Lauflicht

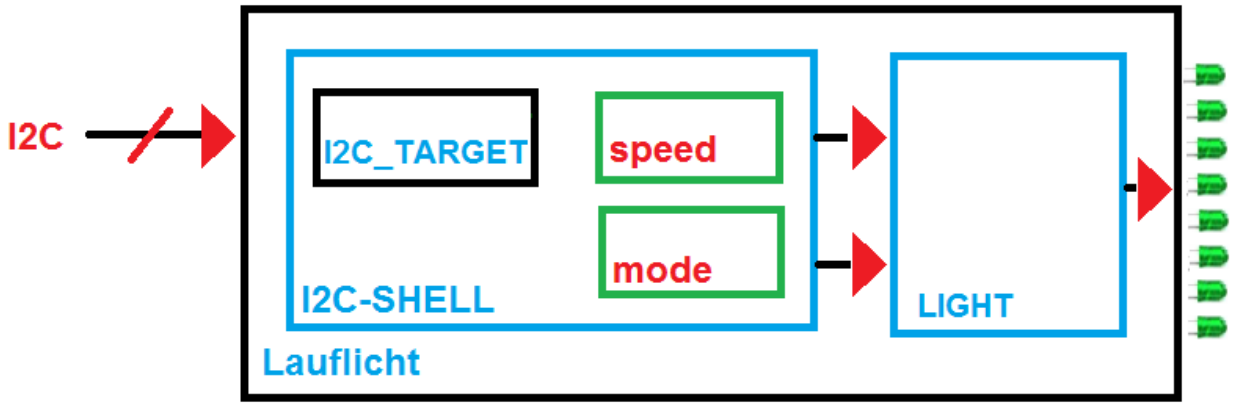
1 Inhalt

2 Abbildungsverzeichnis

3 Allgemeines

Entwicklung eines Lauflichts welches über I2C konfiguriert werden kann.

- 8 LED's
- Die Lauflicht Geschwindigkeit kann in 16 Stufen von 0,1s Schrittdauer bis 1,6s Schrittdauer eingestellt werden
- Konfiguration von 8 verschiedenen Lauflicht Mustern.



I2C_TARGET wird mit Hilfe des Quartus IP-Catalogs erstellt.

4 Hardware - Schnittstellen

4.1 Lauflicht PINS

FPGA / CYC1000 TRENZ

Signal	DIR	FPGA-PIN	BOARD ANSCHUSS	Beschreibung
res_n_i	in	N6	-	System Reset
clk_i	in	M2	-	System Clock **12.5 MHz**
I2C				
scl_i	in	R14	32	I2C Clock
sda_io	in / out	T15	31	I2C Daten
❖ *LED-OUTPUTS*				
LED0	out	M6	PRINT LED	TEST LED
LED1	out	T4	PRINT LED	TEST LED
LED2	out	T3	PRINT LED	TEST LED
LED3	out	R3	PRINT LED	TEST LED
LED4	out	T2	PRINT LED	TEST LED
LED5	out	R4	PRINT LED	TEST LED
LED6	out	N5	PRINT LED	TEST LED
LED7	out	N3	PRINT LED	TEST LED

ESP32

Signal	DIR	PIN	ESP - BOARD	Beschreibung
I2C INTERFACE				
scl_o	out	SCL / 22	PIN 32	I2C Clock
sda_io	in/out	SDA / 21	PIN 31	I2C DATA
CONFIGS				
CONF_1	in	A7 / 35	PIN 8	SE SW SPEC
CONF_2	in	A18 / 25	PIN 9	SE SW SPEC
CONF_3	in	A19 / 26	PIN 10	SE SW SPEC

5 Software Schnittstelle

5.1.1 Registerbeschreibung

5.1.1.1 SPEED- Register

I2C Register Address: 0

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	Bit(3:0)
-	-	-	-	speed(3:0)
-	-	-	-	0: step-size: 0,1s 1: step-size: 0,2s … 15: step-size:1,5s
				Default: 0x02

5.1.1.2 MODE- Register

I2C Register Address: 1

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	Bit(2:0)
-	-	-	-	-	MODE(2:0)
-	-	-	-	-	Mode 0 … Mode 7
					Default: 0x0

MODE	Bit(2:0)
0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 … 1 0 0 0 0 0 0 0
1	0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 … 1 1 0 0 0 0 0 0
2	1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 …

3	0 1	1 0	0 1	1 0	0 1	1 0	0 1	1 0
4	1 0 0	1 1 0	0 1 1	0 0 1	1 0 0	1 1 0	0 1 1	0 0 1
5	-							
6	-							
7	-							

6 Implementierung

6.1 Erstellen des Projekts in Quartus

Projektverzeichnis: <…/Lauflicht_cyc100/synthese>

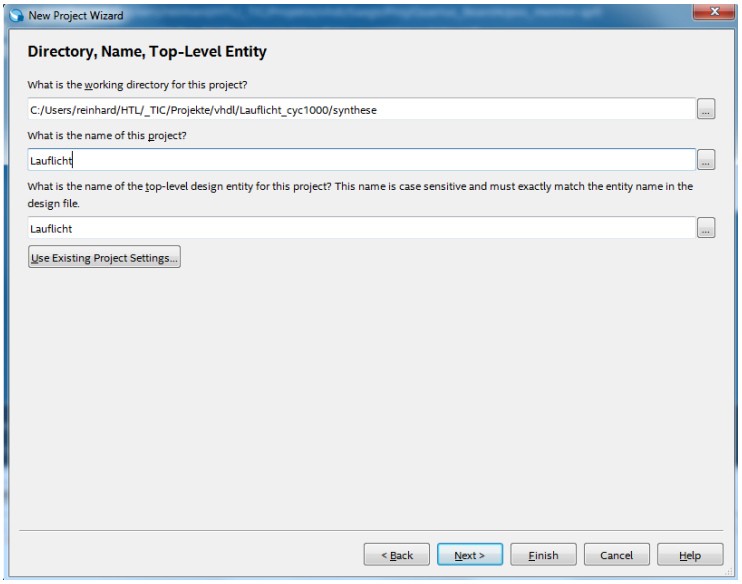


Abbildung 1 Synthese Verzeichnis + Top-Level Entity

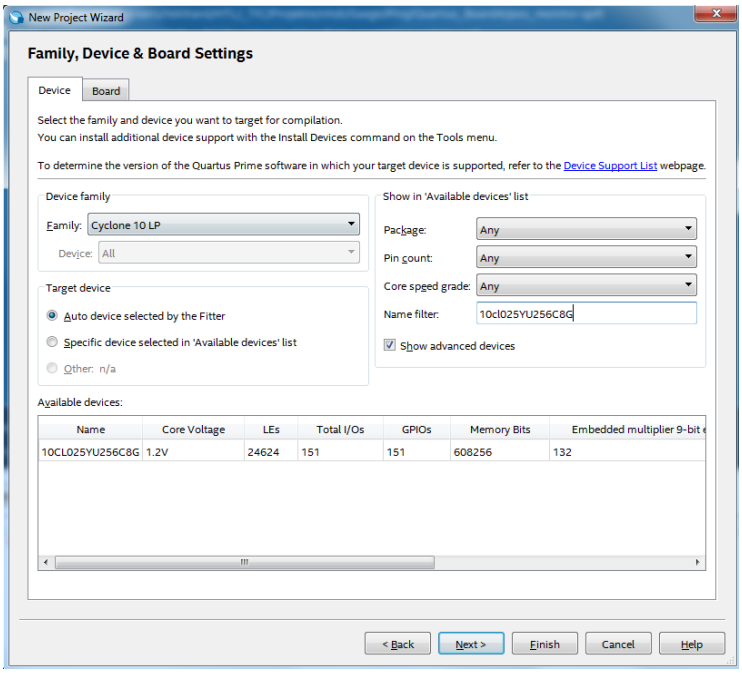


Abbildung 2 Auswahl des FPGA für das TRENZ Board CYC100 <10CL025YU256C8G>

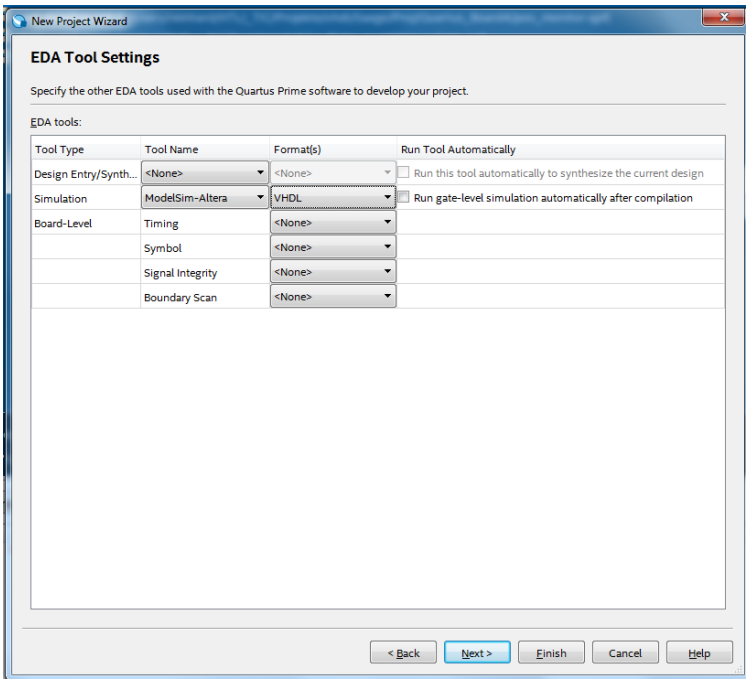


Abbildung 3 Simulation-Einstellung: Modelsim Altera

Summary

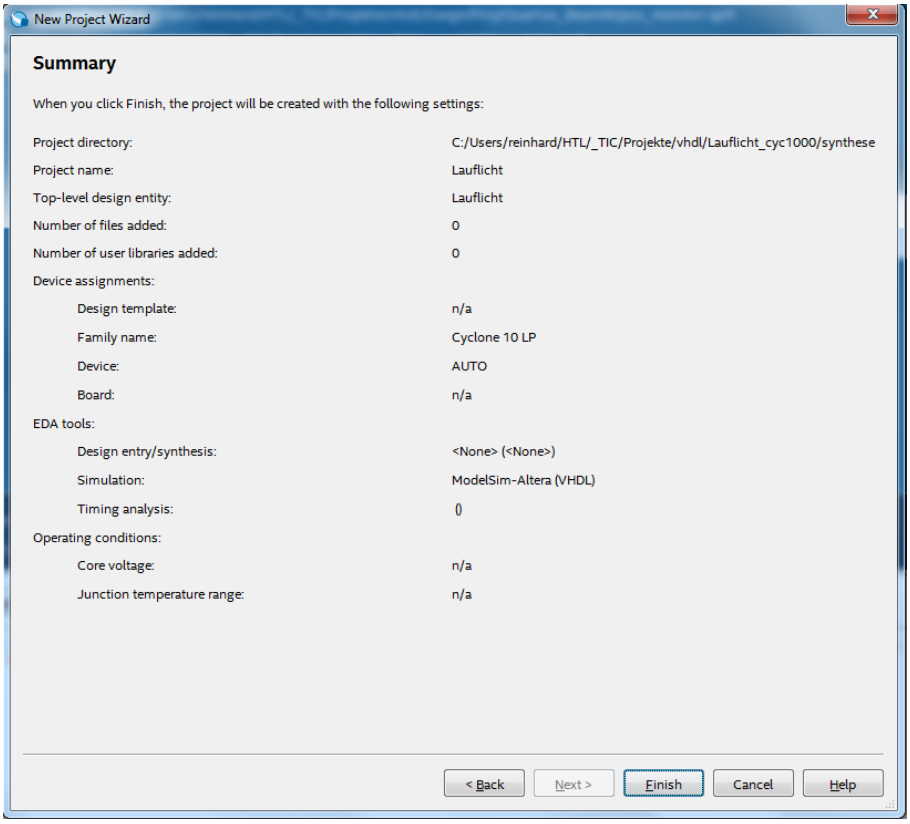


Abbildung 4 Summary

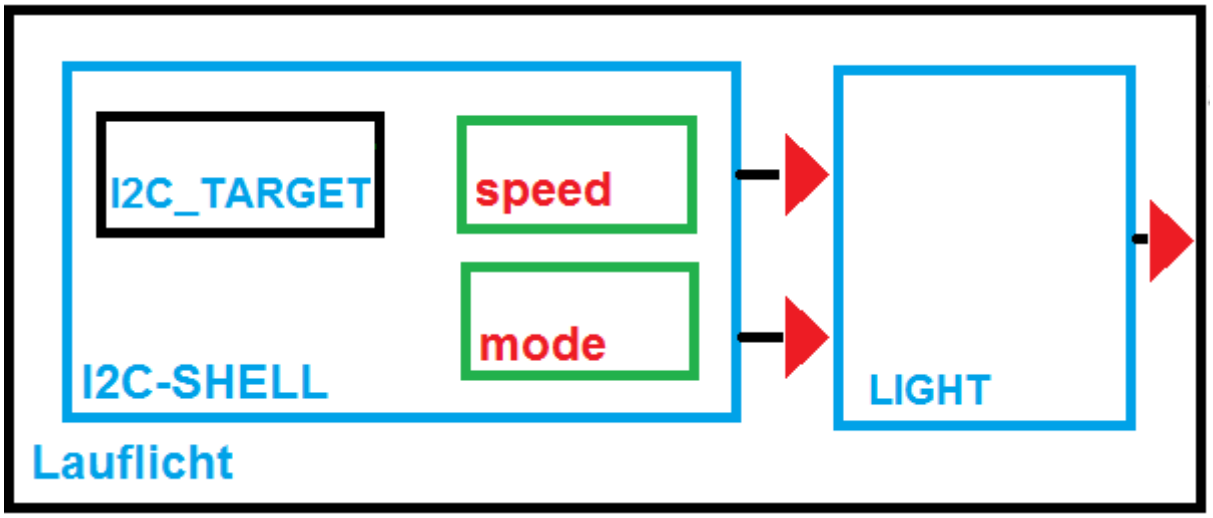
12.05.2019

6.2 Pin-Belegung FPGA

Signal	DIR	FPGA-PIN	Beschreibung
res_n_i	in	N6	System Reset
clk_i	in	M2	System Clock **12.5 MHz**
scl_i	in	R14	I2C Clock
sda_io	in / out	T15	I2C Daten
LED0	out	M6	TEST LED
LED1	out	T4	TEST LED
LED2	out	T3	TEST LED
LED3	out	R3	TEST LED
LED4	out	T2	TEST LED
LED5	out	R4	TEST LED
LED6	out	N5	TEST LED
LED7	out	N3	TEST LED

6.3 Komponenten des Lauflichts

6.3.1 Übersicht der einzelnen Komponenten des Lauflichts



- Top-Entity Lauflicht
- I2C_TARGET (Erstellen eines I2C Slave Device mit Hilfe des Quartus IP CATALOG)
- I2C_SHELL (Lauflicht Register über I2C schreibbar/lesbar)
- Light (LED Signalgenerierung)

6.3.2 TOP-Entity (Lauflicht)

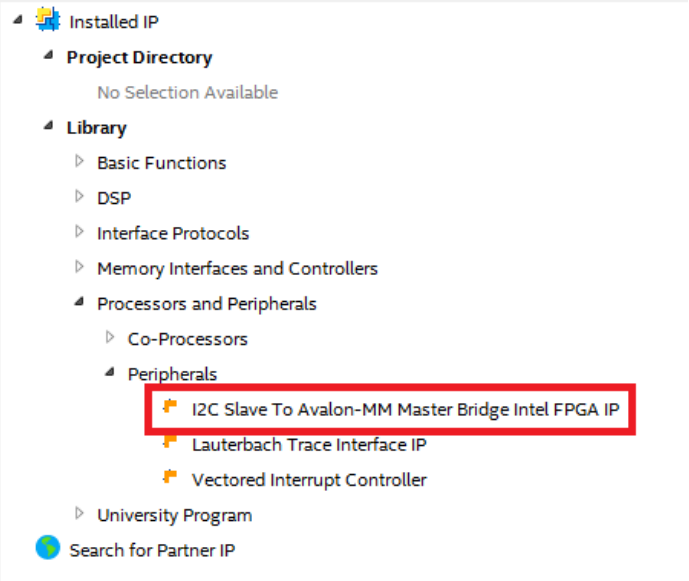
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4
5   entity lauflicht is
6       port (
7
8               -- --------------------
9               -- system           --
10              -- --------------------
11
12              res1_n_i            : in std_ulogic;  -- system reset
13              sys_clk_i           : in std_ulogic;  -- system clock
14
15              scl                 : in std_logic;
16              sda_io              : inout std_logic;
17
18              led_o               : out std_ulogic
19      );
20
21      architecture lauflicht_rtl of lauflicht is
22
23
24      begin
25
26
27      end lauflicht_rtl;
```

Abbildung 5 Signale der TOP-LEVEL Entity

6.3.3 I2C_Target

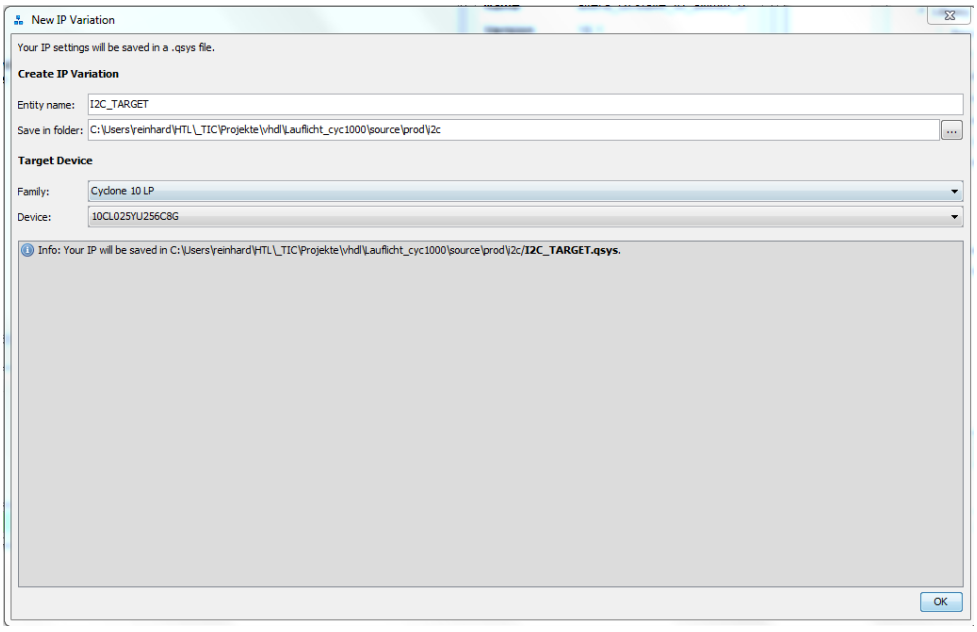
6.3.3.1 Erstellen der I2C-Avalon Bridge (Quartus IP Katalog)

Die Lauflicht Register werden über die I2C Schnittstelle beschrieben bzw. sind auch über I2C Lesbar (Erweiterung). Ab Quartus 16.X wird die I2C-Memory master – Avalon Bridge angeboten.



- Verzeichnis und FPGA festlegen

⇨ **Es sollte die Bridge in einem eigenen Verzeichnis erstellt werden, um die Übersichtlichkeit des Projekts nicht zu gefährden.**



- Erstellen der I2C - Avalon Memory Bridge (Slave!)

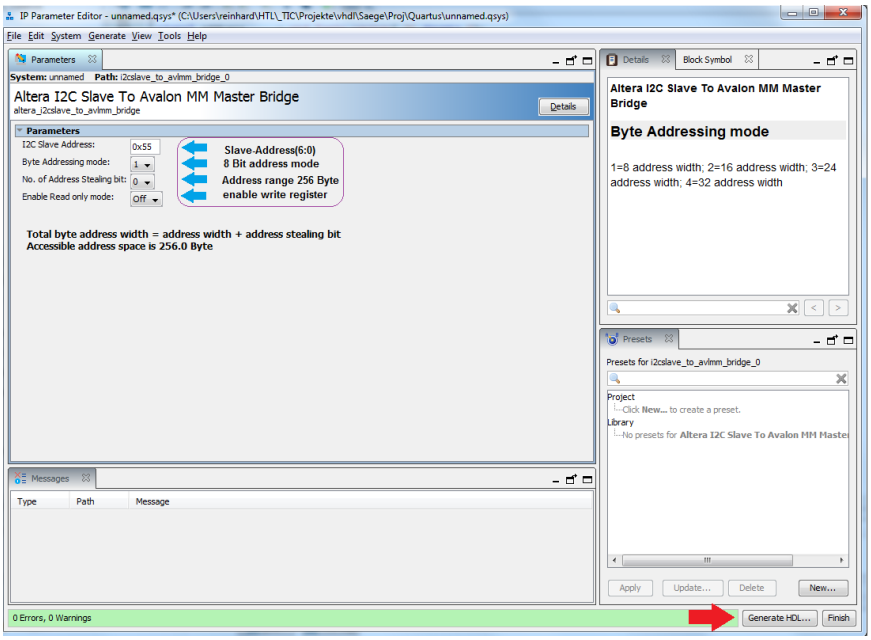


Abbildung 6 Konfiguration der I2C Avalon Bridge

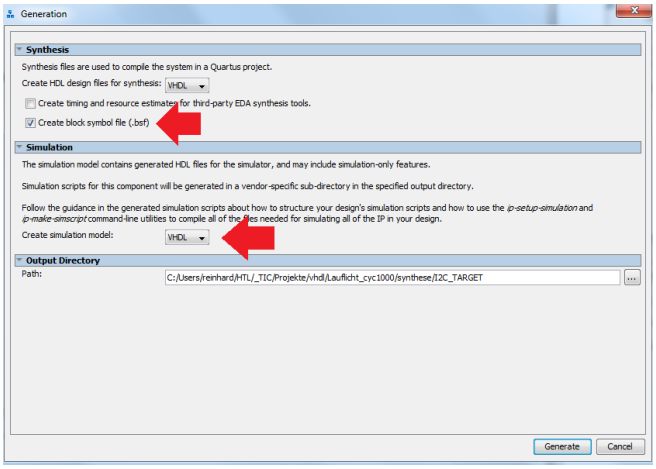
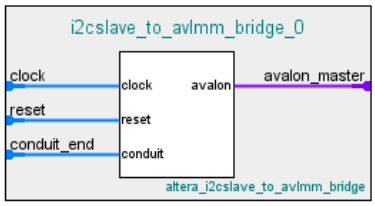


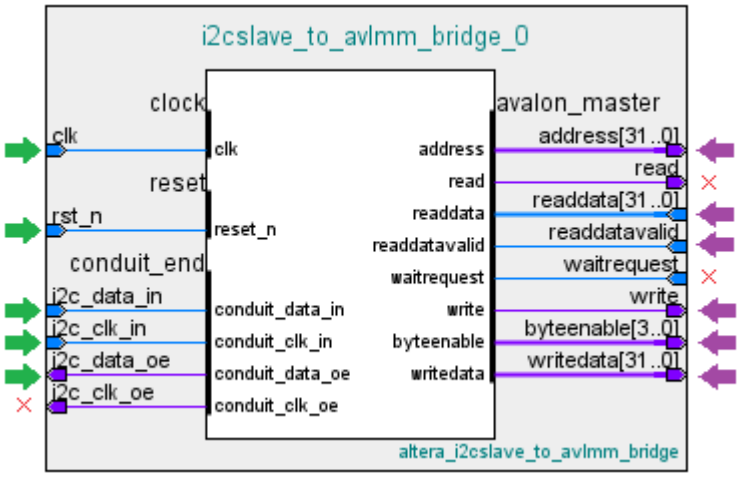
Abbildung 7 Generieren eines Simulationsmodells

- Erstellte Komponenten



- Schnittstellen

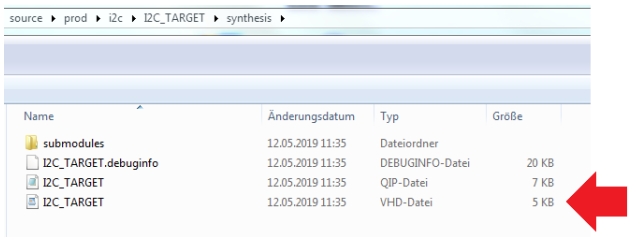
Für das Lauflicht benötigen wir die mit Pfeilen markierten Signale

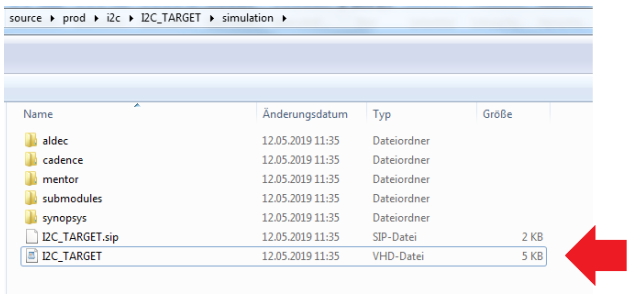


6.3.3.2 I2C Bridge zum Projekt hinzufügen.

Für die Synthese muss nur das File I2C_TARGET.qsys includiert werden. In der I2C Shell (Entity I2C_SHELL) wird die Komponente I2C_TARGET hinzugefügt.

QSYS erzeugt folgendes vhdl File im Verzeichnis Synthese und Simulation:





In beiden Verzeichnissen befinden sich zwei identische Files <I2C_TARGET.VHD>.

```
C:\Users\reinhard\HTL\_TIC\Projekte\vhdl\Lauflicht_cyc1000\source\prod\i2c\I2C_TARGET>cmp ./simulation/I2C_TARGET.vhd ./synthesis/I2C_TARGET.vhd
C:\Users\reinhard\HTL\_TIC\Projekte\vhdl\Lauflicht_cyc1000\source\prod\i2c\I2C_TARGET>
```

In den Verzeichnissen <submodules> können die verilog Files unterschiedlich sein. Quartus findet die erforderlichen submoduls selbstständig.

6.3.4 I2C_Shell

Funktion:

Schreiben / *Lesen* der Register speed(3:0), mode(3:0) über I2C. Die Shell verwendet keinen TRI Bus, generiert aber das Signal sda_oe für den TRI Buffer in der Top Entity.

Signal	*DIR*	Beschreibung
res_n_i	*in*	System Reset
clk_i	*in*	System Clock **12.5 MHz**
scl_i	*in*	I2C Clock
sda_i	*in*	I2C Empfangs Daten
sda_o	*out*	I2C Sende Daten
sda_oe	*out*	IC2 Data OE
mode_o(3:0)	*out*	mode Register
mode_valid_o	*out*	*mode data valid*
speed_o(3:0)	*in*	speed Register
speed_valid_o	*out*	*speed data valid*

Die I2C_SHELL verwendet die Komponente I2C_TARGET

```
component i2c_target is
port (
        address       : out std_logic_vector(31 downto 0);                    -- avalon_master.address
        read          : out std_logic;                                        --              .read
        readdata      : in  std_logic_vector(31 downto 0) := (others => '0'); --      do st        .readdata
        readdatavalid : in  std_logic                     := '0';             --              .readdatavalid
        waitrequest   : in  std_logic                     := '0';             --              .waitrequest
        write         : out std_logic;                                        --              .write
        byteenable    : out std_logic_vector(3 downto 0);                     --              .byteenable
        writedata     : out std_logic_vector(31 downto 0);                    --              .writedata
        clk           : in  std_logic                     := '0';             --         clock.clk
        i2c_data_in   : in  std_logic                     := '0';             --   conduit_end.conduit_data_in
        i2c_clk_in    : in  std_logic                     := '0';             --              .conduit_clk_in
        i2c_data_oe   : out std_logic;                                        --              .conduit_data_oe
        i2c_clk_oe    : out std_logic;                                        --              .conduit_clk_oe
        rst_n         : in  std_logic                     := '0'              --         reset.reset_n
  );
end component i2c_target;
```

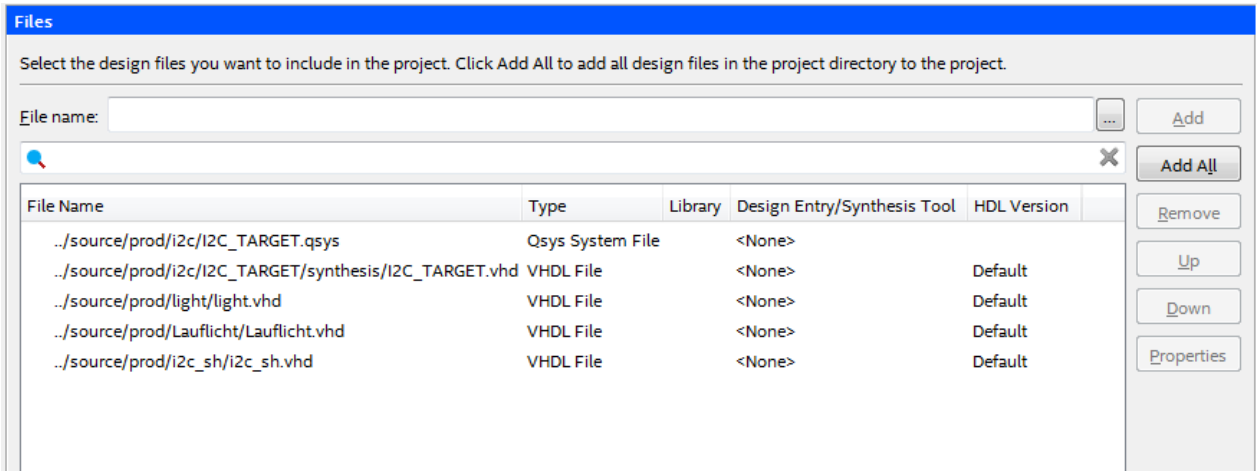
```
    i2c_target_1 : i2c_target
    port map (
        address       => address,
        read          => open,
        readdata      => i2c_test,
        readdatavalid => vcc,
        waitrequest   => gnd,
        write         => wr,
        byteenable    => be,
        writedata     => wr_data,
        clk           => sys_clk_i,

        i2c_data_in   => sda_i,
        i2c_clk_in    => scl_i,
        i2c_data_oe   => i2c_data_oe,
        i2c_clk_oe    => open,  -- no multi master support
        rst_n         => res_n_i
    );
```

I2C_TARGET instanziiert die altera_i2cslave_to_avlmm_bridge.

```
i2cslave_to_avlmm_bridge_0 : component altera_i2cslave_to_avlmm_bridge
      generic map (
            I2C_SLAVE_ADDRESS => "1010101",
            BYTE_ADDRESSING   => 1,
            ADDRESS_STEALING  => 0,
            READ_ONLY         => 0
      )
      port map (
            clk           => clk,           --         clock.clk
            address       => address,       -- avalon_master.address
            read          => read,          --               .read
            readdata      => readdata,      --               .readdata
            readdatavalid => readdatavalid, --               .readdatavalid
            waitrequest   => waitrequest,   --               .waitrequest
            write         => write,         --               .write
            byteenable    => byteenable,    --               .byteenable
            writedata     => writedata,     --               .writedata
            rst_n         => rst_n,         --         reset.reset_n
            i2c_data_in   => i2c_data_in,   --   conduit_end.conduit_data_in
            i2c_clk_in    => i2c_clk_in,    --               .conduit_clk_in
            i2c_data_oe   => i2c_data_oe,   --               .conduit_data_oe
            i2c_clk_oe    => i2c_clk_oe     --               .conduit_clk_oe
      );
```

Für die Synthese müssen folgende Komponenten hinzugefügt werden:

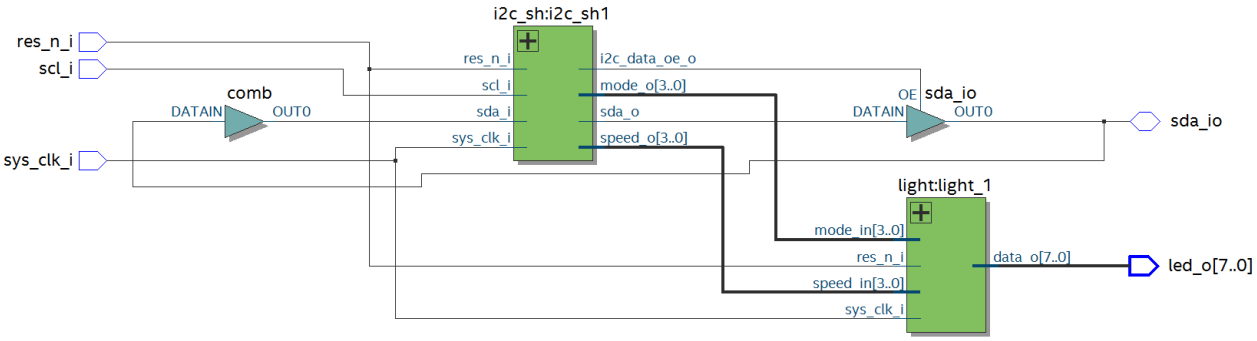


6.3.5 Light

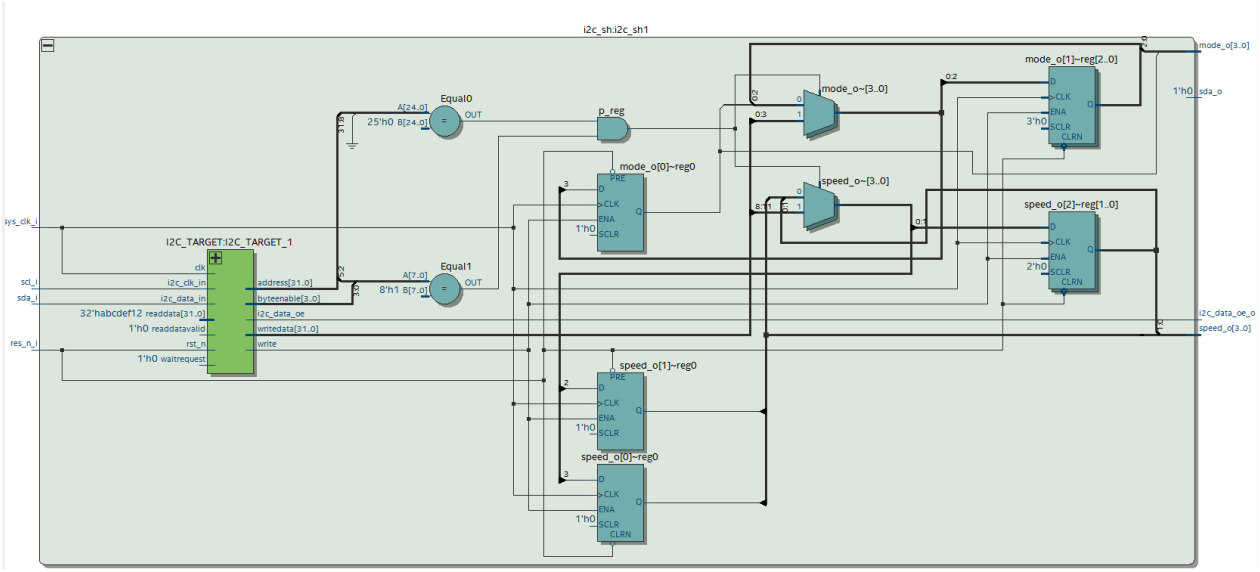
6.3.6 Top Level Entity

7 Synthese Ergebnis

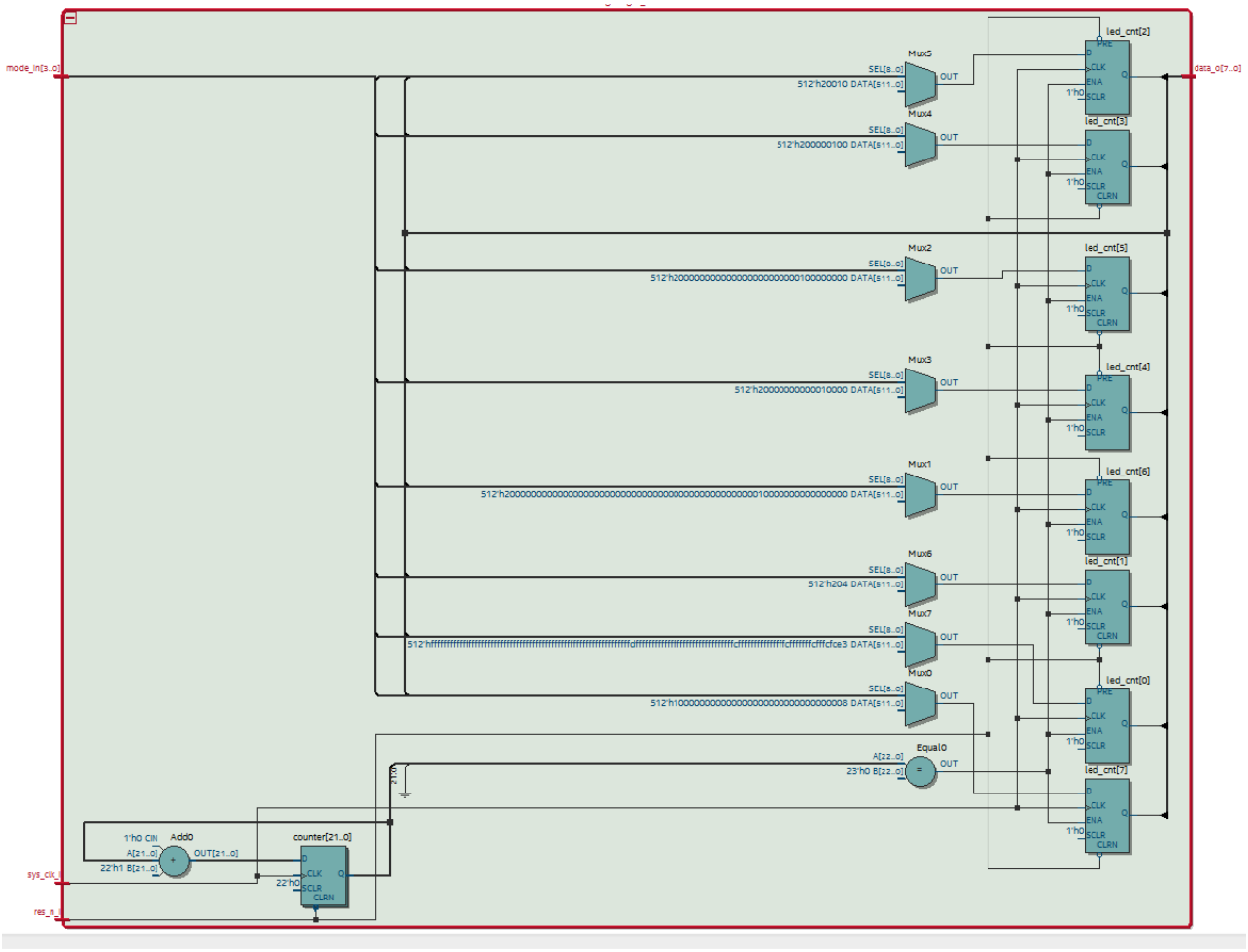
- Top Level (lauflicht)



- I2c_shell



- Light


mode_in[3..0]
data_o[7..0]
sys_clk
res_n
Mux5
Mux4
Mux2
Mux3
Mux1
Mux6
Mux7
Mux0
Equal0
Add0
counter[21..0]
led_cnt[2]
led_cnt[3]
led_cnt[5]
led_cnt[4]
led_cnt[6]
led_cnt[1]
led_cnt[0]
led_cnt[7]

7.1.1 Schreib/Lese Zyklen der I2C – Avalon - Bridge

Device Adresse(**6:0**): „101-0101“ (0x55)

- Schreiben: **Device_ID(6:0)** & R/!W = 101 0101 & 0 = „0xAA“ (101 0101 0 = 0xAA)
- Lesen: **Device_ID(6:0)** & R/!W = 101 0101 & 1 = „0xAB“ (101 0101 1 = 0xAB)

Timing

Abbildung 8 I2C Bus Zyklus

8 Testbench