

LogicBox V4.1

1 Einführung.....	3
2 Geräte.....	4
3 Basiskarten.....	4
4 Submodule.....	5
5 USB.....	6
6 CBus.....	9
7 Module.....	11
8 Signale.....	12
9 LogicPool.....	14
9.1 ADC (SU706).....	14
9.2 ADC4 (SU728).....	19
9.3 ADC8 (SU702).....	21
9.4 ADC16 (SU712, SU720).....	23
9.5 ADC16F (SU712, SU720).....	25
9.6 ADCF (SU706).....	27
9.7 AMPNF (SU715).....	32
9.8 BUSMONITOR.....	33
9.9 COINCCOUNTER.....	35
9.10 COUNT.....	37
9.11 DAC (SU710).....	45
9.12 DAC1 (SU728).....	47
9.13 DAC16 (SU713, SU721).....	49
9.14 DAC16S (SU713, SU721).....	51
9.15 DACR (SU710).....	54
9.16 DELAY (SU711).....	56
9.17 DIO (SU700, ..).....	59
9.18 DISCR (SU703).....	61
9.19 FIFO.....	79
9.20 FREQUENCY.....	82
9.21 GATEGEN.....	84
9.22 LED (SU700, ..).....	89
9.23 LOGIC.....	91
9.24 LUT.....	99
9.25 MULTIPLICITY.....	101
9.26 OFFSGAIN.....	103
9.27 PFIFO (SU730).....	105
9.28 PID.....	108
9.29 PLOGIC.....	110
9.30 PRAM (SU730).....	112

9.31 QDC (SU717).....	114
9.32 RAM.....	118
9.33 RANDOMPULSER.....	120
9.34 SEQUENCER.....	125
9.35 SEQUENCER32.....	128
9.36 SETUP.....	131
9.37 SFIFO (SU705, SU716).....	133
9.38 SU.....	134
9.39 SYNC.....	136
9.40 TDC.....	138
9.41 TEMPERATURE.....	143
9.42 TOSLINK (SU724, SU727).....	145
9.43 WINDOW.....	147
10 Version History.....	149
11 Inbetriebnahme.....	151
11.1 Geräteüberprüfung.....	151
11.2 Rechneranschluß.....	151
11.3 Kommunikation.....	151
12 Synchronisation.....	153

1 Einführung

Dieses Dokument beschreibt ein universelles Steuerungs- und Datenerfassungs-System (**LogicBox**) auf der Basis von unterschiedlichen FPGA-Basiskarten (**DL7xx**) und zusätzlichen digitalen und analogen I/O-Karten (**SU7xx**).

Für den Benutzer stellt sich das System als eine Summe von unmittelbar nutzbaren Funktionsmodulen (**LogicPool**) dar, die im Gerät beliebig zu einer Gesamtschaltung verdrahtet werden können.

FPGAs (Field Programmable Gate Arrays) sind moderne elektronische Bauteile, die erlauben eine immense Vielzahl von logischen Funktionen kompakt und mit sehr hoher Geschwindigkeit zu erzeugen. Die entsprechenden Funktionen werden typischerweise aus einem Grundvorrat von logischen Verschaltungen (LUTs=Lookup Tables) und Speicherelementen (Flipflops, Register) realisiert. Die eigentliche Verschaltung (Konfiguration) wird durch „Schalter“ bestimmt, deren jeweilige Stellung in einem Speicher (RAM) abgelegt sind. Damit ist der Baustein jederzeit durch einen Ladevorgang mit neuen Funktionen programmierbar!

Mit vom Hersteller bereitgestellten CAD-Tools kann dieser Speicherinhalt und damit die entsprechende Funktion des Bausteins bestimmt werden. Im allgemeinen wird die Funktionsweise durch eine spezifische Hardware-Beschreibungssprache (z.B. VHDL) vom Anwender definiert und festgelegt. Dies erfordert allerdings sehr profunde Kenntnisse in der Programmierung und insgesamt einen auch zeitaufwändigen Entwicklungsprozess.

Die Idee der „LogicBox“ ist es nun, dem Anwender schon eine Vielzahl von vorgefertigten Funktionsmodulen bereitzustellen, die der typischen Signalverarbeitung in der Physik entsprechen und die die gewohnte Abstraktionsebene der Funktionsverarbeitung widerspiegeln. Dabei handelt es sich im wesentlichen um digitale Einzelsignale, die durch einfache Logikfunktionen (AND, OR, ..), Speicherelemente (FFs), Zähler (GATEGENERATOR) aber auch durch komplexere Funktionen (z.B. HISTOGRAMMER) verarbeitet und ausgewertet werden.

Jedes Modul hat eine gewisse Anzahl von Eingängen und Ausgängen. Welche Gesamtfunktion damit realisiert wird, kann durch die einfache Verschaltung der Ausgänge auf die Eingänge vom Benutzer selbst bestimmt und festgelegt werden. In diesem Sinne muss lediglich noch eine Art Verbindungsliste geladen werden, die dem Datenfluß der Verarbeitung entspricht. Die Anbindung an die digitale und analoge Aussenwelt wird durch spezielle IO-Submodule (Digital-IOs, Komparatoren, ADCs, DACs, ...) gewährleistet.

Nach dem Einschalten sind alle Eingänge zunächst offen und damit keine Verschaltung und keine Funktion vorhanden. Dies muss in einem Initialisierungsprozess (Setup), am besten mit Unterstützung eines grafischen Programmierertools (z.B. LabVIEW) erfolgen. Anschließend können gewisse Parameter der Module gesetzt oder ausgelesen werden und damit eine komplexe Steuerung der Funktionsmodule und damit des Messprozesses durchgeführt werden. Dieses Prinzip erfordert also (zumindest für die Initialisierung) immer einen externen Rechner, der typischerweise über eine USB-Schnittstelle verbunden ist. Bei bestimmten Geräten ist es auch möglich, dass ein vom Anwender generiertes Setup-File permanent in die LogicBox geladen wird und dann bereits beim Einschalten automatisch zur gewünschten Initialisierung verwendet wird.

2 Geräte

Ein LogicBox-Gerät setzt sich aus der zugrunde liegenden Basiskarte (DL7xx), den aufgesteckten Hardware-SubModulen (SU7xx) und den zusätzlich in der FPGA-Firmware realisierten Funktionsmodulen zusammen. Jede dieser Konfigurationen entspricht einer speziellen LogicPool-Konfigurationsnummer (LPn).

3 Basiskarten

Die Basiskarten enthalten im wesentlichen die digitale Logic (FPGA), die Stecker für die IO-Subkarten, entsprechende Interfacelogik an den Rechner sowie die Spannungsversorgung.

Im folgenden Überblick sind alle zur Zeit verfügbaren Basiskarten aufgeführt:

DL7xx

Typ	Aufbau	IF	FPGA	Sonstiges
DL700	3HE/4 SU	USB 1.1	XC3S400	Prototyp
DL701	3HE/4 SU	USB 1.1	XC3S400	
DL702	6HE/4 SU	VME	Altera	nicht kompatibel
DL703	3HE/4 SU	Parallel	XC3S400	
DL704	3HE/4 SU	Parallel	-	
DL705	3HE/4 SU	USB 2.0	XC3S400	
DL706	NIM/4 SU	USB 2.0	XC3S400	
DL707	3HE/4 SU	USB 2.0	XC3S4000	
DL708	NIM/4 SU	USB 1.1	XC3S400	
DL709	NIM/8 SU	USB 2.0	XC3S4000	
DL710	6HE/4 SU	VME	XC3S4000	
DL711	NIM/8 SU	SU7xx	XC3S6xx	Memory, SFP

Jeder verwendete FPGA-Baustein enthält nur eine maximale Anzahl bestimmter Ressourcen und legt damit in der Summe die Anzahl der möglichen Funktionsmodule fest.

Ein praktikabler Füllgrad (Slices) liegt etwa bei < 80..90% (siehe Gerätetabelle!).

FPGAs

FPGA	Slices	BRAM	GCLK
XC3S400	3584	16=288Kbit	8
XC3S4000	27648	96=1728Kbit	8
XC3S6xxx			

4 Submodule

Die IO-Submodulkarten (SU7xx) stellen das Interface zur physikalischen Außenwelt dar und passen digitale und analoge Signale an die FPGA-Logic auf der Basiskarte an.

Zur Zeit sind die folgenden Submodule verfügbar:

SU7xx

Typ	Beschreibung
SU700	5x TTL I/O – LEMO COAX
SU701	16x TTL I/O – multi connector
SU702	8x ADC, Diff, bipolar
SU703	..4x Discriminator and 1..x TTL I/O –COAX
SU704	5x NIM/TTL I/O – LEMO COAX
SU705	RAM 8M*16
SU706	1x ADC (100 Mhz) ,2x TTL I/O – LEMO COAX
SU707	8x LVDS I/O
SU708	Cascade IF50 Interface
SU709	8x Temperaturesensor
SU710	2x Fast DAC (100 Mhz)
SU711	6x programmable Delayline 0,5ns .. 128 ns
SU712	16x ADC (5 us, 14 Bit)
SU713	16x DAC (14 Bit)
SU714	HAL25 Interface
SU715	NF-Amplifier with prog. Gain
SU716	RAM 16M*32
SU717	QDC, Ladungsempfindlicher ADC
SU718	5x ECL/TTL DIO
SU719	2ch Pressuremeter
SU720	2x8 Ch. ADC, ISO
SU721	2x8 Ch. DAC, ISO
SU722	5x TTL DIO, ISO
SU723	DDS
SU724	Dual TOSLink IF; 2x DIO
SU725	8x ECL Input
SU726	32x LED
SU727	3x Dual TOSLink IFs
SU728	4x ADC (16b, 1us); 1x DAC (16b, 1us)
SU729	SD-Card IF
SU730	8M * 16b Pseudo Static RAM
SU731	4x HSwitch

5 USB

Das USB- Interface mit der LogicBox stellt eine Reihe von Kommandos zur Kommunikation mit dem System bzw. den internen Modulen zur Verfügung. Alle Befehle sind Byte-orientiert, bzw. durch ein ASCII-Zeichen gegeben. Adressen und Daten (1 bis 4 Bytes) sind binär, die Reihenfolge von Wort- und Langwort-Transfers ist Big-Endian (most significant Bytes zuerst!).

System

Kommando	Senden	Empfangen	Bedeutung
#		4 Bytes	Sende ID (Date & Time)
R			System Reset

Transfer

Kommando	Senden (d)	Empfangen	Bedeutung
A	4 Bytes		Setzt Adresspointer A31..0
E	3 Bytes		Setzt Adresspointer A23..0
M	2 Bytes		Setzt Adresspointer A15..0
S	1 Byte		Setzt Adresspointer A7..0
a		4 Bytes	Liest Adresspointer A31..0
+			Erhöht Adresspointer A31..0 um 1
-			Erniedrigt Adresspointer A31..0 um 1
N	2 Bytes		Setzt Zähler N15..0 für Blocktransfer(A+) autoinc
F	2 Bytes		Setzt Zähler N15..0 für FIFOtransfer (A)
L	N* 4 Bytes		Schreibt N Langwort(e)(A) (*)
l		N* 4 Bytes	Liest N Langwort(e)(A) (*)
T	N* 3 Bytes		Schreibt N Triple(s)(A) (*)
t		N* 3 Bytes	Liest N Triple(s)(A) (*)
W	N* 2 Bytes		Schreibt N Word(s)(A) (*)
w		N* 2 Bytes	Liest N Word(s)(A) (*)
B	N* 1 Byte		Schreibt N Byte(s)(A) (*)
b		N* 1 Byte	Liest N Byte(s)(A) (*)
D	4 Bytes		Dump data to Address for N/F

(*) Blocktransfers:

Durch den Befehl N bzw. F wird der Blocktransfer für eine bestimmte Anzahl von Transfers vorbereitet und dann durch einen Transferbefehl (L,T,W,B bzw. l,t,w,b) ausgeführt.

Beim Befehl ,N' werden alle Transfers auf aufeinanderfolgenden Adressen durchgeführt (der Adresspointer wird bei jedem Read- oder Writezyklus um 1 erhöht).

Beim Befehl ,F' werden alle Transfers auf der gleichen Adresse (typ. FIFO) durchgeführt.

CRC Check

(zur Zeit noch nicht implementiert!)

Kommando	Senden	Empfangen	Bedeutung
?		1 Byte	Sendet CRC Prüfsumme aller gesendeten Bytes
!		1 Byte	Sendet CRC Prüfsumme aller empfangenen Bytes

Beispiel:

Gesendet	Empfangen	Bedeutung
,#'	00000100	ID-Nummer des Moduls=256
,A'00000001		Setzt Adresspointer A=1
,S'0A		Setzt Adresspointer A=10
,a'	0000000A	Liest Adresspointer
,L'00000200		Schreibt auf akt. Adresse (A=10) den 32Bit Wert=512
,B'FF		Schreibt den Bytewert=255 auf die Adresse A=10
,w'	02FF	Liest den Wert 767 von Adresse A=10
,+'		Erhöht den Adresscounter um 1 (A=11)
,W'0304		Schreibt den Wert 772 nach A=11
,N'0102		Setzt Blocktransfercounter (Autoincr) auf 258
,-'		Setzt Addresscounter wieder auf A=10
,w'	FF04....	Blocktransfer von 258 words (=516 Bytes) mit Autoincrement

(Zahlenformat in HEX; ,'=ASCII)

Transfargeschwindigkeiten

FPGA/FTDI: FullSpeed (12Mbps):

Write: $\text{Burst}=64\text{Bytes}/(64*500\text{ns}+26\mu\text{s}) = \mathbf{1,1\text{MBps}}$;

Read:

LabVIEW Blocktransfer:

Write: ca. **1MBps** (gemessen);

Read: ca. **500kBps** (gemessen, Blocksize=40000 Bytes);

LabVIEW Program (E,S,I):

63 Hz

LabVIEW Program (I):

63 Hz

FPGA/FX2:HighSpeed (480Mbps):

Write: $\text{Burst}=512\text{Bytes}; 4\text{Bytes}/(4*40\text{ns}+40\text{ns}) = \mathbf{20\text{MBps}}$

Read:

LabVIEW:

Write: ca. **9MBps** (gemessen);

Read: ca. **1,8MBps** (gemessen, Blocksize=80000 Bytes);

LabVIEW Program (E,S,I):

330 Hz

LabVIEW Program (I):

790..2000 Hz

6 CBus

Ein spezielles Modul im FPGA (CONTROLLER) setzt die Interface-Befehle als Master für entsprechende Datentransfers zu den Funktionsmodule um und ist mit allen Modulen über einen **32** Bit Address und **32** Bit Datenbus verbunden.

Verschiedene Kontrollsignal (RD, WR, RDY) steuern den Zugriff für jeden Transfer.

Die zeitliche Abfolge ist im folgenden beschrieben:

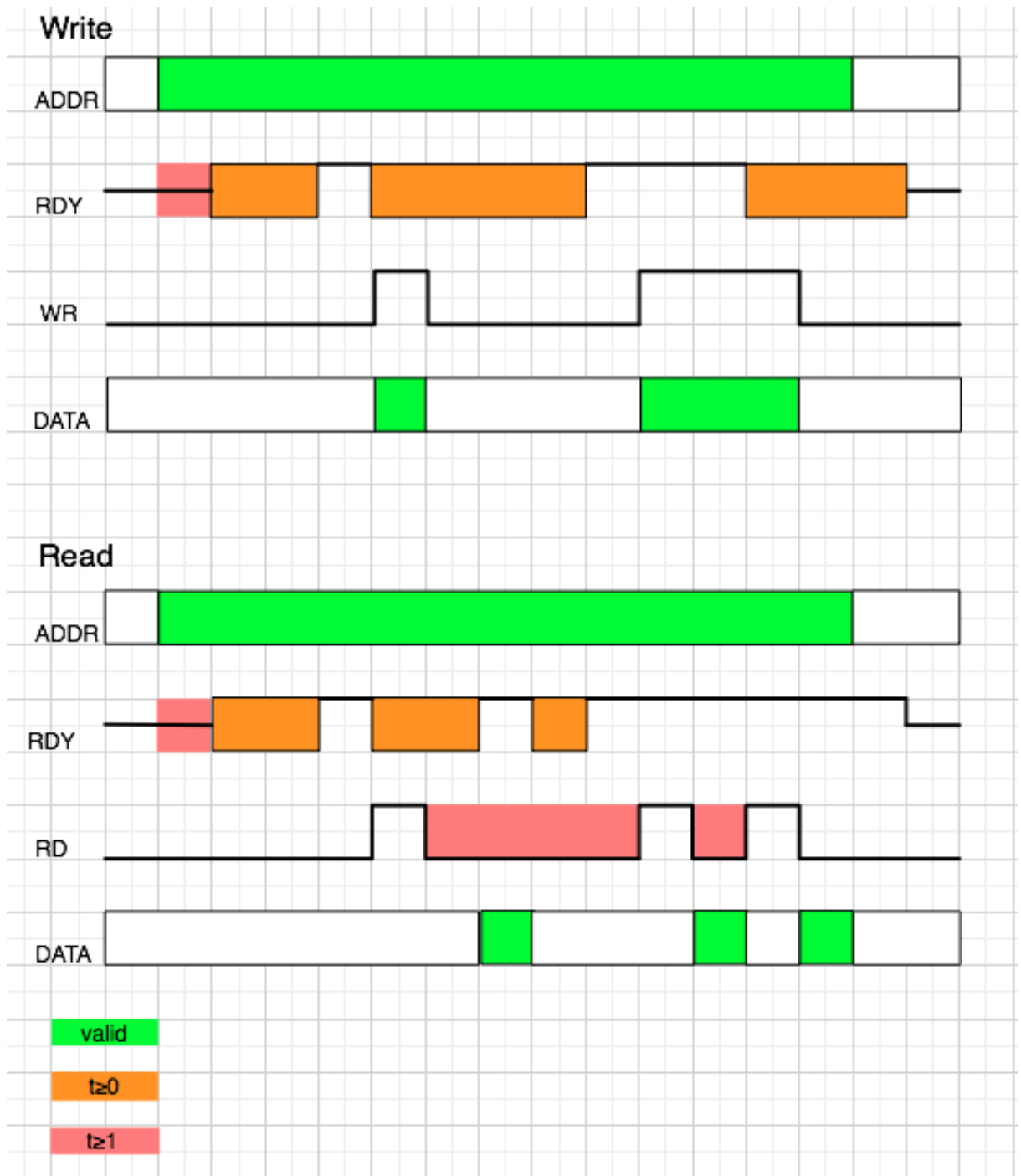
Write

	Master	Slave	Bedeutung
1	Adresse anlegen	Adresse bearbeiten	Adressierung
2		(RDY <= ,0')	(Wait!)
3		RDY <= ,1'	Ready
4	If RDY='1': Daten anlegen WR <= ,1';	RDY <= ,0'/1' Daten einlesen	Wait/Ready Datentransfer ggf. nächster Zyklus (4)
5	WR <= ,0'		Ende Zyklus

Read

	Master	Slave	Bedeutung
1	Adresse anlegen	Adresse bearbeiten	Adressierung
2		(RDY <= ,0')	(Wait!)
3		RDY <= ,1'	Ready
4	If RDY='1': RD <= ,1'	RDY <= ,0'/1'	Wait/Ready
5	RD <= ,0' Daten einlesen	Daten anlegen	ggf. nächster Zyklus (4)

CBus Timing



7 Module

Viele Funktionsmodule benötigen und beziehen sich auf speziell zugeordnete Hardware-Submodule (SU7xx). Insbesondere zur Ein- und Ausgabe müssen alle externen Signale zunächst an die internen FPGA-Pegel angepasst werden bzw. zur Ausgabe entsprechend geformt und getrieben werden. Typische Beispiele sind DIO, DISCRIMINATOR oder ADC/DAC.

Spezielle Hardware kann auch erforderlich sein, wenn bestimmte Verarbeitungen dies erfordern, da diese prinzipiell nicht im FPGA erzeugt werden können oder einen zu hohen Aufwand erfordern würde, z.B. DELAY und RAM.

Andere Module verarbeiten im Prinzip nur die interne Signale und sind deshalb von externer Hardware unabhängig. Diese werden komplett im FPGA realisiert, wie z.B. LOGIC oder GATEGEN.

Eine detaillierte Beschreibung ist in den folgenden Modulbeschreibungen (LogicPool) gegeben.

Adressierung

Alle LogicPool-Module werden über eine 24 Bit Adresse angesprochen:

Adressbits	Wertebereich	Bedeutung
A31..A24		Irrelevant (nur bei Memory verwendet!)
A23..A16	0..255	Typadresse eines LogicPool-Moduls (z.B. „T“, „L“, ...)
A15..A8	0..255	Moduladresse eines LogicPool-Moduls
A7..A0	0..255	Subadresse für beliebige Parameter

Subadresse=0: Jedes Modul muss sich hier mit dem Byte0= 0..254 melden.

Dieser Wert ist gegebenenfalls bereits der Anschlusswert für den ersten Ausgang (s.u). Falls das Modul keinen Signalausgang besitzt, wird der Wert 0 ausgegeben!

Falls ein Modul auf einer Moduladresse nicht existiert, wird immer der Wert &FF (bzw. 255) gelesen! Damit kann ein entsprechendes Programm (z.B. SETUP in LabVIEW) alle Module auf Existenz abfragen und eine Tabelle der vorhandenen Module anlegen.

Folgende Informationen sind bei Langwortauslese in jedem Modul (V4.x) verfügbar:

Byte3=MajVersion , Byte2=SubVersion, Byte1=Model, Byte0=Status&Anschluß

Alle weiteren Modulparameter auf weiteren Subadressen sind modulspezifisch. Die Datenwortbreite ist beliebig zwischen 1 und 4 Bytes.

8 Signale

Single

Typischerweise hat jedes Modul einen oder mehrere Signal-Ausgänge und Signal-Eingänge.

Jeder Ausgang (**OUT_**) hat eine speziellen Anschlussnummer (**1..126**), die bei jedem Modul ausgelesen werden kann. Zusätzlich kann jederzeit der logische Zustand, der an diesem Ausgang anliegt, im obersten Bit 7 abgefragt werden. In der Regel sind alle Ausgänge auf den Schreib-Subadressen 0..n angeordnet.

Jeder Eingang (**IN_**) ist mit einem Multiplexer (Schalter) versehen, der erlaubt einen beliebigen Ausgang auf den Eingang durchzuschalten. Damit kann jede beliebige Verschaltung aller Module realisiert werden. Die Multiplexer werden über Register, entsprechend der gewünschten Verschaltung, auf den gewünschten Anschlusswert gesetzt. In der Regel sind alle Eingänge auf den Schreib-Subadressen 0..n angeordnet.

Beispiel: Der Ausgang des Moduls T10 soll auf den Eingang des Moduls I3 geschaltet werden!

Der Befehl ‚E’’T’0A00’b’ liefert z.B. den Anschlusswert = 02.

Damit verschaltet der Befehl ‚E’’I’0300’B’02 das Signal wie gewünscht.

Spezialbedeutungen:

Durch Setzen des MSB des Multiplexer-Registers kann jeder Eingang zusätzlich invertiert werden, so dass hierfür keine zusätzlichen Module verwendet werden müssen (z.B.: ‚E’’I’0300’B’82).

Der Anschlusswert **0** kennzeichnet einen offenen (nicht verschalteten) Eingang, bzw. Ausgang und wird von den Modulen für spezielle Funktionen oder Verhalten abgefragt.

Der Anschlusswert **127** (128) kennzeichnet einen festen LOW-Pegel.

Der Anschlusswert **255** kennzeichnet einen festen HIGH-Pegel.

Der Anschlusswert **128** kennzeichnet einen kurzen Trigger-Puls, anschließend wird auf 127 gesetzt (zur Zeit noch nicht implementiert!).

Es können zur Zeit 1..126 Module intern verschaltet werden.

BUS

Module können auch einen BUS- Eingang bzw. BUS-Ausgang besitzen. Damit können auch Daten mit bis zu 32Bit zwischen den Modulen weitergereicht werden.

BUS-IOs sind bestimmten Signal-IOs zugeordnet. Die Signale können ganz normal als digitale Signale verwendet werden und sind in diesem Fall typischerweise STROBE-Signale, deren positive Flanke die BUS-Daten für gültig erklären.

Alle BUS-Ausgänge sind immer 32 Bit groß. Nichtrelevante Bits sind auf 0 gesetzt!

Alle BUS-Eingänge sind entsprechend der Funktion des empfangenden Moduls u.U. kleiner als 32 Bit! Die Verschaltung bestimmt, welche Bits übertragen werden!

9 LogicPool

Im folgenden sind alle Funktionsmodule im einzelnen beschrieben. Diese werden bei der Synthese der Firmware, ggf. abhängig von den verwendeten IO-Subkarten, in der gewünschten Anzahl und soweit die Ressourcen des FPGA dies erlauben generiert und bilden jeweils den Funktionsvorrat für jedes Gerät.

9.1 ADC (SU706)

Mit diesem Modul wird der ADC-Baustein auf dem entsprechendem SU706-Modul angesteuert und die digitalisierten Daten über den BUS zur weiteren Verarbeitung ausgegeben. Die Daten können auch direkt ausgelesen werden.

Die effektive Abtastrate (Aufzeichnung) ist durch einen Teiler (Divider=1..typ. 2^{16}) für entsprechend langsamere Raten skalierbar.

Die Datenwandlung wird kontinuierlich mit 100 MHz Abtastrate durchgeführt und über eine frei programmierbare Schwelle wird immer ein entsprechendes Ausgangssignal DISCR (High für $>$ Schwelle) geliefert. Dieses Signal kann folglich auch für eine interne Triggerung bzw. Erzeugung eines Gate-Signals durch Rückführung (eventuell auch negiert!) am Gate-Eingang des gleichen Moduls verwendet werden.

Für die Aufzeichnung gibt es zwei grundsätzliche Modi:

Wave (Mode=0): In diesem Fall werden die Rohdaten des ADC ausgegeben.

Samples=0: Sampling über die Dauer von GATE

Samples=1: Sample (unabh. von Divider) bei jeder steigenden Flanke von GATE

Samples=n: nach Trigger (steigenden Flanke von GATE) werden n Samples ausgegeben.

Presamples=0: Jeder Trigger erzeugt sofort n Samples.

Presamples>0: Das Presampling beginnt nach Laden von PreSamples.

Erst nach der gewünschte Anzahl von PreSamples wird der Trigger freigegeben.

Parameter (Mode<>0): In diesem Modus werden bestimmte Parameter (siehe Bild 2) aus der Waveform ermittelt und nur diese für jedes einzelne Gate-Signal in wählbarer Kombination nacheinander ausgegeben:

GateWidth: Die Dauer des Gates in Abtastwerten ($10 \text{ ns} * \text{Divider}$)

GateSum: Summe aller ADC-Werte innerhalb des Gates (max16Bit: GAIN!)

GateMin: Minimaler ADC-Wert innerhalb des Gates

GateMinPos: Position des Minimums in Abtastwerten nach Beginn des Gates

GateMax: Maximaler ADC-Wert innerhalb des Gates

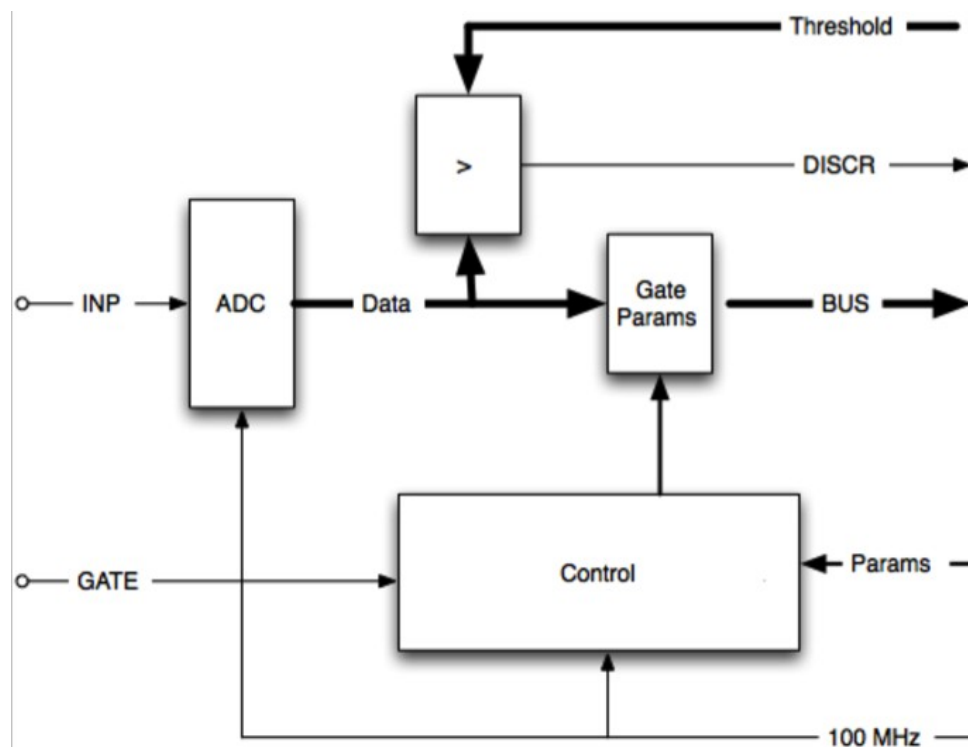
GateMinPos: Position des Maximums in Abtastwerten nach Beginn des Gates
 Min: Minimaler ADC-Wert ausserhalb des Gates (Baseline)
 Max: Maximaler ADC-Wert ausserhalb des Gates

Für beide Modis gilt, dass typischerweise nach dem Start gewartet wird bis die gewünschte Anzahl von Werten im FIFO steht und diese dann im Blocktransfer ausgelesen werden. Anschließend wird das Modul zurückgesetzt und der Aufzeichnungsvorgang neu gestartet.

Ansonsten werden entsprechend dem Auslesen des FIFOs einfach Daten nachgeschoben.

Falls das FIFO dabei noch voll ist, wird das Flag OVFL gesetzt. Dies tritt typischerweise auf, wenn die Schreibrate > Leserate ist.

Blockschaltbild



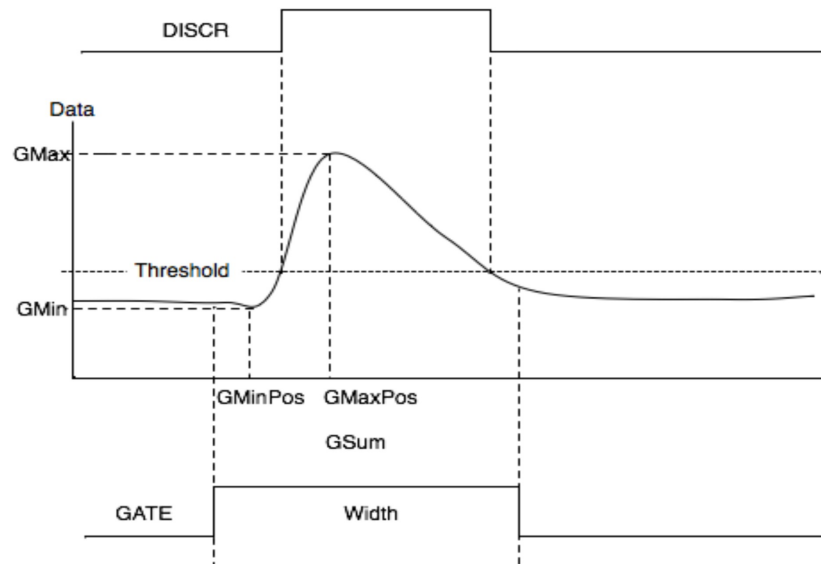
Der **ADC** Baustein sampelt und konvertiert kontinuierlich den analogen Eingang **Inp** mit der Systemclock von **100 MHz**. Die 14 Bit Daten **Data** werden dabei in einem Vergleich (>) mit der **Threshold** verglichen und daraus ein Ausgangssignal **DISCR** abgeleitet.

Der Dual-Port Speicherbaustein **FIFO** speichert die Daten entweder direkt oder nur die entsprechenden Signalparameter, die in der Einheit **GateParams** ermittelt werden. Das FIFO kann auch als Histogrammer mit Speicher betrieben werden welcher die Häufigkeiten nach Datenwerten sortiert aufsummiert.

Das Einschreiben der Daten kann abhängig von einem **Divider** entsprechend auch mit einer 100 MHz/Divider langsameren Frequenz erfolgen und so langsamere Abtastraten realisiert werden.

Die Einheit **TriggerCntrl** steuert den Ablauf und das Einschreiben der Daten entsprechend **PreSamples** (nach Start) und dem anliegenden **GATE**.

Gate Parameter



Das Bild zeigt die ermittelten Parameter abhängig von einem typischen Eingangspuls und einem entsprechendem GATE-Signal. Gegebenenfalls kann das Ausgangssignal DISCR aus dem ADC-Modul direkt durch Rückführung auf das GATE (interner Trigger mit Threshold) verwendet werden!

GSum ist die Summe (Integral) aller ADC-Werte während der GATE-Dauer.

Zusätzlich wird jeweils der minimale und maximale ADC-Wert vor dem aktuellen GATE ermittelt (nicht gezeichnet!).

Memory Map

,A'	Read	Write
0	OUT_BUS	IN_GATE
1	OUT_DISCR	Samples(W)
2	DATA(W,L)	Divider(W)
3	Running(0)	MinWidth(W)&Threshold(W)
4		Mode(0=Wave) & Start (0) GWidth (1) GSUM* (2) GMin (3) GMinPos (4) Gmax (5) GmaxPos (6) Min (7) Max
5		Offset(W)
6		PreSamples(W) & Start

OUT_BUS: Anschlusswert für Ausgang BUS

OUT_DISCR: Anschlusswert für Ausgang DISCR

DATA: aktuelle Daten

Running: Modul zeichnet Daten auf

IN_GATE: setzt Eingang GATE

Samples: Anzahl der Daten nach Trigger

Divider: Einstellung der Abtastfrequenz

MinWidth: Minimale Pulsbreite zur Verarbeitung

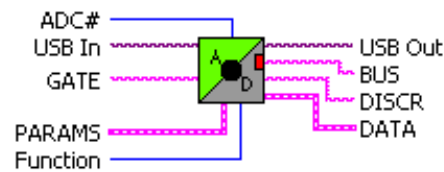
Threshold: Schwelle für Puls zur Verarbeitung

Mode: Auswahl des Aufzeichnungs- bzw. Verarbeitungsmodus (siehe oben)

Offset: Verschiebung der ADC-Daten in der Amplitude

Presamples: Anzahl der Samples vor Trigger

LabVIEW Interface



ADC.vi

14 Bit ADC Module with max. 100 MHz sampling rate for recording of raw data and pulse parameters.
Data will be output on BUS or can be read independently.
Output "DISCR" goes true for ADC data \geq "Threshold" (sampling rate=100MHz).
Support: SU706

Modul can be run in two Modes:

1. MODE=0 (all OFF): Raw ADC Data will be recorded according DIVIDER
 SAMPLES=0: Data recording as long as GATE=High
 SAMPLES=1: Every rising edge of GATE produces one Data Sample
 SAMPLES=n: Rising edge of GATE triggers recording of n Samples (after Presamples!)
2. MODE \neq 0: each bit determines which parameter will be recorded on every GATE=High
 (only when GWidth \geq MinWidth)

PARAMS:

Divider: actual sampling rate $f=100 \text{ MHz}/\text{Divider}$
 PreSamples: starts presampling for given number, then trigger is valid
 Samples: number of recordings (see Mode)
 Threshold: determines output DISCR.
 MinWidth: only record Pulsparams with GWidth \geq MinWidth
 -Offset: subtract from recorded values (does not influence Threshold!)
 Mode: record parameters (selecting none will record raw ADC data!)
 "GWidth": duration of "GATE" (in sampling steps)
 "GSum": Sum of ADC values over duration of "GATE" (32 Bit raw)
 "GMin": Minimum value within "GATE"
 "GMinPos": Position of "GMin" after start of "GATE"
 "GMax": Maximum value within "GATE"
 "GMaxPos": Position of "GMax" after start of "GATE"
 "Min": Minimum value before "GATE"
 "Max": Maximum value before "GATE"

DATA:

Running: true if recording (module stops after n Samples).
 Data: 32b

Function:

Connect: connects in&outputs and loads all parameters.
 Set GATE: set input
 Get BUS: get output
 Get DISCR: get output
 Write Params&Clear: loads parameters & clears pulse parameters
 Read Data: return last ADC value and RUNNING
 Write Presamples & Start: starts presampling

ADC#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.2 ADC4 (SU728)

Das Modul besitzt einen **4 Kanal, 16 Bit** Analog-Digital-Wandler für Signale mit einer Wandelzeit von **1 us** pro Kanal wobei ein Eingangsbereich von **-5..+5 Volt** erfasst wird.

Für eine extern getriggerte Anwendung löst ein Signal (TRIG) mit steigender Flanke den Kovertierungsprozess aus, in welchem nach der Konvertierungszeit die Daten am Ausgangsbus (BUS) erscheinen und z.B. in einem Memory-Modul (FIFO) abgespeichert werden können.

Durch Laden der Kanalnummer wird ebenfalls die Konvertierung gestartet und der Datenwert (16 Bit) kann auch einfach ausgelesen werden.

Memory Map

,k'	Read	Write
0	OUT_BUS	IN_TRIGGER
1	Data(W)	Channel(B) (0..3)

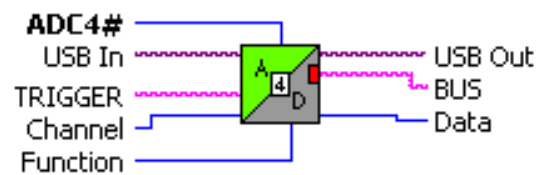
OUT_BUS: Ausgangssignal für Datenbus

Data: ADC Wert

IN_TRIG: Externer Start der Konvertierung

Channel: Kanal & Start Konvertierung

LabVIEW Interface



ADC4.vi

16 Bit ADC with 4 selectable channels.
 Support: SU728.
 Conversion triggered by TRIGGER (max 1MHz) and setting a channel.
 Data available at TRIGGER from previous sampling
 Voltage range: -5V.. 5V

I/O:

TRIGGER: rising edge starts conversion and readout
 BUS: Bus Strobe for 16b Databus with TRIGGER

Parameter/Data:

Channel: 0..3
 Data: ADC data of last conversion

Function:

Connect: connect and initialize all parameters
 Set TRIGGER: set state/connection
 Get BUS: get state
 Read ADC: get data
 Write Channel: set input channel & Convert

ADC4B#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.3 ADC8 (SU702)

Das Modul SU702 besitzt einen **8 Kanal, 14 Bit** Analog-Digital-Wandler für langsamere Signale. Die Wandelzeit pro Kanal beträgt etwa **5 us**.

Alle Kanäle sind **differentiell** mit eigenen Anpassungsverstärkern ausgeführt und besitzen einen bipolaren Eingangsbereich von **-4..+4 Volt**.

Durch Laden des entsprechenden Cntrl-Bytes (siehe Datenblatt MAX1149) wird die Konfiguration, der gewünschte Kanal und damit die Konvertierung gestartet. Anschließend kann der Datenwert (14 Bit) ausgelesen werden.

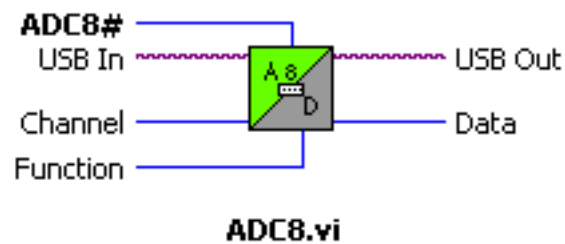
Memory Map

,W'	Read	Write
0	0	Cntrl Ch1-8
1	Data(W)	

Data: Datenwert

Cntrl Ch1..8: Kontrollwert zur Auswahl und Start der Konvertierung

LabVIEW Interface



14 Bit ADC with 8 channels
Support: SU702
Conversion triggered by calling.
Conversion time approx. 5 us.
Voltage range: -4V.. 4V

Parameter:
Channel: 0..7

Functions:
Connect: read version
Convert & Read Date: start conversion and get data

ADC8#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

9.4 ADC16 (SU712, SU720)

Das Modul besitzt einen **16 Kanal, 14 Bit** Analog-Digital-Wandler für langsamere Signale. Die Wandelzeit pro Kanal beträgt etwa **5 us** wobei ein Eingangsbereich von **0..2,5 Volt** erfasst wird. Jeweils 2 Kanäle können auch **differentiell** beschaltet werden.

Durch Laden des entsprechenden Cntrl-Bytes (siehe Datenblatt MAX1149) wird die Konfiguration, der gewünschte Kanal und damit die Konversion gestartet und anschließend kann der Datenwert (14 Bit) ausgelesen werden. Busy wird beim Start gesetzt und geht nach ca. 6us auf Low wenn die Daten gültig sind.

Für eine extern getriggerte Anwendung löst ein Signal (TRIG) mit steigender Flanke den Kovertierungsprozess aus, in welchem nach der Konvertierungszeit die Daten sowie der aktuelle Kanal am Ausgangsbus (BUS) erscheinen und z.B. in einem Memory-Modul (FIFO) abgespeichert werden können.

Der Parameter **MaxChannel** bestimmt welche Kanäle nacheinander konvertiert werden.

MaxChannel=0: nur Kanal 0 wird konvertiert!

MaxChannel=n: Kanal 0 bis n werden nacheinander konvertiert.

Memory Map

,K'	Read	Write
0	OUT_BUS	IN_TRIG
1	Busy(15), Data(W)	Cntrl Ch1-8
		Cntrl Ch 9-16
		MaxChannel

OUT_BUS: Channel(19..16) & „00“ & ADC(13..0)

Busy: High bei Konvertierung

Data: Datenwert

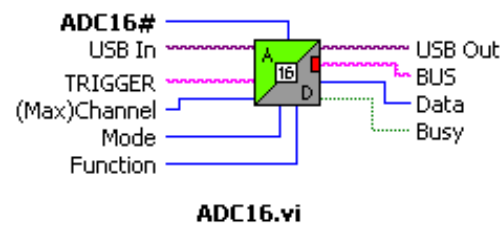
IN_TRIG: Externer Start der Konvertierung

Cntrl Ch1..8: Kontrollwert für Kanal 1..8

Cntrl Ch 9..16: Kontrollwert für Kanal 9..16

MaxChannel: Kanalnummer für sequentielle Konvertierung mehrerer Kanäle

LabVIEW Interface



16 channel 14 Bit ADC
 Support: SU712, SU720
 Conversion triggered by calling or TRIGGER (after 1,76us)
 Conversion time approx. 6 us.
 Voltage range: 0.. 2,5V

Inputs:

TRIGGER: rising edge triggers data conversion & advances to next channel until maxChannel

Outputs:

BUS: Strobe for 14 b Data & Address
 Data: 14b
 Busy: if converting

Functions:

Connect: set maxChannel for conversion on TRIGGER
 Set TRIGGER: set state
 Get BUS: get Strobe state
 Convert & Read ADC: trigger conversion according CHANNEL and read data when not Busy
 Write maxChannel: set maxChannel for conversion on TRIGGER
 Read ADC: read after TRIGGER conversion & Busy

ADC16B#: number of module (must be unique)

Mode:

Single: 0, 1, 2, ..
 Differential: 0-1, 2-3, ..
 Unipolar: GND..+2.5V
 Bipolar: COM .. +2.5V (COM can be used as reference on SU712!)

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.5 ADC16F (SU712, SU720)

Das Modul besitzt einen **16 Kanal, 14 Bit** Analog-Digital-Wandler mit einer Wandelzeit von etwa **5 us** /Kanal wobei ein Eingangsbereich von **0..2,5 Volt** erfasst wird. Die Funktionsweise im weiteren entspricht einem 16 Kanal U/f-Konverter.

Durch den Parameter **MaxChannel** kann die Sequenz der Wandlung (Kanal 0..MaxChannel) eingestellt werden. Die Dauer der Sequenz ist durch den Parameter **Divider** (*10ns) pro Kanal bestimmt.

Alle ADC-Werte können jederzeit beliebig ausgelesen werden. Gleichzeitig werden alle ADC-Werte in entsprechende Signal-Pulse mit einer Frequenz analog der Spannungen generiert und ausgegeben.

Die Erzeugung der Frequenzen wird über eine Tabelle (RAM) gesteuert. Dabei werden die 10 höherwertigen Bit über die Tabelle in einen 24 Bit (Period) Wert transformiert. Die jeweilige Ausgangsfrequenz ergibt sich nach der Formel $f=50\text{MHz}/(\text{Period}+1)!$

Memory Map

,v'	Read	Write
0	OUT_N1	Divider(L)
1		MaxChannel (0..15)
2		RAM+ (T) (Period-1)
3		RAMAddr(10b)
..		
15	OUT_N16	
16	ADC_1(W)	
..		
31	ADC_16(W)	

OUT_N1..OUT_N16: Signalausgänge für Frequenz(ADC)

ADC_1..ADC16: ADC Daten

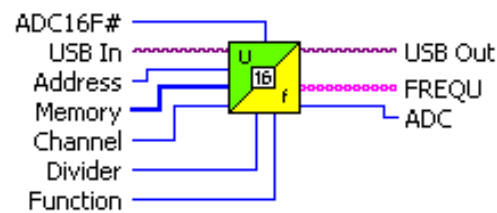
Divider: bestimmt Konvertierungsrate (= 10ns*Divider)

MaxChannel: Kanalnummer für sequentielle Konvertierung mehrerer Kanäle

RAM+: Memorydaten (autoincr) für ADC/Perioden Konvertierung

RAMAddr: Startadresse

LabVIEW Interface



ADC16F.vi

16 Channel Voltage to Frequency (U/f) Converter

All 16 Inputs from a related SU712, 720 will be converted sequentially (1..Channel) with a given time ($10\text{ns} \cdot \text{Divider}$). The 14bit values will be collected in registers which can be read arbitrary in ADC(Channel) and will be used to generate a frequency at the corresponding outputs FREQU (0..15).

An intermediate Memory converts each ADC-value with the 10 significant bits to a 24bit period time which is used to generate the frequency ($f = 50\text{MHz} / (\text{Period} + 1)$).

ADC16F#: number of module (must be unique)

Function:

Connect: connects all outputs and sets all parameters

Get ADC (Channel): return ADC voltage from Channel

Write Divider: set sequencer time ($> 600 = 6\mu\text{s}$)

Write maxChannel (Channel): set maximum channel for conversion

Write Memory: Fill memory with auto increment for ADC/Period conversion

Write Address: set starting address

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.6 ADCF (SU706)

Mit diesem Modul wird der ADC-Baustein auf dem entsprechendem SU706-Modul angesteuert und die digitalisierten Daten in einen FIFO-Speicherbereich (typ. 1024 Werte) auf dem FPGA übernommen.

Alternativ ist auch die Aufzeichnung der gewählten Daten in einem Histogramm (0..1023 * 16Bit), das jederzeit ohne Totzeit ausgelesen werden kann, möglich. In diesem Fall ist darauf zu achten, dass durch den Skalierungsfaktor $GAIN = 2^{(-n)}$ (binärer Faktor!) die Werte soweit herunterskaliert werden, dass sie in das Histogramm (0..1023) passen!

Werte können nur histogrammiert werden, wenn die Rate < 50 MHz ist! Die effektive Abtastrate (Aufzeichnung) ist durch einen Teiler (Divider=1..typ. 2^{**16}) für entsprechend langsamere Raten skalierbar.

Die Datenwandlung wird kontinuierlich mit 100 MHz Abtastrate durchgeführt und über eine frei programmierbare Schwelle wird immer ein entsprechendes Ausgangssignal DISCR (High für > Schwelle) geliefert. Dieses Signal kann folglich auch für eine interne Triggerung bzw. Erzeugung eines Gate-Signals durch Rückführung (eventuell auch negiert!) am Gate-Eingang des gleichen Moduls verwendet werden.

Für die Aufzeichnung gibt es zwei grundsätzliche Modi:

Wave (Mode=0): In diesem Fall werden die Rohdaten direkt aus dem ADC mit der entsprechenden Abtastrate abgespeichert. Hierbei wird aus der steigenden Flanke des Gate-Signals der Trigger für die Aufzeichnung abgeleitet. Die Aufzeichnung beginnt bereits mit dem Start für die gewünschte Anzahl von PreSamples und wird dann mit dem Trigger bis zur vollständigen Füllung des FIFOs beendet. Im Memory steht dann sowohl die Vorgeschichte (PreSamples) wie die ADC-Werte nach dem Trigger (PostSamples). Der Auslesevorgang kann nach dem Triggerereignis gestartet werden.

Parameter (Mode<>0): In diesem Modus werden bestimmte Parameter (siehe Bild 2) aus der Waveform ermittelt und nur diese für jedes einzelne Gate-Signal in wählbarer Kombination nacheinander in das FIFO-Memory abgespeichert:

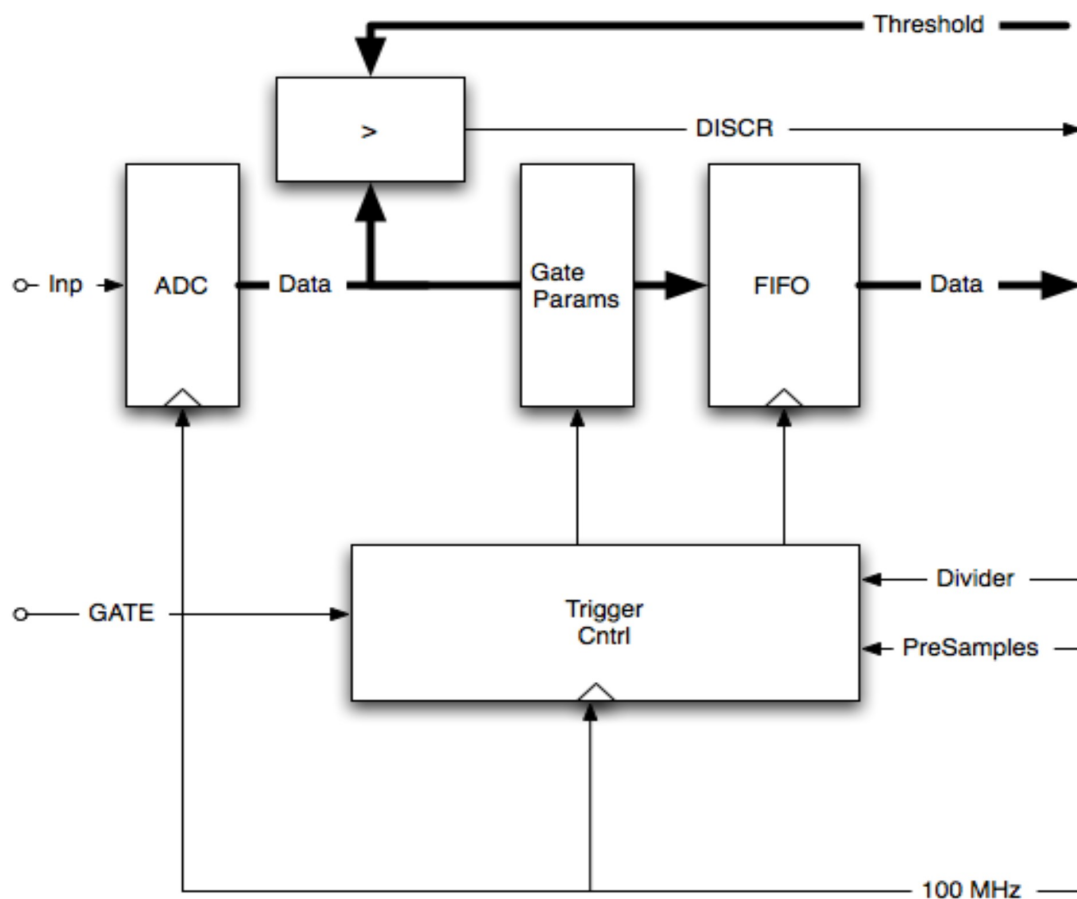
- GateWidth: Die Dauer des Gates in Abtastwerten ($10 \text{ ns} * \text{Divider}$)
- GateSum: Summe aller ADC-Werte innerhalb des Gates (max16Bit: GAIN!)
- GateMin: Minimaler ADC-Wert innerhalb des Gates
- GateMinPos: Position des Minimums in Abtastwerten nach Beginn des Gates
- GateMax: Maximaler ADC-Wert innerhalb des Gates
- GateMaxPos: Position des Maximums in Abtastwerten nach Beginn des Gates
- Min: Minimaler ADC-Wert ausserhalb des Gates (Baseline)
- Max: Maximaler ADC-Wert ausserhalb des Gates

Der Auslesevorgang kann sofort nach dem Start erfolgen.

Für beide Modis gilt, dass typischerweise nach dem Start gewartet wird bis die gewünschte Anzahl von Werten im FIFO steht und diese dann im Blocktransfer ausgelesen werden. Anschließend wird das Modul zurückgesetzt und der Aufzeichnungsvorgang neu gestartet.

Ansonsten werden entsprechend dem Auslesen des FIFOs einfach Daten nachgeschoben. Falls das FIFO dabei noch voll ist, wird das Flag OVFL gesetzt. Dies tritt typischerweise auf, wenn die Schreibrate > Leserate ist.

Blockschaltbild



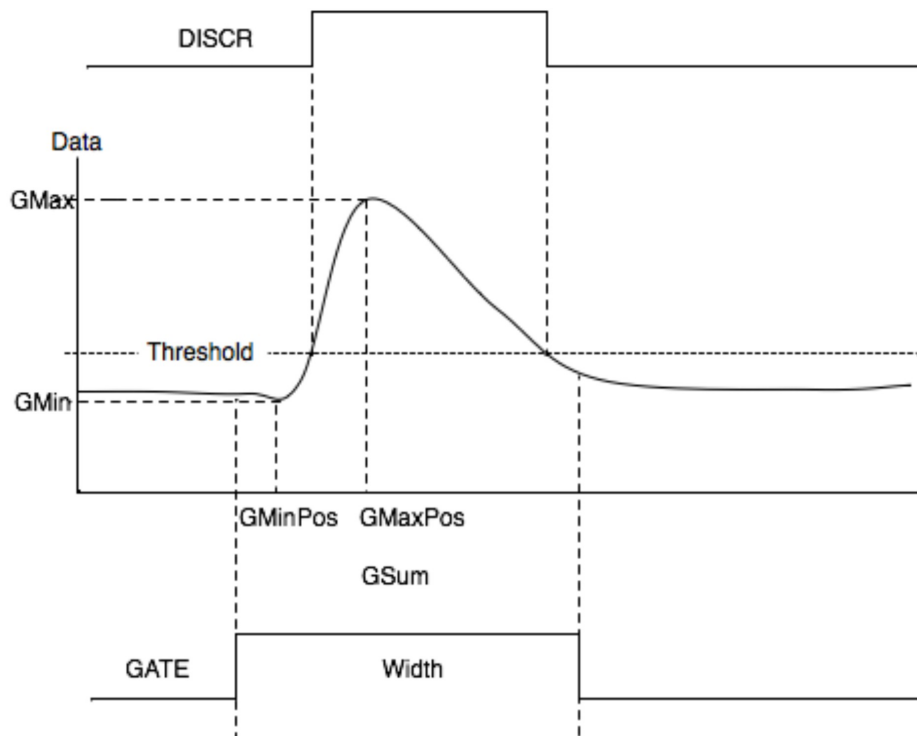
Der **ADC** Baustein sampelt und konvertiert kontinuierlich den analogen Eingang **Inp** mit der Systemclock von **100 MHz**. Die 14 Bit Daten **Data** werden dabei in einem Vergleich (>) mit der **Threshold** verglichen und daraus ein Ausgangssignal **DISCR** abgeleitet.

Der Dual-Port Speicherbaustein **FIFO** speichert die Daten entweder direkt oder nur die entsprechenden Signalparameter, die in der Einheit **GateParams** ermittelt werden. Das FIFO kann auch als Histogrammer mit Speicher betrieben werden welcher die Häufigkeiten nach Datenwerten sortiert aufsummiert.

Das Einschreiben der Daten kann abhängig von einem **Divider** entsprechend auch mit einer $100 \text{ MHz}/\text{Divider}$ langsameren Frequenz bzw. Abtastrate realisiert werden.

Die Einheit **TriggerCntrl** steuert den Ablauf und das Einschreiben der Daten entsprechend **PreSamples** (nach Start) und dem anliegenden **GATE**.

Gate-Parameter



Das Bild zeigt die ermittelten Parameter abhängig von einem typischen Eingangspuls und einem entsprechendem GATE-Signal. Gegebenenfalls kann das Ausgangssignal DISCR aus dem ADC-Modul direkt durch Rückführung auf das GATE (interner Trigger mit Threshold) verwendet werden!

GSum ist die Summe (Integral) aller ADC-Werte während der GATE-Dauer.

Zusätzlich wird jeweils der minimale und maximale ADC-Wert vor dem aktuellen GATE ermittelt (nicht gezeichnet!).

Memory Map

,a'	Read	Write
0	OUT_DISCR	IN_GATE
1		IN_WRITE
2	FIFO(W,L)	FIFO(W,L)
3	OVFL(15)&Run(14)&Count(10..0)	MinWidth(W)&Threshold(W)
4		PreSamples(9..0) & Clear
5		Mode(0=Wave) & Start (8) GWidth (9) GSUM* (10) GMin (11) GMinPos (12) Gmax (13) GmaxPos (14) Min (15) Max
6		Histo(7)& Gain(6..0)
7		Offset(W)
8		Divider(W)

OUT_DISCR: Anschlusswert für Ausgang DISCR

FIFO: lesen und schreiben der Daten

OVFL: FIFO voll, bzw. Daten zu groß für Histogramm

Run: Modul zeichnet Daten auf

Count: 0..1024, Füllgrad des FIFO

IN_GATE: setzt Eingang GATE

IN_WRITE: setzt Eingang WRITE für verzögertes Einschreiben. Wenn Offen, werden Daten sofort eingeschrieben!

MinWidth: Minimale Pulsbreite zur Verarbeitung

Threshold: Schwelle für Puls zur Verarbeitung

Presamples: Anzahl der Samples vor Trigger

Mode: Auswahl des Aufzeichnungs- bzw. Verarbeitungsmodus (siehe oben)

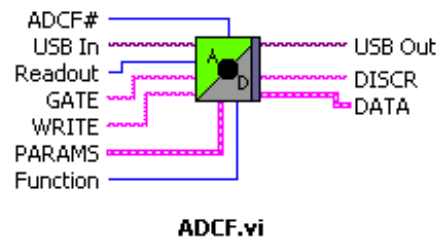
Histo: Aufzeichnung Histogramm oder FIFO

Gain: Verschiebung zur Aufzeichnung um entsprechende Stellen nach rechts!

Offset: Verschiebung der ADC-Daten in der Amplitude

Divider: Einstellung der Abtastfrequenz

LabVIEW Interface



14 Bit ADC Module with max. 100 MHz sampling rate for recording/histogramming of raw data and parameters. A special WRITE input allows delayed recording.

Support: SU706

Output "DISCR" goes true for ADC data \geq "Threshold" (sampling rate=100MHz).

Modul can be run in two Modes:

1. "Mode"=all OFF: Rising edge of "GATE" triggers recording of raw ADC data with "PreSamples" in "FIFO" (typ. 1024 words)
2. "Mode"=ANY: each value determines which parameter will be recorded on every "GATE" into FIFO, only when $GWidth \geq MinWidth$ (see Mode)

PARAMS:

Divider: actual sampling rate $f=100 \text{ MHz}/\text{Divider}$

Threshold: determines output DISCR.

MinWidth: only record Pulsparams with $GWidth \geq MinWidth$

PreTrigger: number of recordings before trigger.

Readout: number of data on readout.

Gain $2^{\wedge}n$: reducing gain factor (use in Histo mode to scale down to 0..1024!)

-Offset: subtract from recorded values (does not influence Threshold!)

FIFO/HISTO: recording mode

Mode: record parameters (selecting none will record raw ADC data on trigger!)

"GWidth": duration of "GATE" (in sampling steps)

"GSum": Sum of ADC values over duration of "GATE" (32 Bit raw)

"GMin": Minimum value within "GATE"

"GMinPos": Position of "GMin" after start of "GATE"

"GMax": Maximum value within "GATE"

"GMaxPos": Position of "GMax" after start of "GATE"

"Min": Minimum value before "GATE"

"Max": Maximum value before "GATE"

DATA:

Running: true if recording (module stops only on "CLEAR").

Overflow: FIFO: true if memory full; HISTO: data >1024

Count: FIFO: number of data in memory; HISTO: 1024

Data: array of raw data, gate params or histogram

Function:

Connect: connects in&outputs and loads all parameters.

Set GATE: set input

Set WRITE: if not connected will store data immediately into memory

Get DISCR: get output

Write Params&Clear: loads parameters, clears FIFO and starts Recording

Read Status: return Running & COUNT.

Read Data: return array of FIFO.

ADCH#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.7 AMPNF (SU715)

In Verbindung mit der Hardware SU715 werden zwei NF-Verstärkerkanäle mit programmierbarer Verstärkung realisiert.

Die Verstärkung ist im Bereich von ca. 1 (Gain=255)..100 (Gain=1) einstellbar.

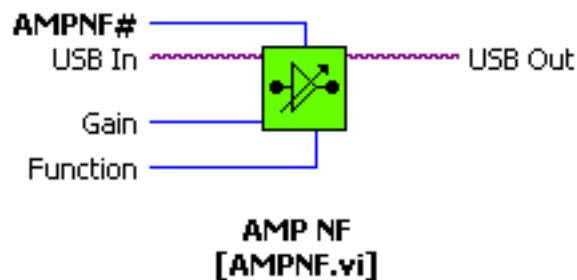
Memory Map

,V'	Read	Write
0	0	Channel1(8) & Gain

Channel1: Selektion von Kanal 0/1

Gain: einstellbare Verstärkung

LabVIEW Interface



NF-Amplifier with programmable gain
 Support: SU715 (2 Channels)
 Gain range (Gain=0..255): 101,5 .. 0,7
 Voltage range: $V_{pp}=10V$

AMPNF#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.8 BUSMONITOR

Bestimmte Module im LogicPool übertragen Daten über BUS-Verbindungen. Dieses Modul dient im wesentlichen zur Überwachung und zum Testen.

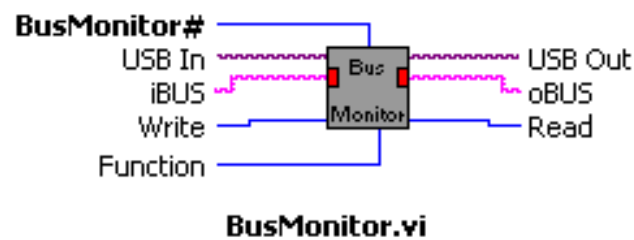
Daten, die von einem Modul gesendet und am Eingang **IN_BUS** empfangen werden, werden in einem Lese-Register **DATA** gespeichert und können von dort gelesen werden.

Daten, die in das Schreib-Register **DATA** geschrieben werden, werden am Ausgang **OUT_BUS** an ein weiteres Modul gesendet.

Memory Map

,x'	Read	Write
0	OUT_BUS	IN_BUS
1	Data (L)	Data (L)

LabVIEW Interface



Bus Monitor
allows access to internal Bus structure.

INPUTS:

iBUS: input Bus strobe

OUTPUTS:

oBUS: output Bus strobe

PARAMS:

Address: address according range

Addr/Data: according range

Function:

Connect: connect module

Set iBUS: set input strobe

Get oBUS: get state of strobe

Write Address: Write data to oBUS

Read Data: Read data from iBUS

BusMonitor#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi

9.9 COINCCOUNTER

In vielen kernphysikalischen Experimenten muss als Trigger die Koinzidenz von mehreren Detektorsignalen bestimmt und ggf. gezählt werden!

Dieses Modul hat eine bestimmte Anzahl von Eingängen und einen Ausgang AND, der die entsprechende Koinzidenz durch ein logisches Signal anzeigt. Dieses Signal kann als Trigger verwendet werden und wird gleichzeitig in einem Zähler (rising_edge) gezählt.

Eingänge, die nicht beschaltet werden (open), werden dabei automatisch nicht berücksichtigt und damit die Multiplizität entsprechend festgelegt.

Memory Map

,N'	Read	Write
0	OUT_AND	IN_Input 1
..	Counter(L)	...
n-1		IN_Input n
8		Clear Counter

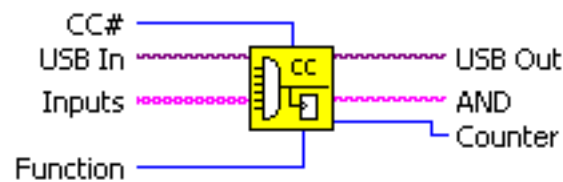
OUT_AND: anschlusswert für Ausgang AND

Counter: Zählerwert

IN_Input1..n: setzt Eingänge 1..n

Clear Counter: löscht Zähler

LabVIEW Interface



COINCCOUNTER.vi

Coincidence module with 8 inputs (A,B,C,D,E,F,G,H) , one AND output and counter.
If all connected (!) inputs go high, the AND output is high and the rising edge will be counted.

V0: 8 inputs, 32bit counter

V1: 4 inputs, 24bit counter

Function:

Connect: connects all in/outputs and sets all parameters

Set A: change input A

...

Set H: change input H

Get AND: return state of AND

Read Counter: get counter

Clear Counter: set counter=0

CC#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.10 COUNT

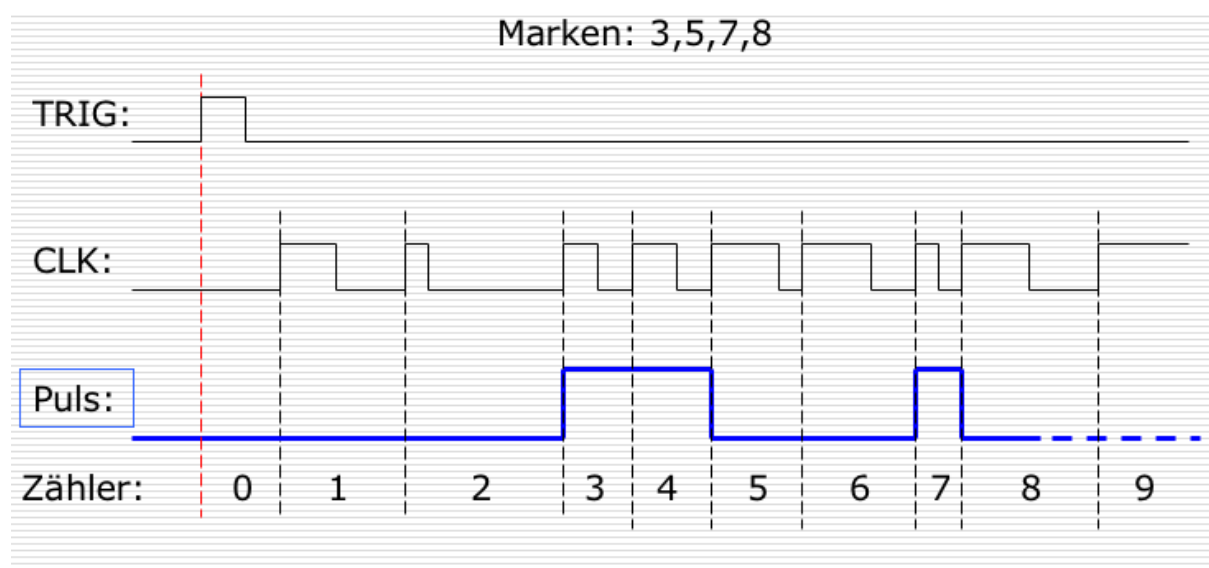
ACHTUNG: Dieses Modul wird ab LP V4.0 nicht mehr unterstützt! Alternativ kann das Modul GATEGEN verwendet werden!

Dieses Modul realisiert verschiedene Funktionen, die sich im wesentlichen durch das Abzählen eines externen oder internen Signals ergeben. Dabei wird abhängig von programmierten Pulsmarken ein beliebiger Ausgangspuls erzeugt. Der Zähler und die Pulsmarken sind typischerweise in 32 Bit ausgelegt.

Der Parameter Mode legt das Verhalten des Moduls und die Bedeutung der Eingänge fest:

- 0: COUNTER
- 1: PULSER
- 2: ASYNC PULSER

Im Modus PULSER wird der Zählvorgang durch den Eingang TRIG (Eingang A) gestartet. Gezählt werden die Signale an CLK (Eingang B). Jedes Erreichen einer Pulsmarke ändert den logischen Zustand von PULS (siehe folgende Zeichnung):



Besonderheiten:

Falls am Eingang B, CLK kein Signal angeschlossen ist (MUX_B=0), wird automatisch die interne Systemclock von 100 MHz verwendet!

Falls der Eingang A, TRIG nicht angeschlossen ist (MUX_A=0), wird der Zähler unmittelbar gestartet und die programmierte Pulsfolge immer automatisch bei Zähler 0 wiederholt. Damit kann eine freilaufende Clock mit komplexer Pulsfolge realisiert werden.

In diesem Fall kennzeichnet nur eine Marke (beliebig) die maximale Frequenz (Systemclock=100MHz).

Im Modus COUNTER sind die beiden Eingänge A und B gleichwertig und in einer AND

Funktion verknüpft. Damit kann ein Eingang als Gate für den anderen Eingang verwendet werden. Gezählt werden dann die resultierenden steigenden Flanken von A AND B. Der Zähler kann jederzeit ausgelesen werden. Abhängig von den eingespeicherten Pulsmarken schaltet auch hier der Ausgang um.

Memory Map

,C'	Read	Write
0	OUT_PULS	IN_A
1	Counter(L)	IN_B
2		Mode, Clear Counter 0 Counter 1 Pulser 2 Async Pulser (P(1)=-1)
3		Pulsmarken, Clear Counter(L)
..n		Pulsmarken, Clear Counter(L)

OUT_PULS: Status(7) & Anschlußnummer für Ausgang PULS

Counter: Zählerwert

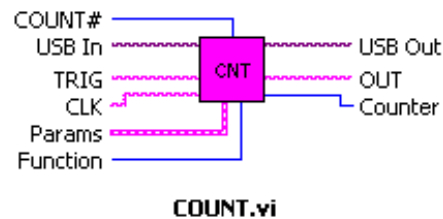
IN_A: setzt Eingang A

IN_B: setzt Eingang B

Mode: Funktionsmodus (s.o.)

Pulsmarken: entsprechend Firmware (typ. 4)

LabVIEW Interface



Basic Counter module with 2 inputs (TRIG, CLK) and 1 output (OUT).
Loaded Parameters determine the specific behaviour:
(see VIs CLOCK, COUNTER, PULSER, SCALER for detailed description!)

COUNTER:

Counting without limit.

Both inputs are combined via an AND function into one counting signal (rising edge). Therefore one can be used as a GATE input for the other.

PULSER:

Last Marker stops and resets counter=0.

TRIG: rising edge at TRIG starts counting. If not connected (open), counting restarts automatically at last marker (Clock mode).

CLK: external Clock, which is counted. If not connected (open), internal Clock (100 MHz) is used.

PULSER ASYNC:

If first marker=0 then Puls starts immediately with TRIG.

Params

Mode: Counter;/Pulser;/Pulser Async

Marker: every Marker=counter toggles output OUT.

Function:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Set CLK: change input CLK

Get OUT: return output OUT

Write Params: load params & clear counter

Read Counter: read current counter

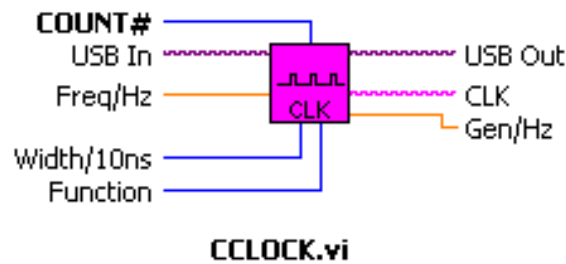
Write Mode & Clear: clear counter

COUNT#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

LabVIEW Interface



CLOCK module for generating free running clocks.

All frequencies on output CLK are derived from the system clock (100 MHz) by a divider (typ. 32 Bit).

Freq/Hz: input of frequency in Hz

High/ns: specifies length of High state in 10 ns (=0 generates 50% duty cycle!)

Gen/Hz: output of actual generated frequency in Hz

Function:

Connect: connects all in/outputs and sets all parameters

Get CLK: return output state of CLK

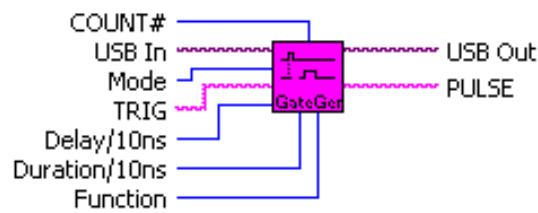
Write Frequency: load frequency and High parameter

COUNT#: number of module (must be unique in group COUNT)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

LabVIEW Interface



Gate Generator [CGATEGEN.vi]

Simple Gate- (Pulse-) Generator with Delay (32 bit) and Duration (32 bit) after trigger.

Inputs:

TRIG: starts single pulse with rising edge

Outputs:

PULSE: pulse after trigger with Delay and Duration

Params:

Mode "Sync Start": puls starts and stops always synchronized to internal 100 MHz clock

Mode "Async Start": pulse starts immediately with trigger (Delay=0!, stop is synchronized!).

Functions:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Get PULSE: return output PULSE

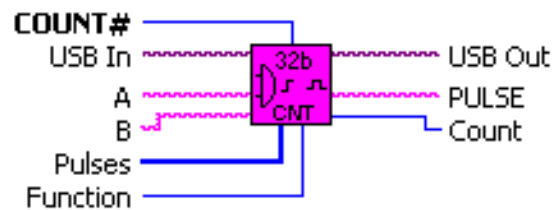
Write Params: Load Mode, Delay and Duration

COUNT#: number of module (must be unique in group COUNT)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

LabVIEW Interface



CCOUNTER.vi

Counter module with 2 inputs (A, B) and 1 output (PULSE).
 Output PULSE will be switched (toggled) on every coincidence of counter and input parameter (Pulses) values.
 Both inputs A,B are combined via an AND function into one counting signal (rising edge).
 Therefore one can be used as a GATE input for the other.

Function:

Connect: connects all in/outputs and sets all parameters

Set A: change input A

Set B: change input B

Get PULSE: return state of output PULSE

Read Counter: return current Counter value

Clear Counter: Set Counter=0

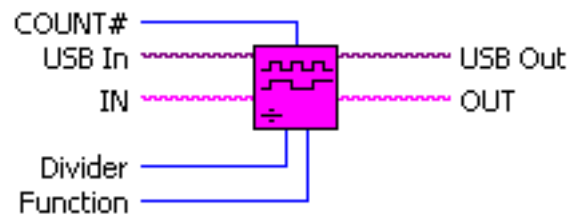
Write Pulses: load Pulses values

COUNT#: number of module (must be unique in group COUNT)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

LabVIEW Interface



DIVIDER **[DIVIDER.vi]**

Module for dividing input clocks in frequency.

An input clock at IN will be divided according to an integer divider ($1..2^{**32}-1$) in frequency at OUT. The change in polarity on OUT is caused by rising edges on IN.

Function:

Connect: connects all in/outputs and sets all parameters

Set IN: change input TRIG

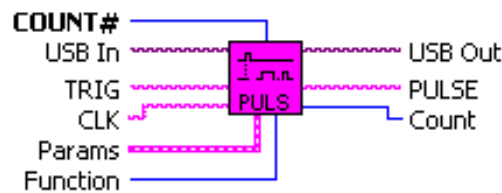
Get OUT: return state of output OUT

Write Divider: load divider parameter

COUNT#: number of module (must be unique in group COUNT)

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

LabVIEW Interface



PULSER.vi

Module for generating single (triggered) or clock pulses.

Each Pulses parameter value toggles the output PULSE at coincidence with the internal counter, which takes the input CLK (rising edge) as counting events (Count). If CLK is not connected, the system clock=100MHz is taken.

Pulse will be started at rising edge of input TRIG. If not connected, the pulse will restart automatically at the last Pulses value (CLOCK mode).

Mode Synchr. Start synchronizes start with system clock 100 MHz.

Mode Asynchr. Start starts Pulse (first value=0) immediately with TRIG.

Function:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Set CLK: change input CLK

Get PULSE: return output OUT

Write Pulses: Load Params (Pulses)

Read Counter: get current Counter value

Clear Counter: set Counter=0

COUNT#: number of module (must be unique in group COUNT)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.11 DAC (SU710)

Digital-Analog-Wandler zur Generierung schneller analoger Signalformen mit **14 Bit Auflösung** (Arbitrary Function Generator).

Die Daten können dabei über einen BUS mit einer Rate von bis zu **100 MHz** in das Modul eingespielt oder einfach in ein Datenregister geschrieben werden.

Der Ausgangsspannungsbereich liegt zwischen $-2 \dots +2$ V;

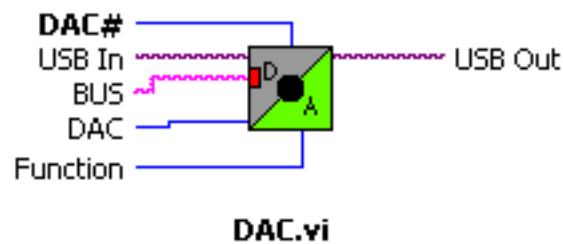
Memory Map

,E'	Read	Write
0	0	IN_BUS
1		DAC(W)

IN_BUS: Eingangssignal

DAC(W): Datenregister zur Konvertierung

LabVIEW Interface



Fast 14 Bit DAC module with max. 100MHz clock rate and BUS input
Support: SU710 (-2V..2V; Ri=50 Ohm)

Params:

DAC: 14b Data value

Inputs:

BUS: Strobe for 14b Data

Function:

Connect: connects all in/outputs and sets all parameters

Set BUS: set input

Write DAC: loads DAC value

DACB#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.12 DAC1 (SU728)

Das Modul besitzt einen **16 Bit** Digital-Analog-Wandler für Signale mit einer Einstellzeit pro Kanal von etwa **1 us** bei einem Ausgangsbereich von **-5..+5 Volt**.

Durch Laden eines 2 Byte Datenwortes (siehe Datenblatt) wird der Spannungswert gesetzt.

Über einen Eingangsbus können die Daten auch z.B. aus einem Memory (RAM) eingeschrieben werden.

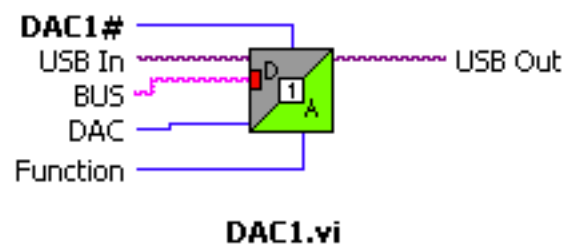
Memory Map

,h'	Read	Write
0	0	IN_BUS
		Data(W)

IN_BUS: Daten für DAC & Strobe

Data: Datenwert (siehe Datenblatt)

LabVIEW Interface



16 Bit DAC with 1 Channels
Support: SU728
Output range -5V..+5V

Params
DAC: DAC value (16 Bit)

Inputs:
BUS: Strobe for 16b Data

Function
Connect: connect module & initialize
Set BUS: connect BUS
Write DAC: load DAC

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

9.13 DAC16 (SU713, SU721)

Das Modul besitzt einen **16 Kanal, 14 Bit** Digital-Analog-Wandler für Signale mit einer Einstellzeit pro Kanal von etwa **6 us** bei einem Ausgangsbereich von **0..2,5 Volt** bzw. **0..5 Volt**.

Durch Laden eines 3 Byte Datenwortes (siehe Datenblatt AD5390) wird die Konfiguration, der gewünschte Kanal sowie der Spannungswert gesetzt.

Über einen Eingangsbus können die Daten sowie der Kanal auch z.B. aus einem Memory (RAM) eingeschrieben werden.

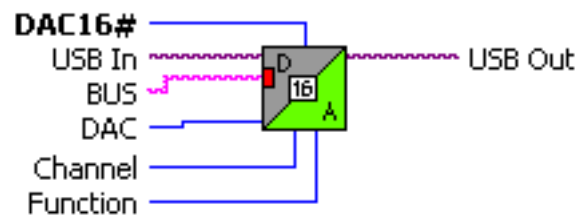
Memory Map

,H'	Read	Write
0	0	IN_BUS
		Data(T)

IN_BUS: Channel(19..16) & „00“ & DAC13..0) & Strobe

Data: Kanal und Datenwert (siehe Datenblatt AD5390)

LabVIEW Interface



DAC16.vi

14 Bit DAC with 16 Channels and Bus input
 Support: SU713, SU721
 Output range 0..5.000V (resp. 0..2.500V).

Parameter

Channel: 0..15 channel number
 DAC: DAC value

Inputs

BUS: Strobe for 14b Data

Function

Initialize: Ref=2.5V internal
 Initialize 1/2 Ref: Ref=1.25V internal (output range is half!)
 Set BUS: set input
 Write DAC: load DAC value.

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.14 DAC16S (SU713, SU721)

Das Modul besitzt einen **16 Kanal, 14 Bit** Digital-Analog-Wandler für analoge Signale mit einer Einstellzeit von etwa **6 us** pro Kanal bei einem Ausgangsbereich von **0..2,5 Volt** bzw. **0..5 Volt** sowie einem Sequenzer zur Realisierung beliebiger analoger Kurvenformen.

Durch Laden eines 3 Byte Datenwortes (siehe Datenblatt AD5390) wird die Konfiguration, der gewünschte Kanal sowie der Spannungswert gesetzt.

Im Sequenzermodus werden nach einem Start durch TRIGGER über den REQUEST-Ausgang entsprechend Daten von einem externen Memory angefordert. Die Daten werden über den BUS-Eingang eingeschrieben.

Die zeitliche Steuerung erfolgt entweder mit der Systemclock (100MHz) oder mit einer externen Clock. Ein zusätzlicher Parameter Divider erlaubt jeweils bei reduzierter Auflösung extrem lange Zeiten.

Die Daten werden entsprechend dem Format vom Sequenzer entweder als Zeit- oder als DAC-Information interpretiert und ausgeführt. Zeitinformationen starten sofort einen Timer, der nach der Zeit TIME (bestimmt durch Clock und Divider) alle DAC-Kanäle auf die inzwischen vorgeladenen DAC-Werten setzt.

Dabei haben die Daten (32b) folgendes Format:

Zeitinformation: „1“ & TIME(30..0)

DAC-Information: „0“ & “00000000“ & Kanal(19..16) & Wert(13..0)

Es ist darauf zu achten, dass die Zeit des Timers ausreicht um alle gewünschten Kanäle auf die neuen Werte zu setzen!

Durch das spezielle Datenwort= „FFFFFFF“ wird die Sequenz angehalten und kann durch einen erneuten Trigger fortgesetzt werden.

Durch das spezielle Datenwort= „80000000“ wird die Sequenz endgültig gestoppt!

Beide Zustände können in Statusflags abgefragt werden.

Memory Map

,w'	Read	Write
0	OUT_Request	IN_BUS
1	Sequence(2), Running(1), Loading(0)	IN_TRIGGER
2		IN_CLOCK
3		Divider(L) & Arm
4		Data(T) & Stop

OUT_Request: Ausgangssignal für Memory Request

Sequence, Running, Loading: Status Flags

IN_BUS: Eingangssignal für Data & Strobe

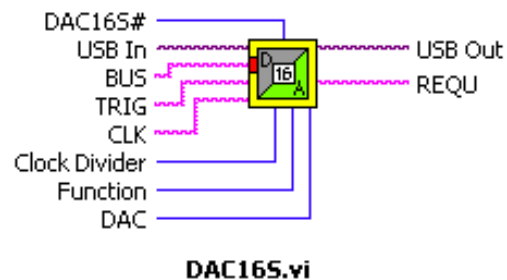
IN_TRIGGER: Startsignal

IN_CLOCK: externe Clock

Divider: Clock Teiler

Data: Kanal und Datenwert (siehe Datenblatt AD5390)

LabVIEW Interface



14 Bit DAC with Sequencer and BUS Input

BUS delivers either Time (D(31)=1) or with D(31=0), channel(19..16) & DACvalues(13..0).
DACvalues will only be preloaded for next load of Timestep!

Time="11..1": Sequencehaltss with Running=false, Sequence=true;
Time="10..0": Sequence stops with Running=false, Sequence=false;

Support: SU713, SU721

Output range 0..5.000V (resp. 0..2.500V).

Clock internal (100MHz) or external (CLK) with Divider.

Inputs:

DAC16S#: Module number (16 channels)

BUS: Memory Input

TRIG: Sequencer starts on rising edge (Latency of 100ns)

CLK: Clock for global synchronisation (max. 40MHz), if not connected = 100MHz

DAC: for direct load: channel(19..16) & DACvalues(13..0)

Clock Divider: 32b scaler for input frequency

Outputs:

REQU: signal for Memory to request next data

Functions:

Connect: Ref=2.5V internal

Connect 1/2 Ref: Ref=1.25V internal (output range is half!)

SET BUS: set input BUS

SET TRIG: set input TRIG

SET CLK: set input CLK (internal clock if open)

GET REQU: read output REQU

Read Status: Sequence(2) & Running(1) & Loading(0)

Write Divider: divider for CLK (100 MHz)

Write DAC: load DAC values immediately & Stop Sequence

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.15 DACR (SU710)

Generierung beliebiger analoger Signale bis zu einer Samplingrate von bis zu **100 MHz** bei **14 Bit Auflösung** (Arbitrary Function Generator).

Durch einen Divider (16 Bit) kann die Samplingrate entsprechend verlangsamt werden. Die Daten werden dabei aus einem vorgeladenen RAM ausgelesen und der Ablauf immer wieder neu durch einen Trigger gestartet. In einem Continuous-Mode kann der Ablauf automatisch neu gestartet werden um damit ein kontinuierliches Signal zu erzeugen.

Der Ausgangsspannungsbereich liegt zwischen $-2 \dots +2$ V;

Memory Map

,'e'	Read	Write
0	OUT_BUSY	IN_TRIG
1		Continuous(0) & Reset
2		Divider(W)
3		RAM_Addr (W)
4		RAM+ (W)

OUT_BUSY: Anschlusswert für Ausgang BUSY

IN_TRIG: setzt Eingang TRIG

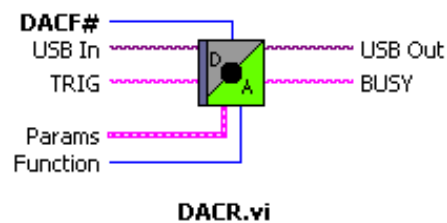
Continuous: Autorepeat Mode

Divider: Samplingrate

RAM_Addr Adresspointer

RAM+: Memory

LabVIEW Interface



Fast 14 Bit DAC module with max. 100MHz clock rate and AFG (Arbitrary FunctionGenerator)

Support: SU710 (-2V..2V; Ri=50 Ohm)

Values, stored in an onboard memory (typ. 0..1024), will be read out after a trigger on input TRIG with selectable rate (Divider) and converted to analog voltages. This allows to generate an arbitrary analog time function (AFG= Arbitrary Function Generator).

In mode "Continuous" the sequence will be repeated automatically for periodic signals. The output BUSY denotes the running of the module.

If the module is not started, the value of address=0 is generated.

Inputs:

TRIG: rising edge starts DAC Function

Outputs:

BUSY: true on running

Params:

Divider: divider for any frequency $f = 100\text{MHz}/\text{Divider}$.

Continuous: chooses auto repeating mode.

Addr: start address for loading DATA.

DATA: data array (14 Bit). The length of the array determines the length of the generated function.

Function:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Get BUSY: return output state BUSY

Write Params&Clear: loads alle parameters and stops

DACF#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.16 DELAY (SU711)

Realisierung einer von der Systemclock unabhängigen Verzögerung eines beliebigen Signals.

Die Verzögerung oder Durchlaufzeit ist für jedes Modul separat und unabhängig mit $n=0..255$ einstellbar. Damit ist jede mit diesem Baustein erzeugte Funktion jitterfrei bzw. immer synchron zum Signaleingang oder Trigger.

In einem speziellen Modus MonoFlop wird die Verzögerung zur Erzeugung eines asynchronen Pulses verwendet.

Mehrere, auch gemischte, Bereiche sind möglich.

Die Auflösung und der Bereich der Verzögerung ist bausteinabhängig, wobei eine Grundlaufzeit von typ. 16.5 ns zu beachten ist.

Delays

Baustein	Step/ns	Bereich/ns
DS1023-25	0.25	0..63.75
DS1023-50	0.50	0..127.5
DS1023-100	1	0..255
DS1023-200	2	0..510
DS1023-500	5	0..1275

. ACHTUNG: bestimmte DL7xx-Karten unterstützen nur 5 Delaymodule!

Memory Map

,Z'	Read	Write
0	OUT_Delay	IN_Signal
1		Delay
2		MonoFlop(0)

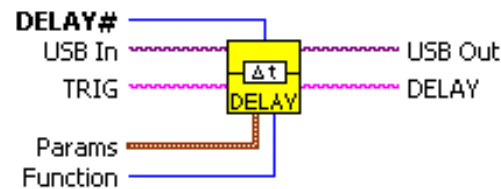
OUT_Delay: Anschlußwert für Ausgang Delay

IN_Signal: Eingang Signal

Delay: Verzögerungswert

MonoFlop: Modus Asynchrones FlipFlop

LabVIEW Interface



DELAY.vi

Programmable Delay for fine tuning of pulse delays (transmission lines) and pulse generating.

Support: SU711

These delays are completely synchronous to any input signal TRIG (no jitter). The base delay and the range of delays is determined by the delay components on SU711.

In Mode "Monoflop" this delay will be used to generate jitter free pulses with a programmable length.

Delay/ns: delay (typ. $(30 \text{ ns} + 0..255) * 0.5 \text{ ns}$).

Mode:

Delay: delays input signal TRIG for Delay

Monoflop: generates a single pulse with length Delay after TRIG (rising edge)

Function:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Get DELAY: return current state of DELAY output

Write Delay: loads Delay value (0..255)

Write Mode: loads Mode

DELAY#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

Steckerbelegung

Pin	Signal	Bedeutung
1	+ 5V	Spannungsversorgung
2	+5V	Spannungsversorgung
3	D0	Dateneingang
4	D1	Dateneingang
5	D2	Dateneingang
6	D3	Dateneingang
7	D4	Dateneingang
8	D5	Dateneingang
9	D6	Dateneingang
10	D7	Dateneingang
11	Input 1	Signaleingang 1
12	LE 1	Latch Enable 1
13	Output 1	Signalausgang 1
14	Ref/PWM 1	Reference/Puls Width 1
15	Input 2	Signaleingang 2
16	LE 2	Latch Enable 2
17	Output 2	Signalausgang 2
18	Ref/PWM 2	Reference/Puls Width 2
19	Input 3	Signaleingang 3
20	LE 3	Latch Enable 3
21	Output 3	Signalausgang 3
22	Ref/PWM 3	Reference/Puls Width 3
23	Input 4	Signaleingang 4
24	LE 4	Latch Enable 4
25	Output 4	Signalausgang 4
26	Ref/PWM 4	Reference/Puls Width 4
27	Input 5	Signaleingang 5
28	LE 5	Latch Enable 5
29	Output 5	Signalausgang 5
30	Ref/PWM 5	Reference/Puls Width 5
31	Input 6	Signaleingang 6
32	LE 6	Latch Enable 6
33	Output 6	Signalausgang 6
34	Ref/PWM 6	Reference/Puls Width 6
35	GND	Spannungsversorgung und Signalreferenz
36	GND	Spannungsversorgung und Signalreferenz

9.17 DIO (SU700, ...)

Dieses Modul unterstützt auf allen SU-Karten nahezu alle digitalen Ein- und Ausgänge, wie TTL, NIM, LVDS, o.ä, und besitzt einen Eingang DO (Ausgangssignal) und einen Ausgang DI (Eingangssignal).

Falls die Hardware dies unterstützt (SU704) kann ein Relais zur Zuschaltung eines Abschlusswiderstands von 50 Ohm betätigt werden.

Memory Map

,T'	Read	Write
0	OUT_DI	IN_DO
1		NIM(1) & Termination(0)

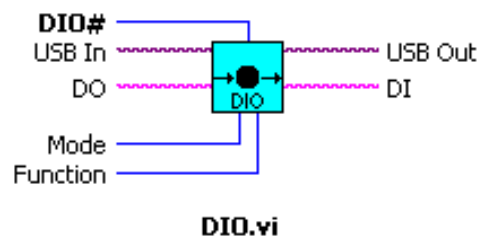
OUT_DI: Liefert Anschlusswert & Status für das Eingangssignal

IN_DO: Setzt DO für das Ausgangssignal

Termination: Setzt 50Ohm Relais (SU704)

NIM: wählt NIM Pegel (SU704)

LabVIEW Interface



Digital Input/Output.

Support: SU700, SU701, SU703, SU704, SU706, SU707

According to hardware this supports either TTL,, TTL_coax, NIM or DIFFERENTIAL I/Os.

Each connector can be used as either an output (DO) or input (DI) or both.

If used as an input, terminal DO (driving output) must not be connected, as this enables the output driver and would override any input signals.

Please obey termination rules as defined for the specific used hardware!

Params:

Mode: set according to used hardware module and level standards.

(SU704 supports switchable 50 Ohm termination!)

Debounce/ms: debounce delay for suppressing intermittent signals (1..255 ms). debounce=0 disables this function!

Function:

Connect: connects all in/outputs and sets all parameters

Set DO: change input DO

Get DI: return output state at DI

Write Mode: load appr. mode

DIO#: number of connector (must be unique!)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.18 DISCR (SU703)

Ein analoges Eingangssignal wird mit einer schnellen Komparator-Schaltung (Discriminator) als digitales Signal DISCR verfügbar. Über entsprechende Parameter wird die Eingangsscharakteristik eingestellt:

Threshold:

Die Schwelle (-2.5V..+2.5V) für das analoge Eingangssignal wird über einen 12Bit DAC gesetzt. Die oberen 4 Bits bestimmen den Kanal pro Modul (siehe Datenblatt!).

Hysterese:

Die Hysterese (0..60mV) für das analoge Eingangssignal wird über einen 12Bit DAC gesetzt. Die oberen 4 Bits bestimmen den Kanal pro Modul.

Im Model 1 kann durch einen eingebauten GateGenerator das Signal DISCR auf einen Puls mit einstellbarer Länge verlängert werden (Stretch).

Memory Map

,D'	Read	Write
0	OUT_DISCR	Threshold(W)
1		Hysterese(W)
2		Stretch

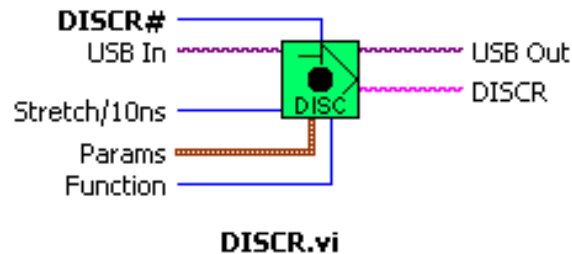
OUT_DISCR: liefert Anschlusswert für DISCR

Threshold: Einstellung für Schwelle -2,5V .. +2,5V

Hysterese: Einstellung für Hysterese 0..60mV

Stretch: Eingangssignal wird auf Normsignal 10..2550 ns geformt; 0=Originalsignal

LabVIEW Interface



Discriminator module for analog inputs.

Support SU703.

The analog input (connector) is compared to a programable Threshold (typ. $-2.5V \dots +2.5V$) to generate a resulting digital output signal (DISCR).

A programable Hysteresis voltage (ty. $0 \dots 60mV$) can suppress noise which causes intermittent switching.

V1 supports a gate generator for 10 ns..2550 ns pulses on input rising edge!

Please note, that there is an inherent 10 ns jitter (related to input) on this gate!

Function:

Connect: connects all in/outputs and sets all parameters

Get DISCR: return output state at DISCR

Write Threshold: load threshold DAC

Write Hysteresis: load hysteresis DAC

Write Stretch: load gate generator 0..255

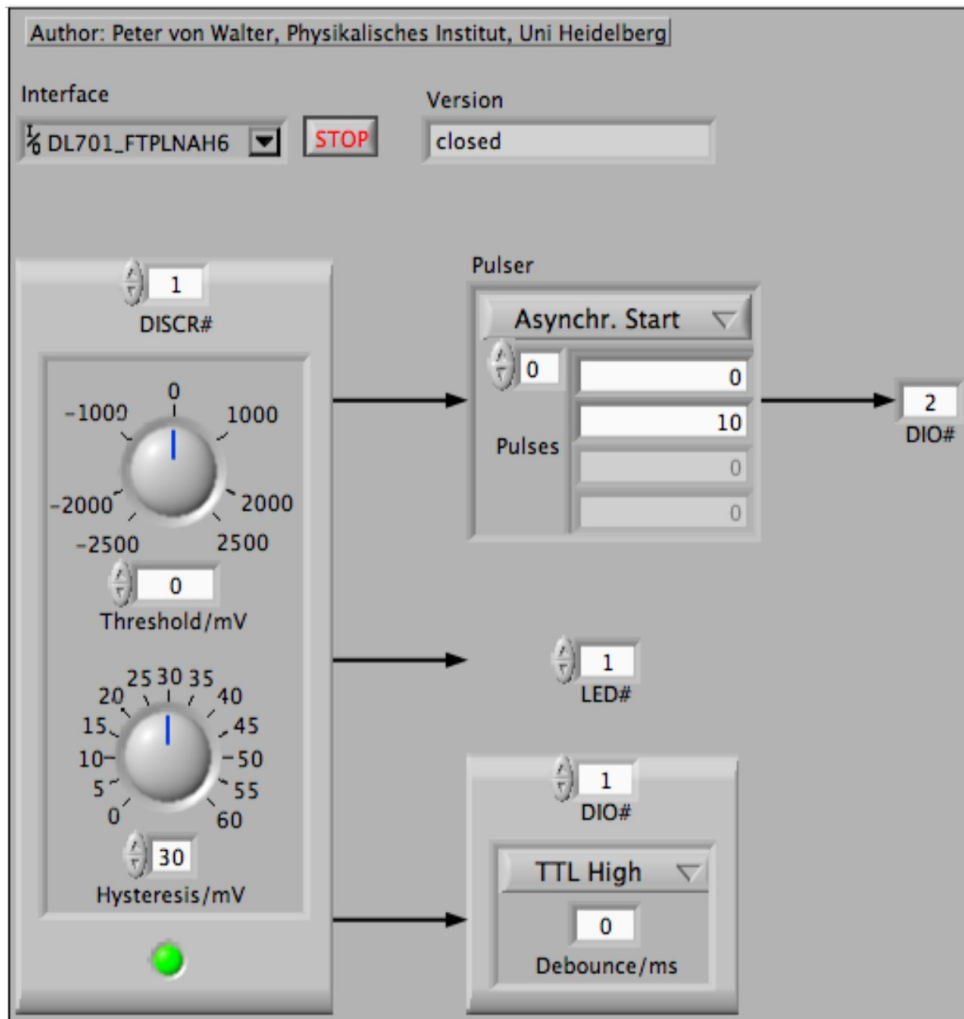
DISCR#: number of module (must be unique)

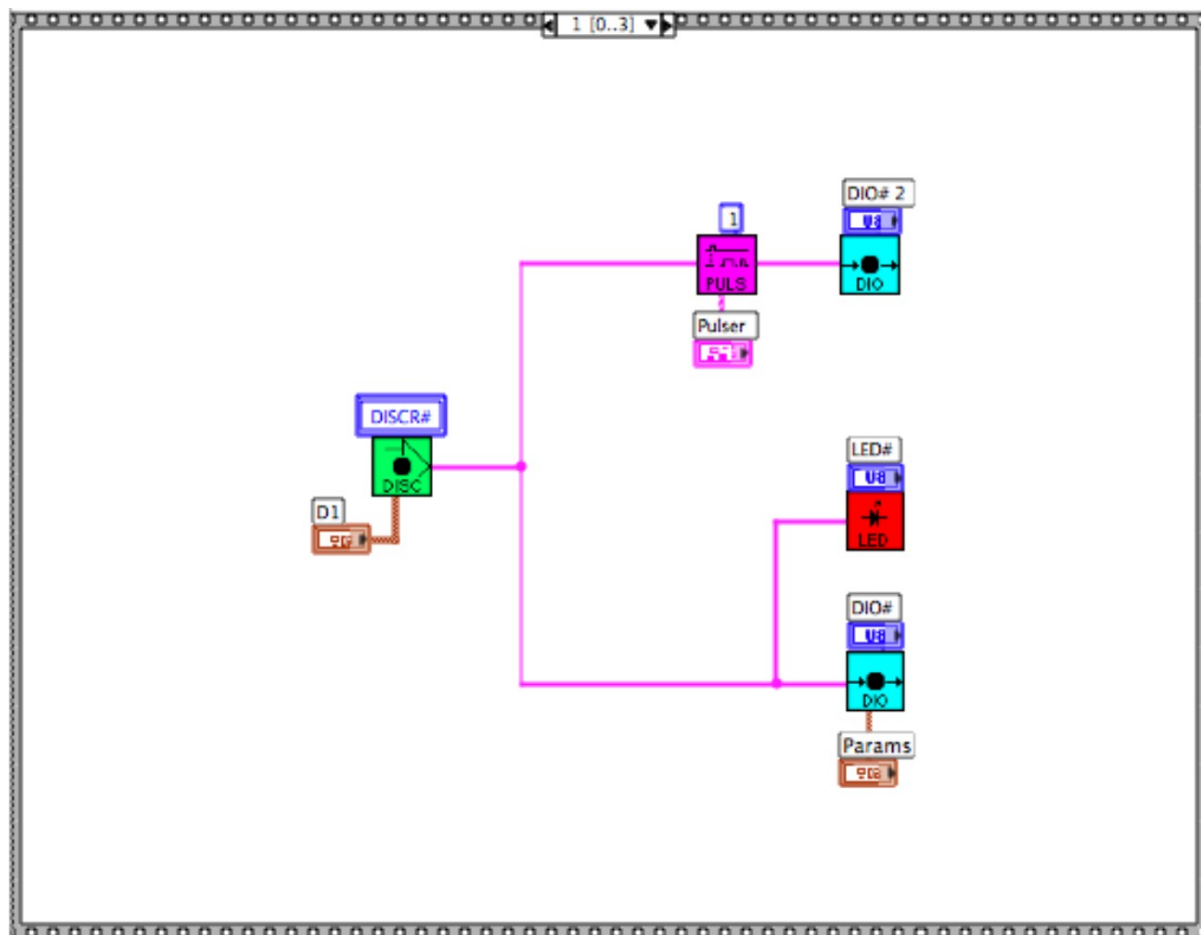
USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

Performance

Die folgenden Messungen sollen die Eigenschaften des Discriminators SU703, sowie eines nachgeschalteten Pulsers (Pulsstretcher) zeigen.





Auf der LogicBox ist oben gezeigte Konfiguration realisiert:

Das analoge Eingangssignal aus einem schnellen Pulser wird zur Beobachtung an einem Scope (Kanal 3, violett) hochohmig vorbeigeführt und (nach 4 ns) am Eingang DISCR#=1 (50 Ohm) in die SU703 eingespeist.

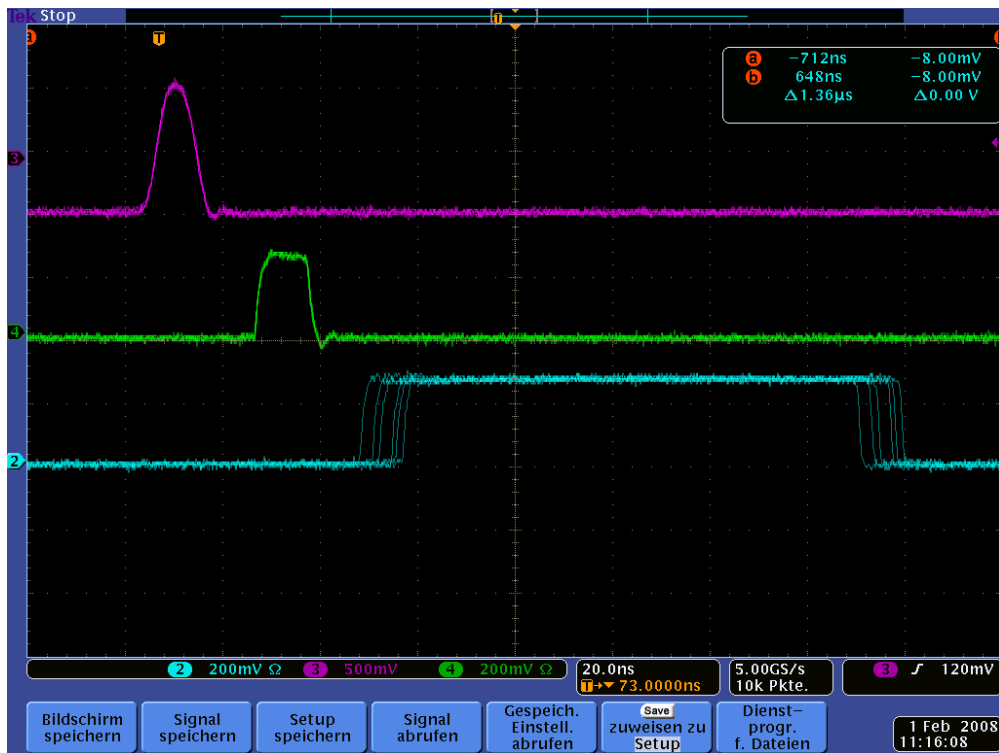
Das Signal wird durch den Discriminator über die einstellbare Schwelle in ein digitales Signal umgewandelt, intern weitergeleitet und so direkt wieder an den TTL-Ausgang DIO#=1 ausgegeben und über 3 ns Kabel am Kanal 4, grün (50 Ohm) angezeigt.

Die parallele Ausgabe an der LED#=1 dient lediglich zur Kontrolle!

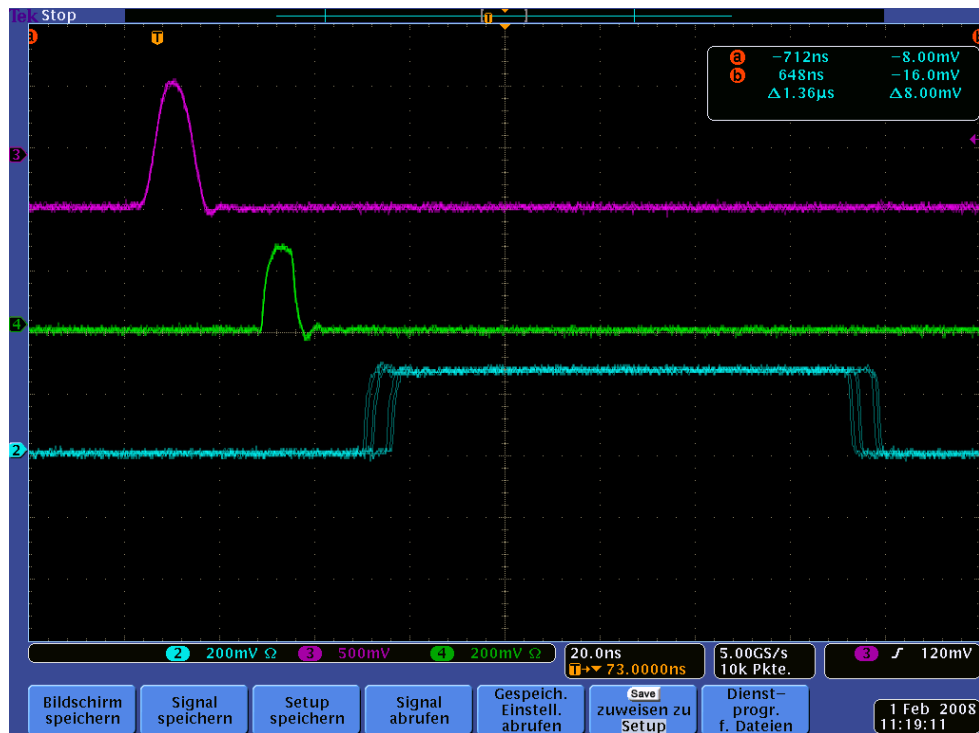
Gleichzeitig liegt das Discriminatorsignal am Triggereingang eines Pulsers, der frei programmierbar ist und durch die steigende Flanke getriggert wird. Dieses Signal wird ebenfalls an einen TTL-Ausgang DIO#=2 ausgegeben und über 3 ns am Kanal 2, blau, (50 Ohm) angezeigt.

Schwellenabhängigkeit

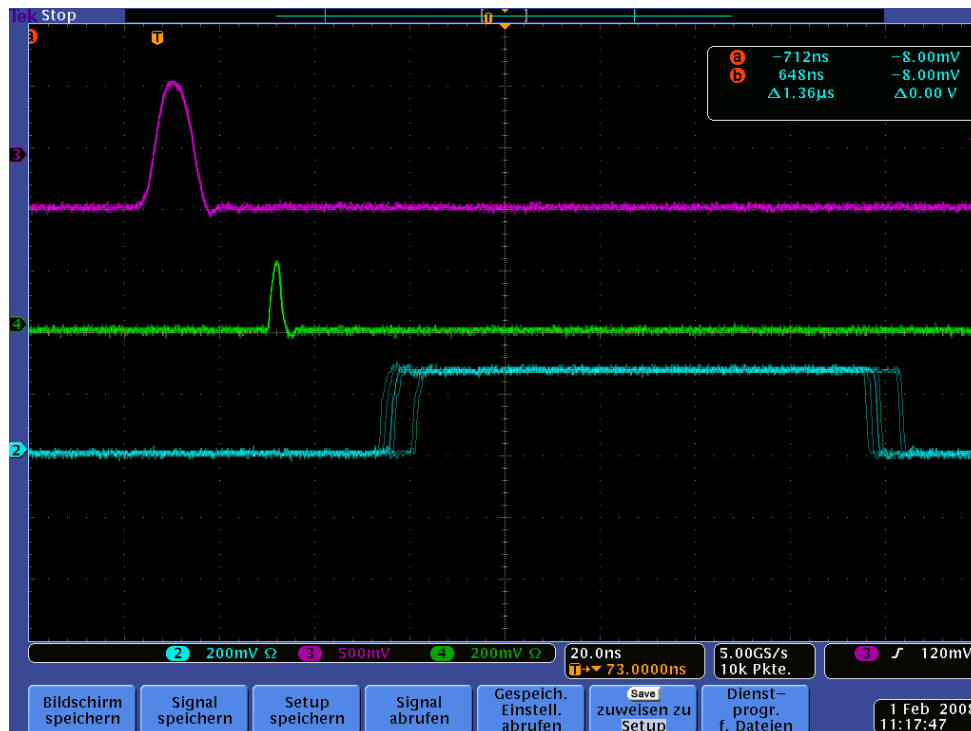
Im folgenden wird bei einem typischen Eingangsimpuls (HP8012B) von $-500\text{mV} \dots +500\text{mV}$, minimal einstellbare Pulsdauer ca. 10 ns, die Abhängigkeit von der Schwellen-Einstellung des Discriminators gezeigt.



Schwelle: -400mV; Hysterese: 30mV



Schwelle: 0mV; Hysterese: 30mV



Schwelle: +490mV; Hysterese: 30mV

Man kann im wesentlichen die Variation auf die Ausgangspulsbreite (grün) sehen. Der Ausgang des nachgeschalteten Pulsers wird nicht beeinflusst. Dieser arbeitet im synchronen Modus, d.h. Start und Ende sind durch die interne Systemclock (100 MHz) bestimmt, damit ergibt sich der typische Timejitter von 10 ns am Anfang und Ende!

Eingangspulsbreite und Signalthöhe

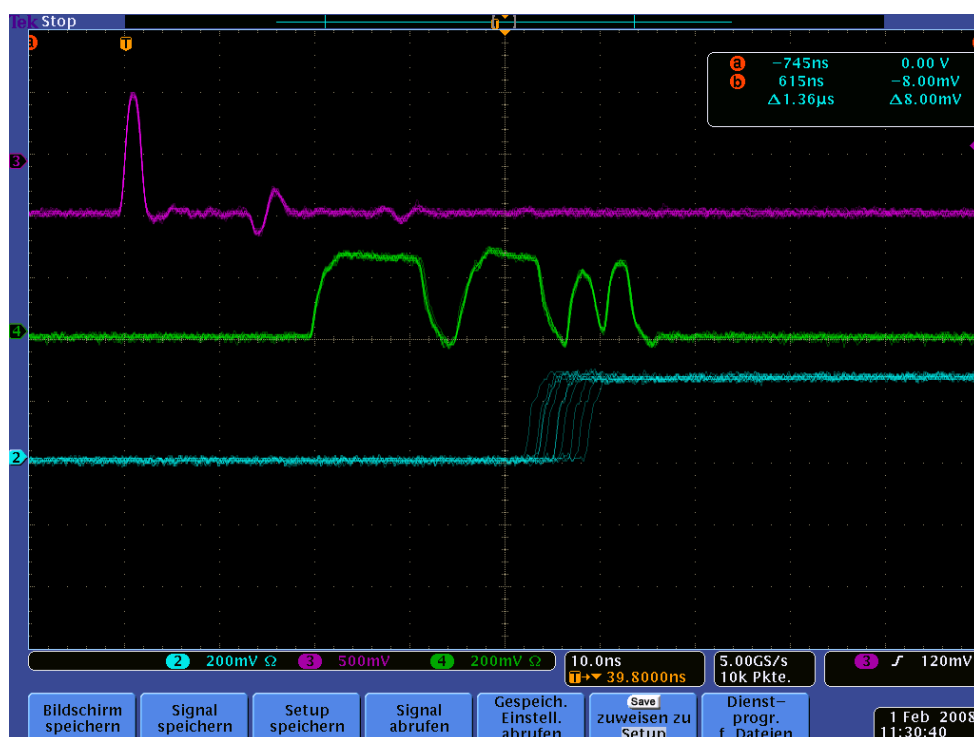
Im folgenden wird getestet, inwieweit sich die Eingangspulsbreite und die Signalthöhe (EH 129) auf die Triggerfähigkeit des Pulsstretchers (Pulser) auswirkt.

ACHTUNG: Zeitskala 10 ns/Div.



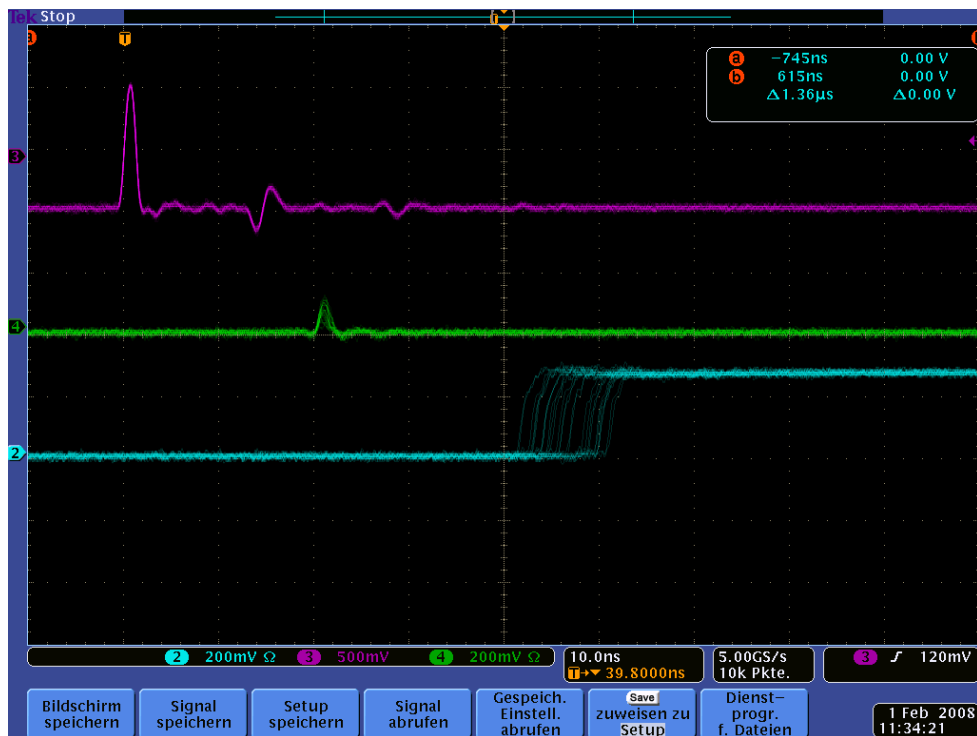
Schwelle: 0mV; Hysterese: 30mV

Hier ist die minimal einstellbare Impulsbreite (ca. 2 ns) des Eingangssignals gezeigt. Der Ausgang des Puls-Stretchers zeigt keine Variation und triggert in der gleichen Weise.



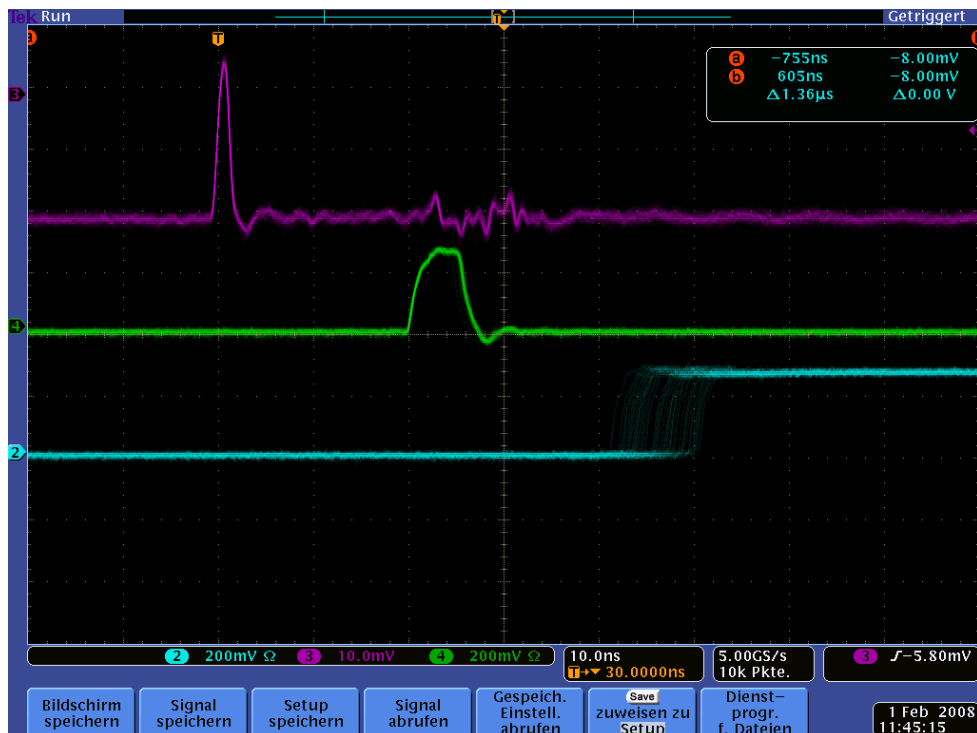
Schwelle: -490 mV; Hysterese: 0mV

Die Schwelle liegt nahezu in der Baseline des Signals. Kleine Reflexionen und Rauschen führen zu einem mehrfachen Ausgangssignal. Der Puls-Stretcher wird in der gleichen Weise durch den ersten Puls getriggert und hat die gleiche Länge, da kein Retrigger stattfindet.



Schwelle: +490 mV; Hysterese: 0mV

Hier liegt die Schwelle sehr hoch am Peak. Der Ausgangspuls wird so kurz, dass der TTL-Treiber mit seinen Anstiegszeiten an die Grenze kommt. Das interne Signal reicht jedoch ohne Probleme zur Triggerng des Puls-Stretchers.



Schwelle: 0 mV; Hysterese: 0mV

Die Eingangsamplitude ist hier bis auf ca. 20 mVpp abgesenkt. Ausgangspuls und

Pulstretcher arbeiten auch hier ohne Probleme!

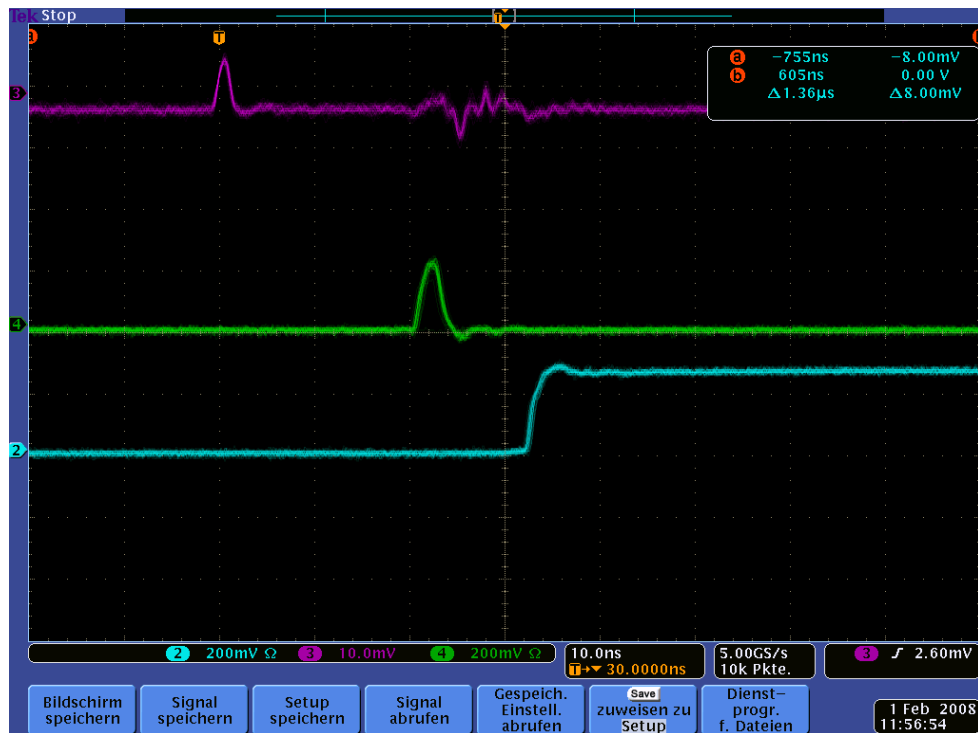


Bild 9: Schwelle: 0 mV; Hysterese: 0mV

Die Eingangsamplitude ist hier bis auf ca. 10 mVpp abgesenkt.

Der Pulser arbeitet im Asynchron-Modus ohne Jitter beim Start des Pulses!

Pulser Synchron/Asynchron-Modus

Der Pulser kann in zwei Modi betrieben werden:

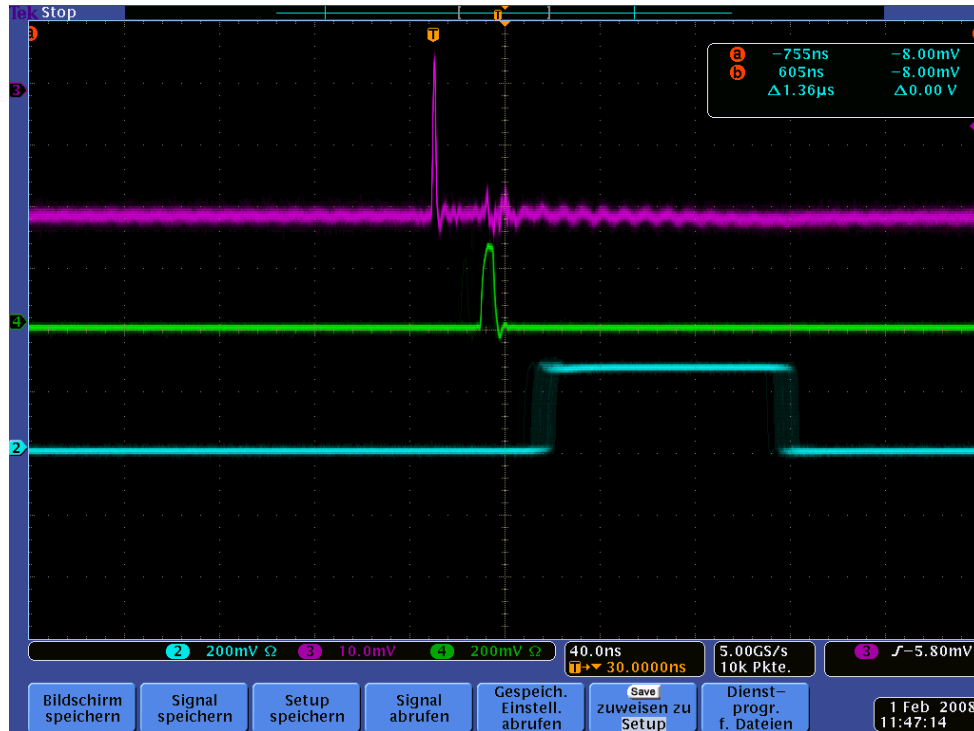
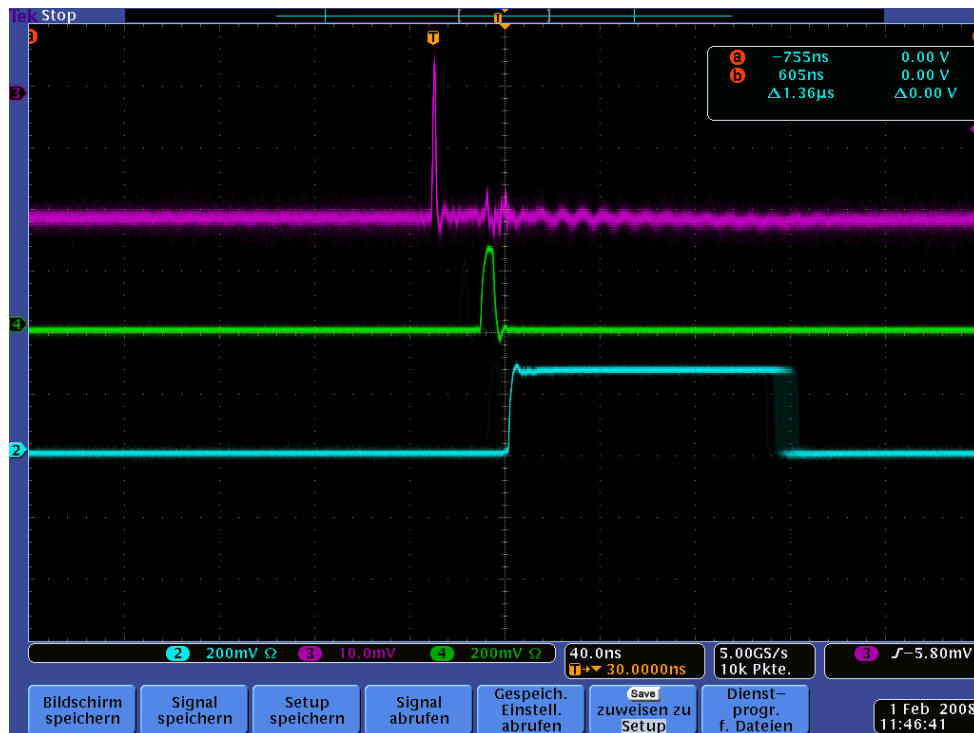


Bild 10: Synchron Modus

Beginn und Ende des Pulses sind systembedingt mit einem Jitter=10 ns behaftet.
Verzögerung (Delay) und Dauer sind frei einstellbar.

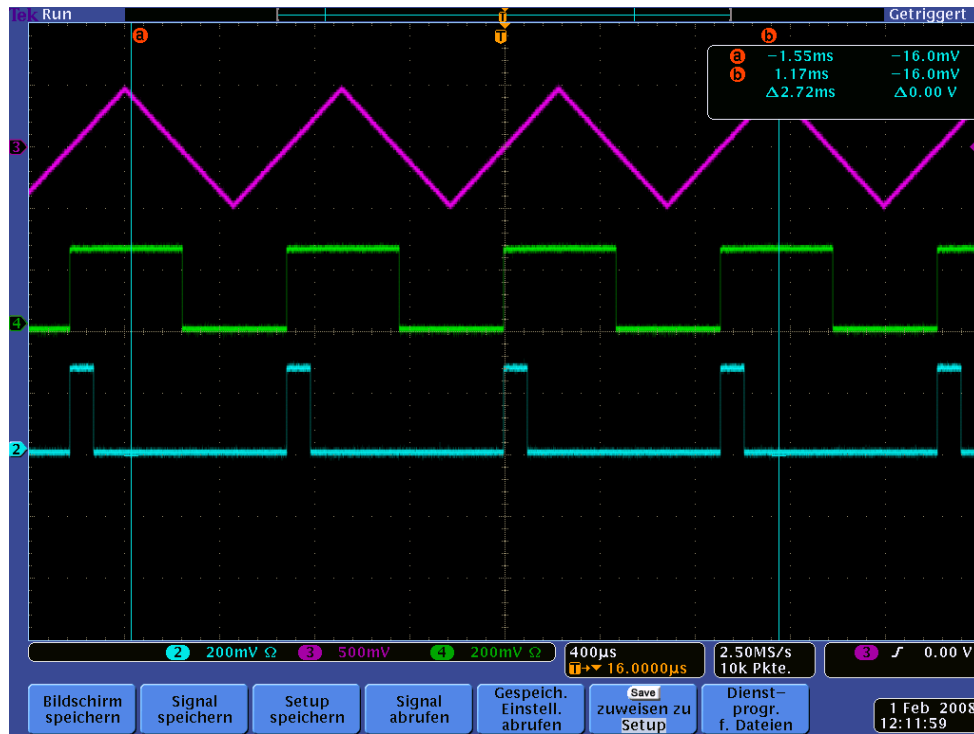


Asynchron Modus

Bei Delay=0 kann der Start des Pulses synchron zum Trigger erfolgen (ohne Jitter). Das Ende des Pulses ist allerdings immer synchron zur Systemclock (mit Jitter).

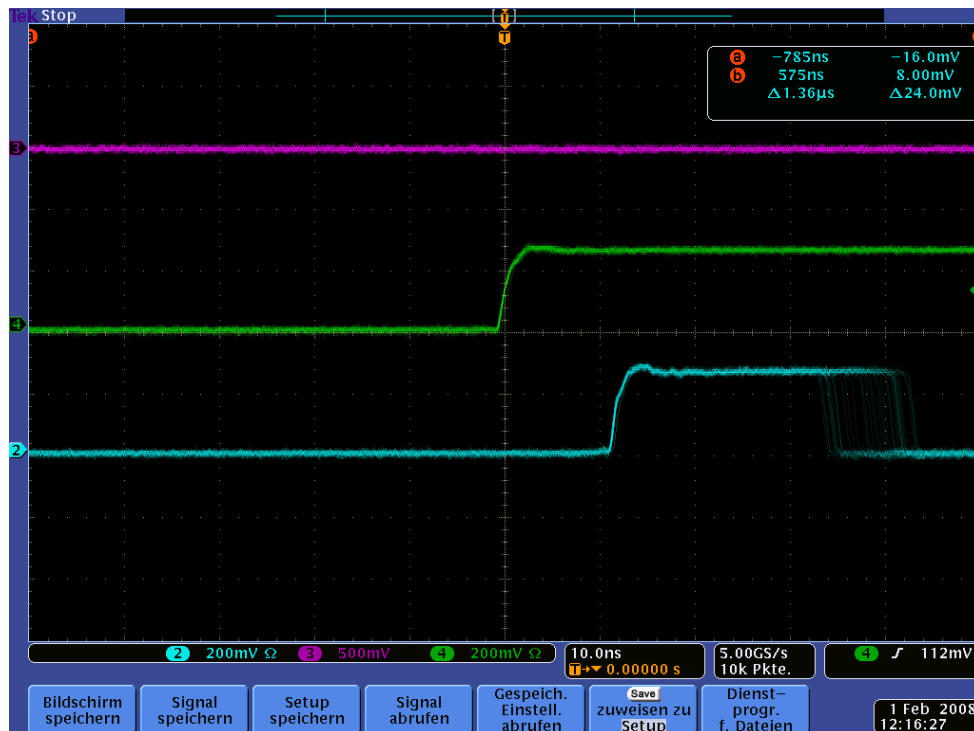
Discriminator bei langsamen Pulsen

Die folgenden Bilder zeigen das Verhalten bei sehr langsamen Eingangssignalen (Slewrate=1V/400us) und deren Durchgang durch die Discriminatorschwelle.



Schwelle: 0 mV; Hysterese: 30mV

Man sieht die typischen Kurven des Discriminatorsausgangs und des Pulsers (Dauer= 100us).



Schwelle: 0 mV; Hysterese: 30mV

Eine zeitlich genauere Betrachtung am Schwellendurchgang zeigt das saubere Schaltverhalten. Durch den Hysteresewert von 30 mV wird das Rauschen wirksam unterdrückt!

Der Pulser arbeitet hier im Asynchron-Modus. Man beachte, dass sich zur eingestellten Zeitdauer = 10 ns ein bestimmter Grundbetrag addiert!



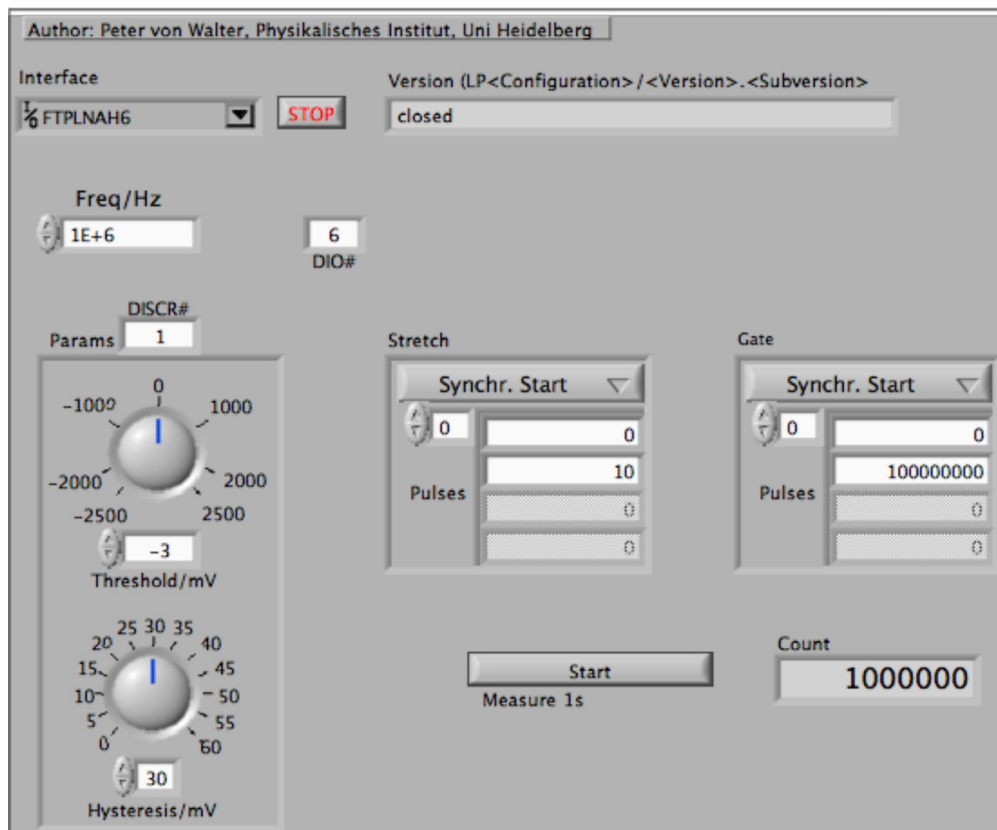
Schwelle: 0 mV; Hysterese: 10mV

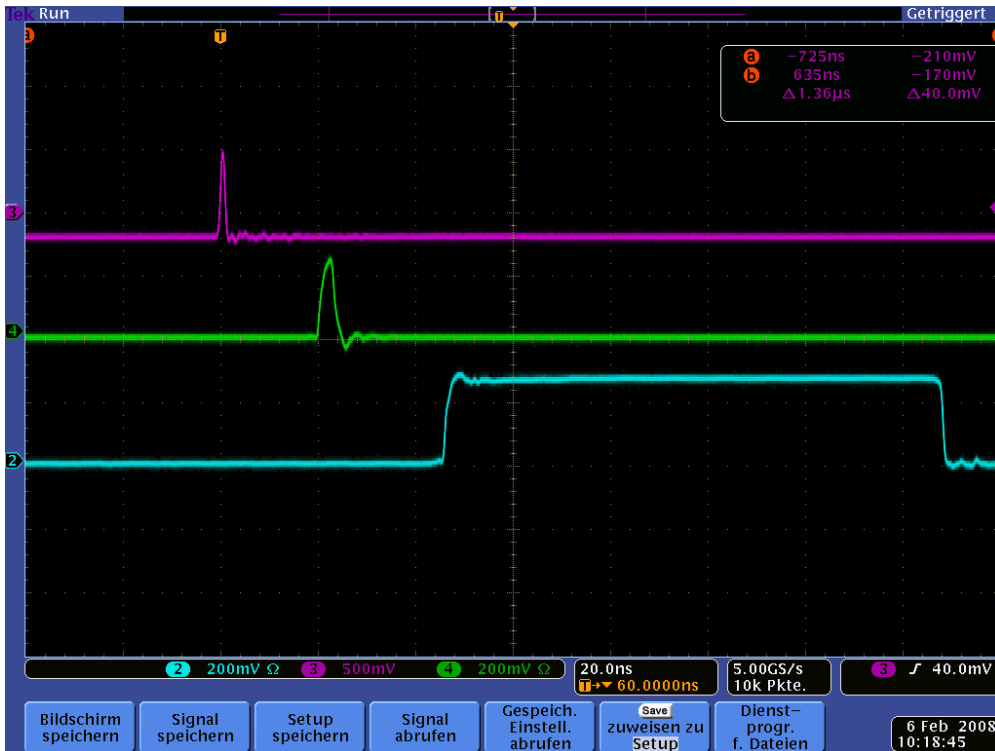
Hier arbeitet der Discriminator im Grenzbereich.

Durch die niedrige Hysterese (und die geringe Slewrate!) bewirkt das Rauschen des Eingangssignals ein Mehrfachtriggern der nachfolgenden Schaltungen.

Efficiency

Mit folgender Schaltung wurde versucht, die Efficiency des Discriminators mit nachgeschaltetem Stretcher zu messen!





Violett: Analog-Input; Grün: Discr.-Ausgang; Blau: Stretcher-Ausgang
Schwelle: -100mV; Hysterese: 30mV;



Violett: Analog-Input; Grün: Discr.-Ausgang; Blau: Stretcher-Ausgang
Schwelle: -3mV; Hysterese: 30mV;

Hier ist wieder der Grenzfall gezeigt, dass der Discr.-Ausgang durch die relativ hohe Schwelle nahezu verschwindet (wird zu kurz!). Der Stretcher ist dadurch aber nicht beeinträchtigt. Es

ist zu beachten, dass sogar bei einer Schwelle von +100 mV der Stretcher jeden Puls sieht!

Nach dem Stretcher (2), der hier auf 100 ns eingestellt ist, werden die Pulse in einem Zähler (4) gezählt, der durch ein manuell getriggertes Gate (1) aufgemacht wird.

Man beachte, dass alle verwendeten Zeitglieder von der internen 100 MHz Systemclock abgeleitet sind und sich deshalb voll synchrone (hochgenaue) Verhältnisse ergeben. Insbesondere ist jetzt der Eingangspuls immer synchron zu allen internen Zeitabläufen (Stretcher).

Auf dem Frontpanel ist die ermittelte Zählrate im Feld „Count“ abzulesen. Diese beträgt immer genau 1000000 (bzw. 1000001), was bedeutet, dass über den Messzyklus von 1s alle (!) Pulse gezählt werden und keine Pulse verloren gehen. Die Efficiency beträgt also in diesem Aufbau immer 100%!.

Genau genommen, kann die Zählrate auch ab und zu 1000001 betragen, was auf einen systematischen Effekt durch den auf 100 ns verlängerten Puls zurückzuführen ist. In diesem Fall wird ein durch das Gate angeschnittener Puls mitgezählt.

9.19 FIFO

Dieses Modul speichert Daten, die von entsprechenden anderen Modulen generiert und ausgegeben werden über den Eingang BUS in einem Speicher ab.

Hier gibt es zwei Modi:

FIFO:

Daten werden zeitlich nacheinander in einem Port eingeschrieben und können zeitlich unabhängig in der gleichen Reihenfolge über ein anderes Port wieder ausgelesen werden. Die Anzahl (COUNT) der im Speicher vorhandenen Datenworte (maximum typ. 1024) kann jederzeit ausgelesen werden und z.B. mit einem Blocktransfer geleert werden. Das Ausgangssignal FULL wird gesetzt ($FULL > 0$) wenn die Anzahl der Datenworte einen programmierbaren Wert erreicht oder übersteigt.

$FULL > 0$: In diesem Fall werden nur noch Daten weiter eingeschrieben bis die maximale Anzahl der Daten erreicht ist und dann OVFL gesetzt.

$FULL = 0$: In diesem Fall werden auch bei einem Überlauf die Daten weiter eingeschrieben und alte Daten überschrieben. Der Auslesepointer wird dabei mit dem Einschreibepointer mitgesetzt. In dieser Betriebsart kann also kontinuierlich eine Vorgeschichte aufgezeichnet werden, bis das Einschreiben definitiv gestoppt wird!

Solange die mittlere Auslesegeschwindigkeit größer als die Einschreibgeschwindigkeit bleibt, gehen keine Daten verloren. Andernfalls kann es zu einem Überlauf kommen, der in einem Flag (OVFL) signalisiert wird.

HISTO:

Die Daten werden sofort über das Eingangsport entsprechend Ihrem Wert in ein Histogramm (Länge typ. 1024) einsortiert. Das Histogramm kann jederzeit über das Ausleseport komplett ausgelesen werden. Hier gehen keine Daten verloren. Falls der Eingangswert größer als die Länge des Speichers ist, wird dies mit dem Signal OVFL signalisiert!

Der Eingang WRITE erlaubt die Unterdrückung bzw. das verzögerte Einschreiben des aktuellen Datenwortes an BUS. Falls WRITE nicht beschaltet ist, werden alle Daten sofort gespeichert.

Memory Map

,F'	Read	Write
0	OUT_FULL	IN_BUS
1	OVFL(15), COUNT(10..0)	IN_WRITE
2	MEMORY(W,L)	MEMORY(W,L)
3		HISTO(15), Full(10..0) & Clear

OUT_FULL: Ausgangssignal für COUNT \geq Full

OVFL: signalisiert Überlauf

COUNT: Anzahl der Daten im FIFO (HISTO = 1024)

MEMORY: Auslesen der Daten

IN_BUS: setzt Eingang BUS

IN_WRITE: setzt Eingang WRITE

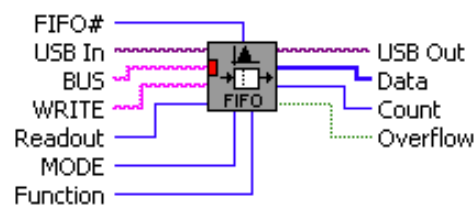
MEMORY: zum Löschen des Histogramms

HISTO: Modus Histogramm (1) / FIFO (0)

Full: setzt Ausgangssignal FULL bei COUNT \geq Full

Clear: setzt Addresscounter für FIFO und zum Löschen des Histogramms auf 0

LabVIEW Interface



FIFO.vi

Memory Module with FIFO and HISTOGRAMMER functionality.

INPUTS:

BUS: input for data with data strobe

WRITE: if connected is used for strobing data to memory

Readout: number of data words to be read (0=default: all data will be read)

DATA:

Overflow: FIFO: true if memory full; HISTO: data >1024

Write to FIFO continues on memory Full (read pointer=write pointer)

Count: FIFO: number of data in memory; HISTO: 1024

Data: array of data or histogram

Function:

Connect: connects in&outputs and loads all parameters.

Set BUS: set data BUS strobe

Set WRITE: set WRITE input

Write Memory&Clear:select memory mode, clear FIFO

Read Status: return state of OVERFLOW & COUNT.

Read Data: read FIFO or HISTOGRAM.

Clear HISTO: set Histogram values to 0

FIFO_HISTO#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.20 FREQUENCY

Dient zur vereinfachten Ermittlung von Eventraten bzw. Frequenzen am Eingang TRIGGER. Dies erfolgt abhängig von dem Parameter TIME in zwei Modi:

Time=0: Ein interner Zähler zählt die Zeit (Periode) in 10ns Schritten zwischen zwei Triggerereignissen (rising edge). Dies ist vorteilhaft bei sehr langsamen Raten.

Time=n: Hier wird repetitiv ein Messzyklus mit $n \cdot 10\text{ns}$ gestartet und während dieser Zeit die Anzahl der Triggerereignisse gezählt. Dies ist vorteilhaft bei hohen Raten bzw. entspricht einer Mittelung über das Zeitfenster.

Das Ergebnis (COUNT) in beiden Modi wird ohne Totzeit immer sofort in ein Register übernommen und kann jederzeit ohne Fehler ausgelesen werden. Für das Abspeichern in schneller Datenfolge werden die Daten auch am Ausgang BUS weitergegeben und können z.B. in ein Memorymodul (FIFO_HISTO) abgespeichert werden.

Memory Map

,Y'	Read	Write
0	OUT_BUS	IN_TRIGGER
1	COUNT (L)	Time/10ns (L)

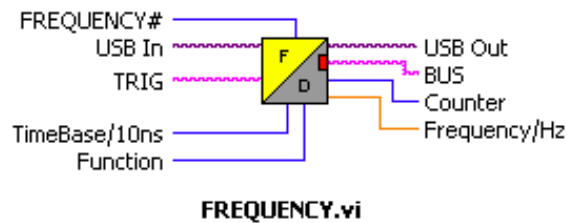
OUT_BUS: Anschlussnummer Ausgang BUS

COUNT: Zählerwert

IN_TRIGGER: setzt Eingang TRIGGER (Zählevents)

Time: Zeitparameter

LabVIEW Interface



Frequency Counter Module with Bus output
 evaluates Trigger frequency in two modes:
 TimeBase=0: Frequency=1/ time between two TRIG signals (low frequency mode!)
 TimeBase=n: Frequency= Count/TimeBase (high frequency mode!)
 and compares to threshold.

INPUTS:

TRIG: input frequency meter (rising edge)

OUTPUTS:

BUS: data strobe

PARAMS:

TimeBase/10ns: TimeBase for high frequency mode
 Gain 2[^]: multiplying factor

OUTPUT:

Frequency/Hz: evaluated frequency of TRIG

Function:

Connect: connects to input
 Set TRIG: change input TRIG
 Get BUS: get state
 Write TimeBase: set timebase in 10ns (32 Bit)
 Read Frequency: get frequency
 Write Gain2[^]: set multiplying factor

FrequencyB#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.21 GATEGEN

GATEGEN besteht aus einem 32b/24b/16b Counter, der mit der Auflösung von 10 ns nach einem TRIGGER (rising edge) einen Puls (PULSE) mit einstellbarer Verzögerung und Dauer erzeugt.

Durch Setzen des Parameters RETRIGGER wird bei einem neuen Trigger während des Pulses der Ablauf komplett neu gestartet. Ansonsten wird dieser Trigger unterdrückt. Ein zusätzlicher ENABLE-Eingang erlaubt die Unterdrückung des Triggers bzw. Pulses. Es ist darauf zu achten, dass der Ausgangspuls gegenüber dem Triggersignal systembedingt mit einem Jitter von 10 ns behaftet ist.

Durch bestimmte Parameter bzw. Belegung von TRIGGER und ENABLE ist eine unterschiedliche Funktionsweise möglich:

CLOCK: TRIGGER = open;

ENABLE = Enable Clock

Pulse: LOW=(Delay+1)*10ns; HIGH=Duration*10ns;

Pulse=100MHz; Delay=0; Duration=0;

COUNTER (simple): Duration=0; ENABLE=open;

Delay=Threshold: wenn Counter \geq Threshold geht PULSE=HIGH

Counter kann immer direkt ausgelesen werden!

COUNTER (enabled): Duration=0;

Delay=Threshold: wenn CounterRegister \geq Threshold geht PULSE=HIGH

Counter zählt nur wenn ENABLE=HIGH

EndOfEnable speichert Counter in Register für Auslese

TRIGGER=open: Counter zählt interne Systemclock (100MHz) (TimeCounter)

Models

G0: 16b Counter

G1: 24b Counter

G2: 32b Counter

Memory Map

,G'	Read	Write
0	OUT_PULSE	IN_TRIGGER
1	Running(0)	IN_ENABLE
2	COUNTER (L,T,W) & Clear	DELAY(L,T,W) & Reset
3		DURATION(L,T,W) & Reset
4		RETRIGGER(0) & Reset

OUT_PULSE: liefert Anschlusswert für PULSE

Running: 1 wenn Delay und Pulse

COUNTER: Wert des internen Zählers

Clear: setzt Zählerregister = 0 (und PULSE=Low)

IN_TRIGGER: setzt TRIGGER Eingang

IN_ENABLE: Signaleingang für TRIGGER

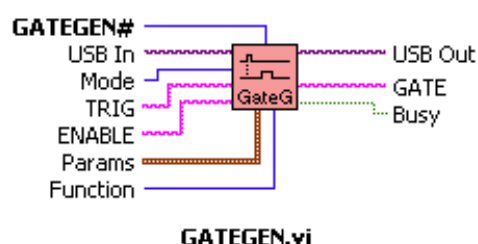
DELAY: Zeit/10ns für Anfangsverzögerung

DURATION: Zeit/10ns für Dauer des Pulses

RETRIGGER: Trigger Modus

Reset: setzt auf Anfangszustand

LabVIEW Interface



Pulse generator (GateGenerator) with programmable Delay and Duration.
Can be also used as Clock and Event & Time Counter!

Modes:

Clock: TRIG=open; PULS: LOW=(Delay+1)*10 ns & HIGH=Duration*10 ns
Delay=0, Duration=0: 100MHz

Counter: Duration=0; Delay=Threshold;
EndOfEnable stores Counter to Register & resets Counter
ENABLE=open: each Count will be stored in Register
TRIG=open: counts with 100MHz (TimeCounter)

Input:

TRIG: Trigger (rising edge) or Gate for Time Counter (see Modes).
ENABLE: enable counting, clock or trigger

Params:

Delay/10ns: delay after trigger (see Modes).
Duration/10ns: duration after delay (see Modes).
Non Retrigger/Retrigger: trigger mode

Busy: state of running

Function:

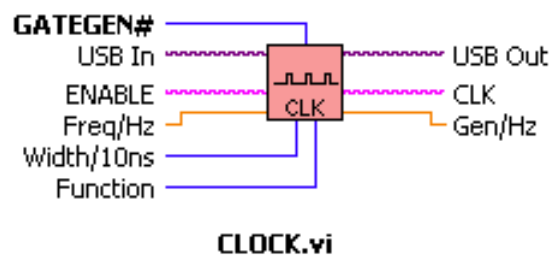
Connect: connects all in/outputs and sets all parameters
Set TRIG: change input TRIG
Set ENABLE: input state
Get PULSE: return output state at PULSE
Write Params: load delay & duration parameter
Read Busy: get Busy state
Read Counter: read current counter state
Write Triggermode & Clear Counter: set Triggermode & resets counter to 0

GATEGEN#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

Zur vereinfachten Anwendung in LabVIEW sind die weiteren Funktionen mit eigenen SubVIs realisiert.

ACHTUNG: Dies sind keine unabhängigen Funktionsmodule sondern diese benutzen GATEGEN!



CLOCK module for generating free running clocks.

All frequencies on output CLK are derived from the system clock (100 MHz) by a divider.

INPUTS:

ENABLE: enable/disable Clock (synchronous)

Params:

Freq/Hz: input of frequency in Hz

Width/10ns: specifies length of High state in 10 ns (=0 generates 50% duty cycle!)

Gen/Hz: output of actual generated frequency in Hz

Function:

Connect: connects all in/outputs and sets all parameters

Get CLK: return output state of CLK

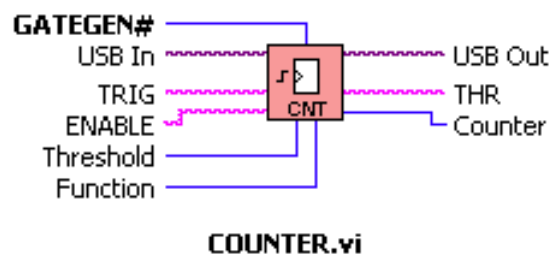
Set ENABLE: input state

Write Params: load frequency and Width parameter

GATEGEN#: number of module (must be unique in group GATEGEN)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!



Counter module with threshold output
also used as Time Counter

INPUTS:

TRIG: counts on rising edge (TRIG=open: counts with 100MHz)
ENABLE: enable counting (ENABLE=open: continue counting)
EndOfEnable stores counter in register & resets counter

OUTPUTS:

THR: goes high on counter \geq Threshold

Functions:

Connect: connects all in/outputs and sets all parameters
Set TRIG: change input TRIG
Set ENABLE: enables counting;
Get THR: get state of counter \geq Threshold
Read Counter: return current Counter Register value; will be cleared on readout
Clear Counter: Set Counter=0

GATEGEN#: number of module (must be unique in group GATEGEN)

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

9.22 LED (SU700, ..)

Auf vielen SU7xxx Modulen sind Leuchtdioden angebracht, die jedoch beliebig für jedes Signal in der Schaltung verwendet werden können. Dabei ist durch eine Stretch-Schaltung mit 10 ms Dauer gewährleistet, dass auch kürzere Pulse deutlich sichtbar angezeigt werden.

Ein DIVIDER kann die Blinkfrequenz entsprechend herunterteilen (M1).

Models

M0: Standard

M1: mit DIVIDER für langsamere Darstellung von Blinkfrequenzen

Memory Map

M0:

,I'	Read	Write
0	0	IN_LED

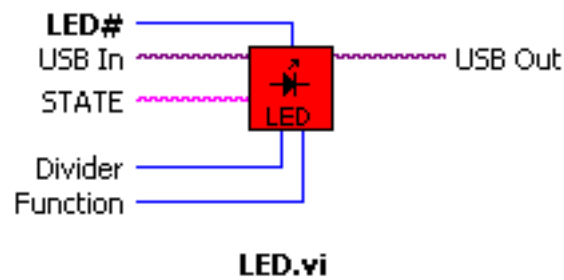
M1:

,I'	Read	Write
0	0	IN_LED
1		DIVIDER (T)

IN_LED: setzt LED Eingang

DIVIDER: Blinkrate wird entsprechend heruntersgesetzt

LabVIEW Interface



Module for LED indicators.

Support: SU700, SU703, SU704, SU706, SU726, ...

Input STATE sets LED accordingly and stretches short pulses (20 ms) for visibility.

V1: supports rate Divider

Function:

Connect: connects to input

Set STATE: change input STATE

Write Divider: $1..2^{24}-1$ (V1) (0= no Divider!)

LED#: number of LED (must be unique).

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.23 LOGIC

Jedes Logicmodul hat drei Eingänge **A**, **B**, **C** sowie einen Ausgang **OUT** und realisiert eine Vielzahl von einfachen Logikverarbeitungen.

Durch einen Parameter **Mode** kann das Verhalten des Moduls im einzelnen spezifiziert werden:

0: **OR** (OUT = A OR B OR C)

1: **AND** (OUT = A AND B AND C)

2: **XOR** (OUT = A XOR B XOR C)

3: FlipFlop

RS-FF:

Eingang A: statisch Set

Eingang B: offen (nicht angeschlossen)

Eingang C: statisch Reset (dominant)

D-FF:

Eingang A: Dateneingang

Eingang B: CLK (steigende Flanke)

Eingang C: statisch Reset (dominant)

4: **MUX** (OUT = A AND C OR B AND NOT C)

Der Parameter **FF** erlaubt das direkte Setzen des internen Registers:

0: FF wird gelöscht

1: FF wird gesetzt

Memory Map

,L'	Read	Write
0	OUT_STATE	IN_A
1		IN_B
2		IN_C
3		Mode= 0 OR 1 AND 2 XOR 3 RSFF/DFF 4 MUX
4		FF(0)

OUT_STATE: liefert Anschlusswert für STATE

IN_A: setzt Eingang A

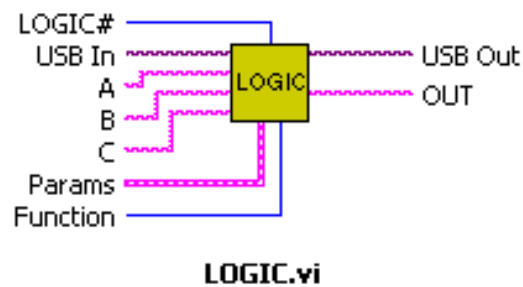
IN_B: setzt Eingang B

IN_C: setzt Eingang C

Mode: setzt Logicmode (siehe Tabelle)

FF: setzt internes FlipFlop

LabVIEW Interface



Basic Logicmodule with 3 inputs (A,B,C) and 1 output (OUT).
Loaded Parameters determine the specific behaviour:
(see specific VIs for more detailed description!)

Params

Mode: OR/AND/XOR/FF

FlipFlop: Internal Register

LOGIC#: number of module (must be unique)

Function:

Connect: connects all in/outputs and sets all parameters

Set A: change input A

Set B: change input B

Set C: change input C

Get OUT: return output OUT

WriteParams: load parameters

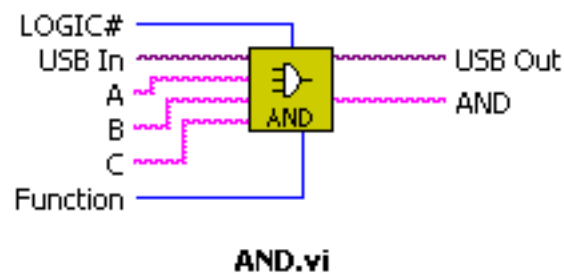
Write FF: load Flip Flop

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

Zur vereinfachten Anwendung in LabVIEW sind die weiteren Funktionen mit eigenen SubVIs realisiert.

ACHTUNG: Dies sind keine unabhängigen Funktionsmodule sondern diese benutzen das Modul LOGIC!



Module with logical AND function.

$AND = A \text{ AND } B \text{ AND } C;$

Unwired inputs are LOW!

LOGIC#: number of module (must be unique in group LOGIC)

Function:

Connect: connects A,B,AND. Open lines are set LOW!

Set A: set input A to LOW, HIGH, OPEN oder SIGNAL.

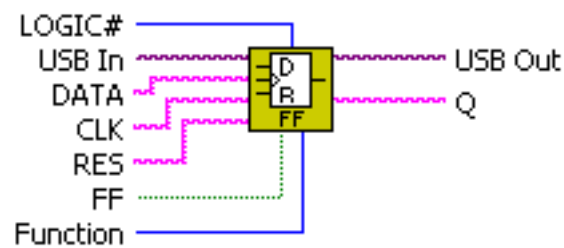
Set B: set input B to LOW, HIGH, OPEN oder SIGNAL.

Set C: set input B to LOW, HIGH, OPEN oder SIGNAL.

Get AND: return current state of output AND.

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!



DFF.vi

Data sensitive flip flop (FF).
 State of DATA will be stored in FF on rising edge of CLK.
 RES resets FF.
 Output Q reflects state of FF.

LOGIC#: number of module (must be unique in group LOGIC)

Function:

Connect: connects all in/outputs and sets all parameters

Set DATA: change input DATA

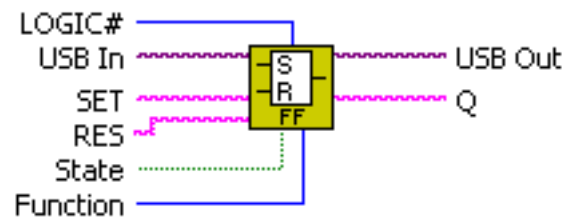
Set CLK: change input CLK

Set RES: change input RES

Get Q: return output Q

Write FF: set FF according to STATE

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

**RSFF.vi**

State sensitive flip flop (FF).
 SET sets FF to true, RES resets to false.
 RESET is dominant.
 Output Q reflects state of FF.

LOGIC#: number of module (must be unique in group LOGIC)

Function:

Connect: connects all in/outputs and sets all parameters

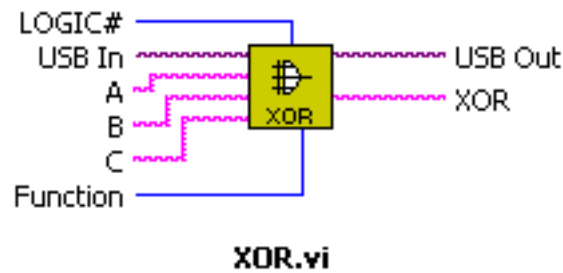
Set SET: change input SET

Set RES: change input RES

Get Q: return output Q

Write FF: set FF according to STATE

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!



Module with logical XOR function.
 $XOR = A XOR B XOR C;$

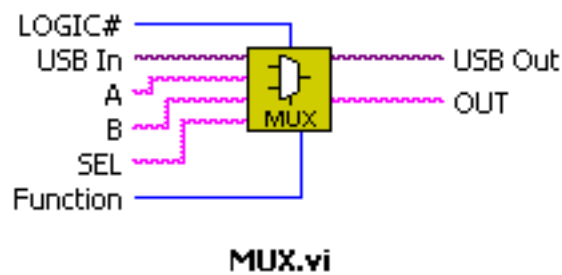
Unwired inputs are LOW!

LOGIC#: number of module (must be unique in group LOGIC)

Function:

Connect: connects A,B,AND. Open lines are set LOW!
 Set A: set input A to LOW, HIGH, OPEN oder SIGNAL.
 Set B: set input B to LOW, HIGH, OPEN oder SIGNAL.
 Set C: set input B to LOW, HIGH, OPEN oder SIGNAL.
 Get AND: return current state of output AND.

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!



Module with logical MUX function.
 $OUT = A$ when $SEL=HIGH$ else B ;

Unwired inputs are LOW!

LOGIC#: number of module (must be unique in group LOGIC)

Function:

Connect: connects A,B,AND. Open lines are set LOW!

Set A: set input A to LOW, HIGH, OPEN oder SIGNAL.

Set B: set input B to LOW, HIGH, OPEN oder SIGNAL.

Set SEL: set MUX selector.

Get OUT: return current state of output.

USB In and USB Out are related to the selected USB interface!
 No connection uses a global parameter, set by OPEN.vi!

9.24 LUT

Zur Linearisierung bestimmter Datenquellen ist eine frei programmierbare Lookup-Tabelle (LUT) sinnvoll.

Die 16b Eingangsdaten **IN_BUS** werden als Adresse für ein Memory (1024*16b) verwendet. Dabei sind entsprechend nur die oberen 10 Bit tatsächlich als Adresse, die einen 16b Ausgangswert an **OUT_BUS** liefern, vorhanden. Die unteren 6 Bit werden hier zur linearen Interpolation verwendet.

$$\text{Value(Input)} = \text{Value(N)} + ((\text{Value(N+1)} - \text{Value(N)}) * \text{LowPart})/2^{**6};$$

Im **CountMode** wird durch einen einfachen Trigger am Eingang **IN_BUS** das Memory sequentiell angesprochen!

Memory Map

,r'	Read	Write
0	OUT_BUS	IN_BUS
1		CountMode(0) & Address=0
2		Memory+ (W)

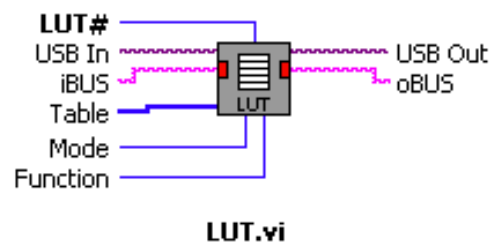
OUT_BUS: anschlusswert für 16b Ausgang BUS

IN_BUS: Eingang BUS

CountMode: sequentieller Modus

Memory+: Speicherwert

LabVIEW Interface



Look Up Table
for Linearisation

INPUTS:
iBUS: 16b

OUTPUTS:
oBUS: 16b

Value(N): MemoryValue for Address N (10b MSB)
P: Address (6b LSB)
Output(Input)= Value(N) + ((Value(N+1) - Value(N)) * P)/2⁶;

PARAMS:
Table: 1024 * 16b Array

Mode:
BUS: oBUS(iBUS)
Counter: next Value on rising edge of BUS Strobe

Function:
Connect: connect module
Set iBUS: set input strobe
Get oBUS: get state of strobe
Write Table: Write Table
Set Mode: BUS or Counter

LUT#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

9.25 MULTIPLICITY

In vielen kernphysikalischen Experimenten muss als Trigger die Koinzidenz (Multiplizität) von mehreren Detektorsignalen bestimmt werden!

Dieses Modul hat 8 Eingängen und eine gleiche Anzahl von Ausgängen, die die entsprechende Multiplizität durch ein logisches Signal anzeigen. Diese Signale können z.B. als Trigger verwendet werden.

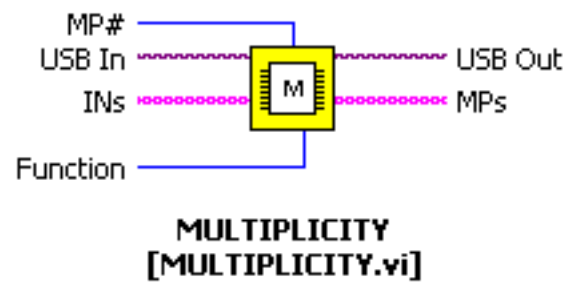
Memory Map

,n'	Read	Write
0	OUT_1	IN_1
..
7	OUT_8	IN_8

OUT_1..8: Ausgänge 1..8 entsprechend n-facher Multiplizität

IN_1..8: Eingänge 1..8

LabVIEW Interface



Module with n inputs and n outputs for according n -fold multiplicity.

M0: $n=8$

M1: $n=4$

Inputs:

INs (1.. n):

Outputs:

MPs (1.. n): for according multiplicity of inputs

Functions:

Connect: connects all in/outputs and sets all parameters

Set INs: change input A..H

Get MPs: return state of multiplicity

MP#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.26 OFFSGAIN

Dieses Modul besitzt einen Dateneingang (iBUS) und einen Datenausgang (oBUS) und skaliert die Daten im Durchfluß über die Parameter OFFSET und GAIN entsprechend.

Dies dient zur Anpassung an weitere Datenmodule wie z.B. das Histogrammer-Modul. Die aktuellen Daten können auch jederzeit direkt ausgelesen werden (DATA).

Parameter:

OFFSET: Dieser Wert wird jeweils zu den Daten addiert (positiv und negativ)

GAIN: Hier handelt es sich im wesentlichen um eine Verschiebung der Daten um entsprechende Bitstellen nach rechts. Dies entspricht jeweils einer Division um 2^{*n} .

Memory Map

,f'	Read	Write
0	OUT_BUS	IN_BUS
1	BUS_Data(L)	Offset(L)
2		Gain

OUT_BUS: Ausgangssignal für Daten

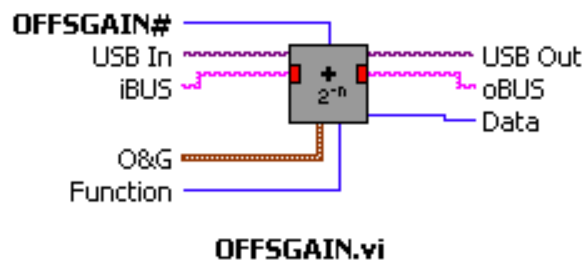
BUS_Data: zum direkten Auslesen der konvertierten Daten

IN_BUS: Eingangssignal für Daten

Offset: addiert zu Eingangsdaten

Gain: verschiebt um entsprechende Stellen nach rechts (Division 2^n)

LabVIEW Interface



Scaling of BUS signals

INPUTS:

iBUS: input data with data strobe

OUTPUTS:

oBUS: output data with data strobe

PARAMS:

Offset: add value to data

Gain 2^{-n} : shift data accordingly to right (divide)
(use in Histo mode to scale down to 0..1024!)

Function:

Connect: connects in&outputs and loads all parameters.

Set iBUS: set data BUS strobe

Get oBus: get state of output BUS

Write Params: load parameters

Read Data: read current oBUS data

OFFSGAIN#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.27 PFIFO (SU730)

Dieses Modul speichert Daten, die von entsprechenden anderen Modulen generiert und ausgegeben werden über den Eingang BUS in einem Speicher ab.

Der Speicher ist mit einem externen Memory-Modul (SU730) mit entsprechender Größe (Pseudo-Static-Memory 4M*32b) realisiert.

Hier gibt es zwei Modi:

FIFO:

Daten werden zeitlich nacheinander in einem Port eingeschrieben und können zeitlich unabhängig von der gleichen Reihenfolge über ein anderes Port wieder ausgelesen werden.

Die Anzahl der im Speicher vorhandenen Datenworte (maximum typ. 1024) kann jederzeit ausgelesen werden (COUNT) und z.B. mit einem Blocktransfer geleert werden.

Solange die mittlere Auslesegeschwindigkeit größer als die Einschreibgeschwindigkeit bleibt, gehen keine Daten verloren. Andernfalls kann es zu einem Überlauf kommen, der in einem Flag (OVFL) signalisiert wird.

Bei einem Überlauf werden die Daten weiter eingeschrieben, alte Daten werden überschrieben. Der Auslesepointer wird dabei mit dem Einschreibpointer mitgesetzt.

HISTO:

Die Daten werden sofort über das Eingangsport entsprechend ihrem Wert in ein Histogramm einsortiert. Das Histogramm kann jederzeit über das Ausleseport komplett ausgelesen werden. Hier gehen keine Daten verloren.

Falls der Eingangswert größer als die Länge des Speichers ist, wird dies mit dem Signal OVFL signalisiert!

Folgende Histogrammseiten in binärer Abstufung können programmiert werden:

64 pages mit 65536 Bins @ 32b ... 65536 pages mit 64 Bins @ 32b

Der Eingang WRITE erlaubt die Unterdrückung bzw. das verzögerte Einschreiben des aktuellen Datenwortes an BUS.

Falls WRITE nicht beschaltet ist, werden alle Daten sofort gespeichert.

Die mittlere Einschreibrate beträgt etwa 10MHz.

Memory Map

,M'	Read	Write
0	0	IN_BUS
1	OVFL(31), COUNT(23..0)	IN_WRITE
2	A_Address(L)	A_Address(L)
3	B_Address(L)	B_Address(L)
4	Data(B_Addr+)(L)	Data(B_Addr+)(L)
5	HistoBits(19..16), HistoPage(15..0)	HistoBits(19..16), HistoPage(15..0)
6		HistoMode(0) & Clear

OVFL: signalisiert Überlauf

COUNT: Anzahl der Daten im FIFO

A_Address: Adresse Port A (High Priority)

B_Address: Adresse Port B (Low Priority)

Data: Datenwort für Port B

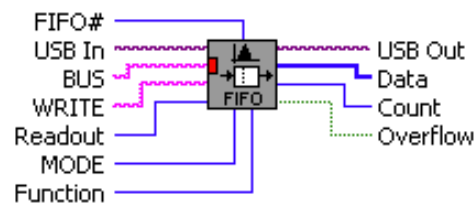
HistoBits: Anzahl der BINs

HistPage: Anzahl der Seiten

HistoMode: 0=FIFO/1=Histogramm

Clear: setzt Addresscounter für FIFO und zum Löschen des Histogramms auf 0

LabVIEW Interface



FIFO.vi

Memory Module with FIFO and HISTOGRAMMER functionality.

INPUTS:

BUS: input for data with data strobe

WRITE: if connected is used for strobing data to memory

Readout: number of data words to be read (0=default: all data will be read)

DATA:

Overflow: FIFO: true if memory full; HISTO: data >1024

Write to FIFO continues on memory Full (read pointer=write pointer)

Count: FIFO: number of data in memory; HISTO: 1024

Data: array of data or histogram

Function:

Connect: connects in&outputs and loads all parameters.

Set BUS: set data BUS strobe

Set WRITE: set WRITE input

Write Memory&Clear:select memory mode, clear FIFO

Read Status: return state of OVERFLOW & COUNT.

Read Data: read FIFO or HISTOGRAM.

Clear HISTO: set Histogram values to 0

FIFO_HISTO#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.28 PID

Diese Modul realisiert einen digitalen Regler mit Proportional-, Integral- und Differentialanteil (PID).

Die Daten von einem Dateneingabemodul (z.B. ADC) werden mit einer bestimmten Datenrate über den Eingang IN_BUS angenommen und daraus die Regelabweichung vom Sollwert (SetValue) bestimmt. Die Regelabweichung wird über den PID-Algorithmus in einen entsprechenden Stellwert (CtrlValue) verrechnet und mit der Eingangstaktrate an den Ausgang OUT_BUS weitergegeben. Dieser Wert kann in einem Datenausgabemodul (z.B. DAC) an die analoge Regelstrecke ausgegeben werden.

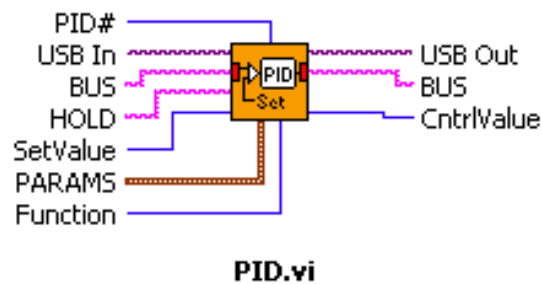
Die Regelparameter (KP, KI, KL) für den Regler können in einem weiten Bereich eingestellt werden.

Ein spezieller Signal-Eingang HOLD erlaubt das Einfrieren des Stellwertes auf den momentanen Wert für beliebig lange Zeit.

Memory Map

,p'	Read	Write
0	OUT_BUS	IN_BUS
1	CntrlValue(W) Integral(L)	IN_HOLD SetValue(W)
2		KP (L)
		KI (L)
		KD (L)
		Clear

LabVIEW Interface



PID Controller Module

Inputs:

BUS: Strobe for 14b Data (IsValue)

Outputs:

BUS: Strobe for 14b Data (CntrlValue)

PARAMS:

KP: proportional factor ($2^{10}=1$)

Ki: integral factor ($2^{10}=1$)

KD: differential factor ($2^{10}=1$)

Functions:

Connect: load initial parameters

Set BUS: set input

Get BUS: get output

Write KP: write KP factor

Write KI: write KI factor

Write KD: write KD factor

Write SetValue: 14b

Read CntrlValue: 14b

Read Integral: read integral component

PID#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.29 PLOGIC

Modul für komplexe Logikfunktionen.

Dabei werden 8 Eingänge über eine LookUp-Tabelle (RAM) auf 8 Ausgänge abgebildet! Das RAM (256 Bytes) kann vom Benutzer beliebig geladen (programmiert) werden.

Memory Map

,P'	Read	Write
0	OUT_1	IN_1
1	OUT_2	IN_2
2	OUT_3	IN_3
3	OUT_4	IN_4
4	OUT_5	IN_5
5	OUT_6	IN_6
6	OUT_7	IN_7
7	OUT_8	IN_8
8		StartAddress
9		RAM (autoinc)

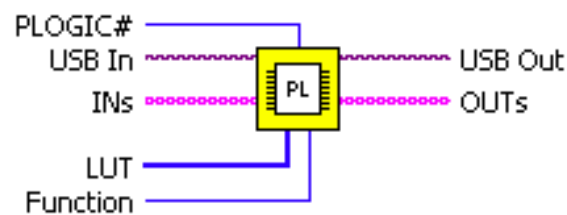
OUT_1..8: Ausgangssignale 1..8

IN_1..8: Eingangssignale 1..8

StartAddress: Adresse für Ladevorgang

RAM: 256 Datenwerte (autoincrement)

LabVIEW Interface



PLOGIC.vi

Logicmodule with 8 inputs (INs) and 8 outputs (OUTs).
 Any passive Logic can be programmed by loading a 256*8 LUT-Memory.
 There is no clock involved!
 Not connected inputs are LOW!

Params

LUT: 256* 8 memory

PLOGIC#: number of module (must be unique)

Function:

Connect: connects all in/outputs and sets all parameters

Set INs: change input 1..8

Get OUTs: return output OUT 1..8

Write LUT: load memory

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.30 PRAM (SU730)

Für die Steuerung von bestimmten Modulen (z.B. Sequenzer) wird ein größeres Memory benötigt. Basierend auf einem pseudostatischen Speicherbaustein (PSRAM) wird hier ein Memorymodul mit 4M*32b realisiert. Die Bandbreite beträgt etwa 80 MB/s.

Durch ein Request-Signal TRIGGER (rising_edge) wird immer ein nächster Wert, entsprechend einem Adresspointer (BUS_Address) ausgelesen und über den Anschluß BUS an ein angeschlossenes Modul weitergegeben. Anschließend wird der Adresspointer um 2 erhöht (Zählweise = 16b Wort!).

Das Memory kann über einen weiteren Port (niedrigere Priorität) jederzeit über einen eigenen Adresspointer (Address) beschrieben und ausgelesen werden.

Memory Map

,m'	Read	Write
0	OUT_BUS	IN_TRIGGER
1	Status	Mode
2	BUS_ADDRESS(T)	BUS_ADDRESS(T)
3	ADDRESS(T)	ADDRESS(T)
4	DATA(ADDRESS+)(L)	DATA(ADDRESS+)(L)

OUT_BUS: Ausgangssignal für die Daten

IN_TRIG: Eingangssignal TRIGGER

Status: (noch nicht unterstützt)

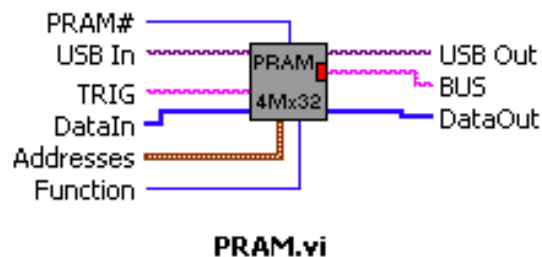
Mode: (noch nicht unterstützt)

BUS_ADDRESS: Adresspointer für BUS Port (16b Wort)

ADDRESS: Adresspointer für DATA Port (16b Wort)

DATA: Daten Port

LabVIEW Interface



Static RAM
Support: SU705

Inputs:
TRIG: request next data on BUS

Outputs:
BUS with 32b data

Parameters:
Address: Addresspointer for data write and read (counting 16b-words!)
BUS_Address: Addresspointer for BUS
DataIn: Data for loading memory
DataOut: readout of memory

Functions:
Connect: set inputs and outputs
Set TRIG: set TRIG state
Get BUS: get BUS state
Write BUS_Address: BUS Addresspointer
Write Address: set Addresspointer
Write Data+: write DATAin (Longword) to Address.
Read Word: read from Address to DATAout (Longword) .

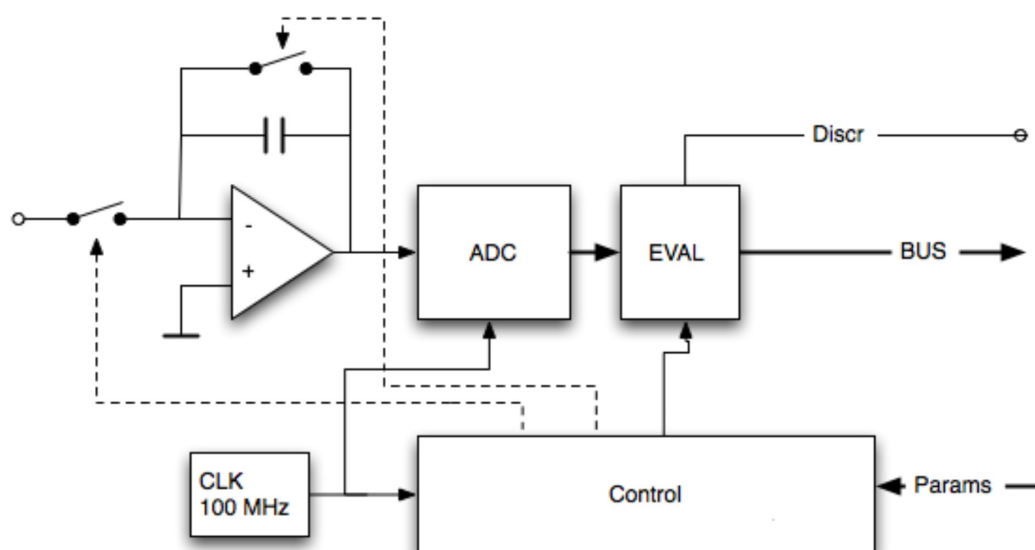
PSRAM#: number of module

USB In and USB Out are related to the selected USB interface!
No connection uses a global parameter, set by OPEN.vi!

9.31 QDC (SU717)

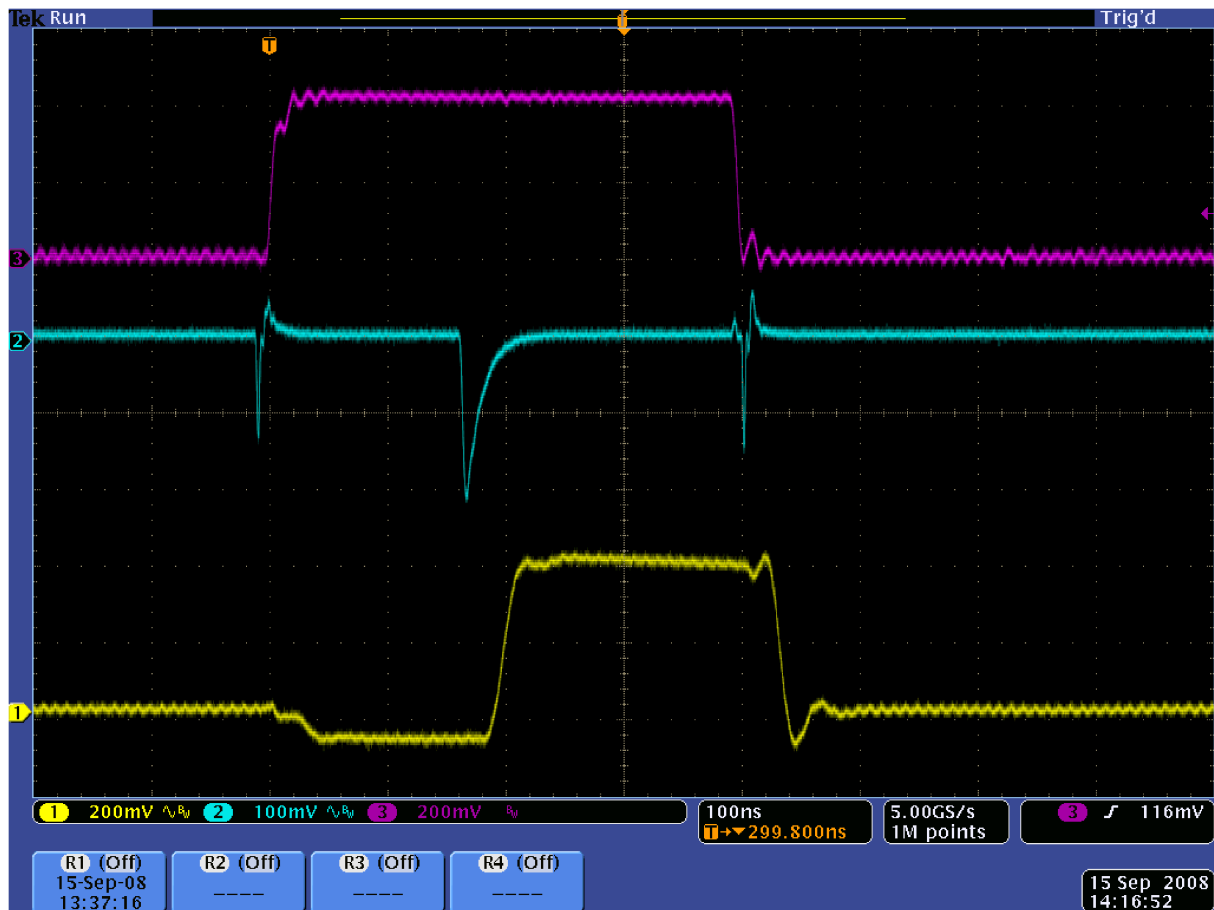
Dieses Modul dient zur Messung von Ladungsimpulsen aus einem Photodetektor.

Blockschaltbild

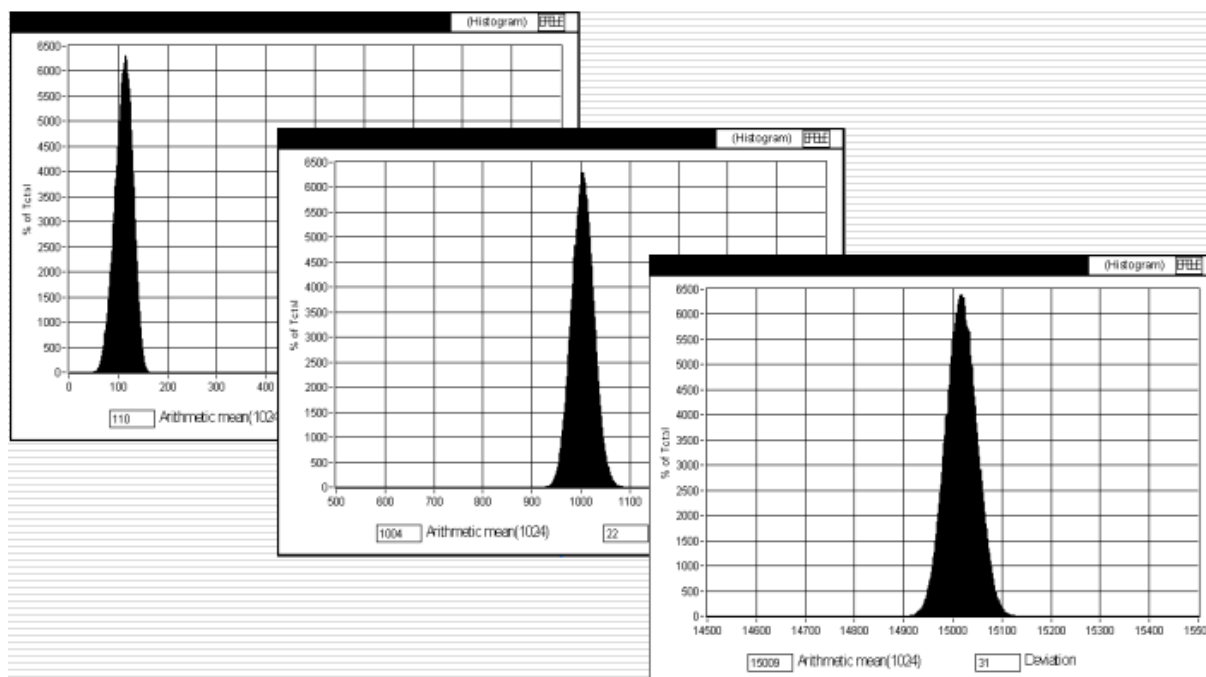


Die Ladungen am Eingang werden über einen geschalteten Integrator aufgesammelt und die Integrationsspannung über einen schnellen ADC (100 MHz) gemessen. Durch mehrmaliges Messen über den typischen Integrationszeitraum von hier 300 ns können in einer nachgeschalteten Verrechnungsschaltung bestimmte Fehler unterdrückt werden.

Die Daten können über einen BUS zu einem nachgeschalteten Memory zur sofortigen Aufzeichnung weitergegeben werden.



Typische Signalformen mit den typischen Fehlern, die durch das Schalten verursacht werden. Der Fehler kann durch zweimaliges Sampling (und z.B. durch Differenzbildung) an verschiedenen programmierbaren Stellen verringert werden.



Memory Map

,Q'	Read	Write
0	OUT_BUS	IN_TRIGGER
1	ADC(W)	Gate(W)
2		StartS(15)&StopS(14)&StartDelay(7..0)
3	(SPI)	(SPI)
4		StopDelay(7..0) & Clear

OUT_BUS: Anschlussnummer für Ausgang BUS

ADC: Datenwert für letzte Messung

IN_TRIGGER: setzt Eingang TRIGGER

Gate: Dauer/10ns für die Integration

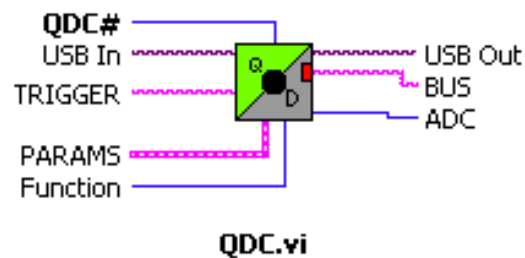
StartS, StopS: Zeitpunkt für ADC Messung bezogen auf Gate

Falls beide Flags gesetzt, wird $ADC(Stop) - ADC(Start)$ berechnet!

StartDelay: Zeitpunkt/10ns nach/vor Start Gate für ADC-Messung

StopDelay: Zeitpunkt/10ns nach/vor Stop Gate für ADC-Messung

LabVIEW Interface



Gated Integrator with ADC and baseline correction.
Support: SU717

Inputs:

TRIGGER: Rising edge starts Gate for integration and data sampling.

Outputs:

BUS: Data & Strobe for ADC

PARAMS:

Gate/10ns: Length of Gate after Trigger in 10 ns

StartDelay: sampling point after start of GATE (incl. ADC-Pipeline=18)

StopDelay: sampling point after end of GATE (incl. ADC-Pipeline=18)

StartData, StopData Flags:

00: StopData-StartData will be recorded (15 bit signed!)

01: Only StopData value recorded

10: Only StartData value recorded

11: StartData & StopData values recorded

Function:

Connect: connects in&outputs and loads all parameters.

Set TRIGGER: change input TRIGGER.

Get BUS: return Strobe

Write Params&Clear: loads parameters, clears FIFO

Read ADC: read data value

QDC#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.32 RAM

Für die Steuerung von bestimmten Modulen (z.B. Sequenzer) wird ein schnelles Memory benötigt. Basierend auf dem internen Speicher im FPGA wird hier ein Memorymodul mit 1024*32b realisiert. Die Bandbreite beträgt max. 400 MB/s.

Durch ein Request-Signal TRIG (rising_edge) wird immer ein nächster Wert, entsprechend einem Adresspointer (BUS_Address) ausgelesen und über den Anschluss BUS an ein angeschlossenes Modul weitergegeben. Anschließend wird der Adresspointer um 1 erhöht (Zählweise 32b Langwort!)

Das Memory kann über einen weiteren Port (niedrigere Priorität) jederzeit über einen eigenen Adresspointer (Address) beschrieben und ausgelesen werden.

Memory Map

,g'	Read	Write
0	OUT_BUS	IN_TRIG
1	Status	Mode
2	BUS_ADDRESS(T)	BUS_ADDRESS(T)
3	ADDRESS(T)	ADDRESS(T)
4	DATA(ADDRESS+)(L)	DATA(ADDRESS+)(L)

OUT_BUS: Anschlusswert von Ausgang BUS

IN_TRIG: setzt Eingang TRIG

Status: (noch nicht verwendet)

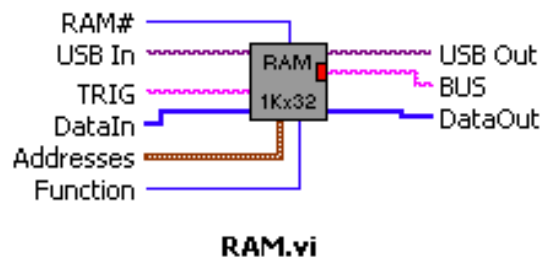
Mode: (noch nicht verwendet)

BUS_ADDRESS: Adresspointer für BUS Port (Zählweise 32b Langwort!)

ADDRESS: Adresspointer für DATA Port (Zählweise 32b Langwort!)

DATA: Daten Port

LabVIEW Interface



FPGA RAM

Inputs:

TRIG: request next data on BUS

Outputs:

BUS with data

Parameters:

Address: Addresspointer for data write and read (counting longwords!)

BUS_Address: Addresspointer for BUS

DataIn: Data for loading memory

DataOut: readout of memory

Functions:

Connect: set inputs and outputs

Set TRIG: set TRIG state

Get BUS: get BUS state

Write BUS_Address: BUS Addresspointer

Write Address: set Addresspointer

Write Data+: write DATAin to Address

Read Word: read from Address to DATAout

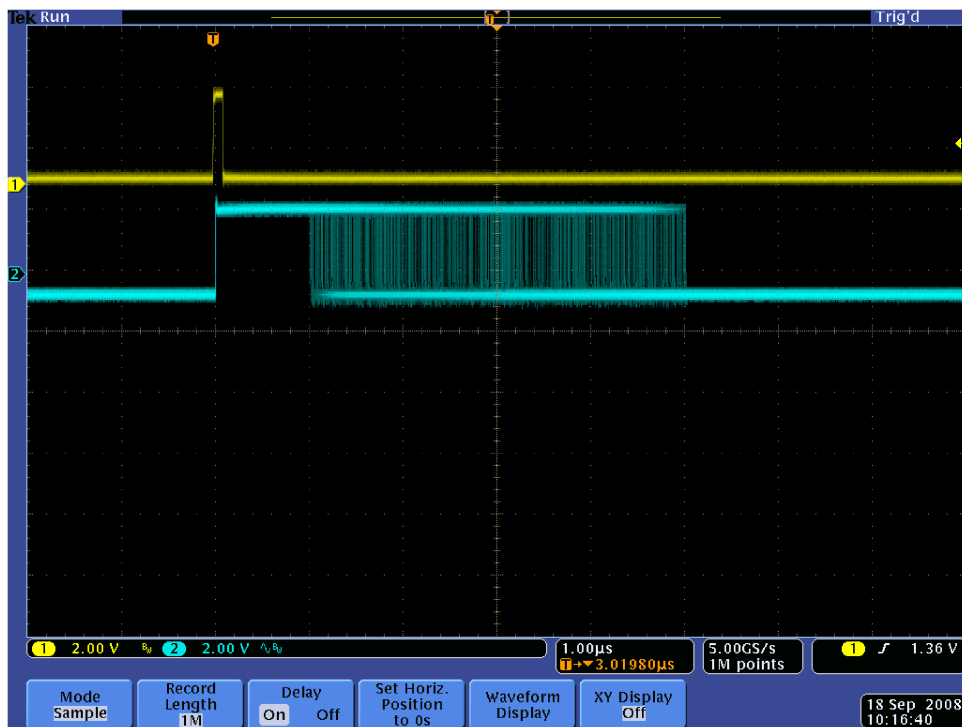
RAM#: number of module

USB In and USB Out are related to the selected USB interface!

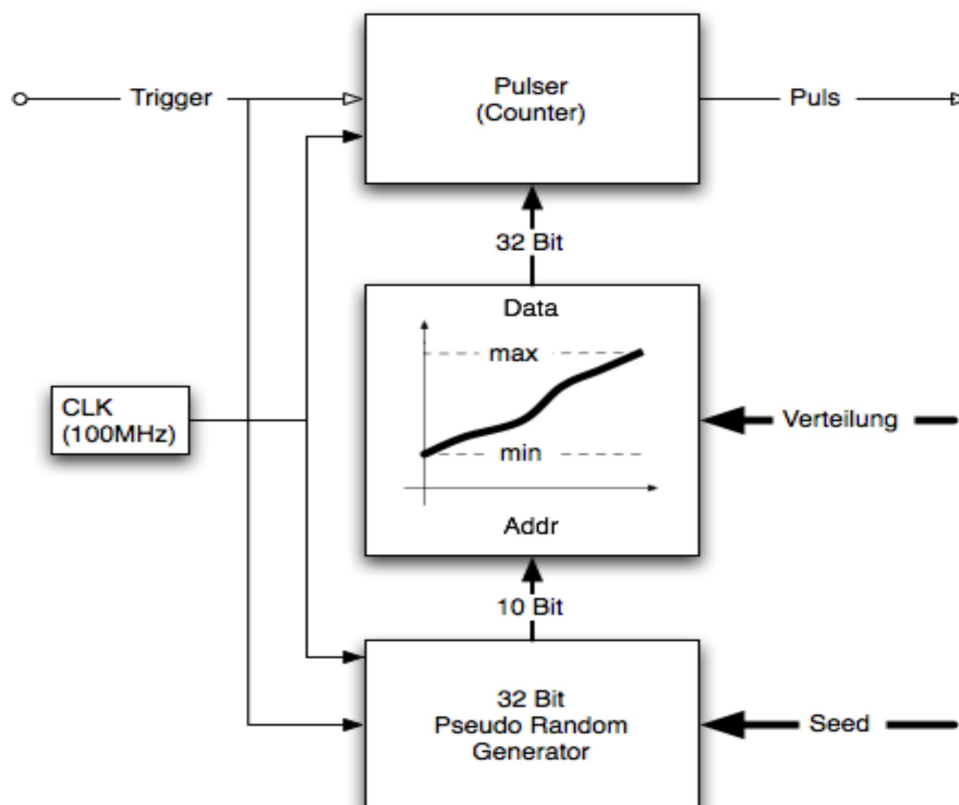
No connection uses a global parameter, set by OPEN.vi!

9.33 RANDOMPULSER

Für das Testen von Auswerteschaltungen an Detektoren ist ein Pulsgenerator mit variabler bzw. zufälliger Pulsdauer vorteilhaft. Die Verteilung der Pulsdauer wird über eine ladbare Verteilungstabelle bestimmt.



Blockschaltbild



Durch den Trigger wird über einen Pseudo-Random-Generator eine zufällige Adresse (10 Bit) generiert, die über ein ladbares Memory die eigentliche Pulsdauer bestimmt. Mit dieser Zahl wird der Pulsgenerator gestartet. Nach Ablauf des Pulses kann beliebig über einen Trigger ein erneuter Puls gestartet werden oder der Pulsausgang wird (invertiert) mit dem Trigger verbunden um einen freilaufenden Pulser mit variabler Zeitdauer zu realisieren.

Memory Map

,U'	Read	Write
0	OUT _ PULSE	IN_TRIGGER
1		Seed(L) (0=Linear)
2		Address(W) = RandomMax
3		Memory(L) (Address+)

OUT_PULSE: anschlusswert für Ausgang PULSE

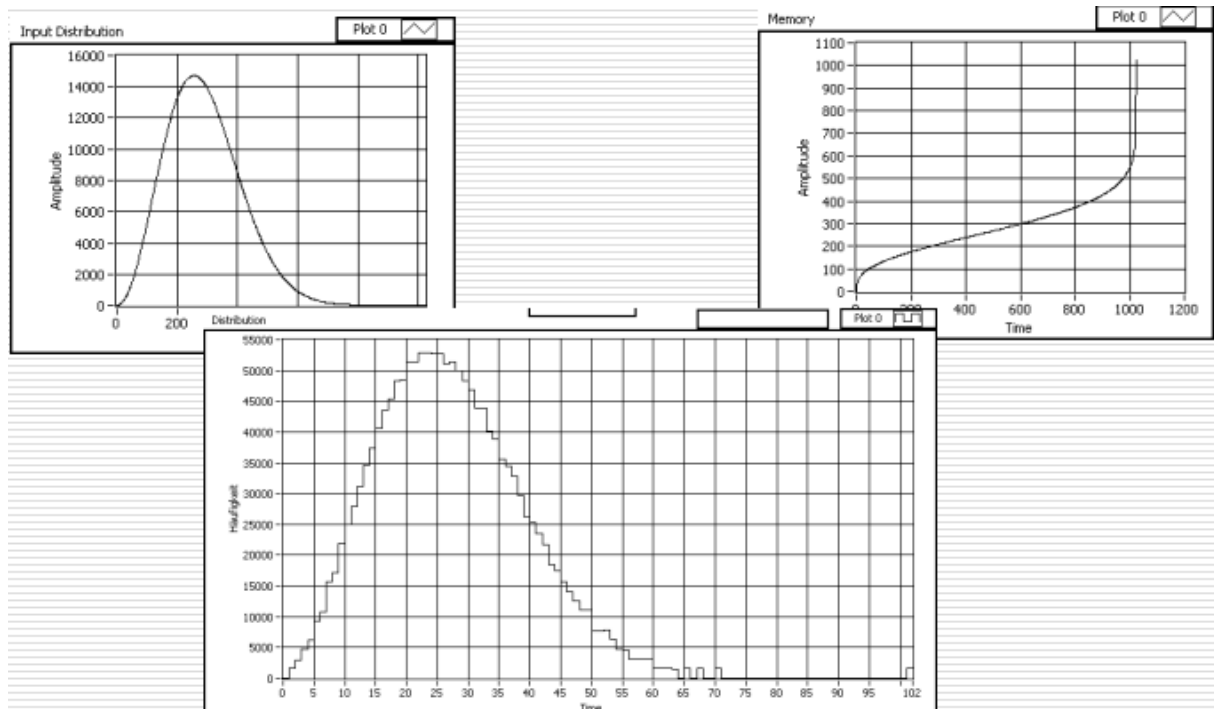
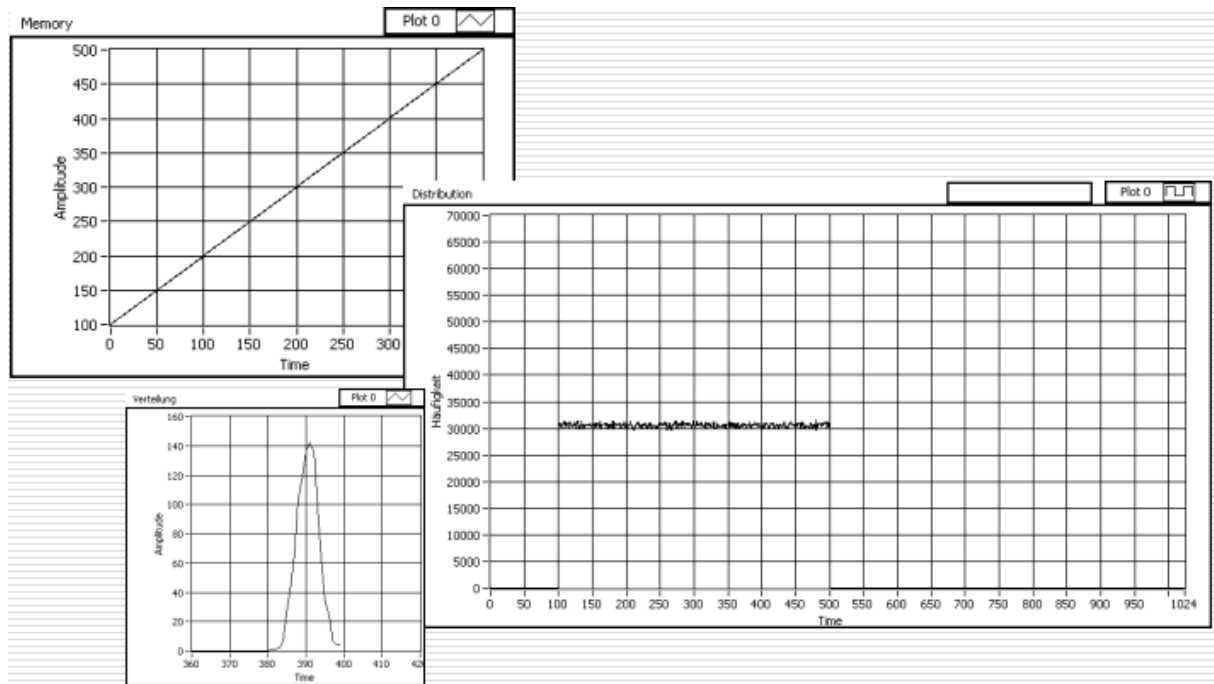
IN_TRIGGER: setzt Eingang TRIGGER

Seed: Anfangswert für Randomgenerator (0 = Linear)

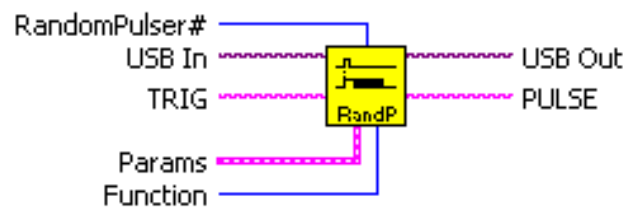
Address: Adresszähler

Memory: Speicher für Verteilung

Beispiele



LabVIEW Interface



RandomPulser.vi

Pulse Generator with random distributed time length.
Distribution of pulse length is determined by Memory data.

Params:

Seed: Random Seed (restarts Random Generator)

Addr: start address for loading DATA.

DATA: data array (14 Bit) for time distribution

Function:

Connect: connects all in/outputs and sets all parameters

Set TRIG: change input TRIG

Get PULSE: return output state PULSE

Write Params&Clear: loads alle parameters and Resets

RandomPulser#: number of module (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.34 SEQUENCER

Dieses Modul erzeugt zeitliche Signale auf 8 Kanälen mit der Auflösung von max. 10 ns über einen Zeitbereich von ca. 40 sec. Die Zeitauflösung und der Zeitbereich sind zusätzlich über einen weiten Bereich durch eine externe Clock und einen Verteiler (32b) veränderbar.

Die Information über den zeitlichen Ablauf sowie der Zustand der Kanäle wird in einem 8b (Kanal) + 32b (Time) Speicher mit 1024 Worten abgespeichert.

Das Eingangssignal TRIG startet den Sequenzer mit der Information im ersten Wort und arbeitet den Speicher nach folgendem Algorithmus bis zu zwei Markern im Memory ab.

Jedes Speicherwort startet mit der Information Time einen Zeitzähler und setzt alle Kanäle (8..1) entsprechend der Bits (7..0) im Byte Kanal. Dieser Zeitzähler wird entweder mit der internen Systemclock (100MHz) oder einer angeschlossenen externen Clock (CLK) getaktet. Zusätzlich kann durch Divider (32b) diese Clock noch um einen beliebigen, ganzzahligen Teiler heruntergeteilt werden. Nach Ablauf der gewünschten Zeit wird das nächste Wort mit den Informationen Kanal + Time verwendet.

Nach dem Start werden die beiden Flags Sequence & Running gesetzt.

Bei Erreichen der Zeitinformation (11...11) wird der Ablauf angehalten, das Bit Running gelöscht und erst bei einem erneuten Trigger die Sequenz fortgesetzt.

Bei Erreichen der Zeitinformation (00...00) wird der Ablauf angehalten, beide Bits Running und Sequence gelöscht und die Sequenz beendet.. Der Speicherindex wird auf 0 gesetzt!

Beispiel

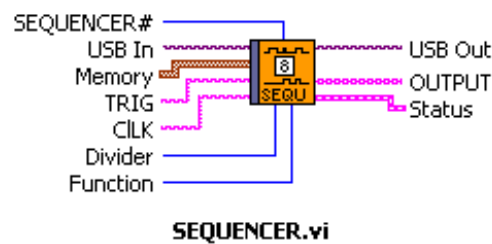
(interne Clock, Divider=1)

Memory	Kanal	Time	Bedeutung
0	255	1000	Bei Trigger werden alle Kanal auf High, gesetzt
1	0	2000	Nach 10us werden alle Kanäle auf Low gesetzt
2	6	10000	Nach 20us wird Kanal 2 u. 3 auf High gesetzt
3	9	500	Nach 100us wird Kanal 1 u. 4 auf High gesetzt, alle anderen Kanäle gehen wieder auf Low
4	8	4294967295 (alles ,1')	Nach 5us bleibt Kanal 4 auf High, 1 geht auf Low Sequenzer hält an und wartet auf neuen Trigger
5	255	3000	Bei Trigger wieder alle Kanal auf High, gesetzt
6	0	0	Nach 30us gehen alle Kanäle auf Low, Die Sequenz ist beendet! Ein neuer Trigger startet wieder bei Memory=0

Memory Map

,S'	Read	Write
0	OUT_1	IN_TRIGGER
1	OUT_2	IN_CLOCK
2	OUT_3	RAM_Addr(W)
3	OUT_4	RAM_Time+(L)
4	OUT_5	RAM_Channel+(B)
5	OUT_6	Divider(L)
6	OUT_7	Pattern(B) & Stop & RAM_Addr=0 & ReadAddr=0
7	OUT_8	Continous(0) & Channels=0
8	Sequence(1),Running(0)	

LabVIEW Interface



8 Channel Digital Sequencer.

Time (32b) and Channel (8b) in a memory array determine the length and the channel pattern for each step after Trigger. In each step the Channel pattern will be loaded and hold then for the according Time.

All channels will be set to '0' after the last time!

Clock is either internal (100 MHz) or external (CLK) and can be divided (Divider) by any number (32b).

Time="0...0": Sequence stops (Sequence=false; Running=false)

Time="1...1": Sequence halts (Sequence=true; Running=false)

Time=1 AND TimeBase=1 is NOT allowed!

Input:

TRIG: rising edge starts sequencer

CLK: external Clock (<40MHz) (internal 100 MHz if not connected)

Output:

OUTPUT: array of 8 channels

Function:

Connect: connects all in and sets all parameters

Set TRIG: set TRIG input

Set CLK: set CLK input (0 uses internal 100MHz)

Get OUTPUT: read states of all channels

Read Status: get running& sequence flags

Write Channels: use Memory(0) channel info for immediate load (clear Continuous mode)

Write Memory: load Time&Channel infos

Write TimeBase: load TimeBase(32b)

Continuous ON: set auto start mode

Continuous OFF: clear auto start mode

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.35 SEQUENCER32

Dieses Modul erzeugt zeitliche Signale auf 8 Kanälen mit der Auflösung von max. 10 ns und über einen Zeitbereich von ca. 40 sec. Die Zeitauflösung und der Zeitbereich sind zusätzlich über einen weiten Bereich durch eine externe Clock und einen Verteiler (32b) veränderbar.

Die Information über den zeitlichen Ablauf sowie der Zustand der Kanäle wird in einem 32b (Kanal) + 32b (Time) Speicher mit 1024 Worten abgespeichert.

Das Eingangssignal TRIG startet den Sequenzer mit der Information im ersten Wort und arbeitet den Speicher nach folgendem Algorithmus bis zu zwei Markern im Memory ab.

Jedes Speicherwort startet mit der Information Time einen Zeitzähler und setzt alle Kanäle (32..1) entsprechend der Bits (31..0) im Byte Kanal. Dieser Zeitzähler wird entweder mit der internen Systemclock (100MHz) oder einer angeschlossenen externen Clock (CLK) getaktet. Zusätzlich kann durch Divider (32b) diese Clock noch um einen beliebigen, ganzzahligen Teiler heruntergeteilt werden. Nach Ablauf der gewünschten Zeit wird das nächste Wort mit den Informationen Kanal + Time verwendet.

Nach dem Start werden die beiden Flags Sequence & Running gesetzt.

Bei Erreichen der Zeitinformation 4294967295 (alles ,1') wird der Ablauf angehalten, das Bit Running gelöscht und erst bei einem erneuten Trigger die Sequenz fortgesetzt.

Bei Erreichen der Zeitinformation 0 (alles ,0') wird der Ablauf angehalten, beide Bits Running und Sequence gelöscht und die Sequenz beendet.. Der Speicherindex wird auf 0 gesetzt!

Beispiel

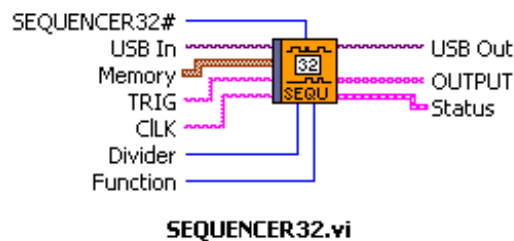
(interne Clock, Divider=1)

Memory	Kanal	Time	Bedeutung
0	255	1000	Bei Trigger werden alle Kanal auf High, gesetzt
1	0	2000	Nach 10us werden alle Kanäle auf Low gesetzt
2	6	10000	Nach 20us wird Kanal 2 u. 3 auf High gesetzt
3	9	500	Nach 100us wird Kanal 1 u. 4 auf High gesetzt, alle anderen Kanäle gehen wieder auf Low
4	8	4294967295	Nach 5us bleibt Kanal 4 auf High, 1 geht auf Low Sequenzer hält an und wartet auf neuen Trigger
5	255	3000	Bei Trigger wieder alle Kanal auf High, gesetzt
6	0	0	Nach 30us gehen alle Kanäle auf Low, Die Sequenz ist beendet! Ein neuer Trigger startet wieder bei Memory=0

Memory Map

,s'	Read	Write
0	OUT_1	IN_TRIGGER
1	OUT_2	IN_CLOCK
2	OUT_3	RAM_Addr(W)
3	OUT_4	RAM_Time+(L)
4	OUT_5	RAM_Channel+(B)
5	OUT_6	Divider(L)
6	OUT_7	Pattern(B) & Stop & RAM_Addr=0 & ReadAddr=0
7	OUT_8	Continous(0) & Channels=0
7	OUT_9	
7	OUT_10	
7	OUT_11	
7	OUT_12	
7	OUT_13	
7	OUT_14	
7	OUT_15	
7	OUT_16	
7	OUT_17	
7	OUT_18	
7	OUT_19	
7	OUT_20	
7	OUT_21	
7	OUT_22	
7	OUT_23	
7	OUT_24	
7	OUT_25	
7	OUT_26	
7	OUT_27	
7	OUT_28	
7	OUT_29	
7	OUT_30	
7	OUT_31	
7	OUT_32	
8	Sequence(1),Running(0)	

LabVIEW Interface



32 Channel Digital Sequencer.

Time (32b) and Channel (32b) in a memory array determine the length and the channel pattern for each step after Trigger. In each step the Channel pattern will be loaded and hold then for the according Time.

All channels will be set to '0' after the last time!

Clock is either internal (100 MHz) or external (CLK) and can be divided (Divider) by any number (32b).

Time="0...0": Sequence stops (Sequence=false; Running=false)

Time="1...1": Sequence halts (Sequence=true; Running=false)

Time=1 AND TimeBase=1 is NOT allowed!

Input:

TRIG: rising edge starts sequencer

CLK: external Clock (>40MHz) (internal 100 MHz if not connected)

Output:

OUTPUT: array of 32 channels

Function:

Connect: connects all in and sets all parameters

Set TRIG: set TRIG input

Set CLK: set CLK input (0 uses internal 100MHz)

Get OUTPUT: read states of all channels

Read Status: get running & sequence flags

Write Channels: use Memory(0) channel info for immediate load (clear Continuous mode)

Write Memory: load Time&Channel infos

Write TimeBase: load TimeBase(32b)

Continuous ON: set auto start mode

Continuous OFF: reset auto start mode

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.36 SETUP

Dieses spezielle Modul (,§'=\$A6) dient zur Auslese von Informationen bezüglich der Firmware (Konfiguration, Version), der verwendeten Basiskarte und der verwendeten Subkarten.

ACHTUNG: Das USB-Kommando # liefert ab Vers. 3.x das Datum & Uhrzeit (32b) der Erstellung der Firmware!

Memory Map

,§'	Read	Write
0	LP(W) & MajorVers(B) & SubVers(B)	
1	DL-Basiskarte	
2	SU-Karte auf M0	
3	SU-Karte auf M1	
4	SU-Karte auf M2	
5	SU-Karte auf M3	
6	SU-Karte auf M4	
7	SU-Karte auf M5	
8	SU-Karte auf M6	
9	SU-Karte auf M7	

LP: LogicPool Firmware Konfiguration (z.B. „1“); MajVers.SubVers: (z.B. „4.0“)

DL-Basiskarte: ..FF = unknown; n = DL7n

SU-Karte: ..FF = empty; n = SU7n

LabVIEW Interface



SETUP.vi

List Setup information:

- System
- Hardware Modules
- Firmware Modules

Dieses Modul verwendet die Firmware-Informationen aus den Modulen SETUP und den weiteren Funktionsmodulen um in Verbindung mit dem SubVI „LogicPool“ diese in Textform auszugeben!

9.37 SFIFO (SU705, SU716)

NOCH NICHT IMPLEMENTIERT!

Dieses Modul basiert auf statischen Speicher-Modulen und entspricht FIFO bzw. PFIFO!

Memory Map

,'	Read	Write
0	0	IN_BUS
1	OVFL(15), COUNT(10..0)	IN_WRITE
2	MEMORY(W,L)	MEMORY(W,L)
3		HISTO(0) & Clear

OVFL: signalisiert Überlauf

COUNT: Anzahl der Daten im FIFO (HISTO = 1024)

MEMORY: Auslesen der Daten

IN_BUS: setzt Eingang BUS

IN_WRITE: setzt Eingang WRITE

MEMORY: zum Löschen des Histogramms

HISTO: Modus Histogramm (1) / FIFO (0)

Clear: setzt Addresscounter für FIFO und zum Löschen des Histogramms auf 0

9.38 SU

Zur direkten Ansteuerung aller Pins (Pin 3..34) der Submodul-Stecker (P0..P3) ist dieses Modul verwendbar und somit insbesondere für Tests nützlich.

Alle Pins können immer über **Inputs** gelesen werden und über **Outputs** gesetzt werden!

In jedem einzelnen Fall wird über das entsprechende Bit in **Enable** der Ausgangstreiber eingeschaltet oder der Eingang von außen ermöglicht.

Memory Map

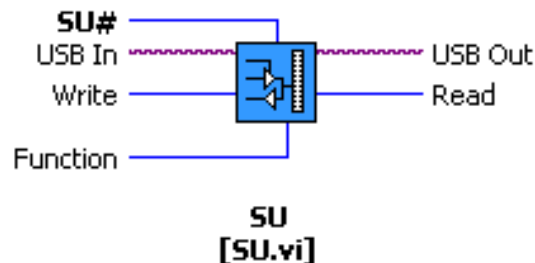
,X'	Read	Write
0	0	Enable 34..3(L)
1	Inputs 34..3(L)	Outputs 34..3(L)

Inputs: Liefert Eingangswerte für jeden Pin

Enable: Schaltet entsprechende Pins auf Ausgang

Outputs: Setzt Ausgangswerte für jeden Pin

LabVIEW Interface



Submodule Pin Control.

Support: DL701, DL705, DL706, DL707, DL708

Input Module selects on of the four SU connectors (SU0..SU3) on the LogicBox and allows to controll all Pins (3..34) as output with enable and input.

Note that this function has to be configured in firmware accordingly!

Values can be set with 32 Bits in terminal "Write" and read back in terminal "Read" where Bits 0..31 correspond directly to the Pin numbers 3..34.

Function:

Read: return state of all Pins.

Write: set state of all Pins.

Enable: drive all Pins (use as output, otherwise as input!).

SU#: number of SU connector (S#=M#+1; 1..4)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.39 SYNC

Dieses Modul dient zur Signalanpassung eines digitalen Eingangssignals und besitzt verschiedene Modi:

Differentiator:

Die steigende Flanke des Eingangssignals erzeugt einen kurzen Puls von 20ns, der synchron von der internen Systemclock abgeleitet wird.

Differentiator (Start Async):

Die steigende Flanke des Eingangssignals erzeugt einen kurzen Puls von mindestens 20ns. Der Beginn des Pulses wird direkt (asynchron) von der steigenden Flanke des Eingangssignals abgeleitet. Das Ende des Pulses ist synchron zur internen Systemclock.

Synchronizer:

Die Dauer (Beginn und Ende) des Eingangssignals wird mit der internen Systemclock synchronisiert.

Synchronizer (Start Async):

Die Dauer (Ende) des Eingangssignals wird mit der internen Systemclock synchronisiert. Der Beginn des Pulses wird direkt (asynchron) von der steigenden Flanke des Eingangssignals abgeleitet.

Debouncer/10ns:

Entsprechend dem Parameter DEBOUNCE, werden kürzere Pulse als $n \cdot 10\text{ns}$ unterdrückt.

Debouncer/1ms:

Entsprechend dem Parameter DEBOUNCE, werden kürzere Pulse als $n \cdot 1\text{ms}$ unterdrückt.

Memory Map

,I'	Read	Write
0	OUT_N	IN_A
1		Mode= 0 Differentiator 1 Differentiator (Start Async.) 2 Synchronizer 3 Synchronizer (Start Async.) 4 Debouncer/10ns 5 Debouncer/1ms
2		Debounce

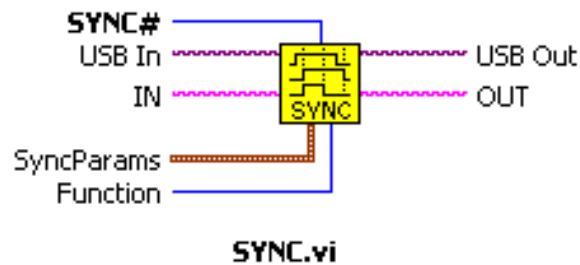
OUT_N: Anschlusswert für Ausgang N

IN_A: setzt Eingang A

Mode: Funktionsmodus (s.o.)

Debounce: Wert für Entprellung

LabVIEW Interface



Synchronizer and Debouncer MODULE

Params

Mode: SS-START/AS-START/SS-PULSE/AS-PULSE/DEBOUNCE10ns,1ms

Debounce: time value in 10ns or 1ms

SYNC#: number of module (must be unique)

Function:

Connect: connects all in/outputs and sets all parameters

Set IN: change input IN

Get OUT: return output OUT

WriteParams: load parameters

WriteDebounce: load debounce time

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.40 TDC

Über das TDC-Modul werden Zeitinformationen von den Eingängen GATE und CHANNEL während der Dauer von GATE bzw. GATETIME über einen beliebig langen Zeitraum ermittelt. Dabei können Signale (rising edge) mehrfach und gleichzeitig auf unterschiedlichen Kanälen auftreten. Ereignisse werden als Event-Marker mit Kanal und Zeitinformation erfasst.

Die Zeitauflösung ist entweder durch eine externen CLOCK gegeben oder nutzt die interne Systemclock von 100 MHz. Diese Zeitbasis kann jeweils zusätzlich durch den Parameter DIVIDER in einem weiten Maße verlängert werden.

Die Länge der Zeitmessung ist abhängig von der Breite des Zeitzählers (typ. 32/24 Bit) und wird ggf. durch die Aufzeichnung eines OVERRUN-Markers beliebig verlängert.

Alle Events werden nur temporär für die direkte Auslese gespeichert bzw. können über den BUS auf ein externes Memory (FIFO/HISTOGRAMMER) weitergeleitet werden!

Models

M0: 32b Time; 1 Channel, Multihit

M1: 24b Time, 8 Channel, Multihit/Multichannel

Start Of Gate, End Of Gate

Es ist zu beachten, dass SOG und EOG nicht als solche markiert werden, sondern lediglich als Zeitinformation sichtbar sind. Es kann deshalb vorkommen, dass zu diesen Zeitpunkten auch ein Channel-Event auftritt und diese beiden Ereignisse nicht unterschieden werden können!

Um Zweideutigkeiten zu vermeiden kann die Erfassung von SOG und EOG abgeschaltet werden!

Der TDC kann in zwei Modis betrieben werden:

GATE Mode

Dieser Modus ist mit GATETIME=0 ausgewählt!

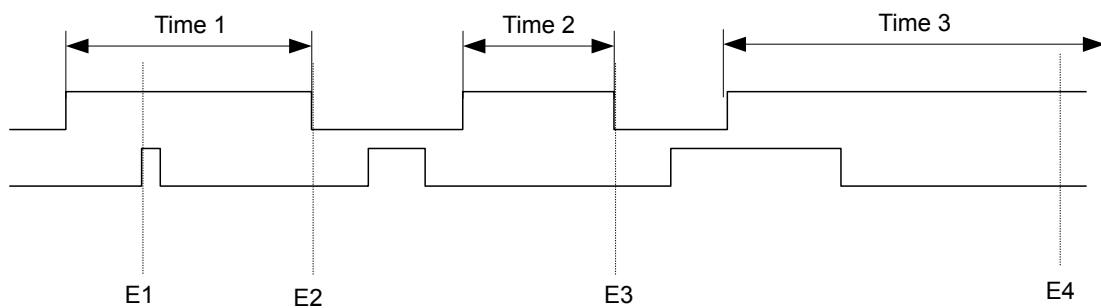
In diesem Fall wird die Ermittlung der Zeitevents nur während GATE = HIGH durchgeführt.
Auslösende Events können sein:

Channels: rising edge während GATE

EndOfGate: falls Flag EOG=0

Overrun: falls GATE > Zeitzähler

Das folgende Bild zeigt einen typischen Verlauf für den GATE Mode:



Event	Ch n (31..24)	Time (23..0)	Bemerkung
E1	10000001	Time(Time1)	Channel 1 & 8
E2	00000000	Time(Time1)	EndOfGate
E3	00000000	Time(Time2)	EndOfGate
E4	00000000	(...11111111)	Overrun

TRIGGER Mode

Dieser Modus ist mit $GATETIME > 0$ ausgewählt!

In diesem Fall wird die Ermittlung der Zeitevents mit der steigenden Flanke an GATE (Trigger) gestartet und mit dem Ablauf von GATETIME gestoppt. Während dieser Zeit kann der Aufzeichnungsvorgang mit einem Trigger neu gestartet werden, wenn das Flag RETRIGGER gesetzt ist!

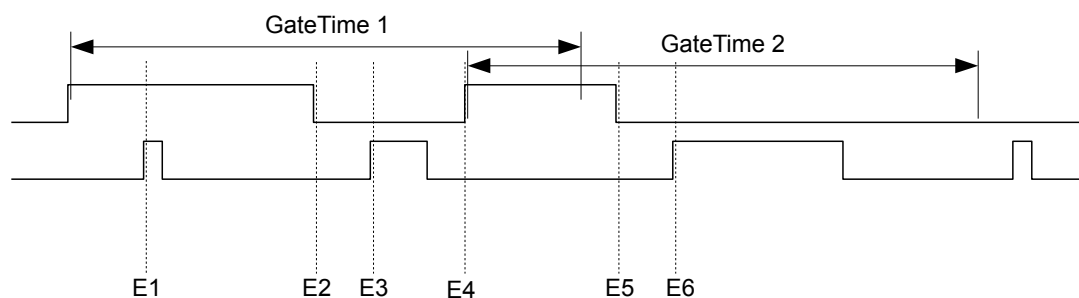
Auslösende Events können sein:

Channels: rising edge während GATETIME

EndOfGate: falls Flag EOG=0

StartOfGate: falls Flag SOG=0; im RETRIGGER Fall!

Das folgende Bild zeigt einen typischen Verlauf für den TRIGGER Mode mit Retrigger:



Event	Ch n (31..24)	Time (23..0)	Bemerkung
E1	00000001	Time(GateTime1)	Channel 1
E2	00000000	Time(GateTime1)	EndOfGate
E3	00010100	Time(GateTime1)	Channel 3 & 5
E4	00000000	Time(GateTime1)	StartOfGate (within GateTime1)
E5	00000000	Time(GateTime2)	EndOfGate
E6	00000010	Time(GateTime2)	Channel 2

Memory Map

M0:

,B'	Read	Write
0	OUT_BUS	IN_GATE
1	EVENT(L)	IN_CLOCK
2		DIVIDER (L)
3		IN_Ch1
11		Retrigger(2) & DisableSOG(1) & DisableEOG(0)
12		GateTime (L)

M1:

,B'	Read	Write
0	OUT_BUS	IN_GATE
1	EVENT(L)	IN_CLOCK
2		DIVIDER (L)
3		IN_Ch1
..		...
10		IN_Ch8
11		Retrigger(2) & DisableSOG(1) & DisableEOG(0)
12		GateTime (L)

OUT_BUS: Anschlussnummer für Strobe und Datenbus

EVENT: Wert des letzten (!) Events.

IN_GATE Setzt Anschlussnummer für GATE

IN_CLOCK Setzt Anschlussnummer für externe CLOCK.

Wenn nicht geladen (<>0) wird interne 100 MHz Clock verwendet

DIVIDER: Bestimmt die Zeitbasis von 10ns*n bzw. CLOCK/n:

IN_Ch1: Setzt Anschlussnummer für Kanal 1

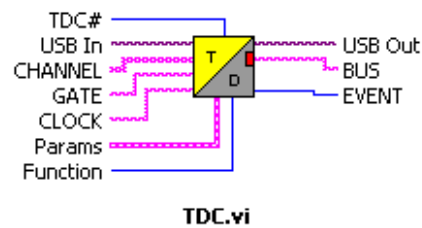
IN_Ch 2..8: Setzt Anschlussnummer für Kanäle 2..8

DisableEOG: Unterdrückung des EndOfGate Timemarkers (Vx.3)

DisableSOG: Unterdrückung des StartOfGate Timemarkers (Vx.5)

GateTime: Aufzeichnungszeit im Trigger-Mode (Vx.5)

LabVIEW Interface



TDC Module.

Time will be recorded in Time Markers with $\text{DIVIDER} * 10\text{ns}$ resolution.

Timebase can also be $\text{CLOCK}/\text{DIVIDER}$ if connected!

EVENTS can be read out after each last event or streamed via BUS to a memory module.

TDC can run in two modes:

GATED: ($\text{GATE TIME}=0$) GATE starts and stops recording;

TRIGGERED: ($\text{GATE TIME}>0$) Rising Edge of GATE starts recording until GATE TIME
if enabled can be retriggered during recording with restart of time counter

SOG and EOG flags determine whether StartOfGate and/or EndOfGate produces time markers.

V0: & V1: 1 Channel, Multihit TDC

Whenever there is a rising edge on CHANNEL (1 Array input) the current time will be recorded and streamed to BUS.
Data stays on bus until next measurement.

V1: 1..8 Channel, Multichannel/Multihit TDC

Whenever there is a rising edge on any signal in CHANNEL (8 Array input) the current time and channel info will be recorded and streamed to BUS.

Every time overflow will be recorded as well (as an 'empty' channel marker) and allows to record further on.

Event is Channel(31..24) & Time(23..0).

Function:

Connect: connects all in/outputs and sets all parameters

Set GATE: change input GATE

Set CHANNEL: change input CHANNEL (set inputs to '0' if not used)

Set CLOCK: change input CLOCK

Get BUS: get BUS state

Read Event: read Ch&Time and clear (data=0)!

Write PARAMS: write all parameters

TDC#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.41 TEMPERATURE

Der Temperatursensor SMT16030 liefert ein digitales Ausgangssignal bei dem der Temperaturwert im Tastverhältnis des Pulses codiert ist.

$$\text{duty cycle} = 0.320 + 0.00470 * t \quad (t = \text{temperature in } ^\circ\text{C})$$

Dieses Tastverhältnis an **IN_TEMP** wird mit jeweils 16b Genauigkeit ausgezählt, in **High- & Low_Time** ausgegeben und kann zur Ermittlung der Temperatur verwendet werden.

Memory Map

,t'	Read	Write
0	0	IN_TEMP
1	High(31..16) & Low(15..0)	

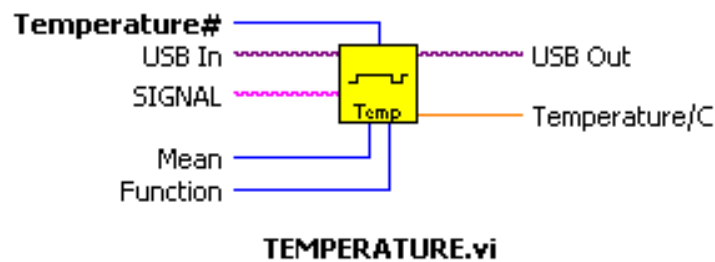
High & Low: Zeit/10ns

IN_TEMP: Eingang TEMP (Temperatursensor)

Siehe Datenblatt:

<http://www.smartec.nl/pdf/DSSMT16030.PDF>

LabVIEW Interface



Temperature Module
evaluates temperature from digital input (SMT16030)

INPUTS:

SIGNAL: input frequency from sensor

PARAMS:

Mean: samples multifold and calculates mean

OUTPUT:

Temperature/C: Celsius degree

Function:

Connect: connects to input

Set SIGNAL: change input SIGNAL

Read Temperature: get temperature value (acc. Mean)

Temperature#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.42 TOSLINK (SU724, SU727)

Für die Übertragung von Daten über eine serielle Leitung kann das Puls-Shift-Verfahren (BiPhase) verwendet werden. Hierbei wird z.B. für das Datenbit ‚1‘ ein Signalübergang von ‚0‘ auf ‚1‘ und für das Datenbit ‚0‘ ein Signalübergang von ‚1‘ auf ‚0‘ (180 Grad Phasenverschoben) festgelegt werden. Da die Abtastfrequenz mit übertragen wird, lässt sich diese beim Empfänger leicht rekonstruieren.

Das Modul codiert entsprechend einen Datenstrom auf **OUT_SEND** nach Einschreiben eines 24b Datenwortes auf **DataSend**. Dabei bestimmt der Parameter **SendSpeed** die Geschwindigkeit (z.B. 9 = 5 MHz).

Ein empfangener Datenstrom auf **IN_RECV** steht nach Setzen des **RecvFlag** im Register **DataRecv** zur Auslese bereit. Die Empfangsgeschwindigkeit wird mit **RecvSpeed** (z.B. 16 = 5 MHz) eingestellt.

Falls Daten nicht rechtzeitig vor Eintreffen neuer Daten ausgelesen werden, tritt ein Overflow auf.

Memory Map

‚j‘	Read	Write
0	OUT_SEND	IN_RECV
1	Overflow(1) & RecvFlag(0)	RecvSpeed(W) & SendSpeed (W)
2	DataRecv (T)	DataSend (T)

OUT_Send: Anschlußwert Ausgang SEND

Overflow: Datenüberlauf

RecvFlag: Neue Daten verfügbar

DataRecv: empfangene Daten

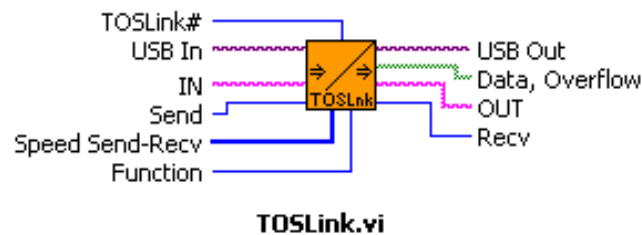
IN_RECV: Eingang RECV

RecvSpeed: typ 16 (5MHz)

SendSpeed: typ 9 (5MHz)

DataSend: zu sendende Daten

LabVIEW Interface



TOSLink Module (TOSLink)
sends and receives data in BiPhase code (24 Bit)

INPUTS:

IN: digital input

OUTPUTS:

OUT: digital output

PARAMS:

Send: data to be sent

Recv: data received

Speed (send, receive): divider for system clock ($n \cdot 10\text{ns}$)

Data: signals valid data received (cleared on read)

Overflow: signals overflow on received (cleared on read)

Function:

Connect: connects to input

Set IN: change input IN

Get OUT: read OUT state

Read Status: read DataRdy

Read Data: read input

Write Data: write & send to output

Write Speed: set speed for Send & Receive

TOSLink#: number of module (must be unique).

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

9.43 WINDOW

Dieses Modul besitzt eine Dateneingang BUS und setzt entsprechend den Signalausgang WITHIN auf High wenn die Bedingung

$$\text{LOWER} \leq \text{Eingangsdaten} < \text{UPPER}$$

erfüllt ist.

Memory Map

,d'	Read	Write
0	OUT_WITHIN	IN_BUS
1		LOWER (L)
2		UPPER (L)

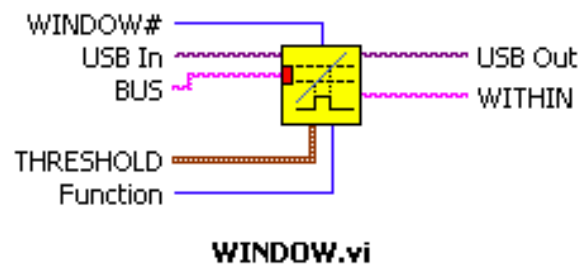
OUT_WITHIN: Anschlußwert für Ausgang WITHIN

IN_BUS: setzt Dateneingang BUS

LOWER: Untere Datengrenze

UPPER: Obere Datengrenze

LabVIEW Interface



Detect BUS data within given Window

INPUTS:

BUS: input data with data strobe

OUTPUTS:

WITHIN: true if data is $\text{LOWER} \leq \text{BUS} < \text{UPPER THRESHOLD}$

THRESHOLD:

UPPER: upper limit for window

LOWER: lower limit for window

Function:

Connect: connects in&outputs and loads all parameters.

Set BUS: set data BUS strobe

Get WITHIN: get state of output signal

Write THRESHOLD: loads parameters

WINDOW#: module number (must be unique)

USB In and USB Out are related to the selected USB interface!

No connection uses a global parameter, set by OPEN.vi!

10 Version History

Grundsätzlich gilt, dass Kompatibilität innerhalb der Hauptversionsnummer (Vn.x) gewährleistet ist. Unterversionsnummern (Vx.n) kennzeichnen Erweiterungen oder Fehlerberichtigungen.

V1.x

Erster Prototyp

Module: LOGIC, COUNTER, DIO, DISCR, LED, ..

Nicht mehr in Verwendung!

V2.x

Global Version ID & Date

#: Firmware Konfiguration & Version

§: 0: Datum

1..4: Module (SU7xx)

Module: ADCs, DACs (incl. Memory), DELAY, TDC, ..

V2.8

Code 127 = ,0' im MUX hinzugefügt.

Damit ändert sich beim Wechsel Code 255\127 (=1\0) nur 1 Bit und damit entstehen keine Glitches an den VI-Eingängen.

Firmware V2.8 kompatibel mit VIs V2.7!

Falls Code 127 benutzt wird, muss dies in B_S.vi (V2.8) geändert werden!

V2.10

Kommerziell verfügbare Version (WIENER)!

V3.0

Auslese Datum und Firmware-Konfiguration (Setup Modul) geändert.

#: Datum

§: 0: Firmware Konfiguration & Version

1: System (DL7xx)

2..9: Module (SU7xx)

Konsequente Versionsüberwachung mit Vis

V3.1

Module-ID (Anschlussnummer) um Statusbit erweitert (NICHT kompatibel zu 3.0).

V3.2

Module: SEQUENCER, PID, FREQUENCY, ..

V3.3

Teilweise Module mit Datenbus

V4.0

Module mit Datenbus

Module: RAM, FIFO, LUT, PLOGIC, ...

GATEGEN mit 2 Eingängen (nicht komp. zu 3.x)

COUNTER ersetzt!

LabVIEW

Icons (Picture, Terminals) neu und vereinheitlicht

Namen teilweise geändert

MUX Code: 0&127 =0; 128&255=1 (255=0 auf 255=1 geändert)

V4.1

Zusätzliche MUX Funktion in LOGIC.

ADC16 und DAC16 unterstützen Kanalinformation auf dem BUS!

TDC (V0.1, V1.1) erweitert

11 Inbetriebnahme

Bei Erstinstallation und ggf. bei auftretenden Problemen sind die folgenden Schritte nacheinander durchzuführen.

11.1 Geräteüberprüfung

Als erstes ist es sinnvoll zu überprüfen, ob das vorhandenes Gerät (LogicBox) eigenständig funktioniert, also eine funktionierende Firmware besitzt, und alle Submodule (SU7xx) am richtigen Ort gesteckt sind. Die USB-Verbindung sollte zunächst nicht verbunden sein!

1. Anlegen der geforderten Betriebsspannung (typ. 6V)
Dabei muss jetzt die **+5V-LED** an der Frontplatte (bzw. Rückseite) leuchten!
2. Beim Drücken der **RESET-Taste** muß die **BUSY-LED** und **alle LEDs** auf den Submodulen aufleuchten.

11.2 Rechneranschluß

Üblicherweise erfolgt der Anschluß an den PC über den USB-Bus.

1. Verrbinden der LogicBox mit PC über ein geeignetes **USB-Kabel**.
(U.U. ist hier auch die Verwendung eines USB-Isolators ratsam)
2. Wenn der Rechner ein neues Gerät meldet, muss ggf. der **Treiber** zunächst geladen werden. Geeignete Treiber für unterschiedliche Geräte (Readme.txt!) finden sich auf unserer Website unter:
http://www.physi.uni-heidelberg.de/Einrichtungen/EW/Geraete/DL700_LogicBox/Driver/
3. Nach erfolgreicher Anmeldung kann im **Gerätemanager** überprüft werden, ob das Gerät ordnungsgemäß installiert wurde. Hier ist auch die **Seriennummer** für die spätere Kommunikation ersichtlich.
4. Falls im weiteren noch Probleme auftauchen, kann (nur DL705, DL706, DL707, DL709) noch kontrolliert werden, ob die aktuelle Firmware für den USB-Controller (FX2) geladen ist. Dazu dient ein entsprechendes Programm **FX2_Update!**

11.3 Kommunikation

Zur Überprüfung der Kommunikation ist ein geeignetes Programm erforderlich!

Falls die vorhandenen Testprogramme in LabVIEW genutzt werden, sind diese (unter Windows) folgendermaßen zu installieren:

1. Im Ordner **.../LabVIEW.../user.lib** ist zunächst der komplette Interface-Ordner **LogicPool** abzulegen! Dieser ist ebenfalls auf der Website:
http://www.physi.uni-heidelberg.de/Einrichtungen/EW/Geraete/DL700_LogicBox/
im Ordner „LabVIEW“ in einem entsprechenden zip-File zu finden.

Damit sind alle Interface-VIs im LogicPool als Toolpalette in LabVIEW verfügbar!

2. WICHTIG: Der Ordner **LogicPool /USB** enthält verschiedene VIs zur Kommunikation mit dem Treiber. Hier ist das geeignete VI (siehe ReadMe.txt!) auszuwählen und in „**USB.vi**“ umzubenennen!
3. Nun kann z.B. über das vorhandene Programm (Datei) „**SETUP.vi**“ im Ordner „TestsDemos“ eine Überprüfung der Kommunikation und das Auslesen der Firmware-Konfiguration durchgeführt werden.
Nach Aufruf des Programms wird im Feld: **USBIn** die **Seriennummer** des Gerätes eingetragen und das Programm gestartet!
(Bei lediglich einer LogicBox im System ist als Seriennummer auch die Zahl 0 ausreichend!)
4. Das Programm liest nun verschiedene Informationen zur abgespeicherten Firmware aus, z.B.:

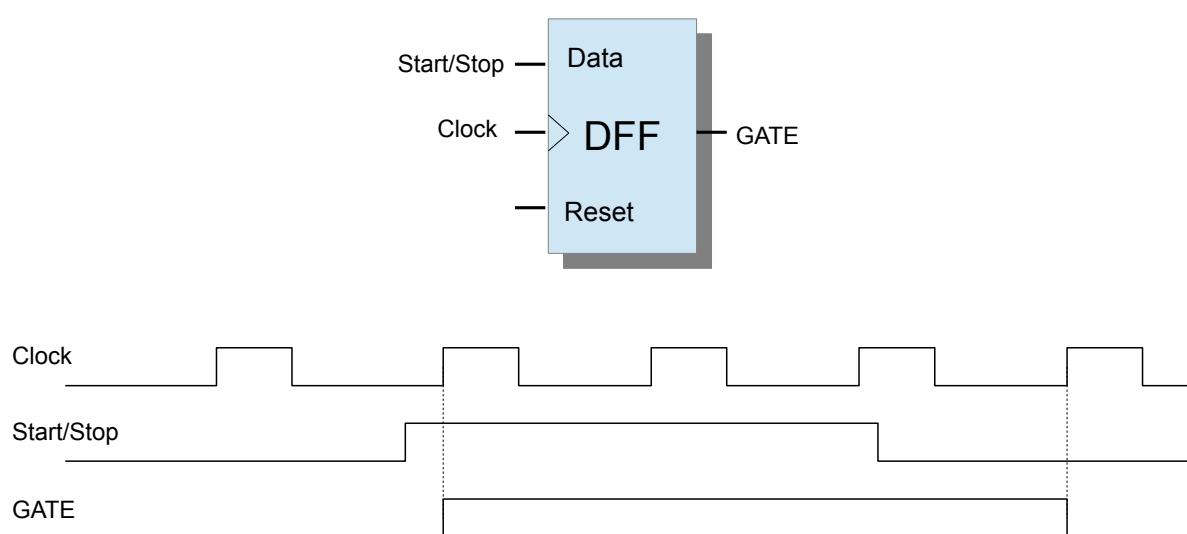
```
LP 10 / V4.0; 23.11.11 15:3:23
----- LogicBox
DL706: NIM USB 2.0 XC35400
----- SubModules
0: SU712_0: ADC (K0), 16ch, 14b, 6 us; BUS; 6mA; 108
1: SU703_0: Discriminator (4D0); TTL_Coax (1T0); LED (SIO); 370mA; 141
2: SU706_0: ADC (A0), 14b, 100 Mhz; Digital IO TTL_Coax (2T0); SU706-1,2; 62mA; 520
3: SU706_0: ADC (A0), 14b, 100 Mhz; Digital IO TTL_Coax (2T0); SU706-1,2; 62mA; 520
----- LogicPool
AO_1(1.0)3: Analog Digital Converter, 100MHz, 14 Bit, BUS; (ADC); 429
AO_2(1.0)4: Analog Digital Converter, 100MHz, 14 Bit, BUS; (ADC); 429
DO_1(1.0)13: Analog Input Discriminator; (DISCR); 6
```


12 Synchronisation

In vielen Fällen ist es notwendig, bestimmte Abläufe mit dem Rechner, bzw. den Rechner und dessen Aktionen mit Ereignissen im Experiment abzustimmen.

Da der zeitliche Ablauf vom Rechner (Start/Stop) in der Regel nicht exakt gesetzt werden kann, sollten notwendige Torschaltungen (GATE) in diesen Fällen immer mit entsprechender Hardware (z.B. DFF) auf die Ereignisse (Clock) synchronisiert werden.

Das folgende Bild zeigt das Prinzip der Synchronisation, wobei diese auch durch andere Schaltungsmodule (Counter, GateGen,...), die auf Signale triggern, erfolgen kann.



Das Signal GATE kann vom Rechner überwacht werden, um wiederum neue Messzyklen in geordneter Weise zu starten.

Gegebenenfalls kann es sinnvoll sein, mit zwei DFF, die auf unterschiedliche Flanken synchronisieren, eine Torschaltung aufzubauen um Glitches an den Enden zu vermeiden!