion is needed. The solution is to use 5V tolerant drivers for bidirectional signals and unidirectional signals from the computer to the expansion board. A 16-bit bidirectional transceiver, NXP 74LVCH162245A, a component with built-in 30 ohm series termination resistors, solves this. For signals going from the FPGA back to the computer, the device SN74LVC07A is used. It has TTL level tolerant open-drain outputs which fits well with the open collector inputs on the Amiga.

RMII is chosen over MII as Media Independen Interface for the Ethernet PHY. This reduces pin-count. As a bonus, the selected chip has a PLL that can feed the FPGA with a clock signal.

1.5 Tools

The following software tools have been used in the development of the GRETA hardware and HDL.

Arch Linux
Operating system, tools and infrastructure.

KiCad, build (2013-may-18)-stable
Schematic capture and PCB layout.

GNU Make
Dependecy tracking for compiling documentation and HDL code.

GHDL 0.30dev (20100112)
Functional simulation of HDL code.

Xilinx ISE 14.6
Development environment for Xilinx FPGA.

ASM-One V1.20
Macro-assembler and development environment for the Amiga computer.

Git
Revision control.

LibreOffice Draw
Vector graphics.

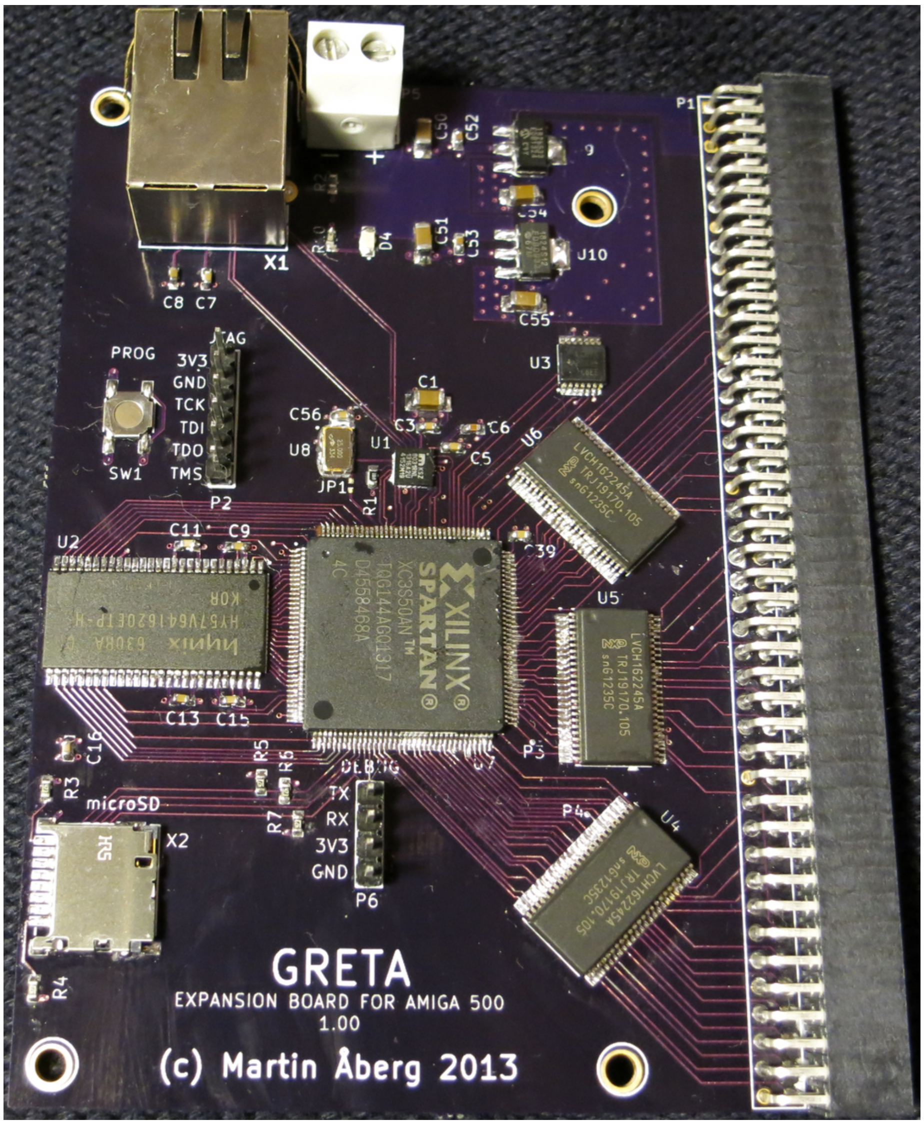


Figure 1.2: Top side of GRETA expansion board. On the right edge is the connector to the Amiga computer expansion bus. The expansion connector has all its signals connected to three 16-bit transceivers and a 6-bit open drain driver.
Down to the left is the microSD card slot and above it the SDRAM. At top left is the RJ45 Ethernet connector and next to it the connector for external power. The large chip in the middle is the Xilinx FPGA. Above it are clock oscillator and Ethernet PHY.
Between the expansion connector and external power are two voltage regulators which generate 1.2V and 3.3V.

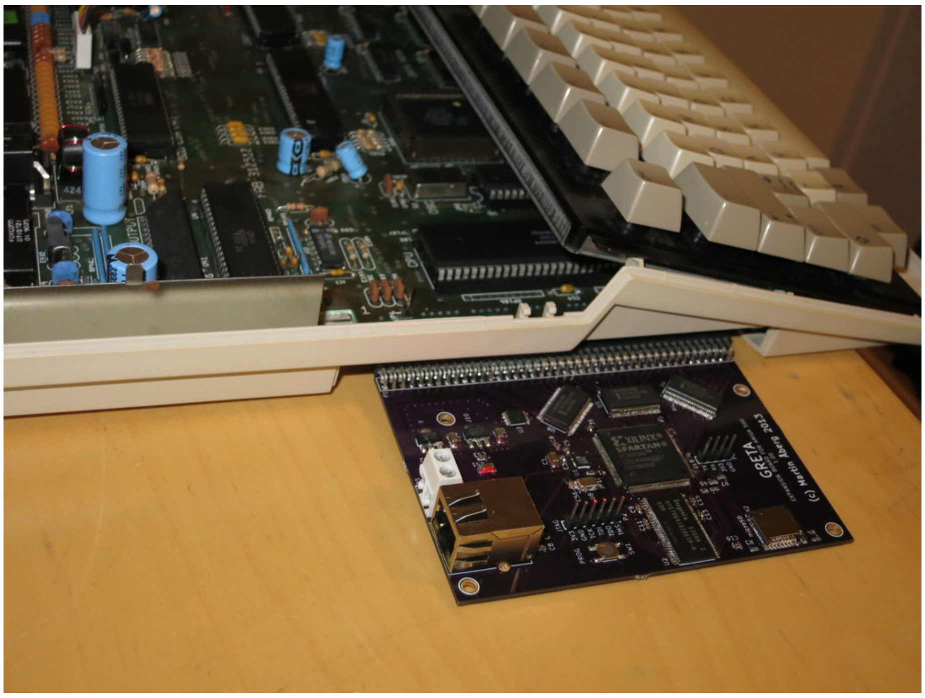


Figure 1.3: The expansion board connects on the left side of the computer. The large chip parallel to the expansion is the Motorola MC68000 CPU.

VIM
: Text editing.

LaTeX
: Typesetting text documents.

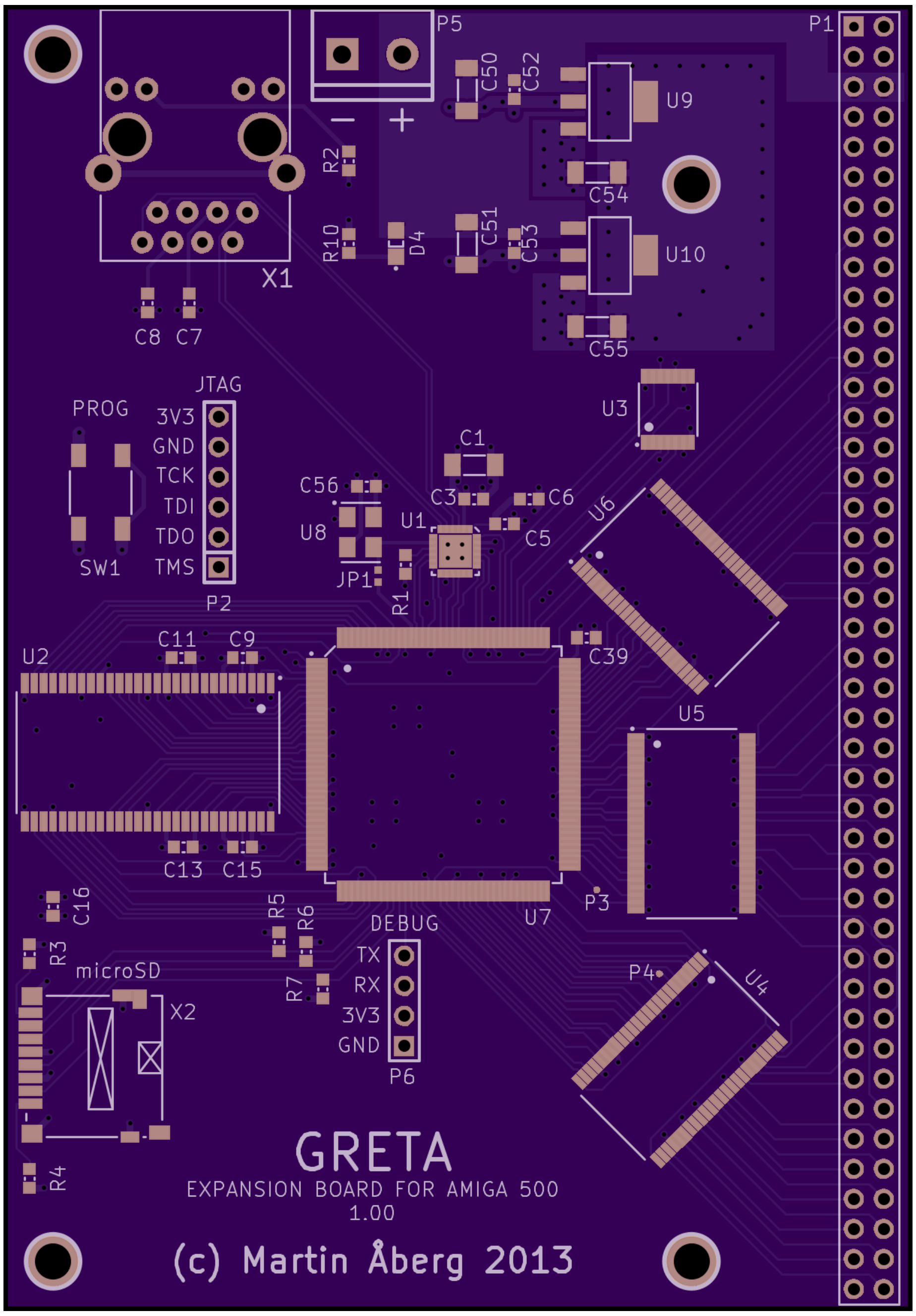


Figure 1.4: PCB top.

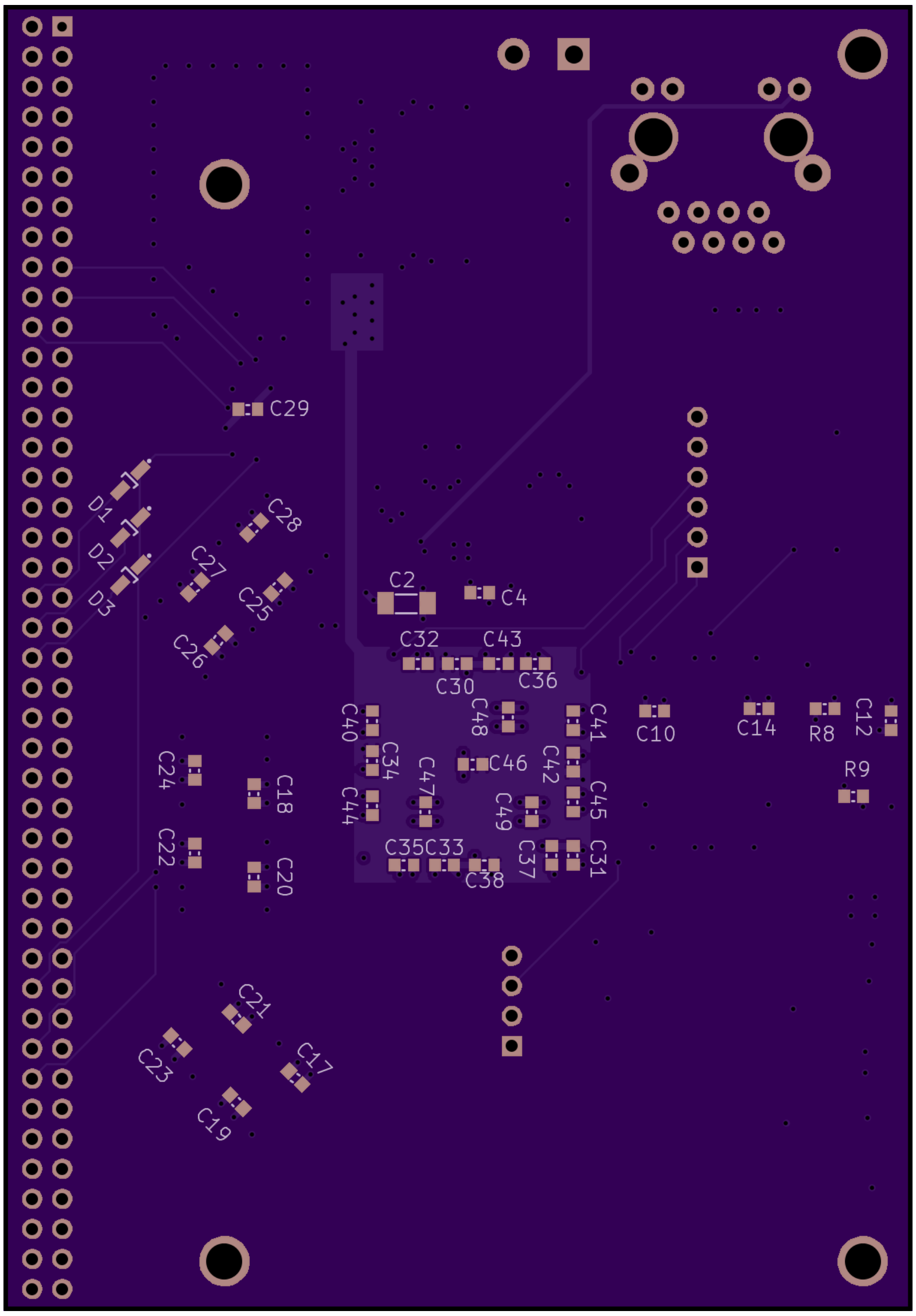


Figure 1.5: PCB bottom.

Chapter 2

HDL Overview

This chapter gives an overview of the GRETA HDL architecture. All VHDL source files are included in appendix B. During the devolopment of the HDL components, a set of test benches were developed. These test benches are not included in the listings in this document. They are however provided in the GRETA file distribution.

The top-level module is contained in `greta.vhdl` with the port definition described in table 2.1. The VHDL package in file `greta_pkg.vhdl` contains type definitions, address constants, bit constants and helper functions for the other HDL modules. Even though several custom types are used in the design, the top level ports only use types defined in `IEEE.STD_LOGIC_1164`.

Figure 2.1 illustrates how the HDL components are related to each other.

A clock synthesizer unit (DCM) in the FPGA is used to generate the 133 MHz clock which is used in the internal logic. It is commonly referred to as the `clk` signal.

2.1 AUTOCONFIG

According to the AUTOCONFIG protocol of the Amiga computer ZORRO II specification [1], a peripheral device can be in one of the states `UNCONFIGURED`, `CONFIGURED` or `SHUT_UP`. After reset, every device is `UNCONFIGURED` and responds to bus accesses in the address space `$E80000-$E8FFFF` if its `nCONFIG_IN` signal is asserted. The system software reads certain registers from the device in this area and eventually issues a write to relocate the device to another address space, typically somewhere inside `$E90000-$EFFFFF` or `$200000-$9FFFFF` depending on size requirements. The device is now `CONFIGURED` and passes its `nCONFIG_OUT` signal to the next device's `nCONFIG_IN`. This procedure is repeated until all devices in the system are configured.

If the system for some reason choses not to configure a device, the device is forced into the `SHUT_UP` state by writing to a dedicated register. In this state the device just passes on its `nCONFIG_OUT` and must never respond again until the next reset.

The AUTOCONFIG space is the address area from `$E80000-$EFFFFF`. Inside this area there are eight equaly sized slots.

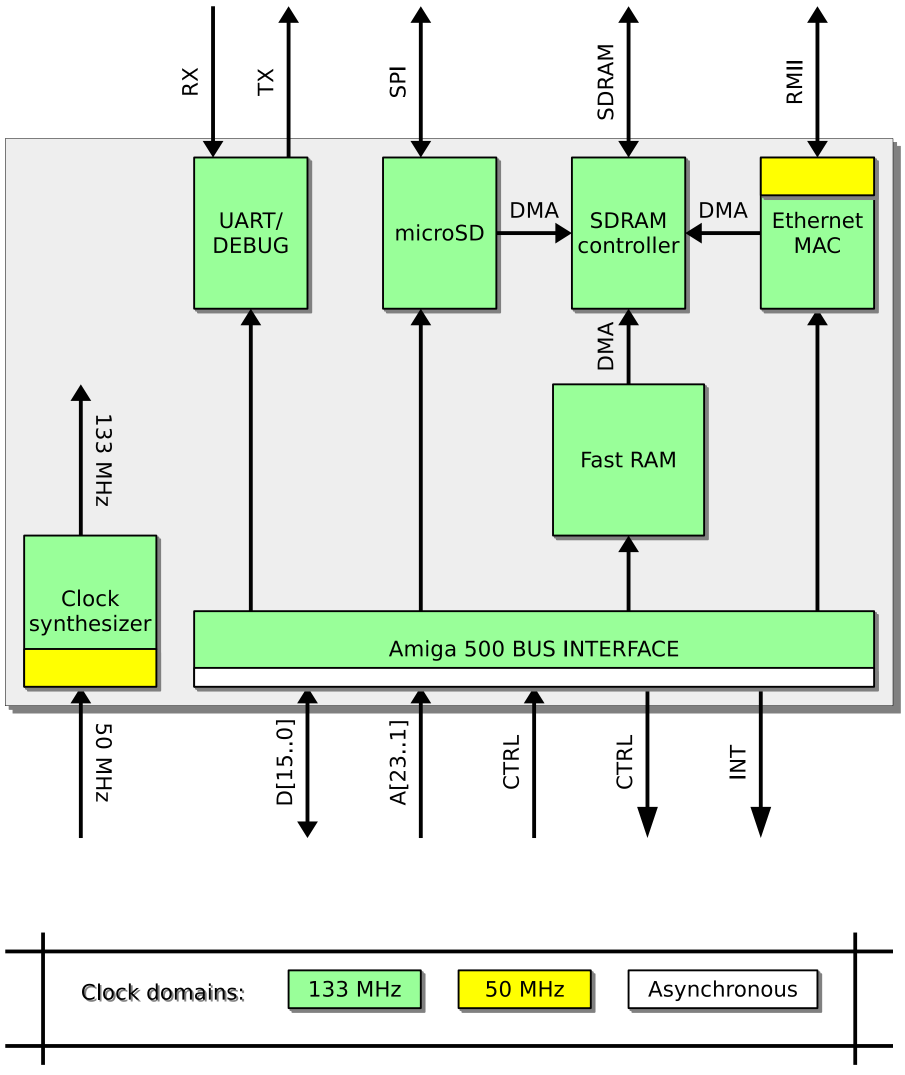


Figure 2.1: HDL block diagram of GRETA. Each box represents one HDL component. All components are implemented using VHDL and are realised in FPGA.

Port	*Direction*	*Description*
CLK25MHZ	in	If JP1 is open then this input is undefined.
		Amiga 500 expansion bus
RL_nRST	in	Reset
RL_nAS	in	Address Strobe
RL_nUDS	in	Data Upper Strobe
RL_nLDS	in	Data Lower Strobe
RL_RnW	in	Read/Write
RL_CDAC	in	Amiga system clock with 90° phase offset
RL_nOVR	out	Override internal generation of nDTACK
RL_nINT2	out	Amiga interrupt level 2
RL_nINT6	out	Amiga interrupt level 6
RL_nINT7	out	Non-maskable interrupt
RL_nDTACK	out	Data Acknowledge
RL_A[23:1]	in	Address bus
RL_D[15:0]	inout	Bidirectinal data bus
RL_D_nOE	out	Output enable for data bus transceiver
RL_D_TO_GRETA	out	Direction of data bus transceiver
		SDRAM interface
SDRAM_CLK	out	Clock
SDRAM_CKE	out	Clock Enable
SDRAM_nRAS	out	Row Address Strobe
SDRAM_nCAS	out	Column Address Strobe
SDRAM_nWE	out	Write Enable
SDRAM_UDQM	out	Upper Data Input/Output Mask
SDRAM_LDQM	out	Lower Data Input/Ouput Mask
SDRAM_BA[1:0]	out	Bank Address
SDRAM_A[11:0]	out	Address
SDRAM_DQ[15:0]	inout	Data bus
		microSD interface (SPI)
SPI_CLK	out	Clock (SCLK)
SPI_nCS	out	Chip Select (SS)
SPI_DO	out	Data Output (MOSI)
SPI_DI	in	Data Input (MISO)
		Ethernet PHY (RMII)
PHY_nRST	out	Reset
PHY_MDC	out	MDIO Clock
PHY_MDIO	inout	MDIO Data I/O
RMII_REF_CLK	in	RMII Clock (50 MHz)
RMII_CRS_DV	in	RMII Carrier Sense/Data Valid
RMII_RXD[1:0]	in	RMII Receive Data
RMII_TXD[1:0]	out	RMII Transmit Data
RMII_TX_EN	out	RMII Transmit Data Enable
		UART/DEBUG interface
DEBUG_RX	in	
DEBUG_TX	out	

Table 2.1: GRETA top level ports.

Definitions and helper functions for the AUTOCONFIG protocol can be found in the VHDL package `autoconfig_pkg`.

Chapter 3

Bus interface

The bus decoder is responsible for decoding asynchronous MC68000 bus signals and exporting a synchronous read/write protocol to the GRETA peripherals.

3.1 Specification

The internal bus protocl is presented in table 3.1.

A pulse on `req` means that `nwe`, `addr` and `wdata` are valid and will remain so for at least 34 `clk` periods[1].

Each GRETA peripheral device holds its `rdata` output valid as long as the peripheral is addressed, and zero otherwise. This allows for all GRETA peripherals to have their rdata outputs OR-wired to the bus interface `rdata` input. An addressed GRETA peripheral must respond to a read request in 34 `clk` periods[2] by asserting its `dev_select` signal. The individual `dev_select` signals are OR:ed together at the top level. The bus interface `dev_selected` is used to determine if the bus transceivers shall drive data towards the Amiga.

The Amiga ZORRO II expansion bus protocol also defines the signal `XRDY` which can be used to delay individual bus accesses. `XRDY` is not used in the GRETA design and is not even connected to the GRETA hardware.

3.2 Implementation

All input signals from the Amiga expansion bus are synchronized before use in the GRETA clock domain. Control signals have two synchronizer stages. Stability of sampled data and address bus signals are assured by the control signals according to the MC68000 bus protocol. Furthermore, all inputs and outputs have their registers in the FPGA IO pads.

[1]To be decided. This is a safe minimum.
[2]To be decided. This is a safe minimum.

Port	*Type*	*Direction*	*Description*
`clk`	`std_logic`	in	GRETA clock
`cpu_reset`	`std_logic`	out	Synchronous reset, activated by Amiga.
			Internal bus interface
`req`	`std_logic`	out	Pulsed once for each 68000 read/write
`nwe`	`bus_nwe`	out	Upper and lower byte select for write. "11" means read.
`addr`	`bus_addr`	out	Word destination for read and write transfers
`wdata`	`bus_data`	out	WRITE data
`rdata`	`bus_data`	in	READ data

Table 3.1: Bus interface, internal interface ports.

Chapter 4

SDRAM Controller

The SDRAM controller offers a user interface to a physical SDRAM chip. Design goals are to provide determinism and an easy to use control interface. Performance demands are set by client access requirements.

All timings in this chapter referes to a *Hynix* SDRAM chip clocked at 133 MHz and CAS latency of three.

4.1 Time resource estimations for SDRAM clients

The access time requirements for each SDRAM client presented in the previous chapter are presented in this section and summarised in Table 4.1.

4.1.1 Fast RAM Controller

For the ZORRO II bus interface on an NTSC Amiga 500, accesses can be performed at most once every fourth 7.16MHz clock cycle [3]. The bus read period and deadlines are derived from [3] and actual signal measurements.

Deadline for a bus read is limited by the time at which Motorola 68000 bus samples read data (falling edge leading to bus cycle state S7). For bus write, the deadline is effectively the time at which a following of bus read may occur. Requests from bus read and bus write can never be active at the same time

Client	*Period*	*Execution time*	*Deadline*
Fast RAM	55	8	34
Disk	85	8	85
MAC TX	213	8	213
MAC RX	213	8	213
Auto refresh	2083	9	2083

Table 4.1: Real time requirements for clients issuing DMA operations on SDRAM. All time values are in units of 133 MHz clock periods, 7.5ns. Read and write execution times are based on single 16 bit word access with CAS latency of 3 cycles. Auto refresh is 9 cycles for this configuration.

so they are combined to one task in Table 4.1 to simplify calculations. As a consequence of this, timing budget for the bus interface is a bit pessimistic.

4.1.2 Disk Controller

A bitrate of 25Mbit/s for Secure Digital over SPI gives a maximum throughput of 2.98MiB/s. The number of 7.5ns periods per 16bit SDRAM access is $85 \leq \frac{16bit}{25Mbit/s} \cdot \frac{1}{7.5ns}$.

4.1.3 Ethernet MAC

Periods and deadlines for MAC RX and MAC TX are based on 10Mbit Ethernet in full-duplex mode. With the presented timing guarantees, only a very narrow FIFO buffer is required to transfer data between the Ethernet PHY clock domain and SDRAM.

The SDRAM access period time for a sustained transfer in each direction, expressed as 7.5ns periods, is $213 \leq \frac{16bit}{10Mbit/s} \cdot \frac{1}{7.5ns}$.

If support for two word bursts is added to the SDRAM controller, then 100Mbit full-duplex can be supported without affecting the real time properties of the other RAM clients. (Though this is possible in theory, the Amiga 500 running a TCP/IP stack with SANA-II driver interface would never handle rates above 10Mbit anyway.)

4.1.4 Auto refresh

The SDRAM requires 64 refresh operation every 4096 ms. In terms of 7.5ns periods, this is $2083 \leq \frac{64ms}{4096} \cdot \frac{1}{7.5ns}$.

4.2 SDRAM Scheduling

To solve the SDRAM scheduling problem, a static off-line approach is used. A time table is pre-calculated, in which each client has its dedicated time slot. An offset pointer iterates over the calendar slots. When the pointer has reached the end of the table it restarts from the top again.

This satisfies the goal of determinism and simplicity. Of course, it must be assured that each client will meet its deadline.

To simplify matters, the accessed bank is always precharged after an access. Furthermore, if a client does not issue a request in its designated time slot, the SDRAM will carry out an unused read operation anyway. Note that self refreshes are scheduled just as an ordinary client.

Also note that scheduling of requests to the SDRAM resource as presented in this section, will result in accesses not being served in the same time order as they were requested. Because of this, any of the peripherals must send its interrupt back to the computer only after an access has been acknowledged by the RAM arbiter. Anyhow, requests made by the same client will always be served in the same order as they were issued.

Table 4.2 illustrates the client calendar. The tightest timing budget is for a *Disk* request arriving at time 10 and finishes at time 89, resulting in a response time of 79 time units.

Offset	*Time*	*Client*
0	0 - 8	Fast RAM
1	9 - 17	Disk
2	18 - 26	Fast RAM
3	27 - 35	MAC TX
4	36 - 44	Fast RAM
5	45 - 53	MAC RX
6	54 - 62	Fast RAM
7	63 - 71	Auto refresh
0	72 - 80	Fast RAM
1	81 - 89	Disk
⋮	⋮	⋮

Table 4.2: Calendar for static scheduling of the SDRAM resource. The unit of time offset is one period of a 133 MHz clock (7.5ns). Calendar has eight positions and repeats after offset 7.

Member	`Type`	*Description*
`req`	`std_logic`	Assert high to issue an SDRAM request.
`nwe`	`bus_nwe`	Upper and lower byte select for write. `"11"` means read.
`addr`	`bus_addr`	Target address for request.
`wdata`	`bus_data`	Write data.

Table 4.3: Record type definition for `ram_bus.` The type is used by SDRAM clients.

4.3 Specification

Each client communicates with the SDRAM controller using an in/out signal pair with types `ram_bus` and `std_logic` respectively. An example is the `fast/fast_ack` pair. All clients share the read data bus `rdata`. See table 4.3 for a description of the `ram_bus` record type.

Transfers are 16 bit wide but for write accesses, it is possible to select target bytes individually with the mask signal `nwe`.

To do an SDRAM request, the client sets its `req` signal high and holds it high until the corresponding `ack` signal goes high (pulse). The record members `nwe`, `addr` and `wdata` must be constant for the duration of asserted `req`. If it was a read request, then `rdata` is valid only in the same clock cycle as `ack` is pulsed.

Table 4.4 illustrates the internal HDL interface to the SDRAM controller.

Port	`Type`	*Direction*	*Description*
`clk`	`std_logic`	in	GRETA clock
`reset`	`std_logic`	in	Synchronous reset
`ready`	`std_logic`	out	Controller is ready for requests
			Fast RAM client
`fast`	`ram_bus`	in	Control structure for accesses by Disk
`fast_ack`	`std_logic`	out	Acknowledge signal for Disk
			microSD client
`disk`	`ram_bus`	in	Control structure for accesses by Disk
`disk_ack`	`std_logic`	out	Acknowledge signal for Disk
			Ethernet MAC client
`mactx`	`ram_bus`	in	Control structure for accesses by MAC TX
`mactx_ack`	`std_logic`	out	Acknowledge signal for MAC TX
`macrx`	`ram_bus`	in	Control structure for accesses by MAC RX
`macrx_ack`	`std_logic`	out	Acknowledge signal for MAC RX
			Common read data bus
`rdata`	`bus_data`	out	Read data, only valid at the corresponding `_ack=1`

Table 4.4: SDRAM component, interface for internal ports.

Chapter 5

Fast RAM Controller

The Fast RAM controller is responsible for implementing the AUTOCONFIG protol for configuring RAM at system start and then responds to Fast RAM accesses in the Amiga address range `$200000-$9FFFFF`.

This Amiga address range needs 23 bits to describe 16-bit word locations. This is a 16MiB range but only 8MiB in this range is used for accessing actual RAM. An important observation is that bit 23 is not significant for addressing to the SDRAM.

In fact, bit 23 can be ignored. This can be seen by splitting the Fast RAM address range into three parts and observe the mappings on SDRAM (table 5.1.

5.1 Specification

The HDL port interface of the Fast RAM component consists of the bus interface and the SDRAM interface as described above.

Amiga address	`SDRAM address`	`Size`
`$200000-$3FFFFF` (bit 23=0)	`$200000-$9FFFFF`	2MiB
`$400000-$7FFFFF` (bit 23=0)	`$400000-$7FFFFF`	4MiB
`$800000-$9FFFFF` (bit 23=1)	`$000000-$1FFFFF`	2MiB

Table 5.1: Mapping of the external RAM in the Amiga address space to SDRAM address space.

Chapter 6

Disk Controller

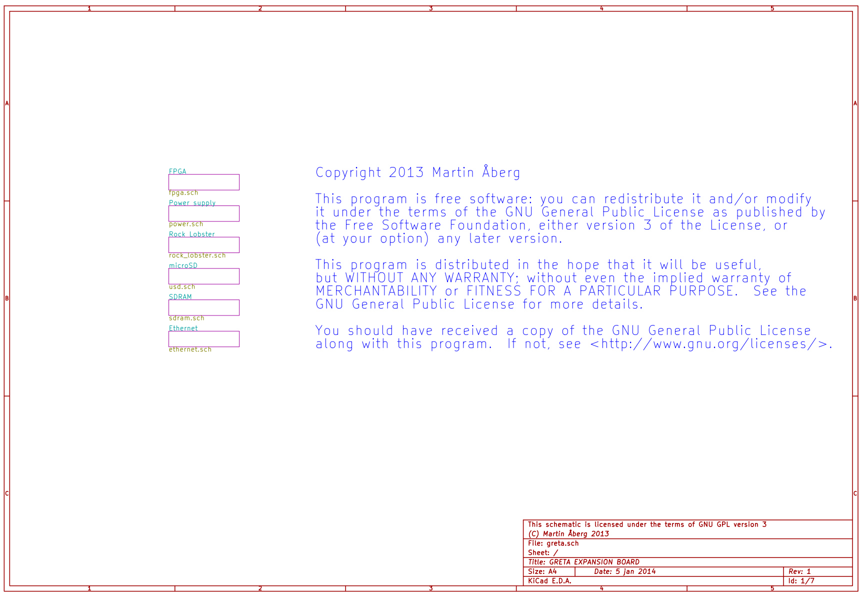
6.1 Specification

6.1.1 User registers

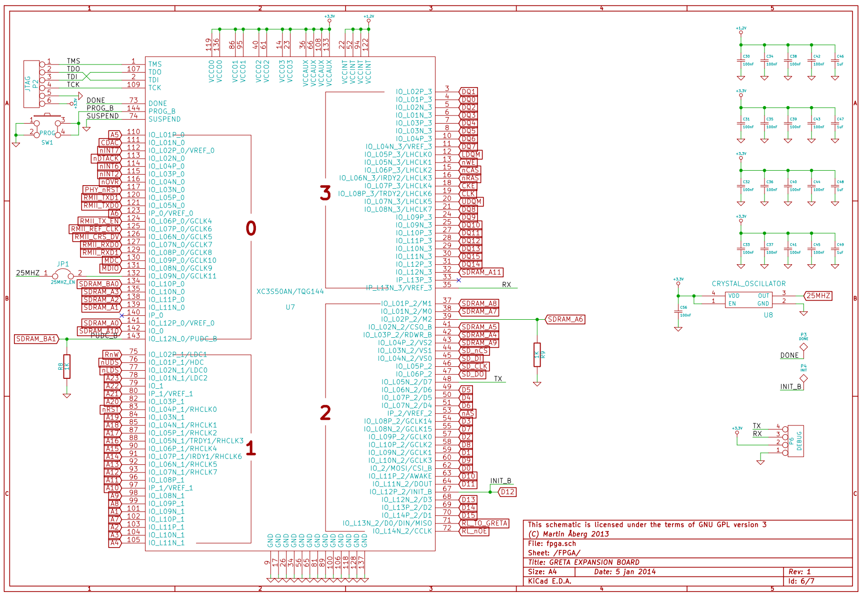
6.2 Implementation

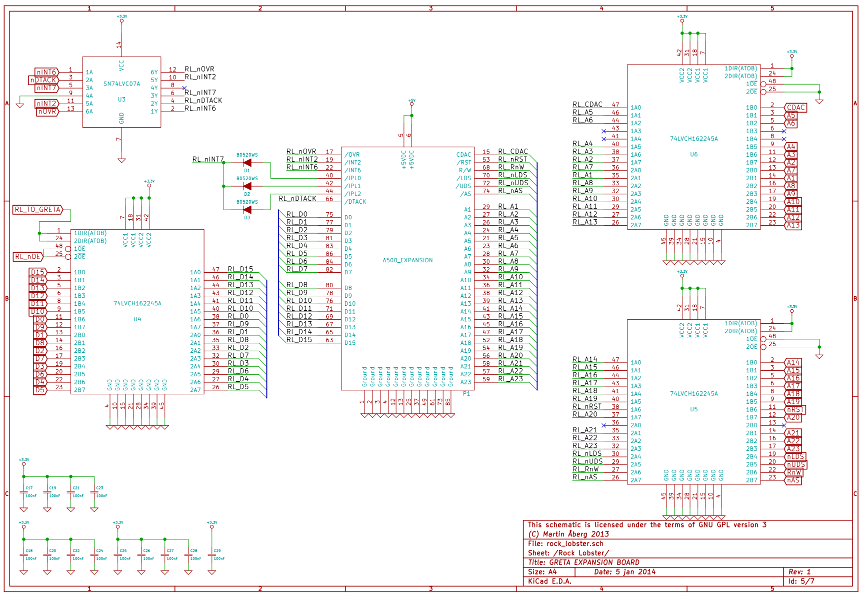
Appendix A

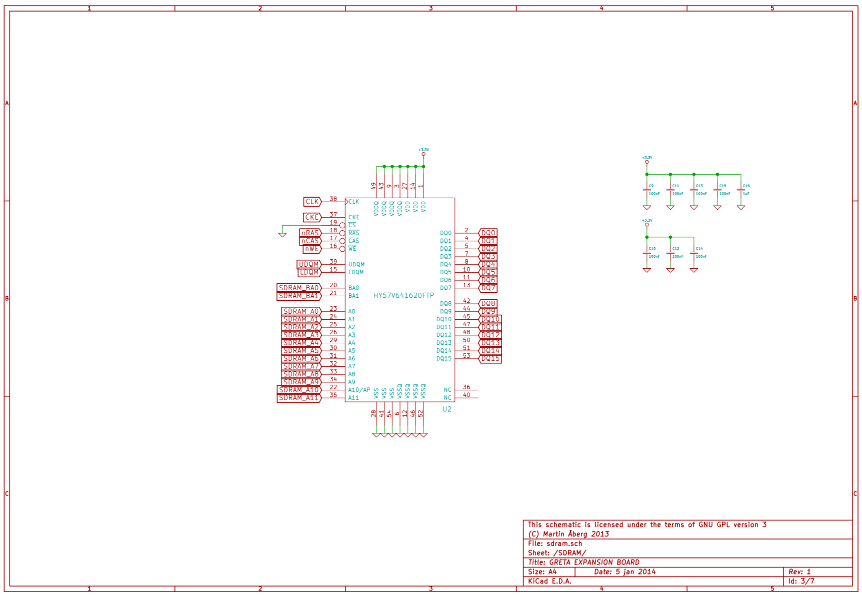
Electronic schematics

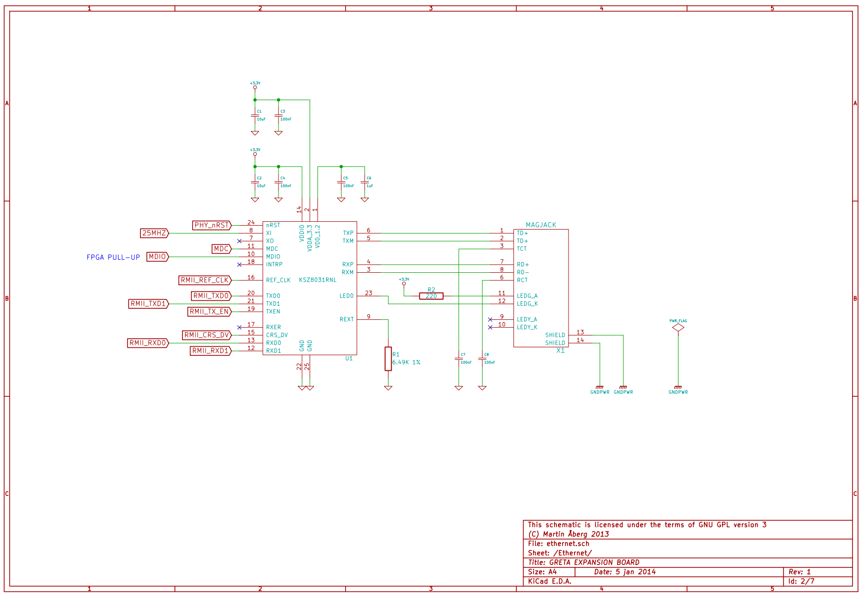


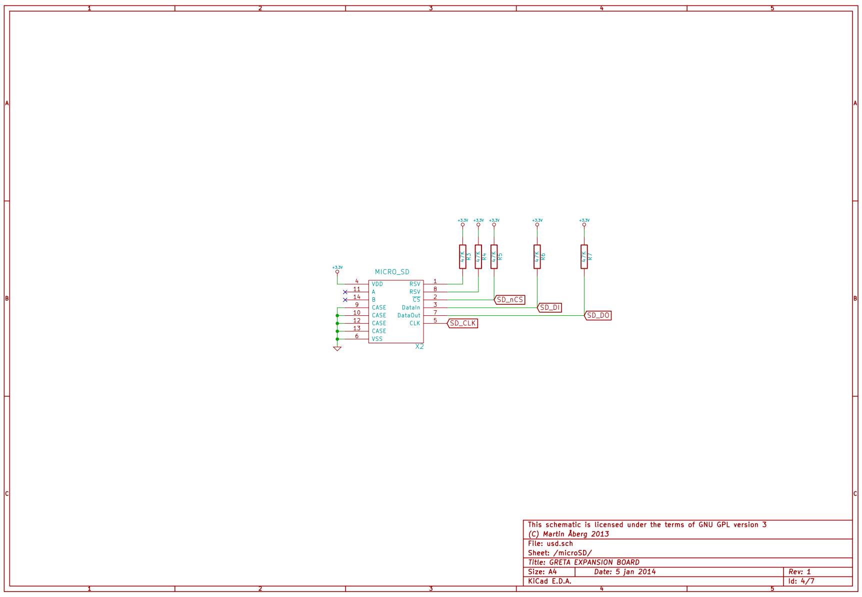
This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: greta.sch
Sheet: /
Title: GRETA EXPANSION BOARD
Size: A4 Date: 5 jan 2014 Rev: 1
KiCad E.D.A. Id: 1/7

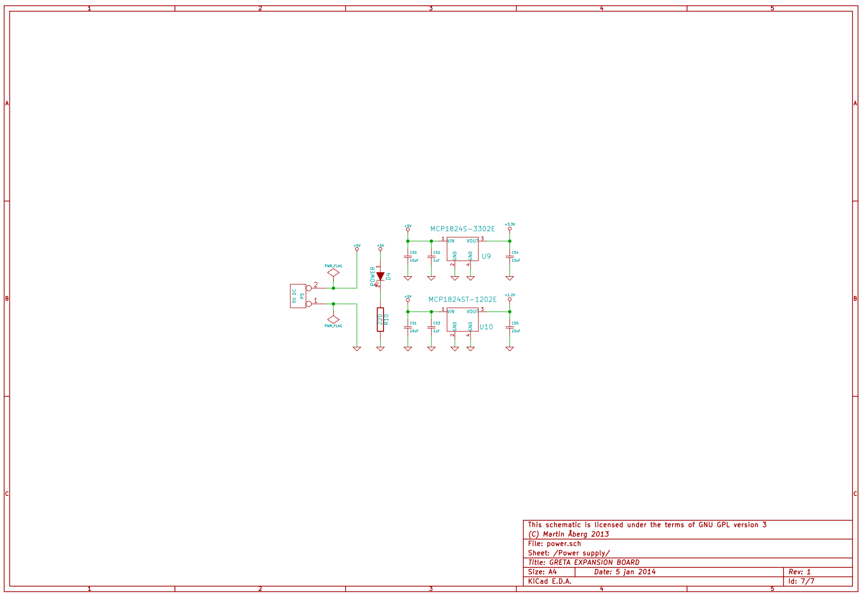

This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: fpga.sch
Sheet: /FPGA/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 6/7
XC3S50AN/TQG144
U7
CRYSTAL_OSCILLATOR
U8
JTAG
PROG
SW1
DEBUG
25MHZ
DONE
INIT_B
TX
RX


This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: rock_lobster.sch
Sheet: /Rock Lobster/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 5/7
SN74LVC07A
U3
74LVCH162245A
U4
U5
U6
A500_EXPANSION
P1
B0520WS
RL_TO_GRETA
RL_nOE


HY57V641620FTP
U2
CLK
CKE
nRAS
nCAS
nWE
UDQM
LDQM
SDRAM_BA0
SDRAM_BA1
SDRAM_A0
SDRAM_A1
SDRAM_A2
SDRAM_A3
SDRAM_A4
SDRAM_A5
SDRAM_A6
SDRAM_A7
SDRAM_A8
SDRAM_A9
SDRAM_A10
SDRAM_A11
DQ0
DQ1
DQ2
DQ3
DQ4
DQ5
DQ6
DQ7
DQ8
DQ9
DQ10
DQ11
DQ12
DQ13
DQ14
DQ15
This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: sdram.sch
Sheet: /SDRAM/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 3/7


KSZ8031RNL
U1
MAGJACK
X1
PHY_nRST
25MHZ
MDC
FPGA PULL-UP
MDIO
RMII_REF_CLK
RMII_TXD0
RMII_TXD1
RMII_TX_EN
RMII_CRS_DV
RMII_RXD0
RMII_RXD1
R1
6.49K 1%
R2
220
GNDPWR
PWR_FLAG
This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: ethernet.sch
Sheet: /Ethernet/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 2/7


MICRO_SD
X2
VDD
A
B
CASE
CASE
CASE
CASE
VSS
RSV
RSV
CS
DataIn
DataOut
CLK
+3.3V
47K R3
47K R4
47K R5
47K R6
47K R7
SD_nCS
SD_DI
SD_DO
SD_CLK
This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: usd.sch
Sheet: /microSD/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 4/7


5V DC P5
PWR_FLAG
PWR_FLAG
+5V
POWER D4
220 R10
MCP1824S-3302E
U9
VIN
VOUT
GND
GND
C50 10uF
C52 1uF
C54 10uF
+3.3V
MCP1824ST-1202E
U10
C51 10uF
C53 1uF
C55 10uF
+1.2V
This schematic is licensed under the terms of GNU GPL version 3
(C) Martin Åberg 2013
File: power.sch
Sheet: /Power supply/
Title: GRETA EXPANSION BOARD
Size: A4
Date: 5 jan 2014
Rev: 1
KiCad E.D.A.
Id: 7/7

Appendix B

HDL code

B.1 GRETA types, constants and functions

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package greta_pkg is
  subtype bus_addr        is std_logic_vector(23 downto  1);
  subtype bus_addr24      is std_logic_vector(23 downto  0);
  subtype autoconfig_slot is std_logic_vector(18 downto 16);
  subtype bus_data        is std_logic_vector(15 downto  0);
  subtype bus_data8       is std_logic_vector( 7 downto  0);
  subtype nibble          is std_logic_vector( 3 downto  0);

  -- Auto-config space. Boards appear here before the system
  -- relocates them to their final address.
  constant AUTOCONFIG_BASE  : bus_addr := x"E8_000" & o"0";
  constant AUTOCONFIG_MASK  : bus_addr := x"F8_000" & o"0";
  constant AUTOCONFIG_SLOT0 : autoconfig_slot := o"0";

  -- Because the Fast RAM area overlaps bit 23, we split it
  -- into three separate areas.
  constant FAST_RAM_BASE2 : bus_addr := x"20_000" & o"0";
  constant FAST_RAM_BASE4 : bus_addr := x"40_000" & o"0";
  constant FAST_RAM_BASE8 : bus_addr := x"80_000" & o"0";
  constant FAST_RAM_MASK2 : bus_addr := x"E0_000" & o"0";
  constant FAST_RAM_MASK4 : bus_addr := x"C0_000" & o"0";
  constant FAST_RAM_MASK8 : bus_addr := x"E0_000" & o"0";

  constant UPPER  : natural := 1;
  constant LOWER  : natural := 0;

  subtype bus_nwe is std_logic_vector(UPPER downto LOWER);

  constant WRITE_UPPER  : bus_nwe := (UPPER => '0', LOWER => '1');
  constant WRITE_LOWER  : bus_nwe := (UPPER => '1', LOWER => '0');
  constant WRITE_WORD   : bus_nwe := (UPPER => '0', LOWER => '0');
  constant READ_WORD    : bus_nwe := (UPPER => '1', LOWER => '1');

  type ram_bus is
  record
    req    : std_logic;
    nwe    : bus_nwe;
    addr   : bus_addr;
    wdata  : bus_data;
  end record;

  -- Note that use of the ec_BaseAddress register is
  -- tricky.  The system will actually write twice. First
  -- the low order nybble is written to the ec_BaseAddress
  -- register+2 (D15-D12). Then the entire byte is written
  -- to ec_BaseAddress (D15-D8). This allows writing of a
  -- byte-wide address to nybble size registers.
  constant ec_BaseAddress : natural := 16#48#;
  constant ec_ShutUp      : natural := 16#4C#;

  constant FAST_PRODUCT_NUMBER  : std_logic_vector := x"1";
  constant DISK_PRODUCT_NUMBER  : std_logic_vector := x"2";
  constant ERT_ZORROII          : nibble := "1100";
  constant ERT_MEMLIST          : nibble := "0010";
  constant ERT_DIAGVALID        : nibble := "0001";

  constant ERT_CHAINEDCONFIG    : nibble := "1000";
  constant ERT_MEMSIZE_8MB      : nibble := "0000";
  constant ERT_MEMSIZE_64K      : nibble := "0001";

  constant ERF_MEMSPACE         : nibble := "1000";
  constant ERF_NOSHUTUP         : nibble := "0100";

  -- A special "hacker" Manufacturer ID number is reserved
  -- for test use: 2011 ($7DB).  When inverted this will look
  -- like $F824.
  constant HACKER_MANUFACTURER  :
   std_logic_vector(15 downto 0) := x"07DB";

  function is_autoconfig_reg(n : natural; a : bus_addr)
    return boolean;

  function is_autoconfig(a : bus_addr)
    return boolean;

  function get_autoconfig_slot(a : bus_addr)
    return autoconfig_slot;

  function is_fast_ram(a : bus_addr)
    return boolean;
```

```
end;

package body greta_pkg is

  function is_autoconfig_reg(n : natural; a : bus_addr)
    return boolean is
    variable reg_addr : std_logic_vector(6 downto 0);
  begin
    reg_addr := std_logic_vector(to_unsigned(n, reg_addr'length));
    return reg_addr(6 downto 1) = a(6 downto 1);
  end;

  function is_autoconfig(a : bus_addr)
    return boolean is
  begin
    return (a and AUTOCONFIG_MASK) = AUTOCONFIG_BASE;
  end;

  function get_autoconfig_slot(a : bus_addr)
    return autoconfig_slot is
  begin
    return a(autoconfig_slot'range);
  end;

  function is_fast_ram(a : bus_addr)
    return boolean is
  begin
    return (
      ((a and FAST_RAM_MASK2) = FAST_RAM_BASE2) or
      ((a and FAST_RAM_MASK4) = FAST_RAM_BASE4) or
      ((a and FAST_RAM_MASK8) = FAST_RAM_BASE8)
    );
  end;

end;
```

B.2 GRETA

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--
library ieee;
use ieee.std_logic_1164.all;
use work.greta_pkg.all;

entity greta is
  port(
    --- EXTERNAL CLOCK
    CLK25MHZ        : in    std_logic;

    --- ROCK LOBSTER
    RL_nRST         : in    std_logic;
    RL_nAS          : in    std_logic;
    RL_nUDS         : in    std_logic;
    RL_nLDS         : in    std_logic;
    RL_RnW          : in    std_logic;
    RL_CDAC         : in    std_logic;
    RL_nOVR         : out   std_logic;
    RL_nINT2        : out   std_logic;
    RL_nINT6        : out   std_logic;
    RL_nINT7        : out   std_logic;
    RL_nDTACK       : out   std_logic;
    RL_A            : in    std_logic_vector(23 downto 1);
    RL_D            : inout std_logic_vector(15 downto 0);
    RL_D_nOE        : out   std_logic;
    RL_D_TO_GRETA   : out   std_logic;

    --- SDRAM
    SDRAM_CLK       : out   std_logic;
    SDRAM_CKE       : out   std_logic;
    SDRAM_nRAS      : out   std_logic;
    SDRAM_nCAS      : out   std_logic;
    SDRAM_nWE       : out   std_logic;
    SDRAM_UDQM      : out   std_logic;
    SDRAM_LDQM      : out   std_logic;
    SDRAM_BA        : out   std_logic_vector(1 downto 0);
    SDRAM_A         : out   std_logic_vector(11 downto 0);
    SDRAM_DQ        : inout std_logic_vector(15 downto 0);

    --- SECURE DIGITAL (SPI)
    SPI_CLK         : out   std_logic;
    SPI_nCS         : out   std_logic;
    SPI_DO           : out   std_logic;
    SPI_DI           : in    std_logic;

    --- ETHERNET PHY (RMII)
    PHY_nRST         : out   std_logic;
    PHY_MDC          : out   std_logic;
    PHY_MDIO         : inout std_logic;
    RMII_REF_CLK     : in    std_logic;
    RMII_CRS_DV      : in    std_logic;
    RMII_RXD         : in    std_logic_vector(1 downto 0);
    RMII_TXD         : in    std_logic_vector(1 downto 0);
    RMII_TX_EN       : out   std_logic;

    --- UART/DEBUG
    DEBUG_RX         : in    std_logic;
    DEBUG_TX         : out   std_logic
  );
end;

architecture structural of greta is
  signal clk               : std_logic;
  signal clk180            : std_logic;
  signal cpu_reset         : std_logic;
  signal dcm_locked        : std_logic;
  signal req               : std_logic;
  signal nwe               : bus_nwe;
  signal addr              : bus_addr;
  signal wdata             : bus_data;
  signal rdata             : bus_data;

  signal fast_select       : std_logic;
  signal fast_rdata        : bus_data;
  signal fast_config_in    : std_logic;
  signal fast_config_out   : std_logic;

  signal sdram_rdata       : bus_data;

  signal sdram_fast        : ram_bus := (
    req => '0',
    nwe => READ_WORD,
    addr => (others => '0'),
    wdata => (others => '0')
  );
  signal sdram_fast_ack    : std_logic := '0';

  signal disk_select       : std_logic;
  signal disk_rdata        : bus_data;
  signal disk_config_in    : std_logic;
  signal disk_config_out   : std_logic;

  signal sdram_disk        : ram_bus := (
    req => '0',
    nwe => READ_WORD,
    addr => (others => '0'),
    wdata => (others => '0')
  );
  signal sdram_disk_ack    : std_logic := '0';

  signal dev_select        : std_logic;

begin
```

```
-- debug outputs
DEBUG_TX <= disk_config_out;

dev_select <= fast_select or disk_select;
fast_config_in <= '1';
disk_config_in <= fast_config_out;
rdata <= fast_rdata or disk_rdata;

-- Enable PLL.
PHY_nRST <= '1';
PHY_MDC <= 'Z';
RMII_TX_EN <= '0';

-- Generate 133.3 MHz clock from 50 MHz PHY clock.
dcm_0 : entity work.dcm_greta
port map(
  CLKIN_IN        => RMII_REF_CLK,
  CLKFX_OUT       => clk,
  CLKFX180_OUT    => clk180,
  CLKIN_IBUFG_OUT => open,
  LOCKED_OUT      => dcm_locked
);

-- SDRAM clock output uses a DDR output buffer. See Xilinx
-- documentation for generic and port descriptions.
ODDR2_inst : ODDR2
generic map(
  DDR_ALIGNMENT => "NONE",
  INIT => '0',
  SRTYPE => "SYNC")
port map (
  Q   => SDRAM_CLK,
  C0  => clk,
  C1  => clk180,
  CE  => '1',
  D0  => '0',
  D1  => '1',
  R   => '0',
  S   => '0'
);

bus_interface_0 : entity work.bus_interface
port map(
  clk             => clk,
  cpu_reset       => cpu_reset,
  dcm_locked      => dcm_locked,

  dev_select      => dev_select,
  req             => req,
  nwe             => nwe,
  addr            => addr,
  wdata           => wdata,
  rdata           => rdata,

  --- ROCK LOBSTER
  nRST            => RL_nRST,
  nAS             => RL_nAS,
  nUDS            => RL_nUDS,
  nLDS            => RL_nLDS,
  RnW             => RL_RnW,
  CDAC            => RL_CDAC,
  nOVR            => RL_nOVR,
  nINT2            => RL_nINT2,
  nINT6            => RL_nINT6,
  nINT7            => RL_nINT7,
  nDTACK           => RL_nDTACK,
  A                => RL_A,
  D                => RL_D,
  D_nOE            => RL_D_nOE,
  D_TO_GRETA       => RL_D_TO_GRETA
);

sdram_0 : entity work.sdram
port map(
  clk              => clk,
  reset            => cpu_reset,
  ready            => open,

  --- CLIENTS
  disk             => sdram_disk,
  disk_ack         => sdram_disk_ack,
  fast             => sdram_fast,
  fast_ack         => sdram_fast_ack,
  rdata            => sdram_rdata,

  --- SDRAM
  SDRAM_CKE        => SDRAM_CKE,
  SDRAM_nRAS       => SDRAM_nRAS,
  SDRAM_nCAS       => SDRAM_nCAS,
  SDRAM_nWE        => SDRAM_nWE,
  SDRAM_UDQM       => SDRAM_UDQM,
  SDRAM_LDQM       => SDRAM_LDQM,
  SDRAM_BA         => SDRAM_BA,
  SDRAM_A          => SDRAM_A,
  SDRAM_DQ         => SDRAM_DQ
);

fast_0 : entity work.fast
port map(
  clk          => clk,
  reset        => cpu_reset,

  req          => req,
  nwe          => nwe,
  dev_select   => fast_select,
  addr         => addr,
  wdata        => wdata,
  rdata        => fast_rdata,
  config_in    => fast_config_in,
  config_out   => fast_config_out,

  ram          => sdram_fast,
  ram_ack      => sdram_fast_ack,
  ram_rdata    => sdram_rdata
);

disk_0 : entity work.disk
port map(
  clk          => clk,
  reset        => cpu_reset,

  req          => req,
  nwe          => nwe,
  dev_select   => disk_select,
  addr        => addr,
  wdata       => wdata,
  rdata       => disk_rdata,
  config_in   => disk_config_in,
  config_out  => disk_config_out,

  ram         => sdram_disk,
  ram_ack     => sdram_disk_ack,
  ram_rdata   => sdram_rdata,

  SPI_CLK     => SPI_CLK,
  SPI_nCS     => SPI_nCS,
  SPI_DO      => SPI_DO,
  SPI_DI      => SPI_DI
);

end;
```

B.3 Bus Interface

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--

-- This component is responsible for translating asynchronous
-- MC68000 bus accesses to a protocol suitable for the
-- AUTOCONFIG component and the other upstream components.
library ieee;
use ieee.std_logic_1164.all;
use work.greta_pkg.all;

entity bus_interface is
  port(
    --- GRETA
    clk             : in    std_logic;
    cpu_reset       : out   std_logic;
    dcm_locked      : in    std_logic;
    req             : out   std_logic;
    nwe             : out   bus_nwe;
    addr            : out   bus_addr;
    wdata           : out   bus_data;
    rdata           : in    bus_data;
    dev_select      : in    std_logic;
    --- ROCK LOBSTER
    nRST            : in    std_logic;
    nAS             : in    std_logic;
    nUDS            : in    std_logic;
    nLDS            : in    std_logic;
    RnW             : in    std_logic;
    CDAC            : in    std_logic;
    nOVR            : out   std_logic;
    nINT2           : out   std_logic;
    nINT6           : out   std_logic;
    nINT7           : out   std_logic;
    nDTACK          : out   std_logic;
    A               : in    bus_addr;
    D               : inout bus_data;
    D_nOE           : out   std_logic := '1';
    D_TO_GRETA      : out   std_logic := '1'
  );
end;

architecture rtl of bus_interface is
  signal nRST_sync        : std_logic_vector(1 downto 0) := "00";
  signal nAS_sync        : std_logic_vector(1 downto 0) := "11";
  signal nUDS_sync       : std_logic_vector(2 downto 0);
  signal nLDS_sync       : std_logic_vector(2 downto 0);
  signal RnW_sync        : std_logic_vector(1 downto 0);

  signal cpu_reset_int : std_logic := '1';

  type bus_state is (S1, W1, W2, R1);
  signal state : bus_state := S1;

begin

  synchronizers : process(clk)
  begin
    if rising_edge(clk) then
      nRST_sync   <= nRST_sync(0) & nRST;
      nAS_sync    <= nAS_sync(0) & nAS;
      RnW_sync    <= RnW_sync(0) & RnW;
      nUDS_sync   <= nUDS_sync(1 downto 0) & nUDS;
      nLDS_sync   <= nLDS_sync(1 downto 0) & nLDS;

      wdata   <= D;
      addr    <= A;

      -- NOTE: clk may be in any state, so the following logic
      -- may get trashed...
      if (dcm_locked = '1') and (nRST_sync(1) = '1') then
        cpu_reset_int <= '0';
      else
        cpu_reset_int <= '1';
      end if;
    end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      if cpu_reset_int = '1' then
        D_nOE       <= '1';
        D_TO_GRETA  <= '1';
        req <= '0';
        state <= S1;
      else
        D_nOE       <= '1';
        D_TO_GRETA  <= '1';
        req <= '0';
        D <= "ZZZZZZZZZZZZZZZZ";

        case state is
        when S1 =>
          if nAS_sync(1) = '0' then
            -- Transfer
            if (RnW_sync(1) = '0') then
              -- Write
              state <= W1;
            else
              -- Read, wait for nUDS and nLDS to become stable.
              if (nUDS_sync(1) = '0') or (nLDS_sync(1) = '0') then
                req <= '1';
                nwe <= READ_WORD;
                D_TO_GRETA <= '0';
                state <= R1;
              end if;
            end if;
          end if;

        when R1 =>
          D_TO_GRETA <= '0';
          D_nOE <= not dev_select;
          D <= rdata;
          if (nUDS_sync(1) = '1') and (nLDS_sync(1) = '1') then
            state <= S1;
            D_nOE <= '1';
            D <= "ZZZZZZZZZZZZZZZZ";
          end if;

        when W1 =>
          D_nOE <= '0';
          -- Wait for nUDS and nLDS to become stable.
          if ((nUDS_sync(2) = nUDS_sync(1))) and
             ((nLDS_sync(2) = nLDS_sync(1))) and
             ((nUDS_sync(2) = '0') or (nLDS_sync(2) = '0')) then
            nwe <= nUDS_sync(1) & nLDS_sync(1);
            req <= '1';
            -- wdata and addr is driven elsewhere.
            state <= W2;
          end if;

        when W2 =>
          D_nOE <= '0';
          if nAS_sync(1) = '1' then
            D_nOE <= '1';
            state <= S1;
          end if;

        end case;

      end if;
    end if;
  end process;

  cpu_reset <= cpu_reset_int;

  nOVR        <= '1';
  nINT2       <= '1';
  nINT6       <= '1';
  nINT7       <= '1';
  nDTACK      <= '1';

end;
```

B.4 SDRAM Controller

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--

-- RAM arbiter and controller
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.greta_pkg.all;

entity sdram is
  port(
    clk         : in    std_logic;
    reset       : in    std_logic;
    ready       : out   std_logic := '0';

    --- CLIENTS
    fast        : in    ram_bus;
    fast_ack    : out   std_logic;
    disk        : in    ram_bus;
    disk_ack    : out   std_logic;
    rdata       : out   bus_data;

    --- SDRAM
    SDRAM_CKE   : out   std_logic;
    SDRAM_nRAS  : out   std_logic;
    SDRAM_nCAS  : out   std_logic;
    SDRAM_nWE   : out   std_logic;
    SDRAM_UDQM  : out   std_logic;
    SDRAM_LDQM  : out   std_logic;
    SDRAM_BA    : out   std_logic_vector( 1 downto  0);
    SDRAM_A     : out   std_logic_vector(11 downto  0);
    SDRAM_DQ    : inout std_logic_vector(15 downto  0) :=
     (others => 'Z')
  );
end;

architecture rtl of sdram is
  constant NCLIENTS : integer := 2;
  constant AP : integer := 10;

  subtype init_counter_t is unsigned(14 downto 0);
  signal init_counter : init_counter_t := (others => '0');
  subtype tick_t is unsigned(3 downto 0);
  signal tick : tick_t := (others => '0');

  subtype offset_t is unsigned(1 downto 0);
  signal offset : offset_t := (others => '0');

  type state_t is (INIT_NOP, INIT_PRE, INIT_AR, INIT_MRS,
   INIT_DONE);
  signal state : state_t := INIT_NOP;

  -- nRAS, nCAS, nWE
  subtype cmd_t is std_logic_vector(2 downto 0);
  constant CMD_MRS     : cmd_t := "000";
  constant CMD_AR      : cmd_t := "001";
  constant CMD_PRE     : cmd_t := "010";
  constant CMD_ACT     : cmd_t := "011";
  constant CMD_WRITE   : cmd_t := "100";
  constant CMD_READ    : cmd_t := "101";
  constant CMD_NOP     : cmd_t := "111";
  signal cmd           : cmd_t := CMD_NOP;

  signal addr : bus_addr := (others => '0');
  signal nwe : bus_nwe := READ_WORD;
  signal refresh : std_logic := '0';
  signal acks : std_logic_vector(NCLIENTS - 1 downto 0) :=
    (others => '0');
  signal acks_pend : std_logic_vector(NCLIENTS - 1 downto 0) :=
    (others => '0');
  signal wdata : bus_data;

begin
--  psl default clock is rising_edge(clk);
--  psl assert
--    always({state = INIT_DONE and fast.req='1' and acks(0) = '0'} |=>
--      {[*0 to 26]; acks(0)='1'});
--  psl assert
--    always({state = INIT_DONE and disk.req='1' and acks(1) = '0'} |=>
--      {[*0 to 44]; acks(1)='1'});
  SDRAM_nRAS  <= cmd(2);
  SDRAM_nCAS  <= cmd(1);
  SDRAM_nWE   <= cmd(0);
  SDRAM_CKE   <= '1';

  fast_ack <= acks(0);
  disk_ack <= acks(1);

  counters: process(clk)
  begin
    if rising_edge(clk) then
      -- Manage counters.
      if tick(tick'high) = '1' then
        tick <= (others => '0');
        offset <= offset + 1;
      else
        tick <= tick + 1;
      end if;
    end if;
  end process;

  state_machine: process(clk)
  begin
    if rising_edge(clk) then
```

```
cmd <= CMD_NOP;
acks <= (others => '0');
-- Sample read data. Only use together with the
-- distributed ack.
rdata <= SDRAM_DQ;
SDRAM_DQ <= (others => 'Z');

if reset = '1' then
  ready <= '0';
  init_counter <= (others => '0');
  -- NOP
  SDRAM_A <= (others => '0');
  SDRAM_A(AP) <= '1';
  SDRAM_BA <= (others => '0');
  SDRAM_UDQM <= '1';
  SDRAM_LDQM <= '1';
  state <= INIT_NOP;
else
  init_counter <= init_counter + 1;

  case state is
  when INIT_NOP =>
    if init_counter(14 downto 12) = "111" then
      -- Anything greater than 200 us * 133.333 MHz
      cmd <= CMD_PRE;
      state <= INIT_PRE;
    end if;

  when INIT_PRE =>
    SDRAM_A(AP) <= '0';
    if init_counter(1) = '1' then
      -- "111000000000010"
      -- Pre lasts for 3 cycles.
      state <= INIT_AR;
    end if;

  when INIT_AR =>
    if init_counter(3 downto 0) = "0011" then
      -- "111000000000011"
      -- "111000000010011"
      -- "111000000100011"
      -- "111000000110011"
      -- "111000001000011"
      -- "111000001010011"
      -- "111000001100011"
      -- "111000001110011"
      cmd <= CMD_AR;
    end if;
    if init_counter(7) = '1' then
      cmd <= CMD_MRS;
      -- SDRAM_A is zero. Set bits according to MRS.
      -- CAS Latency 3.
      -- A(6) <= '0';
      SDRAM_A(5) <= '1';
      SDRAM_A(4) <= '1';
      -- Burst length 1.
      -- SDRAM_A(2) <= '0';
      -- SDRAM_A(1) <= '0';
      -- SDRAM_A(0) <= '0';
      state <= INIT_MRS;
    end if;
```

```
when INIT_MRS =>
  if init_counter(0) = '1' then
    state <= INIT_DONE;
  end if;

when INIT_DONE =>
  ready <= '1';
  if nwe = WRITE_UPPER then
    SDRAM_UDQM <= '0';
    SDRAM_LDQM <= '1';
  elsif nwe = WRITE_LOWER then
    SDRAM_UDQM <= '1';
    SDRAM_LDQM <= '0';
  else
    SDRAM_UDQM <= '0';
    SDRAM_LDQM <= '0';
  end if;

  -- Registered outputs
  case tick is
  when x"0" =>
    -- Transfer offset, tick, req => acks_pend.
    null;
  when x"1" =>
    -- ACT
    SDRAM_BA  <= addr(22 downto 21);
    SDRAM_A   <= addr(20 downto  9);
    if refresh = '0' then
      cmd <= CMD_ACT;
    else
      cmd <= CMD_AR;
    end if;
  when x"2" =>
    -- NOP
  when x"3" =>
    -- NOP
  when x"4" =>
    --  Auto Precharge
    SDRAM_A(AP) <= '1';
    SDRAM_A(7 downto 0) <= addr(8 downto 1);
    if refresh = '1' then
      -- NOP
    elsif nwe = READ_WORD then
      cmd <= CMD_READ;
      -- READ + AP
    else
      SDRAM_DQ <= wdata;
      cmd <= CMD_WRITE;
    end if;
  when x"5" =>
    -- NOP
  when x"6" =>
    -- NOP
  when x"7" =>
    -- NOP
  when others =>
    -- NOP
    acks <= acks_pend;
    nwe <= READ_WORD;
  end case;
end case;
```

```
        case offset is
        when "00" | "10" =>
          refresh <= '0';
          addr <= fast.addr;
          wdata <= fast.wdata;
          if tick = 0 then
            acks_pend(0) <= fast.req;
            acks_pend(1) <= '0';
          end if;
          if tick = 1 and fast.req = '1' then
            nwe <= fast.nwe;
          end if;

        when "01" =>
          refresh <= '0';
          addr <= disk.addr;
          wdata <= disk.wdata;
          if tick = 0 then
            acks_pend(0) <= '0';
            acks_pend(1) <= disk.req;
          end if;
          if tick = 1 and disk.req = '1' then
            nwe <= disk.nwe;
          end if;

        when others =>
          refresh <= '1';
          addr <= disk.addr;
          wdata <= disk.wdata;
          acks_pend <= "00";
        end case;
      end if;
    end if;
  end process;
end;
```

B.5 Fast RAM Controller

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--

-- Amiga Fast RAM
library ieee;
use ieee.std_logic_1164.all;
use work.greta_pkg.all;

entity fast is
  port(
    clk         : in    std_logic;
    reset       : in    std_logic;

    req         : in    std_logic;
    nwe         : in    bus_nwe;
    dev_select  : out   std_logic;
    addr        : in    bus_addr;
    wdata       : in    bus_data;
    rdata       : out   bus_data;
    config_in   : in    std_logic;
    config_out  : out   std_logic;

    ram         : out   ram_bus;
    ram_ack     : in    std_logic;
    ram_rdata   : in    bus_data
  );
end;

architecture rtl of fast is
  type fast_state is (
    UNCONFIGURED, SHUT_UP_FOREVER, CONFIGURED, RAM_ACCESS
  );

  signal state : fast_state := UNCONFIGURED;
  signal addr_autoconfig0 : boolean := false;
  signal addr_fast_ram : boolean := false;
  signal dev_select_reg : std_logic := '0';
  signal rom_rdata : std_logic_vector(15 downto 12);
  signal rdata_reg : bus_data := (others => '0');
  signal addr_low7 : std_logic_vector(7 downto 0);

begin
```

```
process(clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      state <= UNCONFIGURED;
      ram.req <= '0';
    else
      case state is
      when UNCONFIGURED =>
        if (
          req = '1' and nwe(UPPER) = '0' and
          addr_autoconfig0 and config_in = '1'
        ) then
          if is_autoconfig_reg(ec_ShutUp, addr) then
            state <= SHUT_UP_FOREVER;
          elsif is_autoconfig_reg(ec_BaseAddress, addr) then
            state <= CONFIGURED;
          end if;
        end if;

      when SHUT_UP_FOREVER =>
        null;

      when CONFIGURED =>
        if req = '1' and addr_fast_ram then
          ram.req <= '1';
          state <= RAM_ACCESS;
        end if;

      when RAM_ACCESS =>
        if ram_ack = '1' then
          ram.req <= '0';
          state <= CONFIGURED;
        end if;
      end case;
    end if;
  end if;
end process;

-- Address comparator for the unconfigured device.
addr_autoconfig0 <=  is_autoconfig(addr) and
                       get_autoconfig_slot(addr) = "000";
-- Address comparator for the configured device.
addr_fast_ram   <=  is_fast_ram(addr);

-- Register for rdata back to bus_interface.
process(clk)
begin
  if rising_edge(clk) then
    -- Create register stage for dev_select and rdata. This
    -- improves performance as the signals are heavy on logic
    -- and are used as output.
    if (addr_autoconfig0 and state = UNCONFIGURED and
     config_in = '1')
    or  (addr_fast_ram and state /= UNCONFIGURED and
     state /= SHUT_UP_FOREVER)

    then
      -- Selected for AUTOCONFIG or Fast RAM
      dev_select_reg <= '1';
      if state = UNCONFIGURED then
        -- Output AUTOCONFIG ROM data
```

```
          rdata_reg(15 downto 12) <= rom_rdata;
        elsif ram_ack = '1' then
          -- Reload rdata when SDRAM has given us a read word.
          rdata_reg <= ram_rdata;
        end if;
      else
        -- We must give zeroes out when not selected.
        dev_select_reg <= '0';
        rdata_reg <= x"0000";
      end if;
    end if;
  end process;

  -- AUTOCONFIG ROM
  addr_low7 <= '0' & addr(6 downto 1) & '0';
  with addr_low7 select
    rom_rdata <=
      -- ERT_MEMLIST links memory into memory free list.
      (ERT_ZORROII or ERT_MEMLIST)              when x"00",
      (ERT_CHAINEDCONFIG or ERT_MEMSIZE_8MB)  when x"02",
      not (FAST_PRODUCT_NUMBER)                 when x"06",
      not (HACKER_MANUFACTURER(15 downto 12)) when x"10",
      not (HACKER_MANUFACTURER(11 downto  8)) when x"12",
      not (HACKER_MANUFACTURER( 7 downto  4)) when x"14",
      not (HACKER_MANUFACTURER( 3 downto  0)) when x"16",
      "0000"                                    when x"40",
      "0000"                                    when x"42",
      "1111" when others;

  dev_select  <= dev_select_reg;
  rdata       <= rdata_reg;
  config_out  <=  '0' when state = UNCONFIGURED else
                  '1';
  ram.nwe <= nwe;
  ram.addr <= addr;
  ram.wdata <= wdata;
end;
```

B.6 Disk Controller

```
--
-- Copyright (C) 2013 Martin Åberg
--
--  This program is free software: you can redistribute it
--  and/or modify it under the terms of the GNU General Public
--  License as published by the Free Software Foundation,
--  either version 3 of the License, or (at your option)
--  any later version.
--
--  This program is distributed in the hope that it will
--  be useful, but WITHOUT ANY WARRANTY; without even the
--  implied warranty of MERCHANTABILITY or FITNESS FOR A
--  PARTICULAR PURPOSE.  See the GNU General Public License
--  for more details.
--
--  You should have received a copy of the GNU General
--  Public License along with this program.  If not, see
--  <http://www.gnu.org/licenses/>.
--
library ieee;
use ieee.std_logic_1164.all;
use work.greta_pkg.all;

entity disk is
  port(
    clk         : in    std_logic;
    reset       : in    std_logic;

    req         : in    std_logic;
    nwe         : in    bus_nwe;
    dev_select  : out   std_logic;
    addr        : in    bus_addr;
    wdata       : in    bus_data;
    rdata       : out   bus_data;
    config_in   : in    std_logic;
    config_out  : out   std_logic;

    ram         : out   ram_bus;
    ram_ack     : in    std_logic;
    ram_rdata   : in    bus_data;

    --- SECURE DIGITAL (SPI)
    SPI_CLK     : out   std_logic;
    SPI_nCS     : out   std_logic;
    SPI_DO      : out   std_logic;
    SPI_DI      : in    std_logic
  );
end;

architecture rtl of disk is
  type disk_state is (
    UNCONFIGURED, SHUT_UP_FOREVER, CONFIGURED
  );

  signal state : disk_state := UNCONFIGURED;
  signal addr_autoconfig_disk : boolean := false;
  signal dev_select_reg : std_logic := '0';
  signal rom_rdata : std_logic_vector(15 downto 12);
  signal disk_rdata : bus_data := x"c0de";
  signal rdata_reg : bus_data := (others => '0');
  signal addr_low7 : std_logic_vector(7 downto 0);
  signal slot : std_logic_vector(2 downto 0) := "000";

begin

  SPI_CLK <= 'Z';
  SPI_nCS <= 'Z';
  SPI_DO  <= 'Z';

  process(clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        state <= UNCONFIGURED;
        ram.req <= '0';
        slot <= "000";
      else
        case state is
        when UNCONFIGURED =>
          if (
            req = '1' and nwe(UPPER) = '0' and
            addr_autoconfig_disk and config_in = '1'
          ) then
            if is_autoconfig_reg(ec_ShutUp, addr) then
              state <= SHUT_UP_FOREVER;
            elsif is_autoconfig_reg(ec_BaseAddress, addr) then
              state <= CONFIGURED;
              slot <= wdata(10 downto 8);
            end if;
          end if;

        when SHUT_UP_FOREVER =>
          null;

        when CONFIGURED =>
          null;

        end case;
      end if;
    end if;
  end process;

  -- Address comparator for the unconfigured or configured
  -- device.
  addr_autoconfig_disk <= is_autoconfig(addr) and
                          get_autoconfig_slot(addr) = slot;

  -- Register for rdata back to bus_interface.
  process(clk)
  begin
    if rising_edge(clk) then
      -- Create register stage for dev_select and rdata. This
      -- improves performance as the signals are heavy on logic
      -- and are used as output.
      if (addr_autoconfig_disk and config_in = '1' and
       state /= SHUT_UP_FOREVER) then
        -- We are selected.
        dev_select_reg <= '1';
        if state = UNCONFIGURED then
          -- Output AUTOCONFIG ROM data
          rdata_reg(15 downto 12) <= rom_rdata;
        else
```

```
          -- Reload rdata with disk register data.
          rdata_reg <= disk_rdata;
        end if;
      else
        -- We must give zeroes out when not selected.
        dev_select_reg <= '0';
        rdata_reg <= x"0000";
      end if;
    end if;
  end process;

  -- AUTOCONFIG ROM
  addr_low7 <= '0' & addr(6 downto 1) & '0';
  with addr_low7 select
    rom_rdata <=
      -- ERT_MEMLIST links memory into memory free list.
      (ERT_ZORROII)                          when x"00",
      (ERT_CHAINEDCONFIG or ERT_MEMSIZE_64K)  when x"02",
      not (DISK_PRODUCT_NUMBER)              when x"06",
      not (HACKER_MANUFACTURER(15 downto 12)) when x"10",
      not (HACKER_MANUFACTURER(11 downto  8)) when x"12",
      not (HACKER_MANUFACTURER( 7 downto  4)) when x"14",
      not (HACKER_MANUFACTURER( 3 downto  0)) when x"16",
      "0000"                                 when x"40",
      "0000"                                 when x"42",
      "1111" when others;

  dev_select  <= dev_select_reg;
  rdata       <= rdata_reg;
  config_out  <=  '0' when state = UNCONFIGURED else
                  '1';
  ram.nwe <= nwe;
  ram.addr <= addr;
  ram.wdata <= wdata;
end;
```

Appendix C

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders,

ADDENDUM: How to use this License for your documents

Bibliography

[1] Commodore Business Machines, *Commodore A500/A2000 Techincal Reference Manual.* 1986, 1987.

[2] Richard Stallman, Roland Pesch, Stan Shebbs, et al, *Debugging with GDB: The GNU Source-Level Debugger.* Free Software Foundation, 10th Edition, 2014. `http://www.gnu.org/software/gdb/documentation/`

[3] Motorola, Inc, *M68000 User's Manual.* Prentice-Hall, Inc, 8th Edition, 1990.