# R&S®TS-PIO5 Digital LVDS Functional Test Module User Manual

1178394002
Version 05

ROHDE&SCHWARZ
Make ideas real

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

- R&S®TS-PIO5 (1525.5807.02)

1178.3940.02 | Version 05 | R&S®TS-PIO5

In this manual, products from Rohde & Schwarz are written without the ® symbol, e.g. R&S®TS-PIO5 is written as R&S TS-PIO5.

# 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 โดยจะมีการจัดส่งข้อมูลด้านความปลอดภัย-หลายภาษามาให้พร้อมกับผลิตภัณฑ์ ปฏิบัติตามคำแนะนำในการติดตั้งที่ให้ไว้

Pilihan atau aksesori ini direka bentuk untuk produk Rohde & Schwarz yang ter-tentu. Maklumat keselamatan berbilang bahasa disertakan bersama produk. Ikut ara-han 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 especí-fico. 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-PIO5

This manual describes the function and operation of the R&S TS-PIO5 digital LVDS functional test module for use in the test system versatile platform R&S TSVP. The module is controlled by the PXI bus. The board occupies just one slot at the front of the R&S TSVP, therefore making it possible to set up extremely high-performance and compact measuring systems.

The R&S TS-PIO5 digital functional test module is used wherever digital circuits are tested by flexibly programmable static or dynamic stimulation and the response is recorded. Bidirectional LVDS, LVTTL and RS485 transceiver channels are available for this purpose.

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-PIO5 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.

A soft panel is available for interactive 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-PIO5 digital functional test module.

- Two MDR connectors, each with 10 LVDS channels (8 data, 1 universal channel, 1 clock)
  LVDM bidirectional, terminated with 100 Ω
- 2 LVTTL signals per MDR connector
- +5 V / max. 0.30 A available at every MDR connector, protected by fuse and diode
- 10 RS485 transceiver channels available via a contact strip on the board
- Sample rate up to 200 Msample/s
- Maximum resolution 5 ns
- Memory depth 2 Mpattern (32 bit)
  – 10 bit RS485
  – 4 bit LVTTL
  – 18 bit LVDS
- Synchronization/triggering via
  – PXI trigger bus
  – SMB clock/trigger input and output
  – PXI 10 MHz clock
- 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-PIO5 module is designed as a long plug-in module for mounting in the front of PXI based base units.

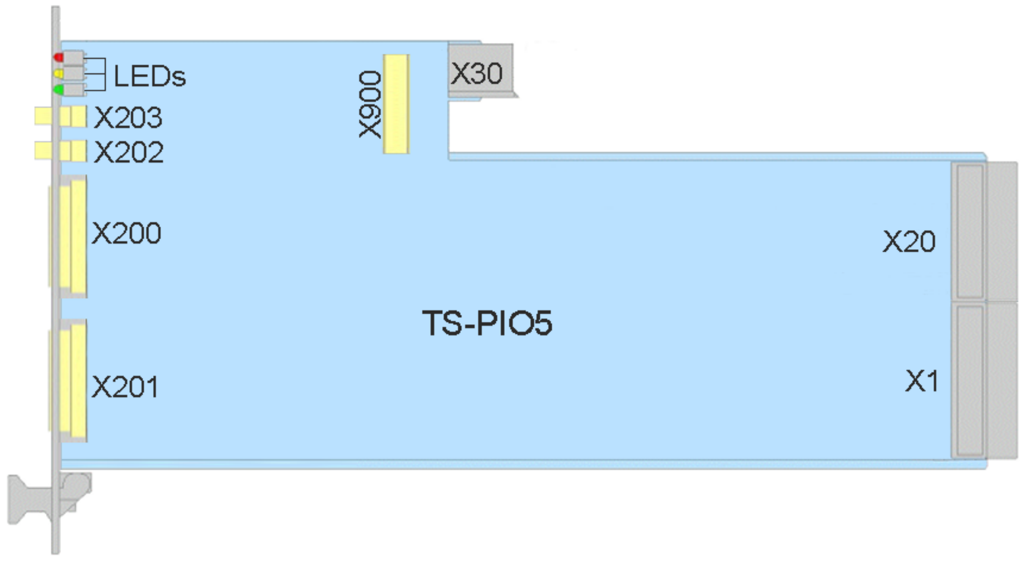


*Figure 4-1: Interfaces on the R&S TS-PIO5 module*

| | |
|---|---|
| LEDs | = Chapter 4.1, "Status LEDs", on page 15 |
| X1 / X20 | = Chapter 4.2, "Connectors X1 and X20", on page 16 |
| X30 | = unused (intended to provide mechanical stability) |
| X200 / X201 | = Chapter 4.3, "Connectors X200 and X201", on page 16 |
| X202 / X203 | = Chapter 4.4, "Connectors X202 and X203", on page 16 |
| X900 | = Chapter 4.5, "Connector X900", 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 based

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

See Chapter C.1, "Connector X1", on page 87 and Chapter C.2, "Connector X20", on page 89 for a detailed description of the connectors.

## 4.3 Connectors X200 and X201

**Type**: MDR-26

**Label**: "Digital A" (X200) and "Digital B" (X201)

Digital interfaces to connect LVDS units under test to the module.

Note that the digital interfaces are not compatible with the I/Q ports of other Rohde & Schwarz devices.

See Chapter C.4, "Connector X200 (Digital A)", on page 92 and Chapter C.5, "Connector X201 (Digital B)", on page 94 for a detailed description of the connectors.

## 4.4 Connectors X202 and X203

**Type**: Female SMB

**Label**: "CLK" (X202) and "TRIG" (X203)

Interface to connect clock and trigger signals to the module (input or output).

See Chapter C.6, "Connector X202 (CLK)", on page 95 and Chapter C.7, "Connector X203 (TRIG)", on page 95 for a detailed description of the connectors.

## 4.5 Connector X900

**Type**: Double row .025" square post terminal strip (26 pins)

Interface to feed RS485 signals into the unit under test using a ribbon cable and an application-specific front panel section in the adjacent slot.

See Chapter C.8, "Connector X900 (RS485)", on page 96 for a detailed description of the connectors.

# 5 Installing the module

The R&S TS-PIO5 is a module installed at the front panel of the PXI based R&S TSVP base units.

1. Install the module as described in the user manuals for the base units.
2. Take the system into operation as described in the user manuals for the base unit.

# 6 Typical applications

The R&S TS-PIO5 digital functional test module is used for testing digital modules or devices that are controlled using LVDS signals. 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. Digital input patterns that measure output signals are applied for this purpose.

The R&S TS-PIO5 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)
- Downloads, e.g. for flash components

## 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 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

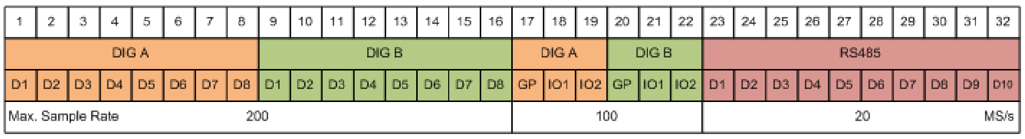
The R&S TS-PIO5 digital functional test module enables FPGA-based stimuli and response operation of the LVDS, LVTTL as well as RS485 transceiver paths with different maximum data rates depending on signal type.

In the static operating mode, the driver software controls the output and reading in of data via the PCI bus.

In the dynamic operating mode, a DDR3 RAM initially buffers the data in order to then output it with the maximum possible sampling rate on the user interface. When data is read in, it is buffered in the DDR3 RAM at the given data rate so that this data can then be processed by the driver software.

The digital channels of the R&S TS-PIO5 module are divided into logical groups according to the path characteristics. Depending on the maximum sample rate, individual paths can be used together for one transmission. Figure 7-1 lists the maximum sample rates for the individual paths.

If the maximum data rate of 200 Msample/s is used, channels 1 to 16 are used dynamically and channels 17 to 32 are used statically. At sample rates of 25 Msample/s to 100 Msample/s, channels 1 to 22 are used dynamically and the rest can be operated statically. At sample rates of 20 Msample/s and lower, all channels can be used dynamically. The channels that can be used dynamically can be switched bit by bit between static and dynamic mode. The outputs can be individually deactivated statically and, at sample rates below 100 Msample/s, also dynamically. This must be taken into consideration when configuring the pattern set.



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIG A | | | | | | | | DIG B | | | | | | | | DIG A | | | DIG B | | | RS485 | | | | | | | | | |
| D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | GP | IO1 | IO2 | GP | IO1 | IO2 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 |
| Max. Sample Rate | | | | | | | | 200 | | | | | | | | 100 | | | | | | 20 | | | | | | | | | MS/s |

*Figure 7-1: Digital channels of R&S TS-PIO5 modules*

For synchronous operation, the functional test module itself provides the clock or, operating as a slave, can also link the data transmission to an external clock.

All settings as well as clock generation take place on the module itself, which means that no additional stimulus modules are necessary.

### 7.1.2 LVDS channels

The R&S TS-PIO5 digital functional test module provides 18 LVDS ports (DIGA_Dx, DIGA_GP, DIGB_Dx and DIGB_GP) for bidirectional operation. As a result of the LVDM driver technology, the outputs can be operated with termination at both ends. 100 Ω termination resistors are used at the outputs of the R&S TS-PIO5 functional test module. The TX/RX paths are permanently connected to each other, which allows a self-test to be performed without external cabling. An integrated open-circuit fail-safe circuit ensures that a stable condition is maintained even if inputs are open.

### 7.1.3 LVTTL channels

The R&S TS-PIO5 digital functional test module provides 4 LVTTL ports (DIGA_IOx and DIGB_IOx) in 3.3 V logic for bidirectional operation. They are designed to be 5 V tolerant. Built-in pull-up resistors ensure a stable high state even if inputs are open.

### 7.1.4 Clock paths

For synchronous data transmission in both directions, the R&S TS-PIO5 functional test module itself provides a clock. This clock can be tapped on a designated LVDS channel (DIGA_CLKIN and DIGB_CLKIN) at both MDR connectors or at the SMB female connector CLK.

In slave mode, sampling is determined by an external clock, and for this purpose the clock is applied at one of the connections described here.

The digital channels can be operated in different scenarios as required. The clock provided on the module (Clock Master) is available for outputting or recording the patterns. Alternatively, they can be used together with an external clock (Clock Slave) that is applied at the SMB female connector CLK or at the CLK pins of the LVDS connectors Digital A and Digital B.

*Table 7-1: Configuration of clock distribution and timing*

| **Scenario based on TS-PIO5** | **Clock frequency**<br>**MHz** | **Timing pattern** | **Possible sample rate – direct (DDR)**<br>**Msample/s** |
|---|---|---|---|
| Clock Master | 100<br>(clock output) | Integer factor of 100 | 0 to 100 (200) |
| Clock Slave | 20 to 100<br>(clock input)<br>0 to 20<br>(trigger input) | | 0 to 100 (40 to 200)<br>With low sample rates, use trigger input |

| Delay gradation | Delay | Use |
|---|---|---|
| Delay 1 | 2.5 ns, typical (2 ns to 4 ns) | Offset the sampling time point within the active clock phase by a fixed time |
| Delay 2 | 180° (inverted clock) | Offset sampling to the negative edge (20 MHz to 100 MHz) |
| Delay 3 | N clock cycles | Sampling only after N clock cycles of the system clock (20 MHz to 100 MHz) |
| Delay 4 | Combination of delay 1 to 3 | Further delays can be implemented by freely combining the delays. |

## 7.1.5 RS485 transceiver channels

The module has ten RS485/RS422 transceiver channels provided on a pin connector. They enable data to be transmitted at a RS485/RS422-compatible level; the FW does not, however, contain an RS485 protocol.

The paths have a switchable termination of 120 Ω. The TX/RX paths are permanently connected to each other, which allows a self-test to be performed without external cabling. An integrated open-circuit fail-safe circuit ensures that a stable high state is maintained even if inputs are open.

## 7.1.6 Memory management in driver

Stimulus data records that are to be loaded to the module using `rspio5_LoadData` are first stored in the memory management of the driver. An almost unlimited number of data records can be created here. For each data record, the driver assigns an ID via which this data record can then, if required, be identified and loaded to the module.

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 `rspio5_ConfigureStimMode()` and `rspio5_ConfigureRespMode()` functions.

## 7.1.7 Stimulus memory

2 Mpattern 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 loaded to the buffer memory of the driver using the `rspio5_LoadData()` functions. These functions return an ID. This ID is then used to load the stimulus memory to the module.

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 buffer memory in the driver is emptied using `rspio5_DiscardData()`.

### 7.1.8 Response memory

The recorded response data is also stored in a 2 Mpattern memory.

### 7.1.9 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-PIO5, the static digital test can be performed using IVI Digital functions or the low-level driver functions.

### 7.1.10 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 with a resolution of 10 ns. 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 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 trigger event and the start of the data sampling at the inputs. The pattern rate is the reciprocal of the pattern period.

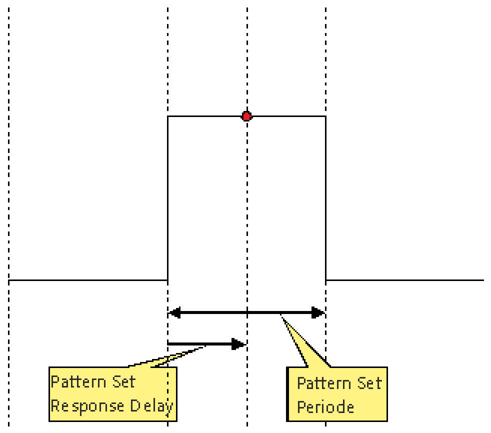


*Figure 7-2: Pattern set period and response delay*

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.

The R&S TS-PIO5 output channel drivers can be deactivated so that they are in tristate mode. Is a signal to be driven on a channel which was in tristate mode for a fairly long time (> 20 us) then it can take up to 10 ns until the driver gets active again. In this case the first pattern to be output can be reduced in length for max. 10 ns.

## 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

Instrument 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.

All the functions of the device driver are described fully in the online help and in the LabWindows/CVI function panels.

## 7.2.1 Digital test with device driver function

### 7.2.1.1 Initialization

- `rspio5_init`
- `rspio5_InitWithOptions`
- `rspio5_close`

### 7.2.1.2 Auxiliary function

- `rspio5_reset`
- `rspio5_LockSession`
- `rspio5_UnlockSession`
- `rspio5_self_test`
- `rspio5_revision_query`

### 7.2.1.3 Error queries

- `rspio5_error_query`
- `rspio5_GetErrorInfo`
- `rspio5_ClearErrorInfo`
- `rspio5_error_message`

### 7.2.1.4 Digital test with low-level driver functions

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

**Static digital test with low-level driver functions**

- `rspio5_SetDoutState`
- `rspio5_GetDinState`

**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 examples).

- `rspio5_ConfigureStimMode`
- `rspio5_ConfigureRespMode`
- `rspio5_ConfigureStimTiming`

- `rspio5_ConfigureRespTiming`
- `rspio5_LoadData`
- `rspio5_LoadStimBuffer`
- `rspio5_ExecutePattern`
- `rspio5_AbortExecution`
- `rspio5_FetchPatternResponseData`
- `rspio5_FetchPatternResponseDataFull`
- `rspio5_DiscardData`

### 7.2.1.5 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**

- `rspio5_ClearPattern`
- `rspio5_ConfigureChannelOpcode`
- `rspio5_ConfigureMode`
- `rspio5_ConfigureGroupOpcode`
- `rspio5_ConfigureLargeGroupOpcode`
- `rspio5_CreatePattern`
- `rspio5_GetChannelName`
- `rspio5_GetChannelOpcode`
- `rspio5_GetGroupOpcode`
- `rspio5_ConfigureStaticResponseDelay`
- `rspio5_ExecuteStaticPattern`
- `rspio5_FetchStaticChannelOpcode`
- `rspio5_GetStaticChannelName`
- `rspio5_FetchStaticChannelListResults`
- `rspio5_FetchStaticChannelResult`
- `rspio5_FetchStaticChannelData`
- `rspio5_FetchStaticChannelListData`
- `rspio5_AbortPatternSet`
- `rspio5_BeginPatternSetLoading`
- `rspio5_ClearPatternSet`
- `rspio5_ConfigurePatternSetMaxTime`

- `rspio5_ConfigurePatternSetTiming`
- `rspio5_CreatePatternSet`
- `rspio5_ExecutePatternSet`
- `rspio5_FetchPatternSetResult`
- `rspio5_GetDynamicChannelName`
- `rspio5_GetDynamicPatternCount`
- `rspio5_FetchDynamicChannelOpcode`
- `rspio5_GetPatternSetCount`
- `rspio5_GetPatternSetExecutedPatternCount`
- `rspio5_GetPatternSetLoadedPatternCount`
- `rspio5_GetPatternSetName`
- `rspio5_InitiatePatternSet o`
- `rspio5_LoadDynamicPattern`
- `rspio5_WaitUntilPatternSetComplete`
- `rspio5_FetchDynamicChannelListResults`
- `rspio5_FetchDynamicChannelListPatternResults`
- `rspio5_FetchDynamicChannelResult`
- `rspio5_FetchDynamicPatternResult`
- `rspio5_FetchDynamicPatternListResults`
- `rspio5_FetchDynamicChannelData`
- `rspio5_FetchDynamicChannelListData`
- `rspio5_FetchDynamicChannelListPatternData`

### 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 text editor.

The file containing the pattern set is loaded during runtime to one or more R&S TS-PIO5 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.

#### 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.

INPUTS are inputs of the UUT and therefore correspond to OUT1 to OUT32 of the R&S TS-PIO5 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 Configuring DIOMGR

The waveform file contains the logical name of the inputs and outputs. Assignment of physical channels on the R&S TS-PIO5 modules to logical names takes place in the file `application.ini`.

```
[bench->823916]
DIODevice = device->pio5
DIOChannelTable = io_channel->823916

[io_channel->823916]

E_COM         = pio5!out1
E_EC1         = pio5!out2
E_EC2         = pio5!out3
E_EC3         = pio5!out4
N14CR0805170  = pio5!out5

N13HCPL31500  = pio5!in1
N13HCPM31500  = pio5!in2
N13HCPN31500  = pio5!in3
N13HCPO31500  = pio5!in4

bus0          = pio5!out20
bus1          = pio5!out21
bus2          = pio5!out22
bus3          = pio5!out23
bus4          = pio5!out24
bus5          = pio5!out25
bus6          = pio5!out26
bus7          = pio5!out27
```

Like all GTSL libraries, the DIO manager is configured in an `application.ini` file. The keyword `DIODevice` refers to the R&S TS-PIO5 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 7-2: 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`<br>`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 7-3: 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 7-4: 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 Loading 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-PIO5 module, the timing must be converted to a fixed framework that 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 `rspio5_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".

#### 7.2.2.5 Executing 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. Details concerning the driver functions can be found in the GTSL help contained in the driver software.

### 7.3.1 Configuration of stimulus and response channels

Whether the driver is to be active or inactive can be set for each channel.

- `rspio5_SetDoutState`

The settings are made for one or more channels depending on the parameters in the function call. The measurement channel inputs are permanently connected to the output drivers of the same channel. For the response mode of a channel, the respective driver is set inactive in the function call.

### 7.3.2 Time setting 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 characteristics of the stimulus channels during output of a pattern set, the `rspio5_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.

R&S TS-PIO5 can output stimulus data not before 50 ns after the trigger event occurred. By means of parameter Trigger Delay an additional delay time can be configured.

**Fast modes: 100MS and 200MS**

The 100MS and 200MS modes can be set using the low-level driver function `rspio5_ConfigureStimTiming`.

Tristate control is not possible in either of the two "fast modes". Tristate channels are set statically.

The "Pattern Period" parameter is fixed. The corresponding parameter in `rspio5_ConfigureStimTiming` must be defined, although without any impact.

**Example:**

Channels DIGITAL B set as outputs and DIGITAL A as inputs

DIGITAL A set as static channels to Tristate

```
rspio5_SetDoutState (vi, 0x000000FF, RSPIO5_STIM_MODE_TRISTATE
```

DIGITAL B set as dynamic outputs

```
rspio5_ConfigureStimMode (vi, RSPIO5_VAL_CTRL_32BIT, 0x0000FF00)
```

## 7.3.3 Time setting 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 `rspio5_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. Even if a stimulus trigger delay of 0.0 ns is configured with function `rspio5_ConfigureStimTiming`, R&S TS-PIO5 can output stimulus data not before 50 ns after the trigger event occurred. However, sampling of input channels starts immediately after the trigger event occurs, if parameter response trigger delay of function `rspio5_ConfigureRespTiming` is set to 0.0 ns.

Example:

- Stimulus pattern period: 50 ns
- Response pattern period: 50 ns
- Stim. trigger delay: 0.0 ns
- Resp. trigger delay: 25.0 ns

In this example the first sampling of the input signal is done 25 ns before the first pattern is output on R&S TS-PIO5 output channels. If the first sampling shall be done in the middle of the first output pattern, the response trigger delay has to be increased to 75 ns. A possible signal delay time caused by the device under test has to be added to the response trigger delay time.

**Fast modes: 100MS and 200MS**

The 100MS and 200MS Modes can be set using the low-level driver function `rspio5_ConfigureRespTiming`.

Tristate control is not possible in either of the two "fast modes". Tristate channels are set statically.

The "Pattern Period" parameter is fixed. The corresponding parameter in `rspio5_ConfigureRespTiming` must be defined, although without any impact.

In the 200MS Mode, the response data is read in with rising and falling clock edge. The response trigger delay, however, can be set in whole clock steps only. Depending on the signal delay, it is therefore possible that, due to the set delay, the response data cannot be saved synchronously to the stimulus data. The stimulus and response memories are then offset by one pattern. This may result in loss of the first or last response pattern. This must be taken into consideration when generating the stimulus data. The following table shows the correlation.

| Index | Stimulus pattern | Response pattern |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 3 | 4 |
| 4 | 4 | xxx |

### 7.3.4 Configuration of data width in dynamic mode

- `rspio5_ConfigureStimMode`
- `rspio5_ConfigureRespMode`

If required, the digital channels can be configured in different combinations for simultaneous static and dynamic operation. This allows the configuration of e.g. control signals 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. Each channel can be switched individually.

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, all 32 channels are always dynamic.

## 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 structure of the trigger units, see the following figures:

- Figure B-6
- Figure B-7
- Figure B-8
- Figure B-9
- Figure B-10

**Inputs of trigger unit:**

On the input side, the `rspio5_ConfigHWTriggerInput` function is used to configure whether and which inputs (PXI0 to PXI7, TRIG, 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. Alternatively, the unit can also be triggered using a software trigger (e.g. the `rspio5_executePattern` function starts the processing of a pattern set by means of a software trigger by calling the `rspio5_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 `rspio5_ConfigureStimTiming` and `rspio5_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 PXI7 lines as well as on TRIG in order to, for example, trigger a measurement on a different PXI board. Various trigger sources are available for starting the controllers:

***Table 7-5: Trigger sources***

| **Designation** | **Remarks** |
|---|---|
| SW Trigger | Sequence control starts immediately after arming (when the `rspio5_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. |
| TRIG | LVTTL input at the front female connector X203; a positive or negative edge starts the sequence. Triggering via TRIG is configured with the `rspio5_ConfigHWTriggerInput` function. It is used to define the edge and to activate/deactivate triggering via TRIG. |

| Designation | Remarks |
|---|---|
| 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 `rspio5_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, can also be used as the trigger source. This is again also configured using `rspio5_ConfigHWTriggerInput` as well as `rspio5_ConfigDinComparator` in order to define the condition for the input pattern comparator. |

The `rspio5_ConfigHWTriggerInput` function determines the trigger source. Afterwards the hardware trigger sources are armed using the `rspio5_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 `rspio5_InitiateSWTrigger` function has to be called with the mask RSPIO5_IT_MASK_IT1 only. If only recording is required, the function must be called using only RSPIO5_IT_MASK_IT2. After the available number of patterns have been processed, the associated sequence control switches back to the "Stopped" state. The current state of both sequence controls can be queried using the `rspio5_GetItStatus` function. By calling the `rspio5_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 `rspio5_AbortExecution` function can be used to switch sequence control to the default state.

### 7.4.2 Generation of trigger signals

The R&S TS-PIO5 module is able to generate trigger signals on the following lines:

***Table 7-6: Trigger outputs***

| Designation | Remarks |
|---|---|
| TRIG | LVTTL output on the front connector (XTO) |
| PXI0 to PXI7 | PXI trigger lines on the backplane |

For a change to occur on the trigger lines, an event that provokes the trigger pulse must be assigned to the selected line. Here it is possible to select which trigger unit (IT1/IT2) and which type of output signal are to be used.

*Table 7-7: Triggers of trigger events*

| Designation | Remarks |
|---|---|
| General purpose trigger | The `rspio5_InitiateSWTrigger` function generates a pulse on the configured trigger lines. For this purpose, the PXI line on which a trigger signal is to be generated is selected using the `rspio5_ConfigPxiTrigOut` function. The length of pulse is dependent on the processing speed of the computer. |
| ITn ACTIVE | Triggered when a trigger event is received. Pulse while the pattern set is being output. Routed to the hardware outputs PXI or TRIG by calling the `rspio5_ConfigPxiTrigOut` function. |
| ITn OUT | Trigger block outputs a pulse for each pattern. The pulses are relatively short (minimum pattern rate/2). Routed to the hardware outputs PXI or TRIG by calling the `rspio5_ConfigPxiTrigOut` or `rspio5_ConfigXTO` function. |
| ITn STATUS | Routed to the hardware outputs PXI or TRIG by calling the `rspio5_ConfigPxiTrigOut` or `rspio5_ConfigXTO` function. |
| ITn START | A pulse after the trigger condition has been detected. Routed to the hardware outputs PXI or TRIG by calling the `rspio5_ConfigPxiTrigOut` or `rspio5_ConfigXTO` function. |

It is also possible to set the polarity of the trigger signal for the individual outputs. In addition, the driver stages for the trigger lines can be disabled. All settings are made using the `rspio5_ConfigurePxiTrigOut` and `rspio5_ConfigXTO` function.

See also the example for trigger response and trigger programming.

## 7.5 PWM

The module supports the output of PWM signals at any output.

PWM is configured using the `rspio5_ConfigurePWM` function. The frequency is set in Hz and the duty cycle is specified in %.

The `rspio5_SetStatePWM` function is used to switch PWM on/off for individual channels.

The resolution is 10 ns when the 100 MHz internal reference clock is used. A divider generates the lower frequencies. With this method there are restrictions in choice of frequencies and duty cycles.

The following examples show the relationship at the highest frequencies.

- 50.0 MHz: 50 %
- 33.3 MHz: 33 %, 66 %
- 25.0 MHz: 25 %, 50 %, 75 %

## 7.6 Frequency measurement

The R&S TS-PIO5 supports frequency measurement at inputs DIGA, DIGB, TRIG and PX0 to PX7. The `rspio5_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 gate time of 0 and a defined number of pulses (gateCounts) are specified, the module measures the time required for a certain number of pulses to arrive. The value is also converted into a frequency.

## 7.7 Selection of system clock/clock output

Using the `rspio5_ConfigureClock` function, it is possible to switch between the four different clock inputs:

- Internal clock
- CLK (SMB female connector)
- DIG A
- DIG B

Furthermore, external clocks can be offset by 180° and/or 2.5 ns.

If necessary, the system clock can be applied to one of the above-mentioned outputs, excluding the signal source.

## 7.8 Implementation example: LVDS <-> TTL

Using the R&S TS-PIO5 digital functional test module, fast TTL signals can be transmitted safely over any distance. For this purpose, the LVDS signal must be switched to the TTL level at the destination or, in the reverse direction, the TTL signal must be switched to LVDS.

### 7.8.1 Supply voltage

+5 V (max. 0.3 A) is available at the DIGITAL A and DIGITAL B connectors. This can be switched to the TTL voltage 3.3 V by means of a switching converter or linear regulator.

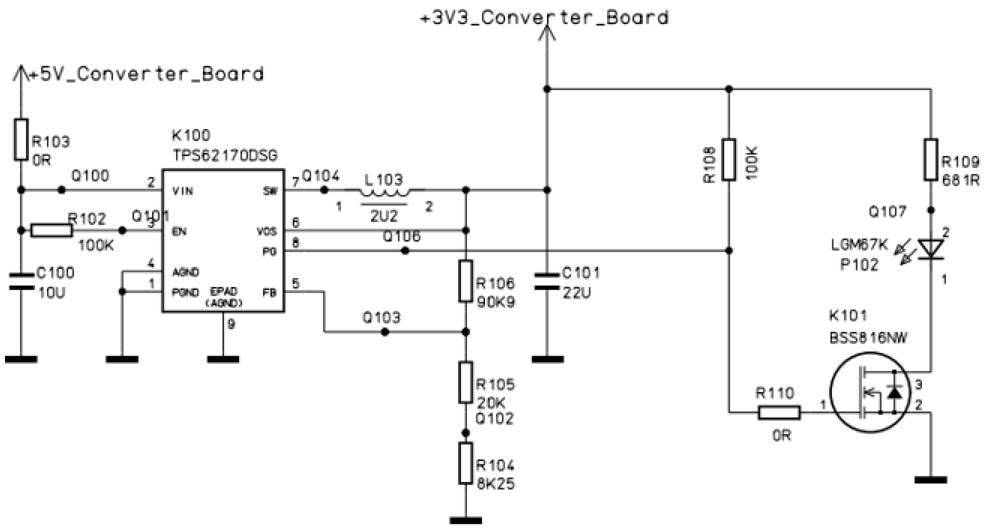


*Figure 7-3: 5 V/3.3 V switching converter*

## 7.8.2 Level converter

The following circuit converts LVDS to TTL and vice versa. The direction of conversion can be determined using jumper X211.

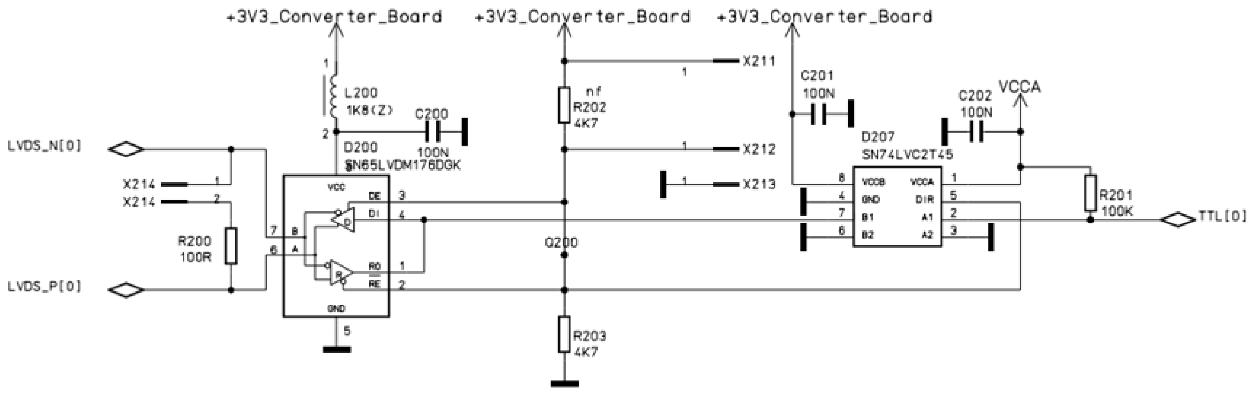


*Figure 7-4: 5 V/3.3 V switching converter*

## 7.8.3 TTL level

The following jumper layout is used to set the TTL level to 5 V, 3.3 V or an externally supplied logic voltage.

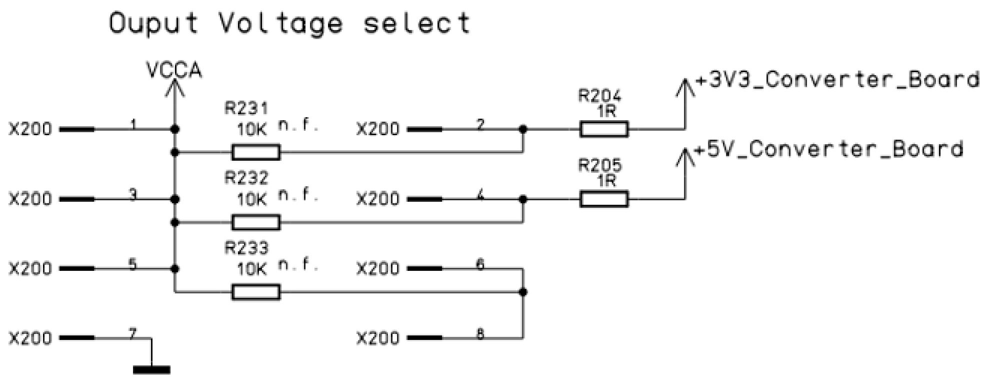


*Figure 7-5: Selection of different TTL levels*

## 7.8.4 EMC

It is advisable to filter the signals directly at the DIGITAL A/B connector.

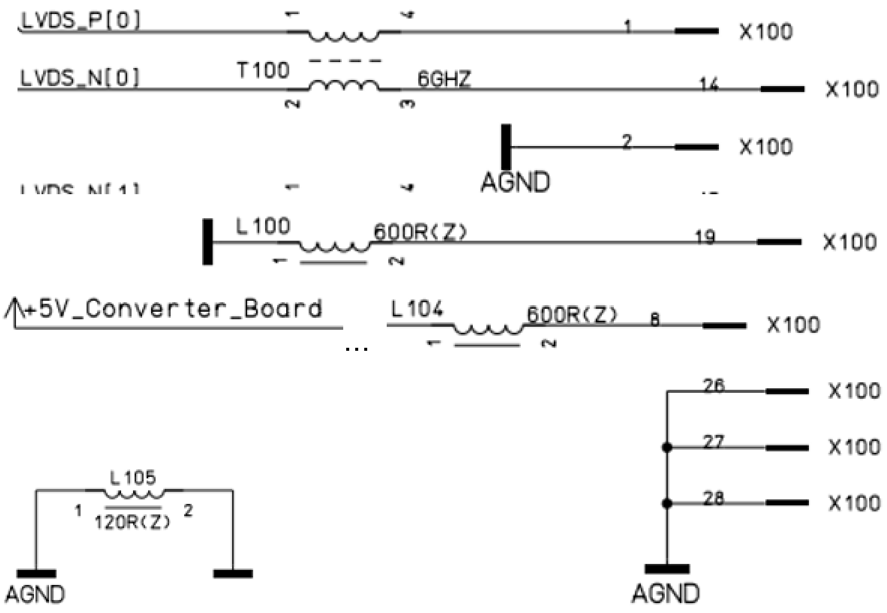


*Figure 7-6: ECM filter*

## 7.8.5 Possible parts list

The table below lists examples of components that can be used to implement the circuit:

| **Component** | **Type** | **Manufacturer** |
|---|---|---|
| DIGITAL A/B connector, 26 Position | 10226-1210 VE | 3M |
| LVDS-TTL transceiver | SN65LVDM176 | Texas Instruments |
| Dual-supply transceiver (2x) | SN74LVC2T45 | Texas Instruments |
| Dual-supply transceiver (1x) | SN74LVC1T45 | Texas Instruments |
| Step-down switching converter | TPS62170DSG | Texas Instruments |
| Coil for switching converter | XFL3012-222ME | Coilcraft |
| Common-mode filter (LVDS) | ACM2012H-900-2P | TDK |
| +5 V and GND filter (600R) | BLM15AG601SN1D | Murata |
| AGND/GND filter (120R) | BLM31PG121SN1L | Murata |

The following R&S article is supported as connecting cable:

1415.0201.02 SMU-Z6 CABLE TVR290 (2 m long).

# 8 Software

## 8.1 Driver software

A LabWindows IVI driver is available for actuation of the R&S TS-PIO5 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 |
|---|---|---|
| `rspio5.dll` | `<GTSL directory>\Bin` | Driver |
| `rspio5.chm` | `<GTSL directory>\Bin` | Help file |
| `rspio5.fp` | `<GTSL directory>\Bin` | LabWindows/CVI function panel file, function panels for CVI development environment |
| `rspio5.sub` | `<GTSL directory>\Bin` | LabWindows/CVI attribute file. This file is required by some "function panels". |
| `rspio5.lib` | `<GTSL directory>\Bin` | Import library |
| `rspio5.h` | `<GTSL directory>\Include` | Header file for driver |

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

The R&S TS-PIO5 LabWindows/CVI device driver uses channel names IN1 to IN32 respectively OUT1 to OUT32. Some driver functions need a channel name as input parameter. Only this channel names are allowed there.

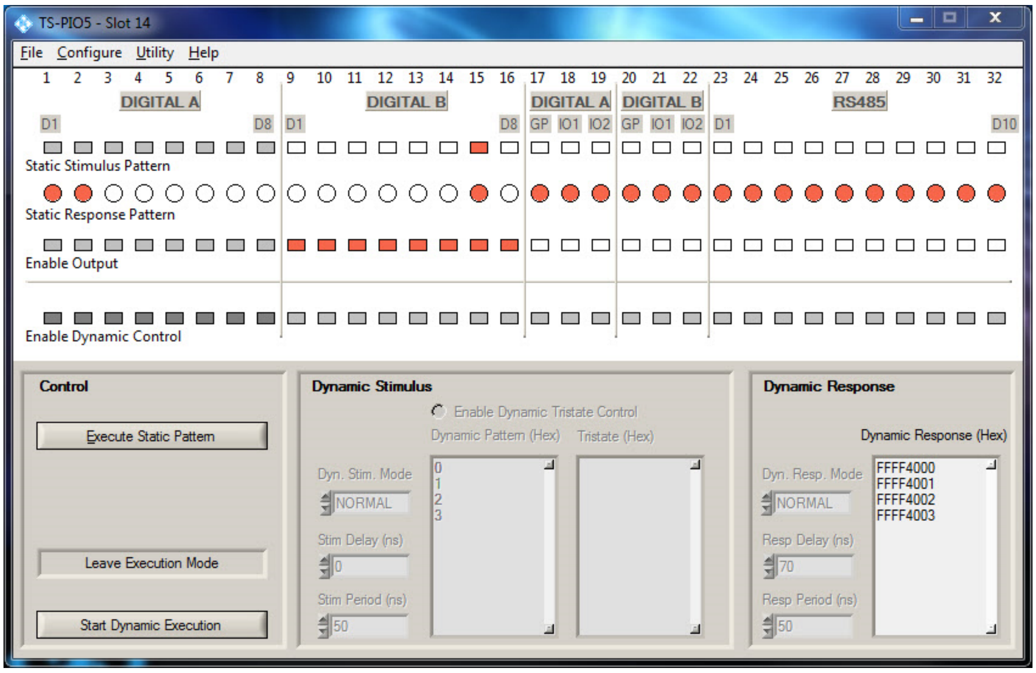
The following table shows the correlation of LabWindows/CVI driver channel names with hardware channel names mentioned in this manual.

| Channels name | Hardware channels name |
|---|---|
| IN1 / OUT1 | DIGA_D1 |
| IN2 / OUT2 | DIGA_D2 |
| IN3 / OUT3 | DIGA_D3 |
| IN4 / OUT4 | DIGA_D4 |
| IN5 / OUT5 | DIGA_D5 |
| IN6 / OUT6 | DIGA_D6 |
| IN7 / OUT7 | DIGA_D7 |

| Channels name | Hardware channels name |
|---|---|
| IN8 / OUT8 | DIGA_D8 |
| IN9 / OUT9 | DIGB_D1 |
| IN10/ OUT10 | DIGB_D2 |
| IN11 / OUT11 | DIGB_D3 |
| IN12 / OUT12 | DIGB_D4 |
| IN13 / OUT13 | DIGB_D5 |
| IN14 / OUT14 | DIGB_D6 |
| IN15 / OUT15 | DIGB_D7 |
| IN16 / OUT16 | DIGB_D8 |
| IN17 / OUT17 | DIGA_GP |
| IN18 / OUT18 | DIGA_IO1 |
| IN19 / OUT19 | DIGA_IO2 |
| IN20 / OUT20 | DIGB_GP |
| IN21 / OUT21 | DIGB_IO1 |
| IN22 / OUT22 | DIGB_IO2 |
| IN23 / OUT23 | RS485_D1 |
| IN24 / OUT24 | RS485_D2 |
| IN25 / OUT25 | RS485_D3 |
| IN26 / OUT26 | RS485_D4 |
| IN27 / OUT27 | RS485_D5 |
| IN28 / OUT28 | RS485_D6 |
| IN29 / OUT29 | RS485_D7 |
| IN30/ OUT30 | RS485_D8 |
| IN31 / OUT31 | RS485_D9 |
| IN32 / OUT32 | RS485_D10 |

## 8.2 Soft panel

The software package of the R&S TS-PIO5 module includes a soft panel (see Figure 8-1).



Dyn. Stim. Mode: NORMAL
Stim Delay (ns): 0
Stim Period (ns): 50
Dynamic Pattern (Hex): 0 1 2 3
Dyn. Resp. Mode: NORMAL
Resp Delay (ns): 70
Resp Period (ns): 50
Dynamic Response (Hex): FFFF4000 FFFF4001 FFFF4002 FFFF4003

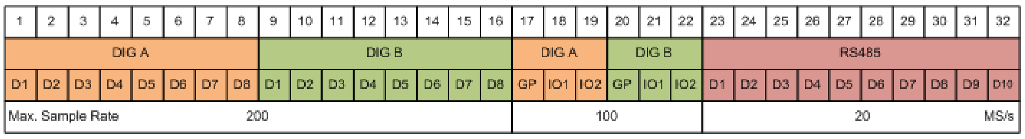
*Figure 8-1: Softpanel R&S TS-PIO5*

The soft panel is based on the IVI driver and enables interactive operation of the module.

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

## 8.2.1 Configuration of digital channels

The digital channels of the R&S TS-PIO5 module are divided into logical groups according to the path characteristics. Depending on the maximum sample rate, individual paths can be used together for one transmission.



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIG A | | | | | | | | DIG B | | | | | | | | DIG A | | | DIG B | | | RS485 | | | | | | | | | |
| D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | GP | IO1 | IO2 | GP | IO1 | IO2 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 |
| Max. Sample Rate | | | | | | | 200 | | | | | | | | | | | | 100 | | | | | | | 20 | | | | | MS/s |

*Figure 8-2: Digital channels of R&S TS PIO5 module*

The stimulus patterns as well as the response data to be included can be defined bit by bit.

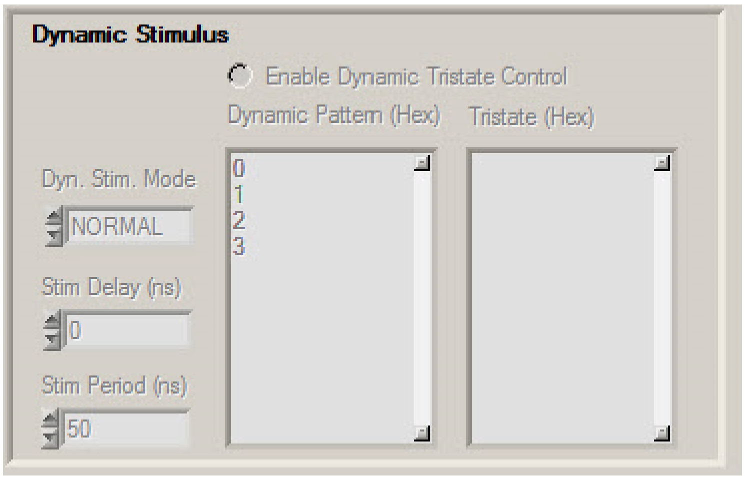
The outputs can be deactivated individually. This must be taken into consideration when configuring the pattern set.

It is possible to define bit by bit whether outputs are to be deactivated statically or whether their state can be decided dynamically in the pattern set.

### 8.2.2 Time settings for data transmission

For dynamic control:

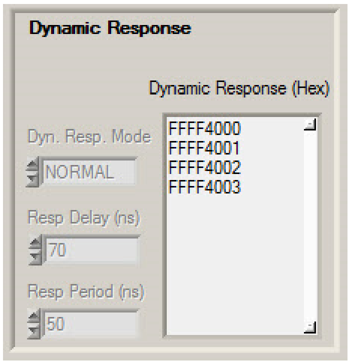
- Configure the desired channels for dynamic control
- Select appropriate the dynamic mode
- Configure delays for stimulus and response
- Select a pattern period for stimulus and response
- Define a pattern set (with tri state information if necessary)



Dyn. Stim. Mode: NORMAL
Stim Delay (ns): 0
Stim Period (ns): 50

Dynamic Pattern (Hex):
```
0
1
2
3
```

*Figure 8-3: Dynamic Stimulus Configuration*



Dyn. Resp. Mode: NORMAL
Resp Delay (ns): 70
Resp Period (ns): 50

Dynamic Response (Hex):
```
FFFF4000
FFFF4001
FFFF4002
FFFF4003
```

*Figure 8-4: Dynamic Response Configuration*

## 8.3 Programming examples

### 8.3.1 Digital test of "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 (pio5Application.ini) are described in Chapter 7.2.2, "Digital test with DIO manager", on page 27. 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 GTSL.pdf".

#### 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", "pio5Application.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;

  /* configure stimulus and response levels */
  DIOMGR_ConfigureStimulus (0, residDiomgr, stimChannels, "DRIVING", 0.0, 0.0,
    &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigureStimulus");

  DIOMGR_ConfigureResponse (0, residDiomgr, respChannels, "HYSTERESIS",
    0.0, 0.0, &errorOccurred, &errorCode, errorMessage);
  cs("DIOMGR_ConfigureResponse");

  /* load the waveform. Patten 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");
}
```

### 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 "rspio5.h"

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::13::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 = rspio5_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", &vi);
  /* check return value */
  chk ("rspio5_InitWithOptions");

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

    /* close driver */
    sta = rspio5_close(vi);
    chk ("rspio5_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];

    rspio5_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 module for dynamic test and result collection */
  sta = rspio5_ConfigureMode(vi, RSPIO5_VAL_EXECUTE_DYNAMIC,
    RSPIO5_VAL_COLLECT_ALL);
  chk ("rspio5_ConfigureMode");

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

  /* configure the timing */
  sta = rspio5_ConfigurePatternSetTiming(vi, patternSetName, 1.0e-6, 0.5e-6);
  chk ("rspio5_ConfigurePatternSetTiming");

  /* run the test */
  sta = rspio5_ExecutePatternSet(vi, patternSetName, 1000);
  chk ("rspio5_ExecutePatternSet");

  /* fetch pass/fail result */
  sta = rspio5_FetchPatternSetResult(vi, patternSetName, &patternSetResult);
  chk ("rspio5_FetchPatternSetResult");

  if ( RSPIO5_VAL_RESULT_FAIL == patternSetResult )
  {
    printf("Pattern set failed\n");
    diagnosis ();
  }
  else
  {
    printf("Pattern set passed\n");
  }

  /* clear pattern set */
  sta = rspio5_ClearPatternSet(vi, patternSetName);
  chk ("rspio5_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 = 0;

  /* create a pattern set */
  sta = rspio5_CreatePatternSet(vi, patternSetName);
  chk ("rspio5_CreatePatternSet");

  /* create a pattern */
  sta = rspio5_CreatePattern(vi, &ph);
  chk ("rspio5_CreatePattern");

  /* start loading */
  sta = rspio5_BeginPatternSetLoading(vi, patternSetName);
  chk ("rspio5_BeginPatternSetLoading");

  /*** 1. pattern : stim all zero, resp don't care */
```

```
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x0);
chk ("rspio5_ConfigureGroupOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i1, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_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 = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IH);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x01);
chk ("rspio5_ConfigureGroupOpcode");
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_LoadDynamicPattern");

/*** 3. pattern: all stim tristate, bus = 02 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x02);
chk ("rspio5_ConfigureGroupOpcode");
```

```
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_LoadDynamicPattern");

/*** 4. pattern: all stim = 0, bus = 03 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x03);
chk ("rspio5_ConfigureGroupOpcode");
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_LoadDynamicPattern");

/*** 5. pattern: N14CR0805170 = 1, bus = 04 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IH);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x04);
chk ("rspio5_ConfigureGroupOpcode");
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_LoadDynamicPattern");

/*** 6. pattern: all stim tristate, bus = 05 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x05);
chk ("rspio5_ConfigureGroupOpcode");
/* load pattern */
sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
chk ("rspio5_LoadDynamicPattern");

/*** 7. pattern: all stim = 0, bus = 06, resp N13HCPN31500 = 1 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
```

```
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x06);
  chk ("rspio5_ConfigureGroupOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, i3, RSPIO5_VAL_OPCODE_OH);
  chk ("rspio5_ConfigureChannelOpcode");
  /* load pattern */
  sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
  chk ("rspio5_LoadDynamicPattern");

  /* ... etc ... */

  /* last pattern : all tristate / don't care */
  sta = rspio5_ConfigureGroupOpcode(vi, ph, out, RSPIO5_VAL_GROUP_IGNORE, 0x0);
  chk ("rspio5_ConfigureGroupOpcode");
  sta = rspio5_ConfigureGroupOpcode(vi, ph, in, RSPIO5_VAL_GROUP_IGNORE, 0x0);
  chk ("rspio5_ConfigureGroupOpcode");
  /* load pattern */
  sta = rspio5_LoadDynamicPattern(vi, patternSetName, ph);
  chk ("rspio5_LoadDynamicPattern");

  /* loading finished */
  sta = rspio5_EndPatternSetLoading(vi, patternSetName);
  chk ("rspio5_EndPatternSetLoading");

  /* pattern is no longer used */
  sta = rspio5_ClearPattern(vi, ph);
  chk ("rspio5_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 = rspio5_GetPatternSetExecutedPatternCount (vi, patternSetName,
  &executedPatternCount);
chk ("rspio5_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 = rspio5_FetchDynamicPatternListResults (vi, patternSetName,
  0, executedPatternCount, executedPatternCount, pPatterns, &actualSize);
chk ("rspio5_FetchDynamicPatternListResults");

/* calculate number of failed patterns */
failedPatterns = 0;
for ( pattern = 0; pattern < executedPatternCount; pattern++ )
{
  if ( RSPIO5_VAL_RESULT_FAIL == pPatterns[pattern] )
  {
    failedPatterns++;
  }
}

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

/* upload data and results for in1 - in4 */
sta = rspio5_FetchDynamicChannelListPatternData(vi, patternSetName,
  0, executedPatternCount, "in1,in2,in3,in4",
  numChannels * executedPatternCount, pData, &actualSize);
chk ("rspio5_FetchDynamicChannelListPatternData");

sta = rspio5_FetchDynamicChannelListPatternResults (vi, patternSetName,
  0, executedPatternCount, "in1,in2,in3,in4",
  numChannels * executedPatternCount, pResults, &actualSize);
chk ("rspio5_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 RSPIO5_VAL_DATA_HIGH:
          printf("1");
          break;
        case RSPIO5_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 RSPIO5_VAL_RESULT_PASS:
          printf(" ");
          break;
        case RSPIO5_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 "rspio5.h"

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::13::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 = rspio5_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", &vi);
  /* check return value */
  chk ("rspio5_InitWithOptions");

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

    /* close driver */
    sta = rspio5_close(vi);
    chk ("rspio5_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];

    rspio5_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 module for static test and result collection */
  sta = rspio5_ConfigureMode(vi, RSPIO5_VAL_EXECUTE_STATIC,
    RSPIO5_VAL_COLLECT_ALL);
  chk ("rspio5_ConfigureMode");

  /* configure the timing */
  sta = rspio5_ConfigureStaticResponseDelay (vi, 0.5e-6);
  chk ("rspio5_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 = rspio5_CreatePattern(vi, &ph);
  chk ("rspio5_CreatePattern");

  /*** 1. pattern : stim all zero, resp don't care */
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
  chk ("rspio5_ConfigureChannelOpcode");
  sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x0);
  chk ("rspio5_ConfigureGroupOpcode");
  sta = rspio5_ConfigureChannelOpcode(vi, ph, i1, RSPIO5_VAL_OPCODE_IOX);
  chk ("rspio5_ConfigureChannelOpcode");
```

```
sta = rspio5_ConfigureChannelOpcode(vi, ph, i2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_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 = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IH);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x01);
chk ("rspio5_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 3. pattern: all stim tristate, bus = 02 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x02);
chk ("rspio5_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 4. pattern: all stim = 0, bus = 03 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x03);
chk ("rspio5_ConfigureGroupOpcode");
```

```
executePattern(ph, patternIdx++);

/*** 5. pattern: N14CR0805170 = 1, bus = 04 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IH);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x04);
chk ("rspio5_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 6. pattern: all stim tristate, bus = 05 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IOX);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x05);
chk ("rspio5_ConfigureGroupOpcode");

executePattern(ph, patternIdx++);

/*** 7. pattern: all stim = 0, bus = 06, resp N13HCPN31500 = 1 */
sta = rspio5_ConfigureChannelOpcode(vi, ph, o1, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o2, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o3, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o4, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, o5, RSPIO5_VAL_OPCODE_IL);
chk ("rspio5_ConfigureChannelOpcode");
sta = rspio5_ConfigureGroupOpcode(vi, ph, bus, RSPIO5_VAL_GROUP_INPUT, 0x06);
chk ("rspio5_ConfigureGroupOpcode");
sta = rspio5_ConfigureChannelOpcode(vi, ph, i3, RSPIO5_VAL_OPCODE_OH);
chk ("rspio5_ConfigureChannelOpcode");

executePattern(ph, patternIdx++);

/* ... etc ... */

/* last pattern : all tristate / don't care */
sta = rspio5_ConfigureGroupOpcode(vi, ph, out, RSPIO5_VAL_GROUP_IGNORE, 0x0);
chk ("rspio5_ConfigureGroupOpcode");
```

```
  sta = rspio5_ConfigureGroupOpcode(vi, ph, in, RSPIO5_VAL_GROUP_IGNORE, 0x0);
  chk ("rspio5_ConfigureGroupOpcode");

  executePattern(ph, patternIdx++);

  /* pattern is no longer used */
  sta = rspio5_ClearPattern(vi, ph);
  chk ("rspio5_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 = rspio5_ExecuteStaticPattern(vi, patternHandle);
  chk ("rspio5_ExecuteStaticPattern");

  sta = rspio5_FetchStaticPatternResult (vi, &patternResult);
  chk ("rspio5_FetchStaticPatternResult");

  if ( RSPIO5_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 = rspio5_FetchStaticChannelListData(vi, "in1,in2,in3,in4",
      numChannels, data, &actualSize);
    chk ("rspio5_FetchStaticChannelListData");

    sta = rspio5_FetchStaticChannelListResults(vi, "in1,in2,in3,in4",
      numChannels, results, &actualSize);
    chk ("rspio5_FetchStaticChannelListResults");

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

      switch ( data[channel] )
      {
        case RSPIO5_VAL_DATA_HIGH:
          printf("high ");
          break;
        case RSPIO5_VAL_DATA_LOW:
          printf("low  ");
          break;
        default:
          printf("?    ");
          break;
      }

      switch ( results[channel] )
      {
        case RSPIO5_VAL_RESULT_PASS:
          printf("pass");
          break;
        case RSPIO5_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 "rspio5.h"

#define PATTERN_COUNT       8

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::13::INSTR";

static ViUInt32 stimData[PATTERN_COUNT] = {
  0x00000000,
  0x00000010,
  0x0000001C, /* tri state */
  0x00000000,
  0x00000010,
  0x0000001C, /* tri state */
  0x00000000,
  /* etc. */
  0x00000000
};

static ViUInt32 portEnable[PATTERN_COUNT] = {
  0xFFFFFFFF,
  0xFFFFFFFF,
  0xFFFFFFFC, /* OTU1, OUT2 tri state */
  0xFFFFFFFF,
  0xFFFFFFFF,
  0xFFFFFFFC, /* OTU1, OUT2 tri state */
  0xFFFFFFFF,
  /* etc. */
  0x00000000  /* OUT1 to OUT32 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 = rspio5_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio5_InitWithOptions");

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

    /* close the driver */
    sta = rspio5_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];

    rspio5_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 )
{


  /* configure 32 bit wide static operation */
  sta = rspio5_ConfigureStimMode (vi, RSPIO5_VAL_CTRL_STATIC, 0x00000000 );
  chk ("rspio5_ConfigureStimMode");

  /* configure 32 bit wide static operation */
  sta = rspio5_ConfigureRespMode(vi, RSPIO5_VAL_CTRL_STATIC);
  chk ("rspio5_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;

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


  for (loopIdx = 0; loopIdx < PATTERN_COUNT; loopIdx++)
  {
   sta = rspio5_SetDoutState(vi, 0xFFFFFFFF, stimData[loopIdx], portEnable[loopIdx]);
   chk ("rspio5_SetDoutState");

   Delay(100.0e-6);

   sta = rspio5_GetDinState(vi, &response);
   chk ("rspio5_GetDinState");

   printf("%3d | 0x%08X | 0x%08X | 0x%08X\n", loopIdx, stimData[loopIdx],
   portEnable[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 "rspio5.h"

#define PATTERN_COUNT       8
#define PATTERN_PERIOD      2.0e-6
#define FETCH_TIMEOUT       0.1

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::13::INSTR";

static ViUInt32 stimData[PATTERN_COUNT] = {
  0x00000000,
  0x00000010,
  0x0000001C, /* tri state */
  0x00000000,
  0x00000010,
  0x0000001C, /* tri state */
  0x00000000,
  /* etc. */
  0x00000000
};

static ViUInt32 stimTristate[PATTERN_COUNT] = {
  0x00000000,
  0x00000000,
  0x00000003, /* tri state */
  0x00000000,
  0x00000000,
  0x00000003, /* tri state */
  0x00000000,
  /* etc. */
  0xFFFFFFFF  /* tri state */
};

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

/* data buffer */
static RSPIO5_DATA_32BIT_TRISTATE stimulus[PATTERN_COUNT];
static RSPIO5_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 = rspio5_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio5_InitWithOptions");

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

    /* close the driver */
    sta = rspio5_close(vi);
    chk ("rspio5_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];

    rspio5_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 triggerDelayStim = 0.0;
  ViReal64 triggerDelayResp = PATTERN_PERIOD / 2.0;

  /* configure 32 bit wide dynamic operation: OUT1 .. OUT32 will be operated
     dynamically; no static channels */
  sta = rspio5_ConfigureStimMode (vi, 0xFFFFFFFF, RSPIO5_VAL_CTRL_32BIT_TRISTATE);
  chk ("rspio5_ConfigureStimMode");

  /* configure 32 bit wide dynamic operation */
  sta = rspio5_ConfigureRespMode(vi,RSPIO5_VAL_CTRL_DYNAMIC);
  chk ("rspio5_ConfigureRespMode");

  /* configure stimulus timing */
  sta = rspio5_ConfigureStimTiming(vi, triggerDelayStim, PATTERN_PERIOD,
                RSPIO5_DYN_STIM_MODE_NORMAL,PATTERN_COUNT);
  chk ("rspio5_ConfigureStimTiming");

  /* configure response timing */
  sta = rspio5_ConfigureRespTiming(vi, triggerDelayResp, PATTERN_PERIOD,
                RSPIO5_DYN_RESP_MODE_NORMAL, PATTERN_COUNT);
  chk ("rspio5_ConfigureRespTiming");

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

  /* start pattern execution */
  sta = rspio5_ExecutePattern(vi, memId);
  chk ("rspio5_ExecutePattern");

  /* read the response data */
  sta = rspio5_FetchPatternResponseData(vi, sizeof(response),
    (ViAddr *)response, FETCH_TIMEOUT, & respByteCount);
  chk ("rspio5_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 = rspio5_DiscardData(vi, memId);
    chk ("rspio5_DiscardData");
  }
```

### 8.3.5.4 Generation of a pattern set

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

  /* 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];

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

   /* load data to stimulus RAM */
  sta = rspio5_LoadData (vi, (ViAddr)stimulus, sizeof(stimulus),
                RSPIO5_VAL_DATA_TYPE_STIM, & memId);
  chk ("rspio5_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-PIO5 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-PIO5 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) can be routed to pin TRIG of the front connector.

### 8.3.6.1 Main program

```
/* Example using low level driver functions for triggered execution */
#include <ansi_c.h>
#include "rspio5.h"

#define PATTERN_COUNT       8
#define PATTERN_PERIOD      1.0e-6

/* adapt the resource descriptor to your test system! */
static char resDesc[] = "PXI5::13::INSTR";

/* data buffer */
static RSPIO5_DATA_32BIT response[PATTERN_COUNT];
static RSPIO5_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 = rspio5_InitWithOptions(resDesc, VI_TRUE, VI_TRUE, "Simulate=0", & vi);
  /* check return value */
  chk ("rspio5_InitWithOptions");
  if (VI_SUCCESS == sta)
  {
    runTest();

    /* close the driver */
    sta = rspio5_close(vi);
    chk ("rspio5_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];

    rspio5_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;

  /* configure 32 bit wide dynamic operation without tri-state control */
  sta = rspio5_ConfigureStimMode(vi, 0xFFFFFFFF, RSPIO5_VAL_CTRL_32BIT);
  chk ("rspio5_ConfigureStimMode");

  sta = rspio5_ConfigureRespMode(vi, RSPIO5_VAL_CTRL_DYNAMIC);
  chk ("rspio5_ConfigureRespMode");

  /* configure stimulus timing */
```

```
sta = rspio5_ConfigureStimTiming(vi, 0.0, PATTERN_PERIOD,
  RSPIO5_DYN_STIM_MODE_NORMAL, PATTERN_COUNT);
chk ("rspio5_ConfigureStimTiming");

/* configure response timing */
sta = rspio5_ConfigureRespTiming(vi, PATTERN_PERIOD / 2.0,
  PATTERN_PERIOD, RSPIO5_DYN_RESP_MODE_NORMAL, PATTERN_COUNT);
chk ("rspio5_ConfigureRespTiming");

/* load data to stimulus RAM */
sta = rspio5_LoadData(vi, (ViAddr)stimulus, sizeof(stimulus),
  RSPIO5_VAL_DATA_TYPE_STIM, & memId);
chk ("rspio5_LoadData");

 /* configure trigger logic blocks to output the clock pulses  */
sta = rspio5_ConfigItTrigOut(vi,RSPIO5_IT_MASK_IT1 |RSPIO5_IT_MASK_IT2,
              RSPIO5_VAL_TRIG_IT_OUT);
chk ("rspio5_ConfigItTrigOut");

/* configure XTO to output the signal of trigger logic block 1 (stimulus) */
sta = rspio5_ConfigXTO(vi,RSPIO5_VAL_TRIG_IT1);
chk ("rspio5_ConfigXTO");

  /* general purpose trigger should output a pulse at PXI0 */
sta = rspio5_ConfigPxiTrigOut(vi,RSPIO5_TRIG_MASK_PXI0,RSPIO5_VAL_TRIG_GP,
 RSPIO5_TRIG_MASK_PXI0,RSPIO5_TRIG_MASK_PXI0);
chk ("rspio5_ConfigPxiTrigOut");

/* stimmulus and response should be triggert by PXI0 */
sta = rspio5_ConfigHWTriggerInput(vi,RSPIO5_IT_MASK_IT1 |RSPIO5_IT_MASK_IT2,
 RSPIO5_TRIG_MASK_PXI0,RSPIO5_TRIG_MASK_PXI0, RSPIO5_VAL_FALLING_EDGE);
chk ("rspio5_ConfigHWTriggerInput");

/* load the stimulus buffer for triggered pattern generation */
sta = rspio5_LoadStimBuffer (vi, memId);
chk ("rspio5_LoadStimBuffer");

/* arm trigger logic blocks */
sta = rspio5_EnableHWTrigger(vi,RSPIO5_IT_MASK_IT1 |RSPIO5_IT_MASK_IT2,
 RSPIO5_IT_MASK_IT1 |RSPIO5_IT_MASK_IT2);
chk ("rspio5_EnableHWTrigger");

/* generate the general purpose trigger pulse at PXI0 */
sta = rspio5_InitiateSWTrigger(vi,RSPIO5_IT_MASK_GP);
chk ("rspio5_InitiateSWTrigger");

/* wait until the pattern execution has finished; read the response data */
sta = rspio5_FetchPatternResponseData(vi, sizeof(response),
  (ViAddr *)response, 0.1, & respByteCount);
chk ("rspio5_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 = rspio5_DiscardData(vi, memId);
  chk ("rspio5_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.

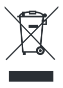


***Figure 9-1: Labeling in line with EU directive WEEE***

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 79.

## 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-PIO5 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-PIO5 module as well as a detailed block diagram.

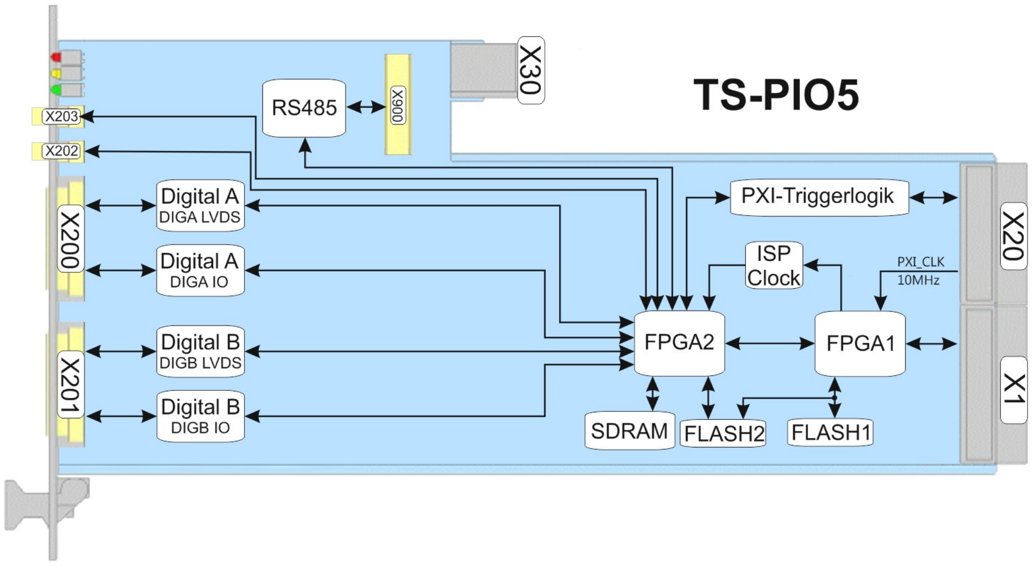


*Figure B-1: Functional block diagram of R&S TS-PIO5 module*

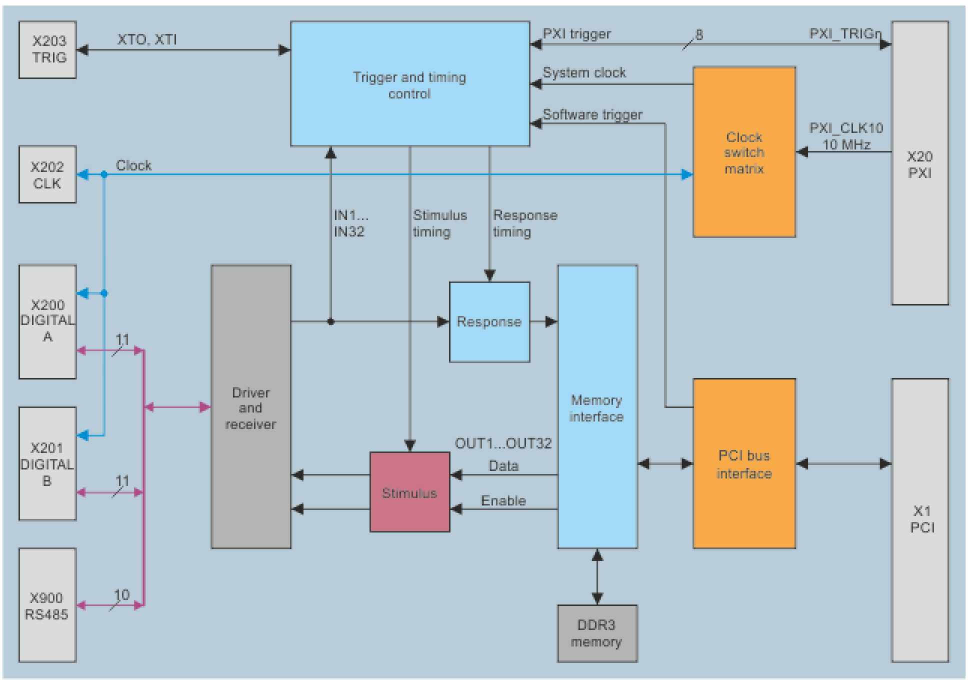


*Figure B-2: Detailed block diagram of R&S TS-PIO5 module*

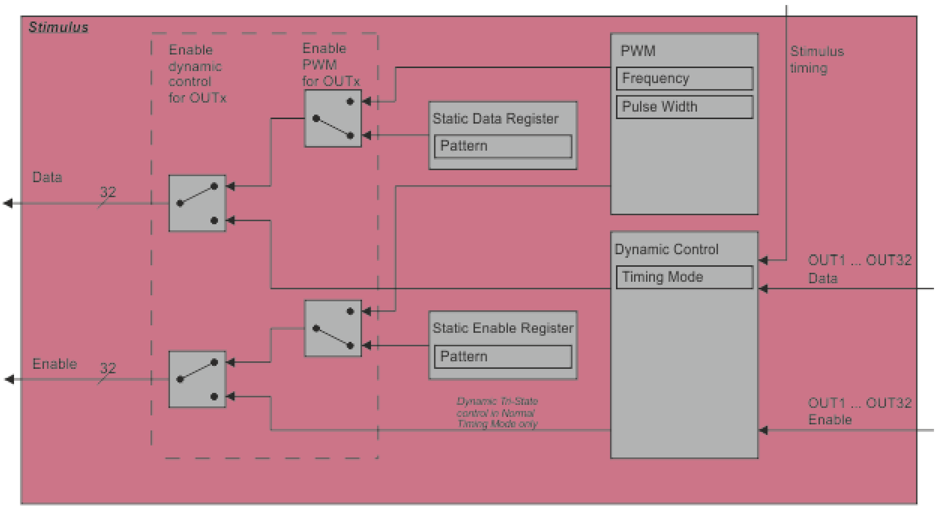


*Figure B-3: Stimulus*

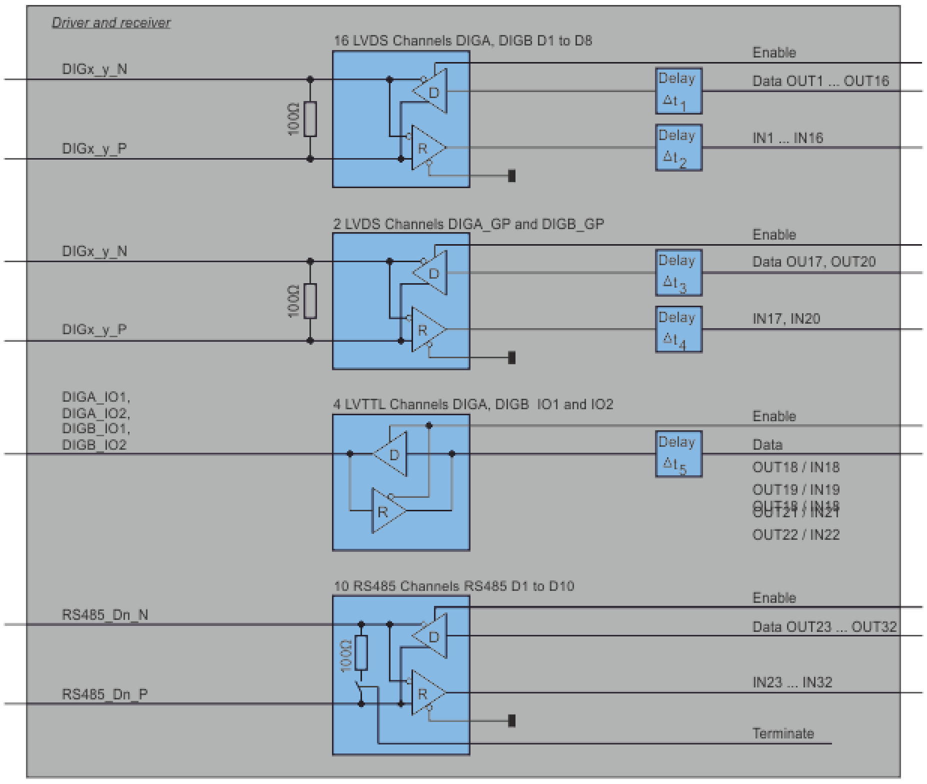


*Figure B-4: Driver and receiver*

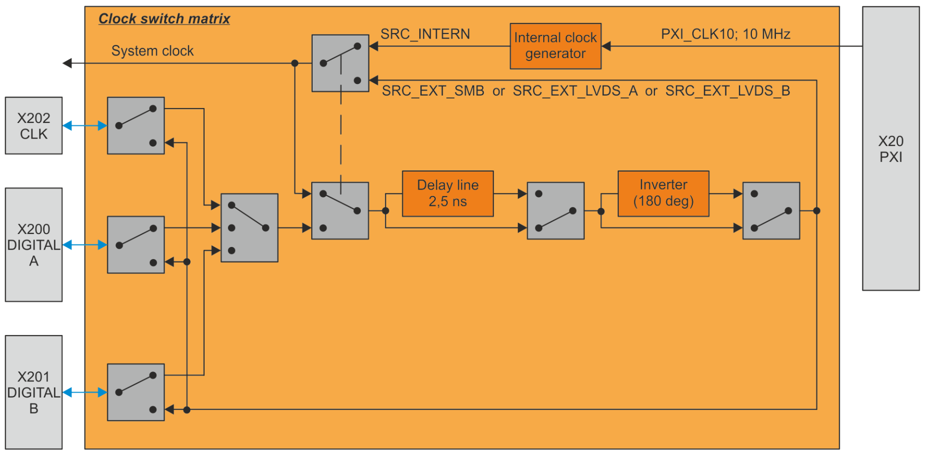


*Figure B-5: Clock switch matrix*

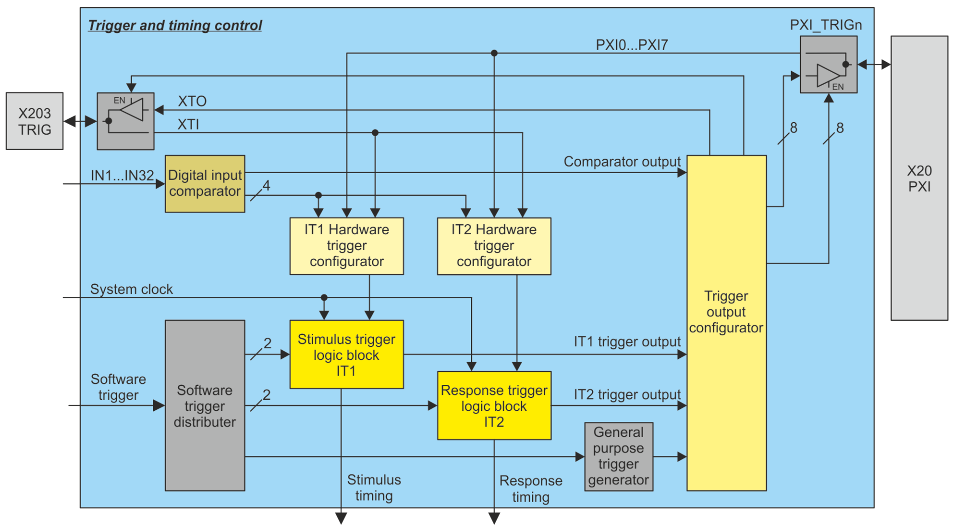

Trigger and timing control
PXI_TRIGn
PXI0...PXI7
X203 TRIG
EN
XTO
XTI
EN
8
8
X20 PXI
IN1...IN32
Digital input comparator
4
Comparator output
IT1 Hardware trigger configurator
IT2 Hardware trigger configurator
System clock
Trigger output configurator
Stimulus trigger logic block IT1
IT1 trigger output
2
Software trigger
Software trigger distributer
2
Response trigger logic block IT2
IT2 trigger output
General purpose trigger generator
Stimulus timing
Response timing

*Figure B-6: Trigger and timing control*

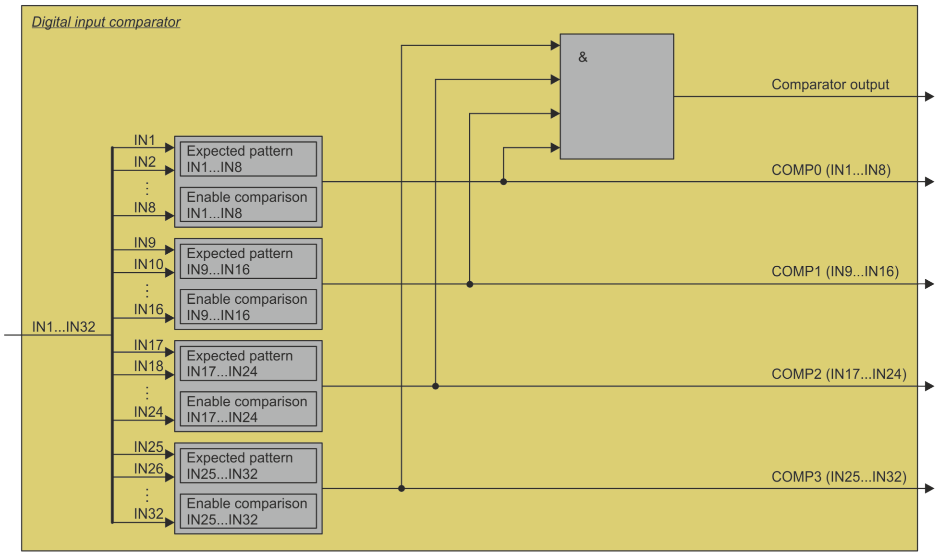

Digital input comparator
&
Comparator output
IN1
IN2
IN8
Expected pattern IN1...IN8
Enable comparison IN1...IN8
COMP0 (IN1...IN8)
IN9
IN10
IN16
Expected pattern IN9...IN16
Enable comparison IN9...IN16
COMP1 (IN9...IN16)
IN1...IN32
IN17
IN18
IN24
Expected pattern IN17...IN24
Enable comparison IN17...IN24
COMP2 (IN17...IN24)
IN25
IN26
IN32
Expected pattern IN25...IN32
Enable comparison IN25...IN32
COMP3 (IN25...IN32)

*Figure B-7: Digital input comparator*

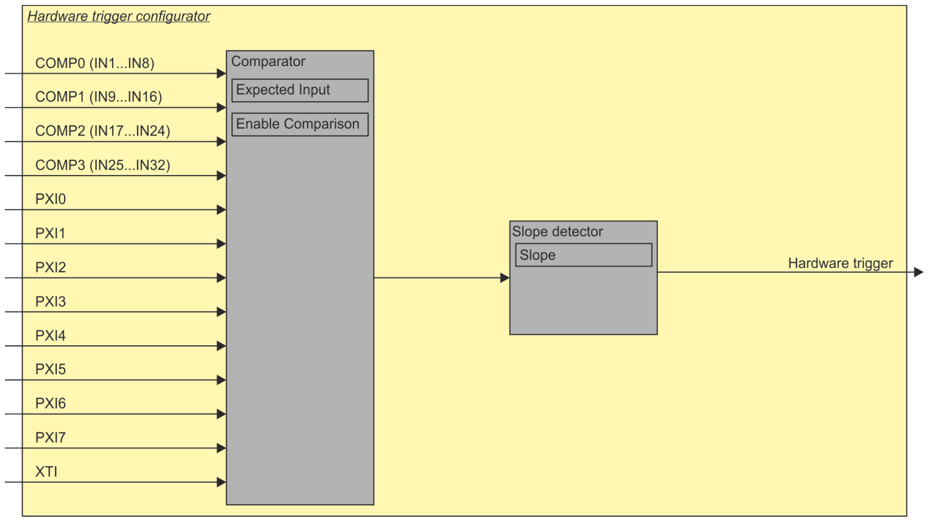


*Figure B-8: Hardware trigger configurator*

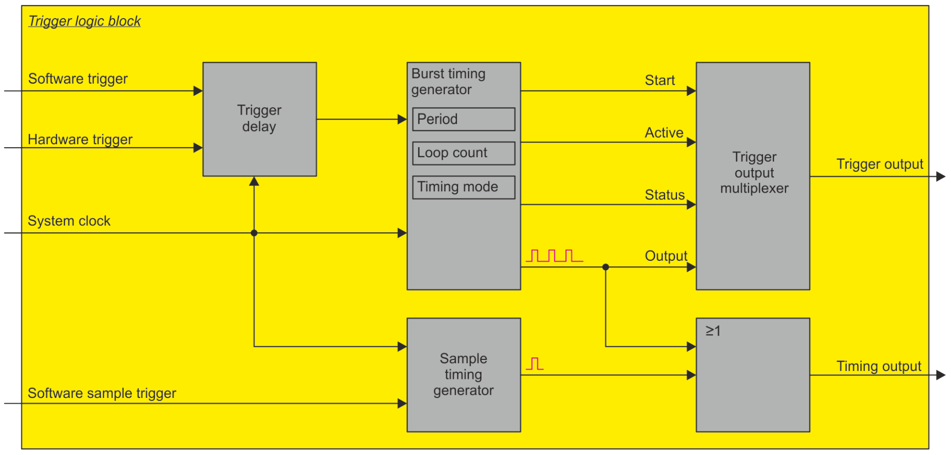


*Figure B-9: Trigger logic block*

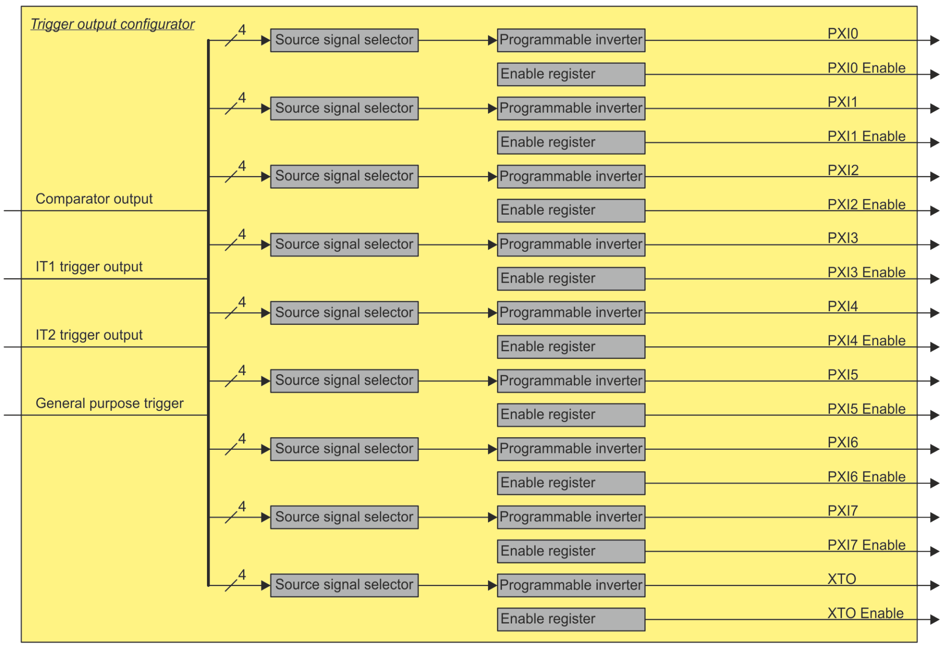


*Figure B-10: Trigger output configurator*

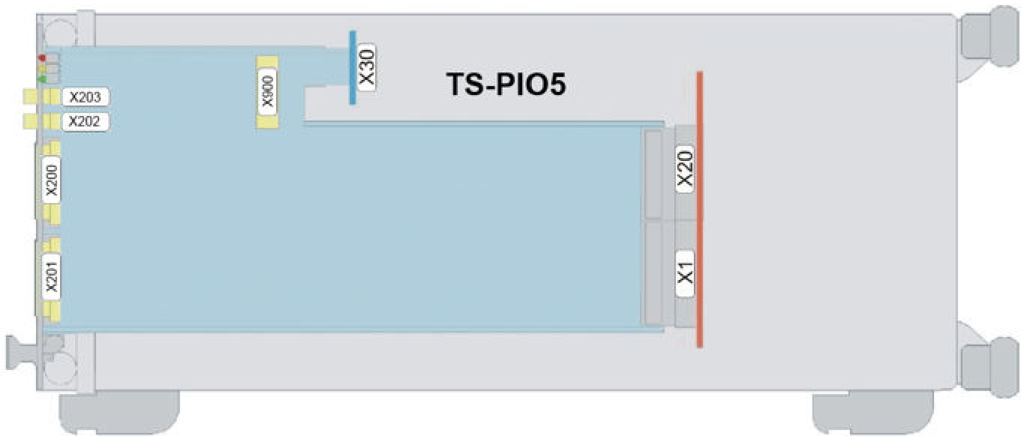


*Figure B-11: R&S TS-PIO5 modules in R&S CompactTSVP*

# C Interface description

## C.1 Connector X1

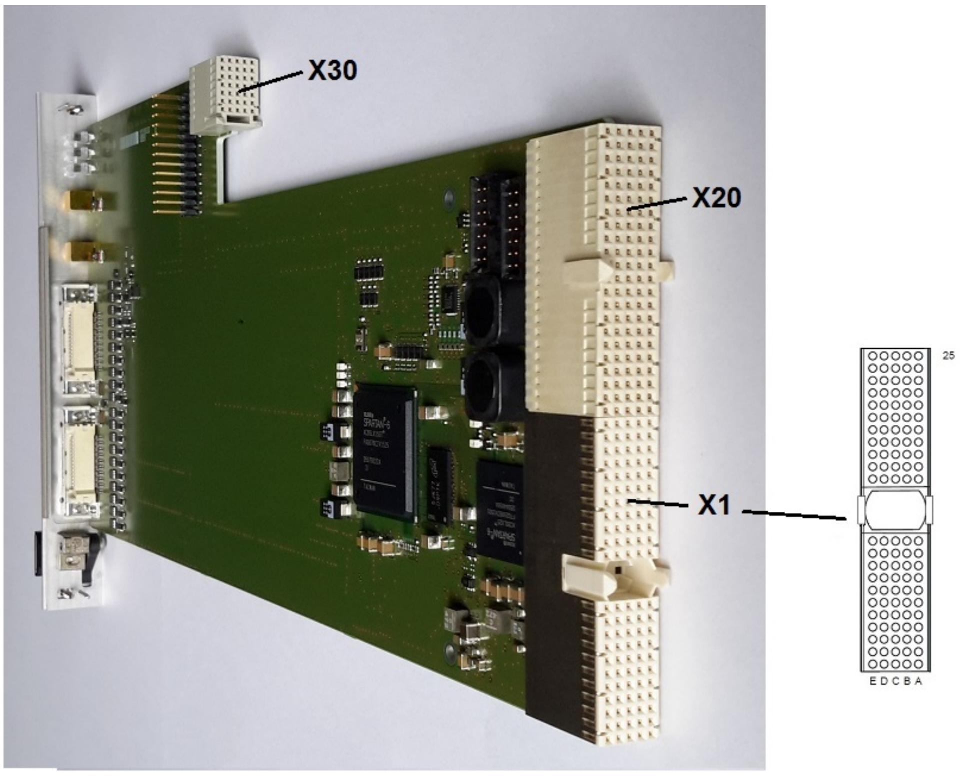
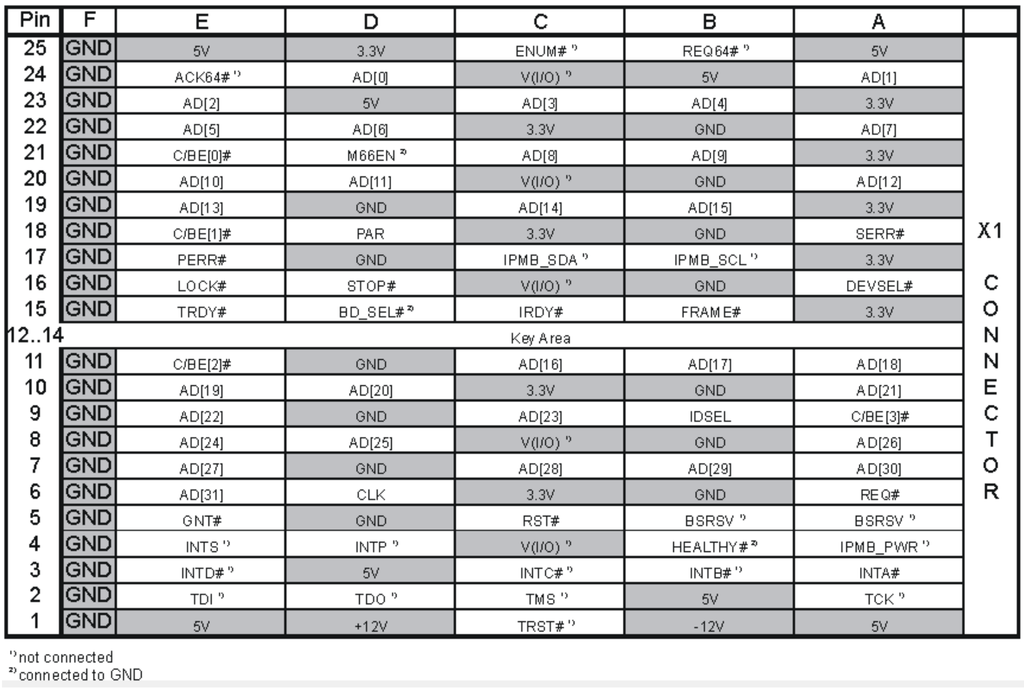


*Figure C-1: R&S TS-PIO5 connector X1*



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

[1] not connected
[2] connected to GND

*Figure C-2: Pin assignment of R&S TS-PIO5 connector X1*

## C.2 Connector X20

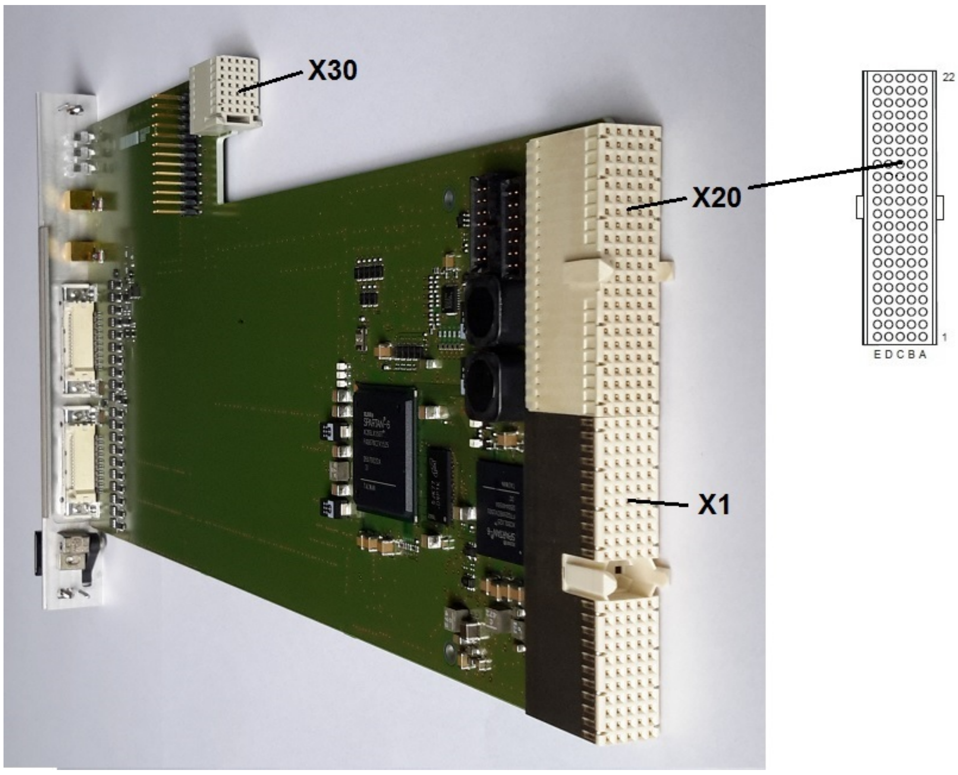


*Figure C-3: R&S TS-PIO5 connector X20*

*Table C-1: Pin assignment of R&S TS-PIO5 connector X20*

| Pin | E | D | C | B | A |
|---|---|---|---|---|---|
| **22** | GA0 | GA1 | GA2 | GA3 | GA4 |
| **21** | | | | | |
| **20** | | GND | | | |
| **19** | | | | GND | |
| **18** | PXI_TRIG6 | GND | PXI_TRIG5 | PXI_TRIG4 | PXI_TRIG3 |
| **17** | PXI_CLK10 | | | GND | PXI_TRIG2 |
| **16** | PXI_TRIG7 | GND | | | PXI_TRIG1 |
| **15** | | | | GND | |
| **14** | | | | | |
| **13** | | | | | |
| **12** | | | | | |
| **11** | | | | | |
| **10** | | | | | |

| Pin | E | D | C | B | A |
|---|---|---|---|---|---|
| 9 | | | | | |
| 8 | | | | | |
| 7 | | | | | |
| 6 | | | TDO_EXT | | |
| 5 | | | TDI_EXT | | |
| 4 | | | TMS_EXT | | |
| 3 | RSA0 | RRST# | TCK_EXT | GND | RSDO |
| 2 | | RSDI | RSA1 | RSA2 | RSCLK |
| 1 | | | | GND | RCS# |

## C.3 Connector X30

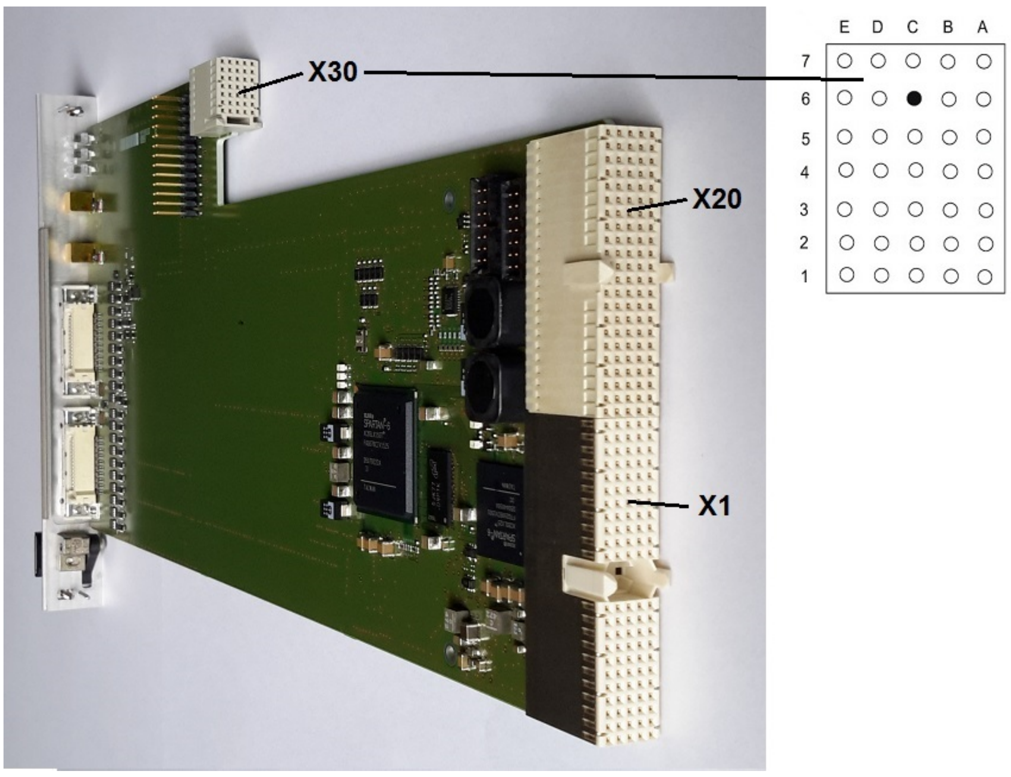


*Figure C-4: R&S TS-PIO5 connector X30*

*Table C-2: Pin assignment of R&S TS-PIO5 connector X30*

| | E | D | C | B | A |
|---|---|---|---|---|---|
| 7 | | | | | |
| 6 | | | GND | | |

| | E | D | C | B | A |
|---|---|---|---|---|---|
| 5 | | | | | |
| 4 | | | | | |
| 3 | | | | | |
| 2 | | | | | |
| 1 | | | | | |

## C.4 Connector X200 (Digital A)

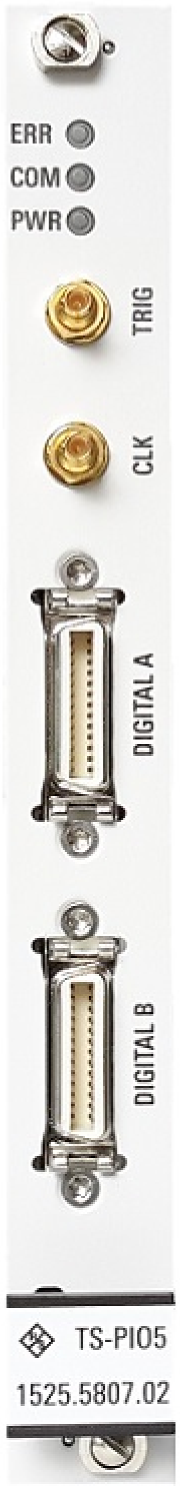


*Figure C-5: R&S TS-PIO5 connector X200 (MDR-26 Position)*

*Table C-3: Pin assignment of R&S TS-PIO5 connector X200*

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | DIGA_CLKIN_P | 14 | DIGA_CLKIN_N |
| 2 | AGND [2)] | 15 | DIGA_D1_N |
| 3 | DIGA_D1_P | 16 | DIGA_D2_N |
| 4 | DIGA_D2_P | 17 | DIGA_D3_N |
| 5 | DIGA_D3_P | 18 | DIGA_GP_N |
| 6 | DIGA_GP_P | 19 | GND [3)] |
| 7 | DIGA_IO1 | 20 | DIGA_IO2 |
| 8 | +5 V [1)] | 21 | DIGA_D4_N |
| 9 | DIGA_D4_P | 22 | DIGA_D5_N |
| 10 | DIGA_D5_P | 23 | DIGA_D6_N |
| 11 | DIGA_D6_P | 24 | DIGA_D7_N |
| 12 | DIGA_D7_P | 25 | DIGA_D8_N |
| 13 | DIGA_D8_P | 26 | AGND [2)] |

[1)] +5 V / max. 0.3 A supply voltage for an external adapter box

[2)] AGND connection to module ground via choke

[3)] GND connection to module ground via choke

The following R&S article is supported as connecting cable:

1415.0201.02 SMU-Z6 CABLE TVR290 (2 m long).

## C.5 Connector X201 (Digital B)

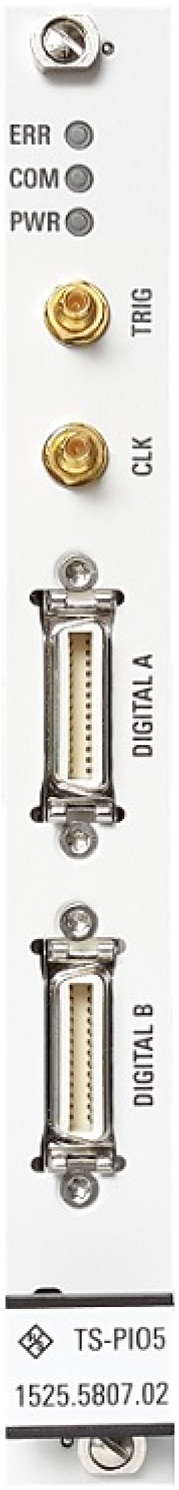


*Figure C-6: R&S TS-PIO5 connector X201 (MDR-26 Position)*

*Table C-4: Pin assignment of R&S TS-PIO5 connector X201*

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| **1** | DIGB_CLKIN_P | **14** | DIGB_CLKIN_N |
| **2** | AGND [2)] | **15** | DIGB_D1_N |
| **3** | DIGB_D1_P | **16** | DIGB_D2_N |
| **4** | DIGB_D2_P | **17** | DIGB_D3_N |
| **5** | DIGB_D3_P | **18** | DIGB_GP_N |
| **6** | DIGB_GP_P | **19** | GND [3)] |
| **7** | DIGB_IO1 | **20** | DIGB_IO2 |
| **8** | +5 V [1)] | **21** | DIGB_D4_N |
| **9** | DIGB_D4_P | **22** | DIGB_D5_N |
| **10** | DIGB_D5_P | **23** | DIGB_D6_N |
| **11** | DIGB_D6_P | **24** | DIGB_D7_N |
| **12** | DIGB_D7_P | **25** | DIGB_D8_N |
| **13** | DIGB_D8_P | **26** | AGND [2)] |

[1)] +5 V / max. 0.3 A supply voltage for an external adapter box

[2)] AGND connection to module ground via choke

[3)] GND connection to module ground via choke

The following R&S article is supported as connecting cable:

1415.0201.02 SMU-Z6 CABLE TVR290 (2 m long).

## C.6 Connector X202 (CLK)

50 ohm SMB device plug for clock signals. Output level 3.3 V. As an input, this port is 5 V tolerant. Valid frequency range 20 MHz to 100 MHz.

## C.7 Connector X203 (TRIG)

50 ohm SMB device plug for trigger signals. Output level 3.3 V. As an input, this port is 5 V tolerant. Maximum input frequency 20 MHz.

## C.8 Connector X900 (RS485)

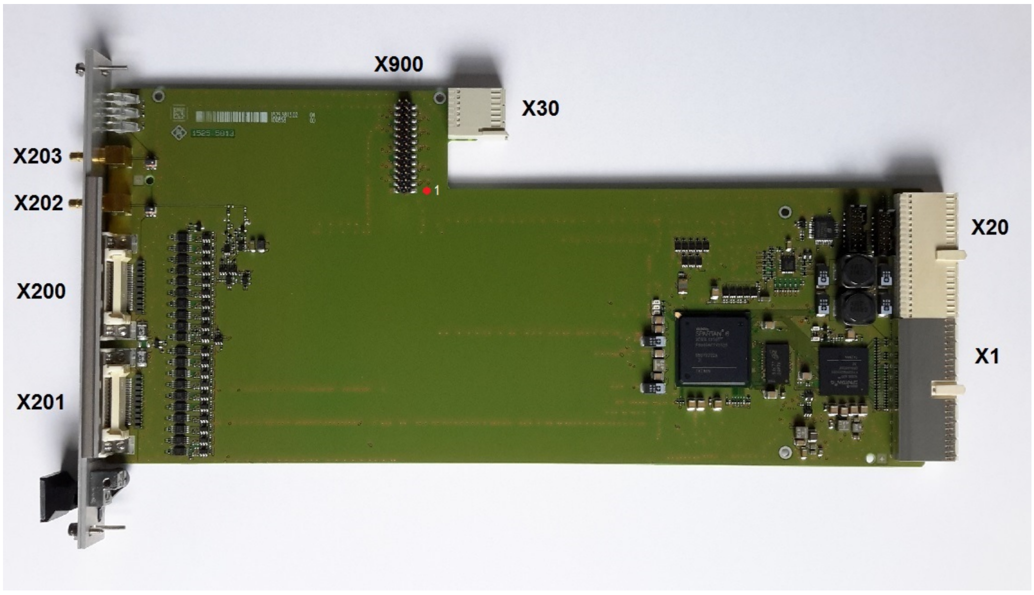


*Figure C-7: R&S TS-PIO5 connector X900*

*Table C-5: Pin assignment of R&S TS-PIO5 connector X900*

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 26 | GND | 25 | GND |
| 24 | RS485_D10_N | 23 | RS485_D10_P |
| 22 | RS485_D9_N | 21 | RS485_D9_P |
| 20 | RS485_D8_N | 19 | RS485_D8_P |
| 18 | RS485_D7_N | 17 | RS485_D7_P |
| 16 | RS485_D6_N | 15 | RS485_D6_P |
| 14 | GND | 13 | GND |
| 12 | RS485_D5_N | 11 | RS485_D5_P |
| 10 | RS485_D4_N | 9 | RS485_D4_P |
| 8 | RS485_D3_N | 7 | RS485_D3_P |
| 6 | RS485_D2_N | 5 | RS485_D2_P |
| 4 | RS485_D1_N | 3 | RS485_D1_P |
| 2 | GND | 1 | GND |