Technische Universiät Berlin

Fakultät IV: Elektrotechnik und Informatik

System-on-Chip + ARM Lab

Final Report: Ethernet controlled RGB LED-Cube

Tobias Minn

Studiengang: Technische Informatik

Arne Salzwedel

Studiengang: Technische Informatik

Julian Gog

Studiengang: Technische Informatik

Max Thonagel

Studiengang: Technische Informatik

July 8, 2018

1 Requirements Specification

The aim for this project was to build an 6x6x6 RGB LED-Cube, which can be controlled via an ethernet interface. Also we developed a program to generate the ethernet frames and tested everything with some animated patterns.

2 Functional Specification

In this section we list all required and optional functions the LED-Cube should provide.

- Hardware: Building an 6x6x6 RGB LED-Cube with corresponding PCB to connect to the provided FPGA Eval-Board
- Software: Implementation of computer program to communicate with the LED-Cube.
- HDL:
  - Ethernet MAC/Core: implement an Ethernet MAC to interface with the Ethernet PHY which is located on the eval board. The Ethernet MAC is responsible for decoding the ethernet frame sent by the computer program and to transfer the received data to the hardware module in the FPGA which is responsible for controlling the LED-Cube.
  - LED-Cube Controller: implement hardware which is responsible for receiving the decoded data from the ethernet MAC and controlling/interfacing the LED-Cube hardware. This includes the switching of the levels of the cube, since the LED-Cube is level-multiplexed.
- Functions:
  - switch every LED on and off separately
  - control the color of every LED (red, green, blue)
  - LED frame sending rate: 30 Hz
  - Multiplexing frequency of the LED-Cube levels: 1kHz
- optional Functions:
  - Software PWM which is controlled by the computer program, by sending ethernet frames at high rates and modulating the on-/offtimes
  - Ability of the computer program to generate and send some predefined (fixed) animations to the LED-Cube.

3 High-Level Design

Partitioning of the project:

- Physical Hardware: Building an 6x6x6 RGB LED-Cube and develop a PCB with LED drivers to mount the cube.
- FPGA Hardware: Implementation of the Ethernet MAC and the LED-Cube controller, as well as programming of the ARM core in the FPGA
- Computer program: Implementation of the computer program needed to generate the ethernet frames. Optional: implementation of some predefined (fixed) animations.

3.1 Hardware Top Level Blockdiagram

Figure 1 shows the whole System-on-Chip where the Ethernet Core/MAC and the Student Module were implemented.

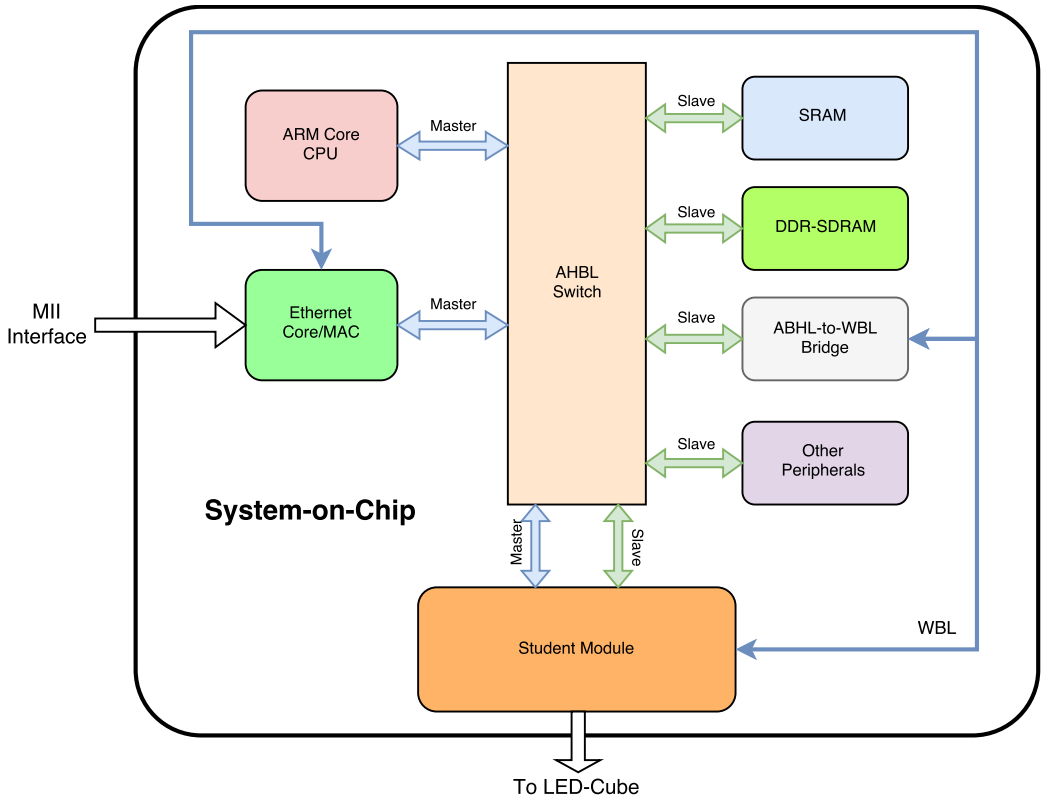


Figure 1: Hardware Top Level Blockdiagram of the whole System-on-Chip

3.2 Software Top Level Blockdiagram

The software template of the Armmini OS was used to implement the software for the SoC ARM processor. Figure 2 shows the overall software flow of the system-on-chip. The software flow starts at the main function an initializes the system. This includes setting up the basic interrupt system and the serial monitor, as well as some timer modules. After the system initialization, the user code is executed. The user functions will initialize the ethernet module and the led cube controller. The first action is to register our FIQ interrupt handler for the level switching of the LED-Cube controller in the base interrupt system of the system-on-chip. Then the student lights module and the LED-Cube controller are configured with default values. The ethernet AHBL bus master is configured with a memory address where to write the ethernet payload. In the last step the ethernet module gets enabled.

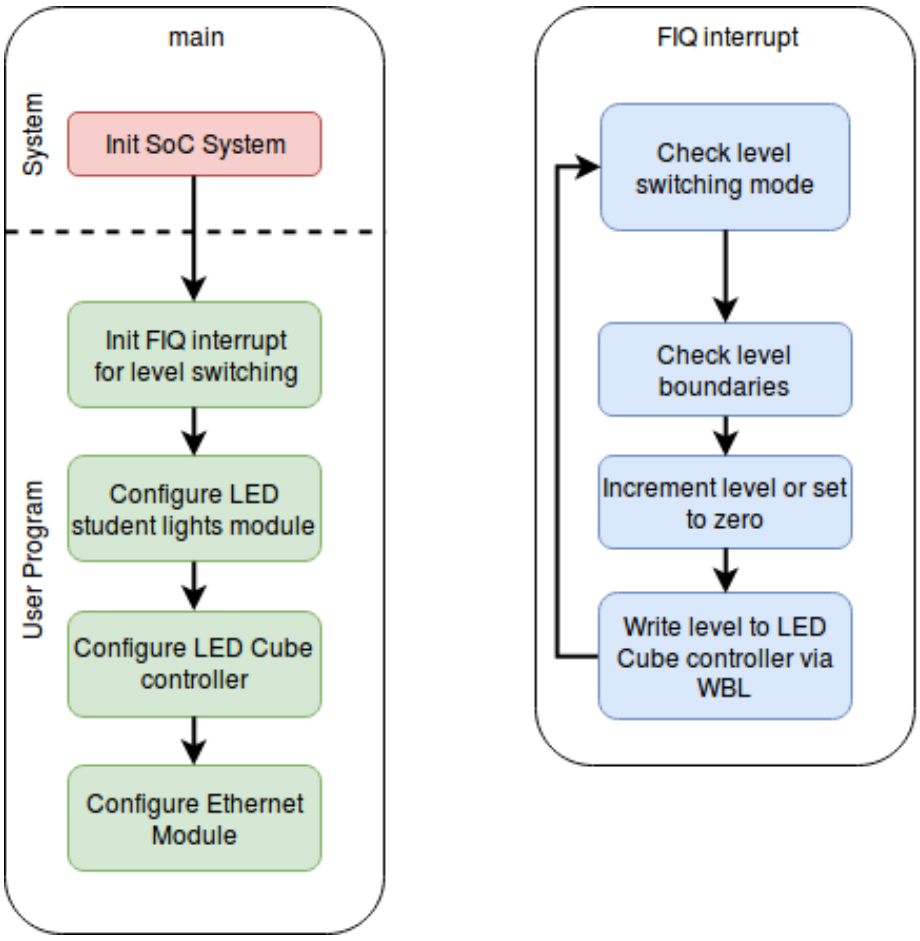


Figure 2: Overview of the software flow of the System-on-Chip

3.3 Interface Specification

This section gives an overview of the inputs and outputs of the System-on-Chip. Table 1 contains all inputs and outputs of the system with the corresponding bit width and a short description.

Signal name	Direction	Description
Ethernet Core/MAC		
RX_CLK	INPUT	Ethernet PHY Receive clock
RXD (3 downto 0)	INPUT	Ethernet PHY nibble output
RXDV	INPUT	Data valid signal of Ethernet PHY
RXERR	INPUT	Error signal in case of frame error of Ethernet PHY
Student Module		
RGB_OUT (107 downto 0)	OUTPUT	LED Output for the current level (36 RGB leds)
LEVEL_OUT (5 downto 0)	OUTPUT	Activates the currently active led level of the cube (one hot encoded)

Table 1: Overview of all input and output signals of the System-on-Chip

4 Module description

4.1 Ethernet Core/MAC

This hardware module in the FPGA is responsible for receiving, decoding and saving the ethernet frame in memory, which is sent by the computer program. The module is divided in submodules

which will be described in this section. A high-level overview is shown in figure 3. A comprehensive blockdiagram with all connections can be found in the appendix. The submodules are:

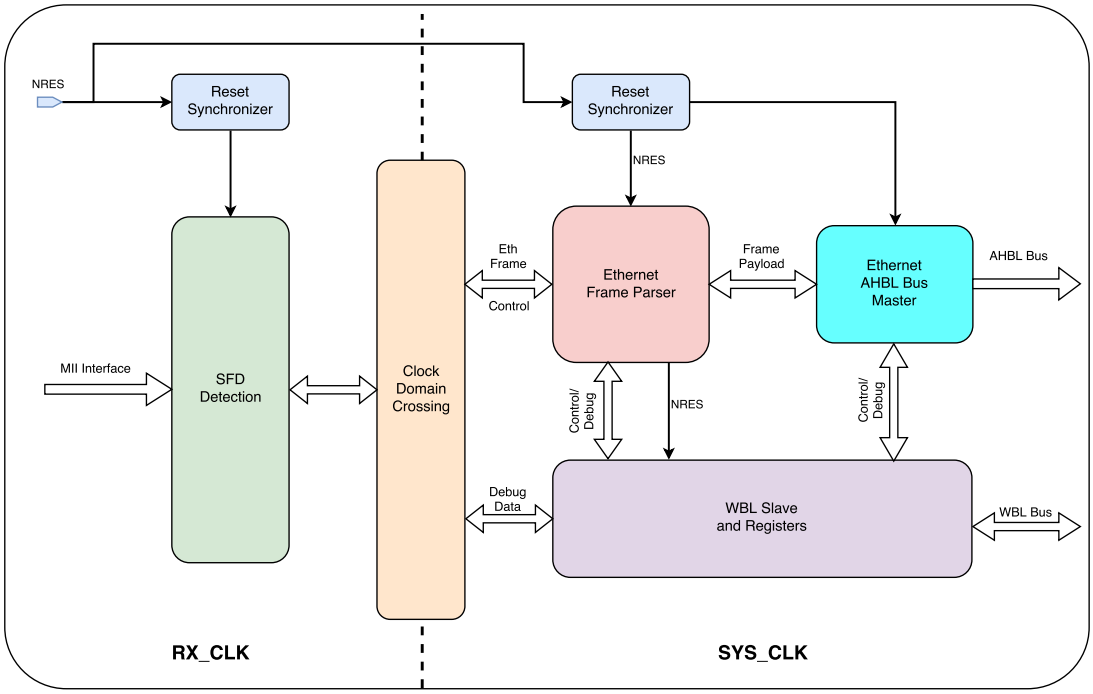


Figure 3: Simplified Ethernet Core/MAC Blockdiagram with the important submodules

- SDF Detection: This module searches for the *Start of Frame Delimiter* and gathers four sequential nibbles and concatinates them to a 16-bit value. This value will then be handed over to the *Ethernet Parser* module via CDC (*Clock Domain Crossing*)
- Ethernet Parser: The *Ethernet Parser* module is responsible to decode the received ethernet frame, check if it is valid, as well as hand it over to the *Bus Master* in the ethernet core.
- Ethernet Bus Master: The *Ethernet Bus Master* is responsible to save the received ethernet frame payload to a programmable memory address.
- WBL Slave: The *WBL Slave* is responsible for the control communication with the system. It contains the registers which are used to configure the Ethernet Core/MAC

4.1.1 SDF Detection

Functionality

- searches for the *Start of Frame Delimiter* after the preamble of the ethernet frame
- gathers four sequential nibbles and concatinates them to a 16-bit value
- hands over the 16-bit value to the *Ethernet Parser* module via CDC (*Clock Domain Crossing*)

Register Interface see Appendix for register definition

Finite State Machines The state machine implemented in this module can be seen on page 18.

4.1.2 Ethernet Parser

Functionality

- receives frame from the SDF detection module (16 bit at a time)
- decodes all relevant information of the frame, like
  - Source MAC address
  - Destination MAC address
  - Frame type
  - Frame length
  - Payload
  - CRC checksum
- discards frame if it is invalid or not meant to be for the FPGA
- checks if destination MAC address is MAC address configured in the MAC destination register
- saves MAC address of sender
- hands over frame payload to the ethernet bus master

Register Interface see Appendix for register definitione

Finite State Machines The main state machine implemented in this module can be seen on page 19. Additional state machines for debug and syncing can be seen on page 20 and page 21 respectively.

4.1.3 Ethernet Bus Master

Functionality

- receives payload from the *Ethernet Parser* module
- writes received payload to specified address in memory

Register Interface see Appendix for register definition

Finite State Machines The main state machine implemented in this module can be seen on page 22.

4.1.4 WBL Slave module

Functionality

- is responsible for the control communication with the system
- contains register to configure the Ethernet Core/MAC module

Register Interface see Appendix for register definition

Finite State Machines The main state machine implemented in this module can be seen on page 23.

4.2 Student module

This hardware module in the FPGA contains the exercises implemented during the lab, as well as the *LED-Cube controller* module, which is responsible for driving the pins of the externally connected LED-Cube. The *LED-Cube controller* also multiplexes the levels of the LED-Cube. Figure 4 shows a simplified block diagram of the student module for better understanding. A comprehensive blockdiagram of the student module with all connections can be found in the appendix.

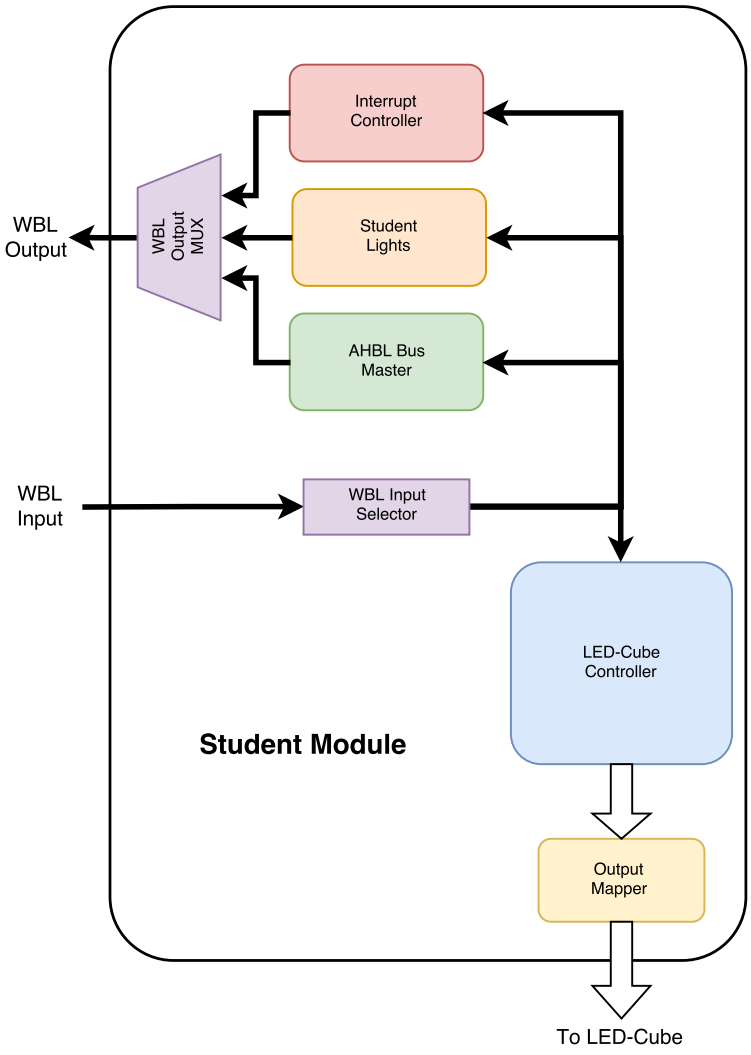


Figure 4: Simplified block diagram of the student module with all important submodules

4.2.1 LED-Cube controller

Functionality

- multiplexes the levels of the LED-Cube
- contains registers to configure the LED-Cube controller
- WBL state machine enables communication with the remaining system
- configuration of external LED-Cube hardware

Register Interface see Appendix for register definition

Finite State Machines The main state machine implemented in this module can be seen in figure **??**.

5 PC Software

A computer program for Linux controls the LED-Cube by generating and sending ethernet frames. The software is written in C and provides a user interface based on GTK+ (see figure 5).

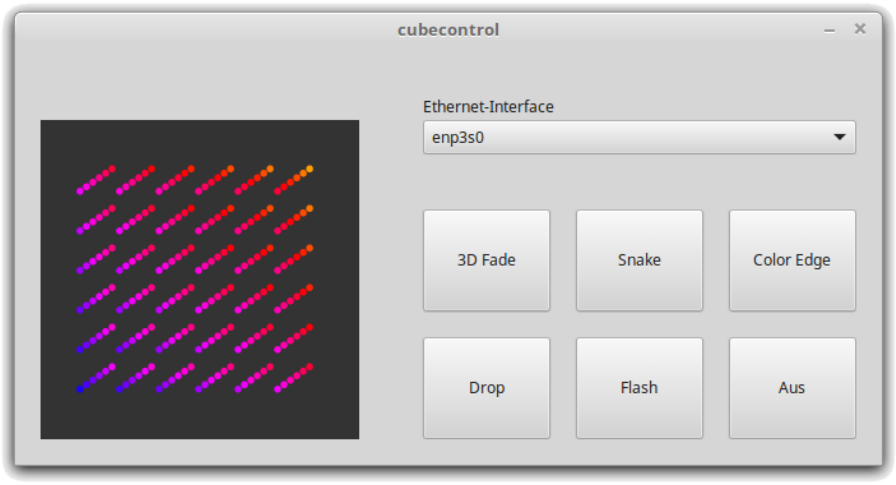


Figure 5: PC Software User Interface

The GUI allows the user to select the ethernet adapter, on which the cube is connected, and control the animation effect. He can choose between six preprogrammed led patterns. Each pattern is animated differently and uses randomly selected colors or movement directions to create a wider variety of effects.
The program uses four different threads, which run widely independent. This allows to send ethernet frames at maximum rate regardless of the animation processing speed.
Double buffers are used for the data transfer between the threads. A mutex for each buffer ensures that during flipping no read access can occur. This prevents tearing effects on the cube.

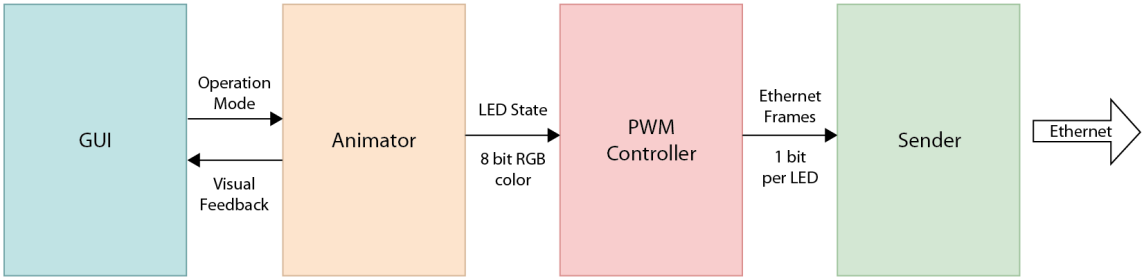


Figure 6: PC Software Thread Structure

Function of each Thread:

- GUI: user control and visual feedback
- Animator: runs the selected animation pattern on a virtual cube with 8-bit RGB color
- PWM Controller: Converts the 8-bit RGB color values into the ethernet frame with a single bit per LED. Color mixing and dimming is realised by variation of on- an off-times
- Sender: sends the ethernet frames continuously at the highest possible rate

6 LED-Cube Hardware

To be able to test the LED-Cube safely, we developed a custom PCB for the evaluation board we used (Avnet LX60). The whole schematic and the layout files for the PCB can be found in appendix C. The PCB connects to the eval board via two 140 pin connector. There was a need for two connectors, since the System-on-Chip controls a whole LED-Cube level (36 RGB LEDs = 108 pins) and the level itself (6 pins needed) at once. Since the layout of the connector specifies a ground and a power connection on every other pin, one connector did not suffice. Figure 7 shows the basic connection scheme of the LEDs.

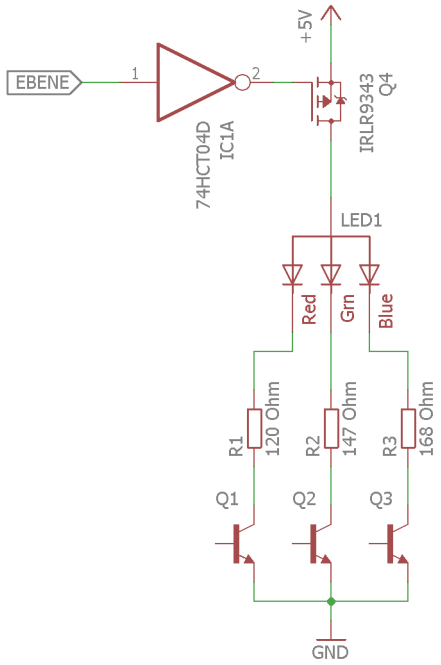


Figure 7: Basic LED connection scheme

An inverter drives a p-channel MOSFET, which is responsible to turn on the corresponding level. Only one level of the LED-Cube is turned on at the same time. The circuit uses Darlington-Arrays to sink the current, that is limited by a resistor per led, to ground and turn the corresponding LED on.

7 Results

7.1 Measurements

7.1.1 Ethernet frame measurement

After successfully testing the cube we performed several measurements to verify the correction functionality of the LED-Cube. Therefore we used a logic analyzer to trace the incoming ethernet frames. Figure 8 shows the measurement of a transfer of several ethernet frames. We analyzed the sending rate of the frame and verified the a sending rate of 82.78 kHz.

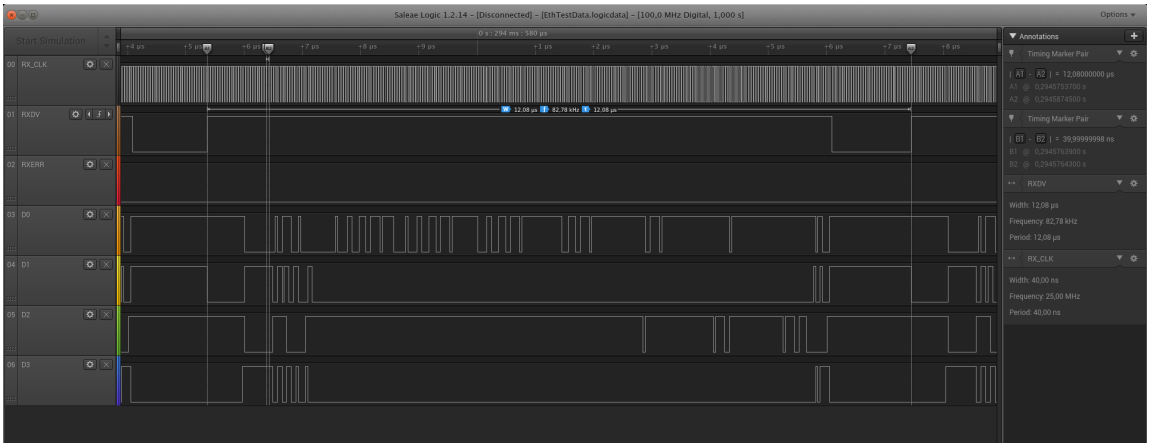


Figure 8: Measurement of a transfer of several ethernet frames

On frame contains the following data:

- Preamble : 7 Bytes
- Start of frame delimiter : 1 Byte
- MAC destination : 6 Bytes
- MAC source : 6 Bytes
- Frame length : 2 Bytes
- Payload of information for LEDs : 108 Bytes
- CRC checksum : 4 Bytes

This means one frame contains 134 Bytes in total. With a sending rate of 82.78 kHz we get a transfer rate of 10.57 MByte per second. This is almost the maximum transfer rate (12.5 MByte per second) for the Ethernet(100BASE-X) that was used.

7.1.2 Level Switching

The screenshot in figure 9 shows the level switching frequency recorded with an oscilloscope.

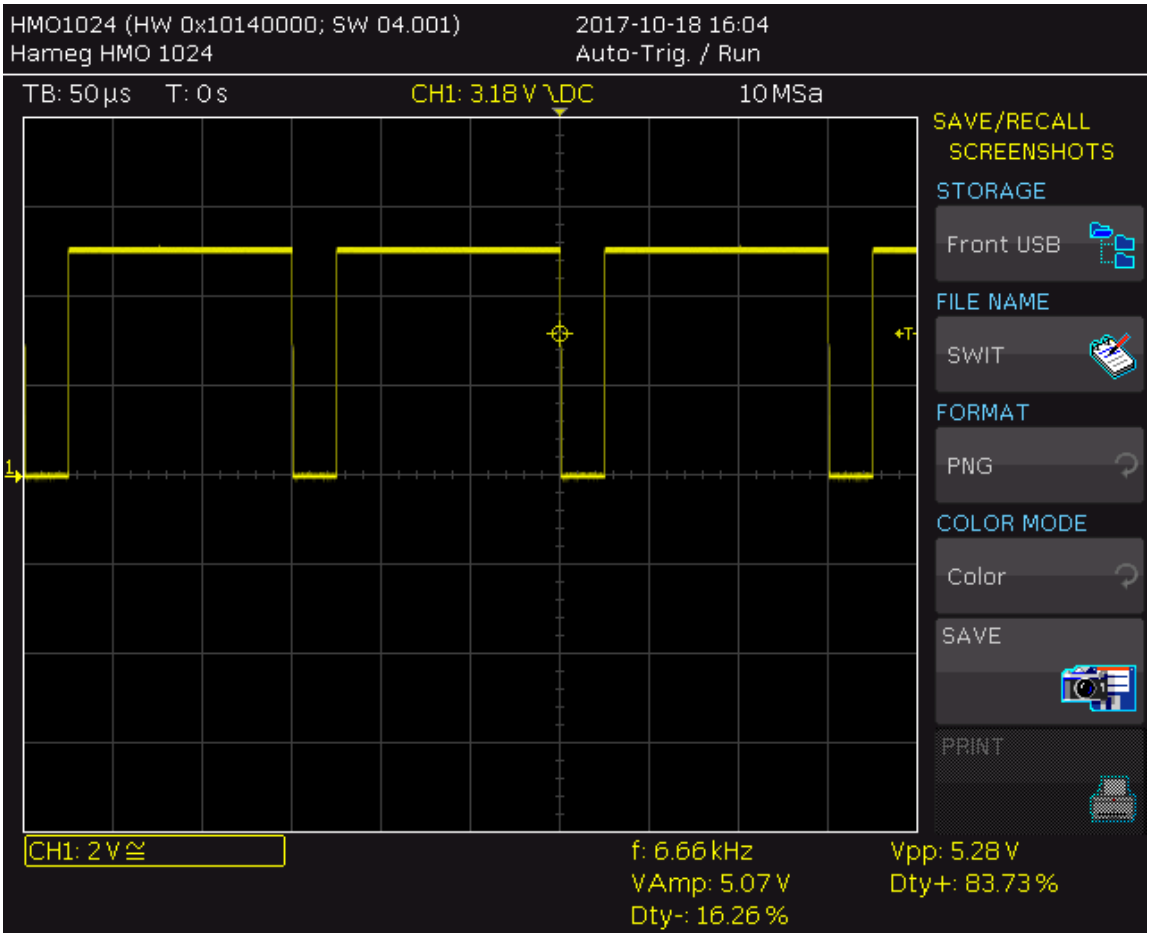


Figure 9: Measurement of the level switching with an oscilloscope

In figure 10 the level switching (blue) is measured beside an led (yellow) which is only activated in the same level.

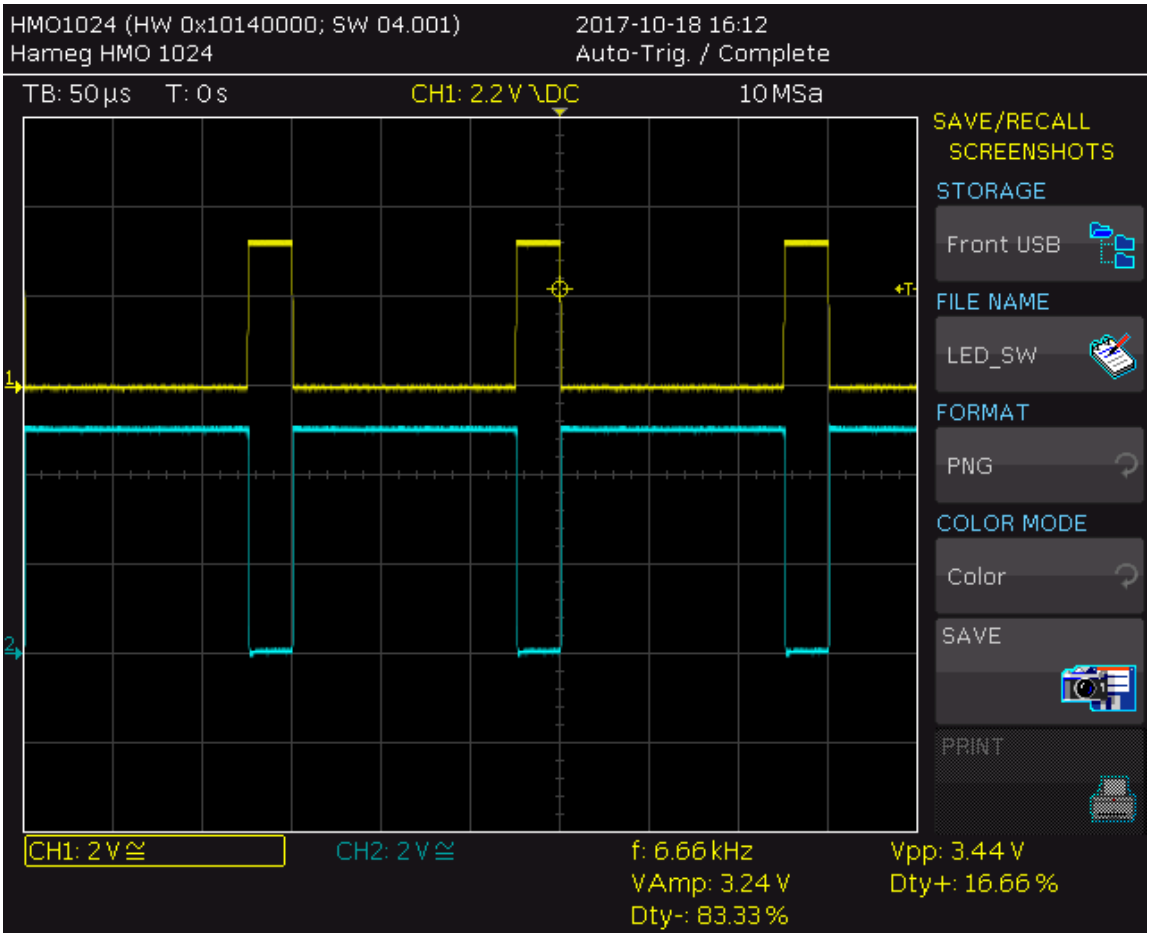


Figure 10: Measurement of the level switching (blue) and one led (yellow) only activated in this level with an oscilloscope

7.1.3 Software PWM

Figure 11 shows the resulting software PWM measured with an oscilloscope.

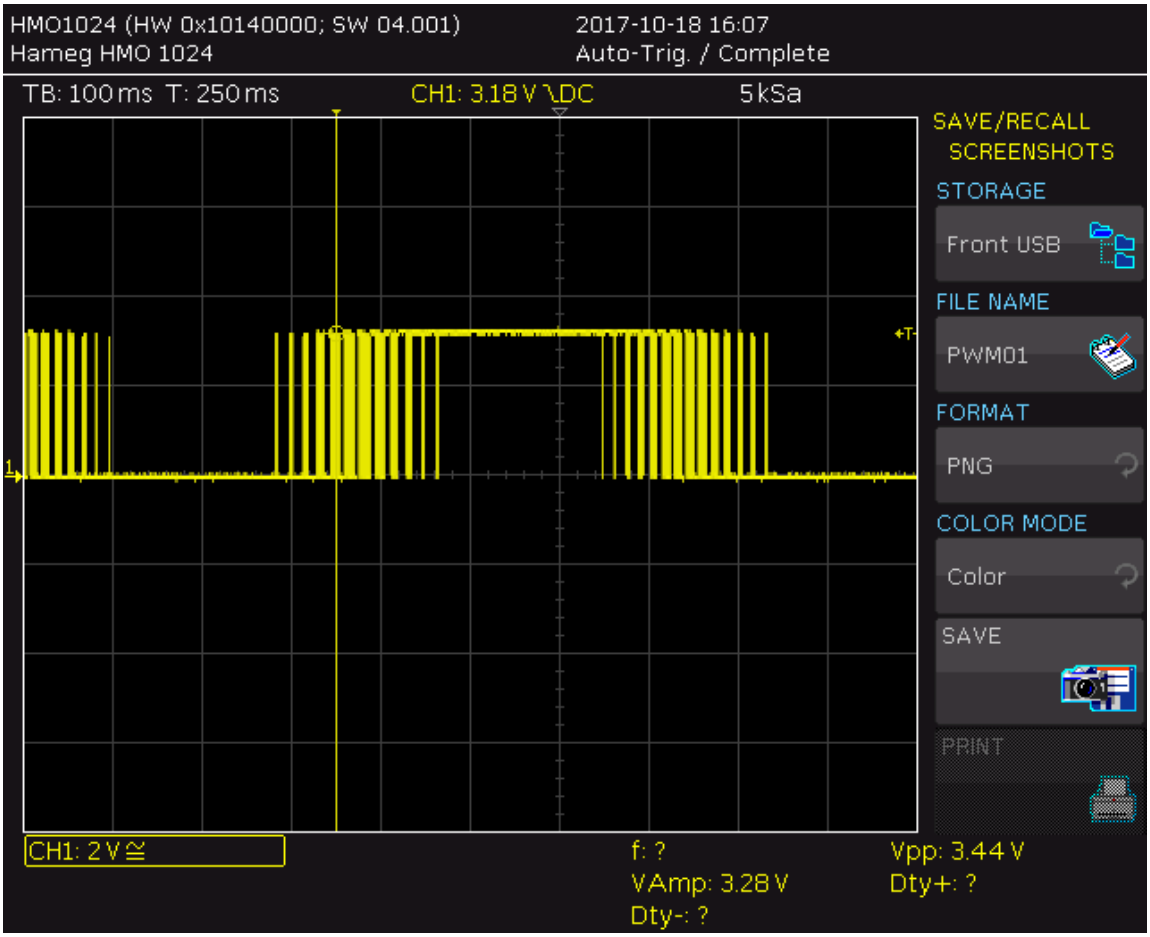


Figure 11: Measurement of the software PWM

7.2 Photos and Videos

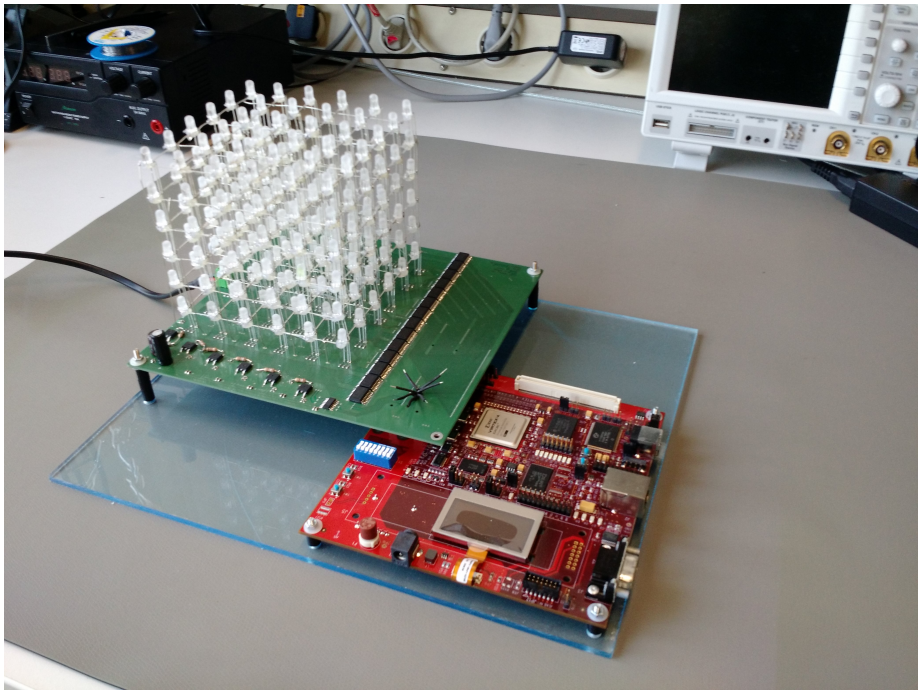
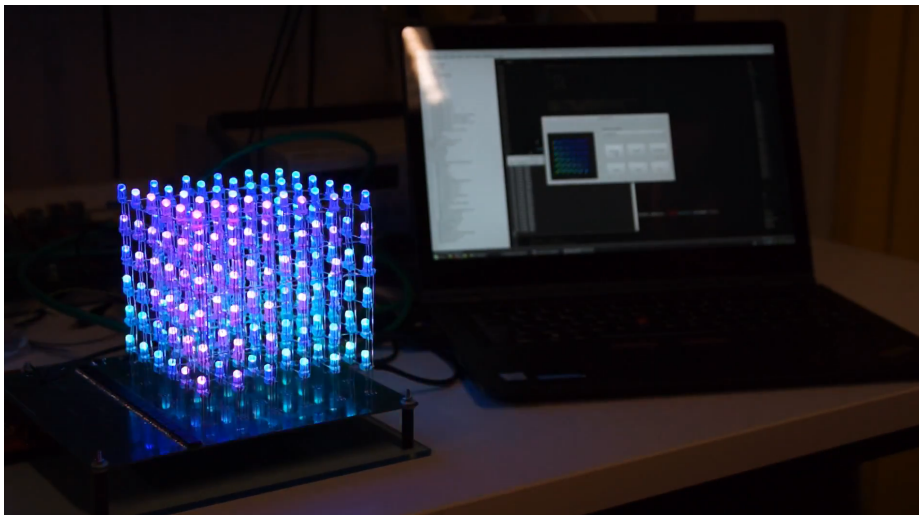
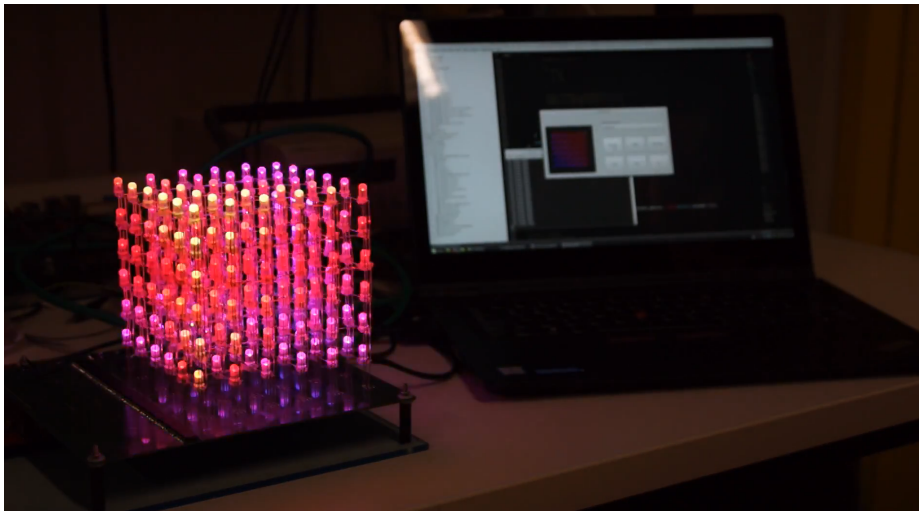
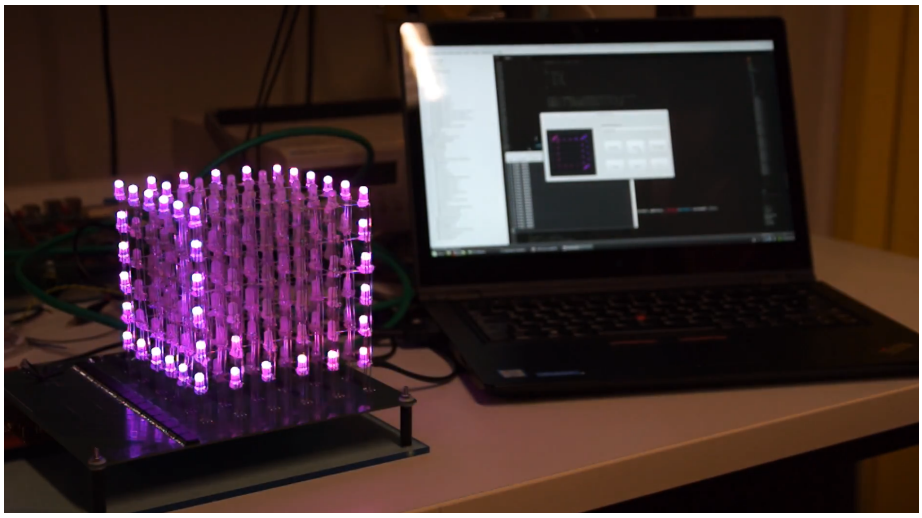


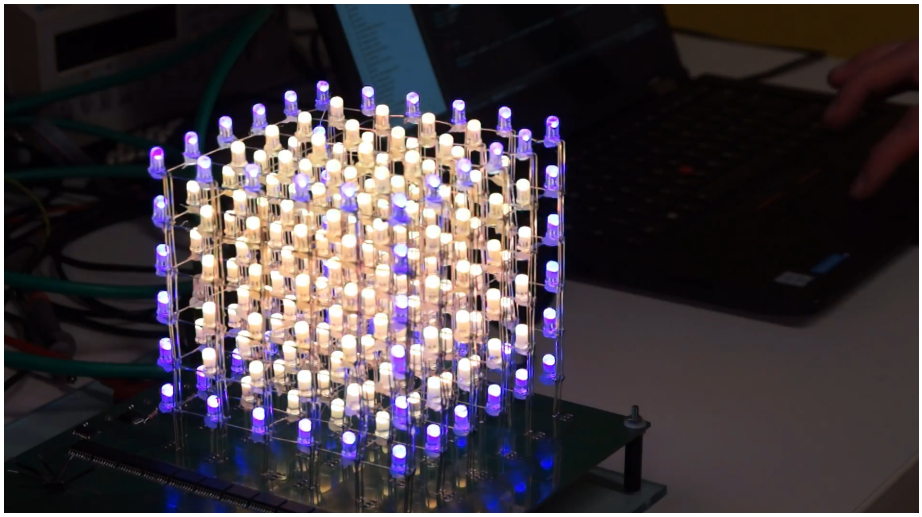
Figure 12: LED-Cube board attached to the FPGA board

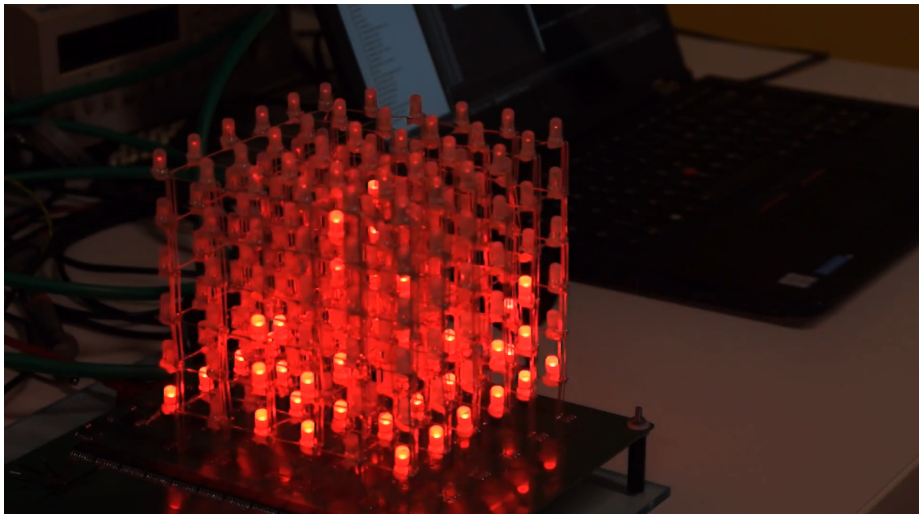
A demonstration video is available on Youtube: `https://www.youtube.com/watch?v=e2JtOskrisI`

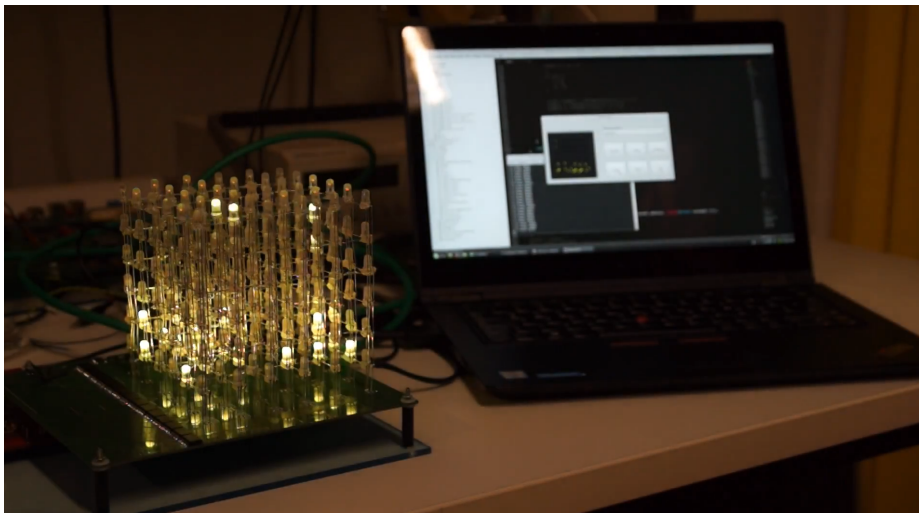
s 









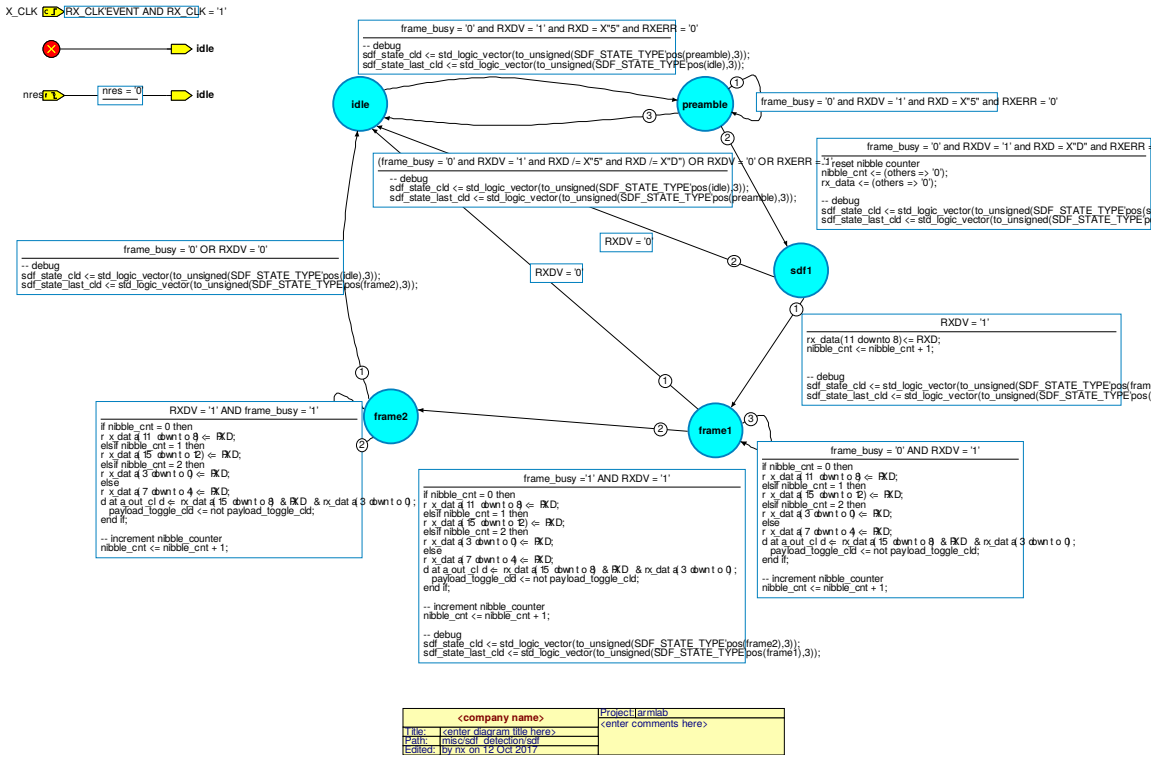


A State Machines

Global Actions **Concurrent Statements** **Architecture Declarations** **Signal Status** **State Register Statements** **Process Declarations**
Pre Actions:
Post Actions:

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

SIGNAL	MODE	DEFAULT	RESET	Clocked Process: ASSIGN IN
payload_toggle	OUT	'0'	'0'	Output Process:
data_out	OUT	(others => '0')	CLKD	<auto>
last_recv	LOCAL	last_recv	(others => '0')	CLKD sdf
sdf_state	OUT	sdf_state_cld	(others => '0')	CLKD <auto>
sdf_state_last	OUT	sdf_state_last_cld	(others => '0')	CLKD <auto>
nibble_cnt	LOCAL	nibble_cnt	(others => '0')	CLKD sdf
rx_data	LOCAL	rx_data	(others => '0')	CLKD sdf
abort	OUT	'0'	CLKD	<auto>



<company name>	Project: armlab
Title: <enter diagram title here>	<enter comments here>
Path: misc/sdf_detection/sdf	
Edited: by nx on 12 Oct 2017	

Global Actions **Concurrent Statements** **Architecture Declarations** **State Register Statements** **Process Declarations**

Pre Actions:

Post Actions:

Clocked Process:

Output Process:

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME ASSIGN IN
word_buffer	LOCAL	word_buffer	(others => '0')	CLKD RX
frame_size_out	OUT	frame_size_out_cld	(others => '0')	CLKD RX
frame_size	LOCAL	frame_size	(others => '0')	CLKD RX
frame_data_cnt	LOCAL	frame_data_cnt	(others => '0')	CLKD RX
frame_CRC	LOCAL	frame_CRC	(others => '0')	CLKD RX
word_buffer_out	OUT	word_buffer_out_cld	(others => '0')	CLKD RX
MAC_dest	LOCAL	MAC_dest	X"000000000000"	CLKD RX
counter	LOCAL	counter	(others => '0')	CLKD RX
MAC_src	LOCAL	MAC_src	X"000000000000"	CLKD RX
rx_data	LOCAL	rx_data	(others => '0')	CLKD sync
new_data	LOCAL	new_data_buf	'0'	CLKD sync
rx_data_toggle_old	LOCAL	rx_data_toggle_old	'0'	CLKD sync
word_change_toggle	OUT	word_change_toggle_cld	'0'	CLKD RX
frame_busy	OUT	frame_busy_cld	'0'	CLKD RX
new_data_buf	LOCAL	new_data_buf	'0'	CLKD sync
frame_mac_src	OUT	frame_mac_src_cld	(others => '0')	CLKD <auto>
frame_mac_dest	OUT	frame_mac_dest_cld	(others => '0')	CLKD <auto>
frame_first_word	OUT	frame_first_word_cld	(others => '0')	CLKD <auto>
parse_state_out	OUT	parse_state_out_cld	(others => '0')	CLKD <auto>

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

PROBLEM: Which clock for state machine?
- RXD is synchron to RXCLK, can we use it here?
- if not how can we synchronize RXCLK to our system clock?

IMPORTANT: Check if bitmask for reception are correct
- may be a problem because of LSB first

clk: clk'EVENT AND clk = '1'

nres = '0'

idle

idle

-- wait for new data
new_data = '0'

TIMEOUT

idle

wait 5
discard

abort = '1'
frame_busy_cld <= '0';

new_data = '1'
-- first nibble of MAC address, LSB first?
MAC_dest(15 downto 0) <= rx_data;
-- set MAC nibble counter to zero
counter <= to_unsigned(0,16);
-- signal frame processing
frame_busy_cld <= '1';

(counter = 2 AND MAC_dest /= MAC_addr)
frame_busy_cld <= '0';

-- check if MAC address is completely received
counter /= 2 AND new_data = '1'
-- increment MAC_counter
counter <= counter + 1;
-- shift MAC_dest up
MAC_dest(47 downto 16) <= MAC_dest(31 downto 0);
-- and add new byte at the bottom
MAC_dest(15 downto 0) <= rx_data;

check_MAC_dest

counter = 2 AND MAC_dest = MAC_addr AND new_data = '1'
-- reset MAC_counter
counter <= to_unsigned(0,16);
-- add first byte at bottom bits
MAC_src(15 downto 0) <= rx_data;

-- for debug
frame_mac_dest_cld <= MAC_dest;

new_data = '1' AND frame_size > to_unsigne
frame_busy_cld <= '0';

abort = '1'
frame_busy_cld <= '0';

counter /= 2 AND new_data = '1'
-- increment MAC_counter
counter <= counter + 1;
-- shift MAC_src up
MAC_src(47 downto 16) <= MAC_src(31 downto 0);
-- and add new byte at the bottom
MAC_src(15 downto 0) <= rx_data;

check_MAC_src

counter = 2 AND new_data = '1'
-- reset MAC_counter
counter <= to_unsigned(0,16);

-- receive ethertype
frame_size(15 downto 0) <= unsigned(rx_data);

-- for debug
frame_mac_src_cld <= MAC_src;

EtherTypeSize

abort = '1'
frame_busy_cld <= '0';

abort = '1'
frame_busy_cld <= '0';

new_data = '1' AND frame_size <= to_unsigned(108,16)
frame_data_cnt <= (frame_size/2) -1;
frame_size_out_cld <= frame_size;

counter <= to_unsigned(0,16);

-- Set most significant 16bit
word_buffer(15 downto 0) <= rx_data;

frame_data_cnt /= 0 and new_data = '1'
word_buffer(31 downto 16) <= word_buffer(15 downto 0);
word_buffer(15 downto 0) <= rx_data;
frame_data_cnt <= frame_data_cnt -1;
counter <= counter + 1;

if counter = 1 then
word_buffer(31 downto 16) <= (others => '0');
counter <= to_unsigned(0,16);
word_buffer_out_cld <= word_buffer;
word_change_toggle_cld <= not word_change_toggle;

-- for debug
if (frame_size-3) = frame_data_cnt then
frame_first_word_cld <= word_buffer;
end if;
end if;

Frame

frame_data_cnt = 0 and new_data = '1'
counter <= to_unsigned(0,16);

if counter /= 0 then
word_buffer_out_cld <= word_buffer;
word_buffer <= (others => '0');
word_change_toggle_cld <= not word_change_toggle;
end if;

-- shift frame_size up
frame_CRC(31 downto 16) <= frame_CRC(15 downto 0);
-- and add new nibble at the top
frame_CRC(15 downto 0) <= unsigned(rx_data);

CRC

counter /= 1 AND new_data = '1'
-- increment MAC_counter
counter <= counter + 1;
-- shift frame_size up
frame_CRC(31 downto 16) <= frame_CRC(15 downto 0);
-- and add new nibble at the top
frame_CRC(15 downto 0) <= unsigned(rx_data);

abort = '1'
frame_busy_cld <= '0';

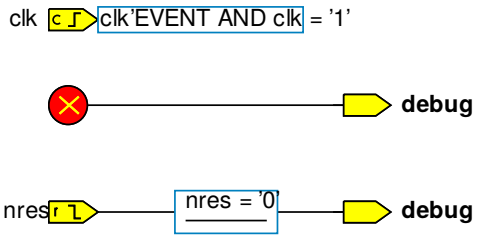
counter = 1
-- reset frame_busy_cld flag
frame_busy_cld <= '0';

<company name>	Project:\armlab
Title: <enter diagram title here>	<enter comments here>
Path: misc/ether_parse_package/parser	
Edited: by nx on 23 Aug 2017	

misc/ether_parse_package/parser [dbg]

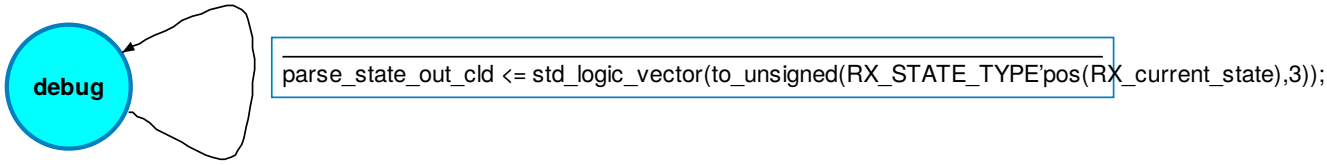
Global Actions **Concurrent Statements** **Architecture Declarations** **State Register Statements** **Process Declaratio**
Pre Actions: **Clocked Process:**
Post Actions: **Output Process:**

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;



Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME	ASSIGN IN
word_buffer	LOCAL	word_buffer	(others => '0')	CLKD	RX
frame_size_out	OUT	frame_size_out_cld	(others => '0')	CLKD	RX
frame_size	LOCAL	frame_size	(others => '0')	CLKD	RX
frame_data_cnt	LOCAL	frame_data_cnt	(others => '0')	CLKD	RX
frame_CRC	LOCAL	frame_CRC	(others => '0')	CLKD	RX
word_buffer_out	OUT	word_buffer_out_cld	(others => '0')	CLKD	RX
MAC_dest	LOCAL	MAC_dest	X"000000000000"	CLKD	RX
counter	LOCAL	counter	(others => '0')	CLKD	RX
MAC_src	LOCAL	MAC_src	X"000000000000"	CLKD	RX
rx_data	LOCAL	rx_data	(others => '0')	CLKD	sync
new_data	LOCAL	new_data_buf	'0'	CLKD	sync
rx_data_toggle_old	LOCAL	rx_data_toggle_old	'0'	CLKD	sync
word_change_toggle	OUT	word_change_toggle_cld	'0'	CLKD	RX
frame_busy	OUT	frame_busy_cld	'0'	CLKD	RX
new_data_buf	LOCAL	new_data_buf	'0'	CLKD	sync
frame_mac_src	OUT	frame_mac_src_cld	(others => '0')	CLKD	<auto>
frame_mac_dest	OUT	frame_mac_dest_cld	(others => '0')	CLKD	<auto>
frame_first_word	OUT	frame_first_word_cld	(others => '0')	CLKD	<auto>
parse_state_out	OUT	parse_state_out_cld	(others => '0')	CLKD	<auto>



<company name>	Project: armlab
Title: <enter diagram title here>	<enter comments here>
Path: misc/ether_parse_package/parser	
Edited: by nx on 23 Aug 2017	

misc/ether_parse_package/parser [sync]

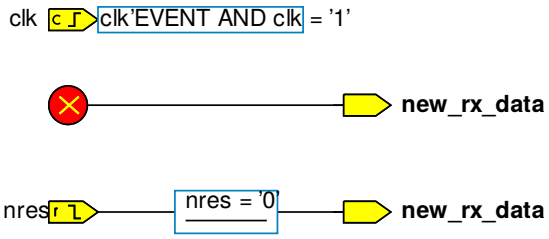
Global Actions **Concurrent Statements** **Architecture Declarations** **State Register Statements** **Process Declarations**

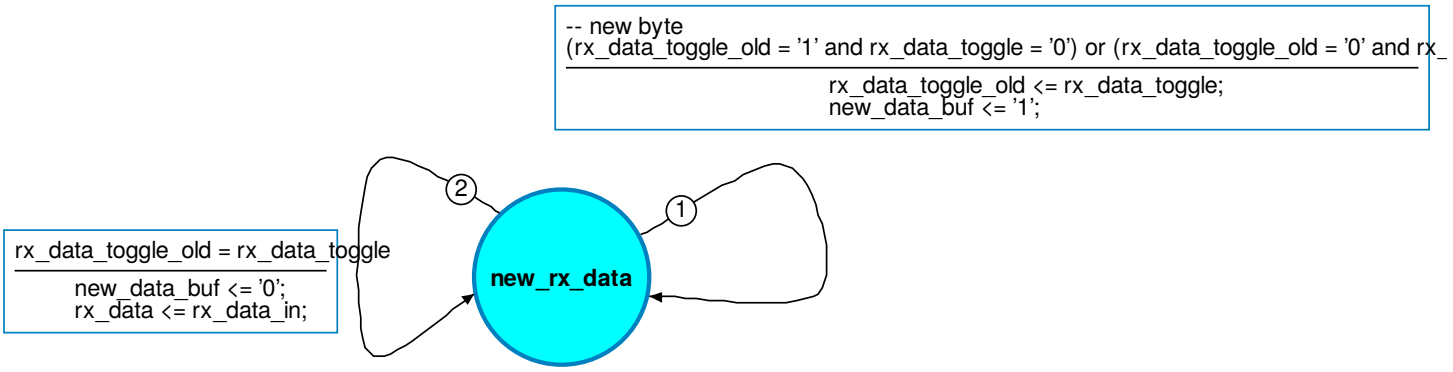
Pre Actions:
Post Actions:

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;

Signal Status

SIGNAL	MODE	DEFAULT	RESET	**Clocked Process:** SCHEME AS
word_buffer	LOCAL	word_buffer	(others => '0')	CLKD RX
frame_size_out	OUT	frame_size_out_cld	(others => '0')	CLKD
frame_size	LOCAL	frame_size	(others => '0')	CLKD RX
frame_data_cnt	LOCAL	frame_data_cnt	(others => '0')	CLKD
frame_CRC	LOCAL	frame_CRC	(others => '0')	CLKD R
word_buffer_out	OUT	word_buffer_out_cld	(others => '0')	CLKD
MAC_dest	LOCAL	MAC_dest	X"000000000000"	CLKD
counter	LOCAL	counter	(others => '0')	CLKD RX
MAC_src	LOCAL	MAC_src	X"000000000000"	CLKD
rx_data	LOCAL	rx_data	(others => '0')	CLKD sync
new_data	LOCAL	new_data_buf	'0'	CLKD sync
rx_data_toggle_old	LOCAL	rx_data_toggle_old	'0'	CLKD syr
word_change_toggle	OUT	word_change_toggle_cld	'0'	CLKD
frame_busy	OUT	frame_busy_cld	'0'	CLKD RX
new_data_buf	LOCAL	new_data_buf	'0'	CLKD sync
frame_mac_src	OUT	frame_mac_src_cld	(others => '0')	CLKD
frame_mac_dest	OUT	frame_mac_dest_cld	(others => '0')	CLKD
frame_first_word	OUT	frame_first_word_cld	(others => '0')	CLKD
parse_state_out	OUT	parse_state_out_cld	(others => '0')	CLKD





<company name>	Project: armlab
Title: <enter diagram title here>	<enter comments here>
Path: misc/ether_parse_package/parser	
Edited: by nx on 23 Aug 2017	

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
frame_addr_internal	LOCAL	frame_addr_internal	X"DEADBEEF"	CLKD
sp_HWDATA_buf	LOCAL		(others => '0')	CLKD
word_change_toggle_old	LOCAL	word_change_toggle_old	'0'	CLKD
counter	LOCAL	counter	(others => '0')	CLKD
sp_HADDR	OUT		(others => '0')	CLKD
sp_HSIZE	OUT		HSIZE_DWORD	CLKD
sp_HTRANS	OUT		HTRANS_IDLE	CLKD
sp_HWDATA	OUT	sp_HWDATA_buf	(others => '0')	CLKD
sp_HWRITE	OUT		'0'	CLKD
word_buf	LOCAL	word_buf	(others => '0')	CLKD
fiq	OUT	fiq_int	'0'	CLKD
fiq_int	LOCAL	fiq_int	'0'	CLKD

Process Declarations
Clocked Process:
Output Process:

Global Actions **Concurrent Statements** **Architecture Declarations** **State Register Statements**
Pre Actions:
Post Actions:

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY global;
USE global.global_defs.all;
USE global.bus_defs.all;
```

clk: clk'EVENT AND clk = '1'

```
LIBRARY ahbl_switch;
USE ahbl_switch.utils_ahbl_pkg.ALL;
```

ahbl_idle

nres: nres = '0' → ahbl_idle

word_change_toggle /= word_change_toggle_old and sp_HREADY = '1'

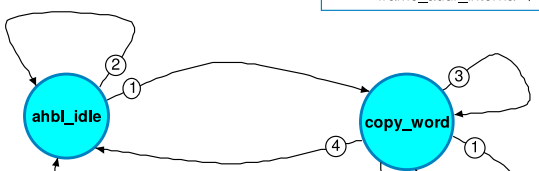
```
-- reset fiq_cld ???
fiq_int <= '0';

-- update toggle signal
word_change_toggle_old <= word_change_toggle;
-- copy destination address from ethernet register file
sp_HADDR_cld <= frame_dest;
sp_HTRANS_cld <= HTRANS_NONSEQ;
--pipelining
if frame_size >= 4 then
sp_HSIZE_cld <= HSIZE_DWORD;
sp_HWDATA_buf <= word;
counter <= frame_size -4;
else
sp_HSIZE_cld <= HSIZE_BYTE;
sp_HWDATA_buf (31 downto 8) <= (others => '0');
sp_HWDATA_buf (7 downto 0) <= word(7 downto 0);
word_buf (23 downto 0) <= word(31 downto 8);
counter <= frame_size -1;
end if;

sp_HWRITE_cld <= '1';
frame_addr_internal <= unsigned(frame_dest) + 4;
```

```
-- No start signal, stay idle.
sp_HADDR_cld <= (others=>'0');
sp_HSIZE_cld <= HSIZE_DWORD;
sp_HTRANS_cld <= HTRANS_IDLE;
sp_HWDATA_cld <= (others=>'0');
sp_HWRITE_cld <= '0';

frame_addr_internal <= (others => '0');
```



sp_HREADY = '0' or (word_change_toggle = word_change_toggle_old

```
-- The selected bus slave is not
-- ready. Wait and hold the control
-- lines until it becomes ready again.
```

word_change_toggle /= word_change_toggle_old and sp_HREADY = '1' and c

```
-- update toggle signal
word_change_toggle_old <= word_change_toggle;

-- Put the next word on the bus.
sp_HADDR_cld <= frame_addr_internal;
sp_HSIZE_cld <= HSIZE_DWORD;
sp_HTRANS_cld <= HTRANS_NONSEQ;
sp_HWDATA_buf <= word; --pipelining
sp_HWRITE_cld <= '1';

counter <= counter - 4;
frame_addr_internal <= frame_addr_internal + 4;
```

sp_HREADY = '1' AND counter <= 0

```
-- No words left:
-- Memset operation done.
sp_HTRANS_cld <= HTRANS_IDLE;
fiq_int <= '1';
```

sp_HREADY = '1' AND counter <= 0

```
-- No words left:
-- Memset operation done.
sp_HTRANS_cld <= HTRANS_IDLE;
fiq_int <= '1';
```

sp_HREADY = '1' and counter > 0

```
-- update toggle signal
word_change_toggle_old <= word_change_toggle;

-- Put the next word on the bus.
sp_HADDR_cld <= frame_addr_internal;
sp_HSIZE_cld <= HSIZE_BYTE;
sp_HTRANS_cld <= HTRANS_NONSEQ;
--sp_HWDATA_buf(31 downto 8) <= (others => '0');
sp_HWDATA_buf <= word;--_buf(7 downto 0); --pipelining
sp_HWRITE_cld <= '1';

--word_buf(23 downto 0) <= word_buf(31 downto 8);

counter <= counter - 1;
frame_addr_internal <= frame_addr_internal + 1;
```

sp_HREADY = '0' and counter > 0

word_change_toggle /= word_change_toggle_old and sp_HREADY = '1' and counter < 4

```
-- update toggle signal
word_change_toggle_old <= word_change_toggle;

-- Put the next word on the bus.
sp_HADDR_cld <= frame_addr_internal;
sp_HSIZE_cld <= HSIZE_BYTE;
sp_HTRANS_cld <= HTRANS_NONSEQ;
--sp_HWDATA_buf(31 downto 8) <= (others => '0');
sp_HWDATA_buf <= word;--(7 downto 0); --pipelining
sp_HWRITE_cld <= '1';

--word_buf(23 downto 0) <= word(31 downto 8);

counter <= counter - 1;
frame_addr_internal <= frame_addr_internal + 1;
```

<company name>		Project: armlab
Title:	<enter diagram title here>	<enter comments here>
Path:	misc/eth_bm/fsm	
Edited:	by nx on 22 Sep 2017	

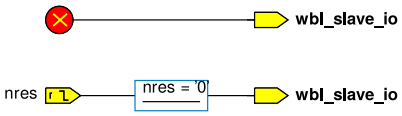
Global Actions **Concurrent Statements** **Architecture Declarations** **State Register Statements** **Process Declarations**
Pre Actions: **Clocked Process:**
Post Actions: **Output Process:**

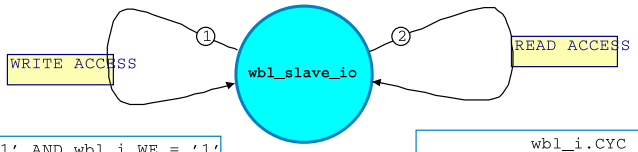
Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY global;
USE global.ethernet_reg.ALL;
USE global.global_defs.ALL;

clk | clk'EVENT AND clk = '1'

Signal Status

SIGNAL	MODE	DEFAULT	RESET	SCHEME
mac_dest	LOCAL	mac_dest	X"AAAAAAAAAAAA"	CLKD
frame_dest	LOCAL	frame_dest	X"DEADBEEF"	CLKD
MAC_enable	OUT	MAC_enable_cld	'0'	CLKD
wbl_o	OUT	(DAT=>X"00DEAD00")		COMB
frame_dest_out	OUT	frame_dest	X"DEADBEEF"	CLKD
mac_dest_out	OUT	mac_dest	X"AAAAAAAAAAAA"	CLKD
frame_dbg	LOCAL	frame_dbg_in	(others => '0')	CLKD
sdf_state	LOCAL	sdf_state_in	(others => '0')	CLKD
sdf_state_last	LOCAL	sdf_state_last_in	(others => '0')	CLKD





wbl_i.CYC = '1' AND wbl_i.WE = '1'

```
-- Write access
CASE wbl_i.ADR(ETHERNET_RANGE) IS
W    HEN ADR_ETHERNET_MAC_ADDR_HIGH =>
m        ac_dest(47 downto 16) <= wbl_i.DAT;
W    HEN ADR_ETHERNET_MAC_ADDR_LOW =>
m        ac_dest(15 downto 0) <= wbl_i.DAT(15 downto 0);
-    - WHEN ADR_ETHERNET_FRAME_LENGTH =>
-        - frame_length <= wbl_i.DAT;
W    HEN ADR_ETHERNET_FRAME_DEST =>
f        rame_dest <= unsigned(wbl_i.DAT);
W  HEN ADR_ETHERNET_MODE =>
M     AC_enable_cld <= wbl_i.DAT(0);

W    HEN others =>
R        EPORT "Invalid write access to address " &
T             o_Hex(wbl_i.ADR)
S             EVERITY WARNING;
END CASE;
```

wbl_i.CYC = '1' AND wbl_i.WE = '0'

```
-- Read access
CASE wbl_i.ADR(ETHERNET_RANGE) IS
W    HEN ADR_ETHERNET_MAC_ADDR_HIGH =>
w       bl_o.DAT <= mac_dest(47 downto 16);
W    HEN ADR_ETHERNET_MAC_ADDR_LOW =>
w       bl_o.DAT <= (others => '0');
w       bl_o.DAT(15 downto 0) <= mac_dest(15 d
W    HEN ADR_ETHERNET_FRAME_LENGTH =>
w       bl_o.DAT <= (others => '0');
w       bl_o.DAT(15 downto 0) <= std_logic_vec
W    HEN ADR_ETHERNET_FRAME_DEST =>
w       bl_o.DAT <= std_logic_vector(frame_des
W  HEN ADR_ETHERNET_MODE =>
w     bl_o.DAT <= (0 => MAC_enable_cld, others

-   - more debug registers
W  HEN ADR_ETHERNET_DEBUG_MAC_DEST_HIGH =>
w     bl_o.DAT <= rx_frame_mac_dest(47 downto
   WHEN ADR_ETHERNET_DEBUG_MAC_SRC_HIGH =>
w     bl_o.DAT <= rx_frame_mac_src(47 downto 1
W  HEN ADR_ETHERNET_DEBUG_FRAME_BEGIN =>
w     bl_o.DAT <= frame_dbg(1023 downto 992);
W  HEN ADR_ETHERNET_DBG_STATES =>
w     bl_o.DAT <= (others => '0');
w     bl_o.DAT(2 downto 0) <= sdf_state_last;
w     bl_o.DAT(6 downto 4) <= sdf_state;
w     bl_o.DAT(10 downto 8) <= parse_state;

W    HEN others =>
w       bl_o.DAT <= X"00DEAD00";
R       EPORT "Invalid read access to address
             To_Hex(wbl_i.ADR)
S            EVERITY WARNING;
END CASE;
```

<company name>		Project:armlab
Title:	<enter diagram title here>	<enter comments here>
Path:	misc/wbl_io/fsm	
Edited:	by nx on 16 Aug 2017	

B Register Definitions

Hardware modules

NOTE: you have to define module tables here, which have the name ahbl_modules, wbl_modules or test_modules. You can also add other information to this document, it will be silently ignored by the regdef flow. This will be part of your project documentation!

AHBL bus

...

WBL bus

NOTE: student_bm is already defined here, so the busmaster demo runs correctly. Extend the following table to fit your needs!

Base address	Name	Description	Address width
0x90800000	student_lights_ctrl	WBL modules	27
0x91000000	student_bm	AHBL master module	27
0x90000000	student_irq_ctrl	WBL modules	27
0x91800000	student_led_cube	WBL modules	27

Test bus

...

Ethernet register interface

Register MODE holds mode bits.

0x00	**rw**	**ETHERNET_MODE**	
bits	**name**	**description**	**default**
0	enable	Enable Module = 1, Disable Module = 0	‘0’

Register MAC_ADDR holds MAC address of FPGA.

0x04	**rw**	**ETHERNET_MAC_ADDR_HIGH**	
bits	**name**	**description**	**default**
31 .. 0	mac_dest_high	Contains upper part of MAC address of FPGA from which it is available	(others => ‘0’)

Register MAC_ADDR holds MAC address of FPGA.

0x08	**rw**	**ETHERNET_MAC_ADDR_LOW**	
bits	**name**	**description**	**default**
31 .. 0	mac_dest_low	Contains lower part of MAC address of FPGA from which it is available (valid bits 15:0)	(others => ‘0’)

Register Frame_length holds the size of the payload of the ethernet frame.

0x0C	**r**	**ETHERNET_FRAME_LENGTH**	
bits	**name**	**description**	**default**
15 .. 0	frame_length	Frame length register: holds the size of the payload of the ethernet frame	0x00000000

Register Frame_length holds the size of the payload of the ethernet frame.

0x10	**r**	**ETHERNET_FRAME_DEST**	
bits	**name**	**description**	**default**
31 .. 0	frame_dest	Frame destination register: holds the address of the memory where to put the frame	0xDEADBEEF

Register IRQ_STATUS.

0x14	**rw**	**ETHERNET_IRQ_STATUS**	
bits	**name**	**description**	**default**
0	frame_written	Indicates successful write of frame to memory, is zeroed when new frame destination is written	‘0’

Register IRQ_MODE.

0x18	**rw**	**ETHERNET_IRQ_MODE**	
bits	**name**	**description**	**default**
0	frame_written_en	Enables (1) or disables (0) the propagation of the frame_written irq	‘0’

Register DEBUG_MAC_DEST.

0x1C	**rw**	**ETHERNET_DEBUG_MAC_DEST_HIGH**	
bits	**name**	**description**	**default**
0..31	mac_dest	Holds the MAC destination address of the last received frame	‘0’

Register DEBUG_MAC_SRC.

0x20	**rw**	**ETHERNET_DEBUG_MAC_SRC_HIGH**	
bits	**name**	**description**	**default**
0..31	mac_src	Holds the MAC source address of the last received frame	‘0’

Register DEBUG_FRAME_BEGIN.

0x28	**rw**	**ETHERNET_DEBUG_FRAME_BEGIN**	
bits	**name**	**description**	**default**
0..31	frame_begin	Holds the first word of the last received frame	‘0’

Register DBG_STATES.

0xA4	**rw**	**ETHERNET_DBG_STATES**	
bits	**name**	**description**	**default**
11..8	parse_state	Current eth parse state	‘000’
7..4	sdf_state	Current sdf state	‘000’
3..0	sdf_state_last	Last sdf state	‘000’

Student led_cube register interface

Register data holds data for each RGB LED Pin.

0x00	**rw**	**STUDENT_LED_CUBE_MODE**	
bits	**name**	**description**	**default**
0	Enable	Enables LED Cube Operation	0
1	Auto Level Switch	If Bit is 0 : Cube Controller Module automatically switches levels dependent on delay in STUDENT_LED_CUBE_DELAY register If Bit is 1 : Levels will be switched using the Frame complete Interrupt	0
31 .. 1	Mode	Potential LED Cube Modes	(others =>'0')

Register level holds the currently active led cube level.

0x04	**r**	**STUDENT_LED_CUBE_LEVEL**	
bits	**name**	**description**	**default**
31 .. 0	level	Level register: hold the currently active LED Cube level (unsigned int)	0x00000000

Register delay holds the 32 bit value that delays level switching in respect to clock frequency.

0x08	**rw**	**STUDENT_LED_CUBE_DELAY**	
bits	**name**	**description**	**default**
31 .. 0	delay	Delay register: Holds 32 bit value that delays level switching in respect to clock frequency	0x0000000

The following register contain the actual LED Cube Data. Each 32 Bit Register holds all RGB values for 8 RGB LEDs

0x0C	**rw**	**STUDENT_LED_CUBE_DATA_0_7**	
bits	**name**	**description**	**default**
3 .. 0	RGB_0	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_1	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_2	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_3	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_4	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_5	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_6	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_7	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x10	**rw**	**STUDENT_LED_CUBE_DATA_8_15**	
bits	**name**	**description**	**default**
3 .. 0	RGB_8	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_9	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_10	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_11	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_12	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_13	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_14	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_15	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x14	rw	STUDENT_LED_CUBE_DATA_16_23	
bits	**name**	**description**	**default**
3 .. 0	RGB_16	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_17	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_18	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_19	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_20	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_21	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_22	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_23	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x18	rw	STUDENT_LED_CUBE_DATA_24_31	
bits	**name**	**description**	**default**
3 .. 0	RGB_24	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_25	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_26	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_27	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_28	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_29	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_30	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_31	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x1C	rw	STUDENT_LED_CUBE_DATA_32_39	
bits	**name**	**description**	**default**
3 .. 0	RGB_32	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_33	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_34	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_35	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_36	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_37	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_38	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_39	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x20	rw	STUDENT_LED_CUBE_DATA_40_47	
bits	**name**	**description**	**default**
3 .. 0	RGB_40	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_41	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_42	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_43	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_44	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_45	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x20	rw	STUDENT_LED_CUBE_DATA_40_47	
27 .. 24	RGB_46	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_47	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x24	rw	STUDENT_LED_CUBE_DATA_48_55	
bits	**name**	**description**	**default**
3 .. 0	RGB_48	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_49	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_50	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_51	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_52	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_53	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_54	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_55	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x28	rw	STUDENT_LED_CUBE_DATA_56_63	
bits	**name**	**description**	**default**
3 .. 0	RGB_56	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_57	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_58	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_59	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_60	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_61	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_62	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_63	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x2C	rw	STUDENT_LED_CUBE_DATA_64_71	
bits	**name**	**description**	**default**
3 .. 0	RGB_64	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_65	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_66	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_67	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_68	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_69	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_70	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_71	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x30	rw	STUDENT_LED_CUBE_DATA_72_79	
bits	**name**	**description**	**default**
3 .. 0	RGB_72	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_73	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_74	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_75	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x30	rw	STUDENT_LED_CUBE_DATA_72_79	
19 .. 16	RGB_76	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_77	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_78	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_79	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x34	rw	STUDENT_LED_CUBE_DATA_80_87	
bits	**name**	**description**	**default**
3 .. 0	RGB_80	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_81	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_82	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_83	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_84	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_85	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_86	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_87	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x38	rw	STUDENT_LED_CUBE_DATA_88_95	
bits	**name**	**description**	**default**
3 .. 0	RGB_88	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_89	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_90	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_91	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_92	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_93	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_94	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_95	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x3c	rw	STUDENT_LED_CUBE_DATA_96_103	
bits	**name**	**description**	**default**
3 .. 0	RGB_96	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_97	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_98	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_99	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_100	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_101	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_102	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_103	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x40	rw	STUDENT_LED_CUBE_DATA_104_111	
bits	**name**	**description**	**default**
3 .. 0	RGB_104	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_105	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_106	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x40	rw	STUDENT_LED_CUBE_DATA_104_111	
15 .. 12	RGB_107	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_108	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_109	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_110	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_111	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x44	rw	STUDENT_LED_CUBE_DATA_112_119	
bits	**name**	**description**	**default**
3 .. 0	RGB_112	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_113	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_114	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_115	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_116	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_117	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_118	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_119	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x48	rw	STUDENT_LED_CUBE_DATA_120_127	
bits	**name**	**description**	**default**
3 .. 0	RGB_120	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_121	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_122	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_123	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_124	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_125	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_126	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_127	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x4c	rw	STUDENT_LED_CUBE_DATA_128_135	
bits	**name**	**description**	**default**
3 .. 0	RGB_128	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_129	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_130	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_131	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_132	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_133	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_134	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_135	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x50	rw	STUDENT_LED_CUBE_DATA_136_143	
bits	**name**	**description**	**default**
3 .. 0	RGB_136	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x50	rw	STUDENT_LED_CUBE_DATA_136_143	
7 .. 4	RGB_137	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_138	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_139	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_140	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_141	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_142	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_143	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x54	rw	STUDENT_LED_CUBE_DATA_144_151	
bits	**name**	**description**	**default**
3 .. 0	RGB_144	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_145	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_146	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_147	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_148	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_149	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_150	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_151	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x58	rw	STUDENT_LED_CUBE_DATA_152_159	
bits	**name**	**description**	**default**
3 .. 0	RGB_152	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_153	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_154	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_155	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_156	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_157	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_158	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_159	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x5c	rw	STUDENT_LED_CUBE_DATA_160_167	
bits	**name**	**description**	**default**
3 .. 0	RGB_160	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_161	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_162	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_163	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_164	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_165	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_166	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_167	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x60	rw	STUDENT_LED_CUBE_DATA_168_175	
bits	**name**	**description**	**default**
3 .. 0	RGB_168	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_169	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_170	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_171	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_172	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_173	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_174	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_175	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x64	rw	STUDENT_LED_CUBE_DATA_176_183	
bits	**name**	**description**	**default**
3 .. 0	RGB_176	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_177	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_178	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_179	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_180	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_181	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_182	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_183	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x68	rw	STUDENT_LED_CUBE_DATA_184_191	
bits	**name**	**description**	**default**
3 .. 0	RGB_184	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_185	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_186	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_187	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_188	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_189	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_190	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_191	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x6c	rw	STUDENT_LED_CUBE_DATA_192_199	
bits	**name**	**description**	**default**
3 .. 0	RGB_192	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_193	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_194	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_195	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_196	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_197	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_198	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_199	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x70	rw	STUDENT_LED_CUBE_DATA_200_207	
bits	**name**	**description**	**default**
3 .. 0	RGB_200	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_201	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_202	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_203	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_204	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_205	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_206	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_207	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

0x74	rw	STUDENT_LED_CUBE_DATA_208_215	
bits	**name**	**description**	**default**
3 .. 0	RGB_208	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
7 .. 4	RGB_209	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
11 .. 8	RGB_210	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
15 .. 12	RGB_211	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
19 .. 16	RGB_212	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
23 .. 20	RGB_213	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
27 .. 24	RGB_214	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0
31 .. 28	RGB_215	Bit 0: Red, Bit 1: Green, Bit 2: Blue, Bit 3: Dummy	0x0

Register delay holds the 32 bit value that delays the switching between level drivers

0x78	rw	STUDENT_LED_CUBE_GAP_DELAY	
bits	**name**	**description**	**default**
31 .. 0	gap_delay	Delay register: Holds 32 bit value that delays the switching between level drivers	0x0000000

Student IRQ_Ctrl register interface

Register all_en enable FIQ/IRQ.

0x00	**rw**	**STUDENT_IRQ_CTRL_ALL_EN**	
bits	**name**	**description**	**default**
31 .. 0	all_en	1 = Enable all interrupts. 0 = Disable all interrupts.	0x00000000

Switch between normal and testing mode.

0x04	**rw**	**STUDENT_IRQ_CTRL_TEST_EN**	
bits	**name**	**description**	**default**
31 .. 0	mode	1 = Testing mode. 0 = Normal mode.	0x00000000

Set IRQ in testing mode.

0x08	**rw**	**STUDENT_IRQ_CTRL_TEST_IRQ**	
bits	**name**	**description**	**default**
31 .. 0	irq_set	In testing mode: Every bit represents a specific interrupt.	0x00000000

Register status holds the FIQ/IRQ set-flags of all interrupts.

0x14	**r**	**STUDENT_IRQ_CTRL_ALL_STATUS**	
bits	**name**	**description**	**default**
31 .. 0	all_status	Set-flags of all interrupts. Interrupt 0 to 31	0x00000000

Register all_no holds number of the lowest interrupt set.

0x18	**r**	**STUDENT_IRQ_CTRL_ALL_NO**	
bits	**name**	**description**	**default**
31 .. 0	all_no	Number of the lowest interrupt set	0x00000000

Register fiq_mask to mask interrupts as FIQs.

0x20	**rw**	**STUDENT_IRQ_CTRL_FIQ_MASK**	
bits	**name**	**description**	**default**
31 .. 0	fiq_mask	Mask specific FIQ.	0x00000000

Register fiq_status holds the FIQ set-flags.

0x24	**r**	**STUDENT_IRQ_CTRL_FIQ_STATUS**	
bits	**name**	**description**	**default**
31 .. 0	fiq_status	Set-flags of FIQs.	0x00000000

Register fiq_no holds number of the lowest FIQ set.

0x28	**r**	**STUDENT_IRQ_CTRL_FIQ_NO**	
bits	**name**	**description**	**default**
31 .. 0	fiq_no	Number of the lowest FIQ set	0x00000000

Register irq_mask to mask interrupts as IRQs.

0x30	**rw**	**STUDENT_IRQ_CTRL_IRQ_MASK**	
bits	**name**	**description**	**default**

0x30	**rw**	**STUDENT_IRQ_CTRL_IRQ_MASK**	
31 .. 0	irq_mask	Mask specific IRQ.	0x00000000

Register irq_status holds the IRQ set-flags.

0x34	**r**	**STUDENT_IRQ_CTRL_IRQ_STATUS**	
bits	**name**	**description**	**default**
31 .. 0	irq_status	Set-flags of IRQs.	0x00000000

Register fiq_no holds number of the lowest IRQ set.

0x38	**r**	**STUDENT_IRQ_CTRL_IRQ_NO**	
bits	**name**	**description**	**default**
31 .. 0	irq_no	Number of the lowest IRQ set	0x00000000

Student lights register interface

Register mode hold the mode of the running light.

0x00	**r**	**STUDENT_LIGHTS_MODE**	
bits	**name**	**description**	**default**
2 .. 0	mode	Mode register: Sets the mode of the running light 0: MODE_PING 1: MODE_RIGHT 2: MODE_LEFT 3: MODE_STOP	0x00000000

Register pattern holds the pattern to be displayed.

0x04	**r**	**STUDENT_LIGHTS_PATTERN**	
bits	**name**	**description**	**default**
7 .. 0	pattern	Pattern register: hold the currently displayed pattern	0x0000003C

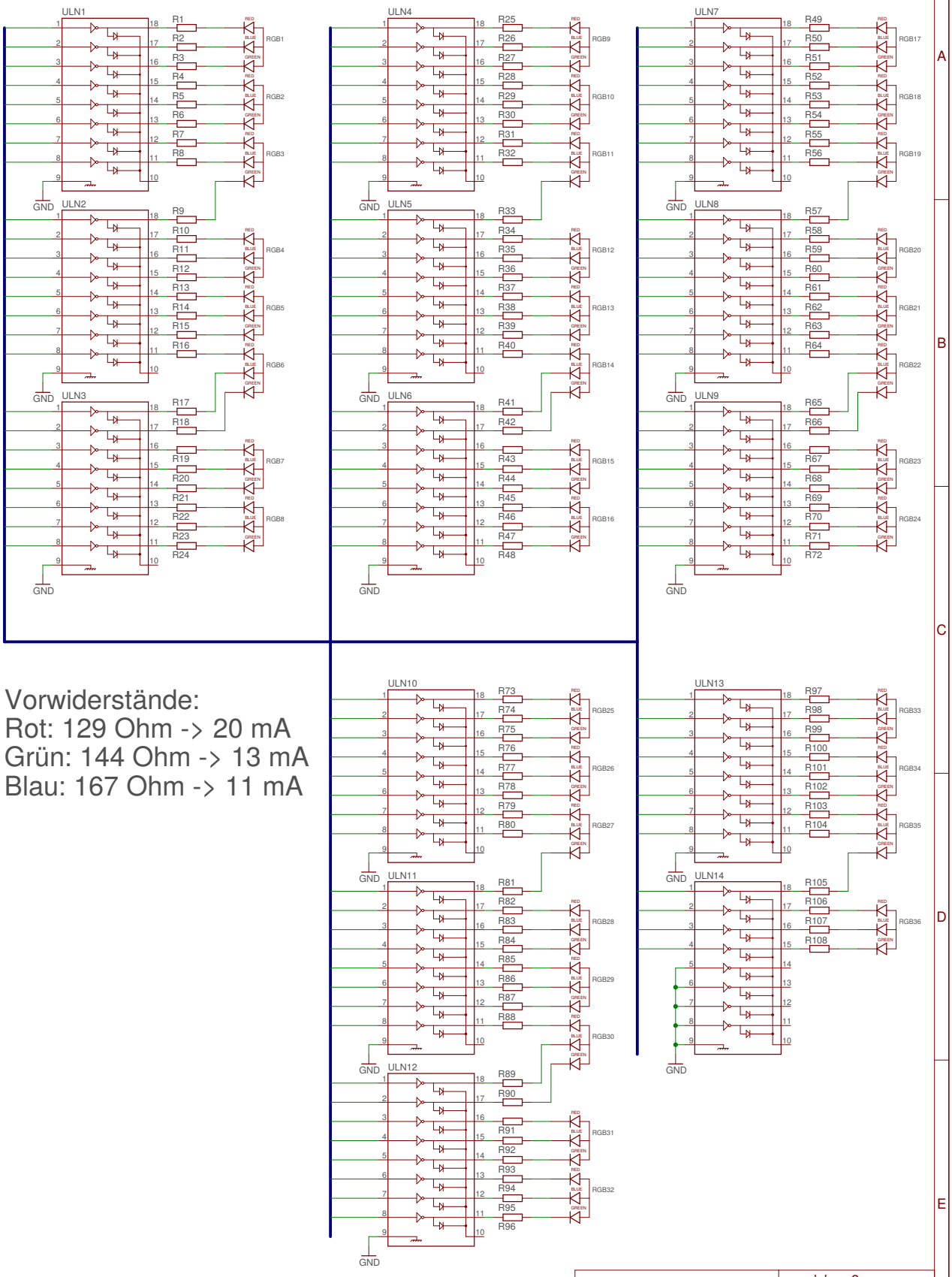
Register status holds the current status of the leds.

0x08	**r**	**STUDENT_LIGHTS_STATUS**	
bits	**name**	**description**	**default**
7 .. 0	status	Status register: Shows the current status of the running light in the lower 8 bit of the 32 bit register	pattern

Register delay specifies a delay in clock cycles.

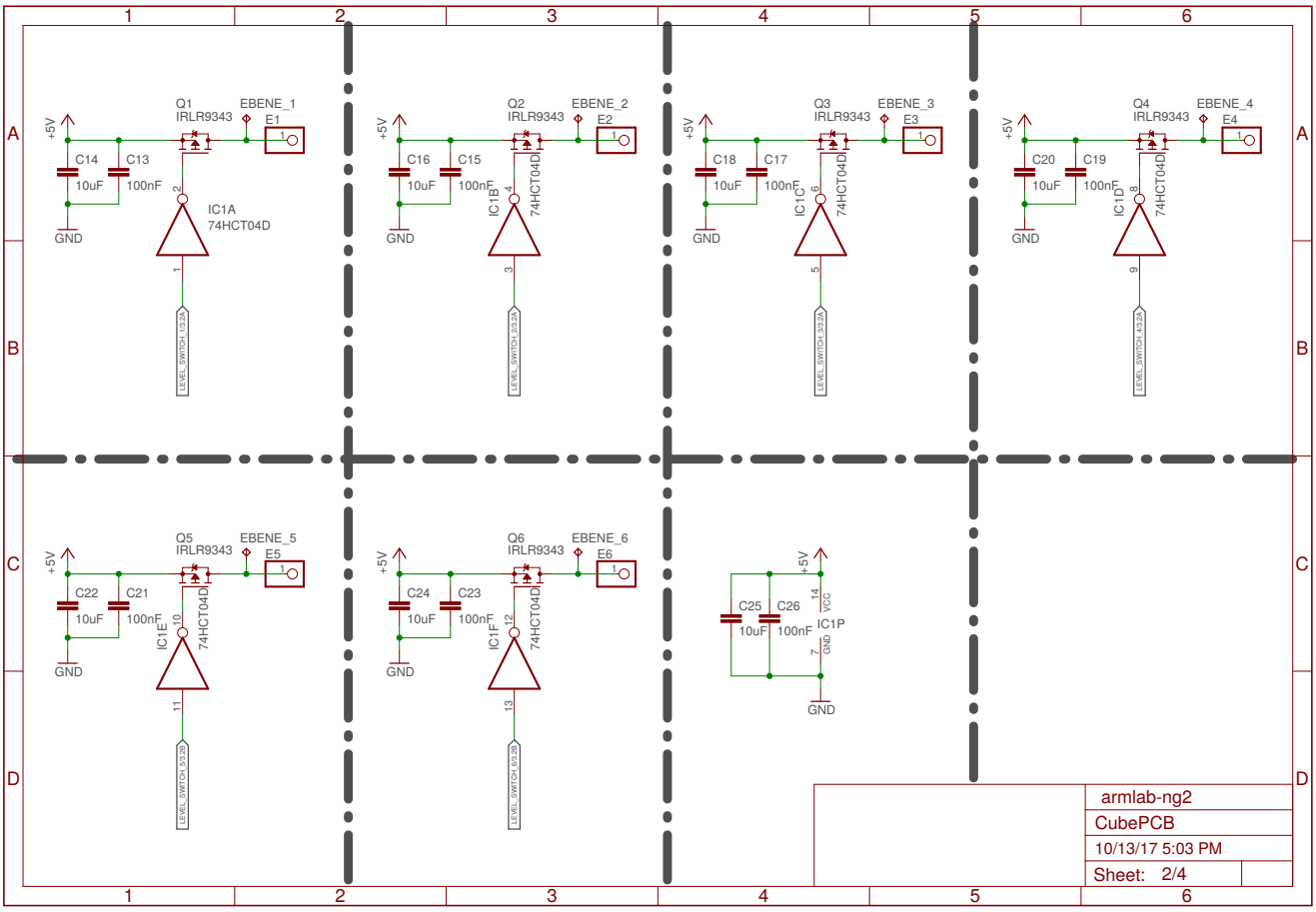
0x0C	**rw**	**STUDENT_LIGHTS_DELAY**	
bits	**name**	**description**	**default**
31 .. 0	delay	Delay register: sets the delay in clock cycles as 32 bit unsigned value	0x0000000A

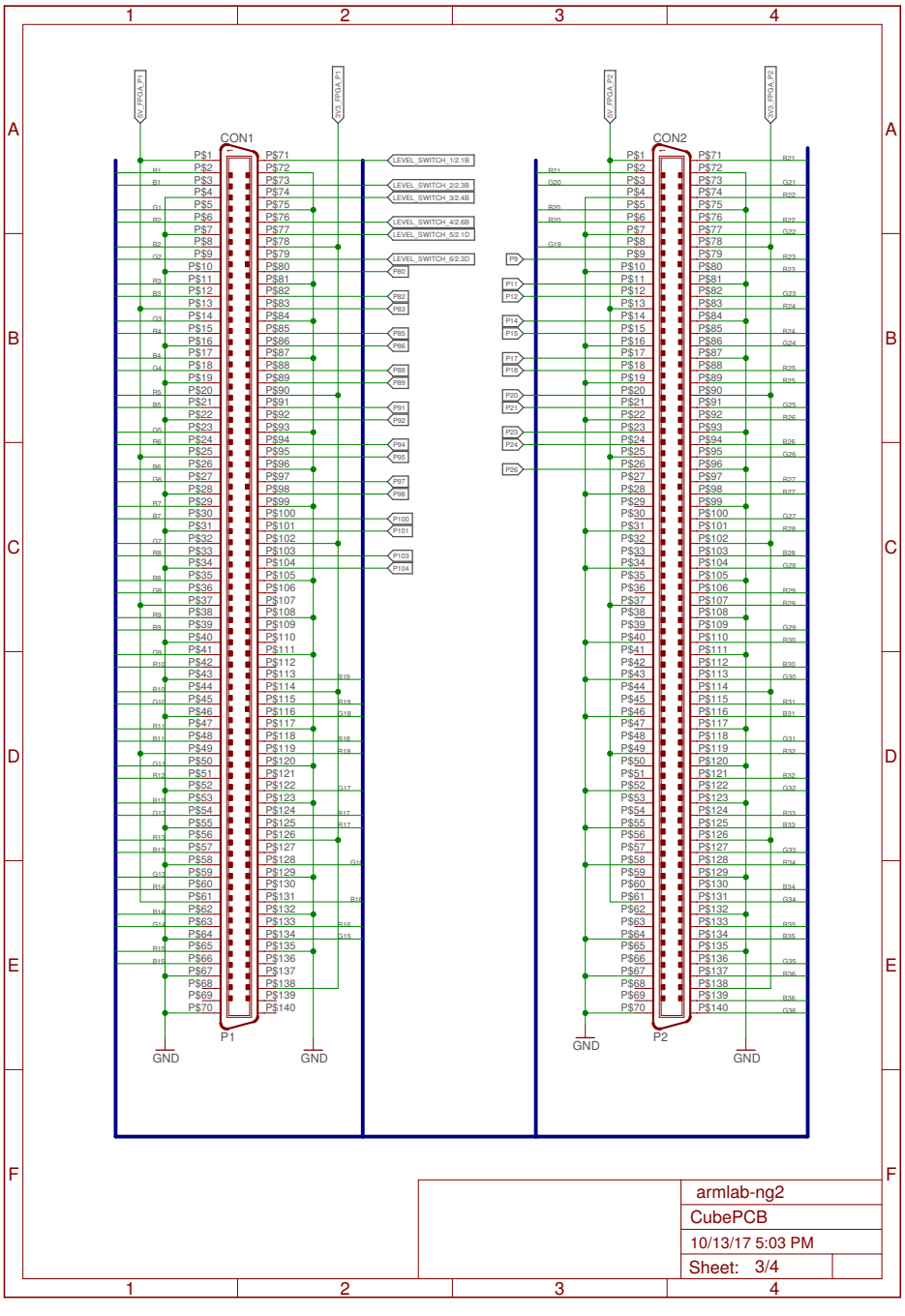
C LED-Cube PCB Files



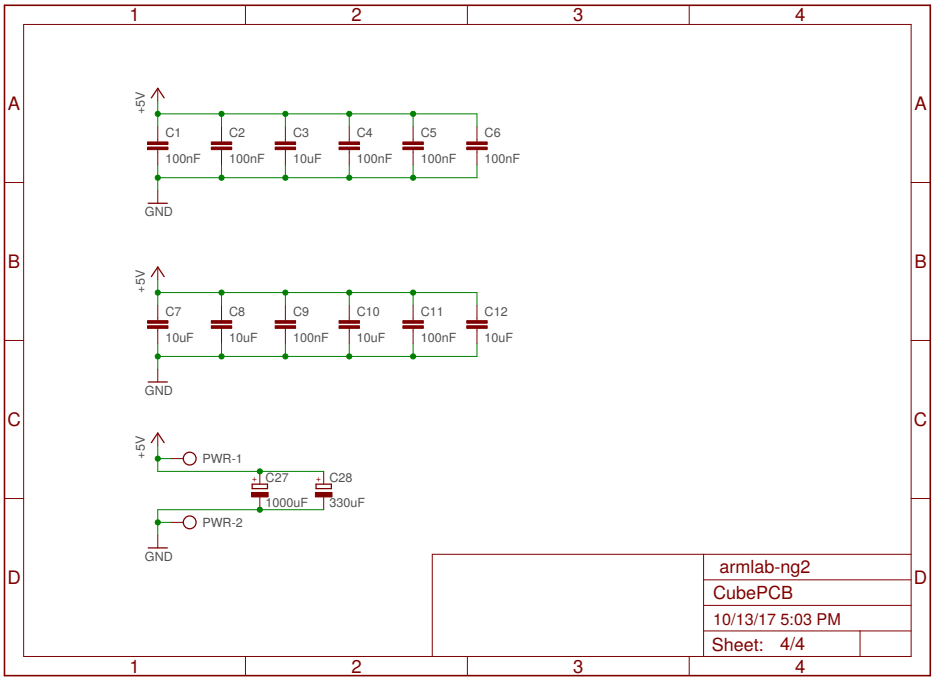
Vorwiderstände:
Rot: 129 Ohm -> 20 mA
Grün: 144 Ohm -> 13 mA
Blau: 167 Ohm -> 11 mA

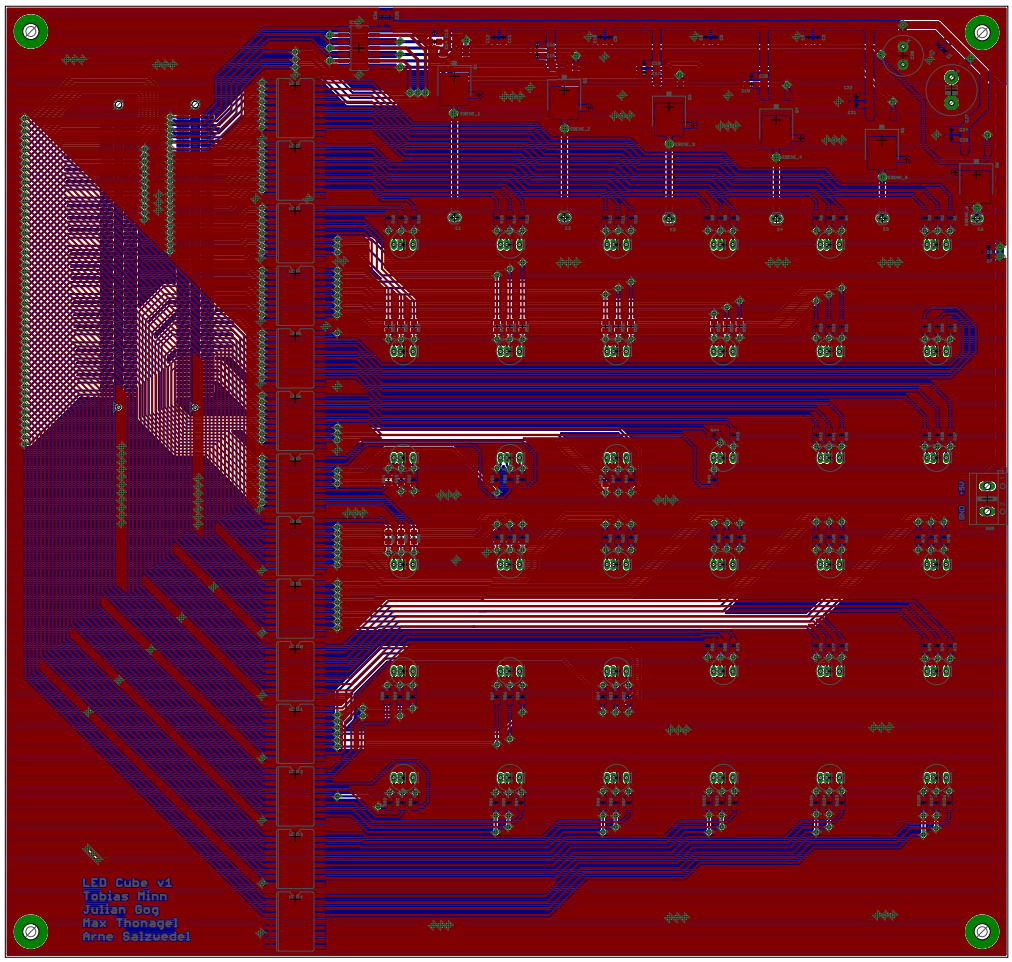
	armlab-ng2
	CubePCB
	10/13/17 5:03 PM
	Sheet: 1/4

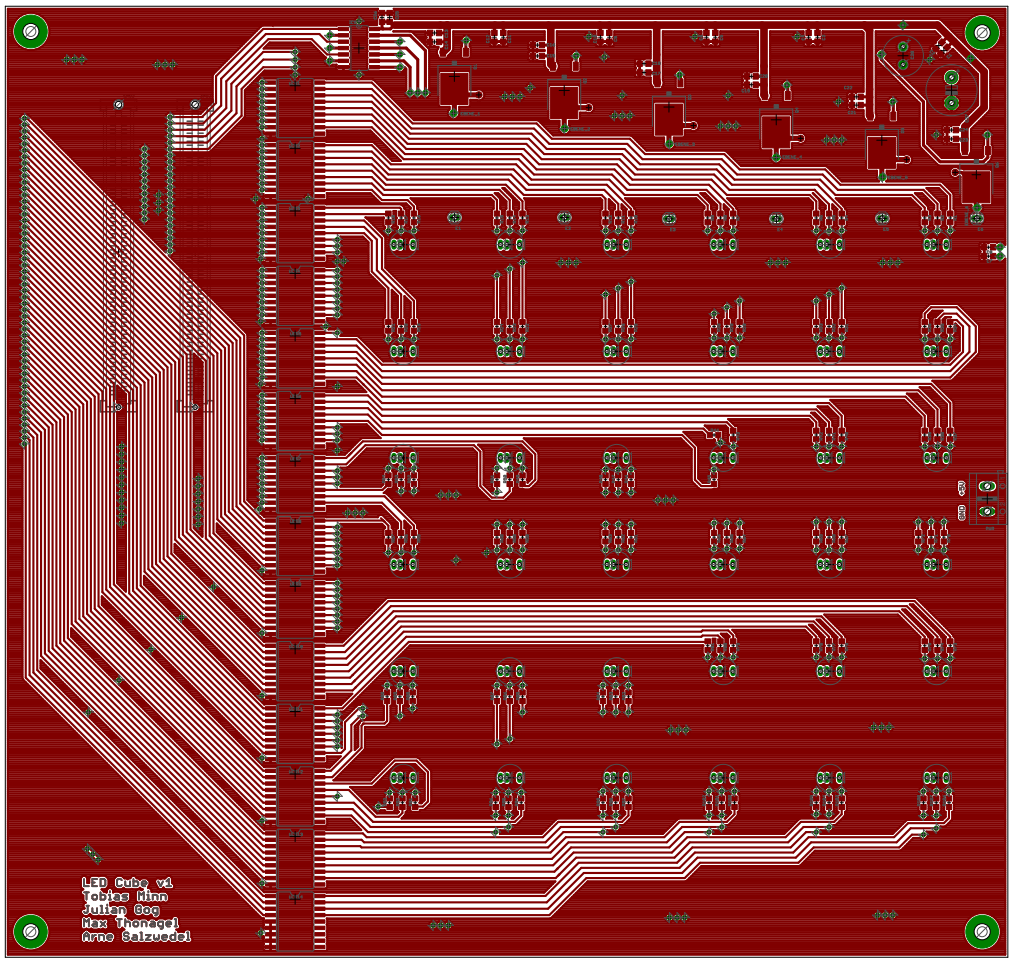

armlab-ng2
CubePCB
10/13/17 5:03 PM
Sheet: 2/4

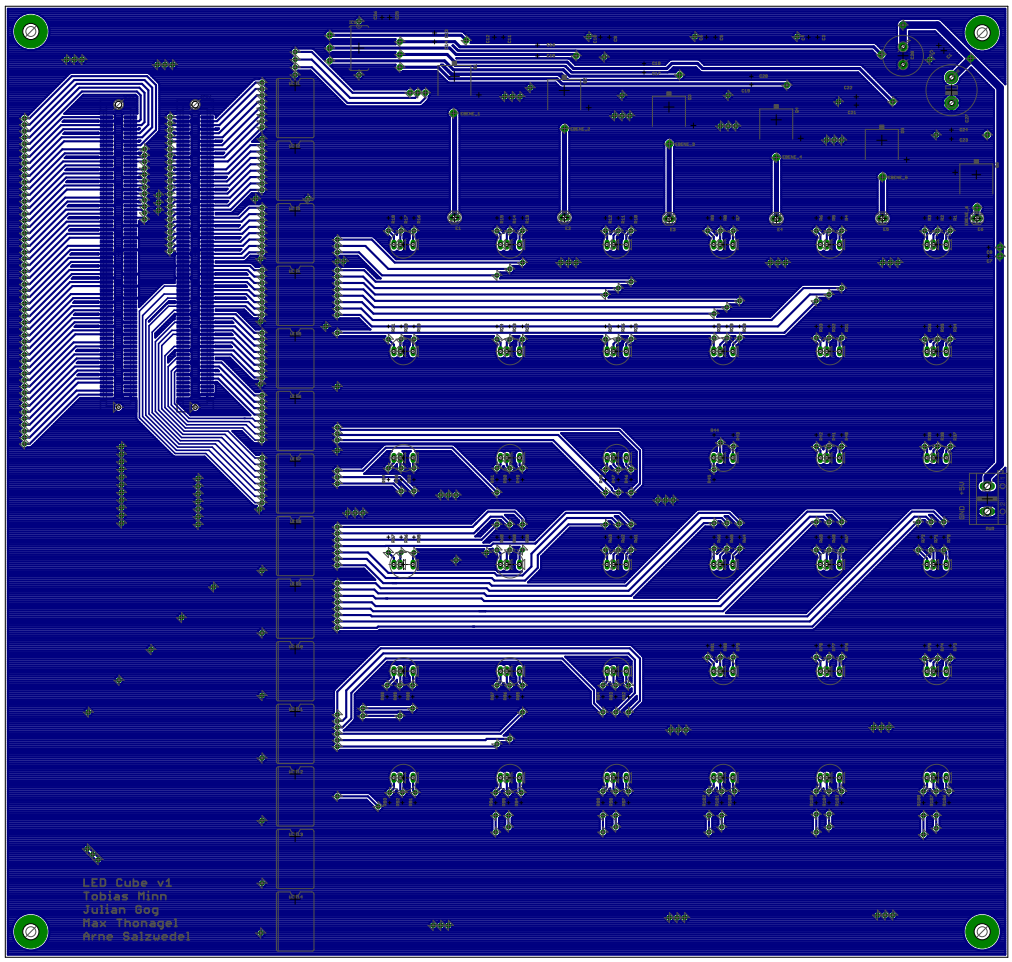


	armlab-ng2
	CubePCB
	10/13/17 5:03 PM
	Sheet: 3/4


1
2
3
4
A
B
C
D
+5V
C1
100nF
C2
100nF
C3
10uF
C4
100nF
C5
100nF
C6
100nF
GND
+5V
C7
10uF
C8
10uF
C9
100nF
C10
10uF
C11
100nF
C12
10uF
GND
+5V
PWR-1
C27
1000uF
C28
330uF
PWR-2
GND
armlab-ng2
CubePCB
10/13/17 5:03 PM
Sheet: 4/4


LED Cube v1
Tobias Minn
Julian Gog
Max Thonagel
Arne Salzuedel


LED Cube v1
Tobias Hinn
Julian Gog
Max Thonagel
Arne Salzwedel


LED Cube v1
Tobias Minn
Julian Gog
Max Thonagel
Arne Salzwedel