BeagleY-AI

Release 1.0.20240609-wip

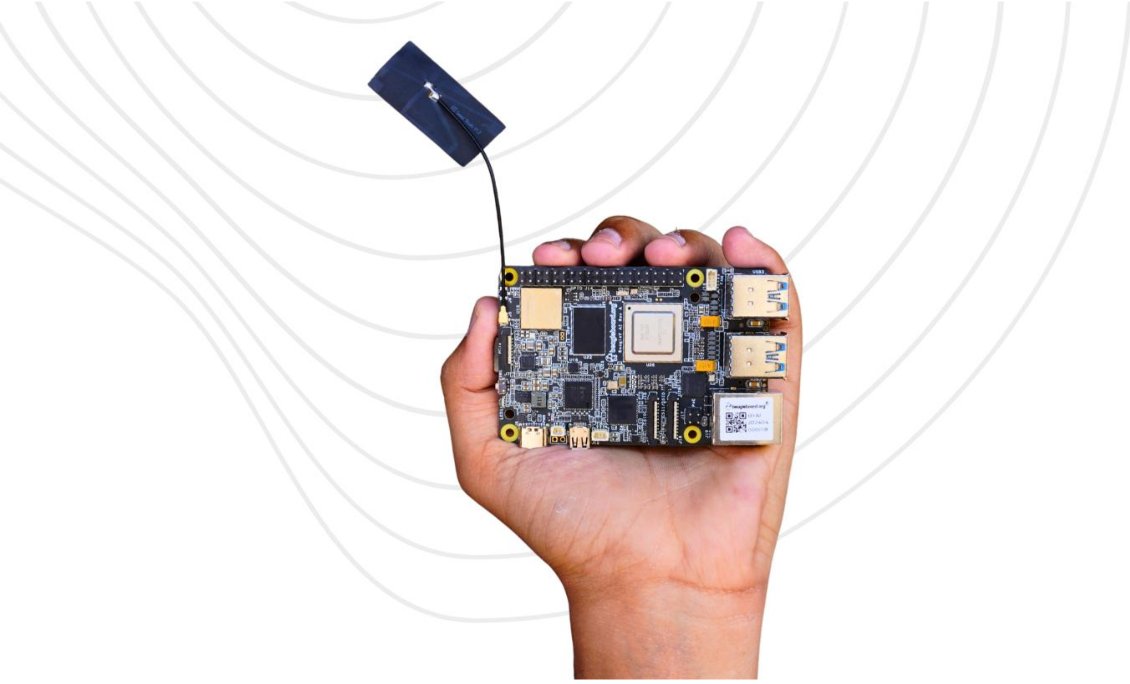
BeagleBoard.org Foundation

Jun 09, 2024

Table of contents

Work in progress

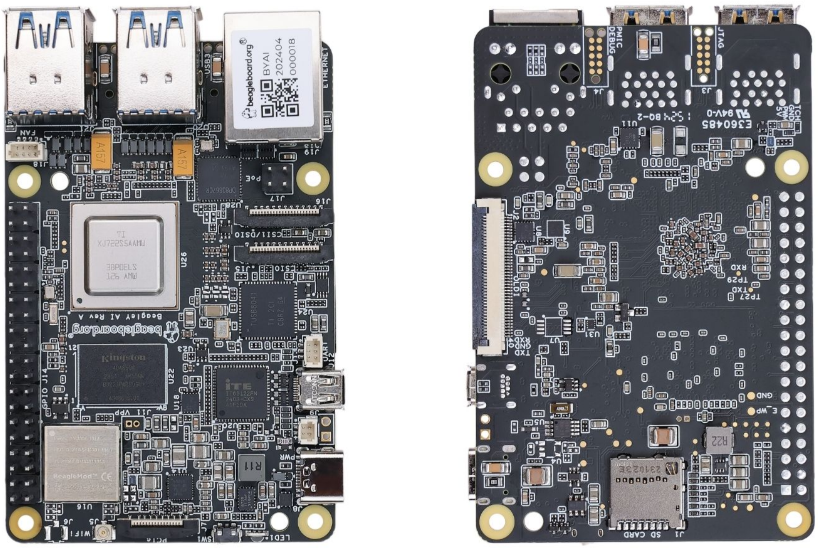
BeagleY-AI is an open-source single board computer based on the Texas Instruments AM67A Arm-based vision processor.



Chapter 1

Introduction

BeagleY-AI is an open-source single board computer designed for edge AI applications.



1.1 Detailed overview

BeagleY-AI is based on the Texas Instruments AM67A Arm-based vision processor. It features a quad-core 64-bit Arm®Cortex®-A53 CPU subsystem at 1.4GHz, Dual general-purpose C7x DSP with Matrix Multiply Accelerator (MMA) capable of 4 TOPs each, Arm Cortex-R5 subsystem for low-latency I/O and control, a 50 GFlop GPU, video and vision accelerators, and other specialized processing capability.

Table 1.1: BeagleY-AI features

Feature	Description
Processor	Texas Instruments AM67A, Quad 64-bit Arm® Cortex®-A53 @1.4 GHz, multiple cores including Arm/GPU processors, DSP, and vision/deep learning accelerators
RAM	4GB LPDDR4
Wi-Fi	Beagleboard BM3301, 802.11ax Wi-Fi
Bluetooth	Bluetooth Low Energy 5.4 (BLE)
USB Ports	4 x USB 3.0 TypeA ports supporting simultaneous 5Gbps operation, 1 x USB 2.0 TypeC, supports USB 2.0 device mode
Ethernet	Gigabit Ethernet, with PoE+ support (requires separate PoE HAT)
Camera/Display	2 x 4-lane MIPI camera connector (one connector muxed with DSI capability)
Display Output	1 x HDMI display, 1 x OLDI display, 1 x DSI MIPI Display
Real-time Clock (RTC)	Supports external coin-cell battery for power failure time retention
Debug UART	1 x 3-pin debug UART
Power	5V/3A DC power via USB-C
Power Button	On/Off included
PCIe Interface	PCI-Express® Gen3 x 1 interface for fast peripherals (requires separate M.2 HAT or other adapter)
Expansion Connector	40-pin header
Fan connector	1 x 4-pin fan connector, supports PWM control and fan speed measurement
Storage	microSD card slot with UHS-1 support
Tag Connect	1 x JTAG, 1 x External PMIC programming port

1.1.1 AM67A SoC

The AM67A scalable processor family is based on the evolutionary Jacinto™ 7 architecture, targeted at Smart Vision Camera and General Compute applications and built on extensive market knowledge accumulated over a decade of TI's leadership in the Vision processor market. The AM67A family is built for a broad set of cost-sensitive high performance compute applications in Factory Automation, Building Automation, and other markets.

Some Applications include:

- Human Machine Interface (HMI)
- Hospital patient monitoring
- Industrial PC
- Building security system
- Off-highway vehicle
- Test and measurement
- Energy storage systems
- Video Surveillance
- Machine Vision
- Industrial mobile robot (AGV/AMR)
- Front camera systems

The AM67A provides high performance compute technology for both traditional and deep learning algorithms at industry leading power/performance ratios with a high level of system integration to enable scalability and lower costs for advanced vision camera applications. Key cores include the latest Arm and GPU processors for general compute, next generation DSP with scalar and vector cores, dedicated deep learning and traditional algorithm accelerators, an integrated next generation imaging subsystem (ISP), video codec, and MCU cores. All protected by industrial-grade security hardware accelerators.

Tip: For more information about AM67A SoC you can checkout https://www.ti.com/product/AM67A

1.2 Board components location

1.2.1 Front components

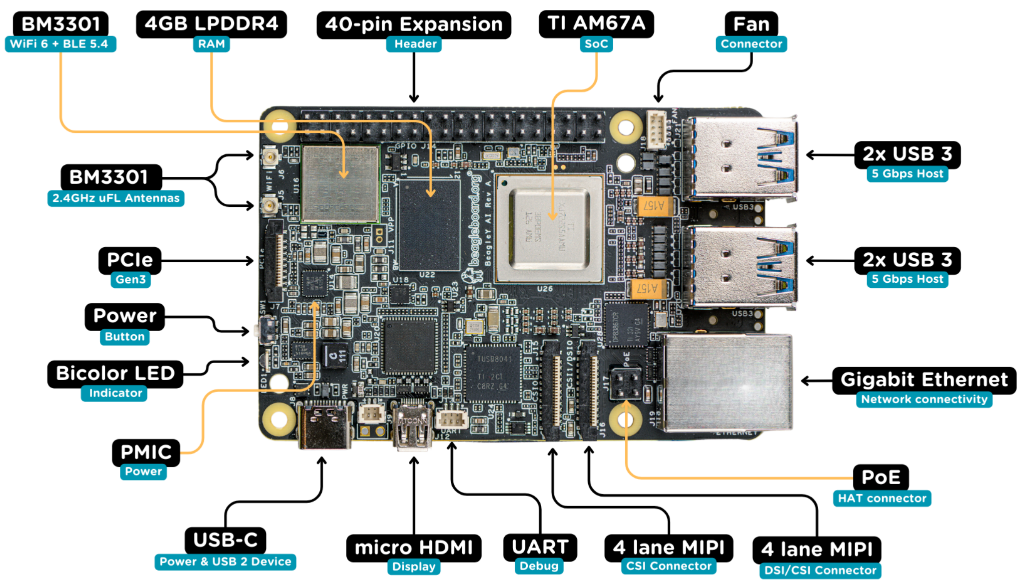


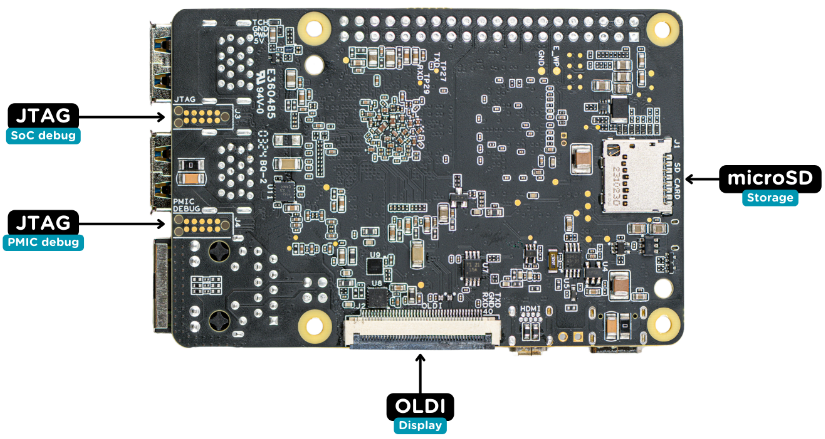
Table 1.2: BeagleY-AI board front components location

Feature	Description
WiFi/BLE	Beagleboard BM3301 with 802.11ax Wi-Fi & Bluetooth Low Energy 5.4 (BLE)
RAM	4GB LPDDR4
Expansion	40pin Expansion header compatible with HATs
SoC	TI AM67A Arm®Cortex®-A53 4 TOPS vision SoC with RGB-IR ISP for 4 cameras, machine vision, robotics, and smart HMI
Fan	4pin Fan connector
USB-A	4 x USB 3 TypeA ports supporting simultaneous 5Gbps operation host ports
Network Connectivity	Gigabit Ethernet
PoE	Power over Ethernet HAT connector
Camera/Display	1 x 4-lane MIPI camera/display transceivers, 1 x 4-lane MIPI camera
Debug UART	1 x 3-pin JST-SH 1.0mm debug UART port
Display Output	1 x HDMI display
USB-C	1 x Type-C port for power, and supports USB 2 device
PMIC	Power Management Integrated Circuit for 5V/5A DC power via USB-C with Power Delivery support
Bicolor LED	Indicator LED
Power button	ON/OFF button
PCIe	PCI-Express® Gen3 x 1 interface for fast peripherals (requires separate M.2 HAT or other adapter)

1.2.2 Back components

Table 1.3: BeagleY-AI board back components location

Feature	Description
Tag-Connect	1 x JTAG & 1 x Tag Connect for PMIC NVM Programming
Display output	1 x OLDI display
Storage	microSD card slot with support for high-speed SDR104 mode


JTAG
SoC debug
JTAG
PMIC debug
microSD
Storage
OLDI
Display

Chapter 2

BeagleY-AI Quick Start

2.1 What's included in the box?

When you purchase a BeagleY-AI, you'll get the following in the box:

1. BeagleY-AI
2. 2.4GHz antenna
3. Quick-start card

Todo: Attaching antennas instructions for BeagleY-AI

Todo: BeagleY-AI unboxing video

2.2 Getting started

To get started your BeagleY-AI you need the following:

1. 5V @ 3A power supply
2. MicromicroSD card (32GB)
3. *Boot Media (Software image)*

You may need additional accessories based on the mode of operation, you can use your BeagleY-AI in different ways.

1. *USB Tethering by directly connecting via USB type-c port*
2. *Headless connection via UART debug port*
3. *Standalone connection with Monitor and other peripherals attached*

Easiest option is to connect the board directly to your PC or Laptop using a USB type-C to type-c cable. There is only one USB type-C port on board, if you choose to use a dedicated power supply for first time setup, you may choose to access the board via any other methods listed above.

2.3 Power Supply

To power the board you can either connect it to a dedicated power supply like a mobile charger or a wall adapter that can provide 5V ≥ 3A. Checkout the docs power supply page for power supply recommendations.

Note: Instead of using a power supply or power adapter if you are using a Type-C to Type-C cable to connect the board to your laptop/PC then make sure it can supply at least 1000mA.

2.4 Boot Media (Software image)

Todo: Update this section to use latest boot media (software image) for BeagleY-AI.

Download the boot media from https://www.beagleboard.org/distros/beagley-ai-debian-xfce-12-5-2024-03-25 and flash it on a micro microSD card using using Balena Etcher following these steps:

1. Select downloaded boot media
2. Select microSD card
3. Flash!

Tip: For more detailed steps checkout the beagleboard-getting-started under support section of the documentation.

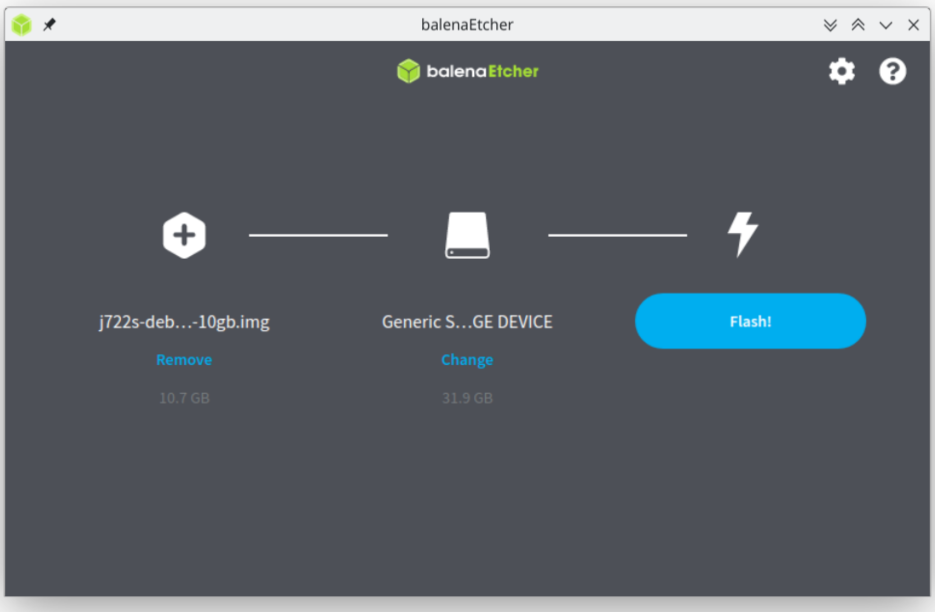


Fig. 2.1: Flashing BeagleY-AI boot image (software image) to microSD card

Once the microSD card is flashed you should see BOOT and rootfs mounted on your system as shown in image below,

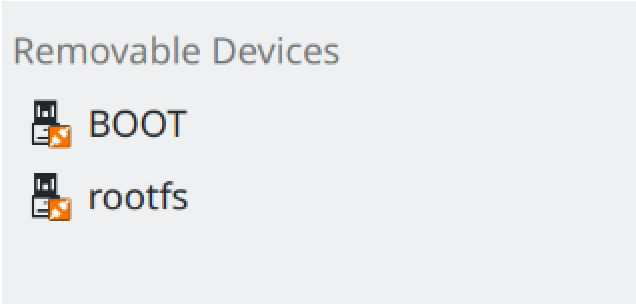


Fig. 2.2: Flashed microSD card mounted partitions

Under BOOT partition open sysconf.txt to edit login username and password.

In sysconf.txt file you have to edit the two lines highlighted below.

```
# user_name - Set a user name for the user (1000)
#user_name=beagle ①

# user_password - Set a password for user (1000)
#user_password=FooBar ②
```

① If boris is your username, update #user_name=beagle to user_name=boris

② If bash is your password, update #user_password=FooBar to user_password=bash

Note: Make sure to remove # from in front of these lines else the lines will still be interpreted like a comment and your username & password will not be updated.

Once username and password are updated, you can insert the microSD card into your BeagleY-AI as shown in the image below:

2.5 USB Tethering

Note: If you are using the board with a fan or running a computationally intensive task you should always power the board with a dedicated power supply that can supply 5V ≥ 3A (15W+).

As per USB standards these are the current at 5V that each downstream USB port type can (max) supply:

- USB Type-A 3.x port - 900mA (4.5W)
- USB Type-C 1.2 port - 1500mA (7.5W) to 3000mA (15W)

Thus it's recommended to use type-C to type-C cable.

To initially test your board, you can connect the board directly to your computer using a type-C to type-C cable shown in the image below.

2.5.1 SSH connection

After connecting, you should see the power LED glow, and soon just like with other Beagles, BeagleY-AI will create a virtual wired connection on your computer. To access the board, open up a terminal (Linux/Mac) or command prompt (Windows) and use the SSH command as shown below.

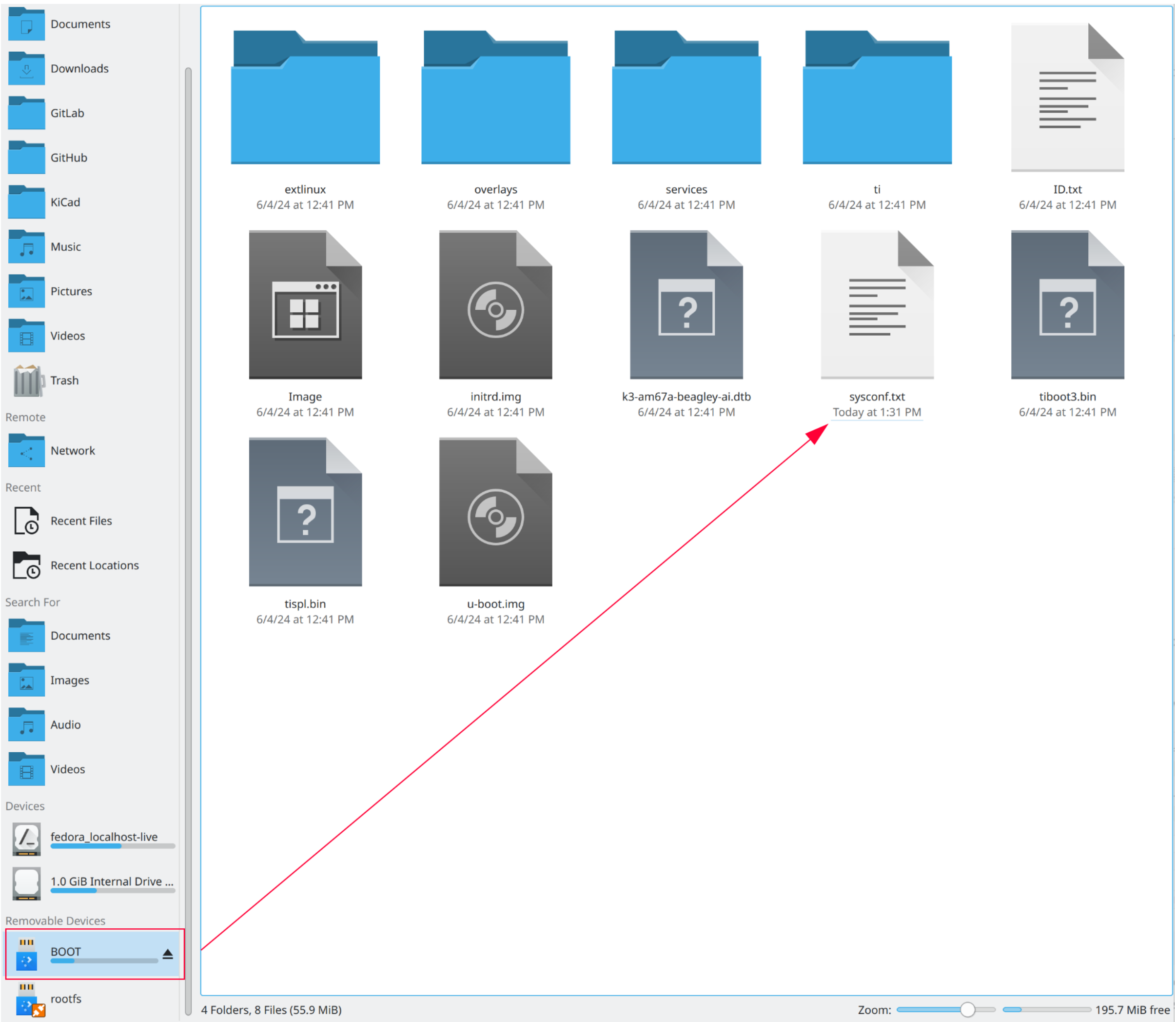


Fig. 2.3: sysconf file under BOOT partition

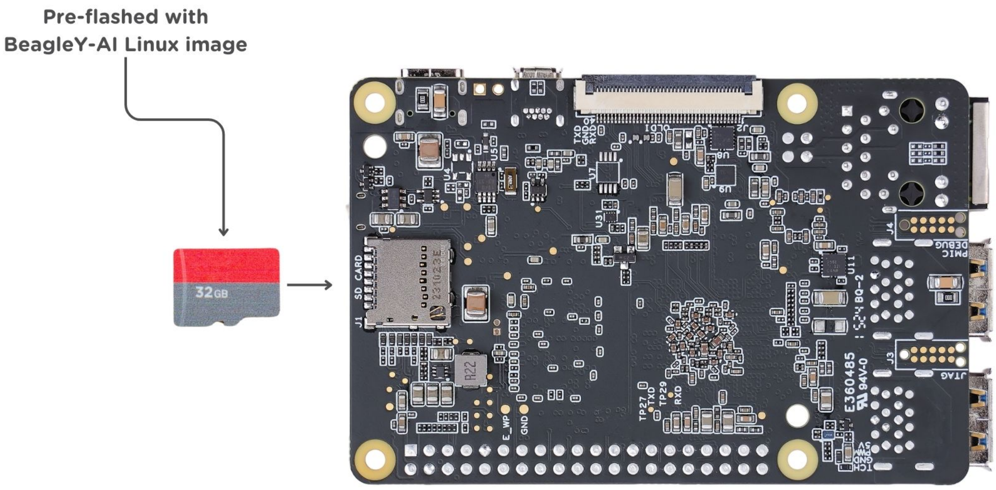


Fig. 2.4: Insert microSD card in BeagleY-AI

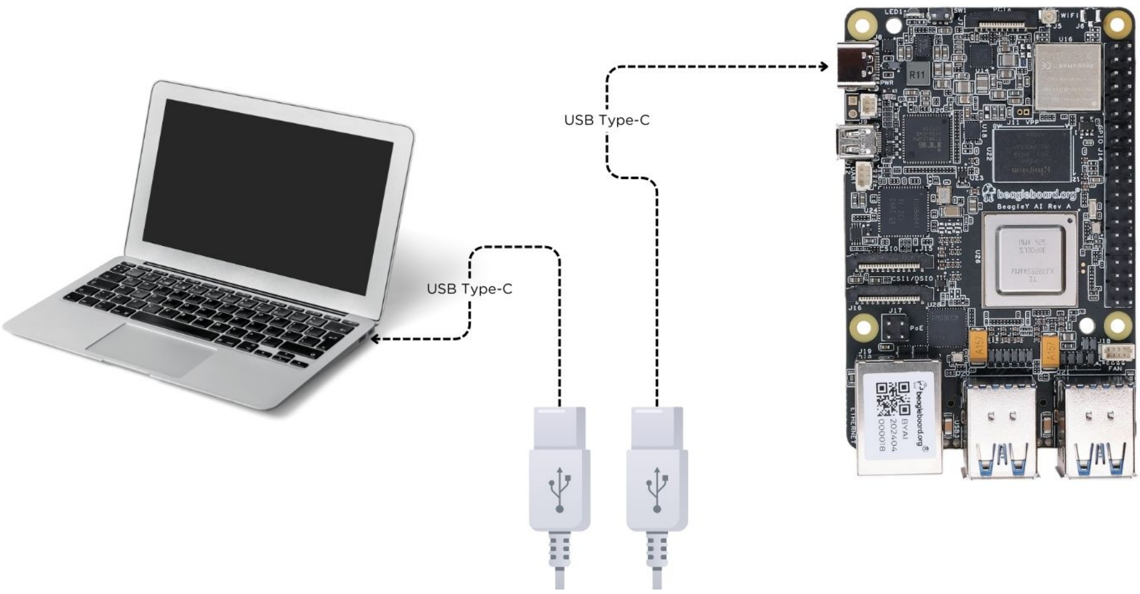


Fig. 2.5: BeagleY-AI tethered connection

```
ssh debian@192.168.7.2
```

Tip: If you are not able to find your beagle at `192.168.7.2` make sure to checkout start-browse-to-beagle to resolve your connection issue.

Important: If you have not updated your default username and password during *Boot Media (Software image)*, you must update the default password at this step to something safer.

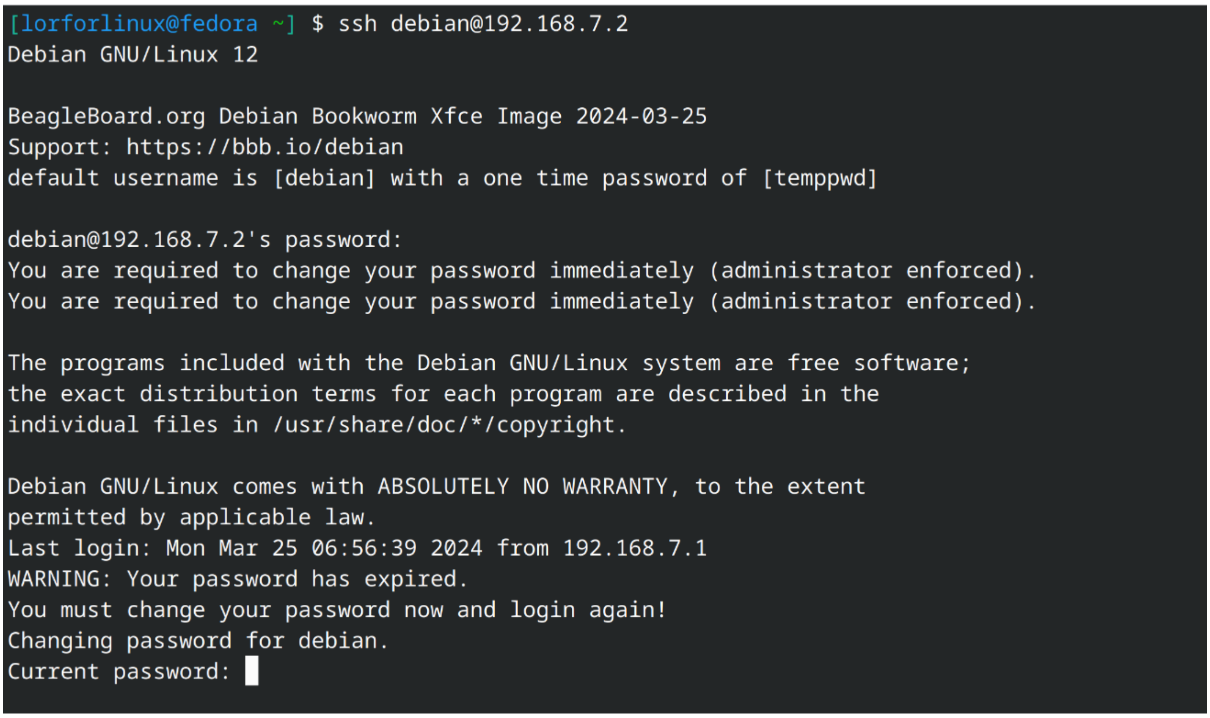


Fig. 2.6: BeagleY-AI SSH connection

2.5.2 UART connection

Your BeagleY-AI board creates a UART connection (No additional hardware required) when tethered to a Laptop/PC which you can access using `Putty` of `tio`. On a linux machine it may come up as `dev/ttyACM*`, it will be different for Mac and Windows operatig systems. To find serial port for your system you can checkout this guide.

- If you are on linux, try `tio` with default setting using command below,

```
tio /dev/ttyACM0
```

With this you have the access to BeagleY-AI terminal. Now, you can connect your board to *WiFi*, try out all the *cool demos* and explore all the other ways to access your BeagleY-AI listed below.

- *Connecting to WiFi*
- *Demos and tutorials*

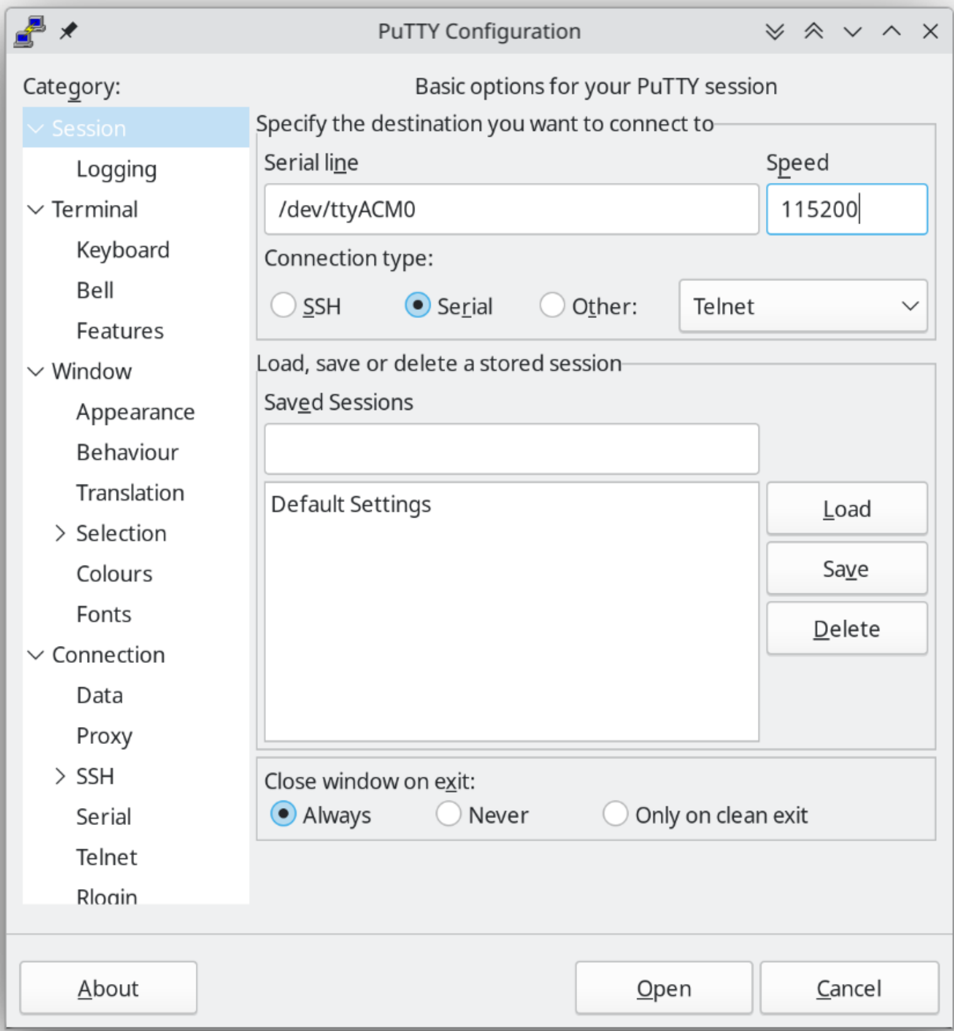


Fig. 2.7: Putty serial connection

2.5.3 Headless connection

If you want to run your BeagleY-AI in headless mode, you need Raspberry Pi Debug Probe or similar serial adapter.

Todo: Add images and description for this section.

2.5.4 Standalone connection

To setup your BeagleY-AI for standalone usage, you need the following additional accessories,

1. HDMI monitor
2. micro HDMI to full-size HDMI cable
3. Wireless keyboard & mice combo
4. Ethernet cable (Optional)

Make sure you have the microSD card with boot media (software image) inserted in to the BeagleY-AI. Now connect,

1. microHDMI to BeagleY-AI and full size HDMI to monitor
2. keyboard and mice combo to one of the four USB port of BeagleY-AI
3. Power supply to USB type-c connector of BeagleY-AI

The connection diagram below provides a clear representation of all the connections,

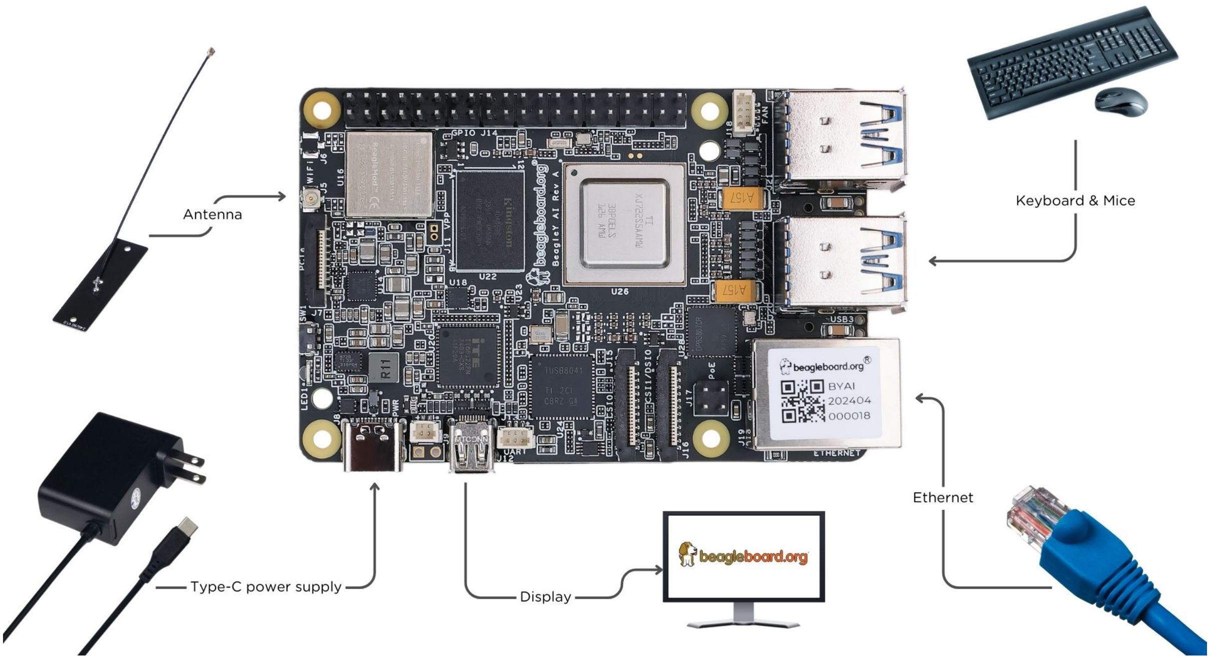


Fig. 2.8: BeagleY-AI standalone connection

If everything is connected properly you should see four penguins on your monitor.

When prompted, log in using the updated login credentials you updated during the USB tethering step.

Fig. 2.9: BeagleY-AI boot penguins

Important: You can not update login credentials at this step, you must update them during boot media (software image) micrSD card flashing or USB tethering step!

Once logged in you should see the splash screen shown in the image below:

Test network connection by running ping 8.8.8.8

Explore and build with your new BeagleY-AI board!

2.6 Connecting to WiFi

We have two options to connect to WiFi,

1. *nmtui*
2. *iwctl*

2.6.1 nmtui

- Enable `NetworkManager`

```
sudo systemctl enable NetworkManager
```

- Start `NetworkManager`

```
sudo systemctl start NetworkManager
```

- Start `nmtui` application

```
sudo nmtui
```

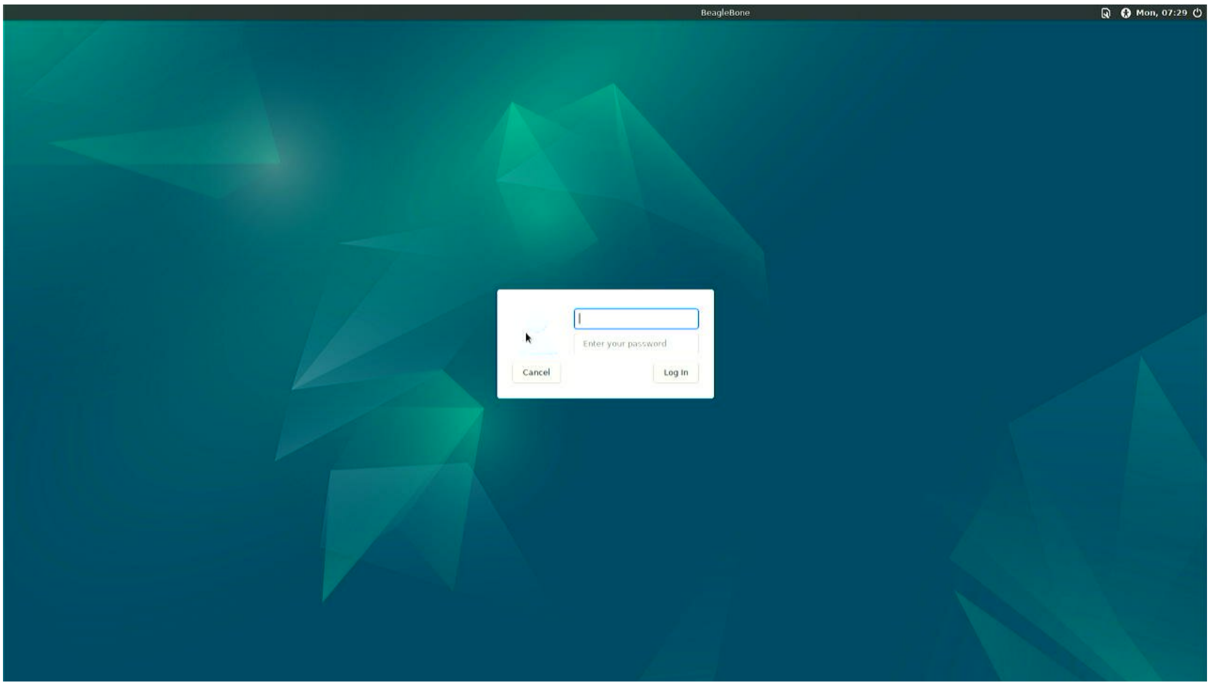


Fig. 2.10: BeagleY-AI XFCE desktop login

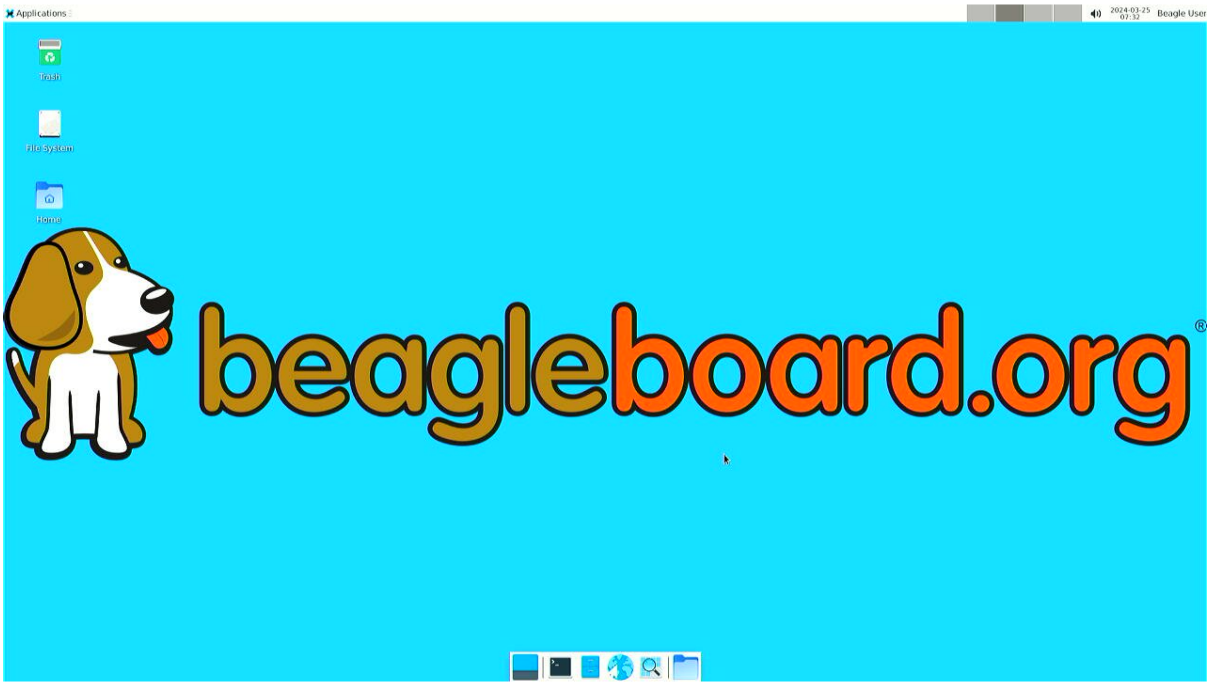


Fig. 2.11: BeagleY-AI XFCE home screen

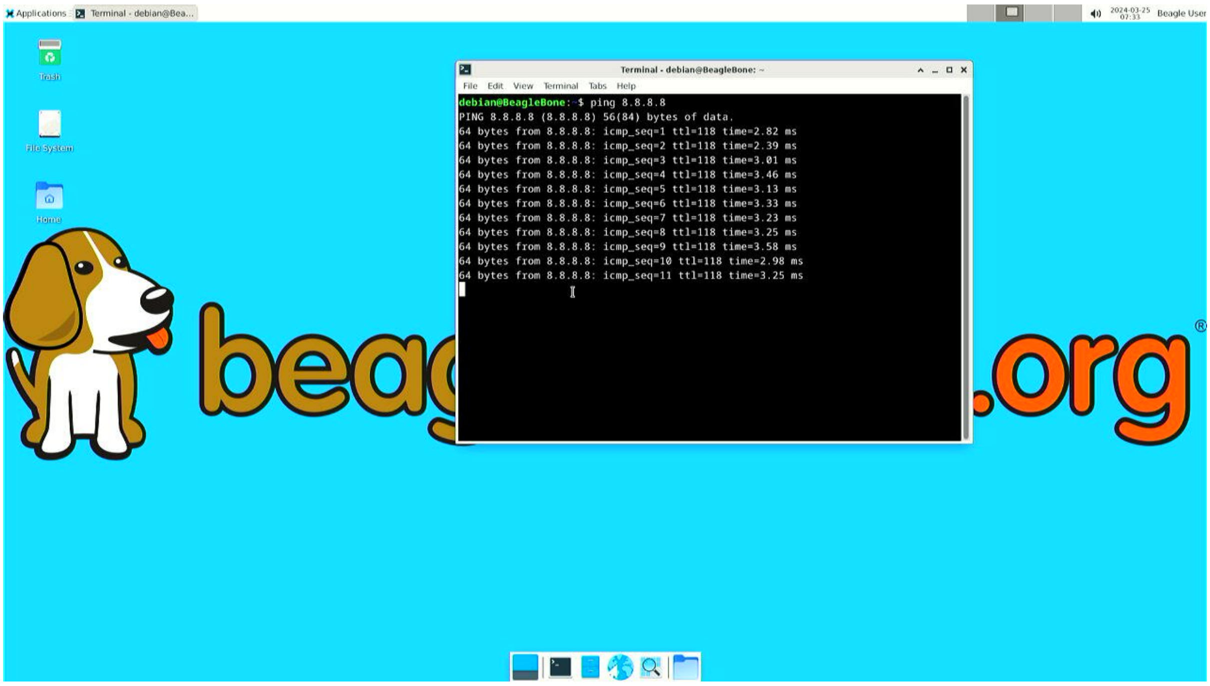


Fig. 2.12: BeagleY-AI network ping test

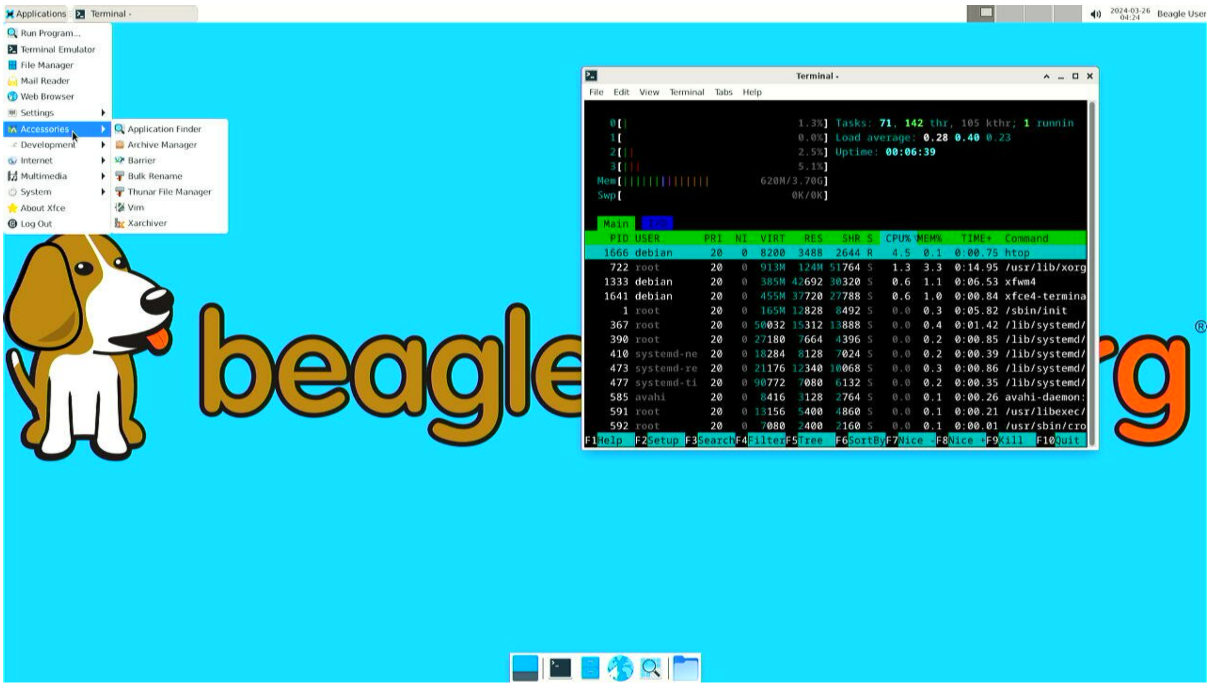


Fig. 2.13: BeagleY-AI running htop

- To navigate, use the `arrow keys` or press `Tab` to step forwards and press `Shift+Tab` to step back through the options. Press `Enter` to select an option. The `Space bar` toggles the status of a check box.
- You should see a screen as shown below, here you have to press `Enter` on `Acticate a connection` option to activate wired and wireless connection options.

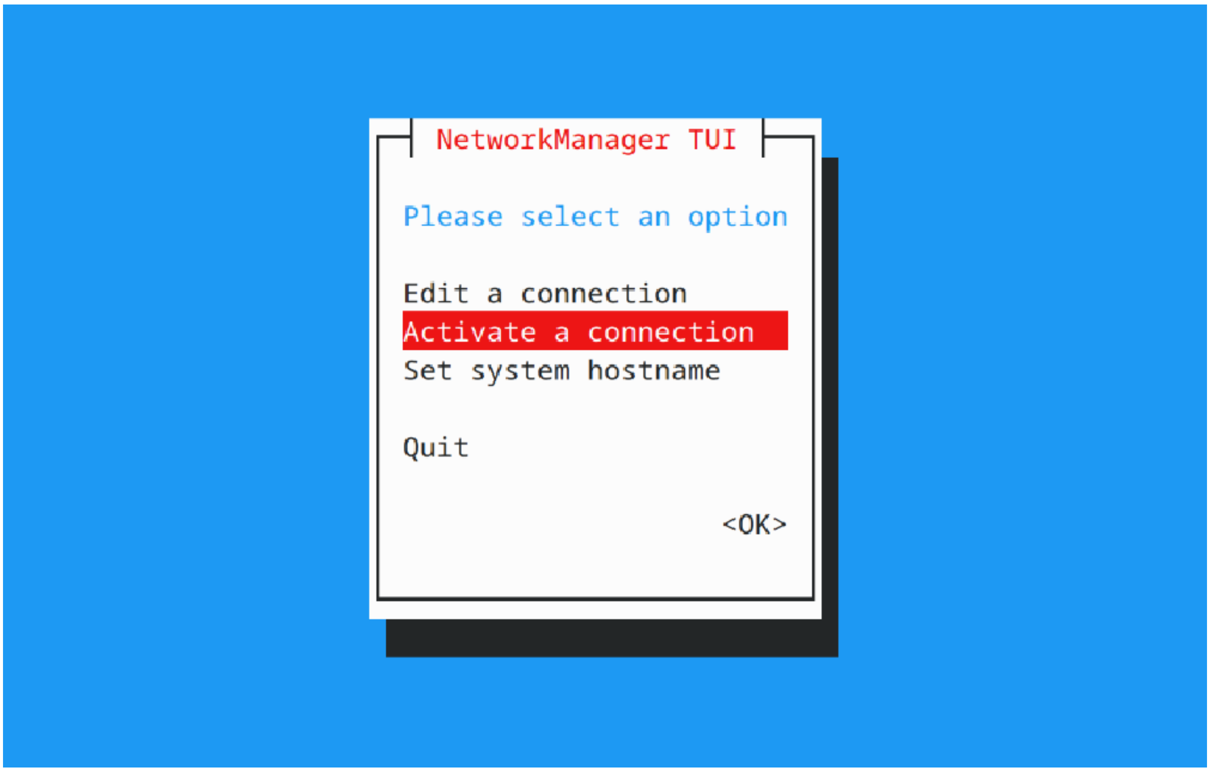


Fig. 2.14: NetworkManager TUI

There under `WiFi` section press `Enter` on desired access point and provide password to connect. When successfully connected press `Esc` to get out of the `nmtui` application window.

2.6.2 iwctl

Once board is fully booted and you have access to the shell, follow the commands below to connect to any WiFi access point,

- To list the wireless devices attached, (you should see wlan0 listed)

```
iwctl device list
```

- Scan WiFi using,

```
iwctl station wlan0 scan
```

- Get networks using,

```
iwctl station wlan0 get-networks
```

- Connect to your wifi network using,

```
iwctl --passphrase "<wifi-pass>" station wlan0 connect "<wifi-name>"
```

- Check wlan0 status with,

```
iwctl station wlan0 show
```

- To list the networks with connected WiFi marked you can again use,

```
iwctl station wlan0 get-networks
```

- Test connection with ping command,

```
ping 8.8.8.8
```

2.7 Attach fan

Todo: add instructions to attach raspberrypi official fan.

2.8 Demos and Tutorials

- *Booting from NVMe Drives*

Chapter 3

Design and specifications

Work in progress

Todo: Add details about all the schematic sections.

If you want to know how BeagleY-AI is designed and the detailed specifications, then this chapter is for you. We are going to attept to provide you a short and crisp overview followed by discussing each hardware design element in detail.

Tip: For board files, 3D model, and more, you can checkout the BeagleY-AI repository on OpenBeagle.

3.1 Block diagram and overview

3.2 SoC

3.3 Boot modes

3.4 Power sources

3.5 PMIC

3.6 General connectivity and expansion

3.7 Buttons and LEDs

3.8 Networking

3.9 Ethernet

3.10 Memory, media, and storage

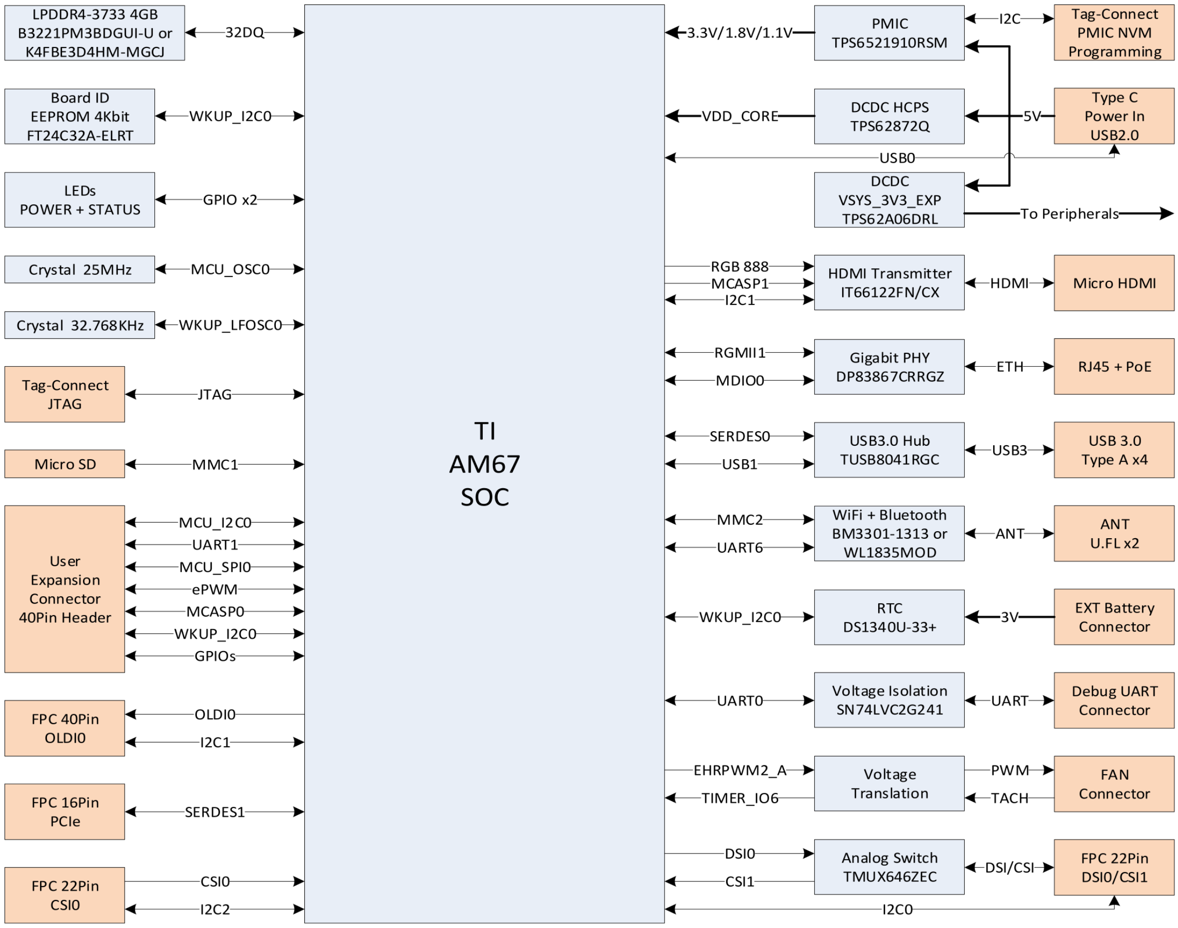


Fig. 3.1: BeagleY-AI block diagram

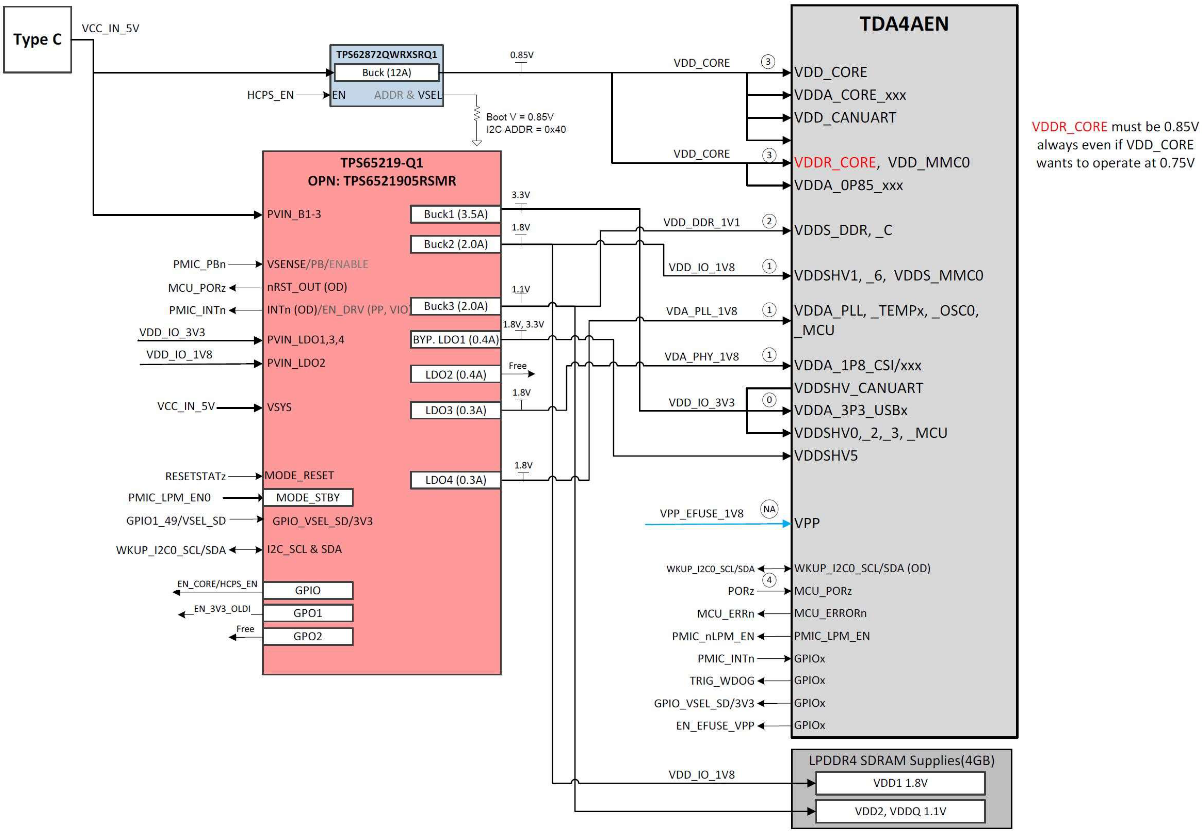


Fig. 3.2: BeagleY-AI power distribution network

3.11 Multimedia I/O

3.12 Debug ports

3.13 Mechanical Specifications

3.13.1 Dimensions & Weight

Table 3.1: Dimensions & weight

Parameter	Value
Size	85 x 56 x 20 mm
Max heigh	20mm
PCB Size	85 x 56 mm
PCB Layers	14 layers
PCB Thickness	1.6mm
RoHS compliant	Yes
Gross Weight	110 g
Net Weight	50 g

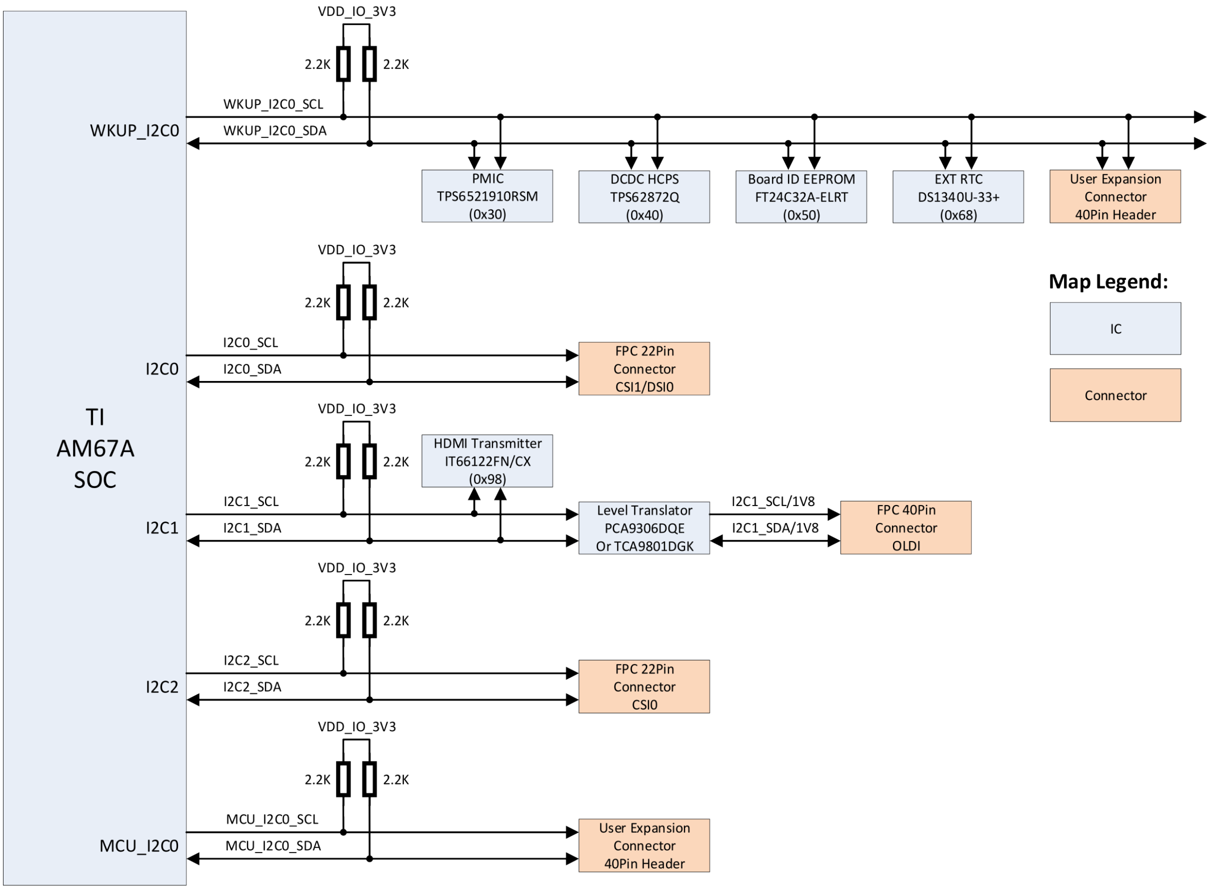


Fig. 3.3: BeagleY-AI I2C tree

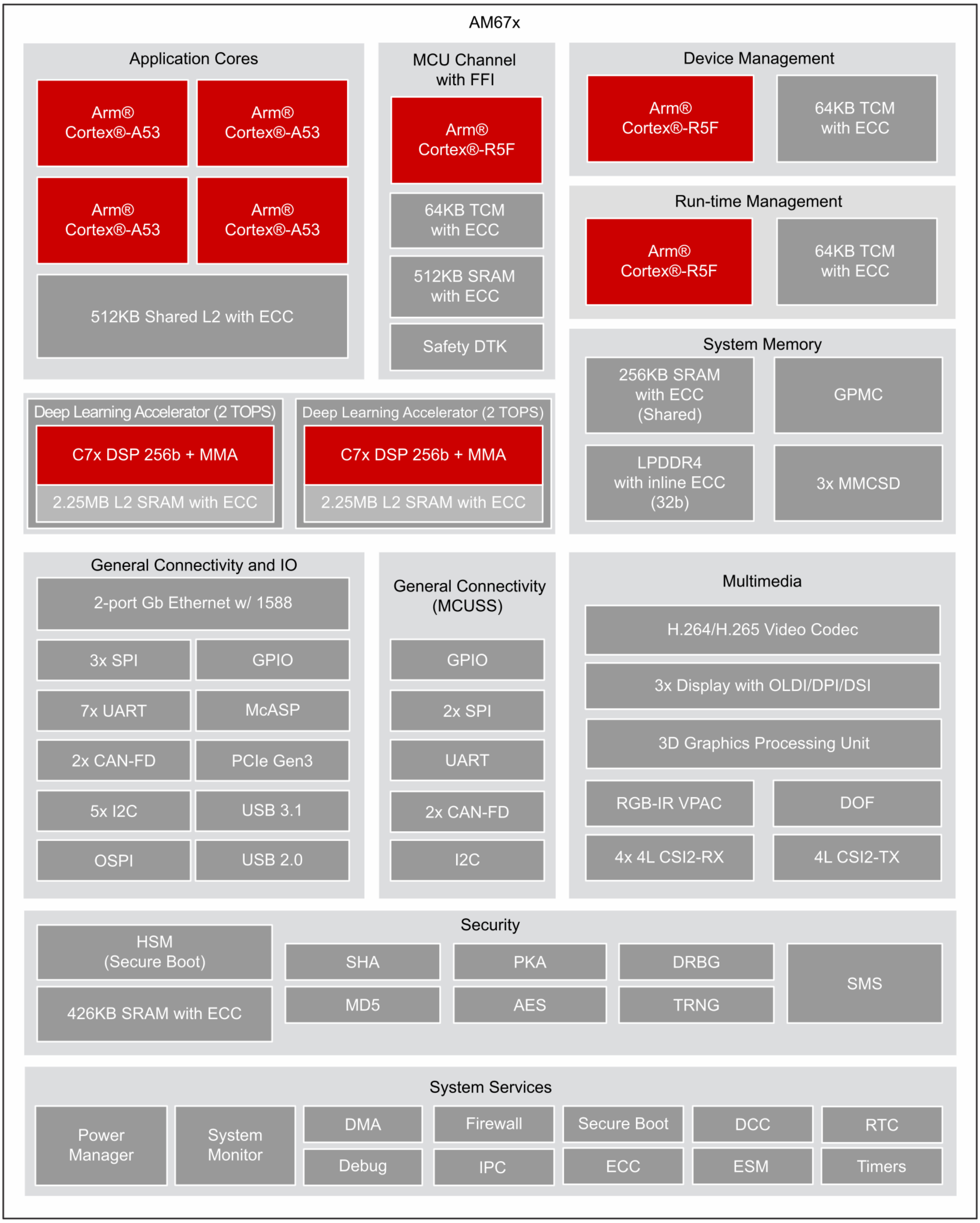


Fig. 3.4: AM67A block diagram

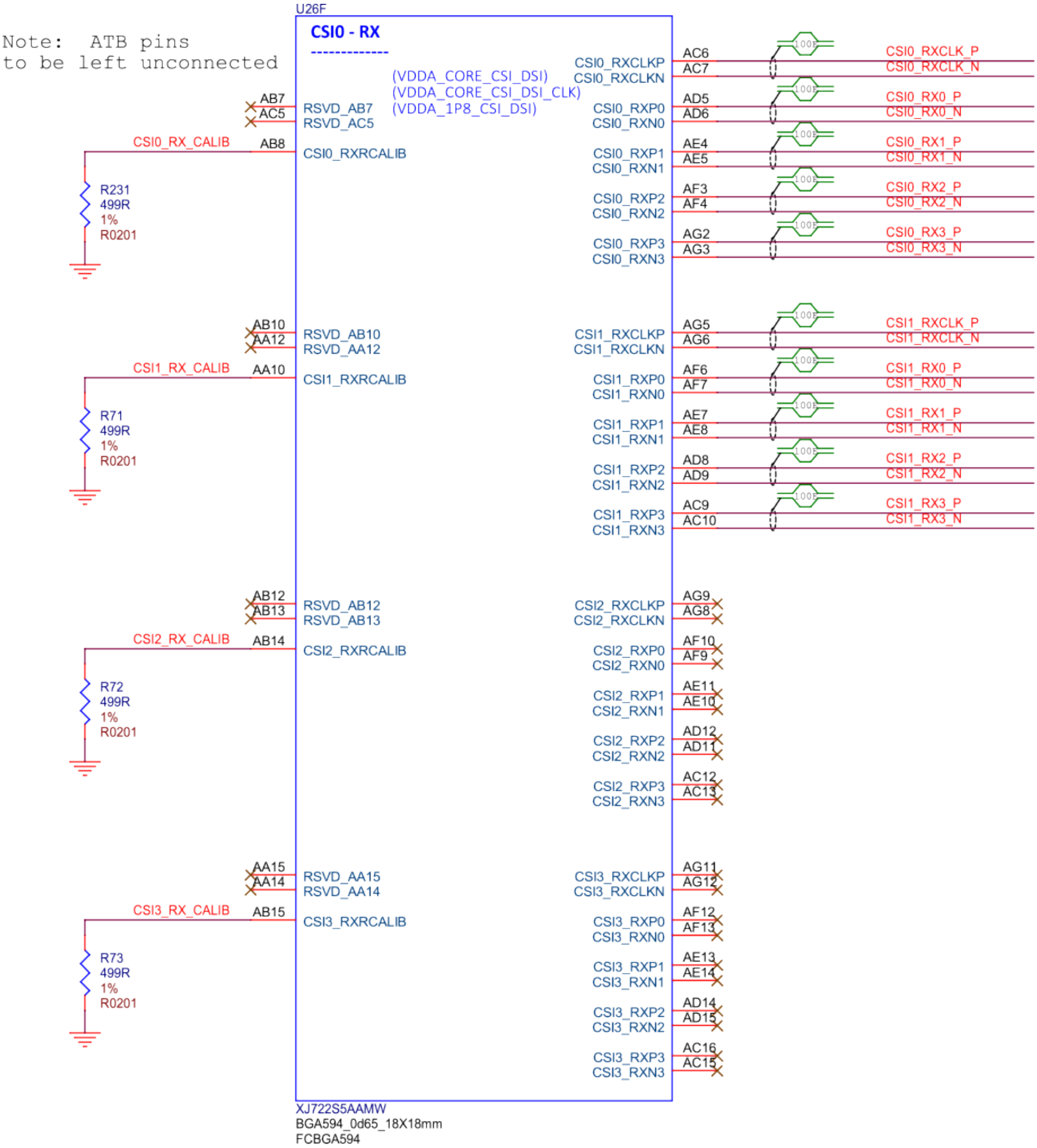


Fig. 3.5: BeagleY-AI SoC CSI1, CSI2, and CSI3

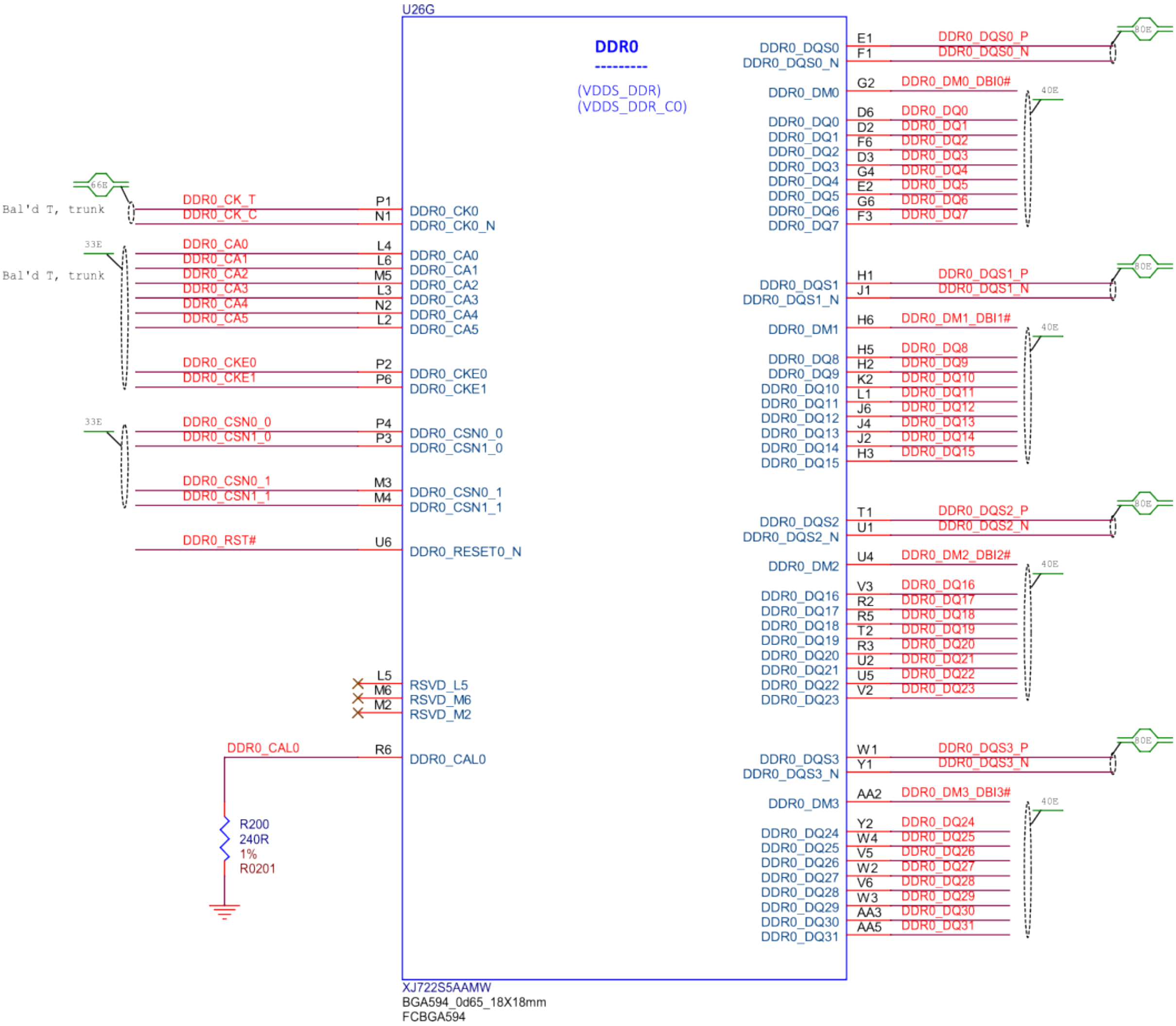


Fig. 3.6: BeagleY-AI SoC DDR0 connections

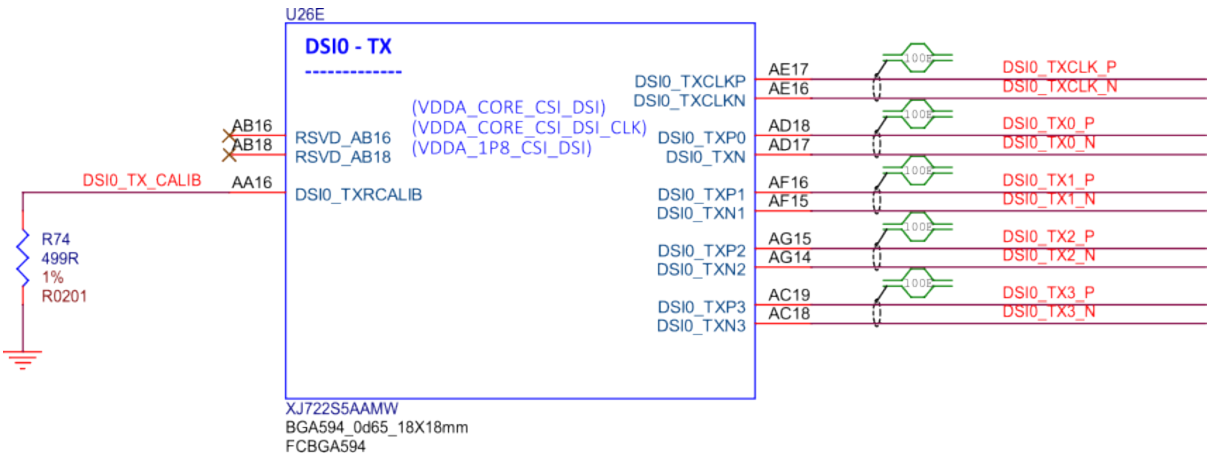


Fig. 3.7: BeagleY-AI SoC DSI0 TX connections

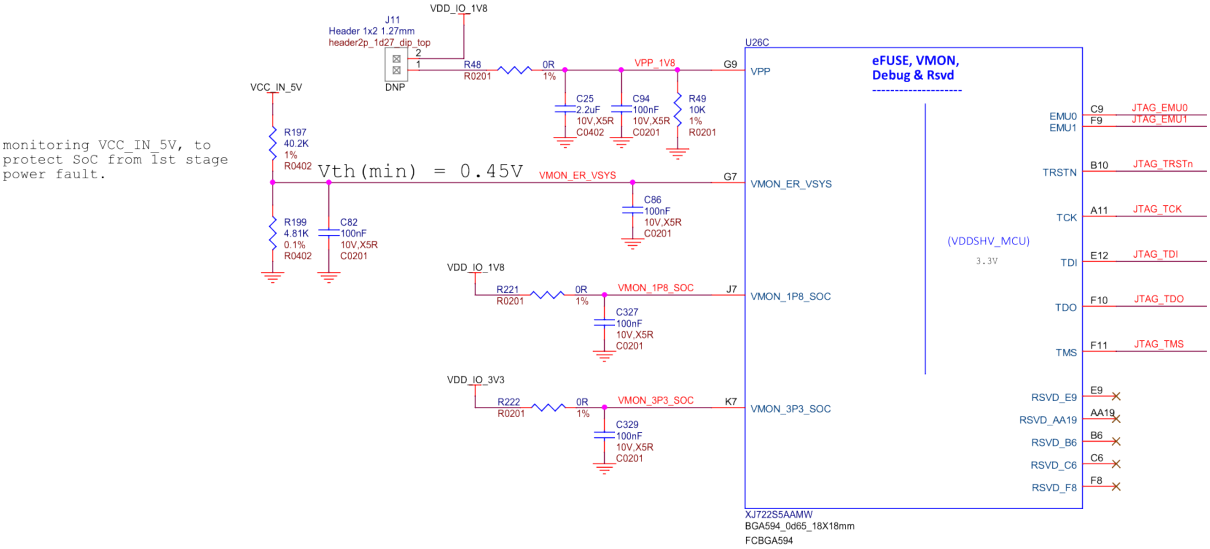


Fig. 3.8: BeagleY-AI SoC eFUSE, VMON, Debug, and RSVD

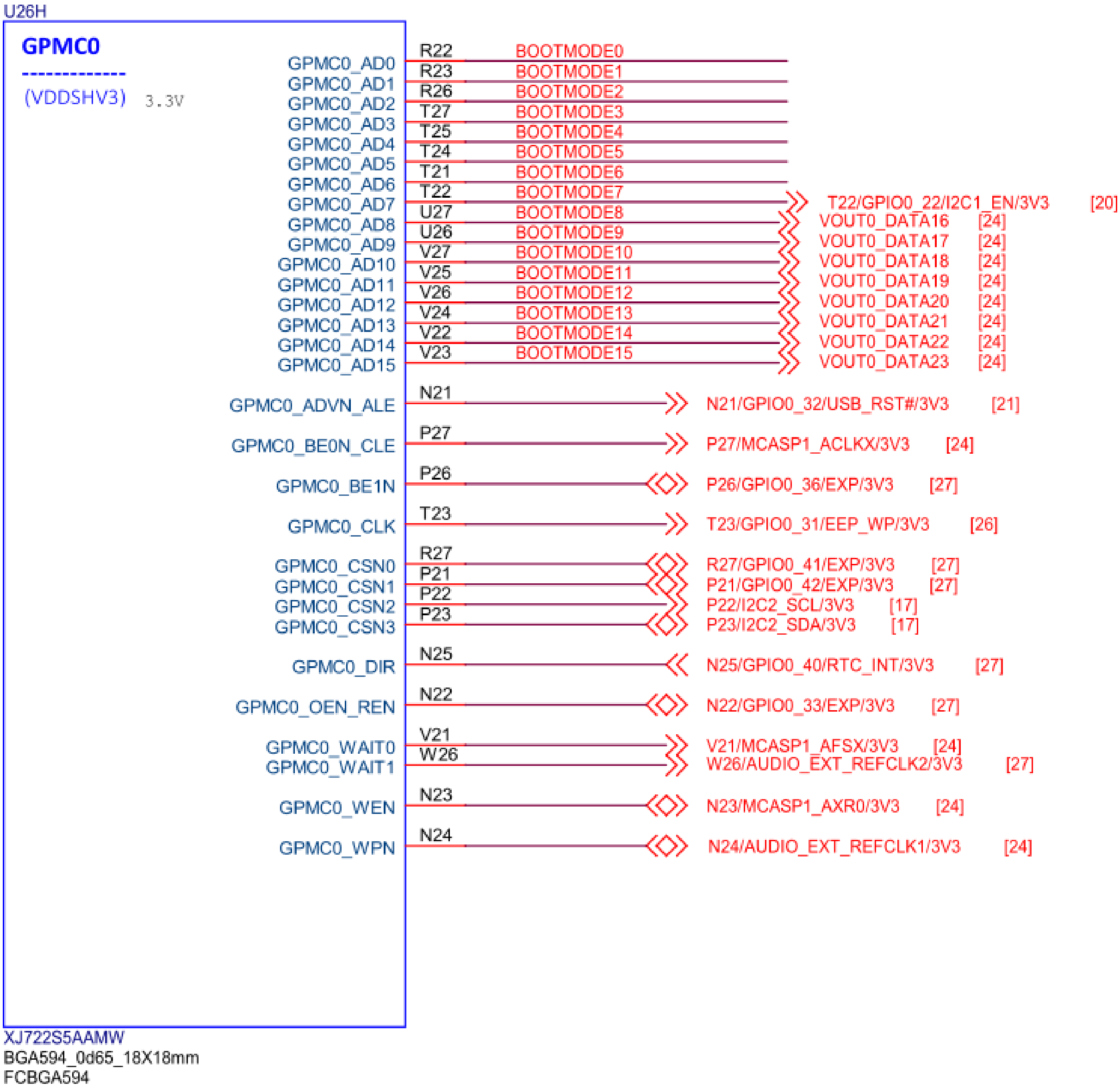


Fig. 3.9: BeagleY-AI SoC GPMC0

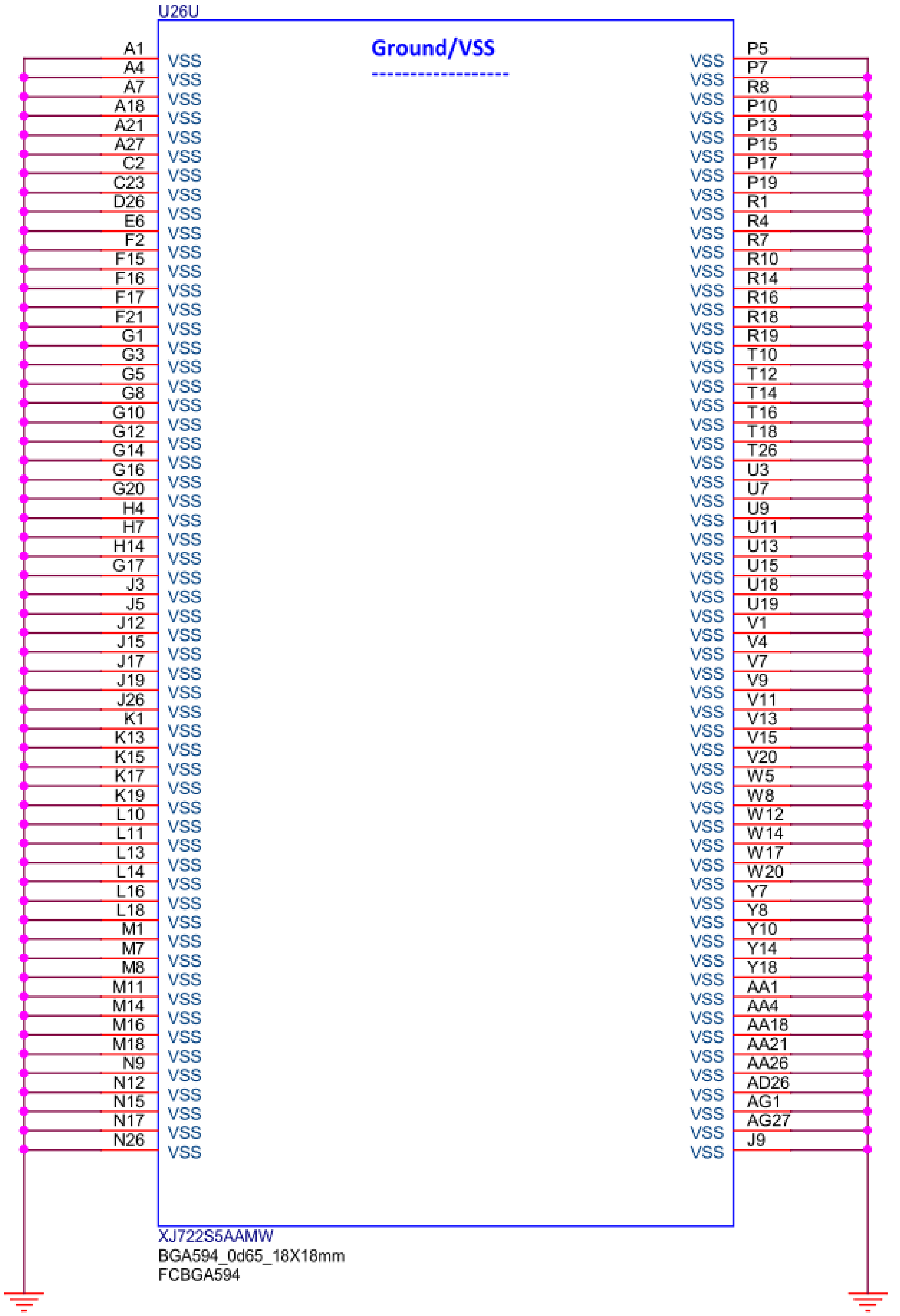


Fig. 3.10: BeagleY-AI SoC ground connections

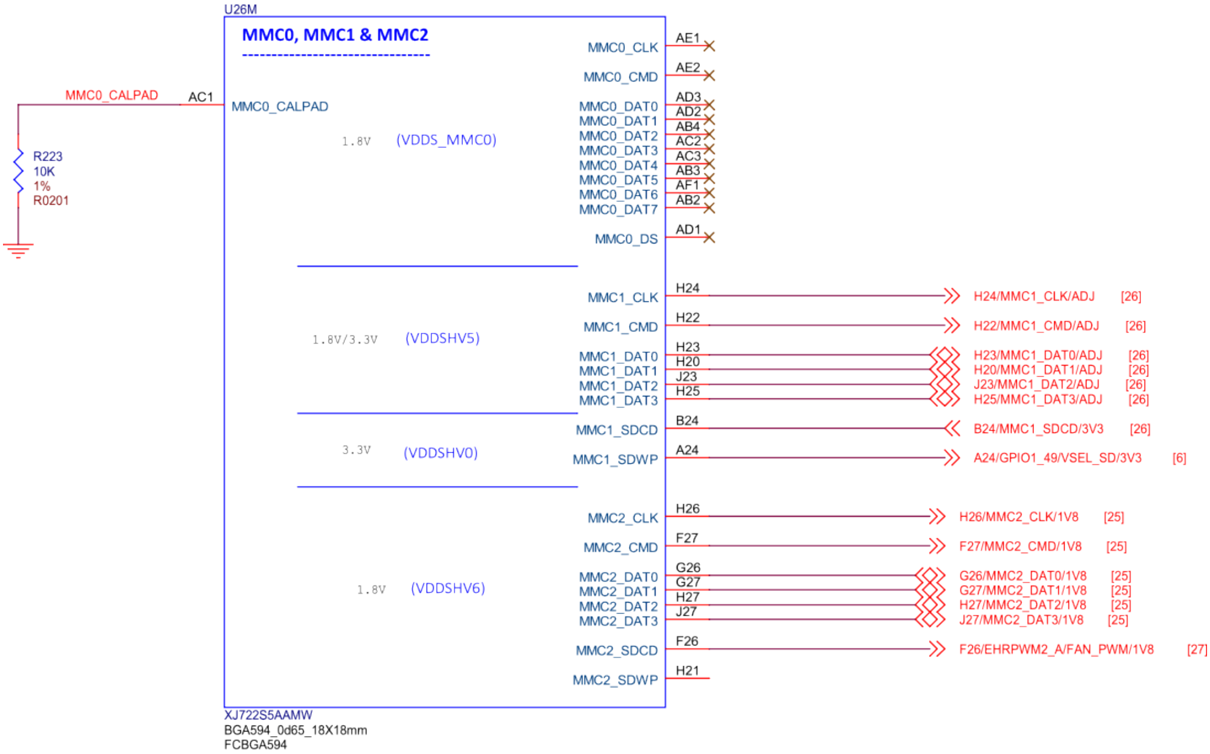


Fig. 3.11: BeagleY-AI SoC MMC0, MMC1, and MMC2

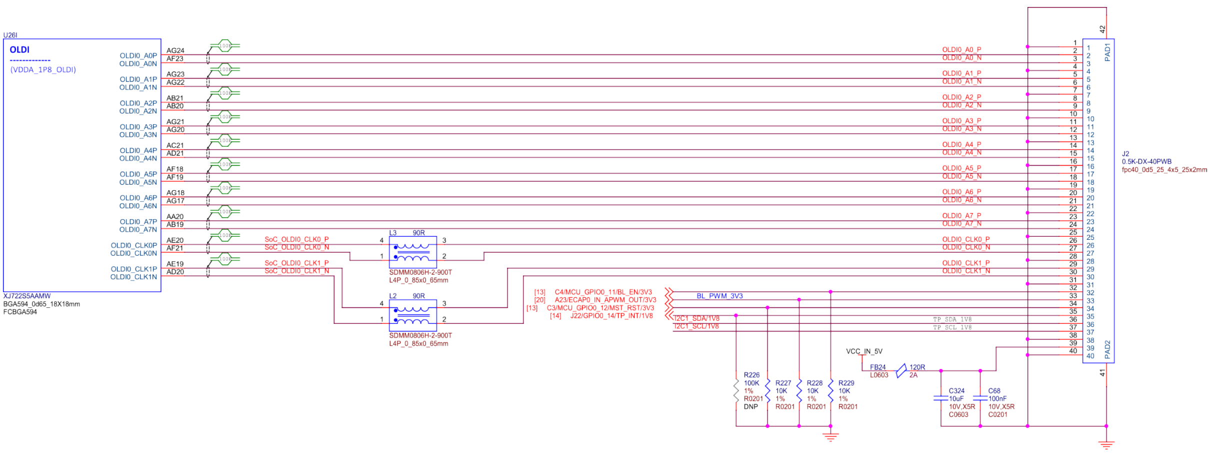


Fig. 3.12: BeagleY-AI SoC OLDI

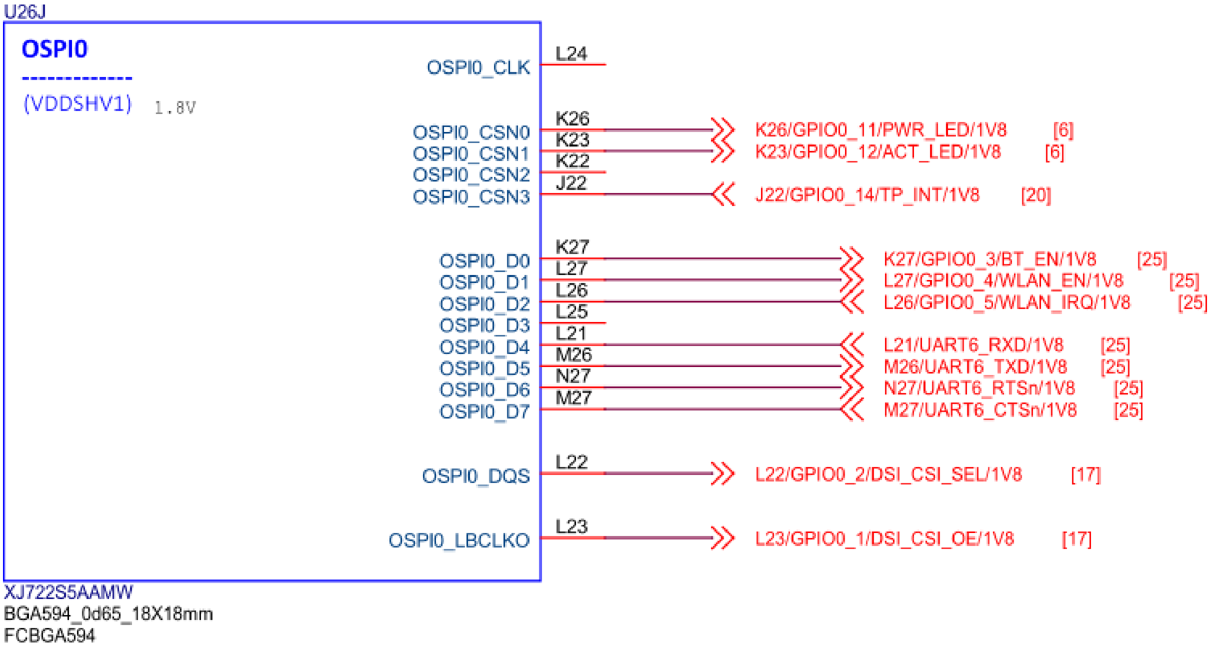


Fig. 3.13: BeagleY-AI SoC OSPI0

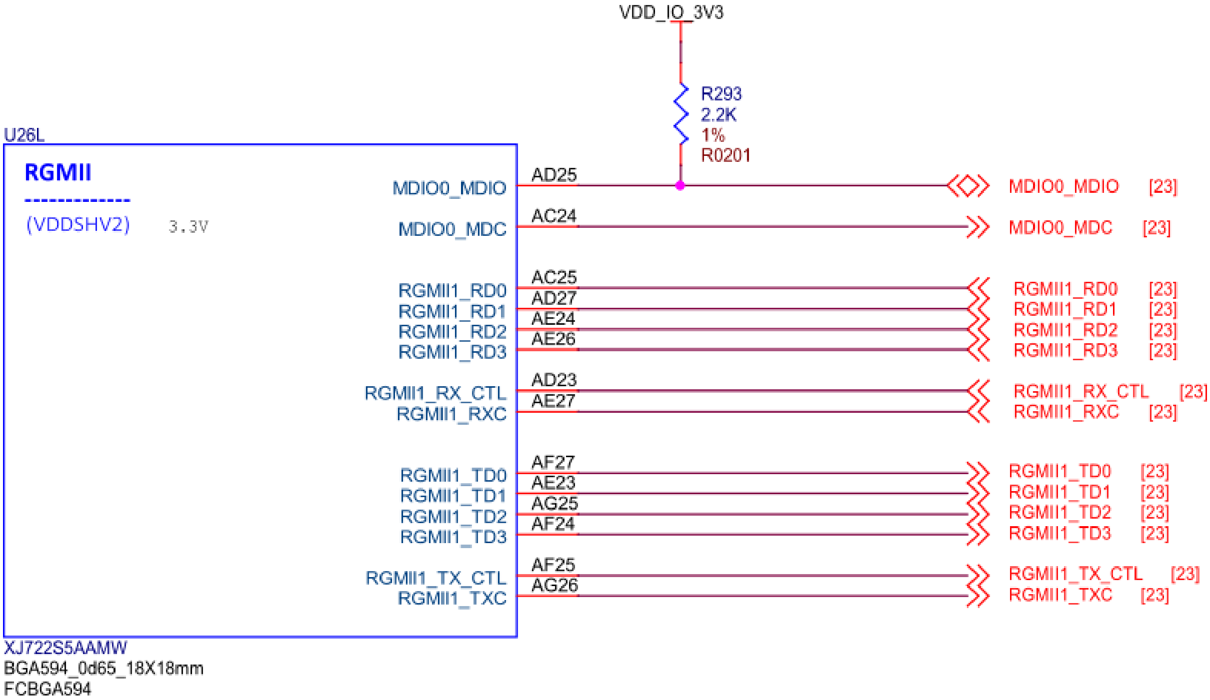


Fig. 3.14: BeagleY-AI SoC RGMII

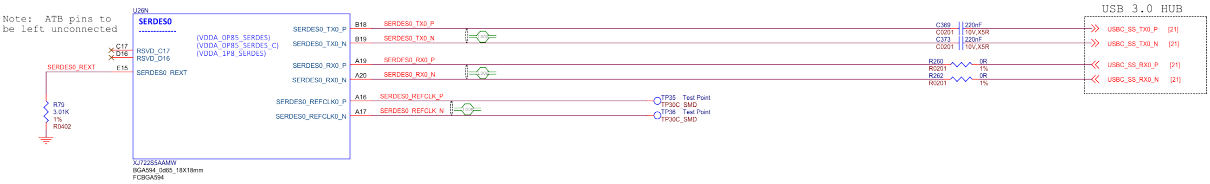


Fig. 3.15: BeagleY-AI SoC SERDES0

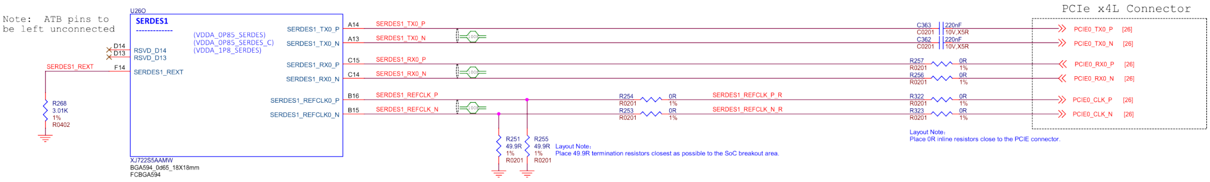


Fig. 3.16: BeagleY-AI SoC SERDES1

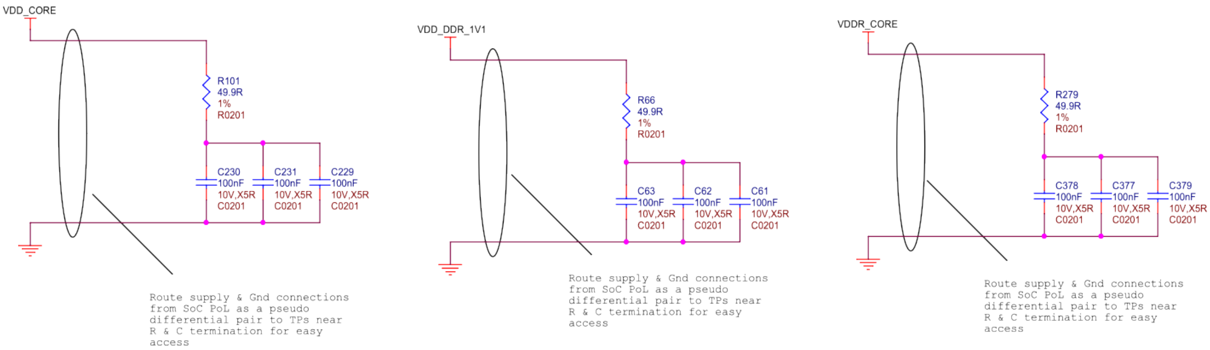


Fig. 3.17: BeagleY-AI SoC supply noise kelvin sensing

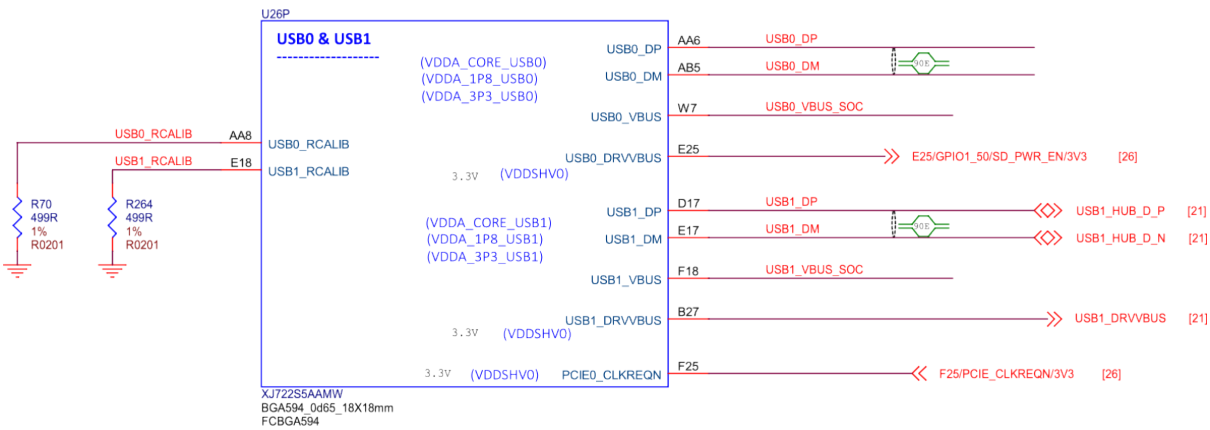


Fig. 3.18: BeagleY-AI SoC USB0 and USB1

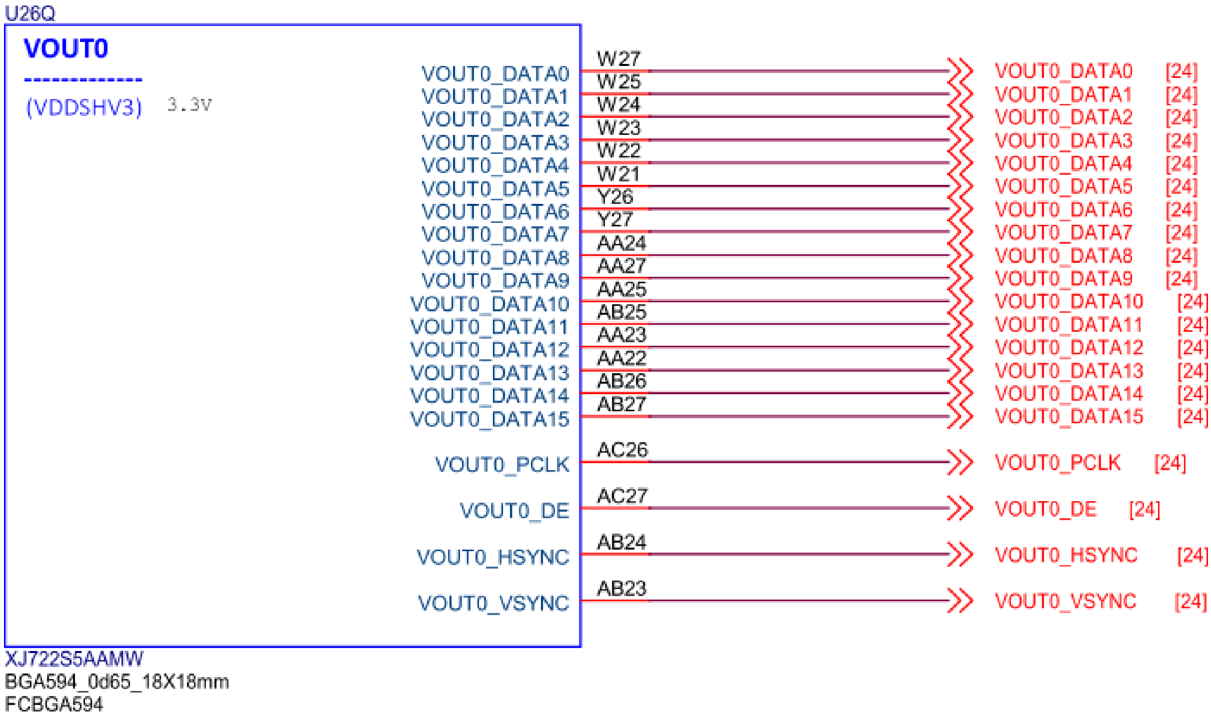


Fig. 3.19: BeagleY-AI SoC VOUT

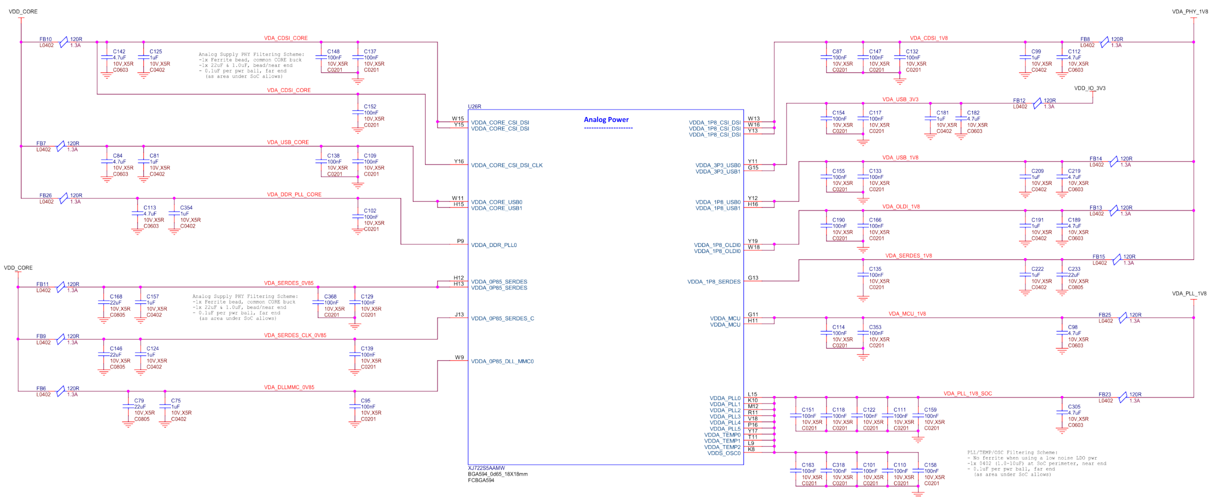


Fig. 3.20: BeagleY-AI SoC analog power1

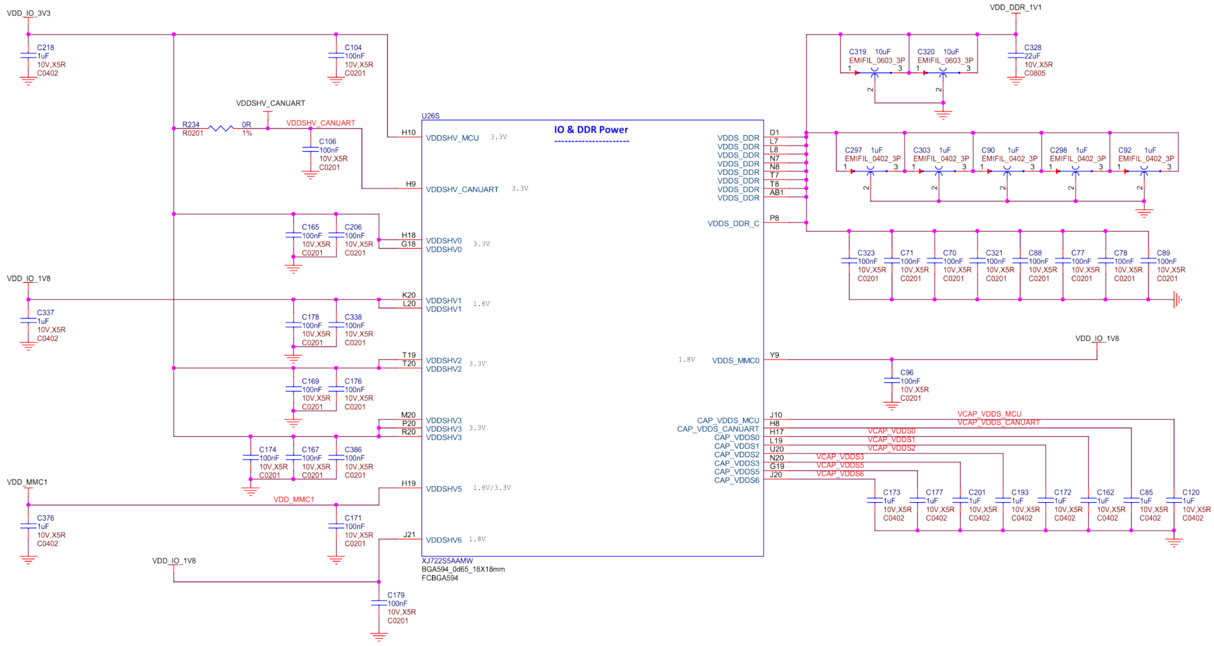


Fig. 3.21: BeagleY-AI AI SoC IO and DDR power2

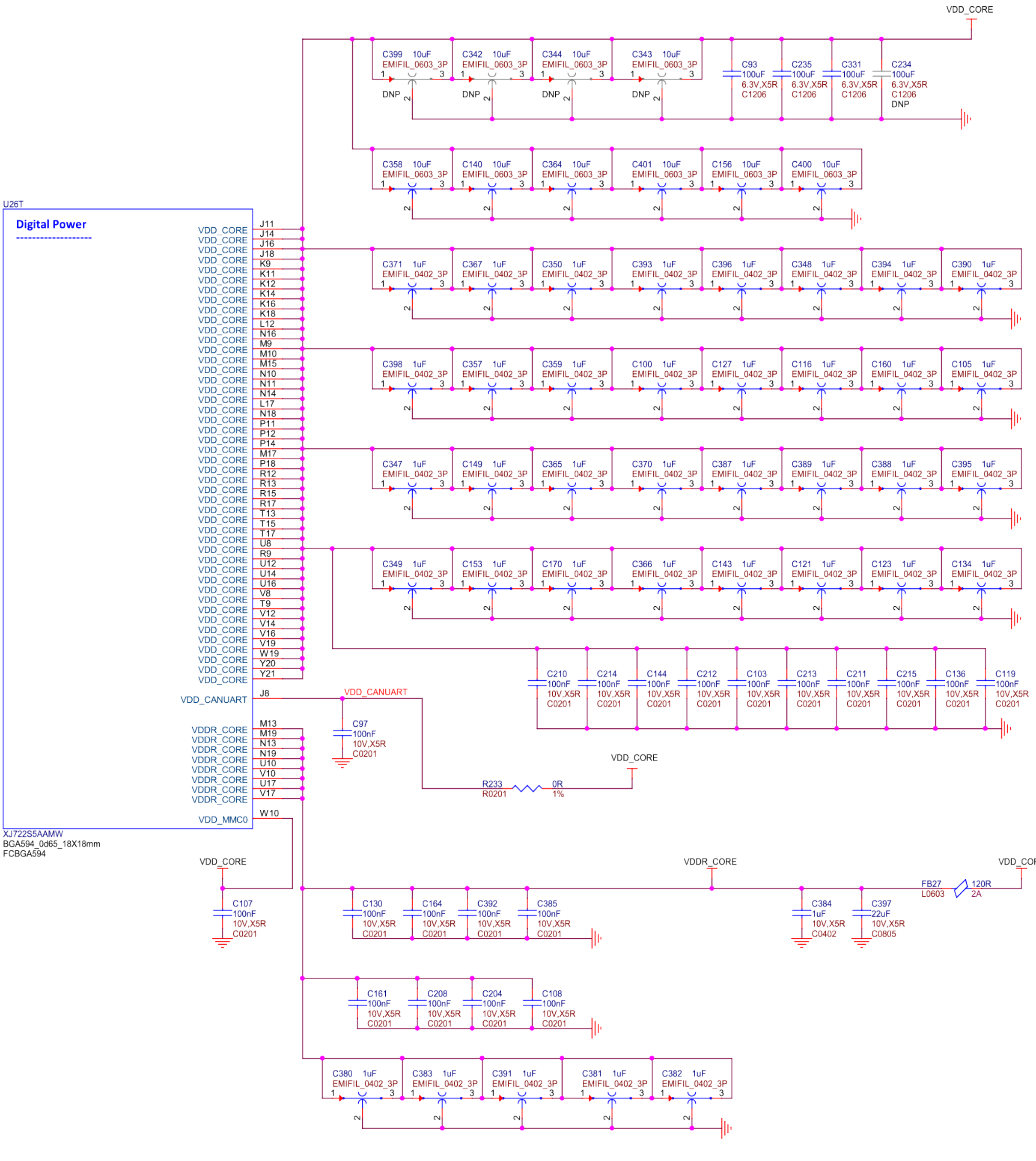


Fig. 3.22: BeagleY-AI SoC digital power3

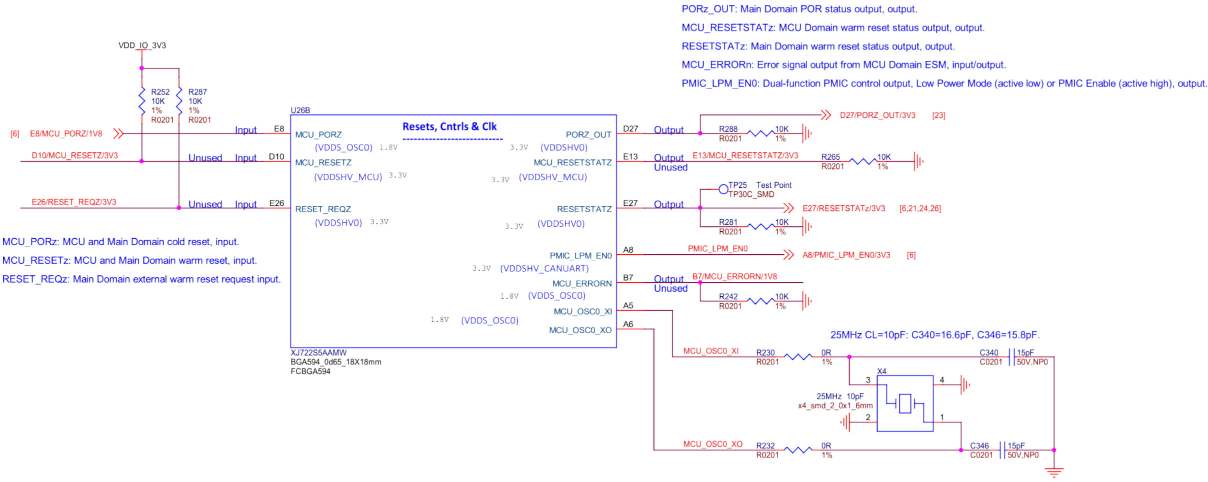


Fig. 3.23: BeagleY-AI SoC Reset, Cntrls, and Clk

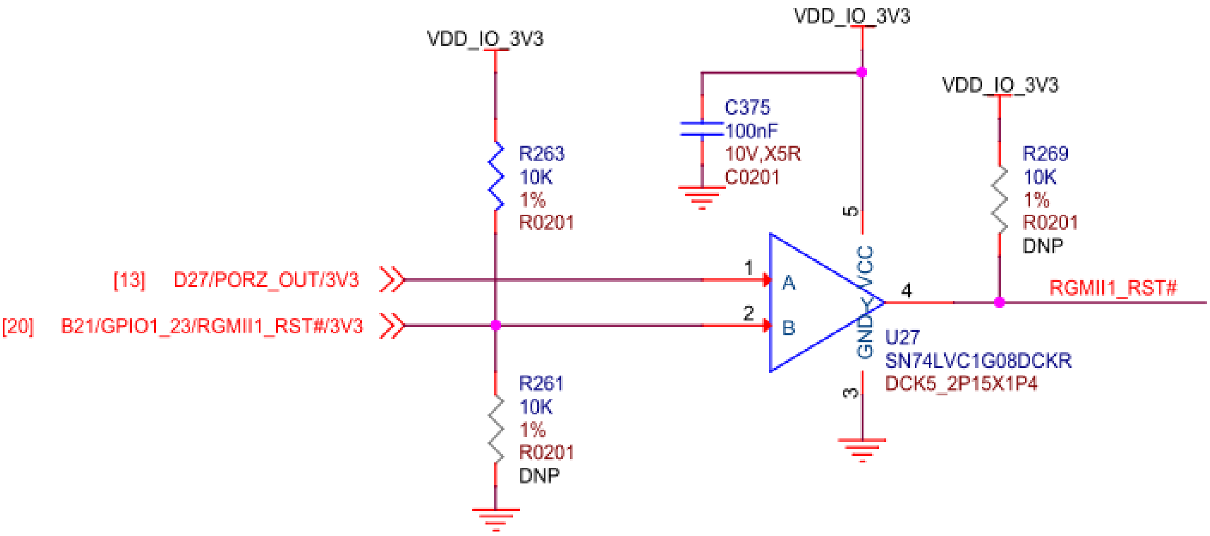


Fig. 3.24: BeagleY-AI SoC RGMII1 RST

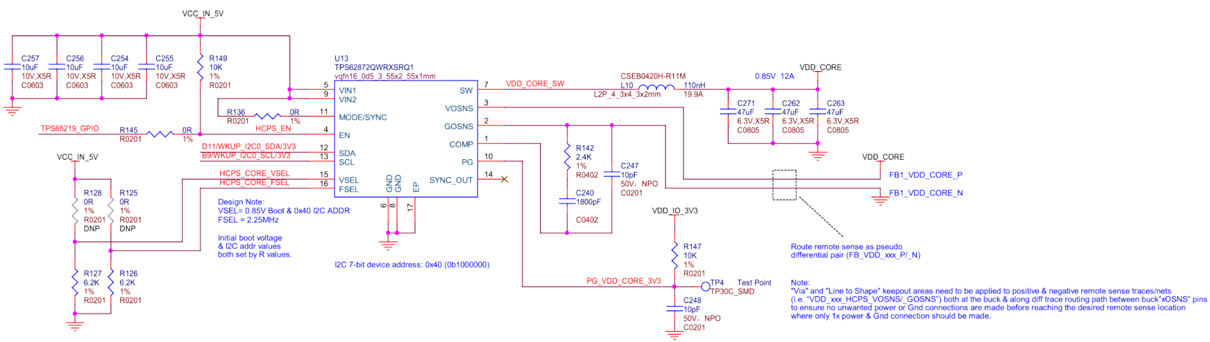


Fig. 3.25: BeagleY-AI VDD core hcps

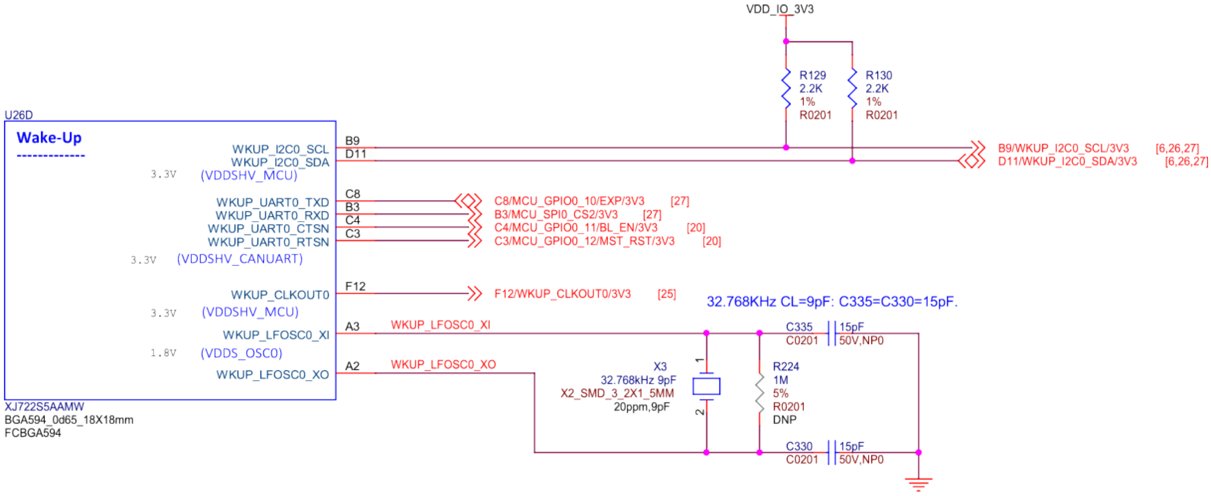


Fig. 3.26: BeagleY-AI wkup reset cntrls osc

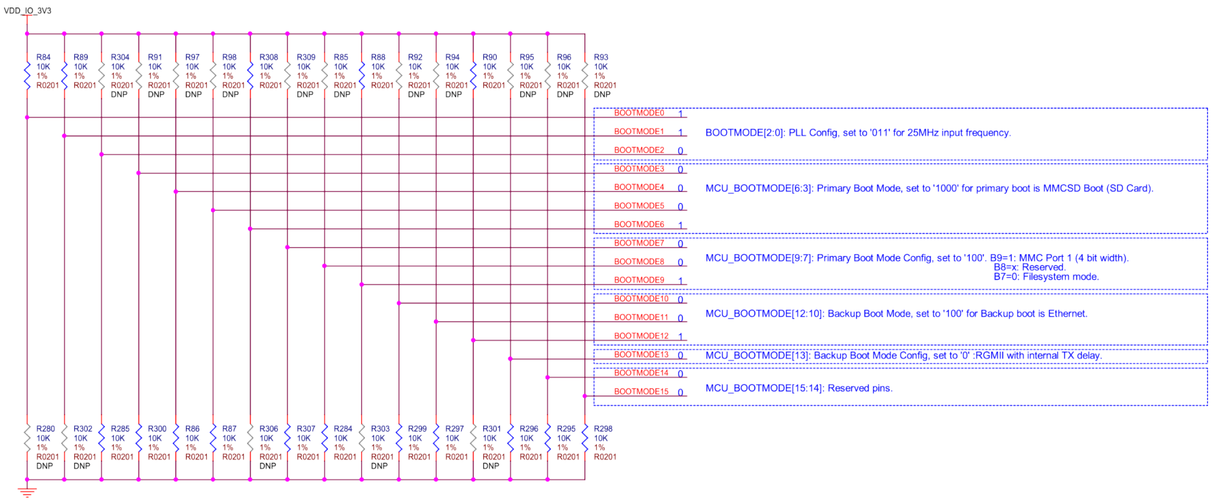


Fig. 3.27: BeagleY-AI boot modes

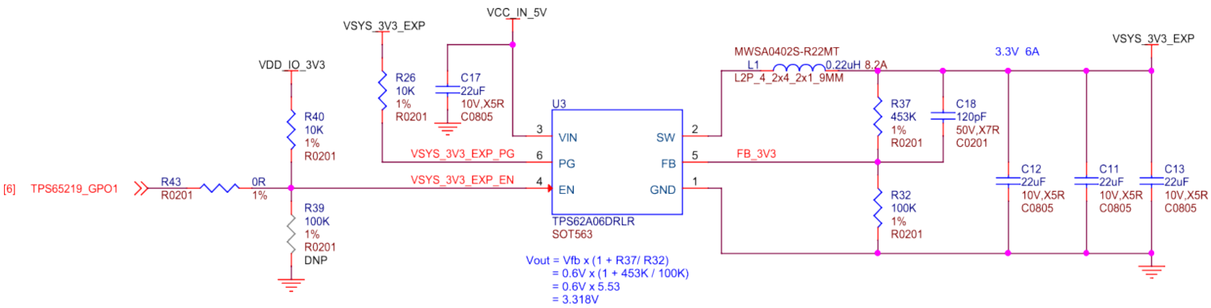


Fig. 3.28: BeagleY-AI VSYS 3V3

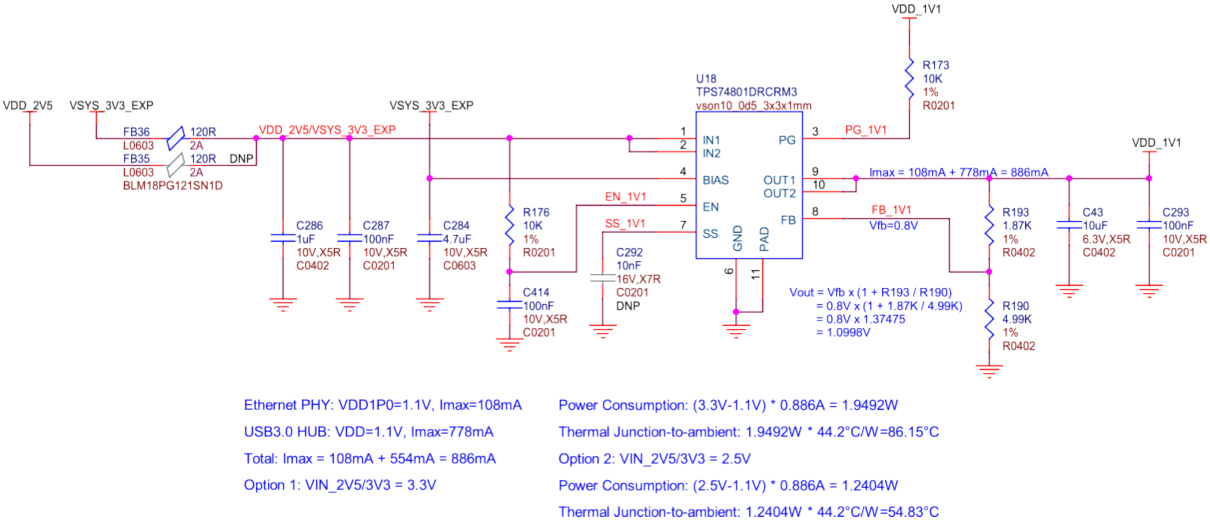


Fig. 3.29: BeagleY-AI 3V3/V5 to 1V1 LDO

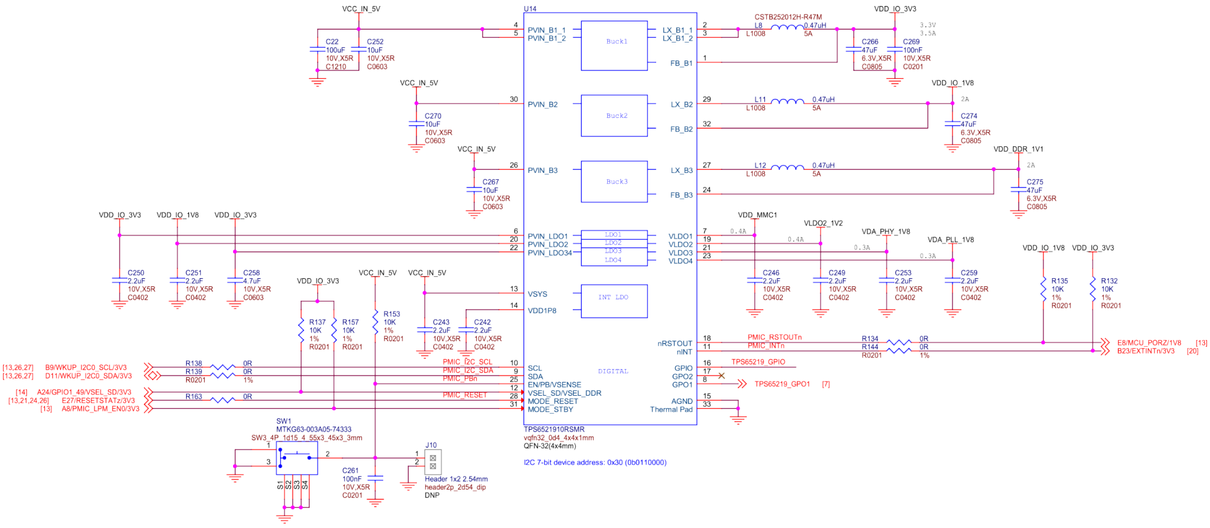


Fig. 3.30: BeagleY-AI PMIC

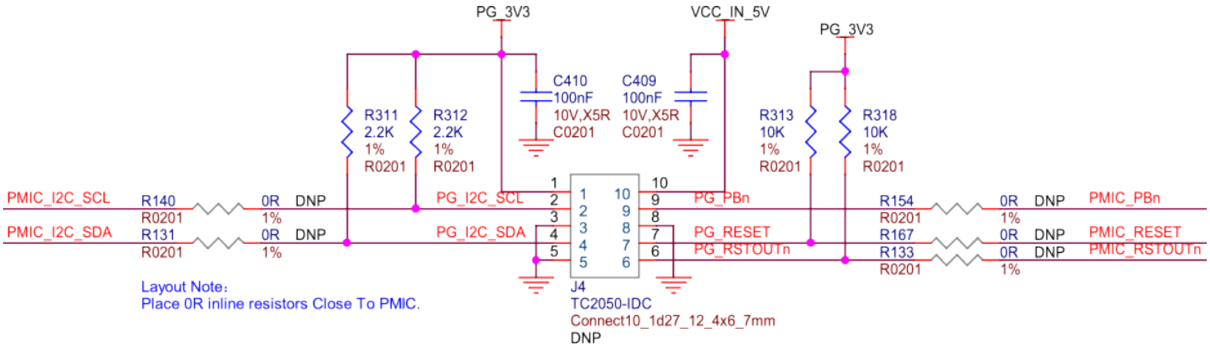


Fig. 3.31: BeagleY-AI PMIC NVM programming interface

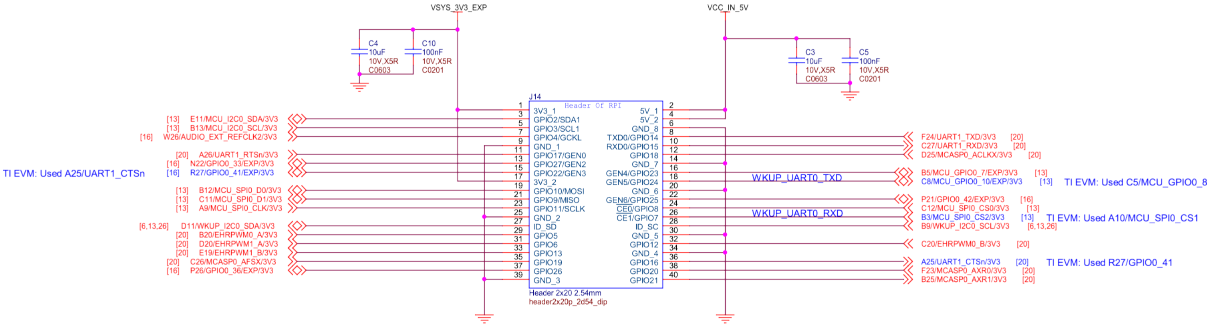


Fig. 3.32: BeagleY-AI user expansion connector

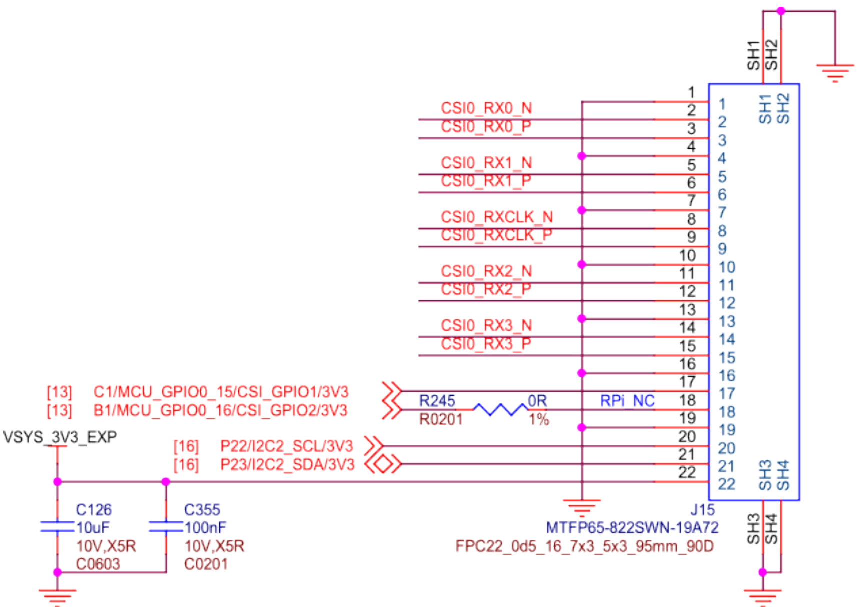


Fig. 3.33: BeagleY-AI RPI CSI

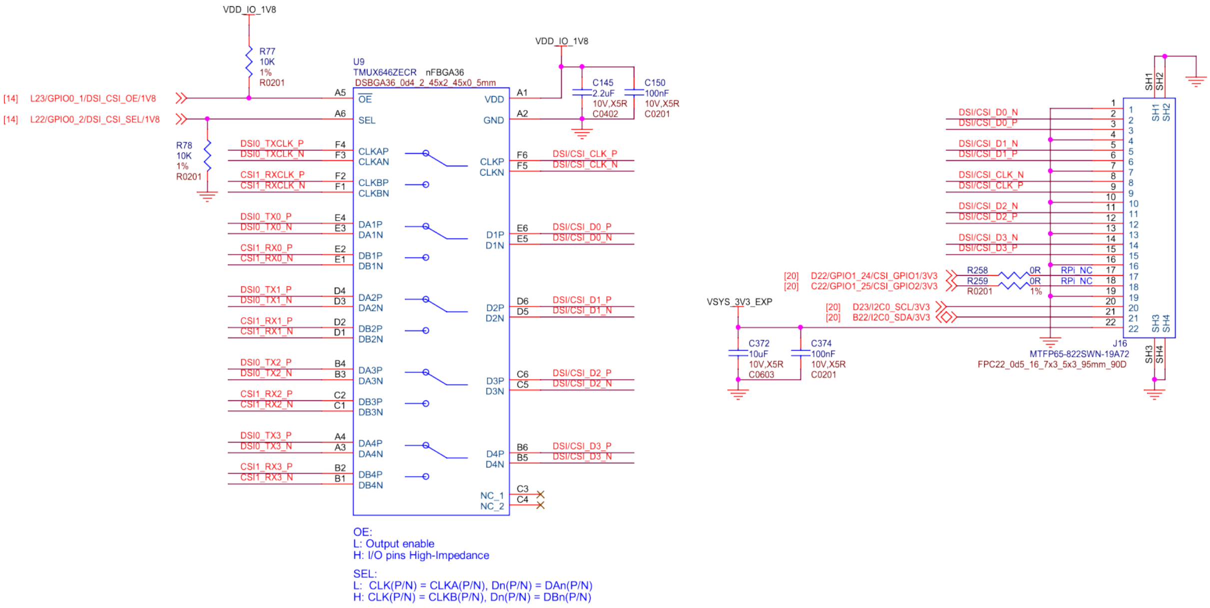


Fig. 3.34: BeagleY-AI RPI DSI/CSI

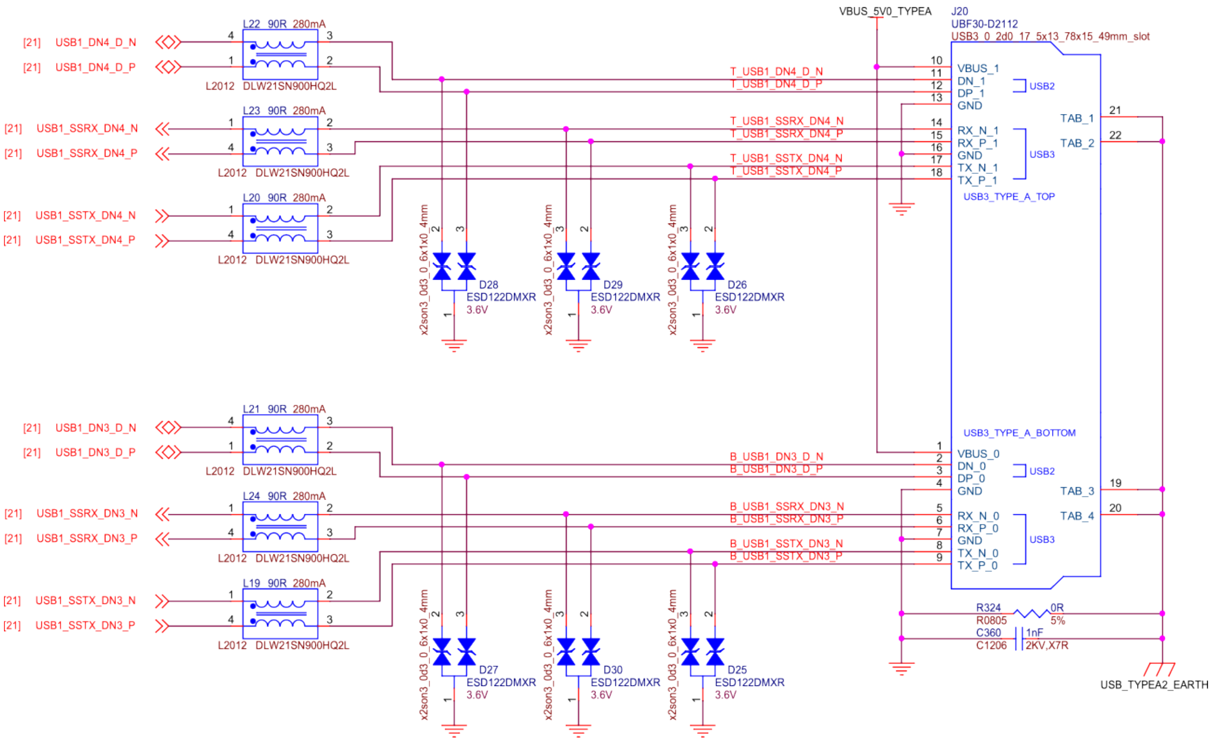


Fig. 3.35: BeagleY-AI dual USB1

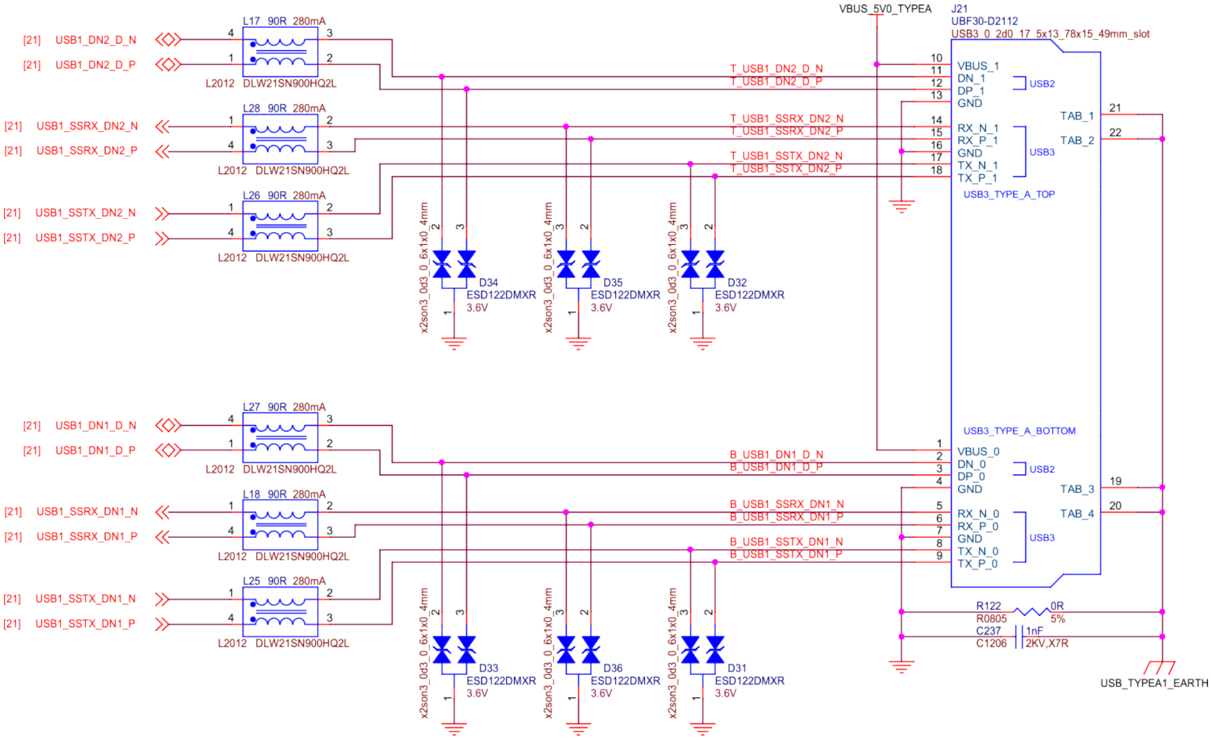


Fig. 3.36: BeagleY-AI dual USB2

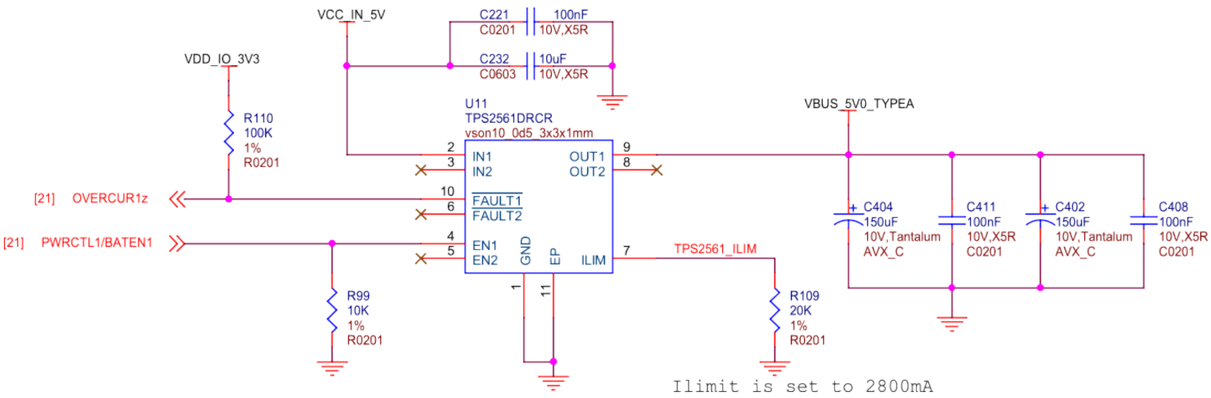


Fig. 3.37: BeagleY-AI dual USB current limiter

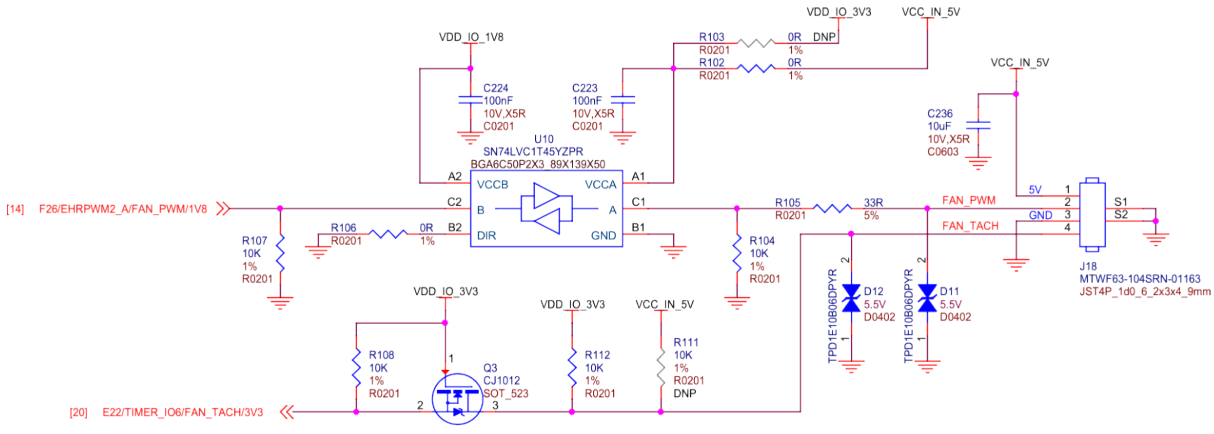


Fig. 3.38: BeagleY-AI fan connector

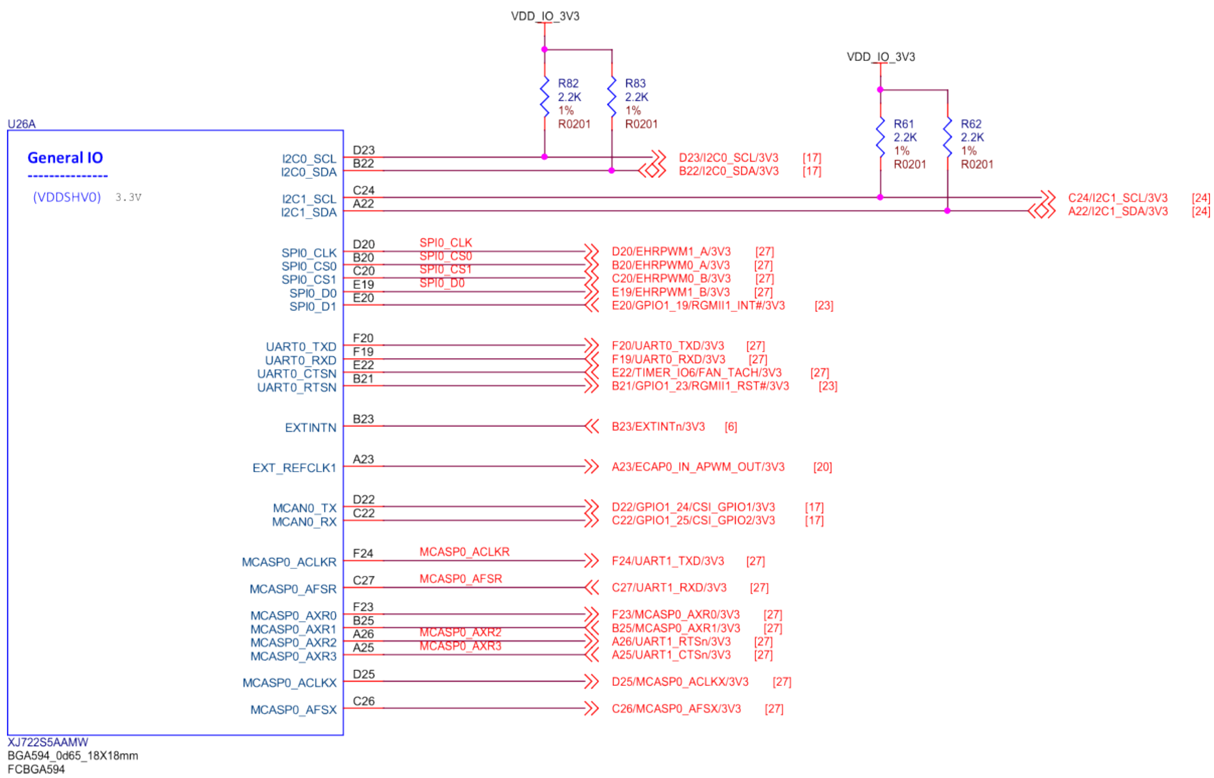


Fig. 3.39: BeagleY-AI general IO

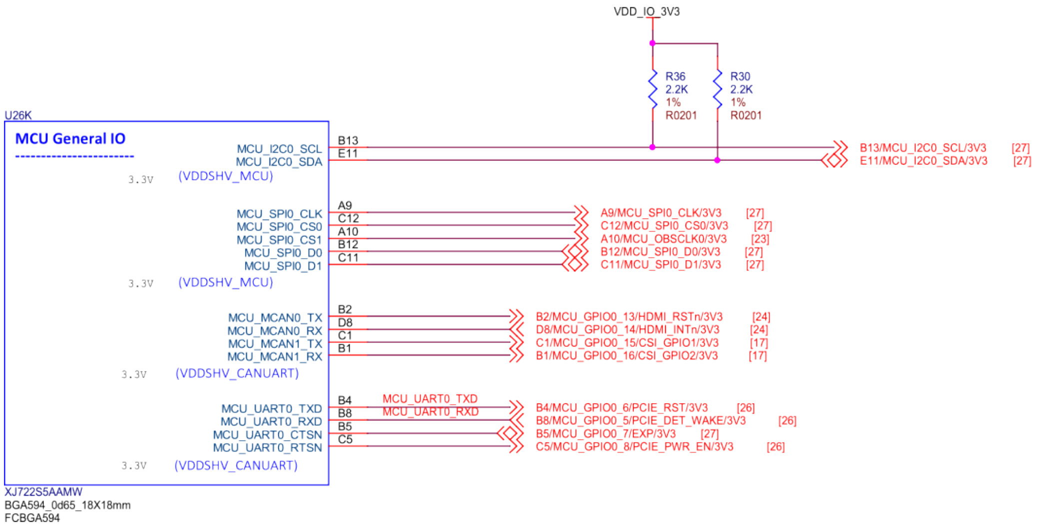


Fig. 3.40: BeagleY-AI MCU general IO

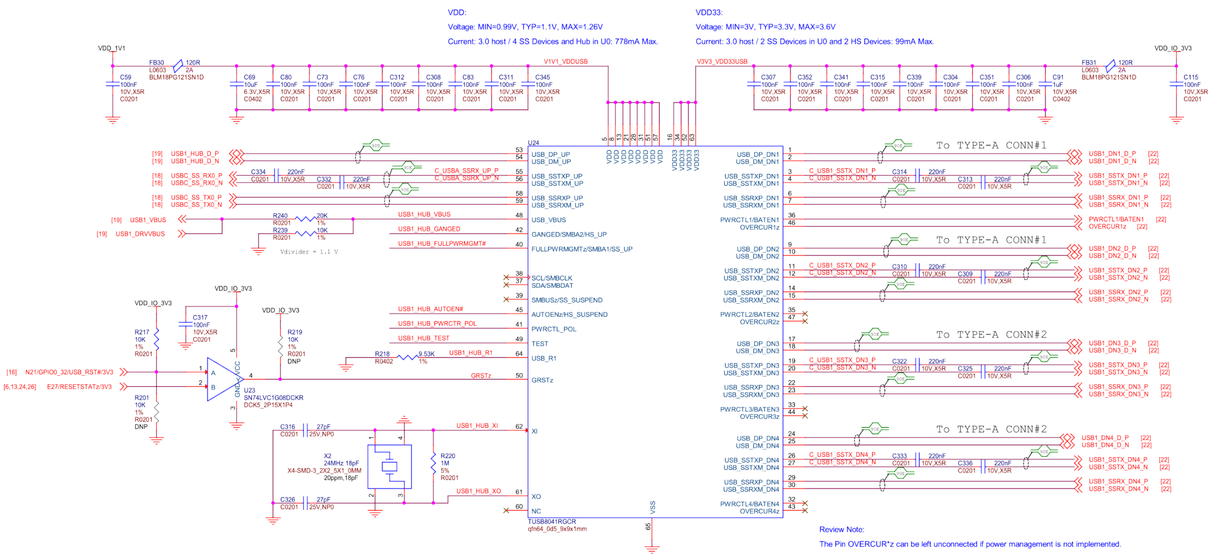


Fig. 3.41: BeagleY-AI USB3 hub

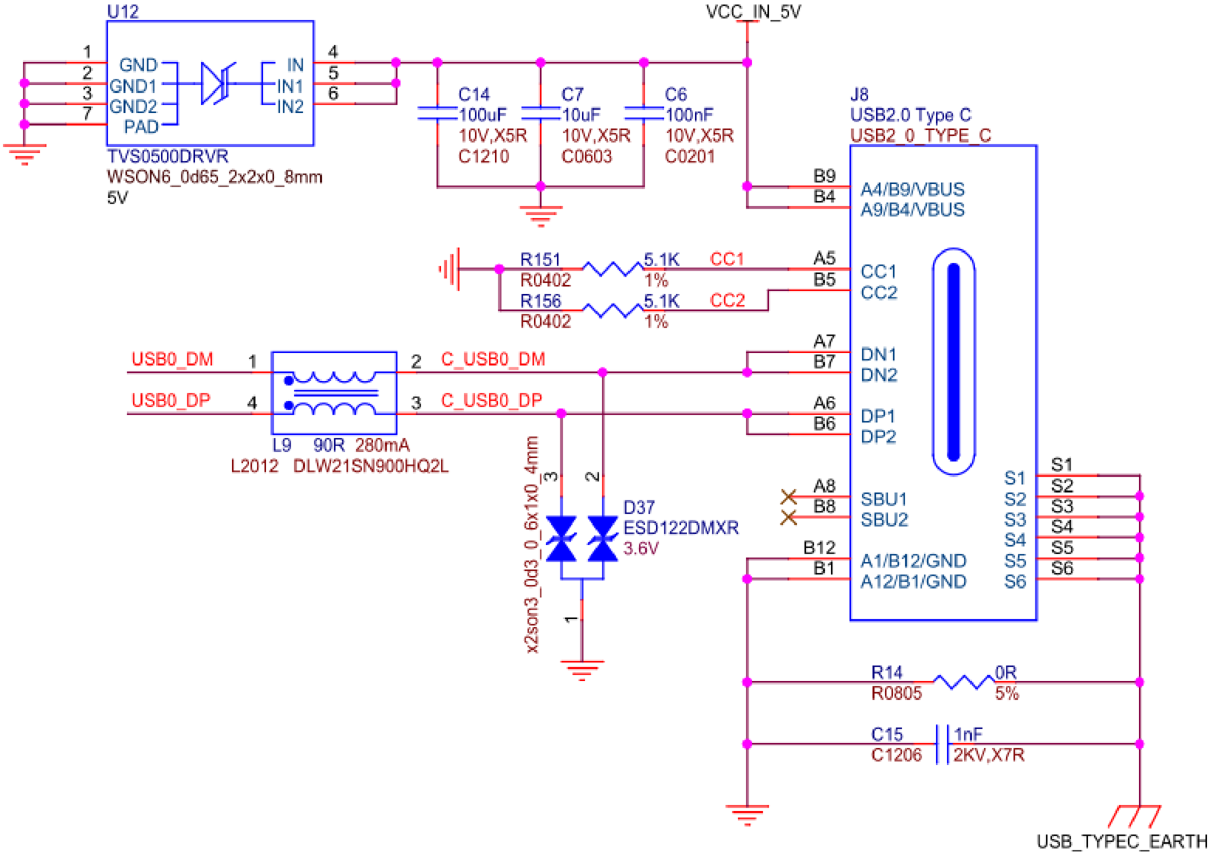


Fig. 3.42: BeagleY-AI USB-C

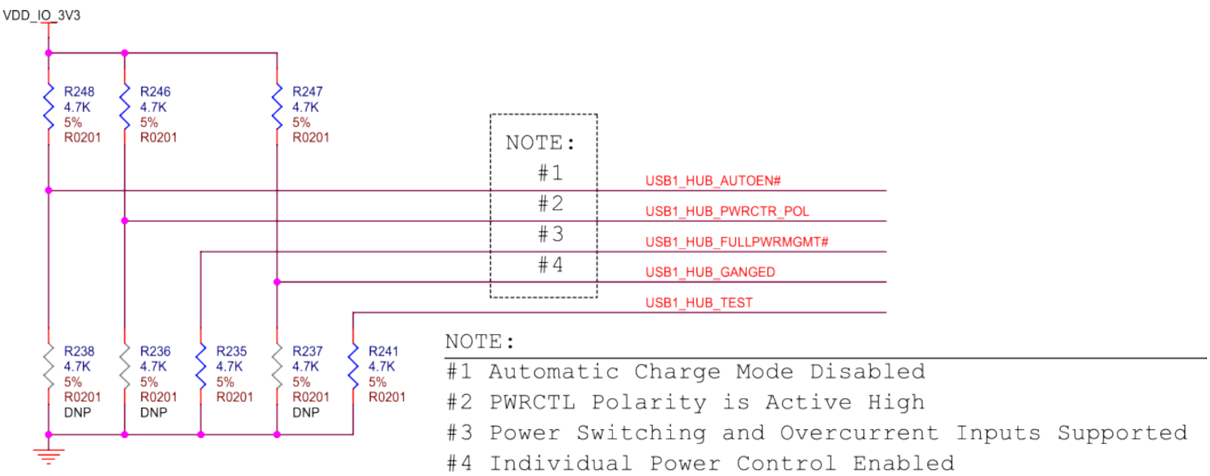


Fig. 3.43: BeagleY-AI USB hub config

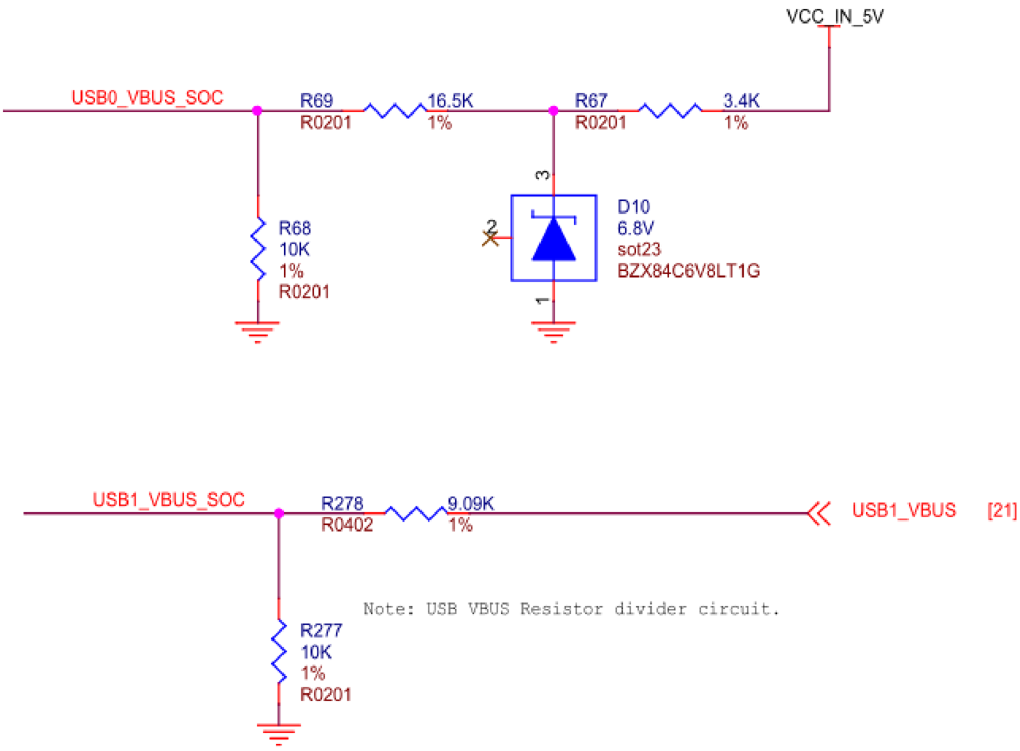


Fig. 3.44: BeagleY-AI USB VBUS resistor divider circuit

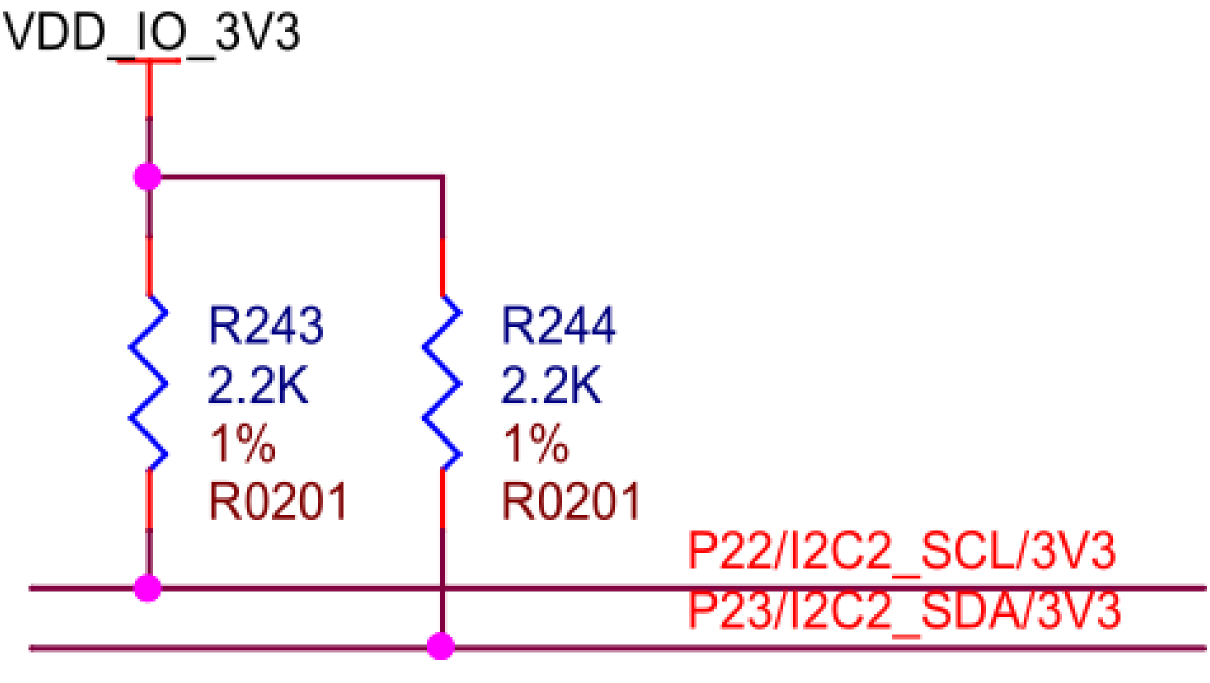


Fig. 3.45: BeagleY-AI I2C2 pull-up resistors

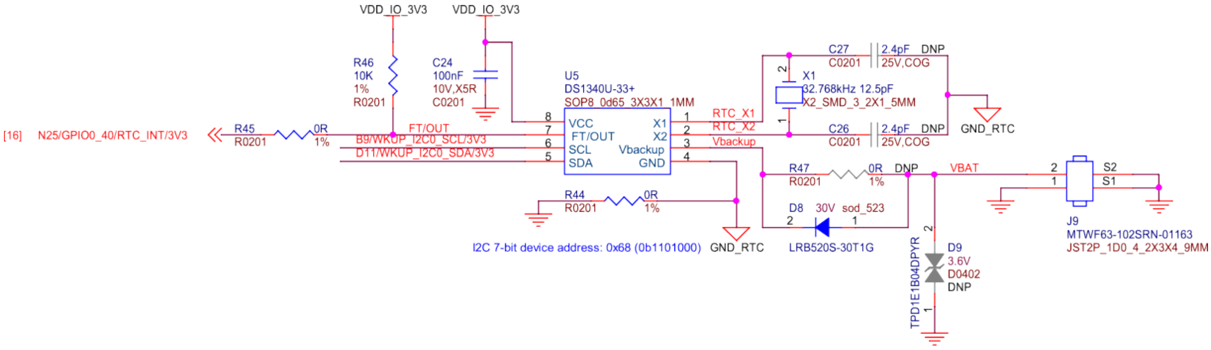


Fig. 3.46: BeagleY-AI I2C ext RTC

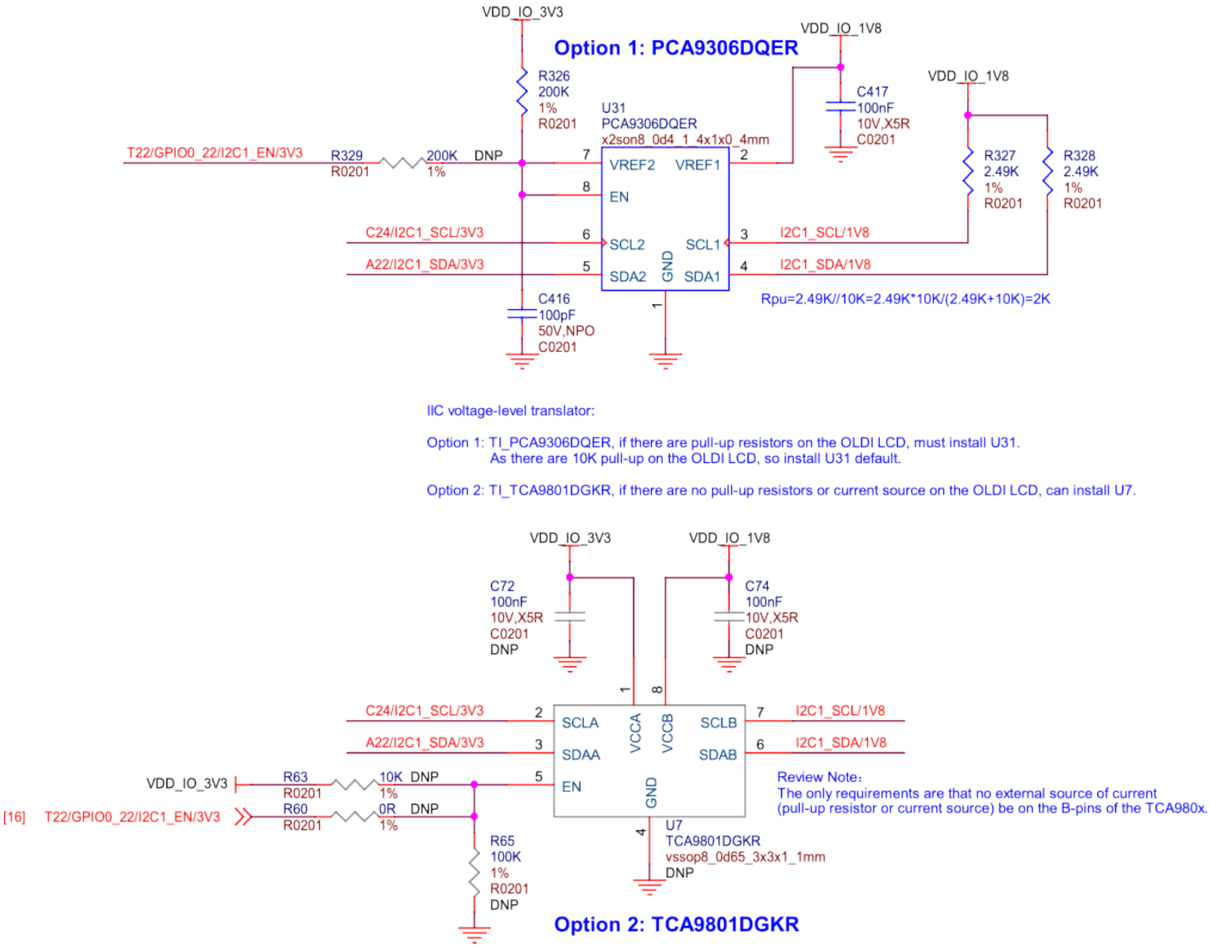


Fig. 3.47: BeagleY-AI voltage level translator

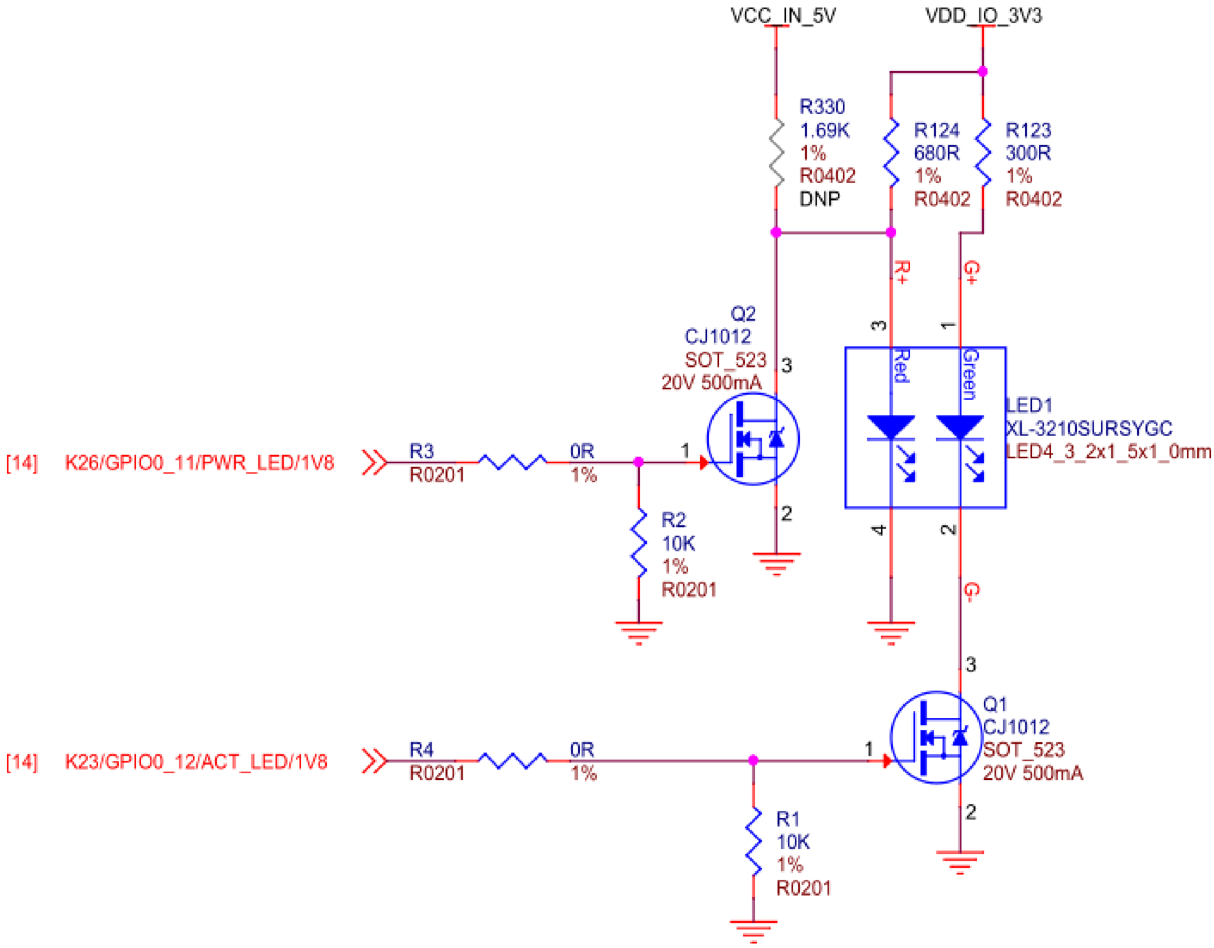


Fig. 3.48: BeagleY-AI LEDs

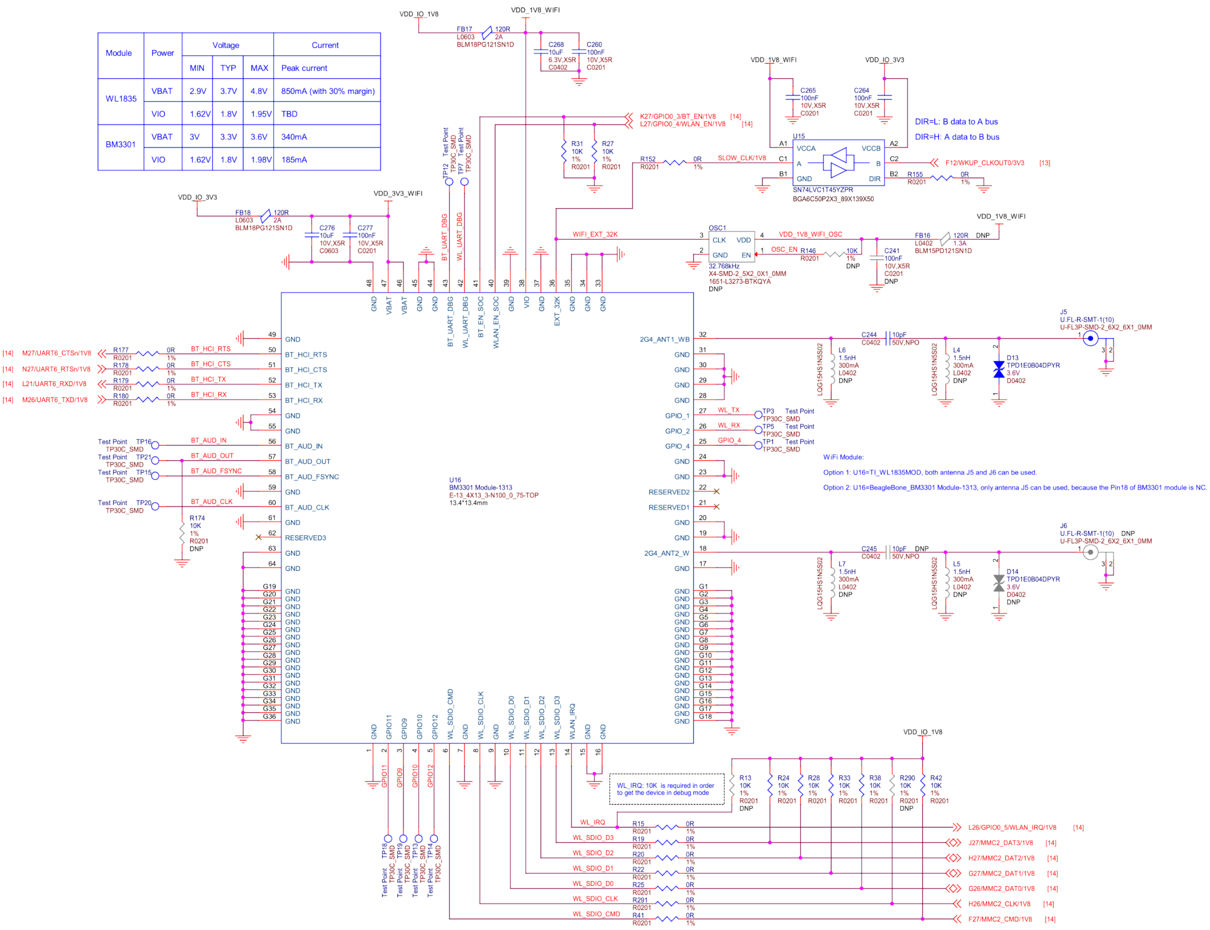


Fig. 3.49: BeagleY-AI WiFi module

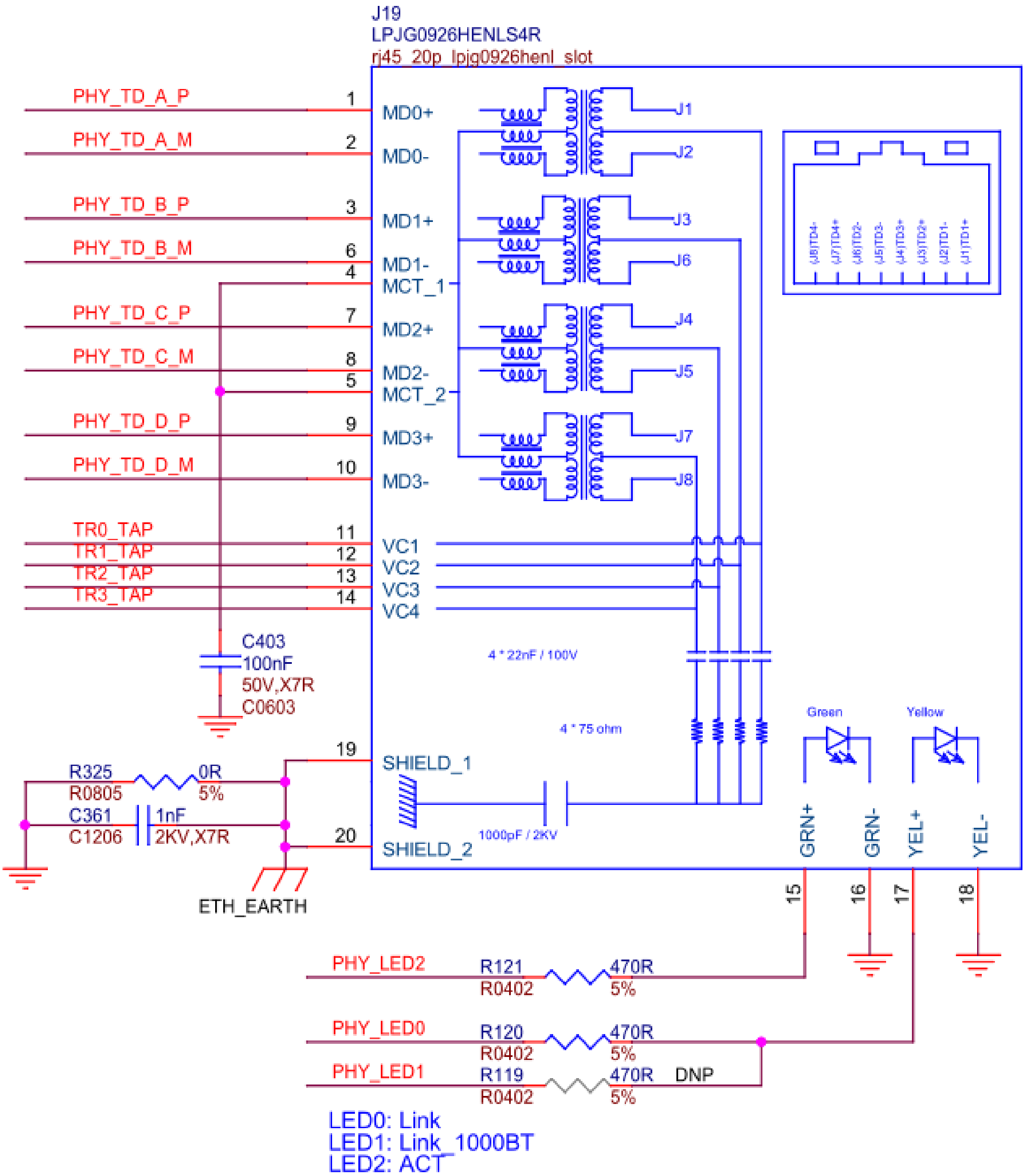


Fig. 3.50: BeagleY-AI ethernet connector

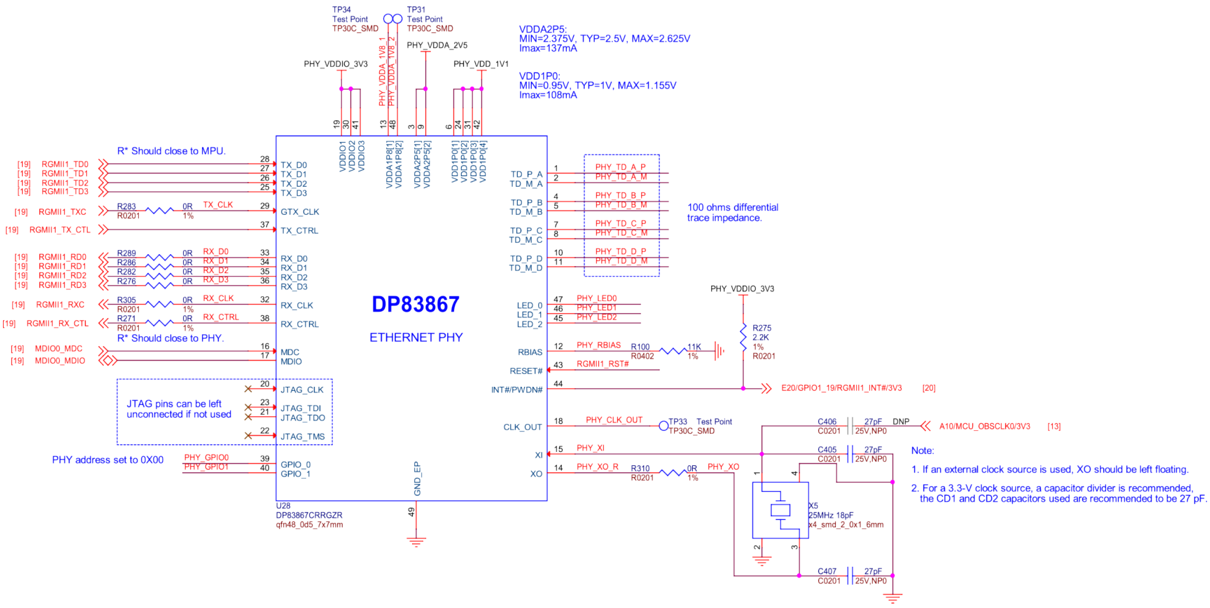


Fig. 3.51: BeagleY-AI ethernet DP83867

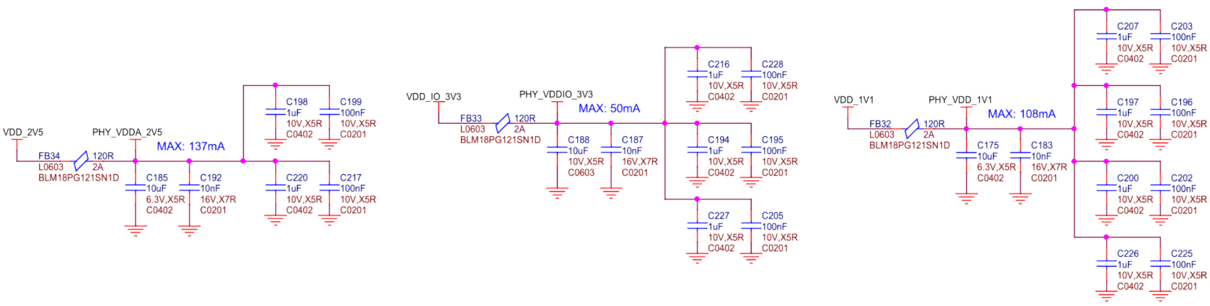


Fig. 3.52: BeagleY-AI ethernet phy caps

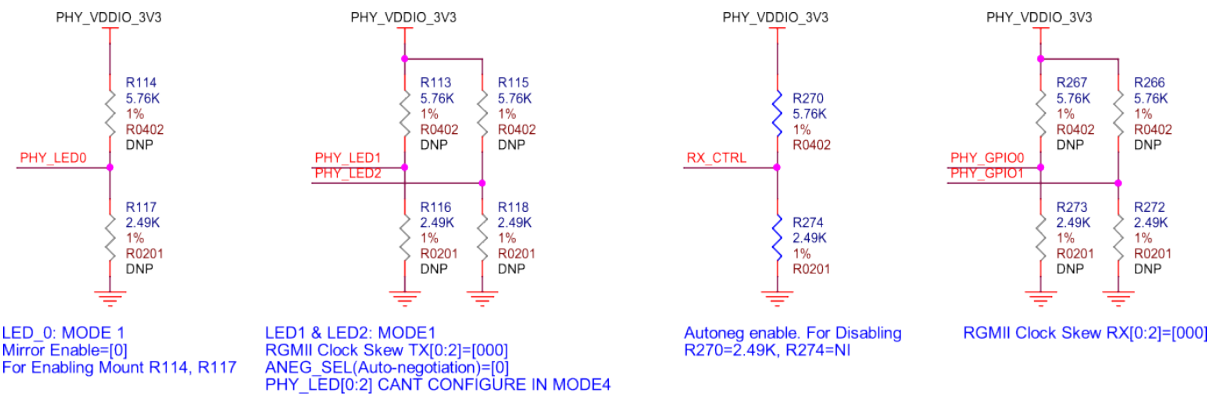


Fig. 3.53: BeagleY-AI ethernet phy misc

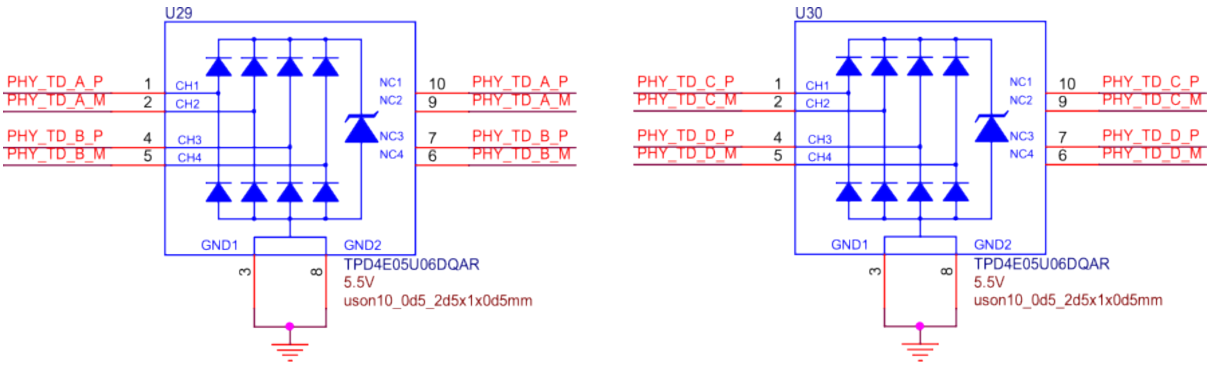


Fig. 3.54: BeagleY-AI ethernet phy protection

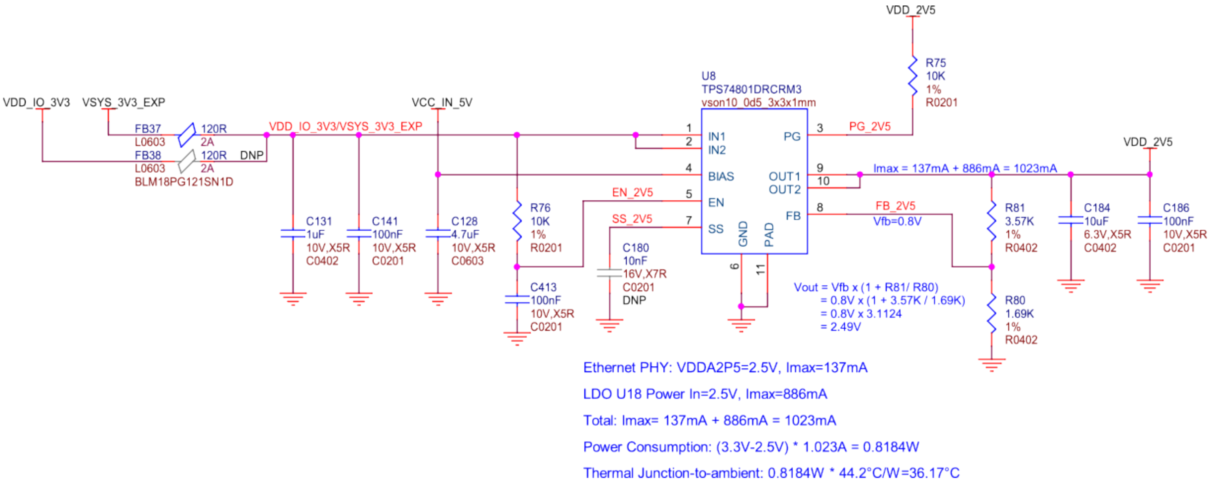


Fig. 3.55: BeagleY-AI ethernet power 3V3 to 2V5

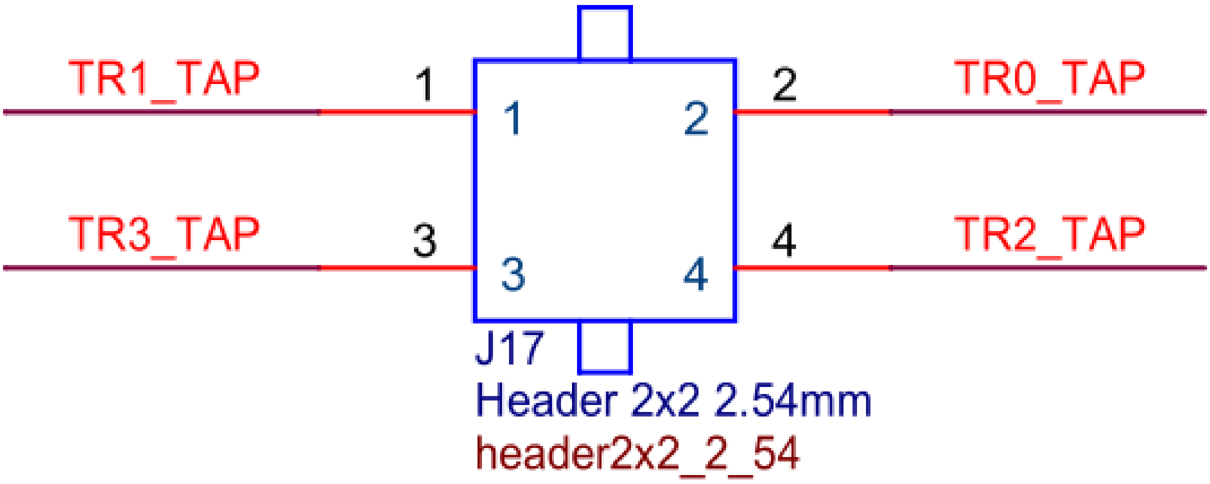


Fig. 3.56: BeagleY-AI PoE header

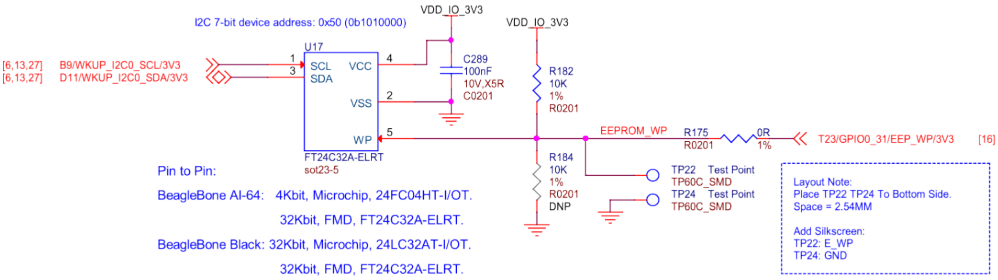


Fig. 3.57: BeagleY-AI board id eeprom

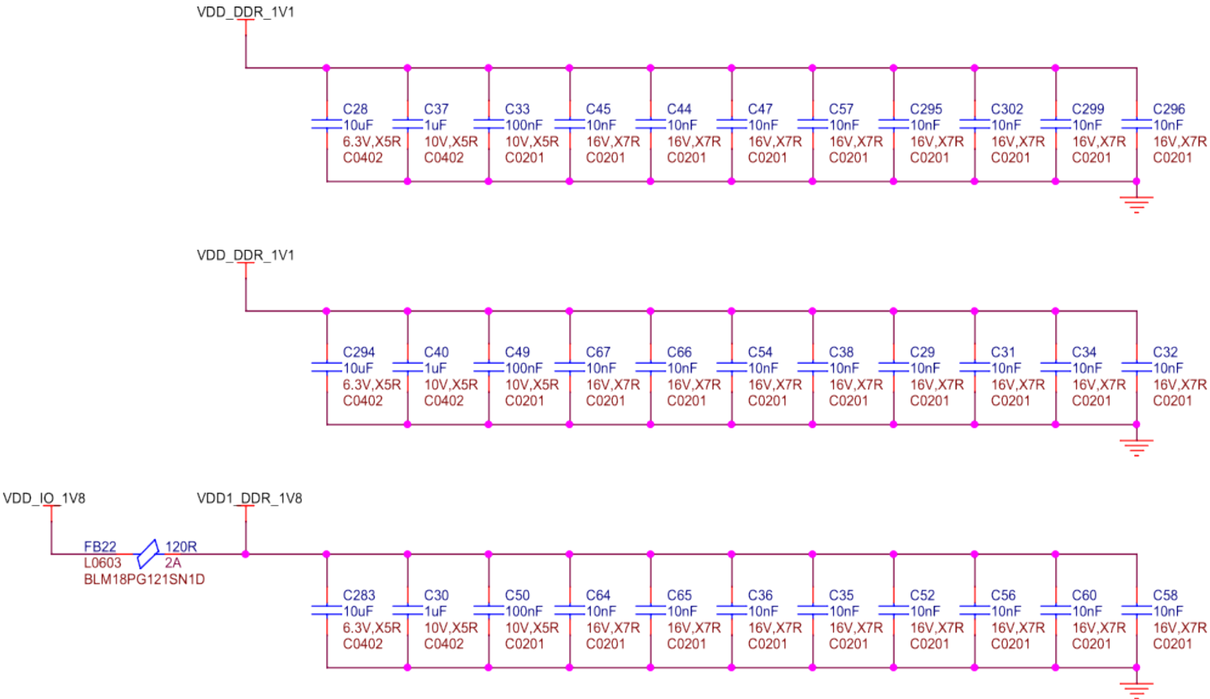


Fig. 3.58: BeagleY-AI DDR caps

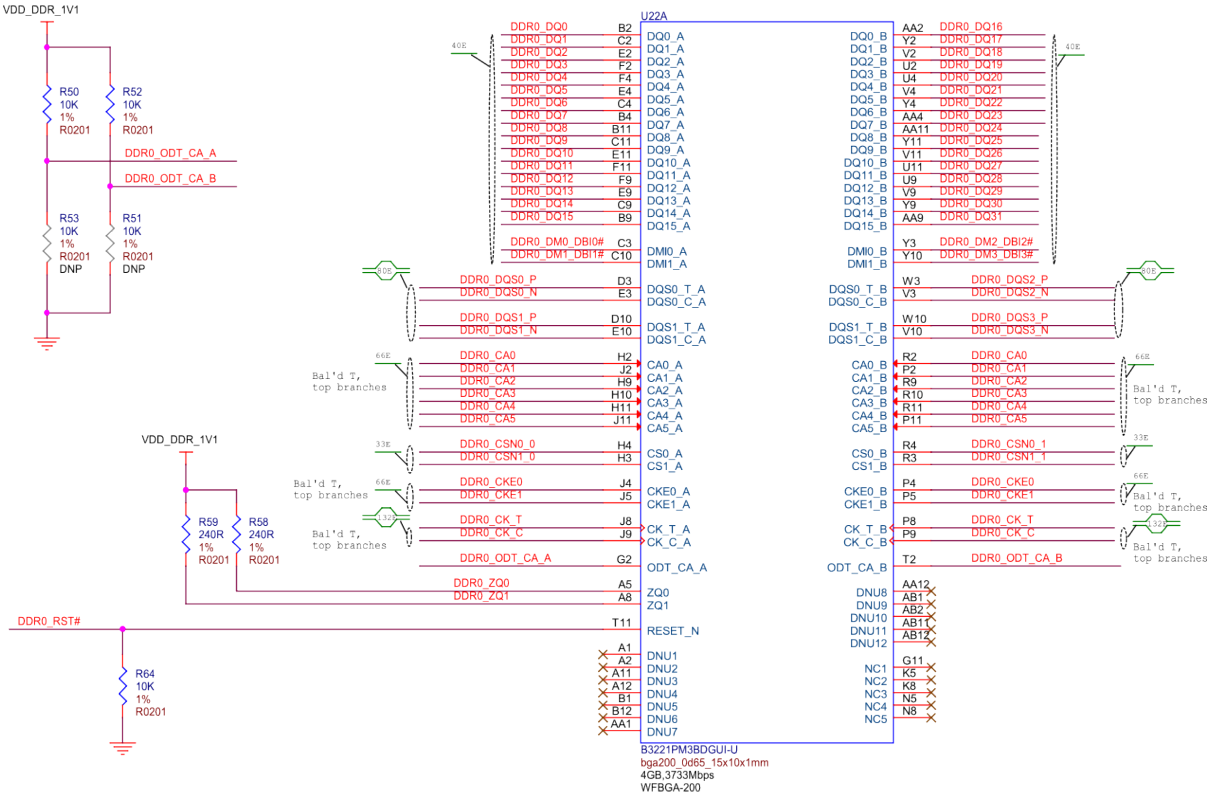


Fig. 3.59: BeagleY-AI DDR

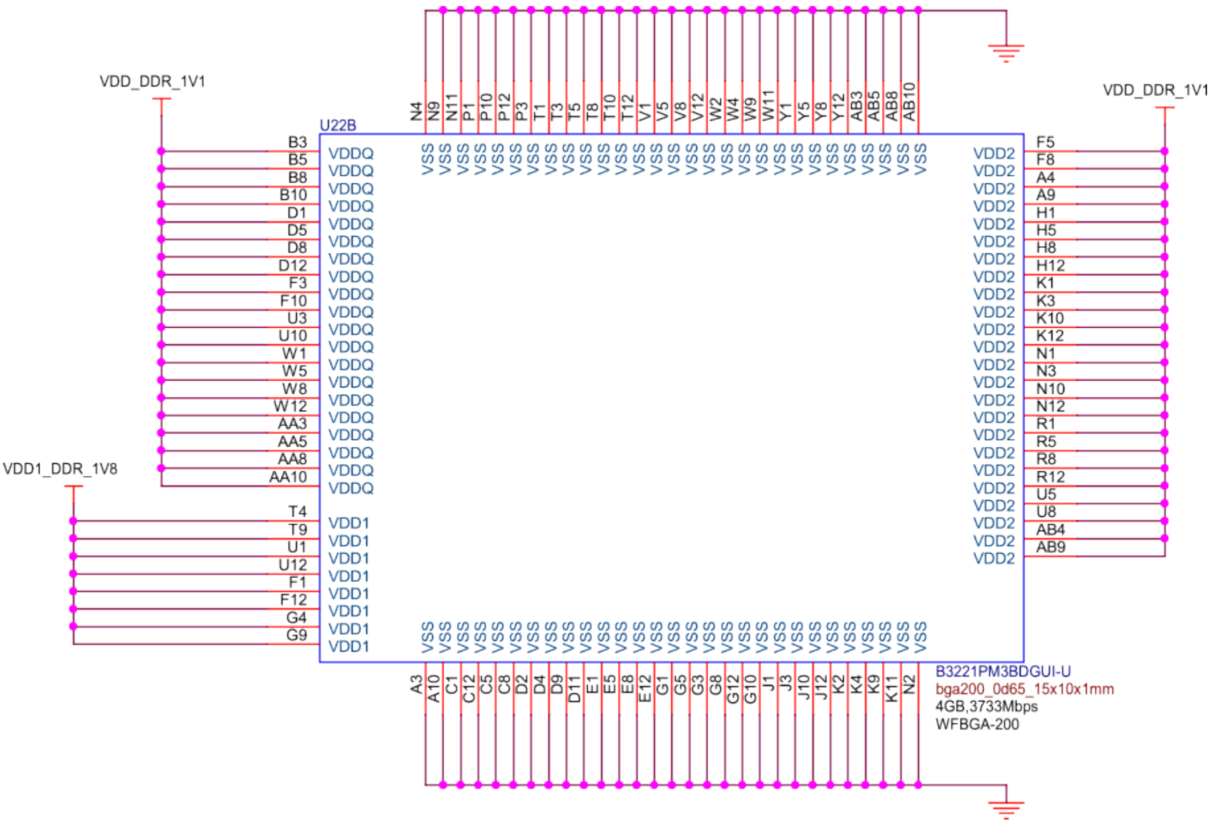


Fig. 3.60: BeagleY-AI DDR power

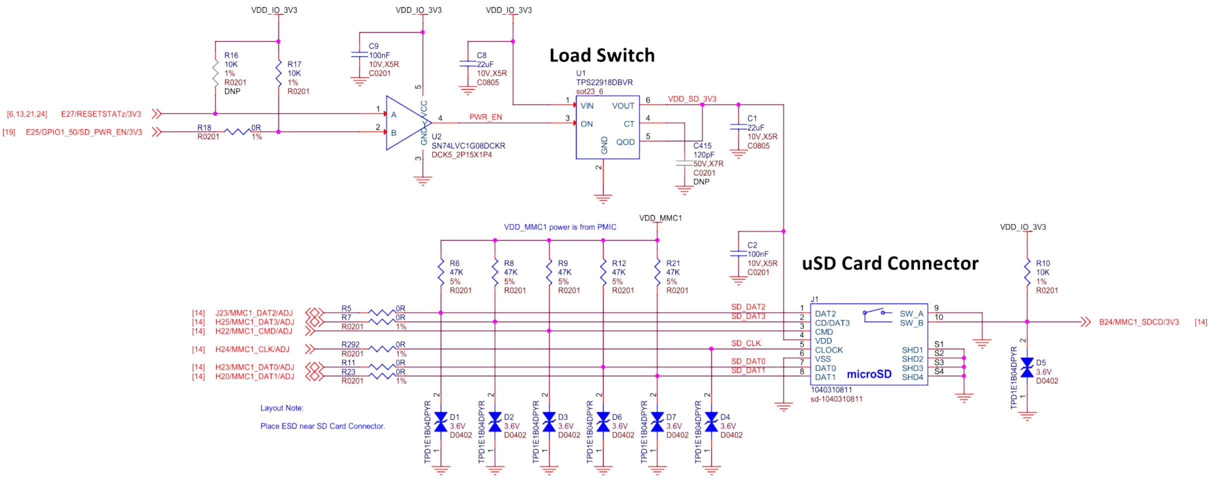


Fig. 3.61: BeagleY-AI microSD card interface

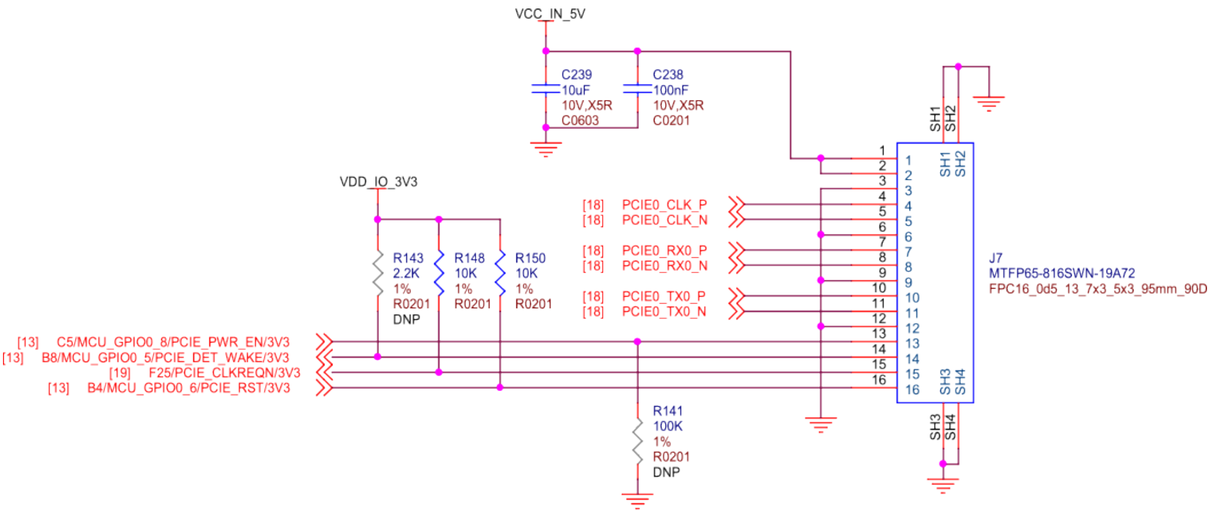


Fig. 3.62: BeagleY-AI PCIE connector

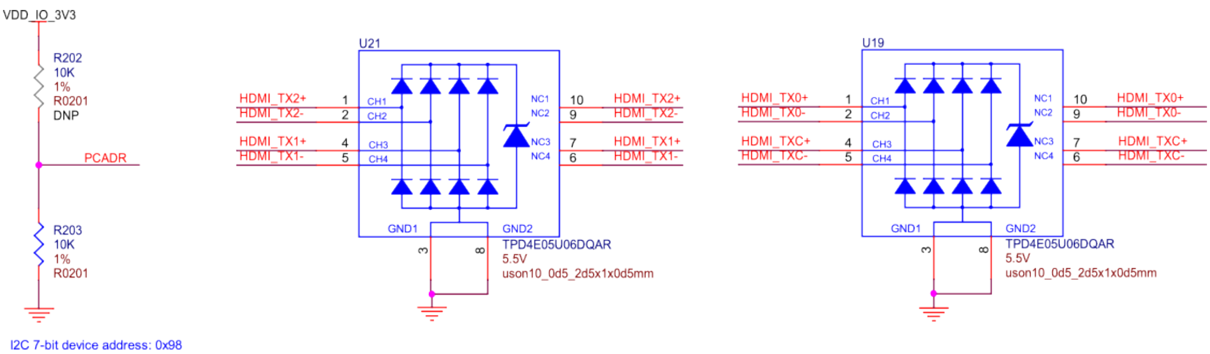


Fig. 3.63: BeagleY-AI HDMI addr protection

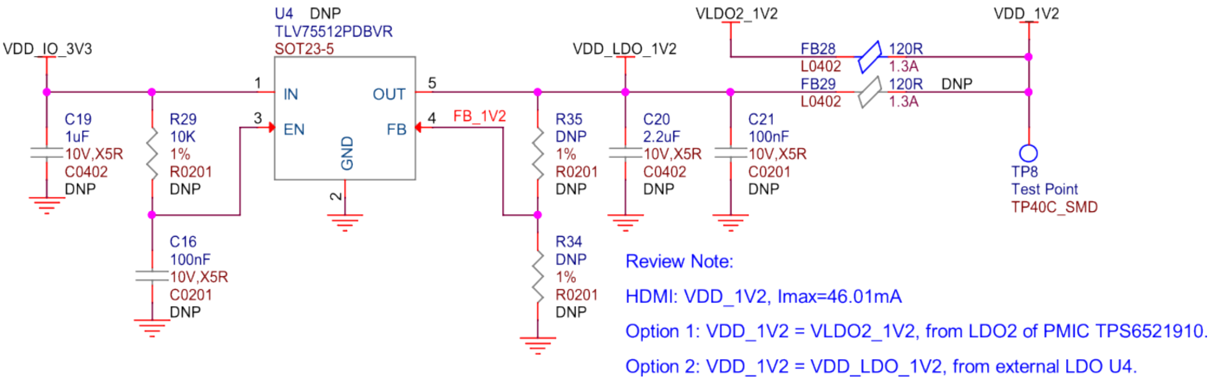


Fig. 3.64: BeagleY-AI HDMI power

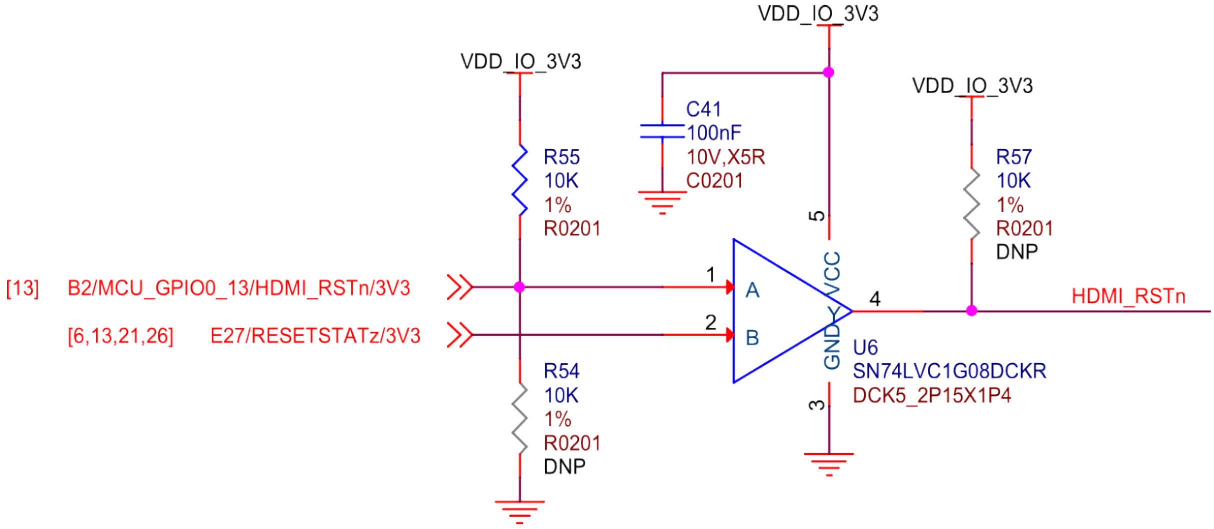


Fig. 3.65: BeagleY-AI HDMI reset

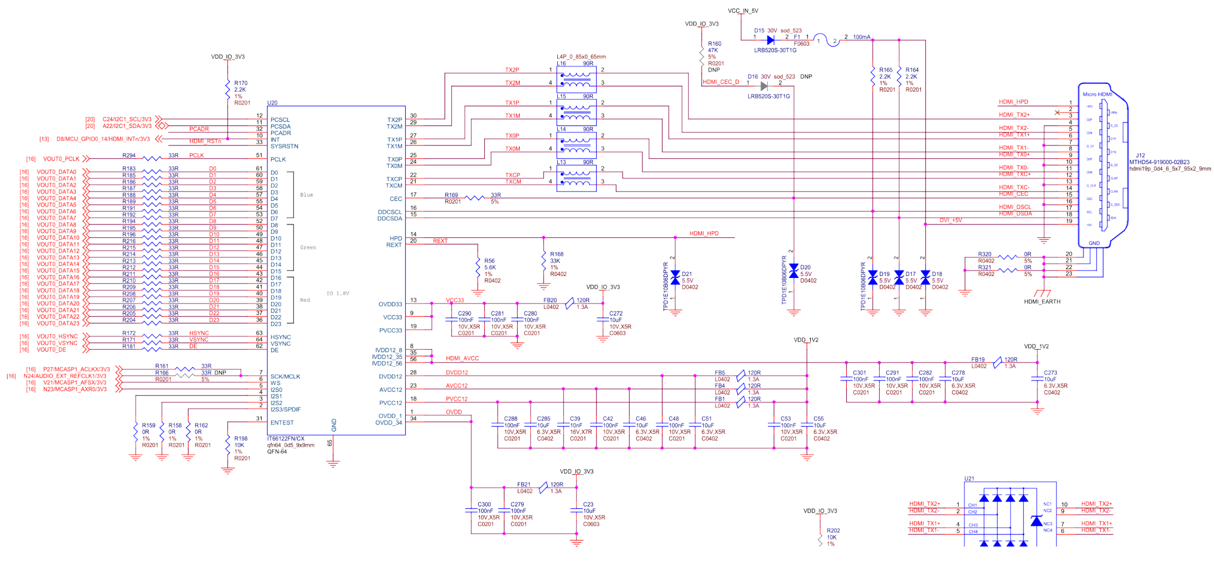


Fig. 3.66: BeagleY-AI RGB888 to HDMI

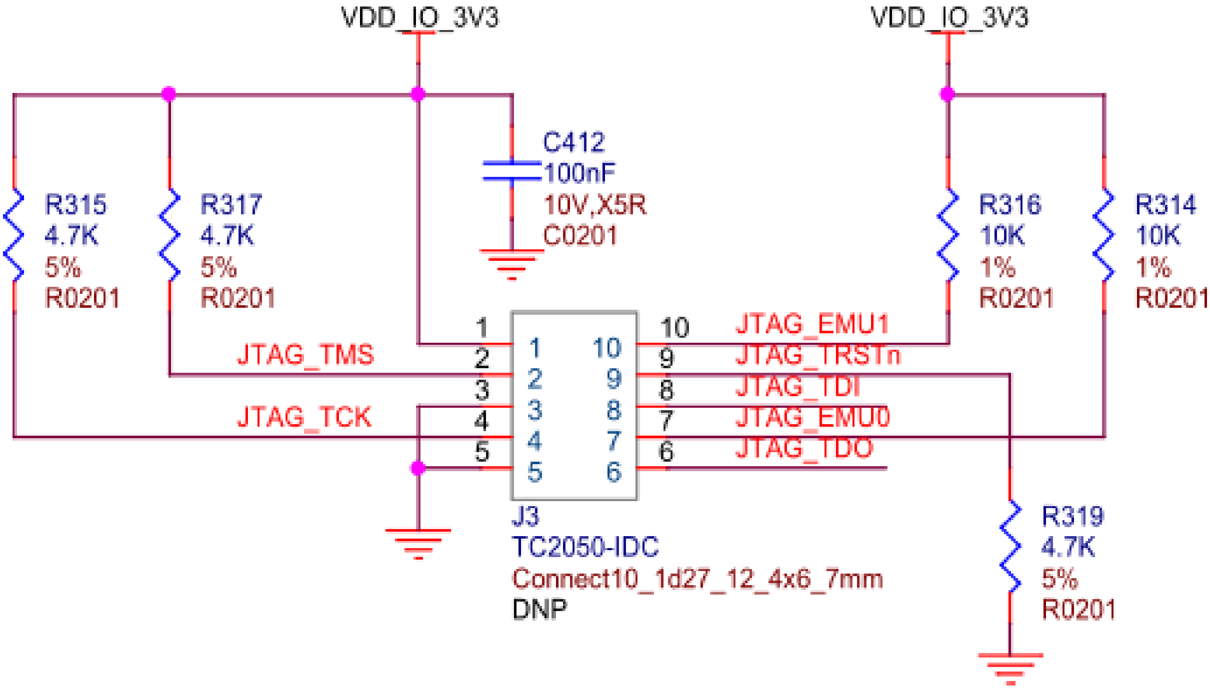


Fig. 3.67: BeagleY-AI Tag-Connect

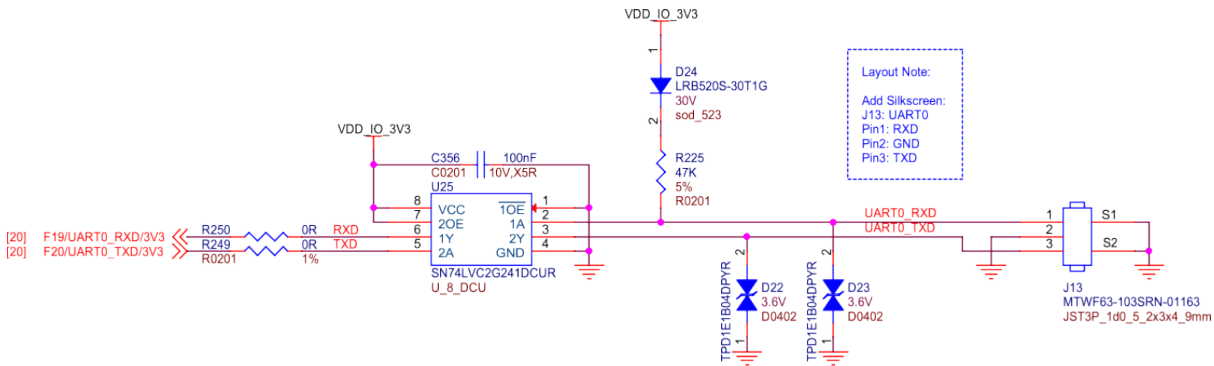


Fig. 3.68: BeagleY-AI debug UART port

Chapter 4

Expansion

Todo: Describe how to build expansion hardware for BeagleY-AI

4.1 PCIe

For software reference, you can see how PCIe is used on NVMe HATs.

- *Booting from NVMe Drives*
- *Using IMX219 CSI Cameras*
- *Using the on-board Real Time Clock (RTC)*

Chapter 5

Demos and tutorials

5.1 Using GPIO

Work in progress

Todo: Add information about software image used for this demo.

GPIO stands for **General-Purpose Input/Output**. It's a set of programmable pins that you can use to connect and control various electronic components.

You can set each pin to either **read signals (input)** from things like buttons and sensors or **send signals (output)** to things like LEDs and motors. This lets you interact with and control the physical world using code!

A great resource for understanding pin numbering can be found at pinout.beagley.ai

> **Warning:** BeagleY-AI GPIOs are 3.3V tolerant, using higher voltages **WILL DAMAGE** the processor!

5.1.1 Pin Numbering

You will see pins referenced in several ways. While this is confusing at first, in reality, we can pick our favorite way and stick to it.

The two main ways of referring to GPIOs is **by their number**, so GPIO 2, 3, 4 etc. as seen in the diagram below. This corresponds to the SoC naming convention. For broad compatibility, BeagleY-AI re-uses the Broadcom GPIO numbering scheme used by RaspberryPi.

The second (and arguably easier) way we will use for this tutorial is to use the **actual pin header number** (shown in dark grey)

So, for the rest of the tutorial, if we refer to **hat-08-gpio** we mean the **8th pin of the GPIO header**. Which, if you referenced the image below, can see refers to **GPIO 14 (UART TX)**

If you are curious about the "real" GPIO numbers on the Texas Instruments AM67A SoC, you can look at the board schematics.

5.1.2 Required Hardware

For the simple blink demo, all that is needed is an LED, a Resistor (we use 2.2K here) and 2 wires.

Similarly, a button is used for the GPIO read example, but you can also just connect that pin to 3.3V or GND with a wire to simulate a button press.

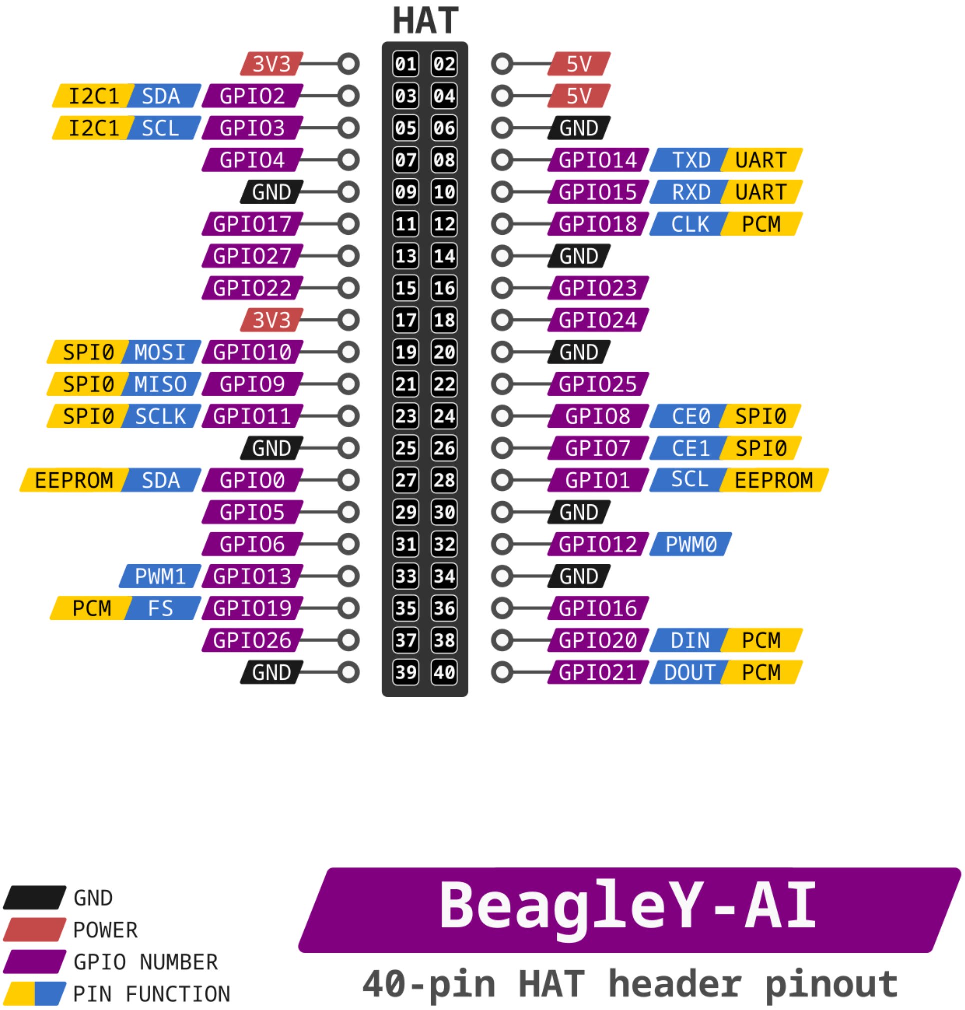


Fig. 5.1: BeagleY-AI pinout

Todo: Add fritzing diagram and chapter on Pin Binding here

5.1.3 GPIO Write

Before using any pin with HAT Pin number we need to configure it using command below,

```
sudo beagle-pin-mux --pin hat-08 --mode gpio
```

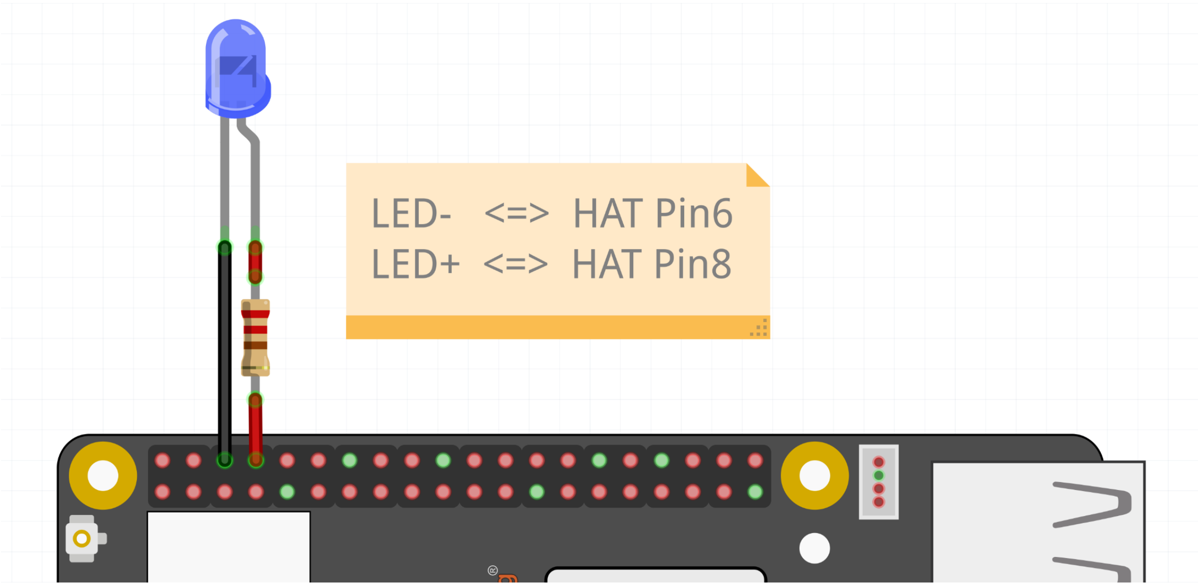


Fig. 5.2: LED connected to HAT Pin8

At it's most basic, we can set a GPIO using the **gpioset** command.

- To set HAT **Pin 8** to **ON**:

```
gpioset hat-08-gpio 0=1
```

- To set HAT **Pin 8** to **OFF**:

```
gpioset hat-08-gpio 0=0
```

5.1.4 Blink an LED

Let's create a script called **blinky.sh**,

- Create the file,

```
touch blinky.sh
```

- Open the file using `nano` editor,

```
nano blinky.sh
```

- Copy paste the code below to `blinky.sh` file,

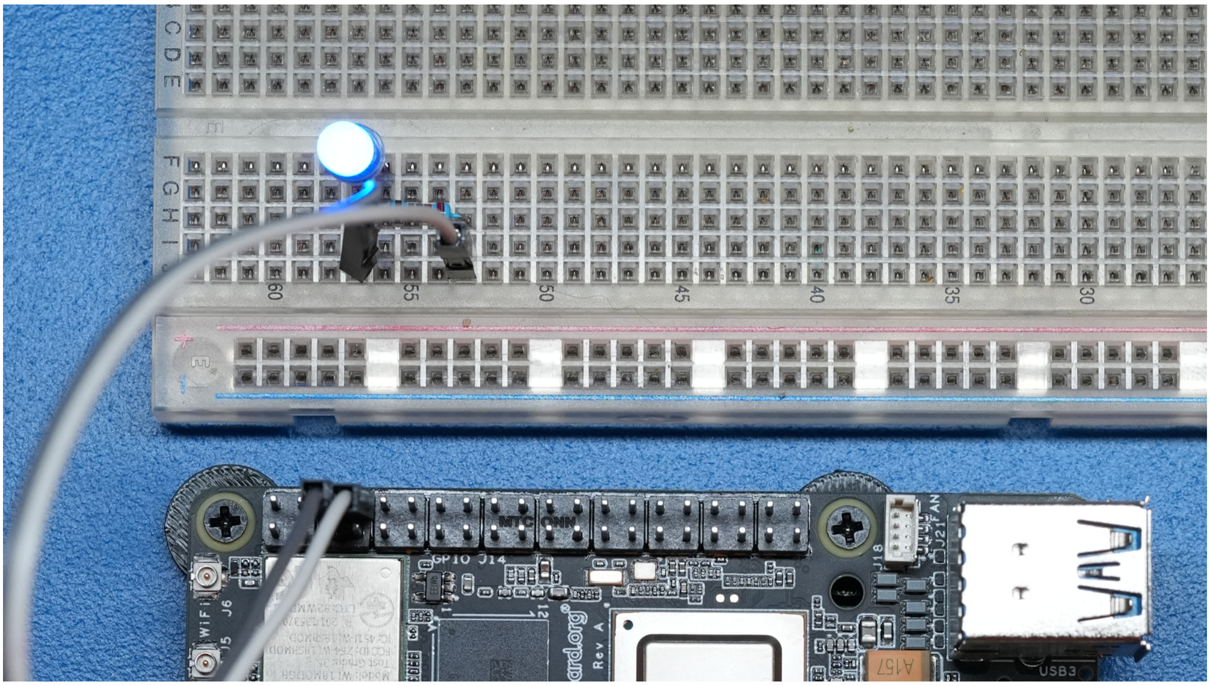


Fig. 5.3: GPIO ON state

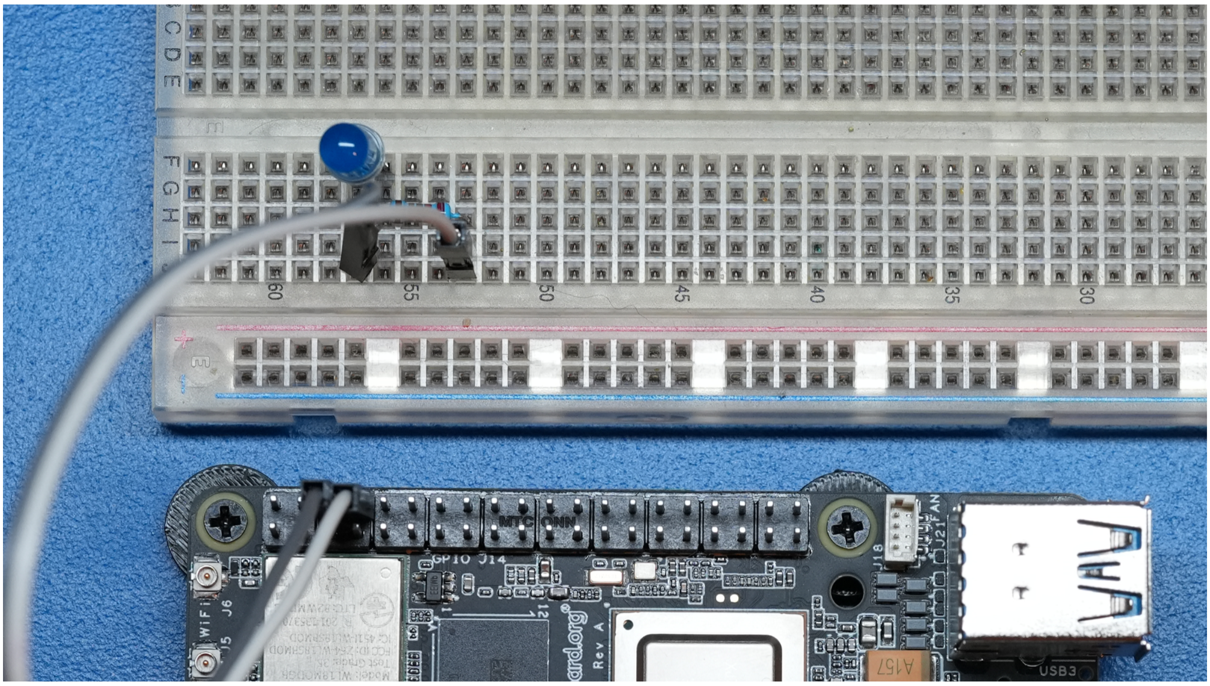


Fig. 5.4: GPIO OFF state

```
#!/bin/bash

while :
do
        gpioset hat-08-gpio 0=1
        sleep 1
        gpioset hat-08-gpio 0=0
        sleep 1
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `blinky.sh` script by typing:

```
bash blinky.sh
```

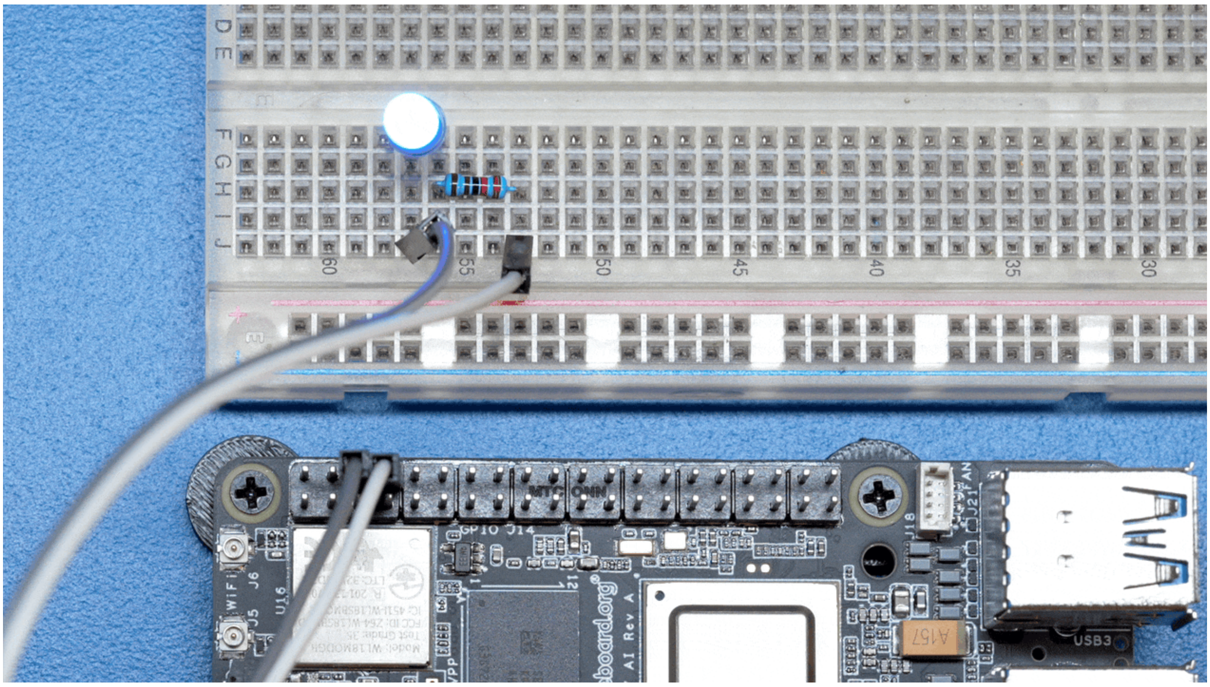


Fig. 5.5: LED blinking

- You can exit the `blinky.sh` progrm by pressing `CTRL + C` on your keyboard.

Understanding the code

```
#!/bin/bash

while :
do
   gpioset hat-08-gpio 0=1 ①
   sleep 1 ②
   gpioset hat-08-gpio 0=0 ③
   sleep 1 ④
done
```

The script is an infinite `while` loop in which we do the following:

① set the HAT Pin 8 as 1 (HIGH)

② Wait 1 Second

③ set the HAT Pin 8 as 0 (LOW)

④ Wait 1 Second

5.1.5 Read a Button

A push button simply completes an electric circuit when pressed. Depending on wiring, it can drive a signal either "Low" (GND) or "High" (3.3V).

We will connect our Button between HAT Pin 12 (GPIO18) and Ground (GND).

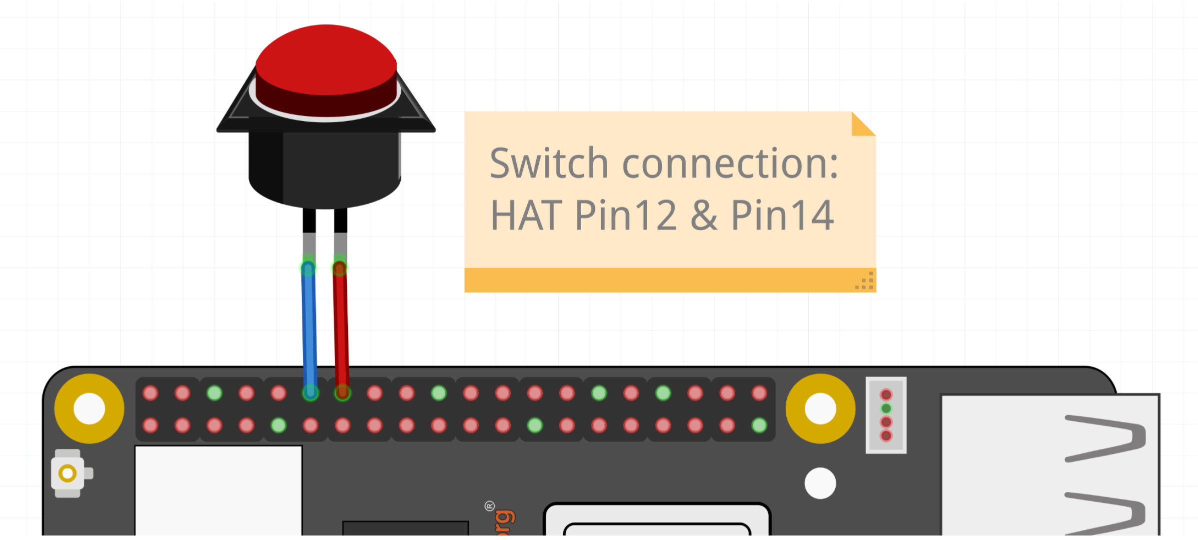


Fig. 5.6: Button connected to HAT Pin12

- Configure pin12 as `gpio` using command below,

```
sudo beagle-pin-mux --pin hat-12 --mode gpio-pu
```

The cool part is since we have an internal pull-up resistor, we don't need an external one! The pull resistor guarantees that the Pin stays in a known (HIGH) state unless the button is pressed, in which case it will go LOW.

- Reading GPIOs can be done using the `gpioget` command

```
gpioget hat-12-gpio-pu 0
```

Results in `1` if the Input is held `HIGH` or `0` if the Input is held `LOW`

Let's create a script called `button.sh` to continuously read an input pin connected to a button and print out when it's pressed!

- Create the file,

```
touch button.sh
```

- Open the file using `nano` editor,

```
nano button.sh
```

- Copy paste the code below to `button.sh` file,

```
#!/bin/bash

while :
do
        if (( $(gpioget hat-12-gpio-pu 0) == 0))
        then
             echo "Button Pressed!"
        fi
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `button.sh` script by typing:

```
bash button.sh
```

- You can exit the `button.sh` by pressing `Ctrl + C` on your keyboard.

5.1.6 Combining the Two

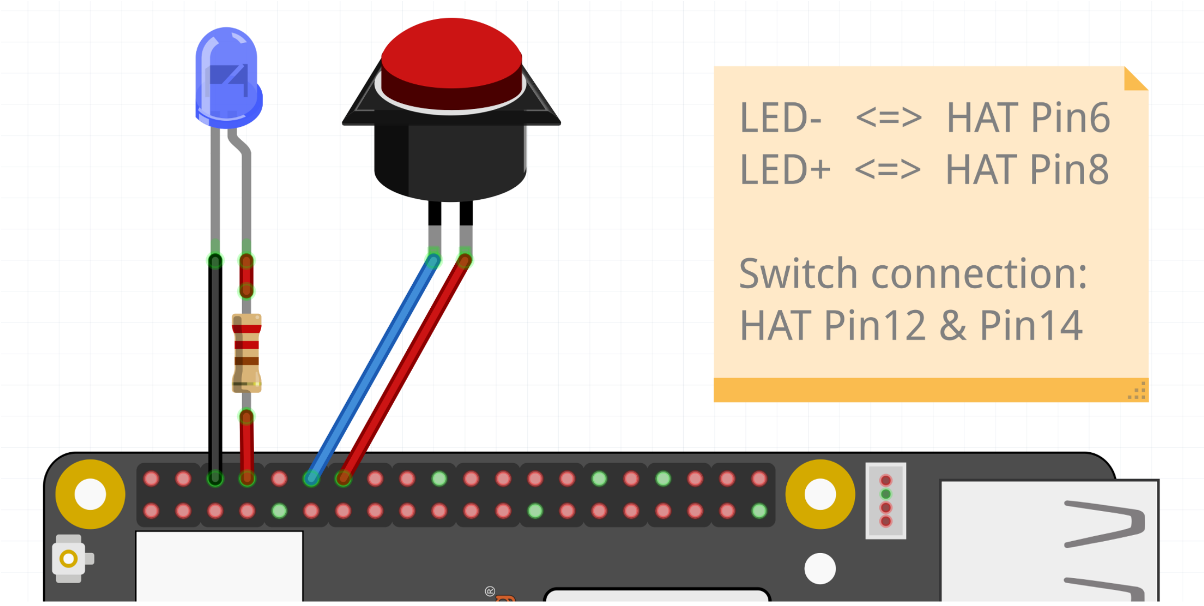


Fig. 5.7: Button connected to HAT Pin12 & LED connected to HAT Pin8

Now, logically, let's make an LED match the state of the button.

Let's create a script called **blinkyButton.sh**:

- Create the file,

```
touch blinkyButton.sh
```

- Open the file using `nano` editor,

```
nano blinkyButton.sh
```

- Copy paste the code below to `blinkyButton.sh` file,

```
#!/bin/bash

while :
   do
           if (( $(gpioget hat-12-gpio-pu 0) == 0))
           then
                   gpioset hat-08-gpio 0=1
           else
                   gpioset hat-08-gpio 0=0
           fi
   done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `blinkyButton.sh` script by typing:

```
bash blinkyButton.sh
```

This means when we see HAT Pin 12 go LOW, we know the button is pressed, so we set HAT Pin 8 (our LED) to ON, otherwise, we turn it OFF.

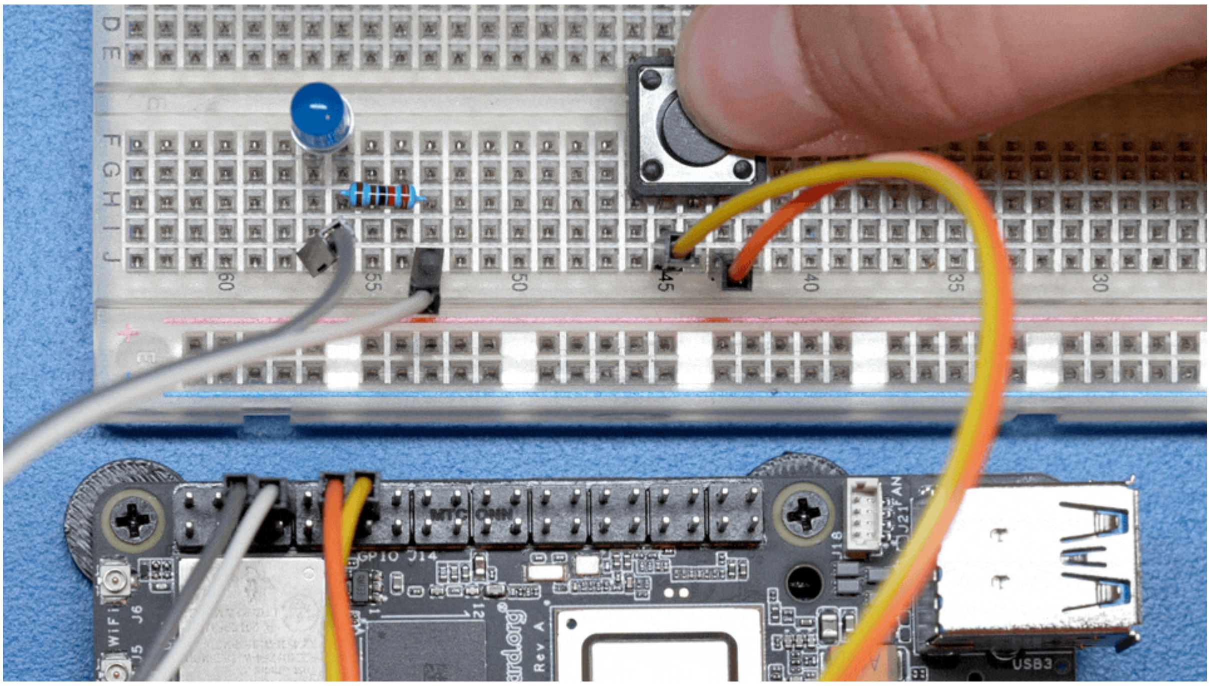


Fig. 5.8: LED is ON when button is pressed

- You can exit the `blinkyButton.sh` program by pressing `Ctrl + C` on your keyboard.

5.1.7 Understanding Internal Pull Resistors

Pull-up and pull-down resistors are used in digital circuits to ensure that inputs to logic settle at expected levels.

- `Internal pull-up resistors` connects the pin to a high voltage level (e.g., 3.3V) to ensure the pin input reads as a logic high (1) when no active device is pulling it low.
- `Internal pull-down resistors` connects the pin to ground (GND) to ensure the input reads as a logic low (0) when no active device is pulling it high.

These resistors prevent floating inputs and undefined states.

By default, all GPIOs on the HAT Header are configured as **Inputs with Pull-up Resistors Enabled**.

This is important for something like a button, as without it, once a button is released, it goes in an "undefined" state!

To configure Pull-ups on a per-pin basis, we can use pass the following arguments within **gpioget or gpioset**:

```
-B, --bias=[as-is|disable|pull-down|pull-up] (defaults to 'as-is')
```

The "Bias" argument has the following options:

- **as-is** - This leaves the bias as-is... quite self explanatory
- **disable** - This state is also known as High-Z (high impedance) where the Pin is left Floating without any bias resistor
- **pull-down** - In this state, the pin is pulled DOWN by the internal 50KΩ resistor
- **pull-up** - In this state, the pin is pulled UP by the internal 50KΩ resistor

For example, a command to read an input with the Bias intentionally disabled would look like this:

```
gpioget --bias=disable hat-08-gpio 0
```

Pull resistors are a foundational block of digital circuits and understanding when to (and not to) use them is important.

This article from SparkFun Electronics is a good basic primer - Link

5.1.8 Troubleshooting

- **My script won't run!**

Make sure you gave the script execute permissions first and that you're executing it with a `./` before

- To make it executable:

```
chmod +X scriptName.sh
```

- To run it:

```
./scriptName.sh
```

5.1.9 Bonus - Turn all GPIOs ON/OFF

- Copy and paste this with the button on the right to turn **all pins ON**.

```
gpioset hat-03-gpio 0=1 ;\ gpioset hat-05-gpio 0=1 ;\ gpioset hat-08-gpio␣
↪0=1 ;\ gpioset hat-10-gpio 0=1 ;\ gpioset hat-11-gpio 0=1 ;\ gpioset hat-12-
↪gpio 0=1 ;\ gpioset hat-13-gpio 0=1 ;\ gpioset hat-15-gpio 0=1 ;\ gpioset␣
↪hat-16-gpio 0=1 ;\ gpioset hat-18-gpio 0=1 ;\ gpioset hat-19-gpio 0=1 ;\␣
↪gpioset hat-21-gpio 0=1 ;\ gpioset hat-22-gpio 0=1 ;\ gpioset hat-23-gpio␣
↪0=1 ;\ gpioset hat-24-gpio 0=1 ;\ gpioset hat-26-gpio 0=1 ;\ gpioset hat-29-
↪gpio 0=1 ;\ gpioset hat-31-gpio 0=1 ;\ gpioset hat-32-gpio 0=1 ;\ gpioset␣
↪hat-33-gpio 0=1 ;\ gpioset hat-35-gpio 0=1 ;\ gpioset hat-36-gpio 0=1 ;\␣
↪gpioset hat-37-gpio 0=1 ;\ gpioset hat-40-gpio 0=1
```

- Similarly, copy and paste this to turn **all pins OFF**.

```
gpioset hat-03-gpio 0=0 ;\ gpioset hat-05-gpio 0=0 ;\ gpioset hat-08-gpio␣
↪0=0 ;\ gpioset hat-10-gpio 0=0 ;\ gpioset hat-11-gpio 0=0 ;\ gpioset hat-12-
↪gpio 0=0 ;\ gpioset hat-13-gpio 0=0 ;\ gpioset hat-15-gpio 0=0 ;\ gpioset␣
↪hat-16-gpio 0=0 ;\ gpioset hat-18-gpio 0=0 ;\ gpioset hat-19-gpio 0=0 ;\␣
```

(continues on next page)

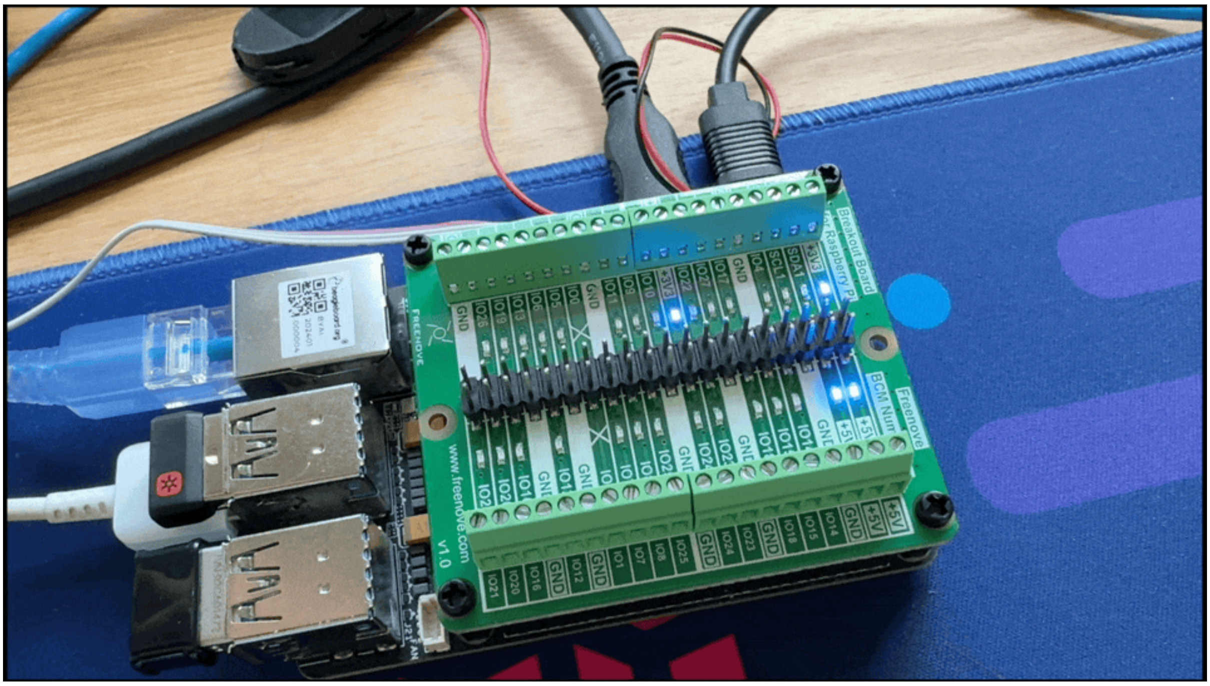


Fig. 5.9: All HAT GPIO toggle

(continued from previous page)

```
↪gpioset hat-21-gpio 0=0 ;\ gpioset hat-22-gpio 0=0 ;\ gpioset hat-23-gpio␣
↪0=0 ;\ gpioset hat-24-gpio 0=0 ;\ gpioset hat-26-gpio 0=0 ;\ gpioset hat-29-
↪gpio 0=0 ;\ gpioset hat-31-gpio 0=0 ;\ gpioset hat-32-gpio 0=0 ;\ gpioset␣
↪hat-33-gpio 0=0 ;\ gpioset hat-35-gpio 0=0 ;\ gpioset hat-36-gpio 0=0 ;\␣
↪gpioset hat-37-gpio 0=0 ;\ gpioset hat-40-gpio 0=0
```

5.1.10 Going Further

- pinout.beagley.ai
- GPIOSet Documentation
- GPIOGet Documentation

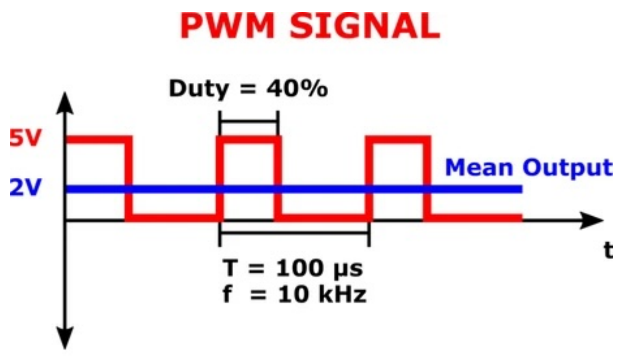
5.2 Pulse Width Modulation (PWM)

Work in progress

Todo: Add further testing steps, results, and images..

5.2.1 What is it

`PWM` or `Pulse Width Modulation`, is a technique used to control the amount of power delivered to an electronic device by breaking up the power signal into discrete ON and OFF periods. The amount of time the signal spends ON during each cycle determines the output power level (brightness of the LED).



5.2.2 How do we do it

To configure HAT pin8 as PWM pin using `beagle-pin-mux` execute the command below,

```
sudo beagle-pin-mux --pin hat-08 --mode pwm
```

Let's create a script called `fade.sh` that cycles through LED brightness on HAT pin8 by changing PWM duty cycle.

```
touch fade.sh
```

Now open the file with nano editor,

```
nano fade.sh
```

In the editor copy paste the script content below,

```
#!/bin/bash

PWMPIN="/sys/devices/platform/bus@f0000/23000000.pwm/pwm/pwmchip3/pwm1"


echo 1000 > $PWMPIN/period
echo 0 > $PWMPIN/duty_cycle
echo 0 > $PWMPIN/enable
sleep 1

for i in {1..500};
do
        echo $i > $PWMPIN/duty_cycle
        echo 1 > $PWMPIN/enable
        echo $i
        sleep 0.0005
done

for i in {500..1};
do
    echo $i > $PWMPIN/duty_cycle
    echo 1 > $PWMPIN/enable
    echo $i
    sleep 0.0005
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `fade.sh` script by typing:

```
bash fade.sh
```

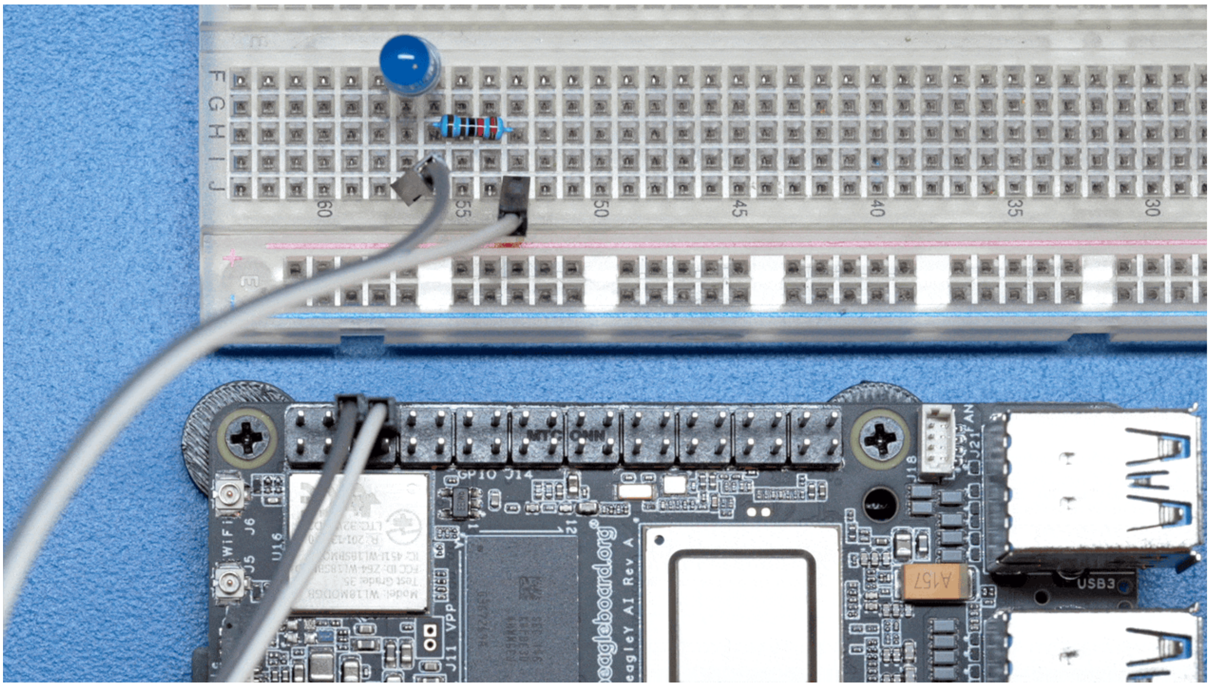


Fig. 5.10: LED PWM fade demo

- You can exit the `fade.sh` program by pressing `Ctrl + C` on your keyboard.

Todo: Add section about driving Servo Motors at 50KHz

5.2.3 Troubleshooting

Todo: Fill out empty section

5.2.4 Going Further

Todo: Fill out empty section

5.3 Using the on-board Real Time Clock (RTC)

Todo: Add specific actions rather than notes that this is a work-in-progress.

Real Time Clocks (RTCs) provide precise and reliable timekeeping capabilities, which are beneficial for applications ranging from simple timekeeping to complex scheduling and secure operations.

Without an RTC, a computer must rely on something called Network Time Protocol (NTP) to obtain the current time from a network source. There are many cases however where an SBC such as BeagleY-AI may not have a constant or reliable network connection. In situations such as these, an RTC allows the board to keep time even if the network connection is severed or the board loses power for an extended period of time.

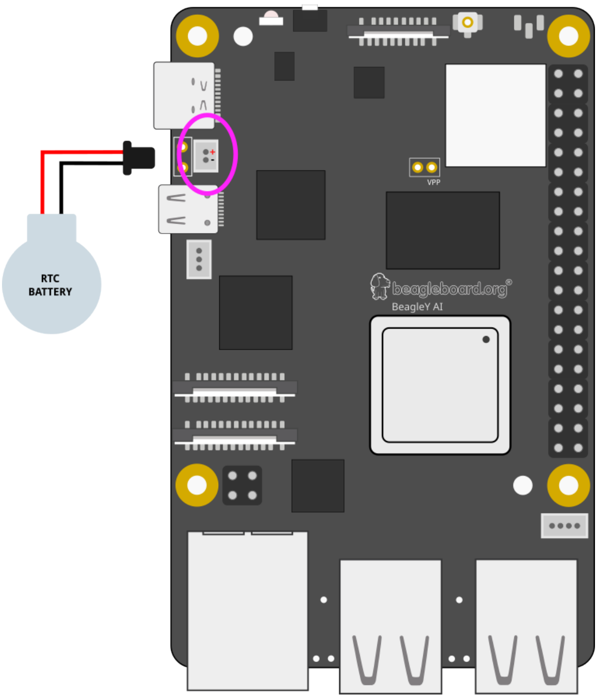
Fortunately, BeagleY-AI comes with a built-in DS1340 RTC for all your fancy time keeping needs!

5.3.1 Required Hardware

BeagleY provides a **1.00 mm pitch, 2-pin JST SH connector** for a coin cell battery to enable the RTC to keep time even if power is lost to the board.

These batteries are available from several vendors:

- Raspberry Pi 5 RTC Battery via Adafruit
- Raspberry Pi 5 RTC Battery via DigiKey
- CR2023 battery holder for Pi 5 via Amazon



5.3.2 Uses for an RTC

1. **Maintaining Accurate Time:** RTCs provide an accurate clock that continues to run even when the SBC is powered down. This is crucial for maintaining the correct time and date across reboots.
2. **Timestamping:** Many applications need to know the current time for timestamping data, logs, or events. For example, IoT devices may need to log sensor data with precise timestamps.
3. **Scheduling Tasks:** In some applications, tasks need to be scheduled at specific times. An RTC allows the SBC to keep track of time accurately, ensuring that tasks are performed at the correct times.
4. **Network Synchronization:** If the SBC is part of a larger network, having an accurate time helps with synchronizing data and events across the network.
5. **Standby Power Efficiency:** Many RTCs operate with a very low power requirement and can keep time even when the rest of the board is in a low-power or sleep mode. This helps in reducing overall power consumption.

5.3.3 Reading time

Note: You must set the time before being able to read it. If you don't do this first, you'll see errors. You may connect your BeagleY-AI to a network so it can get time from an NTP server.

Reading the current time on the RTC is achieved using the **hwclock** command.

```
debian@BeagleY:~$ sudo hwclock
2024-05-10 00:00:02.224187-05:00
```

5.3.4 Setting time

You can set time manually by running the following command:

```
hwclock --set --date "10/05/2024 21:01:05"
```

5.3.5 Diving Deeper

There are actually two different "times" that your Linux system keeps track of.

- System time, which can be read using the **date** or **timedatectl** commands
- RTC (hardware) time which can be read using the **hwclock** command shown above.

Comparing the time, we see something interesting, they're different!

You can just type "date" but the format will be different, so we add some extra instructions to match the format.

```
debian@BeagleBone:~$ date +%Y-%m-%d' '%H:%M:%S.%N%:z
2024-05-10 21:06:50.058595373+00:00

debian@BeagleBone:~$ sudo hwclock
2024-05-10 21:06:56.692874+00:00
```

But why? We see here that our system and hardware clock are over 9 seconds apart!

Ok, in this particular case we set the HW clock slightly ahead to illustrate the point, but in real life "drift" is a real problem that has to be dealt with. Environmental conditions like temperature or stray cosmic rays can cause electronics to become ever so slightly out of sync, and these effects only grow over time unless corrected. It's why RTCs and other fancier time keeping instruments implement various methods to help account for this such as temperature compensated oscillators.

Let's fix our hardware clock. We assume here that the system clock is freshly synced over NTP so it's going to be our true time "source".

```
debian@BeagleBone:~$ sudo hwclock --systohc
```

Let's write a simple script to get the two times, we'll call it **getTime.sh**:

```
HWTIME=$(sudo hwclock)
echo "RTC - ${HWTIME} "

SYSTIME=$(date +%Y-%m-%d' '%H:%M:%S.%N%:z)
echo "SYS - ${SYSTIME} "
```

Now let's run it!

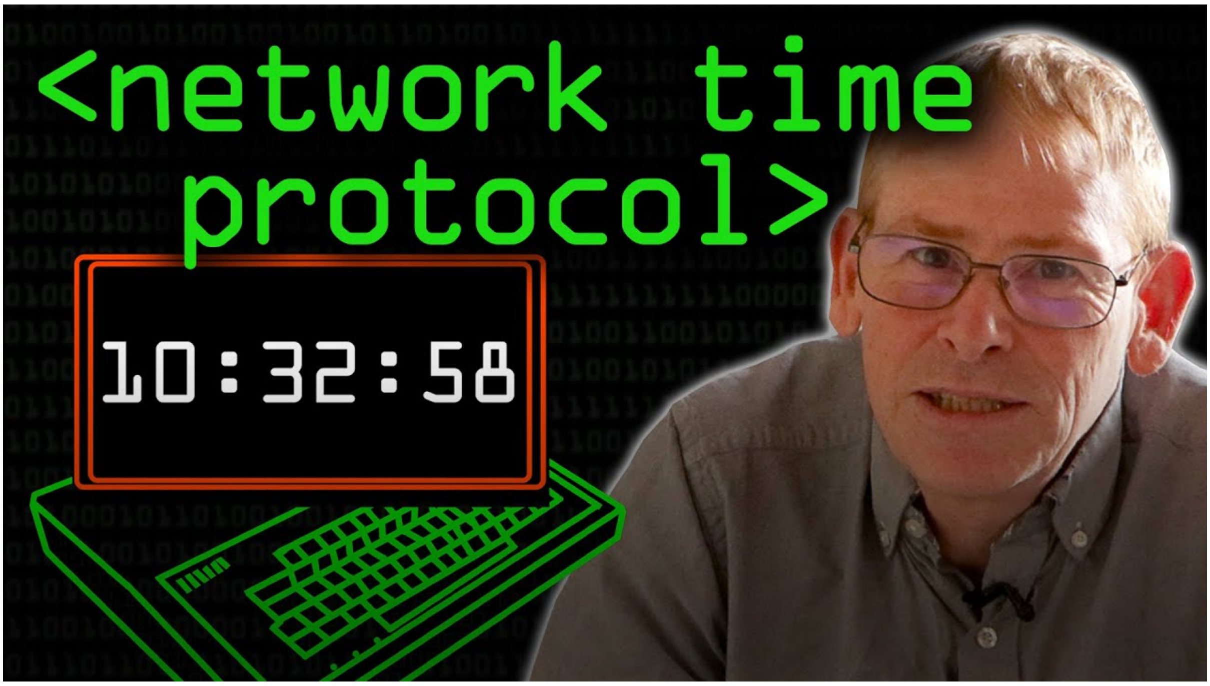


Fig. 5.11: https://youtu.be/BAo5C2qbLq8

```
debian@BeagleBone:~$ sudo chmod +x getTime.sh
debian@BeagleBone:~$ ./getTime.sh

RTC - 2024-05-10 21:52:58.374954+00:00
SYS - 2024-05-10 21:52:59.048442940+00:00
```

As we can see, we're still about a second off, but this is because it takes a bit of time to query the RTC via I2C.

If you want to learn more, the **Going Further** at the end of this article is a good starting point!

5.3.6 Troubleshooting

The most common error results from not having initialized the RTC at all. This usually happens if the system is powered on without an RTC battery and without a network connection.

In such cases, you should be able to read the time after setting the time as follows:

```
debian@BeagleBone:~$ sudo hwclock --systohc

debian@BeagleBone:~$ sudo hwclock
2024-05-10 21:06:56.692874+00:00
```

5.3.7 Going Further

Consider learning about topics such as time keeping over GPS and Atomic Clocks!

There are some good YouTube videos below to provide sources for inspiration.

Network Time Protocol - Computerphile

Nanosecond Clock Sync - Jeff Geerling

Fig. 5.12: https://youtu.be/RvnG-ywF6_s

Using GPS with PPS to synchronize clocks over the network

Work in progress

Todo: Add further testing steps, results, and images.

5.4 Using PCA9685 Motor Drivers

There are several such "Motor and Servo Driver HATs" available on Amazon, Adafruit and other marketplaces. While different manufacturers implement them slightly differently, the operating principle remains the same.

This guide aims to show you examples for two, namely the Xicoolee and Adafruit variants and how you can modify the example Python userspace library for other variations.

5.4.1 Operating Principle

The NXP PCA9685 is a simple 16-channel, 12-bit PWM controller that communicates over I2C.

While originally designed as an LED driver, it's ability to output PWM also makes it suitable as a Servo Motor driver.

In addition, to add the ability to drive DC motors, some board designers add one or two Toshiba TB6612FNG dual motor drivers as shown in the schematic below.

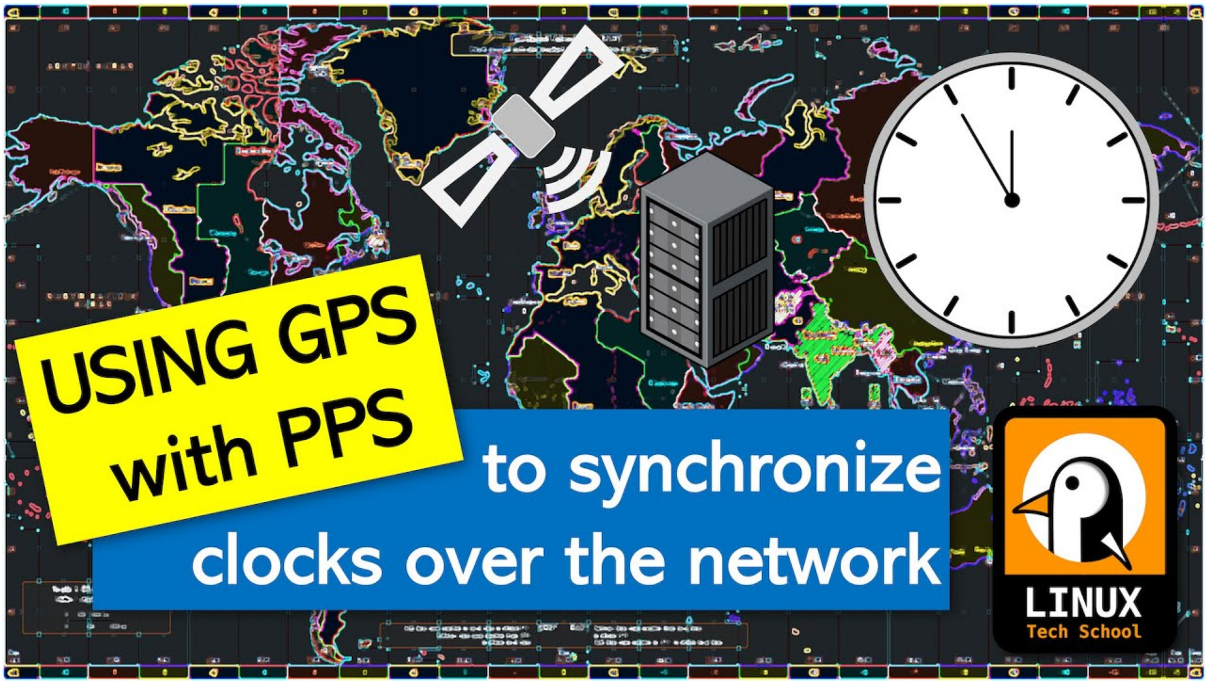
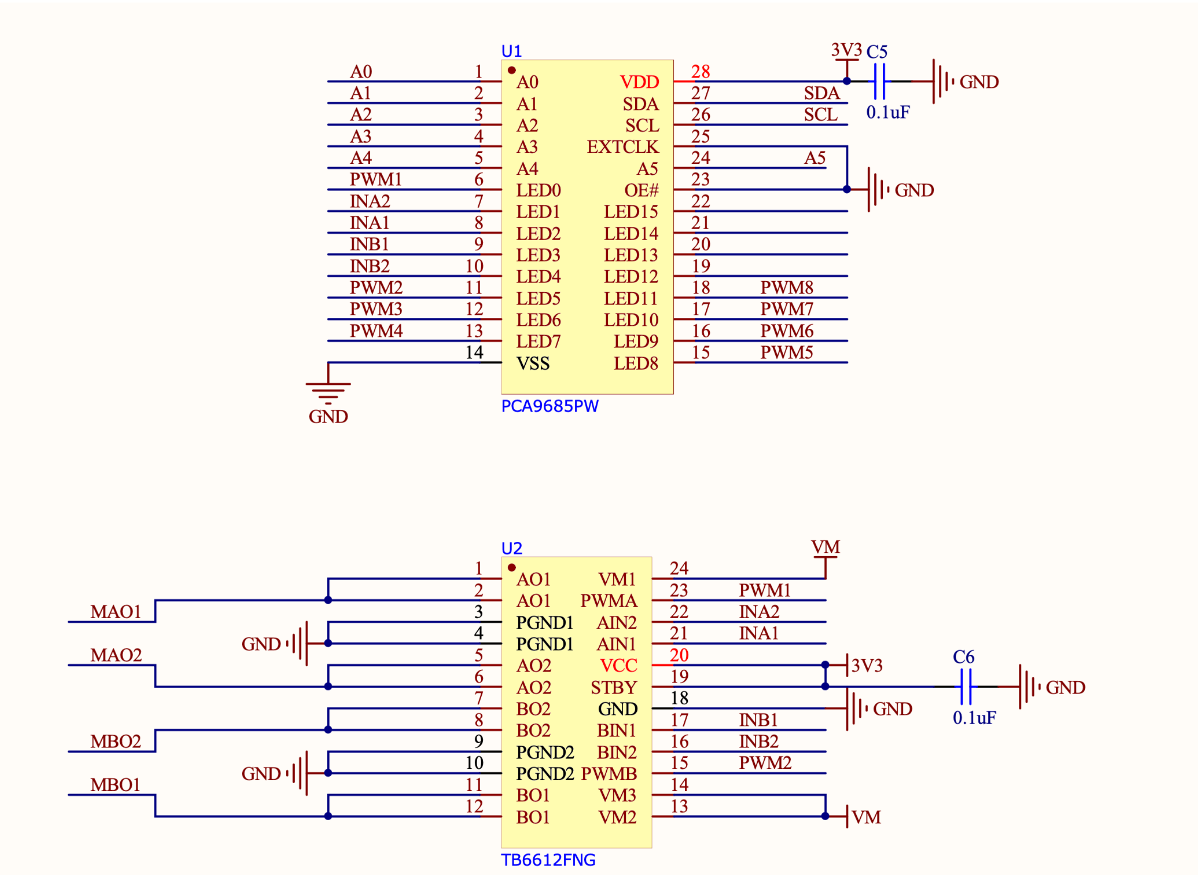


Fig. 5.13: https://youtu.be/7aTZ66ZL6Dk



If we look at the Xicoolee board and compare it to the schematic, we see that indeed Servo Channels 3-8 on the PCB Silkscreen match pins 12 through 18 of the PWM Driver, while PWM1, PWM2, INA1/2 and INB1/2 are used in conjunction with the TB6612FNG.

Looking at the TB6612FNG Datasheet, we can see that the IN pins for Channels A and B (INAx, INBx) are used to control the direction or "mode" of the DC motor, while the PWM signal controls the rotation speed for that particular channel.

TOSHIBA TB6612FNG

H-SW Control Function

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Thus, we can use the decoder table above to infer that to drive motor channel A at 50% speed clockwise, we would set the PCA9685 to output INA1 High, INA2 Low and PWM1 at a 50% duty cycle.

If we wanted to go counter-clockwise, we would simply swap things around so INA1 was Low, INA2 was High and assuming we want to keep the same rotation speed, PWM1 at a 50% duty cycle.

Lastly, we have the option for a "Short Brake" for the motors but please note that it is not recommended to keep motors in this state as that shorts the coils internally and will cause them to heat up over time. If you want to stop your motor, you should issue a "Short brake" state followed by a short delay to allow the motor to physically stop rotating and then leave the motor in the "Stop" state (which de-energizes the coils) by setting IN1 and IN2 to LOW.

But enough theory, let's use some actual code to make things spin…

5.4.2 Using Adafruit ServoKit

If you are looking to drive Servo motors accurately and not particularly interested in driving DC motors, you may consider using the Adafruit ServoKit library which simplifies this type of use case. As with all python modules, make sure you do so inside a virtual environment as shown below!

```
mkdir project-name && cd project-name
python3 -m venv .venv
source .venv/bin/activate
sudo pip3 install --upgrade setuptools
sudo pip install --upgrade adafruit-python-shell
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-
↪Scripts/master/raspi-blinka.py
sudo python raspi-blinka.py
pip3 install adafruit-circuitpython-servokit adafruit-circuitpython-
↪busdevice adafruit-circuitpython-register
```

From here, you should be able to run some example code such as the following:

```
import time
from adafruit_servokit import ServoKit
```

(continues on next page)

(continued from previous page)

```
# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=16)

kit.servo[0].angle = 180
kit.continuous_servo[1].throttle = 1
time.sleep(1)
kit.continuous_servo[1].throttle = -1
time.sleep(1)
kit.servo[0].angle = 0
kit.continuous_servo[1].throttle = 0
```

To explore ServoKit further, check out the ServoKit Github Page and Examples

5.4.3 Python User-space Driver

As mentioned before, the PCA9685 is a rather simple I2C device, so the driver for it is equally simple: PCA9685.py

Simply download this to the root of your project and you are most of the way there.

From there, you simply need an import statement and to define the driver instance:

```
from PCA9685 import PCA9685

pwm = PCA9685(0x60, debug=False) #Default I2C Address for the shield is 0x60
pwm.setPWMFreq(50) #Most Servo Motors use a PWM Frequency of 50Hz
```

You can now drive LEDs or servo motors by issuing the following command (replacing pin and dutyCycle with your particular values):

```
pwm.setDutycycle(pin, dutyCycle)
```

5.4.4 WaveShare Motor and Servo Driver HAT

Waveshare writes some of the better documentation for these types of Motor Driver HATs

Todo: Add more information on Waveshare motor & servo driver HAT.

5.4.5 XICOOLEE Motor and Servo Driver HAT

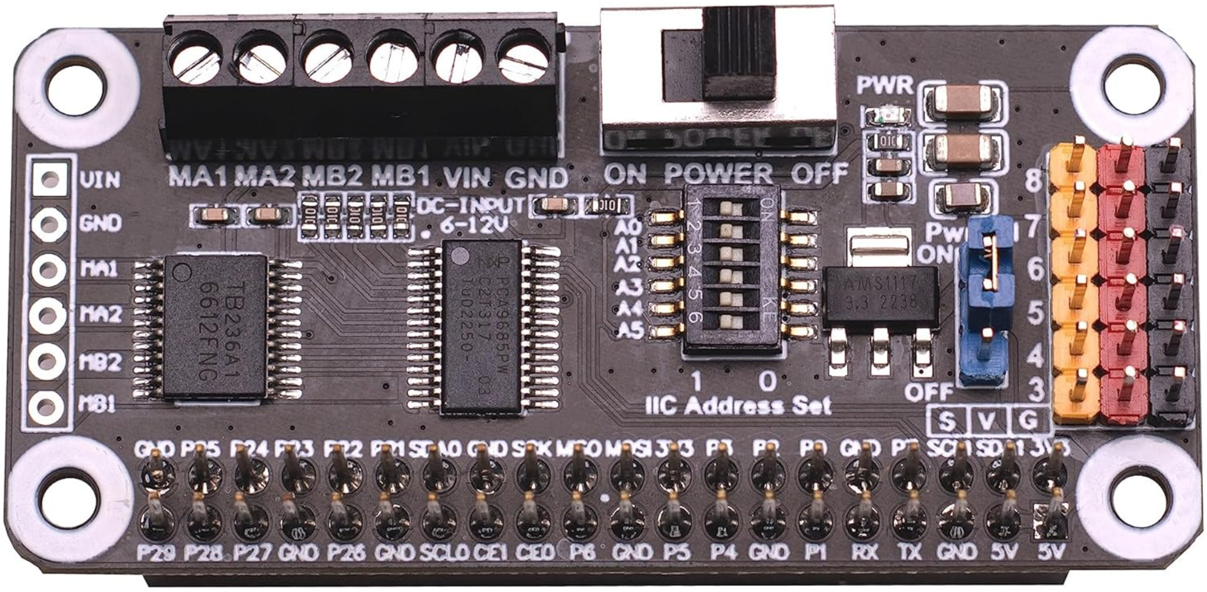


Photo Credit - Xicoolee

Looking at the schematic for the Xicoolee HAT, we see that we need to define our DC motor pins as follows:

```
#Xicoolee TB6612FNG

self.PWMA = 0
self.AIN1 = 2
self.AIN2 = 1
self.PWMB = 5
self.BIN1 = 3
self.BIN2 = 4
```

We can then run some simple example code as shown below:

```
#!/usr/bin/python

from PCA9685 import PCA9685
import time

Dir = [
    'forward',
    'backward',
]
pwm = PCA9685(0x40, debug=False)
pwm.setPWMFreq(50)

class MotorDriver():
    def __init__(self):
        # Match these to your particular HAT!
        self.PWMA = 0
        self.AIN1 = 2
        self.AIN2 = 1
        self.PWMB = 5
        self.BIN1 = 3
        self.BIN2 = 4

    def MotorRun(self, motor, index, speed):
        if speed > 100:
            return
```

(continues on next page)

(continued from previous page)

```
        if(motor == 0):
            pwm.setDutycycle(self.PWMA, speed)
            if(index == Dir[0]):
                print ("1")
                pwm.setLevel(self.AIN1, 0)
                pwm.setLevel(self.AIN2, 1)
            else:
                print ("2")
                pwm.setLevel(self.AIN1, 1)
                pwm.setLevel(self.AIN2, 0)
        else:
            pwm.setDutycycle(self.PWMB, speed)
            if(index == Dir[0]):
                print ("3")
                pwm.setLevel(self.BIN1, 0)
                pwm.setLevel(self.BIN2, 1)
            else:
                print ("4")
                pwm.setLevel(self.BIN1, 1)
                pwm.setLevel(self.BIN2, 0)

    def MotorStop(self, motor):
        if (motor == 0):
            pwm.setDutycycle(self.PWMA, 0)
        else:
            pwm.setDutycycle(self.PWMB, 0)

print("this is a motor driver test code")
Motor = MotorDriver()

print("forward 2 s")
Motor.MotorRun(0, 'forward', 100)
Motor.MotorRun(1, 'forward', 100)
time.sleep(2)

print("backward 2 s")
Motor.MotorRun(0, 'backward', 100)
Motor.MotorRun(1, 'backward', 100)
time.sleep(2)

print("stop")
Motor.MotorStop(0)
Motor.MotorStop(1)
```

5.4.6 Adafruit DC & Stepper Motor HAT

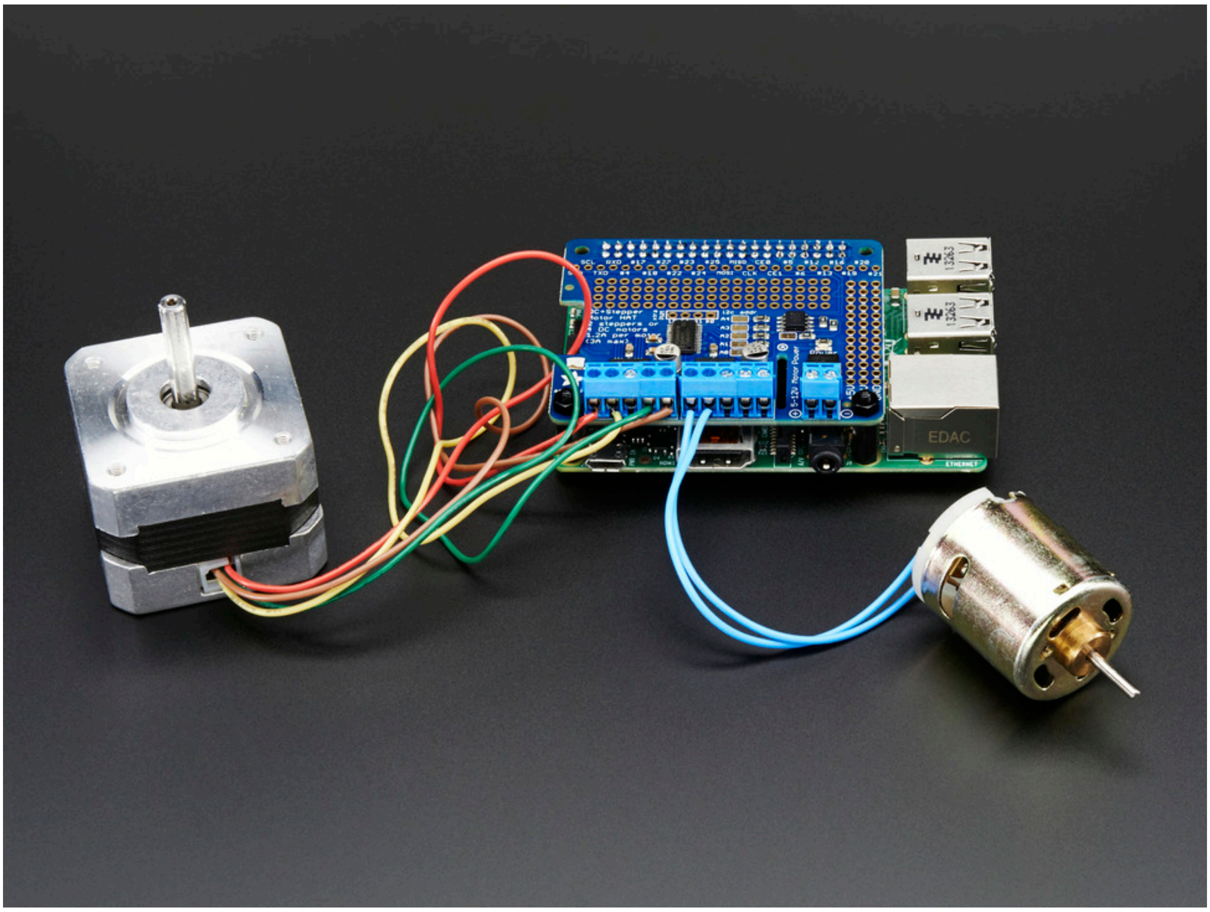


Photo Credit - Adafruit

Looking at the schematic for the Adafruit HAT, we see that we need to define our DC motor pins as follows:

```
#Adafruit TB6612FNG #1

self.PWMA = 8
self.AIN1 = 10
self.AIN2 = 9
self.PWMB = 13
self.BIN1 = 11
self.BIN2 = 12

#Adafruit TB6612FNG #2

self.PWMA_2 = 2
self.AIN1_2 = 4
self.AIN2_2 = 3
self.PWMB_2 = 7
self.BIN1_2 = 5
self.BIN2_2 = 6
```

Todo: Expand on running 2 DC motor objects

5.5 Booting from NVMe Drives

Work in progress

Todo: Add further testing steps, results, and images.

BeagleY-AI supports a PCI-Express x1 interface which enables data rates of up to 1GB/s for high speed expansion.

Note: While the SoC supports PCI-e Gen 3, the flat-flex connector required by HATs is only rated for PCI-e Gen 2, so, as is the case with other similar boards in this form factor, actual transfer speeds may be limited to Gen 2, depending on a variety of layout and environmental factors

This enables it to take advantage of standard PC NVMe drives which offer exponentially higher random and sequential read/write speeds as well as improved endurance over SD cards or traditional eMMC storage.

While the boot-ROM on the AM67 SoC does not support direct boot-to-NVMe, we can use a method where we boot U-Boot from the SD Card and then use it to load the Linux filesystem from external NVMe storage.

5.5.1 Verified HATs and Drives

Most/All HATs and NVMe drives should work, but the following have been verified to work as part of writing this guide:

HATs:

1. Geekworm X1001 PCIe to M.2 Key-M
2. Geekworm X1000 PCIe M.2 Key-M

NVMe drives:

1. Kingston OM3PDP3512B (512GB 2230)
2. Kingston NV2 (512GB 2280)

Drive Adapters (3D Printable):

The X1000 above uses the slightly uncommon 2242 drive size, so, an adapter may be required to mount a 2230 drive.

1. A simple adapter from `@eliasjonsson` on Printables works great - https://www.printables.com/model/578236-m2-ssd-2230-to-2242
2. Similar adapters exist for 2230 to 2280 for example such as this one from `@nzalog` - https://www.printables.com/model/217264-2230-to-2280-m2-adapter-ssd

5.5.2 Step by step

Note: This article was written using the BeagleY-AI Debian XFCE 12.5 2024-03-25 image.

Step 1. Boot from SD Normally

Grab the latest BeagleY-AI SD Image from (BeagleBoard.org/distros.)

Once logged in and at the terminal, make sure your system is up to date (a reboot is also recommended after updating)

```
sudo apt-get update && sudo apt-get full-upgrade -y
sudo reboot
```

Step 2. Verify that your NVMe drive is detected

The command `lspci` will list the attached PCI Express devices on the system:

```
debian@BeagleY:~$ lspci
```

You should see an output similar to the following, where the first entrance is the SoC internal PCI Express bridge device and the second device listed is your NVMe drive, in this case, a Kingston OM3PDP3 drive.

```
00:00.0 PCI bridge: Texas Instruments Device b010
01:00.0 Non-Volatile memory controller: Kingston Technology Company, Inc.␣
↪OM3PDP3 NVMe SSD (rev 01)
```

Now that we know the PCIe device is detected, let's see if it's recognized as a Storage Device:

The command `lsblk` will list the attached storage devices on the system:

```
debian@BeagleY:~$ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mmcblk1      179:0    0 29.7G  0 disk
├─mmcblk1p1 179:1    0  256M  0 part /boot/firmware
├─mmcblk1p2 179:2    0    4G  0 part [SWAP]
└─mmcblk1p3 179:3    0 25.5G  0 part /
nvme0n1      259:0    0 476.9G  0 disk
└─nvme0n1p1 259:1    0 476.9G  0 part
```

Here we see that two devices are connected, `mmcblk1` corresponds to our SD card, and `nvme0n1` corresponds to our NVMe drive, so everything is ready to go!

If your drives aren't listed as expected, please check the Troubleshooting section at the end of this document.

Step 3. Copy your filesystem and modify extlinux.conf for NVMe boot

A variety of useful scripts are available in `/opt/`, one of them enables us to move our micro-sd contents to NVMe and make BeagleY-AI boot from there directly.

The following 3 commands will change your U-boot prompt to boot from NVMe by default, but the serial boot menu will still enable you to fall back to SD boot or other modes if something happens.

Note: This will copy the entire contents of your SD card to the NVMe drive, so expect it to take upwards of 15 minutes. This only needs to be run one time

```
sudo cp -v /opt/u-boot/bb-u-boot-beagley-ai/beagley-ai-microsd-to-nvme-w-
↪swap /etc/default/beagle-flasher
sudo beagle-flasher-mv-rootfs-to-nvme
sudo reboot
```

Enjoy NVMe speeds!

Now that we've run the scripts above, you should see that lsblk now reports that our / or root filesystem is on the `nvme0n1p1` partition, meaning we are successfully booting from the NVMe drive.

It's subtle, but the change can be seen by running `lsblk` again.

```
debian@BeagleY:~$ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mmcblk1      179:0    0 29.7G  0 disk
├─mmcblk1p1 179:1    0  256M  0 part /boot/firmware
├─mmcblk1p2 179:2    0    4G  0 part
└─mmcblk1p3 179:3    0 25.5G  0 part
nvme0n1      259:0    0 476.9G  0 disk
└─nvme0n1p1 259:1    0 476.9G  0 part /
```

Congratulations!

5.5.3 Troubleshooting

While most setups should work, it is possible that a combination of Software, Hardware or both can result in minor issues. Here are some ideas for troubleshooting on your own:

Check that your cables are plugged in and oriented correctly

The flat-flex ribbon cable will only connect correctly one way, so ensure the orientation is correct with your expansion HAT manual and that the ribbon cable is correctly seated.

A note on power-hungry drives

While most drives can be powered as-is with only the ribbon cable, some drives, especially high end full-size 2280 drives may consume more power than normal for an M.2 connector. For such cases, some HAT expansions will provide a means of providing external supplemental power. If your drive is not detected, it may be worthwhile to try using a drive from a different manufacturer as a troubleshooting step.

As a side note, since 2230 drives are normally designed to run in Laptops, they tend to also consume less power than their desktop counterparts and as such, are a "safer" option.

Check the Linux Kernel Logs for PCI:

You should see something similar to below without further errors:

```
debian@BeagleY:~$ dmesg | grep "PCI"
[    0.005276] PCI/MSI: /bus@f0000/interrupt-controller@1800000/msi-
↪controller@1820000 domain created
[    0.158546] PCI: CLS 0 bytes, default 64
[    3.674209] j721e-pcie-host f102000.pcie: PCI host bridge to bus 0000:00
[    3.742406] pci 0000:01:00.0: 7.876 Gb/s available PCIe bandwidth,␣
↪limited by 8.0 GT/s PCIe x1 link at 0000:00:00.0 (capable of 31.504 Gb/s␣
↪with 8.0 GT/s PCIe x4 link)
[    4.915630] pci 0000:00:00.0: PCI bridge to [bus 01]
```

Still having issues?

Post questions on the forum under the tag "beagley-ai".

5.6 Using IMX219 CSI Cameras

Work in progress

Todo: Add further testing steps, results, and images.

To enable an IMX219 CSI camera, modify the following file: */boot/firmware/extlinux/extlinux.conf*

We can check the available list of Device Tree Overlays as such:

```
debian@BeagleBone:~$ ls /boot/firmware/overlays/ | grep "beagley"
k3-am67a-beagley-ai-csi0-imx219.dtbo
k3-am67a-beagley-ai-csi0-ov5640.dtbo
k3-am67a-beagley-ai-csi1-imx219.dtbo
k3-am67a-beagley-ai-dsi-rpi-7inch-panel.dtbo
k3-am67a-beagley-ai-lincolntech-185lcd-panel.dtbo
```

5.6.1 Using CSI Port 0

Then, add the following line to load the IMX219 CSI0 DTBO:

```
fdtoverlays /overlays/k3-am67a-beagley-ai-csi0-imx219.dtbo
```

Your /boot/firmware/extlinux/extlinux.conf file should look something like this:

```
label microSD (default)
   kernel /Image
   append console=ttyS2,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4␣
↪rootwait net.ifnames=0
   fdtdir /
   fdt /ti/k3-j722s-beagley-ai.dtb
   fdtoverlays /overlays/k3-am67a-beagley-ai-csi0-imx219.dtbo
   initrd /initrd.img
```

Now reboot...

```
debian@BeagleBone:~$ ls /dev/ | grep "video"
video0
video1
video2
```

5.6.2 Using CSI Port 1

5.6.3 Troubleshooting

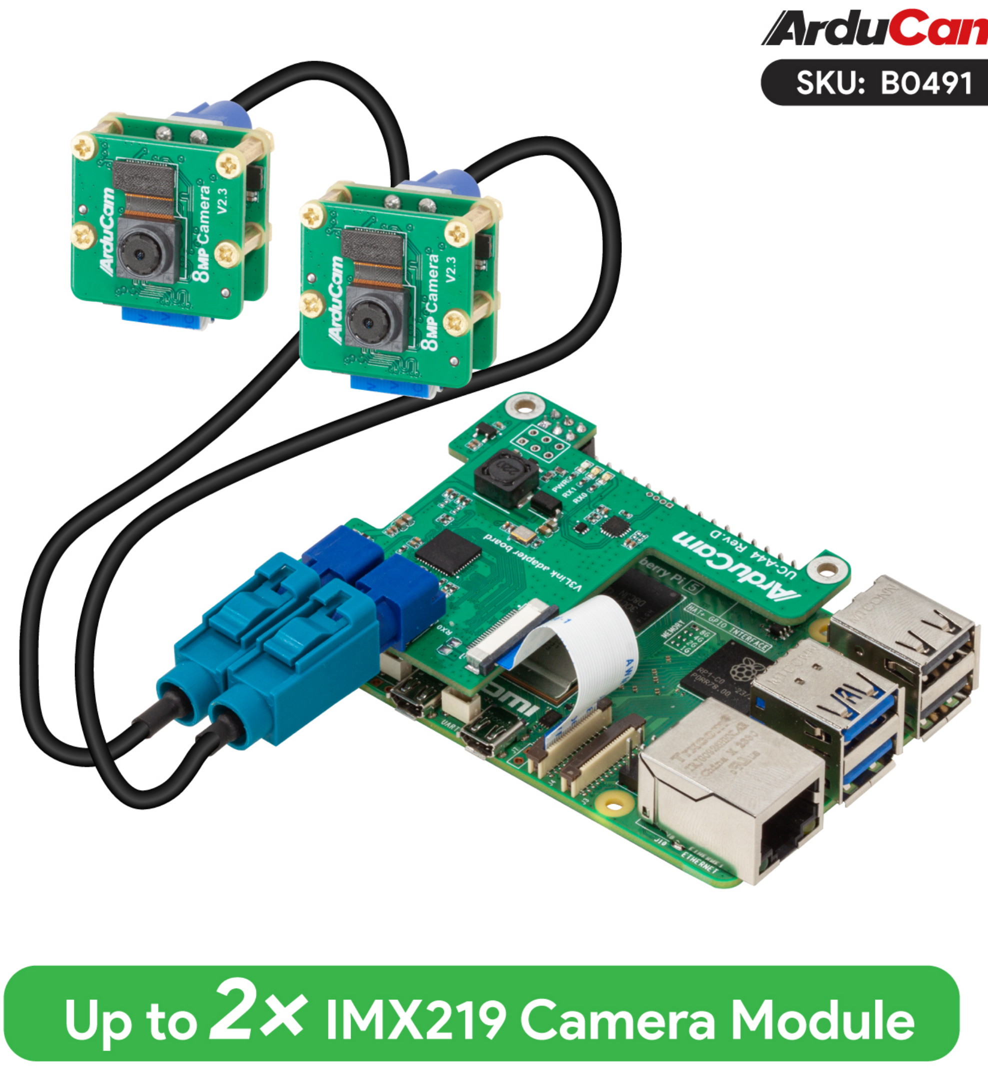
```
Found /extlinux/extlinux.conf
Retrieving file: /extlinux/extlinux.conf
beagley-ai microSD (extlinux.conf)
   1:      microSD Recovery
   2:      microSD (RPI 7inch panel)
   3:      microSD (lincolntech-185lcd panel)
   4:      microSD (csi0 imx219)
   5:      microSD (csi1 imx219)
   6:      microSD (csi0 ov5640)
   7:      microSD (default)
Enter choice: 4
   4:      microSD (csi0 imx219)
```

5.7 Using the Arducam Dual V3Link Camera Kit

Work in progress

Todo: Add further testing steps, results, and images.

The Arducam Dual V3Link Camera Kit is an IMX219 based kit that leverages Texas Instruments' FPDLink technology to enable using two CSI cameras over a single port up to 15 meters away using twisted pair cables.



Note: Unlike the larger quad-camera kit, the dual camera kit aims to simplify the software stack and improve interoperability with the Raspberry Pi and other non-TI SBCs by forgoing the ability to support multi-stream CSI inputs. This means that it is limited to "switching" between the two FPDLink inputs but has the benefit of not requiring additional drivers beyond support for the base CSI camera driver (IMX219 in this case)

5.7.1 Initial Hardware Connection

Simply plug in the HAT into the BeagleY GPIO header and connect the CSI header as shown below.

Either CSI header may be connected but make sure you use the corresponding CSI port DTS when enabling your "camera".

Todo: ADD CSI 0/1 Header Location photo.

5.7.2 Verify that the HAT is connected

The Arducam HAT should present itself as an I2C device on Bus 1.

To check that the I2C Bus looks like we expect:

```
sudo i2cdetect -r -y 1
```

To verify actual communication with the FPDlink device, we issue the following command:

```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x01 r1
```

5.7.3 Switching CSI Channels

The channel numbering for FPDLink goes from 1 to 2 (as opposed to counting from 0 as is the case for CSI)

Thus, to select video output from channel 1:

```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x01
```

To switch to channel 2:

```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x02
```

5.7.4 Troubleshooting

For additional documentation and support, see the Arducam Docs.

Chapter 6

Support

All support for BeagleY-AI design is through BeagleBoard.org community at BeagleBoard.org forum.

6.1 Production board boot media

Todo: Add production boot media link in `_static/epilog/production.image` and reference it here.

6.2 Certifications and export control

6.2.1 Export designations

- HS: 8471504090
- US HS: 8543708800
- UPC: 640265311062
- EU HS: 8471707000
- COO: CHINA

6.2.2 Size and weight

- Bare board dimensions: 85 x 56 x 20 mm
- Bare board weight: 50 g
- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 110g

6.3 Additional documentation

6.3.1 Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleY-AI design repository.

6.3.2 Software docs

For BeagleY-AI specific software projects you can checkout all the BeagleY-AI project repositories group.

6.3.3 Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tag/beagley-ai

6.3.4 Pictures

6.4 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

6.4.1 Board Changes

For all changes, see https://openbeagle.org/beagley-ai/beagley-ai. Versions released into production are noted below.

Table 6.1: BeagleY-AI board change history

Rev	Changes	Date	By