Konfigurierbare eingebettete Systeme

Beuth Hochschule - Wintersemester 2018/19

Laborübung 1

Gruppe *Index*
Omid Rahimian Mashhadi Mat.Nr.: 872958
Torsten Michael Schenk Mat.Nr.: 838995

Abgabedatum: 12.11.2018

Seitenanzahl: 8

Inhaltsverzeichnis

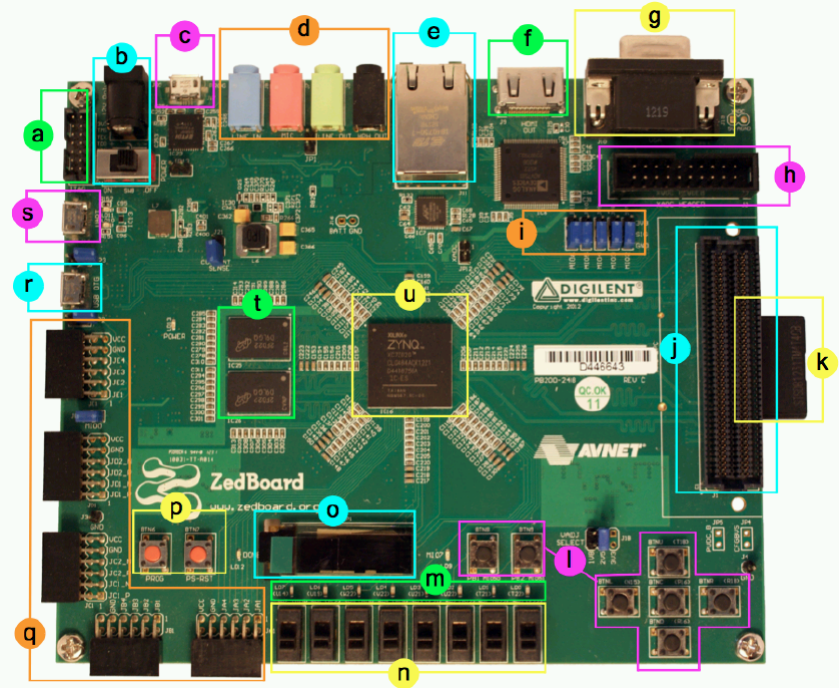
1 Vorwort

Bei der Recherche zur Bearbeitung der Übungen wurden viele englischsprachige Webseiten zu rate gezogen. Generell kann man sagen, dass englische Fachbegriffe sich im Bereich FPGA und embedded Design etabliert haben, so dass eine Übersetzung eher verwirren als helfen würde. Daher haben wir uns entschieden, die **englischen** Bezeichner und Beschreibungen beizubehalten.
Um Codeabschnitte besser von Beschreibungen besser unterscheiden zu können, wurde eine eigene Schriftart verwendet:

```
Kommandozeilen Eingaben und Codesnippets werden wie HIER dargestellt.
```

2 Aufgabe 1

In der Laborübung wurde das ZedBoard Zynq-7000 eingesetzt. Es umfasst als **PL** den Artix-7 FPGA mit 85K Logic Cells (Device Z-7020, Part: XC7Z020) und als **PS** den Dual-core ARM Cortex-A9 MPCore™ mit 866 MHz.

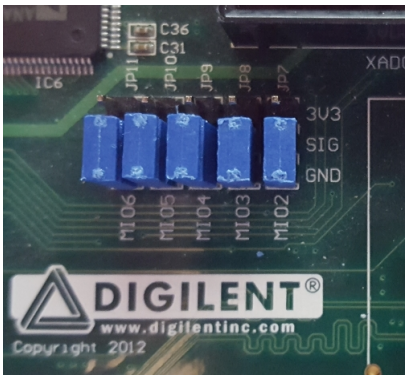


ZedBoard mit Xilinx Zynq-7000 SoC

Wichtigste Anschlüsse für Laborübung 1

- b) Netzanschluss
- c) USB-JTAG (Programmierung)
- s) USB-UART (serielle Schnittstelle)
- i) Jumper (Konfiguration eines Programmierungsanschlusses)

Jumper (i) für JTAG Programmierung wurde für Laborübung 1 verwendet. Über das JTAG Interface kann via USB progammiert werden. Die Beispielprogramme liegen jedoch auf der mitgelieferten SD-Karte.



Jumper Konfiguration für JTAG Programmierung

Jumper (i) die Programmierung von der SD-Karte aus.

Jumper Konfiguration für SD-Karten Programmierung

2.1 Vorbereitung zur Laborübung

Memory Mapped I/O und isolated I/O sind zwei Methoden um Input-Output Operationen zwischen CPU und der Peripherie auszuführen.

Memory Mapped I/O
Es wird der gleiche Adressbus verwendet, um den primären Speicher und den Speicher der Hardwaregeräte anzusteuern. Das bedeutet, die Befehle um bestimmte Bereiche im Speicher anzusprechen, können ebenfalls verwendet werden um die Speicherbereiche der Hardware anzusprechen.

Isolated I/O
Auf der anderen Seite, verwendet Isolated I/O separate Befehle um den primären und den Gerätespeicher anzusprechen. In einem solchen Fall liegen zwei separate Adressenbereiche vor. Dies können z.B. separate I/O Pins an der CPU oder ein kompletter eigener Bus sein. Da der primäre Speicherbereich von dem Geräteadressbereich getrennt ist, spricht man von Isolated I/O. Isolated IO benötigt spezielle Befehle um zu schreiben und zu lesen.

3 Aufgabe 2

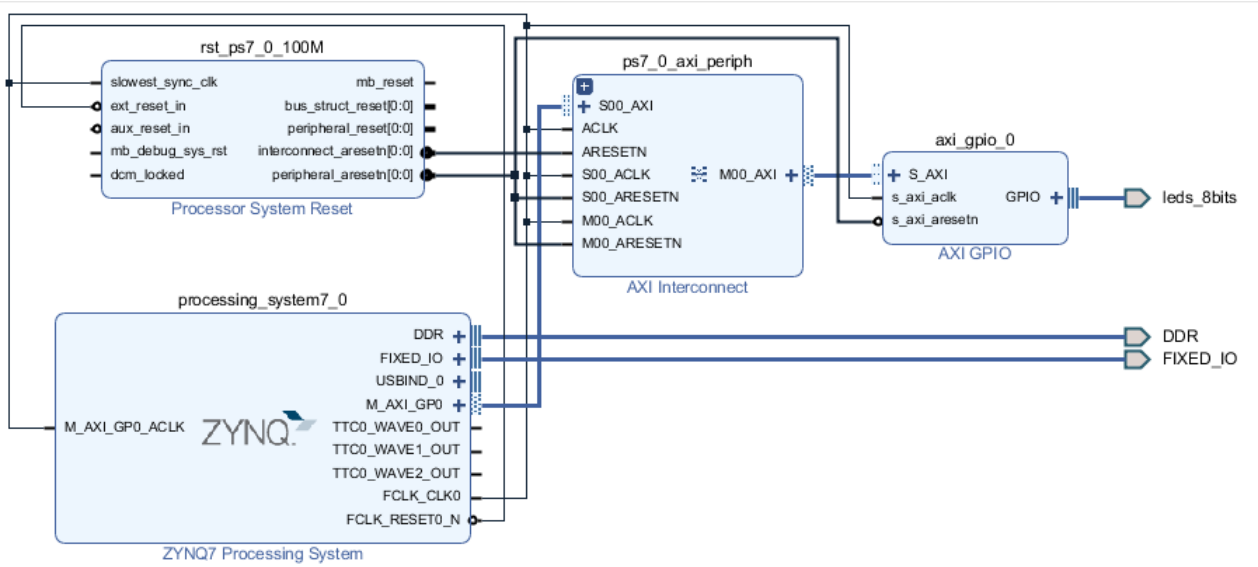
Konfiguration des IP Cores zur Realisierung der Kommunikation zwischen Prozessor und Peripherie als Blockdesign in Vivado 2016.2. Eingebunden in diese wurden zusätzlich die 8Bit-LED-Anzeige und anschließend der Aufbau in VHDL synthetisiert.

3.1 Beschreibung der Funktion XGpio_DiscreteWrite

3 Checks ob Pointer valide, Status Ready, welcher Channel aktiv ist und ob zwei Kanäle von der Hardware unterstützt werden. Danach werden Basispointer und Adressoffset addiert und an XGPio_WriteReg übergeben. Kanäle (Registerbänke) liegen im Speicher nacheinander, ein Umschalten erfolgt über einen Adressoffset.

3.2 Woran/Wo sieht man die Antwort von der Vorbereitung im Code?

In der Funktion XGPIO_Initialize wird über die DeviceID ein Instanz-Zeiger (InstancePtr) erzeugt und initialisiert. Dieser Instanz-Zeiger wird später verwendet, um die genauen Speicheradressen zu berechnen. Diese werden zum Konfigurieren der I/Os als (Ein- oder Ausgang) und zum Setzen der einzelnen Bits verwendet.



Vivado: Schemaplan der IP-Cores

Ausgehend von einer Basisadresse, werden alle weiteren Offsets dazu gezählt.

XPAR_AXI_GPIO_0_BASEADDR

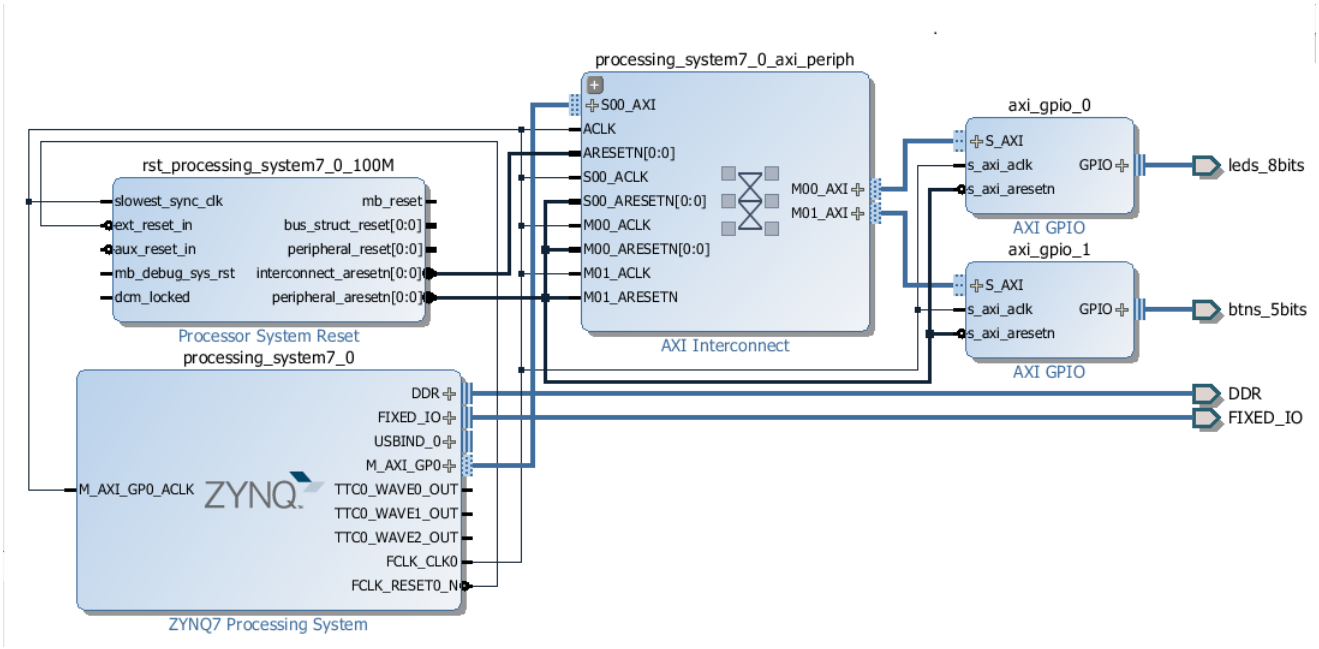
0x41200000

Channel 1 (8Byte)		Channel 2 (8Byte)	
Data 4Byte	Data direction 4Byte	Data 4Byte	Data direction 4Byte

Speicherbereich zum Setzen der Richtung und der Daten

4 Aufgabe 3

Konfiguration des IP Cores zur Realisierung der Kommunikation zwischen Prozessor und Peripherie als Blockdesign in Vivado 2016.2. Eingebunden in diese wurden zusätzlich die 8Bit-LED-Anzeige und die Push-Buttons. Der Aufbau wurde anschließend in VHDL synthetisiert.

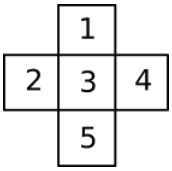


Vivado: Schemaplan der IP-Cores

Programmcode zur Abfrage der Buttons and Zuweisung des Bit-Musters direkt an die LEDs.

```
#include  <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"
//  Definitions
#define  LEDS_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define  BTNS_DEVICE_ID  XPAR_AXI_GPIO_1_DEVICE_ID

#define  LED_DELAY  1000000
#define  LED_CHANNEL 1
#define  printf  xil_printf

XGpio  Gpio;
XGpio LEDInst , BTNInst ;
static int btn_value ;

int  LEDOutputExample( VOID ) {
volatile  int  Delay;
// loop  forever  blinking  the  LED.
while( 1 ) {
// read buttons
btn_value = XGpio_DiscreteRead (&BTNInst , 1);
//  write  button value  to LEDs
XGpio_DiscreteWrite( &LEDInst , LED_CHANNEL , btn_value );
```

```
for ( Delay = 0;  Delay  < LED_DELAY; Delay++ );
}
return  XST_SUCCESS;
}

int  main()
{
init_platform ();

int status ;
// ----------------------------------------------------
// INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
// ----------------------------------------------------
// Initialise LEDs
status = XGpio_Initialize (&LEDInst , LEDS_DEVICE_ID );
if ( status != XST_SUCCESS )
return XST_FAILURE ;

// Initialise Push Buttons
status = XGpio_Initialize (&BTNInst , BTNS_DEVICE_ID );
if ( status != XST_SUCCESS )
return XST_FAILURE ;

// Set LEDs direction to outputs
XGpio_SetDataDirection (&LEDInst,1,0x00);
// Set all buttons direction to inputs
XGpio_SetDataDirection (&BTNInst,1,0xFF);

print("Hello  meine  LED\n\r");

LEDOutputExample ();
cleanup_platform ();
return  0;
}
```



Zuordnung der Buttons

Das zugehörige Bitmuster pro Button ist:
Button1 BTNU: 0001 0000b, 0x10
Button2 BTNL: 0000 0100b, 0x04

Button3 BTNC: 0000 0001b, 0x01
Button4 BTNR: 0000 1000b, 0x08
Button5 BTNO: 0000 0010b, 0x02

5 Glossar

Beschreibung der wichtigsten Abkürzungen, die in der Übung verwendet werden.

Glossar
PS Processing System, z.B. PS memory
PL Programmable Logic, z.B. PL memory
AXI Advanced eXtensible Interface as protocol for Intellectual Property (IP) cores