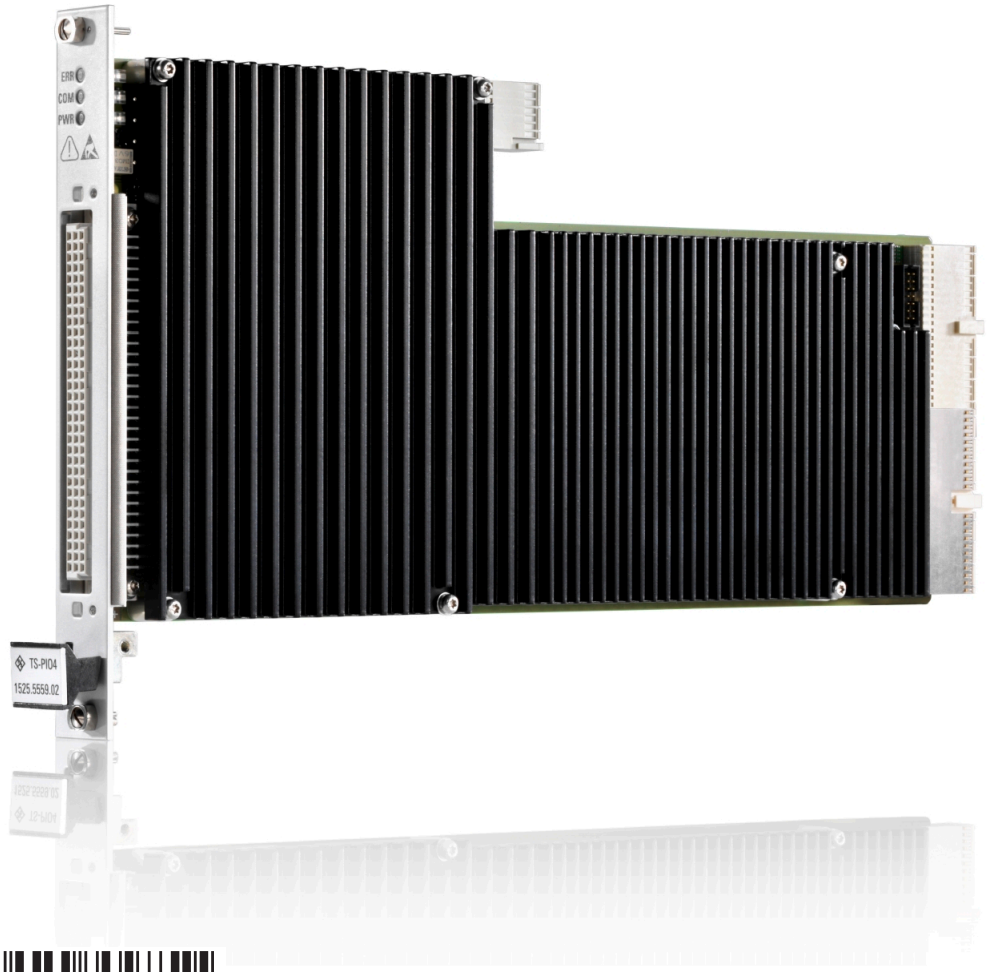
R&S®TS-PIO4
Digitales Funktionstestmodul
Bedienhandbuch



1178.3192.03 ─ 01

Messtechnik

Bedienhandbuch

ROHDE&SCHWARZ

Dieses Handbuch beschreibt das folgende R&S®TSVP Modul:

- R&S®TS-PIO4 (1525.5559.02)

Mühldorfstr. 15, 81671 München, Germany
Telefon: +49 89 41 29 - 0
Fax: +49 89 41 29 12 164
E-mail: info@rohde-schwarz.com
Internet: www.rohde-schwarz.com

Im vorliegenden Handbuch werden folgende Abkürzungen verewendet: R&S®TS-PIO4 wird abgekürzt mit R&S TS-PIO4.

Grundlegende Sicherheitshinweise

Lesen und beachten Sie unbedingt die nachfolgenden Anweisungen und Sicherheitshinweise!

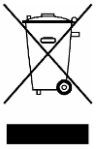
Alle Werke und Standorte der Rohde & Schwarz Firmengruppe sind ständig bemüht, den Sicherheitsstandard unserer Produkte auf dem aktuellsten Stand zu halten und unseren Kunden ein höchstmögliches Maß an Sicherheit zu bieten. Unsere Produkte und die dafür erforderlichen Zusatzgeräte werden entsprechend der jeweils gültigen Sicherheitsvorschriften gebaut und geprüft. Die Einhaltung dieser Bestimmungen wird durch unser Qualitätssicherungssystem laufend überwacht. Das vorliegende Produkt ist gemäß beiliegender EU-Konformitätsbescheinigung gebaut und geprüft und hat das Werk in sicherheitstechnisch einwandfreiem Zustand verlassen. Um diesen Zustand zu erhalten und einen gefahrlosen Betrieb sicherzustellen, muss der Benutzer alle Hinweise, Warnhinweise und Warnvermerke beachten. Bei allen Fragen bezüglich vorliegender Sicherheitshinweise steht Ihnen die Rohde & Schwarz Firmengruppe jederzeit gerne zur Verfügung.

Darüber hinaus liegt es in der Verantwortung des Benutzers, das Produkt in geeigneter Weise zu verwenden. Das Produkt ist ausschließlich für den Betrieb in Industrie und Labor bzw., wenn ausdrücklich zugelassen, auch für den Feldeinsatz bestimmt und darf in keiner Weise so verwendet werden, dass einer Person/Sache Schaden zugefügt werden kann. Die Benutzung des Produkts außerhalb des bestimmungsgemäßen Gebrauchs oder unter Missachtung der Anweisungen des Herstellers liegt in der Verantwortung des Benutzers. Der Hersteller übernimmt keine Verantwortung für die Zweckentfremdung des Produkts.

Die bestimmungsgemäße Verwendung des Produkts wird angenommen, wenn das Produkt nach den Vorgaben der zugehörigen Produktdokumentation innerhalb seiner Leistungsgrenzen verwendet wird (siehe Datenblatt, Dokumentation, nachfolgende Sicherheitshinweise). Die Benutzung des Produkts erfordert Fachkenntnisse und zum Teil englische Sprachkenntnisse. Es ist daher zu beachten, dass das Produkt ausschließlich von Fachkräften oder sorgfältig eingewiesenen Personen mit entsprechenden Fähigkeiten bedient werden darf. Sollte für die Verwendung von Rohde & Schwarz-Produkten persönliche Schutzausrüstung erforderlich sein, wird in der Produktdokumentation an entsprechender Stelle darauf hingewiesen. Bewahren Sie die grundlegenden Sicherheitshinweise und die Produktdokumentation gut auf und geben Sie diese an weitere Benutzer des Produkts weiter.

Die Einhaltung der Sicherheitshinweise dient dazu, Verletzungen oder Schäden durch Gefahren aller Art auszuschließen. Hierzu ist es erforderlich, dass die nachstehenden Sicherheitshinweise vor der Benutzung des Produkts sorgfältig gelesen und verstanden sowie bei der Benutzung des Produkts beachtet werden. Sämtliche weitere Sicherheitshinweise wie z.B. zum Personenschutz, die an entsprechender Stelle der Produktdokumentation stehen, sind ebenfalls unbedingt zu beachten. In den vorliegenden Sicherheitshinweisen sind sämtliche von der Rohde & Schwarz Firmengruppe vertriebenen Waren unter dem Begriff „Produkt“ zusammengefasst, hierzu zählen u. a. Geräte, Anlagen sowie sämtliches Zubehör.

Symbole und Sicherheitskennzeichnungen

Symbol	Bedeutung	Symbol	Bedeutung
	Achtung, allgemeine Gefahrenstelle Produktdokumentation beachten		EIN-/AUS (Versorgung)
	Vorsicht beim Umgang mit Geräten mit hohem Gewicht		Stand-by-Anzeige
	Gefahr vor elektrischem Schlag		Gleichstrom (DC)
	Warnung vor heißer Oberfläche		Wechselstrom (AC)
	Schutzleiteranschluss		Gleichstrom/Wechselstrom (DC/AC)
	Erdungsanschluss		Gerät entspricht den Sicherheits-anforderungen an die Schutzklasse II (Gerät durchgehend durch doppelte / verstärkte Isolierung geschützt.
	Masseanschluss des Gestells oder Gehäuses		EU - Kennzeichnung für Batterien und Akkumulatoren. Das Gerät enthält eine Batterie bzw. einen Akkumulator. Diese dürfen nicht über unsortierten Siedlungsabfall entsorgt werden, sondern sollten getrennt gesammelt werden. Weitere Informationen siehe Seite 7.
	Achtung beim Umgang mit elektrostatisch gefährdeten Bauelementen		EU - Kennzeichnung für die getrennte Sammlung von Elektro- und Elektronikgeräten. Elektroaltgeräte dürfen nicht über unsortierten Siedlungsabfall entsorgt werden, sondern müssen getrennt gesammelt werden. Weitere Informationen siehe Seite 7.
	Warnung vor Laserstrahl Produkte mit Laser sind je nach ihrer Laser-Klasse mit genormten Warnhinweisen versehen. Laser können aufgrund der Eigenschaften ihrer Strahlung und aufgrund ihrer extrem konzentrierten elektromagnetischen Leistung biologische Schäden verursachen. Für zusätzliche Informationen siehe Kapitel „Betrieb“ Punkt 7.		

Signalworte und ihre Bedeutung

Die folgenden Signalworte werden in der Produktdokumentation verwendet, um vor Risiken und Gefahren zu warnen.

kennzeichnet eine unmittelbare Gefährdung mit hohem Risiko, die Tod oder schwere Körperverletzung zur Folge haben wird, wenn sie nicht vermieden wird.

kennzeichnet eine mögliche Gefährdung mit mittlerem Risiko, die Tod oder (schwere) Körperverletzung zur Folge haben kann, wenn sie nicht vermieden wird.

kennzeichnet eine Gefährdung mit geringem Risiko, die leichte oder mittlere Körperverletzungen zur Folge haben könnte, wenn sie nicht vermieden wird.

weist auf die Möglichkeit einer Fehlbedienung hin, bei der das Produkt Schaden nehmen kann.

Diese Signalworte entsprechen der im europäischen Wirtschaftsraum üblichen Definition für zivile Anwendungen. Neben dieser Definition können in anderen Wirtschaftsräumen oder bei militärischen Anwendungen abweichende Definitionen existieren. Es ist daher darauf zu achten, dass die hier beschriebenen Signalworte stets nur in Verbindung mit der zugehörigen Produktdokumentation und nur in Verbindung mit dem zugehörigen Produkt verwendet werden. Die Verwendung von Signalworten in Zusammenhang mit nicht zugehörigen Produkten oder nicht zugehörigen Dokumentationen kann zu Fehlinterpretationen führen und damit zu Personen- oder Sachschäden führen.

Betriebszustände und Betriebslagen

Das Produkt darf nur in den vom Hersteller angegebenen Betriebszuständen und Betriebslagen ohne Behinderung der Belüftung betrieben werden. Werden die Herstellerangaben nicht eingehalten, kann dies elektrischen Schlag, Brand und/oder schwere Verletzungen von Personen, unter Umständen mit Todesfolge, verursachen. Bei allen Arbeiten sind die örtlichen bzw. landesspezifischen Sicherheits- und Unfallverhütungsvorschriften zu beachten.

1. Sofern nicht anders vereinbart, gilt für R&S-Produkte folgendes:
   als vorgeschriebene Betriebslage grundsätzlich Gehäuseboden unten, IP-Schutzart 2X, Verschmutzungsgrad 2, Überspannungskategorie 2, nur in Innenräumen verwenden, Betrieb bis 2000 m ü. NN, Transport bis 4500 m ü. NN, für die Nennspannung gilt eine Toleranz von ±10%, für die Nennfrequenz eine Toleranz von ±5%.
2. Stellen Sie das Produkt nicht auf Oberflächen, Fahrzeuge, Ablagen oder Tische, die aus Gewichts- oder Stabilitätsgründen nicht dafür geeignet sind. Folgen Sie bei Aufbau und Befestigung des Produkts an Gegenständen oder Strukturen (z.B. Wände und Regale) immer den Installationshinweisen des Herstellers. Bei Installation abweichend von der Produktdokumentation können Personen verletzt, unter Umständen sogar getötet werden.
3. Stellen Sie das Produkt nicht auf hitzeerzeugende Gerätschaften (z.B. Radiatoren und Heizlüfter). Die Umgebungstemperatur darf nicht die in der Produktdokumentation oder im Datenblatt spezifizierte Maximaltemperatur überschreiten. Eine Überhitzung des Produkts kann elektrischen Schlag, Brand und/oder schwere Verletzungen von Personen, unter Umständen mit Todesfolge, verursachen.

Elektrische Sicherheit

Werden die Hinweise zur elektrischen Sicherheit nicht oder unzureichend beachtet, kann dies elektrischen Schlag, Brand und/oder schwere Verletzungen von Personen, unter Umständen mit Todesfolge, verursachen.

1. Vor jedem Einschalten des Produkts ist sicherzustellen, dass die am Produkt eingestellte Nennspannung und die Netznennspannung des Versorgungsnetzes übereinstimmen. Ist es erforderlich, die Spannungseinstellung zu ändern, so muss ggf. auch die dazu gehörige Netzsicherung des Produkts geändert werden.
2. Bei Produkten der Schutzklasse I mit beweglicher Netzzuleitung und Gerätesteckvorrichtung ist der Betrieb nur an Steckdosen mit Schutzkontakt und angeschlossenem Schutzleiter zulässig.
3. Jegliche absichtliche Unterbrechung des Schutzleiters, sowohl in der Zuleitung als auch am Produkt selbst, ist unzulässig. Es kann dazu führen, dass von dem Produkt die Gefahr eines elektrischen Schlags ausgeht. Bei Verwendung von Verlängerungsleitungen oder Steckdosenleisten ist sicherzustellen, dass diese regelmäßig auf ihren sicherheitstechnischen Zustand überprüft werden.
4. Sofern das Produkt nicht mit einem Netzschalter zur Netztrennung ausgerüstet ist, beziehungsweise der vorhandene Netzschalter zu Netztrennung nicht geeignet ist, so ist der Stecker des Anschlusskabels als Trennvorrichtung anzusehen.
   Die Trennvorrichtung muss jederzeit leicht erreichbar und gut zugänglich sein. Ist z.B. der Netzstecker die Trennvorrichtung, darf die Länge des Anschlusskabels 3 m nicht überschreiten.
   Funktionsschalter oder elektronische Schalter sind zur Netztrennung nicht geeignet. Werden Produkte ohne Netzschalter in Gestelle oder Anlagen integriert, so ist die Trennvorrichtung auf Anlagenebene zu verlagern.
5. Benutzen Sie das Produkt niemals, wenn das Netzkabel beschädigt ist. Überprüfen Sie regelmäßig den einwandfreien Zustand der Netzkabel. Stellen Sie durch geeignete Schutzmaßnahmen und Verlegearten sicher, dass das Netzkabel nicht beschädigt werden kann und niemand z.B. durch Stolperfallen oder elektrischen Schlag zu Schaden kommen kann.
6. Der Betrieb ist nur an TN/TT Versorgungsnetzen gestattet, die mit höchstens 16 A abgesichert sind (höhere Absicherung nur nach Rücksprache mit der Rohde & Schwarz Firmengruppe).
7. Stecken Sie den Stecker nicht in verstaubte oder verschmutzte Steckdosen/-buchsen. Stecken Sie die Steckverbindung/-vorrichtung fest und vollständig in die dafür vorgesehenen Steckdosen/-buchsen. Missachtung dieser Maßnahmen kann zu Funken, Feuer und/oder Verletzungen führen.
8. Überlasten Sie keine Steckdosen, Verlängerungskabel oder Steckdosenleisten, dies kann Feuer oder elektrische Schläge verursachen.
9. Bei Messungen in Stromkreisen mit Spannungen U_{eff} > 30 V ist mit geeigneten Maßnahmen Vorsorge zu treffen, dass jegliche Gefährdung ausgeschlossen wird (z.B. geeignete Messmittel, Absicherung, Strombegrenzung, Schutztrennung, Isolierung usw.).
10. Bei Verbindungen mit informationstechnischen Geräten, z.B. PC oder Industrierechner, ist darauf zu achten, dass diese der jeweils gültigen IEC 60950-1 / EN 60950-1 oder IEC 61010-1 / EN 61010-1 entsprechen.
11. Sofern nicht ausdrücklich erlaubt, darf der Deckel oder ein Teil des Gehäuses niemals entfernt werden, wenn das Produkt betrieben wird. Dies macht elektrische Leitungen und Komponenten zugänglich und kann zu Verletzungen, Feuer oder Schaden am Produkt führen.

12. Wird ein Produkt ortsfest angeschlossen, ist die Verbindung zwischen dem Schutzleiteranschluss vor Ort und dem Geräteschutzleiter vor jeglicher anderer Verbindung herzustellen. Aufstellung und Anschluss darf nur durch eine Elektrofachkraft erfolgen.
13. Bei ortsfesten Geräten ohne eingebaute Sicherung, Selbstschalter oder ähnliche Schutzeinrichtung muss der Versorgungskreis so abgesichert sein, dass alle Personen, die Zugang zum Produkt haben, sowie das Produkt selbst ausreichend vor Schäden geschützt sind.
14. Jedes Produkt muss durch geeigneten Überspannungsschutz vor Überspannung (z.B. durch Blitzschlag) geschützt werden. Andernfalls ist das bedienende Personal durch elektrischen Schlag gefährdet.
15. Gegenstände, die nicht dafür vorgesehen sind, dürfen nicht in die Öffnungen des Gehäuses eingebracht werden. Dies kann Kurzschlüsse im Produkt und/oder elektrische Schläge, Feuer oder Verletzungen verursachen.
16. Sofern nicht anders spezifiziert, sind Produkte nicht gegen das Eindringen von Flüssigkeiten geschützt, siehe auch Abschnitt "Betriebszustände und Betriebslagen", Punkt 1. Daher müssen die Geräte vor Eindringen von Flüssigkeiten geschützt werden. Wird dies nicht beachtet, besteht Gefahr durch elektrischen Schlag für den Benutzer oder Beschädigung des Produkts, was ebenfalls zur Gefährdung von Personen führen kann.
17. Benutzen Sie das Produkt nicht unter Bedingungen, bei denen Kondensation in oder am Produkt stattfinden könnte oder ggf. bereits stattgefunden hat, z.B. wenn das Produkt von kalter in warme Umgebung bewegt wurde. Das Eindringen von Wasser erhöht das Risiko eines elektrischen Schlages.
18. Trennen Sie das Produkt vor der Reinigung komplett von der Energieversorgung (z.B. speisendes Netz oder Batterie). Nehmen Sie bei Geräten die Reinigung mit einem weichen, nicht fasernden Staublappen vor. Verwenden Sie keinesfalls chemische Reinigungsmittel wie z.B. Alkohol, Aceton, Nitroverdünnung.

Betrieb

1. Die Benutzung des Produkts erfordert spezielle Einweisung und hohe Konzentration während der Benutzung. Es muss sichergestellt sein, dass Personen, die das Produkt bedienen, bezüglich ihrer körperlichen, geistigen und seelischen Verfassung den Anforderungen gewachsen sind, da andernfalls Verletzungen oder Sachschäden nicht auszuschließen sind. Es liegt in der Verantwortung des Arbeitsgebers/Betreibers, geeignetes Personal für die Benutzung des Produkts auszuwählen.
2. Bevor Sie das Produkt bewegen oder transportieren, lesen und beachten Sie den Abschnitt "Transport".
3. Wie bei allen industriell gefertigten Gütern kann die Verwendung von Stoffen, die Allergien hervorrufen - so genannte Allergene (z.B. Nickel) - nicht generell ausgeschlossen werden. Sollten beim Umgang mit R&S-Produkten allergische Reaktionen, z.B. Hautausschlag, häufiges Niesen, Bindehautrötung oder Atembeschwerden auftreten, ist umgehend ein Arzt aufzusuchen, um die Ursachen zu klären und Gesundheitsschäden bzw. -belastungen zu vermeiden.
4. Vor der mechanischen und/oder thermischen Bearbeitung oder Zerlegung des Produkts beachten Sie unbedingt Abschnitt "Entsorgung", Punkt 1.

5. Bei bestimmten Produkten, z.B. HF-Funkanlagen, können funktionsbedingt erhöhte elektromagnetische Strahlungen auftreten. Unter Berücksichtigung der erhöhten Schutzwürdigkeit des ungeborenen Lebens müssen Schwangere durch geeignete Maßnahmen geschützt werden. Auch Träger von Herzschrittmachern können durch elektromagnetische Strahlungen gefährdet sein. Der Arbeitgeber/Betreiber ist verpflichtet, Arbeitsstätten, bei denen ein besonderes Risiko einer Strahlenexposition besteht, zu beurteilen und zu kennzeichnen und mögliche Gefahren abzuwenden.
6. Im Falle eines Brandes entweichen ggf. giftige Stoffe (Gase, Flüssigkeiten etc.) aus dem Produkt, die Gesundheitsschäden verursachen können. Daher sind im Brandfall geeignete Maßnahmen wie z.B. Atemschutzmasken und Schutzkleidung zu verwenden.
7. Falls ein Laser-Produkt in ein R&S-Produkt integriert ist (z.B. CD/DVD-Laufwerk), dürfen keine anderen Einstellungen oder Funktionen verwendet werden, als in der Produktdokumentation beschrieben, um Personenschäden zu vermeiden (z.B. durch Laserstrahl).
8. EMV Klassen (nach EN 55011 / CISPR 11; sinngemäß EN 55022 / CISPR 22, EN 55032 / CISPR 32)

   Gerät der Klasse A:
   Ein Gerät, das sich für den Gebrauch in allen anderen Bereichen außer dem Wohnbereich und solchen Bereichen eignet, die direkt an ein Niederspannungs-Versorgungsnetz angeschlossen sind, das Wohngebäude versorgt.
   Hinweis: Diese Einrichtung kann wegen möglicher auftretender leitungsgebundener als auch gestrahlten Störgrößen im Wohnbereich Funkstörungen verursachen. In diesem Fall kann vom Betreiber verlangt werden, angemessene Maßnahmen durchzuführen.
   Gerät der Klasse B:
   Ein Gerät, das sich für den Betrieb im Wohnbereich sowie in solchen Bereichen eignet, die direkt an ein Niederspannungs-Versorgungsnetz angeschlossen sind, das Wohngebäude versorgt.

Reparatur und Service

1. Das Produkt darf nur von dafür autorisiertem Fachpersonal geöffnet werden. Vor Arbeiten am Produkt oder Öffnen des Produkts ist dieses von der Versorgungsspannung zu trennen, sonst besteht das Risiko eines elektrischen Schlages.
2. Abgleich, Auswechseln von Teilen, Wartung und Reparatur darf nur von R&S-autorisierten Elektrofachkräften ausgeführt werden. Werden sicherheitsrelevante Teile (z.B. Netzschalter, Netztrafos oder Sicherungen) ausgewechselt, so dürfen diese nur durch Originalteile ersetzt werden. Nach jedem Austausch von sicherheitsrelevanten Teilen ist eine Sicherheitsprüfung durchzuführen (Sichtprüfung, Schutzleitertest, Isolationswiderstand-, Ableitstrommessung, Funktionstest). Damit wird sichergestellt, dass die Sicherheit des Produkts erhalten bleibt.

Batterien und Akkumulatoren/Zellen

Werden die Hinweise zu Batterien und Akkumulatoren/Zellen nicht oder unzureichend beachtet, kann dies Explosion, Brand und/oder schwere Verletzungen von Personen, unter Umständen mit Todesfolge, verursachen. Die Handhabung von Batterien und Akkumulatoren mit alkalischen Elektrolyten (z.B. Lithiumzellen) muss der EN 62133 entsprechen.

1. Zellen dürfen nicht zerlegt, geöffnet oder zerkleinert werden.
2. Zellen oder Batterien dürfen weder Hitze noch Feuer ausgesetzt werden. Die Lagerung im direkten Sonnenlicht ist zu vermeiden. Zellen und Batterien sauber und trocken halten. Verschmutzte Anschlüsse mit einem trockenen, sauberen Tuch reinigen.

3. Zellen oder Batterien dürfen nicht kurzgeschlossen werden. Zellen oder Batterien dürfen nicht gefahrbringend in einer Schachtel oder in einem Schubfach gelagert werden, wo sie sich gegenseitig kurzschließen oder durch andere leitende Werkstoffe kurzgeschlossen werden können. Eine Zelle oder Batterie darf erst aus ihrer Originalverpackung entnommen werden, wenn sie verwendet werden soll.
4. Zellen oder Batterien dürfen keinen unzulässig starken, mechanischen Stößen ausgesetzt werden.
5. Bei Undichtheit einer Zelle darf die Flüssigkeit nicht mit der Haut in Berührung kommen oder in die Augen gelangen. Falls es zu einer Berührung gekommen ist, den betroffenen Bereich mit reichlich Wasser waschen und ärztliche Hilfe in Anspruch nehmen.
6. Werden Zellen oder Batterien, die alkalische Elektrolyte enthalten (z.B. Lithiumzellen), unsachgemäß ausgewechselt oder geladen, besteht Explosionsgefahr. Zellen oder Batterien nur durch den entsprechenden R&S-Typ ersetzen (siehe Ersatzteilliste), um die Sicherheit des Produkts zu erhalten.
7. Zellen oder Batterien müssen wiederverwertet werden und dürfen nicht in den Restmüll gelangen. Akkumulatoren oder Batterien, die Blei, Quecksilber oder Cadmium enthalten, sind Sonderabfall. Beachten Sie hierzu die landesspezifischen Entsorgungs- und Recycling-Bestimmungen.

Transport

1. Das Produkt kann ein hohes Gewicht aufweisen. Daher muss es vorsichtig und ggf. unter Verwendung eines geeigneten Hebemittels (z.B. Hubwagen) bewegt bzw. transportiert werden, um Rückenschäden oder Verletzungen zu vermeiden.
2. Griffe an den Produkten sind eine Handhabungshilfe, die ausschließlich für den Transport des Produkts durch Personen vorgesehen ist. Es ist daher nicht zulässig, Griffe zur Befestigung an bzw. auf Transportmitteln, z.B. Kränen, Gabelstaplern, Karren etc. zu verwenden. Es liegt in Ihrer Verantwortung, die Produkte sicher an bzw. auf geeigneten Transport- oder Hebemitteln zu befestigen. Beachten Sie die Sicherheitsvorschriften des jeweiligen Herstellers eingesetzter Transport- oder Hebemittel, um Personenschäden und Schäden am Produkt zu vermeiden.
3. Falls Sie das Produkt in einem Fahrzeug benutzen, liegt es in der alleinigen Verantwortung des Fahrers, das Fahrzeug in sicherer und angemessener Weise zu führen. Der Hersteller übernimmt keine Verantwortung für Unfälle oder Kollisionen. Verwenden Sie das Produkt niemals in einem sich bewegenden Fahrzeug, sofern dies den Fahrzeugführer ablenken könnte. Sichern Sie das Produkt im Fahrzeug ausreichend ab, um im Falle eines Unfalls Verletzungen oder Schäden anderer Art zu verhindern.

Entsorgung

1. Batterien bzw. Akkumulatoren, die nicht mit dem Hausmüll entsorgt werden dürfen, darf nach Ende der Lebensdauer nur über eine geeignete Sammelstelle oder eine Rohde & Schwarz-Kundendienststelle entsorgt werden.
2. Am Ende der Lebensdauer des Produktes darf dieses Produkt nicht über den normalen Hausmüll entsorgt werden, sondern muss getrennt gesammelt werden.
   Rohde & Schwarz GmbH & Co.KG ein Entsorgungskonzept entwickelt und übernimmt die Pflichten der Rücknahme- und Entsorgung für Hersteller innerhalb der EU in vollem Umfang. Wenden Sie sich bitte an Ihre Rohde & Schwarz-Kundendienststelle, um das Produkt umweltgerecht zu entsorgen.

3. Werden Produkte oder ihre Bestandteile über den bestimmungsgemäßen Betrieb hinaus mechanisch und/oder thermisch bearbeitet, können ggf. gefährliche Stoffe (schwermetallhaltiger Staub wie z.B. Blei, Beryllium, Nickel) freigesetzt werden. Die Zerlegung des Produkts darf daher nur von speziell geschultem Fachpersonal erfolgen. Unsachgemäßes Zerlegen kann Gesundheitsschäden hervorrufen. Die nationalen Vorschriften zur Entsorgung sind zu beachten.
4. Falls beim Umgang mit dem Produkt Gefahren- oder Betriebsstoffe entstehen, die speziell zu entsorgen sind, z.B. regelmäßig zu wechselnde Kühlmittel oder Motorenöle, sind die Sicherheitshinweise des Herstellers dieser Gefahren- oder Betriebsstoffe und die regional gültigen Entsorgungsvorschriften einzuhalten. Beachten Sie ggf. auch die zugehörigen speziellen Sicherheitshinweise in der Produktdokumentation. Die unsachgemäße Entsorgung von Gefahren- oder Betriebsstoffen kann zu Gesundheitsschäden von Personen und Umweltschäden führen.

Weitere Informationen zu Umweltschutz finden Sie auf der Rohde & Schwarz Home Page.

Quality management and environmental management

Certified Quality System
ISO 9001

Certified Environmental System
ISO 14001

Sehr geehrter Kunde,
Sie haben sich für den Kauf eines Rohde&Schwarz Produktes entschieden. Sie erhalten damit ein nach modernsten Fertigungsmethoden hergestelltes Produkt. Es wurde nach den Regeln unserer Qualitäts- und Umweltmanagementsysteme entwickelt, gefertigt und geprüft. Rohde&Schwarz ist unter anderem nach den Managementsystemen ISO 9001 und ISO 14001 zertifiziert.

Der Umwelt verpflichtet
- Energie-effiziente, RoHS-konforme Produkte
- Kontinuierliche Weiterentwicklung nachhaltiger Umweltkonzepte
- ISO 14001-zertifiziertes Umweltmanagementsystem

Dear customer,
You have decided to buy a Rohde&Schwarz product. This product has been manufactured using the most advanced methods. It was developed, manufactured and tested in compliance with our quality management and environmental management systems. Rohde&Schwarz has been certified, for example, according to the ISO 9001 and ISO 14001 management systems.

Environmental commitment
- Energy-efficient products
- Continuous improvement in environmental sustainability
- ISO 14001-certified environmental management system

Cher client,
Vous avez choisi d'acheter un produit Rohde&Schwarz. Vous disposez donc d'un produit fabriqué d'après les méthodes les plus avancées. Le développement, la fabrication et les tests de ce produit ont été effectués selon nos systèmes de management de qualité et de management environnemental. La société Rohde&Schwarz a été homologuée, entre autres, conformément aux systèmes de management ISO 9001 et ISO 14001.

Engagement écologique
- Produits à efficience énergétique
- Amélioration continue de la durabilité environnementale
- Système de management environnemental certifié selon ISO 14001

ROHDE&SCHWARZ
1171.0200.11 V 05.01
1171020011

Customer Support

Technischer Support – wo und wann Sie ihn brauchen

Unser Customer Support Center bietet Ihnen schnelle, fachmännische Hilfe für die gesamte Produktpalette von Rohde & Schwarz an. Ein Team von hochqualifizierten Ingenieuren unterstützt Sie telefonisch und arbeitet mit Ihnen eine Lösung für Ihre Anfrage aus - egal, um welchen Aspekt der Bedienung, Programmierung oder Anwendung eines Rohde & Schwarz Produktes es sich handelt.

Aktuelle Informationen und Upgrades

Um Ihr Gerät auf dem aktuellsten Stand zu halten sowie Informationen über Applikationsschriften zu Ihrem Gerät zu erhalten, senden Sie bitte eine E-Mail an das Customer Support Center. Geben Sie hierbei den Gerätenamen und Ihr Anliegen an. Wir stellen dann sicher, dass Sie die gewünschten Informationen erhalten.

Europa, Afrika, Mittlerer Osten	Tel. +49 89 4129 12345 customersupport@rohde-schwarz.com
Nordamerika	Tel. 1-888-TEST-RSA (1-888-837-8772) customer.support@rsa.rohde-schwarz.com
Lateinamerika	Tel. +1-410-910-7988 customersupport.la@rohde-schwarz.com
Asien/Pazifik	Tel. +65 65 13 04 88 customersupport.asia@rohde-schwarz.com
China	Tel. +86-800-810-8228 / +86-400-650-5896 customersupport.china@rohde-schwarz.com

ROHDE&SCHWARZ

1171.0200.21-06.00

Inhalt

1 Anwendungen

1.1 Allgemeines

Dieses Handbuch beschreibt Funktion und Betrieb des ROHDE&SCHWARZ digitalen Funktionstestmoduls R&S TS-PIO4 für die Verwendung in der Test System Versatile Platform R&S CompactTSVP. Die Hardware wird als CompactPCI-Karte realisiert, die nur einen Slot im frontseitigen Bereich des TSVP belegt.

Das digitale Funktionstestmodul R&S TS-PIO4 kommt überall dort zum Einsatz, wo digitale Schaltungen durch flexibel programmierbare statische oder dynamische Stimulation und durch Aufzeichnung der Reaktion getestet werden.

Die deterministische, simultane Stimulation und Aufzeichnung von digitalen Signalen ermöglicht eine realitätsnahe Nachbildung von Betriebsbedingungen. Damit das exakte und vorhersagbare Zeitverhalten zum Ausgeben, Erfassen und Analysieren der Bitmuster eingehalten werden kann, stehen auf dem Modul ein großer lokaler Speicher und eine autarke Ablaufsteuerung zur Verfügung. Umfangreiche Triggermöglichkeiten über den PXI-Triggerbus ermöglichen die Synchronisierung mit weiteren R&S TS-PIO4 Modulen oder anderen Mess- und Stimulusmodulen. Dadurch lässt sich die Anzahl der Digitalkanäle in einer Anwendung erweitern. Weiterhin sind Messungen möglich, bei denen Signale synchron erfasst werden sollen.

Die Programmierbarkeit von Ausgangspegeln und Eingangsschwellwerten in 8 Ports mit je 4 Kanälen ermöglicht eine optimale Anpassung an die Anforderungen unterschiedlicher Logikfamilien. Der Einfluss von Störsignalen im Prüfaufbau kann durch die Einstellbarkeit einer Hysterese bei den Eingangsschwellen minimiert werden.

Die Schutzbeschaltungen gegen Kurzschlüsse, Gegenspannungen und Überspannungen tragen zur Robustheit des digitalen Funktionstestmoduls R&S TS-PIO4 bei. Durch den äußerst Platz sparenden Aufbau der I/O-Schutzbeschaltung und Signalkonditionierung belegt das R&S TS-PIO4 nur einen CompactPCI/PXI Slot. Dies ermöglicht den Aufbau von sehr leistungsfähigen und kompakten Messsystemen.

Das Digitale Funktionstestmodul R&S TS-PIO4 ist für die Testplattform R&S CompactTSVP bestimmt. Die Ansteuerung des Moduls erfolgt über den CompactPCI Bus.

Zur Bedienung des Moduls steht ein Softpanel zur Verfügung. Die Ansteuerung erfolgt über einen IVI-C Treiber.

1.2 Eigenschaften

Eigenschaften des digitalen Funktionstestmoduls R&S TS-PIO4.

- 32 digitale Eingänge und 32 digitale Ausgänge eingeteilt in 8 Ports mit je 4 Kanälen
- Programmierbarer Ausgangspegelbereich -6 V bis 10 V Ausgangswiderstand 30 Ω (typ.)

- Auflösung 14 Bit
- Ausgangsstrom bis 150 mA pro Kanal, 350 mA pro Port
- Tristate Steuerung im dynamischen und statischen Modus
- Zwei programmierbare Eingangsschaltschwellen pro Port für Hysterese oder Pegel Monitoring
- Eingänge mit Ausgängen verbindbar
- Patternrate bis 40 MS/s pro Kanal • 12,5 nsec. Auflösung
- Speichertiefe 2 MSample (32 bit Daten)
- Externer Takteingang
- Synchronisierung/Triggerung via PXI Trigger Bus und via XTI (TTL)
- Timer/Counter Funktion
- FPGA basierende Flexibilität
- Autarke Selbsttestfähigkeit
- LabWindows IVI-C Treiber verfügbar
- Einsatz im R&S CompactTSVP

1.3 Einsatzmöglichkeiten

Das Digitale Funktionstestmodul R&S TS-PIO4 dient dem Test digitaler Baugruppen oder Geräte. Ein solcher Funktionstest prüft die Gesamtfunktion einer digitalen Schaltung unter möglichst realen Betriebsbedingungen. Dazu werden digitale Eingangsmuster angelegt, die Ausgangssignale gemessen und mit den Sollwerten verglichen.

Das Digitale Funktionstestmodul R&S TS-PIO4 kann unter anderem für folgende Aufgaben eingesetzt werden:

- Digitaler Funktionstest (Low-Speed/High-Speed, IO Steuerung)
- Bitmusterstimulierung (Low-Speed/High-Speed, digitale Busse)
- Bitmustermessung (Low-Speed/High-Speed)
- Überwachung von Pegelzustandsänderungen (Patterntrigger)
- Digitaler Funktionstest auf Bauteilebene (keine Nodeforcing- bzw. Backdriving-Fähigkeit)
- Downloads, z.B. für Flash-Bausteine, seriell und parallel

Ein typischer Funktionstest besteht aus folgenden Komponenten:

- Anpassung der Pin-Elektronik an die Prüflingsumgebung (Logikpegel und Logikfamilie)
- Definition des Sensor-Strobes (Timing)
- Definition des Stimulus- und Messverhaltens
- Auswertung des Testergebnisses

1.3.1 Digitaler Funktionstest- Statisch

Beim statischen digitalen Funktionstest werden Eigenschaften geprüft, bei denen es mehr auf das richtige Zusammenwirken der Logikbausteine und weniger auf den Nachweis von zeitkritischen Grenzwerten ankommt. Die Anwendung gibt die zu stimulierenden Muster aus, liest die Prüflingsreaktion über die Moduleingänge ein und vergleicht sie mit der erwarteten Reaktion. Der Vergleich mit der Vorgabe ergibt dann eine PASS/FAIL-Aussage. Der Vergleich erfolgt in der Applikation. Da das Zeitverhalten (Dauer, wie lange ein Pattern ansteht; Abtastzeitpunkt der Eingänge) im Wesentlichen vom Host-Rechner kontrolliert wird, können hiermit keine zeitkritischen bzw. zeitlich genau definierten Abläufe getestet werden. Beim statischen Test ist die Sequenz des Ablaufs deterministisch, d.h. die zeitliche Abfolge der Pattern sowie des Einlesens ist vorhersagbar. Nicht vorhersagbar ist jedoch, wie lange der Abstand von einem Pattern zum nächsten sein wird und auch nicht wie groß der Abstand vom Anlegen eines Patterns zum Einlesen der Daten. Hier beeinflusst das Betriebssystem des Hostrechners (Taskwechsel usw.) in nicht vorhersagbarer Weise die Laufzeiten.

1.3.2 Digitaler Funktionstest - Dynamisch

Mit dem Echtzeittest wird die Gesamtfunktion des Digitalteils eines Prüflings unter möglichst realen Betriebsbedingungen geprüft. Dazu werden digitale Muster (Vektoren bzw. Pattern-Sets) mit genau definierter, meist hoher Taktrate und präzisem zeitlichen Verhalten an die Prüflingsanschlüsse angelegt und die Reaktionen aufgezeichnet. Auch die Aufzeichnung erfolgt mit einem genau definierten, reproduzierbaren Zeitverhalten. Grundbedingung für ein exaktes, vorhersagbares Timing ist, dass die Muster für den Stimulus in Speichern "hinter den Treibern" abgelegt sind und mit definiertem Zeitverhalten und bei hoher Geschwindigkeit abgearbeitet werden können. Dafür verfügt das Modul über eine eigene Ablaufsteuerung im FPGA. Ebenso werden die aufgezeichneten Eingangsdaten zunächst in einem Speicher auf dem Modul abgelegt und später von der Applikation auf dem Hostrechner abgeholt und für eine PASS/FAIL-Entscheidung ausgewertet.

2 Ansicht

Bild 2-1 zeigt das Digitale Funktionstestmodul R&S TS-PIO4 (ohne Kühlkörper).

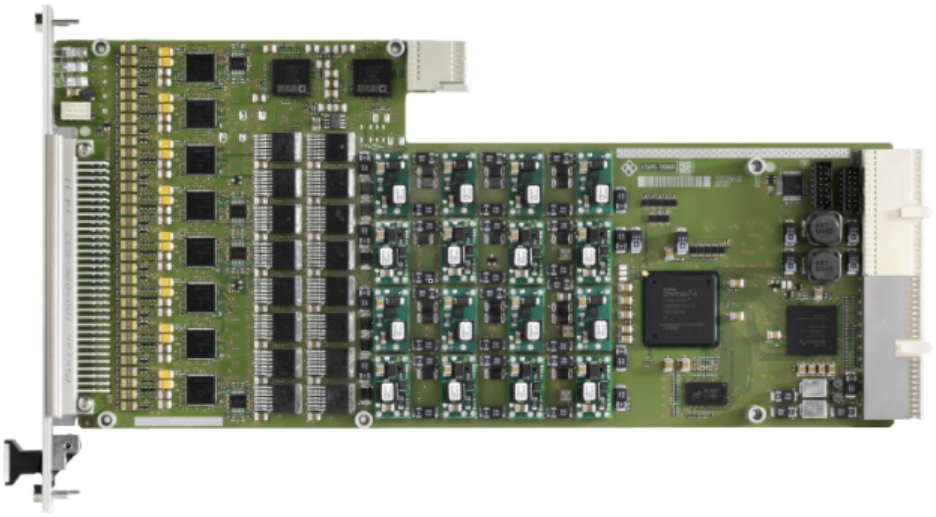


Bild 2-1: Ansicht des Moduls

3 Blockschaltbilder

In diesem Abschnitt wird sowohl das Funktionsblockschaltbild des Moduls R&S TS-PIO4 als auch das detaillierte Blockschaltbild dargestellt.

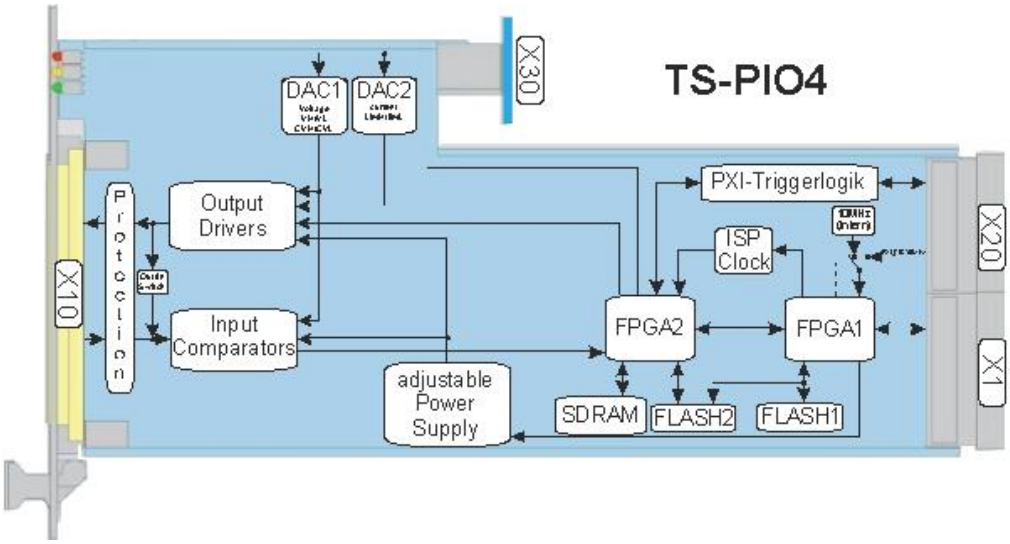


Bild 3-1: Funktionsblockschaltbild des Moduls R&S TS-PIO4

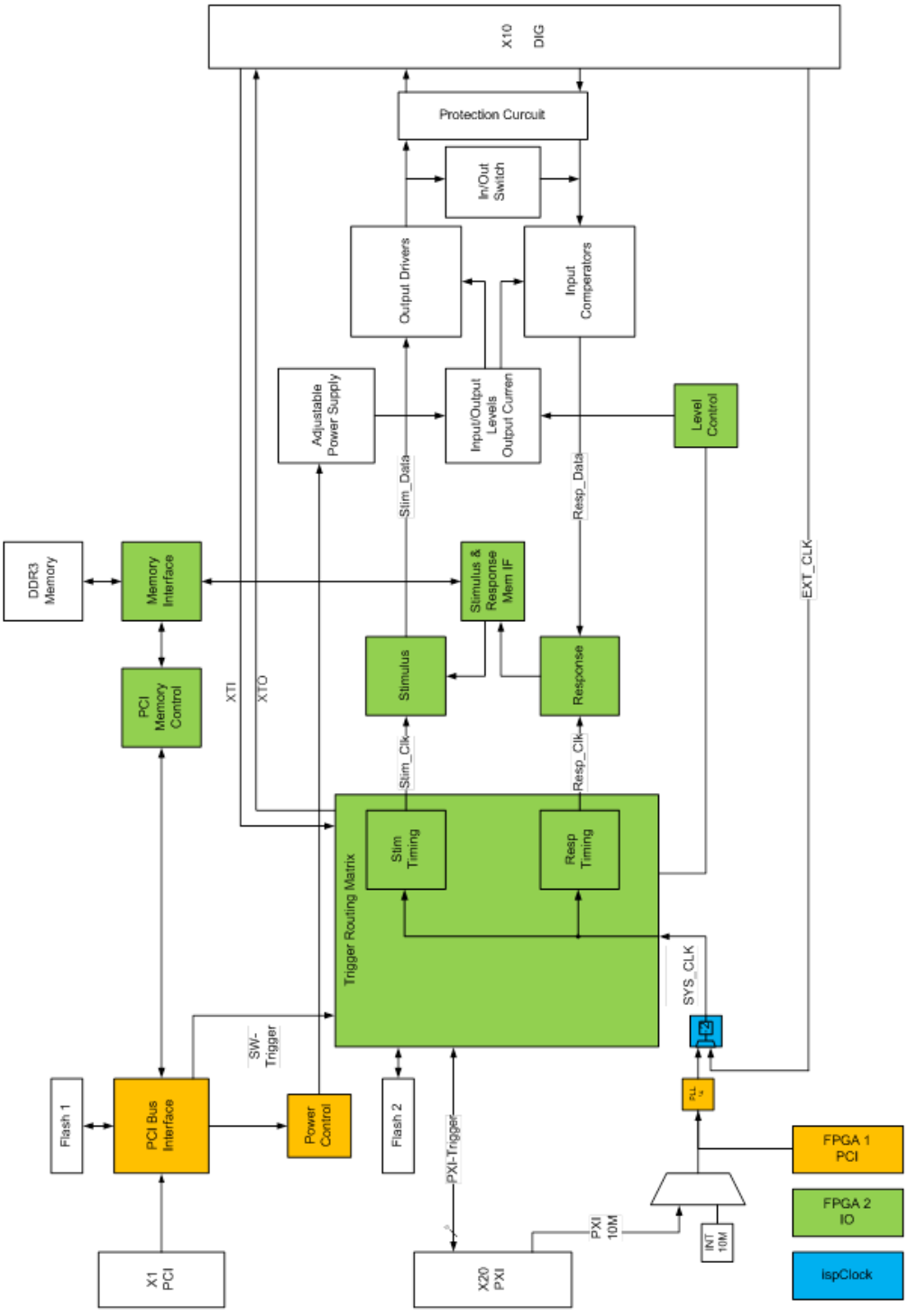


Bild 3-2: Detailliertes Blockschaltbild des Moduls R&S TS-PIO4

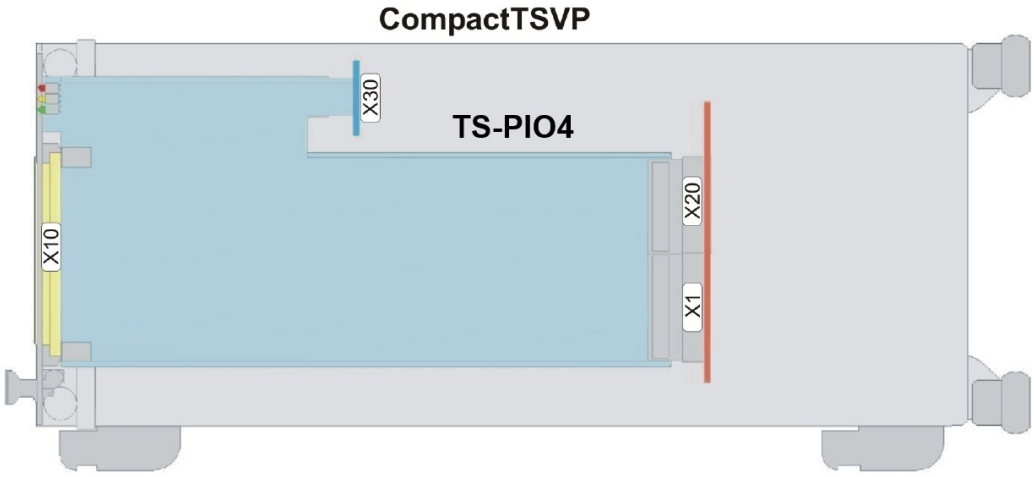


Bild 3-3: R&S TS-PIO4 im R&S CompactTSVP

4 Aufbau

4.1 Mechanischer Aufbau

Das Modul R&S TS-PIO4 ist als langes cPCI-Einsteckmodul und für den frontseitigen Einbau in den R&S CompactTSVP ausgeführt.

Die Höhe der Platine des Moduls beträgt 3 HE (134 mm). Um ein sicheres Einschieben in den R&S CompactTSVP zu gewährleisten, ist die Frontblende mit einem Führungsstift bestückt. Die Arretierung des Moduls geschieht mit den beiden Befestigungsschrauben der Frontblende.

Die frontseitige Schnittstelle X10 dient zum Anschluss von Prüflingen.

Der Stecker X30 dient nur der mechanischen Stabilität. Die Kontakte des Steckers sind nicht belegt.

Die Schnittstelle X20/X1 verbinden das Modul R&S TS-PIO4 mit der cPCI-Backplane/ PXI-Steuer-Backplane.

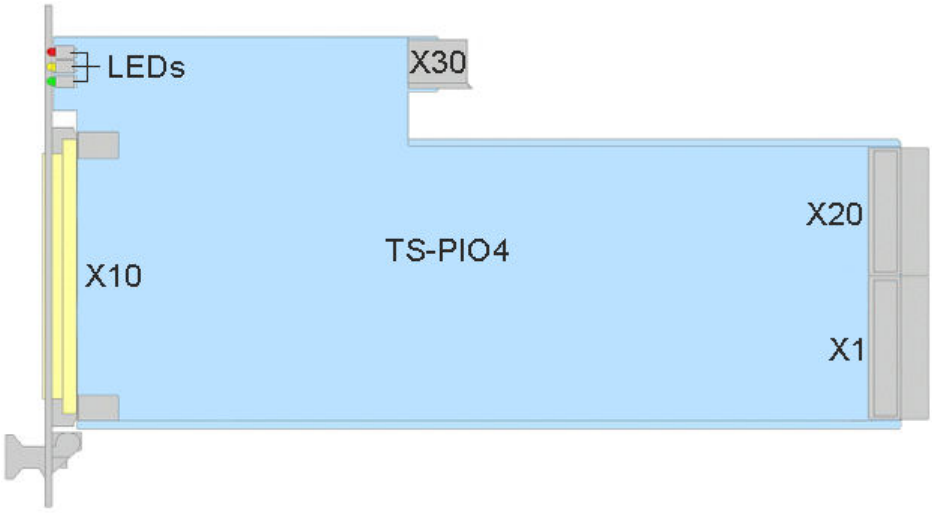


Bild 4-1: Anordnung der Schnittstellen am Modul R&S TS-PIO4

4.2 Schnittstellen

Kurzzeichen	Verwendung
X1	Backplane cPCI Bus
X10	Prüfling (UUT)
X20	Backplane Extension (PXI), Rear-I/O
X30	Analog Bus (nicht genutzt)

Eine detaillierte Schnittstellenbeschreibung mit Signalbelegung an den Steckverbindern befindet sich in .Kapitel 9, "Schnittstellenbeschreibung", auf Seite 80

4.3 Anzeigeelemente

Auf der Frontseite des R&S TS-PIO4 sind drei Leuchtdioden (LED) mit folgender Bedeutung angeordnet:

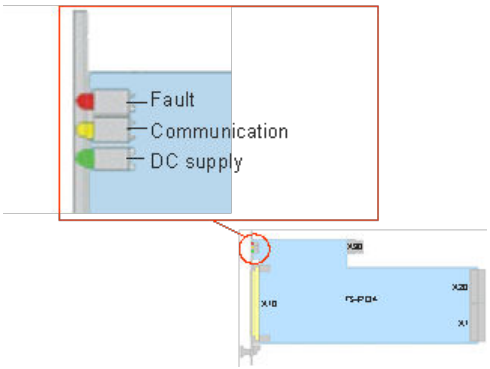


Bild 4-2: LED Anzeigen des R&S TS-PIO4

LED	Beschreibung
ERR (rot)	Fehlerzustand: Leuchtet, wenn nach dem Einschalten der Versorgungsspannung ein Fehler beim Einschalttest auf dem Modul R&S TS-PIO4 entdeckt wird.
COM (gelb)	Kommunikation: Leuchtet bei Datenverkehr über das Interface auf.
PWR (grün)	Versorgungsspannung: Leuchtet, wenn alle nötigen Versorgungsspannungen anliegen

5 Funktionsbeschreibung

5.1 Übersicht

5.1.1 Allgemeines

Das digitale Funktionstestmodul R&S TS-PIO4 stellt 8 Ports mit je 4 digitalen Eingangs- und Ausgangs-Pins zur Verfügung. Hierzu besitzt das Modul ungemultiplexte Digitalpins, d. h. hinter jedem Pin steht ein eigener Digitalkanal zum Stimulieren und Messen zur Verfügung. Des Weiteren können auf dem Modul die Ausgangsspannungen der Treiber sowie die Komparatorschwellen der Eingänge in bestimmten Grenzen frei programmiert werden. Damit lassen sich praktisch für jede Anwendung die passenden Logikpegel erzeugen. Alle Einstellungen sowie die Takterzeugung erfolgen auf dem Modul selbst, so dass keine weiteren Stimulus-Module notwendig sind.

Der nutzbare Pegelbereich hängt von der Konfiguration der Treiberreferenzen ab. Der gesamte Pegelbereich ist auf 4 Bereiche aufgeteilt. Die Treiberpegel werden aus der CPCI-Versorgung (+5 V und +3.3 V) erzeugt.

Die Applikationssoftware muss einen passenden Spannungsbereich einstellen, bevor die Ausgangsspannungen und Komparatorschwellen konfiguriert werden können. Der Gerätetreiber führt hier eine Plausibilitätsprüfung durch und verhindert, dass ungültige oder schädliche Werte eingestellt werden.

Tabelle 5-1: Spannungsbereiche

Range	VL[1]	VH[2]	CVL[3] / CVH[4]
3	+1,5 V....+6,0 V	+1,5 V....+10,0 V	+1.5 V....+7.1 V
2	-1,8 V....+2,7 V	-1,8 V....+10,0 V	-1.8 V....+7.1 V
1	-3,5 V....+1,0 V	-3,5 V....+10,0 V	-3.5 V....+7.1 V
0	-6,0 V.... -1,5 V	-6,0 V.... +8,0 V	-6.0 V....+5.1 V

[1]VL: Ausgangspegel Low

[2]VH: Ausgangspegel High

[3]CVL: Komparatorpegel Low

[4]CVH: Komparatorpegel High

Alle Ausgänge (OUTx) können bei Bedarf mit ihrem zugehörigen Eingang (INx) verbunden werden und somit stimulieren, messen und das Treiben von Signalen überwachen. Alle Treiberkanäle können in den hochohmigen Zustand (TRI-STATE) versetzt werden.

Die zeitliche Steuerung von Datenausgabe und Einlesen der Antwortsignale erfolgt auf dem Modul gesteuert durch FPGAs.

5.1.2 Ports

Bedingt durch Strukturen in der Hardware sind die digitalen Ein- und Ausgänge in so genannte Ports eingeteilt.

Das Modul R&S TS-PIO4 stellt 32 Ausgänge (OUT1 bis OUT32 und 32 Eingänge (IN1 bis IN32) zur Verfügung. Jeweils 4 Kanäle sind zu einem Port (PORT0 bis PORT7) zusammengefasst.

Viele Funktionen des Treibers beziehen sich auf diese Portstruktur.

Tabelle 5-2: Portstruktur

Port	Kanal OUTx / INx	Bitmaske
0	1…4	RSPIO4_MASK_PORT0
1	5…8	RSPIO4_MASK_PORT1
2	9…12	RSPIO4_MASK_PORT2
3	13…16	RSPIO4_MASK_PORT3
4	17…20	RSPIO4_MASK_PORT4
5	21…24	RSPIO4_MASK_PORT5
6	25…28	RSPIO4_MASK_PORT6
7	29…32	RSPIO4_MASK_PORT7

5.1.3 Speicher

5.1.3.1 Speicherverwaltung im Treiber

Die digitalen Kanäle können je nach Bedarf in unterschiedlichen Kombinationen für gleichzeitigen statischen und dynamischen Betrieb konfiguriert werden. Für die dynamischen Datensätze stehen deshalb verschiedene Formate (Datentypen) zur Verfügung.

Stimulus-Datensätze, die über `rspio4_LoadData` auf das Modul geladen werden sollen, werden zunächst in der Speicherverwaltung abgelegt. Für jeden Datensatz weist der Treiber eine ID zu, über die dieser Datensatz dann bei Bedarf identifiziert und zur Ausführung gebracht werden kann.

Erfolgt der Zugriff über die Treiberfunktionen nach IVI Digital, läuft dies im Hintergrund ab und braucht vom Anwender nicht beachtet zu werden. Es werden dann auch automatisch die passenden Datenformate gewählt.

Die Interpretation der Daten erfolgt im Treiber entsprechend des Modes, in den das Modul zuvor durch die Funktionen `rspio4_ConfigureStimMode()` und `rspio4_ConfigureRespMode()` versetzt wurde.

Aus Gründen der Abwärtskompatibilität kann das Attribut `RSPIO4_ATTR_PORT_HANDLING` mit Hilfe der Funktion `rspio4_ConfigurePortHandling` auf den Wert *RSPIO4_PORT_HANDLING_PDFT* gesetzt werden. In diesem Fall emuliert die Treibersoftware ein Modul mit 4 Ports mit je 8 Kanälen. Die Tristate-Information in den Datenstrukturen wird bei dieser Einstellung portspezifisch interpretiert.

Die Datei `rspio4.h` stellt diese Datentypen zur Verfügung. Für Details siehe `rspio4.h` sowie die Hilfedatei des Treibers.

Tabelle 5-3: Datentypen

Mode (siehe `rspio4.h`)	**Datentyp (siehe `rspio4.h`)**
RSPIO4_VAL_CTRL_STATIC	
RSPIO4_VAL_CTRL_4BIT	RSPIO4_DATA_4BIT (Stim. und Resp.)
RSPIO4_VAL_CTRL_8BIT	RSPIO4_DATA_8BIT (Stim. und Resp.)
RSPIO4_VAL_CTRL_16BIT	RSPIO4_DATA_16BIT (Stim. und Resp.)
RSPIO4_VAL_CTRL_32BIT	RSPIO4_DATA_32BIT (Stim. und Resp.)
RSPIO4_VAL_CTRL_4BIT_TRISTATE	RSPIO4_DATA_4BIT_TRISTATE (Stim.) RSPIO4_DATA_4BIT (Resp.)
RSPIO4_VAL_CTRL_8BIT_TRISTATE	RSPIO4_DATA_8BIT_TRISTATE (Stim.) RSPIO4_DATA_8BIT (Resp.)
RSPIO4_VAL_CTRL_16BIT_TRISTATE	RSPIO4_DATA_16BIT_TRISTATE(Stim.) RSPIO4_VAL_CTRL_16BIT (Resp.)
RSPIO4_VAL_CTRL_32BIT_TRISTATE	RSPIO4_DATA_32BIT_TRISTATE(Stim.) RSPIO4_VAL_CTRL_32BIT (Resp.)

5.1.3.2 Stimulus-Speicher

Auf dem Modul stehen 2MSample Speicher für Stimulusdaten zur Verfügung. D.h. es kann maximal ein Pattern Set mit 2 x 1024 x 1024 Daten Vektoren plus 2 x 1024 x 1024 Enable Vektoren definiert und auf das Modul geladen werden.

Die Datensätze gelangen zunächst über die Funktion `spio4_LoadData()` in den Stimulus-Speicher. Diese Funktion liefert eine ID zurück. Über diese ID kann dann der Datensatz im Stimulus-Speicher zur Ausführung gebracht werden.

Je nach verwendeter Funktion zum Ausführen eines Pattern Sets geschieht dies automatisch oder muss manuell ausgelöst werden. (z.B. IVI Digital Funktionen machen diesen Ablauf komplett im Hintergrund).

Mit Hilfe der Funktion `rspio4_DiscardData()` kann der Datensatz aus dem Stimulusspeicher entfernt werden.

5.1.3.3 Response-Speicher

Die aufgezeichneten Response Daten befinden sich in einem ebenfalls 2 MSample fassenden Speicher. Dieser enthält immer abwechselnd in je einem 32 Bit Wert die Ergebnisse der High Komparatoren und der Low Komparatoren.

Die Standardfunktionen bewerten diese Ergebnisse entsprechend dem eingestellten Komparatormodus

- COMP Fensterkomparator
- HYST Hysterese

Um bei Bedarf auch die Information bzgl. einer Eingangsspannung in einem verbotenen Bereich zu erhalten, wurden dem Treiber Funktionen hinzugefügt, welche die Validität der Daten jedes Kanals auswerten und zurückliefern. (Siehe auch Kapitel 5.3.3, "Konfiguration der Eingangs-Kanäle", auf Seite 30.)

5.1.4 Statischer Digitaltest

Beim statischen Digitaltest wird der Ablauf für das Anlegen von Pattern an den Ausgängen sowie das Einlesen der Eingänge vom Steuerrechner aus kontrolliert. Das hat zur Folge, dass die zeitliche Dauer der einzelnen Pattern sowie der Abtastzeitpunkt für die Messung bezogen auf den Patternbeginn nie genau vorausgesagt werden kann, da hier Einflüsse des Hostrechners und dessen Betriebssystem eingehen.

Der statische Digitaltest ist daher geeignet für die Überprüfung des logischen Zusammenspiels von Komponenten, zur Überprüfung von Spannungsschwellen und anderen Abläufen, wenn es dabei nicht auf die Verifikation bzw. das genaue Einhalten von zeitlichen Bedingungen ankommt.

Der statische Digitaltest kann bei R&S TS-PIO4 mittels IVI Digital Funktionen oder mit den Low Level Treiberfunktionen ausgeführt werden.

5.1.5 Dynamischer Digitaltest

Im einfachsten Fall ist die Patternperiode für Ausgabe (Stimulus) und Erfassung (Response) gleich. Stimulus und Response werden vom gleichen Trigger gestartet und laufen synchron zueinander ab.

Die Prüflingsantwort muss in der Regel gegenüber dem Patternbeginn verzögert gemessen werden. Die Verzögerung (Response Delay) ist zwischen 0 s und der Patternperiode einstellbar. Diese Verzögerung wird von der Anwendung so festgelegt, dass zum Zeitpunkt der Abtastung stabile Daten vom Prüfling her anliegen. Dieser Wert wird letztlich von den Verzögerungszeiten im Prüfling bestimmt. Wird die Verzögerung größer als die Patternperiode, so erfolgt die Abtastung dann schon innerhalb der nächsten Periode der Stimulus Pattern. Auch dies kann in bestimmten Applikationen sinnvoll sein.

Der Start des Tests erfolgt durch Software- oder Hardwaretrigger. Die Pattern werden mit einer festen Taktrate ausgegeben. Die Patterndauer ist die Zeit, während der ein

Pattern eines Pattern-Sets an den Ausgängen anliegt. Innerhalb dieser Zeit werden in der Regel auch die Antworten des Prüflings an den Eingängen aufgezeichnet.

Das „Response Delay“ ist der Zeitversatz zwischen dem Beginn eines Pattern und der Abtastung der Daten an den Eingängen. Die Patternrate ist der Kehrwert der Patterndauer.

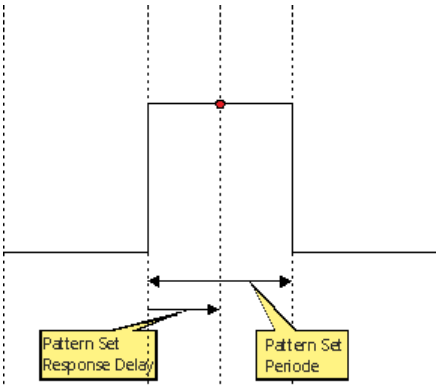


Bild 5-1: "Pattern Set Periode" und "Response Delay"

Ein Pattern-Set ist eine Menge von Pattern (Vektoren), die in einem Ablauf nach Empfang eines Triggerereignisses abgearbeitet werden. Die Abarbeitung wird durch eine Softwarefunktion oder durch einen Hardwaretrigger gestartet. Die Ausführung erfolgt in Echtzeit und ohne irgendwelche Einflüsse durch das Betriebssystem des Steuerrechners. Die Software kann abfragen, ob die Ausführung noch läuft, die laufende Ausführung abbrechen oder auf deren Ende warten.

5.2 Programmierung Digitaler Tests

TSVP ist eine offene Plattform hinsichtlich Hardware und Software. Die Software besteht typisch aus einer Anzahl von Treibern und Bibliotheken, die aus einem vom Anwender erstellten Testprogramm oder Sequencer aufgerufen werden.

Im Allgemeinen gibt es zwei Möglichkeiten auf TSVP Module zuzugreifen.

- Gerätetreiber
- High Level GTSL Bibliotheken

Gerätetreiberfunktionen bieten Zugriff auf alle Möglichkeiten der Hardware. Symbolische Namen und modulübergreifende Funktionalität werden hingegen nicht unterstützt.

Die Funktionen im Gerätetreiber bieten zwei Interfaces zum Programmieren von Digitaltests:

- Digitaltest mit Low Level Funktionen
- Digitaltest mit IVI Digital Funktionen

High Level Bibliotheken bieten eine standardisierte Programmierschnittstelle und ermöglichen gewisse Abstraktionen gegenüber der darunter liegenden Hardware, z.B. symbolische Signalnamen und die Behandlung mehrerer paralleler Module.

Zur Bedienung der Digitalfunktionen des Moduls steht die GTSL Bibliothek „DIO Manager“ (`DIOMGR.DLL`) zur Verfügung.

5.2.1 Digitaltests mit Gerätetreiberfunktionen

5.2.1.1 Initialisierung

- `rspio4_init`
- `rspio4_InitWithOptions`
- `rspio4_close`

5.2.1.2 Hilfsfunktionen

- `rspio4_reset`
- `rspio4_LockSession`
- `rspio4_UnlockSession`
- `rspio4_self_test`
- `rspio4_revision_query`

5.2.1.3 Fehlerabfragen

- `rspio4_error_query`
- `rspio4_GetErrorInfo`
- `rspio4_ClearErrorInfo`
- `rspio4_error_message`

5.2.1.4 Funktionen des IVI Switch Anteils

Es gibt nur die zwei Kanäle "GND" und "GNDNO" auf dem Modul R&S TS-PIO4, die mit den "IVI Switch Functions" geschaltet werden können. Dabei wird bei das GND-Relais bedient.

- `rspio4_Connect`
- `rspio4_Disconnect`
- `rspio4_DisconnectAll`
- `rspio4_GetPath`
- `rspio4_SetPath`
- `rspio4_CanConnect`
- `rspio4_IsDebounced`
- `rspio4_WaitForDebounce`

5.2.1.5 Digitaltest mit Low Level Treiberfunktionen

Um die Leistung der Karte optimal zu nutzen, bietet sich der Digitaltest mit Low Level Funktionen an.

Statischer Digitaltest mit Low Level Treiberfunktionen

- `rspio4_SetDoutState`
- `rspio4_SetDoutPort`
- `rspio4_ConnectInOut`
- `rspio4_GetDinState`
- `rspio4_GetDinHighAndLowComp`

Dynamischer Digitaltest mit Low Level Treiberfunktionen

Teilweise werden diese Funktionen auch zur Grundeinstellung bei dynamischen Tests nach IVI Digital benötigt (siehe Kapitel 7.3.4, "Statische Patternausgabe mit Low Level Treiberfunktionen", auf Seite 65).

- `rspio4_ConfigDinComparator`
- `rspio4_ConfigureStimMode`
- `rspio4_ConfigureRespMode`
- `rspio4_ConfigureStimTiming`
- `rspio4_ConfigureRespTiming`
- `rspio4_LoadData`
- `rspio4_LoadStimBuffer`
- `rspio4_AppendStimBuffer`
- `rspio4_ExecutePattern`
- `rspio4_AbortExecution`
- `rspio4_FetchPatternResponseData`
- `rspio4_FetchPatternResponseDataFull`
- `rspio4_DiscardData`

5.2.1.6 Digitaltest nach IVI Digital

IVI Digital ist ein Standard für die Ansteuerung digitaler Testinstrumente. Der Treiber bietet IVI Digital konforme Aufrufe. IVI Digital unterstützt sowohl den statischen als auch den dynamischen Digitaltest.

Die Verwendung der IVI Digital Funktionen erfordert hostseitig einen sehr hohen Rechenoverhead und ist relativ unflexibel bezüglich der Hardware, da IVI Digital eine Standardisierung ist und nicht auf leistungssteigernde Besonderheiten der Hardware eingehen kann. Es ist z.B. nicht möglich, nach statischen und dynamischen Kanälen aufzuteilen.

Funktionen für den Digitaltest nach IVI Digital

- `rspio4_ClearPattern`

- rspio4_ConfigureChannelOpcode
- rspio4_ConfigureMode
- rspio4_ConfigureGroupOpcode
- rspio4_ConfigureLargeGroupOpcode
- rspio4_CreatePattern
- rspio4_GetChannelName
- rspio4_GetChannelOpcode
- rspio4_GetGroupOpcode
- rspio4_ConfigureStaticResponseDelay
- rspio4_ExecuteStaticPattern
- rspio4_FetchStaticChannelOpcode
- rspio4_GetStaticChannelName
- rspio4_FetchStaticChannelListResults
- rspio4_FetchStaticChannelResult
- rspio4_FetchStaticChannelData
- rspio4_FetchStaticChannelListData
- rspio4_AbortPatternSet
- rspio4_BeginPatternSetLoading
- rspio4_ClearPatternSet
- rspio4_ConfigurePatternSetMaxTime
- rspio4_ConfigurePatternSetTiming
- rspio4_CreatePatternSet
- rspio4_ExecutePatternSet
- rspio4_FetchPatternSetResult
- rspio4_GetDynamicChannelName
- rspio4_GetDynamicPatternCount
- rspio4_FetchDynamicChannelOpcode
- rspio4_GetPatternSetCount
- rspio4_GetPatternSetExecutedPatternCount
- rspio4_GetPatternSetLoadedPatternCount
- rspio4_GetPatternSetName
- rspio4_InitiatePatternSet
- rspio4_LoadDynamicPattern
- rspio4_WaitUntilPatternSetComplete
- rspio4_FetchDynamicChannelListResults
- rspio4_FetchDynamicChannelListPatternResults
- rspio4_FetchDynamicChannelResult
- rspio4_FetchDynamicPatternResult
- rspio4_FetchDynamicPatternListResults
- rspio4_FetchDynamicChannelData

- `rspio4_FetchDynamicChannelListData`
- `rspio4_FetchDynamicChannelListPatternData`

Statischer Digitaltest nach IVI Digital

Auch hier sind statische Tests möglich, also Tests, bei denen die Steuerung vom Hostrechner durchgeführt wird und die dadurch ein nicht vollständig bestimmtes Zeitverhalten haben. Siehe Beispiel Kapitel 7.3.3, "Statische Patternausführung mit IVI Digital", auf Seite 58.

Dynamischer Digitaltest nach IVI Digital

Wie man im Beispiel zum statischen Digitaltest nach IVI Digital sieht, wird für jeden Kanal ("OUT1" … "OUT32", "IN1" ... "IN32") ein sogenannter Op Code erzeugt. Dieser steuert, ob ein Kanal High oder Low treiben soll oder ggfs. nach Tristate schalten soll. Das so erzeugte Pattern wird dann sofort ausgeführt und liegt an den Ausgängen an.

Beim dynamischen Digitaltest nach IVI Digital werden die erzeugten Pattern in ein Pattern-Set gespeichert.

Nach dem Festlegen eines Timings wird das komplette Pattern-Set zur Ausführung gebracht.

5.2.2 Digitaltest mit DIO Manager

Jedes Pattern-Set wird in einer eigenen Datei abgespeichert. Diese Datei kann manuell mit einem Texteditor bearbeitet werden.

Die Datei mit dem Pattern-Set wird zur Laufzeit auf ein bzw. mehrere R&S TS-PIO4 Module geladen und dann ausgeführt. Die Ergebnisse können über Funktionsaufrufe zurückgelesen werden. Alternativ können die Ergebnisse auch in einer Datei mit gleichem Format abgespeichert werden. So können auch Unterschiede zwischen erwartetem und gemessenem Verhalten über einen Dateivergleich leicht erkannt werden.

Siehe auch Beispiel Kapitel 7.3.1, "Digitaltest mit der Bibliothek DIO Manager", auf Seite 44.

5.2.2.1 Dateiformat

Das Dateiformat wurde ursprünglich vom Altera Quartus Waveform Generator definiert.

```
GROUP CREATE bus = bus[7] bus[6] bus[5] bus[4] bus[3] bus[2] bus[1] bus[0];
INPUTS N14CR0805170 E_COM E_EC1 E_EC2 E_EC3 bus;
OUTPUTS N13HCPN31500 N13HCPM31500 N13HCPO31500 N13HCPL31500;
UNIT ns;
RADIX HEX;
PATTERN
    0.0> 0 0 0 0 0 00 = X X X X
 1000.0> 1 0 0 0 0 01 = X X X X
```

```
 2000.0> Z Z Z Z Z 02 = X X X X
 3000.0> 0 0 0 0 0 03 = X X X X
 4000.0> 1 0 0 0 0 04 = X X X X
 5000.0> Z Z Z Z Z 05 = X X X X
 6000.0> 0 0 0 0 0 06 = 1 X X X
 7000.0> 0 1 0 0 0 07 = 0 X X X
 8000.0> Z Z Z Z Z 08 = X X X X
 9000.0> 0 0 0 0 0 09 = X 1 X X
10000.0> 0 0 0 0 1 0A = X 0 X X
11000.0> Z Z Z Z Z 0B = X X X X
12000.0> 0 0 0 0 0 0C = X X 1 X
13000.0> 0 0 0 1 0 0D = X X 0 X
14000.0> Z Z Z Z Z 0E = X X X X
15000.0> 0 0 0 0 0 0F = X X X 1
16000.0> 0 0 1 0 0 10 = X X X 0
17000.0> Z Z Z Z Z 11 = X X X X
18000.0> X X X X X X = X X X X
;
```

```
GROUP CREATE bus = bus[7] bus[6] bus[5] bus[4] bus[3] bus[2]
bus[1] bus[0];
```

Diese Anweisung ist optional. Hiermit werden Gruppen von Pins zu Bussen zusammengefasst. Es können auch mehrere Busse definiert werden.

```
INPUTS N14CR0805170 E_COM E_EC1 E_EC2 E_EC3 bus;
```

```
OUTPUTS N13HCPN31500 N13HCPM31500 N13HPCO31500 N13HCPL31500;
```

Die INPUTS und OUTPUTS Anweisungen definieren die Kanalnamen oder Gruppennamen. Wichtig ist dabei: INPUTS sind Eingänge des UUT und entsprechen daher den OUT1 ... OUT32 des R&S TS-PIO4 Moduls. OUTPUTs sind UUT Ausgänge und entsprechen daher den Eingängen IN1 ... IN32 des Moduls.

```
UNIT ns;
```

Definiert die Einheit der Zeitstempel in der Datei.

```
RADIX HEX;
```

Definiert die Basis für die Werte bei Bussen. Der DIO Manager unterstützt nur HEX.

```
PATTERN
```

```
0.0> 0 Z Z Z Z 00 = X X X X;
```

.....

Alle auf PATTERN folgenden Zeilen definieren dieses Pattern-Set. Jede Zeile stellt ein Pattern zu einer bestimmten Zeit dar. Die letzte Zeile wird ignoriert und dient nur als Endekennung. Alle Kanäle sind X. Die Zeitstempel müssen nicht äquidistant sein.

Value	for INPUTS	for OUTPUTS
0	drive low	expect low
1	drive high	expect high

Value	for INPUTS	for OUTPUTS
Z	high impedance	(not used)
X	(not used)	don't care

Für Gruppen wird ein hexadezimaler Wert eingegeben. Die Sonderfälle X und Z geben an, dass alle Kanäle der betreffenden Gruppe diesen Zustand annehmen.

5.2.2.2 Konfiguration DIO Manager

Die Waveform-Datei enthält die logischen Namen der Eingänge und Ausgänge. Die Zuweisung von physikalischen Kanälen auf den R&S TS-PIO4 Modulen zu logischen Namen erfolgt in der Datei `application.ini`.

```
[bench->823916]
DIODevice = device->pio4
DIOChannelTable = io_channel->823916

[io_channel->823916]

E_COM          = pio4!out1
E_EC1          = pio4!out2
E_EC2          = pio4!out3
E_EC3          = pio4!out4
N14CR0805170   = pio4!out5

N13HCPL31500   = pio4!in1
N13HCPM31500   = pio4!in2
N13HCPN31500   = pio4!in3
N13HCPO31500   = pio4!in4

bus0           = pio4!out20
bus1           = pio4!out21
bus2           = pio4!out22
bus3           = pio4!out23
bus4           = pio4!out24
bus5           = pio4!out25
bus6           = pio4!out26
bus7           = pio4!out27
```

Wie alle GTSL Bibliotheken wird der DIO Manager in einer `application.ini` Datei konfiguriert. Das Schlüsselwort `DIODevice` verweist auf das R&S TS-PIO4 Modul im `physical.ini`. Wenn mehr als ein Digitalmodul verwendet werden soll, dann wird ein DIODevice2, DIODevice3, … hinzugefügt.

Die „DIO Channel Table“ verweist auf die zugehörige Tabelle in der Datei.

Die Channel Table enthält auf der linken Seite die logischen Namen und rechts die Verknüpfung mit den physikalischen Kanälen auf den einzelnen Modulen.

Kanalnamen für Gruppen (z.B. Busse) haben keine eckigen Klammern mehr. Hier heißen sie z.B. "bus0" während sie in der Waveform Datei "bus[0]" heißen.

Kanalnamen sind nicht case-sensitiv.

5.2.2.3 Struktur eines Testprogramms

Tabelle 5-4: Diese Funktionen müssen einmal zu Beginn des Testprogramms aufgerufen werden.

Funktion	Anmerkungen
`RESMGR_Setup`	Aufruf des Resource Manager
`DIOMGR_Setup`	Aufruf des DIO Manager und Initialisierung des TS-PDFT Moduls
`DIOMGR_ConfigureStimulus` `DIOMGR_ConfigureResponse`	Konfiguration logischer Levels für Ein- und Ausgänge
`DIOMGR_LoadWaveform`	Laden eines Pattern Sets aus einer Datei in den Speicher.
`DIOMGR_ConfigurePatternSetTiming`	Definition des Timings eines Pattern Sets

Tabelle 5-5: Diese Funktionen werden typisch in einer Schleife über mehrere UUTs aufgerufen.

Funktion	Anmerkungen
`DIOMGR_ExecutePatternSet`	Ausführen eines Pattern Sets und Rückgabe des Ergebnisses (pass/fail)
Optionale Funktionen:	Werden im allgemeinen nur aufgerufen, wenn das Pattern Set fehlerhaft ist
`DIOMGR_SaveWaveform`	Speichern der Ergebnisse als TBL Datei
`DIOMGR_GetPatternSetExecutedPatternCount`	Auslesen der ausgeführten Patterns
`DIOMGR_GetPatternSetFailedPatternCount`	Auslesen der fehlgerhaften Patterns
`DIOMGR_GetPatternSetFailedChannelCount`	Auslesen der fehlerhaften Kanäle
`DIOMGR_GetPatternSetFailedChannelNames`	Auslesender fehlerhaften Kanäle als Liste von Namen (comma-separated)
`DIOMGR_GetPatternSetChannelData`	Auslesen der aktuellen Response Message (measured data) für einen Kanal
`DIOMGR_GetPatternSetChannelResults`	Auslesen der Ergebnisse für einen Kanal (pass/fail)

Tabelle 5-6: Nachdem alle Tests ausgeführt wurden, dienen diese Funktionen zum Aufräumen.

Funktion	Anmerkungen
`DIOMGR_Cleanup`	Beenden des DIO Manager
`RESMGR_Cleanup`	Beenden des Resource Manager

5.2.2.4 Ports

Das R&S TS-PIO4 Modul ist in 8 Ports zu je 4 Kanälen aufgeteilt. Jeder Port kann auf unterschiedliche Treiber- und Sensorpegel eingestellt werden. Wenn unterschiedliche Pegel in einem Programm verwendet werden, müssen `DIOMGR_ConfigureStimulus()` und `DIOMGR_ConfigureResponse()` je einmal für jede „Logikfamilie“ aufgerufen werden und die entsprechenden Pegel einstellen.

5.2.2.5 Laden der Waveform-Datei

Die Waveform Datei definiert einen Pattern-Set unter Angabe von Zeitstempeln. Diese Zeitstempel müssen nicht äquidistant sein. Normalerweise wird immer dann eine neue Zeile angelegt, wenn sich mindestens ein Signal ändert.

```
PATTERN
    0.0> 0 0 0 0 0 00 = X X X X
 1000.0> 1 0 0 0 0 01 = X X X X
 1001.0> 1 1 0 0 0 01 = X X X X
 1050.3> 1 1 1 0 1 01 = 1 1 1 1
 2100.0> Z Z Z Z Z FF = X X X X
etc.
```

Wenn die Waveform auf das R&S TS-PIO4 Modul geladen wird, muss das Timing auf einen festen Rahmen umgerechnet werden, der durch die Pattern Set Periode des Moduls bestimmt ist. Die Ladefunktion verwendet den Parameter „timeGrid“ um die Abtastwerte der Waveform in einem festen Zeitraster zu erzeugen. Jeder Abtastwert wird dann in ein Pattern umgerechnet und im Speicher des Moduls abgelegt.

Wenn ein „timeGrid“ von 500 ns im obigen Beispiel eingestellt wird, ergeben sich folgende Abtastwerte:

Pattern	Time	Inputs	Outputs
1	0 ns	0 0 0 0 0 00	X X X X
2	500 ns	0 0 0 0 0 00	X X X X
3	1000 ns	1 0 0 0 0 01	X X X X
4	1500 ns	1 1 1 0 1 01	1 1 1 1
5	2000 ns	1 1 1 0 1 01	1 1 1 1
6	2500 ns	Z Z Z Z Z FF	X X X X

Einige Pattern werden wiederholt (1 und 2, 4 und 5) und einige werden nicht erkannt, da sie zu kurz sind und zwischen den Abtastzeitpunkten liegen (z.B. Pattern am Zeitstempel 1001.0).

Zeitstempel und Parameter „timeGrid“ müssen nicht zwangsläufig die tatsächliche Ausführungsgeschwindigkeit darstellen. Die tatsächliche Ausführungsgeschwindigkeit wird über die Funktion `rspio4_ConfigurePatternSetTiming()` eingestellt. Das bedeutet auch, dass die Waveform Datei nicht modifiziert werden muss, wenn der Test mit einer anderen Pattern Rate ablaufen soll.

Beim Schreiben der Waveform Datei sollte man Folgendes beachten:

- Äquidistante Zeitstempel verwenden
- Zeitstempel sollen das echte Timing wiederspiegeln, also genau das Timing, das für den Test verwendet wird.
- Abstand zwischen den Zeitstempeln als Wert für den „timeGrid“ Parameter verwenden.
- Abstand zwischen den Zeitstempeln als „Pattern Set Period“ verwenden.

5.2.2.6 Ausführung Pattern Set

Das Pattern Set kann synchron oder asynchron ausgeführt werden. Im ersten Fall kehrt die Funktion `DIOMGR_ExecutePatternSet()` erst zurück, wenn die Ausführung abgeschlossen ist (oder die maximale Zeit überschritten wurde). Im zweiten Fall wird die Ausführung gestartet bzw. der Trigger armiert und die Funktion kehrt zurück ohne auf die eigentliche Ausführung des Pattern Sets zu warten. Die Funktion `DIOMGR_WaitUntilPatternSetComplete()` kann benutzt werden, um auf das Ende der Ausführung zu warten.

5.3 Konfiguration der Digitalkanäle

Das Modul bietet eine Reihe von konfigurierbaren Parametern. Dieser Abschnitt beschreibt die Konfigurationsmöglichkeiten und führt die entsprechenden Treiberfunktionen auf. Für Details zu den Treiberfunktionen sei auf die GTSL Hilfe der Treibersoftware verwiesen.

5.3.1 Einstellung Spannungsbereich

Die Anwendung muss einen Bereich einstellen, der die gewünschten Spannungen für die High und Low Pegel der Ausgänge sowie die erforderlichen Komparatorschwellen für die Eingänge enthält. Jedem Port kann ein eigener Spannungsbereich zugeordnet werden..

Das Umschalten der Spannungsbereiche verursacht lange Einschwingzeiten auf dem Modul, die im Gerätetreiber berücksichtigt werden. Beim Verstellen des Spannungsbereichs werden auch alle Treiber hochohmig geschaltet.

Die Spannungsbereiche sind in Tabelle Spannungsbereiche aufgelistet.

- `rspio4_ConfigurePortVoltageRange`

5.3.2 Konfiguration der Stimulus-Kanäle

Je Port können Einstellungen für die Ausgangsspannungen programmiert werden.

- `rspio4_ConfigureStimPort`

Die Einstellungen erfolgen für einen oder mehrere Ports abhängig von den Parametern im Funktionsaufruf. Die Strombegrenzung gilt immer für die summierten Ausgangsströme aller 4 Treiber eines Ports.

Durch den Ausgangswiderstand des Treibers und die Widerstände in den Ausgangsschutzschaltungen (zusammen ca. 30 Ω) entsteht bei Stromfluss ein Spannungsabfall, der bei der Pegeleinstellung gegebenenfalls berücksichtigt werden muss.

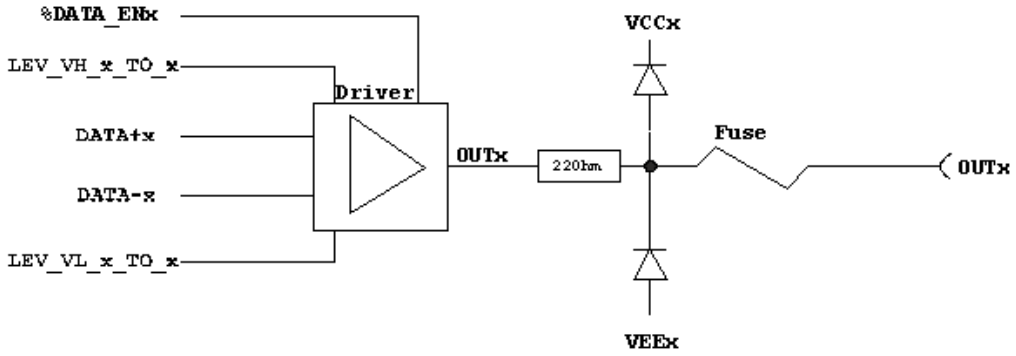


Bild 5-2: Konfiguration der Stimulus-Kanäle

5.3.3 Konfiguration der Eingangs-Kanäle

- `rspio4_ConfigureRespPort`

Die Messkanal-Eingänge haben vor den Komparatoren eine Schutzschaltung, die wie folgt aufgebaut ist.

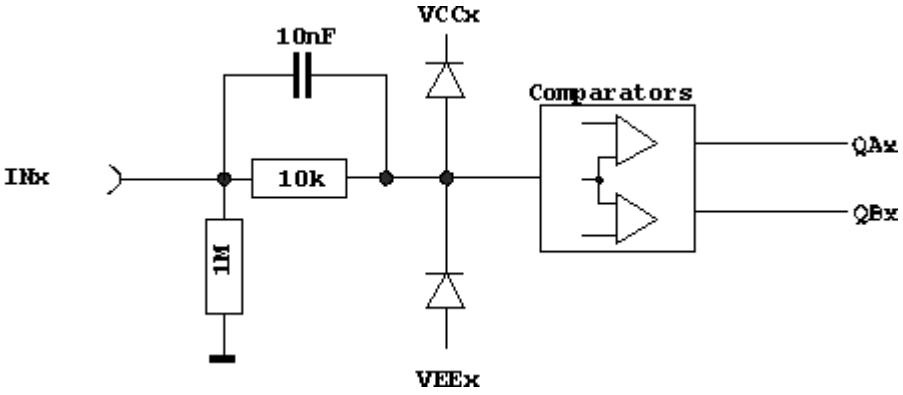


Bild 5-3: Konfiguration der Eingangs-Kanäle

Jeder Eingang ist auf zwei Komparatoren geführt, deren Ansprechschwelle einstellbar ist. Dadurch lässt sich eine Hysterese bei der Bewertung von Signalen realisieren. Die Schwellen können mit Hilfe der Treiberfunktion `rspio4_ConfigureRespPort` gesetzt werden. Für jeden Port können damit individuelle Werte eingestellt werden.

Das Ergebnis der Signalbewertung eines Kanals ist **1**, wenn der Eingangspegel größer als die Schwelle für den High-Pegel ist.

Das Ergebnis der Signalbewertung eines Kanals ist **0**, wenn der Eingangspegel kleiner als die Schwelle für den Low-Pegel ist.

Befindet sich der Eingangspegel zwischen den Schwellwerten, wird „Forbidden Zone“ als Ergebnis geliefert. Über die Low Level Treiberfunktion `rspio4_GetDinHighAndLowComp` können die Zustände beider Komparatoren ausgelesen werden.

Ist der Komparator auf den Modus „COMP“ eingestellt, arbeiten die beiden Komparatoren als Fensterkomparator. Es wird eine **1** erkannt, wenn der Eingangspegel zwischen den Schwellen liegt und eine **0** wenn er unter der Schwelle für Low oder über der Schwelle für High liegt.

5.3.4 Zeiteinstellungen für die Datenausgabe

Diese Einstellung ist sowohl bei dynamischen Tests nach IVI Digital als auch bei dynamischen Tests mit den Low Level Treiberfunktionen erforderlich.

Zur Konfiguration des zeitlichen Verhaltens der Stimuluskanäle bei der Ausgabe eines Pattern-Sets wird mit der Funktion `rspio4_ConfigureStimTiming` die Triggerverzögerung, die Patterndauer sowie die Anzahl der auszugebenden Pattern eingestellt (Trigger Delay, Pattern Period, Loop Count). Die Triggerverzögerung ist die Wartezeit zwischen dem Triggerereignis und der Ausgabe des ersten Patterns. Die Patterndauer ist die Zeit, während der ein einzelnes Pattern anliegt.

5.3.5 Zeiteinstellungen für die Datenerfassung

Diese Einstellung ist sowohl bei dynamischen Tests nach IVI Digital als auch bei dynamischen Tests mit den Low Level Treiberfunktionen erforderlich.

Für die Erfassung der Eingangssignale können mit der Funktion `rspio4_ConfigureRespTiming` die Zeitparameter eingestellt werden. Wie bei der Datenausgabe stehen auch hier die Triggerverzögerung, die Patterndauer sowie die Anzahl der einzulesenden Pattern (Response Trigger Delay, Response Pattern Period, Loop Count) als Parameter zur Verfügung.

Die Triggerverzögerung bestimmt die Wartezeit zwischen dem Triggerereignis und der ersten Abtastung. Das Response Trigger Delay wird so eingestellt, dass die Prüflingsdaten zu diesem Zeitpunkt stabil am Eingang anstehen. Im typischen Fall, in dem Stimulus Pattern Period und Response Pattern Period gleich sind, bedeutete ein Delay von 0.0, in jedem Pattern unmittelbar am Beginn des Pattern abgetastet würde, ein Delay größer als die Pattern Periode würde innerhalb des nächsten oder eines anderen Pattern abtasten.

Beispiel:

Stimulus Pattern Period: 50 ns

Response Pattern Period: 50 ns

Stim Trigger Delay: 0.0 ns

Resp. Trigger Delay: 25.0 ns

Die Ausgänge werden zum Zeitpunkt 0 geändert. 25.0 ns später, hier also in der Mitte des Patterns, werden die Daten abgetastet. Hier wird also vorausgesetzt, dass der Prüfling schon nach weniger als 25.0 ns stabile Daten ausgibt.

5.3.6 Konfiguration der Datenbreite im Dynamischen Betrieb

- `rspio4_ConfigureStimMode`
- `rspio4_ConfigureRespMode`

Die digitalen Kanäle können je nach Bedarf in unterschiedlichen Kombinationen für gleichzeitigen statischen und dynamischen Betrieb konfiguriert werden. Dadurch lassen sich z.B. Steuersignale, die während ganzer Testsequenzen auf konstanten Pegel sind, konfigurieren. Sie müssen dann nicht mehr Teil der dynamischen Daten sein. Dadurch vereinfacht sich auch die Erstellung der Pattern-Sets.

Diese Art von Konfiguration ist nur möglich, wenn die Programmierung über die Low Level Treiberfunktionen erfolgt. Bei Verwendung der Treiberfunktionen nach IVI Digital ist die dynamische Breite immer 32 Bit.

Die möglichen bzw. gewählten Konstellationen müssen jedoch bei der Adapterverdrahtung beachtet werden.

Möglich sind folgende Konfigurationen (siehe auch `rspio4.h`).

Mode	Konfiguration
`RSPIO4_VAL_CTRL_STATIC`	Statischer Betrieb
`RSPIO4_VAL_CTRL_4BIT`	4 dynamische DOUT oder DIN: OUT1..4 / IN 1..4 kein Tristate
`RSPIO4_VAL_CTRL_8BIT`	8 dynamische DOUT oder DIN: OUT1..8 / IN 1..8 kein Tristate
`RSPIO4_VAL_CTRL_16BIT`	16 dynamische DOUT oder DIN: OUT1..16 / IN 1..16 kein Tristate
`RSPIO4_VAL_CTRL_32BIT`	32 dynamische DOUT oder DIN: OUT1..32 / IN 1..32 kein Tristate
`RSPIO4_VAL_CTRL_4BIT_TRISTATE`	4 dynamische DOUT oder DIN: OUT1..4 / IN 1..4 Tristate Information in Daten
`RSPIO4_VAL_CTRL_8BIT_TRISTATE`	8 dynamische DOUT oder DIN: OUT1..8 / IN 1..8 Tristate Information in Daten

Mode	Konfiguration
RSPIO4_VAL_CTRL_16BIT_TRISTATE	16 dynamische DOUT oder DIN: OUT1..16 / IN 1..16 Tristate Information in Daten
RSPIO4_VAL_CTRL_32BIT_TRISTATE	32 dynamische DOUT oder DIN: OUT1..32 / IN 1..32 Tristate Information in Daten

Im IVI Digital konformen Betrieb wird immer der Modus RSPIO4_VAL_CTRL_32BIT_TRISTATE verwendet.

5.3.7 Leistungsaufnahme

5.3.7.1 Abschätzung der Leistungsaufnahme

Die gesamte Leistungsaufnahme des R&S TS-PIO Moduls sollte 40 W nicht überschreiten und hängt von der Betriebsart der Ein- und Ausgangskanäle ab.

In der folgenden Beschreibung wird die Berechnung der maximalen Leistungsaufnahme für die am häufigsten verwendeten Betriebsbereiche erläutert.

VH = Ausgangspegel High

VL = Ausgangspegel Low

Bereich 3 mit den Einstellungen VL = +1.5 V...+6.0 V und VH = +1,5 V...+10.0 V

Bereich 2 mit den Einstellungen VL = -1.8 V...+2.7 V and VH = -1.8 V...+10.0 V

Bereich 1 mit den Einstellungen VL = -3.5 V...+1.0 V and VH = -3.5 V...+10.0 V

Bereich 0 mit den Einstellungen VL = -6.0 V...-1.5 V and VH = -6.0 V...+8.0 V

Ausgangskanäle

Tabelle 5-7: Maximale Leistungsaufnahme pro Gruppe (bestehend aus 4 Kanälen)

Bereich	Leistungsaufnahme in einer Gruppe [W]	Maximale Leistung in einer Gruppe mit vier Kanälen
3	#CH * 0.021 * f[MHz] * (0.421 + 0.233 * U_swing)	< 3.9 W
2	#CH * 0.036 * f[MHz] * (0.548 + 0.172 * U_swing)	< 3.9 W
1	#CH * 0.047 * f[MHz] * (0.605 + 0.143 * U_swing)	< 3.9 W
0	#CH * 0.051 * f[MHz] * (0.609 + 0.137 * U_swing)	< 3.9 W

#CH: Anzahl der aktiven Kanäle in einer Gruppe mit vier Kanälen

f: IO Umschaltfrequenz (= 1/2 Sample Rate), Tastverhältnis 50%, MHz

U_swing : Spannungsschwankung des IO Signals (VH-VL), V

Beispiel:

Vier aktive Kanäle in einer Gruppe mit einer Umschaltfrequenz von 20 MHz (entspricht einer Sample Rate von 40 MSamples), VL = +1.5 V und VH = +4.5 V.

- U_swing = VH - VL = 4.5 V - 1.5 V = **3.0 V**
- Die Werte für VL und VH können in den Bereichen 2 und 3 benutzt werden

Berechnung der Leistungsaufnahme in einer Gruppe:

- *Bereich 3:* 4 * 0.021 * 20 * (0.421 + 0.233 * 3.0) = 1.88 W => **OK** (< 3.9 W)
- *Bereich 2:* 4 * 0.036 * 20 * (0.548 + 0.172 * 3.0) = 3.06 W => **OK** (<3.9 W)

Aufgrund der beschränkten Leistungsaufnahme der Gruppe ist der Bereich 3 zu verwenden.

Tabelle 5-8: Maximale Leistungsaufnahme pro Modul

Bereich	Leistungsaufnahme der aktiven IO Kanäle [W]	Maximale Leistungsaufnahme eines Moduls [W]
3	#GRP * #CH * 0.021 * f[MHz] * (0.421 + 0.233 * U_swing) + ∑P_load	< 25 W
2	#GRP * #CH * 0.036 * f[MHz] * (0.548 + 0.172 * U_swing) + ∑P_load	< 22 W
1	#GRP * #CH * 0.047 * f[MHz] * (0.605 + 0.143 * U_swing) + ∑P_load	< 20 W
0	#GRP * #CH * 0.051 * f[MHz] * (0.609 + 0.137 * U_swing) + ∑P_load	< 19 W

#GRP: Anzahl der aktiven Gruppen (eine Gruppe besteht aus max. vier Kanälen), max. 8 Gruppen

#CH: Anzahl der aktiven Kanäle in einer Gruppe mit vier Kanälen

f: IO Umschaltfrequenz (= 1/2 Sample Rate), Tastverhältnis 50%, MHz

U_swing: Spannungsschwankung des IO Signals (VH-VL), V

Beispiel:

Vier aktive Kanäle und 8 aktive Gruppen mit einer Umschaltfrequenz von 20 MHz (entspricht einer Sample Rate von 40 MSamples), VL = +1.5 V und VH = +4.5 V.

- U_swing = VH - VL = 4.5 V - 1.5 V = **3.0 V**
- Die Werte für VL und VH können in den Bereichen 2 und 3 benutzt werden

Berechnung der Leistungsaufnahme für ein Modul:

- *Bereich 3:* 8 * 4 * 0.021 * 20 * (0.421 + 0.233 * 3.0) = 15.05 W => **OK** (< 25 W)
- *Bereich 2:* 8 * 4 * 0.036 * 20 * (0.548 + 0.172 * 3.0) = 24.51 W => **N.OK** (> 22W)

In diesem Fall kann nur der Bereich 3 verwendet werden.

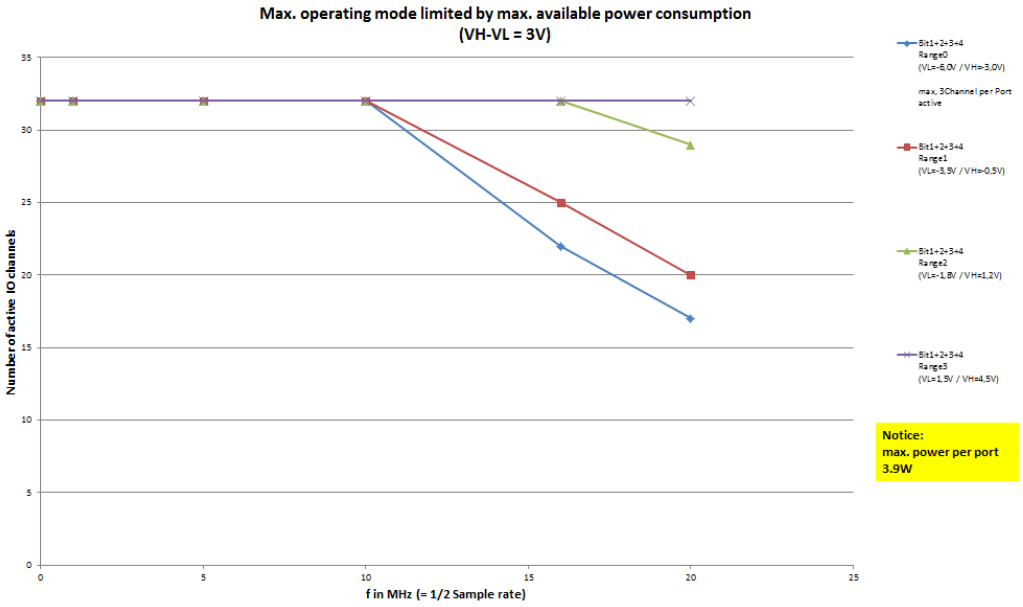


Bild 5-4: Max. Betriebsbereich in Abhängigkeit der maximalen Leistungsaufnahme

Eingangskanäle

Empfangskanäle (Comparators) nehmen beim Umschalten in Abhängigkeit von Betriebsbereich und Frequenz zusätzlich Verlustleistung auf.

Die folgende Tabelle gibt die ungefähre zusätzliche Leistungsaufnahme für die Betriebsbereiche bei folgenden Ausgangswerten an:

- Max. Umschaltfrequenz: 20 MHz
- Spannungsschwankung: 3.3 V
- Anzahl der Kanäle: 32 Eingangskanäle

Range	Zusätzliche Leistungsaufnahme (typ.)
3	10.0 W
2	5.5 W
1	12.5 W
0	13.5 W

Gesamte Leistungsaufnahme

Zur Abschätzung der gesamten Leistungsaufnahme müssen die Ein- und Ausgangskanäle berücksichtigt werden.

5.3.7.2 Schutzmechanismus

Wird während des Betriebs die maximal erlaubte Leistungsaufnahme überschritten, werden die IOs durch eine Schutzschaltung deaktiviert. Der Aufruf einer Treiberfunk-

tion führt in diesem Fall zu einem Fehler. Mit der Funktion `rspio4_reset` können die IOs reaktiviert werden.

5.4 Triggerung und Ablaufsteuerung

5.4.1 Triggereinheiten

Auf dem Modul sind zwei Triggereinheiten vorhanden. Die Triggereinheiten steuern die Ausgabe von Stimulus Daten sowie das Einlesen der Daten an den digitalen Eingängen. Triggereinheit IT1 ist zuständig für die Ausgabe der Stimulus Daten, Triggereinheit IT2 ist verantwortlich für das Einlesen der Response Daten.

Die Triggereinheiten können auf verschiedene Eingangsbedingungen konfiguriert werden. Außerdem kann konfiguriert werden, welches Ausgangssignal sie erzeugen sollen und wohin dieses geroutet wird.

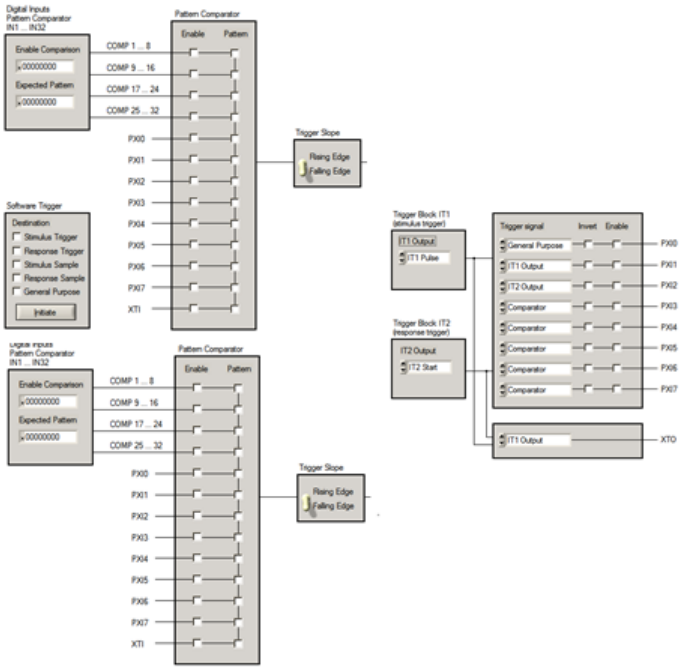


Bild 5-5: Aufbau der Triggereinheiten für Stimulus und Response

Eingänge der Trigger Einheit:

Auf der Input Seite wird mit der Funktion `rspio4_ConfigHWTriggerInput` konfiguriert, ob und welche Eingänge (PXI0 … PXI7, XTI, 4 Ausgänge des Digital Inputs Pattern Comparator) mit welchem logischen Zustand als Hardware Trigger verwendet werden sollen und welche Flanke verwendet wird.

Die Funktion `rspdft_ConfigDinComparator` erlaubt die Einstellung des Digital Inputs Pattern Comparators.

Alternativ kann die Unit auch über einen Software Trigger ausgelöst werden (z.B. startet die Funktion `rspio4_executePattern` die Abarbeitung eines Pattern Sets mittels eines Software-Triggers über einen Aufruf der Funktion `rspio4_InitiateSWTrigger`)

Wurde die Trigger Einheit durch ein Ereignis gestartet, so werden so viele Stimulus- bzw. Response Pattern abgearbeitet, wie über die Konfigurationsfunktionen `rspio4_ConfigureStimTiming` und `rspio4_ConfigureRespTiming` in den Parametern `Loop Count` eingestellt wurden.

Ausgänge der Trigger Einheit:

Der Ausgang der Trigger Einheit dient dem Ansteuern anderer Funktionen. Die Auswahl eines bestimmten Typs von Ausgangssignal beeinflusst nicht die interne Funktion (Stimulus ausgeben, Response einlesen).

Dieses Ausgangssignal (z.B. ACTIVE) kann auf die PXI0 .. 7 Leitungen sowie auf XTO ausgegeben werden, um z.B. eine Messung auf einer anderen CompactPCI Karte auszulösen.

5.4.2 Empfang von Triggersignalen

Zum Starten der Steuerungen stehen verschiedene Triggerquellen zur Verfügung:

Bezeichnung	Anmerkung
SW - Trigger	Die Ablaufsteuerung startet sofort nach der Armierung (beim Aufruf der Funktion `rspio4_InitiateSWTrigger`). Das ist der Standardablauf, wenn ein Pattern Set unter Programm-Kontrolle abgearbeitet werden soll. Einstellung siehe auch im Beispiel für dynamischen Test mit Low Level Funktionen.
„XTI“	TTL Eingang am frontseitigen Stecker X10; eine positive oder negative Flanke startet den Ablauf. Konfiguriert wird eine Triggerung auf XTI über die Funktion `rspio4_ConfigHWTriggerInput`. Damit wird die Flanke festgelegt und Triggerung über XTI ein- bzw. ausgeschaltet.
PXI0 to PXI7	Positive oder negative Flanken auf diesen Leitungen starten den Ablauf. Die Einstellung der aktiven Flanke und welches Triggersignal verwendet wird, erfolgt wiederum mit `rspio4_ConfigHWTriggerInput`. Werden mehrere Triggerquellen angewählt, so ist das äuqivalent einer ODER-Verknüpfung.
Pattern Comparator	Ein Pattern Comparator, der die Zustände der 32 Eingänge (IN) auswertet kann ebenfalls als Triggerquelle verwendet werden. Konfiguriert wird dies auch wieder über `rspio4_ConfigHWTriggerInput` sowie `rspio4_ConfigDinComparator` um die Bedingung für den Input Pattern Comparator zu definieren.

Die Funktion `rspio4_ConfigHWTriggerInput` legt die Triggerquelle fest. Danach werden mit der Funktion `rspio4_EnableHWTrigger` die Hardware-Trigger-Sourcen armiert und die adressierten Ablaufsteuerungen befinden sich im Zustand „Waiting“. Falls ein Software-Trigger ausgelöst wurde, geht die zugehörige Steuerung sofort in den Zustand „Running“ über. Ansonsten findet der Zustandswechsel erst nach dem Eintreffen des Triggerereignisses statt. In diesem Zustand wird dann die Anzahl Pattern im Stimulusspeicher (Stim Loop Count) ausgegeben und die Anzahl Pattern (Resp Loop Count) im Referenzspeicher aufgezeichnet.

Falls nur Pattern ausgegeben werden sollen, muss die Funktion `rspio4_InitiateSWTrigger` nur mit der Maske RSPIO4_IT_MASK_IT1 aufgerufen werden. Wenn nur aufgezeichnet werden soll, muss die Funktion allein mit RSPIO4_IT_MASK_IT2 aufgerufen werden.

Nachdem die vorhandene Anzahl Pattern abgearbeitet wurde, geht die zugehörige Ablaufsteuerung in den Zustand „Stopped“ zurück.

Der aktuelle Zustand kann mit der Funktion `rspio4_GetItStatus` für beide Ablaufsteuerungen abgefragt werden. Mit Hilfe der Funktion `rspio4_WaitUntilPatternSetComplete` kann im Testprogramm auf das Ende der Ausführung gewartet werden.

Wenn sich die Ablaufsteuerung im Zustand „Waiting“ bzw. „Running“ befindet, können einige Treiberfunktionen nicht ausgeführt werden. Diese Funktionen liefern in diesem Fall eine Fehlermeldung. Bei Bedarf kann die Ablaufsteuerung mit der Funktion `rspio4_AbortExecution` in den Grundzustand gebracht werden.

5.4.3 Generierung von Trigger-Signalen

Das Modul R&S TS-PIO4 ist in der Lage, Triggersignale auf folgenden Leitungen zu generieren:

Bezeichnung	Anmerkung
XTO	TTL Ausgang am frontseitigen Stecker
PXI0 ... PXI7	PXI Triggerleitungen auf der Backplane

Damit eine Änderung auf den Triggerleitungen stattfindet, muss der ausgewählten Leitung ein Ereignis zugeordnet werden, das den Triggerpuls auslöst.

Mit Hilfe der Funktion `rspio4_ConfigXTO` können folgende Signale auf den Ausgang geroutet werden:

- Ausgang Pattern Comparator
- Ausgang IT1
- Ausgang IT2

Mit Hilfe der Funktion `rspio4_ConfigPxiTrigOut` können folgende Signale auf eine ausgewählte PXI Triggerleitung geschaltet werden:

- Ausgang Pattern Comparator
- Ausgang IT1
- Ausgang IT2
- Ausgang General Purpose Trigger Generator

Zusätzlich kann mit dieser Funktion das ausgewählte Signal invertiert werden und der Ausgangstreiber der PXI Triggerleitung aktiviert werden.

Die Funktion `rspio4_InitiateSWTrigger` dient zum Starten der folgenden Funktionsblöcken unter Softwarekontrolle:

- Trigger Logik Block IT1 (Ablaufsteuerung für Datenausgabe)
- Trigger Logik Block IT2 (Ablaufsteuerung für Datenerfassung)
- Sample Trigger Stimulus (Ausgabe eines einzelnen Pattern aus den Stimulusspeicher)
- Sample Trigger Response (Aufzeichnung eines einzelnen Pattern in den Responsespeicher)
- General Purpose Trigger Generator (Erzeugung eines einzelnen Triggerpulses)

Mit Hilfe der Funktion `rspio4_ConfigItTrigOut` kann festgelegt werden, welches Signal die Triggerlogikblöcke IT1 und IT2 ausgeben sollen, nachdem sie entweder per Software (`rspio4_InitiateSWTrigger`) oder über ein konfiguriertes Hardwaresignal (`rspio4_ConfigHWTriggerInput`) gestartet wurden. Die folgende Tabelle zeigt die Optionen:

Bezeichnung	Anmerkung
ITn ACTIVE	Puls während das Pattern Set ausgegeben wird.
ITn OUT	Trigger Block gibt für jedes Pattern einen Impuls aus. Die Pulse sind relativ kurz (minimale Pattern Rate/2).
ITn STATUS	Signalisiert den internen Status des Triggerblocks. Das Signal wird beim Empfangen des Triggereignisses aktiv. Erst nach dem Rücksetzen des Triggerblocks wird das Signal wieder inaktiv.
ITn START	Ein Puls nachdem die Triggerbedingung erkannt worden ist.

Siehe auch Beispiel Kapitel 7.3.6, "Getriggerte Patternausführung", auf Seite 73.

5.5 PWM

Das Modul unterstützt die Ausgabe von PWM Signalen an beliebigen Ausgängen (OUTx).

Die PWM wird über die Funktion `rspio4_ConfigurePWM` konfiguriert. Die Einstellung der Frequenz erfolgt in Hz und das Tastverhältnis wird in % angegeben.

Um die PWM für einzelne Kanäle ein-/bzw. auszuschalten wird die Funktion `rspio4_SetStatePWM` verwendet.

Die Zeitauflösung beträgt 12.5 ns wenn der interne Referenztakt von 80 MHz verwendet wird. Über einen Teiler werden niedrigere Frequenzen erzeugt. Durch dieses Verfahren gibt es Einschränkungen bei der Wahl von Frequenz und Tastverhältnis.

Die folgenden Beispiele zeigen die Zusammenhänge bei den höchsten Frequenzen:

- 40 MHz: 0 %, 50 %, 100 %
- 26.7 MHz: 0 %, 33 %, 66 %, 100 %
- 20 MHz: 0 %, 25 %, 50 %, 75 %, 100 %

5.6 Frequenzmessung

Das R&S TS-PIO4 Modul unterstützt die Frequenzmessung auf den Eingängen IN1 … IN32. Verwendet wird dazu die Funktion `rspio4_FrequencyMeasurement`. Abhängig von den übergebenen Parameter werden zwei Methoden unterschieden:

- Angabe einer Gate Time: Es werden solange Pulse gezählt bis die Gate Time abgelaufen ist. Dadurch lässt sich der Frequenzwert berechnen. Der Parameter „gateCount“ wird auf 0 gesetzt.
- Bei Angabe einer Gate Time gleich 0 und einer definierten Anzahl an Pulsen (gateCounts) misst das Modul die Zeit, die benötigt wird, bis eine bestimmte Anzahl von Impulse angekommen ist. Der Wert wird auch in eine Frequenz umgerechnet.

5.7 Bidirektionale Kanäle

Jeder Eingang kann individuell mit dem zugehörigen Ausgang über einen Analogschalter verbunden werden. Dadurch ist ein Rücklesen der Ausgänge bzw. das Einlesen einer Prüflingsreaktion bei abgeschalteten Ausgängen (Tri-State) möglich. Die Verschaltung erfolgt mit Hilfe der Funktion `rspio4_ConnectInOut`.

5.8 Externer Takteingang

Über den externen Takteingang (EXT_CLK) können verschiedene Anwendungen realisiert werden. Eine Möglichkeit ist die Erzeugung von Ausgangsfrequenzen, die mittels internem Frequenzteiler nicht erreichbar sind, alternativ kann die Messkarte auf den Takt eines Prüflings synchronisiert werden. Die minimal zulässige Eingangsfrequenz beträgt 20 MHz, die maximale 40 MHz. Alle kleineren Frequenzen lassen sich über den internen Teiler erzeugen.

Die Taktsignalquelle (interner PXI 10 MHz Takt bzw. eingespeist über EXT_CLK) kann mit der Funktion `rspio4_ConfigureClock` ausgewählt werden. Wenn ein externer Takt gewählt wird, muss mit dem Parameter `ExternClockFreq` die Taktfrequenz

angegeben werden. Bei Auswahl des internen Takt wird dieser Parameter ignoriert. Mit der Funktion `rspio4_GetPllLockedStatus` kann der Zustand der Takt-PLL abgefragt werden. Sie signalisiert ob der Takt erfolgreich verwendet werden kann.

Durch den geänderten Takt wird die Zeitbasis der Karte verändert, dadurch vergrößert sich z.B. die Schrittgröße bei den einstellbaren Triggerverzögerungen.

5.9 Synchronisation mehrerer Module

Mehrere Module können über die verschiedenen Triggerein- bzw. Ausgänge synchronisiert Daten ausgeben bzw. aufnehmen. Die internen PXI-Triggerleitungen sind nicht längenkompensiert, so dass es zu relevanten Laufzeitunterschieden kommen kann. Die höchste Genauigkeit kann durch längenkompensierte Leitungen zwischen XTO des Masters und den XTIs erreicht werden.

5.10 AUX-Kanäle

Vom rückwärtigen Steckverbinder X20 werden die Signale AUX1, AUX2, AUX3 und AUX4 je einmal direkt zum frontseitigen Steckverbinder X10 geführt. Die Stromtragfähigkeit beträgt je 1A.

5.11 GND Relais

Das GND-Relais verbindet die Signale GNDNO mit GND am frontseitigen Stecker X10. Dadurch kann ein Prüfling wahlweise mit GND verbunden werden. Bei In-Circuit-Tests muss der Prüfling erdfrei vorliegen während bei Funktionstests häufig eine Verbindung zwischen Prüflingsmasse und Systemmasse vorteilhaft ist.

Im Grundzustand ist diese Verbindung offen.

6 Inbetriebnahme

6.1 Installation des Moduls R&S TS-PIO4

Zur Installation des Einsteckmoduls R&S TS-PIO4 ist wie folgt vorzugehen:

ACHTUNG

Beschädigung der Backplane durch verbogene Pins

Durch verbogene Pins kann die Backplane dauerhaft beschädigt werden.

Die Backplane-Steckverbinder sind auf verbogene Pins zu überprüfen.

Verbogene Pins müssen ausgerichtet werden.

Beim Einschieben des Einsteckmoduls ist dieses mit beiden Händen zu führen und vorsichtig in die Backplane-Steckverbinder einzudrücken.

1. Herunterfahren und Ausschalten des R&S CompactTSVP.
2. Auswahl eines geeigneten frontseitigen Steckplatzes.
3. Entfernen der entsprechenden Teilfrontplatte am TSVP-Chassis durch Lösen der Schrauben.
4. Das Einsteckmodul mit mäßigem Druck einschieben
5. Der obere Fangstift des Einsteckmoduls muss in die rechte Bohrung, der untere in die linke Bohrung am TSVP-Chassis geführt werden.

   Das Einsteckmodul ist richtig eingeschoben, wenn ein deutlicher Anschlag zu spüren ist.
6. Die Schrauben oben und unten an der Frontplatte des Einsteckmoduls festschrauben.

7 Software

7.1 Treibersoftware

Für die Ansteuerung des digitalen Funktionstestmoduls R&S TS-PIO4 steht ein Lab-Windows IVI Treiber zur Verfügung. Alle Zusatzfunktionen der Hardware werden über spezifische Erweiterungen des Treibers bedient. Der Treiber ist Bestandteil der ROHDE & SCHWARZ GTSL-Software. Alle Funktionen des Treibers sind in der Online-Hilfe und in den LabWindows/CVI Function-Panels ausführlich dokumentiert. Bei der Treiberinstallation werden die folgenden Softwaremodule installiert:

Modul	Pfad	Anmerkung
`rspio4.dll`	`<GTSL Verzeichnis>\Bin`	Treiber
`rspio4.chm`	`<GTSL Verzeichnis>\Bin`	Hilfedatei
`rspio4.fp`	`<GTSL Verzeichnis>\Bin`	LabWindows/CVI-Function-Panel-File, Function-Panels für CVI-Entwicklungsumgebung
`rspio4.sub`	`<GTSL Verzeichnis>\Bin`	LabWindows/CVI-Attribute-Datei. Diese Datei wird von einigen „Function Panels“ benötigt.
`rspio4.lib`	`<GTSL Verzeichnis>\Bin`	Import-Bibliothek
`rspio4.h`	`<GTSL Verzeichnis>\Include`	Header-Datei zum Treiber

Zum Betrieb des Treibers sind die IVI- und VISA-Bibliotheken der Firma National Instruments notwendig.

7.2 Softpanel

Dem Software-Paket des Moduls R&S TS-PIO4 ist ein Softpanel beigefügt (siehe Bild 7-1). Das Softpanel setzt auf dem IVI Treiber auf und ermöglicht die interaktive Bedienung des Moduls.

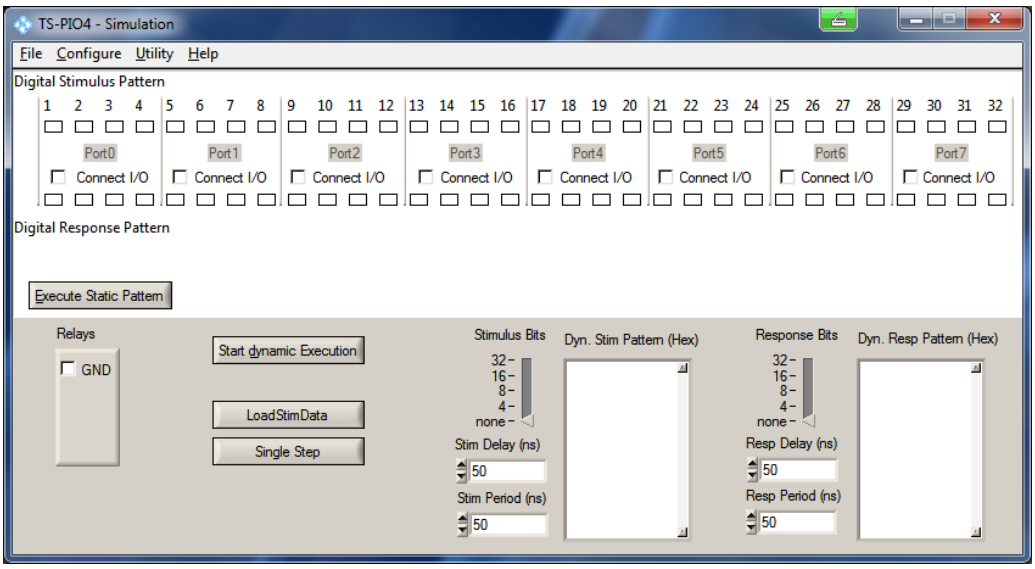


Bild 7-1: Softpanel R&S TS-PIO4

Die Bedienung der Softpanels ist in der *Software Description R&S GTSL* beschrieben.

Im R&S TS-PIO4 Softpanel lassen sich statische und dynamische Bitmuster an den Ausgängen einstellen sowie Eingangszustände einlesen.

In weiteren Dialogen, die aus dem Configure-Menü gestartet werden, können die Spannungsbereiche, die High- und Low-Treiberpegel sowie die Schwellspannungen der Eingangskomparatoren eingestellt werden.

Außerdem ist es möglich die Konfiguration der Stimulus- und Response-Triggereinheiten durchzuführen sowie Trigger-Ereignisse auszulösen. und die Einstellungen für die PWM vorzunehmen.

7.3 Programmierbeispiele R&S TS-PIO4

7.3.1 Digitaltest mit der Bibliothek DIO Manager

Dieses Beispiel zeigt die Ausführung eines Digitaltest mit Hilfe der GTSL Bibliothek DIO Manager (DIOMGR). Die Eingabedatei (`823916.tbl`) und die zugehörige Konfigurationsdatei (`pio4Application.ini`) sind beschrieben in Kapitel 5.2.2, "Digitaltest mit DIO Manager", auf Seite 24. Zusätzliche Informationen zum Aufbau der Konfigurationsdateien (Physical Layer Configuration File und Application Layer Configuration File) und zur Funktion des Resource Managers (RESMGR) sind im Handbuch *Software Description R&S GTSL* zu finden.

7.3.1.1 Hauptfunktion

```
/* Programming example with DIOMGR */
#include <ansi_c.h>
#include "resmgr.h"
#include "diomgr.h"

static short errorOccurred;
static long  errorCode;
static char  errorMessage[GTSL_ERROR_BUFFER_SIZE];
static long  residDiomgr;

static char benchName[]      = "bench->823916";
static char fileName[]       = "823916.tbl";
static char fileNameResult[] = "823916result.tbl";

/* list of stimulus channels */
static char stimChannels[] = "N14CR0805170,E_COM,E_EC1,E_EC2,E_EC3";
static char respChannels[] = "N13HCPN31500,N13HCPM31500,"
                             "N13HCPO31500,N13HCPL31500";

/* prototypes */
static void cs ( char * funcName );
static void runTest ( void );
static void diagnosis ( void );

/* FUNCTION ******************************************************************/
/* loads the libraries and runs the test
*******************************************************************************/
int main (int argc, char *argv[])
{
  printf("Example using DIOMGR functions for dynamic pattern execution\n\n");

  /* setup libraries */
  RESMGR_Setup (0, "physical.ini", "pio4Application.ini",
    &errorOccurred, &errorCode, errorMessage);
  cs("RESMGR_Setup");

  if ( ! errorOccurred )
  {
    DIOMGR_Setup (0, benchName, &residDiomgr,
      &errorOccurred, &errorCode, errorMessage);
    cs("DIOMGR_Setup");
  }

  if ( ! errorOccurred )
  {
    runTest ( );
  }
```

```
  /* cleanup libraries */
  DIOMGR_Cleanup (0, residDiomgr, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_Cleanup");

  RESMGR_Cleanup ( 0, &errorOccurred, &errorCode, errorMessage);
  cs("RESMGR_Cleanup");

  printf("\nPress 'Enter' to terminate\n");
  getchar();

  return 0;
}
```

7.3.1.2 Fehlerbehandlung

Diese Funktion überprüft die Rückgabewerte einer Funktion der GTSL Bibliotheken (RESMGR, DIOMGR). Im Fehlerfall wird eine Nachricht ausgegeben.

```
/* FUNCTION ******************************************************************/
/* checks the return status of a library call
*******************************************************************************/
static void cs ( char * funcName )
{
  if ( errorOccurred )
  {
    printf ("%s returned 0x%08X\n%s\n\n", funcName, errorCode, errorMessage);
  }
}
```

7.3.1.3 Ausführung des Digitaltests

```
/* FUNCTION ******************************************************************/
/* Configures the module for digital test, loads the test and executes it.
   The results are uploaded from the module and stored as a result TBL file.
*******************************************************************************/
static void runTest ( void )
{
  long numPatterns;
  long patternSetResult;
  char usedChannels[512];

  /* configure voltage ranges */
  strcpy(usedChannels, stimChannels);
  strcat(usedChannels, ",");
  strcat(usedChannels, respChannels);

  DIOMGR_ConfigureVoltageRange (0, residDiomgr, usedChannels, "VOLTAGE_RANGE_2",
    &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigureVoltageRange");
```

```
  /* configure stimulus and response levels */
  DIOMGR_ConfigureStimulus (0, residDiomgr, stimChannels, "TTL", 3.3, 0.1,
    &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigureStimulus");

  DIOMGR_ConfigureResponse (0, residDiomgr, respChannels, "HYSTERESIS",
    0.8, 2.0, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigureResponse");

  /* load the waveform. Pattern set name = file name */
  DIOMGR_LoadWaveform (0, residDiomgr, fileName, "TBL", 1.0e-6, fileName,
    &numPatterns, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_LoadWaveform");

  /* configure the timing */
  DIOMGR_ConfigurePatternSetTiming (0, residDiomgr, fileName, 1.0e-6,
    0.5e-6, 1.0, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigurePatternSetTiming");

  /* run the test */
  DIOMGR_ExecutePatternSet (0, residDiomgr, fileName, "SYNCHRONOUS",
    &patternSetResult, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ExecutePatternSet");

  if ( DIOMGR_VAL_RESULT_FAILED == patternSetResult )
  {
    printf ( "Pattern set failed\n" );
    diagnosis ();
  }
  else
  {
    printf ( "Pattern set passed\n" );
  }

  DIOMGR_SaveWaveform (0, residDiomgr, fileName, fileNameResult, "TBL",
    &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_SaveWaveform");

  DIOMGR_UnloadWaveform (0, residDiomgr, fileName,
    &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_UnloadWaveform");
}
```

7.3.1.4 Auswertung der fehlerhaften Pattern

```
/* FUNCTION ****************************************************************/
/* reads information about pattern set failures and prints it to stdout
```

```
*****************************************************************************/
static void diagnosis ( void )
{
  long executedPatternCount;
  long failedPatternCount;
  long failedChannelCount;
  char failedChannelNames[1024];
  long bufferSize;
  char * pData = NULL;
  char * pResults = NULL;
  char * token = NULL;
  int i;
  int numFail;

  /* read results about pattern set failure */
  DIOMGR_GetPatternSetExecutedPatternCount (0, residDiomgr, fileName,
    &executedPatternCount, &errorOccurred, &errorCode, errorMessage );
  cs("DIOMGR_GetPatternSetExecutedPatternCount");
  printf ( "%d patterns executed\n", executedPatternCount );

  DIOMGR_GetPatternSetFailedPatternCount (0, residDiomgr, fileName,
    &failedPatternCount, &errorOccurred, &errorCode, errorMessage );
  cs("DIOMGR_GetPatternSetFailedPatternCount");
  printf ( "%d patterns failed\n", failedPatternCount );

  DIOMGR_GetPatternSetFailedChannelCount (0, residDiomgr, fileName,
    &failedChannelCount, &errorOccurred, &errorCode, errorMessage );
  cs("DIOMGR_GetPatternSetFailedChannelCount");
  printf ( "%d channels failed:\n", failedChannelCount );

  DIOMGR_GetPatternSetFailedChannelNames (0, residDiomgr, fileName,
    sizeof(failedChannelNames), failedChannelNames,
    &errorOccurred, &errorCode, errorMessage );
  cs("DIOMGR_GetPatternSetFailedChannelNames");
  printf ( "   %s\n", failedChannelNames );

  /* allocate memory for data and pass/fail */
  bufferSize = executedPatternCount + 1;
  pData = malloc ( bufferSize );
  pResults = malloc ( bufferSize );

  /* cycle thru the channel names, output data and mark errors */
  token = strtok (failedChannelNames, ",");
  while ( token )
  {
    /* "token" contains a failed channel name */
    DIOMGR_GetPatternSetChannelData ( 0, residDiomgr, fileName, token,
      bufferSize, pData, &errorOccurred, &errorCode, errorMessage );
    cs("DIOMGR_GetPatternSetChannelData");
```

```
    DIOMGR_GetPatternSetChannelResults ( 0, residDiomgr, fileName, token,
      bufferSize, pResults, &errorOccurred, &errorCode, errorMessage );
    cs("DIOMGR_GetPatternSetChannelResults");

    /* 1=passed, 0=failed. Replace "failed" by an X and "passed" by a space */
    numFail = 0;
    for ( i=0; i<bufferSize; i++ )
    {
      switch ( pResults[i] )
      {
        case '0':
          pResults[i] = 'X';
          numFail++;
          break;
        case '1':
          pResults[i] = ' ';
          break;
        default:
          break;
      }
    }

    printf ( "\n%s : %d fails\n", token, numFail );
    printf ( "- Data    : %s\n", pData );
    printf ( "- Results : %s\n", pResults );

    /* get next channel name */
    token = strtok ( NULL, "," );
  }


  free ( pData );
  free ( pResults );
}
```

7.3.2 Dynamische Patternausführung mit IVI Digital

7.3.2.1 Hauptfunktion

Der Rückgabewert eines Gerätetreiberaufrufs wird in der modulglobalen Variable `sta` abgelegt. Mit Hilfe der Funktion `chk()` wird der Statuscode überprüft.

```
/* Example using IVI functions for dynamic pattern execution */
#include <ansi_c.h>
#include "rspio4.h"

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::11::INSTR";
```

```
static char patternSetName[] = "823916";

/* channel names */
static char o1[] = "OUT1";
static char o2[] = "OUT2";
static char o3[] = "OUT3";
static char o4[] = "OUT4";
static char o5[] = "OUT5";

static char bus[] = "OUT20-OUT27";
static char out[] = "OUT1-OUT31";

static char i1[] = "IN1";
static char i2[] = "IN2";
static char i3[] = "IN3";
static char i4[] = "IN4";

static char in[] = "IN1-IN32";

static ViStatus sta;
static ViSession vi;

/* prototypes */
static void chk ( char * funcName );
static void runTest ( void );
static void createPatternSet ( void );
static void diagnosis ( void );

/* FUNCTION ******************************************************************/
/* loads the driver and runs the test
*******************************************************************************/
int main (int argc, char *argv[])
{
  printf("Example using IVI functions for dynamic pattern execution\n\n");

  /* open driver */
  sta = rspio4_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", &vi);
  /* check return value */
  chk ("rspio4_InitWithOptions");

  if ( VI_SUCCESS == sta )
  {
    runTest();

    /* close driver */
    sta = rspio4_close(vi);
    chk ("rspio4_close");
  }

  printf("\nPress 'Enter' to terminate\n");
```

```
  getchar();

  return 0;
}
```

7.3.2.2 Fehlerbehandlung

Diese Funktion überprüft die Rückgabewerte eines Treiberaufrufs. Im Fehlerfall wird eine Fehlermeldung ausgegeben.

```
/* FUNCTION ******************************************************************/
/* checks the return status of a driver call
*******************************************************************************/
static void chk ( char * funcName )
{
  if ( sta != VI_SUCCESS )
  {
    char errorMessage[256];

    rspio4_error_message(vi, sta, errorMessage);
    printf ("%s returned 0x%08X; %s\n", funcName, sta, errorMessage);
  }
}
```

7.3.2.3 Ausführung des Digitaltests

```
/* FUNCTION ******************************************************************/
/* configures the digital test, loads the test and executes it.
*******************************************************************************/
static void runTest ( void )
{
  ViInt32 patternSetResult;

  /* configure stimulus and response levels */
  sta = rspio4_ConfigurePortVoltageRange(vi,
    RSPIO4_MASK_PORT_ALL, RSPIO4_PORT_VOLTAGE_RANGE_2);
  chk ("rspio4_ConfigurePortVoltageRange");

  sta = rspio4_ConfigureStimPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_STIM_MODE_TTL, 3.3, 0.0, 0.1);
  chk ("rspio4_ConfigureStimPort");

  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, 2.0, 0.8);
  chk ("rspio4_ConfigureRespPort");

  /* configure module for dynamic test and result collection */
  sta = rspio4_ConfigureMode(vi, RSPIO4_VAL_EXECUTE_DYNAMIC,
    RSPIO4_VAL_COLLECT_ALL);
```

```
  chk ("rspio4_ConfigureMode");

  /* create and load the pattern set */
  createPatternSet();

  /* configure the timing */
  sta = rspio4_ConfigurePatternSetTiming(vi, patternSetName, 1.0e-6, 0.5e-6);
  chk ("rspio4_ConfigurePatternSetTiming");

  /* run the test */
  sta = rspio4_ExecutePatternSet(vi, patternSetName, 1000);
  chk ("rspio4_ExecutePatternSet");

  /* fetch pass/fail result */
  sta = rspio4_FetchPatternSetResult(vi, patternSetName, &patternSetResult);
  chk ("rspio4_FetchPatternSetResult");

  if ( RSPIO4_VAL_RESULT_FAIL == patternSetResult )
  {
    printf("Pattern set failed\n");
    diagnosis ();
  }
  else
  {
    printf("Pattern set passed\n");
  }

  /* clear pattern set */
  sta = rspio4_ClearPatternSet(vi, patternSetName);
  chk ("rspio4_ClearPatternSet");
}
```

7.3.2.4 Erzeugen des Pattern-Sets

```
/* FUNCTION ***************************************************************/
/* creates and loads a pattern set
   channel assignment:

    E_COM          = out1
    E_EC1          = out2
    E_EC2          = out3
    E_EC3          = out4
    N14CR0805170   = out5
    bus            = out20 - out27

    N13HCPL31500   = in1
    N13HCPM31500   = in2
    N13HCPN31500   = in3
    N13HCPO31500   = in4
```

```
*****************************************************************************/
static void createPatternSet ( void )
{
  ViInt32 ph;

  /* create a pattern set */
  sta = rspio4_CreatePatternSet(vi, patternSetName);
  chk ("rspio4_CreatePatternSet");

  /* create a pattern */
  sta = rspio4_CreatePattern(vi, &ph);
  chk ("rspio4_CreatePattern");

  /* start loading */
  sta = rspio4_BeginPatternSetLoading(vi, patternSetName);
  chk ("rspio4_BeginPatternSetLoading");

  /*** 1. pattern : stim all zero, resp don't care */
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x0);
  chk ("rspio4_ConfigureGroupOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, i1, RSPIO4_VAL_OPCODE_IOX);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, i2, RSPIO4_VAL_OPCODE_IOX);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, i3, RSPIO4_VAL_OPCODE_IOX);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, i4, RSPIO4_VAL_OPCODE_IOX);
  chk ("rspio4_ConfigureChannelOpcode");
  /* load pattern */
  sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
  chk ("rspio4_LoadDynamicPattern");
  // NOTE : previously assigned channels keep their channel opcode in
  //        the following pattern. Therefore only the changes have to be
  //        programmed

  /*** 2. pattern: N14CR0805170 = 1, bus = 01 */
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IH);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x01);
  chk ("rspio4_ConfigureGroupOpcode");
```

```
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/*** 3. pattern: all stim tristate, bus = 02 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x02);
chk ("rspio4_ConfigureGroupOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/*** 4. pattern: all stim = 0, bus = 03 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x03);
chk ("rspio4_ConfigureGroupOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/*** 5. pattern: N14CR0805170 = 1, bus = 04 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IH);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x04);
chk ("rspio4_ConfigureGroupOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/*** 6. pattern: all stim tristate, bus = 05 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
```

```
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x05);
chk ("rspio4_ConfigureGroupOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");
/*** 7. pattern: all stim = 0, bus = 06, resp N13HCPN31500 = 1 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x06);
chk ("rspio4_ConfigureGroupOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, i3, RSPIO4_VAL_OPCODE_OH);
chk ("rspio4_ConfigureChannelOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/* ... etc ... */

/* last pattern : all tristate / don't care */
sta = rspio4_ConfigureGroupOpcode(vi, ph, out, RSPIO4_VAL_GROUP_IGNORE, 0x0);
chk ("rspio4_ConfigureGroupOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, in, RSPIO4_VAL_GROUP_IGNORE, 0x0);
chk ("rspio4_ConfigureGroupOpcode");
/* load pattern */
sta = rspio4_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio4_LoadDynamicPattern");

/* loading finished */
sta = rspio4_EndPatternSetLoading(vi, patternSetName);
chk ("rspio4_EndPatternSetLoading");

/* pattern is no longer used */
sta = rspio4_ClearPattern(vi, ph);
chk ("rspio4_ClearPattern");
```

```
}
```

7.3.2.5 Auswertung der fehlerhaften Pattern

```
/* FUNCTION ***************************************************************/
/* reads information about pattern set failures and prints it to stdout
*****************************************************************************/
static void diagnosis ( void )
{
  ViInt32 executedPatternCount;
  ViInt32 numChannels = 4;  /* in1 - in4 */
  ViInt32 * pResults;
  ViInt32 * pData;
  ViInt32 * pPatterns;
  ViInt32 actualSize;
  int pattern;
  int channel;
  int idx;
  int failedPatterns;

  sta = rspio4_GetPatternSetExecutedPatternCount (vi, patternSetName,
    &executedPatternCount);
  chk ("rspio4_GetPatternSetExecutedPatternCount");

  printf("%d patterns executed\n", executedPatternCount);

  pResults  = calloc(numChannels * executedPatternCount, sizeof(ViInt32));
  pData     = calloc(numChannels * executedPatternCount, sizeof(ViInt32));
  pPatterns = calloc(executedPatternCount, sizeof(ViInt32));

  /* upload pattern results */
  sta = rspio4_FetchDynamicPatternListResults (vi, patternSetName,
    0, executedPatternCount, executedPatternCount, pPatterns, &actualSize);
  chk ("rspio4_FetchDynamicPatternListResults");

  /* calculate number of failed patterns */
  failedPatterns = 0;
  for ( pattern = 0; pattern < executedPatternCount; pattern++ )
  {
    if ( RSPIO4_VAL_RESULT_FAIL == pPatterns[pattern] )
    {
      failedPatterns++;
    }
  }

  printf("%d patterns failed\n", failedPatterns);

  /* upload data and results for in1 - in4 */
```

```
sta = rspio4_FetchDynamicChannelListPatternData(vi, patternSetName,
  0, executedPatternCount, "in1,in2,in3,in4",
  numChannels * executedPatternCount, pData, &actualSize);
chk ("rspio4_FetchDynamicChannelListPatternData");

sta = rspio4_FetchDynamicChannelListPatternResults (vi, patternSetName,
  0, executedPatternCount, "in1,in2,in3,in4",
  numChannels * executedPatternCount, pResults, &actualSize);
chk ("rspio4_FetchDynamicChannelListPatternResults");
for ( channel = 0; channel < numChannels; channel ++ )
{
  printf("\nin%d :\n", channel+1);

  /* data */
  printf("- Data    : ");
  for ( pattern = 0; pattern < executedPatternCount; pattern++ )
  {
    idx = pattern * numChannels + channel;
    switch ( pData[idx] )
    {
      case RSPIO4_VAL_DATA_HIGH:
        printf("1");
        break;
      case RSPIO4_VAL_DATA_LOW:
        printf("0");
        break;
      default:
        printf("?");
        break;
    }
  }
  printf("\n");

  /* results */
  printf("- Results : ");
  for ( pattern = 0; pattern < executedPatternCount; pattern++ )
  {
    idx = pattern * numChannels + channel;
    switch ( pResults[idx] )
    {
      case RSPIO4_VAL_RESULT_PASS:
        printf(" ");
        break;
      case RSPIO4_VAL_RESULT_FAIL:
        printf("X");
        break;
      default:
        printf("?");
        break;
    }
```

```
    }
    printf("\n");
  }

  free(pResults);
  free(pData);
  free(pPatterns);
}
```

7.3.3 Statische Patternausführung mit IVI Digital

7.3.3.1 Hauptfunktion

```
/* Example using IVI functions for static pattern execution */
#include <ansi_c.h>
#include "rspio4.h"

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::11::INSTR";

/* channel names */
static char o1[] = "OUT1";
static char o2[] = "OUT2";
static char o3[] = "OUT3";
static char o4[] = "OUT4";
static char o5[] = "OUT5";

static char bus[] = "OUT20-OUT27";
static char out[] = "OUT1-OUT31";
static char i1[] = "IN1";
static char i2[] = "IN2";
static char i3[] = "IN3";
static char i4[] = "IN4";

static char in[] = "IN1-IN32";

static ViStatus sta;
static ViSession vi;

/* prototypes */
static void chk ( char * funcName );
static void runTest ( void );
static void executePatternSet ( void );
static void executePattern ( ViInt32 patternHandle, ViInt32 patternIdx );
static void diagnosis ( void );

/* FUNCTION ****************************************************************/
```

```
/* loads the driver and runs the test
*****************************************************************************/
int main (int argc, char *argv[])
{
  printf("Example using IVI functions for static pattern execution\n\n");

  /* open driver */
  sta = rspio4_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", &vi);
  /* check return value */
  chk ("rspio4_InitWithOptions");

  if ( VI_SUCCESS == sta )
  {
    runTest();

    /* close driver */
    sta = rspio4_close(vi);
    chk ("rspio4_close");
  }

  printf("\nPress 'Enter' to terminate\n");
  getchar();

  return 0;
}
```

7.3.3.2 Fehlerbehandlung

Diese Funktion überprüft die Rückgabewerte eines Treiberaufrufs. Im Fehlerfall wird eine Fehlermeldung ausgegeben.

```
/* FUNCTION ******************************************************************/
/* checks the return status of a driver call
*****************************************************************************/
static void chk ( char * funcName )
{
  if ( sta != VI_SUCCESS )
  {
    char errorMessage[256];

    rspio4_error_message(vi, sta, errorMessage);
    printf ("%s returned 0x%08X; %s\n", funcName, sta, errorMessage);
  }
}
```

7.3.3.3 Ausführung des Digitaltests

```
/* FUNCTION ******************************************************************/
/* configures the digital test, loads the test and executes it.
```

```
*****************************************************************************/
static void runTest ( void )
{
  /* configure stimulus and response levels */
  sta = rspio4_ConfigurePortVoltageRange(vi,
    RSPIO4_MASK_PORT_ALL, RSPIO4_PORT_VOLTAGE_RANGE_2);
  chk ("rspio4_ConfigurePortVoltageRange");

  sta = rspio4_ConfigureStimPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_STIM_MODE_TTL, 3.3, 0.0, 0.1);
  chk ("rspio4_ConfigureStimPort");

  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, 2.0, 0.8);
  chk ("rspio4_ConfigureRespPort");

  /* configure module for static test and result collection */
  sta = rspio4_ConfigureMode(vi, RSPIO4_VAL_EXECUTE_STATIC,
    RSPIO4_VAL_COLLECT_ALL);
  chk ("rspio4_ConfigureMode");

  /* configure the timing */
  sta = rspio4_ConfigureStaticResponseDelay (vi, 0.5e-6);
  chk ("rspio4_ConfigureStaticResponseDelay");

  /* execute the pattern set */
  executePatternSet();
}
```

7.3.3.4 Ausführung eines Pattern-Sets

```
/* FUNCTION ******************************************************************/
/* creates and executes a pattern set
   channel assignment:

    E_COM          = out1
    E_EC1          = out2
    E_EC2          = out3
    E_EC3          = out4
    N14CR0805170   = out5
    bus            = out20 - out27

    N13HCPL31500   = in1
    N13HCPM31500   = in2
    N13HCPN31500   = in3
    N13HCPO31500   = in4
*****************************************************************************/
static void executePatternSet ( void )
{
```

```
ViInt32 ph;
ViInt32 patternIdx = 1;

/* create a pattern */
sta = rspio4_CreatePattern(vi, &ph);
chk ("rspio4_CreatePattern");

/*** 1. pattern : stim all zero, resp don't care */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x0);
chk ("rspio4_ConfigureGroupOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, i1, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, i2, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, i3, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, i4, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");

executePattern(ph, patternIdx++);

// NOTE : previously assigned channels keep their channel opcode in
//        the following pattern. Therefore only the changes have to be
//        programmed

/*** 2. pattern: N14CR0805170 = 1, bus = 01 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IH);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x01);
chk ("rspio4_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);
/*** 3. pattern: all stim tristate, bus = 02 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IOX);
```

```
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x02);
chk ("rspio4_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 4. pattern: all stim = 0, bus = 03 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x03);
chk ("rspio4_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 5. pattern: N14CR0805170 = 1, bus = 04 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IH);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x04);
chk ("rspio4_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 6. pattern: all stim tristate, bus = 05 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IOX);
chk ("rspio4_ConfigureChannelOpcode");
sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x05);
chk ("rspio4_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 7. pattern: all stim = 0, bus = 06, resp N13HCPN31500 = 1 */
sta = rspio4_ConfigureChannelOpcode(vi, ph, o1, RSPIO4_VAL_OPCODE_IL);
```

```
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o2, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o3, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o4, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, o5, RSPIO4_VAL_OPCODE_IL);
  chk ("rspio4_ConfigureChannelOpcode");
  sta = rspio4_ConfigureGroupOpcode(vi, ph, bus, RSPIO4_VAL_GROUP_INPUT, 0x06);
  chk ("rspio4_ConfigureGroupOpcode");
  sta = rspio4_ConfigureChannelOpcode(vi, ph, i3, RSPIO4_VAL_OPCODE_OH);
  chk ("rspio4_ConfigureChannelOpcode");

  executePattern(ph, patternIdx++);

  /* ... etc ... */

  /* last pattern : all tristate / don't care */
  sta = rspio4_ConfigureGroupOpcode(vi, ph, out, RSPIO4_VAL_GROUP_IGNORE, 0x0);
  chk ("rspio4_ConfigureGroupOpcode");
  sta = rspio4_ConfigureGroupOpcode(vi, ph, in, RSPIO4_VAL_GROUP_IGNORE, 0x0);
  chk ("rspio4_ConfigureGroupOpcode");

  executePattern(ph, patternIdx++);

  /* pattern is no longer used */
  sta = rspio4_ClearPattern(vi, ph);
  chk ("rspio4_ClearPattern");
}
```

7.3.3.5 Ausführung eins einzelnen Patterns

```
/* FUNCTION ******************************************************************/
/* executes a single pattern
*******************************************************************************/
static void executePattern ( ViInt32 patternHandle, ViInt32 patternIdx )
{
  ViInt32 patternResult;

  sta = rspio4_ExecuteStaticPattern(vi, patternHandle);
  chk ("rspio4_ExecuteStaticPattern");

  sta = rspio4_FetchStaticPatternResult (vi, &patternResult);
  chk ("rspio4_FetchStaticPatternResult");

  if ( RSPIO4_VAL_RESULT_FAIL == patternResult )
  {
    printf("Pattern %d failed\n", patternIdx);
```

```
    diagnosis ();
  }
  else
  {
    printf("Pattern %d passed\n", patternIdx);
  }
}
```

7.3.3.6 Auswertung eines fehlerhaften Patterns

```
/* FUNCTION ******************************************************************/
/* reads information about pattern failures and prints it to stdout
*******************************************************************************/
static void diagnosis ( void )
{
  ViInt32 numChannels = 4;  /* in1 - in4 */
  ViInt32 results[4];
  ViInt32 data[4];
  ViInt32 actualSize;
  int channel;

  /* upload data and results for in1 - in4 */
  sta = rspio4_FetchStaticChannelListData(vi, "in1,in2,in3,in4",
    numChannels, data, &actualSize);
  chk ("rspio4_FetchStaticChannelListData");

  sta = rspio4_FetchStaticChannelListResults(vi, "in1,in2,in3,in4",
    numChannels, results, &actualSize);
  chk ("rspio4_FetchStaticChannelListResults");

  for ( channel = 0; channel < numChannels; channel ++ )
  {
    printf("  in%d ", channel+1);

    switch ( data[channel] )
    {
      case RSPIO4_VAL_DATA_HIGH:
        printf("high ");
        break;
      case RSPIO4_VAL_DATA_LOW:
        printf("low  ");
        break;
      default:
        printf("?    ");
        break;
    }

    switch ( results[channel] )
    {
```

```
      case RSPIO4_VAL_RESULT_PASS:
        printf("pass");
        break;
      case RSPIO4_VAL_RESULT_FAIL:
        printf("fail");
        break;
      default:
        printf("not available");
        break;
    }

    printf("\n");
  }
}
```

7.3.4 Statische Patternausgabe mit Low Level Treiberfunktionen

7.3.4.1 Hauptfunktion

```
/* Example using low level driver functions for static pattern execution */
#include <utility.h>
#include <ansi_c.h>

#include "rspio4.h"

#define PATTERN_COUNT       8

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::11::INSTR";

static ViUInt32 stimData[PATTERN_COUNT] = {
  0x00000000,
  0x00000010,
  0x0000001F, /* tri state */
  0x00000000,
  0x00000010,
  0x0000001F, /* tri state */
  0x00000000,
  /* etc. */
  0xFFFFFFFF
};

static ViUInt16 portEnable[PATTERN_COUNT] = {
  0xFF,
  0xFF,
  0xFC, /* port 0 and port 1 tri state (OUT1 to OUT8) */
  0xFF,
```

```
  0xFF,
  0xFC, /* port 0 and port 1 tri state (OUT1 to OUT8) */
  0xFF,
  /* etc. */
  0x00  /* port 0 to port 7 tri state (OUT1 to OUT32) */
};

static ViStatus sta;
static ViSession vi;

/* prototypes */
static void chk ( char * funcName );
static void runTest ( void );
static void executePatternSet( void );

/* FUNCTION ******************************************************************/
/* loads the driver and runs the test
*******************************************************************************/
int main(int argc, char *argv[])
{
  printf("Use of low level driver functions for static pattern execution\n\n");

  /* open a session to the device driver */
  sta = rspio4_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio4_InitWithOptions");

  if (VI_SUCCESS == sta)
  {
    runTest();

    /* close the driver */
    sta = rspio4_close(vi);
    chk ("rspio4_close");
  }
  printf("\nPress 'Enter' to terminate\n");
  getchar();

  return 0;
}
```

7.3.4.2 Fehlerbehandlung

Diese Funktion überprüft die Rückgabewerte eines Treiberaufrufs. Im Fehlerfall wird eine Nachricht ausgegeben.

```
/* FUNCTION ******************************************************************/
/* checks the return status of a driver call
*******************************************************************************/
```

```
static void chk ( char * funcName )
{
  if ( sta != VI_SUCCESS )
  {
    char errorMessage[256];

    rspio4_error_message(vi, sta, errorMessage);
    printf ("%s returned 0x%08X; %s\n", funcName, sta, errorMessage);
  }
}
```

7.3.4.3 Ausführung des Digitaltests

```
/* FUNCTION ******************************************************************/
/* configures the digital test and executes it
*******************************************************************************/
static void runTest ( void )
{
  ViReal64 voltageHigh = 3.3;
  ViReal64 voltageLow = 0.0;
  ViReal64 currentLimit = 0.1;
  ViReal64 thresholdHigh = 2.0;
  ViReal64 thresholdLow = 0.8;

  /* set voltage range;   */
  sta = rspio4_ConfigurePortVoltageRange(vi,
    RSPIO4_MASK_PORT_ALL, RSPIO4_PORT_VOLTAGE_RANGE_2);
  chk ("rspio4_ConfigurePortVoltageRange");

  /* configure driver voltage levels */
  sta = rspio4_ConfigureStimPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_STIM_MODE_TTL, voltageHigh, voltageLow, currentLimit);
  chk ("rspio4_ConfigureStimPort");

  /* configure sensor: set all ports, use hysteresis comparator mode */
  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, thresholdHigh, thresholdLow);
  chk ("rspio4_ConfigureRespPort");

  /* configure module for dynamic test and result collection */
  sta = rspio4_ConfigureMode(vi, RSPIO4_VAL_EXECUTE_STATIC,
    RSPIO4_VAL_COLLECT_ALL);
  chk ("rspio4_ConfigureMode");

  /* configure 32 bit wide dynamic operation: OUT1 .. OUT32 will be operated
     dynamically; no static channels */
  sta = rspio4_ConfigureStimMode (vi, RSPIO4_VAL_CTRL_STATIC);
  chk ("rspio4_ConfigureStimMode");
```

```
  /* configure 32 bit wide dynamic operation */
  sta = rspio4_ConfigureRespMode(vi, RSPIO4_VAL_CTRL_STATIC);
  chk ("rspio4_ConfigureRespMode");

  /* execute the pattern set */
  executePatternSet();
}
```

7.3.4.4 Ausführung eines Pattern-Sets

```
/* FUNCTION ******************************************************************/
/* executes the pattern set
*******************************************************************************/
static void executePatternSet( void )
{
  int      loopIdx, portIdx;
  ViUInt32 response;
  ViUInt32 portMask;
  ViChar   enableInfo[9];
  ViUInt32 channelMask = 0xFFFFFFFF;
  ViUInt16 enablePortMask = RSPIO4_MASK_PORT_ALL;
  ViReal64 uutResponseDelay = 0.001;

  printf("Idx | Stimulus   | Enable     | Response  \n");
  printf("----+------------+------------+-----------\n");

  for (loopIdx = 0; loopIdx < PATTERN_COUNT; loopIdx++)
  {
    sta = rspio4_SetDoutPort(vi, channelMask,
       stimData[loopIdx], enablePortMask, portEnable[loopIdx]);
    chk ("rspio4_SetDoutPort");

    Delay(uutResponseDelay);

    sta = rspio4_GetDinState(vi, &response);
    chk ("rspio4_GetDinState");

    portMask = RSPIO4_MASK_PORT0;
    for (portIdx = 7; portIdx >= 0; portIdx--)
    {
      if (portEnable[loopIdx] & portMask)
      {
        enableInfo[portIdx] = 'F';
      }
      else
      {
        enableInfo[portIdx] = '0';
      }
      portMask = portMask << 1;
```

```
    }

    printf("%3d | 0x%08X | 0x%s | 0x%08X\n", loopIdx,
      stimData[loopIdx], enableInfo, response);
  }
}
```

7.3.5 Dynamische Patternausführung mit Low Level Treiberfunktionen

7.3.5.1 Hauptfunktion

```
/* Example using low level driver functions for dynamic pattern execution */
#include <ansi_c.h>
#include "rspio4.h"

#define PATTERN_COUNT       8
#define PATTERN_PERIOD      1.0e-6
#define FETCH_TIMEOUT       0.1

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::11::INSTR";

static ViUInt32 stimData[PATTERN_COUNT] = {
  0x00000000,
  0x00000010,
  0x0000001F, /* tri state */
  0x00000000,
  0x00000010,
  0x0000001F, /* tri state */
  0x00000000,
  /* etc. */
  0xFFFFFFFF
};

static ViUInt32 stimTristate[PATTERN_COUNT] = {
  0x00000000,
  0x00000000,
  0x00000003, /* port 0 and port 1 tri state (OUT1 to OUT8) */
  0x00000000,
  0x00000000,
  0x00000003, /* port 0 and port 1 tri state (OUT1 to OUT8) */
  0x00000000,
  /* etc. */
  0x000000FF  /* port 0 to port 7 tri state (OUT1 to OUT32) */
};

static ViStatus sta;
```

```
static ViSession vi;
static ViUInt16 memId;

/* data buffer */
static RSPIO4_DATA_32BIT_TRISTATE stimulus[PATTERN_COUNT];
static RSPIO4_DATA_32BIT          response[PATTERN_COUNT];

/* prototypes */
static void chk ( char * funcName );
static void runTest ( void );
static void createPatternSet ( void );
/* FUNCTION ******************************************************************/
/* loads the driver and runs the test
*******************************************************************************/
int main(int argc, char *argv[])
{
  printf("Use of low level driver functions for dynamic pattern execution\n\n");

  /* open a session to the device driver */
  sta = rspio4_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio4_InitWithOptions");

  if (VI_SUCCESS == sta)
  {
    runTest();

    /* close the driver */
    sta = rspio4_close(vi);
    chk ("rspio4_close");
  }

  printf("\nPress 'Enter' to terminate\n");
  getchar();

  return 0;
}
```

7.3.5.2 Fehlerbehandlung

Diese Funktion überprüft die Rückgabewerte eines Treiberaufrufs. Im Fehlerfall wird eine Nachricht ausgegeben.

```
/* FUNCTION ******************************************************************/
/* checks the return status of a driver call
*******************************************************************************/
static void chk ( char * funcName )
{
  if ( sta != VI_SUCCESS )
  {
```

```
    char errorMessage[256];

    rspio4_error_message(vi, sta, errorMessage);
    printf ("%s returned 0x%08X; %s\n", funcName, sta, errorMessage);
  }
}
```

7.3.5.3 Ausführung des Digitaltests

```
/* FUNCTION ******************************************************************/
/* configures the digital test, loads the test and executes it.
*******************************************************************************/
static void runTest ( void )
{
  ViUInt32 respByteCount;
  ViInt32 loopIdx;

  ViReal64 voltageHigh = 3.3;
  ViReal64 voltageLow = 0.0;
  ViReal64 currentLimit = 0.1;
  ViReal64 thresholdHigh = 2.0;
  ViReal64 thresholdLow = 0.8;

  ViReal64 triggerDelayStim = 0.0;
  ViReal64 triggerDelayResp = PATTERN_PERIOD / 2.0;

  /* set voltage range;   */
  sta = rspio4_ConfigurePortVoltageRange(vi,
    RSPIO4_MASK_PORT_ALL, RSPIO4_PORT_VOLTAGE_RANGE_2);
  chk ("rspio4_ConfigurePortVoltageRange");

  /* configure driver voltage levels */
  sta = rspio4_ConfigureStimPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_STIM_MODE_TTL, voltageHigh, voltageLow, currentLimit);
  chk ("rspio4_ConfigureStimPort");

  /* configure sensor: set all ports, use hysteresis comparator mode */
  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, thresholdHigh, thresholdLow);
  chk ("rspio4_ConfigureRespPort");

  /* configure module for dynamic test and result collection */
  sta = rspio4_ConfigureMode(vi, RSPIO4_VAL_EXECUTE_DYNAMIC,
    RSPIO4_VAL_COLLECT_ALL);
  chk ("rspio4_ConfigureMode");

  /* configure 32 bit wide dynamic operation: OUT1 to OUT32 will be operated
     dynamically; no static channels */
  sta = rspio4_ConfigureStimMode (vi, RSPIO4_VAL_CTRL_32BIT_TRISTATE);
```

```
  chk ("rspio4_ConfigureStimMode");

  /* configure 32 bit wide dynamic operation */
  sta = rspio4_ConfigureRespMode(vi, RSPIO4_VAL_CTRL_32BIT);
  chk ("rspio4_ConfigureRespMode");

  /* configure stimulus timing */
  sta = rspio4_ConfigureStimTiming(vi, triggerDelayStim,
    PATTERN_PERIOD, PATTERN_COUNT);
  chk ("rspio4_ConfigureStimTiming");

  /* configure response timing */
  sta = rspio4_ConfigureRespTiming(vi, triggerDelayResp,
    PATTERN_PERIOD, PATTERN_COUNT);
  chk ("rspio4_ConfigureRespTiming");

  /* create and load the pattern set */
  createPatternSet();

  /* start pattern execution */
  sta = rspio4_ExecutePattern(vi, memId);
  chk ("rspio4_ExecutePattern");

  /* read the response data */
  sta = rspio4_FetchPatternResponseData(vi, sizeof(response),
    (ViAddr *)response, FETCH_TIMEOUT, & respByteCount);
  chk ("rspio4_FetchPatternResponseData");

  /* evaluate response data */
  printf("Response data:\n");

  for (loopIdx = 0; loopIdx < PATTERN_COUNT; loopIdx++)
  {
    printf("%2d 0x%08X\n", loopIdx, response[loopIdx].data);
  }

  /* free stimulus memory */
  sta = rspio4_DiscardData(vi, memId);
  chk ("rspio4_DiscardData");
}
```

7.3.5.4 Erzeugen eines Pattern-Sets

```
/* FUNCTION ******************************************************************/
/* creates and loads a pattern set
*******************************************************************************/
static void createPatternSet ( void )
{
  int      loopIdx, portIdx;
```

```
  ViUInt32 portMask;
  ViChar   triStateInfo[9];

  /* generate stimulus data */
  printf("Stimulus data:\n");

  for (loopIdx = 0; loopIdx < PATTERN_COUNT; loopIdx++)
  {
    /* add index information to OUT20 to OUT27 */
    stimulus[loopIdx].data = stimData[loopIdx] | (loopIdx << 19);
    stimulus[loopIdx].tristate = stimTristate[loopIdx];

    portMask = RSPIO4_MASK_PORT0;
    for (portIdx = 7; portIdx >= 0; portIdx--)
    {
      if (stimulus[loopIdx].tristate & portMask)
      {
        triStateInfo[portIdx] = 'F';
      }
      else
      {
        triStateInfo[portIdx] = '0';
      }
      portMask = portMask << 1;
    }

    printf("%2d 0x%08X\n", loopIdx, stimulus[loopIdx].data);
    printf("   0x%s\n", triStateInfo);
  }

  /* load data to stimulus RAM */
  sta = rspio4_LoadData (vi, (ViAddr)s_stimData, sizeof(s_stimData),
    RSPIO4_VAL_DATA_TYPE_STIM, & memId);
  chk ("rspio4_LoadData");
}
```

7.3.6 Getriggerte Patternausführung

In diesem Beispiel triggert ein Puls auf der Triggerleitung PXI0 der TSVP Backplane die Patternausgabe. Der Triggerpuls wird vom R&S TS-PIO4 Modul selbst erzeugt. Andere Messmodule im TSVP Rahmen können dieses Signal auch verwenden, um eine getriggerte Messung durchzuführen. Es ist natürlich auch möglich weitere R&S TS-PIO4 Module im System mit diesem Signal zu starten. In dem Bespiel werden die Stimulus – und Responselogik (IT1 und IT2) vom gleichen Signal (PXI0) gestartet. Zusätzlich wird der ausgegebene Takt der Stimuluslogik (Internal Trigger Logic Block 1 bzw. IT1) auf den Pin XTO des frontseitigen Steckers geroutet.

7.3.6.1 Hauptprogramm

```
/* Example using low level driver functions for triggered execution */
#include <ansi_c.h>
#include "rspio4.h"

#define PATTERN_COUNT       8
#define PATTERN_PERIOD      1.0e-6

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::11::INSTR";

/* data buffer */
static RSPIO4_DATA_32BIT response[PATTERN_COUNT];
static RSPIO4_DATA_32BIT stimulus[PATTERN_COUNT] = {
  0x00000000,
  0x00000001,
  0x00000002,
  0x00000003,
  0x00000004,
  0x00000005,
  0x00000006,
  0x00000007
};

static ViStatus sta;
static ViSession vi;
static ViUInt16 memId;

/* prototypes */
static void chk ( char * funcName );
static void runTest ( void );

/* FUNCTION ******************************************************************/
/* loads the driver and runs the test
*******************************************************************************/
int main(int argc, char *argv[])
{
  printf("Use of low level driver functions for triggered execution\n\n");

  /* open a session to the device driver */
  sta = rspio4_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio4_InitWithOptions");

  if (VI_SUCCESS == sta)
  {
    runTest();

    /* close the driver */
```

```
    sta = rspio4_close(vi);
    chk ("rspio4_close");
  }

  printf("\nPress 'Enter' to terminate\n");
  getchar();

  return 0;
}
```

7.3.6.2 Fehlerbehandlung

```
/* FUNCTION ****************************************************************/
/* checks the return status of a driver call
*******************************************************************************/
static void chk ( char * funcName )
{
  if ( sta != VI_SUCCESS )
  {
    char errorMessage[256];

    rspio4_error_message(vi, sta, errorMessage);
    printf ("%s returned 0x%08X; %s\n", funcName, sta, errorMessage);
  }
}
```

7.3.6.3 Getriggerter Digitaltest

```
/* FUNCTION ****************************************************************/
/* configures the digital test, loads the test and executes it.
*******************************************************************************/
static void runTest ( void )
{
  ViUInt32 respByteCount;
  ViInt32 loopIdx;

  /* set voltage range;   */
  sta = rspio4_ConfigurePortVoltageRange(vi,
    RSPIO4_MASK_PORT_ALL, RSPIO4_PORT_VOLTAGE_RANGE_2);
  chk ("rspio4_ConfigurePortVoltageRange");

  /* configure driver voltage levels */
  sta = rspio4_ConfigureStimPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_STIM_MODE_TTL, 3.3, 0.0, 0.1);
  chk ("rspio4_ConfigureStimPort");

  /* configure sensor: set all ports, use hysteresis comparator mode */
  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, 2.0, 0.8);
```

```
chk ("rspio4_ConfigureRespPort");

/* configure module for dynamic test and result collection */
sta = rspio4_ConfigureMode(vi, RSPIO4_VAL_EXECUTE_DYNAMIC,
  RSPIO4_VAL_COLLECT_ALL);
chk ("rspio4_ConfigureMode");

/* configure 32 bit wide dynamic operation */
sta = rspio4_ConfigureStimMode(vi, RSPIO4_VAL_CTRL_32BIT);
chk ("rspio4_ConfigureStimMode");

sta = rspio4_ConfigureRespMode(vi, RSPIO4_VAL_CTRL_32BIT);
chk ("rspio4_ConfigureRespMode");

/* configure stimulus timing */
sta = rspio4_ConfigureStimTiming(vi, 0.0, PATTERN_PERIOD, PATTERN_COUNT);
chk ("rspio4_ConfigureStimTiming");

/* configure response timing */
sta = rspio4_ConfigureRespTiming(vi, PATTERN_PERIOD / 2.0,
  PATTERN_PERIOD, PATTERN_COUNT);
chk ("rspio4_ConfigureRespTiming");

/* load data to stimulus RAM */
sta = rspio4_LoadData (vi, (ViAddr)stimulus, sizeof(stimulus),
  RSPIO4_VAL_DATA_TYPE_STIM, & memId);
chk ("rspio4_LoadData");

/* connect all outputs to inputs */
sta = rspio4_ConnectInOut (vi, RSPIO4_MASK_PORT_ALL, RSPIO4_MASK_PORT_ALL);
chk ("rspio4_ConnectInOut");

/* configure trigger logic blocks to output the clock pulses  */
sta = rspio4_ConfigItTrigOut(vi, RSPIO4_IT_MASK_IT1 | RSPIO4_IT_MASK_IT2,
  RSPIO4_VAL_TRIG_IT_OUT);
chk ("rspio4_ConfigItTrigOut");

/* configure XTO to output the signal of trigger logic block 1 (stimulus) */
sta = rspio4_ConfigXTO(vi, RSPIO4_VAL_TRIG_IT1);
chk ("rspio4_ConfigXTO");


/* general purpose trigger should output a pulse at PXI0 */
sta = rspio4_ConfigPxiTrigOut(vi, RSPIO4_TRIG_MASK_PXI0, RSPIO4_VAL_TRIG_GP,
  RSPIO4_TRIG_MASK_PXI0, RSPIO4_TRIG_MASK_PXI0);
chk ("rspio4_ConfigPxiTrigOut");


/* stimmulus and response should be triggert by PXI0 */
sta = rspio4_ConfigHWTriggerInput(vi, RSPIO4_IT_MASK_IT1 | RSPIO4_IT_MASK_IT2,
```

```
    RSPIO4_TRIG_MASK_PXI0, RSPIO4_TRIG_MASK_PXI0,
                                        RSPIO4_VAL_FALLING_EDGE);
  chk ("rspio4_ConfigHWTriggerInput");

  /* load the stimulus buffer for triggered pattern generation */
  sta = rspio4_LoadStimBuffer (vi, memId);
  chk ("rspio4_LoadStimBuffer");

  /* arm trigger logic blocks */
  sta = rspio4_EnableHWTrigger(vi, RSPIO4_IT_MASK_IT1 | RSPIO4_IT_MASK_IT2,
    RSPIO4_IT_MASK_IT1 | RSPIO4_IT_MASK_IT2);
  chk ("rspio4_EnableHWTrigger");

  /* wait until all pending hardware configurations have settled */
  sta = rspio4_WaitForDebounce(vi, 100);
  chk ("rspio4_WaitForDebounce");

  /* generate the general purpose trigger pulse at PXI0 */
  sta = rspio4_InitiateSWTrigger(vi, RSPIO4_IT_MASK_GP);
  chk ("rspio4_InitiateSWTrigger");

  /* wait until the pattern execution has finished; read the response data */
  sta = rspio4_FetchPatternResponseData(vi, sizeof(response),
    (ViAddr *)response, 0.1, & respByteCount);
  chk ("rspio4_FetchPatternResponseData");

  /* evaluate response data */
  printf("Idx | Stimulus   | Response  \n");
  printf("----------------------------\n");

  for (loopIdx = 0; loopIdx < PATTERN_COUNT; loopIdx++)
  {
    printf("% 3d | 0x%08X | 0x%08X\n",
      loopIdx, stimulus[loopIdx].data, response[loopIdx].data);
  }

  /* free stimulus memory */
  sta = rspio4_DiscardData(vi, memId);
  chk ("rspio4_DiscardData");
}
```

8 Selbsttest

Das Digitale Funktionstestmodul R&S TS-PIO4 besitzt integrierte Selbsttestfähigkeit. Folgende Tests sind implementiert:

- LED-Test
- Einschalttest
- TSVP-Selbsttest

8.1 LED Test

Nach dem Einschalten leuchten alle drei LEDs für ca. eine Sekunde. Dies signalisiert, dass die 5 V-Versorgungsspannung anliegt und alle LEDs in Ordnung sind. Folgende Aussagen können über die verschiedenen Anzeigezustände gemacht werden:

LED	Beschreibung
eine einzelne LED leuchtet nicht	Hardwareproblem auf dem Modul; LED defekt
alle LEDs leuchten nicht	+5 V-Versorgungsspannung fehlt

8.2 Einschalttest

Parallel zum LED-Test verläuft der Einschalttest. Folgende Aussagen können über die verschiedenen Anzeigezustände der LEDs gemacht werden:

LED	Beschreibung
PWR LED (grün) an	alle Versorgungsspannungen vorhanden
PWR LED (grün) aus	mindestens eine Versorgungsspannung fehlt
ERR LED (rot) aus	es liegt kein Fehler vor
ERR LED (rot) an	Laden des FPGA ist fehlgeschlagen

8.3 TSVP Selbsttest

Im Rahmen des TSVP-Selbsttests wird ein tiefgehender Test des Moduls R&S TS-PIO4 durchgeführt und ein ausführliches Protokoll generiert. Dies geschieht über die “ Selbsttest Support Library“.

Das Analoge Stimulus- und Messmodul R&S TS-PSAM wird als Messeinheit im TSVP-Selbsttest zur Überprüfung der Triggerleitung PXI_TRIG[0-7] verwendet.

Informationen zum Starten des Selbsttests und zur Reihenfolge der notwendigen Arbeitsschritte sowie eine detaillierte Beschreibung der geprüften Parameter und Abläufe befindet sich im *Service Manual R&S CompactTSVP / R&S PowerTSVP*.

9 Schnittstellenbeschreibung

9.1 R&S TS-PIO4

9.1.1 Steckverbinder X10

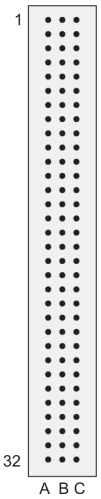


Bild 9-1: Steckverbinder X10 (Ansicht: Steckseite)

Tabelle 9-1: Belegung R&S TS-PIO4 Steckverbinder X10

	A	B	C
1		AUX1	EXT_CLK
2		AUX2	
3		AUX3	
4		AUX4	
5	OUT1	OUT2	OUT3
6	IN1	IN2	IN3
7	OUT4	OUT5	OUT6
8	IN4	IN5	IN6
9	OUT7	OUT8	GNDNO
10	IN7	IN8	GND
11	OUT9	OUT10	OUT11
12	IN9	IN10	IN11
13	OUT12	OUT13	OUT14

	A	B	C
14	IN12	IN13	IN14
15	OUT15	OUT16	GNDNO
16	IN15	IN16	GND
17	OUT17	OUT18	OUT19
18	IN17	IN18	IN19
19	OUT20	OUT21	OUT22
20	IN20	IN21	IN22
21	OUT23	OUT24	GNDNO
22	IN23	IN24	GND
23	OUT25	OUT26	OUT27
24	IN25	IN26	IN27
25	OUT28	OUT29	OUT30
26	IN28	IN29	IN30
27	OUT31	OUT32	GNDNO
28	IN31	IN32	GND
29	XTO		
30	XTI		
31			GND
32			CHA-GND

9.1.2 Steckverbinder X20

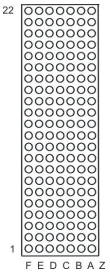


Bild 9-2: Steckverbinder X20 (Ansicht: Steckseite)

Pin	F	E	D	C	B	A	Z	
22		GA0	GA1	GA2	GA3	GA4		
21		OD8						
20		+5V	GND	+5V	AUX1A_COM	AUX2A_COM		
19		AUX1A_COM	AUX2A_COM	+5V	GND	-12V		
18		PXI_TRIG6	GND	PXI_TRIG5	PXI_TRIG4	PXI_TRIG3		J20
17		PXI_CLK10	PO_2	PO_1	GND	PXI_TRIG2		
16		PXI_TRIG7	GND	PO_3	PXI_TRIG0	PXI_TRIG1		
15			+5V	PO_4	GND			
14	NC	AUX1A_NC	AUX1A_NO		AUX3A_NO	AUX3A_NC	NC	C
13	NC	AUX1A_NC	AUX1A_NO		AUX3A_NO	AUX3A_NC	NC	O
12	NP	AUX1A_COM	AUX2A_NO		AUX4A_NO	AUX3A_COM	NP	N
11	NP	AUX1A_COM	AUX2A_NO	IL1	AUX4A_NO	AUX3A_COM	NP	N
10	NC	AUX2A_COM	AUX2A_NC		AUX4A_NC	AUX4A_COM	NC	E
9	NC	AUX2A_COM	AUX2A_NC		AUX4A_NC	AUX4A_COM	NC	C
8	NC	AUX1B_COM	AUX1B_NO		AUX3B_NC	AUX3B_COM	NC	T
7	NC	AUX1B_COM	AUX1B_NC	IL2	AUX3B_NO	AUX3B_COM	NC	O
6	NC	AUX2B_COM	AUX2B_NO		AUX4B_NC	AUX4B_COM	NC	R
5	NC	AUX2B_COM	AUX2B_NC		AUX4B_NO	AUX4B_COM	NC	
4	NC						NC	
3		RSA0	RRST#		GND	RSDO		
2		+12V	RSDI	RSA1		RSCLK		
1		+5V			GND	RCS#		

Bild 9-3: Belegung Steckverbinder X20

9.1.3 Steckverbinder X30

Tabelle 9-2: Belegung R&S TS-PIO4 Steckverbinder X30

	E	D	C	B	A
7					
6			GND		
5					
4					
3					
2					
1					

9.1.4 Steckverbinder X1 (cPCI Bus)

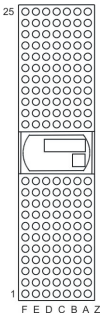


Bild 9-4: Steckverbinder X1 (Ansicht: Steckseite)

Pin	F	E	D	C	B	A	Z	
25	GND	5V	3.3V	ENUM#	REQ64#	5V	GND	
24	GND	ACK64#	AD[0]	V(I/O)	5V	AD[1]	GND	
23	GND	AD[2]	5V	AD[3]	AD[4]	3.3V	GND	
22	GND	AD[5]	AD[6]	3.3V	GND	AD[7]	GND	
21	GND	C/BE[0]#	M66EN	AD[8]	AD[9]	3.3V	GND	
20	GND	AD[10]	AD[11]	V(I/O)	GND	AD[12]	GND	X1
19	GND	AD[13]	GND	AD[14]	AD[15]	3.3V	GND	
18	GND	C/BE[1]#	PAR	3.3V	GND	SERR#	GND	
17	GND	PERR#	GND	IPMB_SDA	IPMB_SCL	3.3V	GND	C
16	GND	LOCK#	STOP#	V(I/O)	GND	DEVSEL#	GND	O
15	GND	TRDY#	BD_SEL#	IRDY#	FRAME#	3.3V	GND	N
12..14				Key Area				N
11	GND	C/BE[2]#	GND	AD[16]	AD[17]	AD[18]	GND	E
10	GND	AD[19]	AD[20]	3.3V	GND	AD[21]	GND	C
9	GND	AD[22]	GND	AD[23]	IDSEL	C/BE[3]#	GND	T
8	GND	AD[24]	AD[25]	V(I/O)	GND	AD[26]	GND	O
7	GND	AD[27]	GND	AD[28]	AD[29]	AD[30]	GND	R
6	GND	AD[31]	CLK	3.3V	GND	REQ#	GND	
5	GND	GNT#	GND	RST#	BSRSV	BSRSV	GND	
4	GND	INTS	INTP	V(I/O)	HEALTHY#	IPMB_PWR	GND	
3	GND	INTD#	5V	INTC#	INTB#	INTA#	GND	
2	GND	TDI	TDO	TMS	5V	TCK	GND	
1	GND	5V	+12V	TRST#	-12V	5V	GND	

Bild 9-5: Belegung Steckverbinder X1

10 Technische Daten

Die technischen Daten des Moduls R&S TS-PIO4 sind in den entsprechenden Datenblättern angegeben.

Bei Diskrepanzen zwischen Angaben in diesem Handbuch und den Werten im Datenblatt gelten die Datenblattwerte.