

HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse/ Jahrgang: 5BHEL	Gruppe: 6b	Übungsleiter: Prof. Dum
Übungsnummer: III/ 3	Übungstitel: Digitaler Entwurf-CPU	
Datum der Übung: 08.03.2016	Teilnehmer: Dräxler, Gorke, Marceta, Mayerhofer, Mischling	
Datum der Abgabe: 05.04.2016	Schritfführer: Marceta	Unterschrift:

	Beurteilung
Aufgabenstellung	
Dokumentation	
Messschaltungen	
Messtabellen	
Berechnungen	
Programmlistings	
Auswertung	
Diagramme	
Berechnungen	
Simulationen	
Schlußfolgerungen	
Kommentare	
Inventarliste	
Messprotokoll	
Form	
Summe	

Allgemeiner Teil

Titel der Übung: Digitaler Entwurf-CPU
Übungsnummer: III/3
Übungsplatz: T8
Datum der Übung: 08.03.2016
Klasse: 5BHEL
Schriftführer: Marceta
Übungsteilnehmer: Dräxler
Gorke
Marceta
Mayerhofer
Mischling

Inhaltsverzeichnis

Allgemeiner Teil	1
1. Aufgabenstellung	2
2. Basiswissen MARS	2
2.1. Programmcode	2
2.2. Kompilierung des Programmes	2
2.3. Mars-Tools	4
2.4. Verwendete Tools	4
2.4.1. Instruction Counter	5
2.4.2. Instruction Statistics	5
2.4.3. Data Cache Simulator	6
2.4.4. Memory Reference Visualisation	6
3. Fibonacci	6
3.1. Instruction Counter	7
3.2. Instruction Statistic	7
3.3. Data Cache Simulator	8
3.4. Memory Reference Visualisation	8
4. Programm	9
4.1. Programm für VHDL anpassen	9
4.2. Implementierung in VHDL-Projekt	11
4.3. Topsheet der CPU	11
4.4. fourbitlow-Entity	11
4.5. UCF-File anpassen	12
4.5. Testbench der Gesamtfunktion	12
5. Hardwaretest	13
6. Erkenntnis	14
7. Messprotokoll	15
8. Inventarliste	16

1. Aufgabenstellung

Die Aufgabenstellung war es ein MIPS-Assemblerprogramm mittels dem Programm MARS DIE zu entwerfen. Dieses geschriebene Programm soll in eine VHDL-ROM exportiert werden und in der CPU ausgeführt werden, welche in VHDL realisiert worden ist.

2. Basiswissen MARS

Zur Vorbereitung auf die Laborübung wurde das Beispielprogramm „htlmips1.asm“ ausgeführt und analysiert.

2.1. Programmcode

```
# htlmips1.asm

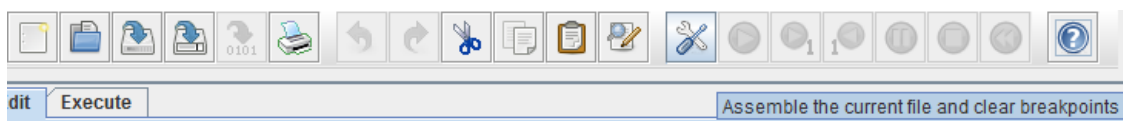
.eqv iport_1    0x1010
.eqv oport_1    0x1010

.text

M1: lw $s2, iport_1      # read from input port 1
    addiu $s3,$s2,-1     # modify
    sw $s3, oport_1      # output result
    j M1                 # loop
```

Das Beispielprogramm liest von der Adresse iport_1[0x1010] ein speichert den Wert im Register \$s2. Desweiteren wird als nächstes die Manipulation der Daten durchgeführt. Es wird in das Register der Wert von s2 um 1 dekrementiert gespeichert. Der Inhalt des Registers s3 wird nun wieder auf derselben Adresse ausgegeben. Das Programm wurde in einer Endlosschleife realisiert.

2.2. Kompilierung des Programmes



Mit diesem Button wird das Programm in den Maschinencode umgewandelt, somit kann es ausgeführt werden. Der Editor wird minimiert und man kommt in das „Execute-Fenster“. In diesem Fenster kann das Programm gestartet werden.

Text Segment					
Bkpt	Address	Code	Basic	Source	
☐	0x00000000	0x8c121010	lw \$18,0x00001010(\$0)	9: M1: lw \$s2, 0x1010	# read from input port 1
☐	0x00000004	0x2653ffff	addiu \$19,\$18,0xffff...	10: addiu \$s3,\$s2,-1	# modify
☐	0x00000008	0xac131010	sw \$19,0x00001010(\$0)	11: sw \$s3, 0x1010	# output result
☐	0x0000000c	0x08000000	j 0x00000000	12: j M1	# loop

Der gelb markierte Bereich ist der Befehl, welcher als nächstes ausgeführt wird. Hier sieht man die Breakpoints, Adresse, Assemblercode und den Befehl.

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0x00000000		
\$t1	9	0x00000000		
\$t2	10	0x00000000		
\$t3	11	0x00000000		
\$t4	12	0x00000000		
\$t5	13	0x00000000		
\$t6	14	0x00000000		
\$t7	15	0x00000000		
\$s0	16	0x00000000		
\$s1	17	0x00000000		
\$s2	18	0x00000000		
\$s3	19	0x00000000		
\$s4	20	0x00000000		
\$s5	21	0x00000000		
\$s6	22	0x00000000		
\$s7	23	0x00000000		
\$t8	24	0x00000000		
\$t9	25	0x00000000		
\$k0	26	0x00000000		
\$k1	27	0x00000000		
\$gp	28	0x00001800		
\$sp	29	0x00003ffc		
\$fp	30	0x00000000		
\$ra	31	0x00000000		
pc		0x00000000		
hi		0x00000000		
lo		0x00000000		

In diesem Fenster ist zu sehen, welche Register es gibt und deren Eigenschaften wie Name, Nummer und der Wert.

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Hier ist der Speicher zu sehen.

Das Programm kann Step per Step abgearbeitet werden oder automatisch durchlaufen werden.



Automatischer Durchlauf, kann mittels Pause- bzw. Stopp-Taste abgebrochen werden.



Schritt per Schritt kann das Programm durchgesprungen werden



Mit dieser Taste kann die Ausführung des Programmes pausiert werden



Hiermit kann das Programm angehalten werden



Mit diesem Button kann ein Befehl zurückgesprungen werden



Setzt das Programm wieder zum Anfangszeitpunkt zurück

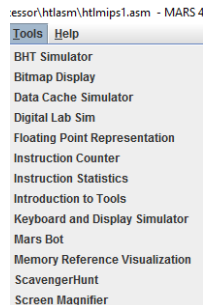


Um das Programm bis zu einem bestimmten Befehl laufen zu lassen, kann ein Breakpoint gesetzt werden, um einen Breakpoint zu setzen muss man einfach in der Zeile, wo man pausieren möchte ein „Hacker!“ setzen.

2.3.Mars-Tools

Der Simulator stellt einige Tools zur Verfügung. Die Tools können über den Menüpunkt aufgerufen werden, wie man unten auf dem Screenshot sehen kann.

Anmerkung: Es ist nur eine Auflistung von den Tools welche Mars zur Verfügung stellt, es wurden nicht alle Tools verwendet. Auf die verwendeten Tools wird noch näher eingegangen.



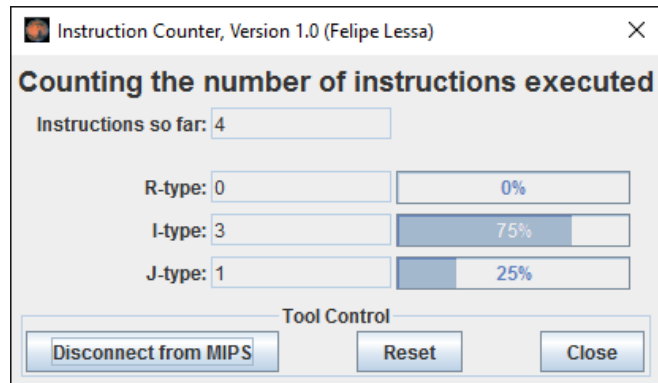
2.4. Verwendete Tools

- Instruction Counter
- Instruction Statistics
- Data Cache Simulator
- Memory Reference Visualisation

2.4.1. Instruction Counter

Instruction Counter zeigt die Befehlsklassen an, diese sind unterteilt in Register-, Immediate- und Jump-Befehle.

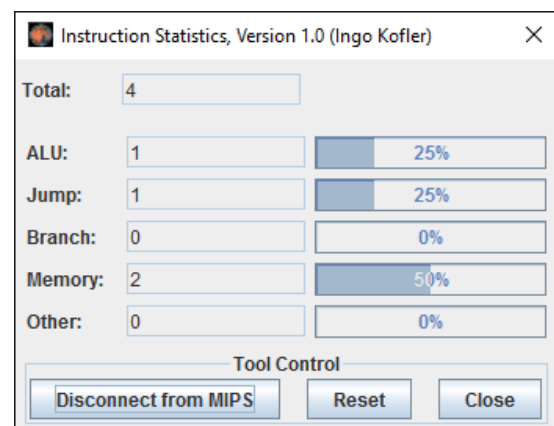
Bevor ein Tool funktionstüchtig ist muss man es mit MIPS verbinden. Danach wird einmal der Befehl ausgeführt und man erhält die Zählwerte der Befehlsklassen. Auf diesem Screenshot ist zu sehen die Gesamtanzahl der Instructions und die einzelnen Befehlsklassen, wie oft diese ausgeführt worden sind.



2.4.2. Instruction Statistics

Instruction Statistics unterteilt die Befehle nach ALU-, Jump-, Branch-, und Speicher-Zugriff.

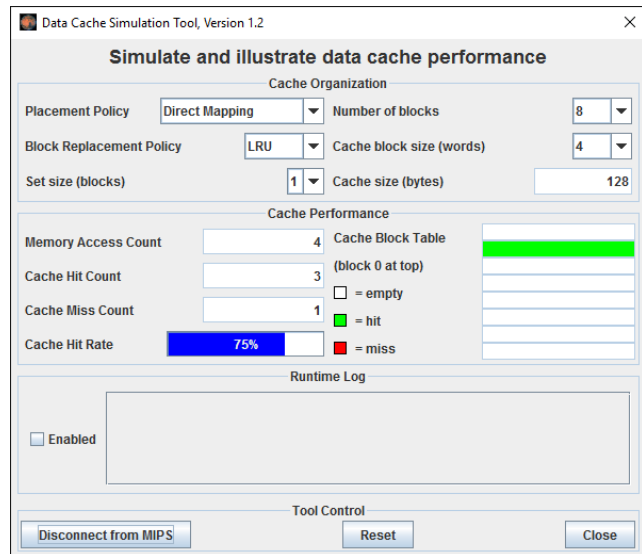
Bevor ein Tool funktionstüchtig ist muss man es mit MIPS verbinden. Danach wird einmal der Befehl ausgeführt und man erhält die Unterteilung. Hier ist wieder die Gesamtanzahl zu sehen und die einzelnen Zugriffe.



2.4.3. Data Cache Simulator

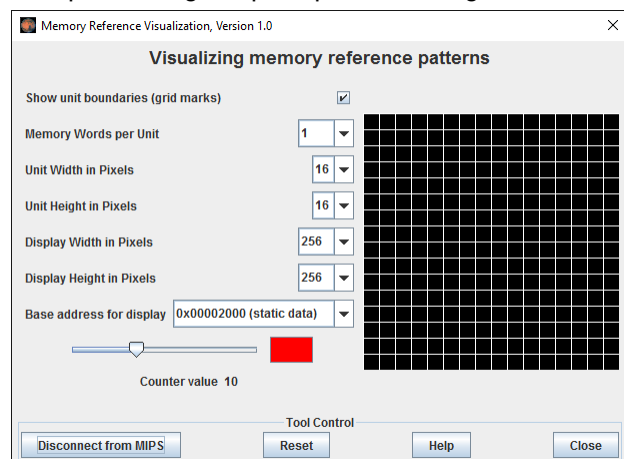
Data Cache Simulator zeigt die Performance des Caches an.

Hier kann man unterscheiden zwischen Direkt, Vollasoziativ und Set Assoziativen Cache. Auf diesem Screenshot wurde ein Direkter Cache verwendet. Man sieht wie oft auf den Speicher zugegriffen worden ist, wie viele Cache-Hit und Cache-Miss es gibt und die Cache-Hit-Rate.



2.4.4. Memory Reference Visualisation

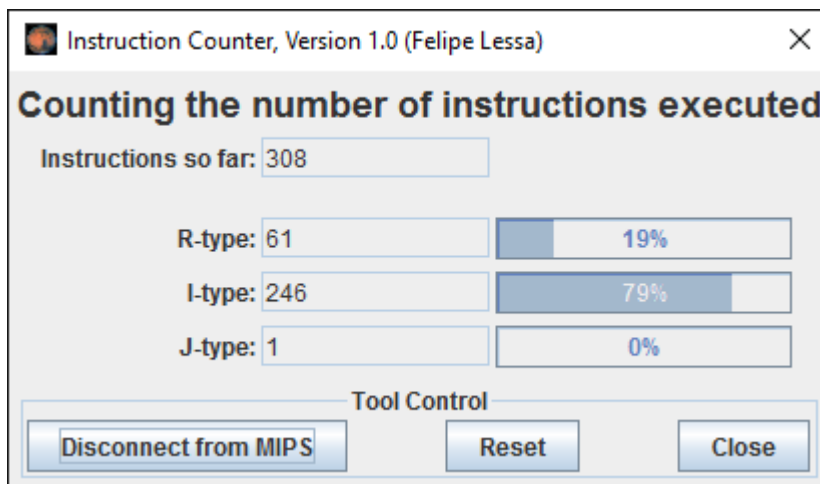
Memory Reference Visualisation stellt die Anzahl der Speicherzugriffe pro Speicherzelle grafisch dar.



3. Fibonacci

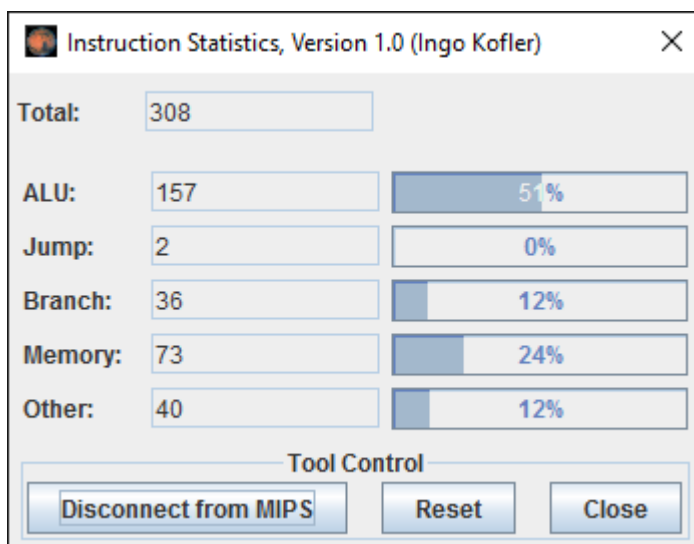
Die erste Aufgabe war es mit dem Simulator Mars das Fibonacci Demo-Programm zu assemblieren. Es sollten die Anzahl der Befehle, die Anzahl der Befehlsklassen Statistik, der Cache und Memory Reference Visualisation festgesellt werden.

3.1. Instruction Counter



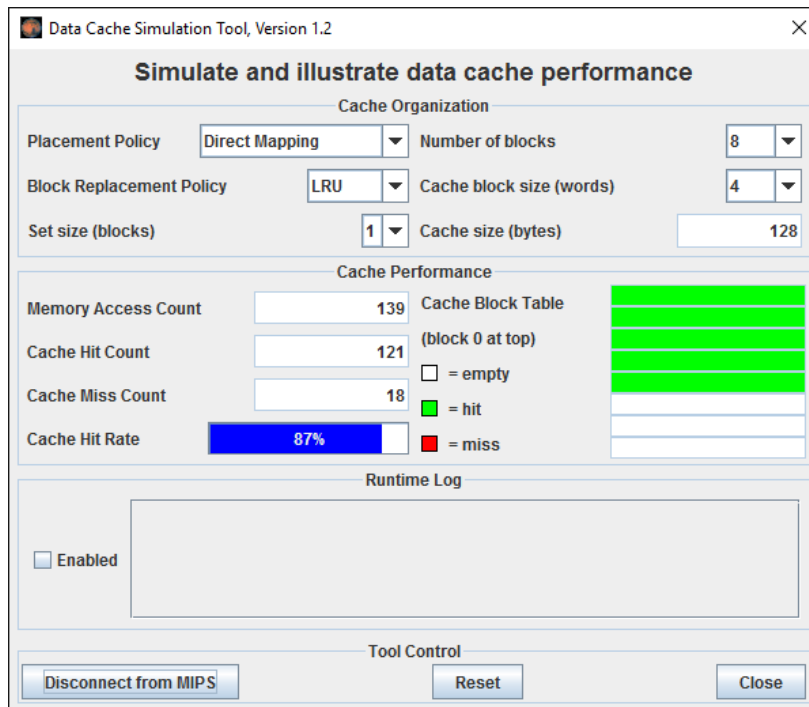
Anmerkung: Es wurden gesamt 308 Instructions ausgeführt, davon 61 Register-, 246 Immediate- und ein Jump-Befehle.

3.2. Instruction Statistic



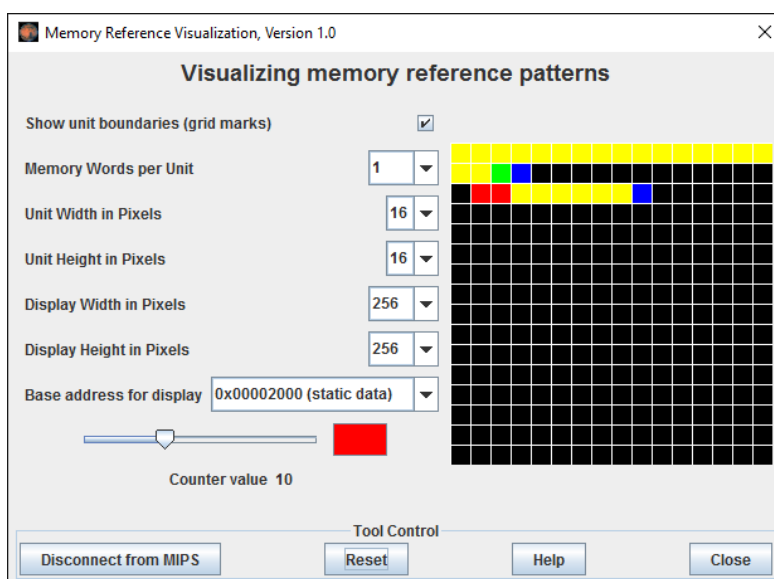
Anmerkung: Es wurden gesamt 308 Instructions ausgeführt, davon 157 ALU-, 2 Jump-, 36 Branch-, 73 Speicher-, und 40 andere-Zugriffe.

3.3. Data Cache Simulator



Anmerkung: Es wurden gesamt 139 Speicherzugriffe ausgeführt, davon waren 121 ein Cache Hit und 18 ein Cache Miss.

3.4. Memory Reference Visualisation



4. Programm

```
.eqv iport_1    0x1010
.eqv iport_2    0x1020

.eqv oport_1    0x1010

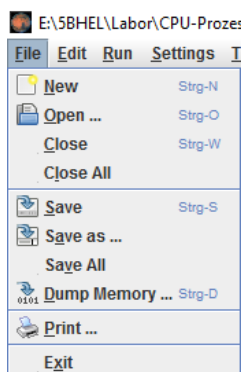
.text

J1:    addi $t0, $zero, 2    # init counting-variable
        addi $s2, $zero, 0    # init s2 (outputregister)
        lw $s0, iport_1      # read from input port 1
        lw $s1, iport_2      # read from input port 2
J2:    addu $s2, $s2, $s0    # add s0 to s2
        addu $s2, $s2, $s1    # add s1 to s2
        addi $t0, $t0, -1    # decrease t0
        bne $t0, $zero, J2    # jump if t0 is 0
        # --> s2 = (s0 + s1)*2
        sw $s2, oport_1      # output result
        j J1                  # loop
```

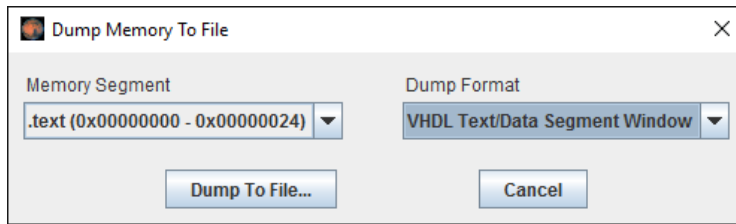
Das Programm liest von 2 Input-Ports(iport1[0x1010] & iport_2[0x1020]). Man bleibt solange in der Schleife bis die Zählvariable t0 auf 0 ist, das bedeutet, man durchläuft diese zweimal. Daraus folgt, die Formel $s2 = (s0 + s1) * 2$. S2 wird auf dem Output-Port (oport_1[0x1010]) ausgegeben.

4.1. Programm für VHDL anpassen

Das Programm kann mittels dem Menüpunkt File→“Dump Memory“ exportiert werden.



Um den Maschinencode kompatibel mit VHDL zu machen, musste man beim „Dump Format“→“VHDL Text/Data Segment Window“ auswählen, wie man an dem unten beigefügtem Screenshot sehen kann.



Bemerkung: Um exportieren zu können, wird der Maschinencode benötigt.

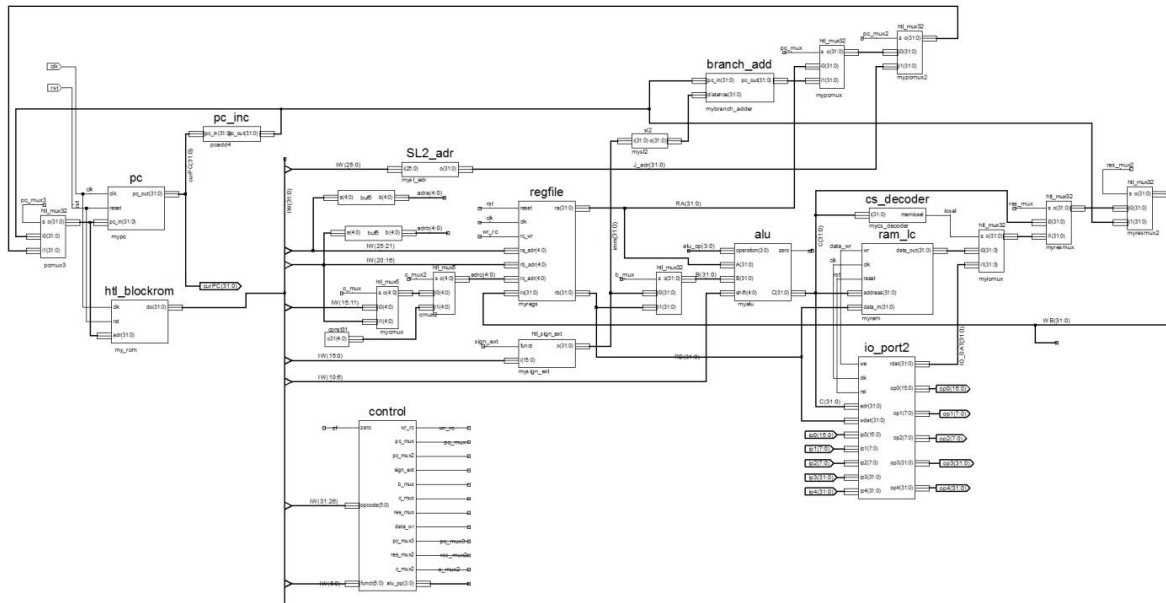
Das folgende erstellte ROM sah wie folgt aus.

```
constant rom: rom_type := (
    -- [Address]      Disassemble      | Source
    (x"20080002"), -- [0x00000000] addi $8,$0,0x00000002 | J1 : addi $t0, $zero, 2 # init counting-variable
    (x"20120000"), -- [0x00000004] addi $18,$0,0x00000000 | addi $s2, $zero, 0 # init s2 (outputreg)
    (x"8c101010"), -- [0x00000008] lw $16,0x00001010($0) | lw $s0, 0x1010 # read from input port 1
    (x"8c111020"), -- [0x0000000c] lw $17,0x00001020($0) | lw $s1, 0x1020 # read from input port 2
    (x"02509021"), -- [0x00000010] addu $18,$18,$16 | J2: addu $s2, $s2, $s0 # add s0 to s2
    (x"02519021"), -- [0x00000014] addu $18,$18,$17 | addu $s2, $s2, $s1 # add s1 to s2
    (x"2108ffff"), -- [0x00000018] addi $8,$8,0xffffffff | addi $t0, $t0, -1 # decrease t0
    (x"1500fffc"), -- [0x0000001c] bne $8,$0,0xffffffffc | bne $t0, $zero, J2 # jump if t0 is 0
    (x"ac121010"), -- [0x00000020] sw $18,0x00001010($0) | sw $s2, 0x1010 # output result
    (x"08000000"), -- [0x00000024] j 0x00000000 | j J1 # loop

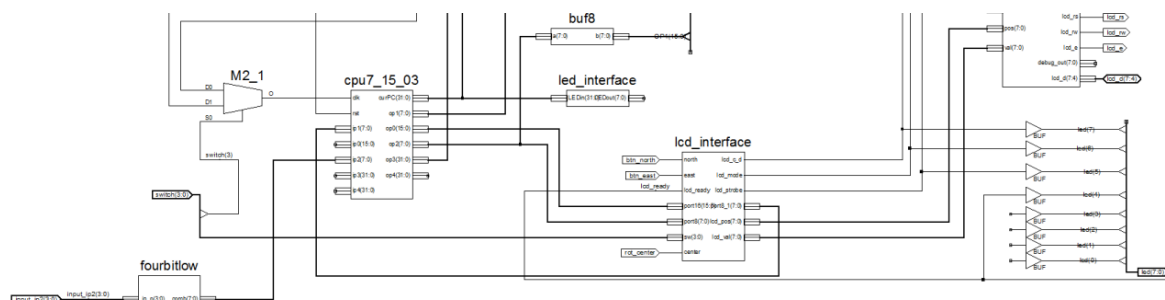
    -- Den Rest mit NOP (sll $zero, $zero, 0) füllen
    OTHERS => x"0000_0000"
);
```

4.2. Implementierung in VHDL-Projekt

4.3. Topsheet der CPU



Die ROM-Table wurde bei der Entity „hll_blockrom“ eingefügt, somit ist das eigene Programm im ROM.



Da wir 2 Input-Ports gewählt haben, musste eine DIP-Switch mittels externer Hardware inkludiert werden, dies musste man softwaretechnisch lösen. Deshalb wurde das Entity „fourbitlow“ geschrieben. Dieses Entity fügt für die bits 0:3 die Schalterstellung ein und für 7:4 wird ein „0er“ geschrieben.

4.4.fourbitlow-Entity

```
entity fourbitlow is
    Port ( comb : out STD_LOGIC_VECTOR (7 downto 0);
          in_p  : in STD_LOGIC_VECTOR (3 downto 0));
end fourbitlow;

architecture Behavioral of fourbitlow is

begin
    comb(7 downto 4) <= "0000"; --fügt 0er ein
    comb(3 downto 0) <= in_p;  --liest vom Schalter
end Behavioral;
```

4.5.UCF-File anpassen

```
# Period constraint for 50MHz operation
#
NET "clk" PERIOD = 20.0ns HIGH 50%;
#
# I/O constraints for Spartan-3E Starter Kit (Rev.B)
# soldered 50MHz Clock
NET "clk" LOC = "C9" | IOSTANDARD = LVTTTL;
# Simple LEDs
NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 4;
# SPI devices
NET "spi_sck" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_sdi" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_sdo" LOC = "N10" | IOSTANDARD = LVCMOS33;
NET "spi_rom_cs" LOC = "U3" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_amp_cs" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_adc_conv" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_dac_cs" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_amp_shdn" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "spi_amp_sdo" LOC = "E18" | IOSTANDARD = LVCMOS33;
NET "spi_dac_clr" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8;
NET "strataflash_oe" LOC = "C18" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_ce" LOC = "D16" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "strataflash_we" LOC = "D17" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "platformflash_oe" LOC = "T3" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;

# LCD display
NET "lcd_rs" LOC = "L18" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_rw" LOC = "L17" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_e" LOC = "M18" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<4>" LOC = "R15" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<5>" LOC = "R16" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<6>" LOC = "P17" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
NET "lcd_d<7>" LOC = "M15" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 2;
# Simple I/O ports on J1
NET "input_ip2<0>" LOC = "B4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "input_ip2<1>" LOC = "A4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "input_ip2<2>" LOC = "D5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
NET "input_ip2<3>" LOC = "C5" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 4;
# Simple switches
NET "switch<0>" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP;
NET "switch<1>" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP;
NET "switch<2>" LOC = "H18" | IOSTANDARD = LVTTTL | PULLUP;
NET "switch<3>" LOC = "N17" | IOSTANDARD = LVTTTL | PULLUP;
# Press buttons
NET "btn_north" LOC = "V4" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "btn_east" LOC = "H13" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "btn_south" LOC = "K17" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "btn_west" LOC = "D18" | IOSTANDARD = LVTTTL | PULLDOWN;
#NET "btn_west" CLOCK_DEDICATED_ROUTE = FALSE;
NET "rot_center" LOC = "V16" | IOSTANDARD = LVTTTL | PULLDOWN;
# End of File
```

Das UCF-File wurde bearbeitet.

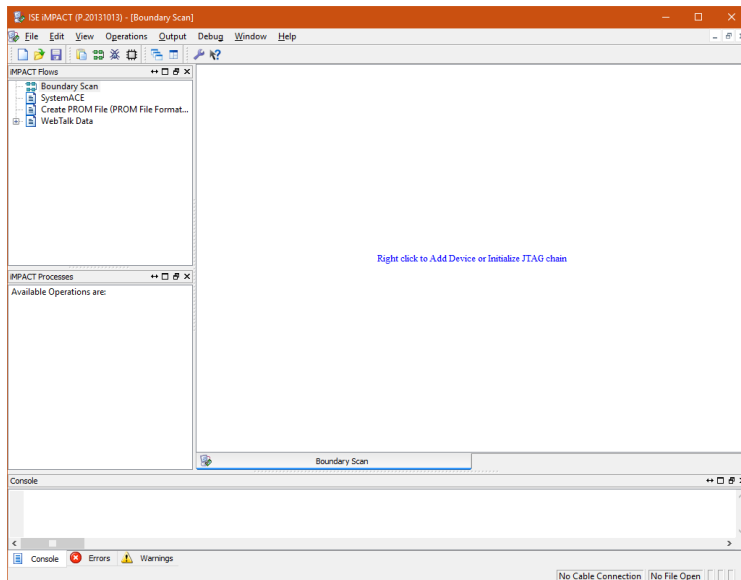
4.5. Testbench der Gesamtfunktion

Dieser Teil der Übung wurde aufgrund eines Missverständnisses nicht realisiert.

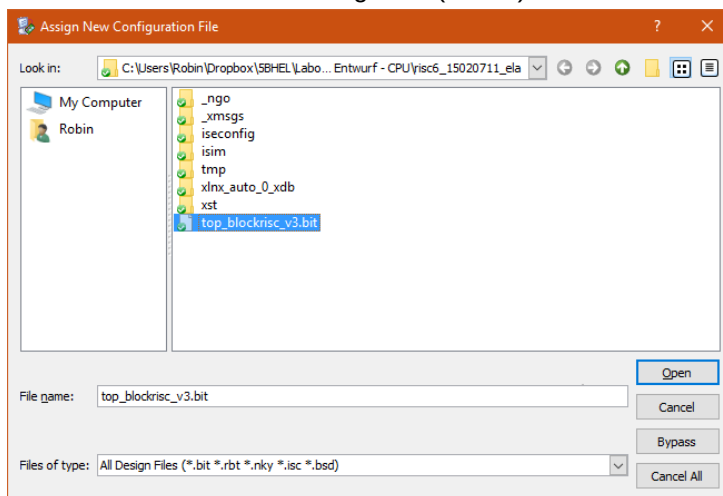
5. Hardwaretest

Die fehlerfreie Kompilierung des Programmes wurde getestet, in dem man ein „Generate Programming File“ erstellte. Muss das Programm auf die Hardware zu flashen, wurde iMPACT verwendet.

Um ISE iMPACT starten zu können, musste man „Configure Target Device“ starten. Danach sollte sich dieses Fenster öffnen.

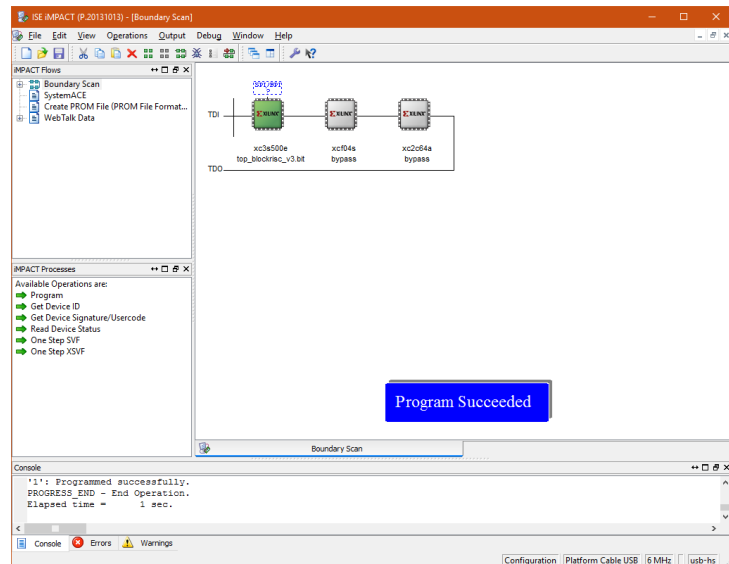


Nun musste man auf einen Rechtsklick auf „Boundary Scan“ machen → wählte man Initialize Chain → wieder ein Rechtsklick → Programm(FPGA) auswählen.



Hier musste das File ausgewählt werden, welches auch geflasht werden sollte

Falls alles einwandfrei funktioniert hatt, sollte Programm Succeeded kommen.



Die Funktion wurde erfolgreich auf dem Board getestet.

6. Erkenntnis

- Verständnis der CPU und deren Arbeitsweise
- Einführung in Mars
- Auffrischung von Assembler
- Auffrischung von VHDL

7.Messprotokoll

8. Inventarliste