# WBDC User's Guide

T. B. H. Kuiper

Revision of September 26, 2015

# Preface

**Chapter 1, Introduction,** is a quick overview of WideBand DownConverter receiver class and its capabilities.

**Chapter 2, Command Line,** gives basic commands for direct control of the equipment but also gives a simple, high-level view of the software organization.

**Chapter 3, Monitor and Control System,** provides the details of the monitor and control system, such as component addresses and latch[1] logic states.

**Chapter 4, Configurations,** explains how a WBDC is one element, the `Receiver`, in an over-all observing system. Configurations describe how the signals flow between these elements.

**Chapter ??, Servers and Clients** , describes the over-all monitor and control software architecture and how be be a client.

**Appendix A** gives a quick introduction to Python in which the M&C software is written. A convention used in this document is that if an object is shown capitalized (*e.g.* `Receiver` and in typewriter font, it is also a Python *class*. A class is a programming unit that serves as a template for objects that are similar to each other. In general, a single word in typewriter font refers to an attribute while one followed by empty parentheses is a method.

**Appendix B** describes the WBDC subsystems, with photos to identify them for diagnosis and repair.

**Appendix C** has the details of the LabJack configurations such as port functions (digital *vs* analog, input `vs` output, etc.

[1] A latch is an electronic logic device that retains the state (0 or 1) assigned to it.

**Appendix D** descibes the circular polarization convention used in astronomy and how two orthogonal linear polarizations can be converted to counter-rotating circular polarizations.

# Contents

# List of Figures

# List of Code Examples

# Acronyms and Technical Terms

**ADC** - analog-to-digital converter.
**AIN** - LabJack® analog input port.
**DAC** - digital-to-analog converter.
**DIO** - LabJack® digital I/O.
**IF** - intermediate frequency (signal)
**I/O** - input/output
**I/Q** - in-phase/quadrature-phase, the two components of a complex signal.
**K-band** - the frequency range 18-26.5 GHz.
**LO** - local oscillator.
**L/U** - lower sideband/upper sideband, a signal pair obtained from I/Q.
**MSB** - most significant bit (or byte, depending on context).
**PIN** - P-type/insulator/N-type, a type of diode.
**RF** - radio frequency (signal)
**WBDC** - wide-band down-converter, a class of receiver.

# Chapter 1

# Introduction

## 1.1 Hardware Overview

A Wide-Band Down-Converter processes dual-polarization K-band signals from two feed horns. There are two versions of the WBDC. The feed switching, polarization selection, and down-conversion are basically the same in both WBDC designs.

WBDC1 has two dual-polarization RF sections, one for 22 and one for 24 GHz. Two switch-selectable LOs determine which band is processed by the one IF section.

WBDC2 has five dual-polarization RF sections and 20 IF sections (2 feeds $\times$ 2 polarizations $\times$ 5 bands). The internals of WBDC2 are described in Subsection B.2.

All functions are controlled by a motherboard through LabJack® general-purpose I/O interfaces. The WBDC1 LabJacks are accessed through a USB hub by an external computer. WBDC2 LabJacks are controlled with a Raspberry Pi embedded computer accessed via ethernet.

### 1.1.1 Input Switching

The first stage of the WBDC allows the two dual-polarization down-converters to be switched between the two feeds.

### 1.1.2 Polarization Conversion

The four RF signals are split into two bands (22 and 24) in WBDC1 and five bands in WBDC2. After the band-selecting RF filters, the linearly polarized signals from each feed can be switched into a quadrature hybrid to be converted to circular polarization. This is discussed further in Appendix D PIN diode attenuators are provided to balance the signals into these hybrids.

### 1.1.3 Sideband Separation

In the closed modules on the far right are the down-converters, which have complex mixers putting out two IFs from 0 to 1000 MHz, with a 90° phase difference. Both IFs go from -1000 MHz to +1000 MHz but the negative and positive frequencies are mixed together. These may be considered the real and imaginary components of a complex signal. There are switches which can direct each I/Q pair into a quadrature hybrid to convert them to an upper and lower sideband pair.

## 1.2 Monitor and Control

Monitor and control is done with two or three LabJack® general purpose I/O devices.

### 1.2.1 LabJack 1

This LabJack with local ID 1 (on the center left of the motherboard) controls the WBDC motherboard. The motherboard has groups of eight latches which are used to control and sense switches, and to select analog monitoring points. The LabJack's ADC channels read currents, voltages, and temperatures.

### 1.2.2 Labjack 2 (and 3)

LabJack 2 (and 3 for WBDC2), on the box walls at the ends of the row of filters, control PIN diode attenuators. There are four voltage-controlled RF attenuators (PIN diodes) for each down-converter sub-band. The voltages are generated in LabJack TickDACs attached to the terminals of LabJack 2 (and 3 for WBDC2).

# Chapter 2

# Command Line

The program `wbdc2_prompt.py`[1] provides the user with a direct Python command line interface in a terminal session on the embedded Raspberry Pi. This command line has access to all the equipment in the specified configuration. It simply configures the hardware and software as described in Chapter 4.

To command the receiver only, the commands are simply[2]

```
In [1]: from MonitorControl.Receivers.WBDC.WBDC2.WBDC2hwif import WBDC2hwif
In [2]: receiver = WBDC2hwif("WBDC-2")
```

**Snippet 2.1:** Ipython example of creating a receiver object from a `WBDC2hwif` class.

The programming style used is "object oriented", so that a command takes the form `object.method()` where `object` is an "instance" of a "class'. In this implementation, classes may have private classes. Think of systems and subsystems, so that a command can also have the form `parent.child.method()`. Classes can also have instances of other public classes as "attributes", which leads to the same command structure. In the program `wbdc2_prompt.py`, `receiver` is an instance of the `WBDC2` class.

The Python `help()` command can be very useful at the command line. It can be invoked at any level of the hierarchy. These examples

```
help(receiver)
help(receiver.get_crossover)
help(receiver.crossSwitch)
help(receiver.crossSwitch.get_state)
```

will become clearer below.

[1] In `/usr/local/lib/python2.7/DSN-Sci-packages/MonitorControl/Receivers/WBDC/apps`.

[2] This example uses `ipython`, a powerful Python command line interface, described in Appendix A.

## 2.1 High-level Commands

At the moment the commands described here are only implemented for `WBDC_prompt`. When using a `WBDC2hwif` instance, mid-level and low-level commands must be used.

### 2.1.1 Feed Switch Control

The WBDC class of receivers have two feed horns with two linear polarizations, "E" and "H". There are four input channels called "F1E", "F1H", "F2E" and "F2H". These feed a pair of switches through which the receivers connect to the feeds. In the default state (logic state 0), "F1E"→"R1E" and "F1H"→"R1H". In the *set* state (logic state 1), "F1E"→"R2E" and "F1H"→"R2H". The commands for this are shown in Snippet 2.2. The keyword `crossover` is

```
receiver.get_crossover()
receiver.set_crossover(crossover=False)
```

**Snippet 2.2:** Sensing and controlling the transfer switch.

not required in this case but is mnemonically useful. The value `True` sets the switch.

### 2.1.2 Polarization Mode

Each of the four receiver sections feeds five bandpass filters. "Band 18" goes from 17 to 19 GHz, "Band 20" from 19 to 21, and so on to "Band 26" from 25 to 27 GHz. There are ten sections which can convert the linearly polarized modes "E" and "H" produced by the front end to circularly polarized modes "L" and "R".

The command for sensing and controlling the polarization mode are given in Snippet 2.3. `receiver.get_pol_modes()` returns a Python `dict` (short for "dictionary") which is a key:value

```
receiver.get_pol_modes()
receiver.set_pol_modes(circular=False)
```

**Snippet 2.3:** Sensing and controlling the polarization mode.

array with the states of all the polarization sections. The command `receiver.set_pol_modes()` sets all the polarization sections in the entire receiver to the designated state. The default state, that is, if `set_pol_modes()` is invoked without specifying an argument, is `False`. This command automatically invokes `get_pol_modes()` as its last step.

### 2.1.3 Down-conversion Mode

The outputs from the polarization sections go into down-converters which have names like "DC['R1-18P1']" (receiver 1, band 18, polarization 1) which reflects the fact that the polarization is now ambiguous. "P1" can be "E" or "L" while "P2" can be "H" or "R" depending on the state of the polarization section providing the signal.

There are quadrature hybrids which may be used to convert the in-phase and quad-phase IFs to lower sideband (-1000 to 0 MHz) and upper sideband (0 to 1000 MHz) IFs. The default state (logic 0) of these sections is L/U mode; the optional I/Q mode is logic state 1.

The commands for sensing and controlling the down-converter section IF mode is shown in Snippet 2.4. The first command returns a Python `dict` with the states of all the downconverters.

```
receiver.get_IF_mode()
set_IF_mode(SB_separated=False)
```

**Snippet 2.4:** Sensing and controlling the down-converter mode.

All the downconverters can be set to the same state with the command `set_IF_mode()`. It also returns a Python `dict` with the states of all the downconverters.

### 2.1.4 Analog Monitoring

The analog inputs to LabJack 1 are connected to various analog monitoring points. The power supply voltages, currents to various subsystems, temperatures at three locations and the power levels into the RF sections can all be monitored.

Analog inputs 0 and 1 return currents and voltages respectively. AIN2 and AIN3 return RF power and temperature. The address of the current and voltage monitor point is sent to latchgroup 1 at address 0. The address of the RF power and temperature is sent to latchgroup 1 at address 2.

The command `receiver.readAnalogs(latchAddress)` returns a Python dict with all the analog data available through that latch group.

### 2.1.5 Other

For completeness we note here one other high-level command which can be useful for debugging the equipment.

```
receiver.check_IO_config()
```

reports on the configuration of the WBDC2's three LabJack multi-purpose I/O devices. Section C.3 shows a typical output and the proper configuration of the LabJacks.

## 2.2 Mid-level Commands

These commands which can be used with the receiver sub-systems.

### 2.2.1 Feed Switch Control

The crossover switch subsystem can be addressed directly. The commands to check and set the state of the cross-over switch are shown in Snippet 2.5. These have the same effect as the

```
receiver.crossSwitch.get_state()
receiver.crossSwitch.set_state(True)
```

**Snippet 2.5:** Sensing and controlling the crossover switch directly.

commands shown in Snippet 2.2.

It can happen that the two switches are not in the same state. Then `get_crossover()` will give an error message. The state of the individual switches can be queried with the commands in Snippet 2.6. The switches can be set individually with similar `set_state()` commands.

```
receiver.crossSwitch['E'].get_state()
receiver.crossSwitch['H'].get_state()
```

**Snippet 2.6:** Sensing the state of the individual crossover switches.

### 2.2.2 Polarization Mode

The state of the individual polarization sections can be queried and set with the commands in Snippet 2.7. The *set* state (logic level 1) means that the conversion is made. The default state

```
receiver.pol_sec['R120'].get_state()
receiver.pol_sec['R120'].set_state(True)
receiver.pol_sec['R1-18'].atten['H'].set_atten(10)
```

**Snippet 2.7:** Commands to sense and set the mode of individual polarization sections and to balance their inputs.

is `False` (logic level 0). For the polarization sections to function properly, the two incoming signals must be balanced. For that purpose, each polarization section has two attenuators which can be adjusted. (The balancing procedure is described elsewhere.) The attenuation can be

queried with a corresponding `get_atten()` command but it does not read the hardware. It only reports what the last `set_atten()` command did. The attenuation specified is relative to the minimum attenuation of the attenuator, which is in the range of 7 to 9 dB.

The state of a single down-converter hybrid can be queried and set with commands like those in Snippet 2.8.

```
receiver.DC['R1-18P1'].get_state()
receiver.DC['R1-18P1'].set_state(True)
```

**Snippet 2.8:** Commands for sensing and setting a down-converter I/Q hybrid.

## 2.3 Low Level Commands

### 2.3.1 LabJacks

At the heart of the monitor and control subsystem are two (or three) LabJacks. (They can also provide other functions, like timers, which we don't use.)

There are eight I/O lines available at screw terminals on the unit. These can all function as digital inputs or outputs, or analog inputs, according to the value of some configuration bytes. They are called FIO0-FIO7 when functioning as digital ports and AIN0-AIN7 when functioning as analog ports. In addition, the DB25 connector on the LabJacks provides an additional 12 digital IO lines, called EIO0-EIO7 and CIO0-CIO3. Each LabJack also has two analog outputs called DAC0 and DAC1, which we don't use.

Communication with the LabJack consists mainly of reading and writing bytes to the FIO, DIO and CIO groups.

### 2.3.2 Digital Monitor and Control

All the switches described above are controlled by latches. The latches are logically grouped, as shown in Table 2.1, with up to eight latches per group. Latch groups are read as bytes, as shown by the examples in Snippet 2.9. The bit assignment for each latch group is described in

```
In [7]: Math.decimal_to_binary(receiver.lg['P2I2'].read())
Out[7]: '00000000'
In [12]: Math.decimal_to_binary(receiver.lg['PLL'].read())
Out[12]: '01110100'
```

**Snippet 2.9:** Setting a latch group.

Table 2.1: Latch Groups for Digital Monitor and Control

| Latch Group | Function |
|---|---|
| X | controls the cross-over switches |
| R1P | controls the receiver 1 polarization sections |
| R2P | controls the receiver 1 polarization sections |
| PLL | monitors the cross-over switch state and LO phase lock |
| R1I1 | receiver 1 IF group 1 (bands 18 and 20) |
| R1I2 | receiver 1 IF group 2 (bands 22, 24 and 26) |
| R2I1 | receiver 2 IF group 1 (bands 18 and 20) |
| R2I2 | receiver 2 IF group 2 (bands 22, 24 and 26) |

Subsection 3.1.1. There is a corresponding `write()` command which could be used in this way

```
receiver.lg['X'].write(int('00000011',2))
```

to set the cross-over switches.

Setting an individual latch involves

1. reading the byte from the latch group of interest,
2. changing the desired bit, and
3. writing the modified byte to the latch group.

### 2.3.3 Attenuator Control

The attenuators are PIN diodes controlled with a voltage (or more strictly, the current resulting from an applied voltage). The voltages are supplied by dual-channel ("A" and "B") digital-to-analog converters. The DACs are connected to a LabJack multi-function IO interface.

In the example in Snippet 2.10 an instance of a DAC is created and the A side is set to -2 V. The first way avoids creating an enduring instance of the DAC. However, it is pretty

```
from Receiver.Interfaces.LabJack import LJTickDAC

LJTickDAC(receiver.LJ[2],'R1-18')['A'].setVoltage(-2)

vs = LJTickDAC(receiver.LJ[2],'R1-18')['A']
vs.setVoltage(-2)
```

**Snippet 2.10:** Two ways of instantiating and setting a DAC.

cumbersome so the second way, which creates an instance of a DAC, although taking two steps, is usually preferred. And there might be need to use `vs` again.

The LJTickDAC is an "attribute" of the polarization section and so also be addressed as `receiver.pol_sec['R1-18'].tdac['A']`.

Using this command to control the attenuation requires knowledge of the attenuation *vs* control voltage curve. This is discussed elsewhere.

# Chapter 3

# Monitor and Control System

## 3.1 Motherboard Control

Motherboard signals are monitored and controlled with digital latches (logical devices which preserve a specified logic state). The latches are connected to serial-in/parallel-out registers. These registers are addressed using the EIO out bits of LabJack 1. The data are then clocked into the register using the SDI and SCK signals. Table 3.1 shows how the LabJack ports are used to control the motherboard signals. The state of NLOAD is normallly high. When checking status (or reading bits), NLOAD is toggled low for at least 10 mS then high to store the information for reading. CS-BUS is the global enable. The normal state is high. This is set to low when programming latches or reading data. It is set high when done.

LabJack 1 is capable of addressing 256 latches on a WBDC motherboard. They are grouped in 32 sets of 8 latches. The first four of the latch groups, at addresses 0–3, 8–11, 16–19, ..., 80–83, ..., 248–251, are writing to control bits. The second four of the latch group, at addresses 4–7, 12–15, 20–23, ...,84–87, ..., 252–255, are for reading from control or status bits (*e.g.* switch position indicators).

Latch group address bits A0-A7 (ports EIO0–EIO7 on LabJack 1) are used to address a specific 8-bit latch group for writing or reading. The latch data are bytes sent or read serially, MSB first. The latch group address encoded with the EIO bits consists of three parts:

**EIO7-EIO3** encodes the motherboard digital module (DM = 1, 2, or 3) address.
**EIO2** indicates write if 0 and read if 1.
**EIO1-EIO0** along with EIO2, selects the latch group (LG = 1-4) in a digital module.

So latches in write-mode have addresses ending in 0–3. Latches in read-mode have addresses ending in 4–7.

The DM 1 bit address assignment differs for WBDC1 and WBDC2. DM 1 LG 1 has a EIO value (address) of 80 (0101 0000) in WBDC1 and 8 (0000 1000) in WBDC2. LG 2 in any DM has EIO0=1. WBDC2 DM 2 LG 1 has address 16 (0001 0000).

Table 3.1: Motherboard Signals and LabJack Ports

| **Labjack** | | **WBDC Motherboard** | |
|---|---|---|---|
| **Name** | **Channel** | **Name** | **Function** |
| AIN0 | 0 | IMON | Analog input measuring supply currents |
| AIN1 | 1 | VMON | Analog Input measuring supply voltages |
| AIN2 | 2 | TEMP | Analog input measuring temperatures |
| AIN3 | 3 | RF | Analog input measuring RF power |
| FI04 | 4 | | |
| FI05 | 5 | | |
| FI06 | 6 | | |
| FI07 | 7 | SDO | Digital input from readback |
| EI00 | 8 | A0 | Latch address LSB |
| EI01 | 9 | A1 | .. |
| EI02 | 10 | A2 | .. |
| EI03 | 11 | A3 | .. |
| EI04 | 12 | A4 | .. |
| EI05 | 13 | A5 | .. |
| EI06 | 14 | A6 | .. |
| EI07 | 15 | A7 | Latch address MSB |
| CI00 | 16 | SCK | Serial Clock input |
| CI01 | 17 | SDI | Data input |
| CI02 | 18 | NLOAD | Load data to/from latches |
| CI03 | 19 | CS-BUS | Eable writing to or reading from latch |

### 3.1.1 Digital Module 1

Seven latch groups are assigned to digital monitoring, that is, the program reads the latch states.

#### Feed Crossover Switch

The feed crossover switches are controlled with DM 1 LG 1 using bits L1A0 and L1A1 (A0, A1 on latchgroup 1): logic 0 for through and logic 1 for crossed. The commanded state can be read at the same bits (A0, A1 at LG address 12). However, the actual states of the cross-over switches are read at L4A0 and L4A1 (LG 3).

#### Polarization Hybrids

**WBDC1** (to be provided).

**WBDC2** receiver chain 1 polarization hybrids are controlled by latchgroups 2 and 3. The receiver chain 1 polarization hybrids are controlled by L2A0–L2A4. The receiver chain 2 polarization hybrids are controlled by L3A0–L3A4.

| Band | Control Bits |
|---|---|
| 18 | L2A0 L3A0 |
| 20 | L2A1 L3A1 |
| 22 | L2A2 L3A2 |
| 24 | L2A3 L3A3 |
| 26 | L2A4 L3A4 |

Logic level 0 is for bypassing the hybrids, that is, linear polarization. Logic level 1 is for converting linear to circular, that is, X and Y polarization to L and R.

### Local Oscillator Lock

Latch Group 4 monitors the state of the transfer switches and the local oscillator phase-locked loops

| Pol | Status Bit |
|---|---|
| X | L4A0 |
| Y | L4A1 |
| **Band** | **Status Bit** |
| 18 | L4A2 |
| 20 | L4A3 |
| 22 | L4A4 |
| 24 | L4A5 |
| 26 | L4A6 |

### Latch LEDs

The monitor bits should match the LEDs in the respective digital module. In WBDC1 the DM is to the top right With the hinge of the lid at the bottom, the LabJacks are upside down. The LEDs are in LSB -> MSB order and grouped as

```
LG 3 address 82 or 86    LG 4 address 83 or 87
LG 1 address 80 or 84    LG 2 address 81 or 85
```

If the box is mounted on a wall or ceiling and the lid is hanging down, the order is more conventional.

In WBDC2 there are three DMs at the top left of the lids, numbered from left to right. DM 3 is a spare. The (upside down) LED ordering is

```
LG addr 10 or 14   LG addr 11 or 15         LG addr 18 or 22   LG addr 19 or 23
LG addr  8 or 12   LG addr  9 or 13         LG addr 16 or 20   LG addr 17 or 21
```

### 3.1.2 Digital Module 2

This only applies to WBDC2. Four latch groups are used to control the I/Q hybrids. Receiver chain 1 I/Q hybrids are controlled by latch groups 1 and 2 (low address). Receiver chain 2 I/Q hybrids are controlled by latch groups 3 and 4 (low address). The two bypass switches for each hybrid are controlled with one bit. Logic level 1 is for IQ; logic level 0 is for LU. This is the bit assigment:

| | Rec 1 | | Rec 2 | |
|---|---|---|---|---|
| Band | Pol1 | Pol2 | Pol1 | Pol2 |
| 18 | L1A0 | L1A1 | L3A0 | L3A1 |
| 20 | L1A2 | L1A3 | L3A2 | L3A3 |
| 22 | L2A0 | L2A1 | L4A0 | L4A1 |
| 24 | L2A2 | L2A3 | L4A1 | L4A2 |
| 26 | L2A4 | L2A5 | L4A3 | L4A4 |

### 3.1.3 WBDC1 Analog Monitoring

Latch addresses 0 and 2 address the registers used to select analog monitoring points. bit pattern sent to latch address 0 selects a current and a voltage to be connected to AIN0 and AIN1. A bit pattern sent to latch address 2 selects a thermistor to be connected to AIN2. AIN3 is not used in this receiver.

(Register addresses to be provided.)

### 3.1.4 WBDC2 Analog Monitoring

#### Latch 1 (Address 0)

This latch addresses the register used to select voltage and current monitoring points. Voltage points are selected using bits 0-2 and read at AIN1. Currents are selected using bits 3-6 and read at AIN0. The addresses can be ANDed to select any voltage point with any current point.

The monitor point address bits for voltage are shown in Table 3.2. The actual voltages are the byte read times the scale factor shown in Table 3.2

The monitor point addresses for currents are shown in Table 3.3. The actual currents are the value read offset by -0.026.

Table 3.2: Monitor Points for Voltages

| **Latch** | | | | | |
|---|---|---|---|---|---|
| **Address** | **Bits** | **Input** | **Supply** | | **Scale** |
| 0 | xxxxx000 | AIN1 | +6 V | digital | 4.0211 |
| 0 | xxxxx001 | AIN1 | +6 V | analog | 4.0278 |
| 0 | xxxxx010 | AIN1 | +16 V | | 10.5446 |
| 0 | xxxxx011 | AIN1 | +12 V | | 10.5827 |
| 0 | xxxxx100 | AIN1 | -16 V | | -10.5446 |

Table 3.3: Monitor Points for Currents

| **Latch** | | | | |
|---|---|---|---|---|
| **Address** | **Bits** | **Input** | **Supply** | |
| 0 | x0000xxx | AIN0 | +6 V | MB digital |
| 0 | x0001xxx | AIN0 | +6 V | MB analog |
| 0 | x0010xxx | AIN0 | -16 V | MB |
| 0 | x0011xxx | AIN0 | +16 V | R1 FE |
| 0 | x0100xxx | AIN0 | +16 V | R2 FE |
| 0 | x0101xxx | AIN0 | +16 V | R1 BE |
| 0 | x0110xxx | AIN0 | +16 V | R2 BE |
| 0 | x0111xxx | AIN0 | +16 V | LDROs |
| 0 | x1000xxx | AIN0 | +16 V | MB |
| 0 | x1001xxx | AIN0 | +6 V | R1 FE |
| 0 | x1010xxx | AIN0 | +6 V | R2 FE |
| 0 | x1011xxx | AIN0 | -16 V | R1 FE |
| 0 | x1100xxx | AIN0 | -16 V | R2 FE |
| 0 | x1101xxx | AIN0 | -16 V | R1 BE |
| 0 | x1110xxx | AIN0 | -16 V | R2 BE |

### Latch 2 (Address 1)

This latch selects temperature monitor point with bits 0–2 to be read at AIN3 and RF detector voltages with bits 3–6 to be read at AIN2. Table 3.4 gives the monitor address bits for temperature. The conversion from measured volts to temperature is

$$T = (V_{AIN3} + 0.2389275) * 23.549481$$

The monitor points for RF detector power are given by Table 3.5. [1] The actual reading is

[1] The bit patterns for the detectors still need to be verified.

Table 3.4: Monitor Points for Temperatures

| Address | Bits | Input | Supply | Location |
|---|---|---|---|---|
| 1 | xxxxx000 | AIN3 | | R1 RF plate |
| 1 | xxxxx001 | AIN3 | | R2 RF plate |
| 1 | xxxxx010 | AIN3 | | BE plate |

Table 3.5: Monitor Points for RF Detectors

| Address | Bits | Input | Location |
|---|---|---|---|
| 1 | x0000xxx | AIN2 | R1 E-plane |
| 1 | x0001xxx | AIN2 | R2 E-plane |
| 1 | x0010xxx | AIN2 | R1 H-plane |
| 1 | x0011xxx | AIN2 | R2 H-plane |

(data-0.004)*2.0064.

# Chapter 4

# Configurations

A WBDC is one element of a receiving system, specifically a `Receiver`. The receiver obtains its signals from a front end and delivers signals to a backend. A configuration is a description of the signal flow from the point where the antenna focuses radiation on a feed to the point where data are processed by a computer. At the very minimum, a receiver must know something about where its signal comes from and what properties were imposed by the preceding equipment.

Configurations are stored in the `DSN-Sci-packages` directory under `MonitorControl/Configurations`. The configuration used by the program `wbdc2_prompt.py` is shown in Snippet 4.1. This shows that the `Receiver` is a class `WBDC2` device with the name "WBDC2". It has four inputs named "F1E", "F1H", "F2E" and "F2H" which were introduced in Subsection 2.1.1. They obtain their signals from front end outputs with the same names. They don't have to be the same but it was natural in this case. In general, port names can be the label for the port on the device.

In this case the description stops at the WBDC. However, a phantom `FrontEnd` object (one that can't be controlled or monitored) had to be defined because the `Receiver` expects a signal source. The `FrontEnd` needs a `Telescope` for a signal.

The `Observatory` provides a context for the `Telescope`. There are devices which depend on the observatory, rather than a particular telescope. The `Telescope` object has one output which provides the signal to one or more input ports of the `FrontEnd`. The `Observatory` object has information not specific to a `Telescope`, such as weather data.

The above configuration description does not yet contain any back-ends, since no back-end servers have yet been implemented at CDSCC. However, the distribution of IF signals from WBDC2 to down-stream devives has been implemented through a manually controlled patch panel. The description of the patching is maintained in a spreadsheet, which can be interrogated this way:

```
from MonitorControl import Observatory, Telescope, ClassInstance
from MonitorControl.FrontEnds import FrontEnd
from MonitorControl.FrontEnds.K_band import K_4ch
from MonitorControl.Receivers import Receiver
from MonitorControl.Receivers.WBDC.WBDC2 import WBDC2

import logging
module_logger = logging.getLogger(__name__)

def station_configuration(roach_loglevel=logging.WARNING):
  """
  Configuration for the K-band system on DSS-43 using WBDC2

  Feed 1 (F1) is at 024-0.016, F2 at 024+0.016.  The polarizations are linear,
  E and H.  There are so many receiver outputs that it is simpler to let the
  software generate them.
  """
  observatory = Observatory("Canberra")
  telescope = Telescope(observatory, dss=43)
  front_end = ClassInstance(FrontEnd, K_4ch, "K",
                    inputs = {'KF1': telescope.outputs[telescope.name],
                              'KF2': telescope.outputs[telescope.name]},
                    output_names = [['F1E','F1H'],
                                    ['F2E','F2H']])
  IFswitch = None
  receiver = ClassInstance(Receiver, WBDC2, "WBDC-2",
                           inputs = {'F1E': front_end.outputs["F1E"],
                                     'F1H': front_end.outputs["F1H"],
                                     'F2E': front_end.outputs["F2E"],
                                     'F2H': front_end.outputs["F2H"]})
  clock = None
  backend = None
  return observatory, telescope, front_end, receiver, IFswitch, clock, backend
```

**Snippet 4.1:** Configuration description used by the program `wbdc2_prompt.py` to test WBDC2 in the lab.

```
In [6]: from MonitorControl.Configurations.CDSCC.FO_patching import \
        DistributionAssembly
In [7]: da = DistributionAssembly()
In [8]: da.report_patching()
Out[8]:
  { 1: 'R1-18P1IF1',  2: 'R1-18P1IF2',  3: 'R1-20P1IF1',  4: 'R1-20P1IF2',
    5: 'R1-22P1IF1',  6: 'R1-22P1IF2',  7: 'R1-22P2IF1',  8: 'R1-22P2IF2',
    9: 'R2-22P1IF1', 10: 'R2-22P1IF2', 11: 'R2-22P2IF1', 12: 'R2-22P2IF2',
   13: 'R1-24P1IF1', 14: 'R1-24P1IF2', 15: 'R1-26P1IF1', 16: 'R1-26P1IF2'}
```

# Chapter 5

# Servers and Clients

The entire system described in the previous chapter is operated through a collection of servers. Our goal is to have each device controlled by a servers and to control the entire system through a top server which controls all the others. The top server should "know" the state of the entire system at all times.

All servers and clients communicate using the Pyro package. This package must also be installed on any client. A local object which provides the remote server functionality is obtained with this function:

```
In [2]: from support.pyro import get_device_server
```

## 5.1 K-band System Server

## 5.2 Front-end Server

```
In [1]: from support.pyro import get_device_server
In [2]: K2 = get_device_server("K2_Server-crux", "crux")

In [9]: K2.set_WBDC(14)   # set front end in loads
In [10]: K2.set_WBDC(16)

In [11]: K2.read_pms()
Out[11]:
  [(1, 'Tue Jul 21 20:11:05 2015', -40.347999999999999),
   (2, 'Tue Jul 21 20:11:05 2015', -39.119999999999997),
   (3, 'Tue Jul 21 20:11:05 2015', -40.502000000000002),
   (4, 'Tue Jul 21 20:11:05 2015', -40.573999999999998)]
```

```
In [12]: K2.set_WBDC(13)  # set front ends in sky
In [13]: K2.set_WBDC(15)

In [14]: K2.read_pms()
Out[14]:
  [(1, 'Tue Jul 21 20:11:52 2015', -45.186999999999998),
   (2, 'Tue Jul 21 20:11:52 2015', -43.600999999999999),
   (3, 'Tue Jul 21 20:11:52 2015', -45.0),
   (4, 'Tue Jul 21 20:11:52 2015', -45.392000000000003)]
```

## 5.3 WBDC1 Server

## 5.4 WBDC2 Server

A local object corresponding to the remote WBDC2 controller is obtained this way:

```
In [3]: wbdc = get_device_server("wbdc2hw_server-dss43wbdc2","crux")
```

`wbdc2hw_server-dss43wbdc2` is the host name of the Raspberry Pi inside WBDC2. `crux` is the host running the Pyro server.

### 5.4.1 Examples

**Polarization Sections**

```
In [6]: wbdc.get_pol_sec_states()
Out[6]:
  {'R1-18': 0, 'R1-20': 0, 'R1-22': 0, 'R1-24': 0, 'R1-26': 0,
   'R2-18': 0, 'R2-20': 0, 'R2-22': 0, 'R2-24': 0, 'R2-26': 0}
```

**Down-converter (Mixer) Sections**

```
In [7]: wbdc.get_DC_states()
Out[7]:
  {'R1-18P1': 0, 'R1-18P2': 0, 'R1-20P1': 0, 'R1-20P2': 0, 'R1-22P1': 0,
   'R1-22P2': 0, 'R1-24P1': 0, 'R1-24P2': 0, 'R1-26P1': 0, 'R1-26P2': 0,
   'R2-18P1': 0, 'R2-18P2': 0, 'R2-20P1': 0, 'R2-20P2': 0, 'R2-22P1': 0,
   'R2-22P2': 0, 'R2-24P1': 0, 'R2-24P2': 0, 'R2-26P1': 0, 'R2-26P2': 0}
```

## 5.5 Radiometer Server

# Appendix A

# Un Soupçon de Python

The DSN Radio Astronomy Monitor and Control system uses the object-oriented features of Python. Arguably, object-oriented program can be said to be based on Plato's theory of forms. Python objects are analogous to real objects while classes correspond to forms or ideals, which are abstractions based on real objects.

## A.1 iPython

One can simply type `python` at a shell prompt to get the standard Python command line interface. However, `ipython` is preferable because it has powerful extra features. One that is used most often is *tab completion* illustrated in Snippet A.1. This is very useful if you can't remember the name of an attribute or method.

One can also create an instance of a specific subsystem, as shown in Snippet A.2.

## A.2 Classes

A class is a natural way of describing a real-world object which has properties (*attributes*) and can do things (*methods* or functions). The most basic class in the M&C system is `Device` which has the attributes `name`, `inputs` and `outputs`, which are instance of `Port` classes, and `data` about the device, such as the location of a telescope or the bandwidth of a receiver. Most classes discused here are sub-classes of `Device`, which means that they "inherit" these attributes.

In the hope that it is more enlightening than confusing, Snippet A.3 (page 25) gives an example of a class definition. The `__init__()` method creates an instance of this class[1], which inherits attributes from the class `Device`. The second line creates the object and the rest of the code assigns values to its attributes.

[1]Creating an object from a class is called *instantiation*, making an `instance` of a class.

```
In [2]: receiver
Out[2]: WBDC2 "WBDC-2"
In [3]: receiver.crossSwitch
Out[3]: TransferSwitch "WBDC transfer switch"
In [4]: receiver.crossSwitch.keys()
Out[4]: ['H', 'E']
In [5]: receiver.crossSwitch.Xswitch
Out[5]: MonitorControl.Receivers.WBDC.WBDC2.Xswitch
In [6]: receiver.crossSwitch.Xswitch.<Tab>
receiver.crossSwitch.Xswitch.base
receiver.crossSwitch.Xswitch.get_state
receiver.crossSwitch.Xswitch.has_key
receiver.crossSwitch.Xswitch.keys
receiver.crossSwitch.Xswitch.mro
receiver.crossSwitch.Xswitch.set_state
receiver.crossSwitch.Xswitch.update_signals
```

**Snippet A.1:** Example of creating a `WBDC2` instance and then getting the classes corresponding to its `crossSwitch` and `crossSwitch.Xswitch` attributes, and then a list of the latter's attributes and methods.

```
In [1]: from Electronics.Instruments.PINatten import PINattenuator
In [2]: from Electronics.Interfaces.LabJack import *
In [3]: lj = connect_to_U3s()
In [4]: tdac = LJTickDAC(lj[2],'R1-18')
In [5]: vs = tdac['A']
In [8]: att = PINattenuator('R1-18-H', vs)
In [9]: att.<Tab>
att.VS                    att.define_coefs          att.set_atten
att.atten                 att.fit_calibration_data  att.useful_range
att.base                  att.get_atten
att.controlVoltage        att.name
```

**Snippet A.2:** Creating an instance of a `PINattenuator` and discovering its attributes and methods.

```
class Telescope(Device):
  def __init__(self, obs, dss=0, LO=None, active=True):
    name = "DSS-"+str(dss)
    Device.__init__(self, name)
    self.inputs = {obs.name:obs}
    self['longitude'], self['latitude'], self['elevation'], tz, name, diam = \
            get_geodetic_coords(dss=int(dss))
    self['geo-x'],self['geo-y'],self['geo-z'] = \
            get_cartesian_coordinates('DSS '+str(dss))
    self.outputs[self.name] = Port(self, self.name, signal=Beam(str(dss)))
    self.outputs[self.name].signal['dss'] = dss
```

**Snippet A.3:** Stripped-down definition of the `Telescope` class.

# Appendix B

# WBDC Hardware

## B.1 WBDC1

## B.2 WBDC2

Figure B.2 shows the interior of WBDC2. Mounted in the DSS-43 R&D cone the side at the bottom of the photograph will be at the top, with the lid hanging down when open.

The motherboard is on the lid. The RF enters through four SMA bulkhead feed-throughs on the left. The IFs exit on the right. The crossover switches are the two modules in the center to the far left in Figure B.2. The RF splitters which feed the bandpass filters (row of modules to the left of center) are the larger blue modules at the top and bottom on the far left. The PIN diodes ahead of the polarization sections are under the black cables and DB9 connectors. The quadrature hybrids for polarization mode conversion are to the right of the black DB9 connectors.

Figure B.3 shows the twenty down-converter sections, each of which has two outputs.

The outputs and their labels are shown in Figure B.4. The polarizations are labeled "H" and "E" which refers to the native polarization of the orthomode outputs. These are the polarizations of the outputs if the polarization hybrids are bypassed. Since the polarizations could also be "L" and "R", the labels "P1" and "P2" are also used in other contexts.

| hybrid state: | 0 | 1 |
|---|---|---|
| P1 | E | L |
| P2 | H | R |

| hybrid state: | 1 | 0 |
|---|---|---|
| IF1 | I | U |
| IF2 | Q | L |

The ports are also labeled "Q/L" and "I/U" which refers to the types of IF which depend on the states of the down-converter quadrature hybrids. Since the IF type depends on the state of the down-converter hybrids, the terms "IF1" and "IF2" also also used. Note that the logic

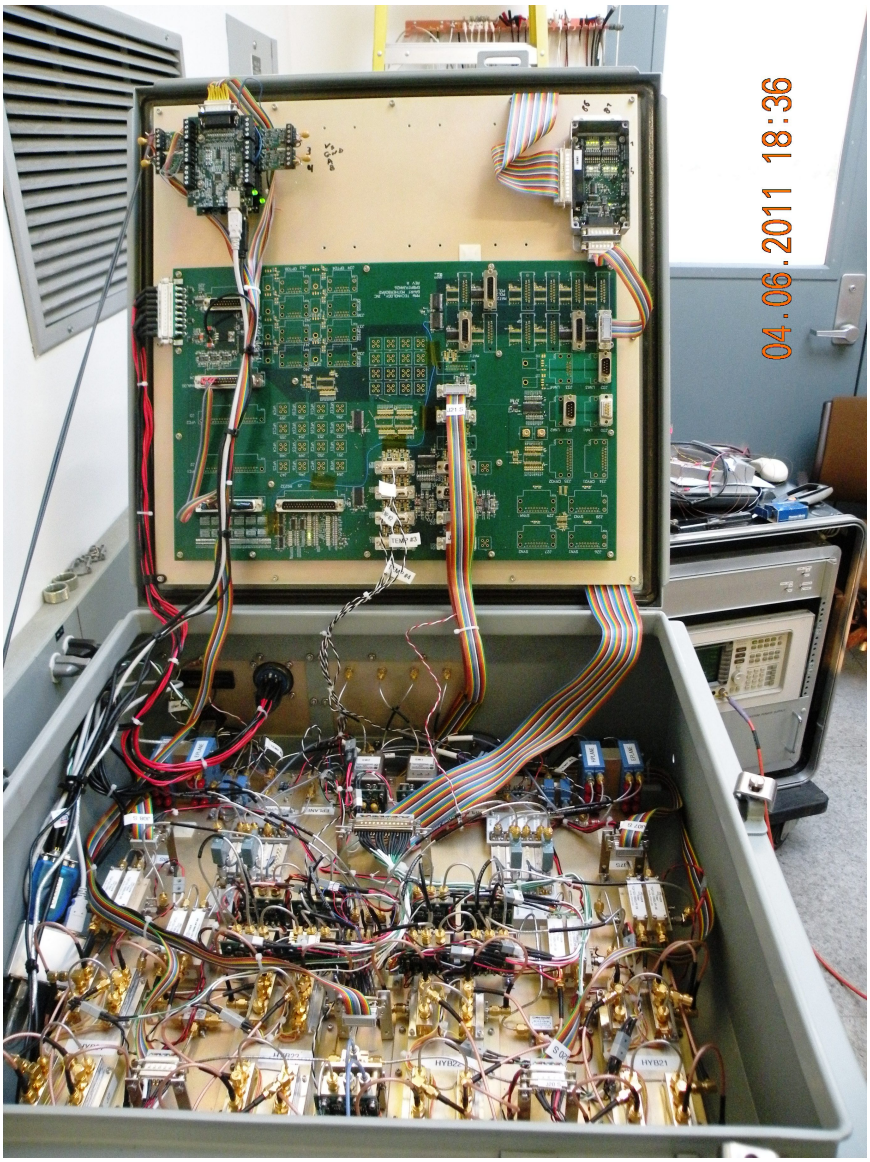


Figure B.1: Interior of WBDC1.

is intentionally inverted so that the default state is L/U, that is, the signals go through the hybrids.

These outputs are routed to an IF distribution panel shown in Figure B.5. This panel uses

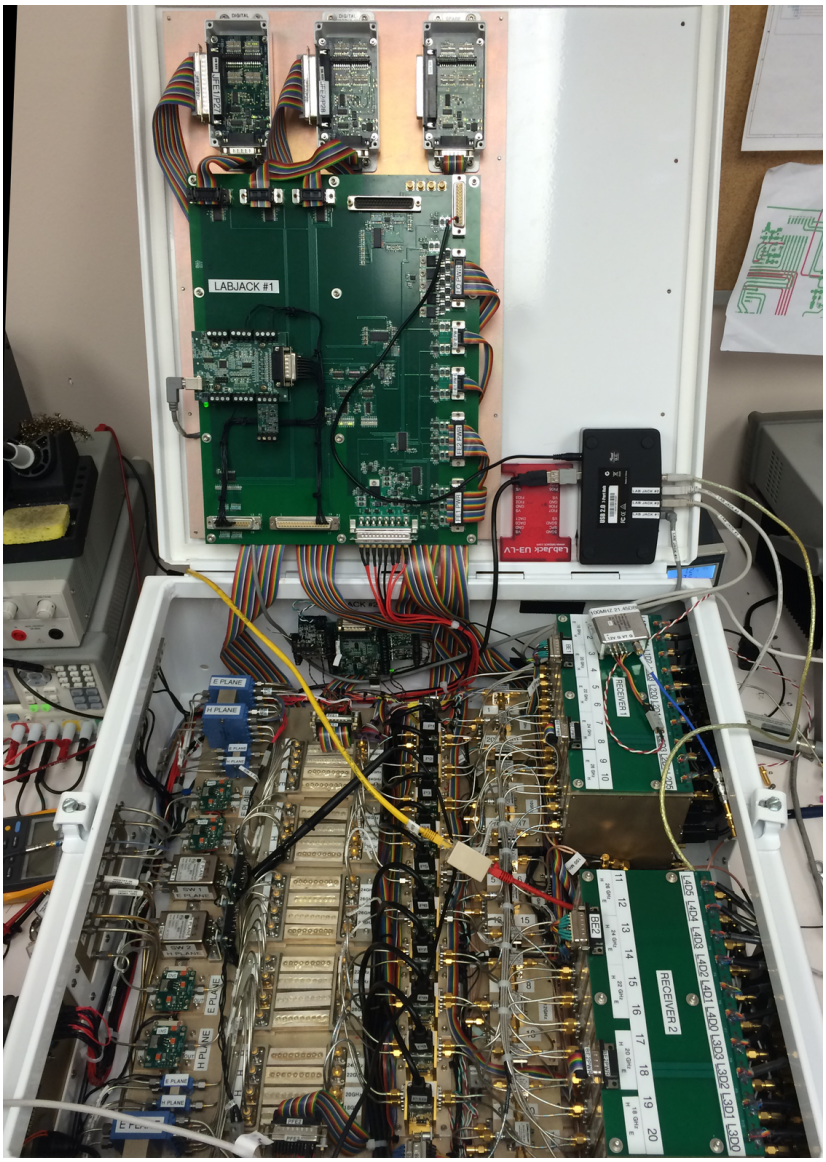


Figure B.2: Interior of WBDC2.

the P1/2 and IF1/2 notation.

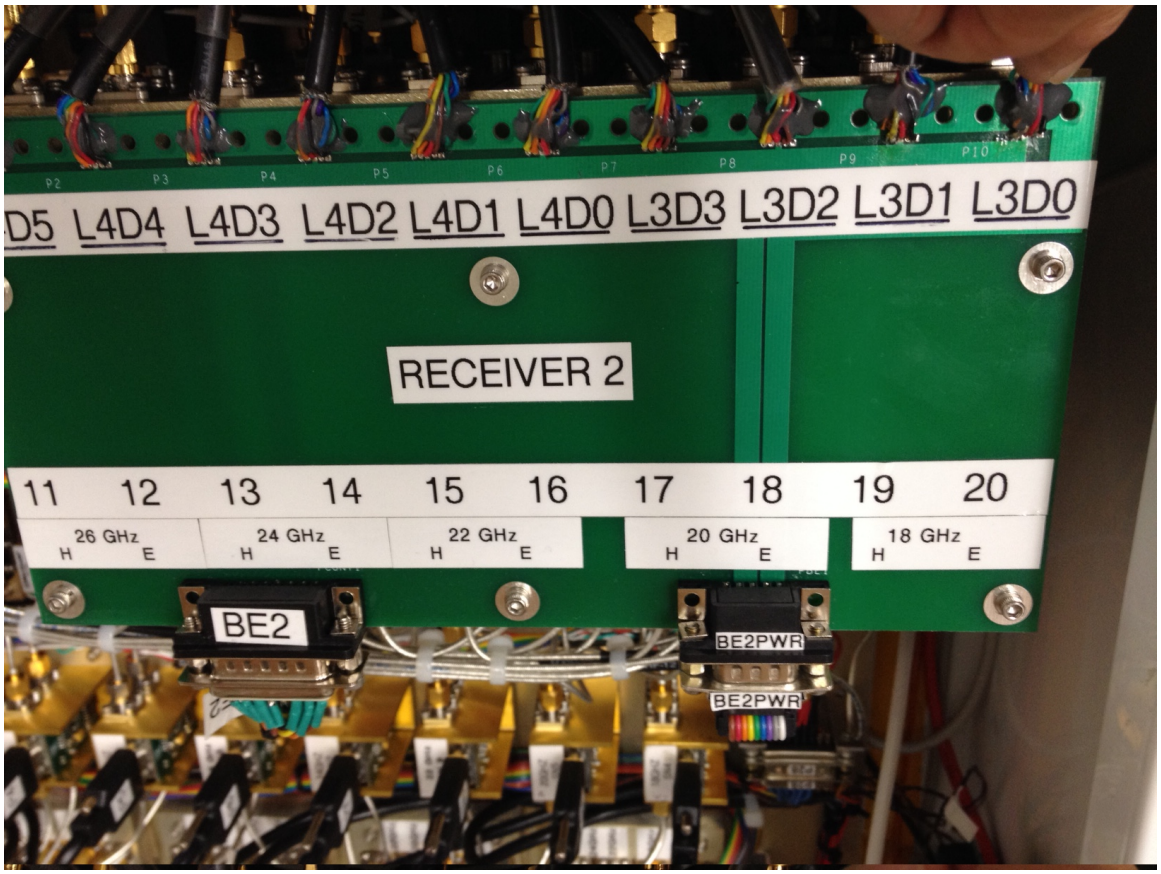


Figure B.3: Down-converters in WBDC2. The labels L$x$D$y$ refer to LG$\mathtt{x}$ pin A$y$ in DM 2 as discussed in section 3.1.2.(Photo by L. Teitelbaum)

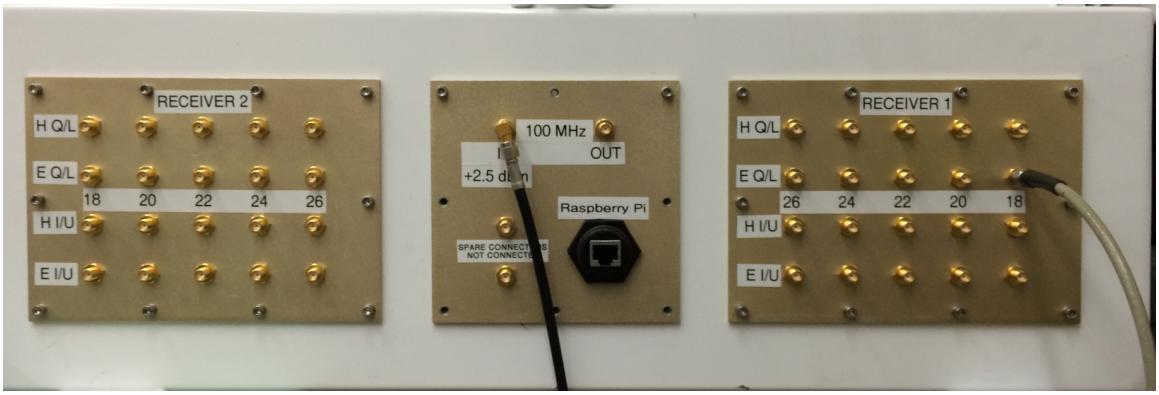


Figure B.4: Outputs of WBDC2.


Figure B.5: WBDC2 IF patch panel. 16 of the 40 outputs can be routed via optic fibers to the control room.

# Appendix C

# LabJack Configuration

## C.1 LabJack 1

This monitors and controls the motherboard.

```
FIO0-FIO3 - AIN: monitor voltages, current, temperatures
FIO4-FIO6 -      not used
FIO7      - DO:  read digital in
EIO0-EIO7 - DO:  select latch by address
CIO0-CIO3 - DO:  set WBDC signals
```

So this is the LabJack Configuration

```
CIOBitDir     1111
EIOBitDir 11111111
FIOBitDir 00000000
FIOAnalog 00001111
```

## C.2 LabJack 2 (and 3)

This controls the LJTickDAC® voltage sources. All EIO and CIO ports are assigned to controlling this function and so are digital outputs.

## C.3 Configuration Check

An initial checkout might be as shown in Snippet C.1. The normal state of the WBDC1 LabJacks is something like

```
In [2]: from MonitorControl.Electronic.Interfaces.LabJack import *
In [3]: lj = connect_to_U3s()
In [4]: report_IO_config(lj)
```

**Snippet C.1:** Commands for checking the LabJack I/O configuration.

```
=============== U3 I/O Configurations ================
          U3 local ID          1        2
          ------------- -------- --------
              CIOBitDir 00001111 00000000
               CIOState 00001111 00001111
             DAC1Enable 00000000 00000000
              EIOAnalog 00000000 00000000
               EIOState 01010111 11111111
         EnableCounter0    0.000    0.000
         EnableCounter1    0.000    0.000
                  FAIN0    0.001
                  FAIN1    1.455
                  FAIN2    0.000
                  FAIN3    0.000
              FIOAnalog 00001111 00000000
              FIOBitDir 00000000 00000000
               FIOState 11110000 11111111
  NumberOfTimersEnabled        0        0
          Temperature  295.569  295.507
     TimerCounterConfig       64       64
  TimerCounterPinOffset        4        4
```

# Appendix D

# Polarization Conversion

## D.1 IAU Definition of Circular Polarization

As reported in the Proceedings of the Fifteenth General Assembly [IAU(1974)], the following resolution was adopted by IAU Commissions 25 (Stellar Photometry and Polarimetry) and 40 (Radio Astronomy):

*RESOLVED, that the frame of reference for the Stokes parameters is that of Right Ascension and Declination with the position angle of the electric-vector maximum, $\theta$, starting from North and increasing through East. Elliptical polarization is defined in conformity with the definitions of the Institute of Electrical and Electronics Engineers (IEEE Standard 211)*[IEEE(1969)].

**Left-Handed (Counterclockwise) Polarized Wave** (LCP): An elliptically polarized electromagnetic wave in which the rotation of the electric field vector with time is counterclockwise for a stationary observer looking in the direction of the wave normal.

**Right-Handed (Clockwise) Polarized Wave** (RCP): An elliptically polarized electromagnetic wave in which the rotation of the electric field vector with time is clockwise for a stationary observer looking in the direction of the wave normal.

This means that the polarization of incoming radiation for which the position angle of the electric vector measured at a fixed point in space increases with time is described as right-handed and positive. This can be seen in the lower panel of Figure D.1[1]

[1]To view these images relax the eye muscles as if looking at a distant object. A person not experienced in this type of viewing might first look at a distant object and then slide the gaze down to the figure without refocusing. Another method is to place a sheet of cardboard with one edge down the middle of the figure and the other edge between the eyes. It is probably easier to focus first on a label above or below the spiral, and then slide the gaze downwards.

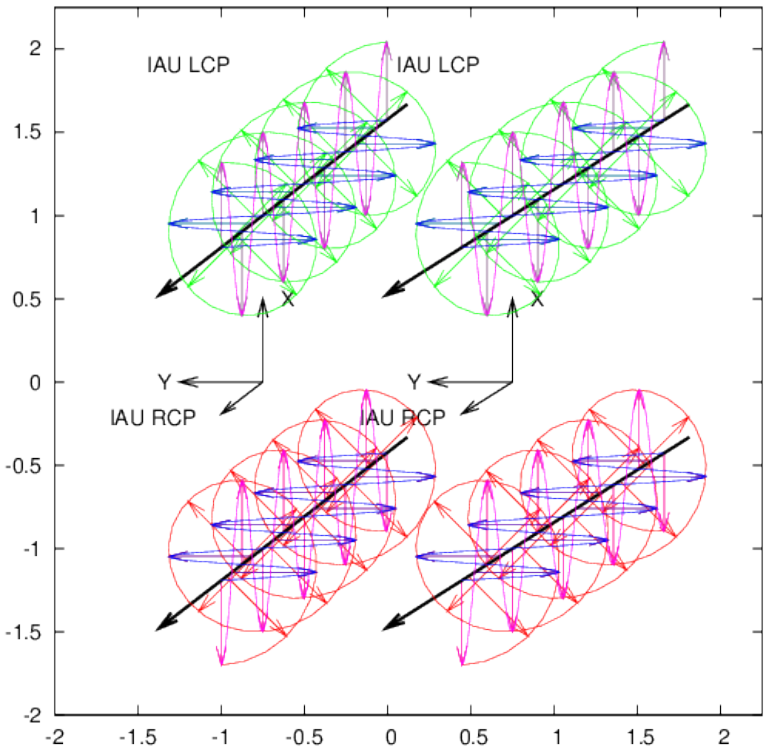


Figure D.1: A right circularly polarized wave (LCP - green) and left circularly polarized wave (RCP - red) as defined by the IAU are shown decomposed into two orthogonal linearly polarized waves (X - magenta, Y - blue). The thick arrows show the direction of propagation.

## D.2 Changing Between Linear and Circular Polarization

Figure D.1 shows that linear polarization can be converted to circular polarization. If the vertical (H-plane or Y-axis) wave lags behind the horizontal (E-plane or X-axis) wave by 90° of phase delay the resulting polarization will be RCP. If the vertical (H-plane or Y-axis) wave leads the horizontal (E-plane or X-axis) wave by 90° of phase delay the resulting polarization will be LCP.

Optically, this can be accomplished by passing the beam through a particular thickness of material which has different indices of refraction for two orthogonally polarized components. Similarly, a waveguide section can be produced to perform the same function.

In the cm, mm, and submm wave domains, there exist devices which can separate a radio wave into two orthogonally polarized components. A quadrature hybrid can convert between the two modes, as shown in Figure D.2. If port A were receiving vertical polarization (H-plane

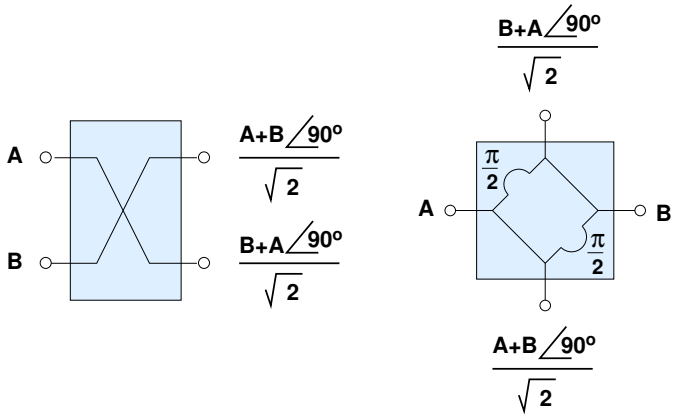


Figure D.2: A quadrature hybrid combines two input signals so that one is delayed 90° with respect to the other. There are two outputs which are symmetrically delayed. The figure on the left shows a conventional schematic. The one on the right is a more illustrative schematic.

or Y-axis) and B horizontal polarization (E-plane or X-axis) then at the port labelled A+∠90°B, the vertical polarization lags behind the horizontal and it would put out RCP. The port labelled B+∠90°A would put out LCP.

From this one can see that, if there is an option to bypass the hybrid, it depends which way the hybrid is put in the circuit which determines whether the output ports are E/L and H/R or are E/R and H/L.

# Appendix E

# Sideband Separation

If the LO is in the middle of the RF band, then the mixing products will have both positive frequencies from the upper sideband and negative frequencies from the lower sideband. Both are present in the IF. A complex mixer puts out two IFs which are separated 90° in phase. These two IFs can be considered the real and imaginary parts of a complex signal. The upper and lower sidebands can be recovered from a complex signal.

The outputs from a complex mixer can be phased together to separate the upper and lower sidebands. This is done with a 90° hybrid, shown in Figure D.2, which re-combines the two quadrature phase mixing products. To see this, consider the outputs of the two mixers to be the sum of the USB and the LSB part of the down-converted spectrum with the frequencies explicitly positive and negative, *i.e.*,

$$V_R = \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\cos(-\omega_L t + \Delta\phi_L), \quad \text{(E.1)}$$

$$= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \Delta\phi_L), \quad \text{(E.2)}$$

$$V_I = \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \Delta\phi_U) + \frac{V_L}{\sqrt{2}}\sin(-\omega_L t + \Delta\phi_L), \quad \text{(E.3)}$$

$$= \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \Delta\phi_U) - \frac{V_L}{\sqrt{2}}\sin(\omega_L t - \Delta\phi_L) \quad \text{(E.4)}$$

where $U$ refers to the upper sideband and $L$ the lower sideband. Then if these signals are combined in a quadrature hybrid, the effect along the paths with the $\pi/2$ delay is that the delayed signal at the summation point is that which entered the delay section $t - \pi/2\omega$ earlier. This means that we subtract $\pi/2$ from the angular terms in Equations E.1 and E.3 as illustrated in Figure E.1.

Subtracting $\pi/2$ from the angular terms is equivalent to subtracting $\pi/2$ from the phases associated with the positive frequency terms and adding $\pi/2$ to the phases associated with the

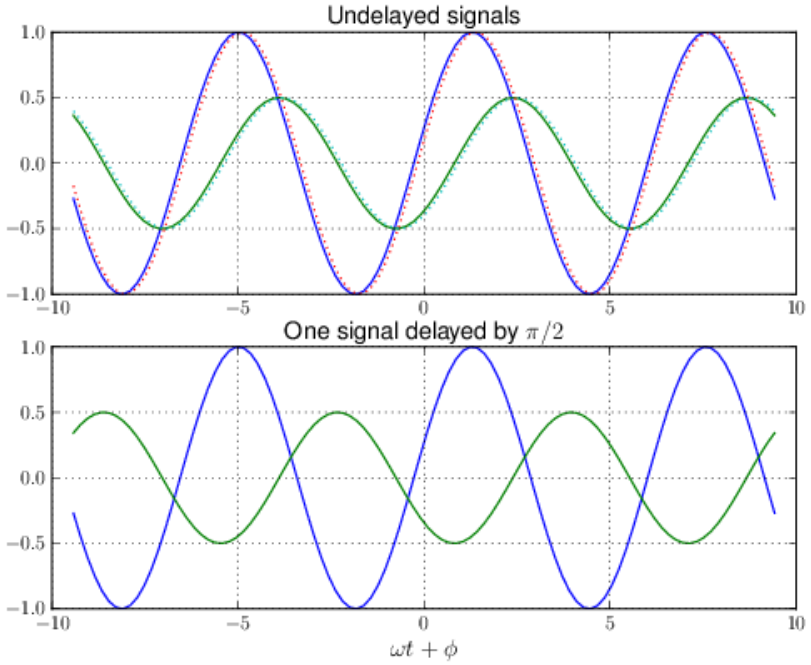


Figure E.1: The angular term for the green line in the lower panel is $\pi/2$ smaller than in the upper panel, showing that the effect is to delay the signal. (The dotted lines show a slighly earlier time for a positive frequency signal; so the signal moves left with time.)

negative frequency terms in Equations E.2 and E.4. The outputs from the hybrid will then be

$$\begin{aligned}
\frac{V_R}{\sqrt{2}} + \frac{V_I \angle 90^\circ}{\sqrt{2}} &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&\quad + \frac{V_U}{\sqrt{2}}\sin(\omega_U t + \phi_U - \frac{\pi}{2}) - \frac{V_L}{\sqrt{2}}\sin(\omega_L t - \phi_L - \frac{\pi}{2}) \\
&= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&\quad - \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L) \\
&= \sqrt{2}V_L\cos(\omega_L t + \phi_L) \qquad \text{(E.5)} \\
\frac{V_R \angle 90^\circ}{\sqrt{2}} + \frac{V_I}{\sqrt{2}} &= \frac{V_U}{\sqrt{2}}\cos(\omega_U t + \phi_U - \frac{\pi}{2}) + \frac{V_L}{\sqrt{2}}\cos(\omega_L t - \phi_L - \frac{\pi}{2})
\end{aligned}$$

$$
\begin{aligned}
& + \frac{V_U}{\sqrt{2}} \sin(\omega_U t + \phi_U) - \frac{V_L}{\sqrt{2}} \sin(\omega_L t - \phi_L) \\
= \quad & \frac{V_U}{\sqrt{2}} \sin(\omega_U t + \phi_U) + \frac{V_L}{\sqrt{2}} \sin(\omega_U t - \phi_L) \\
& + \frac{V_U}{\sqrt{2}} \sin(\omega_U t + \phi_U) - \frac{V_L}{\sqrt{2}} \sin(\omega_U t - \phi_L) \\
= \quad & \sqrt{2} V_U \sin(\omega_U t + \phi_U) \qquad \text{(E.6)}
\end{aligned}
$$

Note that adding $\pi/2$ to the image band terms in Equations E.1 and E.3 is like subtracting $\pi/2$ from the image band terms in Equations E.2 and E.4, *i.e.* the sign of the $\pi/2$ added is the same as the sign of the frequency.

The time-averaged power in each band is

$$
\begin{aligned}
P_{USB} &= V_{USB}^2 \\
&= 2V_U^2 \left\langle \cos^2(\omega_{IF} t + \phi_U) \right\rangle \\
&= V_U^2 \qquad \text{(E.7)} \\
P_{LSB} &= V_L^2 \qquad \text{(E.8)}
\end{aligned}
$$

Equations E.7 and E.8, when added together, give the power in both sidebands.

# Bibliography

[IAU(1974)] IAU. 1974, in Proc. IAU General Assembly 15, IAU (Dordrecht: Reidel), 166

[IEEE(1969)] IEEE. 1969, IEEE Transactions on Antennas and Propagation, AP-17, 270

# Index