GYRO2 TESTER FPGA

Design Specification

Revision 3.0

By Charles Dickinson

Prepared for

Contents

List of Figures:

List of Tables

Description

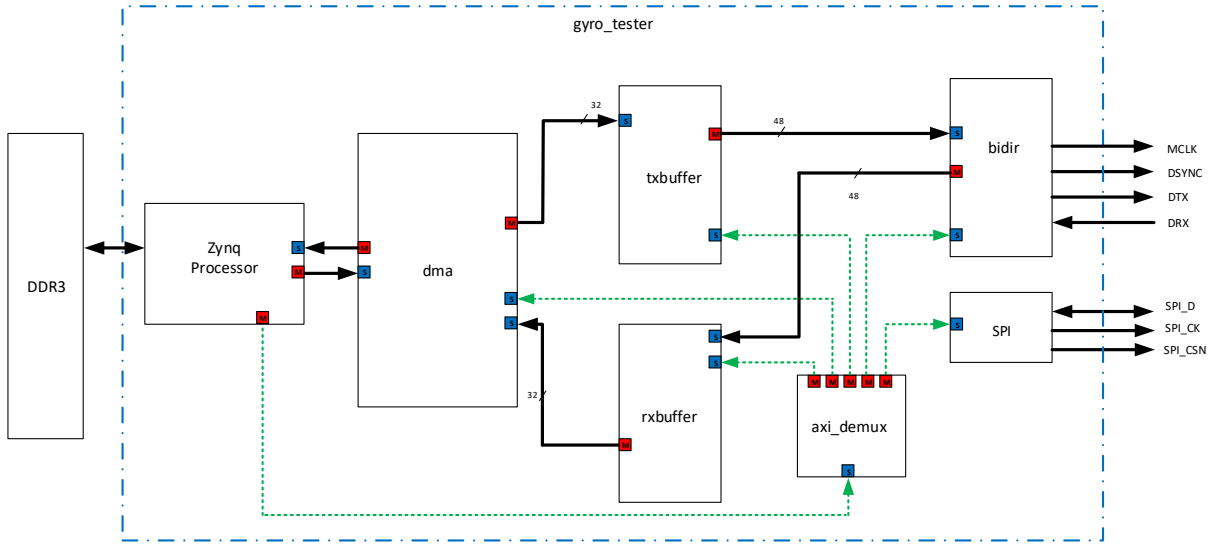
The Gyro tester FPGA device is a validation platform specifically designed to test the features of the Gyro2 ASIC by CSS. The device should be able to source and capture data on the high speed serial interface of the Gyro2 ASIC. Both sourced and captured data should be kept in the on board DDR3 of the Gyro tester system. It should also interface the custom SPI like programming interface of the Gyro2 ASIC.

Features

- Provides a 48K by 16bit word transmit buffer for sourcing data into the Gyro2 ASIC
- Provides a 48K by 16bit word capture buffer for receiving back data from the Gyro2 ASIC
- Provides a simple Register to SPI interface for both sourcing and capturing SPI communications to the Gyro2 ASIC.
- Uses the standard Xilinx AXI-DMA IP in direct memory access mode.
- Ability to turn off the outputs to avoid damaging the device under test during power up.
- Uses the Xilinx Zynq Processor for all communications and DMA access.
- All internal data interfaces are AXI-S (AMBA Streaming interface)
- All internal register access is controlled via the Zynq Processor over AXI4 interfaces
- Implements various loopbacks for testing
- Implements various internal test pattern for testing
- Sample C code for various types of tests is provided.

Block Diagram

Figure 1 Gyro Tester Block Diagram

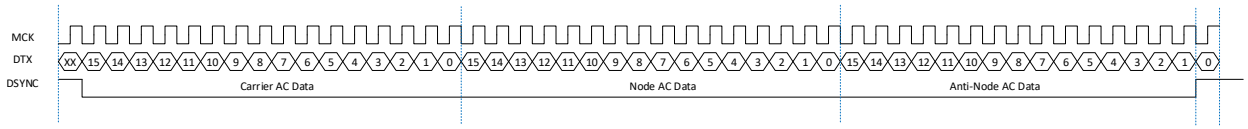


Transmit Data Path

The Transmit data path is responsible for fetching data from the DDR to the TX fifo buffer. Once the buffer is filled with 48K 16bit words it is then sent to the serializer found within the Bidir block. Serial data is sent out on the divided down MCK output clock.

Serial TX data should look like the timing diagram below:

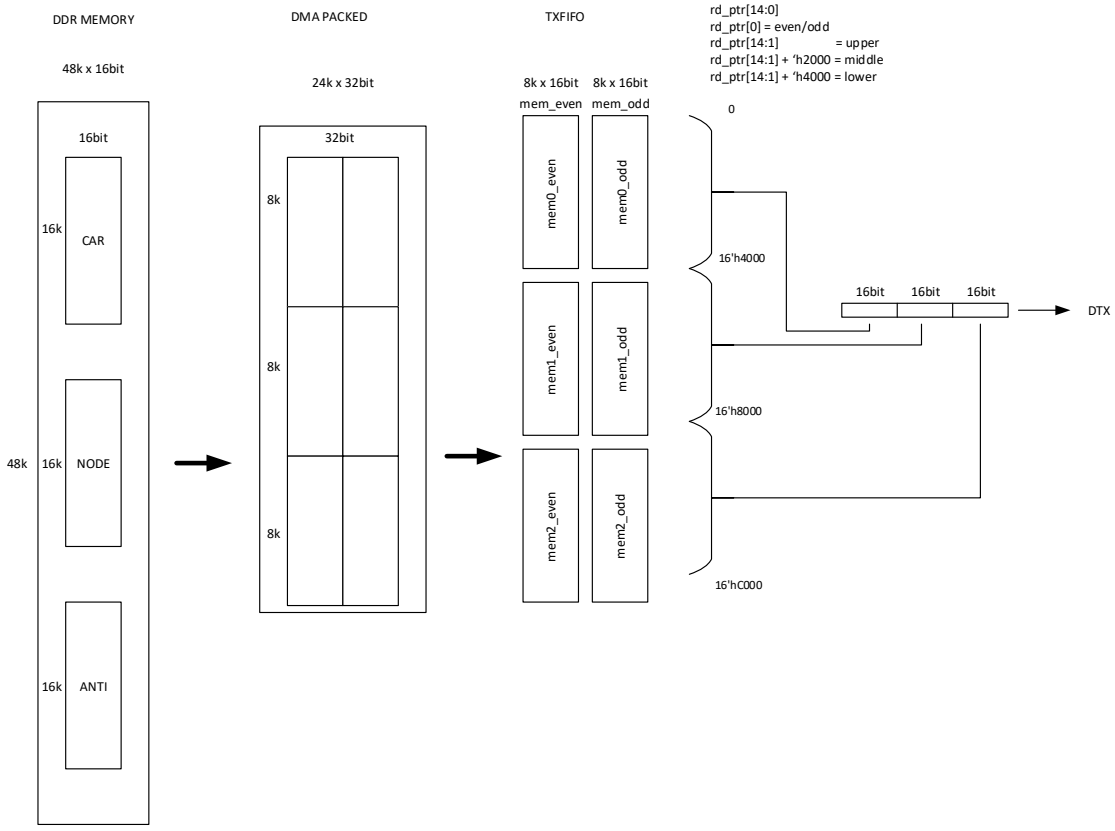
Figure 2 Serial TX data timing diagram



For the actual implementation internally, the data must go through a few transforms to work with the different IP and as a matter of convenience for loop back testing. The data is stored on the DDR as 16bit words, then the data is DMA packed at 32bit words. Also, the required output serial data is 48bit (3x16bit) words. For this reason, the TX fifo buffer will output 48bit words. The following diagram illustrates the required transforms on the data.

It should be noted that the memory requirement for the TX fifo buffer will be 2 instances of 24Kx16bit memories. The Xilinx compiler will most likely use its own block ram types to achieve this.

Figure 3 TX Data Transformations

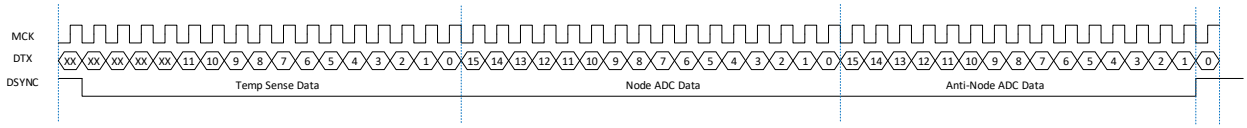


Receive Data Path

The receive data path is responsible for capturing the incoming high-speed serial data. Then storing it internally in the RX fifo buffer. Once the buffer indicated it is ready to the DMA, it will pack and transmit the data to the DDR in the form of 48K 16bit data. Serial data is captured out on the divided down MCK output clock.

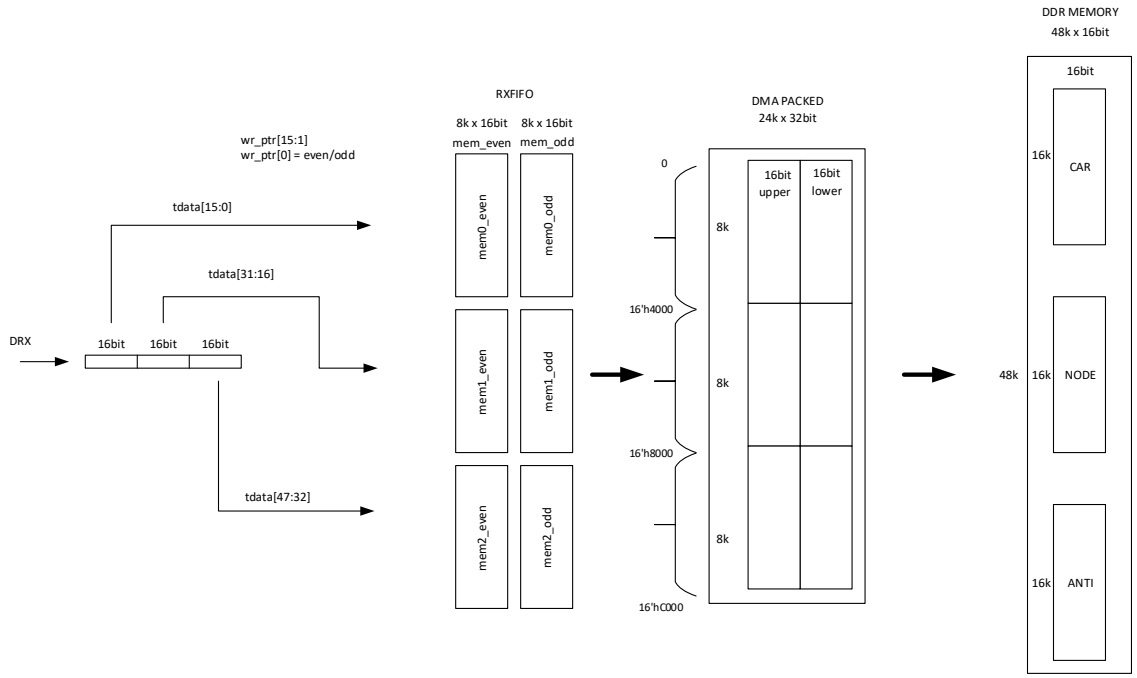
Serial RX data should look like the timing diagram below

Figure 4 Serial RX data timing diagram



Much like the transmit side the receiver also must do a few transforms on the data to facilitate both the delivery across a packed 32bit DMA interface as well as implement testing loopbacks from the 48bit transmit path. The data flow will look like the following diagram:

Figure 5 RX Data Transformations



From this we can see that the actual memory requirements of the RX path are 8 instances of 8K x 16bit memories. It is assumed that Xilinx compiler will choose its own block memory devices for these elements. Although it would not matter if it decided to use a sea of flops either.

RXBYP Feature

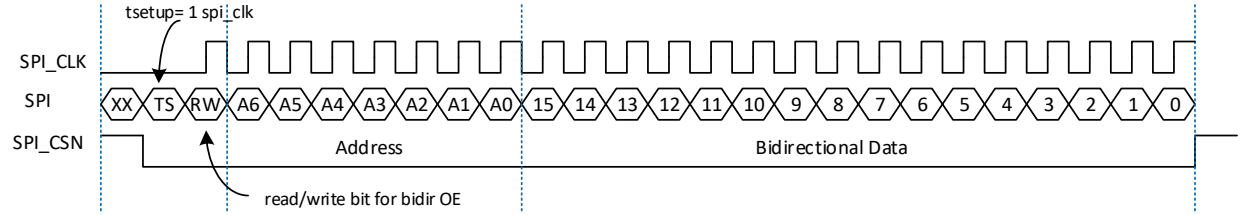
For a more flexible way to collect different size data sets a RX Bypass was added to the design. This will bypass the RX Buffer memory and allow direct streaming of any size data set to the DDR. Once the bypass is set via the registers you must also set the max_byte_len (see the rx fifo register 2). This value is the expect packet size expressed in bytes. Just Note that the DMA has a similar setting as well. To avoid errors the rx length must be equal to or less than the length set in the DMA. IF you go over the DMA will go into a bad state.

Also provided with the C code is a standalone script to deinterleave the RX bypass data. As without the use of the RX memory the data comes out interleaved. This script expects the data to be in a file called gyro_data.txt. See the testcase_src directory for an example.

SPI Register Interface

The SPI interface is a custom implementation of the SPI protocol. It should follow the below timing diagram:

Figure 6 SPI serial register interface

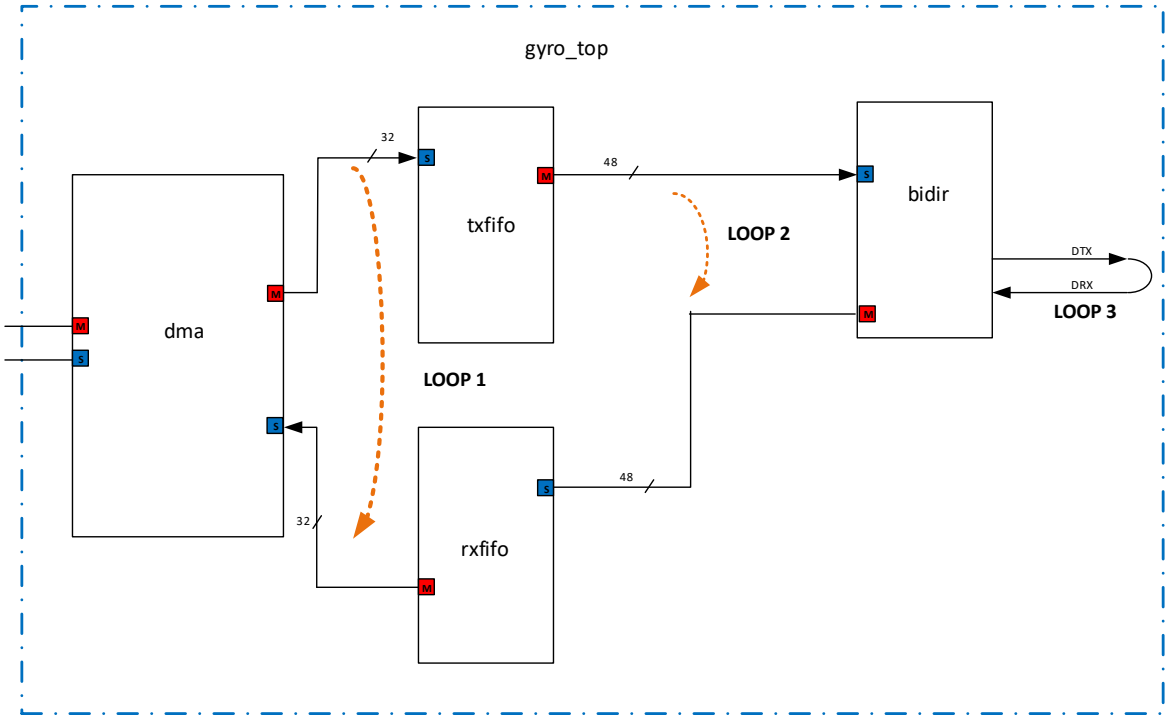


Data is directly written via the register interface from the Zynq processor over AXI Lite. See the register description later in this document for more info.

Loopbacks

The Gyro Tester design has been equipped with 3 functional loopback modes for help in testing the FPGA and the board before the ASIC is delivered. These implemented via internal AXI Switches provided by Xilinx IP. They can be turned on and off with their register interfaces accessed by AXI Lite with the Zynq processor. See the register description and sample code for more information on how to turn on and off the loopbacks.

Figure 7 Loopbacks for testing



Test Patterns

Test patterns are internally generated counting patterns to prefill the tx rx buffers or prefill the Shift Registers. This if for testing and debug. For the Transmit side if the test pattern is active the incoming tx data from the DMA in ignored and the tx buffer fifo is filled with a simple counting pattern. For the receive side the captured serial stream on the DRX is ignored and the rx buffer fifo is filled with a similar fixed counting pattern. Similarly, for the shift registers the rx serial stream is ignored and patterns are injected at either the rx or tx. For more information on how to turn these on see the register descriptions for the TX, RX buffers and Bidir shift register interface.

Vivado Functional Testbench

The rtl was fully tested inside vivado using their generated functional models and system Verilog api function calls. These functions which interact with the Zynq processor BFM can be found in the document ds941-zynq-ultra-ps-e-vip.pdf found within the “docs” section of the project. The goal is too fully cover most of the features of the design with directed tests. Then each of these system Verilog directed tests should have a corresponding C-code directed test for validation on the actual zedboard unit. The provided xsim within the Vivado tool is free and works well enough to accomplish full functional testing without any extra licenses required.

Table 1 Functional Test Cases

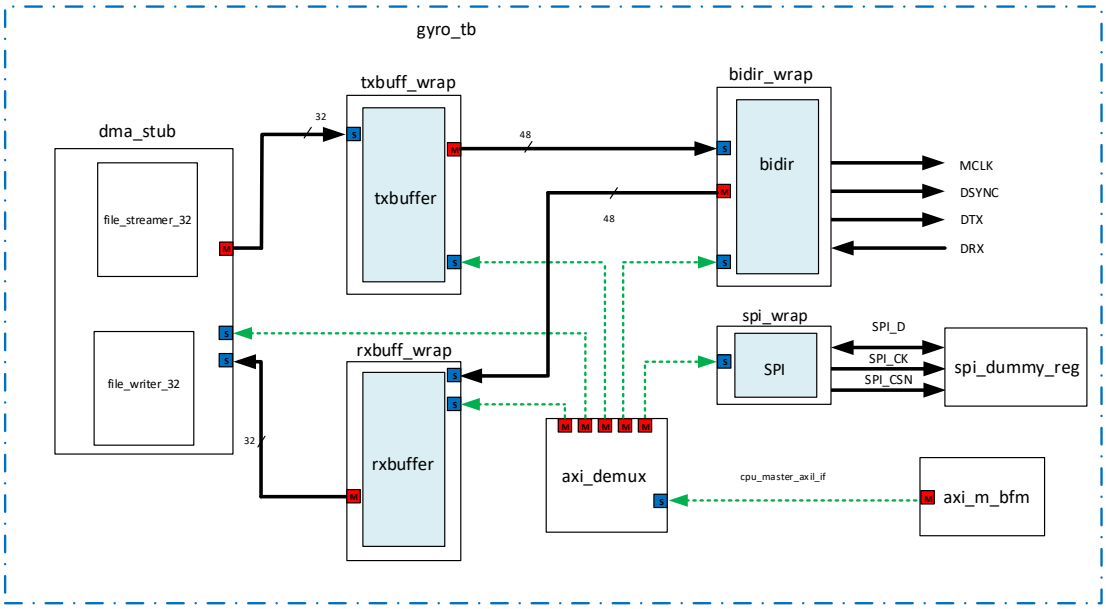
System Verilog test	Corresponding C code test	Description

test_case_loop1.sv	test_case_loop1.c	Fills the DDR with a pattern uses DMA TX/RX to send it through Loop1
test_case_loop2.sv	test_case_loop2.c	Uses both the TX ad RX Buffer memories to loop DMA data over the internal 48bit interface.
test_case_loop2_txpattern.sv	test_case_loop2_txpattern.c	Counting pattern is generated inside the TX Buffer and looped internally over the 48bit interface to RX buffer and then to the RX DMA path.
test_case_rx_pattern.sv	test_case_rx_pattern.c	A counting pattern is generated inside the RX Buffer and sent out the RX DMA path.
test_case_loop3_txpattern.sv	test_case_loop3_txpattern.c	A counting pattern is generated inside the TX Buffer and sent to the serial output, looped over to the Full RX path including the RX Buffer memory.
test_case_loop3_srtx_pattern.sv	test_case_loop3_srtx_pattern.c	A reverse counting pattern is generated inside the TX shift register and looped across the serial interface to the Full RX path including the RX Buffer memory.
test_case_srrx_pattern.sv	test_case_srrx_pattern.c	A counting pattern is generated inside the RX shift register and sent down the full RX path including the RX buffer memory.
test_case_loop3.sv	test_case_loop3.c	A full serial loop back of the entire system. Uses both Buffer memories.
test_case_spi.sv	test_case_spi.c	A simple test of write and reads of external SPI registers
test_case_loop2_rxbyp.sv	test_case_loop2_rxbyp.c	DMA data is again looped over across the 48bit internal interface; however, this time the RX buffer memory is bypassed.
test_case_loop3_rxbyp.sv	test_case_loop3_rxbyp.c	DMA data is sent through the full external serial loop of the system; however, this time the RX buffer memory is bypassed.

Standalone Functional Testbench

A full functional framework was created so that the main features of the gyro2 tester fpga could be simulated in standard Verilog. The testbench does not using any of the Xilinx IP so more flexibility, less complexity when developing new features or testing using different simulation tools. For this first revision NCSIM tools by cadence were used, although there is no reason it would not function well with either VCS or modelsim as well.

Figure 8 Standalone testbench block diagram



The main original rtl code to the gyro2 design is shown above in blue. This code needs to be fully covered in a functional testbench platform. Wrappers and stubs were written to ease integration using system Verilog standard interfaces. This will emulate how vivado visually hooks the blocks up in its tool.

For the DMA emulation a simple file reader and writer were written for streaming 32bit hex data from a file. The receiver is writing the data out to a file in the same format. This allows us to compare the files and ensure data continuity in all the loopback modes.

For the Zynq processor we only require a flexible AXI lite functional model. This will handle all the AXI write commands written into the registers. It will also be able to perform AXI lite reads to any of the internal registers.

An AXI demux block was required to communicate between the one master and multiple slaves. This is shown as the axi_demux.

A simple 4 register external SPI block was written for emulating the SPI interface on the Gyro2 design and ensuring the design would both properly write and readback registers.

Register Map

Table 2 Register Map

Address	Name	Access Type	Default	Description
0x43C0_0000	AXI SW0 Control	R/W	0000_0000	General Control
0x43C0_0040	AXI SW0 M0 Addr	R/W	8000_0000	M1 Selector value
0x43C1_0000	SPI Control 0	R/W	0000_0000	
0x43C1_0004	SPI Address	R/W	0000_0000	
0x43C1_0008	SPI Data	R/W	0000_0000	
0x43C1_000C	SPI Read Data	R		
0x43C2_0000	BiDir Control 0	R/W	0000_0000	
0x43C2_0004	BiDir Control 1	R/W	0000_0000	
0x43C2_0008	BiDir Control 2	R/W	0000_0000	
0x43C2_000C	BiDir Status	R		
0x43C3_0000	RX FIFO Control 0	R/W	0000_0000	
0x43C3_0004	RX FIFO Control 1	R/W	0000_0000	
0x43C3_0008	RX FIFO Control 2	R/W	0000_0000	
0x43C3_000C	RX FIFO Status	R		
0x43C4_0000	TX FIFO Control 0	R/W	0000_0000	
0x43C4_0004	TX FIFO Control 1	R/W	0000_0000	
0x43C4_0008	TX FIFO Control 2	R/W	0000_0000	
0x43C4_000C	TX FIFO Status	R		
0x43C5_0000	AXI SW1 Control	R/W	0000_0000	General Control
0x43C5_0040	AXI SW1 M0 Addr	R/W	8000_0000	M0 Selector value
0x43C5_0044	AXI SW1 M1 Addr	R/W	8000_0000	M1 Selector value
0x43C6_0000	AXI SW2 Control	R/W	0000_0000	General Control
0x43C6_0040	AXI SW2 M0 Addr	R/W	8000_0000	M0 Selector value
0x43C6_0044	AXI SW2 M1 Addr	R/W	8000_0000	M1 Selector value
0x43C7_0000	AXI SW3 Control	R/W	0000_0000	General Control
0x43C7_0040	AXI SW3 M0 Addr	R/W	8000_0000	M0 Selector value
0x43C7_0044	AXI SW3 M1 Addr	R/W	8000_0000	M1 Selector value
0x4040_0000	MM2S_DMACR	R/W	0000_0000	MM2S DMA Control
0x4040_0004	MM2S_DMASR	R	0000_0000	MM2S DMA Status
0x4040_0018	MM2S_SA	R/W	0000_0000	MM2S Source Address. Lower 32 bits of address.

0x4040_001C	MM2S_SA_MSB	R/W	0000_0000	MM2S Source Address. Upper 32 bits of address.
0x4040_0028	MM2S_LENGTH	R/W	0000_0000	MM2S Transfer Length (Bytes)
0x4040_0030	S2MM_DMACR	R/W	0000_0000	S2MM DMA Control
0x4040_0034	S2MM_DMASR	R	0000_0000	S2MM DMA Status
0x4040_0048	S2MM_DA	R/W	0000_0000	S2MM Destination Address. Lower 32 bit address.
0x4040_004C	S2MM_DA_MSB	R/W	0000_0000	S2MM Destination Address. Upper 32 bit address.
0x4040_0058	S2MM_LENGTH	R/W	0000_0000	S2MM Buffer Length Bytes

Table 3 AXI SW0 Control Register (0x43C0_0000)

Name	Bits	Description
Reserved	0	Reserved
REG_UPDATE	1	Register Update. MUX registers are double buffered. Writing '1' updates the registers and issues a soft reset to the core (for approximately 16 cycles.)
Reserved	31:2	Reserved

Table 4 AXI SW0 M0 Addr (0x43C0_0040)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 5 SPI Control 0 (0x43C1_0000)

Name	Bits	Description
SPI Update	0	SPI update. auto clearing pulse starts transaction.
SPI Direction	1	SPI direction 0=write ; 1=read
SPI clock Div	3:2	Div16,Div8,Div,4,Div2
SPI Output en	4	1 = enable the spi outputs 0 = disable all spi outputs
Reserved	30:5	Reserved
SPI Busy	31	SPI is currently shifting data. Should be polled for new transactions

Table 6 SPI Address (0x43C1_0004)

Name	Bits	Description
SPI Address	6:0	SPI supports a 7 bit address field
Reserved	31:7	Reserved

Table 7 SPI Data (0x43C1_0008)

Name	Bits	Description
SPI Data	15:0	SPI supports a 16 bit write data
Reserved	31:16	Reserved

Table 8 SPI Read Data (0x43C1_000C)

Name	Bits	Description
SPI Read Data	15:0	SPI read back data
Reserved	31:16	Reserved

Table 9 BiDir Control 0 (0x43C2_0000)

Name	Bits	Description
Reserved	15:0	Reserved
Clock Div	19:16	values 0-7 change the output and txclk frequency 50Mhz – 12.5Mhz
Reserved	23:20	Reserved
Mode	25:24	00 Normal; 01 Loopback DRX=DTX
Reserved	28:26	Reserved
RX_testpattern	29	setting to 1 will inject test pattern at the input shift register
TX_testpattern	30	setting to 1 will inject test pattern at the output shift register
Soft Reset	31	1 will hold outputs into reset mode.

Table 10 BiDir Control 1 (0x43C2_0004)

Name	Bits	Description
Start Output	0	Enables the output serial stream
Start Input	4	Enables the input serial stream

Table 11 BiDir Control 2 (0x43C2_0008)

Name	Bits	Description
Enable	0	General purpose enable turns entire block on/off
Reserved	31:1	Reserved

Table 12 BiDir Control 3 (0x43C2_000C)

Name	Bits	Description
Reserved	31:0	Reserved

Table 13 RX FIFO Control 0 (0x43C3_0000)

Name	Bits	Description
Fifo Push En	0	Enables the Rx Fifo Push (filling of fifo).
Fifo Pop En	1	Enables the Draining of the RX fifo

Reserved	3:2	Reserved
rx_byp_en	4	Bypasses the Memory of the RX Buffer direct reading to DMA
Reserved	11:5	Reserved
rxfull_irq_en	12	Interrupt enable for the RX full
Reserved	15:13	Reserved
Test Pattern	16	Test fill pattern into the RX Buffer. Disables incoming data
Reserved	31:17	Reserved

Table 14 RX FIFO Control 1 (0x43C3_0004)

Name	Bits	Description
clear	0	Write 1 to clear bit. Soft reset for the RXFIFO pointers.
Reserved	31:1	Reserved

Table 15 RX FIFO Control 2 (0x43C3_0008)

Name	Bits	Description
rx_max_len	31:0	For use with the RX Buffer Bypass option. Need to know the transfer length in bytes for the DMA last flag.

Table 16 RX FIFO Status (0x43C3_000C)

Name	Bits	Description
write pointer	15:0	value of the current fifo write pointer
Reserved	29:16	Reserved
RX Fifo Empty	30	The fifo is empty
RX Fifo Full	31	The fifo is full condition.

Table 17 TX FIFO Control 0 (0x43C4_0000)

Name	Bits	Description
Tx Fifo Enable	0	Tx Enable
Reserved	15:1	Reserved
Test Pattern	16	Test fill pattern into the TX Buffer. Disables incoming data
Reserved	31:1	Reserved

Table 18 TX FIFO Control 1 (0x43C4_0004)

Name	Bits	Description
clear	0	Write 1 to clear bit. Soft reset for the TXFIFO pointers.
Reserved	31:0	Reserved

Table 19 TX FIFO Status RevID (0x43C4_0008)

Name	Bits	Description
Chip Rev	31:0	Revision ID of the FPGA RTL code

Table 20 TX FIFO Status (0x43C4_000C)

Name	Bits	Description
write pointer	15:0	value of the current fifo write pointer
Reserved	29:16	Reserved
TX Fifo Empty	30	The fifo is empty
TX Fifo Full	31	The fifo is full condition.

Table 21 AXI SW1 Control Register (0x43C5_0000)

Name	Bits	Description
Reserved	0	Reserved
REG_UPDATE	1	Register Update. MUX registers are double buffered. Writing '1' updates the registers and issues a soft reset to the core (for approximately 16 cycles.)
Reserved	31:2	Reserved

Table 22 AXI SW1 M0 Addr (0x43C5_0040)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 23 AXI SW1 M1 Addr (0x43C5_0044)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 24 AXI SW2 Control Register (0x43C6_0000)

Name	Bits	Description
Reserved	0	Reserved
REG_UPDATE	1	Register Update. MUX registers are double buffered. Writing '1' updates the registers and issues a soft reset to the core (for approximately 16 cycles.)
Reserved	31:2	Reserved

Table 25 AXI SW2 M0 Addr (0x43C6_0040)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 26 AXI SW2 M1 Addr (0x43C6_0044)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 27 AXI SW3 Control Register (0x43C7_0000)

Name	Bits	Description
Reserved	0	Reserved
REG_UPDATE	1	Register Update. MUX registers are double buffered. Writing '1' updates the registers and issues a soft reset to the core (for approximately 16 cycles.)
Reserved	31:2	Reserved

Table 28 AXI SW3 M0 Addr (0x43C7_0040)

Name	Bits	Description
Mix_MUX	3:0	Mux Value
MIx_DISABLE	31	Set to 1 to explicitly disable

Table 29 DMA Registers (0x4040_0000) - (0x4040_0058)

**See the DMA 7.1 specification page 13-37 for the register descriptions. Included In the documents folder as Xilinx_axi_dma_rev71.pdf

APPENDIX A - Direct Register Mode (Simple DMA)

Direct Register Mode (Scatter Gather Engine is disabled) provides a configuration for doing simple 0, a DMASR.IOC_Irq asserts
for the associated channel and if enabled generates an interrupt out.

A DMA operation for the MM2S channel is set up and started by the following sequence:

1. Start the MM2S channel running by setting the run/stop bit to 1 (MM2S_DMACR.RS =1). The halted bit (DMASR.Halted) should deassert indicating the MM2S channel is running.

2. If desired, enable interrupts by writing a 1 to MM2S_DMACR.IOC_IrqEn and MM2S_DMACR.Err_IrqEn. The delay interrupt, delay count, and threshold count are not used when the AXI DMA is configured for Direct Register Mode.

3. Write a valid source address to the MM2S_SA register. If AXI DMA is configured for an address space greater than 32, then program the MM2S_SA MSB register. If the AXI DMA is not configured for Data Re-Alignment, then a valid address must be aligned or undefined results occur. What is considered aligned or unaligned is based on the stream data width. When AXI_DMA is configured in Micro Mode, it is your responsibility to specify the correct address. Micro DMA does not take care of the 4K boundary.
For example, if Memory Map Data Width = 32, data is aligned if it is located at word offsets (32-bit offset), that is 0x0, 0x4, 0x8, 0xC, and so forth. If DRE is enabled and Streaming Data Width < 128, then the Source Addresses can be of any byte offset.

4. Write the number of bytes to transfer in the MM2S_LENGTH register. A value of zero written has no effect. A non-zero value causes the MM2S_LENGTH number of bytes to be read on the MM2S AXI4 interface and transmitted out of the MM2S AXI4-Stream interface. The MM2S_LENGTH register must be written last. All other MM2S registers can be written in any order. In the case of Micro DMA, this value cannot exceed [Burst_length * (Memory Mapped Data Width)/8].

A DMA operation for the S2MM channel is set up and started by the following sequence:

1. Start the S2MM channel running by setting the run/stop bit to 1 (S2MM_DMACR.RS = 1). The halted bit (DMASR.Halted) should deassert indicating the S2MM channel is running.

2. If desired, enable interrupts by writing a 1 to S2MM_DMACR.IOC_IrqEn and S2MM_DMACR.Err_IrqEn. The delay interrupt, delay count, and threshold count are not used when the AXI DMA is configured for Direct Register Mode.

3. Write a valid destination address to the S2MM_DA register. If AXI DMA is configured for an address space greater than 32, program the S2MM_DA MSB register.

4. If the AXI DMA is not configured for Data Re-Alignment then a valid address must be

aligned or undefined results occur. What is considered aligned or unaligned is based on the stream data width. For example, if Memory Map Data Width= 32, data is aligned if it is located at word offsets (32-bit offset), that is, 0x0, 0x4, 0x8, 0xC, and so forth. If DRE is enabled and Streaming Data Width < 128 then the Destination Addresses can be of any byte offset.

APPENDIX B - LOOPBACK C code Settings

1) Normal Mode

```
XAxi_WriteReg(SW0_REG1, 0x00000001);
XAxi_WriteReg(SW0_REG0, 0x00000002);

XAxi_WriteReg(SW1_REG1, 0x80000000);
XAxi_WriteReg(SW1_REG2, 0x00000000);
XAxi_WriteReg(SW1_REG0, 0x00000002);

XAxi_WriteReg(SW2_REG1, 0x00000000);
XAxi_WriteReg(SW2_REG2, 0x80000000);
XAxi_WriteReg(SW2_REG0, 0x00000002);

XAxi_WriteReg(SW3_REG1, 0x00000000);
XAxi_WriteReg(SW3_REG0, 0x00000002);
```

2) Loop 1 Active

```
XAxi_WriteReg(SW0_REG1, 0x00000000);
XAxi_WriteReg(SW0_REG0, 0x00000002);

XAxi_WriteReg(SW1_REG1, 0x00000000);
XAxi_WriteReg(SW1_REG2, 0x80000000);
XAxi_WriteReg(SW1_REG0, 0x00000002);
```

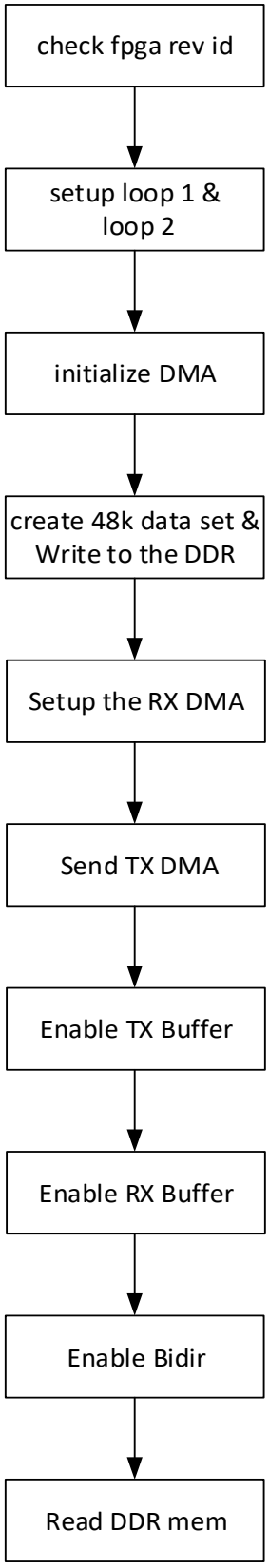
3) Loop 2 Active

```
XAxi_WriteReg(SW0_REG1, 0x00000001);
XAxi_WriteReg(SW0_REG0, 0x00000002);

XAxi_WriteReg(SW1_REG1, 0x80000000);
XAxi_WriteReg(SW1_REG2, 0x00000000);
XAxi_WriteReg(SW1_REG0, 0x00000002);

XAxi_WriteReg(SW2_REG1, 0x80000000);
XAxi_WriteReg(SW2_REG2, 0x00000000);
XAxi_WriteReg(SW2_REG0, 0x00000002);

XAxi_WriteReg(SW3_REG1, 0x00000001);
XAxi_WriteReg(SW3_REG0, 0x00000002);
```

4) Loop 3 Active

```
XAxi_WriteReg(BIDIR_REG0, 0x01000000);
```

APPENDIX C – Operation flow diagram



APPENDIX D – Implementation results

Figure 9 Layout

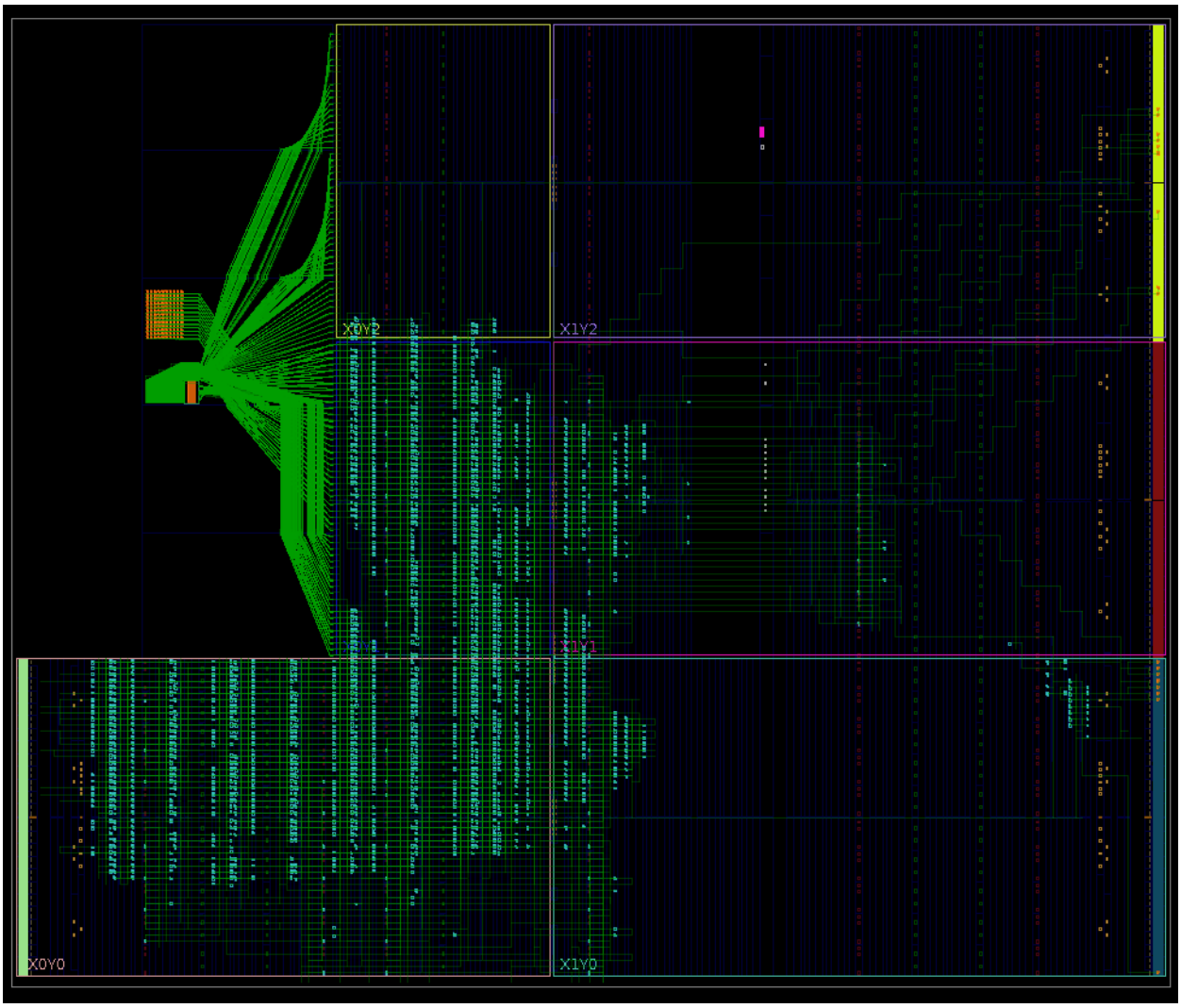


Table 30 Utilization

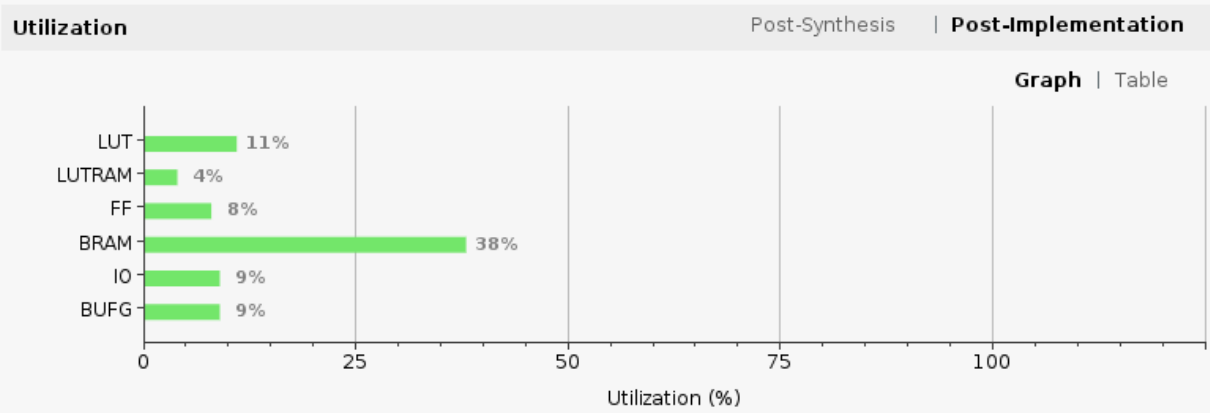


Table 31 Timing and Power

Timing

Worst Negative Slack (WNS):	1.408 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	28669

Implemented Timing Report

Power

Total On-Chip Power:	**1.838 W**
Junction Temperature:	**46.2 °C**
Thermal Margin:	38.8 °C (3.2 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

APPENDIX E – code directory tree

All the C testcase sample code can be found here:
*/gyro2tester/vitis/testcases_src/

All the System Verilog testcase code can be found here:
*/gyro2tester/vivado/ip_repo/testbench

System Verilog design source are all under:
*/gyro2tester/vivado/ip_repo

Standalone Testbench can be found:
*/ gyro2tester/funcsim

Additional Documentation can be found:
*/ gyro2tester/docs

Vivado Project file:
*/gyro2tester/vivado/project/gyro2_tester.xpr

Vivado synthesis timing constraints:
*/gyro2tester/vivado/project/gyro2_tester.srcs/constrs_1/imports/new/gyro_constraints.vdc