Electronics, Hardware, Networking, Reverse Engineering

Reverse engineering of the Starlink Ethernet adapter

Oleg Kutkov / March 7, 2022

The new generation of the Starlink terminal was released at the end of 2021. The Dishy antenna is square now, and a completely redesigned indoor unit combines a WiFi router and power supply. The new design should be more cost-effective, so there is no AUX Ethernet port on the router, only WiFi.
Later SpaceX released an official Ethernet adapter that brings up the port back.

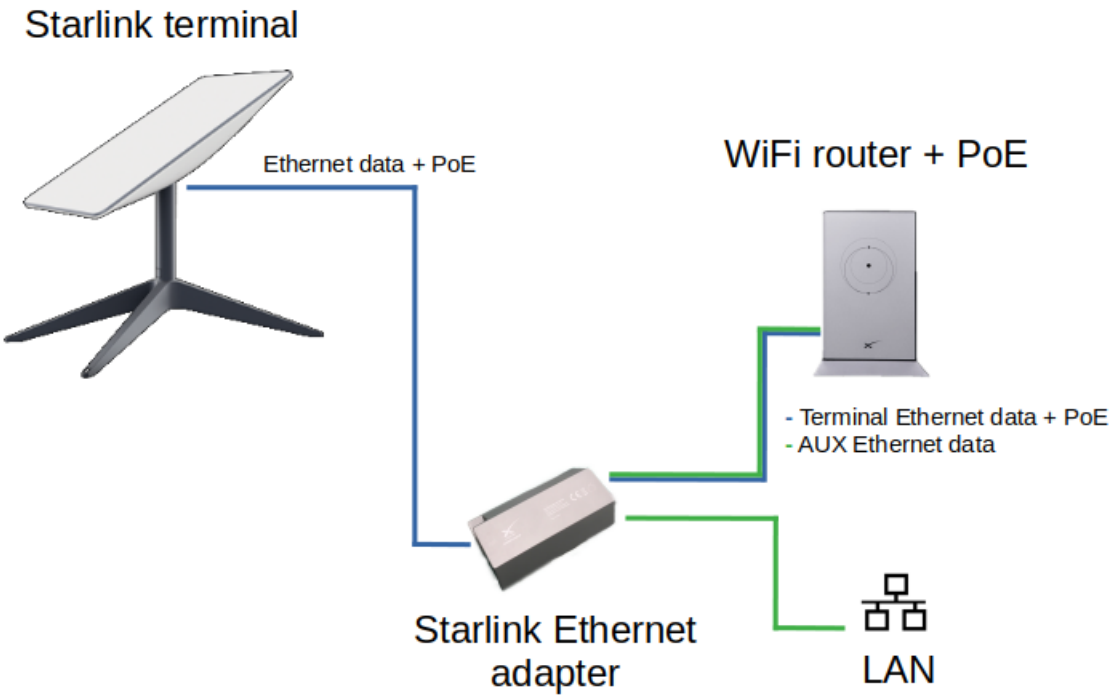
To be honest, I'm not a fan of this solution. I don't have the new router yet, so I can't tell if there are any design limitations inside the device.
Probably SpaceX has some usage stats that show most users use only WiFi.

I managed to find the adapter and reverse engineering this simple device. Just why not 🙃

The Starlink ethernet adapter has the Ethernet port, Dishy proprietary port, and Dishy proprietary connector.

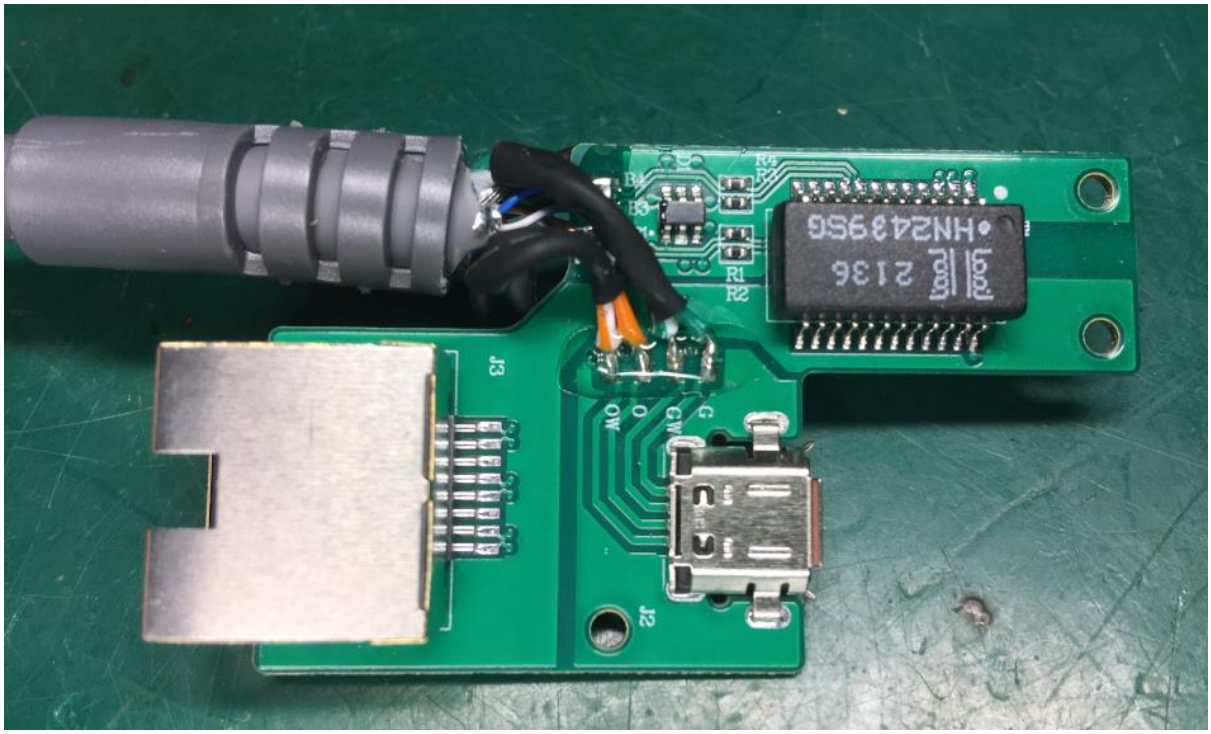
The new waterproof connector was introduced in the new Dishy design. Previously they used standard Ethernet.

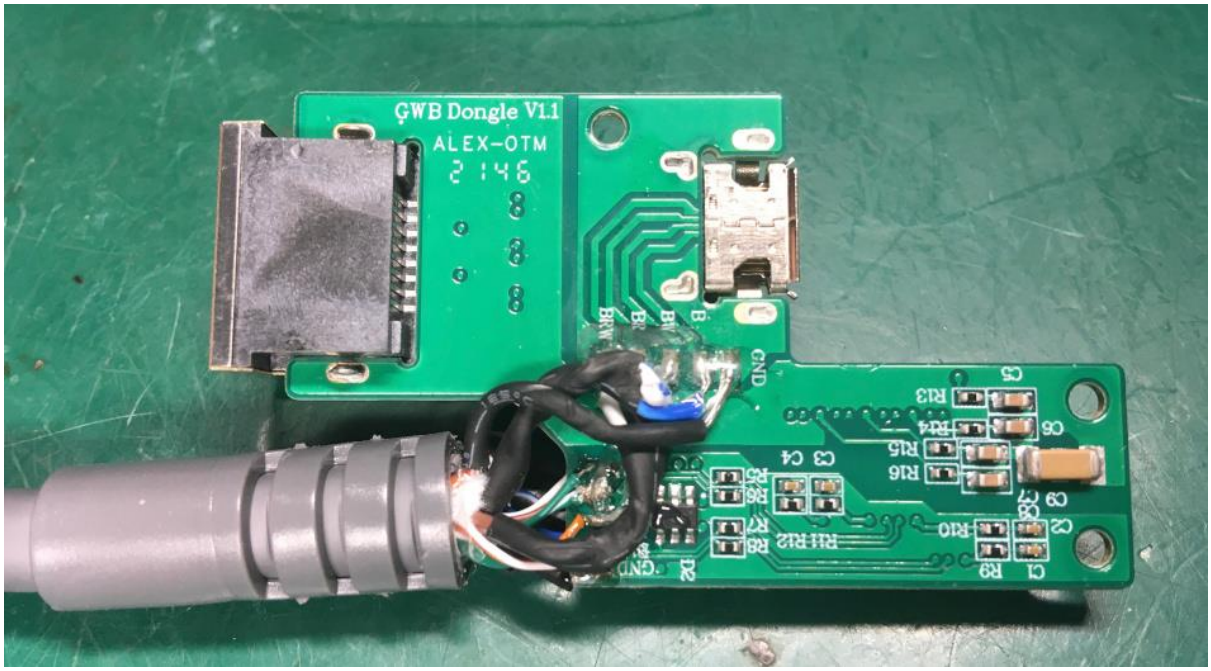
Basically, this adapter connects the Dishy terminal and the WiFi router.



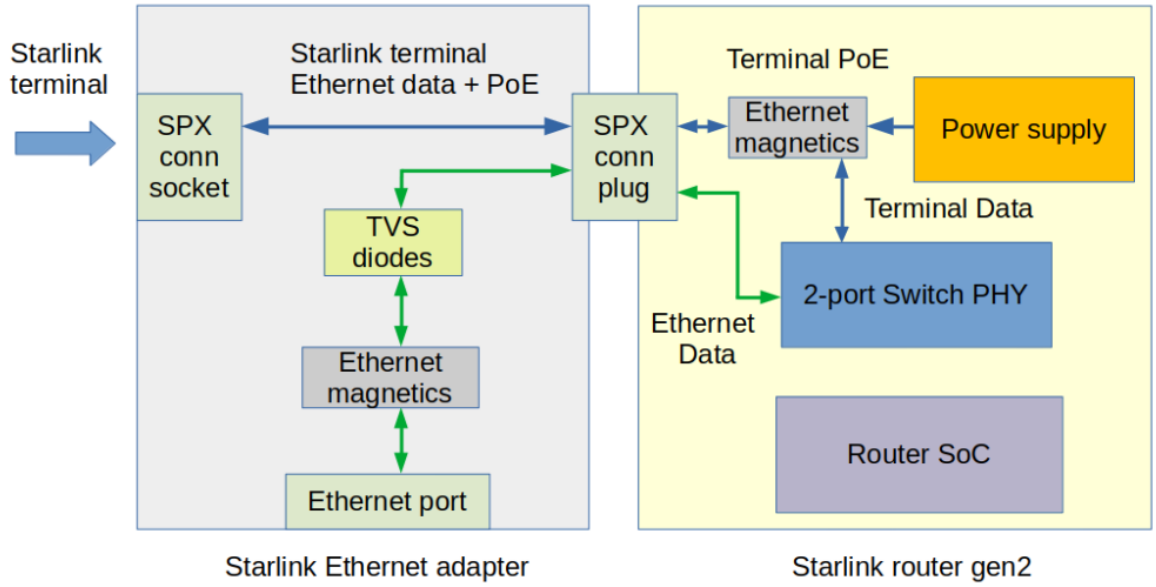
It’s easy to open the plastic cover with a flat screwdriver. Not much in there:

The Double-sided board contains connectors and an Ethernet circuit.



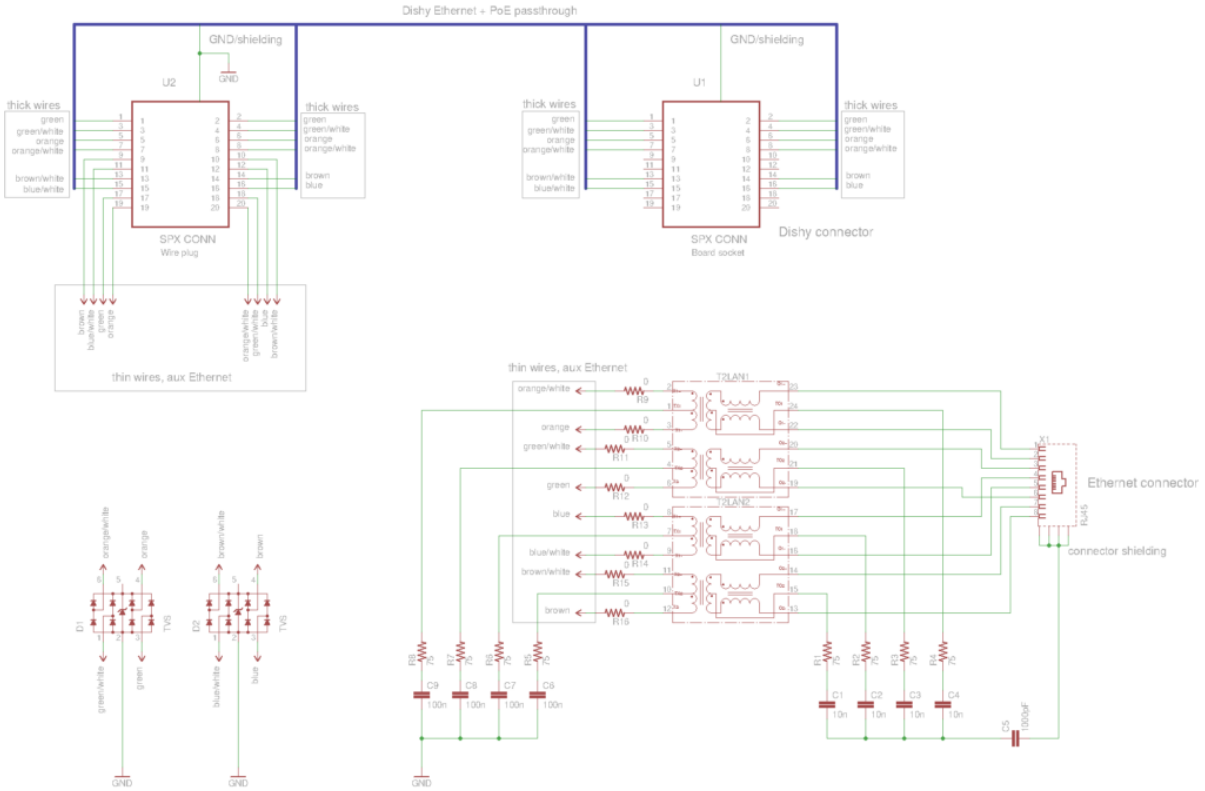


So, this is just a Dishy Ethernet + PoE passthrough and AUX Ethernet transformer with a required minimal circuit.
This means that the router contains a 2-port switch.



The first port is used for Dishy. All required Ethernet frontend is on the router board. The second port is used for the AUX Ethernet, but the Ethernet transformer is now in this adapter. It saves a few bucks and some space on the router board. But I don't think this is good to move Ethernet magnetics so far away from the PHY IC.

Here is a schematic of the adapter:



Click on the image for the full resolution schematic of the Starlink adapter.

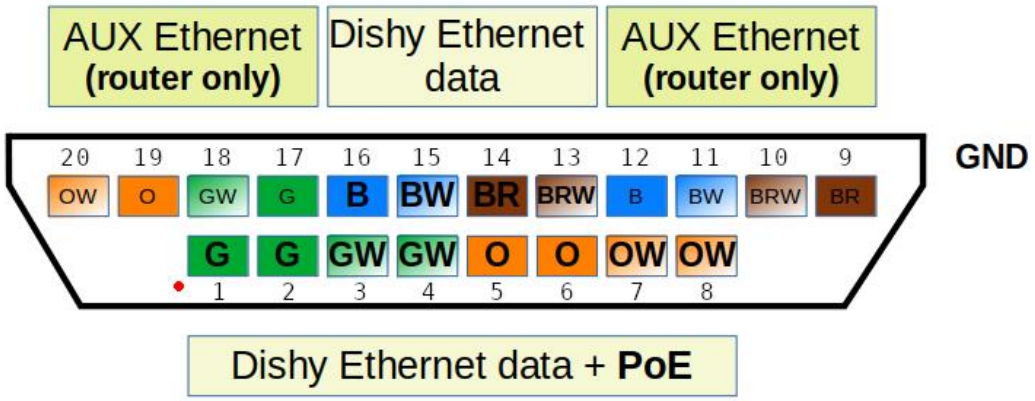
The short adapter cable contains two twisted-pair lines with separate shieldings.
Thicker wires are used for the Dishy Ethernet+PoE passthrough.
Thinner wires connect the AUX Ethernet circuit.

Both lines share common ground.

The Starlink proprietary connector it's a 20-pin interface that looks like a mini-HDMI but is not the same.
There is an SPX marking on the connector, so I guess it's custom-made for SpaceX.

Here is a pinout of the socket (router and adapter inputs):

SpaceX Starlink proprietary connector pinout. Square Dishy.
Router + External Ethernet adapter



Please note that contacts 9, 10, 11, 12, 17, 18, 19, and 20 are used only on the Router side. These are AUX Ethernet data lines.

It would be nice to get the new router and see if it's possible to integrate this circuit inside the router.

UPDATES, June 14

SPX proprietary connector it's a USB-C connector in an unusual shape. You can bend the USB-C to fit the SpaceX one.

The terminal use 802.3bt PoE that is carried over all eight lines: 1-2, 3-4, 5-6, 7-8, 13, 14, 15, 16.
You can find the custom PoE injector schematic in the comments below.

I managed to get the new router, so there will be some updates.

Thanks for reading.

Electronics (Https://Olegkutkov.Me/Category/Electronics/),
Firmware (Https://Olegkutkov.Me/Category/Firmware/),
Hardware (Https://Olegkutkov.Me/Category/Hardware/),
Linux Kernel (Https://Olegkutkov.Me/Category/Linux-Kernel/),
Networking (Https://Olegkutkov.Me/Category/Networking/),
Reverse Engineering (Https://Olegkutkov.Me/Category/Reverse-Engineering/)

Analysis and reverse-engineering of the original Starlink router

Oleg Kutkov (Https://Olegkutkov.Me/Author/Oleg_kutkov/) / December 25, 2021

While waiting for my Dishy, I decided to find and buy the Starlink router separately.
Sure, it might be just a WiFi router, but it was very curious what's inside.
Spoiler: there are some interesting implementation details.
Lucky enough, I found the router on eBay. It's the first generation of the router. Currently, it's impossible to buy (separately) the second generation of the router since it was presented a month ago.

Honestly, I love the design of this thing. It reminds me of something from the good old sci-fi.

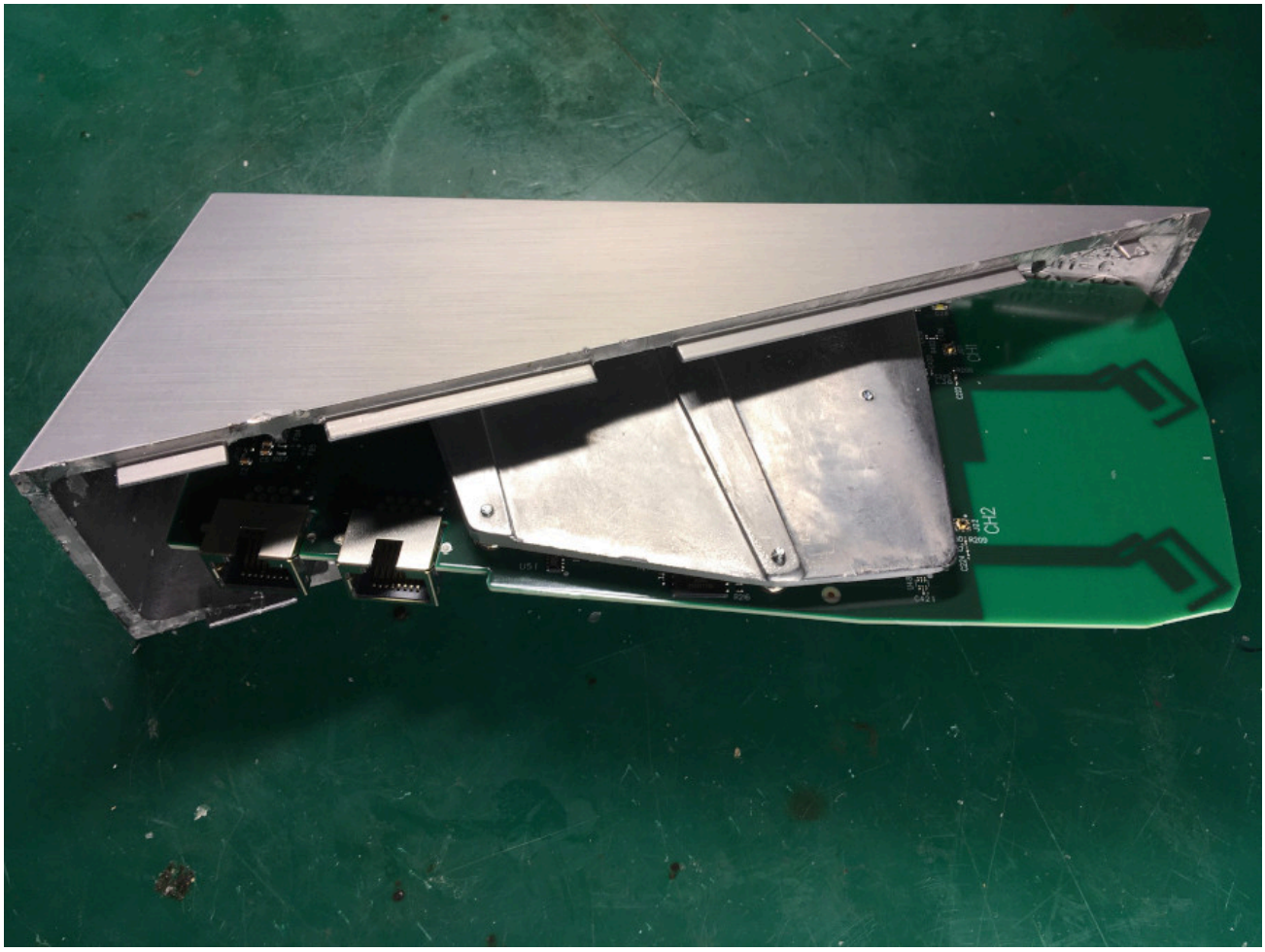
(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_header_pict.jpg)

The whole device looks rock solid and right-balanced. It’s a shame that this enclosure is not designed to be easily opened. There is no glue, but there are no screws also.

Six hooks hold metal and plastic parts of the router together, three on each side. To open this thing, you need to push the hooks somehow. This damages plastic and even aluminum. But it’s better than nothing, I guess.

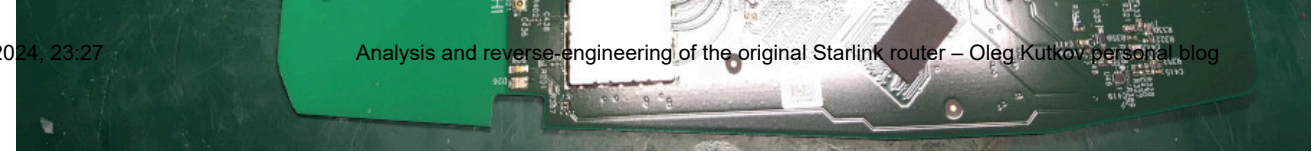
Router internals

The whole metal part of the housing works as a heatsink.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_first_open1.jpg)

A relatively simple single board.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_first_open2.jpg)

PCB

The heart of the router is a popular Qualcomm IPQ4018 (https://www.qualcomm.com/products/ipq4018) SoC: quad-core ARM Cortex A-7, 802.11ac WiFi 5GHz, and 2.4 GHz support, two channels both. Additionally, this SoC integrates a crypto engine and switch engine with hardware NAT and traffic steering.

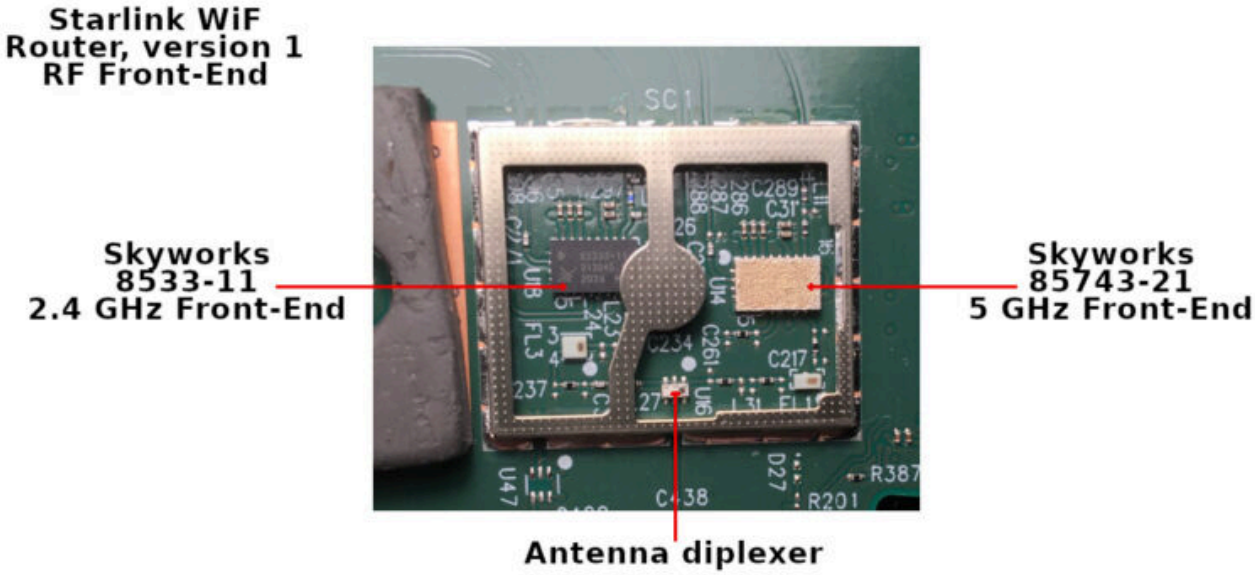
Here are all components of the board with a brief description:

TPS2378 (https://www.ti.com/lit/ds/symlink/tps2378.pdf) implements IEEE802.3at PoE standard and could handle up to 100V input.

W25N01GV (https://www.winbond.com/hq/product/code-storage-flash-memory/qspinand-flash/?__locale=en&partNo=W25N01GV) it's a serial NAND Flash IC. The router operating system is stored on this chip.

Honestly, I'm not impressed by these dual-band antennas. Sure, they can cover a small apartment or a few rooms, but nothing more.

Here is what under the RF cans covers:



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_rf_can.jpg)

(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_pcb_bottom.jpg)

The main voltage converter is LM5116 (https://www.ti.com/lit/ds/symlink/lm5116.pdf). Starlink PoE is 56 volts, but the router could run fine at 24 volts from a network switch.

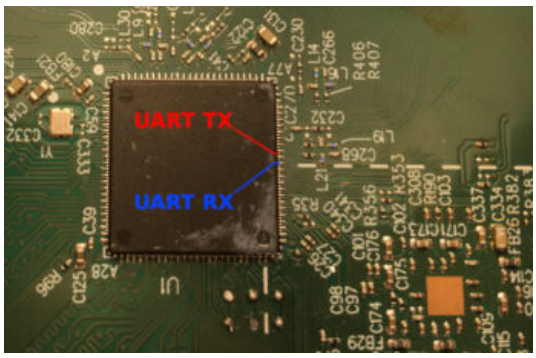
The Ethernet switch is QCA8072 (https://content.codico.com/fileadmin/media/download/datasheets/qualcomm/qualcomm_qca8072.pdf) – dual-port, 10/100/1000 Mbps tri-speed Ethernet PHY. IPQ807x it's a typical solution for the IPQ40xx platform.

GD25Q128B (https://datasheetspdf.com/pdf/796156/GigaDevice/GD25Q128B/1) it's an SPI NOR flash with Qualcomm bootloader, u-boot, and some additional data.

The most interesting part is the STSAFE-A (https://www.st.com/content/st_com/en/products/embedded-software/secure-mcu-software/stsw-stsa110-ssl.html) chip. This special MCU provides secure storage, authentication, and some cryptographic functions, fully supported by the OpenSSL. The MCU is used to store board configuration and certificates. I'll cover this below.

Recreated router block diagram:

(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_block_diagram3.png)



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_uart_pins.jpg)Unfortunately, I couldn't trace the UART interface. I know where is the corresponding CPU pins, but it looks like those pins are not used for the serial console. Plus TX pin is almost buried under the CPU, and it's tough to get to it.

There are a few test points on the board. Some of them related to the NAND and NOR. Some are voltages control and so on.
I checked all the test pads and got nothing. Anyway, it's not a big deal.

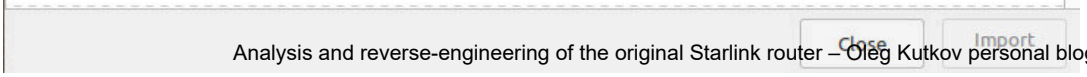
Firmware

Both NOR and NAND were removed from the board and dumped. All the data below were found with binwalk (https://github.com/ReFirmLabs/binwalk) and hex editor.
It's not a secret that the router firmware is based on the OpenWRT. Here is the official SpaceX repository: https://github.com/SpaceExplorationTechnologies/starlink-wifi (https://github.com/SpaceExplorationTechnologies/starlink-wifi)
Sure, the GitHub repository contains only GPL code without any SpaceX proprietary components that drive the router.

UBOOT_0	0x190000	u-boot ELF binary 0
UENV_1	0x290000	u-boot environment variables 1
UBOOT_1	0x2D0000	u-boot ELF binary 1

Trust Zone firmware protects the boot process. This means that only a “valid” signed bootloader and Linux kernel are allowed. Each bootloader contains a security certificate. Those certificates could be extracted with a simple `dd` command. The u-boot certificate, for example:

```
dd if=GD25Q128B@WSON8.BIN of=cert0.der bs=1 skip=4002248 count=1165
```



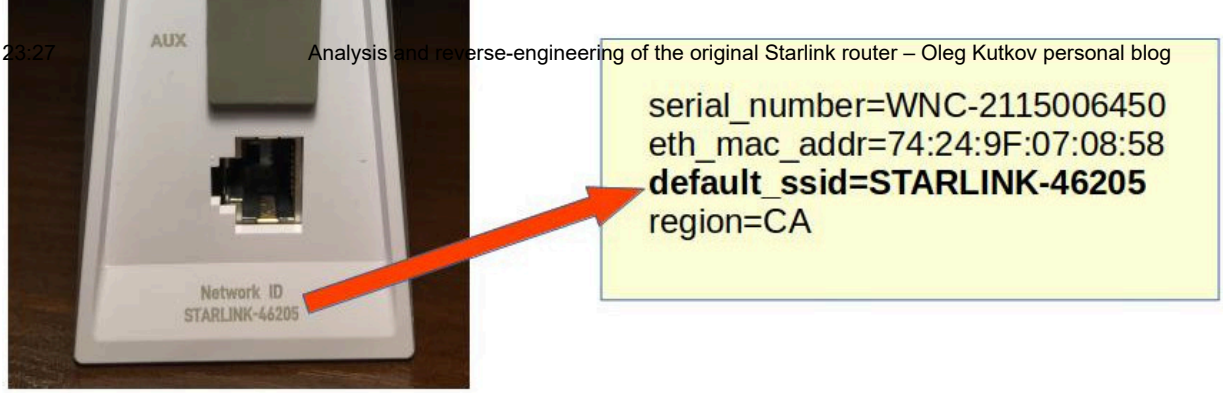
(https://olegkutkov.me/wp-content/uploads/2021/12/u-boot-cert.jpg)

Also, there are multiple u-boot binaries with separate environments. It could be done for redundancy or different modes. I'm not sure.
The u-boot sources are here: https://github.com/SpaceExplorationTechnologies/starlink-wifi/tree/master/qca/src/uboot-1.0 (https://github.com/SpaceExplorationTechnologies/starlink-wifi/tree/master/qca/src/uboot-1.0)

It seems that the first u-boot uses some default environment. Env variables at offset **0x180000**:

```
baudrate=115200
bootcmd=bootipq
bootdelay=2
ipaddr=192.168.1.11
burn-fail=0
burn-in=0
count=0
```

Also, this bootloader loads the default environment as defined there (https://github.com/SpaceExplorationTechnologies/starlink-wifi/blob/master/qca/src/uboot-1.0/common/sysinfo_common.c#L58).

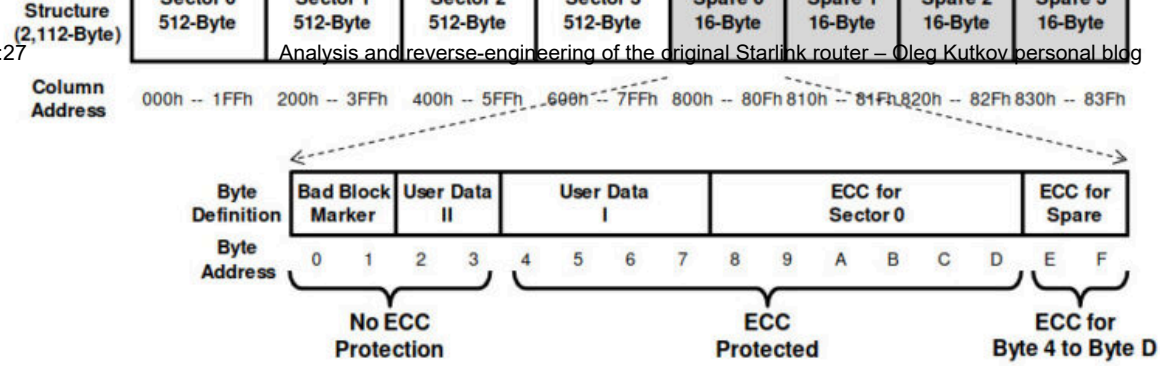


(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_network_id_sysconf.jpg)

Additionally, the NOR flash contains WiFi calibration data, a.k.a. “ART” partition. The QCA WiFi driver uses this data; I’m not covering this in the current article.

NAND image

The NAND flash contains the Linux kernel image and rootfs. But it’s somewhat tricky to get a usable image of the NAND.
OOB (Out Of Band) data – it’s a spare area adjacent to a page of data. Generally, OOB exists in NAND Flash to enable ECC (Error Correction Code) and bad block management.
Let’s check the W25N01GV (https://www.winbond.com/resource-files/W25N01GV%20Rev%20Q%20051721.pdf) NAND datasheet.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_nand_page.jpg)

This means that each 2048 data page contains an additional tail of 64 bytes. Those OOB chunks should be separated from actual data and processed correctly.
Initially, I tried to use this Python script:

```
        print "reading block %d page %d: offset %08X" % (blkno, pgno, fileoff)

        inf.seek(fileoff, 0)
        block = inf.read(blklen)
        buf = ""

        for i in range(NAND_PAGE_SIZE):
            buf += block[i]

       return s

    for blkn in range(1024):
        for pagen in range(64):
            res = read_page(inf, blkn, pagen)
            of.write(res)
```

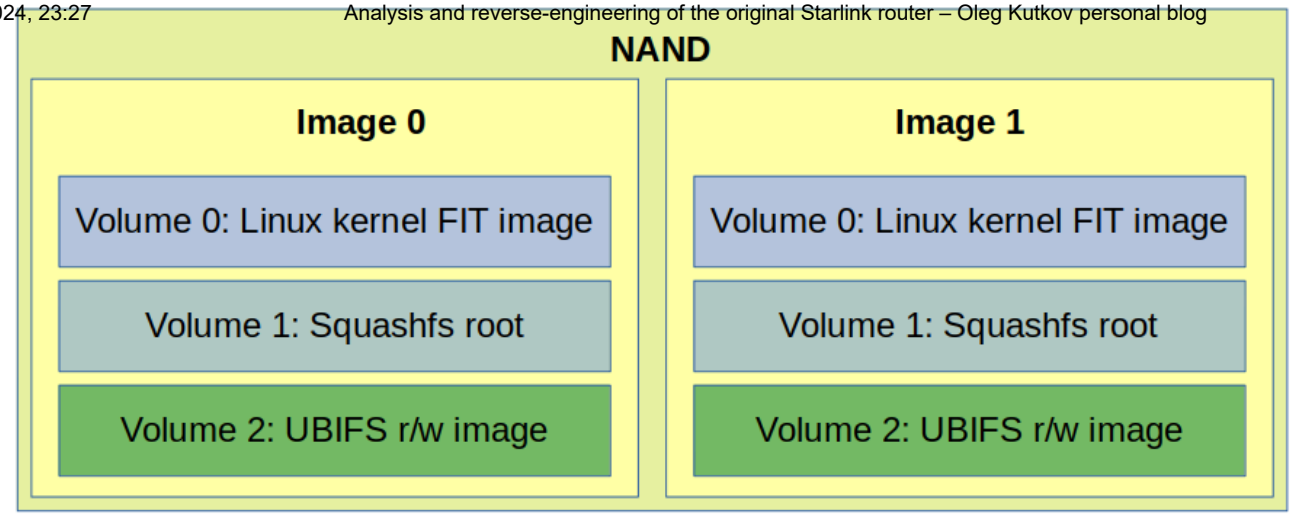
Page size, sectors count, and block count were taken from the datasheet.

Run and test result:

```
$ python oob_strip.py W25N01GV@WSON8.BIN test_nand_out.bin

$ file test_out.bin
test_out.bin: UBI image, version 1
```

```
        Volume Name:ubi_rootfs
        Volume Name:rootfs_data
        PEB Range: 512 - 1023

        Volume: kernel
        ----------------------
        Volume: ubi_rootfs
        ----------------------
        Volume: rootfs_data
        ----------------------
******
    Image: 1899964099
    ----------------------
        Image Sequence Num: 1899964099
        Volume Name:kernel
        Volume Name:ubi_rootfs
        Volume Name:rootfs_data
        PEB Range: 0 - 511

        Volume: kernel
        ----------------------
        Volume: ubi_rootfs
        ----------------------
        Volume: rootfs_data
        ----------------------
```

Great result. Here we have two images with three volumes in each image.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_nand_layout1.png)
Two identical images are used for redundancy and easy firmware upgrade procedure.

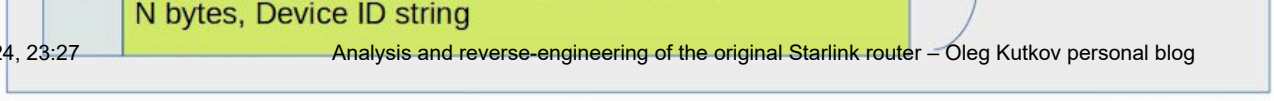
Volume 0 contains the Linux kernel FIT image (https://www.thegoodpenguin.co.uk/blog/u-boot-fit-image-overview/) with signature.
Volume 1 it's a squashfs (https://tldp.org/HOWTO/SquashFS-HOWTO/whatis.html) rootfs image with OpenWRT operating system and SpaceX software.
Volume 2 is quite interesting. It's an r/w ubifs (http://www.linux-mtd.infradead.org/doc/ubifs.html) image with the router configuration files:

```
│   │   ├── thermal
│   │   ├── ubootenv
│   │   ├── upnpd
│   │   ├── WifiConfig
│   │   └── wireless
│   ├── conntrackd
│   │   └── conntrackd.conf
│   ├── crontabs
│   │   └── root
│   ├── dnsmasq.conf
│   ├── dropbear
│   │   ├── authorized_keys
│   │   └── dropbear_rsa_host_key

***
```

This volume is part of the OpenWrt Extroot (https://openwrt.org/docs/guide-user/additional-software/extroot_configuration) and mounted as /overlay (https://en.wikipedia.org/wiki/OverlayFS)

/etc/config/WifiConfig it's a binary file used by the SpaceX proprietary software.
It's interesting that this file is protected by a set of permissions. No one can read it except the owner.

Sure, the file format is unknown, but it looks quite straightforward.
This is what I figured out:
This file is used to generate OpenWrt uci wireless config (https://openwrt.org/docs/guide-user/network/wifi/basic) at runtime.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_wifi_config_binary_format_fix.jpg)

Kernel

The **FIT image** contains multiple device tree blobs, Linux kernel, and Starlink Router Attest Certs.

```
4003052 0x3D14EC device tree image (dtb)
4033728 0x3D8CC0 device tree image (dtb)
4058660 0x3DEE24 device tree image (dtb)
4083428 0x3E4EE4 Certificate in DER format (x509 v3), header length: 4, se
4084593 0x3E5371 Certificate in DER format (x509 v3), header length: 4, se
4085544 0x3E5728 Certificate in DER format (x509 v3), header length: 4, se
4089868 0x3E680C Certificate in DER format (x509 v3), header length: 4, se
4091033 0x3E6C99 Certificate in DER format (x509 v3), header length: 4, se
4091984 0x3E7050 Certificate in DER format (x509 v3), header length: 4, se
```

The secure boot chain uses those certs to verify the validity of the kernel.

Why are there so many dtb sections? The single platform requires a single dtb. Those dtb's are support for different QCA reference platforms.
It's just a typical solution. It looks like SpaceX didn't change anything in the OpenWrt build system.

The Linux kernel image is gzip-compressed, and it's elementary to extract build configuration:

```
$ binwalk kernel-starlink | grep Image -A1
268 0x10C gzip compressed data, maximum compression, has original file nam
3754412 0x3949AC device tree image (dtb)
```

Image size: **3754412 – 268** = **3754144**

```
7761920 0x767000 ELF, 32-bit LSB shared object, ARM, version 1 (SYSV)
```

The first one at **0x5AC888**:

```
dd if=linux_image of=kernel_config.gz bs=1 skip=5949576 count=224895

gzip -d kernel_config.gz

$ head -n10 kernel_config
#
# Automatically generated file; DO NOT EDIT.
# Linux/arm 4.4.60 Kernel Configuration
#
CONFIG_ARM=y
CONFIG_ARM_HAS_SG_CHAIN=y
CONFIG_MIGHT_HAVE_PCI=y
CONFIG_SYS_SUPPORTS_APM_EMULATION=y
CONFIG_HAVE_PROC_CPU=y
CONFIG_STACKTRACE_SUPPORT=y
```

Linux kernel 4.4.60!

```
        # Pet every 10 seconds for 20 days (1,728,000 seconds)
        for var in `seq 1 172800`; do echo 1 ; sleep 10; done > /dev/watchdog
    }
```

So, Starlink internet is down for ~1 minute every 20 days? 🙄

But the most interesting binary is `/usr/sbin/wifi_control`

It's a giant application that controls everything, the whole router operation. This app is written in Go (https://go.dev/), and it's really hard to disassemble.
I can tell that wifi_control is responsible for (at least):

1. LAN configuration
2. iptables firewall configuration
3. WiFi configuration
4. Thermal management
5. Stats and Telemetry management
6. Captive portal
7. Interaction with mobile application

It seems that many params like IP addresses and URLs are just hardcoded in the application.

This wifi_control initially starts from `/etc/init.d/boot` script:

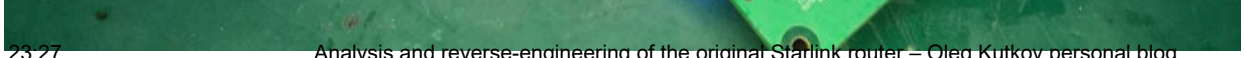
```
wifi_control --board=$(cat /etc/board) --early_init
```

Booting up

Lack of the UART makes no fun. I decided to use 3rd-party hardware and run the Starlink router software.
It's good that ipq40xx is so popular, so there are plenty of routers (https://openwrt.org/docs/techref/targets/ipq40xx#devices_with_this_target) for experiments. I decided to use the IPQ4019 reference board AP.dk04 that I happen to have. IPQ4019 it's a *slightly* different SoC from the upper range that shares the same CPU core and basic modules. This means it's almost fully compatible and could be used.

There are two significant differences:
– 5-port Ethernet switch QCA8075 instead of 2-ports QCA8072
– Parallel NAND instead of serial.

But it's not an issue.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_dk04.jpg)

Is it possible to run Starlink's u-boot on a different board? Yes, it's possible.
But it's not so fun. Bootloader verifies flash JEDEC id and looks for security chip.

(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_uboot-_run.jpeg)
Interesting note that the first u-boot image is the 2012 version, and the second is 2016.

I decided to build the u-boot and Linux kernel from the SpaceX GitHub repository and boot with Starlink rootfs

Building the Starlink OpenWrt

The SpaceX repository (https://github.com/SpaceExplorationTechnologies/starlink-wifi) is based on quite ancient OpenWrt 15.05.1 (https://openwrt.org/releases/15.05/start) “Chaos Calmer” release.

I couldn’t compile the repo on my Linux Mint 20.2, so I decided to use Docker (https://www.docker.com/) with Ubuntu 16.04 (LTS) environment. I will not cover the installation and initial configuration of the Docker.

This is my `Dockerfile` :

```
        git \
        less \
        libjansson-dev \
        libncurses5-dev \
        libssl-dev \
        nodejs \
        python-m2crypto \
        python-minimal \
        sharutils \
        subversion \
        unzip \
        vim \
        squashfs-tools \
        wget

RUN groupadd -g $GID user && \
useradd --create-home --gid $GID --uid $UID user

WORKDIR /var/build/starlink-wifi
```

Put the Dockerfile to the Starlink repository top dir and run:

```
docker build . -t starlink-wifi-build -f Dockerfile
```

Run the Docker:

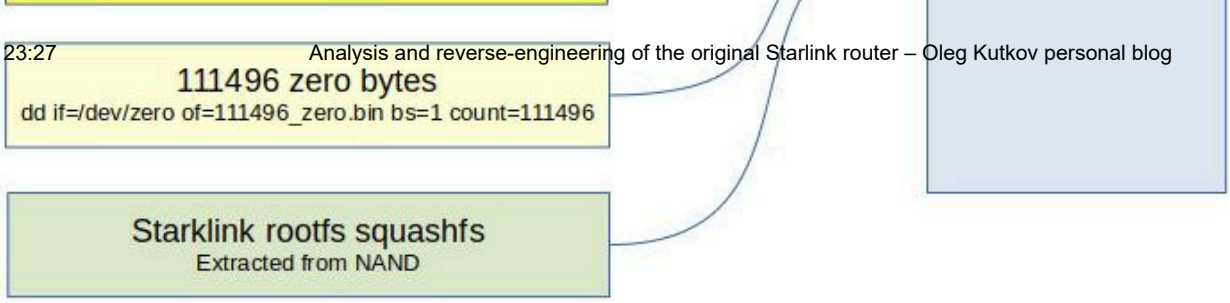
Run `make menuconfig` and go to `Boot Loaders --->`
Make sure that `uboot-ipq40xx..... U-boot for ipq40xx based platforms` is selected, and everything other is not selected.

```
< > fwupgrade-tools.................. U-boot images tools (dumpimage, mkimage)
< > lk-ipq40xx..................... lk bootloader for IPQ40xx based platforms
< > lk-ipq806x..................... lk bootloader for IPQ806x based platforms
< > sysupgrade-helper.............. U-boot images tools (dumpimage, mkimage)
< > uboot-2016-ipq40xx..................... U-boot for IPQ40xx based platforms
< > uboot-2016-ipq40xx_standard
< > uboot-2016-ipq6018..................... U-boot for IPQ6018 based platforms
< > uboot-2016-ipq806x..................... U-boot for IPQ806x based platforms
< > uboot-2016-ipq806x_standard
< > uboot-2016-ipq807x..................... U-boot for IPQ807x based platforms
<*> uboot-ipq40xx..................... U-boot for ipq40xx  based platforms
< > uboot-ipq806x......................... U-boot for IPQ806x based platforms
< > uboot-ipq806x-fwupgrade-tools... U-boot images tools (dumpimage, mkimage)
```

(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_repo_uboot_select.png)
Then:

```
cp nand_extracted_kernel_config target/linux/ipq806x/config-4.4
make V=s -jN
```

Where `N` is required, threads count. Typically, this value equals the number of the CPU cores.
If the build is failed re-run with -j1 to see the error.



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_madeup_nor.jpg)

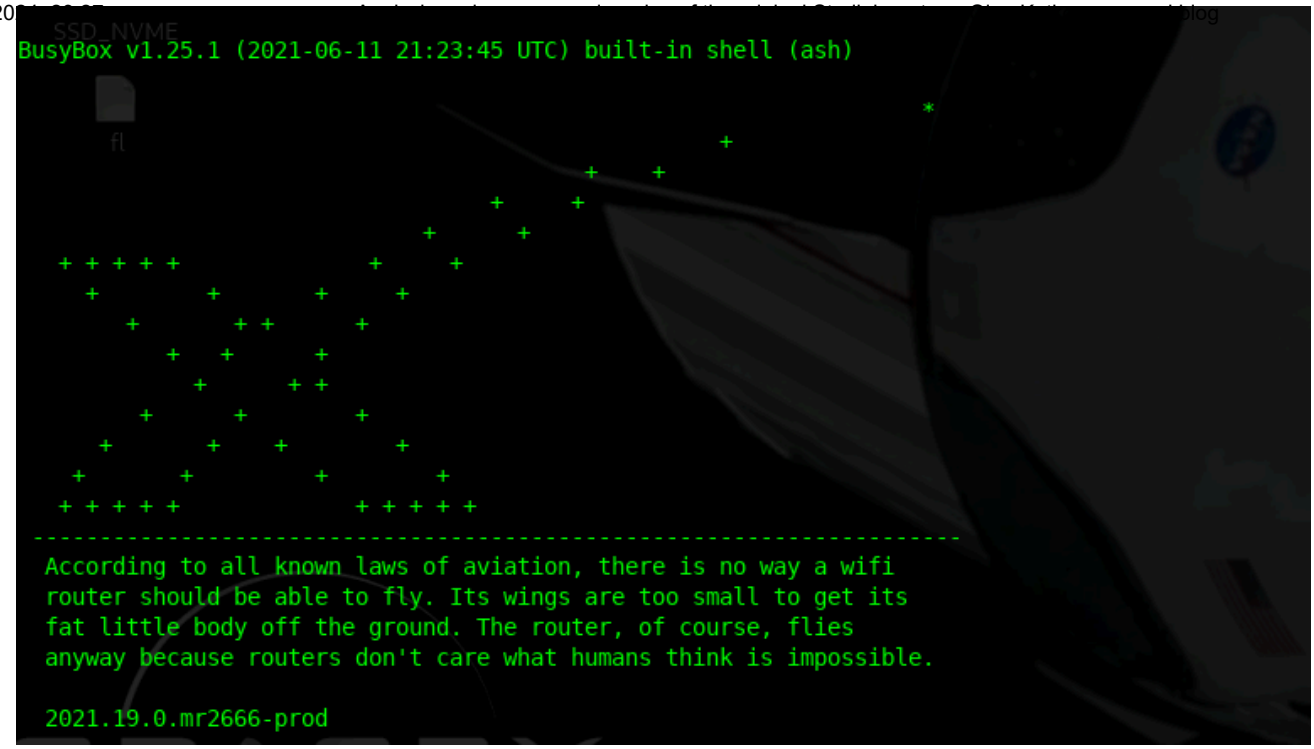
SBL and u-boot env were borrowed from the original DK04 QSDK firmware.

```
dd if=/dev/zero of=111496_zero.bin bs=1 count=111496
cat qca_bootloader_only.bin u-boot.bin 111496_zero.bin nand_extract/squash
```

There is no kernel. I decided to boot it up over the network. Why not?
I put my `openwrt-ipq806x-qcom-ipq4019-ap.dk04.1-c1-fit-uImage.itb` to a local TFTP (https://linuxhint.com/install_tftp_server_ubuntu/) server directory.

Resulting starlink_nor.bin was burned to a spare NOR flash.

After the board start, some configuration is required in the u-boot console:

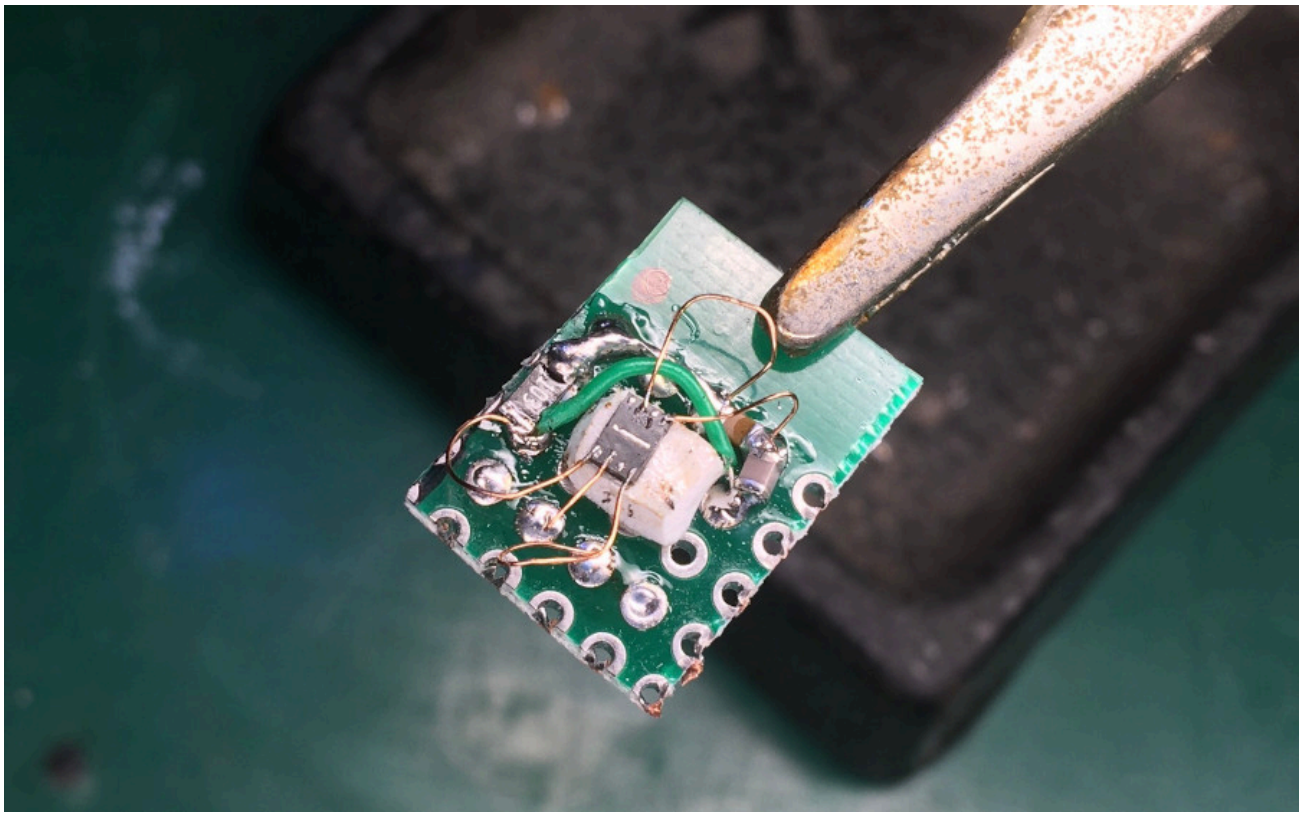


(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_console_motd.png)Yay!

It's possible to configure the network and internet access:

```
brctl addbr br-lan
brctl addif br-lan eth0
ip addr add 192.168.1.21/24 broadcast 192.168.1.255 dev eth0
ip link set dev eth0 up
ip route add default via 192.168.1.1
```

The STSAFE-A110 is in UFDFPN8 2×3 mm package. I decided to build an adapter for easy connection.



(https://olegkutkov.me/wp-content/uploads/2021/12/STSAFE-A110_adapter.jpg)

Also, there is a minimum required set of components: bypass capacitor and Reset circuit.

(https://olegkutkov.me/wp-content/uploads/2021/12/stsafe_connected_to_raspberry.jpg)

Quick test:

Download and extract the package. Then go to STSAFE-A_OpenSSL_Engine_V1.2.0

Open make.in and edit OpenSSL paths. In the case of a standard SSL installation, this should look like this:

```
OPENSSL_INC = /usr/include/openssl
OPENSSL_LIB = /usr/lib
OPENSSL_BIN = /usr/bin
```

Then run:

```
make
sudo make install
sudo ldconfig
```

Next create file `openssl.conf.stsafe` with the following content:

```
OPENSSL_CONF=./openssl.conf.stsafe openssl engine Stsafe
```

There should be a successful pairing at the command output:

```
Main : Now let's pair ...
About to call StSafeA_LocalEnvelopeKeySlotQuery: 0x2907d0, 0x2907e0, 0x290
StSafeA_LocalEnvelopeKeySlotQuery: StatusCode=0 slot 0: presence flag =1
StSafeA_LocalEnvelopeKeySlotQuery: StatusCode=0 slot 1: presence flag =1
StatusCode=0 ---HostKeySlot = 0xbef11ccc, pStSafeA->InOutBuffer.LV.Data =
HostKeySlot->HostKeyPresenceFlag: 1
Main : stsafe_pairing success
```

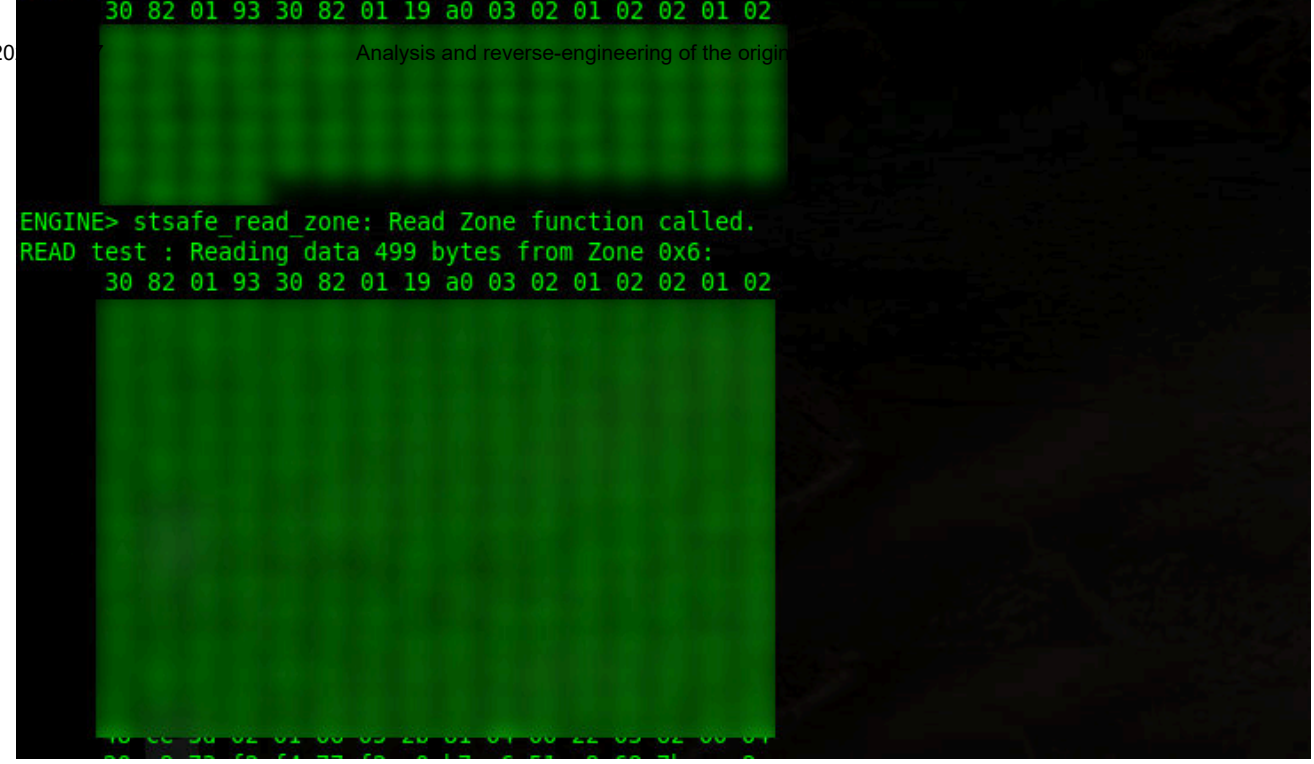
The next step is to run the `test suit` to get some useful output.

```
cd Examples/stsafe_engine_test_suite/
```

Please note that some tests might fail. I commented those failed tests in `test_stsafe_engine.c`

Build and run the tool:

```
make
./test_stsafe_engine
```



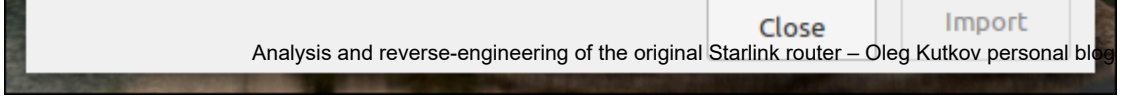
(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_stsafe_test_suit.jpg)

Sorry for the blurred parts 🙄

Those hex data could be copied to a text file and then converted to a valid `pem` file:

```
cat device_cert.txt | xxd -r -p | openssl x509 -inform DER -out device_cer
```

Result:



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_dev_pem.png)

What's about `wifi_control` ? Let's run it on Raspberry!
Is it even possible? Sure, both Starlink router and Raspberry Pi use the "same" 32-bit ARM CPU core.
Plus, the Go binaries are pretty self-contain since there is built-in runtime.

Make some preparations first.

```
# echo "v1 > /etc/board
# echo "2021.19.0.mr2666-prod" > /etc/version    # or version of your firm
# ln -s /bin/true /usr/local/bin/devinfo_access
```

Copy `WifiConfig` from the real system to your Raspberry `/etc/`

Raspberry Pi uses the second I2C bus /dev/i2c-1
But wifi_control is looking for /dev/i2c-0
It's easy to fix with udev alias. Create `/etc/udev/rules.d/99_sr0.rules` with the following content:
`KERNEL=="i2c-1", SYMLINK+="i2c-0"`

Then run `sudo udevadm control --reload-rules; sudo udevadm trigger` .
Now we are good to go.

It looks like gRPC (https://grpc.io/), and protobuf it's the primary API/communication protocol (https://hackaday.com/2021/12/17/this-week-in-security-log4j-pdf-cpu-and-i-hack-starlink/) in the Starlink world.

Now I'm trying to reverse the I2C communication protocol to understand how wifi_control interacts with STSAFE. There is a lot of data traffic on this bus.
Also, I would like to figure out how all those certs are working. And how to communicate with the Starlink servers. It would be great to request a firmware update file or something.

Stay tuned!

About Oleg Kutkov

View all posts by Oleg Kutkov → (https://olegkutkov.me/author/oleg_kutkov/)

HackRF SuperCluster (https://olegkutkov.me/2021/11/29/hackrf-supercluster/)

Reverse engineering of the Starlink Ethernet adapter (https://olegkutkov.me/2022/03/07/reverse-engineering-of-the-starlink-ethernet-adapter/)

10 thoughts on “Analysis and reverse-engineering of the original Starlink router”

someone says:

January 4, 2022 at 12:34 am (https://olegkutkov.me/2021/12/25/analysis-and-reverse-engineering-of-the-original-starlink-router/#comment-1103)

https://www.esat.kuleuven.be/cosic/blog/dumping-and-extracting-the-spacex-starlink-user-terminal-firmware/ (https://www.esat.kuleuven.be/cosic/blog/dumping-and-extracting-the-spacex-starlink-user-terminal-firmware/)

Pingback: Ukraine engineer talks testing SpaceX’s new Starlink service - Verve times (https://vervetimes.com/ukraine-engineer-talks-testing-spacexs-new-starlink-service/)

Pingback: Ukraine engineer talks testing SpaceX’s new Starlink service - Reporter Door (https://reporterdoor.com/2022/03/01/ukraine-engineer-talks-testing-spacexs-new-starlink-service/)

Pingback: Ukraine engineer talks testing SpaceX’s new Starlink service - GamesVot (https://gamesvot.com/2022/03/01/ukraine-engineer-talks-testing-spacexs-new-starlink-service/)

Pingback: Ukraine engineer talks testing SpaceX’s new Starlink service - Tech Scurry - The Latest Tech News Of The World (https://techscurry.com/general/ukraine-engineer-talks-testing-spacexs-new-starlink-service/46711/)

Pingback: Ukraine engineer talks testing SpaceX’s new Starlink service (https://silverrtech.com/ukraine-engineer-talks-testing-spacexs-new-starlink-service/)

Pingback: Initial analysis of the Starlink router gen2 – Oleg Kutkov personal blog (https://olegkutkov.me/2022/03/01/initial-analysis-of-the-starlink-router-gen2/)

Leave a Reply

Your email address will not be published. Required fields are marked *

Post Comment

CATEGORIES

dc-dc-converters/)

Reworking Gen2 Starlink router from SPX connector to 8P8C (RJ45) (https://olegkutkov.me/2023/03/18/reworking-gen2-starlink-router-from-spx-connector-to-8p8c-rj45/)

Data logger for UNI-T UT800 multimeters (https://olegkutkov.me/2022/07/18/data-logger-for-uni-t-ut800-multimeters/)

RECENT COMMENTS

Ed on How to power Starlink terminal from 12V without PoE injector and DC/DC converters (https://olegkutkov.me/2023/06/09/how-to-power-starlink-terminal-from-12v-without-poe-injector-and-dc-dc-converters/#comment-4809)

Oleg Kutkov (https://olegkutkov.me) on How to power Starlink terminal from 12V without PoE injector and DC/DC converters (https://olegkutkov.me/2023/06/09/how-to-power-starlink-terminal-from-12v-without-poe-injector-and-dc-dc-converters/#comment-4783)

Steven on How to power Starlink terminal from 12V without PoE injector and DC/DC converters (https://olegkutkov.me/2023/06/09/how-to-power-starlink-terminal-from-12v-without-poe-injector-and-dc-dc-converters/#comment-4775)

Oleg Kutkov (https://olegkutkov.me) on Reverse engineering of the Starlink Ethernet adapter (https://olegkutkov.me/2022/03/07/reverse-engineering-of-the-starlink-ethernet-adapter/#comment-4747)

SITE SEARCH

Search … Search

Support author

You can send donations on PayPal (https://www.paypal.com/myaccount/transfer/homepage) using my email contact@olegkutkov.me

Thank you for your support!

QCA8072 Dual-Port 10/100/1000 Mbps Ethernet Transceiver

QCA8072 Product Overview

The QCA8072 Ethernet transceiver is a dual-port, 10/100/1000 Mbps tri-speed Ethernet PHY. The QCA8072 Ethernet transceiver provides physical layer functions for half/full-duplex 10BASE-Te, 100BASE-TX, and full-duplex 1000BASE-T Ethernet to transmit and receive data over standard Category 5 (CAT-5) unshielded twisted pair cable.

The QCA8072 includes two SerDes. One can be configured to PSGMII or QSGMII for connection with MAC. The other can be configured to SGMII for connection with MAC or fiber port combined with copper port1 to form a combo port.

The QCA8072 Ethernet transceiver integrates Green ETHOS® power saving technologies which significantly save power in both active operation and idle condition. Green ETHOS® power saving schemes include ultra-low power in cable unplugged mode or port power down mode, as well as automatically optimized power saving based on cable length. The QCA8072 Ethernet transceiver supports standard IEEE 802.3az Energy Efficient Ethernet (EEE) and furthermore the Wake-on-LAN (WoL) feature to manage and regulate total system power requirements.

The key features of the IEEE 802.3az standard include:

- 10BASE-Te: Reduced transmit amplitude
- 100BASE-TX and 1000BASE-T: Low Power Idle (LPI) mode to turn off unused analog and digital blocks to save power when data traffic is idle

The QCA8072 Ethernet transceiver embeds Cable Diagnostics Test (CDT) technology for measuring cable length, detecting the cable status, and identifying remote and local PHY malfunctions, bad or marginal patch cord segments or connectors.

The QCA8072 Ethernet transceiver requires only a single 3.3 V power supply. Embedded regulators are used to generate other required voltages.

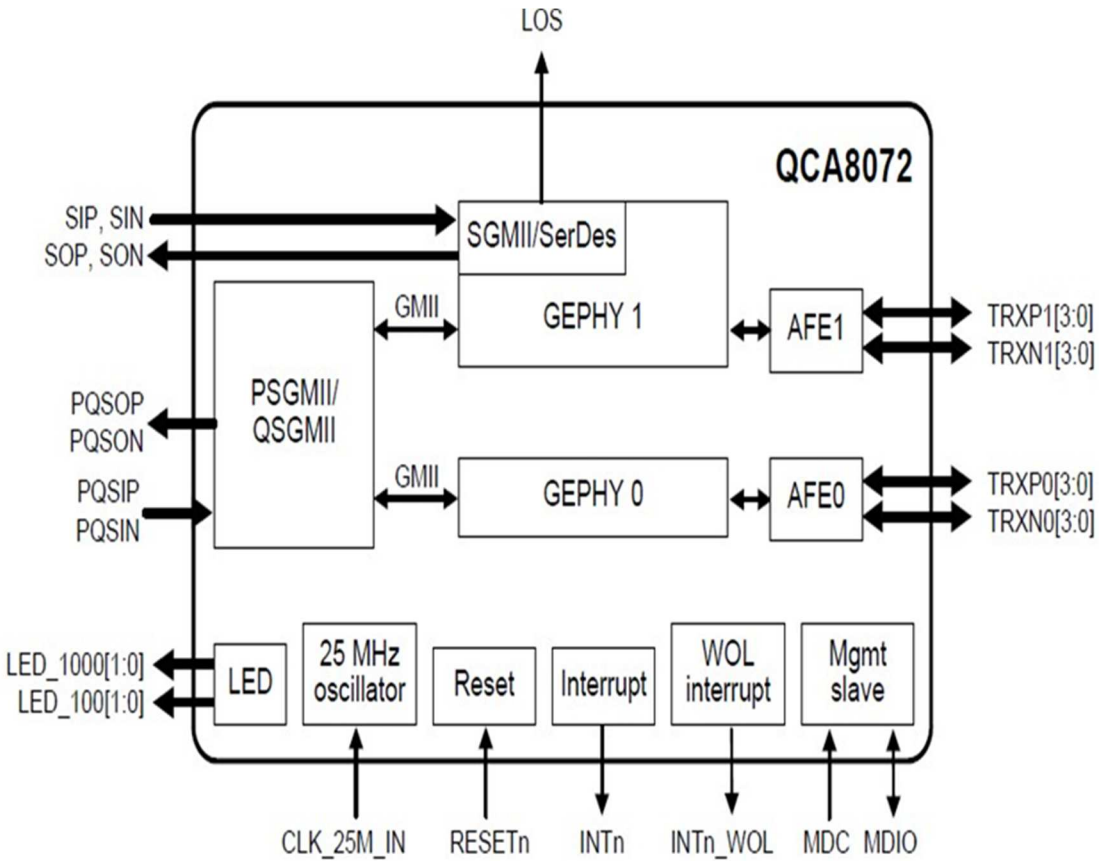
Solution Highlights & Specifications

- Supports three working modes with power-on strapping configuration:
  - Two 1000BASE-T/100BASE-TX/10BASE-Te ports with PSGMII to MAC
  - One 1000BASE-T/100BASE-TX/10BASE-Te port and one combo port (fiber/copper)with PSGMII to MAC
  - Two 1000BASE-T/100BASE-TX/10BASE-Te ports with QSGMII and SGMII to MAC
- The combo port supports auto media detection with programmable priority
- The fiber port supports 1000BASE-X/100BASE-FX
- Management interface (MDIO) supports broadcast write
- CRC checker and packet counter
- Green ETHOS® power saving modes:
  - Automatic power saving at media disconnected state
  - Automatic power saving per cable length
  - Software power down
- IEEE 802.3az EEE
- Wake-on-LAN (WoL) to detect magic packet and notify the sleeping system to wake up
- Fully integrated digital adaptive equalizers, echo cancellers, and Near End Crosstalk (NEXT)

cancellers

- Robust Contact Electro-Static Discharge (ESD) protection without external protection circuit
- Robust Lightening Surge performance without external protection circuit
- Robust operation over up to 140 meters of CAT5e cable
- Automatic Channel Swap (ACS)
- Automatic MDI/MDIX crossover
- Automatic polarity correction
- IEEE 802.3u compliant auto-negotiation
- Jumbo frame support up to 9 KB (full-duplex)
- Management interface supports 1.5/1.8/2.5/3.3 V I/O voltage.
- 25 MHz single-ended clock input
- Multiple loopback modes for diagnostics
- Cable Diagnostic Test (CDT)
- Single power supply: 3.3 V, optional for internal switch regulator or external regulator for core voltage
- 9 mm × 9 mm, 108-pin DR-QFN package
- Industry temperature option available
- Heatsink-free design for commercial temperature part

QCA8072 System Architecture



Qualcomm ETHOS

Qualcomm ETHOS technologies provide customers with industry-leading low-power and solution size to enable Fast or Gigabit Ethernet connectivity in networking equipment, consumer electronics and computing platforms. Qualcomm PHY, controller and switch solutions support the lEEE 802.3az standard for Energy Efficient Ethernet, to extend battery-charges on computing platforms and deliver power-efficiencies in networking equipment. Qualcomm also enables incremental power-saving techniques to offer our customers the very lowest power Ethernet in the industry today. The unmatched efficiency and advanced carrier-class features of Qualcomm ETHOS solutions give customers a competitive edge when designing products for energy-conscious consumers and businesses.

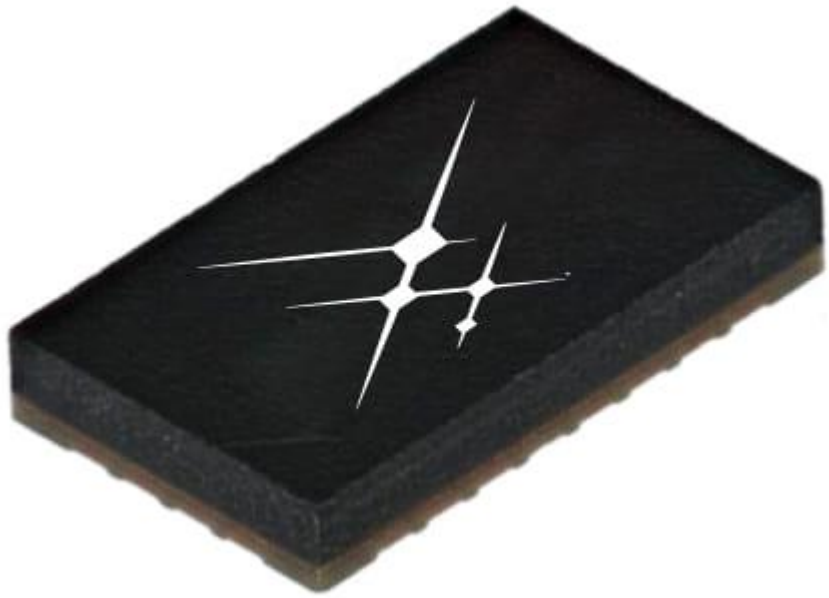
https://www.skyworksinc.com/en/Products/Front-end-Modules/SKY85333-11

SKY85333-11

2.4 GHz WLAN Front-End Module

The SKY85333-11 is a highly integrated, 2.4 GHz front-end module (FEM) incorporating a 2.4 GHz single-pole, double-throw (SPDT) transmit/receive (T/R) switch, a 2.4 GHz high-gain lownoise amplifier (LNA) with bypass, and a 2.4 GHz power amplifier (PA) intended for high-power 802.11ax applications and systems.

The LNA and PA disable functions ensure low leakage current in the off mode. An integrated logarithmic power detector is included to provide closed-loop power control over 25 dB of dynamic range. A digital pre-distortion (DPD) coupler is provided to further improve



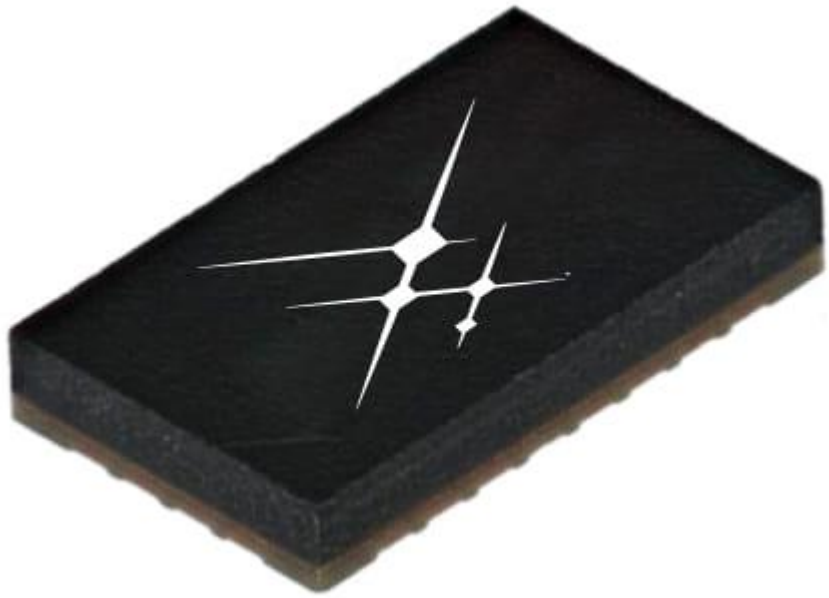
https://www.skyworksinc.com/en/Products/Front-end-Modules/SKY85333-11

SKY85333-11

2.4 GHz WLAN Front-End Module

The SKY85333-11 is a highly integrated, 2.4 GHz front-end module (FEM) incorporating a 2.4 GHz single-pole, double-throw (SPDT) transmit/receive (T/R) switch, a 2.4 GHz high-gain lownoise amplifier (LNA) with bypass, and a 2.4 GHz power amplifier (PA) intended for high-power 802.11ax applications and systems.

The LNA and PA disable functions ensure low leakage current in the off mode. An integrated logarithmic power detector is included to provide closed-loop power control over 25 dB of dynamic range. A digital pre-distortion (DPD) coupler is provided to further improve



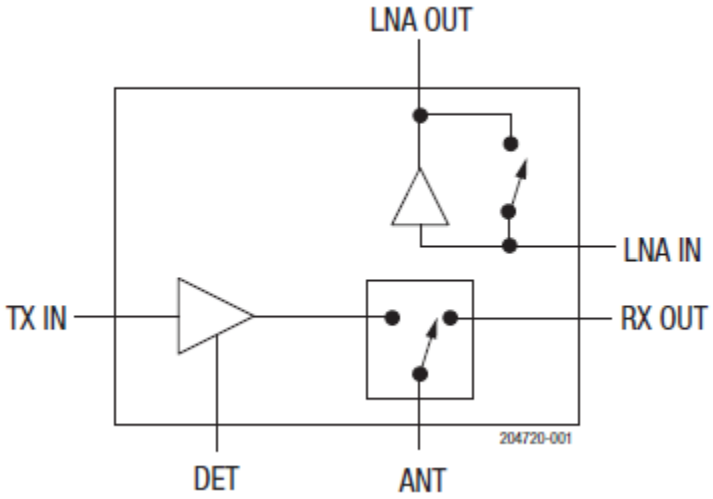
SKY85743-21

5 GHz High-Power WLAN Front-End Module

The SKY85743-21 is a highly integrated, 5 GHz front-end module (FEM) incorporating a 5 GHz single-pole, double-throw (SPDT) transmit/receive (T/R) switch, a 5 GHz high-gain low-noise amplifier (LNA) with bypass, and a 5 GHz power amplifier (PA) intended for high-power 802.11ax applications and systems.

The LNA and PA disable functions ensure low leakage current in the off mode. An

integrated logarithmic power detector is included to provide closed-loop power control over 20 dB of dynamic range.



A diplexer is used for combining (and separating) satellite and terrestrial (antenna) signals. Typically you would use one diplexer at the satellite dish and antenna on the roof to combine the signals together. These combined signals travel down a single coaxial cable to the wall socket where a second diplexer is used to separate the signals and voltage to 2 separate cables. One for satellite and the other for the antenna. Very useful when it is near impossible to run a second cable for the satellite dish. Simply share the antenna cable for both.

LM5116

ACTIVE

6-100V Wide Vin, Current Mode Synchronous Buck Controller

alarm

Notifications**Order now**

DATA SHEET

-

document-pdfAcrobat

LM5116 Wide Range Synchronous Buck Controller datasheet (Rev. I)PDF|HTML

A newer version of this product is available

open-in-new

Compare alternates

Same functionality with different pin-out to the compared device

LM5146

ACTIVE

100-V synchronous buck DC/DC controller with wide duty-cycle range

Smaller device footprint optimized for EMI requirements.

https://content.codico.com/fileadmin/media/download/datasheets/qualcomm/qualcomm_qca8072.pdf

STSAFE-A100

NRND

Save to MyST

Authentication & Brand protection secure solution

Download datasheet

Overview

Sample & Buy

Documentation

CAD Resources

Tools & Software

Quality & Reliability

Product overview

Description

All features

You might also...

Recommended for you

Recommended replacement product: STSAFE-A110

Description

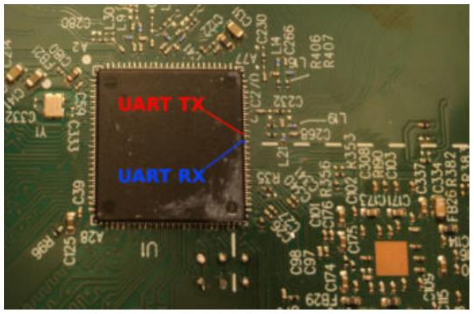
The STSAFE-A100 is a highly secure solution that acts as a secure element providing authentication and data management services to a local or remote host. It consists of a full turnkey solution with a secure operating system running on the latest generation of secure microcontrollers.

The STSAFE-A100 can be integrated in IoT (Internet of things) devices, smart-home, smart-city and industrial applications, consumer electronics devices, consumables

and accessories.

Reverse Engineering Starlink Wifi.pdf

Page: 7 / 40 Find:



(https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_uart_pins.jpg)Unfortunately, I couldn't trace the UART int I know where is the corresponding CPU pins, but it looks like those pins are not used for the seria console. Plus TX pin is almost buried under the CPU, and it's tough to get to it.

https://olegkutkov.me/wp-content/uploads/2021/12/starlink_router_uart_pins.jpg

There are a few test points on the board. Some of them related to the NAND and NOR. Some are voltages control and so on.
I checked all the test pads and got nothing. Anyway, it's not a big deal.

ENG US 12:01 AM 26/06/2024