Anleitung 4: Systemsentwurf mit dem LMB-IO-Modul

14. November 2014

Zusammenfassung

Der bisher vom Wizzard autmatisch eingebauet AXI-Bus ist für Einprozessorsysteme mit einfachen EA-Schnittstellen überdimensioniert und langsam. In dieser Anleitung wird als Alternative das einfachere und schnellere LMB-IO-Modul verwendet, dass zusätzlich mit Timern, einer seriellen Schnittstelle und Interrupt-Controller konfiguriert werden kann. Im Software-Teil folgen Testbeispiele für die Nutzung der Festwert-Timer, der programmierbaren Timer, die zusätzliche UART und die Ansteuerung der 7-Segment-Anzeige.

1 Konfiguration der Hardware

Das zu entwerfende System in Abb. 1 gleicht im Wesentlichen dem dem Minimalsystem aus der ersten Anleitung. Nur ist das IO-Modul an den lokalen Datenspeicherbus, statt an den AXI-Bus angeschlossen und enthält alle IO-Schnittstellen für die Taster, Schalter, Leuchtdioden und die 7-Segment-Anzeige sowie mehrere Timer und eine UART. Die Ausgänge »seg« und »an« steuern die 7-Segmentanzeige an. »sw« ist der Bus für die 8 Eingabeschalter, »btn« der für die 5 Taster, »RsTx« das serielle Sendesignal und »RsTx« das serielle Empfangssignal. In der nächsten Schritt-für-Schritt-Anleitung werden an dieses System integrierte Logikanalysatoren angeschlossen, um die Signalverläufe bei der Programmabarbeitung genauer zu untersuchen.

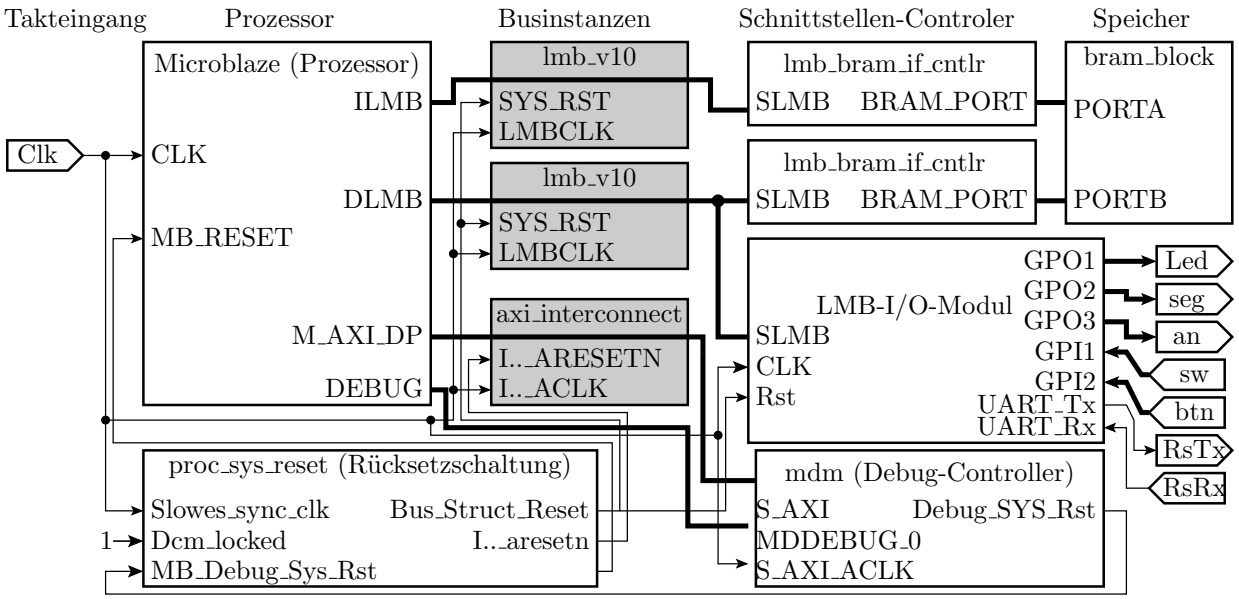


Abbildung 1: Das zu entwerfende Rechnersystem

1.1 Erstellung der Konfiguration

Für ein System mit LMB-IO-Modul gibt es keinen Wizzard. Dieser Abschnitt beschreibt deshalb Schritt für Schritt die eigenständige Zusammenstellung des Rechnersystems aus Bausteinen des IP-Katalogs. Zuerst ist ein leeres Projekt anzulegen:

- »Xilinx Plattform Studio« starten
- »Create New Blank Project«

Die Einstellungen für den FPGA (Abb. 2) sind dieselben, wie in der bisherigen Projekten. Von den Kästchen unter »Advanced Options« ist keines auszuwählen.

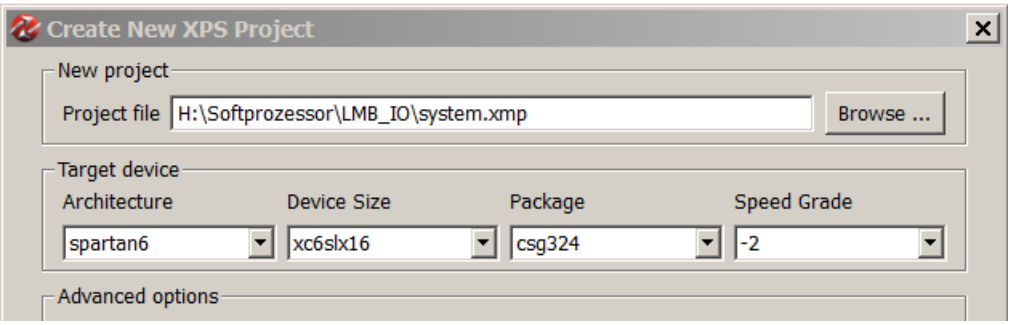


Abbildung 2: Einstellfenster für die Grunddaten beim Anlegen des Projekts

Nach Anlegen des »leeren« Projekts sind aus dem IP-Katalog folgende Cores zu übernehmen und entsprechend Abb. 3 bis 7 zu konfigurieren:

- aus »Processor« einen »MicroBlaze«,
- aus »Bus and Bridge« zwei »Local Memory Bus (LMB)« zur Ankopplung des Daten- und Befehlsbus-Controllers sowie der IO-Einheit, und einen »AXI Interconnect« für die Ankopplung des Debug-Moduls,
- aus »Memory and Memory Controller« zwei »LMB BRAM Controller« und einen Block RAM (BRAM) Block,
- aus »Debug« das »Micro Blaze Debug Module (MDM)«,
- aus »Clock, Reset and Interrupt« das »Processor System Reset Module« und
- aus »General Purpose IO« das »LMB I/O Module«.

Der Microblaze soll als »Typical« mit »Enable Debug« und ohne Caches konfiguriert werden (Abb. 3 a). Unter »Advanced« (Abb. 3 b) können Zusatzeinheiten für die Befehlssatzerweiterung ausgewählt werden:

- »Barrel Shifter« für Shift- und Rotationsbefehle um eine variable Anzahl von Bitstellen,
- »Floating Point Unit« für Gleitkomma-Befehle, ...

Für unser Beispiel soll nur der Barrel Shifter und der Festkomma-Multiplizierer ausgewählt werden. Unter »Debug« (Abb. 3 c) muß »Number of PC Breakpoints« mindesten eins bleiben. Bei »Buses« ist wie in Abb. 3 d »AXI« und »FSL« auszuwählen. Unter »Exeptions«, Caches«, »MMU«, »PVR« und »Interconnect Setting for BUSIF« bleibt alles abgewählt.

Für das Debug-Modul soll wie in Abb. 4 »Enable JTAG UART« am AXI-Bus aktiviert und in Vorbereitung des Einbaus eines integriereten Logikanalysators »BSCAN location« auf

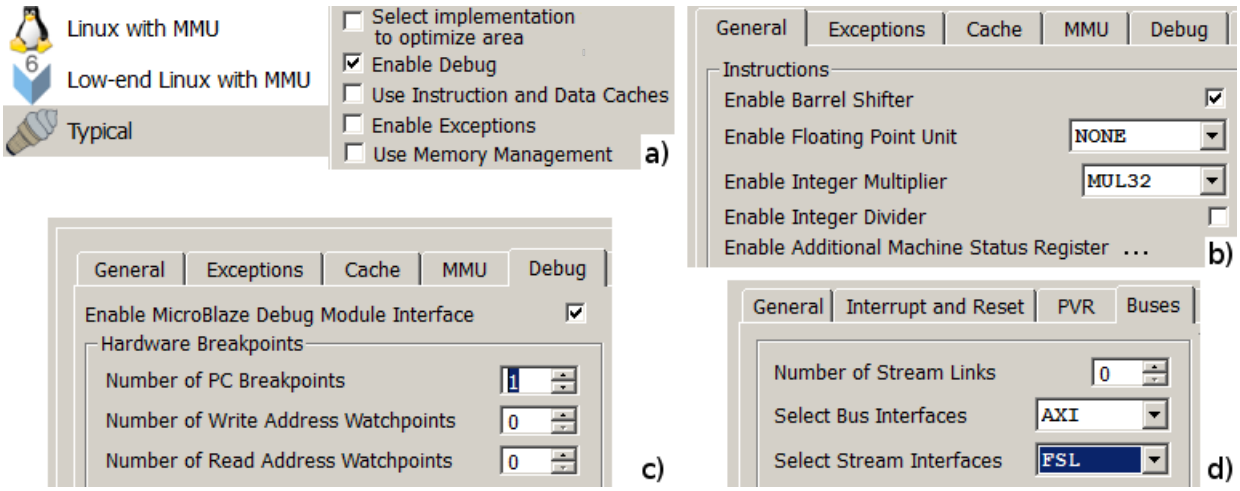


Abbildung 3: Zu wählende Konfigurationseinstellungen für den Microblaze

»ICON« umgestellt werden. Die Adresseinstellungen unter dem Reiter »System« erfolgen später mit »Generate Adresses« und unter »Interconnect Settings« bleiben alle Einstellungen auf »Bypass« bzw. »0 (None)«.

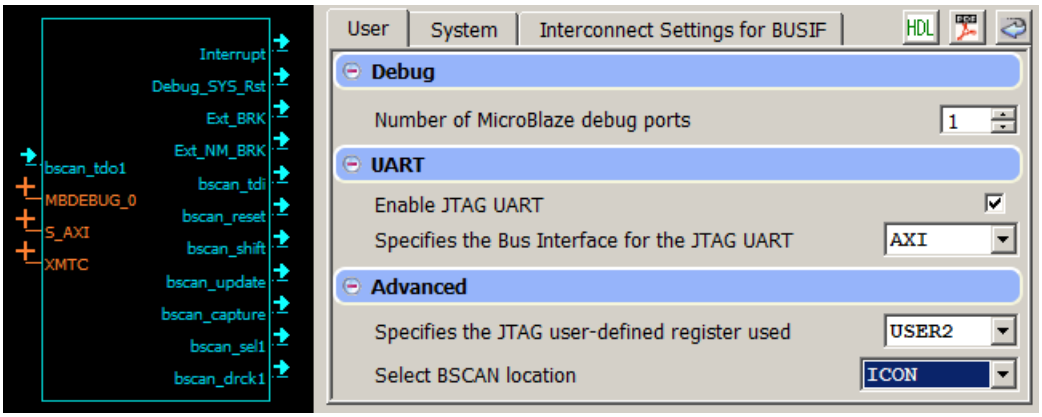


Abbildung 4: Konfigurationseinstellungen für das Debug-Modul

Als nächstes folgt die Beschreibung der Konfigurationseinstellungen für das »LMB I/O Module«. Die Adresseinstellungen unter »System« erfolgen auch hier später mit »Generate Adresses«. Unter dem nächsten Reiter »UART« sollen Sender und Empfänger mit 9600 Baud, 8 Datenbits, einem Stoppbit und ohne Paritätsbit aktiviert werden (Abb. 5). Interrupts werden in diesem Projekt nicht verwendet und sind in den nicht gezeigten Auswahlfeldern abzuwählen.

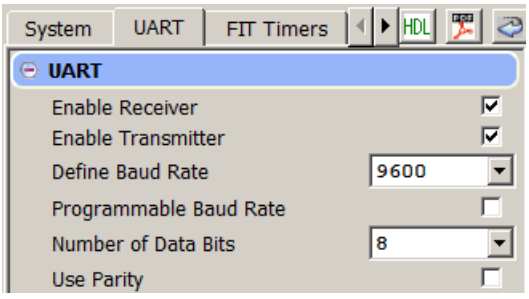


Abbildung 5: Konfigurationseinstellungen für die UART des IO-Moduls

Unter dem Reiter »FIT Timer« lassen sich maximal 4 Festwert-Timer konfigurieren (Abb. 6 a). Drei davon sollen genutzt werden:

- FIT Timer 1 zur Erzeugung eines Ereignisses alle 1 µs und als Vorteiler,
- FIT Timer 2 zur Erzeugung eines Ereignisses alle 100 µs und
- FIT Timer 3 zur Erzeugung eines Ereignisses alle 1 s.

Der Teilerfaktor »Number of Clocks ..« errechnet sich aus der Zeit zwischen den Ereignissen geteilt durch die Taktperiode von 10 ns. »Generate Interrupts« ist zu aktivieren, weil auch für eine Polling-Abfrage auf Timer-Ereignisse die Interrupt-Flags genutzt werden. Unter dem Reiter »PIT Timer« lassen sich maximal 4 programmierbare Timer konfigurieren, von denen nur der erste als 32-Bit-Timer mit FIT 1 als Vorteiler und Reload-Option genutzt werden soll. Auch die programmierbaren Teiler müssen, wenn das Programm auf Überlauf testen soll, mit »Generate Interrupt« konfiguriert werden.

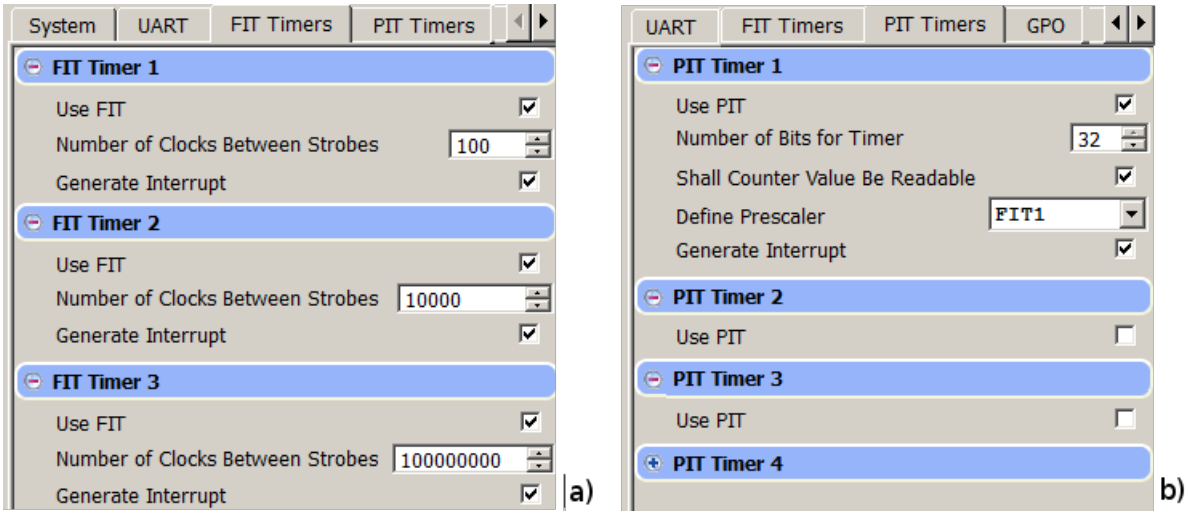


Abbildung 6: Konfiguration der Timer des IO-Moduls

Unter den Reitern »GPO« und »GPI« können bis zu vier 32-Bit-Ausgabe- bzw. Eingabeports konfoguriert werden (Abb. 7). Davon sollen genutzt werden:

- GPO 1 als 8-Bit-Ausgabeport für die Led's,
- GPO 2 als 8-Bit-Ausgabeport für die Segmenteingänge der 7-Segmentanzeige (7 Segmente plus der Punkt),
- GPO 3 als 4-Bit-Port für die Ziffernauswahl der 7-Segmentanzeige,
- GPI 1 als 8-Bit-Eingabeport für die Schalter und
- PGI 2 als 5-Bit-Eingabeport für die Taster auf der Baugruppe.

Der Ausgabewert nach der Initialisierung sei für alle Ausgänge null und die Eingänge sollen keine Interrupts erzeugen.

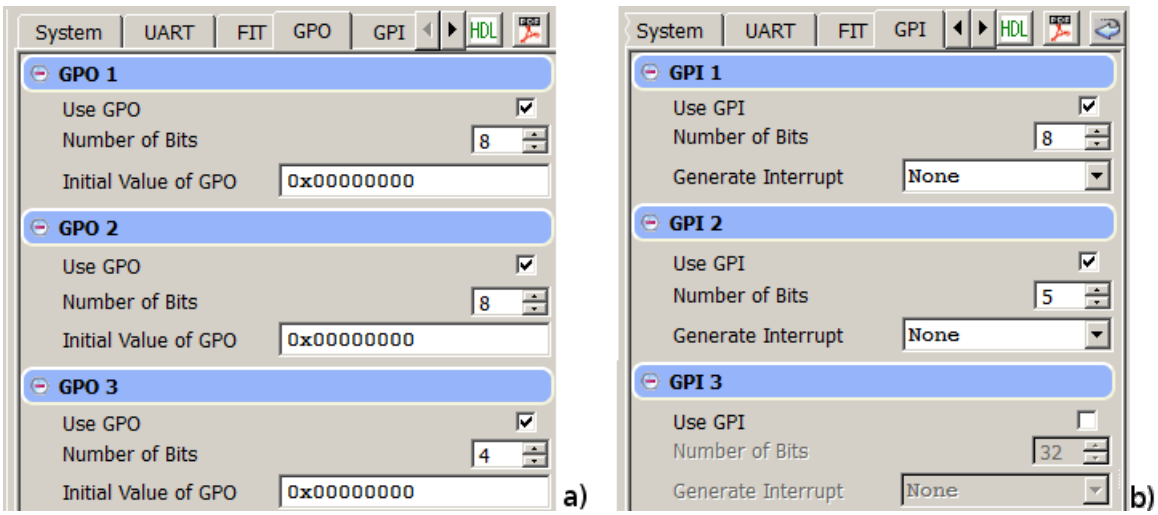


Abbildung 7: Konfiguration der parallelen Ports

In der Busverdrahtungsansicht (Abb. 8) sind an den AXI-Bus der AXI-Datenport des Prozessors und der AXI-Port des Debug-Moduls anzuschließen. Der lokale Speicherbus (LMB) für Daten verbindet den Prozessor-DLMB mit einem Speicherkontroller und dem IO-Modul. Der zweite LMB ist für Befehle und verbindet den Prozessor-ILMB mit dem zweiten Speicherkontroller. Beide Speicherkontroller sind mit Ports des Dualport-Spereicherblocks auf dem FPGA zu verbinden. Des weiteren wird der Bus zwischen Debug-Modul und Prozessor benötigt.

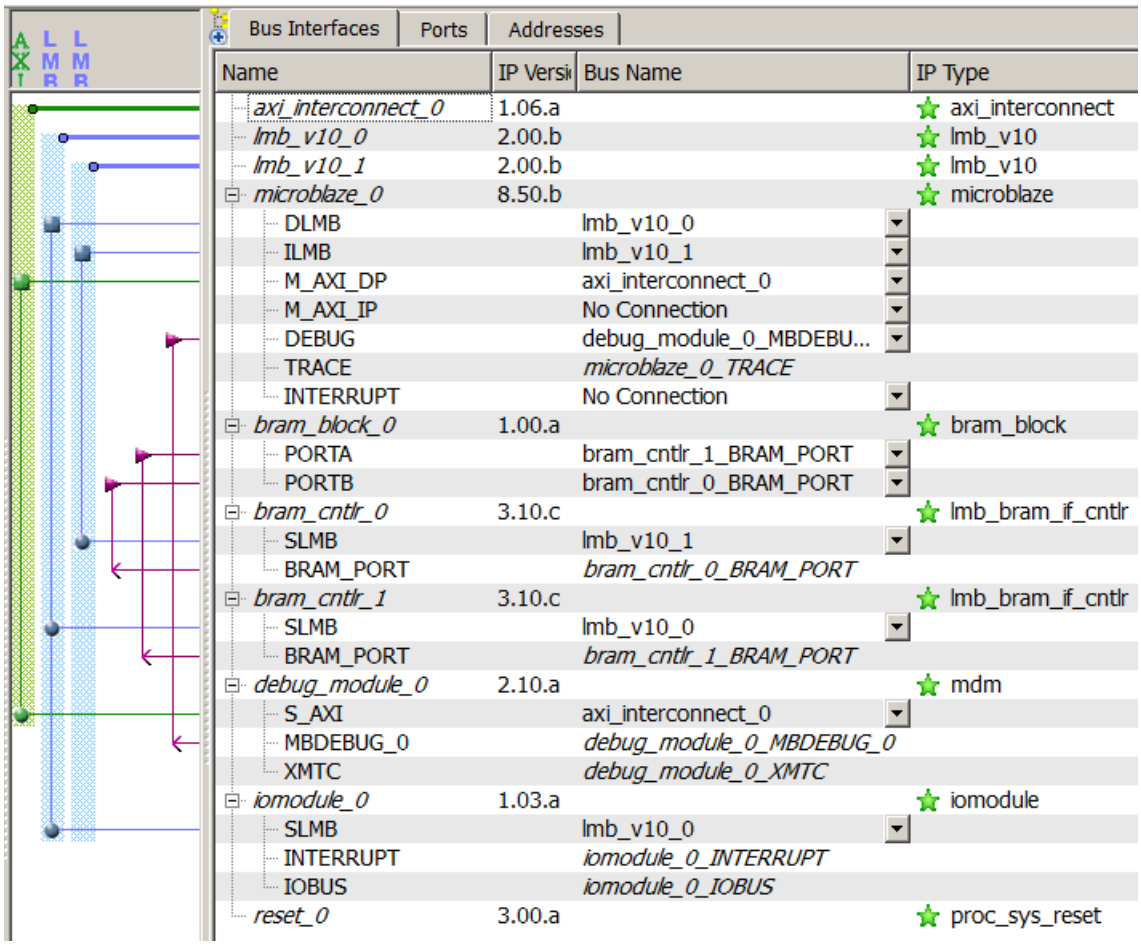


Abbildung 8: Erforderliche Busverbindungen

Die weitere Verdrahtung erfolgt unter dem Reiter »Ports« und gleicht im Wesentlichen der aus der ersten Schritt-für-Schritt-Anleitung. Abb. 9 zeigt die Nicht-Bus-Verbindungen an der Reset-Schaltung. Diese hat als Eingabetakt zur Abtastung der eingangsseitigen Reset-Signale (Slowest_sync_clk) den Eingabetakt der Baugruppe und als einziges Reset-Eingabesignal das vom Debug-Modul. Der »Dcm-locked«-Eingang, an dem zu signalisieren ist, dass der Takt stabil anliegt, ist dauerhafte »1« (net_vcc). Ausgangsseitig initialisiert »MB_Reset« den Prozessor, »Bus_Struct_Reset« über die lokalen Speicherbusse die abgeschlossenen Controller, »Peripherial Reset« das IO-Modul und das negierte Rücksetzsignal »Interconnect_aresetn« den AXI-Bus und die angeschlossenen Slaves.

Name	Connected Port	Direction	Class	IP Type
reset_0				proc_sys_reset
Slowest_sync_clk	External Ports::Clk	I	CLK	
Ext_Reset_In		I	RST	
Aux_Reset_In		I	RST	
MB_Debug_Sys_Rst	debug_module_0::Debug_SYS_Rst	I	RST	
Dcm_locked	net_vcc	I		
MB_Reset	microblaze_0::MB_RESET	O	RST	
Bus_Struct_Reset	lmb_v10_1::SYS_RST lmb_v10_0::SYS_RST	O	RST	
Peripheral_Reset	iomodule_0::Rst	O	RST	
Interconnect_aresetn	axi_interconnect_0::INTERCONNECT_ARESETN	O	RST	

Abbildung 9: Anschlusssignale an der Reset-Schaltung

An das IO-Modul wird der externe Takt und das Reset-Signal für periphere Einheiten angeschlossen. Die Interrupt- und Toggle-Ausgänge der Timer sowie der Interrupt-Eingang bleiben unbeschaltet. Das Sendesignal »UART_Tx« und das Empfangssignal UART_Rx sowie die parallelen Ein- und Ausgänge werden mit exterenen Anschlüssen verbunden (Abb. 10).

Name	Connected Port	Direction	Range	Class	IP Type
iomodule_0					iomodule
CLK	External Ports::Clk	I		CLK	
Rst	reset_0::Peripheral_Reset	I		RST	
FIT1_Interrupt		O		INTERRUPT	
FIT1_Toggle		O			
FIT2_Interrupt		O		INTERRUPT	
FIT2_Toggle		O			
FIT3_Interrupt		O		INTERRUPT	
FIT3_Toggle		O			
PIT1_Interrupt		O		INTERRUPT	
PIT1_Toggle		O			
INTC_Interrupt	L to H: No Connection	I		INTERRUPT	
(IO_IF) uart_0	Connected to External Ports				
UART_Rx	External Ports::RsRx	I			
UART_Tx	External Ports::RsTx	O			
(IO_IF) gpio_0	Connected to External Ports				
GPI1	External Ports::sw	I	[7:0]		
GPI2	External Ports::btn	I	[4:0]		
GPO1	External Ports::Led	O	[7:0]		
GPO2	External Ports::seg	O	[7:0]		
GPO3	External Ports::an	O	[3:0]		

Abbildung 10: Anschlusssignale am LMB-IO-Modul

Das Debug-Modul liefert das eingangsseitige Rücksetzsignal für die Reset-Schaltung. An die Boundry-Scan-Anschlüsse »bscan_..« wird im nächsten Abschnitt der Chipscope-Controller »ICON« angeschlossen (Abb. 11). Der Speicher »bram_block..« darüber und die BRAM-Controller »bram_cntlr..« haben außer Busverbindungen keine weiteren Anschlüsse.

Name	Connected Port	Direction	Class	IP Type
bram_block_0				bram_block
bram_cntlr_0				lmb_bram_if_cntlr
bram_cntlr_1				lmb_bram_if_cntlr
debug_module_0				mdm
Interrupt		O	INTERRUPT	
Debug_SYS_Rst	reset_0::MB_Debug_Sys_Rst	O	RST	
bscan_tdi		O		
bscan_reset		O	RST	
bscan_shift		O		
bscan_update		O		
bscan_capture		O		
bscan_sel1		O		
bscan_drck1		O	CLK	
bscan_tdo1		I		
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_0			
S_AXI_ACLK	External Ports::Clk	I	CLK	

Abbildung 11: Anschlusssignale am Debug-Modul

An den Prozessor und die Bausteine für die Modellierung der Busse wird jeweils der Takt und ein Reset-Signal von der Reset-Schaltung angeschlossen (Abb. 12).

Name	Connected Port	Direction	Range	Class	IP Type
axi_interconnect_0					axi_interconnect
INTERCONNECT_ACLK	External Ports::Clk	I		CLK	
INTERCONNECT_ARESETN	reset_0::Interconnect_aresetn	I		RST	
lmb_v10_0					lmb_v10
LMB_Clk	External Ports::Clk	I		CLK	
SYS_Rst	reset_0::Bus_Struct_Reset	I		RST	
lmb_v10_1					lmb_v10
LMB_Clk	External Ports::Clk	I		CLK	
SYS_Rst	reset_0::Bus_Struct_Reset	I		RST	
microblaze_0					microblaze
MB_RESET	reset_0::MB_Reset	I		RST	
DBG_STOP		I			
MB_Halted		O			
MB_Error		O			
WAKEUP		I	[0:1]		
SLEEP		O			
DBG_WAKEUP		O			
LOCKSTEP_MASTER_OUT		O	[0:4...		
LOCKSTEP_SLAVE_IN		I	[0:4...		
LOCKSTEP_OUT		O	[0:4...		
(BUS_IF) DLMB	Connected to BUS lmb_v10_0				
CLK	External Ports::Clk	I		CLK	
(BUS_IF) ILMB	Connected to BUS lmb_v10_1				
CLK	External Ports::Clk	I		CLK	
(BUS_IF) M_AXI_DP	Connected to BUS axi_interconnect_0				
CLK	External Ports::Clk	I		CLK	
(BUS_IF) M_AXI_IP	Connected to External Ports				
CLK	External Ports::Clk	I		CLK	

Abbildung 12: Anschlussignale am Prozessor und den Busbausteinen

Die externen Anschlüssen müssen dieselben Namen wie in der ucf-Datei haben (Abb. 13). Der Eingabetakt »Clk« wird an neun Bausteineingänge weitergeleitet. Für ihn muss in der Spalte »Class« der Eintrag »CLK« stehen und in der Spalte »Frequency« manuell die Frequenz 100.000.000 Hz (100 MHz) eingetragen werden. Die Indexbereiche der Busanschlüsse an den parallelen Schnittstellen (Led etc.) sind hier absteigend definiert. Das vermeidet die bitgespiegelte Ein- und Ausgabe in den vorherigen Projekten.

Name	Connected Port	Direction	Range	Class	Frequency(Hz)
External Ports					
Clk	reset_0::Slowest_sync_clk microblaze_0::[DLMB:ILMB:M_AXI_DP:M... lmb_v10_1::LMB_CLK lmb_v10_0::LMB_CLK iomodule_0::CLK debug_module_0::[S_AXI]::S_AXI_ACLK bram_cntlr_1::[SLMB]::LMB_Clk bram_cntlr_0::[SLMB]::LMB_Clk axi_interconnect_0::[S_AXI_CTRL]::INT...	I		CLK	100000000
Led	iomodule_0::[gpio_0]::GPO1	O	*[7:0]*	NONE	
RsRx	iomodule_0::[uart_0]::UART_Rx	I		NONE	
RsTx	iomodule_0::[uart_0]::UART_Tx	O		NONE	
an	iomodule_0::[gpio_0]::GPO3	O	*[3:0]*	NONE	
btn	iomodule_0::[gpio_0]::GPI2	I	*[4:0]*	NONE	
seg	iomodule_0::[gpio_0]::GPO2	O	*[7:0]*	NONE	
sw	iomodule_0::[gpio_0]::GPI1	I	*[7:0]*	NONE	

Abbildung 13: Verdrahtung der Reset-Schaltung

Nach der Festlegung der Busverbindungen und der übrigen Verdrahtung sind unter dem Reiter »Adresses« über die Schaltfläche »Generate Adresses« den Bus-Controllern, dem Debug-Modul und dem IO-Modul Adressbereiche zuzuordnen (Abb. 14).

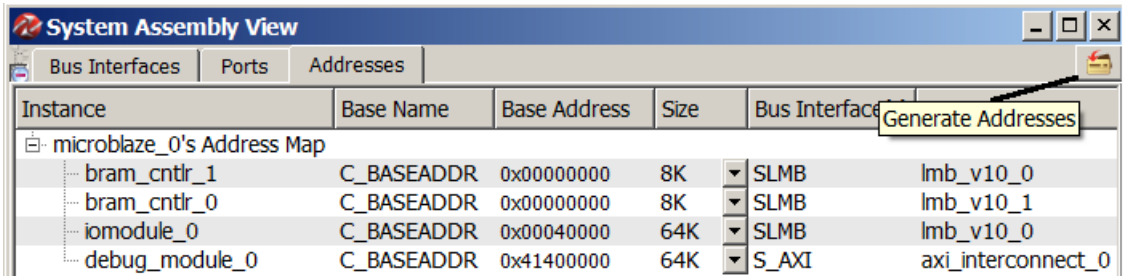


Abbildung 14: Festlegung des Pfads für das Software-Projekt

Die ucf-Datei mit den Gehäusezuordnungen für die Leuchtdioden, Schalter etc. ist die Datei »system.ucf« auf der Web-Seite in der Tabellenzeile des Projekts. Sie vor dem Übersetzen in die des Projekts zu kopieren oder in das Projekt einzubinden. Abschließend ist mit »Run DRCs« zu kontrollieren, dass keine Entwurfsregeln verletzt sind, die Bitdatei zu erzeugen und die Entwurfsdaten nach SDK zu expotieren. Der Design-Space für den Export sei ».../Softprozessor/SW_LMB_IO.

2 Testbeispiele

Die Entwicklung und der Test der Programmbeispiele umfasst wieder die Schritte

1. Anlegen des Board-Support-Packages,
2. Programmieren des FPGAs,
3. Für jedes Testbeispiel:
   (a) Anlegen des Projekts,
   (b) Erstellen und Übersetzen der C-Quelldatei,

(c) Erstellen der »Run Configuration«,

(d) Programmstart und Test.

Im Weiteren werden nur die C-Programme und die zu erwartenden Testergebnisse besprochen. Die übrigen Schritte sind aus den vorherigen Schritt-für-Schritt-Anleitungen zu übernehmen.

Die kompletten C-Programm enthalten außer den hardware-spezifischen Funktionen, die in den nachfolgenden Abschnitten besprochen werden, Testausgaben für die Konsole über die JTAG-Uart. Diese nutzen jeweils die zusätzliche Header :

```
#include <xparameters.h>
#include <xuartlite_l.h>
```

(siehe vorherige Schritt-für-Schritt-Anleitungen). Darüber hinaus ist in jedem Testbeispiel eine Funktion

```
void sendStringJTAG(char *str){
  while (*str != 0) {
    XUartLite_SendByte(XPAR_UARTLITE_0_BASEADDR, *str);
    str++;
  }
}
```

zur Ausgabe von Zeichenketten zu Testzwecken definiert. Diese Funktion erhält als Aufrufparameter einen Zeiger auf eine Zeichenkette, die auch direkt im Funktionsaufruf definiert werden kann, z.B.:

```
sendStringJTAG("Test Festwert-Timer:\r\n");
```

2.1 Test eines Festwert-Timers

Ein Festwert-Timer invertiert nach einer Zeit

$$t_\mathrm{W} = n \cdot t_\mathrm{P}$$

(t_W – Wartezeit; t_P – Taktperiode, im Beispiel 10 ns; n – in der Hardware-Konfiguration festgelegter Teilerwert) sein Toggle-Ausgang und setzt das Interrupt-Bit[1]. Die Interrupt-Bits gehören zum Interrupt-Statusregister und sind vom Programm nur lesbar. Gelöscht werden sie durch schreiben einer eins in dasselbe Bit des Interrupt-Bestätigungsregisters.

Für die Nutzung der Festwert-Timer benötigt das Testprogramm folgende Registeradressen und Bitnummern[2]:

```
#define XPAR_IOMODULE_0_BASEADDR 0x40000 // (vergl Abb. 14)
#define ioadr_led XPAR_IOMODULE_0_BASEADDR + 0x10 //für Testzwecke
#define ioadr_IRQ_STATUS XPAR_IOMODULE_0_BASEADDR + 0x30
#define ioadr_IRQ_ACK XPAR_IOMODULE_0_BASEADDR + 0x3C
#define FIT1_EventNr 7
#define FIT2_EventNr 8
#define FIT3_EventNr 9
```

[1]Das Interrupt-Bit der Timer ist auch als Ausgabesignal des LMB-IO-Moduls z.B. zum Anschluss eines integrierten Logikanalysators verfügbar.

[2]Nachzulesen in der pdf-Dokumetation des IO-Moduls, das im Konfigurationsdialog auswählbar ist.

Taster	Mitte	oben (BTNU)	links (BTNL)	unten (BTND)	rechts (BTNR)
Wartezeit t_w in s	0.25	0.5	1	2	4

Tabelle 1: Die den Tastern zugeordnete Wartezeiten

Die nachfolgende Funktion wartet auf das Ereignisbit mit der Nummer »IRQ_Nr« und löscht es:

```
#include <xio.h>
void WaitEvent(u8 IRQ_Nr){
  while(!(XIo_In32(ioadr_IRQ_STATUS) & 1<<IRQ_Nr)); // warte auf Ereignisbit
  XIo_Out32(ioadr_IRQ_ACK, 1<<IRQ_Nr);              // lösche Ereignisbit
}
```

Das nachfolge Testprogramm nutzt diese Funktion und Timer FIT 3, dessen Ereignisperiode 1 s ist, um den binären Ausgabewert auf den Led's jede Sekunde um eins zu erhöhen:

```
void main(){
 u8 ct=0;
 while(1){
  WaitEvent(FIT3_EventNr); // Warte auf FIT3-Ereignis
  XIo_Out8(ioadr_led, ct); // Ausgabe des Zählwerts auf Led
  ct++;                    // Zählwert um 1 erhöhen
 }
}
```

Das komplette Programm steht in der Datei »test_fit.c« auf der Web-Seite.

2.2 Test des programmierbaren Timers

Ein programmierbarer Timer hat auch eine Ereignisbit und zusätzlich ein Lade- und ein Steuerregister:

```
#define PIT1_EventNr 3
#define ioadr_PIT1_PRELOAD XPAR_IOMODULE_0_BASEADDR + 0x40
#define ioadr_PIT1_CONTROL XPAR_IOMODULE_0_BASEADDR + 0x48
#define ioadr_sw  XPAR_IOMODULE_0_BASEADDR + 0x20
#define ioadr_btn XPAR_IOMODULE_0_BASEADDR + 0x24
```

Die zusätzlich definierten Eingabeadressen zum Einlesen der Schalter- und Tasterwerte werden zu Testzwecken benötigt.

Das Kontrollregister hat zwei Bits. Bit 0 schaltet den Timer und Bit 1 den Auto-Reload-Modus ein. Das nachfolgende Testprogramm liest ständig die Schalter- und Tasterwerte ein. Wenn ein Taster gedrückt ist und beim Lesen zuvor keiner gedrückt war, wird, wenn der Schalter sw(2) eingeschaltet ist, das Laderegister mit dem Wert »zwei hoch Tasternummer mal 125.000« geladen. Tabelle 1 zeigt die den fünf Tastern zugeordneten Wartezeiten. Wenn Schalter sw(3) eingeschaltet ist, werden bei Tasterbetätigung (nur oder zusätzlich) die Schalterwerte von sw(0) und sw(1) in die Bits 0 und 1 des Kontrollregisters geladen. Ist das Ereignisbit nach der Zählzeit gesetzt, wird es gelöscht, der Zählwert ausgegeben und danach erhöht.

```
void main(){
 u8 btn, btn_d, sw, ct=0;
 while(1){
```

```
  btn_d= btn;
  btn  = XIo_In8(ioadr_btn);
  sw   = XIo_In8(ioadr_sw);
  // wenn eine Taste gedrück ist und vorher keiner gedrückt war
  if (btn && !btn_d){
   // wenn Schalter sw(2) eingeschaltet ist, Ladewert schreiben
   if (sw & 1<<2)
    XIo_Out32(ioadr_PIT1_PRELOAD, ((btn & 0x1E)+1)*125000);
   // wenn Schalter sw(3) eingeschaltet ist, Steuerbits schreiben
   if (sw & 1<<3)
    XIo_Out8(ioadr_PIT1_CONTROL, sw & 0b11);
  }
                                              // wenn Ereignisbit gesetzt
  if (XIo_In32(ioadr_IRQ_STATUS) & (1<<PIT1_EventNr)){
   XIo_Out32(ioadr_IRQ_ACK, 1<<PIT1_EventNr); // Ereignisbit löschen
   XIo_Out8(ioadr_led, ct);                   // Zählwert ausgeben
   ct++;                                      // Zähler erhöhen
  }
 }
}
```

Das komplette Programm steht in der Datei »test_pit.c« auf der Web-Seite. Zum Testen sollen zuerst die Schalter sw(0) bis sw(3) eingeschaltet sein. Bei jedem Tastendruck wird der Ladewert und der Steuerwert »Auto-Reload« und »Zähler ein« geschrieben. Die Leds zählen mit der dem Taster zugeordneten Verzögerung hoch. In einem nächsten Versuch ist sw(3) auszuschalten und eine Taste zu betätigen. Der Reload-Wert wird überschrieben und der Steurwert bleibt. Untersuchen Sie, wie der Timer darauf reagiert.

Untersuchen Sie mit der Schalterfolge in der nachfolgenden Tabelle und Betätigung von BTND nach jeder Schalteränderung, wie der Timer auf Änderungen der Steuerbits reagiert:

sw(0)	1	1	0	1	1	1	1
sw(1)	1	1	1	1	0	1	1
sw(2)	1	0	0	0	0	0	1
sw(3)	1	1	1	1	1	1	1

Zu beobachten sein müsste, dass mit sw(0) das Zeitzählen aus und wieder einschaltbar ist. Mit sw(1) aus wird der nächste Reload-Vorgang verhindert, so dass der Timer nur noch ein Ereignis generiert und dann stehen bleibt. Damit der Timer »wiederanläuft« muss der Reload-Wert neu geschrieben geschrieben werden.

2.3 Test der UART

Die UART des IO-Moduls wird über drei Register mit den nachfolgenden Adressen gesteuert:

```
#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
```

Wenn die UART ein Zeichen empfangen hat, setzt sie das Statusbit »Rx Valid Data« (Statusbit 0). Das Lesen des Zeichens aus dem Eingaberegister, das die Adresse »ioadr_UART_RX« hat, löscht das Bit wieder. Die nachfolgende Funktion wartet auf ein Zeichen und gibt den Zeichenwert zurück:

```
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);
}
```

Vor dem Senden ist zu warten, bis das Statusbit »Tx Used« (Statusbit 3) nicht mehr gesetzt ist. Dann wird der Zeichenwert in das Senderegister, das die Adresse »ioadr_UART_TX« hat, geschrieben. Dabei wird »Tx Used« erneut gesetzt. Die nachfolgende Funktion versendet das ihr beim Aufruf übergebene Zeichen:

```
void sendc(u8 c){
  // warte solange "Tx Used" (UART_STATUS.3) gesetzt ist
  while (XIo_In8(ioadr_UART_STATUS) & (1<<3)){};
  XIo_Out8(ioadr_UART_TX, c);
}
```

Das nachfolgende Programm warte in einer Endlosschleife auf den Empfang eines Zeichens, gibt den Bytewert auf die Led's aus und schickt das Zeichen zurück:

```
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
 }
}
```

Das komplette Programm ist »test_uart.c« auf der Web-Seite. Zum Ausprobieren sind:

- der UART-USB-Anschluss der Versuchsbaugruppe mit dem PC zu verbinden,
- auf dem PC das Programm HTerm zu starten,
- im HTerm der richtige Port[3] und die Verbindungsparameter 9600 Baud, 8 Daten-, 1 Stopp- und kein Paritätsbit einzustellen und
- nach Start des zu testenden Programms auf dem Softprozessor Zeichen zu senden.

2.4 Test der 7-Segment-Anzeige

Die Ansteuerung einer 7-Segment-Anzeige verlanget einen zeitgesteuerten zyklischen Ablauf, in dem immer für eine feste Zeitdauer von $t_A \approx 1\,\mu s \ldots 1\,ms$ eines der vier Anoden-Signale aktiviert und gleichzeitig der Segmentcode für diese Ziffer ausgegeben wird. Der Anoden-Wert für »Ziffer an« ist dabei null und ein Segment leuchtet, wenn das zugehörige Segmentausgabebit null ist.

Die Ausgaberegister für die 7-Segment-Anzeige haben die nachfolgenden Adressen:

```
#define ioadr_seg XPAR_IOMODULE_0_BASEADDR + 0x14
#define ioadr_an  XPAR_IOMODULE_0_BASEADDR + 0x18
```

[3] Als Port ist der auszuwählen, der neu erscheint, wenn das USB-Kabel eingesteckt wird.

Ferner nutzt das nachfolgende Testprogramm das Empfangs- und das Statusregister der UART, das Interrupt- und Interrupt-Status-Register für Timer-Ereignisse und hier speziell den Timer FIT 2:

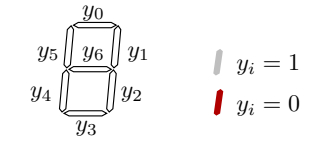
```
#define ioadr_UART_RX     XPAR_IOMODULE_0_BASEADDR + 0
#define ioadr_UART_STATUS XPAR_IOMODULE_0_BASEADDR + 0x08
#define ioadr_IRQ_STATUS  XPAR_IOMODULE_0_BASEADDR + 0x30
#define ioadr_IRQ_ACK     XPAR_IOMODULE_0_BASEADDR + 0x3C
#define FIT2_EventNr 8
```

Das nachfolgende Testprogramm wartet in einer Enlosschleife ständig auf Timer-Ereignisse und darauf, dass die UART ein Zeichen empfängt. Bei jedem Timer-Ereignis wird eine der vier Hex-Ziffern einer vorzeichenfreien 16-Bit-Zahl für eine Zeit t_w angezeigt und der Ziffenzähler modulo-4 erhöht. Bei einem Byteempfang wird der anzuzeigende Zahlenwert mit 2^8 multipliziert und der empfangene Bytewert addiert.

```
void main(){
 u8 i=0; u16 dat=0;
 while(1){
  // wenn ein Timer-Ereignis
  if (XIo_In32(ioadr_IRQ_STATUS) & 1<<FIT2_EventNr){
   XIo_Out32(ioadr_IRQ_ACK, 1<<FIT2_EventNr);  // Lösche Ereignisbit
   XIo_Out8(ioadr_an, ~(1<<i)); // Auswahl der Ziffer
   XIo_Out8(ioadr_seg, conf_seg7(0xF&(dat>>(4*i))));
   i = (i+1) & 0b11;
  }
  // wenn ein Zeichen empfangen wird
  if (XIo_In8(ioadr_UART_STATUS) & (1<<0)){
   dat = dat<<8 + XIo_In8(ioadr_UART_RX);
  }
 }
}
```

Die Ausgabewerte für die Segmente werden dabei aus dem 4-Bit-Hex-Wert der Ziffer mit dem nachfolgenden Unterprogramm gebildet:

```
u8 conf_seg7(u8 x){
 switch (x){
  case  0: return 0b11000000;
  case  1: return 0b11111001;
  case  2: return 0b10100100;
  case  3: return 0b10110000;
  case  4: return 0b10011001;
  case  5: return 0b10010010;
  case  6: return 0b10000010;
  case  7: return 0b11111000;
  case  8: return 0b10000000;
  case  9: return 0b10010000;
  case 10: return 0b10001000; //A
  case 11: return 0b10000011; //B
  case 12: return 0b11000110; //C
  case 13: return 0b10100001; //D
```



x	0000	0001	0010	0011
y	0	1	2	3
x	0100	0101	0110	0111
y	4	5	6	7
x	1000	1001	1010	1011
y	8	9	A	b
x	1100	1101	1110	1111
y	C	d	E	F

```
    case 14: return 0b10000110; //E
    case 15: return 0b10001110; //F
    default: return 0b11111111; // sonst aus
   }
  }
```

Das komplette Programm steht in der Datei »test_seg7.c« auf der Web-Seite. Zum Testen ist wie für den UART-Test zusätzlich zum Programm das HTerm zu starten. Die 7-Segment-Anzeige zeigt nach dem Programmstart eine null und nach Zeichenempfang die Hexadezimalwerte der beiden letzten empfangenen Zeichen an.