Schritt-für-Schritt-Anleitung zur Nutzung von Interrupts

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

22. April 2016

Zusammenfassung

Das Rechnersystem auf der vorherigen Anleitung wird Hardware-mäßig so erweiter werden, dass der serielle Datenempfang, der Überlauf von Timer 3 und die Beträtigung einer Taste auf der Versuchsbaugruppe Interrupts auslösen.

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

In denbeiden vergangenen Versuchsanleitungen wurde das Rechnersystem aus Abb. # konfiguriert und untersucht. Es besteht aus dem Prozessor, an den über lokale Speicherbusse (LMB's) und LMB-Controller ein Blockspeicher als Adress- und Datenspeicher angeschlossen ist, Debug-Modul etc.

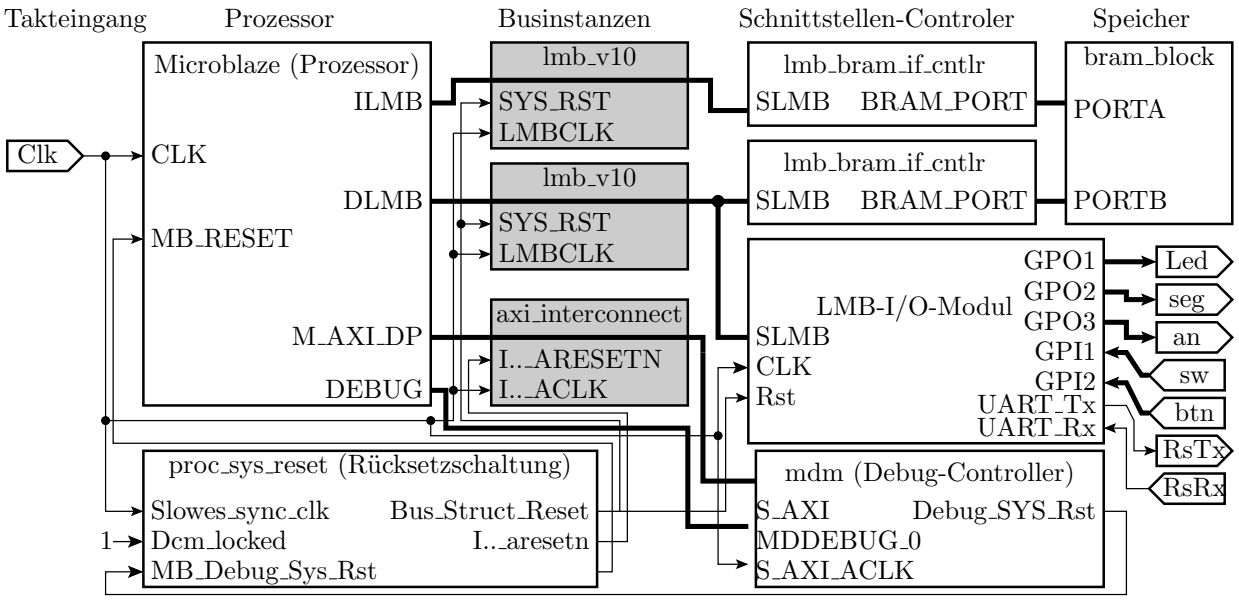
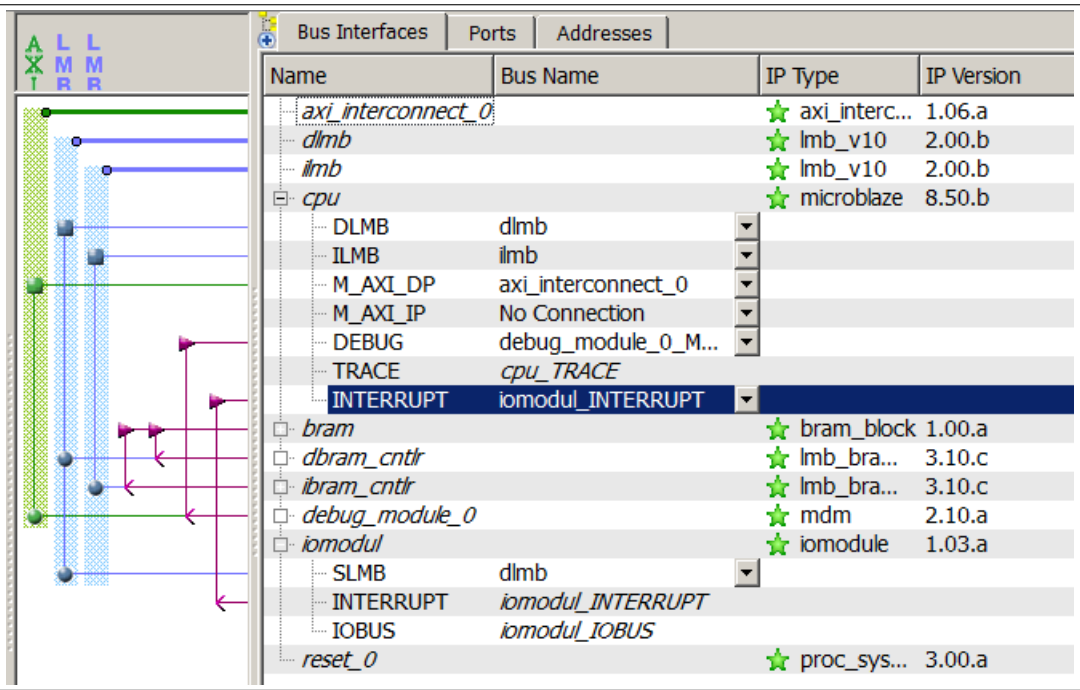


Abbildung 1: Rechnersystem ohne integrierte Logikanalysatoren

Das IO-Modul hat bereits einen eingebauten Interrupt-Controller, der noch fertig konfiguriert und dessen Interrupt-Ausgang an den Prozessor angeschlossen werden muss. Es ist wieder ein neues Projekt anzulegen. Das schrittweise Vorgehen ist in den Anleitung zuvor beschrieben. Das zu importierende mhs- und das in das Projekt zu kopierende ucf-File sind die des vorherigen Projekts ohne Logikanalysator. An der Hardware ist der Interruptausgang des IO-Moduls mit dem Interrupteingang des Prozessors zu verbinden. Weiterhin sind unter »Config IP..« für das IO-Modul

- unter dem Reiter »UART« die Hakten bei »Implement Receive Interrupt« und »Implement Transmit Interrupt« zu setzen.

Algorithm 1 x



- unter dem Reiter »GPI« unter »GPI2« für »Generate Interrupts« »Rising Edge« auszuwählen

Das mhs-File auf der Web-Seite enthält bereits diese Änderungen.

- mhs- u ucf aus dem LMB_LA oder von der Web-Seite
- interuptVerbindung.png

2 Logikanlysator

Zur Beobachtung der Funktionsweise der Ereignisbehandlung ist ein integrierter Logikanalysator einzufügen. Kontrolle Der Logikanalysator ist mit den »ChipScope Inserter« einzufügen. Das Vorgehen ist dasselbe wie in Anleitung #, Abschnitt #. Abweichend ist nur die Trigger und Aufzeichnungsbreite »Trigger Width« 41 und für »Net Connections die in Abb. 2 dargestellten Signalzuordnungen zu wählen. Wie in den Versuchen zuvor sollen die Leuchdiodenausgabe »Led_x_OBUF«, die auf den LMB ausgegebene Befehlsadressbits »/cpu/INSTR_ADDR<20..31>, deren Gültigkeitssignal »/cpu/I_AS« und die Trace-Port ausgegebene Befehlsadresse und das Gültigkeitssignal für die fertig gestellten Befehle aufgezeichnet werden. Das Signal ».._Interupt« ist das globale Interrupt-Signale vom IO-Modul zum Prozessor. »../cier..> sind Bits des Interruptfreigaberegisters und »cipr..« Bits des Interrupt-Statusregisters. Bit 2 ist jeweils für den Empfangsinterrupt der seriellen Schnittstelle, Bit 9 für den Interrupt durch Festwert-Timer FIT3 und Bit 12 für durch Tasterbetätigung ausgelöste Interrupts.

3 Testprogramm Polling

Ein exterenes Gerät signalisiert sine Anforderungen immer durch Setzen eines Ereignisbits. Im IO-Modul sind das die Statusbits im Interuppt-Stusregister. Ein Rechnerprogramm kann auf exterene Ereignisse auf zwei Arten reagieren:

- Polling (periodische Abfrage der Ereignisbits auf die zu reagieren ist oder

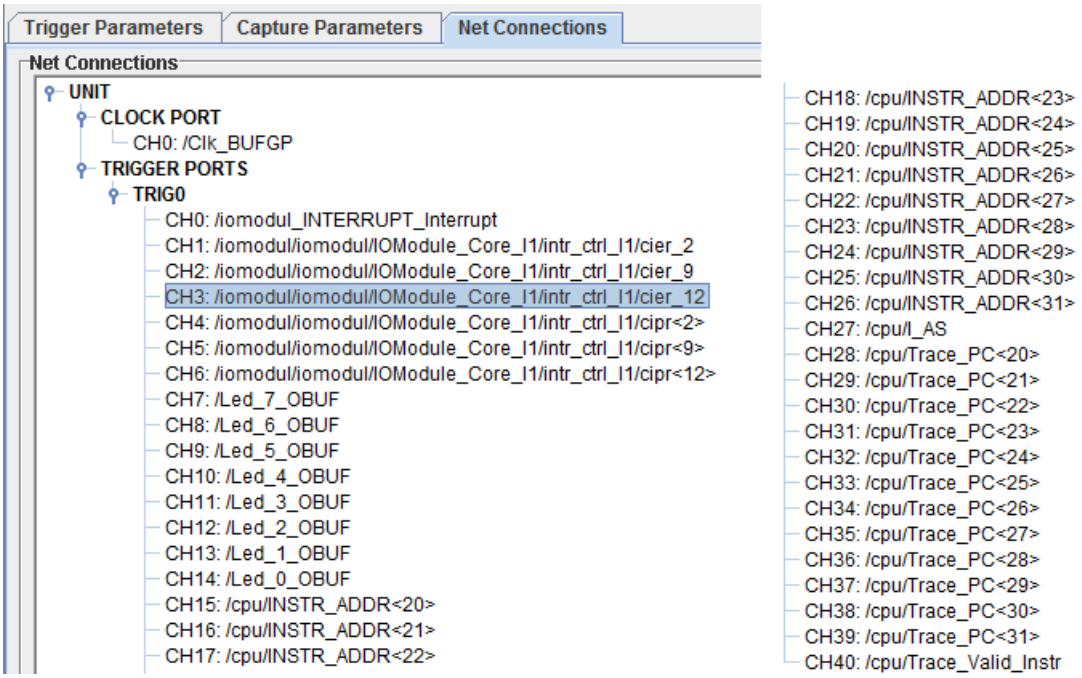
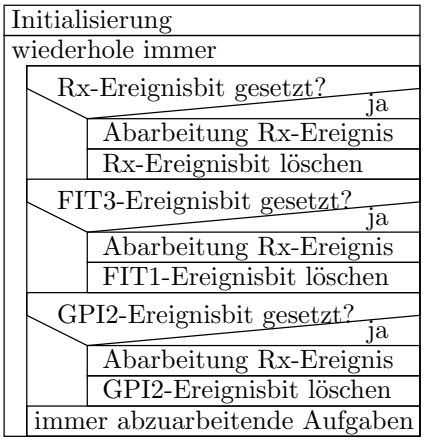


Abbildung 2: NetConnAn den Logikanalysator anzuschließende Signale

Algorithm 2 Polling



- Interrupt (automatische Programmunterbrechung, wenn das Ereignisbit gesetzt wird und Einfügen einer Interrupt-Serviceroutine).

Bei einem Polling ist das Programm als Endlosschleife zu schreiben, in dem zyklisch jedes Ereignisbit abgefragt wird. Wenn es gesetzt ist, wird eine Befehlssequenz, meist ein Unterprogramm zur Reaktion auf das Ereignis eingeschoben und das Ereignisbit gelöscht (Abb. 2).

Im nachfolgenden Programm wird in der Endlosschleife das Statuswort gelesen, nacheinander für jedes der drei überprüften Ereignisbits – »btnr_UART« für den Empfang über die serielle Schnittstelle, ...

```
#include <xio.h>
#define IOMODUL_BASEADDR 0x40000
#define io_adr_led IOMODUL_BASEADDR + 0x10 // Led-Ausgabe #define io_adr_IRQ_STATUS
#define io_adr_IRQ_ACK IOMODUL_BASEADDR + 0x3C
#define btnr_UART_RX 2
#define btnr_FIT3 9
#define btnr_GIO2 12
```

```
int main(){
u32 IRQ_Status;
u8 ct_RX_Int=0, ct_FIT3_Int=0, ct_GIO2_Int=0, w;
while(1){
IRQ_Status = XIo_In32(io_adr_IRQ_STATUS);
if (IRQ_Status & (1<<btnr_UART_RX)){
ct_RX_Int = (ct_RX_Int +1) % 4;
XIo_Out32(io_adr_IRQ_ACK, 1<<btnr_UART_RX);
}  if (IRQ_Status & (1<<btnr_FIT3)){  ct_FIT3_Int=(ct_FIT3_Int+1) % 4;
XIo_Out32(io_adr_IRQ_ACK, 1<<btnr_FIT3);
}  if (IRQ_Status & (1<<btnr_GIO2)){
ct_GIO2_Int=(ct_GIO2_Int+1) % 4;  XIo_Out32(io_adr_IRQ_ACK, 1<<btnr_GIO2);
}
w = ct_RX_Int | ct_FIT3_Int<<3 | ct_GIO2_Int<<6;  XIo_Out8(io_adr_led, w);
}
}
```

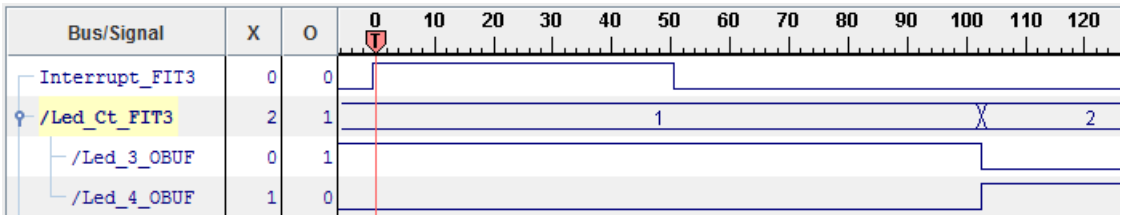
4 Interrupt

Das Beispielprogramm benötigt zusätzlich die Funktion »microblaze_enable_interrupts()« aus »mb_interface.h«, die das Interruptbit im Statusregister des Prozessors anschaltet.

```
#include <xio.h>
#include <mb_interface.h>
#define IOMODUL_BASEADDR 0x40000
#define io_adr_led IOMODUL_BASEADDR + 0x10 // Led-Ausgabe #define io_adr_IRQ_STATUS
#define io_adr_IRQ_ENABLE IOMODUL_BASEADDR + 0x38
#define io_adr_IRQ_ACK IOMODUL_BASEADDR + 0x3C #define btnr_UART_RX 2 #define btnr_
#define btnr_GIO2 12
u8 ct_RX_Int=0, ct_FIT3_Int=0, ct_GIO2_Int=0;
void myISR(void) __attribute__((interrupt_handler));
void myISR(void){
u32 IRQ_Status = XIo_In32(io_adr_IRQ_STATUS);
if (IRQ_Status & (1<<btnr_UART_RX)){
ct_RX_Int = (ct_RX_Int +1) & 0b11;
}
if (IRQ_Status & (1<<btnr_FIT3)){
ct_FIT3_Int=(ct_FIT3_Int+1) & 0b11;
}
if (IRQ_Status & (1<<btnr_GIO2)){
ct_GIO2_Int=(ct_GIO2_Int+1) & 0b11;
}
XIo_Out32(io_adr_IRQ_ACK, IRQ_Status);
}
int main(){
u8 w;
u32 EnableMask = 1<<btnr_UART_RX | 1<<btnr_FIT3 | 1<<btnr_GIO2;
XIo_Out32(io_adr_IRQ_ENABLE, EnableMask);
microblaze_enable_interrupts();
while(1){
```

```
w = ct_RX_Int | ct_FIT3_Int<<3 | ct_GIO2_Int<<6;  XIo_Out8(io_adr_led, w);
}
}
---
000006f4 <main>:
 6f4: 3021ffe4  addik r1, r1, -28
 6f8: f9e10000  swi r15, r1, 0
 6fc: 30601204  addik r3, r0, 4612
 700: b0000004  imm 4
 704: f8600038  swi r3, r0, 56
 708: b000ffff  imm -1
 70c: b9f4fb64  brlid r15, -1180
// 270 <microblaze_enable_interrupts>
 710: 80000000  or r0, r0, r0
714: b0000004  imm 4
 718: 30a00010  addik r5, r0, 16
 71c: b0000000  imm 0
 720: e08008e1  lbui r4, r0, 2273 // 8e1 <ct_FIT3_Int>
 724: 60840008  muli r4, r4, 8
 728: b0000000  imm 0
 72c: e06008e0  lbui r3, r0, 2272 // 8e0 <ct_GIO2_Int>
 730: 60630040  muli r3, r3, 64
 734: 80641800  or r3, r4, r3
 738: b0000000  imm 0
 73c: e08008e2  lbui r4, r0, 2274 // 8e2 <ct_RX_Int>
 740: 80632000  or r3, r3, r4
 744: a46300ff  andi r3, r3, 255
 748: f0650000  sbi r3, r5, 0
74c: b800ffd0  bri -48 // 71c
-----
00000010 <_vector_interrupt>:
  10: b0000000  imm 0
  14: b8080630  brai 1584 // 630 <myISR>
-----
00000630 <myISR>:
 630: 3021ffe4  addik r1, r1, -28
634: f9e10000  swi r15, r1, 0
 638: f8610008  swi r3, r1, 8
 63c: f881000c  swi r4, r1, 12
 640: f9610010  swi r11, r1, 16
 644: fa210014  swi r17, r1, 20
 648: fa410018  swi r18, r1, 24
64c: 95608001  mfs r11, rmsr
 650: f9610004  swi r11, r1, 4
 654: b0000004  imm 4
658: e8600030  lwi r3, r0, 48
 65c: a4830004  andi r4, r3, 4
 660: be040024  beqid r4, 36 // 684  664: a4830200  andi
r4, r3, 512
668: b0000000  imm 0
```

Algorithm 3 Interrupt




```
 66c: e08008e2  lbui r4, r0, 2274 // 8e2 <ct_RX_Int>  670:
30840001  addik r4, r4, 1
 674: a4840003  andi r4, r4, 3
678: b0000000  imm 0
 67c: f08008e2  sbi r4, r0, 2274 // 8e2 <ct_RX_Int>  680:
a4830200  andi r4, r3, 512
 684: be040024  beqid r4, 36 // 6a8
688: a4831000  andi r4, r3, 4096
 68c: b0000000  imm 0
 690: e08008e1  lbui r4, r0, 2273 // 8e1 <ct_FIT3_Int>
 694: 30840001  addik r4, r4, 1
698: a4840003  andi r4, r4, 3
 69c: b0000000  imm 0
 6a0: f08008e1  sbi r4, r0, 2273 // 8e1 <ct_FIT3_Int>
 6a4: a4831000  andi r4, r3, 4096
6a8: bc04001c  beqi r4, 28 // 6c4
6ac: b0000000  imm 0
 6b0: e08008e0  lbui r4, r0, 2272 // 8e0 <ct_GIO2_Int>
 6b4: 30840001  addik r4, r4, 1
 6b8: a4840003  andi r4, r4, 3
 6bc: b0000000  imm 0
 6c0: f08008e0  sbi r4, r0, 2272 // 8e0 <ct_GIO2_Int>
 6c4: b0000004  imm 4
 6c8: f860003c  swi r3, r0, 60
 6cc: e9e10000  lwi r15, r1, 0
 6d0: e9610004  lwi r11, r1, 4
 6d4: 940bc001  mts rmsr, r11
 6d8: e8610008  lwi r3, r1, 8
 6dc: e881000c  lwi r4, r1, 12
 6e0: e9610010  lwi r11, r1, 16
 6e4: ea210014  lwi r17, r1, 20
 6e8: ea410018  lwi r18, r1, 24
 6ec: b62e0000  rtid r14, 0
 6f0: 3021001c  addik r1, r1, 28
```

Triggereinstellung ».._Interupt == 1«, alle anderen Eingabebits beliebig.

»Interrupt_FIT3« ist »../intr_ctrl/cipr<9>« Von der Interrupt-Annahme bis zur Deaktivierung des Interruptbits dauert es etwa 50 Takte. Danach vergehen auch nocheinmal ca. 50 Takte, bis das Hauptprogramm die Led-Ausgabe aktualisiert.

Algorithm 4 Signalverläufe