User Guide
A 400MHz 1024-channel, 2-input, 10GbE Spectrometer for the Parkes Radio Telescope, v1.0

Peter McMahon
MeerKAT, University of Cape Town and CASPER*

April 2008, *updated* August 2008

Abstract

The "Parkes Spectrometer", v1.0, was built for the Parkes Radio Telescope as an evaluation design for a new multi-beam system. It is a dual polarization, power spectrometer with a 1024 channel Polyphase Filter Bank, using the CASPER IBOB hardware platform. A 30kHz readout rate (i.e. 30,000 spectra per second) was targeted, and this necessitated the use of 10Gbit/s Ethernet.

This User Guide details how to install and operate this spectrometer.

1 Introduction

The "Parkes Spectrometer" (henceforth "*Parspec*") is based on the IBOB hardware platform, and was built using the BWRC/CASPER[1] Simulink toolflow and DSP libraries [1].

This User Guide, as the name suggests, does not include detailed descriptions of the internals of the spectrometer design. Merely enough detail is

*Center for Astronomy Signal Processing and Electronics Research, University of California, Berkeley

[1]The Simulink toolflow for the BEE2 and IBOB boards was developed by the Berkeley Wireless Research Center, and the DSP libraries were developed by CASPER.

given to allow the user to take advantage of all the features and configuration options that are available.

2 Hardware Setup

Figure 1 shows an IBOB with labels on the relevant connectors and ports required for setting up the Parspec. To set up the hardware, perform the following steps:

1. Arrange airflow[2] for IBOB FPGA. This can take the form of a fan directly mounted on the heatsink, or a fan blowing from bottom-to-top or top-to-bottom over the heatsink (left-to-right, or right-to-left in Figure 1).

2. Connect the clock, 1PPS and analogue polarization signals ("Polarization 1" and "Polarization 2") to the ADC board. The clock should have level 0dBm, and the signals should ideally have power levels below -10dBm[3]. The 1PPS signal should be 0-2V minimum, 0-3V nominal, and 0-5V maximum, with 50Ω termination. The 1PPS connection is optional.

3. Connect the IBOB's 100Mbit Ethernet port to a control computer (whose IP address is modifiable) using standard Ethernet twisted-pair cable.

4. Connect the IBOB's 10Gbit Ethernet port labeled "XAUI1" to the data recorder computer – either directly (if the computer has a 10Gbit Ethernet NIC), or via a 10GbE/1GbE (CX4/RJ45) switch.

5. Connect the power cable to the IBOB. The red wire should be at +5V, and the black wire should be ground. The power supply should be able to supply at least 10A.

6. Turn on the power supply. If your board came with a pre-programmed PROM, then the design should be loaded from the PROM into the FPGA and the board is ready to be configured. If not, use a Xilinx JTAG programmer to program the FPGA with the Parspec design bitstream.

[2]A rule-of-thumb test during operation to verify that the airflow is sufficient is to make sure that the heatsink is not too hot to hold.

[3]Power levels above 0dBm may damage the hardware, and signals with power higher than -10dBm may cause overflow in the FFT.

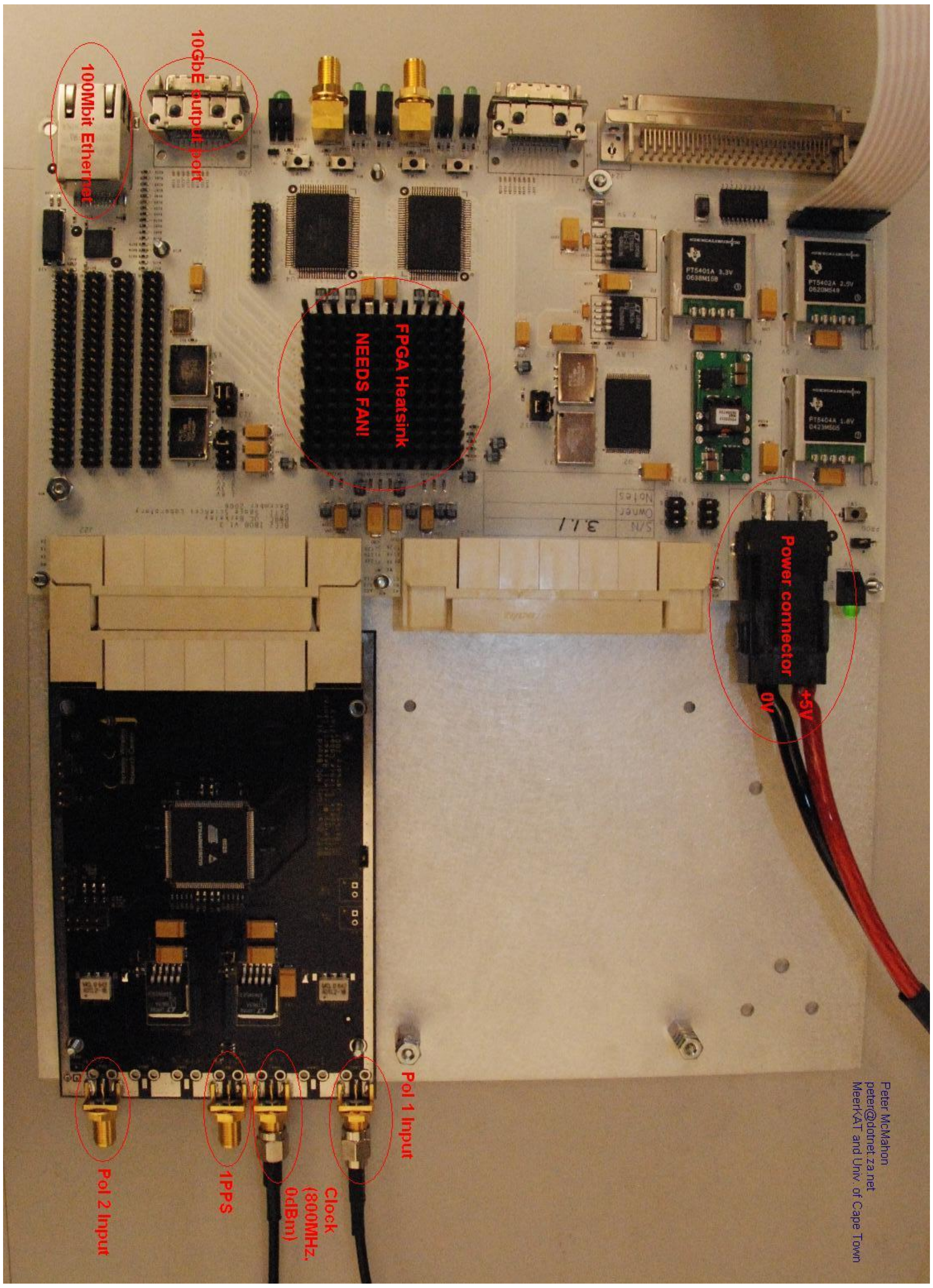

Figure 1: IBOB with labeled connectors and ports.

3 Software Configuration

There are two main categories of settings that need to be configured to enable successful operation of Parspec. These are *connectivity* settings and *spectrometer data* settings. This however is a categorization only for convenience in this guide, and in practice all configuration options are accessible in the same way on the IBOB.

All configuration of the design is done over a simple telnet terminal known as *TinyShell*. TinyShell features a set of commands that lets you interrogate and modify registers and RAMs in the Parspec design.

Once the IBOB is turned on, it should be possible to connect to the board over telnet (port 23) by running a telnet client on the control computer (see "Hardware Setup" section). The IBOB runs a telnet server on port 23. Your IBOB's IP address is defined by a set of jumpers on the board; if you don't know your board's IP address, the easiest way to check is to connect to the board using a serial connection[4] and type `ifconfig` (followed by `<enter>`).

It is important to note that it is **necessary to configure all the settings described in this guide** – none are optional.

3.1 Connectivity Settings

Convincing your IBOB to send data to your data recorder computer requires that the 10GbE connection be configured appropriately. It also may require some configuration changes on your switch (if applicable) and data recorder computer.

3.1.1 IBOB Configuration

You need to set up two IP addresses and two UDP ports: the sending IP address and port of the 10GbE interface/connection on the IBOB, and the destination IP address and port (i.e. that of the data recorder computer). You also need to inform the IBOB of the MAC addresses of both the sending and receiving interfaces (the sending interface, on the IBOB, is given an almost-arbitrary MAC address, but the receiving interface MAC address

[4]If you don't know how to do this, please contact your friendly local CASPER representative.

must be correct).

All settings are provided as integers, either in decimal or hexadecimal format, rather than as strings.

1. Set up the destination IP and UDP port using the following commands:
   ```
   regwrite reg_ip 0x0a000004
   regwrite reg_10GbE_destport0 4001
   ```
   This results in a destination IP address of 10.0.0.4 and port of 4001 being set (note that `0a` = 10, and it becomes obvious how the IP address is encoded as hex).

2. Set up the source interface MAC, source IP, source UDP port, and destination MAC in an ARP table using the following commands:
   ```
   line 1:  write l xd0000000 xffffffff
   line 2:  setb x40000000
   line 3:  writeb l 0 x00000060
   line 4:  writeb l 4 xdd47e301
   line 5:  writeb l 8 x00000000
   line 6:  writeb l 12 x0a000001
   line 7:  writeb b x16 x0f
   line 8:  writeb b x17 xa0
   line 9:  writeb l x3020 x00000030
   line 10:  writeb l x3024 x486377c1
   line 11:  writeb b x15 xff
   line 12:  write l xd0000000 x0
   ```

   These commands write to memory-mapped registers in the address-space.

   `line 1` isn't actually about connectivity at all, but rather the ADC mode. Since this is the section where we're describing the hardware commands, it seemed like an appropriate place to put it. Just enter it and forget about it.

   `line 2` sets the base address from which the remainder of the commands operate.

   `line 3` and `line 4` set the source MAC address. This is nearly arbitrary – just make sure it is a valid MAC address (in particular, the first two hex digits should be zeros). In this example, the MAC address is set to `00:60:dd:47:e3:01`.

`line 5` sets the gateway IP address. In this example, the gateway IP address is set to 0.0.0.0. This address does not need to be set correctly unless delivery to an IP on a different subnet is required.

`line 6` sets the source IP address. In this example, the source IP address is set to 10.0.0.1.

`line 7` and `line 8` set the source UDP port. In this example, this is set to 4000 (`0fa0` is the hexadecimal representation of 4000).

`line 9` and `line 10` set the ARP table entry for the destination IP address whose fourth (and final) byte is `04` (the first three bytes are assumed to be the same as those of the source IP address). In this example `3020` refers to the address where the first two bytes of the MAC for address 10.0.0.4 are stored, and `3024` refers to the address where the final four bytes of the MAC are stored. In this case, the MAC of the destination machine (IP 10.0.0.4) was set to `00:30:48:63:77:c1`. This needs to be set accurately.

Since the Parspec design only sends data to the single IP address specified using `regwrite reg_ip 0xXXYYZZWW`, you only need to make sure that you have an entry for IP `XXYYZZWW` in your ARP table. Note that it is assumed in the ARP table that first three bytes of the destination IP address are the same as the first three bytes of the source (10GbE) IP address. e.g. if you set the source 10GbE IP address to be 10.0.0.1, the destination IP is assumed to be of the form 10.0.0.x. It is also possible to send data to a destination IP that is not on the same subnet as the IBOB 10GbE interface. In this case, it is necessary to set the gateway in `line 5` and to create an appropriate ARP table entry for the gateway IP.

Clearly there are scenarios where you may wish to send data to an IP address whose final byte is not 4. This requires both setting the destination IP register accordingly, and creating the appropriate ARP entry. The ARP entries for the even IP addresses can be set as follows: the IP address x.x.x.2 entry is stored at address offsets `3010` and `3014`; x.x.x.4 at offsets `3020` and `3024`; x.x.x.6 at offsets `3030` and `3034`, and so on. The ARP entries for the odd IP addresses can be set as follows:

the IP address x.x.x.1 entry is stored at address offsets `3008` and `300c`; x.x.x.3 at offsets `3018` and `301c`; x.x.x.5 at offsets `3028` and `302c`, and so on. The entries from x.x.x.1 to x.x.x.255 are addressable.

3.1.2 10GbE/1GbE Switch Configuration

The switch needs to be set up to allow *jumbo* packets. The UDP payload size of Parspec packets is 2056 bytes.

3.1.3 Data Recorder Computer Configuration

The data recorder also needs to be set up to allow jumbo packets. In Linux, it is usually possible to allow large packets by setting the "MTU" to be sufficiently large with this command: `ifconfig ethX mtu 9000`. This sets the MTU on Ethernet interface ethX (this should be your 10GbE-connected interface) to 9000 bytes.

3.2 Spectrometer Data Settings

There are three settings that Parsec provides for manipulating the data output: the *accumulation length* (which necessarily constrains the output data rate), post-PFB (but pre-accumulator) *scaling*, and (post-accumulator) *bit selection*.

3.2.1 Accumulation Length

You need to set the accumulation length, which defines how many filter bank power outputs are added to each other before that output is transmitted. When you modify (including setting for the first time) the accumulation length, you need to modify what is known as the "sync period" setting too, according to a given formula.

The accumulation length is set using the register `reg_acclen`, with the caveat that the register value is one less then actual accumulation length: the command `regwrite reg_acclen 12` sets the accumulation length to 13. The minimum accumulation length is 2 (i.e. the smallest value you can set `reg_acclen` to is 1).

You need to set another register, `reg_sync_period`, to the following value[5]: $n{\cdot}k{\cdot}2048$, where k is the accumulation length (i.e. k is `reg_acclen`+1), and n is an arbitrary integer (we recommend that you use $n = 100$). For example, if $k = 13$ and $n = 100$, then $100 \cdot 13 \cdot 2048 = 2662400$. So to have an accumulation length of 13, you could call `regwrite reg_acclen 12` and `regwrite reg_sync_period 2662400`.

Incidentally, the accumulation length of 13 results in a spectral dump rate[6] of approximately 30,048 spectra per second, and hence a data rate of approximately 471Mbits/second[7]. At the minimum accumulation length of 2, the dump rate is approximately 195,313 spectra per second, and hence the data rate is approximately 2.99Gbits/second.

The maximum accumulation length is 1024 (which corresponds to a dump rate of approximately 381 spectra per second), but due to the possibility of overflow, it is only advisable to set such a large accumulation length if the input signal power is sufficiently low. The maximum accumulation length irrespective of input signal power is 256 (which corresponds to a dump rate of approximately 1,526 spectra per second).

For pulsar studies it is typically important to know the sampling (integration) time T_{sample}. This can be calculated directly from the accumulation length. Specifically, assuming that the ADC is clocked at 800MHz, $T_{sample} = [((\texttt{reg_acclen} + 1) \times 512)/200000000]$s[8].

3.2.2 Scaling and Bit Selection

The ADC samples data with 8-bits of precision, but in the FPGA design, this bitwidth is gradually increased to 32, which is the bitwidth of the data coming out of the accumulator. However, not all 32-bits are outputted over the 10GbE connection – in the final stage, 8 out of the 32 bits are selected.

[5]If you're curious about how this formula is derived, please see ref. [2]. For Parspec, the sync period is $n \cdot k \cdot \text{LCM(reorder orders)} \cdot \text{PFB taps} \cdot \frac{\text{FFT length}}{\text{\# simultaneous inputs}} = n \cdot k \cdot \text{LCM}(2, 2, 2) \cdot 2 \cdot \frac{2048}{4} = n \cdot k \cdot 2048$.

[6]These calculations assume a sampling clock rate of 800MHz.

[7]The reader may notice that this data rate is sufficiently low that it can easily be carried by 1GbE. Why then use an expensive and occasionally troublesome 10GbE connection? Simply because the 100Mbit connection on the IBOB is not sufficient to support the output data rate for 30kHz readout, and the next fastest Ethernet port available on the IBOB is the 10GbE port.

[8]This formula is derived using the fact that with a sampling clock of 800MHz, the FPGA is clocked at 800MHz/4=200MHz and a spectrum is produced every 512 clock cycles.

This selection is user-controlled, but it is not arbitrary: the user must pick one of four bit selection options: bits 0-7, 8-15, 16-23 or 24-31.

Which 8 of the 32 bits you select relies on several factors. You typically want to select the most significant bits that are not zero (possibly with some allowance for "room" for RFI), and which collection of 8 bits these most significant non-zero bits will be in depends primarily on: a. *accumulation length*, b. *input signal power*, and c. *the scaling parameter*. Since it is difficult to calculate which collection of 8 bits you should choose to output, and it is time-consuming and error-prone to use a trial-and-error approach, Parspec provides a quick way to check: type `bramdump scope_output1\bram` to see the full 32-bits for polarization 1, and `bramdump scope_output3\bram` for polarization 2.

The command "`bramdump`" displays the entire contents of a single block RAM. Without loss of generality let us discuss polarization 1 and `scope_output1\bram`. The contents of `scope_output1\bram` is the even[9] channels of the accumulated spectrum of polarization 1. With `bramdump`, each channel value will be outputted on a new line – the first column will be the address (i.e. channel number), and the third will be the binary representation. After the 512 even channels have been outputted, 1536 lines containing zeros will be displayed – you can disregard these lines.

Once you know which bits are toggling, you can set the bit selection parameter. For example, let's say that you note that the largest value in any bin you see in polarization 1 is `00000000000000000001000100110001`. Clearly if you're forced to output only 8 bits (you are), you would like to output the most significant non-zero bits. Therefore in this case, you would want to output the second set of 8 bits, namely bits 8-15 (which are `00010001` for this particular value).

The command to set the output bit selection is `regwrite reg_output_bitselect`

[9] "Why just the even channels?", you ask. Well, the motivation for this feature is to allow the user to get a sense of the overall power in the spectrum, and hence which bits the user should select. For this purpose looking at just one "typical" FFT bin should be sufficient, so being able to look at many bins is actually just a crutch to help you ensure that you don't accidentally output the wrong set of bits because you observed the levels of an atypical bin. We feel that seeing half the bins in the spectrum is sufficient for this purpose. However, it would be nice to allow the user to read out an entire spectrum for other reasons, but due to resource constraints on the FPGA this is not possible, and only the readout of half of the spectra of each of the two inputs is possible.

`X` where $X \in \{0, 1, 2, 3\}$. Specifically if $X = 0$, then the bits 0-7 are selected; $X = 1$ corresponds to bits 8-15 being selected, and so on. This bit selection setting applies to both polarizations[10].

What happens if the 32-bit test output shows that your bins have values that are one bit over the boundary of two collections of 8-bits? For example, let's say you have a maximum (and fairly typical) bin value of `00000001101101010000101100101001`. There is a single bit in the uppermost bit collection (bits 24-31) – this poses a problem: if you select the uppermost bits, then you will effectively only have 1 bit of precision, but if you select the bits 16-23 you will get invalid data (since the most significant non-zero bit is missing in some, perhaps all, cases!). You clearly wouldn't have this problem if you could select which 8 bits you want without any restrictions, and in this example, you could profitably select the bits 20-27, or some such range. Unfortunately such arbitrary selections aren't possible, but it is possible to effectively shift the bits using *scaling* to get the same result. Again using the example presented, suppose you set the bit selection to bits 16-23, and shifted the data to the right by 4 places (i.e. a division by $2^4 = 16$): you would get the same net result as not performing any shifting, and selecting bits 20-27.

Parspec doesn't provide a mechanism to shift bits directly, but it provides an equivalent: there is an option to multiple the FFT output by an arbitrary 18-bit number *before* the power detection (c.f. the block diagram in Appendix C). Two scaling parameters are provided: one for polarization 1 and another for polarization 2. To set the scaling factors use the command `regwrite reg_coeff_polA B` where $A \in \{1, 2\}$ is the polarization input and $B \in \{0, 1, 2, \ldots, 2^{18} - 1\}$ is the scaling value. The 18-bit scaling value is a fixed point number with binary point at 12, so the coefficient $2^{12} = 4096$ corresponds to a scaling of 1 (i.e. don't change the bits at all). Because the scaling happens before the power detection, you need to scale by the square-root of what you would otherwise do. To shift by one bit to the right in the output, you need to divide by two, and hence a coefficient of $4096/\sqrt{2} \approx 4096/1.414 \approx 2897$ would be used. Similarly shifting one bit to the left corresponds to multiplication by two, and hence the coefficient would be set to $4096 \times \sqrt{2} \approx 4096 \times 1.414 \approx 5792$. More generally, to shift to the right in the output by n bits, you need to set a scaling coefficient of $4096/\sqrt{2^n}$, and to shift to the left in the output by n bits, you need to set a

[10]You can use scaling to match polarization powers (it's assumed that the powers of the input signals will not be dramatically different).

scaling coefficient of $4096 \times \sqrt{2^n}$.

Caution should be exercised when setting scaling coefficients, and it is especially important to watch out for overflow and underflow. The scaling multiplication is implemented using saturation logic, so if you set a coefficient that is too high and results in the scaled value being saturated, you should see a saturated output emerge from the accumulator[11]. If you set the scaling coefficient too small, and underflow occurs, you should be able to detect this on the output by the lack of precision. In any case, suffice to say, you should use the scaling factors as fine-grained control over your bits – to move them between two adjacent bit collections – and not as a way to shift by more than 4 binary places. Also note that there is more precision available on the low-end than at the high-end, so it is better to divide (i.e. multiply by scaling values less than 1) than it is to increase the values of the data.

A basic plan for setting the bit selection and scaling coefficient parameters

Set your scaling coefficients to 1 (i.e. `reg_coeff_polX=4096`). Set your accumulation length to what you will use in practice, and then look at the full 32 bits of the accumulator output (for every second bin in a spectrum) using `bramdump scope_output1\bram` for polarization 1 (and `bramdump scope_output3\bram` for polarization 2). Work out which set of 8 bits contains most of the bits you would like to output (typically a couple of bits preceding, and the remainder of the bits following the most sigificant non-zero bit). For example, if a typical output is 00000001**01001011**1010011000011101, then you should pick bits 16-23 (in bold). You can then use the scaling to move the bits 3 places to the right so that 1.) most importantly, the most significant "1" bit appears in your selected 8-bits and 2.) your average power is what you want it to be (one might typically aim for an average power of approximately 40 on the scale of 0 to 255). In this example, you might want to shift the bits 3 places to the right, so you'd want to divide by 2^3, which implies that you should set your scaling coefficient to be $1/\sqrt{2^3}$. This means you should set `reg_coeff_polX=`$4096/\sqrt{2^3} \approx 4096/2.8284 \approx 1448$. When you do this, the scope for the same polarization should output something like: 00000000**00101011**01011111011000011. (Obviously the value won't be exactly the same, but the point is that the most significant non-zero bit will have moved to be in the output, and there is room to capture big pulses that

[11]This is true if the accumulation length is less than 256 – if it is more, then there is a possibility of overflow within the accumulator, and the accumulator does not contain saturation logic.

are far above the average power.)

We recommend that you set the scaling coefficients as close to 1 (i.e. `reg_coeff_polX=4096`) as possible to reduce the chance of overflow or underflow. Using the strategy described here should help you to do this.

4 10GbE Packet Format

The output packet format is very simple. The spectrometer outputs UDP packets whose payloads have the following structure:

Counter							
$P_1(0)$	$P_1(1)$	$P_2(0)$	$P_2(1)$	$P_1(2)$	$P_1(3)$	$P_2(2)$	$P_2(3)$
...							
$P_1(1020)$	$P_1(1021)$	$P_2(1020)$	$P_2(1021)$	$P_1(1022)$	$P_1(1023)$	$P_2(1022)$	$P_2(1023)$

A single packet contains a single 1024-channel spectrum for both polarizations. Here the counter is 64-bits wide, and all the remaining (data) entries are 8-bits wide. Thus the total size of a single packet payload is 2056 bytes. $P_x(y)$ is the power of polarization x in bin/channel y.

The counter in the packet is the value of an internal counter in the spectrometer that is only reset on an ARM/1PPS event (described in the section below). This counter increments every IBOB clock cycle (i.e. nominally it will increase in value by 200,000,000 every second). This results in the counter value incrementing by $acclen \times 512$ (where $acclen$ is the accumulation length, and is equal to `reg_acclen` $+ 1$) between each spectrum. Thus it is possible to detect dropped packets by inspecting the counter value of the most recently received packet, and if has incremented by an amount other than $acclen \times 512$ compared to the counter value in the next most recently received packet, then packet loss must have occurred.

4.1 Receiving 10GbE/1GbE Packets on the Data Recorder Computer

The Parspec IBOB will be connected either directly to a computer that has a 10GbE NIC, or to a 10GbE/1GbE switch via 10GbE, which is in turn connected to a computer using 1GbE. Regardless, the computer will receive a stream of UDP packets that have the format described in this section.

A quick way to test that the IBOB has been set up correctly and that packets are being outputted is to run a network protocol analyzer, such as Wireshark [3], on the data recorder computer. Because the data rate from the IBOB is expected to be high, you should only need to run a capture for approximately one second to save an ample amount of test data. To check that everything is working, you should verify that:

1. There are packets arriving on the Ethernet interface you expect them to.
2. The packet size of the packets is 2056 bytes.
3. The counter in the packet payload is increasing by the correct number (for an accumulation length of 13, the value the counter should be incremented by is $13 \times 512 = 6656$).
4. The payload contains spectral bin values that seem reasonable. To aid in this, it is helpful to have a test tone attached to one polarization input (and no signal on the other). Use the 32-bit test mode described in the *Scaling and Bit Selection* section to ensure that the 8-bits being outputted are correct, and then check in the UDP packet payload on the computer that the values appear as you expect them to (i.e. on a tone test, nearly all values should be zero, with non-zero values only at the spectral bin(s) corresponding to the tone frequency).

Once you have established that the packets are arriving at the computer reliably, and contain correct data, the next recommended test step is to capture data with gulp [4] (a network capture program that stores packets in pcap format) and process it. The command `gulp -i ethX > datadump.log` will capture packets directly from the interface `ethX` to the file `datadump.log` until the `gulp` process is killed. There is a sample C program that accompanies this User Guide that will process such a dump file and output the received spectral data as text files. (You would likely never actually do this in production, due to the increase in data size, but this program demonstrates nicely how to interpret pcap format packet dumps and the Parspec packet format. The output text files can easily be graphed in MATLAB to verify the correctness of the spectrum.)

5 Precise Timing using ARM and 1PPS

The purposes of the inclusion of a counter value in output packets are to enable the detection of packet loss, and to enable accurate timing of spectral data. When the IBOB is powered up, an internal counter starts counting with every clock cycle (nominally at 200MHz). Because the startup time is in general not known with precision, the counter value is, on its own, not very useful for time-tagging.

However, we provide a means to reset the counter at a precisely known time, and this enables the user to determine the time a spectrum arrived very accurately. The scheme is fairly simple: the control computer (the one connected to the IBOB via the 100MbitE port) must be set up to use NTP so that its clock is accurate to within a few tens of milliseconds. At approximately half-way through a chosen second (e.g. at approximately the time 14:26:53.5) the control computer should toggle the register `reg_arm` over telnet (i.e. call the command `regwrite reg_arm 0`, and then immediately follow this with `regwrite reg_arm 1`; the computer should be connected to the IBOB via telnet before doing this, so that it is guaranteed that this toggle will happen within approximately 0.4 seconds). This will *arm* the IBOB, which means that it is set to a state such that when the next 1PPS signal arrives (i.e. on the next second), the internal counter will reset to value 0. The counter will then not be reset until the next time the IBOB is re-armed and another 1PPS signal arrives. In our example, this means that at precisely (with the accuracy of the 1PPS signal and the IBOB clock source) the time 14:26:54.0, the counter will be reset. The data recorder computer can be informed that this is the case, and thus it will be aware of the precise time-stamp of each spectrum from that point on, based on the counter value.

Acknowledgements

The development of the Parkes Spectrometer was funded by the MeerKAT project, South Africa's National Research Foundation, and National Science Foundation Grant No. 0619596. Dan Werthimer was responsible for the conception of this project. We thank Glen Langston (with help from John Ford, Scott Ransom and Paul Demorest) at NRAO, and Andrew Jameson and Willem van Straten at Swinburne for beta testing the design and this documentation.

Appendix A: Specifications Sheet for Fast-readout Dual Spectrometer

Each Spectrometer	
Frequency channels:	1024 (2048 real samples per spectrum)
Signal input:	5MHz - 400MHz *or* 400MHz - 800MHz (2nd Nyquist zone) *or* 800MHz - 1.2GHz (3rd Nyquist zone) -20dBm to -10dBm (-15dBm nominal) 50Ω SMA
Integration time:	Minimum: 5.12μs (195kHz spectral dump rate) Maximum: 655μs (1.5kHz spectral dump rate)
Polyphase filter:	2 taps, Hamming window
Output:	Test mode: 100Mbit Ethernet. 32-bits per spectral bin. Observing mode: 10Gbit Ethernet. 8-bits per spectral bin.
Both Spectrometers	
Clock input:	800MHz, 0dBm to +4dBm, 50Ω SMA
1PPS input:	0 to 3V pulse nominal (into 50Ω) 2V minimum, 5V maximum. Optional.
Power input:	5V, 7A
Mechanical:	1x IBOB and 1x iADC board on a 6U, 8HP plate.
Control and monitor:	Set up accumulation and corresponding sync period. Set up IP addresses, ports, MAC addresses, and ARP table. Set scaling: 18-bits, binary point at 12. Set output bit selection. Set ARM (optional).

Appendix B: Sample Setup TinyShell Telnet Commands

```
// Set accumulation length and corresponding sync period
regwrite reg_acclen 12
regwrite reg_sync_period 1331200

// Set bitselection to output bits 8-15
regwrite reg_output_bitselect 1

// Set scaling coefficients to 1
regwrite reg_coeff_pol1 4096
regwrite reg_coeff_pol2 4096

// Set destination IP to 10.0.0.4 and destination port to 4001
regwrite reg_ip 0x0a000004
regwrite reg_10GbE_destport0 4001

// Move to 10GbE configuration
write l xd0000000 xffffffff
setb x40000000

// Set IBOB 10GbE MAC to 00:60:dd:47:e3:01
writeb l 0 x00000060
writeb l 4 xdd47e301

// Set Gateway IP address to 0.0.0.0
writeb l 8 x00000000
// Set IBOB 10GbE IP address to 10.0.0.1
writeb l 12 x0a000001
// Set IBOB 10GbE source port to 4000
writeb b x16 x0f
writeb b x17 xa0

// Set destination MAC address for IP x.x.x.4 to 00:30:48:63:77:c1
writeb l x3020 x00000030
writeb l x3024 x486377c1

writeb b x15 xff
write l xd0000000 x0
```

Appendix C: Block Diagram

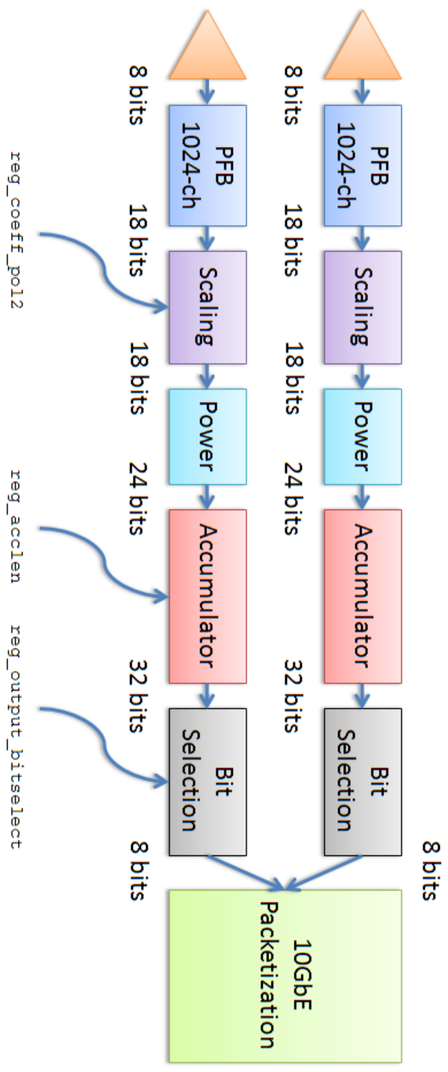

Figure 2: Block diagram of the Parspec FPGA design. The register inputs to the second spectrometer are shown.

References

[1] Aaron Parsons et. al. PetaOp/Second FPGA Signal Processing for SETI and Radio Astronomy. *Proc. 10th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, November 2006.

[2] Henry Chen, Peter McMahon and Aaron Parsons. Sync Pulse Usage in CASPER DSP Blocks. *CASPER Technical Memo 24*, August 2008.

[3] http://www.wireshark.org

[4] http://staff.washington.edu/corey/gulp/