Orange Tree Technologies

ZestET2

User Guide

Version	1.00
Date	1st May 2015

Version	Date	Comment
1.00	01/05/15	First Version

Support

If you have any questions please email **support@orangetreetech.com**

Disclaimer

This document provides outline information only. Orange Tree Technologies reserves the right to change this document without notice. Orange Tree Technologies makes no warranty of any kind, expressed or implied, with regard to any information contained in this document, including, but not limited to, the implied warranties of merchantability or fitness for any particular purpose. Further, Orange Tree Technologies does not warrant the accuracy or completeness of the information, text, graphics, or other items contained in this document. Orange Tree technologies cannot accept any liability for loss or damages arising from the use of this manual or the use of products described in it.

Orange Tree Technologies products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use Orange Tree Technologies products in these types of equipment or applications. For all restrictions on use of Orange Tree Technologies products see Orange Tree Technologies Terms and Conditions of Sale.

Contents

List of Figures

1 Introduction

The ZestET2 family are FPGA modules with a very high performance Gigabit Ethernet interface. For the highest possible performance the Ethernet communications protocols are implemented in hardware in Orange Tree Technologies' proprietary GigExpedite chip ("GigEx"). This chip is known as a TCP/IP Offload Engine (TOE) as the communications protocols are offloaded from the FPGA and are run instead in GigEx hardware. The user doesn't have to implement any Ethernet communications protocols, and the User FPGA is completely free for the user application.

ZestET2-NJ is the first module of the family and the NJ suffix (No Jack) means that there is no RJ45 on the module and this needs to be put on the user's carrier board. This gives flexibility in positioning the RJ45 jack and minimises the overall height of the RJ45 and the module.

ZestET2 can be used in a variety of applications where an FPGA with Ethernet interface is required, and its very small size makes it particularly suitable for OEM applications where it can be integrated into users' products.

For a quick start with ZestET2-NJ please refer to Appendix A for ZestET2-NJ installation on the ZestET2-NJ Breakout Board and section 9 for software installation and running the examples.

Features

- **Xilinx Artix-7 XC7A35T User FPGA**
- **512MBytes DDR3 memory**
- **105 User I/O pins with I/O voltage from 1.2v to 3.3v**
- **GigEx chip TCP and UDP offload engine**
- **User-programmable SPARC-compatible CPU**
- **User FPGA programmable over Ethernet, JTAG or from Flash**
- **8MBytes User FPGA Flash programmable over Ethernet or JTAG**

- **10/100/1000BASE-T IEEE 802.3 compliant**
- **Support for ARP, IPv4, TCP, UDP, ICMP, IGMP, PTP, HTTP protocols**
- **Multicast UDP support**

- **Simple register based GigEx control interface from User FPGA**
- **Up to 16 simultaneous high speed Ethernet connections**
- **Embedded web server with 256KBytes of Flash for user web pages**

- **Ethernet IEEE1588 PTP version 2 synchronisation**
- **Synchronous Ethernet (SyncE) clock recovery**
- **Precision timed trigger output generation**
- **Precision timestamping of event inputs**
- **1 Pulse Per Second (1PPS) output to User FPGA**

- **Single 3.3V module supply voltage**
- **Form factor 40x50mm (ZestET2-NJ)**
- **Industrial temperature**
- **Integrated MDI interface termination resistors minimising external components**
- **Ethernet link and activity LED drivers**

A block diagram of the ZestET2-NJ module is shown in Figure 1.

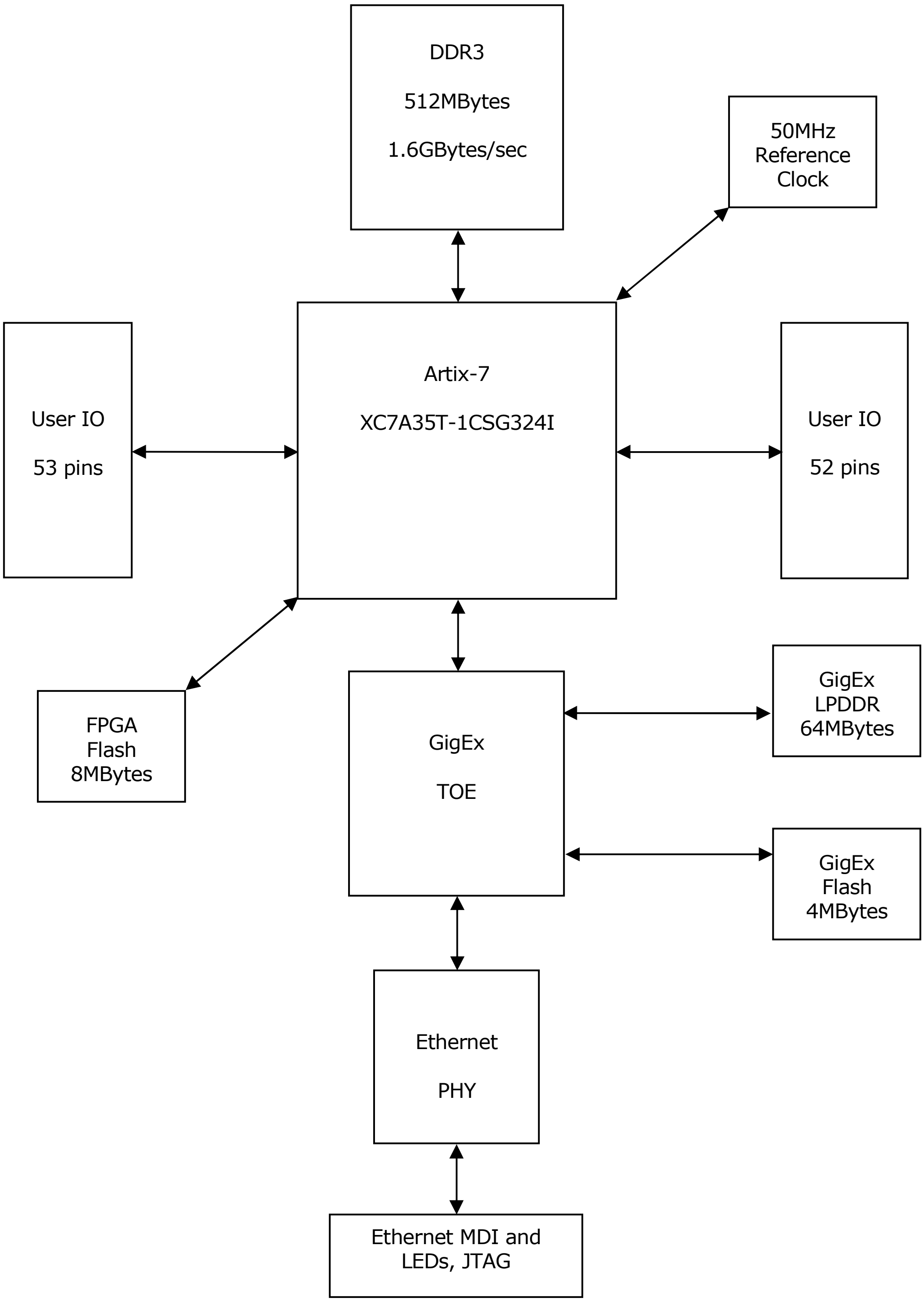


Figure 1. ZestET2-NJ Block Diagram

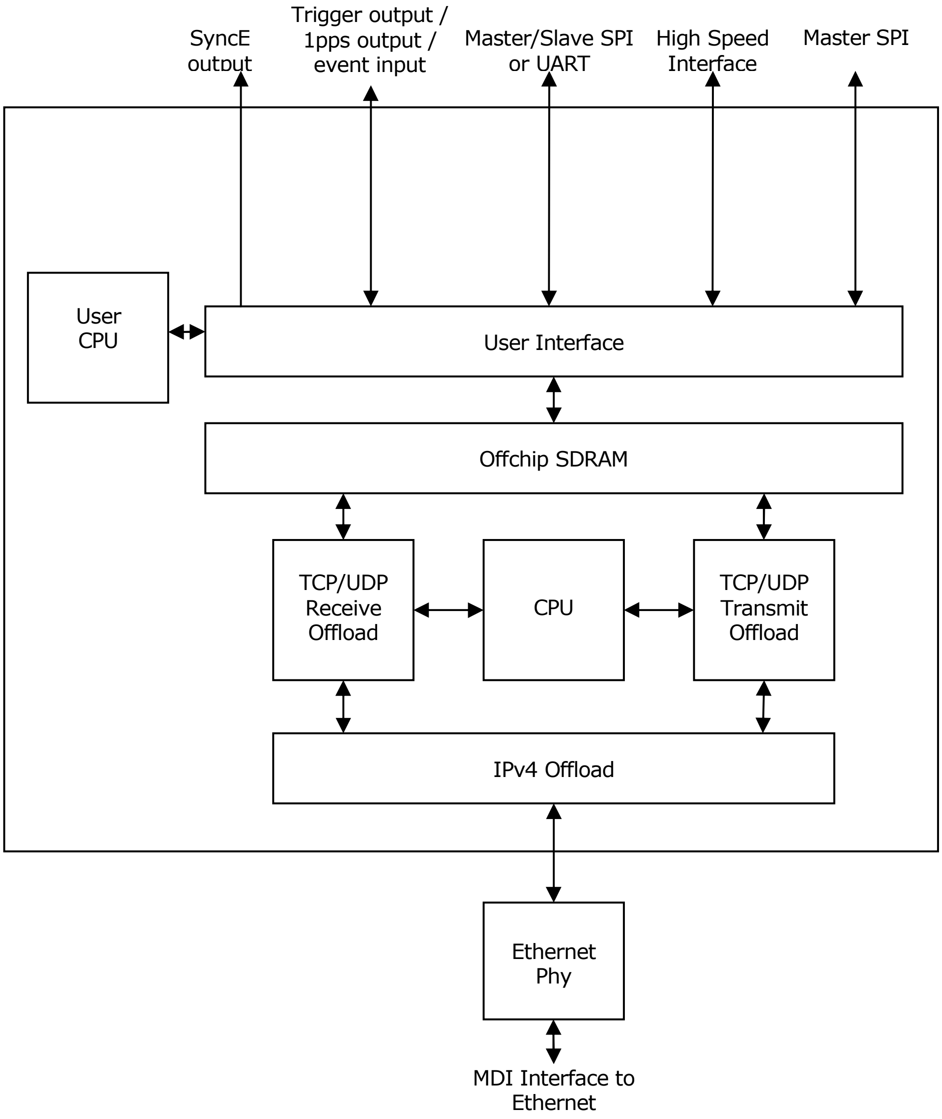


Figure 2. ZestET2 GigExpedite Block Diagram

2 Overview

The proprietary GigEx TOE chip contains all the functionality for connecting the Artix-7 FPGA to Ethernet. The interface between the FPGA and GigEx is a simple high speed bus that can be configured via an internal web server to run in one of four modes. The FPGA sets up the required number of Ethernet channels (up to 16) and then simply writes or reads each channel as a streaming interface. GigEx handles the entire Ethernet protocol. It can run at up to Gigabit Ethernet speed.

There are also low speed interfaces between GigEx and the FPGA – Master SPI controlled by GigEx (used to configure the FPGA on ZestET2), and Slave SPI or UART controlled by the FPGA. The Synchronous Ethernet clock output provides a 125MHz clock synchronised to all the other compatible devices on the network. The IEEE1588 PTP (Precision Time Protocol) trigger output and event input provide a way of synchronising devices over the network.

On the network side, ZestET2-NJ connects to the Ethernet magnetics on the user's board. These can be a discrete chip or integrated into the RJ45 network jack.

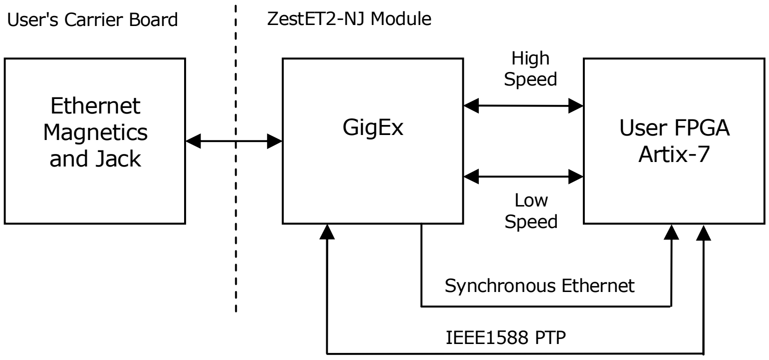


Figure 3. ZestET2-NJ System Overview

3 Signal Description

3.1 Pin Assignments

The ZestET2 module has three IO connectors. J1 and J2 are for the 105 User FPGA IO pins. On ZestET2-NJ, J3 is for the Ethernet magnetics MDI, Ethernet LED's and User FPGA JTAG. J1 has 53 User FPGA IO pins, and J2 has 52 User FPGA IO pins. The pinouts of the three connectors are shown below.
'GND' is ground or 0V
'NC' is no connection and should not be connected to any signal on the carrier board

J1			
VCCO_35	1	2	IO16(1)
IO16(0)	3	4	IO16(2)
GND	5	6	GND
IO35(0)	7	8	IO35(2)
IO35(1)	9	10	IO35(3)
GND	11	12	GND
IO35(4)	13	14	IO35(6)
IO35(5)	15	16	IO35(7)
GND	17	18	GND
IO35(8)	19	20	IO35(10)
IO35(9)	21	22	IO35(11)
GND	23	24	GND
IO35(12)	25	26	IO35(14)
IO35(13)	27	28	IO35(15)
GND	29	30	GND
IO35(16)	31	32	IO35(18)
IO35(17)	33	34	IO35(19)
GND	35	36	GND
IO35(20)	37	38	IO35(22)
IO35(21)	39	40	IO35(23)
GND	41	42	GND
IO35(24)	43	44	IO35(26)
IO35(25)	45	46	IO35(27)
GND	47	48	GND
IO35(28)	49	50	IO35(30)
IO35(29)	51	52	IO35(31)
GND	53	54	GND
IO35(32)	55	56	IO35(34)
IO35(33)	57	58	IO35(35)
GND	59	60	GND
IO35(36)	61	62	IO35(38)
IO35(37)	63	64	IO35(39)
GND	65	66	GND
IO35(40)	67	68	IO35(42)
IO35(41)	69	70	IO35(43)
GND	71	72	GND
IO35(44)	73	74	IO35(46)
IO35(45)	75	76	IO35(47)
GND	77	78	GND
IO35(48)	79	80	IO35(49)

J2			
VCCO_16	1	2	VCCO_15
IO16(3)	3	4	IO16(4)
GND	5	6	GND
IO15(0)	7	8	IO15(2)
IO15(1)	9	10	IO15(3)
GND	11	12	GND
IO15(4)	13	14	IO15(6)
IO15(5)	15	16	IO15(7)
GND	17	18	GND
IO15(8)	19	20	IO15(10)
IO15(9)	21	22	IO15(11)
GND	23	24	GND
IO15(12)	25	26	IO15(14)
IO15(13)	27	28	IO15(15)
GND	29	30	GND
IO15(16)	31	32	IO15(18)
IO15(17)	33	34	IO15(19)
GND	35	36	GND
IO15(20)	37	38	IO15(22)
IO15(21)	39	40	IO15(23)
GND	41	42	GND
IO15(24)	43	44	IO15(26)
IO15(25)	45	46	IO15(27)
GND	47	48	GND
IO15(28)	49	50	IO15(30)
IO15(29)	51	52	IO15(31)
GND	53	54	GND
IO15(32)	55	56	IO15(34)
IO15(33)	57	58	IO15(35)
GND	59	60	GND
IO15(36)	61	62	IO15(38)
IO15(37)	63	64	IO15(39)
GND	65	66	GND
IO15(40)	67	68	IO15(42)
IO15(41)	69	70	IO15(43)
GND	71	72	GND
IO15(44)	73	74	IO15(46)
IO15(45)	75	76	IO15(47)
GND	77	78	GND
IO15(48)	79	80	IO15(49)

J3 - ZestET2-NJ			
User FPGA JTAG TCK	**1**	**2**	GND
GND	**3**	**4**	MDI_P0
User FPGA JTAG TMS	**5**	**6**	MDI_N0
User FPGA JTAG TDI	**7**	**8**	GND
GND	**9**	**10**	MDI_P1
User FPGA JTAG TDO	**11**	**12**	MDI_N1
User FPGA JTAG VREF (1.8V)	**13**	**14**	GND
GND	**15**	**16**	MDI_P2
NC	**17**	**18**	MDI_N2
NC	**19**	**20**	GND
GND	**21**	**22**	MDI_P3
NC	**23**	**24**	MDI_N3
NC	**25**	**26**	GND
NC	**27**	**28**	ETH_LED0
3V3_IN	**29**	**30**	ETH_LED1
3V3_IN	**31**	**32**	NC
3V3_IN	**33**	**34**	GND
3V3_IN	**35**	**36**	GND
3V3_IN	**37**	**38**	GND
3V3_IN	**39**	**40**	GND

3.2 Pin Description

Pin Number	Pin Name	Type	Description
J1 1	VCCO_35	PWR	User FPGA Bank 35 VCCO power in, 1.2 to 3.3V
J1 3,2,4 J2 3,4	IO16(0..4)	I/O	User FPGA Bank 16 I/O. IO standard must be compatible with voltage on VCCO_16
J1	IO35(0..49)	I/O	User FPGA Bank 35 I/O. IO standard must be compatible with voltage on VCCO_35
J2 1	VCCO_16	PWR	User FPGA Bank 16 VCCO power in, 1.2 to 3.3V
J2 2	VCCO_15	PWR	User FPGA Bank 15 VCCO power in, 1.2 to 3.3V
J2	IO15(0..49)	I/O	User FPGA Bank 15 I/O. IO standard must be compatible with voltage on VCCO_15
J3 4 J3 6 J3 10 J3 12 J3 16 J3 18 J3 22 J3 24	MDI_P[0] MDI_N[0] MDI_P[1] MDI_N[1] MDI_P[2] MDI_N[2] MDI_P[3] MDI_N[3]	I/O	ZestET2-NJ - Media Dependent Interface The 4 MDI differential pairs form the Ethernet interface connection. They should be connected to the magnetics on the carrier board as described in section 3.3.2. The PHY device contains an internal 100 ohm termination resistor between P/N in each pair.
J3 28 J3 30	nETH_LED0 nETH_LED1	O	Ethernet activity LEDs The LEDs indicate the presence of an Ethernet link and also activity on the link. The nETH_LEDx signals are active low - i.e. they are 0 when the LED should be lit. The mapping for LED0 and 1 is as follows: nETH_LED1 nETH_LED0 1000Mbps, no activity: Off On 1000Mbps, activity: Off Blink 100Mbps, no activity: On Off 100Mbps, activity: Blink Off 10Mbps, no activity: On On 10Mbps, activity: Blink Blink No link: Off Off
J3 1 J3 5 J3 7 J3 11 J3 13	JTAG TCK JTAG TMS JTAG TDI JTAG TDO JTAG VCC	I I I O O	User FPGA JTAG with 1.8V IO standard JTAG VCC is the 1.8V reference voltage output
J3 29, 31, 33, 35, 37, 39	3V3_IN	PWR	Module power in, 3.3V +/-5%

All pins labelled GND should be connected to Ground. All pins labelled NC must be left unconnected on the carrier board.

3.3 ZestET2-NJ Carrier Board

This section contains information and guidelines for designing the carrier board for ZestET2-NJ.

3.3.1 Mechanical

ZestET2-NJ is a 40 x 50 mm module that plugs flat onto the user's carrier board. Figure 4 shows the module dimensions.

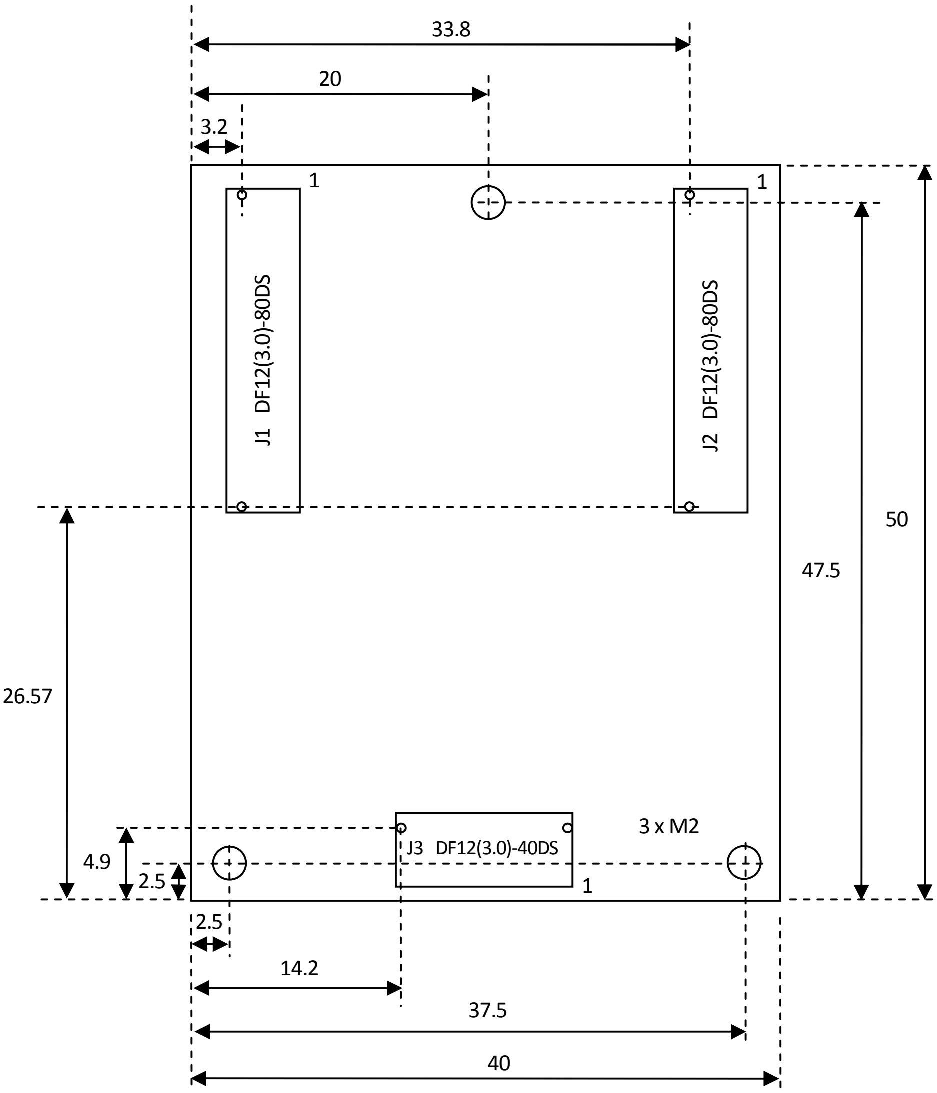


Figure 4. ZestET2-NJ Module Dimensions in mm (View from Top)

The maximum height of the module above the top surface of the carrier board is 7.30 +0.56/-0.36 mm.

There are three module connectors, which are Hirose 80-way and 40-way DF12 0.5mm pitch receptacles with 3mm stacking height when combined with the header on the carrier board. The required headers on the carrier board are:

J1, J2	Hirose DF12(3.0)-80DP-0.5V
J3	Hirose DF12(3.0)-40DP-0.5V

The pin assignments are shown in the tables in section 3.1.

3.3.2 Magnetics and Ethernet Jack

The Marvell 88E1512 PHY is voltage mode and therefore the magnetics should be selected for voltage mode PHY's.

The PHY-side centre taps should be connected individually to signal ground via capacitors (e.g. 10nF) (they should not be connected together to power, which is done with current mode PHY's).

Since Auto MDI/MDIX (auto crossover) is enabled, the magnetics should not have a common mode choke on the PHY side unless 3 coil common mode chokes are used.

Tracks from J3 to the magnetics should be a short as possible (e.g. less than 30mm); should have track lengths matched; and should have 100 +/- 10 ohms differential impedance. Termination components are not required as the PHY has them integrated on-chip.

RJ45 jacks with integrated magnetics may be used. The ZestET2-NJ Breakout board uses the Halo HFJ11-1G41E-L12RL (tab down), alternatively the Halo HFJT1-1G41-L12RL (tab up) can be used, both have no common mode choke on the PHY side. GigEx has also been tested the Wurth 749 911 161 4A (tab up, and 3-core common mode choke on PHY side).

Follow the PCB layout guidelines for the magnetics and/or RJ45 jack that you use.

3.3.3 Ethernet LEDs

ZestET2-NJ has provision for driving two LEDs for Ethernet status, as shown in section 3.2. The signals are active-low and open-drain so require pull-up resistors to a convenient voltage rail (absolute max 30V) to give the required voltage and current for the LEDs on the carrier board, as shown in Figure 5. They are driven by NXP NX3008NBKT FETs. The LEDs can be integrated in the Ethernet jack.

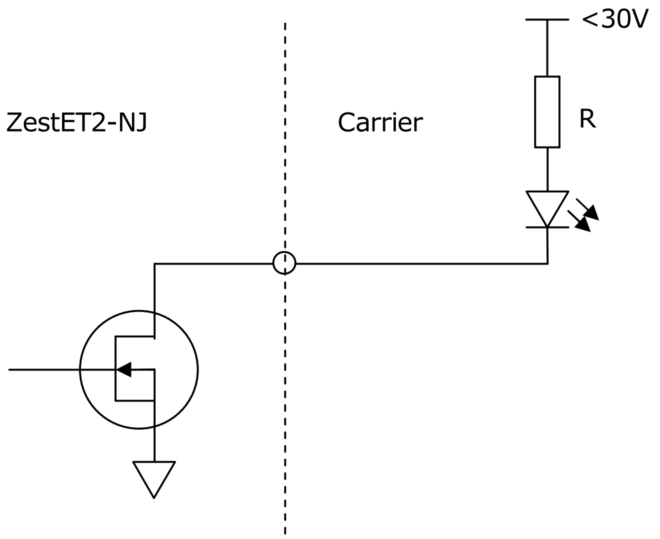


Figure 5. ZestET2-NJ Ethernet LED Circuit

3.3.4 Power

The ZestET2-NJ module is powered from 3.3V on J3. The power supply 3V3_IN should be within 150mV of 3.3V and the maximum sustained current drawn with the User FPGA full of logic has been estimated to be 1.5A, but this does depend on your application. With the User FPGA configured with a simple loopback application transferring data over Ethernet at 100MBytes/sec in both directions, the current is 0.9A. The Xilinx XPower Estimator (XPE) tool or Vivado power analysis features can be used to estimate the extra current required for your application. The User FPGA part number is XC7A35T-1CSG324I.

VCCO_15, 16 and 35 power the User FPGA IO banks 15, 16 and 35 respectively for the User IO signals on J1 and J2. Their IO Standard nominal voltages can be 1.2 to 3.3V +/- 5%. The Xilinx XPower Estimator (XPE) tool or Vivado power analysis features can be used to estimate the current required on each power rail for your application.

Power supply ripple and noise should be less than 50mV peak-to-peak. For both 3V3_IN and VCCO, pi filters (capacitor, ferrite bead, capacitor) may be used to minimise power supply noise entering the module.

3.3.5 Electrical

All the User IO signals on ZestET2-NJ have tracks with 50 ohms +/- 10% single ended impedance and 100 ohms +/-10% differential impedance within the differential pairs. The tracks on the carrier board should also have matching impedance, according to whether they are used as single ended signals or differential pairs.

The track lengths of the User IO signals in each FPGA IO bank are matched to +/- 1mm, and across all IO banks to +/- 2.5mm.

To avoid electrical interference, there should be no components on the carrier board underneath ZestET2-NJ. Trackwork underneath ZestET2-NJ should be minimised, and ideally the area on the top surface of the PCB underneath ZestET2-NJ should be a continuous copper plane with multiple vias to digital ground.

3.3.6 Environmental

All the components on ZestET2-NJ are industrial temperature range -40 to +85 deg C. For most components this is the operating ambient temperature range, but for the GigEx chip and the User FPGA the operating temperature range is specified as -40 to +100 deg C for the internal semiconductor junction of the chip.

For GigEx, in order to keep this junction temperature below 100 deg C, the maximum operating ambient temperature depends on the orientation of the module and whether there is any air flow as follows:

Cooling Arrangement	Maximum Ambient Temperature
Convection cooling - module vertical in open air When the module is horizontal the maximum ambient temperature is reduced by about 3 deg C	60 deg C
Forced air cooling - 500 linear feet per minute	80 deg C

These maximum operating ambient temperatures are estimates based on measurements taken at a normal office temperature. Note that any air flow gives significantly better cooling than convection in still air.

For the User FPGA (part number XC7A35T-1CSG324I) the maximum ambient temperature will depend on the power dissipated by your application. The Xilinx XPower Estimator (XPE) tool or Vivado power analysis features can be used to estimate this. The thermal resistances of the chip package can be found in Xilinx document UG475 7 Series FPGAs Packaging and Pinout.

4 GigEx Interfaces

The GigEx device provides the following interfaces:

- ZestET2-NJ Media Dependent Ethernet Interface which is the connection to the Ethernet network. This interface connects directly to the Ethernet magnetics and RJ45 connector on the user's carrier board.
- High Speed Data Interface which can operate in a number of different modes for convenient interfacing to the FPGA. Using this interface the FPGA can read and write the control registers in the GigEx device and transmit and receive data to and from the network. The High Speed Data Interface can operate as a simple 8 or 16 bit SRAM interface or in FIFO and Direct modes.
- SPI Master Interface. The SPI master interface is used to configure the FPGA over Ethernet.
- SPI Slave Interface. The SPI slave interface provides an alternative way of accessing the control registers in the GigEx device. In some modes of operation, the high speed interface is dedicated to data transmission and reception over the network and the control registers can only be accessed by the slave SPI interface or the UART.
- UART Interface. The low speed interface can alternatively be configured as a standard UART at various baud rates. When in this mode, ASCII commands can be sent to the GigEx device to control the network connections and query status. When using this mode, the high speed interface can be dedicated to transmitting and receiving data on the network.
- Synchronous Ethernet Clock. This output provides a 125MHz clock which will be synchronised with all the other SyncE clocks on compatible devices on the same network.
- IEEE1588 PTP Trigger/Event port. This port is used to generate pulses synchronised to other devices on the network or to capture the time of an event in the attached hardware. The single pin can be configured as a precisely timed trigger output, a 1 pulse per second output or an event capture input to the GigEx.
- GPIO pins. GigEx can configure some of its IO pins as general purpose inputs and outputs. When not in use as part of one of the other interfaces these pins can be used to set and read pins on the FPGA via the network.

4.1 ZestET2-NJ Media Dependent Ethernet Interface

The Media Dependent Ethernet Interface consists of 4 differential pairs that connect to the Ethernet magnetics. The PHY (Marvell 88E1512) is a voltage mode PHY and the magnetics must be compatible with voltage mode PHYs - see 3.3.2 for recommendations. The PHY contains the 100 ohm differential termination resistors between P and N in each pair.

There are also two signals for driving LEDs to indicate Ethernet status. They are active low open drain signals driven by NXP NX3008NBKT FETs.

Signal Name	Type	Description
MDI_P[0] MDI_N[0] MDI_P[1] MDI_N[1] MDI_P[2] MDI_N[2] MDI_P[3] MDI_N[3]	I/O	Media Dependent Interface The 4 MDI differential pairs form the Ethernet interface connection. They should be connected to the magnetics on the carrier board as described in section 3.3.2.
nETH_LED0	O	Ethernet activity LEDs

<table>
<tr><td>nETH_LED1</td><td></td><td>The LEDs indicate the presence of an Ethernet link and also activity on the link. The nETH_LEDx signals are active low - i.e. they are 0 when the LED should be lit. The mapping is as follows:
<table>
<tr><th></th><th>nETH_LED1</th><th>nETH_LED0</th></tr>
<tr><td>1000Mbps, no activity</td><td>Off</td><td>On</td></tr>
<tr><td>1000Mbps, activity</td><td>Off</td><td>Blink</td></tr>
<tr><td>100Mbps, no activity</td><td>On</td><td>Off</td></tr>
<tr><td>100Mbps, activity</td><td>Blink</td><td>Off</td></tr>
<tr><td>10Mbps, no activity</td><td>On</td><td>On</td></tr>
<tr><td>10Mbps, activity</td><td>Blink</td><td>Blink</td></tr>
<tr><td>No link</td><td>Off</td><td>Off</td></tr>
</table>
</td></tr>
</table>

4.2 High Speed Interface

The High Speed interface consists of a synchronous set of pins that can be configured to operate in a number of different modes to simplify connection of an external device. The interface can be used to access the control registers in the GigEx and to transmit and receive data on the network.

The four modes of operation are:

- 16 Bit SRAM interface. In this mode, the interface is configured to look like conventional SRAM allowing simple connection to a processor bus. The interface is comprised of a 16 bit bidirectional data bus, a 10 bit address bus and control strobes. The data is pipelined in a similar manner to ZBT SRAM. Data can be transferred across the network on 16 high speed channels.

- 8 Bit SRAM interface. In this mode, the interface is configured to look like two separate 8 bit SRAM ports - one for writing to the GigEx and one for reading from the GigEx. Each port consists of an 8 bit unidirectional data bus, a 10 bit address bus and control strobes. This mode allows simultaneous data transmission in both directions. Data can be transferred across the network on 16 high speed channels.

- FIFO interface. In this mode, the interface is configured to look like separate transmit and receive FIFOs. Each port consists of an 8 bit unidirectional data bus, a 3 bit address bus, 8 FIFO full flags and a control strobe. The 3 bit address and 8 full flags allow data transfer on up to 8 network channels with the remaining 8 channels available for data transfer on the low speed interface. The GigEx acts as a master for receive data, writing to the attached user device and as a slave for transmit data where the external user device writes to GigEx. In this mode, the high speed interface is for data transfer only and the control registers must be accessed either via the SPI slave interface or via the UART interface.

- Direct interface. In this mode, 32 general purpose IO pins are used to transfer raw data to/from the GigEx. The 32 bits can be configured to be inputs or outputs in groups of 8 allowing flexible bidirectional transfers to/from a 'dumb' device attached to the GigEx. In this mode, the high speed interface is for data transfer only and the control registers must be accessed either via the SPI slave interface or via the UART interface.

4.2.1 16 Bit SRAM Interface

The 16 bit SRAM interface is a single data rate interface running at up to 125MHz with the external device as master of the bus and the GigEx as the slave. The bus protocol is a simple read/write protocol similar to ZBT SRAM. ZBT (Zero Bus Turnaround) means that no bus cycles are lost between writes and reads or between reads and writes. Waveforms for read, write and read/write turnaround are shown below in Figure 6.

The latency between the control strobes / address bus and the data bus is 2 cycles for both read data and write data. Common latencies for reads and writes means that no idle cycles are needed on the data bus when turning round between reads and writes but it does require that the external device delays write data relative to the write strobes and address.

The 16 Bit SRAM Interface consists of the following signals:

Signal Name	Type	Description
CLK	I/O	High Speed Interface Clock All other signals on the high speed bus are synchronous to this clock. The clock can be driven from the external user device into the GigEx in which case the frequency can be between 5MHz and 125MHz. Alternatively, GigEx can generate a fixed 125MHz clock out to the connected user device. For correct functionality, there must always be a clock present on this pin. Therefore, if the connected user device does not generate a clock on this pin then GigEx must be set to generate a clock out to the user device.
nINT	O	High Speed Interface Interrupt Active low output from the GigEx which will be set when various events occur. The interrupt conditions can be enabled and distinguished with the register map described in section 5.
DQ[15..0]	I/O	Bidirectional 16 bit data bus. When transferring data, DQ[15..8] is the first byte on the network for both writes and reads.
A[9..0]	I	10 bit register byte address. Since this is a 16 bit interface, A0 should always be 0.
nCS	I	Active low chip select strobe.
nWE	I	Active low write strobe.
nBE[1..0]	I	Active low byte write enable strobes
EOF[1..0]	O	End of frame data qualifier. This is generated by the GigEx on data read from the network. EOF1 is active when DQ[15..8] forms the last byte of the frame. EOF0 is active when DQ[7..0] forms the last byte of the frame. Data transmitted to the network does not need the end of frame marked.
HEAD	O	UDP header data qualifier. This is generated by the GigEx on data read from the network when the data is part of the UDP header. Data transmitted to the network does not need the header marked.
LENGTH	O	Frame length qualifier. This is generated by the GigEx on data read from the network when the data is the 16 bit frame length. Data transmitted to the network does not need the length marked.

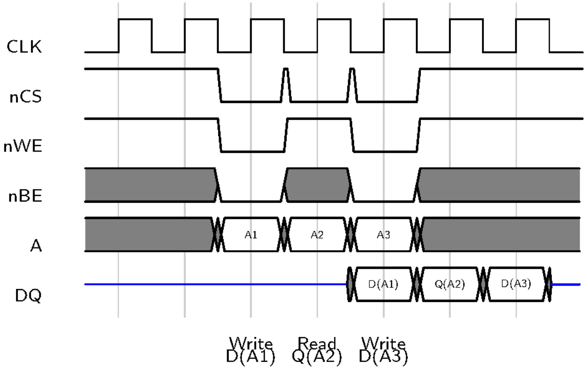


Figure 6. 16 bit SRAM interface read and write operations

The EOF, HEAD and LENGTH signals are valid when reading data from the network. They are used to qualify the type of data being read. Frames of data received from the network are prefixed by the length of the frame and (in the case of UDP frames) a frame header. The LENGTH signal indicates that read data is the frame length. The frame length is 16 bits with most significant byte in DQ[15..8]. The HEAD signal indicates that the read data is part of the UDP header. See section 4.2.3 for the description of the UDP header. The first word on the interface is header bytes 0 and 1 with Byte Offset 0 in DQ[15..8]. The EOF signal pair indicates that the read data contains the last byte of frame data either in the top byte (EOF[1] is set) or the bottom byte (EOF[0] is set). Figure 7 shows the frame read process for a UDP frame with an 8 byte payload. Note that received TCP frames are prefixed by a length but do not have a header. The read process for a TCP frame with an 8 byte payload is shown in Figure 8**.**

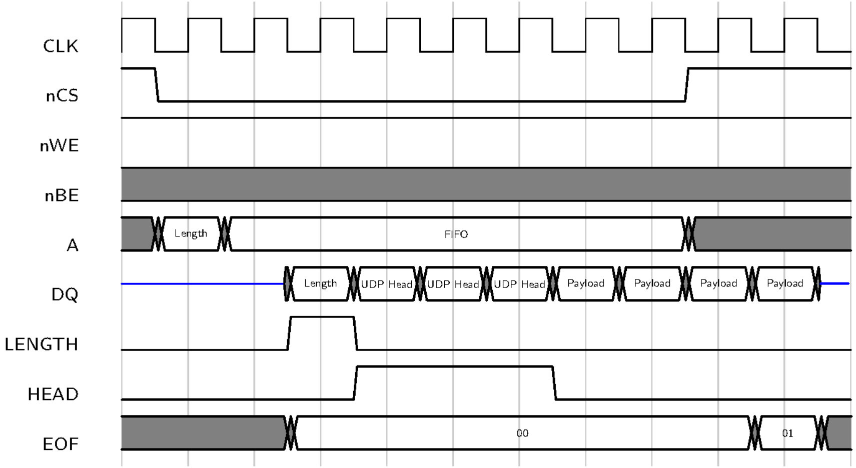


Figure 7. Signal activity during UDP frame reads from the network using 16 bit SRAM interface

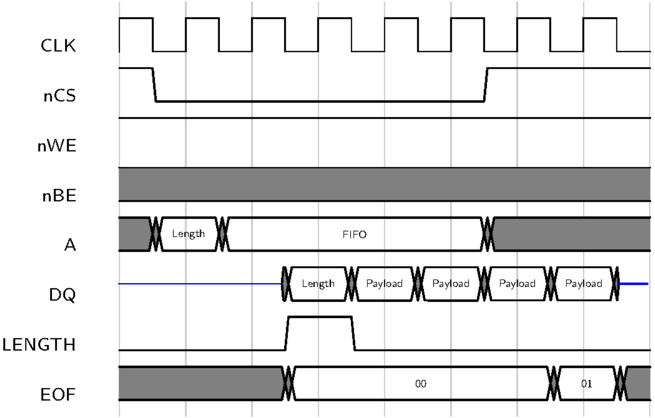


Figure 8. Signal activity during TCP frame reads from the network using 16 bit SRAM interface

4.2.2 8 Bit SRAM Interface

The 8 bit SRAM interface is a pair of single data rate interfaces running at up to 125MHz with the external device acting as master of the bus and the GigEx acting as slave. The interface is a pair of 8 bit interfaces - one read port and one write port. Waveforms for read, write and read/write turnaround are shown in Figure 9.

The latency between the control/address and data during reads is 2 cycles. Write data must be presented to the GigEx in the same clock cycle as the address and control strobes.

The 8 Bit SRAM Interface consists of the following signals:

Signal Name	Type	Description
CLK	I/O	High Speed Interface Clock All other signals on the high speed bus are synchronous to this clock. The clock can be driven from the external user device into the GigEx in which case the frequency can be between 5MHz and 125MHz. Alternatively, GigEx can generate a fixed 125MHz clock out to the connected user device. For correct functionality, there must always be a clock present on this pin. Therefore, if the connected user device does not generate a clock the GigEx must be set to generate a clock out to the user device.
nINT	O	High Speed Interface Interrupt Active low output from the GigEx which will be set when various events occur. The interrupt conditions can be enabled and distinguished with the register map described in section 5.
D[7..0]	I	8 bit write data port.
Q[7..0]	O	8 bit read data port.
DA[9..0]	I	10 bit write byte address.
QA[9..0]	I	10 bit read byte address.
nWE	I	Active low write strobe.
nRE	I	Active low read strobe.

EOF	O	End of frame data qualifier. This is generated by the GigEx on data read from the network when the data is the last byte of the frame. Data transmitted to the network does not need the end of frame marked.
HEAD	O	UDP header data qualifier. This is generated by the GigEx on data read from the network when the data forms part of the UDP header. Data transmitted to the network does not need the header marked.

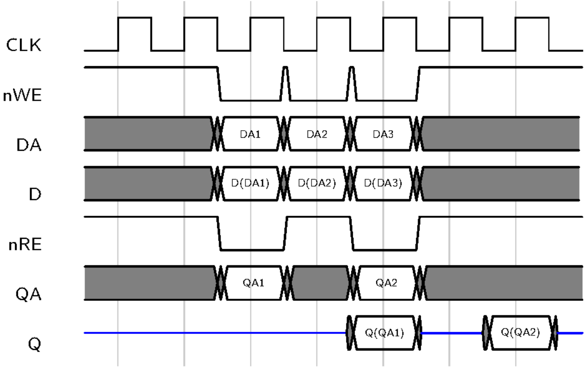


Figure 9. 8 bit SRAM interface read and write operations

The EOF and HEAD signals are valid when reading data from the network. They are used to qualify the type of data being read. Frames of data received from the network are prefixed by the length of the frame and (in the case of UDP frames) a frame header. The frame length is 16 bits and sent most significant byte first. The HEAD signal indicates that read data is part of the UDP header. The EOF signal indicates that the read data contains the last byte of data in the frame. Figure 10 shows the frame read process for a UDP frame with a 3 byte payload. See section 4.2.3 for the description of the UDP header. Note that received TCP frames are prefixed by a length but do not have a header. Figure 11 shows the frame read process for a 9 byte TCP frame.

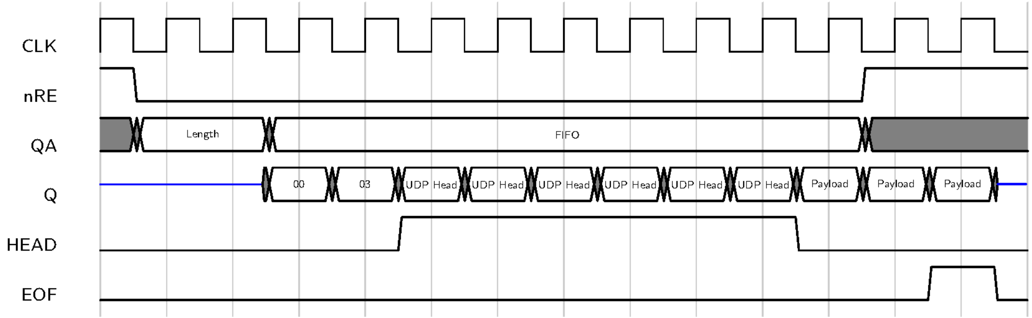


Figure 10. Signal activity during UDP frame reads from the network using 8 bit SRAM interface

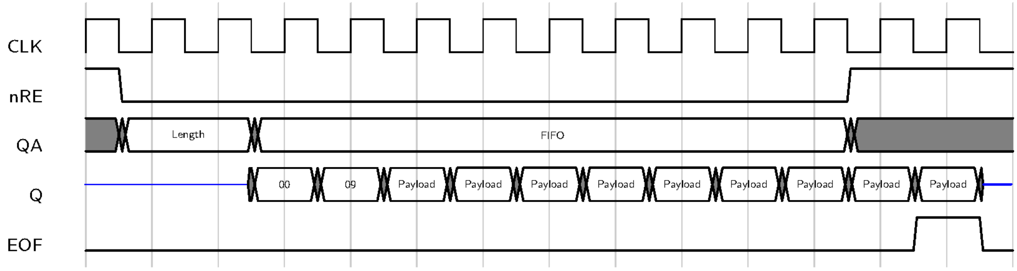


Figure 11. Signal activity during TCP frame reads from the network using 8 bit SRAM interface

4.2.3 FIFO Interface

The FIFO interface is a pair of single data rate interfaces - one read and one write - running at up to 125MHz. Each interface allows access to one of eight channels of data and data flow is regulated using a full flag for each channel. When transmitting data on the network, the external device should check the state of the active low nTF signal for the appropriate channel before writing to the FIFO with the address, data and nTX strobe. When receiving data from the network, the GigEx will check the state of the appropriate nRF flag before setting the channel address and asserting the active low nRx strobe along with the data.

Note that there is a 2 cycle latency associated with the nRF and nTF flags. When receiving data, the GigEx may assert the nRx strobe for up to two cycles after the external device asserts the nRF flag. Likewise, the external device is free to assert the nTx strobe for up to 2 cycles following the GigEx asserting the nTF flag. The external device should treat the nRF flag as an 'almost full' flag and be prepared to accept 2 extra bytes of data following its assertion.

The FIFO interface does not allow access to the GigEx register map. Control of the network connections must take place using the SPI slave interface (section 4.3.1), UART interface (section 4.3.2) or the auto-connection mechanism controlled by the web server interface (section 6.1.4).

The FIFO interface allows access to channels 0 to 7 of the GigEx. Channels 8 to 15 cannot be used with the FIFO interface but remain available for data transfers using the low speed SPI slave or UART interfaces.

Figure 12 shows the signals during FIFO read and write accesses for UDP frames. Note that for UDP frames received from the network, the length of the datagram is received before the UDP header and finally the payload. For UDP frames transmitted to the network, the requested datagram length is sent to the network first followed by the payload. The frame lengths are 16 bits and sent most significant byte first. The UDP header received from the network is as follows, with byte offset 0 corresponding to Header0 in Figure 12:

Byte Offset	Description
0	Remote transmitting IP address most significant byte (bits [31..24])
1	Remote transmitting IP address bits [23..16]
2	Remote transmitting IP address bits [15..8]
3	Remote transmitting IP address least significant byte (bits [7..0])
4	Remote transmitting port most significant byte

5	Remote transmitting port least significant byte

Figure 13 shows TCP transfers over the FIFO interface. Note that no frame length or header is required for TCP transfers. All data written to the FIFO interface is sent over the network and only payload data received from the network is written out of the FIFO interface to the user device.

The FIFO interface consists of the following signals:

Signal Name	Type	Description
CLK	I/O	High Speed Interface Clock All other signals on the high speed bus are synchronous to this clock. The clock can be driven from the external user device into the GigEx in which case the frequency can be between 5MHz and 125Mhz. Alternatively, GigEx can generate a fixed 125MHz clock out to the connected user device. For correct functionality, there must always be a clock present on this pin. Therefore, if the connected user device does not generate a clock the GigEx must be set to generate a clock out to the user device.
nINT	O	High Speed Interface Interrupt Active low output from the GigEx which will be set when various events occur. The interrupt conditions can be enabled and distinguished with the register map described in section 5.
D[7..0]	I	8 bit write data port.
Q[7..0]	O	8 bit read data port.
TC[2..0]	I	3 bit transmit channel address.
RC[2..0]	O	3 bit receive channel address.
nTx	I	Active low transmit data strobe.
nRx	O	Active low receive data strobe.
nTF[7..0]	O	8 active low transmit full flags to inhibit data transfer from the external device
nRF[7..0]	I	8 active low receive full flags to inhibit data transfer to the external device

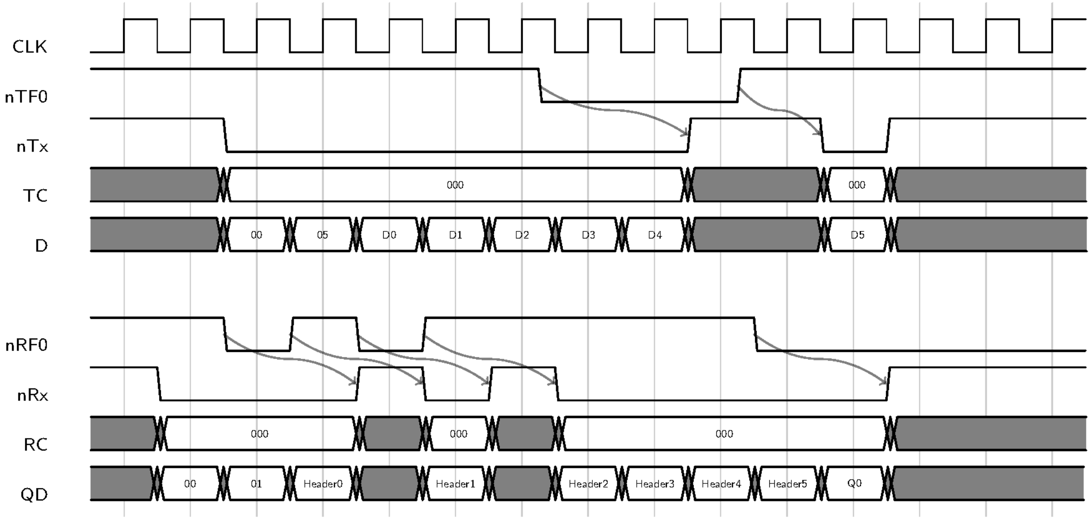


Figure 12. FIFO interface read and write operations for UDP frames

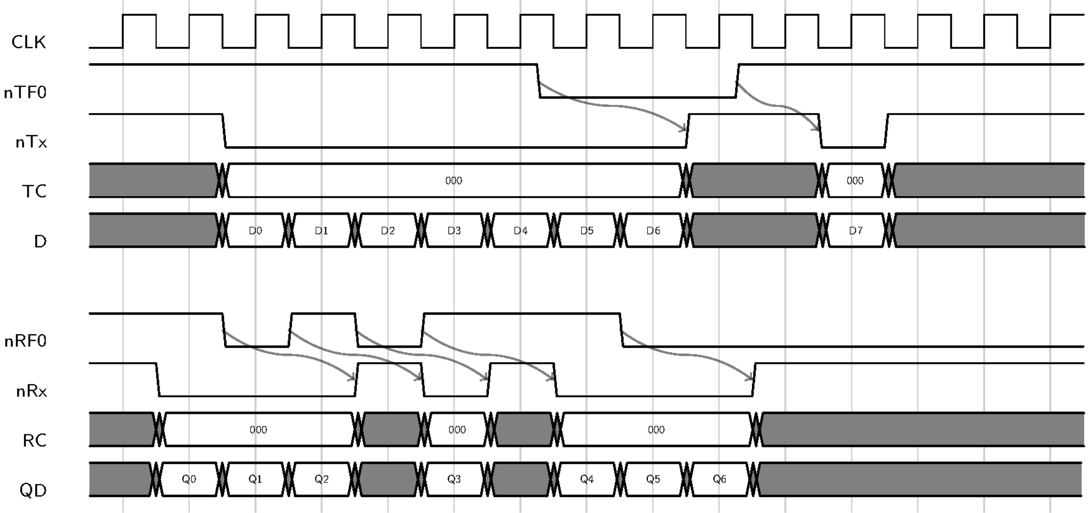


Figure 13. FIFO interface read and write operations for TCP frames

4.2.4 Direct Interface

The direct interface is a set of 32 IO pins whose states can be directly transmitted to or set by the network interface. It allows 'dumb' devices to be connected to the network and be controlled by a remote client.

The 32 pins can be assigned as inputs and outputs in groups of 8. i.e. there can be 0, 8, 16, 24 or 32 inputs and 0, 8, 16, 24 or 32 outputs subject to a total limit of 32 pins. For example, the pins can be assigned as 0 input and 32 outputs, 16 input and 16 outputs, 8 inputs and 16 inputs or any other similar combination. Outputs from the GigEx are on pins B[(m-1)..0] where m is the number of output pins and inputs to the GigEx are on pins B[31..(32-n)] where n is the number of input pins. For example, for 8 outputs from and 16 inputs to GigEx, the outputs are on pins B[7..0] and the inputs are on pins B[31..16].

The direct interface also has a number of control strobes associated with it which can be used to hold off data transfer or to detect FIFO underflow and FIFO overflow conditions where the network is running too fast or too slow for the user interface.

If the network is unable to supply data fast enough to update the pins on every clock cycle then the pins' state will be preserved until further data is available from the network. If the network is unable to transmit data fast enough from the pins then data will be lost. All data pins are updated synchronously with the user interface clock.

The direct interface does not allow access to the GigEx register map. Control of the network connections must take place using the SPI slave interface (section 4.3.1), UART interface (section 4.3.2) or the auto-connection mechanism controlled by the web server interface (section 6.1.4).

The direct interface only allows access to channel 0 of the GigEx and only allows data to be transferred using the TCP protocol. This is because UDP requires a mechanism for transferring the UDP datagram lengths and remote IP address and port number which is not supported by the direct interface. Channels

1 to 15 can only be accessed by the low speed SPI slave or UART interfaces when the high speed interface is in Direct mode.

The direct interface consists of the following signals:

Signal Name	Type	Description
CLK	I/O	High Speed Interface Clock All other signals on the high speed bus are synchronous to this clock. The clock can be driven from the external user device into the GigEx in which case the frequency can be between 5MHz and 125MHz. Alternatively, GigEx can generate a fixed 125MHz clock out to the connected user device. For correct functionality, there must always be a clock present on this pin. Therefore, if the connected user device does not generate a clock the GigEx must be set to generate a clock out to the user device.
nINT	O	High Speed Interface Interrupt Active low output from the GigEx which will be set when various events occur. The interrupt conditions can be enabled and distinguished with the register map described in section 5.
B[31..0]	I/O	32 general purpose input or output pins.
nRx	O	Active low strobe indicating that output data on the B pins is valid.
nRxEmpty	O	Active low flag indicating that the receive FIFO has become empty because the network is unable to supply data fast enough to service the output pins. The previous value of the B pins will be preserved until data becomes available from the network.
nRxFull	I	Active low flag to hold off data transfers from the GigEx. Note that there is a latency of 2 cycles associated with this flag so data transfers will continue for 2 cycles after this flag is asserted. This pin can be held high for continuous data reception.
nTx	I	Active low transmit data strobe. Data on the B input pins will be sent to the network on cycles when this strobe is low. This pin can be held low to continuously transmit data to the network.
nTxFull	O	Active low transmit FIFO full flag. This flag will be asserted when the transmit FIFO fills due to the network being unable to keep up with the transmission rate.

Example data transfers using the Direct interface are shown below. In this example, 8 bits are used to receive data from the network and 16 bits are used to transmit data to the network. Figure 14 shows normal behaviour when data is received and transmitted without interruptions. Figure 15 shows the case where the external device is unable to receive data as fast as it is received from the network. The external device asserts nRxFull which causes GigEx to pause data reception. Reception will be resumed when the external device de-asserts nRxFull. If the network is unable to transmit data as fast as the external device provides it then GigEx will assert nTxFull at which point the external device must de-assert nTx to prevent data loss. Figure 16 shows the case where the network is unable to supply data fast enough to service the receive data pins. When no data from the network is available, nRxEmpty will be asserted by GigEx and the value on the B output pins will be preserved.

Since the B pin value is preserved when data is not available from the network, the value of the pins can be changed slowly by sending data to GigEx only when the pins' state needs to be changed. Similarly, the data rate to the network can be restricted by only asserting nTx when the value of the B pins needs to be sent to the remote device.

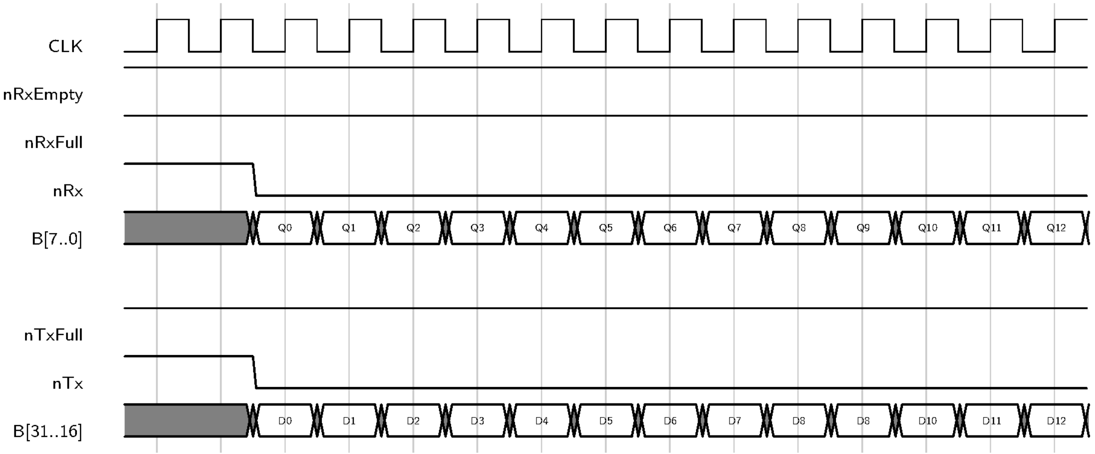


Figure 14. Direct interface showing normal behaviour

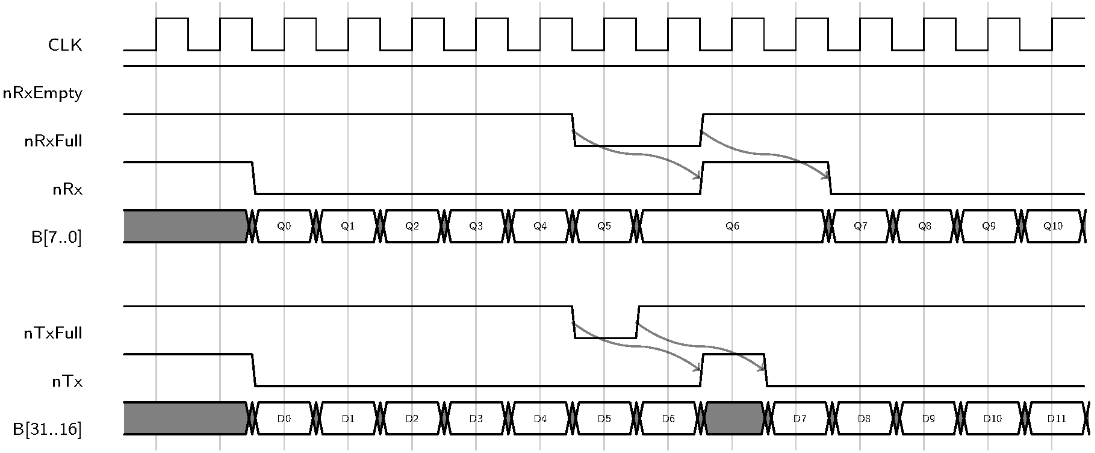


Figure 15. Direct interface showing behaviour when receive and transmit FIFOs fill

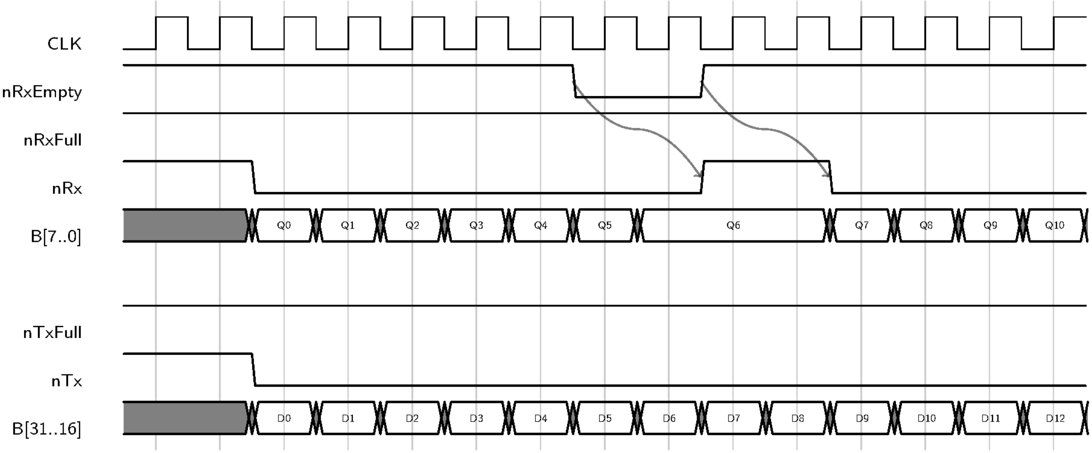


Figure 16. Direct interface showing behaviour when receive FIFO empties

4.3 Low Speed Interface

4.3.1 SPI Master/Slave Interface

The SPI master/slave interface can be configured to control an external device (master mode) or can be used as an alternative method for reading and writing the GigEx registers (slave mode). In slave mode, it is effectively a serial version of the 8-bit or 16-bit interface, and so can also write and read channels to transfer data over the network. When using the 8 or 16 bit SRAM interface mode the SPI slave can be used in addition to the SRAM interface. When using the FIFO or Direct interface modes the slave SPI and UART interfaces provide the only method of accessing the GigEx control registers.

The ZestET2 board uses the Master/Slave port as a SPI master to access the user flash via Ethernet and to configure the FPGA. After FPGA configuration, the SPI port is switched into slave mode so that the FPGA can access the GigEx registers.

The SPI slave interface allows 16 bit access to the register map when the main interface is in 16 bit SRAM mode. When the main interface is in any other mode the SPI slave interface allows 8 bit access to the register map.

Note that simultaneous access to the same register from both the SRAM interface and the SPI slave interface will result in undefined behaviour.

The SPI slave interface consists of the following signals. These are shared with the UART interface so only one of the SPI master/slave and UART can be active at any time.

Signal Name	Type	Description
SPIs_nCS	I	Active low slave SPI chip select.
SPIs_SCK	I	Slave SPI clock signal.
SPIs_MISO	O	Slave SPI Master In, Slave Out signal for data transfer from the GigEx.
SPIs_MOSI	I	Slave SPI Master Out, Slave In signal for data transfer to the GigEx.

The Slave SPI access patterns are shown below. Each access starts with a 4 bit command which is 1100 for a write and 0111 for a read. The command is followed by a 2 bit field which contains the active high byte enable bits for a 16 bit write and should be 11 for all other accesses. The next field is the 10 bit register byte address. For 16 bit accesses, the lowest address bit (A0) must be 0. For write accesses, the address is followed immediately by 8 or 16 bits of data. For read accesses, 8 dummy bits are inserted before the GigEx generates 8 or 16 bit of data.

Example accesses are shown in Figure 17 to Figure 20.

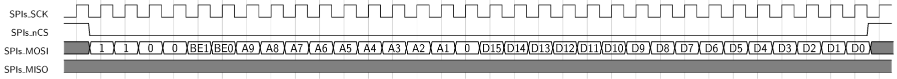

SPIs.SCK
SPIs.nCS
SPIs.MOSI
SPIs.MISO
1 1 0 0 BE1 BE0 A9 A8 A7 A6 A5 A4 A3 A2 A1 0 D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

Figure 17. 16 Bit Slave SPI Write Access

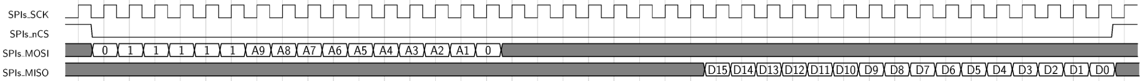

SPIs.SCK
SPIs.nCS
SPIs.MOSI
SPIs.MISO
0 1 1 1 1 1 A9 A8 A7 A6 A5 A4 A3 A2 A1 0
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

Figure 18. 16 Bit Slave SPI Read Access

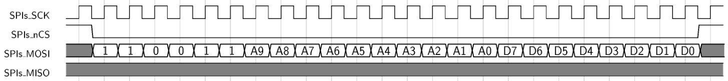

SPIs.SCK
SPIs.nCS
SPIs.MOSI
SPIs.MISO
1 1 0 0 1 1 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 D7 D6 D5 D4 D3 D2 D1 D0

Figure 19. 8 Bit Slave SPI Write Access

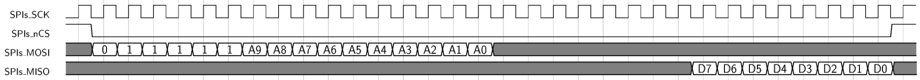


Figure 20. 8 Bit Slave SPI Read Access

4.3.2 UART

The UART interface provides an ASCII based interface for sending commands to the GigEx. These commands can simply read and write the registers as an alternative to the high speed or SPI slave interfaces or they can also be more complex commands to establish network connections, update flash settings and so on. It is also possible to transmit and receive data on a network connection through the UART to allow, for example, simple telnet connections.

Like the SPI Slave Interface, the UART Interface is effectively a serial version of the 8-bit and 16-bit interfaces. However the UART command set includes higher level commands than writing and reading registers, e.g. for setting up channels and transferring data over channels.

The UART interface consists of the following signals. These are shared with the SPI slave interface so only one of the SPI slave and UART can be active at any time.

Signal Name	Type	Description
TX	O	Signal for data from the GigEx to the external device

RX	I	Signal for data from the external device to the GigEx
CTS	I	Active low 'Clear to Send' handshaking signal. When active, the UART is able to transmit data on the TX pin. When inactive, data transmission on TX will be paused.
RTS	O	Active low 'Ready to Send' handshaking signal. When active, the UART is able to accept data on the RX pin. When inactive, data transfer on the RX pin should be paused by the external device.

The UART uses standard asynchronous protocols to transmit characters. The web server or UART command processor allow the baud rate to be changed between 9600 and 921600 baud, the number of data bits and parity can be set and the hardware flow control signals can be enabled and disabled. Refer to sections 6.3.1 and 6.5 for details of these settings.

Note that the UART is a TTL interface running at the same voltage as the main register interface (1.8V). It can therefore be driven directly by the User FPGA.

The UART commands are described in detail in section 6.3.

4.4 SPI Master Interface

The SPI master interface provides a method for controlling an external slave device from the network. For example, the firmware for an attached processor can be downloaded from the network without interaction from the processor.

The SPI master interface is controlled by command messages sent to the network control port of the GigEx. See section 6.4 for details of these messages.

The SPI master mode pins can also be switched into GPIO mode. On the ZestET2 board, the SPI Master interface is used as GPIO to control the FPGA configuration pins so the SPI master interface cannot be used.

The SPI master interface consists of the following signals.

Signal Name	Type	Description
SPIm_nCS	O	Active low slave device chip select.
SPIm_SCK	O	Slave SPI clock signal.
SPIm_MISO	I	Slave SPI Master In, Slave Out signal for data transfer to the GigEx.
SPIm_MOSI	O	Slave SPI Master Out, Slave In signal for data transfer from the GigEx.

4.5 IEEE1588 PTP Timing Interface

GigEx supports the IEEE1588 Precision Time Protocol mechanism for time synchronisation across devices in the network with three modes of operation. Firstly, a hardware event can be detected and precisely timestamped. Secondly, a hardware signal can be generated at a predetermined time and thirdly a 1 pulse per second signal can be generated synchronised to a master 1 second timer elsewhere on the network. There is a single pin to provide these three operations so only one may be selected at any time.

Refer to section 6.3.1 for details of how to configure the trigger/event pin from the UART or section 6.5 for details of configuring it from the web server.

GigEx contains an internal clock which is synchronised to a master clock on the network using the Precision Time Protocol. All other devices on the network which support PTP will have a similar internal clock so events and pulses will be synchronised across multiple devices on the network.

For synchronisation to take place, a master PTP clock device must be present on the network as GigEx will only function as a PTP slave device.

4.6 SyncE Interface

GigEx supports the Synchronous Ethernet (SyncE) standard to recover a 125MHz clock from the connected network and output it on the SyncE pin. The SyncE physical layer protocol ensures that this clock is synchronised to the most stable clock source in the network. All other devices connected to the network which support this standard will also recover a clock which will be synchronised to the same stable clock source, and therefore will be synchronised to each other including the ZestET2.

4.7 GPIO pins

GigEx can switch 24 of its pins into a GPIO mode. In this mode each pin can be set as an input or as an output with a specified value. When a pin is not needed for the High speed or Low speed interfaces it can be used as a direct signal between the FPGA and the host computer. The state of the GPIO pins can be written to or read from over Ethernet or can be controlled using the built in web server in GigEx.

The GPIO pin number mapping is shown in section 8.4.

4.8 ZestET2 Interface Usage

The previous sections have described the interfaces available on the GigEx device. Some of these interfaces are used for specific purposes on the ZestET2 board as described below, and are not available to the user. All the interface connections are shown in Figure 21 below.

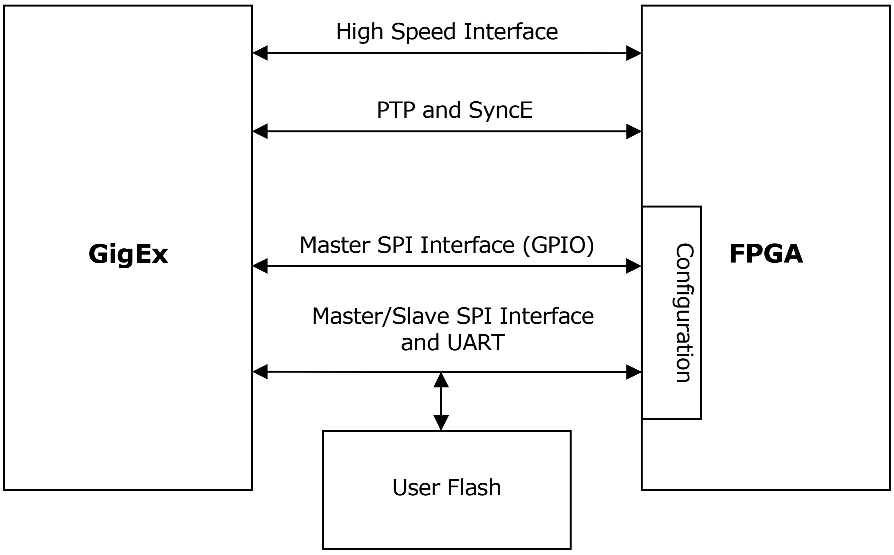


Figure 21. Use of the GigEx interfaces on ZestET2

The ZestET2 board uses the Master/Slave SPI and Master SPI interfaces to configure the FPGA and access the user flash via Ethernet. The Master SPI port is switched into GPIO mode and the 4 pins are used to access the nPROGRAM, nINIT, DONE and configuration mode pins on the FPGA. Control of these pins is handled by the support software.

The Master/Slave SPI port is used as a master to configure the FPGA using slave serial mode and to read and write the user flash from Ethernet. After configuration, the Master/Slave port is switched into either SPI slave or UART mode so the FPGA can access the user flash and the slave SPI or UART ports on the GigEx to access the control registers.

Note that when the GigEx low speed interface is in UART mode the user flash is not accessible either for FPGA configuration or from the GigEx or FPGA interfaces.

To avoid contention on the SPI bus, the FPGA must be deconfigured to allow GigEx access to the flash. The host software will automatically clear the contents of the FPGA if it needs to access the flash via Ethernet.

On power up, if the low speed interface is configured as slave SPI (i.e. not UART) then the FPGA will automatically configure from the user flash if a suitable bitstream is present. A bitstream can be uploaded to the flash over Ethernet using the ZestET2 CPanel application from the installation CD, or over JTAG using the Xilinx Vivado tools and a Xilinx JTAG pod.

5 GigEx Control Registers

The operation of the GigEx device is controlled by a set of registers which can be accessed from the high speed interface (when in 16 or 8 bit SRAM modes) or from the SPI slave interface or via the UART. The external user device is responsible for avoiding conflicts when accessing the registers from multiple interfaces simultaneously.

The register address space is 10 bits wide accessed with A[9..0] byte address. Registers are in groups where the group address is formed by A9 to A5. The address within the group is formed by A4 to A0. When accessing registers in 16 bit SRAM mode either over the high or low speed interface, A0 must be 0. Data is then presented on 16 data lines. For all other modes, accesses are 8 bits with the upper 8 bits of the register (bits 15:8) accessed when A0 is 0 and the lower 8 bits of the register (bits 7:0) accessed when A0 is 1. For example:

	16 bit control register	
	Bits 15:8	**Bits 7:0**
8 bit register access	Address A9:A0 = xxxxxxxxx0	Address A9:A0 = xxxxxxxxx1
16 bit register access	Address A9:A0 = xxxxxxxxx0	

5.1 Register Map

The address map for all registers is shown in the table below. Addresses not listed in the table are undefined and should not be accessed.

Group 0-15 Registers: Network connection management

These registers are accessed when A9 is 0. A8 to A5 form the network channel number from 0 to 15 inclusive.

A4:A1	Read/Write	Description
0	Write Only	Local port register
1		Connection IP address register (high 16 bits)
	Read Only	TCP Server Mode: Connected remote IP address
	Write Only	TCP Client Mode: Target remote IP address
	Write Only	UDP Receive Mode: Remote IP address filter
	Write Only	UDP Transmit Mode: Target remote IP address
	Write Only	Multicasting send: Target remote IP address
	Write Only	Multicasting receive: Local IP address filter
2		Connection IP address register (low 16 bits)
	Read Only	TCP Server Mode: Connected remote IP address
	Write Only	TCP Client Mode: Target remote IP address
	Write Only	UDP Receive Mode: Remote IP address filter
	Write Only	UDP Transmit Mode: Target remote IP address
	Write Only	Multicasting send: Target remote IP address
	Write Only	Multicasting receive: Local IP address filter
3		Remote port register
	Read Only	TCP Server Mode: Connected remote port
	Write Only	TCP Client Mode: Target remote port
	Write Only	UDP Receive Mode: Remote port filter
	Write Only	UDP Transmit Mode: Target remote port

	Write Only Write Only	Multicasting send: Target remote port Multicasting receive: Local IP address filter
4	Write Only	Payload size and time to live register Bits 15-8: Maximum payload size for UDP send in 32 byte units Bits 7-0: Time to live
5	R/W	Interrupt enable (write) and status (read) Bit 15-4: Reserved. Must be set to 0. Bit 3: State change interrupt Bit 2: To network buffer not full interrupt Bit 1: To network buffer empty interrupt Bit 0: From network data available interrupt
6	Write only Write only Write only R/W	Connection control and status Bit 15-7: Reserved. Must be set to 0. Bit 6: 1 – Pause transmission, 0 – Enable transmission Bit 5: 1 - Enable channel, 0 - Reset channel Bit 4: 1 - TCP connection, 0 - UDP connection Bit 3-0: Request state change (write), Current state (read) States are: 0: CLOSED 1: LISTEN (TCP server mode) 2: CONNECT (TCP client mode) 3: ESTABLISHED
7	R/W	From network frame length (read) To network datagram length (write) (UDP only)
8	R/W	Data to/from network connection FIFO

Group 16 Registers: Global network settings

Group 16 registers are registers related to the global network settings of the GigEx. They are accessed when A9:A5 are 0x10 (i.e. from addresses 0x200 to 0x21F).

A4:A1	Read/Write	Description
0	R/W	Local IP address (high 16 bits)
1	R/W	Local IP address (low 16 bits)
2	R/W	Local subnet mask (high 16 bits)
3	R/W	Local subnet mask (low 16 bits)
4	R/W	DCHP/AutoIP settings Bit 0: Enable AutoIP Bit 1: Enable DHCP
6	R/W	Gateway IP address (high 16 bits)
7	R/W	Gateway IP address (low 16 bits)
8	R/W	Jumbo frame maximum length in bytes
9	R/W	GigEx network control local port number
10	R/W	GigEx web server local port number
11	R/W	Update network settings register Writes: Bit 0: Write 1 to this bit to initiate update of network settings Reads: Bit 0: When read as 1 the network settings are being updated Bit 1: When read as 1 the update failed

15	R	Link status register Bit 4: Link Status (1 - connected, 0 - not connected) Bit 3: Duplex (1 - half duplex, 0 - full duplex) Bit 2: 1Gbps Bit 1: 100Mbps Bit 0: 10Mbps

Group 17 Registers: PTP/IEEE1588 control and status registers

Group 17 registers are registers related to the timing settings of the GigEx. They are accessed when A9:A5 are 0x11 (i.e. from addresses 0x220 to 0x23F).

A4:A1	Read/Write	Description
0	R/W	PTP time register Write to this address to latch the current PTP time. Read from this address to shift out next bits of the latched PTP time.
1	R/W	Event time register Write to this address to latch the latest event time. Read from this address to shift out next bits of the latched event time.
2	W	Trigger/1PPS pulse width in units of 100ns
5	R/W	Trigger/Event control and status Writes: Bit 7: Event interrupt enable Bit 0: Trigger output control - 0 for single shot, 1 for one pulse per second Reads: Bit 7: Event interrupt status. Read clears interrupt condition.
6	W	Trigger time most significant word (Bits 63:48)
7	W	Trigger time (Bits 47:32)
8	W	Trigger time (Bits 31:16)
9	W	Trigger time least significant word (Bits 15:0)

Group 18 Registers: Device information registers

Group 18 registers are registers providing information about the GigEx device. They are accessed when A9:A5 are 0x12 (i.e. from addresses 0x240 to 0x25F).

A4:A1	Read/Write	Description
0	R	GigEx hardware version Read from this register to find the version of GigEx hardware fitted to the board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312. When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.
1	R	GigEx software version Read from this register to find the version of GigEx software running on the

		board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312. When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.

Group 23 Registers: General Purpose User Comms Control and Status Registers

Group 23 registers are registers related to the general purpose communication mechanisms of the GigEx. They are accessed when A9:A5 are 0x17 (i.e. from addresses 0x2E0 to 0x2FF).

A4:A1	Read/Write	Description
0	R/W	User settings flash control/status register Writes: Bit 0: Write 1 to this bit to initiate copy of user settings registers to flash Reads: Bit 0: When read as 1 the copy to flash is in progress
5	R/W	Mailbox interrupt enable/status register Writes: Bit 6: Write 1 to enable mailbox interrupts Reads: Bit 6: Mailbox interrupt status. Read clears interrupt condition.
6	W	Mailbox to user CPU register. Writing a 1 to bit 0 will cause an interrupt on the user CPU. This can be used to inform the user CPU that the external device has written to the general purpose user comms registers and the CPU should take some action.

Groups 24-31 Registers: General Purpose User Comms

Groups 24-31 registers form a bank of 256 bytes of general purpose registers for passing values from the GigEx to the external user device and a second set of 256 bytes for passing values from the external device to the GigEx. The two sets of registers are at the same range of addresses. They are accessed when A9:A8 are 0x3 (i.e. from addresses 0x300 to 0x3FF).

The registers from the GigEx to the external device can be preserved in flash memory for boot time settings. They can also be set by a control message from the network (see section 6.4) or from the web server (see section 6.5) allowing simple settings to be configured by web pages or setup information to be passed from a remote device on the network. The external device can read these registers through the high or low speed interfaces.

The registers from the external device to GigEx can be written by the high or low speed interfaces. They can be read using a control message from a remote device on the network (see section 6.4) or by the GigEx web server (see section 6.5) allowing status results to be reported from the external device to a web page or remote client.

5.2 Register Descriptions

Each 16 bit control register is described in detail below. In the address values, 'cccc' indicates the 4 bit channel number. A0 is marked a 'b' for byte select and should be 0 for 16 bit accesses (i.e. when the interface is configured as 16 bit SRAM mode). For 8 bit accesses it should be 0 to access the top 8 bits of data and 1 to access the bottom 8 bits of data.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	0	0	0	b	**Local port**

The local port register specifies the local TCP or UDP port number for a connection. This port number is used as the source port for UDP and TCP data from the GigEx device and is used to filter data received by the GigEx device from the network. Only data received by the GigEx device with a destination port that matches the value in this register is passed through the GigEx device on this connection.

If a source port value of 0 is set then GigEx will select an unused port number to use for both transmit and receive. This is only useful for TCP client applications or for UDP transmitting applications where the source port of packets does not need to be known by the local user application.

The Local Port should only be changed when the channel state is inactive. It is not possible to change the Local Port when there is data in the GigEx channel buffer and any attempt to do so will result in undefined behaviour. It is however possible to change the Local Port of one channel when any of the other 16 channels are active.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	0	0	1	b	**Connection IP Address High 16 bits**
0	c	c	c	c	0	0	1	0	b	**Connection IP Address Low 16 bits**

The connection IP address is used to store remote IP addresses on a per-channel basis. The upper 16 bits are stored in one register and the lower 16 bits in a second register.

When the GigEx device is acting as a TCP server (i.e. listening for connections from a remote client), the Connection IP Address is read only and is set to the IP address of the remote client by the GigEx device when a connection is made.

When the GigEx device is acting as a TCP client (i.e. initiating connections to a remote server), the Connection IP Address is write only and is the IP address of the remote server.

When acting as a UDP receiver, the connection IP address will filter data from the network. Only data received by the GigEx device with a source address matching the Connection IP address will pass through the GigEx device. Setting the Connection IP address to 0.0.0.0 will allow all data with a source port matching the Remote Port register and a destination port matching the Local Port register through. Setting the Connection IP Address to 0.0.0.0. and the Remote Port to 0000 is permissible and allows all data matching the Local Port register through.

When acting as a UDP transmitter, the connection IP address will form the destination IP address for data sent by the GigEx device. Note that this must be a valid IP address and cannot be 0.0.0.0 as described above. Therefore, it is not possible to receive data from all IP addresses and transmit to a specified address using a single connection. To achieve this functionality, two of the 16 channels must be used.

When acting as a multicast receiver (i.e. listening for data on a multicast address), the connection IP address is write only and is the multicast address to listen on.

When acting as a multicast transmitter (i.e. sending data to a multicast address), the connection IP address is write only and is the multicast address to which data is sent.

The Connection IP address should only be changed when the channel is disabled. It is not possible to change the Connection IP address when there is data in the GigEx channel buffer and any attempt to do so will result in undefined behaviour. It is however possible to change the Connection IP address of one channel when any of the other 16 channels are active.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	0	1	1	b	**Remote port**

The remote port register is used to store remote TCP or UDP ports on a per-channel basis.

When the GigEx device is acting as a TCP server (i.e. listening for connections from a remote client), the Remote Port is read only and is set to the TCP port of the remote client by the GigEx device when a connection is made.

When the GigEx device is active as a TCP client (i.e. initiating connections to a remote server), the Remote Port is write only and is the TCP port of the remote server.

When acting as a UDP receiver, the remote port will filter data from the network. Only data received by the GigEx device with a source port matching the value in the Remote Port register will pass through the GigEx device. Setting the Remote Port to 0000 will allow all data with a source IP address matching the Connection IP Address register and a destination port matching the Local Port register setting through. Setting the Remote Port to 0000 and the Connection IP Address to 0.0.0.0. is permissible and allows all data matching the Local Port register through.

When acting as a UDP transmitter, the remote port will form the destination port for data sent by the GigEx device. Note that this must be a valid UDP port and cannot be 0000 as described above. Therefore, it is not possible to receive data from all ports and transmit to a specified port using a single connection. To achieve this functionality, two of the 16 channels must be used.

When acting as a multicast receiver (i.e. listening for data on a multicast address), the remote port behaviour is the same as a normal UDP receiver.

When acting as a multicast transmitter (i.e. sending data to a multicast address), the remote port behaviour is the same as a normal UDP transmitter.

The Remote Port should only be changed when the channel is disabled. It is not possible to change the Remote Port when there is data in the GigEx channel buffer and any attempt to do so will result in undefined behaviour. It is however possible to change the Remote Port on one channel when any of the other 16 channels are active.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	1	0	0	b	**Payload Size and Time To Live**

The payload size and time to live register is split into two 8 bit fields:
Bits 15 to 8: Payload size, in multiples of 32 bytes
Bits 7 to 0: Time to live, in number of hops

The Payload size field is used during UDP transmission to control IP fragmentation of datagrams. UDP datagrams can be up to 64kbytes in length which means they may have to be split across multiple frames on the network using IP fragmentation. The Payload size field is used to control the length of the fragments when IP fragmentation is required.

The value of this field will depend on the network infrastructure and receiver capabilities. The maximum safe value for most networks is 1500 bytes (i.e. set the payload size register to 46 for 1472 bytes). Some Gigabit devices support Jumbo frames which are larger than this value. The maximum value the GigEx device supports is 7488 bytes (i.e. a value of 234 in the Payload Size register).

The Time to Live (TTL) field is used during UDP and TCP transmission to limit the number of device hops that a transmitted frame will make. This is used to confine traffic to a limited section of a large network and is particularly important in multicast transmissions to prevent data 'leaking' beyond the boundaries of a relevant network section. For non-multicast transmissions, a 'normal' value for the TTL is 128.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	1	0	1	b	**Interrupt Enable and Status**

Each channel is able to generate 4 different interrupts depending on data received/transmitted and connection state changes. When any of the 4 interrupts from any of the 16 connections is active and enabled the nINT pin is asserted. These interrupts can be individually enabled and disabled for a channel. Writing to the register modifies the write only interrupt enable register. Reading from the register retrieves the read only interrupt status register.

Writing to the Interrupt Enable register enables (bit set to 1) or disables (bit set to 0) an interrupt.

Reading from the Interrupt Status register reports the state of each interrupt. The state can be active even if the enable bit is not set but in this case the nINT pin is inactive.

The interrupts are as follows:

Bit 15-4: Reserved. Must be set to 0 when writing.
Bit 3: Connection state change
Bit 2: Data buffer to network not full
Bit 1: Data buffer to network empty
Bit 0: Data buffer from network not empty

The 'Connection state change' interrupt is set when the state of the connection changes. This may be because a remote device has connected or disconnected or because a connection has successfully been initiated to a remote device. A read of the interrupt status register has the side effect of clearing this interrupt.

The 'Data buffer to network not full' interrupt is set to indicate that there is space in the outgoing buffer for this channel and more data can be written to the channel. When active, at least 64k bytes of space is available in the buffer allowing a burst of up to this length to be written without re-checking the status bit. This interrupt will remain active until the data buffer is full.

The 'Data buffer to network empty' interrupt is set when the outgoing buffer for this channel is empty. This is used to determine when all data has been sent to the network indicating it is safe to close the connection without losing data. This interrupt will remain active until the data buffer is not empty.

The 'Data buffer from network not empty' interrupt is set when new data has been received from the network on a channel. On receiving the interrupt, the frame length should be read from the frame length register. It is then safe to read the complete frame from the data FIFO.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	1	1	0	b	**Connection Control and Status**

The connection control register (write only) is used to control the state of the connection. The connection status register (read only) is used to report state changes for a connection.

The control/status register is organised as follows:
Bit 15-7: Reserved. Must be set to 0 when writing.
Bit 6: Pause transmission
Bit 5: Channel enable
Bit 4: TCP (value of 1) or UDP (value of 0) channel
Bit 3-0: Connection status

The 'pause transmission' bit can be set to 1 to temporarily suspend transmission on a channel. This may be useful to allow large amounts of data to be written to an outgoing channel without it being transmitted. When the pause bit is set back to 0 the GigEx will begin transmitting the buffered data in large packets rather than multiple small packets.

The ‘channel enable’ bit should be set to 1 when a channel is active. When set to 0 the channel is disabled and all associated incoming and outgoing buffers are flushed. It is important to check that the outgoing buffer is empty using the buffer empty interrupt status before disabling a channel to avoid loss of data.

When the channel is disabled, the connection state should be set to CLOSED. If a channel is CLOSED but not disabled then the data FIFOs will not be flushed and the data in the buffers will be available for transfers once the channel is re-connected (i.e. the state switches back to ESTABLISHED).

The ‘TCP or UDP channel’ bit should be set to 1 for a TCP connection or 0 for a UDP connection.

The following states and values of bits 3-0 are defined (other values are undefined and should not be used):

0: CLOSED
1: LISTEN (TCP only)
2: CONNECT (TCP only)
3: ESTABLISHED

A request to change the state is initiated by writing to the ‘Connection Control’ register. The GigEx device will act on this request and will report any state changes through the ‘Connection Status’ register. The ‘Connection Status’ register may also change at the request of the remote device (e.g. if the remote device closes the connection). The change of reported state is accompanied by a ‘Connection state change’ interrupt.

The normal flow for a TCP server application is as follows:

Connection Control (to GigEx)		**Connection Status (from GigEx)**
CLOSED		CLOSED
LISTEN		
		LISTEN
	Remote device connects	
		ESTABLISHED
ESTABLISHED		
	Transfer data to/from network	
	……	
	Remote device disconnects	
		CLOSED
CLOSED		

Note that the user application must respond to a connection status of ESTABLISHED by writing ESTABLISHED back to the connection control register. This acknowledges receipt of the status change. Similarly, the user application should respond to the state change to CLOSED by writing CLOSED back to the control register.

The normal flow for a TCP client application is as follows:

Connection Control (to GigEx)		**Connection Status (from GigEx)**
CLOSED		CLOSED
CONNECT		
		CONNECT
	Connection established	
		ESTABLISHED
ESTABLISHED		
	Transfer data to/from network	
	……	
CLOSED		
	Connection disconnected	
		CLOSED

Note that the user application must respond to a connection status of ESTABLISHED by writing ESTABLISHED back to the connection control register. This acknowledges receipt of the status change.

The normal flow for a UDP application is as follows:

Connection Control (to GigEx)		**Connection Status (from GigEx)**
CLOSED		CLOSED
ESTABLISHED		
		ESTABLISHED
	Transfer data to/from network	
	……	
CLOSED		
		CLOSED

Note that the user application must wait for the GigEx to respond with an ESTABLISHED status before data can be transmitted.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	0	1	1	1	b	**Frame length**

The frame length register reports the length of frames received by the GigEx device from the network for a connection. It must be read only once per received frame and only when the 'Data buffer from network not empty' interrupt status is set and before the data itself is read.

When transmitting UDP data, the frame length register is used to tell the GigEx device how long the next datagram to the network is and must be written only when the 'Data buffer to network not full' interrupt status is set and immediately before the data itself is written to the GigEx 'Data FIFO' register. ***The frame length register must not be written during TCP transfers***. The transmitted frame length for TCP connections is determined by the TCP protocol.

The flow for receiving TCP data from the GigEx device is:

Receive 'Data buffer from network not empty' interrupt
Read frame length register
Read x bytes from the associated 'Data FIFO' register where x is the value read from the frame length register

The flow for transmitting data on a TCP connection is:

Receive 'Data buffer to network not full' interrupt
Write up to 64kBytes to the associated connection FIFO register

The flow for receiving UDP data from the GigEx device is:

Receive 'Data buffer from network not empty' interrupt
Read frame length register
Read UDP header (i.e. remote IP address high and low words and remote port)
Read x bytes from the associated connection FIFO register where x is the value read from the frame length register

The first 6 bytes of data read on each received UDP datagram can be used to determine the origin of the datagram. On the 16 bit SRAM interface these bytes will be presented in the first 3 reads from the data FIFO register as follows:

Header offset (bytes)	Data
0	Remote IP address (high 16 bits)
2	Remote IP address (low 16 bits)
4	Remote port

For the 8 bit SRAM and FIFO interfaces these bytes will be presented in the first 6 reads from the data FIFO:

Header offset (bytes)	Data
0	Remote IP address (most significant byte, bits 31:24)
1	Remote IP address (bits 23:16)
2	Remote IP address (bits 15:8)
3	Remote IP address (least significant byte, bits 7:0)
4	Remote port most significant byte
5	Remote port least significant byte

Note that these 6 bytes are not included in the value read from the frame length register (i.e. the total length of the read will be 6+frame length register bytes).

The flow for transmitting data on a UDP connection is:

Receive 'Data buffer to network not full' interrupt
Write the next datagram length to the frame length register
Write x bytes to the associated connection FIFO register where x is the value written to the frame length register

Note that for UDP connections, the bytes written to the connection FIFO register immediately after the frame length register are always the first bytes in the transmitted datagram. This allows higher level protocols such as RTP to be used where the higher level protocol header must be the first bytes in the UDP datagram.

For UDP, datagrams can be up to 64kbytes long. If the datagram length set by the frame length register is longer than the 'Payload Size' register setting then the GigEx device will implement IP fragmentation when transmitting the datagram. For TCP, the datagram sizes are determined by the TCP protocol and IP fragmentation is never used.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
0	c	c	c	c	1	0	0	0	b	**Data FIFO**

Each channel has a data FIFO register for reading data received from the network by the GigEx device and for writing data to the GigEx device for transmission on the network. These FIFO registers can be accessed at any time allowing for interleaving of data between multiple channels on a word-by-word basis.

When sending an odd number of bytes with the 16 bit SRAM interface mode, the byte enable signal nBE must be used to mask bytes.

When in 16 bit SRAM mode, if the frame length is odd then the last read from the data FIFO will only contain 1 byte of data. The last read will contain valid data in DQ15-DQ8 and undefined data in DQ7-DQ0. Data received first from the network will be presented on DQ15-DQ8. Data received second will be in bits DQ7-DQ0.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	0	0	0	0	b	**Local IP Address High 16 bits**
1	0	0	0	0	0	0	0	1	b	**Local IP Address Low 16 bits**

The local IP address register reports the current setting of the GigEx IP address. The IP address can be determined by the GigEx device using any of the following methods:

1. Fixed IP address stored in non-volatile memory
2. DHCP client. Requires a DHCP server to be present on the network
3. AutoIP

The method used can be set through the UART interface, by using a control port message or via the web server. See sections 6.3, 6.4 and 6.5. It can also be set with the DHCP/AutoIP enable register (see below).

After power up, no accesses to the GigEx device should be attempted until the IP address has changed from 0.0.0.0.

The local IP address can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	0	0	1	0	b	**Local Subnet Mask High 16 bits**
1	0	0	0	0	0	0	1	1	b	**Local Subnet Mask Low 16 bits**

The local subnet mask register reports the current setting of the GigEx subnet mask. The subnet mask can be determined by the GigEx device using any of the following methods:

1. Fixed mask stored in non-volatile memory
2. DHCP client. Requires a DHCP server to be present on the network
3. AutoIP

The method used can be set through the UART interface, by using a control port message or via the web server. See sections 6.3, 6.4and 6.5. It can also be set with the DHCP/AutoIP enable register (see below).

The subnet mask can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	0	1	0	0	b	**DHCP/AutoIP Enable Register**

DHCP and AutoIP are two alternative methods for acquiring an IP address on a network. They both allow a device to automatically assign their own IP settings. DHCP requires a DHCP server to be present on the network whereas AutoIP is a self contained protocol requiring no additional equipment. GigEx supports both protocols along with the use of a fixed IP address stored in flash memory. Please refer to your network administrator to get the correct settings for this register.

The DHCP/AutoIP register can be read to determine which of the protocols in currently enabled. The following bits will read as 1 when the protocol is enabled or 0 if disabled.

Bit 1: DHCP Enable bit
Bit 0: AutoIP Enable bit

The enable bits can be changed by writing to this register before writing to the Update Network Settings register (see below). They can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	0	1	1	0	b	**Gateway IP Address High 16 bits**
1	0	0	0	0	0	1	1	1	b	**Gateway IP Address Low 16 bits**

The gateway IP address register reports the current setting of the GigEx gateway. The gateway can be determined by the GigEx device using any of the following methods:

1. Fixed mask stored in non-volatile memory
2. DHCP client. Requires a DHCP server to be present on the network
3. AutoIP

The method used can be set through the UART interface, by using a control port message or via the web server. See sections 6.3, 6.4 and 6.5. It can also be set with the DHCP/AutoIP enable register (see below).

The gateway IP address can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	1	0	0	0	b	**Jumbo Frame Limit Register**

The standard upper limit for frames on a network is 1500 bytes. Some gigabit Ethernet devices including GigEx support longer, or 'Jumbo', frames. The Jumbo frame limit register can be used to configure the maximum length of frame that GigEx will generate on the network. Since not all devices support Jumbo frames, this register should be set to the lowest value supported by any devices that GigEx will communicate with (including network infrastructure devices such as switches).

The Jumbo frame setting is expressed in bytes between 1500 and 7500 and must be a multiple of 4 bytes.

Reading the Jumbo frame register will return the current setting. The value can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	1	0	0	1	b	**Network Control Port Register**

GigEx opens a TCP port which can be used to pass control message to the device. These control messages are detailed in section 6.4 below. The Network Control Port register specifies which port number is opened for these messages. The default port number is 20480.

Reading the register will return the current setting. The value can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	1	0	1	0	b	**Web Server HTTP Port Register**

The GigEx web server normally operates by opening TCP port 80 for listening. This is the standard HTTP port number but it can be changed with the HTTP port register.

Reading the register will return the current setting. The value can be changed by writing to this register before writing to the Update Network Settings register (see below). It can also be set with the UART, control port message or web server interfaces. See sections 6.3, 6.4 and 6.5.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	1	0	1	1	b	**Update Network Settings Register**

The Update Network Settings register can be used to write new network settings to flash memory on the GigEx. The new settings should be written to the following registers:

Local IP Address High and Low registers
Local Subnet Mask High and Low registers
Gateway IP Address High and Low registers
DHCP/AutoIP Enable register
Jumbo Frame Limit register
Network Control Port register

Web Server HTTP Port register

Once these registers have been written, writing a 1 to bit 0 of the Update Network Settings Register will initiate a write of these settings to the flash. Reading the Update Network Settings Register will return a 1 in bit 0 when the flash write is in progress and a 0 when the write is complete. A 1 in bit 1 of the register when read will indicate that the write to flash failed. This will be because some of the settings in the network settings registers are incorrect or invalid.

When the network settings have been changed the GigEx will re-initialise. This will close any open connections so the update process should only be attempted when no network connections are active.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	0	1	1	1	1	b	**Link Status**

The link status register is a read only register used to report the network connection status from the PHY component. The individual bits are arranged as follows:

Bit 4: Link Status. This bit is 1 when the GigEx device has successfully connected and synchronised to a physical link. It is 0 if a valid link cannot be detected.

Bit 3: Duplex. This bit is set to 1 when the GigEx device is connected to a half duplex link and 0 when connected to a full duplex link.

The link speed is reported in the lower 3 bits as follows with the relevant bit being set to 1 and the others 0:

Bit 2: 1Gbps
Bit 1: 100Mbps
Bit 0: 10Mbps

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	1	0	0	0	0	b	**PTP Time Register**

GigEx maintains an internal free running timer which can be read by the external device through the PTP Time Register. If a PTP master clock is present on the network then GigEx will synchronise its internal time to the master clock to give a precise time of day and date reading. This time will be synchronised across all PTP devices on the network. If a PTP master clock is not present then the PTP Time Register will reflect the time elapsed since the GigEx booted.

The PTP Time Register is 64 bits long. The upper 32 bits are elapsed seconds and the bottom 32 bits are the fraction of seconds expressed in nanoseconds. When synchronised to a PTP master clock, the IEEE1588 epoch is used such that a time reading of 0 seconds equates to midnight on January 1st 1970.

Bits 63:32	Bits 31:0
Time in seconds	Fraction of seconds in nanoseconds

The GigEx device provides a mechanism for reading the PTP Time Register atomically. Writing any value to the PTP Time Register will not overwrite the register but will latch all 64 bits of the time and prevent their update. Reading from the PTP Time Register will then shift out the 64 bit value 16 bits or 8 bits at a time (depending on whether the interface is in 16 bit SRAM mode or not). The most significant word/byte is read first. The process for reading the time in 16 bit SRAM mode is:

1. Write any value to the PTP Time Register. The time of this write will be the exact time registered ready for reading
2. Read the most significant 16 bits from the PTP time register
3. Read two more words for bits 47:32 and 31:16 of the time
4. Read the least significant 16 bits from the PTP time register

For all other interface modes the process is:

1. Write any value to the PTP Time Register. The time of this write will be the exact time registered ready for reading
2. Read the most significant byte from the PTP time register
3. Read 6 more bytes for bits 55:48 then 47:40 then 39:32 then 31:24 then 23:16 then 15:8 of the time
4. Read the least significant byte from the PTP time register

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	1	0	0	0	1	b	**Event Time Register**

GigEx provides a mechanism for recording and timestamping the occurrence of an external hardware event. When configured as an event pin, a rising edge on the TrigEvent pin will latch the current value of the PTP Timer Register in the Event Time Register. The external device can then read back this register to determine the exact time of the event. When a PTP master clock is present this time will be synchronised to the master clock and hence to all other PTP devices on the network. When a PTP master clock is not present then the PTP Time Register will reflect the time elapsed since the GigEx booted.

The GigEx device provides a mechanism for reading the event time register atomically. Writing any value to the Event Time Register will not overwrite the register but will latch all 64 bits of the time and prevent their update should a further event occur before the time can be read out. Reading from the Event Time Register will then shift out the 64 bit value 16 bits or 8 bits at a time (depending on whether the interface is in 16 bit SRAM mode or not). The most significant word/byte is read first. The process for reading the event time in 16 bit SRAM mode is:

1. Write any value to the Event Time Register. This will prevent the update of the event time if further events occur during the reading process.
2. Read the most significant 16 bits from the Event Time register
3. Read two more words for bits 47:32 and 31:16 of the time
4. Read the least significant 16 bits from the Event Time register

For all other interface modes the process is:

1. Write any value to the Event Time Register. This will prevent the update of the event time if further events occur during the reading process.
2. Read the most significant byte from the Event Time register
3. Read 6 more bytes for bits 55:48 then 47:40 then 39:32 then 31:24 then 23:16 then 15:8 of the time
4. Read the least significant byte from the Event Time register

The event pin is shared with the trigger output pin so only one of these features can be used. The direction of the pin can be set via the UART (see section 6.3) or with the web configuration server (see section 6.5).

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	1	0	0	1	0	b	**Pulse Width Register**

The Pulse Width Register controls the length of the pulse generated on the TrigEvent pin both for single trigger mode and 1 pulse per second mode. The length is expressed in units of 100 nanoseconds. Therefore if this register has a value of 10,000 then pulse will be 1 millisecond long.

The initial value for the pulse width is stored in flash memory and can be set with the web configuration server (see section 6.5). Writing to the pulse width register allows the pulse width to be changed dynamically.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	1	0	1	0	1	b	**Trigger/Event Control and Status Register**

GigEx can generate an interrupt to the external device when it detects an event on the TrigEvent pin when in Event mode. Writing to bit 7 of the Trigger/Event Control and Status Register enables or disables this interrupt generation. The status of the interrupt condition can be checked by reading bit 7 of this register which has the side effect of clearing the interrupt.

When in trigger mode, GigEx can generate a single trigger pulse at a predefined time or it can generate a 1 pulse per second output synchronised to the second counter in the PTP Time Register. The mode is controlled by bit 0 of this register which should be 0 for single trigger event or 1 for a 1 pulse per second trigger.

Writing to the Trigger/Event Control and Status Register:
Bit 7: Event interrupt enable
Bit 0: Single event (0) or 1PPS (1) trigger mode

Reading from the Trigger/Event Control and Status Register:
Bit 7: Event interrupt status. Side effect of clearing the interrupt condition when read.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	0	1	0	1	1	0	b	**Trigger Time bits 63:48**
1	0	0	0	1	0	1	1	1	b	**Trigger Time bits 47:32**
1	0	0	0	1	1	0	0	0	b	**Trigger Time bits 31:16**
1	0	0	0	1	1	0	0	1	b	**Trigger Time bits 15:0**

The Trigger Time register is used when in single trigger mode to specify the time that the trigger should be generated. The time is in the same format as the PTP Time Register - the top 32 bits are seconds and the bottom 32 bits are fractions of a second expressed in nanoseconds. When the value of the PTP Time Register passes the Trigger Time Register a single pulse will be generated on the TrigEvent pin of GigEx.

It is important that this register is updated atomically to avoid the generation of spurious triggers. The register must be written from top to bottom. i.e. bits 63:48 must be written first followed by bits 47:32 then bits 31:16 and finally bits 15:0. The Trigger Time Register will be updated by GigEx only when the bottom bits have been written. For 8 bit writes, bits 63:56 must be written first and bits 7:0 must be written last.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	1	0	0	0	0	0	b	**GigEx Hardware Version number**

The Hardware Version Number register can be read to find the version of GigEx hardware fitted to the board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312.

When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	0	1	0	0	0	0	1	b	**GigEx Software Version number**

The Software Version Number register can be read to find the version of GigEx software running on the board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312.

When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	1	1	1	0	0	0	0	b	**User Settings Flash Control/Status Register**

GigEx provides 256 bytes of flash memory for an external device connected to the high or low speed bus to store some settings which will be preserved across power cycles. The User Settings Flash Control/Status Register controls when these settings are written to the flash.

Writing a 1 to bit 0 of this register will copy the contents of the 256 bytes of General Purpose User Comms Registers bank A (see description below of General Purpose User Comms Registers) from the external device into the flash. The new contents of the flash will then be copied out to the second bank B of General Purpose User Comms registers for reading by the external device. On power up the flash contents will always be read into the bank B registers to the external device so they can be used for non-volatile settings for the external device. This flow is shown in Figure 22, which also includes the flow of accessing the General Purpose User Comms Registers from the external device and a network device (see later description of General Purpose User Comms Registers).

Writing to the User Settings Flash Control/Status Register:

Bit 0: When changed from 0 to 1 a write to the flash from the general purpose user comms registers is initiated.

Reading from the User Settings Flash Control/Status Register:

Bit 0: When read as a 1 the flash update is in progress.

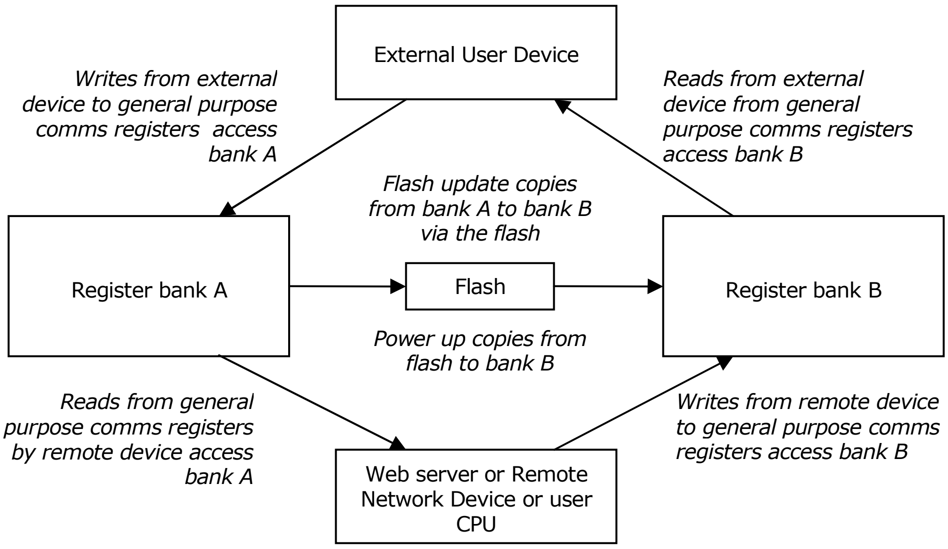

External User Device
Writes from external device to general purpose comms registers access bank A
Reads from external device from general purpose comms registers access bank B
Flash update copies from bank A to bank B via the flash
Register bank A
Flash
Register bank B
Power up copies from flash to bank B
Reads from general purpose comms registers by remote device access bank A
Writes from remote device to general purpose comms registers access bank B
Web server or Remote Network Device or user CPU

Figure 22. Update of Flash from General Purpose User Comms Registers

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	1	1	1	0	1	0	1	b	**Mailbox Interrupt Enable/Status Register**

The Mailbox Interrupt Enable/Status register provides a mechanism for a remote device on the network or the user CPU to generate an interrupt to the external user device connected to the GigEx by the low or high speed bus. This could be used, for example, to inform the external device that the remote device or CPU has updated some of the General Purpose User Comms Register values and that they should be re-read by the external device.

Writing to the Mailbox Interrupt Enable/Status Register:

Bit 6: When 1, the mailbox interrupt enable will be set and the status of the interrupt will be reflected on the nINT pin. When 0, the nINT pin will not be updated with the mailbox interrupt status but the interrupt status will still be reported in this register.

Reading from the Mailbox Interrupt Enable/Status Register:

Bit 6: When read as a 1 the interrupt is active. Reading has the side effect of clearing the interrupt condition.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	0	1	1	1	0	1	1	0	b	**Mailbox to User CPU Register**

The Mailbox to User CPU register can be used by the external device to generate an interrupt to the User CPU. When the external device writes 1 to bit 0 of this register an interrupt will be generated on the user CPU. Assuming the interrupt is enabled, the User CPU will jump to its interrupt service address where it can perform operations to service the external device. The interrupt will be cleared when the User CPU reads from the Mailbox from External Interface Control/Status register. See section 7.2.1 for details on User CPU interrupt handling.

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Register
1	1	x	x	x	x	x	x	x	b	**General Purpose User Comms Registers**

GigEx provides 256 bytes in each direction to pass user control and status information between a remote network device, the embedded web server and an external user device connected to the high or low speed bus, see Figure 22. There are two banks of registers, one to the external device and one from the external device. When the external user device writes to one of these registers it can be read by the embedded web server or by a remote device over the network using a command sent to the network command port (see section 6.4). Similarly, a remote device can write to a register using a command over the network or the web server can write to the register ready for the external device to read.

Note that the dual banks of registers means that a value written to a register cannot be read back from the same device. i.e. the external device cannot read back values it writes to the general purpose registers.

It is also possible for the external device to make use of the flash storage on GigEx using these general purpose registers. Refer to the User Settings Flash Control/Status register above for details. On power up, the values of the registers to the external device will be initialised from flash memory allowing a set of non-volatile settings to be preserved across power cycles.

See section 6.5 for details of how to access these registers from the embedded web server and section 6.4 for details of how to write to these registers from a remote device on the network.

6 GigEx Programming

6.1 Communicating via Ethernet

GigEx implements 16 simultaneous high speed network channels. The following section describes how to implement communications over GigEx channels.

Each of the 16 channels has an associated state. Requests are made to the GigEx device to change the state of a connection. The GigEx device will act on this request and report changes in state back to the user. The time taken to act on a request may depend on external devices and could be delayed by some time or may never happen. For example, a request to connect to a remote TCP device must wait until the remote device is ready for the connection which may take a long time and may never happen if the remote device never becomes ready.

The required steps to communicate over TCP and UDP connections are slightly different and will be presented separately.

6.1.1 TCP Connections

The default state for connections is CLOSED. To act as a TCP server, the user should request a state change to LISTEN. To act as a TCP client, the user should request a state change to CONNECT. At this point, the GigEx device will sample the settings on the other control registers for the channel and begin listening on a port or initiate a connection to a remote device.

When a connection is made either from a remote device or to a remote device, the GigEx device will report a state change to ESTABLISHED. The user must acknowledge this by setting the state control register to ESTABLISHED.

Once the connection has been established, both server and client behaviour is identical. An established connection can be used to transfer data between the external user device attached to the GigEx high or low speed bus and the remote device on the network.

The user can request that a connection is closed by requesting a state change to CLOSED. When the connection has been closed, the GigEx device will report a state change to CLOSED.

If the remote end closes the connection the GigEx device will report a state change to CLOSED. The user must respond by setting the ‘Connection Control’ register to CLOSED.

The state diagram for TCP connections is shown in Figure 23.

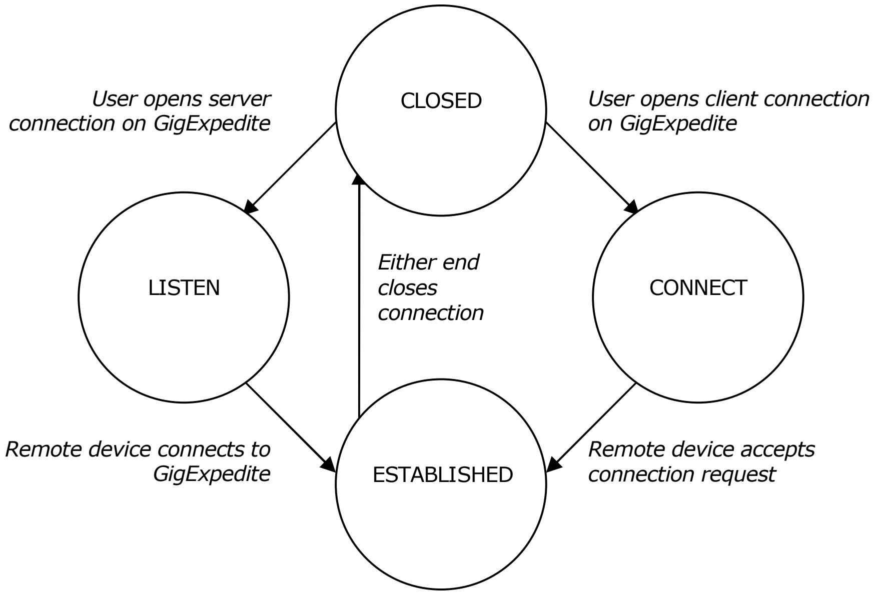

CLOSED
LISTEN
CONNECT
ESTABLISHED
User opens server connection on GigExpedite
User opens client connection on GigExpedite
Either end closes connection
Remote device connects to GigExpedite
Remote device accepts connection request

Figure 23. State transition diagram for TCP connections

Examples of pseudo code for various scenarios are shown below.

TCP server

The following pseudo code shows how to use the GigEx device as a TCP server.

```
Initialisation:
    Write p to the 'Local Port' register (where p is the local port to listen on)
    Write 0x2E80 to the 'Payload size/TTL' register to set the maximum payload size to 1472 bytes
        and time to live to 128 (note the maximum payload size is ignored for TCP)
    Write 0x000D to the 'Interrupt enable' register to enable state change, data from and data to
        network interrupts.
    Write enable, TCP and LISTEN (0x0031) to the 'Connection control' register to start listening on
        the connection

When an interrupt is received:
    Read 'Interrupt status' register
    If 'State change' interrupt received (bit mask 0x0008),
        Read 'Connection State' register
        If new state is CLOSED,
            Clean up local state
            Write CLOSED (0x0030) to 'Connection Control' register
        If new state is ESTABLISHED,
            Write ESTABLISHED (0x0033) to 'Connection Control' register
    If 'Data buffer from network not empty' interrupt received (bit mask 0x0001),
        Read 'Frame length' register
        Read n bytes from 'Data FIFO' register where n is the value read from the 'Frame length'
            register
    If 'Data buffer to network not full' interrupt received (bit mask 0x0004) and data is available to
        send and connection state is ESTABLISHED,
        Write up to 64kbytes of data to 'Data FIFO' register
```

Note that the data transfers to/from the 'Data FIFO' register could take place as DMA transfers since once the transfer has started no intervention by the local application is required.

It is possible to close the connection from the server end by writing CLOSED to the connection state. This will request a termination of the connection (i.e. send TCP FIN message).

It may be desirable to delay setting the Connection Control register to CLOSED until the data buffers are empty. This will be when the 'Data buffer from network not empty' interrupt is clear and the 'Data buffer to network empty' interrupt is set. However, it may be that the remote end has closed unexpectedly preventing the GigEx device from transmitting all the data in its buffers. In this case, the 'Data buffer to network empty' interrupt will not become set and the buffer must be flushed by disabling the channel (i.e. write 0x0000 to the Connection state register).

If the channel is deactivated when CLOSED is received (i.e. bit mask of 0x0010 or 0x0000 is written instead of 0x0030) then all data received from the network or queued for transmission to the network on this connection will be lost. By writing 0x0030, the connection is left enabled and data will remain in the GigEx buffers until it has been read or until the connection is re-established.

It is possible to start queuing data for transmission to the network before the connection is established. Any queued data will be transmitted as soon as the connection becomes established. It is also possible for data from the network to remain in the GigEx buffers after a status of CLOSED is reported.

TCP client

The following pseudo code shows how to use the GigEx device as a TCP client.

Initialisation:

```
    Write p to the 'Local Port' register (where p is the local port to use to connect). This can be any
        free port number or 0 for GigEx to choose the port number.
    Write the remote IP address to connect to into the 'Connection IP address' registers
    Write the remote port to connect to into the 'Remote Port' register
    Write 0x2E80 to the 'Payload size/TTL' register to set the maximum payload size to 1472 bytes
        and time to live to 128 (note the maximum payload size is ignored for TCP)
    Write 0x000D to the 'Interrupt enable' register to enable state change, data from and data to
        network interrupts.
    Write enable, TCP and CONNECT (0x0032) to the 'Connection control' register to attempt
        connection to the remote device
```

When an interrupt is received:

```
    Read 'Interrupt status' register
    If 'State change' interrupt received (bit mask 0x0008),
        Read 'Connection State' register
        If new state is CLOSED,
            Clean up local state
            Write CLOSED (0x0030) to 'Connection Control' register
        If new state is ESTABLISHED,
            Write ESTABLISHED (0x0033) to 'Connection Control' register
    If 'Data buffer from network not empty' interrupt received (bit mask 0x0001),
        Read 'Frame length' register
        Read n bytes from 'Data FIFO' register where n is the value read from the 'Frame length'
            register
    If 'Data buffer to network not full' interrupt received (bit mask 0x0004) and data is available to
        send and connection state is ESTABLISHED,
        Write up to 64kbytes of data to 'Data FIFO' register
```

Note that the data transfers to/from the 'Data FIFO' register could take place as DMA transfers because once the transfer has started no intervention by the local application is required.

To close the connection to the server, write CLOSED (0x0030) to the 'Connection Control' register. When the connection has been successfully closed, the 'Connection state' register will be set to CLOSED and a state change interrupt will be generated.

If the channel is disabled instead of CLOSED (i.e. value of 0x0010 or 0x0000 is written to the Connection Control register instead of 0x0030) then all data received from the network or queued for transmission to the network will be lost. By writing 0x0030, the channel is left enabled and data will remain in the GigEx buffers until it has been read or until the connection is re-established.

It is possible to start queuing data for transmission to the network before the connection is established. Any queued data will be transmitted as soon as the connection becomes established. It is also possible for data from the network to remain in the GigEx buffers after a status of CLOSED is reported.

6.1.2 UDP connections

UDP connections are similar to TCP connections but they do not use the LISTEN and CONNECT states. A state transition request from CLOSED to ESTABLISHED is used to open a connection and a transition from ESTABLISHED to CLOSED is used to close a connection. When the 'Connection Control' register is set to ESTABLISHED and the GigEx 'Connection Status' is set to ESTABLISHED in response then data can be transmitted or received on the connection.

For UDP connections, the TCP bit (bit 4) should be zero in all writes to the 'Connection Control' register.

For incoming data, each frame will be preceded by the UDP header data detailing the source (i.e. remote) IP address and port number. For outgoing data, the datagram length must be written to the 'Frame Length' register immediately before the data itself is written.

Packets will be received from the network only if they match the GigEx Local IP Address and the Local Port of a connection.

The 'Connection IP Address' register can be used to filter received data. When set to 0.0.0.0 data will be received from any IP address. When set to a valid IP address only data received from that remote address will pass through the GigEx device.

The 'Remote Port' register can be used to filter received data. When set to 0 data will be received from any port on the remote device. When set to a valid port number, only data from that port on the remote device will pass through the GigEx device. Setting the Connection IP Address to 0.0.0.0. and the Remote Port to 0000 is permissible and allows all data matching the Local Port register through.

The following pseudo code shows how to use the GigEx as a UDP device:

```
Initialisation:
    Write p to the 'Local Port' register (where p is the local port to use for the connection). This can
        be any free port number or 0 for GigEx to choose the port number (for transmission
        only).
    Write the remote IP address into the 'Connection IP address' registers.
    Write the remote port into the 'Remote Port' register.
    Write 0x2E80 to the 'Payload size/TTL' register to set the maximum payload size to 1472 bytes
        and time to live to 128
    Write 0x000D to the 'Interrupt enable' register to enable state change, data from and data to
        network interrupts.
    Write enable, UDP and ESTABLISHED (0x0023) to the 'Connection control' register to open the
        UDP connection

When an interrupt is received:
    Read 'Interrupt status' register
    If 'State change' interrupt received (bit mask 0x0008),
        Read 'Connection State' register
        If new state is CLOSED,
                Clean up local state
                Write CLOSED (0x0020) to 'Connection Control' register
                Mark connection as closed
        If new state is ESTABLISHED,
                Mark connection as open
If 'Data buffer from network not empty' interrupt received (bit mask 0x0001),
        Read 'Frame length' register
        Read 6 bytes of UDP header from 'Data FIFO' register
        Read n bytes from 'Data FIFO' register where n is the value read from the 'Frame length'
                register
If 'Data buffer to network not full' interrupt received (bit mask 0x0004) and data is available to
        send and connection is open,
        Write datagram length to 'Frame length' register
        Write up to 64k datagram data to 'Data FIFO' register
```

When reading a frame from the GigEx, the data will be preceded by UDP header data. The frame header is as follows:

Header offset (bytes)	Data
0	Remote IP address (high 16 bits)
2	Remote IP address (low 16 bits)
4	Remote port

Note that the 6 bytes of header data are not included in the value read from the frame length register (i.e. the total length of the read will be 6+frame length register bytes).

The remote IP address can be a multicast address. See Section 6.1.3 for details on multicasting.

6.1.3 Multicast UDP connections

The GigEx device supports multicast transmission and reception on UDP connections. Note that TCP multicasting is not supported.

To implement multicast transmission, set the 'Connection IP Address' register to be a multicast address in the range 224.0.0.0 to 239.255.255.255 and the 'Remote port' register to be the multicast port to transmit to. If the destination address is in this multicast range, the GigEx device will automatically handle this as a multicast transmission. The 'Local Port' register will be used as the source port for transmissions.

To implement multicast reception, set the 'Connection IP Address' register to be a multicast address in the range 224.0.0.0 to 239.255.255.255 and the 'Remote port' register to the correct multicast port to listen to or 0 to receive all multicast data for this address. The GigEx device will automatically handle this as a multicast reception including issuing all IGMP messages to join the correct multicast group. When the connection is closed, the IGMP messages to leave the multicast group will be issued by the GigEx device.

Note that to receive multicast messages from remote devices it may be necessary to have a multicast router present on the network. This will depend on your network infrastructure.

6.1.4 Auto Connection Management

In some circumstances, GigEx is able to automatically manage the 16 data channels' state. Where a channel will only ever be used in the same manner by an application then GigEx can handle all state control leaving the user application to simply transfer data to and from the remote network device. For example, if a channel will only be used as a TCP server on a fixed port then GigEx can be programmed to open the port for listening on power up, manage the connection from a remote device and immediate re-open the channel for listening when the channel is closed from either end. This functionality can be useful for simple networking applications which concentrate mainly on data transfers rather than complex applications communicating with multiple remote devices.

Auto connection can be enabled and configured either via the UART (see section 6.3) or via the embedded web configuration server (see section 6.5).

During auto connection, the connection status registers will remain active so the external device can monitor these registers to determine when the connection is established or closed. The external user device can also write to the connection control register to close the connection from the local side which will also force GigEx to attempt to re-open the connection automatically.

By default, when GigEx closes a connection it will not reset the data channel. This means that any data in the incoming or outgoing buffers will be left for when the next connection is made. The external device can flush the data buffers by writing a 0 to the appropriate connection control register.

When the external interface is in 8 or 16 bit SRAM mode, the external device must monitor the data FIFO state interrupts and transfer data to and from the network as required. When in FIFO or direct mode, the external device does not need to access any registers as GigEx is in full control of both the connection management and data transfers.

Auto-connection management also allows a channel to be configured to connect to the UART once the connection is established to allow data transfers between the remote device and the device attached to the UART.

6.2 Synchronous Ethernet, PTP and IEEE1588 Support

GigEx provides a number of mechanisms for synchronisation of devices across a network. At a low level, GigEx provides a synchronous Ethernet clock output recovered from the network which will be tightly coupled to similar SyncE outputs on other devices. At a higher level, GigEx supports the PTP protocol for synchronising timestamps between network devices and can generate a trigger or 1 pulse per second output to external hardware, or capture and timestamp an event from external hardware.

A single pin connected to the FPGA combines the functionality of trigger output, 1 pulse per second output and event input. The function of the pin can be configure with the UART (see section 6.3) or via the embedded web configuration server (see section 6.5).

6.2.1 Synchronous Ethernet

Synchronous Ethernet is a mechanism for recovering a clock from the Ethernet link. The recovered clock will be tightly synchronised between devices on the same network (subject to intermediate switches and routers supporting the protocol). GigEx outputs the 125MHz recovered clock on a pin connected to the FPGA.

6.2.2 PTP Time Synchronisation

Precision Time Protocol (PTP) is a mechanism for closely synchronising times across network devices. It relies on a master clock device present on the network supplying time information to the slave devices. GigEx acts as a slave only device and contains a time of day register for external devices connected to the user interface. The time of day register will be updated in collaboration with the master clock. If a master PTP clock is not present on the network, the time of day register will contain the time since power up and will be updated by the local oscillator on the ZestET2 module.

See section 5.2 for details of how to access the PTP time register.

6.2.3 Trigger Generation

GigEx can generate two forms of trigger to an external device. It can generate a one-shot trigger at a specified time in the future or it can generate a repeating 1 pulse per second trigger. If a PTP master clock is present on the network then these triggers will be tightly synchronised between devices on the network.

A one-shot trigger can be set by programming the required trigger time into the Trigger Time register. When the value of the PTP time register crosses the value in the Trigger Time register, a single pulse will be generated on the trigger output pin.

When in 1 pulse per second mode, a pulse will be generated on the trigger output pin when the seconds field of the PTP time register increments.

See section 5.2 for details of how to access the PTP trigger control registers.

6.2.4 Event Capture

When set as an event input, a rising edge on the event pin will cause the value of the PTP time register to be latched. This will generate an interrupt to the external device connected to GigEx (assuming the Event interrupt enable bit is set in the Trigger/Event Control and Status register). When the external device receives this interrupt it can read the Event Time register to determine the exact time of the rising edge of the event.

See section 5.2 for details of how to access the PTP event control registers.

6.3 UART Commands

GigEx can be controlled either via the high speed parallel register interface or through the low speed secondary interface. In FIFO and Direct modes, the high speed interface does not allow access to the control registers so the low speed interface is the only way to control the device. The low speed interface can be set to either SPI slave mode or UART mode. When the low speed interface is switched into UART mode, the GigEx can be controlled using an ASCII command set for convenient programming from an attached CPU. The supported ASCII commands are detailed below.

Each command must be terminated by a new line (ASCII character 10) or carriage return (ASCII character 13). Commands containing a '?' character will return the status of various values in the GigEx. Other commands are used to control the GigEx settings.

Using the ASCII command set it is possible to perform all functions available through the high speed interface allowing a simple (but slower) control mechanism from external processors with embedded UARTs. It is also possible to connect the UART to one of the high speed network channels to allow direct data transfer between the network and the device attached to the UART. See section 6.3.3. The UART command set includes higher level commands than writing and reading registers, for example for setting up channels and transferring data over channels.

In the examples below, the commands sent to the GigEx are shown in italics (e.g. ***AT+R=0***) . The responses from the GigEx are shown in normal font (e.g. **AT+R=0,0**).

Command parameters in square brackets [] are optional. If an optional parameter is used then all the previous optional parameters in the list must also be used. e.g. in the list of optional parameters *[Parameter4, Parameter5, Parameter6, Parameter7]*, if Parameter5 is used then Parameter4 must also be used.

Error messages returned by GigEx are of the form **+ERROR:ParameterN** where N is the number of the parameter sent by the external device UART that contains the error.

Numerical parameters can be passed either as decimal values or as hexadecimal with a prefix of '**0x**'. IP addresses should be passed as strings enclosed in double quotation marks. For example **"192.168.1.100"**.

GigEx will also generate asynchronous messages when events occur on the Ethernet link or when the state of channels changes. These messages are described in section 6.3.2.

The UART settings can be controlled by the web configuration server as well as through the UART itself. The default settings are 8 bits, 1 stop bit, no parity, no handshaking and 115200 baud.

6.3.1 UART Control Messages

AT	**Heartbeat**

Parameters

None

Return Text

AT	Function succeeded
+ERROR:Parameter0	End of line not found after command

Description

The **AT** command can be used to confirm that the UART link is active and functioning. If the GigEx returns **AT** to the command then it has received and understood the command and it is alive and functioning.

Example

```
AT
AT
```

ATE	**Enable/Disable echo**

Parameters

None

Return Text

+ATE=ON	Function succeeded - echo now turned on
+ATE=OFF	Function succeeded - echo now turned off
+ERROR:Parameter0	End of line not found after command

Description

The **ATE** command enables or disables character echoing. When echo is enabled, all characters received from the UART by GigEx are immediately transmitted back to the UART which can be useful when accessing the serial port from a terminal as it gives feedback to confirm the command has been entered correctly.

Example

> ***ATE***
> **+ATE=OFF**
> ***ATE***
> **+ATE=ON**

AT+IPR=*<rate>*	**Set UART baud rate**
AT+IPR?	**Query UART baud rate**

Parameters

rate	Baud rate for future UART commands. Can be 9600, 19200, 38400, 57600, 115200, 230400, 460800 or 921600

Return Text

+IPR=*<rate>*	Function succeeded - current baud rate returned
+ERROR:Parameter0	Illegal baud rate specified

Description

The **AT+IPR** command changes the baud rate of the UART for future accesses. The **AT+IPR?** command returns the current baud rate setting. Note that if the baud rate is changed, the response to setting a new baud rate will be sent at the new baud rate setting.

Example

> ***AT+IPR?***
> **+IPR=9600**
> ***AT+IPR=19200***
> **+IPR=19200**

AT+IFC=*<flow>*	**Set UART flow control mode**
AT+IFC?	**Query UART flow control mode**

Parameters

flow	Flow control mode for future UART commands. A value of 0 indicates flow control is disabled. A value of 1 indicates RTS/CTS hardware flow control is enabled.

Return Text

+IFC=*<flow>*	Function succeeded - current flow control mode returned
+ERROR:Parameter0	Illegal flow control mode specified

Description

The **AT+IFC** command changes the flow control mode of the UART for future accesses. The **AT+IFC?** command returns the current flow control mode setting. Note that if the flow control mode is changed, the response to setting a new flow control mode will be sent with the new flow control mode setting.

The UART supports no flow control (setting is 0) or hardware flow control using the RTS/CTS signals (setting is 1).

Example

```
AT+IFC=0
+IFC=0
AT+IFC?
+IFC=0
```

AT+ICF=*<datastop> [,<parity>]*	**Set UART data, stop and parity bits**
AT+ICF?	**Query UART data, stop and parity bits**

Parameters

datastop	Data/stop bit combination setting (see below).
parity	Optional parity setting

Return Text

+ICF=*<datastop>, <parity>*	Function succeeded - current data stop and parity settings returned
+ERROR:Parameter0	Illegal data/stop bit combination specified
+ERROR:Parameter1	Illegal parity bit setting specified

Description

The **AT+ICF** command changes the number of data bits and stop bits and controls whether a parity bit is included. The **AT+ICF?** command returns the current data, stop and parity bit settings. Note that if the UART mode is changed, the response to setting a new UART mode will be sent with the new UART mode setting.

The valid values for *datastop* and *parity* are shown below. If the optional parity setting is not specified, a default value of 0 will be used.

datastop	***parity***	**Data bits**	**Stop bits**	**Parity**
1		8	2	None
2	0	8	1	Odd
2	1	8	1	Even
3		8	1	None
4		7	2	None
5	0	7	1	Odd
5	1	7	1	Even
6		7	1	None

Example

```
AT+ICF=3
+ICF=3,0
AT+ICF=5,1
+ICF=5,1
AT+ICF?
+ICF=5,1
```

AT+VERSION?	**Query firmware version**

Parameters

None

Return Text

+VERSION="<*hwversion*> **-** <*swversion*>**"**	Function succeeded - current firmware version returned
+ERROR:Parameter0	End of line not found after command

Description

The **AT+VERSION?** command returns the current firmware version. The first value indicates the hardware revision, the second value indicates the software revision. A 'F' suffix on either value indicates that GigEx is operating in fallback mode. Fallback mode is a special state entered when the main firmware is corrupted due to a failed firmware upgrade. Fallback mode has limited functionality and indicates that the firmware upload failed and should be reattempted as soon as possible.

Example

```
AT+VERSION?
+VERSION="2.0 - 2.0"
```

AT+INTERFACE=*<clkout>***,***<ptpdir>***,***<mode>* [**,***<txwidth>***,***<rxwidth>*]	
	Set user interface mode
AT+INTERFACE?	**Query user interface mode**

Parameters

clkout	User interface clock direction. A value of 1 sets the clock as an output from GigEx, a value of 0 sets it as an input to GigEx.
ptpdir	PTP trigger/event direction. A value of 1 enables the PTP trigger or 1 pulse per second output. A value of 0 enables PTP event capture.
mode	High speed interface mode. A value of 0 sets the interface into 16 bit SRAM mode, 1 sets 8 bit SRAM mode, 2 sets FIFO mode and 3 sets Direct mode.
txwidth	Optional. Direct mode transmit bits. Must be a multiple of 8 and be between 0 and 32 inclusive.
rxwidth	Optional. Direct mode receive bits. Must be a multiple of 8 and be between 0 and 32 inclusive.

Return Text

+INTERFACE=*<clkout>***,***<ptpdir>***,***<mode>***,***<txwidth>***,***<rxwidth>*	
	Function succeeded - current interface mode returned
+ERROR:Parameter0	Illegal clock direction specified
+ERROR:Parameter1	Illegal PTP direction specified
+ERROR:Parameter2	Illegal mode specified
+ERROR:Parameter3	Illegal transmit bits specified
+ERROR:Parameter4	Illegal receive bits specified

Description

The **AT+INTERFACE** command changes the mode of the user interface. The **AT+INTERFACE?** command returns the current user mode setting.

Example

```
AT+INTERFACE?
+INTERFACE=0,1,2,8,8
AT+INTERFACE=1,0,0
+INTERFACE=1,0,0,8,8
```

AT+LINK?	**Query Ethernet link status**

Parameters

None

Return Text

+LINK=*<state>***,***<duplex>***,***<rate>*	Function succeeded - current link status returned
+ERROR:Parameter0	End of line not found after command

Description

The **AT+LINK?** command returns the current Ethernet link status. The Ethernet link status consists of three parts: whether the link is up or down, whether the link is running in full or half duplex mode and finally the bit rate of the connection. The link state is reported as **UP** or **DOWN**, the duplex mode is reported as **FULL** or **HALF** and the rate is reported as **1000**, **100**, **10** or **0**.

Example

> ***AT+LINK?***
> **+LINK=UP, FULL, 1000**

AT+IPADDR=*<IPAddr>***,***<SubNet>***,***<Gateway>* [**,***<DHCP>***,***<AutoIP>***,***<Jumbo>***,***<HTTP>***,***<Control>*]	
	Set network IP settings
AT+IPADDR?	**Query network IP settings**

Parameters

IPAddr	Fixed IP address setting
SubNet	Fixed subnet mask setting
Gateway	Fixed gateway IP address setting
DHCP	Optional. Enable or disable DHCP client. A value of 1 enables the DHCP client and a value of 0 disables it
AutoIP	Optional. Enable or disable AutoIP client. A value of 1 enables the AutoIP client and a value of 0 disables it
Jumbo	Optional. Value of maximum permitted transmit frames in bytes. Should be between 1500 and 7500 bytes and should be a multiple of 4 bytes
HTTP	Optional. Port for configuration web server. Normally set to the standard HTTP port of 80.
Control	Optional. Port for the control interface. Normally set to 0x5001 but can be changed to free this port for the user's application.

Return Text

+IPADDR=*<IPAddr>***,***<SubNet>***,***<Gateway>***,***<DHCP>***,***<AutoIP>***,***<Jumbo>***,***<HTTP>***,***<Control>,<MAC>*	
	Function succeeded - current network IP settings returned
+ERROR:Parameter0	Illegal fixed IP address specified
+ERROR:Parameter1	Illegal fixed subnet mask specified
+ERROR:Parameter2	Illegal gateway IP address specified
+ERROR:Parameter3	Illegal value for enable/disable DHCP client setting
+ERROR:Parameter4	Illegal value for enable/disable AutoIP setting
+ERROR:Parameter5	Illegal value for maximum transmit frame length
+ERROR:Parameter6	Illegal value for HTTP web server port
+ERROR:Parameter7	Illegal value for control interface port

Description

The **AT+IPADDR** command changes the network IP address settings. The **AT+IPADDR?** command returns the current network IP address settings. In addition to the parameters listed above, the **AT+IPADDR?** command also returns the MAC address of the GigEx.

Settings will be written to the flash of the module and the GigEx will be rebooted if necessary. Note that all open connections should be closed before attempting to change the network settings.

Example

AT+IPADDR?
+IPADDR="192.168.1.100", "255.255.255.0", "192.168.1.1", 0, 0, 1500, 80, 20481, 0050c2a7b001
AT+IPADDR="192.168.1.101","255.255.255.0","192.168.1.3",0,0
+IPADDR="192.168.1.101", "255.255.255.0", "192.168.1.3", 0, 0, 1500, 80, 20481, 0050c2a7b001

AT+SETTINGSTOFLASH	**Copy current user settings to flash memory**

Parameters

None

Return Text

OK	Function succeeded - current user settings written to flash
+ERROR:Parameter0	End of line not found after command

Description

The **AT+SETTINGSTOFLASH** command copies the contents of the 256 bytes of General Purpose User Comms registers from the external device into the flash. The new contents of the flash will then be copied out to the second bank of General Purpose User Comms registers back to the external device. On power up the flash contents will always be read into the registers to the external device so they can be used for non-volatile settings for the external device. See Figure 22 for details of this process.

Example

AT+SETTINGSTOFLASH
OK

AT+W=*<address>,<data> [,<byteenable>]*	**Write to user register**

Parameters

address	10 bit address of register to write to
data	Data to write to register

byteenable	Optional byte enable bits for 16 bit writes

Return Text

OK	Function succeeded
+ERROR:Parameter2	Illegal byte enable value specified
+ERROR:Address	Address out of range

Description

The **AT+W** command writes to one of the user control registers exactly as if it had been written by the high speed interface. For 8 and 16 bit SRAM modes, this provides an alternative mechanism for writing to the registers. For FIFO and direct modes where the registers are not accessible by the high speed interface this provides the only way to write to these registers.

When in 16 bit SRAM mode, the **AT+W** command performs a 16 bit write. The bottom bit of the 10 bit address must always be 0. The optional *byteenable* parameter is a 2 bit byte enable mask so a value of 2 writes only the most significant byte of data, a value of 1 writes the least significant byte of data and a value of 3 writes both bytes of data.

When in any other user interface mode, the **AT+W** command performs an 8 bit write. The *byteenable* parameter is not used in these modes.

The **AT+W** command allows data transfer by writing to the data FIFO register address.

Example

```
AT+W=0x380,0x123
OK
```

AT+R=<*address*>	**Read from user register**

Parameters

address	10 bit address of register to read from

Return Text

+R=<*address*>,<*data*>	Function succeeded - data read from register is returned
+ERROR:Address	Address out of range

Description

The **AT+R** command reads from one of the user control registers exactly as if it had been read by the high speed interface. For 8 and 16 bit register modes, this provides an alternative mechanism for reading the registers. For FIFO and direct modes where the registers are not accessible by the high speed interface this provides the only way to read these registers.

When in 16 bit register modes, the **AT+R** command performs a 16 bit read. The bottom bit of the 10 bit address must always be 0 and the data returned will be a 16 bit value.

When in any other user interface mode, the **AT+R** command performs an 8 bit read and the data returned will be an 8 bit value.

The **AT+R** command allows data transfer across the network by reading from the data FIFO register address.

Example

```
AT+R=0x202
+R=0x202,0x0164
```

AT+TCPLISTEN=<*channel*>**,**<*localport*>[**,**<*TTL*>]
Opening listening TCP server connection

Parameters

channel	Index of channel to open for listening. Valid values are 0 to 15 inclusive.
localport	Local port number to listen on. Value is a 16 bit number.
TTL	Optional 8 bit value to use for time to live field in transmitted packets. If not specified, a value of 128 will be used.

Return Text

OK	Function succeeded - connection is open for listening
+ERROR:Parameter0	Channel number is out of range 0-15
+ERROR:Parameter1	Local port number is out of range 0-65535
+ERROR:Parameter2	Time to live value is out of range 0-255
+ERROR	Failed to open connection for listening

Description

The **AT+TCPLISTEN** command opens one of the 16 available channels to listen for TCP connections from a remote device. It is a convenient wrapper function that programs the user control registers in a single command.

If successful, an asynchronous **+STATE** response will be returned when the channel is listening for connections (see section 6.3.2).

Example

```
AT+TCPLISTEN=4,8080,64
OK
```

AT+TCPCONNECT=<*channel*>**,**<*localport*>**,**<*remoteaddr*>**,**<*remoteport*>[**,**<*TTL*>]
Make TCP connection to remote device

Parameters

channel	Index of channel to open for connection. Valid values are 0 to 15 inclusive.
localport	Local port number to use for channel. Value is a 16 bit number.
remoteaddr	IP address of remote device to connect to.
remoteport	Port on remote device to connect to.
TTL	Optional 8 bit value to use for time to live field in transmitted packets. If not specified, a value of 128 will be used.

Return Text

OK	Function succeeded - channel is attempting to connect
+ERROR:Parameter0	Channel number is out of range 0-15
+ERROR:Parameter1	Local port number is out of range 0-65535
+ERROR:Parameter2	Illegal remote IP address specified
+ERROR:Parameter3	Remote port number is out of range 0-65535
+ERROR:Parameter4	Time to live value is out of range 0-255
+ERROR	Failed to open connection

Description

The **AT+TCPCONNECT** command opens one of the 16 available channels and attempts to make a TCP connection to a remote device. It is a convenient wrapper function that programs the user control registers in a single command.

If successful, an asynchronous **+STATE** response will be returned when the connection is established (see section 6.3.2).

Example

```
AT+TCPCONNECT=5,0,"192.168.1.20",0x1001
OK
```

AT+TCPDATA=<*channel*>	**Switch to data transfer mode for channel**

Parameters

channel	Index of channel to transfer data on. Valid values are 0 to 15 inclusive.

Return Text

+ERROR:Parameter0	Channel number is out of range 0-15

Description

The **AT+TCPDATA** command switches the UART into data transfer mode and connects it to the specified channel number. All data sent to the UART will be transmitted on the requested high speed network channel and all data received by the channel from the network will be transmitted by the UART to the external device.

The data connection can be terminated by sending ASCII character 3 to the GigEx UART or if the TCP connection is shut down by either end. Since ASCII character 3 is used to switch out of data mode, an escape character must be used to allow transmission of the value 3 to the TCP connection and ASCII character 16 is used for this purpose. The escape sequences are therefore:

ASCII character 3	Switch out of data mode
ASCII character 16 followed by character 3	Send ASCII character 3 to TCP connection
ASCII character 16 followed by character 16	Send ASCII character 16 to TCP connection

Note that data transferred from the network to the external device via the UART is not escaped.

Example

AT+TCPDATA=5
Data received from TCP connection
Data to transmit to TCP connection

AT+TCPCLOSE=<*channel*>	**Close TCP connection**

Parameters

channel	Index of channel to close. Valid values are 0 to 15 inclusive.

Return Text

OK	Function succeeded - channel will be closed
+ERROR:Parameter0	Channel number is out of range 0-15

Description

The **AT+TCPCLOSE** command closes an active TCP channel.

If successful, an asynchronous **+STATE** response will be returned when the connection is closed (see section 6.3.2).

Example

AT+TCPCLOSE=5
OK

AT+UDPCONNECT=<*channel*>**,**<*localport*>**,**<*remoteaddr*>**,**<*remoteport*>[**,**<*payload*>**,**<*TTL*>] **Make UDP connection to remote device**

Parameters

channel	Index of channel to open for connection. Valid values are 0 to 15 inclusive.
localport	Local port number to use for channel. Value is a 16 bit number.
remoteaddr	Remote IP address for outgoing data and filter for remote IP addresses for incoming data.
remoteport	Remote port for outgoing data and filter for remote port numbers for incoming data.
payload	Optional 8 bit value for the maximum transmit payload size in 32 byte units. If not specified, a value of 128 will be used indicating a maximum payload size of 4096 bytes. Datagrams longer than this payload size will be transmitted as fragmented frames.
TTL	Optional 8 bit value to use for time to live field in transmitted packets. If not specified, a value of 128 will be used.

Return Text

OK	Function succeeded - channel is opening
+ERROR:Parameter0	Channel number is out of range 0-15
+ERROR:Parameter1	Local port number is out of range 0-65535
+ERROR:Parameter2	Illegal remote IP address specified
+ERROR:Parameter3	Remote port number is out of range 0-65535
+ERROR:Parameter4	Payload size is out of range 0-255
+ERROR:Parameter5	Time to live value is out of range 0-255
+ERROR	Failed to open connection

Description

The **AT+UDPCONNECT** command opens one of the 16 available channels for UDP data transfer. It is a convenient wrapper function that programs the user control registers in a single command.

If the channel is to be used to transmit UDP data to a remote device then the *remoteaddr* and *remoteport* parameters specify the details of the remote device to transmit to.

If the channel is to be used to receive UDP data from a remote device then the *remoteaddr* and *remoteport* parameters can be used to filter incoming data. Only data with a source address and port matching the *remoteaddr* and *remoteport* values will pass through GigEx. If either value is set to 0 then all data from any IP address and/or port will pass through GigEx.

If *remoteaddr* is a multicast address then data will be sent to or received from that multicast address. GigEx will manage the IGMP messages required to join and leave the multicast group automatically.

If successful, an asynchronous **+STATE** response will be returned when the channel is ready for data transfers (see section 6.3.2).

Example

AT+UDPCONNECT=6,0,"192.168.1.20",123
OK

AT+UDPDATA=<*channel*> [,<*length*>]	**Switch to data transfer mode for channel**

Parameters

channel	Index of channel to transfer data on. Valid values are 0 to 15 inclusive.
length	Optional length for transmitting fixed sized datagrams.

Return Text

+ERROR:Parameter0	Channel number is out of range 0-15
+ERROR:Parameter1	Length is out of range 0-65535

Description

The **AT+UDPDATA** command switches the UART into data transfer mode and connects it to the specified high speed network channel number. All data sent to the UART will be transmitted on the requested channel and all data received by the channel will be transmitted by the UART to the external device.

To transmit data with varying datagram lengths, the *length* parameter should be omitted. The 2 byte datagram length is written to the UART (most significant byte first) followed by the datagram payload. If all data to be transmitted has the same datagram length then the datagram length can be specified by the *length* parameter and only payload data needs to be written to the UART.

Data received from the network is transmitted out of the UART to the external device attached. If the *length* parameter is omitted, the two byte payload length will be transmitted on the UART first (most significant byte first) followed by the UDP header as follows:

Header offset (bytes)	Data
0	Remote IP address (most significant 8 bits, bits 31:24)
1	Remote IP address (bits 23:16)
2	Remote IP address (bits 15:8)
3	Remote IP address (least significant 8 bites, bits 7:0)
4	Remote port most significant byte
5	Remote port least significant byte

The data connection can be terminated by sending ASCII character 3 to the UART or if the UDP channel is closed by writing to the control registers from the high speed interface. Since ASCII character 3 is used to switch out of data mode, an escape character must be used to allow transmission of the value 3 to the TCP connection and ASCII character 16 is used for this purpose. The escape sequences are therefore:

ASCII character 3	Switch out of data mode
ASCII character 16 followed by character 3	Send ASCII character 3 to UDP connection
ASCII character 16 followed by character 16	Send ASCII character 16 to UDP connection

Note that data transferred from the network to the external device via the UART is not escaped.

Example

> ***AT+UDPATA=6,1000***
> **Data received from UDP connection**
> ***Data to transmit to UDP connection***

AT+UDPCLOSE=*<channel>*	**Close UDP channel**

Parameters

channel	Index of channel to close. Valid values are 0 to 15 inclusive.

Return Text

OK	Function succeeded - channel will be closed
+ERROR:Parameter0	Channel number is out of range 0-15

Description

The **AT+UDPCLOSE** command closes an active UDP channel.

If successful, an asynchronous **+STATE** response will be returned when the channel is closed (see section 6.3.2).

Example

> ***AT+UDPCLOSE=5***
> **OK**

AT+CONNECTIONSTATE?=*<channel>*	**Query state of connection**

Parameters

channel	Index of channel to query. Valid values are 0 to 15 inclusive.

Return Text

+STATE=*<channel>***,***<state>*	Function succeeded - channel state returned
+ERROR:Parameter0	Channel number is out of range 0-15

Description

The **AT+CONNECTIONSTATE?** command returns the current state of one of the 16 channels. The state can be one of **'CLOSED'**, **'LISTEN'**, **'CONNECT'** or **'ESTABLISHED'**.

Example

> ***AT+CONNECTIONSTATE?=2***

```
+CONNECTIONSTATE=2,ESTABLISHED
```

AT+DATASTATE?=<*channel*>	**Query state of data FIFO**

Parameters

channel — Index of channel to query. Valid values are 0 to 15 inclusive.

Return Text

+DATASTATE=<*channel*>**,**<*txstate*>**,**<*rxstate*>	Function succeeded - FIFO state returned
+ERROR:Parameter0	Channel number is out of range 0-15

Description

The **AT+DATASTATE?** command returns the current state of the data FIFO of one of the 16 channels. The *txstate* value can be one of **'TxEmpty'**, **'TxFull'** or **'TxNotFull'**. The *rxstate* value can be one of **'RxEmpty'** or **'RxNotEmpty'**.

Example

```
AT+DATASTATE?=2
+DATASTATE=2, "TxEmpty", "RxEmpty"
```

AT+AUTOOPEN=<*channel*>,<*enable*>[,<*TCP*>,<*server*>,<*localport*>,<*remoteaddr*>, <*remoteport*>,<*payload*>,<*TTL*>,<*UART*>]	**Enable autoopen for channel**
AT+AUTOOPEN?=<*channel*>	**Query autoopen settings for channel**

Parameters

channel — Index of channel to change settings for. Valid values are 0 to 15 inclusive.

enable — Enable or disable auto open on the channel. A value of 1 enables auto open, a value of 0 disables it.

TCP — A value of 1 sets the channel to be TCP, a value of 0 sets the channel to be UDP.

server — A value of 1 sets the TCP mode to be server (i.e. listening for connections), a value of 0 sets the TCP mode to client (i.e. attempting to connect to remote device). Ignored for UDP channels.

localport — Local port number to use for channel. Value is a 16 bit number.

remoteaddr — For TCP clients, the remote IP address to connect to. For UDP channels, the remote IP address for outgoing data and filter for remote IP addresses for incoming data.

remoteport
For TCP clients, the remote port to connect to. For UDP channels, the remote port for outgoing data and filter for remote port numbers for incoming data.

payload
Optional 8 bit value for the maximum transmit payload size in 32 byte units. If not specified, a value of 128 will be used indicating a maximum payload size of 4096 bytes. Datagrams longer than this payload size will be transmitted as fragmented frames. Ignored for TCP channels.

TTL
Optional 8 bit value to use for time to live field in transmitted packets. If not specified, a value of 128 will be used.

UART
Optional. A value of 1 indicates that the UART should switch to data mode on this channel when a connection is made and switch out of data mode when a connection is lost. A value of 0 (default) indicates that the UART should remain in command mode until the external device requests otherwise.

Return Text

+AUTOOPEN=*<channel>,<enable>,<TCP>,<server>,<localport>,<remoteaddr>,<remoteport>,*
<payload>,<TTL>,<UART>
Function succeeded - auto open settings returned

+ERROR:Parameter0	Channel number is out of range 0-15
+ERROR:Parameter4	Local port number is out of range 0-65535
+ERROR:Parameter5	Illegal remote IP address specified
+ERROR:Parameter6	Remote port number is out of range 0-65535
+ERROR:Parameter7	Payload size is out of range 0-255
+ERROR:Parameter8	Time to live value is out of range 0-255

Description

The **AT+AUTOOPEN** command controls the auto open mechanism on one of the channels. The GigEx device is able to manage channel connections without interaction from the external device. This feature is called auto open and can be controlled through the UART command interface or through the web configuration server. When auto open is enabled on a channel, GigEx will open a TCP channel as a server or client or initialise a UDP channel automatically. When a connection is closed by either end, GigEx will immediately re-open the channel for the next connection.

For example, one channel may be used as a command interface for the user application. If auto open is enabled, GigEx can open and listen on a port number ready for connections from remote devices. When a connection is made, the external device connected to GigEx can transfer data over the channel. When either the external device or the remote device closes the connection, GigEx will immediately re-open the port ready for another connection from a remote device. The local external device therefore only has to handle data transfers over the channel - GigEx handles all control and state changes internally.

The parameters for TCP and UDP auto open control are similar to those for manually opening and closing connections. In addition, the *UART* parameter will automatically connect the UART in data mode to a channel when a connection is established in a similar way to the **AT+TCPDATA** or **AT+UDPDATA** commands above. In the previous example, this would allow the external device to receive, process and reply to commands on the automatically controlled port via the UART.

Example

> ***AT+AUTOOPEN=8,1,1,0,0,"192.168.1.20",1001***
> **+AUTOOPEN=8,1,1,0,0,"192.168.1.20",1001,128,128,0**
> ***AT+AUTOOPEN?=8***
> **+AUTOOPEN=8,1,1,0,0,"192.168.1.20",1001,128,128,0**

6.3.2 UART Status Messages

GigEx can report asynchronous state changes by sending messages through the UART. These messages can occur at any time when the UART is not in data mode.

+LINK	**Ethernet link status change**

Message Text

+LINK=<*state*>**,**<*duplex*>**,**<*rate*>	Current link status returned

Description

This message will be sent when the state of the Ethernet link has changed. The link status consists of three parts: whether the link is up or down, whether the link is running in full or half duplex mode and finally the bit rate of the connection. The link state is reported as **UP** or **DOWN**, the duplex mode is reported as **FULL** or **HALF** and the rate is reported as **1000**, **100**, **10** or **0**.

Example

> **+LINK=UP, FULL, 1000**

+CONNECTIONSTATE	**State of channel has changed**

Message Text

+CONNECTIONSTATE=<*channel*>**,**<*state*>	Channel state returned

Description

This message will be sent when the connection state of one of the 16 channels has changed. The state can be one of **'CLOSED'**, **'LISTEN'**, **'CONNECT'** or **'ESTABLISHED'**.

Example

> ***AT+CONNECTIONSTATE?=2***
> **+CONNECTIONSTATE=2,ESTABLISHED**

6.3.3 Data Transfers Between the UART and Network Devices

GigEx allows the UART to be connected directly to one of the network channels for data transfer between a remote network device and the device attached to the UART. This can be achieved by issuing the **AT+TCPDATA** or **AT+UDPDATA** data commands once a connection has been established or with the auto-open feature when the UART connection is enabled.

When transferring data over a TCP connection, only payload data is transferred across the UART.

When transferring data over a UDP connection, if the **AT+UDPDATA** command was used to connect the UART to the channel with the optional *length* parameter specified then only payload data is transferred across the UART. If the *length* parameter was not specified or if auto-open was used to connect the UART then datagram length and UDP header data will be included in the data stream. Data transmitted to the network must be preceded by a 2 byte datagram length (sent most significant byte first). Data received from the network will be preceded by a 2 byte datagram length (sent most significant byte first) and a 6 byte UDP header as shown in the description of the **AT+UDPDATA** command.

If a connection is closed then the UART will switch out of data mode automatically.

6.4 Control Port Messages

GigEx maintains an open TCP control port on the network to process control messages that access various features in the device. A remote network device can send control messages to the GigEx device through this port.

A remote device sends a command request to the GigEx by opening a TCP connection to the control port of the GigEx. It then sends a packet of bytes consisting of a command number and appropriate parameters and waits for a response packet back from GigEx. The format of the command and response packets is given below.

Command 0xEA	**Get GigEx GUID**

Command Request Format

Byte Offset	Description
0	Command request code - always 0xEA
1-3	Dummy bytes (to round up to 4 byte boundary).

Response Format

Byte Offset	Description
0	Command request code - always 0xEA
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)
4-19	16 bytes of GUID

Description

The get GUID command can be used to read the 16 byte globally unique identifier (GUID) from the GigEx device. The GUID is a user programmable field that can be used to store information about the configuration of the board that the user can use to modify the behaviour of their application. For example, the ZestET2 board may be used in two different user products. The GUID can be used to differentiate between the two products from the remote device.

The GUID can be set with the Set GigEx GUID command below or via the web configuration server (see section 6.5).

Command 0xEB	Set GigEx GUID

Command Request Format

Byte Offset	Description
0	Command request code - always 0xEB
1-3	Dummy bytes (to round up to 4 byte boundary).
4-19	16 bytes of GUID

Response Format

Byte Offset	Description
0	Command request code - always 0xEB
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)

Description

The set GUID command can be used to write the 16 byte globally unique identifier (GUID) to the GigEx device. The GUID is a user programmable field that can be used to store information about the configuration of the board that the user can use to modify the behaviour of their application. For example, the ZestET2 board may be used in two different user products. The GUID can be used to differentiate between the two products from the remote device.

The GUID can be read with the GetGUID command above, with the ZestET2GetCardInfo host function (see section 9.5) or from the web configuration server (see section 6.5).

Command 0xEC	Set/Get GPIO Pin Settings

Command Request Format

Byte Offset	Description
0	Command request code - always 0xEC
1	Write settings to non-volatile memory. Set to 1 to copy the values to non-volatile memory. Set to 0 to just set the pin values now.
2	Update outputs. Set to 1 to update the output states. Set to 0 to just read back the input states.
3	Dummy byte (to round up to 4 byte boundary)
12-15	GPIO Select. Byte 12 is the most significant byte (GPIO31:24), byte 15 is the least significant byte (GPIO7:0). A 1 bit sets the pin in GPIO mode, a 0 sets the pin in conventional mode (i.e. high speed or low speed interface functionality).
16-19	GPIO Enable. Byte 16 is the most significant byte (GPIO31:24), byte 19 is the least significant byte (GPIO7:0). A 1 bit sets the GPIO pin as an output from GigEx, a 0 sets the GPIO pin as an input to GigEx.
20-23	GPIO Data Out. Byte 20 is the most significant byte (GPIO31:24), byte 23 is the

	least significant byte (GPIO7:0). A 1 bit sets the GPIO pin high, a 0 sets the GPIO pin low.

Response Format

Byte Offset	Description
0	Command request code - always 0xEC
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)
12-15	GPIO Select. Byte 12 is the most significant byte (GPIO31:24), byte 15 is the least significant byte (GPIO7:0). A 1 bit indicates the pin is in GPIO mode, a 0 indicates the pin is in conventional mode (i.e. high speed or low speed interface functionality).
16-19	GPIO Enable. Byte 16 is the most significant byte (GPIO31:24), byte 19 is the least significant byte (GPIO7:0). A 1 bit indicates the GPIO pin is an output from GigEx, a 0 indicates the GPIO pin is an input to GigEx.
20-23	GPIO Data Out. Byte 20 is the most significant byte (GPIO31:24), byte 23 is the least significant byte (GPIO7:0). The GPIO data out value returns the current output value being driven onto the GPIO pins.
24-27	GPIO Data In. Byte 24 is the most significant byte (GPIO31:24), byte 27 is the least significant byte (GPIO7:0). The GPIO data in value returns the current input value of the GPIO pins.

Description

The Set/Get GPIO Pin Settings command can be used to control the functionality of the GPIO pins on GigEx and to write a value to or read a value from the GPIO pins.

When a GPIO select bit is high, the pin is selected as a GPIO pin. When a GPIO select bit is low the pin has its normal functionality and forms part of the high speed or low speed interface.

When a GPIO enable bit is high, the GPIO pin drives the corresponding bit of the GPIO Data Out register onto the pin. When a GPIO enable bit is low, the pin is an input and the current state can be read from the GPIO Data In register.

Command 0xEF Set GigEx Settings

Command Request Format

Byte Offset	Description
0	Command request code - always 0xEF
1-3	Dummy bytes (to round up to 4 byte boundary).
4-7	Mask of settings to change. 32 bit value sent most significant byte first Bit 0: Set to 1 to change the IP address Bit 1: Set to 1 to change the gateway Bit 2: Set to 1 to change the subnet mask Bit 3: Set to 1 to change the HTTP web server port Bit 4: Set to 1 to change the control port Bit 5: Set to 1 to change the Auto IP setting Bit 6: Set to 1 to change the DHCP setting
8-11	Fixed IP address. Byte 8 is the first byte of the IP address, byte 11 is the last.
12-15	Fixed gateway IP address. Byte 12 is the first byte of the IP address, byte 15 is the last.
16-19	Fixed subnet mask. Byte 16 is the first byte of the mask, byte 19 is the last.
20-21	HTTP web server port. 2 byte port number sent most significant byte first.
22-23	Control port. 2 byte port number sent most significant byte first.
24	Enable AutoIP. Set to 0 to disable AutoIP, set to non-zero to enable AutoIP.

25	Enable DHCP. Set to 0 to disable DHCP, set to non-zero to enable DHCP.
26-27	Dummy bytes (to round up to 4 byte boundary)

Response Format

Byte Offset	Description
0	Command request code - always 0xEF
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)

Description

The set settings command can be used to change the current network settings of the GigEx device. Individual settings can be changed by setting bits in the programming mask at offset 4. Settings will be written to the flash of the module and the GigEx will be rebooted if necessary. Note that all open connections should be closed before attempting to change the network settings.

Command 0xF0 Get GigEx Settings

Command Request Format

Byte Offset	Description
0	Command request code - always 0xF0
1-3	Dummy bytes (to round up to 4 bytes)

Response Format

Byte Offset	Description
0	Command request code - always 0xF0
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)
4-5	Software version. 2 byte version number sent most significant byte first. Bit 15: When set to 1, the GigEx is operating in fallback mode Bit 14-0: Version number multiplied by 100 For example, a version of 0x00c8 equates to version 2.00
6-7	Hardware version. 2 byte version number sent most significant byte first. Bit 15: When set to 1, the GigEx is operating in fallback mode Bit 14-0: Version number multiplied by 100 For example, a version of 0x00c8 equates to version 2.00
8-11	Board serial number sent most significant byte first
12-15	IP Address. Byte 12 is the first byte of the IP address, byte 15 is the last.
16-19	Gateway IP address. Byte 16 is the first byte of the IP address, byte 19 is the last.
20-23	Subnet mask. Byte 20 is the first byte of the mask, byte 23 is the last.
24-25	HTTP web server port. 2 byte port number sent most significant byte first.
26-27	Control port. 2 byte port number sent most significant byte first.
28-33	MAC address. Byte 28 is the leftmost byte, byte 33 is the rightmost byte
34-35	Dummy bytes (to round up to 4 byte boundary)

Description

The get settings command can be used to request details of the current network settings of the GigEx device. The IP addresses returned are the current active values rather than the settings stored in flash so if DHCP or AutoIP are enabled the values will be the automatically assigned values.

Command 0xF1	**Reboot GigEx**

Command Request Format

Byte Offset	Description
0	Command request code - always 0xF1
1-3	Dummy bytes (to round up to 4 byte boundary)

Response Format

Byte Offset	Description
0	Command request code - always 0xF1
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)

Description

The reboot command will schedule a reboot of the GigEx. The reboot will take place after the response has been transmitted. Note that all open connections should be closed before attempting to reboot the GigEx device.

Command 0xF6	**Write general purpose user comms register**

Command Request Format

Byte Offset	Description
0	Command request code - always 0xF6
1	Offset into user comms registers to write to.
2-3	16 bit value to write to register (most significant byte sent first).

Response Format

Byte Offset	Description
0	Command request code - always 0xF6
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)

Description

The write general purpose user comms register control command sets the value of one of the general purpose registers to the external device attached to the GigEx. These registers can be read by the external user device at address 0x300 - 0x3ff (i.e. A9 and A8 are both 1). Note that one set of registers passes values from the network to the external user device and a second set passes values from the external user device to the network, see Figure 22. Therefore reading from an address using command 0xF7 below will not read back the value written with command 0xF6.

Command 0xF7	Read general purpose user comms register

Command Request Format

Byte Offset	Description
0	Command request code - always 0xF7
1	Offset into user comms registers to read from.
2-3	Dummy bytes (to round up to 4 byte boundary)

Response Format

Byte Offset	Description
0	Command request code - always 0xF7
1	Status byte. 0 for success, non-zero for failure
2-3	16 bit value read from register (most significant byte sent first)

Description

The read general purpose user comms register control command reads the value of one of the general purpose registers from the external user device attached to the GigEx. These registers can be written by the external user device at address 0x300 - 0x3ff (i.e. A9 and A8 are both 1). Note that one set of registers passes values from the network to the external user device and a second set passes values from the external user device to the network, see Figure 22. Writing to an address using command 0xF6 above will not cause the value to be read back with command 0xF7.

Command 0xF8	Set mailbox interrupt

Command Request Format

Byte Offset	Description
0	Command request code - always 0xF8
1-3	Dummy bytes (to round up to 4 byte boundary)

Response Format

Byte Offset	Description
0	Command request code - always 0xF8
1	Status byte. 0 for success, non-zero for failure
2-3	Dummy bytes (to round up to 4 byte boundary)

Description

The set mailbox interrupt command will generate an interrupt to the external device attached to GigEx. This can be used to inform the external device that some of the general comms registers have been changed and to trigger a read of these registers by the external device.

The nINT signal to the external device will be asserted if the mailbox interrupt enable bit is set in the mailbox control register (see section 5.2). The mailbox interrupt status bit will be set even if the interrupt enable bit is not set. The mailbox interrupt will be cleared when the external device reads the mailbox interrupt status register.

6.5 Embedded Web Server

GigEx contains an embedded web server to configure the GigEx device settings and report device status. By default, the web server uses HTML pages embedded in the flash on the ZestET2 module but it is possible to replace these pages with a set of user pages to customise the look of the configuration pages to suit the user application. The user pages can be uploaded using the CPanel application (see section 9.2) and will replace the entire set of default pages. User pages can consist of HTML, JavaScript and CSS pages along with JPEG, GIF and PNG images. Use of embedded JavaScript in the pages can provide dynamic content but host side scripting is not supported.

The built-in web server can be accessed with any web browser by navigating to the IP address of the GigEx device. By default, this will be at address http://192.168.1.100. If the web server port setting is to be changed from the default 80, then the new setting will have to be specified in the address. For example: http://192.168.1.100:99 for port 99.

If the IP address or HTTP Port of the GigEx is changed the web browser will no longer be able to display the configuration page. You must browse to the new address and port to see the updated configuration. When using AutoIP or DHCP, the new address may not be known in which case you should use the CPanel application to search for the module and the IP address of the GigEx will then be displayed.

If a password has been set for the board, you must log in to the GigEx device to change any of the settings. The user name is fixed as ‘admin’.

! **There is no way to recover a lost configuration password. If you lose the password, the only way to change the settings is to return the board to the factory.**

The web server uses a special macro expansion format to embed dynamic values into the HTML pages it serves. Macros take the form of **$***macroname***$** and will be expanded out into ASCII text as the HTML page is served. In this way, current setting values can be displayed in HTML pages without requiring host side scripting.

Values can be sent back to GigEx via HTML forms generating POST messages. For example:

```
<form id="config" method="post" action="network.html">
  <input type="text" name="IPADDR" value="$IPADDR$" /> <br />
  <input type="text" name="SUBNET" value="$SUBNET$" /> <br />
  <input type="text" name="GATEWAY" value="$GATEWAY$" /> <br />

  <input type="submit" name="Update" value="Update" class="button">
</form>
```

In this example, 3 macros are used: `$IPADDR$`, `$SUBNET$` and `$GATEWAY$`, each of which will expand out into the corresponding IP address (e.g. "0xc0a80164") when the page is served. Clicking on the Update button will submit the user modified values via a POST message which will cause the IP address settings to be changed in the GigEx flash.

POST message values will be checked by the web server to avoid programming illegal values into the GigEx flash. However, no feedback will be given by the web server if an incorrect value is specified so client side scripting should be used to check values and report meaningful errors to the user.

The full source HTML for the default web server is provided on the ZestET2 Support Software CD to further demonstrate how to develop user web pages.

The full list of macros is given below. Default settings are shown in section 6.6.

Macro	Example Expansion	Description
$SERIAL$	5123	Serial number of ZestET2 module
$VERSION$	3.01 - 3.02	Hardware version followed by software version. If either version has a suffix of 'F' then the GigEx is operating in fallback mode due to a failed firmware update.
$PRODUCTID$	1	Product ID that the GigEx is fitted to. A product ID of 1 indicates the board is a ZestET2
$PRODUCTSUBTYPE$	0	Product sub-type. The product subtype indicates which features the product has enabled. Currently always reads as 0.
$GUID$	0000000000000000000 0000000000000	16 byte GUID value in Hex. Contains the user Globally Unique ID programmed into the board
$MACADDR$	00-50-c2-a7-10-38	MAC address of GigEx consisting of 6 hexadecimal pairs of characters
$DOTTEDIPADDR$	192.168.1.100	IP address of GigEx in dotted notation. This is the current IP address of the device not the fixed IP address configuration. e.g. if the GigEx is using DHCP then this will be the address received from the DHCP server.
$IPADDR$	0xc0a80164	IP address of GigEx in hex notation. This is the current IP address of the device not the fixed IP address configuration. e.g. if the GigEx is using DHCP then this will be the address received from the DHCP server.
$SUBNET$	0xffffff00	Subnet mask of GigEx in hex notation. This is the current mask of the device not the fixed IP address configuration. e.g. if the GigEx is using DHCP then this will be the mask received from the DHCP server.
$GATEWAY$	0xc0a80101	IP address of gateway used by GigEx in hex notation. This is the current gateway IP address not the fixed IP address configuration. e.g. if the GigEx is using DHCP then this will be the gateway address received from the DHCP server.
$HTTPPORT$	80	Current port opened by the HTTP web server.
$CONTROLPORT$	20481	Current port opened by the control message processor in GigEx.
$JUMBO$	1500	Current maximum transmit frame size.
$PASSWORD$	Password	Current access password used by the GigEx web server.
$ENABLEPASS$	0	A value of 0 indicates that password protection on the web server is disabled. A value of 1 indicates that password protection on the web server is enabled.
$ENABLEDHCP$	0	A value of 0 indicates that the DHCP client in GigEx is disabled. If DHCP and AutoIP are both disabled then GigEx will used a fixed IP address.
$ENABLEAUTOIP$	0	A value of 0 indicates that the AutoIP client in GigEx is disabled. If DHCP and AutoIP are both disabled then GigEx will used a fixed IP address.
$ENABLEGDBSERVER$	1	A value of 0 indicates the GDB debug server for the user CPU in GigEx is disabled. A value of 1 indicates the GDB debug server is enabled.
$GDBSERVERPORT$	1000	Current port opened by the GDB debug server.
$INTERFACECLOCKDIR$	0	A value of 0 indicates that the user interface clock is an input to GigEx from the external device. A value of 1 indicates that GigEx is generating a 125MHz clock output to the external device.

$INTERFACEMODE$	0	Current mode of the user interface: 0: 16 bit SRAM register mode 1: 8 bit SRAM register mode 2: FIFO mode 3: Direct mode
$INTERFACERXWIDTH$	2	In direct mode, the number of bytes of the data bus passing data from the network to the external device.
$INTERFACETXWIDTH$	2	In direct mode, the number of bytes of the data bus passing data from the external device to the network.
$INTERFACESERIAL$	0	A value of 0 indicates that the low speed interface is operating as a UART. A value of 1 indicates that the low speed interface is operating as a slave SPI bus.
$INTERFACEBAUDRATE$	115200	Current baud rate setting of the UART.
$INTERFACEDATALENGTH$	8	Current number of data bits used by the UART.
$INTERFACEPARITY$	0	Current parity setting of the UART: 0: No parity 1: Odd parity 2: Even parity
$INTERFACESTOPBITS$	1	Current number of stop bits (1 or 2) used by the UART.
$INTERFACEFLOWCONTROL$	0	A value of 0 indicates that UART flow control is switched off. A value of 1 indicates that hardware CTS/RTS flow control is enabled.
$INTERFACEECHO$	0	A value of 0 indicates that echo is turned off on the UART. A value of 1 indicates that echo is turned on on the UART (i.e. each character received by the UART will be transmitted out of the UART).
$TRIGGERDIR$	0	A value of 0 indicates that the IEEE1588/PTP trigger/event pin is an input to GigEx for recording events. A value of 1 indicates that the pin is an output from GigEx to generate triggers or 1 pulse per second signals.
$TRIGGERTYPE$	0	A value of 0 indicates that the IEEE1588/PTP trigger output pin will be a single shot trigger. A value of 1 indicates that the IEEE1588/PTP trigger pin will be a 1 pulse per second output.
$TRIGGERPULSEWIDTH$	10	Current width of the IEEE1588/PTP trigger and 1 pulse per second output specified in units of 100 ns.
$AUTOCONNENABLE***x***$	0	A value of 0 indicates that auto-open is disabled on a channel. A value of 1 indicates that auto-open is enabled on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNTCP***x***$	0	A value of 0 indicates that the auto-open mode of a channel should be UDP. A value of 1 indicates that auto-open mode should be TCP. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNSERVER***x***$	0	A value of 0 indicates that a channel should be auto-opened as a TCP client (i.e. attempting to connect to a remote device). A value of 1 indicates that a channel should be auto-opened as a TCP server (i.e. waiting for a connection from a remote device). The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNLOCALPORT***x***$	1000	Local port to use on an auto-open operation on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNREMOTEPORT***x***$	2000	Remote port to use on an auto-open operation on a channel. The channel is specified by replacing the ***x***

		with a decimal value between 0 and 15.
$AUTOCONNREMOTEIP***x***$	0xc0a80120	Remote IP address (in hex format) to use on an auto-open operation on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNTTL***x***$	128	Value of time to live to use on an auto-open operation on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNMTU***x***$	128	Value of maximum transmit size to use on an auto-open operation on a channel. The value will be multiplied by 32 bytes and used to limit the transmit size of UDP data before fragmentation occurs on a datagram. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$AUTOCONNUART***x***$	0	A value of 1 indicates that a successful auto-open connection on a channel will connect the network data to the UART for immediate transmission/reception. See section 6.3.3 for details. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
$USERVALUE***x***$	12ab	16 bit hexadecimal value (with no prefix) read from one of the general purpose user comms registers from the external device. The address of the register is specified by replacing the ***x*** with a decimal value between 0 and 127.
$USERSETTINGINTERRUPT$	0	A value of 1 indicates that the user mailbox interrupt will be set when the one of the $USERSETTING***x***$ values is updated. A value of 0 indicates that the interrupt will not be set.
$USERSETTING***x***$	cd34	16 bit hexadecimal value (with no prefix) written to one of the general purpose user comms registers to the external device. The address of the register is specified by replacing the ***x*** with a decimal value between 0 and 127.
$GPIOSELECT*x*$	0	A value of 1 indicates that a GigEx pin is in GPIO mode, a value of 0 indicates the pin is in normal function mode (i.e. high speed or low speed interface mode). The pin is specified by the ***x*** which should be a decimal value from 0 to 23.
$GPIOENABLE*x*$	0	A value of 1 indicates that a GigEx GPIO pin is an output (when selected), a value of 0 indicates the pin is an input. The pin is specified by the ***x*** which should be a decimal value from 0 to 23.
$GPIOOUTVALUE*x*$	0	A value of 1 indicates that a GigEx GPIO pin is driven high (when selected and enabled), a value of 0 indicates the pin will be driven low (when selected and enabled). The pin is specified by the ***x*** which should be a decimal value from 0 to 23.
$GPIOINVALUE*x*$	0	A value of 1 indicates that a GigEx GPIO pin is currently high, a value of 0 indicates the pin is currently low. The pin is specified by the ***x*** which should be a decimal value from 0 to 23.

The full list of POST values is given below. Unless otherwise stated, numerical values can be hexadecimal (when prefixed by 0x), octal (when prefixed by 0) or decimal otherwise. Default settings are shown in section 6.6.

POST Value	**Example**	**Description**

GUID	00000000000000000000000000000000	16 byte GUID value in Hex. Contains the user Globally Unique ID to program into the board.
IPADDR	0xc0a80164	Fixed IP address of GigEx. For example, 0xc0a80164 = 192.168.1.100.
SUBNET	0xffffff00	Fixed sub net mask of GigEx. For example, 0xffffff00 = 255.255.255.0.
GATEWAY	0xc0a80101	Fixed IP address of gateway. For example, 0xc0a80101 = 192.168.1.1.
HTTPPORT	80	Port to be opened by the embedded HTTP web server. This must be a value between 1 and 65534 inclusive.
CONTROLPORT	20481	Port to be opened by the control message processor in GigEx. This must be a value between 1 and 65534 inclusive.
JUMBO	1500	Maximum transmit frame size. This must be a value between 1500 and 7500 bytes inclusive and must be divisible by 4.
ENABLEDHCP	0	A value to enable or disable the DHCP client in GigEx. A value of 0 indicates that the DHCP client in GigEx should be disabled. If DHCP and AutoIP are both disabled then GigEx will used a fixed IP address.
ENABLEAUTOIP	0	A value to enable or disable the AutoIP client in GigEx. A value of 0 indicates that the AutoIP client in GigEx should be disabled. If DHCP and AutoIP are both disabled then GigEx will use a fixed IP address.
ENABLEGDBSERVER	1	A value to enable or disable the GDB debug server for the user CPU in GigEx. A value of 0 indicates that the GDB server in GigEx should be disabled.
GDBSERVERPORT	1000	Port to be opened by the GDB debug server for the user CPU. This must be a value between 1 and 65534 inclusive.
ENABLEPASS	0	A value to enable or disable password protection on the GigEx web server. A value of 0 indicates that password protection on the web server should be disabled. A value of 1 indicates that password protection on the web server should be enabled.
PASSWORD	password	Access password to be used by the GigEx web server. Should be a string of up to 14 characters.
INTERFACECLOCKDIR	0	A value to control the direction of the user interface clock. A value of 0 indicates that the user interface clock will be an input to GigEx from the external device. A value of 1 indicates that GigEx will generate a 125MHz clock output to the external device.
INTERFACEMODE	0	Value to control the mode of the GigEx user interface: 0: 16 bit SRAM register mode 1: 8 bit SRAM register mode 2: FIFO mode 3: Direct mode
INTERFACERXWIDTH	2	A value to control the number of bytes of the data bus passing data from the network to the external device when the user interface is in direct mode.
INTERFACETXWIDTH	2	A value to control the number of bytes of the data bus passing data from the external device to the network when the user interface is in direct mode.
INTERFACESERIAL	0	A value to control the mode of the low speed user interface. A value of 0 indicates that the low speed interface should operate as a UART. A value of 1 indicates that the low speed interface should operate as a slave SPI bus.
INTERFACEBAUDRATE	115200	Value to control the baud rate setting of the UART. Valid values are 9600, 19200, 38400, 57600, 115200,

		230400, 460800 and 921600.
INTERFACEDATALENGTH	8	A value indicating the number of data bits to be used by the UART. Valid values are 7 and 8.
INTERFACEPARITY	0	A value indicating the number and polarity to use for the parity bits of the UART. Valid values are: 0: No parity 1: Odd parity 2: Even parity
INTERFACESTOPBITS	1	A value indicating the number of stop bits to be used by the UART. Valid values are 1 and 2.
INTERFACEFLOWCONTROL	0	A value used to enable or disable the hardware flow control of the UART. A value of 0 indicates that UART flow control should be switched off. A value of 1 indicates that hardware CTS/RTS flow control should be enabled.
INTERFACEECHO	0	A value to enable or disable character echo on the UART. A value of 1 indicates that echo should be enabled on the UART (i.e. each character received by the UART will be transmitted out of the UART).
TRIGGERDIR	0	A value to control the direction of the IEEE1588/PTP trigger/event pin on GigEx. A value of 0 indicates that the IEEE1588/PTP trigger/event pin will be an input to GigEx for recording events. A value of 1 indicates that the pin will be an output from GigEx to generate triggers or 1 pulse per second signals.
TRIGGERTYPE	0	A value to control the mode of the IEEE1588/PTP trigger/event pin on GigEx. A value of 0 indicates that the IEEE1588/PTP trigger output pin will be a single shot trigger. A value of 1 indicates that the IEEE1588/PTP trigger pin will be a 1 pulse per second output.
TRIGGERPULSEWIDTH	10	A value controlling the width of the trigger and 1 pulse per second output signal. The value is specified in units of 100 ns with a valid range of 1 to 65535 (i.e. 100 ns to 6.5 ms).
AUTOCONNENABLE***x***	0	Value to enable or disable auto connection management on a channel. A value of 0 indicates that auto-open will be disabled on a channel. A value of 1 indicates that auto-open will be enabled on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNTCP***x***	0	Value to control whether a channel is auto-opened in TCP or UDP mode. A value of 0 indicates that the auto-open mode of a channel should be UDP. A value of 1 indicates that auto-open mode should be TCP. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNSERVER***x***	0	Value to control whether a channel is auto-opened in TCP client or server mode. A value of 0 indicates that a channel should be auto-opened as a TCP client (i.e. attempting to connect to a remote device). A value of 1 indicates that a channel should be auto-opened as a TCP server (i.e. waiting for a connection from a remote device). The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNLOCALPORT***x***	1000	Value to control the local port value for an auto-opened channel. Valid values for the port are 0 to 65535. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNREMOTEPORT***x***	2000	Value to control the remote port to use on an auto-open operation on a channel. Valid values for the port

		are 0 to 65535. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNREMOTEIP***x***	0xc0a80102	Remote IP address to use on an auto-open operation on a channel. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15. For example, 0xc0a80102 = 192.168.1.2.
AUTOCONNTTL***x***	128	Value to control the time to live to use on an auto-open operation on a channel. Valid values are 1 to 255. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNMTU***x***	128	Value to control the maximum transmit size to use on an auto-open operation on a channel. The value will be multiplied by 32 bytes and used to limit the transmit size of UDP data before fragmentation occurs on a datagram. Valid values are 1 to 255. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
AUTOCONNUART***x***	0	Value to enable connection of an auto-open channel to the UART for data transmission. A value of 1 indicates that a successful auto-open connection on a channel will connect the network data to the UART for immediate transmission/reception. See section 6.3.3 for details. The channel is specified by replacing the ***x*** with a decimal value between 0 and 15.
USERSETTINGINTERRUPT	0	Value to control whether to assert the interrupt to the user interface when the POST message has been processed. A value of 1 indicates that the user mailbox interrupt will be set. A value of 0 indicates that the interrupt will not be set.
USERSETTING***x***	cd34	16 bit hexadecimal value (with no prefix) to be written to one of the general purpose user comms registers to the external device. Valid values are 0000 to ffff. The address of the register is specified by replacing the ***x*** with a decimal value between 0 and 127.
GPIOSELECT***x***	0	Set to 1 to put a GigEx pin in GPIO mode, set to 0 to switch the pin into normal function mode (i.e. high speed or low speed interface mode). The pin is specified by the ***x*** which should be a decimal value from 0 to 23.
GPIOENABLE***x***	0	Set to 1 to make a GigEx GPIO pin an output, set to 0 to make the pin an input. The pin is specified by the ***x*** which should be a decimal value from 0 to 23.
GPIOOUTVALUE***x***	0	Set to 1 to drive a GigEx GPIO pin high (when the pin is in GPIO mode and enabled), set to 0 to drive the pin low. The pin is specified by the ***x*** which should be a decimal value from 0 to 23.

6.6 Configuring Settings

The GigEx device has various settings that are stored in non-volatile memory. These settings can be programmed using AT commands over the UART (see section 6.3.1), via the built-in web server (see section 6.5), over the network via the control port (see section 6.4) or via the user register map (see section 5.2). The settings are as follows:

Setting	Default	UART	HTTP	Cont Port	Regs	Description
GUID	0000000000000 0000000000000		●	●		User defined globally unique identifier.

	000000					
Fixed IP	192.168.1.100	●	●	●	●	Fixed IP address of the GigEx device. Used when AutoIP and DHCP are disabled.
Subnet	255.255.255.0	●	●	●	●	Fixed subnet setting. Used when AutoIP and DHCP are disabled.
Gateway	192.168.1.1	●	●	●	●	Fixed gateway setting. Used when AutoIP and DHCP are disabled.
DHCP	Disabled	●	●	●	●	Enable DHCP client to acquire network address automatically. Disables fixed IP settings.
AutoIP	Disabled	●	●	●	●	Enable AutoIP client to acquire network address automatically. Disables fixed IP settings. If DHCP and AutoIP are both enabled, DHCP will be attempted first before falling back to AutoIP configuration if no DHCP server is present.
Web server port	80	●	●	●	●	Port for configuration web server. Changing this allows the user to implement their own web server on the default HTTP port of 80.
Control port	0x5001	●	●	●	●	Port used for control messages (e.g. writing to master SPI port or writing to the user comms registers). Changing this allows the user to use the default 0x5001 port for their own application.
Jumbo frames	1500	●	●		●	Length of maximum transmit frame in bytes. Should be between 1500 and 7500 bytes and should be a multiple of 4 bytes.
Enable Password	Off		●			Enables password protection of the configuration settings.
Password			●			Setting a password protects the configuration options.
High Speed Interface Clock Direction	Input	●	●			Direction of the high speed user interface clock pin. Can be an input from the external device or a fixed 125MHz clock output.
High Speed Interface Mode	16 Bit SRAM	●	●			Mode of the high speed user interface. Can be 8 or 16 bit SRAM register based, FIFO based or direct data transfer.
Direct mode receive width	2 bytes	●	●			Number of pins of the direct mode bus that transfer data from the network to the external device.

Direct mode transmit width	2 bytes	●	●			Number of pins of the direct mode bus that transfer data from the external device to the network.
Low speed interface mode	Slave SPI		●			Mode of the low speed serial interface. Can be slave SPI or UART.
UART baud rate	115200	●	●			UART baud rate setting.
UART data bits	8	●	●			Number of bits in a byte on the UART interface.
UART stop bits	1	●	●			Number of stop bits for the UART interface.
UART parity bits	0	●	●			Number and type of parity to use for the UART interface.
UART flow control	Disabled	●	●			Enable/disable hardware RTS/CTS flow control on the UART interface.
UART character echo	Enabled	●	●			Enable/disable character echoing on the UART interface.
IEEE1588/PTP trigger pin direction	Output	●	●			Direction setting for the IEEE1588/PTP interface pin. Can be set to a trigger output or an event input pin.
IEEE1588/PTP trigger mode	Single shot	●	●		●	Mode of operation of the IEEE1588/PTP trigger pin. Can be a single shot trigger output or a 1 pulse per second output.
IEEE1588/PTP trigger pulse width	10*100ns	●	●		●	Width of the IEEE1588/PTP trigger output in units of 100 ns.
Auto-open	Disabled	●	●			Auto-open settings for each of the 16 high speed network channels.
User general purpose comms registers	0	●	●	●	●	Values for the 256 bytes of general purpose user settings registers.
GPIO pins	All pins selected as normal function (i.e. high speed or low speed interface mode)		●	●		Mode and state of GigEx GPIO pins.

7 User CPU

The GigEx device fitted to the ZestET2 contains a CPU which is free for the user to add their own functionality. The CPU does not have to be programmed for the GigEx to operate but it can be used to implement higher level protocols on top of TCP and UDP which would otherwise be complex or require too many resources to implement in the FPGA. For example, the CPU could be used to implement RTP (real-time transport protocol), SMB (server message block or file sharing) or GigE Vision protocols where the CPU handles the complex setup, status and control messages and the FPGA concentrates on transferring the high speed data.

The CPU has access to the same register set as the FPGA plus some additional CPU only registers. It can communicate with the FPGA via the GPIO pins and the user general purpose comms registers and can transfer data to and from the network via the TCP/IP offload engine. A block diagram of the CPU interfaces is shown in Figure 24 below.

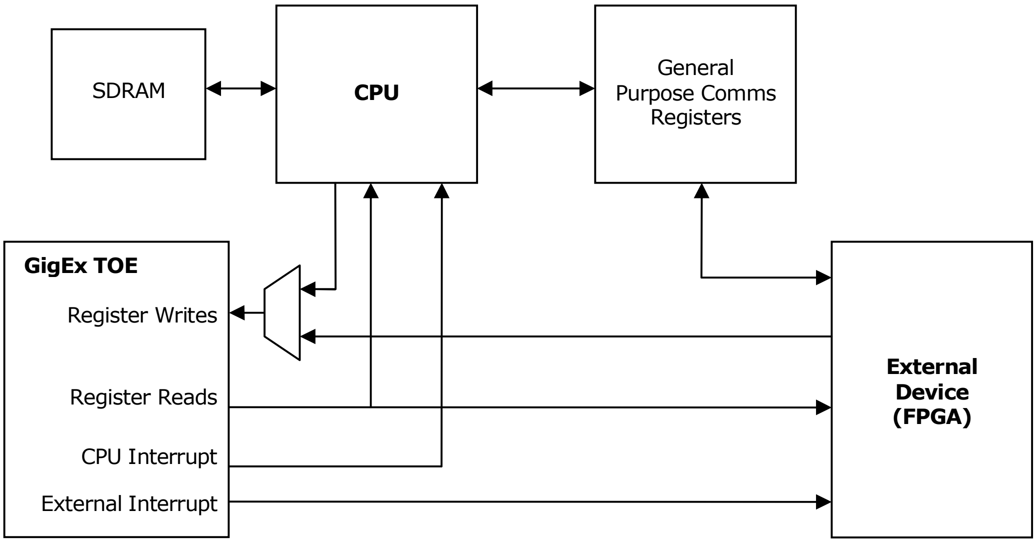


Figure 24. Diagram showing links between user CPU, GigEx and External FPGA

GigEx includes a built in GDB server which allows programming and debugging of the CPU via the network link and a standard Eclipse and GCC development environment is supplied on the installation CD.

7.1 CPU Overview

The user CPU is a 32 bit RISC processor based on the SPARC v8 architecture. It is clocked at 66MHz and has 8MBytes of SDRAM and 256kbytes of program flash memory dedicated to it. Full details of the SPARC architecture can be found in the SPARCv8.pdf manual in the Documents folder of the installation CD which should be read in conjunction with this manual. Specific implementation details of the CPU architecture are given in this section.

7.1.1 Processor State Register (PSR)

Both the *impl* and *ver* fields of the PSR are fixed at 0.

The CPU does not implement a floating point or coprocessor unit so the *EF* and *EC* fields have no effect. None of the floating point or coprocessor instructions can be executed.

The CPU has one interrupt source which has a priority level of 1. Therefore, interrupts can be globally enabled by setting the *PIL* field of the PSR to 0 and disabled by setting *PIL* to 1 or above.

7.1.2 Traps

The CPU Trap Base Register (TBR) is fixed at address 0 so the trap table must always be located at this address. The CPU implements the following traps as described in the SPARC reference manual:

Trap	*tt* value	Description
reset	0x00	Reset vector. The processor jumps to this trap on power up.
window_overflow	0x05	A SAVE instruction caused current window pointer (CWP) to point to an invalid window in the window invalid mask (WIM) register.
window_underflow	0x06	A RESTORE or RETT instruction caused CWP to point to an invalid window in the WIM register.
trap_instruction	0x80-0xff	Software trap T*icc* instruction executed.
interrupt_level_1	0x11	External interrupt received

No other traps are implemented. There is no error state so executing a trap with ET=0 will not cause the CPU to halt.

7.1.3 Unimplemented Instructions

The CPU implements all instructions described in the SPARC reference manual except SWAP and SWAPA. Since there is no floating point unit or co-processor unit, the FP and CP instructions are also unimplemented.

7.1.4 Register Windows

The CPU implements 8 register windows giving a total of 8*16+7 = 135 user registers.

7.1.5 Instruction Timing

Assuming all instructions are in the cache, all memory operations target data in the cache and that there are no pipeline conflicts, all instructions operate in a single clock cycle except the following:

Instruction	Clock cycles	Description
CALL, Taken B*icc,* JMPL	5	Any taken branch
UDIV, SDIV	36	Division of 64 bit by 32 bit number with 32 bit result
LDD	2	Double word load
LDSTUB	2	Load/Store atomic
STD [rs1+imm]	2	Double work store to constant offset address
STx [rs1+rs2]	2	Non-double word store to register offset address
STD [rs1+rs2]	3	Double word store to register offset address
T*icc*	6	Taken software trap

The CPU implements full register forwarding between all pipeline stages so no bubbles or pipeline stalls are caused by use of registers in consecutive cycles.

7.2 CPU Address Map

The user CPU has access to 8MBytes of SDRAM and 256kBytes of program flash. The flash can only be used to store a boot program which will be loaded on power up. The flash can be programmed using the CPanel application included in the installation CD. Small amounts of non-volatile data can be stored in the user general purpose comms registers (see section 5.2).

The SDRAM is located in the bottom 8 MBytes of the address space.

CPU Address	Register	Description
0x00000000- 0x007fffff	SDRAM memory	8 MBytes of dedicated SDRAM memory. The trap table must appear at address 0 which is the reset address of the CPU.

The CPU also has access to a set of control and status registers. These include access to the complete set of registers available to the external interface and some additional CPU only registers. Unless otherwise stated, all regsters are 16 bits wide and can be accessed as 8 or 16 bit values (when in 16 bit SRAM mode) or 8 bit values (when in other modes). The registers are big endian so bits 15:8 appear at A0=0 and bits 7:0 appear at A0=1. It is not possible to read/write two registers with a single 32 bit access. The registers common with the external interface appear between addresses 0x80000000 and 0x800003ff as shown below. Refer to section 5 for details of these registers.

CPU Address	Register	Description
0x80000000	Channel 0 local port register	
0x80000002	Channel 0 connection IP address register (high 16 bits)	
0x80000004	Channel 0 connection IP address register (low 16 bits)	
0x80000006	Channel 0 remote port register	
0x80000008	Channel 0 payload size and time to live register	Bits 15-8: Maximum payload size for UDP send in 32 byte units Bits 7-0: Time to live
0x8000000a	Channel 0 interrupt enable (write) and status (read)	Bit 15-4: Reserved. Must be set to 0. Bit 3: State change interrupt Bit 2: To network buffer not full interrupt Bit 1: To network buffer empty interrupt Bit 0: From network data available interrupt
0x8000000c	Channel 0 connection control and status	Bit 15-7: Reserved. Must be set to 0. Bit 6: 1 – Pause transmission, 0 – Enable transmission Bit 5: 1 - Enable channel, 0 - Reset channel Bit 4: 1 - TCP connection, 0 - UDP connection Bit 3-0: Request state change (write), Current state (read) States are: 0: CLOSED 1: LISTEN (TCP server mode) 2: CONNECT (TCP client mode) 3: ESTABLISHED
0x8000000e	Channel 0 from network frame length (read)	

	Channel 0 to network datagram length (write) (UDP only)	
0x80000010	Channel 0 data to/from network connection FIFO	
0x80000020-0x8000003f	Channel 1 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000040-0x8000005f	Channel 2 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000060-0x8000007f	Channel 3 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000080-0x8000009f	Channel 4 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800000a0-0x800000bf	Channel 5 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800000c0-0x800000df	Channel 6 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800000e0-0x800000ff	Channel 7 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000100-0x8000011f	Channel 8 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000120-0x8000013f	Channel 9 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000140-0x8000015f	Channel 10 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000160-0x8000017f	Channel 11 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000180-0x8000019f	Channel 12 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800001a0-0x800001bf	Channel 13 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800001c0-0x800001df	Channel 14 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x800001e0-0x800001ff	Channel 15 connection control registers	Register map is the same as for channel 0 from 0x80000000-0x8000001f
0x80000200	Local IP address (high 16 bits)	
0x80000202	Local IP address (low 16 bits)	
0x80000204	Local subnet mask (high 16 bits)	
0x80000206	Local subnet mask (low 16 bits)	
0x80000208	DCHP/AutoIP settings	Bit 0: Enable AutoIP Bit 1: Enable DHCP
0x8000020c	Gateway IP address (high 16 bits)	
0x8000020e	Gateway IP address (low 16 bits)	
0x80000210	Jumbo frame maximum length in bytes	
0x80000212	GigEx network control local port number	
0x80000214	GigEx web server local port number	
0x80000216	Update network settings register	Writes: Bit 0: Write 1 to this bit to initiate update of network settings Reads: Bit 0: When read as 1 the network settings are being updated Bit 1: When read as 1 the update failed

0x8000021e	Link status register	Bit 4: Link Status (1 - connected, 0 - not connected) Bit 3: Duplex (1 - half duplex, 0 - full duplex) Bit 2: 1Gbps Bit 1: 100Mbps Bit 0: 10Mbps
0x80000220	PTP time register	Write to this address to latch the current PTP time. Read from this address to shift out next bits of the latched PTP time.
0x80000222	Event time register	Write to this address to latch the latest event time. Read from this address to shift out next bits of the latched event time.
0x80000224	Trigger/1PPS pulse width in units of 100ns	
0x8000022a	Trigger/Event control and status	Writes: Bit 7: Event interrupt enable Bit 0: Trigger output control - 0 for single shot, 1 for one pulse per second Reads: Bit 7: Event interrupt status. Read clears interrupt condition.
0x8000022c	Trigger time most significant word (Bits 63:48)	
0x8000022e	Trigger time (Bits 47:32)	
0x80000230	Trigger time (Bits 31:16)	
0x80000232	Trigger time least significant word (Bits 15:0)	
0x80000240	GigEx hardware version	Read from this register to find the version of GigEx hardware fitted to the board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312. When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.
0x80000242	GigEx software version	Read from this register to find the version of GigEx software running on the board. The version is multiplied by 100. For example, version 3.12 will read as decimal 312. When the top bit of the version is set (bit 15), GigEx is operating in a fallback mode due to a failed flash firmware update.
0x800002e0	User settings flash control/status register	Writes: Bit 0: Write 1 to this bit to initiate copy of user settings registers to flash Reads: Bit 0: When read as 1 the copy to flash is in progress
0x800002ea	Mailbox interrupt enable/status register	Writes: Bit 6: Write 1 to enable mailbox interrupts Reads: Bit 6: Mailbox interrupt status. Read clears interrupt condition.
0x80000300- 0x800003ff	General purpose user comms registers	

The following registers sections describe the registers that are only accessible from the CPU. Details of the registers are given following the address map table below.

CPU Address	Register	Description
0x8000800a 0x8000802a 0x8000804a 0x8000806a 0x8000808a 0x800080aa 0x800080ca 0x800080ea 0x8000810a 0x8000812a 0x8000814a 0x8000816a 0x8000818a 0x800081aa 0x800081ca 0x800081ea	CPU channel 0 interrupt enable CPU channel 1 interrupt enable CPU channel 2 interrupt enable CPU channel 3 interrupt enable CPU channel 4 interrupt enable CPU channel 5 interrupt enable CPU channel 6 interrupt enable CPU channel 7 interrupt enable CPU channel 8 interrupt enable CPU channel 9 interrupt enable CPU channel 10 interrupt enable CPU channel 11 interrupt enable CPU channel 12 interrupt enable CPU channel 13 interrupt enable CPU channel 14 interrupt enable CPU channel 15 interrupt enable	Mirror of interrupt enable registers at addresses 0x80000XXa but these registers enable interrupts to the CPU rather than to the external interrupt pin connected to the FPGA.
0x8000822a	CPU PTP event interrupt enable	Mirror of the PTP event interrupt enable register at address 0x8000022a but this register enables interrupts to the CPU rather than to the external interrupt pin connected to the FPGA.
0x800082ea	CPU mailbox interrupt enable	Mirror of the mailbox interrupt enable register at address 0x800002ea but this register enables interrupts to the CPU rather than to the external interrupt pin connected to the FPGA.
0x800082ee	Mailbox from external interface control/status	Writes: Bit 0: Writing a 1 to bit 0 will enable interrupts from the external interface to the CPU Reads: Bit 0: A 1 indicates that the external interface interrupted the CPU by writing to the Mailbox to user CPU register. Reading from this register will clear the interrupt status.
0x800082f2	Mailbox to external interface	Writing a 1 to bit 0 will generate a mailbox interrupt to the external interface
0x80008300- 0x800083ff	General purpose user comms registers	Writing to this address range writes to the general purpose user comms registers which can be read from the external interface (or by the CPU at address 0x80000300-0x800003ff). Reading from this address range reads from the general purpose user comms registers which can be written by the external interface (or by the CPU at address 0x80000300-0x800003ff).
0x8000c000	Interval timer value/limit	32 bit register Reads: Reads from this register return a cycle counter clocked at the CPU rate. The counter wraps around when it hits the limit value. Writes: Writes to this address set the limit value for the clock counter. The counter will wrap around when it hits this value. The CPU rate is approximately 66MHz but if precise

		timing is required the application should use the PTP timer register which is in units of nanoseconds.
0x8000c004	Interval timer interrupt control/status	32 bit register Writes: Writing a 1 to bit 0 of this register will enable an interrupt when the interval counter has a value of 0. Reads: Reading a 1 from bit 0 of this register indicates that the interval timer has generated an interrupt. Reading from this register will clear the interrupt state.
0x8000c008	GPIO output enable and value register (GPIO[23:16])	Writing to this register controls whether a GPIO pin is operating as an output or input and the value being driven onto the pin when it's an output. Reading from this register returns the current output enable and output value states. This register controls GPIO pins 23:16.
0x8000c00c	GPIO output enable and value register (GPIO[15:0])	Writing to this register controls whether a GPIO pin is operating as an output or input and the value being driven onto the pin when its an output. Reading from this register returns the current output enable and output value states. This register controls GPIO pins 15:0.
0x8000c010	GPIO pin function select	Writing a 1 to a bit in this register sets a pin as GPIO. Writing a 0 to a bit in this register sets a pin as having its normal function. Reading from this register returns whether the pin is a GPIO or normal pin. Bit 0 (LSB) controls GPIO0 up to bit 23 which controls GPIO23
0x8000c014	GPIO input value and interrupt enable	Reading from this register returns the current state of all the GPIO pins. Writing 1 to bit 0 of this register enables the GPIO interrupt from GPIO16. When a 1 is present on GPIO16 and the enable bit is set the CPU will be interrupted.

Details of the CPU only registers are given below.

CPU Address	Register
0x8000800a	CPU channel 0 interrupt enable
0x8000802a	CPU channel 1 interrupt enable
0x8000804a	CPU channel 2 interrupt enable
0x8000806a	CPU channel 3 interrupt enable
0x8000808a	CPU channel 4 interrupt enable
0x800080aa	CPU channel 5 interrupt enable
0x800080ca	CPU channel 6 interrupt enable
0x800080ea	CPU channel 7 interrupt enable
0x8000810a	CPU channel 8 interrupt enable
0x8000812a	CPU channel 9 interrupt enable
0x8000814a	CPU channel 10 interrupt enable
0x8000816a	CPU channel 11 interrupt enable
0x8000818a	CPU channel 12 interrupt enable
0x800081aa	CPU channel 13 interrupt enable
0x800081ca	CPU channel 14 interrupt enable
0x800081ea	CPU channel 15 interrupt enable

The CPU channel interrupt enable registers are used to route network interrupts to the user CPU for handling. They are similar to the equivalent external interface interrupt enable registers and have the following bits:

Bit 15-4: Reserved. Must be set to 0 when writing.
Bit 3: Connection state change
Bit 2: Data buffer to network not full
Bit 1: Data buffer to network empty
Bit 0: Data buffer from network not empty

When a 1 is written to any of these bits, the CPU will receive an interrupt when the interrupt condition is true. Writing a 0 will disable the corresponding interrupt to the CPU.

The source of the interrupt can be determined by reading the appropriate Interrupt Enable and Status register from address 0x80000xxa.

Note that an interrupt condition can generate an interrupt to the CPU or to the external interface or to both destinations but care must be taken when designing the interrupt handler in the CPU and FPGA to avoid conflicts.

CPU Address	Register
0x8000822a	CPU PTP event interrupt enable

GigEx can generate an interrupt to the CPU when it detects an event on the TrigEvent pin when in Event input mode. Writing to bit 7 of the CPU PTP event interrupt enable register enables or disables this interrupt generation. The status of the interrupt condition can be checked by reading bit 7 of the trigger/event control and status register at address 0x8000022a which has the side effect of clearing the interrupt.

This register is equivalent to the interrupt enable bit of the Trigger/Event Control and Status Register but it enables the interrupt on the CPU rather than on the external interface to the FPGA.

Note that an interrupt condition can generate an interrupt to the CPU or to the external interface or to both destinations but care must be taken when designing the interrupt handler in the CPU and FPGA to avoid conflicts.

CPU Address	Register
0x800082ea	CPU mailbox interrupt enable

The CPU mailbox interrupt enable register provides a mechanism for a remote device on the network to generate an interrupt to the CPU. This could be used, for example, to inform the CPU that the remote device has updated some of the General Purpose User Comms Register values and that they should be re-read by the CPU.

Writing to the CPU mailbox interrupt enable register:
Bit 6: When 1, the mailbox interrupt enable will be set and the CPU will receive an interrupt when the remote device sets the mailbox. When 0, the CPU will not receive the interrupt.

The state of the interrupt can be read from the Mailbox Interrupt Enable/Status Register at address 0x800002ea.

This register is equivalent to the interrupt enable bit of the Mailbox Interrupt Enable/Status Register but it enables the interrupt on the CPU rather than on the external interface to the FPGA.

Note that an interrupt condition can generate an interrupt to the CPU or to the external interface or to both destinations but care must be taken when designing the interrupt handler in the CPU and FPGA to avoid conflicts.

CPU Address	Register
0x800082ee	Mailbox from External Interface Control/Status

GigEx provides a mechanism for the external device (FPGA) to interrupt the CPU. This could be used, for example, to inform the CPU that the external device has updated some of the General Purpose User Comms Register values and that they should be re-read by the CPU.

Writing to the Mailbox from External Interface Control/Status register:
Bit 0: When 1, the mailbox interrupt enable will be set and the CPU will receive an interrupt when the external device sets the mailbox. When 0, the CPU will not receive the interrupt.

Reading from the Mailbox from External Interface Control/Status register:
Bit 0: When 1, the external device has set the mailbox. Reading from this register has the side effect of clearing the interrupt condition.

The external device can set the mailbox interrupt to the CPU by writing a 1 to bit 0 of the Mailbox to User CPU Register.

CPU Address	Register
0x800082f2	Mailbox to External Interface

GigEx provides a mechanism for the CPU to interrupt the external device (FPGA). This is the same mailbox as used to interrupt the external device from a remote network device and the status can be read by the external device via the Mailbox Interrupt Enable/Status Register.

Writing to the Mailbox to External Interface register:
Bit 0: When 1, the mailbox interrupt condition will be set and an interrupt will be generated to the external device. The external device can clear the interrupt condition by reading the Mailbox Interrupt Enable/Status Register.

CPU Address	Register
0x80008300-0x800083ff	General purpose user comms registers

GigEx provides a set of registers that can be used to transfer data between GigEx and the external device (FPGA). There are two sets of registers - one set for communication from GigEx to the external device and one set for communication from the external device to GigEx. Each set consists of 256 bytes of data. See Figure 22 for a diagram of the general purpose user comms registers.

The User CPU is able to read and write both sides of both sets of registers. In this way it can send and receive data to/from the external device (FPGA) or send/receive data to/from a remote network device.

The view of the general purpose user comms registers at address 0x80000300-0x800003ff is the same as the view seen by the external device. The view of the general purpose user comms registers seen at address 0x80008300-0x800083ff is the other side of the registers. For example, if the CPU writes to the register at address 0x80008300, the FPGA can read the same value at address 0x300. The views of the registers are shown in Figure 25.

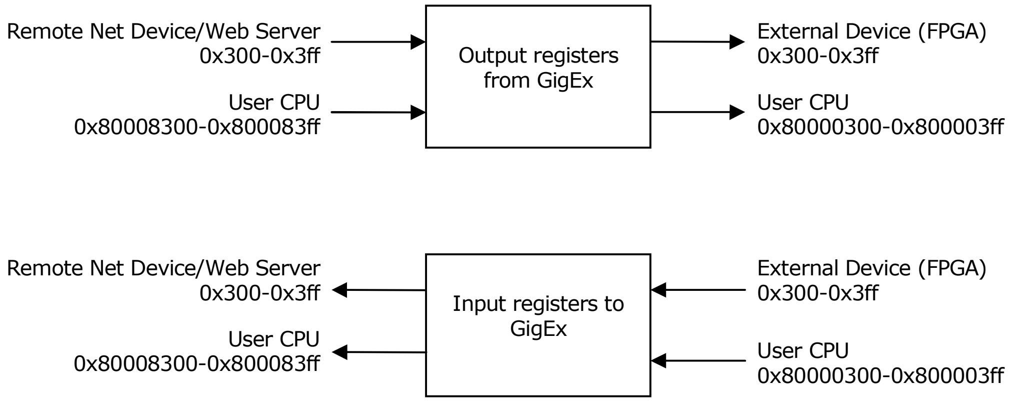


Figure 25. Addresses of General Purpose User Comms Registers from CPU, GigEx and FPGA

CPU Address	Register
0x8000c000	Interval timer value/limit
0x8000c004	Interval timer interrupt control/status

The CPU in GigEx has access to a programmable interval timer which can generate an interrupt to the CPU at fixed intervals. This can be useful for generating tick counters, timeouts and other time related functions within the CPU. The interval timer is controlled by two 32 bit registers.

Writing to the Interval timer value/limit register:

> Writing to the interval limit register sets the value at which the interval timer wraps around to 0. When the timer wraps round to zero, an interrupt condition to the CPU is set. If the interrupt enable register is set the CPU will receive an interrupt. Setting the limit value controls the period of the interrupts to the CPU.

Reading from the Interval timer value register:

> Reading from the Interval timer register will return the current count value of the timer. The timer counter increments on every CPU clock cycle up to the limit value at which point it wraps round to zero.

Writing to the Interval timer interrupt control/status register:

> Writing a 1 to bit 0 of the Interval timer interrupt control/status will enable the timer interrupt to the CPU. When the timer wraps round to zero and the interrupt is enabled, the CPU will receive an interrupt.

Reading from the Interval timer interrupt control/status register:

> Bit 0 of this register will be 1 when the timer interrupt condition is set. Reading from this register has the side effect of clearing the interrupt condition.

CPU Address	Register
0x8000c008	GPIO output enable and value register high (GPIO[23:16])
0x8000c00c	GPIO output enable and value register low (GPIO[15:0])
0x8000c010	GPIO pin function select
0x8000c014	GPIO input value and interrupt enable

GigEx can switch 24 of its pins into a general purpose input/output (GPIO) mode. In this mode each pin can be set as an input or as an output with a specified value. In some modes, there are spare pins in the user interface and when a pin is not needed for the High speed or Low speed interfaces it can be used as a direct signal between the FPGA and the user CPU or remote network device. These registers allow the user CPU to control the state of the GPIO pins.

The GPIO pin number mapping is shown in section 8.4.

The GPIO control registers are all 32 bits wide. The bits of the GPIO output enable and value register high register are shown below where E is the output enable bit and D is the data output bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																GP23		**GP22**		**GP21**		**GP20**		**GP19**		**GP18**		**GP17**		**GP16**	
																E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D

The bits of the GPIO output enable and value register low register are shown below where E is the output enable bit and D is the data output bit.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15		**GP14**		**GP13**		**GP12**		**GP11**		**GP10**		**GP9**		**GP8**		**GP7**		**GP6**		**GP5**		**GP4**		**GP3**		**GP2**		**GP1**		**GP0**	
E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D	E	D

Writing to the GPIO output enable and value registers:

> Writing a 0 to the Enable (E) bit sets a GPIO pin as an output. Writing a 1 to the enable bit sets the pin as an input.
>
> Writing a 1 to the Data (D) bit sets the GPIO output pin high. Writing a 0 to the data bit sets the pin low.

Reading from the GPIO output enable and value registers reads the current value of the register.

Writing to the GPIO pin function select register:

> Writing a 1 to bit n of the GPIO pin function select register sets the corresponding pin as a GPIO pin. Writing a 0 to the bit sets the pin as having its normal function (i.e. high or low speed interface pin).

Reading from the GPIO pin function select register reads the current value of the register.

Writing to the GPIO input value and interrupt enable register:

> Writing a 1 to bit 0 of the GPIO interrupt enable register enables a high value on GPIO16 to generate an interrupt to the user CPU. GPIO16 must be set in GPIO mode (function select bit high) and set as an input (E bit high). The interrupt condition will be cleared when the GPIO pin value is taken low by the external device. This allows an external device to directly interrupt the user CPU.

Reading from the GPIO input value and interrupt enable register:

Reading from the GPIO input value register returns the current state of the GPIO pins. When set as an input, bit n of the register will be 1 when the GPIOn pin is high and 0 when the GPIOn pin is low.

7.2.1 CPU Interrupt Handling

The CPU has a single interrupt connected at priority level 1. This is an OR of the various interrupt sources and the exact source must be deduced in the ISR from the various interrupt status registers. The CPU has its own set of interrupt enable registers so that interrupts can be routed either to the external interface (FPGA) or to the CPU depending on which device is expected to handle the cause.

A complete list of interrupt sources is as follows:

Network engine interrupts.

Each of the 16 channels can generate 4 different interrupts:

Data received from network
Transmit FIFO to network not full
Transmit FIFO to network empty
Connection state change

Each of these interrupts has an enable bit to route it to the external device (FPGA). This enable bit can be written by the external device (A4:1 set to 0101) or the user CPU (at address 0x80000nna where nn represents the channel number).

Each of these interrupts has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x80008nna).

PTP event interrupt

The PTP engine can generate an interrupt when it captures an event on the external event input pin.

This interrupt has an enable bit to route it to the external device (FPGA). This enable bit can be written by the external device (A9:1 set to 100010101) or the user CPU (at address 0x8000022a).

This interrupt has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x8000822a).

Mailbox to External Device interrupt

There is a mailbox interrupt that can be generated to the external device. This interrupt can be set by the web server (see section 6.5), by a remote network device (see section 6.4), or by the user CPU by writing to 0x800082f2.

This interrupt has an enable bit to route it to the external device (FPGA). This enable bit can be written by the external device (A9:1 set to 101110101) or the user CPU (at address 0x800002ea).

This interrupt has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x800082ea).

Mailbox from External Device interrupt

There is a mailbox interrupt that can be generated from the external device. This interrupt can be written by the external device (A9:1 set to 101110110).

This interrupt has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x800082ee).

Interval Timer interrupt

There is an interrupt that can be generated at a fixed interval.
This interrupt has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x8000c004).

GPIO16 interrupt

There is an interrupt that can be generated by the external device setting GPIO16 high.
This interrupt has an enable bit to route it to the user CPU. This enable bit can be written by the user CPU (at address 0x8000c014).

When the user CPU receives an interrupt, it first checks the value of the *PIL* and *ET* bits in the PSR (see the SPARC reference manual on the installation CD for details). If *PIL* is 0 and *ET* is 1 then the CPU will jump to the interrupt handler trap address (0x110). The installation CD includes a low level library routine to handle interrupts cleanly. Refer to section 7.3.3 for details of this library.

7.3 CPU Programming

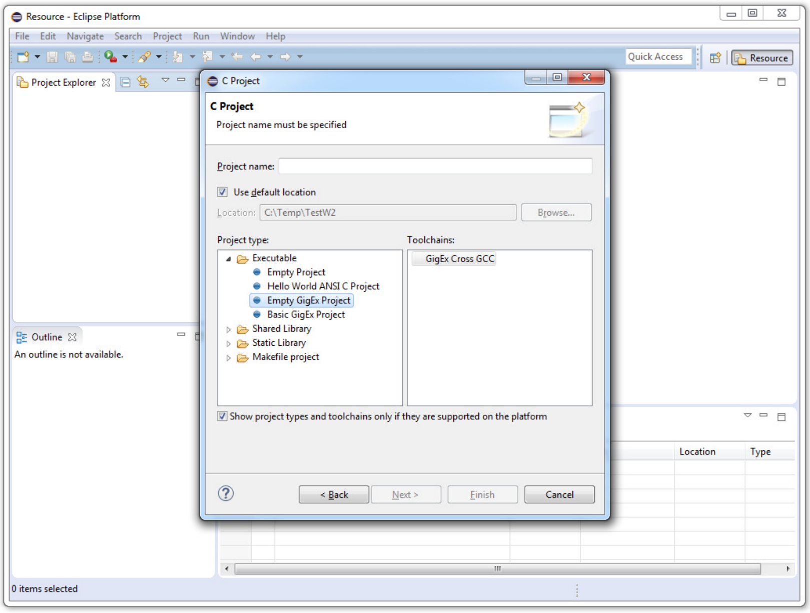
The ZestET2 installation CD includes Eclipse and GCC tools for programming the CPU in C. These are standard, off-the shelf tools supplied pre-built for Windows for convenience. The Eclipse implementation includes a new project type for GigEx to simplify the configuration of the GCC compiler and debugger. Since most applications for the CPU will not need an operating system, a low level library is supplied to handle the 'bare metal' functions of the CPU to allow the user to concentrate on the main application.

7.3.1 Programming the CPU Using Eclipse and GCC

The ZestET2 installation CD includes pre-built versions of the GNU GCC compiler for Windows and Linux. The GCC compiler is a cross-compiler targeting SPARC architectures and uses unmodified GNU source code which can be downloaded from http://gcc.gnu.org

User code can be compiled from the command line or in the Eclipse development environment. To create a new project in Eclipse, follow these steps.

1. First, launch the Eclipse IDE and select a new or existing workspace location.
2. Select 'File' from the Eclipse menu and 'New' then 'Project'
3. Select 'C Project' or 'C++ Project' as appropriate and click 'Next'. You should have the following options displayed:



C Project

C Project

Project name must be specified

Project name:

☑ Use default location

Location: C:\Temp\TestW2 Browse...

Project type:
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Empty GigEx Project
  - Basic GigEx Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
GigEx Cross GCC

☑ Show project types and toolchains only if they are supported on the platform

An outline is not available.

0 items selected

You can now select 'Empty GigEx Project' which will create a project with all the correct compiler flags but without any source files or 'Basic GigEx Project' which includes a starting C/C++ file.

4. Enter the name of the project in the 'Project name:' box and click Finish

You should now have a project where you can develop your application. Building the application will result in an ELF output file in the output Debug or Release directory that you can download to the CPU. The application can be copied into the flash on ZestET2 using the CPanel application (see section 9.3) or it can be downloaded into SDRAM over the network connection using the GDB debugger (see section 7.3.2).

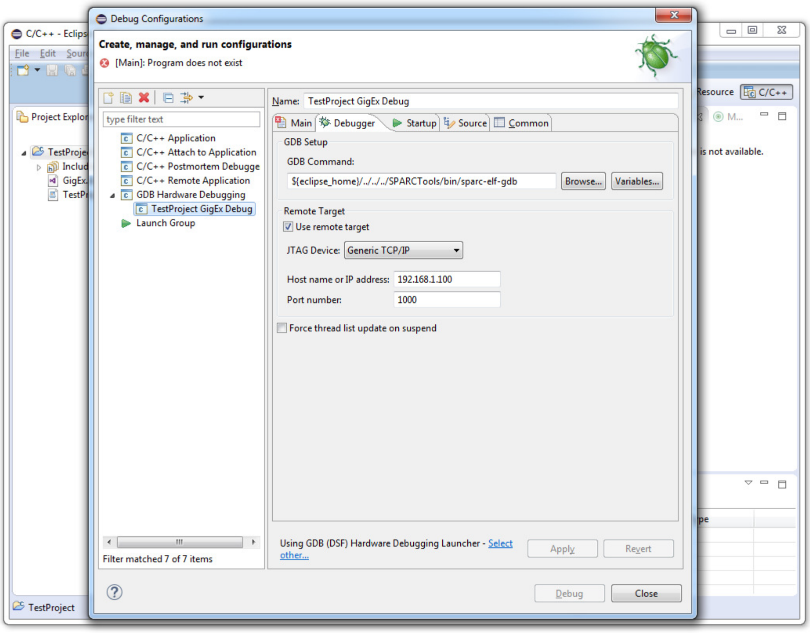
When building from the command line, the following options should be specified:

Option	Compiler or Linker	Description
-gdwarf-2	Compiler	Set debug output format to DWARF2. This is compatible with the GDB debugger supplied on the CD.
-mcpu-v8	Compiler	Target the SPARC v8 architecture
-msoft-float	Compiler	Generate software floating point routines rather than hardware instructions (the CPU has no hardware floating point unit)
-nostartfiles	Linker	Do not include startup files (the startup code is included in the GigEx low level library)
-nodefaultlibs	Linker	Do not include the standard C libraries (they must be included before the GigEx low level library - see below)
-lc -lGigEx	Linker	Include the C standard library and the GigEx low level library (in that order)
-T GigEx.ld	Linker	Include the GigEx linker control script which describes the CPU memory map for the linker.

7.3.2 Debugging Applications Using Eclipse and GDB

Applications can be debugged on the user CPU across the network connection by using the GDB debugger. This debugger is normally run from within the Eclipse environment and includes powerful debugging tools to read variables and memory locations, set breakpoints, step code and so on. The creation of a GigEx project as described in section 7.3.1 will automatically create a debug profile to execute the application. However, if the GigEx IP address is not set to the default of 192.168.1.100 or the GDB server port has been changed in the GigEx web configuration server then the debug profile will have to be changed to match the new settings as follows:

1. Select the 'Run' menu in the Eclipse IDE and choose 'Debug Configurations...'
2. Select the test project debug configuration under 'GDB Hardware Debugging' and choose the 'Debugger' tab. You should have a window as shown below:



Debug Configurations

Create, manage, and run configurations

[Main]: Program does not exist

Name: TestProject GigEx Debug

GDB Setup

GDB Command: ${eclipse_home}/../../../SPARCTools/bin/sparc-elf-gdb

Remote Target

☑ Use remote target

JTAG Device: Generic TCP/IP

Host name or IP address: 192.168.1.100

Port number: 1000

□ Force thread list update on suspend

Using GDB (DSF) Hardware Debugging Launcher - Select other...

Filter matched 7 of 7 items

3. Set the 'Host Name or IP address' to the IP address of the ZestET2. Set the 'Port number' to the GDB port number set in the GigEx web configuration server.
4. Click 'Debug' to launch the debugger which will download and start the application on the user CPU.

7.3.3 Low Level Support Library

All user CPU applications should be linked to the supplied libGigEx.a static library file (from the CPU\Lib directory on the installation CD). This library contains the trap tables and default trap handlers for the CPU. It also contains a number of utility functions to manage the low level features of the CPU. These functions are detailed below.

GIGEX_WRITE8(*a*, *d***)**
GIGEX_READ8(*a***)**
GIGEX_WRITE16(*a*, *d***)**
GIGEX_READ16(*a***)**
GIGEX_WRITE32(*a*, *d***)**
GIGEX_READ32(*a***)**

Parameters

a	Address of register to access.
d	Data to write to register.

Return Value

GIGEX_WRITE8	No return value
GIGEX_READ8	Value read from register
GIGEX_WRITE16	No return value
GIGEX_READ16	Value read from register
GIGEX_WRITE32	No return value
GIGEX_READ32	Value read from register

Description

GIGEX_READn and **GIGEX_WRITE**n are a set of macros that perform reads and writes of a specified size to/from the GigEx registers. They ensure that the read/write occurs immediately even when the C optimiser is turned on.

The *a* parameter is the address of the register and the *d* parameter (writes only) is the data to write. The read macros return the value read from the register.

For example:

```
/* Write to Channel 0 connection IP address register (high 16 bits) */
GIGEX_WRITE16(0x80000002, 0xc0a8);

/* Read from Channel 0 connection IP address register (high 16 bits) */
Val = GIGEX_READ16(0x80000002);
```

GigExInterruptInstallHandler

GIGEX_ISR_HANDLER GigExInterruptInstallHandler(GIGEX_INTERRUPT_TYPE *Type*,
GIGEX_ISR_HANDLER *Fn*, **void** **Arg***);**

Parameters

Type	Type of interrupt to install handler for.
Fn	User function to call on receiving interrupt.
Arg	Argument to pass to user function when called.

Return Value

Returns pointer to the previously installed trap handler or NULL if no previous handler installed.

Description

GigExInterruptInstallHandler can be used to install a user function that will be called in the event of an interrupt occurring. The GigEx low level library will perform the low level trap handling before switching the CPU into user mode and executing the user function. On returning from the user function the low level library will handle clean up and return from the trap.

At present, the *Type* parameter must be **GIGEX_INTERRUPT**.

The user function must be of the following type:

```
void InterruptHandler(void *Arg);
```

where *Arg* will be the value passed into **GigExInterruptInstallHandler** call.

For example:

```
/* Interrupt handler */
void IntHandler(void *Arg)
{
       *(int *)Arg ++;
       /* NB: Must clear cause of interrupt here */
}

...

/* Install handler and pass in pointer to counter variable)
int CallCount = 0;
GigExInterruptInstallHandler(GIGEX_INTERRUPT, IntHandler, &CallCount);
```

GigExInterruptEnableAll
GigExInterruptDisableAll

void GigExInterruptEnableAll (void);
void GigExInterruptDisableAll (void);

Parameters

None.

Return Value

None.

Description

GigExInterruptEnableAll can be used to enable all interrupt sources. **GigExInterruptDisableAll** can be used to disable all interrupt sources. The functions mask interrupts by changing the *PIL* bits of the processor PSR. These functions can be used to protect a critical section of code.

Note that reset value of PIL has interrupts disabled so there should be a call to **GigExInterruptEnableAll** at the start of the CPU program after the user interrupt handler is installed with a call to **GigExInterruptInstallHandler**.

For example:

```
/* Protect critical section of code */
GigExInterruptDisableAll();
VolatileCounter++;
GigExInterruptEnableAll();
```

GigExSetTimerInterval

void GigExSetTimerInterval (unsigned long *Interval***);**

Parameters

Interval	Interval in clock cycles for the periodic interrupt

Return Value

None.

Description

GigExSetTimerInterval can be used to set the limit of the interval timer. The interval timer will wrap around at this limit and generate an interrupt.

For example:

```
/* Set timer limit to 100 ms */
GigExSetTimerInterval(GIGEX_TIMER_FREQ/10);
```

GigExGetTime

unsigned long GigExGetTime(void);

Parameters

None

Return Value

Clock cycle count since last periodic interrupt.

Description

GigExGetTime will return the number of clock cycles elapsed since the last periodic interval.

For example:

```
        /* Get time in seconds since last interrupt */
        Time = GigExGetTime()/(float)GIGEX_TIMER_FREQ;
```

7.3.4 Example Communication and Co-operation Between CPU and FPGA

GigEx provides a number of mechanisms for communicating between the user CPU and the external device (FPGA). These mechanisms are important in applications where the workload is shared between the CPU performing high level functions and the external device generating high speed data. For example, the CPU may handle complex control and status operations or generate infrequent packet headers while the external device gathers high speed data for transmission on the network. The mechanisms offered are 256 bytes of shared general purpose user comms registers in each direction, mailbox interrupts that can be generated in each direction, GPIO pins and a GPIO interrupt input to the user CPU.

The exact use of these mechanisms is application dependent but a suggested sequence would be:

User CPU	**External Device (FPGA)**
1. Generate packet header and write to network	1. Wait for mailbox interrupt from CPU
2. Write parameters to outgoing general purpose user comms registers	
3. Generate outgoing mailbox interrupt	
4. Wait for incoming mailbox interrupt	2. Read parameters from outgoing general purpose user comms registers
	3. Transmit high speed data to network
	4. Write status results back to incoming general purpose user comms registers
	5. Generate incoming mailbox interrupt to CPU
5. Read status results from incoming general purpose user comms registers	6. Go to stage 1.
6. Go to stage 1.	

At stage 3, the user CPU may prefer to set a GPIO pin high to indicate to the external device that it should start transferring data. Alternatively, the external device could poll the general purpose user comms registers for a particular value to trigger the data transfer.

At stage 1, the external device could monitor a GPIO pin to trigger the transfer. At stage 5 it could use GPIO16 as a GPIO interrupt to the user CPU or it could write a completion value to the general purpose user comms registers which is polled for by the CPU.

8 FPGA Programming

The Xilinx Artix-7 FPGA fitted to the ZestET2 board can be programmed using the Xilinx Vivado design suite. The FPGA is supported by the WebPack edition of the tools which can be downloaded for free from the Xilinx website.

The ZestET2 installation CD includes various example designs and logic cores which can be used as a starting point for new FPGA designs. The example programs are described in section 9.4.

The following sections detail the connections to the FPGA on ZestET2. The pin definitions are contained in four Xilinx constraint files (.xdc files) on the installation CD. The appropriate constraint file can be added to the Xilinx Vivado design project to lock the pin locations in the correct positions for each mode. The constraint files are located in the Constraints directory as follows:

ZestET2_16BitSRAM.xdc
ZestET2_8BitSRAM.xdc
ZestET2_FIFO.xdc
ZestET2_Direct.xdc

8.1 Clock Connections

ZestET2 has a 50MHz reference crystal fitted which is connected to a clock input pin on the FPGA. The clock is a 1v5 input only pin.

FPGA Ball Number	Signal
T5	RefClk (50MHz)

8.2 FPGA SDRAM Connections

This section contains information about the connections between the GigEx device and the DDR3 SDRAM memory. The SDRAM interface runs at 1v5 and all outputs should have fast slew rate set.

FPGA Ball Number	IO Standard	Input Termination	Signal
U8	LVCMOS15		RESETn
K6	SSTL15		ODT
K3	DIFF_SSTL15		CK
L3	DIFF_SSTL15		CKn
K5	SSTL15		CASn
L6	SSTL15		RASn
P5	SSTL15		CKE
M6	SSTL15		WEn
L5	SSTL15		BA0
P3	SSTL15		BA1
N6	SSTL15		BA2
N5	SSTL15		A0
M1	SSTL15		A1
N4	SSTL15		A2
L4	SSTL15		A3
P2	SSTL15		A4

M4	SSTL15		A5
R2	SSTL15		A6
M3	SSTL15		A7
T1	SSTL15		A8
M2	SSTL15		A9
P4	SSTL15		A10
N1	SSTL15		A11
N2	SSTL15		A12
L1	SSTL15		A13
R1	SSTL15		A14
V5	SSTL15		LDM
U2	DIFF_SSTL15	UNTUNED_SPLIT_60	LDQS
V2	DIFF_SSTL15	UNTUNED_SPLIT_60	LDQSn
T4	SSTL15	UNTUNED_SPLIT_60	DQ0
U4	SSTL15	UNTUNED_SPLIT_60	DQ1
V1	SSTL15	UNTUNED_SPLIT_60	DQ2
V4	SSTL15	UNTUNED_SPLIT_60	DQ3
R3	SSTL15	UNTUNED_SPLIT_60	DQ4
U3	SSTL15	UNTUNED_SPLIT_60	DQ5
U1	SSTL15	UNTUNED_SPLIT_60	DQ6
T3	SSTL15	UNTUNED_SPLIT_60	DQ7
R5	SSTL15		UDM
U9	DIFF_SSTL15	UNTUNED_SPLIT_60	UDQS
V9	DIFF_SSTL15	UNTUNED_SPLIT_60	UDQSn
U6	SSTL15	UNTUNED_SPLIT_60	DQ8
R7	SSTL15	UNTUNED_SPLIT_60	DQ9
U7	SSTL15	UNTUNED_SPLIT_60	DQ10
R6	SSTL15	UNTUNED_SPLIT_60	DQ11
V7	SSTL15	UNTUNED_SPLIT_60	DQ12
T6	SSTL15	UNTUNED_SPLIT_60	DQ13
V6	SSTL15	UNTUNED_SPLIT_60	DQ14
T8	SSTL15	UNTUNED_SPLIT_60	DQ15

8.3 FPGA User IO connections

This section contains information about the connections between the GigEx device and the user IO connectors. Pins form three FPGA banks are wired out to the IO connectors (banks 15, 16 and 35) and each bank can operate with its own voltage. The pins can be set to any supported IO standard compatible with the matching VCC voltage level. See section 3.2 for details of the VCC pins.

FPGA Ball Number	Signal	FPGA Ball Number	Signal
C12	IO15(0)	B12	IO15(1)
B11	IO15(2)	A11	IO15(3)
D14	IO15(4)	C14	IO15(5)
D12	IO15(6)	D13	IO15(7)
A13	IO15(8)	A14	IO15(9)
B13	IO15(10)	B14	IO15(11)
A15	IO15(12)	A16	IO15(13)
D15	IO15(14)	C15	IO15(15)

B18	IO15(16)	A18	IO15(17)
B16	IO15(18)	B17	IO15(19)
E18	IO15(20)	D18	IO15(21)
C16	IO15(22)	C17	IO15(23)
E17	IO15(24)	D17	IO15(25)
E15	IO15(26)	E16	IO15(27)
F15	IO15(28)	F16	IO15(29)
F13	IO15(30)	F14	IO15(31)
H16	IO15(32)	G16	IO15(33)
H14	IO15(34)	G14	IO15(35)
G18	IO15(36)	F18	IO15(37)
H17	IO15(38)	G17	IO15(39)
J14	IO15(40)	H15	IO15(41)
K13	IO15(42)	J13	IO15(43)
J17	IO15(44)	J18	IO15(45)
K15	IO15(46)	J15	IO15(47)
K16	IO15(48)	G13	IO15(49)
D9	IO16(0)	B8	IO16(1)
A8	IO16(2)	C11	IO16(3)
D10	IO16(4)	D8	IO35(0)
C7	IO35(1)	E7	IO35(2)
D7	IO35(3)	B7	IO35(4)
B6	IO35(5)	A6	IO35(6)
A5	IO35(7)	C6	IO35(8)
C5	IO35(9)	C4	IO35(10)
B4	IO35(11)	A4	IO35(12)
A3	IO35(13)	B3	IO35(14)
B2	IO35(15)	C2	IO35(16)
C1	IO35(17)	B1	IO35(18)
A1	IO35(19)	D5	IO35(20)
D4	IO35(21)	E3	IO35(22)
D3	IO35(23)	E2	IO35(24)
D2	IO35(25)	F1	IO35(26)
E1	IO35(27)	E6	IO35(28)
E5	IO35(29)	G6	IO35(30)
F6	IO35(31)	F4	IO35(32)
F3	IO35(33)	G4	IO35(34)
G3	IO35(35)	H2	IO35(36)
G2	IO35(37)	H1	IO35(38)
G1	IO35(39)	H6	IO35(40)
H5	IO35(41)	J4	IO35(42)
H4	IO35(43)	J3	IO35(44)
J2	IO35(45)	K2	IO35(46)
K1	IO35(47)	F5	IO35(48)
J5	IO35(49)		

8.4 FPGA GigEx Connections

This section contains information about the connections between the GigEx device and the User FPGA.

All pins between the FPGA and GigEx operate at 1v8. Output pins from the FPGA should be set to have a drive strength of 4mA and a fast slew rate.

The high speed interface can be in one of four modes: 16 bit SRAM interface, 8 bit SRAM interface, FIFO interface or Direct mode. The connection names for each mode are shown in the table below.

FPGA Ball Number	16-bit	8-bit	FIFO	Direct	GPIO
P15	CLK	CLK	CLK	CLK	
L16	nINT	nINT	nINT	nINT	
R15	nCS	nRE	nRF0	nRxFull	
U12	nWE	nWE	nTx	nTx	
R11	nBE1	DA9	nRF1		
V12	nBE0	DA8	nRF2		
M16	HEAD	HEAD	nTF7	nRx	GPIO19
T10	LEN	DA7	TC2	nTxFull	GPIO18
N15		DA6	TC1	nRxEmpty	GPIO17
U11	EOF0	EOF	nRF3		GPIO16/GPIOInt
V10	EOF1	DA5	TC0	B31	GPIO15
V11		DA4	nRF4	B30	GPIO14
M14		DA3	nRF5	B29	GPIO13
T9		DA2	nRF6	B28	GPIO12
T13		DA1	nRF7	B27	GPIO11
R10		DA0	nTF6	B26	GPIO10
N14	A9	QA9	nTF5	B25	GPIO9
P14	A8	QA8	nTF4	B24	GPIO8
T11	A7	QA7	RC2	B23	GPIO7
U13	A6	QA6	RC1	B22	GPIO6
V14	A5	QA5	RC0	B21	GPIO5
R12	A4	QA4	nRx	B20	GPIO4
R13	A3	QA2	nTF3	B19	GPIO3
T14	A2	QA2	nTF2	B18	GPIO2
U14	A1	QA1	nTF1	B17	GPIO1
T15	A0	QA0	nTF0	B16	GPIO0
R16	DQ15	D7	D7	B15	
U16	DQ14	D6	D6	B14	
T16	DQ13	D5	D5	B13	
V15	DQ12	D4	D4	B12	
R17	DQ11	D3	D3	B11	
V16	DQ10	D2	D2	B10	
U17	DQ9	D1	D1	B9	
V17	DQ8	D0	D0	B8	
N16	DQ7	Q7	Q7	B7	
P18	DQ6	Q6	Q6	B6	
R18	DQ5	Q5	Q5	B5	
M17	DQ4	Q4	Q4	B4	

N17	DQ3	Q3	Q3	B3	
U18	DQ2	Q2	Q2	B2	
T18	DQ1	Q1	Q1	B1	
M18	DQ0	Q0	Q0	B0	

The low speed interface can be configured as a SPI slave interface or as a UART. The FPGA pin connections are shown below. Note that pin E9 is the FPGA configuration clock pin. To access this pin you must instantiate the STARTUPE2 primitive in your design. Refer to the Xilinx documentation for further details of this primitive.

FPGA Ball Number	SPI Slave	UART	User Flash
E9	SPIs_SCLK	CTS	SPIs_SCLK
K18	SPIs_MISO	RX	SPIs_MISO
K17	SPIs_MOSI	TX	SPIs_MOSI
M13	SPIs_nSCS	RTS	
L13			SPIs_nFCS

The PTP and SyncE connections are shown below

FPGA Ball Number	Signal
P17	SyncE
L18	PTPTrigger/PTPEvent

8.4.1 High Speed Interface Signals

The High Speed interface consists of 1 clock pin, 40 general purpose control/data pins and 1 interrupt pin. The 40 general purpose pins can take one of 4 functions depending on the mode of the high speed interface. In the two register modes the external device is always the master of the interface. The type column in the tables below is I for input to GigEx (i.e. output from FPGA) or O for output from GigEx (i.e. input to FPGA) or I/O for bidirectional signals. Refer to section 4.2 for further details of the signals in each mode.

Pin Name	Type	Description
CLK	I/O	High Speed Interface Clock All other signals on the high speed bus are synchronous to this clock. The clock can be driven from the external user device into the GigEx in which case the frequency can be between 5MHz and 125MHz. Alternatively, GigEx can generate a fixed 125MHz clock out to the connected user device. The direction can be controlled via the configuration web server or via the UART command interface. For correct functionality, there must be a free running clock on this pin. Therefore, if the connected user device does not generate a clock in to GigEx then GigEx must be set to generate a clock out to the connected device.
nINT	O	High Speed Interface Interrupt Active low output from the GigEx which will be set when various events occur. The interrupt conditions can be enabled and distinguished with the

		register map described in section 5.

16 bit register mode

Pin Name	Type	Description
DQ0	I/O	DQ[15..0] : 16 bit bidirectional data bus
DQ1	I/O	
DQ2	I/O	A[9..0] : 10 bit register byte address. A0 should always be 0.
DQ3	I/O	
DQ4	I/O	EOF[1..0] : 2 bit End of Frame indicator. EOF1 will be high when DQ[15..8] is the last byte in a received frame. EOF0 will be 1 when DQ[7..0] is the last byte in a received frame.
DQ5	I/O	
DQ6	I/O	
DQ7	I/O	
DQ8	I/O	LEN : Length field indicator. This signal will be high when the data being read on DQ[15..0] is a frame length value.
DQ9	I/O	
DQ10	I/O	
DQ11	I/O	HEAD : UDP header field indicator. This signal will be high when the data being read on DQ[15..0] is UDP frame header information.
DQ12	I/O	
DQ13	I/O	
DQ14	I/O	nBE[1..0] : Active low byte write enables. nBE1 should be low to indicate the data on DQ[15..8] is valid during a write. nBE0 should be low to indicate the data on DQ[7..0] is valid during a write.
DQ15	I/O	
A0	I	
A1	I	
A2	I	nWE : Active low write enable.
A3	I	
A4	I	nCS : Active low chip select.
A5	I	
A6	I	
A7	I	
A8	I	
A9	I	
EOF1	O	
EOF0	O	
LEN	O	
HEAD	O	
nBE0	I	
nBE1	I	
nWE	I	
nCS	I	

8 bit register mode

Pin Name	Type	Description
Q0	O	Q[7..0] : 8 bit data output from the GigEx
Q1	O	
Q2	O	D[7..0] : 8 bit data input to the GigEx
Q3	O	
Q4	O	QA[9..0] : 10 bit read address to the GigEx
Q5	O	
Q6	O	DA[9..0] : 10 bit write address to the GigEx
Q7	O	

D0	I	EOF : End of Frame field indicator. This signal will be high when the data being read on Q[7..0] is the last byte in a frame. HEAD : UDP header field indicator. This signal will be high when the data being read on Q[7..0] is UDP frame header information. nWE : Active low write enable. nRE : Active low read enable.
D1	I	
D2	I	
D3	I	
D4	I	
D5	I	
D6	I	
D7	I	
QA0	I	
QA1	I	
QA2	I	
QA3	I	
QA4	I	
QA5	I	
QA6	I	
QA7	I	
QA8	I	
QA9	I	
DA0	I	
DA1	I	
DA2	I	
DA3	I	
DA4	I	
DA5	I	
EOF	O	
DA6	I	
DA7	I	
HEAD	O	
DA8	I	
DA9	I	
nWE	I	
nRE	I	

FIFO mode

Pin Name	Type	Description
Q0	O	Q[7..0] : 8 bit receive data output from the GigEx D[7..0] : 8 bit transmit data input to the GigEx nTF[7..0] : Active low transmit FIFO full flags from the GigEx to the external device. nRF[7..0] : Active low receive FIFO full flags to the GigEx from the external device. nRx : Active low receive data valid flag from the GigEx to the external device. RC[2..0] : Receive data channel number from the GigEx to the external device.
Q1	O	
Q2	O	
Q3	O	
Q4	O	
Q5	O	
Q6	O	
Q7	O	
D0	I	
D1	I	
D2	I	
D3	I	
D4	I	
D5	I	
D6	I	
D7	I	

nTF0	O	nTx : Active low transmit data valid flag from the external device to the GigEx. TC[2..0] : Transmit data channel number from the external device to the GigEx.
nTF1	O	
nTF2	O	
nTF3	O	
nRx	O	
RC0	O	
RC1	O	
RC2	O	
nTF4	O	
nTF5	O	
nTF6	O	
nRF7	I	
nRF6	I	
nRF5	I	
nRF4	I	
TC0	I	
nRF3	I	
TC1	I	
TC2	I	
nTF7	O	
nRF2	I	
nRF1	I	
nTx	I	
nRF0	I	

Direct or "Bit banging" mode

Pin Name	Type	Description
B0	I/O	B[31..0] : 32 bit general purpose IO pins. These can be assigned as inputs or outputs in groups of 8 bits. nRxEmpty : Active low receive FIFO empty signal. This indicates that the receive FIFO inside the GigEx has become empty due to no data from the network being available and that the data output on the B[] pins has been interrupted. nTxFull : Active low transmit FIFO full signal. This indicates that the transmit FIFO inside the GigEx has become full due to network congestion. Any further data written to the B[] pins will be lost. nRx : Active low strobe indicating that output data on the B[] pins is valid. nTx : Active low strobe indicating that input data on B[] pins is valid. nRxFull : Active low flag to the GigEx indicating that the external device is busy. This will pause data output on the B[] pins.
B1	I/O	
B2	I/O	
B3	I/O	
B4	I/O	
B5	I/O	
B6	I/O	
B7	I/O	
B8	I/O	
B9	I/O	
B10	I/O	
B11	I/O	
B12	I/O	
B13	I/O	
B14	I/O	
B15	I/O	
B16	I/O	
B17	I/O	
B18	I/O	
B19	I/O	
B20	I/O	
B21	I/O	
B22	I/O	
B23	I/O	

B24 B25 B26 B27 B28 B29 B30 B31 nRxEmpty nTxFull nRx nTx nRxFull	I/O I/O I/O I/O I/O I/O I/O I/O O O O I I	

GPIO pins

Pin Name	Type	Description
GPIO[23:0]	I/O	GigEx provides the ability to switch the function of 24 of its pins into a GPIO mode. In this mode the pin can be configured as an output with a fixed value or as an input. The GPIO pin can be controlled over Ethernet, via the web configuration server or from the User CPU. Pins should only be switched into GPIO mode if they have no other function in the currently selected mode. For example, GPIO14 can be used in 16 bit SRAM mode but not in 8 bit SRAM mode or FIFO mode.

8.4.2 Low Speed Interface Signals

SPI Interfaces

Pin Name	Type	Description
SPIs_nSCS SPIs_SCK SPIs_MISO SPIs_MOSI	I I O I	Slave SPI interface The SPI slave interface can be used as an alternative to the high speed interface for accessing the register map from the external device connected to GigEx. In FIFO and Direct high speed modes, the only way to access the register map is through the slave SPI or UART interface. The slave SPI interface pins are shared with the UART interface.
SPIs_nFCS	I	Flash chip select line. To access the user flash chip, use SPIs_SCK, SPIs_MISO and SPIs_MOSI from the slave serial interface but assert SPIs_nFCS rather than SPIs_nSCS.

UART Interface

Pin Name	Type	Description
TX RX	O I	UART interface The UART can be used to send ASCII commands to control the GigEx and to receive status from the GigEx.

		In FIFO and Direct high speed modes, the only way to access the register map is through the slave SPI or UART interface. The UART is a TTL interface running at the same voltage as the main register interface (1.8V). It can therefore be driven directly by the User FPGA. The UART pins are shared with the SPI slave interface.
CTS RTS	I O	UART handshaking signals. CTS is an active low 'Clear to send' signal. When active, the UART is able to transmit data on the TX pin. When inactive, data transmission on TX will be paused. RTS is an active low 'Ready to send' signal. When active, the UART is able to accept data on the RX pin. When inactive, data transfer on the RX pin should be paused. The UART is a TTL interface running at the same voltage as the main register interface (1.8V). It can therefore be driven directly by the User FPGA. The UART pins are shared with the SPI slave interface. Note that on ZestET2, the 'UART Flow pin swap' option must be ticked in the web configuration server to place RTS and CTS on the correct pins.

Synchronous Ethernet Interface

Pin Name	Type	Description
SyncE	O	Synchronous Ethernet clock output. The SyncE clock output is a 125MHz clock that will be synchronised across all synchronous Ethernet devices on the network.

1588 Trigger/Event

Pin Name	Type	Description
TrigEvent	I/O	IEEE1588 PTP trigger output and event input. The IEEE1588 PTP trigger output pin can be set to generate a pulse at an exact moment in time. The output will be synchronized across all PTP compliant devices in the network. The pulse can be generated as a one off event or as a 1 pulse per second output. Alternatively, the pin can be used as an input to the GigEx to capture an event. A rising edge on the pin will latch a timestamp inside the GigEx which can then be read out of the register interface.

8.5 FPGA Cores

The ZestET2 Software Support CD contains Verilog and VHDL source for each FPGA interface. These cores can form the starting point of a FPGA design for the ZestET2. The following cores are provided:

Core name	Description
ZestET2_Clocks	Instantiates clock primitives to generate clocks for the Ethernet interface (125MHz) and RAM controller (200 and 400MHz) from the external 50MHz reference clock.
ZestET2_GigExPhy16	Low level physical interface to the GigEx when in 16 bit SRAM mode.
ZestET2_GigExPhy8	Low level physical interface to the GigEx when in 8 bit SRAM mode.
ZestET2_GigExPhyFIFO	Low level physical interface to the GigEx when in FIFO mode.
ZestET2_GigExPhyDirect	Low level physical interface to the GigEx when in Direct mode.
ZestET2_GigExSPI	Master SPI controller to read/write registers in GigEx via the slave SPI port.
ZestET2_SDRAMTop	Custom version of DDR3 SDRAM controller generated by the Xilinx Vivado MIG tool.

8.5.1 ZestET2_Clocks

The ZestET2_Clocks core uses the FPGA's PLLs to generate clocks for the Ethernet and DDR3 SDRAM interfaces. The core has the following ports:

Port Name	Direction	Description
RST	I	Global power on reset signal. This should be high after configuration to reset the PLLs and then low to allow normal operation of the core.
RAMRST	O	Reset output to the SDRAM controller. This signal will be high until the PLLs have achieved lock and are generating valid clock outputs.
RefClk	I	50MHz reference clock fed from the crystal on the ZestET2 board.
Clk200	O	200MHz output clock used for the SDRAM controller IO calibration circuits. Should be connected to the Clk200 input on the SDRAM controller
EthClk	O	125MHz output clock for the Ethernet circuitry. The GigEx device can run at lower clock rates than 125MHz so the ZestET2_Clocks core can be modified if 125MHz is not a suitable frequency for the design.

8.5.2 ZestET2_GigExPhy16

The ZestET2_GigExPhy16 core instantiates low level primitives to form the physical interface to the GigEx device when operating in 16 bit SRAM mode. For correct operation, the instantiating code must set the CLOCK_RATE parameter (Verilog) or generic (VHDL) to the correct frequency of the CLK port. For example, in Verilog:

```
ZestET2_GigExPhy16 #(.CLOCK_RATE(125000000)) GigExPhyInst (
        .CLK(EthClk),
        ...
};
```

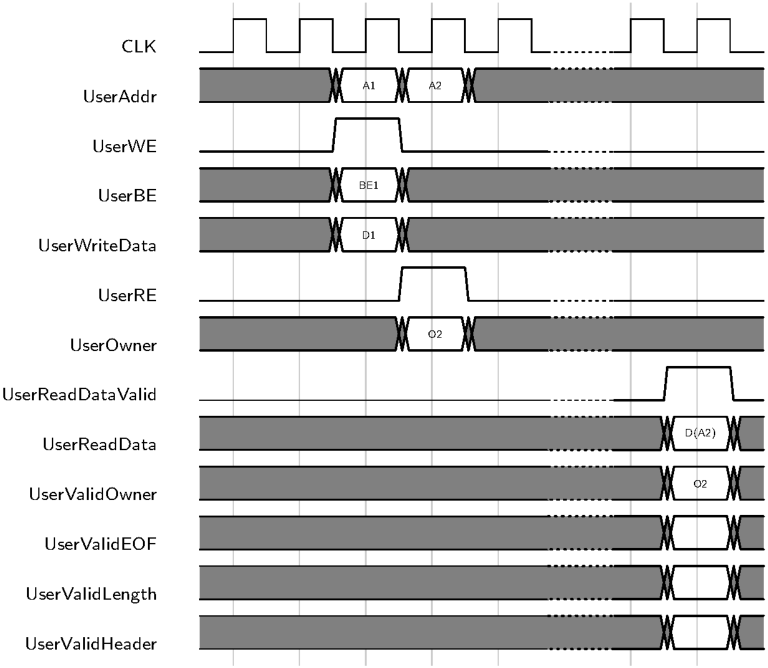
or in VHDL:

```
GigExPhyInst : entity work.ZestET2_GigExPhy16
        generic map (
            CLOCK_RATE => 125000000
        )
        port map (
            CLK => EthClk,
            ...
        );
```

The core has the following ports:

Port Name	Direction	Description
CLK	I	Interface clock. This is the clock that will be used to transfer data between the FPGA and GigEx. All other ports on the core are synchronous to this clock.
GigEx_Clk GigEx_nCS GigEx_nWE GigEx_Addr GigEx_nBE GigEx_Data GigEx_EOF GigEx_Length GigEx_Header GigEx_nInt	O O O O O I/O I I I I	Interface to GigEx device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the GigEx device.
UserWE UserRE UserAddr UserBE UserWriteData UserReadData UserReadDataValid UserOwner UserValidOwner UserValidEOF UserValidLength UserValidHeader UserInterrupt	I I I I I O O I O O O O O	User interface to/from main FPGA design. To write data to GigEx, UserAddr, UserBE and UserWriteData should be set to the required values in the same cycle as UserWE is asserted for 1 clock cycle. To read data from GigEx, UserAddr and UserOwner should be set to the required values in the same cycle as UserRE is asserted for a single clock cycle. UserOwner is a user defined value that will be returned from the core along with the valid data. It can be used to determine which read data belongs to which read access. When the read operation has completed, the core will set UserReadData to the data read from GigEx in the same cycle as it asserts UserReadDataValid for a single cycle. Also valid in this cycle will be the UserValidOwner (set to the value of UserOwner when UserRE was asserted), UserValidEOF, UserValidLength and UserValidHeader (which qualify the type of data read when accessing the data FIFO of GigEx). The number of cycles of latency for a read is not fixed but is determined automatically by the core depending on the frequency of CLK. UserInterrupt is an active high signal that will be set when GigEx is generating an interrupt.

An example transaction on the user interface of the core is shown below.



8.5.3 ZestET2_GigExPhy8

The ZestET2_GigExPhy8 core instantiates low level primitives to form the physical interface to the GigEx device when operating in 8 bit SRAM mode. For correct operation, the instantiating code must set the CLOCK_RATE parameter (Verilog) or generic (VHDL) to the correct frequency of the CLK port. For example, in Verilog:

```
ZestET2_GigExPhy8 #(.CLOCK_RATE(125000000)) GigExPhyInst (
        .CLK(EthClk),
        ...
};
```

or in VHDL:

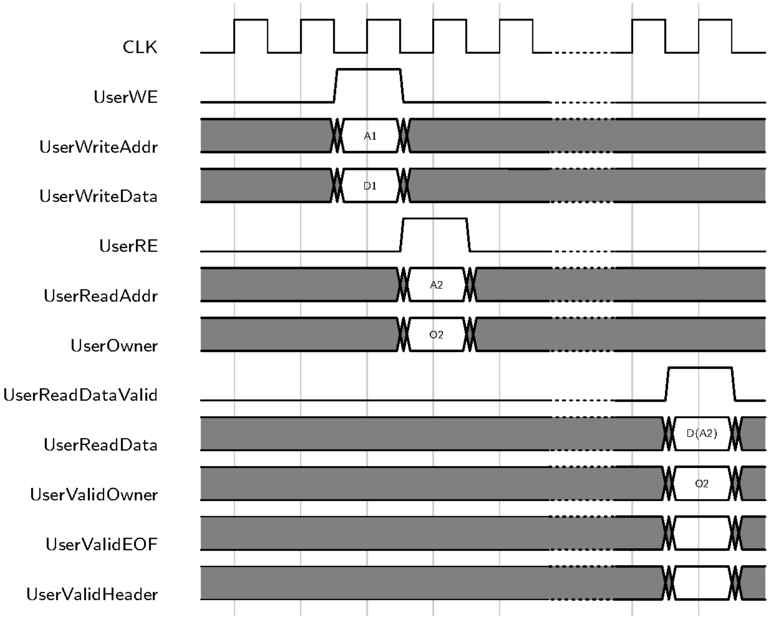
```
GigExPhyInst : entity work.ZestET2_GigExPhy8
        generic map (
            CLOCK_RATE => 125000000
        )
        port map (
            CLK => EthClk,
            ...
        );
```

The core has the following ports:

Port Name	Direction	Description
CLK	I	Interface clock. This is the clock that will be used to transfer data between the FPGA and GigEx. All other ports on the core are synchronous to this clock.
GigEx_Clk GigEx_nRE GigEx_nWE	O O O	Interface to GigEx device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the GigEx device.

GigEx_QA GigEx_DA GigEx_Q GigEx_D GigEx_EOF GigEx_Header GigEx_nInt	O O I O I I I	
UserWE UserRE UserWriteAddr UserReadAddr UserWriteData UserReadData UserReadDataValid UserOwner UserValidOwner UserValidEOF UserValidHeader UserInterrupt	I I I I I O O I O O O O	User interface to/from main FPGA design. To write data to GigEx, UserWriteAddr and UserWriteData should be set to the required values in the same cycle as UserWE is asserted for 1 clock cycle. To read data from GigEx, UserReadAddr and UserOwner should be set to the required values in the same cycle as UserRE is asserted for a single clock cycle. UserOwner is a user defined value that will be returned from the core along with the valid data. It can be used to determine which read data belongs to which read access. When the read operation has completed, the core will set UserReadData to the data read from GigEx in the same cycle as it asserts UserReadDataValid for a single cycle. Also valid in this cycle will be the UserValidOwner (set to the value of UserOwner when UserRE was asserted), UserValidEOF and UserValidHeader (which qualify the type of data read when accessing the data FIFO of GigEx). The number of cycles of latency for a read is not fixed but is determined automatically by the core depending on the frequency of CLK. UserInterrupt is an active high signal that will be set when GigEx is generating an interrupt.

An example transaction on the user interface of the core is shown below.



8.5.4 ZestET2_GigExPhyFIFO

The ZestET2_GigExPhyFIFO core instantiates low level primitives to form the physical interface to the GigEx device when operating in FIFO mode. For correct operation, the instantiating code must set the CLOCK_RATE parameter (Verilog) or generic (VHDL) to the correct frequency of the CLK port. For example, in Verilog:

```
ZestET2_GigExPhyFIFO #(.CLOCK_RATE(125000000)) GigExPhyInst (
        .CLK(EthClk),
        ...
};
```

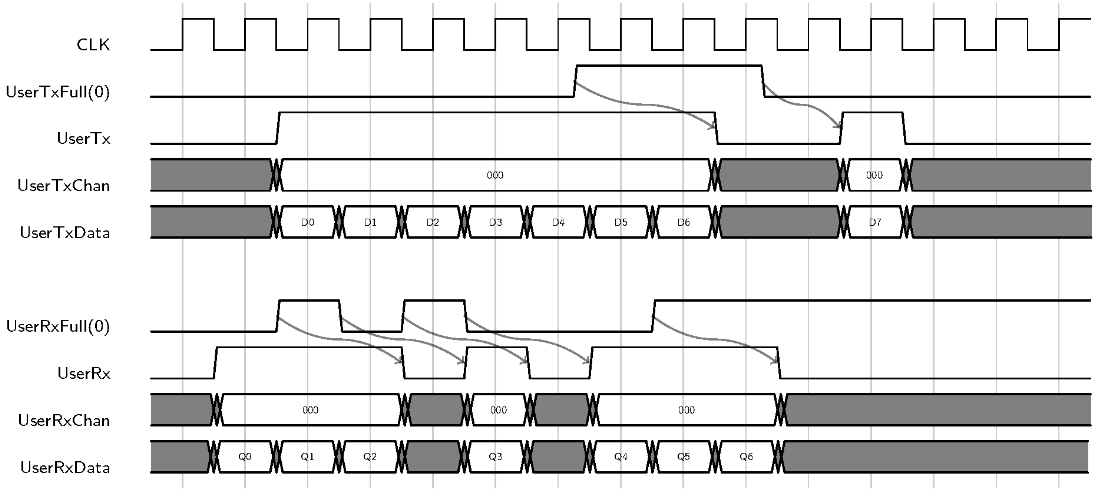
or in VHDL:

```
GigExPhyInst : entity work.ZestET2_GigExPhyFIFO
        generic map (
            CLOCK_RATE => 125000000
        )
        port map (
            CLK => EthClk,
            ...
        );
```

The core has the following ports:

Port Name	Direction	Description
CLK	I	Interface clock. This is the clock that will be used to transfer data between the FPGA and GigEx. All other ports on the core are synchronous to this clock.
GigEx_Clk GigEx_nTx GigEx_TxChan GigEx_TxData GigEx_nTxFull GigEx_nRx GigEx_RxChan GigEx_RxData GigEx_nRxFull GigEx_nInt	O O O O I I I I O I	Interface to GigEx device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the GigEx device.
UserTx UserTxChan UserTxData UserTxFull UserRx UserRxChan UserRxData UserRxFull UserInterrupt	I I I O O O O I O	User interface to/from main FPGA design. To write data to GigEx, UserTxChan and UserTxData should be set to the required values in the same cycle as UserTx is asserted for 1 clock cycle. Data can only be written to GigEx when the bit of UserTxFull corresponding to UserTxChan is low. When GigEx receives data it will set UserRxChan and UserRxData in the same cycle as it asserts UserRx. GigEx will not assert UserRx when the bit of UserRxFull corresponding to the receive data channel is 1 (subject to a latency period). UserInterrupt is an active high signal that will be set when GigEx is generating an interrupt.

An example transaction on the user interface of the core is shown below.



8.5.5 ZestET2_GigExPhyDirect

The ZestET2_GigExPhyDirect core instantiates low level primitives to form the physical interface to the GigEx device when operating in direct mode. For correct operation, the instantiating code must set the CLOCK_RATE parameter (Verilog) or generic (VHDL) to the correct frequency of the CLK port. The TX_BITS and RX_BITS parameters must be set to match the Tx width and Rx width settings in the GigEx configuration web server. For example, in Verilog:

```
ZestET2_GigExPhyDirect
        #(.CLOCK_RATE(125000000), .TX_BITS(16), .RX_BITS(16))
GigExPhyInst (
        .CLK(EthClk),
        ...
};
```

or in VHDL:

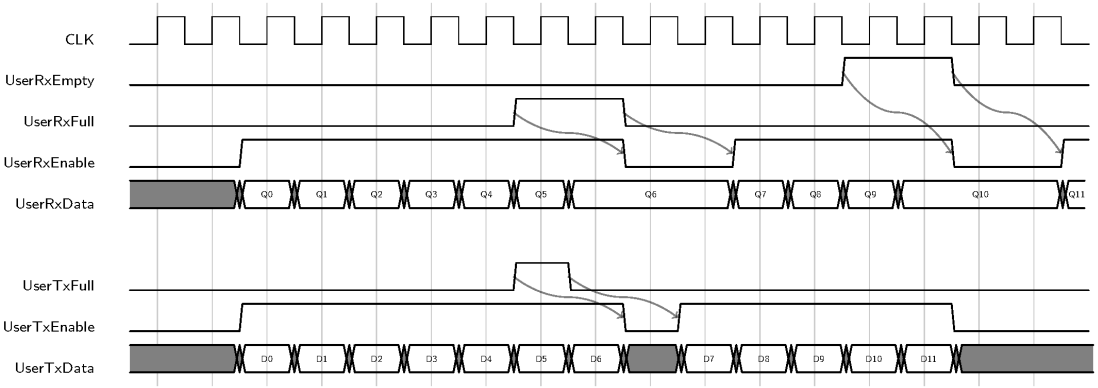
```
GigExPhyInst : entity work.ZestET2_GigExPhyDirect
        generic map (
            CLOCK_RATE => 125000000,
            TX_BITS => 16,
            RX_BITS => 16
        )
        port map (
            CLK => EthClk,
            ...
        );
```

The core has the following ports:

Port Name	Direction	Description
CLK	I	Interface clock. This is the clock that will be used to transfer data between the FPGA and GigEx. All other ports on the core are synchronous to this clock.
GigEx_Clk GigEx_Data GigEx_nTxEnable GigEx_nRxBusy GigEx_nRxEnable	O I/O O O I	Interface to GigEx device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the GigEx device.

GigEx_nRxEmpty GigEx_nTxFull GigEx_nInt	I I I	
UserTxEnable UserTxData UserRxEnable UserRxData UserRxBusy UserRxEmpty UserTxFull UserInterrupt	I I O O I O O O	User interface to/from main FPGA design. To write data to GigEx, UserTxData should be set to the required values in the same cycle as UserTxEnable is asserted for 1 clock cycle. If UserTxFull is high then the outgoing FIFO has filled due to the network not being able to keep up with the data rate and further data will be lost. When GigEx receives data it will set UserRxData in the same cycle as it asserts UserRxEnable. GigEx will not assert UserRxEnable when UserRxBusy is high (subject to a latency period). UserRxEmpty will be set if the receive data underflows indicating that data on UserRxData is not valid. UserInterrupt is an active high signal that will be set when GigEx is generating an interrupt.

An example transaction on the user interface of the core is shown below.



8.5.6 ZestET2_GigExSPI

The ZestET2_GigExPhySPI core is a master SPI controller suitable for driving the slave SPI port on GigEx. For correct operation, the instantiating code must set the CLOCK_DIV parameter (Verilog) or generic (VHDL) to a suitable value to divide the input clock to provide the SPI master clock. The BIT_WIDTH parameter should be set to 8 or 16 depending on the operating mode of the GigEx User Interface (i.e. set to 16 for 16 bit SRAM mode or 8 for all other modes). For example, in Verilog:

```
ZestET2_GigExPhySPI
        #(.CLOCK_DIV(1), .BIT_WIDTH(8))
GigExSPIInst (
        .CLK(EthClk),
        ...
};
```

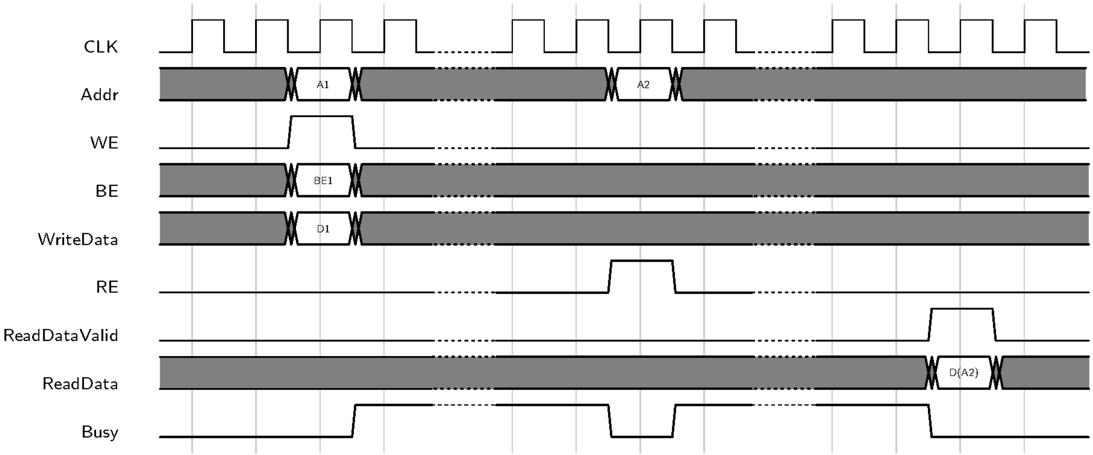
or in VHDL:

```
GigExSPIInst : entity work.ZestET2_GigExSPI
        generic map (
            CLOCK_DIV => 1,
            BIT_WIDTH => 8
        )
        port map (
            CLK => EthClk,
            ...
        );
```

The core has the following ports:

Port Name	Direction	Description
RST	I	Global reset. Should be pulsed high immediately after FPGA configuration.
CLK	I	Interface clock. This is the clock that will be used to transfer data between the FPGA and GigEx. All other ports on the core are synchronous to this clock. The SPI master clock is derived from this clock at a rate specified by the CLK_DIV parameter.
SPI_nCS SPI_SCK SPI_DataOut SPI_DataIn	O O O I	Interface to GigEx device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the GigEx device.
WE BE RE Addr WriteData ReadData ReadDataValid Busy	I I I I I O O O	User interface to/from main FPGA design. To write data to GigEx, Addr, BE and WriteData should be set to the required values in the same cycle as WE is asserted for 1 clock cycle. This should only be done when Busy is low. To read data from GigEx, Addr should be set to the correct value in the same cycle as RE is asserted for 1 clock cycle. This should only be done when Busy is low. When the data is returned from the core, ReadData will contain the data in the same cycle as ReadDataValid is asserted for a single cycle.

An example transaction on the user interface of the core is shown below.

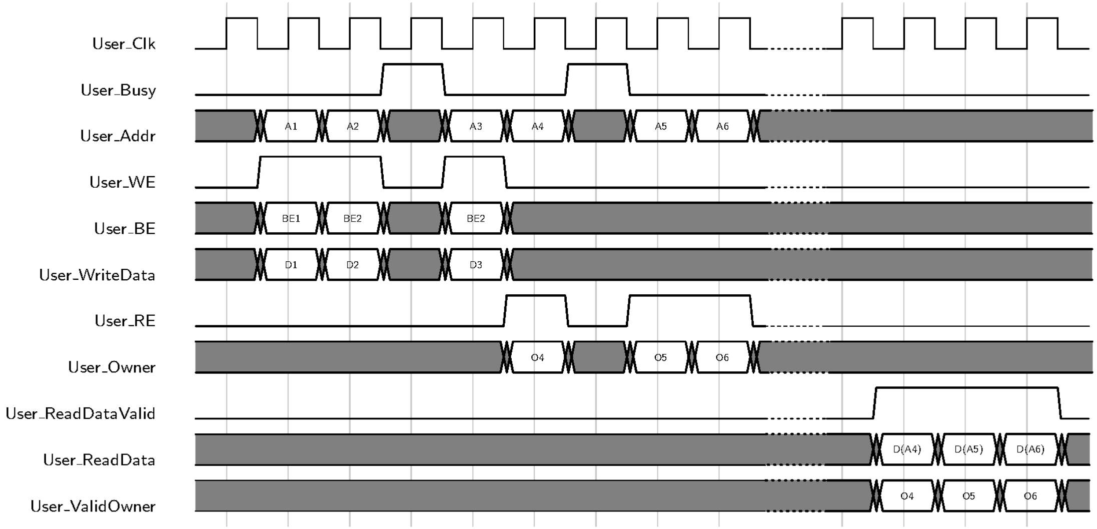


8.5.7 ZestET2_SDRAMTop

The ZestET2_SDRAM directory contains the SDRAM controller core as generated by the Xilinx Memory Interface Generator tool. Please refer to the Xilinx documentation for details of the operation of this core. The ZestET2_SDRAMTop file contains a wrapper around the Xilinx MIG core with the following interface:

Port Name	Direction	Description
RST	I	Global reset. Should be pulsed high immediately after FPGA configuration.
Clk200 Clk50 User_Clk	I I O	Interface clocks. Clk200 is a 200MHz reference clock for the IO logic calibration. This is generated from the ZestET2_Clocks core. Clk50 is the 50Mhz reference clock from the crystal on the ZestET2 board. This must be fed directly from the crystal to avoid adding jitter. User_Clk is an output from the core. All the user interface signals should by synchronous to this clock.
SDRAM_DQ SDRAM_DQS_N SDRAM_DQS_P SDRAM_A SDRAM_BA SDRAM_RASn SDRAM_CASn SDRAM_WEn SDRAM_RSTn SDRAM_CK_P SDRAM_CK_N SDRAM_CKE SDRAM_ODT SDRAM_DM	I/O I/O I/O O O O O O O O O O O O	Interface to SDRAM device. These ports should be connected to the top level IO ports of the design. They should be direct connections to the SDRAM device.
User_Addr User_WE User_RE User_Busy User_BE User_WriteData User_ReadData User_ReadDataValid User_Owner User_ValidOwner User_InitDone	I I I I I I O O I O O	User interface to/from main FPGA design. To write data to the SDRAM, User_Addr, User_BE and User_WriteData should be set to the required values in the same cycle as User_WE is asserted for 1 clock cycle. This should only be done when User_Busy is low. To read data from the SDRAM, User_Addr should be set to the correct value in the same cycle as User_RE is asserted for 1 clock cycle. This should only be done when User_Busy is low. When the data is returned from the core, User_ReadData will contain the data in the same cycle as User_ReadDataValid is asserted for a single cycle. User_Owner can be set to any value in the cycle that User_RE is asserted. If this is the case then User_ValidOwner will be set to the same value in the cycle of the corresponding User_ReadValid assertion. In this way the FPGA code can match data returned from the SDRAM with the access that requested the data. User_InitDone will be low until the initialisation of the SDRAM controller is complete. No accesses should be attempted on the user interface until this signal goes high.

An example transaction on the user interface of the core is shown below.



8.6 Building Designs With Xilinx Vivado

FPGA designs can be built with the Xilinx Vivado design tools. The FPGA is supported by the WebPack edition of Vivado which is available as a free download from the Xilinx website. No special options are required to build designs for the ZestET2 FPGA. The examples designs on the installation CD (see section 9.4 for details) can be used as starting points for developing new FPGA projects in Vivado.

9 Host Software

The software package CD includes a Windows Installer utility and a Linux .tgz file containing the support files. On Windows, to install the software, run the setup.exe utility from the ZestET2 Support Software CD and follow the on-screen instructions. On Linux, extract the .tgz file to a suitable location on your hard disk.

The software will be installed in the following folder structure:

Folder	Contents
Constraints	Xilinx Vivado constraint files containing pinout details for FPGA designs.
CPU	User CPU tools and libraries
Documents	Product documentation including the User Guide (this document)
Examples	Examples for host and FPGA code showing how to use various features of the ZestET2
Inc	Host support library C include file
Lib	Host support library C static library file
DLL	Host support library C dynamic library file
License	Copies of the software license agreements
Utils	Contains utility to automatically detect and control ZestET2 boards on the network (CPanel.exe) and a program to convert FPGA .bit files to C code for embedding in host application.
Source	Example HTML code for the embedded web server
Verilog	Verilog source for the FPGA interface cores
VHDL	VHDL source for the FPGA interface cores

The ZestET2 ships with the following default network settings. Depending on your network configuration, you may need to alter the settings of the ZestET2 board. The ZestET2 should initially be plugged in to a network port which can communicate with the default settings which may require temporarily altering the network settings of your host PC.

These settings can be programmed using a built-in web server. See Section 6.5 for details of how to change these settings.

Setting	Default	Description
Fixed IP	192.168.1.100	Fixed IP address of the GigEx device.
Subnet	255.255.255.0	Fixed subnet setting.
Gateway	192.168.1.1	Fixed gateway setting.
DHCP	Disabled	DHCP client disabled.
AutoIP	Disabled	AutoIP client disabled.
HTTP port	80	Port for configuration web server.
Control port	20481	Port for control of GigEx chip.
Jumbo frame	1500	Longest transmit frame size (in bytes).
Password	Disabled	Administration password disabled.

Plug the module into the carrier board connected to the network (either your own custom board or the ZestET2-NJ Breakout Board, see Appendix A), and power it up. You should now attempt to open the configuration web server at address http://192.168.1.100 and, if possible, run the example programs to confirm correct operation of the ZestET2.

9.1 Ethernet Connection Troubleshooting

If you cannot browse to the configuration server or the control panel utility CPanel.exe does not detect the board then the following steps can be used to establish communications.

1. The ZestET2 ships with a default IP address of 192.168.1.100. The PC used to communicate with the ZestET2 must have an IP address in the same subnet. This means that the PC IP address must be set to 192.168.1.x where x is any number between 1 and 255 excluding 100. On a Windows PC, the IP address of a network port can be set from the 'Network Connection' dialog by double clicking the appropriate connection, selecting properties, then 'Internet Protocol (TCP/IP)' and properties. The IP address should be set along with a subnet mask of 255.255.255.0.

   Once communications have been established with the ZestET2, the IP address of the board can be modified to any suitable value for your network (see Section 6.5). Following this, the PC IP address can be changed back to its original value.

2. You should ensure that your firewall is set to allow communications between the PC and the ZestET2. Please refer to your firewall documentation for instructions on allowing access to remote devices. If in doubt, disable your firewall until communications with the ZestET2 have been established.

3. Check that you can 'ping' the ZestET2 board. Open a command prompt (from the Start Menu, type cmd in 'Search programs and files' and select cmd.exe). Then type:

   ```
   ping 192.168.1.100
   ```

   You should see a response from the ZestET2. If not, double check stages 1-3 above.

4. Check that you can browse to the GigEx configuration web page by opening the following address in your web browser:

   http://192.168.1.100

5. Check that the ZestET2 Control Panel can detect the ZestET2. The Control Panel is located in the Start Menu under Orange Tree Technologies and ZestET2. The Control Panel uses UPnP M-SEARCH messages sent to UDP port 1900 on multicast address 239.255.255.250 to detect the modules. Your network firewalls and routers must be set up to allow a path from your PC to the ZestET2 allowing these packets through for the Control Panel to detect the board.

9.2 Bit2C

Bit2C.exe converts Xilinx .bit FPGA configuration files to C header files. The C header file contains a static array definition with the raw data from the .bit file. This array can be passed to the **ZestET2RegisterImage** function to obtain a handle suitable for the **ZestET2Configure** function. In this way, .bit files can be linked into your application executable to avoid having multiple files.

For example:

```
Bit2C config.bit array.c
```

This will convert the **config.bit** file generated by the Xilinx place and route tools into a file called **array.c** which contains definitions of the variables **arrayLength** and **arrayImage**. You can then configure the FPGA by calling the following functions:

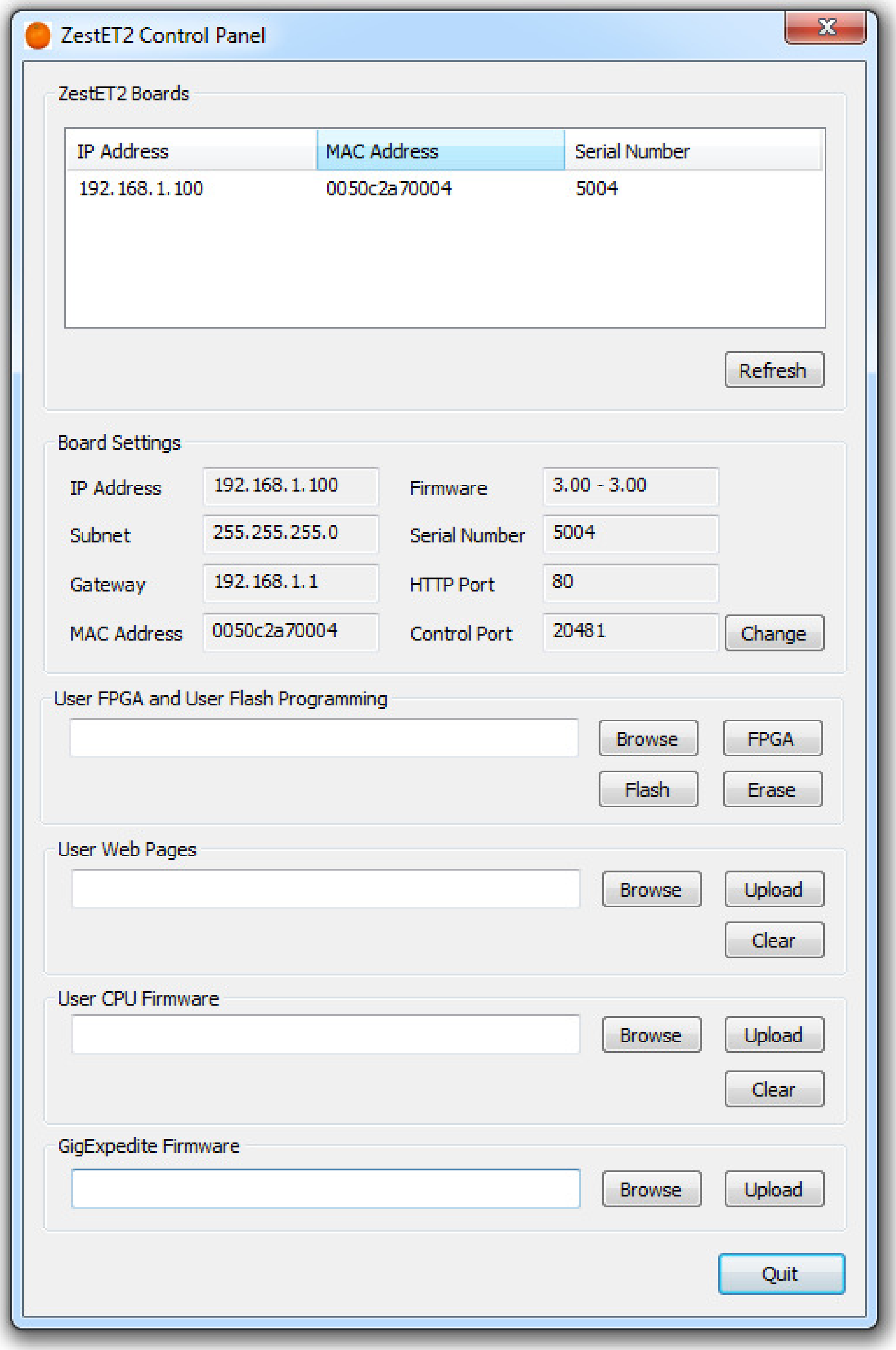
```
extern void *arrayImage;
extern unsigned long arrayLength;
ZESTET2_IMAGE Image;
ZESTET2_HANDLE Handle;

/* Register the FPGA configuration image */
ZestET2RegisterImage(arrayImage, arrayLength, &Image);

/* Configure the FPGA from the image */
ZestET2Configure(Handle, Image);
```

9.3 Control Panel Application (CPanel)

The ZestET2 Software Support CD includes a Windows application to help detect and control ZestET2 modules on the network. CPanel is not supported under Linux. A screenshot of the application is shown in Figure 26. The top window shows all the ZestET2 cards that have been detected on the network. Note that CPanel will scan all networks on the PC using UPnP messages but the network settings of the PC port and the ZestET2 module must be compatible (i.e. they must be on the same subnet).



ZestET2 Control Panel

ZestET2 Boards

IP Address	MAC Address	Serial Number
192.168.1.100	0050c2a70004	5004

Board Settings

IP Address	192.168.1.100	Firmware	3.00 - 3.00
Subnet	255.255.255.0	Serial Number	5004
Gateway	192.168.1.1	HTTP Port	80
MAC Address	0050c2a70004	Control Port	20481

User FPGA and User Flash Programming

User Web Pages

User CPU Firmware

GigExpedite Firmware

Figure 26. Screenshot of the CPanel utility

Selecting a board from the list of detected boards will populate the information fields with details about that board to aid in identification. Clicking 'Change' will open the web configuration server to allow you to change the board settings.

The user FPGA can be configured by clicking 'Browse' in the 'User FPGA and User Flash Programming' section and selecting a suitable Xilinx FPGA .bit file. Clicking 'FPGA' will configure the FPGA. Clicking 'Flash' will write the .bit file to user flash memory so that the FPGA will configure automatically on power up. Clicking 'Erase' will erase the user FPGA flash.

The default user web pages can be changed to suit the user application by uploading replacement HTML pages. The 'User Web Pages' box should be populated with a directory containing the replacement web pages. CPanel will scan the directory and any sub-directories and upload any files with the extension '.htm', '.html', '.js', '.css', '.jpg', '.gif' or '.png'. The root directory must contain a file called 'index.html' which will be the home page of the web site. Clicking 'Upload' will upload the web pages to the flash on the ZestET2 board replacing the default pages. Clicking 'Reset' will remove the custom web pages and revert to the default web pages. The default web pages are supplied as source on the CD to provide a starting point for development. See section 6.5 for details of how to design custom web pages for the embedded web server.

The ZestET2 board has 256kbytes of flash available for a user CPU application to be uploaded to execute at power up. Click 'Browse' under User CPU firmware to select the ELF file to upload. (ELF files are the output file from the GCC compiler tools). Click 'Upload' to upload the file to flash and click 'Clear' to remove an existing program from flash. The program will execute when GigEx next powers up.

CPanel also provides the mechanism for uploading replacement firmware into the GigEx flash. Any replacement firmware will be supplied by Orange Tree Technologies and firmware should only be uploaded under the instruction of the Orange Tree Support Department.

During firmware update, the external device should not attempt to access the GigEx device to prevent upsetting the update procedure. Also, the power supply to GigEx must remain on until the update is complete to prevent corruption of the firmware.

GigEx contains a fallback copy of firmware which will be used if it detects that the firmware update failed resulting in corrupted firmware. When fallback firmware is being used, the firmware version number will have a suffix of 'F' (e.g. '3.0F - 3.0F'). The fallback firmware is limited in functionality so the full firmware update should be re-attempted as soon as possible after entering this state.

CPanel detects the ZestET2 boards using UPnP multicast M-SEARCH messages. GigEx will respond to suitable M-SEARCH messages with a UDP packet describing the IP address of the GigEx device. By parsing this response, a host application can discover and communicate with all GigEx devices on the network.

9.4 Example Programs

The ZestET2 Software Support CD contains a number of example programs illustrating the use of the ZestET2 module in each of its modes.

9.4.1 Example 1: 16 Bit SRAM mode TCP/IP transfers

Example 1 illustrates use of the ZestET2 module in 16 bit SRAM mode. The GigEx user interface must be set to 16 bit SRAM mode with the clock as 'Input From User' from the web configuration server before running this example. The GigEx device is initialised by the FPGA as a TCP server listening on a single port. The host application makes a connection to the ZestET2 at which point the FPGA sends 500 MBytes of ramp data to the host and receives 500 MBytes of ramp data from the host. The transfer in each direction is timed and the data transfer rate is reported.

The FPGA application is supplied in Verilog and VHDL along with project files for the Xilinx Vivado synthesis and place and route software. The C host program is supplied as a Visual C++ project for Windows and with a makefile for Linux compilation.

The FPGA application consists of a state machine which could form the basis of an FPGA only application transferring data across a network without a user processor in the system.

9.4.2 Example 2: 8 Bit SRAM mode TCP/IP transfers and DDR3 SDRAM

Example 2 illustrates the use of the ZestET2 module in 8 bit SRAM mode. It also illustrate the use of the DDR3 SDRAM. The GigEx user interface must be set to 8 bit SRAM mode with the clock as 'Input From User' from the web configuration server before running this example.

Example 2 is similar to Example1 but the data is passed through the DDR3 SDRAM buffer before being written to the network. The example illustrates the use of the Xilinx MIG DDR3 SDRAM controller core in a real design.

The FPGA application is supplied in Verilog and VHDL along with project files for the Xilinx Vivado synthesis and place and route software. The C host program is supplied as a Visual C++ project for Windows and with a makefile for Linux compilation.

9.4.3 Example 3: FIFO mode UDP transfers

Example 3 illustrates the use of the ZestET2 module as a UDP server with the user interface in FIFO mode. The GigEx user interface must be set to FIFO mode with the clock as 'Input From User' from the web configuration server before running this example. The FPGA application opens a connection to receive data from the network. When a single byte packet containing a payload of '1' is received, it opens a second channel to transmit UDP data back to the device that sent the control packet. When a single byte packet containing a payload of '0' is received on the control channel the data channel is closed.

The FPGA application uses the slave SPI interface to program the GigEx registers as the registers are not accessible via the high speed interface when in FIFO mode.

The host application sends a start byte packet to the control channel before receiving data from the data channel. The data transfer is timed and the transfer rate is reported.

The FPGA application is supplied in Verilog and VHDL along with project files for the Xilinx Vivado synthesis and place and route software. The host program is supplied as a Visual C++ project for Windows and with a makefile for Linux compilation.

The FPGA application consists of a state machine which could form the basis of an FPGA only application transferring data across a network without a user processor in the system.

9.4.4 Example 4: Direct mode and Auto-open

Example 4 illustrates the use of the ZestET2 module in direct mode along with the auto-open feature for connection management. The FPGA is configured to loop back 16 output pins from the user interface to 16 input pins into the user interface. The data transmit strobe is a simple combination of the receive data strobe and the transmit full flag. The example is designed to illustrate how the direct mode can be used to connect gluelessly to external devices.

The User interface must be set in Direct mode with the clock set to 'Input from User'. The 'Receive From Net Width' and 'Transmit To Net Width' should both be set to 2 bytes.

The auto-open feature of GigEx can be used to manage network connections automatically. For this example, a single channel is configured as a TCP server listening on a port. When the host application connects to GigEx, data can be transferred across the high speed interface. When the host application closes the connection, GigEx immediately starts listening for the next connection on the port.

For this example, the GigEx web server must be used to initialise the auto-open channel with the following settings:

Channel 0:

Auto	Enabled
TCP	Enabled
Server	Enabled
Local Port	5002
TTL	128
UART	Disabled

The host application opens a TCP connection to the ZestET2 module and sends data to it. It then receives the loopback data and checks that it is correct.

The FPGA application is supplied in Verilog and VHDL along with project files for the Xilinx synthesis and place and route software. The host program is supplied as a Visual C++ project for Windows and with a makefile for Linux compilation.

9.4.5 Example 5: User CPU TCP/IP transfers

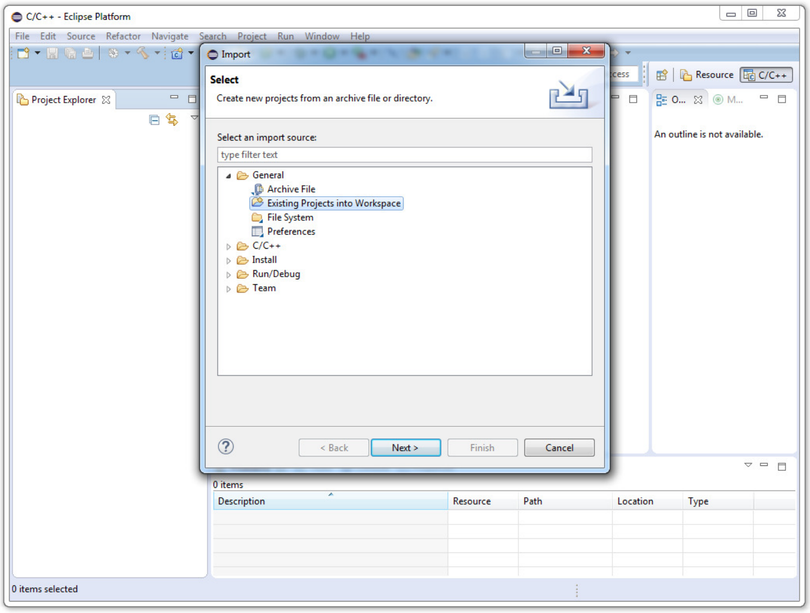
Example 5 illustrates the use of the User CPU in GigEx. The FPGA is not used in this example which runs a very simple web server to shown how to open connections and transfer data using the user CPU.

The example includes starting point code for standard socket functions that interfaces to the high speed network engine in GigEx. It opens a web server on port 8080 which just returns an example text string when a web browser open http://192.168.1.100:8080

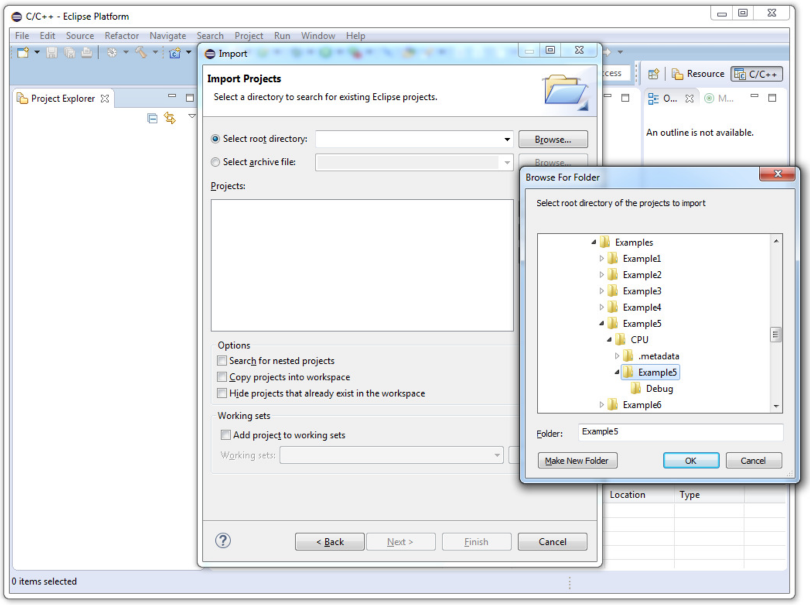
The example is supplied in the form of an Eclipse project file with source code. It can be downloaded and run from the Eclipse development environment via GDB or uploaded into the flash on the ZestET2 module where is will execute at every power on.

To open the example, first start Eclipse from the Start Menu. When prompted for a workspace, choose a new location to store your workspace files (refer to the Eclipse documentation at www.eclipse.org for details of workspaces). This will create a new, blank workspace. Form the Eclipse menu, choose 'Window' then 'Open Perspective' and 'C/C++'. Then close the Welcome window by clicking on the cross on the 'Welcome' tab.

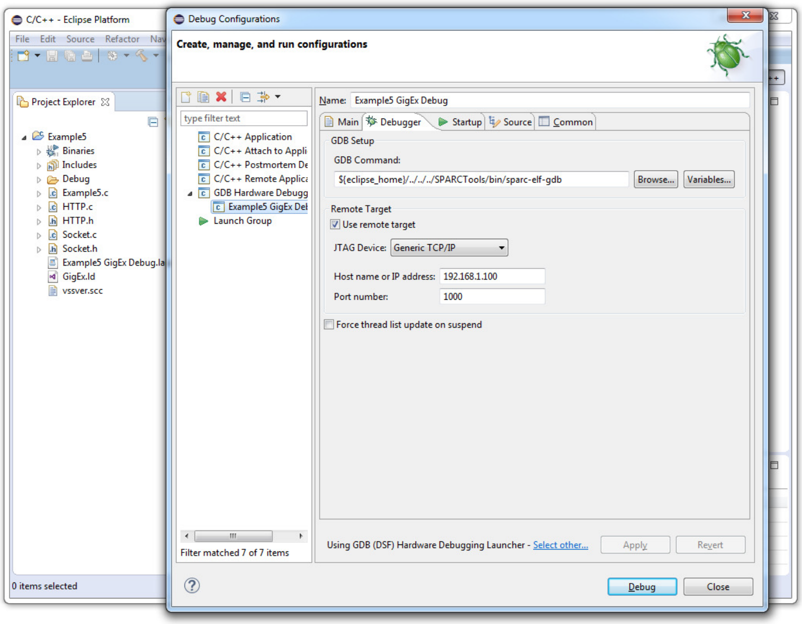
To import the example into your new workspace, choose 'File' and 'Import' from the Eclipse menu. Then choose 'General' and 'Existing Projects into Workspace' followed by 'Next' as shown below.



Then choose 'Select root directory' and browse to the Examples\Example5\CPU\Example5 directory on your PC and click 'Finish' as shown below:



You should now have an open project in your Eclipse environment. To run the project, you must use the configuration web server on GigEx to set the interface mode to 8 bit SRAM and the Interface clock direction to 'Output to user'. (The FPGA is not configured and there must always be a clock present on the user interface). You can download and run the example on the board by selecting 'Run' and 'Debug Configurations' from the Eclipse menu. Then select the Example5 debug configuration and enter the correct IP address and GDB server port of the ZestET2 as shown below:



Debug Configurations

Create, manage, and run configurations

Name: Example5 GigEx Debug

GDB Setup

GDB Command: ${eclipse_home}/../../../SPARCTools/bin/sparc-elf-gdb

Remote Target

☑ Use remote target

JTAG Device: Generic TCP/IP

Host name or IP address: 192.168.1.100

Port number: 1000

☐ Force thread list update on suspend

Using GDB (DSF) Hardware Debugging Launcher - Select other...

Filter matched 7 of 7 items

0 items selected

When you press 'Debug', GDB will download the application to the user CPU and start execution. Execution will halt at the first line of the main function. To continue execution, press F8. You should now be able to browse to http://192.168.1.100:8080 in your web browser and see the results of the web server running in the user CPU.

It is also possible to use the CPanel application to upload the example 5 ELF file to the the flash on ZestET2 so that it executes on power up. Refer to section 9.3 for details of the CPanel application.

9.4.6 Example 6: Network Attached Storage Data Logger

Example 6 illustrates the use of the User CPU in GigEx alongside the FPGA to implement complex higher level protocols on ZestET2 without compromising data transfer rates. The application is a high speed data logger that writes data from the FPGA to a network attached disk. The User CPU is used to handle complex low bandwidth operations such as logging in to the disk server and opening and closing files. When a file is opened, the CPU builds the write command headers before handing control to the FPGA to write the data directly to the network.

Before running example6, the web configuration server must be used to set GigEx into FIFO mode with the interface clock direction set to 'Input from user'. The CPU project can be opened by following the same import procedure as example5 above. The settings at the top of Example6.c must be edited to point to the correct NAS disk server.

The FPGA program must be downloaded to the FPGA either using the CPanel application (section 9.3) or using a Xilinx JTAG pod. The user CPU program can then be run from within Eclipse using the GDB debugger as described in the section above about example 5.

When the program is running, it will write a ramp data file to the NAS disk for 10 seconds before closing the file.

It is also possible to program the FPGA BIT file and Example 6 ELF file into flash so that they execute and start recording on power up of the ZestET2.

A possible optimisation to increase the bandwidth of recording would be to build the CIFS write command headers in the FPGA instead of in the CIFSWriteFileHighSpeed function. The CPU would still handle the complex login and file creation protocols leaving the FPGA to simply build a CIFS write command header and handle the status response from the NAS.

9.5 Host Library Functions

The ZestET2 host support software includes a host library to allow access to the board. The host library consists of a static or dynamic C library file (.lib or .dll file for windows, .a file for Linux) and an associated C header file (.h file). Both 32 and 64 bit versions of the library are provided on the installation CD.

To use the ZestET2 support library in your own code, you must include the header file at the start of your program. For example:

```
#include <ZestET2.h>
```

The header file must be in your compiler include path. For details of how to set the include path, refer to your compiler manuals.

Your program must then be linked with the ZestET2.lib (Windows) or libZestET2.a (Linux) library file. Windows applications must also be linked with the standard Windows sockets library named ws2_32.lib which should ship with your complier package. For details of how to link additional static libraries, refer to your compiler manuals.

For details of the functions in the user library, see below.

9.5.1 Host Library Function Reference

ZestET2Init

ZESTET2_STATUS ZestET2Init(void);

Parameters

None

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_SOCKET_ERROR	Error initializing low level socket library

Description

ZestET2Init initializes the ZestET2 host library. It must be called before any of the other ZestET2 functions can be called.

ZestET2Close

ZESTET2_STATUS ZestET2Close(void);

Parameters

None

Return Value

ZESTET2_SUCCESS	Function succeeded

Description

ZestET2Close closes the ZestET2 library and must be called after all other ZestET2 functions have been called.

ZestET2CountCards

ZESTET2_STATUS ZestET2CountCards(unsigned long ****NumCards*,**
ZESTET2_CARD_INFO *****CardInfo*,**
int *Wait***);**

Parameters

NumCards	Pointer to location to receive total number of ZestET2 cards detected in the system.
CardInfo	Pointer to buffer to receive list of card descriptions in the system. May be NULL.
Wait	Number of milliseconds to wait for ZestET2 devices to respond to the discovery request. This wait is per Ethernet interface on the PC.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete operation

Description

ZestET2CountCards can be used to determine the number and types of cards on the network. It uses UPnP messages to query how many ZestET2 devices are present and then queries the device characteristics and settings.

CardInfo should point to a pointer to receive the start of the list of card information. The returned buffer is an array of **ZESTET2_CARD_INFO** structures, one for each card detected. This pointer must be passed to **ZestET2FreeCards** to deallocate the memory.

Wait should specify the delay for a ZestET2 to respond to the UPnP messages. If the network infrastructure is slow or busy or there are many network hops to the ZestET2 device then it may be necessary to increase this value. Normally a setting of 1000 milliseconds is adequate. If the PC has more than one Ethernet interface, the scan is performed on each interface in turn and this delay applies to each Ethernet port.

The **ZESTET2_CARD_INFO** structure used to describe a card is as follows:

```
typedef struct
{
    unsigned char IPAddr[4];
    unsigned short ControlPort;
    unsigned long Timeout;
    unsigned short HTTPPort;
    unsigned char MACAddr[6];
    unsigned char Subnet[4];
    unsigned char Gateway[4];
    unsigned long SerialNumber;
    unsigned long FirmwareVersion;
    unsigned long HardwareVersion;
    unsigned char GUID[16];
```

```
        } ZESTET2_CARD_INFO;
```

IPAddr is the current IP address of the GigEx device on the card.

ControlPort is the control port used by the GigEx to receive control messages. This defaults to 0x5001 but can be changed to free the port for user applications. See Section 6.4 for details.

Timeout is the length of time, in milliseconds, that blocking operations should wait for before returning **ZESTET2_TIMEOUT**. Time outs allow the user program to recover cleanly when communication with the card fails.

HTTPPort is the current web server port of the GigEx device on the card.

MACAddr is a unique, factory-set physical network address (MAC address) for the card.

Subnet is the current subnet mask of the GigEx device on the card.

Gateway is the current IP address of the gateway used by the GigEx device on the card.

SerialNumber is a unique, factory-set serial number for the card.

FirmwareVersion gives the current version of the firmware programmed into the GigEx device. This is multiplied by 100 so a value of 201 corresponds to a firmware version of 2.01. If bit 15 is set (i.e. mask of 0x8000) then the GigEx is operating in fallback mode following a failed firmware upgrade.

HardwareVersion gives the current version of the hardware for this ZestET2. This is multiplied by 100 so a value of 201 corresponds to a firmware version of 2.01. If bit 15 is set (i.e. mask of 0x8000) then the GigEx is operating in fallback mode following a failed firmware upgrade.

GUID is the user programmable unique ID setting for the board. The unique ID can be used to identify user product sub-types. For example, if the ZestET2 is used in two different products the GUID can be set differently for each product so the host software can identify each product type.

This data structure is used to identify a card to other ZestET2 functions. If the IP address and control port of a device is already known then there is no need to call **ZestET2CountCards**. In this case, the `IPAddr`, `ControlPort` and `Timeout` fields of a **ZESTET2_CARD_INFO** structure can be initialized directly before being passed to any of the other ZestET2 functions. The other fields are for information only and are not used by the other ZestET2 functions.

Example use:

```
        unsigned long NumCards;
        ZESTET2_CARD_INFO *Cards;

        /* Get the number of cards in the system */
        ZestET2CountCards(&NumCards, &Cards, 1000);

        ...

        /* Free the card information structure */
        ZestET2FreeCards(Cards);
```

ZestET2GetCardInfo

ZESTET2_STATUS ZestET2GetCardInfo(ZESTET2_CARD_INFO ****Info*);**

Parameters

Info	Pointer to structure to receive information about the card.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	The *Info* parameter is NULL
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_SOCKET_ERROR	Error communicating with low level socket library
ZESTET2_SOCKET_CLOSED	Low level socket was closed unexpectedly
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2GetCardInfo can be used to obtain information about a card. This information is normally returned by the **ZestET2CountCards** function. However, if the device IP address and control port are already known this function can be used to query the card and fill in the remaining fields. See the description of **ZestET2CountCards** for further details of the **ZESTET2_CARD_INFO** data structure.

On entry, the `IPAddr`, `ControlPort` and `Timeout` fields of the *Info* structure must be valid. On successful completion, all remaining fields will also be valid.

For example:

```
ZESTET2_CARD_INFO Info;

/* Fill in known fields */
Info.IPAddr[0] = 192;
Info.IPAddr[1] = 168;
Info.IPAddr[2] = 1;
Info.IPAddr[3] = 100;
Info.ControlPort = 0x5001;
Info.Timeout = 1000;

/* Obtain remaining information about the card */
ZestET2GetCardInfo(&Info);
```

ZestET2FreeCards

ZESTET2_STATUS ZestET2FreeCards (ZESTET2_CARD_INFO ****Info*);**

Parameters

Info	Pointer to structure containing card information.

Return Value

ZESTET2_SUCCESS	Function succeeded

Description

ZestET2FreeCards can be used to free the card information data structure returned by the **ZestET2CountCards** function.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *Cards;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 1000);

...

/* Free the card information structure */
ZestET2FreeCards(Cards);
```

ZestET2RegisterErrorHandler

ZESTET2_STATUS ZestET2RegisterErrorHandler(
ZESTET2_ERROR_FUNC *Function***);**

Parameters

Function	Pointer to error handler function to receive all error notifications for this application.

Return Value

ZESTET2_SUCCESS	Function succeeded

Description

ZestET2RegisterErrorHandler can be called to install a central error handling routine for a user program. If any of the following ZestET2 library function calls fail, the user error handler will be called giving details of the failure. This mechanism means that status codes need not be checked for every library call which simplifies code considerably.

ZESTET2_ERROR_FUNC is a function declared as follows:

```
typedef void (*ZESTET2_ERROR_FUNC)(const char *Function,
                                   ZESTET2_CARD_INFO *CardInfo,
                                   ZESTET2_STATUS Status,
                                   const char *Msg);
```

Function is a string containing the name of the function that failed. *CardInfo* is the card information structure of the card being accessed at the time of the failure. This may be NULL for functions that do not relate to a single card. *Status* is the status code describing the failure and *Msg* is a string describing the failure.

For example:

```
/* Error handler function */
void ErrorHandler(const char *Function,
                  ZESTET2_CARD_INFO *CardInfo,
                  ZESTET2_STATUS Status, const char *Msg)
{
       printf("**** Function %s returned an error\n", Function);
       printf("**** 0x%08lx : \"%s\"\n\n", Status, Msg);
       exit(1);
}

void main(void)
{
       ZestET2RegisterErrorHandler(ErrorHandler);

       /* Other calls to ZestET2 library here */
       /* Note that the return code need not be checked
          as ErrorHandler will be called for any return values
          not equal to ZESTET2_SUCCESS */
}
```

ZestET2GetErrorMessage

ZESTET2_STATUS ZestET2GetErrorMessage(ZESTET2_STATUS *Status*,
char *****Buffer*);**

Parameters

Status	ZestET2 error code for which description is required.
Buffer	Pointer to location to receive error code description string.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_STATUS_CODE	Status code is out of range

Description

ZestET2GetErrorMessage can be called to obtain a descriptive message for a return status from a ZestET2 library function. On return, the string pointed to by *Buffer* will be a description of the *Status* return code. This is a static string and so does not need to be freed.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *CardInfo;

/* Find cards on the network */
Status = ZestET2CountCards(&NumCards, &CardInfo, 2000);

if (Status!=ZESTET2_SUCCESS)
{
        char *Buffer;

        ZestET2GetErrorMessage(Status, &Buffer);
        printf("Error : %s\n", Buffer);
}
```

ZestET2ConfigureFromFile

ZESTET2_STATUS ZestET2ConfigureFromFile(ZESTET2_CARD_INFO ****CardInfo*,**
char ****FileName*);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
FileName	Name of .bit file to use to configure the FPGA.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_FILE_NOT_FOUND	File not found
ZESTET2_FILE_ERROR	Error while reading file
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the requested operation
ZESTET2_ILLEGAL_FILE	File format is not recognised
ZESTET2_INVALID_PART_TYPE	Illegal FPGA part type
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2ConfigureFromFile can be used to configure the user FPGA on the ZestET2 card from a .bit file generated by the Xilinx place and route software. It configures the FPGA directly from the file on disk. Refer to **ZestET2RegisterImage** and **ZestET2Configure** for details of how to configure the FPGA from configuration data in memory.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *CardInfo;

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Configure the FPGA from a .bit file */
ZestET2ConfigureFromFile(&(CardInfo[0]), "test.bit");
```

! **Configuring the FPGA with an incorrect BIT file can damage your hardware. Ensure that FPGA pins are connected correctly and do not drive against peripherals on the board.**

ZestET2LoadFile

ZESTET2_STATUS ZestET2LoadFile(**char ****FileName***,**
ZESTET2_IMAGE ****Image*);**

Parameters

FileName	Name of .bit file to use to configure the FPGA.
Image	Pointer to location to receive FPGA configuration image.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_FILE_NOT_FOUND	File not found
ZESTET2_FILE_ERROR	Error while reading file
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the requested operation
ZESTET2_ILLEGAL_FILE	File format is not recognised
ZESTET2_INVALID_PART_TYPE	Illegal FPGA part type

Description

ZestET2LoadFile can be used to load a .bit file generated by the Xilinx place and route software into memory for future configuration of the FPGA on the ZestET2 card. It is possible to load many configuration files during initialization and select the correct FPGA image to use during execution of the program. This reduces overhead at configuration time.

Image is a pointer to a location to receive the configuration image handle. This handle can be used in future calls to **ZestET2Configure**. The handle should be freed by calling **ZestET1FreeImage** when the configuration image is no longer needed.

ZestET1LoadFile and **ZestET1Configure** allow decoupled loading of the bit file and configuration of the FPGA. Refer to **ZestET1RegisterImage** and **ZestET1Configure** for details of how to configure the FPGA from configuration data in memory.

For example:

```
unsigned long NumCards;
ZESTET1_CARD_INFO *CardInfo;
ZESTET1_IMAGE Image;

/* Load the .bit file */
ZestET1LoadImage(“test.bit”, &Image);

/* Other initialization operations here */

/* Find cards on the network */
ZestET1CountCards(&NumCards, &CardInfo, 2000);
```

```
/* Other execution operations here */

/* Configure the FPGA from the image */
ZestET1Configure(&(CardInfo[0]), Image);
```

ZestET2Configure

ZESTET2_STATUS ZestET2Configure(ZESTET2_CARD_INFO ****CardInfo*,**
ZESTET2_IMAGE *Image***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Image	FPGA configuration image to use to configure the FPGA.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_IMAGE_HANDLE	Attempt to use illegal configuration image handle
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2Configure configures the user FPGA on the ZestET2 card from a configuration image. The configuration image can be created from a .bit file by calling **ZestET2LoadImage** or from data in memory by calling **ZestET2RegisterImage**.

Example creating the image from .bit file on disk:

```
ZESTET2_IMAGE Image;
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;

/* Load the .bit file */
ZestET2LoadImage("test.bit", &Image);

/* Other initialization operations here */

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Other execution operations here */

/* Configure the FPGA from the image */
ZestET2Configure(&(CardInfo[0]), Image);
```

Example creating the image from buffer in memory:

```
extern void *Buffer;
extern unsigned long Length;
ZESTET2_IMAGE Image;
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
```

```
/* Register the FPGA configuration image */
ZestET2RegisterImage(Buffer, Length, &Image);

/* Other initialization operations here */

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Other execution operations here */

/* Configure the FPGA from the image */
ZestET2Configure(&(CardInfo[0]), Image);
```

! **Configuring the FPGA with an incorrect BIT file can damage your hardware. Ensure that FPGA pins are connected correctly and do not drive against peripherals on the board.**

ZestET2RegisterImage

ZESTET2_STATUS ZestET2RegisterImage(void ****Buffer*,**
unsigned long *BufferLength***,**
ZESTET2_IMAGE ****Image*);**

Parameters

Buffer	Buffer containing FPGA configuration data. Normally generated by Bit2C utility.
BufferLength	Length, in bytes, of the configuration data.
Image	Pointer to location to receive FPGA configuration image.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the requested operation
ZESTET2_INVALID_PART_TYPE	Illegal FPGA part type

Description

ZestET2RegisterImage creates an FPGA configuration image from an array of raw configuration data. The FPGA can be configured from this image by calling **ZestET2Configure**.

Buffer points to the start of the raw configuration data and *BufferLength* is the number of bytes of configuration data to use. The Bit2C utility supplied on the ZestET2 installation disk provides a simple way to generate compatible C arrays from .bit files.

Image is a pointer to a location to receive the configuration image handle. This handle can be used in future calls to **ZestET2Configure**. The handle should be freed by calling **ZestET2FreeImage** when the configuration image is no longer needed.

ZestET2RegisterImage and **ZestET2Configure** allow .bit files to be linked into host executables to reduce the number of host files required. Refer to **ZestET2ConfigureFromFile**, **ZestET2LoadImage** and **ZestET2Configure** for details of how to configure the FPGA from configuration data in a file.

For example:

```
extern void *Buffer;
extern unsigned long Length;
ZESTET2_IMAGE Image;
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;

/* Register the FPGA configuration image */
ZestET2RegisterImage(Buffer, Length, &Image);

/* Other initialization operations here */
/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Other execution operations here */

/* Configure the FPGA from the image */
ZestET2Configure(&(CardInfo[0]), Image);
```

ZestET2FreeImage

ZESTET2_STATUS ZestET2FreeImage(ZESTET2_IMAGE *Image***);**

Parameters

Image	FPGA configuration image to free.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_IMAGE_HANDLE	Attempt to use illegal configuration image handle

Description

ZestET2FreeImage should be called when a configuration image handle is no longer needed. It is used to free resources allocated during **ZestET2LoadImage** and **ZestET2RegisterImage** functions.

For example:

```
extern void *Buffer;
extern unsigned long Length;
ZESTET2_IMAGE Image;
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;

/* Register the FPGA configuration image */
ZestET2RegisterImage(Buffer, Length, &Image);

/* Other initialization operations here */

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Other execution operations here */

/* Configure the FPGA from the image */
ZestET2Configure(&(CardInfo[0]), Image);

/* Free resources associated with the handle */
ZestET2FreeImage(Image);
```

ZestET2ProgramFlashFromFile

ZESTET2_STATUS ZestET2ProgramFlashFromFile(
**ZESTET2_CARD_INFO ****CardInfo*,
char ****FileName*);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
FileName	Name of .bit file to use to configure the FPGA.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_FILE_NOT_FOUND	File not found
ZESTET2_FILE_ERROR	Error while reading file
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the requested operation
ZESTET2_ILLEGAL_FILE	File format is not recognised
ZESTET2_INVALID_PART_TYPE	Illegal FPGA part type
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2ProgramFlashFromFile can be used to program a .bit file generated by the Xilinx place and route software into the ZestET2 user FPGA boot flash. After programming, the user FPGA will configure with this .bit file on power up. **ZestET2ProgramFlashFromFile** programs the flash directly from the file on disk. Refer to **ZestET2RegisterImage** and **ZestET2ProgramFlash** for details of how to program the flash from configuration data in memory.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *CardInfo;

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Program the flash from a .bit file */
ZestET2ProgramFlashFromFile(&(CardInfo[0]), "test.bit");
```

ZestET2ProgramFlash

ZESTET2_STATUS ZestET2ProgramFlash(
ZESTET2_CARD_INFO ****CardInfo*,**
ZESTET2_IMAGE *Image***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Image	FPGA configuration image to use to configure the FPGA.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_IMAGE_HANDLE	Attempt to use illegal configuration image handle
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2ProgramFlash programs the user FPGA boot flash on a ZestET2 card from a configuration image. The configuration image can be created from a .bit file by calling **ZestET2LoadImage** or from data in memory by calling **ZestET2RegisterImage**.

Example creating the image from .bit file on disk:

```
ZESTET2_IMAGE Image;
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;

/* Load the .bit file */
ZestET2LoadImage("test.bit", &Image);

/* Other initialization operations here */

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Other execution operations here */

/* Program user flash from the image */
ZestET2ProgramFlash(&(CardInfo[0]), Image);
```

ZestET2EraseFlash

ZESTET2_STATUS ZestET2EraseFlash(ZESTET2_CARD_INFO ****CardInfo*);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2EraseFlash erases the user FPGA boot flash on a ZestET2 card. After calling this function, the user FPGA will no longer configure on power up.

Example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Erase user flash */
ZestET2EraseFlash(&(CardInfo[0]));
```

ZestET2OpenConnection

ZESTET2_STATUS ZestET2OpenConnection(**ZESTET2_CARD_INFO *** *CardInfo*,
ZESTET2_CONNECTION_TYPE *Type,*
unsigned short *Port,*
unsigned short *LocalPort,*
ZESTET2_CONNECTION * *Connection***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Type	Specifies whether connection should be UDP or TCP.
Port	Port on ZestET2 to connect to.
LocalPort	Local port to use for connection. Can be 0 for 'don't care'.
Connection	Pointer to receive handle for open connection.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_INVALID_CONNECTION_TYPE	*Type* must be **ZESTET2_TYPE_TCP** or **ZESTET2_TYPE_UDP**

Description

ZestET2OpenConnection opens a connection for communication with a ZestET2 card. It provides a simple layer on top of standard socket functions to transfer data to or from a card. For more complex data transfers, the standard socket API should be used.

ZestET2OpenConnection will attempt to connect as a client to the specified port on the ZestET2 using either TCP or UDP as specified. The user device attached to the ZestET2 should have initialised the ZestET2 as a server on the corresponding port number (i.e. the local port register on GigEx should match the *Port* parameter) for data transfers to take place.

Once data transfer is complete, the connection handle should be passed to **ZestET2CloseConnection**.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
ZESTET2_CONNECTION Connection;

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 1000);

/* Open a TCP connection to port 5002 on a card */
ZestET2OpenConnection(&(CardInfo[0]), ZESTET2_TYPE_TCP,
                      5002, 0, &Connection);
```

```
/* ... Transfer data to/from card ... */

/* Close the connection */
ZestET2CloseConnection(Connection);
```

ZestET2CloseConnection

ZESTET2_STATUS ZestET2CloseConnection(ZESTET2_CONNECTION *Connection***);**

Parameters

Connection	Handle of open connection.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	*Connection* is NULL
ZESTET2_ILLEGAL_CONNECTION	Attempt to use illegal connection handle

Description

ZestET2CloseConnection closes a connection previously opened by **ZestET2OpenConnection**. It provides a simple layer on top of standard sockets to transfer data to or from a card. For more complex data transfers, the standard socket API should be used.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
ZESTET2_CONNECTION Connection;

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Open a TCP connection to port 5002 on a card */
ZestET2OpenConnection(&(CardInfo[0]), ZESTET2_TYPE_CONNECTION,
                      5002, 0, &Connection);

/* ... Transfer data to/from card ... */

/* Close the connection */
ZestET2CloseConnection(Connection);
```

ZestET2WriteData

ZESTET2_STATUS ZestET2WriteData(ZESTET2_CONNECTION *Connection*,
**void ****Buffer,*
unsigned long *Length,*
**unsigned long ****Written,*
unsigned long *Timeout***);**

Parameters

Connection	Handle of open connection. See **ZestET2OpenConnection**.
Buffer	Pointer to data to write to ZestET2.
Length	Number of bytes to write to ZestET2.
Written	Number of bytes actually written to ZestET2.
Timeout	Length of timeout for write in milliseconds.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_CONNECTION	Attempt to use illegal connection handle
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while talking to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2WriteData writes data to a ZestET2 card. It provides a simple layer on top of standard sockets to write data to a card. For more complex data transfers, the standard socket API should be used.

ZestET2WriteData will block until data transfer is complete or until a timeout occurs.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
ZESTET2_CONNECTION Connection;
unsigned long Written;
char Buffer[65536];

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Open a TCP connection to port 5002 on a card */
ZestET2OpenConnection(&(CardInfo[0]), ZESTET2_TYPE_TCP,
                      5002, 0, &Connection);

/* Write data to a card */
ZestET2WriteData(Connection, Buffer, sizeof(Buffer), &Written, 500);
```

```
/* Close the connection */
ZestET2CloseConnection(Connection);
```

ZestET2ReadData

ZESTET2_STATUS ZestET2ReadData(ZESTET2_CONNECTION *Connection*,
**void ****Buffer,*
unsigned long *Length,*
**unsigned long ****Read,*
unsigned long *Timeout***);**

Parameters

Connection	Handle of open connection. See **ZestET2OpenConnection**.
Buffer	Pointer to buffer to receive data from ZestET2.
Length	Number of bytes to read from ZestET2.
Read	Number of bytes actually read from ZestET2.
Timeout	Length of timeout for read in milliseconds.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_ILLEGAL_CONNECTION	Attempt to use illegal connection handle
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2ReadData reads data from a ZestET2 card. It provides a simple layer on top of standard sockets to read data from a card. For more complex data transfers, the standard socket API should be used.

ZestET2ReadData will block until data transfer is complete or until a timeout occurs.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
ZESTET2_CONNECTION Connection;
unsigned long Read;
char Buffer[65536];

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Open a TCP connection to port 5002 on a card */
ZestET2OpenConnection(&(CardInfo[0]), ZESTET2_TYPE_TCP, 5002, 0,
                      &Connection);

/* Read data from a card */
ZestET2ReadData(Connection, Buffer, sizeof(Buffer), &Read, 5000);
```

```
        /* Close the connection */
        ZestET2CloseConnection(Connection);
```

ZestET2WriteRegister

ZESTET2_STATUS ZestET2WriteRegister(**ZESTET2_CARD_INFO** ***CardInfo*,**
unsigned long *Addr,*
unsigned short *Data***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Addr	Address of register to write to. Must be between 0 and 127.
Data	16 bit data to write to register.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_ILLEGAL_PARAMETER	*Addr* was out of range
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2WriteRegister writes to one of the general purpose comms registers to the external device attached to the GigEx. These registers can be read by the external user device from GigEx at address 0x300 - 0x3ff (i.e. A9 and A8 are both 1). Note that one set of registers passes values from the network to the external device and a second set passes values from the external device to the network, see Figure 22. Therefore reading from the same address using **ZestET2ReadRegister** below will not read back the value written with this function **ZestET2WriteRegister**.

ZestET2WriteRegister will block until data transfer is complete or until a timeout occurs.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *Cards;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 1000);

/* Write to general purpose register at offset 63 */
ZestET2WriteRegister(&(Cards[0]), 63, 0x1234);
```

ZestET2ReadRegister

ZESTET2_STATUS ZestET2ReadRegister(**ZESTET2_CARD_INFO** ***CardInfo,**
unsigned long *Addr,*
unsigned short ****Data*);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Addr	Address of register to write to. Must be between 0 and 127.
Data	16 bit data read from register.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_ILLEGAL_PARAMETER	*Addr* was out of range
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2ReadRegister reads the value of one of the general purpose registers from the external device attached to the GigEx. These registers can be written to GigEx by the external user device at address 0x300 - 0x3ff (i.e. A9 and A8 are both 1). Note that one set of registers passes values from the network to the external device and a second set passes values from the external device to the network, see Figure 22. Therefore writing to the address using **ZestET2WriteRegister** above will not cause the value to be read back with this function **ZestET2ReadRegister**.

ZestET2ReadRegister will block until data transfer is complete or until a timeout occurs.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *Cards;
unsigned short Val;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 2000);

/* Read from general purpose register at offset 48 */
ZestET2WriteRegister(&(Cards[0]), 48, &Val);
```

ZestET2WriteFlash

**ZESTET2_STATUS ZestET2WriteFlash(ZESTET2_CARD_INFO ****CardInfo*,
unsigned long *Address,*
**void ****Buffer,*
unsigned long *Length***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Address	Start address in flash to write to.
Buffer	Pointer to data to write to flash.
Length	Length of data to write in bytes.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2WriteFlash writes arbitrary data to the ZestET2 user flash. The base address to write to is specified by *Address*. If the user flash is also used to configure the user FPGA at power on then the FPGA configuration data will reside at address 0. The user flash is 8Mbytes in size and the FPGA configuration data starts at address 0. The remainder of the flash is free for user use. The flash can be read from the FPGA with a suitable SPI logic core and could be used for storage of non-volatile parameters or Microblaze software code.

ZestET2WriteFlash manages all aspects of writing to the flash including sector alignment, erasing and read-modify-write accesses where necessary.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
char Buffer[65536];

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Write to the flash at 1MByte offset */
ZestET2WriteFlash (&(CardInfo[0]), 0x100000,
                   Buffer, sizeof(Buffer));
```

ZestET2ReadFlash

**ZESTET2_STATUS ZestET2ReadFlash(ZESTET2_CARD_INFO ****CardInfo*,
unsigned long *Address,*
**void ****Buffer,*
unsigned long *Length***);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Address	Start address in flash to read to.
Buffer	Pointer to buffer to receive data from flash.
Length	Length of data to read in bytes.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out

Description

ZestET2ReadFlash reads data from the ZestET2 user flash. The base address to read from is specified by *Address*.

For example:

```
ZESTET2_CARD_INFO *CardInfo;
unsigned long NumCards;
char Buffer[65536];

/* Find cards on the network */
ZestET2CountCards(&NumCards, &CardInfo, 2000);

/* Read from the flash at 1MByte offset */
ZestET2ReadFlash (&(CardInfo[0]), 0x100000,
                  Buffer, sizeof(Buffer));
```

ZestET2SetInterrupt

ZESTET2_STATUS ZestET2SetInterrupt(ZESTET2_CARD_INFO ****CardInfo*);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2SetInterrupt generates an interrupt to the external device attached to GigEx. This can be used to inform the external device that some of the general comms registers have been changed and to trigger a read of these registers by the external device.

The nINT signal to the external device will be asserted if the mailbox interrupt enable bit is set in the mailbox control register (see section 5.2). The mailbox interrupt status bit will be set even if the interrupt enable bit is not set. The mailbox interrupt will be cleared when the external device reads the mailbox interrupt status register.

ZestET2SetInterrupt will block until data transfer is complete or until a timeout occurs.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *Cards;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 2000);

/* Set mailbox interrupt on ZestET2 */
ZestET2SetInterrupt(&(Cards[0]));
```

ZestET2SetGPIO

ZESTET2_STATUS ZestET2SetGPIO (**ZESTET2_CARD_INFO** **CardInfo,*
unsigned long *Select,*
unsigned long *Enable,*
unsigned long *DataOut,*
int *NonVolatile* **);**

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Select	Mask of bits to select between GPIO pins and normal function of pins
Enable	Mask of bits selecting whether a GPIO pin is an input or output
DataOut	Mask of bits to drive onto GPIO pins when selected and enabled
NonVolatile	Non-zero to write the values to the non-volatile RAM or zero for temporary settings

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_UNSUPPORTED	The GigEx device installed does not support this operation
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2SetGPIO controls the settings of GPIO pins on the GigEx device. 24 pins on GigEx can be set as GPIO inputs or outputs to directly signal the FPGA. See section 0 for details of the 24 available pins.

Each bit in *Select*, *Enable* and *DataOut* corresponds to one GPIO pin. When the *Select* bit is high, the pin will be in GPIO mode, when 0 it will be in normal mode (i.e. high speed or low speed interface functionality). When the *Enable* bit is high, the GPIO pin will be an output (when selected), when low it will be an input. When the *DataOut* bit is high, the pin will be driven high, when low it will be driven low (when selected and enabled).

Passing a non-zero value for *NonVolatile* will also copy the values to non-volatile memory so they will be preserved across a power cycle.

For example:

```
unsigned long NumCards;
ZESTET2_CARD_INFO *Cards;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 2000);
```

```
/* Set GPIO14 as an output and drive it high */
ZestET2SetGPIOPIReadWrite(&(Cards[0]), 1<<14, 1<<14, 1<<14, 0);
```

ZestET2GetGPIO

ZESTET2_STATUS ZestET2GetGPIO (**ZESTET2_CARD_INFO** ***CardInfo,***
**unsigned long ****Select,*
**unsigned long ****Enable,*
unsigned long **DataOut,*
unsigned long ****DataIn* **);

Parameters

CardInfo	Details of target ZestET2 card. See **ZestET2CountCards**.
Select	Pointer to location to receive current mask of bits to select between GPIO pins and normal function of pins (can be NULL)
Enable	Pointer to location to receive current mask of bits selecting whether a GPIO pin is an input or output (can be NULL)
DataOut	Pointer to location to receive current mask of bits to drive onto GPIO pins when selected and enabled (can be NULL)
DataIn	Pointer to location to receive current mask of values read from the GPIO pins (can be NULL)

Return Value

ZESTET2_SUCCESS	Function succeeded
ZESTET2_NULL_PARAMETER	NULL was used illegally as one of the parameter values
ZESTET2_UNSUPPORTED	The GigEx device installed does not support this operation
ZESTET2_OUT_OF_MEMORY	Not enough memory to complete the request
ZESTET2_INTERNAL_ERROR	An unspecified internal error occurred
ZESTET2_TIMEOUT	Operation timed out
ZESTET2_SOCKET_ERROR	Error while connecting to remote socket
ZESTET2_SOCKET_CLOSED	Remote device closed the socket early

Description

ZestET2GetGPIO reads the status of GPIO pins on the GigEx device. 24 pins on GigEx can be set as GPIO inputs or outputs to directly signal the FPGA. See section 0 for details of the 24 available pins.

Each bit in *Select*, *Enable* and *DataOut* corresponds to one GPIO pin. When the *Select* bit is high, the pin will be in GPIO mode, when 0 it will be in normal mode (i.e. high speed or low speed interface functionality). When the *Enable* bit is high, the GPIO pin will be an output (when selected), when low it will be an input. When the *DataOut* bit is high, the pin will be driven high, when low it will be driven low (when selected and enabled).

The *DataIn* mask contains the current state of the GPIO pins and is used to read the pin values when they are configured as inputs.

For example:

```
        unsigned long NumCards;
```

```
ZESTET2_CARD_INFO *Cards;
unsigned long DataIn;

/* Get the number of cards in the system */
ZestET2CountCards(&NumCards, &Cards, 2000);

/* Read the current state of the GPIO pins */
ZestET2GetGPIOPIReadWrite(&(Cards[0]), NULL, NULL, NULL, &DataIn);
```

10 Electrical Characteristics

10.1 DC Specifications

The tables below show the operating conditions and DC electrical characteristics of the ZestET2 module.

Parameter	Condition	Min	Typ	Max	Units
Module power supply voltage 3V3_IN		3.15	3.30	3.45	V
Module power supply current I_{CC3V3}	3V3_IN = 3.45V			0.9[1]	A
GigEx-User FPGA Interface voltage generated on board		1.71	1.80	1.89	V
Ambient Operating Temperature t_A	Convection cooling [2]	-40		60	^{O}C
	Forced air cooling 500 lfm	-40		80	^{O}C

(1) Measured transferring data over Ethernet at 100MBytes/sec in both directions and with simple loopback program in the User FPGA.

(2) Convection cooling temperature range is with module vertical, when horizontal maximum temperature is reduced by 3 ^{O}C.

10.2 AC Specifications

The following tables provide switching characteristics over the operating temperature and voltage range shown in sections 3.3.6 and 10.1.

10.2.1 User Interface Clock

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{DutyIn}	Input clock high/low duty cycle	40	60	%
$T_{DutyOut}$	Output clock high/low duty cycle	45	55	%
T_{RISE}	Input clock rise time		2	ns
T_{FALL}	Input clock fall time		2	ns

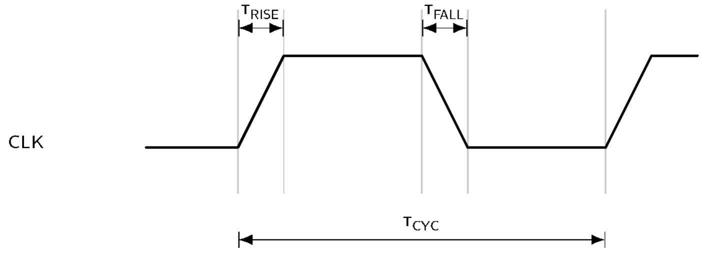


Figure 27. GigEx User Interface Clock Switching Characteristics

10.2.2 16 Bit SRAM Interface

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{CO}	Data output valid after CLK rise		4.2	ns
T_{DOH}	Data output hold after CLK rise	1.5		ns
T_{AS}	Address setup before CLK rise	0.0		ns
T_{DS}	Data setup before CLK rise	0.0		ns
T_{CES}	Chip select setup before CLK rise	0.0		ns
T_{WES}	Write enable setup before CLK rise	0.0		ns
T_{BES}	Byte enable setup before CLK rise	0.0		ns
T_{AH}	Address hold time after CLK rise	1.0		ns
T_{DH}	Data hold time after CLK rise	1.0		ns
T_{CEH}	Chip select hold after CLK rise	1.0		ns
T_{WEH}	Write enable hold after CLK rise	1.0		ns
T_{BEH}	Byte enable hold after CLK rise	1.0		ns

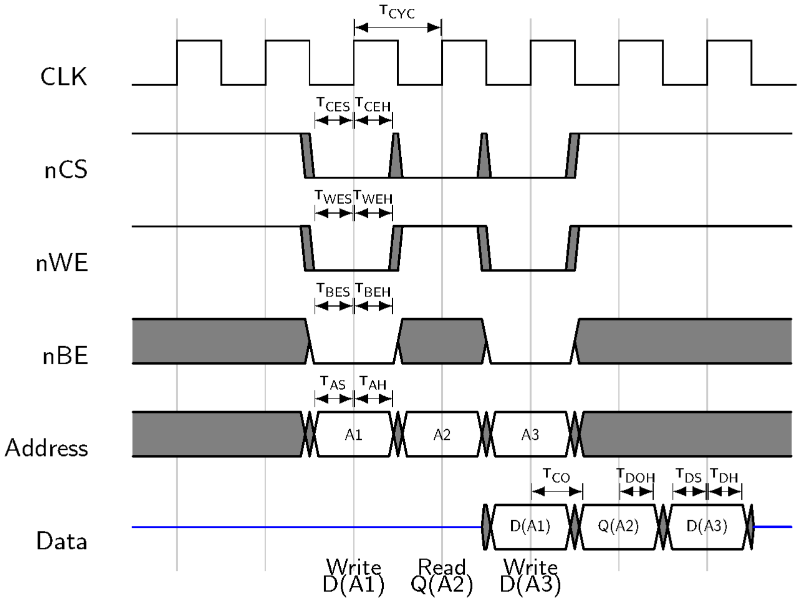


Figure 28. GigEx 16 Bit SRAM Interface Switching Characteristics

10.2.3 8 Bit SRAM Interface

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{CO}	Data output valid after CLK rise		4.2	ns
T_{DOH}	Data output hold after CLK rise	1.5		ns
T_{DAS}	Address setup before CLK rise	0.0		ns
T_{DS}	Data setup before CLK rise	0.0		ns
T_{QAS}	Address setup before CLK rise	0.0		ns
T_{WES}	Write enable setup before CLK rise	0.0		ns
T_{RES}	Read enable setup before CLK rise	0.0		ns
T_{DAH}	Address hold after CLK rise	1.0		ns
T_{DH}	Data hold time after CLK rise	1.0		ns
T_{QAH}	Address hold after CLK rise	1.0		ns
T_{WEH}	Write enable hold after CLK rise	1.0		ns
T_{REH}	Read enable hold after CLK rise	1.0		ns

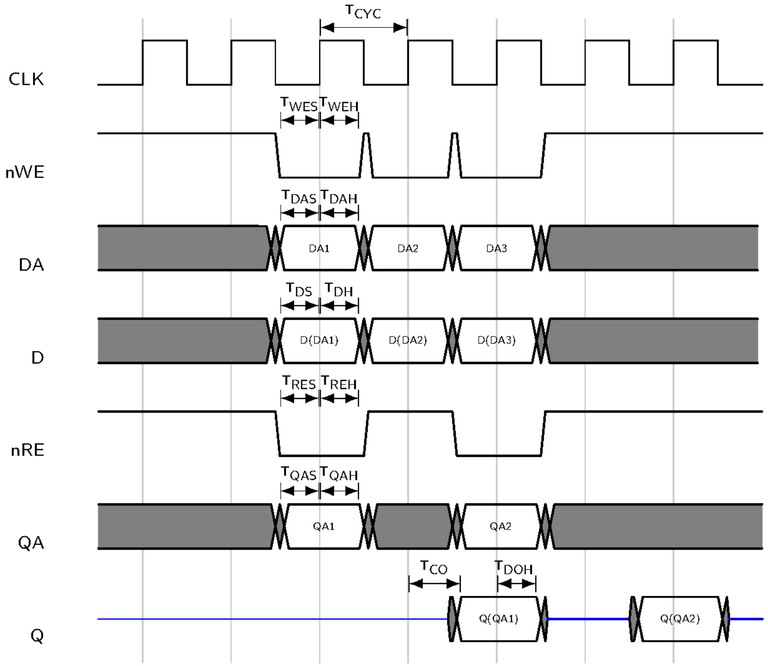


Figure 29. GigEx 8 Bit SRAM Interface Switching Characteristics

10.2.4 FIFO Interface

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{COTF}	Transmit full flag output valid after CLK rise		4.2	ns
T_{TFH}	Transmit full flag hold after CLK rise	1.5		ns
T_{TXS}	Transmit enable setup before CLK rise	0.0		ns
T_{TXH}	Transmit enable hold after CLK rise	1.0		ns
T_{TFFL}	Latency from transmit full to transmit disable		2	cycles
T_{TFRL}	Latency from transmit not full to transmit enable	0		cycles
T_{TCS}	Transmit channel setup before CLK rise	0.0		ns
T_{TCH}	Transmit channel hold after CLK rise	1.0		ns
T_{TDS}	Transmit data setup before CLK rise	0.0		ns
T_{TDH}	Transmit data hold after CLK rise	1.0		ns
T_{RFS}	Receive flag setup before CLK rise	0.0		ns
T_{RFH}	Receive flag hold after CLK rise	1.0		ns
T_{CORX}	Receive enable output valid after CLK rise		4.2	ns
T_{RXH}	Receive enable output hold after CLK rise	1.5		ns
T_{RXL}	Latency from receive full to receive disable		2	cycles
T_{CORC}	Receive channel output valid after CLK rise		4.2	ns
T_{RCH}	Receive channel output hold after CLK rise	1.5		ns
T_{COQ}	Receive data output valid after CLK rise		4.2	ns
T_{QH}	Receive data output hold after CLK rise	1.5		ns

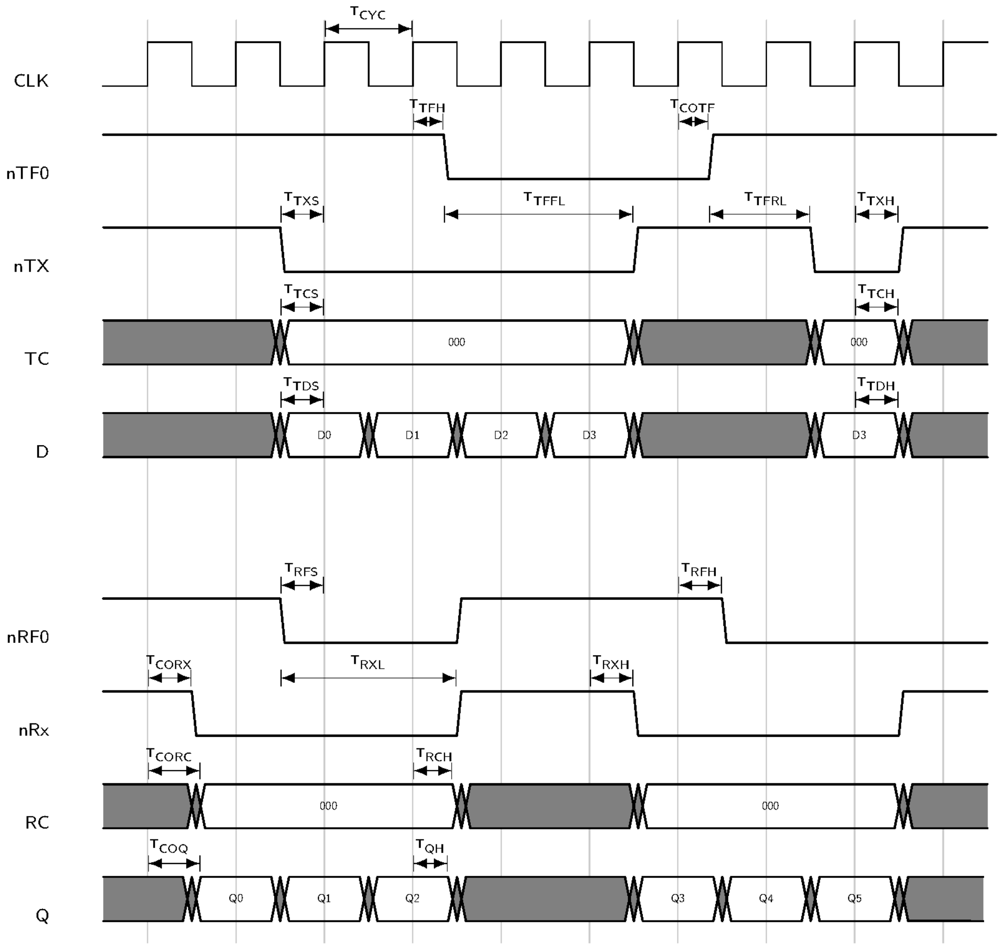


Figure 30. GigEx FIFO Interface Switching Characteristics

10.2.5 Direct Interface

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{CORF}	Receive empty output valid after CLK rise		4.2	ns
T_{RFH}	Receive empty output hold after CLK rise	1.5		ns
T_{RFS}	Receive full flag setup before CLK rise	0.0		ns
T_{RFH}	Receive full flag hold after CLK rise	1.0		ns
T_{CORX}	Receive enable output valid after CLK rise		4.2	ns
T_{RXH}	Receive enable output hold after CLK rise	1.5		ns
T_{RXLF}	Receive full to receive disable latency		2	cycles
T_{COQ}	Data output valid after CLK rise		4.2	ns
T_{QH}	Data output hold after CLK rise	1.5		ns
T_{COTF}	Transmit full flag valid after CLK rise		4.2	ns
T_{TFH}	Transmit full flag hold after CLK rise	1.5		ns
T_{TXS}	Transmit enable setup before CLK rise	0.0		ns
T_{TXLF}	Transmit full to transmit disable latency		2	cycles
T_{TXH}	Transmit enable hold after CLK rise	1.0		ns
T_{TDS}	Transmit data setup before CLK rise	0.0		ns
T_{TDH}	Transmit data hold after CLK rise	1.0		ns

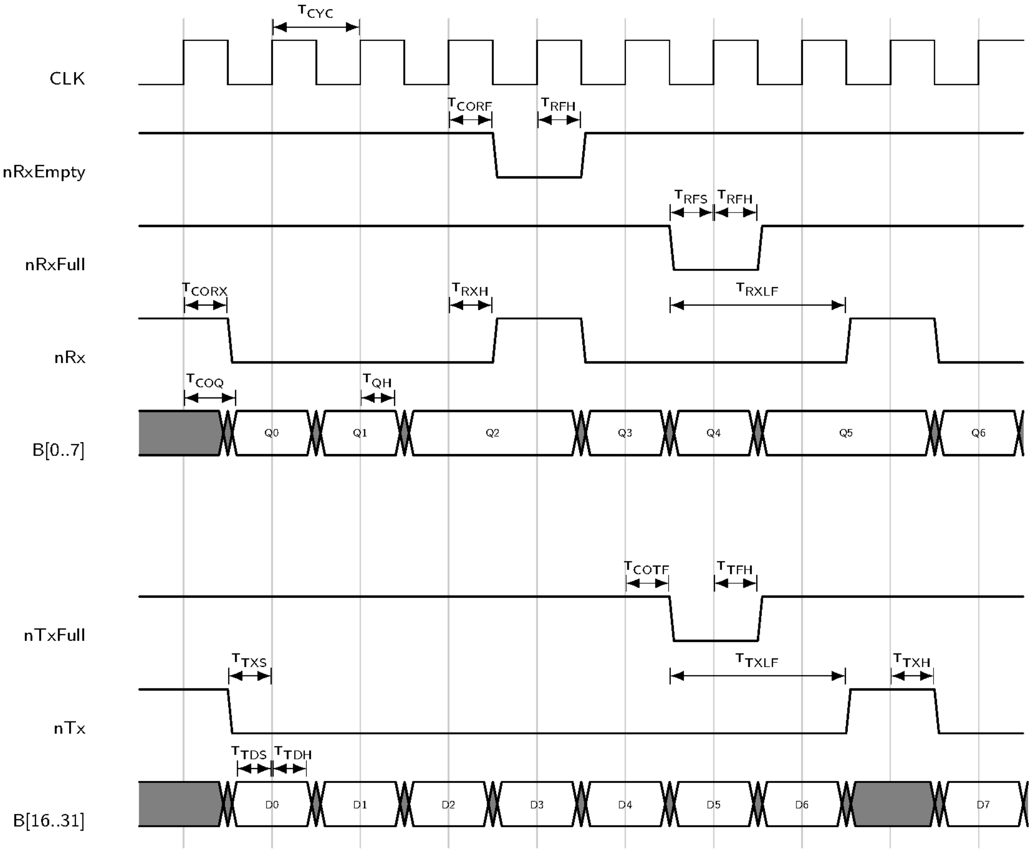


Figure 31. GigEx Direct Interface Switching Characteristics

10.2.6 SPI Slave Interface

Parameter	Description	Min	Max	Units
T_{CYC}	Interface clock cycle time	8	1000	ns
F_{MAX}	Interface clock frequency	1	125	MHz
T_{CSS}	Chip select setup before CLK rise	3.5		ns
T_{CSH}	Chip select hold after CLK rise	1.0		ns
T_{DS}	Data setup before CLK rise	0.0		ns
T_{DH}	Data hold after CLK rise	4.5		ns
T_{COQ}	Data output valid after CLK rise		7.1	ns

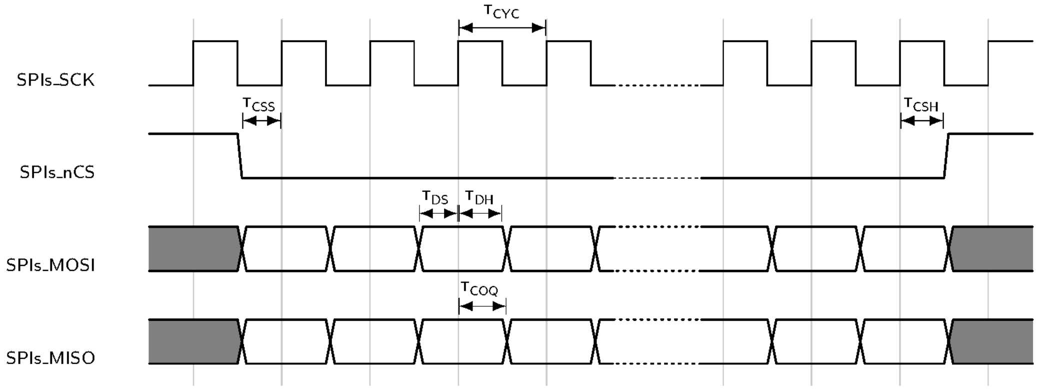


Figure 32. GigEx SPI Slave Interface Switching Characteristics

Appendix A. ZestET2-NJ Module and ZestET2-NJ Breakout Board Installation

This section describes the installation of the ZestET2-NJ module and ZestET2-NJ Breakout board. This is applicable only if the ZestET2-NJ Breakout board was ordered. Please check that the following items are in the package sent to you and contact Orange Tree Technologies if any are missing:

1. ZestET2-NJ module
2. ZestET2-NJ Breakout board
3. ZestET2 Support Software CD
4. Ethernet cable
5. Power supply cable
6. Installation Instructions printed sheet
7. Known Issues printed sheet (if there are any known issues)

A.1 System Requirements

1. Host computer or Ethernet switch or hub with a port that can run at 10Mbps or 100Mbps or 1000Mbps.

2. Host computer attached to the network running Windows XP, Windows Vista, Windows 7 or Windows 8 operating systems.

3. A 12V power supply is required, with current rating 1A to 3A depending on whether an FMC is also fitted. The supplied power supply cable has 4mm banana plugs for plugging into a lab power supply, and the other end plugs into J10. Alternatively a wall adapter can power the board via the 2.5mm power jack J11 with inner pin positive.

4. Operating ambient temperature range - see section 3.3.6.

Please read fully and then follow these installation instructions. The Reference section below explains terms used in the installation instructions and has pinouts for all connectors and jumpers.

A.2 Hardware Installation

1. **Ideally all these installation operations should be performed in an anti-static environment with an anti-static workbench and anti-static wrist-straps. Alternatively if this is not possible you should touch only the edges of the boards and earth yourself regularly during installation by touching an unpainted earthed metal surface.**

2. Place the ZestET2-NJ Breakout board on a flat surface close enough to the host PC so that the Ethernet cable reaches between them.

3. Plug ZestET2-NJ into the ZestET2-NJ Breakout board.

4. Set the jumpers J4, J5, J13 and J14 as required. The board is supplied with the default settings:
   - J4 and J5 -ZestET2-NJ User FPGA VCCO set to 3.3V

- J13 - VADJ source is VADJ internal
- J14 - VADJ Internal voltage is 3.3V

5. Connect 12V power to the board using one of the following two options:
   - J10 – 2-pin header with pin 2 positive
   - J11 - power jack with 2.5mm inner, 5.5mm outer, inner positive

! **The power pins on the two connectors above are connected together, so only either J10 or J11 connectors should be used to supply power to the board.**

6. If VADJ External is required connect this to J12.

7. Plug the Ethernet cable into ZestET2-NJ Ethernet connector J1 and the host computer or Ethernet hub. It does not matter whether the host computer is switched on or off.

8. If the host computer is switched off then switch it on now.

A.3 Software Installation

Refer to section 9 for software installation for ZestET2-NJ and running the examples. There is no further software to be installed for the ZestET2-NJ Breakout board.

Correct operation can be shown by pinging the ZestET2-NJ "ping 192.168.1.100" or browsing to the ZestET2-NJ's configuration server at 192.168.1.100 in a web browser.

The examples supplied with the Support Software package can be run from the host computer.

A.4 ZestET2-NJ Breakout Board Reference

The ZestET2-NJ Breakout Board is a carrier board for the ZestET2-NJ module. It provides the Ethernet jack and breaks out the ZestET2-NJ User FPGA IO signals to two IO connectors. One IO connector is an FMC connector (FPGA Mezzanine Card) that accepts any standard FMC module with a low pincount (LPC) connector. FMC's are available from a variety of manufacturers and offer many types of IO such as analogue, video, RF, and serial communications. The other IO connector is a standard 52-pin 0.1 inch pitch socket.

There is a JTAG chain with the User FPGA of the ZestET2-NJ module first in the chain, followed by the FMC module. The reference voltage for the signals in the JTAG connector J9 is 3.3V and this is converted on-board to 1.8V for the ZestET2-NJ User FPGA. When an FMC is not present it is taken out of the JTAG chain automatically, FMC TDI is connected to FMC TDO via a buffer controlled by the 'FMC Present' signal, which is driven low by any FMC when it is plugged in and pulled high when it is not plugged in.

Power for the board is 12V because FMC requires 12V. An on-board DC/DC converter supplies 3.3V to the ZestET2-NJ module.

The IO voltage for the FMC connector J7 is called VADJ. This can be supplied externally via J12 or generated internally by an on-board DC/DC converter whose voltage is set by J14. The IO voltage VCCO of the ZestET2-NJ User FPGA is set by J4 and J5 and when an FMC is plugged in VADJ and VCCO must be the same voltage.

The IO voltage for ZestET2-NJ User FPGA is called VCCO. There are three IO banks on the User FPGA with pins connected to the ZestET2-NJ IO connectors - banks 15, 16 and 35. There are only three Bank 16 signals and these are all connected to the IO socket J8. The Bank 15 and 35 signals are distributed between the FMC connector J7 and the IO socket J8. VCCO for Banks 16 and 35 is set by J4 and VCCO for Bank 15 is set by J5. Both J4 and J5 select between VADJ, VCCO_HDR and 3.3V. VCCO_HDR is supplied from a pin on the IO socket J8 so that a device connected to J8 can set the IO voltage of the ZestET2-NJ User FPGA signals connected to J8.

Note that Banks 15 and 35 have signals connected to both J7 and J8, so if using both J7 and J8 the IO voltages of ZestET2-NJ User FPGA (set by J4 and J5) and the devices connected to J7 (VADJ set by J13 and either J14 or the J12 supply) and J8 must be the same.

Figure 33 shows the connectors and jumpers on ZestET2-NJ Breakout Board, and descriptions follow.

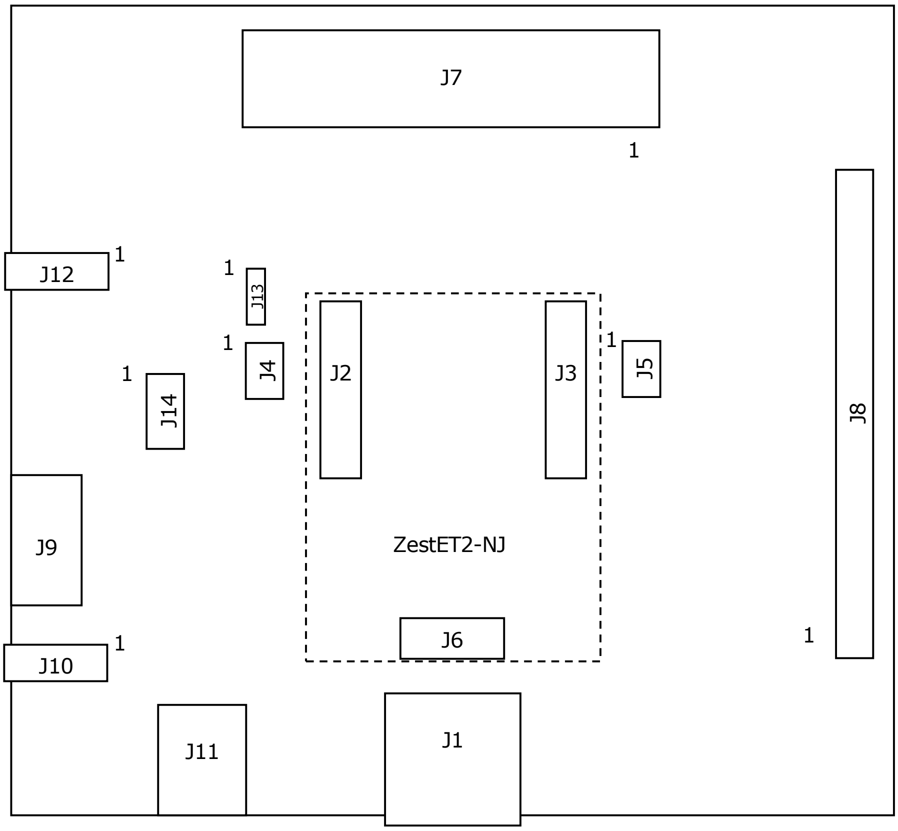


Figure 33. ZestET2-NJ Breakout Board Connectors and Jumpers

J1 Ethernet jack - Halo HFJ11-1G41E-L12RL (tab down)

J2 ZestET2-NJ connector - Hirose DF12(3.0)-80DP-0.5V

J3 ZestET2-NJ connector - Hirose DF12(3.0)-80DP-0.5V

J4 Jumper selector for ZestET2-NJ User FPGA IO Banks 16 and 35 VCCO power to ZestET2-NJ

J4 Jumper Position	Power In Source
1-2	VCC_VADJ
3-4	VCCO_HDR
5-6	3.3V

J5 Jumper selector for ZestET2-NJ User FPGA IO Bank 15 VCCO power to ZestET2-NJ

J5 Jumper Position	Power In Source
1-2	VCC_VADJ
3-4	VCCO_HDR
5-6	3.3V

J6 ZestET2-NJ connector - Hirose DF12(3.0)-40DP-0.5V

J7 FMC Low Pin Count (LPC) IO connector - Samtec ASP-134603-01

On the LPC connector, rows A, B, E, F, J and K are unpopulated. The other rows are assigned as follows.
'GND' is ground or 0V
'NC' is not connected
'LA*' are IO signals with functionality specified by the FMC. Those signals with "CC" suffix are clock compatible and are preferred signals for clock inputs to the FMC.

		J7 FMC row C	
Pin	FMC signal	ZestET2-NJ signal	Notes
C1	GND	GND	
C2	DP0_C2M_P	NC	Multi-gigabit transceiver data pair (NC)
C3	DP0_C2M_N	NC	
C4	GND	GND	
C5	GND	GND	
C6	DP0_M2C_P	NC	Multi-gigabit transceiver data pair (NC)
C7	DP0_M2C_N	NC	
C8	GND	GND	
C9	GND	GND	
C10	LA06_P	IOBank15_24	
C11	LA06_N	IOBank15_25	
C12	GND	GND	
C13	GND	GND	
C14	LA10_P	IOBank15_34	
C15	LA10_N	IOBank15_35	
C16	GND	GND	
C17	GND	GND	
C18	LA14_P	IOBank15_36	
C19	LA14_N	IOBank15_37	
C20	GND	GND	
C21	GND	GND	
C22	LA18_P_CC	IOBank35_20	
C23	LA18_N_CC	IOBank35_21	
C24	GND	GND	

C25	GND	GND	
C26	LA27_P	IOBank35_36	
C27	LA27_N	IOBank35_37	
C28	GND	GND	
C29	GND	GND	
C30	SCL	IOBank16_0	I2C clock
C31	SDA	IOBank16_4	I2C data
C32	GND	GND	
C33	GND	GND	
C34	GA0	GND	I2C geographical address of the FMC
C35	12P0V	VCC_12V0	12V power to the FMC
C36	GND	GND	
C37	12P0V	VCC_12V0	12V power to the FMC
C38	GND	GND	
C39	3P3V	VCC_3V3	3.3V power to the FMC
C40	GND	GND	

		J7 FMC row D	
Pin	**FMC signal**	**ZestET2-NJ signal**	**Notes**
D1	PG_C2M	FMC_PG_C2M	Power good asserted by breakout board when on-board VADJ and 3.3V are within tolerance
D2	GND	GND	
D3	GND	GND	
D4	GBTCLK0_M2C_P	NC	Differential pair reference clock for the multi-gigabit DP data signals (NC)
D5	GBTCLK0_M2C_N	NC	
D6	GND	GND	
D7	GND	GND	
D8	LA01_P_CC	IOBank15_28	
D9	LA01_N_CC	IOBank15_29	
D10	GND	GND	
D11	LA05_P	IOBank15_16	
D12	LA05_N	IOBank15_17	
D13	GND	GND	
D14	LA09_P	IOBank15_20	
D15	LA09_N	IOBank15_21	
D16	GND	GND	
D17	LA13_P	IOBank15_30	
D18	LA13_N	IOBank15_31	
D19	GND	GND	
D20	LA17_P_CC	IOBank35_22	
D21	LA17_N_CC	IOBank35_23	
D22	GND	GND	
D23	LA23_P	IOBank35_16	
D24	LA23_N	IOBank35_17	
D25	GND	GND	
D26	LA26_P	IOBank35_28	
D27	LA26_N	IOBank35_29	
D28	GND	GND	

D29	TCK	FMC_TCK	FMC JTAG TCK input
D30	TDI	FMC_TDI	FMC JTAG TDI input
D31	TDO	FMC_TDO	FMC JTAG TDO output
D32	3P3VAUX	VCC_3V3	3.3V auxiliary power to the FMC
D33	TMS	FMC_TMS	FMC JTAG TMS input
D34	TRST_L	NC	FMC JTAG TRST_L input
D35	GA1	GND	I2C geographical address of the FMC
D36	3P3V	VCC_3V3	3.3V power to the FMC
D37	GND	GND	
D38	3P3V	VCC_3V3	3.3V power to the FMC
D39	GND	GND	
D40	3P3V	VCC_3V3	3.3V power to the FMC

Pin	FMC signal	J7 FMC row G ZestET2-NJ signal	Notes
G1	GND	GND	
G2	CLK1_M2C_P	IOBank15_32	Differential pair reference clock from FMC
G3	CLK1_M2C_N	IOBank15_33	
G4	GND	GND	
G5	GND	GND	
G6	LA00_P_CC	IOBank15_26	
G7	LA00_N_CC	IOBank15_27	
G8	GND	GND	
G9	LA03_P	IOBank15_4	
G10	LA03_N	IOBank15_5	
G11	GND	GND	
G12	LA08_P	IOBank15_12	
G13	LA08_N	IOBank15_13	
G14	GND	GND	
G15	LA12_P	IOBank15_18	
G16	LA12_N	IOBank15_19	
G17	GND	GND	
G18	LA16_P	IOBank15_6	
G19	LA16_N	IOBank15_7	
G20	GND	GND	
G21	LA20_P	IOBank35_4	
G22	LA20_N	IOBank35_5	
G23	GND	GND	
G24	LA22_P	IOBank35_12	
G25	LA22_N	IOBank35_13	
G26	GND	GND	
G27	LA25_P	IOBank35_34	
G28	LA25_N	IOBank35_35	
G29	GND	GND	
G30	LA29_P	IOBank35_26	
G31	LA29_N	IOBank35_27	
G32	GND	GND	
G33	LA31_P	IOBank35_14	
G34	LA31_N	IOBank35_15	

G35	GND	GND	
G36	LA33_P	IOBank35_6	
G37	LA33_N	IOBank35_7	
G38	GND	GND	
G39	VADJ	VCC_VADJ	Adjustable voltage level power to FMC
G40	GND	GND	

J7 FMC row H			
Pin	**FMC signal**	**ZestET2-NJ signal**	**Notes**
H1	VREF_A_M2C	NC	IO reference voltage from FMC (NC)
H2	PRSNT_M2C_L	FMC_PRSNT_M2C_L	FMC Present
H3	GND	GND	
H4	CLK0_M2C_P	IOBank15_14	Differential pair reference clock from FMC
H5	CLK0_M2C_N	IOBank15_15	
H6	GND	GND	
H7	LA02_P	IOBank15_0	
H8	LA02_N	IOBank15_1	
H9	GND	GND	
H10	LA04_P	IOBank15_8	
H11	LA04_N	IOBank15_9	
H12	GND	GND	
H13	LA07_P	IOBank15_22	
H14	LA07_N	IOBank15_23	
H15	GND	GND	
H16	LA11_P	IOBank15_10	
H17	LA11_N	IOBank15_11	
H18	GND	GND	
H19	LA15_P	IOBank15_2	
H20	LA15_N	IOBank15_3	
H21	GND	GND	
H22	LA19_P	IOBank35_0	
H23	LA19_N	IOBank35_1	
H24	GND	GND	
H25	LA21_P	IOBank35_8	
H26	LA21_N	IOBank35_9	
H27	GND	GND	
H28	LA24_P	IOBank35_30	
H29	LA24_N	IOBank35_31	
H30	GND	GND	
H31	LA28_P	IOBank35_18	
H32	LA28_N	IOBank35_19	
H33	GND	GND	
H34	LA30_P	IOBank35_10	
H35	LA30_N	IOBank35_11	
H36	GND	GND	
H37	LA32_P	IOBank35_2	
H38	LA32_N	IOBank35_3	
H39	GND	GND	
H40	VADJ	VCC_VADJ	Adjustable voltage level power to FMC

J8 IO Header - 0.1" pitch 26x2-pin socket Samtec SSW-126-01-G-D
'GND' is ground or 0V
'NC' is no connection and should not be connected to any signal

J8			
3.3V output	**51**	**52**	VADJ output
IOBank16_1	**49**	**50**	IOBank16_2
GND	**47**	**48**	GND
IOBank16_3	**45**	**46**	NC
GND	**43**	**44**	GND
IOBank35_24	**41**	**42**	IOBank35_25
IOBank35_32	**39**	**40**	IOBank35_33
GND	**37**	**38**	GND
IOBank35_38	**35**	**36**	IOBank35_39
IOBank35_40	**33**	**34**	IOBank35_41
IOBank35_42	**31**	**32**	IOBank35_43
GND	**29**	**30**	GND
IOBank35_44	**27**	**28**	IOBank35_45
IOBank35_46	**25**	**26**	IOBank35_47
IOBank35_48	**23**	**24**	IOBank35_49
IOBank15_38	**21**	**22**	IOBank15_39
IOBank15_40	**19**	**20**	IOBank15_41
IOBank15_42	**17**	**18**	IOBank15_43
GND	**15**	**16**	GND
IOBank15_44	**13**	**14**	IOBank15_45
IOBank15_46	**11**	**12**	IOBank15_47
IOBank15_48	**9**	**10**	IOBank15_49
NC	**7**	**8**	NC
NC	**5**	**6**	NC
NC	**3**	**4**	GND
VCCO_HDR input	**1**	**2**	NC

J9 ZestET2-NJ Breakout Board JTAG connector - 2mm pitch, 7x2-pin, right angle shrouded header, Molex 87833-1420

J9 Pin	ZestET2-NJ Breakout Board JTAG signal
1	GND
2	JTAG VREF (3.3V)
3	GND
4	JTAG TMS
5	GND
6	JTAG TCK
7	GND
8	JTAG TDO output
9	GND
10	JTAG TDI input
11	GND
12	NC
13	GND
14	NC

J10 Board 12V In - 2-way right angle Tyco 640457-2

J10 Pin	Signal
1	GND
2	12V

J11 Board 12V In - Right angle DC power jack 2.5mm inner, 5.5mm outer, Switchcraft RAPC712X

J11 Pin	Signal
Outer	GND
Inner	12V

J12 VADJ External input - 2-way right angle Tyco 640457-2

VADJ supplied by an external power supply.

J12 Pin	Signal
1	GND
2	VADJ External

J13 Jumper selector for VADJ

J13 Jumper Position	VADJ Source
1-2	VADJ Internal from on-board power supply U7 with voltage selected by J14
2-3	VADJ External from J12

J14 Jumper selector for VADJ Internal from on-board power supply U7

J14 Jumper Position	VADJ Internal
1-2	3.3V
3-4	2.5V
5-6	1.8V
7-8	1.5V

Appendix B. Glossary

ARP Address Resolution Protocol - used to determine physical addresses of devices on the network.

AutoIP Allows devices to choose their own address on a network without a DHCP server.

Client Network device which is capable of initiating connections to a server device.

DHCP Dynamic Host Configuration Protocol - allows devices to configure their own addresses on a network with a suitable DHCP server.

FMC FPGA Mezzanine Card. A standard format of IO cards for FPGA boards.

GigEx Orange Tree Technologies' proprietary TCP/IP Offload Engine (TOE) chip, full name GigExpedite.

HTTP Hypertext Transfer Protocol - protocol sitting above TCP used to transfer web pages.

ICMP Internet Control Message Protocol - used by devices on the network to communicate control and status data including error information.

IGMP Internet Group Message Protocol - used by devices on the network to join and leave multicast groups.

IPv4 Internet Protocol - defines frame format and checksum to ensure correct delivery of single packets of data. However, due to variable routing paths, electrical interference and the lossy nature of networks it does not guarantee delivery of data or the order or delivery of data.

MAC Media Access Controller - component that transmits and receives packets on a network.

Multicast A network mechanism for sending data packets from one device to multiple recipients.

Phy Electrical interface to network cable.

Port TCP and UDP use the concept of ports to multiplex multiple data streams across the same network. Data is transferred between a port on one device and a port on a second device. Data transfer sessions between port pairs are kept separate.

PTP Precision Time Protocol. Allows devices on a network to synchronise their clocks to a central master time generator.

Server Network device which listens for incoming connections from one or more clients.

TCP Transmission Control Protocol - heavier user level protocol for transferring data between 'ports' on devices. Allows multiple streams of data to run over a single network between two devices. Guarantees data reception and order of reception.

TOE TCP/IP Offload Engine. Ethernet communications protocols are offloaded from the FPGA and are run instead in the TOE chip.

UDP User Datagram Protocol – lightweight user level protocol for transferring data between 'ports' on devices. Allows multiple streams of data to run over a single network between two devices. UDP is lightweight and simple but still unreliable and does not guarantee data reception or order of reception.

UPnP	Universal Plug-and-Play. Allows devices to search for and query the capabilities of other devices on a network.

Appendix C. Licensing

The GigEx firmware contains binary elements of the lwIP TCP/IP stack. The license agreement below applies to those elements.

The GigEx firmware contains binary elements of the PTPd Precision Time Protocol daemon. The license agreement below applies to those elements.