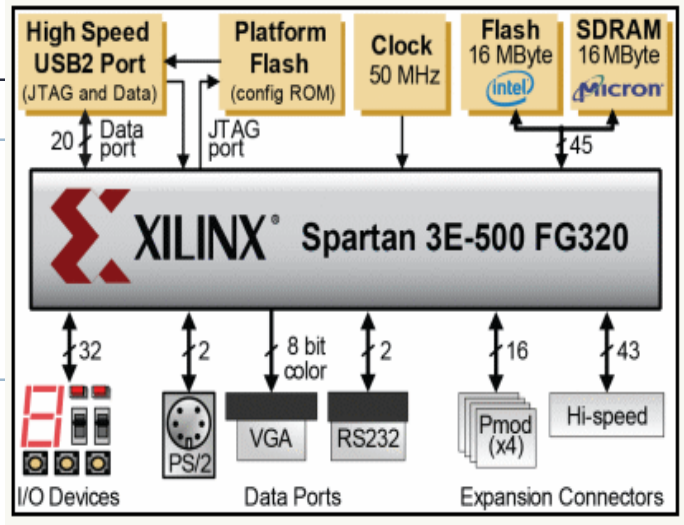
Computer Engineering
WS 2010

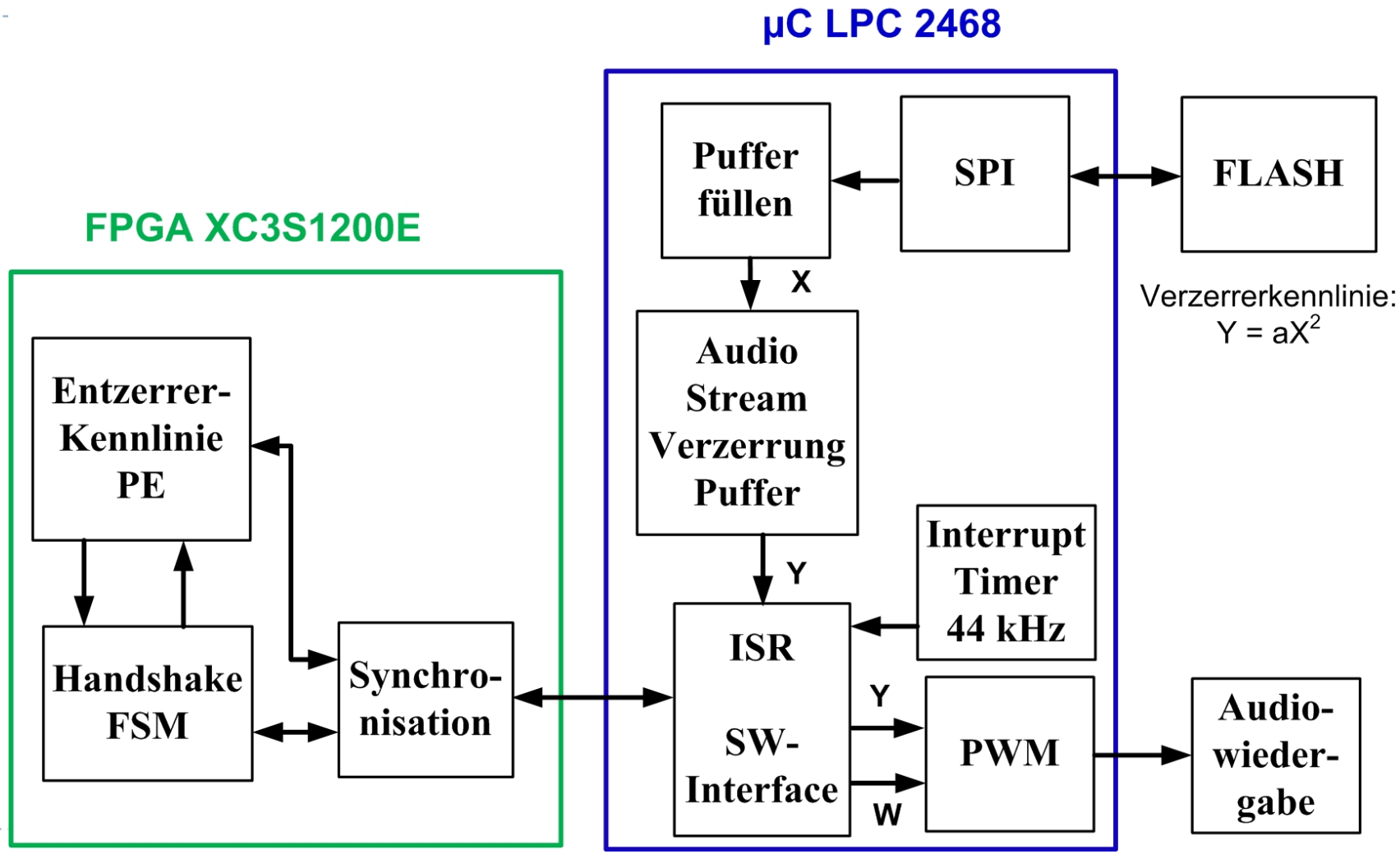
Digitale Systeme



µC – FPGA Kommunikation

- µC – FPGA Kopplung
- Kommunikation zwischen asynchronen Clock-Bereichen
- Vier-Phasen Handshake
- Handshake – FSM
- Tristate - Schnittstellen
- ISR – Kommunikation mit FPGA
- Macros zur Port-Steuerung
- Meßergebnisse

CE-Labor-System



Asynchrone Eingänge

FPGA-intern:

- Synchrone Systeme mit einer Clock als Referenzsignal.
- Taktflankenereignis bestimmt die Datenaufnahme und die Ausgangsaktualisierung.
- Bedingung: $T_{CLK} > T_{PQ} + T_{LOGIK} + T_{SU}$

Externe Eingänge:

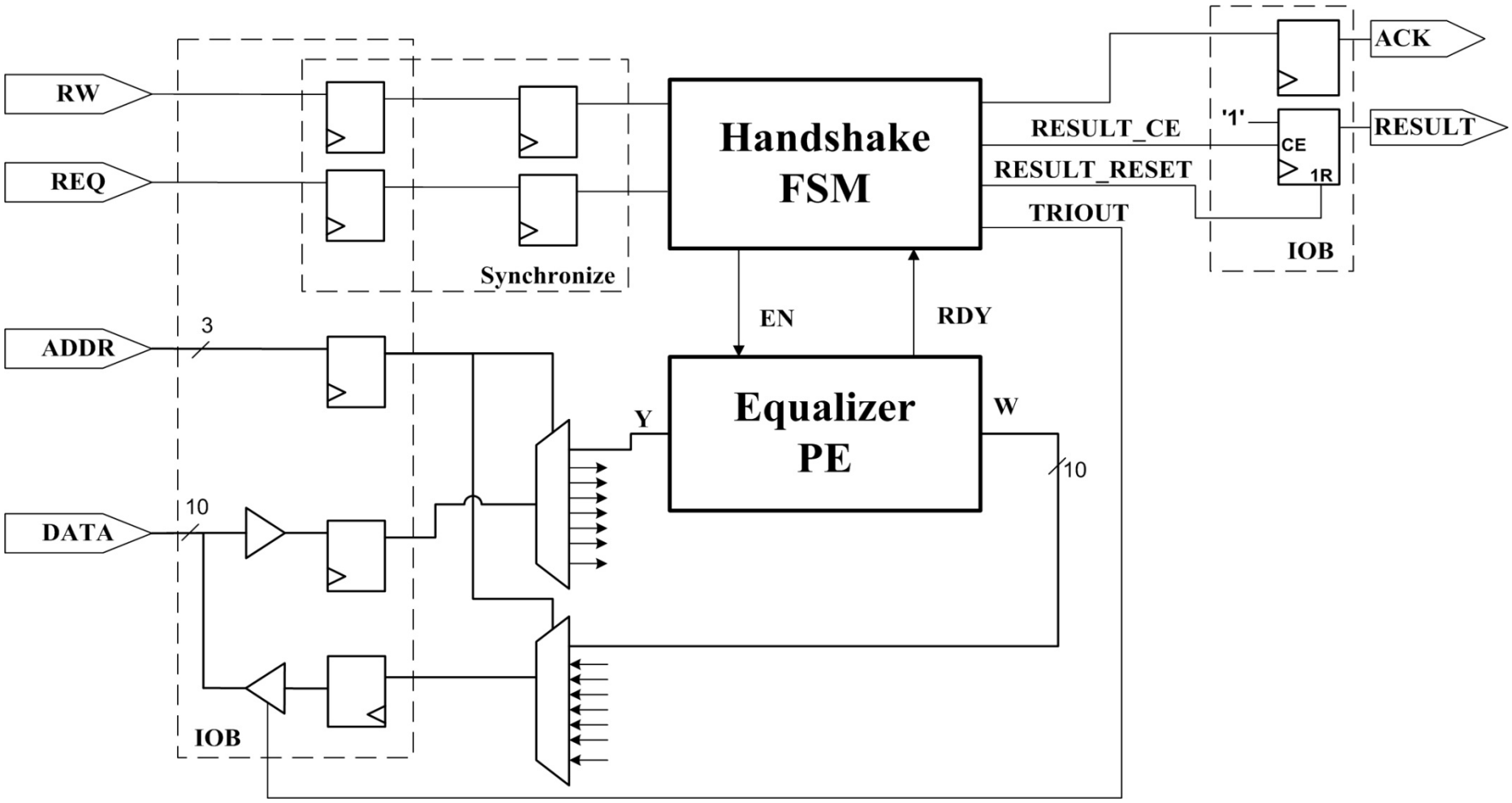
- Pegeländerungen treten unabhängig von der FPGA-Clock auf.
- µC-Clock ist kein ganzes Vielfaches der FPGA-Clock.
- Phasenlage der Clocks ist unbestimmt.
- Variable Taktanzahl pro C-Anweisung.

Asynchrone Kommunikation

Ereignisgesteuerte Effekte an den FPGA-Eingängen erfordern Ansatz ohne Abhängigkeit von einem gemeinsamen oder übertragenen Clock-Signal.

- **Sender-Empfänger-Kommunikation mit einem Protokoll, das verzögerungsunabhängige Signalisierungskonventionen nutzt.**
- **Jede Komponente arbeitet mit seiner eigenen Taktrate.**
- **Nur für Interaktionen findet eine Kommunikation mit synchronisierten Abläufen statt.**

Asynchrone Kommunikation
µC - Beschleuniger



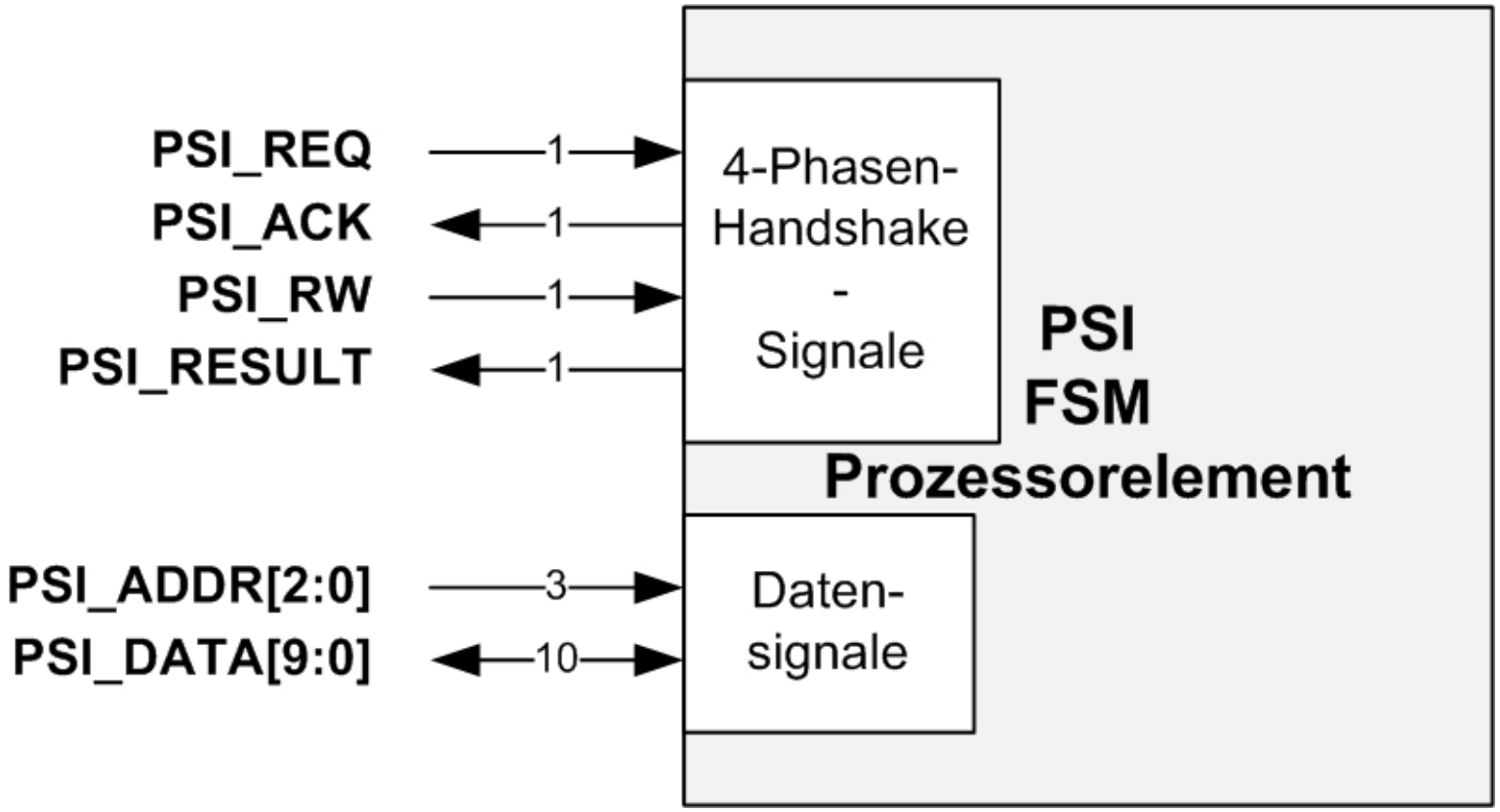
Beschleuniger-Schnittstellen

- **Eingangssignale werden durch D-FFs auf den FPGA-Takt synchronisiert:**

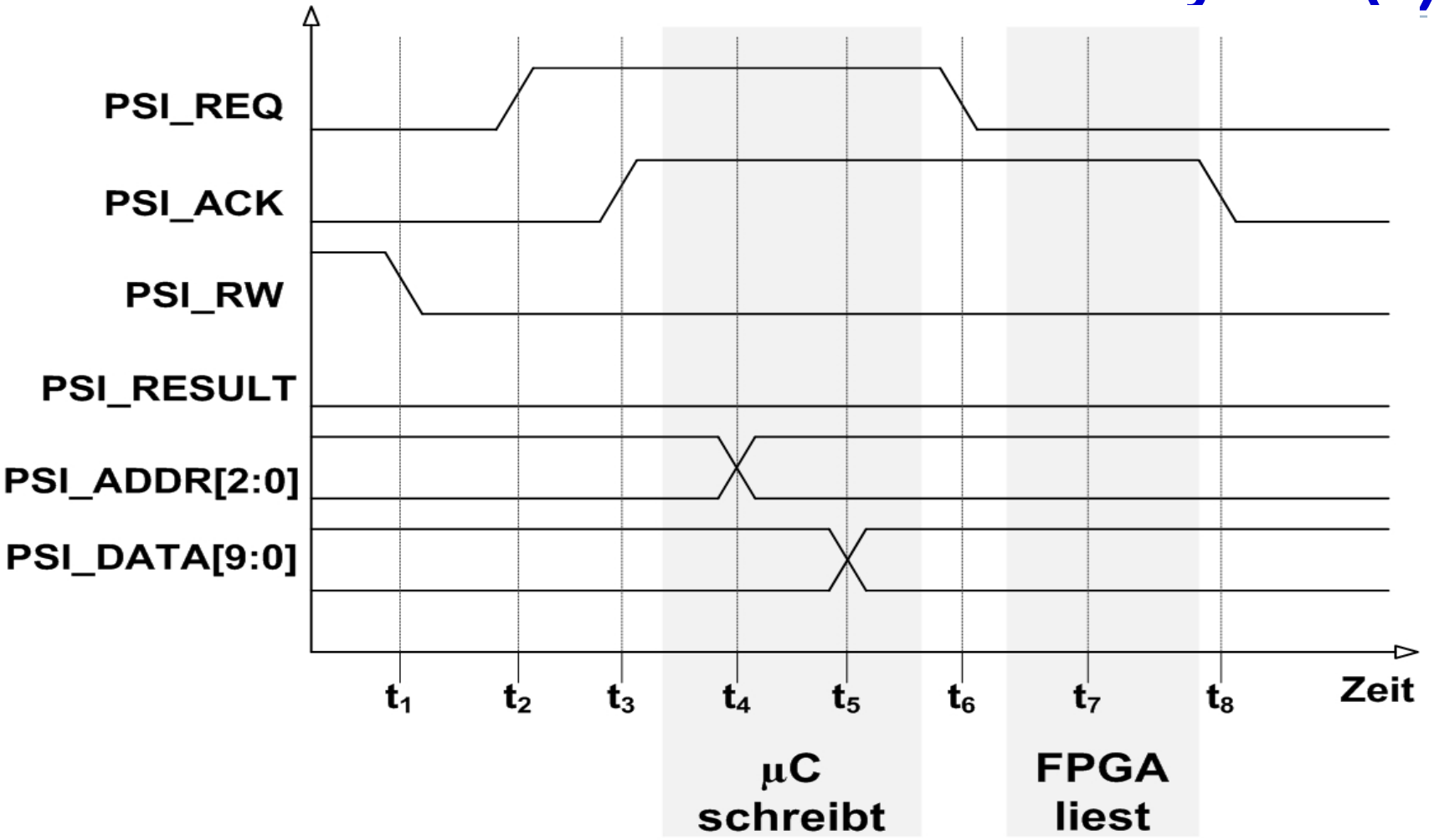
  Pegel ändern sich gleichzeitig.

  Weniger, kürzere Hazards.
- **Ausgangsregister liefern eine parallele Pegelaktualisierung.**
- **D-FFs in den Input-Output-Blocks (IOBs) stehen für beide Richtungen zur Verfügung.**
- **Abstimmungseingangssignale (REQ, RW), die die getaktete Zustandssequenz in der FSM beeinflussen, sind einer speziellen Synchronisation zu unterziehen.**

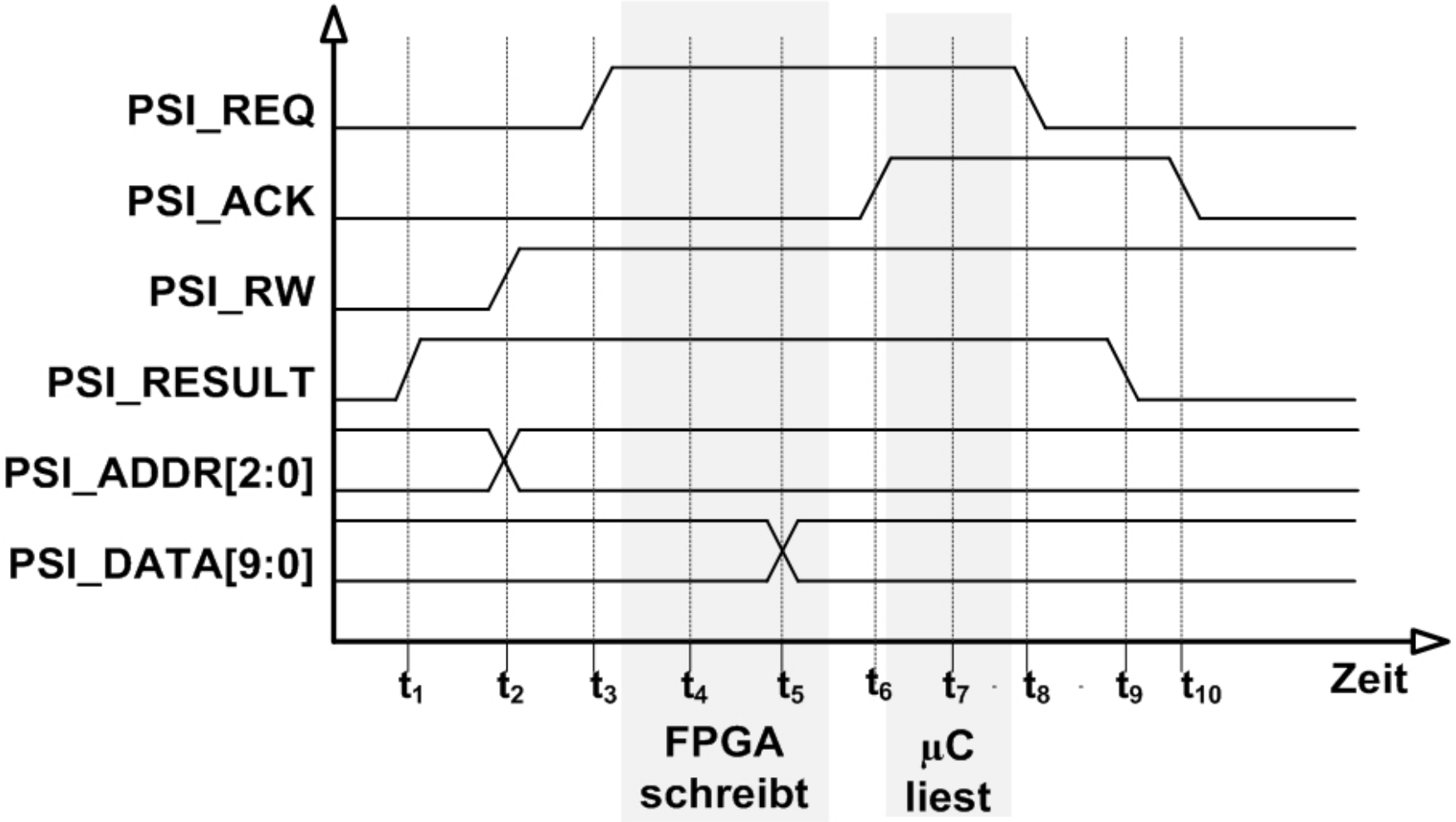
µC – FPGA Interface



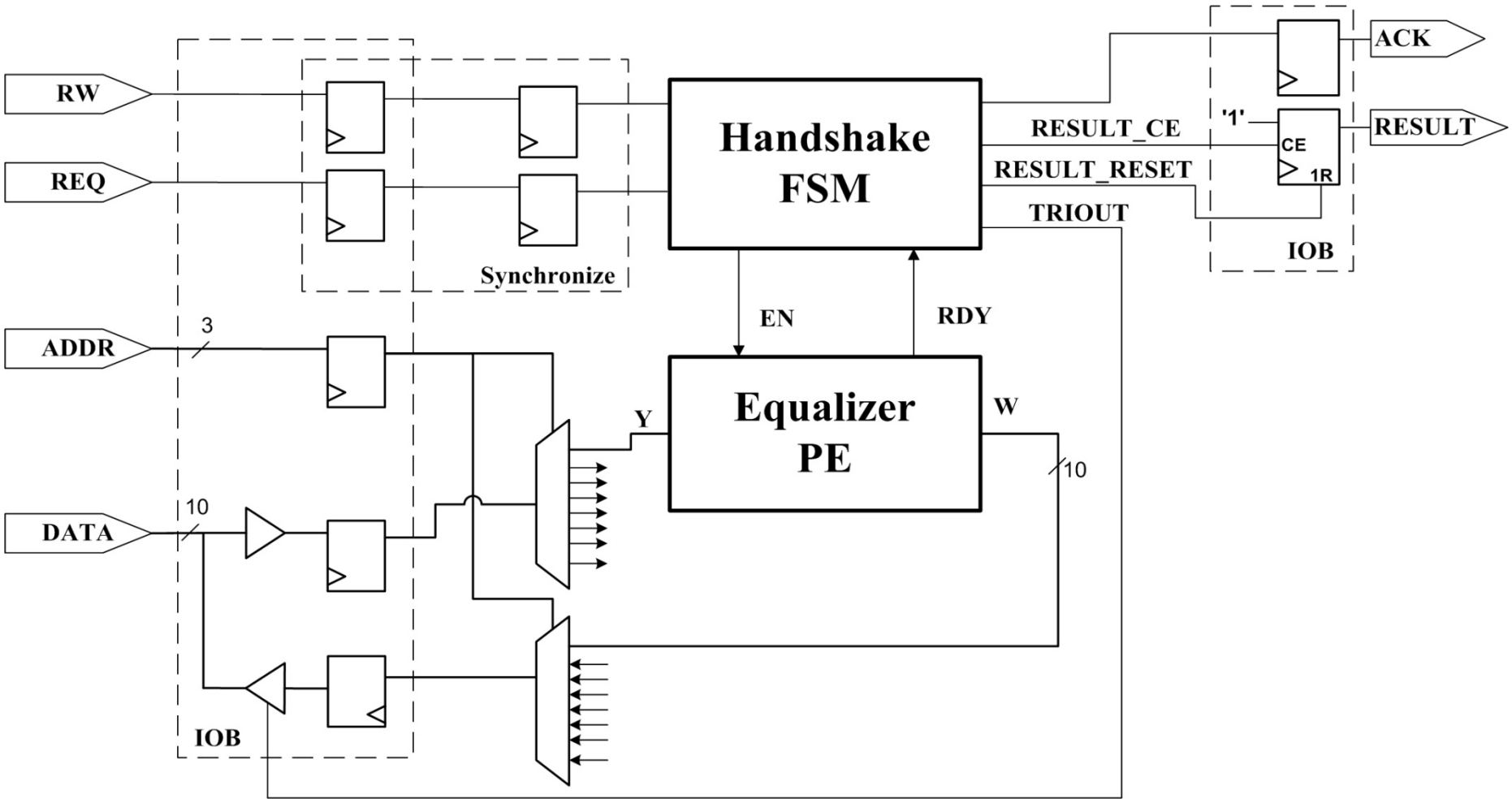
Datentransfer-Phasen im Handshake-Zyklus (1)



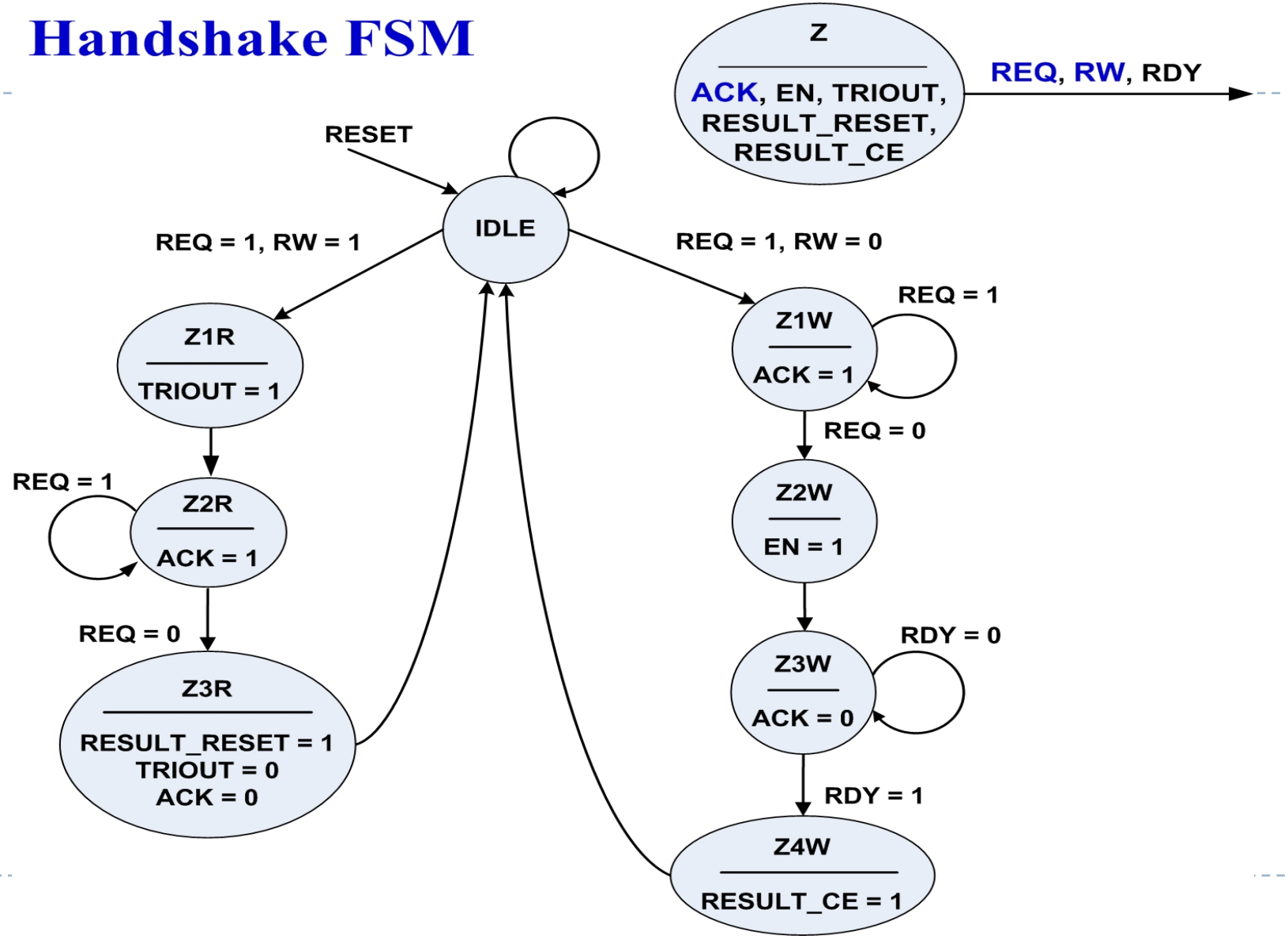
Datentransfer-Phasen im Handshake-Zyklus (2)

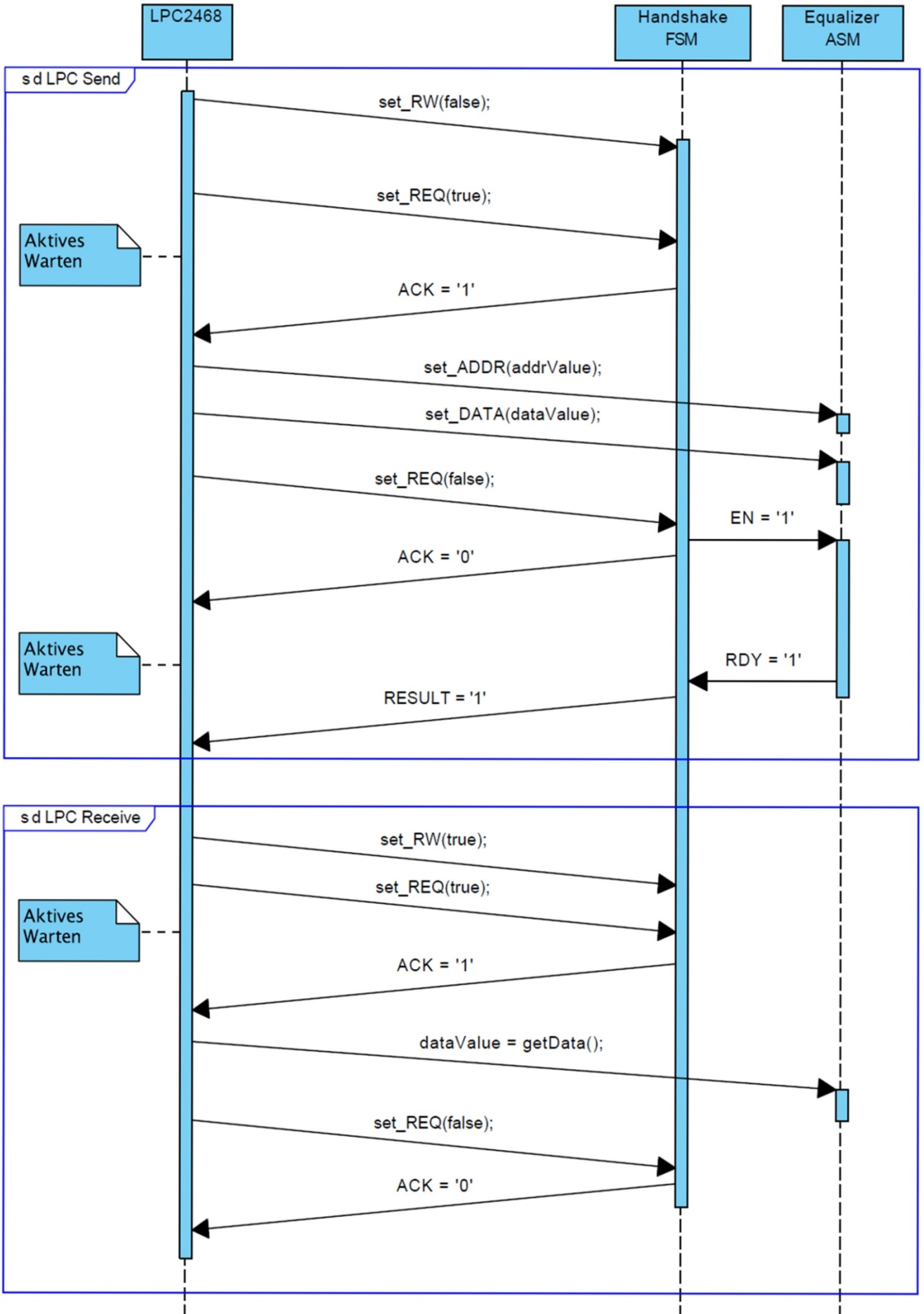


Asynchrone Kommunikation
µC - Beschleuniger

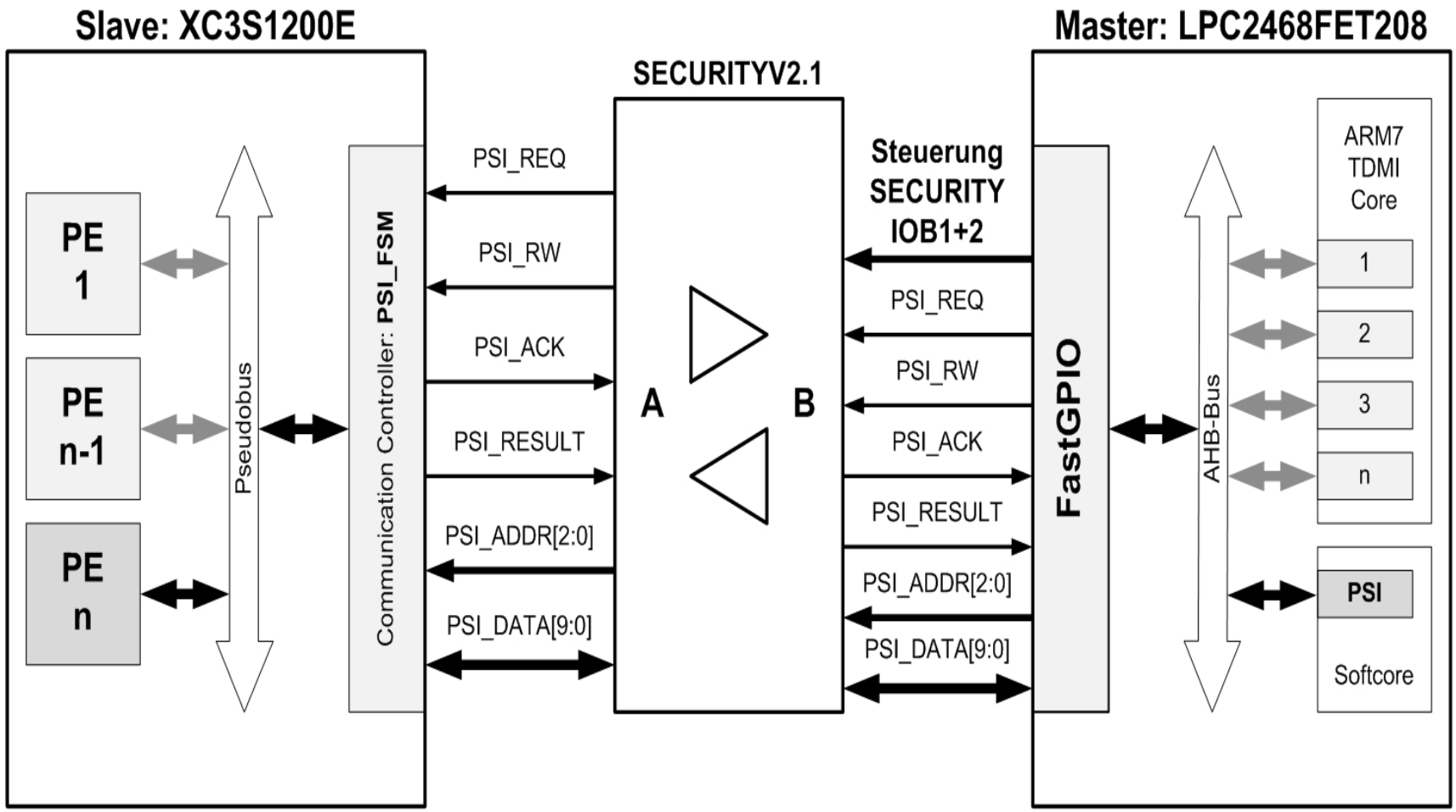


Handshake FSM

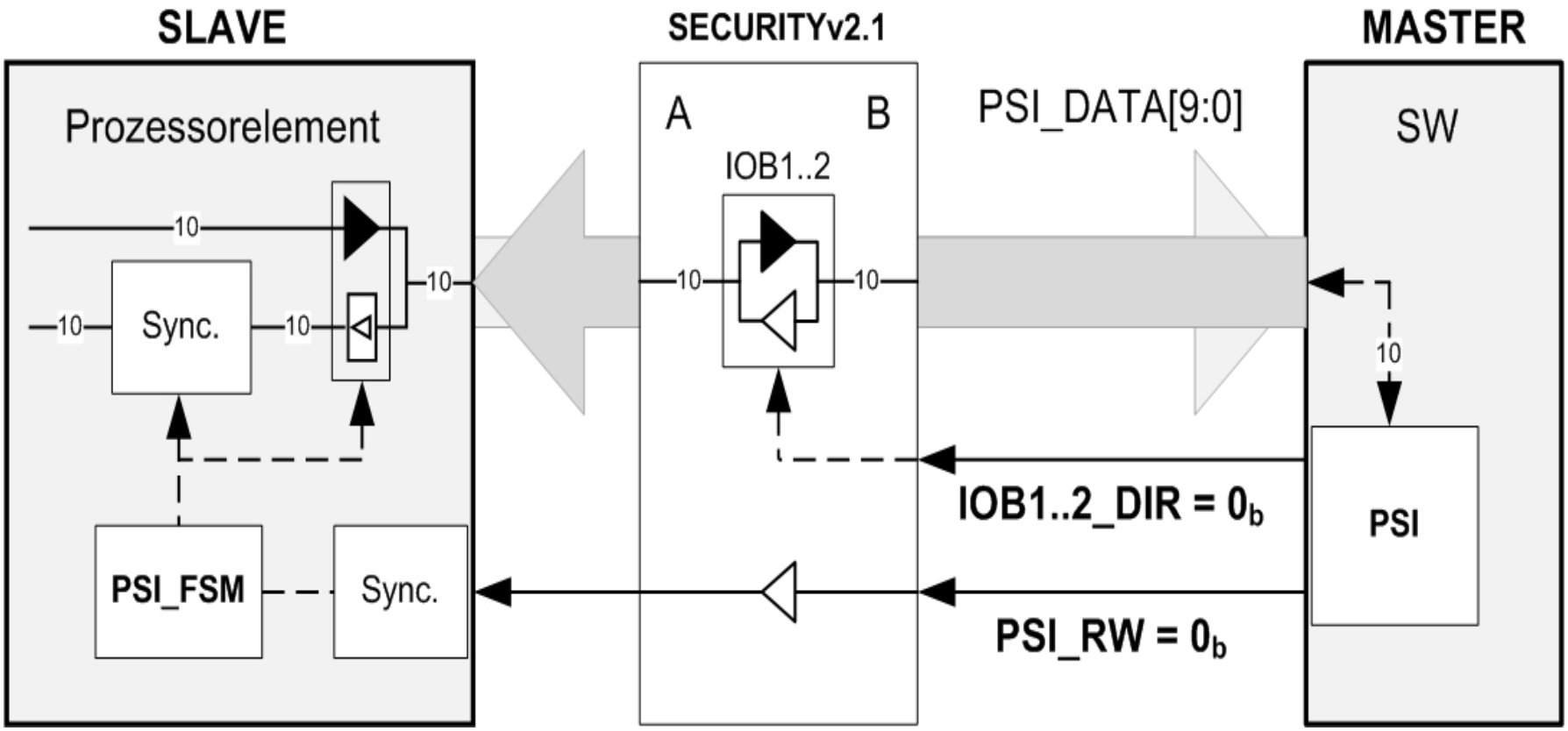



LPC2468
Handshake FSM
Equalizer ASM
s d LPC Send
set_RW(false);
set_REQ(true);
Aktives Warten
ACK = '1'
set_ADDR(addrValue);
set_DATA(dataValue);
set_REQ(false);
EN = '1'
ACK = '0'
Aktives Warten
RDY = '1'
RESULT = '1'
s d LPC Receive
set_RW(true);
set_REQ(true);
Aktives Warten
ACK = '1'
dataValue = getData();
set_REQ(false);
ACK = '0'

Security Board Interface

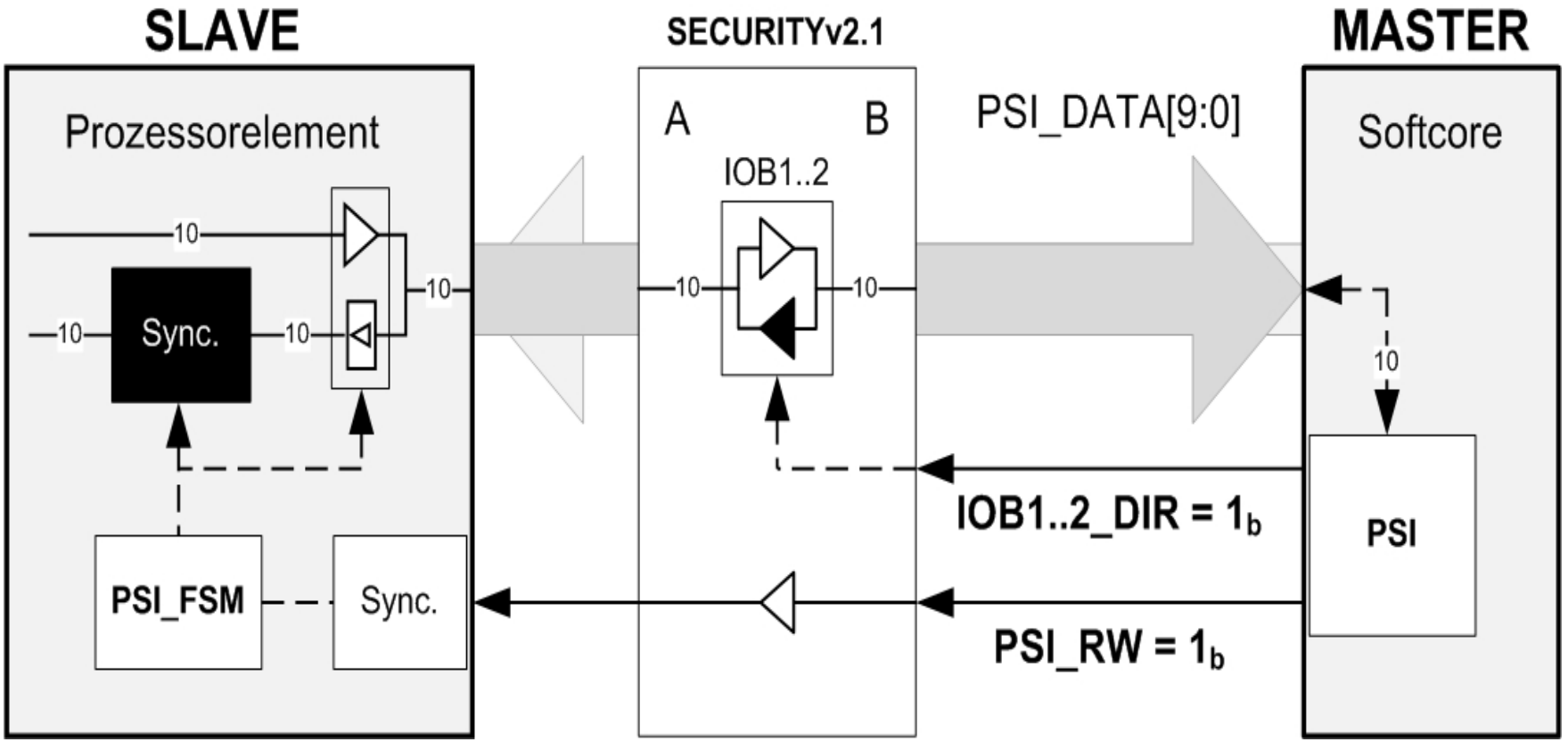


Transfersteuerung µC - FPGA



Schreibvorgang: B -> A

Transfersteuerung FPGA - µC



Lesevorgang: A -> B

Write – Sequenz in ISR

```
// Macro-Definitionen dazu in hwconfig.h
#include <stdint.h>

int32_t outdata;

int32_t *indata;

…

                        /** LPC-2468 Write to FPGA **/
PSI_ENABLE_WRITE;       /* Datenflussrichtung: B -> A */
PSI_SET_REQ;            /* 1. Phase: Anforderung */
PSI_WAIT_ACK_SET;       /* 2. Phase: Bestätigung */
PSI_WRITE_DATA( outdata ); /* 10 Bit Datum schreiben*/
PSI_CLR_REQ;            /* 3. Phase */
PSI_WAIT_ACK_CLR;       /* 4. Phase; ACK = '0' Polling */

…
```

Read – Sequenz in ISR

```
                              /** LPC-2468 Read from FPGA **/
 PSI_WAIT_RESULT_SET;         /* Ergebnisse verfügbar ?*/

 PSI_ENABLE_READ;             /* Datenflussrichtung: A -> B */
 PSI_SET_REQ;                 /* 1. Phase: FPGA schreibt */
 PSI_WAIT_ACK_SET;            /* 2. Phase */
 PSI_READ_DATA( *indata );    /* 10 Bit Datum lesen */
 PSI_CLR_REQ;                 /* 3. Phase: Lesebestätigung */
 PSI_WAIT_ACK_CLR;            /* 4. Phase: Lesesequenz beendet */
/* Datenausgabe an DAC und PWM */
```

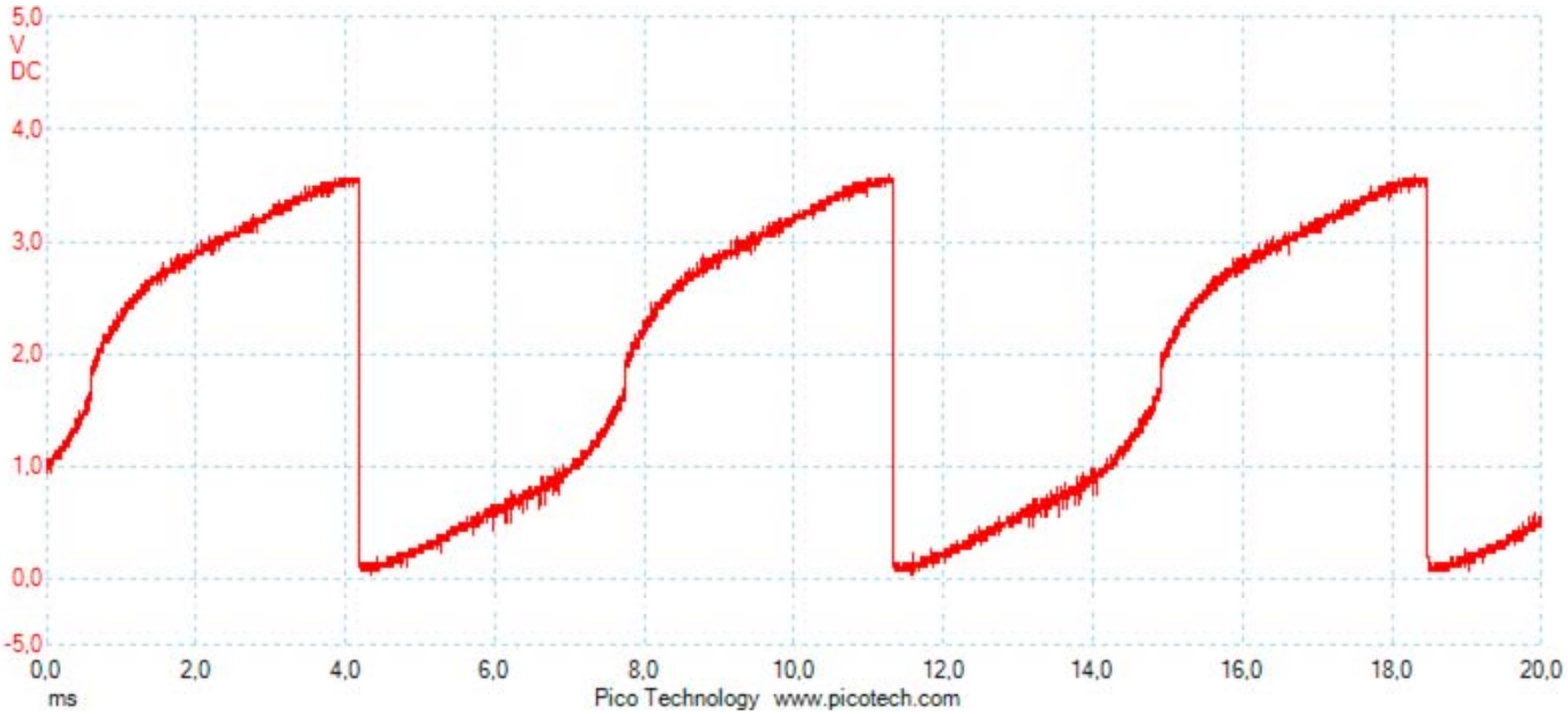
Pin-Belegung

Daten-/Steuersignale				Development Boards						
Name	Typ	Anmerkung	Transfer	TI-LPC	SECURITYv2.1			NEXYS2		
				uC Port[Pin]	Bustreiber	Connector Typ	Connector Pin	FPGA Signal	FPGA Pin	Connector FX2-100 Port-Pin
PSI_REQ	S	Auftrag	unidirektional	P1[12]	IOB4	X2	24	IOB4<3>	F11	J1A-34
PSI_ACK	S	Auftragsbestätigung	unidirektional	P1[0]	IOB3	X2	38	IOB3<0>	G09	J1A-23
PSI_RW	S	Schreiben/Lesen	unidirektional	P1[11]	IOB4	X2	23	IOB4<4>	E12	J1A-35
PSI_RESULT	S	Auftragsfertigstellung	unidirektional	P1[1]	IOB3	X2	36	IOB3<1>	F09	J1A-24
PSI_ADDR[0]	S	Adresse des Prozessorelements	unidirektional	P1[15]	IOB4	X2	33	IOB4<0>	B11	JA1-31
PSI_ADDR[1]				P1[14]			34	IOB4<1>	C11	JA1-32
PSI_ADDR[2]				P1[13]			29	IOB4<2>	E11	JA1-33
PSI_DATA[0]	D	Datum	bidirektional	P0[5]	IOB1	X3	36	IOB1<0>	A4	J1A-07
PSI_DATA[1]				P1[10]		X2	26	IOB1<1>	C3	J1A-08
PSI_DATA[2]				P0[13]			5	IOB1<2>	C4	J1A-09
PSI_DATA[3]				P0[14]			7	IOB1<3>	B6	J1A-10
PSI_DATA[4]				P0[19]		X3	15	IOB1<4>	D5	J1A-11
PSI_DATA[5]				P0[20]			14	IOB1<5>	C5	J1A-12
PSI_DATA[6]				P0[21]		X2	19	IOB1<6>	F7	J1A-13
PSI_DATA[7]				P0[22]			18	IOB1<7>	E7	J1A-14
PSI_DATA[8]				P0[29]	IOB2		6	IOB2<0>	A6	J1A-15
PSI_DATA[9]				P0[30]			15	IOB2<1>	C7	J1A-16
IOB1_DIR	S	IOB1 -> PSI_DATA[7:0]	unidirektional	P2[3]	IOB1	X3	32			
IOB2_DIR	S	IOB2 -> PSI_DATA[9,8]	unidirektional	P2[4]	IOB2	X3	33			

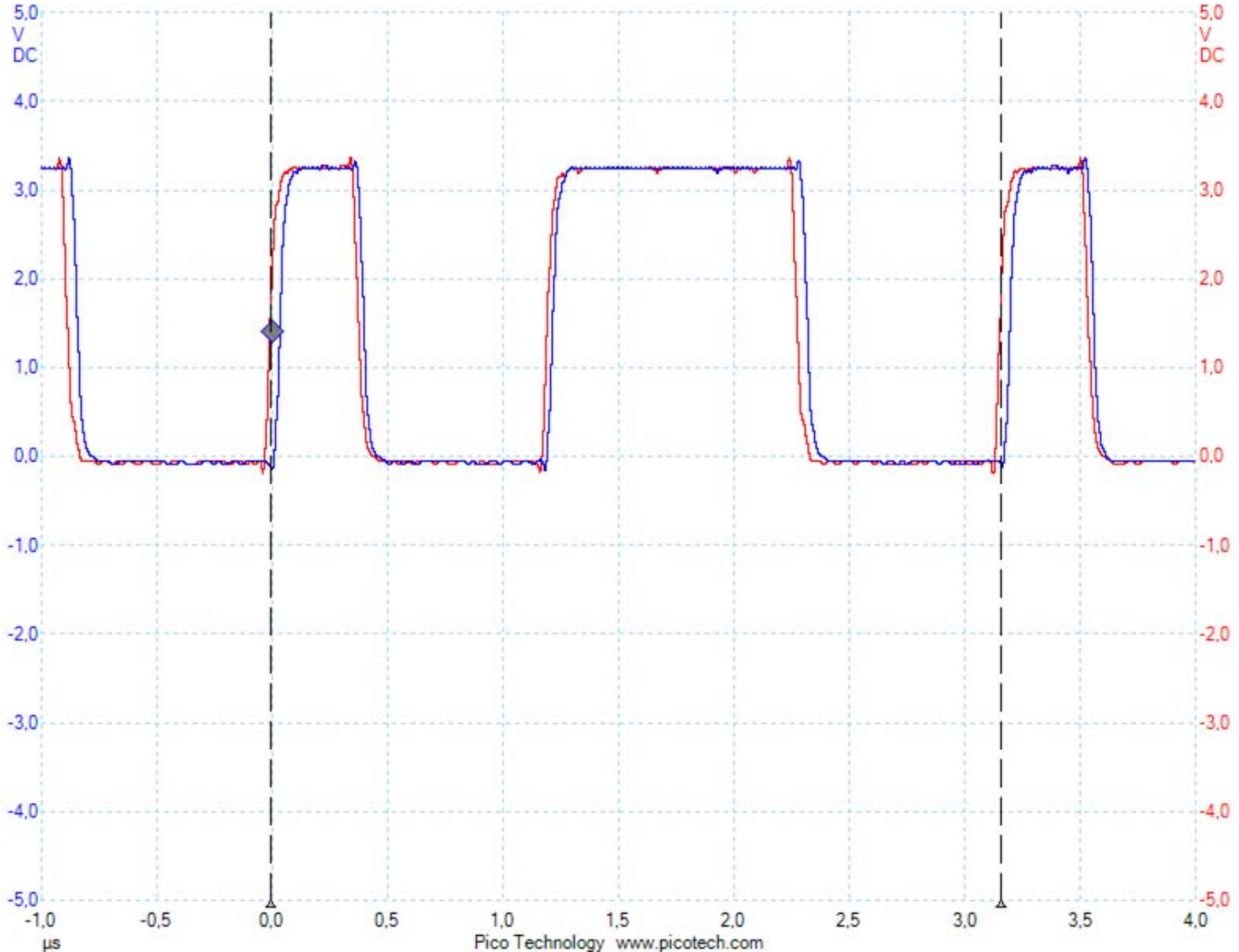
Macros in hwconfig.h

```
//Abfragen von Statusleitungen
#define PSI_ACK_PIN_PORT1     IOB3_DATA0_PIN_PORT1 // P1.0
#define PSI_WAIT_ACK_CLR
          while( (FIO1PIN & (1<<PSI_ACK_PIN_PORT1)) != 0){}
#define PSI_WAIT_ACK_SET
         while( (FIO1PIN & (1<<PSI_ACK_PIN_PORT1)) == 0){}

#define PSI_RESULT_PIN_PORT1  IOB3_DATA1_PIN_PORT1 //P1.1
#define PSI_WAIT_RESULT_SET
       while( (FIO1PIN & (1<<PSI_RESULT_PIN_PORT1)) == 0){}

//Setzen der Steuerleitungen
#define PSI_REQ_PIN_PORT1     IOB4_DATA3_PIN_PORT1 //P1.12
#define PSI_CLR_REQ        FIO1CLR = (1<<PSI_REQ_PIN_PORT1)
#define PSI_SET_REQ        FIO1SET = (1<<PSI_REQ_PIN_PORT1)
```

DAC-Wurzel-Kennlinie



REQ(rot) – ACK(blau)



REC(rot) – RESULT(blau)