



# BOOTRAM68k

Konfigurierbare Speicherbaugruppe mit  
Speichervollausbau, BankBoot-Schaltung und  
Grundprogramm-EPROM.  
für den NDR-Klein-Computer (mit M68000/8 CPU).

Stand: November 2007

Copyright © by Gerald Ebert

**Wichtiger Hinweis:**

Die in dieser Anleitung wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage oder Lizenzrechte Dritter mitgeteilt. Sie sind ausschließlich für private Zwecke und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden. \*)

Alle Schaltungen und technische Angaben in dieser Anleitung wurden vom Autor sorgfältig erarbeitet bzw. zusammengestellt. Trotzdem sind Fehler nicht auszuschließen. Daher kann der Autor weder eine Garantie noch die juristische Verantwortung oder irgend eine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Die Rechte an Firmennamen, Logos und Warenzeichen, die in dieser Anleitung genannt werden, liegen bei den jeweiligen Inhabern.

Vielen Dank an Ulrich Radig für seinen Xilinx-CPLD-Programmer.

\*) Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenz- oder Rechteinhabers einzuholen.

**Inhalt:**

<b>Vorwort</b>	<b>4</b>
<b>1 Kurzbeschreibung der Funktion</b>	<b>4</b>
<b>2 Technische Daten</b>	<b>4</b>
<b>3 Prinzipbeschreibung</b>	<b>5</b>
3.1 Blockschaltbild BOOTRAM68k	5
3.2 Beschreibung des Blockschaltbildes	6
3.3 Die Baugruppe als RAM-Erweiterung	6
3.4 Die BankBoot - Option	7
3.5 Das EPROM für das Grundprogramm	8
3.6 Betrieb mit CPU68000	8
3.7 Die Aktivitäts-/Fehleranzeige	9
<b>4 Aufbauanleitung</b>	<b>9</b>
4.1 Umgang mit ICs	9
4.2 Stückliste	10
4.3 Aufbau Schritt für Schritt	10
4.4 Programmierung des Xilinx-CPLD	12
4.5 Belegung von Steckleisten, Buchsen und Jumpers	12
4.6 Alles Richtig gemacht ?	14
<b>5 Neuerungen und Fehlerkorrekturen</b>	<b>14</b>
<b>6 Literaturhinweise und –nachweise</b>	<b>15</b>
<b>Schaltplan BOOTRAM68k</b>	<b>16</b>
<b>Bestückungsplan</b>	<b>17</b>
<b>Layout Bestückungsseite mit Aufdruck</b>	<b>18</b>
<b>Layout Bestückungsseite</b>	<b>19</b>
<b>Layout Lötseite</b>	<b>20</b>
<b>Schaltplan Xilinx Programmer</b>	<b>21</b>
<b>Bilder von meinem Xilinx Programmer</b>	<b>22</b>
<b>Das BankBoot Programm</b>	<b>23</b>
<b>Das VHDL-Skript für den CPLD</b>	<b>29</b>

## Vorwort

Mitte der 80er Jahre war Speicher sehr teuer; ca. 50 DM für ein 8 kB SRAM (Stand: 12/84). Heutzutage sind SRAMs viiiiel billiger. Der Speichervollausbau mit 960 kB hätte damals mit diesen Bausteinen etwa 6.000 DM gekostet. Jetzt; mit SRAMs hoher Speicherkapazität, nur ca. 7 €. Also warum nicht eine Vollausbau-Speicherbaugruppe mit allen bekannten Optionen bauen ?

Das Vorhaben gestaltete sich aber schwieriger als gedacht. Der erste Versuch, ein Design nur mit TTL-Bausteinen, scheiterte daran, daß nicht alle Bausteine auf eine Eurokarte platzierbar waren. OK ... dafür gibt es aber heutzutage eine Lösung: CPLD.

Also habe ich mir Xilinx ISE-Webpack und ein paar XC9536 besorgt, einen Programmierer gebaut und los geht's ... dachte ich. Denkste, jetzt fängt das Elend erst an. Die Teile haben nämlich so ihr Eigenleben. Das größte Problem sind Taktsignale für FlipFlops bzw. Latches die mehrere Quellen haben wie z.B. Schreiben vom NKC-Bus in ein Register. Nachdem ich megabyteweites Designregeln und Beispiele studiert habe, funktioniert das Design bei mir seit ein paar Wochen ohne Fehler. Jetzt hat mein NKC ordentlich Speicher zur Verfügung und ein komfortables Bootprogramm gleich dazu.

Eine Platine werde ich aus Kostengründen erst mal nicht produzieren lassen. Meine gefädelten Varianten tun's auch.

Da nichts auf dieser Welt perfekt sein kann, werden sich auch hier Fehler eingeschlichen haben. Wenn Sie einen entdecken, so teilen Sie mir dies mit einer möglichst genauen Beschreibung im NKC-Forum (siehe Literatur) mit.

Danke.

## 1 Kurzbeschreibung der Funktion

Der zentrale Teil der Baugruppe sind die SRAMs die den gesamten Adressierungsbereich von 960 kB abdecken (M68008). Eine erweiterte BankBoot-Logik ersetzt außerdem noch die Baugruppe BANKBOOT. Zusätzlich ist auch noch ein Sockel vorhanden, der das Grundprogramm in einem EPROM aufnehmen kann. Dieses wird dann automatisch vom BankBoot-Programm in das RAM kopiert und gestartet.

Für den Betrieb des NKC ist pro Bushälfte nur eine einzige Speicherbaugruppe nötig.

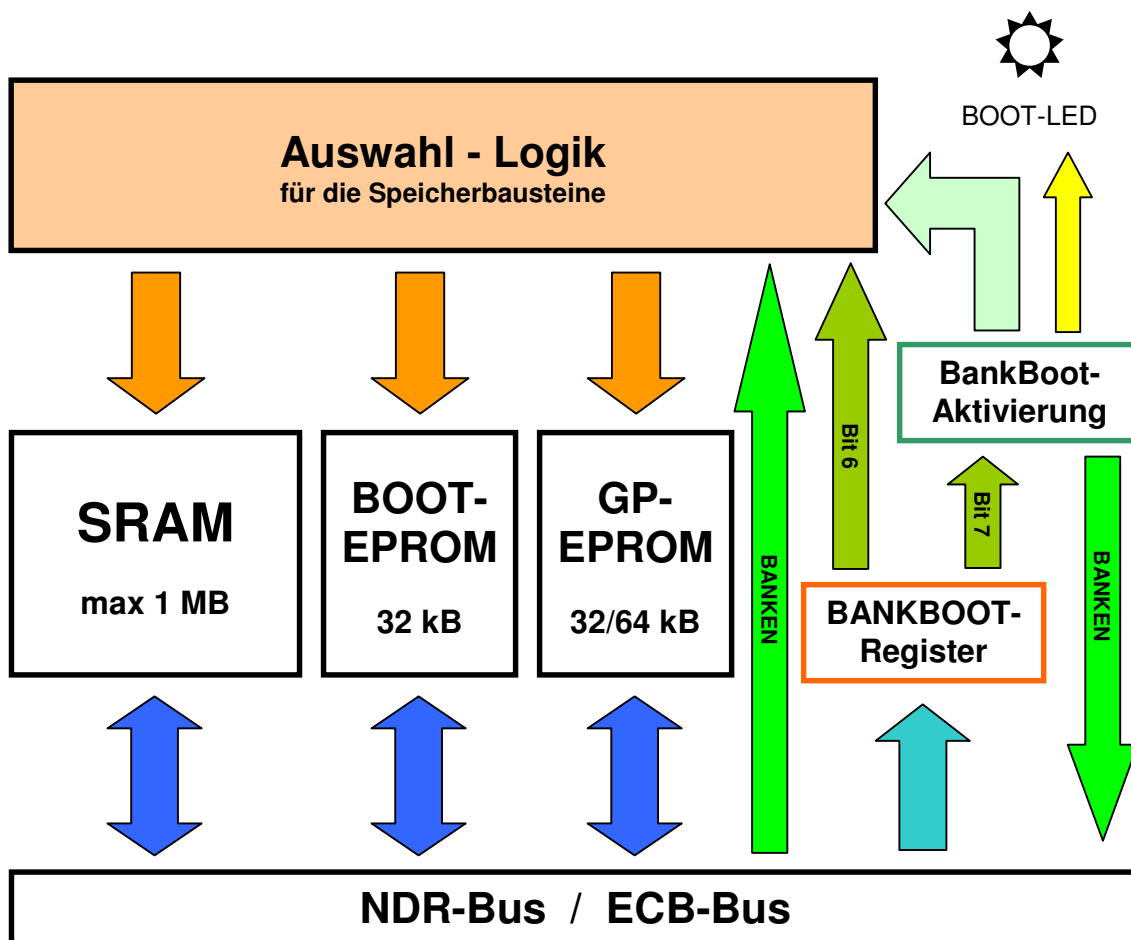
## 2 Technische Daten

- Europakarte 160 x 100 mm doppelseitig mit Bestückungsaufdruck
- NDR-Bus und ECB-Bus
- Stromaufnahme ca. 380 mA (Bestückung mit LS-Typen)
- BankBoot-Logik mit EPROM für das BankBoot-Programm
- EPROM für das Grundprogramm (wird nach dem Start ins RAM kopiert und gestartet).
- Fehleranzeige durch LED

### 3 Prinzipbeschreibung

Diese Baugruppe ersetzt alle bisher bekannten Speicherbaugruppen und die BANKBOOT-Baugruppe. Um die Funktionalität der Baugruppe abzurunden, habe ich zusätzlich eine Logik implementiert, das Grundprogramm in einem EPROM auf die Platine zu bringen. Der beim NKC maximal mögliche Adreßbereich kann mit RAM bestückt werden. Damit die Baugruppe auch in 16-Bit und 32-Bit Systemen verwendet werden kann, war es notwendig die BANKBOOT-Logik mittels Jumper deaktivieren zu können, ohne die Funktion des Bussignals BANKEN zu beeinflussen.

#### 3.1 Blockschaltbild BOOTRAM68k



### 3.2 Beschreibung des Blockschaltbildes

Das zentrale Element der Baugruppe ist die Auswahllogik für die Speicherbausteine. Die Komplexität der Logik ist so hoch, daß man sie nicht mehr mit TTL-Bausteinen auf eine Europakarte bringen kann. Deshalb kommt hier ein CPLD zum Einsatz.

Die Logik besteht aus zwei Teilen; dem Bankboot-Manager und der Speicherbaustein-Auswahllogik. Im Bankboot-Manager werden die Zustände der beiden Steuerbits des Kontrollregisters verwaltet und die Steuersignale für das Einblenden SRAMs, des BOOT-EPROMs und des Grundprogramm-EPROMs erzeugt. Die Speicherbaustein-Auswahllogik ermittelt aus den Signalen vom Bankboot-Manager und den Bussignalen ob und welcher Speicherbaustein angesteuert werden soll, wenn von der CPU Schreib- bzw. Leseanforderungen kommen. Die logischen Verknüpfungen können Sie einfach aus den VHDL-Skript entnehmen.

### 3.3 Die Baugruppe als RAM-Erweiterung

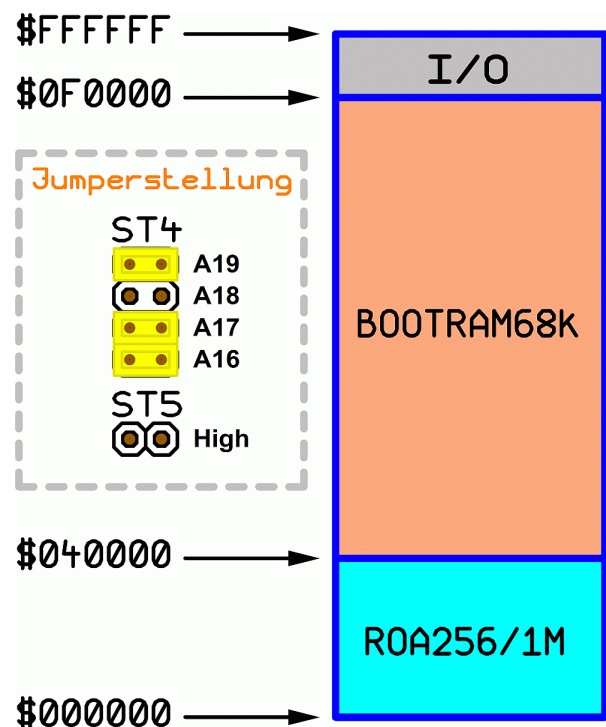
Die Baugruppe soll auch mit bereits im NKC vorhandenen Speicherbaugruppen betrieben werden können. Deshalb muß eine Möglichkeit vorhanden sein, den Adressbereich der anderen Speicherkarten auf der BOOTRAM68k ausblenden zu können. Die BOOTRAM68k unterstützt daher zwei Konfigurationsmodelle, die mittels Jumper eingestellt werden. Mit der Steckleiste ST5 wird das Modell eingestellt und mit der Steckleiste ST4 die Anfangs- bzw. Endadresse, die die BOOTRAM68k verwalten soll.

#### Das Speichermodell 1:

Im hier gezeigten Beispiel befindet sich eine ROA256/1M-Baugruppe im System, die die ersten 256 kB des Adreßbereichs belegt.

ST5 ist offen und veranlaßt die BOOTRAM68k den Adressbereich ab der mit ST4 eingestellten Adresse, hier \$040000, bis zum Ende mit RAM zu belegen.

Die BankBoot-Option funktioniert mit diesem Speichermodell zwar auch, macht aber keinen Sinn.



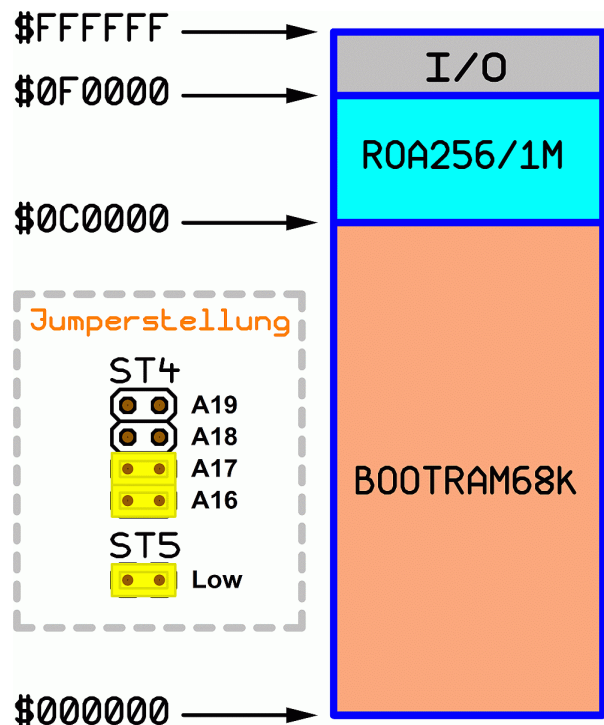
**Das Speichermodell 2:**

In diesem Beispiel befindet sich eine ROA256/1M-Baugruppe im System, die die letzten 192 kB des Adreßbereichs belegt.

ST5 ist hier gejumpert und veranlaßt die BOOTRAM68k den Adressbereich vom Beginn bis zur eingestellten Adresse – 1, hier \$0C0000, mit RAM zu belegen.

Die BankBoot-Option muß bei diesem Speichermodell aktiviert sein. Befinden sich auf der ROA256/1M EPROMs mit einem Grundprogramm, dann wird dieses bevorzugt gestartet.

Dieses Speichermodell muß auch benutzt werden, wenn sich keine weitere Speicherbaugruppe im System befindet. ST4 wird dann nicht gejumpert.

**3.4 Die BankBoot - Option**

Im Gegensatz zur BANKBOOT-Baugruppe muß hier die BankBoot-Option mittels Jumper (ST3) aktiviert werden. Im deaktivierten Zustand verändert die Baugruppe das Bussignal BANKEN nicht. Der entsprechende Portpin des CPLDs ist immer hochohmig, egal welchen Zustand das Bit 7 auf Port **\$C8** hat.

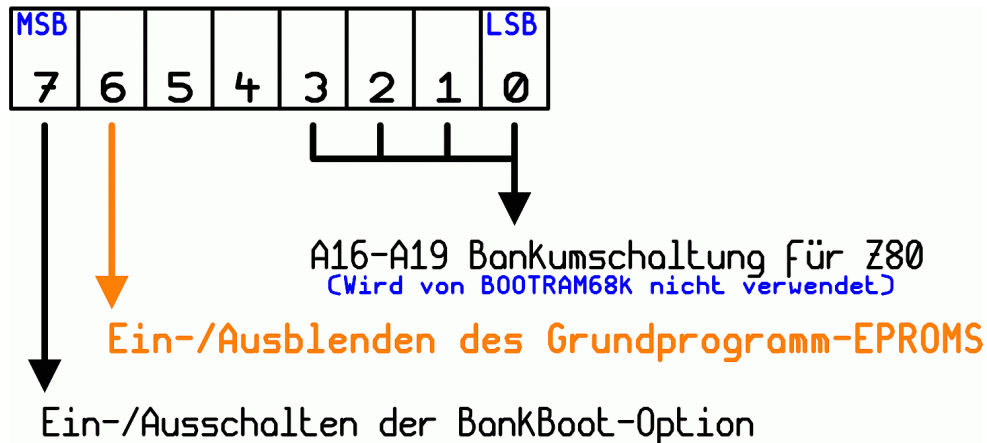
Im aktivierten Zustand verändert die Baugruppe aktiv das Bussignal BANKEN genau so, wie die Baugruppe BANKBOOT.

Im deaktivierten Zustand ändert die BOOTRAM68k das Bussignal BANKEN nicht. Die Baugruppe reagiert aber trotzdem noch auf Änderungen des Bussignales BANKEN, da dieses Signal über einen weiteren Pin am CPLD als Eingangssignal in die Auswertelogik des CPLDs geführt wird. D.h. das BOOT-EPROM wird nur dann eingeblendet, wenn es durch Bit 7 auf Port **\$C8** freigeschaltet ist. Dadurch ist sicher gestellt, daß andere Baugruppen (z.B.: COL256) das Bussignal BANKEN schalten und ihrerseits Speicher einblenden können ohne daß es zu Zugriffskollisionen mit der BOOTRAM68k kommt.

Bei 16-Bit-Systemen ist das Bussignal BANKEN beider Bushälften mit einander verbunden. Deshalb sollte hier die BankBoot-Option nur auf einer BOOTRAM68k-Baugruppe aktiviert sein. Das BankBoot-Programm geht davon aus, daß das bei der Baugruppe auf der Busseite der geraden Adressen der Fall ist.

### 3.5 Das EPROM für das Grundprogramm

Das EPROM für das Grundprogramm kann nicht permanent in den Adressbereich der CPU eingeblendet werden. Deshalb muß es vom Bootprogramm ins RAM kopiert werden. Um auf das EPROM zugreifen zu können, wurde der Port **\$C8** erweitert:



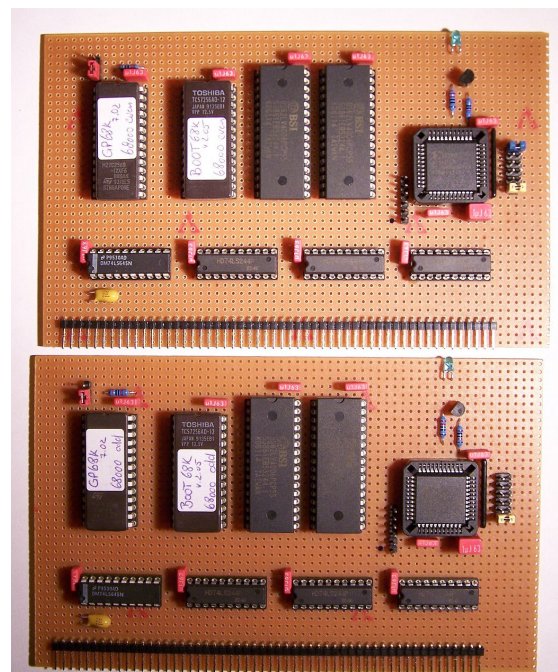
Wenn das Bit 6 auf '1' gesetzt wird, dann blendet die Baugruppe das Grundprogramm-EPROM ab der Adresse \$080000 in den Adressierungsbereich der CPU ein. Anschließend kann das Grundprogramm in den gewünschten Speicherbereich kopiert werden. Danach wird das Bit 6 auf '0' gesetzt und das Grundprogramm-EPROM ist wieder ausgeblendet.

### 3.6 Betrieb mit CPU68000

Zum Betrieb mit der Baugruppe CPU68000 benötigen Sie zwei BOOTRAM68k-Baugruppen. Eine für gerade und eine für ungerade Adressen. Entsprechend müssen BankBoot- und Grundprogramm auf je zwei EPROMs gebrannt werden.

Die BankBoot-Logik darf aber nur auf der Busseite der geraden Adressen aktiviert sein (blauer Jumper auf rechten Seite).

Ansonsten sind beide Platinen identisch.





### **3.7 Die Aktivitäts-/Fehleranzeige**

Die LED leuchtet nach dem Einschalten des NKC. Sie zeigt jetzt an, daß das Boot-EPROM in den Adressebereich der CPU eingeblendet ist. Erlischt diese nach 1-2 Sekunden wieder, so ist alles in Ordnung und das Menü des Grundprogramms ist auf dem Monitor zu sehen.

Leuchtet die LED dagegen permanent, dann ist entweder nicht ausreichend RAM im System verfügbar, um das Bootprogramm zu kopieren oder das Boot-EPROM enthält ein fehlerhaftes BankBoot-Programm.

Blinkt die LED mit langem Intervall, dann kann das Bootprogramm kein Grundprogramm finden. Das Bootprogramm sucht dabei erst im gesamten Adreßbereich der jeweiligen CPU und dann im Grundprogramm-EPROM auf der eigenen Baugruppe.

## **4 Aufbauanleitung**

### **4.1 Umgang mit ICs**

CMOS-Bausteine sind hochempfindlich gegen elektrostatische Aufladung! Bewahren oder Transportieren Sie CMOS-Bausteine nur auf leitenden Schaumstoff! Alle Pins müssen kurzgeschlossen sein.

Achten Sie darauf, daß Sie Verbindung mit einer Erdungsmöglichkeit haben, bevor Sie mit diesen Bausteinen arbeiten. Geeignete ESD-Artikel gibt es im Fachhandel.

## 4.2 Stückliste

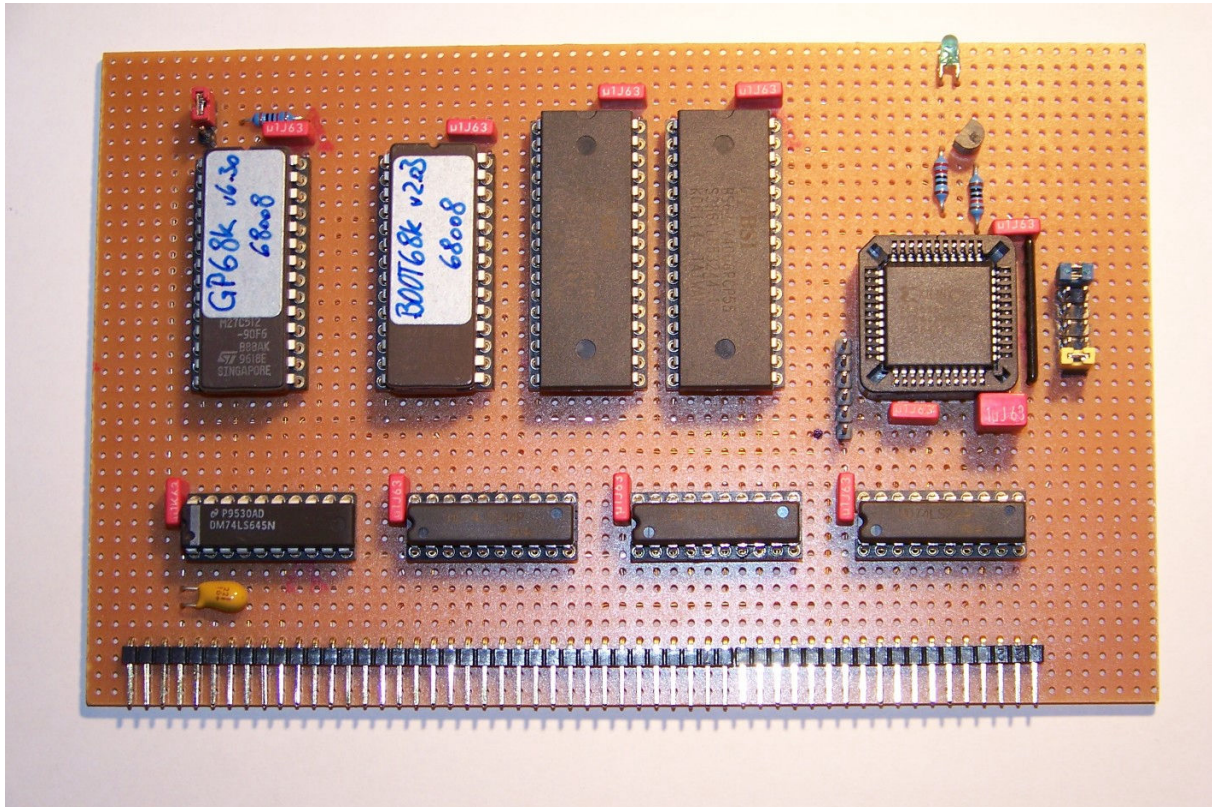
Anzahl	Kennung	Bauteil (klassisch)	Bauteil (alternativ)
1			Platine
2	IC 1, 2		SRAM 628512
1	IC 3		EPROM 27C256
1	IC 4	EPROM 27C256	EPROM 27C512
1	IC 5		Xilinx XC9536PC44-15
1	IC 6	74LS645	74HCT645
3	IC 7 - 9	74LS244	74HCT244
1	T 1		Transistor BC337
1	C 1	Kondensator 22µF Tantal	Kondensator 22 µF Elko
1	C 8	Kondensator 1 µF MKS-2	Kondensator 1 µF Elko
11	C 2 – 7, 9 - 13	Kondensator 100nF MKS-2	Kondensator 100nF X7R
1	R 1		Widerstand 330 Ω
2	R 2, 3		Widerstand 10 kΩ
1	RN 1		Widerstandsnetzwerk 8 x 4,7 kΩ
1	D 1		LED, Standard
2	IC 1, 2		IC-Sockel DIL32
2	IC 3, 4		IC-Sockel DIL28
4	IC 6 – 9		IC-Sockel DIL20
1	IC 5		IC-Sockel PLCC44
1	ST 1	Messerleiste DIN 41612 Bauform C 64 pol	für ECB-Bus
1	ST 2	Stiftleiste 1 x 36 pol & 1 x 18 pol gew.	für NDR-Bus
1	ST 3, 4, 5		Stiftleiste 2 x 6 pol
1	ST 6		Stiftleiste 1 x 3 pol
1	ST 7		Stiftleiste 1 x 6 pol
3	ST 3, 5, 6		Jumper verschiedener Farbe

## 4.3 Aufbau Schritt für Schritt

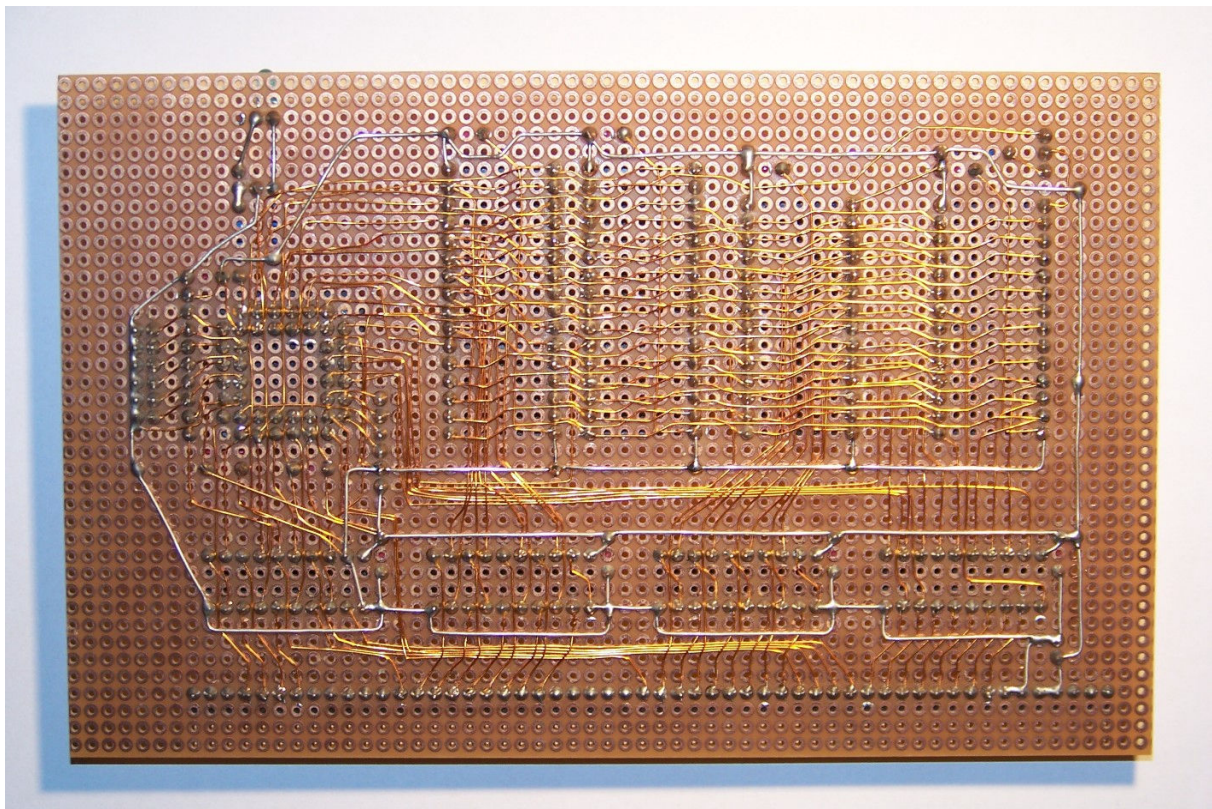
### Entfällt

Eine Platine werde ich aus Kostengründen vorerst nicht produzieren lassen. Deshalb spare ich mir eine detaillierte Anleitung.

Ich habe ich mir eine Prototypenplatine möglichst layoutnah gefädelt. Als Anregung zum Nachbau die Bilder meiner Baugruppe:



Prototypenplatine gefädelt, Bestückungsseite (Vollausbau für CPU 68008)



Prototypenplatine gefädelt, Lötseite



## 4.4 Programmierung des Xilinx-CPLD

### Kurzanleitung:

- ✱ Starten das Programm „Xilinx ISE Webpack“ auf Ihrem PC.
- ✱ Legen Sie ein neues Projekt mit dem Namen „BOOTRAM68k“ an.
- ✱ Wählen Sie als Zielbaustein „XC9536“, „15ns“ und „PLCC44-Gehäuse“ aus.
- ✱ Importieren Sie alle Dateien aus dem Verzeichnis „Xilinx“.
- ✱ Legen Sie „BOOTRAM68k.vhd“ als Hauptmodul fest.
- ✱ Starten Sie jetzt den kompletten Synthesedurchlauf. Eventuell auftretende Warnungen können Sie einfach ignorieren.
- ✱ Schließen Sie jetzt Ihren Programmer an Ihren PC an und verbinden den JTAG-Header mit der BOOTRAM68k-Baugruppe.
- ✱ Starten Sie jetzt das JTAG-Programmiersprogramm iMPACT.
- ✱ Klicken Sie auf die graphische Darstellung des ICs und starten die Programmierung des CPLDs.

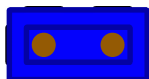
Diese Kurzanleitung setzt voraus, daß Sie sich schon mit Xilinx-CPLDs und dem ISE-Webpack beschäftigt haben. Eine genaue Beschreibung mit allen „Handgriffen“ würde den Rahmen dieser Anleitung sprengen. (Siehe: XILINX Programmable Logic Design)

## 4.5 Belegung von Steckleisten, Buchsen und Jumpern

### ST3:

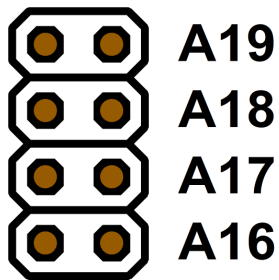


Die BankBoot-Logik ist deaktiviert.  
Das Bussignal BANKEN wird von der Baugruppe  
nicht aktiv geschaltet.



Die BankBoot-Logik ist aktiviert.  
Das Bussignal BANKEN kann jetzt aktiv geschaltet werden.  
Siehe Baugruppe BANKBOOT.

## ST4:



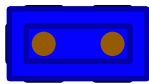
Einstellung der Start- bzw. Endadresse, je nach Einstellung von ST5, die die Baugruppe adressieren soll.

**Keine Jumper ist Standardeinstellung bei Vollausbau**

## ST5:



Der Adreßbereich wird von Adresse aus ST4 bis maximale Adresse (\$FFFFFF) mit dieser Baugruppe adressiert.



Der Adreßbereich wird von \$000000 bis Adresse aus ST4 mit dieser Baugruppe adressiert.

**Standardeinstellung bei Vollausbau**

## ST6:



Grundprogramm-EPROM ist vom

Typ 27C512

für CPU M68008

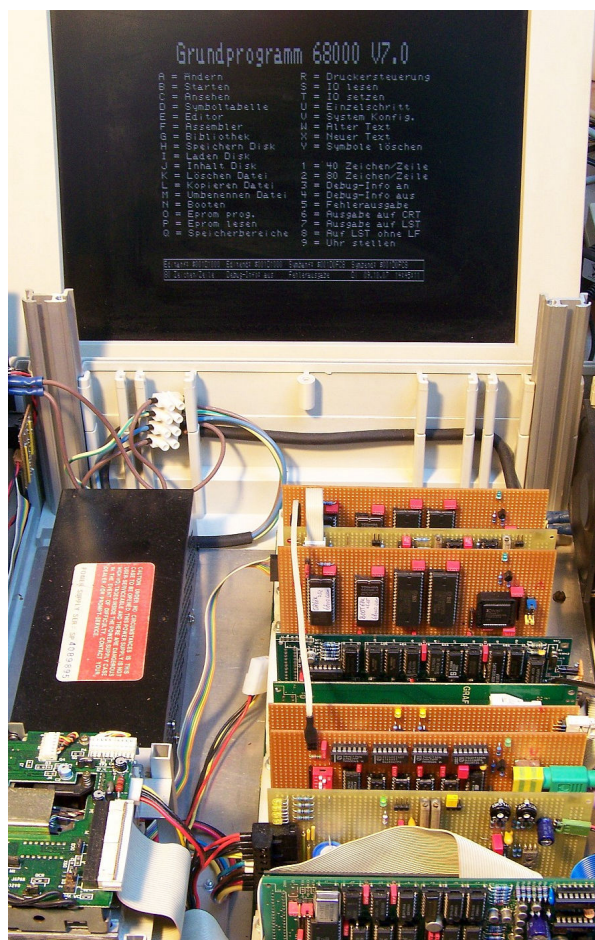
Grundprogramm-EPROM ist vom

Typ 27C256

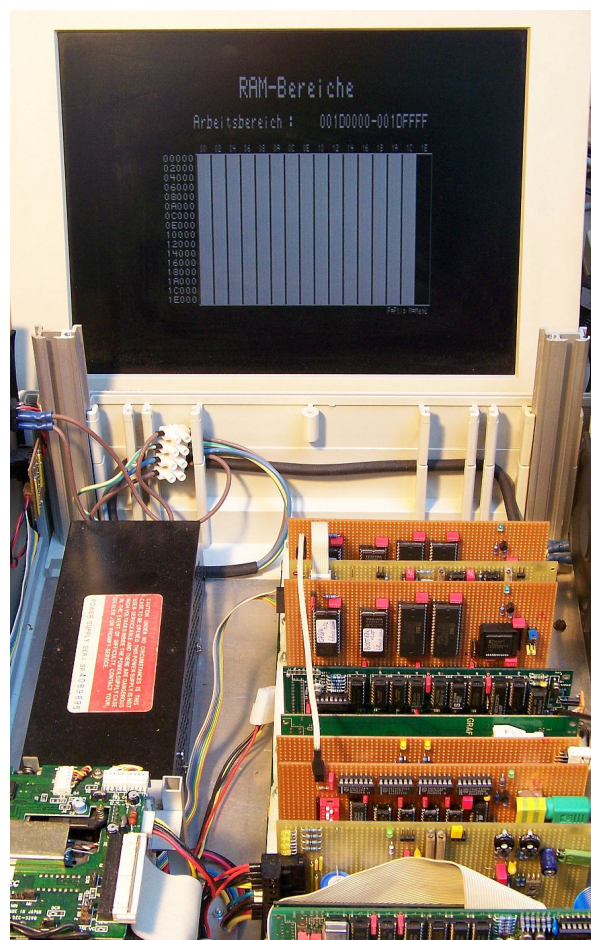
für CPU M68000

## 4.6 Alles Richtig gemacht ?

Wenn kein Fehler beim Bau der Platinen passiert ist und die EPROMs richtig gebrannt sind, sollte Ihr Bildschirm nach dem Einschalten Ihres NKC folgendes anzeigen:



**NDR-Klein-Computer** CPU68000, 2x BOOTRAM68k  
Grundprogramm68000 v7.02 - Hauptmenü



**NDR-Klein-Computer** CPU68000, 2x BOOTRAM68k  
Grundprogramm68000 v7.02 - Speicherbereiche

## 5 Neuerungen und Fehlerkorrekturen

In Laufe der Zeit haben sich leider ein paar Unzulänglichkeiten im Design der Logik des CPLDs gezeigt. Bei CPU-Taktfrequenzen über 8 MHz treten Timingprobleme auf. Diese konnten durch ein Neudesign beseitigt werden. Verwenden Sie zur Programmierung Ihres CPLDs nur die JEDEC-Datei nach dem 23.03.2008.

## 6 Literaturhinweise und –nachweise

### Bücher

- ❖ Rolf-Dieter Klein  
„Rechner modular“  
Der NDR-Klein-Computer – selbstgebaut und programmiert  
Franzis-Verlag, München. ISBN 3-7723-8721-7
- ❖ Rolf-Dieter Klein  
“Die Prozessoren 68000 und 68008“  
Rechnerarchitektur und Sprache im NDR-Klein-Computer  
Franzis-Verlag, München. ISBN 3-7723-7651-7

### Datenblätter

- \* SRAM: BS62LV4006, Rev. 1.1 vom Jan/2004
- \* BANKBOOT Aufbauanleitung
- \* XILINX Programmable Logic Design
- \* XILINX AppNote 073; Designing With XC9500 CPLDs
- \* XILINX AppNote 105; A CPLD VHDL Introduction

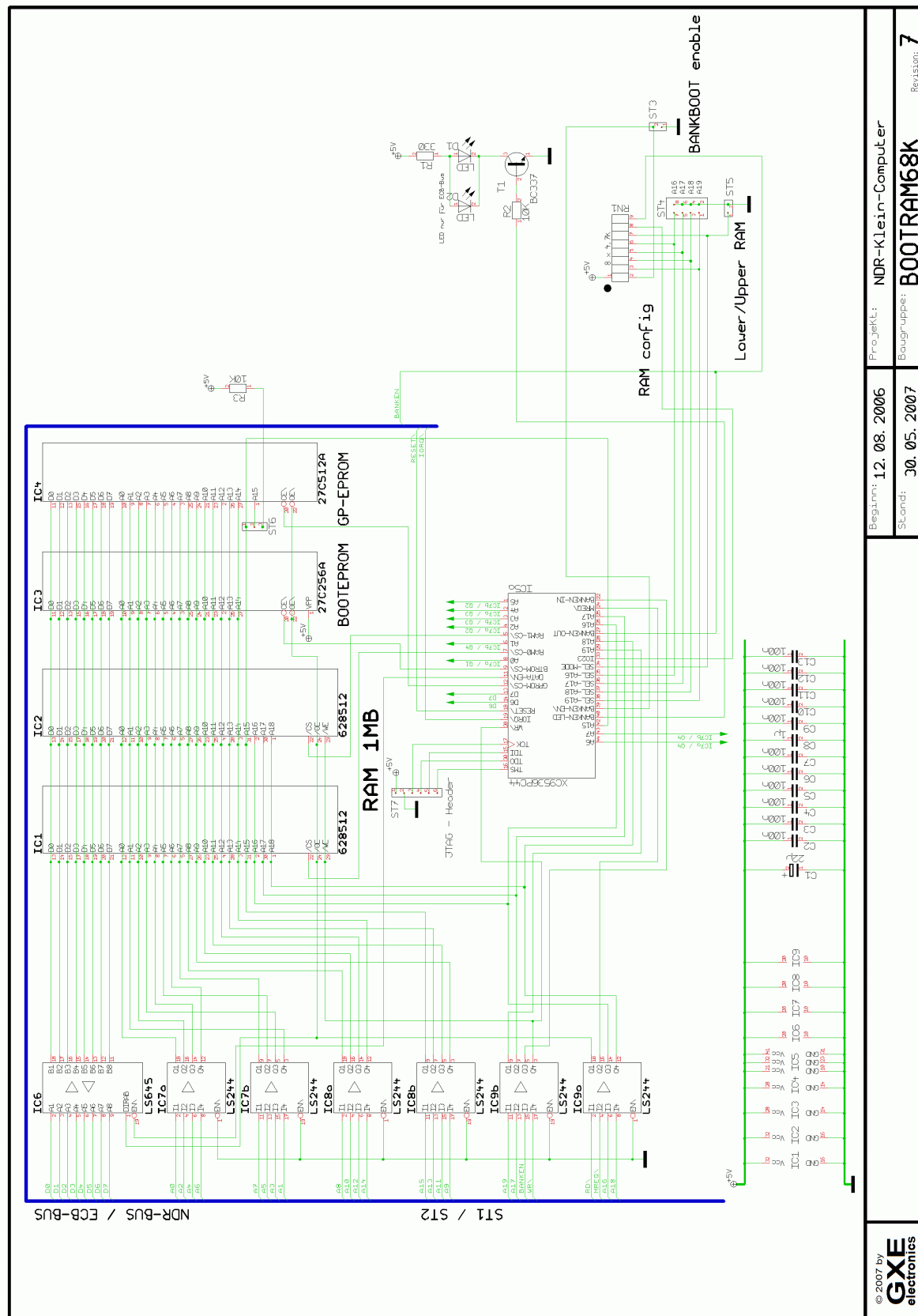
### Quellen im Internet

<http://www.xilinx.com/>  
[http://www.xilinx.com/ise/logic\\_design\\_prod/webpack.htm](http://www.xilinx.com/ise/logic_design_prod/webpack.htm)  
<http://ulrichradig.de/>

### NKC-Forum

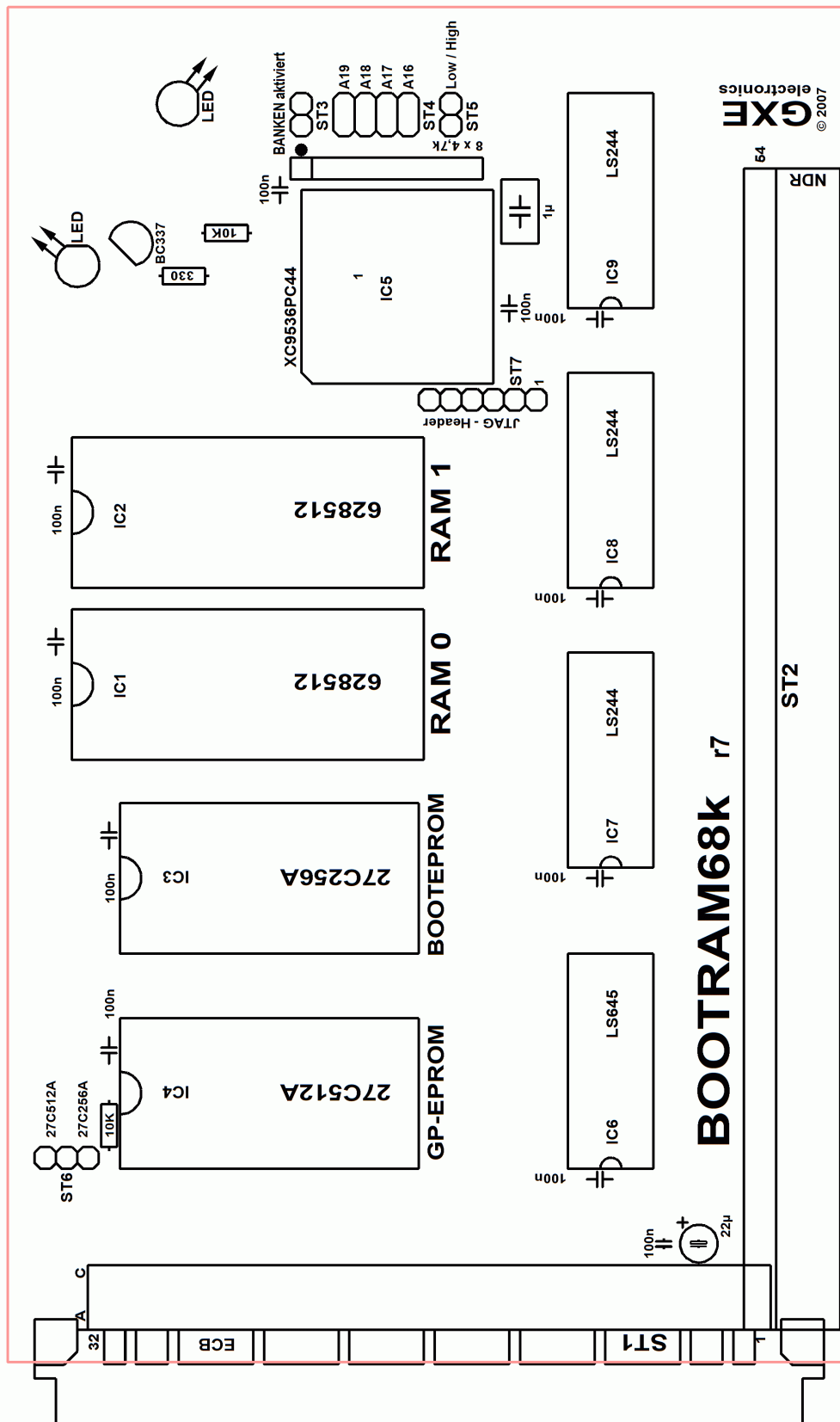
<http://www.drcrazy.de/forum/>

## Schaltplan BOOTRAM68k

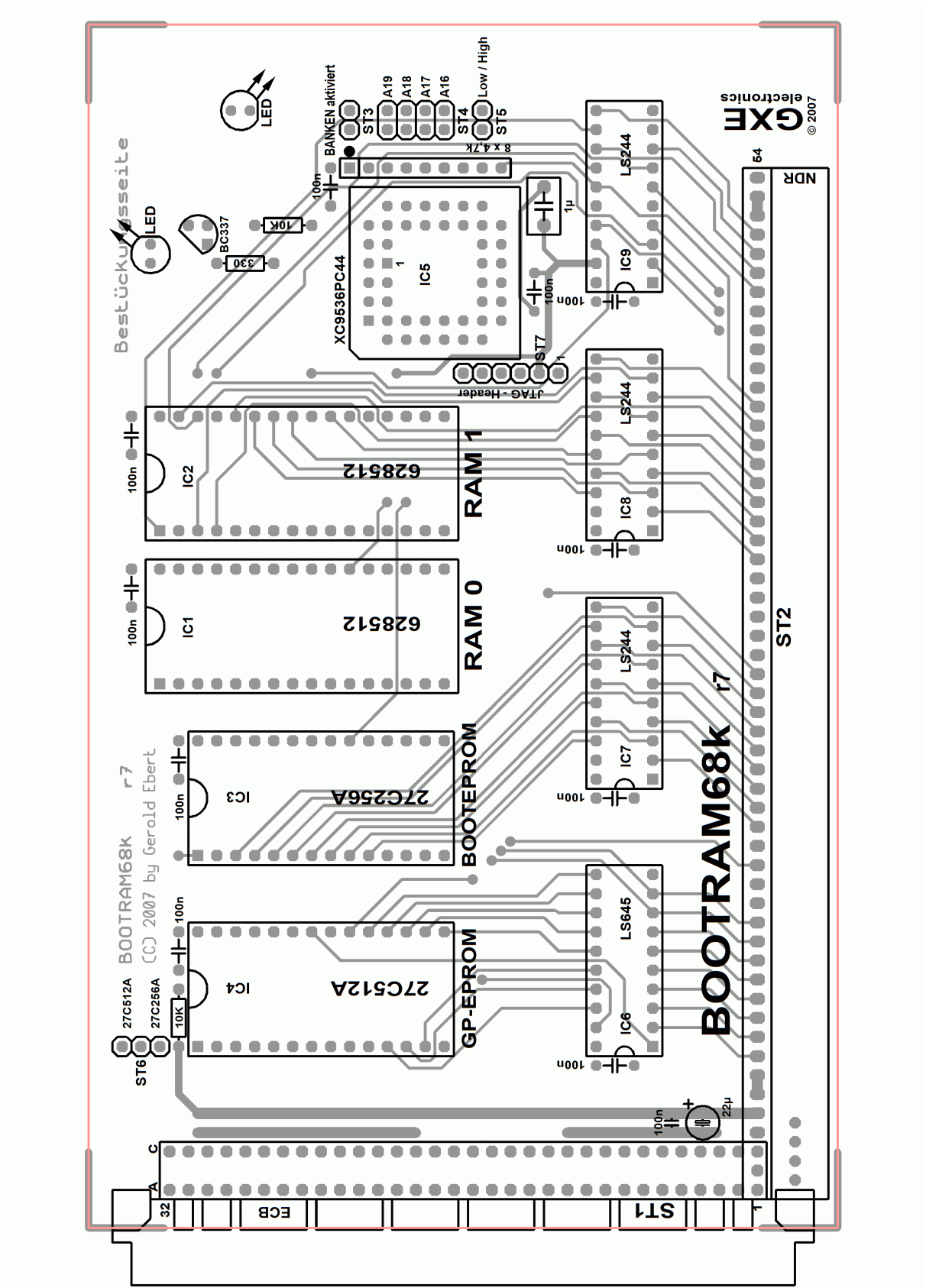




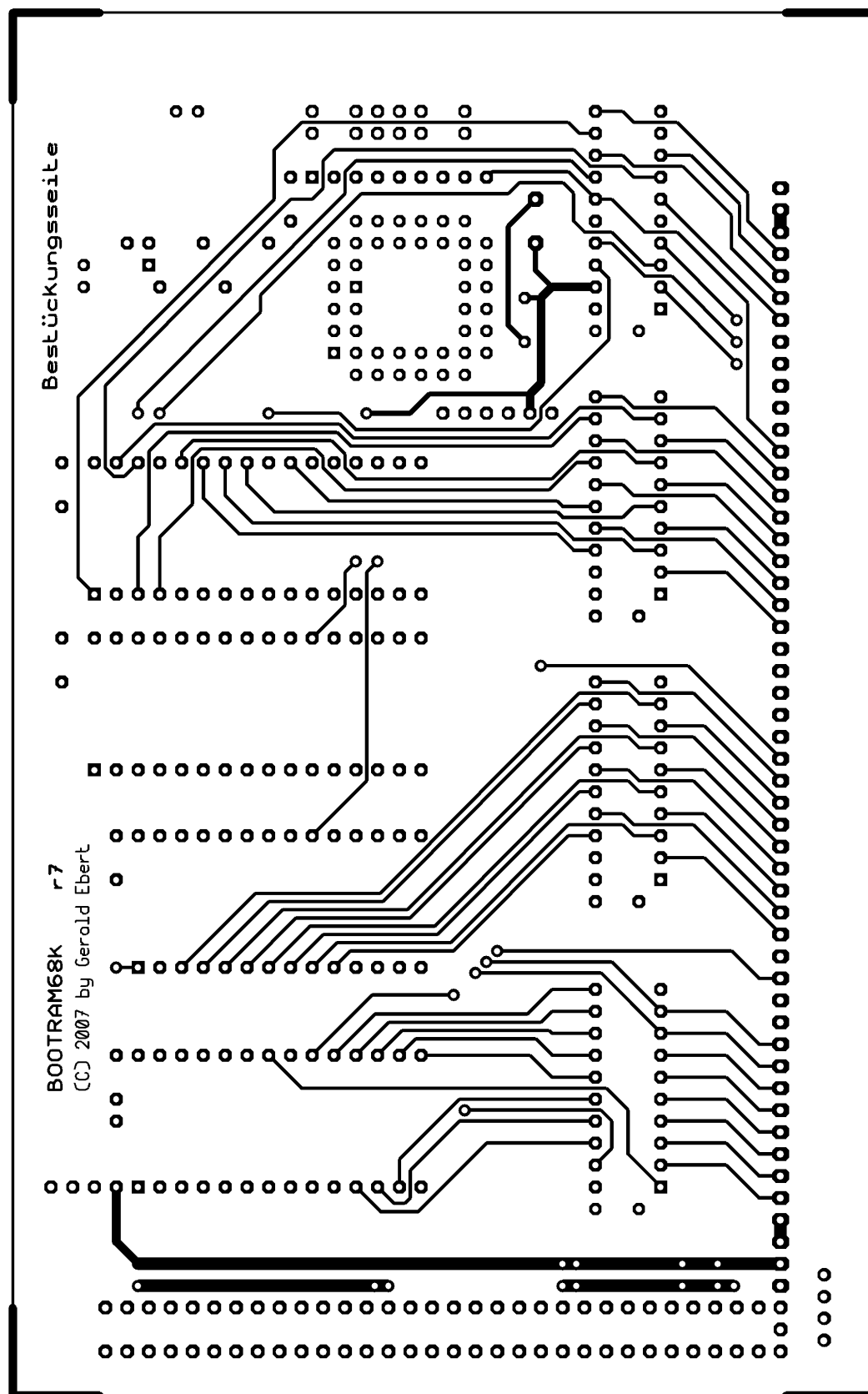
## Bestückungsplan



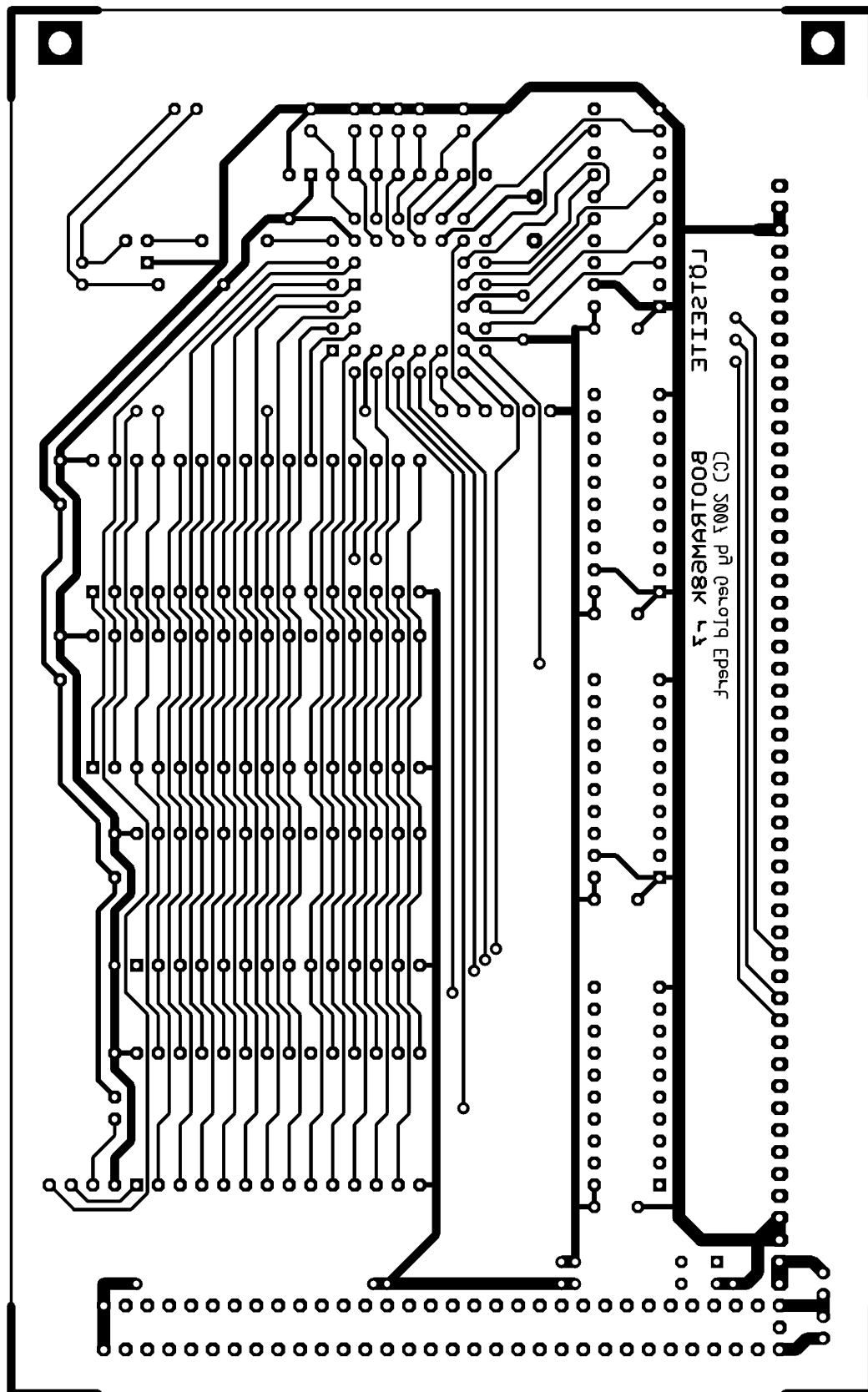
Layout Bestückungsseite mit Aufdruck



## Layout Bestückungsseite

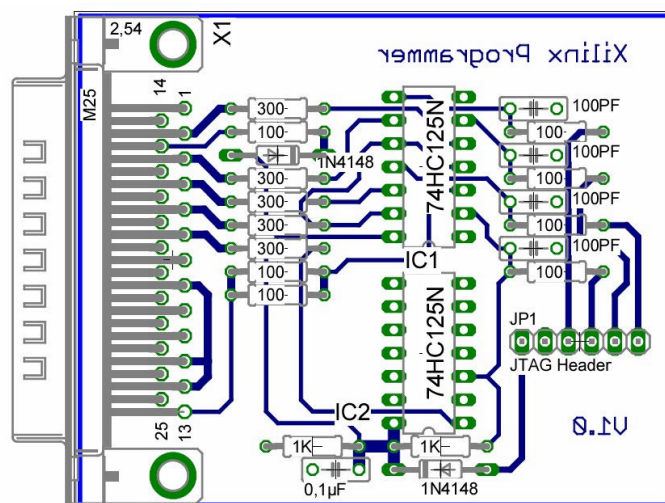
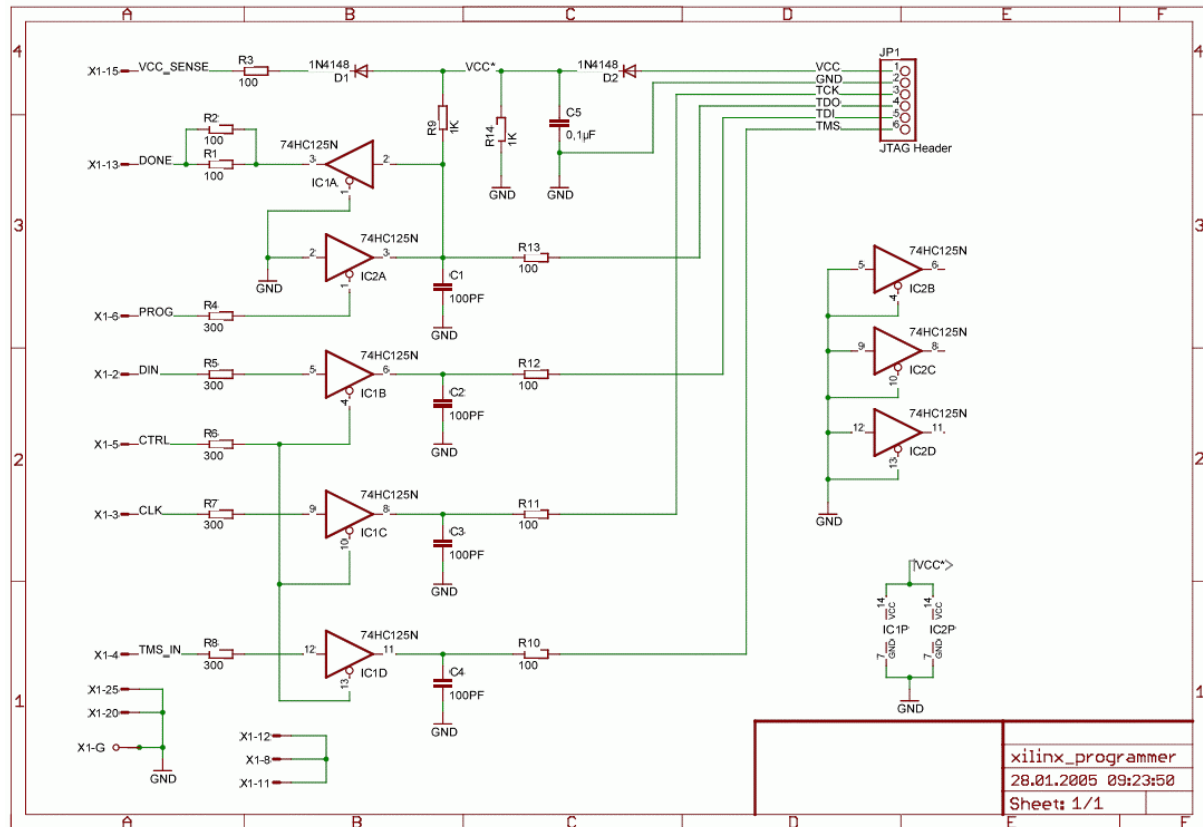


## Layout Lötseite



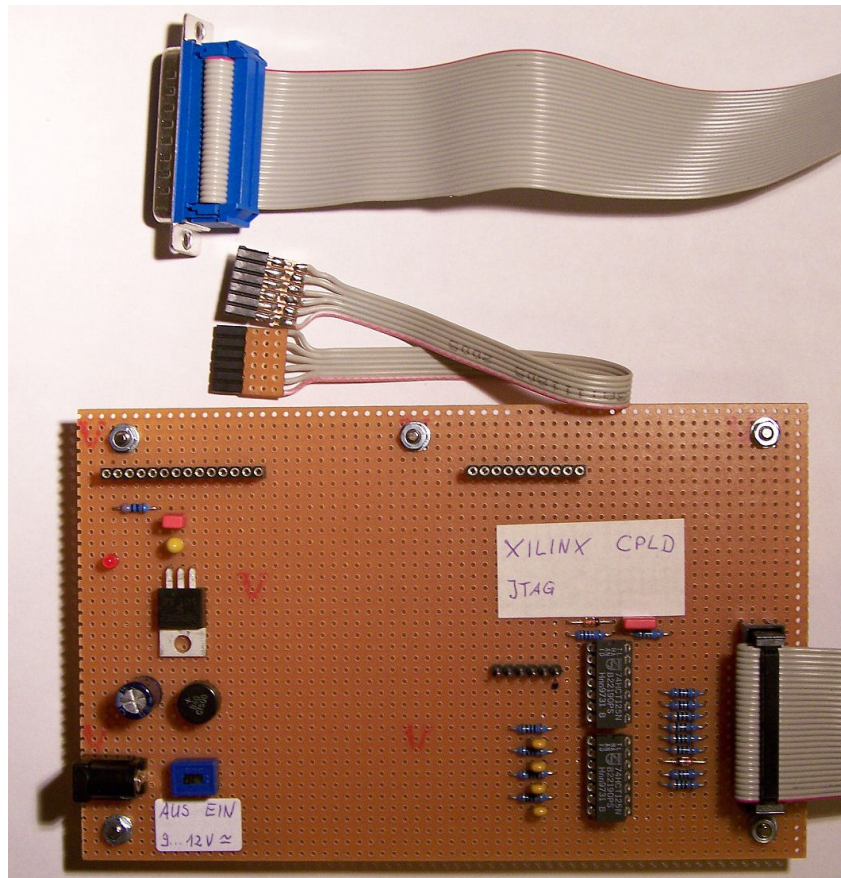
## Schaltplan Xilinx Programmer

Zur Programmierung der Xilinx-CPLDs verwende ich den Programmer von Ulrich Radig. Ich habe seinen Schaltplan und Platinenlayout als Beispiel aufgeföhrt. Eine genaue Beschreibung finden Sie auf seiner Website: <http://ulrichradig.de/>.

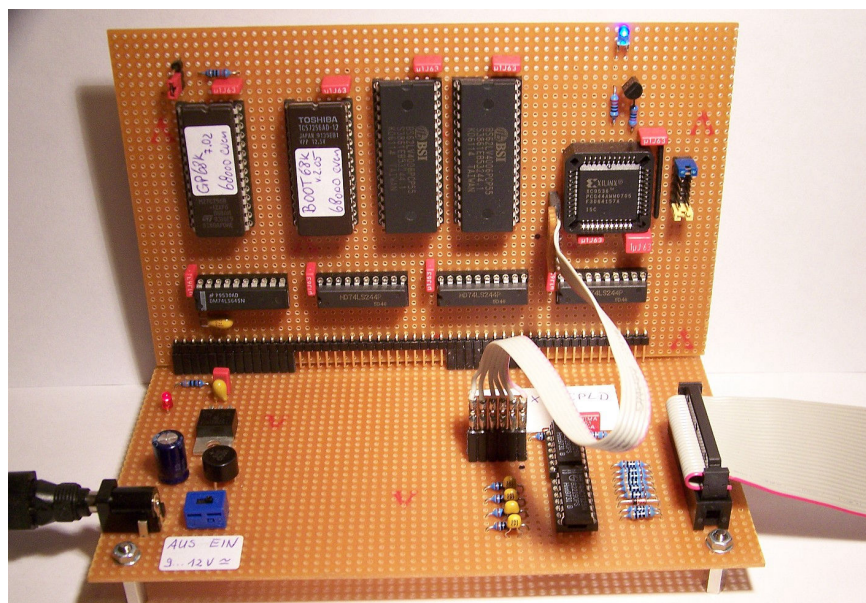


## Bilder von meinem Xilinx Programmer

Mein Xilinx-Programmer ist ein Nachbau des Layouts von Ulrich Radig. Zusätzlich habe ich noch eine einfache Stromversorgung für die NKC-Baugruppe integriert. Das macht die Programmierung noch einfacher ...



... Baugruppe auf den Programmer stecken. JTAG-Header verbinden und los geht's.





# Das BankBoot Programm

```
;*****  
;***'  
;***' .....BOOT-Programm fuer BOOTRAM68k.....'  
;***'  
;***'  
;***'(P)+(C) 2007 Gerald Ebert  
;***'  
;*****  
;*****  
;***' Version 2.06 vom 12.10.2007 '  
;*****  
  
;=====
```

```
; Bootfehler Kennung mittels Blinken der Bankboot-LED:  
  
; Dauerleuchten = Boot-EEPROM nicht vorhanden oder fehlerhaft oder  
;                 Bootprogramm kann nicht in RAM kopiert werden  
; 1x Blinken    = Kein Grundprogramm gefunden  
; 2x Blinken    = Zu wenig RAM. GP-EEPROM kann nicht kopiert werden  
  
;=====
```

```
*>>>>>> LABELS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  
  
* Für die jeweilige CPU diese Labels aus-/kommentieren.  
* Zusätzlich entsprechende Befehle in den Funktionen  
* 'toggled', 'showgpep' und 'hidegpep' aus-/kommentieren.
```

```
*=====  
M68008      =====  
cpu          equ     1                      * M68008  
hideall       equ     $80                   * BOOT-EEPROM und GP-EEPROM ausblenden  
showbogp      equ     $c0                   * Grundprogramm-EEPROM einblenden  
*=====  
M68000      =====  
cpu           equ     2                      * M68000  
hideall        equ     $8080                * BOOT-EEPROM und GP-EEPROM ausblenden  
showbogp        equ     $c0c0               * Grundprogramm-EEPROM einblenden  
*=====
```

```
gp kenn       equ     $5aa58001              * Grundprogramm Kennung  
code_bra      equ     $6000                  * OP-Code fuer bra <d16>  
minram         equ     $00040000             * min. RAMgroesse mit GP-EEPROM = 256 kB  
  
keys           equ     $ffffff69*cpu         * KEY-Schalter + Ruecksetzen  
bankboot       equ     $fffffff8*cpu         * IO-Adr Bankboot-Register  
  
bootstack      equ     $0000fffc*cpu         * Stackadresse nach RESET  
bootdest       equ     $00008000*cpu         * Startadresse des Bootprogramms  
banksiz        equ     $00010000*cpu         * Groesse einer Speicherbank  
gpepradr       equ     $00080000*cpu         * Startadresse des GP-EPROMSs  
adrspond       equ     $000f0000*cpu         * Ende des Adressbereichs
```

```
*>>>>>> VEKTOR-TABELLE <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
offset        $30000  
org            0  
  
dc.l bootstack  
dc.l bootstart  
df.l 64-2, bootstart  
df.l 256-64, 0
```

[illegible]



[illegible]

```

*>>>>>      Ermittle ob sich an Speicheradresse a0 RAM befindet <<<<
*>>>>>>      Hilfsprozedur, Register werden nicht gesichert
;'            a0 <= zu pruefende Speicheradresse

chk1ram:
    movem.l    (a0)+,d0-d3          * Speicherzellen sichern
    movem.l    d4-d7,-(a0)         * Testpattern in Speicher schreiben
    cmp.l      (a0)+,d4             * Prüfe 1. Pattern
    bne.s      chk1ram1            * kein RAM, -> Ende
    cmp.l      (a0)+,d5             * Prüfe 2. Pattern
    bne.s      chk1ram2            * kein RAM, -> Ende
    cmp.l      (a0)+,d6             * Prüfe 3. Pattern
    bne.s      chk1ram3            * kein RAM, -> Ende
    cmp.l      (a0)+,d7             * Prüfe 4. Pattern
    bne.s      chk1ram4            * kein RAM, -> Ende
    movem.l    d0-d3,-(a0)         * Schreibe Speicherzelleninhalt zurueck
    bra        carryclr           * RAM gefunden, -> C = 0

chk1ram1:
    addq.l     #4,a0               * Adresse korregieren
chk1ram2:
    addq.l     #4,a0               * Adresse korregieren
chk1ram3:
    addq.l     #4,a0               * Adresse korregieren
chk1ram4:
    movem.l    d0-d3,-(a0)         * Schreibe Speicherzelleninhalt zurueck
    bra        carryset           * Kein RAM, -> C = 1

*>>>>>      Ermittle ob sich an Speicheradresse a0 RAM befindet <<<<
*>>>>>>      a0 <= zu pruefende Speicheradresse

chkram0:
    movem.l    d0-d7/a1,-(a7)      * Erstmal Register auf Stack retten
    lea        testpatt(pc),a1     * Adresse der Testpattern holen
    movem.l    (a1)+,d4-d7         * Testpattern in Register laden
    bsr        chk1ram             * Prüfe, ob an Adresse in a0 RAM ist
    movem.l    (a7)+,d0-d7/a1      * Hole Register wieder vom Stack
    rts                          * C enthaelt Ergebnis der Pruefung

*>>>>>      Ermittle des Ende des RAM-Bereichs <<<<<<<<<<<<
;'            a0 <= Letzte RAM-Adresse + 1 in 1kB Schritten

chkram:
    movem.l    d0-d7/a1,-(a7)      * Erstmal Register auf Stack retten
    lea        testpatt(pc),a1     * Adresse der Testpattern holen
    movem.l    (a1)+,d4-d7         * Testpattern in Register laden
    moveq      #0,d0
    movea.l    d0,a0               * Beginne Speichertest bei Adr 0
chkram1:
    bsr        chk1ram             * Prüfe, ob an Adresse in a0 RAM ist
    bcs.s      chkram2             * Kein RAM, Fertig.
    adda.l     #$400,a0            * Adresse um 1 kB erhoehen
    cmpa.l     #adrspend,a0        * Adressbereichsende erreicht ?
    bge.s      chkram2             * Ja, Fertig.
    bra.s      chkram1             * -> Weiter testen
chkram2:
    movem.l    (a7)+,d0-d7/a1      * Hole Register wieder vom Stack
    rts

```

```

*>>>>>>    Pruefe, ob an Adresse in a0 ein Grundprogramm ist
; '          a0 => zu testende Adresse

chkgrp:
    cmpi.l    #gpkenn,(a0)      * GP-Kennung prüfen
    bne.s     chkgrp1          * Keine Kennung -> Fehler
    cmpi.l    #cpu,$14(a0)      * CPU-Kennung muß uebereinstimmen
    bne.s     chkgrp1          * Falsche CPU -> Carry=1
    cmpi.w    #code_bra,$20(a0) * Hier muß ein 'bra <d16>' stehen
    bne.s     chkgrp1          * Keine Kennung -> Carry=1
    cmpi.w    #code_bra,$24(a0) * Hier muß ein 'bra <d16>' stehen
    bne.s     chkgrp1          * Keine Kennung -> Carry=1
    bsr       chkram0          * Ist GP im RAM ?
    bcc.s     chkgrp1          * Ja, das kann aber fehlerhaft sein
    bra       carryclr        * GP gefunden

chkgrp1:
    bra       carryset        * Kein GP gefunden

*>>>>>>    Pruefe, ob im Adressbereich ein Grundprogramm ist
; '          a0 <= Adresse der Grundprogramm Kennung

chkgrpssp:
    lea       $400,a0          * Beginne nach der Vektortabelle
chkgrpssp1:
    bsr       chkgrp           * GP-Kennungen OK ?
    bcc       cende            * Ja, Grundprogramm gefunden. Fertig
    adda.l    #$1000,a0        * Adresse um 4 kB erhoehen
    cmpa.l    #adrspend,a0     * Adressbereichsende erreicht?
    bge       carryset        * Ja, Fertig. Kein GP -> Carry=1
    bra.s     chkgrpssp1      * Naechste Adresse pruefen

*>>>>>>    GP-EEPROM einblenden

showgpep:
*===== M68008 =====
*   move.b    #showbogg,bankboot.w * GP-EEPROM einblenden (M68008)
*===== M68000 =====
    move.w    #showbogg,bankboot.w * GP-EEPROM einblenden (M68000)
    rts

*>>>>>>    GP-EEPROM ausblenden

hidegpep:
*===== M68008 =====
*   move.b    #hideall,bankboot.w * GP-EEPROM ausblenden (M68008)
*===== M68000 =====
    move.w    #hideall,bankboot.w * GP-EEPROM ausblenden (M68000)
    rts

```

```

*>>>>>>      Pruefe, ob im GP-EPROM ein Grundprogramm ist

chkggepr:
    movem.l    a0,-(a7)          * Erstmal Register auf Stack retten
    lea        gpepradr,a0      * GP-EPROM Adresse holen
    adda.l     #$400,a0          * Ergibt Adresse der Kennung
    bsr.s      showgpep          * GP-EPROM einblenden
    bsr        chkgp             * GP-Kennungen OK ?
    bcs.s      chk1ggepr         * Nein, kein GP gefunden -> Carry=1
    bsr.s      hidegpep          * GP-EPROM ausblenden
    movem.l    (a7)+,a0          * Hole Register wieder vom Stack
    bra        carryclr          * GP gefunden

chk1ggepr:
    bsr.s      hidegpep          * GP-EPROM ausblenden
    movem.l    (a7)+,a0          * Hole Register wieder vom Stack
    bra        carryset          * Fehlerflag setzen

*>>>>>>      Kopiere Grundprogramm aus GP-EPROM nach a0    <<
; '            a0 => Zieladresse für Grundprogramm

copygp:
    movem.l    d0/d1/a0-a2,-(a7) * Erstmal Register auf Stack retten
    lea        gpepradr,a2      * GP-EPROM Adresse holen
    movea.l    a2,a1
    adda.l     #banksize,a1      * ergibt GP-EPROM Speicherbereich
    cmpa.l     a1,a0             * Liegt Zieladresse im GP-EPROM ?
    bge.s      copygp4           * Nein, oberhalb. -> Direkt kopieren
    movea.l    a0,a1
    adda.l     #banksize,a1      * ergibt Zielspeicherbereich
    cmpa.l     a1,a2             * Liegt Zielbereich im GP-EPROM ?
    bge.s      copygp4           * Nein, unterhalb. -> Direkt kopieren
    moveq      #16-1,d0          * 16 Bloecke sind zu kopieren

copygp1:
    bsr        showgpep          * GP-EPROM einblenden
    moveq      #256-1,d1         * 4 kB pro Block werden kopiert
    lea        cpybuf(pc),a1     * Pufferadresse holen

copygp2:
    move.l     (a2)+,(a1)+        * 16 Bytes kopieren
    move.l     (a2)+,(a1)+
    move.l     (a2)+,(a1)+
    move.l     (a2)+,(a1)+
    dbra       d1,copygp2
    bsr        hidegpep          * GP-EPROM ausblenden
    moveq      #256-1,d1         * 4 kB pro Block werden kopiert
    lea        cpybuf(pc),a1     * Pufferadresse holen

copygp3:
    move.l     (a1)+,(a0)+        * 16 Bytes kopieren
    move.l     (a1)+,(a0)+
    move.l     (a1)+,(a0)+
    move.l     (a1)+,(a0)+
    dbra       d1,copygp3
    dbra       d0,copygp1        * naechsten Block kopieren
    movem.l    (a7)+,d0/d1/a0-a2 * Hole Register wieder vom Stack
    rts

```

[illegible]

## Das VHDL-Skript für den CPLD

## BANKBOOT68K.VHD

```
-- Company:
-- Engineer:          Gerald Ebert
--
-- Create Date:       20:06:31 05/02/2007
-- Design Name:
-- Module Name:        BOOTRAM68k - Behavioral
-- Project Name:        BOOTRAM68k
-- Target Devices:     XC9536-xxPC44
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:           1.20
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;
```

---

ENTITY BOOTRAM68Main IS

```

PORT ( busdata6_7   : IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
      busadr0_7     : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
      busadr15_19   : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
      busiorq0      : IN  STD_LOGIC;
      busmreq0      : IN  STD_LOGIC;
      buswr0        : IN  STD_LOGIC;
      busreset0     : IN  STD_LOGIC;
      busbanken     : IN  STD_LOGIC;           -- BANKEN from NDR/ECB-Bus
      banken_en0    : IN  STD_LOGIC;           -- Bootoption if jumper is set
      ramsel_switch : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
      ramsel_mode   : IN  STD_LOGIC;           -- Lower Memory if jumper is set

      banken_out     : OUT  STD_LOGIC;         -- Simulates open collector gate
      banken_led     : OUT  STD_LOGIC;         -- Drives Boot Indicator LED
      ram0_cs0       : OUT  STD_LOGIC;         -- Chip Select RAM0
      ram1_cs0       : OUT  STD_LOGIC;         -- Chip Select RAM1
      booteprom_cs0  : OUT  STD_LOGIC;         -- Chip Select BootEPROM
      gpeprom_cs0    : OUT  STD_LOGIC;         -- Chip Select GP-EPROM
      databus_en0   : OUT  STD_LOGIC;         -- Enables '645 bus driver

```

END BOOTRAM68Main;

ARCHITECTURE Behavioral OF BOOTRAM68Main IS

```

CONSTANT lowsignal   : STD_LOGIC := '0';
CONSTANT adrbankboot : STD_LOGIC_VECTOR (7 DOWNTO 0) := "11001000";- Port $C8
CONSTANT adrgpeprom  : STD_LOGIC_VECTOR (3 DOWNTO 0) := "1000";  -- Adr $8xxxxxxx

SIGNAL  bankreg : STD_LOGIC_VECTOR (1 DOWNTO 0) := "00";          - Registerbits

SIGNAL  regmatch, iocsen0 : STD_LOGIC;
SIGNAL  komp_ram, komp_adrgp : STD_LOGIC;
SIGNAL  ibanken_out       : STD_LOGIC;
SIGNAL  ichipselectcode   : STD_LOGIC_VECTOR (4 DOWNTO 0);
SIGNAL  chipselect0       : STD_LOGIC_VECTOR (3 DOWNTO 0);

```

BEGIN

```

-----
-- Calc enable signal for registers
regmatch <= '0' WHEN busadr0_7 = adrbankboot ELSE '1';
iocsen0 <= buswr0 OR busiorq0 OR regmatch;

-- Handle BankBoot registers (d6/d7)
bankregister : PROCESS ( busreset0, iocsen0, busdata6_7 )
BEGIN
  IF ( busreset0 = '0' ) THEN
    bankreg <= "00";  -- Default states like original BANKBOOT
  ELSIF ( iocsen0 = '1' AND iocsen0'event ) THEN
    bankreg <= busdata6_7;  -- Enable GP-EPROM if '1'; Disable BootEPROM if '1'
  END IF;
END PROCESS bankregister;

-----
-- Handle bankboot logic and transfer signals
ibanken_out <= NOT(bankreg(1) OR banken_en0) WHEN busadr15_19(0) = '0'
                                                    ELSE '0';          -- high active
banken_led  <= NOT bankreg(1);                      -- high active
-- Simulate open collector gate to pull down BANKEN signal
bankendrv : OBUFE PORT MAP ( i => lowsignal, o => banken_out, e => ibanken_out);

```

```
=====
-- Compare bankaddresses for work modes
komp_ram   <= NOT ramsel_mode WHEN busadr15_19(4 DOWNT0 1) < ramsel_switch ELSE
ramsel_mode;
komp_adrgp <= bankreg(0) WHEN busadr15_19(4 DOWNT0 1) = adrgpeprom ELSE '0';

-- Get memory selection
ichipselcode <= busmreq0 & busbanken & komp_ram & busadr15_19(4) & komp_adrgp;
WITH ichipselcode SELECT
    chipsel0 <= "1110" WHEN "01100",           -- ram0_cs0
                  "1101" WHEN "01110",           -- ram1_cs0
                  "1011" WHEN "01001"|"01011"|"01101"|"01111", -- gpep_cs0
                  "0111" WHEN "00000"|"00001"|"00010"|"00011", -- bbep_cs0
                  "0111" WHEN "00100"|"00101"|"00110"|"00111",
                  "1111" WHEN OTHERS;

-- Transport signals
ram0_cs0   <= chipsel0(0);
ram1_cs0   <= chipsel0(1);
gpeprom_cs0 <= chipsel0(2);
booteprom_cs0 <= chipsel0(3) OR bankreg(1);

databus_en0 <= '0' WHEN chipsel0(2 DOWNT0 0) /= "111"
                  ELSE chipsel0(3) OR bankreg(1);

END Behavioral;
```

## BANKBOOT68MAIN.UCF

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

```
NET "banken_en0" LOC = "P39" ;
NET "banken_led" LOC = "P40" ;
NET "banken_out" LOC = "P27" ;
NET "booteprom_cs0" LOC = "P9" ;
NET "busadr0_7<0>" LOC = "P8" ;
NET "busadr0_7<1>" LOC = "P6" ;
NET "busadr0_7<2>" LOC = "P4" ;
NET "busadr0_7<3>" LOC = "P3" ;
NET "busadr0_7<4>" LOC = "P2" ;
NET "busadr0_7<5>" LOC = "P1" ;
NET "busadr0_7<6>" LOC = "P44" ;
NET "busadr0_7<7>" LOC = "P43" ;
NET "busadr15_19<0>" LOC = "P42" ;
NET "busadr15_19<1>" LOC = "P26" ;
NET "busadr15_19<2>" LOC = "P25" ;
NET "busadr15_19<3>" LOC = "P28" ;
NET "busadr15_19<4>" LOC = "P29" ;
NET "busbanken" LOC = "P22" ;
NET "busdata6_7<0>" LOC = "P14" ;
NET "busdata6_7<1>" LOC = "P13" ;
NET "busiorq0" LOC = "P19" ;
NET "busmreq0" LOC = "P24" ;
NET "busreset0" LOC = "P18" ;
NET "buswr0" LOC = "P20" ;
NET "databus_en0" LOC = "P11" ;
NET "gpeprom_cs0" LOC = "P12" ;
NET "ram0_cs0" LOC = "P7" ;
NET "ram1_cs0" LOC = "P5" ;
NET "ramsel_mode" LOC = "P34" ;
NET "ramsel_switch<0>" LOC = "P35" ;
NET "ramsel_switch<1>" LOC = "P36" ;
NET "ramsel_switch<2>" LOC = "P37" ;
NET "ramsel_switch<3>" LOC = "P38" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE