Flachbandkabel-Tester für das SB-PRAM-Projekt

FoPra am Lehrstuhl für Rechnerarchitektur

Christoph Berg
Fachbereich Informatik, Universität des Saarlandes
cb@wurzelausix.cs.uni-sb.de

9. Oktober 1999

Abstract

When dealing with large parallel computers, many ribbon cables are needed. It is therefore necessary to have a test device for finding broken or short-circuited lines. Such a debugging tool is developed in this paper, using JTAG testability devices. The printed circiut board developed features three high density connectors (50, 80 and 100 pins) and an arbitrary lenght standard connector for cables up to 64 lines wide. The test result is signalled by three traffic light style LEDs.

1 Das Projekt

Bei Flachbandkabeln können verschiedene Fertigungs- und Materialfehler auftreten:

- defekte Leitungen: keine Verbindung im Kabel wegen Kabelbruch, defektem Stecker oder mangelndem Kontakt des Kabels in der Quetschverbindung,
- Kurzschlüsse: Verbindungen zwischen (meist benachbarten) Leitungen durch Materialfehler oder ungenauen Sitz des Steckers
- Fehlerhafte Montage der Stecker am Kabel, z.B. um 180° gedreht.
- unsicherer Sitz des Kabels im Stecker, brüchige Leitungen (Wackelkontakte)

1.1 Testmuster

Zur Erkennung von Fehlern werden Testmuster an das Flachbandkabel angelegt. Da keine aktiven Komponenten beteiligt sind, genügt zur Erkennung von Fehlern ein einfacher Test, bei dem an zum Testen der Leitung i an genau dieser eine Eins angelegt wird, an allen anderen Null (Bitmuster 0^i10^{99-i}). Zum Test der nächsten Leitung wird das Muster um eine Stelle rotiert („Walking-One“). Wird für jede Leitung das Muster auf der anderen Seite des Kabels wieder identisch empfangen, so kann das Kabel als korrekt angesehen werden. Nicht-statische Fehler, d.h. Wackelkontakte werden von diesem Test nicht erkannt, sind der Analyse aber schwer zugänglich.

1.2 Ausführung

Der im Rahmen dieses FoPras entwickelte Flachbandkabel-Tester für das SB-PRAM-Projekt basiert auf dem JTAG-Standard, mit dem es auf einfache Weise möglich ist, Testmuster auf eine Schaltung anzuwenden. Die Stimuli und Reaktionen des Kabels werden über mit JTAG programmierten Treibern gesteuert bzw. ausgelesen, die ihrerseits von einen programmierbaren Baustein gesteuert werden. Zur Unterstützung mehrerer Kabeltypen (High und standard density-Kabel) sind mehrere Stecker auf der Platine vorgesehen, die jeweils paarweise, d.h. für Ein- und Ausgang, bestückt werden. Das Testmuster ist vorberechnet und in einen EPROM gespeichert, genauso die erwarteten auszulesenden JTAG-Daten. Wird eine Abweichung in den relevanten, d.h. zu Eingängen gehörigen Bits festgestelt, signalisiert das Gerät den Fehler über eine LED.

2 Blockschaltbild

Das Schaltbild des Testers (Abbildung 1) besteht aus vier großen Einheiten: den Kabelein- und ausgängen links bzw. rechts, den JTAG-Treibern dazwischen und der Kontrolle, bestehend u.a. aus CPLD und EPROM.

Die symmetrisch aufgebauten **Steckerfelder** bestehen aus drei High density-Steckverbindern für 50-, 80- und 100-polige Kabel, sowie einer Standard-Pfostenleiste, die hier mit 64 Bit Breite ausgeführt ist. Letztere läßt sich durch wenige Änderungen im JTAG-Stream auch für andere Breiten nutzen.[1] Die Eingänge auf der linken Seite sind zusätzlich mit Pullups versehen, um offene Eingänge der Treiber auf einen definierten Wert zu ziehen (defekte Leitungen, obere Bits bei schmalen Kabeln). Pin 1 ist bei allen Steckern oben.

Als **Treiber** werden *SN 74ABT 8652* verwendet. Die JTAG-Pins TDI und TDO werden zu einer Kette verbunden, so daß die Treiber seriell mit den Testdaten versorgt werden können. Dabei wurde der IC für die niederwertigen Leitungen ans Ende gelegt, um im JTAG-Datenstrom Pin 1 an Anfang zu haben, da auch innerhalb eines ICs Pin 1 zuerst beschrieben wird (Abblidung 2).

Die **Kontrolle** wird mit einem *XC 9536* realisiert. Dieses CPLD bietet Platz für einen 16 Bit-Zähler zur Ansteuerung des $64kbit \times 8$-EPROMS (*27C512*) und der Auswertung des Testmusters.

3 Der JTAG-Stream

Die zum Testen nötigen Datenströme werden in einem acht Bit breiten Eprom abgelegt: die JTAG-Signale TDI und TMS werden synchron mit der globalen Clock an die Treiber gegeben; das rücklaufende TDO-Signal wird mit der gespeicherten Version aus dem Eprom verglichen. Da jeweils nur acht Bit breite Segmente des JTAG-Stroms Eingabedaten enthalten, werden die signifikanten Bits mit der `SIG`-Maske ausgewählt. Die vier Kanäle IN100, IN80, IN50 und IN64 enthalten für jede verwendete Steckerbreite den zu erwartenden JTAG-Strom von den Treibern (TDO). Es werden auch die Bits ausgewertet, die nicht zur verwendeten Steckerbreite gehören, damit das Aufstecken eines falschen Kabels erkannt wird.[2] Alle Eingänge sind auf der Platine mit Pull-downs versehen. Mit `EOS` wird das Ende des Datenstroms markiert.

Das Testmuster zur Erkennung von Fehlern wird von einem Perl-Programm erzeugt.

```
#!/usr/bin/perl -w
# stream.pl: cb 990819
use strict;
```

[1] Für die HD-Verbindungen ist man durch die verwendeten Stecksockel auf eine Breite festgelegt.
[2] Ein 80poliges Kabel wird sonst als ein korrektes 50poliges akzeptiert

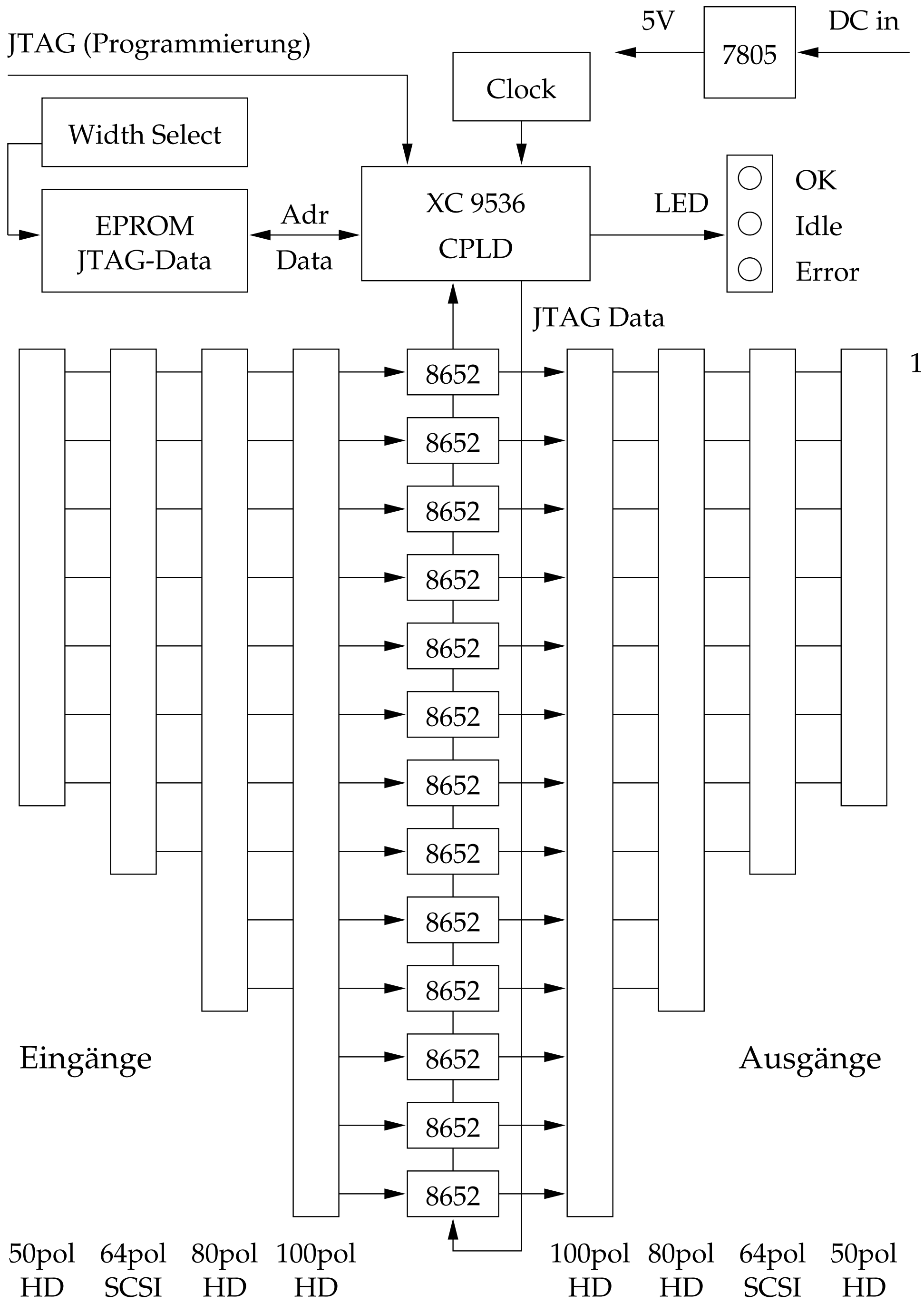

Abbildung 1: Das Blockschaltbild des Kabeltesters

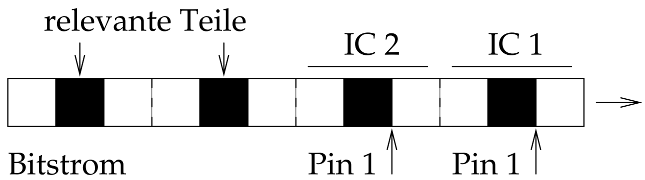

Abbildung 2: Aufbau des Datenstroms

```
# IN100 IN80 IN50 IN64 EOS TMS TDI SIG
#  0x80 0x40 0x20 0x10 0x8 0x4 0x2 0x1

my $SIG    = 0x1;  # mask for input bits
my $OUT1   = 0x2;  # drive 1 to output to TDI
my $TMSL   = 0x0;  # low transition in automaton (TMS)
my $TMSH   = 0x4;  # high...
my $EOS    = 0x8;  # End Of Stream marker
my $IN64   = 0x10; # TDO==1 expected for a 64 bit cable
my $IN50   = 0x20; #                        50
my $IN80   = 0x40; #                        80
my $IN100  = 0x80; #                       100

# calculate input bits for each cable width
sub in_bits {
    my ($pin) = @_;

    return ($pin < 64 && $IN64) | ($pin < 50 && $IN50) |
           ($pin < 80 && $IN80) | ($pin < 100 && $IN100);
}

# generates JTAG stream for a single chip (36 bits long)
# arguments: input/output pin to be tested, base pin number for this chip
sub chip {
    my ($pin_in, $pin_out, $pin_base) = @_;
    my @JTAG_C;

    # Output B
    push @JTAG_C, ($pin_out == ($pin_base + 0) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 1) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 2) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 3) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 4) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 5) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 6) && $OUT1) | 0;
    push @JTAG_C, ($pin_out == ($pin_base + 7) && $OUT1) | 0;
    # Input B (unused)
    push @JTAG_C, 0, 0, 0, 0, 0, 0, 0, 0;
    # Output A (unused)
    push @JTAG_C, 0, 0, 0, 0, 0, 0, 0, 0;
    # Input A
    push @JTAG_C, ($pin_in == ($pin_base + 0) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 1) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 2) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 3) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 4) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 5) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 6) && in_bits($pin_in)) | $SIG;
    push @JTAG_C, ($pin_in == ($pin_base + 7) && in_bits($pin_in)) | $SIG;
    # Control lines
    push @JTAG_C, 0, 0, 0, 0, $OUT1, $OUT1;

    return @JTAG_C;
}
```

```
# generates JTAG stream for a single pin (13 drivers)
sub chips {
    my $pin = $_[0]; my @JTAG_S;

    #                  Input    Output Base pin #
    push @JTAG_S, chip $pin - 1, $pin, 0;
    push @JTAG_S, chip $pin - 1, $pin, 8;
    push @JTAG_S, chip $pin - 1, $pin, 16;
    push @JTAG_S, chip $pin - 1, $pin, 24;
    push @JTAG_S, chip $pin - 1, $pin, 32;
    push @JTAG_S, chip $pin - 1, $pin, 40;
    push @JTAG_S, chip $pin - 1, $pin, 48;
    push @JTAG_S, chip $pin - 1, $pin, 56;
    push @JTAG_S, chip $pin - 1, $pin, 64;
    push @JTAG_S, chip $pin - 1, $pin, 72;
    push @JTAG_S, chip $pin - 1, $pin, 80;
    push @JTAG_S, chip $pin - 1, $pin, 88;
    push @JTAG_S, chip $pin - 1, $pin, 96;

    # raise TMS on last bit
    push @JTAG_S, (pop @JTAG_S | $TMSH);
    return @JTAG_S;
}

# reset automaton by at least five high clocks, walk to Shift-IR state
my @RESET = ($TMSH, $TMSH, $TMSH, $TMSH, $TMSH, $TMSH,
             $TMSL, $TMSH, $TMSH, $TMSL, $TMSL);

# EXTEST is all zeros, raise TMS on last bit (13 drivers, each 8 bits)
my @EXTEST;
foreach (1 .. (8 * 13 - 1)) { push @EXTEST, $TMSL; }
push @EXTEST, $TMSH;

# walk from Exit1-[DI]R to Shift-DR
my @SHIFT_DR = ( $TMSH, $TMSH, $TMSL, $TMSL );

# first flush BSCs (no input/output, since 123 > 96, strip SIG bit)
my @ZERO = map { $_ & 0xfe } chips 123;

# start of stream: Initialize, load instruction, clear
my @INIT = (@RESET, @EXTEST, @SHIFT_DR, @ZERO, @SHIFT_DR);

# pin test stream
my (@TEST, @PINTEST);
foreach my $i (0..100) {
    @TEST = chips $i;
    @PINTEST = (@PINTEST, @TEST, @SHIFT_DR);
    if ($ARGV[0]) {  # verbose flag
        print "Pin $i:\n";  # best viewed with 38*3==114 columns
        printf "%-24s%-24s%-24s%-24s%s\n",
            "Output B", "(Input B)", "(Output A)", "Input", "Control";
        foreach (@TEST) { printf "%02x ", $_; }
        print "\n\n";
```

```
    }
}

# complete test stream
my @STREAM = (@INIT, @PINTEST, $EOS);

# write output
open F, ">stream.bin" or die "Could not write stream.bin: $!";
foreach (@STREAM) { printf F "%c", $_; }
close F;

if ($ARGV[0]) {  # verbose flag
    foreach (@STREAM) { printf "%02x ", $_; print "\n" if $_ == 5; }
    print "\n\n";
}

print "stream.bin written.\n";
```

In der Funktion `chip` wird der 38 Bit lange Datenstrom für einen einzelnen JTAG-Treiber erzeugt. Die verwendeten Signale sind Output B und Input A, die jeweils andere Richtung wird nicht benutzt.

Die Funktion `chips` ruft `chip` 13mal auf, um den gesamten Teststrom für einen Pin zu erhalten. Dabei wird jeweils als Input der Pin des letzten Tests erwartet, da die Ergebnisse erst einen Druchgang später herausgeshiftet werden. Die Kombination push/pop an Ende setzt beim letzten Bit TMS auf high, um dem JTAG-Automaten das Ende des Testmusters zu signalisieren.

Als Präambel des gesamten Teststroms wird der Automat den Grundzustand versetzt und in Shift-DR gebracht (`RESET`), die Instruktion `EXTEST` geladen (13 * 8 Nullen, am Ende wieder der TMS-high-Übergang), die BSC-Register geleert (`ZERO`) und wieder in Shift-DR übergegangen. Für `ZERO` wird einfach der nicht existente Pin 123 getestet, so daß im Teststrom keine Pins gesetzt und keine relevanten erwartet werden, außerdem werden die Maskenbits gelöscht.

Im Anschluß wird in einer Schleife jeder der Pins 0 bis 99 getestet (`PINTEST`). Der Pin „100“ wird getestet, um die Ausgabe des Test für Pin 99 zu erhalten, die dann herausgeshiftet wird. Der gesamte Datenstrom wird am Ende auf Platte geschrieben.

Anhang: Auszug aus der BSDL-Datei für *SN 74 ABT 8652*:

```
------------------------------------------------------------------------------
--  TI SN74ABT8652 SCOPE(TM) OCTAL BUS TRANSCEIVERS AND REGISTERS           --
------------------------------------------------------------------------------
[...]

entity sn74abt8652 is
    generic (PHYSICAL_PIN_MAP : string := "UNDEFINED");

    port (CLKAB:in bit; CLKBA:in bit; SAB:in bit; SBA:in bit;
          OEAB:in bit; OEBA_NEG:in bit;
          A:inout bit_vector(1 to 8); B:inout bit_vector(1 to 8);
          GND, VCC:linkage bit; TDO:out bit; TDI, TMS, TCK:in bit);

    use STD_1149_1_1990.all; -- Get standard attributes and definitions

    attribute PIN_MAP of sn74abt8652 : entity is PHYSICAL_PIN_MAP;
```

```
[...]
    constant DW : PIN_MAP_STRING := "CLKAB:1, CLKBA:28," &
                  "SAB:2, SBA:27, OEAB:3," &
                  "OEBA_NEG:26, GND:7, VCC:21, TDO:13, TMS:14," &
                  "TCK:15, TDI:16, A:(4,5,6,8,9,10,11,12)," &
                  "B:(25,24,23,22,20,19,18,17)";
[...]

    attribute TAP_SCAN_IN    of TDI : signal is true;
    attribute TAP_SCAN_MODE  of TMS : signal is true;
    attribute TAP_SCAN_OUT   of TDO : signal is true;
    attribute TAP_SCAN_CLOCK of TCK : signal is (50.0e6, BOTH);

    attribute INSTRUCTION_LENGTH of sn74abt8652 : entity is 8;
    attribute INSTRUCTION_OPCODE of sn74abt8652 : entity is
            "BYPASS (11111111, 10000001), " &
            "EXTEST (00000000), " &
            "SAMPLE (10000010), " & [...]
            "SCANCT (00001111)  " ;  -- BCR Scan Test Mode
[...]

    attribute BOUNDARY_CELLS    of sn74abt8652 : entity is "BC_1";
    attribute BOUNDARY_LENGTH   of sn74abt8652 : entity is 38;
    attribute BOUNDARY_REGISTER of sn74abt8652 : entity is

    "0  (BC_1, B(1)    ,output3 , X, 37, 0, Z),"  &
    "1  (BC_1, B(2)    ,output3 , X, 37, 0, Z),"  &
    "2  (BC_1, B(3)    ,output3 , X, 37, 0, Z),"  &
    "3  (BC_1, B(4)    ,output3 , X, 37, 0, Z),"  &
    "4  (BC_1, B(5)    ,output3 , X, 37, 0, Z),"  &
    "5  (BC_1, B(6)    ,output3 , X, 37, 0, Z),"  &
    "6  (BC_1, B(7)    ,output3 , X, 37, 0, Z),"  &
    "7  (BC_1, B(8)    ,output3 , X, 37, 0, Z),"  &
    "8  (BC_1, B(1)    ,input   , X),"  &
    "9  (BC_1, B(2)    ,input   , X),"  &
    "10 (BC_1, B(3)    ,input   , X),"  &
    "11 (BC_1, B(4)    ,input   , X),"  &
    "12 (BC_1, B(5)    ,input   , X),"  &
    "13 (BC_1, B(6)    ,input   , X),"  &
    "14 (BC_1, B(7)    ,input   , X),"  &
    "15 (BC_1, B(8)    ,input   , X),"  &
    "16 (BC_1, A(1)    ,output3 , X, 36, 1, Z),"  &
    "17 (BC_1, A(2)    ,output3 , X, 36, 1, Z),"  &
    "18 (BC_1, A(3)    ,output3 , X, 36, 1, Z),"  &
    "19 (BC_1, A(4)    ,output3 , X, 36, 1, Z),"  &
    "20 (BC_1, A(5)    ,output3 , X, 36, 1, Z),"  &
    "21 (BC_1, A(6)    ,output3 , X, 36, 1, Z),"  &
    "22 (BC_1, A(7)    ,output3 , X, 36, 1, Z),"  &
    "23 (BC_1, A(8)    ,output3 , X, 36, 1, Z),"  &
    "24 (BC_1, A(1)    ,input   , X),"  &
    "25 (BC_1, A(2)    ,input   , X),"  &
    "26 (BC_1, A(3)    ,input   , X),"  &
    "27 (BC_1, A(4)    ,input   , X),"  &
```

```
    "28 (BC_1, A(5)    ,input   , X)," &
    "29 (BC_1, A(6)    ,input   , X)," &
    "30 (BC_1, A(7)    ,input   , X)," &
    "31 (BC_1, A(8)    ,input   , X)," &
    "32 (BC_1, SBA     ,input   , X)," &
    "33 (BC_1, SAB     ,input   , X)," &
    "34 (BC_1, CLKBA   ,input   , X)," &
    "35 (BC_1, CLKAB   ,input   , X)," &
    "36 (BC_1, OEBA_NEG,input   , X)," &
    "36 (BC_1, *       ,controlr, 1)," &
    "37 (BC_1, OEAB    ,input   , X)," &
    "37 (BC_1, *       ,controlr, 0) " ;

end sn74abt8652;
```

4 Schaltbilder

Das *Rootsheet* verbindet die einzelnen Unterfunktionseinheiten.

Neben den Steckern zum Anschluß der zu testenden Kabel enthält *Pinin* die Pulldown-Widerstände.

Zentrales Element der *Kontrolle* ist das CPLD *XC 9536*. Neben Clock (1 MHz), dem Test Access Port (TAP) zur Programmierung des CPLD, der 5V-Versorgung mit Verpolungsschutz und den Leuchtdioden zur Signalisierung des Testergebnisses findet sich hier das Eprom, in dem die Testdaten abgelegt sind. Die Signale TDI, TMS, SIG und EOS werden direkt an das CPLD geleitet, das Signal IN wird mittels Jumper aus einem der vier Ausgänge IN64, IN50, IN80 und IN100 ausgewählt. Die JTAG-Signale zu den Test-Treibern werden durch das CPLD durchgeschleift, um eine konstante Verzögerung zu erhalten.

Die *JTAG-Treiber* werden seriell mit den Testdaten über die TDI-Leitung versorgt. Die TMS- und TCK-Leitungen werden direkt an alle Treiber gelegt. Die Steuerleitungen (OEAB, /OEBA etc.) werden alle auf logisch low gesetzt, um beim Anschalten keinen Kurzschluß verursachen zu können. Auch die aktive Terminierung der beiden Clock-Leitungen findet sich hier.

5 Platine

Die *Platine* enthält nur wenige Bedienelemente: die Stecker dienem dem Anschluß der Flachbandkabel, auf der linken Seite ist am kleinen zweipoligen Stecker die Versorgungsspannung (zwischen 8 und 15 V DC) anzulegen.

Auf der rechten Seite wird die Breite des zu testenden Kabels mit einem Jumper eingestellt, dabei muß genau ein Jumper gesetzt werden. Alternativ ist auch die Benutzung eines entsprechend konfektionierten Kabels mit Schalter möglich.

Die Leuchtdioden signalisieren den Zustand des Kabels wie folgt: Grün bedeutet, daß das Kabel den Test für die eingestellte Breite bestanden hat, bei Rot wurden Fehler gefunden, d.h. mindestens eine Leitung hat im Test nicht so wie erwartet reagiert. Die gelbe LED leuchtet, wenn überhaupt keine Verbindung gefunden wurde; im Normalfall, wenn kein Kabel eingesteckt wurde.

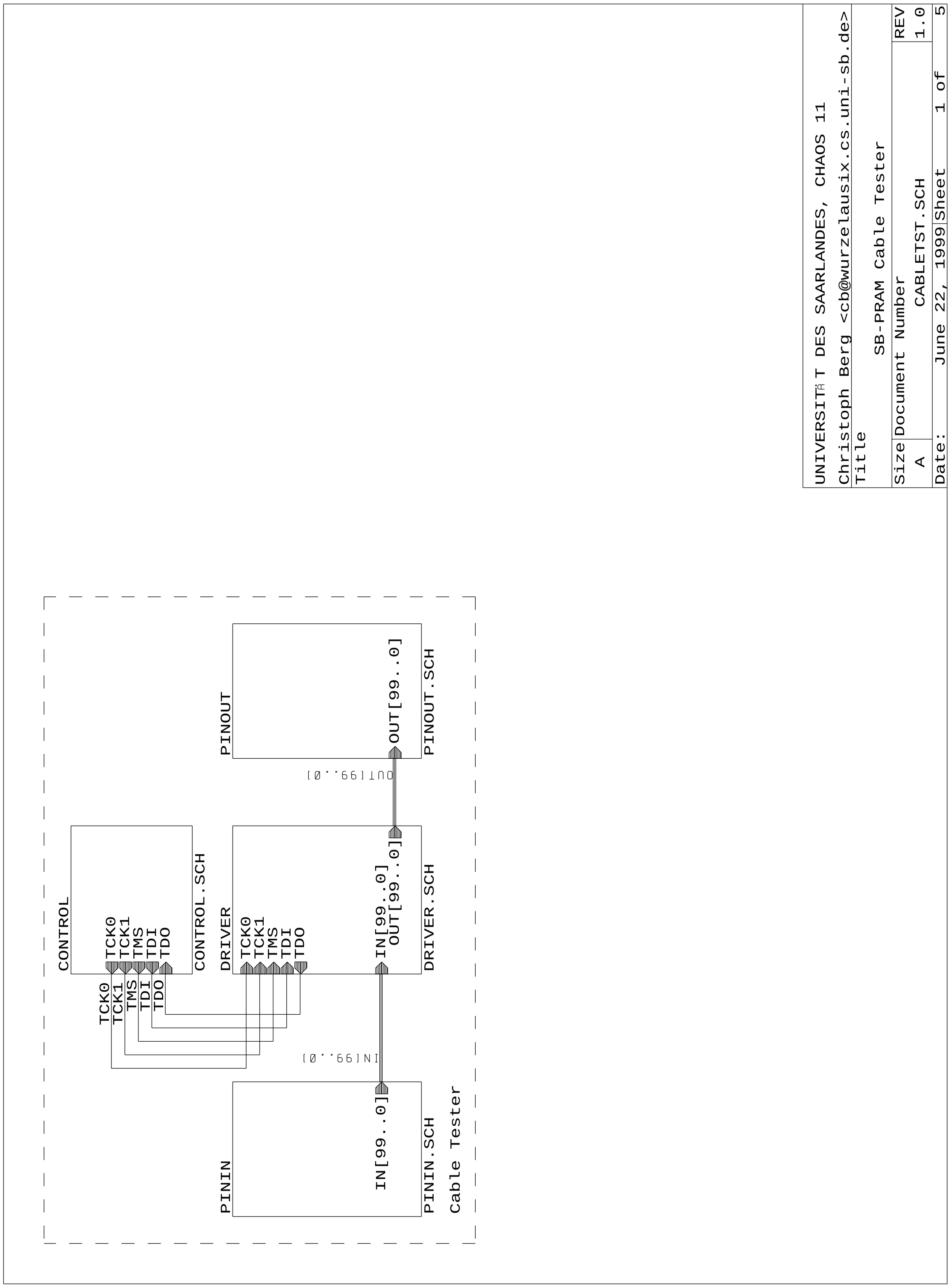

Abbildung 3: Rootsheet

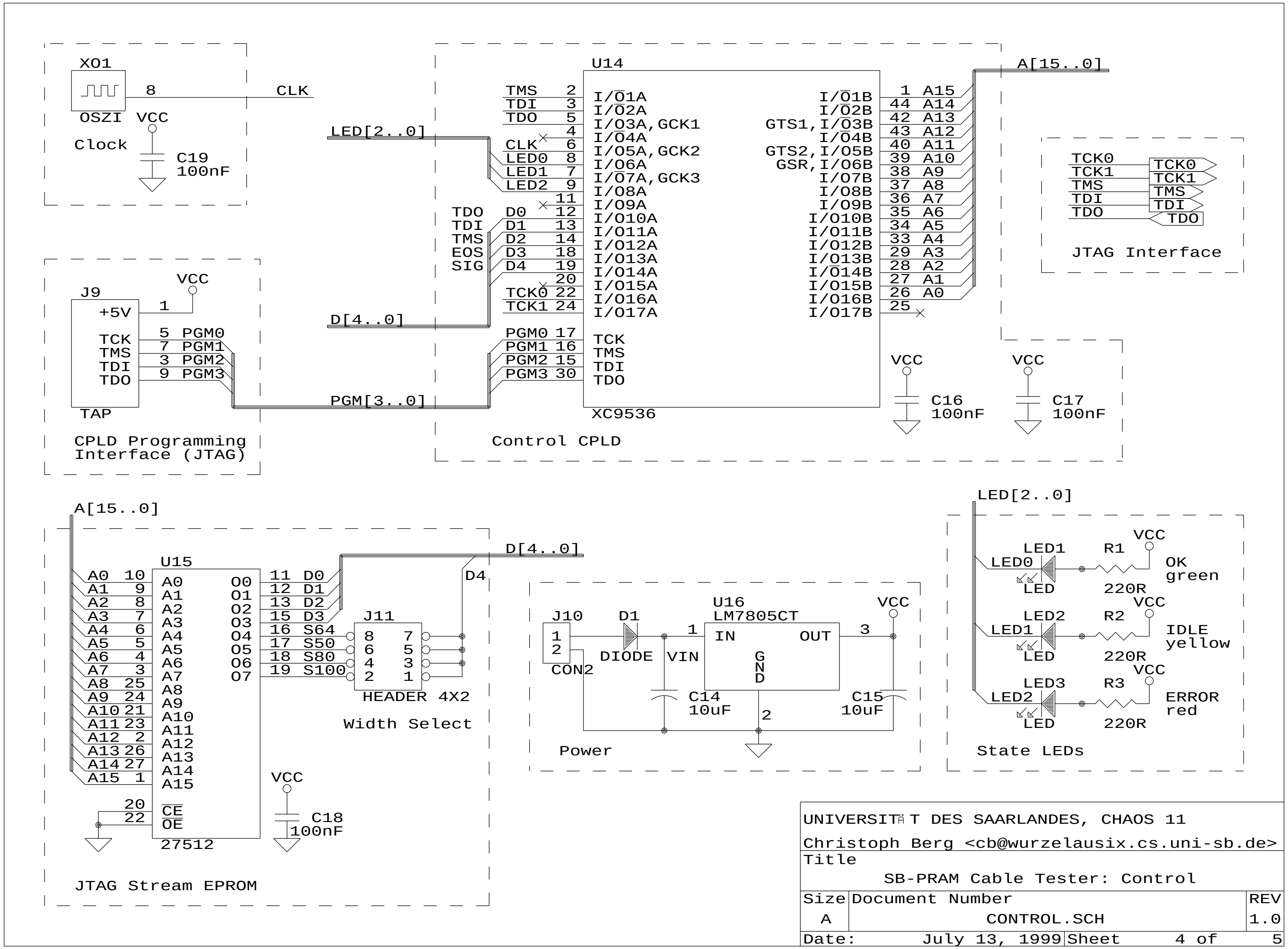

Abbildung 4: Die Kontrolle

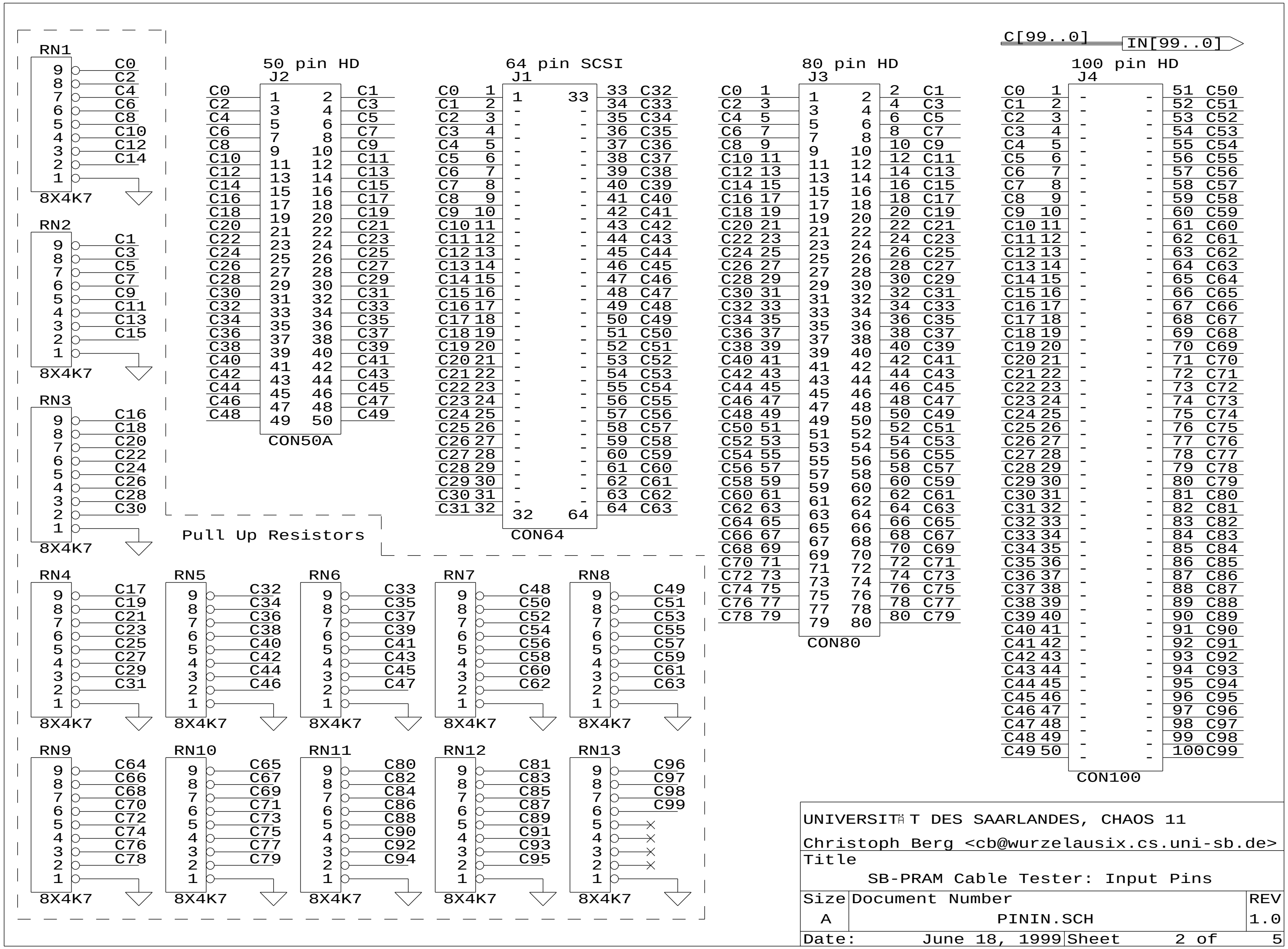

Abbildung 5: Die Eingangsstecker

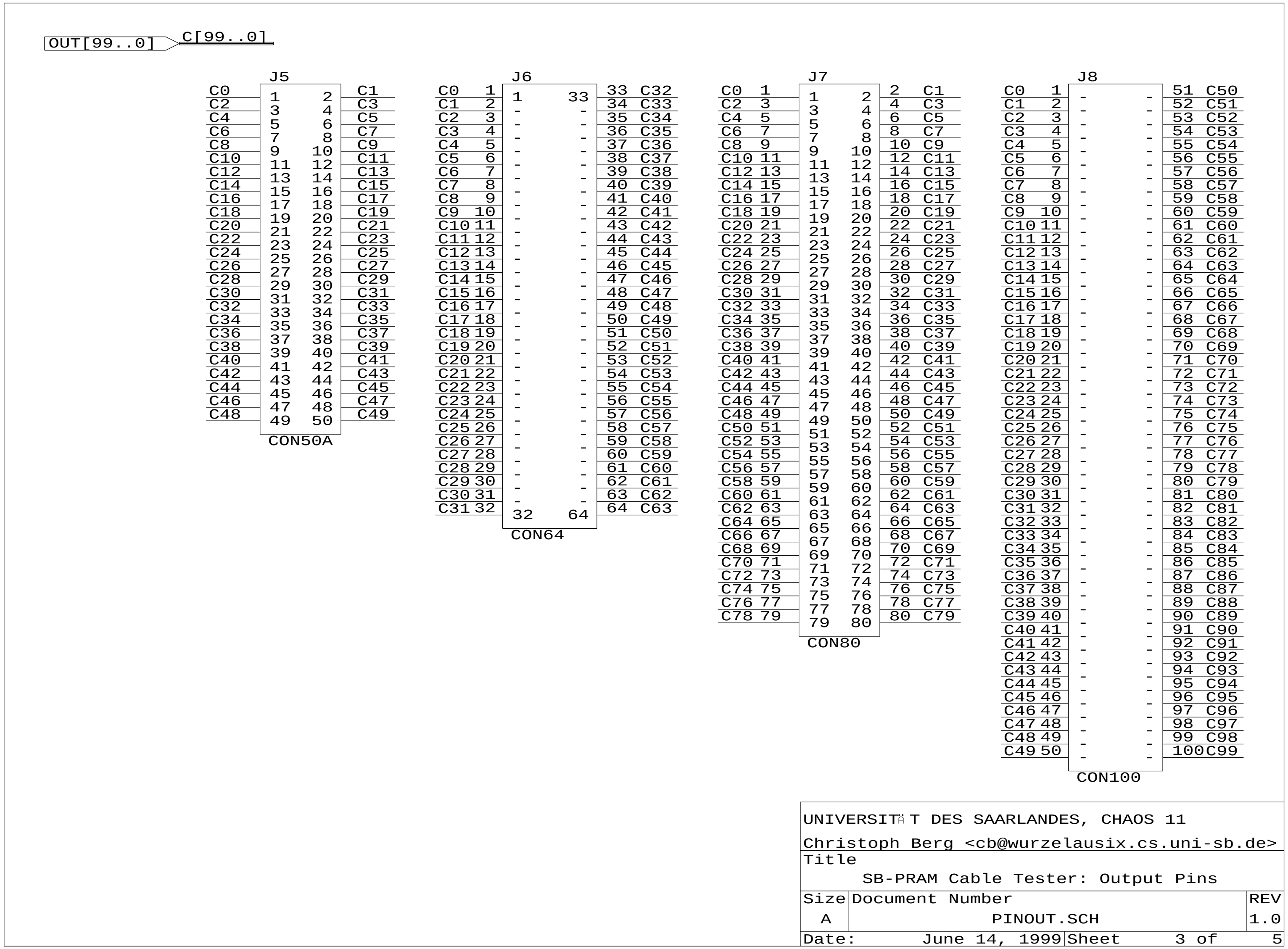

Abbildung 6: Die Ausgangsstecker

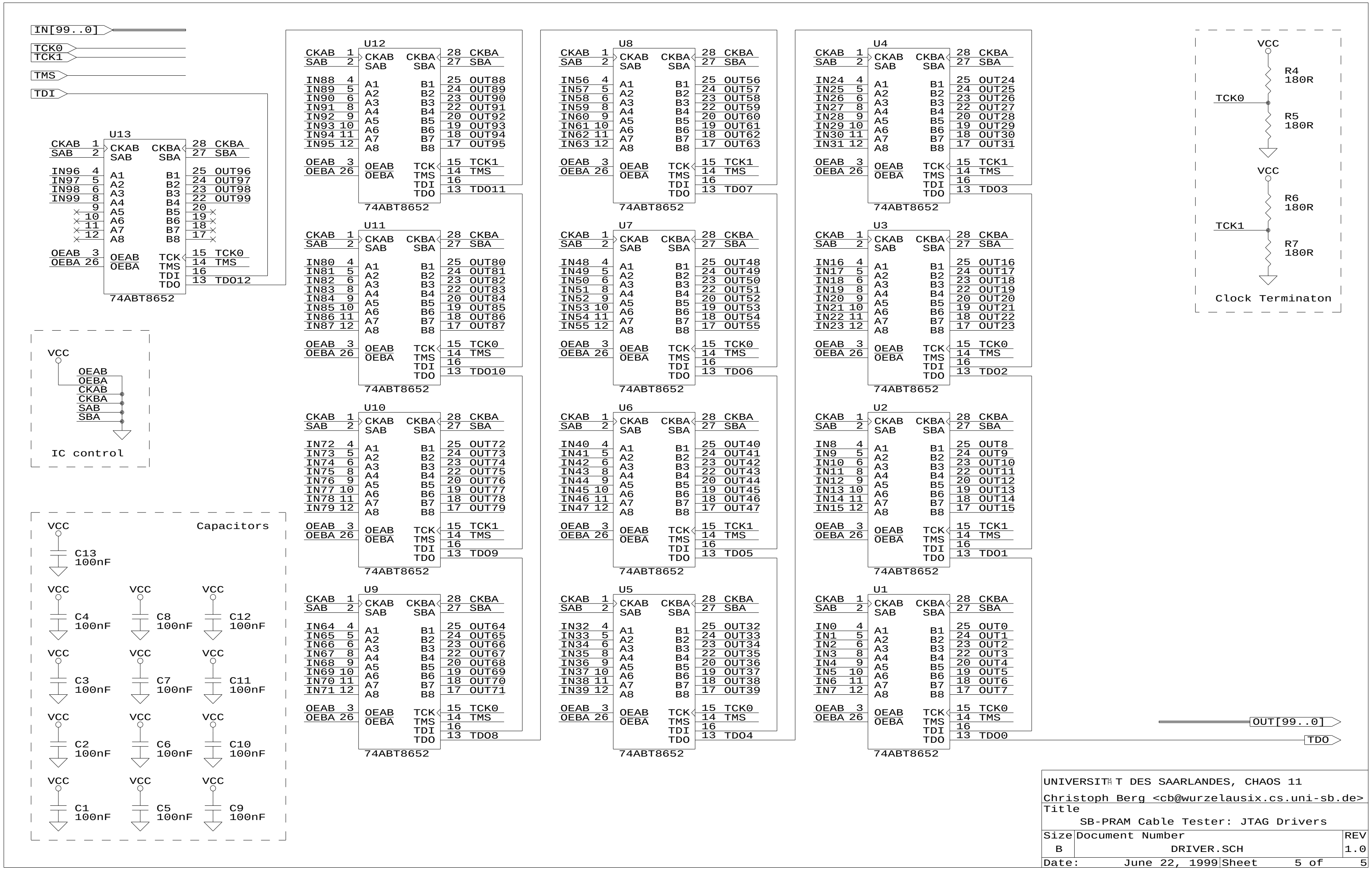

Abbildung 7: Die JTAG-Treiber

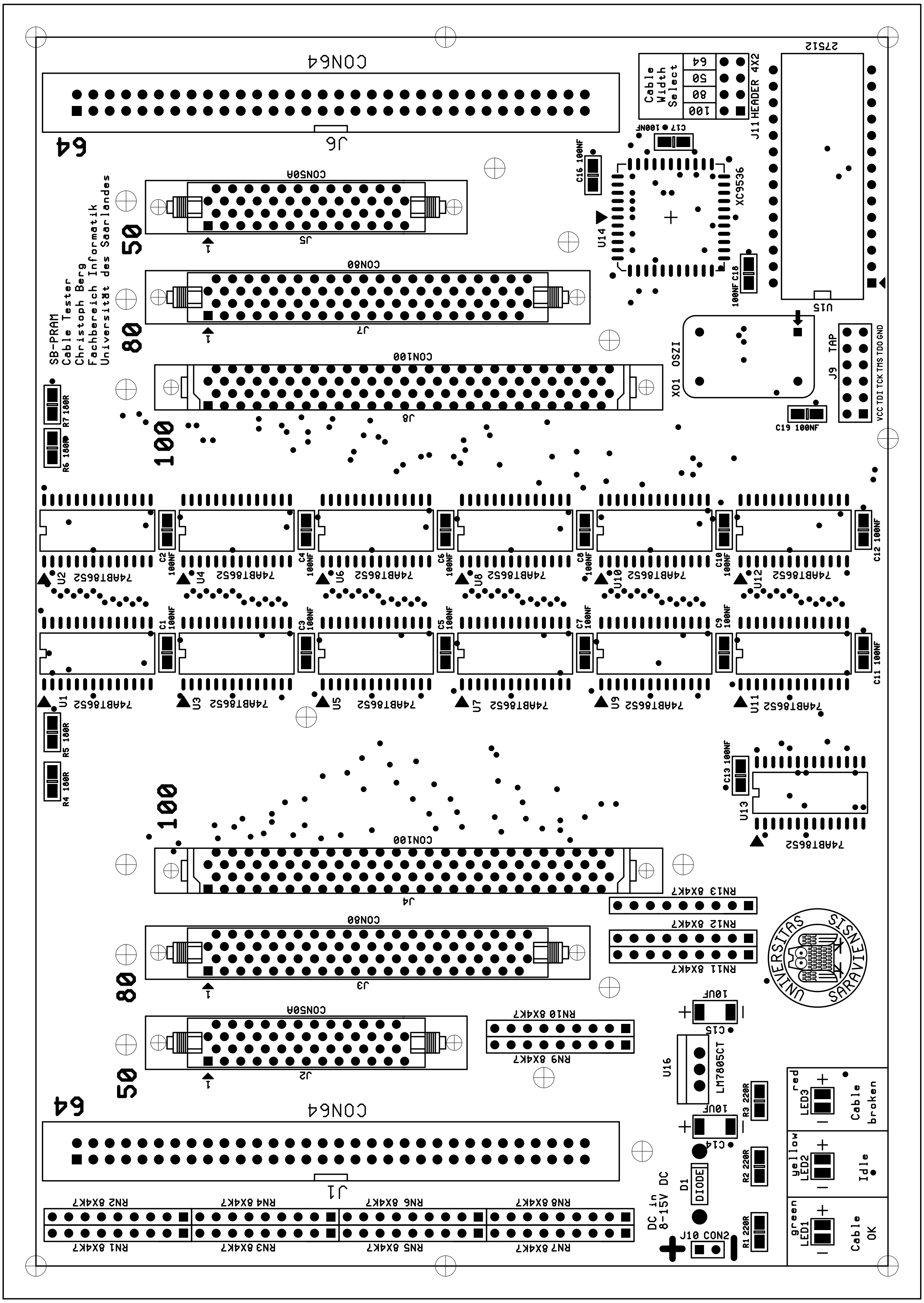

Abbildung 8: Silk Screen, Pads, Vias, Holes

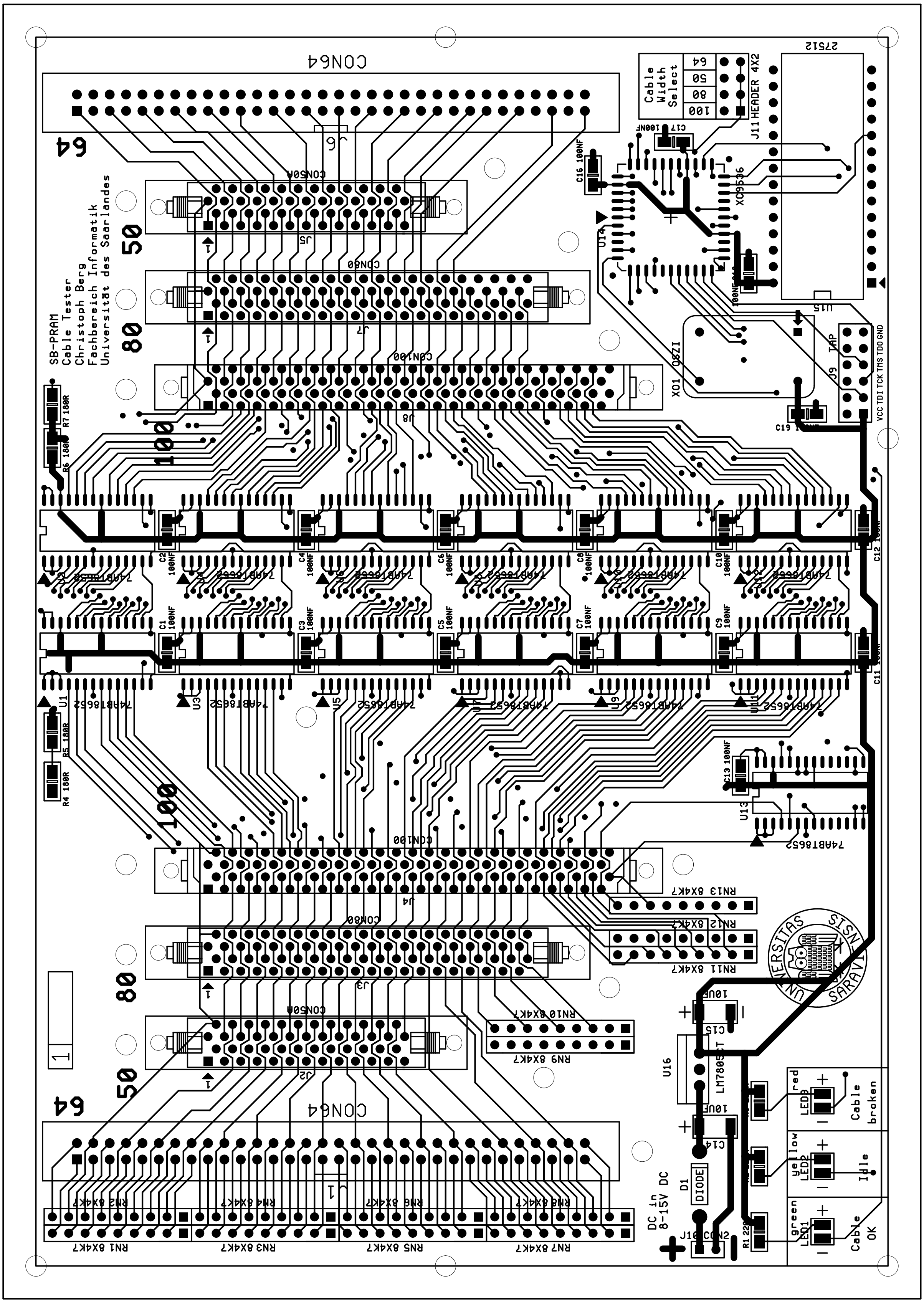

Abbildung 9: Top Copper Layer

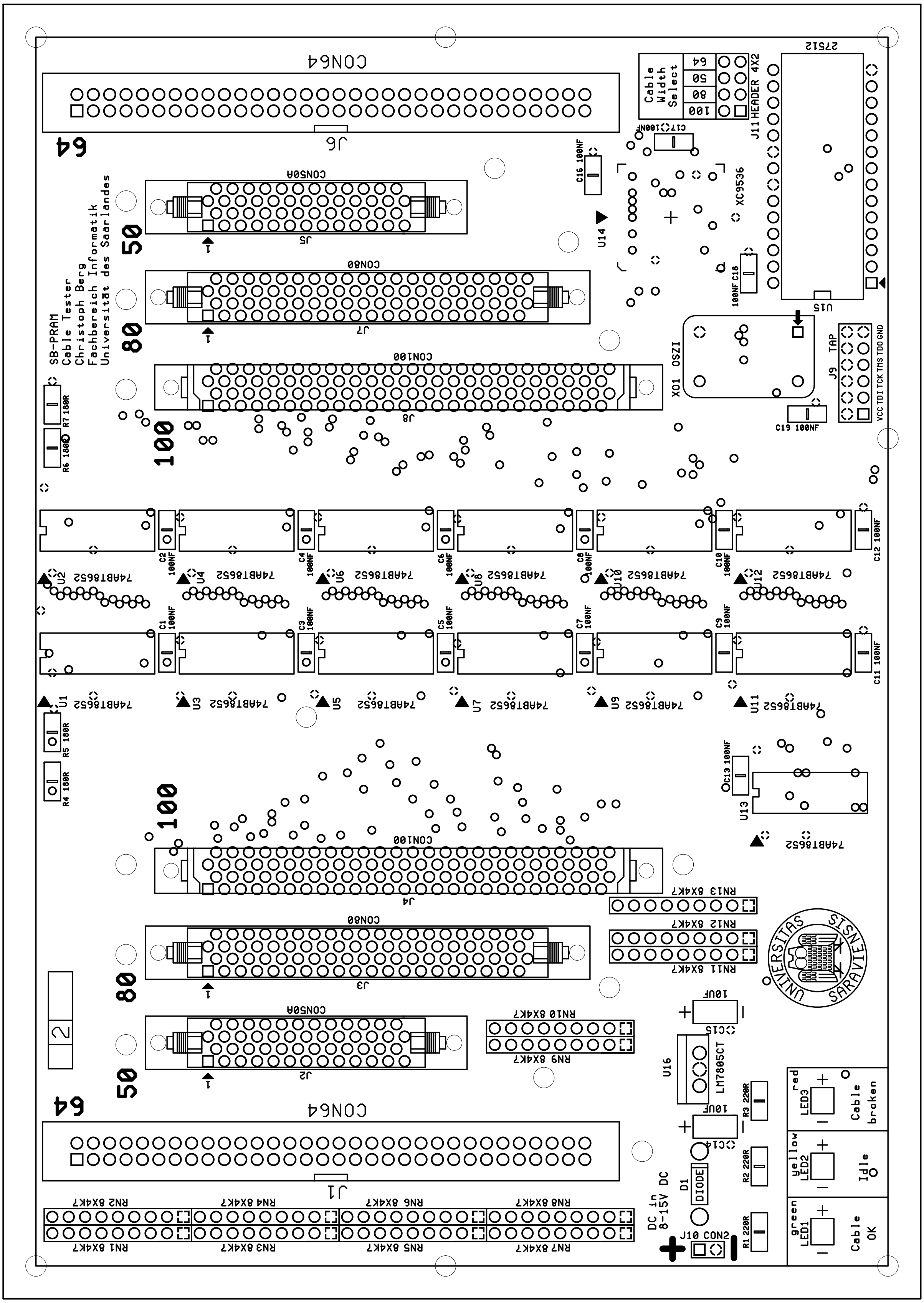

Abbildung 10: Ground Layer

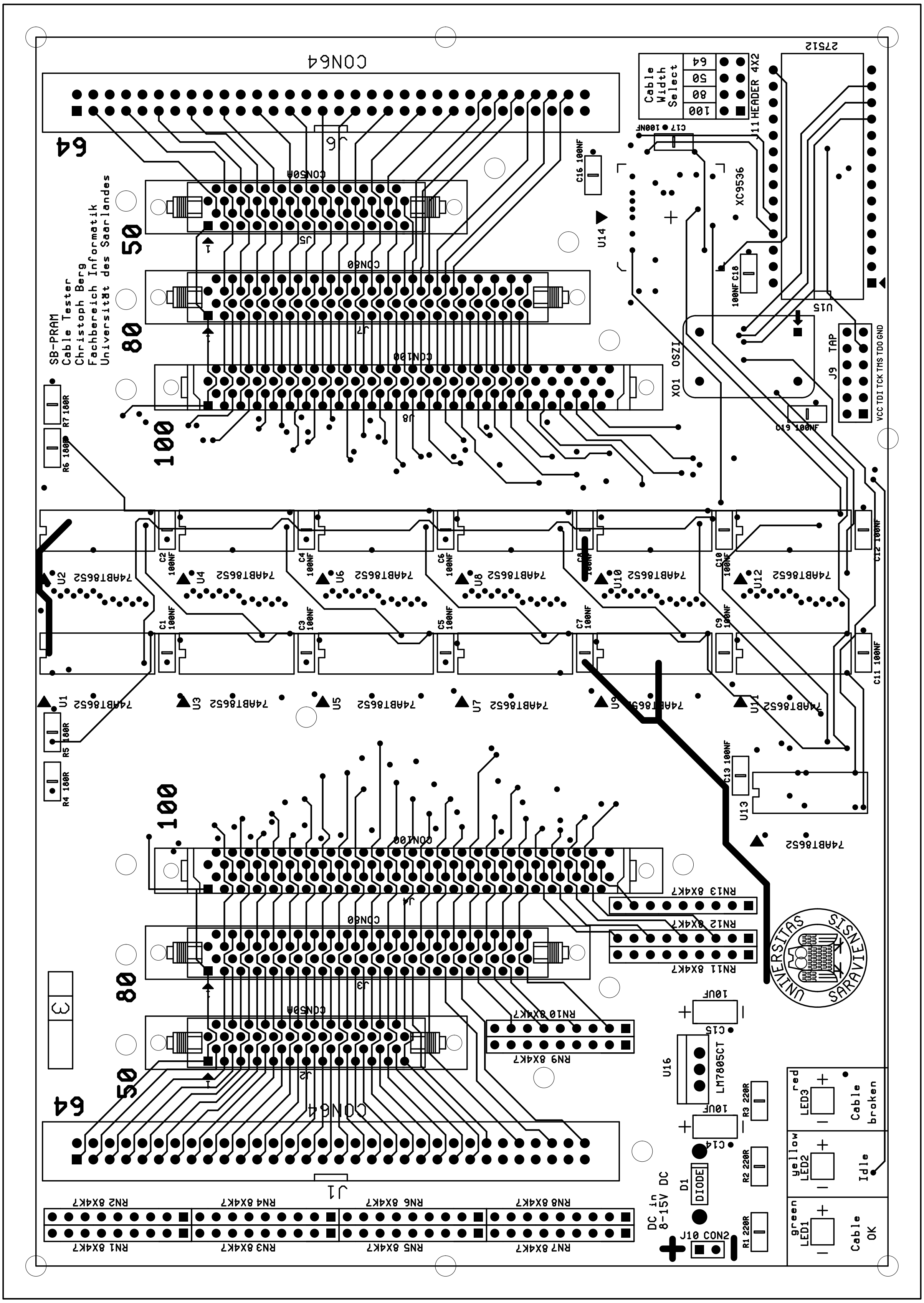

Abbildung 11: Intermediate Copper Layer

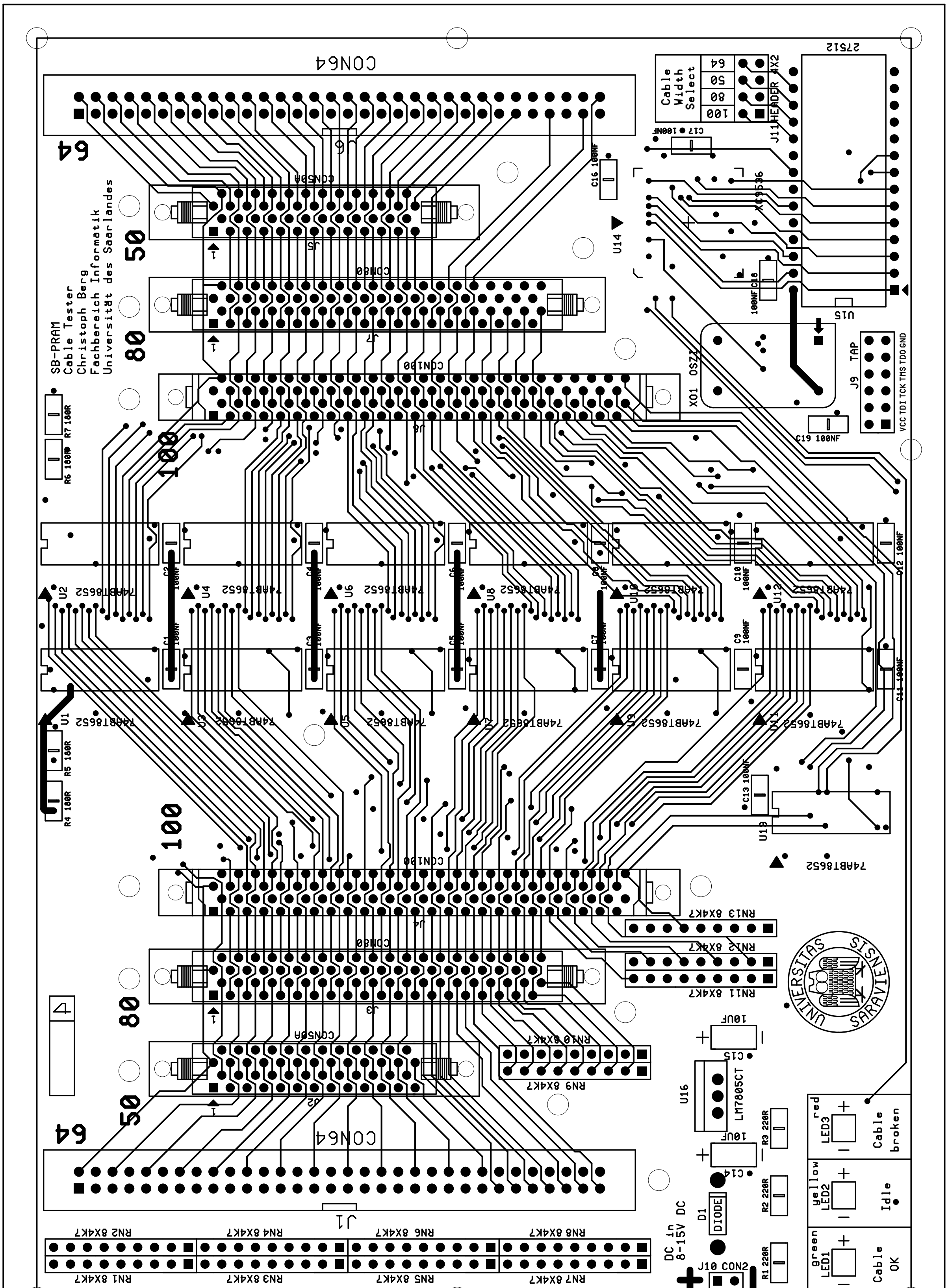

Abbildung 12: Bottom Copper Layer

6 CPLD

Als Programmiersprache wurde ABEL gewählt, da in ihr leicht der für die Ansteuerung des EPROM benötigte 16 Bit-Zähler implementiert werden kann.

Nach der Pindeklaration (der Suffix „e" bei `TDOe`, `TDIe` und `TMSe` bedeutet, daß die Signale aus dem Eprom kommen) folgen zunächst die Gleichungen für den Zähler, der mit `EOS` zurückgesetzt wird. `TDI` und `TMS` werden synchron zur globalen Clock durch das CPLD durchgereicht.

Die Bits `FOUND` und `ERROR` werden über einen Testdurchgang akkumuliert und zeigen an, ob bisher ein Kabel bzw. ein Fehler gefunden wurde. Mit `EOS` werden die beiden Bits gelöscht, geclockt werden sie nur bei relevanten Eingabedaten.

Die Auswertung der beiden Bits geschieht bei `EOS`. Die gelbe LED leuchtet, wenn kein Kabel gefunden wurde. Ansonsten ist das Kabel in Ordnung (grüne LED), wenn kein Fehler gefunden wurde; wurde ein Fehler erkannt, so wird die rote LED aktiviert.

```
module Cablepld
Title 'Cablepld'

Declarations

" global clock
CLK PIN 6;

" JTAG driver interface:
" PLD outputs
" TMS, TDI, TCK0, TCK1 PIN 2, 3, 22, 24 istype 'com';
TCK0, TCK1 PIN  22, 24 istype 'com';
TMS, TDI PIN 2, 3 istype 'reg';
" PLD input
TDO PIN 5;

" state LEDs: LED 0 green, LED 1 yellow, LED 2 red
LEDok, LEDidle, LEDerror PIN 8, 7, 9 istype 'reg';

" the address counter
A15..A0 PIN 1, 44, 42, 43, 40..33, 29..26 istype 'reg';
A = [A15..A0];

" Eprom inputs
SIG, TDIe, TMSe, EOSe, TDOe PIN 12..14, 18, 19;

" internal state bits
FOUND, ERROR, EOS PIN 4, 11, 20 istype 'reg';

Equations

" increment address until end of stream
A := A + 1;
A.CLR = EOS;
A.CLK = CLK;

" JTAG interface equations
TCK0    = CLK;
```

```
TCK1    = CLK;
TDI    := TDIe;
TMS    := TMSe # (ERROR & FOUND);
EOS    := EOSe;
TDI.CLK = !CLK;
TMS.CLK = !CLK;
EOS.CLK = !CLK;

" accumulate bits
FOUND    := FOUND # TDO;
FOUND.CLR = EOS;
FOUND.CE  = SIG # EOS;
FOUND.CLK = CLK;
ERROR    := ERROR # (TDO $ TDOe);
ERROR.CLR = EOS;
ERROR.CE  = SIG # EOS;
ERROR.CLK = CLK;

" display test result
!LEDerror :=  FOUND &  ERROR;
!LEDidle  := !FOUND;
!LEDok    :=  FOUND & !ERROR;
LEDerror.CE = EOS;
LEDidle.CE  = EOS;
LEDok.CE    = EOS;
LEDerror.CLK = CLK;
LEDidle.CLK  = CLK;
LEDok.CLK    = CLK;

end Cablepld
```

Literatur

[1] IEEE Std 1149.1 (JTAG) Testability Primer, Texas Instruments 1996

[2] Xilinx: The Programmable Logic Data Book, Xilinx 1998

[3] Advanced BiCMOS ABT Bus Interface Logic, Texas Instruments 1994

Zusammenfassung

Technische Daten

Größe:	201×143 mm
Stromaufnahme:	max. 500 mA (typ. 450 mA)

Bedienung

- Links Strom anschließen (Steckernetzteil, auf richtige Polung achten!)
- Mit dem Jumper rechts die gewünschte Kabelbreite einstellen (100, 80, 50 oder 64)
- Zu testendes Flachbandkabel einstecken
- An den LEDs das Testergebnis ablesen:
  Grün: das Kabel ist in Ordnung
  Gelb: es ist kein Kabel eingesteckt
  Rot: das Kabel ist fehlerhaft, bitte entsorgen/reparieren.

Bedienungsanleitung für den Kabelproduzenten

Universität
des
Saarlandes
Informatik 0681/302-4490

Bedienungsanleitung:

* Steckernetzteil einstecken
* Am Schalter gewünschte Kabelbreite einstellen: 80polig oder 50polig (breite oder schmale blaue Buchsen)
* Testen:
  - Kabel mit beiden Enden einstecken
  - An den Leuchtdioden das Testergebnis ablesen:
    Grün: Kabel OK
    Gelb: kein K. eingesteckt
    Rot: Kabel defekt
  - nächstes Kabel einstecken
  - nur ein Kabel gleichzeitig testen!

Abbildung 13: Aufkleber