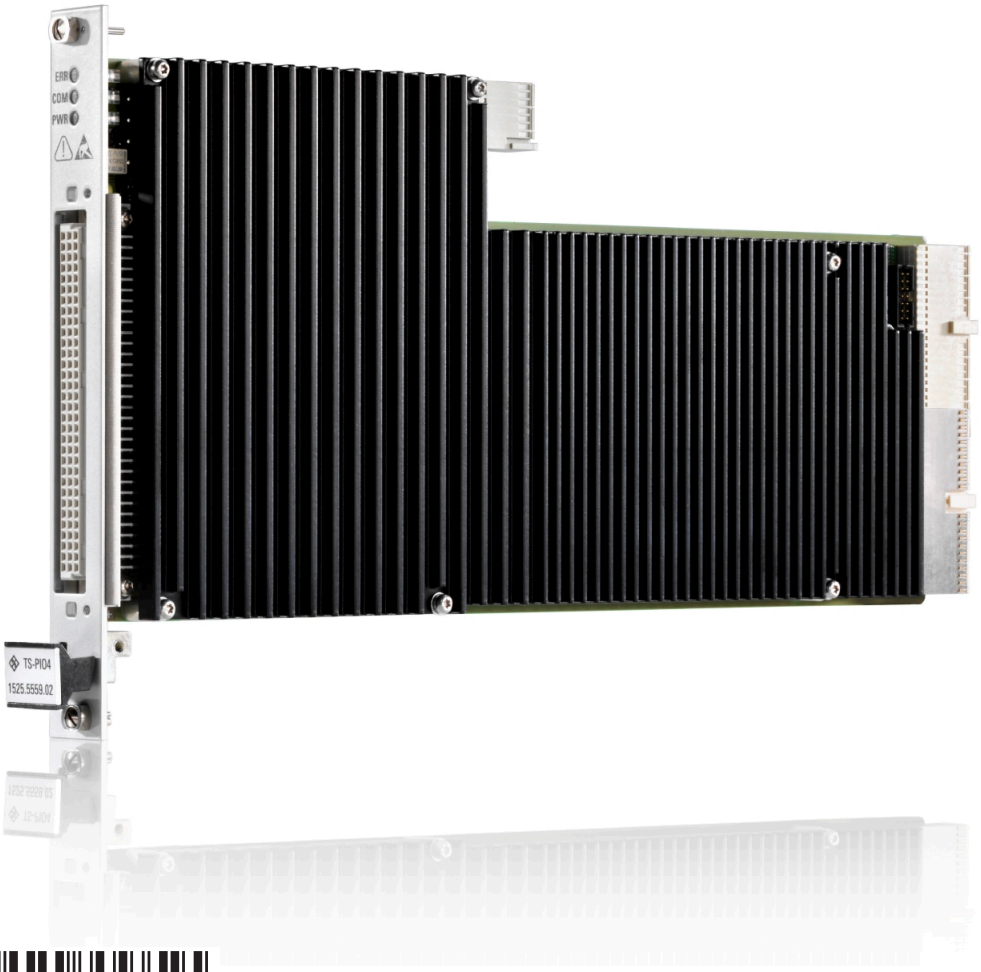
R&S®TS-PIO4
Digital Functional Test Module
User Manual



1178319202
Version 06

ROHDE&SCHWARZ
Make ideas real

This manual describes the following R&S®TSVP module:

- R&S®TS-PIO4 (1525.5559.02)

Muehldorfstr. 15, 81671 Muenchen, Germany
Phone: +49 89 41 29 - 0
Email: info@rohde-schwarz.com
Internet: www.rohde-schwarz.com

1178.3192.02 | Version 06 | R&S®TS-PIO4

The following abbreviations are used in this manual: R&S®TS-PIO4 is abbreviated to R&S TS-PIO4.

Contents

1 Safety information (multilingual)

This option or accessory is designed for a specific Rohde & Schwarz product. Multilingual safety information is delivered with the product. Follow the provided installation instructions.

Esta opción o este accesorio están diseñados para un producto Rohde & Schwarz concreto. El producto va acompañado de información de seguridad en varios idiomas. Siga las instrucciones de instalación puestas a disposición.

Diese Option oder dieses Zubehör ist für ein bestimmtes Rohde & Schwarz Produkt vorgesehen. Mit dem Produkt werden mehrsprachige Sicherheitsinformationen geliefert. Befolgen Sie die mitgelieferten Installationsanweisungen.

Cette option ou cet accessoire est conçu pour un produit Rohde & Schwarz spécifique. Des informations de sécurité multilingues sont fournies avec le produit. Suivez les instructions d'installation fournies.

Questa funzione opzionale o accessoria è progettata per un prodotto Rohde & Schwarz specifico. Con il prodotto sono fornite informazioni sulla sicurezza in formato multilingue. Seguire le istruzioni di installazione allegate.

Esta(e) opção ou acessório foi concebida(o) para um produto específico da Rohde & Schwarz. Serão fornecidas informações de segurança multilingues com o produto. Siga as instruções de instalação fornecidas.

Αυτή η προαιρετική επιλογή ή εξάρτημα έχει σχεδιαστεί για συγκεκριμένο προϊόν Rohde & Schwarz. Μαζί με το προϊόν παρέχονται πληροφορίες ασφαλείας σε πολλές γλώσσες. Ακολουθήστε τις παρεχόμενες οδηγίες εγκατάστασης.

Din l-għażla jew aċċessorju huma mfassla għal prodott Rohde & Schwarz speċifiku. L-informazzjoni multilingwi dwar is-sikurezza hija pprovduta mal-prodott. Segwi l-istruzzjonijiet ipprovduti għall-installazzjoni.

Deze optie of dit accessoire is ontwikkeld voor een specifiek product van Rohde & Schwarz. Het product wordt geleverd met veiligheidsinformatie in meerdere talen. Volg de meegeleverde installatie-instructies.

Denne mulighed eller tilbehørsdel er designet til et specifikt Rohde & Schwarz produkt. En flersproget sikkerhedsanvisning leveres sammen med produktet. Følg de medfølgende installationsanvisninger.

Detta tillval eller tillbehör är avsett för en särskild produkt från Rohde & Schwarz. Säkerhetsinformation på flera språk medföljer produkten. Följ de medföljande installationsanvisningarna.

Tämä vaihtoehto tai lisävaruste on suunniteltu tietylle Rohde & Schwarz -yrityksen tuotteelle. Tuotteen mukana on toimitettu monikieliset turvallisuusohjeet. Noudata annettuja asennusohjeita.

Dette alternativet eller ekstrautstyret er utformet for et spesifikt Rohde & Schwarz produkt. Flerspråklig sikkerhetsinformasjon leveres med produktet. Overhold installasjonsveiledningen som følger med.

See valik või lisaseade on mõeldud konkreetsele Rohde & Schwarz tootele. Tootega on kaasas mitmekeelne ohutusteave. Järgige kaasasolevaid paigaldusjuhiseid.

Šī opcija vai piederums ir izstrādāts īpaši Rohde & Schwarz produktam. Produktam pievienota drošības informācija vairākās valodās. Ievērojiet sniegtos uzstādīšanas norādījumus.

Ši parinktis ar priedas skirti konkrečiam Rohde & Schwarz gaminiui. Su gaminiu pateikiama saugos informacijos keliomis kalbomis. Laikykitės pateikiamų montavimo nurodymų.

Þessi auka- eða fylgibúnaður er hannaður fyrir tiltekna Rohde & Schwarz vöru. Öryggisupplýsingar á mörgum tungumálum fylgja með vörunni. Fylgið meðfylgjandi uppsetningarleiðbeiningum.

Tá an rogha nó an oiriúint seo ceaptha le haghaidh táirge Rohde & Schwarz sonrach. Cuirtear eolas sábháilteachta ilteangach ar fáil leis an táirge. Lean na treoracha suiteála a thugtar.

Эта опция или принадлежность предназначена для конкретного продукта Rohde & Schwarz. В комплект поставки продукта входят инструкции по технике безопасности на нескольких языках. Соблюдайте прилагаемые инструкции по установке.

Ця опція або приладдя призначені для конкретного виробу Rohde & Schwarz. Інструкції з техніки безпеки кількома мовами постачаються разом із виробом. Дотримуйтеся наданих інструкцій зі встановлення.

Ta opcja lub akcesorium jest przeznaczone do określonego produktu Rohde & Schwarz. Dostarczany produkt zawiera informacje w wielu językach dotyczące bezpieczeństwa. Należy postępować zgodnie z dostarczonymi instrukcjami instalacji.

Tato varianta nebo příslušenství je určeno pro konkrétní produkt Rohde & Schwarz. S produktem jsou dodávány vícejazyčné bezpečnostní informace. Řiďte se přiloženými pokyny k instalaci.

Táto verzia alebo príslušenstvo je navrhnutá pre špecifický výrobok Rohde & Schwarz. S výrobkom sa dodávajú viacjazyčné bezpečnostné pokyny. Riaďte sa dodanými pokynmi na inštaláciu.

Ta možnost ali dodatek je zasnovan za določen izdelek podjetja Rohde & Schwarz. Izdelku so priložena varnostna navodila v več jezikih. Upoštevajte priložena navodila za namestitev.

Ezt a beállítást vagy tartozékot egy adott Rohde & Schwarz termékhez tervezték. A termékhez többnyelvű biztonsági információt mellékelünk. Kövesse a mellékelt szerelési utasításokat.

Тази опция или аксесоар са проектирани за специфичен продукт на Rohde & Schwarz. Многоезикова информация за безопасност се доставя с продукта. Следвайте предоставените инструкции за монтаж.

Ova opcija ili oprema namijenjena je za određeni proizvod tvrtke Rohde & Schwarz. Uz proizvod su dostavljene sigurnosne napomene na više jezika. Pratite isporučene upute za ugradnju.

Ova opcija ili pribor je dizajniran za određeni Rohde & Schwarz proizvod. Proizvodu su priložene sigurnosne informacije na više jezika. Slijedite priložena uputstva za instalaciju.

Ova opcija ili dodatni pribor je projektovan za određeni Rohde & Schwarz proizvod. Bezbednosne informacije na više jezika se isporučuju uz proizvod. Sledite dostavljena uputstva za instalaciju.

Această opțiune sau acest accesoriu a fost conceput pentru un produs specific Rohde & Schwarz. Informațiile multilingve privind siguranța sunt livrate împreună cu produsul. Urmați instrucțiunile de instalare furnizate.

Ky opsion ose aksesor është krijuar për një produkt specifik Rohde & Schwarz. Bashkë me produktin jepen edhe informacionet e sigurisë në shumë gjuhë. Ndiqni udhëzimet e dhëna të instalimit.

Оваа опција или додаток се наменети за одреден производ на Rohde & Schwarz. Со производот се испорачани повеќејазични безбедносни упатства. Следете ги дадените упатства за инсталација.

Bu opsiyon veya aksesuar, belirli bir Rohde & Schwarz ürünü için tasarlanmıştır. Çok dilli güvenlik uyarıları ürünle birlikte teslim edilir. Size sağlanan kurulum talimatlarına uyun.

אפשרות זו או האביזר מיועדים למוצר ספציפי של Rohde & Schwarz. מידע רב-לשוני בנושא בטיחות מצורף למוצר. יש לפעול בהתאם להנחיות ההתקנה המצורפות.

تم تصميم هذا الخيار أو الملحق لمنتج معين من منتجات Rohde & Schwarz. يتم تزويد معلومات السلامة متعددة اللغات مع المنتج. اتبع تعليمات التركيب الموضحة.

این قابلیت یا وسیله جانبی منحصراً برای محصول به خصوص Rohde & Schwarz طراحی شده است. اطلاعات ایمنی چندزبانه همراه این دستگاه ارائه شده است. دستورالعمل‌های نصب ارائه شده را دنبال کنید.

اس اختیار یا حصے کو مخصوص Rohde & Schwarz پروڈکٹ کے لئے تیار کیا گیا ہے۔ پروڈکٹ کے ساتھ کثیر السانی زبانوں میں تحفظ کی معلومات فراہم کی جاتی ہیں۔ فراہم کردہ تنصیب کی ہدایات پر عمل کریں۔

Şu opsiýa ýa-da esbap Rohde & Schwarz anyk önüm üçin niýetlenilen. Dürli dildäki howpsuzlyk barada maglumat önüm bilen bile üpjün edilýär. Üpjün edilen gurnama ugrukdyrmalaryny ýerine ýetiriň.

इस विकल्प या एक्सेसरी को एक विशेष Rohde & Schwarz उत्पाद के लिए डिज़ाइन किया गया है. उत्पाद के साथ बहुभाषी सुरक्षा जानकारी दी जाती है. प्रदान किए गए इंस्टालेशन अनुदेशों का पालन करें.

本选件或附件专门设计用于特定的 Rohde & Schwarz 产品。产品随附多种语言版本的安全资讯。谨遵文件中的安装说明。

本オプションアクセサリは、特定の Rohde & Schwarz 製品向けに設計されています。多言語で記載された安全情報が製品に付属します。付属のインストール手順に従ってください。

이 옵션 또는 액세서리는 특정 Rohde & Schwarz 제품용으로 설계되었습니다. 제품과 함께 다국어로 작성된 안전 정보가 제공됩니다. 함께 제공된 설치 지침을 따르십시오.

本選配或配件專門設計用於特定的 Rohde & Schwarz 產品。產品隨附多種語言版本的安全資訊。遵守文件中的安裝說明。

Tùy chọn hoặc phụ kiện này dành riêng cho một sản phẩm Rohde & Schwarz cụ thể. Thông tin an toàn đa ngôn ngữ được cung cấp kèm theo sản phẩm. Thực hiện theo hướng dẫn lắp đặt kèm theo.

ตัวเลือกหรืออุปกรณ์เสริมนี้ออกแบบมาสำหรับผลิตภัณฑ์ Rohde & Schwarz โดยเฉพาะ โดยจะมีการจัดส่งข้อมูลด้านความปลอดภัยหลายภาษามาให้พร้อมกับผลิตภัณฑ์ ปฏิบัติตามคำแนะนำในการติดตั้งที่ให้ไว้

Pilihan atau aksesori ini direka bentuk untuk produk Rohde & Schwarz yang tertentu. Maklumat keselamatan berbilang bahasa disertakan bersama produk. Ikut arahan pemasangan yang diberikan.

Opsi atau aksesori ini dirancang untuk produk Rohde & Schwarz tertentu. Informasi keamanan dalam beberapa bahasa juga disertakan bersama produk. Ikuti petunjuk pemasangan yang disediakan.

Esta opción o este accesorio están diseñados para un producto Rohde & Schwarz en concreto. El producto va acompañado de información de seguridad en varios idiomas. Siga las instrucciones de instalación proporcionadas con el producto.

Esta opção ou acessório foi desenvolvido para um produto Rohde & Schwarz específico. Informações de segurança em vários idiomas acompanham o produto. Siga as instruções de instalação disponibilizadas.

2 Documentation overview

This section provides an overview of the R&S TSVP (test system versatile platform) user documentation.

All documents are delivered with the Generic Test Software Library ("R&S GTSL") installation package. After installing the software, you can open all the documentation from the Windows "Start" menu. Additionally, you can find detailed information about the software interfaces in the "R&S GTSL Help" folder in the Windows "Start" menu.

The user documentation and "R&S GTSL" installation package are also available for download in GLORIS at:

https://gloris.rohde-schwarz.com/

For details, see the R&S TSVP Getting Started manual.

2.1 Getting started manual

Introduces the R&S TSVP (test system versatile platform) and describes how to set up and start working with the product. It includes safety information.

A printed version is delivered with the instrument.

2.2 User manuals

Separate manuals are provided for the base units, the individual plug-in module types, as well as for the control software and the calibration tool:

- Base unit manual
  The base unit user manuals introduce the base units and describes how to set up and operate the product. It includes safety information and information on maintenance and instrument interfaces. It includes the contents of the getting started manual.
- Plug-in module manuals
  Contain the description of the specific modules. Basic information on setting up the R&S TSVP (test system versatile platform) is not included.
- In-System calibration user manuals
  Provide all the information required for installation and operation of the in-system calibration R&S TS-ISC solution.
- Control software
  - R&S GTSL
    Generic Test Software Library
  - R&S EGTSL
    Enhanced Generic Test Software Library
  - R&S IC-Check

Generic Test Software Library

2.3 System manual

Describes the complete R&S TSVP (test system versatile platform) as a whole, including the combined use of R&S CompactTSVP and R&S PowerTSVP, plug-in modules and generic test software. It also includes typical use cases.

Additionally, it describes known installation problems (hardware and software) along with possible solutions.

2.4 Service manual

Describes the self-test to check correct operation, troubleshooting and fault elimination, and contains mechanical drawings and spare part lists.

2.5 Printed safety instructions

Provides safety information in many languages. The printed document is delivered with the product.

2.6 Brochures and specifications

Separate brochures are provided for the base unit, the individual plug-in module types, as well as for the control software. The brochures provide an overview of the base units and each additional module, and also contain the technical specifications. They also list the hardware options and their order numbers, and optional accessories.

2.7 Release notes and open source acknowledgment

The release notes list new features, improvements and known issues of the current software version. In addition, the available firmware versions and the firmware update procedure for plug-in modules are described.

The open-source acknowledgment document provides verbatim license texts of the used open source software.

3 Welcome to the R&S TS-PIO4

This manual describes the function and operation of the Rohde & Schwarz digital functional test module R&S TS-PIO4 for use in the test system versatile platform R&S TSVP. The module is controlled by the PXI bus and occupies one slot on the front side of the R&S TSVP.

The R&S TS-PIO4 digital functional test module is used wherever digital circuits are tested by flexibly programmable static or dynamic stimulation and the response is recorded.

Deterministic, simultaneous stimulation and recording of digital signals makes it possible to simulate operating conditions in a manner that is very close to reality. A large local memory pool and an independent sequence controller are available on the module to ensure that the precise and predictable time response can be maintained for output, recording and analysis of the bit patterns. Extensive trigger options via the PXI trigger bus enable synchronization with additional R&S TS-PIO4 modules or other measurement and stimulus modules. This makes it possible to increase the number of digital channels in an application. Measurements in which signals are to be recorded synchronously are also possible.

Output levels and input thresholds can be programmed in 8 ports each with 4 channels. This allows for optimum adaptability to the requirements of different logic families. The effect of interference signals in the test setup can be minimized by adjusting the hysteresis for the input thresholds.

Safety circuits to protect against short circuits, countervoltages and overvoltages contribute to the robustness of the R&S TS-PIO4 digital functional test module. Due to the extremely space-saving design of the I/O safety circuit and signal conditioning unit, the R&S TS-PIO4 occupies just one PXI slot. This makes it possible to set up very high-performance and compact measuring systems.

A soft panel is available for operation of the module. An IVI-C driver is provided to allow the module to be used under software control.

Features of the R&S TS-PIO4 digital functional test module.

- 32 digital inputs and 32 digital outputs divided into 8 ports each with 4 channels
- Programmable output level range from –6 V to 10 V, output impedance 30 Ω (typ.)
- Resolution 14 bits
- Output current up to 150 mA per channel, 350 mA per port
- Tri-state control in dynamic and static mode
- Two programmable input thresholds per port for hysteresis or level monitoring
- Inputs can be connected to outputs
- Pattern rate up to 40 MS/s per channel x 12.5 nsec. resolution
- Memory depth 2 Msample (32-bit data)
- External clock input
- Synchronization/triggering via PXI trigger bus and via XTI (TTL)
- Timer/counter function
- FPGA-based flexibility

- Standalone self-test capability
- LabWindows IVI-C driver available
- Used in PXI based R&S TSVP base units

4 Module tour

The R&S TS-PIO4 module is designed as a long plug-in module for mounting in the front of PXI based base units.

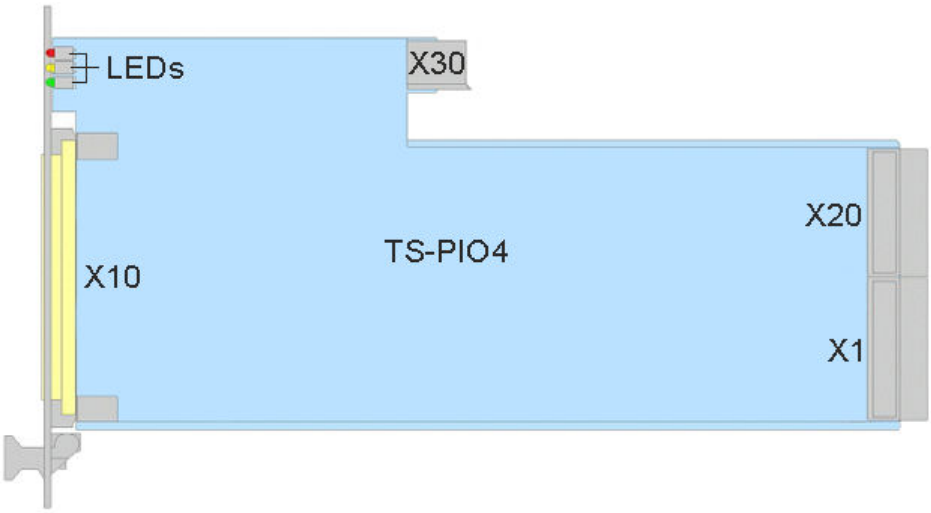


Figure 4-1: Overview of interfaces on R&S TS-PIO4 module

LEDs = Chapter 4.1, "Status LEDs", on page 15
X1 = Chapter 4.2, "Connectors X1 and X20", on page 16
X10 = Chapter 4.3, "Connector X10", on page 16
X20 = Chapter 4.2, "Connectors X1 and X20", on page 16
X30 = Chapter 4.4, "Connector X30", on page 16

Mechanical elements

- Retaining screws on the front to secure the module after installation.
- A locating pin on the front panel of the module that allows you to insert the module correctly into the base unit.

4.1 Status LEDs

The LEDs on the front indicate the current status of the module.

- "PWR" (green LED)
  Indicates that all necessary supply voltages are present.
- "COM" (yellow LED)
  Indicates data exchange via the interface.
- "ERR" (red LED)
  Indicates an error condition if illuminated.

4.2 Connectors X1 and X20

Type: PXI bus

Interface to connect the module to the backplane of PXI based base units.

See Chapter C.4, "Connector X1", on page 93 and Chapter C.2, "Connector X20", on page 91 for a detailed description of the connectors.

4.3 Connector X10

Interface to connect test objects and UUTs to the module.

See Chapter C.1, "Connector X10", on page 90 for a detailed description of the connector.

4.4 Connector X30

Type: Analog bus

Interface to connect the module to the analog bus backplane in the housing of the base unit.

See Chapter C.3, "Connector X30", on page 92 for a detailed description of the connector.

5 Installing the module

The R&S TS-PIO4 is a module installed on the front panel of PXI based base units.

1. Install the module as described in the user manuals of the base units.
2. **WARNING!** Risk of electric shock. The test environment, e.g the UUT or additional power supplies, can supply high voltages to the instruments. In this case, the voltage can also apply to the signal output connectors of the R&S TSVP, in particular the analog bus connector X2.
   Therefore, do not connect or disconnect devices from the X2 connectors while connected to an external power supply or UUT.

   Always connect both ends of the cable connecting the R&S CompactTSVP and R&S PowerTSVP. Thus, you avoid the risk of touching the X2 connector with a possibly hazardous voltage applied.

   Take the system into operation as described in the user manuals of the base units.

6 Typical applications

The R&S TS-PIO4 digital functional test module is used for testing digital modules or devices. This type of functional test is used to check the overall operation of a digital circuit under conditions that are as close to reality as possible. The module does this by applying digital input patterns, measuring the output signals and comparing them with the nominal values.

The R&S TS-PIO4 digital functional test module can be used for tasks such as the following:

- Digital functional test (low-speed/high-speed, IO control)
- Bit pattern stimulation (low-speed/high-speed, digital buses)
- Bit pattern measurement (low-speed/high-speed)
- Monitoring of changes in level state (pattern trigger)
- Digital functional test at component level (no node-forcing or backdriving capability)
- Downloads, e.g. for flash components, serial and parallel

A typical functional test consists of the following elements:

- Adaptation of the pin electronics to the UUT environment (logic level and logic family)
- Definition of the sensor strobe (timing)
- Definition of the stimulus and measuring behavior
- Evaluation of the test result

6.1 Digital functional test – static

Characteristics for which correct interaction of the logic modules is more important than the verification of time-critical limits are tested in the static digital functional test. The application outputs the patterns to be stimulated, reads in the UUT response via the module inputs and compares it with the expected response. Comparison with the nominal values then results in a PASS or FAIL statement. The comparison takes place in the application. As the time response (length of time that a pattern is available; sampling time point of the inputs) is controlled primarily by the host computer, this test cannot be used to test time-critical or chronologically precisely defined sequences. With the static test, the order of the sequence is deterministic, i.e. the chronological order of the patterns as well as the read-in procedure is predictable. How long the interval from one pattern to the next pattern will be and how large the interval from application of a pattern to reading in of the data will be are, however, not predictable. Here, the operating system of the host computer (task switching, etc.) influences the runtimes in a non-predictable way.

6.2 Digital functional test – dynamic

In the real-time test, overall operation of the digital section of a UUT is tested under operating conditions that are as close to reality as possible. For this purpose, digital patterns (vectors and pattern sets) with a precisely defined, usually high clock rate and precise time response are applied at the UUT connections and the responses recorded. Recording is also performed with a precisely defined, reproducible time response. The basic requirement for exact, predictable timing is that the patterns for the stimulus are stored in the memories "downstream of the drivers" and can be processed with a defined time response and at a high speed. To enable this, the module has its own sequence controller in the FPGA. The recorded input data is also initially stored in a memory on the module and later retrieved by the application on the host computer and evaluated for a PASS/FAIL decision.

7 Functions

7.1 Overview

7.1.1 General

The R&S TS-PIO4 digital functional test module provides 8 ports each with 4 digital input pins and output pins. For this purpose, the module has unmultiplexed digital pins, i.e. behind each pin is a separate digital channel. Furthermore, the output voltages of the drivers as well as the comparator thresholds of the inputs can, within certain limits, be freely programmed on the module. This allows the appropriate logic level to be generated for practically every application. All settings as well as clock generation take place on the module itself, which means that no additional stimulus modules are necessary.

Refer to Figure B-1, Figure B-2 and Figure B-3 for details.

The usable level range depends on the configuration of the driver references. The overall level range is divided into 4 sections. The driver levels are generated from the CPCI supply (+5 V and +3.3 V).

The application software must set an appropriate voltage range before the output voltages and comparator thresholds can be configured. Here, the device driver performs a plausibility check and prevents invalid or damaging values from being set.

Table 7-1: Voltage ranges

Range	VL[1]	VH[2]	CVL[3] / CVH[4]
3	+1.5 V to +6.0 V	+1.5 V to +10.0 V	+1.5 V to +7.1 V
2	–1.8 V to +2.7 V	–1.8 V to +10.0 V	–1.8 V to +7.1 V
1	–3.5 V to +1.0 V	–3.5 V to +10.0 V	–3.5 V to +7.1 V
0	–6.0 V to –1.5 V	–6.0 V to +8.0 V	–6.0 V to +5.1 V

[1]VL: Output level Low

[2]VH: Output level High

[3]CVL: Comparator level Low

[4]CVH: Comparator level High

If required, all outputs (OUTx) can be connected to their corresponding input (INx) and therefore stimulate, measure and monitor the driving of signals. All driver channels can be set to the high-impedance state (TRI-STATE).

Timing control of data output and reading in of response signals takes place on the module controlled by FPGAs.

7.1.2 Ports

Due to the structures in the hardware, the digital inputs and outputs are divided into so-called ports.

The R&S TS-PIO4 module provides 32 outputs (OUT1 to OUT32) and 32 inputs (IN1 to IN32). Each port (PORT0 to PORT7) is made up of 4 channels.

Many functions of the driver refer to this port structure.

Table 7-2: Port structure

Port	Channel OUTx / INx	Bit mask
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

7.1.3 Memories

7.1.3.1 Memory management in driver

If required, the digital channels can be configured in different combinations for simultaneous static and dynamic operation. Various formats (data types) are therefore available for the dynamic data records.

Stimulus data records which are to be loaded to the module using `rspio4_LoadData` are first stored in the memory management. For each data record, the driver assigns an ID using which this data record can then, if required, be identified and executed.

If access takes place using the driver functions compliant with IVI Digital, this runs in the background and can be ignored by the user. The appropriate data formats are then also selected automatically.

The data is interpreted in the driver according to the mode to which the module was set beforehand by means of the `rspio4_ConfigureStimMode()` and `rspio4_ConfigureRespMode()` functions.

For reasons of backward compatibility, the attribute `RSPIO4_ATTR_PORT_HANDLING` can be set to the value *RSPIO4_PORT_HANDLING_PDFT* using the `rspio4_ConfigurePortHandling` function. In this case, the driver software emulates a module with 4 ports each with 8 channels. With this setting, the tri-state information in the data structures is interpreted on a port-specific basis.

The file `rspio4.h` provides these data types. For details, see `rspio4.h` as well as the help file of the driver.

Table 7-3: Data types

Mode **(see `rspio4.h`)**	**Data type** **(see `rspio4.h`)**
RSPIO4_VAL_CTRL_STATIC	
RSPIO4_VAL_CTRL_4BIT	RSPIO4_DATA_4BIT (stim. and resp.)
RSPIO4_VAL_CTRL_8BIT	RSPIO4_DATA_8BIT (stim. and resp.)
RSPIO4_VAL_CTRL_16BIT	RSPIO4_DATA_16BIT (stim. and resp.)
RSPIO4_VAL_CTRL_32BIT	RSPIO4_DATA_32BIT (stim. and 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.)

7.1.3.2 Stimulus memory

2 Msample memories for stimulus data are available in the module. This means that max. one pattern set with 2 x 1024 x 1024 data vectors plus 2 x 1024 x 1024 enable vectors can be defined and loaded to the module.

The data records are first transferred to the stimulus memory using the `spio4_LoadData()` function. This function returns an ID. This ID can then be used to execute the data record in the stimulus memory.

Depending on the function used to execute a pattern set, this takes place automatically or must be triggered manually (e.g. IVI Digital functions perform this sequence completely in the background).

The `rspio4_DiscardData()` function is used to remove the data record from the stimulus memory.

7.1.3.3 Response memory

The recorded response data is also stored in a 2 Msample memory. This memory always contains the results of the high comparators and the low comparators alternately in a 32-bit value.

The default functions evaluate these results according to the selected comparator mode:

- COMP window comparator
- HYST hysteresis

To allow information with regard to an input voltage in a prohibited range to be received, functions which evaluate and return the validity of the data of each channel have been added to the driver. (See also Chapter 7.3.3, "Configuration of input channels", on page 34.)

7.1.4 Static digital test

With the static digital test, the sequence for applying patterns at the outputs as well as for reading in the inputs is checked from the control computer. As a result, the duration of the individual patterns as well as the sampling time point for the measurement relative to the beginning of the pattern can never be precisely predicted because the host computer and its operating system are influencing factors here.

The static digital test is therefore suitable for checking the logical interaction of components in order to check voltage thresholds and other sequences in cases where the verification and the precise observance of time conditions are not relevant.

With R&S TS-PIO4, the static digital test can be performed using IVI Digital functions or the low-level driver functions.

7.1.5 Dynamic digital test

In the simplest case, the pattern period is identical for output (stimulus) and recording (response). Stimulus and response are started by the same trigger and run synchronously to each other.

Normally, the UUT response must be measured with a delay relative to the beginning of the pattern. The delay (Response Delay) can be set between 0 s and the pattern period. This delay is defined by the application such that stable data from the UUT is present at the time of sampling. This value is ultimately determined by the delay times in the UUT. If the delay is greater than the pattern period, then sampling already takes place within the next period of the stimulus patterns. This too may be advisable in certain applications.

The test is started by software or hardware triggers. The patterns are output at a fixed clock rate. The pattern duration is the time during which a pattern of a pattern set is available at the outputs. The responses from the UUT are usually also recorded at the inputs during this time.

The "Response Delay" is the time offset between the beginning of a pattern and sampling of the data at the inputs. The pattern rate is the reciprocal of the pattern period.

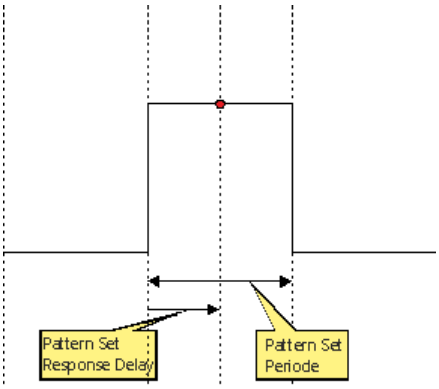


Figure 7-1: "Pattern Set Period" and "Response Delay"

A pattern set is a quantity of patterns (vectors) that are processed in a sequence after a trigger event has been received. Processing is started by a software function or a hardware trigger. Execution takes place in real time and is not affected by the operating system of the control computer. The software is able to query whether execution is still in progress, to interrupt the current execution or to wait until execution has finished.

7.2 Programming of digital tests

TSVP is an open platform with regard to hardware and software. Typically, the software consists of a number of drivers and libraries that are called from a test program or sequencer created by the user.

Generally there are two ways to access TSVP modules.

- Instrument driver
- High-level GTSL libraries

Device driver functions provide access to all hardware options. In contrast, symbolic names and cross-module functionality are not supported.

The functions in the device driver provide two interfaces for programming digital tests:

- Digital test with low-level functions
- Digital test with IVI Digital functions

High-level libraries provide a standardized programming interface and allow certain abstractions with respect to the underlying hardware, e.g. symbolic signal names and the handling of multiple parallel modules.

The GTSL library "DIO Manager" (`DIOMGR.DLL`) is provided for operation of the digital functions of the module.

7.2.1 Digital tests with device driver functions

7.2.1.1 Initialization

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

7.2.1.2 Auxiliary functions

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

7.2.1.3 Error queries

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

7.2.1.4 Functions of IVI Switch component

On the R&S TS-PIO4, there are only the two channels "GND" and "GNDNO" that can be switched using the "IVI switch functions". The GND relay is used for this purpose.

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

7.2.1.5 Digital test with Low-Level driver functions

For optimum utilization of the board's performance, it is recommended to use the digital test with low-level functions.

Static digital test with low-level driver functions

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

Dynamic digital test with low-level driver functions

Some of these functions are also required for configuring the default settings for dynamic tests compliant with IVI Digital (see Chapter 8.3.4, "Static pattern output with Low-Level driver functions", on page 68).

- `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`

7.2.1.6 Digital test compliant with IVI digital

IVI Digital is a standard for the actuation of digital test instruments. The driver provides calls that are compliant with IVI Digital. IVI Digital supports both the static and the dynamic digital test.

Use of the IVI Digital functions requires an extremely high computing overhead on the host side and is relatively inflexible with regard to the hardware because IVI Digital is a standardization and cannot take performance-boosting features of the hardware into consideration. For example, splitting into static and dynamic channels is not possible.

Functions for digital test compliant with 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

Static digital test compliant with IVI Digital

Here too static tests are possible, i.e. tests in which the host computer is responsible for control and which, as a result, do not have a fully defined time response. See the example in Chapter 8.3.3, "Static pattern execution with IVI Digital", on page 61.

Dynamic digital test compliant with IVI Digital

As shown in the example for the static digital test compliant with IVI Digital, a so-called op code is generated for each channel ("OUT1" to "OUT32", "IN1" to "IN32"). This op code controls whether a channel is to drive high or low or whether it is to switch to tri-state. The pattern generated in this way is then executed immediately and is available at the outputs.

In the case of the dynamic digital test compliant with IVI Digital, the generated patterns are saved in a pattern set.

After a timing has been defined, the complete pattern set is executed.

7.2.2 Digital test with DIO manager

Each pattern set is stored in its own file. This file can be edited manually using a test editor.

The file containing the pattern set is loaded during runtime to one or more R&S TS-PIO4 modules and then executed. The results can be read back using function calls. Alternatively, the results can also be stored in a file with the same format. Differences between the expected and measured behavior can then be easily identified by comparing the files.

See also the example in Chapter 8.3.1, "Digital test with "DIO manager" library", on page 47.

7.2.2.1 File format

The file format was originally defined by the Altera Quartus Waveform Generator.

```
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];
```

This instruction is optional. It is used to group pins into buses. It is also possible to define multiple buses.

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

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

The INPUTS and OUTPUTS instructions define the channel names or group names. Here it is important that INPUTS are inputs of the UUT and therefore correspond to OUT1 to OUT32 of the R&S TS-PIO4 module. OUTPUTs are UUT outputs and therefore correspond to the inputs IN1 to IN32 of the module.

```
UNIT ns;
```

Defines the unit of the timestamp used in the file.

```
RADIX HEX;
```

Defines the base for the values in the case of buses. The DIO manager supports HEX only.

```
PATTERN
```

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

.....

All lines following PATTERN define this pattern set. Each line represents a pattern at a certain time. The last line is ignored and is used only as end identification. All channels are X. The timestamps do not have to be equidistant.

Value	for INPUTS	for OUTPUTS
0	drive low	expect low
1	drive high	expect high
Z	high impedance	(not used)
X	(not used)	don't care

A hexadecimal value is entered for groups. The special cases X and Z indicate that all channels of the groups concerned assume this state.

7.2.2.2 Configuration of DIO manager

The waveform file contains the logical name of the inputs and outputs. Assignment of physical channels on the R&S TS-PIO4 modules to logical names takes place in the file `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
```

Like all GTSL libraries, the DIO manager is configured in an `application.ini` file. The keyword `DIODevice` refers to the R&S TS-PIO4 module in the `physical.ini`. If more than one digital module is to be used, a DIODevice2, DIODevice3, and so on is added.

The "DIO channel table" refers to the associated table in the file.

The left side of the channel table contains the logical names and the right side the link to the physical channels on the individual modules.

Channel names for groups (e.g. buses) do not have any brackets. Here they are called "bus0" for example, whereas they are called "bus[0]" in the waveform file.

Channel names are not case-sensitive.

7.2.2.3 Structure of a test program

Table 7-4: These functions must be called once at beginning of test program.

Function	Comments
`RESMGR_Setup`	Call of the resource manager
`DIOMGR_Setup`	Call of the DIO manager and initialization of the TS-PDFT module
`DIOMGR_ConfigureStimulus` `DIOMGR_ConfigureResponse`	Configuration of logical levels for inputs and outputs
`DIOMGR_LoadWaveform`	Loading a pattern set from a file to the memory.
`DIOMGR_ConfigurePatternSetTiming`	Definition of the timing of a pattern set

Table 7-5: These functions are typically called in a loop over multiple UUTs.

Function	Comments
`DIOMGR_ExecutePatternSet`	Execution of a pattern set and return of the result (pass/fail)
Optional functions:	Generally only called if the pattern set is defective (fail)
`DIOMGR_SaveWaveform`	Storage of the results as a TBL file
`DIOMGR_GetPatternSetExecutedPatternCount`	Reading out of the executed patterns
`DIOMGR_GetPatternSetFailedPatternCount`	Reading out of the failed patterns
`DIOMGR_GetPatternSetFailedChannelCount`	Reading out of the failed channels
`DIOMGR_GetPatternSetFailedChannelNames`	Reading out of failed channels as a list of names (comma-separated)
`DIOMGR_GetPatternSetChannelData`	Reading out of the current response message (measured data) for a channel
`DIOMGR_GetPatternSetChannelResults`	Reading out of the results for a channel (pass/fail)

Table 7-6: After all tests have been performed, these functions are used for cleaning up.

Function	Comments
`DIOMGR_Cleanup`	Closes the DIO manager
`RESMGR_Cleanup`	Closes the resource manager

7.2.2.4 Ports

The R&S TS-PIO4 module is divided into 8 ports each with 4 channels. Each port can be set to different driver and sensor levels. If different levels are used in a program, `DIOMGR_ConfigureStimulus()` and `DIOMGR_ConfigureResponse()` must be called once for each "logic family" and the required level must be set.

7.2.2.5 Loading of waveform file

The waveform file defines a pattern set with specification of the timestamps. These timestamps do not have to be equidistant. Normally, a new line is created whenever at least one signal changes.

```
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.
```

When the waveform is loaded to the R&S TS-PIO4 module, the timing must be converted to a fixed framework which is determined by the pattern set period of the module. The loading function uses the "timeGrid" parameter to generate the samples of the waveform in a fixed time grid. Each sample is then converted into a pattern and stored in the memory of the module.

If a "timeGrid" of 500 ns is set in the example above, this would result in the following samples:

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

Some patterns are repeated (1 and 2, 4 and 5) and some are not recognized as they are too short and are between the sampling time points (e.g. patterns at the timestamp 1001.0).

The timestamp and the "timeGrid" parameter do not necessarily have to represent the actual execution speed. The actual execution speed is set using the `rspio4_ConfigurePatternSetTiming()` function. This also means that the waveform file does not need to be modified if the test is to run at a different pattern rate.

The following points should be observed when writing the waveform file:

- Use equidistant timestamps
- Timestamps are to reflect the real timing, i.e. exactly the timing that is used for the test.
- Use the interval between the timestamps as the value for the "timeGrid" parameter.
- Use the interval between the timestamps as the "Pattern Set Period".

7.2.2.6 Execution of pattern set

The pattern set can be executed synchronously or asynchronously. In the first case, the `DIOMGR_ExecutePatternSet()` function does not return until execution has been completed (or the maximum time has been exceeded). In the second case, execution is started or the trigger armed and the function returns without having to wait for actual execution of the pattern set. The `DIOMGR_WaitUntilPatternSetComplete()` function can be used to wait for the end of execution.

7.3 Configuration of digital channels

The module offers a series of configurable parameters. This section describes the configuration options and lists the corresponding driver functions. For details on the driver functions, refer to the GTSL help of the driver software.

7.3.1 Setting of voltage range

The application must set a range that contains the desired voltages for the high and low levels of the outputs as well as the required comparator thresholds for the inputs. Each port can be assigned its own voltage range.

Switchover of the voltage ranges causes long settling times on the module which have to be taken into consideration in the device driver. When the voltage range is adjusted, all drivers are switched to high impedance.

The voltage ranges are listed in the Voltage ranges table.

- `rspio4_ConfigurePortVoltageRange`

7.3.2 Configuration of stimulus channels

Settings for the output voltages can be programmed for each port.

- `rspio4_ConfigureStimPort`

The settings are made for one or more ports depending on the parameters in the function call. The current limit always applies to the total output currents from all 4 drivers of a port.

When current is flowing, a voltage drop that may have to be taken into consideration during level adjustment is caused as a result of the output impedance of the driver and the resistors in the output safety circuits (together approx. 30 Ω).

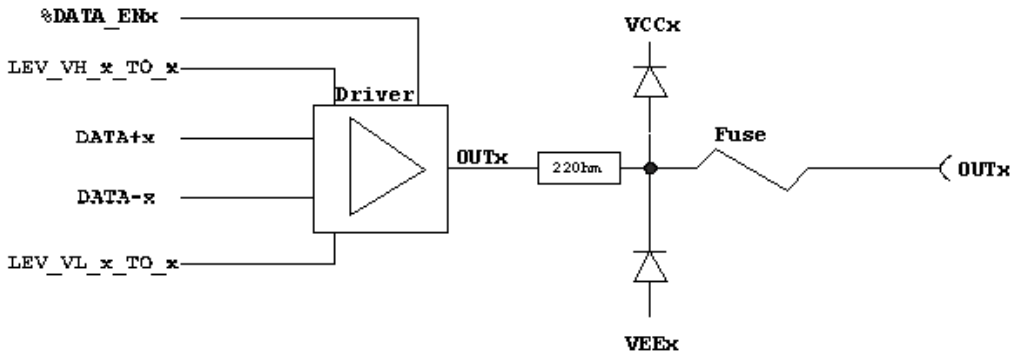


Figure 7-2: Configuration of stimulus channels

7.3.3 Configuration of input channels

- `rspio4_ConfigureRespPort`

The measurement channel inputs have a safety circuit upstream of the comparators. This safety circuit has the following structure:

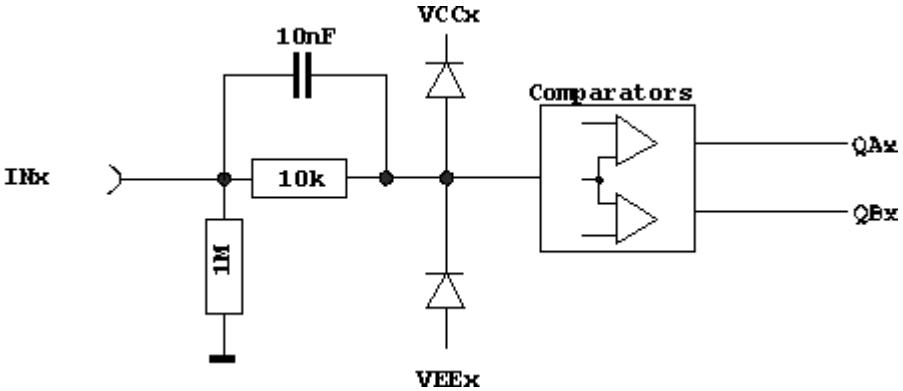


Figure 7-3: Configuration of input channels

Each input is routed to two comparators which have an adjustable threshold. This makes it possible to implement a hysteresis for evaluation of the signals. The thresholds can be set using the driver function `rspio4_ConfigureRespPort`. This allows individual values to be set for each port.

The result of the signal evaluation of a channel is **1** if the input level is greater than the threshold for the high level.

The result of the signal evaluation of a channel is **0** if the input level is less than the threshold for the low level.

If the input level is between the thresholds, "Forbidden Zone" is returned as the result. The states of the two comparators can be read out using the low-level driver function `rspio4_GetDinHighAndLowComp`.

If the comparator is set to "COMP" mode, the two comparators function as window comparators. A **1** is detected if the input level is between the thresholds, and a **0** is detected if the input level is below the threshold for low or above the threshold for high.

7.3.4 Time settings for data output

This setting is required both for dynamic tests in compliance with IVI Digital and for dynamic tests with the low-level driver functions.

To configure the time response of the stimulus channels during output of a pattern set, the `rspio4_ConfigureStimTiming` function is used to set the trigger delay, the pattern duration and the number of patterns to be output (Trigger Delay, Pattern Period, Loop Count). The trigger delay is the wait time between the trigger event and output of the first pattern. The pattern duration is the time during which an individual pattern is present.

7.3.5 Time settings for data recording

This setting is required both for dynamic tests in compliance with IVI Digital and for dynamic tests with the low-level driver functions.

The time parameters required for recording the input signals can be set using the `rspio4_ConfigureRespTiming` function. As with data output, the trigger delay, the pattern duration as well as the number of patterns to be read in (Response Trigger Delay, Response Pattern Period, Loop Count) are available as parameters.

The trigger delay determines the wait time between the trigger event and initial sampling. The response trigger delay is set such that the UUT data is available at the input in a stable condition at this point in time. In the typical case where the Stimulus Pattern Period and Response Pattern Period are identical, a delay of 0.0 meant that in every pattern sampling would take place directly at the beginning of the pattern; a delay greater than the pattern period would mean sampling within the next pattern or within a different pattern.

Example:

Stimulus pattern period: 50 ns

Response pattern period: 50 ns

Stim. trigger delay: 0.0 ns

Resp. trigger delay: 25.0 ns

The outputs are changed at time point 0. The data is sampled 25.0 ns later, i.e. at the middle of the pattern in our example. Here, it is therefore assumed that the UUT outputs stable data after less than 25.0 ns.

7.3.6 Configuration of data width in dynamic mode

- `rspio4_ConfigureStimMode`

- `rspio4_ConfigureRespMode`

If required, the digital channels can be configured in different combinations for simultaneous static and dynamic operation. This, for example, allows the configuration of control signals that remain at a constant level during entire test sequences. They then no longer have to be part of the dynamic data. This also simplifies the creation of pattern sets.

This type of configuration is only possible if programming is performed using the low-level driver functions. If the driver functions compliant with IVI Digital are used, the dynamic width is always 32 bits.

The possible or selected constellations must, however, be taken into consideration during fixture wiring.

The following configurations are possible (see also `rspio4.h`).

Mode	Configuration
`RSPIO4_VAL_CTRL_STATIC`	Static mode
`RSPIO4_VAL_CTRL_4BIT`	4 dynamic DOUT or DIN: OUT 1 to 4 / IN 1 to 4 No tri-state
`RSPIO4_VAL_CTRL_8BIT`	8 dynamic DOUT or DIN: OUT 1 to 8 / IN 1 to 8 No tri-state
`RSPIO4_VAL_CTRL_16BIT`	16 dynamic DOUT or DIN: OUT 1 to 16 / IN 1 to 16 No tri-state
`RSPIO4_VAL_CTRL_32BIT`	32 dynamic DOUT or DIN: OUT 1 to 32 / IN 1 to 32 No tri-state
`RSPIO4_VAL_CTRL_4BIT_TRISTATE`	4 dynamic DOUT or DIN: OUT 1 to 4 / IN 1 to 4 Tri-state information in data
`RSPIO4_VAL_CTRL_8BIT_TRISTATE`	8 dynamic DOUT or DIN: OUT 1 to 8 / IN 1 to 8 Tri-state information in data
`RSPIO4_VAL_CTRL_16BIT_TRISTATE`	16 dynamic DOUT or DIN: OUT 1 to 16 / IN 1 to 16 Tri-state information in data
`RSPIO4_VAL_CTRL_32BIT_TRISTATE`	32 dynamic DOUT or DIN: OUT 1 to 32 / IN 1 to 32 Tri-state information in data

The `RSPIO4_VAL_CTRL_32BIT_TRISTATE` mode is always used in IVI-Digital-compliant operation.

7.3.7 Power consumption

7.3.7.1 Estimation of power consumption

The total power consumption of the R&S TS-PIO module should not exceed 40 W and depends on the operating mode of the input and output channels.

The following description explains calculation of the maximum power consumption for the operating ranges most frequently used.

VH = Output level High

VL = Output level Low

Range 3 with the settings VL = +1.5 V to +6.0 V and VH = +1.5 V to +10.0 V

Range 2 with the settings VL = –1.8 V to +2.7 V and VH = –1.8 V to +10.0 V

Range 1 with the settings VL = –3.5 V to +1.0 V and VH = –3.5 V to +10.0 V

Range 0 with the settings VL = –6.0 V to –1.5 V and VH = –6.0 V to +8.0 V

Output channels

Table 7-7: Maximum power consumption per group (consisting of 4 channels)

Range	Power consumption in a group [W]	Maximum power in a group with four channels
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 : Number of active channels in a group with four channels

f : IO switchover frequency (= 1/2 sample rate), duty cycle 50%, MHz

U_swing : Voltage variation of the IO signal (VH-VL), V

Example:

Four active channels in a group with a switchover frequency of 20 MHz (corresponds to a sample rate of 40 Msamples), VL = +1.5 V and VH = +4.5 V.

- U_swing = VH – VL = 4.5 V – 1.5 V = **3.0 V**
- The values for VL and VH can be used in the ranges 2 and 3

Calculation of the power consumption in a group:

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

Range 3 must be used on account of the limited power consumption of the group.

Table 7-8: Maximum power consumption per module

Range	Power consumption of active IO channels [W]	Maximum power consumption of a module [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 : Number of active groups (a group consists of max. four channels), max. 8 groups

#CH : Number of active channels in a group with four channels

f : IO switchover frequency (= 1/2 sample rate), duty cycle 50%, MHz

U_swing : Voltage variation of the IO signal (VH-VL), V

Example:

Four active channels and 8 active groups with a switchover frequency of 20 MHz (corresponds to a sample rate of 40 Msamples), VL = +1.5 V and VH = +4.5 V.

- U_swing = VH – VL = 4.5 V – 1.5 V = **3.0 V**
- The values for VL and VH can be used in the ranges 2 and 3

Calculation of the power consumption for one module:

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

In this case, only range 3 can be used.

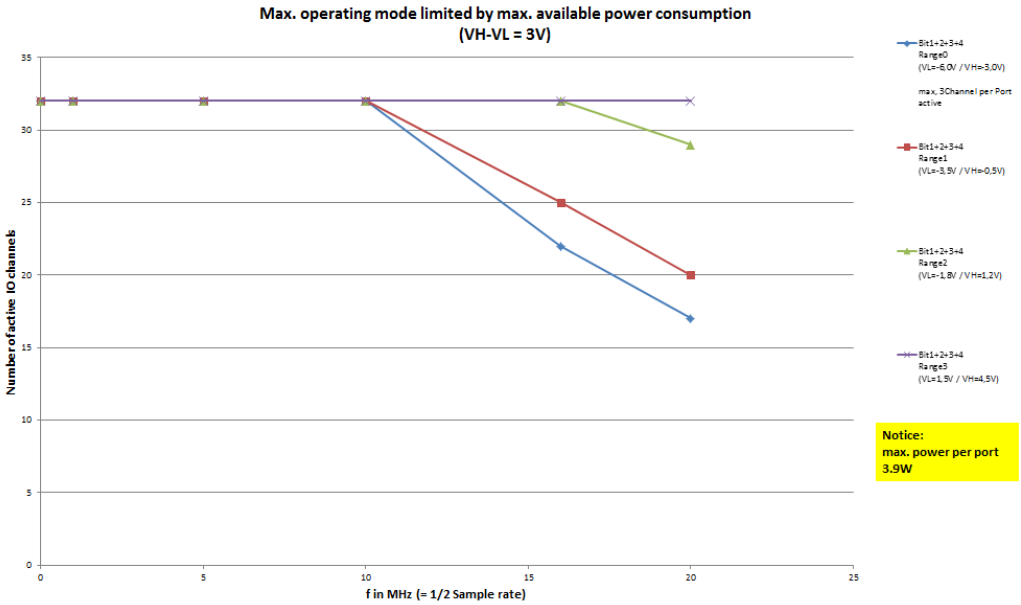


Figure 7-4: Max. operating range as a function of maximum power consumption

Input channels

Depending on the operating range and frequency, receive channels (comparators) are additionally subject to dissipation loss during switchover.

The table below gives the approximate additional power consumption for the operating ranges with the following output values:

- Max. switchover frequency: 20 MHz
- Voltage variation: 3.3 V
- Number of channels: 32 input channels

Range	Additional power consumption (typ.)
3	10.0 W
2	5.5 W
1	12.5 W
0	13.5 W

Total power consumption

The input and output channels must be taken into consideration when estimating the total power consumption.

7.3.7.2 Safety mechanism

If the maximum permissible power consumption is exceeded, the IOs are deactivated by a safety circuit. In this case, an error will occur when a driver function is called. The IOs can be reactivated with the `rspio4_reset` function.

7.4 Triggering and sequence control

7.4.1 Trigger units

There are two trigger units on the module. The trigger units control output of stimulus data as well as reading in of the data at the digital inputs. Trigger unit IT1 is responsible for outputs of the stimulus data, and trigger unit IT2 is responsible for reading in of the response data.

The trigger units can be configured for various input conditions. It is also possible to configure which output signal they are to generate and where this signal is to be routed to.

For the configuration of trigger units, see

- Figure B-4
- Figure B-5
- Figure B-6
- Figure B-7
- Figure B-8

Inputs of trigger unit:

On the input side, the `rspio4_ConfigHWTriggerInput` function is used to configure whether and which inputs (PXI0 to PXI7, XTI, 4 outputs of the digital inputs pattern comparator) are to be used as a hardware trigger and with which logical state, and which edge is used.

The `rspdft_ConfigDinComparator` function allows setup of the digital inputs pattern comparator.

Alternatively, the unit can also be triggered using a software trigger (e.g. the `rspio4_executePattern` function starts the processing of a pattern set by means of a software trigger by calling the `rspio4_InitiateSWTrigger` function).

If the trigger unit was started by an event, then as many stimulus and response patterns are processed as were set using the configuration functions `rspio4_ConfigureStimTiming` and `rspio4_ConfigureRespTiming` in the `Loop Count` parameters.

Outputs of trigger unit:

The output of the trigger unit is used to actuate other functions. Selection of a certain type of output signal does not influence the internal function (output stimulus, read in response).

This output signal (e.g. ACTIVE) can be output on the PXI0 to 7 lines as well as on XTO in order to, for example, trigger a measurement on a different CompactPCI board.

7.4.2 Receiving of trigger signals

Various trigger sources are available for starting the controllers:

Designation	Remarks
SW Trigger	Sequence control starts immediately after arming (when the `rspio4_InitiateSWTrigger` function is called). This is the default sequence if a pattern set is to be processed under program control. For configuration, see also the example for the dynamic test with low-level functions.
"XTI"	TTL input on the front connector X10; a positive or negative edge starts the sequence. Triggering via XTI is configured with the `rspio4_ConfigHWTriggerInput` function. It is used to define the edge and to activate/deactivate triggering via XTI.
PXI0 to PXI7	Positive or negative edges on these lines start the sequence. The active edge and which trigger signal is used are again set using the `rspio4_ConfigHWTriggerInput` function. If multiple trigger sources are selected, then this is equivalent to an OR logic operation.
Pattern Comparator	A pattern comparator which evaluates the states of the 32 inputs (IN) can also be used as the trigger source. This is again also configured using `rspio4_ConfigHWTriggerInput` as well as `rspio4_ConfigDinComparator` in order to define the condition for the input pattern comparator.

The `rspio4_ConfigHWTriggerInput` function determines the trigger source. Afterwards the hardware trigger sources are armed using the `rspio4_EnableHWTrigger` function and the addressed sequence controls are in the "Waiting" state. If a software trigger has been triggered, the associated control immediately switches to the "Running" state. Otherwise the state change does not take place until the trigger event has occurred. In this state, the number of patterns in the stimulus memory (Stim Loop Count) is then output and the number of patterns (Resp Loop Count) in the reference memory is recorded.

If only patterns are to be output, the `rspio4_InitiateSWTrigger` function has to be called with the mask RSPIO4_IT_MASK_IT1 only. If only recording is required, the function must be called using only RSPIO4_IT_MASK_IT2.

After the available number of patterns has been processed, the associated sequence control switches back to the "Stopped" state.

The current state of both sequence controls can be queried using the `rspio4_GetItStatus` function. By calling the `rspio4_WaitUntilPatternSetComplete` function, the sequence control can be made to wait until the end of execution in the test program.

If sequence control is in the "Waiting" or "Running" state, some driver functions cannot be performed. In this case, these functions return an error message. If necessary, the `rspio4_AbortExecution` function can be used to switch sequence control to the default state.

7.4.3 Generation of trigger signals

The R&S TS-PIO4 module is able to generate trigger signals on the following lines:

Designation	Remarks
XTO	TTL output on the front connector
PXI0 to PXI7	PXI trigger lines on the backplane

For a change to occur on the trigger lines, an event which provokes the trigger pulse must be assigned to the selected line.

The following signals can be routed to the output using the `rspio4_ConfigXTO` function:

- Output of pattern comparator
- Output of IT1
- Output of IT2

The following signals can be switched to a selected PXI using the `rspio4_ConfigPxiTrigOut` function:

- Output of pattern comparator
- Output of IT1
- Output of IT2
- Output of general purpose trigger generator

This function can also be used to invert the selected signal and to activate the output driver of the PXI trigger line.

The `rspio4_InitiateSWTrigger` function is used to start the following function blocks under software control:

- Trigger logic block IT1 (sequence control for data output)
- Trigger logic block IT2 (sequence control for data recording)
- Sample trigger stimulus (output of a single pattern from the stimulus memory)
- Sample trigger response (recording of a single pattern to the response memory)
- General purpose trigger generator (generation of a single trigger pulse)

The `rspio4_ConfigItTrigOut` function can be used to determine which signal the trigger logic blocks IT1 and IT2 are to output after they have been started either via software (`rspio4_InitiateSWTrigger`) or by means of a configured hardware signal (`rspio4_ConfigHWTriggerInput`). The table below shows the options:

Designation	Remarks
ITn ACTIVE	Pulse while the pattern set is being output.
ITn OUT	Trigger block outputs a pulse for each pattern. The pulses are relatively short (minimum pattern rate/2).
ITn STATUS	Signals the internal state of the trigger block. The signal becomes active when the trigger event is received. The signal does not become inactive again until the trigger block is reset.
ITn START	A pulse after the trigger condition has been detected.

See also the example in Chapter 8.3.6, "Triggered pattern execution", on page 76.

7.5 PWM

The module supports the output of PWM signals at any output (OUTx).

PWM is configured using the `rspio4_ConfigurePWM` function. The frequency is set in Hz and the duty cycle is specified in %.

The `rspio4_SetStatePWM` function is used to switch PWM on/off for individual channels.

The time resolution is 12.5 ns if the internal reference clock of 80 MHz is used. Lower frequencies are generated using a divider. With this method, there are certain limitations with regard to the choice of frequency and duty cycle.

The following examples show the correlations at the highest frequencies:

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

7.6 Frequency measurement

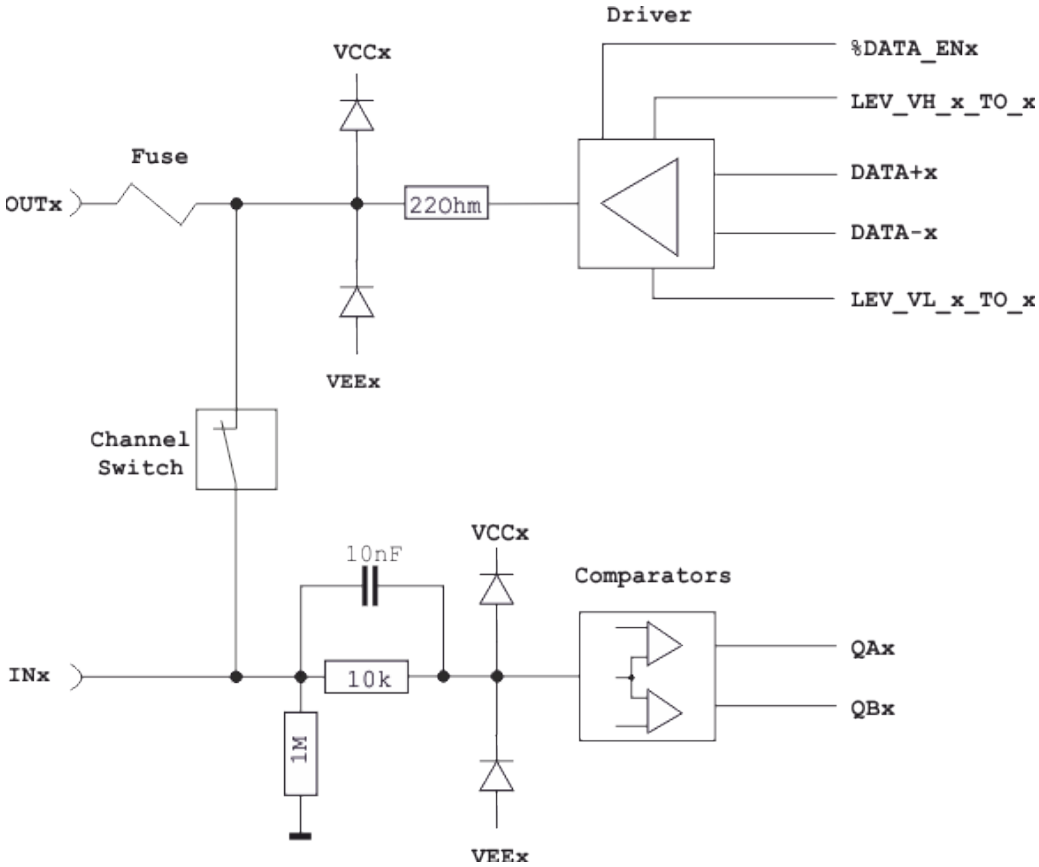
The R&S TS-PIO4 module supports frequency measurement at inputs IN1 to IN32. The `rspio4_FrequencyMeasurement` function is used for this purpose. A distinction is made between two methods, depending on the passed parameters:

- Specification of a gate time: Pulses are counted until the gate time has expired. This allows the frequency value to be calculated. The "gateCount" parameter is set to 0.

- If a defined number of pulses (gateCounts) is specified, the module measures the time required for a certain number of pulses to arrive. The Gate Time parameter is considered a timeout value. If the desired number of pulses is not included into this time, the `rspio4_FrequencyMeasurement` function will return an error message.

7.7 Bidirectional channels

Each input can be individually connected to the associated output by means of an analog switch. This allows reading out of the outputs and reading in of a UUT response with the outputs switched off (tri-state). The connection is set up using the `rspio4_ConnectInOut` function.



This arrangement also allows to monitor the output current. A current via the 22 Ohm safety resistor creates a potential drop. Also, the output resistor of approximately 7 Ohm of the driver needs to be considered. When current is flowing the voltage of the comparator is lower than the set voltage. With appropriate programming of the input comparator the current can be monitored. Because the internal resistance (22 Ohm and 7 Ohm), as well as the input comparators are uncertainties, a reference measurement with a voltage meter (e.g. TS-PSAM) is useful to detect the comparators thresholds.

7.8 External clock input

Various applications can be implemented using the external clock input (EXT_CLK). One possibility is to generate output frequencies that cannot be achieved using an internal frequency divider; alternatively, the measuring card can be synchronized to the clock of a UUT. The minimum permissible input frequency is 20 MHz and the maximum is 40 MHz. All lower frequencies can be generated using the internal divider.

The clock signal source (internal PXI 10 MHz clock or supplied via EXT_CLK) can be selected using the `rspio4_ConfigureClock` function. If an external clock is selected, the clock frequency must be specified using the parameter `ExternClockFreq`. This parameter is ignored if the internal clock is selected. The state of the clock PLL can be queried using the `rspio4_GetPllLockedStatus` function. This function indicates whether the clock can be used successfully.

The changed clock changes the timebase of the card. This increases, for example, the step size in the case of adjustable trigger delays.

7.9 Synchronization of multiple modules

Multiple modules can output and record data in a synchronized manner via the various trigger inputs and outputs. The internal PXI trigger lines are not length-compensated which means that significant delay differences can occur. The highest level of accuracy can be achieved by means of length-compensated lines between the master's XTO and the XTIs.

7.10 AUX channels

Each of the AUX1, AUX2, AUX3 and AUX4 signals are routed from the rear X20 connector directly to the front X10 connector. The current-carrying capacity is 1 A in each case.

7.11 GND relay

The GND relay connects the GNDNO signals to GND at the front X10 connector. This means that a UUT can optionally be connected to GND. With in-circuit tests, the UUT must be floating, whereas with functional tests it is often recommended to have a connection between UUT ground and system ground.

In the default state, this connection is open.

8 Software

8.1 Driver software

A LabWindows IVI driver is available for actuation of the R&S TS-PIO4 digital functional test module. All additional functions of the hardware are controlled using specific extensions of the driver. The driver is part of the ROHDE & SCHWARZ GTSL software. All the functions of the driver are described fully in the online help and in the LabWindows/CVI function panels. The following software modules are installed during driver installation:

Module	Path	Remarks
`rspio4.dll`	`<GTSL directory>\Bin`	Driver
`rspio4.chm`	`<GTSL directory>\Bin`	Help file
`rspio4.fp`	`<GTSL directory>\Bin`	LabWindows/CVI function panel file, function panels for CVI development environment
`rspio4.sub`	`<GTSL directory>\Bin`	LabWindows/CVI attribute file. This file is required by some "function panels".
`rspio4.lib`	`<GTSL directory>\Bin`	Import library
`rspio4.h`	`<GTSL directory>\Include`	Header file for driver

The IVI and VISA libraries from National Instruments are needed to run the driver.

8.2 Soft panel

The software package of the R&S TS-PIO4 module includes a soft panel (see Figure 8-1). The soft panel is based on the IVI driver and enables interactive operation of the module.

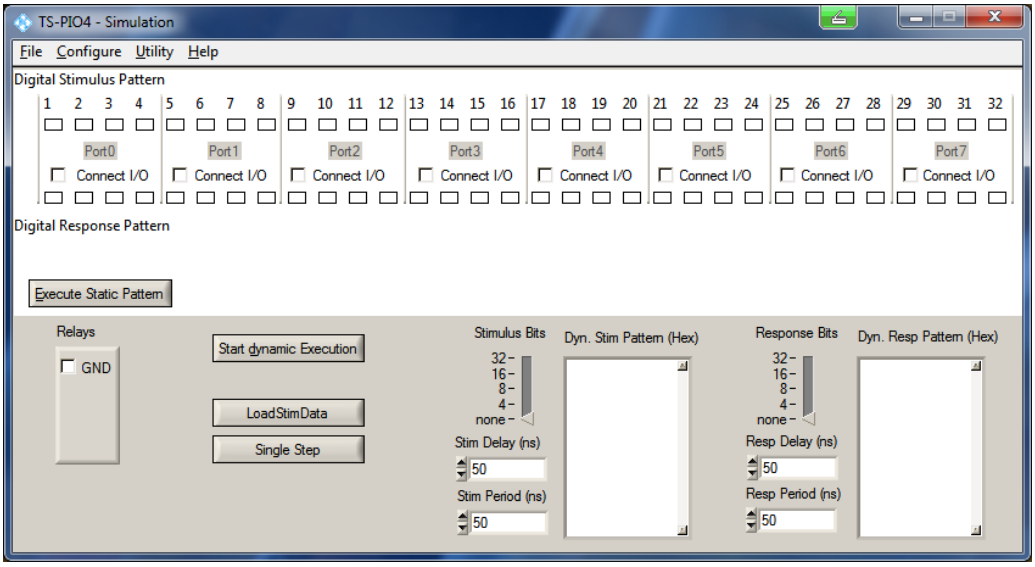


Figure 8-1: Soft panel of R&S TS-PIO4

Operation of the soft panels is described in *Software Description for R&S GTSL*.

In the R&S TS-PIO4 soft panel, it is possible to set static and dynamic bit patterns at the outputs and to read in input states.

The voltage ranges, the high and low driver levels as well as threshold voltages of the input comparators can be set in subsequent dialogs which are opened from the Configure menu.

Furthermore, it is also possible to configure the stimulus and response trigger units as well as to provoke trigger events and to make PWM settings.

8.3 Programming examples for R&S TS-PIO4

8.3.1 Digital test with "DIO manager" library

This example shows the execution of a digital test using the GTSL library "DIO Manager" (DIOMGR). The input file (`823916.tbl`) and the associated configuration file (`pio4Application.ini`) are described in Chapter 7.2.2, "Digital test with DIO manager", on page 28. Additional information on the structure of the configuration files (physical layer configuration file and application layer configuration file) and on the function of the resource manager (RESMGR) can be found in the manual *Software Description R&S GTSL*.

8.3.1.1 Main function

```
/* 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;
}
```

8.3.1.2 Error handling

This function checks the return values of a function from the GTSL libraries (RESMGR, DIOMGR). A message is issued if there is an error.

```
/* 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);
  }
}
```

8.3.1.3 Execution of digital test

```
/* 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");
}
```

8.3.1.4 Evaluation of failed patterns

```
/* 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 );
}
```

8.3.2 Dynamic pattern execution with IVI Digital

8.3.2.1 Main function

The return value of a device driver call is stored in the module-global variable `sta`. The status code is checked using the `chk()` function.

```
/* 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;
}
```

8.3.2.2 Error handling

This function checks the return values of a driver call. An error message is issued if there is an error.

```
/* 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);
  }
}
```

8.3.2.3 Execution of digital test

```
/* 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");
}
```

8.3.2.4 Generation of pattern set

```
/* 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");
```

```
}
```

8.3.2.5 Evaluation of failed patterns

```
/* 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);
}
```

8.3.3 Static pattern execution with IVI Digital

8.3.3.1 Main function

```
/* 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;
}
```

8.3.3.2 Error handling

This function checks the return values of a driver call. An error message is issued if there is an error.

```
/* 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);
  }
}
```

8.3.3.3 Execution of digital test

```
/* 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();
}
```

8.3.3.4 Execution of a pattern set

```
/* 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");
}
```

8.3.3.5 Execution of a single pattern

```
/* 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);
  }
}
```

8.3.3.6 Evaluation of a failed pattern

```
/* 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");
  }
}
```

8.3.4 Static pattern output with Low-Level driver functions

8.3.4.1 Main function

```
/* 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[] = "PXI7::12::INSTR";

static ViUInt32 stimData[PATTERN_COUNT] = {
  0x00000000,
  0x00000010,
  0x0000001F, /* OUT1 to OUT4 tri state */
  0x00000000,
  0x00000010,
  0x0000001F, /* OUT1 to OUT4 tri state */
  0x00000000,
  /* etc. */
  0xFFFFFFFF  /* all channels tri state */
};

static ViUInt32 enableData[PATTERN_COUNT] = {
  0xFFFFFFFF,
  0xFFFFFFFF,
  0xFFFFFFF0, /* OUT1 to OUT4 tri state */
  0xFFFFFFFF,
```

```
  0xFFFFFFFF,
  0xFFFFFFF0, /* OUT1 to OUT4 tri state */
  0xFFFFFFFF,
  /* etc. */
  0x00000000  /* all channels tri state */
};

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);
  chk ("rspio4_InitWithOptions");

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

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

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

  return 0;
}
```

8.3.4.2 Error handling

This function checks the return values of a driver call. A message is issued if there is an error.

```
/* 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);
  }
}
```

8.3.4.3 Execution of digital test

```
/* 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;

  /* configure 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");

  /* use hysteresis comparator mode for all sensors */
  sta = rspio4_ConfigureRespPort(vi, RSPIO4_MASK_PORT_ALL,
    RSPIO4_RESP_MODE_HYST, thresholdHigh, thresholdLow);
  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 32 bit static operation */
  sta = rspio4_ConfigureStimMode (vi, RSPIO4_VAL_CTRL_STATIC);
  chk ("rspio4_ConfigureStimMode");

  sta = rspio4_ConfigureRespMode(vi, RSPIO4_VAL_CTRL_STATIC);
```

```
  chk ("rspio4_ConfigureRespMode");

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

8.3.4.4 Execution of a pattern set

```
/* FUNCTION ***********************************************************************/
/* executes the pattern set
*******************************************************************************/
static void executePatternSet( void )
{
  int      loopIdx;
  ViUInt32 response;

  ViUInt32 enableMask  = 0xFFFFFFFF;

  ViReal64 uutResponseDelay = 0.001;

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

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

    Delay(uutResponseDelay);

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

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

8.3.5 Dynamic pattern execution with Low-Level driver functions

8.3.5.1 Main function

```
/* 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;
}
```

8.3.5.2 Error handling

This function checks the return values of a driver call. A message is issued if there is an error.

```
/* 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);
  }
}
```

8.3.5.3 Execution of digital test

```
/* 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 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");
}
```

8.3.5.4 Generation of a pattern set

```
/* 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");
}
```

8.3.6 Triggered pattern execution

In this example, a pulse on trigger line PXI0 of the TSVP backplane triggers output of the pattern. The trigger pulse is generated by the R&S TS-PIO4 module itself. Other measurement modules in the TSVP frame can also use this signal to perform a triggered measurement. It is, of course, also possible to start other R&S TS-PIO4 modules in the system using this signal. In the example, the stimulus logic and response logic (IT1 and IT2) are started by the same signal (PXI0). In addition, the output clock of the stimulus logic (internal trigger logic block 1 or IT1) is routed to pin XTO of the front connector.

8.3.6.1 Main program

```
/* 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;
}
```

8.3.6.2 Error handling

```
/* 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);
```

```
  }
}
```

8.3.6.3 Triggered digital test

```
/* 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");
}
```

9 Maintenance, storage and disposal

9.1 Storage

Protect the product against dust. Ensure that the environmental conditions, e.g. temperature range and climatic load, meet the values specified in the data sheet.

9.2 Disposal

Rohde & Schwarz is committed to making careful, ecologically sound use of natural resources and minimizing the environmental footprint of our products. Help us by disposing of waste in a way that causes minimum environmental impact.

Disposing electrical and electronic equipment

A product that is labeled as follows cannot be disposed of in normal household waste after it has come to the end of its service life. Even disposal via the municipal collection points for waste electrical and electronic equipment is not permitted.

Figure 9-1: Labeling in line with EU directive WEEE

Rohde & Schwarz has developed a disposal concept for the eco-friendly disposal or recycling of waste material. As a manufacturer, Rohde & Schwarz completely fulfills its obligation to take back and dispose of electrical and electronic waste. Contact your local service representative to dispose of the product.

10 Troubleshooting

If the system is not running properly, try to find the problem with the following tests. If the tests do not help to locate the problem, contact your Rohde & Schwarz service representative.

10.1 LED test

The module has three LEDs on its front panel that indicate its status.

After turning on the system, all LEDs light up for a short time to indicate that the power supply is present and that all LEDs are working.

- A single LED does not light up in that time frame:
  Indicates a faulty LED or faulty LED control.
- All LEDs do not light up during that time frame:
  Indicates that the power supply for the module is faulty.
  Check the status LEDs of the main power supply module in slot A3 and A4.

For rear modules, you have to check the LEDs separately, see "Power-on test for modules with a rear I/O supply module" on page 83.

10.2 Power-on test

The power-on test runs at the same time as the LED test. The following statements can be made regarding the different display states of the LEDs.

- "PWR LED" (green LED) = on
  Indicates that all power supply voltages are present.
- "PWR LED" (green LED) = off
  Indicates that at least one power supply voltage is missing.
- "ERR LED" (red LED) = off
  If the green LED is illuminated at the same time, indicates that the system is working without any errors.
- "ERR LED" (red LED) = on (or blinking)
  Indicates a hardware problem.

Power-on test for modules with a rear I/O supply module

If the green LED indicates a problem with the supply voltage, check the LEDs of the corresponding rear I/O supply module separately. If the LEDs on the rear I/O module also indicate a supply voltage failure, replace the rear I/O module.

10.3 R&S TSVP self-test

The R&S TSVP self-test is an extensive test procedure for the whole system or individual components. After the test is done, you receive a test report for all components that have been tested.

The self-test uses the R&S TS-PSAM module as a measurement unit. The functionality of the modules in the system is ensured by measurements via the analog measurement bus.

For more information about running the system self-test and the test procedures, refer to the R&S TSVP service manual.

10.4 Contacting customer support

Technical support – where and when you need it

For quick, expert help with any Rohde & Schwarz product, contact our customer support center. A team of highly qualified engineers provides support and works with you to find a solution to your query on any aspect of the operation, programming or applications of Rohde & Schwarz products.

Contact information

Contact our customer support center at www.rohde-schwarz.com/support, or follow this QR code:

Figure 10-1: QR code to the Rohde & Schwarz support page

Annex

A Specifications

For an overview of technical specifications of the R&S TS-PIO4 module, refer to the corresponding product brochure / data sheet.

If discrepancies exist between information in this manual and the values in the data sheet, the values in the data sheet take precedence.

B Block diagrams

This section contains a functional block diagram of the R&S TS-PIO4 module as well as a detailed block diagram.

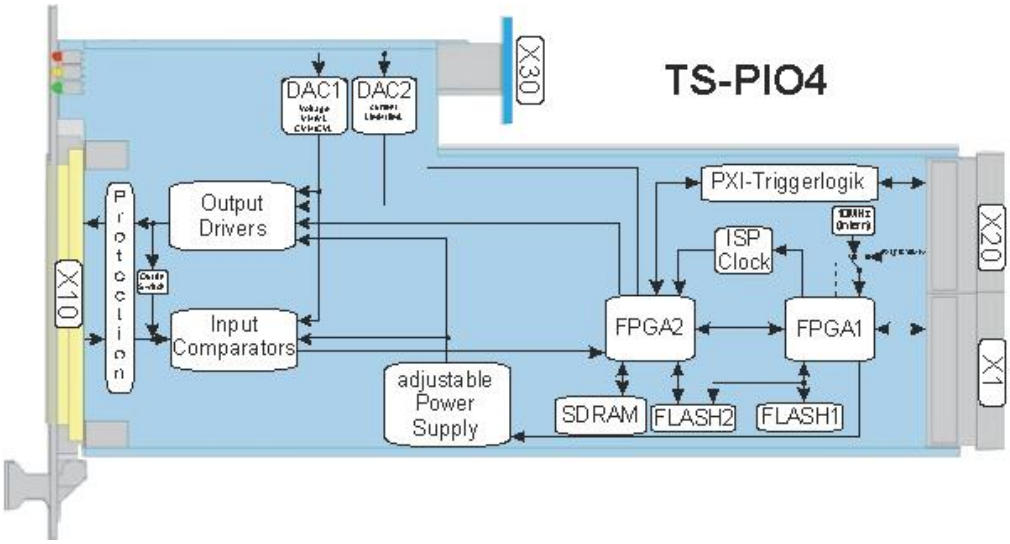


Figure B-1: Functional block diagram of R&S TS-PIO4 module

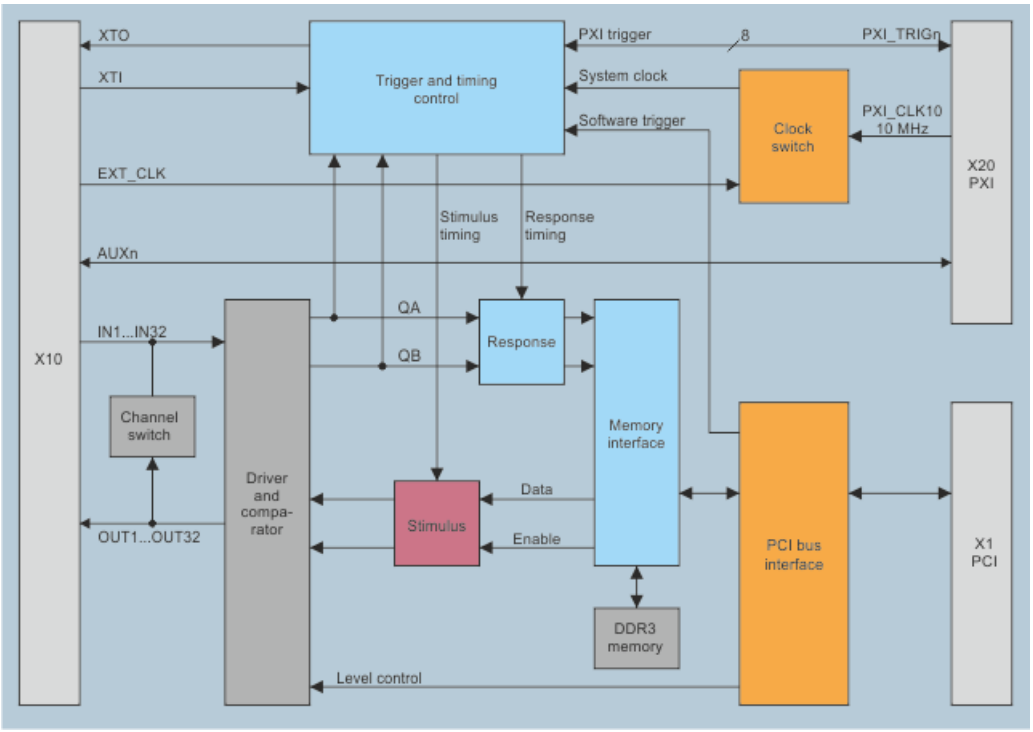


Figure B-2: Detailed block diagram of R&S TS-PIO4 module

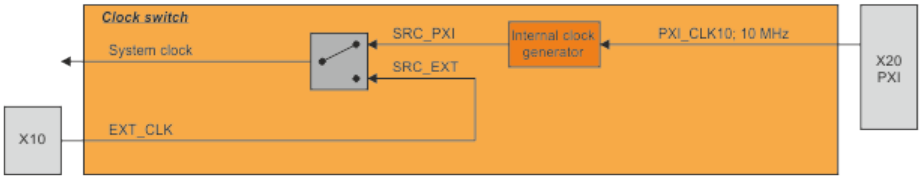


Figure B-3: Clock switch

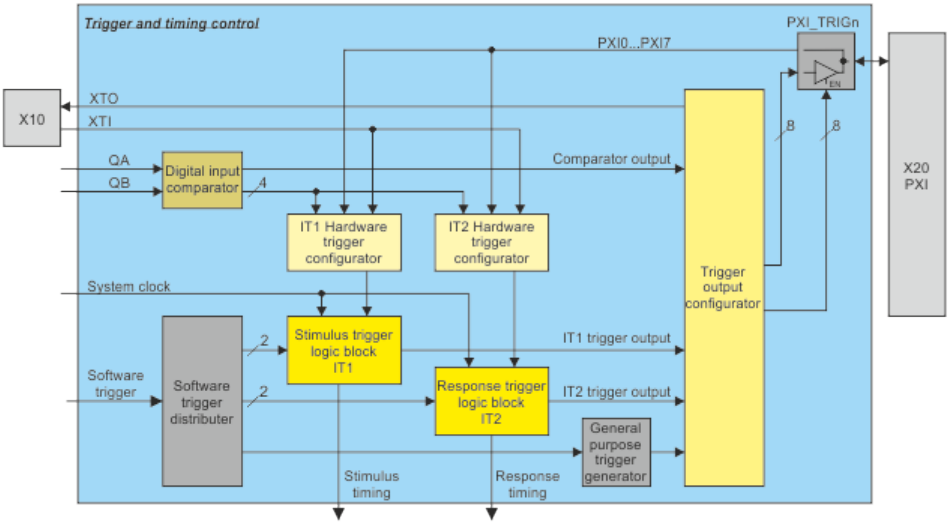


Figure B-4: Trigger und timing control

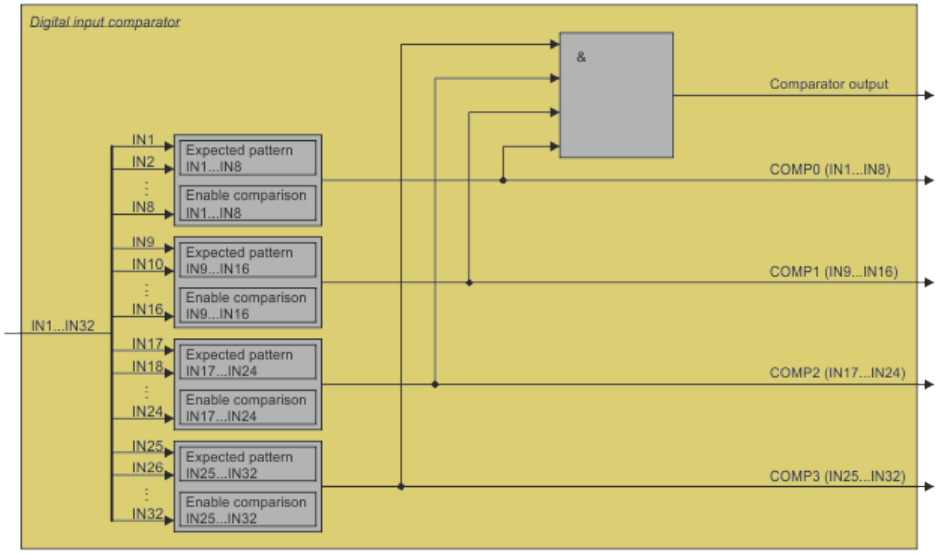


Figure B-5: Digital input comparator

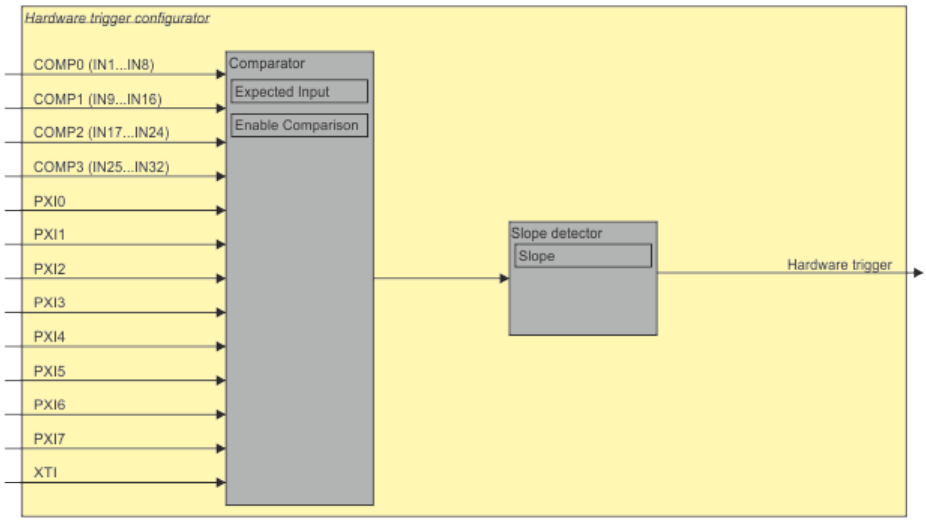


Figure B-6: Hardware trigger configurator

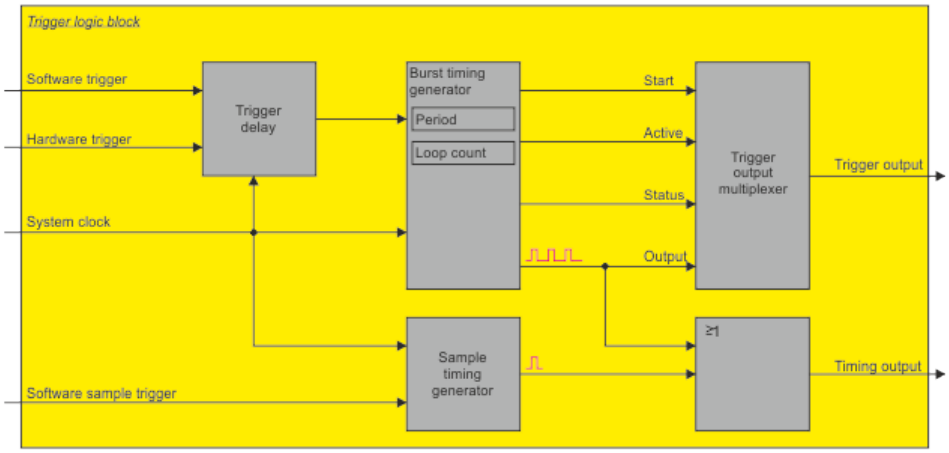


Figure B-7: Trigger logic block

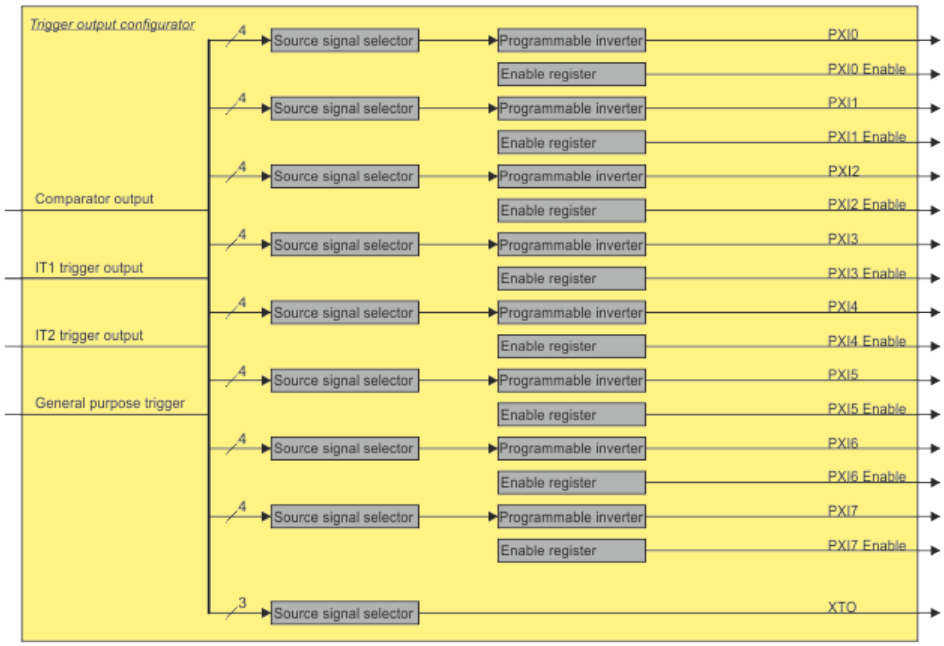


Figure B-8: Trigger output configurator

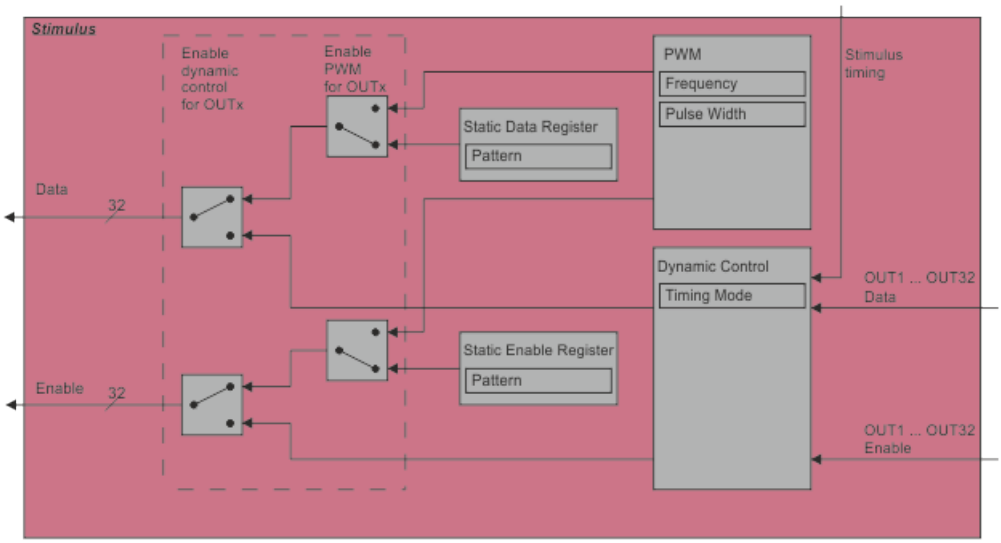


Figure B-9: Stimulus

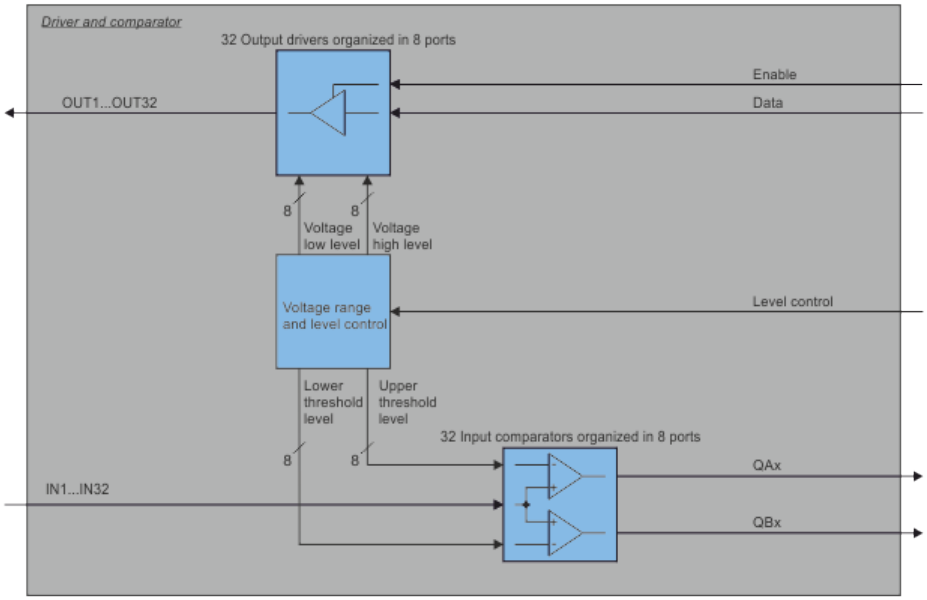


Figure B-10: Driver and comparator

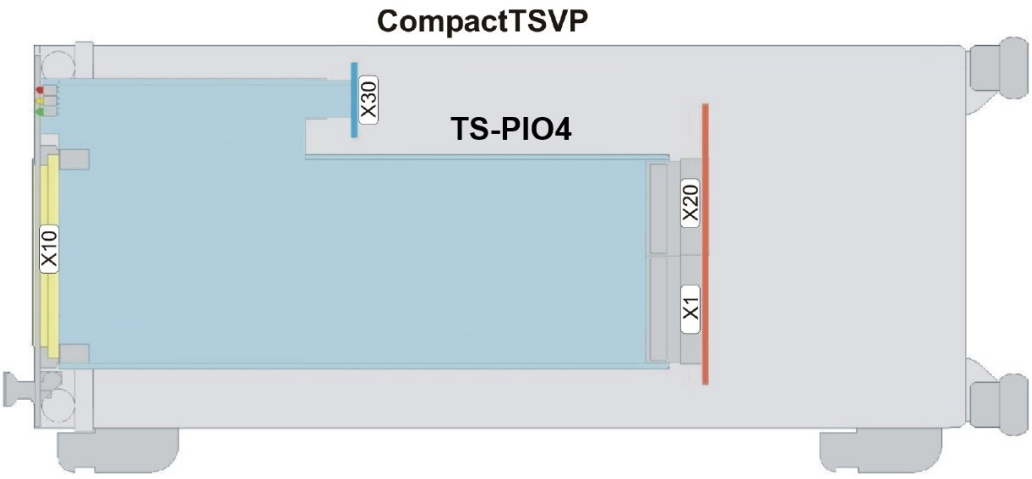


Figure B-11: R&S TS-PIO4 in R&S CompactTSVP

C Interface description

C.1 Connector X10

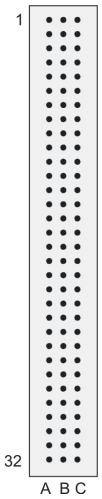


Figure C-1: Connector X10 (mating side)

Table C-1: Pin assignment of R&S TS-PIO4 connector 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
14	IN12	IN13	IN14
15	OUT15	OUT16	GNDNO
16	IN15	IN16	GND

	A	B	C
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

C.2 Connector X20

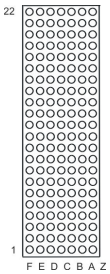


Figure C-2: Connector X20 (mating side)

Pin	F	E	D	C	B	A	Z	
22		GA0	GA1	GA2	GA3	GA4		
21				3.3V				
20			GND		AUX1	AUX2		
19		AUX1	AUX2		GND			
18		PXI_TRIG6	GND	PXI_TRIG5	PXI_TRIG4	PXI_TRIG3		J20
17		PXI_CLK10			GND	PXI_TRIG2		
16		PXI_TRIG7	GND		PXI_TRIG0	PXI_TRIG1		
15					GND			
14	NC						NC	C
13	NC						NC	O
12	NP	AUX1				AUX3	NP	N
11	NP	AUX1				AUX3	NP	N
10	NC	AUX2				AUX4	NC	E
9	NC	AUX2				AUX4	NC	C
8	NC			TMS_EXT_2 N.C.			NC	T
7	NC			TCK_EXT_2 N.C.			NC	O
6	NC			TDO_EXT			NC	R
5	NC			TDI_EXT			NC	
4	NC			TMS_EXT			NC	
3		RSA0	RRST#	TCK_EXT	GND	RSDO		
2			RSDI	RSA1	RSA2	RSCLK		
1					GND	RCS#		
Pin	Z	E	D	C	B	A	F	

Figure C-3: Pin assignment of connector X20

C.3 Connector X30

Table C-2: Assignment of R&S TS-PIO4 connector x30

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

C.4 Connector X1

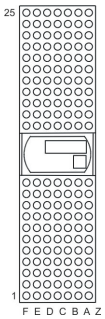


Figure C-4: Connector X1 (mating side)

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	

Figure C-5: Pin assignment of connector X1