Okika Pi.Ka Board Python Programming Guide

Background

Okika Technologies manufactures field programmable analog array (FPAA) chips. The Pi.Ka board is a Raspberry Pi hat (interface board) that includes 4 OTC24000 FPAA chips. The Pi.Ka may also be shipped with 4 Anadigm AN231E04 chips, which are electrically identical. The compatibility between the Okika OTC24000 and the Anadigm AN231E04 allows the Pi.Ka board to be programmed using .ahf configuration files generated by either Okika DynAMx Design Lab software or Anadigm Designer 2 software.

The Raspberry Pi is a popular single-board computer that runs a version of the Linux operating system and provides access to a wide array of open-source software libraries. This programming guide utilizes Python and open-source libraries to configure the FPAA chips directly from the Raspberry Pi. This guide assumes the reader is familiar with Okika's FPAA chips and the DynAMx Design Lab software and has a working knowledge of the Raspberry Pi.

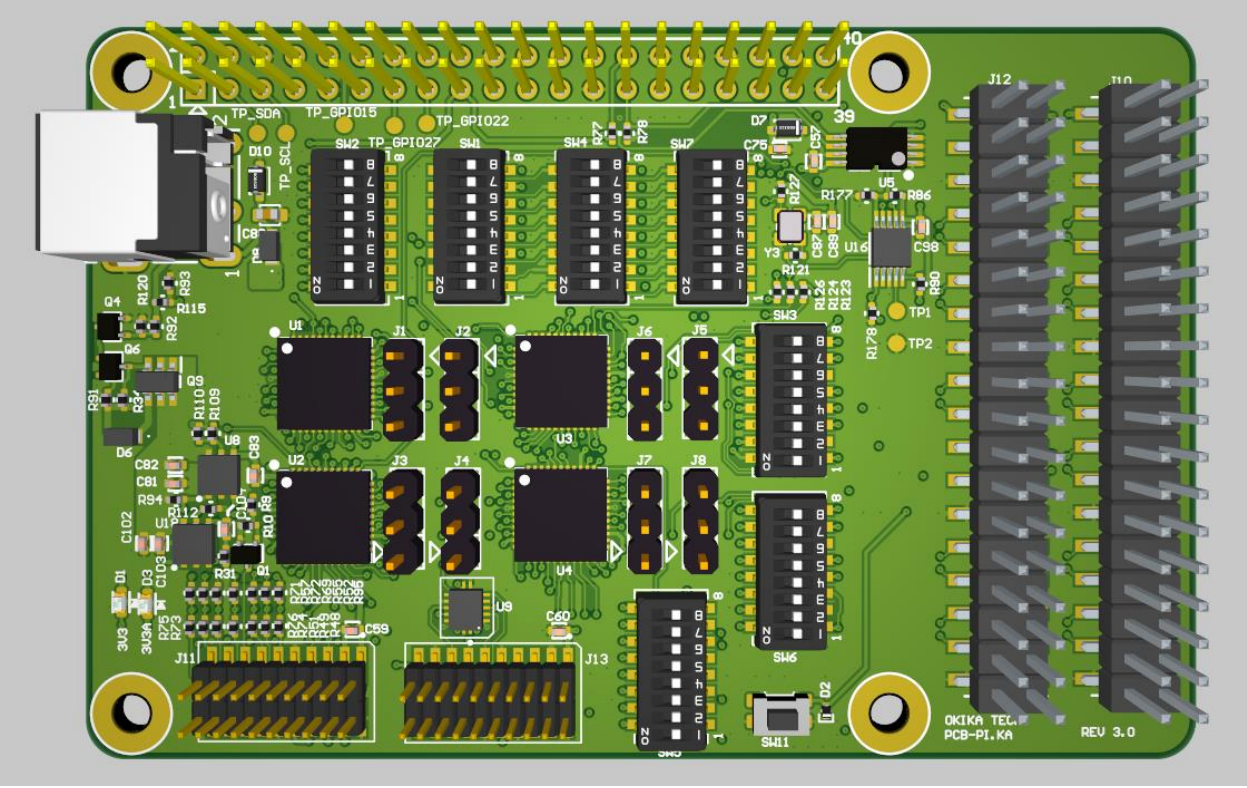
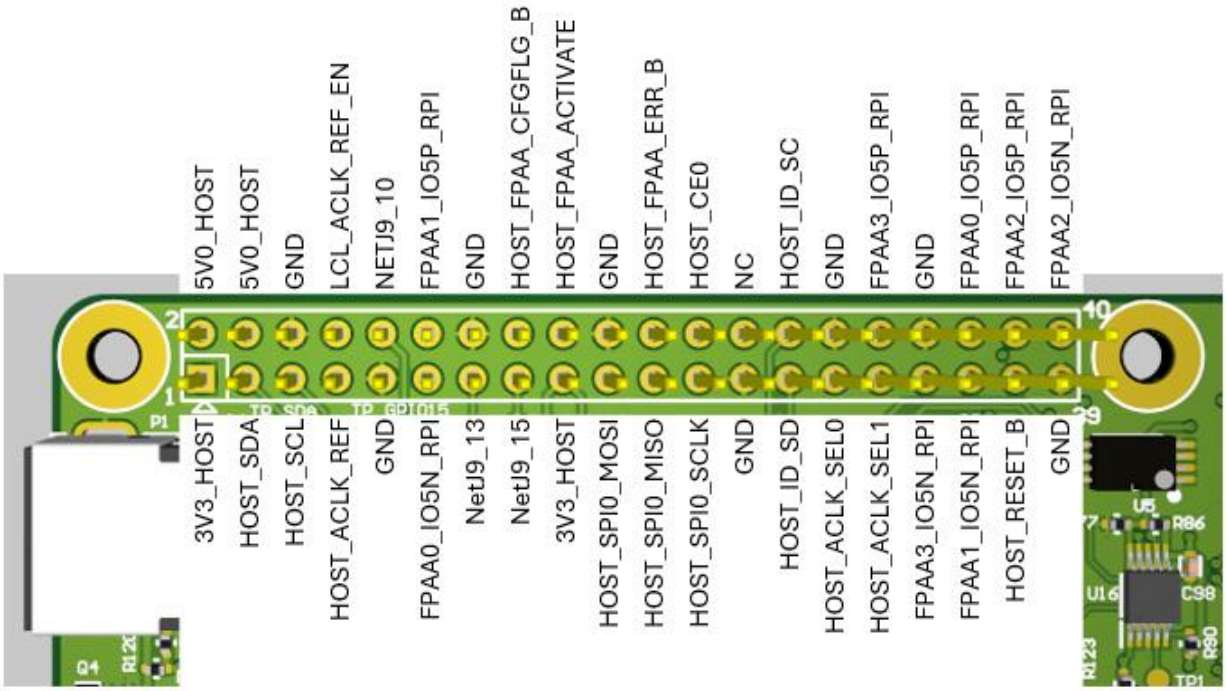


Figure 1 Pi.Ka Board Top View

Raspberry Pi Interface Pinout



Pin	Pi.Ka Name	Raspberry Pi Name
1	3V3_HOST	3.3V Power
3	HOST_SDA	GPIO 2(SDA)
5	HOST_SCL	GPIO 3(SCL)
7	HOST_ACLK_REF	GPIO 4(GPCLK0)
9	GND	Ground
11	FPAA0_IO5N_RPI	GPIO 17
13	NetJ9_13	GPIO 27
15	NetJ9_15	GPIO 22
17	3V3_HOST	3.3V Power
19	HOST_SPI0_MOSI	GPIO 10(MOSI)
21	HOST_SPI0_MISO	GPIO 9(MISO)
23	HOST_SPI0_SCLK	GPIO 11(SCLK)
25	GND	Ground
27	HOST_ID_SD	GPIO 0(ID_SD)
29	HOST_ACLK_SEL0	GPIO 5
31	HOST_ACLK_SEL1	GPIO 6
33	FPAA3_IO5N_RPI	GPIO 13(PWM1)
35	FPAA1_IO5N_RPI	GPIO 19(PCM_FS)
37	HOST_RESET_B	GPIO 26
39	GND	Ground

Pin	Pi.Ka Name	Raspberry Pi Name
2	5V0_HOST	5V Power
4	5V0_HOST	5V Power
6	GND	Ground
8	LCL_ACLK_REF_EN	GPIO 14(TXD)
10	NETJ9_10	GPIO 15(RXD)
12	FPAA1_IO5P_RPI	GPIO 18(PCM_CLK)
14	GND	Ground
16	HOST_FPAA_CFGFLG_B	GPIO 23
18	HOST_FPAA_ACTIVATE	GPIO 24
20	GND	Ground
22	HOST_FPAA_ERR_B	GPIO 25
24	HOST_CE0	GPIO 8(CE0)
26	NC	GPIO 7(CE1)
28	HOST_ID_SC	GPIO1(ID_SC)
30	GND	Ground
32	FPAA3_IO5P_RPI	GPIO 12(PWM0)
34	GND	Ground
36	FPAA0_IO5P_RPI	GPIO 16
38	FPAA2_IO5P_RPI	GPIO 20(PCM_DIN)
40	FPAA2_IO5N_RPI	GPIO 21(PCM_DOUT)

Pin Descriptions

The interface between the Raspberry Pi and the Pi.Ka HAT is separated into functional groups in the following sections.

Configuration

Configuration of the FPAA device is controlled through these configuration pins.

Pin	Pi.Ka Name	Raspberry Pi Name	Description
37	HOST_RESET_B	GPIO 26	Active low FPAA reset signal from the Raspberry Pi. 10K pullup on the Pi.Ka board. Configure as output driving high or as tri-stated input.
16	HOST_FPAA_CFGFLG_B	GPIO 23	Configuration status flag. Open drain output of FPAA. Configure as input to Raspberry Pi and monitor status.
18	HOST_FPAA_ACTIVATE	GPIO 24	Bidirectional open drain pin pulled low by FPAA until the configuration has been loaded. Configure as input to the Raspberry Pi and monitor status.
22	HOST_FPAA_ERR_B	GPIO 25	Bidirectional open drain error pin that is driven low if configuration error is detected. Configure as input to the Raspberry Pi and monitor status.

SPI

The SPI interface is a 4-pin serial interface consisting of a chip select, clock, data input, and data output. The Raspberry Pi is considered the master, and the Pi.Ka is considered the slave.

Pin	Pi.Ka Name	Raspberry Pi Name	Description
19	HOST_SPI0_MOSI	GPIO 10(MOSI)	Serial data output from Raspberry Pi into Pi.Ka
21	HOST_SPI0_MISO	GPIO 9(MISO)	Serial data input to Raspberry Pi from Pi.Ka
23	HOST_SPI0_SCLK	GPIO 11(SCLK)	Serial interface clock provided by Raspberry Pi
24	HOST_CE0	GPIO 8(CE0)	Active low chip select common to all FPAAs on the Pi.Ka board. Connected by jumper to FPAA_CS2_B.

Hat ID (I2C)

A hat identifier value may be stored on an EEPROM with I2C interface at address 0x50. The hat ID EEPROM is provided for customer use but is not programmed by Okika.

Pin	Pi.Ka Name	Raspberry Pi Name	Description
27	HOST_ID_SD	GPIO 0(ID_SD)	Hat ID EEPROM SDA
28	HOST_ID_SC	GPIO1(ID_SC)	Hat ID EEPROM SCL

Clocking

The clock source for the FPAA chips is controlled through the HOST_ACLK_SEL[1:0] pins.

Pin	Pi.Ka Name	Raspberry Pi Name	Description
7	HOST_ACLK_REF	GPIO 4(GPCLK0)	Reference ACLK sourced from Raspberry Pi when HOST_ACLK_SEL[1:0] = 11
8	LCL_ACLK_REF_EN	GPIO 14(TXD)	Enable signal for the local ACLK oscillator
29	HOST_ACLK_SEL0	GPIO 5	Source selection for reference ACLK source. HOST_ACLK_SEL[1:0] = 00: local 16MHz oscillator HOST_ACLK_SEL[1:0] = 01: J11 reference clock from previous stacked Pi.Ka. HOST_ACLK_SEL[1:0] = 10: external reference ACLK source from TP1. HOST_ACLK_SEL[1:0] = 11: Raspberry Pi GPIO 4 (GPCLK0).
31	HOST_ACLK_SEL1	GPIO 6	

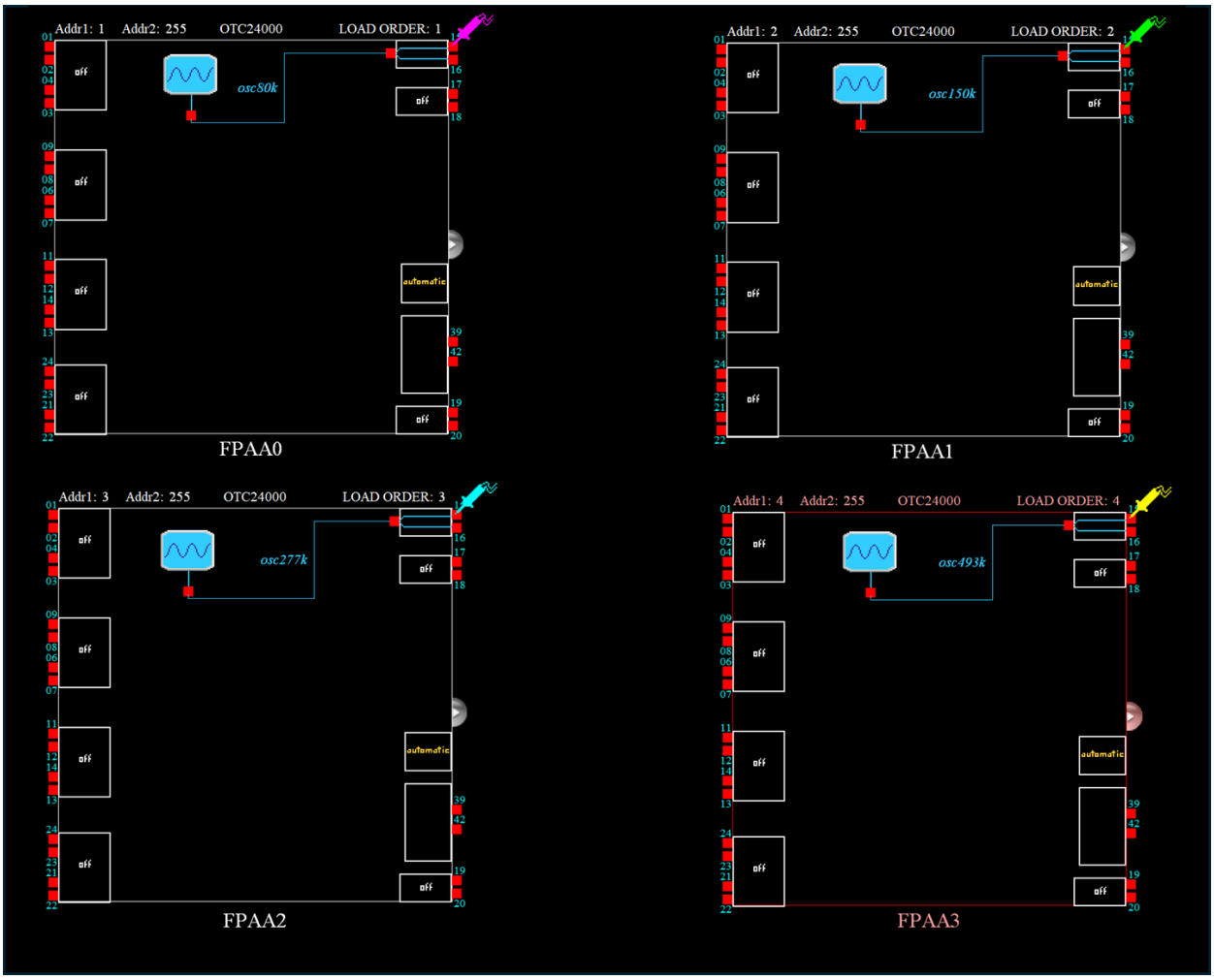
FPAA

Direct connections are provided between two pins on each FPAA and GPIOs on the Raspberry Pi. These pins are intended for customer use such as comparator outputs, ADC data capture, and function gating.

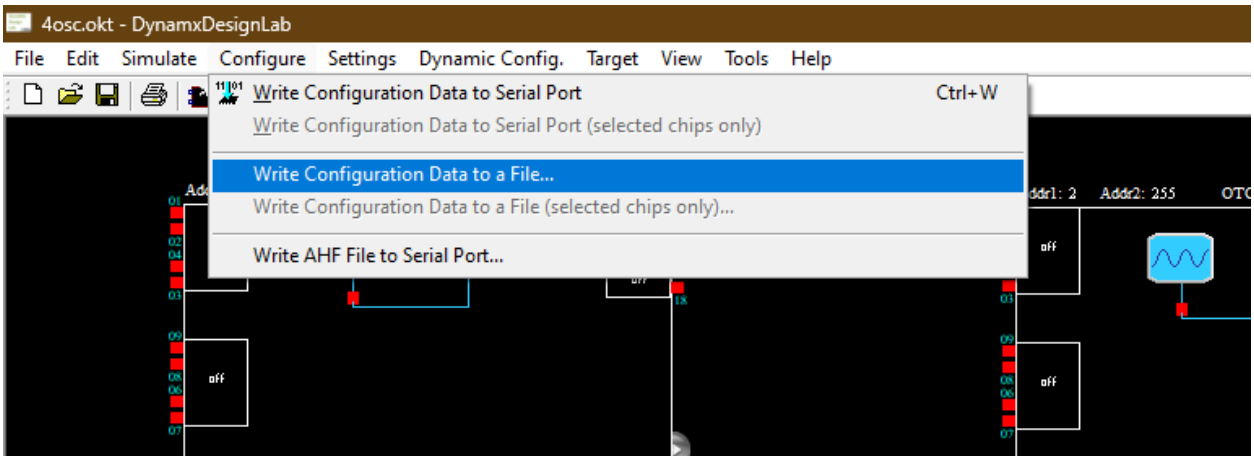
Pin	Pi.Ka Name	Raspberry Pi Name	Description
36	FPAA0_IO5P_RPI	GPIO 16	IO5 P output of FPAA0
11	FPAA0_IO5N_RPI	GPIO 17	IO5 N output of FPAA0
12	FPAA1_IO5P_RPI	GPIO 18(PCM_CLK)	IO5 P output of FPAA1
35	FPAA1_IO5N_RPI	GPIO 19(PCM_FS)	IO5 N output of FPAA1
38	FPAA2_IO5P_RPI	GPIO 20(PCM_DIN)	IO5 P output of FPAA2
40	FPAA2_IO5N_RPI	GPIO 21(PCM_DOUT)	IO5 N output of FPAA2
32	FPAA3_IO5P_RPI	GPIO 12(PWM0)	IO5 P output of FPAA3
33	FPAA3_IO5N_RPI	GPIO 13(PWM1)	IO5 N output of FPAA3

Primary Configuration .ahf File Generation

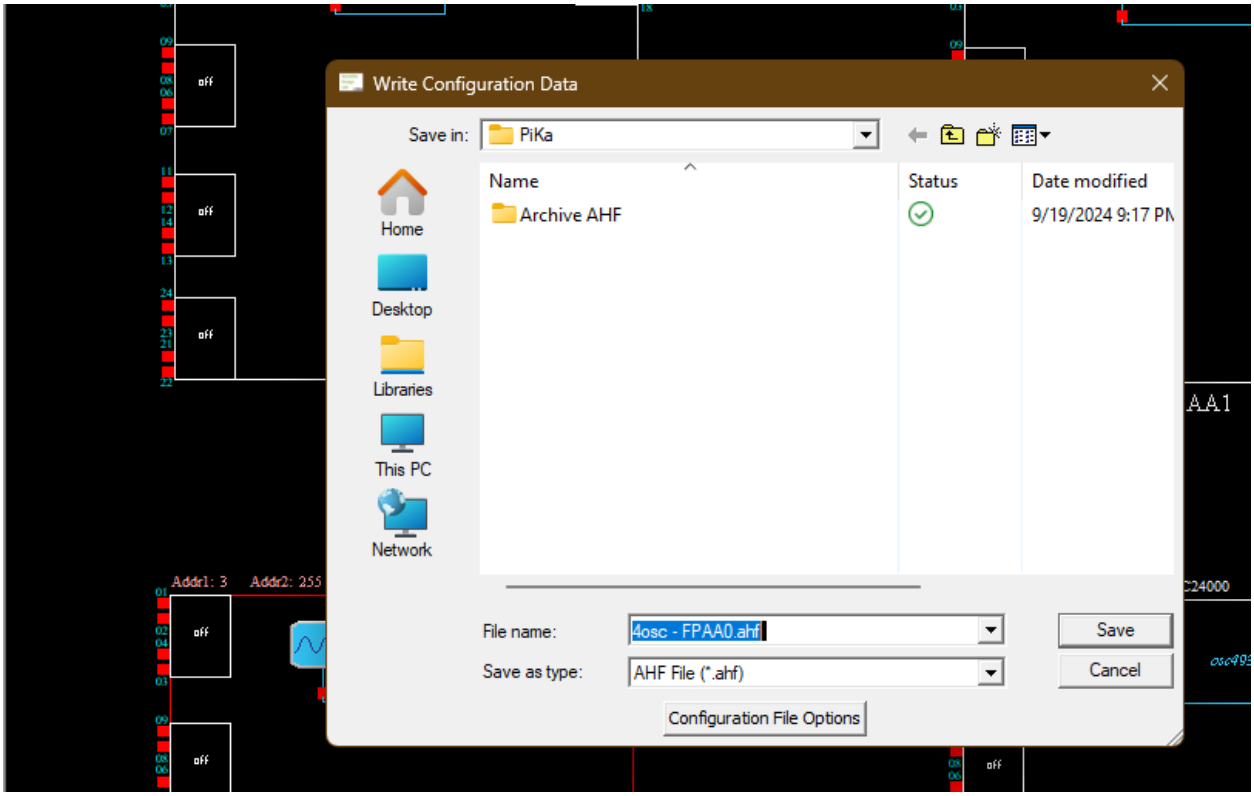
All FPAA's must be loaded with a primary configuration prior to use. The configuration may be changed on the fly using secondary configuration, but this is beyond the scope of this quick- start guide. The primary configuration process will be demonstrated by using DynAMx Design Lab to generate a different sine wave oscillator output from each of the 4 FPAAs on the Pi.Ka board. The configuration will be exported to a .ahf file, and a Python script will be used to import the .ahf file and configure the FPAAs. An oscilloscope can be used on the header pins to confirm that the output of the Pi.Ka board matches the simulation output in DynAMx Design Lab.

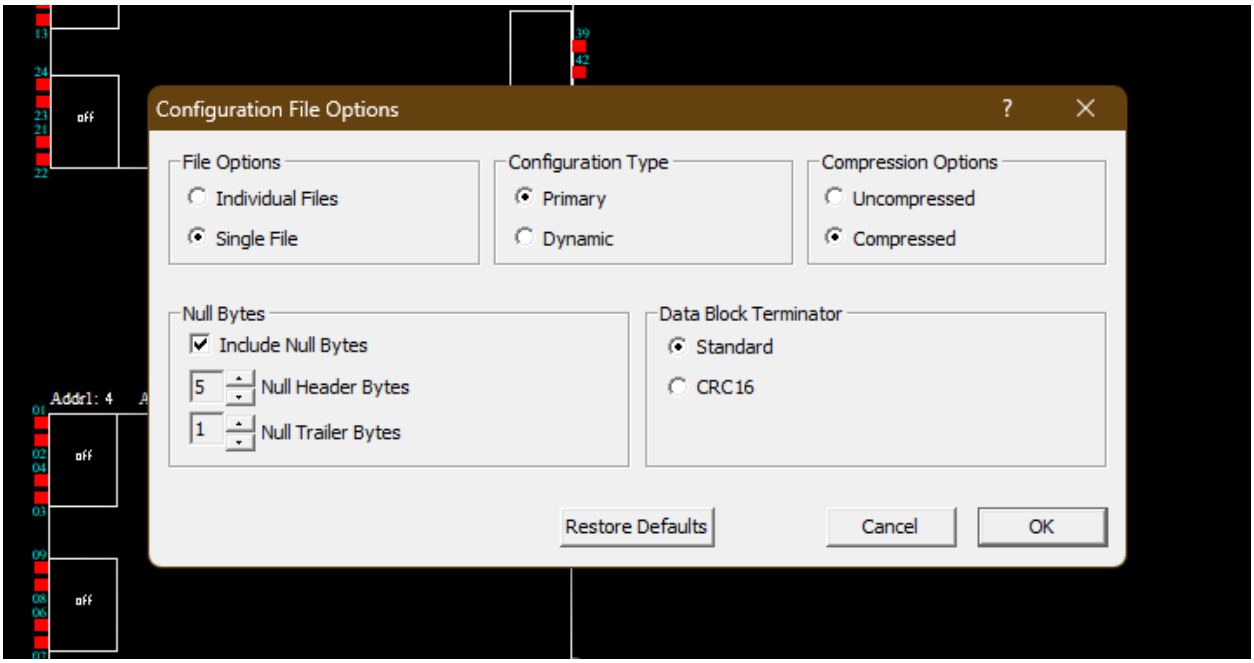


The example circuit above is named 4osc.okt and is available in the Github location along with the Python code and sample .ahf file. To export the configuration, click on the Configure tab and then click on “Write Configuration Data to a File…”.

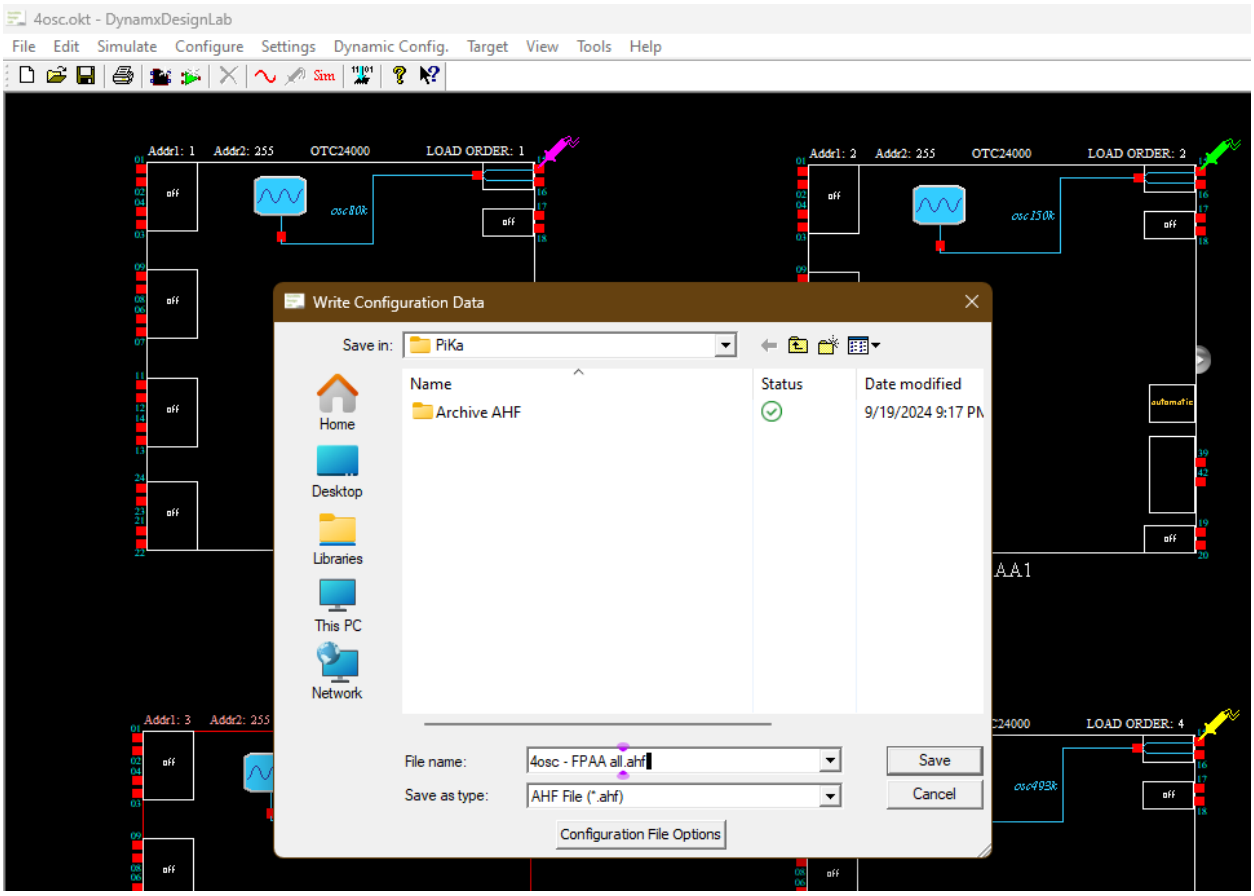


Next, click on the Configuration File Options button. Choose the “Single File” option, “Primary” Configuration Type, and “Standard” Data Block Terminator. The Null Bytes and Compression Options may be left at their default settings. When finished, press “OK” to continue.





Change the File name if desired and then press the Save button to generate the .ahf file. In the example below, the file name has been changed to indicate that the configuration data for all 4 FPAAs is stored in the same file.



Transfer the .ahf file to the Raspberry Pi using any suitable means (e.g. thumb drive or shared network location) and modify the example Python script if necessary to point to the generated .ahf file.

Python Configuration Code

Example Python code demonstrating Pi.Ka configuration can be accessed on Github at the following link:

https://github.com/okikajre/pikapy

Required Python libraries are spidev, RPi.GPIO, time, and sys

The SPI module must be enabled on the Raspberry Pi. Instructions for doing this are included in the comments within the spitest.py file.

The Python example code assigns GPIO names consistent with the Pi.Ka signal names and sets up the necessary port directions and initial states. The chip select pin is controlled as a GPIO to prevent toggling between SPI transfers. The example code uses the 16MHz oscillator on the Pi.Ka board to clock the FPAAs. Commented code shows how the pigpio library could be used to map the Raspberry Pi hardware clock onto the FPAA_ACLK_REF pin.

The FPAA_RESET_B pin is configured low initially to force a reset of the FPAAs at the start of the script. The reset is held for 20ms before being released for 100ms before beginning the configuration process. These delays are much longer than required.

The configuration process begins by sending 8 dummy SPI clocks and checking that the FPAA_ERR_B signal goes high. A message is printed to the screen if the FPAA_ERR_B signal remains low, which most likely indicates that the ACLK is not reaching the FPAAs.

In this test code, the user is prompted to press enter to continue. The contents of the .ahf file are read into a list, stripping out the newline character in the process. Each line is interpreted as a hex integer value. The list created from the file is then sent as a SPI transfer. The limit for the xfer2 function is 4096 bytes, which is more than adequate for the sample .ahf file. Users should be aware of this limit. Alternate configuration files can be evaluated by changing the "ahf_file_name" variable.

After configuration, the sample code waits for the user to press enter before closing the SPI interface and cleaning up the GPIO.

Configuration File Format

FPAA configuration files are exported to a .ahf file that contains one byte per line. For details of the configuration contents, refer to the Okika Technologies OTC24000 FPAA User Manual or the Anadigm Apex dpASP Family User Manual.

The contents of the .ahf file must be loaded into the FPAA chips using a SPI-compatible interface. The first 5 bytes (40 clocks) of each .ahf file are 0s and are provided as part of the data synchronization method. Seven bytes of header data follow, starting at the 6th byte in the .ahf file, which is always 0xD5. The FPAA detects this synchronization byte as the start of the configuration. The next 4 bytes are the device ID. The Okika OTC24000 and Anadigm AN231E04 will only receive the configuration data if the device ID bytes match 0xB7200100. The last two bytes of the header are the ADDR1 byte and control byte.

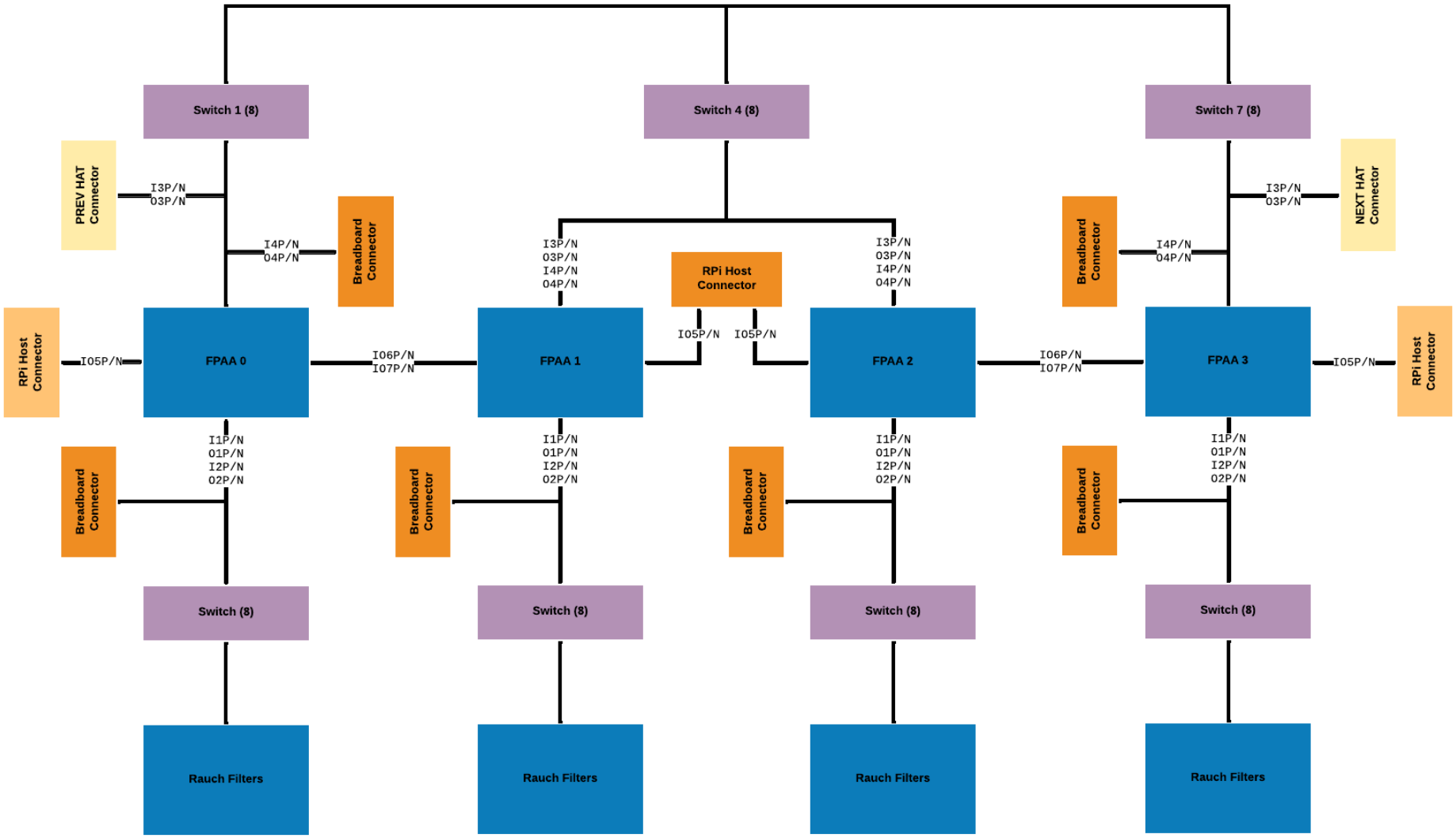
The ADDR1 byte includes the logical primary address that is assigned to the chip during a primary configuration. When multiple FPAA chips are connected in series, the ADDR1 byte is assigned sequentially starting with 1 for the first chip in the chain.

The CONTROL byte instructs the FPAA how to interpret the incoming data, whether to apply any clock divisor settings included in the stream, whether to apply internal pull-ups to the CFGFLGb and ACTIVATE pins, and whether to immediately activate the configuration after it has been fully received. Note that the datasheet says that bit 5 of the CONTROL byte must be 1, but .ahf files may be generated with bit 5 set to 0. This configures the chip for short error pulses but does not otherwise affect the performance of the chip.

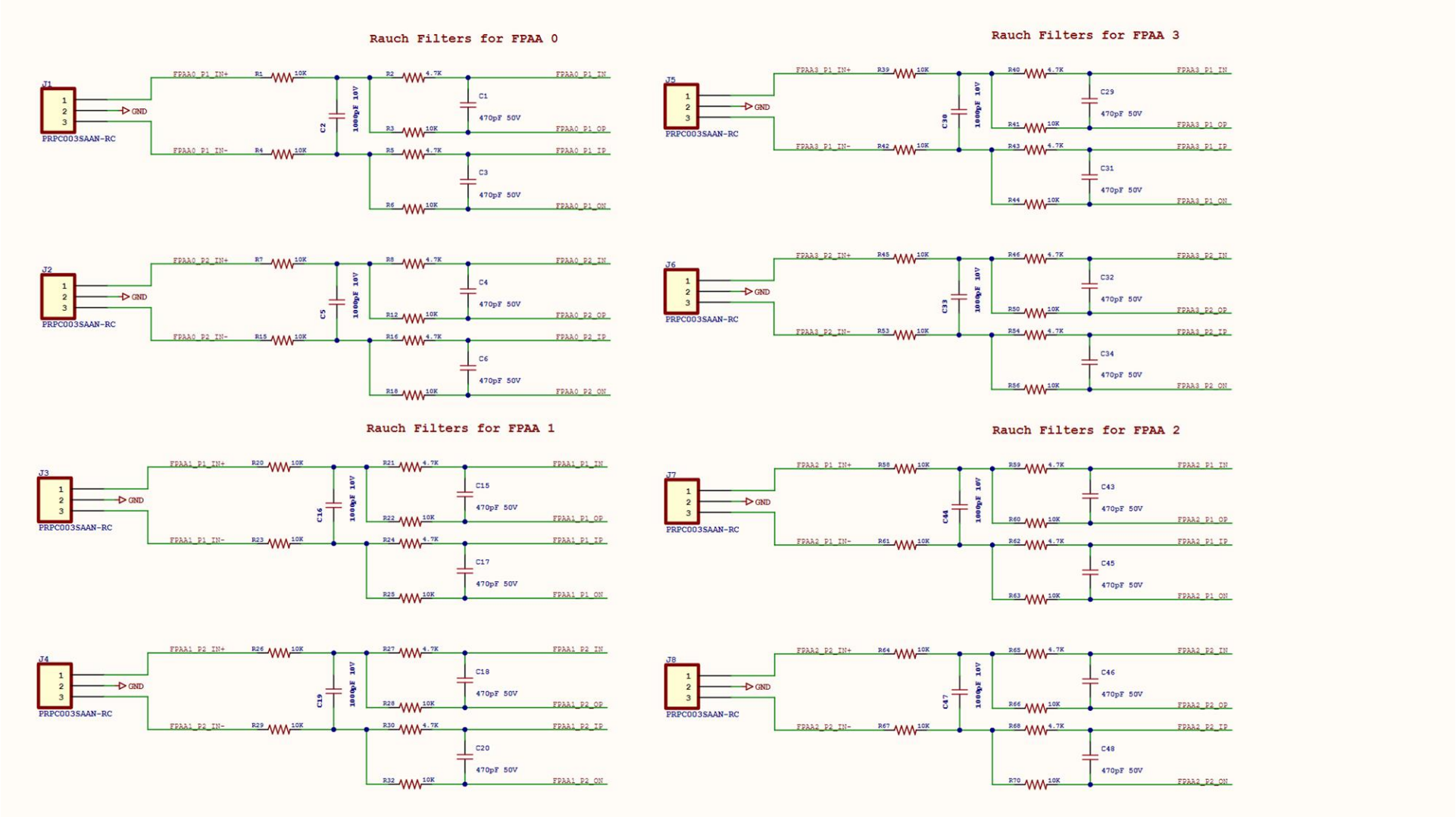
After the header block, one or more data blocks are sent to the FPAA. The first byte of a data block is the BYTE_ADDRESS. If BYTE_ADDRESS[7:6] = 11, there will be additional blocks to follow. If BYTE_ADDRESS[7:6] = 10, this is the last data block for this chip's configuration. Each data block is terminated by either a fixed byte of 0x2A or two bytes of CRC, depending on whether CRC was enabled when the configuration file was generated. For multi-chip configuration files, following the last data

block for one chip, there is a single byte of 0s (8 clocks) before the configuration byte is sent for the next chip.

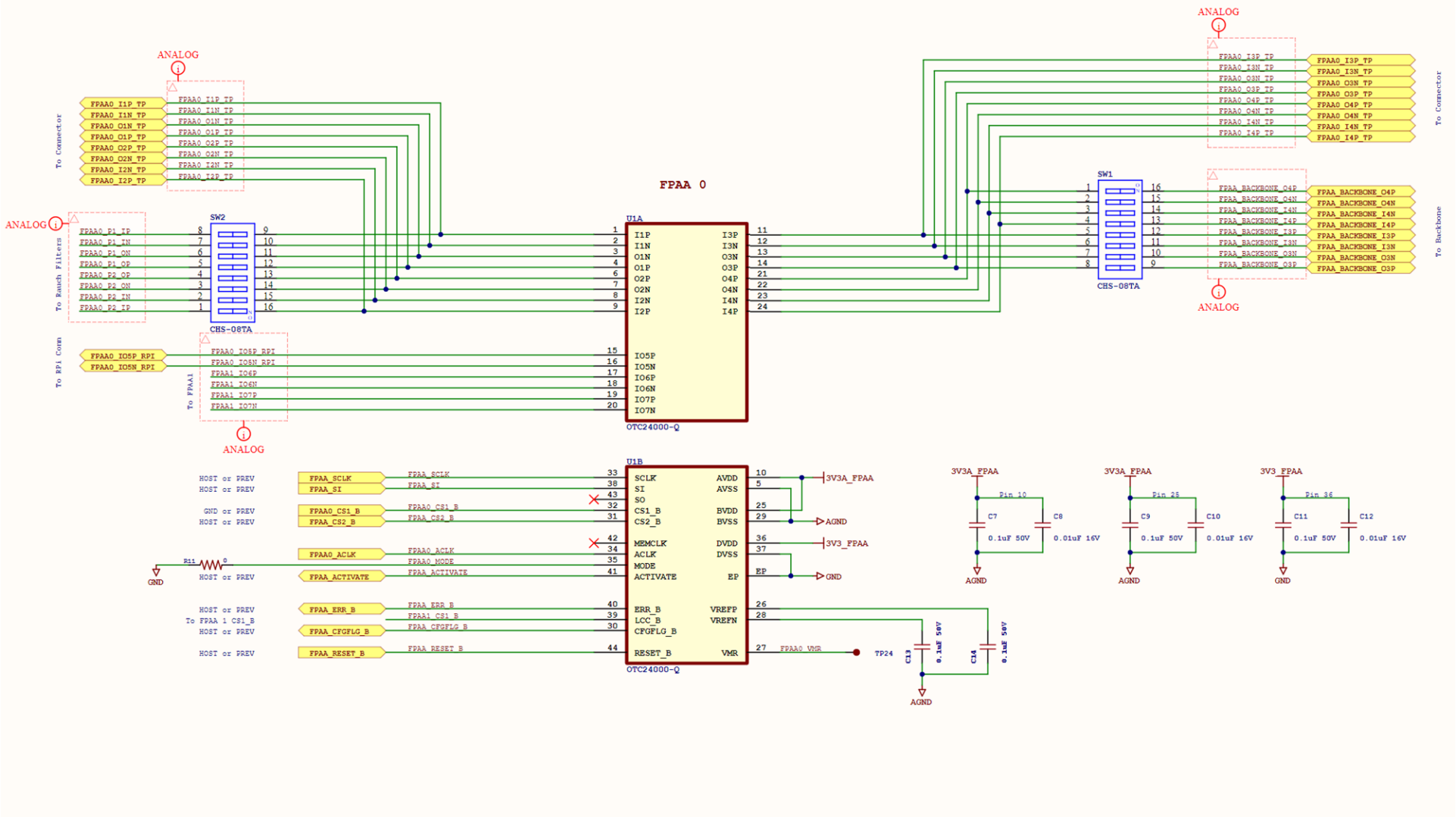
Pi.Ka FPAA Connection Diagram



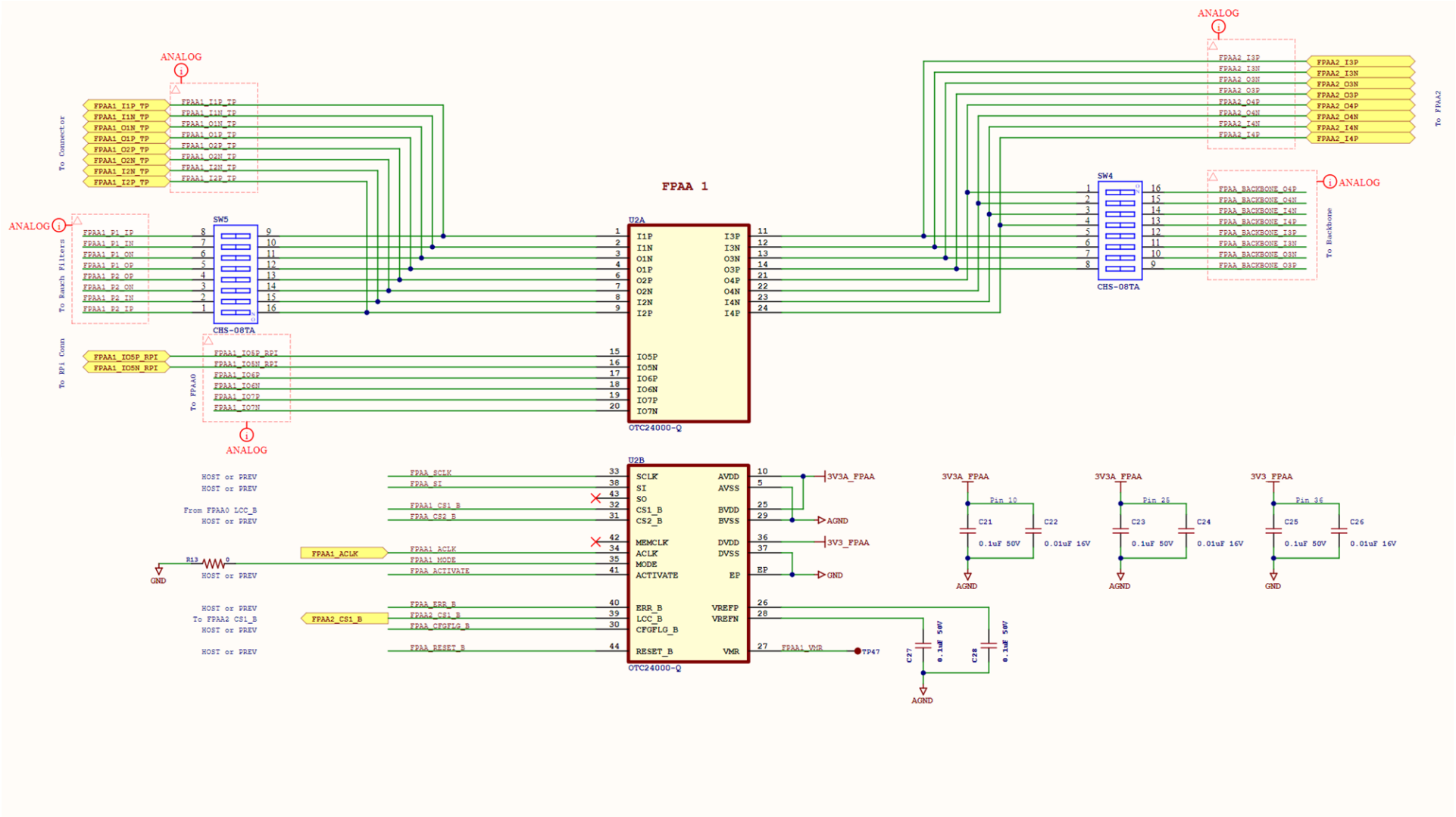
Schematic Page 1: Rauch Filters



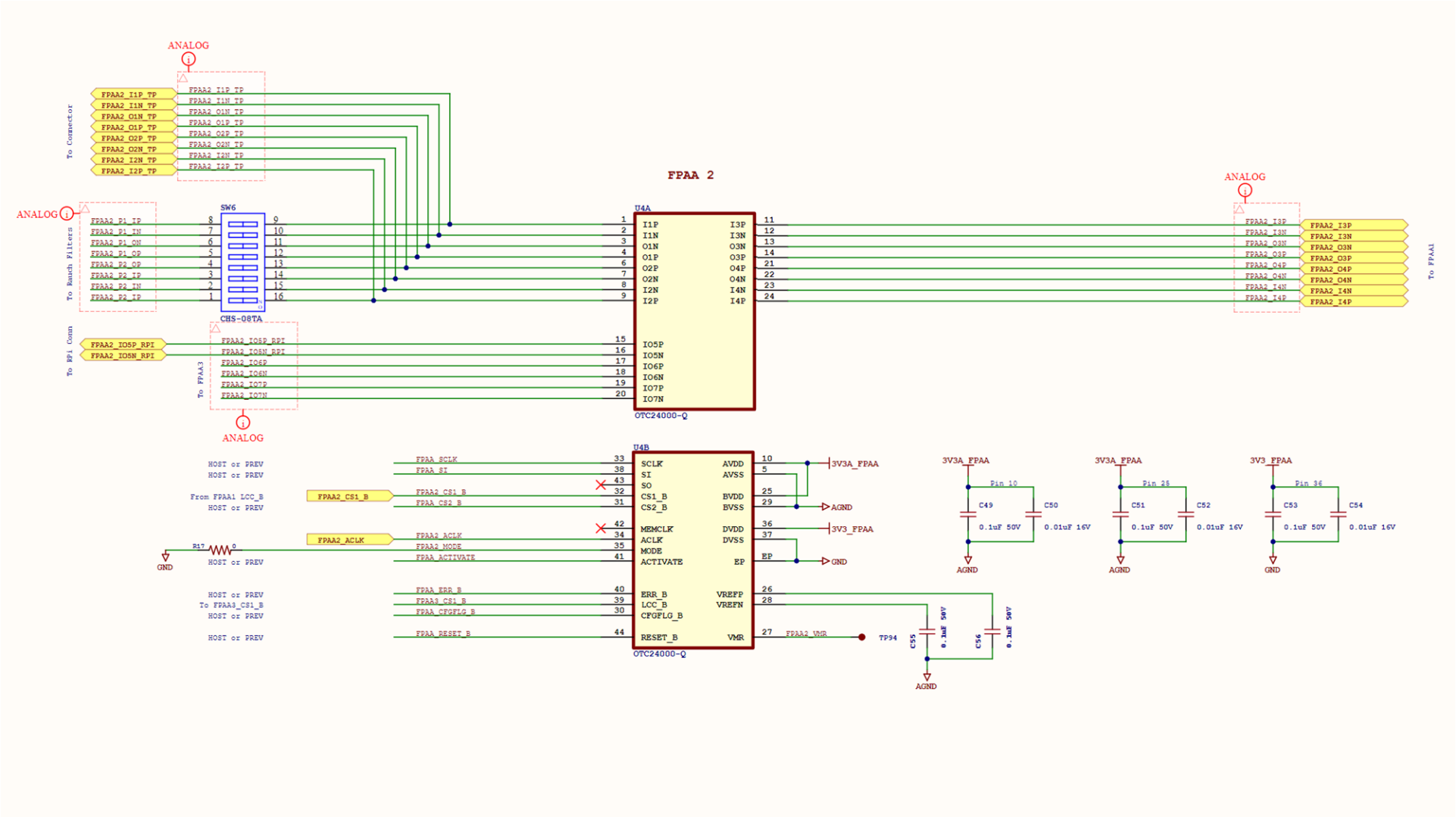
Schematic Page 2: FPAA 0



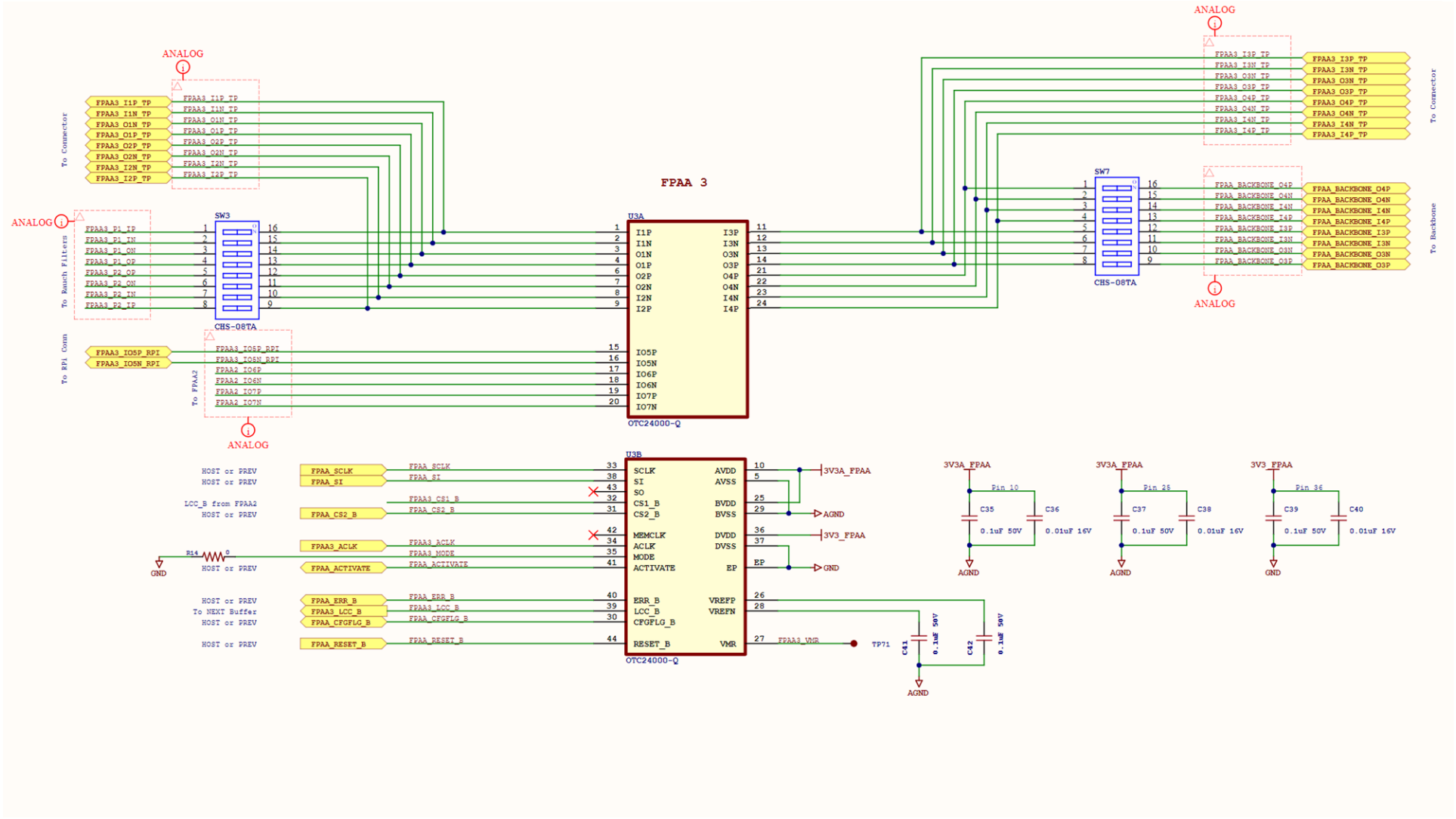
Schematic Page 3: FPAA 1



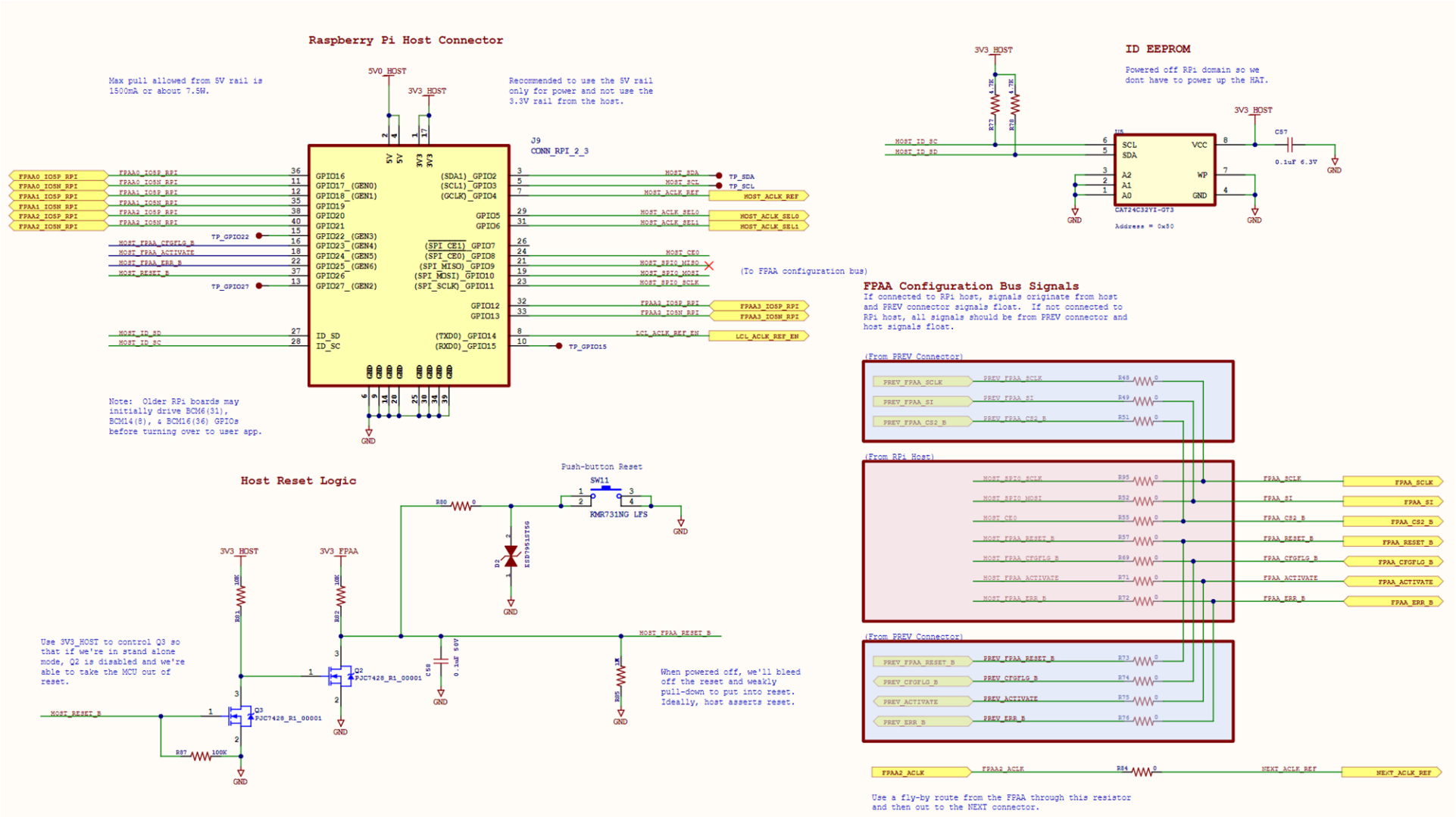
Schematic Page 4: FPAA 2



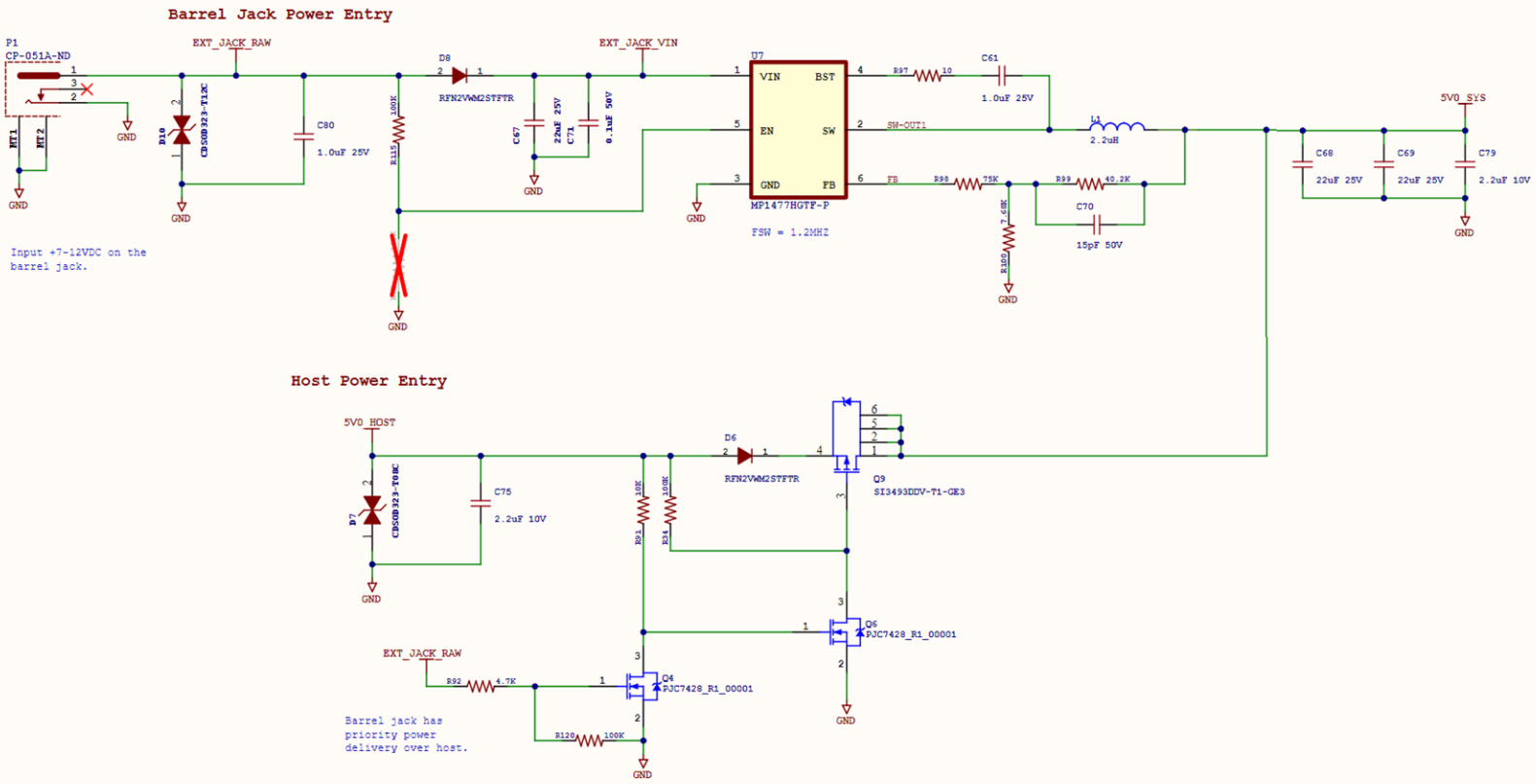
Schematic Page 5: FPAA 3



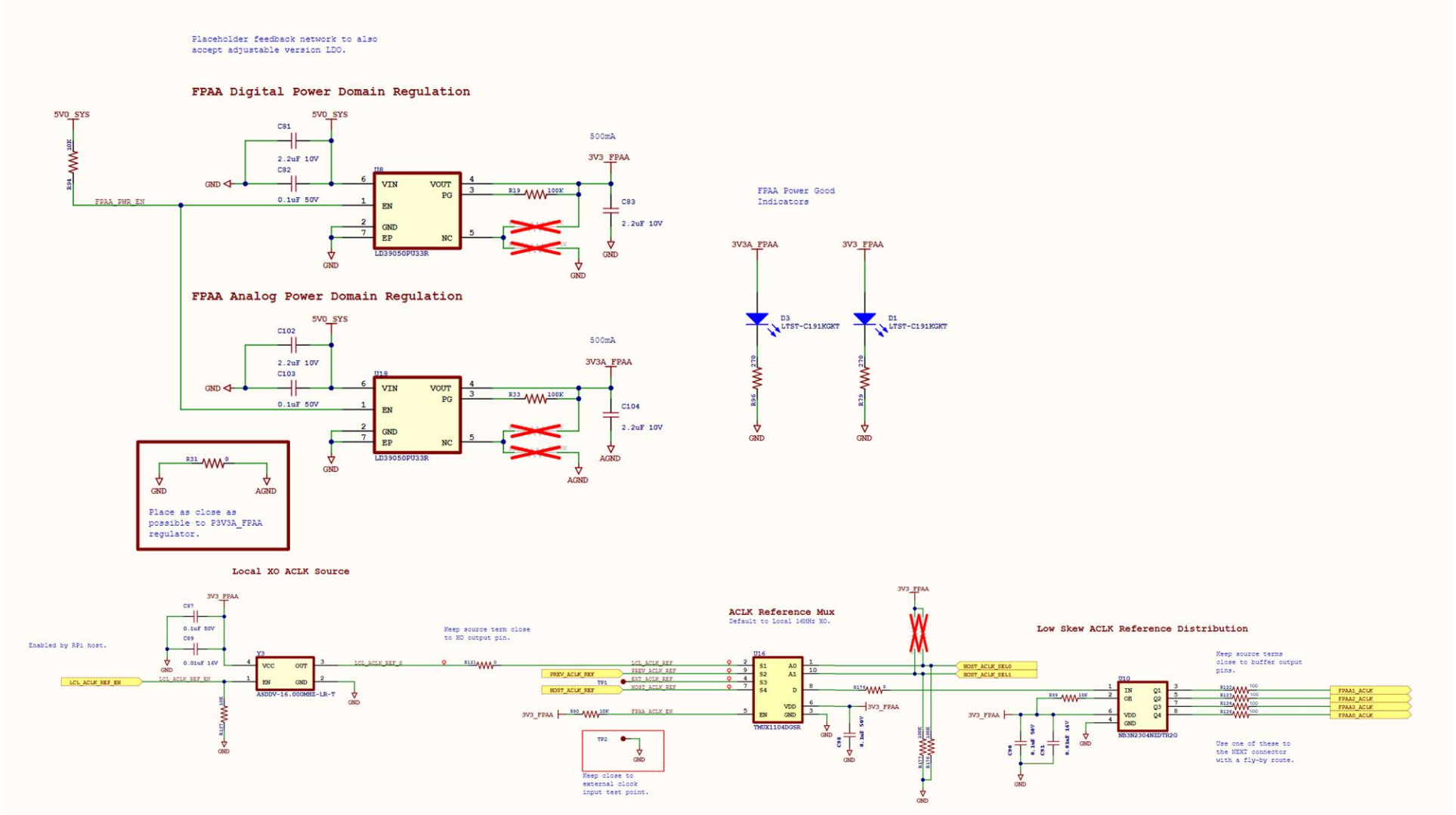
Schematic Page 6: Host and Configuration



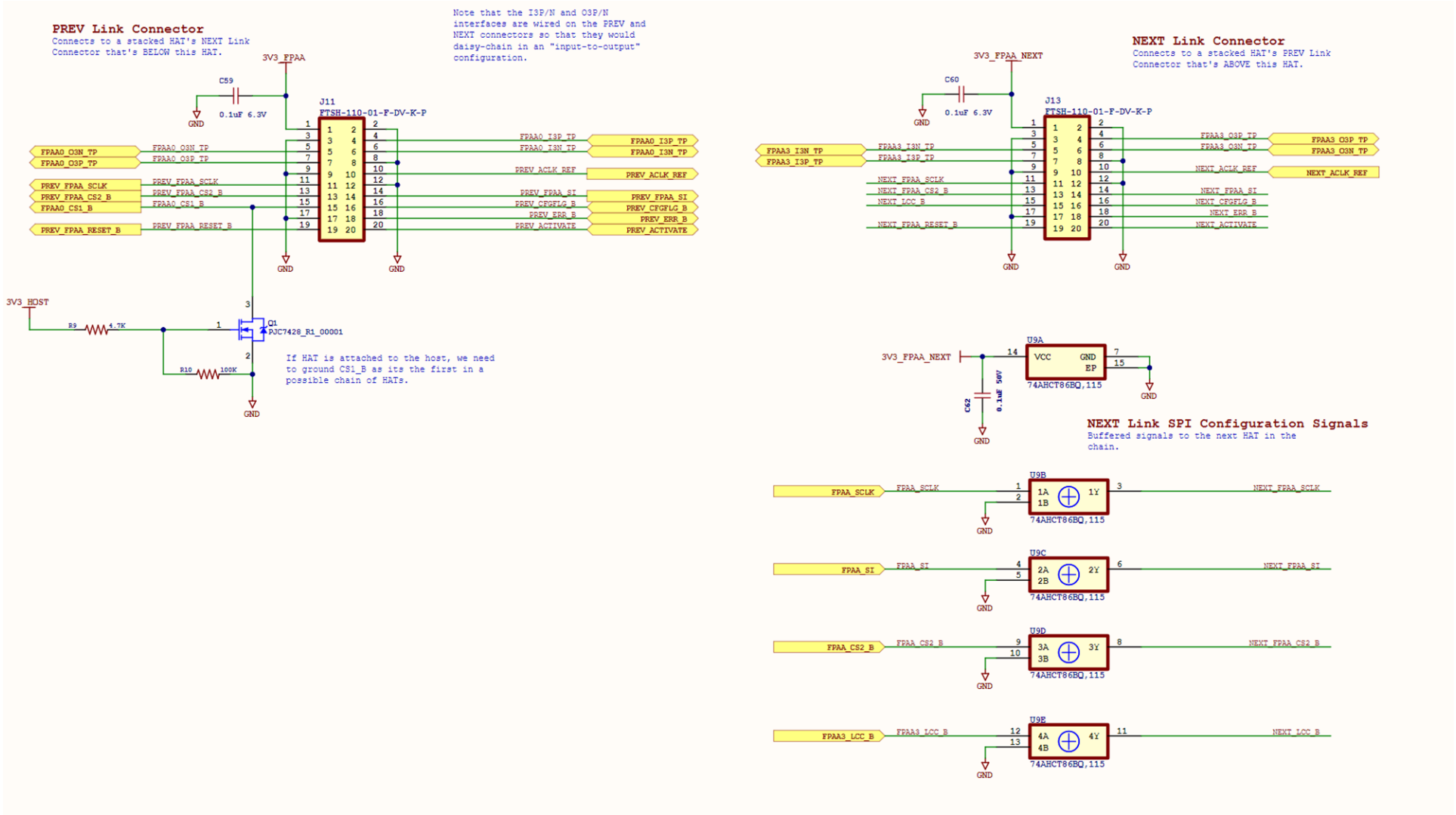
Schematic Page 7: System Power



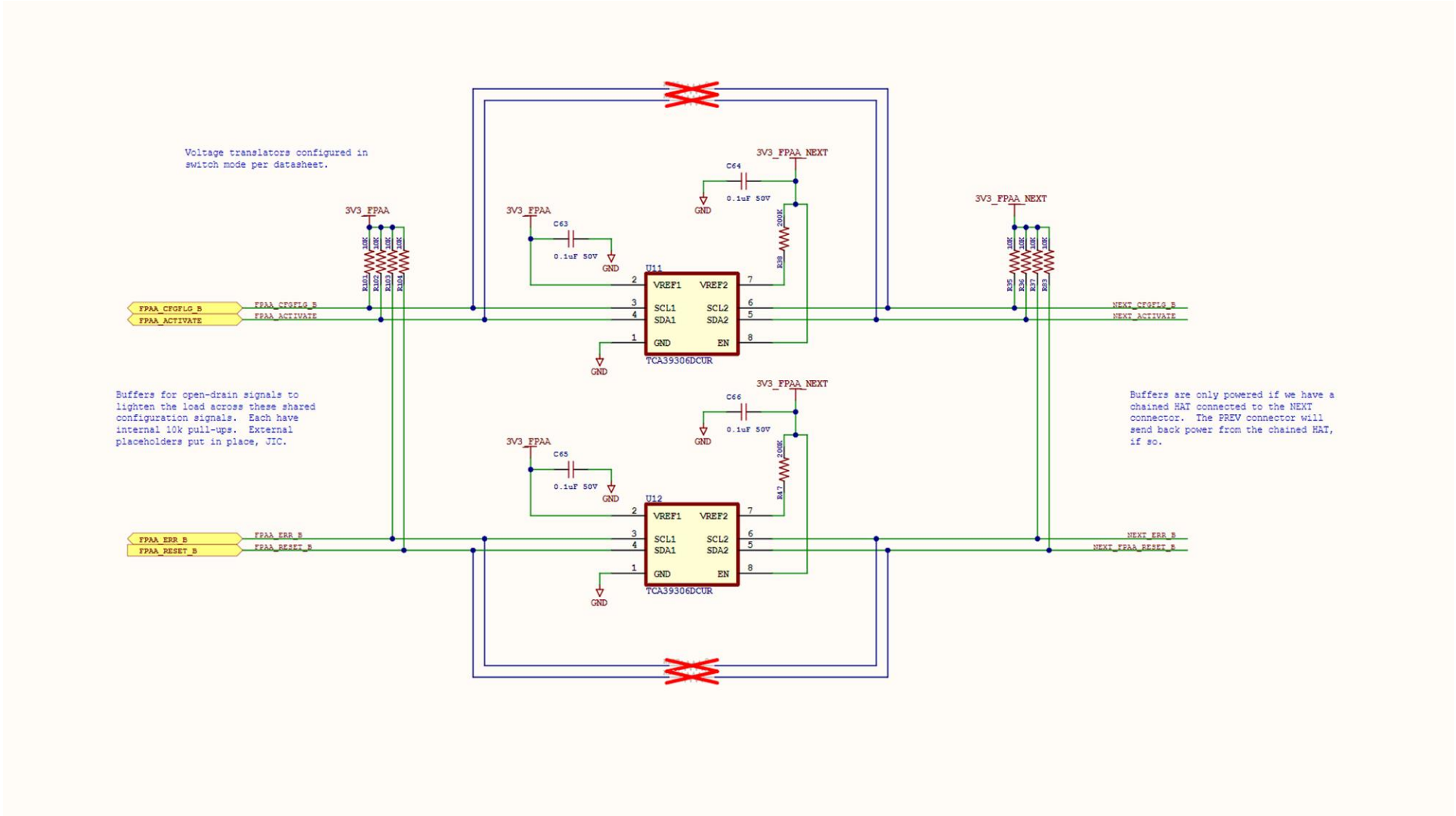
Schematic Page 8: FPAA Power and Clocking

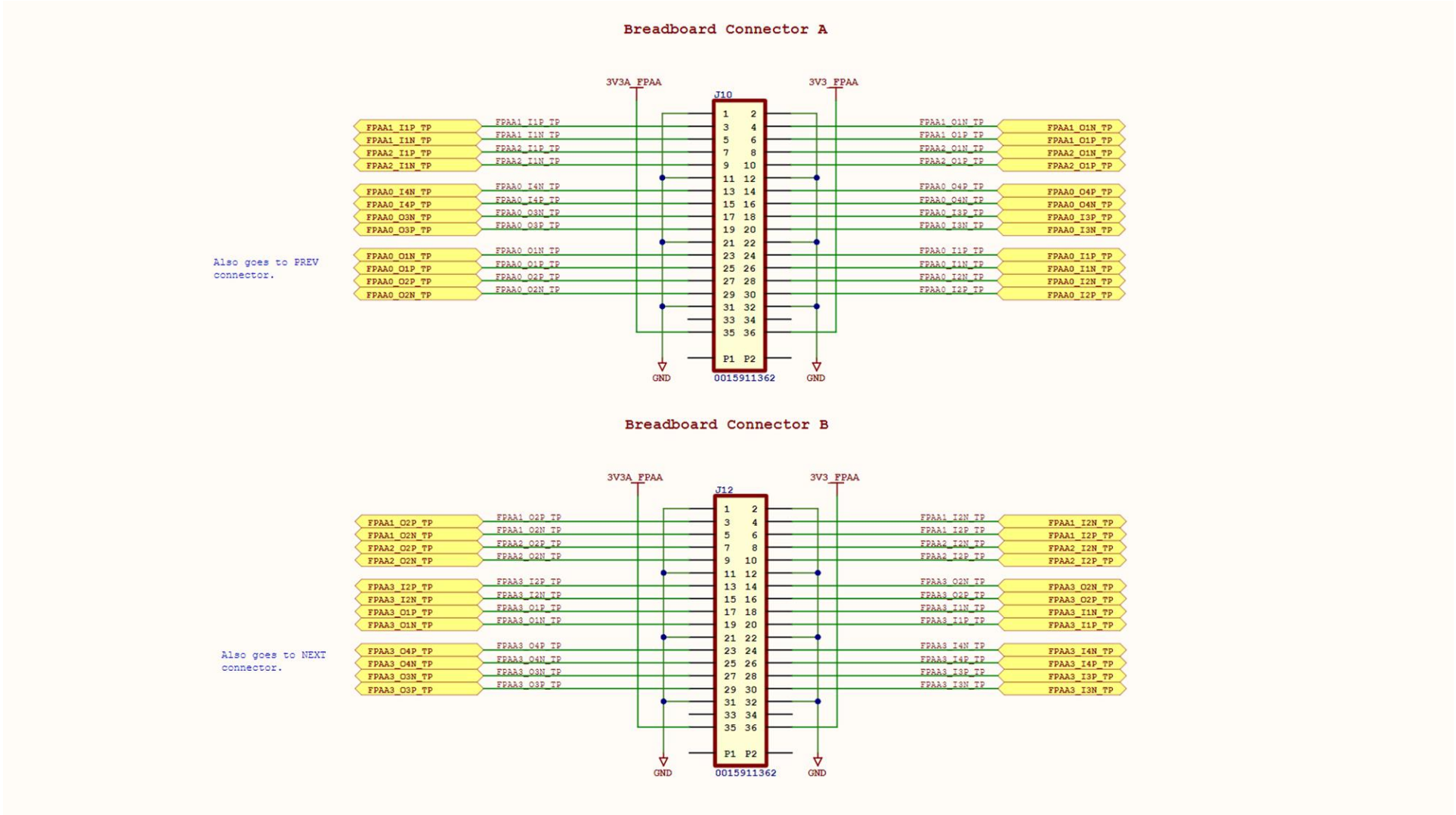


Schematic Page 9: Previous and Next Link Connectors



Schematic Page 10: Configuration Switches




Breadboard Connector A
3V3A_FPAA
3V3_FPAA
J10
FPAA1_I1P_TP
FPAA1_I1N_TP
FPAA2_I1P_TP
FPAA2_I1N_TP
FPAA0_I4N_TP
FPAA0_I4P_TP
FPAA0_O3N_TP
FPAA0_O3P_TP
FPAA0_O1N_TP
FPAA0_O1P_TP
FPAA0_O2P_TP
FPAA0_O2N_TP
Also goes to PREV connector.
FPAA1_O1N_TP
FPAA1_O1P_TP
FPAA2_O1N_TP
FPAA2_O1P_TP
FPAA0_O4P_TP
FPAA0_O4N_TP
FPAA0_I3P_TP
FPAA0_I3N_TP
FPAA0_I1P_TP
FPAA0_I1N_TP
FPAA0_I2N_TP
FPAA0_I2P_TP
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
19 20
21 22
23 24
25 26
27 28
29 30
31 32
33 34
35 36
P1 P2
GND
0015911362
GND
Breadboard Connector B
3V3A_FPAA
3V3_FPAA
J12
FPAA1_O2P_TP
FPAA1_O2N_TP
FPAA2_O2P_TP
FPAA2_O2N_TP
FPAA3_I2P_TP
FPAA3_I2N_TP
FPAA3_O1P_TP
FPAA3_O1N_TP
FPAA3_O4P_TP
FPAA3_O4N_TP
FPAA3_O3N_TP
FPAA3_O3P_TP
Also goes to NEXT connector.
FPAA1_I2N_TP
FPAA1_I2P_TP
FPAA2_I2N_TP
FPAA2_I2P_TP
FPAA3_O2N_TP
FPAA3_O2P_TP
FPAA3_I1N_TP
FPAA3_I1P_TP
FPAA3_I4N_TP
FPAA3_I4P_TP
FPAA3_I3P_TP
FPAA3_I3N_TP
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
19 20
21 22
23 24
25 26
27 28
29 30
31 32
33 34
35 36
P1 P2
GND
0015911362
GND

Pi.Ka Stacking

The Pi.Ka board is designed to support chaining of multiple boards if more than 4 FPAA's are required for an application. Connection between Pi.Ka boards is provided through ribbon cables at J11 and J13. A ribbon cable is connected from J13 of one board to J11 on the next board in the chain. Because power can be applied to each Pi.Ka board through the barrel jack, it is unnecessary to vertically stack the Pi.Ka hats.

The Pi.Ka board supports vertical stacking with other hats and with other Pi.Ka boards. However, all Pi.Ka boards that connect to the Raspberry Pi board 3.3V Power (pins 1 and 17 of the host connector) will have their FPAA0_CS1_B signal pulled low, and each board's FPAA0 will respond as if it were the first chip in the chain. Furthermore, the IO5P and IO5N pins of each FPAA will be connected in parallel to the matching FPAA on the vertically stacked boards, and all of the stacked Pi.Ka boards have the same I2C address for hat ID reads.

In order to vertically stack multiple Pi.Ka boards and chain their configuration programming pins, it is necessary to remove R9 from all Pi.Ka boards except for the first Pi.Ka board in the chain. It is also necessary to remove the following jumper resistors from all but the first board in the stack: R95, R52, R55, R69, R72, R57, R71.

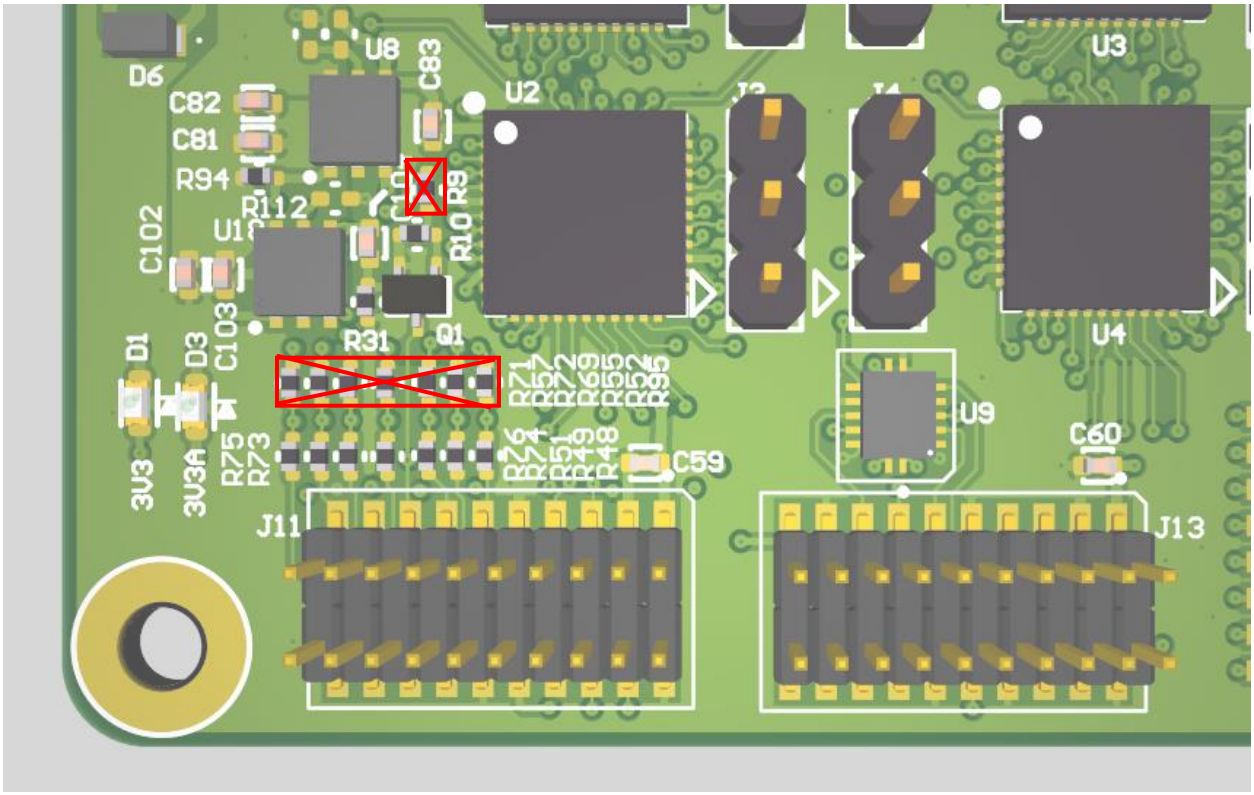


Figure 2- Resistors to Remove on Stacked Pi.Ka Boards

The limit for vertical Pi.Ka board stacking has not been determined and will depend on whether the power is supplied from the barrel jack or the Raspberry Pi. The Pi.Ka board includes buffers for the configuration signals to the next board in the chain.