FPGA DESIGN FOR CAPTURING DATA FROM TI ADCs

HAMZA FRAZ

INTRODUCTION

This document is intended to explain the design used to capture data from TI ADCs into an FPGA and then streaming it to the PC. The document has been partitioned into following sections. *Section I* gives a brief overview of the test equipment setup. *Section II* describes in detail the base FPGA design used to handle data after it has been captured into the FPGA flip-flops. *Section III* outlines the difference between two versions of the extended design used to capture data from Serial Output and Parallel Output ADCs. *Section IV* describes in details configuration registers and the addressing scheme used. *Section V* discusses the partition of design with respect to implementation and relates the functionality to file names used in project. *Appendix A* contains example MATLAB codes used to initialize and capture data from the system under different configurations.

Section I

SYSTEM OVERVIEW

Fig. 1 is a block diagram that illustrates the test equipment setup. Under the currently used setup, the ADC under test is located on an evaluation board, separate from the FPGA board. The Virtex 4 FPGA and the UART-to-USB Bridge are located on TI's TUSB 1200 evaluation board, which sits on top of the ADC board through a LVDS connector. TUSB 1200 connects to the host computer through a USB cable and is seen by the host computer as a COM port, after its driver installation files are in place.

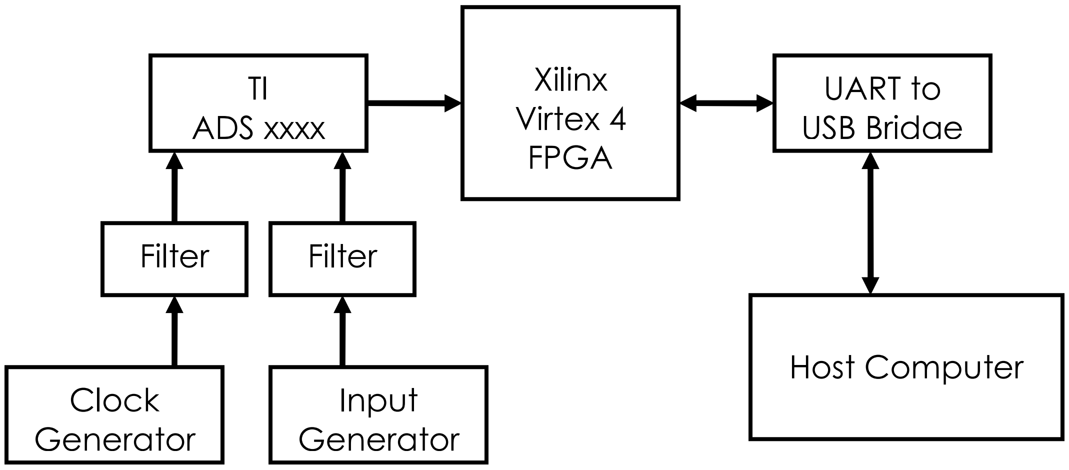

Figure 1 Test Equipment Setup for ADCs

The ADCs under test can be divided into two main categories.

- *Serial output ADCs*
  These ADCs output serial data on LVDS pairs and therefore require a SERDES block in the FPGA for serial to parallel conversion.
- *Parallel output ADCs*
  These ADCs output parallel data on LVDS pairs and therefore do not require any conversion in the FPGA. However, they might source data on the LVDS lines in a

DDR fashion which requires some logic in the FPGA design that separates positive-edge and negative-edge data.

These two categories of ADCs call for two FPGA designs customized appropriately. The base of these two FPGA designs is the same but there are slight differences in the way they capture data from their respective ADCs. The base design shared by the two approaches is explained first. Different use of this base design in the two approaches is explained later.

Section II

BASE DESIGN

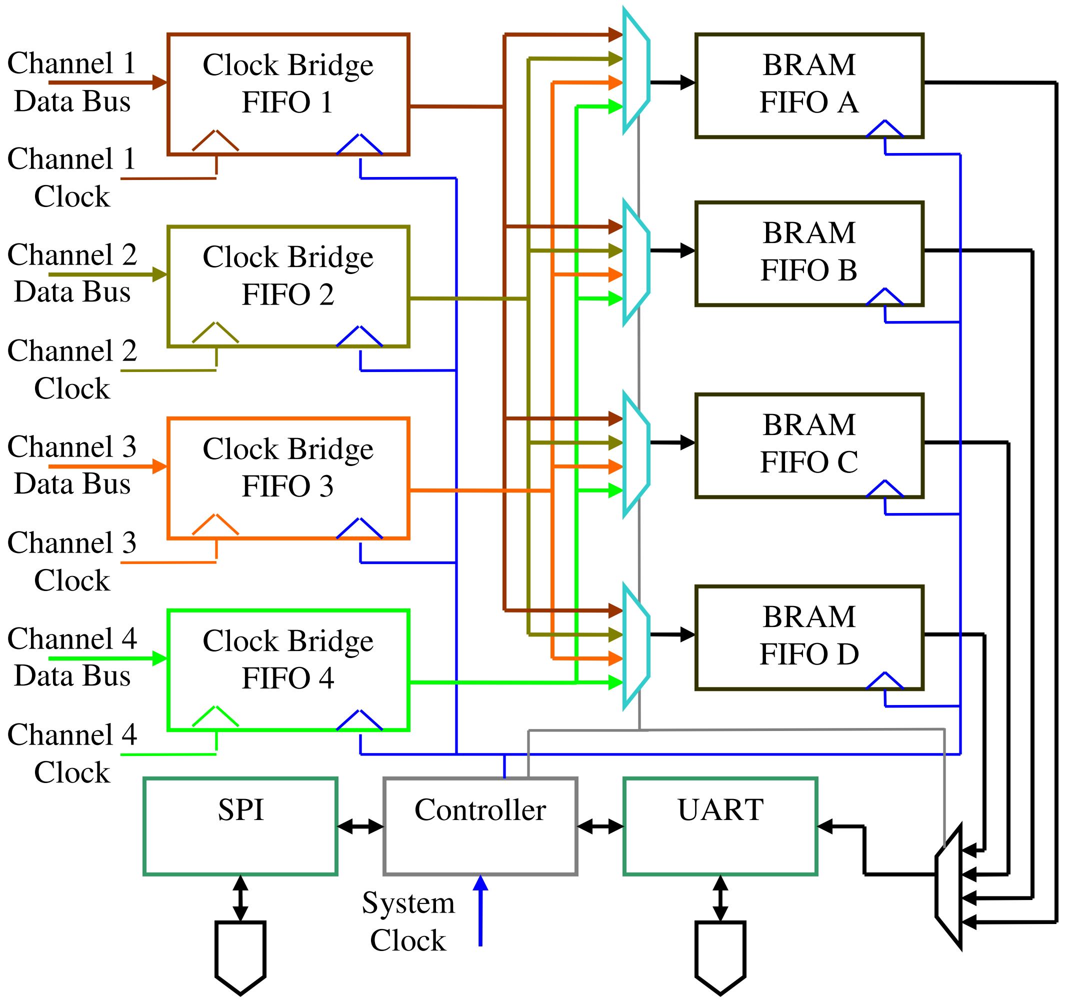

Figure 2 Base Design

Fig. 2 block diagram illustrates details of the base design used to dump data captured by the FPGA into memories.

The base design gets 4 streams of data from entities that capture data from the ADC. The data capturing entity is SERDES block from Xilinx XAPP866, if the ADC under test is a serial output ADC. The data capture entity is a group of Xilinx ISERDES I/O primitives, if the ADC under test is a parallel output ADC. The details of data capturing are discussed in Section III. A description of blocks of Fig. 2 follows.

CLOCK BRIDGE FIFO

The clock bridge FIFO does precisely what its name says: it acts as a bridge for data when it crosses clock domains from channel clock to system clock. The channel clock for each channel can take any value less than 250 MHz and is used as the write clock for respective channel FIFO. All channel clocks are shown in different color in Fig. 2. The system clock is fixed at 250 MHz and is used as the read clock for all channel FIFOs. It is represented in blue color in Fig. 2. A self-addressing FIFO design based on Xilinx XAPP291 is used. Self-addressing FIFO uses block memories to store both data and address information in a single memory location, thus eliminating the need for external FIFO address counters. Each clock bridge FIFO is used for data throttling in continuous mode and therefore has a very small size of 16x36-bits.

DATA SELECT MULTIPLEXER FOR BLOCK RAM

As shown in Fig. 2, there sits a 4-to-1 16-bit multiplexer at the input of each BRAM FIFO which selects based on the control signals from controller, which data stream is presented at the input of the BRAM FIFO for storage. This multiplexer gives the flexibility of storing data from selected or all data streams into the dump BRAM FIFOs.

BRAM FIFO

The purpose of BRAM FIFOs shown in Fig. 2 is data storage. Therefore, they form the biggest component of the FPGA base design in terms of area, consuming 100% of BRAM resources of the Xilinx 4VLX25SF363 FPGA.

Each BRAM FIFO is 16397x16-bits in size. Since, it is extremely hard to run a single memory block of this size at 250 MHz; it has been broken down into three smaller sub-FIFOs that are used in a piggybacked fashion. Two of these sub-FIFOs are 8191x16-bits in size and the third one is 15x16-bits in size. All these components are generated with Xilinx Core-Generator Wizard.

During data capture operation, the BRAM FIFOs are dumped with data sourced from the clock-bridge FIFOs until they are completely filled. The FIFO_FULL event triggers the controller to read data from these FIFOs and provide it to UART for transmission to the host computer.

UART

A simple UART design provided by Xilinx has been used in the base design to interface it to TUSB chip. The UART takes single byte of data from BRAM FIFOs and transmits it to the host computer. It repeats this operation until the end of last BRAM FIFO is reached and FIFO_EMPTY event commences for all BRAM FIFOs.

The UART baud clock is configurable through controller's configuration register and can be set to support any baud rate. More details are provided in the controller design.

SPI

A SPI interface is added to the FPGA base design to allow it to configure ADCs that need to be initialized to appropriate modes before they can begin their normal operation. The SPI can be configured to transmit and receive 8, 16 or 24 bits, address and data bits combined. The data input to SPI comes from the controller.

Similar to UART, the baud rate of SPI can be configured to any value by writing to configuration register in the controller. More details are provided in the controller design.

CONTROLLER

The controller consists of configuration logic as well as state machines that dictate the sequence of operations for capturing and handling ADC data. The controller has been implemented in a de-centralized fashion with each block having its control logic within itself. The overall flow of operations is dictated by user writes to configuration registers. Each configuration register can be written by sending an address byte and a data byte to the FPGA UART.

The sequence of operations for capturing data from ADC and transmitting to the host computer is illustrated in Fig. 3. The sequence shown assumes that UART configuration registers have been configured for the required baud rate.

Fig. 4 illustrates the sequence of writes that need to be initiated by the user to transmit data over SPI. Similarly, Fig. 5 shows the steps to vary skew on data/clock lines coming from the ADC into the FPGA.

All of the configuration registers are readable through UART. When a register read operation is requested, the current design of UART sends the value of the requested register 8 times i.e. 8 identical bytes are transmitted over the UART to the host computer. Fig. 6 illustrated steps to accomplish a read operation through UART.

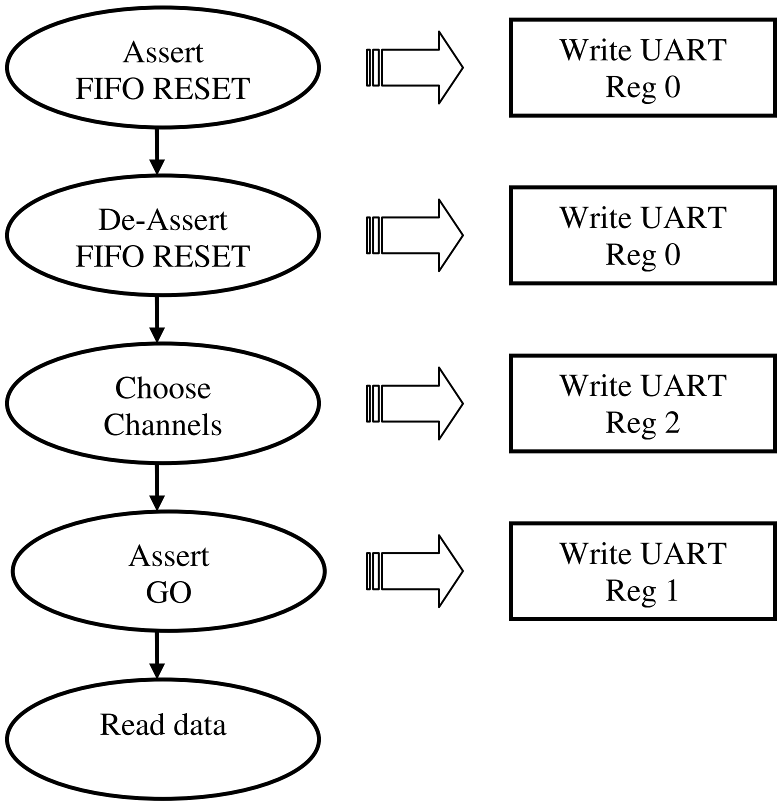

Figure 3 Sequence to initiate Data Capture

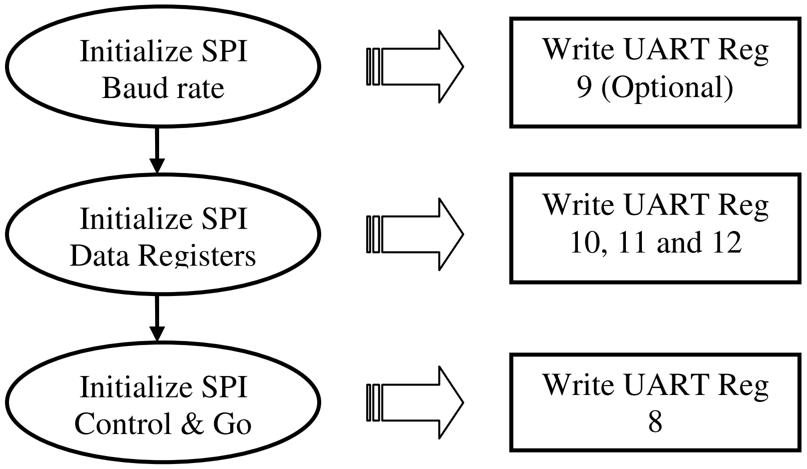
Initialize SPI Baud rate
Write UART Reg 9 (Optional)
Initialize SPI Data Registers
Write UART Reg 10, 11 and 12
Initialize SPI Control & Go
Write UART Reg 8

Figure 4 Sequence to initiate SPI transaction

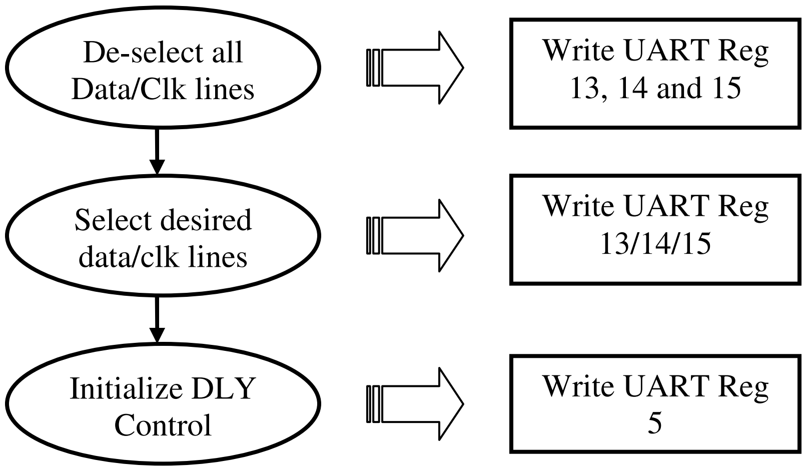
De-select all Data/Clk lines
Write UART Reg 13, 14 and 15
Select desired data/clk lines
Write UART Reg 13/14/15
Initialize DLY Control
Write UART Reg 5

Figure 5 Sequence to de-skew clock/data lines from ADC

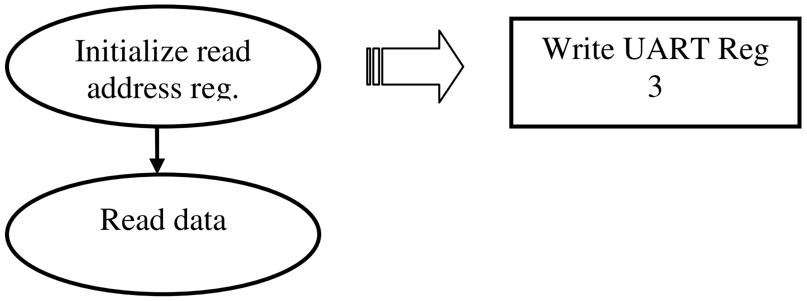

Figure 6 Sequence for Read Register operation

The configuration registers have been discussed in detail in Section IV.

Section III

EXTENDED DESIGN I

Extended design I is used when the ADC under test is a serial output ADC. A block diagram of extended design I is shown in Fig. 7. The XAPP866 is used to convert the serial data being sourced on LVDS pairs by the ADC, to parallel data streams. Each bit stream at the output of XAPP866 can be treated as a data channel, synchronized to its channel clock.

The output of XAPP866 can feed the input of the base design without any glue logic. The base design then can be configured to capture data from any channel.

The system clock needed for the base design is also generated by XAPP866 and is equal to 200 MHz.

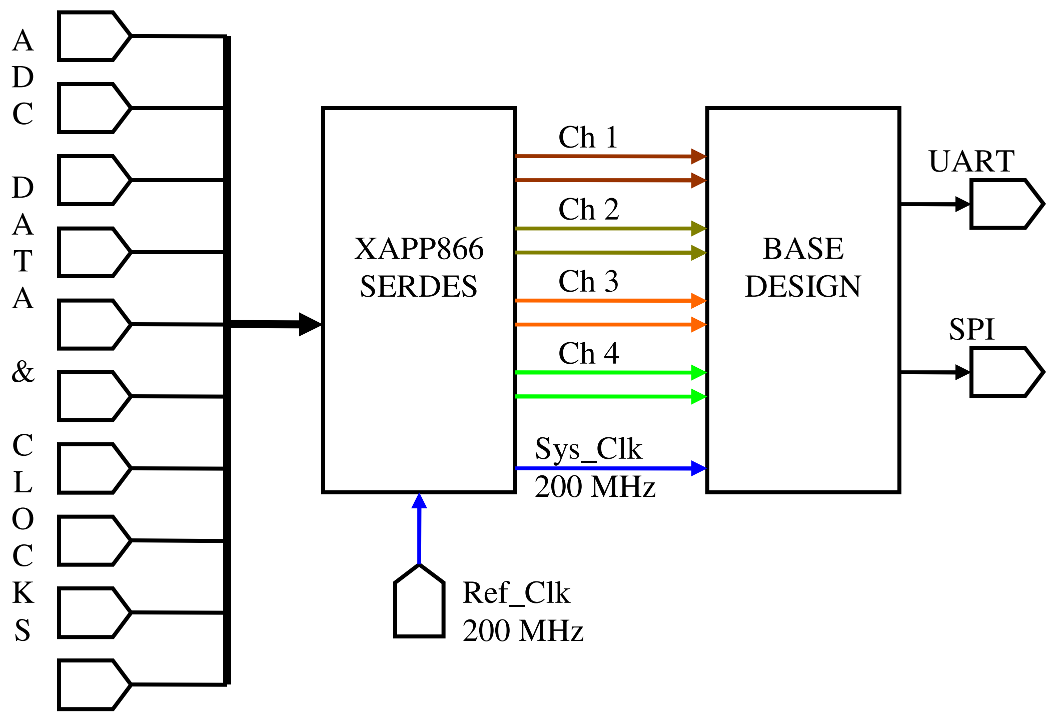

Figure 7 Extended Design I

EXTENDED DESIGN II

Extended design II is used when the ADC under test is a parallel output ADC. It can be used to capture data that is SDR, bit-wise DDR or sample-wise DDR. A block diagram of extended design II is shown in Fig. 8.

The ADC under test drives its output data on LVDS pairs that terminate at the input of Xilinx ISERDES primitives inside the FPGA. Each ISERDES primitive is configured to take in DDR data and convert it into two parallel data streams. These 2 parallel data streams form the input of base design as channel 1 and channel 2. When the ADC under test outputs data in sample-wise DDR fashion, channel 1 and 2 should be captured into FIFOs and retrieved. Channel 3 and channel 4 of the base design are driven by the same data except that zeros have been inserted at every alternate bit location in the output of the ISERDES primitives. This has been done to facilitate extraction of samples when the ADC outputs data in bit-wise DDR fashion. Thus channel 3 and 4 should be captured into FIFOs and retrieved when the ADC under test outputs data is bit-wise DDR.

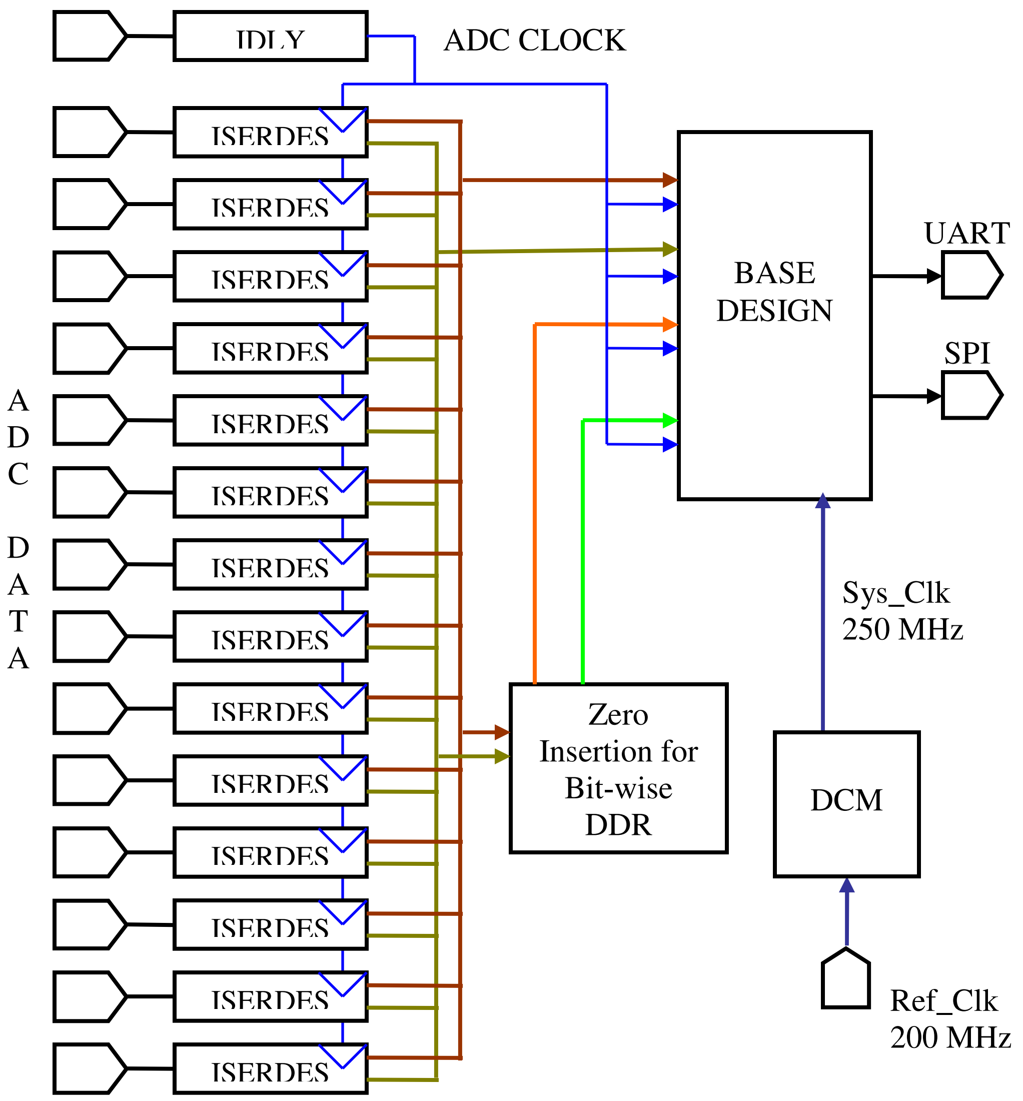

Figure 8 Extended Design II

The system clock needed for the base design is also generated by a wrapper that encapsulates ISERDES primitives and the base design. The system clock is equal to 250 MHz.

Section IV

REGISTER MAP

There are 16 configuration registers that need to be appropriately initialized before any capture operation can take place. All these register can be accessed for read/write operations. Table 1 gives a summary of all the modes that can be programmed through the UART.

Table 1 Summary of Functions Supported by UART Interface

Register Address	Register Functions							
A3-A0	D7	D6	D5	D4	D3	D2	D1	D0
0x0	<RST> Software Reset							
0x1	X							<GO> Start Capture
0x2	X				<EN4> Enable Channel 4	<EN3> Enable Channel 3	<EN2> Enable Channel 2	<EN1> Enable Channel 1
0x3	X				<RA> Read Address for Controller Registers			
0x4	<BRAM_FIFO_SEL>				<BRAM_FIFO_SIZE>			
0x5	<DYRST> Reset IOB Delay	<DYINC> Inc/Dec IOB Delay	<DYVAL> Inc/Dec value (0-63)					
0x6	<UBAUD> UART Baud Rate							
0x7	X				<CTS> UART	<DSR> UART	<DCD> UART	<RI> UART
0x8	<SPIGO> SPI Start	<RCE> SPI Receive Clock Edge	<TCE> SPI Transmit Clock Edge	<SPIBITS> Can be set to 8, 16 or 24				
0x9	<SBAUD> SPI Baud Rate							
0xA	<SPID[07:00]> SPI Transmit data[07:00]							
0xB	<SPID[15:08]> SPI Transmit data[15:08]							
0xC	<SPID[23:16]> SPI Transmit data[23:16]							
0xD	<DQ7> Select data line 7 for Delay Adjust	<DQ6> Select data line 6 for Delay Adjust	<DQ5> Select data line 5 for Delay Adjust	<DQ4> Select data line 4 for Delay Adjust	<DQ3> Select data line 3 for Delay Adjust	<DQ2> Select data line 2 for Delay Adjust	<DQ1> Select data line 1 for Delay Adjust	<DQ0> Select data line 0 for Delay Adjust

0xE	**X**	**X**	**<DQ13> Select data line 13 for Delay Adjust**	**<DQ12> Select data line 12 for Delay Adjust**	**<DQ11> Select data line 11 for Delay Adjust**	**<DQ10> Select data line 10 for Delay Adjust**	**<DQ9> Select data line 9 for Delay Adjust**	**<DQ8> Select data line 8 for Delay Adjust**
0xF				**X**				**<DQS> Select ADC Clock for Delay Adjust**

DESCRIPTION OF SERIAL REGISTERS

Each register function is explained in detail below.

Table 2 UART Register 0

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0				**<RST> Software Reset**				

D7 DOWNTO D0	**<RST>**	**Software Reset for FIFOs**
0x00	Normal operation	
0xFF	Software reset applied - resets all FIFO pointers and memories. The reset is NOT self clearing and requires the user to de-assert it, after it has been asserted.	
0xFF	Default value.	

Table 3 UART Register 1

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
1				**X**				**<GO> Start Capture**

D0 **<GO>**

1	Enables data capture into FIFOs. This bit self clears to 0 after all FIFOs have been filled up and the UART has transmitted all data to the host computer.
0	Default value.

Table 4 UART Register 2

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
2	**X**				**<EN4> Enable Channel 4**	**<EN3> Enable Channel 3**	**<EN2> Enable Channel 2**	**<EN1> Enable Channel 1**

D3 DOWNTO D0	**<EN4:1>** **Channel Enables For Base Design**
0x0	No Data is captured into BRAM FIFOs.
0x1	Data from Channel 1 is captured into all BRAM FIFOs.
0x2	Data from Channel 2 is captured into all BRAM FIFOs.
0x4	Data from Channel 3 is captured into all BRAM FIFOs.
0x8	Data from Channel 4 is captured into all BRAM FIFOs.
0x3	Data from Channels 1 and 2 is captured into BRAM FIFOs 1, 2 and 3, 4 respectively.
0xC	Data from Channels 3 and 4 is captured into BRAM FIFOs 1, 2 and 3, 4 respectively.
0xF	Data from Channels 1, 2, 3 and 4 is captured into BRAM FIFOs 1, 2, 3, 4 respectively.
0x0	Default value.

Table 5 UART Register 3

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
3	**X**				**<RAC> Read Address for Controller Registers**			

D3 DOWNTO D0	**<RAC>** **Read Address Register**
0x0	Sends value of UART register 0 to the host computer.
0x1	Sends value of UART register 1 to the host computer.
0x2	Sends value of UART register 2 to the host computer.
0x3	Sends value of UART register 3 to the host computer.
0x4	Sends value of UART register 4 to the host computer.
0x5	Sends value of UART register 5 to the host computer.
0x6	Sends value of UART register 6 to the host computer.
0x7	Sends value of UART register 7 to the host computer.

0x8	Sends value of UART register 8 to the host computer.
0x9	Sends value of UART register 9 to the host computer.
0xA	Sends value of UART register 10 to the host computer.
0xB	Sends value of UART register 11 to the host computer.
0xC	Sends value of UART register 12 to the host computer.
0xD	Sends value of UART register 13 to the host computer.
0xE	Sends value of UART register 14 to the host computer.
0xF	Sends value of UART register 15 to the host computer.
0x0	Default value.

Table 6 UART Register 4

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
4	**<BRAM_FIFO_SEL>**				**<BRAM_FIFO_SIZE>**			

D3 DOWNTO D0	**<BRAM_FIFO_SIZE> Size of BRAM FIFO that is to be transmitted over UART**
0x8	Default value.
0x1	2K samples (4KB) from each of selected BRAM FIFO
0x2	4K samples (8KB) from each of selected BRAM FIFO
0x4	8K samples (16KB) from each of selected BRAM FIFO
0x8	(16K+13) samples (32KB+26) from each of selected BRAM FIFO

D7 DOWNTO D4	**<BRAM_FIFO_SEL> Select BRAM FIFOs that are to be transmitted overUART**
D7	
1	Default value.
1	Select BRAM_FIFO 4
0	De-select BRAM_FIFO 4
D6	
1	Default value.
1	Select BRAM_FIFO 3
0	De-select BRAM_FIFO 3
D5	
1	Default value.
1	Select BRAM_FIFO 2
0	De-select BRAM_FIFO 2
D4	
1	Default value.
1	Select BRAM_FIFO 1
0	De-select BRAM_FIFO 1

***See Appendix B for a table of register 4 values suitable for capturing data from ADCs, with respect to register 2.*

Table 7 UART Register 5

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
5	**<DYRST> Reset IOB Delay**	**<DYINC> Inc/Dec IOB Delay**	**<DYVAL> Inc/Dec value (0-63)**					

This register is used only in the extended FPGA design for parallel output ADCs.

D5 DOWNTO D0	**<DYVAL>**	**Delay Value**
0x00-0x3F	Contains the value by which the tap pointer of the selected IOB is to be incremented/decremented, depending on <DYINC>.	
0x00	Default value.	

D6	**<DYINC>**	**Increment/Decrement Delay**
0	Decrements tap pointer of selected IOB <DYVAL>.	
1	Increments tap pointer of selected IOB by <DYVAL>.	
0	Default value.	

D7	**<DYRST>**	**Delay Value Reset To Default**
0	Normal Operation.	
1	Resets the tap pointer of selected IOB to default value.	
0	Default value.	

Table 8 UART Register 6

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
6	**<UBAUD> UART Baud Rate**							

D7 DOWNTO D0	**<UBAUD>**	**UART Clock Divider Register**
0x00-0xFF	Contains the dividing factor to generate UART baud clock from system clock. The value is calculated by (ROUND(SYS_CLK /(16*REQ_BAUD_RATE)) – 1). The SYS_CLK is 250 MHz in extended FPGA design for parallel output ADCs and is 200 MHz in extended FPGA design for serial output ADCs.	
0x87/0x6C	The default value for this register is set to achieve a baud rate of 115200 bps and is different in both extended designs.	

Table 9 UART Register 7

Register Address in Hex	Register Functions							
A3-A0	D7	D6	D5	D4	D3	D2	D1	D0
7	X				<CTS> UART	<DSR> UART	<DCD> UART	<RI> UART

D0	**<RI>**	**Modem Control Signal**
1	Drives 1 on UART modem control signal RI.	
0	Drives 0 on UART modem control signal RI.	
1	Default value.	

D1	**<DCD>**	**Modem Control Signal**
1	Drives 1 on UART modem control signal DCD.	
0	Drives 0 on UART modem control signal DCD.	
1	Default value.	

D2	**<DSR>**	**Modem Control Signal**
1	Drives 1 on UART modem control signal DSR.	
0	Drives 0 on UART modem control signal DSR.	
1	Default value.	

D3	**<CTS>**	**Modem Control Signal**
1	Drives 1 on UART modem control signal CTS.	
0	Drives 0 on UART modem control signal CTS.	
1	Default value.	

Table 10 UART Register 8

Register Address in Hex	Register Functions							
A3-A0	D7	D6	D5	D4	D3	D2	D1	D0
8	<SPIGO> SPI Start	<RCE> SPI Receive Clock Edge	<TCE> SPI Transmit Clock Edge	<SPIBITS> Can be set to 8, 16 or 24				

D4 DOWNTO D0	**<SPIBITS>**	**SPI Frame Size**
0x08	SPI frame size is 8 bits, including all data and address bits.	
0x10	SPI frame size is 16 bits, including all data and address bits.	
0x18	SPI frame size is 24 bits, including all data and address bits.	
0x00	Default value	

D5	**<TCE>**	**Transmit Clock Edge**
1	Transmits data to SPI slave at positive edge of SPI clock.	

0	Transmits data to SPI slave at negative edge of SPI clock.
0	Default value.

D6	**<RCE>** **Receive Clock Edge**
1	Receives data from SPI slave at positive edge of SPI clock.
0	Receives data from SPI slave at negative edge of SPI clock.
0	Default value.

D7	**<SPIGO>** **Initiate SPI Transaction**
1	Initiates an SPI transaction. The bits self clears to 0 when the transaction is complete.
0	Default value.

Table 11 UART Register 9

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
9	**<SBAUD> SPI Baud Rate**							

D7 DOWNTO D0	**<SBAUD>** **SPI Clock Divider Register**
0x00-0xFF	Contains the dividing factor to generate SPI baud clock from system clock.
0x63	Default value.

Table 12 UART Register 10

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
A	**<SPID[07:00]> SPI Transmit data[07:00]**							

D7 DOWNTO D0	**<SPID[07:00]>** **SPI Frame data**
0x00-0xFF	These are the 8 LSBs of the SPI frame that is to be transferred to SPI slave device.

Table 13 UART register 11

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
B	**<SPID[15:08]> SPI Transmit data[15:08]**							

D7 DOWNTO D0 **<SPID[15:08]>** **SPI Frame data**

0x00-0xFF	These are the next 8 bits of the SPI frame that is to be transferred to SPI slave device.

Table 14 UART Register 12

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
C	**<SPID[23:16]> SPI Transmit data[23:16]**							

D7 DOWNTO D0	**<SPID[23:16]>** **SPI Frame data**
0x00-0xFF	These are the last 8 bits of the SPI frame that is to be transferred to SPI slave device.

Table 15 UART Register 13

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
D	**<DQ7> Select data line 7 for Delay Adjust**	**<DQ6> Select data line 6 for Delay Adjust**	**<DQ5> Select data line 5 for Delay Adjust**	**<DQ4> Select data line 4 for Delay Adjust**	**<DQ3> Select data line 3 for Delay Adjust**	**<DQ2> Select data line 2 for Delay Adjust**	**<DQ1> Select data line 1 for Delay Adjust**	**<DQ0> Select data line 0 for Delay Adjust**

This register is used only in the extended FPGA design for parallel output ADCs.

D0	**<DQ0>** **Select ADC Bit 0**
1	Selects bit 0 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D1	**<DQ1>** **Select ADC Bit 1**
1	Selects bit 1 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D2	**<DQ2>** **Select ADC Bit 2**
1	Selects bit 2 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D3	**<DQ3>** **Select ADC Bit 3**

1	Selects bit 3 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D4	**<DQ4>** **Select ADC Bit 4**
1	Selects bit 4 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D5	**<DQ5>** **Select ADC Bit 5**
1	Selects bit 5 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D6	**<DQ6>** **Select ADC Bit 6**
1	Selects bit 6 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D7	**<DQ7>** **Select ADC Bit 7**
1	Selects bit 7 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

Table 16 UART Register 14

Register Address in Hex	Register Functions							
A3-A0	D7	D6	D5	D4	D3	D2	D1	D0
E	X	X	<DQ13> Select data line 13 for Delay Adjust	<DQ12> Select data line 12 for Delay Adjust	<DQ11> Select data line 11 for Delay Adjust	<DQ10> Select data line 10 for Delay Adjust	<DQ9> Select data line 9 for Delay Adjust	<DQ8> Select data line 8 for Delay Adjust

This register is used only in the extended FPGA design for parallel output ADCs.

D0	**<DQ8>** **Select ADC Bit 8**
1	Selects bit 8 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D1	**<DQ9>** **Select ADC Bit 9**

1	Selects bit 9 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D2 **<DQ10>** **Select ADC Bit 10**

1	Selects bit 10 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D3 **<DQ11>** **Select ADC Bit 11**

1	Selects bit 11 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D4 **<DQ12>** **Select ADC Bit 12**

1	Selects bit 12 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

D5 **<DQ13>** **Select ADC Bit 13**

1	Selects bit 13 of the parallel data coming from ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

Table 17 UART Register 15

Register Address in Hex	**Register Functions**							
A3-A0	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
F	**X**							**<DQS> Select ADC Clock for Delay Adjust**

This register is used only in the extended FPGA design for parallel output ADCs.

D0 **<DQS>** **Select ADC Clock**

1	Selects DRY signal coming from the ADC for skew adjust. The value of skew adjust is dictated by UART register 5.
0	Default value.

Section V

FILE STRUCTURE FOR EXTENDED DESIGN I

- AppsToplevel.vhd
  - evbif.v
    - evbif_spi.v
      - spi_clkgen.v
      - spi_shift.v
    - evbif_obufdes.v
      - obufdes_safifo.v
        - bram16x36.v
    - evbif_uart.v
      - uart_baudgen.v
      - uart_tx.v
        - kcuart_tx.v
      - uart_rx.v
        - kcuart_rx.v
    - evbif_ctrl.v
    - evbif_dumpmem.v
      - dumpmem_blk.v
        - fifo32b.v
        - fifo16kb.v

FILE STRUCTURE FOR EXTENDED DESIGN II

- ads5463_toplevel.v
  - ads5463_ddrif.v
    - ddrif_infrastructure.v
      - clocks_sysdcm.v
    - ddrif_idelay_ctrl.v
    - ddrif_top_0.v
      - ddrif_iobs_0.v
        - ddrif_data_path_iobs_0.v
  - evbif.v
    - evbif_spi.v
      - spi_clkgen.v
      - spi_shift.v
    - evbif_obufdes.v
      - obufdes_safifo.v
        - bram16x36.v
    - evbif_uart.v
      - uart_baudgen.v
      - uart_tx.v
        - kcuart_tx.v
      - uart_rx.v
        - kcuart_rx.v
    - evbif_ctrl.v

- evbif_dumpmem.v
    - dumpmem_blk.v
        - fifo32b.v
        - fifo16kb.v

Appendix A

Example MATLAB function for Initializing a UART port

```
function s = InitUART(baud);
s = serial('COM9', 'BaudRate', baud);
s.InputBufferSize = 131400;
s.ReadAsyncMode = 'continuous';
s.BytesAvailableFcnCount = ((8*1024-1)*2+16-1)*4*2;
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcn = @instrcallback;
fopen(s);
get(s,{'BaudRate','DataBits','Parity','StopBits','Terminator'})
end
```

Example MATLAB function for Capturing Data from UART port

```
function CaptureData(chnl_cnfg, s)
a = uint8(chnl_cnfg);      % chnl_cnfg Є { 1, 2, 4, 8, 3, 12, 15}
b = uint8(255);
fwrite(s,[0 b 0 0 2 a 1 1]);
end
```

Example MATLAB function for Reading Data from UART port

```
function ReadUARTBuffer(filename,s)
fid1 = fopen(filename,'w');
in = fread(s,s.BytesAvailable/2,'uint16');
out = swapbytes(uint16(in));
fprintf(fid1,'%x\n',out);
fclose(fid1);
end
```

Appendix B

Register 2 = 0xF (Multiple SDR Streams/Using XAPP866)

Register 4 =

xxxx_0001	2K samples (4KB) from each selected blocks
xxxx_0010	4K samples (8KB) from each selected blocks
xxxx_0100	8K samples (16KB) from each selected blocks
xxxx_1000	(16K + 13) samples (32KB + 26) from each selected blocks

Register 2 = 0x3/0xC (DDR ADC)

Register 4 =

0101_0001	2K samples (4KB) each from block 1 and 3
0101_0010	4K samples (8KB) each from block 1 and 3
0101_0100	8K samples (16KB) each from block 1 and 3
0101_1000	(16K+13) samples (32KB + 26) each from block 1 and 3
1111_1000	(16K+13) samples (32KB + 26) each from block 1,2,3 and 4

Register 2 = 0x1/0x2/0x4/0x8 (SDR)

Register 4 =

0001_0001	2K samples (4KB) from block 1
0001_0010	4K samples (8KB) from block 1
0001_0100	8K samples (16KB) from block 1
0001_1000	(16K+13) samples (32KB + 26) from block 1
0011_1000	(16K+13) samples (32KB + 26) each from block 1 and 2
0111_1000	(16K+13) samples (32KB + 26) each from block 1,2 and 3
1111_1000	(16K+13) samples (32KB + 26) each from block 1,2,3 and 4