peralex
the sense of signals

Firmware User Manual

SKARAB MOTHERBOARD

ITEM NUMBER:	PX-123702
ITEM NAME:	SKARAB MOTHERBOARD
DOCUMENT NUMBER:	FUM-123702
ISSUE:	01
PREPARED BY:	Gavin Teague

ISSUE HISTORY

Description of Change	Issue No.	Edited by	Date
Initial Release	1	Gavin Teague	14-October-2015
Updated 40GbE PHY section	2	Matthew Bridges	15-October-2015

Approvals

[illegible]
....
Gavin Teague
Designer

[illegible]
....
Clifford van Dyk
Hardware Design Manager

[illegible]
....
Werner Lourens
QA Representative

[illegible]
....
Name: JASON MANLEY
Client Representative

Contents

1. Introduction

1.1. Scope and Limitations

This document details how to use the SKARAB motherboard firmware and embedded software. It provides details on how to communicate with the Microblaze microcontroller in the SKARAB LRU.

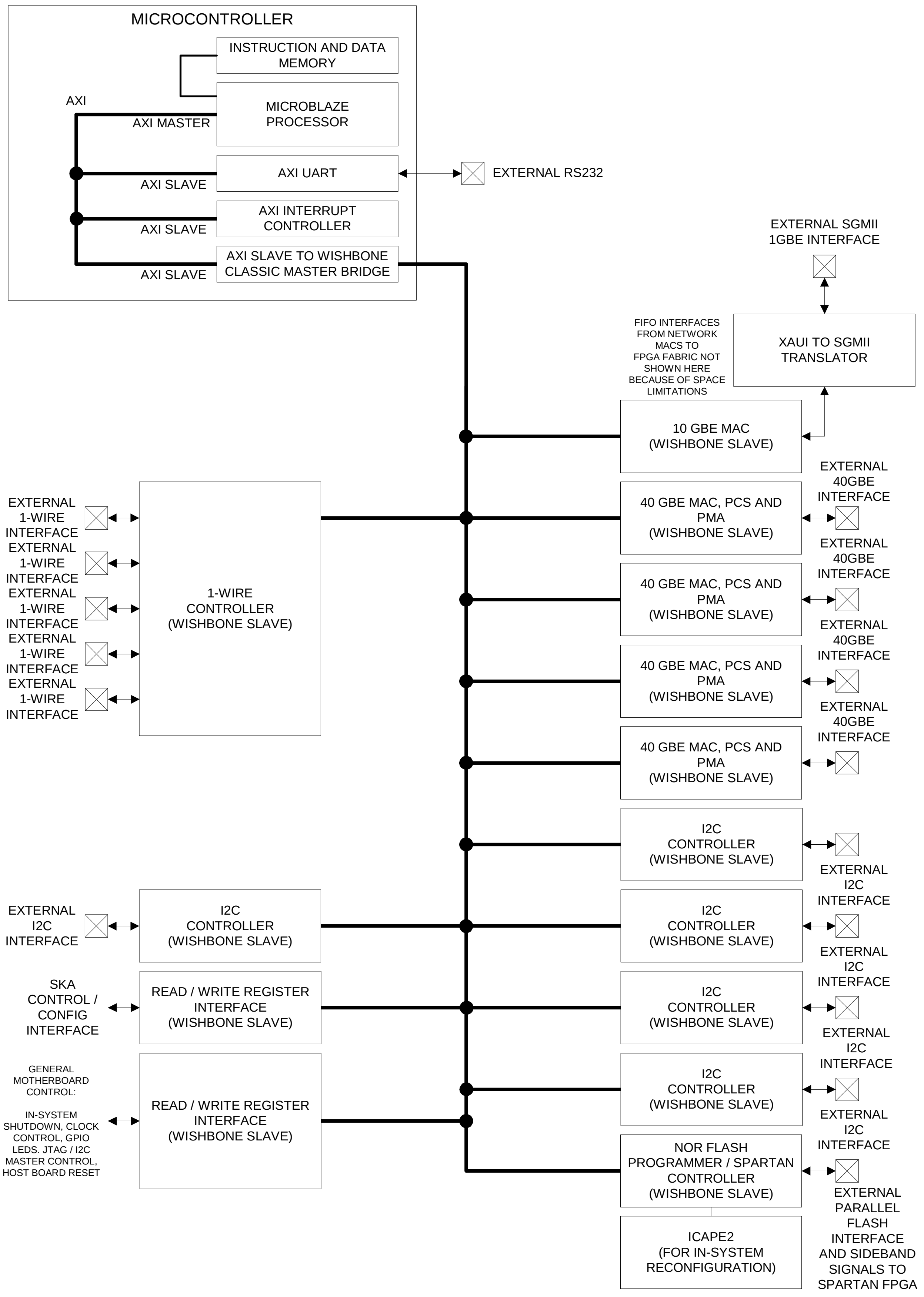
1.2. Applicable Documents

1.3. Document Layout

Section 1 reflects the content of the document, and provides the reader with associated reference material.
Section 2 details the firmware description.
Section 3 details the host API to communicate with the Microblaze microcontroller in the SKARAB LRU.
Section 4 details the embedded software of the Microblaze microcontroller.
Section 5 details the host library code and test software of the SKARAB LRU.

2. Firmware Description

The following is a block diagram of the SKARAB Motherboard firmware:


MICROCONTROLLER
INSTRUCTION AND DATA MEMORY
AXI
AXI MASTER
MICROBLAZE PROCESSOR
AXI SLAVE
AXI UART
EXTERNAL RS232
AXI SLAVE
AXI INTERRUPT CONTROLLER
AXI SLAVE
AXI SLAVE TO WISHBONE CLASSIC MASTER BRIDGE
EXTERNAL SGMII 1GBE INTERFACE
XAUI TO SGMII TRANSLATOR
FIFO INTERFACES FROM NETWORK MACS TO FPGA FABRIC NOT SHOWN HERE BECAUSE OF SPACE LIMITATIONS
10 GBE MAC (WISHBONE SLAVE)
40 GBE MAC, PCS AND PMA (WISHBONE SLAVE)
EXTERNAL 40GBE INTERFACE
EXTERNAL 1-WIRE INTERFACE
1-WIRE CONTROLLER (WISHBONE SLAVE)
I2C CONTROLLER (WISHBONE SLAVE)
EXTERNAL I2C INTERFACE
SKA CONTROL / CONFIG INTERFACE
READ / WRITE REGISTER INTERFACE (WISHBONE SLAVE)
GENERAL MOTHERBOARD CONTROL:
IN-SYSTEM SHUTDOWN, CLOCK CONTROL, GPIO LEDS. JTAG / I2C MASTER CONTROL, HOST BOARD RESET
NOR FLASH PROGRAMMER / SPARTAN CONTROLLER (WISHBONE SLAVE)
EXTERNAL PARALLEL FLASH INTERFACE AND SIDEBAND SIGNALS TO SPARTAN FPGA
ICAPE2 (FOR IN-SYSTEM RECONFIGURATION)

2.1. Project Details

The project was generated in Vivado 2014.3.1. The Vivado project file is FRM123701U1R1.xpr.

The project is located in FRM123701U1R1. There is currently only one revision of firmware for the SKARAB motherboard in source control (R1). This firmware is for the Virtex 7 FPGA on the SKARAB motherboard with the reference designator U1. A separate ISE-based firmware exists for the Spartan 3AN FPGA and this is located in FRM123701U2R1.

The directory structure of the project is as follows:

- Ip_repo: An HDL component was created ('axi_slave_wishbone_classic_master') to bridge between an AXI4 interface and a Wishbone interface. This is used by the microcontroller to provide an external Wishbone interface to all of the Wishbone peripherals in the firmware. This core was loaded into Vivado's Block Design library and instantiated as a block within the microcontroller.
- SDK: This directory contains all of the embedded software files for the microcontroller.
  - Elf: The generated ELF file of the embedded software is automatically copied here. When the firmware is built, the ELF file is accessed at this location and used for the microcontroller.
  - EMB123701U1R1: This directory contains all of the C source files for the microcontroller.
  - EMB123701U1R1_bsp: This directory contains all of the SDK generated board support package files for the microcontroller.
  - FRM123701U1R1: This directory contains the hardware description of the various block components within the microcontroller. This is generated by Vivado and imported by SDK.
  - The other directories in the SDK folder are automatically generated.
- Vivado: This directory contains the Vivado project.
  - BlockDesign: This directory contains all of the Vivado generated files for the microcontroller. The microcontroller was assembled using the Vivado Block Design GUI.
  - Constraints: This directory contains the top level constraints files. There are four different constraint files depending on where the QSFP+ Mezzanine is located ('frm123701u1r1_Mez0.xdc', 'frm123701u1r1_Mez1.xdc', 'frm123701u1r1_Mez2.xdc', 'frm123701u1r1_Mez3.xdc'). These constraints files are with all four 40GBE interfaces on the QSFP+ Mezzanine instantiated. Portions can be commented out depending on the number and location of the 40GBE interfaces used. Note that the 40GBE PHY has its own constraints files that are associated with the 40GBE PHY source files. There is also a constraints file for generating the golden boot image. The golden boot image is what is programmed into address 0x0 of the parallel NOR flash. If there is a problem during the in-system upgrade of the boot flash, then it is the golden image which is loaded. The golden boot image requires different constraints to the normal boot image and hence is a separate file.
  - IP: This directory contains all of the Vivado generated files for the basic Xilinx IP that exists in the design. This is typically for FIFOs or block memories.
  - Source: This directory contains all of the HDL source for the design.
    - SKA_10GBE_MAC: This directory contains the files for the 10GBE MAC. The existing 10GBE MAC was used for the 1GBE interface to maximize code re-use. The CPU interface was modified to be a Wishbone slave.
    - SKA_40GBE_MAC: This directory contains the files for the 40GBE MAC.
    - SKA_40GbE_PHY: This directory contains the files for the 40GBE PCS and 40GBE PMA.
    - TESTBENCH: This directory contains test bench components specifically for use during qualification and board level testing.
    - WISHBONE: This directory contains all of the Wishbone slave components.
    - Other components that don't neatly fit into one of the above directories are located in Source.
  - The other directories in the Vivado folder are automatically generated.

The following files deserve special mention:

- promgen.tcl: This TCL script is used to generate the MCS and HEX files for programming the parallel NOR flash that boots the Virtex 7 FPGA. It generates the MCS and HEX files for all four mezzanine sites.
- soc_version.xdc: In other Peralex products, the concept of a System On Chip version exists. This is a version which combines the HDL firmware version and the embedded software version. When either the HDL firmware version or embedded software version increments, the SOC version increments. The host (via the microcontroller) can read the SOC version from a register.

2.2. Top Level File ‘frm123701u1r1.vhd’

The top level file contains all of the component instantiations. Connections to the FPGA pins are handled by the top level file.

The main clock signal in the firmware is ‘sys_clk’ that runs at 156.25MHz. Almost all components operate off ‘sys_clk’ and are reset by ‘sys_rst’. ‘sys_rst’ is generated using the board level reset. It is also possible to assert ‘sys_rst’ from the microcontroller.

‘mezzanine_enable_delay.vhd’ is used to provide a one second delay between when a mezzanine is enabled and when mezzanine fault checking begins. This is to allow for the delays through the voltage level translators as well as any delays on the mezzanine itself.

‘USR_ACCESSE2’ is used to access the SOC version. The SOC version is read from the constraints file by the Vivado tools when an FPGA image is built. This components allows the host (via the microcontroller) to read the SOC version of an FPGA image.

Minimal AUX circuitry is included in the top level file. A component ‘clock_frequency_measure.vhd’ uses the 156.25MHz system clock to create a one second interval. The frequency of the AUX CLK IN is measured using this component. ‘aux_sync_gen.vhd’ generates a 40 millisecond square wave to drive AUX SYNC OUT. AUX SYNC IN is simply connected to one of the FPGA GPIO pins for probing with an oscilloscope.

All of the Wishbone slaves are instantiated in the top level file. ‘wishbone_interconnect.vhd’ scales depending on the number of Wishbone slaves. It provides the connection between the Wishbone master (microcontroller) and all of the Wishbone slaves. The addressing / indexing of the Wishbone slaves is as follows:

- 0: Board level read/write register interface ‘wishbone_register.vhd’
- 1: DSP read/write register interface ‘wishbone_register.vhd’
- 2: Flash and SDRAM interface controller ‘wishbone_flash_sdram_interface.vhd’
- 3: 1-wire controller ‘wishbone_one_wire.vhd’
- 4: I2C controller for the SKARAB motherboard ‘wishbone_i2c.vhd’
- 5: I2C controller for mezzanine site 0 ‘wishbone_i2c.vhd’
- 6: I2C controller for mezzanine site 1 ‘wishbone_i2c.vhd’
- 7: I2C controller for mezzanine site 2 ‘wishbone_i2c.vhd’
- 8: I2C controller for mezzanine site 3 ‘wishbone_i2c.vhd’
- 9: 1GBE CPU interface ‘kat_ten_gb_eth.vhd’
- 10: 40GBE 0 CPU interface ‘ska_forty_gb_eth.vhd’
- 11: 40GBE 1 CPU interface ‘ska_forty_gb_eth.vhd’
- 12: 40GBE 2 CPU interface ‘ska_forty_gb_eth.vhd’
- 13: 40GBE 3 CPU interface ‘ska_forty_gb_eth.vhd’

Xilinx provide a SGMII core ‘gmii_to_sgmii.vhd’. This SGMII interface is to the 1GBE PHY. The SGMII core comes with a GMII interface. The clock of the GMII interface is ‘gmii_clk’ that runs at 125MHz. The reset for the GMII interface is ‘gmii_rst’.

The 10GBE MAC has a XAUI interface. ‘xaui_to_gmii_translator.vhd’ translates the transmit data from the 10GBE MAC XAUI transmit interface to the GMII transmit interface. Idle words are dropped and feedback throttling is used to prevent the 10GBE MAC from exceeding the maximum available throughput of the SGMII 1GBE interface. ‘gmii_to_xaui_translator.vhd’ translates the received data from the GMII receive interface to the 10GBE MAC XAUI receive interface. The XAUI framing structure is recreated and idle words are inserted between packets.

The 40GBE PHY (PMA and PCS) components are detailed in Section 2.12 40GbE P.

To test the fabric interface of the 1GBE, a packet generator component ‘ska_ten_gb_eth_ramp_source.vhd’ is used. This component interfaces to the fabric interface of the 10GBE MAC (used for the 1GBE) and creates packets with ramp payload data. The number of packets, idle period between packets and the destination IP address of the packets are set from the host (via the microcontroller board register interface). The packet generator increments the size of the packets from smallest to jumbo packets.

A packet checker component ‘ska_ten_gb_eth_ramp_checker.vhd’ checks the packets received on the 1GBE fabric interface. It checks and latches CRC errors. It also checks and latches ramp discontinuities in the packet payload.

Lastly it records the number of packets received so that the host can check (via the microcontroller board register interface) that all packets have been successfully received.

To test the fabric interface of the 40GBE, a packet generator component 'ska_forty_gb_eth_ramp_source.vhd' is used. This component behaves in the same way as the 1GBE equivalent described above. A packet checker component 'ska_forty_gb_eth_ramp_checker.vhd' checks the packets received on the 40GBE fabric interface. This component behaves in the same way as the 1GBE equivalent described above.

'second_gen.vhd' creates a signal that toggles once a second. It uses the 156.25MHz system clock as the reference. This component is used as a timing reference for other components. For example, 'clock_frequency_measure.vhd' uses the timing reference to measure the 40GBE reference clock frequency received from the GTH. It is also used by another copy of 'clock_frequency_measure.vhd' to measure the configuration clock frequency.

Firmware counts the number of system clock cycles it takes for a set number of packets to be transmitted and received. This is used to measure the time it takes for a set number of packets to be transmitted and received. Because the contents and size of the packets are known, this duration is used to measure the throughput of the 1GBE and 40GBE interfaces.

'FPGA_DNA_CHECKER' is used to access the unique FPGA DNA code of each Virtex7 populated on a SKARAB Motherboard.

2.3. Constant and Type Defines 'parameter.vhd'

All constant and type defines are located within 'parameter.vhd'. The only constant and type defines not included here are for the 40GBE PHY. The constants defined here include, among others:

- Firmware version
- Size of register address space
- Locations of different board level read and write registers
- Number of Wishbone slaves
- Number of 40GBE interfaces

2.3.1. Board level read register map

ADDRESS	NAME
0	C_RD_VERSION_ADDR
1	C_RD_BRD_CTL_STAT_0_ADDR
2	C_RD_LOOPBACK_ADDR
3	C_RD_ETH_IF_LINK_UP_ADDR
4	C_RD_MEZZANINE_STAT_ADDR
5	C_RD_USB_STAT_ADDR
6	C_RD_SOC_VERSION_ADDR
7	C_RD_FPGA_DNA_LOW_ADDR
8	C_RD_FPGA_DNA_HIGH_ADDR
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	

22	C_RD_THROUGHPUT_COUNTER_ADDR
23	C_RD_NUM_PACKETS_CHECKED_0_ADDR
24	C_RD_NUM_PACKETS_CHECKED_1_ADDR
25	C_RD_NUM_PACKETS_CHECKED_2_ADDR
26	C_RD_NUM_PACKETS_CHECKED_3_ADDR
27	C_RD_NUM_PACKETS_CHECKED_4_ADDR
28	
29	C_RD_MEZZANINE_CLK_FREQ_ADDR
30	C_RD_CONFIG_CLK_FREQ_ADDR
31	C_RD_AUX_CLK_FREQ_ADDR

2.3.1.1. C_RD_VERSION_ADDR

BITS	DESCRIPTION
31..16	Major version
15..0	Minor version

2.3.1.2. C_RD_BRD_CTL_STAT_0_ADDR

BITS	DESCRIPTION
0	'1' = GMII reset complete
1	'1' = Voltage and current monitor alert asserted
2	'1' = Fan controller alert asserted
3	'1' = Fan controller fault asserted
4	'1' = 1GBE link up
5	'1' = 1GBE interrupt asserted
6	'1' = 1GBE packet checker – checked packets
7	'1' = 1GBE packet checker – ramp fault
8	'1' = 1GBE packet checker – IP address fault
9	'1' = 40GBE packet checker 0 – checked packets
10	'1' = 40GBE packet checker 0 – ramp fault
11	'1' = 40GBE packet checker 0 – IP address fault
12	'1' = 40GBE packet checker 1 – checked packets
13	'1' = 40GBE packet checker 1 – ramp fault
14	'1' = 40GBE packet checker 1 – IP address fault
15	'1' = 40GBE packet checker 2 – checked packets
16	'1' = 40GBE packet checker 2 – ramp fault
17	'1' = 40GBE packet checker 2 – IP address fault
18	'1' = 40GBE packet checker 3 – checked packets
19	'1' = 40GBE packet checker 3 – ramp fault
20	'1' = 40GBE packet checker 3 – IP address fault

2.3.1.3. C_RD_LOOPBACK_ADDR

BITS	DESCRIPTION
31..0	Connect directly to C_WR_LOOPBACK_ADDR for register loopback testing

2.3.1.4. C_RD_ETH_IF_LINK_UP_ADDR

BITS	DESCRIPTION
0	'1' = 1GBE link up
1	'1' = 40GBE 0 link up
2	'1' = 40GBE 1 link up
3	'1' = 40GBE 2 link up
4	'1' = 40GBE 3 link up
16	'1' = 40GBE 0 transmitter enabled
17	'1' = 40GBE 0 transmitter activity
18	'1' = 40GBE 0 receiver link up

19	‘1’ = 40GBE 0 receiver link activity
20	‘1’ = 40GBE 1 transmitter enabled
21	‘1’ = 40GBE 1 transmitter activity
22	‘1’ = 40GBE 1 receiver link up
23	‘1’ = 40GBE 1 receiver link activity
24	‘1’ = 40GBE 2 transmitter enabled
25	‘1’ = 40GBE 2 transmitter activity
26	‘1’ = 40GBE 2 receiver link up
27	‘1’ = 40GBE 2 receiver link activity
28	‘1’ = 40GBE 3 transmitter enabled
29	‘1’ = 40GBE 3 transmitter activity
30	‘1’ = 40GBE 3 receiver link up
31	‘1’ = 40GBE 3 receiver link activity

2.3.1.5. C_RD_MEZZANINE_STAT_ADDR

BITS	DESCRIPTION
0	‘1’ = Mezzanine 0 present
1	‘1’ = Mezzanine 1 present
2	‘1’ = Mezzanine 2 present
3	‘1’ = Mezzanine 3 present
8	‘1’ = Mezzanine 0 fault asserted
9	‘1’ = Mezzanine 1 fault asserted
10	‘1’ = Mezzanine 2 fault asserted
11	‘1’ = Mezzanine 3 fault asserted
16	‘1’ = Mezzanine 0 interrupt asserted
17	‘1’ = Mezzanine 1 interrupt asserted
18	‘1’ = Mezzanine 2 interrupt asserted
19	‘1’ = Mezzanine 3 interrupt asserted

2.3.1.6. C_RD_USB_STAT_ADDR

BITS	DESCRIPTION
3..0	USB PHY to FPGA nibble
8	‘1’ = USB PHY has control of I2C bus

2.3.1.7. C_RD_SOC_VERSION_ADDR

BITS	DESCRIPTION
31..16	SOC Major version
15..0	SOC Minor version

2.3.1.8. C_RD_FPGA_DNA_LOW_ADDR

BITS	DESCRIPTION
31..0	Lower 32 bits of FPGA DNA

2.3.1.9. C_RD_FPGA_DNA_HIGH_ADDR

BITS	DESCRIPTION
31..0	Upper 32 bits of FPGA DNA

2.3.1.10. C_RD_THROUGHPUT_COUNTER_ADDR

BITS	DESCRIPTION
31..0	Number of clocks at 156.25MHz to send and receive all packets

2.3.1.11. C_RD_NUM_PACKETS_CHECKED_0_ADDR

BITS	**DESCRIPTION**
23..0	Number of packets received by 1GBE ramp packet checker

2.3.1.12. C_RD_NUM_PACKETS_CHECKED_1_ADDR

BITS	**DESCRIPTION**
23..0	Number of packet received by 40GBE 0 ramp packet checker

2.3.1.13. C_RD_NUM_PACKETS_CHECKED_2_ADDR

BITS	**DESCRIPTION**
23..0	Number of packet received by 40GBE 1 ramp packet checker

2.3.1.14. C_RD_NUM_PACKETS_CHECKED_3_ADDR

BITS	**DESCRIPTION**
23..0	Number of packet received by 40GBE 2 ramp packet checker

2.3.1.15. C_RD_NUM_PACKETS_CHECKED_4_ADDR

BITS	**DESCRIPTION**
23..0	Number of packet received by 40GBE 3 ramp packet checker

2.3.1.16. C_RD_MEZZANINE_CLK_FREQ_ADDR

BITS	**DESCRIPTION**
31..0	Frequency of GTH reference clock received from the mezzanine

2.3.1.17. C_RD_CONFIG_CLK_FREQ_ADDR

BITS	**DESCRIPTION**
31..0	Frequency of configuration clock

2.3.1.18. C_RD_AUX_CLK_FREQ_ADDR

BITS	**DESCRIPTION**
31..0	Frequency of AUX CLK IN clock

2.3.2. Board level write register map

ADDRESS	**NAME**
0	
1	C_WR_BRD_CTL_STAT_0_ADDR
2	C_WR_LOOPBACK_ADDR
3	C_WR_ETH_IF_CTL_ADDR
4	C_WR_MEZZANINE_CTL_ADDR
5	C_WR_FRONT_PANEL_STAT_LED_ADDR
6	C_WR_BRD_CTL_STAT_1_ADDR
7	
8	
9	

10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	C_WR_RAMP_SOURCE_DESTINATION_IP_3_ADDR
23	C_WR_RAMP_CHECKER_SOURCE_IP_3_ADDR
24	C_WR_RAMP_SOURCE_DESTINATION_IP_2_ADDR
25	C_WR_RAMP_CHECKER_SOURCE_IP_2_ADDR
26	C_WR_RAMP_SOURCE_DESTINATION_IP_1_ADDR
27	C_WR_RAMP_CHECKER_SOURCE_IP_1_ADDR
28	C_WR_RAMP_SOURCE_PAYLOAD_WORDS_ADDR
29	C_WR_RAMP_SOURCE_DESTINATION_IP_0_ADDR
30	C_WR_RAMP_CHECKER_SOURCE_IP_0_ADDR
31	C_WR_NUM_PACKETS_TO_GENERATE_ADDR

2.3.2.1. C_WR_BRD_CTL_STAT_0_ADDR

BITS	**DESCRIPTION**
1	'1' = Specify that received 1GBE fabric data must go to the 1GBE ramp packet checker instead of to the Spartan 3AN (for in-system SDRAM programming)
2	'1' = Trigger a mezzanine fault from the host
3	'1' = Enable generation of packets from the 1GBE ramp packet source
4	'1' = Enable generation of packets from the 40GBE ramp packet source 0
5	'1' = Enable generation of packets from the 40GBE ramp packet source 1
6	'1' = Enable generation of packets from the 40GBE ramp packet source 2
7	'1' = Enable generation of packets from the 40GBE ramp packet source 3
29..27	Configures timer firmware what time duration to measure
30	'1' = Request a reset of the firmware from the host
31	'1' = Request a shutdown of the SKARAB LRU from the host (shutdown will only happen if bit 31 of C_WR_BRD_CTL_STAT_1_ADDR is also high)

2.3.2.2. C_WR_LOOPBACK_ADDR

BITS	**DESCRIPTION**
31..0	Directly connected to C_RD_LOOPBACK_ADDR

2.3.2.3. C_WR_ETH_IF_CTL_ADDR

BITS	**DESCRIPTION**
1	'1' = Do a soft reset of 40GBE PHY 0
2	'1' = Do a soft reset of 40GBE PHY 1
3	'1' = Do a soft reset of 40GBE PHY 2
4	'1' = Do a soft reset of 40GBE PHY 3

2.3.2.4. C_WR_MEZZANINE_CTL_ADDR

BITS	**DESCRIPTION**
0	'1' = Mezzanine 0 enable

1	‘1’ = Mezzanine 1 enable
2	‘1’ = Mezzanine 2 enable
3	‘1’ = Mezzanine 3 enable
8	‘1’ = Mezzanine 0 reset
9	‘1’ = Mezzanine 1 reset
10	‘1’ = Mezzanine 2 reset
11	‘1’ = Mezzanine 3 reset
16	Mezzanine 0 clock select: ‘0’ = mezzanine clock, ‘1’ = SKARAB motherboard clock
17	Mezzanine 1 clock select: ‘0’ = mezzanine clock, ‘1’ = SKARAB motherboard clock
18	Mezzanine 2 clock select: ‘0’ = mezzanine clock, ‘1’ = SKARAB motherboard clock
19	Mezzanine 3 clock select: ‘0’ = mezzanine clock, ‘1’ = SKARAB motherboard clock

2.3.2.5. C_WR_FRONT_PANEL_STAT_LED_ADDR

BITS	**DESCRIPTION**
0	‘1’ = Front panel LED 0 ON
1	‘1’ = Front panel LED 1 ON
2	‘1’ = Front panel LED 2 ON
3	‘1’ = Front panel LED 3 ON
4	‘1’ = Front panel LED 4 ON
5	‘1’ = Front panel LED 5 ON
6	‘1’ = Front panel LED 6 ON
7	‘1’ = Front panel LED 7 ON

2.3.2.6. C_WR_RAMP_SOURCE_DESTINATION_IP_3_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet source destination IP address for 40GBE ramp packet generator 3

2.3.2.7. C_WR_RAMP_CHECKER_SOURCE_IP_3_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet checker desired source IP address for 40GBE ramp packet checker 3

2.3.2.8. C_WR_RAMP_SOURCE_DESTINATION_IP_2_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet source destination IP address for 40GBE ramp packet generator 2

2.3.2.9. C_WR_RAMP_CHECKER_SOURCE_IP_2_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet checker desired source IP address for 40GBE ramp packet checker 2

2.3.2.10. C_WR_RAMP_SOURCE_DESTINATION_IP_1_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet source destination IP address for 40GBE ramp packet generator 1

2.3.2.11. C_WR_RAMP_CHECKER_SOURCE_IP_1_ADDR

BITS	**DESCRIPTION**
31..0	Ramp packet checker desired source IP address for 40GBE ramp packet checker 1

2.3.2.12. C_WR_RAMP_SOURCE_DESTINATION_IP_0_ADDR

BITS	DESCRIPTION
31..0	Ramp packet source destination IP address for 40GBE ramp packet generator 0 and the 1GBE ramp packet generator

2.3.2.13. C_WR_RAMP_CHECKER_SOURCE_IP_0_ADDR

BITS	DESCRIPTION
31..0	Ramp packet checker desired source IP address for 40GBE ramp packet checker 0 and the 1GBE ramp packet checker

2.3.2.14. C_WR_NUM_PACKETS_TO_GENERATE_ADDR

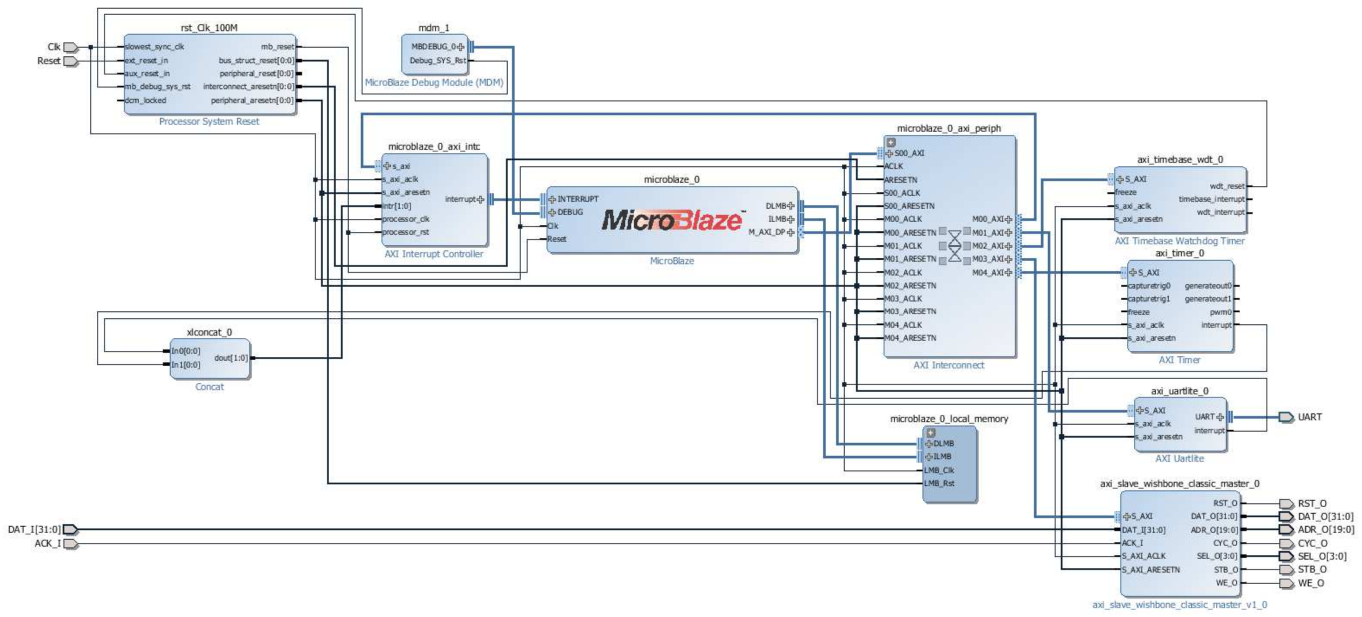
BITS	DESCRIPTION
23..0	Ramp packet source number of packets to generate
31..24	Number of idle clock cycles between each packet

2.3.2.15. C_WR_RAMP_SOURCE_PAYLOAD_WORDS_ADDR

BITS	DESCRIPTION
10..0	Specify the size of the ramp packets to generate. For a cycling packet size between smallest and largest, set to 0

2.4. Microcontroller 'cont_microblaze_wrapper.vhd'

The microcontroller is implemented as a block design using the Vivado Block Design GUI.



The components within the microcontroller are:

- MicroBlaze processor 'microblaze_0' runs out of BRAMs 'microblaze_0_local_memory'. The size of this memory is 128KB.
- Processor System Reset 'rst_clk_100M' is the reset generator. The actual MicroBlaze processor clock frequency is 156.25MHz.
- AXI Interrupt Controller 'microblaze_0_axi_intc' and Concat 'xlconcat_0' handle the interrupts from the timer and the UART.
- MicroBlaze Debug Module (MDM) 'mdm_1'
- AXI Interconnect 'microblaze_0_axi_periph' connects the various AXI peripherals to the MicroBlaze
- AXI Timebase Watchdog Timer 'axi_timebase_wdt_0' provides a watchdog timer to reset the MicroBlaze if it hangs

- AXI Timer 'axi_timer_0' provides a repeating time interval. This is used for status updates (such as driving the QSFP+ Mezzanine LEDs every 100 milliseconds) and timed events (such as retrying DHCP every 10 seconds).
- AXI Uartlite 'axi_uartlite_0' provides a console print out during the execution of the MicroBlaze.
- AXI Slave Wishbone Classic Master 'axi_slave_wishbone_classic_master_0' is a custom peripheral to create a bridge between the AXI bus and the Wishbone bus.

The primary task of the MicroBlaze is to check whether there are any CPU packets waiting to be processed from the 1GBE or 40GBE interfaces. It reads these packets, handles them appropriately and then transmits a response over the corresponding 1GBE or 40GBE interface.

The API to interface with the MicroBlaze over UDP packets is detailed in Section 3.

2.5. Wishbone Interconnect 'wishbone_interconnect.vhd'

Wishbone Interconnect provides the multiplex connection between the single Wishbone master (microcontroller) and the multiple Wishbone slaves. Only a single master is supported. The upper five address bits from the Wishbone master are used to select between the Wishbone slaves.

The standard implemented is Wishbone Classic as detailed in the Wishbone specification 'Wishbone B4 WISHBONE System-on-Chip (SoC) Interconnection Architecture Portable IP Cores'. All transfers are as single read and write standard cycles. All Wishbone slaves are synchronous. The address width of the Wishbone bus is 20 bits and the data width of the Wishbone bus is 32 bits. Further details can be found in the Wishbone specification.

2.6. Wishbone Register 'wishbone_register.vhd'

Wishbone Register is a Wishbone slave that implements 32 read registers (each register is 32 bits wide) and 32 write registers. The read registers are read from the Wishbone bus and provide status feedback to the host (via the microcontroller). The write registers are written from the Wishbone bus and provide control of the board from the host (via the microcontroller).

2.7. Wishbone Flash SDRAM Interface 'wishbone_flash_sdram_interface.vhd'

Wishbone Flash SDRAM Interface is a Wishbone slave. It provides the following functionality:

- Access from the host to the parallel NOR flash on the SKARAB motherboard. This is used for in-system reprogramming of the non-volatile FPGA boot image stored in the 1Gb NOR flash.
- Access to the Spartan 3AN FPGA on the SKARAB motherboard. This is used for programming the warm boot image in the 256Mb SDRAM on the SKARAB motherboard. This is used as a basic continuity check of the connections between the Virtex 7 FPGA and the Spartan 3AN FPGA. This is also used to control sideband signals between the Virtex 7 FPGA and the Spartan 3AN FPGA that provide handshaking for the warm boot process.
- Access to the ICAPE component which is used to trigger a reconfiguration of the Virtex 7 FPGA.
- Access to the SPI interface of the Spartan 3AN embedded flash. This is used for in-system reprogramming of the Spartan 3AN FPGA firmware. The SPI interface to the Spartan 3AN is also used to read the firmware version of the Spartan 3AN.

'icape_controller.vhd' uses a state machine to create the required timing to interface with the ICAPE component 'ICAPE2'. It is controlled by two registers in the register map of Wishbone Flash SDRAM Interface.

'isp_spi_programmer.vhd' uses a state machine to create the required timing to interface with the embedded SPI flash in the Spartan 3AN FPGA. It is controlled by two registers in the register map of Wishbone Flash SDRAM Interface.

The address space of the Wishbone Flash SDRAM Interface is split into two portions. The lower address space relates to memory accesses. This address space is used when reading or writing to the flash or SDRAM as these accesses are direct – a Wishbone read or write transaction is translated into a flash or SDRAM read or write transaction. The upper address space relates to register accesses. This address space is used when reading or writing registers in the Wishbone Flash SDRAM Interface.

The Wishbone Flash SDRAM Interface can operate in different modes. In Flash mode, a state machine is used to create the required timing to interface with the NOR flash. Wishbone writes are translated directly into asynchronous BPI flash writes. Wishbone reads are translated directly into asynchronous BPI flash reads.

In SDRAM program mode, data received from the 1GBE fabric interface is written to the Spartan 3AN (which then buffers the data and writes out to the SDRAM).

In SDRAM read mode, a Wishbone read is translated into reading 32 bits of SDRAM data from the Spartan 3AN. The Spartan 3AN pre-reads the SDRAM data so that it can be read with minimal latency. A Wishbone write does not make sense in this mode.

Accesses to the SPI programmer and ICAPE controller are done through the register interface of the Wishbone Flash SDRAM Interface. As a result, they operate independently of the modes discussed above.

2.7.1. Wishbone Flash SDRAM Interface Register Map

ADDRESS	**DESCRIPTION**
0	C_OUTPUT_MODE_REG
1	C_UPPER_ADDRESS_REG
2	C_CONFIG_IO_REG
3	C_GBE_STATISTICS_REG
4	C_ICAPE_DATA_REG
5	C_ICAPE_CTL_REG
6	C_ISP_SPI_ADDRESS_REG
7	C_ISP_SPI_DATA_CTRL_REG
8	C_CONTINUITY_TEST_OUTPUT_REG

2.7.1.1. C_OUTPUT_MODE_REG

Write:

BITS	**DESCRIPTION**
1..0	Specifies the operating mode: ‘00’ = NOR flash mode, ‘01’ = SDRAM program mode, ‘10’ = SDRAM read mode
2	‘1’ = Enable the output flash drivers, ‘0’ = leave flash pins on the Virtex 7 FPGA as high impedance

Read: nothing

2.7.1.2. C_UPPER_ADDRESS_REG

Write:

BITS	**DESCRIPTION**
16..0	Specify the upper 17 bits of the parallel flash bus (the Wishbone address only provides the lower 12 bits)

Read: nothing

2.7.1.3. C_CONFIG_IO_REG

Write:

BITS	**DESCRIPTION**
0	‘1’ = Clear SDRAM
1	‘1’ = Finished writing image to SDRAM
2	‘1’ = About to do warm boot from SDRAM
3	‘1’ = Finished doing warm boot from SDRAM
4	‘1’ = Reset the SDRAM read address to 0
5	‘1’ = Debug mode to read SDRAM data
8	‘1’ = Debug continuity test mode
9	‘1’ = Debug continuity test low

10	'1' = Debug continuity test high

Read:

BITS	**DESCRIPTION**
6	'1' = Busy booting from SDRAM
7	'1' = Stall writing of SDRAM image to Spartan 3AN

2.7.1.4. C_GBE_STATISTICS_REG

To speed up the programming of the warm boot FPGA image into the SDRAM, the SDRAM programming packets are not processed by the microcontroller. Rather, the SDRAM programming packets are streamed directly from the host to Wishbone Flash SDRAM Interface via the fabric interface of the 1GBE. This significantly reduces the programming time. However, the host needs to know whether all packets were correctly received before triggering a warm reboot. Basic Ethernet packet statistics relating specifically to the SDRAM programming packets is maintained by Wishbone Flash SDRAM Interface.

Write:

BITS	**DESCRIPTION**
0	'1' = Clear Ethernet statistics counters

Read:

BITS	**DESCRIPTION**
15..0	Number of received Ethernet packets
23..16	Number of packets received marked as bad (failed FCS checking)
31..24	Number of packets received marked as overrun (receive FIFO overflow)

2.7.1.5. C_ICAPE_DATA_REG

Write:

BITS	**DESCRIPTION**
31..0	Data to write to the ICAPE component

Read:

BITS	**DESCRIPTION**
31..0	Data read from the ICAPE component

2.7.1.6. C_ICAPE_CTL_REG

Write:

BITS	**DESCRIPTION**
0	'1' = Read transaction, '0' = Write transaction
1	'1' = Start transaction

Read:

BITS	**DESCRIPTION**
0	'1' = Transaction complete

2.7.1.7. C_ISP_SPI_ADDRESS_REG

Write:

BITS	**DESCRIPTION**
22..0	SPI transaction address
31..23	Number of bytes

Read: nothing

2.7.1.8. C_ISP_SPI_DATA_CTRL_REG

Write:

BITS	DESCRIPTION
7..0	Write byte data
8	Rising edge writes single byte into write buffer
9	Rising edge reads single byte from read buffer
12..10	Command: '000' = fast read command, '001' = buffer write command, '010' = buffer program command, '011' = sector erase command, '100' = status command
13	'1' = Start transaction

Read:

BITS	DESCRIPTION
7..0	Read byte data
8	'1' = Transaction complete

2.7.1.9. C_CONTINUITY_TEST_OUTPUT_REG

Write:

BITS	DESCRIPTION
31..0	Continuity test output

Read:

BITS	DESCRIPTION
15..0	Continuity test input

2.8. Wishbone 1-Wire 'wishbone_one_wire.vhd'

The Wishbone 1-Wire is a Wishbone slave. It uses an OpenCores Wishbone component 'sockit_owm.v'. 'wishbone_one_wire.vhd' simply instantiates 'sockit_owm.v' and provides the required Wishbone signaling and address translation. The register map for 'sockit_owm.v' is detailed in the OpenCores 'sockit_owm.v' datasheet (located at "http://opencores.org/project,sockit_owm").

'socket_owm.v' creates a single Wishbone slave with a configurable number of 1-wire interfaces. On the SKARAB motherboard, five 1-wire interfaces exist and are numbered as follows:

- 0: SKARAB motherboard 1-wire interface
- 1: Mezzanine 0 1-wire interface
- 2: Mezzanine 1 1-wire interface
- 3: Mezzanine 2 1-wire interface
- 4: Mezzanine 3 1-wire interface

2.9. Wishbone I2C 'wishbone_i2c.vhd'

The Wishbone I2C is a Wishbone slave. It uses an OpenCores Wishbone component 'i2c_master_top.vhd'. 'wishbone_i2c.vhd' simply instantiates 'i2c_master_top.vhd' and provides the required address translation. The register map for 'i2c_master_top.vhd' is detailed in the OpenCores 'i2c_master_top.vhd' datasheet (located at "http://opencores.org/project,i2c").

There are five instantiations of Wishbone I2C in the SKARAB motherboard. These are as follows:

- 0: SKARAB motherboard I2C interface
- 1: Mezzanine 0 I2C interface
- 2: Mezzanine 1 I2C interface
- 3: Mezzanine 2 I2C interface
- 4: Mezzanine 3 I2C interface

2.10. 10GBE MAC (1GBE) 'kat_ten_gb_eth.vhd'

The OPB interface 'opb_attach.v' of 'kat_ten_gb_eth.vhd' was replaced with 'wishbone_ten_gb_eth_attach.vhd' making it a Wishbone slave. The functionality of this interface is the same as before and has not been changed. It is used to transmit and receive CPU packets over the 1GBE, configure the ARP cache in the transmit MAC and read and write the 'kat_ten_gb_eth.vhd' register interface. The register map of 'kat_ten_gb_eth.vhd' is the same as ROACH2 and will not be detailed here.

The fabric packet interfaces of 'kat_ten_gb_eth.vhd' are unchanged from ROACH2 and will not be detailed here.

Debug ports have been added to the top level of 'kat_ten_gb_eth.vhd' to access the current source IP address, the source MAC address, whether the fabric is enabled, the source port address, the source gateway, the multicast IP address and the multicast IP address mask.

2.11. 40GBE MAC 'ska_forty_gb_eth.vhd'

The behavior of the 40GBE MAC is very similar to the 10GBE MAC. The main difference is that the width of the fabric packet interface has been increased to 256 bits to support 40Gbps. Otherwise, the behavior and signaling of the fabric packet interface is identical to the 10GBE MAC.

The 40GBE MAC consists of the following components:

- 'wishbone_forty_gb_eth_attach.vhd': This is a Wishbone slave. The functionality is the same as the Wishbone slave in the 10GBE MAC. The register map of 'ska_forty_gb_eth.vhd' is the same as ROACH2.
- 'ska_fge_tx.vhd': The functionality is the same as 'tge_tx.v' in the 10GBE MAC.
    - Data from the transmit fabric interface is encapsulated in UDP/IP Ethernet packets. Jumbo packets are supported. A state machine is used to construct the required formatting for the fabric interface UDP/IP packets.
    - To support 40GBE, the width of the fabric packet interface FIFOs has been increased to 256 bits. To provide a similar feel to the 10GBE MAC, this 256 bit bus is handled as four 64 bits busses in parallel. 'app_tx_data' is 256 bits wide. 'app_tx_valid' selects which (or all) of the four 64 bits busses contain valid data. So 'app_tx_valid(0)' relates to 'app_tx_data(63..0)', 'app_tx_valid(1)' relates to 'app_tx_data(127..64)' and so on. Valid data must always be filled from the lowest 64 bits upwards. For example, if a packet consists of 1152 bits, then:
        - Clock 0: app_tx_data(255..0) = valid data, app_tx_valid(3..0) = "1111"
        - Clock 1: app_tx_data(255..0) = valid data, app_tx_valid(3..0) = "1111"
        - Clock 2: app_tx_data(255..0) = valid data, app_tx_valid(3..0) = "1111"
        - Clock 3: app_tx_data(255..0) = valid data, app_tx_valid(3..0) = "1111"
        - Clock 4: app_tx_data(255..128) = don't care, app_tx_data(127..0) = valid data, app_tx_valid(3..0) = "0011"
    - An ARP cache is used to determine the destination MAC address based on the destination IP address assigned to a packet. The microcontroller fills the ARP cache based on responses it receives to ARP requests. ARP requests are continuously generated by the microcontroller once DHCP has completed on an interface.
    - 'ska_fge_tx.vhd' also handles the moving of data and signals from the system clock domain to the 40GBE PHY transmit clock domain.
    - A transmit buffer allows the microcontroller to transmit packets over the 40GBE interface. The transmit buffer only supports the standard Ethernet frame size and only two packets can be buffered. Note that it is the responsibility of the microcontroller to construct the packet structure for CPU packets. One difference to 'tge_tx.v' in the 10GBE MAC is that read back of the CPU transmit buffer is not supported (in other words, the microcontroller cannot read what packet is currently waiting in the transmit buffer to be sent). Reading the 256-bit buffer memory over the 32-bit Wishbone interface causes complications. This does not impact the functionality of the transmit buffer as it is not necessary to read the contents of the CPU transmit buffer before sending. The CPU transmit buffer is 256 bits wide so all CPU transmit packets are rounded up to the next 256 bit boundary.
    - 'tx_valid', 'tx_end_of_frame', 'tx_data', 'tx_dest_ip', 'tx_dest_port', 'tx_overflow', 'tx_afull' are synchronous to 'clk'.
    - 'DAT_I', 'DAT_O', 'ACK_O', 'ADR_I', 'CYC_I', 'SEL_I', 'STB_I', 'WE_I' are synchronous to 'clk'.
- 'ska_mac_tx.vhd': The functionality is the same as 'mac_tx.v' in the 10GBE MAC. Data from 'ska_fge_tx.vhd' is encapsulated in Ethernet MAC frames and transmitted over XLGMII to the 40GBE PHY.
    - The XLGMII framing is the same as XAUI.
    - The Ethernet MAC Frame Check Sequence is calculated by 'ska_mac_tx_crc.vhd'. A soft CRC implementation is used. Since transmit packets can only end of certain defined byte boundaries (bit 16, 64, 80, 128, 144, 192, 208, 256), only certain CRC calculations need to be performed.

  Consequently, no special processing is needed to handle the 'remainder' of a transmit packet that doesn't end on a 256-bit boundary.
  - The minimum Inter Frame Gap between transmitted packets is 40 idle characters (0x07). This is set by the state machine which constructs the Ethernet MAC frames. Tests with jumbo frames of 9216 bytes have shown that the transmitter can maintain in excess of 38Gbps at the application layer (as per the requirements specification).
  - The transmit LED is on when the transmitter is out of reset and the transmitter is enabled. A transmitter is only enabled if a cable is plugged into the corresponding port of the QSFP+ Mezzanine.
  - The transmit LED flashes if a packet is transmitted. If there have been no packets transmitted within the last 1.7 seconds, then the transmit LED stops flashing. This long timeout is because the QSFP+ Mezzanine LEDs are only updated once a second.
  - 'xlgmii_txd', 'xlgmii_txc', 'xlgmii_txled' are synchronous to 'xlgmii_txclk'.
- 'ska_runt_filt_rx.vhd': This component filters runt packets received from the 40GBE PHY over XLGMII and prevents them from reaching the 40GBE MAC receive data path. Runt packets can be generated during plugging and unplugging events. A runt packet is identified as a packet shorter than 62 bytes in length. The Ethernet standard for the shortest packet is 64 bytes. By filtering runt packets, the 40GBE MAC receive data path is simplified.
  - 'ska_runt_filt_rx.vhd' also handles the setting of the receive LEDs. The receive LED is on when link synchronization has been achieved (the four lanes are locked and aligned). The receive LED flashes if a packet is received. If there have been no packets received within the last 1.7 seconds, then the receive LED stops flashing. This long timeout is because the QSFP+ Mezzanine LEDs are only updated once a second.
- 'ska_mac_rx.vhd': The functionality is the same as 'mac_rx.v' in the 10GBE MAC. Packets that have successfully passed through the runt filter are stripped of the Ethernet MAC layer and XLGMII framing.
  - As mentioned before, the data width is 256 bits in order to maintain 40GBE throughput. Pipelining is required to achieve a clock rate of 156.25MHz. Pipelining is also required in the Frame Check Sequence calculation which uses multiple clock cycles to calculate the 'remainder' at the end of packet (details to follow). The shortest Ethernet packet is only 64 bytes long which translates to just two clock cycles at a 256-bit wide data bus. Further, no assumptions can be made regarding the minimum Inter Frame Gap as tests have shown that some 40GBE NICs generate no Inter Frame Gap. The consequence of this is that it is possible for a second 40GBE packet to be arriving while the first 40GBE packet is still being processed. To handle this scenario, two 'ska_mac_rx.vhd' components are instantiated in parallel. They alternate between the received 40GBE packets from the runt filter. Their outputs are buffered in small FIFOs and then combined back into a single data stream (the order of the packets is maintained). By doing this, the firmware is able to process successive minimum size 40GBE packets at minimum Inter Frame Gap
  - The Ethernet MAC Frame Check Sequence is calculated by 'ska_mac_rx_crc_multi.vhd'. A soft CRC implementation is used. The received packets can end on any byte boundary. This requires 32 different CRC calculations to all happen in parallel. Not only does this excessively bloat the design, but it also causes the Vivado tools to struggle to meet timing at 156.25MHz. An alternative implementation is used:
    - A 256-bit CRC is calculated for all portions of the packet that are a full 256 bits wide.
    - The 'remainder' of the packet is split into four 64 bit portions. One clock cycle is dedicated to calculating the CRC of each 64 bit portion. Each 64 bit portion requires a CRC calculation for 8, 16, 24, 32, 40, 48, 56 and 64 bits. If there is no more 'remainder' left to calculate then the CRC is finished and no further clock cycles are required. This implementation reduces the number of required CRCs to only nine (as opposed to 32 for the 'brute-force' method).
  - If a packet fails the Frame Check Sequence, then it is marked as bad.
  - A generic parameter allows the HDL designer to specify whether Frame Check Sequence checking is required on received Ethernet packets.
  - 'ska_mac_rx.vhd' also handles the moving of data and signals from the 40GBE PHY receive clock domain to the system clock domain.
  - 'xlgmii_rxd', 'xlgmii_rxc', 'xlgmii_rxled' are synchronous to 'xlgmii_rxclk'.
- 'ska_fge_rx.vhd': The functionality is the same as 'tge_rx.v' in the 10GBE MAC. Packet data is received from 'ska_mac_rx.vhd'. It is first filtered based on destination MAC address. Packets that match the MAC address of the 40GBE MAC, or broadcast packets or multicast packets are passed to the next layer of filtering. The next layer of filtering determines whether the packet is destined for the fabric interface application layer or the CPU buffer. Only packets that match the destination IP address (or the masked multicast IP address), port address and UDP/IP packet type are passed to the application layer (fabric

interface). All other packets are passed to the CPU buffer.

- The CPU buffer can only process standard Ethernet packet sizes (not jumbo frames). The CPU buffer allows up to eight packets to be stored before packets are dropped. The microcontroller reads the packets from the CPU packet buffer over the Wishbone slave interface. The CPU buffer is 256 bits wide so all CPU packets are rounded up to the next 256 bit boundary.
- The fabric application layer receive FIFOs follow the same structure as in 'tge_rx.v'. The only difference is that the width of the bus has been increased to 256 bits in order to support 40GBE at 156.25MHz. In the same way as the fabric application layer transmit FIFOs, this 256 bit bus has been split into four 64 bit busses in parallel. 'app_rx_data' is 256 bits wide. 'app_rx_valid' selects which (or all) of the four 64 bits busses contain valid data. So 'app_rx_valid(0)' relates to 'app_rx_data(63..0)', 'app_rx_valid(1)' relates to 'app_rx_data(127..64)' and so on.
- The fabric application layer receive FIFOs are able to handle jumbo packets.
- 'rx_valid', 'rx_end_of_frame', 'rx_data', 'rx_source_ip', 'rx_source_port', 'rx_bad_frame', 'rx_overrun', 'rx_overrun_ack' are synchronous to 'clk'.
- Debug ports have been added to the top level of 'kat_ten_gb_eth.vhd' to access the current source IP address, the source MAC address, whether the fabric is enabled, the source port address, the source gateway, the multicast IP address and the multicast IP address mask.

2.11.1. 40GBE Register Map 'wishbone_forty_gb_eth_attach.vhd'

ADDRESS	DESCRIPTION
0	REG_LOCAL_MAC_1
1	REG_LOCAL_MAC_0
3	REG_LOCAL_GATEWAY
4	REG_LOCAL_IPADDR
6	REG_BUFFER_SIZES
8	REG_VALID_PORTS
12	REG_MC_RECV_IP
13	REG_MC_RECV_IP_MASK

2.11.1.1. REG_LOCAL_MAC_1

Write:

BITS	DESCRIPTION
15..0	Source MAC address (47..32)

Read: nothing

2.11.1.2. REG_LOCAL_MAC_0

Write:

BITS	DESCRIPTION
31..0	Source MAC address (31..0)

Read: nothing

2.11.1.3. REG_LOCAL_GATEWAY

Write:

BITS	DESCRIPTION
7..0	Address in ARP cache which contains MAC address of default gateway

Read: nothing

2.11.1.4. REG_LOCAL_IPADDR

Write:

BITS	DESCRIPTION
31..0	Source IP address

Read: nothing

2.11.1.5. REG_BUFFER_SIZES

Write:

BITS	DESCRIPTION
7..0	Write with X"00" to acknowledge reading a packet from the receive buffer
23..16	Packet size in transmit buffer (packet ready to be sent)

Read:

BITS	DESCRIPTION
7..0	Size of packet waiting in receive buffer (X"00" when microcontroller has acknowledged reading packet)
23..16	Size of packet waiting in transmit buffer (X"00" when 40GBE MAC has finished transmitting packet)

2.11.1.6. REG_VALID_PORTS

Write:

BITS	DESCRIPTION
15..0	Source port address
16	'1' = fabric application interface enabled
24	'1' = soft reset (set back to '0' when reset complete)

Read: nothing

2.11.1.7. REG_MC_RECV_IP

Write:

BITS	DESCRIPTION
31..0	Multicast IP address

Read: nothing

2.11.1.8. REG_MC_RECV_IP_MASK

Write:

BITS	DESCRIPTION
31..0	Multicast IP address mask

Read: nothing

2.11.2. 40GBE Address Map 'wishbone_forty_gb_eth_attach.vhd'

REGION	START ADDRESS	END ADDRESS
Registers	0x0000	0x07FF
CPU transmit packet buffer	0x1000	0x17FF
CPU receive packet buffer	0x2000	0x27FF
ARP cache	0x3000	0x37FF

2.12. 40GbE Physical Interface (PHY)

The IEEE standard 802.3 defines the entire family of Ethernet interfaces. There are many forms of Ethernet defined in the standard, however, the 40GbE PHY was designed to support 40GBase-CR4 and 40GBase-SR4.

The 40GbE PHY is made up of 3 components which relate to key clauses of the 40GbE specification:

1. Reconciliation Sublayer (RS), as defined by IEEE Std 802.3™ Clause 81
2. Physical Coding Sublayer (PCS), as defined by IEEE Std 802.3™ Clause 82
3. Physical Medium Attachment (PMA) sublayer, as defined by IEEE Std 802.3™ Clause 83

The Physical Medium Dependant (PMD) sublayer is medium-dependent and is implemented outside of the FPGA fabric. It is therefore not described in this document.

The module "IEEE802_3_XL_PHY.vhd" explained in the next section encapsulates the RS, PCS and PMA. A top level wrapper is provided as a simpler way to instantiate the 40GbE PHY and is explained in section 2.12.6 below.

2.12.1. 40GbE PHY "IEEE802_3_XL_PHY.vhd"

The service interface to the 40GbE PHY is a custom interface containing an array of 4 XLGMII interfaces running at 156.25Mhz, which is a quarter of the standard rate. This gives an interface with an effective 256 bits of data and 32 control bits.

The 40GbE PHY has been designed to be an autonomous module. Standard usage should not require any knowledge of the underlying workings and it does not require any higher level control. However, the 40GbE PHY does contain a reset signal in case there is a module failure as a result of internal or external errors.

Furthermore the 40GbE PHY does provide status signals from different levels of the receive side, as described below:

1. BLOCK_LOCK_O[3:0] is the output of the Block lock state machine. If the clock on that lane has been recovered correctly and the PMA is functioning correctly and outputting valid 64/66b encoded data blocks then this signal will be asserted. A low Bit Error Rate (BER) will not cause BLOCK_LOCK_O to be de-asserted, but a high BER will. This signal would also be asserted in the case that a 10GBASE-R signal was being received.

2. AM_LOCK_O[3:0] is the output of the Alignment Lock state machine. If asserted it signifies that valid Alignment markers from any TX lane are being received from the underlying FIFO, and that all layers below are operating correctly. A low BER will not cause AM_LOCK_O to be de-asserted, but a high BER will.

3. ALIGN_STATUS_O is the output of the Lane Deskew state machine. If asserted it signifies that four receive lanes have been aligned together.

The 40GbE PHY contains its entire clocking infrastructure internally. The 40GbE PHY top level module takes in a reference clock called "SYS_CLK_I" and its inputs and outputs are in this clocking domain, with the exceptions mentioned below. Appropriate clock domain crossing circuitry has already been included internally to achieve this.

The following inputs and outputs are not in the "SYS_CLK_I" domain:

1. "TXN_I", "TXP_I", "RXN_O", "RXP_O" connect to the transceiver serial input and serial output pins.
2. "GTREFCLK_PAD_N_I", GTREFCLK_PAD_P_I" connect to the transceivers reference clock input pins.
3. "GTREFCLK_O" is a clock signal output derived from "GTREFCLK_PAD_N/P_I". If used it should accessed via a clock buffer first.

2.12.2. *Constant and Type Defines "IEEE802_3_XL.vhd"*

This file contains all of the Component Declarations, PHY Global Constants and PHY custom types.
Important types are:

1.	BYTE_t	-> is an 8 bit std_logic_vector.
2.	BYTE_ARRAY_t	-> is array of type BYTE_t.
3.	XLGMII_t	-> is a type containing .C[7:0] for Control bits and .D[63:0] for Data bits.
4.	XLGMII_ARRAY_t	-> is array of XLGMII_t.
5.	BLOCK_t	-> is a type containing .H[1:0] for Header and .P[63:0] for Payload.
6.	BLOCK_ARRAY_t	-> is array of BLOCK_t.

2.12.3. *40GbE RS "IEEE802_3_XL_RS.vhd"*

The 40GbE Reconciliation Sub-layer is responsible for interfacing the MAC layer to the PHY layer.

The 40GbE RS multiplexes and de-multiplexes between the custom "4 XLGMII" interface and a standard XLGMII interface. The XLGMII interface is defined by IEEE Std 802.3™ Clause 81.

Receive side clock correction is also performed in this module.

Transmit side clock correction is not performed. This is because SKARAB fans out the output from a single oscillator to the FPGA reference clock input and transceiver reference clock inputs. A FIFO is present to account for the clock domain crossing from the MAC clock "SYS_CLK_I" to the transmit PHY clock which is generated internally.

2.12.4. *40GbE PCS "IEEE802_3_XL_PCS.vhd"*

The service interface to the 40GbE PCS is XLGMII as defined by IEEE Std 802.3™ Clause 81.

The block diagram show the TX and RX signal paths. For this document the PCS block diagram is separated into TX and RX due to space limitations.

Explanations of each module within the PCS are given underneath. These explanations follow the same order as Ethernet Frame would travel. [Follow the Arrows]

2.12.4.1. *Transmit Path*

The Transmit path takes a single XLGMII stream and encodes it into 64B/66B blocks followed by a scrambling algorithm. The Encoded and scrambled blocks are then distributed into 4 streams and Alignment Markers are added to align the streams on the receive side. Finally the 4 streams are passed to an XLAUI PMA sub Layer.

This is illustrated by Figure 2-1: PCS Transmit Block Diagram.

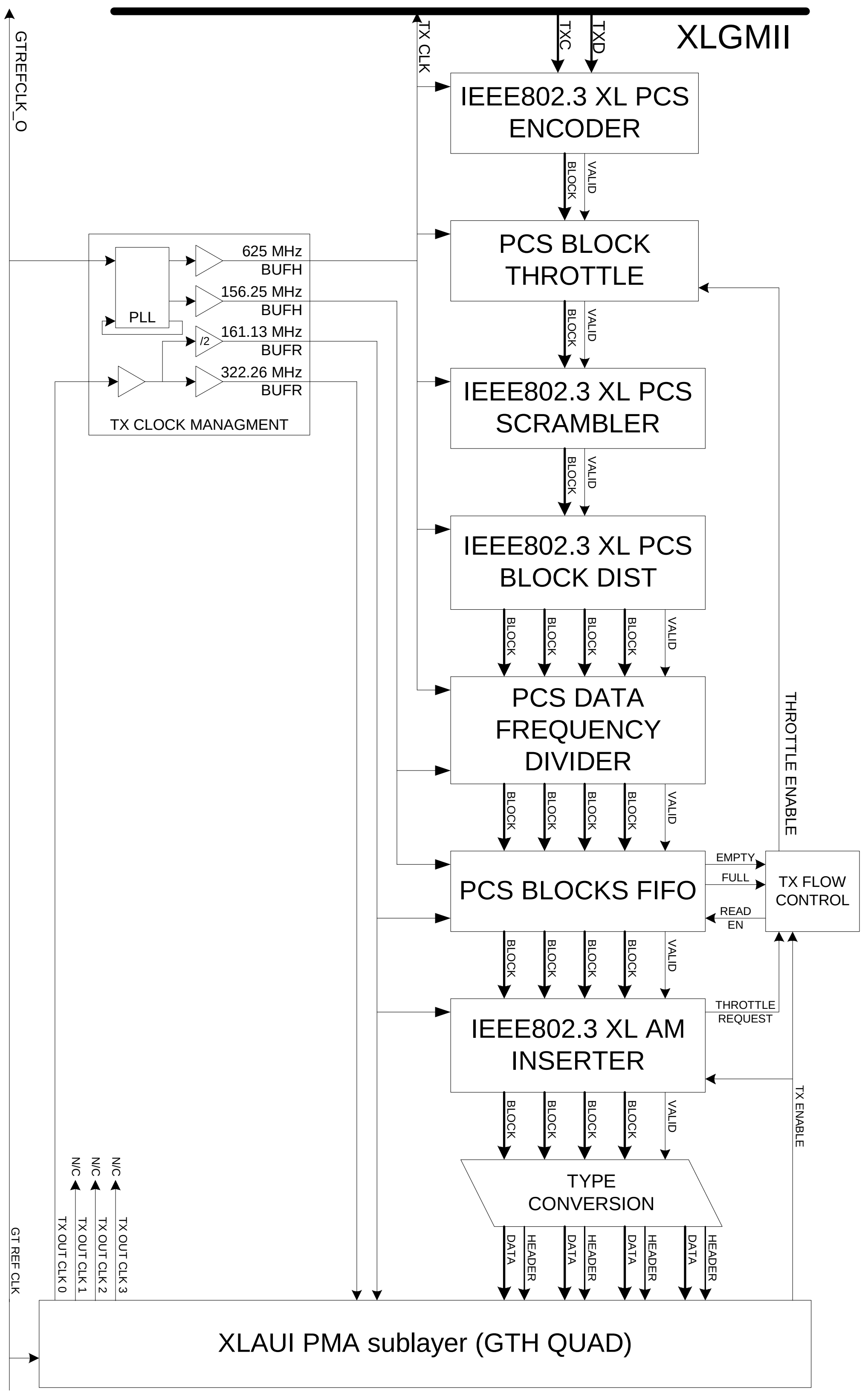


Figure 2-1: PCS Transmit Block Diagram

1. IEEE802_3_XL_PCS_ENCODER is the implementation of the Transmit State Machine of IEEE Std 802.3™ – 2012 FIGURE 82-14. It encodes an XLGMII stream into a stream of unscrambled 64B/66B transmission codes.

2. PCS_BLOCK_THROTTLE is a module to ensure that the TX rate before the Scrambler is always within the tolerances of the underlying modules. Mainly this solves the rate issue resulting from the insertion of Alignment Markers.

3. IEEE802_3_XL_PCS_SCRAMBLER implements the scrambler polynomial specified in Clause 49.2.6, and illustrated by IEEE Std 802.3™ – 2012 Figure 49.8.

4. PCS_DATA_FREQUENCY_DIVIDER moves data from a faster clock domain to a clock domain with is one quarter of the frequency. Both clocks must be sourced from the same PLL so that they are phase aligned within acceptable tolerances as determined by the FPGA tools.

5. IEEE802_3_XL_PCS_BLK_DIST distributes scrambled 64B/66B transmission codes (blocks) into 4 PCS Lanes using a round robin algorithm from lowest to highest lane. See IEEE Std 802.3™ – 2012 Figure 82.6

6. PCS_BLOCKs_FIFO is a FIFO module to handle the crossing of clocks between the PCS and the PMA. The PMA_TX runs at a faster clock and receives data 32 out of 33 clock cycles.

7. IEEE802_3_XL_PCS_AM_INSERTER inserts unique alignment marker blocks into each of the 4 streams after every 16383 blocks per lane. See IEEE Std 802.3™ – 2012 Figure 82.7

2.12.4.2. Receive Path

The Receive path takes four separate streams of 66bit data from the XLAUI PMA sub Layer. It aligns it to Block Edges using the Header bits as markers. It then removes any skew and aligns the four streams to the same order in which they were transmitted. These are then multiplexed together, descrambled and decoded back into a single XLGMII stream.

This is illustrated by Figure 2-2: PCS Receive Block Diagram.

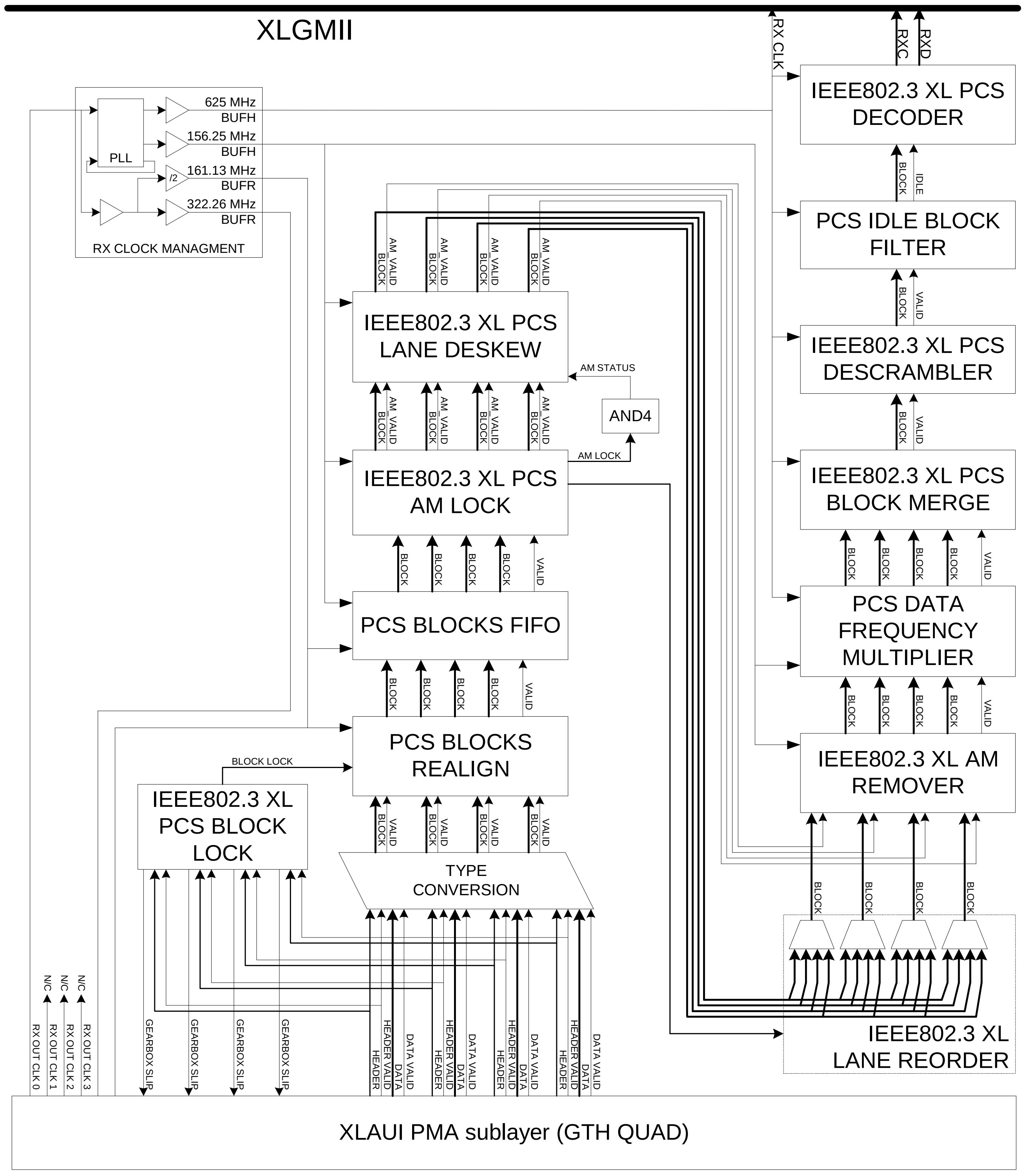


Figure 2-2: PCS Receive Block Diagram

1. IEEE802_3_XL_BLOCK_LOCK is the implementation of the Block Lock State Machine of IEEE Std 802.3™ – 2012 FIGURE 82-10. It ensures output data of the PMA is aligned to scrambled 64B/66B transmission codes block boundaries.

2. PCS_BLOCKs_REALIGN ensures that the 4 streams of data are valid at the same time. The 4 PMA_RX modules transmit data 32 out of 33 clock cycles and these cycles are not synchronized.

3. PCS_BLOCKs_FIFO is a FIFO module to handle the crossing of clocks between the PMA and the PCS. The PMA_RX runs at a faster clock and transmits data 32 out of 33 clock cycles.

4. IEEE802_3_XL_PCS_AM_LOCK is the implementation of the Alignment Marker Lock State Machine of IEEE Std 802.3™ – 2012 FIGURE 82-11.

5. IEEE802_3_XL_PCS_LANE_DESKEW removes all inter-lane skew. According to IEEE Std 802.3™ – 2012 TABLE 82-5 this module must be able to tolerate a maximum skew of 180ns. This implementation was designed to tolerate a skew of 204.8ns as a result of order of two sized memory components.

6. IEEE802_3_XL_PCS_LANE_REORDER accounts for TX and RX lanes not matching as a result of cable crossover or mismatch. Each Lane identifies itself with its unique Alignment Marker code and this module ensures that the lanes are placed in the right order.

7. IEEE802_3_XL_PCS_AM_REMOVER removes the Alignment Markers from the stream. This creates a rate mismatch which must be dealt with higher up, however, at this stage that data is just marked invalid.

8. PCS_DATA_FREQUENCY_MULTIPLIER moves data from a slower clock domain to a clock domain with is four times the frequency. Both clocks must be sourced from the same PLL so that they are phase aligned within acceptable tolerances as determined by the FPGA tools.

9. PCS_BLK_MERGE performs the reverse of IEEE802_3_XL_PCS_BLK_DIST. It multiplexes the 4 PCS Lanes into a single stream of scrambled 64B/66B transmission codes (blocks) in the same order that the stream was transmitted.

10. IEEE802_3_XL_PCS_DESCRAMBLER implements the "reverse" of the scrambler module. Its implementation matches that illustrated by IEEE Std 802.3™ – 2012 Figure 49.10.

11. PCS_IDLE_BLOCK_FILTER is a module to account for the rate change as a result of Alignment Marker removal. This module removes invalid blocks from within Frames and multiplies IDLE blocks where necessary.

12. IEEE802_3_XL_PCS_DECODER is the implementation of the Receive State Machine of IEEE Std 802.3™ – 2012 FIGURE 82-15. It encodes a stream of unscrambled 64B/66B transmission codes into an XLGMII stream.

2.12.5. 40GbE PMA "IEEE802_3_XL_PMA"

The 40GbE PMA is an implementation of the XLAUI standard as defined by IEEE Std 802.3™ – 2012 Annex 83A. It uses 4 Xilinx GTH transceivers running at 10.3125 GHz to achieve the specifications of

The PMA is implemented as a XLAUI core (provided by Xilinx as the only recommended way to instantiate their GTH Transceivers) that was modified to improve clock structure. The details of this core can be found in the Xilinx transceiver wizard (XLAUI) documentation.

The PMA includes HDL code to perform an Endianess change so that the PCS and XLAUI cores are compatible.

2.12.6. 40GbE PHY Wrapper "IEEE802_3_XL_PHY_top.vhd"

The 40GbE PHY is instantiated via a wrapper module called "IEEE802_3_XL_PHY_top". The purpose of this wrapper is to simplify instantiation by converting any custom types to std_logic/std_logic_vector and reducing the signal set provided.

All the PHY RX status signals are "merged" together into a single signal named "LINK_UP_O".

The signal "TEST_PATTERN_ERROR_O" is asserted for one clock cycle every time the test pattern error counter changes. The test pattern monitoring is only active when "TEST_PATTERN_EN_I" is asserted. The link partner must also have its test pattern activated to use this check.

2.13. Wishbone Slave Address Map

WISHBONE SLAVE	**START ADDRESS**
BOARD_REGISTER_ADDR (board level control and status registers)	0x00000
DSP_REGISTER_ADDR (SKA SA register address space)	0x08000
FLASH_SDRAM_SPI_ICAPE_ADDR (flash, SDRAM, SPI and ICAPE interface)	0x10000
ONE_WIRE_ADDR (1-wire interface controller for SKARAB motherboard and all four mezzanine sites)	0x18000
I2C_0_ADDR (I2C interface controller for SKARAB motherboard)	0x20000
I2C_1_ADDR (I2C interface controller for mezzanine site 0)	0x28000
I2C_2_ADDR (I2C interface controller for mezzanine site 1)	0x30000
I2C_3_ADDR (I2C interface controller for mezzanine site 2)	0x38000
I2C_4_ADDR (I2C interface controller for mezzanine site 3)	0x40000
ONE_GBE_MAC_ADDR (1GBE interface for registers, CPU transmit buffer, CPU receive buffer and ARP cache programming)	0x48000
FORTY_GBE_MAC_0_ADDR (40GBE 0 interface for registers, CPU transmit buffer, CPU receive buffer and ARP cache programming)	0x50000
FORTY_GBE_MAC_1_ADDR(40GBE 1 interface for registers, CPU transmit buffer, CPU receive buffer and ARP cache programming)	0x58000
FORTY_GBE_MAC_2_ADDR(40GBE 2 interface for registers, CPU transmit buffer, CPU receive buffer and ARP cache programming)	0x60000
FORTY_GBE_MAC_3_ADDR(40GBE 3 interface for registers, CPU transmit buffer, CPU receive buffer and ARP cache programming)	0x68000

3. Host Management API

The SKARAB motherboard is controlled and status read via UDP/IP packets. The packets can be sent to the SKARAB over a 1GBE or 40GBE interface. Almost all commands consist of a request packet and a corresponding response packet. The format of these packets will be detailed in the following sub-sections. Almost all commands are synchronous: a request packet is sent to the SKARAB and a response packet is expected. The only case where this does not apply is when the SKARAB reboots or shuts down as a result of receiving the packet. Also, the packets to program the SDRAM with a boot FPGA image do not generate responses as this would significantly increase the SDRAM reconfiguration time.

The UDP port address of the control packets (packets used to control and read status of the SKARAB) is 0x7778. The control packets are handled by the microcontroller. The UDP port address of the fabric packets is 0x7148. The fabric packets are generated and handled directly by the FPGA fabric firmware. The subnet mask is 255.255.255.0.

Each type of packet request has a command ID. The value of the command ID in the response is the command ID incremented by one. Below is a table of the currently supported commands and their command IDs:

COMMAND	**ID**
WRITE_REG	0x0001
READ_REG	0x0003
WRITE_WISHBONE	0x0005
READ_WISHBONE	0x0007
WRITE_I2C	0x0009
READ_I2C	0x000B
SDRAM_RECONFIGURE	0x000D
READ_FLASH_WORDS	0x000F
PROGRAM_FLASH_WORDS	0x0011
ERASE_FLASH_BLOCK	0x0013
READ_SPI_PAGE	0x0015
PROGRAM_SPI_PAGE	0x0017
ERASE_SPI_SECTOR	0x0019
ONE_WIRE_READ_ROM_CMD	0x001B
ONE_WIRE_DS2433_WRITE_MEM	0x001D
ONE_WIRE_DS2433_READ_MEM	0x001F
DEBUG_CONFIGURE_ETHERNET	0x0021
DEBUG_ADD_ARP_CACHE_ENTRY	0x0023
GET_EMBEDDED_SOFTWARE_VERS	0x0025
PMBUS_READ_I2C	0x0027
SDRAM_PROGRAM	0x0029
CONFIGURE_MULTICAST	0x002B
DEBUG_LOOPBACK_TEST	0x002D
QSFP_RESET_AND_PROG	0x002F

The UDP payload of each packet starts with a header. The header consists of the command ID as described above. It also consists of a sequence number. The sequence number is a 16-bit value that increments by one for each packet sent. The host checks that the sequence number in the response matches the sequence number in the request.

The header of each packet 'sCommandHeader' is defined as:

TYPE	**PARAMETER**	**DESCRIPTION**
Uint16_t	uCommandType	Command identifier. Response command identifier must be this incremented by one.
Uint16_t	uSequenceNumber	Sequence number. Response sequence number must be the same as in the request.

After the header, the remainder of the UDP payload contain fields specific to the command. The following sections will detail each of these command packets in detail.

3.1. WRITE_REG

The WRITE_REG command is used to write a register in either the board control or DSP register address space.

Request 'sWriteRegReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0001
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBoardReg	1 = writing to board control and status registers 2 = writing to DSP registers
Uint16_t	uRegAddress	Address of register to write to
Uint16_t	uRegDataHigh	Upper 16 bits of write data
Uint16_t	uRegDataLow	Lower 16 bits of write data

Response 'sWriteRegResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0002
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBoardReg	1 = writing to board control and status registers 2 = writing to DSP registers
Uint16_t	uRegAddress	Address of register to write to
Uint16_t	uRegDataHigh	Upper 16 bits of write data
Uint16_t	uRegDataLow	Lower 16 bits of write data
Uint16_t	uPadding[6]	Padding to minimum packet size and 64-bit boundary

3.2. READ_REG

The READ_REG command is used to read a register from either the board control or DSP register address space.

Request 'sReadRegReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0003
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBoardReg	1 = reading from board control and status registers 2 = reading from DSP registers
Uint16_t	uRegAddress	Address of register to read

Response 'sReadRegResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0004
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBoardReg	1 = reading from board control and status registers 2 = reading from DSP registers
Uint16_t	uRegAddress	Address of register to read
Uint16_t	uRegDataHigh	Upper 16 bits of read data
Uint16_t	uRegDataLow	Lower 16 bits of read data
Uint16_t	uPadding[6]	Padding to minimum packet size and 64-bit boundary

3.3. WRITE_WISHBONE

The WRITE_WISHBONE command is used to perform a low level Wishbone write to a Wishbone slave. All accesses to Wishbone slaves are via memory-mapped reads and writes and this command gives low-level direct access to the Wishbone bus.

Request 'sWriteWishboneReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0005
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of Wishbone slave address to write to
Uint16_t	uAddressLow	Lower 16 bits of Wishbone slave address to write to
Uint16_t	uWriteDataHigh	Upper 16 bits of data to write to Wishbone slave
Uint16_t	uWriteDataLow	Lower 16 bits of data to write to Wishbone slave

Response 'sWriteWishboneResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0006
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of Wishbone slave address to write to
Uint16_t	uAddressLow	Lower 16 bits of Wishbone slave address to write to
Uint16_t	uWriteDataHigh	Upper 16 bits of data to write to Wishbone slave
Uint16_t	uWriteDataLow	Lower 16 bits of data to write to Wishbone slave
Uint16_t	uPadding[6]	Padding to minimum packet size and 64-bit boundary

3.4. READ_WISHBONE

The READ_WISHBONE command is used to perform a low level Wishbone read from a Wishbone slave. All accesses to Wishbone slaves are via memory-mapped reads and writes and this command gives low-level direct access to the Wishbone bus.

Request 'sReadWishboneReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0007
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of Wishbone slave address to read from
Uint16_t	uAddressLow	Lower 16 bits of Wishbone slave address to read from

Response 'sReadWishboneResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0008
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of Wishbone slave address to read from
Uint16_t	uAddressLow	Lower 16 bits of Wishbone slave address to read from
Uint16_t	uReadDataHigh	Upper 16 bits of data read from Wishbone slave
Uint16_t	uReadDataLow	Lower 16 bits of data read from Wishbone slave
Uint16_t	uPadding[6]	Padding to minimum packet size and 64-bit boundary

3.5. WRITE_I2C

The WRITE_I2C command is used to perform an I2C write on a selected I2C interface. Up to 32 bytes can be written in a single I2C transaction.

Request 'sWriteI2CReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0009
Uint16_t	uSequenceNumber	Sequence number of request

Uint16_t	uId	Identifier of I2C interface want to write to 0 = SKARAB motherboard I2C 1 = Mezzanine 0 I2C 2 = Mezzanine 1 I2C 3 = Mezzanine 2 I2C 4 = Mezzanine 3 I2C
Uint16_t	uSlaveAddress	I2C slave address of device want to write to
Uint16_t	uNumBytes	Number of bytes to write
Uint16_t	uWriteBytes[32]	Bytes to write

Response 'sWriteI2CResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000A
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of I2C interface want to write to 0 = SKARAB motherboard I2C 1 = Mezzanine 0 I2C 2 = Mezzanine 1 I2C 3 = Mezzanine 2 I2C 4 = Mezzanine 3 I2C
Uint16_t	uSlaveAddress	I2C slave address of device want to write to
Uint16_t	uNumBytes	Number of bytes to write
Uint16_t	uWriteBytes[32]	Bytes to write
Uint16_t	uWriteSuccess	0 = I2C write failed 1= I2C write successful
Uint16_t	uPadding[2]	Padding to 64-bit boundary

3.6. READ_I2C

The READ_I2C command is used to perform an I2C read on a selected I2C interface. Up to 32 bytes can be read in a single I2C transaction.

Request 'sReadI2CReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000B
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of I2C interface want to read from 0 = SKARAB motherboard I2C 1 = Mezzanine 0 I2C 2 = Mezzanine 1 I2C 3 = Mezzanine 2 I2C 4 = Mezzanine 3 I2C
Uint16_t	uSlaveAddress	I2C slave address of device want to read from
Uint16_t	uNumBytes	Number of bytes to read

Response 'sReadI2CResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000C
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of I2C interface want to read from 0 = SKARAB motherboard I2C 1 = Mezzanine 0 I2C 2 = Mezzanine 1 I2C 3 = Mezzanine 2 I2C 4 = Mezzanine 3 I2C
Uint16_t	uSlaveAddress	I2C slave address of device want to read from

Uint16_t	uNumBytes	Number of bytes to read
Uint16_t	uReadBytes[32]	Bytes read
Uint16_t	uReadSuccess	0 = I2C read failed 1= I2C read successful
Uint16_t	uPadding[2]	Padding to 64-bit boundary

3.7. SDRAM_RECONFIGURE

The SDRAM_RECONFIGURE command is used to perform various tasks relating to the programming of the boot SDRAM and the reconfiguration of the Virtex 7 FPGA from the boot SDRAM. It controls and checks the status of the flash SDRAM interface discussed earlier.

Request 'sSdramReconfigureReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000D
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uOutputMode	Specifies the output mode of the flash SDRAM interface 0x0 = Flash mode 0x1 = SDRAM program mode 0x2 = SDRAM read mode
Uint16_t	uClearSdram	1 = Clear the SDRAM so that it does not contain any FPGA image
Uint16_t	uFinishedWritingToSdram	1 = Finished writing FPGA image to SDRAM
Uint16_t	uAboutToBootFromSdram	1 = About to boot from image written to SDRAM
Uint16_t	uDoReboot	1 = Trigger a reboot of the Virtex 7 FPGA from the image stored in SDRAM
Uint16_t	uResetSdramReadAddress	1 = Reset the SDRAM read address so that can start reading out contents of SDRAM at address 0x0
Uint16_t	uClearEthernetStatistics	1 = Clear Ethernet packet statistics relating to packets used to program the FPGA image into the SDRAM
Uint16_t	uEnableDebugSdramReadMode	1 = Enable debug mode where can read the currently stored data in the SDRAM (this is used for board level ATP testing)
Uint16_t	uDoSdramAsyncRead	1 = Used in debug mode to read the contents of the SDRAM, this reads a single 32-bit word from SDRAM and advances the SDRAM read pointer by one (this is used for board level ATP testing)
Uint16_t	uDoContinuityTest	1 = Enable a debug mode where the continuity of the flash bus between the Virtex 7 FPGA and the Spartan 3AN FPGA is checked (this is used for board level ATP testing). The address and sideband control signals are driven from the Virtex 7 FPGA (based on uContinuityTestOutputLow and uContinuityTestOutputHigh) and the Spartan 3AN FPGA echoes the received address and control signals back to the Virtex 7 FPGA over the data lines (which is then read in the response as uContinuityTestReadLow and uContinuityTestReadHigh).
Uint16_t	uContinuityTestOutputLow	Used in the continuity debug mode, this specifies the value to set the lower 16 bits of the bus
Uint16_t	uContinuityTestOutputHigh	Used in the continuity debug mode, this specifies the value to set the upper 16 bits of the bus

Response 'sSdramReconfigureResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000E
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uOutputMode	Specifies the output mode of the flash SDRAM interface 0x0 = Flash mode 0x1 = SDRAM program mode 0x2 = SDRAM read mode
Uint16_t	uClearSdram	1 = Clear the SDRAM so that it does not contain any FPGA image

Uint16_t	uFinishedWritingToSdram	1 = Finished writing FPGA image to SDRAM
Uint16_t	uAboutToBootFromSdram	1 = About to boot from image written to SDRAM
Uint16_t	uDoReboot	1 = Trigger a reboot of the Virtex 7 FPGA from the image stored in SDRAM
Uint16_t	uResetSdramReadAddress	1 = Reset the SDRAM read address so that can start reading out contents of SDRAM at address 0x0
Uint16_t	uClearEthernetStatistics	1 = Clear Ethernet packet statistics relating to packets used to program the FPGA image into the SDRAM
Uint16_t	uEnableDebugSdramReadMode	1 = Enable debug mode where can read the currently stored data in the SDRAM (this is used for board level ATP testing)
Uint16_t	uDoSdramAsyncRead	1 = Used in debug mode to read the contents of the SDRAM, this reads a single 32-bit word from SDRAM and advances the SDRAM read pointer by one (this is used for board level ATP testing)
Uint16_t	uNumEthernetFrames	Number of Ethernet packets received on the 1GBE fabric application layer and programmed into the SDRAM
Uint16_t	uNumEthernetBadFrames	Number of Ethernet packets received on the 1GBE fabric application layer that were reported as bad frames
Uint16_t	uNumEthernetOverloadFrames	Number of Ethernet packets received on the 1GBE fabric application layer that were reported as overload frames
Uint16_t	uSdramAsyncReadDataHigh	Upper 16 bits of 32-bit word read from the SDRAM (used for board level ATP testing)
Uint16_t	uSdramAsyncReadDataLow	Lower 16 bits of 32-bit word read from the SDRAM (used for board level ATP testing)
Uint16_t	uDoContinuityTest	1 = Enable a debug mode where the continuity of the flash bus between the Virtex 7 FPGA and the Spartan 3AN FPGA is checked (this is used for board level ATP testing). The address and sideband control signals are driven from the Virtex 7 FPGA (based on uContinuityTestOutputLow and uContinuityTestOutputHigh) and the Spartan 3AN FPGA echoes the received address and control signals back to the Virtex 7 FPGA over the data lines (which is then read in the response as uContinuityTestReadLow and uContinuityTestReadHigh).
Uint16_t	uContinuityTestReadLow	Lower 16 bits of echoed flash signals
Uint16_t	uContinuityTestReadHigh	Lower 16 bits of echoed flash signals
Uint16_t	uPadding[1]	Padding to 64-bit boundary

3.8. READ_FLASH_WORDS

The READ_FLASH_WORDS command is used to read a block of up to 384 words (each word is 16 bits) from the NOR flash on the SKARAB motherboard.

Request 'sReadFlashWordsReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x000F
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address in the NOR flash that want to read
Uint16_t	uAddressLow	Lower 16 bits of address in the NOR flash that want to read
Uint16_t	uNumWords	Number of 16-bit words to read

Response 'sReadFlashWordsResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0010
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address in the NOR flash that want to read
Uint16_t	uAddressLow	Lower 16 bits of address in the NOR flash that want to read
Uint16_t	uNumWords	Number of 16-bit words to read
Uint16_t	uReadWords[384]	Read words
Uint16_t	uPadding[3]	Padding to 64-bit boundary

3.9. PROGRAM_FLASH_WORDS

The PROGRAM_FLASH_WORDS command is used to program a block to the NOR flash on the SKARAB motherboard. The NOR flash requires that data is programmed in blocks of 512 words (1KB) at a time. This is to make use of the Object Mode Programming mode. In Object Mode Programming, the full 1KB region is available for programming. Since the CPU can only handle standard Ethernet frames sizes, the programming data must be sent over two successive packets. The fields in the packet identify whether it is the first portion or the second portion. Further, buffered programming must be used when using Object Mode Programming. If only the first half of a 1KB region needs to be used (for example, to store configuration data), then Control Mode Programming can be used. In this mode, single words can be written and the data doesn't have to be programmed as a complete block. PROGRAM_FLASH_WORDS supports both modes.

Request 'sProgramFlashWordsReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0011
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address in flash to start programming to
Uint16_t	uAddressLow	Lower 16 bits of address in flash to start programming to
Uint16_t	uTotalNumWords	Total number of 16-bits words to program over one or more Ethernet packets
Uint16_t	uNumWords	Number of words in this Ethernet packet to program
Uint16_t	uDoBufferedProgramming	1 = Perform buffered programming
Uint16_t	uStartProgram	1 = First packet in flash programming, start programming operation in flash
Uint16_t	uFinishProgram	1 = Last packet in flash programming, complete programming operation in flash
Uint16_t	uWriteWords[256]	Words to program

Response 'sProgramFlashWordsResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0012
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address in flash to start programming to
Uint16_t	uAddressLow	Lower 16 bits of address in flash to start programming to
Uint16_t	uTotalNumWords	Total number of 16-bits words to program over one or more Ethernet packets
Uint16_t	uNumWords	Number of words in this Ethernet packet to program
Uint16_t	uDoBufferedProgramming	1 = Perform buffered programming
Uint16_t	uStartProgram	1 = First packet in flash programming, start programming operation in flash
Uint16_t	uFinishProgram	1 = Last packet in flash programming, complete programming operation in flash
Uint16_t	uProgramSuccess	0 = Programming failed 1 = Programming succeeded
Uint16_t	uPadding[2]	Padding to 64-bit boundary

3.10. ERASE_FLASH_BLOCK

The ERASE_FLASH_BLOCK command is used to erase a block in the NOR flash on the SKARAB motherboard.

Request 'sEraseFlashBlockReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0013
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBlockAddressHigh	Upper 16 bits of block address to erase

Uint16_t	uBlockAddressLow	Lower 16 bits of block address to erase

Response 'sEraseFlashBlockResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0014
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uBlockAddressHigh	Upper 16 bits of block address to erase
Uint16_t	uBlockAddressLow	Lower 16 bits of block address to erase
Uint16_t	uEraseSuccess	1 = Erase successful 0 = Erase unsuccessful
Uint16_t	uPadding[7]	Padding to minimum packet size and 64-bit boundary

3.11. READ_SPI_PAGE

The READ_SPI_PAGE command is used to read a page from the SPI flash in the Spartan 3AN FPGA on the SKARAB motherboard. Up to a full page (264 bytes) can be read.

Request 'sReadSpiPageReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0015
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address of page want to read
Uint16_t	uAddressLow	Lower 16 bits of address of page want to read
Uint16_t	uNumBytes	Number of bytes in page want to read

Response 'sReadSpiPageResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0016
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address of page want to read
Uint16_t	uAddressLow	Lower 16 bits of address of page want to read
Uint16_t	uNumBytes	Number of bytes in page want to read
Uint16_t	uReadBytes[264]	Read bytes
Uint16_t	uReadSpiPageSuccess	1 = SPI page read successful 0 = SPI page read unsuccessful
Uint16_t	uPadding[2]	Padding to 64-bit boundary

3.12. PROGRAM_SPI_PAGE

The PROGRAM_SPI_PAGE command is used to program a page to the SPI flash in the Spartan 3AN FPGA on the SKARAB motherboard. Up to a full page (264 bytes) can be programmed. After the page has been programmed by the microcontroller, the microcontroller reads the contents of the same page and includes the read values in the response packet. This is used as verification that the programming completed successfully.

Request 'sProgramSpiPageReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0017
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address of page want to program
Uint16_t	uAddressLow	Lower 16 bits of address of page want to program
Uint16_t	uNumBytes	Number of bytes in page want to program
Uint16_t	uWriteBytes[264]	Data to program

Response 'sProgramSpiPageResp':

TYPE	PARAMETER	DESCRIPTION

Uint16_t	uCommandType	0x0018
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uAddressHigh	Upper 16 bits of address of page want to program
Uint16_t	uAddressLow	Lower 16 bits of address of page want to program
Uint16_t	uNumBytes	Number of bytes in page want to program
Uint16_t	uVerifyBytes[264]	Verification bytes read from same page after programming completes
Uint16_t	uProgramSpiPageSuccess	1 = SPI page programming successful 0 = SPI page programming unsuccessful
Uint16_t	uPadding[2]	Padding to 64-bit boundary

3.13. ERASE_SPI_SECTOR

The ERASE_SPI_SECTOR command is used to erase a sector in the SPI flash in the Spartan 3AN FPGA on the SKARAB motherboard.

Request 'sEraseSpiSectorReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0019
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uSectorAddressHigh	Upper 16 bits of sector address want to erase in SPI flash
Uint16_t	uSectorAddressLow	Lower 16 bits of sector address want to erase in SPI flash

Response 'sEraseSpiSectorResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001A
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uSectorAddressHigh	Upper 16 bits of sector address want to erase in SPI flash
Uint16_t	uSectorAddressLow	Lower 16 bits of sector address want to erase in SPI flash
Uint16_t	uEraseSuccess	1 = SPI sector erasing successful 0 = SPI sector erasing unsuccessful
Uint16_t	uPadding[7]	Padding to minimum packet size and 64-bit boundary

3.14. ONE_WIRE_READ_ROM_CMD

The ONE_WIRE_READ_ROM_CMD command is used to read the 64-bit ROM address of a DS24B33 EEPROM on the specified 1-wire interface.

Request 'sOneWireReadROMReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001B
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface

Response 'sOneWireReadROMResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001C
Uint16_t	uSequenceNumber	Sequence number of request

Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface
Uint16_t	uRom[8]	Read 64-bit ROM address
Uint16_t	uReadSuccess	1 = 1-wire read ROM transaction successful 0 = 1-wire read ROM transaction unsuccessful
Uint16_t	uPadding[4]	Padding to 64-bit boundary

3.15. ONE_WIRE_DS2433_WRITE_MEM

The ONE_WIRE_DS2433_WRITE_MEM command is used to program up to 32 bytes of data to a DS24B33 EEPROM on the specified 1-wire interface.

Request 'sOneWireDS2433WriteMemReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001D
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uDeviceRom[8]	64-bit ROM address of device want to write to
Uint16_t	uSkipRomAddress	1 = Skip ROM addressing stage of 1-wire transaction
Uint16_t	uWriteBytes[32]	Bytes to program
Uint16_t	uNumBytes	Number of bytes to program
Uint16_t	uTA1	Target address 1 (lower byte of target address)
Uint16_t	uTA2	Target address 2 (upper byte of target address)
Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface

Response 'sOneWireDS2433WriteMemResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001E
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uDeviceRom[8]	64-bit ROM address of device want to program
Uint16_t	uSkipRomAddress	1 = Skip ROM addressing stage of 1-wire transaction
Uint16_t	uWriteBytes[32]	Bytes to program
Uint16_t	uNumBytes	Number of bytes to program
Uint16_t	uTA1	Target address 1 (lower byte of target address)
Uint16_t	uTA2	Target address 2 (upper byte of target address)
Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface
Uint16_t	uWriteSuccess	1 = 1-wire program transaction successful 0 = 1-wire program transaction unsuccessful
Uint16_t	uPadding[4]	Padding to 64-bit boundary

3.16. ONE_WIRE_DS2433_READ_MEM

The ONE_WRITE_DS2433_READ_MEM command is used to read up to 32 bytes from a DS24B33 EEPROM on the specified 1-wire interface.

Request 'sOneWireDS2433ReadMemReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x001F
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uDeviceRom[8]	64-bit ROM address of device want to read
Uint16_t	uSkipRomAddress	1 = Skip ROM addressing stage of 1-wire transaction
Uint16_t	uNumBytes	Number of bytes to read
Uint16_t	uTA1	Target address 1 (lower byte of target address)
Uint16_t	uTA2	Target address 2 (upper byte of target address)
Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface

Response 'sOneWireDS2433ReadMemResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0020
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uDeviceRom[8]	64-bit ROM address of device want to read
Uint16_t	uSkipRomAddress	1 = Skip ROM addressing stage of 1-wire transaction
Uint16_t	uReadBytes[32]	Read bytes from 1-wire device
Uint16_t	uNumBytes	Number of bytes to read
Uint16_t	uTA1	Target address 1 (lower byte of target address)
Uint16_t	uTA2	Target address 2 (upper byte of target address)
Uint16_t	uOneWirePort	1-wire interface to access 0 = SKARAB motherboard 1-wire interface 1 = Mezzanine 0 1-wire interface 2 = Mezzanine 1 1-wire interface 3 = Mezzanine 2 1-wire interface 4 = Mezzanine 3 1-wire interface
Uint16_t	uReadSuccess	1 = 1-wire read transaction successful 0 = 1-wire read transaction unsuccessful
Uint16_t	uPadding[4]	Padding to 64-bit boundary

3.17. DEBUG_CONFIGURE_ETHERNET

The DEBUG_CONFIGURE_ETHERNET command is used to program various parameters of a 40GBE MAC (from the host connected to the 1GBE interface). This is purely for debugging purposes and is not needed in normal operation. In normal operation, the MAC address is derived automatically from the serial number of the SKARAB motherboard and the number of the 40GBE interface. The fabric port address is fixed at 0x7148. The address of the gateway in the ARP cache is configured through DHCP. The fabric IP address is obtained through DHCP. The multicast IP address and mask can be set through the CONFIGURE_MULTICAST command. The fabric interface is enabled automatically once DHCP completes.

Request 'sDebugConfigureEthernetReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0021
Uint16_t	uSequenceNumber	Sequence number of request

Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uFabricMacHigh	Bits 47..32 of fabric source MAC address
Uint16_t	uFabricMacMid	Bits 31..16 of fabric source MAC address
Uint16_t	uFabricMacLow	Bits 15..0 of fabric source MAC address
Uint16_t	uFabricPortAddress	Fabric source port address
Uint16_t	uGatewayIPAddressHigh	Upper 16 bits of the gateway IP address
Uint16_t	uGatewayIPAddressLow	Lower 16 bits of the gateway IP address
Uint16_t	uFabricIPAddressHigh	Upper 16 bits of fabric IP address
Uint16_t	uFabricIPAddressLow	Lower 16 bits of fabric IP address
Uint16_t	uFabricMultiCastIPAddressHigh	Upper 16 bits of multicast IP address
Uint16_t	uFabricMultiCastIPAddressLow	Lower 16 bits of multicast IP address
Uint16_t	uFabricMultiCastIPAddressMaskHigh	Upper 16 bits of multicast IP address mask
Uint16_t	uFabricMultiCastIPAddressMaskLow	Lower 16 bits of multicast IP address mask
Uint16_t	uEnableFabricInterface	1 = Enable fabric interface

Response 'sDebugConfigureEthernetResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0022
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uFabricMacHigh	Bits 47..32 of fabric source MAC address
Uint16_t	uFabricMacMid	Bits 31..16 of fabric source MAC address
Uint16_t	uFabricMacLow	Bits 15..0 of fabric source MAC address
Uint16_t	uFabricPortAddress	Fabric source port address
Uint16_t	uGatewayIPAddressHigh	Upper 16 bits of the gateway IP address
Uint16_t	uGatewayIPAddressLow	Lower 16 bits of the gateway IP address
Uint16_t	uFabricIPAddressHigh	Upper 16 bits of fabric IP address
Uint16_t	uFabricIPAddressLow	Lower 16 bits of fabric IP address
Uint16_t	uFabricMultiCastIPAddressHigh	Upper 16 bits of multicast IP address
Uint16_t	uFabricMultiCastIPAddressLow	Lower 16 bits of multicast IP address
Uint16_t	uFabricMultiCastIPAddressMaskHigh	Upper 16 bits of multicast IP address mask
Uint16_t	uFabricMultiCastIPAddressMaskLow	Lower 16 bits of multicast IP address mask
Uint16_t	uEnableFabricInterface	1 = Enable fabric interface
Uint16_t	uPadding[1]	Padding to 64-bit boundary

3.18. DEBUG_ADD_ARP_CACHE_ENTRY

The DEBUG_ADD_ARP_CACHE_ENTRY command is used to add an entry in the ARP cache of a selected 40GBE MAC (from the host through 1GBE). This is purely for debugging purposes and is not needed in normal operation. In normal operation, a sequence of ARP requests are generated for the full subnet that the Ethernet interface is situated on (255.255.255.0). These ARP requests are generated continuously once DHCP completes. An ARP request is generated for a specific IP address every 100 milliseconds so as not to flood the network. The ARP requests are generated for interface 0 first, then 1 and so on. Once all the ARP requests have been generated for interface 4, the loop starts back at interface 0. The responses to the ARP requests are what are used to populate the ARP cache.

Request 'sDebugAddARPCacheEntryReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0023
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uIPAddressLower8Bits	ARP cache has 256 entries, the lowest 8 bits of the destination IP address is used to index into ARP cache
Uint16_t	uMacHigh	Bits 47..32 of MAC address to associate with destination IP address
Uint16_t	uMacMid	Bits 31..16 of MAC address to associate with destination IP address
Uint16_t	uMacLow	Bits 15..0 of MAC address to associate with destination IP address

Response 'sDebugAddARPCacheEntryResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0024
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uIPAddressLower8Bits	ARP cache has 256 entries, the lowest 8 bits of the destination IP address is used to index into ARP cache
Uint16_t	uMacHigh	Bits 47..32 of MAC address to associate with destination IP address
Uint16_t	uMacMid	Bits 31..16 of MAC address to associate with destination IP address
Uint16_t	uMacLow	Bits 15..0 of MAC address to associate with destination IP address
Uint16_t	uPadding[5]	Padding to 64-bit boundary

3.19. GET_EMBEDDED_SOFTWARE_VERS

The GET_EMBEDDED_SOFTWARE_VERS command is used to get the version of the microcontroller embedded software.

Request 'sGetEmbeddedSoftwareVersionReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0025
Uint16_t	uSequenceNumber	Sequence number of request

Response 'sGetEmbeddedSoftwareVersionResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0026
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uVersionMajor	Major version of microcontroller embedded software
Uint16_t	uVersionMinor	Minor version of microcontroller embedded software
Uint16_t	uQSFPBootloaderVersionMajor	Major version of the QSFP+ Mezzanine STM bootloader software version
Uint16_t	uQSFPBootloaderVersionMinor	Minor version of the QSFP+ Mezzanine STM bootloader software version
Uint16_t	uPadding[6]	Padding to minimum packet size and 64-bit boundary

3.20. PMBUS_READ_I2C

The PMBUS_READ_I2C command is used to perform a PMBus I2C read. A PMBus I2C read requires that first a PMBus command is written as an I2C transaction. A repeated start must then be generated and then an I2C read completes the transaction. Since this cannot be performed as a separate write followed by read I2C transaction (because of the requirement for a repeated start), a PMBus I2C read command is implemented. PMBus I2C writes can be performed using the WRITE_I2C command. Up to 32 bytes can be read in a single transaction.

Request 'sPMBusReadI2CBytesReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0025
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	I2C bus that want to perform PMBus I2C read of 0 = SKARAB motherboard I2C bus 1 = Mezzanine 0 I2C bus 2 = Mezzanine 1 I2C bus 3 = Mezzanine 2 I2C bus 4 = Mezzanine 3 I2C bus
Uint16_t	uSlaveAddress	Slave address of PMBus device that want to read
Uint16_t	uCommandCode	PMBus command for the I2C read
Uint16_t	uReadBytes[32]	Not used
Uint16_t	uNumBytes	Number of bytes to read

Response 'sPMBusReadI2CBytesResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0026
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	I2C bus that want to perform PMBus I2C read of 0 = SKARAB motherboard I2C bus 1 = Mezzanine 0 I2C bus 2 = Mezzanine 1 I2C bus 3 = Mezzanine 2 I2C bus 4 = Mezzanine 3 I2C bus
Uint16_t	uSlaveAddress	Slave address of PMBus device that want to read
Uint16_t	uCommandCode	PMBus command for the I2C read
Uint16_t	uReadBytes[32]	Read bytes
Uint16_t	uNumBytes	Number of bytes to read
Uint16_t	uReadSuccess	1 = PMBus I2C read successful 0 = PMBus I2C read unsuccessful
Uint16_t	uPadding[1]	Padding to 64-bit boundary

3.21. SDRAM_PROGRAM

The SDRAM_PROGRAM command is used to program a block of 4096 words to the boot SDRAM. These words would typically contain a portion of the FPGA image that you wish to boot from. The SDRAM_PROGRAM is different to the other commands in that it is directed to the 1GBE fabric interface (so the fabric UDP port is used and not the control port). It is not handled by the microcontroller. Also, no response packet is generated.

Request 'sSdramProgramReq:

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0029
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uFirstPacket	1 = First packet in the sequence of packets to program an FPGA image into the boot SDRAM
Uint16_t	uLastPacket	1 = Last packet in the sequence of packets to program an FPGA image into the boot SDRAM

Uint16_t	uWriteWords[4096]	4096 16-bit words to program into the boot SDRAM (the remainder packet at the end of the FPGA image must always be padded to a 4096 word boundary with 0xFFFF)

3.22. CONFIGURE_MULTICAST

The CONFIGURE_MULTICAST command is used to configure the multicast IP address and multicast IP address mask of a 40GBE MAC. Once these values have been configured by the host over the 1GBE, the microcontroller starts sending IGMP membership messages to the router for all IP addresses in the group. This happens every 60 seconds. If a command is received to reboot the Virtex 7 FPGA from the boot SDRAM, or shut down the SKARAB, then the microcontroller sends IGMP leave messages to the router for all IP addresses in the group.

Request 'sConfigureMulticastReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x002B
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uFabricMultiCastIPAddressHigh	Upper 16 bits of fabric multicast IP address group that joining
Uint16_t	uFabricMultiCastIPAddressLow	Lower 16 bits of fabric multicast IP address group that joining
Uint16_t	uFabricMultiCastIPAddressMaskHigh	Upper 16 bits of fabric multicast IP address mask that defines range of IP addresses in group
Uint16_t	uFabricMultiCastIPAddressMaskLow	Lower 16 bits of fabric multicast IP address mask that defines range of IP addresses in group

Response 'sConfigureMulticastResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x002C
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uFabricMultiCastIPAddressHigh	Upper 16 bits of fabric multicast IP address group that joining
Uint16_t	uFabricMultiCastIPAddressLow	Lower 16 bits of fabric multicast IP address group that joining
Uint16_t	uFabricMultiCastIPAddressMaskHigh	Upper 16 bits of fabric multicast IP address mask that defines range of IP addresses in group
Uint16_t	uFabricMultiCastIPAddressMaskLow	Lower 16 bits of fabric multicast IP address mask that defines range of IP addresses in group
Uint16_t	uPadding[5]	Padding to 64-bit boundary

3.23. DEBUG_LOOPBACK_TEST

The DEBUG_LOOPBACK_TEST command is used to test a 40GBE interface that is configured in loopback. The host sends the desired packet 'payload' over the 1GBE to the microcontroller. The microcontroller constructs a UDP/IP packet and attaches the packet payload provided by the host. The packet is then sent to the 40GBE interface specified. Since the 40GBE interface being tested is in loopback, the packet sent should be received. The microcontroller waits for the packet to be received from the 40GBE interface. There is a timeout to prevent the microcontroller waiting forever if something is wrong with the 40GBE interface. The payload of the received packet is extracted by the microcontroller and sent back to the host over the 1GBE. The host can then check that the packet payload traversed the 40GBE interface successfully and that the received packet payload matches the packet payload transmitted.

Request 'sDebugLoopbackTestReq':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x002D
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uTestData[256]	Test packet payload

Response 'sDebugLoopbackTestResp':

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x002E
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uId	Identifier of Ethernet interface that want to configure 0 = 1GBE MAC 1 = 40GBE 0 MAC 2 = 40GBE 1 MAC 3 = 40GBE 2 MAC 4 = 40GBE 3 MAC
Uint16_t	uTestData[256]	Packet payload of the received packet
Uint16_t	uValid	1 = Received a loopback packet (uTestData is valid) 0 = Didn't receive a loopback packet and timed out waiting (uTestData is invalid)
Uint16_t	uPadding[4]	Padding to 64-bit boundary

3.24. QSFP_RESET_AND_PROG

The QSFP_RESET_AND_PROG command is used to put the QSFP+ Mezzanine STM in bootloader mode (and bootloader programming mode) in order to allow in-system reprogramming of the application software. The programming process is started by sending the QSFP_RESET_AND_PROG command with 'uReset' = '1' and 'uProgram' = '1'. Once programming is complete, the QSFP_RESET_AND_PROG command is sent again, this time with 'uReset' = '0' and 'uProgram' = '0'. The STM returns to bootloader mode, exits bootloader mode and starts executing the new application software.

Request 'sQSFPResetAndProgramReq:

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x002F
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uReset	'1' = Reset the QSFP+ Mezzanine so that it is back in bootloader mode
Uint16_t	uProgram	'1' = When in bootloader mode, enter bootloader programming mode (if were in bootloader programming mode, set to '0' to return to bootloader mode)

Response 'sQSFPResetAndProgramResp:

TYPE	PARAMETER	DESCRIPTION
Uint16_t	uCommandType	0x0030
Uint16_t	uSequenceNumber	Sequence number of request
Uint16_t	uReset	'1' = Reset the QSFP+ Mezzanine so that it is back in bootloader mode

Uint16_t	uProgram	‘1’ = When in bootloader mode, enter bootloader programming mode (if were in bootloader programming mode, set to ‘0’ to return to bootloader mode)

4. Embedded Software Description

This section describes the embedded software for the Microblaze microcontroller. It is intended as an overview of the embedded software. All embedded software functions include a comment header. For function-level documentation, refer to the embedded software source code.

The following source files are included in the SDK project:

- 'constant_defs.h': This file contains the definitions of all major constants and enumerated types. All major global variables are included here. This file also includes a number of defines:
    - NUM_ETHERNET_INTERFACES: Specifies the total number of Ethernet interfaces. The Ethernet interfaces start from the 1GBE interface and then include the 40GBE interfaces. Currently this is set to five (one 1GBE interface and four 40GBE interfaces)
    - DEBUG_PRINT: Specifies whether to print out debug information to the console.
    - DO_40GBE_LOOPBACK_TEST: Specifies whether the 40GBE interfaces are to be tested in loopback. If this is the case, then the microcontroller configures these interfaces manually because DHCP will not complete if the interface is in loopback.
    - DO_1GBE_LOOPBACK_TEST: Specifies whether the 1GBE interface is to be tested in loopback. If this is the case, then the microcontroller configures this interface manually because DHCP will not complete if the interface is in loopback.
    - EMBEDDED_SOFTWARE_VERSION_MAJOR, EMBEDDED_SOFTWARE_VERSION_MINOR: Version information for the embedded software
    - SKARAB_BSP: specifies whether the embedded software is specifically for the stripped down SKARAB BSP package available to third party users
- 'delay.c', 'delay.h': A delay mechanism has been implemented to allow specifying a delay in microseconds.
- 'eth_mac.h', 'eth_mac.c': These files include the functions to configure the 1GBE and 40GBE Ethernet MACs. Functions are also included for sending and receiving CPU packets to and from the Ethernet MACs.
- 'eth_sorter.h', 'eth_sorter.c': These files perform the handling of the CPU packets read from the Ethernet MACs. For each different packet type, there are various packet header checks that are performed. These checks are performed on the IP header (if exists), the UDP header (if exists), the ICMP header (if exists), the ARP packet requests (if exists) and the DHCP packet header (if exists). These checks include calculating and verifying the various header and packet checksums. Once a packet has been verified as being valid, appropriate actions are performed and then the response packets are constructed. The behaviour for the different packet types is as follows:
    - ARP request: if the ARP request is directed at the Ethernet MAC that the packet was received on (matches the IP address of the Ethernet MAC), then an ARP response packet is constructed and transmitted on that Ethernet MAC.
    - ARP response: as ARP responses are received by an Ethernet MAC, the entries are added to the ARP cache of the corresponding Ethernet MAC. If the IP address from the sender in the ARP response matches the IP address of the Ethernet MAC, then an IP address conflict has occurred and DHCP is restarted on that Ethernet interface.
    - ICMP request (PING): if an ICMP request is received by an Ethernet MAC and it matches the IP address of the Ethernet MAC, then an ICMP response packet is constructed and transmitted on that Ethernet MAC.
    - UDP packet: if a UDP packet is received by an Ethernet MAC and it matches the IP address of the Ethernet MAC and it matches the control port address, then a command request has been received from the host. The command identifier is used to identify what command the host is requesting. The appropriate functionality is performed by the microcontroller and a response packet is constructed and transmitted on the Ethernet MAC back to the host.
    - DHCP: a separate DHCP state machine is implemented for each Ethernet interface.
        - The default DHCP state is DHCP_STATE_IDLE.
        - Once an Ethernet interface comes up, the microcontroller constructs a DHCP Discover packet and enters the DHCP_STATE_DISCOVER state. The purpose of the DHCP Discover packet is to determine the IP address of any DHCP servers located on the network. DHCP Discover packets are sent every 10 seconds while in this state.
        - Once a DHCP server responds with a DHCP Offer packet, the DHCP state machine enters the DHCP_STATE_REQUEST state. The microcontroller constructs a DHCP Request packet to now request the IP address offered by the DHCP server. If the DHCP server does not respond to the request within 10 seconds then the DHCP state machine returns to the DHCP_STATE_DISCOVER state and the process if restarted.
        - If the DHCP server responds with a DHCP Ack packet, the DHCP state machine enters the

DHCP_STATE_COMPLETE state. DHCP is now complete for the corresponding Ethernet interface and the received IP address is allocated to the Ethernet MAC. The microcontroller enables sending of ARP requests on this Ethernet MAC so as to populate the ARP cache of the Ethernet MAC and also to confirm that an IP address conflict has not occurred. The fabric interface of the corresponding Ethernet MAC is also enabled. If the DHCP server responds with a DHCP Nack packet, the DHCP state machine returns to the DHCP_STATE_DISCOVER state and the process is restarted.

  - IGMP: a separate IGMP state machine is implemented for each Ethernet interface.
    - The default IGMP state is IGMP_STATE_NOT_JOINED.
    - When the multicast IP address and multicast IP address mask are set by the host, the IGMP state is changed to IGMP_STATE_JOINED_GROUP.
    - In the IGMP_STATE_JOINED_GROUP state, IGMP Membership packets are constructed and transmitted by the microcontroller for each IP address in the IGMP group (defined by the mask). The IGMP Membership packets are sent every 60 seconds. The IGMP Membership packets inform the switch that the SKARAB wants to receive multicast packets for the group on the selected Ethernet interface.
    - If a remote shutdown or reboot of the SKARAB is requested, then the IGMP state is changed to IGMP_STATE_LEAVING. In this state, IGMP Leave report packets are constructed and transmitted by the microcontroller for each IP address in the IGMP group. The IGMP Leave report packets inform the switch that the SKARAB no longer wants to receive multicast packets for the group on the selected Ethernet interface.
    - Once all the IGMP Leave report packets have been sent, the IGMP state returns to IGMP_STATE_NOT_JOINED.
- 'flash_sdram_controller.h', 'flash_sdram_controller.c': These files provide the driver for the Wishbone Flash SDRAM Interface, specifically for the NOR flash, boot SDRAM programming, boot SDRAM reading and sideband signalling for booting from SDRAM. High level flash programming, reading and erasing functions are provided for the NOR flash on the SKARAB motherboard.
- 'i2c_master.h', 'i2c_master.c': These files provide the driver for the Wishbone I2C interfaces. The driver provided with the OpenCores Wishbone I2C component is modified. High level I2C read, write and PMBus read functions are provided for controlling the I2C buses on the SKARAB motherboard and mezzanine sites. Further high level functions are also provided which are unique to the Hitech Global development platform. These functions are not applicable to the SKARAB motherboard.
- 'icape_controller.h', 'icape_controller.c': These files provide the driver for the ICAPE component in the Wishbone Flash SDRAM Interface. A high level function is provided for triggering an in-system reconfiguration of the Virtex 7 FPGA.
- 'isp_spi_controller.h', 'isp_spi_controller.c': These files provide the driver for the SPI component in the Wishbone Flash SDRAM Interface. High level SPI programming, reading and erasing functions are provided for the SPI flash in the Spartan 3AN FPGA.
- 'main.c': This is the top level file for the embedded software. Functionality includes:
  - Timer interrupt handler: a timer interrupt is configured to occur every 100 milliseconds. This creates time intervals for the sending of DHCP and IGMP packets. The timer also controls the reading and updating of the QSFP+ Mezzanine status.
  - Ethernet packet sorter: a CPU packet read from an Ethernet interface is sorted based on the packet and the appropriate action taken.
  - Arp request handler: generates ARP requests at a defined interval.
  - Ethernet link status: monitors the Ethernet link status. If a link has come up then start the DHCP process on that link. If a link has gone down then reset the DHCP state for that link.
  - Update the QSFP+ Mezzanine status: determine if QSFP+ modules are plugged into the QSFP+ Mezzanine. Update the transmit and receive LEDs of the QSFP+ Mezzanine based on whether the corresponding 40GBE MAC is transmitting and/or receiving packets. This is complicated by the fact that the ST microcontroller on the QSFP+ Mezzanine can only process one I2C transaction at a time. Back to back I2C transactions cannot be handled. Subsequently, the reading of module present and updating LEDs is performed over multiple I2C transactions separated by the 100 millisecond timer interface. A state machine keeps track of where the microcontroller is in the update process so that only a single transaction is performed at each time though the main operating loop (so as not to stall the microcontroller while the QSFP+ update process is happening). This function also handles instructing the QSFP+ to exit the bootloader mode and start executing the application software.
  - Initialising the Ethernet parameters: the serial number of the SKARAB motherboard is read from the 1-wire EEPROM and used to construct a unique source MAC address for each Ethernet interface.

- Initialising the QSFP+ Mezzanine parameters: the microcontroller checks all mezzanine sites that have a mezzanine plugged. The 1-wire EEPROM on the mezzanine is read to determine whether it is a QSFP+ Mezzanine or not. The location of the first QSFP+ Mezzanine is stored.
- Initialising the interrupt controller and timer: the interrupt controller and timer are initialised. The timer is configured for a 100 millisecond interval (assuming a system clock frequency of 156.25MHz).
- Read and print out the FPGA DNA: read the Virtex7 DNA from the register interface and print out to the serial port.
- The main operating loop of the microcontroller will be discussed shortly.

- 'one_wire.h', 'one_wire.c': These files provide the driver for the Wishbone 1-Wire interfaces. The OpenCores Wishbone 1-wire driver is modified. High level functions include reading and writing the DS24B33 1-wire EEPROM.
- 'register.h', 'register.c': These files provide the driver for the Wishbone Register interfaces. High level functions include reading and writing the board level and DSP registers.

The main operating loop of the microcontroller is as follows:

- Initialise the I2C interfaces
- Initialise the watchdog timer
- Inform the Spartan 3AN FPGA that finished booting
- Initialise the interrupt controller and timer
- Initialise the QSFP+ Mezzanine
- Initialise the Ethernet interface parameters
- Read and print the Virtex7 FPGA DNA
- Enter a main loop
    - Update the QSFP+ Mezzanine status
    - For each Ethernet interface (1GBE and 40GBE interfaces)
        - Update the Ethernet link up or down status
        - Check if a packet is waiting in the receive buffer of the Ethernet interface
            - If so, read the packet, handle it and transmit a response if necessary
        - Send ARP requests if necessary
        - Send DHCP packets if necessary
        - Send IGMP packets if necessary
    - Check if a reboot is requested
        - If so, wait until have left all IGMP groups and then trigger a reboot
    - Pat the watchdog

5. Host Library Code and Test Software Description

This section describes the host library code for the SKARAB. It is intended as a brief overview of the host library code and test software offered. All host library code functions are commented. For function-level documentation, refer to the host library source code.

5.1. Test Software

5.1.1. Roach3MotherboardProductionTest

Console application for board level ATP and CQATP testing of the SKARAB motherboard and QSFP+ Mezzanine.

5.1.2. Roach3SpartanFlashReconfig

Command line application for reprogramming the SPI flash in the Spartan 3AN. There is no back up firmware image for the Spartan 3AN and so this application should be used with care.

To use this application:

Roach3SpartanFlashReconfig -s -i <roach3_ip_address> -h <host_ip_address> filename.ufp

- filename.ufp is the file name of the Spartan 3AN firmware image that you want to program
- -s specifies a silent install option (the user is not prompted at any time during the execution)
- -i specifies the IP address of the SKARAB
- -h specifies the host IP address (this is not needed in normal operation)

For typical use:

Roach3SpartanFlashReconfig -i <roach3_ip_address> filename.ufp

5.1.3. Roach3VirtexFlashReconfig

Command line application for reprogramming the NOR flash that boots the Virtex 7 FPGA on start up. There is a back up firmware (golden) image in the NOR flash and so this application can be used to perform in-field upgrades of the SKARAB.

To use this application:

Roach3VirtexFlashReconfig -s -e -a <starting_address> -i <roach3_ip_address> -h <host_ip_address> filename.hex

- filename.hex is the Virtex 7 FPGA firmware image that you want to program
- -s specifies the silent install option (the user is not prompted at any time during the execution)
- -e specifies the erase only option (this is not needed in normal operation)
- -a specifies the starting address in flash (this is only needed if different to the default boot address, not needed in normal operation)
- -i specifies the IP address of the SKARAB
- -h specifies the host IP address (this is not needed in normal operation)

For typical use:

Roach3VirtexFlashReconfig –i <roach3_ip_address> filename.hex

5.1.4. Roach3SdramReconfig

Command line application for programming the SDRAM boot flash that is used to perform a reconfiguration of the Virtex 7 FPGA once the SKARAB is running. To speed up the programming of a boot image to SDRAM, the hex file can be pre-processed into a raw bin file. This application provides the functionality for converting a hex file into a pre-processed bin file. Lastly, this application provides the functionality to program the boot SDRAM with a pre-processed bin file.

Roach3SdramReconfig -s -v -r -p -i <roach3_ip_address> -h <host_ip_address> -o <pre_processed.bin> filename.hex

- filename.hex is the files that you want to program the boot SDRAM with
- -s specifies the silent install option
- -v specified the verify option which reads back the contents of the SDRAM to verify that it was programmed correctly (this is purely for debugging purposes as read back from the SDRAM is slow)
- -r specifies whether the SKARAB should reboot automatically once the SDRAM programming completes
- -i specifies the IP address of the SKARAB
- -h specifies the host IP address (this is not needed in normal operation)
- -o specifies the output file name to store the generated pre-processed bin file
- -p specifies that a pre-processed file is being used

To program the boot SDRAM with a standard hex file and reboot:

Roach3SdramReconfig –r -i <roach3_ip_address> filename.hex

To generate a pre-processed bin file based on a standard hex file (the SDRAM is not programmed, only the bin file is created):

Roach3SdramReconfig –o <pre_processed.bin> filename.hex

To program the boot SDRAM with a pre-processed bin file and reboot:

Roach3SdramReconfig –r –p -i <roach3_ip_address> pre_processed.bin

5.2. Host Library Code

The host library code is implemented in Microsoft Visual C++ 2013 Express. The solution file name is 'Roach3Motherboard.sln´and the project is 'Roach3Motherboard.vcxproj'. It is included as a static library in all the test software described above.

The host library code uses the 'ws2_32.lib' library to create and access sockets. The FTDI libraries (ftd2xx.lib, FTCI2C.lib, FTCJTAG.lib) are included in test software that communicates with the FTDI USB PHY over USB.

The host library code consists of the following files:

- 'base.h', 'base.cpp': provides error message exception handling.
- 'CriticalSection.h', 'CriticalSection.cpp': provides critical section protection for multithreaded code.
- 'Socket.h', 'Socket.cpp': provides high level functions for reading and writing to an Ethernet socket. This is used for sending and receiving control packets to and from the microcontroller.
- 'Roach3MotherboardDefs.h': all of the host library code constants and enumerated types are defined here.
- 'Roach3Motherboard.h', 'Roach3Motherboard.cpp': provides all of the functions to access a SKARAB. Two sockets are created when a 'cRoach3Motherboard' object is created: one for the control port and one for the fabric port. The 'cRoach3Motherboard' provides all of the commands defined in the host API. These low level command functions can be used by test software to exercise all functionality offered by the SKARAB motherboard. In addition, higher level functions are built using these lower level command implementations. High level functions simplify access to the various I2C devices on the SKARAB motherboard. High level functions also simplify access to the board level registers.