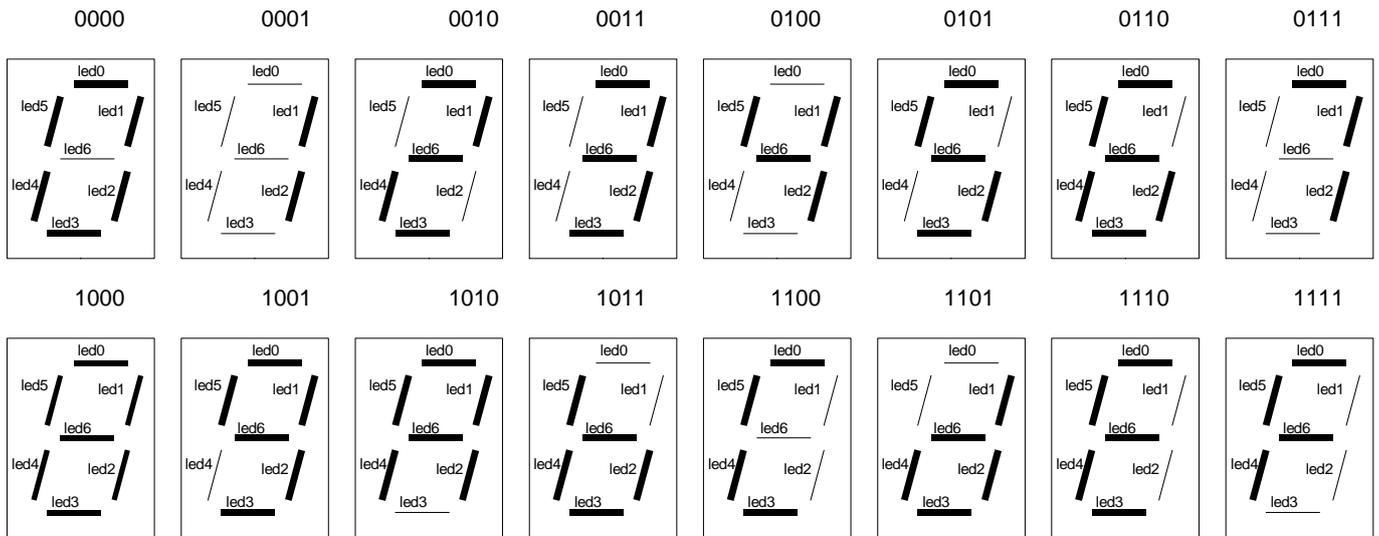
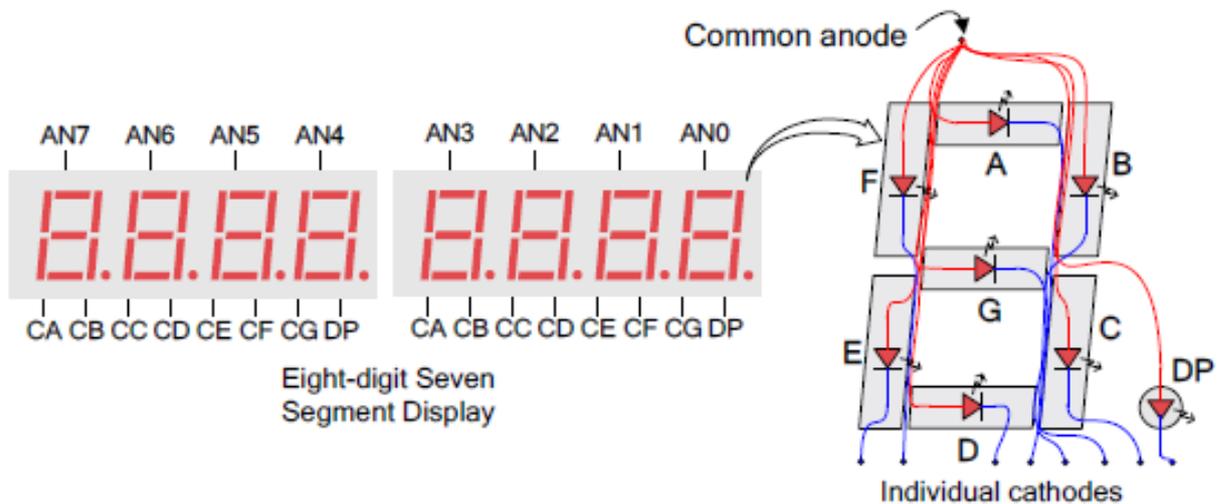


# Aufgabe 1: Kombinatorischer 7-Segment-Decoder



Alle LED's **einer** 7-Segment-Anzeige haben eine gemeinsame Anode, die Kathode eines jeden Segments ist getrennt herausgeführt

Verschaltung:



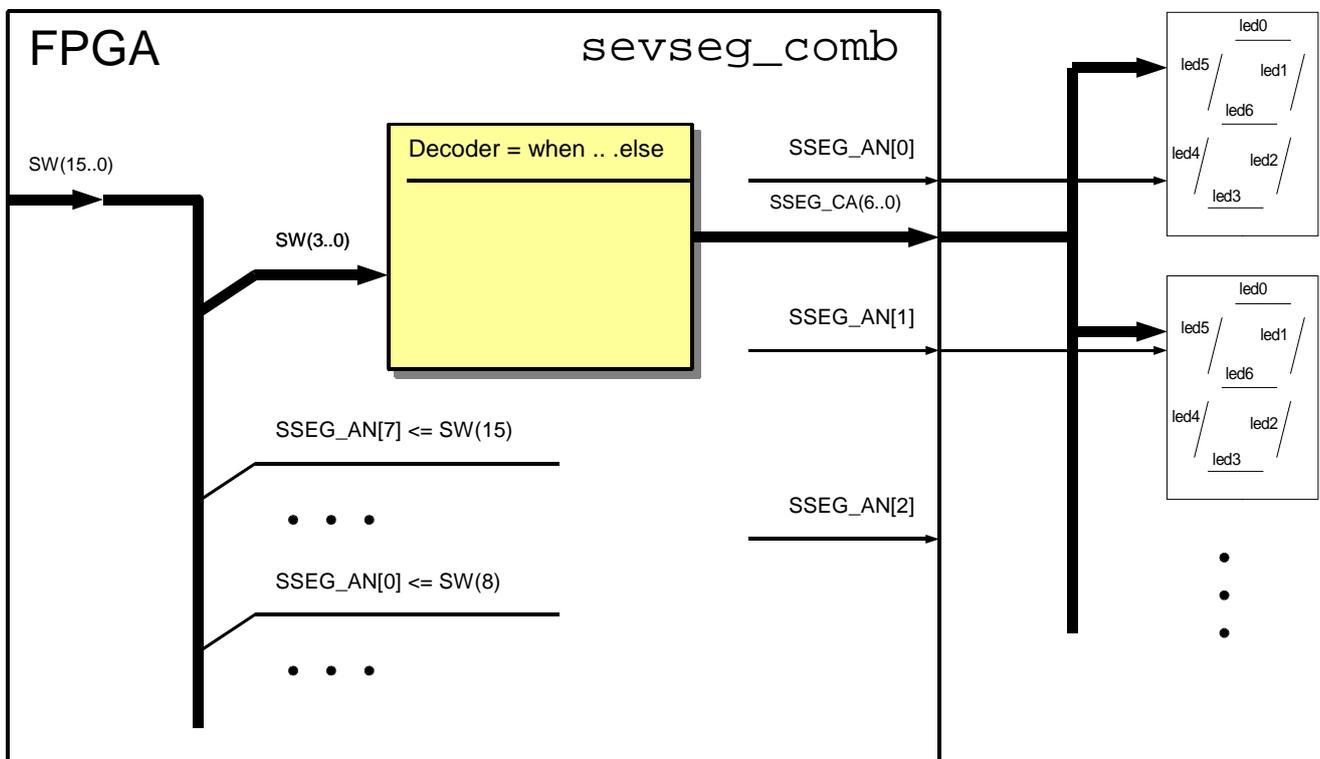
- Die Ausgänge CA .. CG steuern die einzelnen Segmente,
- Ein Einzeldisplay leuchtet, wenn die zugehörige Anode (AN7 .. AN0) einen Low-Pegel aufweist

Die folgende Tabelle zeigt die Zuordnung der darzustellenden Hex-Zahlen zu den Anschlüssen der 7-Segment-Anzeige:

NR	Hex	d3	d2	d1	d0
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	A	1	0	1	0
11	B	1	0	1	1
12	C	1	1	0	0
13	D	1	1	0	1
14	E	1	1	1	0
15	F	1	1	1	1

CG	CF	CE	CD	CC	CB	CA
led6	led5	led4	led3	led2	led1	led0
1	0	0	0	0	0	0
1	1	1	1	0	0	1
0	1	0	0	1	0	0
0	1	1	0	0	0	0
0	0	1	1	0	0	1
0	0	1	0	0	1	0
0	0	0	0	0	1	0
1	1	1	1	0	0	0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	1
1	0	0	0	1	1	0
0	1	0	0	0	0	1
0	0	0	0	1	1	0
0	0	0	1	1	1	0

### Struktur des Decoders



Zuordnung der Signalnamen:

File "Nexys4\_Master.xdc", vom Hersteller mitgeliefert:

Nexys4\_Master.xdc

Die Namen der Ein- und Ausgänge müssen mit den Namen im xdc-File übereinstimmen, dann wird das Signal mit dem dazugehörigen Pin verbunden

Syntax xdc-File:

```
##Bank = 34, Pin name = IO_L21P_T3_DQS_34, Sch name = SW0  
set_property PACKAGE_PIN U9 [get_ports {SW[0]}]  
set_property IOSTANDARD LVCMOS33 [get_ports {SW[0]}]
```

# ist Kommentar, wenn also die beiden obigen Zeilen nicht auskommentiert sind, dann ist das Signal SW(0) mit dem Pin U9 verbunden

Alle vorhandenen Signale müssen eine Entsprechung im xdc-File haben

Namen im xdc-File können geändert werden, besser: Namen im VHDL-File anpassen

Achtung: Laut Norm ist VHDL nicht Case-Sensitiv. Die Zuordnung der Signalnamen im Constraints-File ist es aber schon !

Daher: Wenn im xdc-File ein Name großgeschrieben ist, dann auch im VHDL-File großschreiben (sonst kryptische Fehlermeldungen)

Es soll die Stellung der 4 rechten Schalter als Hex-Wert angezeigt werden:

Signalnamen der Switches: SW0 .. SW15. SW0 ist ganz rechts

Die Schalter SW15 .. SW8 schalten die einzelnen Displays ein oder aus

VHDL-File ist schon vorhanden: sevseg\_comb.vhd

Aufgabenstellung:

Kennenlernen Vivado-Design-Umgebung

- Verzeichnis für Projekt erstellen
- Projekt erstellen: FPGA: XC7A100T1CSG324-3
- vhd-File reinkopieren (Design Sources)
- xdc-File reinkopieren (Design Sources)
- xdc-File Namen überprüfen  
Schaltung kann jetzt nicht simuliert werden, da keine Testbench vorhanden
- Synthese ist möglich: Bitstream erstellen und ins FPGA downloaden
- Test mit Schalterstellungen

Simulation:

Testbench erstellen: File tb\_sevseg.vhd in Projekt reinkopieren, ergänzen, simulieren

## Aufgabe 2: Kombinatorischer 7-Segment-Decoder

### Lernziele:

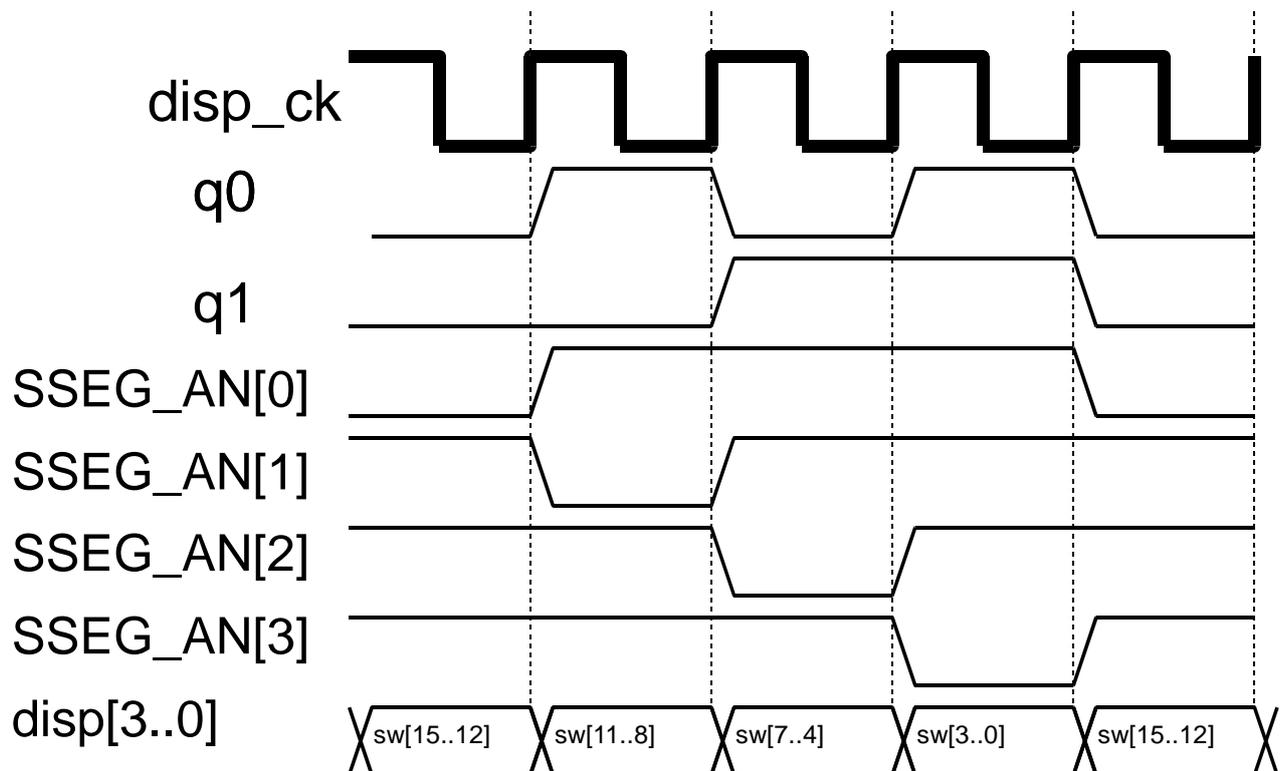
Hierarchisches Design (Einbinden von Komponenten)

Umgang mit sequentiellen Blöcken (process)

Umgang mit vorgefertigter IP (Clock-Wizard)

Unterschiedliche Anzeigen sind nur möglich, wenn jeweils nur eine Anode low wird und der dazugehörige Wert an die Eingänge CA .. CG gelegt werden.

Es sollen jetzt jeweils 4 Schalter jeweils eine Hex-Zahl darstellen. Die Signale der 4 Schalter werden zyklisch auf den 7-Segment-Decoder gelegt und jeweils die zugehörige Anode auf Low gelegt:

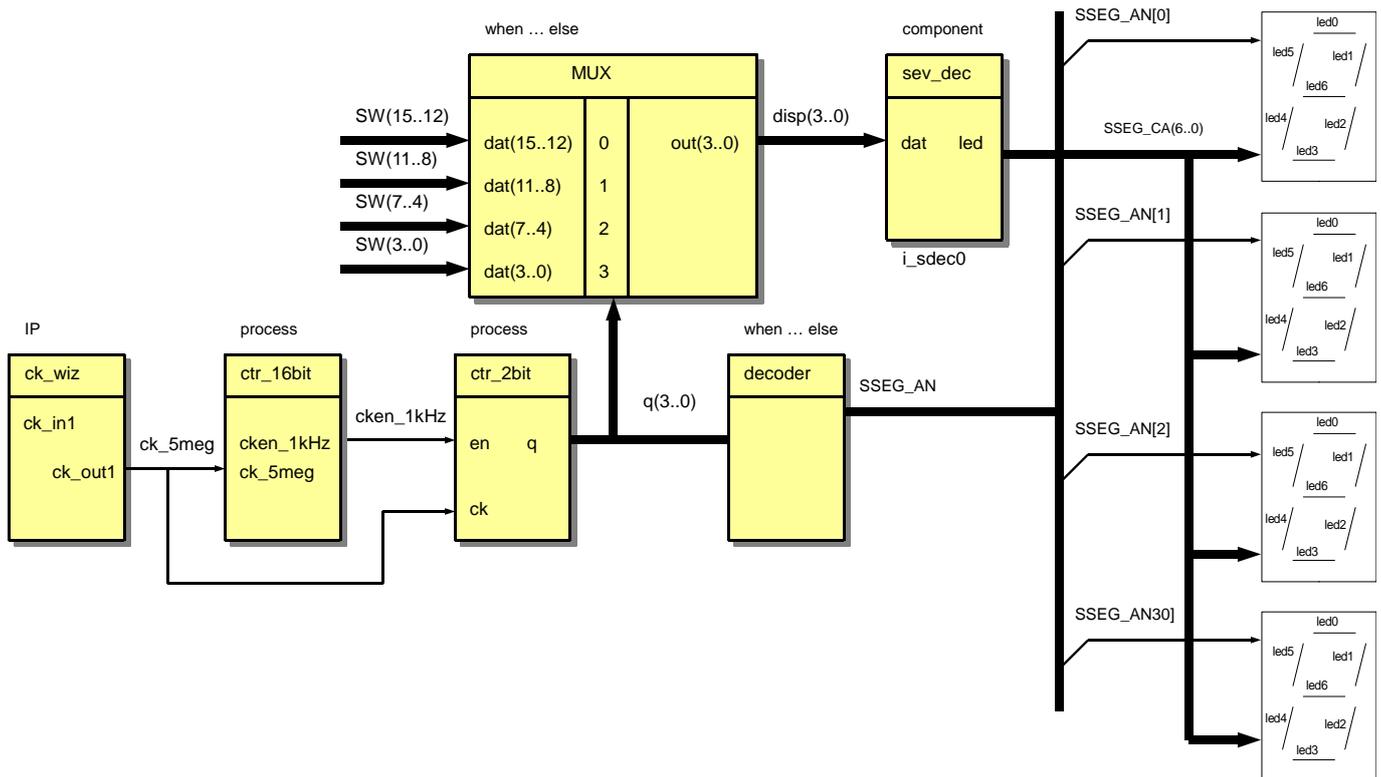


Es steht ein 100MHz-Takt zur Verfügung, ein Segment sollte aber ca. 1ms lang leuchten, damit sich ein flackerfreies, helles Bild ergibt

Der 100MHz-Takt wird daher mit einem Ck-Manager heruntergeteilt, allerdings ist der maximal Teilerfaktor begrenzt, so dass eine Minimalfrequenz von 5MHz erreichbar wird.

Diese steuert eine Enable-Counter, der im Abstand von 1ms einen Enable-Impuls von 200µs liefert, und damit den eigentlichen Display-Zähler weitertaktet

## Struktur des gemuxten Displays



### Vorgehensweise:

- Neues Projekt erstellen
- sevseg\_comb.vhd reinkopieren und so abändern, dass nur ein 4-Bit-Eingangsvektor und ein 6-Bit-Ausgangsvektor übrig bleibt (Anoden werden separat gesteuert)
- Clock-Wizard IP erstellen und konfigurieren (100MHz-Eingang, 5MHz Ausgang)
- Neues VHD-File erstellen:
  - Eingänge: Clock 100Mhz (Board-Clock, siehe Nexys4\_Master.xdc), SW(15..0)
  - Ausgänge: Kathoden (SSEG\_CA(6..0)), Anoden (SSEG\_AN(7 .. 0))
  - Komponenten Clock-Wizard und sev\_dec einbauen (instanzieren)
  - Restliche Schaltungseinheiten als Process bzw. Concurrent-Statement erstellen (siehe Blockschaltbild)
- Testbench erstellen
- Debuggen, simulieren, testen
- Wenn Simulation ok, in FPGA laden und testen

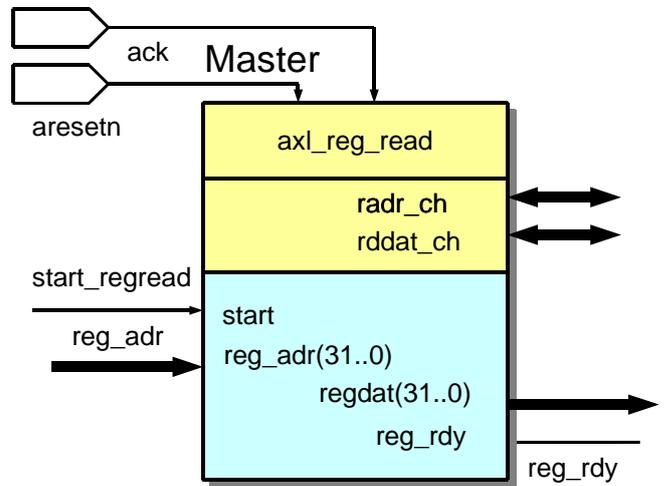
# Aufgabe 3: Kommunikation mit IP (UART) über AXI-Bus

## Lernziele:

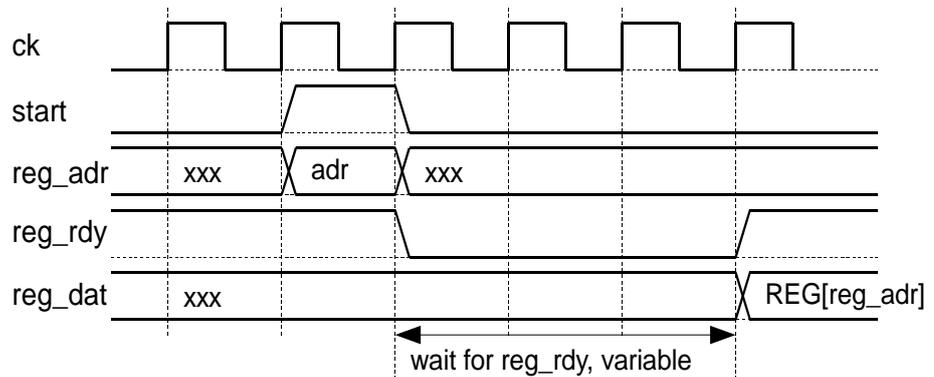
Kennenlernen AXI-Bus (neuer Bus-Standard für Xilinx-IP, ARM-Standard)  
Systematisches Design auf RTL-Ebene

### 3.1 Register-Leser (reg\_read)

Der Register-Leser soll das Lesen eines Registers vereinfachen, indem eine vereinfachte Schnittstelle zur Verfügung gestellt wird:

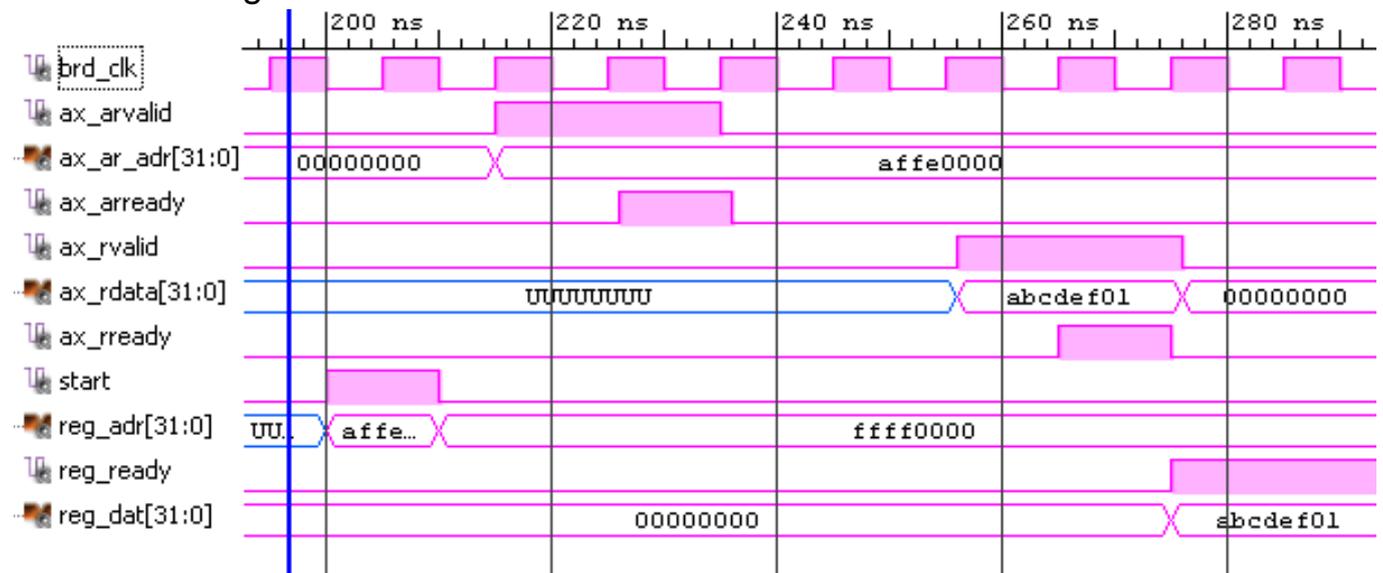


Die vereinfachte Schnittstelle soll über das nebenstehende Timing angesprochen werden:



reg\_read muss die Signale für das AXI-Interface erzeugen bzw. entgegennehmen

Gesamt-Timing:



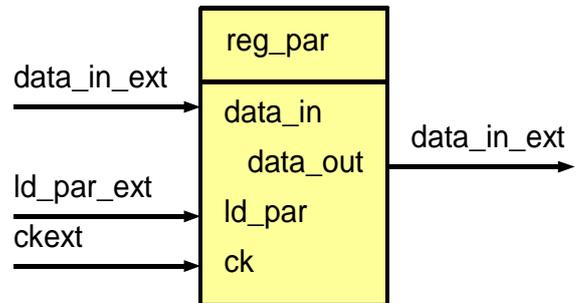
In reg\_read\_timing.ods finden Sie bereits eine Tabelle mit einigen vorbereiteten Signalen

fsm\_reg\_read\_empty.ods stellt eine Datei für die FSM-Synthese dar. Die passenden Signale, Zustände und Übergangsbedingungen müssen noch eingebaut werden.

reg\_read\_empty.vhd enthält eine Rahmendatei für das Design.

Ein Parallelregister ist bereits in der Datei "regpack.vhd" vorbereitet, ein Beispiel für die Instantiation ist in reg\_read\_empty.vhd enthalten.

Anschluss des Registers:



tb\_reg\_read\_empty.vhd enthält den Rahmen für eine Testbench.

tb\_reg\_read\_empty.vhd enthält einen Generator für das Reset-Signal, kann so übernommen werden (reset\_gen\_ctr).

Vorgehensweise:

### RTL-Design

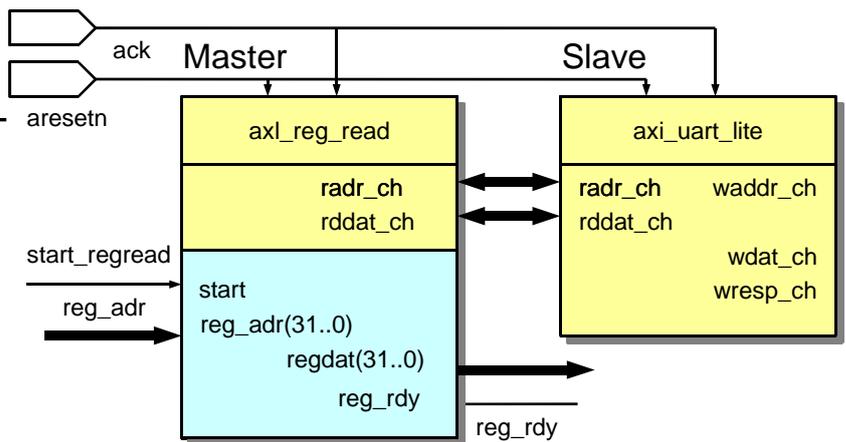
- Registerskizze erstellen, Signalnamen eintragen
  - Falls Sie OpenOffice Draw verwenden möchten, steht unter "example.odg" eine Zeichnung mit Beispiелеlementen zur Verfügung, die kopiert und modifiziert werden können
- Timing-Tabelle reg\_read\_timing.ods ergänzen, evtl. zusätzlich nötige Signale eintragen und Zeitverlauf eintragen
  - Zustandsnamen passend zu den benötigten Steuersignalen eintragen
- FSM entsprechend Timing erstellen, FSM synthetisieren
  - Komponentendeklaration und Instantiations-Template werden im File mit der Endung .vho generiert

### Codierung, Simulation

- Projekt erstellen
- Benötigte Files reinkopieren
- vhd-Files ergänzen
- Testbench ergänzen
- Simulieren

### 3.2 Anwendung des "Register-Lesers" zur Kommunikation mit einer UART-Schnittstelle

Verwendet wird: XILINX-IP "AXI-UARTLITE" aus dem IP-Catalog (Datenblatt: pg142-axi-uartlite.pdf)



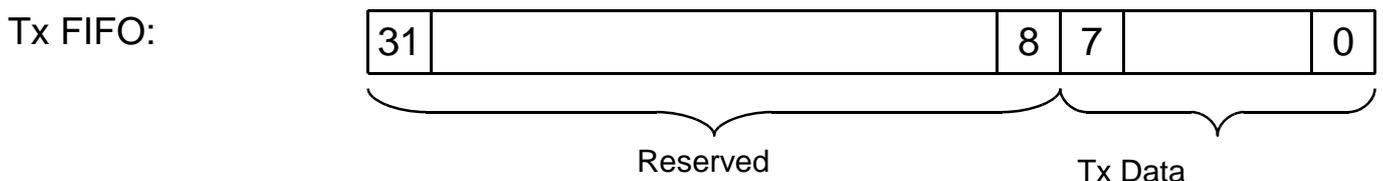
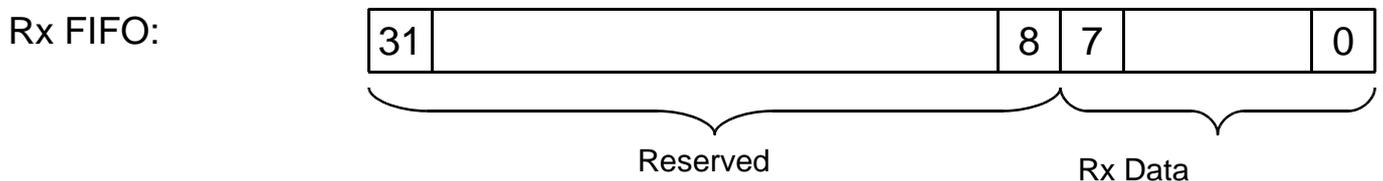
In "uart\_top\_ro" befinden sich vorbereitete VHDL-Rumpf-Files, die die Instanziierung und Signaldeklarationen für "AXI-UARTLITE" bereits enthalten

"uart\_top\_ro" enthält den Reset-Generator für die AXI-Komponente (reset\_gen\_ctr) und den Anschluss des Taktes an ein dezidiertes CK-Netzwerk (clkf\_buf: BUFG)

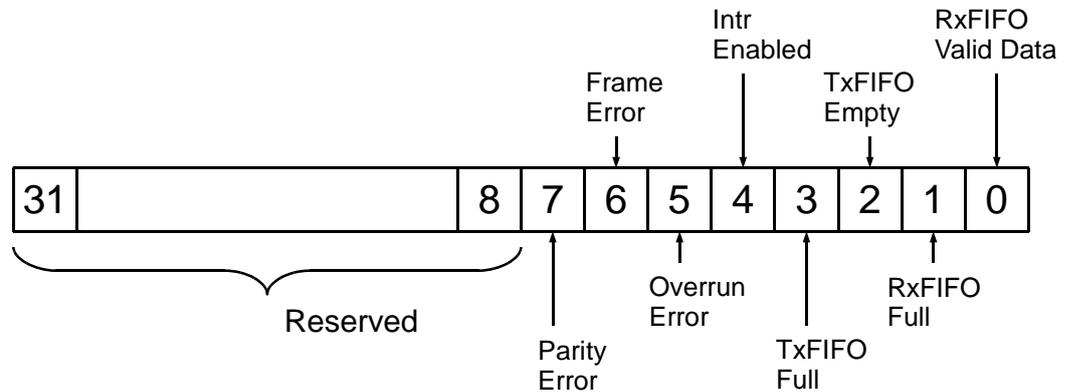
Wenn ein neues IP aus dem Katalog implementiert wird, kann ein Beispiel-Design angelegt werden (Vorgehensweise siehe Dokumentation des jeweiligen IP). Aus dem Beispiel-Design können die Instantiation und die passenden Signaldeklarationen übernommen werden

Die Kommunikation mit AXI-UARTLITE erfolgt über Register:

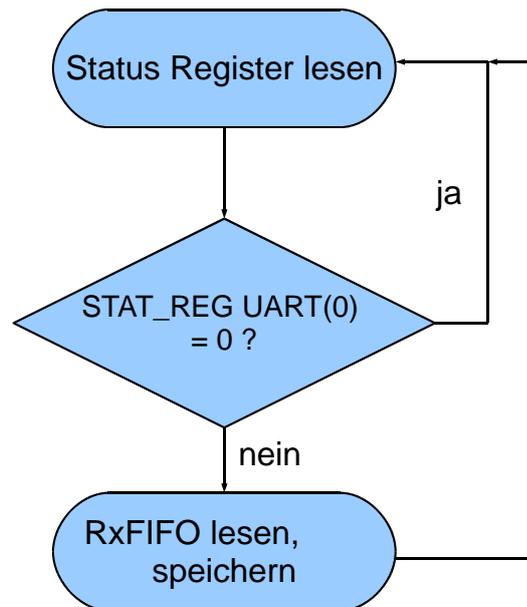
Address	Register Name	Description
0h	Rx FIFO	Receive data FIFO
04h	Tx FIFO	Transmit data FIFO
08h	STAT_REG UART	Lite status register
0Ch	CTRL_REG UART	Lite control register



## Status Register (STAT\_REG)



## Empfangen eines Byte:



## Vorgehensweise:

### RTL-Design

- Registerskizze erstellen, Signalnamen eintragen
- Timing-Tabelle `uart_top_ro_timing_empty.ods` ergänzen, evtl. zusätzlich nötige Signale eintragen und Zeitverlauf eintragen
  - Zustandsnamen passend zu den benötigten Steuersignalen eintragen
- FSM entsprechend Timing erstellen, FSM synthetisieren
  - Komponentendeklaration und Instantiations-Template werden im File mit der Endung `.vho` generiert

### Codierung, Simulation

- Projekt erstellen
- IP AXI-UARTLITE synthetisieren
- Benötigte Files reinkopieren
- vhd-Files ergänzen
- Testbench ergänzen
- Simulieren
- Test Hardware

Zur Testbench:

Siehe Skript "Fortgeschrittenes Testbench-Design" :

Testdaten können aus Datei "test\_data.dta" eingelesen werden

Die 8-Bit-Werte müssen aber noch serialisiert werden

Der folgende Code setzt eine aus der Datei gelesene Zeile in eine serielle Folge der einzelnen Bits um:

```
readline(test_in,in_line);
read(in_line,in_sigs);
--seriell Daten in UUT schaufeln
--Datenfiles sind 8 bit
--Startbit:
rx <= '0';
wait for ??? us;
for i in 0 to 7 loop
    rx <= in_sigs(i);
    wait for ??? us;
end loop;
--Stopbit:
rx <= '1';
wait for ??? us;
```

Die Zeiten in "wait" müssen der gewählten Baudrate angepasst werden

Die Ausgänge "rxfifo\_dat: OUT STD\_LOGIC\_VECTOR (7 downto 0); sind bereits über das File "uart\_top\_ro.xdc" mit den LED-Anschlüssen verbunden Daten, die über ein Terminalprogramm oder "send\_byte\_uart.tcl" (Start mit ased) gesendet werden, werden an den LED's sichtbar

Debugging mit Logic-Analyzer

Die datei "uart\_top\_ro\_raw.vhd" ist bereits mit der Schnittstelle zum Logic-Analyzer ausgestattet (debug : OUT STD\_LOGIC\_VECTOR (15 downto 0))

Diese Signale sind bereits in "uart\_top\_ro.xdc" mit den passenden Anschlüssen des NEYSX-Boards verbunden. An diese Debugging-Schnittstelle können beliebige interne Signale angeschlossen werden.

Das File "uart\_top\_ro.LPF" enthält die Signalnamen, die in "uart\_top\_ro.LPF" mit der Debugging-Schnittstelle verbunden sind

Starten Logic-Analyzer:

Programm "LogicPort Application" starten, File "uart\_top\_ro.LPF" öffnen, Signale müssten sichtbar werden.

Mit "send\_byte\_uart.tcl" können permanent Daten an die Schnittstelle gesandt werden, die resultierenden Signale können jetzt hardwaremäßig überprüft werden.

### 3.3 Vorschläge für weitere Projekte:

- 7-Segment-Display mit einbauen, Anzeige der empfangenen Daten in Hex
- Erweiterung des UART-Empfängers zum Empfänger/Sender:
  - Register-Schreiber entwickeln
  - Register-Schreiber in Gesamt-Projekt einbauen, Statemachine erweitern, z. B. empfangene Daten wieder senden