SER25316HR/SER35316HR

PCI Express Quad Synchronous/Asynchronous Serial Port

User's Manual

BDM-610020136 Rev B

RTD Embedded Technologies, Inc.
AS9100 and ISO 9001 Certified

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803 USA
Telephone: 814-234-8087
Fax: 814-234-5218

www.rtd.com

sales@rtd.com
techsupport@rtd.com

Revision History

Rev A	Initial Release
Rev B	Added AUXA_OUT_SEL, AUXB_OUT_SEL, and AUX_IN_STAT registers. Added TX option to RX_CLOCK_MODE register and RX option to TX_CLOCK_MODE register. Added FIFO_DATA_FORMAT[Reset Offset on Terminate]. Added FB_DMAm_STATn[Terminated]

Advanced Analog I/O, Advanced Digital I/O, aAIO, aDIO, a2DIO, Autonomous SmartCal, "Catch the Express", cpuModule, dspFramework, dspModule, expressMate, ExpressPlatform, "MIL Value for COTS prices", multiPort, PlatformBus, and PC/104EZ are trademarks, and "Accessing the Analog World", dataModule, IDAN, HiDAN, HiDANplus, RTD, and the RTD logo are registered trademarks of RTD Embedded Technologies, Inc. (formerly Real Time Devices, Inc.). PS/2 is a trademark of International Business Machines Inc. PCI, PCI Express, and PCIe are trademarks of PCI-SIG. PC/104, PC/104-Plus, PCI-104, PCIe/104, PCI/104-Express and 104 are trademarks of the PC/104 Consortium. All other trademarks appearing in this document are the property of their respective owners.

Failure to follow the instructions found in this manual may result in damage to the product described in this manual, or other components of the system. The procedure set forth in this manual shall only be performed by persons qualified to service electronic equipment. Contents and specifications within this manual are given without warranty, and are subject to change without notice. RTD Embedded Technologies, Inc. shall not be liable for errors or omissions in this manual, or for any loss, damage, or injury in connection with the use of this manual.

Copyright © 2016 by RTD Embedded Technologies, Inc. All rights reserved.

Table of Contents

1 Introduction **8**

1.1 Product Overview 8

1.2 Board Features 8

1.3 Ordering Information 9

1.4 Contact Information 9
- 1.4.1 Sales Support 9
- 1.4.2 Technical Support 9

2 Specifications **10**

2.1 Operating Conditions 10

2.2 Electrical Characteristics 10

2.1 Functional Characteristics 10

3 Board Connection **11**

3.1 Board Handling Precautions 11

3.2 Physical Characteristics 11

3.3 Connectors and Jumpers 12
- 3.3.1 External I/O Connectors 13
  - CN10, CN11: Synchronous/Asynchronous Serial I/O Connector 13
- 3.3.2 Bus Connectors 14
  - CN1(Top) & CN2(Bottom): PCIe Connector 14
  - CN16: PCI Connector 14
- 3.3.3 Jumpers 14
  - JP500: User ID Jumper 14

3.4 Steps for Installing 15

4 IDAN Connections **16**

4.1 Module Handling Precautions 16

4.2 Physical Characteristics 16

4.3 Connectors 16
- 4.3.1 External I/O Connectors 16
- 4.3.2 Bus Connectors 17
  - CN1(Top) & CN2(Bottom): PCIe Connector 17
  - CN16: PCI Connector 17

4.4 Steps for Installing 18

5 Functional Description **19**

5.1 Universal Synchronous/Asynchronous Receiver/Transmitter (USART) 19
- 5.1.1 Block Diagram 19
- 5.1.2 Overview 19
  - Data Transmission 19
  - Data Reception 20
- 5.1.3 TX FIFO Format 20
  - Header Mode 20
  - Non-Header Mode 21
  - Packed Data 22
  - Non-Packed Data 22
- 5.1.4 RX Data FIFO Format 22
  - Packed Data 22
  - Non-Packed Data 22
- 5.1.5 RX Log FIFO Format 23
- 5.1.6 Character Frame 24
- 5.1.7 Encoding 24
- 5.1.8 SDLC/HDLC Flag 25

Sequential Flag Suppression 25
5.1.9 Idle Character Generation 25
5.1.10 Clocking 25
Clock Sources 25
Transmit Delay 25
Receive Delay 26
Clock Statistics 27
5.1.11 CRC Generation 28

6 Register Address Space 29
Register Types 29
Function Block Mapping 29

6.1 BAR0: General Board Control 30
6.1.1 GBC_FMT (Read-Only) 30
6.1.2 GBC_REV (Read-Only) 30
6.1.3 GBC_BRD_RST (Read/Write) 30
6.1.4 GBC_PDP (Read-Only) 30
6.1.5 GBC_BUILD (Read-Only) 30
6.1.6 GBC_SYS_CLK_FREQ (Read Only) 30
6.1.7 GBC_USER_ID (Read Only) 31
6.1.8 GBC_IRQ_STATUS (Read/Clear) 31
6.1.9 GBC_DIRQ_STATUS (Read/Clear) 31
6.1.10 GBC_EOI (Read/Clear) 31
6.1.11 FBn_ID (Read-Only) 31
6.1.12 FBn_Offset (Read-Only) 31
6.1.13 FBn_Offset_DMA (Read-Only) 31

6.2 BAR2: Functional Block Standard DMA 32
6.2.1 FB_DMAm_Action (Read/Write) 32
6.2.2 FB_DMAm_Last_Action (Read/Write) 32
6.2.3 FB_DMAm_Setup (Read/Write) 32
6.2.4 FB_DMAm_Stat_Used (Read/Write) 33
6.2.5 FB_DMAm_Stat_Invalid (Read/Write) 33
6.2.6 FB_DMAm_Stat_Overflow (Read/Write) 33
6.2.7 FB_DMAm_Stat_Underflow (Read/Write) 33
6.2.8 FB_DMAm_Stat_Complete (Read/Write) 33
6.2.9 FB_DMAm_Current_Buffer (Read-Only) 33
6.2.10 FB_DMAm_COUNT (Read-Only) 33
6.2.11 FB_DMAm_RD_FIFO_CNT(Read-Only) 33
6.2.12 FB_DMAm_WR_FIFO_CNT(Read-Only) 33
6.2.13 FB_DMAm_ADDRESSn (Read/Write) 33
6.2.14 FB_DMAm_SIZEn (Read/Write) 33
6.2.15 FB_DMAm_CTRLn (Read/Write) 34
6.2.16 FB_DMAm_STATn (Read/Clear) 34

6.3 BAR2: Functional Block Header Registers 34
6.3.1 FB_ID (Read-Only) 34
6.3.2 FB_DMA_CHANNELS (Read -Only) 34
6.3.3 FB_DMA_BUFFERS (Read-Only) 34

6.4 BAR2: Universal Synchronous/Asynchronous Receiver/Transmitter (USART) 35
6.4.1 Function Block Register Map 35
6.4.2 MODE (Read/Write) 36
6.4.3 XCVR_MODE (Read/Write) 36
6.4.4 FRAME_FORMAT (Read/Write) 36
6.4.5 ENCODING (Read/Write) 37
6.4.6 FIFO_DATA_FORMAT (Read/Write) 37
6.4.7 DEFAULT_DATA_BITS (Read/Write) 38
6.4.8 IDLE_BITS (Read/Write) 38
6.4.9 IDLE_CHARACTER (Read/Write) 38
6.4.10 ASYNC_CON_LINES (Read/Write) 38
6.4.11 ASYNC_STAT_LINES (Read Only) 39
6.4.12 BIT_PERIOD (Read/Write) 39
6.4.13 CLKA_OUT_SEL (Read/Write) 39
6.4.14 AUXA_OUT_SEL (Read/Write) 40
6.4.15 AUXB_OUT_SEL (Read/Write) 40
6.4.16 AUX_IN_STAT (Read Only) 40

6.4.17 TX_CLOCK_MODE (Read/Write) 40
6.4.18 TX_CLK_DELAY1X (Read/Write) 41
6.4.19 TX_CLK_DELAY2X (Read/Write) 41
6.4.20 TX_SEL_CLK_PERIOD (Read Only) 41
6.4.21 RX_CLOCK_MODE (Read/Write) 41
6.4.22 RX_CLK_DELAY1X (Read/Write) 41
6.4.23 RX_CLK_DELAY2X (Read/Write) 42
6.4.24 RX_SEL_CLK_PERIOD (Read Only) 42
6.4.25 RX_FIRST_EDGE_MIN (Read/Clear) 42
6.4.26 RX_LAST_EDGE_MAX (Read/Clear) 42
6.4.27 RX_SETUP_MIN (Read/Clear) 42
6.4.28 RX_HOLD_MIN (Read/Clear) 42
6.4.29 CRC_POLYNOMIAL (Read/Write) 42
6.4.30 CRC_PRESET_VALUE (Read/Write) 42
6.4.31 CRC_TEST_VALUE (Read/Write) 43
6.4.32 CRC_SIZE (Read/Write) 43
6.4.33 CRC_INVERT_OUTPUT (Read/Write) 43
6.4.34 TX_INT_ENA (Maskable Read/Write) 43
6.4.35 TX_INT_STAT (Read/Clear) 43
6.4.36 TX_BIT_TIME (Read Only) 43
6.4.37 TX_BURST_CLK_PERIOD (Read/Write) 43
6.4.38 TX_BURST_CLK_LENGTH (Read/Write) 43
6.4.39 RX_INT_ENA (Maskable Read/Write) 43
6.4.40 RX_INT_STAT (Read/Clear) 44
6.4.41 RX_BIT_TIME (Read Only) 44
6.4.42 RX_FRAME_SYNC (Read/Write) 44
6.4.43 RX_PACKET_BYTE_CNT_MAX (Read/Write) 44
6.4.44 RX_PACKET_BYTE_CNT_LIMIT (Read/Write) 45
6.4.45 RX_ASYNC_IDLE_TIME_MAX (Read/Write) 45
6.4.46 RX_PACKET_CNT_MAX (Read/Write) 45
6.4.47 RX_BURST_CLK_PERIOD (Read/Write) 45
6.4.48 RX_BURST_CLK_LENGTH (Read/Write) 45
6.4.49 RX_STATUS (Read Only) 45
6.4.50 TX_FIFO_ACCESS (Read/Write) 45
6.4.51 RX_DATA_FIFO_ACCESS (Read/Write) 45
6.4.52 RX_LOG_FIFO_ACCESS (Read/Write) 45

7 Troubleshooting 46

8 Additional Information 47

8.1 PC/104 Specifications 47

8.2 PCI and PCI Express Specification 47

9 Limited Warranty 48

Table of Figures

Figure 1: Board Dimensions 11
Figure 2: Board Connections 12
Figure 3: Example 104™Stack 15
Figure 4: IDAN Dimensions 16
Figure 5: Example IDAN System 18
Figure 6: USART Block Diagram 19
Figure 7: Character Frame 24
Figure 8: Encoding Methods 24
Figure 9: Transmit Clock Delay 26
Figure 10: Receive Clock Delay 27
Figure 11: Receive Statistics 28

Table of Tables

Table 1: Ordering Options 9
Table 2: Operating Conditions 10
Table 3: Electrical Characteristics 10
Table 4: Functional Characteristics 10
Table 5: CN10, CN11: Synchronous/Asynchronous Serial I/O Connector 13
Table 6: JP8 – User ID Jumper 14
Table 7: CN10, CN11: Synchronous/Asynchronous Serial I/O Connector 17
Table 8: TX Data Header 20
Table 9: RX Log 23
Table 10: TX_CLK_DELAY Typical Settings 25
Table 11: RX_CLK_DELAY Typical Settings 26
Table 12: Function Block Mapping 29
Table 13: Base Functional Block 30
Table 14: DMA Registers 32
Table 15: Functional Block Header 34
Table 16: USART Functional Block 35
Table 17: Typical Idle Characters 38
Table 18: Typical CRC Settings 42
Table 19: CRC_SIZE 43

1 Introduction

1.1 Product Overview

The SER25316HR/SER35316HR Synchronous/Asynchronous Serial Port module is a flexible solution to serial communication requirements. It provides four ports that are compatible with SDLC, HDLC, Asynchronous, and many other communications protocols. Software selectable signaling levels allows integration with RS-232, RS-422, and RS-485 systems. Direct Memory Access (DMA) to the host provide for efficient data transport, and allow operating at up to 9.6 Mbps with minimal overhead. The SER25316HR/SER35316HR is a Universal PCIe/104 board that uses a PCIe x1 link.

1.2 Board Features

- PCI Express Bus:
  - PCIe/104 Universal Board (Type 1 or Type 2)
  - PCIe x1 interface (Gen 2, 5 GT/s)
  - PCIe link repopulation allowing increased PCI Express stack capacity
  - In-band interrupts and messages
  - Message Signaled Interrupt (MSI) support
- Four Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Serial Ports
  - General Features
    - Software configurable RS-232, RS-422, and RS-485
    - Maximum bit rate
      - 150 kbps in RS-232 mode
      - 9.6 Mbps in RS-422 and RS-485 mode
    - Supports SDLC, HDLC, Raw Data, and Asynchronous
    - Encoding with NRZ, NRZI, Bi-Phase Mark, Bi-Phase Space, and Manchester
      - Optional data inversion
    - Configurable CRC
      - Supports any 8, 12, 16, or 32-bit polynomial
      - Compatible with SDLC and HDLC
    - Fully configurable Data Frame
      - Data width of 1 to 16 bits
      - Start Bit, Stop Bit, Parity Bit Enable/Disable
    - Full DMA data paths
    - Compatible with non-continuous (bursted) clocks
  - Transmit Features
    - Optional automatic insertion of SDLC/HDLC Flag and CRC
    - Optional transmit Delay
    - Error insertion by using Raw Data
    - Configurable Idle Character
      - 1 to 32 bits
  - Receive Features
    - Optional filtering of repeated SDLC/HDLC Flag characters
    - Receive Log captures statistic of received packet
      - Time of arrival
      - Error
      - Location in receive buffer
      - Packet length
  - Clocking features
    - Transmit and Receive paths are clocked independently
    - Internal (Baud Rate Generator) or External clock
    - Can recover clock from data
    - Clock statistics
      - Measured frequency
      - Setup and Hold Time

1.3 Ordering Information

The SER25316HR/SER35316HR is available with the following options:

Table 1: Ordering Options

Part Number	Description
SER25316HR	PCI/104-Express Synchronous/Asynchronous Serial Port Module
SER35316HR	PCIe/104 Synchronous/Asynchronous Serial Port Module
IDAN- SER25316HR	PCI/104-Express Synchronous/Asynchronous Serial Port Module in IDAN enclosure
IDAN- SER35316HR	PCIe/104 Synchronous/Asynchronous Serial Port Module in IDAN enclosure

The Intelligent Data Acquisition Node (IDAN®) building block can be used in just about any combination with other IDAN building blocks to create a simple but rugged 104™ stack. This module can also be incorporated in a custom-built RTD HiDAN® or HiDANplus® High Reliability Intelligent Data Acquisition Node. Contact RTD sales for more information on our high reliability systems.

1.4 Contact Information

1.4.1 Sales Support

For sales inquiries, you can contact RTD Embedded Technologies sales via the following methods:

Phone: 1-814-234-8087 Monday through Friday, 8:00am to 5:00pm (EST).
E-Mail: sales@rtd.com

1.4.2 Technical Support

If you are having problems with your system, please try the steps in the Troubleshooting section of this manual on page 46.

For help with this product, or any other product made by RTD, you can contact RTD Embedded Technologies technical support via the following methods:

Phone: 1-814-234-8087 Monday through Friday, 8:00am to 5:00pm (EST).
E-Mail: techsupport@rtd.com

2 Specifications

2.1 Operating Conditions

Table 2: Operating Conditions

Symbol	Parameter	Test Condition	Min	Max	Unit
V_{cc5}	5V Supply Voltage		4.75	5.25	V
V_{cc3}	3.3V Supply Voltage		n/a	n/a	V
V_{cc12}	12V Supply Voltage		n/a	n/a	V
V_{cc-12}	-12V Supply Voltage		n/a	n/a	V
T_a	Operating Temperature		-40	+85	C
T_s	Storage Temperature		-55	+125	C
RH	Relative Humidity	Non-Condensing	0	90%	%
MTBF	Mean Time Before Failure	Telcordia Issue 2 30°C, Ground benign, controlled		963,667	Hours

2.2 Electrical Characteristics

Table 3: Electrical Characteristics

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
P	Power Consumption	V_{cc5} = 5.0V		3.0		W
I_{cc}	5V Input Supply Current	Active		0.60		A
	PCIe/104 Bus					
	Differential Output Voltage		0.8		1.2	V
	DC Differential TX Impedance		80		120	Ω
	Differential Input Voltage		0.175		1.2	V
	DC Differential RX Impedance		80		120	Ω
	Electrical Idle Detect Threshold		65		175	mV
	Serial Transceivers					
V_{IH}	Input High Voltage	RS-232 mode	2.0		+15	V
V_{IL}	Input Low Voltage	RS-232 mode	-15		0.6	V
V_{OH}	Output High Voltage	RS-232 mode	5.0	5.5	7.0	V
V_{OL}	Output Low Voltage	RS-232 mode	-7.0	-5.5	-5.0	V
V_{TH}	Input Differential Threshold	RS-422/485 mode	-200	-125	-50	mV
R_{IN}	Input Resistance	RS-422/485 mode TERM = 0 -7V ≤ V_{IN} ≤ 12V	96			kΩ
R_{TERM}	Termination Resistance	RS-422/485 mode TERM = 1 -7V ≤ V_{IN} ≤ 12V	100	120	155	Ω
V_{OD}	Output Differential Voltage	RS-422/485 mode R_L=100Ω (RS-422) R_L=54Ω (RS-485)	 2.0 1.5			 V V
V_{CM}	Driver Common Mode Output Voltage	RS-422/485 mode			3.0	V

2.1 Functional Characteristics

Table 4: Functional Characteristics

Symbol	Parameter	Value	Unit
GBC_SYS_CLK_FREQ	System Clock Frequency	57.6	MHz
	USART TX FIFO Size	4095	D-Words
	USART RX FIFO Size	4095	D-Words
	USART RX Log FIFO Size	511	D-Words

3 Board Connection

3.1 Board Handling Precautions

To prevent damage due to Electrostatic Discharge (ESD), keep your board in its antistatic bag until you are ready to install it into your system. When removing it from the bag, hold the board at the edges, and do not touch the components or connectors. Handle the board in an antistatic environment, and use a grounded workbench for testing and handling of your hardware.

3.2 Physical Characteristics

STEP model is available upon request; contact RTD Tech Support for more information.

- Weight: Approximately 63.5 g (0.14 lbs.)
- Dimensions: 90.17 mm L x 95.89 mm W (3.550 in L x 3.775 in W)

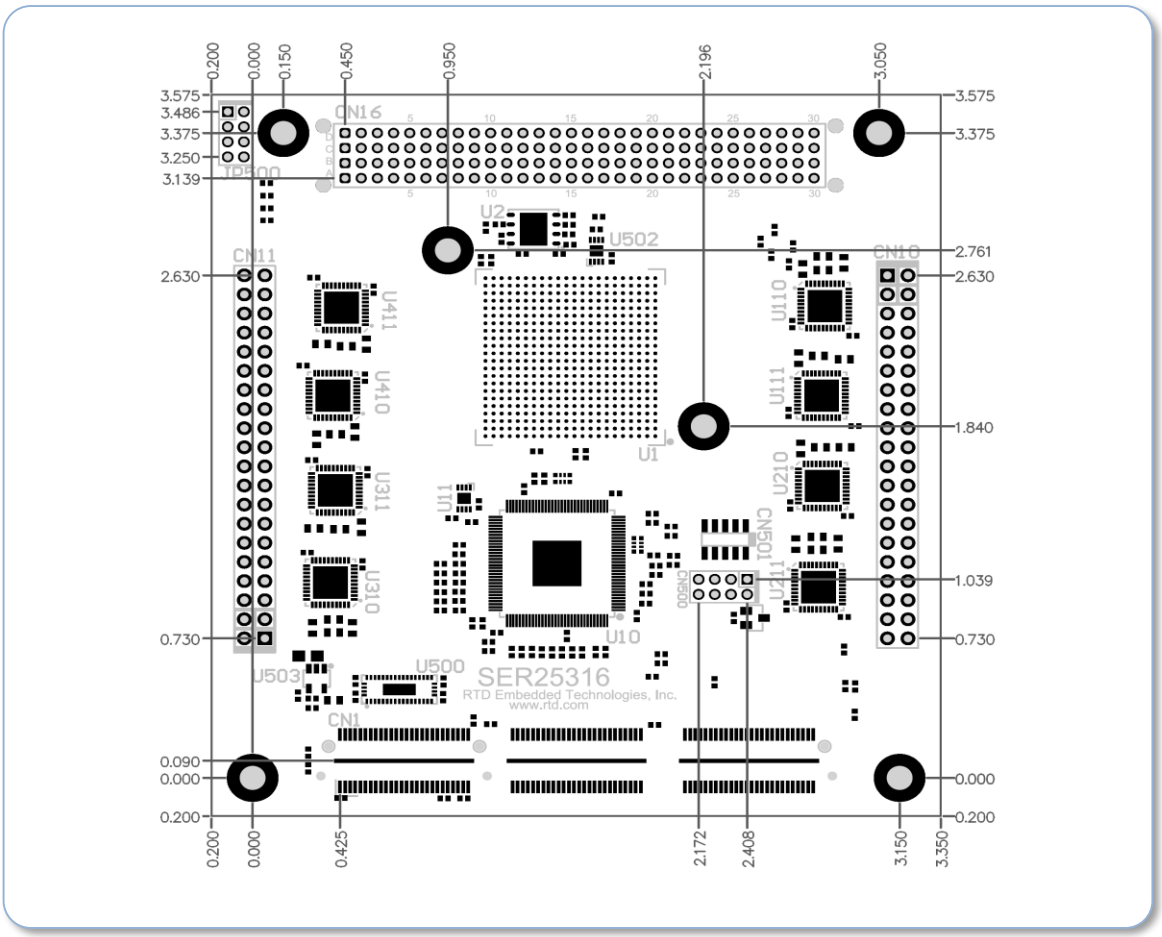


Figure 1: Board Dimensions

3.3 Connectors and Jumpers

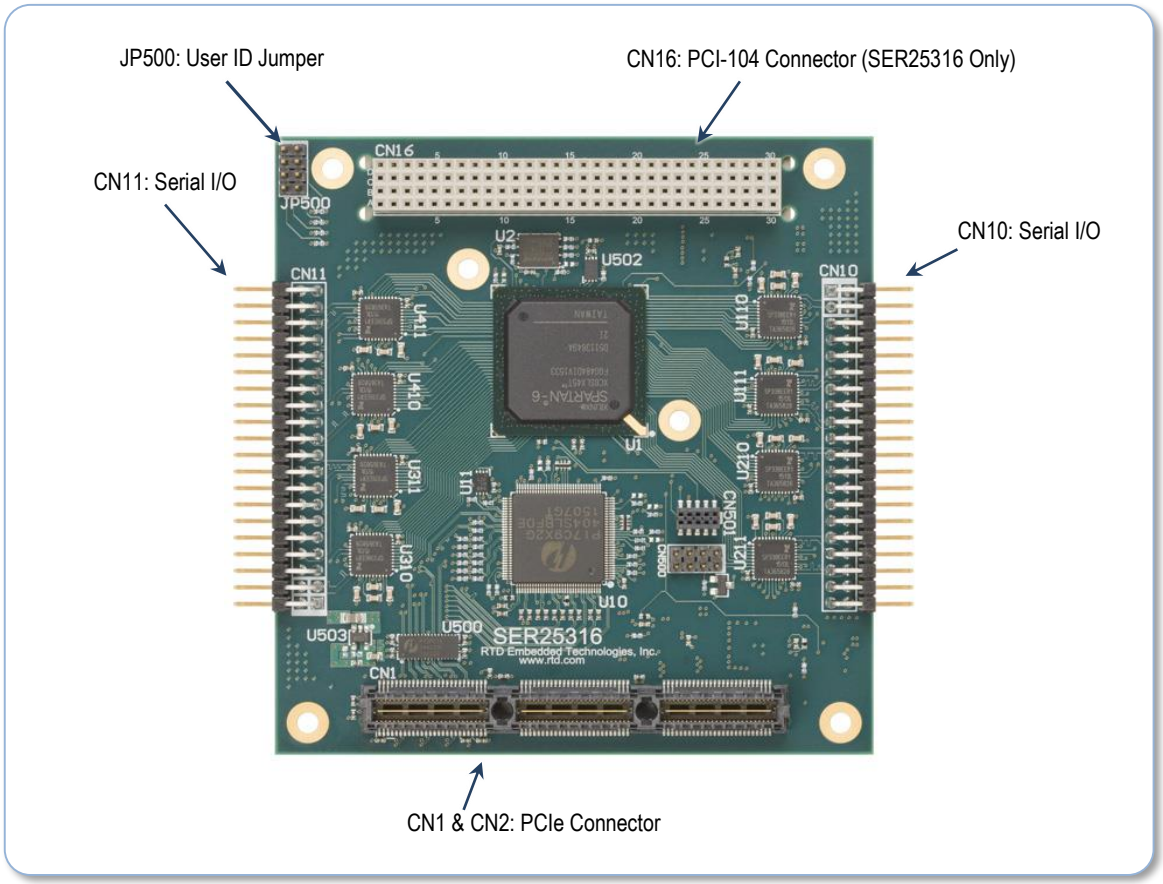


Figure 2: Board Connections

3.3.1 External I/O Connectors

CN10, CN11: Synchronous/Asynchronous Serial I/O Connector

There are two serial port connectors, each of which provides two ports. The Function Block to Port mapping can be found in Table 12 on page 29. Each of the serial ports can be independently configured as RS-232, RS-422, or RS-485. The connector pin assignments for each mode is shown below.

Table 5: CN10, CN11: Synchronous/Asynchronous Serial I/O Connector

CN Pin	Channel	RS-422 Dir	RS-422 Sync Name	RS-422 Sync	RS-422 Async	RS-485 Async	RS-232 Dir	RS-232 Async	RS-232 Sync
1	A	Out	Transmit Data	TXD+	TXD+	D+	Out	TXD	TXD
2	A	Out		TXD-	TXD-	D-	In	DCD	
3	A	In/Out	Aux A	AuxA+			In	CTS	AuxA_In
4	A	In/Out		AuxA-			In	RI	
5	A			GND	GND	GND		GND	GND
6	A			GND	GND	GND		GND	GND
7	A	In/Out	Clock A (normally TX)	ClkA(TX)+			Out		ClkA_Out
8	A	In/Out		ClkA(TX)-			In		ClkB_In
9	A			GND	GND	GND		GND	GND
10	A			GND	GND	GND		GND	GND
11	A	In	Clock B (normally RX)	ClkB(RX)+			Out	DTR	RXS_Out
12	A	In		ClkB(RX)-			Out	RTS	AuxA_Out
13	A			GND	GND	GND		GND	GND
14	A			GND	GND	GND		GND	GND
15	A	In/Out	Aux B	AuxB+			In	DSR	AuxB_In
16	A	In/Out		AuxB-			In		
17	A	In	Receive Data	RXD+	RXD+		In		ClkA_In
18	A	In		RXD-	RXD-		In	RXD	RXD
19	B	Out	Transmit Data	TXD+	TXD+	D+	Out	TXD	TXD
20	B	Out		TXD-	TXD-	D-	In	DCD	
21	B	In/Out	Aux A	AuxA+			In	CTS	AuxA_In
22	B	In/Out		AuxA-			In	RI	
23	B			GND	GND	GND		GND	GND
24	B			GND	GND	GND		GND	GND
25	B	In/Out	Clock A (normally TX)	ClkA(TX)+			Out		ClkA_Out
26	B	In/Out		ClkA(TX)-			In		ClkB_In
27	B			GND	GND	GND		GND	GND
28	B			GND	GND	GND		GND	GND
29	B	In	Clock B (normally RX)	ClkB(RX)+			Out	DTR	RXS_Out
30	B	In		ClkB(RX)-			Out	RTS	AuxA_Out
31	B			GND	GND	GND		GND	GND
32	B			GND	GND	GND		GND	GND
33	B	In/Out	Aux B	AuxB+			In	DSR	AuxB_In
34	B	In/Out		AuxB-			In		
35	B	In	Receive Data	RXD+	RXD+		In		ClkA_In
36	B	In		RXD-	RXD-		In	RXD	RXD
37	B			GND	GND	GND		GND	GND
38				GND	GND	GND		GND	GND
39				GND	GND	GND		GND	GND
40				GND	GND	GND		GND	GND

3.3.2 BUS CONNECTORS

CN1(Top) & CN2(Bottom): PCIe Connector

The PCIe connector is the connection to the system CPU. The position and pin assignments are compliant with the *PCI/104-Express Specification*. (See PC/104 Specifications on page 47)

This is a "Universal" board, and can connect to either a Type 1 or Type 2 PCIe/104 connector.

CN16: PCI Connector

The PCI connector is the connection to PCI peripheral modules. This connector is used only as a pass through connector on this board.

3.3.3 JUMPERS

On all jumpers, pin 1 is designated by a thick white silkscreen line, and a square pad on the PCB.

JP500: User ID Jumper

The User ID Jumper is a four position, user defined jumper block. The jumpers can be read by the FPGA. An installed jumper results in a logic low, and an open jumper results in a logic high.

Table 6: JP8 – User ID Jumper

Position	Description
1-2	User ID bit 0
3-4	User ID bit 1
5-6	User ID bit 2
7-8	User ID bit 3

3.4 Steps for Installing

1. Always work at an ESD protected workstation, and wear a grounded wrist-strap.
2. Turn off power to the PC/104 system or stack.
3. Select and install stand-offs to properly position the module on the stack.
4. Remove the module from its anti-static bag.
5. Check that pins of the bus connector are properly positioned.
6. Check the stacking order; make sure all of the busses used by the peripheral cards are connected to the cpuModule.
7. Hold the module by its edges and orient it so the bus connector pins line up with the matching connector on the stack.
8. Gently and evenly press the module onto the PC/104 stack.
9. If any boards are to be stacked above this module, install them.
10. Attach any necessary cables to the PC/104 stack.
11. Re-connect the power cord and apply power to the stack.
12. Boot the system and verify that all of the hardware is working properly.

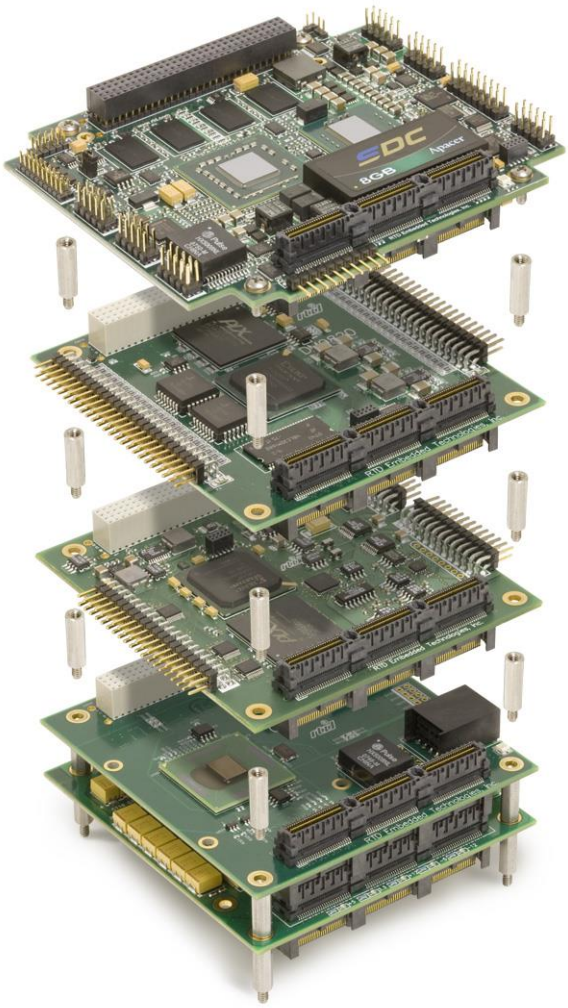


Figure 3: Example 104™ Stack

4 IDAN Connections

4.1 Module Handling Precautions

To prevent damage due to Electrostatic Discharge (ESD), keep your module in its antistatic bag until you are ready to install it into your system. When removing it from the bag, hold the module by the aluminum enclosure, and do not touch the components or connectors. Handle the module in an antistatic environment, and use a grounded workbench for testing and handling of your hardware.

4.2 Physical Characteristics

- Weight: Approximately TBD
- Dimensions: 151.97 mm L x 129.98 mm W x 16.99 mm H (5.983 in L x 5.117 in W x 0.669 in H)

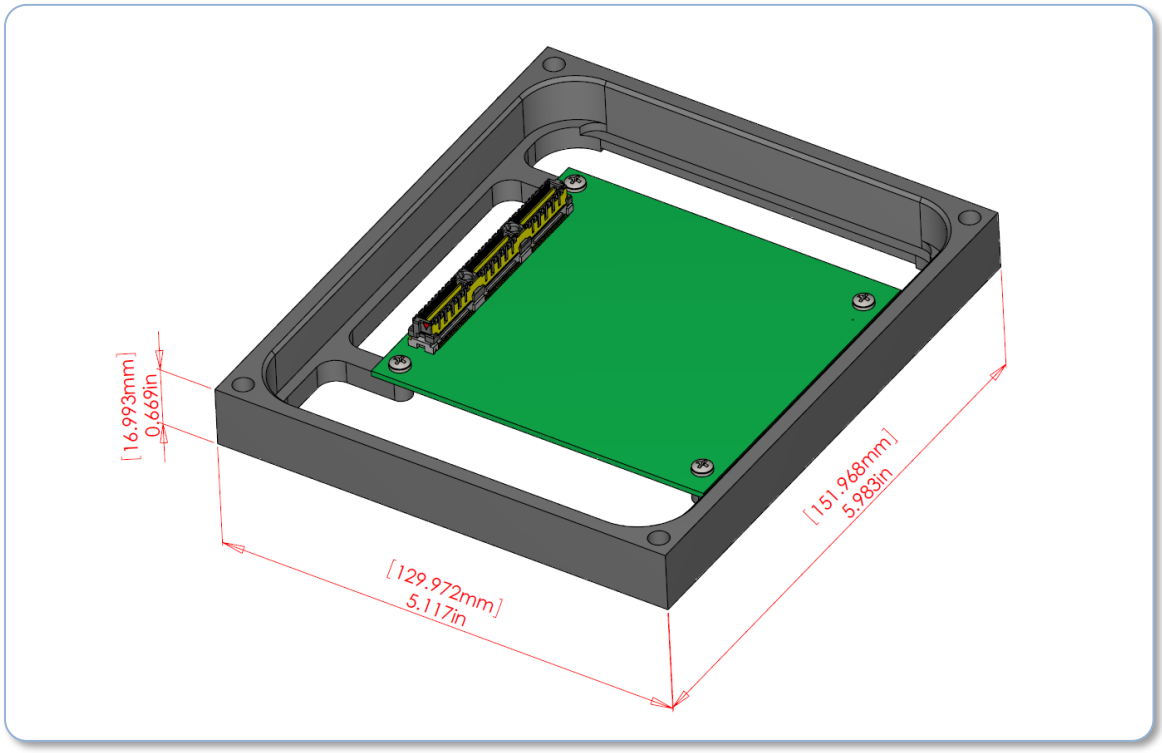


Figure 4: IDAN Dimensions

4.3 Connectors

4.3.1 External I/O Connectors

There are two serial port connectors, each of which provides two ports. The Function Block to Port mapping can be found in Table 12 on page 29. Each of the serial ports can be independently configured as RS-232, RS-422, or RS-485. The connector pin assignments for each mode is shown below.

Table 7: CN10, CN11: Synchronous/Asynchronous Serial I/O Connector

IDAN Pin #	CN Pin	Channel	RS-422 Dir	RS-422 Sync Name	RS-422 Sync	RS-422 Async	RS-485 Async	RS-232 Dir	RS-232 Async	RS-232 Sync
1	1	A	Out	Transmit Data	TXD+	TXD+	D+	Out	TXD	TXD
20	2	A	Out		TXD-	TXD-	D-	In	DCD	
2	3	A	In/Out	Aux A	AuxA+			In	CTS	AuxA_In
21	4	A	In/Out		AuxA-			In	RI	
3	5	A			GND	GND	GND		GND	GND
22	6	A			GND	GND	GND		GND	GND
4	7	A	In/Out	Clock A (normally TX)	ClkA(TX)+			Out		ClkA_Out
23	8	A	In/Out		ClkA(TX)-			In		ClkB_In
5	9	A			GND	GND	GND		GND	GND
24	10	A			GND	GND	GND		GND	GND
6	11	A	In	Clock B (normally RX)	ClkB(RX)+			Out	DTR	RXS_Out
25	12	A	In		ClkB(RX)-			Out	RTS	AuxA_Out
7	13	A			GND	GND	GND		GND	GND
26	14	A			GND	GND	GND		GND	GND
8	15	A	In/Out	Aux B	AuxB+			In	DSR	AuxB_In
27	16	A	In/Out		AuxB-			In		
9	17	A	In	Receive Data	RXD+	RXD+		In		ClkA_In
28	18	A	In		RXD-	RXD-		In	RXD	RXD
10	19	B	Out	Transmit Data	TXD+	TXD+	D+	Out	TXD	TXD
29	20	B	Out		TXD-	TXD-	D-	In	DCD	
11	21	B	In/Out	Aux A	AuxA+			In	CTS	AuxA_In
30	22	B	In/Out		AuxA-			In	RI	
12	23	B			GND	GND	GND		GND	GND
31	24	B			GND	GND	GND		GND	GND
13	25	B	In/Out	Clock A (normally TX)	ClkA(TX)+			Out		ClkA_Out
32	26	B	In/Out		ClkA(TX)-			In		ClkB_In
14	27	B			GND	GND	GND		GND	GND
33	28	B			GND	GND	GND		GND	GND
15	29	B	In	Clock B (normally RX)	ClkB(RX)+			Out	DTR	RXS_Out
34	30	B	In		ClkB(RX)-			Out	RTS	AuxA_Out
16	31	B			GND	GND	GND		GND	GND
35	32	B			GND	GND	GND		GND	GND
17	33	B	In/Out	Aux B	AuxB+			In	DSR	AuxB_In
36	34	B	In/Out		AuxB-			In		
18	35	B	In	Receive Data	RXD+	RXD+		In		ClkA_In
37	36	B	In		RXD-	RXD-		In	RXD	RXD
19	37	B			GND	GND	GND		GND	GND

4.3.2 BUS CONNECTORS

CN1(Top) & CN2(Bottom): PCIe Connector

The PCIe connector is the connection to the system CPU. The position and pin assignments are compliant with the *PCI/104-Express Specification*. (See PC/104 Specifications on page 47)

This is a "Universal" board, and can connect to either a Type 1 or Type 2 PCIe/104 connector.

CN16: PCI Connector

The PCI connector is the connection to PCI peripheral modules. This connector is used only as a pass through connector on this board.

4.4 Steps for Installing

1. Always work at an ESD protected workstation, and wear a grounded wrist-strap.
2. Turn off power to the IDAN system.
3. Remove the module from its anti-static bag.
4. Check that pins of the bus connector are properly positioned.
5. Check the stacking order; make sure all of the busses used by the peripheral cards are connected to the cpuModule.
6. Hold the module by its edges and orient it so the bus connector pins line up with the matching connector on the stack.
7. Gently and evenly press the module onto the IDAN system.
8. If any boards are to be stacked above this module, install them.
9. Finish assembling the IDAN stack by installing screws of an appropriate length.
10. Attach any necessary cables to the IDAN system.
11. Re-connect the power cord and apply power to the stack.
12. Boot the system and verify that all of the hardware is working properly.

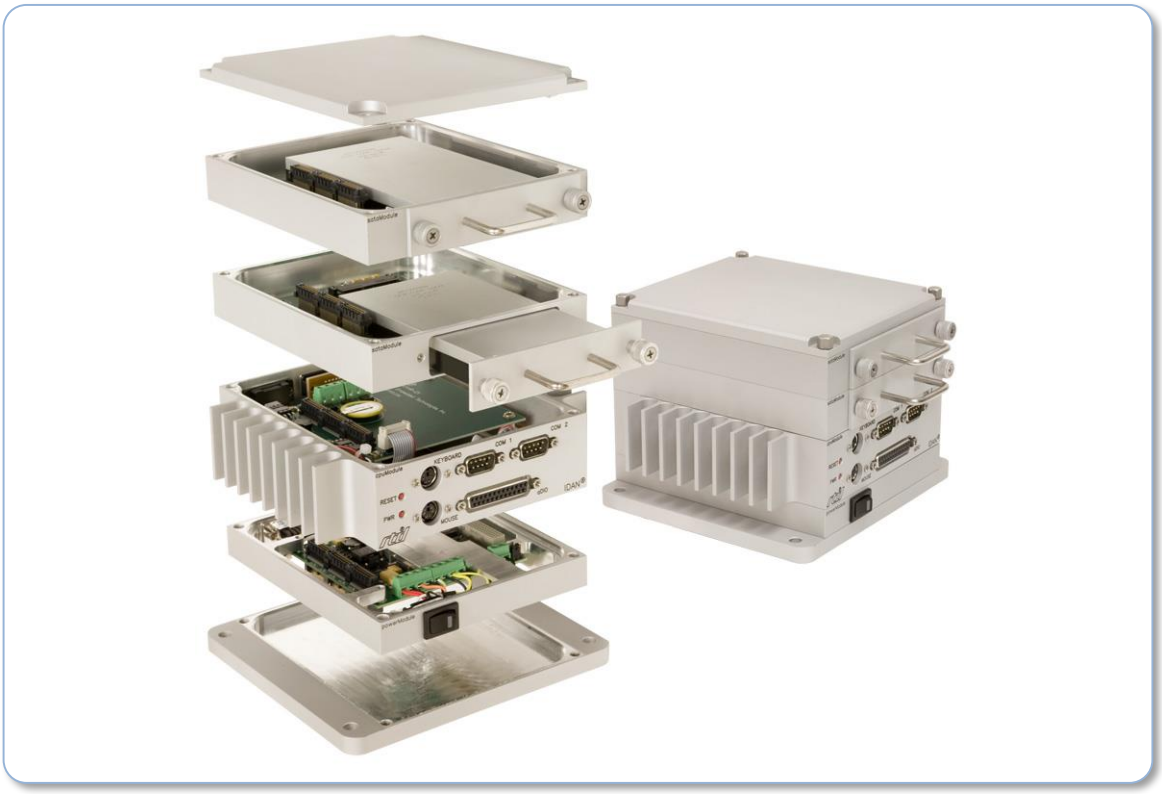


Figure 5: Example IDAN System

5 Functional Description

5.1 Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

5.1.1 Block Diagram

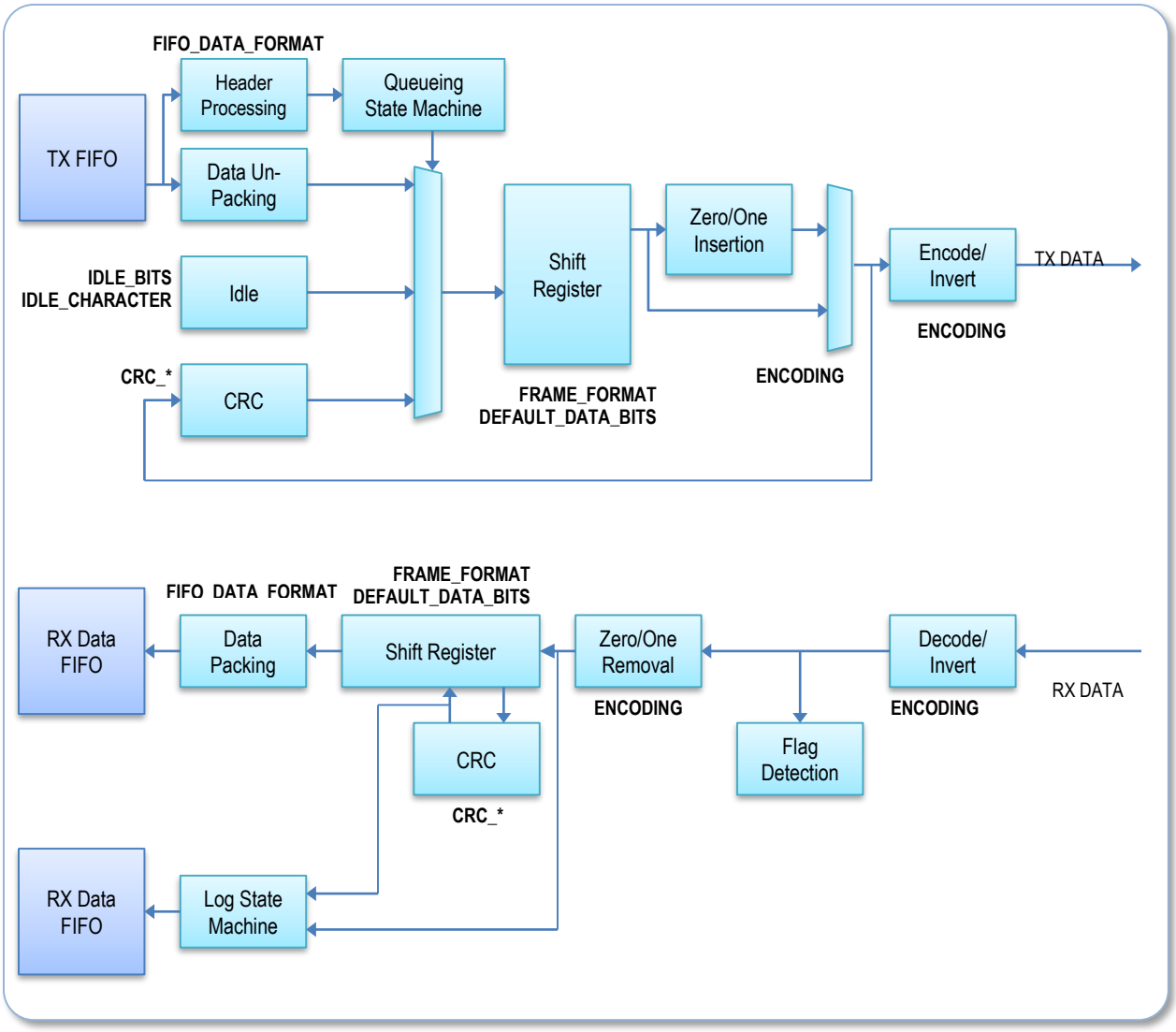


Figure 6: USART Block Diagram

5.1.2 Overview

Data Transmission

Data transmission is initiated by writing data to the TX FIFO. If Header Mode is not used, data transmission will begin as soon as there is data in the TX FIFO.

If Header Mode is used, the hardware has knowledge of the length of the data block (or packet) being sent, and will wait until all of the data to be transmitted is in the TX FIFO before beginning transmission. This prevents the spurious addition of idle characters within the packet. If the packet length is greater than 255 double-words, then packet transmission will start when there are at least 255 double-words in the TX FIFO. Header Mode can also be used to ensure that there are SDLC Flag characters between frames.

The hardware can calculate the CRC of the packet, and append it to the end. See CRC Generation on page 28 for more details.

Whenever there is no longer data in the TX FIFO, the Idle Character is sent. This is defined by the IDLE_BITS and IDLE_CHARACTER registers. The idle character can be a different length than a data character.

Data Reception

Character synchronization can be accomplished several ways. In asynchronous communication, a start-bit is used (FRAME_FORMAT[Start Bit] = 1). In this case, detecting the start bit also provides character synchronization. In HDLC/SDLC communications, a special Flag character is used to provide character synchronization and delineate packet boundaries. This is enabled by setting RX_FRAME_SYNC = Flag.

The hardware can check the CRC at the end of the packet. See CRC Generation on page 28 for more details.

At the end of every packet, an RX Log entry is generated. This is available in a separate data path from the RX Data. The Log provides information such as the length of the packet, the offset of the start of the packet in the data stream, the time that the packet started, and any errors that occurred during the packet.

DMA Buffer Termination

Normally, a DMA buffer is terminated after the number of bytes in FB_DMAm_SIZEn is transferred. However, the end of a packet may occur before the end of the DMA buffer. In that case, the data is "stuck" in the DMA path until it is flushed out with new data.

In the FIFO_DATA_FORMAT register there are bits called "RX Data Packet Terminate" and "RX Log Packet Terminate". When set, the current DMA buffer is terminated for the Data channel and Log channel respectively when any of the following occur:

1. RX_PACKET_CNT_MAX is exceeded. Packets are counted when the SDLC Flag is detected. If RX_PACKET_CNT_MAX = 0, the buffer will be terminated at every packet.

2. RX_PACKET_BYTE_CNT_MAX is exceeded if enabled.

3. RX_ASYNC_IDLE_TIME_MAX is exceeded if enabled.

When the DMA buffer is terminated early, the actual amount that was transferred is written to the FB_DMAm_SIZEn register. Note that this means that after data is moved from the kernel DMA buffer to the user buffer, the FB_DMAm_SIZEn register must be updated with the original size of the buffer. Also, at termination all of the Halt/Loop/Int flags are honored.

5.1.3 TX FIFO FORMAT

Header Mode

When FIFO_DATA_FORMAT [TX Header] is enabled, all data in the transmit FIFO must include a header. The description of the header is shown below. The header must always start on a double-word boundary. If the previous packet did not end on a double-word boundary, the bytes in between are ignored.

Table 8: TX Data Header

	Offset	0x03	0x02	0x01	0x00
Header	0x00	*Reserved*	*Reserved*	CONTROL	ID
	0x04	LENGTH			
	0x08	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	0x0C	START_BIT_TIME			
	0x10	Start of data			

ID

The ID is used to identify the beginning of the header. It must have a value of 0xA5. If the transmit state machine is expecting the beginning of a header, and the ID is not equal to 0xA5, the data is discarded.

CONTROL

The control byte defines how the data in the packet is handled. Bit definitions are below:

- b[0] – **Start Flag:**
  - 0 – **Disabled**: Flag is not prepended.
  - 1 – **Enabled**: Generate an SDLC/HDLC Flag character before the packet.
- b[1] – **End Flag**:
  - 0 – **Disabled**: Flag is not appended.
  - 1 – **Enabled**: Generate an SDLC/HDLC Flag character after the packet.
- b[2] – **CRC:** Calculate CRC on packet.
  - 0 – **Disabled**: CRC is not appended.
  - 1 – **Enabled**: CRC is cleared at the beginning of the packet, and appended at the end.
- b[3] - **Delay Start**:
  - 0 – **Disabled**: Packet is sent immediately.
  - 1 – **Enabled**: Packet is not started until START_BIT_TIME.
- b[4] – **Interrupt:** Generate an interrupt at end of packet.
  - 0 – **Disabled**: No interrupt is generated.
  - 1 – **Enabled**: Interrupt is enabled at the end of the packet.
- b[5] – **Pack Override**: This packet uses the setting from Pack Data below, instead of FIFO_DATA_FORMAT[Pack].
  - 0 – **Disabled**: Data format is set from FIFO_DATA_FORMAT[Pack].
  - 1 – **Enabled**: Data format is set from Pack Data below.
- b[6] - **Pack Data:** Override value for data packing. Only has effect if Pack Override (above) is set.
  - 0 – **Disabled**: Each frame is stored in a 32-bit word along with meta-data.
  - 1 – **Enabled**: Four frames of data only are stored in a 32-bit word.

LENGTH

This sets the number of character frames in this packet. If data packing is enabled, this is in Bytes. If data packing is disabled, this is in Double-Words of FIFO data. This is the length of the data stored in the FIFO only, and does not include any automatically appended CRC or flags.

START_BIT_TIME

If CONTROL[Delay Start] is enabled, the packet will not be transmitted until TX_BIT_TIME is equal to START_BIT_TIME. The range for START_BIT_TIME is from (TX_BIT_TIME) to (TX_BIT_TIME + $2^{31} - 1$). If START_BIT_TIME is outside of that range, the start time is assumed to be in the past and the packet is sent immediately.

Non-Header Mode

When FIFO_DATA_FORMAT [TX Header] is disable, the FIFO only contains data. The data is transmitted as soon as it is available. If any framing characters (e.g. SDLC/HDLC Flag) are required, they must be included in the FIFO. In that case, Packed Data cannot be used.

Packed Data

The transmit data path is 32-bits wide. However, since most serial communications are byte-centric, four bytes of data can be packed into a double-word. The data is stored little-endian. This can be controlled with either FIFO_DATA_FORMAT[Pack] or the CONTROL[Pack Data] bit in the header.

Non-Packed Data

If Data Packing is disabled, there are several control bits that are added:

- b[15:0] - **Data:** Up to 16 bits of data to be sent
- b[19:16] – **Data Bits**: Override value for the number of data bits to send.
- b[20] – **Data Bits Override**
    - 0 – **Disabled**: The number of data bits to send is DEFAULT_DATA_BITS.
    - 1 – **Enabled**: The number of data bits to send is Data Bits (above).
- b[21] – **Framing Bypass**
    - 0 – **Disabled**: Normal.
    - 1 – **Enabled**: Start bit, Stop bit, Parity, Zero-Insertion, and One-Insertion are bypassed.
- b[24] – **CRC First**
    - 0 – **Disabled**: CRC Calculations continue.
    - 1 – **Enabled**: This is the first data element in the CRC calculation.
- b[25] – **CRC Mask**
    - 0 – **Disabled**: This data element is included in CRC calculations.
    - 1 – **Enabled**: This data element is excluded from CRC calculations.
- b[27] – **CRC Insert**
    - 0 – **Disabled**: The Data field is transmitted.
    - 1 – **Enabled**: The CRC is inserted in place of Data.

5.1.4 RX DATA FIFO FORMAT

Packed Data

The receive data path is 32-bits wide. However, since most serial communications are byte-centric, four bytes of data can be packed into a double-word. The data is stored little-endian. This can be controlled with FIFO_DATA_FORMAT[Pack].

Non-Packed Data

If Data Packing is disabled, there are several status bits that are added:

- b[15:0] - **Data:** Up to 16 bits of data received.
- b[24] - **Decoding Error**: Indicates an illegal transition in the FM0, FM1, or Manchester Encoding
- b[25] – **Bit Insertion Error**: Indicates more ones or zeros than what is allowed by the bit insertion rules. This is also asserted when an SLDC Flag is detected.
- b[26] – **Parity Error**: Indicates that a parity error has been detected.
- b[27] – **Stop Error**: Also called Framing Error. Indicates that an insufficient number of Stop bits were received.

- b[28] – **Sync Before End of Data**: Indicates that an SDLC Flag character was detected during the data phase of the frame. This indicates a loss of character sync.

- b[29] – **Sync Before End of Frame**: Indicates that an SDLC Flag character was detected during the frame. Normally the Flag is only detected at the frame boundary. This indicates a loss of character sync.

- b[30] – **Flag Detected**: Indicates that an SDLC Flag character has been detected. When searching for a Flag in the data, only this bit should be checked. The value in the Data field is undefined.

- b[31] – **CRC Ok**: Indicates that a valid CRC has been detected. Note that the CRC is continuously checked, so this does not necessarily indicate that the valid CRC was at the end of the packet.

5.1.5 RX LOG FIFO FORMAT

At the conclusion of every received packet, an entry is stored in the RX Log FIFO. This Log entry can be used to identify where in the RX Data FIFO the packet is stored, error and CRC status, and the bit time at the start of the packet. Log entries are shown below.

Table 9: RX Log

Offset	0x03	0x02	0x01	0x00
0x00	*Reserved*	*Reserved*	STATUS	
0x04	LENGTH			
0x08	START_OFFSET			
0x0C	START_BIT_TIME			

STATUS

This field indicates the status of the packet. If an error occurred at any point during the packet, it is reflected here. Bit definitions are below:

- b[0] - **Decoding Error**: Indicates an illegal transition in the FM0, FM1, or Manchester Encoding

- b[1] – **Bit Insertion Error**: Indicates more ones or zeros than what is allowed by the bit insertion rules. This is masked when an SLDC Flag is detected.

- b[2] – **Parity Error**: Indicates that a parity error has been detected.

- b[3] – **Stop Error**: Also called Framing Error. Indicates that an insufficient number of Stop bits were received.

- b[4] – **Sync Before End of Data**: Indicates that an SDLC Flag character was detected during the data phase of the frame. This indicates a loss of character sync.

- b[5] – **Sync Before End of Frame**: Indicates that an SDLC Flag character was detected during the frame. Normally the Flag is only detected at the frame boundary. This indicates a loss of character sync.

- b[6] – **Long Packet**: Indicates that the packet was prematurely terminated because RX_PACKET_BYTE_CNT_MAX was exceeded.

- b[7] – **CRC Ok**: Indicates that a valid CRC has been detected as the last data value in the packet.

- b[8] – **Data Pack**: Indicated how data is stored.

    - 0 – **Disabled**: Each character frame is stored in a 32-bit word along with meta-data.

    - 1 – **Enabled**: Four frames of data only are stored in a 32-bit word.

- b[9] – **Async Idle**: Indicates that the packet was terminated because the idle time exceeded RX_ASYNC_IDLE_TIME_MAX.

LENGTH

This reports the number of character frames in this packet. If data packing is enabled, this is in Bytes. If data packing is disabled, this is in Double-Words of FIFO data. For SDLC/HDLC frames, this includes the CRC, but does not include the closing flag.

START_OFFSET

This reports the offset in bytes of the start of the packet in the data stream. It is reset when the RX Data DMA channel is reset, or from buffer termination if FIFO_DATA_FORMAT[Reset Offset on Terminate] is Enabled.

START_BIT_TIME

This reports the value of RX_BIT_TIME at the last bit of the first non-flag or non-idle character in the packet.

5.1.6 CHARACTER FRAME

The Character Frame referrers to the Start bit, Stop bits, Parity bit, and Data bits that are transferred for each data character.

For SDLC/HDLC communications, the Character Frame typically consists of only Data.

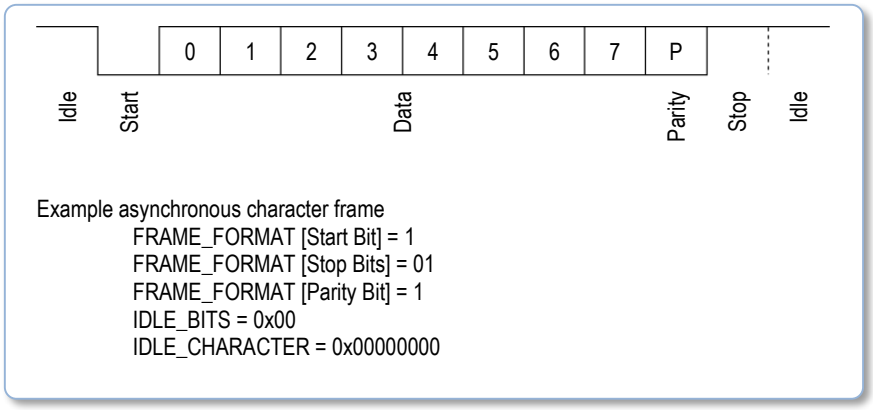


Figure 7: Character Frame

5.1.7 ENCODING

Encoding can be used to embed the clock in the data stream. This allows a simpler channel. The figure below shows the encoding options. Most applications that use a separate clock or asynchronous communications will use NRZ encoding.

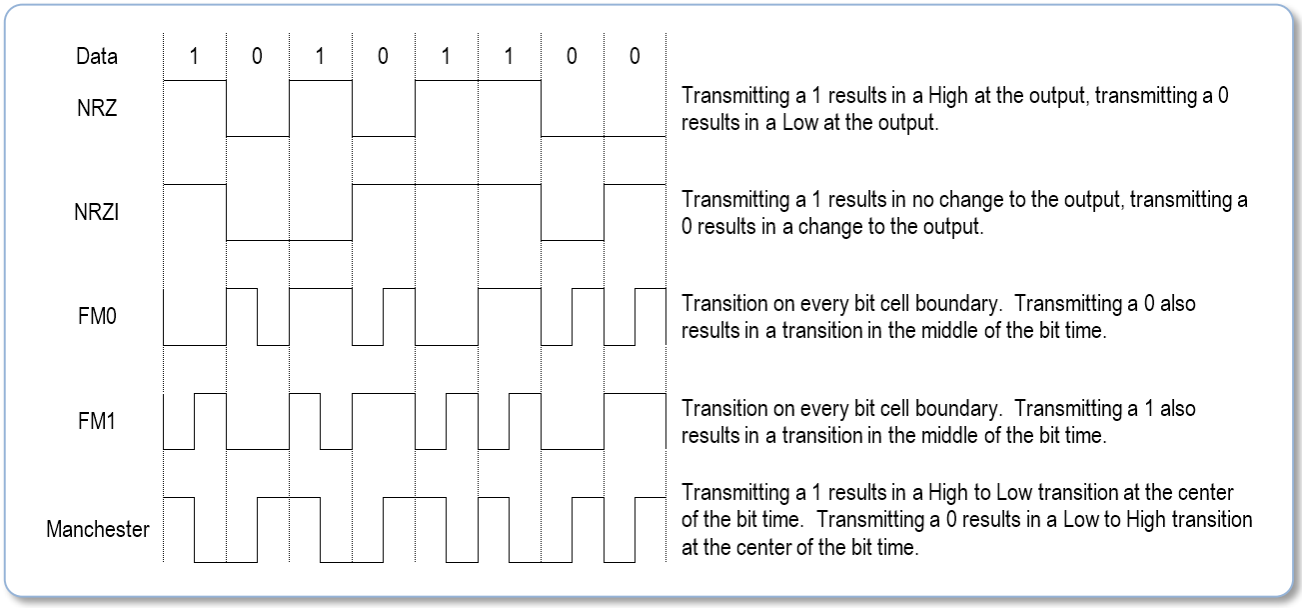


Figure 8: Encoding Methods

5.1.8 SDLC/HDLC FLAG

In SDLC and HDLC communication, the packets are delimited by a Flag character. The Flag has a hex value of 0x7E. Zero insertion is also used in SDLC and HDLC, so a data value 0x7E would have a zero inserted and is therefore differentiated from a Flag. The Flag is also typically used as the idle character.

When transmitting data using Header Mode (see page 20), a Flag can be automatically inserted before and/or after the data packet. When transmitting data without Header Mode, Non-Packed Data must be used (see page 22). The Flag is then represented by a data value of 0x0x0020007E (Framing Bypass set, and a data value of 0x7E).

When receiving data, the Flag character is sent to the RX Data FIFO. However, the data packet as described in the RX Log FIFO does not include the Flag. There will typically be a single Flag character in the RX Data FIFO between every packet.

If a Flag is received when the receiver is not character synchronized, it will still be correctly recognized. In this case, the data in the FIFO may be a value other than 0x7E. If Non-Packed Data is used (see page 22), the "Flag Detected" bit will be set, regardless of the data value. As soon as the Flag is received, the receiver will be character synchronized, as long as RX_FRAME_SYNC is set to "Flag."

The SDLC/HDLC packet may share the starting and ending Flag (i.e. have a single flag between packets). The receiver will also correctly detect flags with a shared zero (011111101111110).

Sequential Flag Suppression

If RX_FRAME_SYNC[Sequential Flag Suppression] is set, repeated flag characters are not sent to the RX FIFO. If there are multiple sequential flags, only the first is stored. This allows packet boundaries to be easily identified without flooding the RX data path with idle flag characters.

5.1.9 IDLE CHARACTER GENERATION

The character used by the transmitter when idle is fully configurable. The character length and value can be set with the IDLE_BITS and IDLE_CHARACTER registers respectively.

5.1.10 CLOCKING

The clocking system provides extensive flexibility. The Transmit and Receive paths have a common bit rate (set with the BIT_PERIOD register). Otherwise the Transmit and Receive paths are clocked independently.

Clock Sources

The clock source for the Transmit and Receive path are chosen with the TX_CLOCK_MODE and RX_CLOCK_MODE register respectively. The rising or falling edge can be selected. The following clock sources are available:

- **BRG:** Baud Rate Generator. The Baud Rate Generator internally generates a bit clock based on the BIT_PERIOD register. In the Receive path, the BRG is synchronized to the transitions of RX Data. It is also suppressed after a Stop Bit is received. This allows the BRG to be used for recovering Asynchronous data.
- **DATA**: The clock is recovered from RX Data. There must be at least one transition on RX Data per bit period, i.e. FM0, FM1, or Manchester encoding must be used.
- **CLKA:** The Clock A Input (CLKA) is used as the clock source. This is typically the transmit clock, and can be an input or an output.
- **CLKB:** The Clock B Input (CLKB) is used as the clock source. This is typically the receive clock, and is an input only.

Transmit Delay

The transmitted data can be delayed relative to the selected clock. The TX_CLK_DELAY1X register selects the delay from the selected transmit clock until the primary data transition. For FM0, FM1, and Manchester encoding there also may be a transition in the middle of the bit time. That can be adjusted with the TX_CLK_DELAY2X register.

Typical settings are shown in the Table below.

Table 10: TX_CLK_DELAY Typical Settings

	TX_CLK_DELAY1X	TX_CLK_DELAY2X
Asynchronous	**0**	**(1/2) * BIT_PERIOD**
NRZ or NRZI Synchronous	**(1/2) * BIT_PERIOD**	**(3/4) * BIT_PERIOD**
FM0, FM1 or Manchester Synchronous with BRG as the clock source	**(1/2) * BIT_PERIOD**	**BIT_PERIOD – 1**
FM0, FM1 or Manchester Synchronous with external clock source	**(1/4) * BIT_PERIOD**	**(3/4) * BIT_PERIOD**

TX_CLK_DELAY1X and TX_CLK_DELAY2X must be set such that they expire prior to the next clock. Any clock that is detected prior to both delays expiring is ignored.

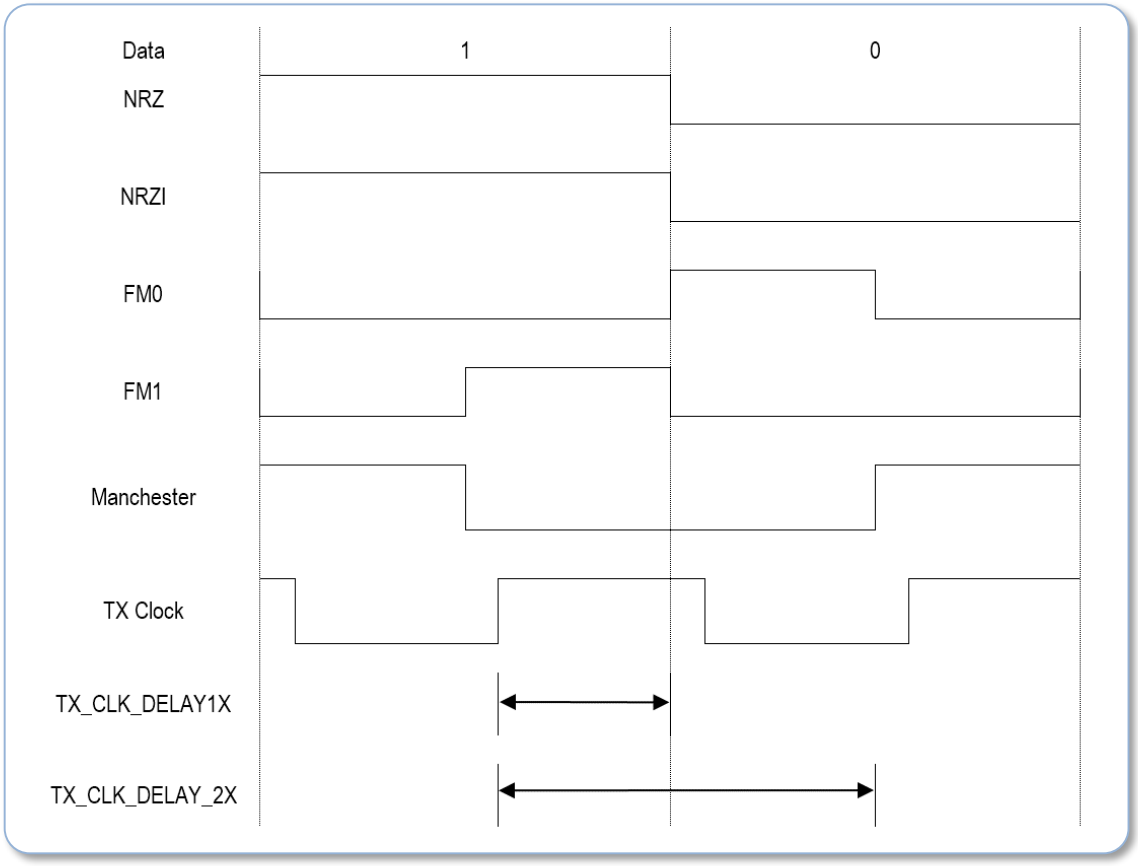


Figure 9: Transmit Clock Delay

Receive Delay

The sample point for the received data can be delayed relative to the selected clock. The RX_CLK_DELAY1X register selects the delay from the selected received clock until the primary data sample point. For FM0, FM1, and Manchester encoding there also may be a sample point in the middle of the bit time. That can be adjusted with the RX_CLK_DELAY2X register.

Typical settings are shown in the Table below.

Table 11: RX_CLK_DELAY Typical Settings

	TX_CLK_DELAY1X	TX_CLK_DELAY2X
Asynchronous	**(1/2) * BIT_PERIOD**	**(3/4) * BIT_PERIOD**
NRZ or NRZI Synchronous	**0**	**(3/4) * BIT_PERIOD**
FM0, FM1 or Manchester Synchronous	**(1/4) * BIT_PERIOD**	**(3/4) * BIT_PERIOD**

RX_CLK_DELAY1X and RX_CLK_DELAY2X must be set such that they expire prior to the next clock. Any clock that is detected prior to both delays expiring is ignored.

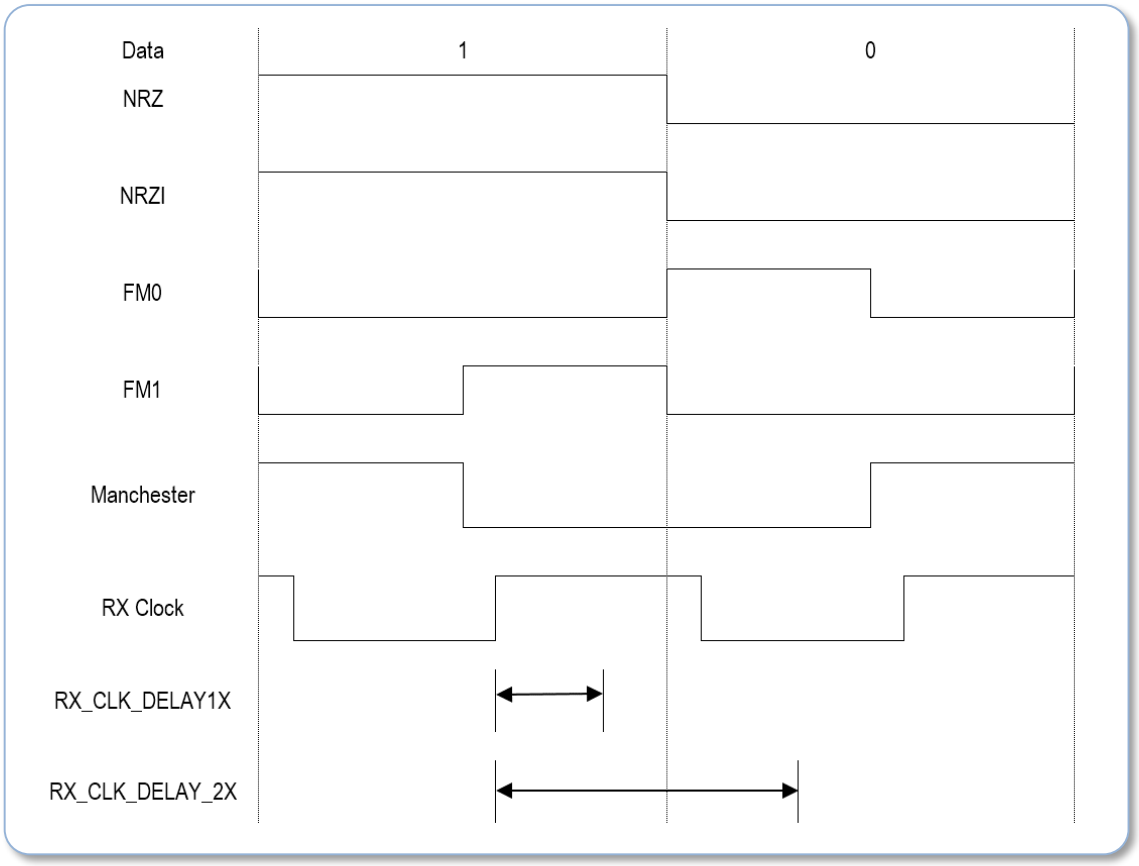


Figure 10: Receive Clock Delay

Clock Statistics

There are several statistics registers that can be used to collect information about the clock and the clock-data relationship during configuration. Once a clock source is selected, the measured period is available in TX_SEL_CLK_PERIOD and RX_SEL_CLK_PERIOD. This can be used to set the value for BIT_PERIOD, and calculate the delay values. The period measurement registers are updated every clock cycle.

The receive clock to data relationship can be measured using the RX_FIRST_EDGE_MIN and RX_LAST_EDGE_MAX registers. These are persistent registers that track the time from the selected clock edge to the first and last data transition respectively. They can be used to set the transmit clock delay registers. For synchronous communications, typical values are:

RX_CLK_DELAY1X = MIN (0, (RX_FIRST_EDGE_MIN + RX_LAST_EDGE_MAX – RX_SEL_CLK_PERIOD)/2)

RX_CLK_DELAY2X = (RX_LAST_EDGE_MAX + RX_SEL_CLK_PERIOD) / 2

This will put the primary sample point half-way between the previous data edge and next data edge, and the secondary sample point half-way between the last edge and the end of the clock period.

After the RX_CLK_DELAY* registers are set, the setup and hold time for the Receive Data can be measured with RX_SETUP_MIN and RX_HOLD_MIN respectively. This will show the actual timing margin during data reception, and can be used as a quantitative indication of the quality of the channel.

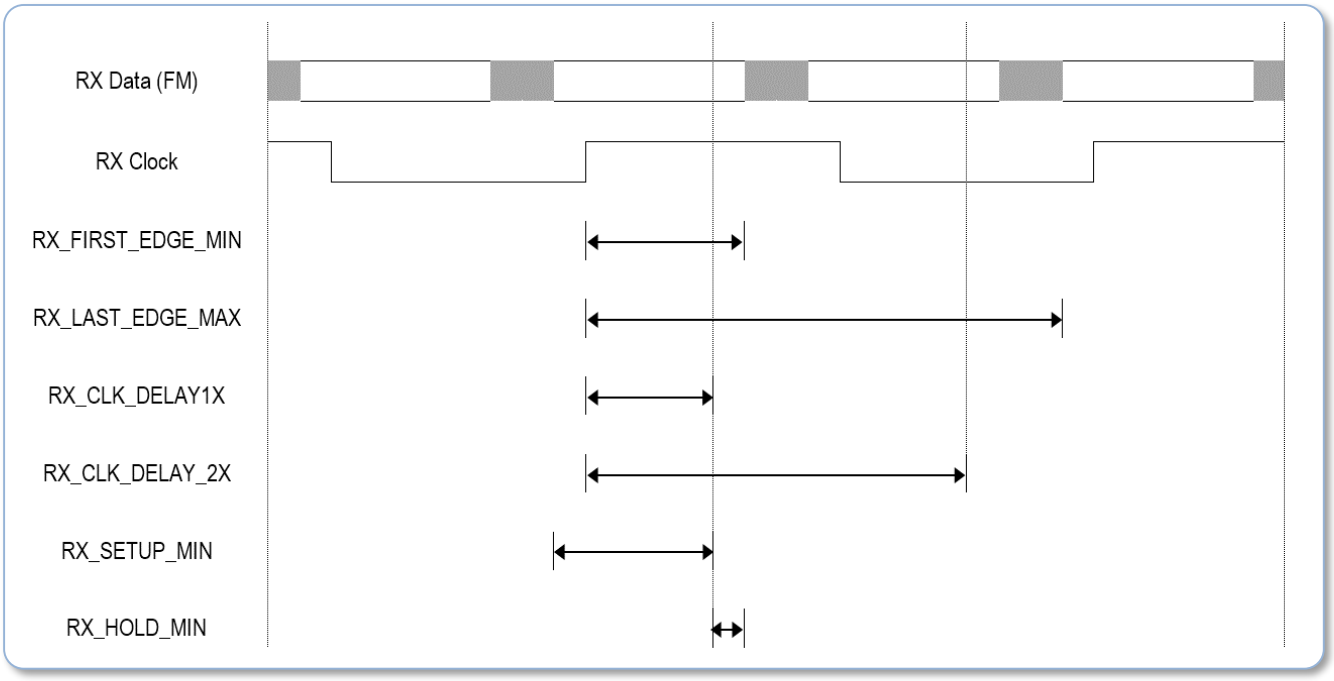


Figure 11: Receive Statistics

5.1.11 CRC GENERATION

The USART provides for automatic generation and detection of a CRC. The CRC block can accommodate any polynomial that is 8, 12, 16, or 32 bits in length. The preset value, value to test against, and output inversion are all adjustable.

During data transmission with Header Mode enabled, the CRC is preset at the beginning of the packet, and inserted into the data stream at the end of the packet if CONTROL[CRC] is set. In this case, the LENGTH field of the Header contains the length of the data only, and does not include the length of the CRC. Alternately, if Non-Packed Data is used, the CRC is preset if the CRC First bit is set, and inserted when the CRC Insert bit is set (the Data Bits and Data Bits Override must also be set). The CRC cannot be generated by the hardware if Non-Header Mode is used with Packed Data.

During data reception, the CRC is preset when an SDLC/HDLC Flag character is received. The CRC is tested at every data frame, and the CRC Ok bit is set if it passes (Non-Packer Data). Note that this bit is set whether or not it is at the end of a packet. In the RX Log FIFO, the CRC Ok bit is set if the last data value in the packet is a correct CRC.

6 Register Address Space

This product was built as a modular design, which allows each board function to have its own Functional Block (FB). Each functional block is designed to work independently. For this reason, we provide individual DMA channels, interrupts, clocks, and FIFOs to each functional block.

The registers are described by their PCIe Base Address Register (BAR), which is defined in the PCI configuration space for this board. The configuration space is generally handled by the operating system. For more information on how to use the configuration space, consult the *PCI Local Bus Specification, Revision 3.0* from the PCI-SIG.

Register Types

There are several different types of registers that are referred to in this section. A description of each type is below.

- Read/Write Registers: The value that is written to this register can also be read back.
- Maskable Registers: This is a 32 bit register that consists of 16-bit data field in the upper word and a 16-bit mask value in the lower word. For each bit in the data field, it is only written to the register if the corresponding bit in the mask field is '1'.
- Sticky Registers: This is a status read register. When a bit in this register has a value of '1', a '1' needs written to that bit to reset the register to '0'. This is typically used for interrupt status registers.
- Read Only: This register can only be read.

NOTE: Writing to Read-Only registers may have unexpected results.

Function Block Mapping

Below is the Function Block slot mapping for the SER25316HR/SER35316HR.

Table 12: Function Block Mapping

Slot	ID	Description	Connector
FB0	0x00014000	Universal Synchronous/Asynchronous Receiver/Transmitter	CN10 Port A
FB1	0x00014000	Universal Synchronous/Asynchronous Receiver/Transmitter	CN10 Port B
FB2	0x00014000	Universal Synchronous/Asynchronous Receiver/Transmitter	CN11 Port A
FB3	0x00014000	Universal Synchronous/Asynchronous Receiver/Transmitter	CN11 Port B

6.1 BAR0: General Board Control

This section contains the board-level interrupt control, as well as a description of all of the Functional Blocks in BAR2. Most of the registers in this section are Read-Only. Although a full implementation requires BAR0 to be 0x400 in length, most implementations will have a shorter length.

Table 13: Base Functional Block

Offset	0x03	0x02	0x01	0x00
0x00	GBC_BRD_RST	GBC_EOI	GBC_REV	GBC_FMT
0x04	GBC_PDP			
0x08	GBC_BUILD			
0x0C	*Reserved*	GBC_USER_ID	GBC_SYS_CLK_FREQ	
0x10	GBC_IRQ_STATUS			
0x14				
0x18	GBC_DIRQ_STATUS			
0x1C				
0x20	FB0_ID			
0x24	FB0_OFFSET			
0x28	FB0_OFFSET_DMA			
0x2C	reserved			
0x30	FB1_ID			
0x34	FB1_OFFSET			
0x38	FB1_OFFSET_DMA			
0x3C	reserved			
…				
0x20+0x10*n	FBn_ID			
0x24+0x10*n	FBn_OFFSET			
0x28+0x10*n	FBn_OFFSET_DMA			
0x2C+0x10*n	reserved			
…				
0x3F0	FB61_ID			
0x3F4	FB61_OFFSET			
0x3F8	FB61_OFFSET_DMA			
0x3FC	reserved			

6.1.1 GBC_FMT (READ-ONLY)

This register contains the format ID that is used in this board.

- B[7:4] Major Format – Changes to this value indicate changes that are not backward compatible.
  - 0 – Initial Major Format
- B[3:0] Minor Format – Changes to this value indicate additional features that are backward compatible.
  - 1 – Initial Minor Format
  - 2 – Addition of FB_DMAm_CTRLn[Pause] flag and Early DMA Buffer Termination.

6.1.2 GBC_REV (READ-ONLY)

This register contains the PDP revision for this board. A=1, B=2, etc.

6.1.3 GBC_BRD_RST (READ/WRITE)

Write a value of 0xAA to this register to reset the board.

6.1.4 GBC_PDP (READ-ONLY)

This register contains the PDP number for this board in decimal.

6.1.5 GBC_BUILD (READ-ONLY)

Unique 32-bit build number for every FPGA build.

6.1.6 GBC_SYS_CLK_FREQ (READ ONLY)

The measured frequency of the system clock. Units are 10 kHz, i.e. (Frequency in Hertz) = (GBC_SYS_CLK_FREQ * 10 kHz). This is measured once after any reset, including GBC_BRD_RST. This value is not available (will read 0) until 100us after reset.

6.1.7 GBC_USER_ID (READ ONLY)

This value is set by jumpers on the board, and is used to identify individual board when there are multiple in the same system.

6.1.8 GBC_IRQ_STATUS (READ/CLEAR)

This is a 64-bit interrupt status register for non-DMA interrupts. Each bit in this register corresponds to one of the Function Blocks; bit 0 corresponds to FB0 (whose ID and OFFSET are at 0x020), etc. Bits 60 through 63 are reserved. This is a "sticky" register, and the driver clears it by writing a '1' to the appropriate bit. The interrupts signaled by this registered are generally handled by a User Mode Delayed Procedure Call. Any status information required to service the interrupt must be latched in the Function Block. This typically means that there will also be a sticky status register in the Function Block.

The Interrupt Service Routing must be able to clear the interrupt without accessing any registers in the Function Block. This means that if a persistent interrupt condition occurs (i.e. FIFO Empty) the GBC_IRQ_STATUS register will initially be set. The ISR will then clear the bit in GBC_IRQ_STATUS and launch the DPC. In this state the interrupt condition still exists, but the status bit is cleared (because we are servicing it). Depending on implementation, that Function Block may not be able to issue another interrupt while in this state. In this state the full status of the board cannot be determined just from the GBC_IRQ_STATUS register. The Function Block should also maintain a status register that will indicate if the interrupt condition still exists, which will be cleared by the DPC. By doing an End-Of-Interrupt (see Section 6.1.10 GBC_EOI (Read/Clear) below) any "hidden" interrupts will be generated again.

6.1.9 GBC_DIRQ_STATUS (READ/CLEAR)

This is a 64-bit interrupt status register for DMA interrupts. Each bit in this register corresponds to one of the Function Blocks; bit 0 corresponds to FB0 (whose ID and OFFSET are at 0x020), etc. Bits 60 through 63 are reserved. This is a "sticky" register, and the driver clears it by writing a '1' to the appropriate bit. The interrupts signaled by this registered are generally handled by the Kernel Mode ISR. This interrupt is only for the Function Block Standard DMA in Section 6.2 on page 32.

6.1.10 GBC_EOI (READ/CLEAR)

This register is used to acknowledge an interrupt. It is used to safeguard against missing an interrupt. At the end of the ISR, write a 0x01 to this register. If there is another interrupt pending in the status registers, the interrupt line is toggled (Legacy Mode), or another interrupt is sent (MSI Mode).

6.1.11 FBN_ID (READ-ONLY)

This is a 32-bit value that identifies the type of Function Block in slot 'n'. Any changes to a Function Block result in a new Block ID. The mappings for the Functional Block ID can be found on page 29. The ID consists of three fields:

- B[31:24] Type Revision – This is used to indicate the addition of features to the function block. Any checks in the software should be of the form (Type Revision > NN) so that the driver doesn't fail with a newer FPGA. Any changes to the FPGA that would break existing software must change the Type, not the Type Revision.
- B[23:16] Subtype – A Subtype is a variant of the Type with one of the following differences. This allows the driver and library to use the same functions, but have different enumerations.
- B[15:0] Type – The Type of the Function Block is used to define the register map. Any time that the positions of registers change, or registers are added or removed require a new Type number. This is loosely separated into
    - B[15:8] Family – i.e. ADC, DAC, Digital I/O, etc.
    - B[7:0] Member

6.1.12 FBN_OFFSET (READ-ONLY)

This is the offset from the beginning of the Functional Block section (typically the offset in BAR2) that this Functional Block resides in.

6.1.13 FBN_OFFSET_DMA (READ-ONLY)

This is the offset from the beginning of the Functional Block section (typically the offset in BAR2) that the Functional Block DMA Registers resides in.

6.2 BAR2: Functional Block Standard DMA

This section describes a standard DMA engine implementation that may be used by the Functional Blocks. Any Functional Block that uses DMA shall use this implementation so that the kernel mode driver may be used to service it, resulting in better performance.

Table 14: DMA Registers

Offset	0x03	0x02	0x01	0x00
D + 0x00	FB_DMAm_Stat_Underflow	FB_DMAm_Stat_Overflow	FB_DMAm_Setup	FB_DMAm_Action
D + 0x04	FB_DMAm_Current_Buffer	FB_DMAm_Count		
D + 0x08	FB_DMAm_RD_FIFO_CNT		FB_DMAm_WR_FIFO_CNT	
D + 0x0C	FB_DMAm_Last_Action	FB_DMAm_Stat_Complete	FB_DMAm_Stat_Invalid	FB_DMAm_Stat_Used
D + 0x10	FB_DMAm_CTRL0	FB_DMAm_STAT0	Reserved	
D + 0x14	Reserved	FB_DMAm_SIZE0		
D + 0x18	FB_DMAm_ADDRESS0			
D + 0x1C				
D + 0x20	FB_DMAm_CTRL1	FB_DMAm_STAT1	Reserved	
D + 0x24	Reserved	FB_DMAm_SIZE1		
D + 0x28	FB_DMAm_ADDRESS1			
D + 0x2C				
…				
D + 0x10 + (0x10 * n)	FB_DMAm_CTRLn	FB_DMAm_STATn	Reserved	
D + 0x14 + (0x10 * n)	Reserved	FB_DMAm_SIZEn		
D + 0x18 + (0x10 * n)	FB_DMAm_ADDRESSn			
D + 0x1C + (0x10 * n)				

6.2.1 FB_DMAm_Action (Read/Write)

After writing to the Action register, the driver should poll the Last_Action register (below) until it reads the same value. This shows that the action has been performed by the DMA state machine. This is especially important when entering and exiting the Clear state.

- 0x00 = Clear: Clear the Current Buffer field, the internal offset counters, and the FIFO. DMA is stopped.
- 0x01 = Go: Starts DMA
- 0x02 = Pause: DMA transfers are stopped, but all internal registers maintain their state. During PAUSE you will still receive Stat_Underflow and Stat_Overflow interrupts. After PAUSE, you may transition to GO or CLEAR.
- 0x03 = Halt: Buffer has been filled that has the HALT bit set, or has attempted to use a buffer with the Valid bit cleared. After HALT, you must transition to CLEAR.

6.2.2 FB_DMAm_Last_Action (Read/Write)

The DMA Engine writes the value of FB_DMAm_Action to this register after it has completed this channel. This indicates to the driver that the last command has been processed. It specifically aids the transition to Clear. When transitioning to Clear, the driver should wait until FB_DMAm_Last_Action indicates that the Clear has been processed before allowing any other Action changes.

The driver may also write a value to this register and then poll the register to see when the value changes. This method can be used to detect when the DMA engine services the channel without an Action change.

6.2.3 FB_DMAm_Setup (Read/Write)

- B0:IntEna: Set to '1' to enable the DMA engine to generate interrupts on completion of a buffer.
- B1:ErrIntEna: Set to '1' to enable the DMA engine to generate interrupts on error.
- B2:Direction (RO or R/W): Set to '1' to transfer from the FPGA to the PCI bus. Clear to '0' to transfer from the PCI bus to the FPGA.
- B3:IgnoreUsed: Set to '1' to prevent an error condition when accessing a buffer with the Used bit set. When "IgnoreUsed" is set, the hardware will still set the Used_Desc bit (below) if the next buffer has the Used bit set. However, it will not generate an interrupt, and will not PAUSE the DMA engine. Examples are continuous output from a DAC, or very large Pre-trigger buffering using system memory.

6.2.4 FB_DMAm_Stat_Used (Read/Write)

This register is used to determine the source of a DMA interrupt. The bits are cleared by writing 0x00 to the byte. Stat_Used will be set regardless of having ErrIntEna set to '1', and regardless of the state of the IgnoreUsed bit (above).

- B0: Used_Desc (R/C). Set to '1' by the DMA engine if it attempting to use a descriptor with the Used bit set.

6.2.5 FB_DMAm_Stat_Invalid (Read/Write)

This register is used to determine the source of a DMA interrupt. The bits are cleared by writing 0x00 to the byte. Stat_Invalid will be set regardless of having ErrIntEna set to '1'.

- B0: Invalid_Desc (R/C). Set to '1' by the DMA engine if it attempting to use a descriptor with the Valid bit cleared.

6.2.6 FB_DMAm_Stat_Overflow (Read/Write)

This register is used to determine the source of a DMA interrupt. The bits are cleared by writing 0x00 to the byte. Stat_Overflow will be set regardless of having ErrIntEna set to '1'.

- B0: Overflow (R/C). Set to '1' by the DMA engine if an overflow occurred on the FIFO.

6.2.7 FB_DMAm_Stat_Underflow (Read/Write)

This register is used to determine the source of a DMA interrupt. The bits are cleared by writing 0x00 to the byte. Stat_Underflow will be set regardless of having ErrIntEna set to '1'.

- B0: Underflow (R/C). Set to '1' by the DMA engine if an underflow occurred on the FIFO.

6.2.8 FB_DMAm_Stat_Complete (Read/Write)

This register is used to determine the source of a DMA interrupt. The bits are cleared by writing 0x00 to the byte.

- B0: Buffer_Complete (R/C). Set to '1' by the DMA engine when a buffer is filled that has the Interrupt bit set.

6.2.9 FB_DMAm_Current_Buffer (Read-Only)

This is the ID for the buffer that will be used for the next access. The driver may use this to track the progress of the DMA activity.

6.2.10 FB_DMAm_COUNT (Read-Only)

This is the offset in the DMA buffer for the next access. The driver may use this to track the progress of the DMA activity.

6.2.11 FB_DMAm_RD_FIFO_CNT(Read-Only)

- B[9:0] This is the amount of data available in the read FIFO in bytes. Software can use this to determine when the FIFO is empty. A value of 0x3FC indicates that there are 1020 or more bytes available.
- B15: RD_EMPTY- '1' indicates that the read FIFO is empty

6.2.12 FB_DMAm_WR_FIFO_CNT(Read-Only)

- B[9:0] This is the amount of space available in the write FIFO in bytes. Software can use this to determine when the FIFO is empty. A value of 0x3FC indicated that there are 1020 or more bytes available.
- B15: WR_FULL- '1' indicates that the read FIFO is full

6.2.13 FB_DMAm_ADDRESSn (Read/Write)

This is the 64-bit PCI address for DMA Channel m, buffer n. It must be double-word aligned (i.e. b[1:0] are reserved).

6.2.14 FB_DMAm_SIZEn (Read/Write)

This is the size in bytes of the buffer for DMA Channel m, buffer n. It must be an integer number of double-words (i.e. b[1:0] are reserved). The actual size is FB_DMAm_SIZEn + 4 Bytes. The maximum buffer size is 16MB.

6.2.15 FB_DMAM_CTRLN (READ/WRITE)

- B0: Valid: Driver sets to '1' to indicate that this contains valid information. The DMA engine will set the error bit and halt if it is ready to use this descriptor and it is not valid.
- B1: Halt: Driver sets to '1' to halt the DMA engine after this buffer is full.
- B2: Loop: Driver sets to '1' to start back at descriptor 0 after this buffer is full. This has a higher priority than the HALT bit.
- B3: Interrupt: Driver sets to '1' to generate an interrupt after this buffer is full.
- B4: Pause: Driver sets to '1' to pause the DMA engine after this buffer is full.

If the last buffer is reached, and the HALT and LOOP bits are both '0', the DMA engine will loop.

If the last buffer is reached, and the HALT and LOOP bits are both '1', the DMA engine will halt and the Current_Buffer will be set to 0.

6.2.16 FB_DMAM_STATN (READ/CLEAR)

- B0: Used (R/C): DMA engine sets to '1' to indicate that it has completely used this descriptor. The driver must clear this bit when it is ready to be used again. The DMA engine will set the error bit and PAUSE if it is ready to use this descriptor and the Used bit is set, unless the IgnoreUsed bit is set. The bits are cleared by writing 0x00 to the byte.
- B1: Terminated (R/C): DMA engine sets to '1' to indicate that this buffer was terminated early at the request of the associated Function Block.

6.3 BAR2: Functional Block Header Registers

Each Functional Block has a standard header as follows.

Table 15: Functional Block Header

Offset	0x03	0x02	0x01	0x00
FB + 0x00	FB_ID			
FB + 0x04	FB_DMA_BUFFERS	FB_DMA_CHANNELS	Reserved	Reserved

6.3.1 FB_ID (READ-ONLY)

This is a repeat of the ID from the General Board Control section. This can be used to validate that the correct Function Block is being accessed.

6.3.2 FB_DMA_CHANNELS (READ -ONLY)

This register contains the number of DMA Channels in this Function Block.

6.3.3 FB_DMA_BUFFERS (READ-ONLY)

This register contains the number of Buffer Descriptors in each DMA Channel.

6.4 BAR2: Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

6.4.1 FUNCTION BLOCK REGISTER MAP

Table 16: USART Functional Block

	Offset	0x03	0x02	0x01	0x00
Header	FB + 0x00	FB_ID			
	FB + 0x04	FB_DMA_BUFFERS	FB_DMA_CHANNELS	*Reserved*	
USART	FB + 0x08	ENCODING	FRAME_FORMAT	XCVR_MODE	MODE
	FB + 0x0C	IDLE_BITS	DEFAULT_DATA_BITS	*Reserved*	FIFO_DATA_FORMAT
	FB + 0x10	IDLE_CHARACTER			
	FB + 0x14	*Reserved*	*Reserved*	ASYNC_STAT_LINES	ASYNC_CON_LINES
	FB + 0x18	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0x1C	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0x20	BIT_PERIOD			
	FB + 0x24	*Reserved*	*Reserved*	*Reserved*	CLKA_OUT_SEL
	FB + 0x28	*Reserved*	AUXB_OUT_SEL	*Reserved*	AUXA_OUT_SEL
	FB + 0x2C	*Reserved*	*Reserved*	*Reserved*	AUX_IN_STAT
	FB + 0x30	*Reserved*	*Reserved*	*Reserved*	TX_CLOCK_MODE
	FB + 0x34	TX_CLK_DELAY1X			
	FB + 0x38	TX_CLK_DELAY2X			
	FB + 0x3C	TX_SEL_CLK_PERIOD			
	FB + 0x40	*Reserved*	*Reserved*	*Reserved*	RX_CLOCK_MODE
	FB + 0x44	RX_CLK_DELAY1X			
	FB + 0x48	RX_CLK_DELAY2X			
	FB + 0x4C	RX_SEL_CLK_PERIOD			
	FB + 0x50	RX_FIRST_EDGE_MIN			
	FB + 0x54	RX_LAST_EDGE_MAX			
	FB + 0x58	RX_SETUP_MIN			
	FB + 0x5C	RX_HOLD_MIN			
	FB + 0x60	CRC_POLYNOMIAL			
	FB + 0x64	CRC_PRESET_VALUE			
	FB + 0x68	CRC_TEST_VALUE			
	FB + 0x6C	*Reserved*	*Reserved*	CRC_INVERT_OUTPUT	CRC_SIZE
	FB + 0x70	TX_INT_ENA			
	FB + 0x74	*Reserved*	*Reserved*	TX_INT_STAT	
	FB + 0x78	TX_BIT_TIME			
	FB + 0x7C	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0x80	*Reserved*	*Reserved*	TX_BURST_CLK_PERIOD	
	FB + 0x84	*Reserved*	*Reserved*	TX_BURST_CLK_LENGTH	
	FB + 0x88	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0x8C	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0x90	RX_INT_ENA			
	FB + 0x94	*Reserved*	*Reserved*	RX_INT_STAT	
	FB + 0x98	RX_BIT_TIME			
	FB + 0x9C	*Reserved*	*Reserved*	*Reserved*	RX_FRAME_SYNC
	FB + 0xA0	RX_PACKET_BYTE_CNT_MAX			
	FB + 0xA4	*Reserved*	*Reserved*	*Reserved*	RX_PACKET_BYTE_CNT _LIMIT
	FB + 0xA8	RX_ASYNC_IDLE_TIME_MAX			
	FB + 0xAC	*Reserved*	*Reserved*	RX_PACKET_CNT_MAX	
	FB + 0xB0	*Reserved*	*Reserved*	RX_BURST_CLK_PERIOD	
	FB + 0xB4	*Reserved*	*Reserved*	RX_BURST_CLK_LENGTH	
	FB + 0xB8	*Reserved*	*Reserved*	*Reserved*	RX_STATUS
	FB + 0xB8 - FB + 0xDC	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0xE0	TX_FIFO_ACCESS			
	FB + 0xE4	RX_DATA_FIFO_ACCESS			
	FB + 0xE8	RX_LOG_FIFO_ACCESS			

Table 16: USART Functional Block

	Offset	0x03	0x02	0x01	0x00
	FB + 0xEC	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0xF0	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0xF4	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0xF8	*Reserved*	*Reserved*	*Reserved*	*Reserved*
	FB + 0xFC	*Reserved*	*Reserved*	*Reserved*	*Reserved*

6.4.2 MODE (READ/WRITE)

Selects the current mode of operation.

- 0x00 – **Halt:** USART is stopped. All counters and state machines are reset. The TXD pin is held at a logical '1'.
- 0x01 – **Idle:** USART is active, but no data is sent or received. Output clocks are active. The TXD pin is held at a logical '1'.
- 0x02 – **Enable:** USART is fully active

6.4.3 XCVR_MODE (READ/WRITE)

Selects the current mode of RS-232/RS-485/RS-422 transceiver.

- b[3:0] - **Mode**
  - 0x00 – **Disable**: Transceiver is disabled.
  - 0x01 – **Loop**: Internal loopback from TX to RX.
  - 0x02 – **RS-232 Asynchronous:** Transceiver is in RS-232 mode with asynchronous control signals.
  - 0x03 – **RS-232 Synchronous:** Transceiver is in RS-232 mode with CLKA and CLKB.
  - 0x04 – **RS-422 Asynchronous / Synchronous**: Transceiver is in RS-422 mode.
  - 0x05 – **RS-485 Asynchronous**: Transceiver is in RS-485 mode. Direction controlled by RTS signal.
- b[4] – **Slew Limit**
  - 0 – No slew limit. Maximum data rate is 20 Mbps (RS-422, RS-485) or 1 Mbps (RS-232)
  - 1 – Output is slew limited. Maximum data rate is 250 kbps.
- b[5] - **Termination**
  - 0 – No termination
  - 1 – 120 Ohm termination on all inputs (RS422 and RS-485 mode only).

6.4.4 FRAME_FORMAT (READ/WRITE)

Selects format of the character frame. For synchronous communications, typically all of these are disabled.

- b[0] - **Start Bit**
  - 0 – **Disabled**: No start bit is used. Typical setting for synchronous communications.
  - 1 – **Enabled**: Start bit is used. Typical setting for asynchronous communications.
- b[2:1] - **Stop Bits**
  - 0x0 – **None**: No stop bits are used. Typical setting for synchronous communication.
  - 0x1 – **One**: One stop bit is used.

- 0x2 – **Two** stop bits are used.

- b[3] - **Parity Bit**
  - 0 – **Disabled**: No parity bit is used.
  - 1 – **Enabled**: Parity bit is used.

- b[4] - **Parity**
  - 0 – **Even**: Parity will result in an even number of bits set.
  - 1 – **Odd**: Parity will result in an odd number of bits set.

6.4.5 ENCODING (READ/WRITE)

Selects the encoding.

- b[2:0] - **Encoding**
  - 0x00 – **NRZ**: Non-return-to-zero. Transmitting a 1 results in a High at the output, transmitting a 0 results in a Low at the output. This is the standard encoding.
  - 0x01 – **NRZI**: Non-Return-to-Zero-Inverted. Transmitting a 1 results in no change to the output, transmitting a 0 results in a change to the output.
  - 0x02 – **FM0**: Bi-Phase Space. There is a transition on every bit cell boundary. Transmitting a 0 also results in a transition in the middle of the bit time.
  - 0x03 – **FM1**: Bi-Phase Mark. There is a transition on every bit cell boundary. Transmitting a 1 also results in a transition in the middle of the bit time.
  - 0x04 – **Manchester**: Bi-Phase Level. Transmitting a 1 results in a High to Low transition at the center of the bit time. Transmitting a 0 results in a Low to High transition at the center of the bit time.

- b[3] - **Invert**: When enabled, the TXD signal is inverted after Encoding, and the RXD is inverted before Decoding
  - 0 – **Disabled**
  - 1 – **Enabled**

- b[4] – **Zero Insertion:** Used in SDLC/HDLC synchronous communications. For a non-flag character, if there are six consecutive "1" bits in the serial stream, the sixth is replaced with "01".
  - 0 – **Disabled**
  - 1 – **Enabled**

- b[5] – **One Insertion:** Analogous to Zero Insertion. For a non-flag character, if there are six consecutive "0" bits in the serial stream, the sixth is replaced with "10".
  - 0 – **Disabled**
  - 1 – **Enabled**

6.4.6 FIFO_DATA_FORMAT (READ/WRITE)

Selects the output encoding.

- b[0] - **TX Header**: When enabled, the data in the transmit FIFO must include a header. See TX FIFO Format section for more details. This is the typical operating mode for SDLC/HDLC communications.
  - 0 – **Disabled**
  - 1 – **Enabled**

- b[1] - **Pack Data**
    - 0 – **Disabled**: Each frame is stored in a 32-bit word along with meta-data.
    - 1 – **Enabled**: Four frames of data only are stored in a 32-bit word.
- b[2] - **RX Data Packet Terminate:** When enabled, the current RX Data DMA buffer is terminated when RX_PACKET_CNT_MAX packets are received, RX_PACKET_BYTE_CNT_MAX is exceeded, or RX_ASYNC_IDLE_MAX is exceeded. The Buffer Descriptor is updated with the actual bytes transferred, and the "Complete" interrupt is generated if enabled.
    - 0 – **Disabled**
    - 1 – **Enabled**
- b[3] - **RX Log Packet Terminate:** When enabled, the current RX Log DMA buffer is terminated when RX_PACKET_CNT_MAX packets are received, RX_PACKET_BYTE_CNT_MAX is exceeded, or RX_ASYNC_IDLE_MAX is exceeded. The Buffer Descriptor is updated with the actual bytes transferred, and the "Complete" interrupt is generated if enabled.
    - 0 – **Disabled**
    - 1 – **Enabled**
- b[4] – **Reset Offset on Terminate:** When enabled, the START_OFFSET field of the RX Log is reset to zero after a buffer termination.
    - 0 – **Disabled**
    - 1 – **Enabled**

6.4.7 DEFAULT_DATA_BITS (READ/WRITE)

Sets the number of data bits in the character frame. This can be overridden by the TX meta-data when FIFO_DATA_FORMAT[Pack Data] is Disabled.

The value of this register is interpreted as below:

DEFAULT_DATA_BITS = (Data bits in frame) - 1

6.4.8 IDLE_BITS (READ/WRITE)

Sets the number of data bits to send for the idle character. The value of this register is interpreted as below:

IDLE_BITS = (Data bits in frame) - 1

6.4.9 IDLE_CHARACTER (READ/WRITE)

The character to transmit when idle, i.e. when there is no data in the transmit FIFO. The number of bits to send is set in IDLE_BITS above. The idle character bypasses framing (start bits, stop bits, etc.) and zero/one insertion. Typical values for the idle character are shown below.

Table 17: Typical Idle Characters

	IDLE_CHARACTER	IDLE_BITS	Note
SDLC/HDLC	0x0000007E	0x07	8-bit Flag character
Asynchronous	0xFFFFFFFF	0x00	1-bit Idle

6.4.10 ASYNC_CON_LINES (READ/WRITE)

Sets the state of the Asynchronous Control lines.

- b[0] - **RTS**
    - 0 – **De-asserted**
    - 1 – **Asserted**

- b[1] - **DTR**
    - 0 – **De-asserted**
    - 1 – **Asserted**

6.4.11 ASYNC_STAT_LINES (READ ONLY)

Reflects the state of the Asynchronous Status lines.

- b[0] - **CTS**
    - 0 – **De-asserted**
    - 1 – **Asserted**
- b[1] - **DSR**
    - 0 – **De-asserted**
    - 1 – **Asserted**
- b[2] - **DCD**
    - 0 – **De-asserted**
    - 1 – **Asserted**
- b[3] - **RI**
    - 0 – **De-asserted**
    - 1 – **Asserted**

6.4.12 BIT_PERIOD (READ/WRITE)

Divider for the bit rate of the port. This must be set even when an external clock is used. The value of this register is interpreted as below:

BIT_PERIOD = [2 * (GBC_SYS_CLK_FREQ * 10 kHz) / (Desired bit rate)] – 1

The minimum value for BIT_PERIOD is 7.

6.4.13 CLKA_OUT_SEL (READ/WRITE)

Selects the source and direction for Clock A.

- b[0] - **Output Enable**
    - 0 – **Disabled:** Clock A is an input only.
    - 1 – **Enabled**: Clock A output is enabled.
- b[1] - **Select**
    - 0 – **TX Clock:** Clock A output source is the TX Clock.
    - 1 – **RX Clock:** Clock A output source is the RX Clock.

6.4.14 AUXA_OUT_SEL (READ/WRITE)

Selects the source and direction for Auxiliary A.

- 0x00 – **Disabled:** Aux A is an input only.
- 0x01– **TX Clock:** Aux A output source is the TX Clock.
- 0x02 – Reserved
- 0x03 – **RX Clock:** Aux A output source is the RX Clock.
- 0x04 – **De-asserted:** Aux A is forced low.
- 0x05 – **Asserted:** Aux A is forced high.

6.4.15 AUXB_OUT_SEL (READ/WRITE)

Selects the source and direction for Auxiliary BS.

- 0x00 – **Disabled:** Aux B is an input only.
- 0x01– **TX Clock:** Aux B output source is the TX Clock.
- 0x02 – Reserved
- 0x03 – **RX Clock:** Aux B output source is the RX Clock.
- 0x04 – **De-asserted:** Aux B is forced low.
- 0x05 – **Asserted:** Aux B is forced high.

6.4.16 AUX_IN_STAT (READ ONLY)

Reflects the state of the Auxiliary Input lines.

- b[0] - **Aux A**
  - 0 – **De-asserted**
  - 1 – **Asserted**
- b[1] - **Aux B**
  - 0 – **De-asserted**
  - 1 – **Asserted**

6.4.17 TX_CLOCK_MODE (READ/WRITE)

Selects the clock mode for the transmit path.

- b[2:0] - **Source**: Select the source of the clock.
  - 0x00 – **BRG:** Baud Rate Generator. The clock is generated based on BIT_PERIOD.
  - 0x01 – **DATA**: The clock is recovered from RX Data. There must be at least one transition on RX Data per bit period, i.e. FM0, FM1, or Manchester encoding must be used.
  - 0x02 – **CLKA**: The Clock A Input (CLKA) is used as the clock source. This is typically the transmit clock, and can be an input or an output.
  - 0x03 – **CLKB**: The Clock B Input (CLKB) is used as the clock source. This is typically the receive clock, and is an input only.

- 0x04- **RX:** The selected Receive clock is used as the clock source.

- b[3] - **Invert**: When enabled, the selected clock is inverted.
  - 0 – **Disabled:** Clock based on the rising edge.
  - 1 – **Enabled**: Clock based on the falling edge.

- b[4] - **Gate**: When enabled, the clock output is inhibited when the TX is idle. The IDLE_CHARACTER will not be sent in this mode.
  - 0 – **Disabled**
  - 1 – **Enabled**

6.4.18 TX_CLK_DELAY1X (READ/WRITE)

Sets the delay from the clock edge selected in TX_CLOCK_MODE until the primary data transition.

TX_CLK_DELAY1X = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Desired delay)

6.4.19 TX_CLK_DELAY2X (READ/WRITE)

Sets the delay from the clock edge selected in TX_CLOCK_MODE until the secondary data transition. This is the transition in the middle of the bit time for FM0, FM1, and Manchester encoding.

TX_CLK_DELAY2X = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Desired delay)

6.4.20 TX_SEL_CLK_PERIOD (READ ONLY)

Returns the measured clock period of the clock selected in TX_CLOCK_MODE. This can be used to verify the actual clock rate to accurately set TX_CLK_DELAY1X and TX_CLK_DELAY2X when using an external clock. The value of this undefined when TX_CLOCK_MODE[Source] = BRG.

TX_SEL_CLK_PERIOD = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Period) -1

6.4.21 RX_CLOCK_MODE (READ/WRITE)

Selects the clock mode for the receive path.

- b[2:0] - **Source**: Select the source of the clock.
  - 0x00 – **BRG:** Baud Rate Generator. The clock is generated based on BIT_PERIOD.
  - 0x01 – **DATA**: The clock is recovered from RX Data. There must be at least one transition on RX Data per bit period, i.e. FM0, FM1, or Manchester encoding must be used.
  - 0x02 – **CLKA**: The Clock A Input (CLKA) is used as the clock source. This is typically the transmit clock, and can be an input or an output.
  - 0x03 – **CLKB**: The Clock B Input (CLKB) is used as the clock source. This is typically the receive clock, and is an input only.
  - 0x04- **TX:** The selected Transmit clock is used as the clock source.

- b[3] - **Invert**: When enabled, the selected clock is inverted.
  - 0 – **Disabled:** Clock based on the rising edge.
  - 1 – **Enabled**: Clock based on the falling edge.

6.4.22 RX_CLK_DELAY1X (READ/WRITE)

Sets the delay from the clock edge selected in RX_CLOCK_MODE until the primary data is sampled.

RX_CLK_DELAY1X = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Desired delay)

6.4.23 RX_CLK_DELAY2X (READ/WRITE)

Sets the delay from the clock edge selected in RX_CLOCK_MODE until the secondary data is sampled. This is the second half of the bit time for FM0, FM1, and Manchester encoding.

RX_CLK_DELAY2X = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Desired delay)

6.4.24 RX_SEL_CLK_PERIOD (READ ONLY)

Returns the measured clock period of the clock selected in RX_CLOCK_MODE. This can be used to verify the actual clock rate to accurately set RX_CLK_DELAY1X and RX_CLK_DELAY2X when using an external clock. The value of this undefined when RX_CLOCK_MODE[Source] = BRG.

RX_SEL_CLK_PERIOD = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Period) -1

6.4.25 RX_FIRST_EDGE_MIN (READ/CLEAR)

Returns the delay from the clock edge selected in RX_CLOCK_MODE until the first transition of RXD. This register is updated every bit period with the minimum value. This can be used in conjunction with RX_LAST_EDGE_MAX to set RX_CLK_DELAY1X and RX_CLK_DELAY2X.

Any write to the register sets the value to 0xFFFFFFFF, effectively restarting the statistic collection.

RX_FIRST_EDGE_MIN = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Delay) -1

6.4.26 RX_LAST_EDGE_MAX (READ/CLEAR)

Returns the delay from the clock edge selected in RX_CLOCK_MODE until the last transition of RXD. This register is updated every bit period with the maximum value. This can be used in conjunction with RX_FIRST_EDGE_MIN to set RX_CLK_DELAY1X and RX_CLK_DELAY2X.

Any write to the register sets the value to 0x00000000, effectively restarting the statistic collection.

RX_LAST_EDGE_MAX = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Delay) -1

6.4.27 RX_SETUP_MIN (READ/CLEAR)

Returns the delay from the last transition of RXD until the sample at RX_CLK_DELAY1X. This register is updated every bit period with the minimum value. This can be used to verify the clock settings.

Any write to the register sets the value to 0xFFFFFFFF, effectively restarting the statistic collection.

RX_SETUP_MIN = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Delay) -1

6.4.28 RX_HOLD_MIN (READ/CLEAR)

Returns the delay from the sample at RX_CLK_DELAY1X until the first transition of RXD. This register is updated every bit period with the minimum value. This can be used to verify the clock settings.

Any write to the register sets the value to 0xFFFFFFFF, effectively restarting the statistic collection.

RX_HOLD_MIN = 2 * (GBC_SYS_CLK_FREQ * 10 kHz) * (Delay) -1

6.4.29 CRC_POLYNOMIAL (READ/WRITE)

Contains the value for the CRC calculation polynomial. See Table 18 below for some common CRC settings.

Table 18: Typical CRC Settings

	CRC_POLYNOMIAL	CRC_PRESET_VALUE	CRC_TEST_VALUE	CRC_SIZE	CRC_INVERT
SDLC/HDLC, 0 Preload	0x1021	0x0000	0x1D0F	0x02	0x01
SDLC/HDLC, 1 Preload	0x1021	0xFFFF	0x1D0F	0x02	0x01
CRC-16	0x8005	0x0000	0x800D	0x02	0x01
CRC-32	0x04C11DB7	0x00000000	0x00000000	0x03	0x00

6.4.30 CRC_PRESET_VALUE (READ/WRITE)

Contains the initial or preset value for the CRC calculation. See Table 18 above for some common CRC settings.

6.4.31 CRC_TEST_VALUE (READ/WRITE)

Contains the expected value for the CRC for an error-free packet that includes the CRC. See Table 18 above for some common CRC settings.

6.4.32 CRC_SIZE (READ/WRITE)

The size of the CRC polynomial. See the Table below. See Table 18 above for some common CRC settings.

Table 19: CRC_SIZE

Size	CRC_SIZE
8-bit	0x00
12-bit	0x01
16-bit	0x02
32-bit	0x03

6.4.33 CRC_INVERT_OUTPUT (READ/WRITE)

Contains the expected value for the CRC for an error-free packet that includes the CRC. See Table 18 above for some common CRC settings.

- 0x00 – **Disabled:** The CRC output is not inverted.
- 0x01 – **Enabled**: The CRC output is inverted. This is the typical for SDLC/HDLC communications.

6.4.34 TX_INT_ENA (MASKABLE READ/WRITE)

Each bit corresponds to an interrupt source. A value of '1' enables the source, and a value of '0' disables it. See below for a description of the sources.

6.4.35 TX_INT_STAT (READ/CLEAR)

Each bit corresponds to an interrupt source. Reading a value of '1' indicates that an event has occurred. Reading a value of '0' indicates that the event has not occurred. Writing a '1' will clear that bit.

- b[0] - **Idle**: This bit is set when the transmitter transitions from non-idle to idle.
- b[1] – **Packet Underrun Error**: When in Header Mode, indicates that the TX FIFO was empty before the end of the packet as defined by the BYTE_COUNT in the TX Header. Idle characters are then inserted until data is available again.

6.4.36 TX_BIT_TIME (READ ONLY)

Contains the number of transmit bit-times that have elapsed. This register is cleared when MODE=Halt. This can be used to schedule transmit packets.

6.4.37 TX_BURST_CLK_PERIOD (READ/WRITE)

Sets the number of bit clocks in a bursted bit-clock cycle for the Transmit Clock. Write a 0 to this register to disable bursted clock. This register may be modified while the USART is active.

Bursted bit-clock allows the effective data rate to be modulated while keeping the clock rate constant. Every TX_BURST_CLK_PERIOD number of clock cycles, the first TX_BURST_CLK_LENGTH clock cycles are generated with their respective data. The remainder of the TX_BURST_CLK_PERIOD, the clock is held low and the data is suspended.

TX_BURST_CLK_PERIOD = (Clock Periods) -1

6.4.38 TX_BURST_CLK_LENGTH (READ/WRITE)

Sets the number of bit clocks per burst in a bursted bit-clock cycle for the Transmit Clock. This register may be modified while the USART is active.

TX_BURST_CLK_LENGTH = (Clock Periods) -1

6.4.39 RX_INT_ENA (MASKABLE READ/WRITE)

Each bit corresponds to an interrupt source. A value of '1' enables the source, and a value of '0' disables it. See below for a description of the sources.

6.4.40 RX_INT_STAT (READ/CLEAR)

Each bit corresponds to an interrupt source. Reading a value of '1' indicates that an event has occurred. Reading a value of '0' indicates that the event has not occurred. Writing a '1' will clear that bit.

- b[0] - **Decoding Error**: Indicates an illegal transition in the FM0, FM1, or Manchester Encoding
- b[1] – **Bit Insertion Error**: Indicates more ones or zeros than what is allowed by the bit insertion rules. This is also asserted when an SLDC Flag is detected.
- b[2] – **Parity Error**: Indicates that a parity error has been detected.
- b[3] – **Stop Error**: Also called Framing Error. Indicates that an insufficient number of Stop bits were received.
- b[4] – **Sync Before End of Data**: Indicates that an SDLC Flag character was detected during the data phase of the frame. This indicates a loss of character sync.
- b[5] – **Sync Before End of Frame**: Indicates that an SDLC Flag character was detected during the frame. Normally the Flag is only detected at the frame boundary. This indicates a loss of character sync.
- b[6] – **Flag Detected**: Indicates that an SDLC Flag character has been detected.
- b[7] – **CRC Ok**: Indicates that a valid CRC has been detected. Note that the CRC is continuously checked, so this does not necessarily indicate that the valid CRC was at the end of the packet.
- b[8]- **Data Ready**: A byte (or word) of data has been received.
- b[9]- **First Flag**: An SDLC/HDLC flag has been detected after a non-flag character. This indicates the end of a packet.
- b[10] – **Async Idle**: Indicates that the packet was terminated because the idle time exceeded RX_ASYNC_IDLE_TIME_MAX.
- b[11] – **Hunt Done:** In SDLC/HDLC mode, indicates that a flag character has been received, and therefore byte synchronization has been achieved.
- b[15] – **Packet Log Sent**: Indicates that a packet log has been transferred to the DMA buffer. Note that does not necessarily happen after all of the received data has been transferred. It is possible to get an interrupt indicating that the Log is available before the Data Buffer Complete interrupt for the RX Data DMA channel.

6.4.41 RX_BIT_TIME (READ ONLY)

Contains the number of receive bit-times that have elapsed. This register is cleared when MODE=Halt. This is the value that is used in the receive log.

6.4.42 RX_FRAME_SYNC (READ/WRITE)

Sets the character frame synchronization mode for the receiver. This is also used to provide byte synchronization. If a Start Bit is used, that will synchronize bytes regardless of the RX_WORD_SYNC setting.

- b[2:0] - **Source**
  - 0x0 – **None:** There is no synchronization. Synchronization will still be provided by the Start Bit, if enabled. This is the typical setting for Asynchronous communications.
  - 0x1 – **Flag**: The byte boundaries are synchronized on the SDLC/HDLC flag character. This is the typical for SDLC/HDLC communications.
- b[3] **– Sequential Flag Suppression:** When set, repeated flag characters are not sent to the RX FIFO. If there are multiple sequential flags, only the first is stored.

6.4.43 RX_PACKET_BYTE_CNT_MAX (READ/WRITE)

Sets maximum packet size in units of character frames as set in DEFAULT_DATA_BITS (typically bytes). This register has no effect when RX_PACKET_BYTE_CNT_MAX[Enable] = 0. When the maximum packet size is exceeded, the packet is terminated, and any additional data is stored in a new packet.

RX_PACKET_BYTE_CNT_MAX = (Desired maximum bytes)

6.4.44 RX_PACKET_BYTE_CNT_LIMIT (READ/WRITE)

Enables or disables the Packet Byte Count Limit Checking set in RX_PACKET_BYTE_CNT_MAX.

- 0x00 – **Disabled:** Packet Byte Count Limit Checking is disabled.
- 0x01 – **Enabled**: Packet Byte Count Limit Checking is enabled. The maximum is set in RX_PACKET_BYTE_CNT_MAX.

6.4.45 RX_ASYNC_IDLE_TIME_MAX (READ/WRITE)

Sets the maximum number of idle bit times to wait before terminating a packet in asynchronous mode. A value of 0 disables packet termination due to idle time. It is also disabled if FRAME_FORMAT[Start] = 0.

RX_ASYNC_IDLE_TIME_MAX= (GBC_SYS_CLK_FREQ * 10 kHz) * (Idle Time)

6.4.46 RX_PACKET_CNT_MAX (READ/WRITE)

Sets the number of packets that the DMA Buffer Termination logic will use to terminate a DMA buffer.

RX_PACKET_CNT_MAX = (Desired number of packets) - 1

6.4.47 RX_BURST_CLK_PERIOD (READ/WRITE)

Sets the number of bit clocks in a bursted bit-clock cycle for the Receive Clock. Write a 0 to this register to disable bursted clock. This register may be modified while the USART is active.

Bursted bit-clock allows the effective data rate to be modulated while keeping the clock rate constant. Every TX_BURST_CLK_PERIOD number of clock cycles, the first TX_BURST_CLK_LENGTH clock cycles are generated with their respective data. The remainder of the TX_BURST_CLK_PERIOD, the clock is held low and the data is suspended.

RX_BURST_CLK_PERIOD = (Clock Periods) -1

6.4.48 RX_BURST_CLK_LENGTH (READ/WRITE)

Sets the number of bit clocks per burst in a bursted bit-clock cycle for the Receive Clock. This register may be modified while the USART is active.

RX_BURST_CLK_LENGTH = (Clock Periods) -1

6.4.49 RX_STATUS (READ ONLY)

Indicates the status of the Receive path.

- b[0] - **Hunt**: '1' indicates receiver is hunting for the SDLC/HDLC Flag. No data is being received.

6.4.50 TX_FIFO_ACCESS (READ/WRITE)

This register provides direct access to the TX DMA FIFO. It can be used to access the data without the use of the DMA engine when the DMA engine for this channel must be set to "Pause."

6.4.51 RX_DATA_FIFO_ACCESS (READ/WRITE)

This register provides direct access to the RX Data DMA FIFO. It can be used to access the data without the use of the DMA engine when the DMA engine for this channel must be set to "Pause."

6.4.52 RX_LOG_FIFO_ACCESS (READ/WRITE)

This register provides direct access to the RX Log DMA FIFO. It can be used to access the data without the use of the DMA engine when the DMA engine for this channel must be set to "Pause."

7 Troubleshooting

If you are having problems with your system, please try the following initial steps:

- **Simplify the System** – Remove modules one at a time from your system to see if there is a specific module that is causing a problem. Perform you troubleshooting with the least number of modules in the system possible.

- **Swap Components** – Try replacing parts in the system one at a time with similar parts to determine if a part is faulty or if a type of part is configured incorrectly.

If problems persist, or you have questions about configuring this product, contact RTD Embedded Technologies via the following methods:

Phone: +1-814-234-8087
E-Mail: techsupport@rtd.com

Be sure to check the RTD web site (http://www.rtd.com) frequently for product updates, including newer versions of the board manual and application software.

8 Additional Information

8.1 PC/104 Specifications

A copy of the latest PC/104 specifications can be found on the webpage for the PC/104 Embedded Consortium:

www.pc104.org

8.2 PCI and PCI Express Specification

A copy of the latest PCI and PCI Express specifications can be found on the webpage for the PCI Special Interest Group:

www.pcisig.com

9 Limited Warranty

RTD Embedded Technologies, Inc. warrants the hardware and software products it manufactures and produces to be free from defects in materials and workmanship for one year following the date of shipment from RTD Embedded Technologies, Inc. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, RTD Embedded Technologies will repair or replace, at its option, any defective products or parts at no additional charge, provided that the product is returned, shipping prepaid, to RTD Embedded Technologies. All replaced parts and products become the property of RTD Embedded Technologies. Before returning any product for repair, customers are required to contact the factory for a Return Material Authorization (RMA) number.

This limited warranty does not extend to any products which have been damaged as a result of accident, misuse, abuse (such as: use of incorrect input voltages, improper or insufficient ventilation, failure to follow the operating instructions that are provided by RTD Embedded Technologies, "acts of God" or other contingencies beyond the control of RTD Embedded Technologies), or as a result of service or modification by anyone other than RTD Embedded Technologies. Except as expressly set forth above, no other warranties are expressed or implied, including, but not limited to, any implied warranties of merchantability and fitness for a particular purpose, and RTD Embedded Technologies expressly disclaims all warranties not stated herein. All implied warranties, including implied warranties for merchantability and fitness for a particular purpose, are limited to the duration of this warranty. In the event the product is not free from defects as warranted above, the purchaser's sole remedy shall be repair or replacement as provided above. Under no circumstances will RTD Embedded Technologies be liable to the purchaser or any user for any damages, including any incidental or consequential damages, expenses, lost profits, lost savings, or other damages arising out of the use or inability to use the product.

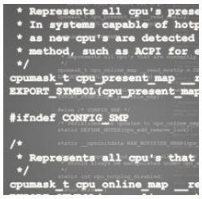
Some states do not allow the exclusion or limitation of incidental or consequential damages for consumer products, and some states do not allow limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

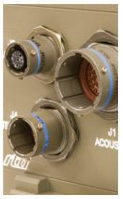
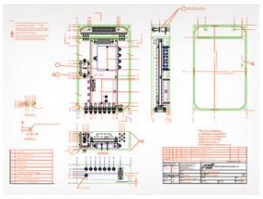
This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803 USA
Telephone: 814-234-8087
Fax: 814-234-5218

www.rtd.com

sales@rtd.com
techsupport@rtd.com



Copyright 2016 by RTD Embedded Technologies, Inc. All rights reserved.