



Modul 2: Projekt

Profibusmonitor

WS2012

Harndt, Martin

MODUL2_PROJEKT_PROFIBUSMONITOR_WS2012_MH.doc

07.05.2013 21:18:00



Inhalt

1	Aufgabenstellung	1-1
2	Konzept	2-1
3	Profibus	3-1
3.1	Elektrische Signalübertragung	3-1
3.2	Steckerbelegung	3-2
3.3	UART-Codierung.....	3-2
3.4	Telegrammtypen und Telegrammaufbau	3-3
4	Projektaufbau	4-1
5	Interface zur Signalumwandlung	5-1
5.1	Schaltplan	5-2
6	Stromversorgung	6-1
7	Aufbau Profibusmonitor.....	7-1
8	VHDL Funktionsmodule.....	8-1
8.1	Modul InAB_INPUT.....	8-1
8.1.1	Impulsdiagramm.....	8-4
8.1.2	Zählerzeitpunkte und Berechnung	8-5
8.1.3	Verhaltensbeschreibung	8-6
8.1.4	Belegung der Ausgangsvariablen.....	8-7
8.1.5	Auswahl Automatentyp.....	8-8
8.1.6	Liste der Prozesse.....	8-8
8.1.7	VHDL-Programm.....	8-9
8.1.8	Testumgebung	8-22
8.1.9	UCF-Datei	8-23
8.1.10	Steuerungsbelegung Testumgebung.....	8-24
8.1.11	Zuordnung Zustand / Anzeige.....	8-24
8.2	Modul BIT_REGISTER.....	8-25
8.2.1	Verhaltensbeschreibung	8-26
8.2.2	Belegung Ausgangsvariablen.....	8-27
8.2.3	Auswahl Automatentyp.....	8-27
8.2.4	Liste der Prozesse.....	8-27
8.2.5	VHDL-Programm.....	8-28
8.2.6	Testumgebung	8-36
8.2.7	UCF-Datei	8-37
8.2.8	Steuerungsbelegung Testumgebung.....	8-38
8.2.9	Zuordnung Zustand / Anzeige.....	8-38
8.3	Modul TELEGRAM_CHECK	8-39
8.3.1	Verhaltensbeschreibung	8-41
8.3.2	Belegung Ausgangsvariablen.....	8-42
8.3.3	Auswahl Automatentyp.....	8-42
8.3.4	Liste der Prozesse.....	8-42
8.3.5	VHDL-Programm.....	8-43
8.3.6	Testumgebung	8-54
8.3.7	UCF-Datei	8-55
8.3.8	Steuerungsbelegung Testumgebung.....	8-56
8.3.9	Zuordnung Zustand / Anzeige.....	8-56
8.4	Modul RS232_TX	8-57
8.4.1	Impulsdiagramm.....	8-58



8.4.2	Zählerzeitpunkt und Berechnung	8-59
8.4.3	Verhaltensbeschreibung	8-60
8.4.4	Belegung Ausgangsvariablen.....	8-61
8.4.5	Auswahl Automatentyp.....	8-61
8.4.6	Liste der Prozesse.....	8-61
8.4.7	VHDL-Programm.....	8-62
8.4.8	Testumgebung	8-67
8.4.9	UCF-Datei	8-68
8.4.10	Steuerungsbelegung Testumgebung.....	8-69
8.4.11	Zuordnung Zustand / Anzeige.....	8-69
9	PROFIBUS_MONITOR	9-1
9.1	Testumgebung	9-1
9.2	UCF-Datei	9-1
9.3	Steuerungsbelegung Testumgebung.....	9-1
9.4	Zuordnung Zustand / Anzeige.....	9-1
10	Ausblick zukünftige Entwicklung	10-1
10.1	SRAM_25MHZ_255_BYTE	10-2
10.2	Lesen und Schreiben	10-3
10.3	Verhaltensbeschreibung	10-5
10.4	Belebung Ausgangsvariablen.....	10-6
10.5	Auswahl Automatentyp.....	10-6
10.6	Liste der Prozesse.....	10-6
10.7	VHDL-Programm.....	10-7
10.8	Testumgebung	10-13
10.9	UCF-Datei	10-14
10.10	Steuerungsbelegung Testumgebung.....	10-15
10.11	Zuordnung Zustand / Anzeige.....	10-15
11	Anhang	11-1
11.1	VHDL-Programm BIT_REGISTER aus Testumgebung	11-1
11.2	VHDL-Programm RS232_TX aus Testumgebung	11-9
12	Literatur / Web-Seiten	12-15
12.1	Verzeichnis Bilder.....	12-15
12.2	Verzeichnis Tabellen.....	12-16
12.3	Verzeichnis Dateien	12-16
12.4	Quelle Bilder	12-16
12.5	Verzeichnis Abkürzungen.....	12-17
12.6	Bereitgestellte Bauteile	12-18



1 Aufgabenstellung

Entwurf und Realisierung eines Profibusmonitors für einen FPGA Schaltkreis.

Der Profibusmonitor hört die auf dem Profibus übertragenen Signale mit und wandelt sie für eine weitere Verarbeitung und Darstellung um. Die ausgegeben Daten können sowohl maschinell verarbeitet oder für einen Betrachter ausgegeben werden.

Als Realisierungsbasis ist das Spartan 3 Development Kit von Digilent mit einem Xilinx FPGA vorgegeben.

2 Konzept

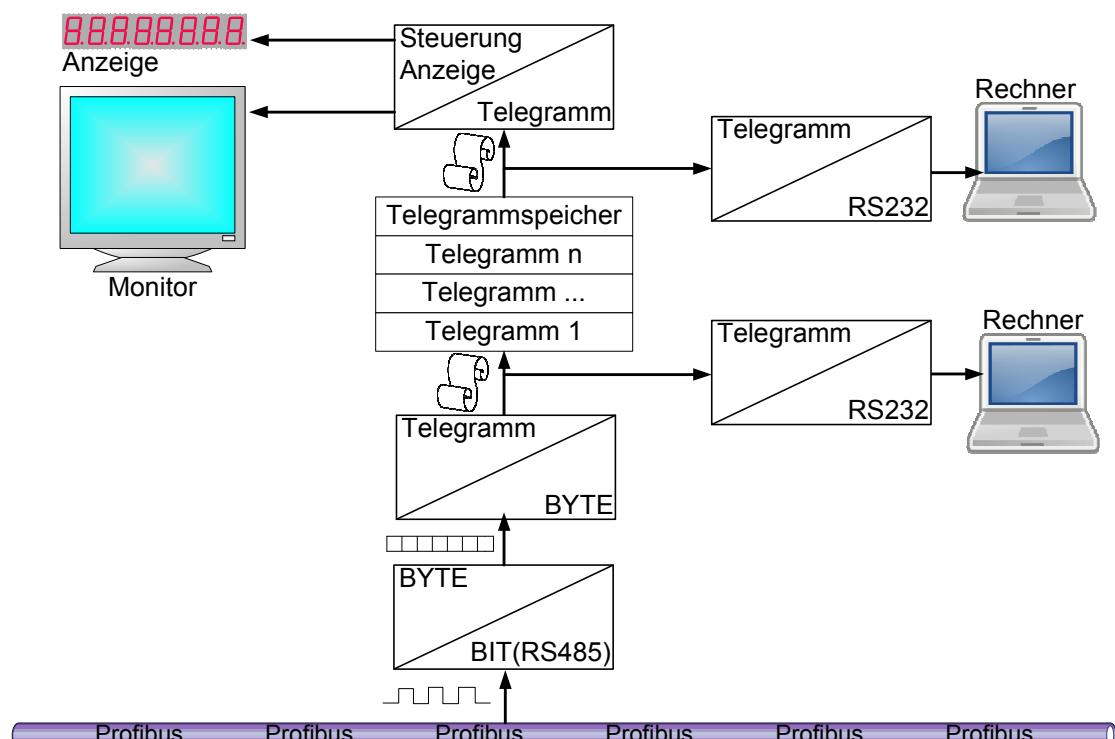


Abbildung 2-1: Konzept Profibusmonitor
/7SEG/ /LAPTOP/ /SCROLL/

Die auf dem Profibus via RS-485 übertragenen Signale werden vom Profibusmonitor in Bits umgewandelt, zu Bytes angeordnet, welche wiederum in Telegrammen zusammengefasst werden. Die Telegramm werden dann über die RS-232 Schnittsstelle an einen Rechner gesendet. Dort können sie ausgewertet oder mit Hilfe eines Terminalprogramms direkt angezeigt werden.

Eine weiterführendes Konzept sah die Speicherung der Telegramme im Profibusmonitor vor welche dann von einem Rechner via RS-232 ausgelesen werden können oder direkt auf eine Anzeige / Monitor angezeigt werden können.



3 Profibus

Der **Process Field Bus** ist ein Standard für einen Feldbus zur Maschinensteuerung in der industriellen Automatisierungstechnik. Den Profibusstandard gibt es seit 1991.

Es gibt drei Varianten des Profibusstandards:

- Profibus DP (Dezentrale Peripherie)
- Profibus PA (Prozess Automation)
- Profibus FMS (Fieldbus Message Specification)

Der Profibusstandard Profibus DP ist die gebräuchlichste Anwendungsform.

Die Variante Profibus PA ist nur in speziellen explosionsgefährdeten Bereichen dem Profibus DP vorziehen oder wenn ein Teilnehmer des Profibus über diesen auch mit Strom versorgt werden soll. Die maximale Datenübertragungsrate von Profibus PA (31,25 kbit/s) ist gegenüber dem Profibus DP (12 Mbit/s) stark reduziert.

Profibus FMS wurde vom Profibus DP abgelöst und ist im aktuellen Profibusstandard nicht mehr enthalten.

Der in diesem Dokument beschriebene Profibusmonitor wurde daher für die Variante Profibus DP entwickelt.

3.1 Elektrische Signalübertragung

Die Signalübertragung des Profibus erfolgt mittels des Standards EIA-485, auch als RS-485 bezeichnet.

Dabei werden zwei Datenleitungen verwendet. Datenleitung A ist invertiert gegenüber Datenleitung B.

Datenleitung A: (invertiert)

Datenleitung B: (Original)

Ist der Signalpegel logisch 1, dann hat:

- Ltg. B positives potential gegenüber Ltg. A
- B: 5V und A:0V

Ist der Signalpegel logisch 0, dann hat:

- Ltg. B negatives potential gegenüber Ltg. A
- B: 0V und A: 5V

Der Ruhezustand des Busses ist logisch 1.

Die Bitübertragung erfolgt mit der Leitungscodierung „Non-Return-to-Zero“ (NRZ).



3.2 Steckerbelegung

Es werden spezielle 9-polige D-Sub Profibus Stecker verwendet, welche einen zuschaltbaren Busabschluss enthalten.

Pin Nr.	Signal	Funktion	
1	Schirm	Schutzerde	nicht empfohlen
2	M24	Masse für 24V Spannung	nicht empfohlen
3	RxD/TxD-P	Datenleitung Plus (B-Leiter)	Pflicht
4	CNTR-P	Repeater Richtungskontrolle	Optional
5	DGND	Daten Masse	Pflicht
6	VP	+5V Speisung für Busabschluss	Pflicht
7	P24	+24V Speisung	nicht empfohlen
8	RxD/TxD-N	Datenleitung Minus (A-Leiter)	Pflicht
9	CNTR-N	Repeater Richtungskontrolle	Optional

Tabelle 3-1: Pinbelegung D-Sub Stecker für Profibus

Für die Speisung der Busabschlüsse müssen dem Profibus mindestens eine Speisung von 100 mA bei 5V zur Verfügung stehen.

Der Pin 1 wird nicht mehr für die Schutzerde verwendet. Diese Funktion übernimmt der Schirm, welcher am Stecker an die Schutzerde angeschlossen ist.

Die Benutzung der 24V Speisung wird nicht mehr empfohlen.

Im Rahmen dieses Projektes sind keine Repeater berücksichtigt worden. Es wird daher nur die Pflichtbelegung für den Profibus mit den Pins 3, 5, 6 und 8 genutzt.

3.3 UART-Codierung

Die Profibustelegramme werden in der UART (Universal Asynchronous Receiver Transmitter) Codierung bitweise übertragen und haben folgenden Aufbau.

Start	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Parität	Stop
Start = 0 ; Parität = gerade ; Stop = 1										

Abbildung 3-1: Aufbau UART-Codierung des Profibus

Übertragungsparameter:

- Ruhezustand ist logisch 1
- Die Übertragung beginnt mit einem SYN (logisch 1) von mindestens 33 Bit Dauer
- Nach dem Stopbit kann unmittelbar wieder ein Startbit folgen
- Kontrolle des Empfängers von Startbit, Stopbit und Paritätsbit



3.4 Telegrammtypen und Telegrammaufbau

Es gibt 5 Telegrammtypen die Anhang ihrer Startsequenz (SD: Start Delimiter) identifiziert werden können.

- SD1: Telegramm ohne Daten (6 Byte)
- SD2: Telegramm mit Daten variabler Länge (10 – 255 Byte)
- SD3: Telegramm mit Daten fester Länge (14 Byte)
- SD4: Tokentelegramm (3 Byte)
- SC: Kurzquittung (1 Byte)

SD1 Telegrammaufbau:

SD1				DA	SA	FC		FCS	ED
-----	--	--	--	----	----	----	--	-----	----

SD2 Telegrammaufbau:

SD2	LE	LER	SD2	DA	SA	FC	PDU	FCS	ED
-----	----	-----	-----	----	----	----	-----	-----	----

SD3 Telegrammaufbau:

SD3				DA	SA	FC	PDU	FCS	ED
-----	--	--	--	----	----	----	-----	-----	----

SD4 Telegrammaufbau:

SD4				DA	SA				
-----	--	--	--	----	----	--	--	--	--

SC Telegrammaufbau:

SC									
----	--	--	--	--	--	--	--	--	--

SD: Start Delimiter

DA: Destination Address

SA: Source Adress

FC: Funktion Code

FCS: Frame Check Sequence

ED: End Delimiter

LE: Length

LER: Length repeated

PDU: Protocol Data Unit

SC: Short Confirmation

Die Start Delimiter (SD), Short Confirmation (SC) und das End Delimiter (SD) haben feste Werte. Hier hexadezimal dargestellt:

Feld:	SD1	SD2	SD3	SD4	SC	ED
Wert:	10 _h	68 _h	A2 _h	DC _h	E5 _h	16 _h

Im Destination Address (DA) Feld können dezimale Werte von 0 bis 127 vorkommen.

Im Source Address (SA) Feld können dezimale Werte von 0 bis 126 vorkommen.

Die Felder Length (LE) und Length repeated (LER) enthalten jeweils den Wert der Bits der Felder DA, SA, FC und die PDU von dezimal 4 bis 249.

Für das Empfangen der Daten wurden nur die Felder SD1 bis SD4, SC, ED, LE und LER ausgewertet. Das ermöglicht eine exakte Bestimmung des Telegrammtyps, der Telegrammlänge und dem Ende des Telegramms.

4 Projektaufbau

Für Testzwecke wurde ein kleines Profibusnetzwerk aufgebaut. Es besteht aus einem PC mit einer SPS-Steuerungssoftware und einer Profibus Masterkarte und einem Gerät das als Profibus Slave fungiert.

An einer Verbindung dazwischen wurde dann über ein Interface und einem VGA-Kabel der Profibusmonitor angeschlossen. Die Telegramme laufen beim Profibus in einem Busnetzwerk, d.h. jeder Teilnehmer empfängt die gesamte Kommunikation auf dem Bus. Für den Projektaufbau genügt also ein kurzes Profibuskabel mit nur drei Anschlüssen.

Der Profibus Master sendet die Steuerungstelegramme an den Profibus Slave, welcher wiederum darauf reagiert und antwortet. Diese Kommunikation konnte dann auf dem Profibusmonitor mitgelesen werden.

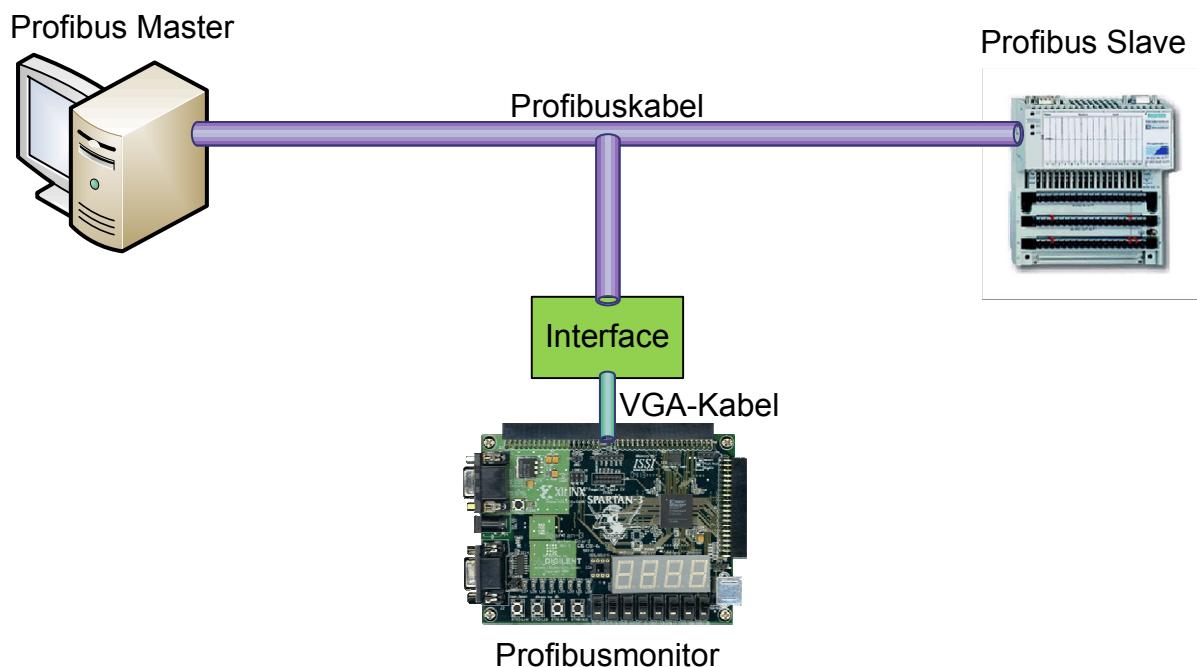


Abbildung 4-1: Aufbau der Profibusinfrastruktur
/SPARTAN3 /PROFI/



5 Interface zur Signalumwandlung

Die Verbindung vom Profibuskabel zum Sparten 3 Development Kit kann nicht direkt erfolgen. Es wurde ein Interface entwickelt mit der 9-poligen D-Sub Buchse für den Profibusstecker und einer 15-poligen VGA-Buchse für den Anschluss des Spartan 3 Development Kits. Gleichzeitig erfolgt die Umwandlung der RS-485 Signale des Profibus in ein binäres Signal. Dieses binäre Signal gelangt dann über ein VGA-Kabel mit Anschluss am Spartan 3 Development Kit in den FPGA zu weiteren Verarbeitung.

Gleichzeitig dient es der elektrischen Entkopplung des Profibusmonitors vom Profibus mit Hilfe eines Optokopplers und eines DC/DC-Wandlers. Die zwei getrennten Stromkreise, P5 und 2P5, werden separat mit Strom versorgt. Dadurch werden störende Einflüsse durch den Profibusmonitor und vom Profibus vermieden. Das Signal aus dem Optokoppler heraus wird mittels eines Operationsverstärkers verstärkt. Ein Schirm dient der Abschirmung von weiteren Störungen.

Das entwickelte Interface kann nur Profibussignale empfangen, nicht senden.

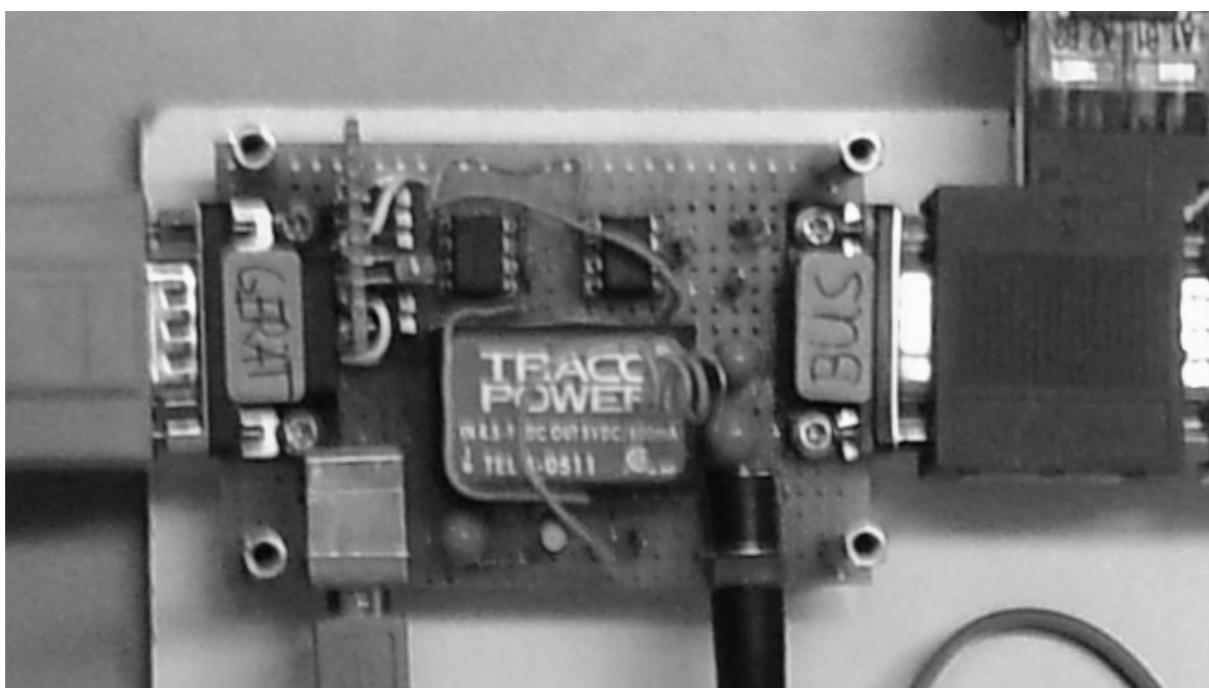
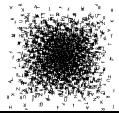


Abbildung 5-1: Interface zur Signalumwandlung



5.1 Schaltplan

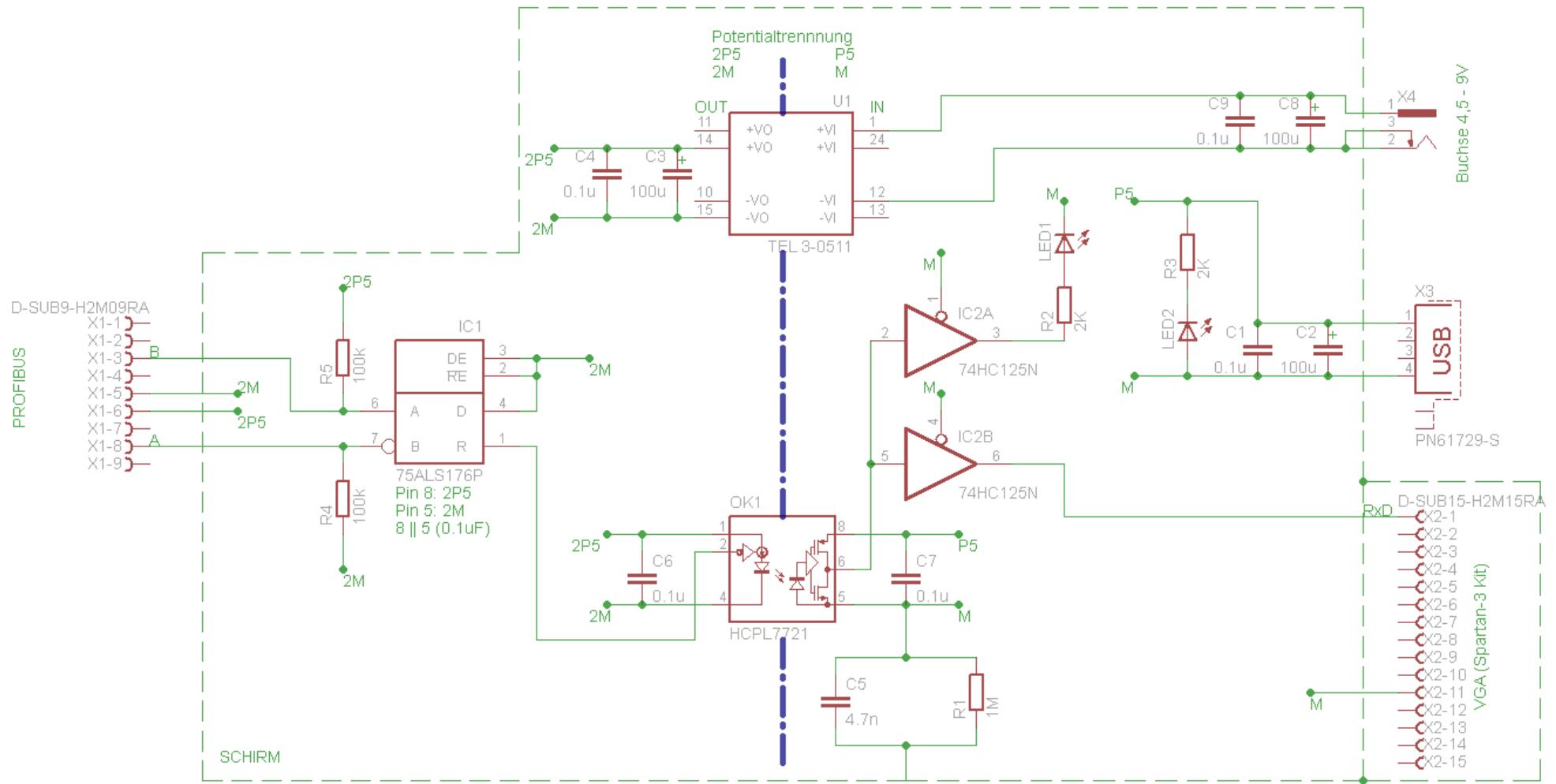


Abbildung 5-2: Schaltplan Interface zur Signalumwandlung



6 Stromversorgung

Die Stromversorgung des Spartan 3 Development Kits und des Interfaces wird von sechs Batterien gespeist und hat drei Anschlüsse. Zwei der Anschlüsse liefern über USB eine Spannung von 5 V und 600mA. Ein weiter Anschluss mit einem Hohlstecker wird direkt von den Batterien gespeist. Die 5V für die USB-Anschlüsse wurden mit einem DC/DC-Wandler realisiert. Eine separate Netzunabhängige Stromversorgung wurde nötig um Störungen des Projektaufbaus zu minimieren. Die im Labor verfügbaren, handelsüblichen Steckernetzteile genügten den Anforderungen daran nicht.

Der Aufbau der Stromversorgung ist nicht Bestandteil dieser Dokumentation. Sie wurde von einem Mitstudenten realisiert. Vielen Dank.



Abbildung 6-1: Stromversorgung

7 Aufbau Profibusmonitor

Der Profibusmonitor umfasst das Spartan 3 Development Kit, das Interface zur Signalamwandlung und die Stromversorgung. Diese Bestandteile wurden zusammen mit dem Profibusslave und dessen Netzteil auf einer Sperrholzplatte befestigt.

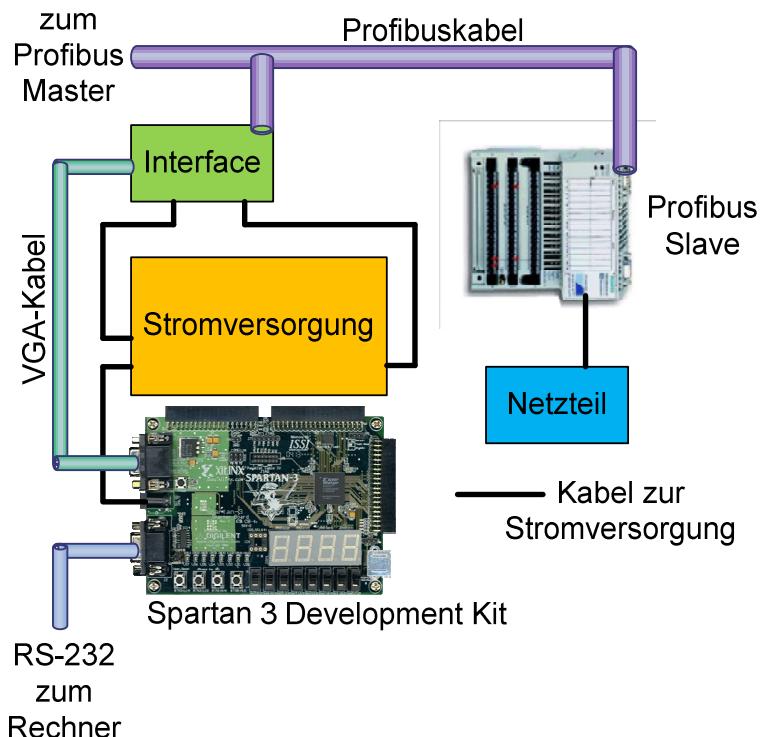


Abbildung 7-1: Profibusmonitor, schematische Darstellung, /SPARTAN3/ /PROFI/

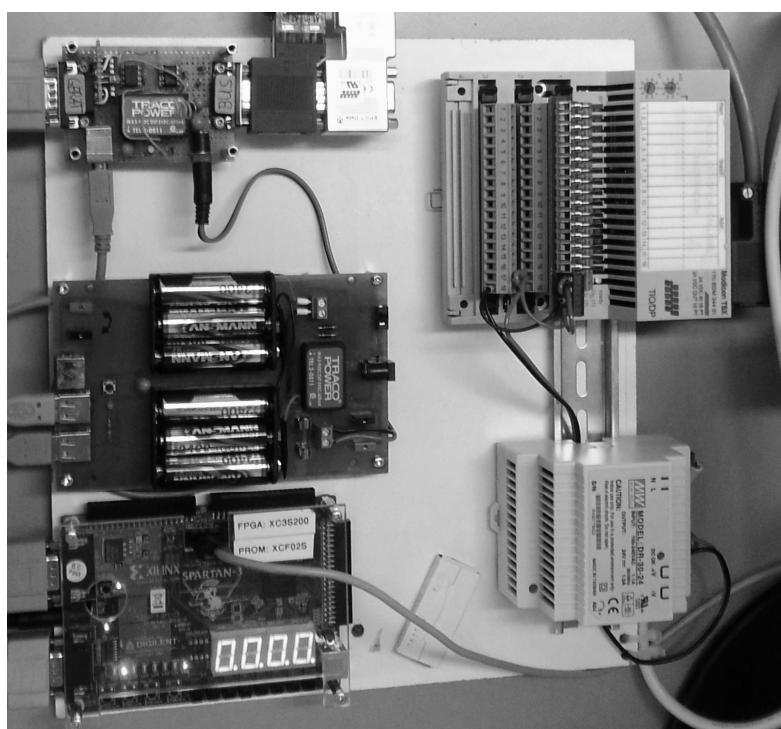


Abbildung 7-2: Profibusmonitor, Foto



8 VHDL Funktionsmodule

Die Erfassung des Signals, das Umwandeln in Bits und die Anordnung zu Telegrammen erfolgt in einzelnen Funktionsmodulen welche in VHDL programmiert und auf dem Spartan 3 Development Kit ausgeführt werden.

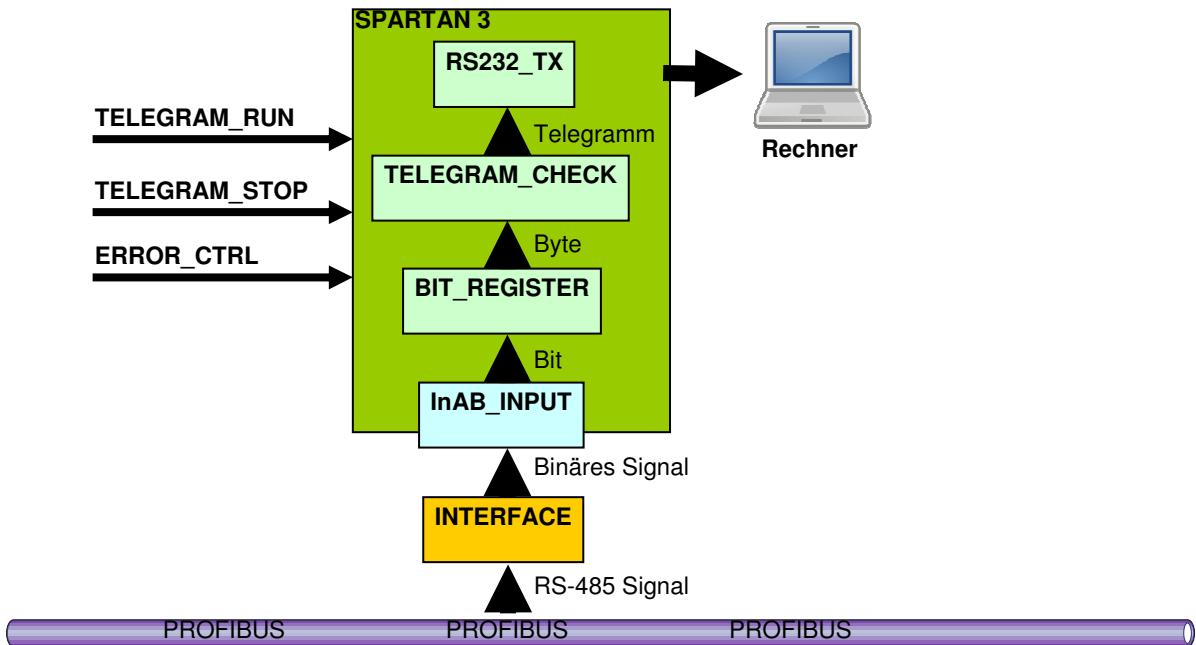


Abbildung 8-1: Funktionsmodule, Anordnung /LAPTOP/

8.1 Modul InAB_INPUT

Das binäre Signal vom Interface wird im Modul InAB_INPUT eingelesen. Mittels eines Zählers (COUNT) werden die Daten- und Paritätsbits vom Start-, Stoppbit getrennt.

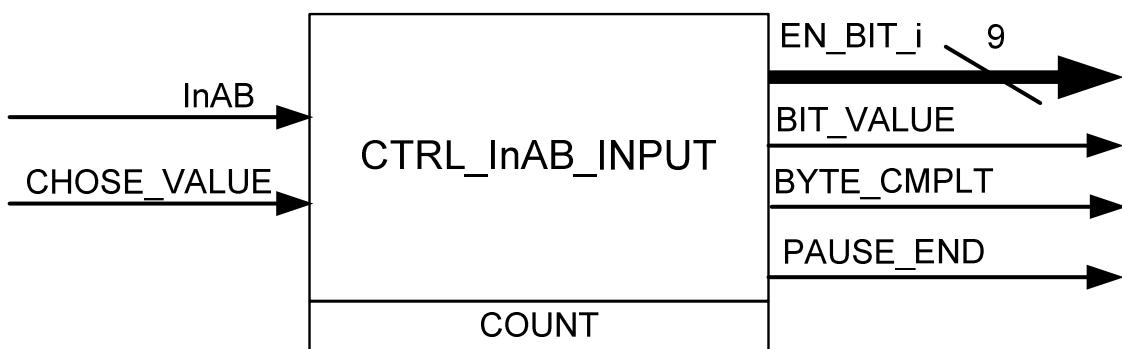
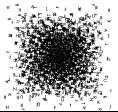


Abbildung 8-2: Wirkungsplan

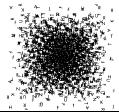


Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
InAB	BOOL	Eingang	1: Profibussignal ist 1
CHOSE_VALUE	BOOL	Eingang	1: Zählerwerte, Schrittbetrieb; 0: normale Zählerwerte
EN_BIT_i	VECTOR, 9 Bit	Ausgang	Einschalten Bit 0-8
BIT_VALUE	BOOL	Ausgang	1: Bitwert ist 1
BYTE_CMPLT	BOOL	Ausgang	1: Byte komplett empfangen
PAUSE_END	BOOL	Ausgang	1: SYN von 33 Bit beendet
COUNT_L	VECTOR, 20 Bit	intern	langer Zähler für Erkennung SYN
COUNT_S	VECTOR, 16 Bit	intern	kurzer Zähler für Erkennung der Bits
EN_BIT_0	BOOL	intern	1: Einschalten Bit 0
EN_BIT_1	BOOL	intern	1: Einschalten Bit 1
EN_BIT_2	BOOL	intern	1: Einschalten Bit 2
EN_BIT_3	BOOL	intern	1: Einschalten Bit 3
EN_BIT_4	BOOL	intern	1: Einschalten Bit 4
EN_BIT_5	BOOL	intern	1: Einschalten Bit 5
EN_BIT_6	BOOL	intern	1: Einschalten Bit 6
EN_BIT_7	BOOL	intern	1: Einschalten Bit 7
EN_BIT_8	BOOL	intern	1: Einschalten Bit 8 (Paritätsbit)
CNTS30	VECTOR, 20 Bit	intern	Zählerwert für t_{SYN30}
CNTT01	VECTOR, 16 Bit	intern	Zählerwert für t_1
CNTT02	VECTOR, 16 Bit	intern	Zählerwert für t_2
CNTT03	VECTOR, 16 Bit	intern	Zählerwert für t_3
CNTT04	VECTOR, 16 Bit	intern	Zählerwert für t_4
CNTT05	VECTOR, 16 Bit	intern	Zählerwert für t_5
CNTT06	VECTOR, 16 Bit	intern	Zählerwert für t_6
CNTT07	VECTOR, 16 Bit	intern	Zählerwert für t_7
CNTT08	VECTOR, 16 Bit	intern	Zählerwert für t_8
CNTT09	VECTOR, 16 Bit	intern	Zählerwert für t_9
CNTT10	VECTOR, 16 Bit	intern	Zählerwert für t_{10}
CNTT11	VECTOR, 16 Bit	intern	Zählerwert für t_{11}
CNTT12	VECTOR, 16 Bit	intern	Zählerwert für t_{12}
CNTT13	VECTOR, 16 Bit	intern	Zählerwert für t_{13}
long_CNTS30	CONSTANT	intern	Wert: 2625A _h
long_CNTT01	CONSTANT	intern	Wert: 0A2C _h
long_CNTT02	CONSTANT	intern	Wert: 1E84 _h
long_CNTT03	CONSTANT	intern	Wert: 32DC _h
long_CNTT04	CONSTANT	intern	Wert: 4735 _h
long_CNTT05	CONSTANT	intern	Wert: 5B8B _h
long_CNTT06	CONSTANT	intern	Wert: 6FE4 _h
long_CNTT07	CONSTANT	intern	Wert: 8441 _h
long_CNTT08	CONSTANT	intern	Wert: 9872 _h
long_CNTT09	CONSTANT	intern	Wert: ACEE _h
long_CNTT10	CONSTANT	intern	Wert: C147 _h
long_CNTT11	CONSTANT	intern	Wert: D59F _h
long_CNTT12	CONSTANT	intern	Wert: D9B1 _h
long_CNTT13	CONSTANT	intern	Wert: E5E6 _h
short_CNTS30	CONSTANT	intern	Wert: 0000A _h



Variablenname	Datentyp	Variablentyp	Informationen / Anweisungen
short_CNTT01	CONSTANT	intern	Wert: 0003 _h
short_CNTT02	CONSTANT	intern	Wert: 0006 _h
short_CNTT03	CONSTANT	intern	Wert: 0009 _h
short_CNTT04	CONSTANT	intern	Wert: 000C _h
short_CNTT05	CONSTANT	intern	Wert: 000F _h
short_CNTT06	CONSTANT	intern	Wert: 0012 _h
short_CNTT07	CONSTANT	intern	Wert: 0015 _h
short_CNTT08	CONSTANT	intern	Wert: 0018 _h
short_CNTT09	CONSTANT	intern	Wert: 001B _h
short_CNTT10	CONSTANT	intern	Wert: 001E _h
short_CNTT11	CONSTANT	intern	Wert: 0021 _h
short_CNTT12	CONSTANT	intern	Wert: 0024 _h
short_CNTT13	CONSTANT	intern	Wert: 002A _h
ST_CTRL	ENUM		Zustandsvariable (ST_CTRL_00, ST_CTRL_01, ST_CTRL_02, ST_CTRL_03, ST_CTRL_04, ST_CTRL_05, ST_CTRL_06, ST_CTRL_07, ST_CTRL_08, ST_CTRL_09, ST_CTRL_0A, ST_CTRL_0B, ST_CTRL_0C, ST_CTRL_0E, ST_CTRL_0F)

Tabelle 8-1: Variablendefinition InAB_INPUT



8.1.1 Impulsdiagramm

Das binäre Eingangssignal welches vom Interface kommend eingelesen wird, entspricht der UART Codierung des Profibus und bildet ein Telegramm. Es gibt 8 Datenbits (Bit 1 bis Bit 8), ein Startbit (Bit 0), ein Paritätsbit (Bit 9) und ein Stopppbit (Bit 10). Vor dem Start einer Übertragung werden mindestens 33 SYN-Bits mit dem logischen Wert 1 gesendet, anschließend ein Startbit welches immer logisch 0 ist. Es folgen die 8 Datenbits, gefolgt von einem Paritätsbit (gerade Parität) und ein Stopppbit das immer logisch 1 ist. Anschließend beginnt ein neues Telegramm mit einem Startbit oder es wird nichts gesendet wobei der Zustand logisch 1 angenommen wird. Die **Abbildung 8-3** zeigt das Telegramm als Impulsdiagramm. Zusätzlich sind die Zählerzeitpunkte (t_{SYN30} und t_1 bis t_{13}) für die Signalabtastung der einzelnen Bits angegeben. Die dargestellten logischen Werte 0 und 1 für Bit 1 bis Bit 8 dienen nur als Beispiel.

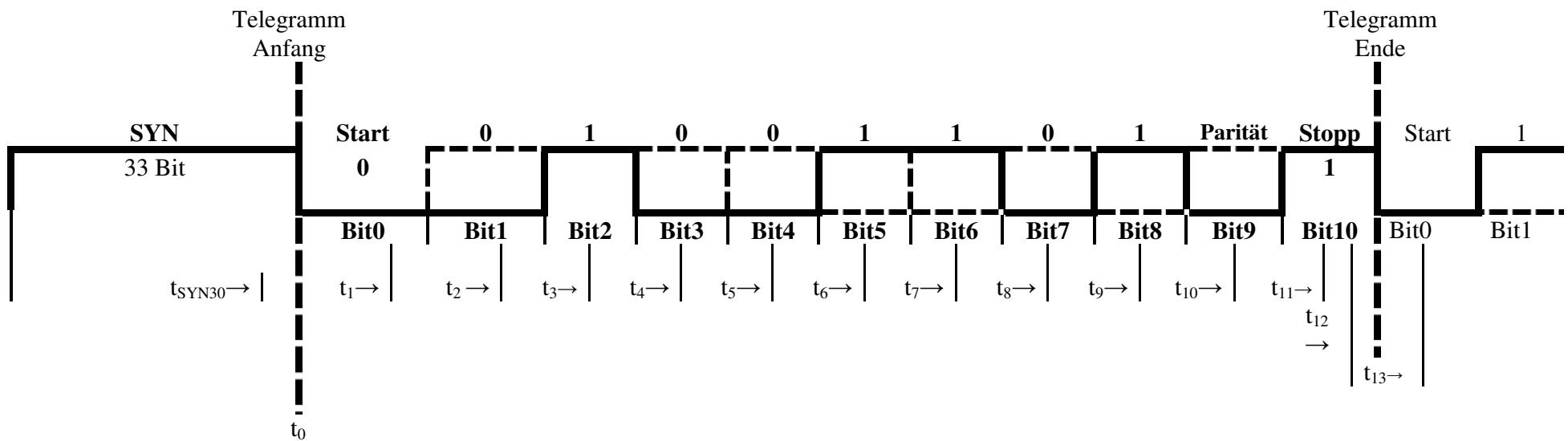
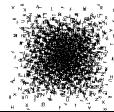


Abbildung 8-3: Impulsdiagramm Eingangssignal InAB_INPUT



8.1.2 Zählerzeitpunkte und Berechnung

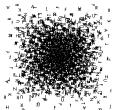
Für die Umwandlung des UART-Signals in ein Telegramm muss das Signal zu einem bestimmten Zeitpunkt abgetastet bzw. erfasst werden. Dazu wird der Systemtakt des Spartan 3 Development Kits benutzt. Der Systemtakt ist 50 MHz (50.000.000 Hz).

Jedes Bit ist dauert eine bestimmte Anzahl von Takten des Systemtakts an. Die Berechnung erfolgt mit der Formel: (n Bit * Systemtakt) / Baudrate
Für t_{SYN30} ist n = 30 für t_1 ist n = 0,5 usw. Die Baudrate beträgt 9600, fest eingestellt am Profibus Master. Das Ergebnis wird abgerundet.

Der Zählerstand lang sind die Werte für den Profibus. Der Zählerstand kurz sind die Werte für den Schrittbetrieb.

Zählerzeitpunkte	Berechnung	Zählerstand	Zählerstand	Zählerstand	Zählerstand	Variablenname
		lang, dezimal	lang, hexadezimal	kurz, dezimal	kurz, hexadezimal	
t_{SYN30}	$(300 * \text{Systemtakt}) / 96000$	156250	2625A	10	A	CNTS30
t_0	\div	0	0	0	0	
t_1	$(5 * \text{Systemtakt}) / 96000$	2604	0A2C	3	3	CNTT01
t_2	$(15 * \text{Systemtakt}) / 96000$	7812	1E84	6	6	CNTT02
t_3	$(25 * \text{Systemtakt}) / 96000$	13020	32DC	9	9	CNTT03
t_4	$(35 * \text{Systemtakt}) / 96000$	18229	4735	12	C	CNTT04
t_5	$(45 * \text{Systemtakt}) / 96000$	23435	5B8B	15	F	CNTT05
t_6	$(55 * \text{Systemtakt}) / 96000$	28644	6FE4	18	12	CNTT06
t_7	$(65 * \text{Systemtakt}) / 96000$	33854	8441	21	15	CNTT07
t_8	$(75 * \text{Systemtakt}) / 96000$	39062	9872	24	18	CNTT08
t_9	$(85 * \text{Systemtakt}) / 96000$	44270	ACEE	27	1B	CNTT09
t_{10}	$(95 * \text{Systemtakt}) / 96000$	49479	C147	30	1E	CNTT10
t_{11}	$(105 * \text{Systemtakt}) / 96000$	54687	D59F	33	21	CNTT11
t_{12}	$(107 * \text{Systemtakt}) / 96000$	55729	D9B1	36	24	CNTT12
t_{13}	$(113 * \text{Systemtakt}) / 96000$	58854	E5E6	42	2A	CNTT13

Tabelle 8-2: Zählerzeitpunkte für Abtastung InAB_INPUT



8.1.3 Verhaltensbeschreibung

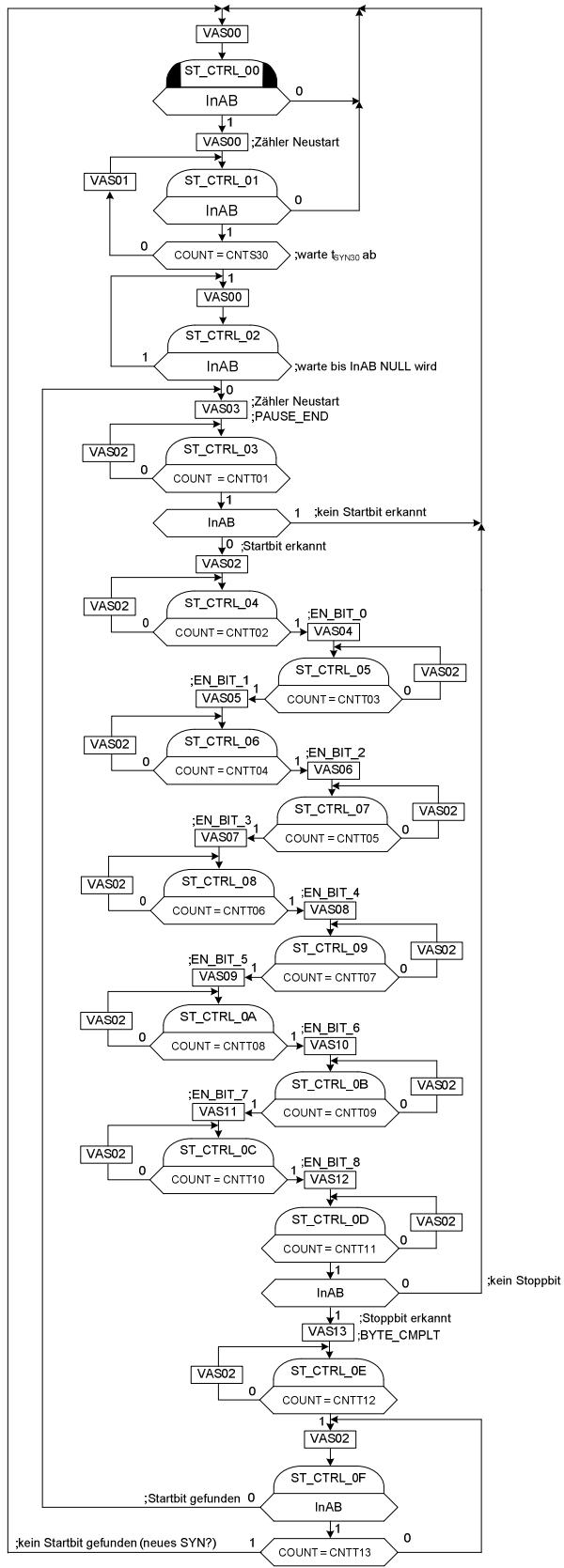
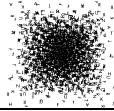


Abbildung 8-4: Programmablaufgraph InAB_INPUT



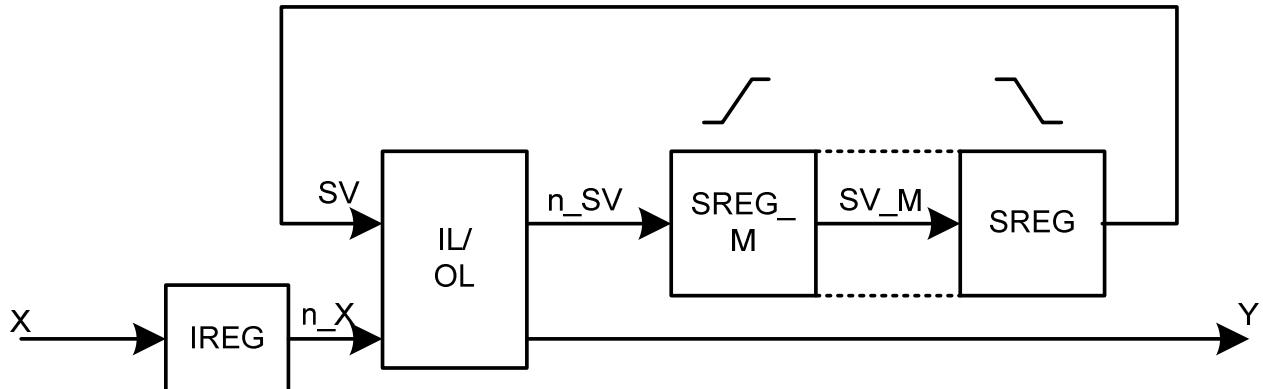
8.1.4 Belegung der Ausgangsvariablen

Bezeichner	n_COUNT_L	n_COUNT_S	EN_BIT_0	EN_BIT_1	EN_BIT_2	EN_BIT_3	EN_BIT_4	EN_BIT_5	EN_BIT_6	EN_BIT_7	EN_BIT_8	BIT_VALUE	BYTE_CMPLT
VAS00	:= 0	:=0	0	0	0	0	0	0	0	0	0	0	0
VAS01	:=COUNT_L+1	:=0	0	0	0	0	0	0	0	0	0	0	0
VAS02	:=0	:= COUNT_S+1	0	0	0	0	0	0	0	0	0	0	0
VAS03	:=0	:=COUNT_S+1	0	0	0	0	0	0	0	0	0	0	0
VAS04	:=0	:=COUNT_S+1	1	0	0	0	0	0	0	0	0	:=InAB	0
VAS05	:=0	:=COUNT_S+1	0	1	0	0	0	0	0	0	0	:=InAB	0
VAS06	:=0	:=COUNT_S+1	0	0	1	0	0	0	0	0	0	:=InAB	0
VAS07	:=0	:=COUNT_S+1	0	0	0	1	0	0	0	0	0	:=InAB	0
VAS08	:=0	:=COUNT_S+1	0	0	0	0	1	0	0	0	0	:=InAB	0
VAS09	:=0	:=COUNT_S+1	0	0	0	0	0	1	0	0	0	:=InAB	0
VAS10	:=0	:=COUNT_S+1	0	0	0	0	0	0	1	0	0	:=InAB	0
VAS11	:=0	:=COUNT_S+1	0	0	0	0	0	0	0	1	0	:=InAB	0
VAS12	:=0	:=COUNT_S+1	0	0	0	0	0	0	0	0	1	:=InAB	0
VAS13	:=0	:=COUNT_S+1	0	0	0	0	0	0	0	0	0	0	1
VAS14	:=0	:= COUNT_S+1	0	0	0	0	0	0	0	0	0	0	0

Tabelle 8-3: Belegung der Ausgangsvariablen InAB_INPUT



8.1.5 Auswahl Automatentyp



IL / OL Input / Output Logic
Eingangs- / Ausgangslogik
IREG Input Register
Eingangsregister
SREG_M State Register Master
Zustandsregister Master
SREG State Register
Zustandsregister (Slave)

✓ Informationsverarbeitung erfolgt mit der 0/1-Flanke
der Taktvariablen
✗ Informationsverarbeitung erfolgt mit der 1/0-Flanke
der Taktvariablen

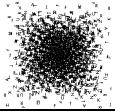
X Eingangsvariablen
n_X Eingangsvariablen, zwischengespeichert
SV State Variables
Zustandsvariablen
n_SV State Variables
Zustandsvariablen
neuer Wert
SV_Master State Variables
Zustandsvariablen
Master
Y Ausgangsvariablen

Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister

8.1.6 Liste der Prozesse

Prozessname	Funktion
IREG_PROC	Eingangsregister
SREG_M_PROC	Zustandsregister, Master
SREG_S_PROC	Zustandsregister, Slave
IL_OL_PROC	Eingangs- / Ausgangslogik
STATE_DISPL_PROC	Ausgabe aktueller und folgender Zustand
SWITCH_VALUES_PROC	Umschaltung zw. Werten für normaler Funktion und Schrittbetrieb

Tabelle 8-4: Liste der Prozesse InAB_INPUT



8.1.7 VHDL-Programm

```
-- CTRL_InAB_INPUT
-- Einlesen des Datenstroms von InAB und Ausgabe als Einzelnes Bit, sowie Signalisierung das Byte
komplet
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 09.10.2012
-- Bearbeiter: mharndt
-- Geaendert: 29.01.2013
-- Umstellung auf: rising_edge(CLK) und falling_edge(CLK) und http://www.sigasi.com/content/clock-
edge-detection
-- Optimierungen aus: http://www.lothar-miller.de/s9y/categories/37-FSM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_InAB_INPUT_VHDL is
    Port (InAB      : in std_logic; --Eingangsvariable, Eingang Profibussignal
          CHOSE_VALUE : in std_logic; --Eingangsvariable, Zählerwert aendern
          EN_BIT_i    : out std_logic_vector (8 downto 0); --Ausgangsvariable, Enable Bit i, 9bit
          BIT_VALUE   : out std_logic; --Ausgangsvariable, Bitwert
          BYTE_CMPLT: out std_logic; --Ausgangsvariabel, Byte empfangen und komplett
          PAUSE_END   : out std_logic; --Ausgangssignal, Pause zu Ende

          CLK         : in std_logic; --Taktvariable
          CLK_IO      : in std_logic; --Taktvariable,
                           --Ein- und Ausgangsregister
          IN_NEXT_STATE: in std_logic; --1:Zustandsuebergang möglich
          RESET       : in std_logic; --1: Initialzustand annehmen

          DISPL1_SV   : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl1, binärzahl
          DISPL2_SV   : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl2, binärzahl
          DISPL1_n_SV : out std_logic_vector (3 downto 0); --Folgezustand Zahl1, binärzahl
          DISPL2_n_SV : out std_logic_vector (3 downto 0)); --Folgezustand Zahl2, binärzahl

end CTRL_InAB_INPUT_VHDL;

architecture Behavioral of CTRL_InAB_INPUT_VHDL is

type TYPE_STATE is
    (ST_CTRL_00, --Zustaende CTRL_9P6_50MHZ
     ST_CTRL_01,
     ST_CTRL_02,
     ST_CTRL_03,
     ST_CTRL_04,
     ST_CTRL_05,
     ST_CTRL_06,
     ST_CTRL_07,
     ST_CTRL_08,
     ST_CTRL_09,
     ST_CTRL_0A, --10
     ST_CTRL_0B, --11
     ST_CTRL_0C, --12
     ST_CTRL_0D, --13
     ST_CTRL_0E, --14
     ST_CTRL_0F);--15

signal SV : TYPE_STATE := ST_CTRL_00; --Zustandsvariable
signal n_SV: TYPE_STATE := ST_CTRL_00; --Zustandsvariable, neuer Wert
signal SV_M: TYPE_STATE := ST_CTRL_00; --Zustandsvariable, Ausgang Master

signal COUNT_L   : std_logic_vector (19 downto 0) := x"00000"; --großer Zaehler, Vektor, 20 Bit
signal n_COUNT_L: std_logic_vector (19 downto 0) := x"00000"; --großer Zaehler, neuer Wert, Vektor,
20 Bit
signal COUNT_L_M : std_logic_vector (19 downto 0) := x"00000"; --großer Zaehler, Ausgang Master,
Vektor, 20 Bit

signal COUNT_S   : std_logic_vector (15 downto 0) := x"0000"; --kleiner Zaehler, Vektor, 16 Bit
signal n_COUNT_S: std_logic_vector (15 downto 0) := x"0000"; --kleiner Zaehler, neuer Wert, Vektor,
16 Bit
signal COUNT_S_M : std_logic_vector (15 downto 0) := x"0000"; --kleiner Zaehler, Ausgang Master,
Vektor, 16 Bit

signal InAB_S    : std_logic := '0'; --Eingangsvariable
```



```
--Zwischengespeichert im Eingangsregister

--signal not_CLK    : std_logic; --negierte Taktvariable
--signal not_CLK_IO: std_logic; --negierte Taktvariable
--      --Ein- und Ausgangsregister

signal STATE_SV     : std_logic_vector (7 downto 0); -- aktueller Zustand in 8 Bit, binär
signal STATE_n_SV   : std_logic_vector (7 downto 0); -- Folgezustand in 8 Bit, binär

signal EN_BIT_0      : std_logic := '0'; --BIT0
signal EN_BIT_1      : std_logic := '0'; --BIT1
signal EN_BIT_2      : std_logic := '0'; --BIT2
signal EN_BIT_3      : std_logic := '0'; --BIT3
signal EN_BIT_4      : std_logic := '0'; --BIT4
signal EN_BIT_5      : std_logic := '0'; --BIT5
signal EN_BIT_6      : std_logic := '0'; --BIT6
signal EN_BIT_7      : std_logic := '0'; --BIT7
signal EN_BIT_8      : std_logic := '0'; --Paritätsbit

signal CNTS30 : std_logic_vector (19 downto 0) := x"00000"; --Zählerwerte
signal CNTT01 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT02 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT03 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT04 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT05 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT06 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT07 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT08 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT09 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT10 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT11 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT12 : std_logic_vector (15 downto 0) := x"0000";
signal CNTT13 : std_logic_vector (15 downto 0) := x"0000";

--Konstanten, lang
constant long_CNTS30 : std_logic_vector := x"2625A"; --20 Bit
constant long_CNTT01 : std_logic_vector := x"0A2C"; --16 Bit
constant long_CNTT02 : std_logic_vector := x"1E84"; --usw.
constant long_CNTT03 : std_logic_vector := x"32DC";
constant long_CNTT04 : std_logic_vector := x"4735";
constant long_CNTT05 : std_logic_vector := x"5B8B";
constant long_CNTT06 : std_logic_vector := x"6FE4";
constant long_CNTT07 : std_logic_vector := x"8441";
constant long_CNTT08 : std_logic_vector := x"9872";
constant long_CNTT09 : std_logic_vector := x"ACEE";
constant long_CNTT10 : std_logic_vector := x"C147";
constant long_CNTT11 : std_logic_vector := x"D59F";
constant long_CNTT12 : std_logic_vector := x"D9B1";
constant long_CNTT13 : std_logic_vector := x"E5E6";

--Konstanten, kurz
constant short_CNTS30 : std_logic_vector := x"0000A"; --10
constant short_CNTT01 : std_logic_vector := x"0003"; --3
constant short_CNTT02 : std_logic_vector := x"0006"; --6
constant short_CNTT03 : std_logic_vector := x"0009"; --9
constant short_CNTT04 : std_logic_vector := x"000C"; --12
constant short_CNTT05 : std_logic_vector := x"000F"; --15
constant short_CNTT06 : std_logic_vector := x"0012"; --18
constant short_CNTT07 : std_logic_vector := x"0015"; --21
constant short_CNTT08 : std_logic_vector := x"0018"; --24
constant short_CNTT09 : std_logic_vector := x"001B"; --27
constant short_CNTT10 : std_logic_vector := x"001E"; --30
constant short_CNTT11 : std_logic_vector := x"0021"; --33
constant short_CNTT12 : std_logic_vector := x"0024"; --36
constant short_CNTT13 : std_logic_vector := x"002A"; --42

begin

--NOT_CLK_PROC: process (CLK) --negieren Taktvariable
--begin
--  not_CLK <= not CLK;
--end process;
--NOT_CLK_IO_PROC: process (CLK_IO) --negieren Taktvariable
--      --Ein- und Ausgangsregister
--begin
--  not_CLK_IO <= not CLK_IO;
--end process;
```



```
IREG_PROC: process (InAB, InAB_S, CLK) --Eingangsregister
begin
  if falling_edge(CLK) --Eingangsregister
    then InAB_S <= InAB;
  end if;
end process;

SREG_M_PROC: process (RESET, n_SV, n_COUNT_L,n_COUNT_S, CLK) --Master
begin
  if (RESET = '1')
  then SV_M      <= ST_CTRL_00;
    COUNT_L_M <= x"00000";
    COUNT_S_M <= x"0000";
  else
    if rising_edge(CLK)
    then
      if (IN_NEXT_STATE = '1')
      then SV_M      <= n_SV;
        COUNT_L_M <= n_COUNT_L;
        COUNT_S_M <= n_COUNT_S;
      else SV_M      <= SV_M;
        COUNT_L_M <= COUNT_L_M;
        COUNT_S_M <= COUNT_S_M;
      end if;
    end if;
  end if;
end process;

SREG_S_PROC: process (RESET, SV_M, COUNT_L_M, COUNT_S_M, CLK) --Slave
begin
  if (RESET = '1')
  then SV      <= ST_CTRL_00;
    COUNT_L <= x"00000";
    COUNT_S <= x"0000";
  else
    if falling_edge(CLK)
    then SV      <= SV_M;
      COUNT_L <= COUNT_L_M;
      COUNT_S <= COUNT_S_M;
    end if;
  end if;
end process;

IL_DL_PROC: process (InAB_S, SV, COUNT_L,COUNT_S, CNTS30, CNTT01, CNTT02, CNTT03, CNTT04, CNTT05,
CNTT06, CNTT07, CNTT08, CNTT09, CNTT10, CNTT11, CNTT12, CNTT13)
begin
  case SV is
    when ST_CTRL_00 =>
      if (InAB_S = '1')
      then
        -- VAS00
        PAUSE_END <= '0';
        n_COUNT_L <= x"00000"; -- großer Zaehler Neustart
        n_COUNT_S <= x"0000"; -- kleiner Zaehler Neustart
        EN_BIT_0 <= '0';
        EN_BIT_1 <= '0';
        EN_BIT_2 <= '0';
        EN_BIT_3 <= '0';
        EN_BIT_4 <= '0';
        EN_BIT_5 <= '0';
        EN_BIT_6 <= '0';
        EN_BIT_7 <= '0';
        EN_BIT_8 <= '0';
        BIT_VALUE <= '0';
        BYTE_CMPLT <= '0';
        n_SV <= ST_CTRL_01; -- Zustandsuebergang
      else
        --VAS00
        PAUSE_END <= '0';
        n_COUNT_L <= x"00000"; -- großer Zaehler nullen
        n_COUNT_S <= x"0000"; -- kleiner Zaehler nullen
        EN_BIT_0 <= '0';
        EN_BIT_1 <= '0';
        EN_BIT_2 <= '0';
        EN_BIT_3 <= '0';
        EN_BIT_4 <= '0';
        EN_BIT_5 <= '0';
```



```
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_00; --InAB = '0'
end if;

when ST_CTRL_01 =>
if (InAB_S = '1')
then
  if (COUNT_L = CNTS30) --156250 -- if (COUNT >=3)
  then
    -- VAS00
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= x"0000";
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_02; -- Zustandsuebergang
  else --not COUNT_L = CNTS30
    --VAS01
    PAUSE_END <= '0';
    n_COUNT_L <= COUNT_L+1;
    n_COUNT_S <= x"0000";
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_01; --Zaehlschleife
  end if;
else --InAB_S = '1'
  --VAS00
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= x"0000";
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= '0';
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_00; -- Zustandsuebergang
end if;

when ST_CTRL_02 =>
if (InAB_S = '0')
then
  -- VAS03
  PAUSE_END <= '1';
  n_COUNT_L <= x"00000"; -- Zaehler Neustart
  n_COUNT_S <= x"0000";
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
```



```
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_03; -- Zustandsuebergang
else
    -- InAB_S = '1'
--VAS00
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= x"00000";
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_02; --warte ab bis InAB wieder Null wird
end if;

when ST_CTRL_03 =>
if (COUNT_S = CNTT01) --2604
then
if (InAB_S = '0') -- Startbit erkannt
then
-- VAS02
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_04; -- Zustandsuebergang
else
    --InAB_S = '1'
-- VAS00
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= x"00000";
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_00;
end if;
else
-- VAS02
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
```



```
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_03; -- Zustandsuebergang
end if;

when ST_CTRL_04 =>
  if (COUNT_S = CNTT02) --7812
  then
    -- VAS04
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '1';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= InAB_S;
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_05; -- Zustandsuebergang
  else
    --n_COUNT < CNTT02
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_04; --Zaehlschleife
  end if;

when ST_CTRL_05 =>
  if (COUNT_S = CNTT03) --13020
  then
    -- VAS05
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '1';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= InAB_S;
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_06; -- Zustandsuebergang
  else
    --n_COUNT < CNTT03
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
  end if;
```



```
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_05; --Zaehlschleife
end if;

when ST_CTRL_06 =>
  if (COUNT_S = CNTT04) --18229
  then
    -- VAS06
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '1';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= InAB_S;
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_07; -- Zustandsuebergang
  else
    --n_COUNT < CNTT04
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_06; --Zaehlschleife
  end if;

when ST_CTRL_07 =>
  if (COUNT_S = CNTT05) --23435
  then
    -- VAS07
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '1';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= InAB_S;
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_08; -- Zustandsuebergang
  else
    --n_COUNT < CNTT05
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_07; --Zaehlschleife
```



```
end if;

when ST_CTRL_08 =>
  if (COUNT_S = CNTT06) --28644
  then
-- VAS08
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '1';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= InAB_S;
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_09; -- Zustandsuebergang
else
  --n_COUNT < CNTT06
-- VAS02
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= '0';
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_08; --Zaehlschleife
end if;

when ST_CTRL_09 =>
  if (COUNT_S = CNTT07) --33854
  then
-- VAS09
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '1';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= InAB_S;
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_0A; -- Zustandsuebergang
else
  --n_COUNT < CNTT07
-- VAS02
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= '0';
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_09; --Zaehlschleife
end if;
```



```
when ST_CTRL_0A =>
  if (COUNT_S = CNTT08) --39062
  then
-- VAS10
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '1';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= InAB_S;
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_0B; -- Zustandsuebergang
else
  --n_COUNT < CNTT08
-- VAS02
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= '0';
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_0A; --Zaehlschleife
end if;

when ST_CTRL_0B =>
  if (COUNT_S = CNTT09) --44270
  then
-- VAS11
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '1';
  EN_BIT_8 <= '0';
  BIT_VALUE <= InAB_S;
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_0C; -- Zustandsuebergang
else
  --n_COUNT < CNTT09
-- VAS02
  PAUSE_END <= '0';
  n_COUNT_L <= x"00000";
  n_COUNT_S <= COUNT_S+1;
  EN_BIT_0 <= '0';
  EN_BIT_1 <= '0';
  EN_BIT_2 <= '0';
  EN_BIT_3 <= '0';
  EN_BIT_4 <= '0';
  EN_BIT_5 <= '0';
  EN_BIT_6 <= '0';
  EN_BIT_7 <= '0';
  EN_BIT_8 <= '0';
  BIT_VALUE <= '0';
  BYTE_CMPLT <= '0';
  n_SV <= ST_CTRL_0B; --Zaehlschleife
end if;

when ST_CTRL_0C =>
  if (COUNT_S = CNTT10) --49479
```



```
then
-- VAS12
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '1';
BIT_VALUE <= InAB_S;
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_0D; -- Zustandsuebergang
else
    --n_COUNT < CNTT10
-- VAS02
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_0C; --Zaehlschleife
end if;

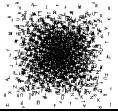
when ST_CTRL_0D =>
if (COUNT_S = CNTT11) --54687
then
if (InAB_S = '0')
then
-- VAS03
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_00; -- Error: Kein Stoppbit, vormals ST_CTRL_05
else
    --InAB_S = '1'
-- VAS13
PAUSE_END <= '0';
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '1';
n_SV <= ST_CTRL_0E; --Stoppbit erkannt
end if; --InAB_S = '0'
else --not COUNT_S = CNTT11
-- VAS02
PAUSE_END <= '0';
```



```
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_0D; --Zaehlschleife
end if; --COUNT_S = CNTT11

when ST_CTRL_0E =>
  if (COUNT_S = CNTT12) --60937
  then
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_OF; -- Zustandsuebergang
  else
    -- n_COUNT < CNTT12
    -- VAS02
    PAUSE_END <= '0';
    n_COUNT_L <= x"00000";
    n_COUNT_S <= COUNT_S+1;
    EN_BIT_0 <= '0';
    EN_BIT_1 <= '0';
    EN_BIT_2 <= '0';
    EN_BIT_3 <= '0';
    EN_BIT_4 <= '0';
    EN_BIT_5 <= '0';
    EN_BIT_6 <= '0';
    EN_BIT_7 <= '0';
    EN_BIT_8 <= '0';
    BIT_VALUE <= '0';
    BYTE_CMPLT <= '0';
    n_SV <= ST_CTRL_0E; --Zaehlschleife
  end if;

when ST_CTRL_OF =>
  if (InAB_S = '1') --Startbit bisher noch nicht gefunden
  then
    if (COUNT_S = CNTT13) --64062
    then
      -- VAS00
      PAUSE_END <= '0';
      n_COUNT_L <= x"00000"; -- Zaehler nullen
      n_COUNT_S <= x"0000"; -- Zaehler nullen
      EN_BIT_0 <= '0';
      EN_BIT_1 <= '0';
      EN_BIT_2 <= '0';
      EN_BIT_3 <= '0';
      EN_BIT_4 <= '0';
      EN_BIT_5 <= '0';
      EN_BIT_6 <= '0';
      EN_BIT_7 <= '0';
      EN_BIT_8 <= '0';
      BIT_VALUE <= '0';
      BYTE_CMPLT <= '0';
      n_SV <= ST_CTRL_00; -- Kein Startbit gefunden (neues SYN?)
    else --not COUNT_S = CNTT13
      -- VAS02
      PAUSE_END <= '0';
```



```
n_COUNT_L <= x"00000";
n_COUNT_S <= COUNT_S+1;
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_0F; --Zaehlschleife
end if; --COUNT_S = CNTT13
else --InAB_S = '0'
-- Startbit gefunden
-- VAS00
PAUSE_END <= '0';
n_COUNT_L <= x"00000"; -- Zaehler Neustart
n_COUNT_S <= x"0000"; -- Zaehler Neustart
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_03; -- Zustandsuebergang
end if;

when others =>
-- VAS00
PAUSE_END <= '0';
n_COUNT_L <= x"00000"; -- Zaehler Neustart
n_COUNT_S <= x"0000"; -- Zaehler Neustart
EN_BIT_0 <= '0';
EN_BIT_1 <= '0';
EN_BIT_2 <= '0';
EN_BIT_3 <= '0';
EN_BIT_4 <= '0';
EN_BIT_5 <= '0';
EN_BIT_6 <= '0';
EN_BIT_7 <= '0';
EN_BIT_8 <= '0';
BIT_VALUE <= '0';
BYTE_CMPLT <= '0';
n_SV <= ST_CTRL_00;
end case;
end process;

--BYTE_IN_PROC: process (EN_BIT_0, EN_BIT_1, EN_BIT_2, EN_BIT_3, EN_BIT_4, EN_BIT_5, EN_BIT_6,
EN_BIT_7, EN_BIT_8) --Umwandlung einzelnes Bit EIN_BIT_0_S bis 8_S in Vector EN_BIT_i
begin
EN_BIT_i(0) <= EN_BIT_0;
EN_BIT_i(1) <= EN_BIT_1;
EN_BIT_i(2) <= EN_BIT_2;
EN_BIT_i(3) <= EN_BIT_3;
EN_BIT_i(4) <= EN_BIT_4;
EN_BIT_i(5) <= EN_BIT_5;
EN_BIT_i(6) <= EN_BIT_6;
EN_BIT_i(7) <= EN_BIT_7;
EN_BIT_i(8) <= EN_BIT_8;
end process;

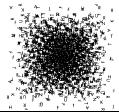
STATE_DISPL_PROC: process (SV, n_SV, STATE_SV, STATE_n_SV) -- Zustandsanzeige
begin
STATE_SV <= conv_std_logic_vector(TYPE_STATE'pos(SV),8); --Zustandsumwandlung in 8 Bit
STATE_n_SV <= conv_std_logic_vector(TYPE_STATE'pos(n_SV),8);

DISPL1_SV(0) <= STATE_SV(0); --Bit0
DISPL1_SV(1) <= STATE_SV(1); --Bit1
DISPL1_SV(2) <= STATE_SV(2); --Bit2
DISPL1_SV(3) <= STATE_SV(3); --Bit3
```

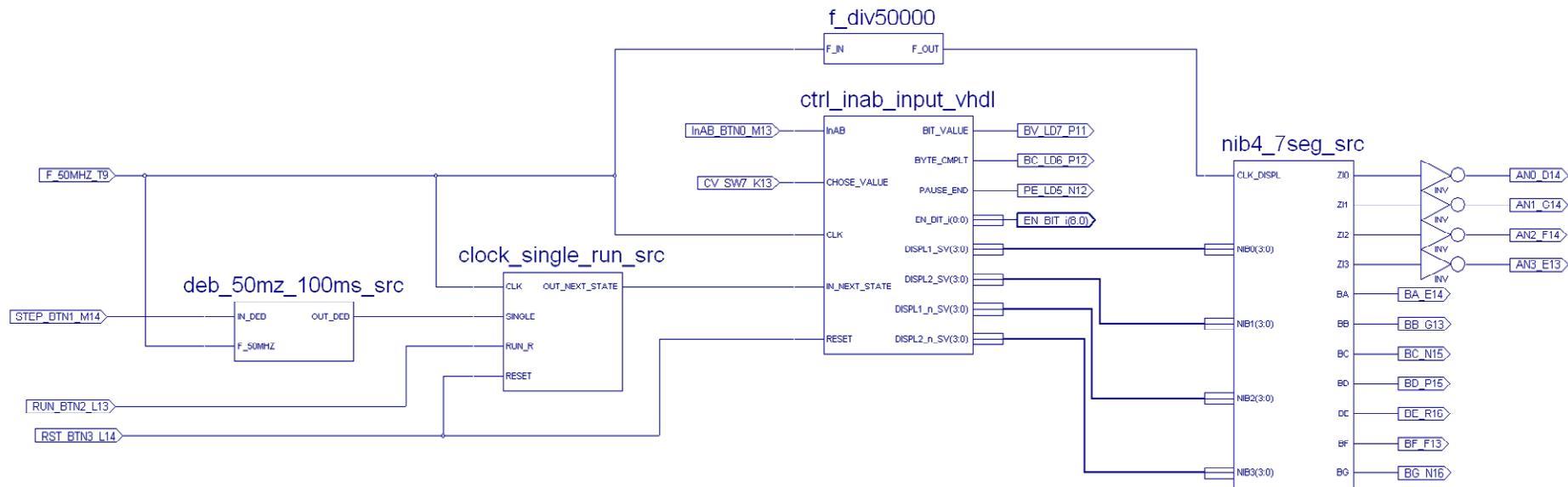


```
DISPL2_SV(0) <= STATE_SV(4); --usw.  
DISPL2_SV(1) <= STATE_SV(5);  
DISPL2_SV(2) <= STATE_SV(6);  
DISPL2_SV(3) <= STATE_SV(7);  
  
--Folgezustand anzeigen  
DISPL1_n_SV(0) <= STATE_n_SV(0);  
DISPL1_n_SV(1) <= STATE_n_SV(1);  
DISPL1_n_SV(2) <= STATE_n_SV(2);  
DISPL1_n_SV(3) <= STATE_n_SV(3);  
  
DISPL2_n_SV(0) <= STATE_n_SV(4);  
DISPL2_n_SV(1) <= STATE_n_SV(5);  
DISPL2_n_SV(2) <= STATE_n_SV(6);  
DISPL2_n_SV(3) <= STATE_n_SV(7);  
end process;  
  
SWITCH_VALUES_PROC: process (CHOSE_VALUE) --Schaltet zw. langen und kurzem Zaehler um  
begin  
if (CHOSE_VALUE = '0')  
then  
    --normale Werte  
    CNTS30 <= long_CNTS30;  
    CNTT01 <= long_CNTT01;  
    CNTT02 <= long_CNTT02;  
    CNTT03 <= long_CNTT03;  
    CNTT04 <= long_CNTT04;  
    CNTT05 <= long_CNTT05;  
    CNTT06 <= long_CNTT06;  
    CNTT07 <= long_CNTT07;  
    CNTT08 <= long_CNTT08;  
    CNTT09 <= long_CNTT09;  
    CNTT10 <= long_CNTT10;  
    CNTT11 <= long_CNTT11;  
    CNTT12 <= long_CNTT12;  
    CNTT13 <= long_CNTT13;  
else  
    --kurze Werte  
    CNTS30 <= short_CNTS30;  
    CNTT01 <= short_CNTT01;  
    CNTT02 <= short_CNTT02;  
    CNTT03 <= short_CNTT03;  
    CNTT04 <= short_CNTT04;  
    CNTT05 <= short_CNTT05;  
    CNTT06 <= short_CNTT06;  
    CNTT07 <= short_CNTT07;  
    CNTT08 <= short_CNTT08;  
    CNTT09 <= short_CNTT09;  
    CNTT10 <= short_CNTT10;  
    CNTT11 <= short_CNTT11;  
    CNTT12 <= short_CNTT12;  
    CNTT13 <= short_CNTT13;  
end if;  
end process;  
end Behavioral;
```

Datei 8-1..\VHDL_Bausteine\CTRL_InAB_INPUT\CTRL_InAB_INPUT_VHDL.vhd



8.1.8 Testumgebung



CTRL_InAB_INPUT_VHDL
DEB_50MHZ_100MS_SRC
CLOCK_SINGLE_RUN_SRC
F_DIV50000
NIB4_7SEG_SRC
INV

Siehe 8.1 Modul InAB_INPUT, Seite 8-1
Siehe /DEB_50MHZ_100MS/, Seite 12-18
Siehe /CLOCK_SINGLE_RUN/, Seite 12-18
Siehe /F_DIV50000/, Seite 12-18
Siehe /NIB_7SEG_SRC/, Seite 12-18
Siehe /INV/, Seite 12-18

Abbildung 8-6: Testumgebung InAB_INPUT



8.1.9 UCF-Datei

#PACE: Start of Constraints generated by PACE

```
#PACE: Start of PACE I/O Pin Assignments
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_LD6_P12" LOC = "P12" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "BV_LD7_P11" LOC = "P11" ;
NET "CV_SW7_K13" LOC = "K13" ;
NET "EN_BIT_i<0>" LOC = "K12" ;
NET "EN_BIT_i<1>" LOC = "P14" ;
NET "EN_BIT_i<2>" LOC = "L12" ;
NET "EN_BIT_i<3>" LOC = "N14" ;
NET "EN_BIT_i<4>" LOC = "P13" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "InAB_BTN0_M13" LOC = "M13" ;
NET "PE_LD5_N12" LOC = "N12" ;
NET "RST_BTN3_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Datei 8-2: ..\VHDL_Bausteine\TEST_CTRL_InAB_INPUT\test_ctrl_inab_input.sch.ucf



8.1.10 Steuerungsbelegung Testumgebung

Buttons	BTN0:	InAB
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	nicht genutzt
	SW1:	nicht genutzt
	SW2:	nicht genutzt
	SW3:	nicht genutzt
	SW4:	nicht genutzt
	SW5:	nicht genutzt
	SW6:	nicht genutzt
	SW7:	CHOOSE_VALUE (0:Werte RUN; 1:Werte Schrittbetrieb)
LEDs	LD0:	EN_BIT_i(0)
	LD1:	EN_BIT_i (1)
	LD2:	EN_BIT_i (2)
	LD3:	EN_BIT_i (3)
	LD4:	EN_BIT_i (4); Kein EN_BIT_i(5) bis (8)
	LD5:	PAUSE_END
	LD6:	BYTE_CMPLT
	LD7:	BIT_VALUE

Tabelle 8-5: Steuerungsbelegung Testumgebung InAB_INPUT

8.1.11 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand verkürzt aus. Siehe auch **Tabelle 8-4: Liste der Prozesse InAB_INPUT**, Seite 8-8.

Beispiel: Die Anzeige „090A“ gibt den aktuellen Zustand ST_CTRL_09 und den folgenden Zustand ST_CTRL_0A an.

Für weitere Zustände siehe **Tabelle 8-1: Variablendefinition InAB_INPUT**, Seite 8-3.



8.2 Modul BIT_REGISTER

Das Funktionsmodul BIT_REGISTER erhält als Eingangssignale die Ausgangssignale des Moduls InAB_INPUT. Es prüft die Parität und wandelt die einzelnen Datenbits in ein Datenbyte um.

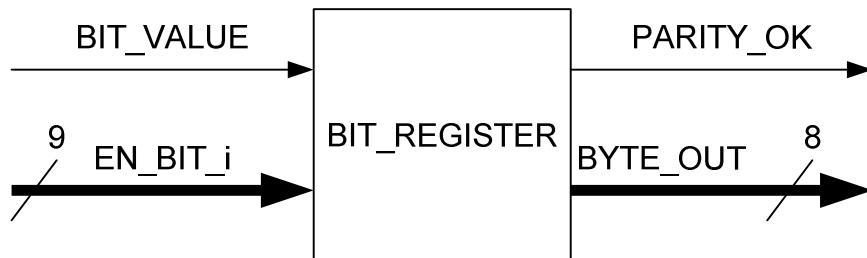


Abbildung 8-7: Wirkungsplan BIT_REGISTER

Variablenname	Datentyp	Variablentyp	Informationen / Anweisungen
EN_Bit_i	VECTOR, 9 Bit	Eingang	Einschalten Bit 0-8
BIT_VALUE	BOOL	Eingang	1: Bitwert ist 1
BYTE_OUT	VECTOR, 8 Bit	Ausgang	Ausgabe Datenbyte
PARITY_OK	BOOL	Ausgang	1: Parität ist korrekt
BYTE_VEC	VECTOR, 9 Bit	intern	Datenbits zur Auswertung der Parität
TYPE_STATE_BR_BIT0	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT0_0, ST_BR_EN_BIT0_1)
TYPE_STATE_BR_BIT1	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT1_0, ST_BR_EN_BIT1_1)
TYPE_STATE_BR_BIT2	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT2_0, ST_BR_EN_BIT2_1)
TYPE_STATE_BR_BIT3	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT3_0, ST_BR_EN_BIT3_1)
TYPE_STATE_BR_BIT4	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT4_0, ST_BR_EN_BIT4_1)
TYPE_STATE_BR_BIT5	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT5_0, ST_BR_EN_BIT5_1)
TYPE_STATE_BR_BIT6	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT6_0, ST_BR_EN_BIT6_1)
TYPE_STATE_BR_BIT7	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT7_0, ST_BR_EN_BIT7_1)
TYPE_STATE_BR_BIT8	ENUM	intern	Zustandsvariable (ST_BR_EN_BIT8_0, ST_BR_EN_BIT8_1)
TMP00	VARIABLE	intern	XOR-Zwischenspeicher 0



Variablenname	Datentyp	Variablentyp	Informationen / Anweisungen
TMP01	VARIABLE	intern	XOR-Zwischenspeicher 1
TMP02	VARIABLE	intern	XOR-Zwischenspeicher 2
TMP03	VARIABLE	intern	XOR-Zwischenspeicher 3
TMP10	VARIABLE	intern	XOR-Zwischenspeicher 4
TMP11	VARIABLE	intern	XOR-Zwischenspeicher 5
TMP20	VARIABLE	intern	XOR-Zwischenspeicher 6

Tabelle 8-6: Variablendefinition BIT_REGSITER

8.2.1 Verhaltensbeschreibung

Der Programmablaufgraph wurde für jedes der 9 Bit von EN_BIT_i einzeln umgesetzt. Die Zustände ST_BR_00 und ST_BR_01 stehen hier also für die Zustandsvariablen ST_BR_EN_BIT0_0 und ST_BR_EN_BIT0_1 usw.

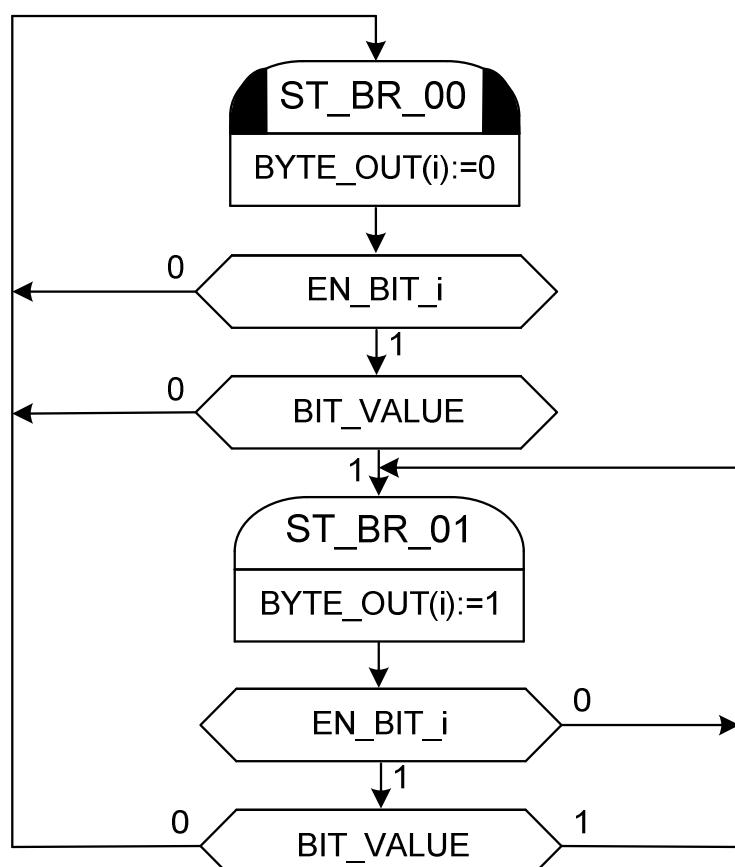


Abbildung 8-8: Programmablaufgraph BIT_REGISTER



8.2.2 Belegung Ausgangsvariablen

Siehe *Abbildung 8-8: Programmablaufgraph BIT_REGISTER*, Seite 8-26.

8.2.3 Auswahl Automatentyp

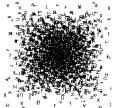
Siehe *Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister*, Seite 8-8.

Der einziger Unterschied ist, dass kein Eingangsregister verwendet wurde. IREG und n_X entfallen. Die Eingangsvariable X geht direkt in die Eingangs- /Ausgangslogik IL/OL.

8.2.4 Liste der Prozesse

Prozessname	Funktion
SREG_M_PROC	Zustandsregister, Master
SREG_S_PROC	Zustandsregister, Slave
BIT_REGISTER_EN_BIT0_PROC	Bitregister Bit 0
BIT_REGISTER_EN_BIT1_PROC	Bitregister Bit 1
BIT_REGISTER_EN_BIT2_PROC	Bitregister Bit 2
BIT_REGISTER_EN_BIT3_PROC	Bitregister Bit 3
BIT_REGISTER_EN_BIT4_PROC	Bitregister Bit 4
BIT_REGISTER_EN_BIT5_PROC	Bitregister Bit 5
BIT_REGISTER_EN_BIT6_PROC	Bitregister Bit 6
BIT_REGISTER_EN_BIT7_PROC	Bitregister Bit 7
BIT_REGISTER_EN_BIT8_PROC	Bitregister Bit 8
PARITY_CHECK_PROC	Paritätsprüfung
STATE_DISPL_PROC	Ausgabe aktueller und folgender Zustand
SWITCH_VALUES_PROC	Umschaltung zw. Werten für normaler Funktion und Schrittbetrieb

Tabelle 8-7: Liste der Prozesse BIT_REGISTER



8.2.5 VHDL-Programm

```
-- CTRL_BIT_REGISTER
-- Einlesen der einzelnen Werte für bestimmte Bits, Berechnung der Parität und Ausgabe als Byte
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 08.01.2013
-- Bearbeiter: mharndt
-- Geaendert: 25.01.2013
-- Umstellung auf: rising_edge(CLK) und falling_edge(CLK) und http://www.sigasi.com/content/clock-
edge-detection
-- Optimierungen aus: http://www.lothar-miller.de/s9y/categories/37-FSM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_BIT_REGISTER is
    Port (EN_BIT_i : in std_logic_vector (8 downto 0); --Eingangsvariable, Einschalten des
          Bitregisters i

          BIT_VALUE : in std_logic; -- Eingangsvariable, Wert des aktuellen Bits
          BYTE_OUT : out std_logic_vector (7 downto 0); --Ausgangsvariable, Byte, 8bit, Vektor
          PARITY_OK : out std_logic; --Ausgangsvariable, Parität i.O.

          CLK : in std_logic; --Taktvariable
          IN_NEXT_STATE: in std_logic; --1:Zustandsuebergang möglich
          RESET : in std_logic); --1: Initialzustand annehmen

end CTRL_BIT_REGISTER;

architecture Behavioral of CTRL_BIT_REGISTER is

type TYPE_STATE_BR_BIT0 is
    (ST_BR_EN_BIT0_0, --Zustaende BIT_REGISTER BIT0
     ST_BR_EN_BIT0_1);

type TYPE_STATE_BR_BIT1 is
    (ST_BR_EN_BIT1_0, --Zustaende BIT_REGISTER BIT1
     ST_BR_EN_BIT1_1);

type TYPE_STATE_BR_BIT2 is
    (ST_BR_EN_BIT2_0, --Zustaende BIT_REGISTER BIT2
     ST_BR_EN_BIT2_1);

type TYPE_STATE_BR_BIT3 is
    (ST_BR_EN_BIT3_0, --Zustaende BIT_REGISTER BIT3
     ST_BR_EN_BIT3_1);

type TYPE_STATE_BR_BIT4 is
    (ST_BR_EN_BIT4_0, --Zustaende BIT_REGISTER BIT4
     ST_BR_EN_BIT4_1);

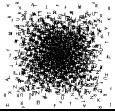
type TYPE_STATE_BR_BIT5 is
    (ST_BR_EN_BIT5_0, --Zustaende BIT_REGISTER BIT5
     ST_BR_EN_BIT5_1);

type TYPE_STATE_BR_BIT6 is
    (ST_BR_EN_BIT6_0, --Zustaende BIT_REGISTER BIT6
     ST_BR_EN_BIT6_1);

type TYPE_STATE_BR_BIT7 is
    (ST_BR_EN_BIT7_0, --Zustaende BIT_REGISTER BIT7
     ST_BR_EN_BIT7_1);

type TYPE_STATE_BR_BIT8 is
    (ST_BR_EN_BIT8_0, --Zustaende BIT_REGISTER BIT8
     ST_BR_EN_BIT8_1);

signal SV_BR_BIT0 : TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0
signal n_SV_BR_BIT0: TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0,
neuer Wert
signal SV_BR_BIT0_M: TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0,
Ausgang Master
```



```
signal SV_BR_BIT1 : TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1
signal n_SV_BR_BIT1: TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1,
neuer Wert
signal SV_BR_BIT1_M: TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1,
Ausgang Master

signal SV_BR_BIT2 : TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2
signal n_SV_BR_BIT2: TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2,
neuer Wert
signal SV_BR_BIT2_M: TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2,
Ausgang Master

signal SV_BR_BIT3 : TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3
signal n_SV_BR_BIT3: TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3,
neuer Wert
signal SV_BR_BIT3_M: TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3,
Ausgang Master

signal SV_BR_BIT4 : TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4
signal n_SV_BR_BIT4: TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4,
neuer Wert
signal SV_BR_BIT4_M: TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4,
Ausgang Master

signal SV_BR_BIT5 : TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5
signal n_SV_BR_BIT5: TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5,
neuer Wert
signal SV_BR_BIT5_M: TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5,
Ausgang Master

signal SV_BR_BIT6 : TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6
signal n_SV_BR_BIT6: TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6,
neuer Wert
signal SV_BR_BIT6_M: TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6,
Ausgang Master

signal SV_BR_BIT7 : TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7
signal n_SV_BR_BIT7: TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7,
neuer Wert
signal SV_BR_BIT7_M: TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7,
Ausgang Master

signal SV_BR_BIT8 : TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8
signal n_SV_BR_BIT8: TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8,
neuer Wert
signal SV_BR_BIT8_M: TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8,
Ausgang Master

signal BYTE_VEC : std_logic_vector (8 downto 0) := b"00000000"; -- Vektor, BIT_REGSITER, vor
Auswertung der Checksume

--signal not_CLK : std_logic; --negierte Taktvariable

--signal TMP00 : std_logic; --temporärer Zwischenwert, Paritätsprüfung
--signal TMP01 : std_logic;
--signal TMP02 : std_logic;
--signal TMP03 : std_logic;
--signal TMP10 : std_logic;
--signal TMP11 : std_logic;
--signal TMP20 : std_logic;

begin

--NOT_CLK_PROC: process (CLK) --negieren Taktvariable
--begin
--  not_CLK <= not CLK;
--end process;

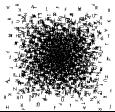
SREG_M_PROC: process (RESET, n_SV_BR_BIT0, n_SV_BR_BIT1, n_SV_BR_BIT2, n_SV_BR_BIT3, n_SV_BR_BIT4,
n_SV_BR_BIT5, n_SV_BR_BIT6, n_SV_BR_BIT7, n_SV_BR_BIT8, CLK) --Master
begin
  if (RESET = '1') then
    SV_BR_BIT0_M <= ST_BR_EN_BIT0_0;
    SV_BR_BIT1_M <= ST_BR_EN_BIT1_0;
    SV_BR_BIT2_M <= ST_BR_EN_BIT2_0;
    SV_BR_BIT3_M <= ST_BR_EN_BIT3_0;
```



```
SV_BR_BIT4_M <= ST_BR_EN_BIT4_0;
SV_BR_BIT5_M <= ST_BR_EN_BIT5_0;
SV_BR_BIT6_M <= ST_BR_EN_BIT6_0;
SV_BR_BIT7_M <= ST_BR_EN_BIT7_0;
SV_BR_BIT8_M <= ST_BR_EN_BIT8_0;
else
  if rising_edge(CLK)
  then
    if (IN_NEXT_STATE = '1')
      then SV_BR_BIT0_M <= n_SV_BR_BIT0;
          SV_BR_BIT1_M <= n_SV_BR_BIT1;
          SV_BR_BIT2_M <= n_SV_BR_BIT2;
          SV_BR_BIT3_M <= n_SV_BR_BIT3;
          SV_BR_BIT4_M <= n_SV_BR_BIT4;
          SV_BR_BIT5_M <= n_SV_BR_BIT5;
          SV_BR_BIT6_M <= n_SV_BR_BIT6;
          SV_BR_BIT7_M <= n_SV_BR_BIT7;
          SV_BR_BIT8_M <= n_SV_BR_BIT8;
    else
      SV_BR_BIT0_M <= SV_BR_BIT0_M;
      SV_BR_BIT1_M <= SV_BR_BIT1_M;
      SV_BR_BIT2_M <= SV_BR_BIT2_M;
      SV_BR_BIT3_M <= SV_BR_BIT3_M;
      SV_BR_BIT4_M <= SV_BR_BIT4_M;
      SV_BR_BIT5_M <= SV_BR_BIT5_M;
      SV_BR_BIT6_M <= SV_BR_BIT6_M;
      SV_BR_BIT7_M <= SV_BR_BIT7_M;
      SV_BR_BIT8_M <= SV_BR_BIT8_M;
    end if;
  end if;
end if;
end process;

SREG_S_PROC: process (RESET, SV_BR_BIT0_M, SV_BR_BIT1_M, SV_BR_BIT2_M, SV_BR_BIT3_M, SV_BR_BIT4_M,
SV_BR_BIT5_M, SV_BR_BIT6_M, SV_BR_BIT7_M, SV_BR_BIT8_M, CLK) --Slave
begin
  if (RESET = '1')
    then SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
        SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
        SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
        SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
        SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
        SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
        SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
        SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
        SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
  else
    if falling_edge(CLK)
    then
      SV_BR_BIT0 <= SV_BR_BIT0_M;
      SV_BR_BIT1 <= SV_BR_BIT1_M;
      SV_BR_BIT2 <= SV_BR_BIT2_M;
      SV_BR_BIT3 <= SV_BR_BIT3_M;
      SV_BR_BIT4 <= SV_BR_BIT4_M;
      SV_BR_BIT5 <= SV_BR_BIT5_M;
      SV_BR_BIT6 <= SV_BR_BIT6_M;
      SV_BR_BIT7 <= SV_BR_BIT7_M;
      SV_BR_BIT8 <= SV_BR_BIT8_M;
    end if;
  end if;
end process;

BIT_REGISTER_EN_BIT_0_PROC:process (SV_BR_BIT0, n_SV_BR_BIT0, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit0
begin
  case SV_BR_BIT0 is
    when ST_BR_EN_BIT0_0 =>
      BYTE_OUT(0)<='0';
      BYTE_VEC(0)<='0';
      if (EN_BIT_i(0) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_EN_BIT0_1
            then n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1;
            else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
          end if;
        else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
      end if;
  end case;
end process;
```



```
when ST_BR_EN_BIT0_1 =>
-- EN_BIT_0_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(0) = 1
BYTE_OUT(0)<='1';
BYTE_VEC(0)<='1';
if (EN_BIT_i(0) = '1')
then
  if (BIT_VALUE = '1')
    then n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1;
    else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
    end if;
  else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1; -- BIT_VALUE = 0
end if;

when others =>
  n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
end case;
end process;

BIT_REGISTER_EN_BIT_1_PROC:process (SV_BR_BIT1, n_SV_BR_BIT1, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
case SV_BR_BIT1 is
when ST_BR_EN_BIT1_0 =>
  BYTE_OUT(1)<='0';
  BYTE_VEC(1)<='0';
  if (EN_BIT_i(1) = '1')
  then
    if (BIT_VALUE = '1')--gehe zu ST_BR_BIT1_1
      then n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1;
      else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
      end if;
    else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
  end if;
when ST_BR_EN_BIT1_1 =>
-- EN_BIT_1_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(1) = 1
  BYTE_OUT(1)<='1';
  BYTE_VEC(1)<='1';
  if (EN_BIT_i(1) = '1')
  then
    if (BIT_VALUE = '1')
      then n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1;
      else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
      end if;
    else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1; -- BIT_VALUE = 0
  end if;
when others =>
  n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
end case;
end process;

BIT_REGISTER_EN_BIT_2_PROC:process (SV_BR_BIT2, n_SV_BR_BIT2, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
case SV_BR_BIT2 is
when ST_BR_EN_BIT2_0 =>
  BYTE_OUT(2)<='0';
  BYTE_VEC(2)<='0';
  if (EN_BIT_i(2) = '1')
  then
    if (BIT_VALUE = '1')--gehe zu ST_BR_BIT2_1
      then n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1;
      else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
      end if;
    else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
  end if;
when ST_BR_EN_BIT2_1 =>
-- EN_BIT_2_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(2) = 1
  BYTE_OUT(2)<='1';
  BYTE_VEC(2)<='1';
  if (EN_BIT_i(2) = '1')
  then
    if (BIT_VALUE = '1')
      then n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1;
      else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
      end if;
```



```
else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1; -- BIT_VALUE = 0
end if;

when others =>
  n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
end case;
end process;

BIT_REGISTER_EN_BIT_3_PROC:process (SV_BR_BIT3, n_SV_BR_BIT3, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
  case SV_BR_BIT3 is
    when ST_BR_EN_BIT3_0 =>
      BYTE_OUT(3)<='0';
      BYTE_VEC(3)<='0';
      if (EN_BIT_i(3) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT3_1
            then n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1;
            else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
            end if;
        else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
        end if;
      when ST_BR_EN_BIT3_1 =>
-- EN_BIT_3_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(3) = 1
      BYTE_OUT(3)<='1';
      BYTE_VEC(3)<='1';
      if (EN_BIT_i(3) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1;
            else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
            end if;
        else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1; -- BIT_VALUE = 0
        end if;
      when others =>
        n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
      end case;
    end process;

BIT_REGISTER_EN_BIT_4_PROC:process (SV_BR_BIT4, n_SV_BR_BIT4, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
  case SV_BR_BIT4 is
    when ST_BR_EN_BIT4_0 =>
      BYTE_OUT(4)<='0';
      BYTE_VEC(4)<='0';
      if (EN_BIT_i(4) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT4_1
            then n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1;
            else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
            end if;
        else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
        end if;
      when ST_BR_EN_BIT4_1 =>
-- EN_BIT_4 = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(4) = 1
      BYTE_OUT(4)<='1';
      BYTE_VEC(4)<='1';
      if (EN_BIT_i(4) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1;
            else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
            end if;
        else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1; -- BIT_VALUE = 0
        end if;
      when others =>
        n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
      end case;
    end process;

BIT_REGISTER_EN_BIT_5_PROC:process (SV_BR_BIT5, n_SV_BR_BIT5, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
```



```
begin
  case SV_BR_BIT5 is
    when ST_BR_EN_BIT5_0 =>
      BYTE_OUT(5)<='0';
      BYTE_VEC(5)<='0';
      if (EN_BIT_i(5) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT5_1
            then n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1;
            else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
          end if;
        else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
      end if;

    when ST_BR_EN_BIT5_1 =>
      -- EN_BIT_5_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(5) = 1
      BYTE_OUT(5)<='1';
      BYTE_VEC(5)<='1';
      if (EN_BIT_i(5) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1;
            else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
          end if;
        else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1; -- BIT_VALUE = 0
      end if;

    when others =>
      n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
  end case;
end process;

BIT_REGISTER_EN_BIT_6_PROC:process (SV_BR_BIT6, n_SV_BR_BIT6, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit6
begin
  case SV_BR_BIT6 is
    when ST_BR_EN_BIT6_0 =>
      BYTE_OUT(6)<='0';
      BYTE_VEC(6)<='0';
      if (EN_BIT_i(6) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT6_1
            then n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1;
            else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
          end if;
        else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
      end if;

    when ST_BR_EN_BIT6_1 =>
      -- EN_BIT_6 = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(6) = 1
      BYTE_OUT(6)<='1';
      BYTE_VEC(6)<='1';
      if (EN_BIT_i(6) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1;
            else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
          end if;
        else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1; -- BIT_VALUE = 0
      end if;

    when others =>
      n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
  end case;
end process;

BIT_REGISTER_EN_BIT_7_PROC:process (SV_BR_BIT7, n_SV_BR_BIT7, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit7
begin
  case SV_BR_BIT7 is
    when ST_BR_EN_BIT7_0 =>
      BYTE_OUT(7)<='0';
      BYTE_VEC(7)<='0';
      if (EN_BIT_i(7) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT7_1
            then n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1;
            else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
```



```
    end if;
  else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end if;

when ST_BR_EN_BIT7_1 =>
-- EN_BIT_7_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(7) = 1
BYTE_OUT(7)<='1';
BYTE_VEC(7)<='1';
if (EN_BIT_i(7) = '1')
then
  if (BIT_VALUE = '1')
  then n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1;
  else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
  end if;
else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1; -- BIT_VALUE = 0
end if;

when others =>
  n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end case;
end process;

BIT_REGISTER_EN_BIT_8_PROC:process (SV_BR_BIT8, n_SV_BR_BIT8, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit8
begin
case SV_BR_BIT8 is
when ST_BR_EN_BIT8_0 =>
  BYTE_VEC(8)<='0';
  if (EN_BIT_i(8) = '1')
  then
    if (BIT_VALUE = '1')--gehe zu ST_BR_BIT8_1
    then n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1;
    else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
    end if;
  else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
  end if;

when ST_BR_EN_BIT8_1 =>
-- EN_BIT_8_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(8) = 1
BYTE_VEC(8)<='1';
if (EN_BIT_i(8) = '1')
then
  if (BIT_VALUE = '1')
  then n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1;
  else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
  end if;
else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1; -- BIT_VALUE = 0
end if;

when others =>
  n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
end case;
end process;

PARITY_CHECK_PROC: process (BYTE_VEC) --Paritätsprüfung (Mit VARIABLEN := , STATT SIGNALEN <=)
variable TMP00, TMP01, TMP02, TMP03, TMP10, TMP11, TMP20 : std_logic;
begin
  TMP00 := BYTE_VEC(0) xor BYTE_VEC(1);
  TMP01 := BYTE_VEC(2) xor BYTE_VEC(3);
  TMP02 := BYTE_VEC(4) xor BYTE_VEC(5);
  TMP03 := BYTE_VEC(6) xor BYTE_VEC(7);

  TMP10 := TMP00 xor TMP01;
  TMP11 := TMP02 xor TMP03;

  TMP20 := TMP10 xor TMP11;

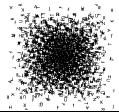
  if (TMP20 = BYTE_VEC(8))
  then PARITY_OK <= '1'; -- Parität korrekt
  else PARITY_OK <= '0'; -- Parität fehlerhaft
  end if;
end process;

--BYTE_OUT_PORC: process (BYTE_VEC) --BYTEausgabe
-- begin
--   BYTE_OUT(0) <= BYTE_VEC(0);
--   BYTE_OUT(1) <= BYTE_VEC(1);
```



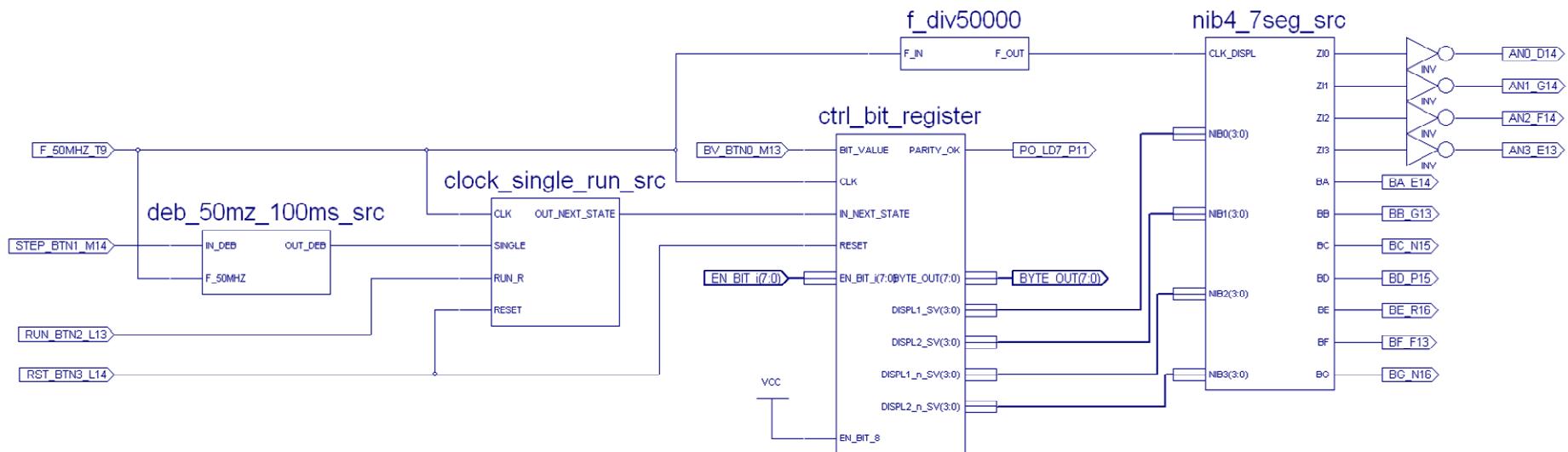
```
-- BYTE_OUT(2) <= BYTE_VEC(2);  
-- BYTE_OUT(3) <= BYTE_VEC(3);  
-- BYTE_OUT(4) <= BYTE_VEC(4);  
-- BYTE_OUT(5) <= BYTE_VEC(5);  
-- BYTE_OUT(6) <= BYTE_VEC(6);  
-- BYTE_OUT(7) <= BYTE_VEC(7);  
--end process;  
  
end Behavioral;
```

Datei 8-3:..\VHDL_Bausteine\CTRL_BIT_REGISTER\CTRL_BIT_REGISTER.vhd



8.2.6 Testumgebung

Um das BIT_REGSITER in der Testumgebung zu Testen wurden einige Anpassungen vorgenommen. EN_BIT_i wurde von 8 Bit auf 7 Bit verkleinert. Das Bit 8 wurde zu EN_BIT_8 und dauerhaft via VCC auf 1 gesetzt. Die Anzeige zeigt nur den Folgezustand und aktuellen Zustand von EN_BIT_i(0) also Bit 0 an. Für die Umsetzung in VHDL siehe 11.1 VHDL-Programm BIT_REGISTER aus Testumgebung, Seite 11-1.



CTRL_BIT_REGISTER
DEB_50MHZ_100MS_SRC
CLOCK_SINGLE_RUN_SRC
F_DIV50000
NIB4_7SEG_SRC
INV

Siehe 8.2 Modul BIT_REGISTER, Seite 8-25
Siehe /DEB_50MHZ_100MS/, Seite 12-18
Siehe /CLOCK_SINGLE_RUN/, Seite 12-18
Siehe /F_DIV50000/, Seite 12-18
Siehe /NIB_7SEG_SRC/, Seite 12-18
Siehe /INV/, Seite 12-18

Abbildung 8-9: Testumgebung BIT_REGISTER



8.2.7 UCF-Datei

#PACE: Start of Constraints generated by PACE

```
#PACE: Start of PACE I/O Pin Assignments
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "BV_BTN0_M13" LOC = "M13" ;
NET "BYTE_OUT<0>" LOC = "K12" ;
NET "BYTE_OUT<1>" LOC = "P14" ;
NET "BYTE_OUT<2>" LOC = "L12" ;
NET "BYTE_OUT<3>" LOC = "N14" ;
NET "BYTE_OUT<4>" LOC = "P13" ;
NET "BYTE_OUT<5>" LOC = "N12" ;
NET "BYTE_OUT<6>" LOC = "P12" ;
NET "EN_BIT_i<0>" LOC = "F12" ;
NET "EN_BIT_i<1>" LOC = "G12" ;
NET "EN_BIT_i<2>" LOC = "H14" ;
NET "EN_BIT_i<3>" LOC = "H13" ;
NET "EN_BIT_i<4>" LOC = "J14" ;
NET "EN_BIT_i<5>" LOC = "J13" ;
NET "EN_BIT_i<6>" LOC = "K14" ;
NET "EN_BIT_i<7>" LOC = "K13" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "PO_LD7_P11" LOC = "P11" ;
NET "RST_BTN3_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Datei 8-4..\VHDL_Bausteine\TEST_CTRL_BIT_REGISTER\test_ctrl_bit_register_sch.ucf



8.2.8 Steuerungsbelegung Testumgebung

Buttons	BTN0:	BIT_VALUE
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	BIT0
	SW1:	BIT1
	SW2:	BIT2
	SW3:	BIT3
	SW4:	BIT4
	SW5:	BIT5
	SW6:	BIT6
	SW7:	BIT7 >> BIT8 via VCC permanent auf 1 gesetzt
LEDs	LD0:	BYTE_OUT(0)
	LD1:	BYTE_OUT(1)
	LD2:	BYTE_OUT(2)
	LD3:	BYTE_OUT(3)
	LD4:	BYTE_OUT(4)
	LD5:	BYTE_OUT(5)
	LD6:	BYTE_OUT(6) >> Kein BYTE_OUT(7)
	LD7:	PARITY_OK

Tabelle 8-8: Steuerungsbelegung Testumgebung BIT_REGISTER

8.2.9 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand für Bit 0 des Funktionsmoduls BIT_REGISTER verkürzt aus. Siehe auch **Tabelle 8-7: Liste der Prozesse BIT_REGISTER**, Seite 8-27 und **Tabelle 8-6: Variablendefinition BIT_REGSITER**, Seite 8-26.



8.3 Modul TELEGRAM_CHECK

Das Funktionsmodul TELEGRAM_CHECK wandelt das die Bytes des Eingangsignals BYTE_IN in ein Telegramm um. Es ermittelt den Telegrammtyp, die Telegrammlänge, das Telegrammende und erkennt Fehler im Telegrammaufbau.

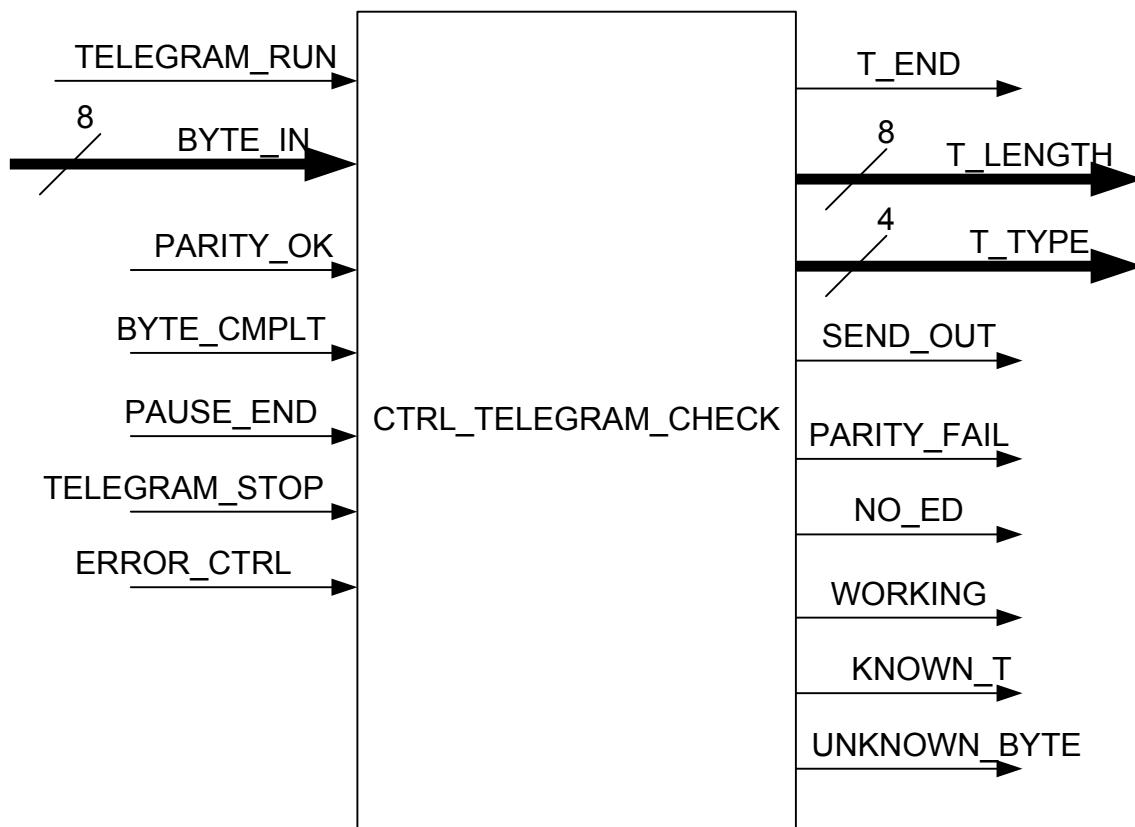


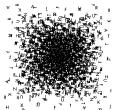
Abbildung 8-10: Wirkungsplan TELEGRAM_CHECK

Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
TELEGRAM_RUN	BOOL	Eingang	1: nächstes Telegramm
BYTE_IN	VECTOR, 8 Bit	Eingang	Eingang Datenbyte
PARITY_OK	BOOL	Eingang	1: Parität ist korrekt
BYTE_CMPLT	BOOL	Eingang	1: Byte komplett empfangen
PAUSE_END	BOOL	Eingang	1: 33 Bit SYN beendet
TELEGRAM_STOP	BOOL	Eingang	1: Stopp nach einem Telegramm
ERROR_CTRL	BOOL	Eingang	1: nach Fehler fortfahren
T_END	BOOL	Ausgang	1: Telegramm Ende erreicht
T_LENGTH	VECTOR, 8 Bit	Ausgang	Telegrammlänge
T_TYPE	VECTOR, 4 Bit	Ausgang	Telegrammtyp (0000: kein Telegramm erkannt, 0001: Telegrammtyp SD1 0010: Telegrammtyp SD2 0011: Telegrammtyp SD3 0100: Telegrammtyp SD4 1000: Telegrammtyp SC)
SEND_OUT	BOOL	Ausgang	1: Senden



Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
PARITY_FAIL	BOOL	Ausgang	1: Paritätsfehler
NO_ED	BOOL	Ausgang	1: Kein Enddelimiter festgestellt
WORKING	BOOL	Ausgang	1: Modul TELEGRAM_CHECK arbeitet
KNOWN_T	BOOL	Ausgang	1: Telegramm erkannt
UNKNOWN_BYTE	BOOL	Ausgang	1: Byte nicht erkannt
COUNT	VECTOR, 8 Bit	intern	Telegrammlänge
ST_TC	ENUM	intern	Zustandsvariable (ST_TC_00, ST_TC_01, ST_TC_02, ST_TC_03, ST_TC_04, ST_TC_05, ST_TC_06, ST_TC_07, ST_TC_08, ST_TC_09, ST_TC_10, ST_TC_11)

Tabelle 8-9: Variablendefinition TELEGRAM_CHECK



8.3.1 Verhaltensbeschreibung

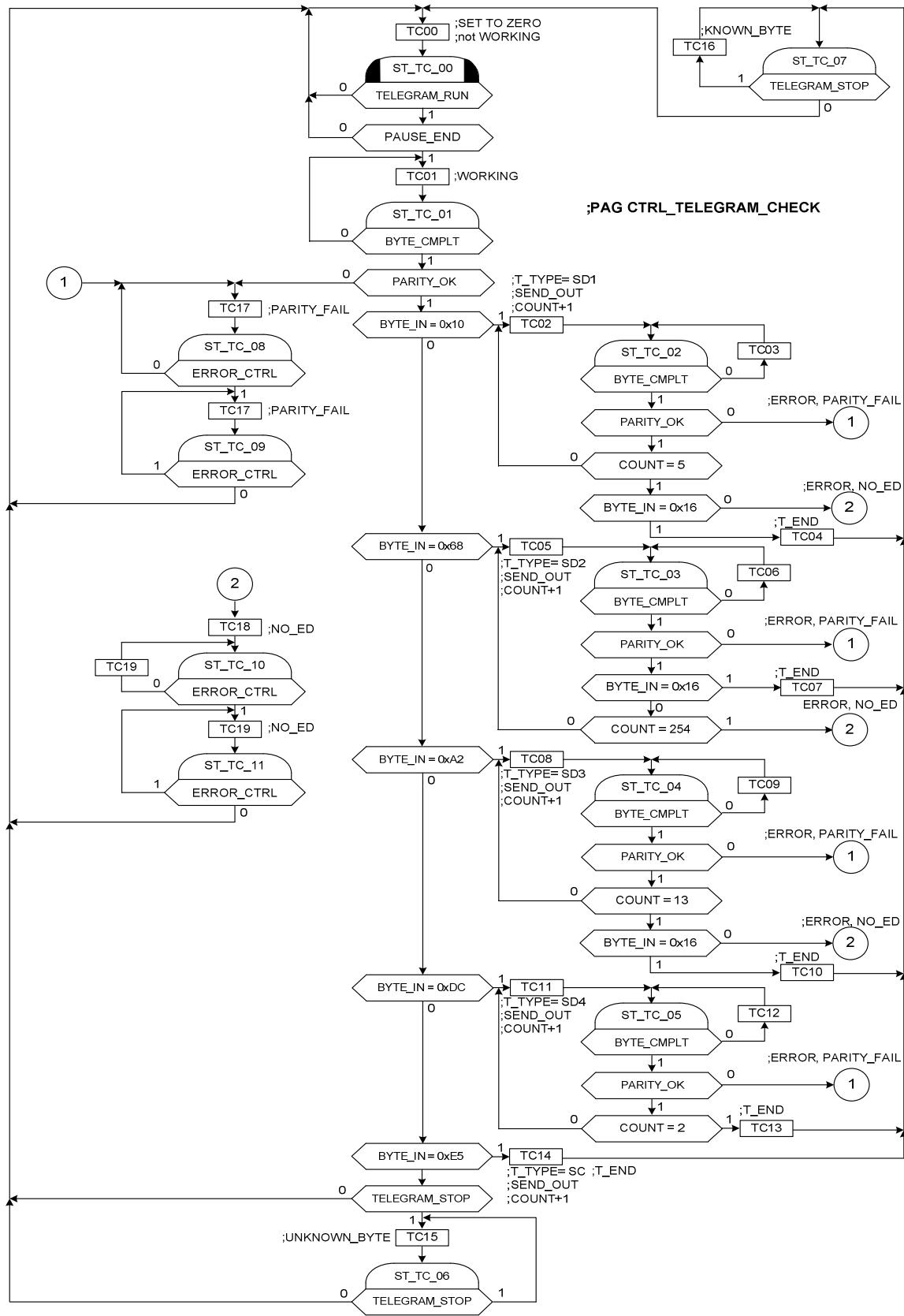
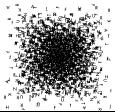


Abbildung 8-11: Programmablaufgraph TELEGRAM_CHECK



8.3.2 Belegung Ausgangsvariablen

Bezeichner	n_COUNT	T_LENGTH	SEND_OUT	T_END	TELEGRAM_TYPE				WORKING	PARITY_FAIL	NO_ED	KNOWN_T	UNKNOWN_BYTE
					BIT 0	BIT 1	BIT 2	BIT 3					
TC00	0	0	0	0	0	0	0	0	0	0	0	0	0
TC01	:=COUNT	0	0	0	0	0	0	0	1	0	0	0	0
TC02	:=COUNT+1	:=COUNT	1	0	0	0	0	1	1	0	0	0	0
TC03	:=COUNT	:=COUNT	0	0	0	0	0	1	1	0	0	0	0
TC04	:=COUNT	:=COUNT	1	1	0	0	0	1	1	0	0	0	0
TC05	:=COUNT+1	:=COUNT	1	0	0	0	1	0	1	0	0	0	0
TC06	:=COUNT	:=COUNT	0	0	0	0	1	0	1	0	0	0	0
TC07	:=COUNT	:=COUNT	1	1	0	0	1	0	1	0	0	0	0
TC08	:=COUNT+1	:=COUNT	1	0	0	0	1	1	1	0	0	0	0
TC09	:=COUNT	:=COUNT	0	0	0	0	1	1	1	0	0	0	0
TC10	:=COUNT	:=COUNT	1	1	0	0	1	1	1	0	0	0	0
TC11	:=COUNT+1	:=COUNT	1	0	0	1	0	0	1	0	0	0	0
TC12	:=COUNT	:=COUNT	0	0	0	1	0	0	1	0	0	0	0
TC13	:=COUNT	:=COUNT	1	1	0	1	0	0	1	0	0	0	0
TC14	:=COUNT+1	:=COUNT	1	1	1	0	0	0	1	0	0	0	0
TC15	:=COUNT	:=COUNT	0	0	0	0	0	0	0	0	0	0	1
TC16	:=COUNT	:=COUNT	0	0	0	0	0	0	0	0	0	1	0
TC17	:=COUNT	:=COUNT	0	0	0	0	0	0	0	1	0	0	0
TC18	:=COUNT	:=COUNT	1	0	0	0	0	0	0	0	1	0	0
TC19	:=COUNT	:=COUNT	0	0	0	0	0	0	0	0	1	0	0

Tabelle 8-10: Belegung Ausgangsvariablen TELEGRAM_CHECK

8.3.3 Auswahl Automatentyp

Siehe **Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister**, Seite 8-8.

Der einziger Unterschied ist, dass kein Eingangsregister verwendet wurde. IREG und n_X entfallen. Die Eingangsvariable X geht direkt in die Eingangs- /Ausgangslogik IL/OL.

8.3.4 Liste der Prozesse

Prozessname	Funktion
SREG_M_PROC	Zustandsregister, Master
SREG_S_PROC	Zustandsregister, Slave
TELEGRAM_CHECK_PROC	Eingangs- / Ausgangslogik
STATE_DISPL_PROC	Ausgabe aktueller und folgender Zustand oder aktueller Zustand und Telegrammlänge

Tabelle 8-11: Liste der Prozesse TELEGRAM_CHECK



8.3.5 VHDL-Programm

```
-- CTRL_TELEGRAM_CHECK
-- Profibus Telegramtyp ermitteln, aktuelle Laenge und Telegram komplett anzeigen
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 02.01.2013
-- Bearbeiter: mharndt
-- Geaendert: 28.01.2013
-- Umstellung auf: rising_edge(CLK) und falling_edge(CLK) und http://www.sigasi.com/content/clock-
edge-detection
-- Optimierungen aus: http://www.lothar-miller.de/s9y/categories/37-FSM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_TELEGRAM_CHECK is

    Port (TELEGRAM_RUN : in std_logic; --Eingangsvariable, Naechstes Telegram
          BYTE_IN : in std_logic_vector (7 downto 0); --Eingangsvariable, Byte, 8bit
          PARITY_OK : in std_logic; --Eingangsvariable, Paritaet i.O.
          BYTE_CMPLT : in std_logic; --Eingangsvariable, BYTE komplett empfangen
          PAUSE_END : in std_logic; --Eingangsvariable, Pause erkannt und beendet
          TELEGRAM_STOP : in std_logic; --Eingangsvariable, nach Telegramm stoppen
          ERROR_CTRL : in std_logic; --Eingangsvariable, Fehlerkontrolle

          T_END : out std_logic; --Ausgangsvariable, Telegramm zu Ende
          T_LENGTH : out std_logic_vector (7 downto 0); --Ausgangsvariable, Telegramlaenge, 8bit
          T_TYPE : out std_logic_vector (3 downto 0); --Ausgangsvariable, Telegramtyp, 4bit
          SEND_OUT : out std_logic; --Ausgangsvariable, Senden
          PARITY_FAIL : out std_logic; --Ausgangsvariable, Paritaetspruefung fehlerhaft
          NO_ED : out std_logic; --Ausgangsvariable, kein Enddelimiter festgestellt
          WORKING : out std_logic; --Ausgangsvariable, TELEGRAM_CHECK arbeitet
          KNOWN_T : out std_logic; --Ausgangsvariable, Telegramm erkannt
          UNKNOWN_BYTE : out std_logic; --Ausgangsvariable, BYTE nicht erkannt

          CLK : in std_logic; --Taktvariable

          IN_NEXT_STATE: in std_logic; --1:Zustandsuebergang möglich
          RESET : in std_logic; --1: Initialzustand annehmen

          DISPL_COUNT : in std_logic; --Eingangsvariable, Zaehler anzeigen
          DISPL1_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl1, binärzahl
          DISPL2_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl2, binärzahl
          DISPL1_n_SV : out std_logic_vector (3 downto 0); --Folgezustand Zahl1, binärzahl
          DISPL2_n_SV : out std_logic_vector (3 downto 0)); --Folgezustand Zahl2, binärzahl

    end CTRL_TELEGRAM_CHECK;

architecture Behavioral of CTRL_TELEGRAM_CHECK is

    type TYPE_STATE is
        (ST_TC_00, --Zustaende TELEGRAM_CHECK
         ST_TC_01,
         ST_TC_02,
         ST_TC_03,
         ST_TC_04,
         ST_TC_05,
         ST_TC_06,
         ST_TC_07,
         ST_TC_08,
         ST_TC_09,
         ST_TC_10,
         ST_TC_11);

    signal SV : TYPE_STATE := ST_TC_00; --Zustandsvariable
    signal n_SV: TYPE_STATE := ST_TC_00; --Zustandsvariable, neuer Wert
    signal SV_M: TYPE_STATE := ST_TC_00; --Zustandsvariable, Ausgang Master

    signal COUNT : std_logic_vector (7 downto 0) := x"00"; -- Vektor, Telegrammlaenge, 8bit
    signal n_COUNT : std_logic_vector (7 downto 0) := x"00"; -- Vektor, Telegrammlaenge, 8bit, neuer Wert
    signal COUNT_M : std_logic_vector (7 downto 0) := x"00"; -- Vektor, Telegrammlaenge, 8bit, Ausgang Master
```



```
signal STATE_SV    : std_logic_vector (7 downto 0) :=  x"00"; -- aktueller Zustand in 8 Bit, binär
signal STATE_n_SV : std_logic_vector (7 downto 0) :=  x"00"; -- Folgezustand in 8 Bit, binär

begin

SREG_M_PROC: process (RESET, n_SV, n_COUNT, CLK) --Master
begin
  if (RESET = '1')
  then SV_M <= ST_TC_00;
     COUNT_M <= x"00";
  else
    if rising_edge(CLK)
    then
      if (IN_NEXT_STATE = '1')
      then
        SV_M <= n_SV;
        COUNT_M <= n_COUNT;
      else
        SV_M <= SV_M;
        COUNT_M <= COUNT_M;
      end if;
    end if;
  end if;
end process;

SREG_S_PROC: process (RESET, SV_M, CLK) --Slave
begin
  if (RESET = '1')
  then
    SV <= ST_TC_00;
    COUNT <= x"00";
  else
    if falling_edge(CLK)
    then
      SV <= SV_M;
      COUNT <= COUNT_M;
    end if;
  end if;
end process;

TELEGRAM_CHECK_PROC: process (SV, COUNT, TELEGRAM_RUN, PAUSE_END, BYTE_CMPLT, PARITY_OK, BYTE_IN,
TELEGRAM_STOP, ERROR_CTRL) --Telegramme erkennen und Ende Telegram erkennen und ausgeben
begin
  case SV is
    when ST_TC_00 =>
      if (TELEGRAM_RUN = '1')
      then
        if (PAUSE_END = '1')
        then
          --TC01
          n_COUNT <= COUNT; --Zähler erhöhen
          T_END <= '0';
          T_LENGTH <= COUNT;
          T_TYPE <= "0000";
          SEND_OUT <= '0';
          PARITY_FAIL <= '0';
          NO_ED <= '0';
          WORKING <= '1'; --arbeitet
          KNOWN_T <= '0';
          UNKNOWN_BYTE <= '0';
          n_SV <= ST_TC_01; --Zustandsübergang
        else
          --TC00
          n_COUNT <= x"00";
          T_END <= '0';
          T_LENGTH <= x"00";
          T_TYPE <= "0000";
          SEND_OUT <= '0';
          PARITY_FAIL <= '0';
          NO_ED <= '0';
          WORKING <= '0';
          KNOWN_T <= '0';
          UNKNOWN_BYTE <= '0';
          n_SV <= ST_TC_00;
        end if;
      else --TELEGRAM_RUN = '0'
        -- TC00
      end if;
    end case;
  end if;
end process;
```



```
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00;
end if;

when ST_TC_01 =>
if (BYTE_CMPLT = '1')
then
if (PARITY_OK = '1')
then
if (BYTE_IN = x"10") --SD1 erkannt
then
--TC02
n_COUNT <= COUNT+1; --Zähler erhöhen
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0001"; --SD1
SEND_OUT <= '1'; --senden
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '1'; --arbeitet
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_02; --Zustandsübergang
else
if (BYTE_IN = x"68") --SD2 erkannt
then
--TC05
n_COUNT <= COUNT+1; --Zähler erhöhen
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0010"; --SD2
SEND_OUT <= '1'; --senden
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '1'; --arbeitet
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_03; --Zustandsübergang
else
if (BYTE_IN = x"A2") --SD3 erkannt
then
--TC08
n_COUNT <= COUNT+1; --Zähler erhöhen
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0011"; --SD3
SEND_OUT <= '1'; --senden
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '1'; --arbeitet
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_04; --Zustandsübergang
else
if (BYTE_IN = x"DC") --SD4 erkannt
then
--TC11
n_COUNT <= COUNT+1; --Zähler erhöhen
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0100"; --SD4
SEND_OUT <= '1'; --senden
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '1'; --arbeitet
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_05; --Zustandsübergang
else
```



```
if (BYTE_IN = x"E5") --SC erkannt
then
    --TC14
    n_COUNT <= COUNT+1; --Zaehler erhöhen
    T_END <= '1'; --Telegram Ende
    T_LENGTH <= COUNT;
    T_TYPE <= "1000"; --SC
    SEND_OUT <= '1'; --senden
    PARITY_FAIL <= '0';
    NO_ED <= '0';
    WORKING <= '1'; --arbeitet
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '0';
    n_SV <= ST_TC_07; --Zustandsübergang
else
    if (TELEGRAM_STOP = '1')
        then
            --TC15
            n_COUNT <= COUNT; --Zaehler bleibt gleich
            T_END <= '0'; --Telegram Ende
            T_LENGTH <= COUNT;
            T_TYPE <= "0000";
            SEND_OUT <= '0';
            PARITY_FAIL <= '0';
            NO_ED <= '0';
            WORKING <= '0';
            KNOWN_T <= '0';
            UNKNOWN_BYTE <= '1'; --Unbekanntes BYTE
            n_SV <= ST_TC_06; --Zustandsübergang
        else
            --TC00
            n_COUNT <= x"00";
            T_END <= '0';
            T_LENGTH <= x"00";
            T_TYPE <= "0000";
            SEND_OUT <= '0';
            PARITY_FAIL <= '0';
            NO_ED <= '0';
            WORKING <= '0';
            KNOWN_T <= '0';
            UNKNOWN_BYTE <= '0';
            n_SV <= ST_TC_00;
        end if; --TELEGRAM_STOP
    end if; --BYTE_IN =x"E5"
    end if; --BYTE_IN = x"DC"
    end if; --BYTE_IN = x"A2"
    end if; --BYTE_IN = x"68"
end if; --BYTE_IN = x"10"
else ----PARITY_OK = '0'
--TC17
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '1'; --Paritaets Fehler
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_08; --Zustandsuebergang
end if; --PARITY_OK = '1'
else --BYTE_CMPLT = '0'
--TC01
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '1'; --laeuft
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_01;
end if; --BYTE_CMPLT = '1'

when ST_TC_02 =>
```



```
if (BYTE_CMPLT = '1')
then
  if (PARITY_OK = '1')
  then
    if (COUNT = x"05")
    then
      if (BYTE_IN = x"16")
      then
        --TC04
        n_COUNT <= COUNT;
        T_END <= '1'; --Telegrammende erkannt
        T_LENGTH <= COUNT;
        T_TYPE <= "0001"; --SD1
        SEND_OUT <= '1'; --senden
        PARITY_FAIL <= '0';
        NO_ED <= '0';
        WORKING <= '1'; --arbeitet
        KNOWN_T <= '0';
        UNKNOWN_BYTE <= '0';
        n_SV <= ST_TC_07; --Zustandsuebergang
    else
      --TC18
      n_COUNT <= COUNT;
      T_END <= '0';
      T_LENGTH <= COUNT;
      T_TYPE <= "0000";
      SEND_OUT <= '1'; --senden
      PARITY_FAIL <= '0';
      NO_ED <= '1'; --kein Enddelimiter
      WORKING <= '0';
      KNOWN_T <= '0';
      UNKNOWN_BYTE <= '0';
      n_SV <= ST_TC_10; --Zustandsuebergang
    end if; --BYTE_IN = x"16"
  else --not COUNT = x"05"
    --TC02
    n_COUNT <= COUNT+1;
    T_END <= '0';
    T_LENGTH <= COUNT;
    T_TYPE <= "0001"; --SD1
    SEND_OUT <= '1'; --senden
    PARITY_FAIL <= '0';
    NO_ED <= '0';
    WORKING <= '1'; --arbeitet
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '0';
    n_SV <= ST_TC_02;
  end if; --COUNT = x"05"
else --PARITY_OK = '0'
  --TC17
  n_COUNT <= COUNT;
  T_END <= '0';
  T_LENGTH <= COUNT;
  T_TYPE <= "0000";
  SEND_OUT <= '0';
  PARITY_FAIL <= '1'; --Paritaets Fehler
  NO_ED <= '0';
  WORKING <= '0';
  KNOWN_T <= '0';
  UNKNOWN_BYTE <= '0';
  n_SV <= ST_TC_08; --Zustandsuebergang
end if; --PARITY_OK = '1'
else --BYTE_CMPLT = '0'
  --TC03
  n_COUNT <= COUNT;
  T_END <= '0';
  T_LENGTH <= COUNT;
  T_TYPE <= "0001"; --SD1
  SEND_OUT <= '0';
  PARITY_FAIL <= '0';
  NO_ED <= '0';
  WORKING <= '1'; --laeuft
  KNOWN_T <= '0';
  UNKNOWN_BYTE <= '0';
  n_SV <= ST_TC_02;
end if; --BYTE_CMPLT = '1'

when ST_TC_03 =>
```



```
if (BYTE_CMPLT = '1')
then
  if (PARITY_OK = '1')
  then
    if (BYTE_IN = x"16")
    then
      --TC07
      n_COUNT <= COUNT;
      T_END <= '1'; --Telegrammende erkannt
      T_LENGTH <= COUNT;
      T_TYPE <= "0010"; --SD2
      SEND_OUT <= '1'; --senden
      PARITY_FAIL <= '0';
      NO_ED <= '0';
      WORKING <= '1'; --laeuft
      KNOWN_T <= '0';
      UNKNOWN_BYTE <= '0';
      n_SV <= ST_TC_07; --Zustandsuebergang
    else
      if (COUNT = x"FE") --254
      then
        --TC18
        n_COUNT <= COUNT;
        T_END <= '0';
        T_LENGTH <= COUNT;
        T_TYPE <= "0000";
        SEND_OUT <= '1'; --senden
        PARITY_FAIL <= '0';
        NO_ED <= '1'; --kein Enddelimiter
        WORKING <= '0';
        KNOWN_T <= '0';
        UNKNOWN_BYTE <= '0';
        n_SV <= ST_TC_10; --Zustandsuebergang
      else --not COUNT = x"FE"
        --TC05
        n_COUNT <= COUNT+1;
        T_END <= '0';
        T_LENGTH <= COUNT;
        T_TYPE <= "0010"; --SD2
        SEND_OUT <= '1'; --senden
        PARITY_FAIL <= '0';
        NO_ED <= '0';
        WORKING <= '1'; --laeuft
        KNOWN_T <= '0';
        UNKNOWN_BYTE <= '0';
        n_SV <= ST_TC_03;
      end if; --COUNT = x"FE"
    end if; --BYTE_IN = x"16"
  else --PARITY_OK = '0'
    --TC17
    n_COUNT <= COUNT;
    T_END <= '0';
    T_LENGTH <= COUNT;
    T_TYPE <= "0000";
    SEND_OUT <= '0';
    PARITY_FAIL <= '1'; --Paritaets Fehler
    NO_ED <= '0';
    WORKING <= '0';
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '0';
    n_SV <= ST_TC_08; --Zustandsuebergang
  end if; --PARITY_OK = '1'
else --BYTE_CMPLT = '0'
  --TC06
  n_COUNT <= COUNT;
  T_END <= '0';
  T_LENGTH <= COUNT;
  T_TYPE <= "0010"; --SD2
  SEND_OUT <= '0';
  PARITY_FAIL <= '0';
  NO_ED <= '0';
  WORKING <= '1'; --laeuft
  KNOWN_T <= '0';
  UNKNOWN_BYTE <= '0';
  n_SV <= ST_TC_03;
end if; --BYTE_CMPLT = '1'

when ST_TC_04 =>
```



```
if (BYTE_CMPLT = '1')
then
  if (PARITY_OK = '1')
  then
    if (COUNT = x"0D") --13
    then
      if (BYTE_IN = x"16")
      then
        --TC10
        n_COUNT <= COUNT;
        T_END <= '1'; --Telegrammende erkannt
        T_LENGTH <= COUNT;
        T_TYPE <= "0011"; --SD3
        SEND_OUT <= '1'; --senden
        PARITY_FAIL <= '0';
        NO_ED <= '0';
        WORKING <= '1'; --laeuft
        KNOWN_T <= '0';
        UNKNOWN_BYTE <= '0';
        n_SV <= ST_TC_07; --Zustandsuebergang
      else --not BYTE_IN = x"16"
        --TC18
        n_COUNT <= COUNT;
        T_END <= '0';
        T_LENGTH <= COUNT;
        T_TYPE <= "0000";
        SEND_OUT <= '1'; --senden
        PARITY_FAIL <= '0';
        NO_ED <= '1'; --kein Enddelimiter
        WORKING <= '0';
        KNOWN_T <= '0';
        UNKNOWN_BYTE <= '0';
        n_SV <= ST_TC_10; --Zustandsuebergang
      end if; --BYTE_IN = x"16"
    else --not COUNT = x"0D"
      --TC08
      n_COUNT <= COUNT+1;
      T_END <= '0';
      T_LENGTH <= COUNT;
      T_TYPE <= "0011"; --SD3
      SEND_OUT <= '1'; --senden
      PARITY_FAIL <= '0';
      NO_ED <= '0';
      WORKING <= '1'; --laeuft
      KNOWN_T <= '0';
      UNKNOWN_BYTE <= '0';
      n_SV <= ST_TC_04;
    end if; --COUNT = x"0D"
  else --PARITY_OK = '0'
    --TC17
    n_COUNT <= COUNT;
    T_END <= '0';
    T_LENGTH <= COUNT;
    T_TYPE <= "0000";
    SEND_OUT <= '0';
    PARITY_FAIL <= '1'; --Paritaets Fehler
    NO_ED <= '0';
    WORKING <= '0';
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '0';
    n_SV <= ST_TC_08; --Zustandsuebergang
  end if; --PARITY_OK = '1'
else --BYTE_CMPLT = '0'
  --TC09
  n_COUNT <= COUNT;
  T_END <= '0';
  T_LENGTH <= COUNT;
  T_TYPE <= "0011"; --SD3
  SEND_OUT <= '0';
  PARITY_FAIL <= '0';
  NO_ED <= '0';
  WORKING <= '1'; --laeuft
  KNOWN_T <= '0';
  UNKNOWN_BYTE <= '0';
  n_SV <= ST_TC_04;
end if; --BYTE_CMPLT = '1'

when ST_TC_05 =>
```



```
if (BYTE_CMPLT = '1')
then
  if (PARITY_OK = '1')
  then
    if (COUNT = x"02") --bei Byte 3
    then
      --TC13
      n_COUNT <= COUNT;
      T_END <= '1'; --Telegrammende erkannt
      T_LENGTH <= COUNT;
      T_TYPE <= "0100"; --SD4
      SEND_OUT <= '1'; --senden
      PARITY_FAIL <= '0';
      NO_ED <= '0';
      WORKING <= '1'; --laeuft
      KNOWN_T <= '0';
      UNKNOWN_BYTE <= '0';
      n_SV <= ST_TC_07; --Zustandsuebergang
    else --not COUNT = x"02"
      --TC11
      n_COUNT <= COUNT+1;
      T_END <= '0';
      T_LENGTH <= COUNT;
      T_TYPE <= "0100"; --SD4
      SEND_OUT <= '1'; --senden
      PARITY_FAIL <= '0';
      NO_ED <= '0';
      WORKING <= '1'; --laeuft
      KNOWN_T <= '0';
      UNKNOWN_BYTE <= '0';
      n_SV <= ST_TC_05;
    end if; --COUNT = x"02"
  else --PARITY_OK = '0'
    --TC17
    n_COUNT <= COUNT;
    T_END <= '0';
    T_LENGTH <= COUNT;
    T_TYPE <= "0000";
    SEND_OUT <= '0';
    PARITY_FAIL <= '1'; --Paritaets Fehler
    NO_ED <= '0';
    WORKING <= '0';
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '0';
    n_SV <= ST_TC_08; --Zustandsuebergang
  end if; --PARITY_OK = '1'
else --BYTE_CMPLT = '0'
  --TC12
  n_COUNT <= COUNT;
  T_END <= '0';
  T_LENGTH <= COUNT;
  T_TYPE <= "0100"; --SD4
  SEND_OUT <= '0';
  PARITY_FAIL <= '0';
  NO_ED <= '0';
  WORKING <= '1'; --laeuft
  KNOWN_T <= '0';
  UNKNOWN_BYTE <= '0';
  n_SV <= ST_TC_05;
end if; --BYTE_CMPLT = '1'

when ST_TC_06 =>
  if (TELEGRAM_STOP = '1')
  then
    --TC15
    n_COUNT <= COUNT;
    T_END <= '0';
    T_LENGTH <= COUNT;
    T_TYPE <= "0000";
    SEND_OUT <= '0';
    PARITY_FAIL <= '0';
    NO_ED <= '0';
    WORKING <= '0';
    KNOWN_T <= '0';
    UNKNOWN_BYTE <= '1'; --Kein bekanntes Startdelimiter-Byte gefunden
    n_SV <= ST_TC_06;
  else
    --TC00
```



```
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00; --Zustandsuebergang
end if;

when ST_TC_07 =>
if (TELEGRAM_STOP = '1')
then
--TC16
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '1'; --Bekanntes Telegramm gefunden
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_07;
else
--TC00
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00; --Zustandsuebergang
end if;

when ST_TC_08 =>
if (ERROR_CTRL = '1')
then
--TC17
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '1'; --Paritaetsfehler festgestellt
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_09; --Zustandsuebergang
else
--TC17
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '1'; --Paritaetsfehler festgestellt
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_08;
end if;

when ST_TC_09 =>
if (ERROR_CTRL = '1')
then
--TC17
n_COUNT <= COUNT;
```



```
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '1'; --Paritaetsfehler festgestellt
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_09;
else
-- TC00
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00; --Zustandsuebergang
end if;

when ST_TC_10 =>
if (ERROR_CTRL = '1')
then
--TC19
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '1'; --Fehlendes Enddelimiter festgestellt
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_11; --Zustandsuebergang
else
--TC19
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '1'; --Fehlendes Enddelimiter festgestellt
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_10;
end if;

when ST_TC_11 =>
if (ERROR_CTRL = '1')
then
--TC19
n_COUNT <= COUNT;
T_END <= '0';
T_LENGTH <= COUNT;
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '1'; --Fehlendes Enddelimiter festgestellt
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_11;
else
-- TC00
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
```



```
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00; --Zustandsuebergang
end if;

when others =>
-- TC00
n_COUNT <= x"00";
T_END <= '0';
T_LENGTH <= x"00";
T_TYPE <= "0000";
SEND_OUT <= '0';
PARITY_FAIL <= '0';
NO_ED <= '0';
WORKING <= '0';
KNOWN_T <= '0';
UNKNOWN_BYTE <= '0';
n_SV <= ST_TC_00;
end case;
end process;

STATE_DISPL_PROC: process (SV, n_SV, STATE_SV, STATE_n_SV, DISPL_COUNT, COUNT) -- Zustandsanzeige
begin
STATE_SV <= conv_std_logic_vector(TYPE_STATE'pos(SV),8); --Zustandsumwandlung in 8 Bit
STATE_n_SV <= conv_std_logic_vector(TYPE_STATE'pos(n_SV),8);

DISPL1_SV(0) <= STATE_SV(0); --Bit0
DISPL1_SV(1) <= STATE_SV(1); --Bit1
DISPL1_SV(2) <= STATE_SV(2); --Bit2
DISPL1_SV(3) <= STATE_SV(3); --Bit3

DISPL2_SV(0) <= STATE_SV(4); --usw.
DISPL2_SV(1) <= STATE_SV(5);
DISPL2_SV(2) <= STATE_SV(6);
DISPL2_SV(3) <= STATE_SV(7);

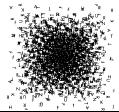
if (DISPL_COUNT = '1')
then
-- Zaehler anzeigen
DISPL1_n_SV(0) <= COUNT(0);
DISPL1_n_SV(1) <= COUNT(1);
DISPL1_n_SV(2) <= COUNT(2);
DISPL1_n_SV(3) <= COUNT(3);

DISPL2_n_SV(0) <= COUNT(4);
DISPL2_n_SV(1) <= COUNT(5);
DISPL2_n_SV(2) <= COUNT(6);
DISPL2_n_SV(3) <= COUNT(7);
else
--Folgezustand anzeigen
DISPL1_n_SV(0) <= STATE_n_SV(0);
DISPL1_n_SV(1) <= STATE_n_SV(1);
DISPL1_n_SV(2) <= STATE_n_SV(2);
DISPL1_n_SV(3) <= STATE_n_SV(3);

DISPL2_n_SV(0) <= STATE_n_SV(4);
DISPL2_n_SV(1) <= STATE_n_SV(5);
DISPL2_n_SV(2) <= STATE_n_SV(6);
DISPL2_n_SV(3) <= STATE_n_SV(7);
end if;
end process;

end Behavioral;
```

Datei 8-5: ..\VHDL_Bausteine\CTRL_TELEGRAM_CHECK\CTRL_TELEGRAM_CHECK.vhd



8.3.6 Testumgebung

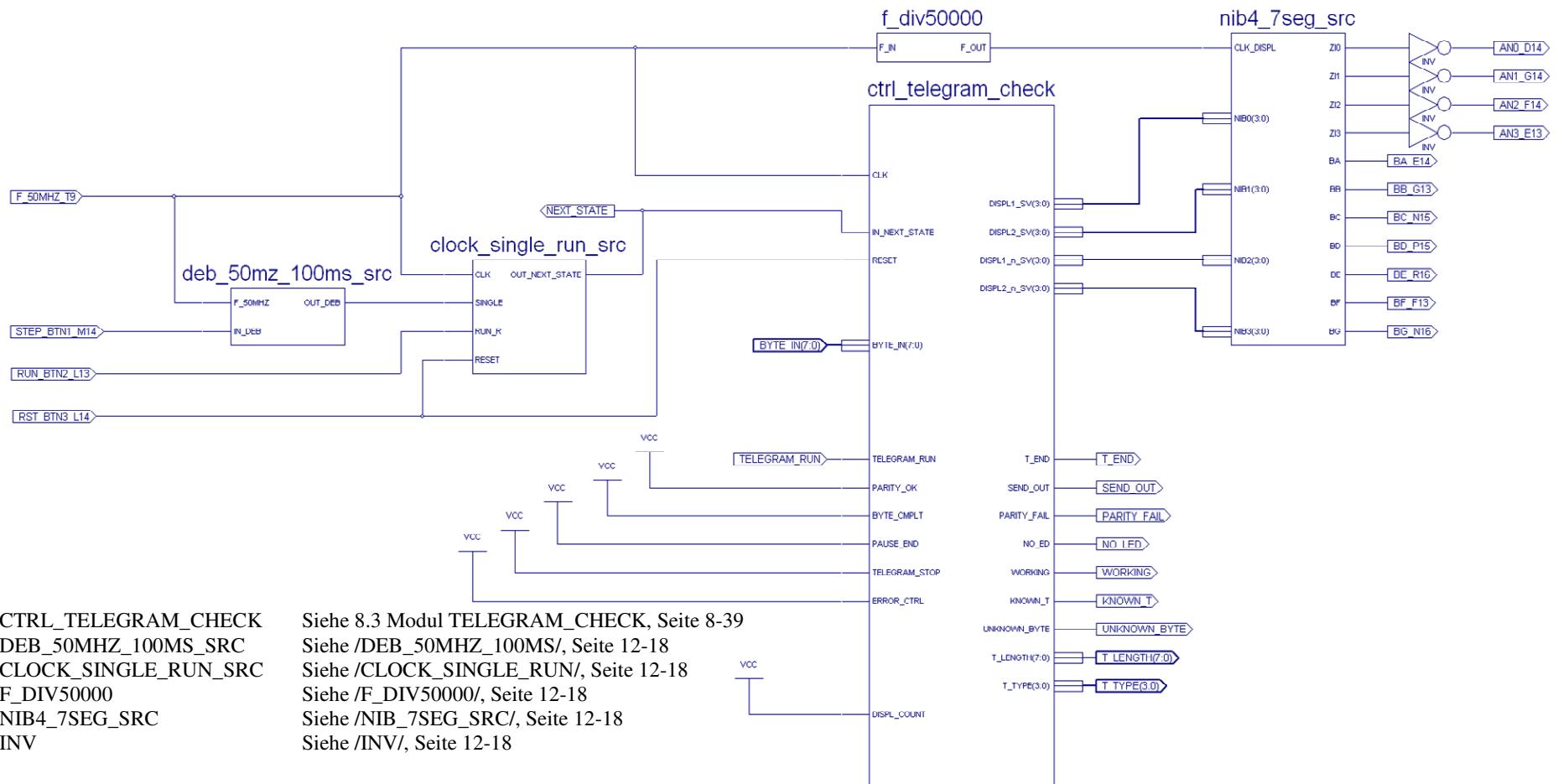


Abbildung 8-12: Testumgebung TELEGRAM_CHECK



8.3.7 UCF-Datei

#PACE: Start of Constraints generated by PACE

```
#PACE: Start of PACE I/O Pin Assignments
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "BYTE_IN<0>" LOC = "F12" ;
NET "BYTE_IN<1>" LOC = "G12" ;
NET "BYTE_IN<2>" LOC = "H14" ;
NET "BYTE_IN<3>" LOC = "H13" ;
NET "BYTE_IN<4>" LOC = "J14" ;
NET "BYTE_IN<5>" LOC = "J13" ;
NET "BYTE_IN<6>" LOC = "K14" ;
NET "BYTE_IN<7>" LOC = "K13" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "KNOWN_T" LOC = "P13" ;
NET "NEXT_STATE" LOC = "K12" ;
NET "NO_LED" LOC = "P14" ;
NET "PARITY_FAIL" LOC = "L12" ;
NET "RST_BTN3_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "SEND_OUT" LOC = "N12" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
NET "T_END" LOC = "P12" ;
NET "TELEGRAM_RUN" LOC = "M13" ;
NET "UNKNOWN_BYTE" LOC = "N14" ;
NET "WORKING" LOC = "P11" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Datei 8-6..\VHDL_Bausteine\TEST_CTRL_TELEGRAM_CHECK\test_ctrl_telegram_check.ucf



8.3.8 Steuerungsbelegung Testumgebung

Buttons	BTN0:	TELEGRAM_RUN
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	BYTE_IN(0)
	SW1:	BYTE_IN(1)
	SW2:	BYTE_IN(2)
	SW3:	BYTE_IN(3)
	SW4:	BYTE_IN(4)
	SW5:	BYTE_IN(5)
	SW6:	BYTE_IN(6)
	SW7:	BYTE_IN(7)
LEDs	LD0:	NEXT_STATE
	LD1:	NO_ED
	LD2:	PARITY_FAIL
	LD3:	UNKNOWN_BYT
	LD4:	KNOWN_T
	LD5:	SEND_OUT
	LD6:	READY
	LD7:	WORKING

Tabelle 8-12: Steuerungsbelegung Testumgebung TELEGRAM_CHECK

8.3.9 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand des Funktionsmoduls TELEGRAM_CHECK verkürzt aus. Siehe auch **Tabelle 8-11: Liste der Prozesse TELEGRAM_CHECK**, Seite 8-42 und **Tabelle 8-9: Variablendefinition TELEGRAM_CHECK**, Seite 8-40



8.4 Modul RS232_TX

Das Modul RS232_TX sendet das Byte vom Eingang SEND_BYTE als Bits weiter wenn eine logische 1 am Eingang SEND anliegt. Die Daten werden über den Ausgang TX mittels RS-232 (19200 Baud, 1 Startbit, 8 Datenbit, 1 Stopabit, keine Parität) gesendet. Sobald ein Datenwort von 8 Bit gesendet wurde signalisiert das Modul am Ausgang READY seine erneute Sendebereitschaft.

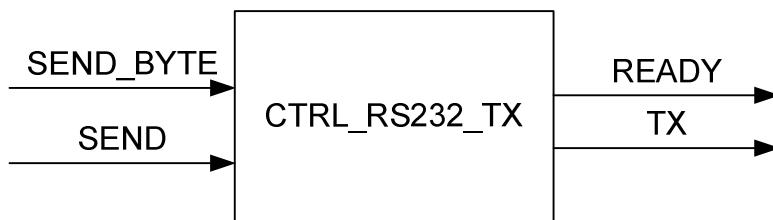
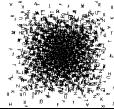


Abbildung 8-13: Wirkungsplan RS232_TX

Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
SEND_BYTE	VECTOR, 8 Bit	Eingang	Eingang Datenbyte
SEND	BOOL	Eingang	1: Senden
TX	BOOL	Ausgang	1: Bit ist 1, 0: Bit ist 0
READY	BOOL	Ausgang	1: bereit zum Senden
COUNT	VECTOR,16 Bit	intern	Zählerwert
CNT01	CONSTANT	intern	Wert: 0A2C
CNT02	CONSTANT	intern	Wert: 1458
CNT03	CONSTANT	intern	Wert: 1E84
CNT04	CONSTANT	intern	Wert: 28B0
CNT05	CONSTANT	intern	Wert: 32DC
CNT06	CONSTANT	intern	Wert: 3D09
CNT07	CONSTANT	intern	Wert: 4735
CNT08	CONSTANT	intern	Wert: 5161
CNT09	CONSTANT	intern	Wert: 5B8D
CNT10	CONSTANT	intern	Wert: 65B9
ST_TX	ENUM	intern	Zustandsvariable (ST_TX_00, ST_TX_01, ST_TX_02, ST_TX_03, ST_TX_04, ST_TX_05, ST_TX_06, ST_TX_07, ST_TX_08, ST_TX_09, ST_TX_10, ST_TX_11)

Tabelle 8-13: Variablendefinition RS232_TX



8.4.1 Impulsdiagramm

Es wird ein binäres Ausgangssignal mit UART-Codierung verwendet um die Datenübertragung via RS-232 zu ermöglichen. Im Ruhezustand (idle) wird das Signal logisch 1 ausgegeben. Es gibt 8 Datenbits (Bit 0 bis Bit 7), ein Startbit (Start) und ein Stopppbit (Stopp). Die Zählerzeitpunkte (t_1 bis t_{10}) geben den Zeitpunkt für eine Umschaltung auf einen neuen Wert an. Die dargestellten logischen Werte 0 und 1 für Bit 1 bis Bit 8 dienen nur als Beispiel.

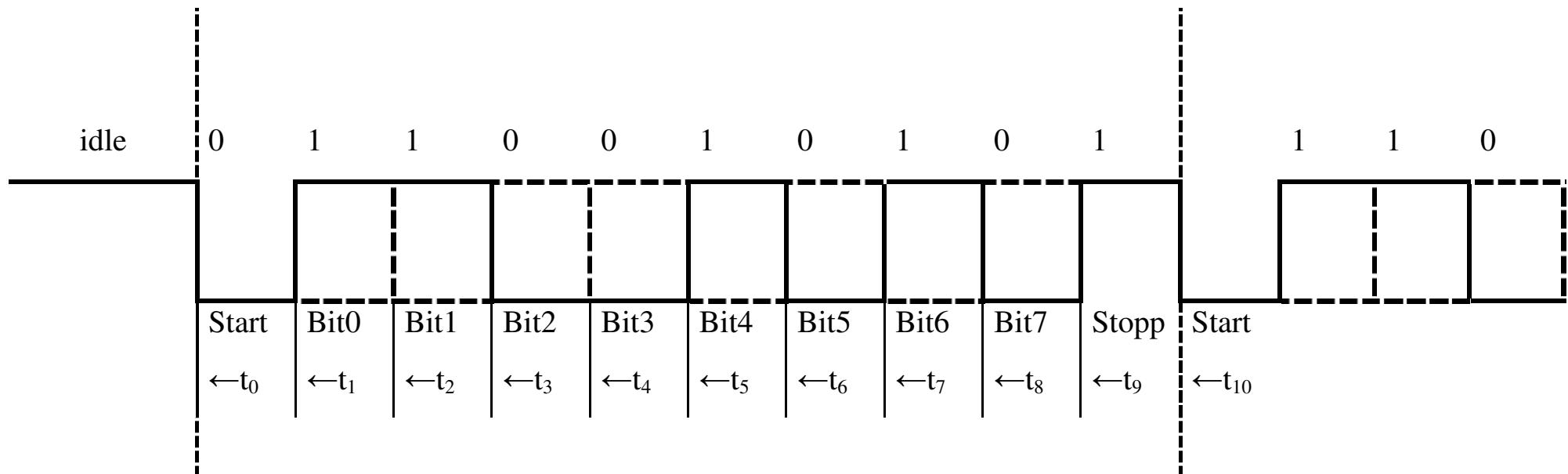
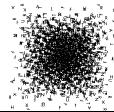


Abbildung 8-14: Impulsdiagramm RS232_TX



8.4.2 Zählerzeitpunkt und Berechnung

Für die Umwandlung des Byte in ein UART-Signals muss ein Bit zu einem bestimmten Zeitpunkt umgeschaltet werden. Dazu wird der Systemtakt des Spartan 3 Development Kits benutzt. Der Systemtakt ist 50 MHz (50.000.000 Hz).

Jedes Bit ist dauert eine bestimmte Anzahl von Takten des Systemtakts an. Die Berechnung erfolgt mit der Formel: (n Bit * Systemtakt) / Baudrate
Für t_1 ist n = 1 für t_2 ist n = 2 usw. Die Baudrate beträgt 19200 Baud. Das Ergebnis wird abgerundet.

Der Zählerstand lang sind die Werte für RS-232. Der Zählerstand kurz sind die Werte für den Schrittbetrieb.

Zeitdauer	Berechnung	Zählerstand	Zählerstand	Zählerstand	Zählerstand	Konstantenname
		lang, dezimal	lang, hexadezimal	kurz, dezimal	kurz, hexadezimal	
t_0		0	0	0	0	÷
t_1	$(1 * \text{Systemtakt}) / 19200$	2604	0A2C	3	3	CNT01
t_2	$(2 * \text{Systemtakt}) / 19200$	5208	1458	6	6	CNT02
t_3	$(3 * \text{Systemtakt}) / 19200$	7812	1E84	9	9	CNT03
t_4	$(4 * \text{Systemtakt}) / 19200$	10416	28B0	12	C	CNT04
t_5	$(5 * \text{Systemtakt}) / 19200$	13020	32DC	15	F	CNT05
t_6	$(6 * \text{Systemtakt}) / 19200$	15625	3D09	18	12	CNT06
t_7	$(7 * \text{Systemtakt}) / 19200$	18229	4735	21	15	CNT07
t_8	$(8 * \text{Systemtakt}) / 19200$	20833	5161	24	18	CNT08
t_9	$(9 * \text{Systemtakt}) / 19200$	23437	5B8D	27	1B	CNT09
t_{10}	$(10 * \text{Systemtakt}) / 19200$	26041	65B9	30	1E	CNT10
÷ = keine Belegung						

Tabelle 8-14: Zählerzeitpunkte RS232_TX



8.4.3 Verhaltensbeschreibung

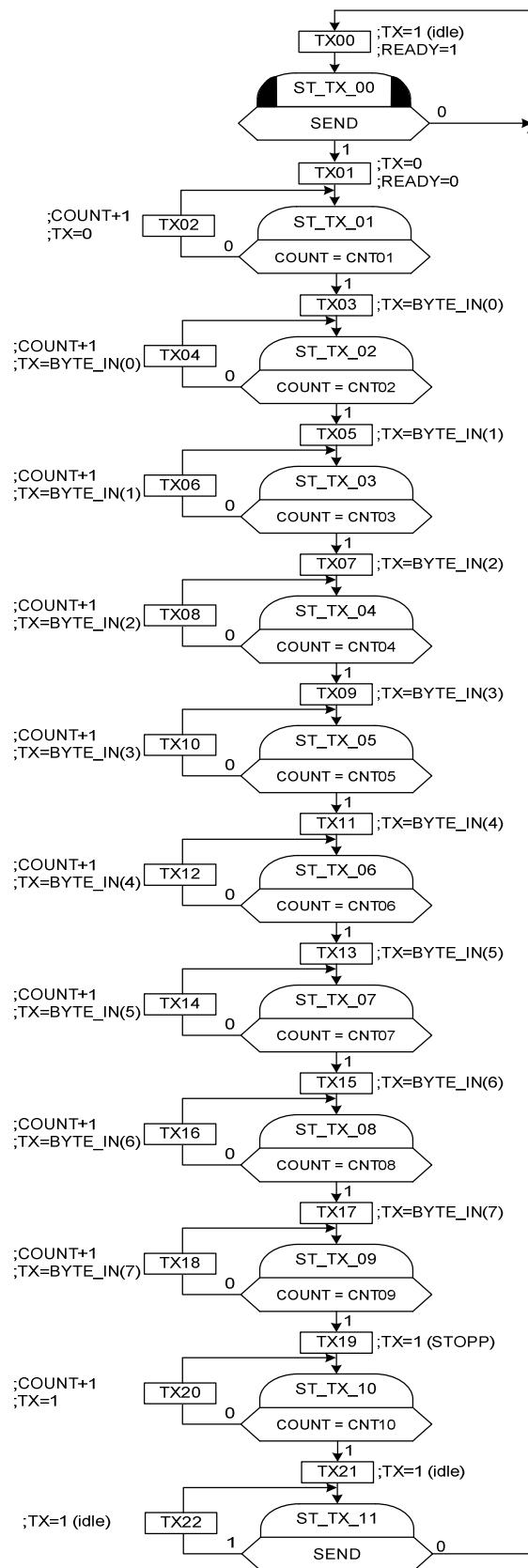


Abbildung 8-15: Programmablaufgraph RS232_TX



8.4.4 Belegung Ausgangsvariablen

Bezeichner	n_COUNT	READY	TX
TX00	:= 0	1	1
TX01	:=0	0	0
TX02	:=COUNT+1	0	0
TX03	:=COUNT+1	0	SEND_BYTE(0)
TX04	:=COUNT+1	0	SEND_BYTE(0)
TX05	:=COUNT+1	0	SEND_BYTE(1)
TX06	:=COUNT+1	0	SEND_BYTE(1)
TX07	:=COUNT+1	0	SEND_BYTE(2)
TX08	:=COUNT+1	0	SEND_BYTE(2)
TX09	:=COUNT+1	0	SEND_BYTE(3)
TX10	:=COUNT+1	0	SEND_BYTE(3)
TX11	:=COUNT+1	0	SEND_BYTE(4)
TX12	:=COUNT+1	0	SEND_BYTE(4)
TX13	:=COUNT+1	0	SEND_BYTE(5)
TX14	:=COUNT+1	0	SEND_BYTE(5)
TX15	:=COUNT+1	0	SEND_BYTE(6)
TX16	:=COUNT+1	0	SEND_BYTE(6)
TX17	:=COUNT+1	0	SEND_BYTE(7)
TX18	:=COUNT+1	0	SEND_BYTE(7)
TX19	:=COUNT+1	0	1
TX20	:=COUNT+1	0	1
TX21	:=0	0	1
TX22	:=0	0	1

Tabelle 8-15: Belegung der Ausgangsvariablen RS232_TX

8.4.5 Auswahl Automatentyp

Siehe Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister, Seite 8-8.

Der einziger Unterschied ist, dass kein Eingangsregister verwendet wurde. IREG und n_X entfallen. Die Eingangsvariable X geht direkt in die Eingangs- /Ausgangslogik IL/OL.

8.4.6 Liste der Prozesse

Prozessname	Funktion
SREG_M_PROC	Zustandsregister, Master
SREG_S_PROC	Zustandsregister, Slave
CTRL_RS232_RX_PROC	Eingangs- / Ausgangslogik

Tabelle 8-16: Liste der Prozesse RS232_RX



8.4.7 VHDL-Programm

```
-- CTRL_RS232_TX
-- Input wird bitweise via RS232 versendet
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 10.01.2013
-- Bearbeiter: mharndt
-- Geaendert: 24.01.2013
-- Umstellung auf: rising_edge(CLK) und falling_edge(CLK) und http://www.sigasi.com/content/clock-
edge-detection

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_RS232_TX_VHDL is
    Port(SEND_BYTE : in std_logic_vector (7 downto 0); --Eingangsvariable, zu Daten Input, 8 bit
          SEND : in std_logic; --Eingangsvariable, Byte OK
          TX : out std_logic; --Ausgangsvariable, Transmit Bit
          READY: out std_logic; --Ausgangsvariable, bereit zum Senden

          CLK : in std_logic; --Taktvariable
          IN_NEXT_STATE: in std_logic; --1: Zustandsuebergang möglich
          RESET : in std_logic); --1: Initialzustand annehmen

end CTRL_RS232_TX_VHDL;

architecture Behavioral of CTRL_RS232_TX_VHDL is

type TYPE_STATE is
    (ST_TX_00, --Zustaende CTRL_RS232_TX
     ST_TX_01,
     ST_TX_02,
     ST_TX_03,
     ST_TX_04,
     ST_TX_05,
     ST_TX_06,
     ST_TX_07,
     ST_TX_08,
     ST_TX_09,
     ST_TX_10,
     ST_TX_11);

signal SV : TYPE_STATE; --Zustandsvariable
signal n_SV: TYPE_STATE; --Zustandsvariable, neuer Wert
signal SV_M: TYPE_STATE; --Zustandsvariable, Ausgang Master

--signal not_CLK : std_logic; --negierte Taktvariable

signal COUNT : std_logic_vector (15 downto 0); --Zaehler, Vektor, 16 Bit
signal n_COUNT : std_logic_vector (15 downto 0); --Zaehler, neuer Wert, Vektor, 16 Bit
signal COUNT_M : std_logic_vector (15 downto 0); --Zaehler, Ausgang Master, Vektor, 16 Bit

--Konstanten, lang 9600 Baud, 1 Startbit, 8 Datenbit, 1 Stoppbit, keine Parität
constant CNT01 : std_logic_vector := x"1458"; --16 Bit
constant CNT02 : std_logic_vector := x"2C98"; --usw.
constant CNT03 : std_logic_vector := x"3D08";
constant CNT04 : std_logic_vector := x"5160";
constant CNT05 : std_logic_vector := x"65B8";
constant CNT06 : std_logic_vector := x"7A10";
constant CNT07 : std_logic_vector := x"8E68";
constant CNT08 : std_logic_vector := x"A2C0";
constant CNT09 : std_logic_vector := x"B718";
constant CNT10 : std_logic_vector := x"CB70";

--Konstanten, lang 19200 Baud, 1 Startbit, 8 Datenbit, 1 Stoppbit, keine Parität
constant CNT01 : std_logic_vector := x"0A2C"; --16 Bit
constant CNT02 : std_logic_vector := x"1458"; --usw.
constant CNT03 : std_logic_vector := x"1E84";
constant CNT04 : std_logic_vector := x"28B0";
constant CNT05 : std_logic_vector := x"32DC";
constant CNT06 : std_logic_vector := x"3D09";
constant CNT07 : std_logic_vector := x"4735";
constant CNT08 : std_logic_vector := x"5161";
constant CNT09 : std_logic_vector := x"5B8D";
```



```
constant CNT10 : std_logic_vector := x"65B9";  
  
begin  
  
--NOT_CLK_PROC: process (CLK) --negieren Taktvariable  
begin  
-- not_CLK <= not CLK;  
end process;  
  
SREG_M_PROC: process (RESET, n_SV, CLK) --Master  
begin  
if (RESET = '1')  
then SV_M      <= ST_TX_00;  
else  
if rising_edge(CLK)  
then  
if (IN_NEXT_STATE = '1')  
then SV_M      <= n_SV;  
COUNT_M <= n_COUNT;  
else SV_M      <= SV_M;  
COUNT_M <= COUNT_M;  
end if;  
end if;  
end if;  
end process;  
  
SREG_S_PROC: process (RESET, SV_M, CLK) --Slave  
begin  
if (RESET = '1')  
then SV      <= ST_TX_00;  
else  
if falling_edge(CLK)  
then SV      <= SV_M;  
COUNT <= COUNT_M;  
end if;  
end if;  
end process;  
  
CTRL_RS232_RX_PROC:process (SV, COUNT, SEND, SEND_BYTE) --Daten über RS232 senden  
begin  
case SV is  
when ST_TX_00 =>  
if (SEND = '1')  
then  
--TX01  
n_COUNT <= x"0000"; -- kleiner Zaehler Neustart  
TX <= '0'; --Startbit  
READY <= '0';  
n_SV <= ST_TX_01; --Zustandsübergang  
else  
--TX00  
n_COUNT <= x"0000"; -- kleiner Zaehler Neustart  
TX <= '1'; --Idle  
READY <= '1'; --Bereit zum Senden  
n_SV <= ST_TX_00; --bleibt im gleichen Zustand  
end if;  
  
when ST_TX_01 =>  
if (COUNT = CNT01) --Zaehler = 5208  
then  
--TX03  
n_COUNT <= COUNT+1; -- Zaehler erhöhen  
TX <= SEND_BYTE(0); --Bit 0  
READY <= '0';  
n_SV <= ST_TX_02; --Zustandsübergang  
else  
--TX02  
n_COUNT <= COUNT+1; -- Zaehler erhöhen  
TX <= '0'; --Startbit  
READY <= '0';  
n_SV <= ST_TX_01; --bleibt im gleichen Zustand  
end if;  
  
when ST_TX_02 =>  
if (COUNT = CNT02) --Zaehler = 11416  
then  
--TX05  
n_COUNT <= COUNT+1; -- Zaehler erhöhen
```



```
TX <= SEND_BYTE(1); --Bit 1
READY <= '0';
n_SV <= ST_TX_03; --Zustandsübergang
else
--TX04
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(0); --Bit 0
READY <= '0';
n_SV <= ST_TX_02; --bleibt im gleichen Zustand
end if;

when ST_TX_03 =>
if (COUNT = CNT03) --Zahler = 15624
then
--TX07
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(2); --Bit 2
READY <= '0';
n_SV <= ST_TX_04; --Zustandsübergang
else
--TX06
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(1); --Bit 1
READY <= '0';
n_SV <= ST_TX_03; --bleibt im gleichen Zustand
end if;

when ST_TX_04 =>
if (COUNT = CNT04) --Zahler = 20832
then
--TX09
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(3); --Bit 3
READY <= '0';
n_SV <= ST_TX_05; --Zustandsübergang
else
--TX08
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(2); --Bit 2
READY <= '0';
n_SV <= ST_TX_04; --bleibt im gleichen Zustand
end if;

when ST_TX_05 =>
if (COUNT = CNT05) --Zahler = 26040
then
--TX11
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(4); --Bit 4
READY <= '0';
n_SV <= ST_TX_06; --Zustandsübergang
else
--TX10
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(3); --Bit 3
READY <= '0';
n_SV <= ST_TX_05; --bleibt im gleichen Zustand
end if;

when ST_TX_06 =>
if (COUNT = CNT06) --Zahler = 31248
then
--TX13
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(5); --Bit 5
READY <= '0';
n_SV <= ST_TX_07; --Zustandsübergang
else
--TX12
n_COUNT <= COUNT+1; -- Zahler erhöhen
TX <= SEND_BYTE(4); --Bit 4
READY <= '0';
n_SV <= ST_TX_06; --bleibt im gleichen Zustand
end if;

when ST_TX_07 =>
if (COUNT = CNT07) --Zahler = 36456
then
```



```
--TX15
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE(6); --Bit 6
READY <= '0';
n_SV <= ST_TX_08; --Zustandsübergang
else
--TX14
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE(5); --Bit 5
READY <= '0';
n_SV <= ST_TX_07; --bleibt im gleichen Zustand
end if;

when ST_TX_08 =>
if (COUNT = CNT08) --Zaehler = 41664
then
--TX17
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE(7); --Bit 7
READY <= '0';
n_SV <= ST_TX_09; --Zustandsübergang
else
--TX16
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE(6); --Bit 6
READY <= '0';
n_SV <= ST_TX_08; --bleibt im gleichen Zustand
end if;

when ST_TX_09 =>
if (COUNT = CNT09) --Zaehler = 46872
then
--TX19
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= '1'; --Stoppbit
READY <= '0';
n_SV <= ST_TX_10; --Zustandsübergang
else
--TX18
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE(7); --Bit 7
READY <= '0';
n_SV <= ST_TX_09; --bleibt im gleichen Zustand
end if;

when ST_TX_10 =>
if (COUNT = CNT10) --Zaehler = 52080
then
--TX21
n_COUNT <= x"0000"; -- Zaehler neustart
TX <= '1'; --Idle
READY <= '0';
n_SV <= ST_TX_11; --Zustandsübergang
else
--TX20
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= '1'; --Stoppbit
READY <= '0';
n_SV <= ST_TX_10; --bleibt im gleichen Zustand
end if;

when ST_TX_11 =>
if (SEND = '0') -- Wenn SEND=0 dann warten auf SEND sonst Idle senden
then
--TX00
n_COUNT <= x"0000"; -- Zaehler neustart
TX <= '1'; --Idle
READY <= '1'; --Bereit zum Senden
n_SV <= ST_TX_00; --Zustandsübergang
else
--TX22
n_COUNT <= x"0000"; -- Zaehler neustart
TX <= '1'; --Idle
READY <= '0';
n_SV <= ST_TX_11; --bleibt im gleichen Zustand
end if;

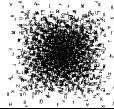
when others =>
```



```
-- TX00
n_COUNT <= x"0000"; -- kleiner Zaehler Neustart
TX <= '1'; --Idle
READY <= '0';
n_SV <= ST_TX_00; --Zustandsübergang
end case;
end process;

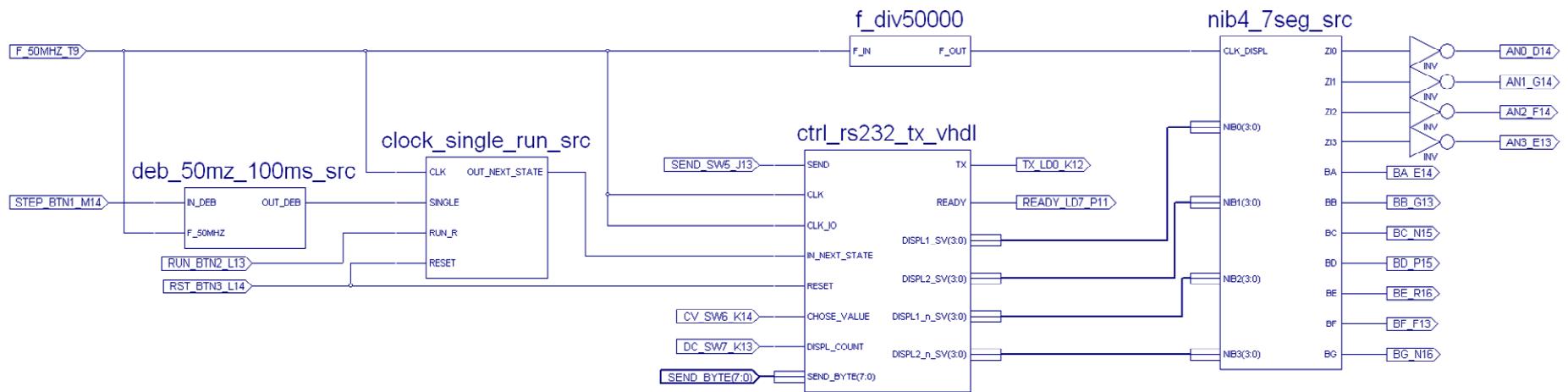
end Behavioral;
```

Datei 8-7: ..\VHDL_Bausteine\CTRL_RS232_TX\CTRL_RS232_TX_VHDL.vhd



8.4.8 Testumgebung

Die Tests können im RUN-Betrieb direkt mit einem via RS-232 angebundenen Rechner durchgeführt und in einem Terminal angezeigt werden. Anpassungen des Funktionsmoduls RS232_TX, wie Anzeige des Zustandes, sind im Anhang 11.2 VHDL-Programm RS232_TX aus Testumgebung, Seite 11-9.



CTRL_RS232_TX_VHDL
DEB_50MHZ_100MS_SRC
CLOCK_SINGLE_RUN_SRC
F_DIV50000
NIB4_7SEG_SRC
INV

Siehe 8.4 Modul RS232_TX, Seite, 8-57
Siehe /DEB_50MHZ_100MS/, Seite 12-18
Siehe /CLOCK_SINGLE_RUN/, Seite 12-18
Siehe /F_DIV50000/, Seite 12-18
Siehe /NIB_7SEG_SRC/, Seite 12-18
Siehe /INV/, Seite 12-18

Abbildung 8-16: Testumgebung RS232_TX



8.4.9 UCF-Datei

```
#PACE: Start of Constraints generated by PACE
```

```
#PACE: Start of PACE I/O Pin Assignments
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "READY_LD7_P11" LOC = "P11" ;
NET "RST_BTN3_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "SEND_BYTE<0>" LOC = "F12" ;
NET "SEND_BYTE<1>" LOC = "G12" ;
NET "SEND_BYTE<2>" LOC = "H14" ;
NET "SEND_BYTE<3>" LOC = "H13" ;
NET "SEND_BYTE<4>" LOC = "J14" ;
NET "SEND_BYTE<5>" LOC = "J13" ;
NET "SEND_BYTE<6>" LOC = "K14" ;
NET "SEND_BYTE<7>" LOC = "K13" ;
NET "SEND_SW5_J13" LOC = "M13" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
NET "TX_LDO_K12" LOC = "R13" ;
```

```
#PACE: Start of PACE Area Constraints
```

```
#PACE: Start of PACE Prohibit Constraints
```

```
#PACE: End of Constraints generated by PACE
```

Datei 8-8: ..\VHDL_Bausteine\TEST_CTRL_RS232_TX\test_ctrl_rs232_tx_schematic.ucf



8.4.10 Steuerungsbelegung Testumgebung

Buttons	BTN0:	SEND
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	SEND_BYT(0)
	SW1:	SEND_BYT(1)
	SW2:	SEND_BYT(2)
	SW3:	SEND_BYT(3)
	SW4:	SEND_BYT(4)
	SW5:	SEND_BYT(5)
	SW6:	SEND_BYT(6)
	SW7:	SEND_BYT(7)
LEDs	LD0:	nicht benutzt
	LD1:	nicht benutzt
	LD2:	nicht benutzt
	LD3:	nicht benutzt
	LD4:	nicht benutzt
	LD5:	nicht benutzt
	LD6:	nicht benutzt
	LD7:	READY

Tabelle 8-17: Steuerungsbelegung Testumgebung RS232_TX

8.4.11 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand des Funktionsmoduls RS232_TX verkürzt aus. Siehe auch **Tabelle 8-16: Liste der Prozesse RS232_TX**, Seite 8-61 und **Tabelle 8-13: Variablendefinition RS232_TX**, Seite 8-57.



9 PROFIBUS_MONITOR

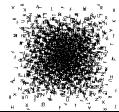
Der PROFIBUS_MONITOR ist die Zusammenschaltung der bisher beschriebenen Funktionsmodule und zusätzlicher, bereitgestellter Funktionsmodule die nicht Bestandteil dieser Dokumentation sind.

Funktionsmodule:	
InAB_INPUT	Siehe 8.1 Modul InAB_INPUT, Seite 8-1
BIT_REGISTER	Siehe 8.2 Modul BIT_REGISTER, Seite 8-25
TELEGRAM_CHECK	Siehe 8.3 Modul TELEGRAM_CHECK, Seite 8-39
RS232_TX	Siehe 8.4 Modul RS232_TX, Seite 8-57

Tabelle 9-1: Beschriebene Funktionsmodule PROFIBUS_MONITOR

Zusätzliche Funktionsmodule:	
DEB_50MHZ_100MS_SRC	Siehe /DEB_50MHZ_100MS/, Seite 12-18
CLOCK_SINGLE_RUN_SRC	Siehe /CLOCK_SINGLE_RUN/, Seite 12-18
F_DIV50000	Siehe /F_DIV50000/, Seite 12-18
NIB4_7SEG_SRC	Siehe /NIB_7SEG_SRC/, Seite 12-18
INV	Siehe /INV/, Seite 12-18

Tabelle 9-2: Zusätzliche Funktionsmodule PROFIBUS_MONITOR



9.1 Testumgebung

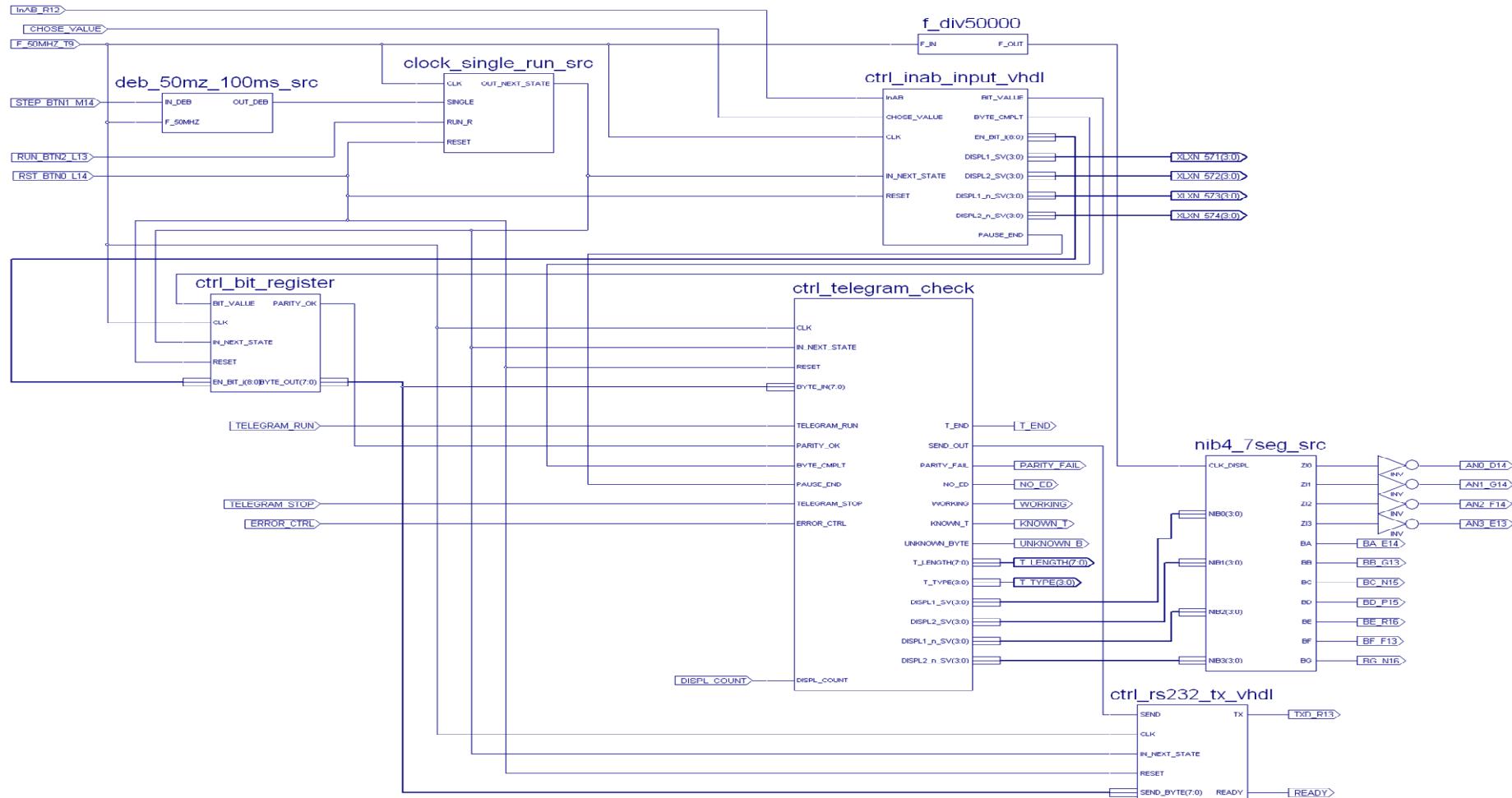


Abbildung 9-1: Testumgebung PROFIBUS_MONITOR



9.2 UCF-Datei

#PACE: Start of Constraints generated by PACE

```
#PACE: Start of PACE I/O Pin Assignments
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "CHOOSE_VALUE" LOC = "J14" ;
NET "DISPL_COUNT" LOC = "J13" ;
NET "ERROR_CTRL" LOC = "K14" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "InAB_R12" LOC = "R12" ;
NET "KNOWN_T" LOC = "L12" ;
NET "NO_ED" LOC = "P13" ;
NET "PARITY_FAIL" LOC = "N12" ;
NET "READY" LOC = "P12" ;
NET "RST_BTN0_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
NET "T_END" LOC = "P11" ;
NET "TELEGRAM_RUN" LOC = "M13" ;
NET "TELEGRAM_STOP" LOC = "K13" ;
NET "TXD_R13" LOC = "R13" ;
NET "UNKNOWN_B" LOC = "P14" ;
NET "WORKING" LOC = "N14" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Datei 9-1..\VHDL_Bausteine\PROFIBUS_MONITOR\profibus_monitor_sch.ucf



9.3 Steuerungsbelegung Testumgebung

Buttons	BTN0:	TELEGRAM_RUN
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	nicht benutzt
	SW1:	nicht benutzt
	SW2:	nicht benutzt
	SW3:	nicht benutzt
	SW4:	CHOSE_VALUE
	SW5:	DISPL_COUNT
	SW6:	ERROR_CTRL
	SW7:	TELEGRAM_STOP
LEDs	LD0:	nicht benutzt
	LD1:	UNKNOWN_BYT
	LD2:	KNOWN_T(elegram)
	LD3:	WORKING
	LD4:	NO_ED
	LD5:	PARTY_FAIL
	LD6:	READY
	LD7:	T_END

Tabelle 9-3: Steuerungsbelegung Testumgebung PROFIBUS_MONITOR

9.4 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand des Funktionsmoduls InAB_INPUT verkürzt aus. Siehe auch **Tabelle 8-4: Liste der Prozesse InAB_INPUT**, Seite 8-8 und **Tabelle 8-1: Variablendefinition InAB_INPUT**, Seite 8-3.



10 Ausblick zukünftige Entwicklung

Der Profibusmonitor kann bisher nur die vom Profibus empfangenen Telegramme via RS-232 an einen Rechner zur Ausgabe weiterleiten.

Eine Erweiterung könnte daher ein Speicher zur Speicherung der empfangenen Telegramme sein. Diese Telegramme könnte man wiederum auf einem Rechner über ein Programm via RS-232 auslesen und anzeigen. Die Daten der Telegramme könnten so in eine sehr viel übersichtlichere und besser verständlichen Form dargestellt werden.

Auch eine Wiedergabe der Telegramme im Speicher auf einer Anzeige oder einem Monitor könnte direkt mit dem Spartan 3 Development Kit realisiert werden. Für die Anzeige auf einem Monitor müsste der Eingang, welcher auf dem VGA-Anschluss liegt, auf einem der anderen Eingänge des Spartan 3 Development Kits umgebaut werden. Eine direkte Anzeige auf den vier 7-Segment Anzeigen des Spartan 3 Development Kit ist ebenfalls denkbar, genauso wie der Anschluss einer zusätzlichen externen Anzeige die mehr als vier Zeichen darstellen kann.

Die Steuerung könnte dabei wahlweise mit den Schaltern und Tastern des Spartan 3 Development Kit erfolgen, externen Tastern und Schaltern oder über eine angeschlossene Tastatur oder Maus.

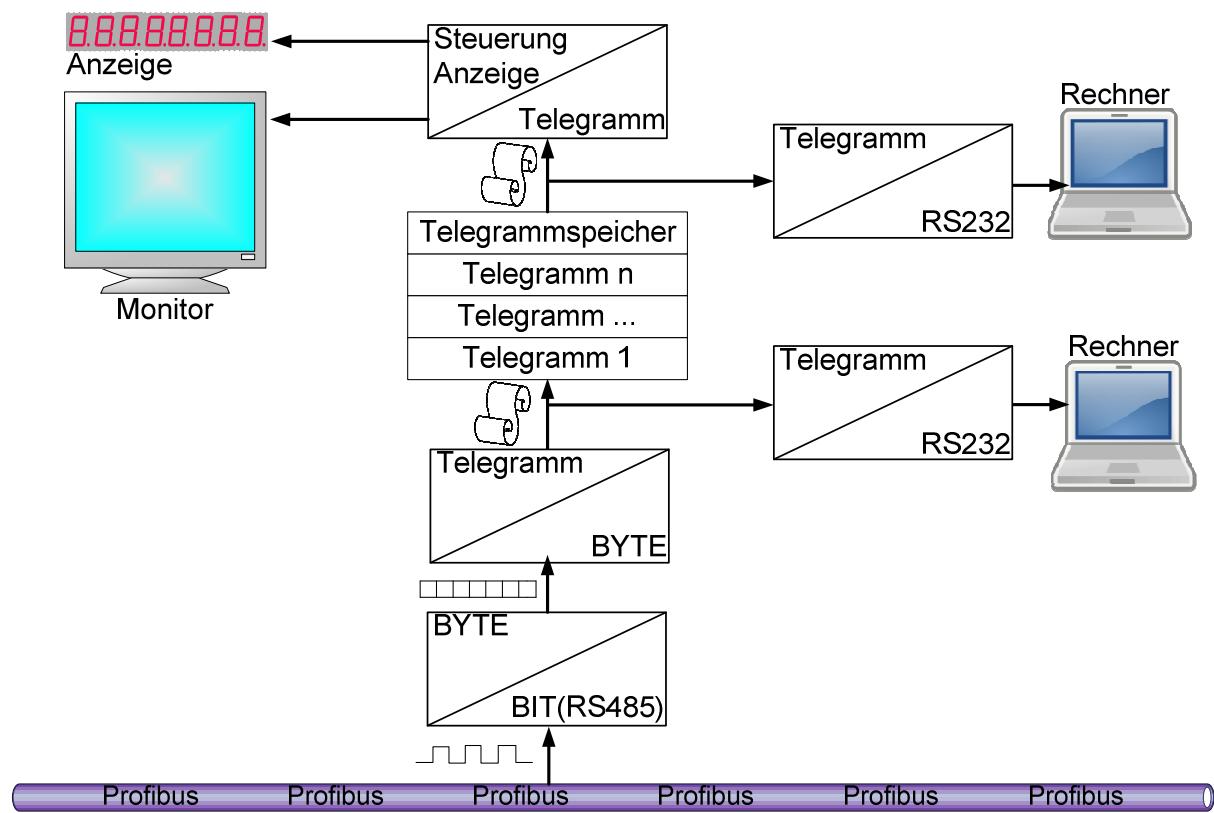


Abbildung 10-1: Konzept Profibusmonitor mit zusätzlichen Erweiterungen
/7SEG/ /LAPTOP/ /SCROLL/



10.1 Testmodul SRAM_25MHZ_255_BYTE

Die Umsetzung des Telegrammspeichers könnte die zwei SRAM-Speicherbausteine des Spartan 3 Developement Kits nutzen. Es wurde daher ein Testfunktionsmodul zum Beschreiben und Lesen des ersten SRAM-Speichermoduls ausgearbeitet. Das Modul beschreibt den Speicherchip 1 des SRAM-Speichermoduls adressweise von Null an aufsteigend. Als Daten werden Werte, beginnen bei $FFFF_h$, geschrieben die sich bei jeder weiteren Adresse um 1 verringern. Bsp.: Auf der ersten Adresse 00000_h werden die Daten $FFFF_h$ geschrieben, auf der letzten Adresse $FFFFF_h$ im Speicherchip werden die Daten 0000_h geschrieben.



Abbildung 10-2: Wirkungsplan TEST_SRAM_25MHZ_255_BYTE

Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
COUNT_DAT_INOUT	VECTOR, 16 Bit	Ein-/Ausgang	Ein-/Auszabe gespeicherter Daten
GO	BOOL	Eingang	1: Start Speicher beschreiben
DISPL_ADR	BOOL	Eingang	0: aktueller Zustand 1: aktuelle Adresse
DISPL_DAT	BOOL	Eingang	0: Folgezustand 1: aktuelle Daten
PLUS	BOOL	Eingang	1: Adresszähler +1
MINUS	BOOL	Eingang	1: Adresszähler -1
COUNT_ADR_OUT	VECTOR, 16 Bit	Ausgang	Ausgabe Adresse
WE	BOOL	Ausgang	Write Enable
OE	BOOL	Ausgang	Output Enable
CE1	BOOL	Ausgang	Chip Enable Chip 1
UB1	BOOL	Ausgang	Upper Byte Enable Chip 1
LB1	BOOL	Ausgang	Lower Byte Enable Chip 1
STOP	BOOL	Ausgang	1: Stop
COUNT_ADR	VECTOR, 18 Bit	intern	Adresszähler
COUNT_DAT	VECTOR, 16 Bit	intern	Datenzähler
COUNT_DAT_INPUT	VECTOR, 16 Bit	intern	Dateninput
WRITE	BOOL	intern	1: schreiben in RAM
ST_RAM	ENUM	intern	Zustandsvariable (ST_RAM_00, ST_RAM_01, ST_RAM_02, ST_RAM_03,



Variablenname	Datentyp	VariablenTyp	Informationen / Anweisungen
			ST_RAM_04, ST_RAM_05, ST_RAM_06, ST_RAM_07, ST_RAM_08, ST_RAM_09)

Tabelle 10-1: Variablendefinition TEST_SRAM_25MHZ_255_BYT

10.2 Lesen und Schreiben

Auf dem Spartan-3 Development Kit werden zwei SRAM Speicherbausteine vom Typ „ISSI IS61LV25616AL-10T“ benutzt. Details siehe Datenblatt /SRAM/ Seite 12-15.

Beim Lesen und Schreiben muss das zeitliche Verhalten (Timing) der Speicherbausteine berücksichtigt werden. Daher wurde hier die Taktung (CLK) von 50MHz auf 25MHz reduziert. Das lässt genug Zeit die Schreib- und Leseoperationen sicher vorzunehmen. Die Nutzung bei 50 MHz ist aber prinzipiell möglich, bedarf aber einer genaueren Betrachtung des Timings.

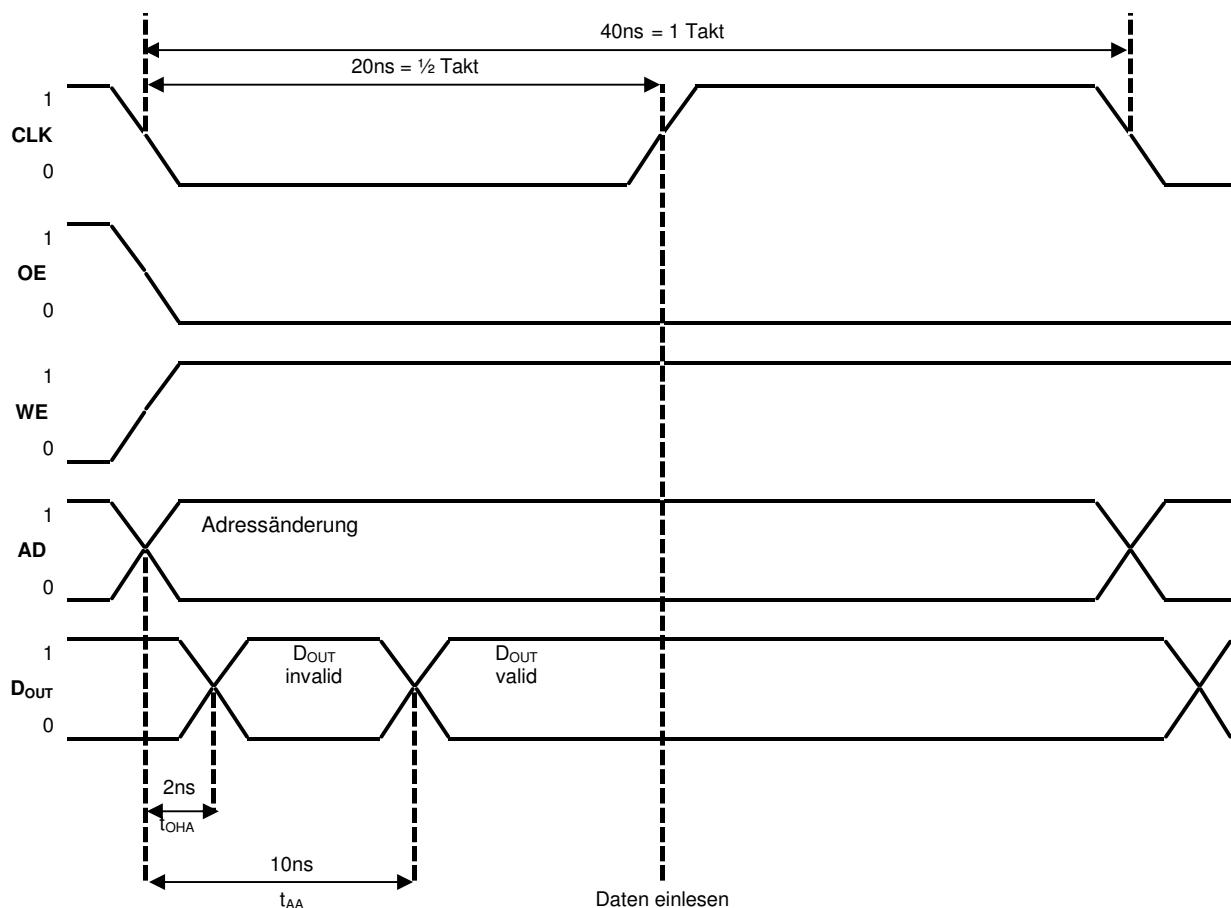


Abbildung 10-3: Impulsdiagramm SRAM lesen

- CLK: Clock
- OE: Output Enable
- WE: Write Enable
- AD: Address
- D_{OUT}: Data Output
- t_{OH}: Output Hold Time
- t_{AA}: Address Access Time

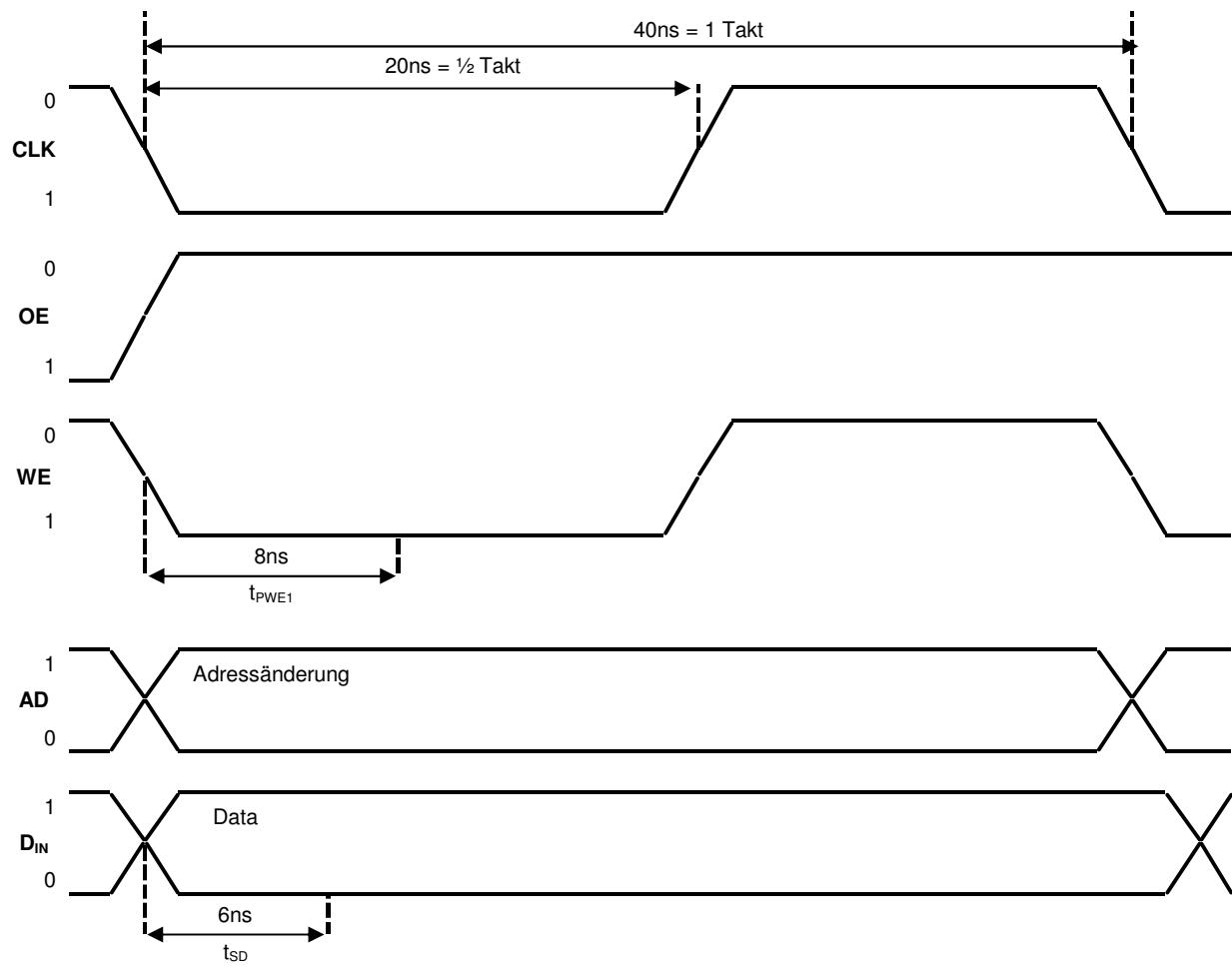


Abbildung 10-4: Impulsdiagramm SRAM schreiben

CLK: Clock
OE: Output Enable
WE: Write Enable
AD: Address
D_{IN}: Data Input
 t_{PWE1} : Write Enable Pulse Width
 t_{SD} : Data Setup to Write End

10.3 Verhaltensbeschreibung

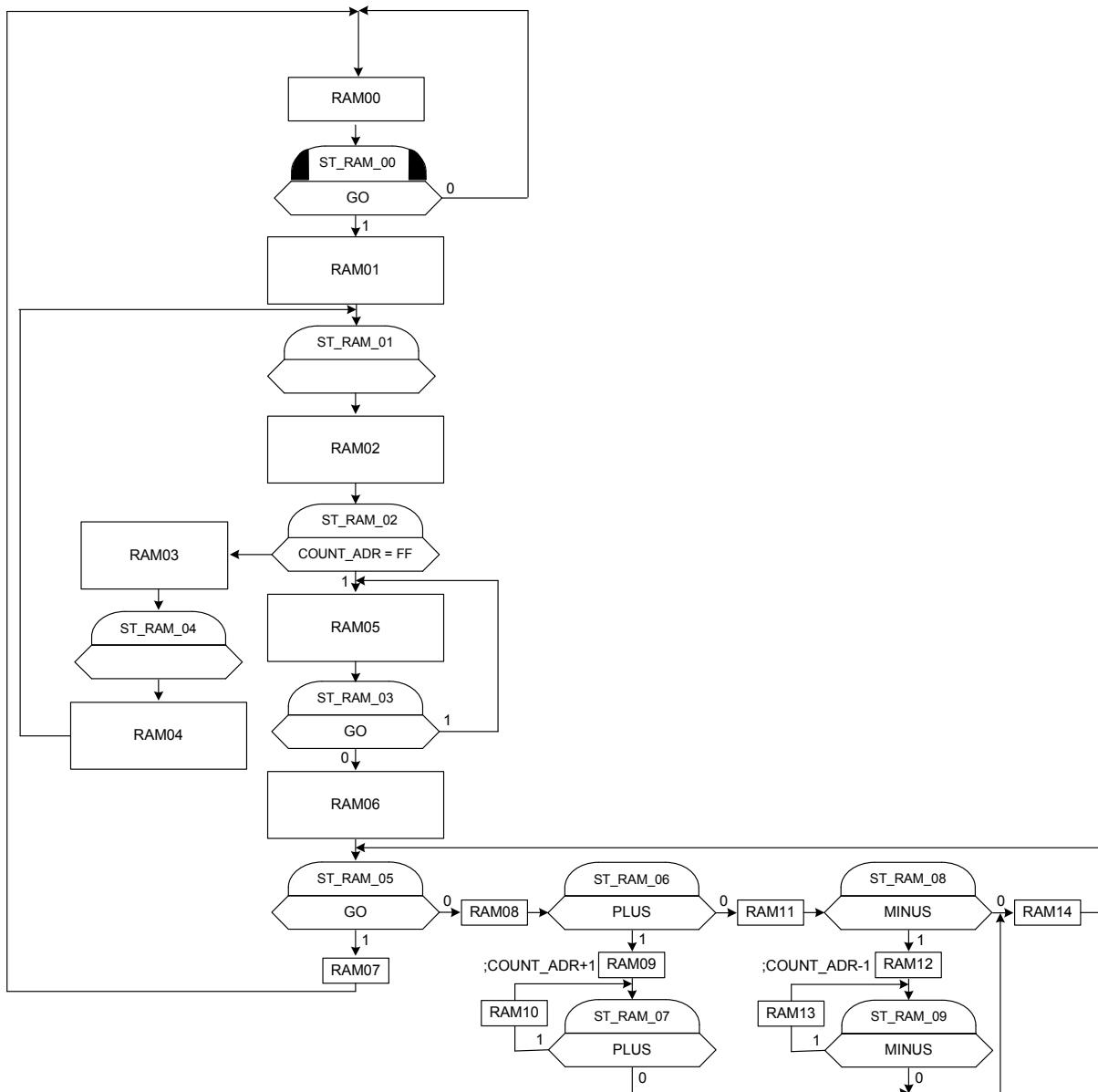
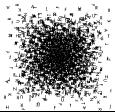


Abbildung 10-5: Wirkungsplan TEST_SRAM_25MHZ_255_BYTE



10.4 Belebung Ausgangsvariablen

Bezeichner	COUNT_ADR	COUNT_DAT	WE (0=aktiv)	OE (0=aktiv)	CE1 (0=aktiv)	STOP (1=aktiv)	n_WRITE
RAM00	00	FF	1	1	1	0	0
RAM01	00	FF	1	1	1	0	0
RAM02	:=COUNT_ADR	:=COUNT_DAT	0	1	0	0	1
RAM03	:=COUNT_ADR+1	:=COUNT_DAT-1	1	1	0	0	0
RAM04	:=COUNT_ADR	:=COUNT_DAT	1	1	0	0	0
RAM05	:=COUNT_ADR	:=COUNT_DAT	1	1	0	0	0
RAM06	00	:=COUNT_DAT	1	1	0	1	0
RAM07	:=COUNT_ADR	:=COUNT_DAT	1	0	0	1	0
RAM08	:=COUNT_ADR	:=COUNT_DAT	1	0	0	1	0
RAM09	:=COUNT_ADR+1	:=COUNT_DAT_INPUT	1	0	0	0	0
RAM10	:=COUNT_ADR	:=COUNT_DAT_INPUT	1	0	0	0	0
RAM11	:=COUNT_ADR	:=COUNT_DAT	1	0	0	1	0
RAM12	:=COUNT_ADR-1	:=COUNT_DAT_INPUT	1	0	0	0	0
RAM13	:=COUNT_ADR	:=COUNT_DAT_INPUT	1	0	0	0	0
RAM14	:=COUNT_ADR	:=COUNT_DAT	1	0	0	1	0

Tabelle 10-2: Belegung der Ausgangsvariablen TEST_SRAM_25MHZ_255_BYTE

10.5 Auswahl Automatentyp

Siehe Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister, Seite 8-8.

10.6 Liste der Prozesse

Prozessname	Funktion
NOT_CLK_PROC	negieren Taktvariable
NOT_CLK_IO_PROC	negieren Taktvariable, Eingangsregister
IREG_PROC	Eingangsregister
SREG_M_PROC	Zustandsregister, Master
SREG_S_PROC	Zustandsregister, Slave
IL_DL_PROC	Eingangs- / Ausgangslogik
STATE_DISPL_PROC	Zustands-/Adressanzeige

Tabelle 10-3: Liste der Prozesse TEST_SRAM_25MHZ_255_BYTE



10.7 VHDL-Programm

```
-- SRAM_25MHZ_255_BYTE
-- beschreibt/liest den SRAM des Spartan 3
-- Ersteller: Martin Harndt
-- Erstellt: 30.11.2012
-- Bearbeiter: mharndt
-- Geaendert: 13.12.2012

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SRAM_25MHZ_255_BYTE is
    Port ( GO : in std_logic;
            COUNT_ADR_OUT : out std_logic_vector(18 downto 0); --Ausgabe Adresse, 19 Byte
            COUNT_DAT_INOUT : inout std_logic_vector(15 downto 0); --Ausgabe gespeicherte
Daten, 16 Byte
            DISPL_ADR : in std_logic; -- umschalten zwischen aktuellen Zustand und Adresse
            DISPL_DAT : in std_logic; -- umschalten zwischen Folgeszustand und Daten
            WE : out std_logic; -- Write Enable
            OE : out std_logic; -- Output Enable
            CE1 : out std_logic; -- Chip Enable
            UB1 : out std_logic; -- Upper Byte Enable
            LB1 : out std_logic; -- Lower Byte Enable
            STOP : out std_logic; -- zum Anzeigen von STOP

            PLUS : in std_logic; -- Adresszähler +1
            MINUS : in std_logic; -- Adresszähler -1
            CLK : in std_logic; --Taktvariable
            CLK_IO : in std_logic; --Taktvariable,
                                --Ein- und Ausgangsregister
            IN_NEXT_STATE : in std_logic; -->1:Zustandsuebergang möglich
            RESET : in std_logic; -->1: Initialzustand annehmen

            DISPL1_SV      : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl1,
binärzahl
            DISPL2_SV      : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl2,
binärzahl
            DISPL1_n_SV   : out std_logic_vector (3 downto 0); --Folgezustand Zahl1, binärzahl
            DISPL2_n_SV   : out std_logic_vector (3 downto 0)); --Folgezustand Zahl2,
binärzahl
end SRAM_25MHZ_255_BYTE;

architecture Behavioral of SRAM_25MHZ_255_BYTE is

type TYPE_STATE is
    (ST_RAM_00, --Zustaende
     ST_RAM_01,
     ST_RAM_02,
     ST_RAM_03,
     ST_RAM_04,
     ST_RAM_05,
     ST_RAM_06,
     ST_RAM_07,
     ST_RAM_08,
     ST_RAM_09);

signal SV : TYPE_STATE; --Zustandsvariable
signal n_SV: TYPE_STATE; --Zustandsvariable, neuer Wert
signal SV_M: TYPE_STATE; --Zustandsvariable, Ausgang Master

signal not_CLK : std_logic; --negierte Taktvariable
signal not_CLK_IO: std_logic; --negierte Taktvariable
                                --Ein- und Ausgangsregister
signal GO_S : std_logic; --Eingangsvariable, --Zwischengespeichert im Eingangsregister
signal PLUS_S : std_logic; --Eingangsvariable, Zwischengespeichert im Eingangsregister
signal MINUS_S : std_logic; --Eingangsvariable, --Zwischengespeichert im Eingangsregister

signal COUNT_ADR : std_logic_vector(18 downto 0); --Adresszaehler, Vektor, 19 bit
```



```
signal n_COUNT_ADR : std_logic_vector(18 downto 0); --Adresszaehler, neuer Wert, Vektor, 19 bit
signal COUNT_ADR_M : std_logic_vector(18 downto 0); --Adresszaehler, Ausgang Master, Vektor, 19 bit

signal COUNT_DAT : std_logic_vector(15 downto 0); --Datenzaehler, Vektor, 15 bit
signal n_COUNT_DAT : std_logic_vector(15 downto 0); --Datenzaehler, neuer Wert, Vektor, 15 bit
signal COUNT_DAT_M : std_logic_vector(15 downto 0); --Datenzaehler, Ausgang Master, Vektor, 15 bit

signal DISPLAY_STATE_SV : std_logic_vector (7 downto 0); -- aktueller Zustand in 8 Bit, binär
signal DISPLAY_STATE_n_SV : std_logic_vector (7 downto 0); -- Folgezustand in 8 Bit, binär

signal COUNT_DAT_INPUT : std_logic_vector(15 downto 0); -- Dateninput
signal WRITE_M : std_logic; --Schreibanzeiger, Ausgang Master, (1=schreiben)
signal n_WRITE : std_logic; --Schreibanzeiger, neuer Wert

begin

NOT_CLK_PROC: process (CLK) --negieren Taktvariable
begin
    not_CLK <= not CLK;
end process;

NOT_CLK_IO_PROC: process (CLK_IO) --negieren Taktvariable, Ein- und Ausgangsregister
begin
    not_CLK_IO <= not CLK_IO;
end process;

IREG_PROC: process (GO, GO_S, not_CLK_IO, PLUS, PLUS_S, MINUS, MINUS_S) --Eingangsregister
begin
    if (not_CLK_IO'event and not_CLK_IO = '1') --Eingangsregister
        then GO_S <= GO;
            PLUS_S <= PLUS;
            MINUS_S <= MINUS;
    end if;
end process;

SREG_M_PROC: process (RESET, n_SV, CLK) --Master
begin
    if (RESET = '1')
        then SV_M      <= ST_RAM_00;
            WRITE_M <= '0';
    else
        if (CLK'event and CLK = '1')
            then
                if (IN_NEXT_STATE = '1')
                    then SV_M      <= n_SV;
                        COUNT_ADR_M <= n_COUNT_ADR;
                        COUNT_DAT_M <= n_COUNT_DAT;
                        WRITE_M <= n_WRITE;
                    else SV_M      <= SV_M;
                        COUNT_ADR_M <= COUNT_ADR_M;
                        COUNT_DAT_M <= COUNT_DAT_M;
                        WRITE_M <= WRITE_M;
                    end if;
                end if;
            end if;
    end if;
end process;

SREG_S_PROC: process (RESET, SV_M, not_CLK) --Slave
begin
    if (RESET = '1')
        then SV      <= ST_RAM_00;
    else
        if (not_CLK'event and not_CLK = '1')
            then SV      <= SV_M;
                COUNT_ADR <= COUNT_ADR_M;
                COUNT_DAT <= COUNT_DAT_M;
            end if;
        end if;
    end process;
```



```
IL_DL_PROC: process (GO_S, SV, COUNT_ADR, COUNT_DAT, PLUS_S, MINUS_S, COUNT_DAT_INPUT)
begin
    --setze fuer alle Zustaende
    n_WRITE <= '0'; --kein Schreiben
    UB1 <= '0'; --Upper Byte Ein (0=Ein 1=Aus)
    LB1 <= '0'; --Lower Byte Ein (0=Ein 1=Aus)
    case SV is
        when ST_RAM_00 =>
            if (GO_S = '1')
                then
                    -- RAM01
                    n_COUNT_ADR <= b"00000000000000000000"; -- Adress Zaehler Neustart
                    n_COUNT_DAT <= b"1111111111111111"; -- Daten Zaehler Neustart
                    WE <= '1'; --Aus (0=Ein 1=Aus) 0
                    OE <= '1'; --Aus (0=Ein 1=Aus)
                    CE1 <= '1'; --Aus (0=Ein 1=Aus)
                    STOP <= '0'; -- Aus(0=Aus 1=Ein)
                    n_SV <= ST_RAM_01; -- Zustandsuebergang
                else
                    --RAM00
                    n_COUNT_ADR <= b"00000000000000000000"; -- Adress Zaehler Neustart
                    n_COUNT_DAT <= b"1111111111111111"; -- Daten Zaehler Neustart
                    WE <= '1'; --Aus 0
                    OE <= '1'; --Aus
                    CE1 <= '1'; --Aus
                    STOP <= '0'; --Aus
                    n_SV <= ST_RAM_00; -- GO = '0'
                end if;

        when ST_RAM_01 =>
            -- RAM02
            n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
            n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
            WE <= '0'; --Ein
            OE <= '1'; --Aus
            CE1 <= '0'; --Ein
            STOP <= '0'; --Aus
            n_WRITE <= '1'; --schreiben
            n_SV <= ST_RAM_02; -- Zustandsuebergang

        when ST_RAM_02 =>
            if (COUNT_ADR = b"1111111111111111")
                then
                    -- RAM05
                    n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
                    n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
                    WE <= '1'; --Aus (0=Ein 1=Aus)
                    OE <= '1'; --Aus (0=Ein 1=Aus)
                    CE1 <= '0'; --Ein (0=Ein 1=Aus)
                    STOP <= '0'; -- Aus(0=Aus 1=Ein)
                    n_SV <= ST_RAM_03; -- COUNT_ADR < FF
                else
                    --RAM03
                    n_COUNT_ADR <= COUNT_ADR+1; -- Adress Zaehler inkrementieren
                    n_COUNT_DAT <= COUNT_DAT-1; -- Daten Zaehler dekrementieren
                    WE <= '1'; --Aus
                    OE <= '1'; --Aus
                    CE1 <= '0'; --Ein
                    STOP <= '0'; --Aus
                    n_SV <= ST_RAM_04; -- COUNT_ADR = FF
                end if;

        when ST_RAM_03 =>
            if (GO_S = '0')
                then
                    -- RAM06
                    n_COUNT_ADR <= b"00000000000000000000"; -- Wert wird null
                    n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
                    WE <= '1'; --Aus (0=Ein 1=Aus)
                    OE <= '1'; --Aus (0=Ein 1=Aus)
                    CE1 <= '0'; --Ein (0=Ein 1=Aus)
                    STOP <= '1'; -- Ein(0=Aus 1=Ein)
                    n_SV <= ST_RAM_05; -- GO_S ='0'
```



```
else
--RAM05
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus
OE <= '1'; --Aus
CE1 <= '0'; --Ein
STOP <= '0'; --Aus
n_SV <= ST_RAM_03; -- GO_S ='1'
end if;

when ST_RAM_04 =>
-- RAM04
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '1'; --Aus (0=Ein 1=Aus)
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '0'; -- Aus(0=Aus 1=Ein)
n_SV <= ST_RAM_01; -- Zustandsübergang

when ST_RAM_05 =>
if (GO_S = '0')
then
-- RAM08
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '0'; --Ein (0=Ein 1=Aus)
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '1'; -- Ein(0=Aus 1=Ein)
n_SV <= ST_RAM_06; -- GO_S ='0'
else
--RAM07
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus
OE <= '0'; --Ein
CE1 <= '0'; --Ein
STOP <= '1'; --Ein
n_SV <= ST_RAM_00; -- GO_S ='1'
end if;

when ST_RAM_06 =>
if (PLUS_S = '1')
then
-- RAM09
n_COUNT_ADR <= COUNT_ADR+1; -- Wert wird erhöht
n_COUNT_DAT <= COUNT_DAT_INPUT; --Daten einlesen
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '0'; --Ein (0=Ein 1=Aus)
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '0'; -- Aus(0=Aus 1=Ein)
n_SV <= ST_RAM_07; -- PLUS_S ='1'
else
--RAM11
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus
OE <= '0'; --Ein
CE1 <= '0'; --Ein
STOP <= '1'; --Ein
n_SV <= ST_RAM_08; -- PLUS_S ='0'
end if;

when ST_RAM_07 =>
if (PLUS_S = '0')
then
-- RAM14
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '0'; --Ein (0=Ein 1=Aus)
```



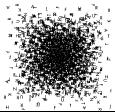
```
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '1'; -- Ein(0=Aus 1=Ein)
n_SV <= ST_RAM_05; -- PLUS_S ='0'
else
--RAM10
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT_INPUT; -- DATEN einlesen
WE <= '1'; --Aus
OE <= '0'; --Ein
CE1 <= '0'; --Ein
STOP <= '0'; --Aus
n_SV <= ST_RAM_07; -- PLUS_S ='1'
end if;

when ST_RAM_08 =>
if (MINUS_S = '1')
then
--RAM12
n_COUNT_ADR <= COUNT_ADR-1; -- Wert wird verringert
n_COUNT_DAT <= COUNT_DAT_INPUT; -- Daten einlesen
WE <= '1'; --Aus
OE <= '0'; --Ein
CE1 <= '0'; --Ein
STOP <= '0'; --Aus
n_SV <= ST_RAM_09; -- MINUS_S ='1'
else
-- RAM14
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '0'; --Ein (0=Ein 1=Aus)
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '1'; -- Ein(0=Aus 1=Ein)
n_SV <= ST_RAM_05; -- PLUS_S ='0'
end if;

when ST_RAM_09 =>
if (MINUS_S = '0')
then
-- RAM14
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT; -- Wert bleibt gleich
WE <= '1'; --Aus (0=Ein 1=Aus)
OE <= '0'; --Ein (0=Ein 1=Aus)
CE1 <= '0'; --Ein (0=Ein 1=Aus)
STOP <= '1'; -- Ein(0=Aus 1=Ein)
n_SV <= ST_RAM_05; -- PLUS_S ='0'
else
--RAM13
n_COUNT_ADR <= COUNT_ADR; -- Wert bleibt gleich
n_COUNT_DAT <= COUNT_DAT_INPUT; -- Daten einlesen
WE <= '1'; --Aus
OE <= '0'; --Ein
CE1 <= '0'; --Ein
STOP <= '0'; --Aus
n_SV <= ST_RAM_09; -- MINUS_S ='1'
end if;

when others =>
-- RAM00
n_COUNT_ADR <= b"00000000000000000000"; -- Adress Zaehler Neustart
n_COUNT_DAT <= b"1111111111111111"; -- Daten Zaehler Neustart
WE <= '1'; --Aus
OE <= '1'; --Aus
CE1 <= '1'; --Aus
STOP <= '0'; --Aus
n_SV <= ST_RAM_00;
end case;
end process;

STATE_DISPL_PROC: process (SV, n_SV, DISPL_STATE_SV, DISPL_STATE_n_SV, DISPL_ADR, DISPL_DAT,
COUNT_ADR, COUNT_DAT) -- Zustandsanzeige
begin
```



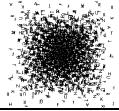
```
DISPL_STATE_SV    <= conv_std_logic_vector(TYPE_STATE'pos(SV),8); --Zustandsumwandlung in 8
Bit
DISPL_STATE_n_SV <= conv_std_logic_vector(TYPE_STATE'pos(n_SV),8);
if (DISPL_ADR = '0')
then
-- Aktuellen Zustand anzeigen
  DISPL1_SV(0) <= DISPL_STATE_SV(0); --Bit0
  DISPL1_SV(1) <= DISPL_STATE_SV(1); --Bit1
  DISPL1_SV(2) <= DISPL_STATE_SV(2); --Bit2
  DISPL1_SV(3) <= DISPL_STATE_SV(3); --Bit3
  DISPL2_SV(0) <= DISPL_STATE_SV(4); --usw.
  DISPL2_SV(1) <= DISPL_STATE_SV(5);
  DISPL2_SV(2) <= DISPL_STATE_SV(6);
  DISPL2_SV(3) <= DISPL_STATE_SV(7);
else
-- Adresse anzeigen (erste 8 Bit)
  DISPL1_SV(0) <= COUNT_ADR(0); --Bit0
  DISPL1_SV(1) <= COUNT_ADR(1); --Bit1
  DISPL1_SV(2) <= COUNT_ADR(2); --Bit2
  DISPL1_SV(3) <= COUNT_ADR(3); --Bit3
  DISPL2_SV(0) <= COUNT_ADR(4); --usw.
  DISPL2_SV(1) <= COUNT_ADR(5);
  DISPL2_SV(2) <= COUNT_ADR(6);
  DISPL2_SV(3) <= COUNT_ADR(7);
end if;

if (DISPL_DAT = '0')
then
-- Folgezustand anzeigen
  DISPL1_n_SV(0) <= DISPL_STATE_n_SV(0);
  DISPL1_n_SV(1) <= DISPL_STATE_n_SV(1);
  DISPL1_n_SV(2) <= DISPL_STATE_n_SV(2);
  DISPL1_n_SV(3) <= DISPL_STATE_n_SV(3);
  DISPL2_n_SV(0) <= DISPL_STATE_n_SV(4);
  DISPL2_n_SV(1) <= DISPL_STATE_n_SV(5);
  DISPL2_n_SV(2) <= DISPL_STATE_n_SV(6);
  DISPL2_n_SV(3) <= DISPL_STATE_n_SV(7);
else
--Daten anzeigen (erste 8 Bit)
  DISPL1_n_SV(0) <= COUNT_DAT(0);
  DISPL1_n_SV(1) <= COUNT_DAT(1);
  DISPL1_n_SV(2) <= COUNT_DAT(2);
  DISPL1_n_SV(3) <= COUNT_DAT(3);
  DISPL2_n_SV(0) <= COUNT_DAT(4);
  DISPL2_n_SV(1) <= COUNT_DAT(5);
  DISPL2_n_SV(2) <= COUNT_DAT(6);
  DISPL2_n_SV(3) <= COUNT_DAT(7);
end if;
end process;

-- Adressen Output
COUNT_ADR_OUT <= n_COUNT_ADR;
-- Daten lesen
COUNT_DAT_INPUT <= COUNT_DAT_INOUT;
-- Daten schreiben
-- Tri-State Buffer control
COUNT_DAT_INOUT <= n_COUNT_DAT when WRITE_M = '1' else (others=>'Z'); --geht nicht in einem
Prozess weil Fehler

end Behavioral;
```

Datei 10-1: ..\VHDL_Bausteine\SRAM_25MHZ_255_BYTExSRAM_25MHZ_255_BYTE.vhd



10.8 Testumgebung

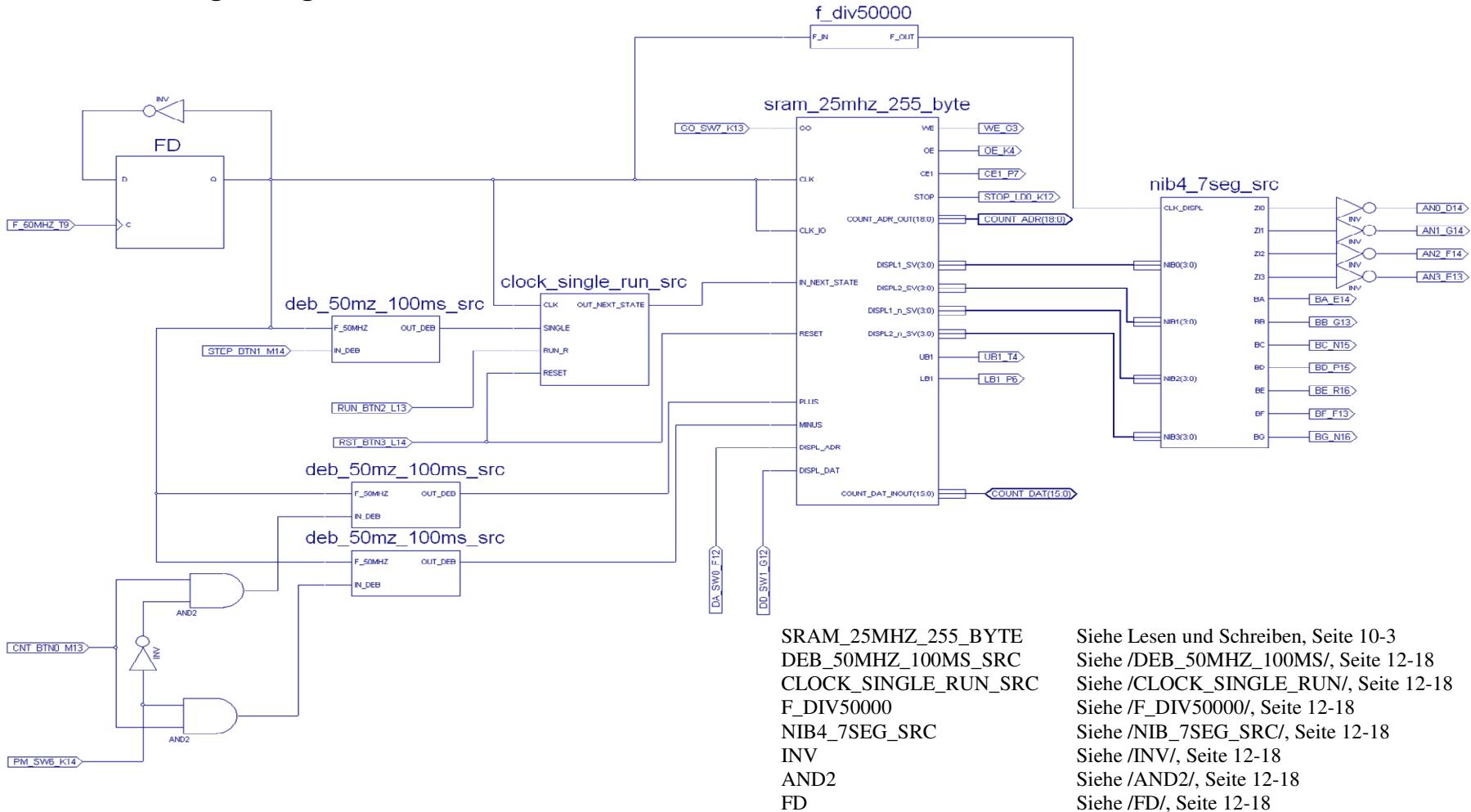


Abbildung 10-6: Testumgebung TEST2_SRAM_25MHZ_255_BYT

SRAM_25MHZ_255_BYTE
DEB_50MHZ_100MS_SRC
CLOCK_SINGLE_RUN_SRC
F_DIV50000
NIB4_7SEG_SRC
INV
AND2
FD

Siehe Lesen und Schreiben, Seite 10-3
Siehe /DEB_50MHZ_100MS/, Seite 12-18
Siehe /CLOCK_SINGLE_RUN/, Seite 12-18
Siehe /F_DIV50000/, Seite 12-18
Siehe /NIB_7SEG_SRC/, Seite 12-18
Siehe /INV/, Seite 12-18
Siehe /AND2/, Seite 12-18
Siehe /FD/, Seite 12-18



10.9 UCF-Datei

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

```
NET "AN0_D14" LOC = "D14" ;
NET "AN1_G14" LOC = "G14" ;
NET "AN2_F14" LOC = "F14" ;
NET "AN3_E13" LOC = "E13" ;
NET "BA_E14" LOC = "E14" ;
NET "BB_G13" LOC = "G13" ;
NET "BC_N15" LOC = "N15" ;
NET "BD_P15" LOC = "P15" ;
NET "BE_R16" LOC = "R16" ;
NET "BF_F13" LOC = "F13" ;
NET "BG_N16" LOC = "N16" ;
NET "CE1_P7" LOC = "P7" ;
NET "CNT_BTN0_M13" LOC = "M13" ;
NET "COUNT_ADR<0>" LOC = "L5" ;
NET "COUNT_ADR<10>" LOC = "G5" ;
NET "COUNT_ADR<11>" LOC = "H3" ;
NET "COUNT_ADR<12>" LOC = "H4" ;
NET "COUNT_ADR<13>" LOC = "J4" ;
NET "COUNT_ADR<14>" LOC = "J3" ;
NET "COUNT_ADR<15>" LOC = "K3" ;
NET "COUNT_ADR<16>" LOC = "K5" ;
NET "COUNT_ADR<17>" LOC = "L3" ;
NET "COUNT_ADR<1>" LOC = "N3" ;
NET "COUNT_ADR<2>" LOC = "M4" ;
NET "COUNT_ADR<3>" LOC = "M3" ;
NET "COUNT_ADR<4>" LOC = "L4" ;
NET "COUNT_ADR<5>" LOC = "G4" ;
NET "COUNT_ADR<6>" LOC = "F3" ;
NET "COUNT_ADR<7>" LOC = "F4" ;
NET "COUNT_ADR<8>" LOC = "E3" ;
NET "COUNT_ADR<9>" LOC = "E4" ;
NET "COUNT_DAT<0>" LOC = "N7" ;
NET "COUNT_DAT<10>" LOC = "F2" ;
NET "COUNT_DAT<11>" LOC = "H1" ;
NET "COUNT_DAT<12>" LOC = "J2" ;
NET "COUNT_DAT<13>" LOC = "L2" ;
NET "COUNT_DAT<14>" LOC = "P1" ;
NET "COUNT_DAT<15>" LOC = "R1" ;
NET "COUNT_DAT<1>" LOC = "T8" ;
NET "COUNT_DAT<2>" LOC = "R6" ;
NET "COUNT_DAT<3>" LOC = "T5" ;
NET "COUNT_DAT<4>" LOC = "R5" ;
NET "COUNT_DAT<5>" LOC = "C2" ;
NET "COUNT_DAT<6>" LOC = "C1" ;
NET "COUNT_DAT<7>" LOC = "B1" ;
NET "COUNT_DAT<8>" LOC = "D3" ;
NET "COUNT_DAT<9>" LOC = "P8" ;
NET "DA_SW0_F12" LOC = "F12" ;
NET "DD_SW1_G12" LOC = "G12" ;
NET "F_50MHZ_T9" LOC = "T9" ;
NET "GO_SW7_K13" LOC = "K13" ;
NET "LB1_P6" LOC = "P6" ;
NET "OE_K4" LOC = "K4" ;
NET "PM_SW6_K14" LOC = "K14" ;
NET "RST_BTN3_L14" LOC = "L14" ;
NET "RUN_BTN2_L13" LOC = "L13" ;
NET "STEP_BTN1_M14" LOC = "M14" ;
NET "STOP_LDO_K12" LOC = "K12" ;
NET "UB1_T4" LOC = "T4" ;
NET "WE_G3" LOC = "G3" ;
```

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

Datei 10-2: ..\VHDL_Bausteine\TEST2_SRAM_25MHZ_255_BYTExtest2_sram_25mhz_255_byte.ucf



10.10 Steuerungsbelegung Testumgebung

Buttons	BTN0:	Adresse erhöhen/verringern
	BTN1:	STEP
	BTN2:	RUN
	BTN3:	RESET
Switches	SW0:	DISPL_ADR (0:SV; 1:COUNT_ADR)
	SW1:	DISPL_DAT (0: n_SV; 1:COUNT_DAT)
	SW2:	nicht benutzt
	SW3:	nicht benutzt
	SW4:	nicht benutzt
	SW5:	nicht benutzt
	SW6:	PLUSMINUS(0:MINUS; 1:PLUS)
	SW7:	GO (0:GO:=0; 1:GO:=1)
LEDs	LD0:	nicht benutzt
	LD1:	nicht benutzt
	LD2:	nicht benutzt
	LD3:	nicht benutzt
	LD4:	nicht benutzt
	LD5:	nicht benutzt
	LD6:	nicht benutzt
	LD7:	STOP

Tabelle 10-4: Steuerungsbelegung Testumgebung TEST2_SRAM_25MHZ_255_BYT

10.11 Zuordnung Zustand / Anzeige

Die Anzeige gibt den aktuellen und folgenden Zustand des Funktionsmoduls Testmodul SRAM_25MHZ_255_BYT verkürzt aus. Siehe auch **Tabelle 10-3: Liste der Prozesse TEST_SRAM_25MHZ_255_BYT**, Seite 10-6 und **Tabelle 10-1: Variablendefinition TEST_SRAM_25MHZ_255_BYT**, Seite 10-3.



11 Anhang

11.1 VHDL-Programm BIT_REGISTER aus Testumgebung

```
-- CTRL_BIT_REGISTER
-- Einlesen der einzelnen Werte für bestimmte Bits, Berechnung der Parität und Ausgabe als Byte
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 08.01.2013
-- Bearbeiter: mharndt
-- Geaendert: 24.04.2013 (Erweiterungen für Testumgebung)
-- Umstellung auf: rising_edge(CLK) und falling_edge(CLK) und http://www.sigasi.com/content/clock-
edge-detection
-- Optimierungen aus: http://www.lothar-miller.de/s9y/categories/37-FSM

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_BIT_REGISTER is
    Port (EN_BIT_i : in std_logic_vector (7 downto 0); --Eingangsvariable, Einschalten des
          Bitregisters i ##(8 downto 0) zum Testen auf (7 downto 0) verringert##%
                  EN_BIT_8 : in std_logic; --Eingangsvariable, Einschalten des Bitregisters 8 ##NurZumTesten
          Eingang dauerhaft auf 1 (Vcc) setzen##%
    BIT_VALUE : in std_logic; -- Eingangsvariable, Wert des aktuellen Bits
    BYTE_OUT : out std_logic_vector (7 downto 0); --Ausgangsvariable, Byte, 8bit, Vektor
    PARITY_OK : out std_logic; --Ausgangsvariable, Parität i.O.

    CLK : in std_logic; --Taktvariable
    IN_NEXT_STATE: in std_logic; --1:Zustandsuebergang möglich
    RESET : in std_logic; --1: Initialzustand annehmen

    DISPL1_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl1, binärzahl
    DISPL2_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl2,
binärzahl
    DISPL1_n_SV : out std_logic_vector (3 downto 0); --Folgezustand Zahl1, binärzahl
    DISPL2_n_SV : out std_logic_vector (3 downto 0)); --Folgezustand Zahl2,
binärzahl

    end CTRL_BIT_REGISTER;

architecture Behavioral of CTRL_BIT_REGISTER is

type TYPE_STATE_BR_BIT0 is
    (ST_BR_EN_BIT0_0, --Zustaende BIT_REGISTER BIT0
     ST_BR_EN_BIT0_1);

type TYPE_STATE_BR_BIT1 is
    (ST_BR_EN_BIT1_0, --Zustaende BIT_REGISTER BIT1
     ST_BR_EN_BIT1_1);

type TYPE_STATE_BR_BIT2 is
    (ST_BR_EN_BIT2_0, --Zustaende BIT_REGISTER BIT2
     ST_BR_EN_BIT2_1);

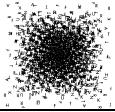
type TYPE_STATE_BR_BIT3 is
    (ST_BR_EN_BIT3_0, --Zustaende BIT_REGISTER BIT3
     ST_BR_EN_BIT3_1);

type TYPE_STATE_BR_BIT4 is
    (ST_BR_EN_BIT4_0, --Zustaende BIT_REGISTER BIT4
     ST_BR_EN_BIT4_1);

type TYPE_STATE_BR_BIT5 is
    (ST_BR_EN_BIT5_0, --Zustaende BIT_REGISTER BIT5
     ST_BR_EN_BIT5_1);

type TYPE_STATE_BR_BIT6 is
    (ST_BR_EN_BIT6_0, --Zustaende BIT_REGISTER BIT6
     ST_BR_EN_BIT6_1);

type TYPE_STATE_BR_BIT7 is
    (ST_BR_EN_BIT7_0, --Zustaende BIT_REGISTER BIT7
```



```
ST_BR_EN_BIT7_1);

type TYPE_STATE_BR_BIT8 is
    (ST_BR_EN_BIT8_0, --Zustaende BIT_REGISTER BIT8
     ST_BR_EN_BIT8_1);

signal SV_BR_BIT0 : TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0
signal n_SV_BR_BIT0: TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0,
neuer Wert
signal SV_BR_BIT0_M: TYPE_STATE_BR_BIT0 := ST_BR_EN_BIT0_0; --Zustandsvariable BIT_REGSITER BIT0,
Ausgang Master

signal SV_BR_BIT1 : TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1
signal n_SV_BR_BIT1: TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1,
neuer Wert
signal SV_BR_BIT1_M: TYPE_STATE_BR_BIT1 := ST_BR_EN_BIT1_0; --Zustandsvariable BIT_REGSITER BIT1,
Ausgang Master

signal SV_BR_BIT2 : TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2
signal n_SV_BR_BIT2: TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2,
neuer Wert
signal SV_BR_BIT2_M: TYPE_STATE_BR_BIT2 := ST_BR_EN_BIT2_0; --Zustandsvariable BIT_REGSITER BIT2,
Ausgang Master

signal SV_BR_BIT3 : TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3
signal n_SV_BR_BIT3: TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3,
neuer Wert
signal SV_BR_BIT3_M: TYPE_STATE_BR_BIT3 := ST_BR_EN_BIT3_0; --Zustandsvariable BIT_REGSITER BIT3,
Ausgang Master

signal SV_BR_BIT4 : TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4
signal n_SV_BR_BIT4: TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4,
neuer Wert
signal SV_BR_BIT4_M: TYPE_STATE_BR_BIT4 := ST_BR_EN_BIT4_0; --Zustandsvariable BIT_REGSITER BIT4,
Ausgang Master

signal SV_BR_BIT5 : TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5
signal n_SV_BR_BIT5: TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5,
neuer Wert
signal SV_BR_BIT5_M: TYPE_STATE_BR_BIT5 := ST_BR_EN_BIT5_0; --Zustandsvariable BIT_REGSITER BIT5,
Ausgang Master

signal SV_BR_BIT6 : TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6
signal n_SV_BR_BIT6: TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6,
neuer Wert
signal SV_BR_BIT6_M: TYPE_STATE_BR_BIT6 := ST_BR_EN_BIT6_0; --Zustandsvariable BIT_REGSITER BIT6,
Ausgang Master

signal SV_BR_BIT7 : TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7
signal n_SV_BR_BIT7: TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7,
neuer Wert
signal SV_BR_BIT7_M: TYPE_STATE_BR_BIT7 := ST_BR_EN_BIT7_0; --Zustandsvariable BIT_REGSITER BIT7,
Ausgang Master

signal SV_BR_BIT8 : TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8
signal n_SV_BR_BIT8: TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8,
neuer Wert
signal SV_BR_BIT8_M: TYPE_STATE_BR_BIT8 := ST_BR_EN_BIT8_0; --Zustandsvariable BIT_REGSITER BIT8,
Ausgang Master

signal BYTE_VEC : std_logic_vector (8 downto 0) := b"000000000"; -- Vektor, BIT_REGSITER, vor
Auswertung der Checksume

signal STATE_SV : std_logic_vector (7 downto 0) := x"00"; -- aktueller Zustand in 8 Bit, binär
signal STATE_n_SV : std_logic_vector (7 downto 0) := x"00"; -- Folgezustand in 8 Bit, binär

--signal not_CLK : std_logic; --negierte Taktvariable

--signal TMP00 : std_logic; --temporärer Zwischenwert, Paritätsprüfung
--signal TMP01 : std_logic;
--signal TMP02 : std_logic;
--signal TMP03 : std_logic;
--signal TMP10 : std_logic;
--signal TMP11 : std_logic;
--signal TMP20 : std_logic;
```

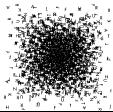


```
begin

--NOT_CLK_PROC: process (CLK) --negieren Taktvariable
--begin
--  not_CLK <= not CLK;
--end process;

SREG_M_PROC: process (RESET, n_SV_BR_BIT0, n_SV_BR_BIT1, n_SV_BR_BIT2, n_SV_BR_BIT3, n_SV_BR_BIT4,
n_SV_BR_BIT5, n_SV_BR_BIT6, n_SV_BR_BIT7, n_SV_BR_BIT8, CLK) --Master
begin
  if (RESET =='1')
    then SV_BR_BIT0_M <= ST_BR_EN_BIT0_0;
    SV_BR_BIT1_M <= ST_BR_EN_BIT1_0;
    SV_BR_BIT2_M <= ST_BR_EN_BIT2_0;
    SV_BR_BIT3_M <= ST_BR_EN_BIT3_0;
    SV_BR_BIT4_M <= ST_BR_EN_BIT4_0;
    SV_BR_BIT5_M <= ST_BR_EN_BIT5_0;
    SV_BR_BIT6_M <= ST_BR_EN_BIT6_0;
    SV_BR_BIT7_M <= ST_BR_EN_BIT7_0;
    SV_BR_BIT8_M <= ST_BR_EN_BIT8_0;
  else
    if rising_edge(CLK)
    then
      if (IN_NEXT_STATE = '1')
        then SV_BR_BIT0_M <= n_SV_BR_BIT0;
        SV_BR_BIT1_M <= n_SV_BR_BIT1;
        SV_BR_BIT2_M <= n_SV_BR_BIT2;
        SV_BR_BIT3_M <= n_SV_BR_BIT3;
        SV_BR_BIT4_M <= n_SV_BR_BIT4;
        SV_BR_BIT5_M <= n_SV_BR_BIT5;
        SV_BR_BIT6_M <= n_SV_BR_BIT6;
        SV_BR_BIT7_M <= n_SV_BR_BIT7;
        SV_BR_BIT8_M <= n_SV_BR_BIT8;
      else
        SV_BR_BIT0_M <= SV_BR_BIT0_M;
        SV_BR_BIT1_M <= SV_BR_BIT1_M;
        SV_BR_BIT2_M <= SV_BR_BIT2_M;
        SV_BR_BIT3_M <= SV_BR_BIT3_M;
        SV_BR_BIT4_M <= SV_BR_BIT4_M;
        SV_BR_BIT5_M <= SV_BR_BIT5_M;
        SV_BR_BIT6_M <= SV_BR_BIT6_M;
        SV_BR_BIT7_M <= SV_BR_BIT7_M;
        SV_BR_BIT8_M <= SV_BR_BIT8_M;
      end if;
    end if;
  end if;
end process;

SREG_S_PROC: process (RESET, SV_BR_BIT0_M, SV_BR_BIT1_M, SV_BR_BIT2_M, SV_BR_BIT3_M, SV_BR_BIT4_M,
SV_BR_BIT5_M, SV_BR_BIT6_M, SV_BR_BIT7_M, SV_BR_BIT8_M, CLK) --Slave
begin
  if (RESET = '1')
    then SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
    SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
    SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
    SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
    SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
    SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
    SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
    SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
    SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
  else
    if falling_edge(CLK)
    then
      SV_BR_BIT0 <= SV_BR_BIT0_M;
      SV_BR_BIT1 <= SV_BR_BIT1_M;
      SV_BR_BIT2 <= SV_BR_BIT2_M;
      SV_BR_BIT3 <= SV_BR_BIT3_M;
      SV_BR_BIT4 <= SV_BR_BIT4_M;
      SV_BR_BIT5 <= SV_BR_BIT5_M;
      SV_BR_BIT6 <= SV_BR_BIT6_M;
      SV_BR_BIT7 <= SV_BR_BIT7_M;
      SV_BR_BIT8 <= SV_BR_BIT8_M;
    end if;
  end if;
end process;
```



```
BIT_REGISTER_EN_BIT_0_PROC:process (SV_BR_BIT0, n_SV_BR_BIT0, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit0
begin
  case SV_BR_BIT0 is
    when ST_BR_EN_BIT0_0 =>
      BYTE_OUT(0)<='0';
      BYTE_VEC(0)<='0';
      if (EN_BIT_i(0) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_EN_BIT0_1
            then n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1;
            else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
          end if;
        else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
      end if;

    when ST_BR_EN_BIT0_1 =>
      -- EN_BIT_0_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(0) = 1
      BYTE_OUT(0)<='1';
      BYTE_VEC(0)<='1';
      if (EN_BIT_i(0) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1;
            else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
          end if;
        else n_SV_BR_BIT0 <= ST_BR_EN_BIT0_1; -- BIT_VALUE = 0
      end if;

    when others =>
      n_SV_BR_BIT0 <= ST_BR_EN_BIT0_0;
  end case;
end process;

BIT_REGISTER_EN_BIT_1_PROC:process (SV_BR_BIT1, n_SV_BR_BIT1, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
  case SV_BR_BIT1 is
    when ST_BR_EN_BIT1_0 =>
      BYTE_OUT(1)<='0';
      BYTE_VEC(1)<='0';
      if (EN_BIT_i(1) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT1_1
            then n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1;
            else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
          end if;
        else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
      end if;

    when ST_BR_EN_BIT1_1 =>
      -- EN_BIT_1_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(1) = 1
      BYTE_OUT(1)<='1';
      BYTE_VEC(1)<='1';
      if (EN_BIT_i(1) = '1')
        then
          if (BIT_VALUE = '1')
            then n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1;
            else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
          end if;
        else n_SV_BR_BIT1 <= ST_BR_EN_BIT1_1; -- BIT_VALUE = 0
      end if;

    when others =>
      n_SV_BR_BIT1 <= ST_BR_EN_BIT1_0;
  end case;
end process;

BIT_REGISTER_EN_BIT_2_PROC:process (SV_BR_BIT2, n_SV_BR_BIT2, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
  case SV_BR_BIT2 is
    when ST_BR_EN_BIT2_0 =>
      BYTE_OUT(2)<='0';
      BYTE_VEC(2)<='0';
      if (EN_BIT_i(2) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT2_1
```



```
        then n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1;
        else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
    end if;
else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
end if;

when ST_BR_EN_BIT2_1 =>
-- EN_BIT_2_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(2) = 1
BYTE_OUT(2)<='1';
BYTE_VEC(2)<='1';
if (EN_BIT_i(2) = '1')
then
    if (BIT_VALUE = '1')
        then n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1;
        else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
    end if;
else n_SV_BR_BIT2 <= ST_BR_EN_BIT2_1; -- BIT_VALUE = 0
end if;

when others =>
    n_SV_BR_BIT2 <= ST_BR_EN_BIT2_0;
end case;
end process;

BIT_REGISTER_EN_BIT_3_PROC:process (SV_BR_BIT3, n_SV_BR_BIT3, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
case SV_BR_BIT3 is
when ST_BR_EN_BIT3_0 =>
    BYTE_OUT(3)<='0';
    BYTE_VEC(3)<='0';
    if (EN_BIT_i(3) = '1')
        then
            if (BIT_VALUE = '1')--gehe zu ST_BR_BIT3_1
                then n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1;
                else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
            end if;
        else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
    end if;
when ST_BR_EN_BIT3_1 =>
-- EN_BIT_3_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(3) = 1
    BYTE_OUT(3)<='1';
    BYTE_VEC(3)<='1';
    if (EN_BIT_i(3) = '1')
        then
            if (BIT_VALUE = '1')
                then n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1;
                else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
            end if;
        else n_SV_BR_BIT3 <= ST_BR_EN_BIT3_1; -- BIT_VALUE = 0
    end if;

when others =>
    n_SV_BR_BIT3 <= ST_BR_EN_BIT3_0;
end case;
end process;

BIT_REGISTER_EN_BIT_4_PROC:process (SV_BR_BIT4, n_SV_BR_BIT4, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
case SV_BR_BIT4 is
when ST_BR_EN_BIT4_0 =>
    BYTE_OUT(4)<='0';
    BYTE_VEC(4)<='0';
    if (EN_BIT_i(4) = '1')
        then
            if (BIT_VALUE = '1')--gehe zu ST_BR_BIT4_1
                then n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1;
                else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
            end if;
        else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
    end if;
when ST_BR_EN_BIT4_1 =>
-- EN_BIT_4 = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(4) = 1
    BYTE_OUT(4)<='1';
    BYTE_VEC(4)<='1';
```



```
if (EN_BIT_i(4) = '1')
then
  if (BIT_VALUE = '1')
    then n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1;
    else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
  end if;
else n_SV_BR_BIT4 <= ST_BR_EN_BIT4_1; -- BIT_VALUE = 0
end if;

when others =>
  n_SV_BR_BIT4 <= ST_BR_EN_BIT4_0;
end case;
end process;

BIT_REGISTER_EN_BIT_5_PROC:process (SV_BR_BIT5, n_SV_BR_BIT5, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit1
begin
  case SV_BR_BIT5 is
    when ST_BR_EN_BIT5_0 =>
      BYTE_OUT(5)<='0';
      BYTE_VEC(5)<='0';
      if (EN_BIT_i(5) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT5_1
            then n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1;
            else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
          end if;
        else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
      end if;
    else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
  end if;

when ST_BR_EN_BIT5_1 =>
-- EN_BIT_5_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(5) = 1
  BYTE_OUT(5)<='1';
  BYTE_VEC(5)<='1';
  if (EN_BIT_i(5) = '1')
    then
      if (BIT_VALUE = '1')
        then n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1;
        else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
      end if;
    else n_SV_BR_BIT5 <= ST_BR_EN_BIT5_1; -- BIT_VALUE = 0
  end if;

when others =>
  n_SV_BR_BIT5 <= ST_BR_EN_BIT5_0;
end case;
end process;

BIT_REGISTER_EN_BIT_6_PROC:process (SV_BR_BIT6, n_SV_BR_BIT6, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit6
begin
  case SV_BR_BIT6 is
    when ST_BR_EN_BIT6_0 =>
      BYTE_OUT(6)<='0';
      BYTE_VEC(6)<='0';
      if (EN_BIT_i(6) = '1')
        then
          if (BIT_VALUE = '1')--gehe zu ST_BR_BIT6_1
            then n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1;
            else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
          end if;
        else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
      end if;
    else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
  end if;

when ST_BR_EN_BIT6_1 =>
-- EN_BIT_6 = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(6) = 1
  BYTE_OUT(6)<='1';
  BYTE_VEC(6)<='1';
  if (EN_BIT_i(6) = '1')
    then
      if (BIT_VALUE = '1')
        then n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1;
        else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
      end if;
    else n_SV_BR_BIT6 <= ST_BR_EN_BIT6_1; -- BIT_VALUE = 0
  end if;

when others =>
```



```
n_SV_BR_BIT6 <= ST_BR_EN_BIT6_0;
end case;
end process;

BIT_REGISTER_EN_BIT_7_PROC:process (SV_BR_BIT7, n_SV_BR_BIT7, BIT_VALUE, EN_BIT_i) --BIT_REGISTER
Bit7
begin
case SV_BR_BIT7 is
when ST_BR_EN_BIT7_0 =>
BYTE_OUT(7)<='0';
BYTE_VEC(7)<='0';
if (EN_BIT_i(7) = '1')
then
if (BIT_VALUE = '1')--gehe zu ST_BR_BIT7_1
then n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1;
else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end if;
else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end if;

when ST_BR_EN_BIT7_1 =>
-- EN_BIT_7_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(7) = 1
BYTE_OUT(7)<='1';
BYTE_VEC(7)<='1';
if (EN_BIT_i(7) = '1')
then
if (BIT_VALUE = '1')
then n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1;
else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end if;
else n_SV_BR_BIT7 <= ST_BR_EN_BIT7_1; -- BIT_VALUE = 0
end if;

when others =>
n_SV_BR_BIT7 <= ST_BR_EN_BIT7_0;
end case;
end process;

BIT_REGISTER_EN_BIT_8_PROC:process (SV_BR_BIT8, n_SV_BR_BIT8, BIT_VALUE, EN_BIT_8) --BIT_REGISTER
Bit8 ##Zum Testen EN_BIT_8 statt EN_BIT_i###
begin
case SV_BR_BIT8 is
when ST_BR_EN_BIT8_0 =>
BYTE_VEC(8)<='0';
if (EN_BIT_8 = '1') --##vor Test: if (EN_BIT_i(8) = '1')##
then
if (BIT_VALUE = '1')--gehe zu ST_BR_BIT8_1
then n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1;
else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
end if;
else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
end if;

when ST_BR_EN_BIT8_1 =>
-- EN_BIT_8_S = 1 und BIT_VALUE = 1 dann setze BYTE_OUT(8) = 1
BYTE_VEC(8)<='1';
if (EN_BIT_8 = '1') --##vor Test: if (EN_BIT_i(8) = '1')##
then
if (BIT_VALUE = '1')
then n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1;
else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
end if;
else n_SV_BR_BIT8 <= ST_BR_EN_BIT8_1; -- BIT_VALUE = 0
end if;

when others =>
n_SV_BR_BIT8 <= ST_BR_EN_BIT8_0;
end case;
end process;

PARITY_CHECK_PROC: process (BYTE_VEC) --Paritätsprüfung (Mit VARIABLEN := , STATT SIGNALEN <=)
variable TMP00, TMP01, TMP02, TMP03, TMP10, TMP11, TMP20 : std_logic;
begin
TMP00 := BYTE_VEC(0) xor BYTE_VEC(1);
TMP01 := BYTE_VEC(2) xor BYTE_VEC(3);
TMP02 := BYTE_VEC(4) xor BYTE_VEC(5);
TMP03 := BYTE_VEC(6) xor BYTE_VEC(7);
```



```
TMP10 := TMP00 xor TMP01;
TMP11 := TMP02 xor TMP03;

TMP20 := TMP10 xor TMP11;

if (TMP20 = BYTE_VEC(8))
then PARITY_OK <= '1'; -- Parität korrekt
else PARITY_OK <= '0'; -- Parität fehlerhaft
end if;
end process;

--##Folgender Prozess eingefügt für Testumgebung und angepasst für die Zustandsanzeige von BIT0##
STATE_DISPL_PROC: process (SV_BR_BIT0, n_SV_BR_BIT0, STATE_SV, STATE_n_SV) -- Zustandsanzeige
begin
    STATE_SV <= conv_std_logic_vector(TYPE_STATE_BR_BIT0'pos(SV_BR_BIT0),8); --Zustandsumwandlung
in 8 Bit
    STATE_n_SV <= conv_std_logic_vector(TYPE_STATE_BR_BIT0'pos(n_SV_BR_BIT0),8);
    --für die Anzeige des Zustandes bei BIT1 muss die Nummerierung von BIT0 auf BIT1 geändert werden
usw.

    --aktueller Zustand
    DISPL1_SV(0) <= STATE_SV(0); --Bit0
    DISPL1_SV(1) <= STATE_SV(1); --Bit1
    DISPL1_SV(2) <= STATE_SV(2); --Bit2
    DISPL1_SV(3) <= STATE_SV(3); --Bit3

    DISPL2_SV(0) <= STATE_SV(4); --usw.
    DISPL2_SV(1) <= STATE_SV(5);
    DISPL2_SV(2) <= STATE_SV(6);
    DISPL2_SV(3) <= STATE_SV(7);

    --Folgezustand anzeigen
    DISPL1_n_SV(0) <= STATE_n_SV(0);
    DISPL1_n_SV(1) <= STATE_n_SV(1);
    DISPL1_n_SV(2) <= STATE_n_SV(2);
    DISPL1_n_SV(3) <= STATE_n_SV(3);

    DISPL2_n_SV(0) <= STATE_n_SV(4);
    DISPL2_n_SV(1) <= STATE_n_SV(5);
    DISPL2_n_SV(2) <= STATE_n_SV(6);
    DISPL2_n_SV(3) <= STATE_n_SV(7);

end process;

--BYTE_OUT_PORC: process (BYTE_VEC) --BYTEausgabe
begin
    BYTE_OUT(0) <= BYTE_VEC(0);
    BYTE_OUT(1) <= BYTE_VEC(1);
    BYTE_OUT(2) <= BYTE_VEC(2);
    BYTE_OUT(3) <= BYTE_VEC(3);
    BYTE_OUT(4) <= BYTE_VEC(4);
    BYTE_OUT(5) <= BYTE_VEC(5);
    BYTE_OUT(6) <= BYTE_VEC(6);
    BYTE_OUT(7) <= BYTE_VEC(7);
end process;

end Behavioral;
```

Datei 11-1..\VHDL_Bausteine\TEST_CTRL_BIT_REGISTER\CTRL_BIT_REGISTER.vhd



11.2 VHDL-Programm RS232_TX aus Testumgebung

```
-- CTRL_RS232_TX
-- Input wird bitweise via RS232 versendet
-- Projekt: PROFIBUS MONITOR
-- Ersteller: Martin Harndt
-- Erstellt: 10.01.2013
-- Bearbeiter: mharndt
-- Geaendert: 10.01.2013

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CTRL_RS232_TX_VHDL is
    Port(SEND_BYTE : in std_logic_vector (7 downto 0); --Eingangsvariable, zu Daten Input, 8 bit
          SEND : in std_logic; --Eingangsvariable, Byte OK
          TX : out std_logic; --Ausgangsvariable, Transmit Bit
          READY: out std_logic; --Ausgangsvariable, bereit zum Senden
          CLK : in std_logic; --Taktvariable
          CLK_IO : in std_logic; --Taktvariable,
          --Ein- und Ausgangsregister
          IN_NEXT_STATE: in std_logic; --1:Zustandsuebergang möglich
          RESET : in std_logic; --1: Initialzustand annehmen
          CHOSE_VALUE : in std_logic; --Eingangsvariable, Zählerwert ändern
          DISPL_COUNT : in std_logic; --Eingangsvariable, Counter anzeigen
          DISPL1_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl1, binärzahl
          DISPL2_SV : out std_logic_vector (3 downto 0); --aktueller Zustand Zahl2, binärzahl
          DISPL1_n_SV : out std_logic_vector (3 downto 0); --Folgezustand Zahl1, binärzahl
          DISPL2_n_SV : out std_logic_vector (3 downto 0)); --Folgezustand Zahl2, binärzahl
end CTRL_RS232_TX_VHDL;

architecture Behavioral of CTRL_RS232_TX_VHDL is

type TYPE_STATE is
    (ST_TX_00, --Zustaende CTRL_RS232_TX
     ST_TX_01,
     ST_TX_02,
     ST_TX_03,
     ST_TX_04,
     ST_TX_05,
     ST_TX_06,
     ST_TX_07,
     ST_TX_08,
     ST_TX_09,
     ST_TX_10,
     ST_TX_11);

signal SV : TYPE_STATE; --Zustandsvariable
signal n_SV: TYPE_STATE; --Zustandsvariable, neuer Wert
signal SV_M: TYPE_STATE; --Zustandsvariable, Ausgang Master
signal not_CLK : std_logic; --negierte Taktvariable
signal not_CLK_IO: std_logic; --negierte Taktvariable
          --Ein- und Ausgangsregister
signal STATE_SV : std_logic_vector (7 downto 0); -- aktueller Zustand in 8 Bit, binär
signal STATE_n_SV : std_logic_vector (7 downto 0); -- Folgezustand in 8 Bit, binär
signal SEND_BYTE_S : std_logic_vector (7 downto 0); --Eingangsvariable, Zwischengespeichern im Eingangsregister
signal SEND_S : std_logic; --Eingangsvariable, Zwischengespeichern im Eingangsregister
signal COUNT : std_logic_vector (15 downto 0); --Zaehler, Vektor, 16 Bit
signal n_COUNT : std_logic_vector (15 downto 0); --Zaehler, neuer Wert, Vektor, 16 Bit
signal COUNT_M : std_logic_vector (15 downto 0); --Zaehler, Ausgang Master, Vektor, 16 Bit

signal CNT01 : std_logic_vector (15 downto 0);
signal CNT02 : std_logic_vector (15 downto 0);
signal CNT03 : std_logic_vector (15 downto 0);
signal CNT04 : std_logic_vector (15 downto 0);
signal CNT05 : std_logic_vector (15 downto 0);
signal CNT06 : std_logic_vector (15 downto 0);
signal CNT07 : std_logic_vector (15 downto 0);
signal CNT08 : std_logic_vector (15 downto 0);
signal CNT09 : std_logic_vector (15 downto 0);
signal CNT10 : std_logic_vector (15 downto 0);

--Konstanten, lang
constant long_CNT01 : std_logic_vector := x"1458"; --16 Bit
```



```
constant long_CNT02 : std_logic_vector := x"2C98"; --usw.
constant long_CNT03 : std_logic_vector := x"3D08";
constant long_CNT04 : std_logic_vector := x"5160";
constant long_CNT05 : std_logic_vector := x"65B8";
constant long_CNT06 : std_logic_vector := x"7A10";
constant long_CNT07 : std_logic_vector := x"8E68";
constant long_CNT08 : std_logic_vector := x"A2C0";
constant long_CNT09 : std_logic_vector := x"B718";
constant long_CNT10 : std_logic_vector := x"CB70";

--Konstanten, kurz
constant short_CNT01 : std_logic_vector := x"0003"; --3
constant short_CNT02 : std_logic_vector := x"0006"; --6
constant short_CNT03 : std_logic_vector := x"0009"; --9
constant short_CNT04 : std_logic_vector := x"000C"; --12
constant short_CNT05 : std_logic_vector := x"000F"; --15
constant short_CNT06 : std_logic_vector := x"0012"; --18
constant short_CNT07 : std_logic_vector := x"0015"; --21
constant short_CNT08 : std_logic_vector := x"0018"; --24
constant short_CNT09 : std_logic_vector := x"001B"; --27
constant short_CNT10 : std_logic_vector := x"001E"; --30

begin

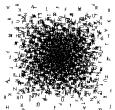
NOT_CLK_PROC: process (CLK) --negieren Taktvariable
begin
    not_CLK <= not CLK;
end process;
NOT_CLK_IO_PROC: process (CLK_IO) --negieren Taktvariable
                    --Ein- und Ausgangsregister
begin
    not_CLK_IO <= not CLK_IO;
end process;

IREG_PROC: process (not_CLK_IO) --Eingangsregister
begin
    if (not_CLK_IO'event and not_CLK_IO = '1') --Eingangsregister
        then SEND_BYTE_S <= SEND_BYTE;
            SEND_S <= SEND;
    end if;
end process;

SREG_M_PROC: process (RESET, n_SV, CLK) --Master
begin
    if (RESET = '1')
        then SV_M      <= ST_TX_00;
    else
        if (CLK'event and CLK = '1')
            then
                if (IN_NEXT_STATE = '1')
                    then SV_M      <= n_SV;
                        COUNT_M <= n_COUNT;
                    else SV_M      <= SV_M;
                        COUNT_M <= COUNT_M;
                end if;
            end if;
        end if;
    end process;

SREG_S_PROC: process (RESET, SV_M, not_CLK) --Slave
begin
    if (RESET = '1')
        then SV      <= ST_TX_00;
    else
        if (not_CLK'event and not_CLK = '1')
            then SV      <= SV_M;
                COUNT <= COUNT_M;
        end if;
    end if;
end process;

CTRL_RS232_RX_PROC:process (SV, COUNT, SEND_S, SEND_BYTE_S, CNT01, CNT02, CNT03, CNT04, CNT05,
CNT06, CNT07, CNT08, CNT09, CNT10) --Daten über RS232 senden
begin
    case SV is
        when ST_TX_00 =>
            if (SEND_S = '1')
                then
```



```
--TX01
n_COUNT <= x"0000"; -- kleiner Zaehler Neustart
TX <= '0'; --Startbit
READY <= '0';
n_SV <= ST_TX_01; --Zustandsübergang
else
--TX00
n_COUNT <= x"0000"; -- kleiner Zaehler Neustart
TX <= '1'; --Idle
READY <= '1'; --Bereit zum Senden
n_SV <= ST_TX_00; --bleibt im gleichen Zustand
end if;

when ST_TX_01 =>
if (COUNT = CNT01) --Zaehler = 5208
then
--TX03
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(0); --Bit 0
READY <= '0';
n_SV <= ST_TX_02; --Zustandsübergang
else
--TX02
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= '0'; --Startbit
READY <= '0';
n_SV <= ST_TX_01; --bleibt im gleichen Zustand
end if;

when ST_TX_02 =>
if (COUNT = CNT02) --Zaehler = 11416
then
--TX05
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(1); --Bit 1
READY <= '0';
n_SV <= ST_TX_03; --Zustandsübergang
else
--TX04
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(0); --Bit 0
READY <= '0';
n_SV <= ST_TX_02; --bleibt im gleichen Zustand
end if;

when ST_TX_03 =>
if (COUNT = CNT03) --Zaehler = 15624
then
--TX07
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(2); --Bit 2
READY <= '0';
n_SV <= ST_TX_04; --Zustandsübergang
else
--TX06
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(1); --Bit 1
READY <= '0';
n_SV <= ST_TX_03; --bleibt im gleichen Zustand
end if;

when ST_TX_04 =>
if (COUNT = CNT04) --Zaehler = 20832
then
--TX09
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(3); --Bit 3
READY <= '0';
n_SV <= ST_TX_05; --Zustandsübergang
else
--TX08
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(2); --Bit 2
READY <= '0';
n_SV <= ST_TX_04; --bleibt im gleichen Zustand
end if;

when ST_TX_05 =>
```



```
if (COUNT = CNT05) --Zaehler = 26040
then
--TX11
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(4); --Bit 4
READY <= '0';
n_SV <= ST_TX_06; --Zustandsübergang
else
--TX10
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(3); --Bit 3
READY <= '0';
n_SV <= ST_TX_05; --bleibt im gleichen Zustand
end if;

when ST_TX_06 =>
if (COUNT = CNT06) --Zaehler = 31248
then
--TX13
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(5); --Bit 5
READY <= '0';
n_SV <= ST_TX_07; --Zustandsübergang
else
--TX12
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(4); --Bit 4
READY <= '0';
n_SV <= ST_TX_06; --bleibt im gleichen Zustand
end if;

when ST_TX_07 =>
if (COUNT = CNT07) --Zaehler = 36456
then
--TX15
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(6); --Bit 6
READY <= '0';
n_SV <= ST_TX_08; --Zustandsübergang
else
--TX14
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(5); --Bit 5
READY <= '0';
n_SV <= ST_TX_07; --bleibt im gleichen Zustand
end if;

when ST_TX_08 =>
if (COUNT = CNT08) --Zaehler = 41664
then
--TX17
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(7); --Bit 7
READY <= '0';
n_SV <= ST_TX_09; --Zustandsübergang
else
--TX16
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(6); --Bit 6
READY <= '0';
n_SV <= ST_TX_08; --bleibt im gleichen Zustand
end if;

when ST_TX_09 =>
if (COUNT = CNT09) --Zaehler = 46872
then
--TX19
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= '1'; --Stoppbit
READY <= '0';
n_SV <= ST_TX_10; --Zustandsübergang
else
--TX18
n_COUNT <= COUNT+1; -- Zaehler erhöhen
TX <= SEND_BYTE_S(7); --Bit 7
READY <= '0';
n_SV <= ST_TX_09; --bleibt im gleichen Zustand
end if;
```



```
when ST_TX_10 =>
  if (COUNT = CNT10) --Zaehler = 52080
    then
      --TX21
      n_COUNT <= x"0000"; -- Zaehler neustart
      TX <= '1'; --Idle
      READY <= '0';
      n_SV <= ST_TX_11; --Zustandsübergang
    else
      --TX20
      n_COUNT <= COUNT+1; -- Zaehler erhöhen
      TX <= '1'; --Stoppbit
      READY <= '0';
      n_SV <= ST_TX_10; --bleibt im gleichen Zustand
    end if;

when ST_TX_11 =>
  if (SEND_S = '0') -- Wenn SEND=0 dann warten auf SEND sonst Idle senden
    then
      --TX00
      n_COUNT <= x"0000"; -- Zaehler neustart
      TX <= '1'; --Idle
      READY <= '1';--Bereit zum Senden
      n_SV <= ST_TX_00; --Zustandsübergang
    else
      --TX22
      n_COUNT <= x"0000"; -- Zaehler neustart
      TX <= '1'; --Idle
      READY <= '0';
      n_SV <= ST_TX_11; --bleibt im gleichen Zustand
    end if;

  when others =>
    -- TX00
    n_COUNT <= x"0000"; -- kleiner Zaehler Neustart
    TX <= '1'; --Idle
    READY <= '0';
    n_SV <= ST_TX_00; --Zustandsübergang
  end case;
end process;

STATE_DISPL_PROC: process (SV, n_SV, DISPL_COUNT, STATE_SV, STATE_n_SV,COUNT) -- Zustandsanzeige
begin
  STATE_SV  <= conv_std_logic_vector(TYPE_STATE'pos( SV),8); --Zustandsumwandlung in 8 Bit
  STATE_n_SV <= conv_std_logic_vector(TYPE_STATE'pos(n_SV),8);
  --aktuellen Zustand anzeigen
  DISPL1_SV(0) <= STATE_SV(0); --Bit0
  DISPL1_SV(1) <= STATE_SV(1); --Bit1
  DISPL1_SV(2) <= STATE_SV(2); --Bit2
  DISPL1_SV(3) <= STATE_SV(3); --Bit3

  DISPL2_SV(0) <= STATE_SV(4); --usw.
  DISPL2_SV(1) <= STATE_SV(5);
  DISPL2_SV(2) <= STATE_SV(6);
  DISPL2_SV(3) <= STATE_SV(7);

  if (DISPL_COUNT ='0')
    then --Folgezustand anzeigen
    DISPL1_n_SV(0)  <= STATE_n_SV(0);
    DISPL1_n_SV(1)  <= STATE_n_SV(1);
    DISPL1_n_SV(2)  <= STATE_n_SV(2);
    DISPL1_n_SV(3)  <= STATE_n_SV(3);

    DISPL2_n_SV(0)  <= STATE_n_SV(4);
    DISPL2_n_SV(1)  <= STATE_n_SV(5);
    DISPL2_n_SV(2)  <= STATE_n_SV(6);
    DISPL2_n_SV(3)  <= STATE_n_SV(7);

  else --Zähler anzeigen
    DISPL1_n_SV(0)  <= COUNT(0);
    DISPL1_n_SV(1)  <= COUNT(1);
    DISPL1_n_SV(2)  <= COUNT(2);
    DISPL1_n_SV(3)  <= COUNT(3);

    DISPL2_n_SV(0)  <= COUNT(4);
    DISPL2_n_SV(1)  <= COUNT(5);
    DISPL2_n_SV(2)  <= COUNT(6);
```



```
DISPL2_n_SV(3) <= COUNT(7);
end if;
end process;

SWITCH_VALUES_PROC: process (CHOOSE_VALUE) --Schaltet zw. langen und kurzem Zaehler um
begin
if (CHOOSE_VALUE = '0')
then
--normale Werte
CNT01 <= long_CNT01;
CNT02 <= long_CNT02;
CNT03 <= long_CNT03;
CNT04 <= long_CNT04;
CNT05 <= long_CNT05;
CNT06 <= long_CNT06;
CNT07 <= long_CNT07;
CNT08 <= long_CNT08;
CNT09 <= long_CNT09;
CNT10 <= long_CNT10;
else
--kurze Werte
CNT01 <= short_CNT01;
CNT02 <= short_CNT02;
CNT03 <= short_CNT03;
CNT04 <= short_CNT04;
CNT05 <= short_CNT05;
CNT06 <= short_CNT06;
CNT07 <= short_CNT07;
CNT08 <= short_CNT08;
CNT09 <= short_CNT09;
CNT10 <= short_CNT10;
end if;
end process;
end Behavioral;
```

Datei 11-2: ..\VHDL_Bausteine\TEST_CTRL_RS232_TX\CTRL_RS232_TX_VHDL.vhd

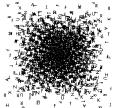


12 Literatur / Web-Seiten

/FELSER/	Prof. Max Felser PROFIBUS Handbuch – Eine Sammlung von Erläuterungen zu PROFIBUS Netzwerken Ausgabe 1.1.5 vom Thursday, December 29, 2011 Berner Fachhochschule (BFH) http://www.profibus.felser.ch/ (abgerufen am 17.04.2013)
/PROFI_POPP/	Manfred Popp PROFIBUS-DP Schnelleinstieg Karlsruhe : PROFIBUS-Nutzerorganisation, 1996
/PROFI_POPP2/	Manfred Popp Profibus-DP-DPV1 : Grundlagen, Tipps und Tricks für Anwender 2., überarb. Aufl. Heidelberg : Hüthig, 2000
/WEBPACK/	Webpack 6.3 Programmpaket der Firma Xilinx zur Programmierung von FPGA-Chips http://www.xilinx.com (abgerufen am 17.04.2013)
/DEV_KIT/	Entwicklerkit der Firma Digilent mit einem FPGA der Firma Xilinx http://www.digilentinc.com (abgerufen am 17.04.2013)
/SRAM/	Datenblatt ISSI IS61LV25616AL-10T 256K x 16 HIGH SPEED ASYNCHRONOUS CMOS STATIC RAM http://www.mouser.com/ds/2/198/61LV25616AL-27059.pdf (abgerufen am 07.05.2013)

12.1 Verzeichnis Bilder

Abbildung 2-1: Konzept Profibusmonitor	2-1
Abbildung 3-1: Aufbau UART-Codierung des Profibus.....	3-2
Abbildung 4-1: Aufbau der Profibusinfrastruktur.....	4-1
Abbildung 5-1: Interface zur Signalumwandlung	5-1
Abbildung 5-2: Schaltplan Interface zur Signalumwandlung	5-2
Abbildung 6-1: Stromversorgung.....	6-1
Abbildung 7-1: Profibusmonitor, schematische Darstellung, /SPARTAN3/ /PROFI/	7-1
Abbildung 7-2: Profibusmonitor, Foto.....	7-1
Abbildung 8-1: Funktionsmodule, Anordnung	8-1
Abbildung 8-2: Wirkungsplan	8-1
Abbildung 8-3: Impulsdiagramm Eingangssignal InAB_INPUT	8-4
Abbildung 8-4: Programmablaufgraph InAB_INPUT	8-6
Abbildung 8-5: Huffmannautomat, ohne Ausgangsregister	8-8
Abbildung 8-6: Testumgebung InAB_INPUT	8-22
Abbildung 8-7: Wirkungsplan BIT_REGISTER	8-25
Abbildung 8-8: Programmablaufgraph BIT_REGISTER	8-26
Abbildung 8-9: Testumgebung BIT_REGISTER	8-36
Abbildung 8-10: Wirkungsplan TELEGRAM_CHECK	8-39
Abbildung 8-11: Programmablaufgraph TELEGRAM_CHECK	8-41
Abbildung 8-12: Testumgebung TELEGRAM_CHECK.....	8-54
Abbildung 8-13: Wirkungsplan RS232_TX	8-57
Abbildung 8-14: Impulsdiagramm RS232_TX	8-58
Abbildung 8-15: Programmablaufgraph RS232_TX	8-60
Abbildung 8-16: Testumgebung RS232_TX	8-67
Abbildung 9-1: Testumgebung PROFIBUS_MONITOR.....	9-1
Abbildung 10-1: Konzept Profibusmonitor mit zusätzlichen Erweiterungen	10-1
Abbildung 10-2: Wirkungsplan TEST_SRAM_25MHZ_255_BYTE	10-2
Abbildung 10-3: Impulsdiagramm SRAM lesen	10-3
Abbildung 10-4: Impulsdiagramm SRAM schreiben.....	10-4
Abbildung 10-5: Wirkungsplan TEST_SRAM_25MHZ_255_BYTE.....	10-5
Abbildung 10-6: Testumgebung TEST2_SRAM_25MHZ_255_BYTE.....	10-13



12.2 Verzeichnis Tabellen

Tabelle 3-1: Pinbelegung D-Sub Stecker für Profibus.....	3-2
Tabelle 8-1: Variablendefinition InAB_INPUT	8-3
Tabelle 8-2: Zählerzeitpunkte für Abtastung InAB_INPUT	8-5
Tabelle 8-3: Belegung der Ausgangsvariablen InAB_INPUT	8-7
Tabelle 8-4: Liste der Prozesse InAB_INPUT	8-8
Tabelle 8-5: Steuerungsbelegung Testumgebung InAB_INPUT	8-24
Tabelle 8-6: Variablendefinition BIT_REGSITER	8-26
Tabelle 8-7: Liste der Prozesse BIT_REGISTER	8-27
Tabelle 8-8: Steuerungsbelegung Testumgebung BIT_REGISTER	8-38
Tabelle 8-9: Variablendefinition TELEGRAM_CHECK	8-40
Tabelle 8-10: Belegung Ausgangsvariablen TELEGRAM_CHECK	8-42
Tabelle 8-11: Liste der Prozesse TELEGRAM_CHECK	8-42
Tabelle 8-12: Steuerungsbelegung Testumgebung TELEGRAM_CHECK	8-56
Tabelle 8-13: Variablendefinition RS232_TX	8-57
Tabelle 8-14: Zählerzeitpunkte RS232_TX	8-59
Tabelle 8-15: Belegung der Ausgangsvariablen RS232_TX	8-61
Tabelle 8-16: Liste der Prozesse RS232_TX	8-61
Tabelle 8-17: Steuerungsbelegung Testumgebung RS232_TX	8-69
Tabelle 9-1: Beschriebene Funktionsmodule PROFIBUS_MONITOR	9-1
Tabelle 9-2: Zusätzliche Funktionsmodule PROFIBUS_MONITOR	9-1
Tabelle 9-3: Steuerungsbelegung Testumgebung PROFIBUS_MONITOR	9-1
Tabelle 10-1: Variablendefinition TEST_SRAM_25MHZ_255_BYTE	10-3
Tabelle 10-2: Belegung der Ausgangsvariablen TEST_SRAM_25MHZ_255_BYTE	10-6
Tabelle 10-3: Liste der Prozesse TEST_SRAM_25MHZ_255_BYTE	10-6
Tabelle 10-4: Steuerungsbelegung Testumgebung TEST2_SRAM_25MHZ_255_BYTE	10-15

12.3 Verzeichnis Dateien

Datei 8-1:..\VHDL_Bausteine\CTRL_InAB_INPUT\CTRL_InAB_INPUT_VHDL.vhd	8-21
Datei 8-2: ..\VHDL_Bausteine\TEST_CTRL_InAB_INPUT\test_ctrl_inab_input_sch.ucf.....	8-23
Datei 8-3:..\VHDL_Bausteine\CTRL_BIT_REGISTER\CTRL_BIT_REGISTER.vhd	8-35
Datei 8-4:..\VHDL_Bausteine\TEST_CTRL_BIT_REGISTER\test_ctrl_bit_register_sch.ucf.....	8-37
Datei 8-5: ..\VHDL_Bausteine\CTRL_TELEGRAM_CHECK\CTRL_TELEGRAM_CHECK.vhd	8-53
Datei 8-6:..\VHDL_Bausteine\TEST_CTRL_TELEGRAM_CHECK\test_ctrl_telegram_check.ucf	8-55
Datei 8-7: ..\VHDL_Bausteine\CTRL_RS232_TX\CTRL_RS232_TX_VHDL.vhd	8-66
Datei 8-8: ..\VHDL_Bausteine\TEST_CTRL_RS232_TX\test_ctrl_rs232_tx_schematic.ucf	8-68
Datei 9-1:..\VHDL_Bausteine\PROFIBUS_MONITOR\profibus_monitor_sch.ucf	9-1
Datei 10-1: ..\VHDL_Bausteine\SRAM_25MHZ_255_BYTE\SRAM_25MHZ_255_BYTE.vhd	10-12
Datei 10-2: ..\VHDL_Bausteine\TEST2_SRAM_25MHZ_255_BYTE\test2_sram_25mhz_255_byte.ucf	10-14
Datei 11-1:..\VHDL_Bausteine\TEST_CTRL_BIT_REGISTER\CTRL_BIT_REGISTER.vhd	11-8
Datei 11-2: ..\VHDL_Bausteine\TEST_CTRL_RS232_TX\CTRL_RS232_TX_VHDL.vhd	11-14

12.4 Quelle Bilder

/7SEG/	http://www.nemsim.com/ece395blimp/fritzing/partsvg/core/breadboard/7-segment%20display.svg (abgerufen am 17.04.2013)
/LAPTOP/	http://web.fh-ludwigshafen.de/rz/home.nsf/Files/745C9DE7E8FC59C6C12576D3002ECB19\$File/200px-Gnome-laptop.svg.png (abgerufen am 17.04.2013)
/PROFI/	http://www.kintercontrol.com/images/product/Modicon%20Momentum.jpg (abgerufen am 17.04.2013)
/SCROLL/	http://openclipart.org/image/250px/svg_to_png/8033/kelan_scroll_-_outline.png (abgerufen am 17.04.2013)
/SPARTAN3/	http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD-top-400.gif (abgerufen am 17.04.2013)



12.5 Verzeichnis Abkürzungen

COUNT	Counter
CTRL	Control
DA	Destination Address
DC	Direct Current
DP	Dezentrale Peripherie
ED	End Delimiter
FC	Function Code
FCS	Frame Check Sequence
FMS	Fieldbus Message Specification
FPGA	Field Programmable Gate Array
LE	Length
LER	Length repeated
NRZ	No-Return-to-Zero
PA	Process Automation
PC	Personal Computer
PDU	Protocol Data Unit
Profibus	Process Field Bus
SA	Source Address
SC	Short Confirmation
SD	Start Delimiter
SPS	Speicher Programmierbare Steuerung
SYN	synchronize
TX	Transmit
UART	Universal Asynchronous Receiver
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WS	Wintersemester



12.6 Bereitgestellte Bauteile

/AND2/ UND-Gatter: Der Ausgang des UND-Gatters wird 1 wenn beide Eingangssignale 1 sind
Mit /WEBPACK/ 6.3 bereitgestelltes Bauteil.



/CLOCK_SINGLE_RUN/ Umschalter: Erlaubt die Umschaltung zwischen Einzel- und Dauertakt



Ordner:
\DVD_STUD\PROJECTS\XILINX_PROJECTS\WP6P3\NEUTRAL\CLOCKING\
CLOCK_SINGLE_RUN

/DEB_50MHZ_100MS/ Entprellerinheit: Entprellt bei einer Taktfrequenz von 50 MHz ein Signal mit etwa 100 ms
Ordner:
\DVD_STUD\PROJECTS\XILINX_PROJECTS\WP6P3\NEUTRAL\DEBOUNCING\
DEB_50MHZ_100MS



/F_DIV50000/ Frequenzteiler 50000 : 1: Teilt die ankommende Frequenz von 50 MHz in 1 KHz
Ordner:
\DVD_STUD\PROJECTS\XILINX_PROJECTS\WP6P3\NEUTRAL\FREQUENCY_DIVIDER\
F_DIV50000



/FD/ D-FlipFlop: Verzögert das Eingangssignal um einen Takt
Mit /WEBPACK/ 6.3 bereitgestelltes Bauteil.



/INV/ Inverter: Gibt das Eingangssignal invertiert auf dem Ausgang aus
Mit /WEBPACK/ 6.3 bereitgestelltes Bauteil.



/NIB_7SEG_SRC/ 7-Segment-Decoder: Ansteuerung von vier 7-Segment-Anzeigen mittels interner Umschaltung über den CLK_DISPL Eingang.
Ordner:
\DVD_STUD\PROJECTS\XILINX_PROJECTS\WP6P3\NEUTRAL\CODING\
\\ NIB4_7SEG

