R2DAQ
ROACH2 Digital Acquisition System
for Project 8 Phase II

André Young
andre.young@cfa.harvard.edu

June 30, 2016

About this document

This document describes the digital acquisition system developed for Project 8 Phase II on the ROACH2 platform.

Document history

Version	Date	Authors	Summary of changes
1.00	June 30, 2016	AY	Initial version

Contents

List of Figures

List of Tables

1 Hardware

General information on the ROACH2 platform and related topics is available from the CASPER [1]. The purpose of this section is to specify the exact hardware make-up that comprise the system in-use.

A summary of the build specifications of the ROACH2 appears in Table 1. More detailed information can be found in the assembly notes in Appendix A.

Component	**Comments**
FPGA	Xilinx Virtex6 XC6VSX475T-1FFG1759C
ADC	2× ASIAA 5 GSps ADC [2] DMUX 1:1 version (8-bit resolution)
Network	2× Quad-SFP+ mezzanine [3] 8× 10 Gbps total datarate
Enclosure	1U mechanical enclosure*
Misc	ROACH2 rev.2 JP2 shunted* (autostart on AC power) 6 dB attenuator on each ADC I input* Clock split internally to each ADC*

* See Appendix A.

Table 1: ROACH2 build specifications.

Figure 1 and Figure 2 show the front and rear panels, respectively, on the ROACH2.

Figure 1: ROACH2 front panel.

Figure 2: ROACH2 rear panel.

2 Installation

This section describes the steps required to configure and install a new ROACH2 system on a Linux server (assuming Debian-like distribution). The steps covered include:

1. configuring ROACH2 to boot from network using DHCP and NFS,
2. configuring the Linux server to provide the necessary services,
3. install software needed to interface with and program the ROACH2, and
4. cable setup of the ROACH2.

2.1 ROACH2 configuration

This section is a customized summary[1] of the notes in [4].

1. Connect the ROACH2 to AC and power it up (if this does not happen automatically).
2. Connect a USB type-A to type-B cable between the ROACH2 (the port labeled "FTDI USB") and the Linux server. The USB interface will show up as four serial devices under `/dev/ttyUSB*`, the third one is usually the right one to connect to (typically `dev/ttyUSB2`). You can also use `lsusb` to help identify the correct USB device; look for **Future Technology Devices International, Ltd FT4232H Quad HS USB-UART/FIFO IC**.
3. As root, do `screen /dev/ttyUSB2 115200` (or whatever you have identified as the correct USB device) which will connect to the device terminal. At this point you want to do a soft reset of the ROACH2, and how to do that depends on how it was last configured.
   (a) If you are faced with a login prompt it likely means that the system has booted from memory. Log in as `root` (blank password) and do `shutdown -r now` to restart it. When the ROACH2 restarts it will give a countdown (usually 5 seconds) during which it allows you to interrupt the boot process. Do this by pressing `ENTER` .

[1] Credit to Laura Vertatschitsch for creating a first version of this set of instructions.

(b) If you see error messages along the line of the system being unable to boot, it was likely set to boot from network. Interrupt the process by doing `Ctrl+c` .

4. You should now arrive at a prompt, indicated by `=>` where you can verify / change the configuration of the ROACH2.

5. Do `printenv ethaddr` to display the MAC address of the 1 Gb (control network) hardware. You will need this to set up DHCP on the server.

6. You can also do `printenv` to show all environment variables. Verify that the variable `bootcmd` is set equal to `run netboot`. If it is not, do `setenv bootcmd run netboot` and then `saveenv` to set the ROACH2 to boot from the network. You can verify that the variable has been updated by doing another `printenv` .

7. Finally, do `reset` which will restart the ROACH2, and if the configuration was done properly you will see it trying to boot from the network. It will of course fail if the server has not been configured yet to serve what the ROACH2 needs.

2.2 Server configuration

The server provides to the ROACH2 its IP settings, a boot image via TFTP, and a filesystem for the PowerPC via NFS. It is assumed that all required system packages (e.g. `nfs-kernel-server`, `nfs-common`, etc., but which may differ among distributions) are installed. To set up DHCP we will use `dnsmasq` but any other service that offers what is needed will do.

1. Identify a network interface on the server through which it can offer DHCP to the ROACH2. Set its IP settings to have a static configuration. Bring down and back up the interface if necessary. For illustration we will assume that interface `eth1` has been configured with address `192.168.1.1` .

2. Create the path `/srv/roach2_boot` .

3. Clone `https://github.com/sma-wideband/r2dbe.git` into a temporary path. We only need the contents in the `/support` subdirectory here.

4. Do an attribute-preserving copy of the boot image doing something like `cp -rfp support/boot /srv/roach2_boot` .

5. Do an attribute-preserving copy of the tarballed filesystem by doing something like `cp -fp support/r2dbe-debian-fs.tar.gz /srv/roach2_boot` . In `/srv/roach2_boot` do `tar xvfz r2dbe-debian-fs.tar.gz` which will create a directory called `debian_stable_devel` . Rename it to something like `mv debian_stable_devel root` .

6. Now we are done with the cloned git repository, delete it if you want to.

7. Edit `/etc/exports` and add the share
`/srv/roach2_boot 192.168.1.0/24(rw,subtree_check,no_root_squash,insecure)`

8. Effect the changes by doing `exportfs -r` and verify that it is successful by doing `showmount -e` .

9. Edit `/etc/dnsmasq.conf` to include the following lines:
```
# serve an IP address, substitute the ROACH2's MAC address below
dhcp-host=02:44:01:02:0a:1b,192.168.1.2
# serve a boot image over TFTP
dhcp-boot=uImage
enable-tftp
tftp-root=/srv/roach2_boot/boot
# serve root filesystem over NFS
dhcp-option=17,192.168.1.1:/srv/roach2_boot/root
# optional extras
dhcp-authoritative
interface=eth1
dhcp-range=192.168.1.128,192.168.1.255,12h
```

10. Add the ROACH2 hostname to `/etc/hosts` by inserting the line
```
# Let us call the ROACH2 by the name 'led'
192.168.1.2 led
```

11. Restart DHCP service by `/etc/init.d/dnsmasq restart` .

12. Connect the ROACH2 to the network on which DHCP is served through the "PPC NET" port, and restart the ROACH2.

At this point the ROACH2 should try to boot over the network using DHCP served to it by the server. Verify that this is working by monitoring it through the serial interface: the output to expect will show the ROACH2 obtaining an IP address, successfully booting, and eventually displaying a login prompt. You can also monitor what the server is doing my monitoring various logs, for example, `tail -f /var/log/syslog` during the restart.

2.3 Software installation

The ROACH2 should now be set up so that you can interact with it from the server, or any other machine that can see it on the network. You can log in to it via `ssh` as `root` and do any number of things that you might do on a Unix-like system. It should be running a service called `tcpborphserver3` , if it is not running you can start it by doing `/etc/init.d/tcpborphserver3` . (This is useful to keep in mind for the future and is one of the things you can check if you have problems programming the ROACH2, which does happen, albeit very rarely.)

A few custom python libraries are needed to program the ROACH2 and to interact with it in a more targeted application. Some standard dependencies are needed to install these libraries and will be assumed to be obtainable. On the server:

1. Install the KATCP protocol `pip install katcp` . This is used to program the ROACH2 and to interact with the gateware.

2. Also do `pip install corr` which provides wrappers to the functionality available in KATCP.

3. Install the python library required to configure the ADC card installed on the ROACH2. Clone it from github using
`git clone https://github.com/sma-wideband/adc_tests.git`
and in `adc5g_tests` do `python setup.py install` .

4. Clone the `phasmid` repository using
`git clone https://github.com/project8/phasmid.git` .

5. Copy any gateware binaries that you intend to program from `phasmid/casper/master/r2daq/bit_files` to the location in the mounted filesystem `/srv/roach2_boot/root/boffiles` where the ROACH2 will be able to see it. The ROACH2 can only be programmed with `.bof` files so if you copied a zipped format you will need to unzip `gunzip bitcode.bof.gz` before attempting to program the ROACH2.

The software setup is now complete.

2.4 Signals and cabling

All that is left to do before you are able to follow the *Getting started* instructions in `phasmid/README.md` is to provide the necessary analog signals and to cable up the data network. The exact configuration depends on the gateware version, however in general the following applies.

1. Supply a -6 dBm 1600 MHz tone to the SMA port labelled `clk` .

2. Provide the analog signal(s) to be digitized to the SMA port(s) labelled `if0` and/or `if1` . Wideband noise at a level of -6 dBm will more or less use the full range of the ADC.

3. Connect the SFP+ cable(s) to the appropriate network port(s) `CH0`–`CH3` in `SLOT0` and/or `SLOT1` .

This concludes installation of the ROACH2 system.

3 Gateware

This section gives an overview of the gateware design. The intention is more to provide the system user with a better understanding of the signal processing and data organization that occurs within the gateware than to detail the logic circuits.

The gateware can be subdivided into different groups: global status and control, analog-to-digital conversion, and digital channel processing. A description of each group is provided in the subsequent sections, followed by a section devoted to discussing version-specific topics.

Monitor and control of the gateware is provided through registers that are built into the gateware and which are accessible from within the PowerPC, and thereby within Python through the use of the installed libraries. In each section devoted to a particular group of gateware logic the relevant input / output registers are listed and described. All registers are 32-bit fields and the interpretation of each are detailed below.

The general description below applie to the build `r2daq_2016_May_18_1148.bof` which is identified as version `0001`. Changes in subsequent versions of the gateware are discussed in the section devoted to that topic.

3.1 Global status and control

This group comprises mostly of time-keeping and reset logic.

A master reset signal is provided which, when triggered restores all subsystems to their initial state.

The time-keeping system counts the number of seconds that have passed since the master reset signal was last released. This value is added to the reference time stored in `unix_time0` to compute the current time. The reference time and current time are both 32-bit integers that measure Unix time, i.e. seconds that have elapsed since 00:00:00 UTC on 1 January 1970.

Input registers The following input registers are associated with this group:

`master_ctrl` Controls the master reset and synchronization signals.

bits	type	comments
0	BOOLEAN	master reset, active high
1	BOOLEAN	manual sync, active high
31..2	n/a	not used

`unix_time0` Defines the reference time.

bits	type	comments
31..0	UINT_32	reference Unix time

Output registers The following output registers are associated with this group:

`master_status` Copies a number of flags output from the FFT cores and packetization blocks. Not recommended for general use.

3.2 Analog-to-digital conversion

The ADC comprizes four cores that each sample the analog input in parallel at one quarter of the sampling rate, i.e. 800 MSps. By interleaving the cores to be 90 degrees out-of-phase with respect to each other, the samples can be combined to produce the sampling at 3200 MSps. The output of the ADC is demultiplex by a factor 16 so that with each cycle the FPGA, which is clocked at 200 MHz is presented with 16 consecutive time-domain samples (4 from each ADC core).

Only a single ADC, the one associated with input `if0` and the `zdok0` interface between the ADC card and the FPGA board is currently available in the bitcode.

Input registers n/a

Output registers n/a

3.3 Digital channel processing

Three digital channels are associated with each ADC to allow for a total of six digital channels to be processed. That is, three digital channels will operate on the same data received from one particular ADC. The digital channels are named `a`–`f`, see Table 2 for associations with each channel. Only channels `a`–`c` and FFT cores `ab`–`cd` are currently implemented. In the following `<x>` and `<xy>` will be used as placeholders for the channel / FFT core designations.

Each digital channel can select a 100 MHz wide band within the 1600 MHz wide signal received from the ADC, and output both a time-domain and frequency-domain sampling of that band. The various steps in processing that are performed to achieve this are described in the following sections.

Digital channel	`a`	`b`	`c`	`d`	`e`	`f`
Implemented	yes	yes	yes	no	no	no
Digital ID	0	1	2	n/a	n/a	n/a
Analog channel	`if0`			`if1`		
IF ID	0			n/a		
FFT core	`ab`		`cd`		n/a	
SFP+ slot	`SLOT0`			n/a		
SFP+ port	`ch0`	`ch1`	`ch2`	n/a	n/a	n/a

Table 2: Digital channel allocation.

3.3.1 First frequency translation

The first frequency translation mixes the 8-bit input data with a complex exponential to position the positive half-spectrum of the 100 MHz wide channel-of-interest to align with an odd-numbered Nyquist zone for sampling at 200 Msps. The desired translation is computed in software (see Section 4), but is selected to be greater than 100 MHz in magnitude to ensure that the negative half-spectrum of the channel is well outside the same Nyquist zone.

Sine and cosine components of the mixing frequency are generated from a direct digital synthesizer (DDS) implemented as a lookup table with 10-bit phase resolution and 11-bit amplitude resolution. The given phase resolution is equivalent to a frequency resolution of

$$\Delta_f = DF_{\text{clk}}/2^{b_\phi} = 3.125 \text{ MHz}, \tag{1}$$

where D is the demux factor, F_{clk} is the FPGA clock frequency, and b_ϕ is the phase bit-resolution.

The frequency is controlled by the register `ddc_1st_<x>_synth_input_dphi` which essentially indicates the phase step Δ_ϕ between consecutive samples and thus determines the frequency. In order to accommodate high enough frequencies such that the phase may wrap within D or fewer samples, the register also sets the phase step $D\Delta_\phi$ associated with every D samples.

Input registers The following input registers are associated with this group:

`ddc_1st_<x>_synth_input_dphi` Controls the DDS.

bits	type	comments
9..0	UINT_10	common phase offset in generated sinusoid signals
19..10	UINT_10	number of $2\pi/2^{b_\phi}$ steps in phase per sample at 3200 MSps
29..20	UINT_10	number of $2\pi/2^{b_\phi}$ steps in phase per D samples
31..30	n/a	not used

Output registers n/a

3.3.2 Bandpass filter and downsampling

The real and imaginary components of the output of the first frequency translation serve as input to a 127-order finite impulse response (FIR) filter. The filter output is downsampled by a factor 16 to deliver a time-domain digital signal at 200 MSps and a demux factor of 1. The purpose of this filter is to suppress signal and noise outside the band-of-interest to a sufficient extent so as to reduce the impact of aliasing that results from the downsampling. The downsampled filter output is,

$$y[16k\Delta_T] = \sum_{i=0}^{127} b_i x[(16k-i)\Delta_T]. \tag{2}$$

The coefficiens are real-valued and specified as 8-bit fixed-point 2's complement with 7 fractional bits each. Coefficients are ordered into groups of 4 (4×8-bit = 32-bit) so that each group is associated with a single 32-bit register. There are 32 such registers for a single 128-coefficient filter.

The output of the filter is multiplied by a single real-valued gain which is specifed as 8-bit fixed-point 2's complement with 4 fractional bits. The gain values for all channels are specified in the register `gain_ctrl`.

The output of the gain is split to two parallel paths, one to the STFT and the other to a second second frequency translation.

Input registers The following input registers are associated with this group:

`ddc_1st_<x>_cb<m>_g<n>` Group of four filter coefficients $\{b_i \,|\, i = 16m + 4n + j,$ $j = 0, 1, 2, 3\}$, where $m = 0, 1, \ldots, 8$ and $n = 0, 1, 2, 3$.

bits	type	comments
7..0	FIX_8_7	0th coefficient ($j = 0$)
15..8	FIX_8_7	1th coefficient ($j = 1$)
23..16	FIX_8_7	2th coefficient ($j = 2$)
31..24	FIX_8_7	3th coefficient ($j = 3$)

`gain_ctrl` Gain applied to FIR filter outputs.

bits	type	comments
7..0	FIX_8_4	gain for channel `a`
15..8	FIX_8_4	gain for channel `b`
23..16	FIX_8_4	gain for channel `c`
31..24	n/a	not used

Output registers n/a

3.3.3 Short-term Fourier transform

The STFT is computed over 8192 complex-valued time-domain samples to produce 8192 complex-valued spectrum samples. Since the first frequency translation and bandpass filter contained the band-of-interest to only the positive half-spectrum, only the 4096 spectrum samples that correspond to positive frequency are retained.

The STFT is implemented as a 13-stage FFT. There is no bit growth in the computation and a software-controlled shift schedule is used to avoid overflow. The shift schedule is a 13-bit field and the value of each bit determines whether the output of the associated stage should be shifted to the right by one position ('1') or not ('0'). A shift schedule of '1101010101010' ensures that overflow never occurs.

Since there are currently only two FFT cores in the design their shift schedules are controlled by a single register `fft_ctrl`.

Each FFT core has an overflow flag that indicates whether an overflow has occurred during the computation of the current STFT. Whenever this occurs a 10-bit counter associated with each FFT core increments by one. These counts are available in the register `fft_status`.

Input registers The following input registers are associated with this group:

`fft_ctrl` Shift schedules for the two FFT cores.

bits	type	comments
12..0	BITFIELD	shift schedule for FFT core `ab`
25..13	BITFIELD	shift schedule for FFT core `cd`

Output registers The following output registers are associated with this group:

`fft_ctrl` Counts the number of overflows that have occurred per FFT core.

bits	type	comments
9..0	n/a	not used
19..10	UINT_10	overflow count for FFT core `cd`
29..20	UINT_10	overflow count for FFT core `ab`
31..30	n/a	not used

3.3.4 Second frequency translation and downsampling

A second frequency translation is implemented to center the band-of- interest, that appears only in the positive half-spectrum after the first downconversion, around zero. This enables the use of a low-pass filter with real coefficients to further suppress noise and downsample the time-domain signal further by a factor 2.

Since the frequency translation required here is fixed and equal to exactly one quarter of the sampling rate, it is efficiently implemented as a combination of swapping real and imaginary components, and sign inversions.

The low-pass filter used in this stage is a 4th-order elliptic infinite impulse response (IIR) filter with fixed coefficients. It is implemented as two bi-quadratic structures, each using a scattered lookahead architecture with transfer function

$$H(z) = \sum_{i=0}^{8} b_i z^{-i} \Big/ \sum_{i=0}^{8} a_i z^{-i}. \tag{3}$$

The coefficients for the first and second bi-quadratic structures are given in Table 3 and Table 4, respectively.

i	0	1	2	3	4	5	6	7	8
b_i	0.81526	0.30303	0.11508	-0.26068	-1.00000	-0.13922	-0.13475	0.13388	0.40923
a_i	1.00000	0.00000	0.00000	0.00000	-1.16956	0.00000	0.00000	0.00000	0.39893

Table 3: First bi-quadratic structure of elliptic low-pass filter.

i	0	1	2	3	4	5	6	7	8
b_i	0.67529	1.00000	0.51696	-0.19710	-0.12551	-0.02349	-0.01849	0.00488	0.00375
a_i	1.00000	0.00000	0.00000	0.00000	-0.06164	0.00000	0.00000	0.00000	0.00098

Table 4: Second bi-quadratic structure of elliptic low-pass filter.

The output of the filter is downsampled by a factor two, to produce 4096 complex-valued time-domain samples in each window over which the STFT is computed in parallel. That is, the data rate at the downsampled output of the filter matches that at the output of the STFT computation.

Input registers n/a

Output registers n/a

3.3.5 Requantization

The data at the output of each of the STFT and second downconversion is complex-valued, and the real and imaginary components are represented as 25-bit 2's complement fixed point with 16 fractional bits, FIX_25_16. Here the data is requantized down to 8-bit per componenent by reinterpreting the input data as FIX_25_24 and then rounding to FIX_8_7.

Input registers n/a

Output registers n/a

3.3.6 Packetization

This block packages the time-domain and frequency-domain data produced after requantization for transmission over the network. Time-domain data and frequency-domain data are transmitted in packet pairs, where the first packet contains frequency-domain data over one STFT window, and the next packet contains time-domain data over the same STFT window.

The packet structure is defined in Listing 1 and contains a header of 32 bytes and a data payload of 8192 bytes. The interpretation of each header field is provided in the following.

```
#define PAYLOAD_SIZE 8192
typedef struct packet {
    // first 64bit word
    uint32_t unix_time;
    uint32_t pkt_in_batch:20;
    uint32_t digital_id:6;
    uint32_t if_id:6;
    // second 64bit word
    uint32_t user_data_1;
    uint32_t user_data_0;
    // third 64bit word
    uint64_t reserved_0;
    // fourth 64bit word
    uint64_t reserved_1:63;
    uint64_t freq_not_time:1;
    // payload
    int8_t data[PAYLOAD_SIZE];
} packet_t;
```

Listing 1: Packet structure.

`unix_time` Timestamp applied to the packet in Unix time as seconds in 32-bit unsigned integer.

`pkt_in_batch` Counts the number of packet pairs that have been sent within the current 16-second window. During each 16- second window there will be exactly 390626 packet pairs, or 781252 packets in total, sent for each digital channel. Packets within a single pair will have the same value for this counter.

`digital_id` This field indicates the digital channel from which the packet was sent. See Table 2 for the allocation of digital identification numbers.

`if_id` This field indicates the analog channel associated with the data. See Table 2 for the allocation of analog channel identification numbers.

`user_data_0` Contains the 32-bit value currently in the register `pkt_<x>_ud0`.

`user_data_1` Contains the 32-bit value currently in the register `pkt_<x>_ud1`.

`reserved_0` 64-bit word not currently in use.

`reserved_1` 63-bit word not currently in use.

`freq_not_time` Indicates whether the packet contains frequency-domain data (‘1’) or time-domain data (‘0’).

Whether the packet contains time- or frequency-domain data, the packing order into the payload `data` is,

$\mathrm{im}(x_0)$	$\mathrm{re}(x_0)$	$\mathrm{im}(x_1)$	$\mathrm{re}(x_1)$	$\cdots$	$\cdots$	$\mathrm{im}(x_K)$	$\mathrm{re}(x_K)$

with $K = 4095$, and re(x) and im(x) taking the real and imaginary components of complex-valued x, respectively. Each element in the data array may be interpreted as a signed 8-bit integer.

Input registers The following input registers are associated with this group:

`pkt_<x>_ud0` Specifies the 32-bit value to copy in the first user-data header field of packets for this channel.

bits	type	comments
31..0	BITFIELD	value to copy into `user_data_0`

`pkt_<x>_ud1` Specifies the 32-bit value to copy in the second user-data header field of packets for this channel.

bits	type	comments
31..0	BITFIELD	value to copy into `user_data_1`

Output registers n/a

3.3.7 Data transmission

The packetized data is transmitted over high-speed network using UDP as the transport layer.

The transmission core for a particular digital channel is associated with a source IP address and port that are configured in software. In addition the core is assigned a MAC address and provided with an ARP table for all devices on the same /24 network.

The packet destination IP address and port are controlled by the software registers `tengbe_<x>_ip` and `tengbe_<x>_port`, respectively.

Each transmission core is also associated with a control register `tenbge_<x>_ctrl` which provides a method by which the core may be reset. This may be necessary if the transmission buffer of the core overflows, however this is not expected to occur during normal operation.

One status register `tenbge_<x>_status` is also associated with each transmission core.

Input registers The following input registers are associated with this group:

`tengbe_<x>_ctrl` Control register for the transmission core.

bits	type	comments
0	BOOLEAN	core reset, active high
31..1	n/a	not used

`tengbe_<x>_ip` Packet destination IP address.

bits	type	comments
31..0	UINT_32	packet destination IP address

tengbe_<x>_port	Packet destination port.
bits	**type**	**comments**
15..0	UINT_16	packet destination port
31..16	n/a	not used

Output registers n/a

3.4 Versions

This section describes any changes to the gateware that have been included since the initial design described in the previous sections.

The gateware contains three registers that indicate various versioning information. These are `rcs_lib` which is associated with the git version control information of the CASPER library used to build the gateware, `rcs_app` which is is associated with the git version control information of the Simulink model of the gateware design, and `rcs_app` which contains a single version number. The Python interface library provides a method to interpret the values of these registers, especially those relating to git information.

The version number in `rcs_app` is updated with each new build of the gateware, and all versions of the gateware are provided in Table 5. Minor changes to the design increment the version number by one, and with each major change the version number is rounded up to the next thousand.

Version number	Binary file
0001	r2daq_2016_May_18_1148.bof

Table 5: Master version table.

Particular notes on each version are provided in the following sections.

Version 0001

As described in Section 3.

4 Software

An Python library for interfacing with the ROACH2 is available on github at `https://github.com/project8/phasmid.git` under the subdirectory `software/monctrl`.

That library is targeted specifically for the gateware design described in this document. Dependencies and usage information are provided on the repository page, and within the library docstrings.

References

[1] Collaboration for Astronomical Signal Processing and Electronics Research. ROACH-2 Revision 2. `https://casper.berkeley.edu/wiki/ROACH-2_Revision_2`, 2016.

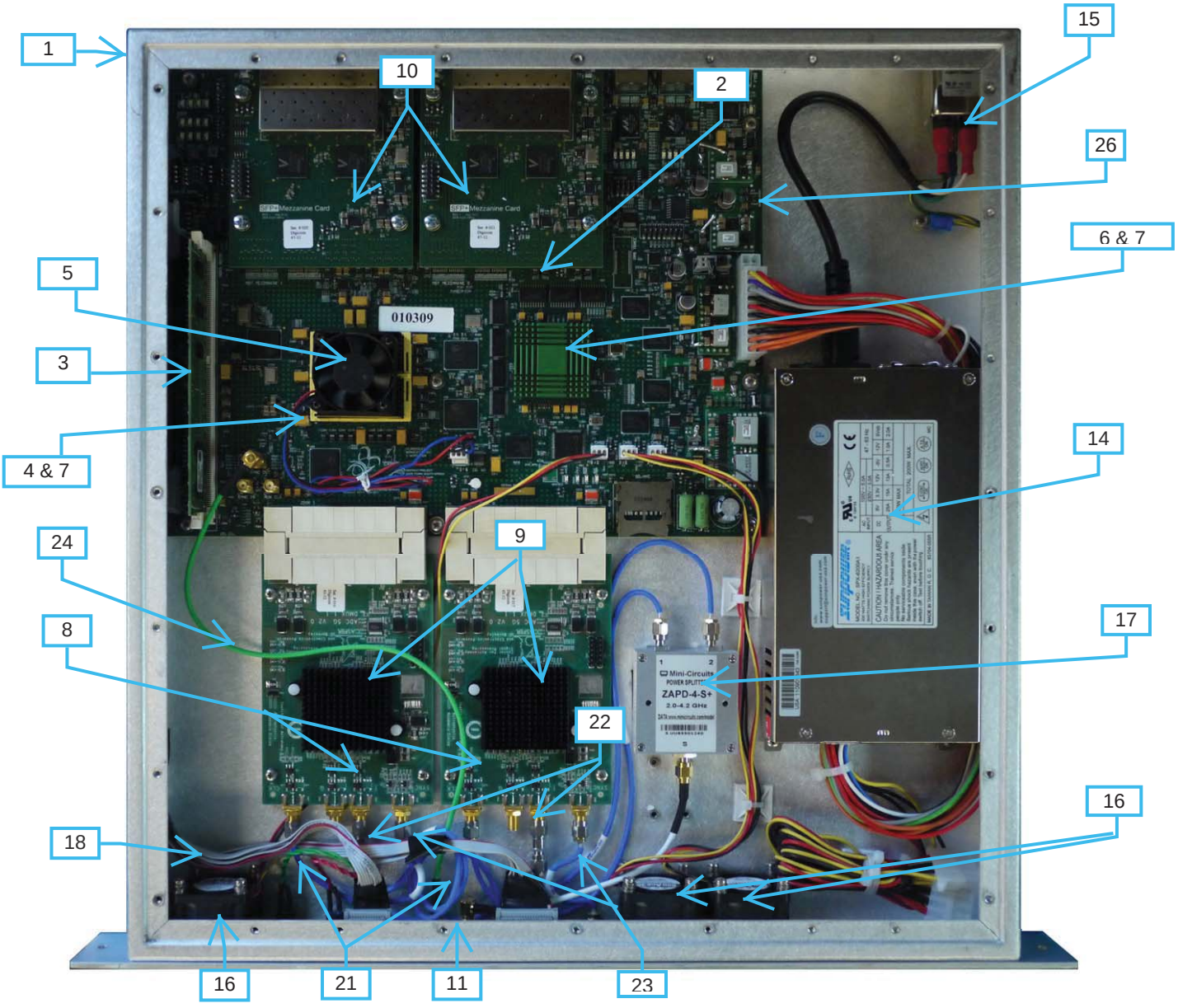
[2] Collaboration for Astronomical Signal Processing and Electronics Research. ADC1x5000-8. https://casper.berkeley.edu/wiki/ADC1x5000-8, 2016.

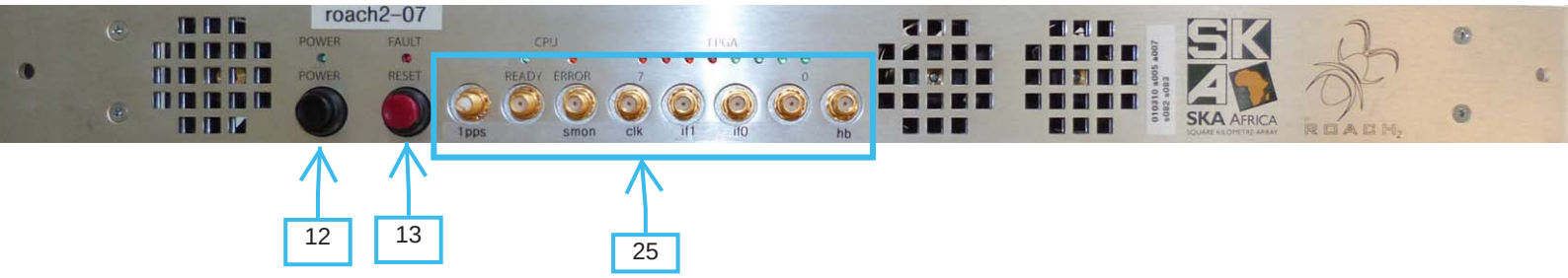
[3] Collaboration for Astronomical Signal Processing and Electronics Research. SFP+. https://casper.berkeley.edu/wiki/SFP%2B, 2016.

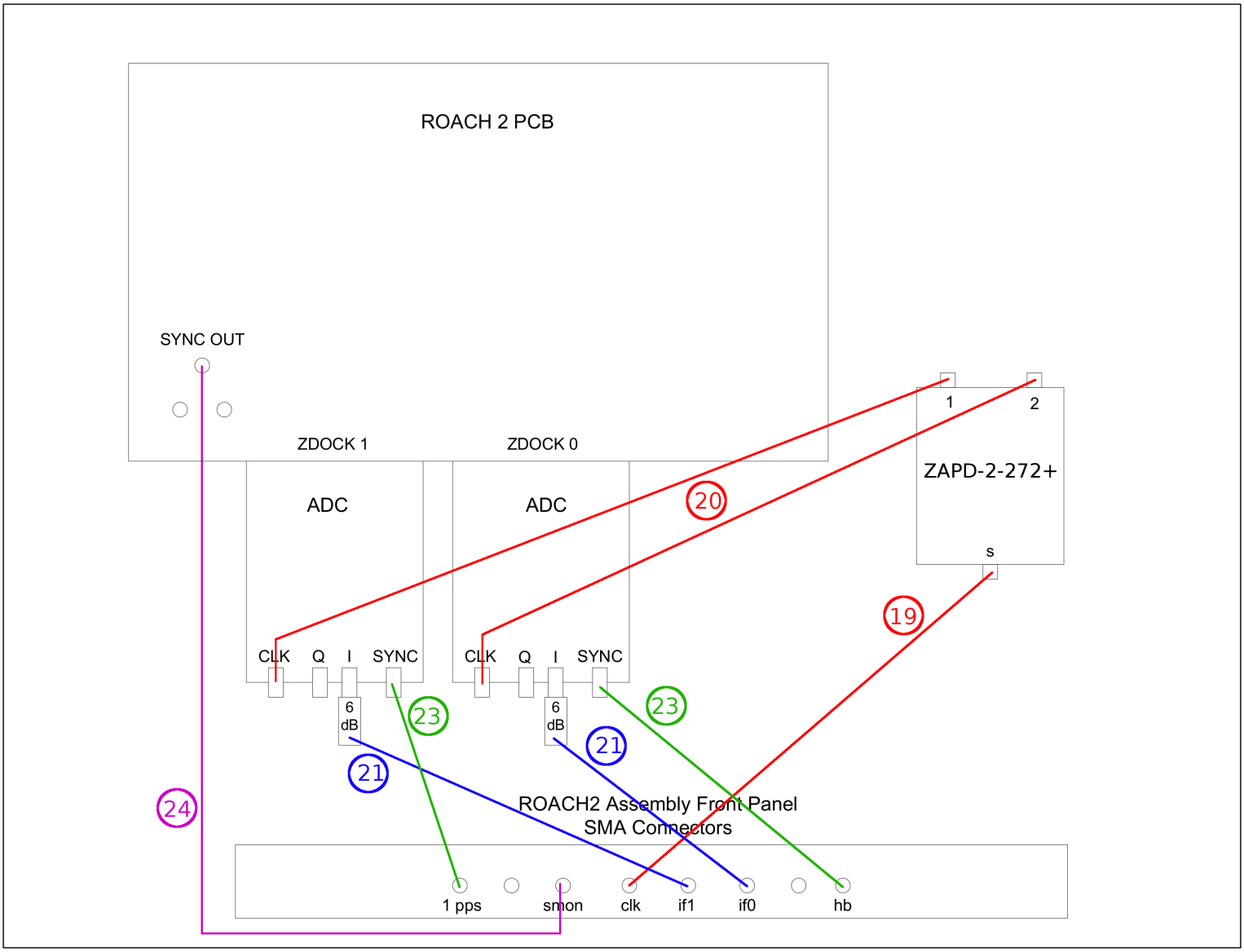
[4] Collaboration for Astronomical Signal Processing and Electronics Research. ROACH NFS guide. https://casper.berkeley.edu/wiki/ROACH_NFS_guide, 2016.

A ROACH2 assembly notes

Assembly notes as on 8 October 2015 compiled by Jonathan Weintroub, John Test, and AY.


1
10
2
15
26
6 & 7
5
3
4 & 7
14
24
9
8
17
22
18
16
16
21
11
23
010309
ZAPD-4-S+


roach2-07
POWER
FAULT
POWER
RESET
CPU
READY
ERROR
FPGA
7
0
1pps
smon
clk
if1
if0
hb
SKA AFRICA
12
13
25

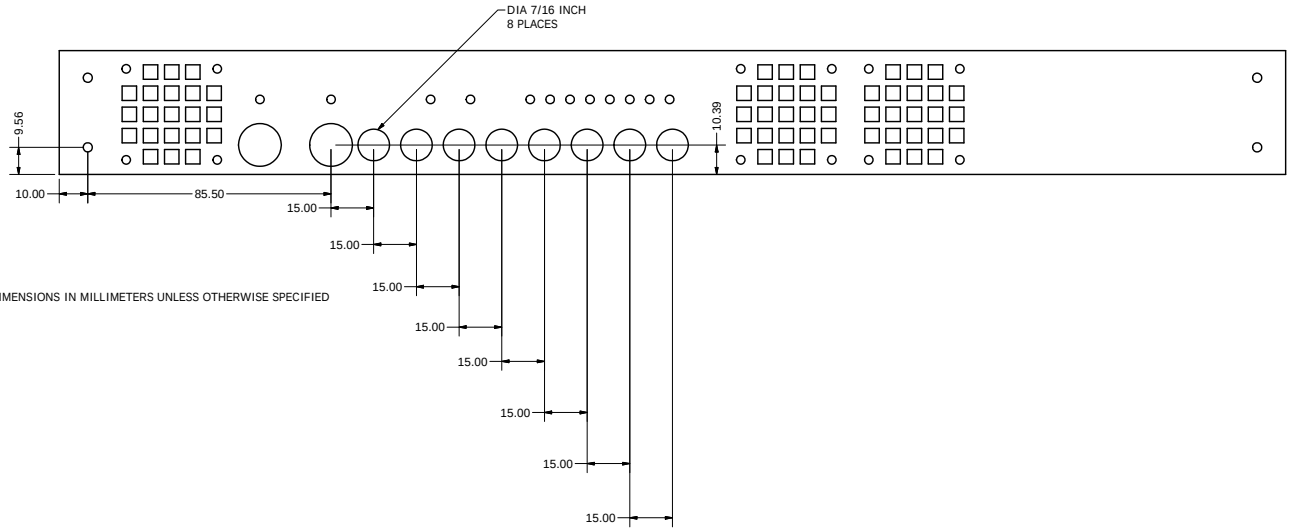

ROACH 2 PCB
SYNC OUT
ZDOCK 1
ZDOCK 0
ADC
ADC
CLK
Q
I
SYNC
CLK
Q
I
SYNC
6 dB
6 dB
1
2
ZAPD-2-272+
s
20
19
23
23
21
21
24
ROACH2 Assembly Front Panel
SMA Connectors
1 pps
smon
clk
if1
if0
hb

Cable	FROM - Front Panel SMA Input	TO - Assembly Connection	Cable length - Connectors
1	1pps	ADC ZDOCK 1 "SYNC"	3"- RA/STR
2	smon	ROACH2 "SYNC OUT"	12"- RA/STR
3	clk	ZAPD-2-272+ Port S	6"- RA/STR
4	if1	*ADC ZDOC 1 "I"	6"- RA/STR
5	if0	*ADC ZDOC 0 "I"	6"- RA/STR
6	hb	ADC ZDOC 0 "SYNC"	3"- RA/STR
Cable	FROM - ZAPD-2-272+	TO - Assembly Connection	Cable length - Connectors
7	Port 1	ADC ZDOCK 1 "CLK"	14" - STR/STR
8	Port 2	ADC ZDOCK 0 "CLK"	14" - STR/STR

*connected to 6dB fixed attenuator on "I" input to ADC

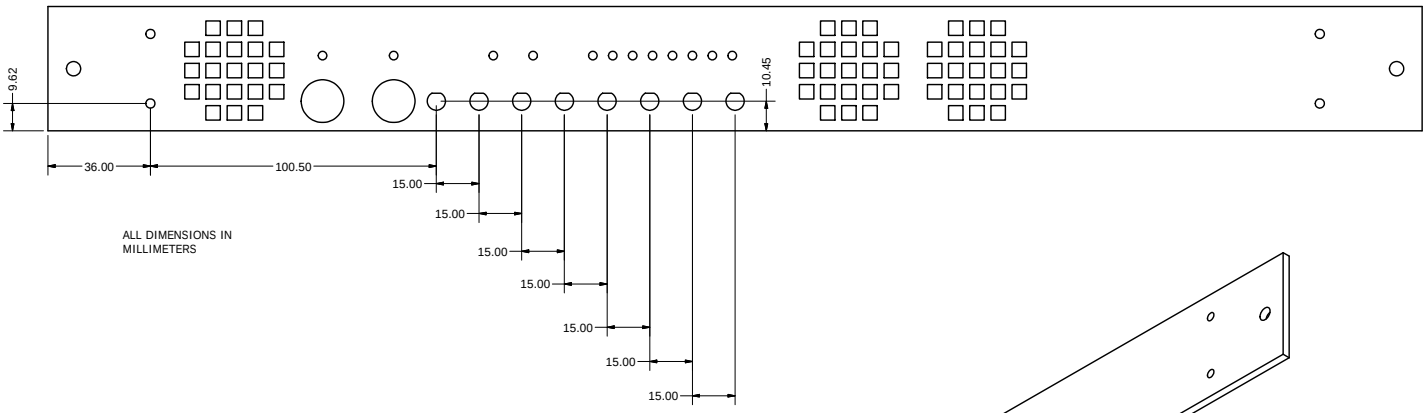
ROACH2 STANDARD SAO Assembly BOM JW, JT, AY 8 Oct 2015 rev 2

Item number	*Quantity*	*Description*	*Vendor*	*Notes*
1	1	1 U enclosure	All Fab	Modded by SAO
2	1	ROACH2 Assy	Digicom	
3	1	DDR3 Dimm		
4	1	FPGA Heat Sink		
5	1	FPGA Fan		Info to be provided
6	1	PowerPC Heat Sink		
7	as req	Thermal grease	Arctic Silver	Arctic Silver
8	2	ADC PCB	Digicom	
9	2	Heat sinks, ADC		
10	2	Quad SFP+ Mez PCB	Digicom	
11	1	Front panel PCB	Digicom	
12	1	power PB switch		
13	1	reset PB switch		
14	1	Power Supply	XEAL TC1UFX2	
15	1	Power Entry Module		
16	3	Fans		
17	1	Power Splitter	MiniCircuits	ZAPD-2-272+ (not the same as shown in photo)
18	1	Ribbon cable assy	Digicom	
19	1	cable, panel_ck-split		see cabling chart
20	2	cable splitter-ADC		see cabling chart
21	2	cable, IF-ADC		see cabling chart
22	2	6dB pads, SMA		inline with IF in
23	2	cables, PPS and HB		see cabling chart
24	1	cable, SMA_out-panel		see cabling chart
25	8	SMA feedthroughs		
26	1	Shunt JP2 "AUTO START"		

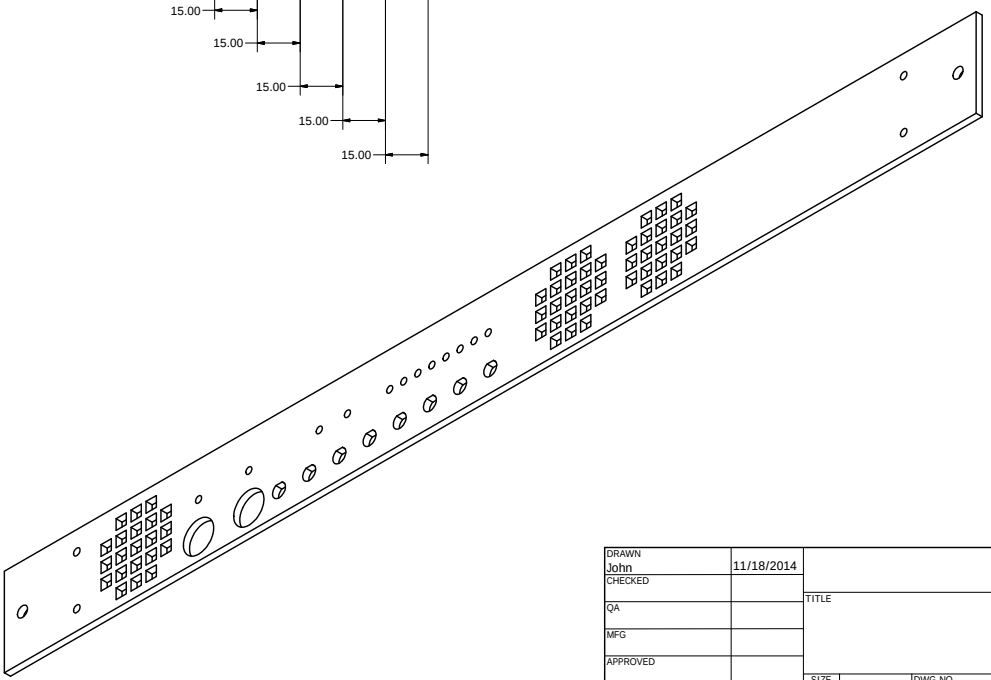


ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED

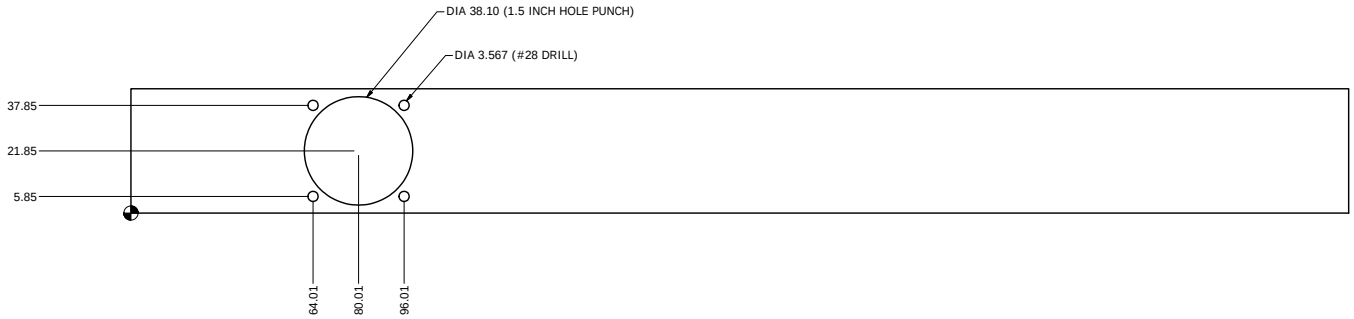
DRAWN John	11/20/2014	
CHECKED		TITLE
QA		
MFG		
APPROVED		SIZE C / DWG NO SMA_Roach2_Front_Chassis / REV
		SCALE / SHEET 1 OF 1



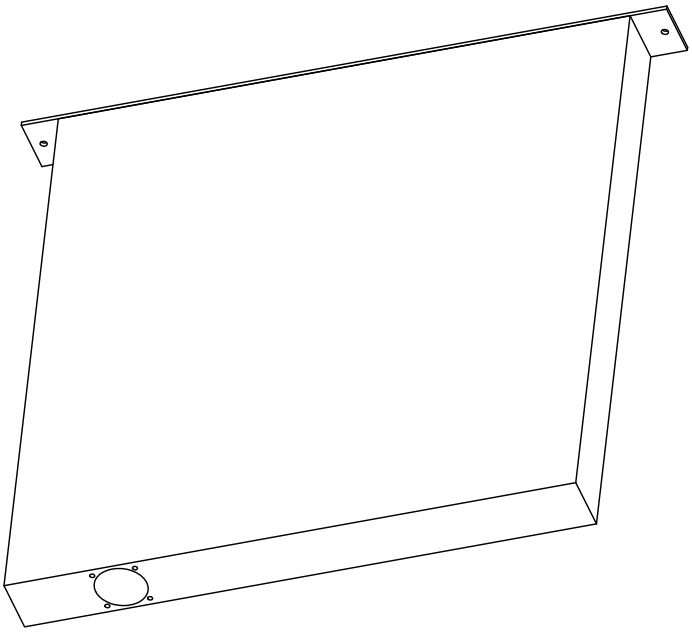
ALL DIMENSIONS IN MILLIMETERS



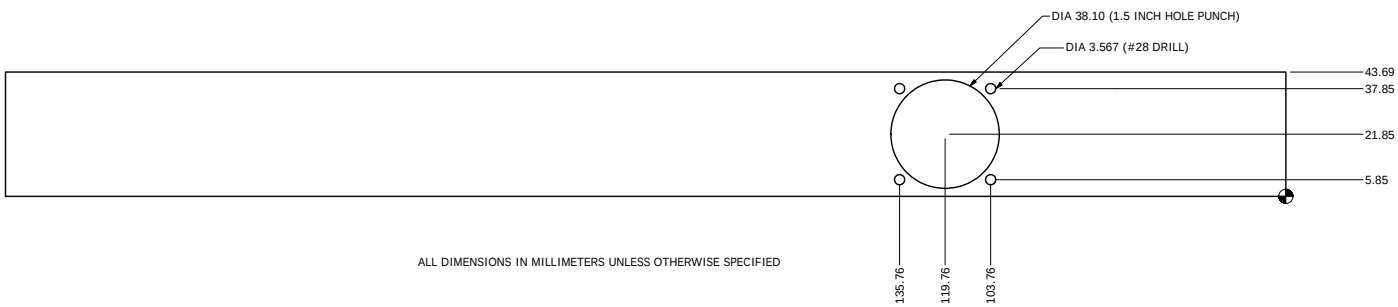
DRAWN John	11/18/2014		
CHECKED		TITLE	
QA			
MFG			
APPROVED		SIZE C	DWG NO SMA_Roach2_Front_Panel REV
		SCALE	SHEET 1 OF 1



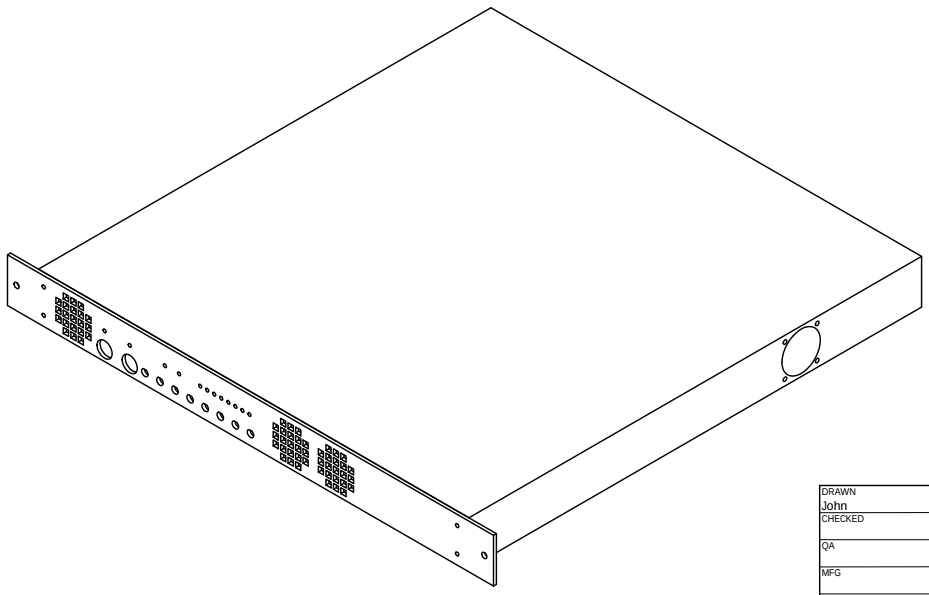
ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED



DRAWN John	11/20/2014		
CHECKED		TITLE	
QA			
MFG			
APPROVED		SIZE C	DWG NO SMA_Roach2_Rear_Chassis_ REV
		SCALE	SHEET 1 OF 1



ALL DIMENSIONS IN MILLIMETERS UNLESS OTHERWISE SPECIFIED



DRAWN John	11/20/2014		
CHECKED		TITLE	
QA			
MFG			
APPROVED		SIZE C	DWG NO SMA_Roach2_Side_Chassis REV
		SCALE	SHEET 1 OF 1