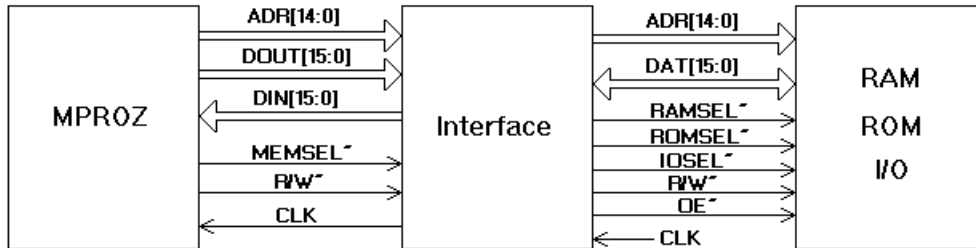


Rechnerorganisation II

1. Übungsblatt

Entwerfen Sie eine möglichst einfache Schaltung auf Register-Transfer-Ebene für einen 16-Bit Minimalprozessor MPROZ mit folgenden Vorgaben:



a) Schnittstellen des Prozessors

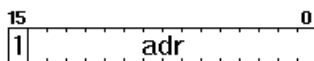
- adr[14:0] : 15-Bit Adreßbus
- din/dout[15:0] : 16-Bit Datenbus
- memsel~ : Zugriff auf ram/rom/io
- r/w~ : Schreib- / Lesezugriff
- clk : Takt

b) Register

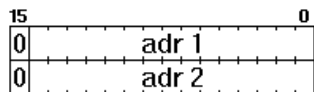
Um den Hardwareaufwand so klein wie möglich zu halten, enthält das Programmiermodell von MPROZ neben einem 15-Bit Befehlszähler (PC) und einem 1-Bit Zustandsflag (F) keine weiteren Register. Nach einem Reset enthalten PC und F den Wert 0.

c) Befehlssatz

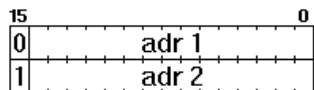
MPROZ unterstützt insgesamt drei Befehle:



Lade adr in PC falls F=0
Lösche F



Addiere den Inhalt von adr1 zu dem Inhalt von adr2 und speichere das Ergebnis in adr2.
Speichere das Übertragsbit der Addition in F.

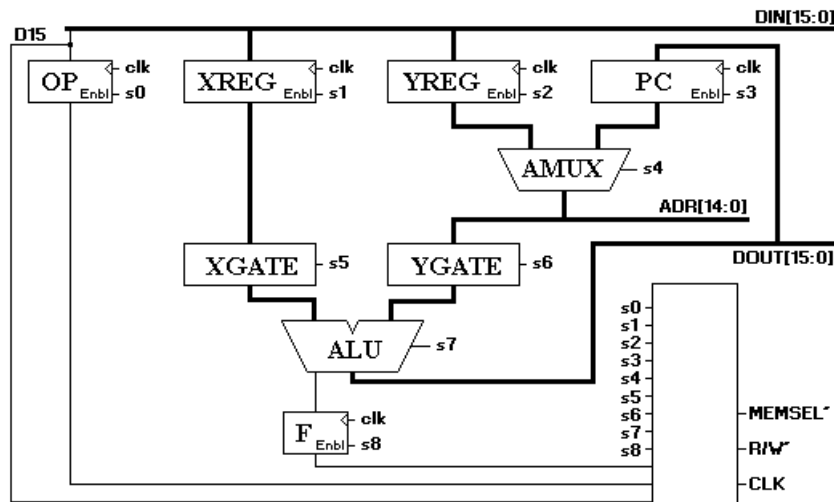


NOR-verknüpfe den Inhalt von adr1 mit dem Inhalt von adr2 und speichere das Ergebnis in adr2.
F=1 falls Ergebnis Null, sonst 0.

Rechnerorganisation II

2. Übungsblatt

1. Gegeben sei das folgende Schaltbild für MPROZ auf Register-Transfer-Ebene:



- a) Wie viele und welche Phasen (Zustände) werden benötigt, um die 3 Befehle des MPROZ abarbeiten zu können? Beschreiben Sie die Vorgänge in den einzelnen Phasen.
 - b) Zeichnen Sie das Zustandsübergangsdiagramm
 - c) Verwenden Sie zur Kodierung der Zustände nicht die minimale Anzahl von D-Flip-Flops, sondern verwenden Sie für jeden Zustand ein eigenes Flip-Flop, d.h. zu jedem Zeitpunkt ist genau ein Zustands-Flip-Flop gesetzt. Bestimmen Sie aus dem Zustandsübergangsdiagramm die Ansteuerfunktionen der Zustands-Flip-Flops.
 - d) Stellen Sie die Wertetabelle für die Ausgangsfunktionen auf und minimieren Sie diese unter Ausnutzung von don't care Feldern.
 - e) Geben Sie die Schaltung für das Steuerwerk ein (siehe Anhang A).
2. Entwerfen Sie die Schaltung für das Interface zum Speicher und der I/O-Hardware. Die Adresse \$0000 - \$3fff sollen dabei das ROM, die Adressen \$4000- \$7dff das RAM und die Adressen \$7e00 - \$7fff die I/O-Hardware ansprechen.
 3. Simulieren Sie den Prozessor (siehe Anhang B)

Rechnerorganisation II

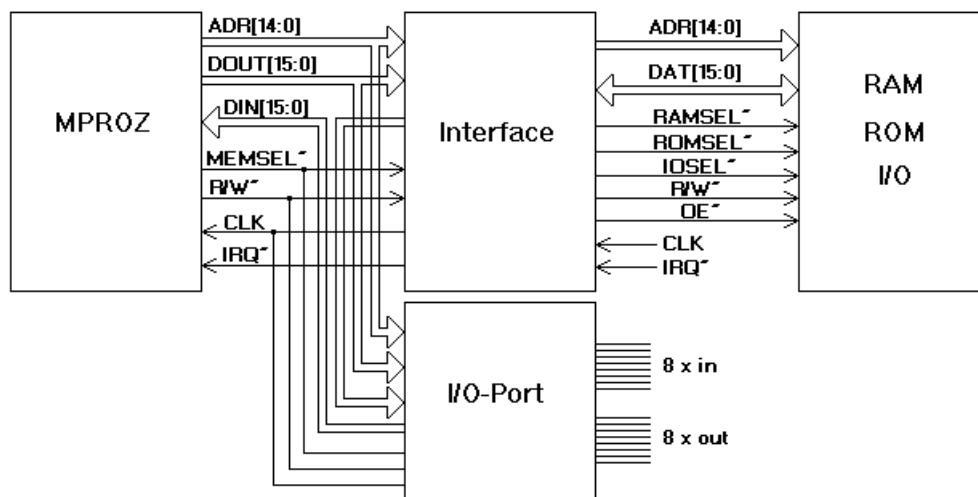
3. Übungsblatt

- Der Minimalprozessor MPROZ soll nun zusätzlich mit einer Interruptunterstützung ausgestattet werden. Dies soll wiederum durch eine möglichst einfache Hardware erfolgen, selbst wenn dies auf Kosten der Leistungsfähigkeit geht. Überlegen Sie insbesondere, ob sich die Schaltung dadurch vereinfachen läßt, daß der Interrupt nur während der Abarbeitung eines Sprungbefehles mit gelöschtem Flag F bearbeitet wird. Damit während der Abarbeitung eines Interrupts kein weiterer Interrupt ausgelöst wird, muß ein Interrupt Enable Flag (IE) hinzugefügt werden. Da es keinen Befehl zum Setzen oder Löschen des IE-Flags gibt, soll dies durch den Zugriff auf eine bestimmte Speicheradresse erfolgen: lesen der Speicherzelle \$4000 setzt das IE-Flag und schreiben der Speicherzelle \$4000 löscht das IE-Flag.

Ändern Sie das Zustandsübergangsdiagramm des MPROZ so ab, daß es nun auch Interrupts unterstützt. Führen Sie dazu so wenig neue Zustände wie möglich ein.

Geben Sie die neue Schaltung ein und simulieren Sie diese (siehe Anhang C).

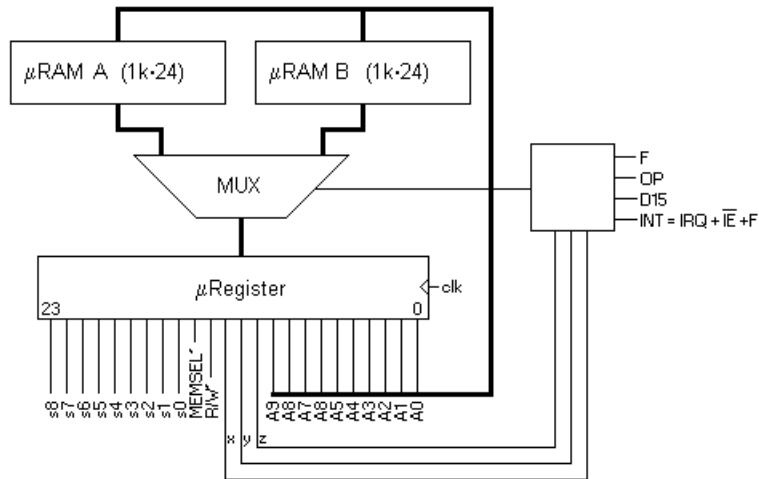
- Um mit MPROZ einfache Ein-/Ausgaben machen zu können ohne zusätzliche I/O-Hardware über den I/O-Adreßraum anzuschließen, soll in MPROZ ein 8-Bit Ein- und ein 8-Bit Ausgangsport integriert werden. Diese Leitungen sollen über die Adresse \$7fff angesprochen werden. Das niederwertige Byte entspricht dabei den Ausgangsleitungen und das höherwertige Byte den Eingangsleitungen. Erweitern Sie die Schaltung von MPROZ entsprechend und simulieren Sie die Schaltung (siehe Anhang C).



Rechnerorganisation II

4. Übungsblatt

Das Steuerwerk des Minimalprozessor MPROZ soll nun durch eine mikroprogrammierte Steuerung ersetzt werden:



Die drei Bits x,y,z bestimmen dabei, ob das Mikrowort A oder B in das Mikrobefehlsregister übernommen wird:

x	y	z	
0	-	0	A
0	-	1	B
1	0	0	A falls F=0 B falls F=1
1	0	1	A falls OP=0 B falls OP=1
1	1	0	A falls D15=0 B falls D15=1
1	1	1	A falls INT=0 B falls INT=1

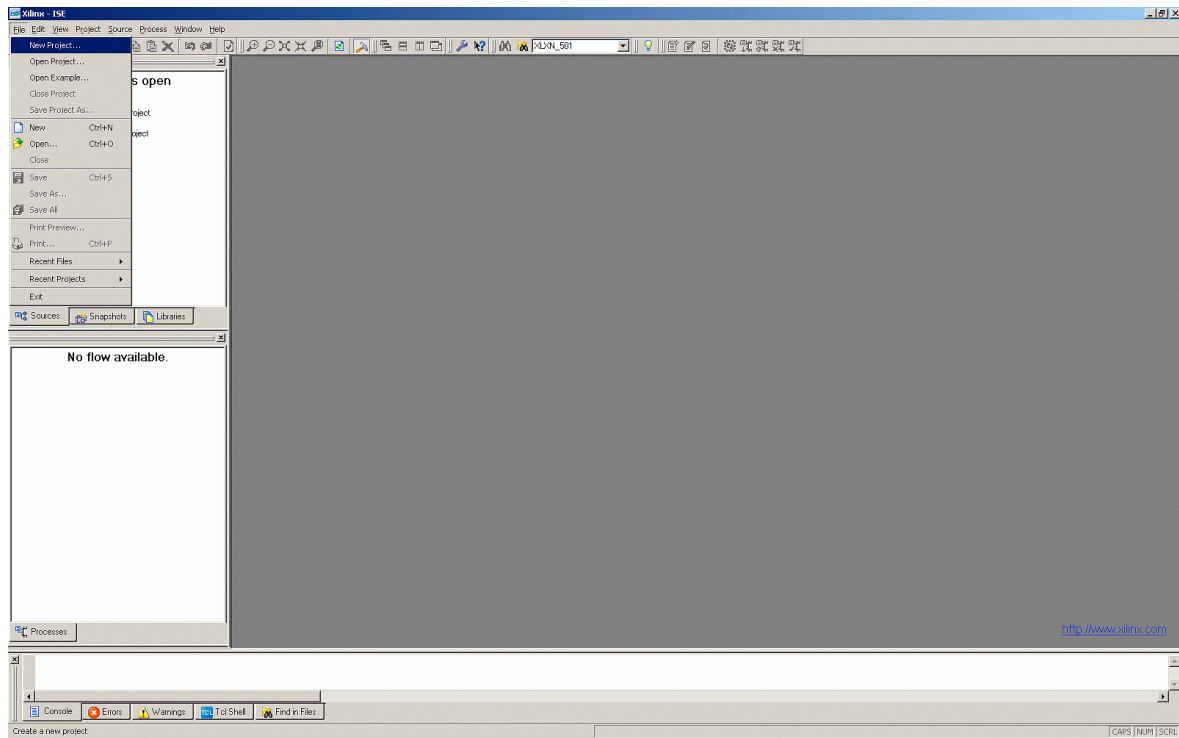
Erstellen Sie das Mikroprogramm und simulieren Sie die Schaltung (siehe Anhang D).

Anhang A

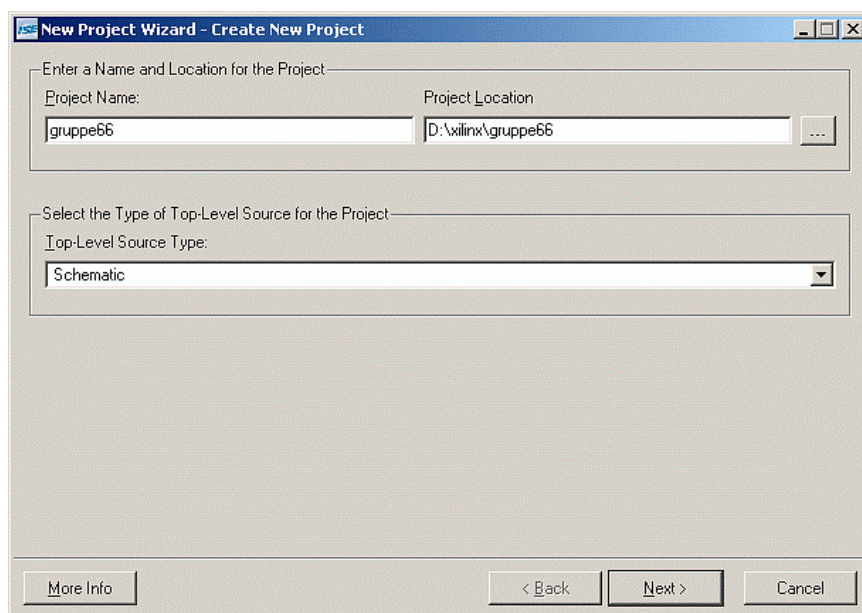
Falls noch nicht geschehen, extrahieren Sie den Anhang (mproz.zip) dieser pdf Datei und entzippen Sie ihn in einen temporären Order. Sie erhalten dann die beiden Order "import" und "addon".

Falls die Xilinx Software noch nicht installiert ist, laden Sie das kostenlose Webpack von Xilinx's Webserver herunter (http://www.xilinx.com/ise/logic_design_prod/webpack.htm) und installieren es. Diese Dokumentation wurde mit Version 9.1 erstellt.

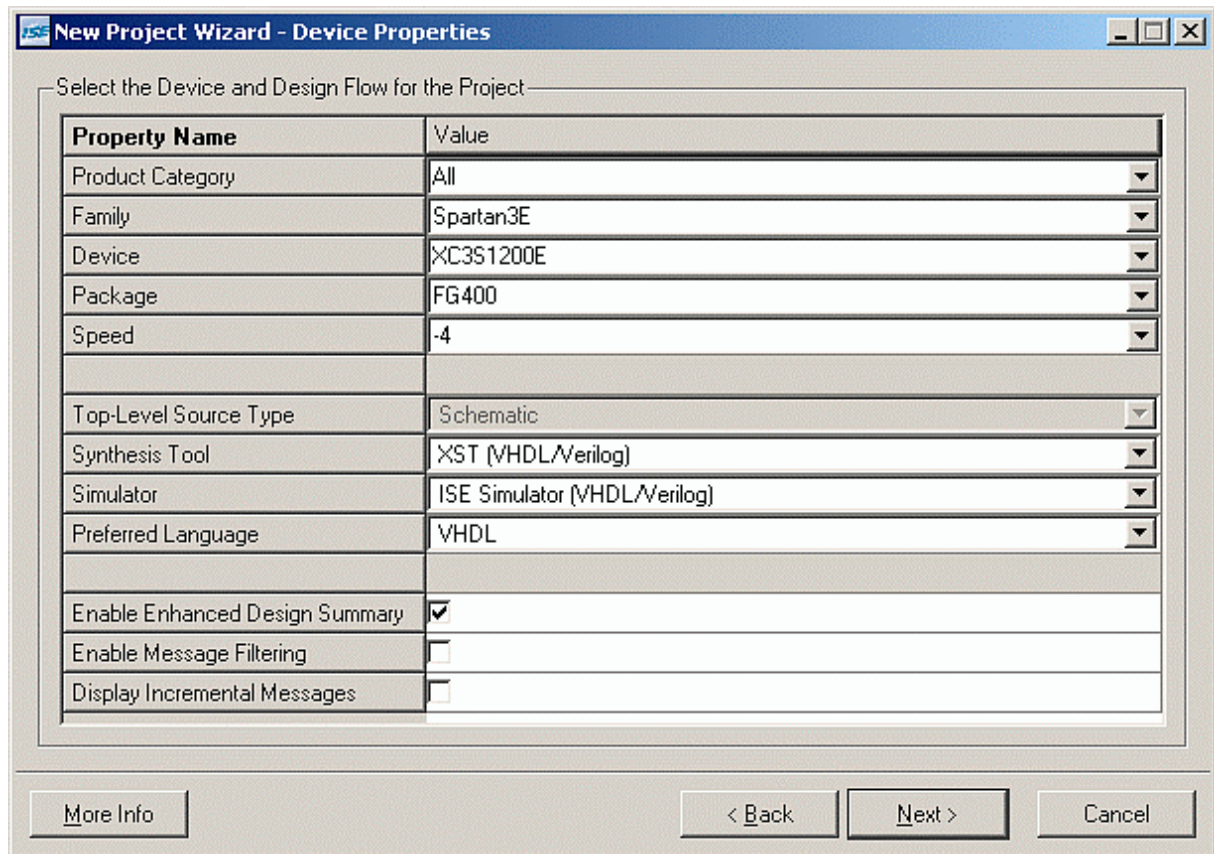
Starten Sie das Programm und öffnen Sie das "File" Menü:



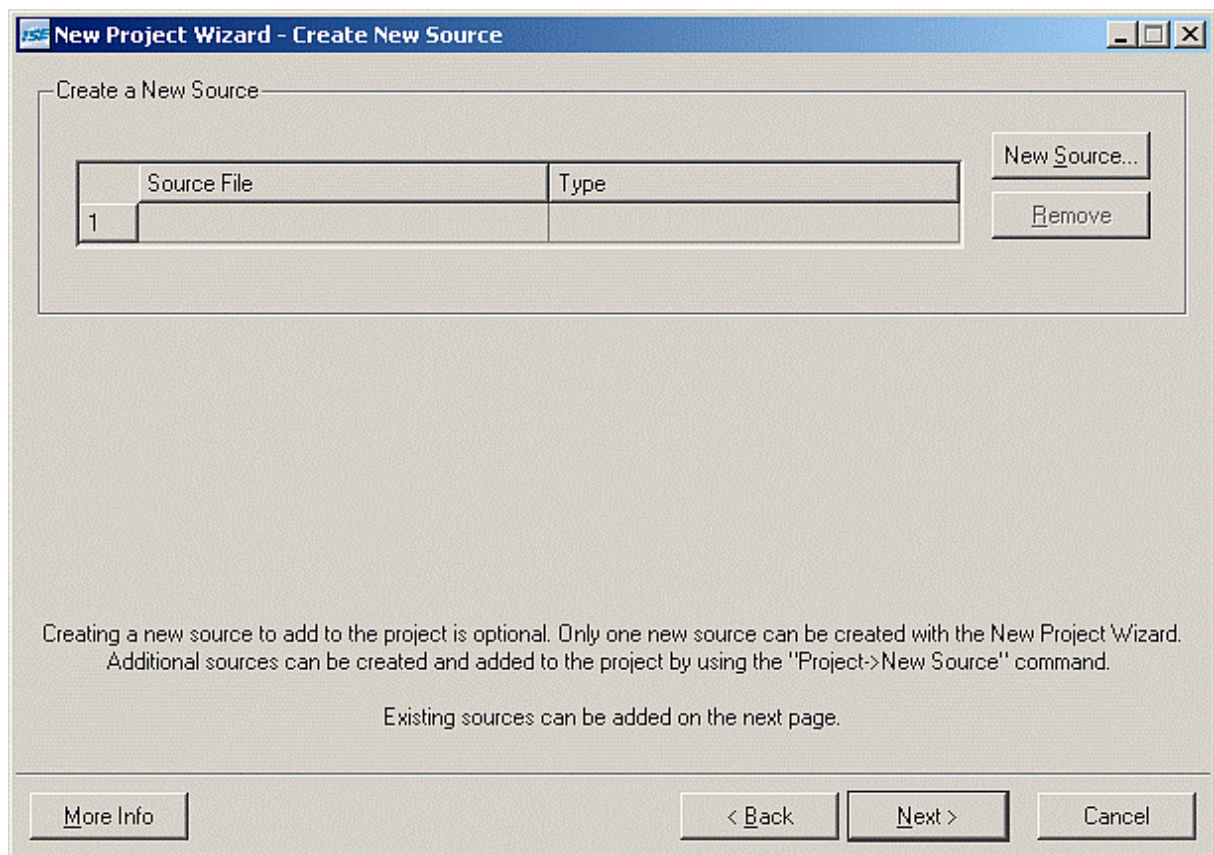
Machen Sie einen Linksklick auf "New Project..."



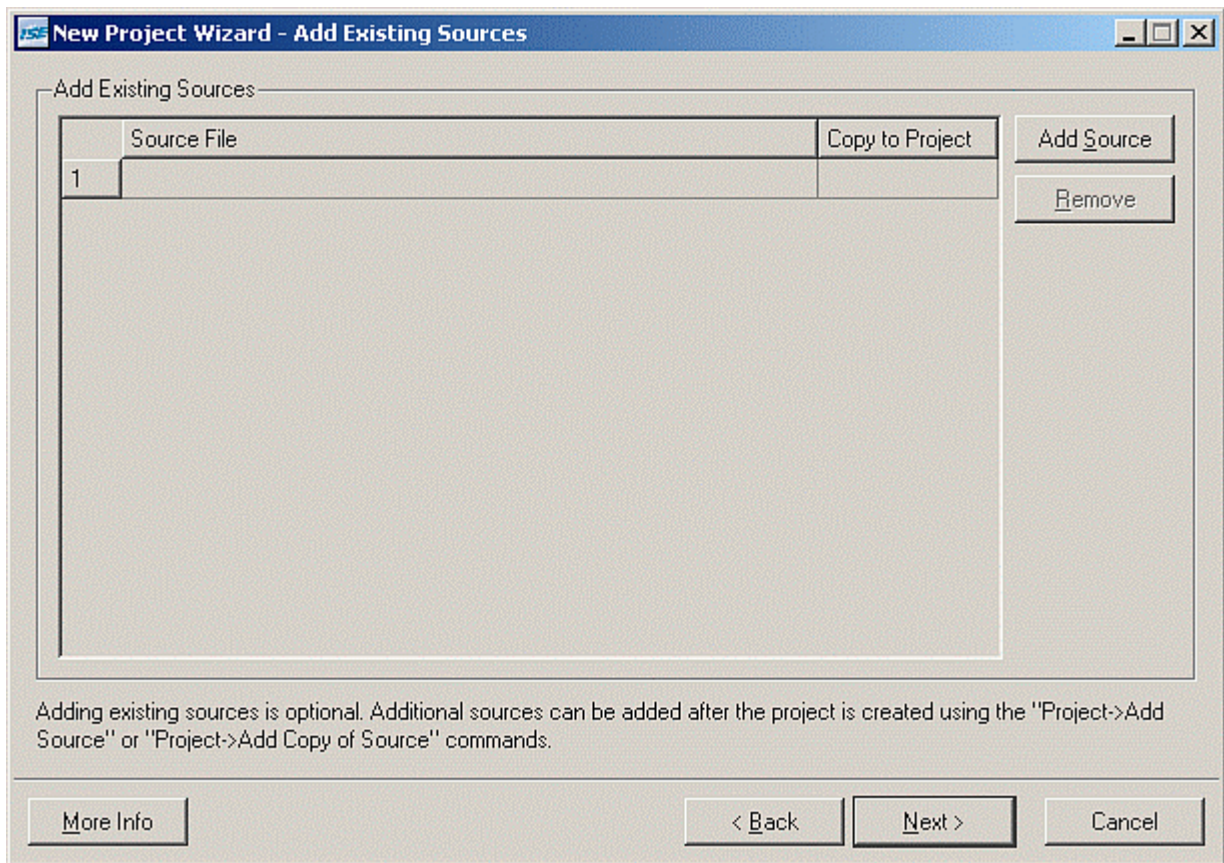
Geben Sie als Projekt Name "gruppexx" (xx= ihre Gruppennummer) ein und klicken Sie dann auf "Next".



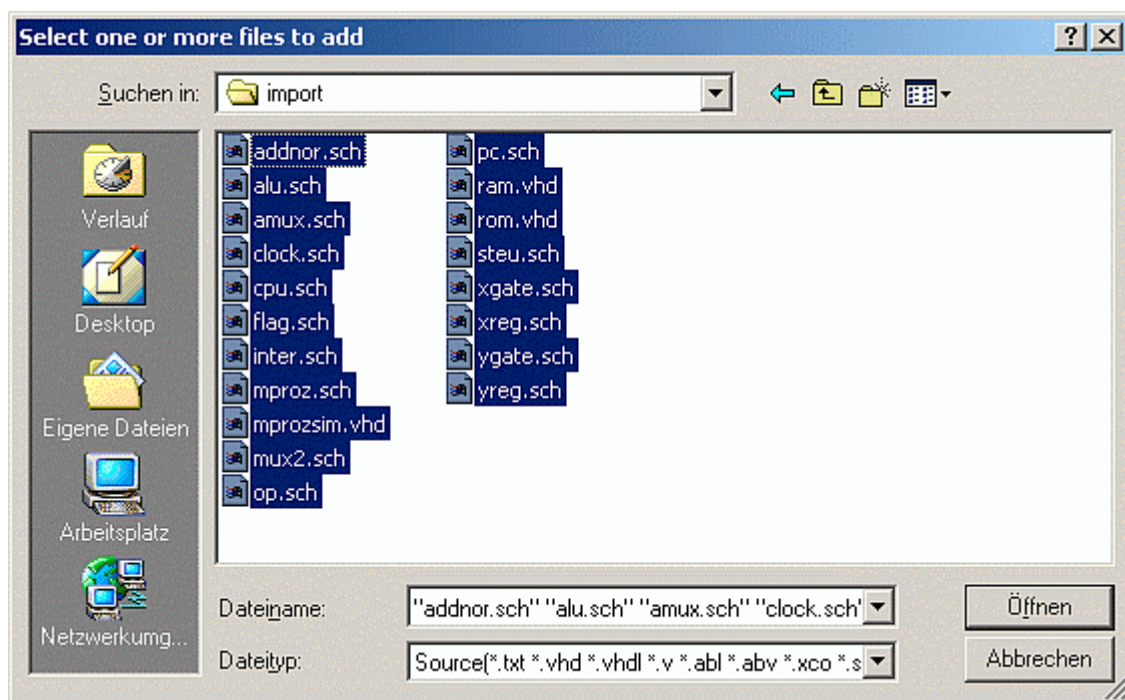
Obige Eingaben auswählen und dann auf "Next" klicken.



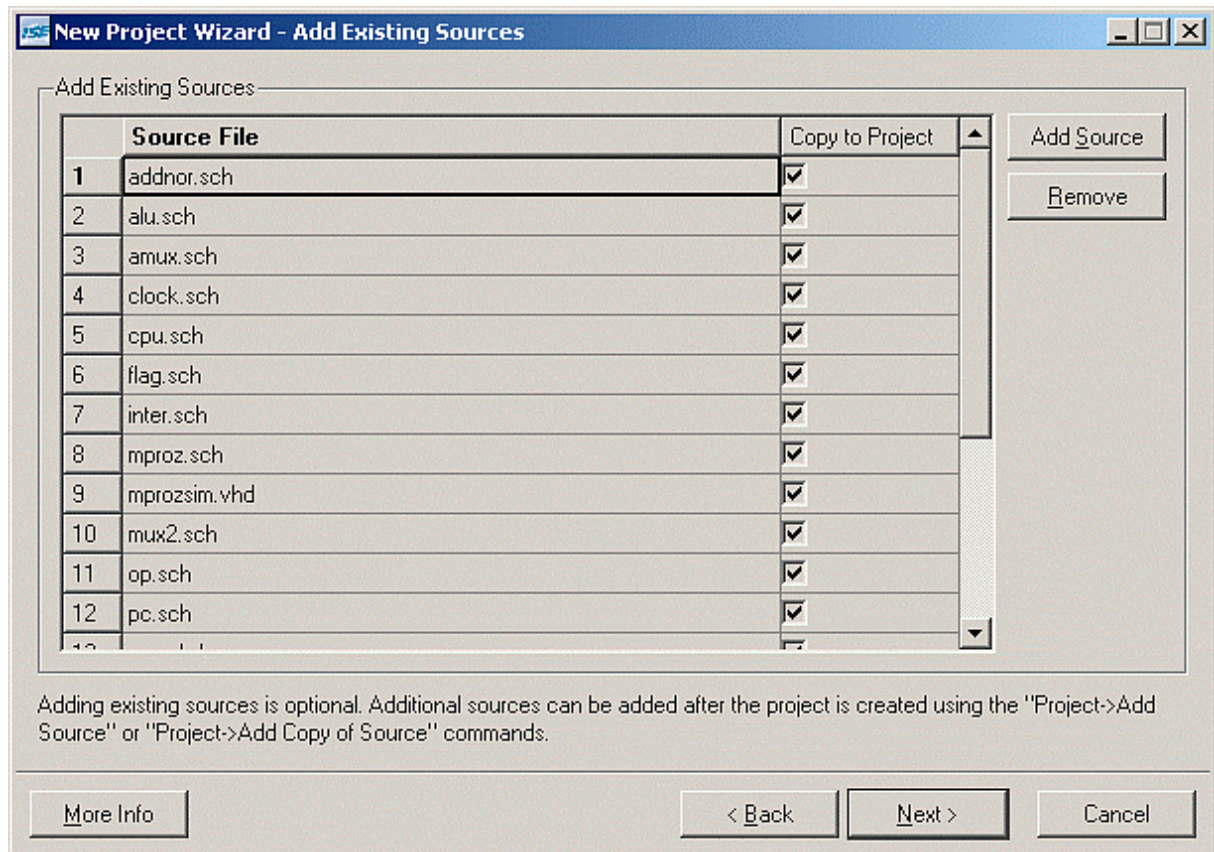
Keine "New Source" generieren, nur auf "Next" klicken.



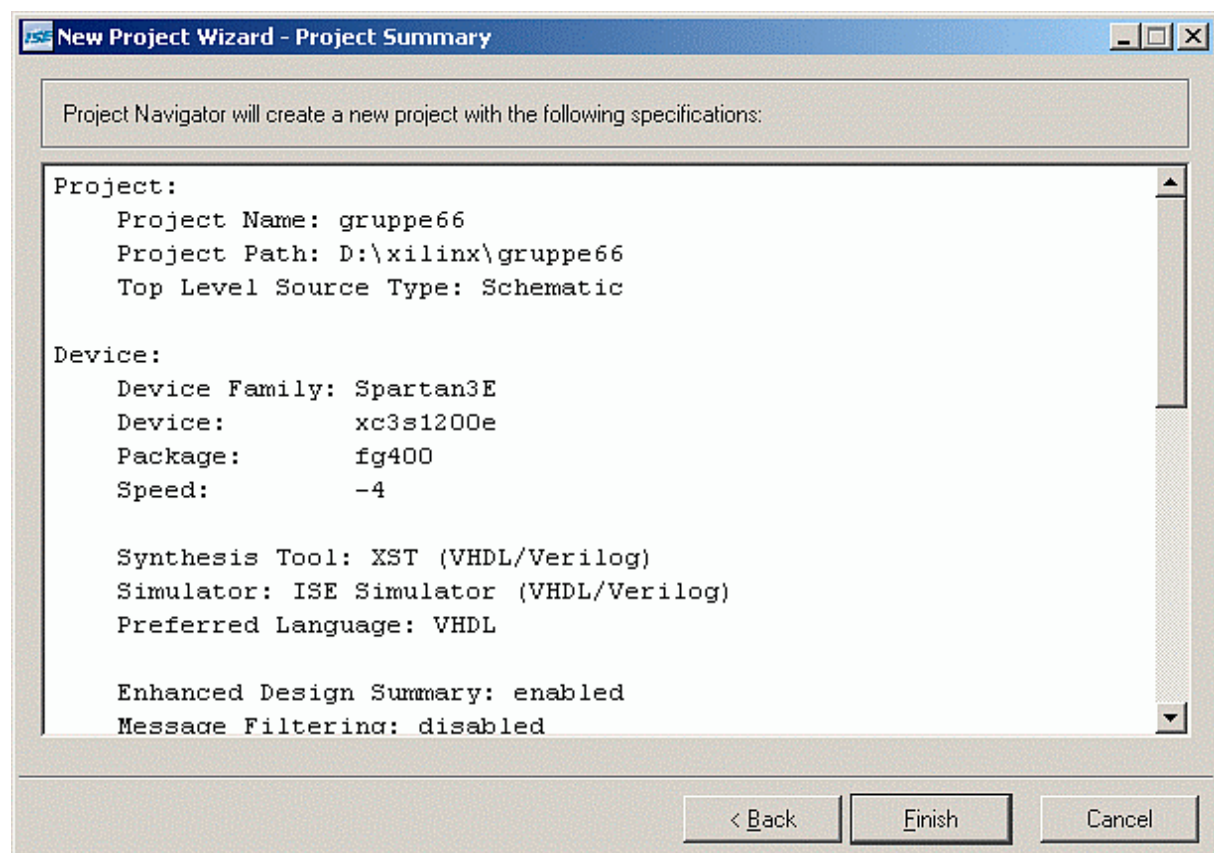
"Add Source" anklicken.



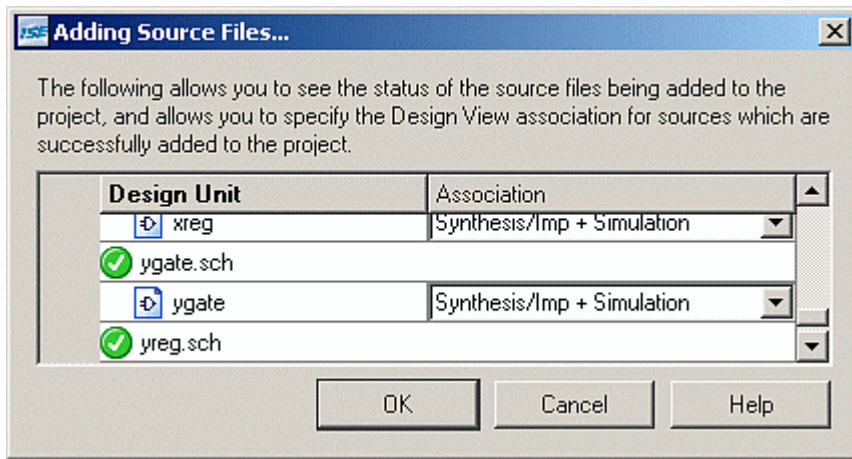
Alle Dateien im "imort" Ordner (aus mproz.zip) auswählen und dann auf "Öffnen" klicken.



"Next" anlicken.

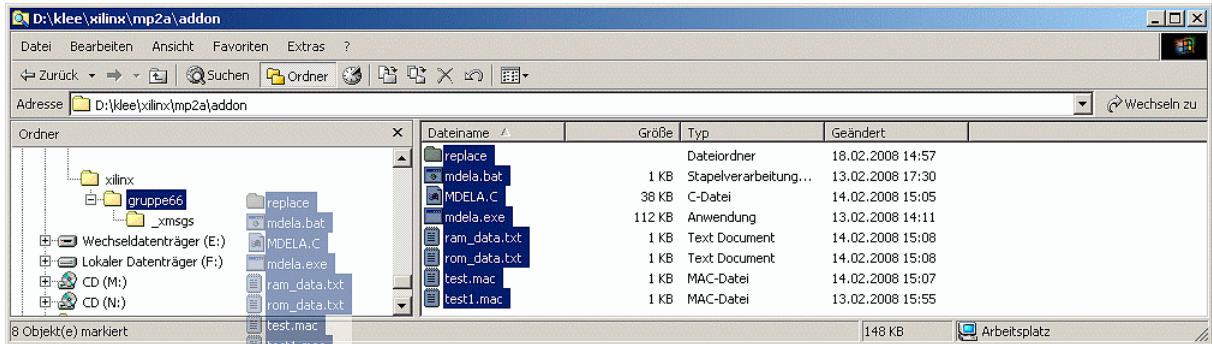


"Finish" anlicken.

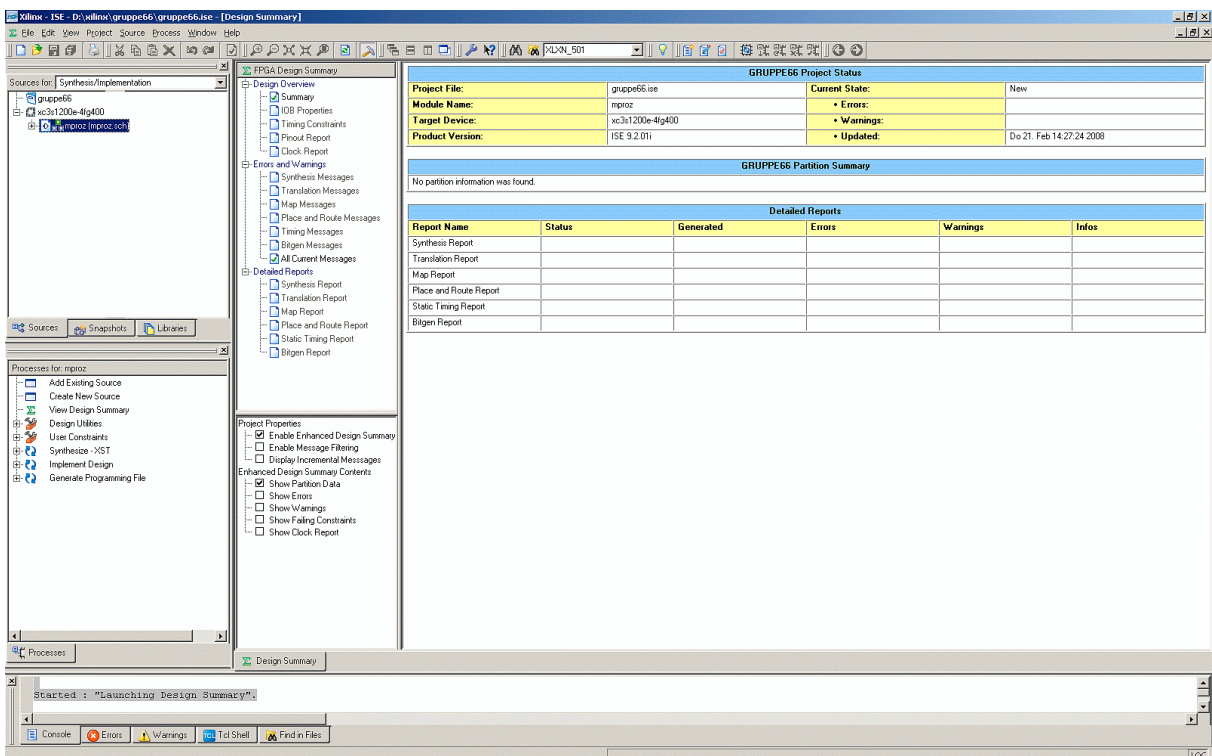


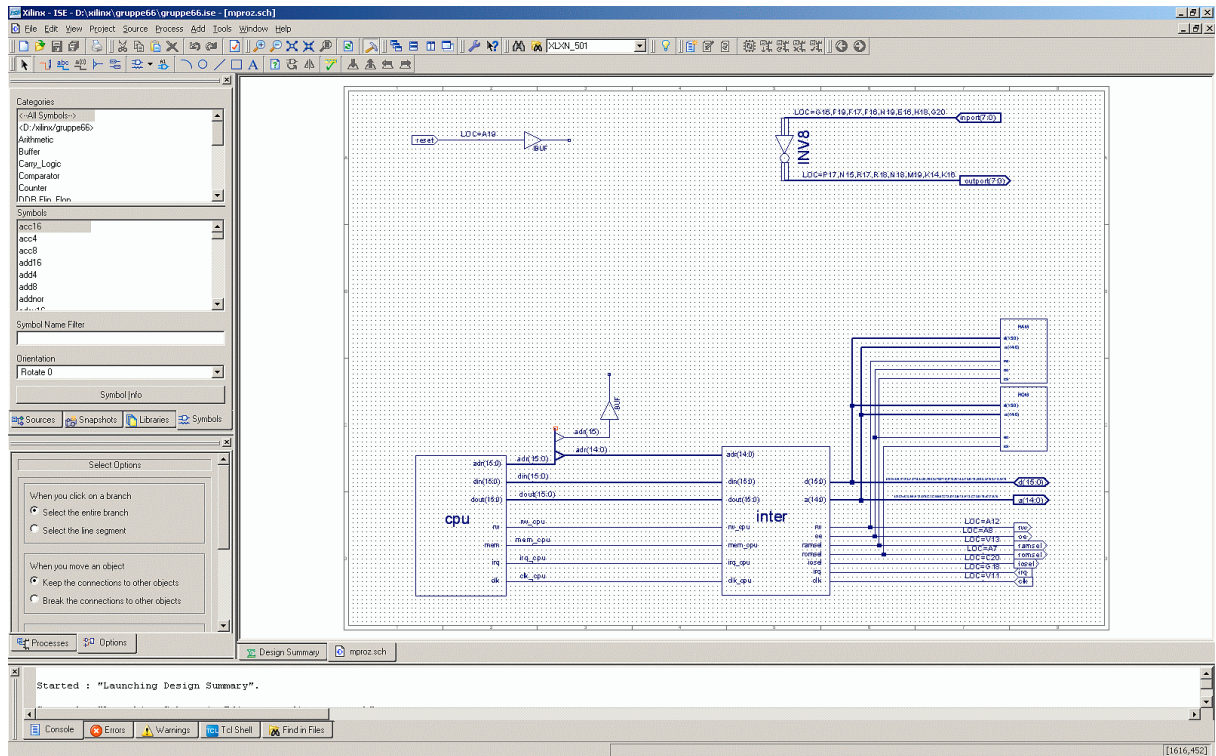
Warten Sie, bis alle Dateien importiert sind und klicken Sie dann auf "OK".

Kopieren Sie nun alle Dateien aus dem Ordner "addon" (aus mproz.zip) in ihr Projekt-Directory.



Doppelklicken sie auf "mproz.sch":



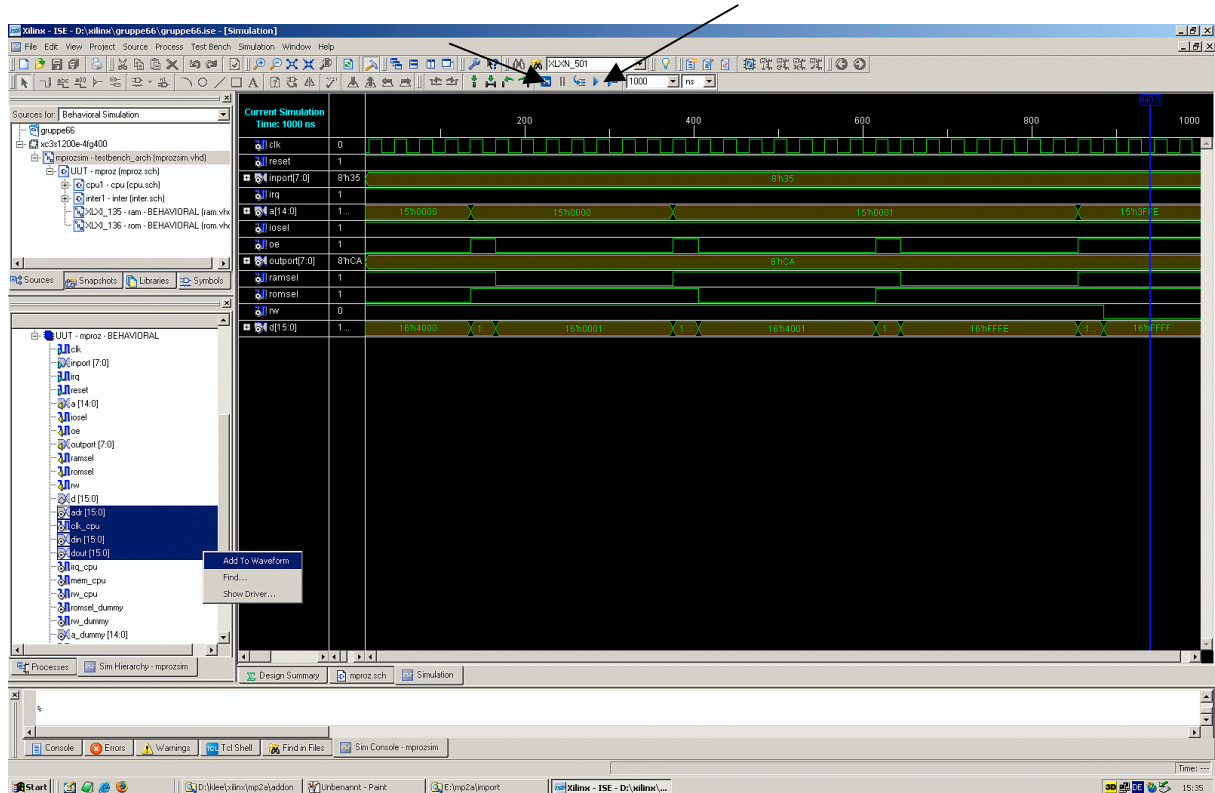


Sie sehen nun die oberste Schaltplanebene des Prozessors. Vervollständigen sie die Schaltpläne in folgender Reihenfolge:

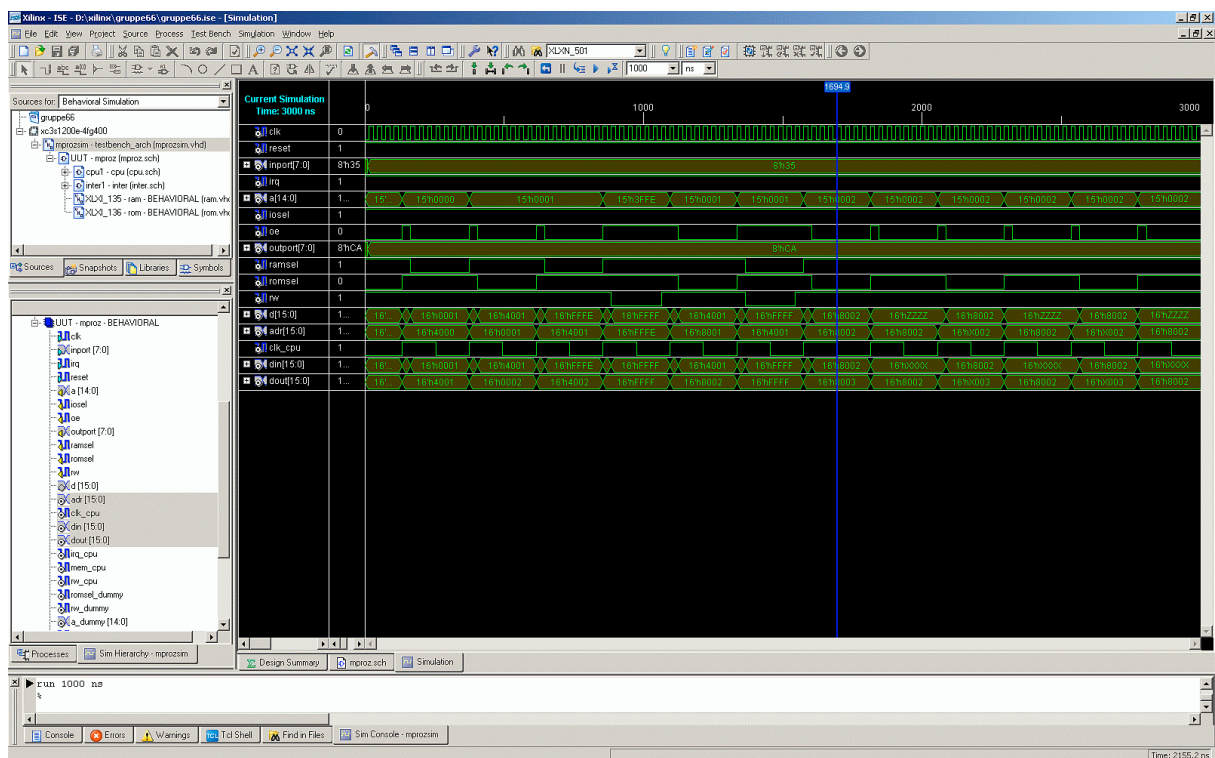
- cpu/op
- cpu/flag
- cpu/xreg
- cpu/yreg
- cpu/pc
- cpu/ygate
- cpu/xgate
- cpu/amux
- cpu/amux/mux2
- cpu/alu
- cpu/alu/addnor
- cpu/steu
- inter

Verwenden Sie dazu ausschließlich einfache Gatter (inv, and, or, nand, nor, xor usw.) mit nicht mehr als fünf (eventuell. negierten) Eingängen und D-FlipFlops (mit und ohne Enable Eingang).

Fügen Sie folgende Signale durch einen Rechtsklick auf die Signalnamen hinzu:
 clk_cpu, adr[15:0], din[15:0] und dout[15:0]

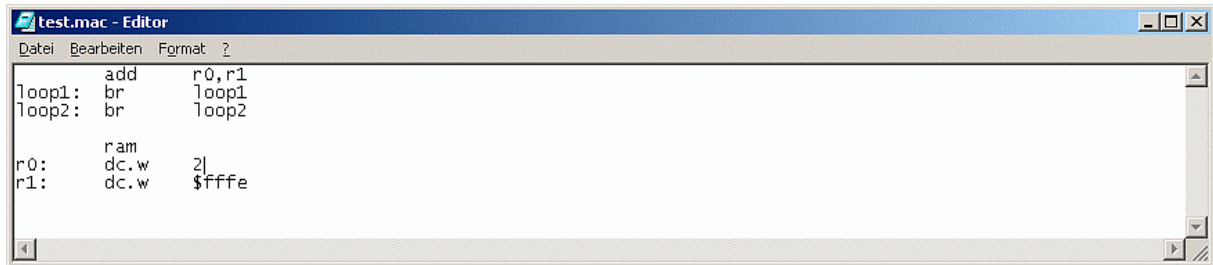


Setzen Sie die Simulation zurück und simulieren Sie ca. 3000 ns:



Sie sehen nun die Simulation des "add" und "br" Befehls (Programm siehe nächste Seite). Da bei der Addition kein Carry erzeugt wurde befindet sich der Prozessor in "loop1". Falls der Prozessor nicht korrekt arbeitet, fügen Sie weiter Signale (z.B. Steuersignale s0-s8, Zustandsignale q0-q7) hinzu, um den Fehler leichter zu finden.

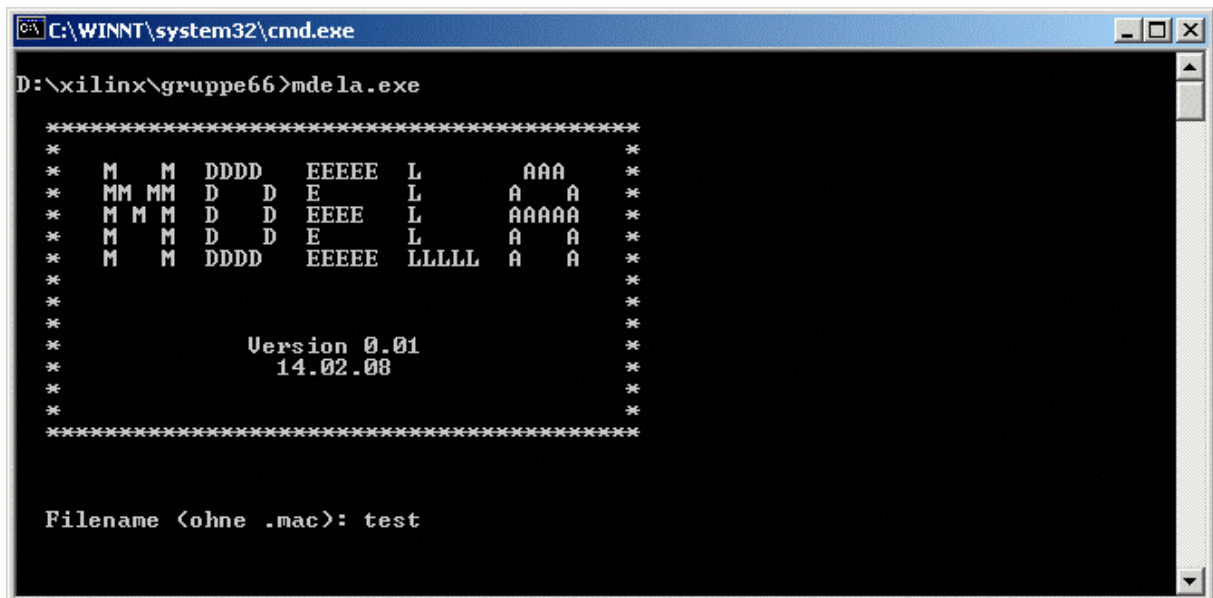
Editieren Sie nun den Quellcode des Programms "test.mac" und ändern den Inhalt von r0 von 1 auf 2. (Siehe Anhang E für eine Beschreibung des Assemblers).



```
test.mac - Editor
Datei Bearbeiten Format ?
loop1: add r0,r1
loop1: br loop1
loop2: br loop2

r0: ram
r0: dc.w 2
r1: dc.w $fffe
```

Speichern Sie die Datei ab und starten MDELA (entweder durch Eingabe von MDELA.EXE in einem CMD Fenster oder durch einen Doppelclick auf die Datei MDELA.BAT im Windows Explorer:



```
C:\WINNT\system32\cmd.exe
D:\xilinx\gruppe66>mde la.exe

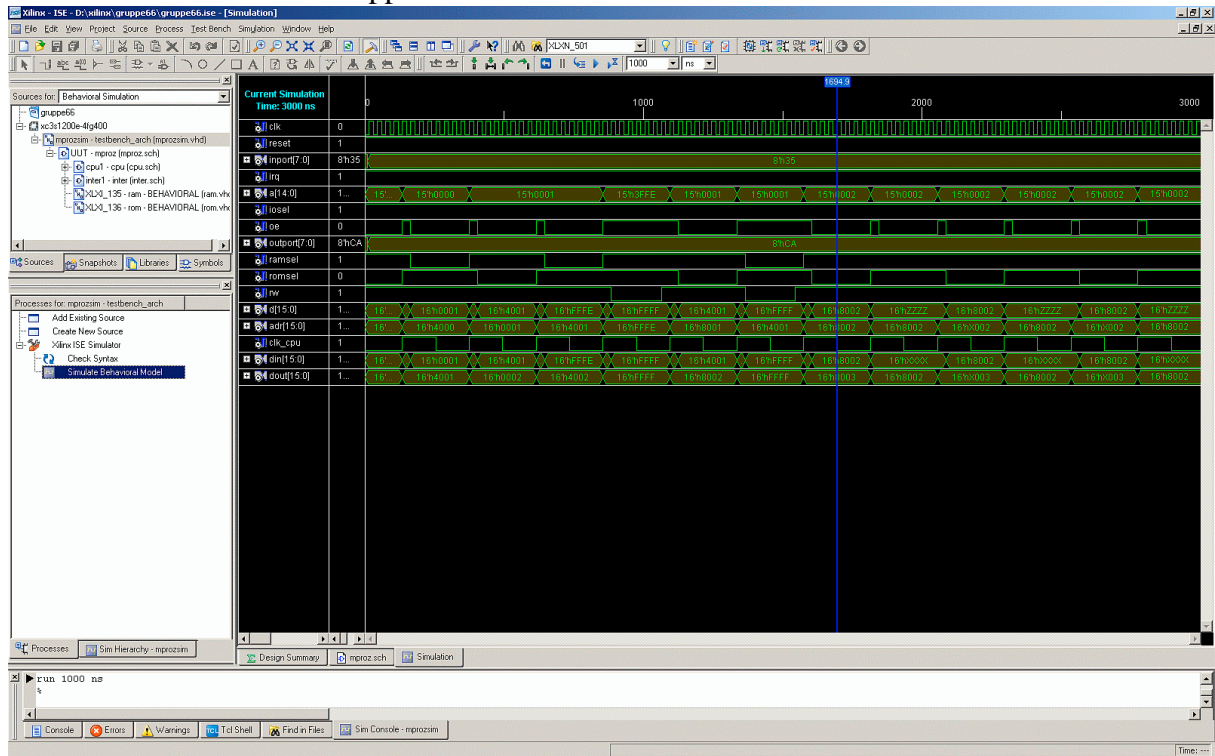
*****
*                                     *
*  M  M  DDDD  EEEEE  L    AAA      *
* MM MM  D  D  E    L    A  A      *
* M M M  D  D  EEEE  L    AAAAA     *
* M  M  D  D  E    L    A  A      *
* M  M  DDDD  EEEEE  LLLLL  A  A    *
*                                     *
*                                     *
*                                     *
*          Version 0.01              *
*          14.02.08                  *
*                                     *
*                                     *
*****

Filename (ohne .mac): test
```

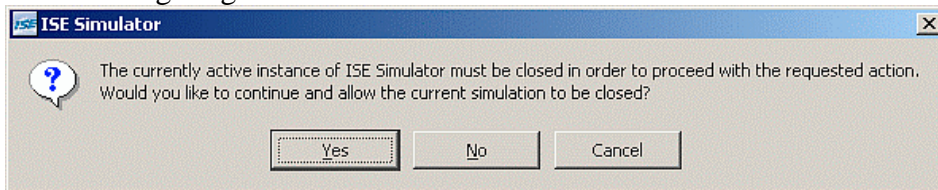
Geben Sie den Programmnamen "test" ein und drücken die Return Taste.

MDELA erzeugt u.a. die Dateien "ram_data.txt" und "rom_data.txt" aus denen der Simulator die Speicherinhalte liest.

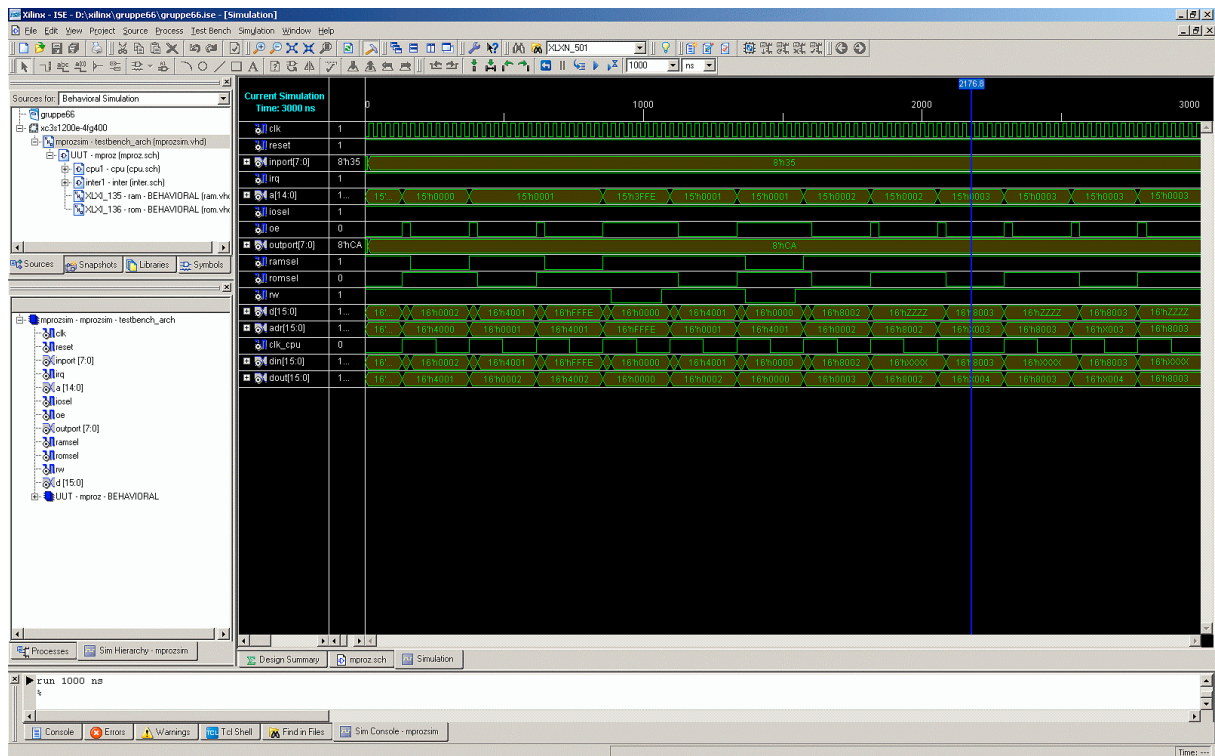
Machen Sie erneut einen Doppelclick auf "Simulate Behavioral Model"



Sie werden gefragt:



Clicken Sie "Yes".

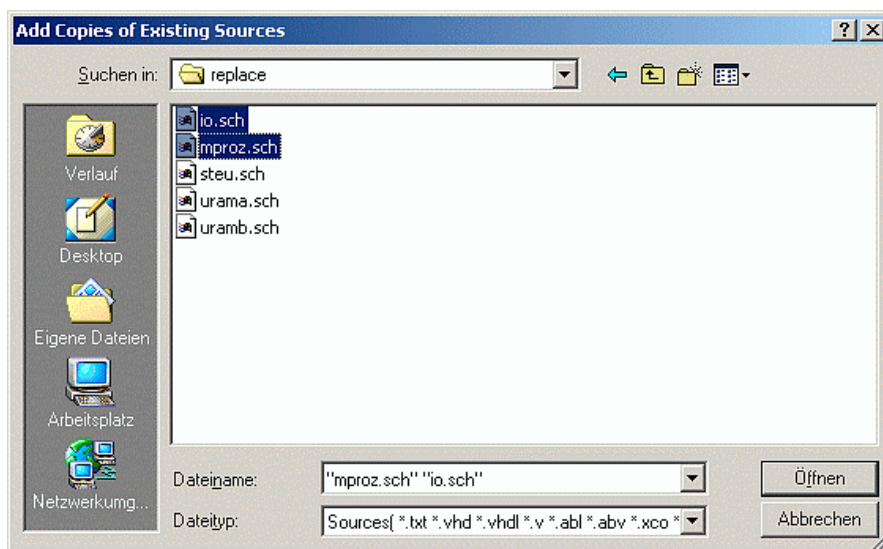
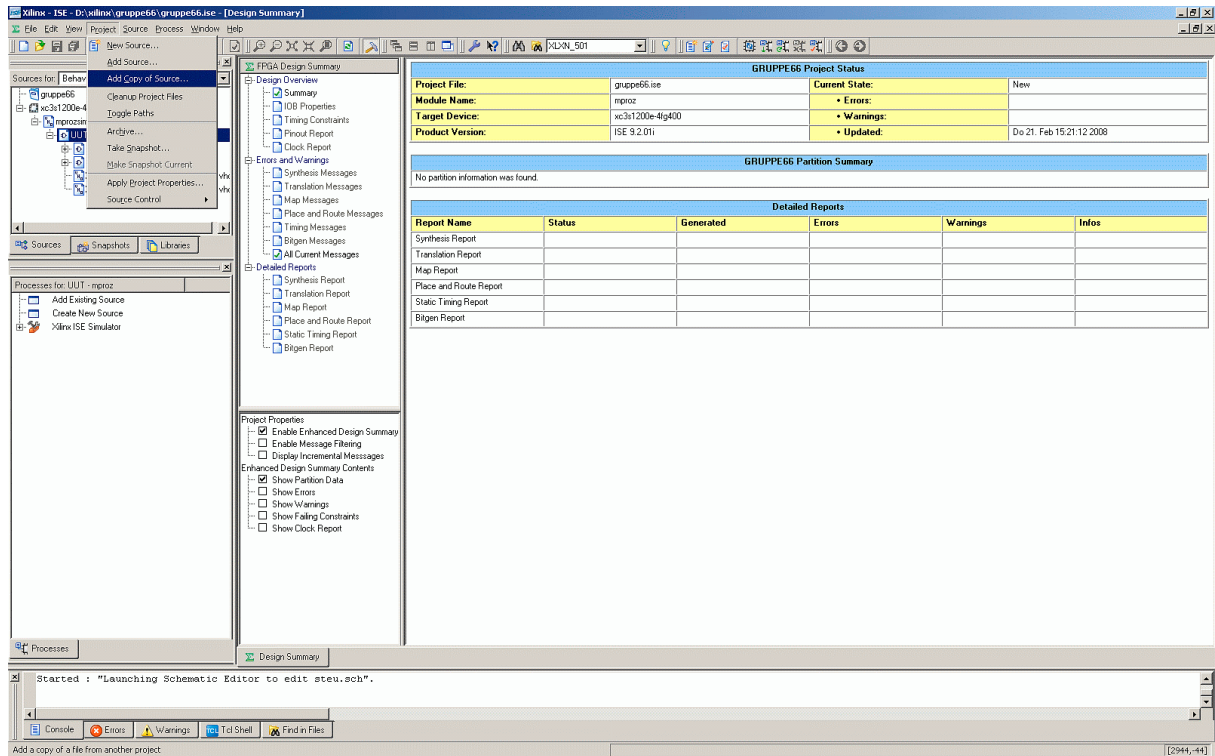


Da nun bei der Addition ein Carry generiert wird, befindet sich der Prozessor in "loop2". Testen Sie auch den nor Befehl (mit und ohne Setzen des Flags).

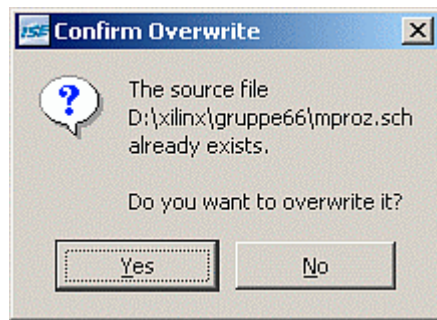
Anhang C

Modifizieren Sie nun das Steuerwerk so, daß es auch Interrupts unterstützt. Ändern Sie das Assemblerprogramm ebenfalls entsprechend ab und testen Sie die korrekte Funktion durch eine erneute Simulation.

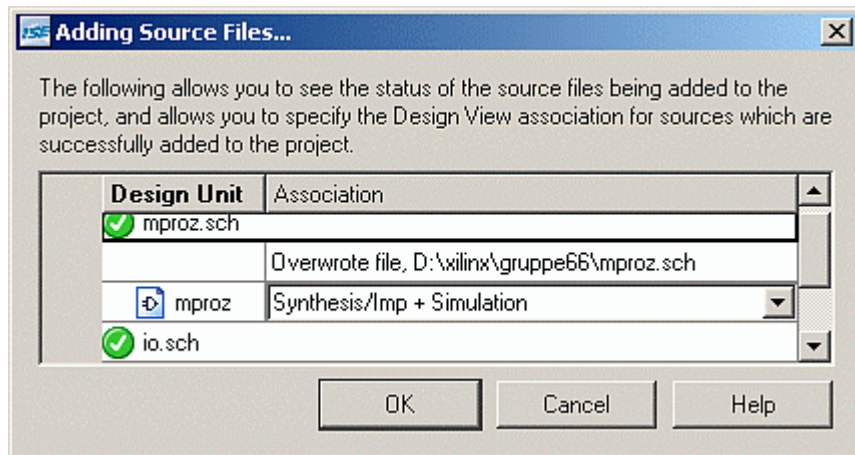
Schließen Sie nun alle Schaltbild- und Simulationsfenster und wählen aus dem Menü "Add Copy of Source":



Selektieren Sie die beiden Dateien io.sch und mproz.sch

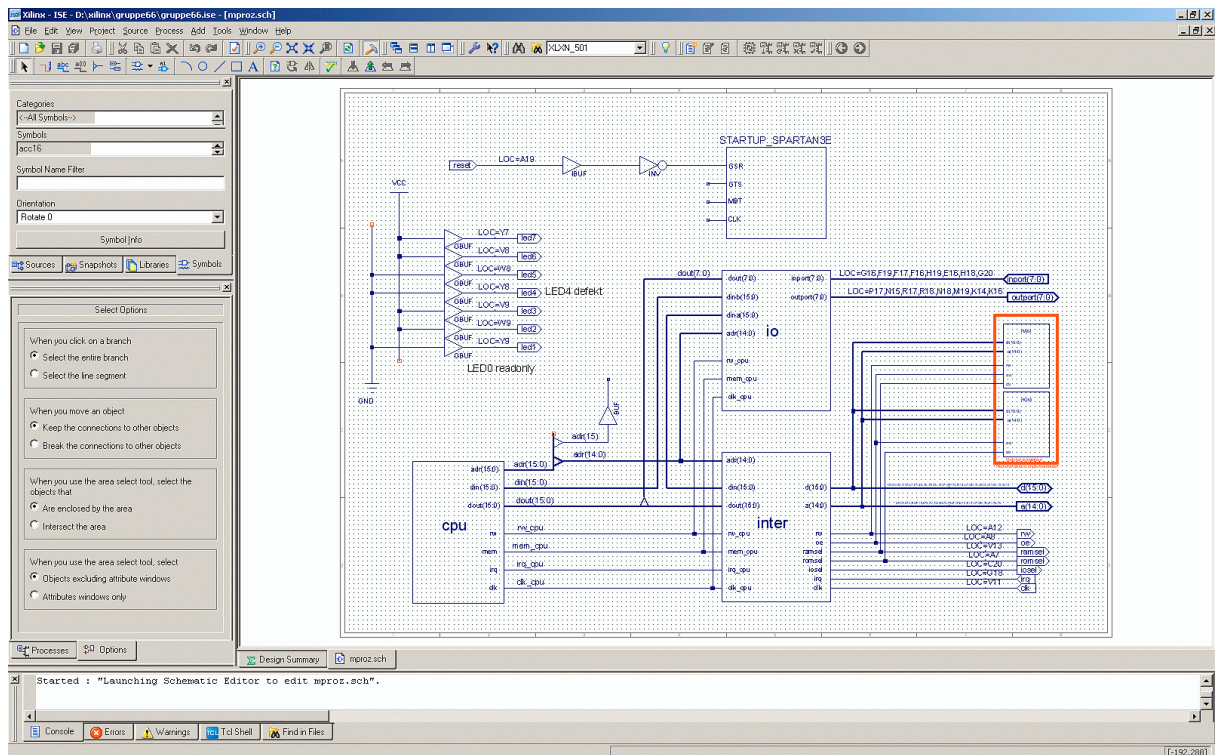


Klicken Sie auf "Yes".



Klicken Sie auf "OK".

Öffnen Sie nun erneut "mproz.sch" und vervollständigen die Schaltung für "io". Überprüfen Sie die korrekte Funktion durch Simulation mit einem entsprechenden Programm.

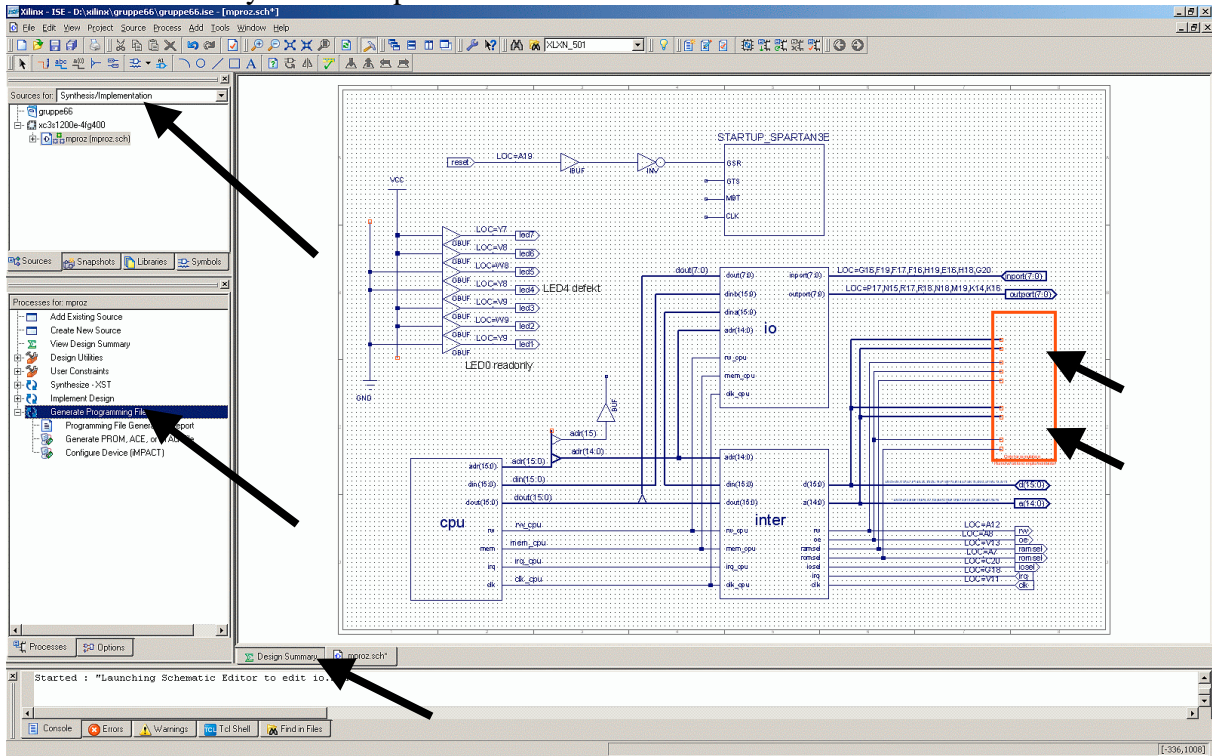


Beschalten Sie die Leuchtdioden so, daß Ihre Gruppennummer als zweistellige Oktalzahl dargestellt wird. Laden Sie nun die Schaltung wie im Anhang D beschrieben auf das Demoboard.

Anhang D

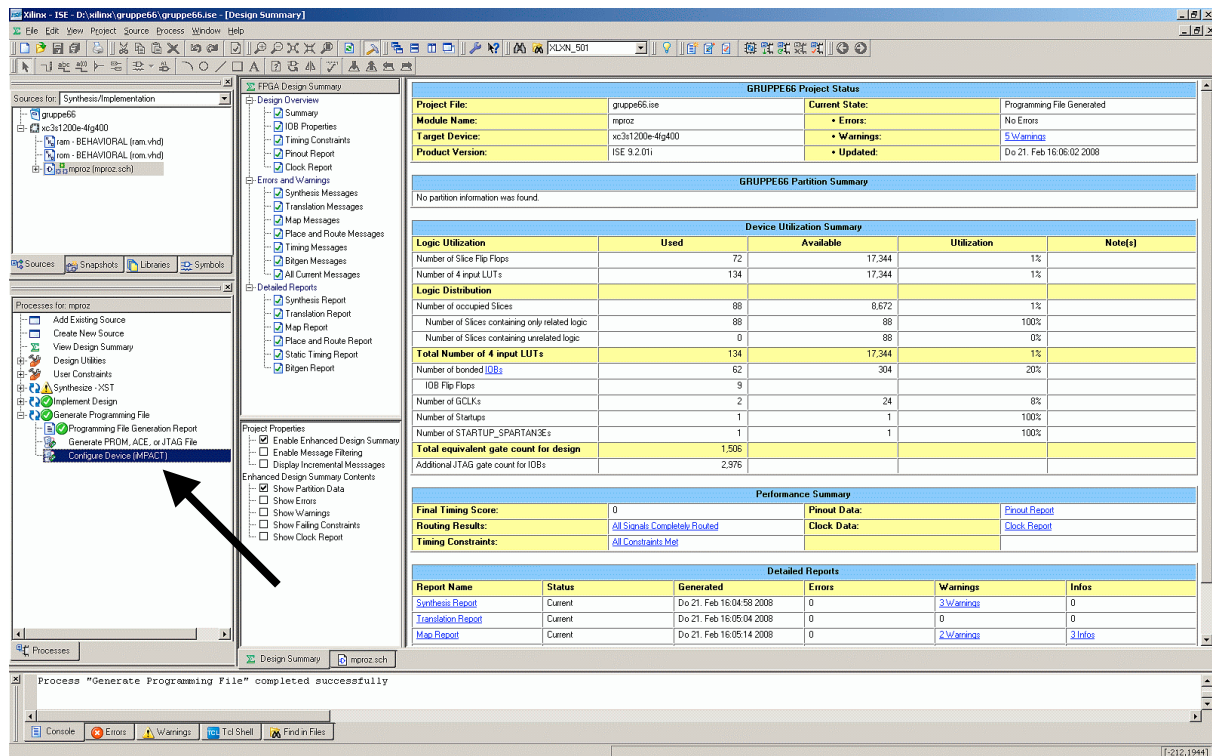
Download der Schaltung auf das Demoboard:.

Selektieren Sie "Synthesis/Implementation" anstatt von "Behavioral Simulation":

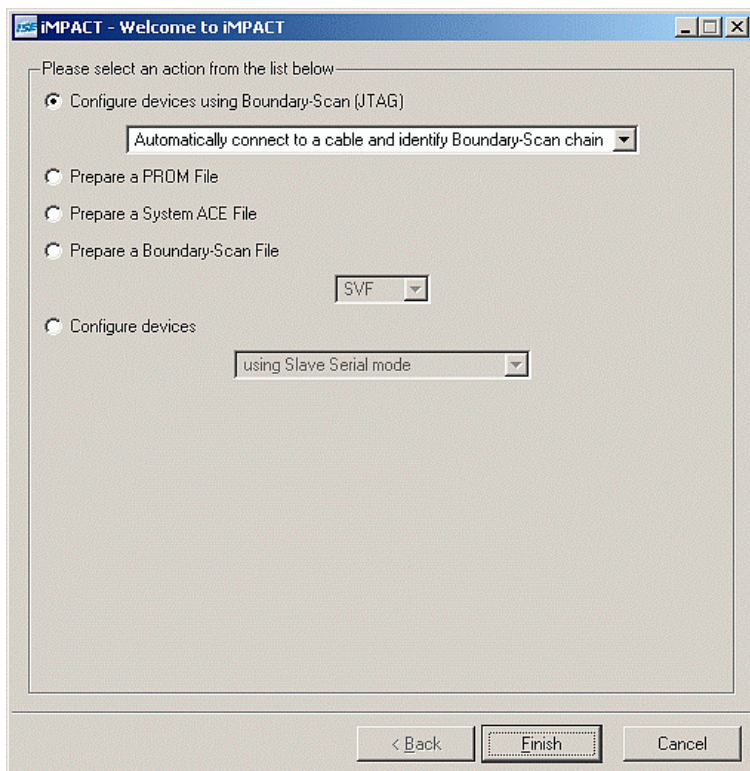


Entfernen Sie RAM und ROM und doppelklicken Sie auf "Generate Programming File":

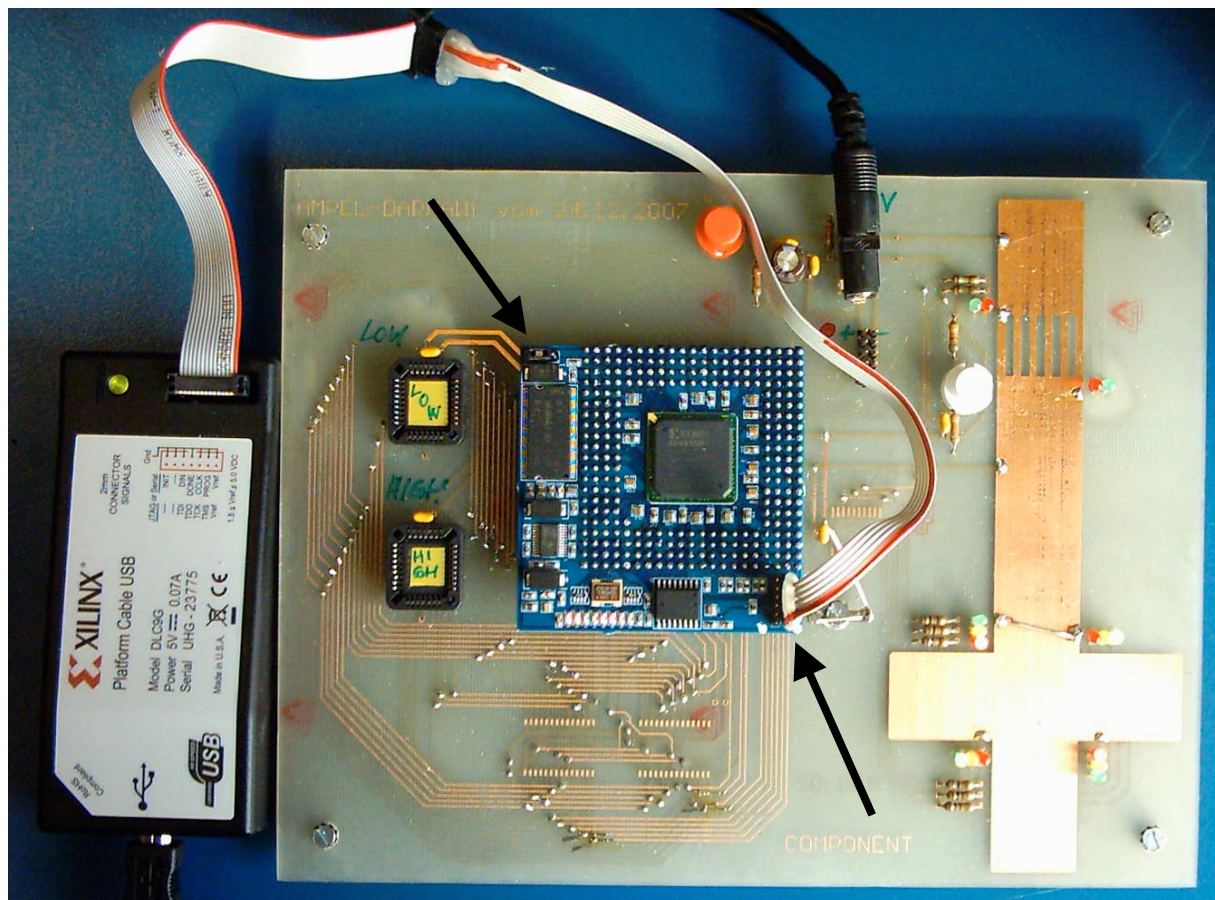
Selektieren Sie den Reiter "Design Summary". Sie sollten sehen::



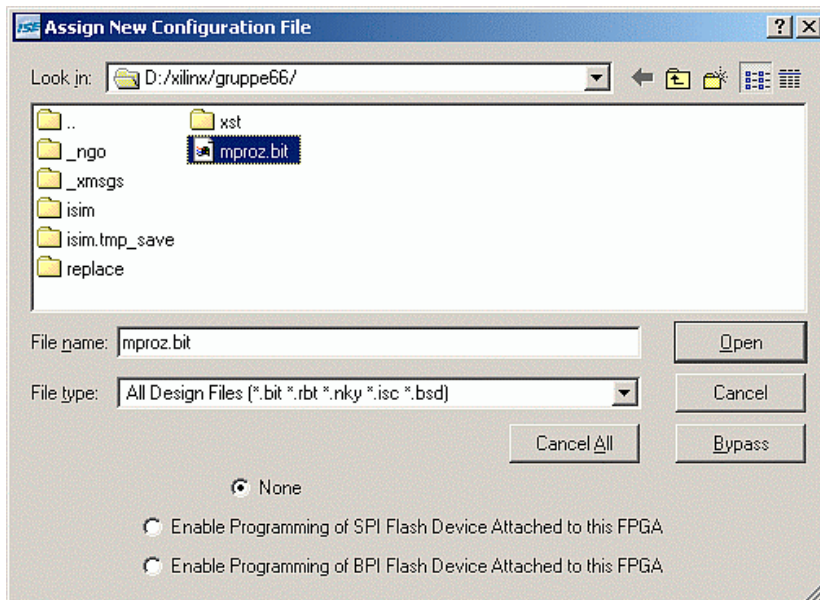
Doppelklicken Sie auf "Configure Device (iMPACT)".



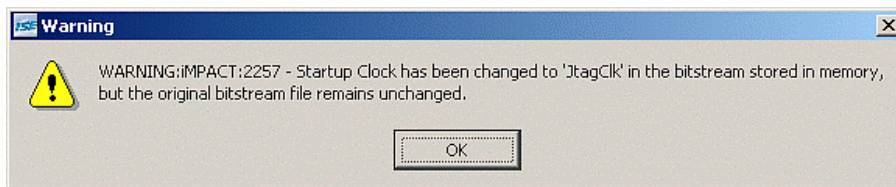
Verbinden Sie das Download-Kabel mit dem Demoboard und klicken "Finish".



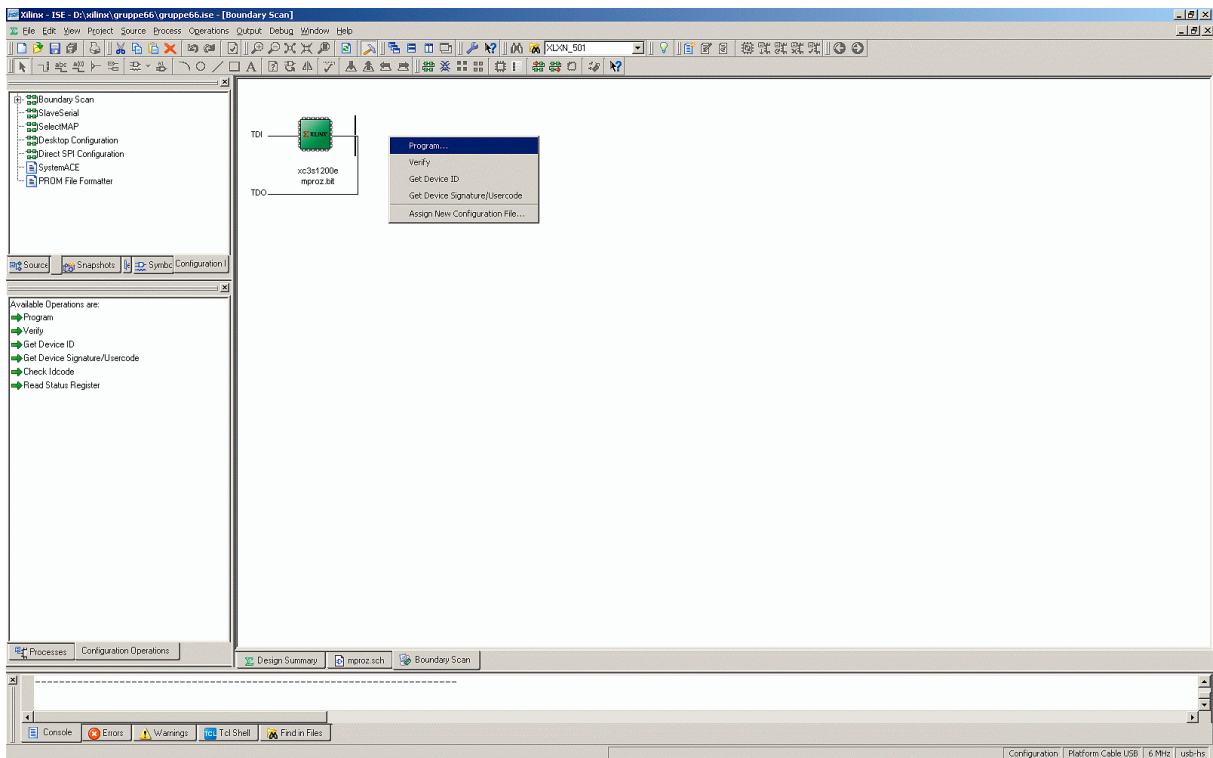
Verbinden Sie das Kabel mit der äußeren Stiftreihe und entfernen den Jumper.



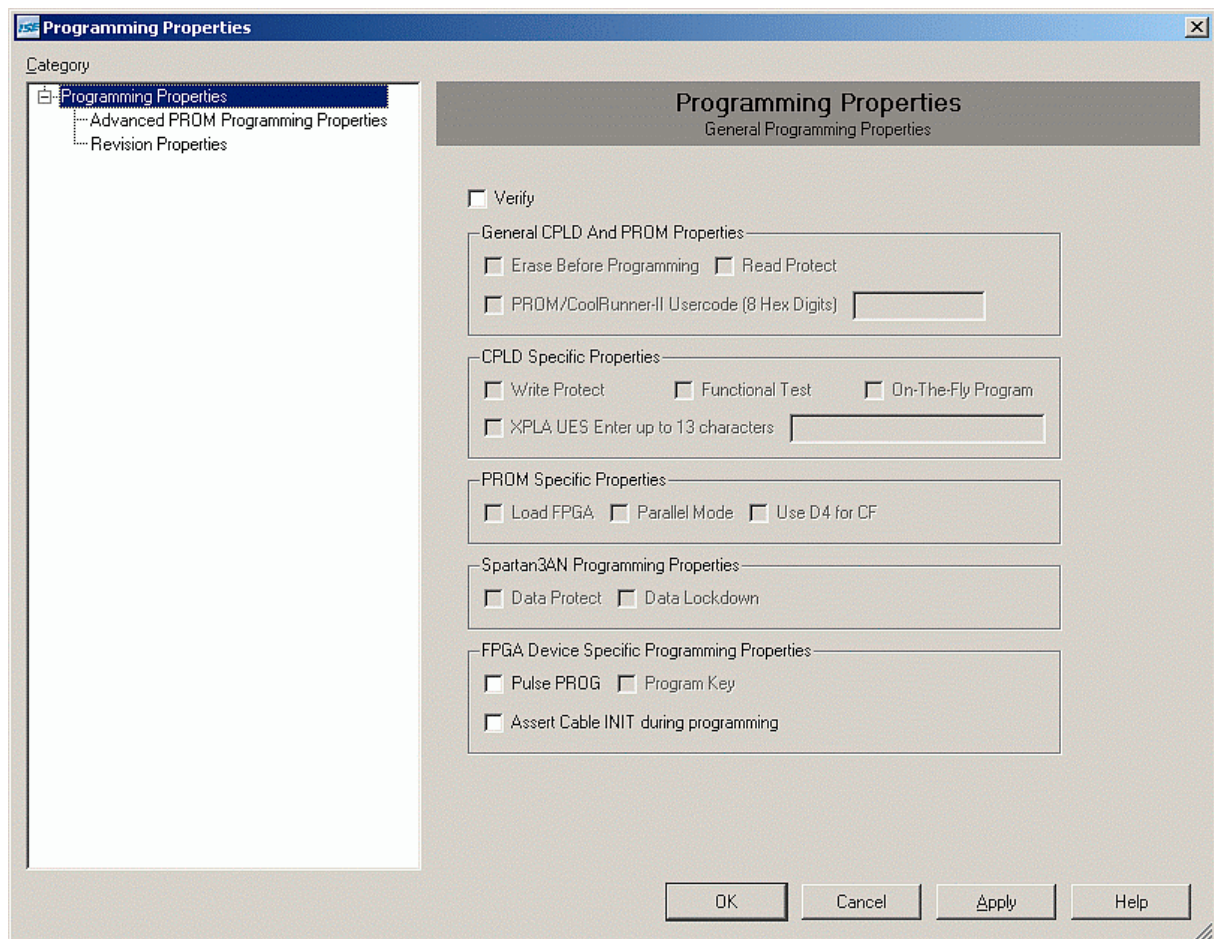
Selektieren Sie "mproz.bit" und klicken "Open"



Clicken Sie OK



Machen Sie eine Rechtsklick auf das FPGA Symbol und selektieren "Program..."



Deselektieren Sie **"Verify"** und klicken dann auf **"OK"**.

Die Ampelsteuerung auf dem Demoboard sollte nun starten (überprüfen Sie die korrekte Gruppennummer auf den LEDs). Lösen Sie einen Interrupt mit weißen Taster aus.

Anhang E

Assembler MDELA

MDELA ist ein sehr einfacher Assembler. Nach dem Start fragt er nach dem Dateinamen der Quelltextdatei (nur Name, die Erweiterung muß immer .mac sein). Der Assembler generiert folgende Ausgabefiles:

```
filename.rom    Binärcode für ROM
filename.ram    Binärcode für RAM
filename.lst    Listing
rom_data.txt    ASCII version von filename.rom für den Simulator
ram_data.txt    ASCII version von filename.ram für den Simulator
```

3.1 MPROZ Befehlssatz

```
br   a           springe zur Adresse a FLAG = 0
bcc  a           wie br, bessere Lesbarkeit nach einem add
bne  a           wie br, bessere Lesbarkeit nach einem nor
add  a,b         b = a+b           FLAG = carry-flag
nor  a,b         b = a nor b       FLAG = zero-flag
```

3.2 Pseudo-Opcodes:

```
dc.b  WERT1,...,WERTn : Deklarieren von Byte-Werten
dc.w  WERT1,...,WERTn : Deklarieren von Wort-Werten
dc.l  WERT1,...,WERTn : Deklarieren von Long-Werten

dc.w_ WERT1,...,WERTn : Deklarieren von Wort-Werten (big endian)
dc.l_ WERT1,...,WERTn : Deklarieren von Long-Werten (big endian)

blk.b  WERT1,WERT2 : WERT1 mal dc.b WERT2 ausfuehren
blk.w  WERT1,WERT2 : WERT1 mal dc.w WERT2 ausfuehren
blk.l  WERT1,WERT2 : WERT1 mal dc.l WERT2 ausfuehren
       : WERT1 muss bereits im PASS1 berechnet werden
       koennen

blk.b  WERT : Current Location Pointer um WERT erhoehen
blk.w  WERT : Current Location Pointer um 2*WERT erhoehen
blk.l  WERT : Current Location Pointer um 4*WERT erhoehen
       : WERT muss bereits im PASS1 berechnet werden
       koennen
```

ACHTUNG: blk.x (x=b,w,l) mit nur einem Parameter veraendert nur den Current Location Pointer @ und hat daher die gleiche Wirkung wie @ = @ + WERT (*1, *2, *4). Es wird keine entsprechende Zahl von 0-Bytes in die Ausgangsdatei geschrieben. D.h. blk.x darf nur am Ende des Programms stehen, es duerfen keine weiteren Anweisungen folgen, die eine Ausgabe von Bytes in die Ausgangsdatei zur Folge haben.

```
odd      : Current Location Pointer durch Ausgabe von 0x00 auf
          den naechsten ungeraden Wert erhoehrt
even     : Current Location Pointer durch Ausgabe von 0x00 auf
          den naechsten geraden Wert erhoehrt
even n   : Current Location Pointer durch Ausgabe von 0x00 auf
          das naechste Vielfache von n erhoehrt
even n,m : Current Location Pointer durch Ausgabe von m auf
          das naechste Vielfache von n erhoehrt
if WERT : Assemblierung wird ausgefuehrt falls WERT <> 0
else    : (WERT muss bereits im PASS1 berechnet werden koennen)
endif
```

list : erhoehrt Listlevel um 1
nolist : erniedrigt Listlevel um 1
(Ein Listing wird erzeugt, falls der Listlevel >= 0 ist.)

label_block: definiert einen neuen Gueltigkeitsbereich fuer normale Labels

WERT-Ausdruecke:

duerfen enthalten

1. Operatoren:

hoechste Prioritaet
() : Klammern
! ~ - : logisches not, binaeres not, Minus-Vorzeichen
* / \ : Multiplikation, Division, Rest (2er-Komplement-Darst.)
+ - : Addition, Subtraktion
<< >> : Linksshift, Rechtsshift
< > <= >= : kleiner, groesser, kleiner-gleich, groesser-gleich
== != : gleich, nicht gleich
& : binaeres AND
^ : binaeres XOR
| : binaeres OR
&& : logisches UND (FALSE = 0; TRUE = alles andere
|| : logisches ODER (TRUE als Ergebnis = 1)
niedrigste Prioritaet

Bei gleicher Prioritaet von links nach rechts ausser bei (! ~ -),
hier von rechts nach links.

2. Operanden

xyz : Labels
1234 : Dezimalzahlen
\$1234 : Hexadezimalzahlen
%1010 : Binaerzahlen
'a' bzw. "a" : Char-Byte-Konstante
'ab' bzw. "ab" : Char-Word-Konstante
'abc' bzw. "abc" : Char-3Byte-Konstante
'abcd' bzw. "abcd" : Char-Long-Konstante

bei dc.b duerfen beliebig lange (max. 100 Zeichen) Char-Konstanten stehen
z.B. dc.b "das Zeichen ' kann ebenfalls zum Einschliessen verwendet werden"

Labels:

Erstes Zeichen muss ein Buchstabe oder @ oder _ oder . oder # sein.
Weitere Zeichen muessen Buchstabe oder Zahl oder @ oder _ oder \$ sein.

Labels die mit _ beginnen sind lokale Labels, d.h. ihr Gueltigkeitsbereich erstreckt sich nur auf den Bereich zwischen zwei normalen Label. Dadurch kann der gleiche lokale Labelname an verschiedenen Stellen des Programms verwendet werden, ohne dass Konflikte auftreten.

Die Wertzuweisung an ein normales Label erfolgt entweder durch das nachstellen eines : oder = nach dem Labelnamen. Im ersten Fall erhaelt das Label den Wert des Current Location Pointer @, im zweiten Fall den Wert des Ausdruckles, der nach dem = folgt. (abc: und abc=@ sind gleichwertig; beachten Sie, dass in beiden Faellen ein neuer Geltungsbereich fuer lokale Labels beginnt.) Der Gueltigkeitsbereich von normalen Labels wird durch das letzte und das naechste Auftreten des Schluesselwortes LABEL_BLOCK begrenzt.

Erfolgt die Wertzuweisung durch :: bzw. durch == so handelt es sich um globale Labels, deren Gueltigkeitsbereich die gesamte Datei ist. Labels die mit _ beginnen sind auch bei Verwendung von :: bzw. == lokale Labels.

Labels die mit @ beginnen kann mehrmals ein Wert zugewiesen werden (nur sinnvoll für = Zuweisung). Der Wert muss jedoch bereits in Pass 1 berechnet werden koennen. Diese Labels sind immer globale Labels auch wenn nur : bzw. = zur Zuweisung verwendet wird. Durch diese Labels beginnt jedoch kein neuer Gueltigkeitsbereich fuer lokale Labels!

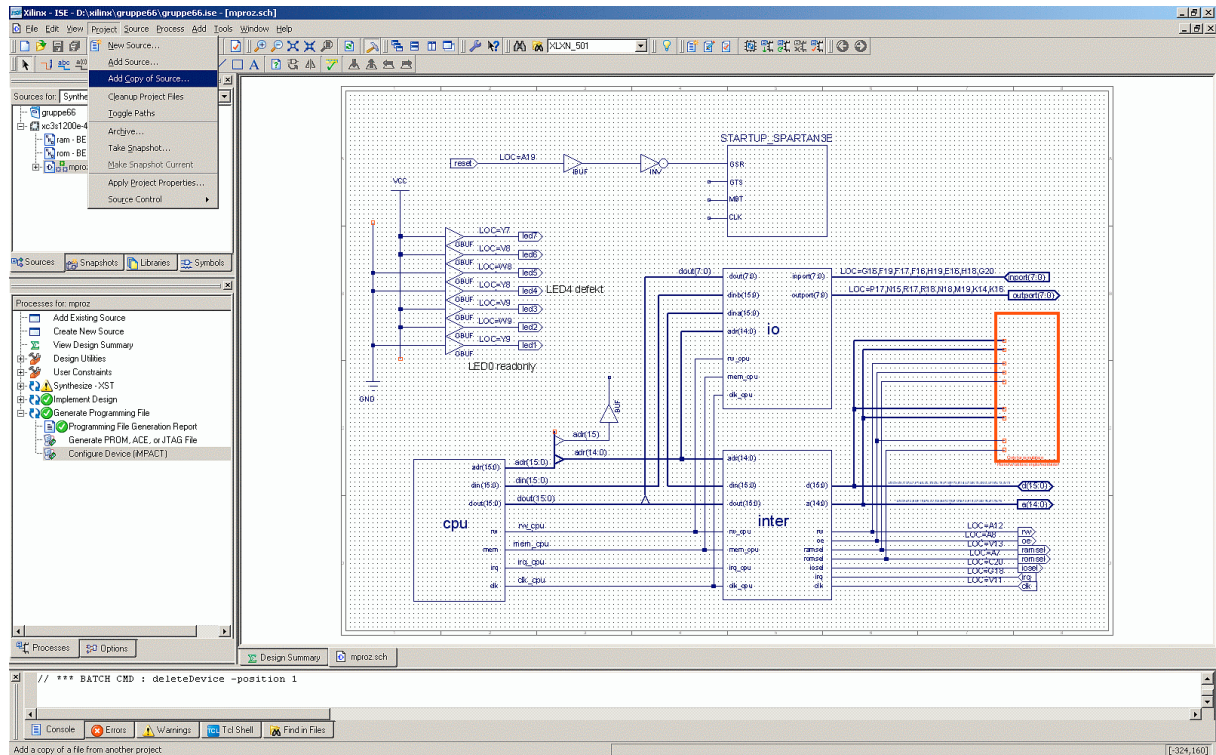
Beginnen normale oder globale Labels mit einem ".", so wird durch diese Labels kein neuer Gueltigkeitsbereich fuer lokale Labels erzeugt.

Das Label mit dem einen Buchstaben @ ist ein reserviertes Label und stellt den Current Location Pointer dar. @ darf innerhalb des Programms (normalerweise) nicht veraendert werden (vgl. Anmerkung zu blk.b).

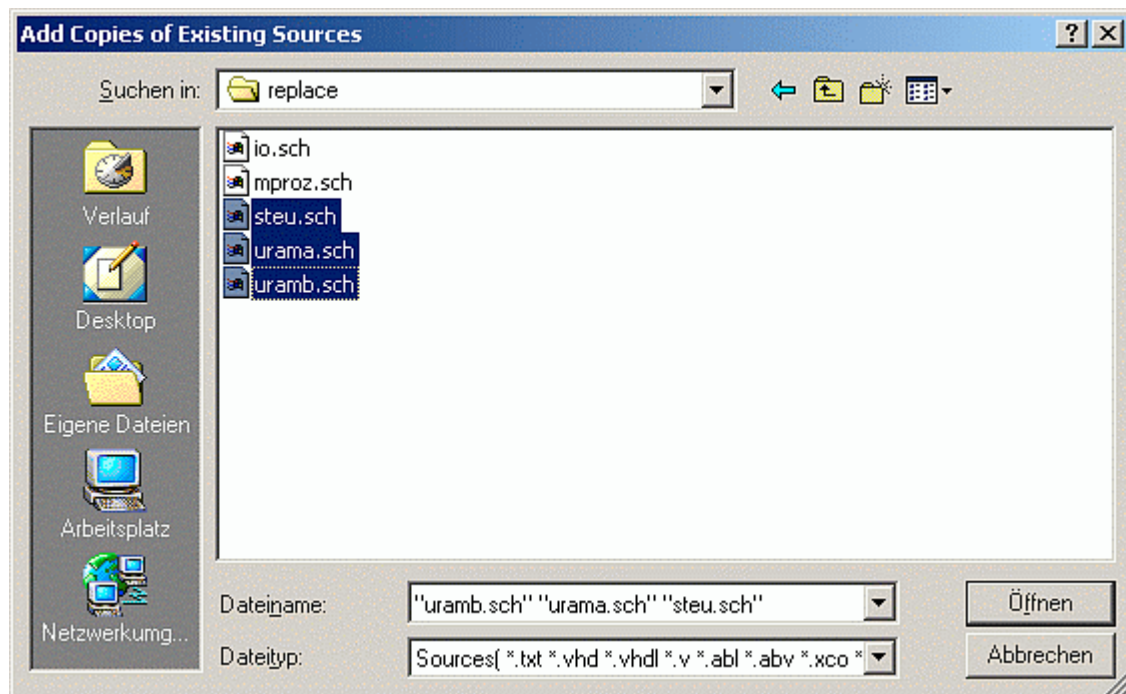
Das Label mit dem Namen @@ ist ein reserviertes Label und stellt den Ausgabe File Pointer dar. @@ darf innerhalb des Programms nicht veraendert werden.

Anhang F

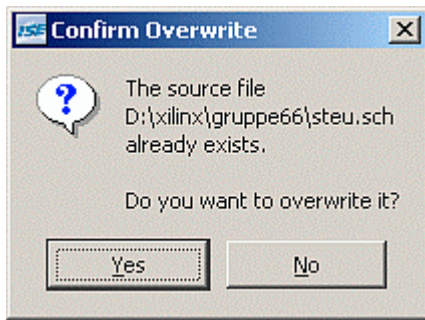
Mikroprogrammiertes Steuerwerk



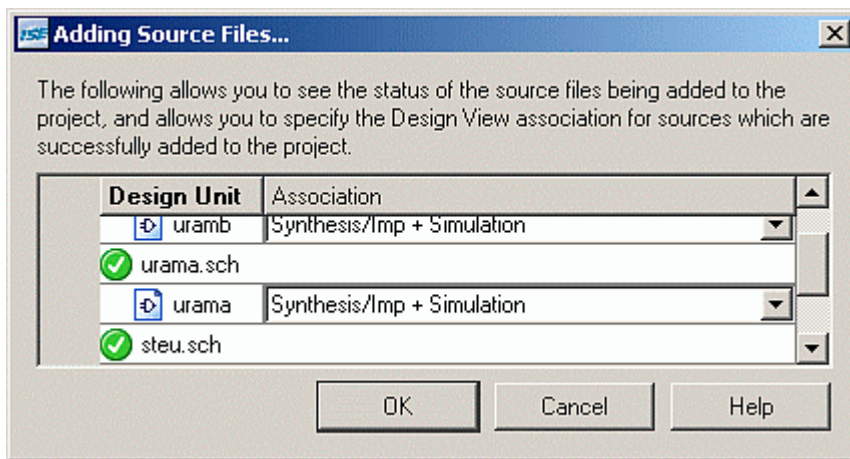
Fügen Sie RAM/ROM wieder ein und selektieren Sie im Menü: "Add Copy of Source"



Wählen Sie die drei Dateien steu.sch, urama.sch und uramb.sch aus und klicken Sie auf "Öffnen"



Clicken Sie auf "Yes".



Clicken Sie auf "OK".

Vervollständigen Sie die Schaltung für das mikroprogrammierte Steuerwerk und geben Sie das entsprechende Mikroprogramm in den Mikroprogrammspeicher ein (Rechtsklick auf die Speicherbausteine -> Eigenschaft). Achten Sie darauf, immer vierstellige Hexzahlen einzugeben (führende Nullen mit eingeben).

Simulieren Sie den Prozessor, generieren ein Bitfile und laden es auf das Demoboard.