VME-DDS Frequency and Phase Control Module, Part I

Module Design Manual

Draft: Sections I and II and III

September 8, 2010
Revised October 15, 2010
Revised July 15, 2011

Craig Drennan

Table of Contents

Revision History

October 15, 2010 – Updated the changes made to the Inter-FPGA interface that were made to be able to route the ADC serial data directly to the Upper FPGA.

July 15, 2011 – Updated the section on the AD7625 analog to digital converter. Corrected the description of how the output strobe is generated and added a timing diagram for the interface.

I. Introduction

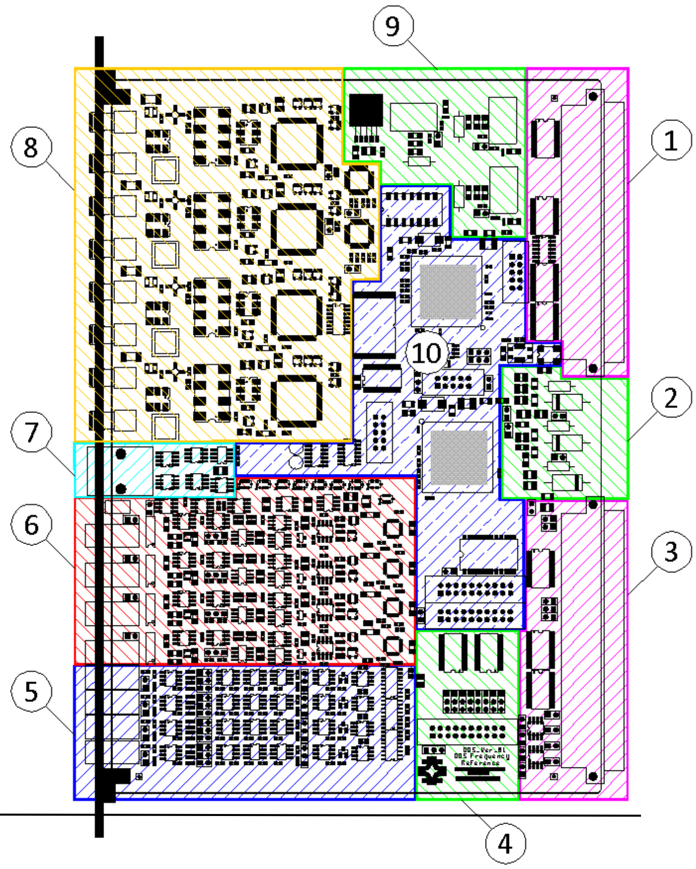
The VME-DDS Module is being developed to replace many of the older electronics in the Fermi Booster Low Level RF (LLRF) control system. This system controls the frequency and phase of a set of radio frequency sine wave reference signals for the High Level RF system controlling the high voltage field in the Booster particle accelerating cavities. The frequency of the signal is ramped from approximately 37.7 MHz to 52.8 MHz over the 36 millisecond acceleration interval. The phase of the reference signal is manipulated during the initial beam capture at the beginning of the acceleration interval and “beam transition” near the middle of the interval. Phase control is also essential in controlling the radial position of the beam in the beam pipe and also phase matching the beam to the frequency reference of the Main Injector accelerator just before extraction.

This document is written to capture the design details of the new frequency and phase reference electronics. An attempt is made to describe how the various functions and subsystems have been implemented and provide specifications so that these functions and subsystems can be troubleshot, upgraded, and extended.

II. General Descriptions and Block Diagrams

The VME-DDS Module is illustrated in Figure II.1 with the major hardware sections highlighted and labeled. The document sub-sections that follow provide brief descriptions of the hardware and provide a few details on its use. There are diagrams that define interface connector pin-outs and that show how to set jumpers for different analog input and output ranges. Further information is intended to provide some perspective and scope.

1. Upper VME Interface.
2. VME Power Input Fuses and Filtering.
3. Lower VME and Rear Transition Card Interface.
4. Eight Digital IO Connections.
5. Four Analog Outputs.
6. Four Analog Inputs.
7. High Speed Serial Inter-Module Interface.
8. Four DDS RF Sine Wave Outputs.
9. Voltage Regulation.
10. Memory and Programmable Logic.



II.1 VME Interface

Along with the Lower VME Interface, the module behaves as a VME Slave device. It will respond to standard supervisory or non-privileged accesses using address lines A[23..2] or extended supervisory or non-privileged accesses using address lines A[31..2]. In either case the upper four address bits are compared to DIP switch settings on the module acting as module selection logic. Data transfers can be either 2 Byte (16 bit) or 4 Byte (32 bit). VME data transfer logic on the module typically responds with the data acknowledge signal (DTACK) within 40 nanoseconds of the data strobe (DS) transfer request. However, completion of the VME transfer may be “held-off” due to other processes using the memory or internal transfer busses. The internal processes are design to avoid holding off the VME transfer for more than 64 us, and typically will not hold-off VME transfers for more than 2 us.

Power supply voltages supplied by the VME bus are fused at the input to the module; 3 Amps for the +5V input, 1 Amp for the +12V and 1 Amp for the -12V.

II.2 Rear Transition Card Interface

The user defined pins of the VME J2/P2 connection have been defined for specific use, but have been buffered using tristate-able transceivers to maintain some degree of flexibility. Table II.2.1, Table II.2.2 and Table II.2.3 map out the pin and buffer configuration of the J2. The table also lists the pin names (function) with regard to the Rear VME Triple Phase Detector module. The significance of the Buffer Group, listed in the tables, is that all signals in a particular group will be either driven or not and have the same transceiver direction.

II.2.1 Rear VME Triple Phase Detector Interface

The Rear VME Triple Phase Detector was, as the name implies, designed as a VME rear transition card (80 mm deep). The module has also been laid out so it can also be mounted in a NIM Module. The module provides three RF phase detectors based on the Analog Devices AD8302. There is also an AD8307 Logarithmic Amplifier / RSSI device for measuring the Booster Beam intensity and a threshold comparator to provide a Beam Gate signal.

The module has four general purpose digital inputs and four general purpose digital outputs. There are amplifiers to buffer the phase detector and RSSI analog signals through the J2/P2 connection to the DDS module or to the faceplate of the transition card (rear of the crate). The gains and offsets of the analog phase detector output signals can be adjusted using digital potentiometers in the circuit. The RSSI, Beam Gate threshold is also adjusted with a digital pot. The J2/P2 interface includes signals that allow these digital pots to be manipulated through, or by, the DDS module. There is also a connection through which external switches and a rotary encoder can manipulate the digital pots through logic in the CPLD.

Finally, the Rear VME card provides a USB-FIFO interface. The USB side can be connected to a PC computer. Drivers for Windows and Linux are available. The 8 bit FIFO side can be connected to the DDS module through the J2/P2 connection.

II.2.2 Compatibility with BLM System Cards

An attempt was made to make the J2/P2 configuration compatible with the “Control Bus” of the Beam Loss Monitor Digitizer and BLM system Controller cards. The thought was that it would be convenient if the DDS module could share a crate with the BLM Digitizer cards, and that it might be interesting if the BLM controller card could be used with the DDS module in some application. However, in the final configuration of the first DDS module prototype there are some conflicts. For the DDS module to work with the BLM Digitizers, the Digitizers would need to be programmed to tri-state the “Error” signal at pin A23 and hold inactive the “Abort_CS/” signal. Both of these signals are outputs of the lower FPGA on the Digitizer module which could be reprogrammed for “Booster” BLM applications. To be compatible with the BLM Controller card the DDS module opamps U150 and U159 would have to be removed from the DDS module and the Controller card “BusAck” signal on pin C26 would have to be tri-stated. The DDS module is not compatible on the Control Bus backplane with BLM system Timing cards or Abort cards. A rear transition card could not be powered from the DDS if used with a Control Bus Backplane.

Table II.2.1 VME-DDS Module J2/P2 Row A Signals

Pin	DDS Module Name	Buffer Group	Rear VME Phase Detector Name	Rear VME Phase Detector Signal Description
A1	CB-A[0]	J2D	USB_TXE_n (input)	USB interface Transmitter Empty
A2	CB-A[1]	J2D	USB_RXF_n (input)	USB interface Receiver Full

A3	CB-A[2]	J2D	BEAM_GATE (input)	Booster Beam Intensity Threshold Gate
A4	CB-A[3]	J2D	DIN_1_J2 (input)	General purpose Digital Input
A5	CB-A[4]	J2D	DIN_2_J2 (input)	General purpose Digital Input
A6	CB-A[5]	J2D	--	
A7	CB-A[6]	J2D	--	
A8	CB-A[7]	J2D	--	
A9	CB-A[8]	J2F	DPOT_A0 (output)	Digital Pot Address Bit 0
A10	CB-A[9]	J2F	DPOT_A1 (output)	Digital Pot Address Bit 1
A11	CB-A[10]	J2F	DPOT_A2 (output)	Digital Pot Address Bit 2
A12	CB-A[11]	J2F	DPOT_U_nD (output)	Digital Pot Up / Down bit
A13	CB-A[12]	J2F	J2_DOUT_1 (output)	General purpose Digital Output
A14	GND		GND	
A15	CB_MREQ_n	J2F	J2_DOUT_2 (output)	General purpose Digital Output
A16	CB_MEMRD_n	J2F	USB_RD_n (output)	USB FIFO interface Read bit
A17	CB_WR_n	J2F	USB_WR_n (output)	USB FIFO interface Write bit
A18	GND		GND	
A19	CB_D[0]	J2E	USB_D[0] (bidir)	USB FIFO interface data bit
A20	CB_D[1]	J2E	USB_D[1] (bidir)	USB FIFO interface data bit
A21	CB_D[2]	J2E	USB_D[2] (bidir)	USB FIFO interface data bit
A22	CB_D[3]	J2E	USB_D[3] (bidir)	USB FIFO interface data bit
A23	CB_D[4]	J2E	USB_D[4] (bidir)	USB FIFO interface data bit
A24	CB_D[5]	J2E	USB_D[5] (bidir)	USB FIFO interface data bit
A25	CB_D[6]	J2E	USB_D[6] (bidir)	USB FIFO interface data bit
A26	CB_D[7]	J2E	USB_D[7] (bidir)	USB FIFO interface data bit
A27	AGND		AGND	
A28	REAR_AO_1		Analog_Rear_1 (output)	Analog output to rear card connector
A29	REAR_AO_2		Analog_Rear_2 (output)	Analog output to rear card connector
A30	REAR_AO_3		--	
A31	REAR_AO_4		--	
A32	AGND		AGND	

Table II.2.2 VME-DDS Module J2/P2 Row B Signals (VME defined pins)

Pin	Name	Buffer Group
B1	+5V_IN_J2	
B2	GND	
B3		
B4	VME_A[24]	J2C
B5	VME_A[25]	J2C
B6	VME_A[26]	J2C
B7	VME_A[27]	J2C

B8	VME_A[28]	J2C
B9	VME_A[29]	J2C
B10	VME_A[30]	J2C
B11	VME_A[31]	J2C
B12	GND	
B13	+5V_IN_J2	
B14	VME_D[16]	J2A
B15	VME_D[17]	J2A
B16	VME_D[18]	J2A
B17	VME_D[19]	J2A
B18	VME_D[20]	J2A
B19	VME_D[21]	J2A
B20	VME_D[22]	J2A
B21	VME_D[23]	J2A
B22	GND	
B23	VME_D[24]	J2A
B24	VME_D[25]	J2A
B25	VME_D[26]	J2A
B26	VME_D[27]	J2A
B27	VME_D[28]	J2A
B28	VME_D[29]	J2A
B29	VME_D[30]	J2A
B30	VME_D[31]	J2A
B31	GND	
B32	+5V_IN_J2	

Table II.2.3 VME-DDS Module J2/P2 Row C Signals

Pin	DDS Module Name	Rear VME Phase Detector Name	Rear VME Phase Detector Signal Description
C1			
C2	+5V	+5V_IN_J2	+5V power supply to rear module
C3			
C4	GND	GND	
C5	GND	GND	
C6	GND	GND	
C7	GND	GND	
C8	GND	GND	
C9	GND	GND	
C10			
C11	+12V	+12V_IN_J2	+12V power supply to rear module
C12			
C13	-12V	-12V_IN_J2	-12V power supply to rear module
C14			
C15	+3.3V		
C16			
C17			
C18	GND	GND	
C19			
C20	+2.5V		
C21			
C22			
C23			
C24	GND	GND	
C25	REAR_AI_1		
C26	AGND	AGND	
C27	REAR_AI_2	PHASE_DIFF_1	Phase Detector 1 analog output signal
C28	AGND	AGND	
C29	REAR_AI_3	PHASE_DIFF_2	Phase Detector 2 analog output signal
C30	AGND	AGND	
C31	REAR_AI_4	PHASE_DIFF_3	Phase Detector 3 analog output signal
C32	AGND	AGND	

II.3 Digital Inputs and Outputs

The first VME-DDS prototype has 4 digital inputs that can be observed by both the Upper FPGA and the Lower FPGA. There are 4 digital outputs available. Either all four digital outputs are controlled by the Upper FPGA or all four are controlled by the Lower FPGA. There are also two digital inputs and two digital outputs available to the Lower FPGA through the Rear VME Triple Phase Detector transition module connected to VME connector J2/P2. There are three-way jumpers on each of the eight digital signals that can be set to have the signal pulled up through 10k Ohms, pulled down through 50 Ohms, or not pulled up or down (not terminated). The digital inputs and outputs are connected to the board at J86. Figure II.3.1 provides some details of the connection and termination. Table II.3.1 gives the pin out for J86. A ribbon cable connects the digital TTL level inputs and outputs with the front panel as shown in Figure II.3.2.

Table II.3.1

Pin	Signal
1	--
3	--
5	Digital Input 1
7	Digital Input 2
9	Digital Input 3
11	Digital Input 4
13	Digital Output 1
15	Digital Output 2
17	Digital Output 3
19	Digital Output 4
All Even Pins	Ground

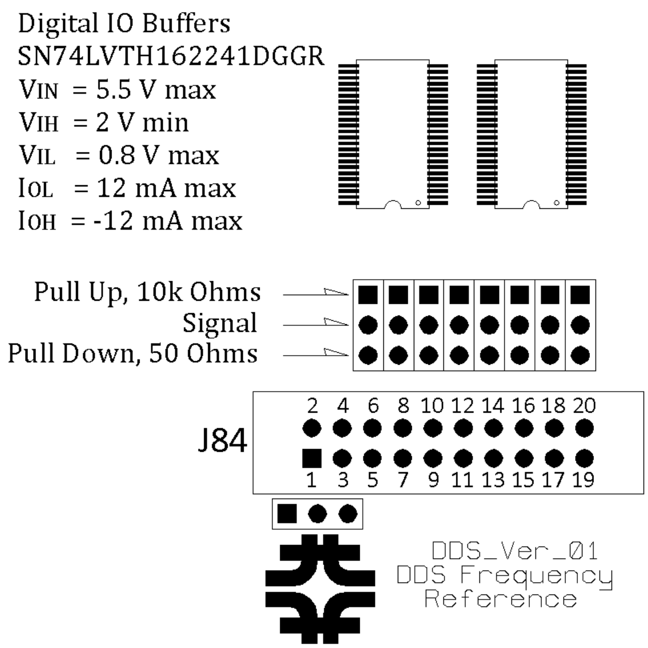


Figure II.3.1 Connection, termination and buffers for the digital inputs and outputs.

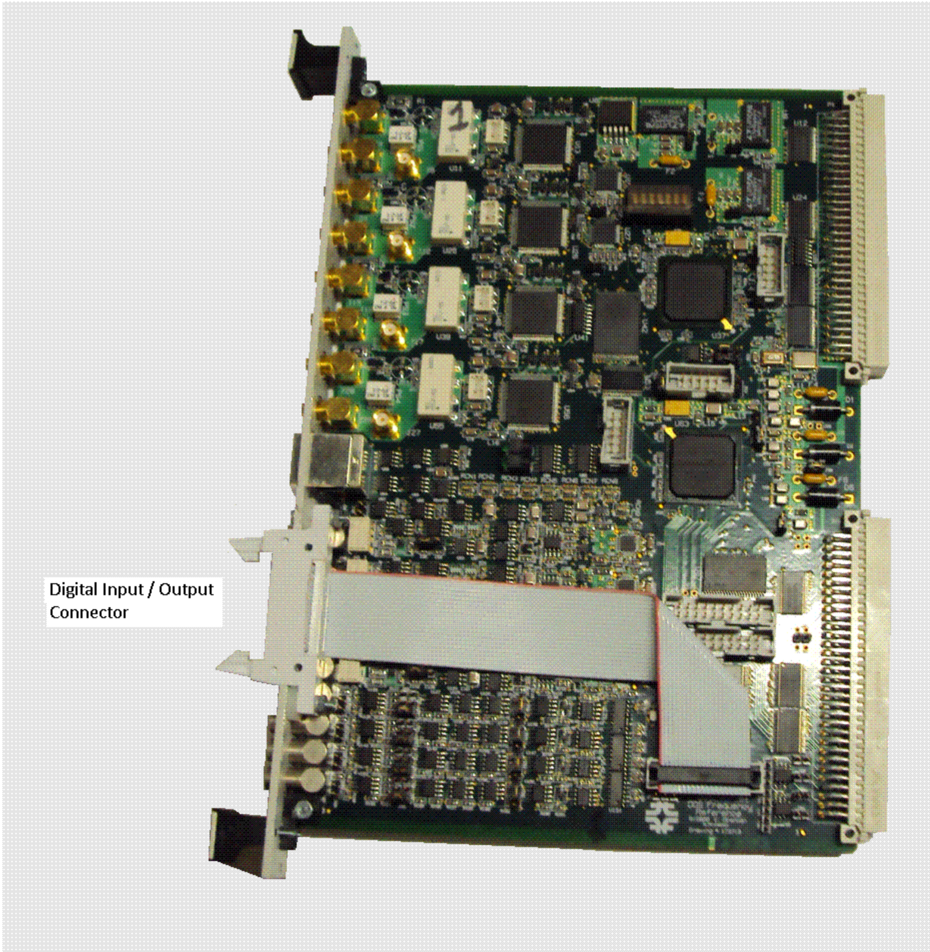


Figure II.3.2 Photo showing the digital IO connection between the board and the front panel.

II.4 Analog Outputs

Four analog outputs are available on the DDS module. Jumpers are available to select different output voltage ranges. Figure II.4.1 indicates the jumper settings for the different ranges. The digital to analog converters that set the voltage outputs have a 16 bit resolution, and are programmed through the Lower FPGA. Each output can be updated every 400 ns. These analog output circuits use digital potentiometers to adjust the outputs' zero and span, the circuits offset and gain. Details on manipulating the digital pots are in another section.

The analog outputs should be able to drive +/- 10 Volts into a load of at least 300 Ohms.

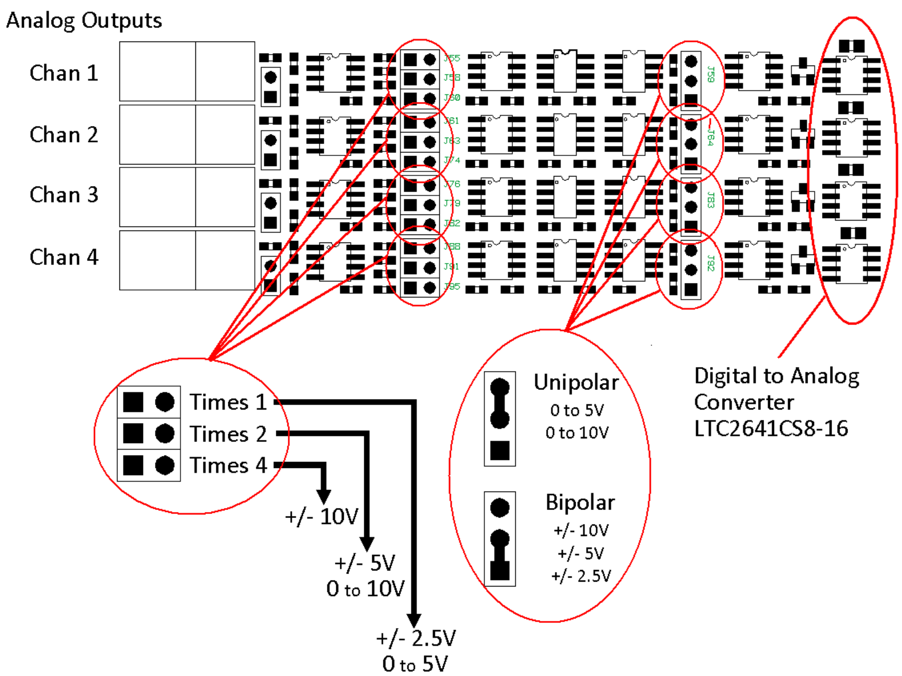


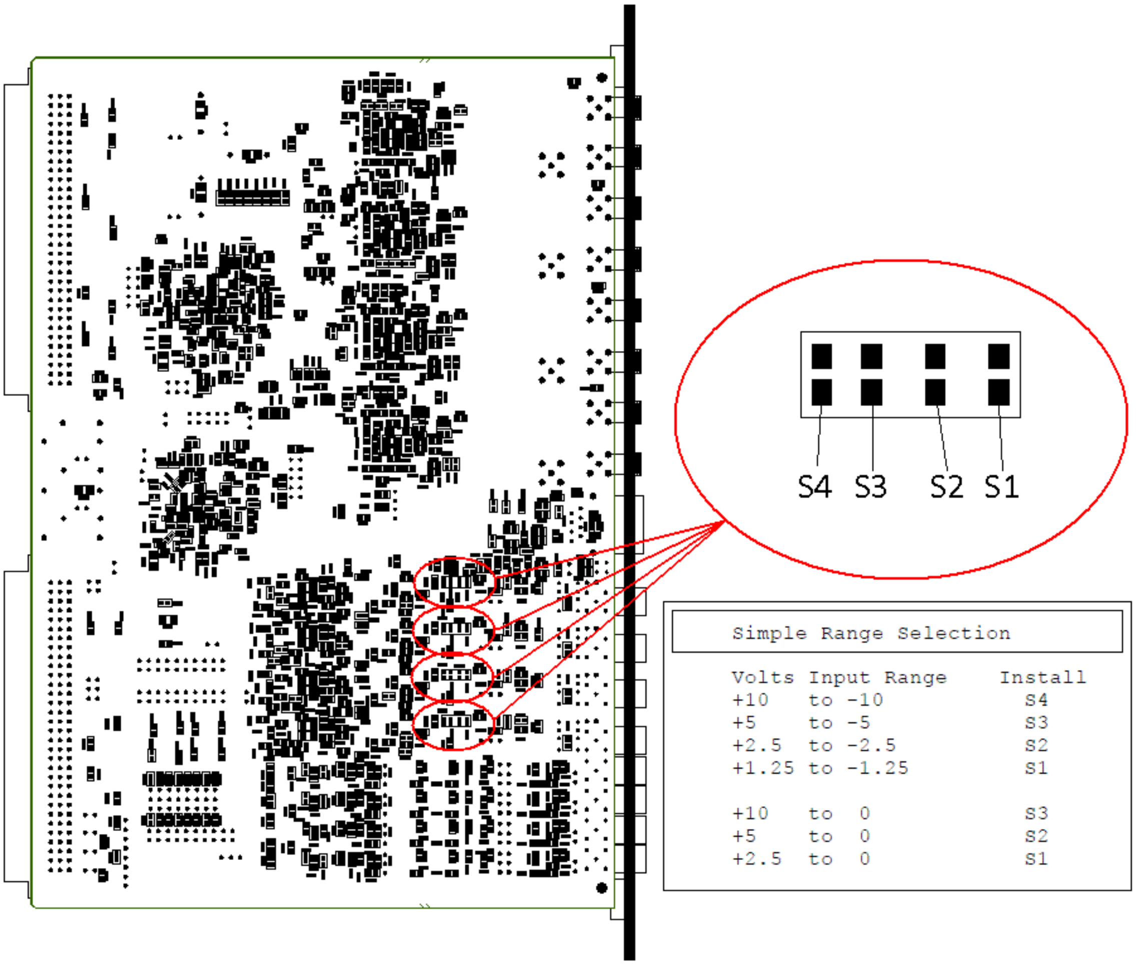
Figure II.4.1 Digital to analog converter circuit jumper settings.

II.5 Analog Inputs

Four analog inputs are available on the DDS module. The inputs can be setup for 4 bipolar voltage ranges and 3 unipolar ranges. Figure II.4.2 indicates how to set the jumpers for the different ranges. The base resolution of the analog to digital converters is 15 bits. Calibration over a +/- 10 Volt input range has shown approximately 20 counts RMS error between the sample mean and the expected value for a given voltage. The analog to digital converters are read constantly at a rate of 5 Msps. The input bandwidth of each input is approximately 1 MHz.

The ADC's are Analog Devices AD7625 SAR type converters. They are not pipelined, hence no internal latency. There is an overall processing latency in getting the data due to the 200 nanoseconds it takes to serially deliver the data to the FPGA. The serial link is LVDS so its noise contribution is very small.

In order to setup a particular input voltage range a 0 Ohm resistor, or a short, needs to be install in the appropriate location on the back side of the module as illustrated in Figure II.5.1. Also three position jumpers on the front side of the module must be set as shown in Figure II.5.2.



Simple Range Selection

Volts Input Range			Install
+10	to	-10	S4
+5	to	-5	S3
+2.5	to	-2.5	S2
+1.25	to	-1.25	S1
+10	to	0	S3
+5	to	0	S2
+2.5	to	0	S1

Figure II.5.1 Analog input range selection, placement of 0 Ohm resistors on back of module

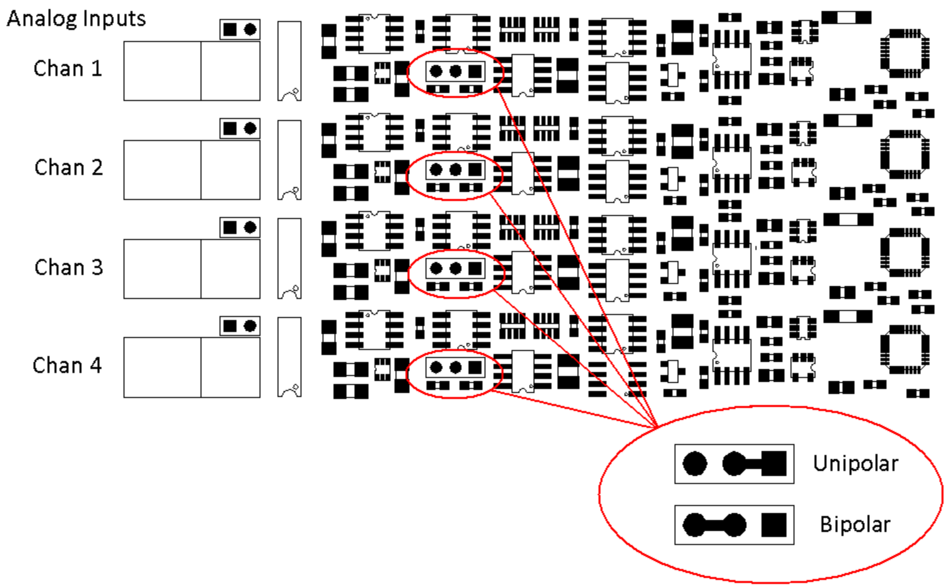


Figure II.5.2 Analog input range Unipolar/Bipolar selection jumpers.

An ADG419 switch is provided at the input to switch in an onboard test voltage for checking the operation of the analog input circuitry. The onboard test voltage also aids in calibrating the inputs. An issue with the switch is that the input diode protection of the switch will conduct current if the input voltage is above the supply rails. The supply rails are +/-12 Volts when the module is powered, but they are zero when the module is not. Voltage applied at the inputs could damage the switch when the module power is off. This concern is addressed by using a normally open relay/switch at each input.

The analog inputs have been designed with a telecom style relay which is opened or close through the Lower FPGA. This is part of the over-voltage protection for the analog inputs. This can be setup to disconnect from the voltage input in the event that an out of range indication be measured by the ADC's. An out of range indication might be caused by a misunderstanding between the user and the established voltage range settings. The relays also protect the inputs when the power on the board is off. Additionally Figure II.5.3 shows the passive components that also provide some protection against input voltages beyond the +/- 12 Volt power supply rails. It is estimated that the inputs will survive +/- 20 Volts continuous on the inputs. The peak short duration voltage has not been tested.

An issue with the first DDS module prototype is that the Analog Devices AD8066AR dual opamps used in the analog input circuits run hotter than necessary and would be excessively hot should the crate fans stop running. A substitute that we intend to use on the next version of the module is the AD8034's. These parts run at half the power and operate with half the slew rate of the AD8066's. The AD8034's slew rate is 80V / us which is still sufficient for our application.

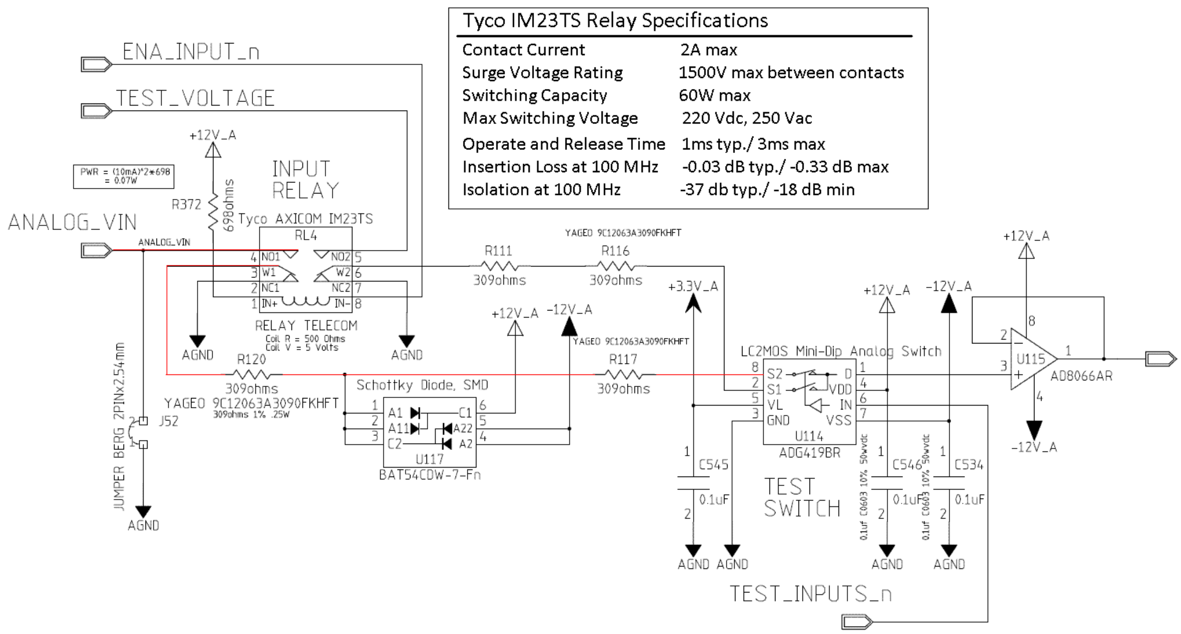


Figure II.5.3 Analog input over-voltage protection.

II.6 High Speed Serial Inter-Module Interface

Full development of the high speed serial inter-module interface is not yet complete. The following is a description of the hardware provisions available on the first prototype and a general system topology for multiple DDS modules to work together to produced up to eight synchronous RF outputs.

II.6.1 Serial Interface Connectors and Cables

The inter-module connector is a Tyco/AMP 558342-1 shielded right-angle PCB 8 position category 5 modular jack. The complicated part is actually the cable and optional connections of the DDS module for providing a pin-to-pin connection between modules, or providing a cross over connection for transmit and receive signals of a bidirectional link between modules. With the cross-over connection both of the DDS modules in a link could be programmed to transmit on the same transmit pin and receive on the same receive pin. The pin-to-pin connection would be appropriate for an inter-module link where there is one DDS module setup to be a master with the other modules setup to be slaves. Figure II.6.1.1 indicates the 0 ohm (short) resistors install for either the cross-over of pin-to-pin configurations. Figure II.6.1.2 provides a pin out for the two types of cables.

A cross-over cable was found for procurement from Digikey. It is a Tyco part no.1435953-2.

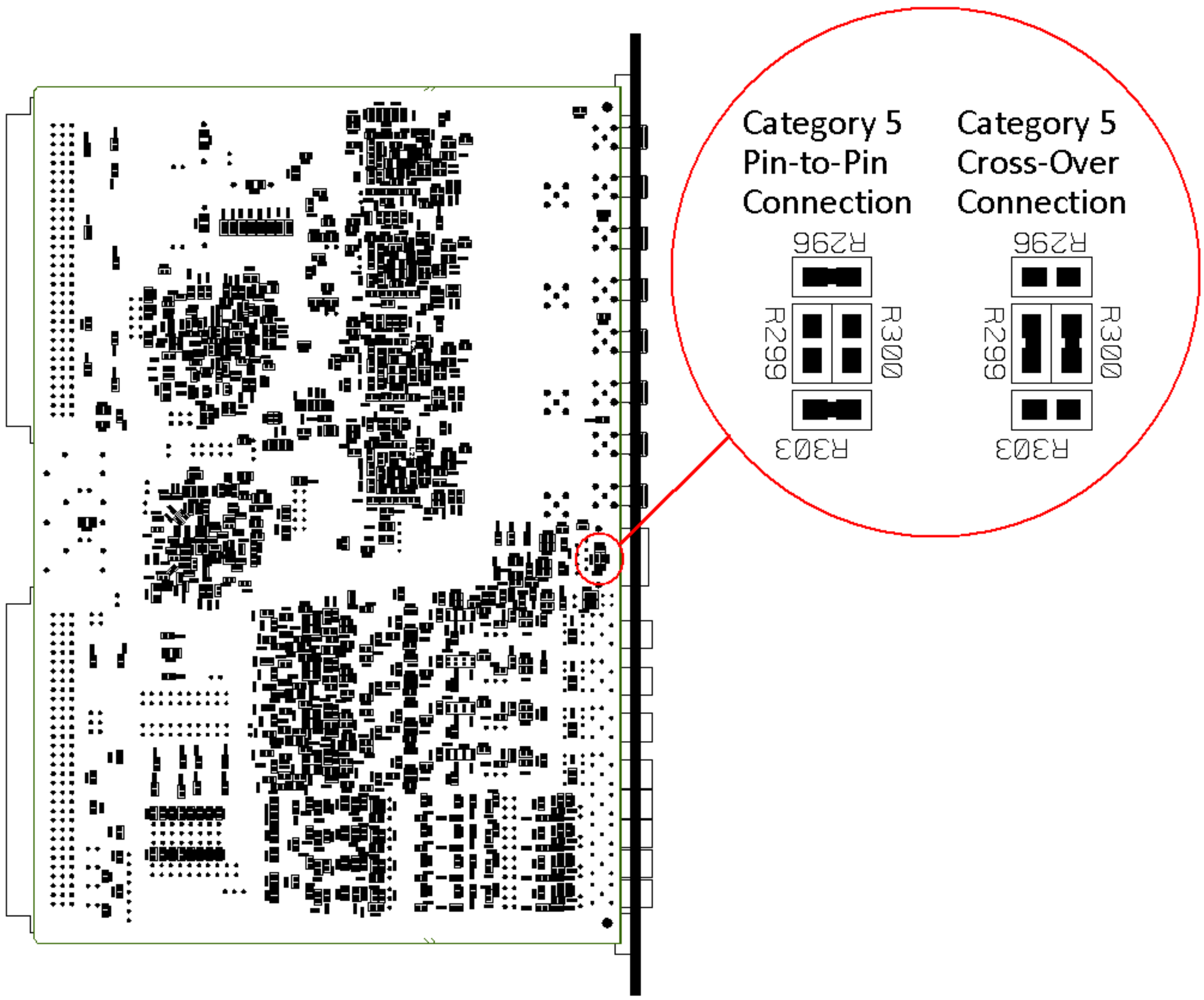


Figure II.6.1 Setup for either the pin-to-pin or cross-over inter-module connection.

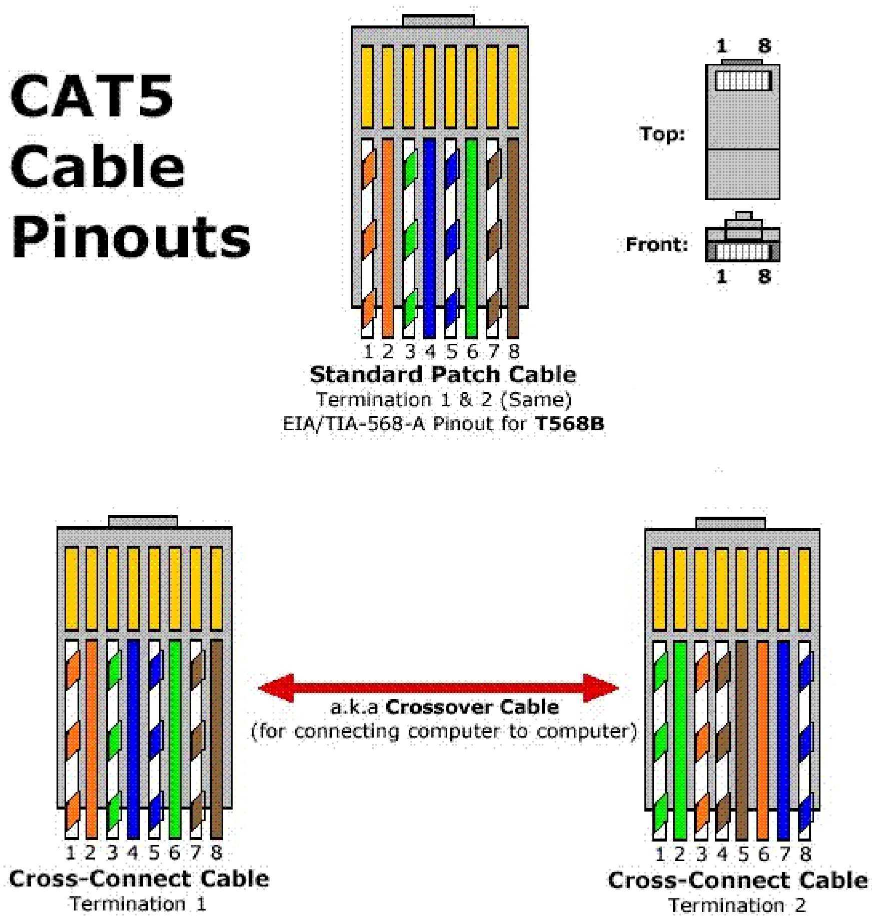


Figure II.6.1.2 Category 5 pin outs for pin-to pin and cross-over cables.

II.6.2 Inter-Module Drivers and Receivers

There are 4 LVDS signals being driven in the inter-module link. There is one dedicated input, one dedicated output, and two that are connected via LVDS to LVTTL transceivers. The dedicated input and output are connected to LVDS IO on the Upper FPGA via SN65LVDS100D LVDS translator/repeaters. The LVDS to LVTTL transceivers are SN65LVDM176D's. Each transceiver is connected to the Upper FPGA by two LVDS signals, one out and one in, via channels in LVDS receiver and transmitter components, DS90LV028AHM and DS90LV027AHM, respectively. Figure II.6.2.1 presents the schematic for the LVDS drivers and receivers.

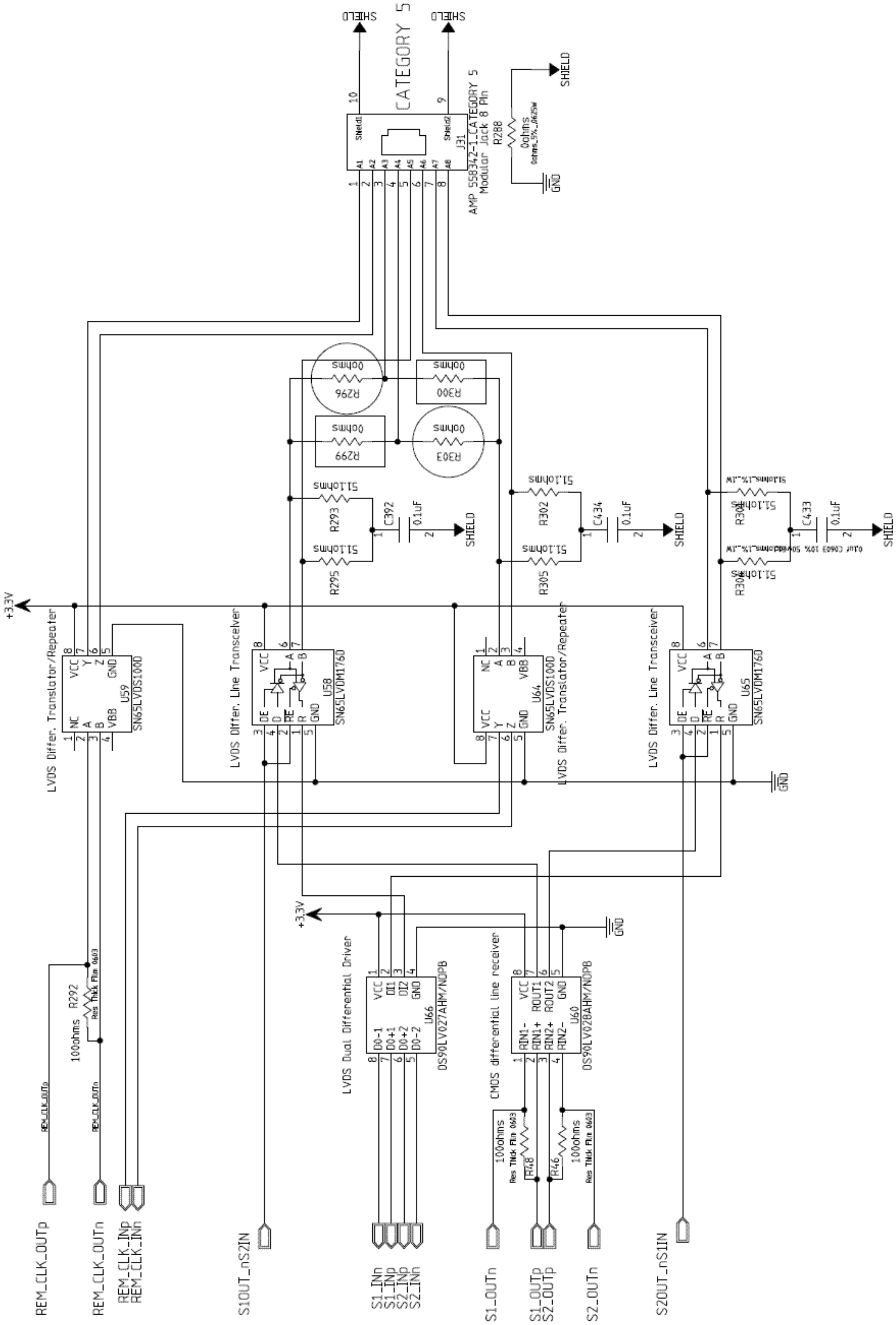


Figure II.6.2.1 The inter-module link driver and receiver circuit.

II.6.3 Inter-Module DDS Synchronization

Figure II.6.3.1 is a block diagram for the proposed inter-module DDS Synchronization utilizing the inter-module link. There are three signals that must be managed to keep all of the DDS integrated circuits in sync. The first is the DDS Reference clock. This is typically going to be multiplied up to a higher clock rate within each DDS component, but we still wish to have minimal phase difference between the clocks presented to each DDS. There is a National Semiconductor LMK01010ISQE programmable clock distribution component that can phase adjust the clocks delivered to the four DDS components on the module. When providing synchronized clocks to DDS components on a second board we will need to phase delay the reference clock on the main board by the amount of interconnection delay to the second. The FPGA PLL that generates the reference clock can be setup to output two clocks with the necessary phase offset. The main board's clock distribution component can also make small adjustments to the phase relationship.

The second signal needed to synchronize multiple DDS components is the DDS Sync signal. Each DDS will generate a DDS Sync Out clock, but only one is used. The DDS Sync is distributed to each DDS, including the part that generated the original.

Each DDS executes a sequence of task internally when generating and updating the stream of digital values that the output DAC will convert into a sinusoidal RF output. This sequence is controlled by an internal state machine. The DDS Sync signals ensure that all of the DDS components' state machines are in the same state at the same time.

The final signal we need to be concerned with is the DDS IO_Update. This signals the DDS when to implement the next frequency and/or phase setting. If the DDS's state machines do not stay in sync or the DDS's do not update the frequency or phase in the same cycle, the resulting RF outputs will quickly lose their proper phase relationships. For the inter-module synchronization the IO_Update signal will have to be timed in relation to the delivery of new Frequency/Phase data across the serial link.

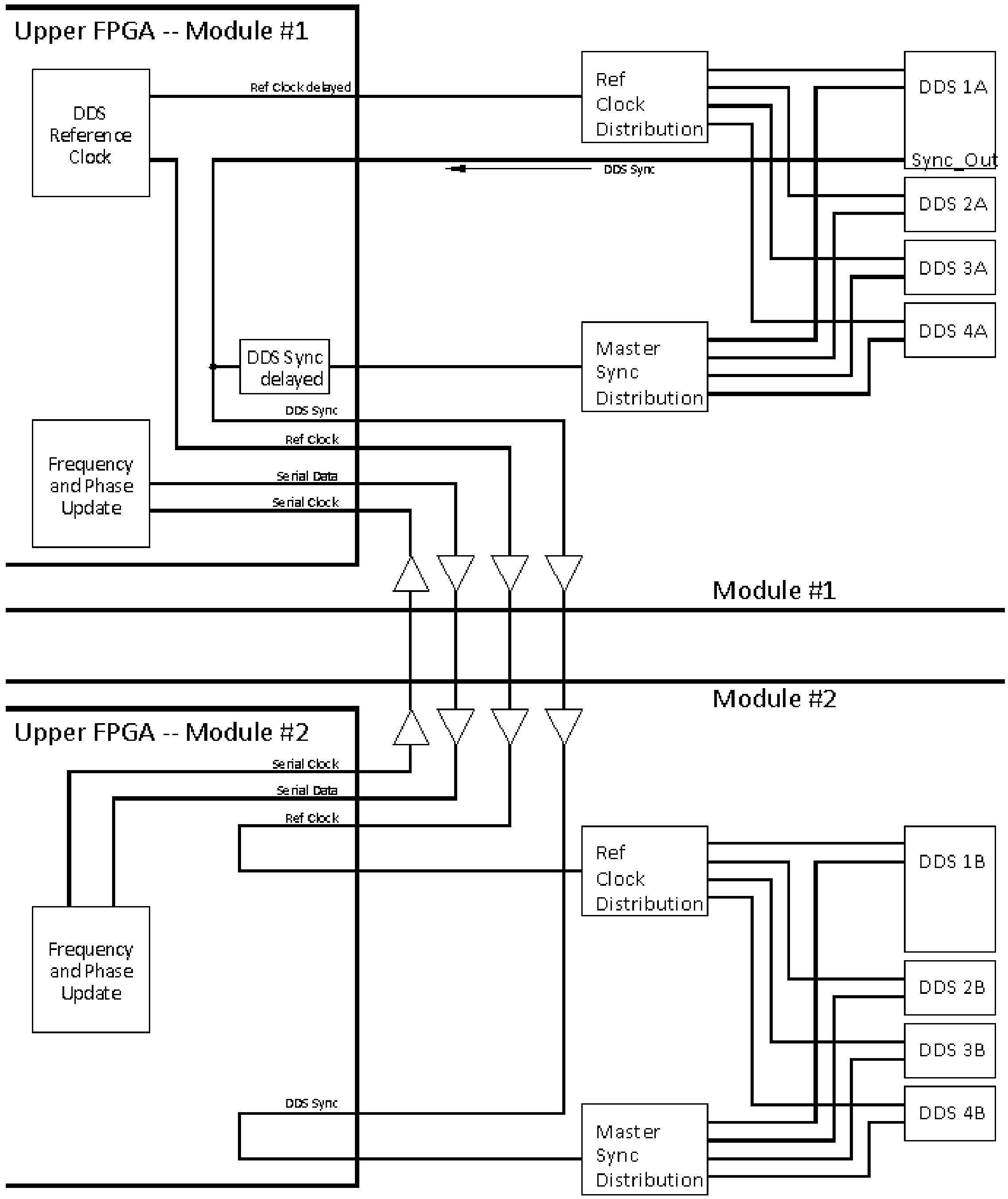


Figure II.6.3.1 Block diagram of the Inter-Module link used to synchronize two DDS modules.

II.7 DDS RF Sine Wave Outputs

The DDS module uses four Analog Devices AD9910BSVZ. These devices are highly programmable and have many different modes of operation. Through early experimentation using two DDS evaluation modules we were able to determine a method for keeping multiple devices in sync and to update each DDS with a new frequency and phase words every micro-second.

The digital data input to the DDS has been limited to going through the serial port to the internal Control Registers. The parallel data interface initially appeared to be an attractive way to rapidly update the frequency and phase. It was found, however, that only a 16 bit frequency offset could be applied through the parallel data interface. Given the desired minimum frequency adjustment we hoped to be able to make and the range of frequencies over which the Booster LLRF had to sweep, 16 bits did not provide sufficient dynamic range.

Using only the serial port we can update both the frequency and phase of the DDS RF output each micro-second. The trick to achieving this update rate is by performing partial loads of the “Single Tone Profile Register” and loading the serial data Least Significant Bit first. LSB first is not the default mode after a DDS Reset, but must be selected during setup. The partial register load writes the 8 bit Instruction Byte, 32 bit Frequency Tuning Word and the 16 bit Phase Offset Word, but skips the remaining 16 bits of the register. Serial data is transferred at 68 Mbps. This is near the limit for the AD9910 serial port. The partial load is always followed by the IO_Update pulse which causes the new data to be loaded, and an IO_Reset which resets the serial interface to accept the next set of bits as the next Instruction Byte. Further details of the setup and control for the DDS components are provided elsewhere in the documentation.

The output of the DDS components needs some form of filtering and amplification before it can be applied to the LLRF system. Filtering has its issues in that the phase shift through the filter will vary as a function of the frequency of our signal. Since we must sweep frequencies and control the signal phase, the filter could add a non-constant phase shift we must deal with. If it becomes necessary, a function can be added to the FPGA computations that compensates for this frequency dependent phase shift in advance of applying each new phase offset word.

Amplification is also necessary and a simple attempt at this was included on the first DDS module prototype. Figure II.7.1 shows the circuit used on each DDS RF output channel and the modifications that were made as a result of tuning the RF outputs to make them work. After tuning we achieved a +10 dBm signal with the harmonic content at least 25 dB below the fundamental frequency.

We are hoping for further improvement on the level of distortion in our RF outputs. The current DDS based LLRF reference maintains its harmonic levels more than 30 dB down over the frequency range. One of the three prototype modules was assembled without the RF amplifier circuit so other amplifiers might be tried.

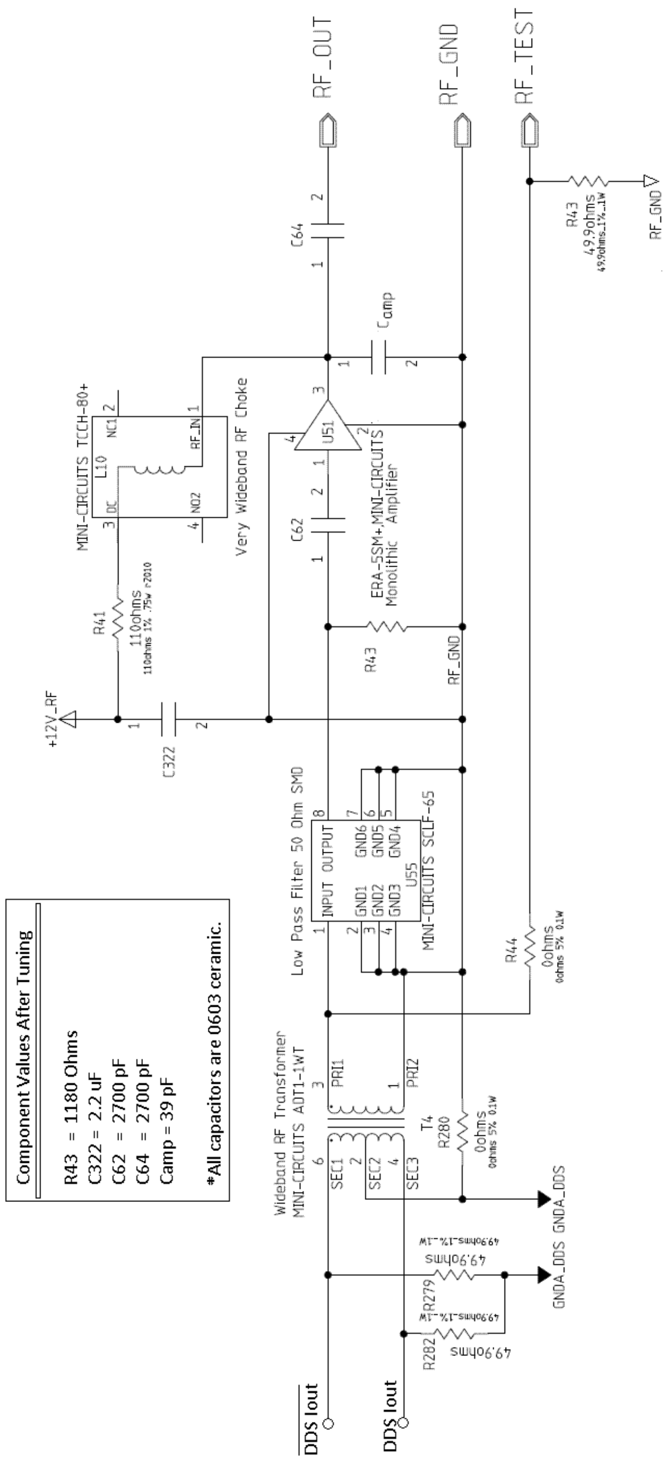


Figure II.7.1 DDS RF output filter and amplifier circuit.

II.8 Voltage Supply and Regulation

Supplying the necessary chip voltages has become a complex endeavor. Table II.8.1 lists the voltages required to power or bias the majority of components on the module. Many of the voltages listed are simply filtered versions of another power supply. Filtering consists of a series ferrite chip inductor separating power planes connected solely at this inductor. Power planes on either side of the inductor are bypassed with several capacitors to an associated ground plane. Ground planes are typically connected at a single point near the filter inductor or directly beneath each mixed signal component such as an ADC or the DDS IC. Many secondary regulators were used to help decouple noise to and from different components on the module. Most of these regulators are very low power and are very small in size (SOT packages). Figure II.8.1 provides some information on how the different voltage planes are arranged and the voltages derived. The diagram illustrates the “family tree”, so to speak, of the origins of each separate voltage used on the board. The bold (green) rectangular groupings indicate which ground plane each voltage is decoupled to.

Table II.8.1 Power supply voltages.

Voltage	Powered Components	Regulator	VME Power Source
+5V	LED Driver, SN7406D VME Open Collector Buffers, SN74S38D	Tied direct to VME	+5 V
+3.3V	FPGA's, Altera EP3C40F484C8N FPGA Config Memory, EPCS16SJ8N Flash Memory, JS28F256P33B95 Clock Dist., LMK01010ISQE, qty 2 DDS, AD9910BSVZ, qty 4 DDS Status Shift Reg, SN74LV166ADR 16 Bit Buffers, SN74LVTH162241DGGR LVDS Drivers LVDS Receivers Digital to Analog Converter, LTC2641CS8-16 SRAM, CY7C1041CV33-15ZC 16 Bit Transceivers, CLVTH162245IDGGREP Clock Oscillator, CB3LV-3C-50M0000-T Clock Oscillator, WINF.075J-50.0MHZ	LTM4604, U7	+5 V
+2.5V	FPGA's, Altera EP3C40F484C8N Analog to Digital Converters, AD7625BCPZ	LTM4604, U20	+5 V
+1.2V	FPGA's, Altera EP3C40F484C8N	LTM4604, U9	+5 V
+2.5V_PLL	FPGA's, Altera EP3C40F484C8N	Filtered Version of +2.5 V	+5 V
+1.2V_PLL	FPGA's, Altera EP3C40F484C8N	Filtered Version of +1.2 V	+5 V
+5V_DDS	DDS Regs for 3.3V_AVDD, MAX8881EUT33+T DDS Regs for 1.8V_AVDD, TPS79318DBVR	Filtered Version of +5V	+5 V
+3.3V_AVDD	DDS, AD9910BSVZ	MAX8881EUT33+T	+5 V
+1.8V_AVDD	DDS, AD9910BSVZ	TPS79318DBVR	+5 V
+1.8V	DDS, AD9910BSVZ	LP3856ES-ADJ, U8	+5 V
+5V_A	Regulator for +3.3V_A, MAX8881EUT33+T Regulator for +2.5V_ADC, TPS79325DBVR	Filtered Version of +5V	+5 V
+3.3V_A	Digital Potentiometers, MCP4021-202E/SN 2.5V DAC Reference, REF3125AIDBZT	MAX8881EUT33+T	+5 V
+2.5V_ADC	Analog to Digital Converters, AD7625BCPZ	TPS79325DBVR	+5 V
+5V_ADC	Analog to Digital Converters, AD7625BCPZ ADC Buffer Amp, AD8139ARDZ	MAX8881EUT50+T	+12 V
+12V_A	Op amps, AD8066AR Input Relays, IM23TS Test Switch, ADG419BR	Filtered Version of +12 V	+12 V

	5V Voltage Reference, ADR02AR		
+12V_RF	RF Amplifier, ERA-5SM+	Filtered Version of +12 V	+12 V
-12V_A	Opamps, AD8066AR	Filtered Version of -12 V	-12 V

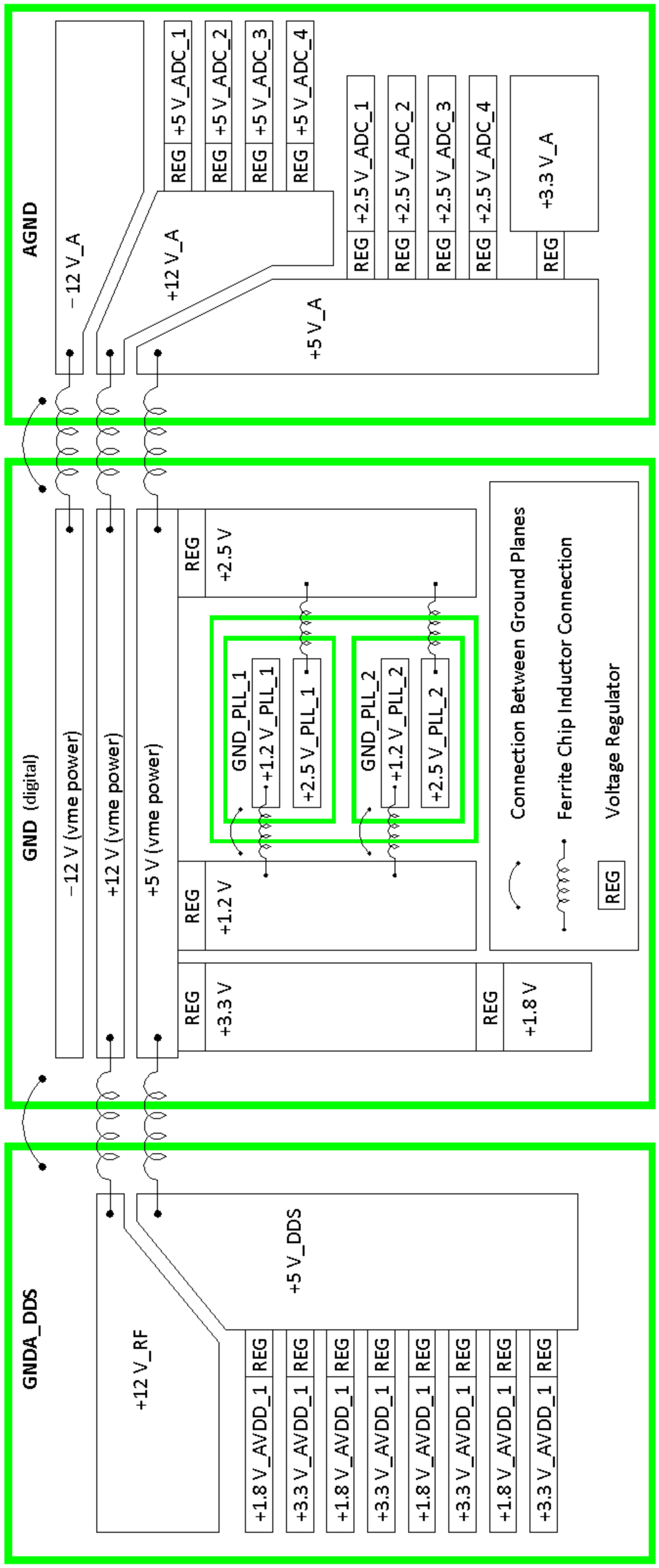


Figure II.8.1 Power distribution diagram.

II.9 Memory on the DDS Module

Memory on the DDS Module includes both non-volatile Flash memory and fast access RAM memory. The Flash memory is a single Intel Strata Flash, part number JS28F256P33B95. The Flash memory is organized into 4 each 16k word blocks and 254 each 64k word blocks, where a word is 16 bits. Its main use is backing up curves and parameters we do not want to lose to an unexpected power failure. Being Flash, whole blocks must be erased before being written. This does require more consideration with regard to when data is backed up. Processes have been written that run in the Upper FPGA for backing up and restoring (booting up) programmable curves and parameters. For the sake of avoiding data corruption should a power failure occur during a backup operation, alternate Flash blocks are used. That is when backing up, the non-active memory block is written. Then when the backup is complete the new block of data is set as the active block to be read from during the next restore operation. The last 32 bit word in each block holds the backup increment value. The block with the highest increment value is the active block. This does require a special procedure when the module is initially put into service, but this can be automated and even performed while the module is in service. The details of this initialization and the details of the backup and boot up processes are given elsewhere in the documentation.

The RAM memory is available in the FPGA's and the Cypress CY7C1041CV33-15ZC 64k x 16 SRAM. Table II.9.1 is a summary of the current memory allocation. As the applications for the DDS module are developed, a detailed memory map is maintained. Memory map information is also kept up-to-date in a single VHDL constants file which is included by VHDL code modules compiled for the FPGA's.

The Shared Memory is dual-ported RAM through which the crate front-end processor reads and writes the application parameters and curves. The protocol has been established in the design of the VXI-DSP modules currently in use in the Booster LLRF controls. Processes are setup in the Upper FPGA for managing the details of updating parameters, servicing requests to read and write the curves, and properly execute the setting and clearing of flags and semaphores according to the established protocol. More details are given elsewhere in the documentation.

Read Only Memory, ROM, has been implemented within the FPGA's for storing parameters and parameter limits that are not to be changed through the front-end processor. These include properties of established curves used in the application, upper and lower limits for programmable parameter settings, and address pointers that define the how the memory has been laid out. The ROM memory is set by assigning the Altera memory blocks a memory initialization file. This file defines the values that are in memory when the FPGA's are configured at power up.

Table II.9.1 Summary of memory allocations as of 8/24/2010

Memory Allocation Summary, 8/24/2010		
Description	Word Size, bytes	No. of Words
Upper FPGA RAM		
Shared Memory, Read-Only	4	1024
Shared Memory, Reserved RAM	4	448
Shared Memory, Readings, Settings, Flags, Curve Request Semaphores, etc.	4	406
Shared Memory, Message Buffer	4	128
Shared Memory, Curve Buffer	4	2048
Curve Table 1, Time-Value Pairs	4	2048
Curve Table 2, Time-Value Pairs	4	2048
Curve Table 3, Time-Value Pairs	4	2048
Curve Table 4, Time-Value Pairs	4	2048
Curve Table 5, Time-Value Pairs	4	2048
Curve Table6, Time-Value Pairs	4	2048
Curve Table 7, Time-Value Pairs	4	2048
Curve Table 8, Time-Value Pairs	4	2048
DDS Setup ROM 1	32	6
Paraphase Curve (curve values in 1 micro-second step)	1024	2
Frequency Curve (curve values in 4 micro-second steps)	10240	4
Lower FPGA RAM		
Frequency Curve, alternate (curve values in 4 micro-second steps)	10240	4
RF Voltage Bias Curve (curve values in 4 micro-second steps)	10240	4
Phase Offset Curve 1 (curve values in 4 micro-second steps)	10240	2
Phase Offset Curve 2 (curve values in 4 micro-second steps)	10240	2
Injection Curve Buffer	1024	4
Paraphase Curve, alternate (curve values in 1 micro-second step)	1024	2
Cypress SRAM		
Phase Offset Curve 3 (curve values in 4 micro-second steps)	10240	2
Phase Offset Curve 4 (curve values in 4 micro-second steps)	10240	2
Phase Offset Curve 5 (curve values in 4 micro-second steps)	10240	2

II.10 FPGA Programmable Logic for Data Processing

II.10.1 General Description of FPGA's

The DDS Module provides two Altera EP3C40F484C8N, Cyclone III FPGA's for providing data processing, memory management, IO interfaces and a front-end processor VME interface. Using two large FPGA's instead of one enormous FPGA provides more IO pins for interfacing the DDS chips and the analog inputs and outputs. The Cyclone III FPGA's provide many LVDS differential pair IO pins. LVDS is a desirable logic standard for maintaining low noise when implementing many high speed serial connections.

Besides providing the desired number of IO connections, the FPGA's also provide 1,161,216 memory bits, 252 embedded 9-bit multiplier elements for data processing, 4 independent PLL clock generating blocks and 39,600 programmable logic elements. It did not seem necessary to fit a DSP or other type of processor onto the board and manage another IDE code development environment. However, development of the Shared Memory management, as used by the current VXI-DSP Booster LLRF modules, was exceedingly more difficult written in VHDL processes than it would have been to write the equivalent in C or even assembly language for a processor. There is also the concern that future extensions or evaluation of the VHDL processes will be more difficult than reading C or assembly language routines. After achieving a fully programmed prototype module that meets the requirements of the LLRF application we will evaluate whether a different mix of FPGA's and processors would be more appropriate. Below is a current list of processes and interfaces implemented on the FPGA's. There are likely more to come. Details on the implementation and operation of these processes are given elsewhere in the documentation.

Upper FPGA

1. VME Interface (J1/P1 signals)
2. Inter-FPGA Interface
3. Process Timer/Sequencer
4. Trigger Management
5. Shared Memory Update Process
6. Curve Buffer Request Process
7. Curve Interpolation Process
8. Injection Curve Management
9. Boot up and Backup Processes
10. Flash Memory Interface
11. DDS Setup Process
12. DDS Synchronization and Serial IO Interface
13. Run Mode DDS Update Process
14. High Speed Serial Interface

Lower FPGA

1. VME Interface (J2/P2 signals)
2. Inter-FPGA Interface

3. Analog Input Interface
4. Analog Output Interface
5. Digital Potentiometer Calibration Interface
6. External SRAM Interface
7. Rear VME Transition Module Interface (Triple Phase Detector)
8. PC / USB Interface (Rear VME)
9. Run Mode Control Data Processing
    a. Booster Paraphase Process
    b. Booster Acceleration Phase-Lock Process
    c. Main Injector Phase-Lock Process

II.10.2 Programming the FPGA's

As noted, the DDS module uses two FPGA's. The FPGA's are configured when power is applied to the module from a single serial EEPROM memory device, an Altera EPCS16SI8N. The EPCS16SI8N configuration device provides 16.7 Mbits of memory. Each of the two EP3C40 Cyclone III FPGA's are specified as having a maximum uncompressed configuration file size of 9.6 Mbits. Hence, the amount of configuration memory could theoretically be a limiting factor as to how much can be programmed into the FPGA's. I do doubt that we will reach this limit. If we do we can consider what we can gain from the available configuration file compression features offered by the devices.

There is some provision in the DDS module design for configuring the FPGA's from the Flash memory chip, but this has not been developed past the board layout stage. Use of the Flash memory for FPGA configuration would allow for programming the FPGA's remotely, in the field, via the front-end crate processor.

The two FPGA's are configured sequentially from the serial EEPROM, first the Upper FPGA and then the Lower. When the individual FPGA configurations are compiled, ***.sof** files are generated. The current procedure is to copy a newly generated "DDS_LOWER.sof" file from the Lower FPGA project directory to the Upper FPGA project directory. Within Quartus, with the Upper FPGA project active, we use the "Convert Programming Files …" utility, under the "File" pull down menu. Within the utility we press the "Open Conversion Setup Data" button and load the file "UPPER-LOWER convert file.cof". The UPPER_LOWER convert file was previously setup to append the "DDS_LOWER.sof" file to the "DDS_UPPER.sof" file to create the "UPPER-LOWER.pof" file, with which we will program the configuration memory. Figure II.10.2.1 is a screen shot of what this utility looks like after the conversion setup file has been loaded. The "Generate" button is clicked to execute the conversion.

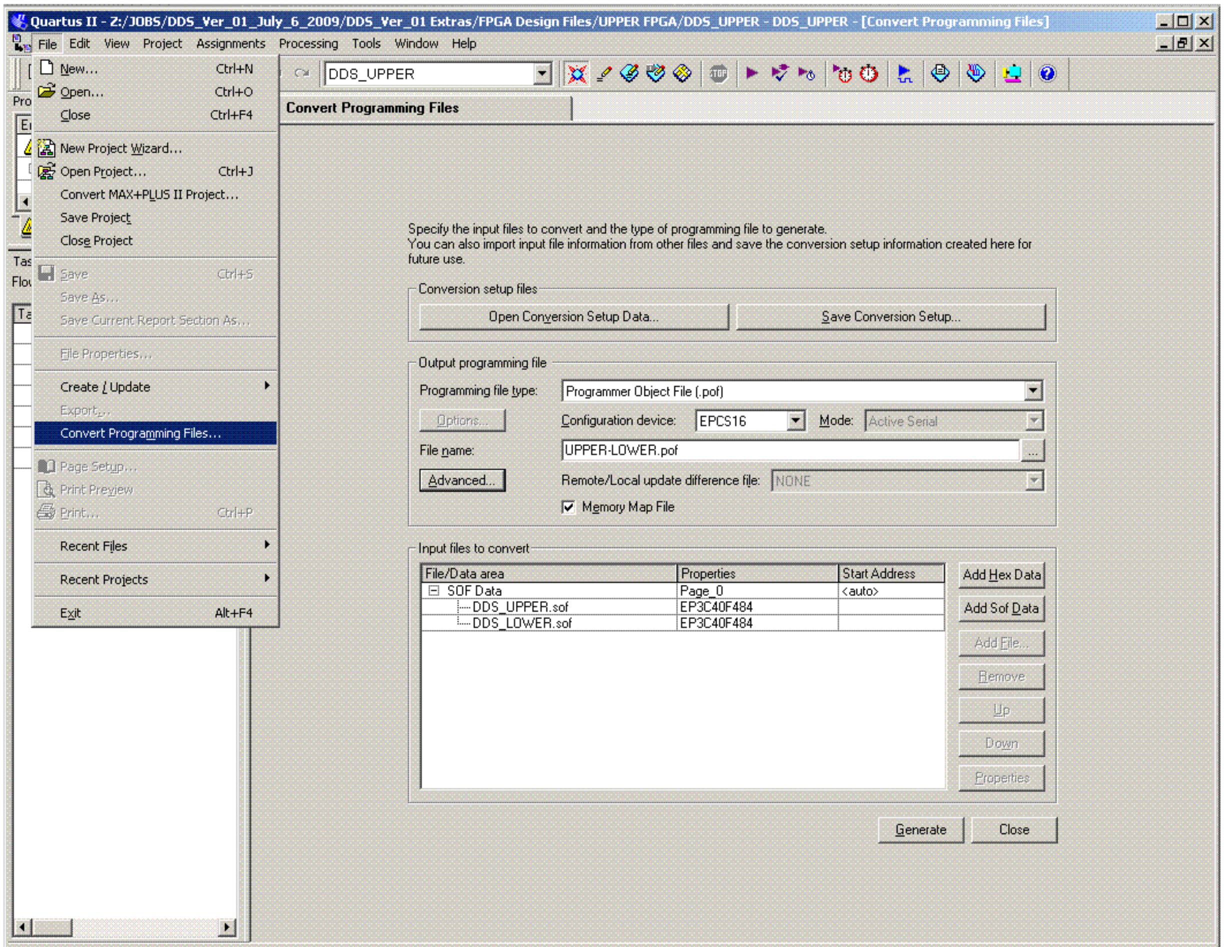


Figure II.10.2.1 Screen shot of the programming file conversion utility

Once the “UPPER-LOWER.pof” file has been generated, or re-generated, we use the configuration device programmer utility within Quartus with an Altera USB-Blaster Cable. To program the EPCS16SI8N device on the DDS module the USB-Blaster, or similar programming cable, is connected to J22. The programmer connection is shown in Figure II.10.2.2.

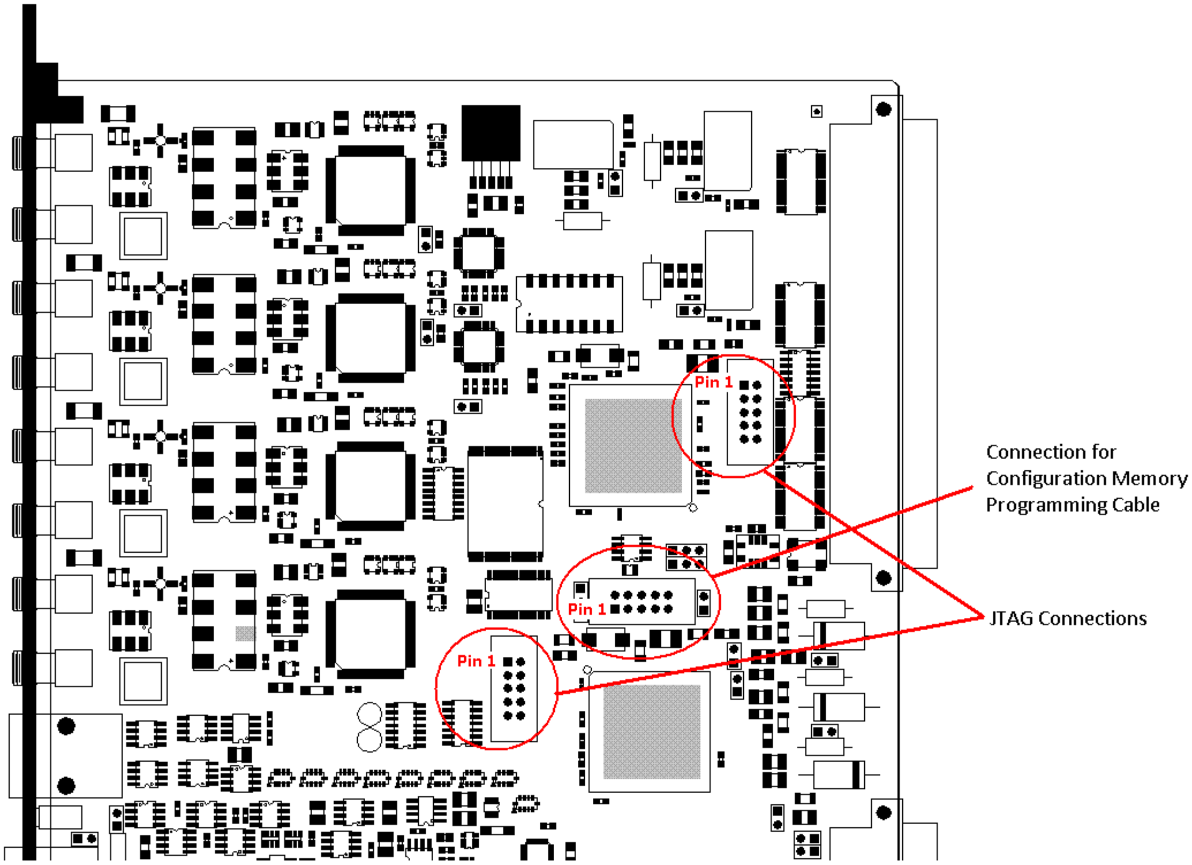


Figure II.10.2.2 FPGA programmer connections and JTAG interface connections

III. FPGA Process and Interface Details

III.1 Inter-FPGA Interface

III.1.1 Introduction

The Inter-FPGA Interface is used to transfer data between the Upper and Lower FPGA's. The interface is shared by three types of processes. The first process is the VME Interface. Not only is there a need for the front-end processor to access memory in both the Upper and Lower FPGA RAM, but every VME transaction requires data bits be exchanged between the two FPGA's. Details on why is given in the section on VME Interface Details.

The second type of processes that use the Inter-FPGA interface are the memory management processes. These include the Shared Memory Update process and the Curve Interpolation process where the Run Mode curves are generated from the Shared Memory curve buffers. Details on these processes are given later also.

The third type of processes is the Run Mode processes where data is exchanged between the Lower FPGA and the Upper FPGA which controls the DDS RF outputs. The Lower FPGA is the location of the Frequency Curve memory.

Table III.1.1 lists the interconnecting signals between the FPGA's, and Figure III.1.1 provides a block diagram of the interface. As seen in the block diagram, the Inter-FPGA interface is controlled by logic in the Upper FPGA. That is, only processes in the Upper FPGA can perform Reads or Writes of memory or registers in the Lower FPGA, not the other way around.

The interface exchanges a 20 bit memory address and 32 bits of data driven by either the Upper or Lower FPGA. The interface topology was chosen with a constraint that only 24 interconnecting signal were available. An alternate approach is possible where 20 interconnect signals are used bi-directionally to transmit the address followed by the upper 16 data bits, followed by the lower 16 data bits. This approach, however, would not allow VME address bits vAA [25..24] to be shared with the Upper FPGA and the "reserved" signal would need to be employed also. The transfer rates are very nearly the same in each case. The topology currently used is a bit more complex (a downside).

The Inter-FPGA Interface, as illustrated in Figure III.1.1, transfers addresses from the Upper FPGA to the Lower FPGA using 5 each, 4 bit shift registers. Data is transferred from either Upper to Lower or Lower to Upper using 8 each, 4 bit shift registers. The shift registers are clocked at 100 MHz. That is 2 times the main logic clock.

When writing data from Upper to Lower, address and data are transferred together followed by the "WR_Lower_mem" signal once the address and data has had a couple logic clock intervals to setup in the Lower PFGA memory. Reading Lower memory requires two cycles. The address is transmitted to the Lower FPGA on the first cycle. On the next cycle the addressed data is transmitted from Lower to Upper.

Table III.1.1 Listing of Inter-FPGA Interface signals

Interface Signal Name	FPGA IO Pin Name	Direction	Function
Serial_Data_Link[7..0]	C_Bus[7..0]	BIDIR	These lines transmit 32 data bits between the FPGA's. The direction of data flow is controlled by the signal RD_Lower_Mem.
Serial_Addr_Link[4..0]	C[12..8]	Upper to Lower	These lines transmit a 20 bit memory address when reading or writing the Lower FPGA Memory.
RD_Lower_Mem	C13	Upper to Lower	This signal controls the direction of the Serial Data Link lines. The Lower FPGA will drive the data lines when this signal is high.
RCV_CLK_FROM_LOWER	C14	Lower to Upper	This is a serial data clock used to synchronize the transfer of data from the Lower FPGA to the Upper FPGA.
WR_Lower_Mem	C15	Upper to Lower	Used as the Write pulse for writes to the Lower FPGA memory and registers.
LD_Lower_mem	C16	Upper to Lower	Sent to latch memory data being read from the Lower FPGA into the serial shift registers.
RD_Lower_sr	C17	Upper to Lower	Signals the Lower FPGA to latch the address and data transmitted from the Upper FPGA.
ENA_shift_out	C18	Upper to Lower	This gates the shift registers in both the Upper and Lower FPGA's to shift.
RCV_CLK_TO_LOWER	C20	Upper to Lower	This is a serial data clock used to synchronize the transfer of address and data from the Upper FPGA to the Lower FPGA.
Ext_Board_Select(*)	DIO_OUT_1A3 DIO_OUT_2A1 (*)	Lower to Upper	This is a VME Interface signal which is the result of a comparison of VME address bits A[31..28] with the board address DIP switches.
vAA[25..24]	--	Lower to Upper	J2/P2 VME address bits. NOT CURRENTLY ROUTED

(*) An external jumper has been soldered onto the boards to tie these signals together on the prototypes. The buffer that normally drives these signals has been permanently tri-stated.

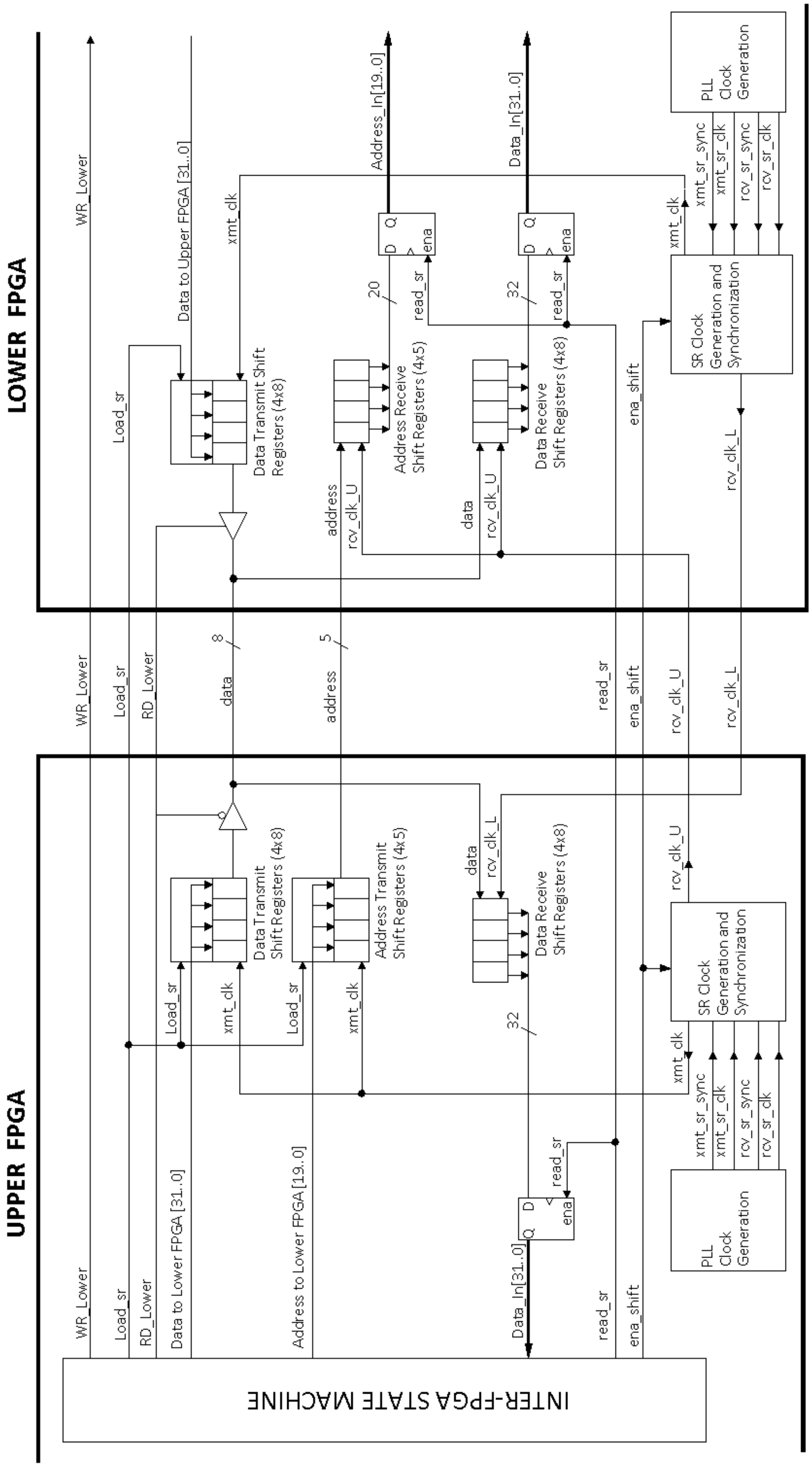


Figure III.1.1 Block diagram of the Inter-FPGA interface

There are additional inter-FPGA signals that are not involved with the data interface. These are associated with the ADC data from ADC1 and ADC2. These were routed directly through the Lower FPGA to the Upper FPGA to reduce latency in processing these analog input values.

Table III.1.2 Listing of other Inter-FPGA signals

Interface Signal Name	FPGA IO Pin Name	Direction	Function
ADC_1_DATA	C19	Lower to Upper	Used to route ADC1 serial data directly to the Upper FPGA.
ADC_1_SCLK	C_IN21	Lower to Upper	Used to route ADC1 serial data directly to the Upper FPGA.
ADC_2_DATA	C_IN22	Lower to Upper	Used to route ADC2 serial data directly to the Upper FPGA.
ADC_2_SCLK	C_IN23	Lower to Upper	Used to route ADC2 serial data directly to the Upper FPGA.
IO_Sample_Cmd	DIO_OUT_1A4(*)	Lower to Upper	Reserved

III.1.2 Sharing the Interface

There are three processes that need to transfer data between the Upper and Lower FPGA's. These are the VME Interface, the Shared Memory Update and Curve Interpolation processes, and Run Mode processes that run when the module is actively controlling a Booster acceleration cycle. The VME Interface is allowed to use the Inter-FPGA link at any time, but might be held-off for a short interval, less than 2 micro-seconds. VME data transfer cycles are easily held-off as long as the cycle is completed within an interval defined by the crate processor. The typical time-out is 64 micro-seconds.

The Run Mode processes only run during the active portion of the 66.67 millisecond (15Hz) Booster cycle. The interval of active control is approximately 38 milliseconds. During the active control interval, Run Mode processes are expected to need the Inter-FPGA link no more than 500 nanoseconds every 1000 nanoseconds. Hence there are gaps in which to permit VME Interface use of the link even during Run Mode.

The Shared Memory Update and Curve Interpolation processes are run only during the 28 milli-second intervals between the active control intervals. These processes are never allowed to run during Run Mode.

So, logic is setup at the input of the Inter-FPGA Interface to receive, latch and service requests from two sources,

Source A: VME interface
Source B: Run/Update processes

When the Inter-FPGA Interface receives either a Read pulse or a Write pulse from either Source A or the active Source B, the leading edge of the pulse is detected and the occurrence of the request is latched. It is expected that the requesting process will wait and will not issue any other Reads or Writes until the requested transfer has been completed. When serviced, the latched request is cleared and a pending request is serviced.

Since VME Interface requests may come in at any time a "Permit_VME" bit is used to avoid servicing VME Interface requests just before a time critical Run Mode transfer is to occur. For instance the Permit_VME bit would be set to disallow VME requests for 100 nanoseconds before the critical Run Mode transfer is scheduled to occur. Any VME request coming in before the 100 nanosecond interval would have time to complete before the critical Run Mode transfer is expected. Similar scheduling can be established just before the transition from the Update Mode to the Run Mode.

III.1.3 Timing the Interface

The serial transfer of address and data between the Upper and Lower FPGA's is accomplished using a transmit clock to shift data out of the transmitting shift registers and a receive clock to shift data into the receiving shift registers. These two clocks, for any transmitter-receiver pair, are generated by the transmitting FPGA. Hence, the receive clock for the data receiver in the Upper FPGA is generated by the Lower FPGA, and vice-versa. To say it one more way, the transmitting FPGA controls the timing of the transfer.

There is a propagation delay for the clocks and data passed between the two FPGA's. By using the clock phase adjustments available on the FPGA PLL clock generation block we can optimize the timing between when the data is shifted out by the transmit clock and when the data bits are shifted in, or latched, by the receive clock at the receive shift registers, in the other FPGA.

There is another issue with the timing of the transmit and receive clocks, in that the clocks are gated on for only four pulses. *Note that the first bit of the data to be received is sitting at the output of the transmitter, input of the receiver, just after the parallel load of the data into the transmit shift register has been accomplished.* So the sequence of shifting should be that the receiver shifts the first data bit in before the transmitter shifts the next bit out. Note next that the clocks are gated by a signal from the Inter-FPGA Interface state machine logic clocked at 50 MHz. The transmit and receive clocks we intend to gate are running at 100 MHz . The clock gate needs to be synchronized to the 100 MHz clocks or we may be unlucky and have the transmit clock edge strike before the receive clock edge (at the other FPGA). So, we use the PLL to generate two more clocks, which we can phase adjust, to synchronize the gate.

Figure III.1.3.1 is a block diagram of the logic used to synchronize the gate and produce the four transmit and receive shift register clocks that transfer the Inter-FPGA data. Figure III.1.3.2 is the resulting timing relationship of the clocks.

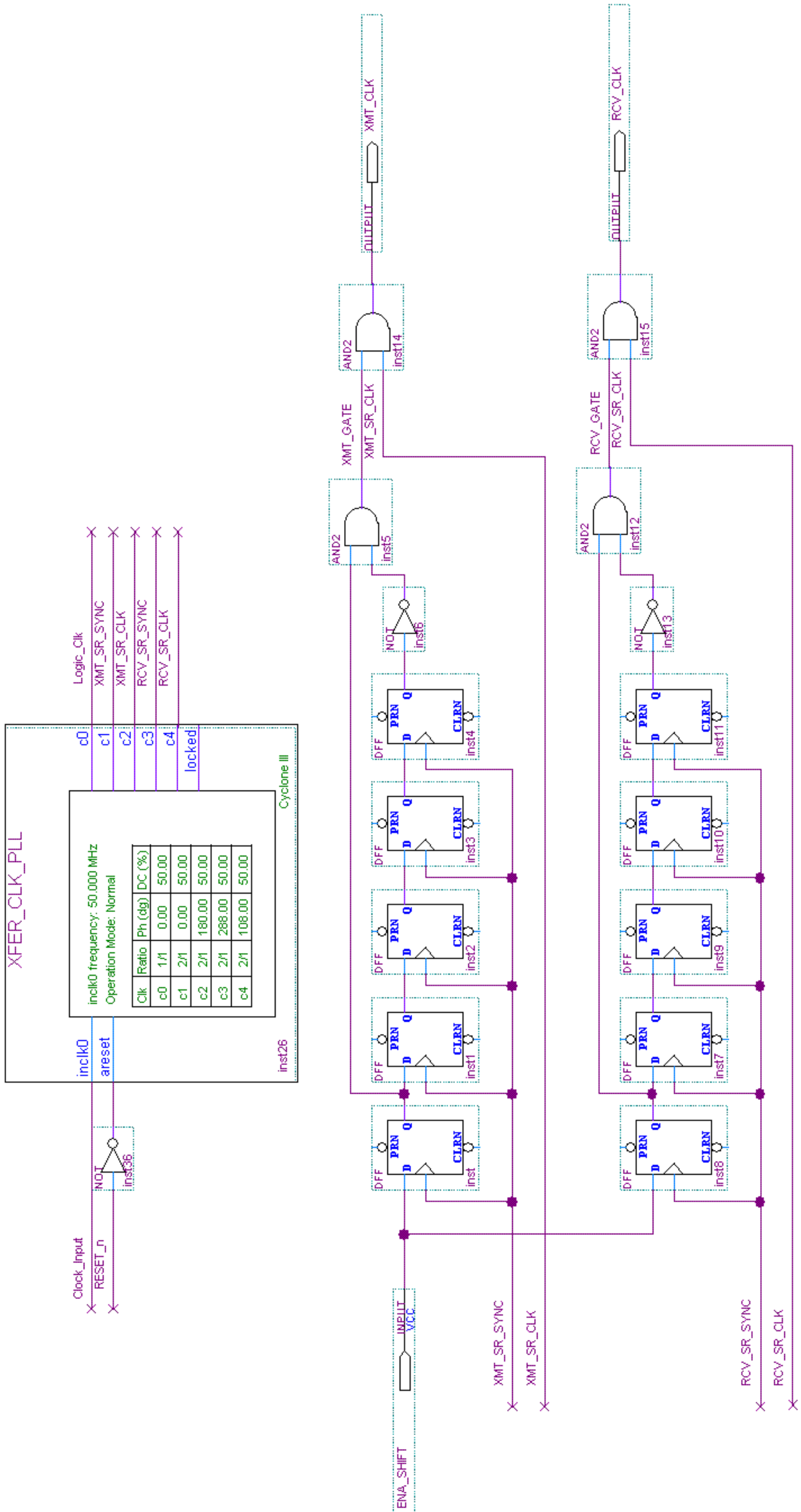


Figure III.1.3.1 Logic used in synchronizing the Inter-FPGA SR Clocks

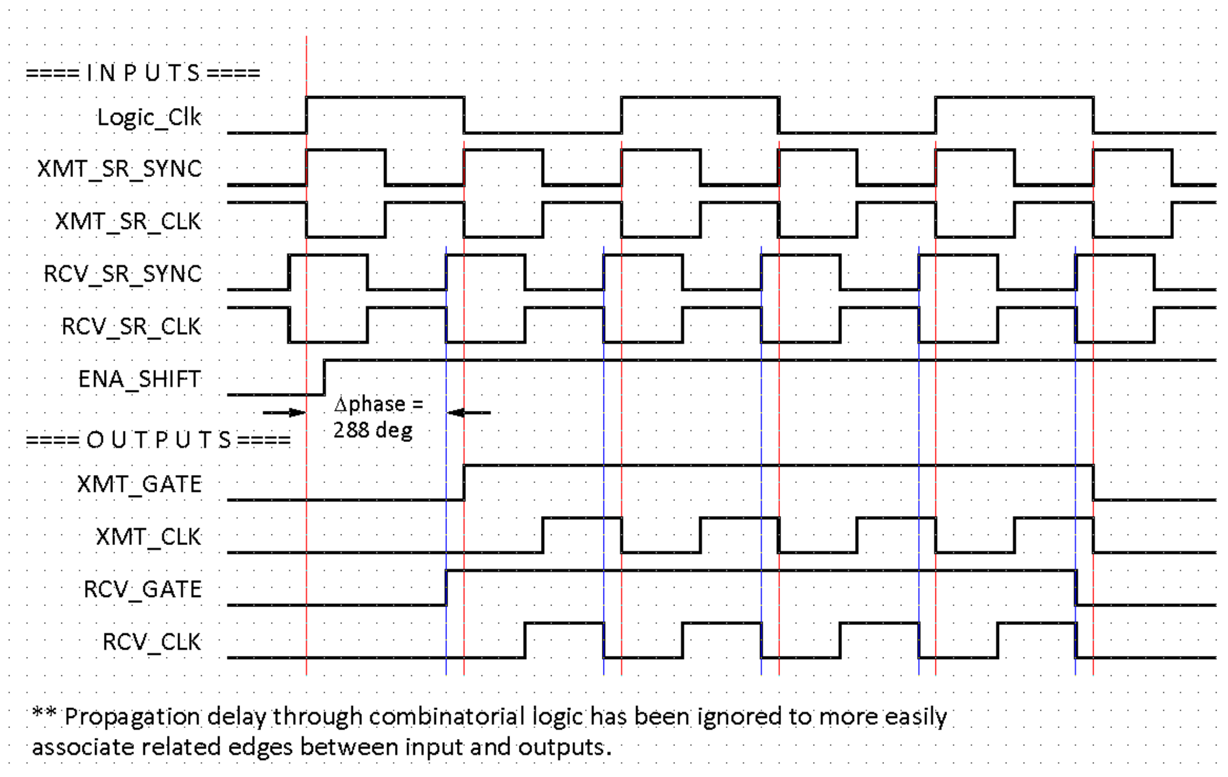


Figure III.1.3.2 Timing diagram of the Inter-FPGA Transmit and Receive SR Clocks

III.2 VME Interface

The VME Interface is used to transfer data between the front-end, crate processor and the DDS module. The interface has already been introduced in Section II.1. Here we wish to document more of the details of the interface as implemented in the FPGA logic.

The VMEbus Specification used as reference is ANSI/IEEE STD1014-1987. The DDS module layout, backplane connection, signal buffers and transceivers, and the mechanical specifications for this 6U module all conform to this standard. The logic abides by all the timing requirements defined for the data transfer cycles that it supports.

Below is a list of features of the standard that are NOT implemented.

1. Nothing in regard to the Arbitration Bus signals other than daisy chaining the Bus Grant signals.
2. Nothing in regard to the Interrupt Bus signals other than bringing IACK/ into the FPGA and daisy chaining IACK_IN to IACK_OUT.
3. There are no “Single-Byte” or “Triple-Byte” accesses, only “Double-Byte” and “Quad-Byte” accesses.
4. Block transfers have not been implemented. This could be a reasonable future upgrade.

We should also note here that the user defined pins on the J2/P2 connection have been defined and must be considered before connecting to J2 backplanes that bus the user defined pins.

The module will respond to the following address modifiers. Standard addressing AM codes indicate that address lines A01-A23 are being used, and Extended addressing AM codes indicate that address lines A01-A31 are being used. It is important for the logic to distinguish between standard and extended addressing since there is a comparison between the most significant 4 address bits and the board address bits. The board address bits are set by DIP switches on the board to locate the board in the VME bus address space.

Table III.2.1 Supported VME addressing modes

HEX Value of AM(5..0)	Function
3F	Standard supervisory block transfer
3E	Standard supervisory program access
3D	Standard supervisory data access
3B	Standard nonprivileged block transfer
3A	Standard nonprivileged program access
39	Standard nonprivileged data access
0F	Extended supervisory block transfer
0E	Extended supervisory program access
0D	Extended supervisory data access
0B	Extended nonprivileged block transfer
0A	Extended nonprivileged program access
09	Extended nonprivileged data access

The simplest description of a VME data transfer cycle, as experienced by the DDS module, is

1. The address bits, address modifier bits and LWORD/ are set.
2. At least 35 nanoseconds later AS/ the address strobe will go low.
3. The WRITE/ signal will be set; low if it is a write operation, high if not. If it is a write operation the data bits will be set on the bus.
4. DS0/ and DS1/, the data strobes, will go low. It could be that the data strobes go low at the same time as the address strobe, but not before.
5. When the address strobe goes low the VME Interface logic will compare the most significant 4 address bit to the 4 board address bits and will signal whether there is a match. If there is not a match nothing else occurs on the DDS module.
6. The address presented is also decoded to determine whether Lower FPGA memory is being addressed.
7. If the board is being addressed and the data strobes go low then a transfer with this board is going to happen and the address is latched for the remainder of the cycle.
8. Data is either latched into the address memory location if it is a write operation, or addressed data is driven onto the VME data bus by the DDS module.
9. The DDS module now pulls the DTACK data acknowledge line low to signal it is ready.
10. Then the bus master module lets the data strobes go high signaling the end of the data transfer cycle.
11. The DDS module immediately lets DTACK go high.

During every VME data transfer cycle the DDS module must exchange data between the Upper and Lower FPGA's using the Inter-FPGA Interface. And it does not matter whether the data being addressed is in the Upper or Lower memory. Note that the physical layout of the module connects the majority of the VME address, data and control lines, from the J1 connector, to the Upper FPGA. However, the VME address bit [31..24] and the VME data bits [31..16] from the J2 connector are connected to the Lower FPGA.

The majority of the RAM memory, particularly the Shared Memory through which the front-end processor interacts with the module, is 32 bit. Table III.2.2 summarizes the data exchanged between the Upper and Lower FPGA's for the four types of VME data transfers.

Table III.2.2 Summary of inter-FPGA transfers necessary for VME data transfers.

Type of VME Transfer	Data transferred between FPGA's
VME Read from Upper FPGA Memory	D[31..16] from the Upper memory is transferred to the VME J2 via the Lower FPGA.
VME Read from Lower FPGA Memory	D[15..0] from the Lower memory is transferred to the VME J1 via the Upper FPGA.
VME Write to Upper FPGA Memory	D[31..16] from VME J2 is transferred to the Upper memory via the Lower FPGA.
VME Write to Lower FPGA Memory	D[15..0] from VME J1 is transferred to the Lower memory via the Upper FPGA.

Routing of data bits D[31..16] and D[15..0] is different for VME Reads and Writes than for other types of Reads and Writes. When the VME Interface is active, using the inter-FPGA interface, it will set Bit 19 of the address it transmits to the Lower FPGA High for all VME data transfers except Lower Memory VME Reads. The logic in the Lower FPGA will use this bit to properly route the data bits, but will clear the bit in the address when being applied to the Lower memory address decoding. The Lower memory and SRAM address space is located between 0x20000 and 0x5FFFF (i.e. Bit 19 is always 0). Listing III.2.1 is the VHDL logic that routes the data bits according to the value of Bit 19. Table III.2.3 assists with the signal name definitions used in the code.

Listing III.2.1 Lower FPGA version of the VME Interface VHDL code

```
VME_Process_Active <= Upper_Addr_To_Mem_in(19);

-- clear extra bit for VME Writes to Lower FPGA Memory
Upper_Addr_To_Mem(19)          <= '0';
Upper_Addr_To_Mem(18 downto 0) <= Upper_Addr_To_Mem_in(18 downto 0);
-----------------------------------------------------------------------------
process (Upper_Addr_To_Mem_in(19 downto 16), VME_Data_In, Upper_Data_To_Mem_in,
Lower_Data_To_Interface, VME_Process_Active)
begin
-----------------------------------------------------------------------------
-- Note VME_Process_Active is not set for VME Reads of the Lower FPGA
```

```
if(VME_Process_Active = '0') then

-- This sets up for an Upper FPGA Write to the Lower FPGA or SRAM memory
      Data_To_Mem(31 downto 16)      <= Upper_Data_To_Mem_in(31 downto 16);
      Data_To_Mem(15 downto  0)      <= Upper_Data_To_Mem_in(15 downto 0);

-- This sets up for an Upper FPGA Read of the Lower FPGA or SRAM memory
      InterFPGA_D_Out(31 downto 16) <= Lower_Data_To_Interface(31 downto 16);
      InterFPGA_D_Out(15 downto  0) <= Lower_Data_To_Interface(15 downto 0);

-- This sets up for a VME Read of the Lower FPGA
      VME_Data_Out(31 downto 16)     <= Lower_Data_To_Interface(31 downto 16);
-----------------------------------------------------------------------------
else  -- VME Write Operation or VME Read of Upper FPGA

-- This sets up for a VME Write to Lower FPGA or SRAM memory
      Data_To_Mem(31 downto 16)      <= VME_Data_In(31 downto 16);
      Data_To_Mem(15 downto  0)      <= Upper_Data_To_Mem_in(15 downto 0);

-- This sets up for a VME Write to Upper FPGA or Flash memory
      InterFPGA_D_Out(31 downto 16) <= VME_Data_In(31 downto 16);
      InterFPGA_D_Out(15 downto  0) <= Lower_Data_To_Interface(15 downto 0);

-- This sets up for a VME Read of the Upper FPGA.
      VME_Data_Out(31 downto 16)     <= Upper_Data_To_Mem_in(31 downto 16);
end if;
end process;
```

Table III.2.3 Definitions for signals in Listing III.2.1

Signal Name	Definition
VME_Process_Active	Set equal to Bit 19.
Upper_Addr_To_Mem_in	This is the address to Lower Memory received from the Upper FPGA via the Inter-FPGA Interface.
Upper_Addr_To_Mem	This is the address to Lower Memory after Bit 19 has been cleared.
Data_To_Mem	Data bus to the Lower Memory.
Upper_Data_To_Mem_in	Data bits received from the Upper FPGA via the Inter-FPGA Interface.
InterFPGA_D_Out	Data bits being transmitted from the Lower FPGA to the Upper FPGA.
Lower_Data_To_Interface	Data from Lower FPGA Memory output data bus.
VME_Data_Out(31 downto 16)	Most significant VME data bits to the J2 VME bus connection.
VME_Data_In(31 downto 16)	Most significant VME data bits from the J2 VME bus connection.

III.3 Address and Data Bus Structure

A description of the available memory on the DDS Module was provided back in Section II. The RAM memory allocated in the FPGA's is generally allocated as dual-port memory. These memory blocks are generated by the Altera software and allow access to the memory cells by two different address data buses at the same time. We take advantage of this to allow the crate front-end processor to have free access to most of the memory in the FPGA via the VME Interface while other processes may be accessing the same memory. VME does still have to share the Inter-FPGA Interface, but this is manageable.

Figure III.3.1 is a block diagram of the structure of the address data bus that connects processes and interfaces to the memory. The data buses within the FPGA are unidirectional. There is a Data To Memory bus and a Data From Memory bus. The Data From Memory bus is generally supplied to the input of every process or interface that requires data from memory. At the output of every process or interface that needs to Read or Write memory there are Address To Memory bits, Data to Memory bits, Read and Write signals, and a Memory Control Enable signal. Only one Memory Control Enable signal is expected to be active at any point in time. This signal connects the address, data, and control signals of the process as the input to the memory. This is basically the function of the Memory Control Mux's shown in Figure III.3.1.

Logically the FPGA memory is grouped in blocks according to function. The blocks labeled Memory Data De-Mux are representative of one level of the memory address decoding. The blocks connect the data from the addressed memory to the Data From Memory bus and, in the FPGA code as written, route the Write pulse to the desired memory block.

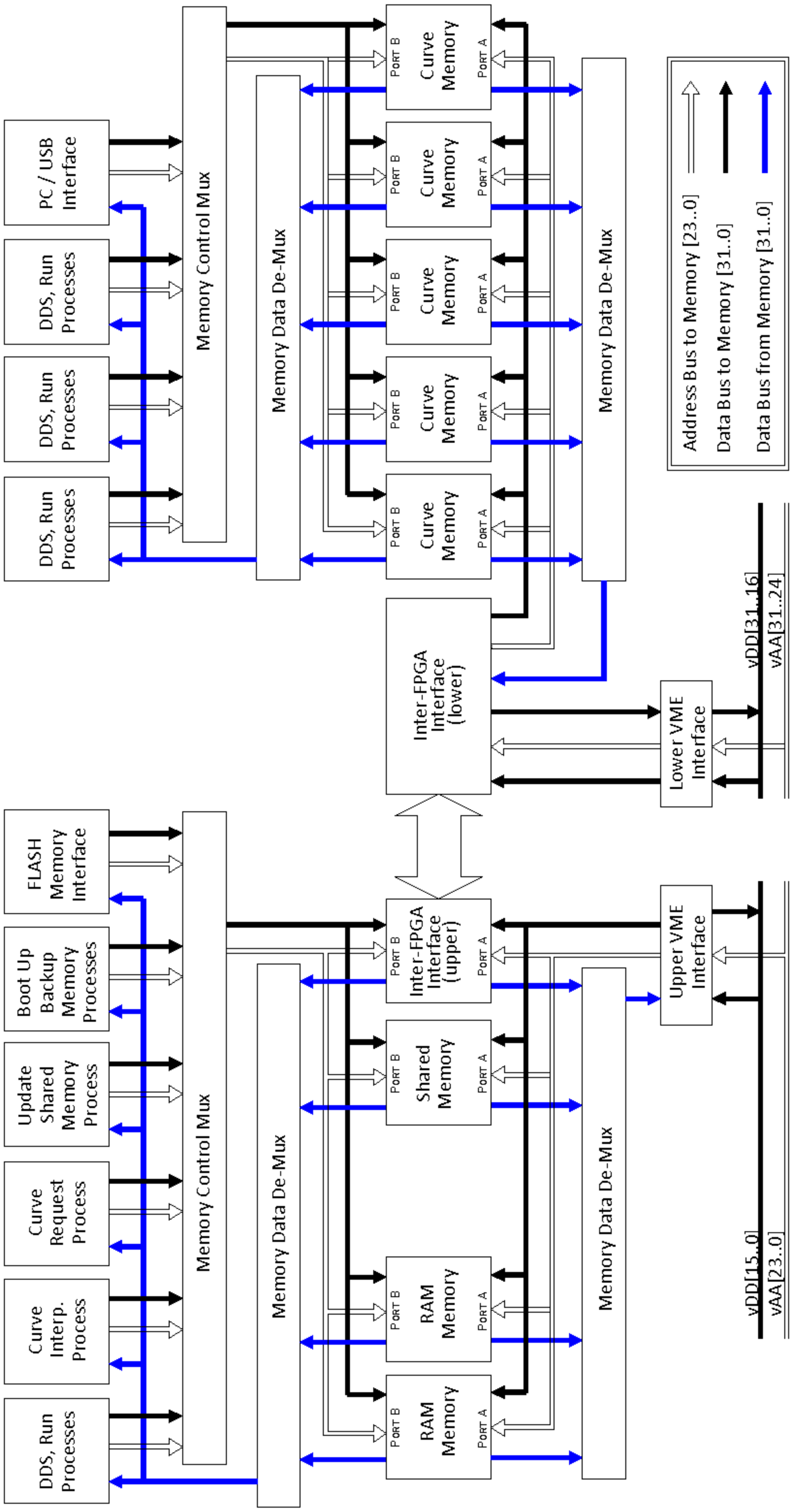


Figure III.3.1 Block diagram of the structure of the address-data bus.

III.4 Shared Memory Parameter Update Process

The Shared Memory is where the crate front-end processor sets and reads all of the parameters, process settings, and curves of the DDS LLRF application. The structure of the interface and associated protocol was established in the design and implementation of the Booster VXI-DSP modules that currently provide several Booster LLRF control functions including the current LLRF Frequency Source.

The purpose of the Update process is to keep the Run Mode parameter registers in-sync with the shared memory used by the higher level processes in the crate controller and beyond in the ACNET control system.

The parameters used by the module are typically scale factors, offsets, time delays, and process flow control switch settings. Three memory words in shared memory are associated with each parameter. There is a "Setting", a "Reading" and an "Update Flag". When the control system changes one of the modules parameters it writes the new value into the Setting of the parameter and sets the parameters Update Flag to a value greater than zero. The Update process polls through the flags. When it finds a set flag it will test the new parameter value against upper and lower limits set in ROM Memory. If the new value is within the limits we write the associated Run Mode parameter register with the new value. Then the parameter register, just written, is read and the value is written into the "Reading", and then the Update Flag is reset to zero.

This process is also used to test every parameter Setting value and setup all of the parameter registers after a boot-up cycle, after the shared memory is initialized from the Flash. This is done by setting the Verify_All signal and then triggering an Update. If the Setting is not within its limits, the default value defined for the parameter is written to the parameter register and the parameters Reading.

III.5 Curve Request Process

The Curve Request Process is another front-end processor interface protocol being used in the current Booster VXI-DSP modules. The purpose of the Curve Request process is to poll all of the Request Curve and New Curve semaphores (flags) in Shared Memory.

When the host processor wishes to read the values for a particular curve in the DDS module it will "set" the associated Request Semaphore for that curve. Once the DDS module has detected this it will copy the values from the requested curves memory table to a common Curve Buffer in shared memory and "clear" the Request Semaphore. The host processor will be watching for the Request Semaphore to be "cleared" and when it is the host processor will retrieve the requested curve from the Curve Buffer.

When the host processor wishes to write a particular curve to the DDS module it will "clear" the CTBFREE semaphore (Curve Table Buffer Free) and will then proceed to write the new curve to the buffer. It will then "set" the New Curve semaphore for the curve that is to be updated with the data in the Curve Buffer. The host processor will also then release or "reset" the CTBFREE semaphore. Once the DDS has detected the New Curve semaphore having been set it will "clear" the CTBFREE semaphore, transfer the curve from the Curve Buffer to the particular curve table in memory, and then "reset" CTBFREE and "clear" the New Curve semaphore.

The curves being transferred in these operations are arrays of value-time pairs. This is the form that curves generated by the ACNET control applications are provided in.

To reiterate, the crate processor sets the table Request semaphore or New Curve semaphore and then will not read or write the buffer until the Curve Request Process has reset the semaphore. The Curve Request Process will read the buffer if the New Curve semaphore has been set. The Curve Request Process will write to the buffer if the table Request semaphore has been set.

The first word in each curve table specifies the number of value-time pairs in the table. Call this variable "tsize". Hence, the actual length of the table in words is 2x the number of pairs. After the number of pairs word follows "tsize" value words, then "tsize" time words. The maximum number of pairs per curve is 1,023. The maximum number of curves the memory and processes are set up to support is 8.

Whenever the Curve Request process writes new information to a curve table, the process will also set the Curve Table Status word for that particular curve to 1.

III.6 Curve Interpolation Process

The purpose of the Curve Interpolation process is to find curves that need to be updated and then do the interpolation necessary to transform a set of value-time pairs (1023 pairs max) into up to 10,240 curve points set at a constant time increment. Curves that do not require interpolation are simply transferred to the Run Mode curve memory.

The curves used in the DDS module applications differ in several respects. As just mentioned, some of the curves need to be interpolated from the value-time pair to a set of values at a fixed time step and some curves come from the ACNET front-end processor with the desired time step and do not need to be interpolated. Some curves are defined using 32 bit (4 byte) values and are addressed in 4 byte steps, while others are defined using 16 bit values and are addressed in 2 byte steps. The time step for the interpolated curves can also vary. The ROM memory defines a Curve Properties word for each of the curves defined for the application of the module. The properties word's definition is given in Table III.6.1

Table III.6.1 Definition of the Curve Properties word

Bit	Definition
31	Interpolated = 1 not interpolated = 0
30	32 bit data = 1 16 bit data = 0
29	Read-only = 1 Settable = 0
28 . . . 20	reserved
19 . . . 16	Time step in microseconds (eg. 8us, 4us, 2us, 1us)
15 . . . 0	Length in number of points (eg. 10,000 points, 1000 points)

The Curve Interpolation Process also makes use of pointers to the curve memory also stored in ROM memory adjacent to the properties words. The initial steps of the Curve Interpolation process is to increment through the Curve Table Status words, set in the Curve Request process, to find a curve that needs to be updated. As we increment through the status words we also increment through the list of curve address pointers and curve properties words so we can latch these words whenever a curve is found that needs updating. The Curve Table Status word is cleared just before the curve updating process begins.

The curve transformation is a simple linear interpolation of points at defined times lying between two of the value-time pairs. The two pairs are denoted as

$[\ T(k), V(k)\ ]$ and $[\ T(k+1), V(k+1)\ ]$.

We find the interpolated curve value at each time t by:

$$C(t) = V(k) + (t - T(k))*(V(k+1) - V(k)) / (T(k+1) - T(k))$$

The inputs to the computation, $T(k)$, $T(k+1)$, $V(k)$ and $V(k+1)$, are read from memory at the beginning and then again whenever time step t becomes greater than $T(k+1)$. Note, $[\ T(k),V(k)\ ]$ is the previous value $[\ T(k+1), V(k+1)\ ]$ and are set by a simple register transfer. The time step t is the output of a counter internal to this process. As a result, in order to produce a new output from the interpolation computation, we only need to increment the time step counter and perform two memory reads when t exceeds $T(k+1)$. There are a maximum of 1023 value-time pairs that need to be read. This is a small fraction of the 10,240 points to be computed. Besides reading new value-time pairs, the other task that consumes time is writing the computation results to the curve memory.

The curve interpolation computation takes approximately 33 clock cycles (660 nanoseconds) to produce a result. However, the computation is pipelined so that once we have incurred the initial delay the computation will produce a result as often as the input is updated. Currently we are planning to update the input every 7 clock cycles (140 nanoseconds). When the [Time, Value] pair must be updated the clock enable for the computation blocks is taken away and the computation halts while this is being done. The computation is started again once all of the inputs, including the time step, have been updated. When the computation is halted, writing results to the curve memory pauses also. This frees the data bus to retrieve $T(k+1)$ and $V(k+1)$ from memory.

The logic implements a second small state machine that updates the time step input variable and signals the main state machine when the [Time, Value] pair must be updated. The small state machine, therefore manages the computation inputs and strobes these into the computation. The main state machine gets into a small/fast loop looking for the computations output strobe and latching results which are immediately written to memory. When the small "input" state machine signals that the [Time, Value] pair must be updated, the main state machine disables the computation clock and branches off to update these inputs. The main state machine signals the small state machine that the update is done and then starts the computation and watches for output strobes again.

The maximum time the Curve interpolation process might take is computed assuming the maximum number of value-time pairs is used in the curve, 1024, and that the final curve has a maximum length of 10,240 points. We also must note that the amount of time needed to compute a point and update the input variables [T(k),V(k)] and [T(k+1),V(k+1)] is 310 nanoseconds. The value is computed to be

1024 * 310 ns + (10240 – 1024)*140 ns + 660 ns = 1.608 milliseconds.

The Curve Interpolation process is not run during the Run Mode when LLRF control is active, but during the Update Mode between Booster Cycles which is approximately 20 milliseconds long.

III.7 Boot-Up and Backup Process

This process implements and manages boot-ups and backups of the Shared Memory block from and to the Flash memory, respectively. When the module is powered up and the FPGA's complete their configuration, this process will load the shared memory from the values that are stored in the Flash memory. Once the shared memory is loaded, the Shared Memory Update and the Curve Interpolation processes are run in an "update all" mode. These two processes will initialize all of the programmable parameters and curves.

This process will also backup the shared memory to a Flash memory block when triggered by the "Backup Trigger" signal.

Two separate Flash memory blocks are used to store and restore the shared memory. The object of this is to reduce the probability that the stored block of values in the Flash should be corrupted due to loss of crate power during a backup. Storage of the shared memory is made alternately to the Flash memory block at 0x0A0000 and at 0x0C0000. The last two 32 bit words in the shared memory, and in the first and second Flash memory blocks, hold an "increment". Each time the shared memory is stored to one of the Flash memory blocks, that blocks "increment" word is incremented. FLASH BLOCK 1 AT 0x0A0000 IS USED FIRST WHEN THE MODULE IS INITIALIZED.

So, when the first Flash block's increment is greater than the second Flash block's increment we will boot from the first, and the next backup will be made to the second. When a successful backup is made to the second Flash block, its increment word is incremented so its increment word will now be equal to the firsts. When the increment words are equal, we will boot from the second Flash block and store the next backup into the first.

The rest of this process is fairly simple. It sets the RAM_Addr_In input to the Flash Interface to the start of Shared memory, and the FLASH_Addr_In input to the Flash Interface to the start of the selected Flash block. It then signals a Write Flash if it is performing a backup or it signals a Read Flash if it is performing a restore or boot-up. The process then waits for the Flash Interface to complete the operation.

When this process performs a backup it will, at the end, increment the appropriate increment value stored in Shared Memory. Note that the increment value used to determine which block to restore or boot from is the increment value in the Flash blocks.

III.8 Flash Interface

III.8.1 Introduction of the Flash Memory interface

The Flash Interface is used to either write a block of RAM memory to Flash memory or read a block of Flash memory to RAM. The blocks of memory are defined by the input pointer to RAM, the input pointer to Flash and the number of words to transfer variable.

Main purpose of the Flash memory is to backup curves and parameters we do not want to lose to an unexpected power failure. Having all of the parameters and curves stored on the module allows the module to resume operation after a power outage without intervention from the front-end crate processor or any other control network function. Being Flash, whole blocks must be erased before being written (programmed). This does make the decision as to when to perform backups to Flash more complicated compared to using battery backed RAM, but it is still a manageable situation. There is also the possibility of using the Flash to configure the FPGA's, but this has not yet been developed.

The Flash memory is a complicated device compared to a simple RAM memory. In the following sections, we will outline the details of the particular component used on the DDS module. Then we will describe the methods and modes employed in the current interface. For a description of all the possible read and write modes and the other features of the chip, you need to have a look at the data sheet, "Intel StrataFlash Embedded Memory (P33)", 11/2007.

III.8.2 The Flash Memory Component Details

The Flash memory component used on the DDS module is the Intel Strata Flash, part number JS28F256P33B95. The Flash memory is organized into 4 each 16k word blocks and 254 each 64k word blocks, where a word is 16 bits. The smaller 16k word blocks are referred to as the Parameter Memory. The Flash component used has the Parameter Memory located at the lowest memory addresses. Figure III.8.2.1 is a quick reference to decode the Flash memory part number. Figure III.8.2.2 is the specific memory map for our part taken from the datasheet. Take note that the "Size (KB)" specifies thousands of bytes (8 bits), where as the hexadecimal address ranges in the table are for the address bits A[24..1] which address 2 byte words (16 bit). Hence the first parameter memory block between 0x000000 and 0x003FFF (0 to 16,383) has 32k bytes, 16k words.

Also note here, that the data to be written to flash is assumed to be 32 bit. The flash itself is 16 bits wide. The least significant word (LSW) is stored at the lower address and the most significant word (MSW) is stored at the next higher address (little-endian).

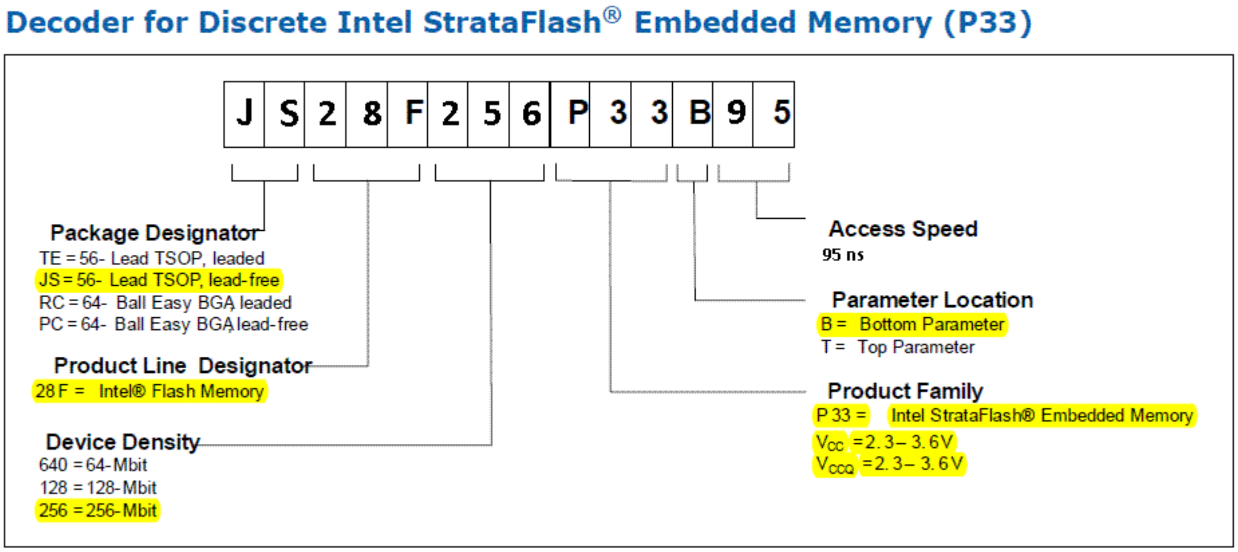


Figure III.8.2.1 Decoded Flash memory part number

	Size (KB)	Blk	256-Mbit
Thirty-One Programming Regions	128	258	FF0000 - FFFFFF
	128	257	FE0000 - FEFFFF
	⋮	⋮	⋮
	128	12	090000 - 09FFFF
	128	11	080000 - 08FFFF
One Programming Region	128	10	070000 - 07FFFF
	⋮	⋮	⋮
	128	4	010000 - 01FFFF
	32	3	00C000 - 00FFFF
	⋮	⋮	⋮
	32	0	000000 - 003FFF

Block size is referenced in K-Bytes where a byte=8 bits. Block Address range is referenced in K-Words where a Word is the size of the flash output bus (16 bits).

Figure III.8.2.2 Specific memory map for the Flash memory taken from the datasheet

III.8.3 Flash Memory Operation

The first important detail to not overlook is that at module power up the Flash memory needs a to have a 500 microsecond reset pulse on its reset input pin RST#. After the reset is released, reads or write operations may not be made for at least 150 nanoseconds. The second important detail is that after a reset, all of the memory blocks within the chip will be locked and will require "un-locking" in order to erase and program (write) the blocks. The Flash Interface will always execute commands to unlock the memory blocks to be used.

Control signal ADV#, address valid, is permanently held Low (active) preventing latching of the address at the device. Also DCLK, which is the data clock used for synchronous reads and writes is held High. Only asynchronous reads and writes are implemented for the interface.

All operations with the Flash memory are executed by first writing to the Command User Interface (CUI). The CUI does not occupy an addressable memory location; it is the mechanism through which the flash device is controlled. It is a sequence that must be followed to start the Flash memory's internal processes to execute the desired operation. Figure III.8.3.1 is a table taken from the datasheet that describes the sequence for each operation and what the address bits and data bits need to be set to during the first and possibly second write.

An industry-standard command sequence invokes program and erase automation. Each erase operation erases one block. The Erase Suspend feature allows system software to pause an erase cycle to read or program data in another block. Program Suspend allows system software to pause programming to read other locations. Data is programmed in word increments (16 bits).

Table 21. Command Bus Cycles

Mode	Command	Bus Cycles	First Bus Cycle			Second Bus Cycle		
			Oper	Addr(1)	Data(2)	Oper	Addr(1)	Data(2)
Read	Read Array	1	Write	DnA	0xFF	-	-	-
	Read Device Identifier	≥ 2	Write	DnA	0x90	Read	DBA + IA	ID
	CFI Query	≥ 2	Write	DnA	0x98	Read	DBA + QA	QD
	Read Status Register	2	Write	DnA	0x70	Read	DnA	SRD
	Clear Status Register	1	Write	DnA	0x50	-	-	-
Program	Word Program	2	Write	WA	0x40/ 0x10	Write	WA	WD
	Buffered Program(3)	> 2	Write	WA	0xE8	Write	WA	N - 1
	Buffered Enhanced Factory Program (BEFP)(4)	> 2	Write	WA	0x80	Write	WA	0xD0
Erase	Block Erase	2	Write	BA	0x20	Write	BA	0xD0
Suspend	Program/Erase Suspend	1	Write	DnA	0xB0	-	-	-
	Program/Erase Resume	1	Write	DnA	0xD0	-	-	-
Block Locking/ Unlocking	Lock Block	2	Write	BA	0x60	Write	BA	0x01
	Unlock Block	2	Write	BA	0x60	Write	BA	0xD0
	Lock-down Block	2	Write	BA	0x60	Write	BA	0x2F
Protection	Program Protection Register	2	Write	PRA	0xC0	Write	PRA	PD
	Program Lock Register	2	Write	LRA	0xC0	Write	LRA	LRD
Configuration	Program Read Configuration Register	2	Write	RCD	0x60	Write	RCD	0x03

Notes:

1. First command cycle address should be the same as the operation's target address.
DBA = Device Base Address (NOTE: needed for dual-die 512Mbit device)
DnA = Address within the device.
IA = Identification code address offset.
QA = CFI Query address offset.
WA = Word address of memory location to be written.
BA = Address within the block.
PRA = Protection Register address.
LRA = Lock Register address.
RCD = Read Configuration Register data on QUAD+ A[15:0] or EASY BGA / TSOP A[16:1].
2. ID = Identifier data.
QD = Query data on DQ[15:0].
SRD = Status Register data.
WD = Word data.
N = Word count of data to be loaded into the write buffer.
PD = Protection Register data.
LRD = Lock Register data.
3. The second cycle of the Buffered Program Command is the word count of the data to be loaded into the write buffer. This is followed by up to 32 words of data.Then the confirm command (0xD0) is issued, triggering the array programming operation.
4. The confirm command (0xD0) is followed by the buffer data.

Figure III.8.3.1 Command bus cycle specification table

Reading the data or status register in the Flash memory is fairly simple. After a reset, Asynchronous Page Mode is the default for reading from the Flash. This is the mode currently employed by the Flash Interface. The entire Read Configuration Register (RCR) defaults to a state that we can use and will not be considered. There are four read states, but only two types of reads that are of interest. The first is Array Read and the second is Status Register Read. Before reading data from the Flash, using the Array Read, the command value 0x00FF is written to the Flash to put the device in Array Read Mode. Then addressing and reading the data is just like reading from RAM. The access time between a valid address to valid data out that can be latched is 95 nanoseconds.

Reading the Status Register is done periodically to see if certain operations with the Flash have completed. This includes unlocking, erasing, and buffered programming. The convenient thing is that after issuing one of these commands the Flash goes into a Read Status Register state. In this state the Status Register is updated and latched onto the Flash data bus on the falling edge of OE# or CE# (whichever occurs first). Bit 7 of the Status Register indicates whether the previous operation has completed and the Flash is "ready" or it is "busy". If the process is found to be busy we do need to toggle OE# and CE# again so the Status Register updates.

Status Register Description

Status Register (SR)							**Default Value = 0x80**
Device Write Status	Erase Suspend Status	Erase Status	Program Status	V_{PP} Status	Program Suspend Status	Block-Locked Status	BEFP Write Status
DWS	ESS	ES	PS	VPPS	PSS	BLS	BWS
7	6	5	4	3	2	1	0

Bit	Name	Description
7	Device Write Status (DWS)	0 = Device is busy; program or erase cycle in progress; SR[0] valid. 1 = Device is ready; SR[6:1] are valid.
6	Erase Suspend Status (ESS)	0 = Erase suspend not in effect. 1 = Erase suspend in effect.
5	Erase Status (ES)	0 = Erase successful. 1 = Erase fail or program sequence error when set with SR[4,7].
4	Program Status (PS)	0 = Program successful. 1 = Program fail or program sequence error when set with SR[5,7]
3	V_{PP} Status (VPPS)	0 = V_{PP} within acceptable limits during program or erase operation. 1 = $V_{PP} < V_{PPLK}$ during program or erase operation.
2	Program Suspend Status (PSS)	0 = Program suspend not in effect. 1 = Program suspend in effect.
1	Block-Locked Status (BLS)	0 = Block not locked during program or erase. 1 = Block locked during program or erase; operation aborted.
0	BEFP Write Status (BWS)	After Buffered Enhanced Factory Programming (BEFP) data is loaded into the buffer: 0 = BEFP complete. 1 = BEFP in-progress.

Figure II.8.3.2 Flash Status Register description

Writing, referred to as programming, is more involved. The Flash address bus is set to the base address of the block to be programmed. After a reset, or after power up, the block to be programmed must be unlocked by writing the command code 0x0060 followed by the confirm command code 0x00D0. Then the block is erased by writing the command code 00x0020 followed by the confirm command code 0x00D0.

<u>The Flash interface needs to release the UPD data bus while the erase is in progress. An erase may take 4 seconds. Once the erase is completed, the process will need to wait for a "bus permit" to proceed.</u>

After the block is erased, buffered programming is performed. Buffered programming begins by writing the command code 0x00E8 to the Flash. The Status Register is then checked to see if the program buffer is available (ready). If not, the command code 0x00E8 is written again and the Status Register is checked again. Once the program buffer is ready we write, to the Flash, a number representing the number of words we wish to write to the buffer minus 1 (N – 1). This number is 31 when we are going to write the whole buffer.

On the next write, a device start address is given along with the first data to be written to the flash memory array. Subsequent writes provide additional device addresses and data. All data addresses must lie within the start address plus the word count. After the last data is written to the buffer, the Buffered Programming Confirm command, 0x00D0, must be issued to the original block address. The Write State machine (WSM) begins to program buffer contents to the flash memory array. The Status Register is again monitored to determine when the programming of the buffer is completed. The buffer programming process is repeated until all of the data we wished to program into the Flash memory block has been programmed in.

During a Shared Memory backup, the buffer will be written 1,280 times. The maximum buffer programming time is 880 microseconds. The programming process alone may take as long as 1.13 milliseconds.

III.9 DDS Setup and Control

III.9.1 AD9910 Serial Interface

This is mainly logic and timing for writing the serial control interface for the four AD9910 DDS components. This module has four main parts

1. There is a 72 bit shift register that is parallel loaded when the state switches from idle to first_bit, and shifts data out on each clock when in the send state.

2. There is a state machine with 4 simple states; idle, first_bit, send, update.
3. There is a shift register counter to signal when the desired number of bits (BitCount) has been sent.
4. Finally, there are strings of flip-flops to synchronize the final data update signals. The registers in the VHDL process SYNC_IO_UPDATE synchronizes the IO_Update signal to the Sync Clock signal output of each DDS. Registers SYNC_TXDONE takes the final output of SYNC_IO_UPDATE and re-synchronizes this edge to the logic clock, SR_Clk, to produce a transmission complete signal, tx_done. tx_done is also sent out as the IO Reset signal when the IO Reset signal is desired.

The shift register accommodates as many as 64 bits of DDS configuration registers and the 8 bit Instruction Byte. The Instruction Byte specifies whether the transfer is initiating a register read or write, and provides the bits for addressing the desired register.

The DDS Frequency and Phase words are loaded through this serial interface. The requirement is to update both the Frequency and Phase words every microsecond. In order to do this, a *partial load* of the Single Tone Profile 0 register is made, loading only the 32 bit Frequency Tuning Word (FTW) and the 16 bit Phase Offset Word (POW). This is a total of 56 bits including the Instruction Byte that are serially shifted out to the DDS's with a 68 MHz serial data clock. Transmission of the data takes 0.8235 microseconds. Following each transmission the interface provides a properly synchronized IO Update pulse and an IO Reset pulse to terminate the partial serial load.

The IO Update signal causes the new frequency and phase settings to be loaded into the DDS's. It is important to update all four DDS's at the same time or we could introduce a phase shift between the DDS's we did not intend. The DDS's register the IO Update signal using the Sync Clock that is derived on the DDS as $1/4^{th}$ the frequency of the System Clock. The typical Sync Clock runs at 120 MHz, a period of 8.333 nanoseconds. Each DDS outputs its own Sync Clock for use in synchronizing the IO Update signal it will receive. The Sync Clocks on each chip are synchronized through another feature of the DDS devices explained elsewhere in this document.

Figure III.9.1.1 illustrates the logic currently used for synchronizing the IO Update signals for the four DDS's. It seems that there may be a redundancy somewhere in all this, but this is making it work at the moment. Further effort on this bit will take place in the near future. The concept behind all of this is that the time offset between the various Sync Clocks is unknown. The DDS multi-chip synchronization circuits are expected to keep the offsets small. However, wish to avoid having one Sync Clock register the IO Update a full clock period before another. Therefore the IOU_Syncd signal, in Figure III.9.1.1, does not go high until all of the Sync Clocks have registered the IO_Update_Enable signal, and the final IO_Update signals will not go low until the last Sync Clock has registered the end of the pulse.

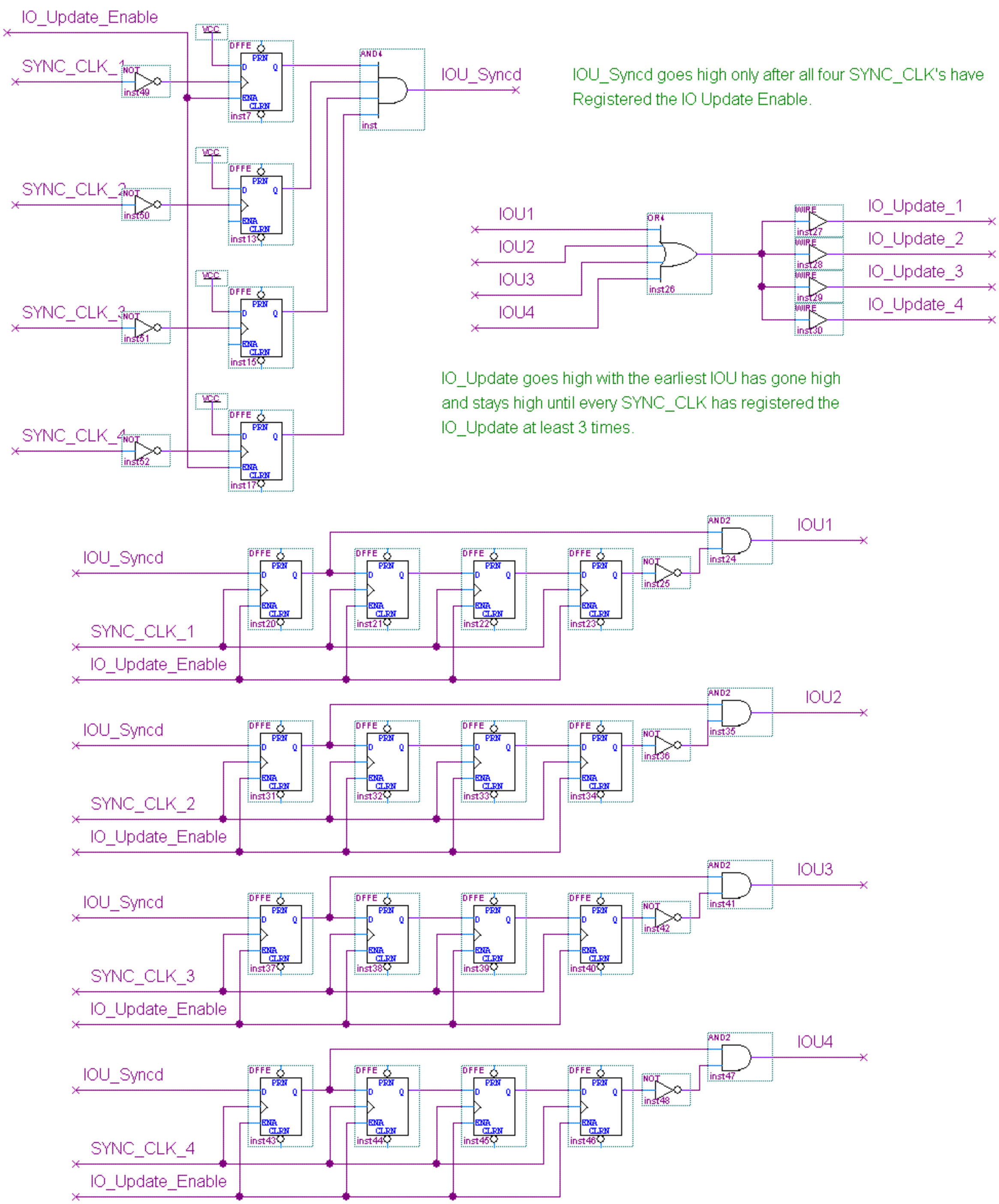


Figure III.9.1.1 Block diagram of the logic used for synchronizing the IO Update signals

III.9.2 Setting Up the DDS Components

This module implements the procedures which setup the DDS registers to get the DDS channels up and running. A ROM memory is established that contains the setup to accomplish this initial configuration of the

DDS's. This is in a memory initialization file (*.mif) pointed to by the Altera Memory LPM Megafunction. The DDS setup process increments through this memory and acts on the setup words.

The setup words in the ROM memory are 48 bits long. The first 8 bits are control bits as described in Table III.9.2.1. These bits allow DDS setup actions to be coded into the list of setup words. The second set of 8 bits is the Instruction Byte defined by the DDS datasheet and described in Table III.9.2.2. The final 32 bits will be what is loaded into the controls registers of the DDS's, if a control register load is called for.

Table III.9.3 gives an example of setup word memory contents and the action taken when these words are loaded into the setup process.

Table III.9.2.1 Description of the control bits in the setup word

Bit	47	46	45	44	43	42	41	40
Name	MSB_nLSB	Do_Reset	Ena_IORESET	Global_nAddr	Pause	reserved	CS1	CS0

MSB_nLSB	Informs the process logic how to pack the Instruction Byte and data into the serial output shift register for transmission. A high indicates the MSB First mode is to be used. A low indicates the LSB First mode is to be used.
Do_Reset	This signals the FPGA logic that the DDS Master Reset is to be pulsed.
Ena_IORESET	This bit causes the IORESET bit to be enabled during the transmission of this setup word. The IORESET is a pulse that is sent at the end the serial transmission to terminate a partial DDS register write. In the case of the DDS setup logic we note that the DDS CFR configuration registers are 32 bits long. The Single Tone Profile registers in which we can write the Amplitude, Phase, and Frequency words are 64 bits long. The trick used for the setup and later in writing the frequency and phase updates is to write only a portion of the profile register, starting from the LSB of the register. For more details see the datasheet page 49-51.
Global_nAddr	Indicates to the process logic that either all of the DDS channels should receive the following setup word or only one of the DDS channels addressed by [CS1,CS0] should receive it. When high all DDS's are written with the same data.
Pause	When set the process logic pauses for 750us before reading the next word from the setup memory. Note the bits in the current word, following the Pause bit, are not transmitted.
CS1, CS0	These bits are used to address a specific DDS channel for reception of the setup word when the Global_nAddr bit is set low.

Table III.9.2.1 Description of the DDS Serial Interface Instruction Byte in the setup word

Bit	39	38	37	36	35	34	33	32
Name	Read_nWrite	x	valid	A4	A3	A2	A1	A0

Read_nWrite	When high it indicates that the FPGA wishes to read from the addressed register. When low it indicates that the FPGA is writing the register. For this setup procedure this bit is always low.
x	This bit is ignored (don't care).
valid	This bit is actually specified by the AD9910 datasheet as another don't care bit. In the Setup word, it is used to signal the process logic that the current setup word is valid when set high. When a setup memory word is encountered with this bit low, the setup is considered finished.
A[4..0]	These bits address the registers inside the DDS's.

Table III.9.2.3 Example of the setup word ROM (1/7/2010)

ROM Memory Address (binary)	ROM Memory Data (hexadecimal)	Action taken by the DDS Setup Process
00000	D820 0000 0000	A 750us Master Reset is signaled for all channels. The pause bit is included with the Do_Reset bit.
00001	9820 0000 0000	A 750us pause is implemented after the reset.
00010	9022 183F 4160	The PLL control register, CFR3 (addr = 0x02) is setup. See the AD9910 data sheet, Pages 58 Table 20, for details of the PLL setup.
00011	9820 0000 0000	Another 750us pause is made to let the PLL settle.
00100	802A 0C00 0000	The multi-chip sync register (addr = 0x0A) is setup for the sync master DDS (channel 1). Note that the Global_nAddressed bit is 0.
00101	812A 0800 0000	The multi-chip sync register (addr = 0x0A) is setup for the sync slave DDS's (this includes channel 1 also).
00110	9020 0000 2001	The two important bits are set in control register CFR1(addr = 0x00) These are bit 13, "Auto-clear phase accumulator" and bit 0 which causes the DDS's to expect the subsequent serial loads to be performed LSB first.

00111	302E 0555 5555	The Single Tone Profile 0 register (addr = 0x0E) is setup with its initial Frequency word. It is assumed that the profile control pins of the DDS's are set to use Profile 0. Note that the MSB_nLSB bit is now low indicating that the Instruction Byte and the data will be sorted properly for loading the serial data LSB first. This is in agreement with us setting the LSB First bit in register CFR1 in the previous step.
01000	302E 0555 5555	The loading of the Profile 0 register is repeated twice more to ensure all channels' phase accumulators have sync'd.
01001	302E 0555 5555	--
01010	1020 0000 0000	Register CFR1 is written again to release the Auto-clear phase accumulator bit causing the accumulators in the DDS's to be synchronized. Also the LSB First bit (bit 0) is cleared to return to MSB First serial IO mode.
01011	0000 0000 0000	The setup word "valid", bit 37, is cleared indicating to the process that the setup is complete.

III.9.3 DDS Reference Clock Distribution and Multi-Chip Synchronization

In order to utilize multiple DDS outputs for the Booster LLRF reference and control, we must be able to reliably synchronize all four DDS channels on the module to one another. First, the reference clock to each DDS must be edge-align with one another. This is accomplished using the National Semiconductor LMK01010 Clock Buffer and Distribution component. This component takes a single reference clock from the FPGA and fans out the clock to each DDS. Each clock output also has a programmable delay in steps of 150 picoseconds up to 2250 picoseconds. The delay applied to the reference clock to each DDS is hardcoded within the LMK01010 Interface process. The process is setup to run at power up or upon a reset.

With the reference clocks aligned, the assumption is that we are going to have aligned system clocks at the output of the PLL clock multipliers in each DDS. Currently the Upper FPGA generates a 10 MHz reference clock that is multiplied up to 480 MHz within each DDS.

Internal to each DDS is a state machine that controls the sequence of processing within them. This state machine is clocked by the system clock. In order for the DDS's to be synchronized, the system clocks must transition the state machines in each DDS at the same time and the state machine in each DDS needs to be in the same state. The multi-chip sync logic in the AD9910 provides a way of starting (actually resetting) the state machines in all DDS together. Each DDS can produce a SYNC_OUT clock that is 1/16th the frequency of the system clock. We use the SYNC_OUT output of one of the DDS's as the master sync and distribute this clock to the SYNC_IN inputs of every DDS, including the one that generated the sync pulse. The multi-chip sync logic provides a programmable delay for each SYNC_IN as well as the SYNC_OUT pulse. The SYNC_OUT pulse is buffered and fanned out to the SYNC_IN inputs using a second LMK01010 Clock Buffer and Distribution component. The SYNC_IN input of each DDS produces a single system clock wide pulse that resets the state machine of that DDS. Refer to the AD9910 datasheet section "Synchronization of Multiple Devices" for more details.

The final step in keeping the DDS's synchronized is to register the IO Update pulse of the serial input interface at the same time. The logic involved here has been described in the section covering the AD9910 Serial Interface.

III.9.4 Read Back of DDS Status Bits

Each of the four DDS components has two status outputs of particular interest. The PLL Lock signal indicates if the PLL multiplier has properly locked to the reference clock input. Synchronization Sample Error indicates whether a properly timed SYNC_IN clock was received for the multi-chip synchronization logic. In order to save IO pins on the Upper FPGA, these status bits are read back to the FPGA via a simple shift register.

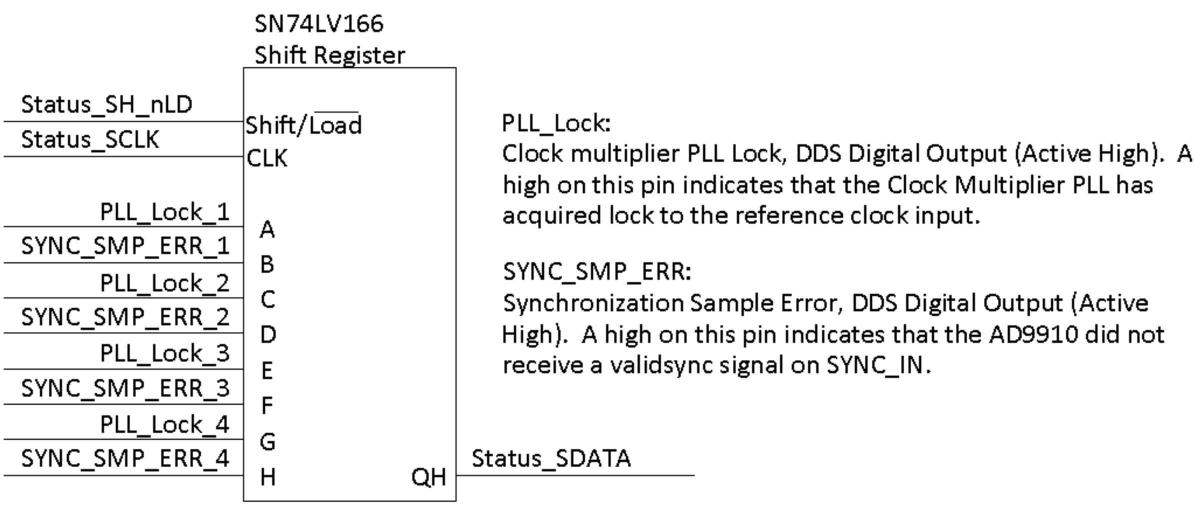


Figure III.9.4.1 Illustration of the DDS Status shift register read back.

The following is taken from the AD9910 data sheet to explain the meaning and use of the Synchronization Sample Error status bit (SYNC_SMP_ERR).

> *The synchronization mechanism depends on the reliable generation of a sync pulse by the edge detection block in the sync receiver. Generation of a valid sync pulse, however, requires proper sampling of the rising edge of the delayed SYNC_IN signal with the rising edge of the local SYSCLK. If the edge timing of these signals fails to meet the setup or hold time requirements of the internal latches in the edge detection circuitry, then the proper generation of the sync pulse is in jeopardy. The setup and hold validation block (see Figure III.9.4.1) gives the user a means to validate that proper edge timing exists between the two signals.*

The setup and hold validation block can be disabled via the sync timing validation disable bit in Control Function Register 2.

The validation block makes use of a user-specified time window (programmable in increments of ~150 ps via the 4-bit sync validation delay word in the multichip sync register). The setup validation and hold validation circuits use latches identical to those in the rising edge detector and strobe generator. The programmable time window is used to skew the timing between the rising edges of the local SYSCLK signal and the rising edges of the delayed SYNC_IN signal. If either the hold or setup validation circuits fail to detect a valid edge sample, the condition is indicated externally via the SYNC_SMP_ERR pin (active high).

The user must choose a sync validation delay value that is a reasonable fraction of the SYSCLK period. For example, if the SYSCLK frequency is 1 GHz (1 ns period), then a reasonable value is 1 or 2 (150 ps or 300 ps). Choosing too large a value can cause the SYNC_SMP_ERR pin to generate false error signals. Choosing too small a value may cause instability.

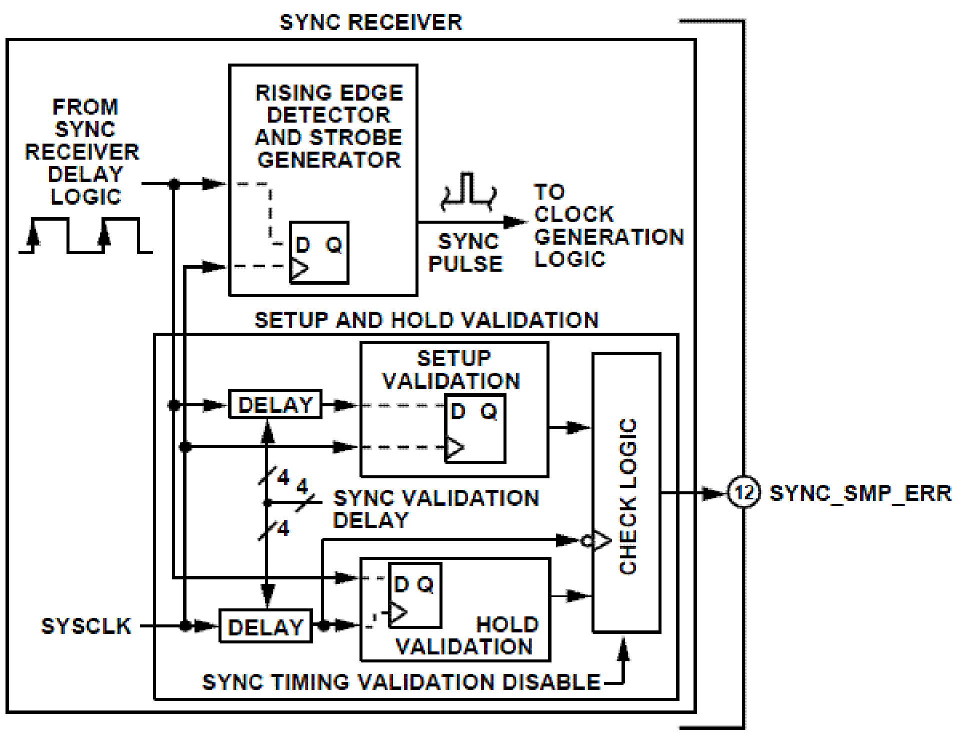


Figure 54. Sync Timing Validation Block

Figure III.9.4.1 Figure from AD9910 data sheet describing the Sync Sample Error function.

III.10 AD7625 Analog to Digital Converter Interface

The AD7625, 16 bit analog to digital converter was chosen for the first prototype for several reasons. The device is low power, only 135 milli-Watts. It is uses a serial output so it uses only a quarter of the IO pins on the FPGA than a parallel output, and it uses LVDS for the signals so it is can update fast and still be low noise. It also has low data latency. The delay between sampling of the analog input and the digital result is less than 300 nanoseconds.

There are two parts to the interface. The ADC Interface block generates two input signals for all four of the AD7625's. The signal ADC convert signal, ADC_CNV, triggers the ADC to start a conversion, and the ADC data clock signal, ADC_CLK, is used for serially clocking data out of the ADC. Figure III.10.1 provides a timing diagram for the following description. See the AD7625 datasheet for full details of the ADC's digital interface (rev0|Page 19 of 24).

The timing in this logic reads data from the ADC at a 5 MHz rate. A new conversion is signaled approximately mid-way through reading out the previous conversion. The data clock is provided at 100 MHz. This is produced with a PLL that multiplies the 50 MHz logic clock by 2. The data clock must be gated on and off with the specified timing relationship with the convert pulse. The timing is accomplished using a counter (timer) and a state machine.

Gating the data clock has some details. The counter/timer and state machine, which produces the data clock gate, is synchronized to the global Logic Clock (50 MHz). This gate is re-synchronized to the PLL output clock (100 MHz) before being used to gate the PLL output clock as the data clock.

NOTE: Timing requires that there be a minimum of 145ns between the rising edge of the ADC convert signal and the first rising edge of the gated data clock. Also the last falling edge of the gated data clock must occur within 110ns after the rising edge of the next ADC convert pulse.

The second process in the interface is a dual (ping-ponged) shift register for receiving serial data from the AD7625. One shift register accepts the serial stream at the same moment the other shift register is holding the last transmitted value for the next stage of data processing. Once a shift register has filled, the roles of the two registers switch. There is one shift register for each ADC.

The shift register operation is controlled with a 5-bit counter. The counter's lower 4-bits count the 16 serial data bits shifted into the registers. The 5th bit controls which of the two shift registers is receiving the serial stream and which is tied to the parallel data output of the module.

The data output strobe is generated by detecting the end of the ADC_CLK clock burst that fills the shift register. A two bit counter increments on the rising edge of the 50 MHz logic clock, but asynchronously resets during the logic high portion of the 100 MHz ADC_CLK. If the counter is incremented by a first logic clock edge, it must be found un-reset on the next logic clock edge. If so, the output strobe signaling new ADC data is fired.

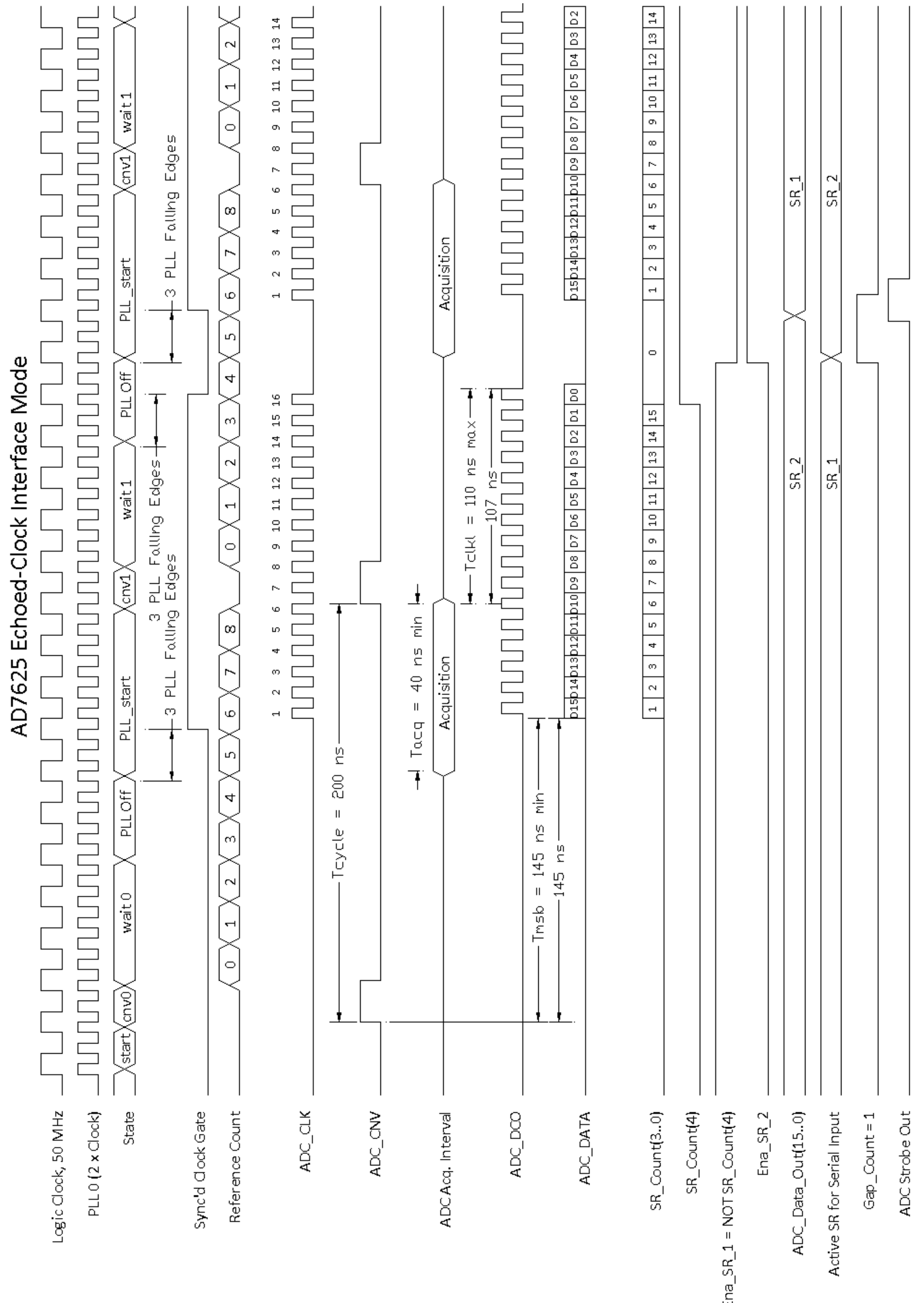
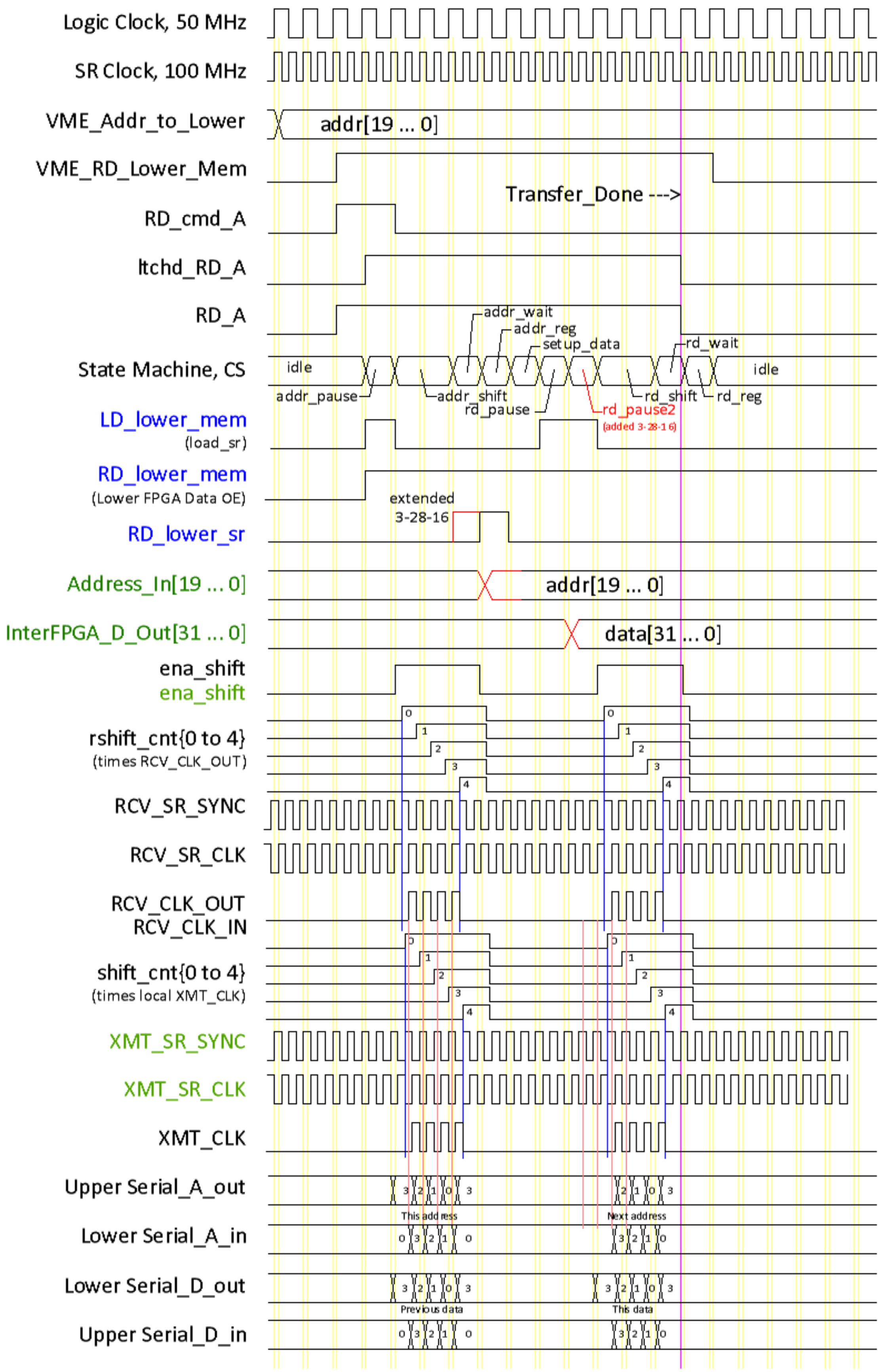
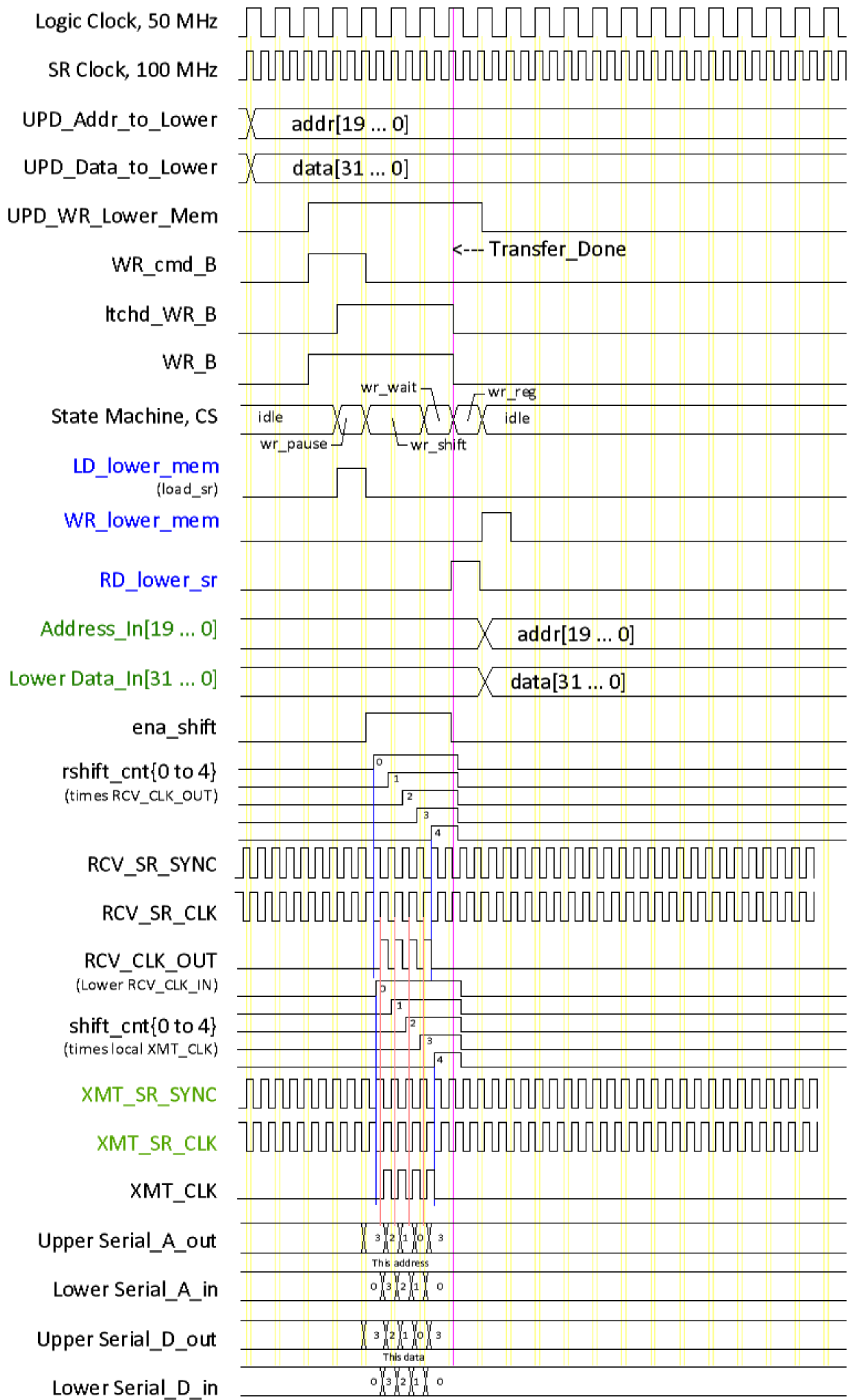


Figure III.10.1 The Analog Input ADC Interface Timing



Logic Clock, 50 MHz
SR Clock, 100 MHz
VME_Addr_to_Lower
addr[19 ... 0]
VME_RD_Lower_Mem
Transfer_Done --->
RD_cmd_A
ltchd_RD_A
RD_A
addr_wait
addr_reg
setup_data
rd_wait
State Machine, CS
idle
idle
addr_pause
addr_shift
rd_pause
rd_shift
rd_reg
rd_pause2
(added 3-28-16)
LD_lower_mem
(load_sr)
RD_lower_mem
(Lower FPGA Data OE)
extended
3-28-16
RD_lower_sr
Address_In[19 ... 0]
addr[19 ... 0]
InterFPGA_D_Out[31 ... 0]
data[31 ... 0]
ena_shift
ena_shift
rshift_cnt{0 to 4}
(times RCV_CLK_OUT)
RCV_SR_SYNC
RCV_SR_CLK
RCV_CLK_OUT
RCV_CLK_IN
shift_cnt{0 to 4}
(times local XMT_CLK)
XMT_SR_SYNC
XMT_SR_CLK
XMT_CLK
Upper Serial_A_out
This address
Next address
Lower Serial_A_in
Lower Serial_D_out
Previous data
This data
Upper Serial_D_in


Logic Clock, 50 MHz
SR Clock, 100 MHz
UPD_Addr_to_Lower
addr[19 ... 0]
UPD_Data_to_Lower
data[31 ... 0]
UPD_WR_Lower_Mem
WR_cmd_B
<--- Transfer_Done
ltchd_WR_B
WR_B
State Machine, CS
idle
wr_pause
wr_wait
wr_shift
wr_reg
idle
LD_lower_mem
(load_sr)
WR_lower_mem
RD_lower_sr
Address_In[19 ... 0]
addr[19 ... 0]
Lower Data_In[31 ... 0]
data[31 ... 0]
ena_shift
rshift_cnt{0 to 4}
(times RCV_CLK_OUT)
RCV_SR_SYNC
RCV_SR_CLK
RCV_CLK_OUT
(Lower RCV_CLK_IN)
shift_cnt{0 to 4}
(times local XMT_CLK)
XMT_SR_SYNC
XMT_SR_CLK
XMT_CLK
Upper Serial_A_out
This address
Lower Serial_A_in
Upper Serial_D_out
This data
Lower Serial_D_in