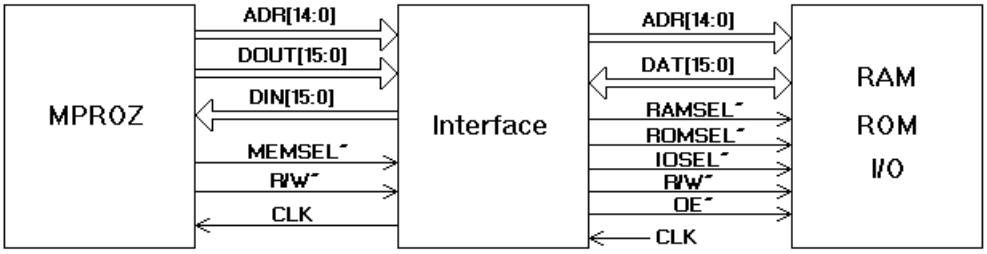
Rechnerorganisation II
1. Übungsblatt

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



a) Schnittstellen des Prozessors

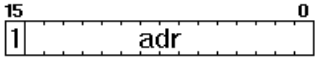
adr[14:0]	: 15-Bit Adreßbus
din/dout[15:0]	: 16-Bit Datenbus
memsel~	: Zugriff auf ram/rom/io
r/w~	Schreib- / Lesezugriff
clk	: Takt

b) Register

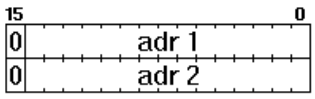
Um den Hardwareaufwand so klein wie möglich zu halten, enthält das Programmiermodell von MPROZ neben einem 15-Bit Befehlszähler (PC) und einem 1-Bit Zustandsflag (F) keine weiteren Register. Nach einem Reset enthalten PC und F den Wert 0.

c) Befehlssatz

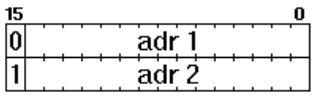
MPROZ unterstützt insgesamt drei Befehle:



Lade adr in PC falls F=0
Lösche F



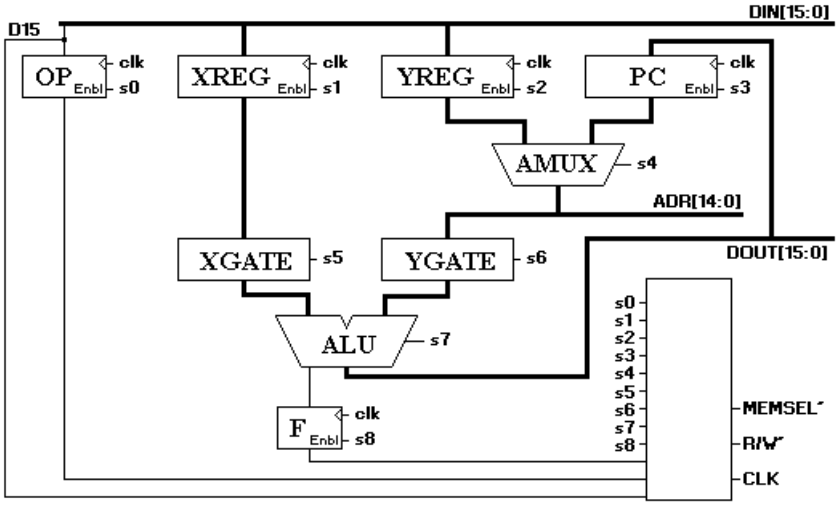
Addiere den Inhalt von adr1 zu dem Inhalt von adr2 und speichere das Ergebnis in adr2.
Speichere das Übertragsbit der Addition in F.



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 Zustands-übergangsdiagramm 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)

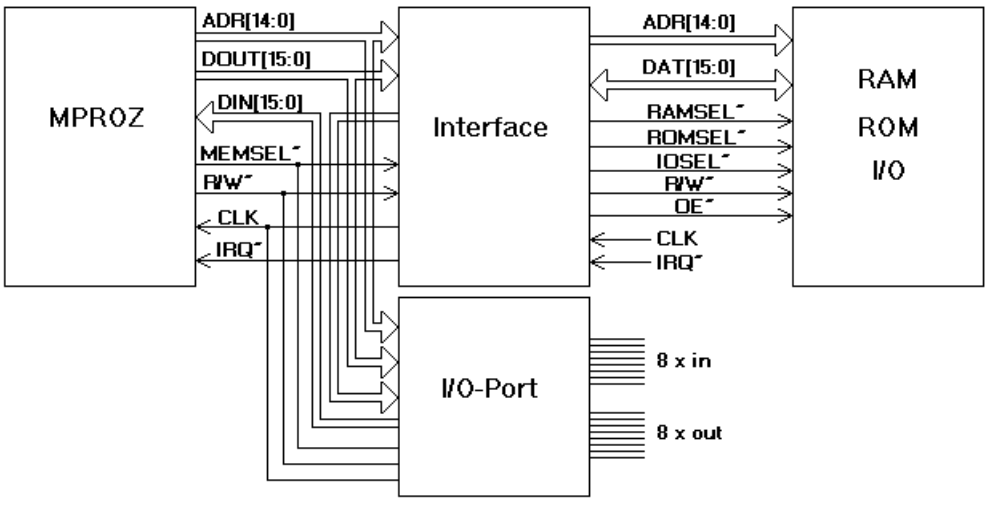
Rechnerorganisation II
3. Übungsblatt

1. 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 sowenig neue Zustände wie möglich ein.

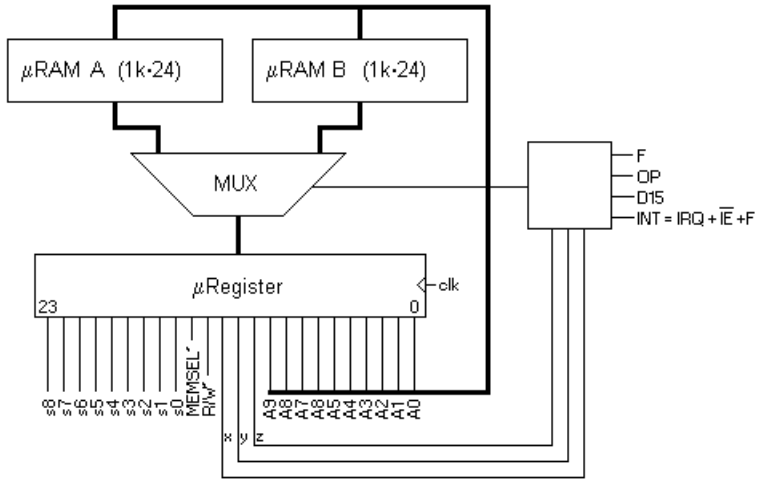
   Geben Sie die neue Schaltung ein und simulieren Sie diese (siehe Anhang C).

2. 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

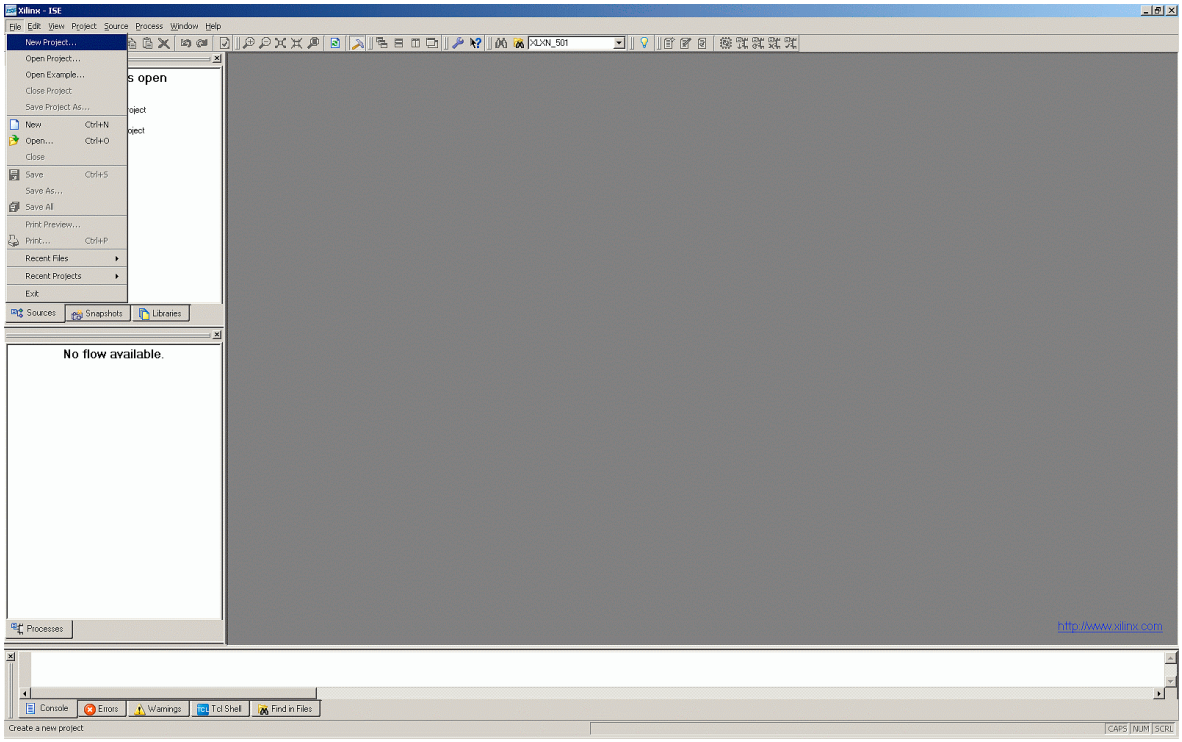
Erstellen Sie das Mikroprogramm und simulieren Sie die Schaltung (siehe Anhang D).

Anhang A

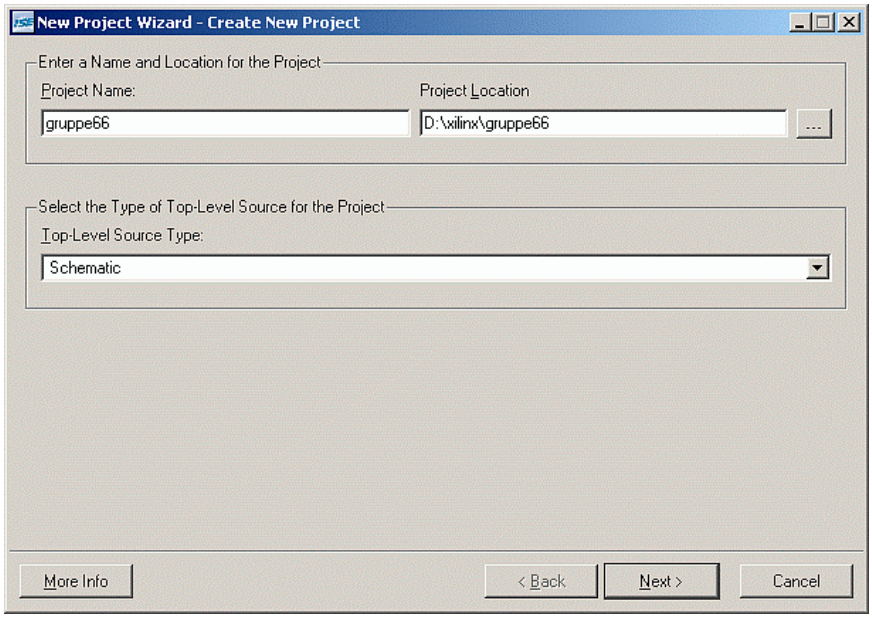
Falls noch nicht geschehen, extrahieren Sie den Anhang (mproz.zip) dieser pdf Datei und entzippen Sie ihn in einen temporären Order. Sie erhalten dann die beiden Order "import" und "addon".

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.

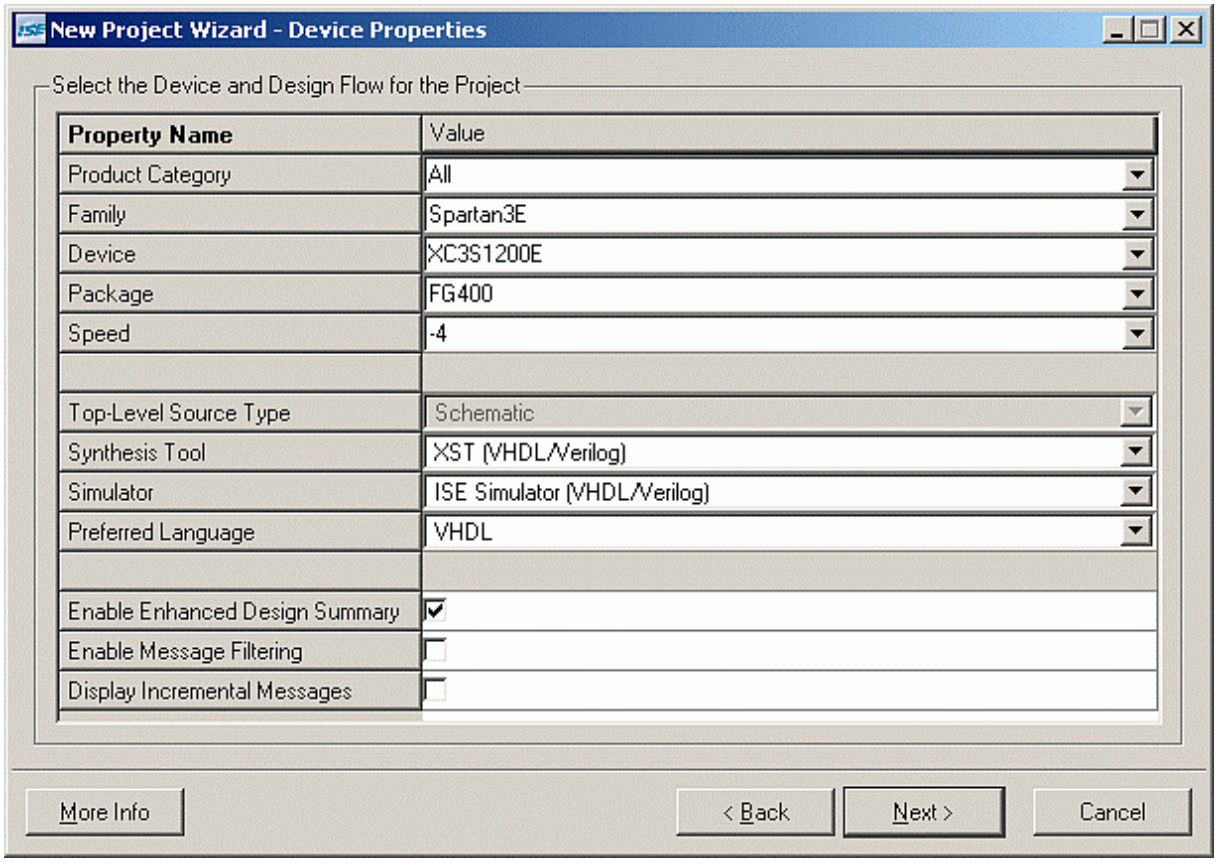
Starten Sie das Programm und öffnen Sie das "File" Menü:



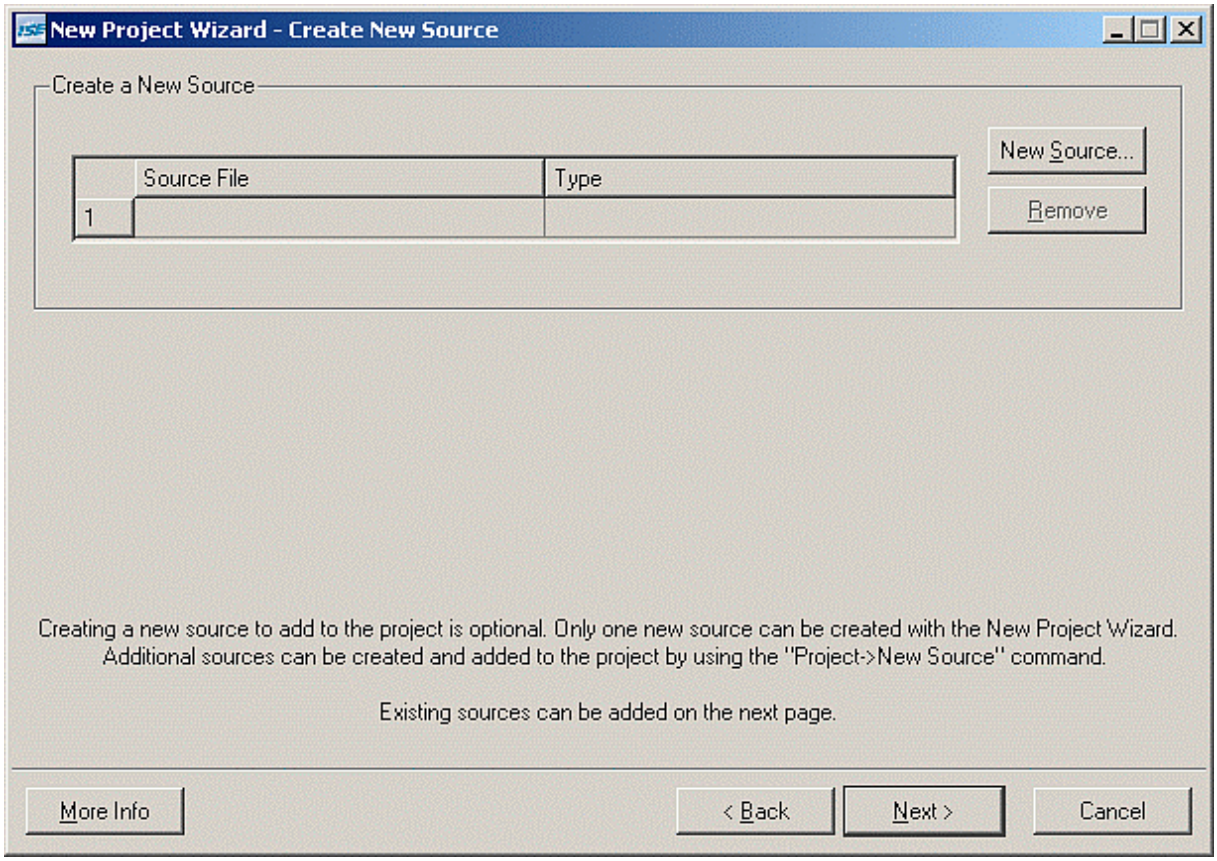
Machen Sie einen Linksclick auf "New Project..."



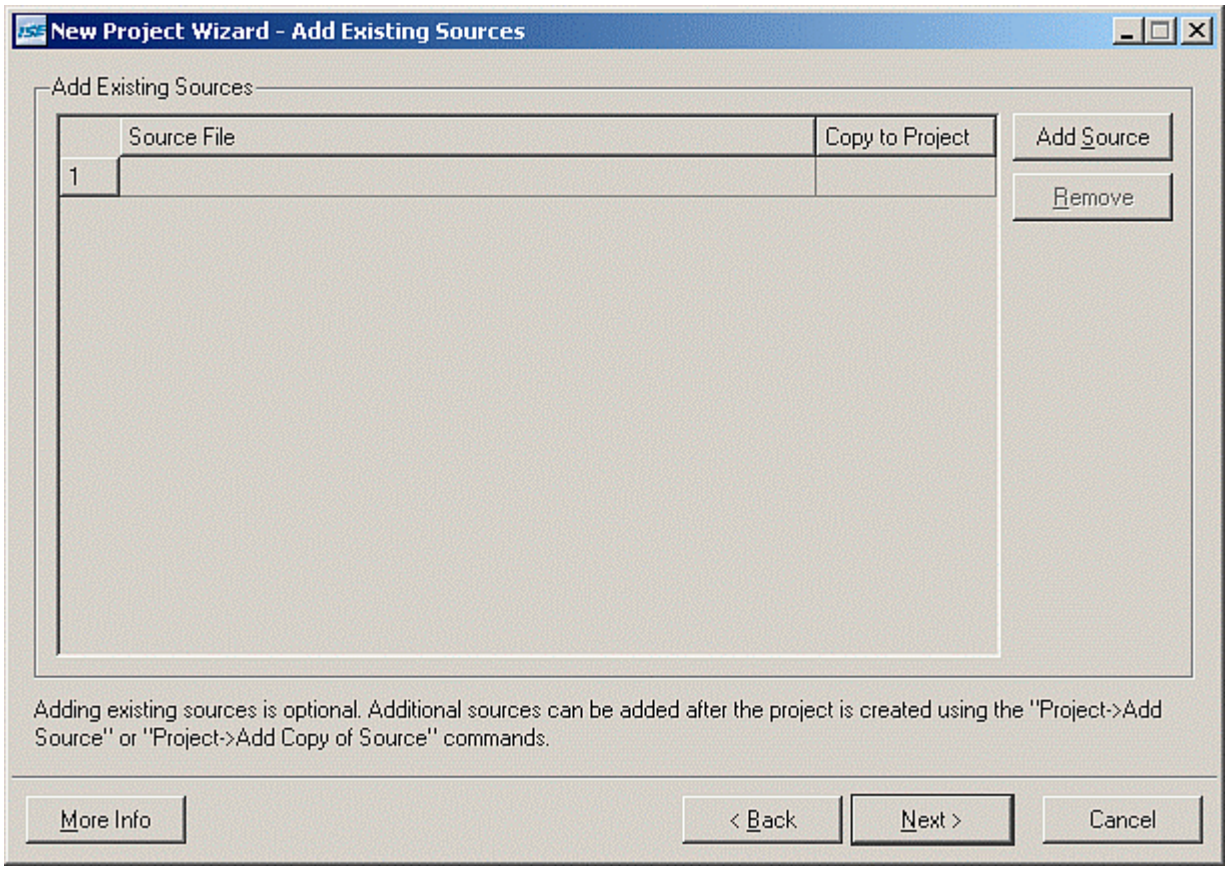
Geben Sie als Projekt Name "gruppexx" (xx= ihre Gruppennummer) ein und clicken Sie dann auf "Next".



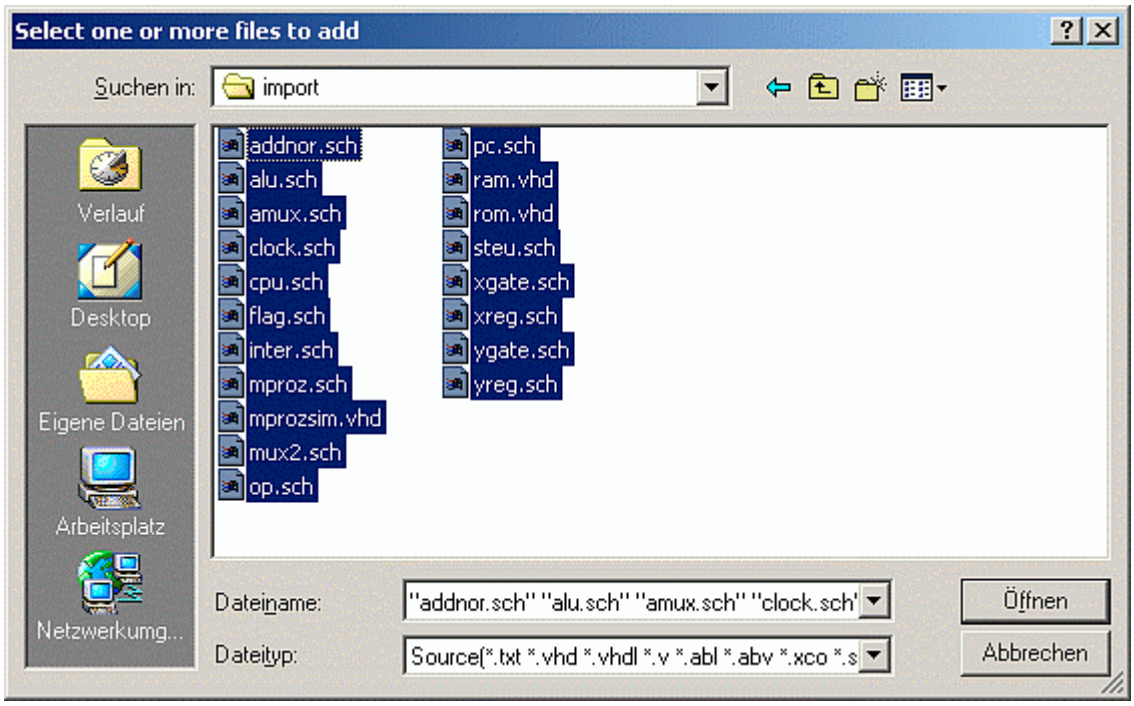
Obige Eingaben auswählen und dann auf "Next" clicken.



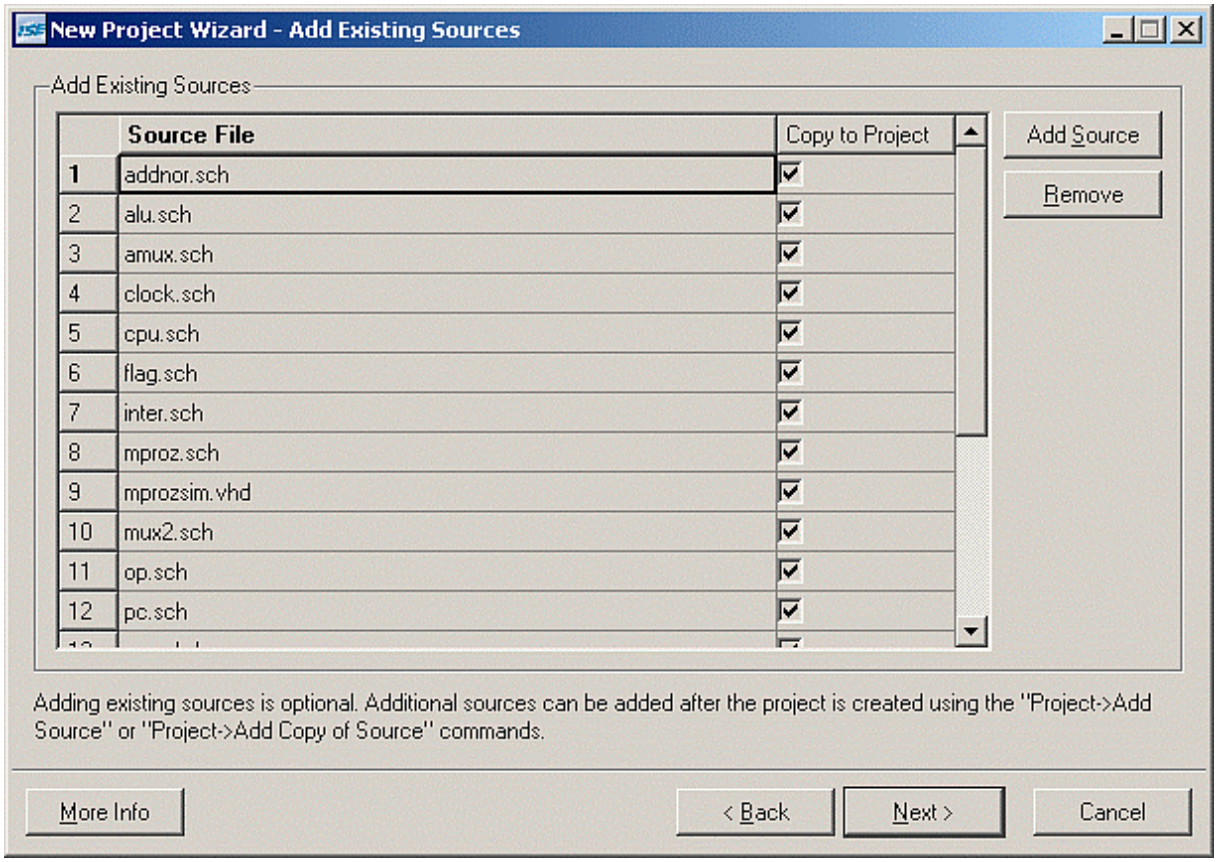
Keine "New Source" generieren, nur auf "Next" clicken.



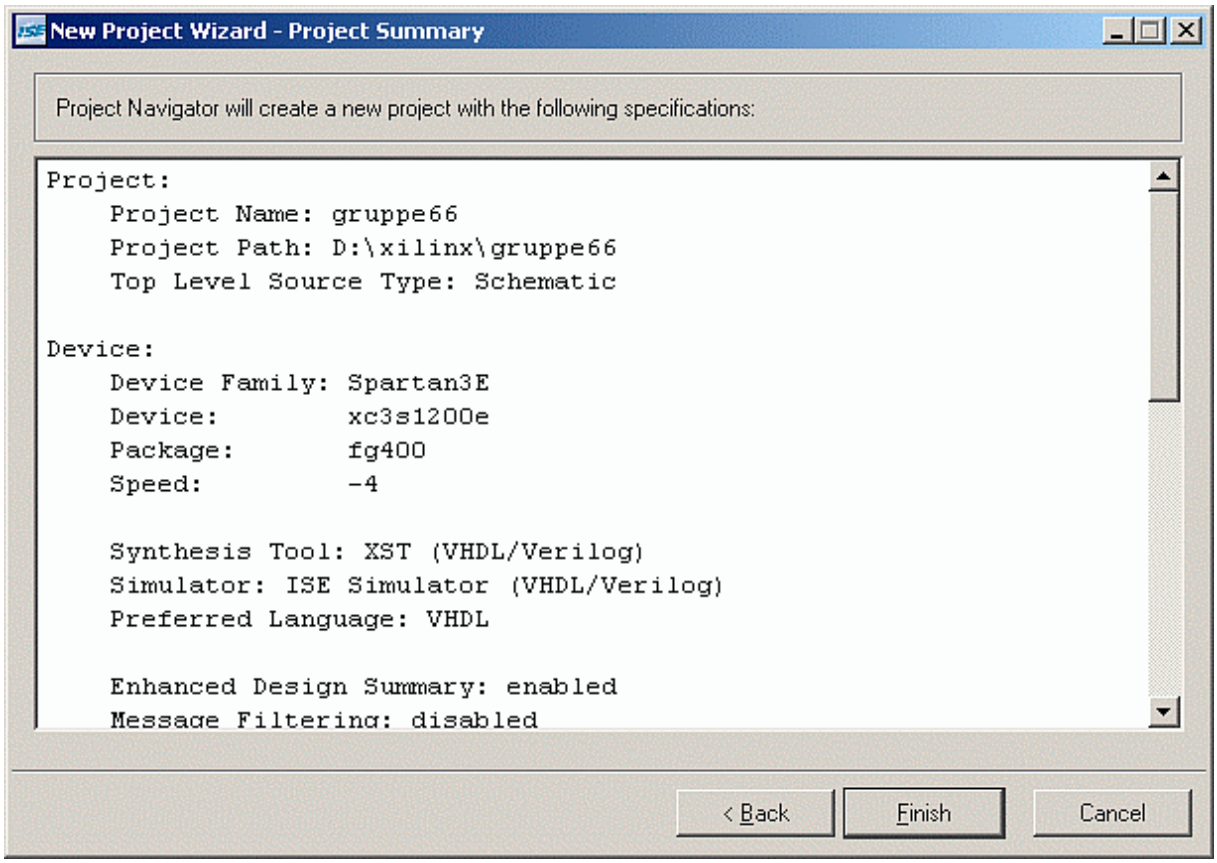
"Add Source" anclicken.



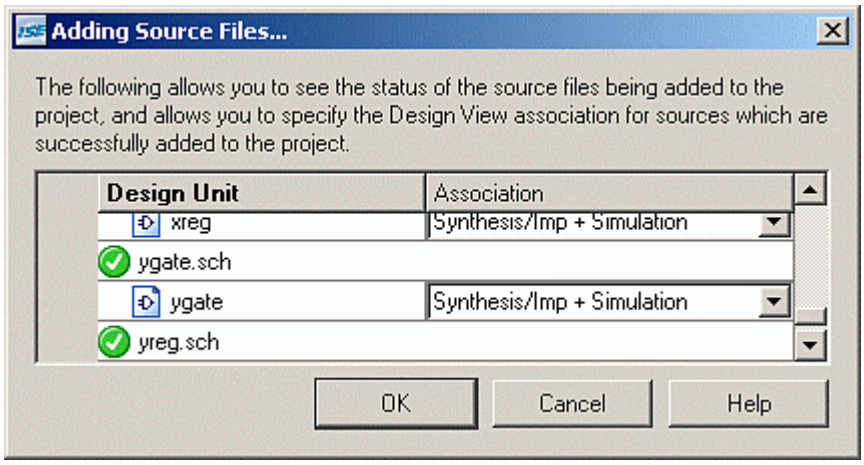
Alle Dateien im "imort" Ordner (aus mproz.zip) auswählen und dann auf "Öffnen" clicken.



"Next" anclicken.

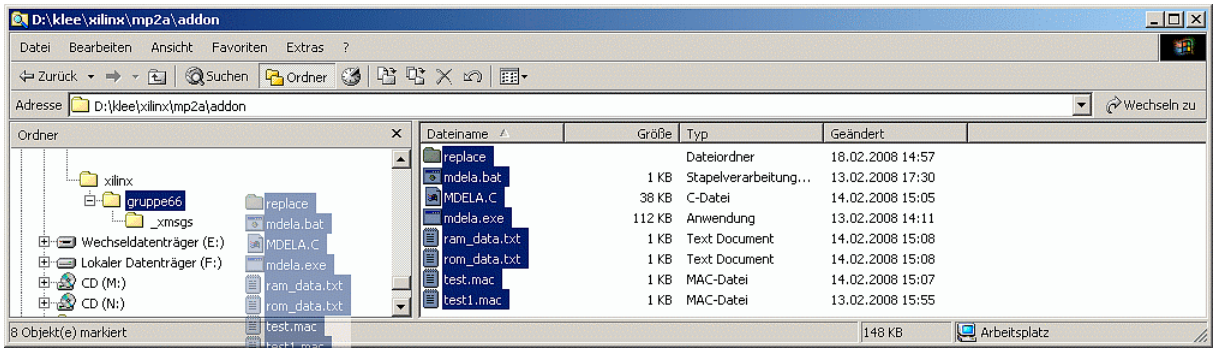


"Finish" anclicken.

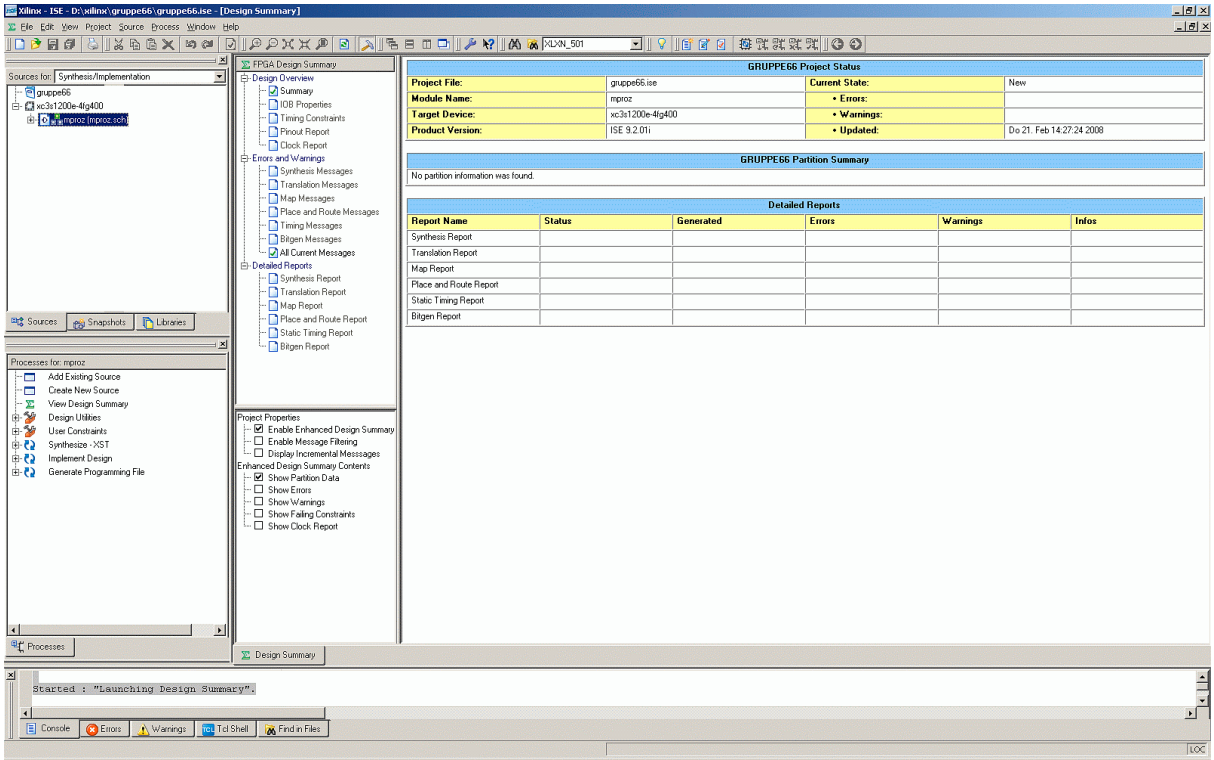


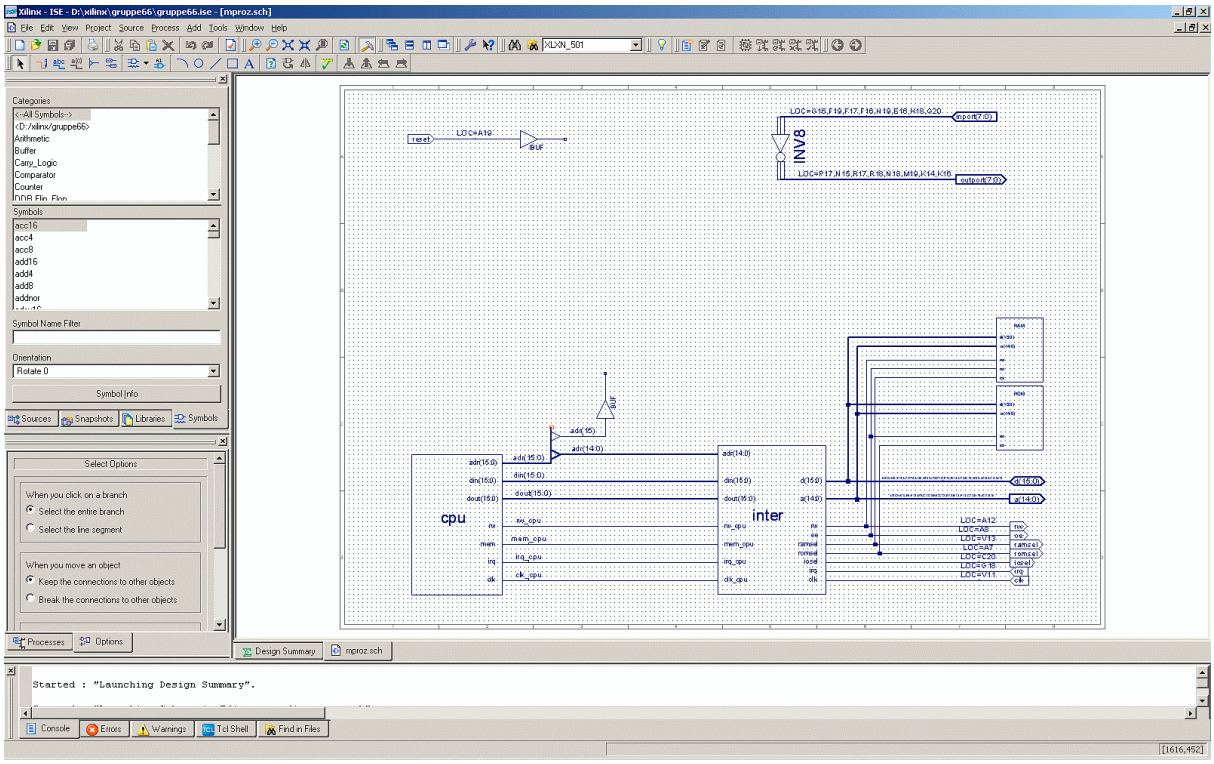
Warten Sie, bis alle Dateien importiert sind und clicken Sie dann auf "OK".

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



Doppelclicken sie auf "mproz.sch":





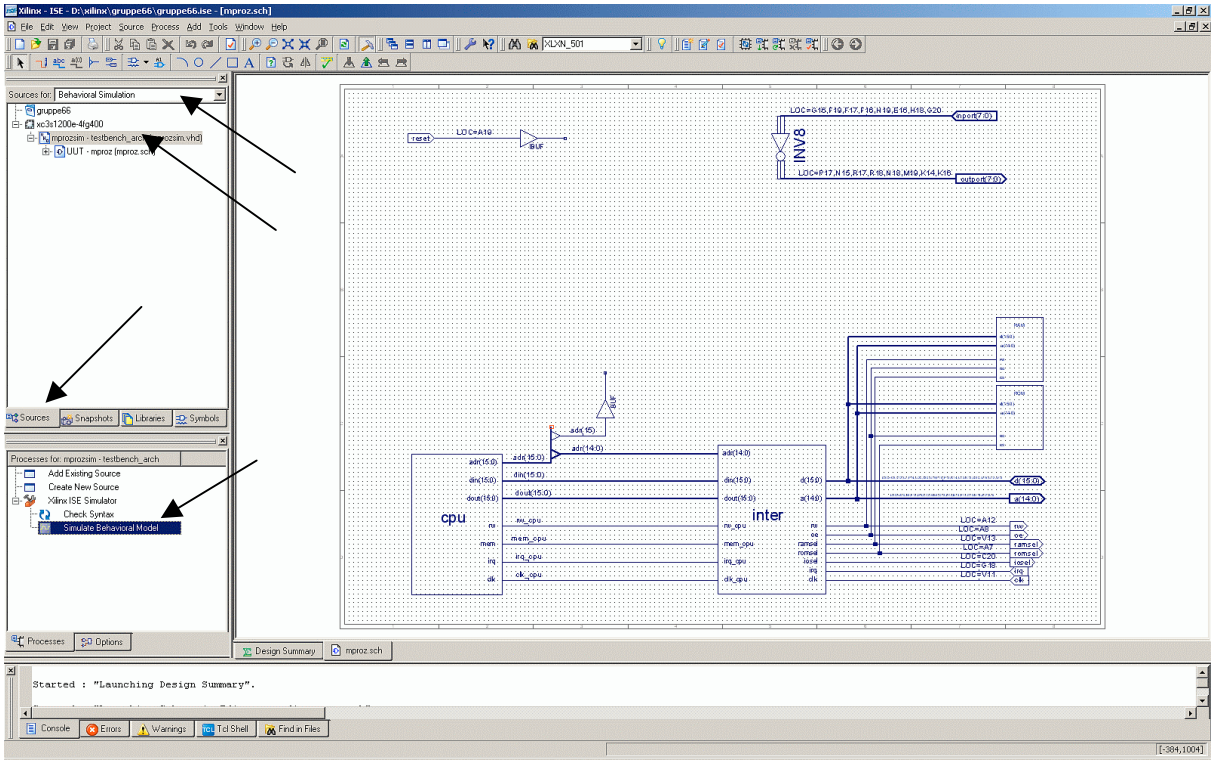
Sie sehen nun die oberste Schaltplanebene des Prozessors. Vervollständigen sie die Schaltpläne in folgender Reihenfolge:

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

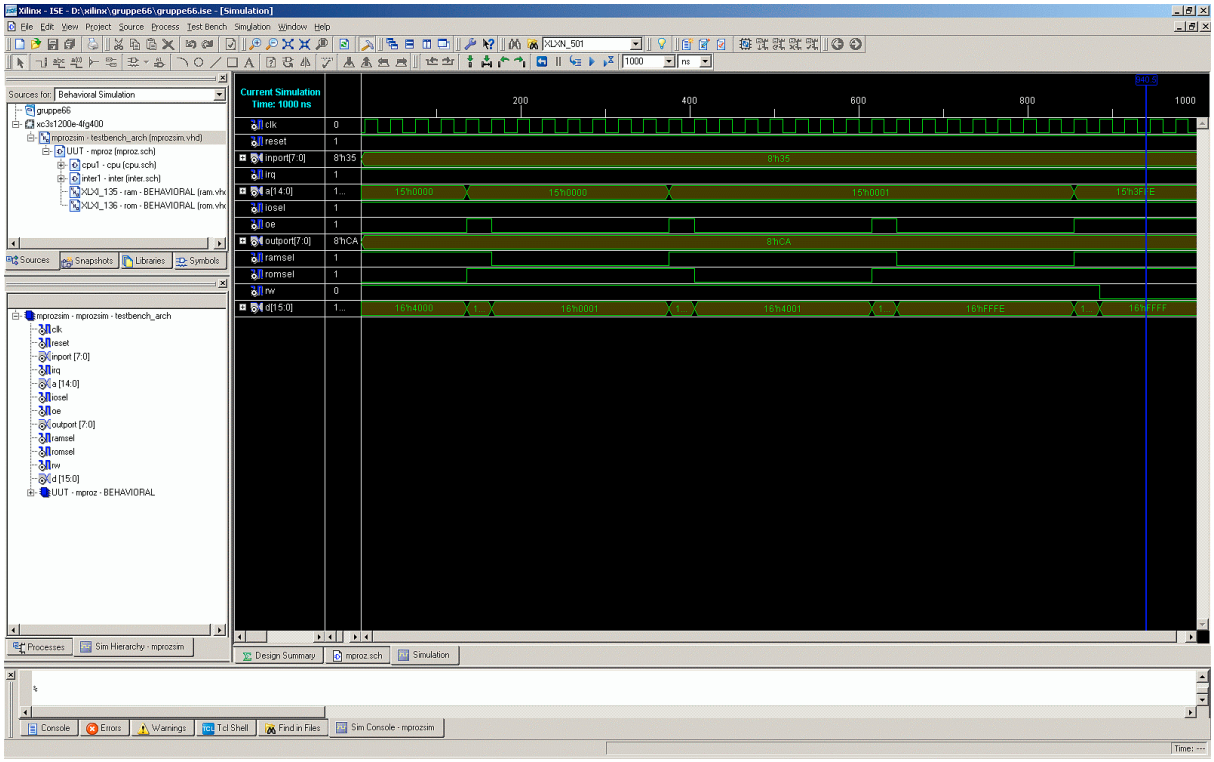
Verwenden Sie dazu ausschließlich einfache Gatter (inv, and, or, nand, nor, xor usw.) mit nicht mehr als fünf (eventuell. negierten) Eingängen und D-FlipFlops (mit und ohne Enable Eingang).

Anhang B

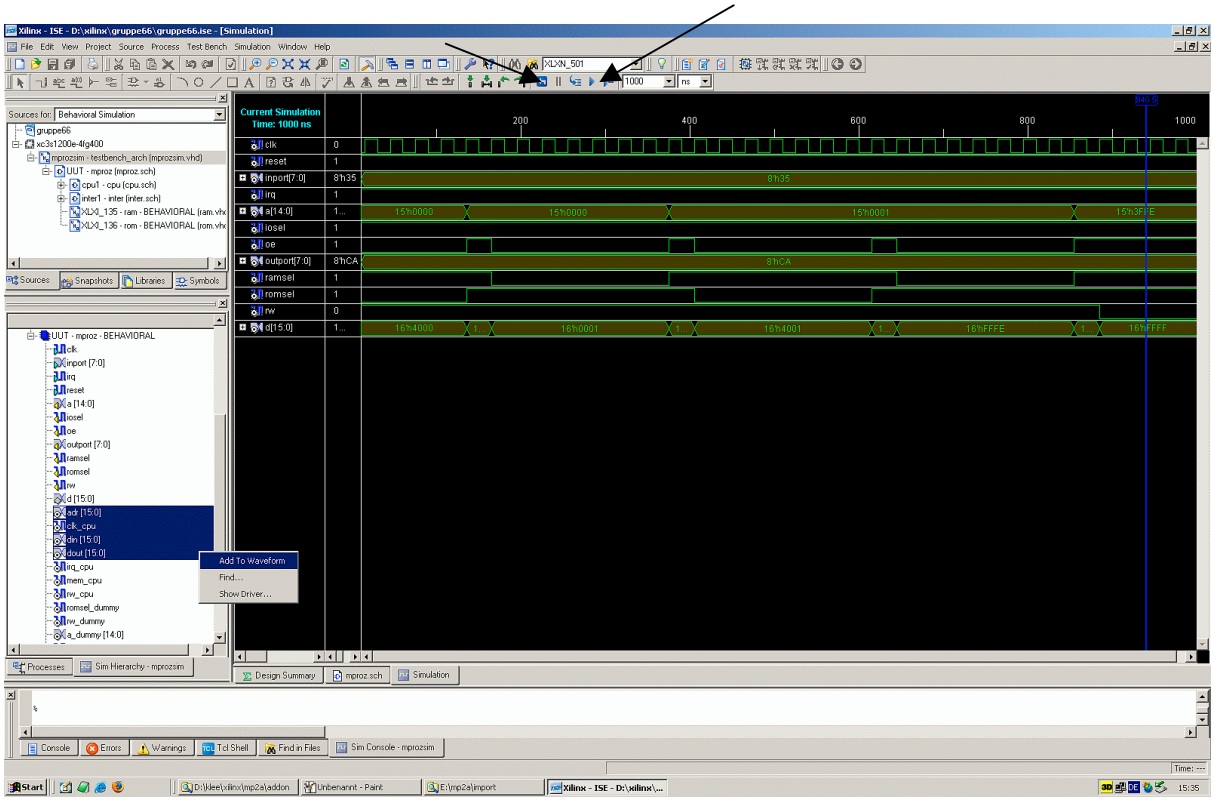
Selektieren Sie den Reiter "Sources" und "Behavioral Simulation" im linken oberen Fenster. Selektieren Sie "simutest - testbench_arch (mprozsim.vhd)" and doppelclicken Sie auf "Simulate Behavioral Model":



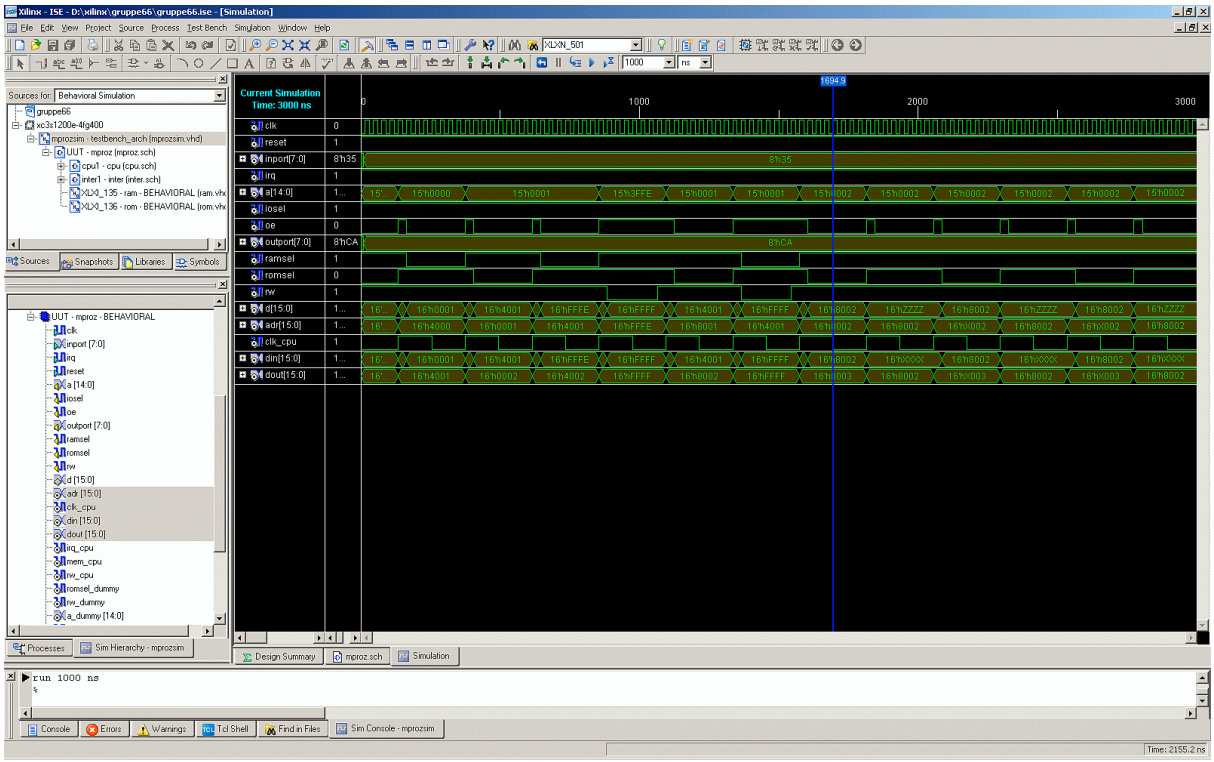
Sie sehen nun:



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

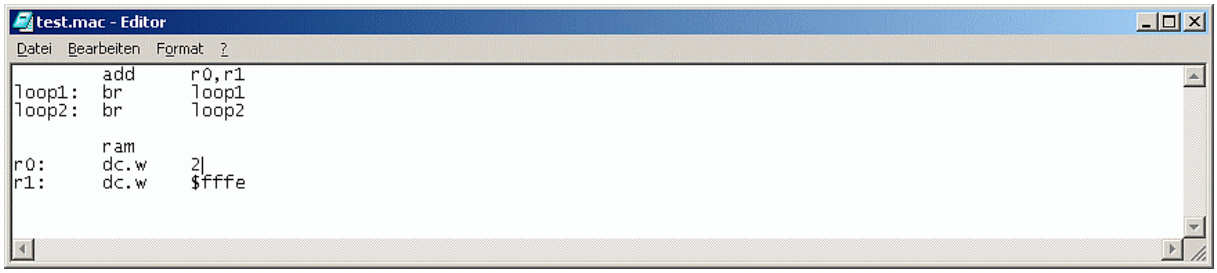


Setzen Sie die Simulation zurück und simulieren Sie ca. 3000 ns:



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).

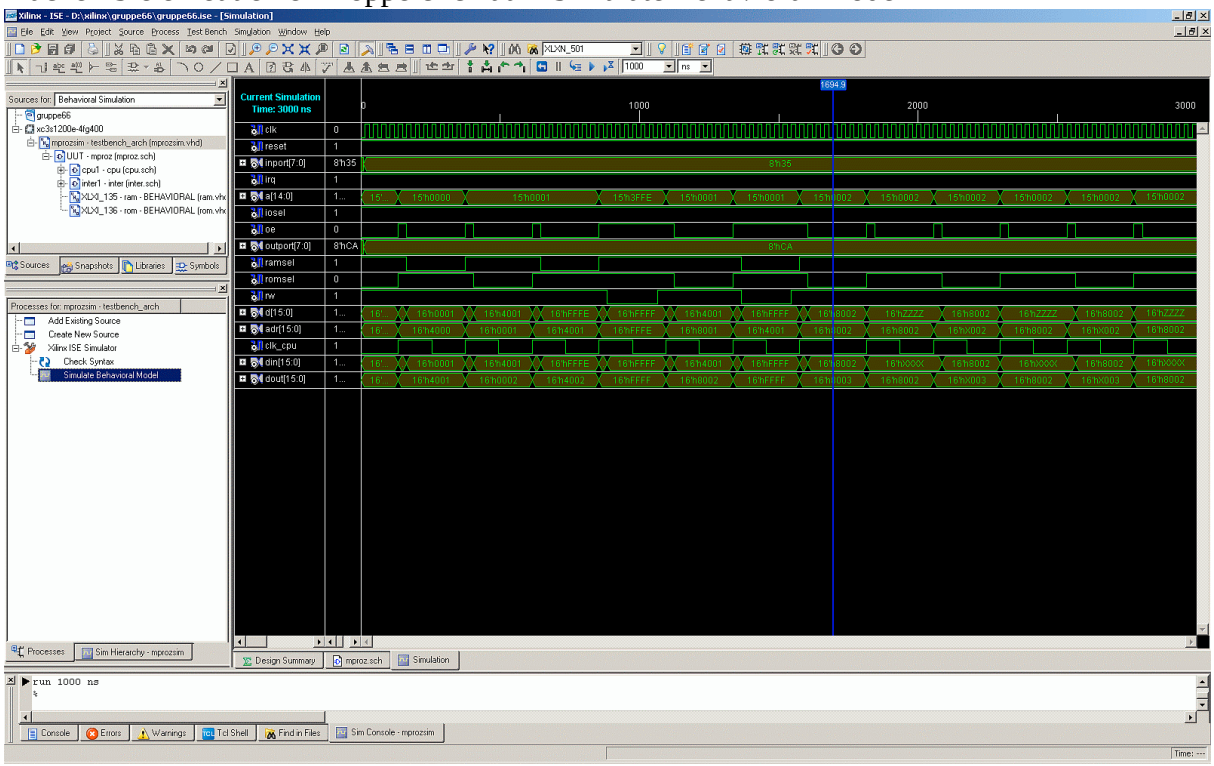


Speichern Sie die Datei ab und starten MDELA (entweder durch Eingabe von MDELA.EXE in einem CMD Fenster oder durch einen Doppelclick auf die Datei MDELA.BAT im Windows Explorer:

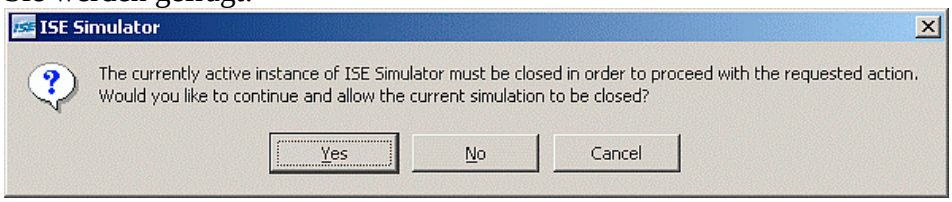
Geben Sie den Programmnamen "test" ein und drücken die Return Taste.

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

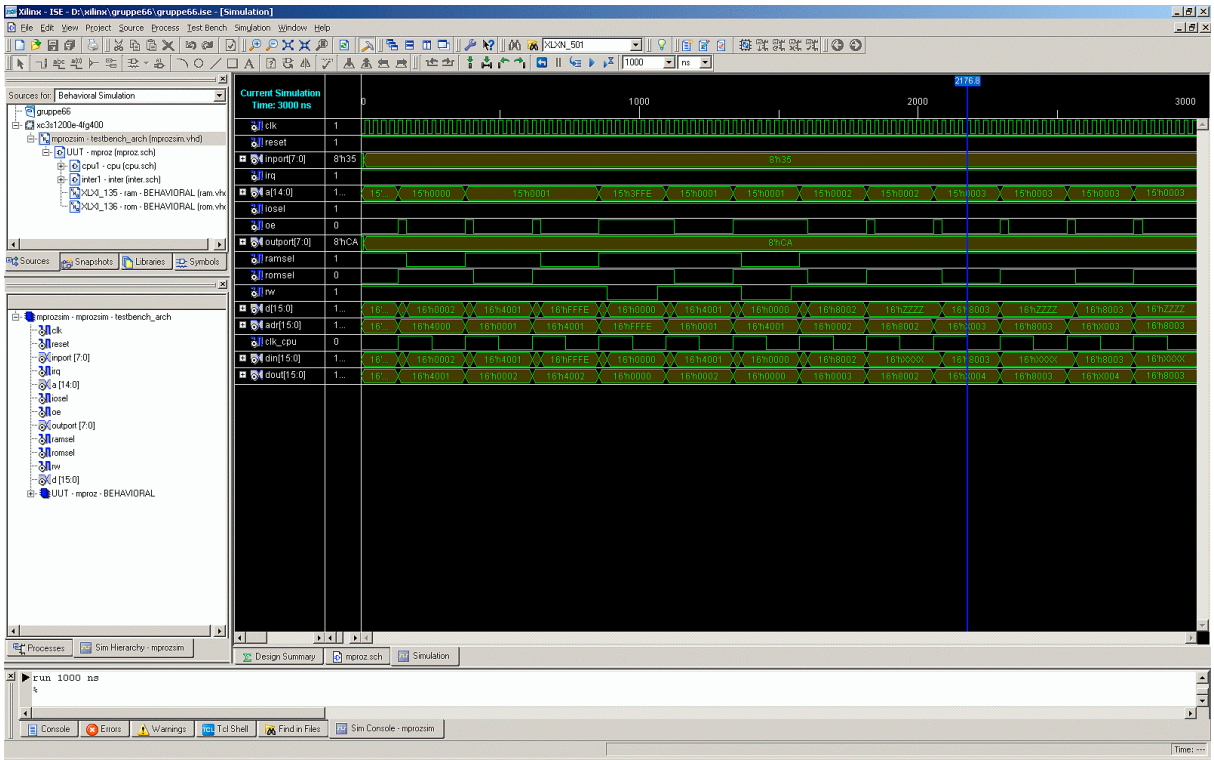
Machen Sie erneut einen Doppelclick auf "Simulate Behavioral Model"



Sie werden gefragt:



Clicken Sie "Yes".

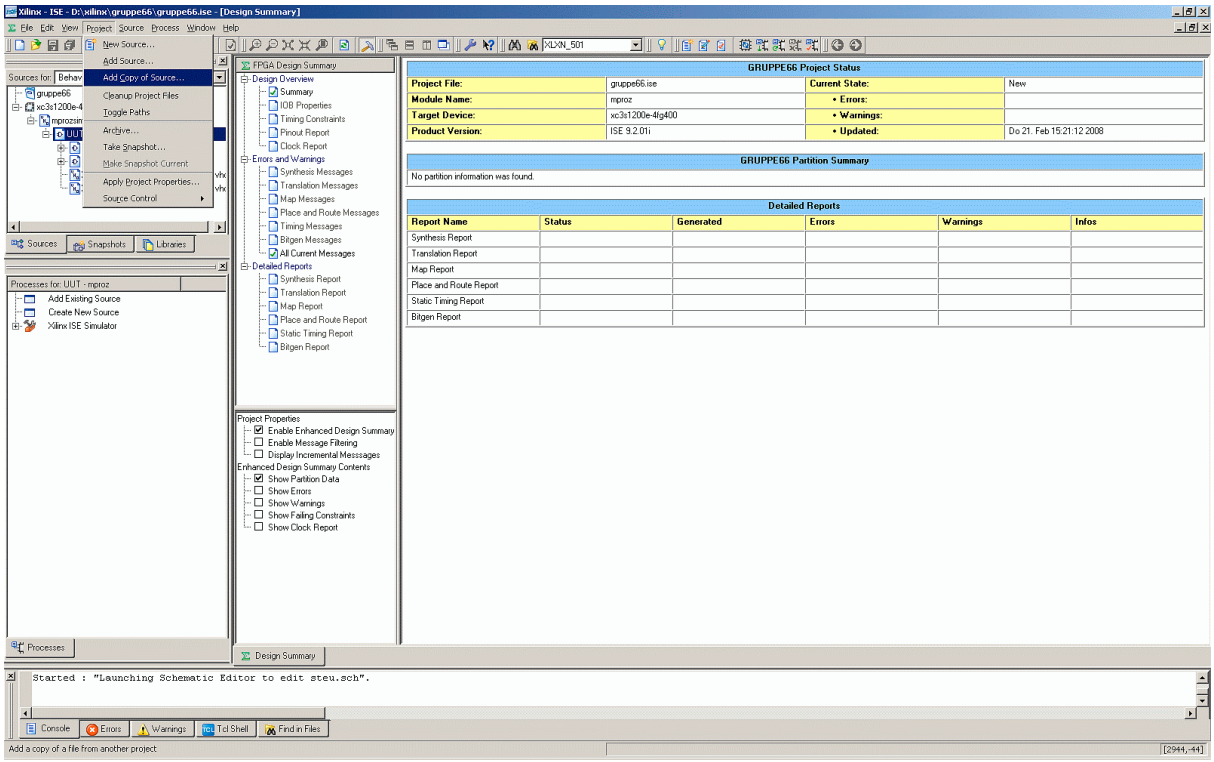


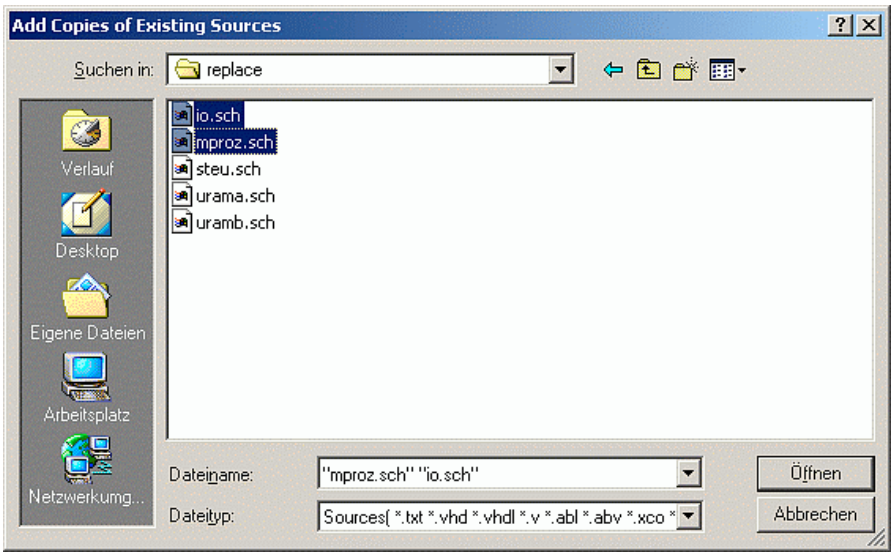
Da nun bei der Addition ein Carry generiert wird, befindet sich der Prozessor in "loop2". Testen Sie auch den nor Befehl (mit und ohne Setzen des Flags).

Anhang C

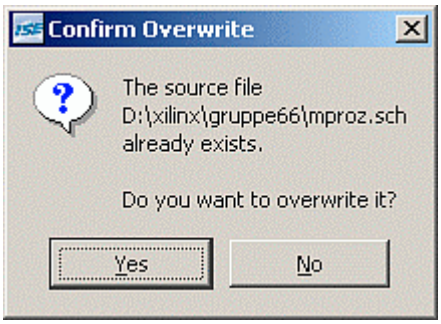
Modifizieren Sie nun das Steuerwerk so, daß es auch Interrupts unterstützt. Ändern Sie das Assemblerprogramm ebenfalls entsprechend ab und testen Sie die korrekte Funktion durch eine erneute Simulation.

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

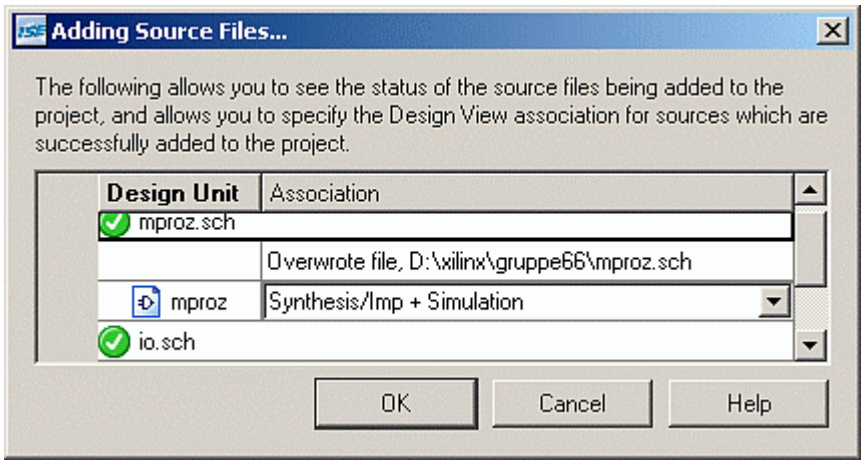




Selektieren Sie die beiden Dateien io.sch und mproz.sch

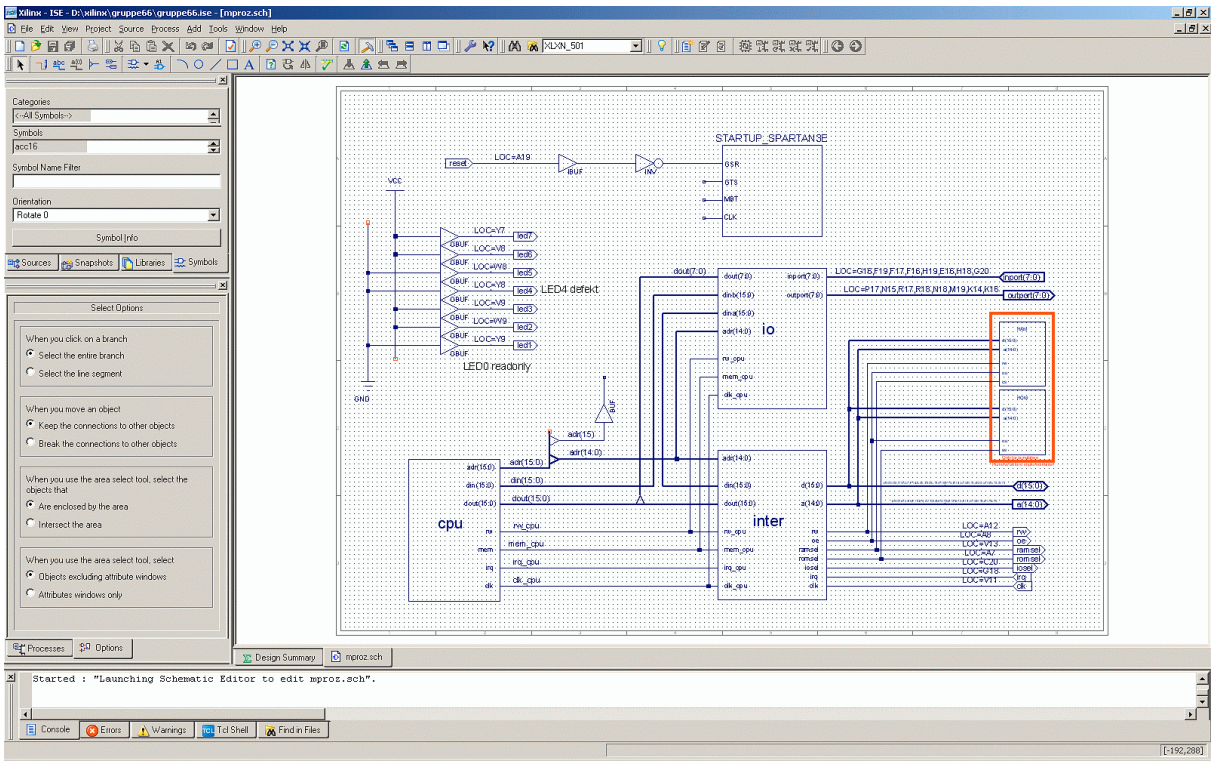


Clicken Sie auf "Yes".



Clicken 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.

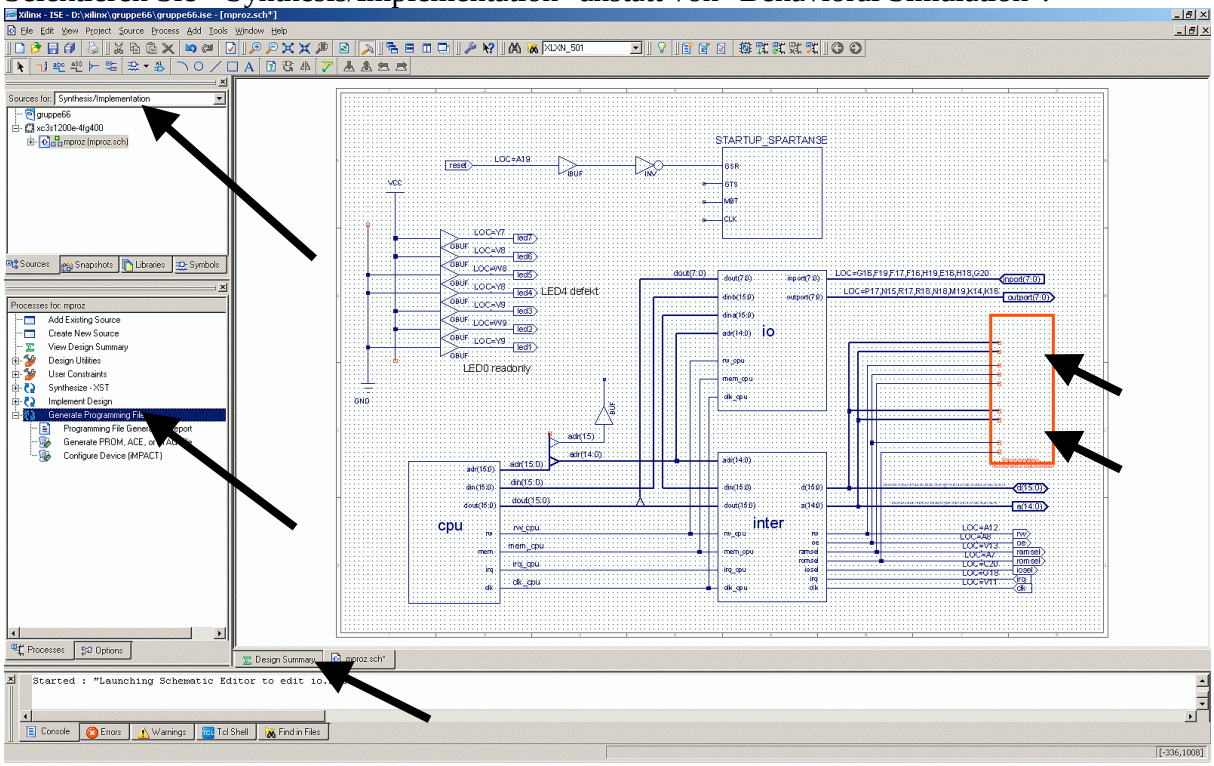


Beschalten Sie die Leuchtdioden so, daß Ihre Gruppennummer als zweistellige Oktalzahl dargestellt wird. Laden Sie nun die Schaltung wie im Anhang D beschrieben auf das Demoboard.

Anhang D

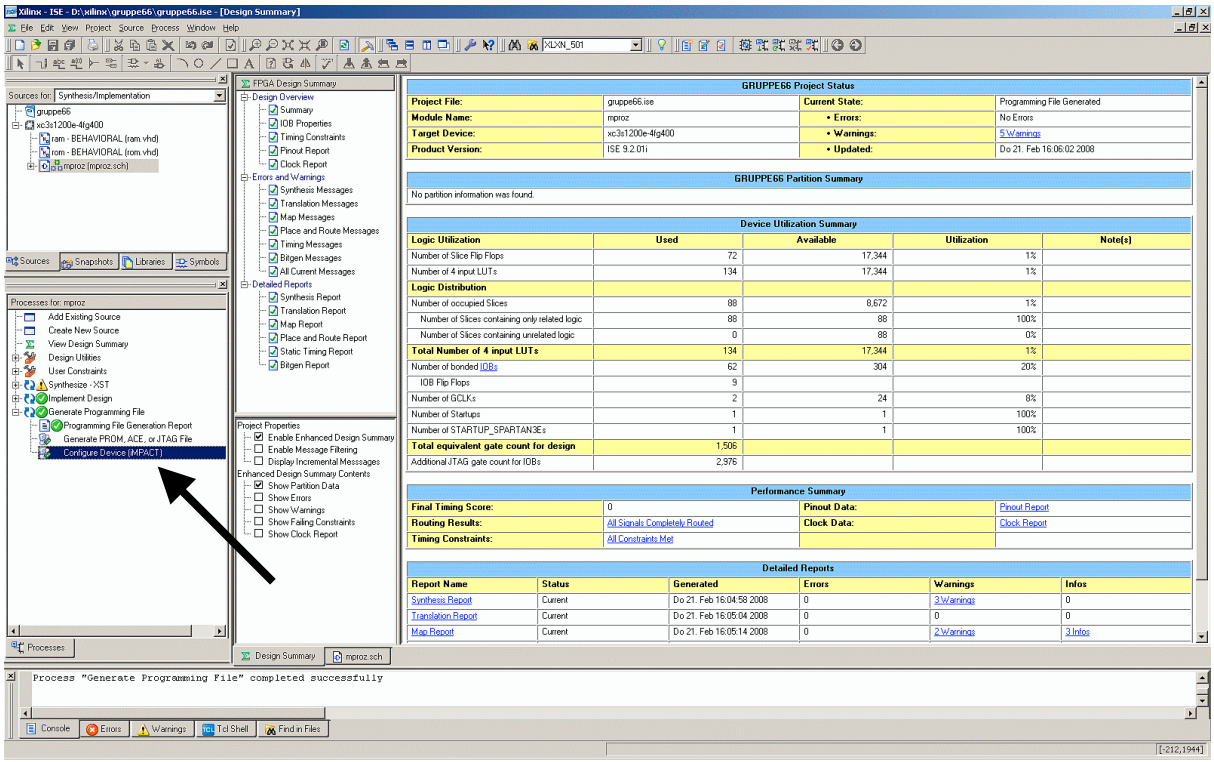
Download der Schaltung auf das Demoboard:.

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

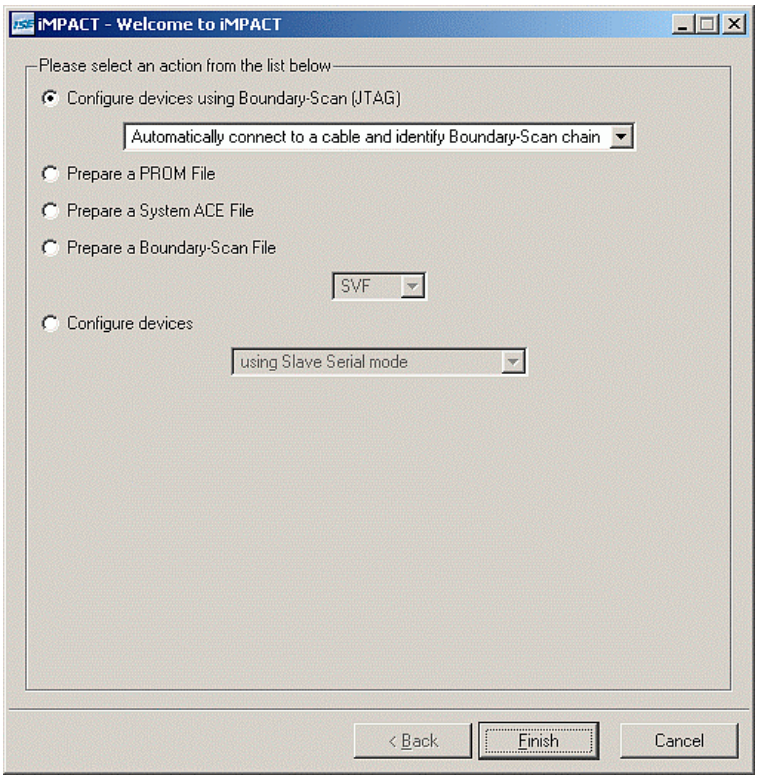


Entfernen Sie RAM und ROM und doppelclicken Sie auf "Generate Programming File":

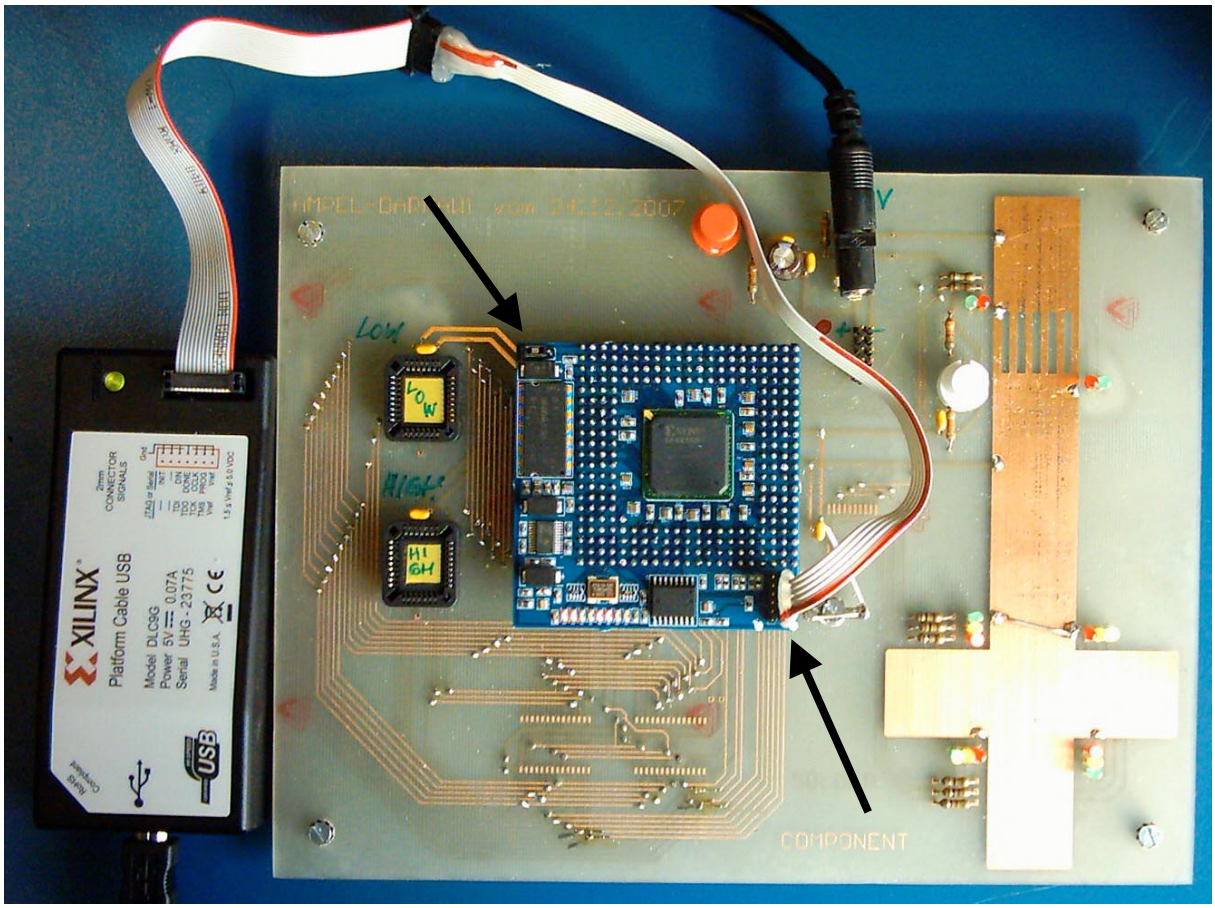
Selektieren Sie den Reiter "Design Summary". Sie sollten sehen::



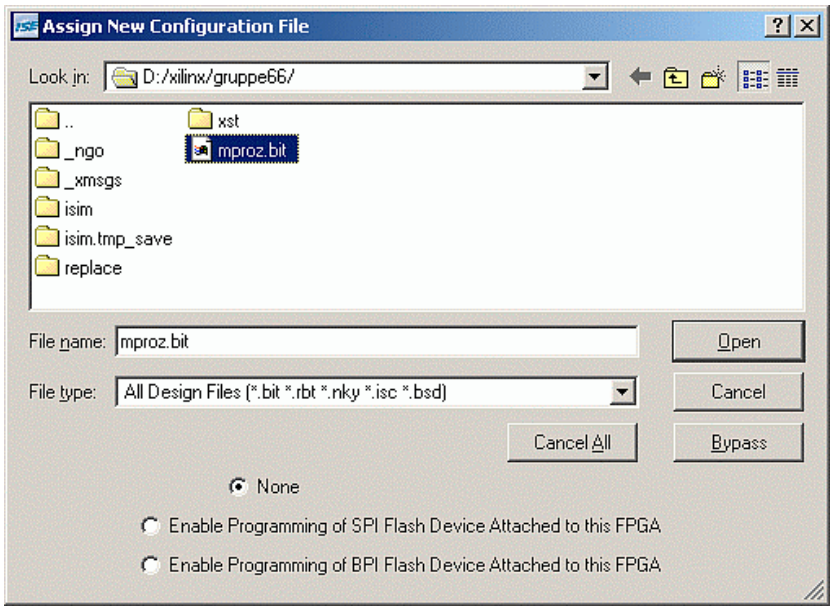
Doppelclicken Sie auf "Configure Device (iMPACT)".



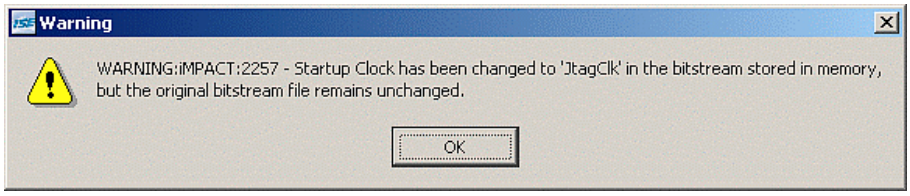
Verbinden Sie das Download-Kabel mit dem Demoboard und clicken "Finish".



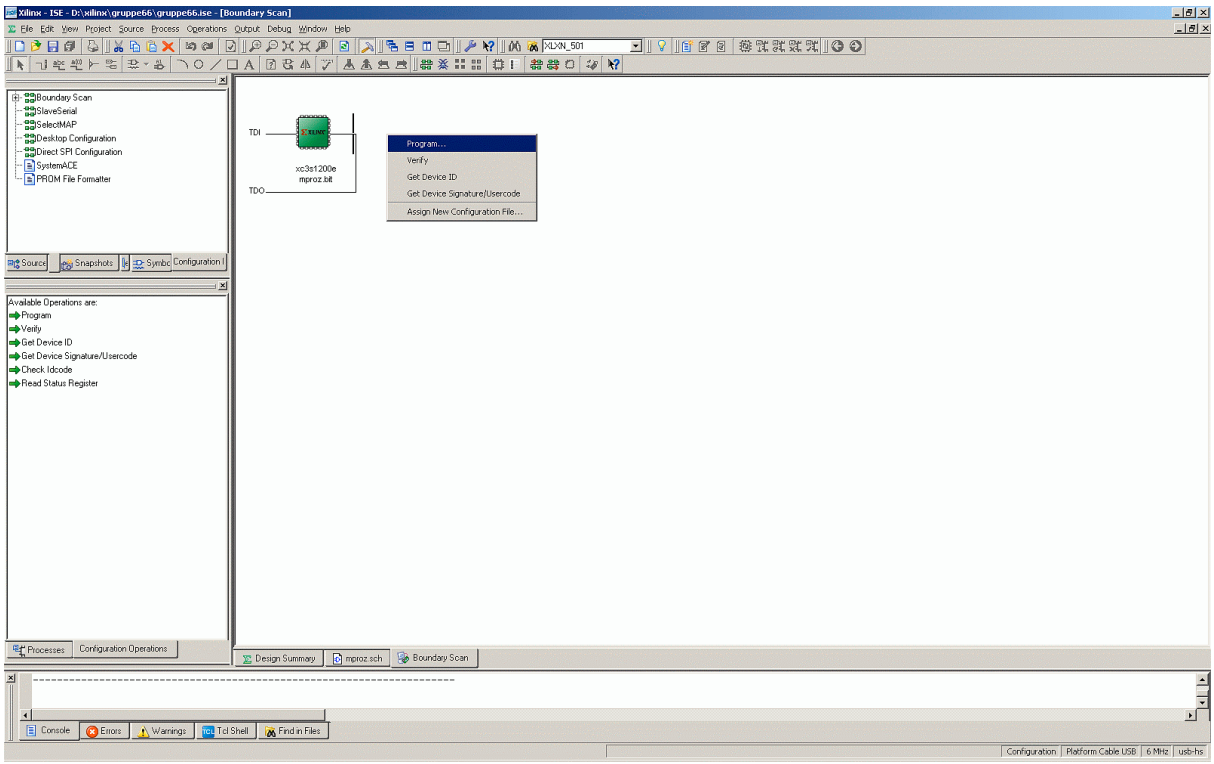
Verbinden Sie das Kabel mit der äußeren Stiftreihe und entfernen den Jumper.



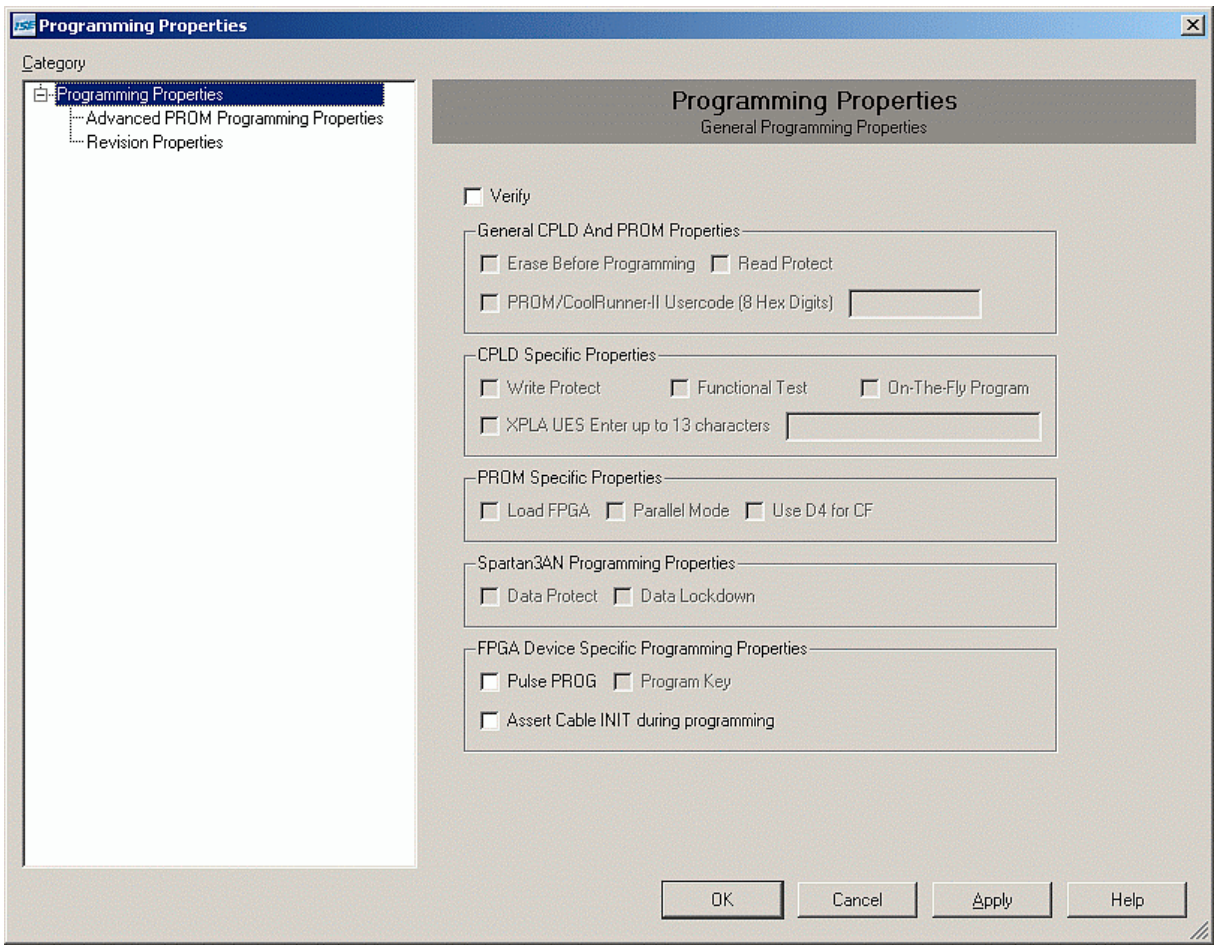
Selektieren Sie "mproz.bit" und clicken "Open"



Clicken Sie OK



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



Deselektieren Sie **"Verify"** und clicken 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 Dateienamen 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 erhoeht
even    : Current Location Pointer durch Ausgabe von 0x00 auf
          den naechsten geraden Wert erhoeht
even n  : Current Location Pointer durch Ausgabe von 0x00 auf
          das naechste Vielfache von n erhoeht
even n,m: Current Location Pointer durch Ausgabe von m auf
          das naechste Vielfache von n erhoeht
if WERT : Assemblierung wird ausgefuehrt falls WERT <> 0
else      (WERT muss bereits im PASS1 berechnet werden koennen)
endif
```

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

label_block: definiert einen neuen Gueltigkeitbereich 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 Gueltigkeits-
bereich erstreckt sich nur auf den Bereich zwischen zwei normalen
Label. Dadurch kann der gleiche lokale Labelname an verschieden
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 Ausdruckes, der nach dem = folgt. (abc: und abc=@ sind gleich-
wertig; 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 Schluessel-
wortes 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!

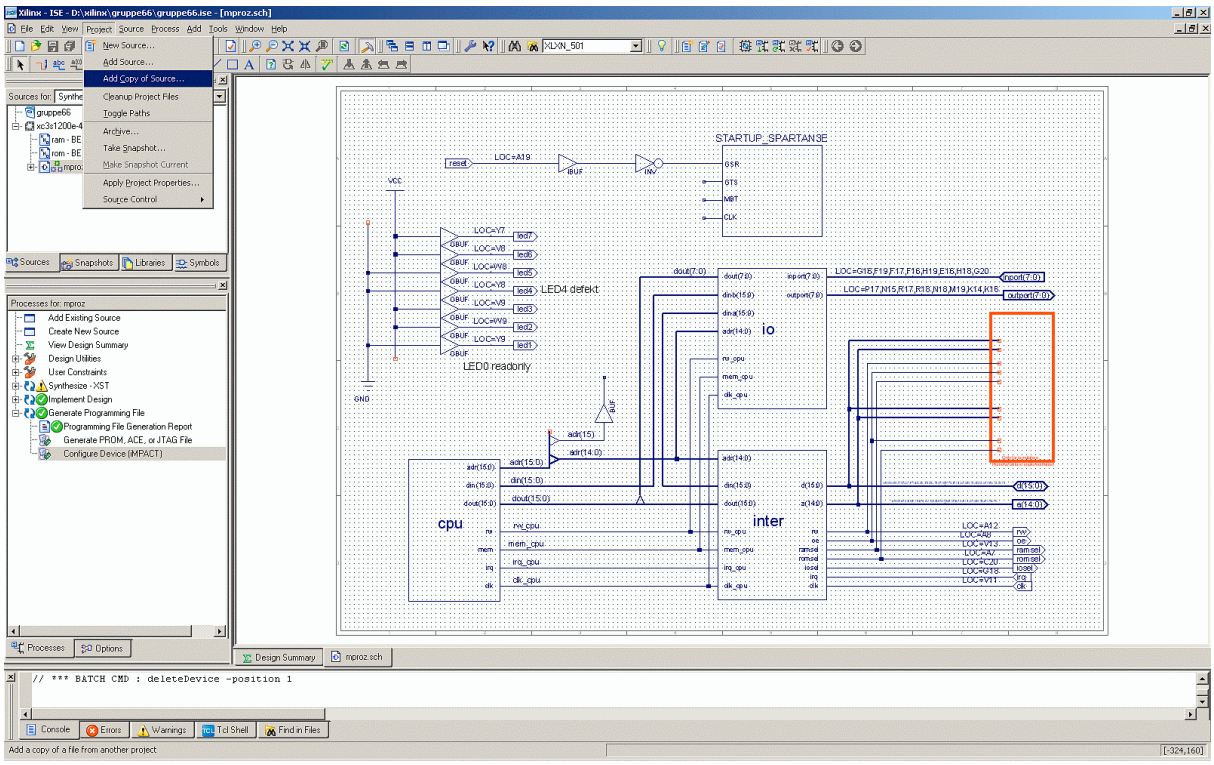
Beginnen normale oder globale Labels mit einem ".", so wird durch diese Labels kein neuer Gueltigkeitsbereich fuer lokale Labels erzeugt.

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

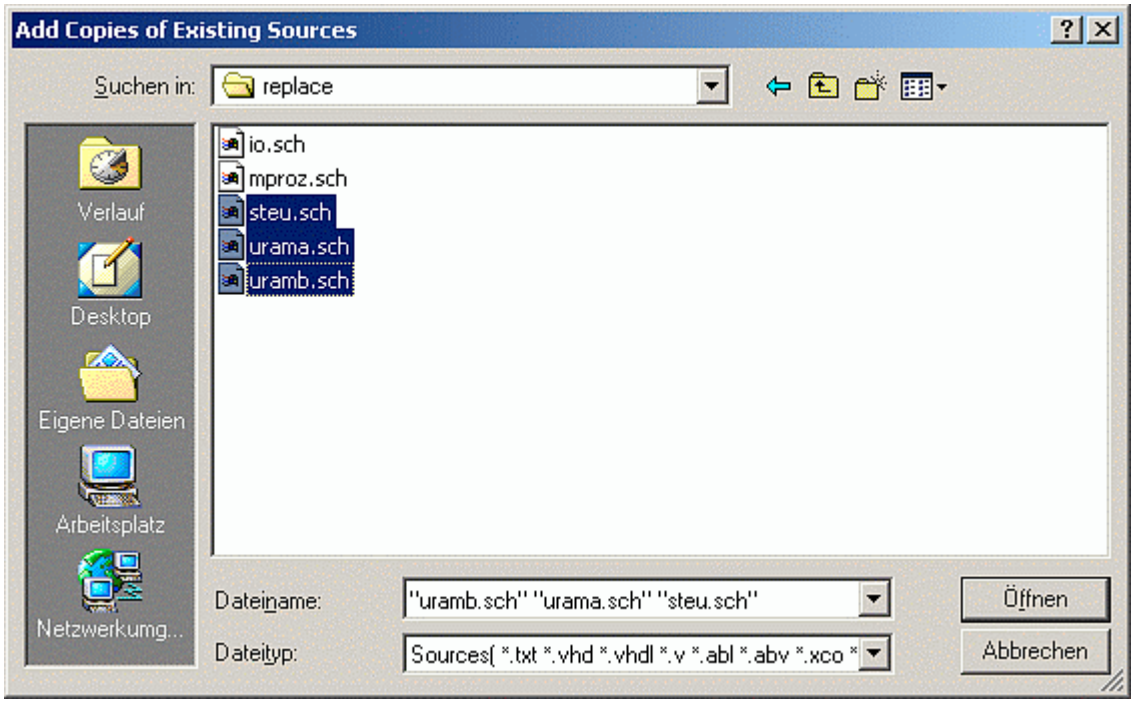
Das Label mit dem Namen @@ ist ein reserviertes Label und stellt den Ausgabe File Pointer dar. @@ darf innerhalb des Programms nicht veraendert werden.

Anhang F

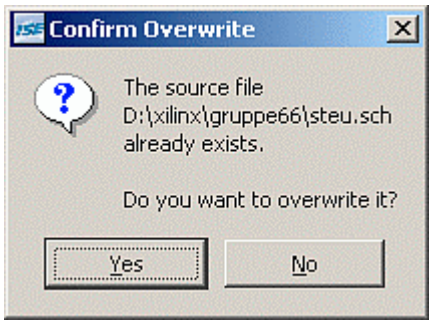
Mikroprogrammiertes Steuerwerk



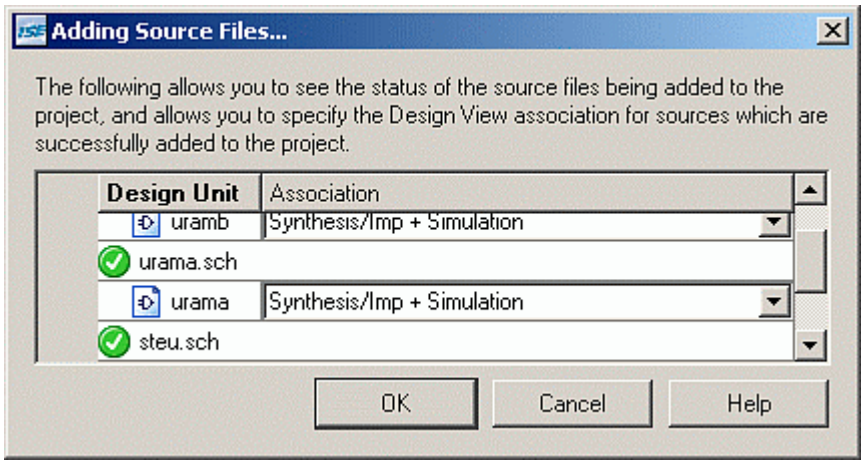
Fügen Sie RAM/ROM wieder ein und selektieren Sie im Menü: "Add Copy of Source"



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



Clicken Sie auf "Yes".



Clicken Sie auf "OK".

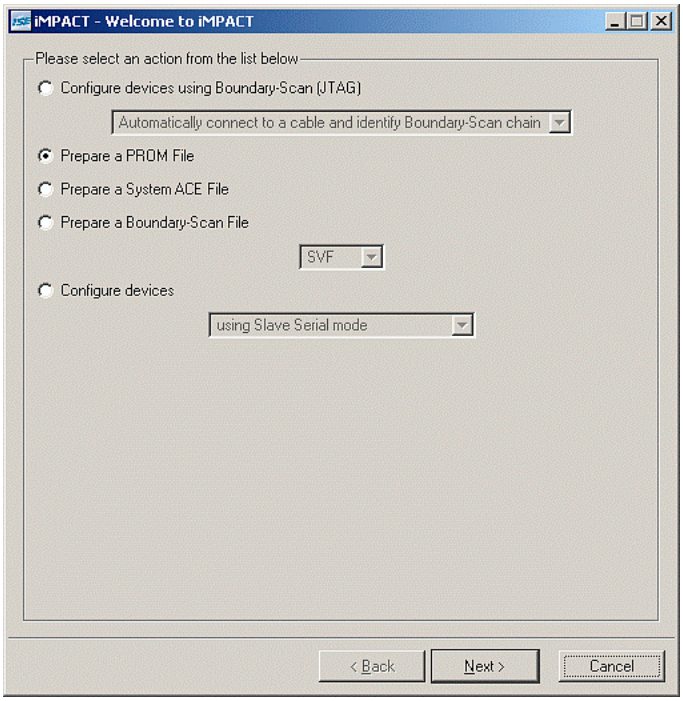
Vervollständigen Sie die Schaltung für das mikroprogrammierte Steuerwerk und geben Sie das entsprechende Mikroprogamm in den Mikroprogrammspeicher ein (Rechtsclick auf die Speicherbausteine -> Eigenschaft). Achten Sie darauf, immer vierstellige Hexzahlen einzugeben (führende Nullen mit eingeben.

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

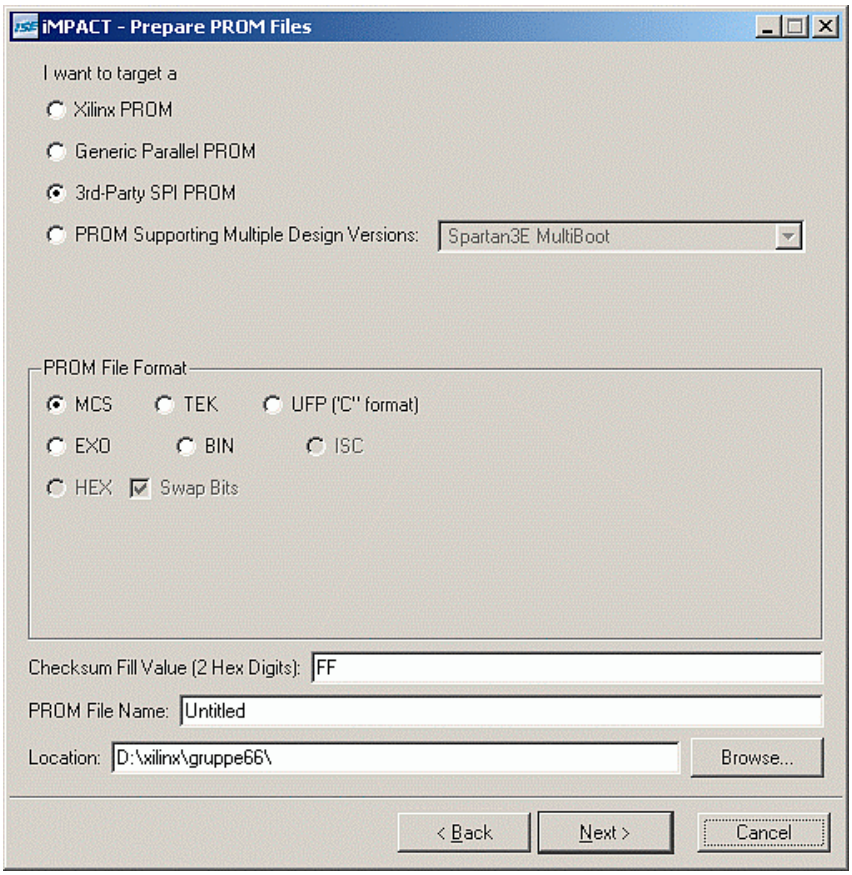
Anhang G (nicht für die Studenten)

Programmieren der Schaltung in das Atmel flash memory auf dem DARNAW1 Board.

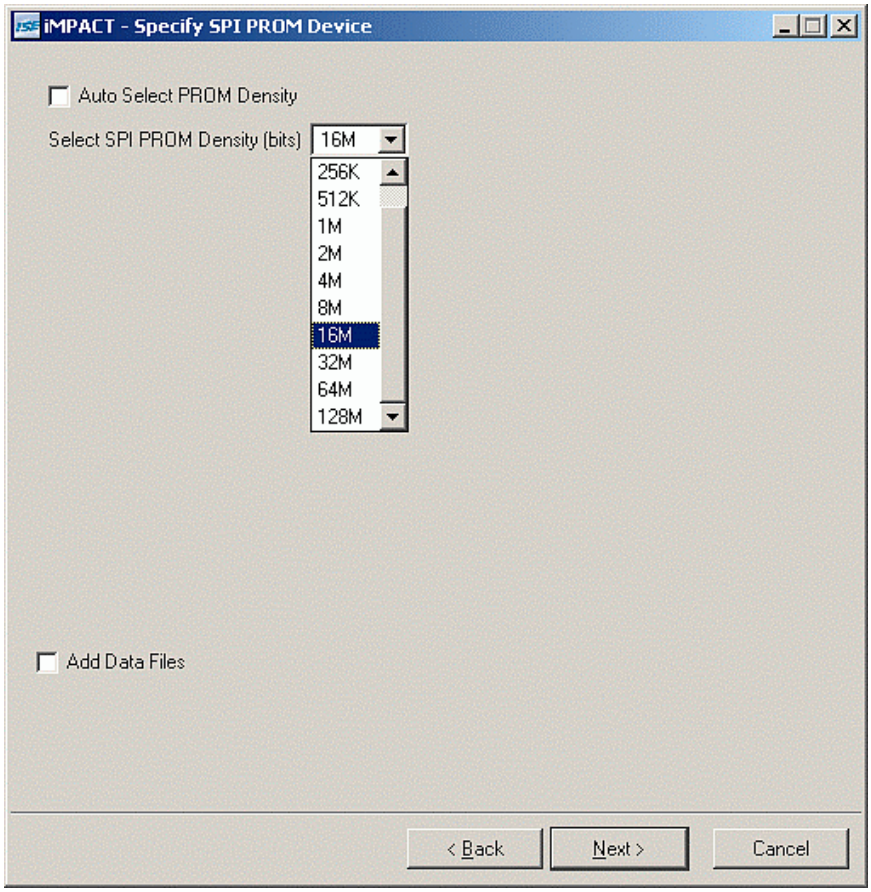
Wie in Anhang D beschrieben: "Generate Programming File" + "Configure Device (iMPACT)".



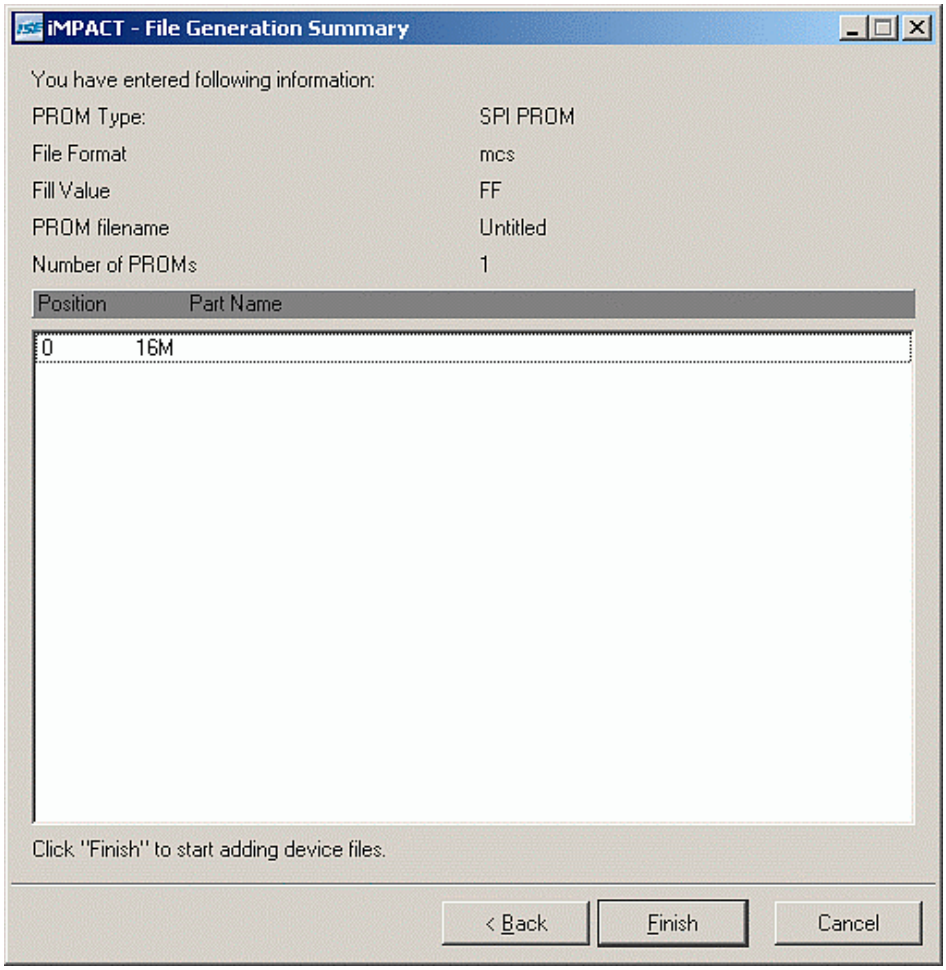
Dieses mal wählen Sie "Prepare a PROM file" und clicken auf "Next".



Wählen Sie "3rd-Party SPI PROM" und clicken auf "Next". Wählen Sie evtl. einen anderen Namen als "Untitled".

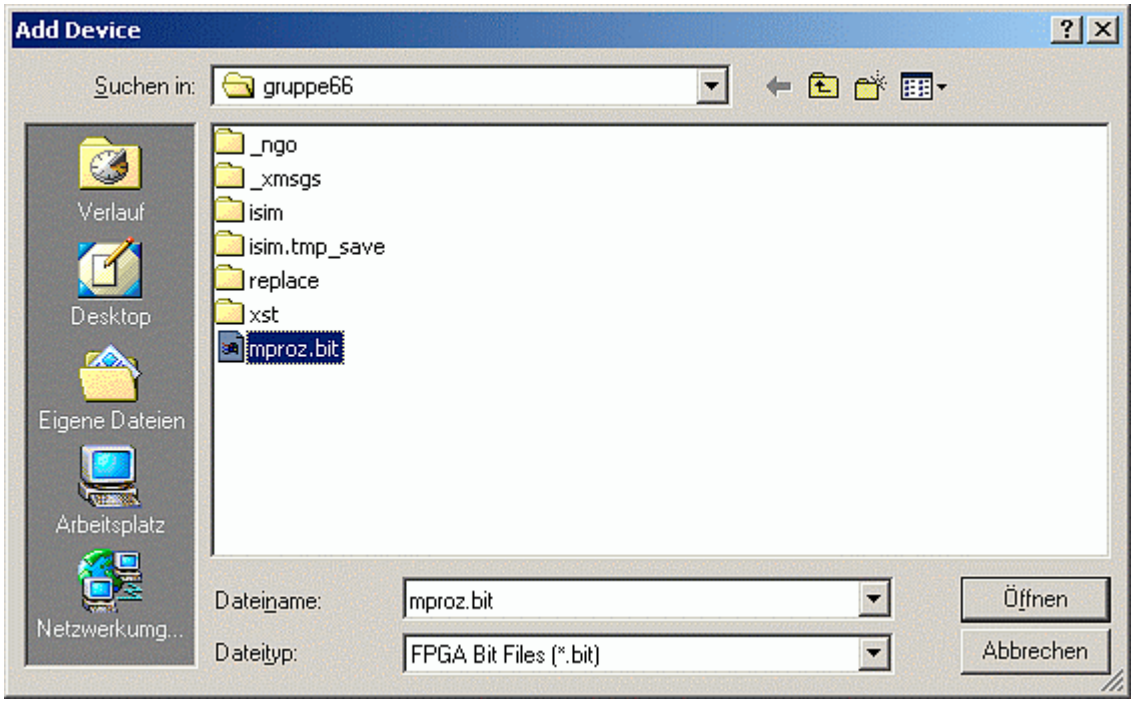


Selektieren Sie 16M und clicken aud "Next".

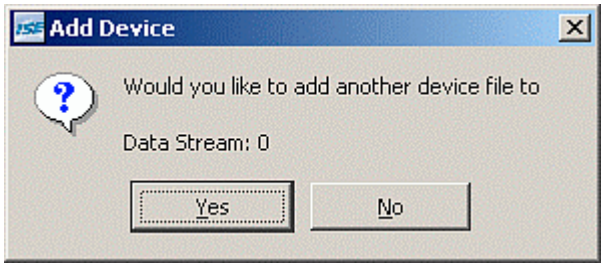


Clicken Sie auf "Finish".

Clicken Sie auf "OK".

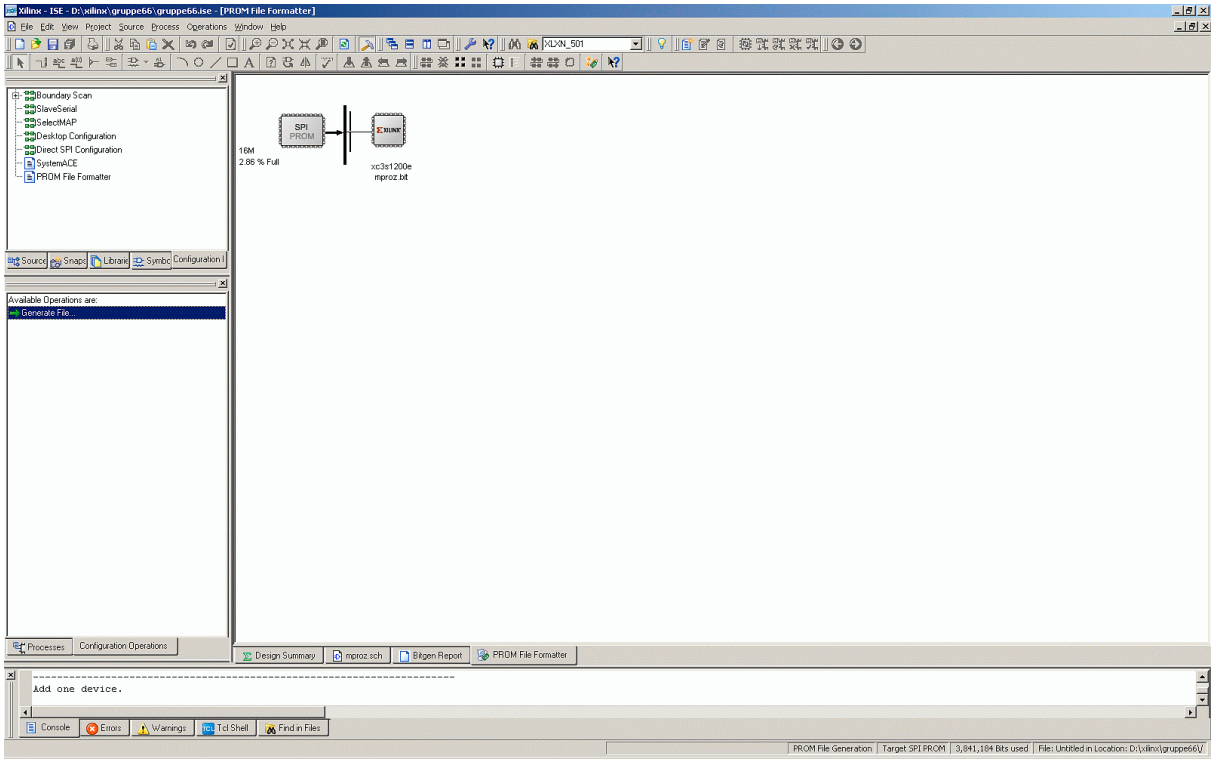


Selektieren Sie "mproz.bit" und clicken Sie auf "Open".

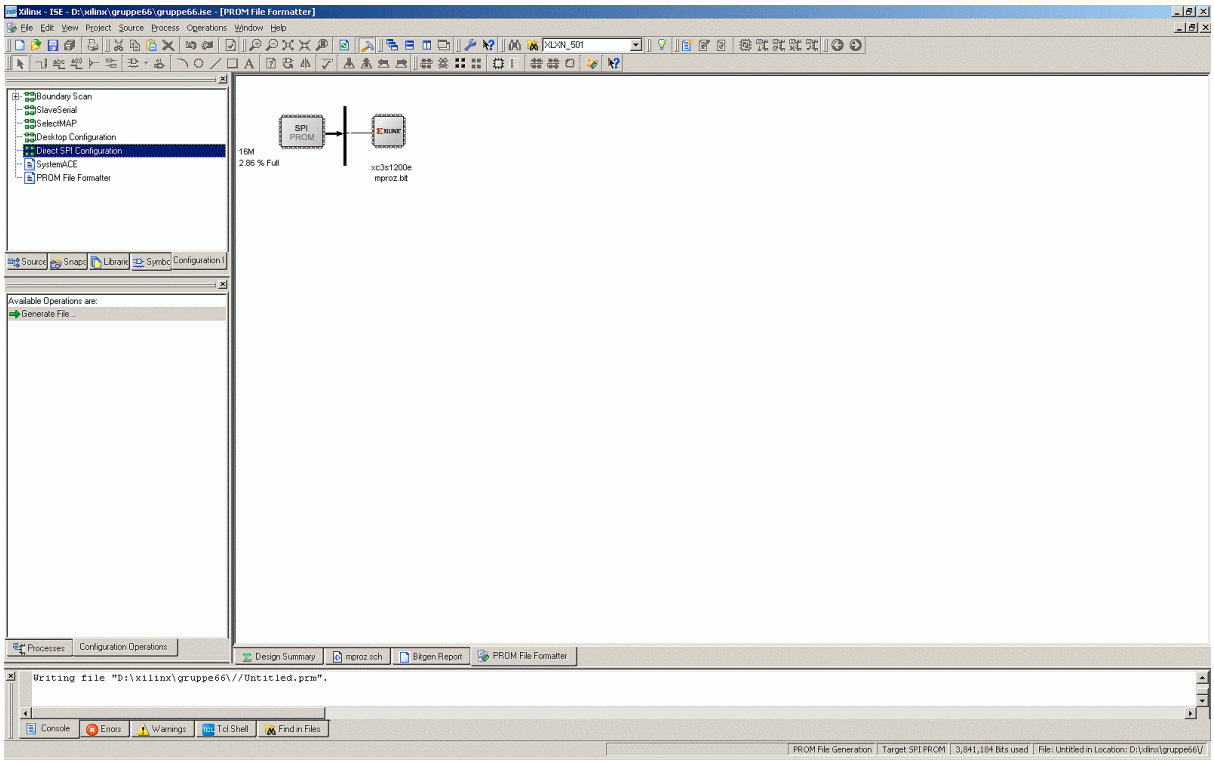


Clicken Sie auf "No".

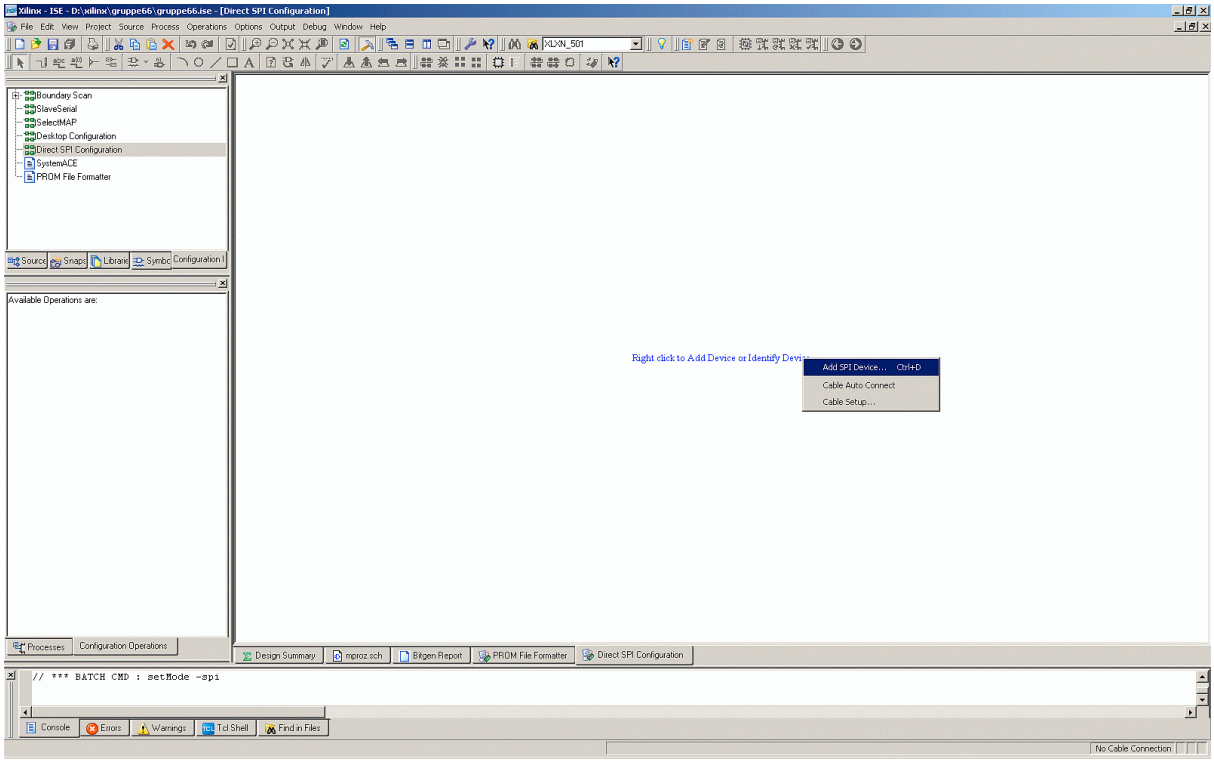
Clicken Sie auf "OK".



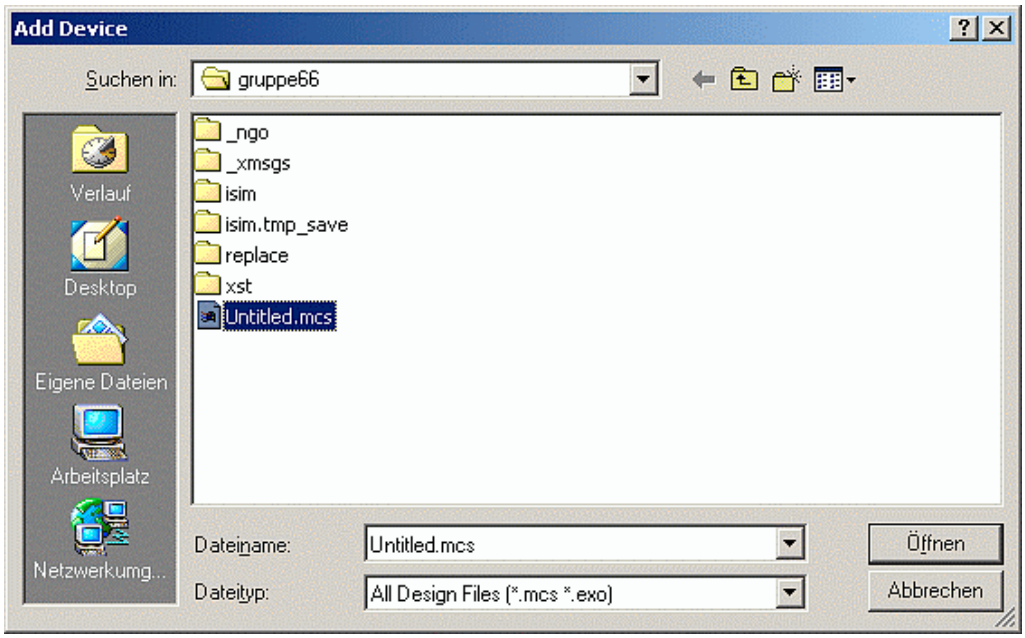
Doppelclicken Sie auf "Generate File".



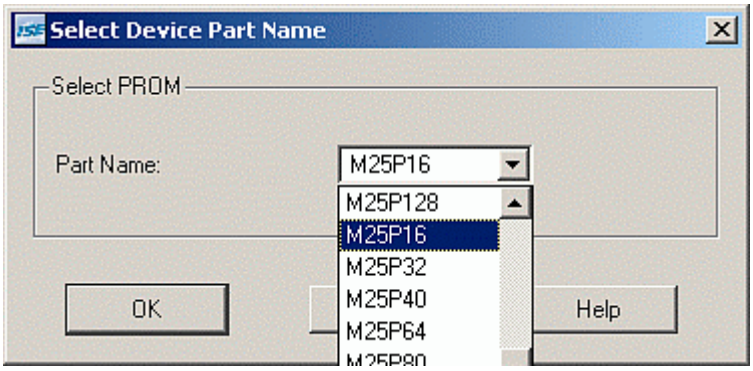
Doppelclicken Sie auf "Direct SPI configuration".



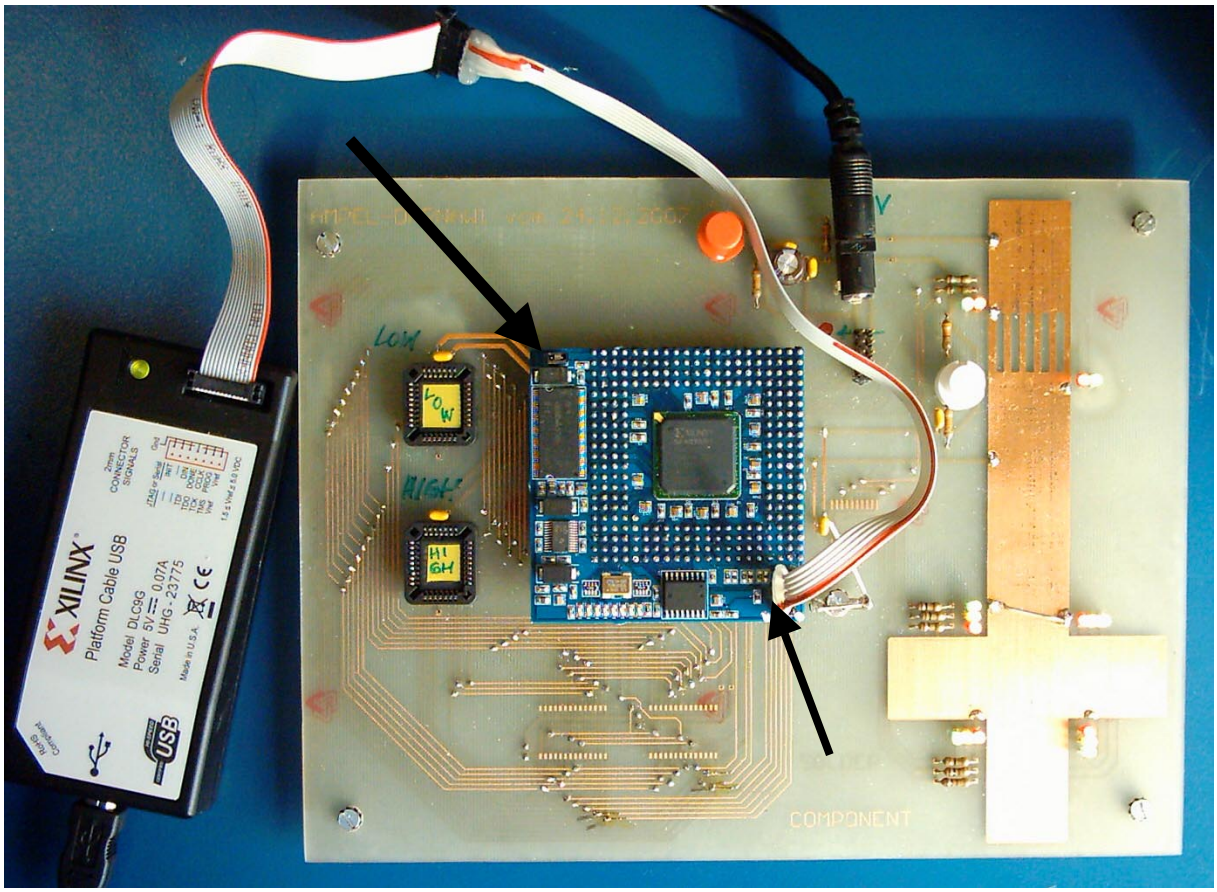
Rechtsclicken Sie auf "Add SPI Device".



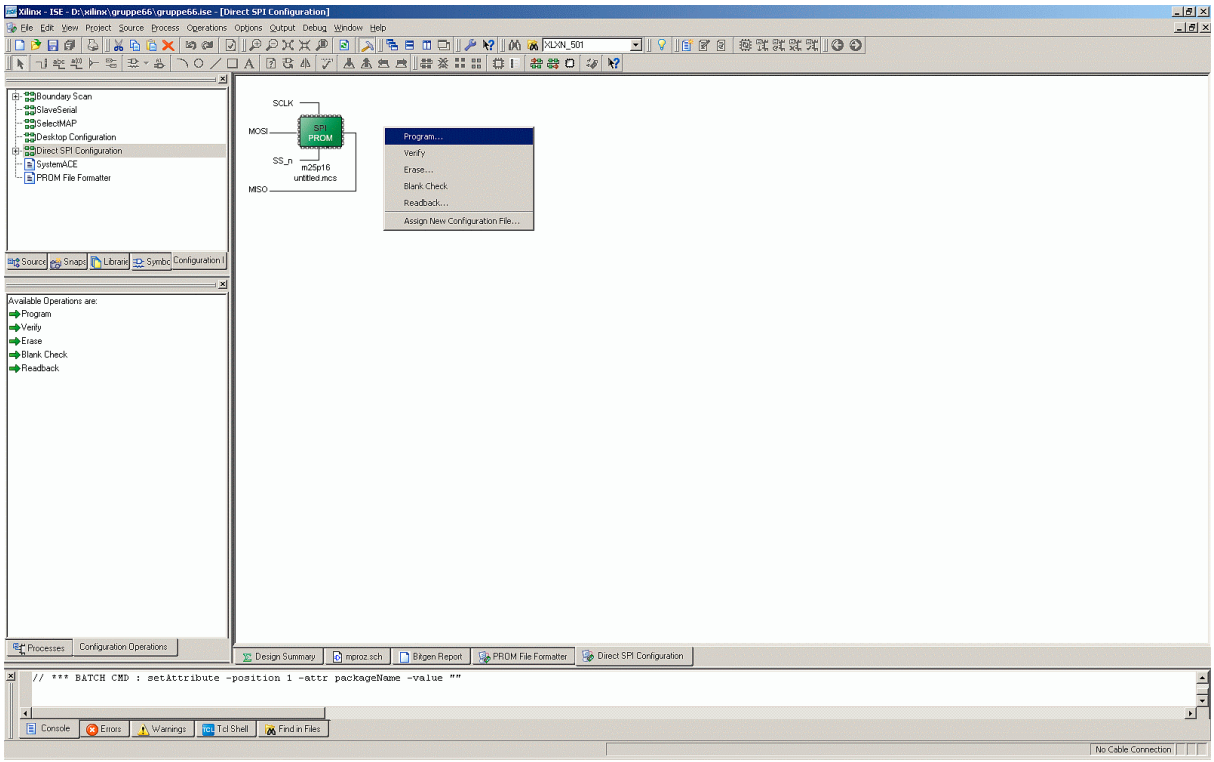
Selektieren Sie "Untiteled.msc" (oder den Namen den Sie gewählt haben) und clicken Sie auf "Open".



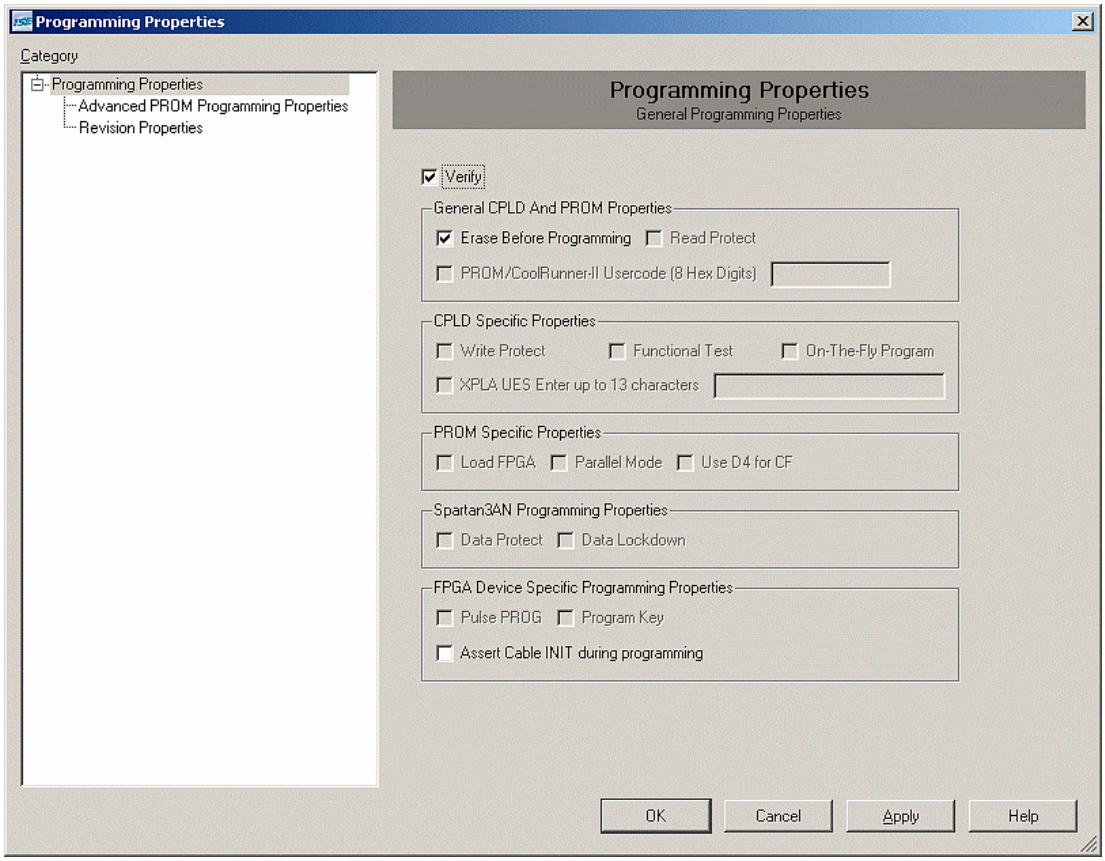
Selektieren Sie "M25P16" und clicken dann auf "OK".



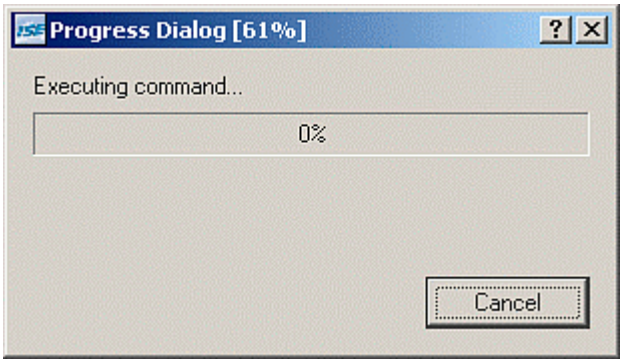
Setzen Sie den Jumper ein und verbinden das Downloadkabel mit der inneren Stiftleiste.



Rechtsclicken Sie auf das SPI PROM Symbol und selektieren Sie "Program..".



Clicken Sie auf "OK".



Warten Sie, bis die Programmierung beendet ist und entfernen dann das Downloadkabel und den Jumper..

Anhang H (nicht für Studenten)

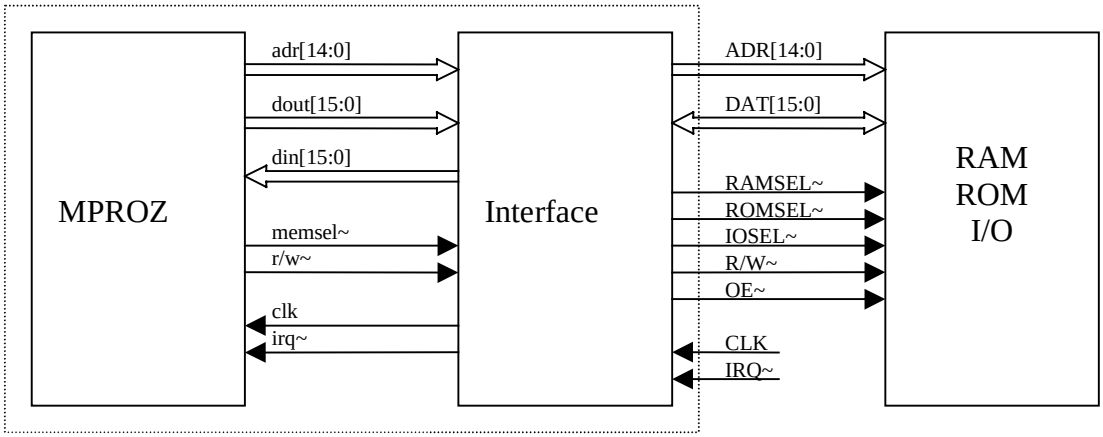
MPROZ – Ein 16-Bit Minimalprozessor

MPROZ ist eine vereinfachte Version des 16-Bit Prozessors XPROZ.

1. Prozessorarchitektur

1.1 Schnittstellenbeschreibung des Prozessors

ADR[14:0]	: 15-Bit Adreßbus
DAT[15:0]	: 16-Bit Datenbus
RAMSEL~	: Zugriff auf RAM
ROMSEL~	: Zugriff auf ROM
IOSEL~	: Zugriff auf I/O
R/W~	: Schreib- / Lesezugriff
OE~	: Output Enable
CLK	: Takt
IRQ~	: Interrupt Request



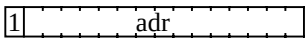
1.2 Register

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.

1.3 Befehlssatz

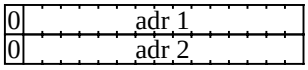
MPROZ unterstützt insgesamt drei Befehle:

br adr



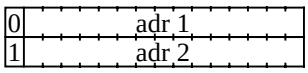
Lade adr in PC falls F=0
Lösche F

add adr1,adr2



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

nor adr1,adr2



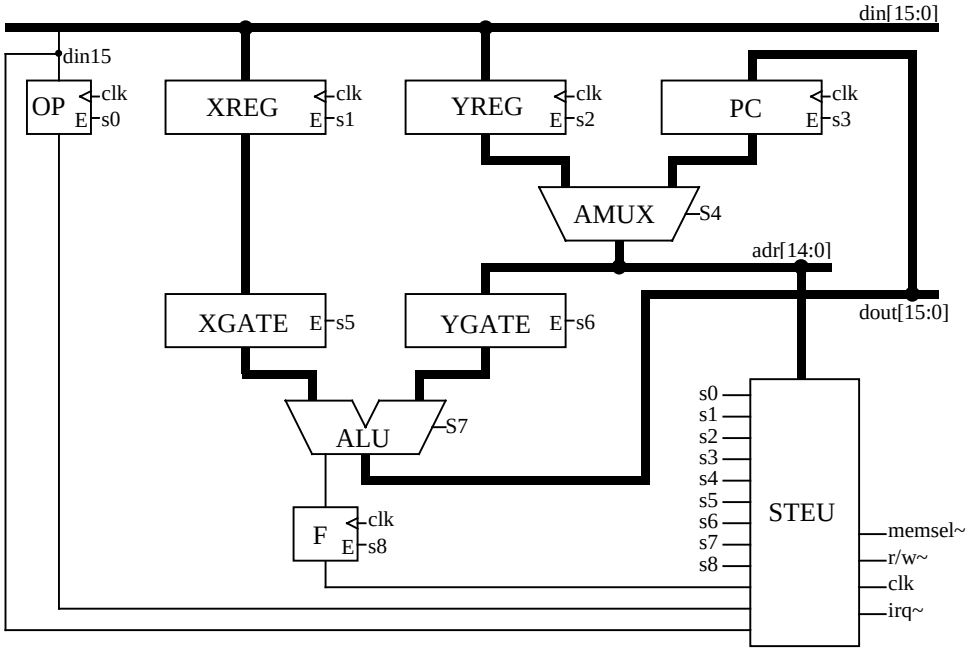
NOR-verknüpfe den Inhalt von adr1 mit dem Inhalt von adr2 und speichere das Ergebnis in adr2.
F=1 falls Ergebnis Null, sonst F=0.

1.4 Interrupt

Da es keinen eigenen Befehl zum Sperren und Freigeben des Interrupts gibt, wird das Interrupt Enable Flag (ie) durch einen Schreib- bzw. Lesezugriff auf die Adresse $4000 gelöscht bzw. gesetzt. Das Interrupt Request Signal (IRQ~) muß solange anliegen, bis der Interrupt bearbeitet wird. MPROZ bearbeitet einen Interrupt dann, wenn das Interrupt Enable Flag (ie) gesetzt ist und ein Sprungbefehl mit gelöschtem F-Flag (d.h. der Sprung würde ausgeführt) bearbeitet wird. Dadurch ist es nicht notwendig, das F-Flag und den Befehlszähler (PC) zu retten, sondern es genügt, den Sprungbefehl zu sichern. Dieser Sprungbefehl wird in der Speicherzelle gesichert, deren Adresse in Speicherzelle 1 steht (normalerweise Adresse $4000, da dann automatisch mit dem Sichern des Befehls das ie-Flag zurückgesetzt wird). Die Interruptroutine selbst beginnt an der Adresse 2. Die Rückkehr aus der Interruptroutine erfolgt durch einen Sprung zur Adresse $4000. Dort steht der gesicherte, durch den Interrupt unterbrochene Sprungbefehl, der nun erneut ausgeführt wird. Gleichzeitig wird durch das Lesen des Befehls aus Adresse $4000 auch das ie-Flag wieder gesetzt.

2. Implementierung

2.1 MPROZ



OP: 1-Bit Register
F: 1-Bit Register
PC: 15-Bit Register
XREG: 16-Bit Register
YREG: 16-Bit Register
AMUX: IF (s4=0) THEN out=in1 ELSE out=in2
XGATE: IF (s5=0) THEN out=$0000 ELSE out=in
YGATE: IF (s6=0) THEN out=$0001 ELSE out=in
ALU: IF (s7=0) THEN out=in1 ADD in2 , F=carry
ELSE out=in1 NOR in2 , F=zero
STEU: generiert Steuersignale s0-s8, memsel~ und r/w~

Zustand	Aktion
q0	PC→adr, PC+1→PC, din→XREG,YREG
q1	YREG→adr, din→XREG
q2	PC→adr, din→ YREG, din15→OP
q3	YREG→adr, din→YREG
q4	XREG [ADD|NOR] YREG → DOUT, [Carry|Zero] → F, DIN→XREG
q5	PC→adr, PC+1→PC, din→ YREG
q6	YREG→adr, XREG+0→dout
q7	[XREG | 1]+0→PC falls F=0, 0→F

int = irq~ + $\overline{ie}$ + F

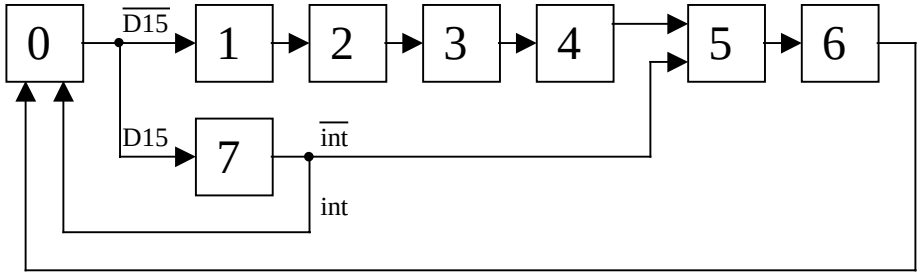
Zustand	s8	s7	s6	s5	s4	s3	s2	s1	s0	memsel	r/w~
q0	0	0	1	0	1	1	1	1	-	0	1
q1	-	-	-	-	0	0	-	1	-	0	1
q2	-	-	-	-	1	0	1	0	1	0	1
$q2^*$	-	0	1	0	1	1	1	0	1	0	1
q3	-	-	-	-	0	0	1	0	0	0	1
q4	1	OP	1	1	0	0	-	1	-	1	0
q5	0	0	1	0	1	1	1	0	-	0	1
q6	0	0	0	1	0	0	-	-	-	0	0
q7	1	0	0	int	-	$\overline{F}$	-	0	-	1	1

$q2^*$: für 3-Adress-Befehle anstatt 2-Adress-Befehle

$s0 = q2$ $\quad s3 = q0 + q5 + q7 \cdot \overline{F}$ $\quad s6 = \overline{q6 + q7}$ $\quad memsel\sim = q4 + q7$

$s1 = q0 + q1 + q4$ $\quad s4 = q0 + q2 + q5$ $\quad s7 = q4 \cdot OP$ $\quad r/w\sim = \overline{q4 + q6}$

$s2 = 1$ $\quad s5 = q4 + q6 + q7 \cdot int$ $\quad s8 = q4 + q7$



$d0 = q6 + q7 \cdot int$ $\quad d3 = q2$ $\quad d6 = q5$

$d1 = q0 \cdot \overline{D15}$ $\quad d4 = q3$ $\quad d7 = q0 \cdot D15$

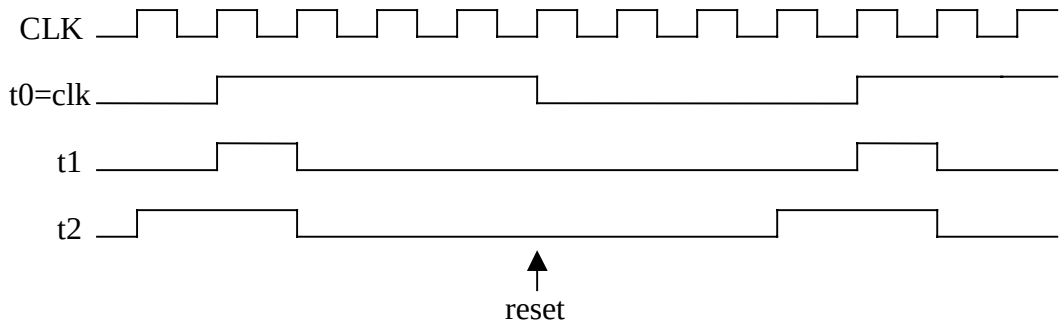
$d2 = q1$ $\quad d5 = q4 + q7 \cdot \overline{int}$

2.2 Interface

Das Interface ist zuständig für die Generierung der Ausgangssignale (RAMSEL~, ROMSEL~, IOSEL~, R/W~ und OE~).

Adreßbereich		
$0000 - $3fff	ROMSEL~	64 kByte
$4000 - $7dff	RAMSEL~	63 kByte
$7e00 - $7fff	IOSEL~	1 kByte

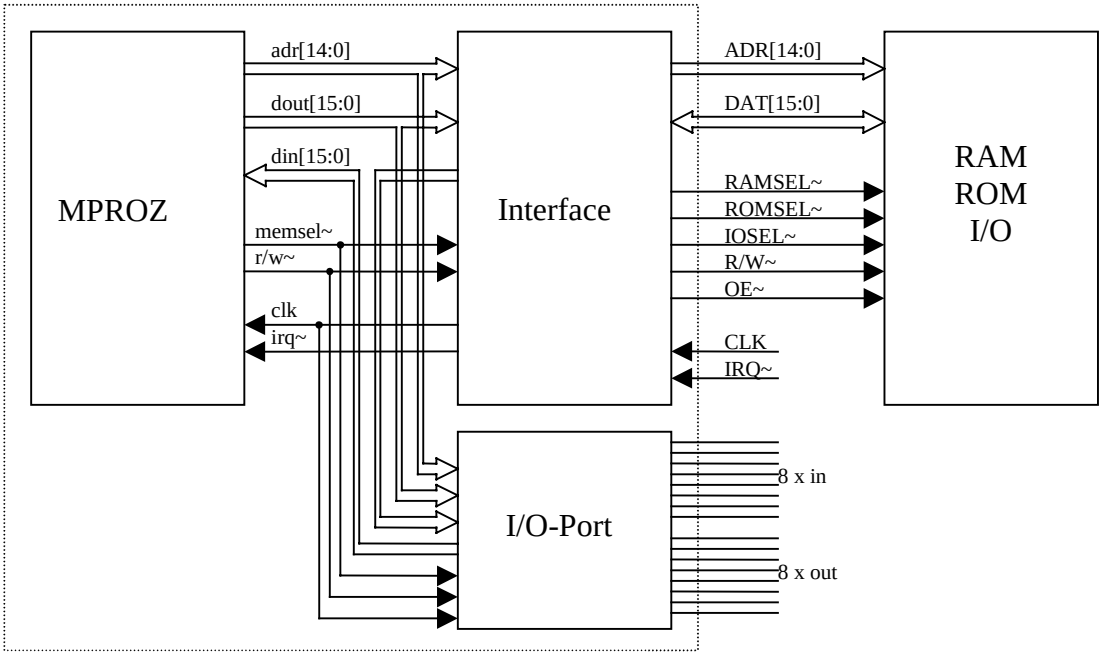
Für das exakte Timing wir das externe Taktsignal durch 8 geteilt.



4. Erweiterungen

4.1 Ein- / Ausgabe

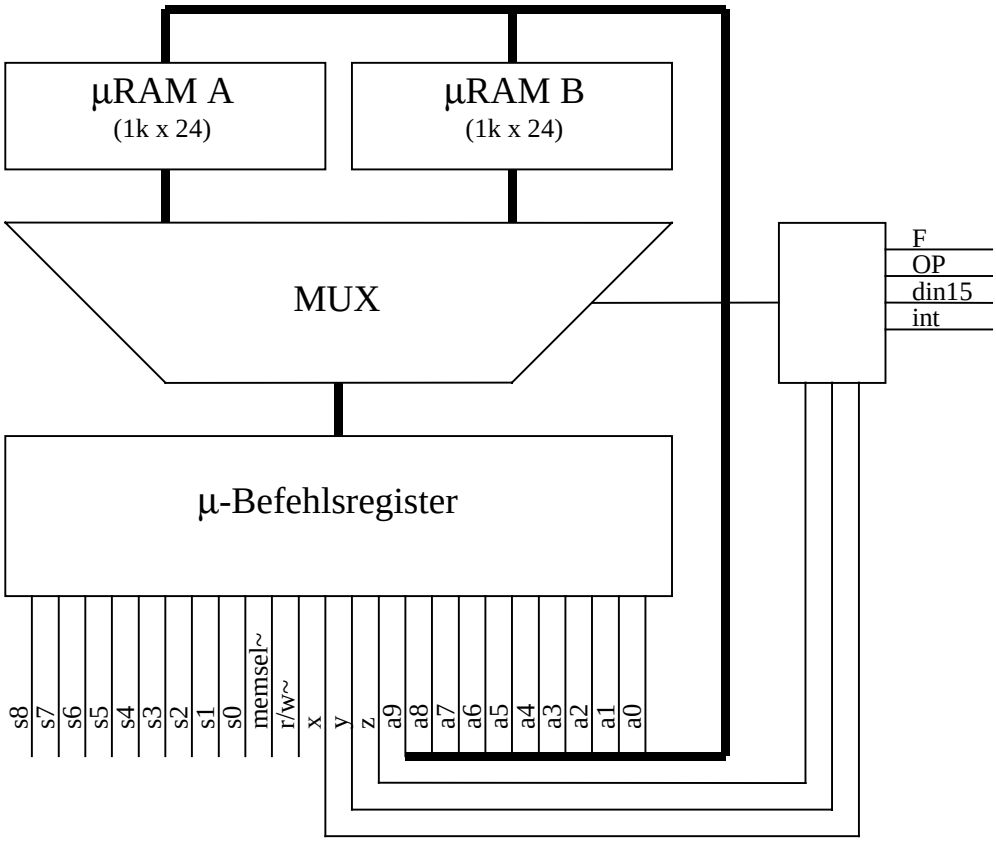
Um einfache Ein- /Ausgaben ohne zusätzliche Hardware machen zu können, wurde MPROZ um einen 8-Bit Eingangs- und einen 8-Bit Ausgangsport erweitert. Dieser Port wird unter der Adresse $7fff angesprochen. Bit 7-0 sind den Ausgangs- und Bit 15-8 den Eingangsleitungen zugeordnet. $7fff angesprochen. Bit 7-0 sind den Ausgangs- und Bit 15-8 den Eingangsleitungen zugeordnet.


MPROZ
adr[14:0]
dout[15:0]
din[15:0]
memsel~
r/w~
clk
irq~
Interface
I/O-Port
ADR[14:0]
DAT[15:0]
RAMSEL~
ROMSEL~
IOSEL~
R/W~
OE~
CLK
IRQ~
8 x in
8 x out
RAM
ROM
I/O

4.2 Mikroprogrammierung

4.2.1 Hardware

Um das Prinzip der Mikroprogrammierung zu zeigen, wurde das Steuerwerk des MPROZ auch in einer mikroprogrammierten Version entworfen:



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

4.2.1 Software

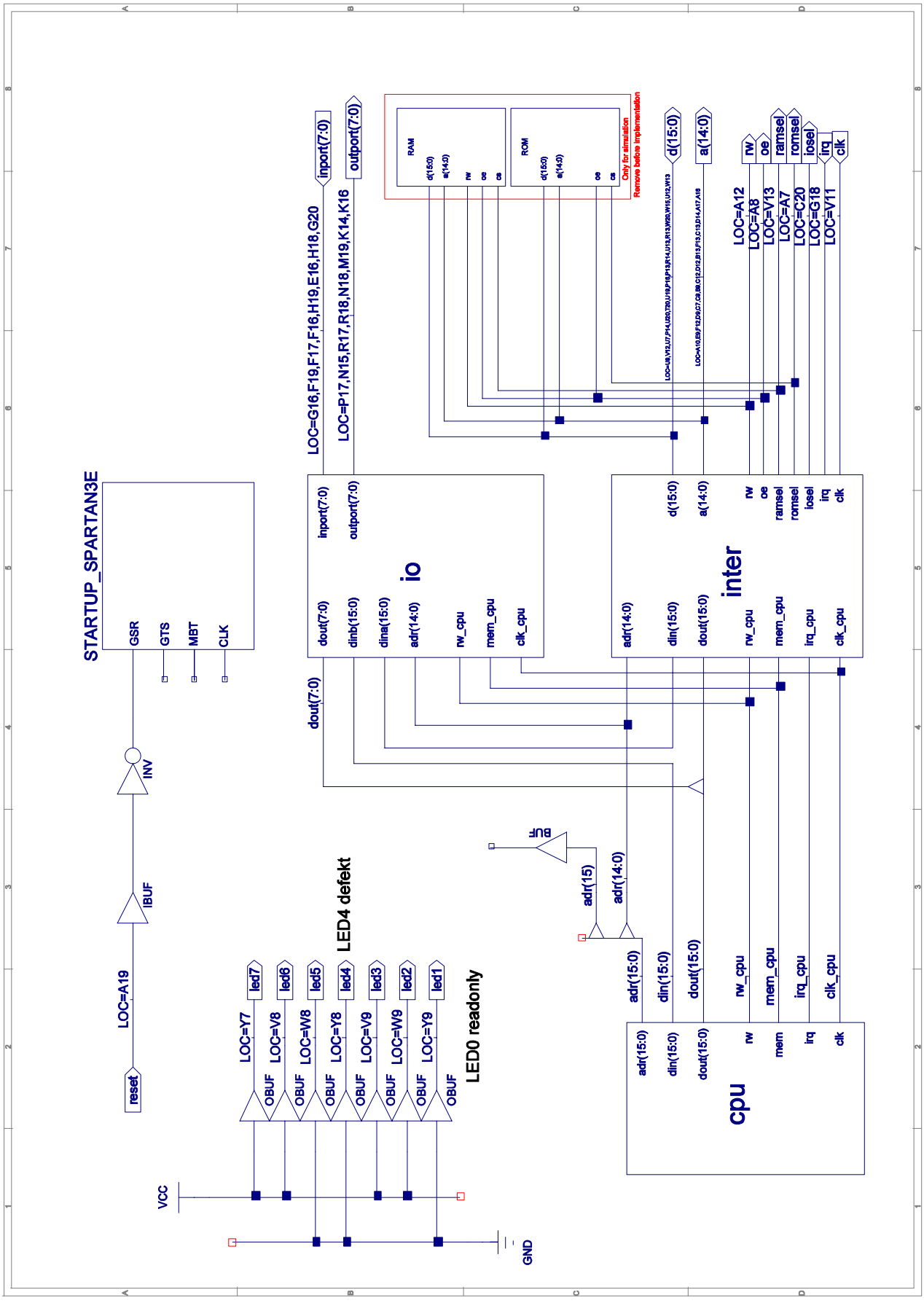
Das Mikroprogramm für das Steuerwerk des MPROZ:

	Mikrowort A						Mikrowort B							
Adresse	s8 s7 s6 s5	s4 s3 s2 s1	s0 memsel~ r/w~ x	y z A9 A8	A7 A6 A5 A4	A3 A2 A1 A0	s8 s7 s6 s5	s4 s3 s2 s1	s0 memsel~ r/w~ x	y z A9 A8	A7 A6 A5 A4	A3 A2 A1 A0	A	B
0	0010	1111	0011	1000	0000	0001	0010	1111	0011	1000	0000	0010	2f3801	2f3802
1	0000	0001	0010	0000	0000	0011	1001	0100	0111	1100	0000	0110	012003	947c06
2	0000	0001	0010	0000	0000	0011	1001	0000	0110	0000	0000	0000	012003	906000
3	0000	1010	1010	0100	0000	0011	0000	0010	0011	0100	0000	0100	0aa403	023404
4	1011	0001	0100	0000	0000	0101	1111	0001	0100	0000	0000	0101	b14005	f14005
5	0010	1110	0010	0100	0000	0101	0001	0000	0001	0000	0000	0000	2e2405	101000
6	1000	0100	0110	0000	0000	0101	0010	1111	0011	1000	0000	0001	846005	2f3801

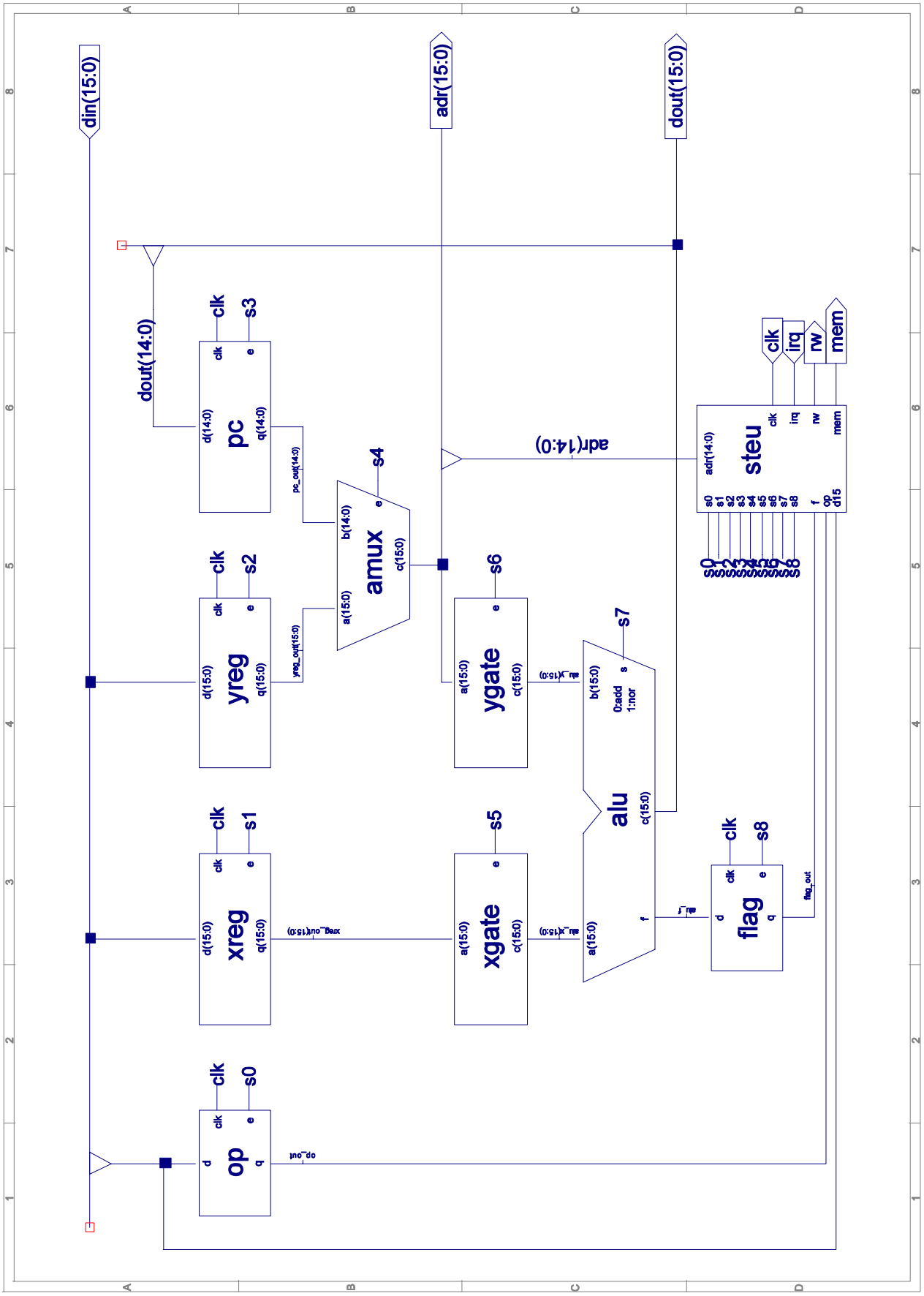
	0	0	1	0	9	1	9	7	8	0	f	1	1	8	0	0	0	0	0	0	0	0	e	f	6	0	1	6	1	3	9	1	0	6	f	b	3	a	0	0	0	0	0	0	0	a	3	0
	5	0	3	1	2	6	2	1	0	5	6	0	0	2	0	0	0	0	0	0	0	7	0	7	1	1	5	3	4	4	4	5	0	1	4	6	4	0	0	0	0	0	0	0	0	1	0	5
	0	0	0	0	0	0	0	0	0	0	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5. Schaltpläne

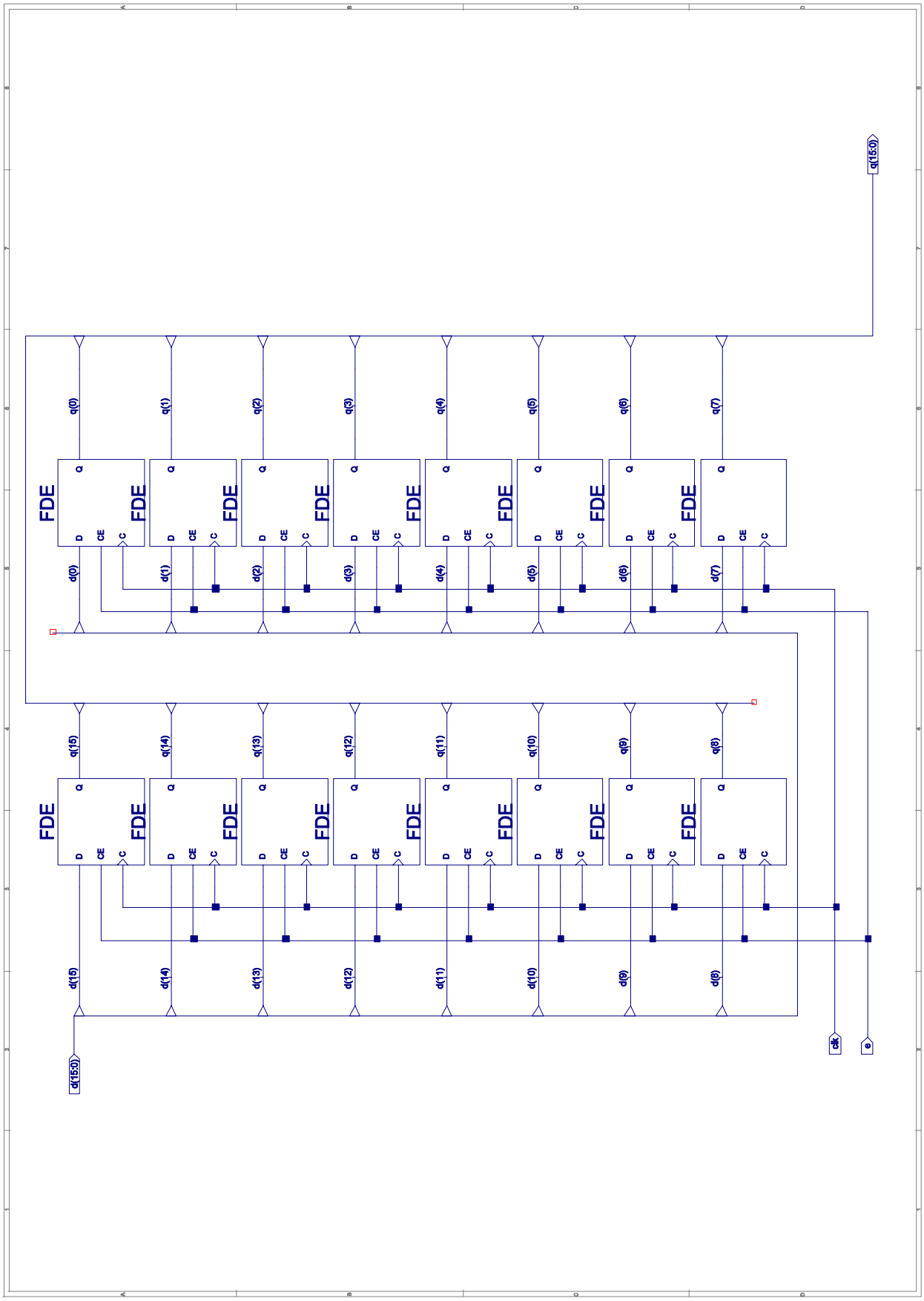
5.1 mproz.sch



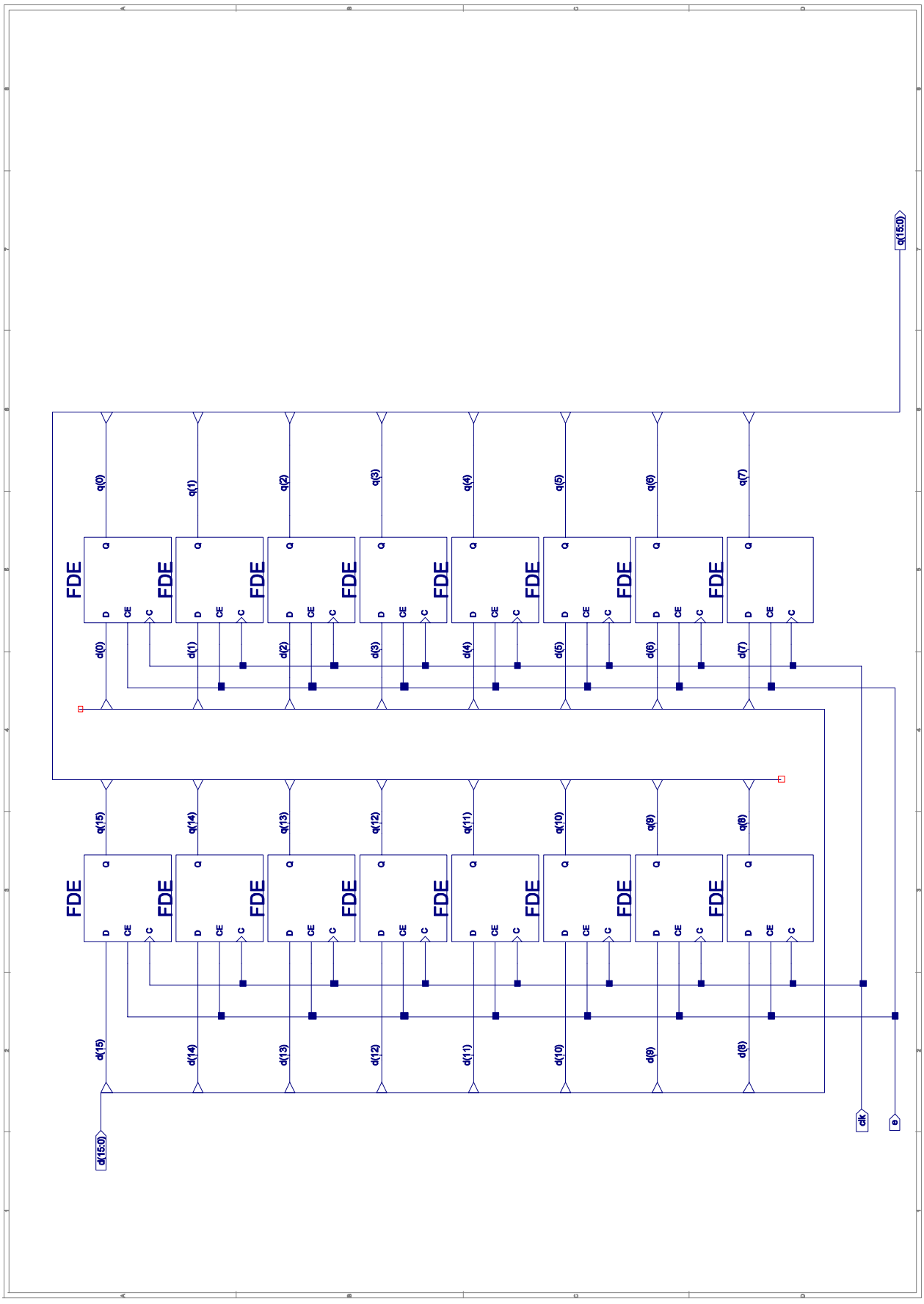
5.2 cpu.sch


din(15:0)
adr(15:0)
dout(15:0)
dout(14:0)
adr(14:0)
clk
irq
rw
mem
pc
pc_out(14:0)
amux
yreg
yreg_out(15:0)
ygate
alu
xreg
xreg_out(15:0)
xgate
flag
flag_out
op
op_out
steu
s0
s1
s2
s3
s4
s5
s6
s7
s8
0:add
1:nor

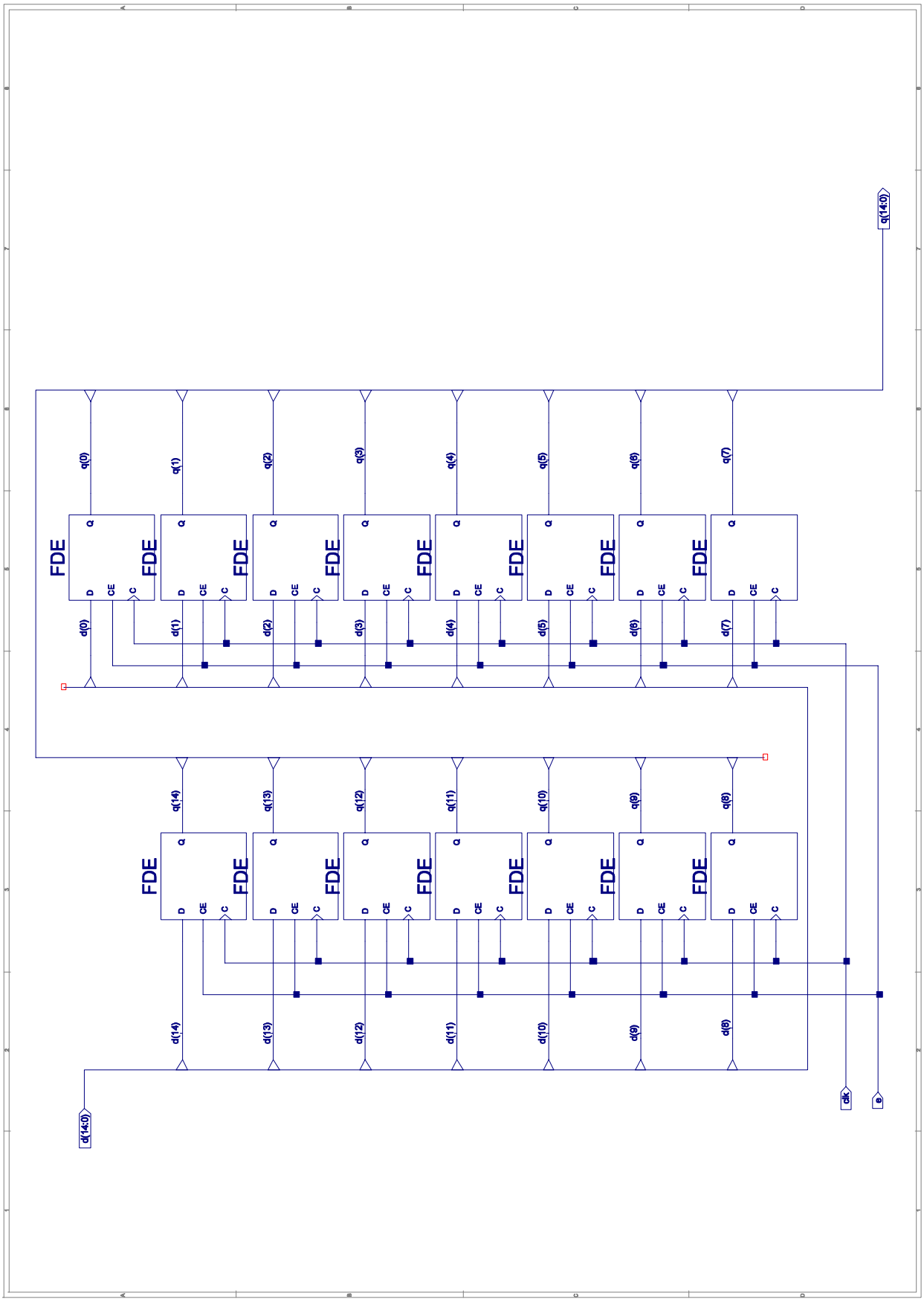
5.3 xreg.sch


q(15:0)
FDE
Q
D
CE
C
q(0)
q(1)
q(2)
q(3)
q(4)
q(5)
q(6)
q(7)
d(0)
d(1)
d(2)
d(3)
d(4)
d(5)
d(6)
d(7)
q(15)
q(14)
q(13)
q(12)
q(11)
q(10)
q(9)
q(8)
d(15)
d(14)
d(13)
d(12)
d(11)
d(10)
d(9)
d(8)
d(15:0)
clk
e

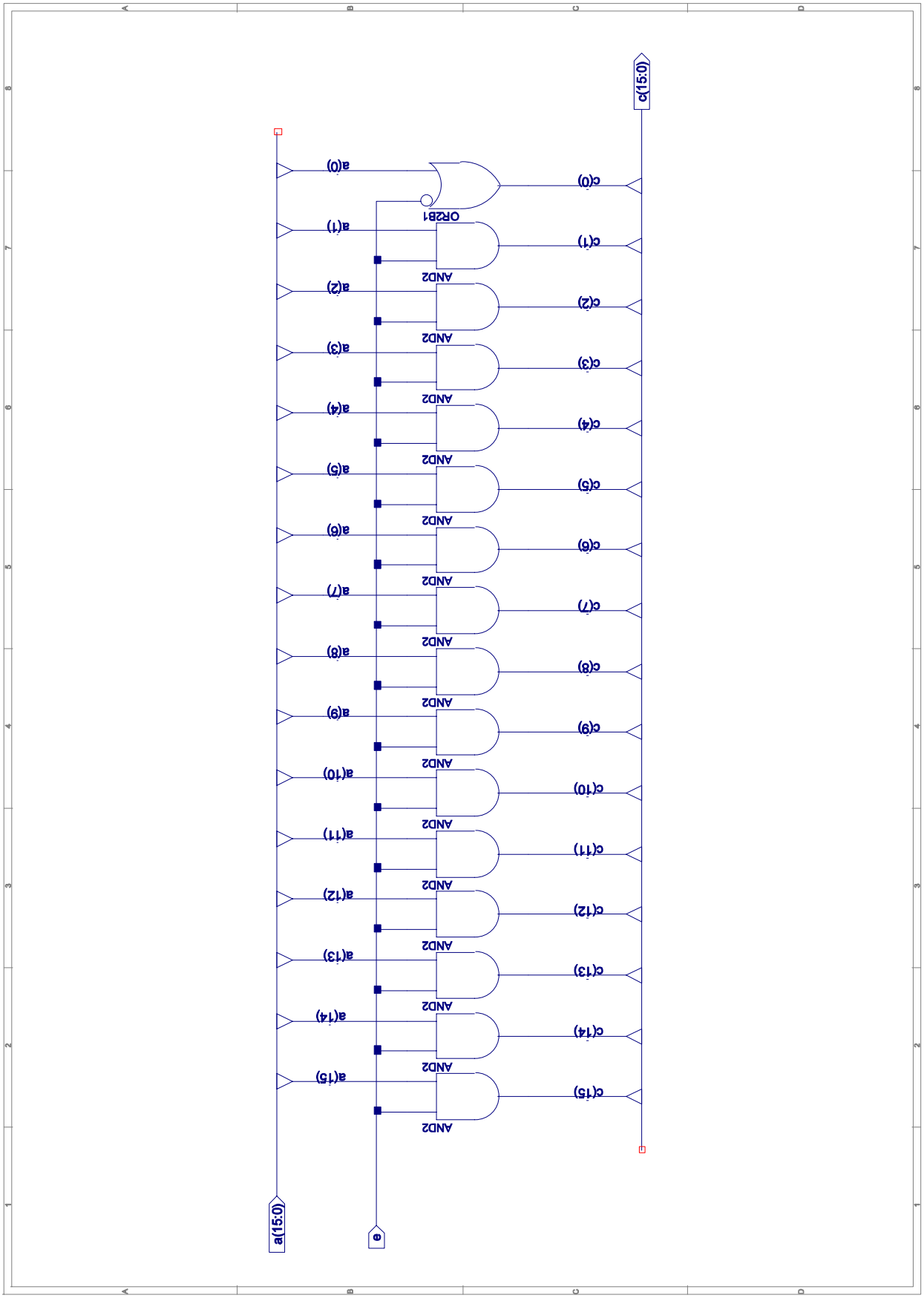
5.4 yreg.sch


q(15:0)
q(0)
q(1)
q(2)
q(3)
q(4)
q(5)
q(6)
q(7)
FDE
Q
D
CE
C
d(0)
d(1)
d(2)
d(3)
d(4)
d(5)
d(6)
d(7)
q(15)
q(14)
q(13)
q(12)
q(11)
q(10)
q(9)
q(8)
d(15)
d(14)
d(13)
d(12)
d(11)
d(10)
d(9)
d(8)
d(15:0)
clk
e

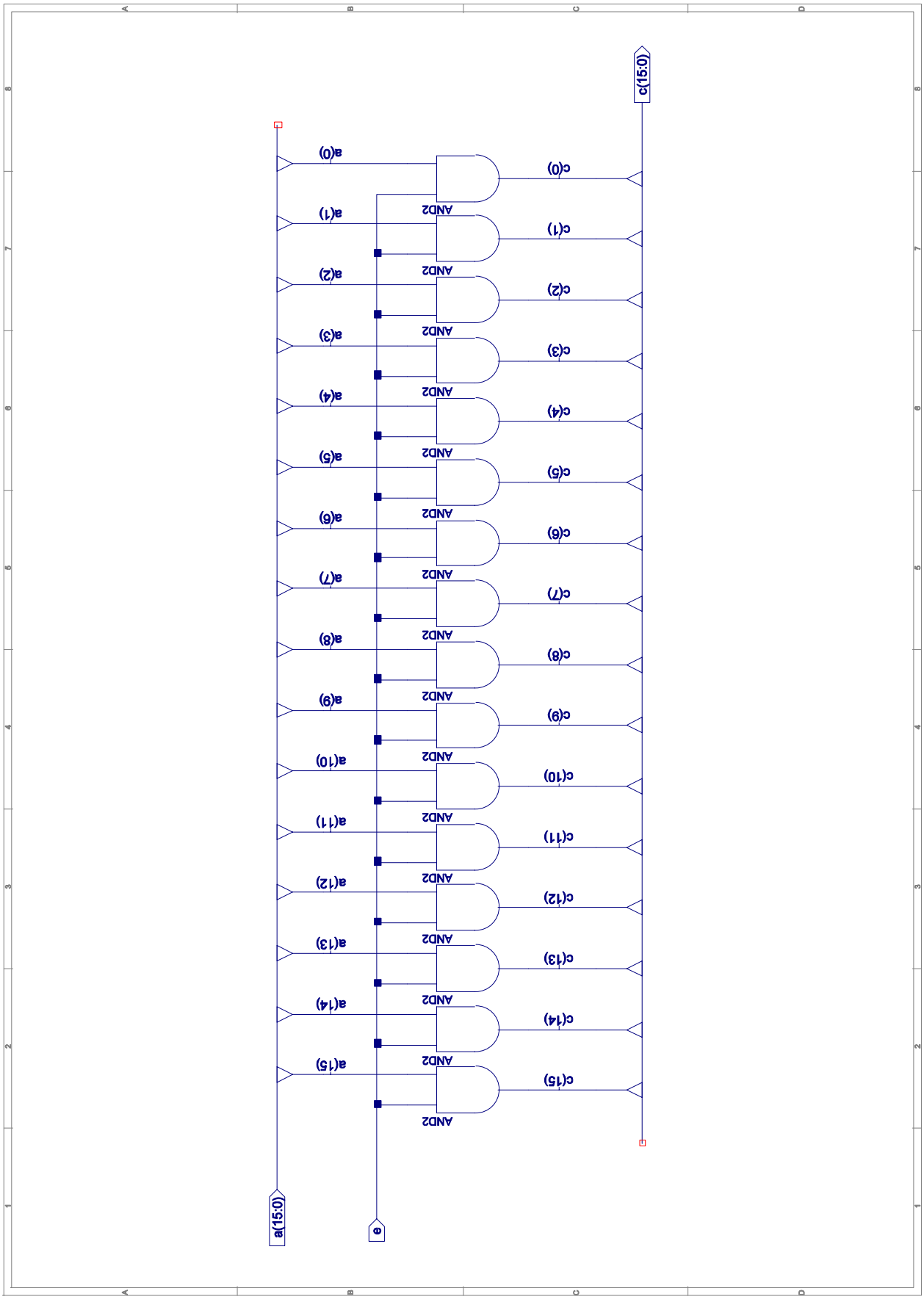
5.5 pc.sch


q(14:0)
FDE
Q
D
CE
C
q(0)
q(1)
q(2)
q(3)
q(4)
q(5)
q(6)
q(7)
d(0)
d(1)
d(2)
d(3)
d(4)
d(5)
d(6)
d(7)
q(14)
q(13)
q(12)
q(11)
q(10)
q(9)
q(8)
d(14)
d(13)
d(12)
d(11)
d(10)
d(9)
d(8)
d(14:0)
clk
ce

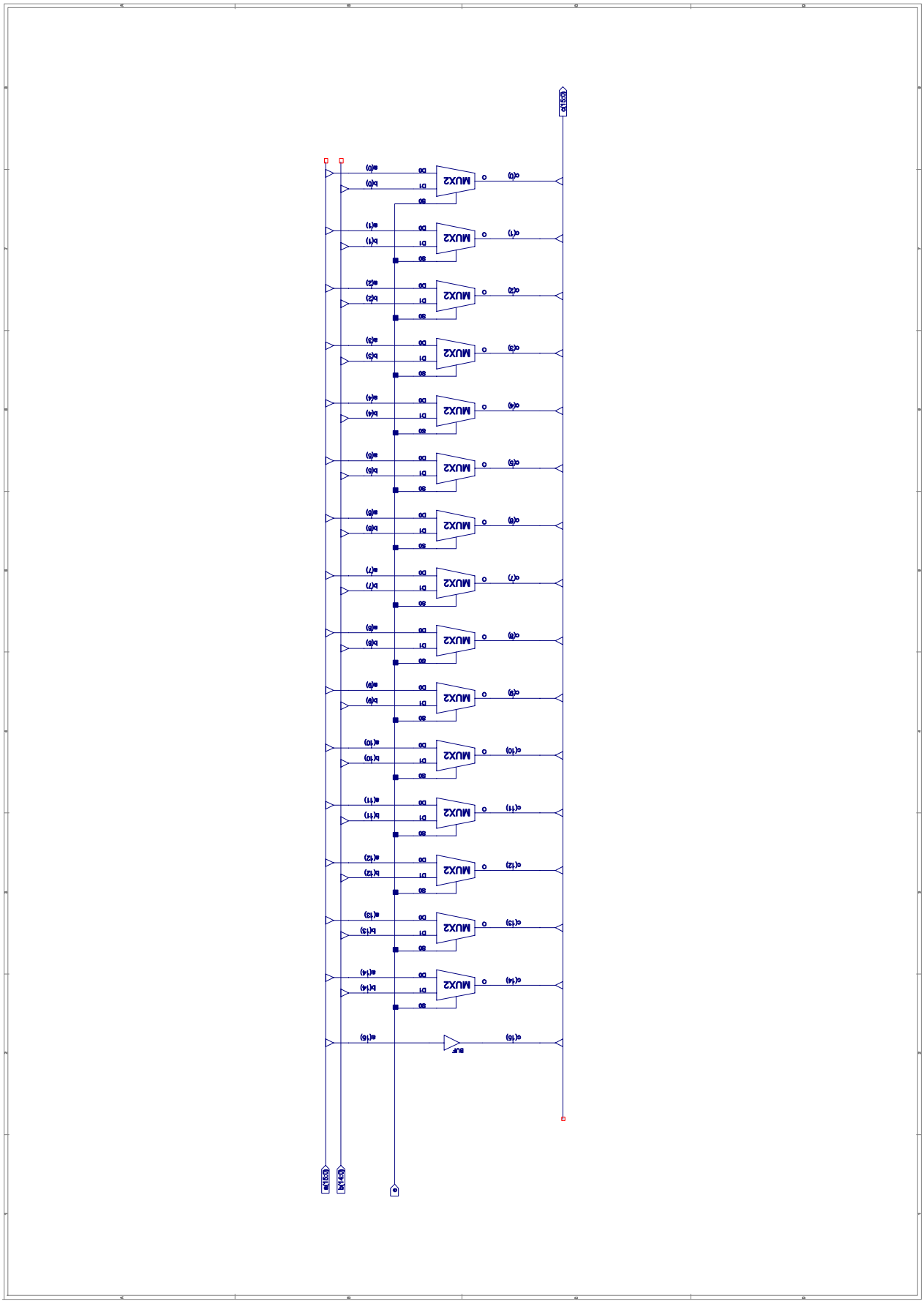
5.6 xgate.sch


c(15:0)
a(0)
c(0)
OR2B1
a(1)
c(1)
AND2
a(2)
c(2)
AND2
a(3)
c(3)
AND2
a(4)
c(4)
AND2
a(5)
c(5)
AND2
a(6)
c(6)
AND2
a(7)
c(7)
AND2
a(8)
c(8)
AND2
a(9)
c(9)
AND2
a(10)
c(10)
AND2
a(11)
c(11)
AND2
a(12)
c(12)
AND2
a(13)
c(13)
AND2
a(14)
c(14)
AND2
a(15)
c(15)
AND2
a(15:0)
a

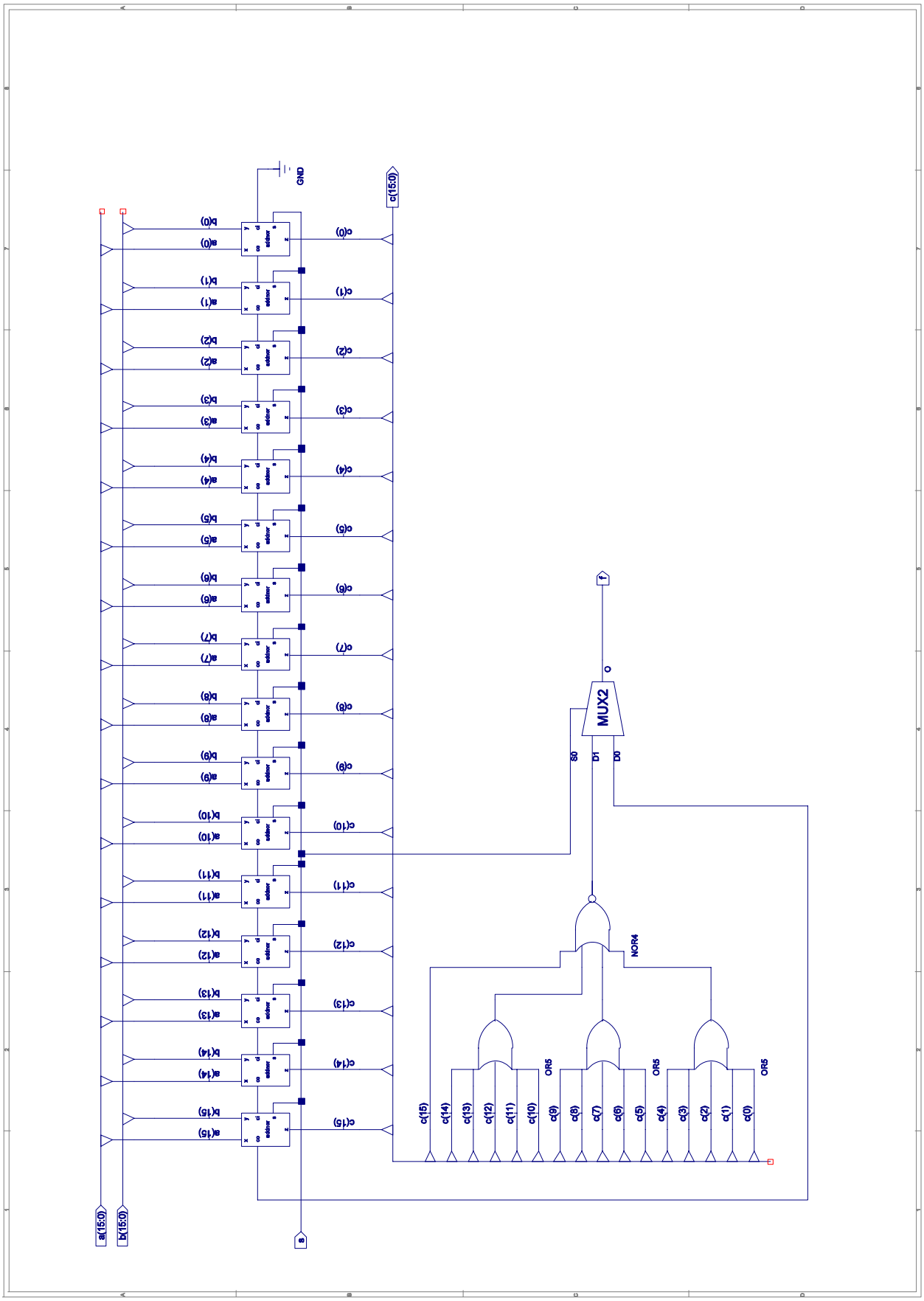
5.7 ygate.sch


c(15:0)
a(0)
c(0)
AND2
a(1)
c(1)
AND2
a(2)
c(2)
AND2
a(3)
c(3)
AND2
a(4)
c(4)
AND2
a(5)
c(5)
AND2
a(6)
c(6)
AND2
a(7)
c(7)
AND2
a(8)
c(8)
AND2
a(9)
c(9)
AND2
a(10)
c(10)
AND2
a(11)
c(11)
AND2
a(12)
c(12)
AND2
a(13)
c(13)
AND2
a(14)
c(14)
AND2
a(15)
c(15)
AND2
a(15:0)
e

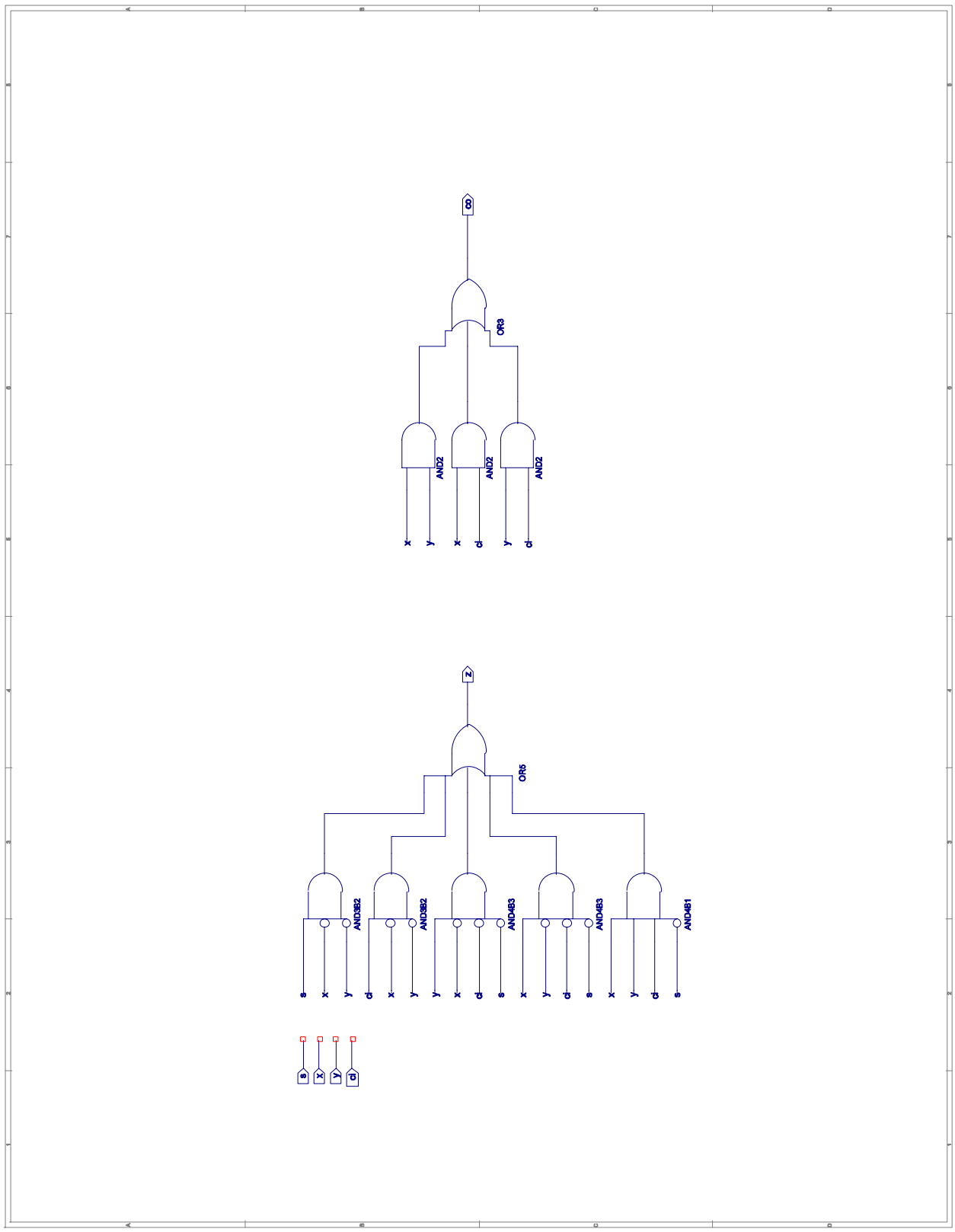
5.8 amux.sch



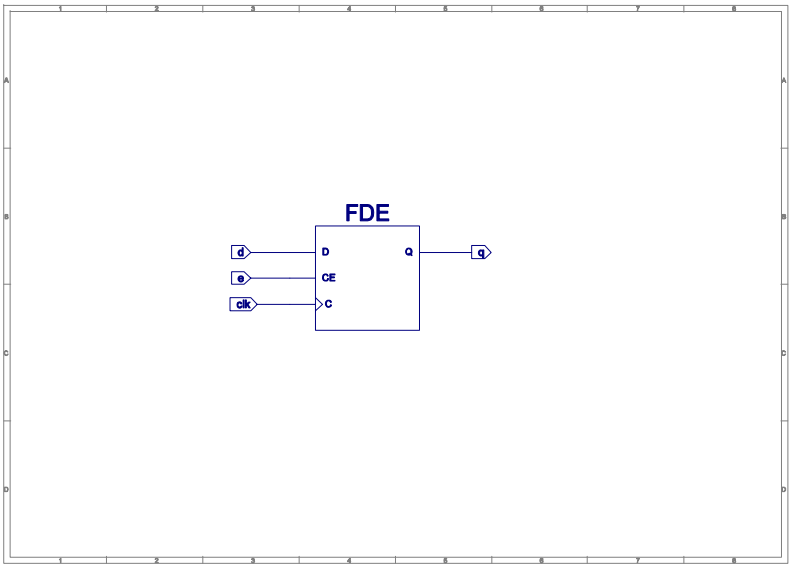
5.9 alu.sch


GND
c[15:0]
a[15:0]
b[15:0]
s
f
MUX2
S0
D1
D0
O
NOR4
OR5
OR5
OR5
c[15]
c[14]
c[13]
c[12]
c[11]
c[10]
c[9]
c[8]
c[7]
c[6]
c[5]
c[4]
c[3]
c[2]
c[1]
c[0]

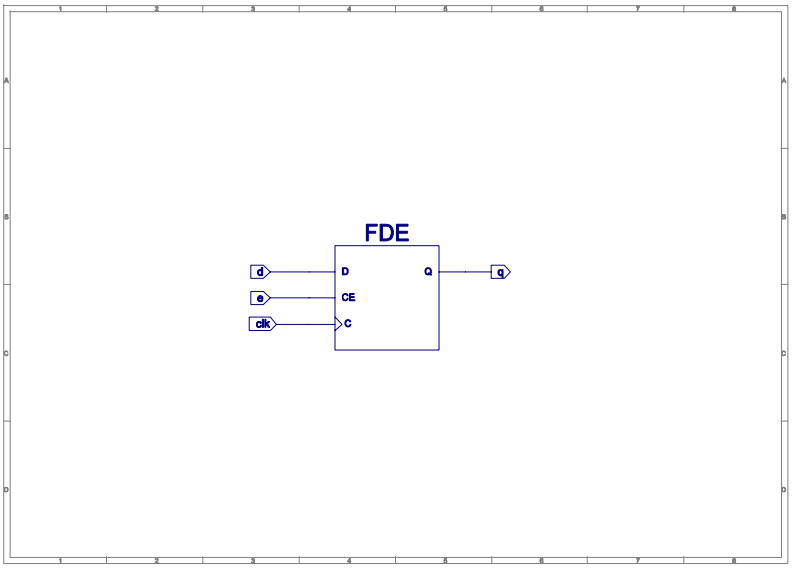
5.10 addnor.sch



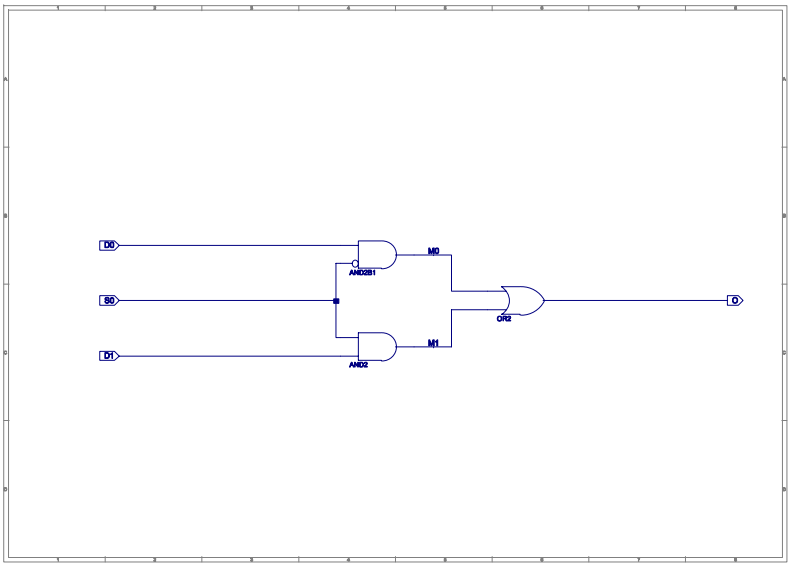
5.11 op.sch



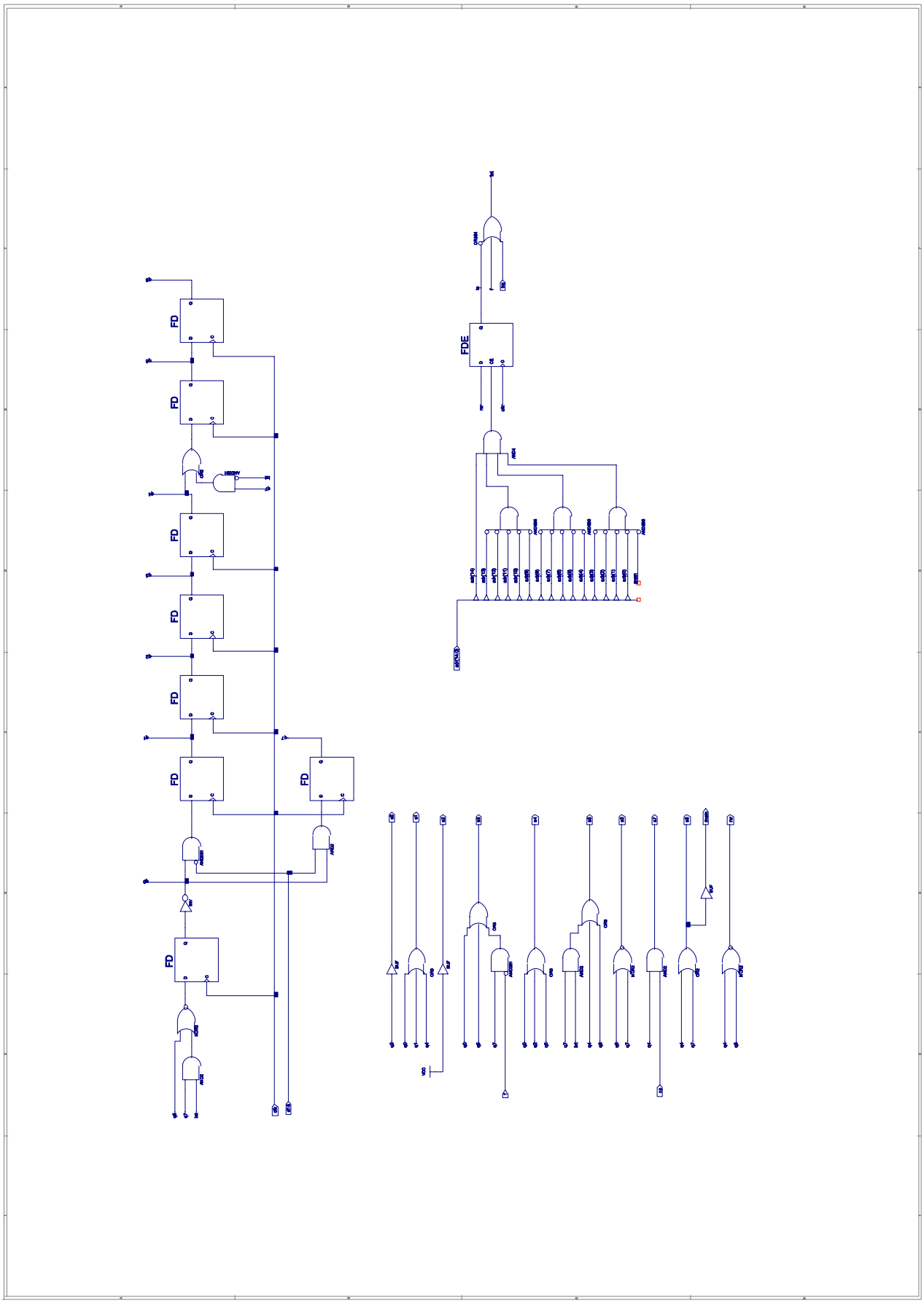
5.12 flag.sch



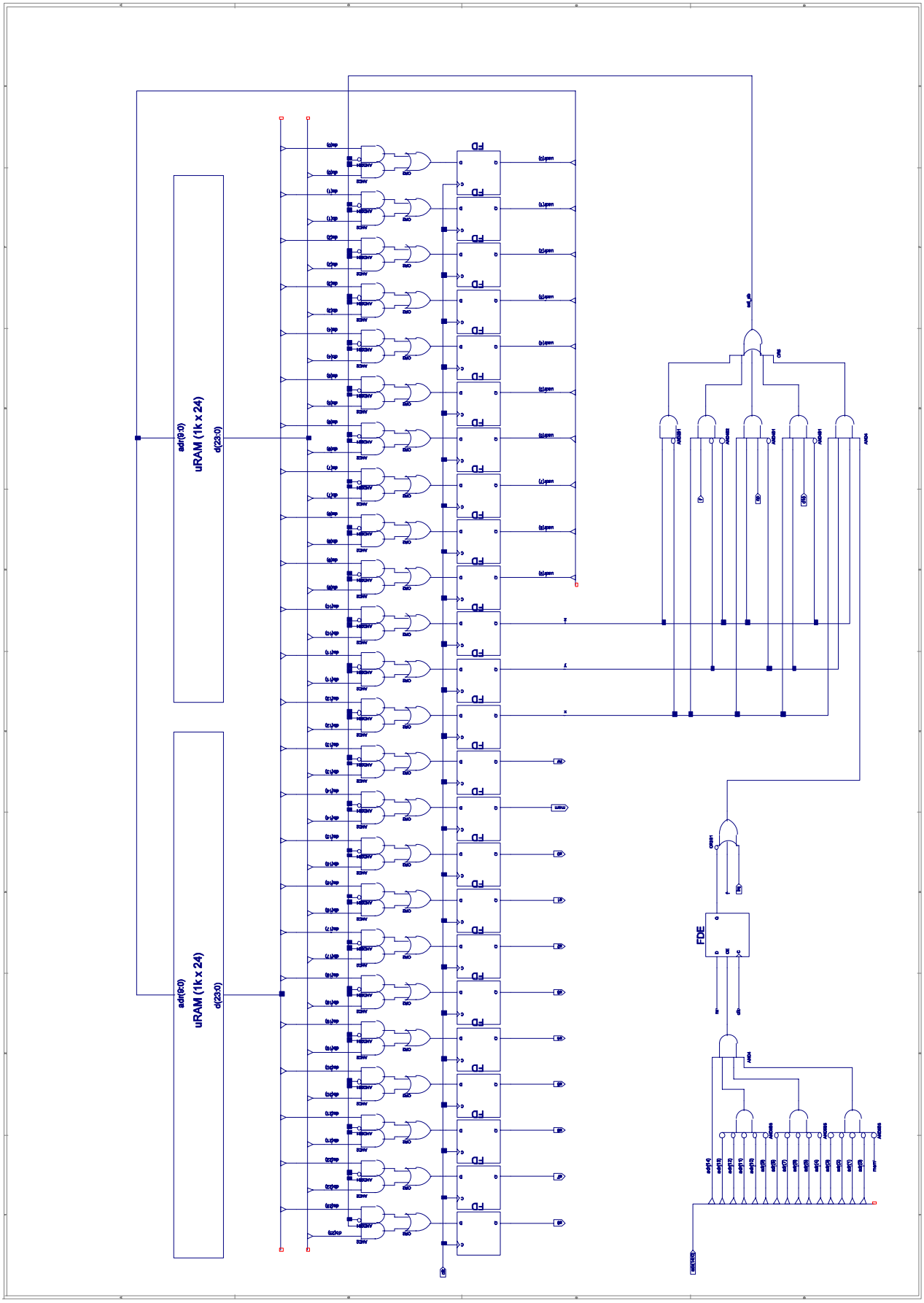
5.13 mux2.sch



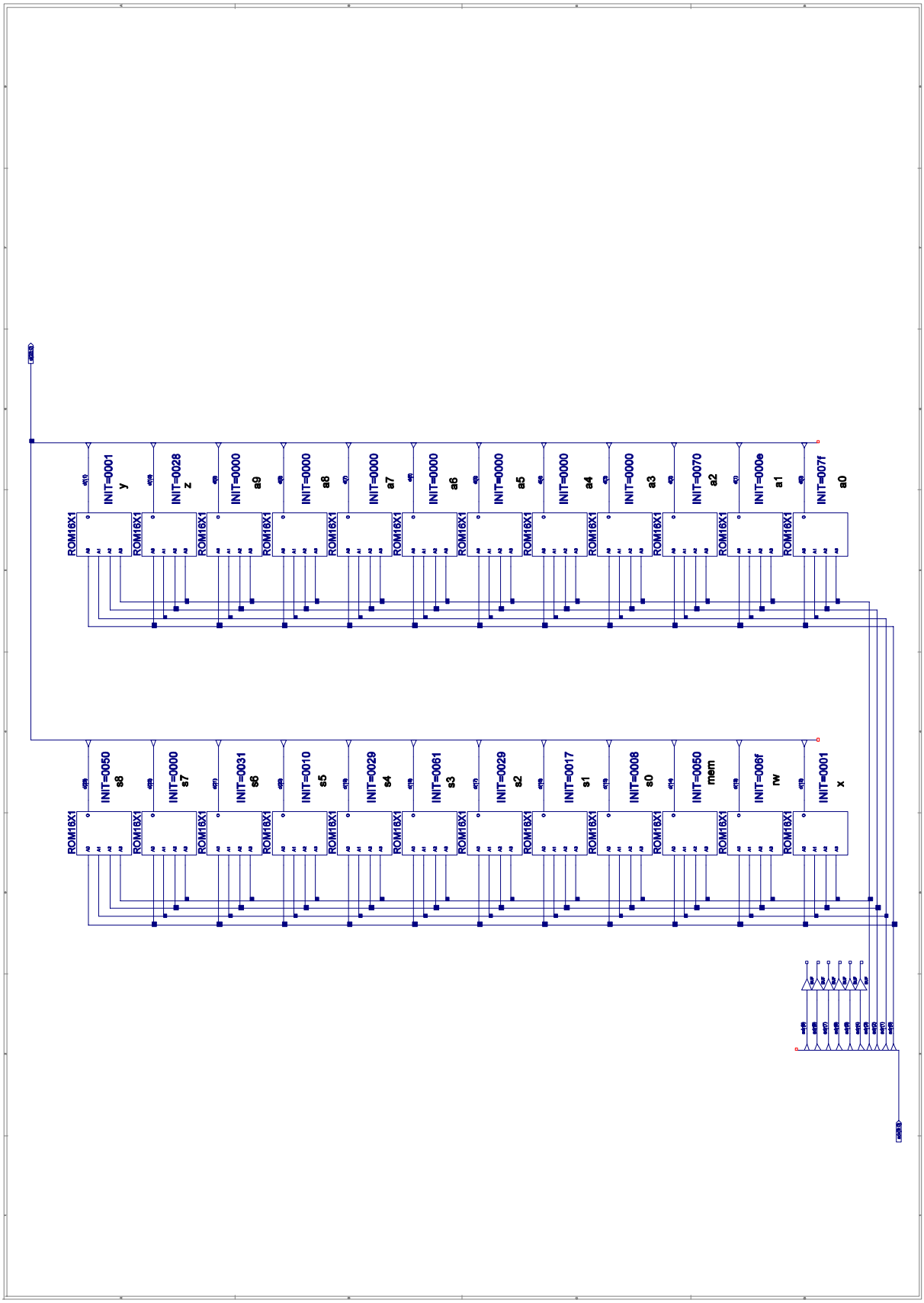
5.14 steu.sch (fest verdrahtet)



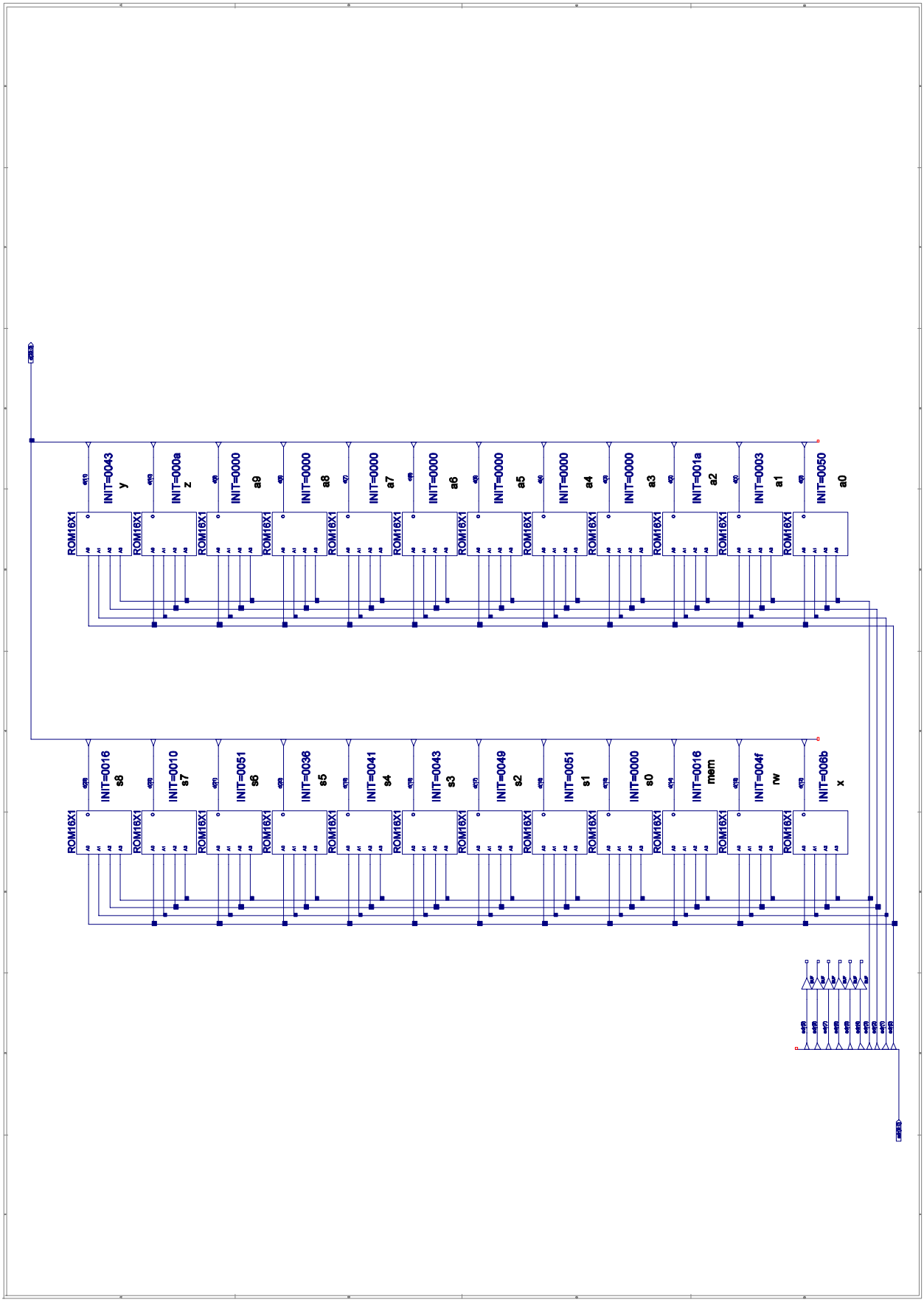
5.15 steu.sch (mikroprogrammiert)



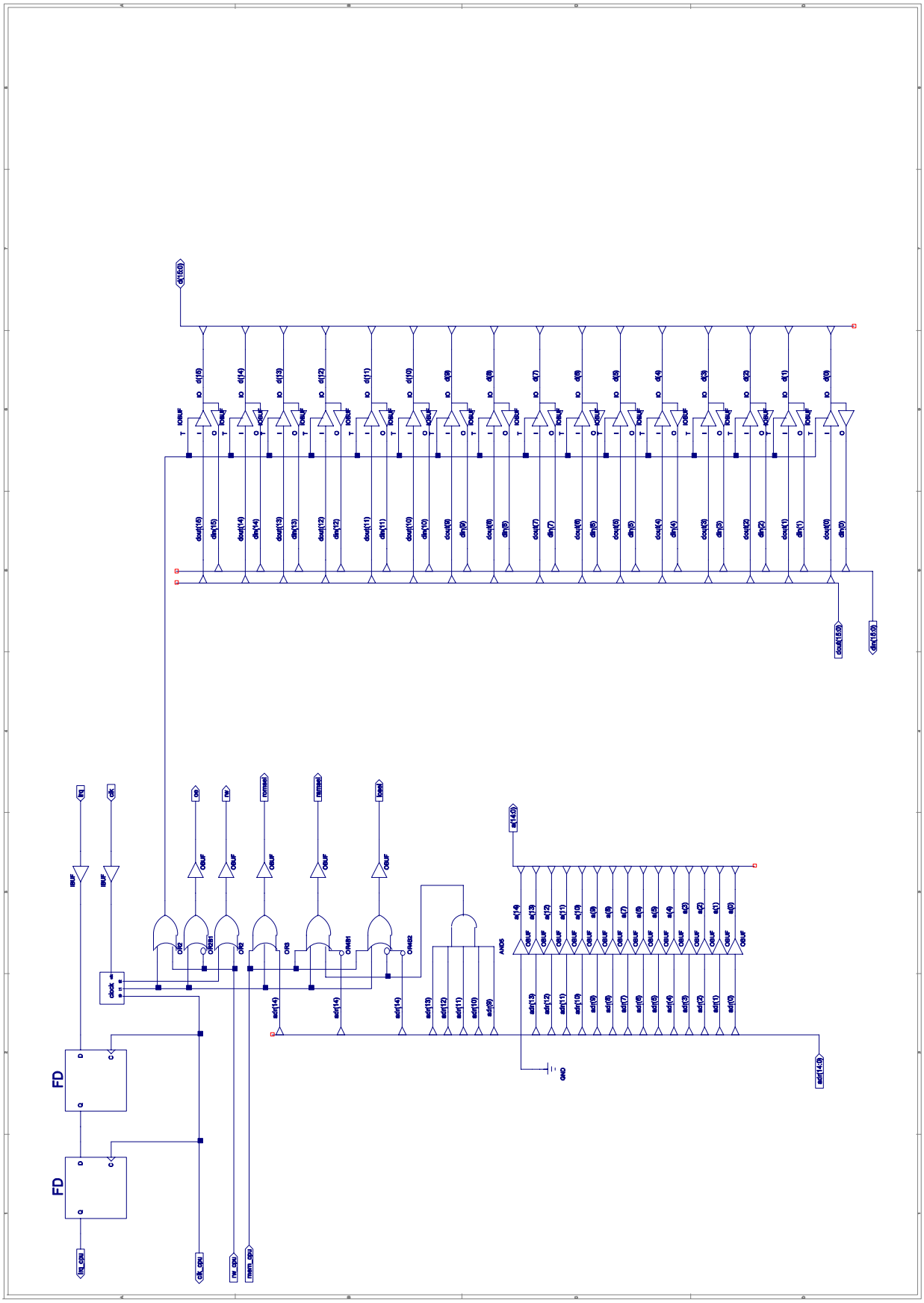
5.16 urama.sch



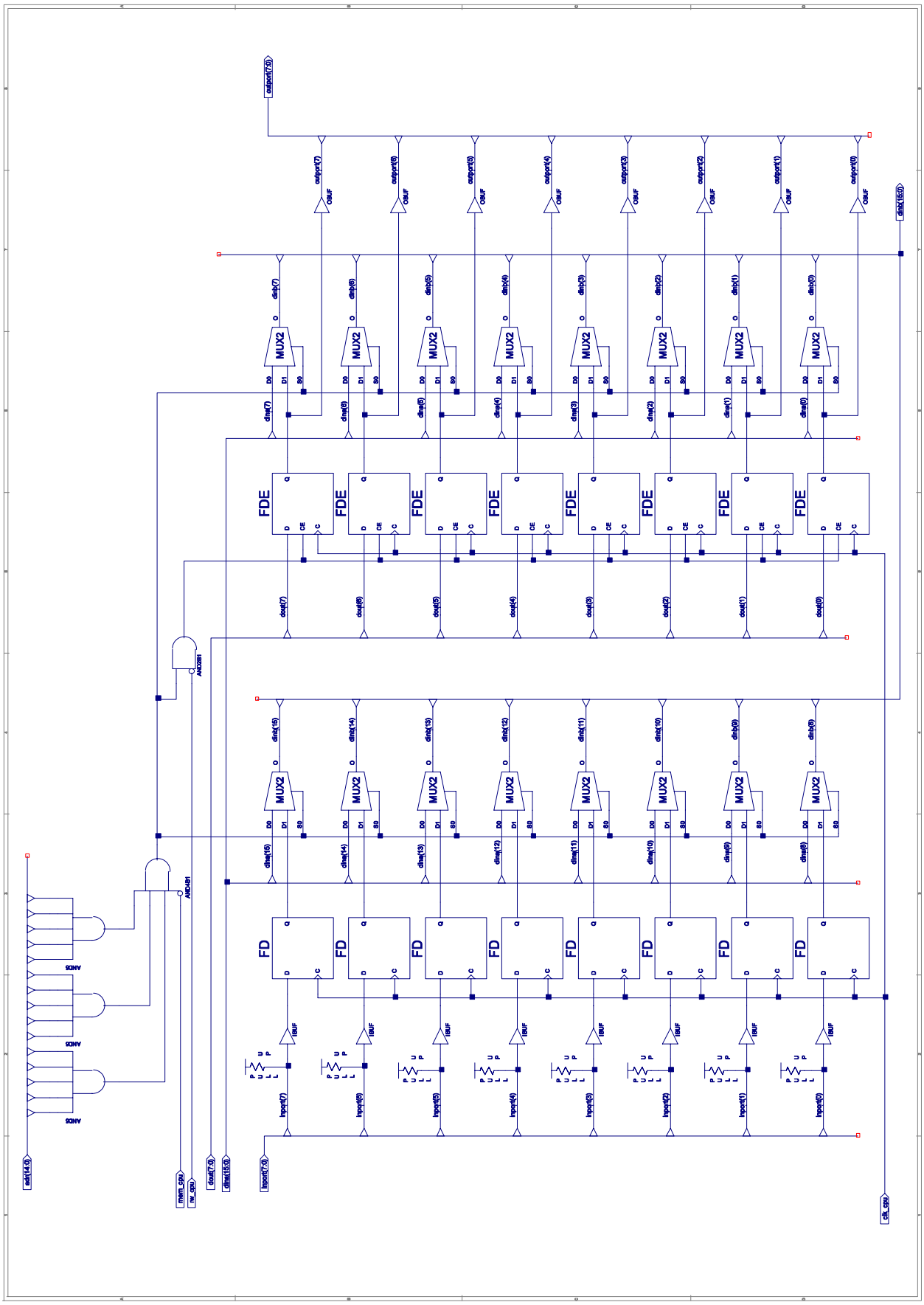
5.17 uramb.sch



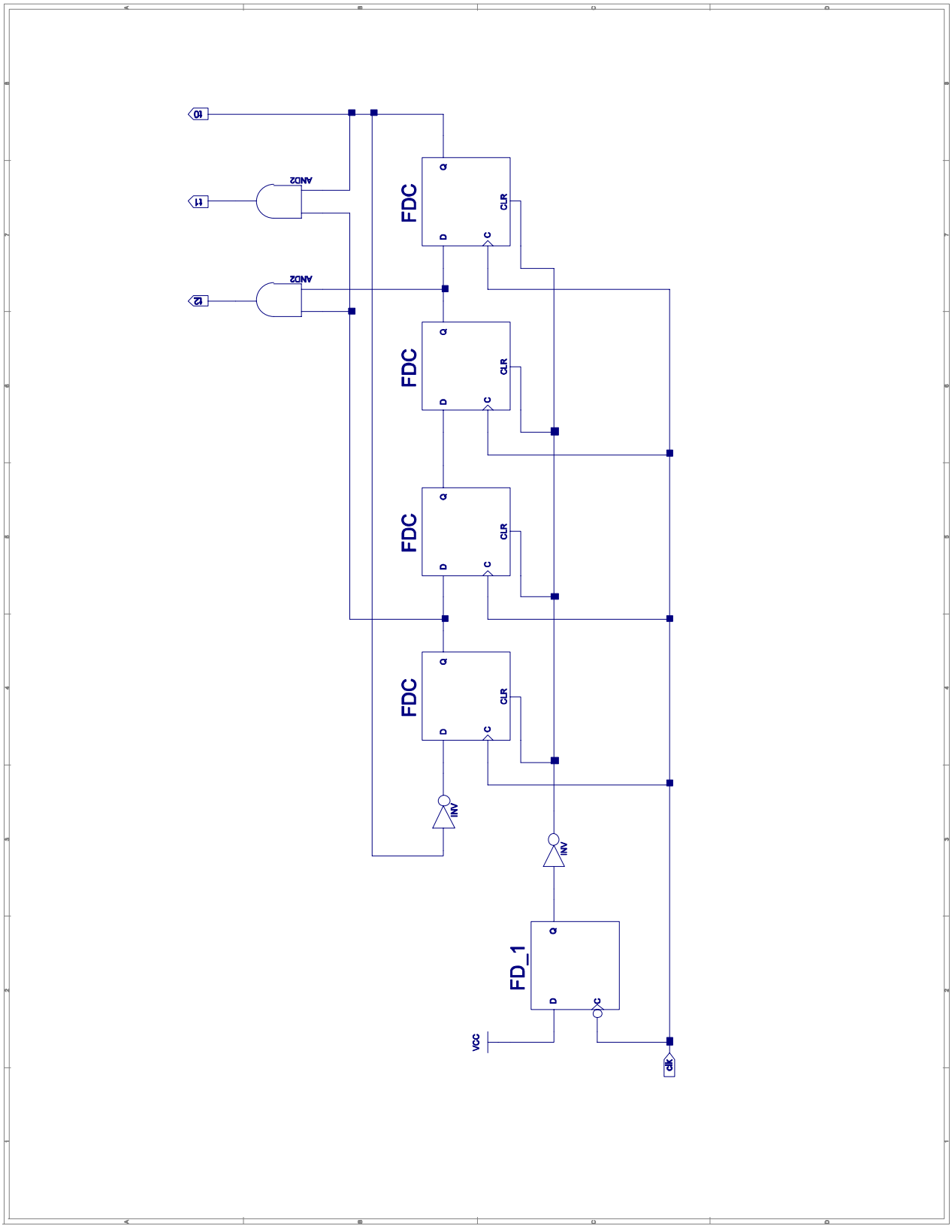
5.18 inter.sch



5.19 io.sch


outport(7:0)
outport(7)
outport(6)
outport(5)
outport(4)
outport(3)
outport(2)
outport(1)
outport(0)
OBUF
din(15:0)
din(7)
din(6)
din(5)
din(4)
din(3)
din(2)
din(1)
din(0)
MUX2
D0
D1
S0
O
dina(7)
dina(6)
dina(5)
dina(4)
dina(3)
dina(2)
dina(1)
dina(0)
FDE
Q
D
CE
C
dout(7)
dout(6)
dout(5)
dout(4)
dout(3)
dout(2)
dout(1)
dout(0)
AND2B1
din(16)
din(14)
din(13)
din(12)
din(11)
din(10)
din(9)
din(8)
FD
IBUF
PULL
UP
inport(7)
inport(6)
inport(5)
inport(4)
inport(3)
inport(2)
inport(1)
inport(0)
AND3
adr(14:0)
mem_cpu
rw_cpu
dout(7:0)
dina(15:0)
inport(7:0)
clk_cpu

5.20 clock.sch



5.21 ram.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--------------------------------------------------------------------------------
--   ____  ____
--  /   /\/   /
-- /___/  \  /    Vendor: Xilinx
-- \   \   \/     Version : 9.1.03i
--  \   \         Application :
--  /   /         Filename : xil_2628_16
-- /___/   /\     Timestamp : 05/03/2007 13:36:33
-- \   \  /  \
--  \___\/\___\
--
--Command:
--Design Name:
--

use std.textio.all;
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity ram is
   port ( a  : in    std_logic_vector (14 downto 0);
          cs : in    std_logic;
          oe : in    std_logic;
          rw : in    std_logic;
          d  : inout std_logic_vector (15 downto 0));
end ram;

architecture BEHAVIORAL of ram is
begin

mem:process
   type memory_array is array (natural range 0 to 32767) of std_logic_vector(15 downto 0);
   variable mem: memory_array;
   variable adr: natural;
   variable dat: natural;
   file file_dat: text is in "ram_data.txt";
   variable s: line;

   begin
      for adr in 0 to 32767 loop
         mem(adr) := "0000000000000000";
      end loop;

               while not endfile(file_dat) loop
                  readline(file_dat,s);
               read(s,adr);
               read(s,dat);
         mem(adr):= conv_std_logic_vector(dat,16);
      end loop;

      loop
        if cs = '0' then
           adr := conv_integer( a );
           if rw = '0' then
              mem(adr) := d(15 downto 0);
              d <= "ZZZZZZZZZZZZZZZZ";
           else
              if oe = '0' then
                 d <= mem(adr);
              else
                 d <= "ZZZZZZZZZZZZZZZZ";
              end if;
           end if;
        else
           d <= "ZZZZZZZZZZZZZZZZ";
                end if;

        wait on cs, rw, oe, a, d;
      end loop;
   end process;
end BEHAVIORAL;

-- synopsys translate_off
configuration CFG_ram of  ram is
   for BEHAVIORAL
   end for;
end CFG_ram;
-- synopsys translate_on
```

5.21 rom.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--------------------------------------------------------------------------------
--   ____  ____
--  /   /\/   /
-- /___/  \  /    Vendor: Xilinx
-- \   \   \/     Version : 9.1.03i
--  \   \         Application :
--  /   /         Filename : xil_1752_16
-- /___/   /\     Timestamp : 05/04/2007 11:28:15
-- \   \  /  \
--  \___\/\___\
--
--Command:
--Design Name:
--

use std.textio.all;
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity rom is
   port ( a  : in    std_logic_vector (14 downto 0);
          cs : in    std_logic;
          oe : in    std_logic;
          d  : inout std_logic_vector (15 downto 0));
end rom;

architecture BEHAVIORAL of rom is
begin

d <= "ZZZZZZZZZZZZZZZZ";

mem:process
   type memory_array is array (natural range 0 to 32767) of std_logic_vector(15 downto 0);
   variable mem: memory_array;
   variable adr: natural;
   variable dat: natural;
   file file_dat: text is in "rom_data.txt";
   variable s: line;

   begin
      for adr in 0 to 32767 loop
         mem(adr) := "0000000000000000";
      end loop;

               while not endfile(file_dat) loop
                  readline(file_dat,s);
               read(s,adr);
               read(s,dat);
         --write(s,adr); write(s," "); write(s,dat); writeline(output,s);
         mem(adr):= conv_std_logic_vector(dat,16);
      end loop;

      loop
         if cs = '0' then
            adr := conv_integer( a );
            if oe = '0' then
               d <= mem(adr);
             else
               d <= "ZZZZZZZZZZZZZZZZ";
            end if;
         else
           d <= "ZZZZZZZZZZZZZZZZ";
         end if;

         wait on cs, oe, a, d;
      end loop;
   end process;
end BEHAVIORAL;

-- synopsys translate_off
configuration CFG_rom of  rom is
   for BEHAVIORAL
   end for;
end CFG_rom;
-- synopsys translate_on
```

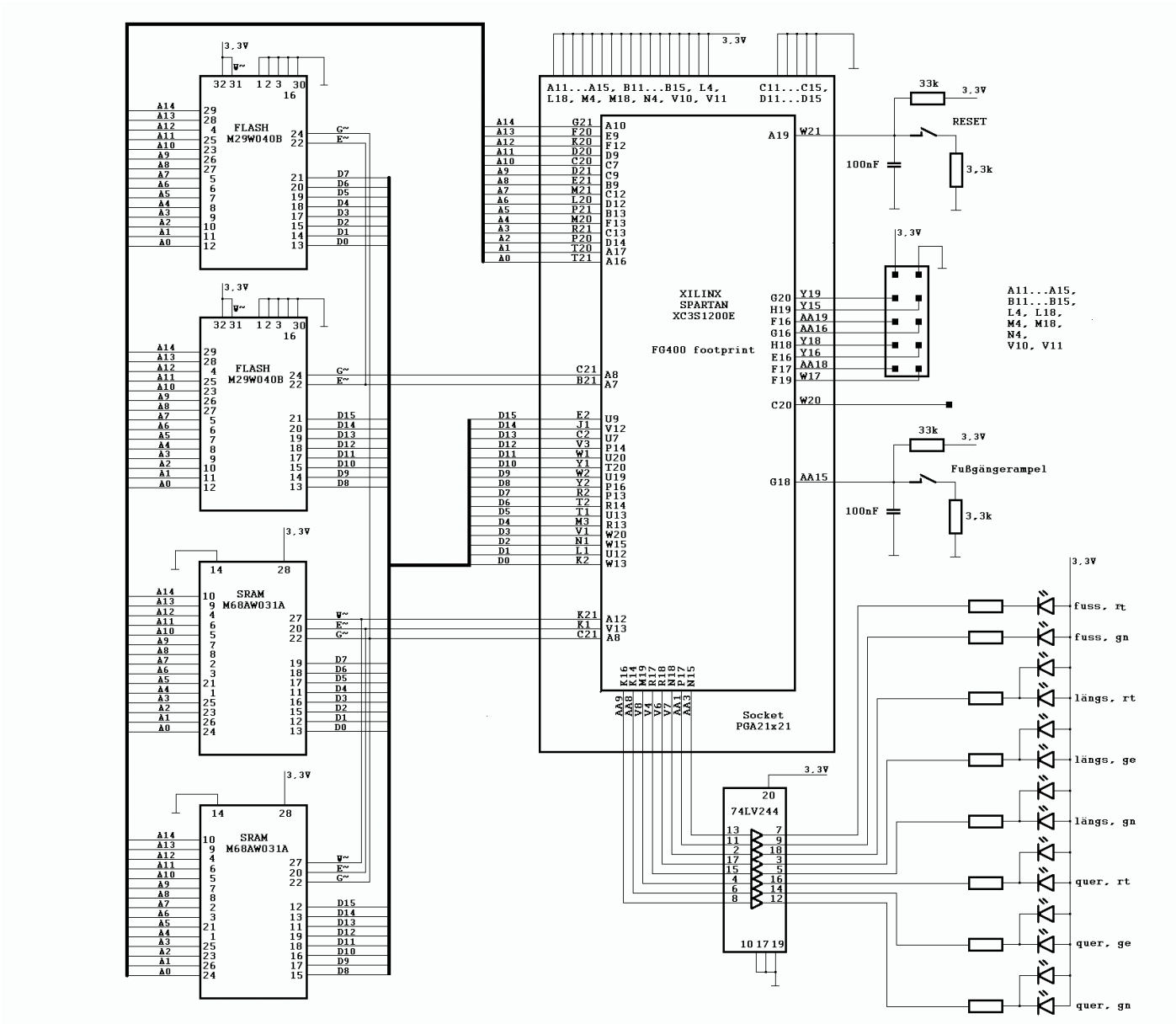
5.21 mprozsim.vhd

```
--------------------------------------------------------------------------------
-- Copyright (c) 1995-2003 Xilinx, Inc.
-- All Right Reserved.
--------------------------------------------------------------------------------
--   ____  ____
--  /   /\/   /
-- /___/  \  /    Vendor: Xilinx
-- \   \   \/     Version : 9.1.03i
--  \   \         Application : ISE
--  /   /         Filename : mprozsim.vhw
-- /___/   /\     Timestamp : Wed May 16 13:13:34 2007
-- \   \  /  \
--  \___\/\___\
--
--Command:
--Design Name: mprozsim
--Device: Xilinx
--

library UNISIM;
use UNISIM.Vcomponents.ALL;
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE STD.TEXTIO.ALL;

ENTITY mprozsim IS
END mprozsim;

ARCHITECTURE testbench_arch OF mprozsim IS
    COMPONENT mproz
        PORT (
            clk : In std_logic;
            reset : In std_logic;
            inport : In std_logic_vector (7 DownTo 0);
            irq : In std_logic;
            a : Out std_logic_vector (14 DownTo 0);
            iosel : Out std_logic;
            oe : Out std_logic;
            outport : Out std_logic_vector (7 DownTo 0);
            ramsel : Out std_logic;
            romsel : Out std_logic;
            rw : Out std_logic;
            d : InOut std_logic_vector (15 DownTo 0)
        );
    END COMPONENT;

    SIGNAL clk : std_logic := '0';
    SIGNAL reset : std_logic := '1';
    SIGNAL inport : std_logic_vector (7 DownTo 0) := "00000000";
    SIGNAL irq : std_logic := '1';
    SIGNAL a : std_logic_vector (14 DownTo 0) := "000000000000000";
    SIGNAL iosel : std_logic := '0';
    SIGNAL oe : std_logic := '0';
    SIGNAL outport : std_logic_vector (7 DownTo 0) := "00000000";
    SIGNAL ramsel : std_logic := '0';
    SIGNAL romsel : std_logic := '0';
    SIGNAL rw : std_logic := '0';
    SIGNAL d : std_logic_vector (15 DownTo 0) := "ZZZZZZZZZZZZZZZZ";

    constant PERIOD : time := 30 ns;
    constant DUTY_CYCLE : real := 0.5;
    constant OFFSET : time := 0 ns;

    BEGIN
        UUT : mproz
        PORT MAP (
            clk => clk,
            reset => reset,
            inport => inport,
            irq => irq,
            a => a,
            iosel => iosel,
            oe => oe,
            outport => outport,
            ramsel => ramsel,
            romsel => romsel,
            rw => rw,
            d => d
        );

        PROCESS    -- clock process for clk
        BEGIN
            WAIT for OFFSET;
            CLOCK_LOOP : LOOP
                clk <= '0';
                WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
                clk <= '1';
```

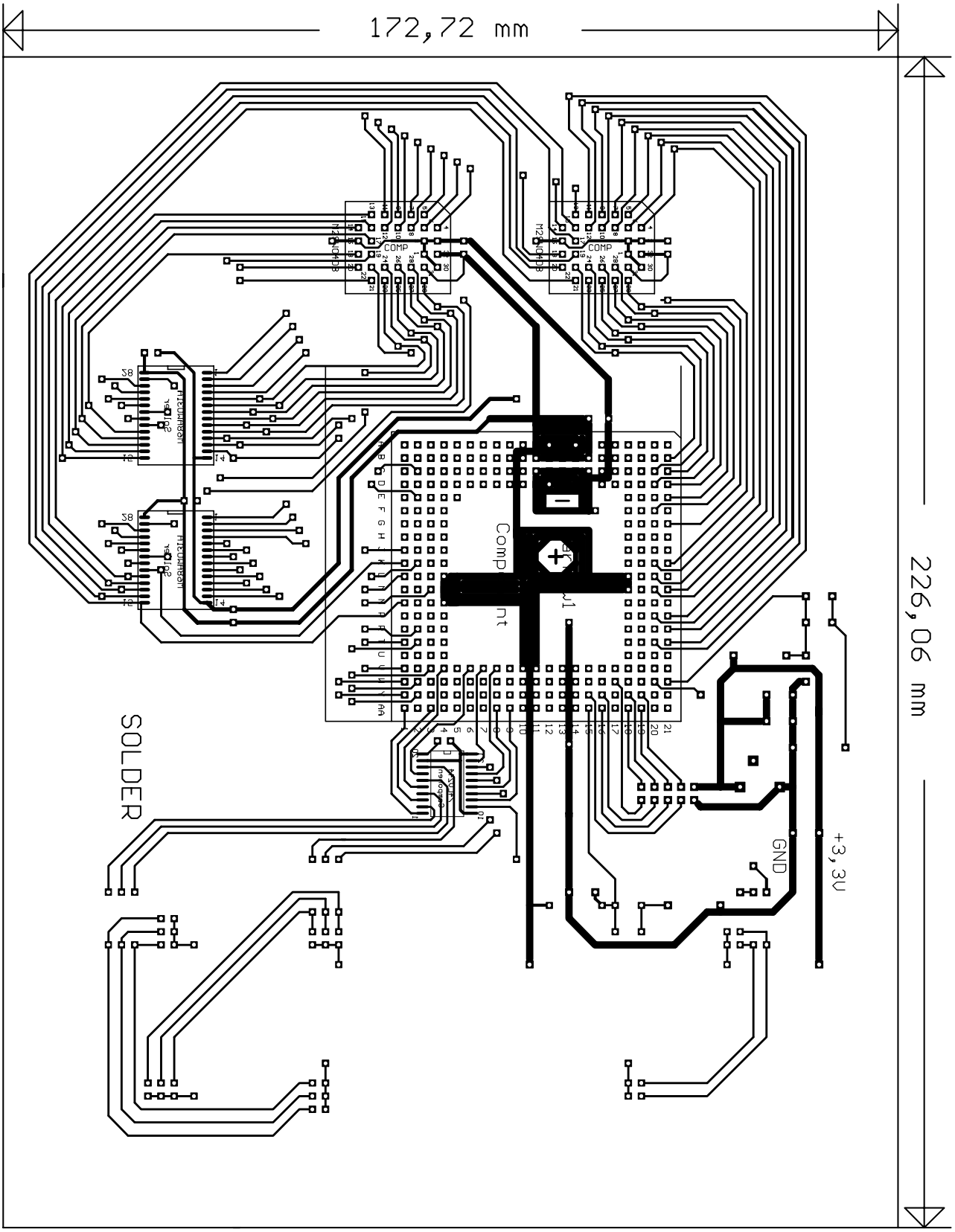
```
            WAIT FOR (PERIOD * DUTY_CYCLE);
        END LOOP CLOCK_LOOP;
    END PROCESS;

    PROCESS
        BEGIN
            WAIT FOR 10 ns;
            inport <= "00110101";
            -- -------------------------------------
            WAIT FOR 50000 ns;
            irq <= '0';
            -- -------------------------------------
            WAIT FOR 5000 ns;
            irq <= '1';
            -- -------------------------------------
            WAIT FOR 5000000 ns;

        END PROCESS;

END testbench_arch;
```

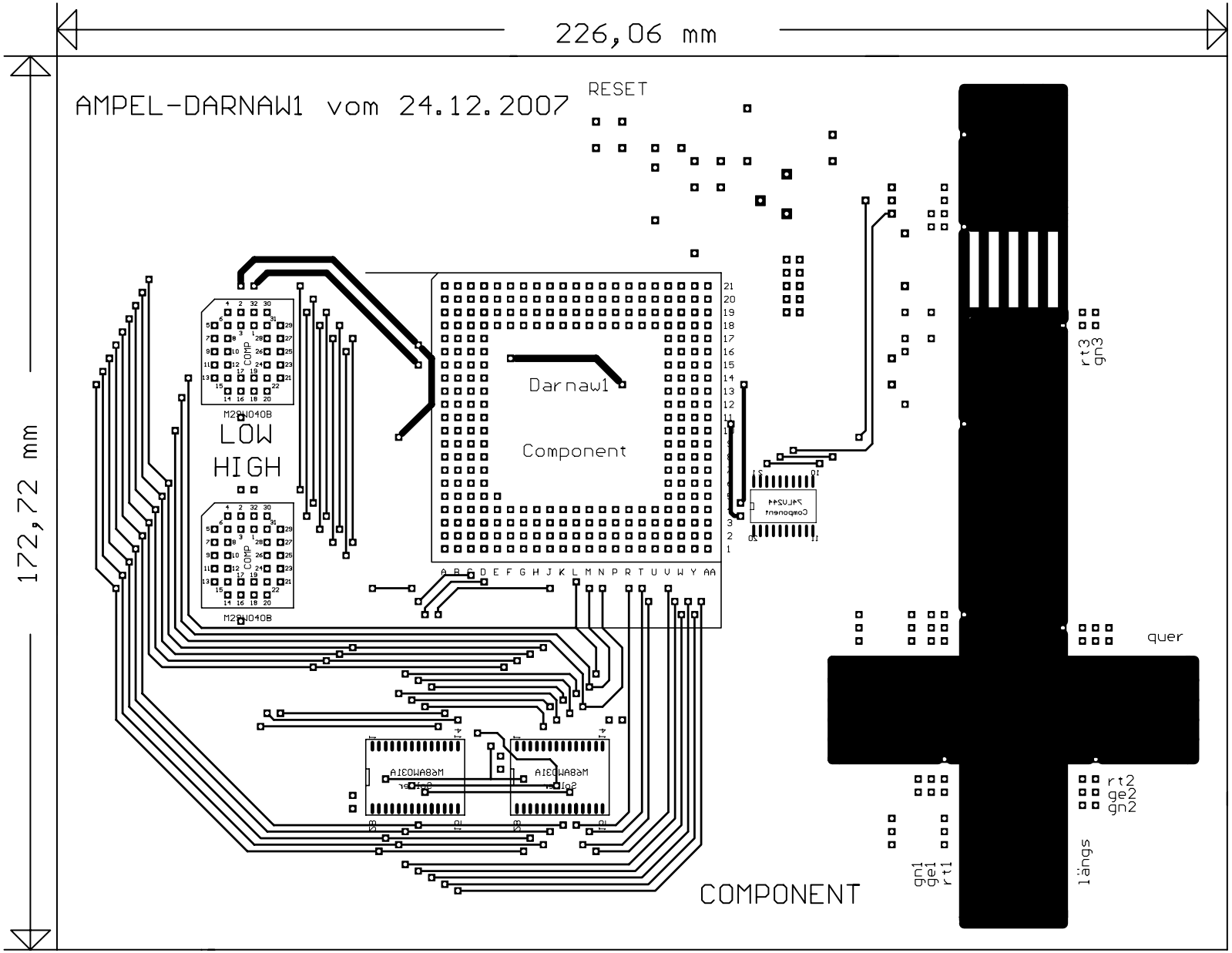
Anhang I (nicht für Studenten)

Dokumentation Demoboard



Signal	FPGA	BOARD		Signal	FPGA	BOARD
				D15	U9	E2
A14	A10	G21		D14	V12	J1
A13	E9	F20		D13	U7	C2
A12	F12	K20		D12	P14	V3
A11	D9	D20		D11	U20	W1
A10	C7	C20		D10	T20	Y1
A9	C9	D21		D9	U19	W2
A8	B9	E21		D8	P16	Y2
A7	C12	M21		D7	P13	R2
A6	D12	L20		D6	R14	T2
A5	B13	P21		D5	U13	T1
A4	F13	M20		D4	R13	M3
A3	C13	R21		D3	W20	V1
A2	D14	P20		D2	W15	N1
A1	A17	T20		D1	U12	L1
A0	A16	T21		D0	W13	K2
IN7	G16	AA16		OUT7	P17	AA1
IN6	F19	W17		OUT6	N15	AA2
IN5	F17	AA18		OUT5	R17	V4
IN4	F16	AA19		OUT4	R18	V6
IN3	H19	Y15		OUT3	N18	V7
IN2	E16	Y16		OUT2	M19	V8
IN1	H18	Y18		OUT1	K14	AA8
IN0	G20	Y19		OUT0	K16	AA9
romsel	A7	B21		LED7	Y7	
ramsel	V13	K1		LED6	V8	
iosel	C20	W20		LED5	W8	
oe	A8	C21		LED4	Y8	
we	A12	K21		LED3	V9	
irq	G18	AA15		LED2	W9	
clk	V11			LED1	Y9	
reset	A19	W21		LED0	readonly	


172,72 mm
226,06 mm
SOLDER
GND
+3,3V


226,06 mm
172,72 mm
AMPEL-DARNAW1 vom 24.12.2007
RESET
Darnaw1
Component
LOW
HIGH
M28W040B
M28W040B
COMP
rt3
gn3
quer
rt2
ge2
gn2
gn1
ge1
rt1
längs
COMPONENT

```
;******
;******    ampel_5.ass vom 10.M„rz 1999    ******
;******    mit Fuág„ngerampel (Interrupt) ******
;******    angepasst an MDELA 25.2.08


        br      auto_amp
        dcw     $4000       ; Adresse zum retten des PC bei einem Interrupt
;
;****** Interruptroutine ab Adresse 0002 !!!    ******
;
;        br      irq         ; Adresse der Interruptroutine
irq:
        move    io,r0   ;
        nor     #ff3f,r0    ; Abfrage, ob FUSS AUS
        bne     set_fuss    ; FUSS AUS ---> FUSS auf rot
        move    io,r0
        add     #00c0,r0    ; FUSS auf ROT
        move    r0,io
set_fuss:
        br      $4000       ; R•cksprung aus Interrupt
        br      $4000       ; R•cksprung aus Interrupt
;
;********** Variablen **********
;
#$7fff: dcw     $7fff
#$ffff: dcw     $ffff
#$0001: dcw     $0001
#0000 : dcw     $0000
#0001 : dcw     $0001
;
;********** Die Zeitschleife "zeit" f•r 22 Sekunden hat im „uáeren
;********** Schleifenregister den Wert $96 und im inneren den Wert
;********** $ffff bei einem 32MHz-Quarz (Teiler 8=250æsec)
;
#02s  : dcw     $0001       ; Zeitwert f•r 0,2 Sekunden
#05s  : dcw     $0003       ; Zeitwert f•r 0,5 Sekunden
#1s   : dcw     $0007       ; Zeitwert f•r 1 Sekunden
#2s   : dcw     $000d       ; Zeitwert f•r 2 Sekunden
#4s   : dcw     $001b       ; Zeitwert f•r 4 Sekunden
#6s   : dcw     $0029       ; Zeitwert f•r 6 Sekunden
#8s   : dcw     $0036       ; Zeitwert f•r 8 Sekunden
#22s  : dcw     $0096       ; Zeitwert f•r 22 Sekunden
;
#0076: dcw  $0076
#00c0: dcw  $00c0
#00db: dcw  $00db
#00eb: dcw  $00eb
#00f1: dcw  $00f1
#00f5: dcw  $00f5
#00f6: dcw  $00f6
#00e3: dcw  $00e3
#005b :dcw  $005b
#ff00 :dcw  $ff00
#ff3f :dcw  $ff3f
#ffbf: dcw  $ffbf
#feff: dcw  $feff
#ff7f: dcw  $ff7f
#ffff: dcw  $ffff
;
;
;********** Programmstart und Initialisierung **********
;
auto_amp:   move    #ff00,io    ; Alle LEDs AN
            move    #1s,r3
            move    auto_ini1,zeit_ret
            br      zeit
auto_ini1:  dcw $8000+auto_ini2
auto_ini2:  move    #ffff,io   ; Alle LEDs AUS
            move    #1s,r3
            move    auto_ini3,zeit_ret
            br      zeit
auto_ini3:  dcw $8000+auto_ini3a

auto_ini3a: move    #1s,r6
            nor     #0000,r6
auto_ini4:  move    #0001,r2   ; LED's nacheinander AN
auto_ini5:  move    r2,r4
            nor     #0000,r4
            move    r4,io       ; Ausgabe
            move    #02s,r3
            move    auto_ini6,zeit_ret
            br      zeit
auto_ini6:  dcw $8000+auto_ini7
auto_ini7:  move    r2,r5
            add     r5,r2
            move    r2,r5
            nor     #feff,r5
            bne     auto_ini5
            add     #0001,r6
            bcc     auto_ini4
auto_ini8:  move    #ffff,io   ; Alle LEDs AUS
            move    #2s,r3
            move    auto_ini9,zeit_ret
```

```
            br      zeit
auto_ini9:  dcw $8000+auto_ini10

auto_ini10: add     $4000,tmp   ; Lesezugriff: Interrupt enable
;
;********** Ampelsteuerung **********
;
auto_1: ;aampel_1: rt    - 6 sec / aampel_2: gn    - 6 sec; Wert:$xxdb

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00db,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io

        move    #6s,r3
        move    a1_s,zeit_ret
        br      zeit
a1_s:   dcw $8000+auto_2

auto_2: ;aampel_1: rt    - 2 sec / aampel_2: ge    - 2 sec; Wert:$xxeb

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00eb,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io

        move    #2s,r3
        move    a2_s,zeit_ret
        br      zeit
a2_s:   dcw $8000+auto_3


auto_3: ;aampel_1: rt/ge - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf1

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00f1,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        move    #2s,r3
        move    a3_s,zeit_ret
        br      zeit
a3_s:   dcw $8000+auto_4

auto_4: ;aampel_1: gn    - 6 sec / aampel_2: rt    - 6 sec; Wert:$xxf6

        move    io,r4   ;
        nor     #ffbf,r4    ; Abfrage, ob FUSS rot
        bne     a4_1        ; FUSS ROT?
        move    #00f6,io    ; Normalbetrieb
        br      a4_2
a4_1:   move    #0076,io    ; FUSS auf gr•n setzen!
a4_2:   move    #6s,r3
        move    a4_s2,zeit_ret
        br      zeit
a4_s1:  dcw $8000+auto_5

        move    #6s,r3
        move    a4_s1,zeit_ret
        br      zeit
a4_s2:  dcw $8000+auto_5

auto_5: ;aampel_1: ge    - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf5

        move    io,r4
        nor     #ff7f,r4    ; FUSS gr•n?
        bne     a5_1
        move    io,r4       ; FUSS nicht gr•n
        nor     #ff3f,r4
        move    #00f5,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        br      a5_2
a5_1:   move    #00f5,io    ; FUSS AUS
a5_2:   move    #2s,r3
        move    a5_s,zeit_ret
        br      zeit
a5_s:   dcw $8000+auto_6

auto_6: ;aampel_1: rt    - 2 sec / aampel_2: rt/ge - 2 sec; Wert:$xxe3

        move    io,r4   ;
        nor     #ff3f,r4    ; FUSS maskieren
        move    #00e3,r5
        nor     #0000,r5
        nor     r5,r4
        move    r4,io
        move    #2s,r3
```

```
        move    a6_s,zeit_ret
        br      zeit
a6_s:   dcw $8000+auto_1

;************   Zeitschleife         *************
;************   Z„hlerwert in r3     ************

zeit:   nor #$ffff,r1
        nor #0000,r3
        add #$0001,r3
w1i:    add #$0001,r1
        bcc w1i
        add #$0001,r3
        bcc w1i
        br  zeit_ret

;************   Umschalten auf RAM   ************
;
        ram
        dcw     0       ; Speicherzelle zum retten des PC bei einem Interrupt


r0:     dcw     0
r1:     dcw     0
r2:     dcw     0
r3:     dcw     0
r4:     dcw     0
r5:     dcw     0
r6:     dcw     0
doio_ret:dcw    0
zeit_ret:dcw    0
tmp:    dcw     0
io=$7fff

;****** Die Fuág„ngerampel schaltet mit aampel_2 (Querrichtung)
;
;auto_1=01xx: ;aampel_1: rt    - 6 sec / aampel_2: gn    - 6 sec; Wert:$xxdb
;auto_2=02xx: ;aampel_1: rt    - 2 sec / aampel_2: ge    - 2 sec; Wert:$xxeb
;auto_3=03xx: ;aampel_1: rt/ge - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf1
;auto_4=04xx: ;aampel_1: gn    - 6 sec / aampel_2: rt    - 6 sec; Wert:$xxf6
;auto_5=05xx: ;aampel_1: ge    - 2 sec / aampel_2: rt    - 2 sec; Wert:$xxf5
;auto_6=06xx: ;aampel_1: rt    - 2 sec / aampel_2: rt/ge - 2 sec; Wert:$xxe3
;fuss_1=07xx: ;fussampel: rot  - Beginn in auto_3 bis auto_6
;fuss_2=08xx: ;fussampel: gr•n - 8 sec in Phase auto_1 und auto_2

end:
```