# GBT-SCA

## THE SLOW CONTROL ADAPTER ASIC FOR THE GBT SYSTEM

## USER MANUAL

Design team:

Sandro Bonacini, Alessandro Caratelli, Rui Francisco, Kostas Kloukinas,
Alessandro Marchioro, Paulo Moreira, Christian Paillard

User manual:

Alessandro Caratelli, Daniel H. Montesinos, Kostas Kloukinas

CERN EP-ESE-ME

**V. 2019-002**

## CONTACTS

| | |
|---|---|
| **GBT-SCA ASIC orders:** | asic.distribution@cern.ch |
| **Information and support:** | GBTX-support@cern.ch |
| **Additional direct contacts:** | alessandro.caratelli@cern.ch<br>kostas.kloukinas@cern.ch<br>dahernan@cern.ch |

## MANUAL MODIFICATIONS

https://espace.cern.ch/GBT-Project/VLDB/Manuals/GBT-SCA_Manual_corrections_history.pdf

# TABLE OF CONTENTS

**CONTACTS** ... **2**

**MANUAL MODIFICATIONS** ... **2**

**TABLE OF CONTENTS** ... **3**

**1. INTRODUCTION** ... **5**

**2. SCA OVERVIEW** ... **6**

**3. SCA ARCHITECTURE** ... **8**

**4. COMMUNICATION** ... **12**

4.1. THE E-LINK LAYER ... 13
4.2. THE SCA CHANNEL COMMAND PROTOCOL LAYER ... 16

**5. THE SCA CONTROLLER CONFIGURATION** ... **19**

5.1. CONTROL REGISTERS ... 19
5.2. COMMANDS ... 21

**6. I$^2$C CHANNELS** ... **22**

6.1. CONFIGURATION REGISTERS ... 22
6.2. START OF TRANSMISSION COMMANDS ... 25
6.3. COMMANDS SUMMARY ... 28
6.4. IO PADS ... 29

**7. SPI CHANNEL** ... **30**

7.1. CONFIGURATION REGISTERS ... 30
7.2. START OF TRANSMISSION ... 33
7.3. COMMAND TABLE ... 33
7.4. IO PADS ... 34

**8. JTAG CHANNEL** ... **35**

8.1. CONFIGURATION REGISTERS ... 35
8.2. TRANSMISSION COMMANDS ... 38
8.3. COMMANDS LIST ... 39
8.4. IO PADS ... 40

**9. PARALLEL INTERFACE (GPIO)** ... **41**

9.1. CONFIGURATION REGISTERS 42
9.2. OPERATION DESCRIPTION 44
9.3. COMMANDS TABLE 45
9.4. IO PADS 46

**10. DAC CHANNEL 47**

10.1. REGISTERS 47
10.2. OPERATION DESCRIPTION 47
10.3. COMMANDS TABLE 49
10.4. IO PADS 49

**11. ADC CHANNEL 50**

11.1. CONFIGURATION REGISTERS 52
11.2. FUNCTIONAL DESCRIPTION 53
11.3. COMMANDS TABLE 54
11.4. INTERNAL TEMPERATURE SENSOR 54
11.5. ILINEARITY MEASUREMENTS 55
11.6. IO PADS 56

**12. ELECTRICAL SPECIFICATIONS 57**

12.1. POWER SUPPLIES 57
12.2. DIGITAL IO PAD CHARACTERISTICS 58
12.3. POWER CONSUMPTION 58
12.4. DECOUPLING 58
12.5. FRONT-END SPECIFICATIONS 59

**13. PACKAGE DESCRIPTION 60**

13.1. MECHANICAL CHARACTERISTICS 60
13.2. PINOUT TOP VIEW 61
13.3. PINOUT BOTTOM VIEW 62
13.4. SOLDERING PROFILE 63
13.5. PINS LIST 63
13.6. SCA PINS RECOMMENDATIONS 66

# 1. INTRODUCTION

The future upgrades of the LHC experiments will increase the beam luminosity leading to a corresponding growth of the amounts of data to be treated by the data acquisition systems. To address these needs, the GBT (Giga-Bit Transceiver optical link) architecture was developed to provide the simultaneous transfer of readout data, timing and trigger signals as well as slow control and monitoring data. The GBT-SCA ASIC, part of the GBT chip-set, has the purpose to distribute control and monitoring signals to the on-detector front-end electronics and perform monitoring operations of detector environmental parameters. In order to meet the requirements of different front-end ASICs used in the experiments, it provides various user-configurable interfaces capable to perform simultaneous operations. It is designed employing radiation tolerant design techniques to ensure robustness against SEUs and TID radiation effects and is implemented in a commercial 130 nm CMOS technology. The Slow Control Adapter (SCA) chip is designed to work in parallel with to the GBT optical link bidirectional transceiver system of which it extends the functionality. This document focuses on the user-visible aspects of the component such as its logical and electrical interfaces, programming features and operating modes. It also includes detailed descriptions and specifications of the chip pin-out and electrical characteristics. To better understand the use of the GBT-SCA in the GBT system, a brief explanation of a GBT system is provided in the next section.

For SCA-V2 ASIC Samples, please contact: asic.distribution@cern.ch

**For references, please use the following publication:**

Caratelli, A., Bonacini, S., Kloukinas, K., Marchioro, A., Moreira, P., De Oliveira, R. and Paillard, C., 2015. The GBT-SCA, a radiation tolerant ASIC for detector control and monitoring applications in HEP experiments. *Journal of Instrumentation, 10(03),* p.C03034.

**doi:** 10.1088/1748-0221/10/03/C03034

# 2. SCA OVERVIEW

The GBT (Giga-Bit Transceiver) system was developed with the purpose to provide on a unique optical link the simultaneous transfer of the three types of information required by the High Energy Physics experiments:

- Readout data (DAQ).
- Timing and trigger information (clock and trigger decisions).
- Detector control and monitoring information.

Figure 2.1 depicts the generic topology of the GBT system while Figure 2.2 shows a typical implementation of on-detector electronics making use of the GBT-chipset.

The GBT-SCA ASIC (Giga-Bit Transceiver - Slow Control Adapter) is an integrated circuit built in a commercial 130 nm CMOS technology and is the part of the GBT chipset which purpose is to distribute control and monitoring signals to the front-end electronics embedded in the detectors. It connects to a dedicated electrical port on the GBTX ASICs through an 80 Mbps dual redundant bidirectional data-link, namely the e-links. The GBT communication is transparent to the slow control protocol. The GBT encodes slow control packets in the counting room, carries it on the optic fibers interlaced with the rest of the traffic, and it delivers the SCA packets unmodified to the GBT-SCA.

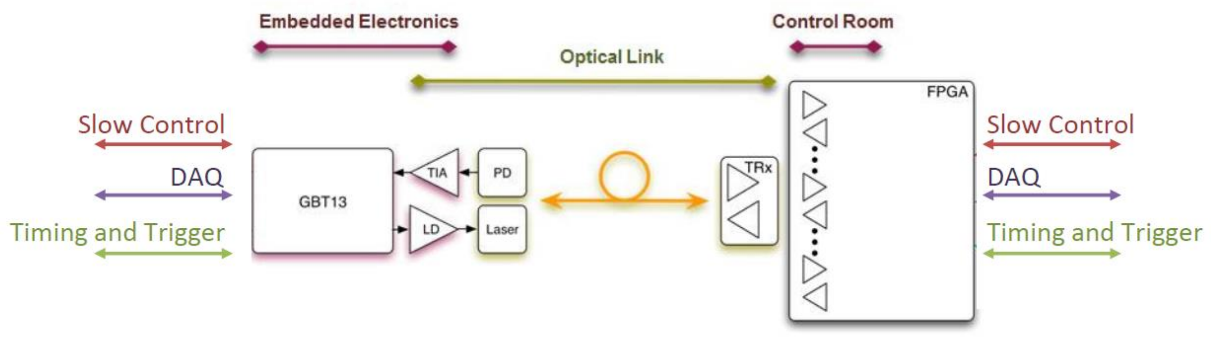


Figure 2.1: Generic topology of the GBT system

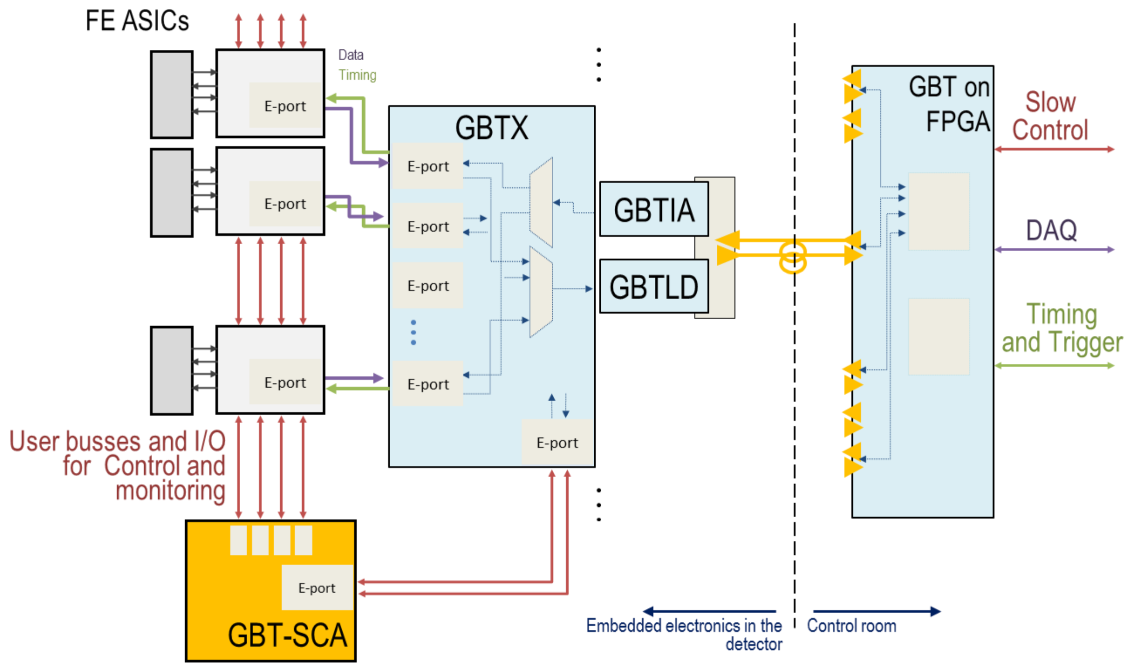


Figure 2.2: Typical implementation of on-detector electronics making use of the GBT-chipset

The SCA represents the embedded node of the system that is responsible to translate the unified packets sent by the control room and redirect to the selected peripheral, through one of the physical ports. In order to meet the requirements of different front-end ASIC in various experiments, the SCA provides a number of user-configurable electrical interface ports, able to perform concurrent data transfer operations. The user interface ports are: 1 SPI master, 16 independent I$^2$C masters, 1 JTAG master and 32 general-purpose IO signals with individual programmable direction and interrupt generation functionality. It also includes 31 analog inputs multiplexed to a 12 bit ADC featuring offset calibration and gain correction as well as four analog output ports controlled by four independent 8-bits DACs.

# 3. SCA ARCHITECTURE

The architecture of the SCA ASIC is shown in Figure 3.1. The SCA is broadly composed of two e-link ports that connect to the GBTX ASICs, a set of user interface ports to connect with the on-detector electronics and a network controller that routes the information between the e-links and the user interfaces.

Typically, the SCA ASIC connects via an e-link to the special purpose slow control e-port of the GBTX ASIC. This dedicated e-port runs at 40MHz double date rate (DDR) mode giving an effective data rate of 80 Mbps. It is also possible to connect the SCA ASIC to any of the other GBTX e-ports as long as its data transfer mode is properly configured for 40MHz DDR operation. This feature permits the scalability of the slow control system and effectively allows the implementation of front-end topologies where a GBT link could be used for slow control only operations.

The disposal of two, functionally identical, e-link ports on the SCA facilitates the implementation of redundancy schemes anticipating failures on the optical links. A possible redundancy scheme can be two GBTX ASICs connected to the same SCA ASIC. In this scheme only one of the e-ports is active at any moment.

The active port is also the source of the 40MHz system clock that synchronizes the SCA internal state machines. The inactive port is properly muted and any activity on the clock or data lines is discarded. Switching over between e-ports is performed on user's demand by issuing a "CONNECT" command which is specially foreseen in the high-level communication protocol as described in Communication 4. One of the e-ports is considered as primary and the other as secondary. On power-up the primary e-port is automatically selected for operation.

Both e-ports communicate with the Network Controller block via an Atlantic interface parallel bus. The Network Controller connects further with all the interface channels via a common Wishbone bus [5]. The interface channels are circuit blocks that implement the functionalities of the user interface ports. The interface channels can operate independently and concurrently. As described in section 3, the SCA uses a packet oriented communication protocol. The Network

Controller block implements the functionalities of routing the data packets to and from the interface channels as well as supervising the operation of the interface channels. The channels can demand attention to transmit data at any time asserting an interrupt line on the internal Wishbone interconnect fabric. A Wishbone bus arbiter using the round robin technique handles the interrupts.

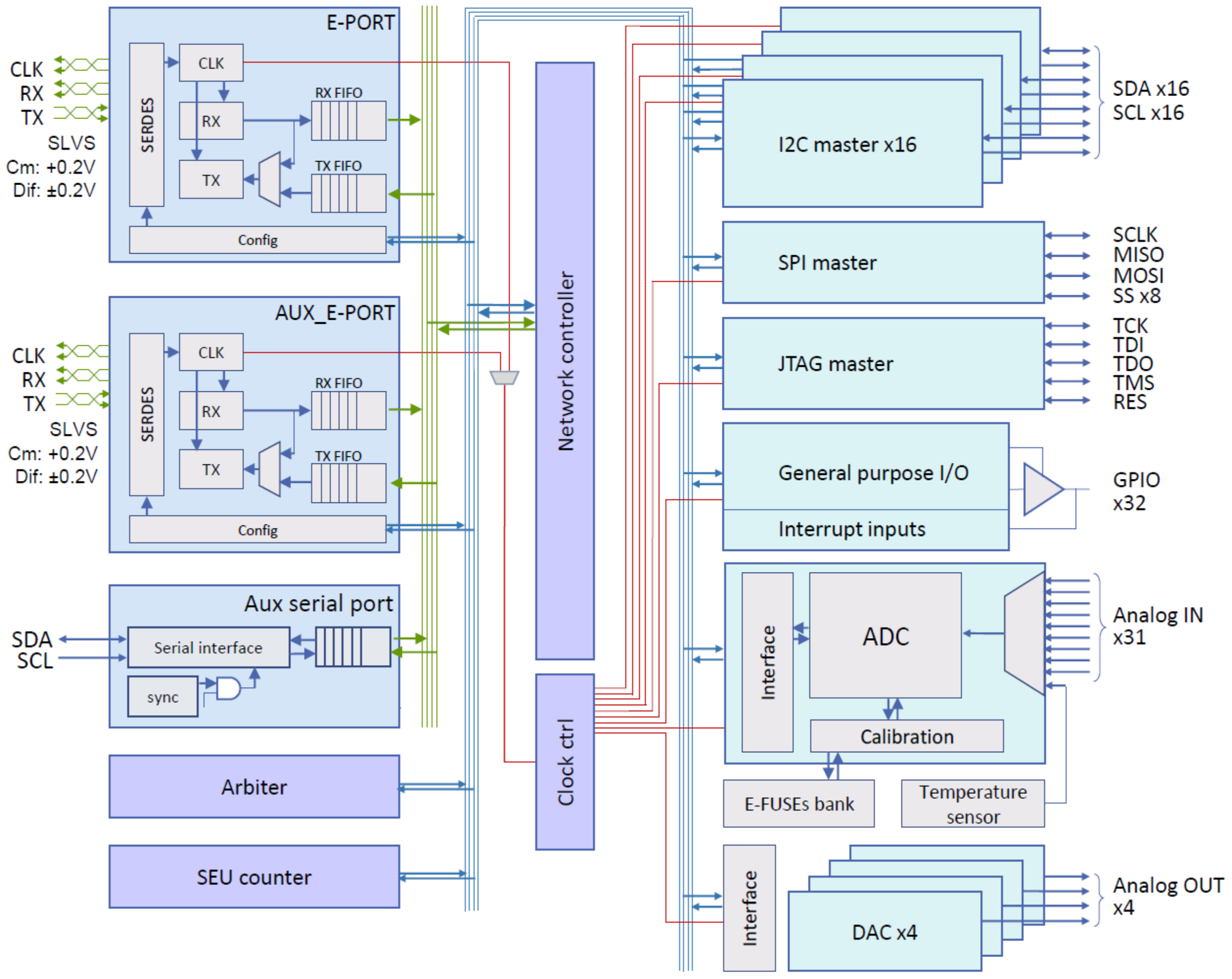


Figure 3.1: GBT-SCA block diagram

An auxiliary I2C port is attached on the internal Atlantic interface bus, bypassing the e-link ports, and can be used for debugging purposes.

The SCA ASIC integrates the following interface channels:

- 1 Parallel Interface Adapter (GPIO) channel featuring 32 General Purpose digital IO lines. Each line can be individually programmed as input or output or in a tri-state mode. Input signals are sampled and registered at the rising or falling edges of the system clock or of an external strobe signal from the user’s application connected on a dedicated input line. Any line configured as input can be programmed to generate an interrupt request to the control room electronics. The electrical levels on all digital IO lines are 0 - 1.5V.

- 16 independent I$^2$C master serial bus channels. The I$^2$C channels feature individually programmable data transfer rates from 100 kHz to 1 MHz and can generate both 7-bit and 10-bit address as well as single-byte and multi-byte I$^2$C bus transactions. They can also perform Read, Write and Read/Modify/Write transactions on the I$^2$C bus. The transactions are initiated by the reception of a user command and executed locally by the channel's state machines. Upon completion a return packet is generated containing user data and status flags. These channels can be individually disabled to reduce power consumption in periods of inactivity.

- 1 SPI serial bus master channel with 8 individual slave select lines. The Serial Peripheral Interface (SPI) channel implements a full duplex synchronous serial bus master with a single transaction length of up to 128 bits and a programmable transfer rate up to 20 MHz. It supports all the standard SPI bus operating modes: 00, 01, 10 and 11. It also integrates 8 independent slave-select lines. The bus frequency spans from 156 kHz up to 20 MHz in 128 user programmable steps. The SPI channel is implemented around a 128-bit shift register that serializes and de-serializes the bit-streams between the MISO and MOSI SPI lines and the internal parallel bus. The SPI channel is protocol agnostic. The user specific protocol is implemented in FPGA circuitry residing at the control room electronics. The SPI channel can be powered down to conserve power.

- 1 JTAG serial bus master channel. The JTAG channel can perform bus transactions of up to 128-bit length. Longer transactions are also possible by segmenting them and having them executed on consecutive channel commands. The interface implements an asynchronous reset line of configurable pulse width. The bus frequency spans from 156 kHz up to 20 MHz in 128 user programmable steps. The JTAG channel is implemented around two 128-bit shift register that serializes and deserializes the bit-streams between the TMS, TDO and TDI lines and the internal parallel bus. The JTAG channel in the SCA does not implement a JTAG master state machine. The JTAG bus cycles will be generated by the FPGA circuitry residing at the control room electronics. The SPI channel can be powered down to conserve power.

- 1 ADC channel with 31 multiplexed analog inputs. The ADC channel block consists of a 32 input analog multiplexer connected to a 12-bit analog to digital converter (ADC). One analog input is internally connected to the embedded temperature sensor while the remaining 31 inputs are available to the user. All inputs feature a switchable 100 uA current source to facilitate the use of externally connected resistance temperature sensors (RTD). The ADC adopts a single-slope Wilkinson architecture. This architecture features circuit simplicity and low power consumption. The long conversation time associated with this architecture is perfectly compatible with the conversion requirements of slow varying parameters like detector leakage current and temperature, power supply voltages etc. The analog input range is 0.0 V to 1.0 V, the maximum conversion rate is 3.5 KHz and the maximum quantization error is 1 LSB. The ADC block features automatic offset cancellation and gain correction circuitry. The comparator offset is removed by performing an offset evaluation cycle before any conversion. The gain error is corrected by multiplying the converted value with a gain calibration coefficient. The gain calibration coefficient is evaluated during production phase for every chip and stored on the on-chip e-fuse bank. The stored coefficient can be overridden by the user to compensate for any possible drifts caused by the radiation environment in the field application.

- 4 DAC channels. There are four independent digital to analog converter (DAC) featuring 8-bit resolution and capable to generate voltage signals in the range of 0.0 V to 1.0 V*.

*_Refer to 10.DAC channel, page 47._

# 4. COMMUNICATION

The SCA implements two independent e-ports to connect via the GBT to the back-end. The two ports are functionally identical and independent. The active one is selected via the CONNECT command as described in Section 4.1. Only the active port provide clock and data to the SCA core. Data and clock received on the inactive port is discarded. Figure 4.1 describe the e-port connectivity in the SCA ASIC.

The slow control system is organized in a point-to-point network topology where a fixed bandwidth of 80 Mbps is allocated by the GBT system for slow control functions. GBT-link Layer connects the GBT ASIC to the Control Room electronics via a point-to-point, bi-directional, 4.8Gbps, optical link using a special, SEU robust protocol. The transport protocol is transparent to the user data. The GBT uses a fixed packet length of 120 bits at 4.8 Gbps transmission rate. A fixed bandwidth of 80 Mbps is allocated for slow control purposes. The communication architecture adopted by the SCA is based on two protocol layers as shown in Figure 4.3:

- The e-link transport protocol
- The SCA channel command protocol.

The communication interfaces of the SCA (referred in this manual as channels) are seen from the control computer as remote independent destination of messages, each one with a particular set of control registers and/or allocated memory locations.

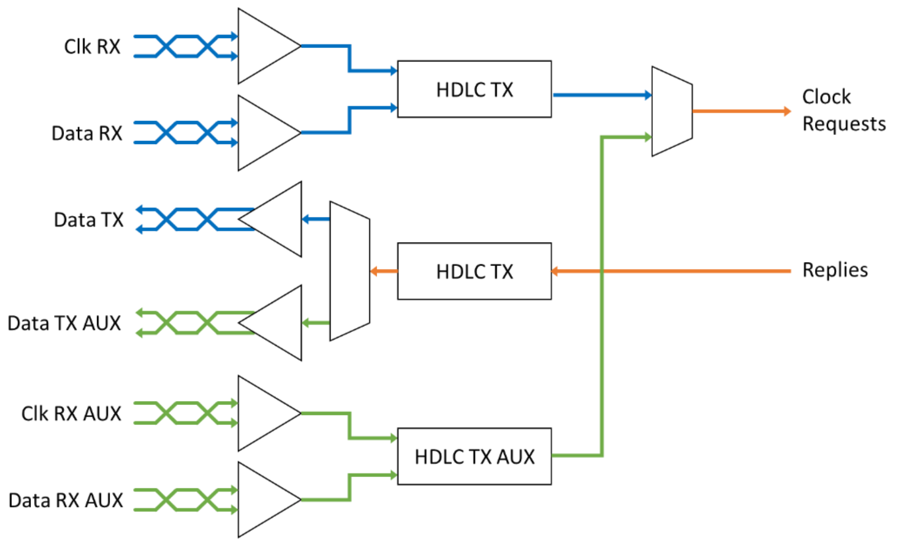
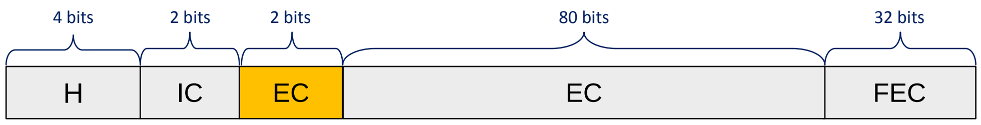


Figure 4.1: Primary and Auxiliary e-link port connectivity in the SCA



- Trigger path: 640 Mbps
- Control path: 160 Mbps
  - 1 internal e-link (for GBT management)
  - 1 external e-link (for GBT-SCA chip) 40 MHz DDR (80 Mbps)
- Data path: 2.56 Gbps
  - 10 e-links at 320 Mbps | 20 e-links at 160 Mbps | 40 e-links at 80 Mbps

Figure 4.2: GBT Frame

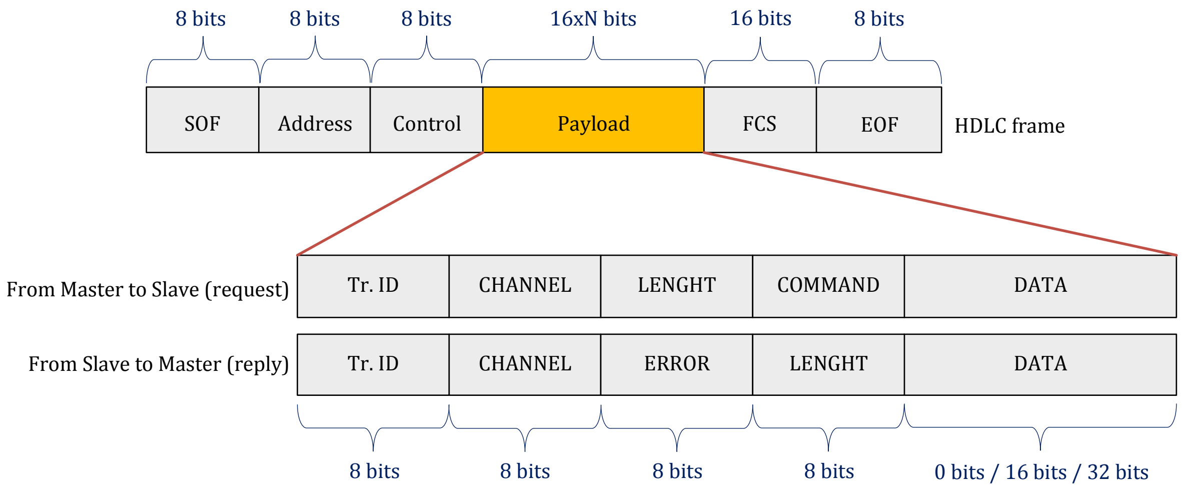


Figure 4.3: SCA Communication protocol - (a) e-Link protocol layer - (b) SCA channel protocol layer

The channels operate independently from each other in order to allow concurrent transactions and perform concurrent transfers to their end-devices. To decouple the operation on the channel ports with respect to the one of the GBT link, all operations on the channels are asynchronous and do not demand an immediate response. All upwards packets are acknowledged via either status or data words depending on the command type.

## 4.1. THE E-LINK LAYER

The e-link port on the SCA ASIC implements a packet oriented full duplex transmission protocol based on the HDLC standard (ISO/IEC 13239:2002) allowing full duplex communication with non-deterministic link latency.

An 8-bit frame delimiter flag (binary 01111110) is provided in the standard. The frame delimiter is composed of six consecutive ‘1s’. The protocol assures that this combination is not found anywhere else in the data bit-stream by any inserting a ‘0’ in any sequence of five consecutive ‘1’ at the transmitter side and by stripping off this trailing ‘0’ at the receiver side. When the receiver detects the sequence "11111" in the data, it removes the "0" added by the transmitter.

Any sequence of more than 6 ones is considered to be frame abort or channel idle signaling and resets the receiver PHY state machine. When idle, the transmitter PHY sends repeatedly a fill-frame sequence composed of 7 ones followed by a zero.

When no information is exchanged, an idle packet is transmitted by the master interface and discarded at the receiver. This packet is a single byte long.

The structure of the HDLC data packet is shown in Figure 4.3. It consists of a frame delimiter character (SOF), an 8-bit address field, an 8-bit control field, a variable length data payload field and a 16-bit frame checksum field (FCS).

- → **The HDLC protocol transmits data in frames of 16 bits.**
- → **Bits within the frame are transmitted from the least significant bit (LSB) to the most significant bit (MSB). For this reason, considering the actual bit-stream as divided in bytes, the transmitted bytes will look like swapped two by two.**

Figure 4.4 shows the HDLC packet payload in the actual transmission order:

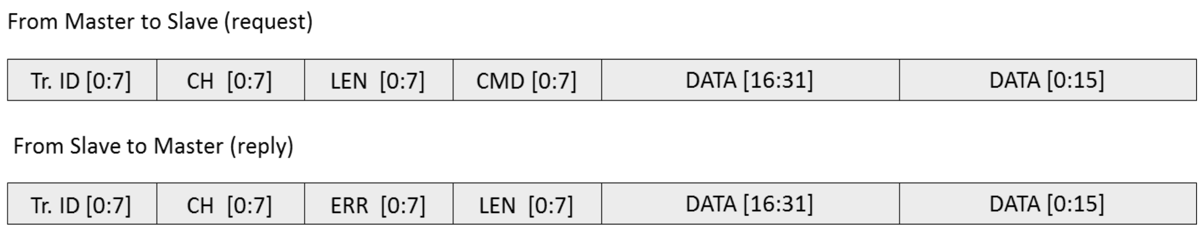


Figure 4.4: HDLC payload bit transmission order

## SOF/EOF:

The frame delimiter field marks the start and end of the frame and contains a pattern composed by six consecutive ‘1’ (binary 01111110, ex 0x7E) to which the receiver PHY can synchronize. Each frame begins and ends with a frame delimiter. A frame delimiter at the end of a frame may also mark the start of the next frame. A sequence of 7 or more consecutive 1-bits within a frame will cause the frame to be aborted. If the frame delimiter sequence is received during a transaction, the communication will abort and the packet discarded. A new ‘start of frame’ needs to be sent to transmit a new message.

## ADDRESS:

It represents the packet destination address. The address is one-byte long. By default, the GBT-SCA use address 0x00.

## CONTROL:

The control field is 1 byte in length and contains frame sequence numbers of the currently transmitted frame and the last correctly received frame. It implements in this way an acknowledgement handshake mechanism between the SCA and the control room electronics. All transmitted HDLC frames require the reception of HDLC response frames with positive acknowledgment. The packets are numbered in the CONTROL field from 0 to 7. There can be a maximum of eight uniquely numbered packets. The master shall wait for the acknowledgement of sent packets before sending new ones. The SCA receiver checks every received packet number against its internal last correctly received packet number and flags a SREJ command in case packets are missing.

The control field is also used to convey three supervisory level commands:

- CONNECT / SABM:

  The Set Asynchronous Balanced Mode command, in this manual referred as CONNECT command, instructs the GBT-SCA about which port (primary or auxiliary) is active. The SCA e-port which receive the command will be the active one while the other one is disabled. Any data packet received on the disabled e-port is discarded with the only exception of packet carrying supervisor level commands: connect, reset and test. The GBT-SCA system clock is provided by the currently active e-port interface. It is therefore suggested to send a ‘reset’ request after switching between e-ports.

- RESET:

  The RESET command resets the SCA e-ports, its internal FIFOs and all the state machines. The e-port interface which receive the RESET command become the active one while the other interface get disconnected implying therefore the same behavior of the CONNECT command.

- TEST:

  The TEST command sets the SCA e-port in loopback mode to facilitate the link verification and debugging operations in the field application. Only the e-link interface in the SCA ASIC is part of the loop-back path. In particular, the TEST command only test the connectivity with the SCA, not his functionality.

Upon reception of the reset, connect and test commands, the slave (SCA) generates a command acknowledge response. In general, it is safer to wait for the response before sending the next command.

### PAYLOAD:

The payload field length is a multiple of 16 bits and depends on the packet structure. The payload is not present in case of a supervision level commands. It carries the GBT-SCA message oriented higher lever protocol described in next section.

### FCS:

A Frame Check Sequence (FCS) field is calculated over the address, control and information field using the CCITT standard 16-bit CRC.

The FCS field is used to detect transmission errors and is 2 bytes in length. The value of the FCS field is calculated over the address, control and information fields using the CCITT standard 16-bit Cyclic Redundancy Check (CRC) defined as: $G(x) = x^{16} + x^{12} + x^5 + 1$

CRC-failing packets are dropped and therefore treated as missing packets. The SREJ command (selective-reject) is sent from the GBT-SCA to the master and contains the numbers of the lost packets. Upon reception of a SREJ command, the master interface can decide whether to resend the lost information. Of course, the master cannot send SREJ commands to the slave. The SREJ command (selective-reject) is sent from the GBT-SCA to the master and contains the numbers of the lost packets. Upon reception of a SREJ command, the master interface can decide whether to resend the lost information. Of course, the master cannot send SREJ commands to the slave.

→ For more information regarding the HDLC protocol, refer to the HDLC standard specifications ISO/IEC 13239:2002. You can find the HDLC standard documentation at the following link: https://espace.cern.ch/GBT-Project/GBT-SCA/Manuals/Forms/AllItems.aspx

## 4.2. THE SCA CHANNEL COMMAND PROTOCOL LAYER

The SCA adopt a command-oriented protocol to address the on-chip interface channels and instruct the execution of specific operations. The SCA command frames are encapsulated in the HDLC e-link transport frames as shown in Figure 4.3. The transaction ID field associates the transmitted commands with the corresponding data replies, allowing the concurrent use of all the SCA channels.

At the reception of any request message, the GBT-SCA replies with an acknowledge message which includes a status flag and eventually data values accordingly to the request. The SCA command frames consist of an 8-bit transaction identification field (ID field), an 8-bit Destination/Source address field, an 8-bit Command/Error Flag field, an 8-bit Length qualifier, and a Data field of variable length.

## Channel field:

The channel field specifies the destination interface of the request message. Each command can be directed to a specific channel interface according to Table 4.1. The access to a channel is denied by the SCA until the operation is concluded and an acknowledge packet is generated.  If the master interface tries to interrogate the channel before that was received the acknowledge packet related to the previous request, an error packet will be returned.

| Channel | Code | Description |
|---|---|---|
| CTRL | 0x00 | SCA configuration registers |
| SPI | 0x01 | Serial Peripheral master Interface |
| GPIO | 0x02 | Parallel I/O interface |
| I2C0 | 0x03 | I$^2$C Serial interface – master 0 |
| I2C1 | 0x04 | I$^2$C Serial interface – master 1 |
| I2C2 | 0x05 | I$^2$C Serial interface – master 2 |
| I2C3 | 0x06 | I$^2$C Serial interface – master 3 |
| I2C4 | 0x07 | I$^2$C Serial interface – master 4 |
| I2C5 | 0x08 | I$^2$C Serial interface – master 5 |
| I2C6 | 0x09 | I$^2$C Serial interface – master 6 |
| I2C7 | 0x0A | I$^2$C Serial interface – master 7 |
| I2C8 | 0x0B | I$^2$C Serial interface – master 8 |
| I2C9 | 0x0C | I$^2$C Serial interface – master 9 |
| I2CA | 0x0D | I$^2$C Serial interface – master 10 |
| I2CB | 0x0E | I$^2$C Serial interface – master 11 |
| I2CC | 0x0F | I$^2$C Serial interface – master 12 |
| I2CD | 0x10 | I$^2$C Serial interface – master 13 |
| I2CE | 0x11 | I$^2$C Serial interface – master 14 |
| I2CF | 0x12 | I$^2$C Serial interface – master 15 |
| JTAG | 0x13 | JTAG serial master interface |
| ADC | 0x14 | Analog to digital converter |
| DAC | 0x15 | Digital to analog converter |

Table 4.1: Channels identification encoding

## Transaction ID field:

Specifies the message identification number. The reply messages generated by the SCA have the same transaction identifier of the request message allowing to associate the transmitted commands with the corresponding replies, permitting the concurrent use of all the SCA channels. It is not required that ID values are ordered. ID values 0x00 and 0xff are reserved for interrupt packets generated spontaneously by the SCA and should not be used in requests.

## Length field:

The length qualifier field specifies the number of bytes contained in the DATA field.

## Command field:

The Command field is present in the frames received by the SCA and indicates the operation to be performed. The operation can refer to a specific internal register of the channel like a configuration register, or to an external bus related operation, such as a start of communication.

A set of valid command is defined for each channel. Channels receiving an invalid command do not execute any action and reply with an error message.

## Error field:

The Error Flag field is present in the channel reply frames to indicate error conditions encountered in the execution of a command. If no errors are found, its value is 0x00. Table 4.2 defines the error-flag encoding.

| Bit number | Error Description |
|---|---|
| 0 | Generic error flag |
| 1 | Invalid channel request |
| 2 | Invalid command request |
| 3 | Invalid transaction number request |
| 4 | Invalid length |
| 5 | Channel not enabled |
| 6 | Channel currently busy |
| 7 | Command in treatment |

Table 4.2: Error flag encoding

## Data field:

The Data field is command dependent field whose length is defined by the length qualifier field. For example, in the case of a read/write operation on a GBT-SCA internal register, it contains the value written/read from the register.

# 5. THE SCA CONTROLLER CONFIGURATION

The GBT-SCA controller is a dedicated logic block inside each GBT-SCA, which is needed mainly for network and internal channels supervision. The GBT-SCA controller is reachable with the same protocol used to transfer data to the other port channels, by sending message with 'CHANNEL' field equal to 0x0.

## 5.1. CONTROL REGISTERS

Four eight-bit registers represents the GBT-SCA generic control registers as in Table 5.1:

| REGISTER | MODE | FUNCTION |
|---|---|---|
| ID | r | Read the chip unique identification number |
| CRB | r/w | Control register B – Channel enable register 1 |
| CRC | r/w | Control register C – Channel enable register 2 |
| CRD | r/w | Control register D – Channel enable register 3 |
| SEU | read | Single event upset counter |

Table 5.1: GBT-SCA generic control registers

Keeping the channels in disable state when not used, allows to reduce the power consumption. When a channel is disabled, the state machines are in fact clock gated in order to reduce the dynamic power consumption. When a channel is re-enabled, it is necessary to reconfigure it. It gets reset during the disable time, in order to avoid SEU related errors in absence of the clock.

### ID – Chip ID Register

A read operation on the ID register return the chip identification number. This 24 bit sequential number is written on internal e-fuses during production testing and allows to identify the specific SCA chip. Every SCA chip has a different ID.  It is not possible to modify or override the chip ID. The e-fuses bank is part of the ADC block. Thus, to read the SCA ID, the ADC block must be enabled before. Also, the SCA ID depends of the SCA version. The SCA version 1 has either 0x000000 or 0xFFFFFF ID value. The SCA version 2 has a unique ID, is written as a sequential number from its production.

| BIT | NAME | FUNCTION | ADDITIONAL INFO |
|---|---|---|---|
| 23-0 | ID | SCA unique identification number | Is not possible to modify the chip ID |

*reset value 0x00.

Table 5.2: ID register

## CRB - Control Register B

The Control Register B define the enable/disable state of the channels as described in Table 5.3:

| BIT | NAME | FUNCTION | ADDITIONAL INFO |
|---|---|---|---|
| 0 | - | - | reserved |
| 1 | ENSPI | SPI serial master interface enable flag | 1 enabled – 0 disabled |
| 2 | ENGPIO | Parallel Input / Output interface enable flag | 1 enabled – 0 disabled |
| 3 | ENI2C0 | $I^2C$ master interface number 0 enable flag | 1 enabled – 0 disabled |
| 4 | ENI2C1 | $I^2C$ master interface number 1 enable flag | 1 enabled – 0 disabled |
| 5 | ENI2C2 | $I^2C$ master interface number 2 enable flag | 1 enabled – 0 disabled |
| 6 | ENI2C3 | $I^2C$ master interface number 3 enable flag | 1 enabled – 0 disabled |
| 7 | ENI2C4 | $I^2C$ master interface number 4 enable flag | 1 enabled – 0 disabled |

*reset value 0x00.

Table 5.3: GBT-SCA Channel enable register B

## CRC - Control Register C

This register controls the enabled channel interfaces as defined in Table 5.4:

| BIT | NAME | FUNCTION | ADDITIONAL INFO |
|---|---|---|---|
| 0 | ENI2C5 | $I^2C$ master interface number 5 enable flag | 1 enabled – 0 disabled |
| 1 | ENI2C6 | $I^2C$ master interface number 6 enable flag | 1 enabled – 0 disabled |
| 2 | ENI2C7 | $I^2C$ master interface number 7 enable flag | 1 enabled – 0 disabled |
| 3 | ENI2C8 | $I^2C$ master interface number 8 enable flag | 1 enabled – 0 disabled |
| 4 | ENI2C9 | $I^2C$ master interface number 9 enable flag | 1 enabled – 0 disabled |
| 5 | ENI2CA | $I^2C$ master interface number 10 enable flag | 1 enabled – 0 disabled |
| 6 | ENI2CB | $I^2C$ master interface number 11 enable flag | 1 enabled – 0 disabled |
| 7 | ENI2CC | $I^2C$ master interface number 12 enable flag | 1 enabled – 0 disabled |

*reset value 0x00.

Table 5.4: GBT-SCA Channel enable register C

## CRD - Control Register D

This register controls the enabled channel interfaces as defined in Table 5.5:

| BIT | NAME | FUNCTION | ADDITIONAL INFO |
|---|---|---|---|
| 0 | ENI2CD | $I^2C$ master interface number 13 enable flag | 1 enabled – 0 disabled |
| 1 | ENI2CE | $I^2C$ master interface number 14 enable flag | 1 enabled – 0 disabled |
| 2 | ENI2CF | $I^2C$ master interface number 15 enable flag | 1 enabled – 0 disabled |
| 3 | ENJTAG | JTAG serial master interface enable flag | 1 enabled – 0 disabled |
| 4 | ENADC | Analog to Digital converter enable flag<br>eFuses/Serial Number reading | 1 enabled – 0 disabled |
| 5 | - | - | - |
| 6 | ENDAC | Digital to Analog converter enable flag | 1 enabled – 0 disabled |

*reset value 0x00.

Table 5.5: GBT-SCA Channel enable register D

### SEU - Single event upset counter

This register stores the value of the single event upset counter. The radiation hardening techniques applied to the chip allows the SCA to be tolerant to the SEU rate expected in the application field. The SEU counter allows to evaluate the average number of upsets that have been corrected, relatively to a small area of the ASIC of about 300 registers.

| BIT | NAME | FUNCTION | ADDITIONAL INFO |
|---|---|---|---|
| 31-0 | SEU | SEU counter value | - |

*reset value 0x00.

Table 5.6: SEU register

## 5.2. COMMANDS

Table 5.7 summarizes the commands to operate on the GBT-SCA generic control registers.

| COMMAND | FUNCTION | TYPE | CH | TRN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|---|
| CTRL_R_ID | Read the Chip ID | TX: | 0x14 | N | V2-> 0xD1<br>V1-> 0x91 | - | - | - | 1 |
| | | RX: | 0x14 | N | 0x00 | - | ID[23:16] | ID[15:8] | ID[7:0] |
| CTRL _W_CRB | Write Control reg. B | TX: | 0 | N | 0x02 | VAL | - | - | - |
| | | RX: | 0 | N | 0x00 | - | - | - | - |
| CTRL _W_CRC | Write Control reg. C | TX: | 0 | N | 0x04 | VAL | - | - | - |
| | | RX: | 0 | N | 0x00 | - | - | - | - |
| CTRL _W_CRD | Write Control reg. D | TX: | 0 | N | 0x06 | VAL | - | - | - |
| | | RX: | 0 | N | 0x00 | - | - | - | - |
| CTRL _R_CRB | Read Control reg. B | TX: | 0 | N | 0x03 | - | - | - | - |
| | | RX: | 0 | N | 0x00 | VAL | - | - | - |
| CTRL _R_CRC | Read Control reg. C | TX: | 0 | N | 0x05 | - | - | - | - |
| | | RX: | 0 | N | 0x00 | VAL | - | - | - |
| CTRL _R_CRD | Read Control reg. D | TX: | 0 | N | 0x07 | - | - | - | - |
| | | RX: | 0 | N | 0x00 | VAL | - | - | - |
| CTRL_R_SEU | Read the SEU counter | TX: | 0x13 | N | 0xF1 | - | - | - | - |
| | | RX: | 0x13 | N | 0x00 | - | - | - | VAL |
| CTRL_C_SEU | Reset the SEU counter | TX: | 0x13 | N | 0xF0 | - | - | - | 0 |
| | | RX: | 0x13 | N | 0x00 | - | - | - | - |

Table 5.7: GBT-SCA generic control registers access commands

# 6. I²C CHANNELS

The GBT-SCA include 16 independent I$^2$C master serial bus with the following features:

- Concurrent operation of all 16 channels
- Individually programmable data transfer rates: 100 KHz, 200 KHz, 400 KHz, 1 MHz
- Supports 7-bit and 10-bit addressing standards
- Supports single-byte and multi-byte I$^2$C read/write bus operations
- Support read-modify-write atomic operations with 'AND', 'OR' or 'XOR' masks.

## 6.1. CONFIGURATION REGISTERS

Table 6.1 lists the registers defined in the I$^2$C channel interface.

| REGISTER | MODE | FUNCTION |
|---|---|---|
| MASK | R/W | Mask register for read-modify-write operations |
| CTRL | R/W | Control register |
| STATUS | R | Status Register |
| DATA | R/W | Data register. Holds transmit or received buffers for multi-byte operations |

Table 6.1: I2C channel registers

### Mask register

The mask register is use in the read-modify-write operation. The value written in the remote I$^2$C slave interface is computed as: '$PD \circ Mask$' where $PD$ represent the previous data in the remote register, '$Mask$' represent the value of the local Mask register and '$\circ$' represent a boolean operation between 'AND', 'OR' and 'XOR', depending on the command. To write or read this register use the commands: I2C_W_MASK and I2C_R_MASK as defined in Table 6.15.

| BIT | NAME | FUNCTION |
|---|---|---|
| **7-0** | MASK | Value of the mask |

*The reset value of this register is 0x00.

Table 6.2: I2C channel MASK register

## Control register

The control register defines the operating mode of the I$^2$C channel. To write or read the control register use the commands: I2C_W_CTRL and I2C_R_CTRL as defined in Table 6.15.

| BIT | NAME | FUNCTION |
|---|---|---|
| 1-0 | FREQ | Select the communication speed:<br>FREQ = 00 -> 100kHz FREQ = 01 -> 200kHz<br>FREQ = 10 -> 400kHz FREQ = 11 -> 1MHz |
| 6-2 | NBYTE | The NBYTE field defines the I2C transmission length expressed in number of bytes.<br>It is used only when multi-byte transmissions occur.<br>Allowed values from 1 to 16. |
| 7 | SCLMODE | Define the SCL functionality. <u>This control bit is present only in SCA V2</u><br>SCLMODE = 0 The SCL pad act as open-drain.<br>SCL value equal to 0 -> Force the SCL line to DGND.<br>SCL value equal to 1 -> SCL line in high impedance.<br>SCLMODE = 1 The SCL pad act as CMOS output.<br>SCL value equal to 0 -> Force the SCL line to DGND.<br>SCL value equal to 1 -> Force the SCL line to DVDD. |

* reset value 0x00.

Table 6.3: I2C control register

## Status register

The status register reports flags concerning the latest I$^2$C bus operation according to Table 6.4.

| BIT | NAME | FUNCTION |
|---|---|---|
| 1-0 | res | - |
| 2 | SUCC | This bit is set when the last I2C transaction was successfully executed. |
| 3 | LEVERR | This bit is set to ‘1’ if the I2C master port finds that the SDA line is pulled low ‘0’ before initiating a transaction. If this happens the I2C bus is probably broken. The bit represents the status of the SDA line and cannot be reset. |
| 4 | Empty | - |
| 5 | INVCOM | This bit is set if an invalid command was sent to the I2C channel. The bit is cleared by a channel reset. |
| 6 | NOACK | This bit is set if the last operation has not been acknowledged by the I2C slave acknowledge. This bit is set/reset at the end of each I2C transaction. |
| 7 | Empty | - |

*The reset value of this register is 0x00.

Table 6.4: I2C channel STATUS register

To read the status register use the commands: I2C_R_STR as defined in paragraph 0. The value of the status register moreover is returned in the data fields of the reply generated by any start of transmission command.

## Data register

The DATA register hold the data bytes to transmit for multi-byte I$^2$C write transactions and the received data bytes for the multi-byte I$^2$C read transactions.

| BIT | NAME | FUNCTION |
|---|---|---|
| 7:0 | BYTE0 | data transmit buffer for multi-byte I2C transactions |
| 15:8 | BYTE1 | data transmit buffer for multi-byte I2C transactions |
| 23:16 | BYTE2 | data transmit buffer for multi-byte I2C transactions |
| 31:24 | BYTE3 | data transmit buffer for multi-byte I2C transactions |
| 39:32 | BYTE4 | data transmit buffer for multi-byte I2C transactions |
| 47:40 | BYTE5 | data transmit buffer for multi-byte I2C transactions |
| 55:48 | BYTE6 | data transmit buffer for multi-byte I2C transactions |
| 63:56 | BYTE7 | data transmit buffer for multi-byte I2C transactions |
| 71:64 | BYTE8 | data transmit buffer for multi-byte I2C transactions |
| 79:72 | BYTE9 | data transmit buffer for multi-byte I2C transactions |
| 87:80 | BYTE10 | data transmit buffer for multi-byte I2C transactions |
| 95:88 | BYTE11 | data transmit buffer for multi-byte I2C transactions |
| 103:96 | BYTE12 | data transmit buffer for multi-byte I2C transactions |
| 111:104 | BYTE13 | data transmit buffer for multi-byte I2C transactions |
| 119:112 | BYTE14 | data transmit buffer for multi-byte I2C transactions |
| 127:120 | BYTE15 | data transmit buffer for multi-byte I2C transactions |

*The reset value of this register is 0x0.

Table 6.5: I2C channel DATA register

Data bytes in multi-byte write transaction are transmitted from BYTE0 to BYTE15. Received data bytes in multi-byte read transaction are located starting from BYTE0 to BYTE15, according to the value of the NBYTE field of the CONTROL register.

To write this register use: I2C_W_DAT0, I2C_W_DAT1, I2C_W_DAT2, I2C_W_DAT3. While to read use the commands: I2C_R_DAT0, I2C_R_DAT1, I2C_R_DAT2, I2C_R_DAT3 as defined in Table 6.15.

## 6.2. START OF TRANSMISSION COMMANDS

Multiple methods to start an I$^2$C communication are implemented:

### Single byte write operation in 7-bit addressing mode

The I2C_S_7B_W command can be used to start an I$^2$C single byte write transmission, using 7-bit addressing standard. The data field of the sent message must contain in the order the slave address followed by the data byte to send. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 4 | I2C_S_7B_W 0x82 | ADDRESS | DATA | - | - |
| **RX:** | I2C channel | N | 4 | Err flag | STATUS | - | - | - |

Table 6.6: Example of I2C single byte write operation in 7-bit addressing mode

### Single byte read operation in 7-bit addressing mode

The I2C_S_7B_R command can be used to start an I$^2$C single byte Read transmission, using 7-bit addressing standard. The data field of the request message must contain the slave address. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 4 | I2C_S_7B_R 0x86 | ADDRESS | - | - | - |
| **RX:** | I2C channel | N | 4 | Err flag | STATUS | DATA | - | - |

Table 6.7: Example of I2C single byte read operation in 7-bit addressing mode

### Single byte write operation in 10-bit addressing mode

The I2C_S_10B_W command can be used to start an I$^2$C single byte Write transmission, using 10-bit addressing standard. The data field of the request message must contain in the order the slave address expressed on 2 bytes followed by the data byte to send. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 6 | I2C_S_10B_W 0x8A | 1110-A9-A8 | A7-A6-A5-A4-A3-A2-A1-A0 | DATA | - |
| **RX:** | I2C channel | N | 4 | Err flag | STATUS | - | - | - |

Table 6.8: Example of I2C single byte write operation in 10-bit addressing mode

In Table 6.8, A7-A0 represents the address bits. According to the I$^2$C standard, the 10-bit addressing mode requires that the most significant bits of the address are 1110. The SCA allows anyway to transmit different values on those bit, permitting to use up to 15 bit for addressing custom slaves (out of standard).

## Single byte read operation in 10-bit addressing mode

The I2C_S_10B_R command can be used to start an I$^2$C single byte read transmission, using 10-bit addressing standard. The data field of the request message must contain in the slave address expressed on 2 bytes. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 4 | I2C_S_10B_R 0x8E | 1110-A9-A8 | A7-A6-A5-A4-A3-A2-A1-A0 | - | - |
| **RX:** | I2C channel | N | 4 | Err flag | STATUS | DATA | - | - |

Table 6.9: Example of I2C single byte read operation in 10-bit addressing mode

## Multi byte write operation in 7-bit addressing mode

The I2C_M_7B_W command can be used to start an I$^2$C multi byte Write transmission, using 7-bit addressing standard. The data field of the request message must contain only the slave address. The data bytes to send needs therefore to be previously uploaded in the DATA register and the number of byte to transmit in the NBYTE field of the control register. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 4 | I2C_M_7B_W 0xDA | ADDRESS | - | - | - |
| **RX:** | I2C channel | N | 6 | Err flag | STATUS | - | - | - |

Table 6.10: Example of I2C multi-byte write operation in 7-bit addressing mode

## Multi byte read operation in 7-bit addressing mode

The I2C_M_7B_R command can be used to start an I$^2$C multi byte Read transmission, using 7-bit addressing standard. The data field of the request message must contain only the slave address. The user needs to previously set the number of byte to receive in the NBYTE field of the control register. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 4 | I2C_M_7B_R 0xDE | ADDRESS | - | - | - |
| **RX:** | I2C channel | N | 4 | Err flag | STATUS | - | - | - |

Table 6.11: Example of I2C multi-byte read operation in 7-bit addressing mode

## Multi byte write operation in 10-bit addressing mode

The I2C_M_10B_W command can be used to start an I$^2$C multi byte Write transmission, using 10-bit addressing standard. The data field of the request message must contain only the slave address, expressed on two bytes. The data bytes to send needs therefore to be previously up-loaded in the DATA register and the number of byte to transmit in the NBYTE field of the control register. The reply message generated by the GBT-SCA include the value of the STATUS register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| TX: | I2C channel | N | 2 | I2C_M_10B_W 0xE2 | 1110-A9-A8 | A7-A6-A5-A4-A3-A2-A1-A0 | - | - |
| RX: | I2C channel | N | 2 | Err flag | STATUS | - | - | - |

Table 6.12: Example of I2C multi-byte write operation in 10-bit addressing mode

## Multi byte read operation in 10-bit addressing mode

The I2C_M_10B_R command can be used to start an I$^2$C single byte read transmission, using 10-bit addressing standard. The data field of the request message must contain in the slave address expressed on 2 bytes. The user needs to previously set the number of byte to receive in the NBYTE field of the control register.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 2 | I2C_M_10B_R 0xE6 | 1110-A9-A8 | A7-A6-A5-A4-A3-A2-A1-A0 | - | - |
| **RX:** | I2C channel | N | 2 | Err flag | STATUS | - | - | - |

Table 6.13: Example of I2C multi-byte read operation in 10-bit addressing mode

## Read modify write atomic operation

The I2C_RMW_AND, I2C_RMW_OR and I2C_RMW_XOR commands allows to execute with one atomic command a read modify and write operation. The data field of the request packet should contain the destination slave address. The I$^2$C interface reads a byte from the specified address, a logical operation is performed with the MASK register value and the result is written back into the same I$^2$C address. An example of operation is:

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | I2C channel | N | 2 | I2C_RMW_AND 0x<br>I2C_RMW_OR 0xC6<br>I2C_RMW_XOR 0xCA | ADDRESS | DATA | - | - |
| **RX:** | I2C channel | N | 2 | Err flag | STATUS | - | - | - |

Table 6.14: Example of I2C Read-Modify-Write atomic operation in 7-bit addressing mode

## 6.3. COMMANDS SUMMARY

The Table 6.15 summarizes the commands accepted by the I2C channel.

| COMMAND | FUNCTION | TYPE | CH | TRN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|---|
| **I2C_W_CTRL** | Write CONTROL register | TX: | I2cx | N | **0x30** | VALUE | -- | -- | -- |
| | | RX: | I2cx | N | flag | -- | -- | -- | -- |
| **I2C_R_CTRL** | Read CONTROL register | TX: | I2cx | N | **0x31** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | VALUE | -- | -- | -- |
| **I2C_R_STR** | Read STATUS register | TX: | I2cx | N | **0x11** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | VALUE | -- | -- | -- |
| **I2C_W_MSK** | Write MASK register | TX: | I2cx | N | **0x20** | VALUE | -- | -- | -- |
| | | RX: | I2cx | N | Flag | -- | -- | -- | -- |
| **I2C_R_MSK** | Read MASK register | TX: | I2cx | N | **0x21** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | VALUE | -- | -- | -- |
| **I2C_W_DATA0** | Write data register bytes 0,1,2,3 | TX: | I2cx | N | **0x40** | BYTE0 | BYTE1 | BYTE2 | BYTE3 |
| | | RX: | I2cx | N | Flag | -- | -- | -- | -- |
| **I2C_R_DATA0** | Read data register bytes 0,1,2,3 | TX: | I2cx | N | **0x41** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | BYTE12 | BYTE13 | BYTE14 | BYTE15 |
| **I2C_W_DATA1** | Write data register bytes 4,5,6,7 | TX: | I2cx | N | **0x50** | BYTE4 | BYTE5 | BYTE6 | BYTE7 |
| | | RX: | I2cx | N | Flag | -- | -- | -- | -- |
| **I2C_R_DATA1** | Read data register bytes 4,5,6,7 | TX: | I2cx | N | **0x51** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | BYTE8 | BYTE9 | BYTE10 | BYTE11 |
| **I2C_W_DATA2** | Write data register bytes 8,9,10,11 | TX: | I2cx | N | **0x60** | BYTE8 | BYTE9 | BYTE10 | BYTE11 |
| | | RX: | I2cx | N | Flag | -- | -- | -- | -- |
| **I2C_R_DATA2** | Read data register bytes 8,9,10,11 | TX: | I2cx | N | **0x61** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | BYTE4 | BYTE5 | BYTE6 | BYTE7 |
| **I2C_W_DATA3** | Write data register bytes 12,13,14,15 | TX: | I2cx | N | **0x70** | BYTE12 | BYTE13 | BYTE14 | BYTE15 |
| | | RX: | I2cx | N | Flag | -- | -- | -- | -- |
| **I2C_R_DATA3** | Read data register bytes 12,13,14,15 | TX: | I2cx | N | **0x71** | -- | -- | -- | -- |
| | | RX: | I2cx | N | Flag | BYTE0 | BYTE1 | BYTE2 | BYTE3 |
| **I2C_S_7B_W** | Start I2C single byte write transaction using 7-bits address | TX: | I2cx | N | **0x82** | ADR [6:0] | DATA | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_S_7B_R** | Start I2C single byte read transaction using 7-bits address | TX: | I2cx | N | **0x86** | ADR [6:0] | -- | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | DATA | -- | -- |
| | | TX: | I2cx | N | **0x8A** | ADR [9:8] | ADR [7:0] | DATA | -- |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **I2C_S_10B_W** | Start I2C single byte write transaction using 10-bits address | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_S_10B_R** | Start I2C single byte read transaction using 10-bits address | TX: | I2cx | N | **0x8E** | ADR [9:8] | ADR [7:0] | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | DATA | -- | -- |
| **I2C_M_7B_W** | Start I2C multi byte write transaction using 7-bits address | TX: | I2cx | N | **0xDA** | ADR [6:0] | -- | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_M_7B_R** | Start I2C multi byte read transaction using 7-bits address | TX: | I2cx | N | **0xDE** | ADR [6 :0] | -- | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_M_10B_W** | Start I2C multi byte write transaction using 10-bits address | TX: | I2cx | N | **0xE2** | ADR [9:8] | ADR [7:0] | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_M_10B_R** | Start I2C multi byte read transaction using 10-bits address | TX: | I2cx | N | **0xE6** | ADR [9:8] | ADR [7:0] | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_RMW_AND** | Start I2C read-modify-write transaction with AND mask | TX: | I2cx | N | | ADR [6:8] | DATA | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_RMW_OR** | Start I2C read-modify-write transaction with OR mask | TX: | I2cx | N | **0xC6** | ADR [6:8] | DATA | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |
| **I2C_RMW_XOR** | Start I2C read-modify-write transaction with XOR mask | TX: | I2cx | N | **0xCA** | ADR [6:8] | DATA | -- | -- |
| | | RX: | I2cx | N | Flag | STATUS | -- | -- | -- |

Table 6.15: I2C channel command list

## 6.4. IO PADS

| | DIRECTION | MAX CURRENT | V RANGE | DESCRIPTION |
|---|---|---|---|---|
| **SDA** | Bidirectional | 8mA | 0V - DVDD | Data serial line pad<br>CMOS digital tristate input/output<br>logic value ‘0’ → force the 0 value on the line logic value ‘1’ → tristate<br>Needs an external pull-up resistor at DVDD |
| **SCL** | Output | 8mA | 0V - DVDD | Clock serial line pad<br>According to the value of the SCLMODE bit in the control register:<br>SCLMODE=0 → CMOS digital output<br>SCLMODE=1 → CMOS tristate output as the SDA pad.<br>It Needs an external pull-up resistor at DVDD |

Table 6.16: I2C channel pad list

# 7. SPI CHANNEL

The GBT-SCA includes a SPI (Serial Peripheral Interface) full duplex synchronous serial bus master with the following features:

- 8 individual slave select lines.
- Single transaction length of up to 128 bits.
- Programmable transfer rate from 156 kHz to 20 MHz.
- Supports all the standard SPI bus operating modes: 00, 01, 10 and 11.

The SPI channel is implemented around a 128-bit shift register that serializes and de-serializes the bit-streams between the MISO and MOSI SPI lines and the internal parallel bus. The transactions are initiated by the reception of a user command and executed locally by the channel's state machines. Upon completion, a return packet is generated containing user data and status flags. The channel can be powered down to reduce power consumption in periods of inactivity.

The SPI master of the SCA is compatible with both the standard implementation: SPI (trademark of Motorola Semiconductor) and Microwire/Plus (trademark of National Semiconductor). It implements a set of registers to configure the interface and the operating modes, to read the received data and write the bytes to transmit. Specific commands are defined to access those registers and to start the communication.

## 7.1. CONFIGURATION REGISTERS

Table 7.1 lists the SPI channel configuration registers.

| REGISTER | MODE | FUNCTION |
|---|---|---|
| CONTROL | r/w | Control register |
| FREQUENCY | r/w | Frequency divider register |
| SLSELECT | r/w | Slave Select register |
| DATA | r/w | Data register which holds transmit or received bits |

Table 7.1: SPI channel configuration registers

## Control register

The **CONTROL** register define the operating mode of the SPI channel. To write or read the control register use the commands: SPI_W_CTRL and SPI_R_CTRL as defined in Table 7.7.

| BIT | NAME | FUNCTION |
|---|---|---|
| 6-0 | LEN | Represents how many bits to transmit during the following transmission.<br>It can assume values from 0 to 127 (value ‘0’ represents 128 bits) |
| 7 | INVSCLK | Invert the SCLK level during inactivity time.<br>INVSCLK = 0 -> SCLK idle level is low<br>INVSCLK = 1 -> SCLK idle level is high |
| 8 | ~~GO/BUSY~~ | Alternative method for starting the SPI transmission/Busy flag. Deprecated in SCA V2.<br>The use of this method is not suggested, please write always value 0 to this bit. |
| 9 | RXEDGE | Define the SCLK sampling edge of the MISO input line<br>RXEDGE = 0 -> MISO signal is sampled on the rising edge of SCLK<br>RXEDGE = 1 -> MISO signal is sampled on the falling edge of SCLK |
| 10 | TXEDGE | Define the SCLK transmit edge of the MOSI output line<br>TXEDGE = 0 -> MOSI signal change on the rising edge of SCLK<br>TXEDGE = 1 -> MOSI signal change on the falling edge of SCLK |
| 11 | MSB/LSB | Define the transmit order of the bits in the transmit FIFO and the position of the received bit in the received FIFO<br>MSB/LSB = 0 -> Bits are transmitted from the most significant to the least significant<br>MSB/LSB = 1 -> Bits are transmitted from the least significant to the most significant |
| 12 | ~~IE~~** | Interrupt enable. Deprecated in SCA-V2. Please always write 1 in this bit. |
| 13 | SSMODE | Define if the Slave Select output signal is automatic or manually controlled.<br>SSMODE = 1 -> When the GO command is sent, the slave select line (chosen with the SLAVESELECT register) is enabled at the beginning of the transmission and disabled at the end of the transmission.<br>SSMODE = 0 -> The slave select pads are manually controlled. Any write operation on the SLAVESELECT register, toggle immediately the corresponding slave select pads.<br>See SLAVE SELECT register for more information. |

*The reset value of the register is 0b-00010000-00000000.

**In SCA V2 bit 12 is reserved, write operations are not influent.

Table 7.2: SPI channel control register

## Frequency register

The value of the **FREQUENCY** register defines serial transmission frequency of the SPI bus. Values from 0 to 65535 are accepted, resulting in a minimum frequency of 305 Hz and a maximum frequency of 20MHz. The transmission frequency is computed as: $f_{TCK} = 2 \cdot \frac{10^7}{DIV+1}$. To write or read the control register use the commands: SPI_W_DIV and SPI_R_DIV as defined in Table 7.7.

| BIT | NAME | FUNCTION |
|---|---|---|
| 15-0 | DIV | Represent the division factor to compute the SPI transmission frequency |

*The reset value of this register is 0.

Table 7.3: SPI channel frequency register

## Slave Select register

The **SLAVE SELECT** register allows to select which of the eight enable lines is activated. Each bit corresponds to one of the SPI slave-enable outputs. It is therefore possible to enable more than one slave enable line at the time (usually used in daisy-chain slave disposition).

The behaviour of the slave select output depends on the SSMODE flag in the control register.

If SSMODE is equal to 1, the slave-enable lines are activated when the SPI transmission starts and released when the SPI transmission ends. If SSMODE is equal to 0, the slave-enable lines are activated when immediately when writing to this register. To write or read the control register use the commands: SPI_W_SS and SPI_R_SS as defined in Table 7.7.

| BIT | NAME | FUNCTION |
|---|---|---|
| 7-0 | SS | Slave Select enable register. Each bit corresponds to one of the slave-select outputs. |

*The reset value of this register is 0.

Table 7.4: SPI channel Slave Select register

## Data register

The **DATA** register represent the transmit/receive buffer for the MOSI and MISO serial lines. Before to start an SPI transmission the user needs to write the data to transmit in the **DATA** register. When the transmission is completed, the **DATA** register holds the received bits from the SPI slave. Transmit and receive buffers share the same flip-flops, it is important to not overwrite the **DATA** register before to read the received bits of the previous transmission.

| BIT | NAME | FUNCTION |
|---|---|---|
| 127:0 | LEN | Data transmit/receive buffer |

*The reset value of this register is 0x0.

Table 7.5: SPI channel DATA register

The valid bits of this register depend on the length value written in LEN (bits 0:6 of **CONTROL** register). To write or read the control register use the SPI_W_DATA<> and SPI_R_DATA<> commands as defined in Table 7.7.

## 7.2. START OF TRANSMISSION

### Start of transmission

The SPI channel defines the **SPI_GO** command to start the serial communication. Before to start the SPI transaction, it is necessary to fill the transmit FIFO writing the DATA register. During the transmission time, the access to the core is refused. Any received packet directed to the SPI module will be acknowledged with a "channel busy" error flag. After the transmission, the SCA notify the end of the operation with an acknowledge packet and enables the access to the SPI core. The user can at this point access the DATA register to read the received bits.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | SPI channel 0x01 | N | 2 | SPI_GO 0x72 | - | - | - | - |
| **RX:** | I2C channel 0x01 | N | 4 | Err flag | - | - | - | - |

Table 7.6: Start of SPI transmission command

## 7.3. COMMAND TABLE

Table 7.7 summarizes the commands accepted by the SPI channel for operations on its registers.

| COMMAND | FUNCTION | REQUEST AND REPLY FORMATS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | TYPE | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| **SPI_W_CTRL** | Write CONTROL register | TX: | 0x01 | N | 6 | **0x40** | -- | -- | CTRL[13:8] | CTRL[7:0] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_CTRL** | Read CONTROL register | TX: | 0x01 | N | 2 | **0x41** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | -- | -- | CTRL[15:8] | CTRL[7:0] |
| **SPI_W_FREQ** | Write frequency divider register | TX: | 0x01 | N | 6 | **0x50** | -- | -- | FREQ[15:8] | FREQ[7:0] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_FREQ** | Read frequency divider register | TX: | 0x01 | N | 2 | **0x51** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | -- | -- | FREQ[15:8] | FREQ[7:0] |
| **SPI_W_SS** | Write slave select register | TX: | 0x01 | N | 6 | **0x60** | -- | -- | -- | SS[7:0] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| | | TX: | 0x01 | N | 2 | **0x61** | -- | -- | -- | -- |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **SPI_R_SS** | Read slave select Register | RX: | 0x01 | N | 6 | Flag | -- | -- | -- | SS[7:0] |
| **SPI_W_MOSI0** | Write MOSI DATA buffer Bits [31:0] | TX: | 0x01 | N | 6 | **0x00** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_MISO0** | Read MOSI DATA buffer Bits [31:0] | TX: | 0x01 | N | 2 | **0x01** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| **SPI_W_MOSI1** | Write MOSI DATA Bits [63:32] | TX: | 0x01 | N | 6 | **0x10** | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_MISO1** | Read MOSI DATA Bits [63:32] | TX: | 0x01 | N | 2 | **0x11** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| **SPI_W_MOSI2** | Write MOSI DATA Bits [95:64] | TX: | 0x01 | N | 6 | **0x20** | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_MISO2** | Read MOSI DATA Bits [95:64] | TX: | 0x01 | N | 2 | **0x21** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| **SPI_W_MOSI3** | Write MOSI DATA Bits [127:96] | TX: | 0x01 | N | 6 | **0x30** | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| | | RX: | 0x01 | N | 2 | Flag | -- | -- | -- | -- |
| **SPI_R_MISO3** | Read MOSI DATA Bits [127:96] | TX: | 0x01 | N | 2 | **0x31** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| **SPI_GO** | Start SPI transaction | TX: | 0x01 | N | 2 | **0x72** | -- | -- | -- | -- |
| | | RX: | 0x01 | N | 6 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |

Table 7.7: SPI channel command list

## 7.4. IO PADS

| PAD NAME | DIRECTION | MAXIMUM CURRENT | VOLTAGE VALUE | DESCRIPTION |
|---|---|---|---|---|
| **TCK** | Output | 12mA | 0V - DVDD | Clock line |
| **MOSI** | Output | 12mA | 0V - DVDD | Serial data master to slave pad |
| **MISO** | Input | 12mA | 0V - DVDD | Serial data slave to master pad |
| **SS0** | Output | 4mA | 0V – DVDD | Slave Select 0 pad |
| **SS1** | Output | 4mA | 0V – DVDD | Slave Select 1 pad |
| **SS2** | Output | 4mA | 0V – DVDD | Slave Select 2 pad |
| **SS3** | Output | 4mA | 0V – DVDD | Slave Select 3 pad |
| **SS4** | Output | 4mA | 0V – DVDD | Slave Select 4 pad |
| **SS5** | Output | 4mA | 0V – DVDD | Slave Select 5 pad |
| **SS6** | Output | 4mA | 0V – DVDD | Slave Select 6 pad |
| **SS7** | Output | 4mA | 0V - DVDD | Slave Select 7 pad |

Table 7.8: SPI channel pad list

# 8. JTAG CHANNEL

The GBT-SCA include a user-programmable JTAG master player with the following features:

- Single bus operation up to 128 bits
- Programmable transfer rate up to 20MHz
- Slave asynchronous reset line of configurable pulse width.
- Configurable transmit and sampling clock edges
- Configurable bit order

The JTAG channel is implemented around two 128-bit shift registers that serialize and reserialize the bit-streams between the TMS, TDO and TDI lines and the internal parallel bus. The JTAG channel in the SCA does not implement a JTAG TAP controller state machine therefore the JTAG bus cycles will be generated by the FPGA circuitry residing at the control room electronics and downloaded on the transmission FIFOs of the GBT-SCA.

The channel can be powered down to reduce power consumption in periods of inactivity. It implements a set of registers to configure the interface and the operating modes, to read the received data and write the bytes to transmit. Specific commands are defined to access those registers and to start the communication.

## 8.1. CONFIGURATION REGISTERS

Table 8.1 lists the JTAG channel configuration registers.

| REGISTER | MODE | FUNCTION |
|---|---|---|
| CONTROL | r/w | Control register |
| FREQUENCY | r/w | Frequency divider register |
| TDO | r/w | Data transmit buffer for the TDO line |
| TDI | r | Data receive buffer for the TDI line |
| TMS | r/w | Data transmit buffer for the TMS line |

Table 8.1: JTAG channel configuration registers

## Control register

The **CONTROL** register define the operating mode of the JTAG channel. To write or read the control register use the commands: JTAG_W_CTRL and JTAG_R_CTRL as defined in Table 8.10.

| BIT | NAME | FUNCTION |
|---|---|---|
| 6-0 | LEN | Number of bits transmitted in the single JTAG operation.<br>Values from 0 to 127. (0 represents 128 bytes transmitted) |
| 7 | -- | -- |
| 8 | BUSY / ~~GO~~ | SCA-V2 -> Busy flag. JTAG bus operation currently going. Write operation not influent.<br>SCA-V1 -> Alternative method for starting the JTAG transmission. Replaced with the JTAG_GO_M command in SCA-V2. |
| 9 | RXEDGE | Define the TCK sampling edge of the TDI input line<br>RXEDGE = 0 -> TDI signal is sampled on the rising edge of TCK (STANDARD)<br>RXEDGE = 1 -> TDI signal is sampled on the falling edge of TCK |
| 10 | TXEDGE | Define the TCK transmit edge of the TDO and TMS output lines<br>TXEDGE = 0 -> TDO and TMS signal change on the rising edge of TCK (STANDARD)<br>TXEDGE = 1 -> TDO and TMS signal change on the falling edge of TCK |
| 11 | MSB/LSB | Define the transmit order of the bits in the transmit FIFO and the position of the received bit in the received FIFO<br>MSB/LSB = 0 -> Bits are transmitted from the most significant to the least significant<br>MSB/LSB = 1 -> Bits are transmitted from the least significant to the most significant |
| 13-12 | -- | -- |
| 14 | INVTCK | Invert the TCK signal<br>INVSCLK = 0 -> TCK idle level is high (STANDARD)<br>INVSCLK = 1 -> TCK idle level is low |
| 15 | -- | Not influent |
| 16 | ~~ARESET~~ | Asynchronous reset enable. Implemented only in SCA-V1.<br>In SCA-V2 the functionality of this bit is replaced with the JTAG_RESET command. |

The reset value of this register is 0b-00010000-00000000.

Table 8.2: JTAG channel control register

## Frequency register

The value of the **FREQUENCY** register defines serial transmission frequency of the JTAG bus. Values from 0 to 65535 are accepted, resulting in a minimum frequency of 305 Hz and a maximum frequency of 20MHz. The frequency is computed as: $f_{TCK} = 2 \cdot \frac{10^7}{DIV+1}$. To write or read the control register use the commands: JTAG_W_CTRL and JTAG_R_CTRL as defined in Table 8.10.

| BIT | NAME | FUNCTION |
|---|---|---|
| 15-0 | DIV | Represent the division factor to compute the JTAG transmission frequency |

*The reset value of this register is 0x0.

Table 8.3: JTAG channel frequency register

### TDO register

The **TDO** register represent the transmit buffer for the TDO serial line. Before to start a JTAG transmission, the user needs to write the data to transmit in the **TDO** register. The valid bits of this register depend on the length value written in LEN (bits 0:6 of **CONTROL** register). To write or read the control register use the JTAG_W_TDO<> and JTAG_R_TDI<> commands as defined in Table 8.10.

| BIT | NAME | MODE | FUNCTION |
|---|---|---|---|
| 127:0 | TDO | R/W | TDO transmit buffer |

*The reset value of this register is 0x0.

Table 8.4: JTAG channel TDO data buffer register

### TMS register

The **TMS** register represent the transmit buffer for the TMS serial line. Before to start a JTAG transmission the user needs to write the data to transmit in the **TMS** register. The valid bits of this register depend on the length value written in LEN (bits 0:6 of **CONTROL** register). To write or read the control register use the JTAG_W_TMS<> and JTAG_R_TMS<> commands as defined in Table 8.10.

| BIT | NAME | MODE | FUNCTION |
|---|---|---|---|
| 127:0 | TMS | R/W | TMS transmit buffer |

*The reset value of this register is 0x0.

Table 8.5: JTAG channel TMS data buffer register

### TDI register

The **TDI** register represent the receive buffer for the TDI serial line. When the bus operation is completed, the **TDI** register holds the received bits from the JTAG slave. The valid bits of this register depend on the length value written in LEN (bits 0:6 of **CONTROL** register). To read the control register use the JTAG_R_TDI<> commands as defined in Table 8.10. Transmit (TDO register) and receive (TDI register) buffers share the same flip-flops. It is important to do not overwrite the **DATA** register for the next transaction before to read the received bits of the previous one.

| BIT | NAME | MODE | FUNCTION |
|---|---|---|---|
| 127:0 | TDI | R | TDI receive buffer |

*The reset value of this register is 0x0.

Table 8.6: JTAG channel TDI data buffer register

## 8.2. TRANSMISSION COMMANDS

### Start of JTAG transmission

The JTAG channel defines the **JTAG_GO** (0xA2) command to start the serial communication. It is necessary to fill the transmit FIFOs writing the TDO and TMS registers before to start the JTAG bus operation. During the transmission time, the access to the core is refused. Any received packet directed to the JTAG channel will be acknowledged with “channel busy” error flag. After the transmission, the SCA notify the end of the operation with an acknowledge packet and enables the access to the JTAG channel. The user can at this point access the TDI register to read the received bits. The JTAG_GO command has the structure on Table 8.7**Error! Reference source not found.** .

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | JTAG 0x13 | N | 2 | JTAG_GO 0xA2 | - | - | - | - |
| **RX:** | JTAG 0x13 | N | 2 | Err flag | - | - | - | - |

Table 8.7: Start of JTAG transmission command

### Start of JTAG transmission in manual mode (SCA-V2 only)

JTAG_GO_M command allows to manually control the JTAG bus behavior. Before to start the JTAG operation, it is necessary to fill the transmit FIFOs writing the TDO and TMS registers. During the transmission time, in this case, the access to the core is allowed. The chip reply immediately with an acknowledge packet and just than the JTAG operation starts. The BUSY flag remain set during the whole duration of the communication and is cleared only at the end. Since there is no notification of the end of the operation, the user software will need to poll continuously the BUSY flag (bit 8 of the control register) to know the status of the operation. Pay attention in this case that the access to the core is not denied by the SCA during the busy time so it is the responsibility of the user application to guarantee to do not overwrite the buffer or the control flag while the communication is still running, unless you are doing this intentionally. The Table 8.8 provides an example of usage of the JTAG-GO_M command.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | JTAG 0x13 | N | 1 | JTAG_GO_M 0xB0 | - | - | - | - |
| **RX:** | JTAG 0x13 | N | 2 | Err flag | - | - | - | - |

Table 8.8: Start of JTAG transmission alternative command

### Send a JTAG reset pulse (SCA-V2 only)

The JTAG channel implements the asynchronous reset pulse line. In order to generate a reset pulse to the slave interface it is necessary to send the **JTAG_ARESET** (0xC0) command. The length of the pulse depends on the value of the **LEN** field of the **CONTROL** register (expressed in multiples of 25ns).

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | JTAG 0x13 | N | 1 | JTAG_ARESET 0xC0 | - | - | - | - |
| **RX:** | JTAG 0x13 | N | 2 | Err flag | - | - | - | - |

Table 8.9: JTAG reset pulse command

## 8.3. COMMANDS LIST

Table 8.10 summarizes the commands accepted by the JTAG channel for operations on its registers and for the start of transmission.

| COMMAND | FUNCTION | REQUEST AND REPLY FORMATS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | TYPE | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| **JTAG_W_CTRL** | Write CONTROL register | TX: | 0x13 | N | 6 | **0x80** | -- | -- | CTRL[15:8] | CTRL[7:0] |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_CTRL** | Read CONTROL register | TX: | 0x13 | N | 2 | **0x81** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 6 | Flag | -- | -- | CTRL[15:8] | CTRL[7:0] |
| **JTAG_W_FREQ** | Write frequency divider register | TX: | 0x13 | N | 6 | **0x90** | -- | -- | FREQ[15:8] | FREQ[7:0] |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_FREQ** | Read frequency divider register | TX: | 0x13 | N | 2 | **0x91** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 6 | Flag | -- | -- | FREQ[15:8] | FREQ[7:0] |
| **JTAG_W_TDO0** | Write TDO DATA Bits [31:0] | TX: | 0x13 | N | 6 | **0x00** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TDI0** | Read TDI DATA Bits [31:0] | TX: | 0x13 | N | 2 | **0x01** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 6 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| **JTAG_W_TDO1** | Write TDO DATA Bits [63:32] | TX: | 0x13 | N | 6 | **0x10** | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TDI1** | Read TDI DATA Bits [63:32] | TX: | 0x13 | N | 2 | **0x11** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 6 | Flag | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| **JTAG_W_TDO2** | Write TDO DATA Bits [95:64] | TX: | 0x13 | N | 6 | **0x20** | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TDI2** | Read TDI DATA | TX: | 0x13 | N | 2 | **0x21** | -- | -- | -- | -- |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Bits [95:64] | RX: | 0x13 | N | 6 | Flag | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| **JTAG_W_TDO3** | Write TDO DATA | TX: | 0x13 | N | 6 | **0x30** | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| | Bits [127:96] | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TDI3** | Read TDI DATA | TX: | 0x13 | N | 2 | **0x31** | -- | -- | -- | -- |
| | Bits [127:96] | RX: | 0x13 | N | 6 | Flag | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| **JTAG_W_TMS0** | Write TMS DATA | TX: | 0x13 | N | 6 | **0x40** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | Bits [31:0] | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TMS0** | Read TMS DATA | TX: | 0x13 | N | 2 | **0x41** | -- | -- | -- | -- |
| | Bits [31:0] | RX: | 0x13 | N | 6 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| **JTAG_W_TMS1** | Write TMS DATA | TX: | 0x13 | N | 6 | **0x50** | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| | Bits [63:32] | RX: | 0x13 | N | 3 | Flag | -- | -- | -- | -- |
| **JTAG_R_TMS1** | Read TMS DATA | TX: | 0x13 | N | 2 | **0x51** | -- | -- | -- | -- |
| | Bits [63:32] | RX: | 0x13 | N | 6 | Flag | D[63:56] | D[55:48] | D[47:40] | D[39:32] |
| **JTAG_W_TMS2** | Write TMS DATA | TX: | 0x13 | N | 6 | **0x60** | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| | Bits [95:64] | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TMS2** | Read TMS DATA | TX: | 0x13 | N | 2 | **0x61** | -- | -- | -- | -- |
| | Bits [95:64] | RX: | 0x13 | N | 6 | Flag | D[95:88] | D[87:80] | D[79:72] | D[71:64] |
| **JTAG_W_TMS3** | Write TMS DATA | TX: | 0x13 | N | 6 | **0x70** | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| | Bits [31:0] | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_R_TMS3** | Read TMS DATA | RX: | 0x13 | N | 2 | **0x71** | -- | -- | -- | -- |
| | Bits [127:96] | RX: | 0x13 | N | 6 | Flag | D[127:120] | D[119:112] | D[111:104] | D[103:96] |
| **JTAG_ARESET** | Send RESET pulse | TX: | 0x13 | N | 2 | **0xC0** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_GO** | Start transmission | TX: | 0x13 | N | 2 | **0xA2** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |
| **JTAG_GO_M** | Start transmission | TX: | 0x13 | N | 2 | **0xB0** | -- | -- | -- | -- |
| | | RX: | 0x13 | N | 2 | Flag | -- | -- | -- | -- |

Table 8.10: JTAG channel command list

## 8.4. IO PADS

| PAD NAME | DIRECTION | MAX CURRENT | VOLTAGE RANGE | DESCRIPTION |
|---|---|---|---|---|
| **TCK** | Output | 12mA | 0V-DVDD | JTAG Clock line pad |
| **TDO** | Output | 12mA | 0V-DVDD | JTAG Serial data master to slave pad |
| **TDI** | Input | 12mA | 0V-DVDD | JTAG Serial data slave to master pad |
| **TMS** | Output | 12mA | 0V–DVDD | JTAG Serial command master to slave pad |
| **ARES** | Output | 4mA | 0V–DVDD | JTAG asynchronous reset pad |

Table 8.11: JTAG pad list

# 9. PARALLEL INTERFACE (GPIO)

The GBT-SCA include a Parallel Interface Adapter (GPIO channel) featuring 32 General Purpose digital IO lines with the following features:

- Each line can be individually programmed as output or input (tristate mode).
- Input/output signals can be sampled at the raising or at the falling edges of the clock.
- Input/output signals can be synchronous with the system clock or of an external strobe signal from the user's application connected on a dedicated input line.
- Any line configured as input can be individually programmed to generate an interrupt request to the control room electronics.

The interface appears as memory mapped. Operations are effective at the write operation on the registers. Each operation is acknowledging with a reply packet. The GPIO channel can be disabled to reduce the power consumption.

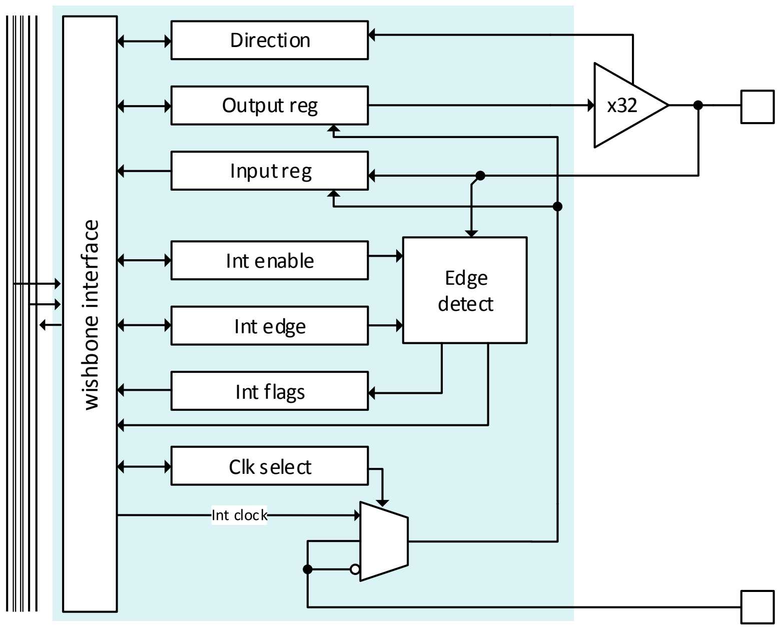


Figure 9.1: Block diagram of the parallel interface channel

## 9.1. CONFIGURATION REGISTERS

The following registers are defined in the GPIO channel interface:

| REGISTER | MODE | FUNCTION |
|---|---|---|
| DATAOUT | r/w | Data OUTPUT register |
| DATAIN | r/w | Data INPUT register |
| DIRECTION | r/w | Input/output select register |
| INTENABLE | r/w | Interrupt enable register |
| INTSEL | r/w | Interrupt line select |
| INTTRIG | r/w | Event edge select for interrupt generation |
| INTS | r/w | Interrupt vector register |
| CLKSEL | r/w | Clock select register |
| EDGESEL | r/w | Clock edge select register |

Table 9.1: GPIO channel configuration registers

### DATAOUT register

The **DATAOUT** register drives the general purpose pads, if set as outputs through the **DIRECTION** reg.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **DATAOUT** | Output bit vector |

*The reset value is 0x00.

Table 9.2: GPIO channel DATAOUT register

### DATAIN register

The **DATAIN** register store the general-purpose inputs.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **DATAIN** | Input bit vector |

*The reset value is 0x00.

Table 9.3: GPIO channel DATAIN register

### DIRECTION register

The **DIRECTION** register select the input or output mode of operation for each I/O.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **DIRECTION** | Input / output mode select for each bit:<br>Bit set to ‘1’ -> Corresponding pad in output mode<br>Bit set to ‘0’ -> Corresponding pad in input mode |

*The reset value is 0x00 -> ALL INPUTS

Table 9.4: GPIO channel DIRECTION register

## INTENABLE register

The **INTENABLE** single-bit register enable the interrupt generation for the GPIO channel:

| BIT | NAME | FUNCTION |
|---|---|---|
| 0 | **INTENABLE** | If '0' -> interrupt generation is disabled<br>If '1' -> interrupt generation is enabled |

*The reset value is 0.

Table 9.5: GPIO channel INTENABLE register

## INTSEL register

The **INTSEL** register defines which general-purpose inputs generate an interrupt to the host. When bit is set, the corresponding general-purpose input can generate an interrupt.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **INTSEL** | Enables for of interrupts generated by general-purpose input signals<br>Bit set to '1' -> corresponding input generates interrupts<br>Bit set to '0' -> corresponding input does not generate interrupts |

*The reset value is 0x00.

Table 9.6: GPIO channel INTSEL register

## INTTRIG register

The **INTTRIG** register defines which edge of a general-purpose input generates an interrupt. Generation of an interrupt must be first enabled in **INTSEL** register and global interrupt enabled in the **INTENABLE** register. When bit is set, corresponding input generates an interrupt when positive edge is encountered. When bit is cleared, corresponding input generates an interrupt when negative edge is encountered.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **INTTRIG** | Triggering of an interrupt<br>Bit set to '1' -> corresponding input generates interrupts on clock rising edge<br>Bit set to '0' -> corresponding input does not generate on the clock falling edge |

*The reset value is 0x00.

Table 9.7: GPIO channel INTTRIG register

### CLKSEL register

The **CLKSEL** define if the input/output signals are latched with the internal system clock or the external strobe signal provided on the GPIO_CLK input pad.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **CLKSEL** | Bit set to '0' -> corresponding input use internal 40MHz clock<br>Bit set to '1' -> corresponding input use external strobe signal |

*The reset value is 0x00.

Table 9.8: GPIO channel CLKSEL register

### EDGESEL register

The **EDGESEL** defines the external strobe edge used to latch the data. When a bit of the **EDGESEL** register is set, the value on the corresponding general-purpose pad, if declared as input, is latched on the negative edge of the external reference signal provided on the opposite pad. Otherwise, the input value is sampled in correspondence of the positive edge.

Clearly, the value in this register has no effect if the corresponding bit in the **CLKSEL** register is set to '0'. In this case infect, the data will be simply sampled on the positive edge of the internal 40MHz clock.

| BIT | NAME | FUNCTION |
|---|---|---|
| 31:0 | **EDGESEL** | Bit set to '0' -> corresponding input sampled on rising edge<br>Bit set to '1' -> corresponding input sampled on negative edge |

*The reset value is 0x00.

Table 9.9: GPIO channel EDGESEL register

## 9.2. OPERATION DESCRIPTION

### Parallel interface as input

To use general-purpose I/O as input only, corresponding bit in **DIRECTION** register must be cleared to select input mode. The **INTENABLE** register and corresponding bit in **INTSEL** register must be cleared as well, to disabled generation of interrupts. The **DATAIN** register reflects registered value of general-purpose input signal.

The value in the **DATAIN** register is updated on the positive edge of system clock if **CLKSEL** appropriate bit is set to '0'. If the bit is set to '1' instead the **DATAIN** register is updated on the edge of the external strobe signal selected by the **EDGESEL** register.

### Parallel interface as output

To enable general-purpose I/O output driver, the corresponding bit in the **DIRECTION** register must be set to ‘1’. The **DATAOUT** register must be written with the value that will drive the output pads. The corresponding bits in **INTSEL** register must be cleared to disable generation of spurious interrupts.

### Parallel interface in interrupt mode

To use general-purpose I/O as input with generation of interrupts, the corresponding bit in **DIRECTION** register must be set to ‘0’ to select the input mode and the **INTSEL** register needs to be set to enable the interrupt generation on the required pad.

To select to generate an interrupt respectively on the rising or falling edge of the input signal, the **INTTRIG** register bits can be set to ‘0’ or to ‘1’ in order trigger an interrupt respectively on the rising or falling edge of the input signal. At this point the global interrupt enable register (**INTENABLE**) must be set to ‘1’. When the interrupt condition is encountered, an interrupt packet is spontaneously generated by the SCA. It is recognizable by his transaction ID equal to 0xFF. The 32 bits data field associated corresponds to the input configuration that has triggered the interrupt. The **INTS** register keeps the latest interrupt vector, in case needs to be read later. Table 9.10: Example of Interrupt packet generated by the GPIO channel

gives an example of interrupt packet.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **RX:** | GPIO 0x02 | **0xFF** | 4 | Err flag 0x00 always | **INTS** [31:24] | **INTS** [23:16] | **INTS** [15:8] | **INTS** [7:0] |

Table 9.10: Example of Interrupt packet generated by the GPIO channel

## 9.3. COMMANDS TABLE

Table 9.11 summarizes the commands accepted by the GPIO channel.

| COMMAND | FUNCTION | REQUEST AND REPLY FORMATS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | TYPE | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_W_DATAOUT | Write register DATAOUT | TX: | 0x02 | N | 4 | **0x10** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_DATAOUT | Read register DATAOUT | TX: | 0x02 | N | 1 | **0x11** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_R_DATAIN | Read register DATAIN | TX: | 0x02 | N | 1 | **0x01** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| GPIO_W_DIRECTION | Write register DIRECTION | TX: | 0x02 | N | 4 | **0x20** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_DIRECTION | Read register DIRECTION | TX: | 0x02 | N | 1 | **0x21** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | | | | | | | | | |
| GPIO_W_INTENABLE | Write register INTENABLE | TX: | 0x02 | N | 4 | **0x60** | -- | -- | -- | D[0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_INTENABLE | Read register INTENABLE | TX: | 0x02 | N | 1 | **0x61** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | -- | -- | -- | D[0] |
| GPIO_W_INTSEL | Write register INTSEL | TX: | 0x02 | N | 4 | **0x30** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_INTSEL | Read register INTSEL | TX: | 0x02 | N | 1 | **0x31** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_W_INTTRIG | Write register INTTRIG | TX: | 0x02 | N | 4 | **0x40** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_INTTRIG | Read register INTTRIG | TX: | 0x02 | N | 1 | **0x41** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_W_INTS | Write register INTS | TX: | 0x02 | N | 4 | **0x70** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_INTS | Read register INTS | TX: | 0x02 | N | 1 | **0x71** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_W_ CLKSEL | Write register CLKSEL | TX: | 0x02 | N | 4 | **0x80** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_ CLKSEL | Read register CLKSEL | TX: | 0x02 | N | 1 | **0x81** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| GPIO_W_ EDGESEL | Write register EDGESEL | TX: | 0x02 | N | 4 | **0x90** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x02 | N | 2 | Flag | -- | -- | -- | -- |
| GPIO_R_ EDGESEL | Read register EDGESEL | TX: | 0x02 | N | 1 | **0x91** | -- | -- | -- | -- |
| | | RX: | 0x02 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |

Table 9.11: GPIO channel command list

## 9.4. IO PADS

| PAD NAME | DIRECTION | MAXIMUM CURRENT | VOLTAGE VALUE | DESCRIPTION |
|---|---|---|---|---|
| **GPIO[31:0]** | Input/Output | 4mA | 0V-DVDD | Bidirectional general purpose I/O pad |
| **GPIOTCK** | Input | 4mA | 0V-DVDD | Input clock signal |

Table 9.12: GPIO pad list

# 10. DAC CHANNEL

The DAC channel implements four Digital to analog converters, independently configurable with a resolution of 8 bits.

- 4 independent digital to analog converters.
- 8 bit resolution.
- Voltage output range 0.0 V to 1.0 V (in theory, refer to Image 10.1).

## 10.1. REGISTERS

The following registers are defined in the DAC channel interface:

| REGISTER | MODE | SIZE | FUNCTION |
|---|---|---|---|
| DAC-A | r/w | 7:0 | Set the analog voltage level on DAC output A |
| DAC-B | r/w | 7:0 | Set the analog voltage level on DAC output B |
| DAC-C | r/w | 7:0 | Set the analog voltage level on DAC output C |
| DAC-D | r/w | 7:0 | Set the analog voltage level on DAC output D |

Table 10.1: DAC configuration registers

## 10.2. OPERATION DESCRIPTION

### Set the output value

**DAC_W_<X>** commands allow setting the analog output voltage on the corresponding channel. The value should be expressed in multiple of the resolution of the DAC:
Value 0x00 -> 0.0 V     Value 0xFF -> 1.25 V as maximum

Even if the theoretical DAC maximum output voltage is 1.0 V, the specification for the DAC output voltage range has to be higher than 1 V over all conditions. The Image 10.1 shows a Monte Carlo simulation of the SCA's DAC output voltage. The Image 10.2 shows a SCA's DAC-ADC loopback over 10 different DACs and ADCs.

Any write operation is immediately acknowledged by the SCA with a reply packet. The output voltage value is maintained until the next write operation.

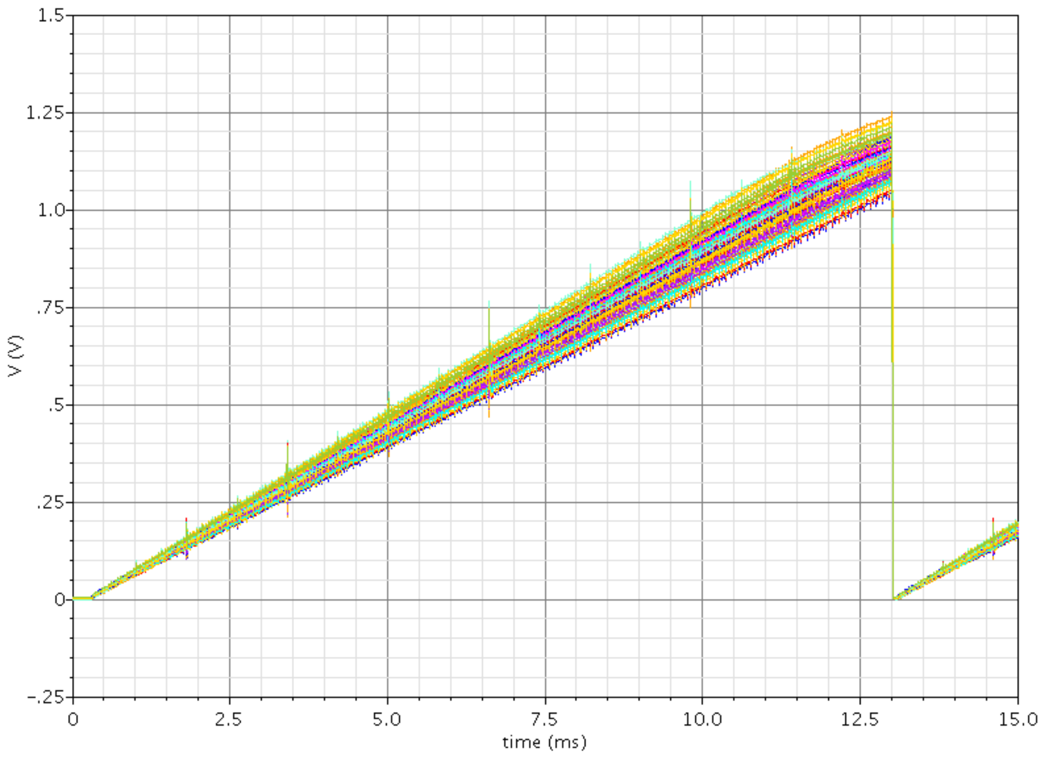


Image 10.1: DAC output voltage range simulation

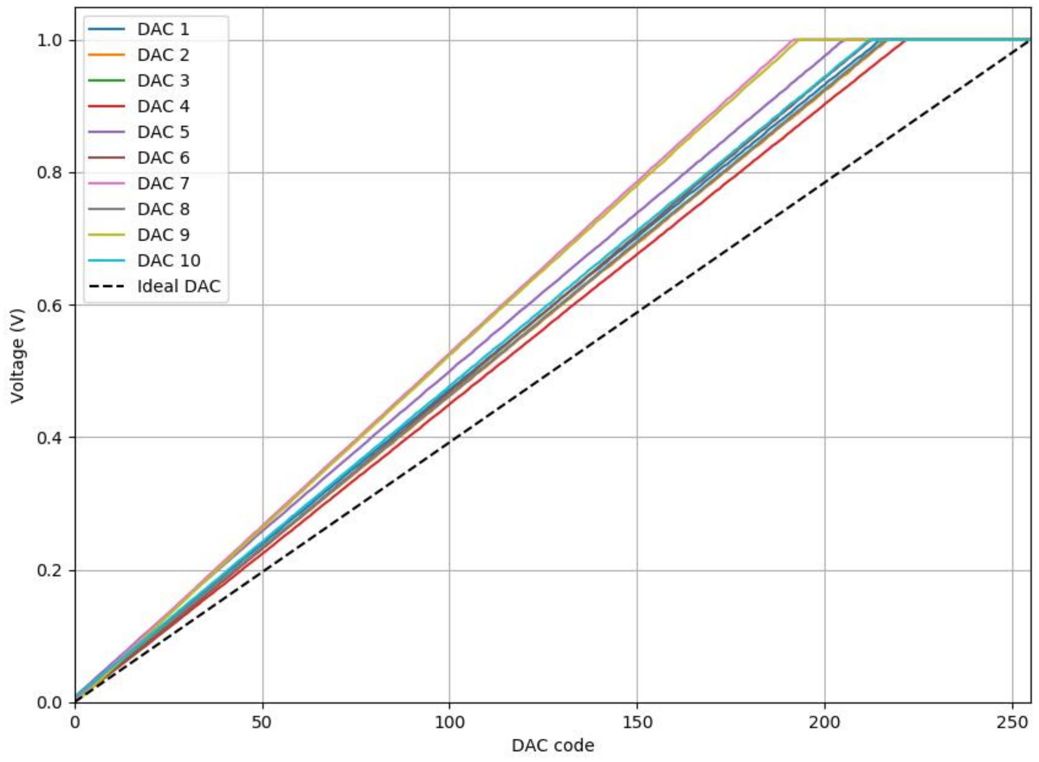


Image 10.2: SCA's DAC-ADC loopback.

### Read back the output value

**DAC_R_<X>** commands allow reading back the value previously set on the corresponding DAC.

## 10.3. COMMANDS TABLE

Table 10.2 summarizes the commands accepted by the DAC channel.

| COMMAND | FUNCTION | REQUEST AND REPLY FORMATS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | TYPE | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| DAC_W_A | Set value on output A | TX: | 0x15 | N | 4 | **0x10** | D[31:24] | -- | -- | |
| | | RX: | 0x15 | N | 1 | Flag | -- | -- | -- | -- |
| DAC_R_A | Read the value of output A | TX: | 0x15 | N | 1 | **0x11** | -- | -- | -- | -- |
| | | RX: | 0x15 | N | 4 | Flag | D[31:24] | -- | -- | -- |
| DAC_W_B | Set value on output B | TX: | 0x15 | N | 4 | **0x20** | D[31:24] | -- | -- | -- |
| | | RX: | 0x15 | N | 1 | Flag | -- | -- | -- | -- |
| DAC_R_B | Read the value of output B | TX: | 0x15 | N | 1 | **0x21** | -- | -- | -- | -- |
| | | RX: | 0x15 | N | 4 | Flag | D[31:24] | -- | -- | -- |
| DAC_W_C | Set value on output C | TX: | 0x15 | N | 4 | **0x30** | D[31:24] | -- | -- | -- |
| | | RX: | 0x15 | N | 1 | Flag | -- | -- | -- | -- |
| DAC_R_C | Read the value of output C | TX: | 0x15 | N | 1 | **0x31** | -- | -- | -- | -- |
| | | RX: | 0x15 | N | 4 | Flag | D[31:24] | -- | -- | -- |
| DAC_W_D | Set value on output D | TX: | 0x15 | N | 4 | **0x40** | D[31:24] | -- | -- | -- |
| | | RX: | 0x15 | N | 1 | Flag | -- | -- | -- | -- |
| DAC_R_D | Read the value of output D | TX: | 0x15 | N | 1 | **0x41** | -- | -- | -- | -- |
| | | RX: | 0x15 | N | 4 | Flag | D[31:24] | -- | -- | -- |

Table 10.2: DAC channel command list

## 10.4. IO PADS

| PAD NAME | DIRECTION | MAXIMUM CURRENT | VOLTAGE VALUE | DESCRIPTION |
|---|---|---|---|---|
| **DAC-0** | Output | | 0V-AVDD | Analog output corresponding to DAC A |
| **DAC-1** | Output | | 0V-AVDD | Analog output corresponding to DAC B |
| **DAC-2** | Output | | 0V-AVDD | Analog output corresponding to DAC C |
| **DAC-3** | Output | | 0V-AVDD | Analog output corresponding to DAC D |

Table 10.3: DAC pad list

# 11. ADC CHANNEL

Features summary:

- 12-bit analog to digital converter.
- 31 analog inputs multiplexed.
- Maximum conversion time of 150 µs (depending on the input voltage).
- |DNL| < 0.4 LSB and |INL| < 2 LSB.
- Embedded temperature sensor.
- Internal voltage reference.
- All inputs feature a switchable 100 uA current source to facilitate the use of externally connected resistance temperature sensors (RTD).
- Internal gain correction and offset cancellation.
- Analog input range:  0.0 V to 1.0 V.
- 12-bit analog to digital converter.
- Maximum quantization error is 1 LSB.
- Operating temperature range: -30 °C to +80 °C.
- Implements internal Gain correction and Offset cancellation.
- Pre-Calibrated in production.
- Applications: Detector leakage current measurement, temperatures, supply voltages

The ADC_IP_3V2 is a 12 bit single slope ADC capable of measuring input voltages from 0 to 1V and temperatures from -30 °C to 80 °C. It includes 31 analog inputs and an embedded temperature sensor multiplexed to an integrative analog to digital converter with automatic offset calibration and gain error correction. All inputs feature a switchable 100 uA current source to facilitate the use of externally connected resistance temperature sensors. Communication with the IP is done through the Wishbone interface and dedicated signals. The ADC core incorporates an analog multiplexer, an automatic offset cancelation circuit and a voltage ramp generation circuit. ADC parameters are stored in the eFuses bank. The IP also includes a bandgap and biasing circuitry. The ADC achieves a |DNL| < 0.4 LSB and |INL| < 2LSB and the power consumption remains below 1.35mW.

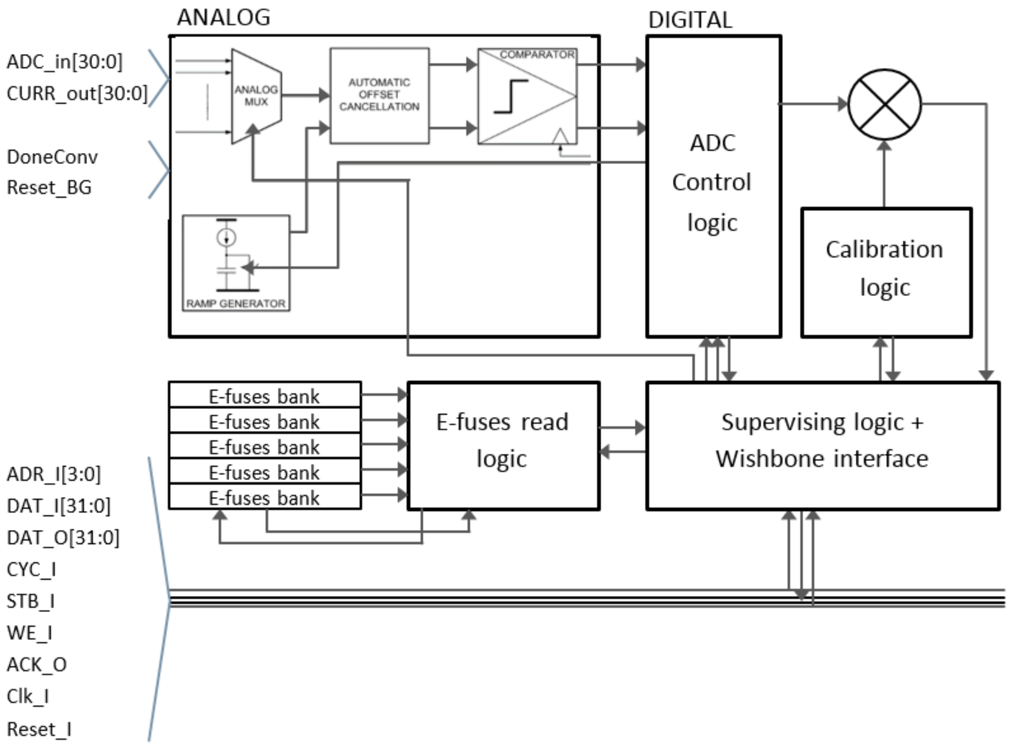


Figure 11.1: Block diagram of the ADC channel

The ADC IP implements an auto-calibrating procedure for the offset cancellation at each conversion. This procedure consists in running a conversion cycle with the input of the comparator shorted to the reference analog ground, switching the comparator inputs in case the offset results to be negative. The ADC IP also implements internal gain correction. Considering that the time needed by the comparator to take a decision is much smaller than the clock cycle and that the decision of the comparator does not depend on the value of the reference input but only on the difference between the two inputs, the full-scale voltage associated with the code 4096 is given by : $V_{FullScale} = Ic(2^N - 1)t_{STEP}$. For process variation in the capacitor or in the bandgap, the gain will have an error. For this reason, the ADC was designed to have the possibility to correct the gain by multiplying the result with a correction factor. In order to avoid missing codes, the ADC was designed to have a gain always lower than the nominal one in all corners, so that the result of conversion is always multiplied with a factor smaller than 1. The correction factor is 12 bit length, representing the decimal part.

## 11.1. CONFIGURATION REGISTERS

The following registers are defined in the ADC channel interface:

| REGISTER | MODE | FUNCTION |
|---|---|---|
| MUX | r/w | Input multiplexer control register |
| CURR | r/w | Current source control register |
| GAIN | r/w | Gain calibration factor register |

Table 11.1: ADC channel configuration registers

### Input select – MUX register

The **MUX** control register allows to select the active input line. Lines from 0 to 31 are associated to the 31 analog input pads. Line 31 select the internal temperature sensor. To write or read the control register use the commands: ADC_W_MUX and ADC_R_MUX as defined in Table 11.6.

| BIT | NAME | FUNCTION |
|---|---|---|
| 5:0 | **INSEL** | Input line select (mux control) |

*The reset value of this register is 0x00.

Table 11.2: ADC channel MUX register

### Current source enable – CURR register

Each bit of the **CURR** register enable/disable the current generator on the corresponding pad. A 100 µA current is present only on the pad under measurement (the input line selected by the MUX control register), if the corresponding bit in the CURR enable register is high. To write or read the control register use ADC_W_CURR and ADC_R_CURR commands as defined in Table 11.6.

| BIT | NAME | FUNCTION |
|---|---|---|
| 30:0 | **CURR** | Current source enable<br>Bit at '0' -> current source disabled on the corresponding pad<br>Bit at '1' -> 100 µA current source enable on the corresponding pad |

*The reset value of this register is 0x00.

Table 11.3: ADC channel CURR register

### Gain calibration register – GAIN register

The **GAIN** register contain the calibration factor for the ADC. At start-up of the chip or after a chip-reset, it is preloaded with the correct calibration value, calculated in production phase and kept in the internal e-fuses. The user has the possibility to read and to override this value if needed.

To write or read the control register use the commands: ADC_W_GAIN and ADC_R_GAIN as defined in

Table 11.6.

| BIT | NAME | FUNCTION |
|---|---|---|
| 15:0 | **GAIN** | Calibration factor |

*The reset value of this register is the calibration value computed in production.

Table 11.4: ADC channel GAIN register

## 11.2. FUNCTIONAL DESCRIPTION

### Start of conversion

The **ADC_GO** command allows to starts the analog to digital conversion on the specified input line selected by the MUX register. During the conversion time, the access to the channel is denied by the SCA. Any command directed to the ADC channel will get an immediate response with a busy flag. At the end of the conversion cycle, the SCA replies with an acknowledge packet containing, in the data field, the conversion result expressed on 12 bits. This result is already corrected from offset and gain errors. The following table gives an example of operation.

| | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
|---|---|---|---|---|---|---|---|---|
| **TX:** | ADC channel 0x14 | N | 4 | ADC_GO 0x02 | -- | -- | -- | 0x01 |
| **RX:** | ADC channel 0x14 | N | 4 | Err flag | -- | -- | D[11:8] | D[7:0] |

Table 11.5: Start of analog to digital conversion

### Get additional conversion information

| | |
|---|---|
| **ADC_R_DATA** | command allows to read the value of the latest conversion. |
| **ADC_R_RAW** | command allows to read the raw value of the conversion, therefore without any offset or gain correction. |
| **ADC_R_OFS** | command allows to read the value of the offset evaluated during the latest conversion |

## 11.3. COMMANDS TABLE

The following table summarizes the commands accepted by the ADC channel for operations on its registers.

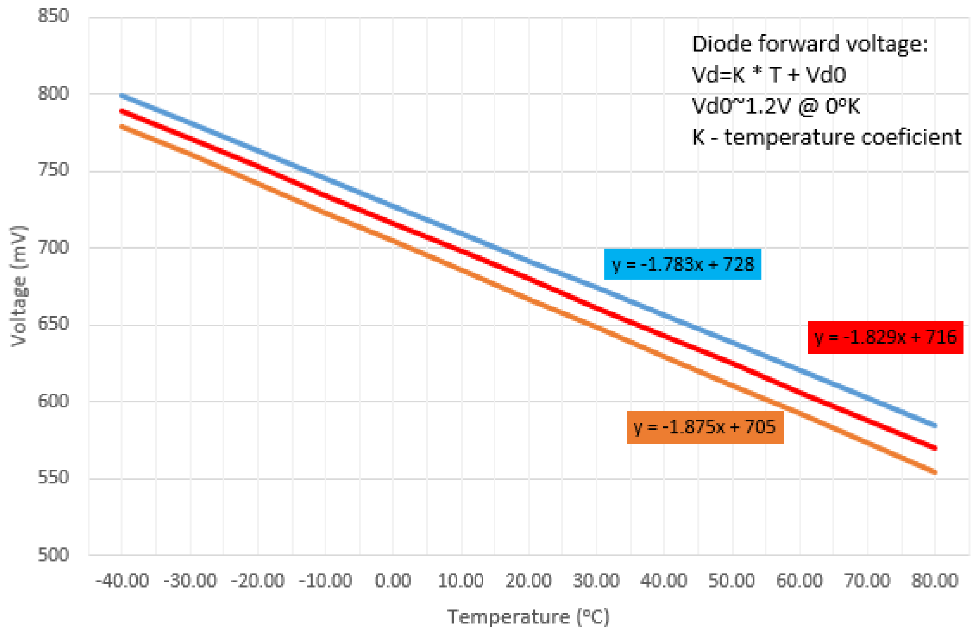
| COMMAND | FUNCTION | REQUEST AND REPLY FORMATS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | TYPE | CH | TRN | LEN | CMD/ER | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| ADC_GO | Start of conversion | TX: | 0x14 | N | 4 | **0x02** | -- | -- | -- | 1 |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | D[11:8] | D[7:0] |
| ADC_W_MUX | Write register INSEL | TX: | 0x14 | N | 4 | **0x50** | -- | -- | -- | D[4:0] |
| | | RX: | 0x14 | N | 1 | Flag | -- | -- | -- | -- |
| ADC_R_MUX | Read register INSEL | TX: | 0x14 | N | 1 | **0x51** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | -- | D[4:0] |
| ADC_W_CURR | Write register | TX: | 0x14 | N | 4 | **0x60** | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| | | RX: | 0x14 | N | 1 | Flag | -- | -- | -- | -- |
| ADC_R_CURR | Read register | TX: | 0x14 | N | 1 | **0x61** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| ADC_W_GAIN | Set value on output A | TX: | 0x14 | N | 4 | **0x10** | -- | -- | D[15:8] | D[7:0] |
| | | RX: | 0x14 | N | 1 | Flag | -- | -- | -- | -- |
| ADC_R_GAIN | Read the value of output A | TX: | 0x14 | N | 1 | **0x11** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | D[15:8] | D[7:0] |
| ADC_R_DATA | Set value on output B | TX: | 0x14 | N | 1 | **0x21** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | D[11:8] | D[7:0] |
| ADC_R_RAW | Read the value of output B | TX: | 0x14 | N | 1 | **0x31** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | D[11:8] | D[7:0] |
| ADC_R_OFS | Set value on output C | TX: | 0x14 | N | 1 | **0x41** | -- | -- | -- | -- |
| | | RX: | 0x14 | N | 4 | Flag | -- | -- | D[11:8] | D[7:0] |

Table 11.6: ADC channel command list

## 11.4. INTERNAL TEMPERATURE SENSOR

Port 31 of the ADC input multiplexer is connected to the internal temperature sensor. In order to perform a temperature measurement, is enough to set the **MUX** control register to value 31 and start an ADC conversion with the **ADC_GO** command as described in the previous paragraph.

The **Error! Reference source not found.** shows the relation between the conversion result and the temperature, for different non-calibrated ASICs.



The sensor should not be used for absolute temperature measurements but only for evaluating temperature variations.

## 11.5. ILINEARITY MEASUREMENTS

### Differential non-linearity

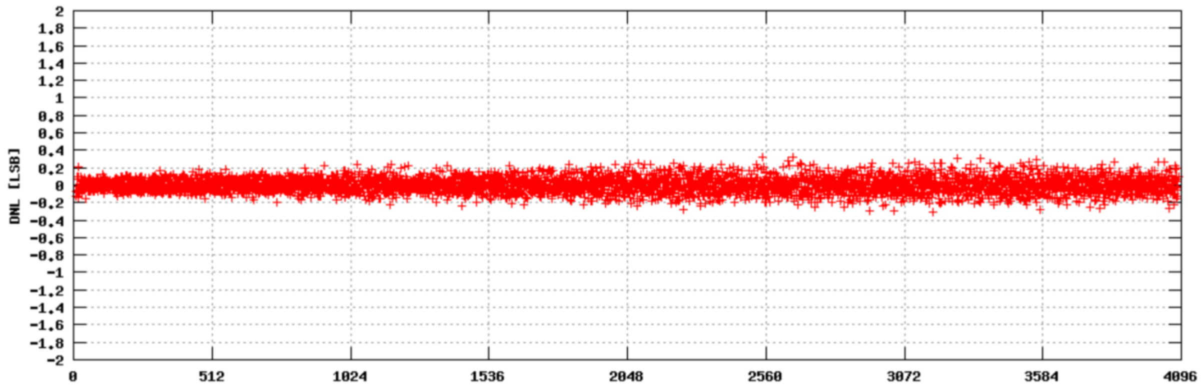


Figure 11.8 – ADC DNL

### Integral non-linearity

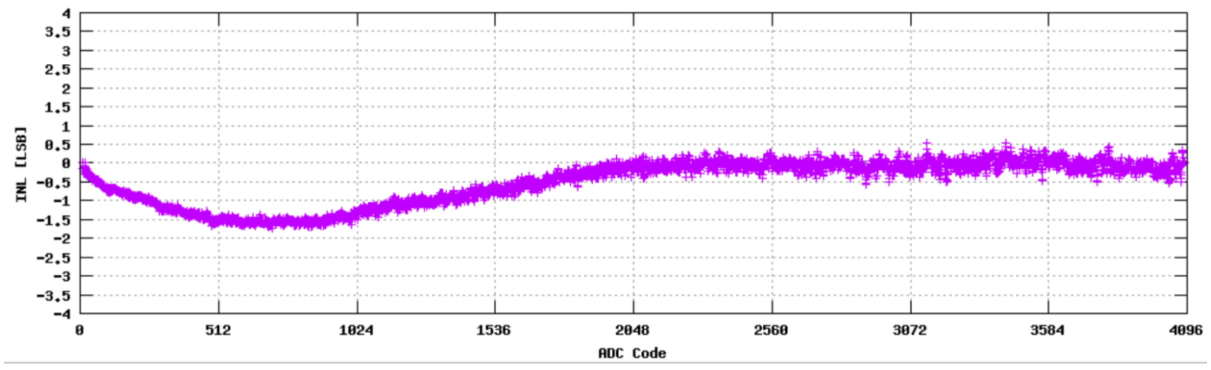


Figure 11.9 – ADC INL

## 11.6. IO PADS

| PAD NAME | DIRECTION | CURRENT SOURCI | VOLTAGE VALUE | DESCRIPTION |
|---|---|---|---|---|
| **ADCIN[30:0]** | Analog Input | 0uA – 100uA | 0.0V < In < 1.0V | Analog input |

Table 11.7: ADC pad list

# 12. ELECTRICAL SPECIFICATIONS

## 12.1. POWER SUPPLIES

Table 12.1 shows the absolute minimum and maximum voltages for the three supplies power the GBT-SCA ASIC.

| NAME | DESCRIPTION | MINIMUM | TYPICAL | MAXIMUM |
|---|---|---|---|---|
| **VDD** | Digital core and e-link drivers/receivers supply | 1.5 V - 10% | **1.5 V** | 1.5 V + 10% |
| **AVDD** | Analog supply (ADC and DAC) | 1.5 V - 10% | **1.5 V** | 1.5 V + 10% |
| **DVDD** | CMOS IO pads supply | 1.5 V - 10% | **1.5 V** | 1.5 V + 10% |

Table 12.1 – Power supply ratings

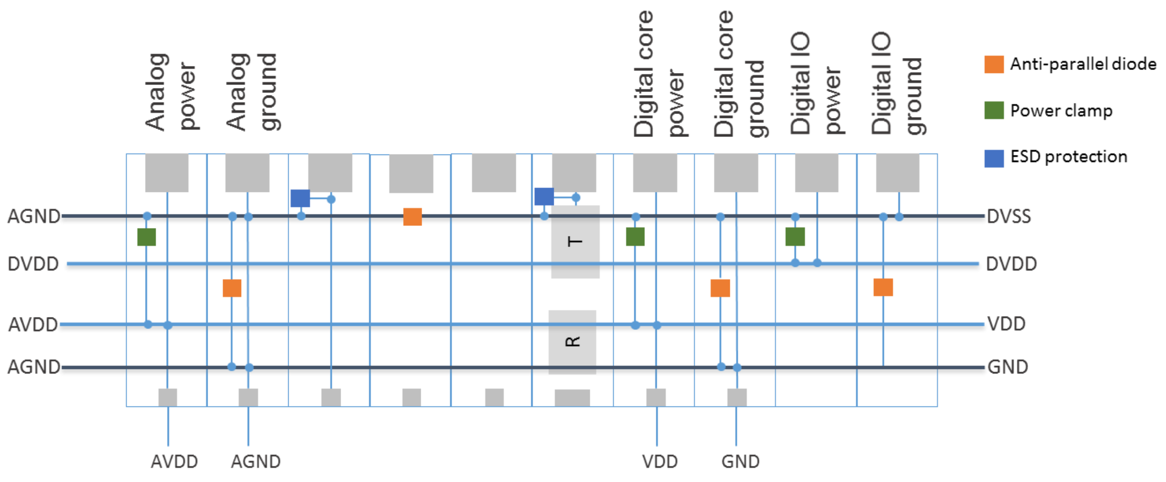


Figure 12.1 – GBT-SCA power distribution scheme

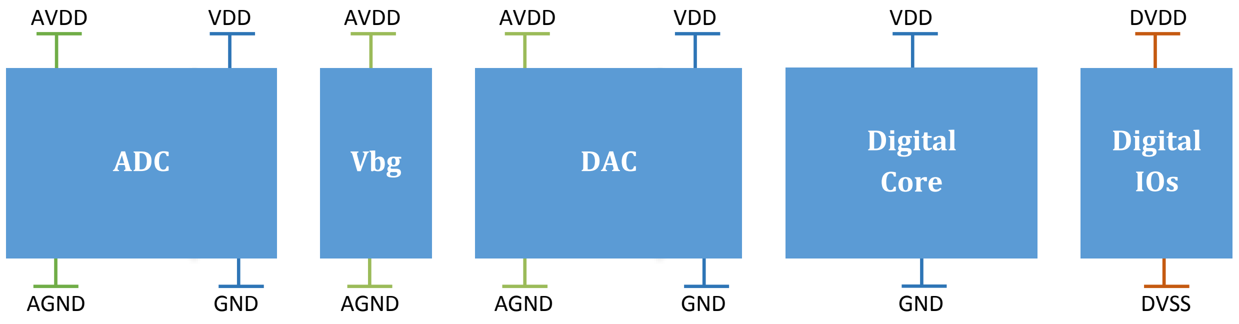


Figure 12.2 – GBT-SCA power supply connectivity

## 12.2. DIGITAL IO PAD CHARACTERISTICS

Table 13.2 shows the electrical specifications for the following IO signals: I2C, GPIO, JTAG and SPI at the nominal power supply voltage DVDD=1.5V. The Imax value for these signals are, Imax(I2C)

| SYMBOL | PARAMETER | CONDITION | MIN. | MAX. |
|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage | CMOS receiver | -0.3V | 0.8V |
| $V_{IH}$ | Input High Voltage | CMOS receiver | | DVDD +0.3V |
| **H** | Hysteresis | -- | +0.3V | +0.5V |
| $V_{OL}$ | Output Low Voltage | $I_{OL} = -I_{MAX}$ | +0.0V | +0.4V |
| $V_{OH}$ | Output High Voltage | $I_{OL} = I_{MAX}$ | | DVDD |

Table 12.2 – IO signals electrical specifications

## 12.3. POWER CONSUMPTION

Measured value at: DVDD = 1.5V, VDD = 1.5V, AVDD = 1.5V

| SUPPLY | TYPICAL | MAXIMUM |
|---|---|---|
| VDD core | 36 mA | 63 mA |
| AVDD analog | 0.5 mA | 0.8 mA |
| DVDD Static supply current | 7.1 mA* | 8.2 mA |

* Static supply current. The IO current depends on the load driven by the pad.
The values in table refers to the case of no load present on the output ports.

Table 12.3 – GBT-SCA power consuption

## 12.4. DECOUPLING

For better performances decouple the three power supplies individually using a 100nF capacitors as closes as possible to the corresponding power pins.

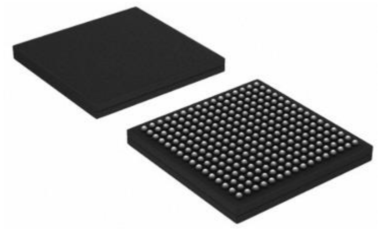
## 12.5. FRONT-END SPECIFICATIONS

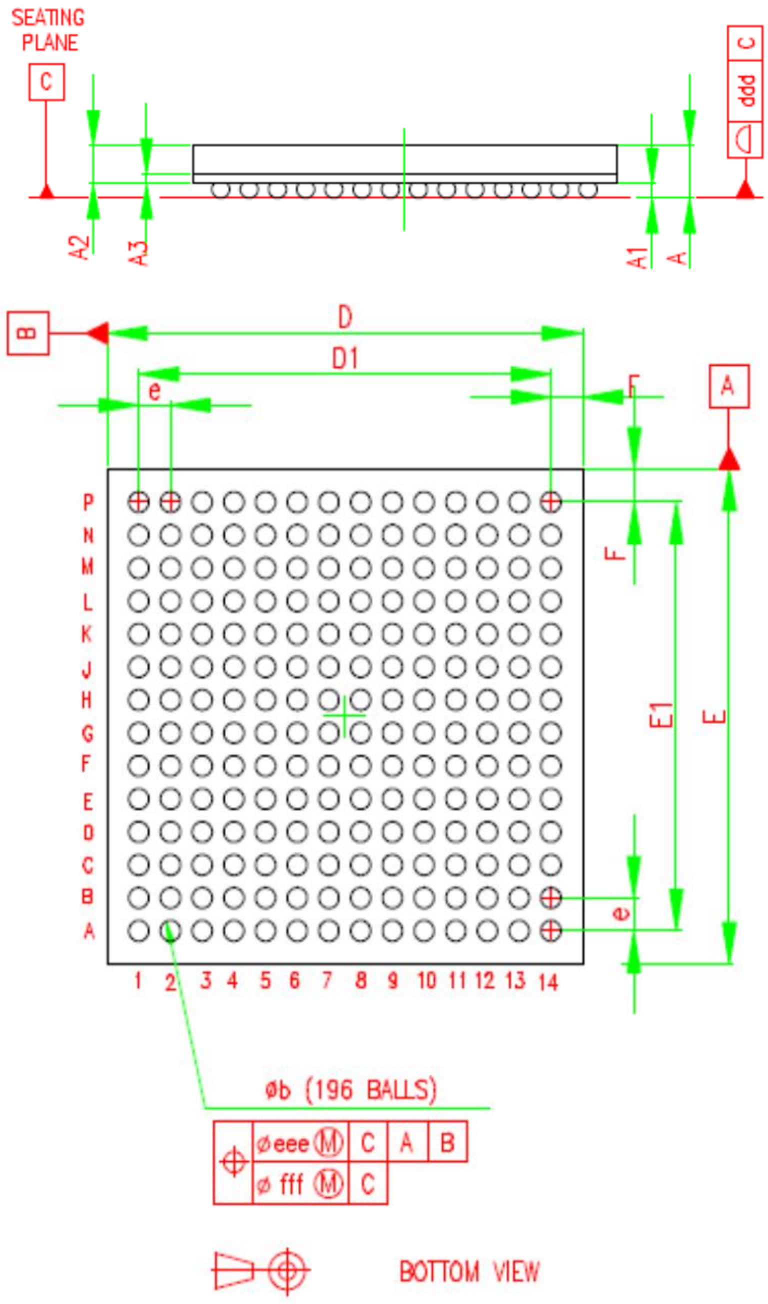
- I2C SDA and SCL lines are pulled-up to 1.5 V. If you want to use the I2C protocol to communicate with your front-end electronics, their I2C lines have to run at 1.5 V too.
- Front-ends that do not have a Schmitt trigger they cannot use the I2C protocol to communicate with the SCA.

# 13. PACKAGE DESCRIPTION

## 13.1. MECHANICAL CHARACTERISTICS

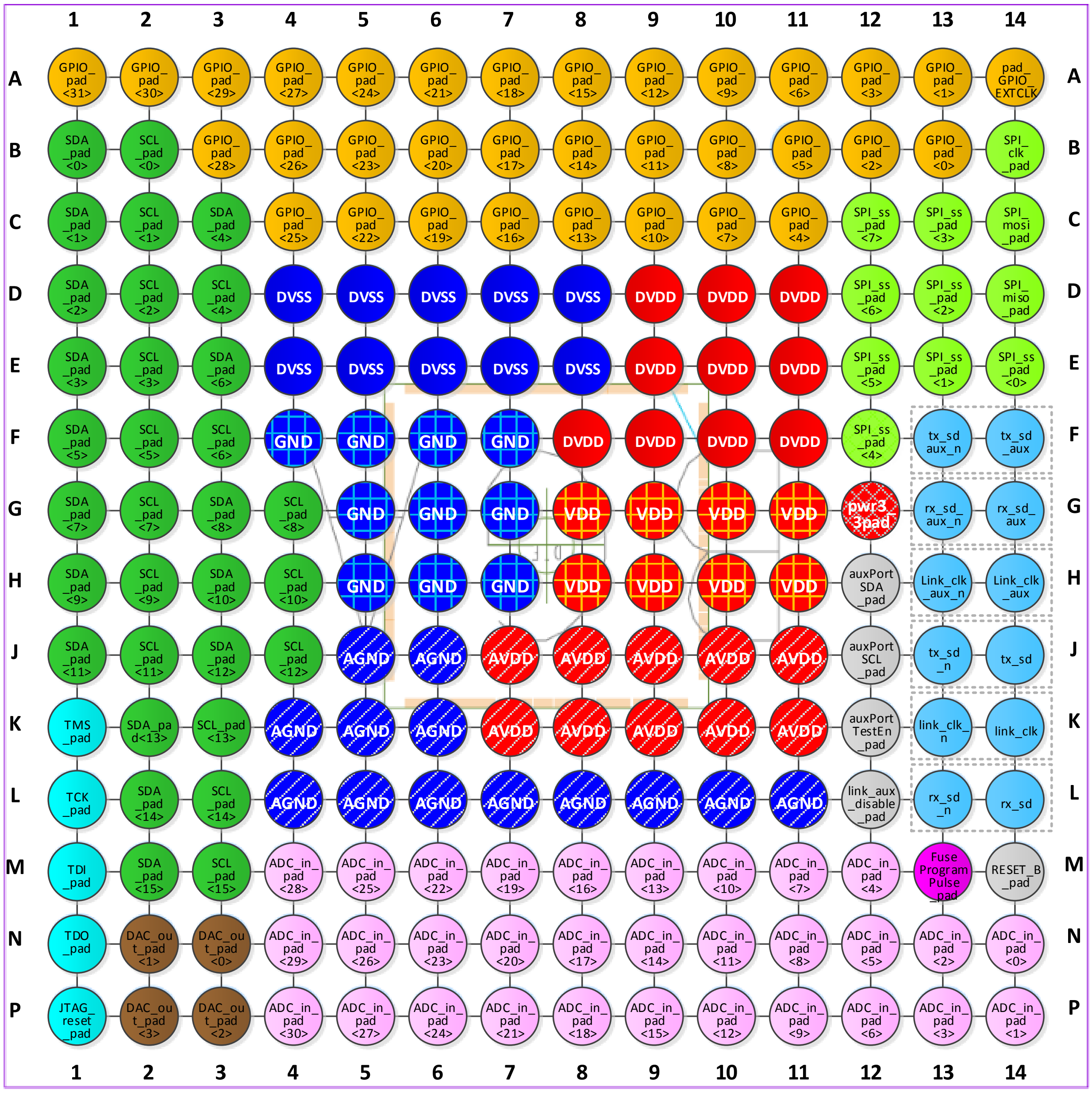
| | |
|---|---|
| **Package Type** | LFBGA |
| **Pitch** | 0.8 |
| **Pin count** | 196 |
| **Ball size** | 0.5mm |





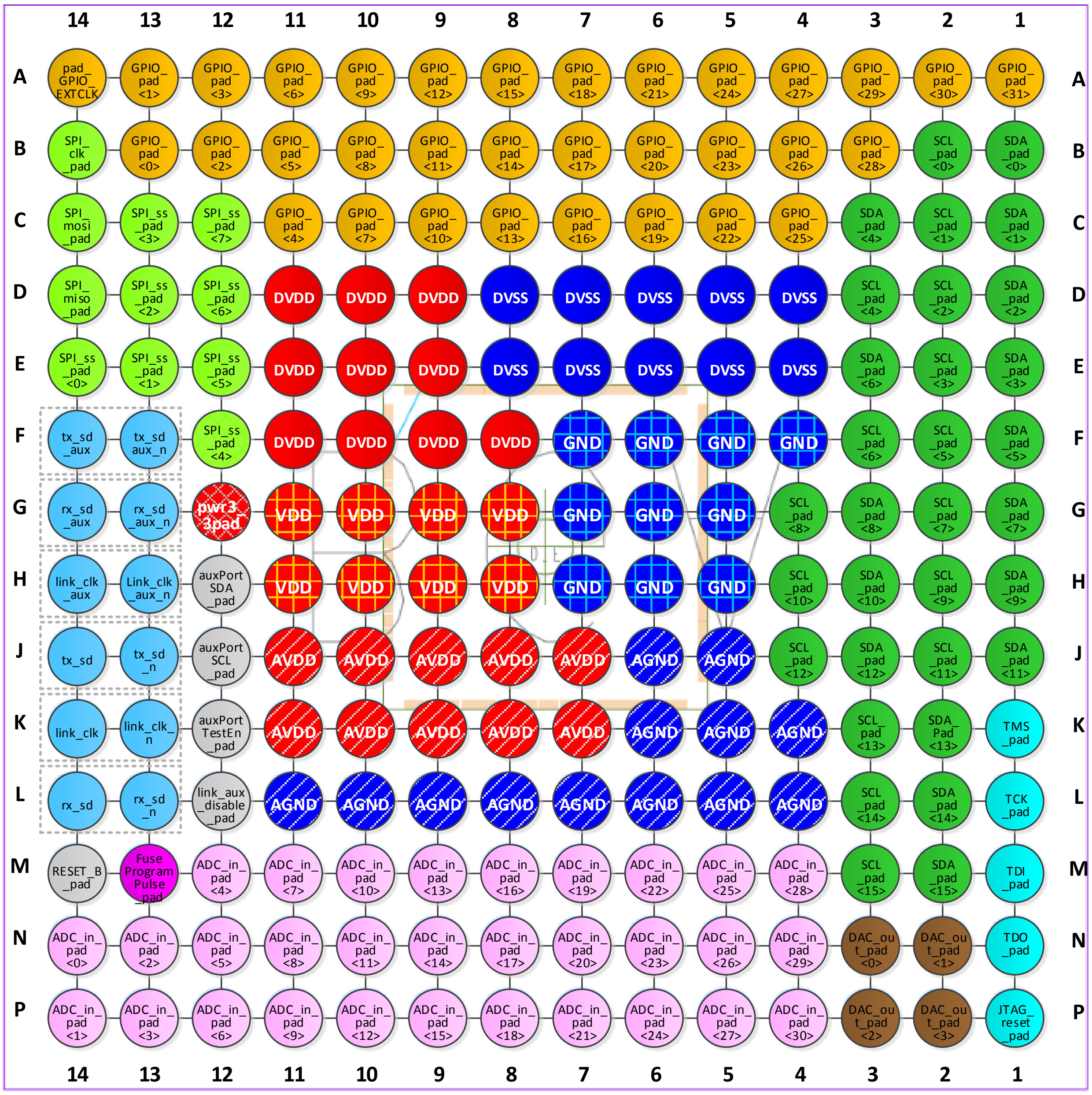
| SYMBOL | MIN | TYP. | MAX |
|---|---|---|---|
| **A** | | | 1.70 |
| **A1** | 0.27 | | |
| **A2** | | 1.08 | |
| **A3** | | 0.28 | |
| **b** | 0.45 | 0.50 | 0.55 |
| **D** | 11.85 | 12.00 | 12.15 |
| **D1** | | 10.40 | |
| **e** | | 0.80 | |
| **E** | 11.85 | 12.00 | 12.15 |
| **E1** | | 10.40 | |
| **F** | | 0.80 | |
| **ddd** | | | 0.12 |
| **eee** | | | 0.15 |
| **fff** | | | 0.08 |

## 13.2. PINOUT TOP VIEW



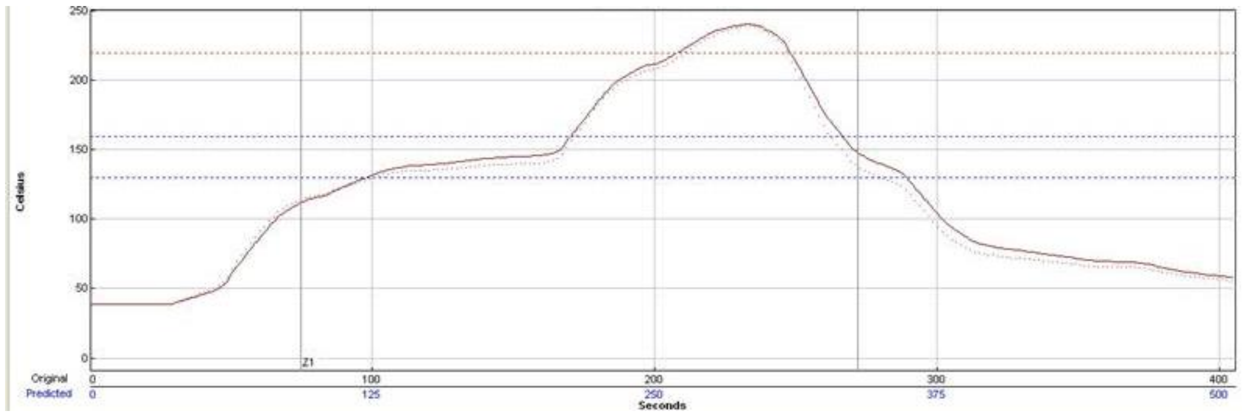
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | GPIO_pad<31> | GPIO_pad<30> | GPIO_pad<29> | GPIO_pad<27> | GPIO_pad<24> | GPIO_pad<21> | GPIO_pad<18> | GPIO_pad<15> | GPIO_pad<12> | GPIO_pad<9> | GPIO_pad<6> | GPIO_pad<3> | GPIO_pad<1> | pad_GPIO_EXTCLK | A |
| B | SDA_pad<0> | SCL_pad<0> | GPIO_pad<28> | GPIO_pad<26> | GPIO_pad<23> | GPIO_pad<20> | GPIO_pad<17> | GPIO_pad<14> | GPIO_pad<11> | GPIO_pad<8> | GPIO_pad<5> | GPIO_pad<2> | GPIO_pad<0> | SPI_clk_pad | B |
| C | SDA_pad<1> | SCL_pad<1> | SDA_pad<4> | GPIO_pad<25> | GPIO_pad<22> | GPIO_pad<19> | GPIO_pad<16> | GPIO_pad<13> | GPIO_pad<10> | GPIO_pad<7> | GPIO_pad<4> | SPI_ss_pad<7> | SPI_ss_pad<3> | SPI_mosi_pad | C |
| D | SDA_pad<2> | SCL_pad<2> | SCL_pad<4> | DVSS | DVSS | DVSS | DVSS | DVSS | DVDD | DVDD | DVDD | SPI_ss_pad<6> | SPI_ss_pad<2> | SPI_miso_pad | D |
| E | SDA_pad<3> | SCL_pad<3> | SDA_pad<6> | DVSS | DVSS | DVSS | DVSS | DVSS | DVDD | DVDD | DVDD | SPI_ss_pad<5> | SPI_ss_pad<1> | SPI_ss_pad<0> | E |
| F | SDA_pad<5> | SCL_pad<5> | SCL_pad<6> | GND | GND | GND | GND | DVDD | DVDD | DVDD | DVDD | SPI_ss_pad<4> | tx_sd aux_n | tx_sd_aux | F |
| G | SDA_pad<7> | SCL_pad<7> | SDA_pad<8> | SCL_pad<8> | GND | GND | GND | VDD | VDD | VDD | VDD | pwr33pad | rx_sd_aux_n | rx_sd_aux | G |
| H | SDA_pad<9> | SCL_pad<9> | SDA_pad<10> | SCL_pad<10> | GND | GND | GND | VDD | VDD | VDD | VDD | auxPort SDA_pad | Link_clk_aux_n | Link_clk_aux | H |
| J | SDA_pad<11> | SCL_pad<11> | SDA_pad<12> | SCL_pad<12> | AGND | AGND | AVDD | AVDD | AVDD | AVDD | AVDD | auxPort SCL_pad | tx_sd_n | tx_sd | J |
| K | TMS_pad | SDA_pad<13> | SCL_pad<13> | AGND | AGND | AGND | AVDD | AVDD | AVDD | AVDD | AVDD | auxPort TestEn_pad | link_clk_n | link_clk | K |
| L | TCK_pad | SDA_pad<14> | SCL_pad<14> | AGND | AGND | AGND | AGND | AGND | AGND | AGND | AGND | link_aux_disable_pad | rx_sd_n | rx_sd | L |
| M | TDI_pad | SDA_pad<15> | SCL_pad<15> | ADC_in_pad<28> | ADC_in_pad<25> | ADC_in_pad<22> | ADC_in_pad<19> | ADC_in_pad<16> | ADC_in_pad<13> | ADC_in_pad<10> | ADC_in_pad<7> | ADC_in_pad<4> | Fuse Program Pulse_pad | RESET_B_pad | M |
| N | TDO_pad | DAC_out_pad<1> | DAC_out_pad<0> | ADC_in_pad<29> | ADC_in_pad<26> | ADC_in_pad<23> | ADC_in_pad<20> | ADC_in_pad<17> | ADC_in_pad<14> | ADC_in_pad<11> | ADC_in_pad<8> | ADC_in_pad<5> | ADC_in_pad<2> | ADC_in_pad<0> | N |
| P | JTAG_reset_pad | DAC_out_pad<3> | DAC_out_pad<2> | ADC_in_pad<30> | ADC_in_pad<27> | ADC_in_pad<24> | ADC_in_pad<21> | ADC_in_pad<18> | ADC_in_pad<15> | ADC_in_pad<12> | ADC_in_pad<9> | ADC_in_pad<6> | ADC_in_pad<3> | ADC_in_pad<1> | P |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |

## 13.3. PINOUT BOTTOM VIEW



| | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | pad_ GPIO_ EXTCLK | GPIO_ pad <1> | GPIO_ pad <3> | GPIO_ pad <6> | GPIO_ pad <9> | GPIO_ pad <12> | GPIO_ pad <15> | GPIO_ pad <18> | GPIO_ pad <21> | GPIO_ pad <24> | GPIO_ pad <27> | GPIO_ pad <29> | GPIO_ pad <30> | GPIO_ pad <31> | A |
| B | SPI_ clk _pad | GPIO_ pad <0> | GPIO_ pad <2> | GPIO_ pad <5> | GPIO_ pad <8> | GPIO_ pad <11> | GPIO_ pad <14> | GPIO_ pad <17> | GPIO_ pad <20> | GPIO_ pad <23> | GPIO_ pad <26> | GPIO_ pad <28> | SCL _pad <0> | SDA _pad <0> | B |
| C | SPI_ mosi _pad | SPI_ss _pad <3> | SPI_ss _pad <7> | GPIO_ pad <4> | GPIO_ pad <7> | GPIO_ pad <10> | GPIO_ pad <13> | GPIO_ pad <16> | GPIO_ pad <19> | GPIO_ pad <22> | GPIO_ pad <25> | SDA _pad <4> | SCL _pad <1> | SDA _pad <1> | C |
| D | SPI_ miso _pad | SPI_ss _pad <2> | SPI_ss _pad <6> | DVDD | DVDD | DVDD | DVSS | DVSS | DVSS | DVSS | DVSS | SCL _pad <4> | SCL _pad <2> | SDA _pad <2> | D |
| E | SPI_ss _pad <0> | SPI_ss _pad <1> | SPI_ss _pad <5> | DVDD | DVDD | DVDD | DVSS | DVSS | DVSS | DVSS | DVSS | SDA _pad <6> | SCL _pad <3> | SDA _pad <3> | E |
| F | tx_sd _aux | tx_sd aux_n | SPI_ss _pad <4> | DVDD | DVDD | DVDD | DVDD | GND | GND | GND | GND | SCL _pad <6> | SCL _pad <5> | SDA _pad <5> | F |
| G | rx_sd _aux | rx_sd _aux_n | pwr3_ 3pad | VDD | VDD | VDD | VDD | GND | GND | GND | SCL _pad <8> | SDA _pad <8> | SCL _pad <7> | SDA _pad <7> | G |
| H | link_clk _aux | Link_clk _aux_n | auxPort SDA _pad | VDD | VDD | VDD | VDD | GND | GND | GND | SCL _pad <10> | SDA _pad <10> | SCL _pad <9> | SDA _pad <9> | H |
| J | tx_sd | tx_sd _n | auxPort SCL _pad | AVDD | AVDD | AVDD | AVDD | AVDD | AGND | AGND | SCL _pad <12> | SDA _pad <12> | SCL _pad <11> | SDA _pad <11> | J |
| K | link_clk | link_clk_ n | auxPort TestEn _pad | AVDD | AVDD | AVDD | AVDD | AVDD | AGND | AGND | AGND | SCL_ pad <13> | SDA_ Pad <13> | TMS _pad | K |
| L | rx_sd | rx_sd _n | link_aux _disable _pad | AGND | AGND | AGND | AGND | AGND | AGND | AGND | AGND | SCL _pad <14> | SDA _pad <14> | TCK _pad | L |
| M | RESET_B _pad | Fuse Program Pulse _pad | ADC_in_ pad <4> | ADC_in_ pad <7> | ADC_in_ pad <10> | ADC_in_ pad <13> | ADC_in_ pad <16> | ADC_in_ pad <19> | ADC_in_ pad <22> | ADC_in_ pad <25> | ADC_in_ pad <28> | SCL _pad <15> | SDA _pad <15> | TDI _pad | M |
| N | ADC_in_ pad <0> | ADC_in_ pad <2> | ADC_in_ pad <5> | ADC_in_ pad <8> | ADC_in_ pad <11> | ADC_in_ pad <14> | ADC_in_ pad <17> | ADC_in_ pad <20> | ADC_in_ pad <23> | ADC_in_ pad <26> | ADC_in_ pad <29> | DAC_ou t_pad <0> | DAC_ou t_pad <1> | TDO _pad | N |
| P | ADC_in_ pad <1> | ADC_in_ pad <3> | ADC_in_ pad <6> | ADC_in_ pad <9> | ADC_in_ pad <12> | ADC_in_ pad <15> | ADC_in_ pad <18> | ADC_in_ pad <21> | ADC_in_ pad <24> | ADC_in_ pad <27> | ADC_in_ pad <30> | DAC_ou t_pad <2> | DAC_ou t_pad <3> | JTAG_ reset _pad | P |
| | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

## 13.4. SOLDERING PROFILE



## 13.5. PINS LIST

| PAD NAME | BALL | TYPE | DESCRIPTION |
|---|---|---|---|
| pad_GPIO_EXTCLK | A-14 | IN | General purpose I/O - strobe |
| auxPortSDA_pad | H-12 | INOUT | Auxiliary I2C Port - SDA pad |
| auxPortSCL_pad | J-12 | IN | Auxiliary I2C Port - SCA pad |
| auxPortTestEn_pad | K-12 | IN | Auxiliary I2C Port - Enable Pad |
| SPI_clk_pad | B-14 | OUT | SPI bus - SCLK pad |
| SPI_mosi_pad | C-14 | OUT | SPI bus - MOSI pad |
| SPI_miso_pad | D-14 | IN | SPI bus - MISO pad |
| SPI_ss_pad<7> | C-12 | OUT | SPI bus - Slave Select n0 pad |
| SPI_ss_pad<6> | D-12 | OUT | SPI bus - Slave Select n1 pad |
| SPI_ss_pad<5> | E-12 | OUT | SPI bus - Slave Select n2 pad |
| SPI_ss_pad<4> | F-12 | OUT | SPI bus - Slave Select n3 pad |
| SPI_ss_pad<3> | C-13 | OUT | SPI bus - Slave Select n4 pad |
| SPI_ss_pad<2> | D-13 | OUT | SPI bus - Slave Select n5 pad |
| SPI_ss_pad<1> | E-13 | OUT | SPI bus - Slave Select n6 pad |
| SPI_ss_pad<0> | E-14 | OUT | SPI bus - Slave Select n7 pad |
| tx_sd_aux_n | F-13 | OUT | Auxiliary E-Port - Transmit Pad (-) |
| tx_sd_aux | F-14 | OUT | Auxiliary E-Port - Transmit Pad (+) |
| rx_sd_aux_n | G-13 | IN | Auxiliary E-Port - Receive Pad (-) |
| rx_sd_aux | G-14 | IN | Auxiliary E-Port - Receive Pad (+) |
| link_clk_aux_n | H-13 | IN | Auxiliary E-Port - Clock Pad (-) |
| link_clk_aux | H-14 | IN | Auxiliary E-Port - Clock Pad (+) |
| link_aux_disable_pad | L-12 | IN | Auxiliary E-Port - Disable Pad |
| tx_sd_n | J-13 | OUT | Primary E-Port - Transmit Pad (-) |
| tx_sd | J-14 | OUT | Primary E-Port - Transmit Pad (+) |
| link_clk_n | K-13 | IN | Primary E-Port - Clock Pad (-) |

| | | | |
|---|---|---|---|
| link_clk | K-14 | IN | Primary E-Port - Clock Pad (-) |
| rx_sd_n | L-13 | IN | Primary E-Port - Receive Pad (-) |
| rx_sd | L-14 | IN | Primary E-Port - Receive Pad (+) |
| FuseProgramPulse_pad | M-13 | IN | E-Fuses Program Pulse Pad |
| pwr3_3pad | G-12 | - | Efuse Program power 3.3V |
| RESET_B_pad | M-14 | IN | Global reset pad - Active Low |
| ADC_in_pad<0> | N-14 | IN | Analog Input n0 |
| ADC_in_pad<1> | P-14 | IN | Analog Input n1 |
| ADC_in_pad<2> | N-13 | IN | Analog Input n2 |
| ADC_in_pad<3> | P-13 | IN | Analog Input n3 |
| ADC_in_pad<4> | M-12 | IN | Analog Input n4 |
| ADC_in_pad<5> | N-12 | IN | Analog Input n5 |
| ADC_in_pad<6> | P-12 | IN | Analog Input n6 |
| ADC_in_pad<7> | M-11 | IN | Analog Input n7 |
| ADC_in_pad<8> | N-11 | IN | Analog Input n8 |
| ADC_in_pad<9> | P-11 | IN | Analog Input n9 |
| ADC_in_pad<10> | M-10 | IN | Analog Input n10 |
| ADC_in_pad<11> | N-10 | IN | Analog Input n11 |
| ADC_in_pad<12> | P-10 | IN | Analog Input n12 |
| ADC_in_pad<13> | M-9 | IN | Analog Input n13 |
| ADC_in_pad<14> | N-9 | IN | Analog Input n14 |
| ADC_in_pad<15> | P-9 | IN | Analog Input n15 |
| ADC_in_pad<16> | M-8 | IN | Analog Input n16 |
| ADC_in_pad<17> | N-8 | IN | Analog Input n17 |
| ADC_in_pad<18> | P-8 | IN | Analog Input n18 |
| ADC_in_pad<19> | M-7 | IN | Analog Input n19 |
| ADC_in_pad<20> | N-7 | IN | Analog Input n20 |
| ADC_in_pad<21> | P-7 | IN | Analog Input n21 |
| ADC_in_pad<22> | M-6 | IN | Analog Input n22 |
| ADC_in_pad<23> | N-6 | IN | Analog Input n23 |
| ADC_in_pad<24> | P-6 | IN | Analog Input n24 |
| ADC_in_pad<25> | M-5 | IN | Analog Input n25 |
| ADC_in_pad<26> | N-5 | IN | Analog Input n26 |
| ADC_in_pad<27> | P-5 | IN | Analog Input n27 |
| ADC_in_pad<28> | M-4 | IN | Analog Input n28 |
| ADC_in_pad<29> | N-4 | IN | Analog Input n29 |
| ADC_in_pad<30> | P-4 | IN | Analog Input n30 |
| DAC_out_pad<0> | N-3 | OUT | Analog Output n0 |
| DAC_out_pad<1> | N-2 | OUT | Analog Output n1 |
| DAC_out_pad<2> | P-3 | OUT | Analog Output n2 |
| DAC_out_pad<3> | P-2 | OUT | Analog Output n3 |
| JTAG_reset_pad | P-1 | OUT | JTAG bus - ARESET pad |
| TDO_pad | N-1 | OUT | JTAG bus - TDO pad |

| | | | |
|---|---|---|---|
| TDI_pad | M-1 | IN | JTAG bus - TDI pad |
| TCK_pad | L-1 | OUT | JTAG bus - TCK pad |
| TMS_pad | K-1 | OUT | JTAG bus - TMS pad |
| SCL_pad<15> | M-3 | OUT | I2C bus n15 - SCL line |
| SDA_pad<15> | M-2 | INOUT | I2C bus n15 - SDA line |
| SCL_pad<14> | L-3 | OUT | I2C bus n14 - SCL line |
| SDA_pad<14> | L-2 | INOUT | I2C bus n14 - SDA line |
| SCL_pad<13> | K-3 | OUT | I2C bus n13 - SCL line |
| SDA_pad<13> | K-2 | INOUT | I2C bus n13 - SDA line |
| SCL_pad<12> | J-4 | OUT | I2C bus n12 - SCL line |
| SDA_pad<12> | J-3 | INOUT | I2C bus n12 - SDA line |
| SCL_pad<11> | J-2 | OUT | I2C bus n11 - SCL line |
| SDA_pad<11> | J-1 | INOUT | I2C bus n11 - SDA line |
| SCL_pad<10> | H-4 | OUT | I2C bus n10 - SCL line |
| SDA_pad<10> | H-3 | INOUT | I2C bus n10 - SDA line |
| SCL_pad<9> | H-2 | OUT | I2C bus n9 - SCL line |
| SDA_pad<9> | H-1 | INOUT | I2C bus n9 - SDA line |
| SCL_pad<8> | G-4 | OUT | I2C bus n8 - SCL line |
| SDA_pad<8> | G-3 | INOUT | I2C bus n8 - SDA line |
| SCL_pad<7> | G-2 | OUT | I2C bus n7 - SCL line |
| SDA_pad<7> | G-1 | INOUT | I2C bus n7 - SDA line |
| SCL_pad<6> | F-3 | OUT | I2C bus n6 - SCL line |
| SDA_pad<6> | E-3 | INOUT | I2C bus n6 - SDA line |
| SCL_pad<5> | F-2 | OUT | I2C bus n5 - SCL line |
| SDA_pad<5> | F-1 | INOUT | I2C bus n5 - SDA line |
| SCL_pad<4> | D-3 | OUT | I2C bus n4 - SCL line |
| SDA_pad<4> | C-3 | INOUT | I2C bus n4 - SDA line |
| SCL_pad<3> | E-2 | OUT | I2C bus n3 - SCL line |
| SDA_pad<3> | E-1 | INOUT | I2C bus n3 - SDA line |
| SCL_pad<2> | D-2 | OUT | I2C bus n2 - SCL line |
| SDA_pad<2> | D-1 | INOUT | I2C bus n2 - SDA line |
| SCL_pad<1> | C-2 | OUT | I2C bus n1 - SCL line |
| SDA_pad<1> | C-1 | INOUT | I2C bus n1 - SDA line |
| SCL_pad<0> | B-2 | OUT | I2C bus n0 - SCL line |
| SDA_pad<0> | B-1 | INOUT | I2C bus n0 - SDA line |
| GPIO_pad<31> | A-1 | INOUT | General purpose I/O pad n31 |
| GPIO_pad<30> | A-2 | INOUT | General purpose I/O pad n30 |
| GPIO_pad<29> | A-3 | INOUT | General purpose I/O pad n29 |
| GPIO_pad<28> | B-3 | INOUT | General purpose I/O pad n28 |
| GPIO_pad<27> | A-4 | INOUT | General purpose I/O pad n27 |
| GPIO_pad<26> | B-4 | INOUT | General purpose I/O pad n26 |
| GPIO_pad<25> | C-4 | INOUT | General purpose I/O pad n25 |
| GPIO_pad<24> | A-5 | INOUT | General purpose I/O pad n24 |

| | | | |
|---|---|---|---|
| GPIO_pad<23> | B-5 | INOUT | General purpose I/O pad n23 |
| GPIO_pad<22> | C-5 | INOUT | General purpose I/O pad n22 |
| GPIO_pad<21> | A-6 | INOUT | General purpose I/O pad n21 |
| GPIO_pad<20> | B-6 | INOUT | General purpose I/O pad n20 |
| GPIO_pad<19> | C-6 | INOUT | General purpose I/O pad n19 |
| GPIO_pad<18> | A-7 | INOUT | General purpose I/O pad n18 |
| GPIO_pad<17> | B-7 | INOUT | General purpose I/O pad n17 |
| GPIO_pad<16> | C-7 | INOUT | General purpose I/O pad n16 |
| GPIO_pad<15> | A-8 | INOUT | General purpose I/O pad n15 |
| GPIO_pad<14> | B-8 | INOUT | General purpose I/O pad n14 |
| GPIO_pad<13> | C-8 | INOUT | General purpose I/O pad n13 |
| GPIO_pad<12> | A-9 | INOUT | General purpose I/O pad n12 |
| GPIO_pad<11> | B-9 | INOUT | General purpose I/O pad n11 |
| GPIO_pad<10> | C-9 | INOUT | General purpose I/O pad n10 |
| GPIO_pad<9> | A-10 | INOUT | General purpose I/O pad n9 |
| GPIO_pad<8> | B-10 | INOUT | General purpose I/O pad n8 |
| GPIO_pad<7> | C-10 | INOUT | General purpose I/O pad n7 |
| GPIO_pad<6> | A-11 | INOUT | General purpose I/O pad n6 |
| GPIO_pad<5> | B-11 | INOUT | General purpose I/O pad n5 |
| GPIO_pad<4> | C-11 | INOUT | General purpose I/O pad n4 |
| GPIO_pad<3> | A-12 | INOUT | General purpose I/O pad n3 |
| GPIO_pad<2> | B-12 | INOUT | General purpose I/O pad n2 |
| GPIO_pad<1> | A-13 | INOUT | General purpose I/O pad n1 |
| GPIO_pad<0> | B-13 | INOUT | General purpose I/O pad n0 |

## 13.6. SCA PINS RECOMMENDATIONS

The SCA pins do not feature any internal pull-down resistors. For that reason, no pin shall be left unconnected, it has to be tied either to VDD or to GND, depending of the users purposes. In addition, the pins that are not going to be used shall be connected to GND directly.

Examples:

- AuxPortTestEn: to GND if the SCA communication is serial or to VDD if is I2C communication (testing purposes).
- If AuxPorTestEn is tied to GND, we have to tie both auxPortSDA and auxPortSCL to GND directly as they are not going to be use.