A New Argus Direct Conversion Receiver and Digital Array Receiver/Processor

Grant Hampson and Steve Ellingson

ElectroScience Laboratory - Ohio State University
1320 Kinnear Road, Columbus, Ohio, 43221

hampson.8@osu.edu, ellingson.1@osu.edu

October 3, 2002

Contents

1 The New Argus Array Receiver

This document describes a new system architecture for Argus. The new architecture has two main goals: low cost design and an architecture which is scaleable in bandwidth as well as number of elements. These two design goals have been met with success. Figure 1 illustrates the basic components of the new Argus system. The new architecture consists of four main components: a Direct Conversion Receiver (DCR), a Digital Receiver/Processor (DRP), a backplane, and a Backplane Controller Interface (BCI). The current design is capable of processing up to 64 (N=64) simultaneous RF inputs from an antenna array with up to 100 kSPS@32-bit digital outputs, for each RF input. The cost of each stage (including components and PCB) is also shown in Figure 1 for reference. The approximate total cost per RF-input is approximately $250.

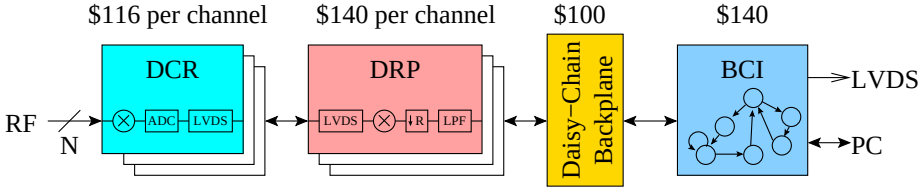


Figure 1: Block diagram of the new ARGUS system architecture.

The DCR has the function of down-converting an RF input signal of frequency 1-2GHz using a low cost direct conversion tuner. This results in 14MHz of bandwidth which is then converted using a low cost dual 8-bit Analog-Digital-Converter (@20 MSPS complex.) The ADC parallel data is serialized into a Low Voltage Differential Signal (LVDS). This serial data stream (400Mbps) can be transported over low cost CAT5 cable to the DRP. The distance between the DCR and DRP can be up to two metres. The DRP transmits the ADC clock via the same CAT5 cable also using LVDS, which simplifies the clock distribution network.

The DRP converts the LVDS serial data stream into a parallel form. There are two possible outputs of the DRP: full bandwidth data of 20 MSPS, or a reduced bandwidth of less than 100 kSPS. The full bandwidth data is useful for diagnostics - such as measuring I-Q imbalance. The I-Q imbalance can then be corrected using the on board FPGA. The bandwidth is reduced using a Digital Down-Converter (DDC.) The DDC can tune digitally to a desired frequency, and then perform decimation and filtering. All the DDCs in the system are synchronized together. The DDCs are programmed across the backplane.

The backplane of the Argus architecture is based on a single LVDS daisy chain. Each DRP contains a LVDS daisy chain input and output. The DRP contains state machines to decode the daisy chain data and act accordingly. The backplane data bus is reduced two traces with a differential impedance of 100Ω. Additionally, there are four LVDS traces which distribute the clock and DDC synchronization signals, as well as power. However, daisy chains do have an inherent disadvantage that if one element in the chain fails then the entire chain fails. It is easy to detect the failed element though.

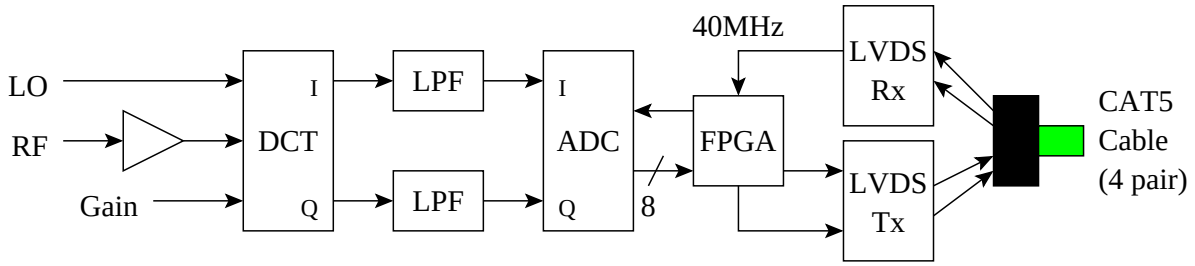
The BCI provides a control interface to a PC containing a digital I/O card. The PC can then program the DDCs and control the output of each DRP. The PC can capture the data for software post processing, or alternatively the data can be post-processed in hardware. Since all data is present on a single LVDS cable it is possible to have multiple real-time hardware beamformers, RFI processors, spectrometers, data recorders, etc..

2 Direct Conversion Receiver

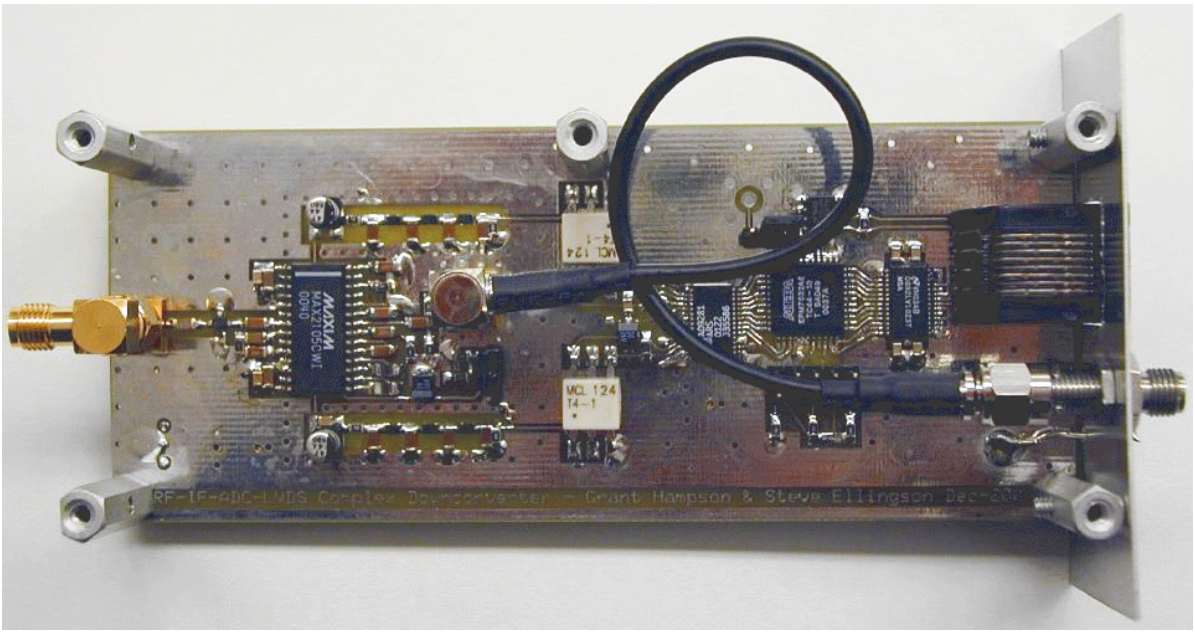
The DCR is first of two stages in the new Argus receiver chain. The DCR is designed to be a low cost interface between the RF and digital domains. A block diagram and photograph of the DCR is illustrated in Figure 2.

The DCR RF-input is has an initial gain stage of 18dB using a ERA-5SM RF amplifier [1], which is then fed to a Direct Conversion Tuner (DCT). The DCT is a MAX2105 [2], and can be tuned over the frequency range of 1-2GHz. If strong RFI is present in the RF input, the DCT AGC fluctuates resulting in unpredictable data. A recommended strategy to avoid this situation is to band limit the RF input. For example, at the ESL laboratory strong interference is present around 1 and 2 GHz. Consequently, the RF input is pre-filtered with a 1200-1800 MHz band pass filter (external to the DCR.)

The amount of gain provided by the DCT is programmable through an analog control voltage. The circuit board provides two options for this voltage, the first is a fixed voltage divider of 2.5V (near maximum gain) and the second is to provide a control voltage to two header pins. The gain setting has little effect on the DCT's response to strong RFI.



(a) Block diagram of the DCR



(b) Photograph of the DCR (without enclosure)

Figure 2: (a) Block diagram and (b) photograph of the DCR.

The LO frequency tunes to the centre frequency of interest, and the DCT output bandwidth is approximately 100MHz. This bandwidth is reduced to 14MHz, using two low pass filters with a cut off frequency of 7MHz. The measured frequency response of this filter is shown in Figure 3. This filter also acts as an anti-aliasing filter for the ADC.

The DCR uses a dual 8-bit ADC to convert the two IF channels. The ADC chosen is the AD9281 [3] which is optimized for situations requiring close matching between two ADCs. The sampling rate of the ADC is set to 20MHz which is sufficient to sample the 14MHz of bandwidth.

The sampling clock is received from the CAT5 cable using a LVDS-to-CMOS level converter (90LV018 [4].) The clock frequency received is double the ADC sampling frequency, 40MHz, and consequently the FPGA contains code to divide this frequency by two. The ADC uses a level-sensitive input to multiplex I and Q data over the same 8-bit data bus. The effective output rate is then 40 Mbytes/second.

The LVDS serializer used is the DS92LV1023 [5] which operates on 10-bit data. The two most significant bits of the LVDS data are set to the state of ADC I/Q select line, such that any devices down the processing chain will know which samples are I and Q. The FPGA contains code to drive the clock and select lines of the ADC, as well as coordinating the timing of the LVDS serializer. Refer to Appendix A for the source code to this FPGA. The FPGA used is an Altera EPM7032AE [6] which is a 3.3V device.

The LVDS serializer uses a total of 12-bits (additional start and stop bits) to transmit the ADC data at a clock frequency of 40MHz. This represents a frequency of 480Mbps. An advantage of using LVDS is that there is minimal electromagnetic interference (EMI) generated [7].

Of particular concern with the LVDS serializer is Repetitive Multi-Transition (RMT), where by the data bits are confused with the start and stop bits. RMT can be avoided by implementation of 8b/10b encoders, but this requires significant FPGA encoding/decoding. In this situation there is sufficient Gaussian noise present that the probability of RMT is sufficiently low to ignore it.

For reference a schematic, Figure 4, and bill of materials, Table 1, are included.

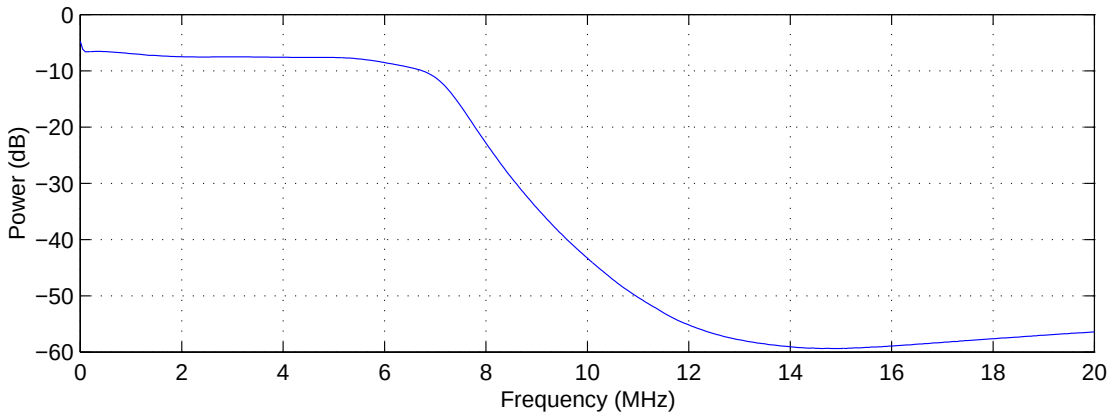
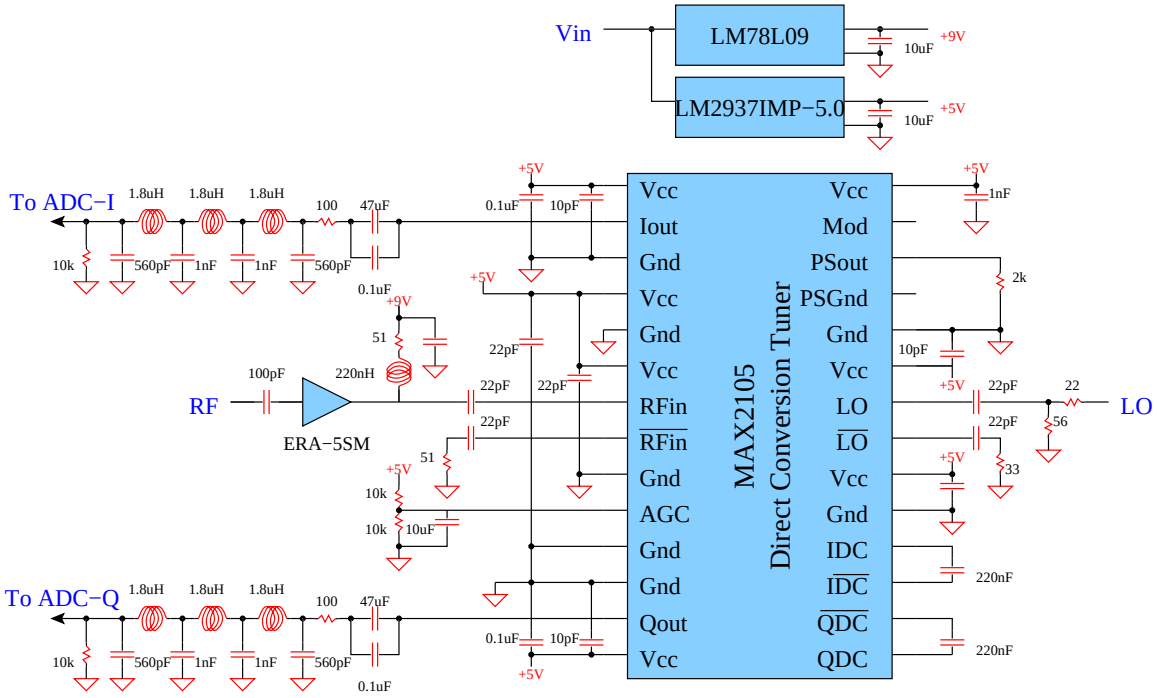
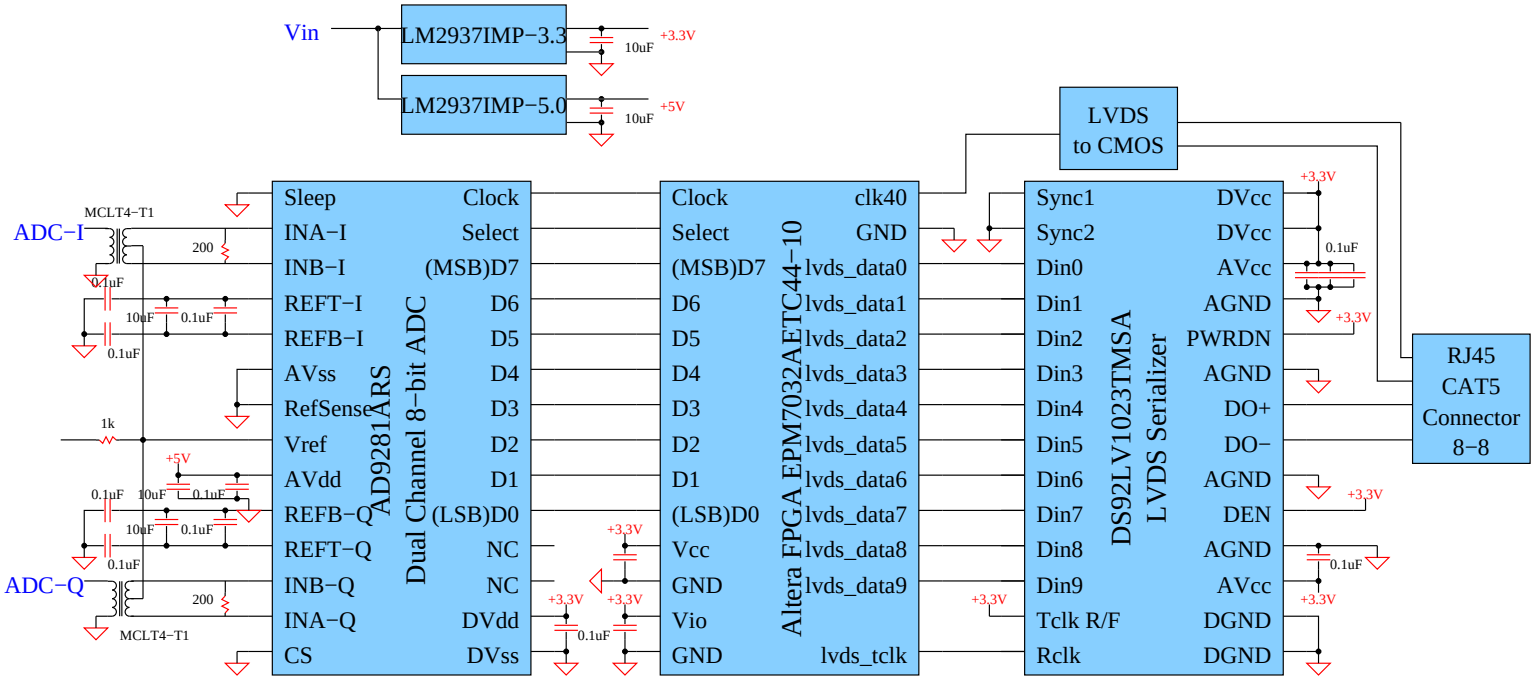


Figure 3: Direct Conversion Receiver ADC anti-aliasing filter measured response (with spectrum analyzer.) The half the ADC sampling frequency ($F_s/2 = 10$MHz) the anti-aliasing filter has approximately 37dB of relative attenuation. The cutoff frequency is approximately 7MHz, giving the overall bandwidth of the system to be 14MHz.



(a) Schematic of the DCR RF front-end



(b) Schematic of the Digital back-end of the DCR

Figure 4: Schematic of the DCR

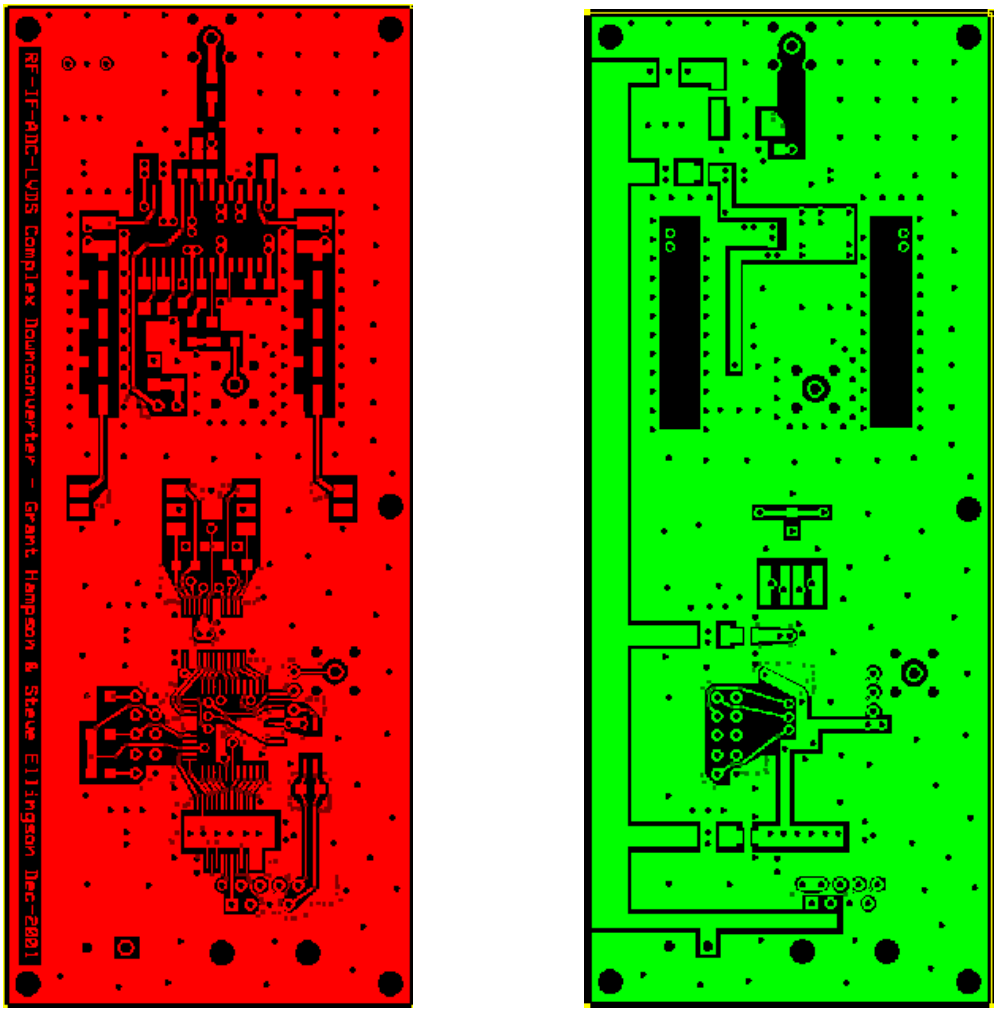
Table 1: Bill of Materials for the Direct Conversion Receiver

Component	Value	Quantity	Supplier	Unit Cost	Total Cost
Circuit board	ad9281rev2.pcb	1	Express PCB	$19.37	$19.37
Integrated Circuits	MAX2105CWI	1	Avnet	$6.92	$6.92
	AD9281ARS	1	Future Active	$5.93	$5.93
	EPM7032AETC44-10	1	Arrow	$2.00	$2.00
	DS92LV1023TMSA	1	Arrow	$13.37	$13.37
	DS90LV018ATM	1	Arrow	$1.33	$1.33
	LM2937IMP-3.3CT-ND	1	Digikey	$1.33	$1.33
	LM2937IMP-5.0CT-ND	2	Digikey	$1.33	$2.66
	LM78L09ACZ-ND	1	Digikey	$0.39	$0.39
Connectors	ARFX1232-ND (RA)	1	Digikey	$4.10	$4.10
	ARFX1231-ND (vert.)	2	Digikey	$2.40	$4.80
	A9044-ND (RJ45:8-8)	1	Digikey	$0.64	$0.64
	929836-09-36-ND (unpr. Hdr.)	1	Digikey	$0.50	$0.50
	281-1435-ND (5mm conn)	1	Digikey	$0.51	$0.51
Resistors	P22ACT-ND (22Ω)	1	Digikey	$0.04	$0.04
	P33ACT-ND (33Ω)	1	Digikey	$0.04	$0.04
	P51ACT-ND (51Ω)	2	Digikey	$0.04	$0.08
	P56ACT-ND (56Ω)	1	Digikey	$0.04	$0.04
	P75ACT-ND (75Ω)	2	Digikey	$0.04	$0.08
	P100ACT-ND (100Ω)	1	Digikey	$0.04	$0.04
	P200ACT-ND (200Ω)	2	Digikey	$0.04	$0.08
	P1.0KACT-ND (1$k\Omega$)	4	Digikey	$0.04	$0.16
	P2.0KACT-ND (2$k\Omega$)	1	Digikey	$0.04	$0.04
	P10KACT-ND (10$k\Omega$)	2	Digikey	$0.04	$0.08
	P51XCT-ND (1W)(51Ω)	1	Digikey	$0.88	$0.88
Capacitors	P9878CT-ND (EMI filter)	4	Digikey	$0.70	$2.80
	PCC100CCT-ND (10pf)	4	Digikey	$0.08	$0.32
	PCC220CCT-ND(22pf)	6	Digikey	$0.08	$0.48
	PCC561BCT-ND (560pF)	4	Digikey	$0.07	$0.28
	PCC102BCT-ND(1000pF)	5	Digikey	$0.08	$0.40
	PCC1812CT-ND(.1uF)	23	Digikey	$0.07	$1.61
	PCS3106CT-ND (10uF)	9	Digikey	$0.54	$4.86
	PCF1128CT-ND (.22uF)	2	Digikey	$0.56	$1.12
	PCE2024CT-ND (47uF)	2	Digikey	$0.31	$0.62
Inductors	PCD1176CT-ND (.22uH)	1	Digikey	$1.15	$1.15
	PCD1190CT-ND (1.8uH)	6	Digikey	$1.45	$8.70
RF Components	T4-1-KK81	2	MiniCircuits	$3.25	$6.50
	ERA-5SM	1	MiniCircuits	$3.90	$3.90
Enclosure	Aluminum Tube	1	OnlineMetals.com	$2.00	$2.00
	Barrel SMA	1	Jameco	$3.25	$3.25
	SMA Lead Assembly	1	Jameco	$3.95	$3.95
	Feed-thru capacitor	1	Digikey	$1.00	$1.00
	Right angle SMA	1	Jameco	$3.29	$3.29
	4-40 screws	8	Jameco	$0.16	$1.28
	1” standoff	5	Jameco	$0.35	$2.80
Total Part Count		121	Total BOM Cost		$115.72

2.1 DCR Enclosures

Figure 5 illustrates the PCB for the DCR. A problem arose that adjacent channels could radiate energy between circuit boards. A solution to the problem was to enclose the DCRs. The simplest and most economical way of enclosing the circuit boards was to use aluminum rectangular tubing (available from `www.onlinemetals.com`.) The tubes were cut to the correct length and the circuit board supported by standoffs inside the tube. Two end plates have holes for RF-input, power, CAT5 cable and the LO.

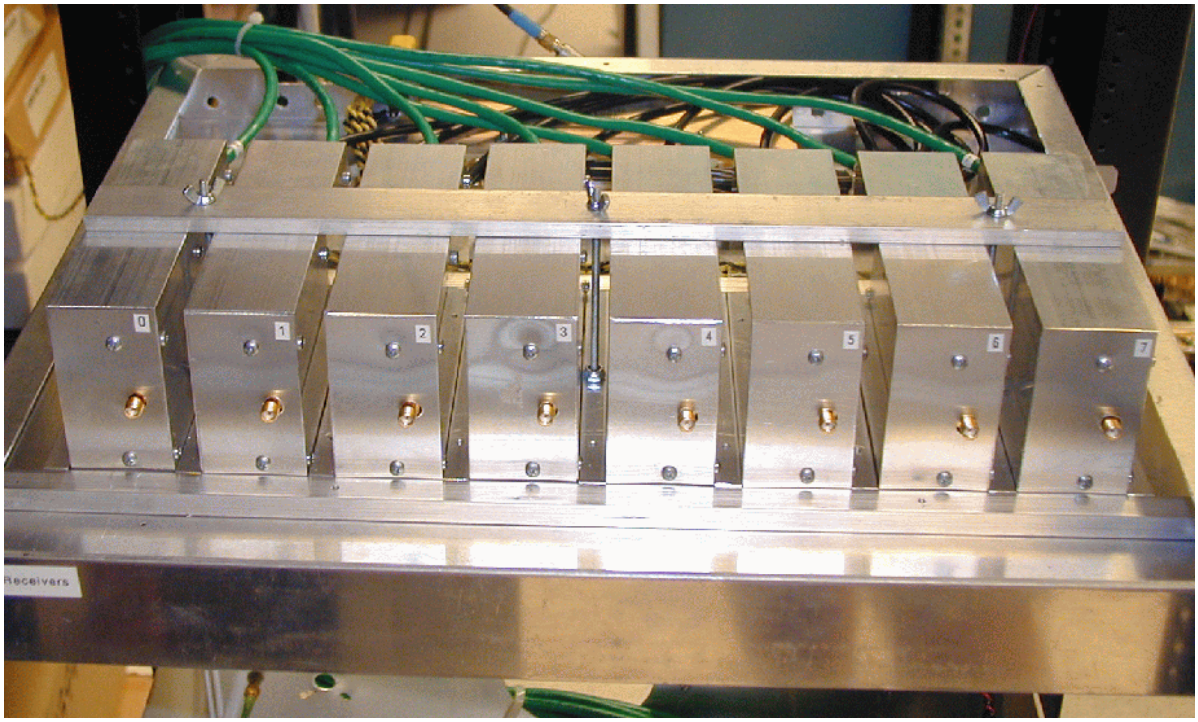
Figure 6 is a photograph of eight enclosed DCR modules. The amount of suppression provided by the enclosures is approximately 30dB. There is no noticeable conducted cross-talk between channels.



(a) PCB Top Side

(b) PCB Bottom Side

Figure 5: Direct Conversion Receiver printed circuit board layout images (actual size.) (a) The PCB top side contains mainly signal routing and a ground plane. The RF input is at the top, and the signal chain runs from top to bottom (Direct conversion receiver, ADC, FPGA and LVDS serializer.) (b) The PCB bottom side mainly contains power traces and also a ground plane.



(a) Front view of the 8 Direct Conversion Receivers

(b) Rear view of the 8 Direct Conversion Receivers

Figure 6: Photographs of eight enclosed direct conversion receivers mounted in a 19 inch tray. (a) The front view illustrates the 8 RF inputs. (b) The rear view illustrates the connections to the rear of the receivers. An LO input is split eight ways using a power divided and distributed through equal length RG58 cables. A CAT5 cable carries the ADC clock (40MHz) as well as the output LVDS data.

3 Digital Array Receiver/Processor

The DRP is the second stage of the new Argus Receiver chain. The DRP is designed to be a low cost digital tuner and decimation stage. A block diagram of the DRP is shown in Figure 7. There are four main parts to the system, the LVDS receive IC for the DCR, the controlling FPGA, the DDC and the LVDS data/control signals from the back plane.

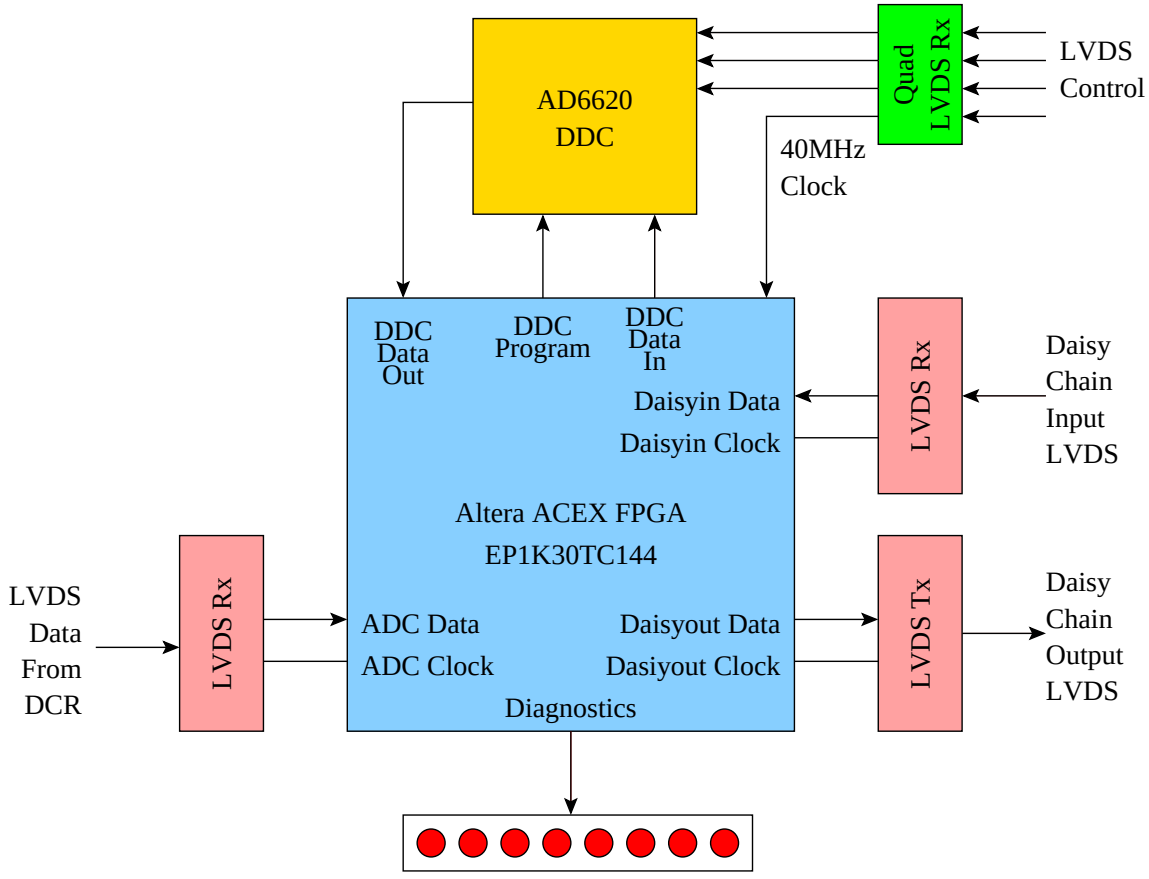


Figure 7: Block diagram of the Digital Receiver/Processor.

The LVDS receive IC [5] for the DCR is always operational and there is a constant stream of ADC samples. The data that is received from the ADC is in offset binary format. To convert this to `2's` complement form 128 is subtracted from the received data using an unsigned subtraction. This data is then output to the DDC input port. The control bits in the LVDS data which identify I and Q samples is used to toggle the A/B input of the DDC. The DDC in this case is the AD6620 [8].

The DDC is programmed via the FPGA. The FPGA is an ACEX series low cost FPGA from Altera [9]. This is a write only interface and there is no read back of the DDC internal registers. The control register data for the DDC originates from the LVDS daisy chain backplane. The FPGA contains code to extract the address and data for the DDC programming port. It takes approximately 100ms to fully program all the DDC registers. Changing the DDC tune frequency is of the order of milli-seconds.

The DDC is kept synchronous with all the other DDCs in the system via three control bits. These control signals originate from the backplane in the form of LVDS signals.

These signals are translated to CMOS levels using quad differential line receiver [10]. All DDCs are required to have identical programming, except for the possibly where they have different tune frequencies. The reference clock for the system also comes through this level translator. A reference clock is required for the LVDS receiver PLLs upon startup.

A majority of the FPGA functionality is taken by the state machines which process data to/from the LVDS daisy-chain backplane. The contents of this FPGA will be discussed in detail in a separate section.

The BOM for the DRP is listed in Table 2 for reference. A majority of the cost is dominated by the DDC and LVDS IC's. These prices are based on very low quantities and significant savings are possible with higher quantities. These prices are of May 2002, and since then they have dropped by at least 10%. The FPGA used is also over sized and lower speed (and size) devices are possible. The FPGA operates off at 2.5V power supply with 3.3V I/O. A photo of the DRP is shown in Figure 8(a), and the PCB layouts are shown in Figure 9 for reference.

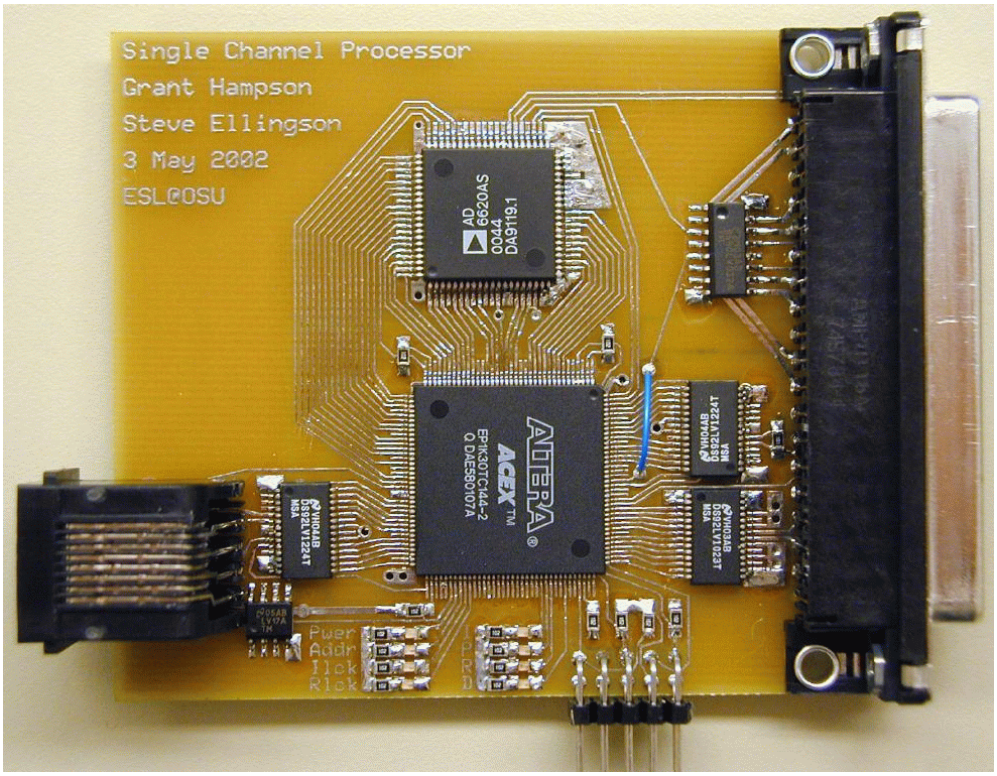
Table 2: Bill of Materials for the Digital Receiver/Processor

Component	Value	Quantity	Supplier	Unit Cost	Total Cost
Circuit board	serialproc.pcb (quantity 8)	1	Express PCB	\$17.62	\$17.62
Integrated Circuits	EP1K30TC144-2	1	Arrow	\$19.75	\$19.75
	AD6620AS	1	Future Active	\$30.77	\$30.77
	P524CT-ND (LED)	8	Digikey	\$0.30	\$2.40
	LM2937IMP-3.3CT-ND	2	Digikey	\$1.33	\$2.66
	LT1118CST-2.5-ND	1	Digikey	\$4.75	\$4.75
	DS90LV032ATM	1	Arrow	\$3.89	\$3.89
	DS92LV1224TMSA	2	Arrow	\$13.61	\$27.22
	DS92LV1023TMSA	1	Arrow	\$13.37	\$13.37
	DS90LV017ATM	1	Arrow	\$1.52	\$1.52
Connectors	A2103-ND (DB37)	1	Digikey	\$4.93	\$4.93
	A9044-ND (RJ8)	1	Digikey	\$0.64	\$0.64
Resistors	P1.0KACT-ND	15	Digikey	\$0.04	\$0.60
	P100ACT-ND	6	Digikey	\$0.04	\$0.24
Capacitors	PCC1812CT-ND(.1uF)	35	Digikey	\$0.07	\$2.45
	PCS6106CT-ND (10uF-35V)	5	Digikey	\$1.38	\$6.90
Total Part Count		82	Total BOM Cost		\$139.71

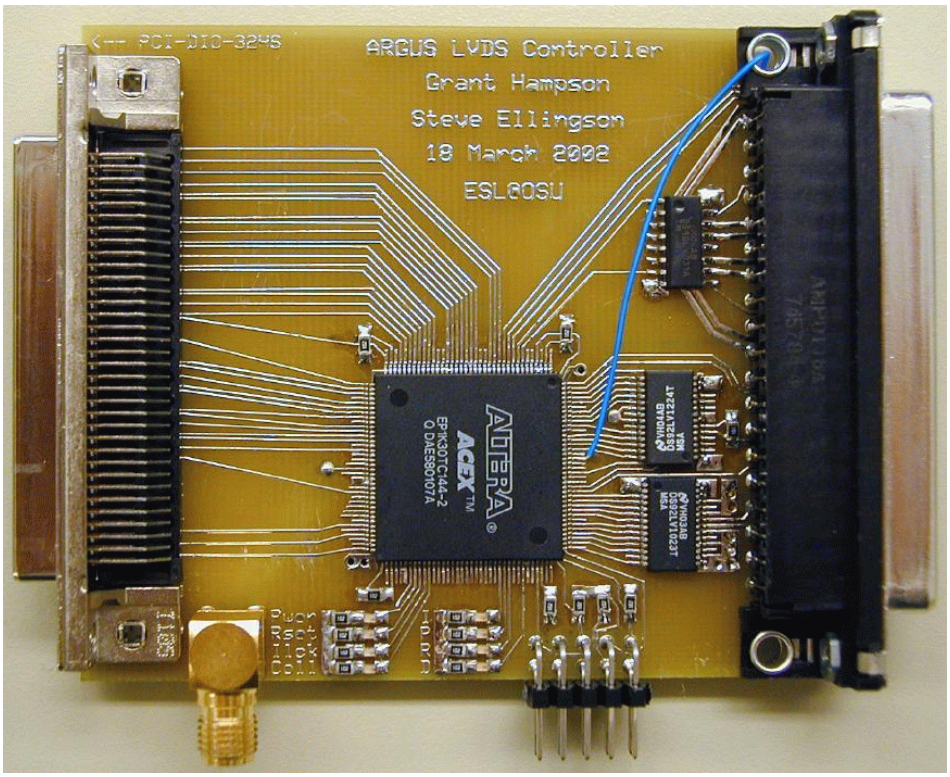
3.1 Array Controller

In additional to the DCR is the array controller. This board is similar to the DRP in that it connects to the LVDS daisy-chain backplane, however it also has a connection to the PC via a digital I/O card. A photo of the array controller is shown in Figure 8(b), and the PCB layout in Figure 10 for reference.

The array controller FPGA is identical to the DCR, but contains a different set of state machines. These state machines are controlled by the PC through the PCI-DIO-32HS digital I/O card [11]. The controller can address up to 64 separate DCR's. The master clock is also distributed from this point as well as the DDC synchronization signals. The state machines for the array controller will be discussed in a separate section.

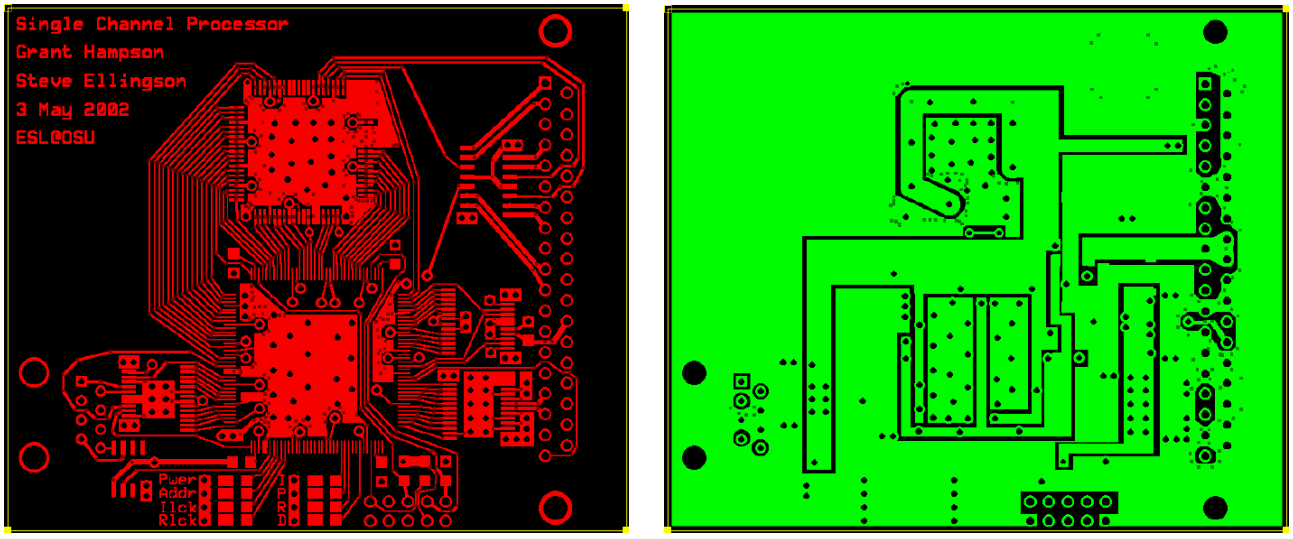


(a) Digital Receiver/Processor



(b) PCI-DIO-32HS Controller

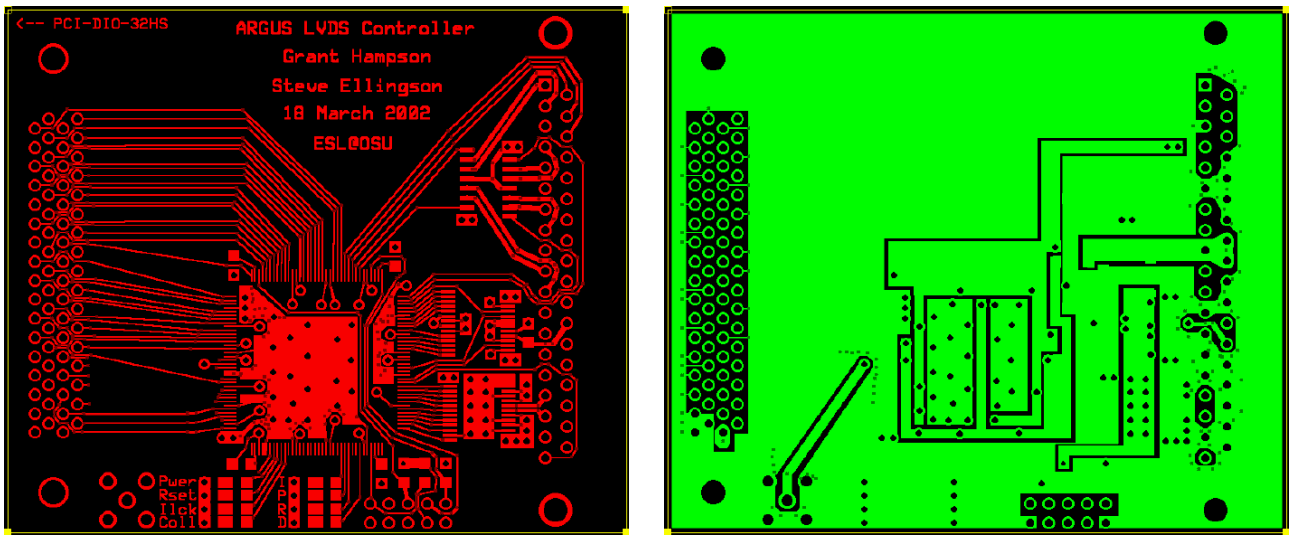
Figure 8: Photographs of the Digital Receiver/Processor and Controller assembled boards.



(a) PCB Top Side

(b) PCB Bottom Side

Figure 9: Digital Processor printed circuit board layout images. (a) The top side contains mainly signal traces. The backplane connector is on the right and the LVDS data from the direct conversion receiver on the left. The DDC is located above the FPGA. The top side contains all signal traces. (b) The bottom side is all power routing and a ground plane.



(a) PCB Top Side

(b) PCB Bottom Side

Figure 10: Digital Controller printed circuit board layout images. (a) The top side contains mainly signal traces. Data flow is from right to left (LVDS daisy chain to PCI-DIO-32HS connector.) (b) The bottom side is all power routing and a ground plane.

3.2 LVDS Backplane

The LVDS backplane is the third dimension in which the DCR's and array controller are connected. The completed backplane and daughter cards are shown in Figure 11. The backplane uses a DB37 connector to connect each of the DCRs. A DB37 connector is not a recommended LVDS connector, however the point to point distance (three inches) is very short. The ability to screw down the DCR's gives a rigid connection.

The PCB layout for the backplane is shown in Figure 12, where the daisy-chain traces can clearly be seen. All the LVDS traces are optimized for 100Ω differential impedance. Each of the DCR's is provided a regulated 5V power supply, and the total power consumption of eight DCRs is approximately 2.5A.

The first DCR connector has the ability to be a master DDC, and the remaining slots are DDC slaves. In this mode the master DDC control signals can be broadcast to the slave DDCs. Alternatively, all DDCs can be slaves and the array controller can generate the appropriate synchronization signals.

Each DRP is addressable in the array. The current address mode allows up to 64 DRP's to be addressed. The address of each DRP is located within the FPGA and consequently they all require a different program.

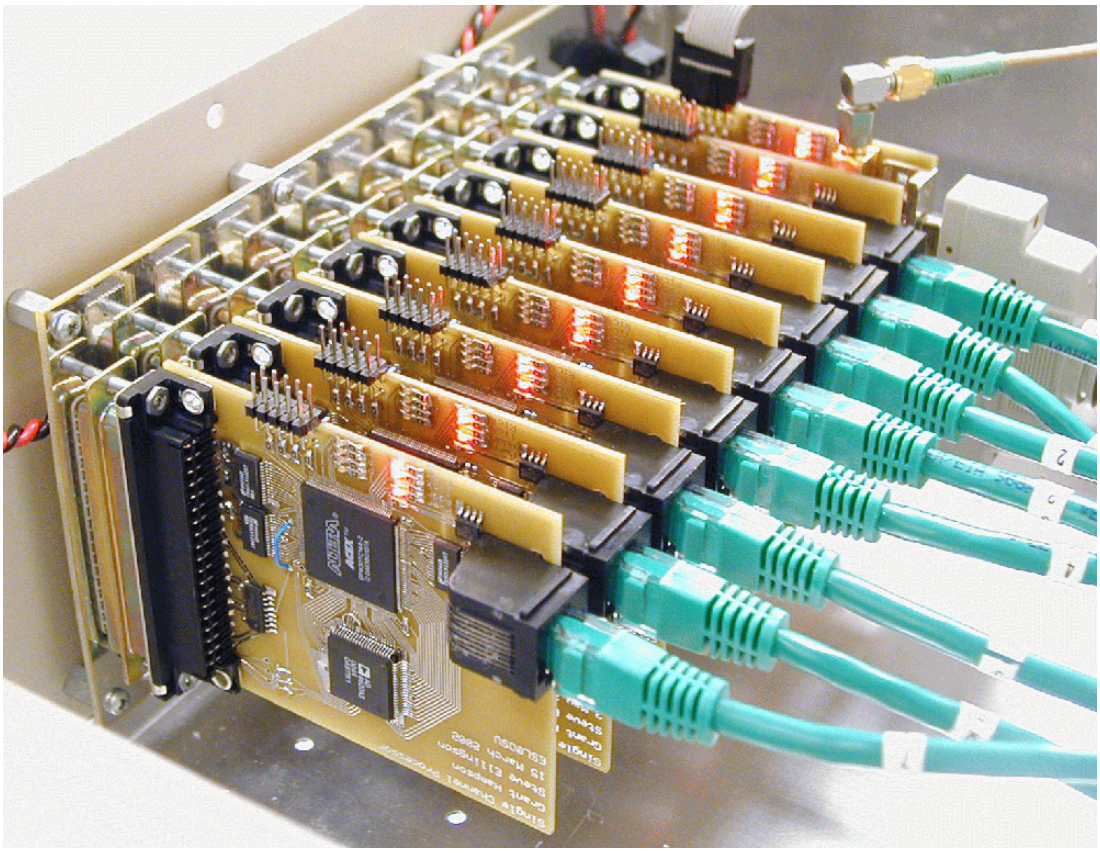
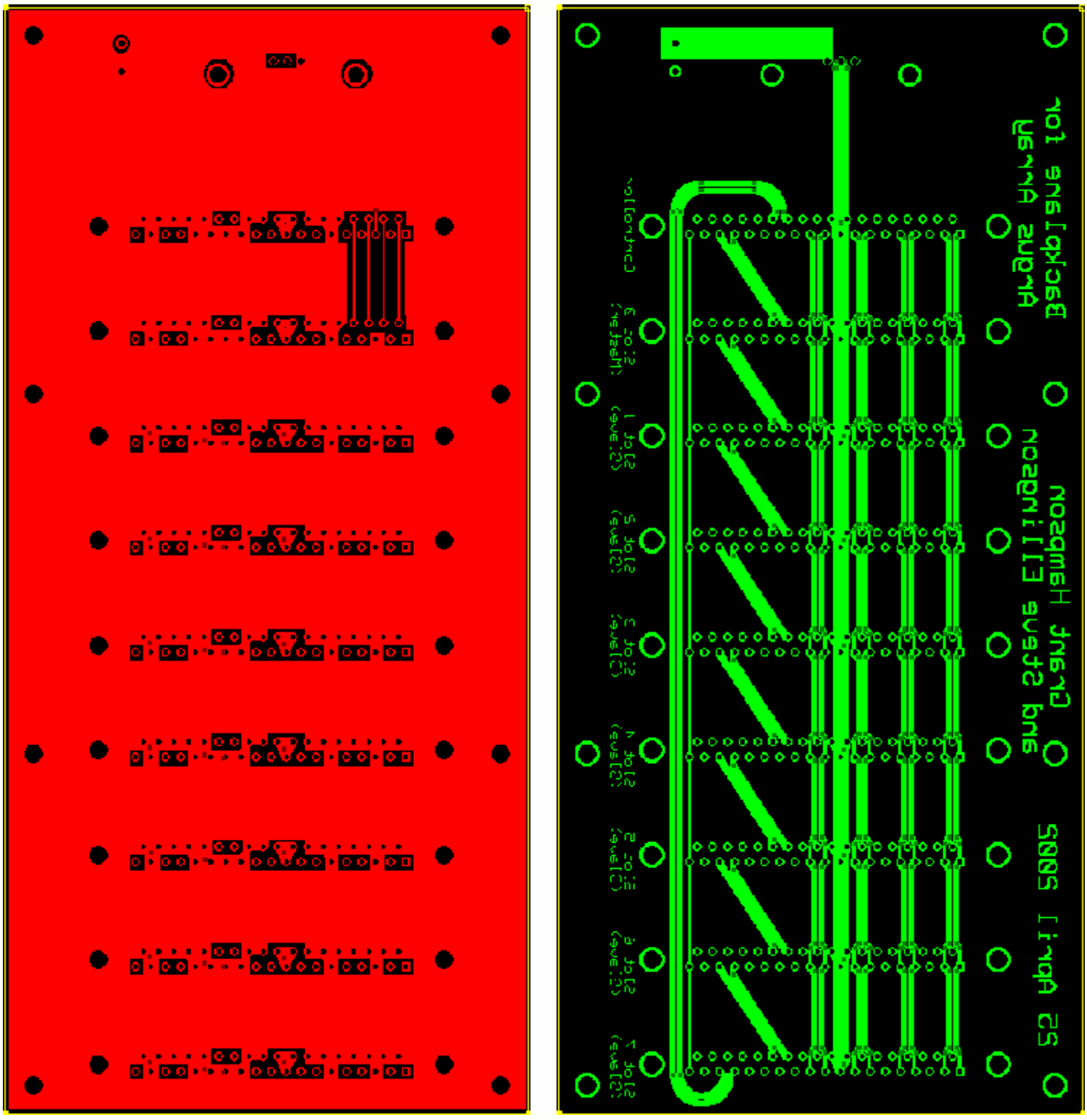


Figure 11: Photograph of the complete Argus Array Receiver. The eight LVDS CAT5 cables from the DCR's enter on the right. A backplane containing eight DRP's and one array controller is shown. The master 40MHz clock enters on the top right. The DCR's receive a clock via the CAT5 cable using LVDS transmit and receive IC's. Each of the FPGAs require a different FPGA program (lost upon power-down) via a JTAG cable.



(a) PCB Top Side

(b) PCB Bottom Side

Figure 12: LVDS backplane printed circuit board layout images. (a) The top side contains a large ground plane which shields the digital receiver/processors from the LVDS signals on the bottom of the backplane. (b) The bottom side contains all the LVDS traces as well as power (+5V). The LVDS daisy chain can be seen on the left side of the layout. The final LVDS output runs the length of the circuit board.

4 FPGA State Machines and Control Logic

This section aims to provide insight into the various modes of operation of the Argus Array Receiver. Firstly, there are two separate components; the DRP and the array controller. For each component there are two main state machines; the first decodes the input daisy chain data, and the second controls what is output.

Firstly, the array controller will be discussed such that the reader will have an introduction to the modes of operation of the array. The PCI-DIO-32HS control interface is defined in Table 3, where the four modes of operation are listed: IDLE, Program DDC, Send RAW Data and Send DDC Data. There are three registers internal to the array controller which set the Channel Address Register, and the other two are related to the DDC (Data and Address registers.)

Table 3: Port assignments for the PCI-DIO-32HS digital I/O card.

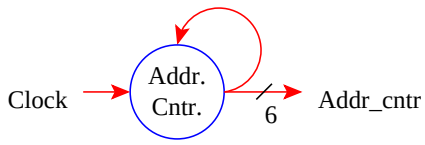
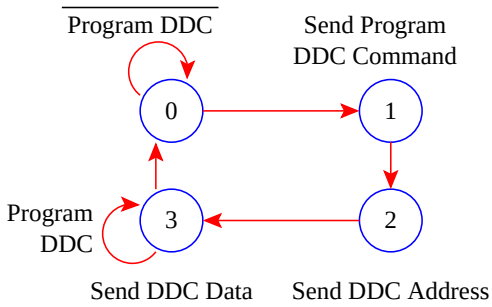
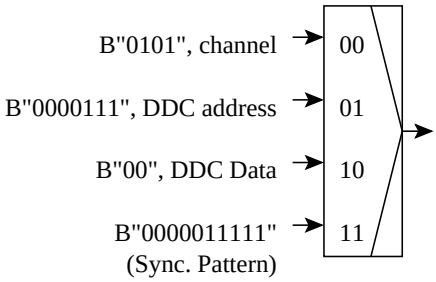
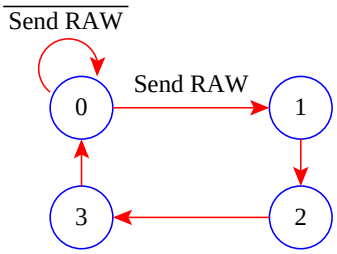
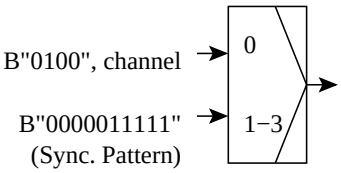
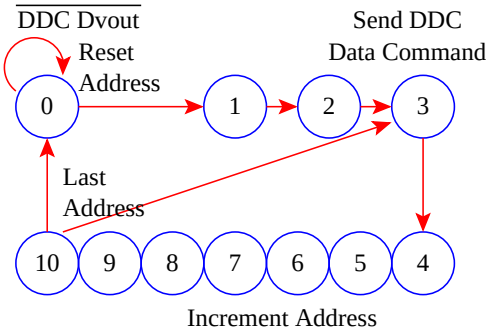
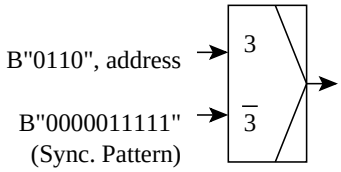
Port	Function
Port A: bit 0	IDLE Command
Port A: bit 1	Program DDC Command
Port A: bit 2	Send RAW Data Command
Port A: bit 3	Send DDC Data Command
Port A: bit 4	Reset
Port A: bits 5-7	Register Address 0-2
Port B: bits 0-7	Register Data Bus
Port C: bits 0-7	Lower Data Byte
Port D: bits 0-7	Upper Data Byte

When programming the DDC's the user must perform several steps:

1. Load the Channel Register with the address of the DRP to be programmed.
2. For each DDC control word:
   (a) Load the DDC Data and Address registers.
   (b) Toggle Port A:bit 1 high and low.

There are two types of data outputs from the system: RAW and DDC. Raw data refers to the raw ADC samples from a single channel. To access this data the user must write to the Channel register the address of desired DRP and assert Port A:bit2. To access DDC data from all DRP's the user simply asserts Port A:bit 3. When changing modes it is recommended that the user assert the IDLE command to clear any previous modes.

The logic and state machines which are located in the array controller are shown in Figure 13. The transmit controller writes data to the LVDS transmit port, and the receiver controller writes data to the PC. During periods where the LVDS Tx data has no meaning a sync pattern is sent to keep the LVDS daisy-chain operational (i.e., no

Mode	State Machine	LVDS Tx Data
IDLE		B"0111", addr_cntr[5..0]
Program DDC		
Send RAW		
Send DDC		

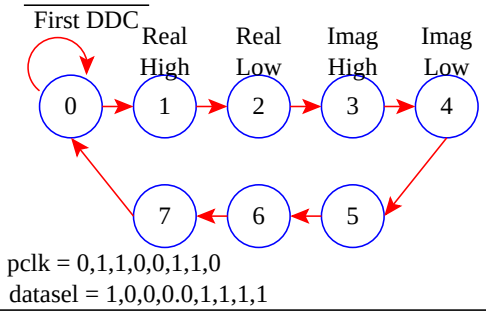
Mode	State Machine	Portc, Portd Data
Send RAW	No Sate Machine Required	portc=imagdata[], portd=realdata[]
Send DDC		portc= imaghigh or imaglow portc= realhigh or reallow

Figure 13: Simplified state machines for transmit and receive in the array controller.

RMT). The two modes which need some documentation are the Send RAW and Send DDC commands.

For the Send RAW command the transmit controller continuously sends the Send RAW command with the address of the channel. If this command is repeatedly sent then RMT occurs - consequently three sync patterns are sent to stop this. The received data is simply written to the PCI-DIO-32HS 16-bit data port.

For the Send DDC command the state machine is more sophisticated. Firstly the state machine waits for the DDC to output a sample, indicated by Dvout. It takes two clock cycles to write this data to a register in the DRP. The Send DDC Data command is then sent with an address and then the controller waits 7 clock cycles before requesting the next address. Only four clock cycles are required to transmit 32-bits of DDC data, so there is a possibility to shorten this statemachine.

The LVDS receive data could be directly broadcast to any post-processing devices (such as a beamformer) since all information is encoded in this stream (DDC address and data.) For this implementation though, the data is decoded and sent to the PCI-DIO-32HS I/O card. The receive state machine looks for the first DDC address (zero in this case) and then writes out the real and imaginary samples. No addresses are written to the PC since a special control mechanism is implemented. The PC can reset the output data state machine and start from address zero.

4.1 DRP Logic and State Machines

The DRP contains more logic than the array controller to decode the incoming data stream. A summary of the logic contained within the DRP is illustrated in Figure 14.

The raw LVDS input is always present and is output to the DDC and a synchronizing FIFO (two clock domains - raw clock and daisy in clock.) The data which is output to the daisy out LVDS port is decided by a master state machine.

The LVDS daisy in data is decoded to determine if a command is address to this module and triggers the master state machine. The master state machine activates smaller state machines depending on the command. For the most part though a majority of the data simply passes from input to output. The LVDS data in contains 10-bits, the first two identifying it as raw data or a command.

The FPGA resources consumed by DRP FPGA is approximately 400LE (when the FIFO is implemented in Logic Elements (LE)), which is relatively small. If IQ correction was to be implemented then significantly more LE would be required (approximately 500 since two 8×8-bit multipliers are required.) Preliminary tests indicate that the the image of a sinusoid is approximately 30dB down. I-Q correction has not been implemented yet but will be included in a future version. The I-Q correction factors will be programmable through the LVDS back plane.

There are some modifications of the DRP logic that could make it slightly smaller and more efficient. This will not be discussed here though.

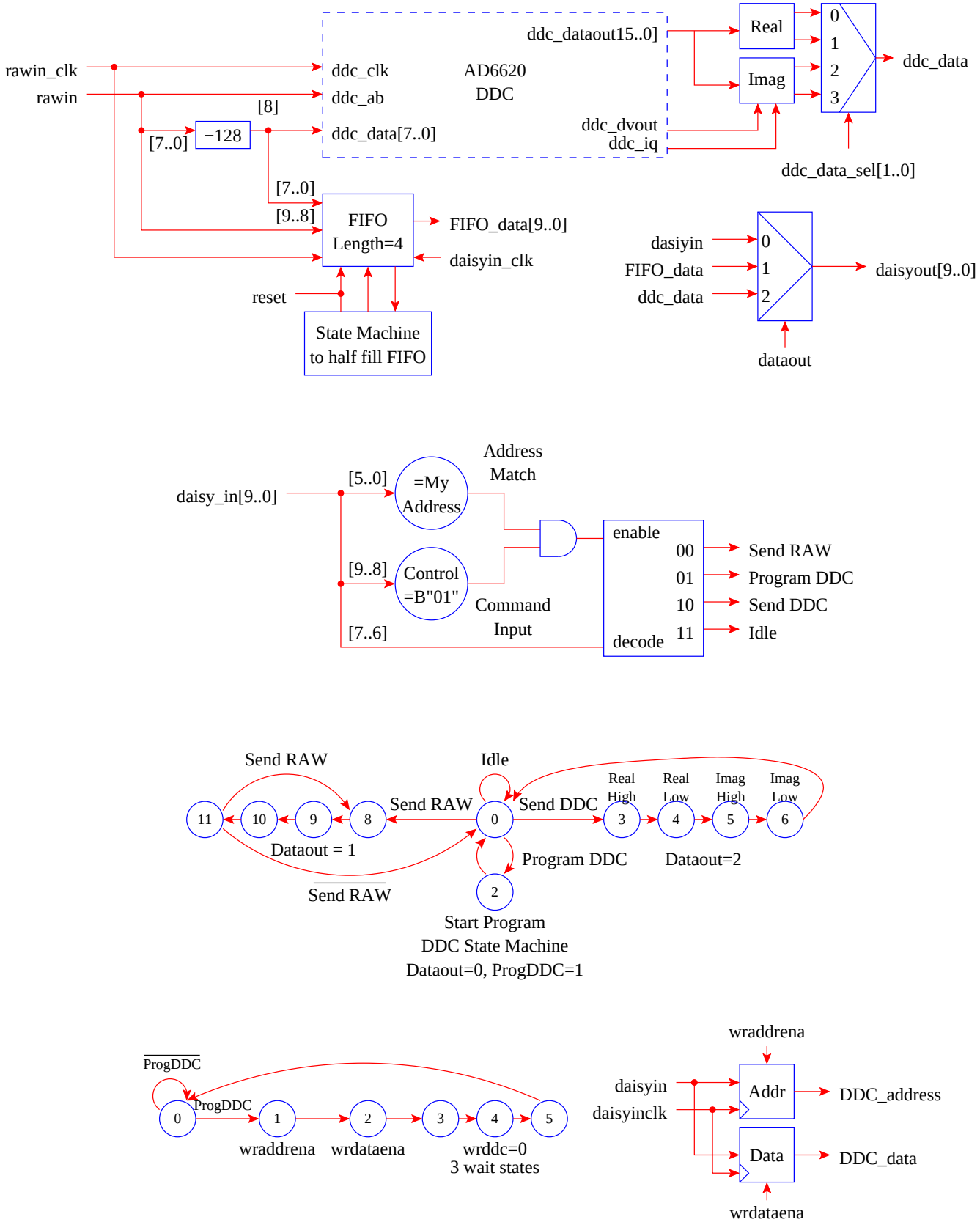


Figure 14: Simplified logic and state machines for the DRP.

Summary and Conclusions

This report has discussed the implementation details of a new Argus Direct Conversion Receiver and Digital Array Receiver/Processor. There are three key developments which make this a cost effective implementation: a Direct Conversion Tuner, LVDS data transport, and a Digital Down Converter. These key technologies have been discussed in detail. The resulting system could be the basis of a larger array receiver.

The authors would like to thank the SETI Institute and OSU for funding part of this research. Additionally, the authors would also like to thank Keith Hampson for his time and effort in building the Direct Conversion Receiver enclosures.

There are several improvements which may improve the performance and lower the cost of the system. These are listed in the following section.

Possible Improvements

List of possible upgrades:

1. Upgrade the DRP to a 4 layer PCB with similar costs ($15 per PCB for quantity 16 (from `pcbexpress.com`). Could have signal layer, 3.3V, 2.5V and a ground layer. Improved performance due to power planes and reduce number of voltage regulators.

2. There are significantly more LVDS devices on the market now and it could be possible to upgrade. For example there is a new 16-bit LVDS Tx/Rx chip specifically designed for back plane use (DS92LV16). It could be possible to upgrade the backplane and increase its bandwidth by a factor two (with appropriate modifications to the state machines.) The DCR bandwidth could also be increased too. There is no significant increase in cost.

3. In the current addressing scheme each FPGA contained a different piece of firmware which contained a different address. It would be beneficial to change this so the address is externally set (using a dip switch) and have all firmware be identical.

4. There is a significant number of voltage regulators in the system which increase costs. It could be possible to place larger power regulators (3.3 and 2.5 volts) on backplane, instead of local regulation.

5. The cost of the DCR enclosures could be reduced from $17 to less than $1 by using a different type of enclosure. The enclosure would be a slice of 5 by 2 inch aluminum tube that is a quarter inch deep. All Connectors will be placed on back side of board and a aluminum sheet cover placed on the other side of the box with six 4-40 screws holding the sandwich together.

6. It might also be able to make DCR PCB a three layer board which would provide a better power and ground plane. The number of voltage regulators could also be reduced on this board.

7. Two additional OPAMPs between the anti-aliasing filters and the ADC inputs to improve matching.

8. Implementing I-Q imbalance correction.

References

[1] *Monolithic Microwave Amplifiers: ERA-5SM*, Mini-Circuits, 2002. http://www.mini-circuits.com/dg03-168.pdf.

[2] *MAX2102/MAX2105 Direct-Conversion Tuner ICs for Digital DBS Applications* , Maxim Semiconductor, October 1998. http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1758.

[3] *Dual Channel 8-bit Resolution CMOS ADC*, Analog Devices, 1999. http://www.analog.com/productSelection/pdf/AD9281_e.pdf.

[4] *3V LVDS Single CMOS Differential Line Receiver*, National Semiconductor, June 1998. http://www.national.com/ds/DS/DS90LV018A.pdf.

[5] *DS92LV1023 and DS92LV1224 40-66 MHz 10 Bit Bus LVDS Serializer and Deserializer*, National Semiconductor, June 2002. http://www.national.com/ds/DS/DS92LV1023.pdf.

[6] *MAX 7000A Programmable Logic Device Data Sheet*, Altera, October 2001. http://www.altera.com/literature/ds/m7000a.pdf.

[7] *LVDS Owner's Manual*, National Semiconductor, March 2000. http://www.national.com/appinfo/lvds/files/LOM_2.pdf.

[8] *65 MSPS Digital Receive Signal Processor Data Sheet*, Analog Devices, 2001. http://www.analog.com/productSelection/pdf/AD6620_a.pdf.

[9] *ACEX1K Programmable Logic Device Family Data Sheet*, Altera, September 2001. http://www.altera.com/literature/ds/acex.pdf.

[10] *DS90LV032A 3V LVDS Quad CMOS Differential Line Receiver*, National Semiconductor, July 1999. http://www.national.com/ds/DS/DS90LV032A.pdf.

[11] *High-Speed 32-bit Digital Pattern I/O and Handshaking*, National Instruments, 2002. http://www.ni.com/pdf/products/us/2mhw332-333e.pdf.

Appendix A: Direct Conversion FPGA Source Code

```
--
-- AD9281 ADC and DS92LV1023 LVDS Serialiser
-- G.Hampson 12 April 2002
--

SUBDESIGN ad9281
(
   clk40,                  -- Clock from LVDS or External clock
   adc_data[7..0]          -- 8-bit output of ADC on rising edge of adc_clk

      :INPUT;

   adc_clk,                -- ADC Clock (new sample on rising edge)
   adc_select,             -- Selects ADC I/Q output (sel=0=Q=imaginary, sel=1=I=real)
   lvds_tclk,              -- LVDS data clock
   lvds_data[9..0]         -- LVDS data bus

      :OUTPUT;
)

VARIABLE
   clk20: DFF;             -- 20MHz clock
   data_latch[9..0]: DFF;  -- 10-bit register

BEGIN
   clk20.clk = clk40;      -- ADC clock is half the LVDS clock
   clk20.d = !clk20.q;

   adc_clk = clk20.q;      -- common Clock and Select lines
   adc_select = clk20.q;

   data_latch[7..0].d = adc_data[7..0]; -- offset binary data
   data_latch[9..8].d = adc_select;     -- encode the IQ data
   data_latch[].clk = clk40;            -- latch data at LVDS rate

   lvds_tclk = clk40;
   lvds_data[9..0] = data_latch[9..0];  -- transmit the LVDS data
END;
```

Appendix B: Digital Receiver/Processor Source Code

```
-- Serial Processor Board
-- Grant Hampson 28 Feb 2002

CONSTANT MYADDRESS = B"000110"; -- 6 bit binary address
INCLUDE "lpm_fifo_dc.inc";      -- include file for FIFO

SUBDESIGN serialproc
(
   clock,                     -- system clock
   reset,                     -- system wide reset

   rawin_data[9..0],          -- LVDS raw data bus
   rawin_clk,                 -- LVDS raw clock
   rawin_lock,                -- LVDS raw lock signal

   daisyin_data[9..0],        -- LVDS daisy chain input data bus
   daisyin_clk,               -- LVDS daisy chain input clock
   daisyin_lock,              -- LVDS daisy chain input lock signal

   ddc_dataout[15..0],        -- DDC output bus
   ddc_dvout,                 -- DDC data valid output
   ddc_iq                     -- DDC data is I/Q

      :INPUT;

   ad_clk,                    -- clock for AD converter (converted to LVDS)

   raw_refclk,                -- LVDS daisy chain input reference clock

   daisyin_refclk,            -- LVDS daisy chain input reference clock

   daisyout_data[9..0],       -- LVDS daisy chain output data bus
   daisyout_clk,              -- LVDS daisy chain output clock

   ddc_datain[7..0],          -- DDC data input
   ddc_ab,                    -- DDC I/Q selection input
   ddc_clk,                   -- DDC clock

   ddc_data[7..0],            -- DDC program data
   ddc_address[2..0],         -- DDC address
   ddc_rw,                    -- DDC write line

   led[7..0]                  -- Status LEDs

      :OUTPUT;
)

VARIABLE
   real_latch[15..0],
   imag_latch[15..0],         -- Registers for DDC output data
   ddc_data_reg[7..0],
   ddc_addr_reg[2..0]: DFFE;  -- Registers to hold the data and address while writing

   control_sm[3..0],          -- Statemachine which is the main control mechanism
```

```
    ddcpr_sm[2..0],           -- State machine for programming DDC
    ddcdata_sm[2..0],         -- State machine for outputting DDC data onto bus
    ddc_out_sel[1..0],        -- For transmission of DDC data
    rawdata_sm: DFF;          -- State machine for capturing data into FIFO - change of clocks

    address_match,            -- Adddress in packet matches mine
    command,                  -- Packet is a command packet
    control[3..0],            -- decoded command packet
    data_out[1..0],           -- multiplexor selection lines
    prog_ddc, ddcdata,        -- Signals which start their state machines
    wraddr, wrdata,           -- Signals from statemachine to write address and data
    ddc_out_data[9..0]: NODE; -- Datapath for Real low/high, Imag high/low

    raw_fifo : lpm_fifo_dc WITH (LPM_WIDTH = 10,
                                 LPM_NUMWORDS = 16,
                                 LPM_WIDTHU = 4,
                                 LPM_SHOWAHEAD = "OFF",
                                 LPM_HINT = "USE_EAB=OFF"); -- in LE

BEGIN
    --
    -- Connection of status LEDs
    --
    led[0] = VCC;             -- power indicator
    led[1] = command;         -- indicates address matches
    led[2] = !daisyin_lock;   -- LVDS daisy chain lock signal
    led[3] = !rawin_lock;     -- LVDS raw lock signal
    led[4] = control[3];      -- idle indicator
    led[5] = control[1];      -- program DDC indicator
    led[6] = control[0];      -- Raw data indicator
    led[7] = control[2];      -- send DDC data indicator

    --
    -- Give the two receive LVDS links a reference clock
    --
    ad_clk = clock;           -- Analogue-Digital converter clock
    raw_refclk = clock;       -- Raw data LVDS reference clock
    daisyin_refclk = clock;   -- Daisy chain in LVDS reference clock
    daisyout_clk = daisyin_clk;-- Daisy chain output clock = input clock

    --
    -- DDC is always receiving/generating data
    --
    ddc_datain[7..0] = rawin_data[7..0] - B"10000000"; -- convert to 2's complement
    ddc_ab = rawin_data[8];
    ddc_clk = rawin_clk;

    real_latch[15..0].d = ddc_dataout[15..0]; -- latch DDC outputs
    imag_latch[15..0].d = ddc_dataout[15..0];
    real_latch[15..0].clk = rawin_clk;        -- DDC clock
    imag_latch[15..0].clk = rawin_clk;
    real_latch[15..0].ena = ddc_iq AND ddc_dvout;
    imag_latch[15..0].ena = (!ddc_iq) AND ddc_dvout;

    --
    -- A FIFO to match Raw and Daisyin data streams on the same clock
```

```
--
raw_fifo.aclr = !reset;
raw_fifo.wrclock = !rawin_clk;
raw_fifo.wrreq = VCC;
raw_fifo.data[7..0] = rawin_data[7..0] - B"10000000"; -- convert to 2's complement
raw_fifo.data[9..8] = rawin_data[9..8];
raw_fifo.rdclock = !daisyin_clk;   -- read data using daisy chain clock

TABLE  -- State Machine for loading FIFO
   rawdata_sm.q, reset, raw_fifo.wrusedw[2] => rawdata_sm.d, raw_fifo.rdreq;
        B"0",     B"0",        B"X"         => B"0",         B"0"; -- reset condition
        B"0",     B"1",        B"0"         => B"0",         B"0"; -- half fill FIFO
        B"0",     B"1",        B"1"         => B"1",         B"1"; -- FIFO is now half FULL
        B"1",     B"1",        B"X"         => B"1",         B"1"; -- read and write data
END TABLE;
rawdata_sm.clk = rawin_clk;
rawdata_sm.clrn = reset;

--
-- Determine the purpose of the incoming data packet on daisy chain input
--
address_match = (daisyin_data[5..0] == MYADDRESS);  -- does it have my address on it?
command = address_match AND (daisyin_data[9..8] == B"01"); -- is it a control packet

control[0] = command AND (daisyin_data[7..6] == B"00"); -- send raw data
control[1] = command AND (daisyin_data[7..6] == B"01"); -- program DDC
control[2] = command AND (daisyin_data[7..6] == B"10"); -- send DDC data
control[3] = command AND (daisyin_data[7..6] == B"11"); -- go to idle state

case data_out[1..0] is   -- determine output data source
   when B"00" => daisyout_data[9..0] = daisyin_data[9..0]; -- input data
   when B"01" => daisyout_data[9..0] = raw_fifo.q[9..0];   -- raw data from FIFO
   when B"10" => daisyout_data[9..0] = ddc_out_data[9..0]; -- ddc data
end case;

--
-- State Machine for incoming commands
--
TABLE
control_sm[].q, control[] => control_sm[].d, data_out[], prog_ddc, ddcdata;
       B"0000",   B"0000" =>        B"0000",      B"00",     B"0",    B"0"; -- idle condition
       B"0000",   B"1000" =>        B"0000",      B"00",     B"0",    B"0"; -- idle mode (NOP)

       B"0000",   B"0001" =>        B"1000",      B"01",     B"0",    B"0"; -- sel raw data
       B"1000",   B"XXXX" =>        B"1001",      B"01",     B"0",    B"0"; -- sel raw data
       B"1001",   B"XXXX" =>        B"1010",      B"01",     B"0",    B"0"; -- sel raw data
       B"1010",   B"XXXX" =>        B"1011",      B"01",     B"0",    B"0"; -- sel raw data
       B"1011",   B"XXX1" =>        B"1000",      B"01",     B"0",    B"0"; -- new raw command
       B"1011",   B"XXX0" =>        B"0000",      B"01",     B"0",    B"0"; -- stop raw command

       B"0000",   B"0010" =>        B"0010",      B"00",     B"1",    B"0"; -- program DDC
       B"0010",   B"XXXX" =>        B"0000",      B"00",     B"1",    B"0"; -- return to idle

       B"0000",   B"0100" =>        B"0011",      B"00",     B"0",    B"1"; -- Start DDC sm
       B"0011",   B"XXXX" =>        B"0100",      B"10",     B"0",    B"1"; -- first byte
       B"0100",   B"XXXX" =>        B"0101",      B"10",     B"0",    B"1"; -- second byte
```

```
        B"0101",   B"XXXX" =>        B"0110",      B"10",     B"0",    B"0"; -- third byte
        B"0110",   B"XXXX" =>        B"0000",      B"10",     B"0",    B"0"; -- fourth byte
    END TABLE;
    control_sm[].clk = daisyin_clk; -- Runs on the daisyin LVDS clock

    --
    -- State Machine for Programming DDC
    --
    TABLE
    ddcpr_sm[].q, prog_ddc => ddcpr_sm[].d, wraddr, wrdata, ddc_rw;
          B"000",     B"0" =>       B"000",  B"0",    B"0",   B"1"; -- reset condition
          B"000",     B"1" =>       B"001",  B"0",    B"0",   B"1"; -- SM activated
          B"001",     B"X" =>       B"010",  B"1",    B"0",   B"1"; -- write address
          B"010",     B"X" =>       B"011",  B"0",    B"1",   B"1"; -- write data
          B"011",     B"X" =>       B"100",  B"0",    B"0",   B"0"; -- write to DDC
          B"100",     B"X" =>       B"101",  B"0",    B"0",   B"0"; -- 2nd clock cycle
          B"101",     B"X" =>       B"000",  B"0",    B"0",   B"0"; -- 3rd clock cycle
    END TABLE;
    ddcpr_sm[].clk = daisyin_clk;  -- Runs on the lvds receive clock

    ddc_addr_reg[2..0].clk = daisyin_clk;
    ddc_addr_reg[2..0].d = daisyin_data[2..0];
    ddc_addr_reg[2..0].ena = wraddr;  -- only written to on 2nd clock cycle
    ddc_address[2..0] = ddc_addr_reg[2..0];

    ddc_data_reg[7..0].clk = daisyin_clk;
    ddc_data_reg[7..0].d = daisyin_data[7..0];
    ddc_data_reg[7..0].ena = wrdata;  -- only written to on 3rd clock cycle
    ddc_data[7..0] = ddc_data_reg[7..0];

    --
    -- State Machine for Writing DDC Data out
    --
    TABLE
    ddcdata_sm[].q, ddcdata => ddcdata_sm[].d, ddc_out_sel[1..0].d;
            B"000",    B"0" =>         B"000",               B"00"; -- reset condition
            B"000",    B"1" =>         B"001",               B"00"; -- SM activated
            B"001",    B"X" =>         B"010",               B"01"; -- write real high
            B"010",    B"X" =>         B"011",               B"10"; -- write real low
            B"011",    B"X" =>         B"100",               B"11"; -- write imag high
            B"100",    B"X" =>         B"000",               B"00"; -- write imag low
    END TABLE;
    ddcdata_sm[].clk = !daisyin_clk; -- needs negative edge for transmit
    ddc_out_sel[].clk = !daisyin_clk;

    case ddc_out_sel[1..0] is
       when B"00" => ddc_out_data[9..0] = (B"10", real_latch[15..8]);
       when B"01" => ddc_out_data[9..0] = (B"10", real_latch[7..0]);
       when B"10" => ddc_out_data[9..0] = (B"10", imag_latch[15..8]);
       when B"11" => ddc_out_data[9..0] = (B"10", imag_latch[7..0]);
    end case;
END;
```

Appendix C: PCI-DIO-32HS Controller Source Code

```
-- Serial Controller Board
-- Grant Hampson 9 April 2002

CONSTANT MAXADDRESS = B"000111";  -- 8 channel system

SUBDESIGN serialcont
(
   clock,                -- system wide clock

   daisyin_data[9..0],   -- LVDS daisy chain input data bus
   daisyin_clk,          -- LVDS daisy chain input clock
   daisyin_lock,         -- LVDS daisy chain input lock signal

   ack,                  -- PCI-DIO-32HS interface
   porta[7..0],          -- control signals
   portb[7..0],          -- data bus

   sync_nco_in,          -- Sync signals from Master DDC
   sync_cic_in,
   sync_rcf_in,
   dv_out_in             -- Signals DDC data is valid

      : INPUT;

   reset,                -- system wide reset
   sys_clock,            -- system wide clock

   daisyin_refclk,       -- LVDS daisy chain input reference clock

   daisyout_data[9..0],  -- LVDS daisy chain output data bus
   daisyout_clk,         -- LVDS daisy chain output clock

   led[7..0],            -- Status LEDs

   req,                  -- interface to PCI-DIO-32HS card
   pclk,
   portc[7..0],
   portd[7..0],

   sync_nco_out,         -- Sync signals to all slave DDCs
   sync_cic_out,
   sync_rcf_out

      : OUTPUT;
)

VARIABLE
porta_reg[7..0], portb_reg[7..0],          -- Register asynchronous control words
ack_reg,                                   -- synchronise ACK input from PC
rawdata_sm[1..0],                          -- state machine for raw data transfer
ddcprog_sm[1..0],                          -- state machine for programming DDC
ddcread_sm[3..0],                          -- state machine for DDC data transfer
data_sel, progddc_sel[1..0],               -- control bits for selecting DDC prog data
ddc_sel,                                   -- selecting DDC command transmit data
```

```
dv_out,                                     -- synchronising register
cnt_rcf[8..0],cnt_cic[7..0],cnt_nco[15..0],  -- counter registers for synchronisation
ddc_reg_en[3..0],                           -- enables for real/imag DDC data registers
ddc_clk_out,                                -- DDC clock which goes to pclk
ddcout_sm[3..0],                            -- state machine for DDC data to PC
ddc_outdata,                                -- selects DDC I/Q output data to PC
reset_addr, inc_addr,                       -- DDC channel counter controls
ddc_cnt_ena, ddc_cnt_reset : DFF;

addr_cnt[5..0], read_cnt[5..0],             -- Channel address counter
address_reg[2..0],                          -- Register for DDC address
channel_reg[5..0],                          -- Channel Address for RAW/Program DDC
data_reg[7..0],                             -- Data written to DDC data port
imag_data_reg[7..0],                        -- registers for buffering raw data
real_data_reg[7..0],
real_low[7..0], real_high[7..0],            -- DDC real output data register
imag_low[7..0], imag_high[7..0] : DFFE;     -- DDC imag output data register

ddc_start,                                  -- detects start of DDC data packet
ddc_done, last_ddc,                         -- signals last DDC channel
ddcout_data[7..0] : NODE;                   -- outputs of DDC write state machine

BEGIN
   --
   -- Connection of status LEDs, etc.
   --
   led[0] = VCC;
   led[1] = reset;
   led[2] = !daisyin_lock;
   led[3] = pclk;
   led[4] = porta_reg[0];-- Idle
   led[5] = porta_reg[1];-- Program
   led[6] = porta_reg[2];-- Raw
   led[7] = porta_reg[3];-- DDC

   reset = porta_reg[4];
   sys_clock = clock;-- this gets broadcast to all processors

   --
   -- Generate synchronisation pulses for the DDCs
   --
   cnt_rcf[].clk = clock;
   cnt_cic[].clk = clock;
   cnt_nco[].clk = clock;
   if cnt_rcf[].q == B"110001111" then
      cnt_rcf[].d = B"000000000";
   else
      cnt_rcf[].d = cnt_rcf[].q + 1;
   end if;
   sync_rcf_out = (cnt_rcf[] == B"000000000");

   if cnt_cic[].q == B"11000111" then
      cnt_cic[].d = B"00000000";
   else
      cnt_cic[].d = cnt_cic[].q + 1;
   end if;
```

```
sync_cic_out = (cnt_cic[] == B"00000000");

if cnt_nco[].q == B"111111111111111" then
   cnt_nco[].d = B"0000000000000000";
else
   cnt_nco[].d = cnt_nco[].q + 1;
end if;
sync_nco_out = (cnt_nco[15..1] == B"000000000000000"); -- Must be 2 clock cycles

-- sync_nco_out = sync_nco_in; -- when channel 0 is a master
-- sync_rcf_out = sync_rcf_in;
-- sync_cic_out = sync_cic_in;

--
-- Three registers hold the relevant information
--
porta_reg[].clk = clock;
porta_reg[].d = porta[];  -- Align clock edges on PC and controller
portb_reg[].clk = clock;
portb_reg[].d = portb[];

channel_reg[].clk = clock;
channel_reg[].ena = porta_reg[7];
channel_reg[].d = portb_reg[5..0];

data_reg[].clk = clock;
data_reg[].ena = porta_reg[6];
data_reg[].d = portb_reg[7..0];

address_reg[].clk = clock;
address_reg[].ena = porta_reg[5];
address_reg[2..0].d = portb_reg[2..0];

--
-- Address Counter (6-bit = up to 64 channels
--
addr_cnt[].clk = !clock;           -- all transmit LVDS works on falling edge
if addr_cnt[].q == MAXADDRESS then -- N-channels
   addr_cnt[].d = B"000000";       -- reset address
else
   addr_cnt[].d = addr_cnt[].q + 1;
end if;
last_ddc = (addr_cnt[].q == B"000000");

--
-- State machine for getting raw data from one channel
--
TABLE
rawdata_sm[].q => rawdata_sm[].d, data_sel.d;
         B"00" =>          B"01",     B"0"; -- send raw data
         B"01" =>          B"10",     B"1"; -- sync pattern
         B"10" =>          B"11",     B"1"; -- sync pattern
         B"11" =>          B"00",     B"1"; -- sync pattern
END TABLE;
rawdata_sm[].clk = !clock; -- Invert clock to allow for LVDS setup times
data_sel.clk = !clock;
```

```
--
-- State machine for programming DDCs
--
TABLE
ddcprog_sm[].q, porta_reg[1] => ddcprog_sm[].d, progddc_sel[1..0].d;
         B"00",         B"0" => B"00",          B"11"; -- idle (send sync)
         B"00",         B"1" => B"01",          B"00"; -- send command
         B"01",         B"X" => B"10",          B"01"; -- send address
         B"10",         B"X" => B"11",          B"10"; -- send data
         B"11",         B"1" => B"11",          B"11"; -- only program DDC once
         B"11",         B"0" => B"00",          B"11"; -- finnished
END TABLE;
ddcprog_sm[].clk = !clock; -- Invert clock to allow for LVDS setup times
progddc_sel[].clk = !clock;

--
-- State machine for getting DDC data
-- Sync NCO is replaced by DVout signal from DDC
--
TABLE
ddcread_sm[].q, dv_out_in, last_ddc => ddcread_sm[].d, (ddc_sel.d,inc_addr.d,reset_addr.d);
     B"0000",        B"0",     B"X" => B"0000",        B"000"; -- wait for DDC DVout, reset
     B"0000",        B"1",     B"X" => B"0001",        B"001"; -- DDC data is ready
     B"0001",        B"X",     B"X" => B"0010",        B"001"; -- real written
     B"0010",        B"X",     B"X" => B"0011",        B"001"; -- imag written
     B"0011",        B"X",     B"X" => B"0100",        B"101"; -- send DDC data command
     B"0100",        B"X",     B"X" => B"0101",        B"001"; -- 2nd clock cycle
     B"0101",        B"X",     B"X" => B"0110",        B"001"; -- 3rd clock cycle
     B"0110",        B"X",     B"X" => B"0111",        B"011"; -- 4th clock cycle
     B"0111",        B"X",     B"X" => B"1000",        B"001"; -- 5th clock cycle
     B"1000",        B"X",     B"X" => B"1001",        B"001"; -- 6th clock cycle
     B"1001",        B"X",     B"X" => B"1010",        B"001"; -- 7th clock cycle
     B"1010",        B"X",     B"0" => B"0011",        B"001"; -- next DDC channel
     B"1010",        B"X",     B"1" => B"0000",        B"001"; -- finished
END TABLE;
ddcread_sm[].clk = !clock;  -- Invert clock to allow for LVDS setup times
ddcread_sm[].clrn = porta_reg[3];
ddc_sel.clk = !clock;
inc_addr.clk = !clock;
reset_addr.clk = !clock;

--
-- Daisy chain output controller
--
daisyout_clk = clock; -- transmit controller LVDS link

if porta_reg[0] == B"1" then
   --
   -- Code for sending all channels into idle state
   --
   addr_cnt[].ena = VCC;
   addr_cnt[].clrn = VCC;
   daisyout_data[9..6] = B"0111"; -- command mode idle
   daisyout_data[5..0] = addr_cnt[5..0]; -- address each channel into idle state
```

```
   elsif porta_reg[1] == B"1" then
      --
      -- Code for Programming DDCs
      -- See state machine above
      --
      case progddc_sel[1..0] is
         when B"00" =>
            daisyout_data[9..6] = B"0101";  -- Program DDC mode
            daisyout_data[5..0] = channel_reg[];  -- this is the channel number
         when B"01" =>
            daisyout_data[9..3] = B"0000111";  -- Send Address
            daisyout_data[2..0] = address_reg[];  -- This is the DDC address
         when B"10" =>
            daisyout_data[9..8] = B"00";  -- Send Data
            daisyout_data[7..0] = data_reg[];  -- This is the DDC data
         when B"11" =>
            daisyout_data[9..0] = B"0000011111";  -- sync pattern
      end case;

      elsif porta_reg[2] == B"1" then
         --
         -- Code for sending RAW data from one channel
         -- Takes 4 clock cycles for this command (see statemachine above)
         --
         if data_sel == B"0" then
            daisyout_data[9..6] = B"0100";  -- command mode raw
            daisyout_data[5..0] = channel_reg[5..0];-- this is the channel number
else
            daisyout_data[9..0] = B"0000011111";  -- sync pattern to keep lock and avoid RMT
   end if;

   elsif porta_reg[3] == B"1" then
      --
      -- Code for sending DDC data from all channels
      -- Takes 8 clock cycles for each DDC (see statemachine above)
      --
      addr_cnt[].ena = inc_addr; -- increments address counter
      addr_cnt[].clrn = reset_addr; -- resets the address counter

      if ddc_sel == B"1" then
         daisyout_data[9..6] = B"0110";  -- command mode DDC
         daisyout_data[5..0] = addr_cnt[5..0]; -- this is the DDC channel number
      else
         daisyout_data[9..0] = B"0000011111";  -- sync pattern to keep lock and avoid RMT
      end if;

   else
      daisyout_data[9..0] = B"0000011111"; -- sync pattern to keep lock and avoid RMT
   end if;

   --
   -- Daisy chain input controller
   --
   daisyin_refclk = clock; -- a reference clock for input LVDS data
   req = VCC;              -- always transmitting data
```

```
--
-- Two registers for raw data output
--
imag_data_reg[7..0].clk = daisyin_clk;
imag_data_reg[7..0].d = daisyin_data[7..0];
imag_data_reg[7..0].ena = daisyin_data[9];  -- write Q = imag
real_data_reg[7..0].clk = daisyin_clk;
real_data_reg[7..0].d = daisyin_data[7..0];
real_data_reg[7..0].ena = !daisyin_data[9];  -- write I = real

--
-- Four registers for DDC output data
--
real_low[].clk = daisyin_clk;  -- connect clocks
real_high[].clk = daisyin_clk;
imag_low[].clk = daisyin_clk;
imag_high[].clk = daisyin_clk;

real_low[].ena = ddc_reg_en[1];  -- connect enables
real_high[].ena = ddc_reg_en[0];
imag_low[].ena = ddc_reg_en[3];
imag_high[].ena = ddc_reg_en[2];

real_low[].d =  daisyin_data[7..0];  -- connect data
real_high[].d = daisyin_data[7..0];
imag_low[].d =  daisyin_data[7..0];
imag_high[].d = daisyin_data[7..0];

--
-- Address Counter for Reading (6-bit = up to 64 channels)
--
read_cnt[].clk = daisyin_clk;
read_cnt[].clrn = ddc_cnt_reset.q;
read_cnt[].ena = ddc_cnt_ena.q;
if read_cnt[].q == MAXADDRESS then  -- N-channels
   read_cnt[].d = B"000000";        -- reset address
else
   read_cnt[].d = read_cnt[].q + 1;
end if;
ddc_done = (read_cnt[].q == B"000000");
ddc_start = (daisyin_data[9..0] == B"0110000000"); -- detect start of DDC
                                                   -- data stream (channel 0)

--
-- State machine for writing DDC data to PC
--
ack_reg.d = ack;
ack_reg.clk = daisyin_clk;
dv_out.d = dv_out_in;
dv_out.clk = daisyin_clk;

ddc_reg_en[3..0].d = ddcout_data[7..4];
ddc_outdata.d = ddcout_data[3];
ddc_clk_out.d = ddcout_data[2];
ddc_cnt_reset.d = ddcout_data[1];
ddc_cnt_ena.d = ddcout_data[0];
```

```
   TABLE
   ddcout_sm[].q, ack_reg, dv_out, ddc_start, ddc_done => ddcout_sm[].d, ddcout_data[7..0];
      B"0000",    B"0",      B"X",      B"X",     B"X" => B"0001", B"00001000"; -- wait for ACK
      B"0000",    B"1",      B"X",      B"X",     B"X" => B"0110", B"00001000"; -- PC ready now
      B"0001",    B"X",      B"X",      B"X",     B"X" => B"0010", B"00001100";
      B"0010",    B"X",      B"X",      B"X",     B"X" => B"0011", B"00001100"; -- provide PCLK
      B"0011",    B"X",      B"X",      B"X",     B"X" => B"0000", B"00001000";

      B"0110",    B"X",      B"0",      B"X",     B"X" => B"0110", B"00001000"; -- wait for DVOUT
      B"0110",    B"X",      B"1",      B"X",     B"X" => B"0111", B"00001010";

      B"0111",    B"X",      B"X",      B"0",     B"X" => B"0111", B"00011010"; -- wait for DDC data
      B"0111",    B"X",      B"X",      B"1",     B"X" => B"1000", B"00011010"; -- write R-high
      B"1000",    B"X",      B"X",      B"X",     B"X" => B"1001", B"00100010"; -- write R-low
      B"1001",    B"X",      B"X",      B"X",     B"X" => B"1010", B"01000110"; -- write I-high
      B"1010",    B"X",      B"X",      B"X",     B"X" => B"1011", B"10000110"; -- write I-low
      B"1011",    B"X",      B"X",      B"X",     B"X" => B"1100", B"00000010"; -- idle
      B"1100",    B"X",      B"X",      B"X",     B"X" => B"1101", B"00001011"; -- idle
      B"1101",    B"X",      B"X",      B"X",     B"X" => B"1110", B"00001110"; -- idle
      B"1110",    B"X",      B"X",      B"X",     B"0" => B"1111", B"00001110"; -- next channel
      B"1110",    B"X",      B"X",      B"X",     B"1" => B"0110", B"00001110"; -- read MAXADDRESS
      B"1111",    B"X",      B"X",      B"X",     B"X" => B"1000", B"00011010"; -- write R-high
   END TABLE;
   ddcout_sm[].clk = daisyin_clk;
   ddcout_sm[].clrn = porta_reg[3];
   ddc_reg_en[].clk = daisyin_clk;
   ddc_clk_out.clk = daisyin_clk;
   ddc_outdata.clk = daisyin_clk;
   ddc_cnt_reset.clk = daisyin_clk;
   ddc_cnt_ena.clk = daisyin_clk;

   if porta_reg[2] == B"1" then          -- RAW data output to PC
      pclk = !daisyin_data[9];           -- 20MHz clock out (will have missing data)
      portd[7..0] = real_data_reg[7..0]; -- connect to PCI-DIO-32HS board
      portc[7..0] = imag_data_reg[7..0];

   elsif porta_reg[3] == B"1" then       -- DDC data output to PC
      pclk = ddc_clk_out;
      if ddc_outdata == B"0" then
         portd[7..0] = real_high[7..0];  -- connect to PCI-DIO-32HS board
         portc[7..0] = real_low[7..0];
      else
         portd[7..0] = imag_high[7..0];
         portc[7..0] = imag_low[7..0];
      end if;

   end if;
END;
```