

Schritt-für-Schritt-Anleitung für den Test des LMB-IO-Systems mit ChipScope

Prof. G. Kemnitz, TU Clausthal, Institut für Informatik

22. April 2016

Zusammenfassung

Das in der vorherigen Anleitung erzeugte System wird in dieser Anleitung um integrierten Logikanalysator für unterschiedliche Testaufgaben erweitert. Zuerst sollen in analoger Weise wie in Anleitung 3 die Trace- und Led-Signale für einen inneren Schleifendurchlauf beim Hochzählen eines Led-Ausgabewerts untersucht werden. Ziel ist die Abschätzung, wie lange der Schreibzugriff über den LMB auf ein Ausgaberegister dauert. Im zweiten Versuch werden die Ausgabesignale des Festwert-Timers FIT1, und wie lange die Programmreaktion auf ein Timer-Event dauert, untersucht. Im dritten Versuch werden die Signale auf dem LMB untersucht. Da es für den LMB keinen Busanalysator wie für den AXI-Bus gibt, wird ein anderes Werkzeug, der »ChipScope Core Inserter« genutzt. Mit dem »Core Inserter« lassen sich auch alternative Signale als Aufzeichnungstakt verwenden. Als Beispiel hierfür wird im letzten Versuch ein integrierter Logikanalysator für die Aufzeichnung des UART-Sende- und Empfangssignal mit dem 500 kHz Signal am Toggle-Ausgang von FIT1 getaktet.

1 Kopieren des LMB-IO-Systems in ein neues Projekt

Um das Projekt der vorherigen Anleitung nicht zu verändern, werden die Entwurfsergebnisse in Form der mhs- und der ucf-Datei in ein neues Projekt übernommen. Dazu ist in EDK ein neues Projekt zu erzeugen. In dem sich öffnenden Fenster ist der FPGA-Typ und das mhs-File des vorherigen Projekts wie in Abb. 1 auszuwählen¹. Nach Erzeugen des neuen Projekt ist die Datei »system.ucf« im Unterverzeichnis »/data« durch die des vorherigen Projekts zu ersetzen.

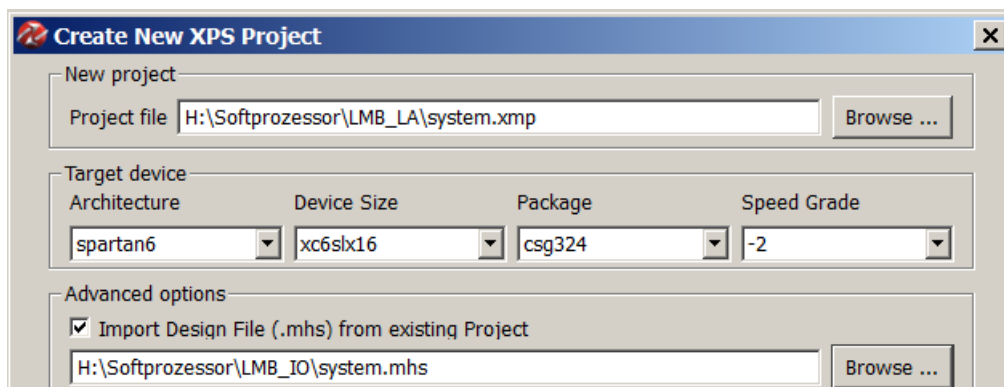


Abbildung 1: Erzeugung eines neuen Projekts der bisherigen Schaltung

¹Alle Auswahlfelder darunter sind abzuwählen.

Abb. 2 zeigt das Blockschaltbild des importierten Rechnersystems ohne integrierte Logikanalysatoren. Es enthält den Prozessor mit je einem LMB (local memory busses) für Befehle und Daten. Beide LMB's sind über einen Speicher-Controller mit je einem Port des Dual-Port-Block-RAM's verbunden. Zusätzlich ist am Daten-LMB das IO-Modul mit den parallelen Schnittstellen, Timern und der UART angeschlossen. Nur der Debug-Controller nutzt weiterhin den AXI-Bus.

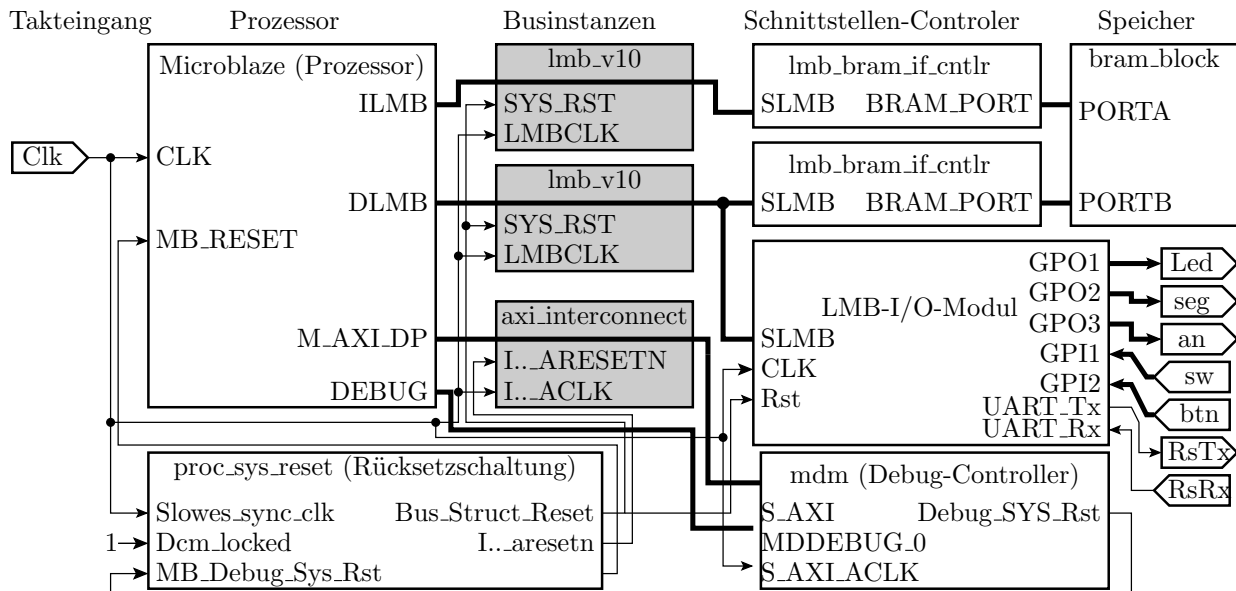


Abbildung 2: Rechnersystem ohne integrierte Logikanalysatoren

In Vorbereitung des Tests mit einem integrierten Logikanalysator empfiehlt es sich, den zu beobachtenden internen Signalen und den Bausteinen, deren Anschlussignale zu beobachten sind, eigene aussagekräftige Namen zu geben. Das geht am einfachsten mit »Replace All« im

```

PORT Clk = Clk_x, DIR = I,...
PORT sw = sw_x, DIR = I,..
PORT btn = btn_x, DIR = I,..
PORT Led = led_x, DIR = 0,..
PORT seg = seg_x, DIR = 0,..
PORT an = an_x, DIR = 0,..
PORT RsRx = Rx, DIR = I
PORT RsTx = Tx, DIR = 0
PORT T500kHz = T500kHz_x, DIR = 0

BEGIN microblaze
  PARAMETER INSTANCE = cpu
  ...
END

BEGIN lmb_v10
  PARAMETER INSTANCE = ilmb
  ...
END

BEGIN lmb_v10
  PARAMETER INSTANCE = dlmb
  ...
END

...
BEGIN lmb_bram_if_cntlr
  PARAMETER INSTANCE = dbram_cntlr
  ...
END

BEGIN lmb_bram_if_cntlr
  PARAMETER INSTANCE = ibram_cntlr
  ...
END

BEGIN bram_block
  PARAMETER INSTANCE = bram
  ...
END

BEGIN iomodule
  PARAMETER INSTANCE = io
  ...
  PORT FIT1_Toggle = T500kHz_x
END

```

Abbildung 3: Umbenennungen und Ergänzungen im mhs-File

mhs-File. Die internen Anschlussignale sollen den Port-Namen mit angehängtem »_x«, der Prozessor den Namen »cpu«, die lokalen Speicherbusse die Namen »ilmb« und »dlmb« (i – instruction; d – data), die Speicher-Controller die Namen »ibram_cntrl« und »dbram_cntrl« und der Block-Ram den Namen »bram« bekommen (Abb. 3). Für den letzten Versuch mit dem 500 kHz-Toggle-Ausgang von Timer »FIT1« als Aufzeichnungstakt des Logikanalysators ist es erforderlich, den Toggle-Ausgang des Timers als Schaltungsausgang herauszuführen².

2 Logikanalysator für die Trace- und Led-Signale

Für den ersten Versuch soll der Logikanalysator wie in Anleitung 3 die Trace-Signale für den Befehlszähler, das Befehlswort und »Befehlswort gültig« sowie die Led-Ausgabe beobachten. Für den zweiten Versuch sollen zusätzlich die Signalverläufe des Toggle- und des Interrupt-Ausgabesignals des Festwert-Timer FIT1 mitgeschrieben werden. Das sind alles Signale, die über das Menü »Debug Configuration« und dort den mittleren Punkt »To monitor arbitrary level signal (adding ILA)« beobachtbar gemacht werden können. Zugang zum Konfigurationsmenü:

```
Debug > Debug Configuration
  Schaltfläche »Add ChipScope Peripheral..«
  Auswahlpunkt »To monitor arbitrary system level signals ..«
```

Im Auswahlfenster »Basic« Abb. 4 sind der Trigger-Gruppe »TRIG0« vom IO-Modul die Led-Ausgabe ».._GPO1« sowie die Timer-Signale »..FIT1_Interrupt..« und »..FIT1_Toggle..« zuzuordnen.

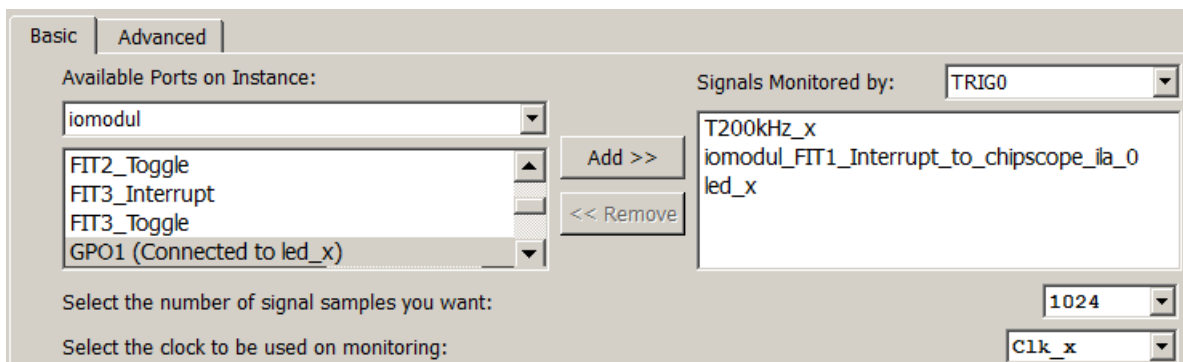


Abbildung 4: Verbindung der zu beobachtenden Ausgänge der IO-Schnittstelle mit »TRIG0«

Die Trace-Signale des Prozessors für die Befehlsadresse, die Befehlsdaten und die Befehls-gültigkeit sollen der Trigger-Gruppe »TRIG1« zugeordnet werden (Abb. 5). Aufzeichnungstakt sei der interne Eingabetakt »Clk_x«.

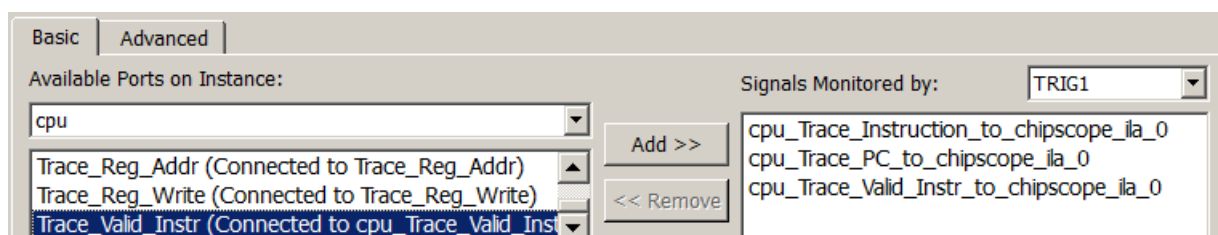


Abbildung 5: Verbindung der Trace-Signale mit »TRIG1«

²Der Netzlistengenerator des IO-Moduls erzeugt sonst keinen Toggle-Ausgang für FIT1.

Nach Schließen der »Debug Configuration« sind nachträglich die Boudary-Scan-Verbindungen zwischen dem Debug-Modul »mdm« und dem Controller des integrierten Logikanalysators »ICON« im mhs-File zu ergänzen:

```

BEGIN mdm
...
PARAMETER C_USE_BSCAN = 1
...
PORT bscan_tdi = v_tdi
PORT bscan_reset = v_reset
PORT bscan_shift = v_shift
PORT bscan_update = v_update
PORT bscan_capture = v_capture
PORT bscan_sel1 = v_sel1
PORT bscan_drck1 = v_drck1
PORT bscan_tdo1 = v_tdo
END
...
BEGIN chipscope_icon
...
PORT tdi_in = v_tdi
PORT reset_in = v_reset
PORT shift_in = v_shift
PORT update_in = v_update
PORT capture_in = v_capture
PORT sel_in = v_sel1
PORT drck_in = v_drck1
PORT tdo_out = v_tdo
END

```

Der Parameter »C_USE_BSCAN = 1« konfiguriert dabei den »mdm« so, dass er den Boundary-Scan-Port für den »ICON« bereitstellt. Anschließend empfiehlt sich die Kontrolle mit »Run DRCs«. Diese sollte weder Fehlermeldungen noch Warnungen liefern. Die Generierung des Bit-Files und der Daten für den Software-Entwurf erfolgt wieder mit

Export Design ▷ Export & Launch SDK

wobei bei »Include bitstream and BMM file« der Haken zu setzen ist. Der Arbeitsraum in SDK für den Software-Entwurf sei »SW_LMB_LA«.

3 Testvorbereitung in SDK

Nach »Export & Launch SDK« ist in SDK als erstes das »Board Support Package« zu erzeugen:

File ▷ New ▷ Board Support Package

Einstellungen »Use Default Location«, »standalone« und ohne Einbindung von weiteren Bibliotheken. Nach dem »Board Support Package« ist wieder eine Anwendung einzurichten:

File ▷ New ▷ Application Project

Der Name sei »ILA_LedTest« und das Template wieder »Empty Application«. Danach ist mit Rechtsklick auf »ILA-LedTest« über

New ▷ Source File

eine Quelldatei mit dem Namen »ila_ledtest.c« anzulegen und in diese das nachfolgende Programm zu schreiben, das die Leuchtdiodenausgabe hochzählt:

```

#include <xparameters.h>
#include <xio.h>
#define ioadr_led XPAR_IOMODUL_BASEADDR + 0x10
void main(){
    u8 dat=0;
    while (1){
        XIo_Out8(ioadr_led, dat);
        dat++;
    }
}

```

Nach Programmeingabe und erfolgreichem Übersetzen ist der FPGA zu programmieren

```
Xilinx Tools > Program FPGA
```

mit

```
Run > Run Configuration
```

Rechtsklick auf »Xilinx C/C++ application« und »New« eine »Run Configuration« einzurichten und das Programm mit »Run« zu starten. Zur Auswertung der aufzuzeichnenden Befehlswoorte und -adressen wird das disassemblierte Maschinenprogramm benötigt. Die Schritte, um dieses aus dem Elf-File des Programms zu gewinnen, sind:

- in SDK eine Xilinx-Shell öffnen:

```
Xilinx Tools > Launch Shell
```

- mit »cd« in das Verzeichnis »ILA_LedTest\Debug« wechseln,
- Disassemblierung und Abspeicherung in eine Textdatei mit

```
mb-objdump -d ILA_LedTest.elf > ILA_LedTest.txt
```

- öffnen der Textdatei »ILA_LedTest.txt« mit einem Editor und Extraktion der interessierenden Programmbereiche.

Das Hauptprogramm mit der zu untersuchenden Schleife beginnt ab der Marke »main« (Befehlsadresse 0x610) und endet mit dem Rücksprung am Ende der Endlosschleife (Befehlsadresse 0x638). In groben Zügen ist das dasselbe Programm wie in Anleitung 3. Zu untersuchen ist, wie viele Takte die Ausführung des Speicherbefehls auf Adresse 0x628, der den Wert der Variablen »dat« an die Led's ausgibt. Bei der Ausgabe über den AXI-Bus waren es nahezu 10 Takte.

```
00000610 <main>:
610: 3021fff4  addik r1, r1, -12 (Stack- und Frame-
614: fa610008  swi  r19, r1, 8  pointer einrichten)
618: 12610000  addk  r19, r1, r0
61c: f0130004  sbi  r0, r19, 4
620: b0000004  imm  4
624: 30600010  addik r3, r0, 16 (r3 := 0x40010)
628: e0930004  lbui r4, r19, 4 (r4 := mem(FP+4))
62c: f0830000  sbi  r4, r3, 0  (mem(r3) := r4)
630: e0730004  lbui r3, r19, 4 (r3 := mem(FP+4))
634: 30630001  addik r3, r3, 1  (r3 := r3 + 1)
638: f0730004  sbi  r3, r19, 4 (mem(FP+4) := r3)
63c: b800ffe4  bri  -28 (Sprung nach 0x620)
```


(FP – Frame-Pointer; FP+4 – Adresse von Dat; 0x40010 – Adresse des Led-Ausgaberegisters.

Abbildung 6: Disassembliertes Hauptprogramm

4 Untersuchung der Trace- und der Led-Ausgabe

Nach dem das Programm auf dem Softprozessor gestartet ist, soll auch das Programm zur Ansteuerung des integrierten Logikanalysators auf dem PC gestartet werden:

Windows-Startmenü ▷ All Programms ▷ Xilinx Design Tools
 ▷ ISE Design ... ▷ ChipScope Pro ▷ ChipScope 64-bit ▷ Analyzer

(Achtung 64-Bitversion verwenden!). Die Verbindung mit dem FPGA wird wieder mit einem Klick auf die Schaltfläche  oben links hergestellt. Der FPGA ist auszuwählen und aus dem Wrapper-Verzeichnis des integrierten Logikanalysator ist die cds-Datei, in der die Signalnamen stehen, zu importieren (Abb. 7).

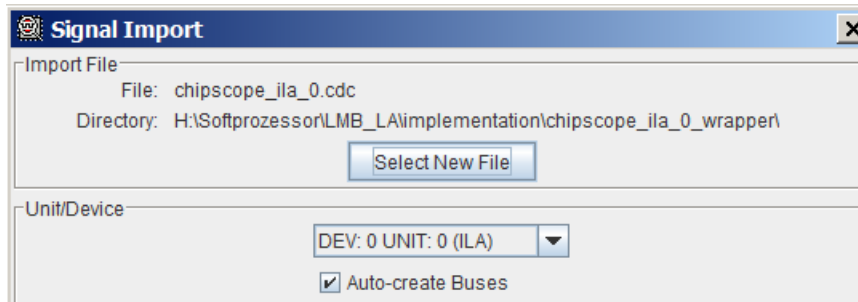


Abbildung 7: Import-Einstellungen für die cdc-Datei

Für den ersten Versuch soll der Logikanalysator triggern, sobald auf die Led's der Wert sieben (binär 0b111) ausgegeben wird (Abb. 8).

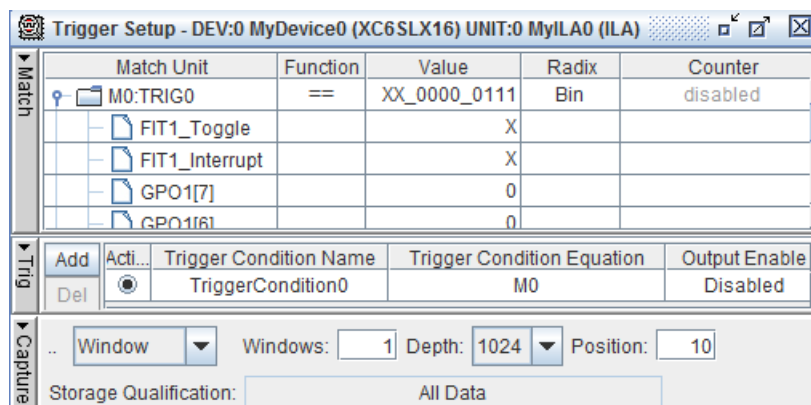




Abbildung 8: Trigger zehn Takte vor der Led-Ausgabe 0b111

Nach Start mit , ausreichendem Zoom und der Einstellung von »Reverse Bus Order« für die Befehlsdaten und -adressen wird für den Schleifendurchlauf von der Ausgabe 0b111 bis zur Ausgabe von 0b1000 an die Led's der Signalverlauf in Abb. 9 aufgezeichnet³. Der Led-Ausgabebefehl mit der Befehlsadresse 0x62C »sbi r4, r3, 0« (mem(r3) := r4) hat im Trace nur zwei Wartetakte davor und keinen Wartetakt danach. Die Ausgabe über den LMB (Local Memory Bus) dauert offenbar nicht länger als drei Takte. Das ist nur ein Drittel der Zeit für eine Datenausgabe über den AXI-Bus. Abb. 10 zeigt die aufgezeichneten Trace-Signal aus Abb. 9 nochmal als Listing, in dem zu allen Befehlsworten mit »Trace_Valid_Instr=1« die Assemblernotation und die Funktion mit angegeben sind. Die Sample-Zählung im Listing beginnt mit null. Bei der Zeitzählung ist der Nullpunkt der Triggerzeitpunkt.

³Hinweis zur Problembeseitigung: Nach jedem Programmneustart in SDK oder, wenn statt der erwarteten Signalverläufe mit Chipscope für viele Takte hintereinander Einsen aufgezeichnet werden, ist mit »JTAG Chain ▷ Close Cable« die Kabelverbindung zu trennen und mit  neu zu öffnen.

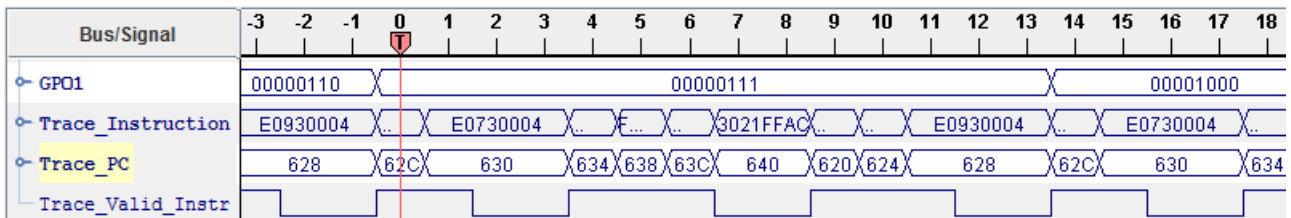


Abbildung 9: Aufgezeichnete Trace-Signale für einen Schleifendurchlauf

Sample / Zeit.	Led-Ausgabe (GPO1)	abgeschlossener Befehl			
		PC	Instruction	disassembl.	Funktion
9 / -1	0000 0110	0x628	ungültig		
10 / 0	0000 0111	0x62c	0xF0830000	sbi r4, r3, 0	mem(r3) := r4
11 / 1	0000 0111	0x630	0xE0730004	lbui r3, r19, 4	r3 := mem(FP+4)
12 / 2	0000 0111	0x630	ungültig		
13 / 3	0000 0111	0x630	ungültig		
14 / 4	0000 0111	0x634	0x30630001	addik r3, r3, 1	r3 := r3 + 1
15 / 5	0000 0111	0x638	0xF0730004	sbi r3, r19, 4	mem(FP+4) := r3
16 / 6	0000 0111	0x63C	0xB800FFE4	bri -28	Sprung nach 0x620
17 / 7	0000 0111	0x640	ungültig		
18 / 8	0000 0111	0x640	ungültig		
19 / 9	0000 0111	0x620	0xB0000004	imm 4	
20 / 10	0000 0111	0x624	0x30600010	addik r3, r0, 16	r3 := 0x40010
21 / 11	0000 0111	0x628	0xF0830000	sbi r4, r3, 0	mem(r3) := r4
22 / 12	0000 0111	0x628	ungültig		
23 / 13	0000 0111	0x628	ungültig		
24 / 14	0000 1000	0x62C	0xE0730004	lbui r3, r19, 4	r3 := mem(FP+4)

Abbildung 10: Listing der Trace-Signale zu Abb. 9 mit disassemblierten Befehlen

5 Untersuchung der Timer-Ausgabesignale

Die Festwert-Timer haben eine in der Hardware-Konfiguration mit »Number of Clocks between Strobes« definierte Zählperiode, keine Ein- und Ausschaltmöglichkeit und zwei Ausgänge, die mit »Toggle« und »Interrupt« bezeichnet sind. Es soll untersucht werden, was an diesen Ausgänge für Signale erzeugt werden. Das Testprogramm sei eine Endlosschleife, die nichts tut:

```
void main(){
    while (1){};
}
```

Mit der Trigger-Bedingung ».._Interrupt=1«, alle anderen Signale »X« und »Position=10« ergeben sich die Signalverläufe in Abb. 11. Alle 100 Takte wird das Interruptsignal für einen Takt aktiv. Im Folgetakt invertiert das Toggle-Signal seinen Wert.

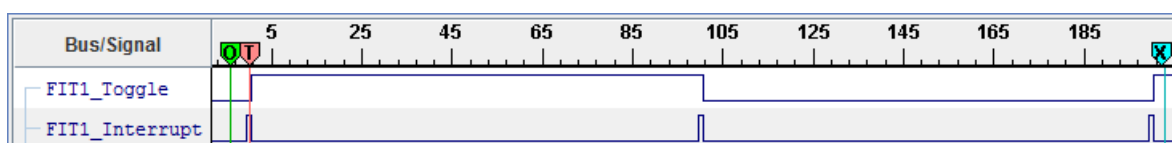


Abbildung 11: Ausgabesignalverläufe des Festwert-Timers FIT1

In einem weiterführenden Experiment wird untersucht, wie lange die Programmreaktion auf ein Timer-Event dauert. Das Programm prüft in einer Endlosschleife das Interrupt-Bit für Timer »FIT1«. Falls es gesetzt ist, wird es gelöscht, der Wert der Variablen »dat« auf die Led's ausgegeben und weitergezählt:

```
#include <xparameters.h>
#include <xio.h>
#define ioadr_led XPAR_IOMODUL_BASEADDR + 0x10
#define ioadr_IRQ_STATUS XPAR_IOMODUL_BASEADDR + 0x30
#define ioadr_IRQ_ACK XPAR_IOMODUL_BASEADDR + 0x3C
#define FIT1_EventNr 7
void main(){
    u8 dat=0;
    while (1){
        if (XIo_In32(ioadr_IRQ_STATUS) & 1<<FIT1_EventNr){
            XIo_Out32(ioadr_IRQ_ACK, 1<<FIT1_EventNr);
            XIo_Out8(ioadr_led, dat);
        }
    }
}
```

Die interessierende Frage ist, wie viele Takte vergehen vom Setzen des Interrupt-Bits bis zur Änderung der Led-Ausgabe. Abb. 12 zeigt mit der Triggerbedingung »GPO1=0x07« und »FIT1_Interrupt=1« aufgezeichneten Signalverläufe. Im Bild vergehen 20 Takte, bis sich nach Aktivierung des Interuptsignals der Led-Wert ändert. Mehrmalige Versuchswiederholung zeigt, dass die Verzögerung auch bis zu 26 Takte sein kann.

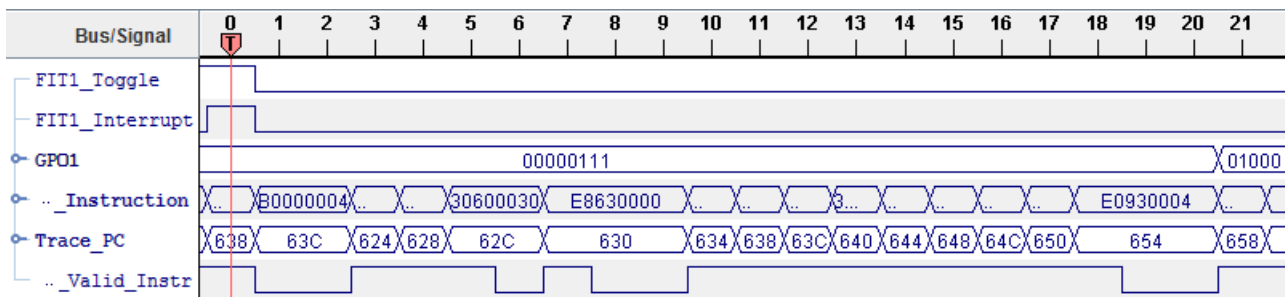


Abbildung 12: Ausgabverläufe des Timer-Led-Programms

Die Interpretation der Signalverläufe erfordert wieder die disassemblierte Befehlsfolge ⁴:

```
loop:                                // Beginn Endlosschleife
624: 80000000  or r0, r0, r0    // nop-Befehl
628: b0000004  imm 4
62c: 30600030  addik r3, r0, 48 // r3 := ioadr_IRQ_STATUS .._IRQ_STATUS
630: e8630000  lwi r3, r3, 0    // r3 := Wert(.._IRQ_STATUS)
634: a4630080  andi r3, r3, 128 // alle Bits außer FIT1_EventNr löschen
638: bc03ffec  beqi r3, -20     // wenn r3=0, Sprung zu loop
63c: b0000004  imm 4
640: 3060003c  addik r3, r0, 60 // r3 := Adresse .._IRQ_ACK
644: 30800080  addik r4, r0, 128 // r4 := (1<<FIT1_EventNr)
648: f8830000  swi r4, r3, 0    // Löschen Ereignisbit
```

⁴Schritte zur Disassemblierung: Auf der Xilinx-Console in das Verzeichnis »..\ILA_FitTest\Debug« wechseln, ein »Dump« vom Elf-File zu erzeugen und mit einem Editor die Befehlsfolge in der Endlosschleife zu extrahieren.


```

64c: b0000004  imm 4
650: 30600010  addik r3, r0, 16 // r3 := Adresse .._led
654: e0930004  lbui r4, r19, 4 // r4 := dat;
658: f0830000  sbi r4, r3, 0 // Led-Ausgabe := dat
65c: e0730004  lbui r3, r19, 4 // r4 := dat;
660: 30630001  addik r3, r3, 1 // r4++;
664: f0730004  sbi r3, r19, 4 // dat := r4
668: b800ffbc  bri -68 // Sprung zu loop

```

Aus den aufgezeichneten Signalverläufen und dem disassemblierten Programmausschnitt ergibt sich das Trace-Listing in Abb. 13:

Zeitpunkt	Interrupt	GPO1 (Led)	abgeschlossener Befehl			
			PC	Instruction	disassembl.	Funktion
0	1	0b111	0x638	0xBC03FFEC	beqi r3, -20	wenn r3=0, Sprung zu loop
1	0	0b111	0x63C	ungültig		
2	0	0b111	0x63C	ungültig		
3	0	0b111	0x624	0x80000000	or r0, r0, r0	nop-Befehl
4	0	0b111	0x628	0xB0000004	imm 4	
5	0	0b111	0x62C	0x30600030	r3, r0, 48	r3 := ioadr_IRQ_STATUS
6	0	0b111	0x62C	ungültig		
7	0	0b111	0x630	0xE8630000	lwi r3, r3, 0	r3 := Wert(.._IRQ_STATUS)
8	0	0b111	0x630	ungültig		
9	0	0b111	0x630	ungültig		
10	0	0b111	0x634	0xA4630080	andi r3, r3, 128	Bits außer FIT1_EventNr löschen
11	0	0b111	0x638	0xBC03FFEC	beqi r3, -20	wenn r3=0, Sprung zu loop
12	0	0b111	0x63C	0xB0000004	imm 4	
13	0	0b111	0x640	0x3060003C	addik r3, r0, 60	
14	0	0b111	0x644	0xA4630080	andi r3, r3, 128	Bits außer FIT1_EventNr löschen
15	0	0b111	0x648	0xF8830000	swi r4, r3, 0	Löschen Ereignisbit
16	0	0b111	0x64C	0xB0000004	imm 4	
17	0	0b111	0x650	0x30600010	addik r3, r0, 16	r3 := Adresse .._led
18	0	0b111	0x654	0xE0930004	lbui r4, r19, 4	r4 := dat
19	0	0b111	0x654	ungültig		
20	0	0b111	0x654	ungültig		
21	0	0b1000	0x658	0xF0830000	sbi r4, r3, 0	Led-Ausgabe := dat

Abbildung 13: Auflistung der Trace-Signale zu Abb. 12

Der Befehl zum Lesen des Interrupt-Statusregisters steht auf Adresse 0x630. In der Beispielaufzeichnung wird dieser Befehl 4 Takte vor Aktivierung und das nächste mal 8 Takte nach der Aktivierung des Interrupt-Signals abgeschlossen. Nach dem erste Lesen ist das auszuwertende Ereignisbit offenbar noch nicht gesetzt, so dass der bedingte Rücksprung auf Adresse 0x638 ausgeführt und das Statusregister noch einmal gelesen wird. Nach dem Lesen des Statusregisters und dem ausgeführten Sprung folgen jeweils zwei »Leertakte«, d.h. es wird in den zwei Folgetakte kein weiterer Befehl abgeschlossen. Nach dem zweiten Lesen des Interrupt-Statusregisters in Schritt sieben folgen wieder die beiden »Leertakte« und dann werden lückenlos acht Befehle fertig gestellt, d.h. weder nach dem »genommenen« Sprung noch nach dem Schreibbefehl zum Löschen des Ereignisbits folgen Leertakte. Der Befehl auf Adresse 0x650 schreibt den neuere Wert auf die Led's und benötigt danach wieder zwei Leertakte. Mit integrierten Logikanalysatoren lässt sich die Abarbeitung von Programmen sehr detailliert untersuchen und testen. Der Arbeitsaufwand hierfür ist jedoch erheblich.

6 Logikanalysatoreinbau mit dem »ChipScope Inserter«

In der »Debug Configuration« sind nur ausgewählte interne und Anschlussignale für den Anschluss an integrierten Logikanalysatoren zugänglich. Nicht anschließbar sind insbesondere Signale von Bussen⁵ Die LMB-Signale sind z.B. unzugänglich.

Einen alternativen Weg mit mehr Gestaltungsspielraum bietet »ChipScope Inserter«. Der »ChipScope Inserter« verlangt eine Netzliste in Form einer ngc-Datei als Eingabe und erzeugt eine geänderte ngc-Datei als Ausgabe, die zusätzlich zur Schaltung den integrierten Logikanalysator enthält. Das Rechnersystem wird ohne Logikanalysatoren entworfen. Dann wird mit »Generate Netlist« die Netzliste erzeugt, mit dem »»ChipScope Inserter« ein (oder mehrer) Logikanalysator(en) eingefügt und die Schaltungsgenerierung mit »Generate Bitstream« fortgesetzt. Für das praktische Vorgehen ist zu empfehlen:

- die Schaltung zuerst ohne die Logikanalysatoren zu entwickeln, zu übersetzen und mit Testprogrammen auszuprobieren,
- dann mit

```
Project ▷ Clean all generated Files
```

alle automatisch erzeugten und auch alle bei einem vorherigen Einbau eines Logikanalysators mit dem »ChipScope Inserter« erzeugten Dateien zu löschen und

- die Schaltungsgenerierung mit »Generate Netlist« ohne alte Dateien in den Entwurfsverzeichnissen zu starten.

Nach der Netzlistenerzeugung ist als Eingabe für den »ChipScope Inserter« eine Kopie der generierten Netzliste mit einem anderen Namen zu erzeugen. Dafür ist aus EDK Xilinx-Konsole zu öffnen

```
Project ▷ Launch Xilinx Shell
```

in das Unterverzeichnis »implementation« zu wechseln

```
cd implementation
```

und das Kopierprogramm mit den Bezeichnern der Eingabe- und der Ausgabe-ngc-Datei zu starten:

```
ngcbuild system.ngc system1.ngc.
```

Der »Core Inserter« wird über das Windows Startmenü im selben Verzeichnis wie der ChipScope-Analyzer gestartet (Abb. 14).

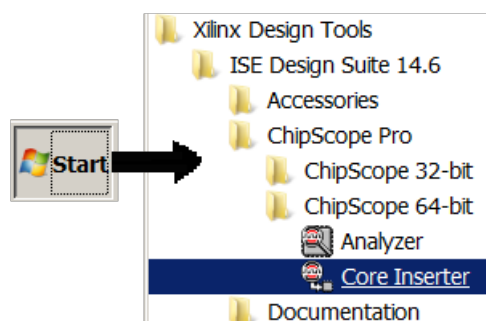


Abbildung 14: Start des »ChipScope Core Inserter«

⁵Ausgenommen sind der AXI- und der PLB-Bus, für die es spezielle Busanalysatoren gibt.

Im Startfenster des »Core Inserters« ist als »Input Design Netlist« die umbenannte Netzliste und als »Output Design Netlist« der Netzlistenbezeichner, mit dem der Generierungsprozess in EDK fortgesetzt wird, sowie der richtige Schaltkreistyp einzustellen (Abb. 15). Im nächsten Fenster des »Core Inserters« ist »USER1« zu belassen und ab dem übernächsten sich öffnenden Fenster »Trigger Parameter« (Abb. 17) beginnt die Konfiguration des Logikanalysators.

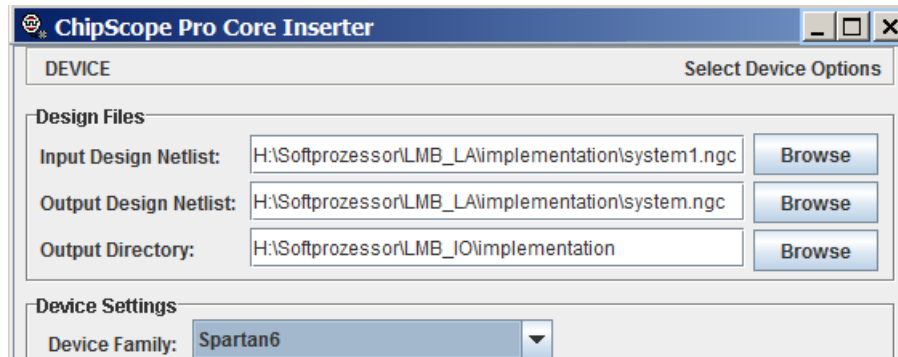


Abbildung 15: Festlegung der Ein- und Ausgabenetzliste für den »Core Inserter«

Der einzubauende Logikanalysator soll die LMB- und Trace-Signale, aus denen die zeitliche Befehlsabarbeitung rekonstruiert werden kann, sowie die Led-Ausgabe mit dem normalen Systemtakt aufzeichnen. Abb. 16 zeigt eine Skizze der Signalverbindungen. Der Prozessor hat getrennte Schnittstellen für Befehle und Daten. Die Befehlsschnittstelle sendet die Befehlsadressen über den »ILMB« und den »ibram_cntlr« an den Block-RAM-Port A und bekommt innerhalb eines Taktes die Befehlsdaten. In den Funktionsblöcken »ILMB« und »ibram_cntlr« steckt wenig Funktionalität. Die Adressen und Daten werden praktisch nur durchgereicht. Die Funktionen und Aufgaben der übrigen Steuersignale sollen später in eigenen Experimenten untersucht werden.

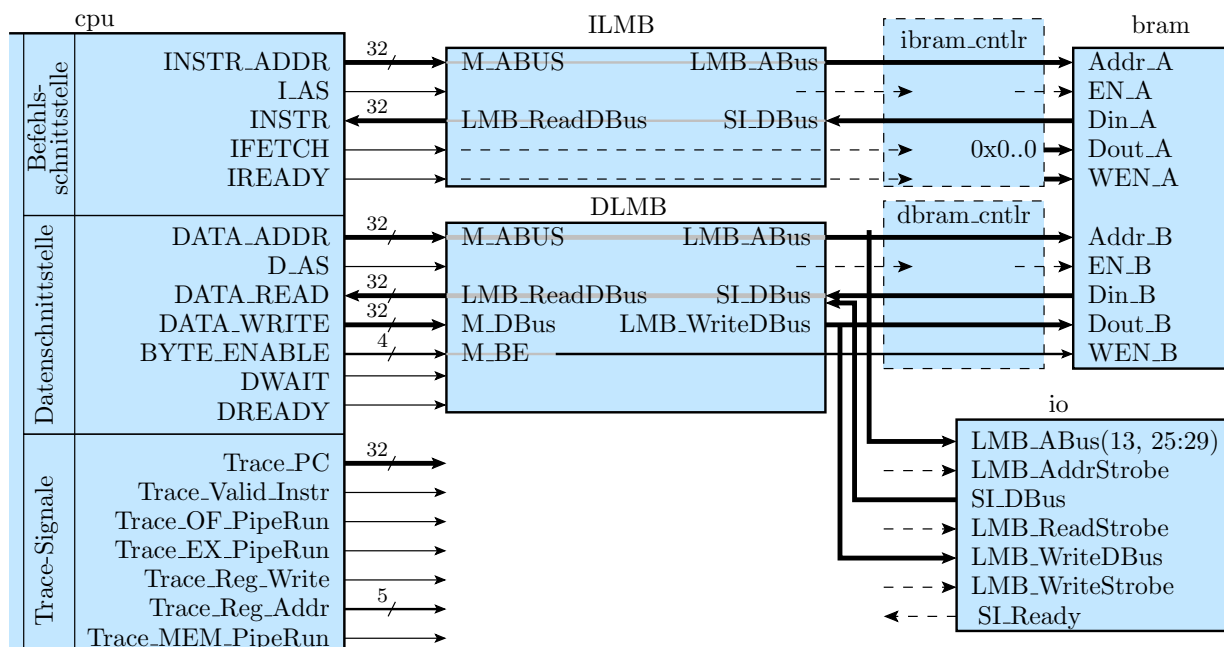


Abbildung 16: LMB- und Trace-Signale des konfigurierten Rechnersystems

Die Datenschnittstelle verhält sich ähnlich. Sie kommuniziert über den »dlmb« und den »dbram_cntlr« mit Port B des Block-RAMs und kann im Gegensatz zum Befehlsbus auch Daten schreiben. Zusätzlich hängt am Datenbus die IO-Einheit, die sich mit dem »bram_cntlr_d«

die vom Prozessor kommenden Daten teilt. Die Adresse und die Ausgabedaten werden vom Prozessor an beide angeschlossenen Slave-Einheiten praktisch nur durchgereicht. Die Lesedaten vom RAM und vom IO-Modul kommen auf getrennten 32-Bit-Signalvektoren am »DLMB« an und werden zu einem 32-Bit-Datenvektor für die Weitergabenzum Prozessor weitergeleitet. Der »DLMB« enthält offenbar einen 64-32Bit-Multiplexer für die vom Prozessor zu lesenden Daten. Für einen Test mit Logikanalysator reicht es, wenn interessierenden Adressen und Daten nur an einer Stelle beobachtet werden.

Am Trace-Port sind auch Adressen und Daten beobachtbar, aber nicht die der Befehle, deren Befehlswoorte gerade geholt werden, sondern die, die gerade fertig werden. Zusammen mit einigen weiteren Signalen lassen sich der interne Verarbeitungsfluss im Prozessor und die aktuellen Registerwerte im Prozessor rekonstruieren.

Nach bisherigen Versuchen darf der einzubauenden Logikanalysator nicht mehr als vier Block-RAMs verwenden⁶. Bei einer Aufzeichnungstiefe von 1024 kann ein aus vier Blockspeichern aufgebauter Logikanalysator bis zu 71 Dateneingänge haben. Die nachfolgende Tabelle zeigt für das folgende Experiment getroffene Signalauswahl und die geplante Signal-Kanal-Zuordnung:

Signal	LA-Kanal	Bedeutung
Led_7_OBUF bis LED_0_OBUF	CH0 bis CH7	LED-Ausgabe
cpu/INSTR_ADDR<20> bis ..<31>	CH8 bis CH19	gesendete Befehlsadresse
cpu/I_AS	CH20	Befehlsadresse gültig
cpu/INSTR<0> bis ..<31>	CH21 bis CH52	gelesenes Befehlswort
cpu/IFETCH	CH53	
cpu/IREADY	CH54	
cpu/TracePC<20> ..<31>	CH55 bis CH66	Trace-Befehlsadresse
cpu/Trace_Valid_Instr	CH67	Trace-Befehlsadresse gültig
cpu/Trace_OF_PipeRun	CH68	
cpu/Trace_EX_PipeRun	CH69	
cpu/Trace_Reg_Write	CH70	

Tabelle 1: Vom Logikanalysator aufzuzeichnende Signale

Die Led-Ausgabe ist bitgespiegelt zugeordnet und die Adressen und Daten sind bitgespiegelt definiert (der höchste Index entspricht dem niederwertigsten Bit). Das ist in der Analyzer-Anzeige wieder zu korrigieren. Da der Befehlsspeicher nur einen 8k-Byte Adressraum hat, werden nur die niederwertigen 13 bit der Befehlsadressen aufgezeichnet, einmal der Adresse, die an den Befehlsspeicher gesendet wird und einmal die, die der Trace-Port ausgibt. Von den zusätzlich aufzuzeichnenden Steuersignalen werden in den Experimenten nur die Gültigkeitssignale ausgewertet. Die restlichen dienen für weiterführende Untersuchungen.

Insgesamt sind 71 Signale zu beobachten. Diese Zahl ist im Fenster »Trigger Parameters« Abb. 17 als »Number of Input Trigger Ports« einzutragen. Im nächsten Fenster »Capture Parameter« sind »Data Depth=1024«, »Data Same As Trigger√«, »Include TRIG0...√« etc. zu belassen. Die insgesamt erforderliche Anzahl von Block-RAMs ist vier.

Im nächsten Eingabefeld »Net Connections« ist mit »Make Connections«, »Clock Signals« in das Zuordnungsmenü Abb. 18 zu wechseln und links unter »/[system]« als Aufzeichnungstakt »Clk_BUF GP« auszuwählen.

Die den ersten Trigger-Eingängen zuzuordnenden Led-Signale sind auch unter »/[system]« zu finden (Abb. 19). Alle anderen den Trigg-Ports zuzuordnenden Signale stehen im Unterbaum

⁶Aus bisher ungeklärten Gründen funktioniert sonst die sich anschließende Bitstromgenerierung nicht.

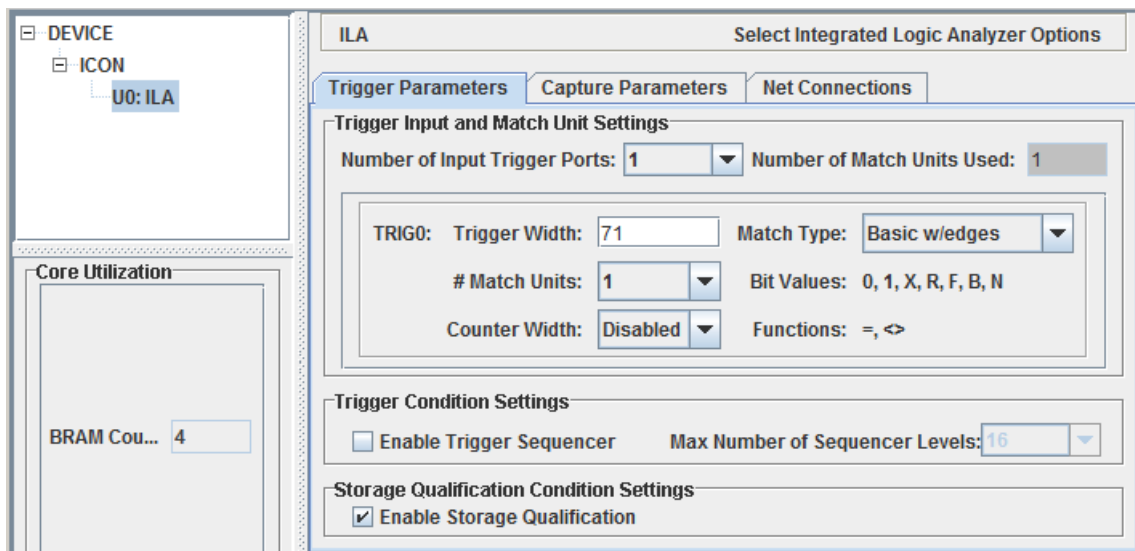


Abbildung 17: Einstellung der »Trigger Parameter«

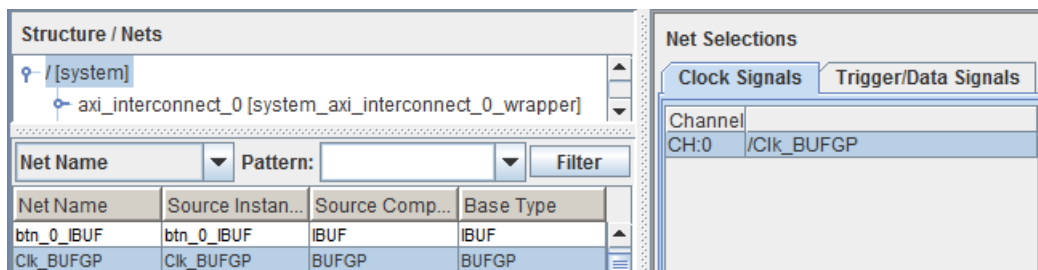


Abbildung 18: Auswahlfenster des Aufzeichnungstakts

»CPU« (Abb. 20). Mit »Net_Name«, einer Teilzeichenkette des Namens als »Pattern« und dann »Filter« lassen sich die zuzuordnenden Signale einfacher finden.

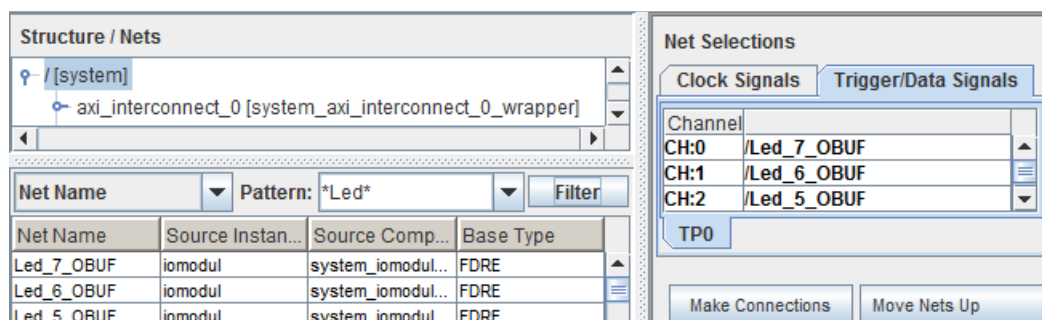


Abbildung 19: Auswahlfenster für die Ein- und Ausgabenetzliste

Nach der Auswahl der Aufzeichnungstakts und der Zuordnung der 71 Datensignale zu den 71 Trigger+Daten-Ports ist das Zuordnungsfenster mit »OK« zu verlassen. Dann sind mit

File > Save As > bus_test.cdc

eine cdc-Datei für den konfigurierten Logikanalysator zu speichern und mit »insert« ein Berechnungsprozess zu starten, der den Logikanalysator in die Netzliste »system.ngc« eingefügt. Das dauert einige Minuten. Dann ist in EDK mit »Generate Bitstream« und »Export Design« die Bitdatei zu erzeugen und die Hardwarebeschreibungsdateien in ein SDK-Projekt zu exportieren. In SDK ist wieder ein Board-Support-Package anzulegen, ein »Application Project« anzulegen

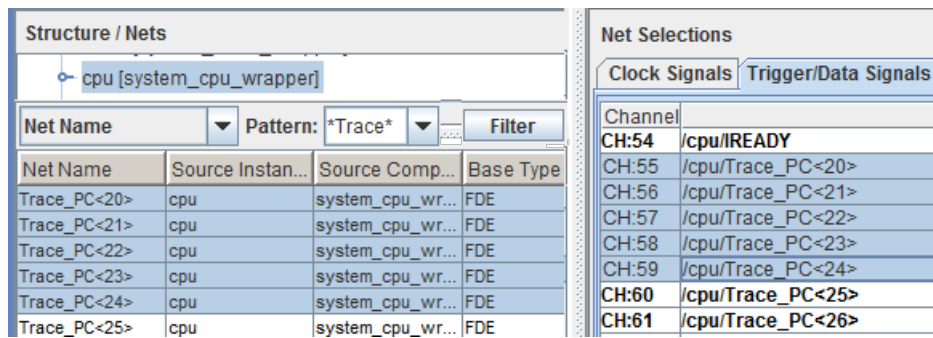


Abbildung 20: Auswahlfenster für die Ein- und Ausgabenetzliste

```
#include <xparameters.h>
#include <xio.h>
#define ioadr_led XPAR_IOMODUL_BASEADDR + 0x10
void main(){
    u8 dat=0;
    while (1){
        XIo_Out8(ioadr_led, dat);
        dat++;
    }
}
```

Befehlsfolge der innersten Schleife:

```
loop:
620: b0000004 imm 4
624: 30600010 addik r3, r0, 16 //r3:=ioadr_led
628: e0930004 lbui r4, r19, 4 //r4:= *adr_dat
62c: f0830000 sbi r4, r3, 0 //*ioadr_led:=r4
630: e0730004 lbui r3, r19, 4 //r3:=*adr_dat
634: 30630001 addik r3, r3, 1 //r3++
638: f0730004 sbi r3, r19, 4 //*adr_dat:=r3
63c: b800ffe4 bri -28 // springe nach loop
```


Abbildung 21: Testprogramm und disassemblierte Befehlsfolge der innersten Schleife

und in diesem eine Programmdatei anzulegen, das Programm einzugeben und zu übersetzen, eine »Run Configuration« zu erzeugen, das Programm zu starten und das ausführbare Programm zu disassemblieren.

Abb. 21 zeigt das Testprogramm und die disassemblierte Befehlsfolge der inneren Schleife. Die lokale Variable »dat« hat offensichtlich die Adresse

```
adr_dat := r19 +4
```

(r19 – Frame-Pointer). Das Ausgaberegister für die Leuchtdioden hat die konstante Adresse »ioadr_led = 0x40010«.

Wenn das Programm läuft, ist der »ChipScope Analyzer« zu starten, mit  die Kabelverbindung zu öffnen, das cdc-File zu importieren, sind die Led-Ausgaben zu einem Bus zusammenzufassen, die Signalvektoren mit »Reverse Bus Order« zu spiegeln, etc. Der Aufzeichnungs-Trigger soll so eingestellt werden, dass die Aufzeichnung 10 Takte vor dem Signalwechsel der Led-Ausgabe nach 0x07 beginnt. Abb. 22 zeigt die aufgezeichneten Signale.

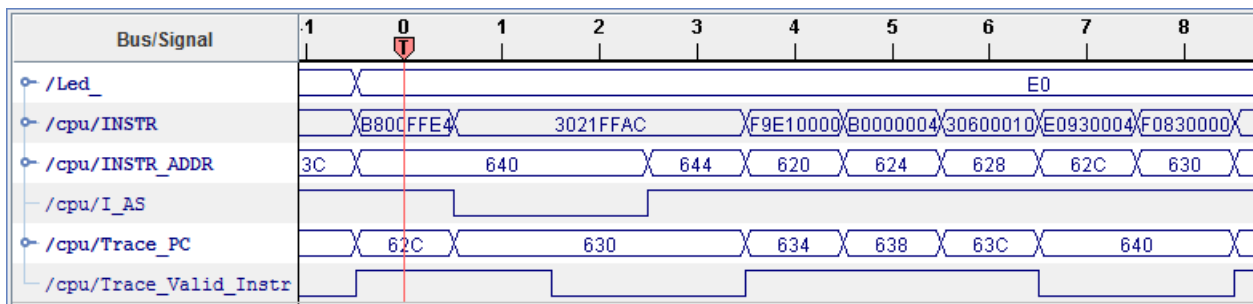


Abbildung 22: Aufgezeichnete Signale

Die ausgesendeten Befehlsadressen sind bei »I_AS=1« gültig. Die zugehörigen Befehlsdaten werden einen Takt verzögert aufgezeichnet. Die Trace-Adressen sind gegenüber den aufgezeichneten Adressen deutlich verzögert. Abb. 23 zeigt die aufgezeichnete Signalfolge ohne Befehlsdaten mit kleinerem »Zoom«. Die Zeitfenster, in denen die Adressen ungültig sind (»INSTR_ADDR« bei »I_AS=0« und »Trace_PC« bei »Trace_Valid_Instr=0«) sind rot gekennzeichnet. Es ist zu erkennen, dass vom Aussenden der Adresse an den Befehlsspeicher bis der Befehlsabschluss am Trace-Port angezeigt wird, fünf bis acht Takte vergehen. Als nächstes könnte man anhand der Trace-Signale für die Befehle, die die Pipeline anhalten (der Speicherbefehl auf Adresse 0x6C und der Sprungbefehl auf Adresse 0x63C) untersuchen, welche Pipeline-Phasen während ihrer Abarbeitung angehalten werden.

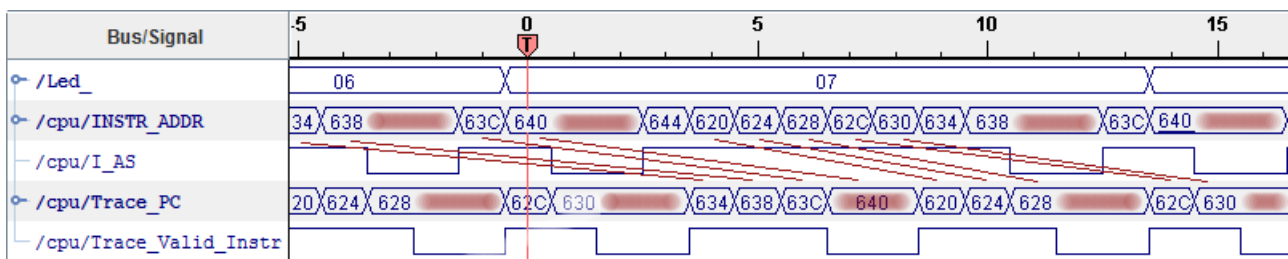


Abbildung 23: Signalaufzeichnung mit kleinerem Zoom und Kennzeichnung ungültiger Adressen

7 Test der seriellen Schnittstelle

Bei einer seriellen Kommunikation mit 9600 Baud dauert die Übertragung eines Bytes etwa 1 ms. Mit den 100MHz-Prozessortakt müsste die Datentiefe des Aufzeichnungsspeichers mindesten 10^5 betragen. Das erfordert eine übermäßig große Anzahl an Block-RAMs. Die Alternative ist die Abtastung mit einem langsameren Takt, z.B. in unserem Projekt mit dem 500 kHz Signal am Toggle-Ausgang von Festwert-Timer FIT1. Die Toggle- und Interrupt-Signale der Timer erscheinen nur in der Netzliste, wenn die Signale in der Schaltung genutzt werden⁷. In Abb. 3 ist, damit er für den Anschluss des Logikanalysators verfügbar bleibt, der IO-Modulausgang »FIT1_Toogel« mit dem Signal »T500kHz_x« und dieses mit dem Port »PORT T500kHz« verbunden.

Der Einbau des neuen Logikanalysators erfolgt wie im vorherigen Projekt:

- Project ▷ Clean all generated Files
- Geräte Netlist

⁷Die Logik für ungenutzte Ausgänge wird offenbar schon bei der Netzlistengenerierung wegoptimiert.

- ngcbuild system.ngc system1.ngc

Dann wird mit dem »ChipScope Inserter« der neue Logikanalysator konfiguriert. Im ersten Fenster des »ChipScope Insterers« bleiben die Einstellungen für »Input Design Netlist« etc. wie in Abb. 15. Im nächsten Fenster bleibt »USER1«. Im Menü für die Trigger-Parameter soll »Trigger Width« dieses mal auf 10 gesetzt werden (Abb. 24).

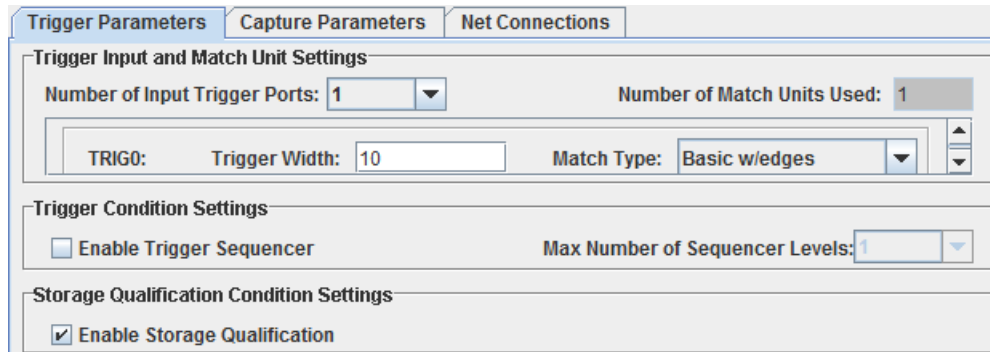


Abbildung 24: »Trigger Parameter« des Logikanalysators für den UART-Test

Im Menü für die Einstellung der »Capture Parameter« soll »Data Depth« auf 2048 vergrößert werden. Der Preis dafür sind zwei statt nur ein Block-RAM (BRAM Count 2, Abb. 25).

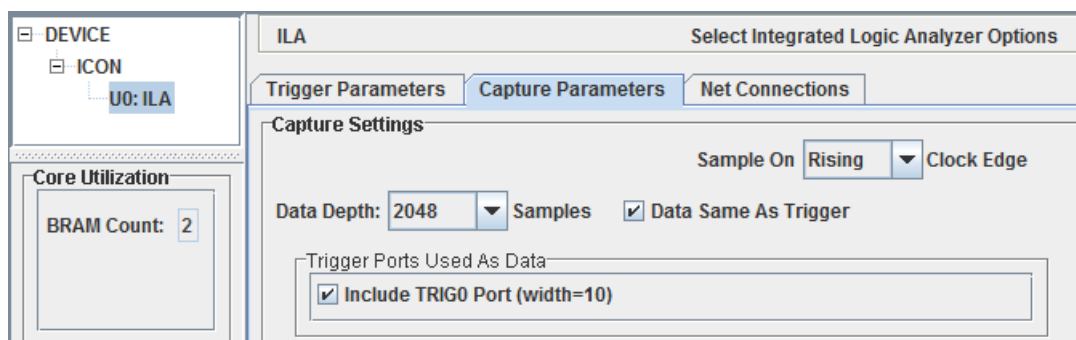


Abbildung 25: »Capture Parameter« des Logikanalysators für den UART-Test

An den Takteingang des Logikanalysators ist wie in Abb. 26 das auf einen Schaltungsausgang geführte Toggle-Signal »T200kHz_OBUF« vom Festwert-Timer FIT1 anzuschließen.

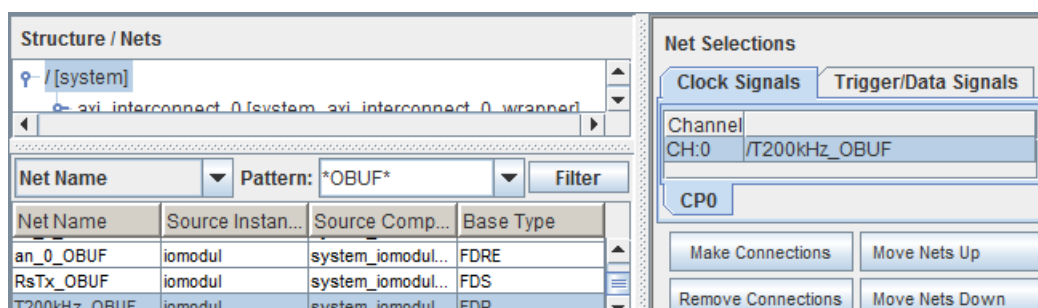


Abbildung 26: Auswahl des Aufzeichnungstakts

An die Trigger- und Dateneingänge des Logikanalysators sollen der serielle Ein- und der serielle Ausgang der UART sowie die Ausgabesignale an die Leds angeschlossen werden (Abb. 26). Nach der Abschluss der Konfiguration ist der Logikanalysator mit »Insert« einzufügen, die Schaltungsgenerierung mit »Generate Bitstream« und »Export Design« abzuschließen. In SDK

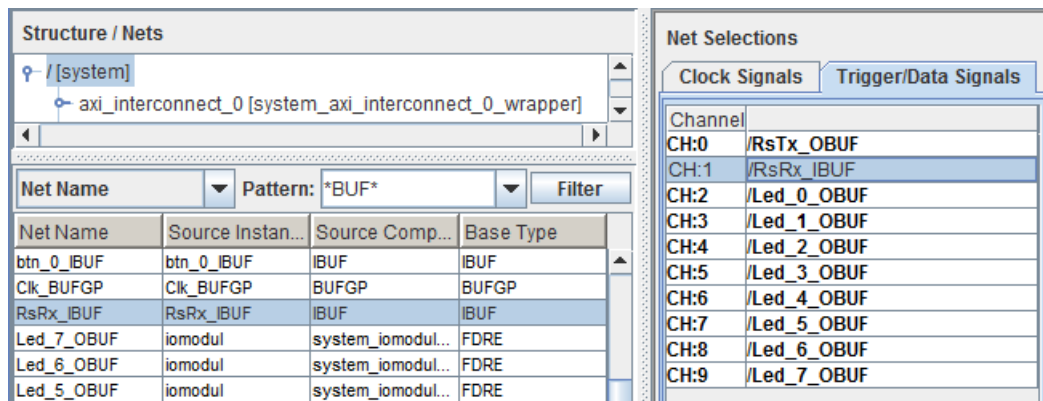


Abbildung 27: Auswahl der zu beobachtenden Signale

ist ein neues »Application Project« mit eine C-Datei mit dem Testprogramm in Abb. 28 zu erzeugen. Das Testprogramm definiert eine Sende- und eine Empfangsfunktion und wartet in einer Endlosschleife immer auf ein Zeichen, gibt den Zeichenwert auf die Led's aus und sendet das Zeichen zurück.

```
#include <xio.h>
#include <xparameters.h>
// Registeradressen siehe [pg052-iomodule[1].pdf, S. 13, Tab. 2-5]
#define ioadr_UART_RX XPAR_IOMODULE_0_BASEADDR + 0
#define ioadr_UART_TX XPAR_IOMODULE_0_BASEADDR + 0x04
#define ioadr_UART_STATUS XPAR_IOMODULE_0_BASEADDR + 0x08
#define ioadr_led XPAR_IOMODULE_0_BASEADDR + 0x10

// Empfangsfunktion für ein einzelnen Zeichen u8 getc(){
// warte bis "Rx Valid Data" (UART_STATUS.0) gesetzt ist
while (!(XIo_In8(ioadr_UART_STATUS) & (1<<0))){};
return XIo_In8(ioadr_UART_RX);
}

// Sendefunktion für ein einzelnes Zeichen void sendc(u8 c){ // warte solange "Tx Used" (
XIo_Out8(ioadr_UART_TX, c);
}

void main(){
    u8 c;
    while (1){
        c = getc(); // Warte auf Empfangszeichen
        XIo_Out8(ioadr_led, c); // Ausgabe auf die Led's
        sendc(c); // Zurücksenden des Zeichens
    }
}
```

Abbildung 28: Testprogramm für den UART-Test

Nach erfolgreicher Programmübersetzung sind

- der FPGA neu zu programmieren,
- ein »Run Configuration« ist anzulegen,
- das Programm mit »Run« zu starten,

- HTerm zu starten und
- zum Test ein Zeichen an den Softprozessor zu senden.

Nach dem Programmstart und dem initialen Programmtest mit HTerm erfolgt ein Wechsel zum »ChipScope Analyzer«. Im »ChipScope Analyzer« muss wie nach jedem Programmstart die Kabelverbindung mit »JTAG Chain ▷ Close Cable« geschlossen und neu geöffnet sowie die neue cdc-Datei, die zuvor mit dem »ChipScope Inserter« abzuspeichern ist, importiert werden. Die Triggereinstellung sei wie in Abb. 29 eine null auf dem Empfangssignal »RsRx_IBUF«.

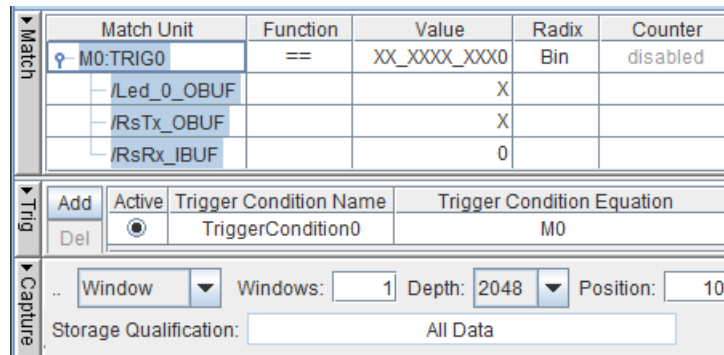


Abbildung 29: Triggereinstellungen

Nach dem Aufzeichnungsstart und dem Versenden des Zeichens »a« wird die Signalfolge in Abb. 30 aufgezeichnet. Die Angaben auf der Zeitachse sind bei einem Aufzeichnungstakt von 500 kHz mit 2 µs zu multiplizieren. Die Übertragung eines Bytes dauert, wie vorhergesagt, etwa 1 ms.

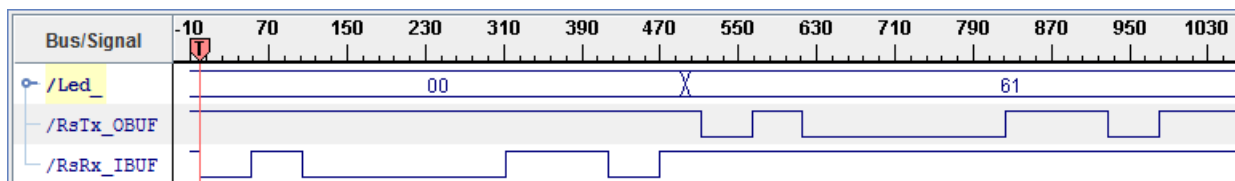


Abbildung 30: Aufgezeichnete Signalverläufe für den UART-Test