GYRO2 TESTER FPGA

Design Specification

Revision 2.0

By Charles Dickinson

Prepared for

Contents

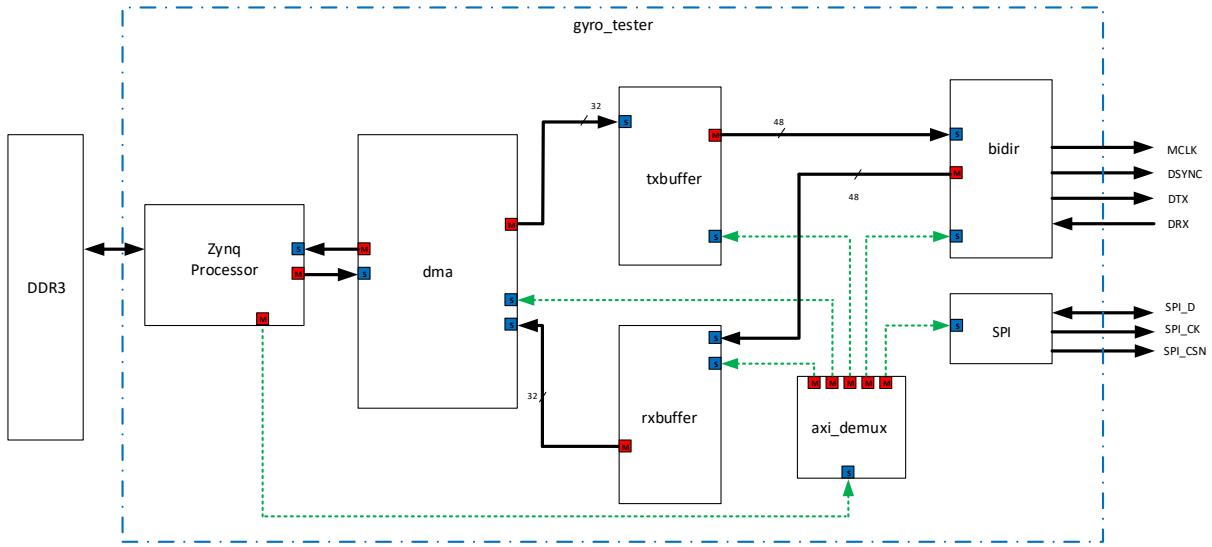
Description

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.
- 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 an various internal test pattern for testing
- Sample C code for various types of tests is provided.

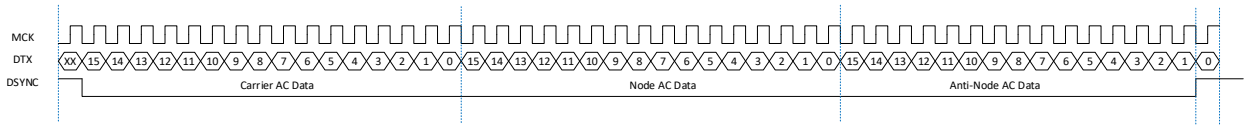
Block Diagram



Transmit Data Path

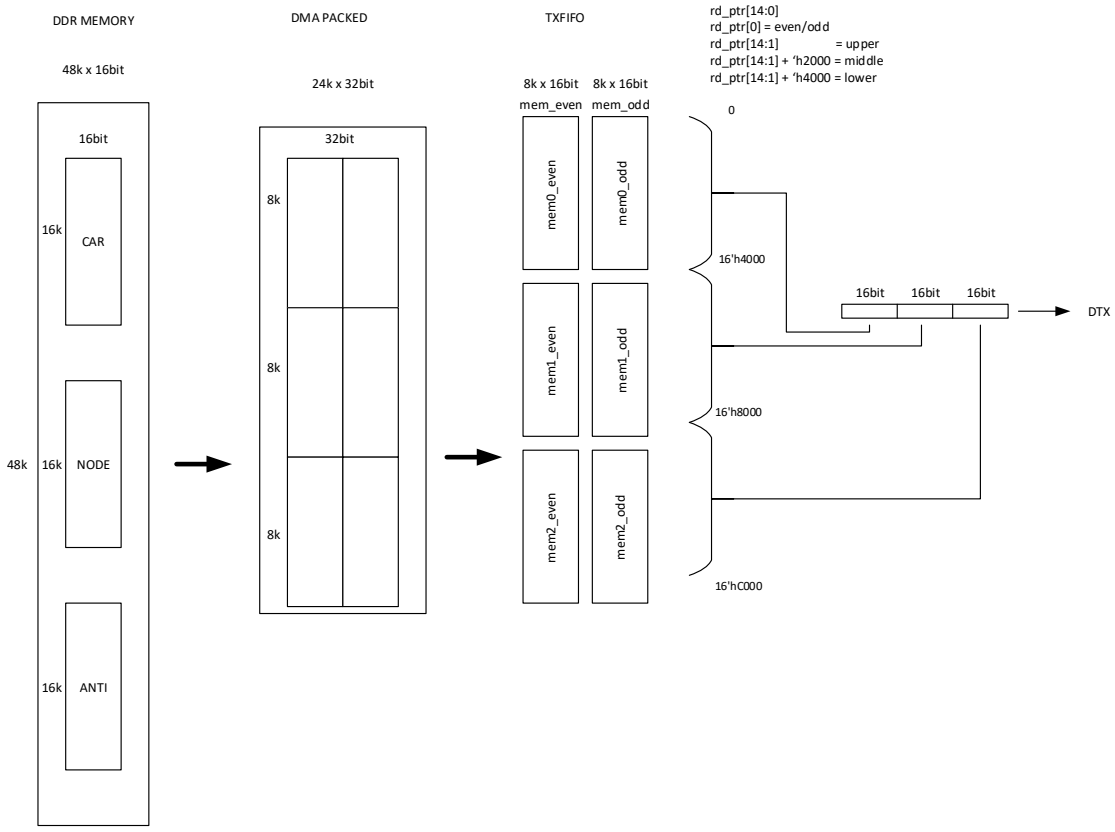
The Transmit data path is responsible for fetching data from the DDR to the TX fifo buffer. Once the buffer is filled with 48K 16bit words it is then sent to the serializer found within the Bidir block. Serial data is sent out on the divided down MCK output clock.

Serial TX data should look like the timing diagram below:



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.

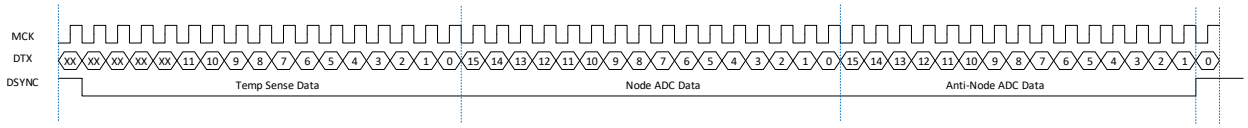
It should be noted that the memory requirement for the TX fifo buffer will be 2 instances of 24Kx16bit memories. The Xilinx compiler will most likely use its own block ram types to achieve this.



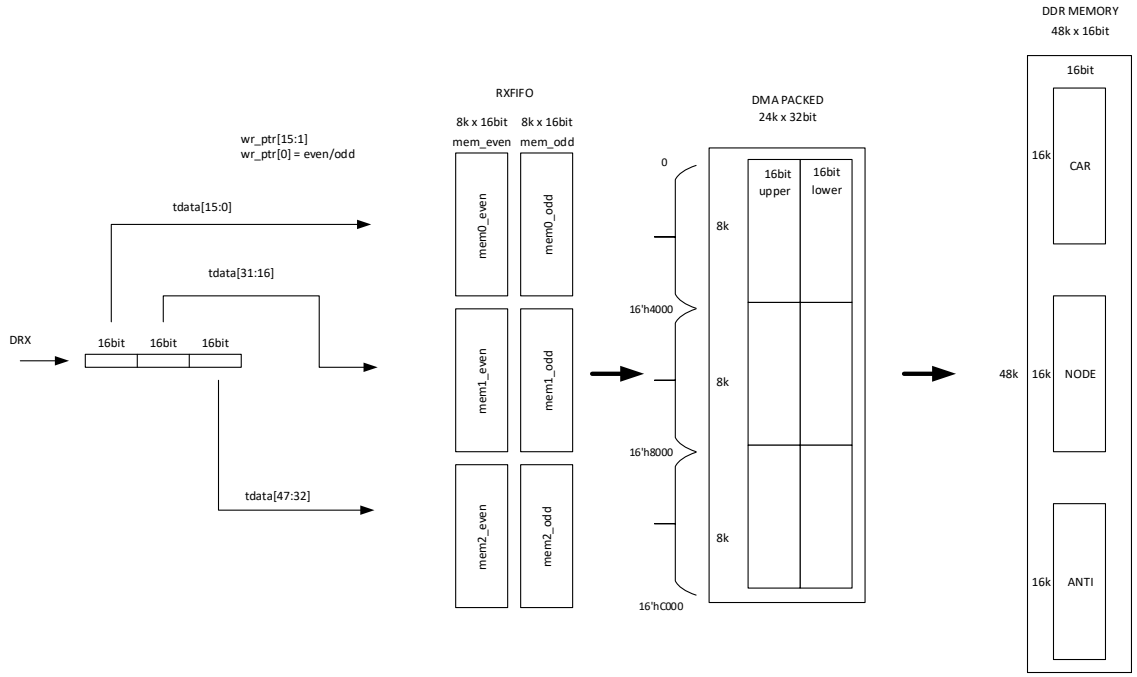
Receive Data Path

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



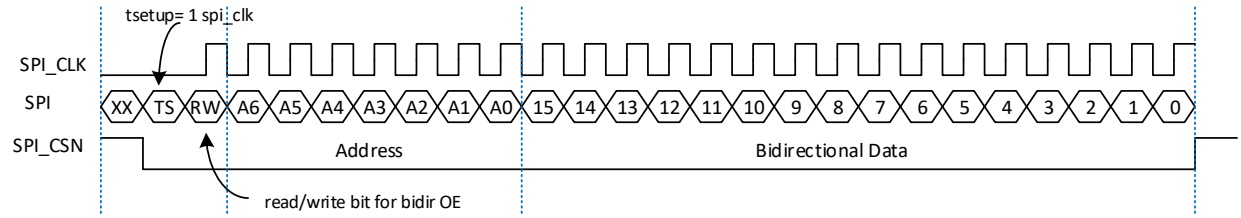
Much like the transmit side the receiver also must do a few transforms on the data to facilitate both the delivery across a packed 32bit DMA interface as well as implement testing loopbacks from the 48bit transmit path. The data flow will look like the following diagram:



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.

SPI Register Interface

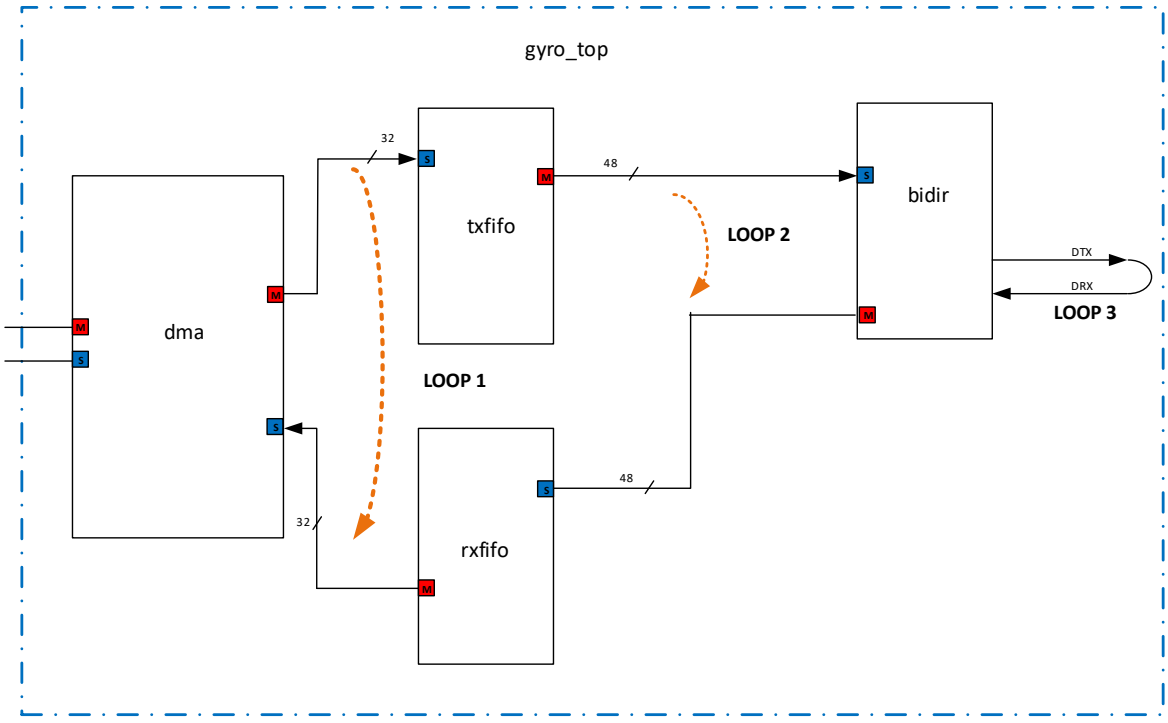
The SPI interface is a custom implementation of the SPI protocol. It should follow the below timing diagram:



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.

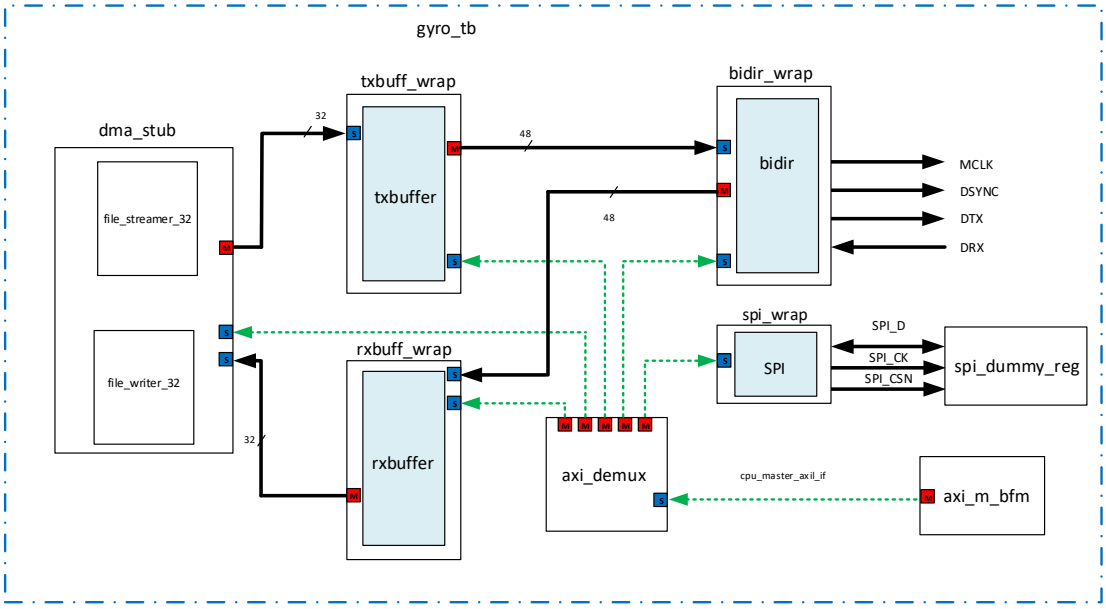


Test Patterns

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.

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.



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

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_0040	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_0040	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_0040	AXI SW3 M1 Addr	R/W	8000_0000	M1 Selector value

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

AXI SW0 M0 Addr (0x43C0_0040)

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

SPI Control 0 (0x43C1_0000)

Name	Bits	Description
SPI Update	0	SPI update. setting a one sends the data and address fields
SPI Direction	1	SPI direction 0=write ; 1=read
SPI clock Div	3:2	Div16,Div8,Div,4,Div2
Reserved	31:4	Reserved

SPI Address (0x43C1_0004)

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

SPI Data (0x43C1_0008)

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

SPI Read Data (0x43C1_000C)

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

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.

BiDir Control 1 (0x43C2_0004)

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

BiDir Control 2 (0x43C2_0008)

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

BiDir Control 3 (0x43C2_000C)

Name	Bits	Description
Reserved	31:0	Reserved

RX FIFO Control 0 (0x43C3_0000)

Name	Bits	Description
Fifo Push En	0	Enables the Rx Fifo Push (filling of fifo).
Reserved	15:1	Reserved
Test Pattern	16	Test fill pattern into the RX Buffer. Disables incoming data
Reserved	31:1	Reserved

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

RX FIFO Control 2 (0x43C3_0008)

Name	Bits	Description
Fifo Pop En	0	Enables the Rx Fifo Pop (drain).
Reserved	31:1	Reserved

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.

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

TX FIFO Control 1 (0x43C4_0004)

Name	Bits	Description
Reserved	31:0	Reserved

TX FIFO Status RevID (0x43C4_0008)

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

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.

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

AXI SW1 M0 Addr (0x43C5_0040)

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

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

AXI SW2 M0 Addr (0x43C6_0040)

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

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

AXI SW3 M0 Addr (0x43C7_0040)

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

APPENDIX A (LOOPBACK C code Settings)

1) Normal Mode

```
////////////////////////////////////////////////
// AXI Stream Switches  1 & 2  NORMAL       //
////////////////////////////////////////////////
XAxiDma_WriteReg(0x43C00000,0x00000040,0x00000001);  // SW0 connect M0 to S1
XAxiDma_WriteReg(0x43C00000,0x00000000,0x00000002);  // Activate

XAxiDma_WriteReg(0x43C50000,0x00000040,0x80000000);  // SW1 deactivate M0
XAxiDma_WriteReg(0x43C50000,0x00000044,0x00000000);  // SW1 connect M1 to S0
XAxiDma_WriteReg(0x43C50000,0x00000000,0x00000002);  // Activate

////////////////////////////////////////////////
// AXI Stream Switches  3 & 4  NORMAL       //
////////////////////////////////////////////////

XAxiDma_WriteReg(0x43C60000,0x00000040,0x00000000);  // SW1 connect M0 to S0
XAxiDma_WriteReg(0x43C60000,0x00000044,0x80000000);  // SW1 deactivate M1
XAxiDma_WriteReg(0x43C60000,0x00000000,0x00000002);  // Activate

XAxiDma_WriteReg(0x43C70000,0x00000040,0x00000000);  // SW0 connect M0 to S0
XAxiDma_WriteReg(0x43C70000,0x00000000,0x00000002);  // Activate
```

2) Loop 1 Active

```
////////////////////////////////////////////////
// AXI Stream Switches  1 & 2   LOOP         //
////////////////////////////////////////////////
XAxiDma_WriteReg(0x43C00000,0x00000040,0x00000000);  // SW0 connect M0 to S0
XAxiDma_WriteReg(0x43C00000,0x00000000,0x00000002);  // Activate

XAxiDma_WriteReg(0x43C50000,0x00000040,0x00000000);  // SW1 connect M0 to S0
XAxiDma_WriteReg(0x43C50000,0x00000044,0x80000000);  // SW1 deactivate M1
XAxiDma_WriteReg(0x43C50000,0x00000000,0x00000002);  // Activate

////////////////////////////////////////////////
// AXI Stream Switches  3 & 4   NORMAL       //
////////////////////////////////////////////////

XAxiDma_WriteReg(0x43C60000,0x00000040,0x00000000);  // SW1 connect M0 to S0
XAxiDma_WriteReg(0x43C60000,0x00000044,0x80000000);  // SW1 deactivate M1
XAxiDma_WriteReg(0x43C60000,0x00000000,0x00000002);  // Activate

XAxiDma_WriteReg(0x43C70000,0x00000040,0x00000000);  // SW0 connect M0 to S0
XAxiDma_WriteReg(0x43C70000,0x00000000,0x00000002);  // Activate
```

3) Loop 2 Active

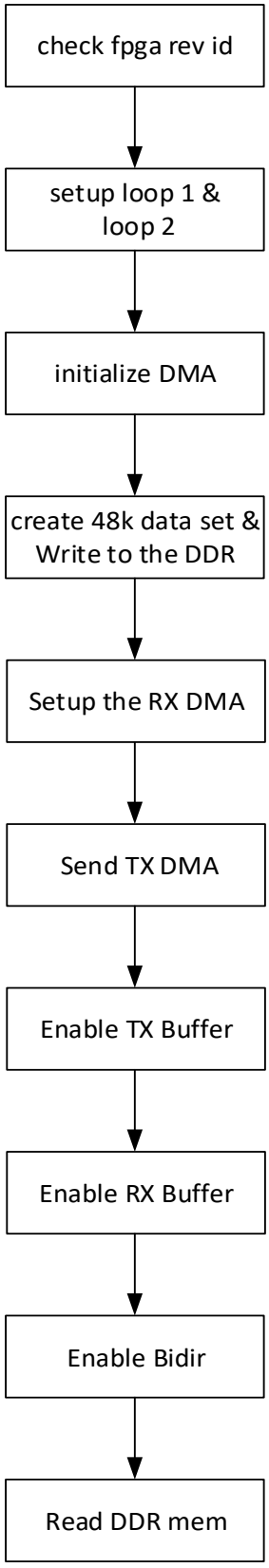
```
////////////////////////////////////////////////
// AXI Stream Switches  1 & 2   NORMAL       //
////////////////////////////////////////////////
XAxiDma_WriteReg(0x43C00000,0x00000040,0x00000001);  // SW0 connect M0 to S1
XAxiDma_WriteReg(0x43C00000,0x00000000,0x00000002);  // Activate
```

```
XAxiDma_WriteReg(0x43C50000,0x00000040,0x80000000);  // SW1 deactivate M0
XAxiDma_WriteReg(0x43C50000,0x00000044,0x00000000);  // SW1 connect M1 to S0
XAxiDma_WriteReg(0x43C50000,0x00000000,0x00000002);  // Activate

////////////////////////////////////////////
// AXI Stream Switches  3 & 4   LOOP        //
////////////////////////////////////////////

XAxiDma_WriteReg(0x43C60000,0x00000040,0x80000000);  // SW1 deactivate M0
XAxiDma_WriteReg(0x43C60000,0x00000044,0x00000000);  // SW1 connect M1 to S0
XAxiDma_WriteReg(0x43C60000,0x00000000,0x00000002);  // Activate

XAxiDma_WriteReg(0x43C70000,0x00000040,0x00000001);  // SW0 connect M0 to S1
XAxiDma_WriteReg(0x43C70000,0x00000000,0x00000002);  // Activate
```

4) Loop 3 Active

```
XAxiDma_WriteReg(0x43C20000,0x00000000,0x01000000);  // Bidir loopback
```

APPENDIX B (C code operation flow diagram)



APPENDIX C (sample code)

```
#include <stdio.h>
#include "xaxidma.h"
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "dma_controller.h"

XAxiDma AxiDma; //DMA device instance definition

int main(){
   init_platform();

   print("Putting the board into Gyro Functional mode\n\r");

   xil_printf("FPGA Build REViD %x \r\n", XAxiDma_ReadReg(0x43C40000,0x00000008));

   xil_printf("Intial Tx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C40000,0x0000000C));
   xil_printf("Intial Rx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C30000,0x0000000C));

   ////////////////////////////////////////////
   // AXI Stream Switches  1 & 2  NORMAL      //
   ////////////////////////////////////////////
   XAxiDma_WriteReg(0x43C00000,0x00000040,0x00000001);  // SW0 connect M0 to S1
   XAxiDma_WriteReg(0x43C00000,0x00000000,0x00000002);  // Activate

   XAxiDma_WriteReg(0x43C50000,0x00000040,0x80000000);  // SW1 deactivate M0
   XAxiDma_WriteReg(0x43C50000,0x00000044,0x00000000);  // SW1 connect M1 to S0
   XAxiDma_WriteReg(0x43C50000,0x00000000,0x00000002);  // Activate

   ////////////////////////////////////////////
   // AXI Stream Switches  3 & 4  NORMAL      //
   ////////////////////////////////////////////

   XAxiDma_WriteReg(0x43C60000,0x00000040,0x00000000);  // SW1 connect M0 to S0
   XAxiDma_WriteReg(0x43C60000,0x00000044,0x80000000);  // SW1 deactivate M1
   XAxiDma_WriteReg(0x43C60000,0x00000000,0x00000002);  // Activate

   XAxiDma_WriteReg(0x43C70000,0x00000040,0x00000000);  // SW0 connect M0 to S0
   XAxiDma_WriteReg(0x43C70000,0x00000000,0x00000002);  // Activate
```

```
XAxiDma_Config *CfgPtr; //DMA configuration pointer

    int Status, Index;
    u16 *TxBufferPtr;
    u16 *RxBufferPtr;
    u16 Value;

    TxBufferPtr = (u16 *)TX_BUFFER_BASE;
    RxBufferPtr = (u16 *)RX_BUFFER_BASE;

    // Initialize memory to all zeros
    for(Index = 0; Index < MAX_PKT_LEN; Index ++){
        TxBufferPtr[Index] = 0x0000;
        RxBufferPtr[Index] = 0x0000;
    }

    // Initialize the XAxiDma device
    CfgPtr = XAxiDma_LookupConfig(DMA_DEV_ID);
    if (!CfgPtr) {
        xil_printf("No config found for %d\r\n", DMA_DEV_ID);
        return XST_FAILURE;
    }

    Status = XAxiDma_CfgInitialize(&AxiDma, CfgPtr);
    if (Status != XST_SUCCESS) {
        xil_printf("Initialization failed %d\r\n", Status);
        return XST_FAILURE;
    }

    if(XAxiDma_HasSg(&AxiDma)){
        xil_printf("Device configured as SG mode \r\n");
        return XST_FAILURE;
    }

    XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DEVICE_TO_DMA);
    XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DMA_TO_DEVICE);

    Value = 0x0000;

    for(Index = 0; Index < MAX_PKT_LEN/2; Index ++){
        TxBufferPtr[Index] = Value;

        if(Value == 0xBFFF){
            Value = 0x0000;
```

```
        }
        else{
            Value = (Value + 1);
        }
    }

    Xil_DCacheFlushRange((UINTPTR)TxBufferPtr, MAX_PKT_LEN);
    Xil_DCacheFlushRange((UINTPTR)RxBufferPtr, MAX_PKT_LEN);

    XAxiDma_Reset(&AxiDma);

////////////////////////////////////////////
// Setup & kick off S2MM channel first    //
// You are required to first turn on the  //
// DMA RX to receive before running the TX //
////////////////////////////////////////////
Status = XAxiDma_S2MMtransfer(&AxiDma,(UINTPTR) RxBufferPtr, MAX_PKT_LEN);

    if (Status != XST_SUCCESS){
        xil_printf("XAXIDMA_DEVICE_TO_DMA transfer failed...\r\n");
        return XST_FAILURE;
    }

xil_printf("RX S2MM DEVICE_TO_DMA READY...\r\n");
////////////////////////////////////////////
// Send the TX DMA into the Device        //
//                                        //
////////////////////////////////////////////
Status = XAxiDma_MM2Stransfer(&AxiDma,(UINTPTR) TxBufferPtr, MAX_PKT_LEN);

    if (Status != XST_SUCCESS){
        xil_printf("XAXIDMA_DMA_TO_DEVICE transfer failed...\r\n");
        return XST_FAILURE;
    }
////////////////////////////////////////////
// Setup TX FIFO                          //
//                                        //
////////////////////////////////////////////
XAxiDma_WriteReg(0x43C40000,0x00000000,0x00000001);  // Enable the TXFIFO

    while(XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE)){
        if (XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE)){
            xil_printf("MM2S channel is busy...\r\n");
        }
```

```
    }

   xil_printf("TX MM2S DMA_TO_DEVICE DATA SENT...\r\n");
   xil_printf("Tx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C40000,0x0000000C));

   ////////////////////////////////////////
   // Setup RX FIFO                      //
   // To Fill Only                       //
   //                                    //
   ////////////////////////////////////////
   XAxiDma_WriteReg(0x43C30000,0x00000000,0x00000001);  // Enable the PUSH
    xil_printf("Rx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C30000,0x0000000C));

   ////////////////////////////////////////
   // Setup BiDir                        //
   // Start the LoopBack and Kick it off //
   ////////////////////////////////////////
    XAxiDma_WriteReg(0x43C20000,0x00000000,0x01070000);  // bit6:Enable BiDir,bit4:clkDiv
    XAxiDma_WriteReg(0x43C20000,0x00000008,0x00000001);
    XAxiDma_WriteReg(0x43C20000,0x00000004,0x00000011);  // Kick off both TX and RX

    xil_printf("Serial Loop Back Set...\r\n");

   ////////////////////////////////////////
   // Setup RX FIFO                      //
   // Here we should have a full RX FIFO //
   // Turn it on to Drain it             //
   ////////////////////////////////////////

   xil_printf("Rx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C30000,0x0000000C));
   XAxiDma_WriteReg(0x43C30000,0x00000008,0x00000001);  // Enable the Pop

      while(XAxiDma_Busy(&AxiDma,XAXIDMA_DEVICE_TO_DMA) ||
XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE)){
           if (XAxiDma_Busy(&AxiDma,XAXIDMA_DEVICE_TO_DMA) == TRUE){
                xil_printf("S2MM channel is busy...\r\n");
           }

           if (XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE)){
                xil_printf("MM2S channel is busy...\r\n");
           }
      }

   xil_printf("RX Fifo is Drained...\r\n");
```

```
   ////////////////////////////////////////////////
   // Check if TX FIFO has Drained                //
   ////////////////////////////////////////////////
   xil_printf("Rx Fifo Levels %x \r\n", XAxiDma_ReadReg(0x43C30000,0x0000000C));

   print("48K Loopback test done Checking memory locations for Data\n\r");

      for(Index = 0; Index < MAX_PKT_LEN/2; Index++) {
            xil_printf("Received data packet %d: RX DATA %x / TX DATA %x\r\n", Index, (unsigned
int)RxBufferPtr[Index], (unsigned int)TxBufferPtr[Index]);
      }

      XAxiDma_Reset(&AxiDma);

   cleanup_platform();
  return 0;
}
```