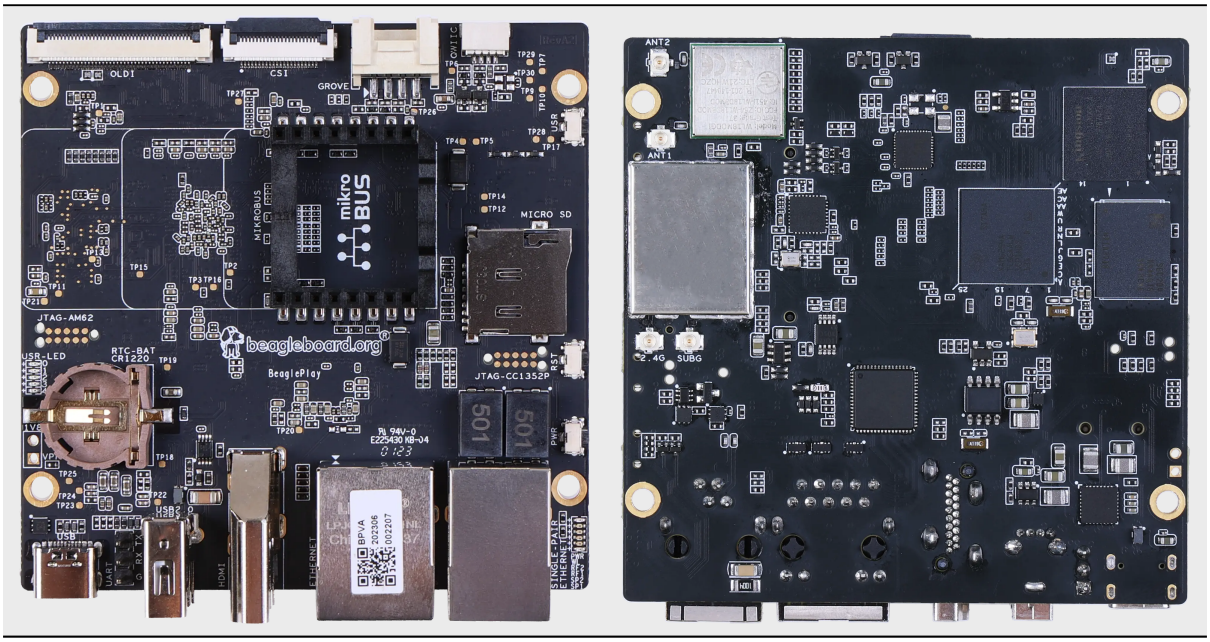
BeaglePlay

BeagleBoard.org Foundation
Oct 18, 2024

Table of contents

BeaglePlay is an open-source single board computer based on the Texas Instruments AM6254 quad-core Cortex-A53 Arm SoC designed to simplify the process of adding sensors, actuators, indicators, human interfaces, and connectivity to a reliable embedded system.

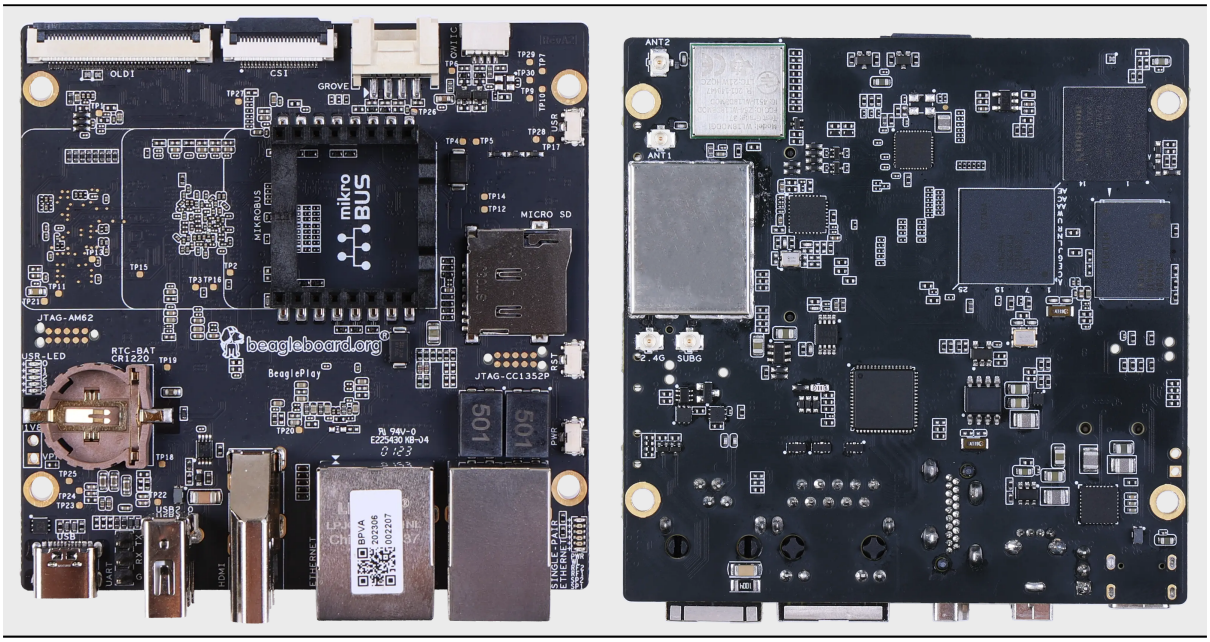


Chapter 1

Introduction

BeaglePlay is an open-source single board computer designed to simplify the process of adding sensors, actuators, indicators, human interfaces, and connectivity to a reliable embedded system. It features a powerful 64-bit, quad-core processor and innovative connectivity options, including WiFi, Gigabit Ethernet, sub-GHz wireless, and single-pair Ethernet with power-over-data-line. With compatibility with 1,000s of off-the-shelf add-ons and a customized Debian Linux image, BeaglePlay makes expansion and customization easy. It also includes ribbon-cable connections for cameras and touch-screen displays, and a socket for a battery-backed real-time-clock, making it ideal for human-machine interface designs. With its competitive price and user-friendly design, we expect BeaglePlay to provide you with a positive development experience. Some of the real world applications for BeaglePlay include:

- Building/industrial automation gateways
- Digital signage
- Human Machine Interface (HMI)
- BeagleConnect sensor gateways



1.1 Detailed overview

BeaglePlay is built around Texas Instruments AM62x Sitara™ Processors which is a Quad-Core Arm® Cortex®-A53 Human-machine-interaction SoC. It comes with 2GB DDR4 RAM, 16GB eMMC storage, Full size HDMI, USB-A host port, USB-C power & connectivity port, serial debug interface, and much more.

Table 1.1: BeaglePlay features

Feature	Description
Processor	TI AM6254 (multicore A53s with R5, M4s and PRUs)
PMIC	TPS6521901
Memory	2GB DDR4
Storage	16GB eMMC
WiFi	• PHY: WL1807MOD (roadmap to next-gen TI CC33XX WiFi 6 & BLE) • Antennas: 2.4GHz & 5GHz
BLE/SubG	• CC1352P7 M4+M0 with BeagleConnect firmware • BeagleConnect Wireless enabled • Antennas: 2.4GHz & SubG IEEE802.15.4 software defined radio (SDR)
Ethernet	• PHY: Realtek RTL8211F-VD-CG Gigabit Ethernet phy • Connector: integrated magnetics RJ-45
Single-pair Ethernet	• BeagleConnect Wired enabled • PHY: DP83TD510E 10Mbit 10BASE-T1L single-pair Ethernet phy • Connector: RJ-11 jack • Power (PoDL): Input: N/A (protection to 12V), Output: 5V @ 250mA
USB type-C	• PD/CC: None, HS shorted to both sides • Power: Input: 5V @ 3A, Output: N/A (USB-C DRP Not supported)
HDMI	• Transmitter: IT66121 • Connector: full-size
Other connectors	• microSD • USB 2.0 type-A (480Mbit) • mikroBUS connector (I2C/UART/SPI/MCAN/MCASP/PWM/GPIO) • Grove connector (I2C/UART/ADC/PWM/GPIO) • QWIIC connector (I2C) • CSI connector compatible with BeagleBone AI-64, Raspberry Pi Zero / CM4 (22-pin) • OLDI connector (40-pin)

1.1.1 AM6254 SoC

The low-cost Texas Instruments AM625 family of application processors are built for Linux® application development. With scalable Arm® Cortex®-A53 performance and embedded features, such as: dual-display support and 3D graphics acceleration, along with an extensive set of peripherals that make the AM62x device

well-suited for a broad range of industrial and automotive applications while offering intelligent features and optimized power architecture as well.

Some of the SoC applications include:

- Industrial HMI
- EV charging stations
- Touchless building access
- Driver monitoring systems

AM625 processors are industrial-grade in the 13 x 13 mm package (ALW) and can meet the AEC-Q100 automotive standard in the 17.2 x 17.2 mm package (AMC). Industrial and Automotive functional safety requirements can be addressed using the integrated Cortex-M4F core and dedicated peripherals, which can all be isolated from the rest of the AM62x processor.

Tip: For more details checkout https://www.ti.com/product/AM625

The 3-port Gigabit Ethernet switch has one internal port and two external ports with Time-Sensitive Networking (TSN) support. An additional PRU module on the device enables real-time I/O capability for customer's own use cases. In addition, the extensive set of peripherals included in AM62x enables system-level connectivity, such as: USB, MMC/SD, CSI Camera interface, OSPI, CAN-FD and GPMC for parallel host interface to an external ASIC/FPGA. The AM62x device also employs advanced power management support for portable and power-sensitive applications.

1.1.2 Board components location

Front

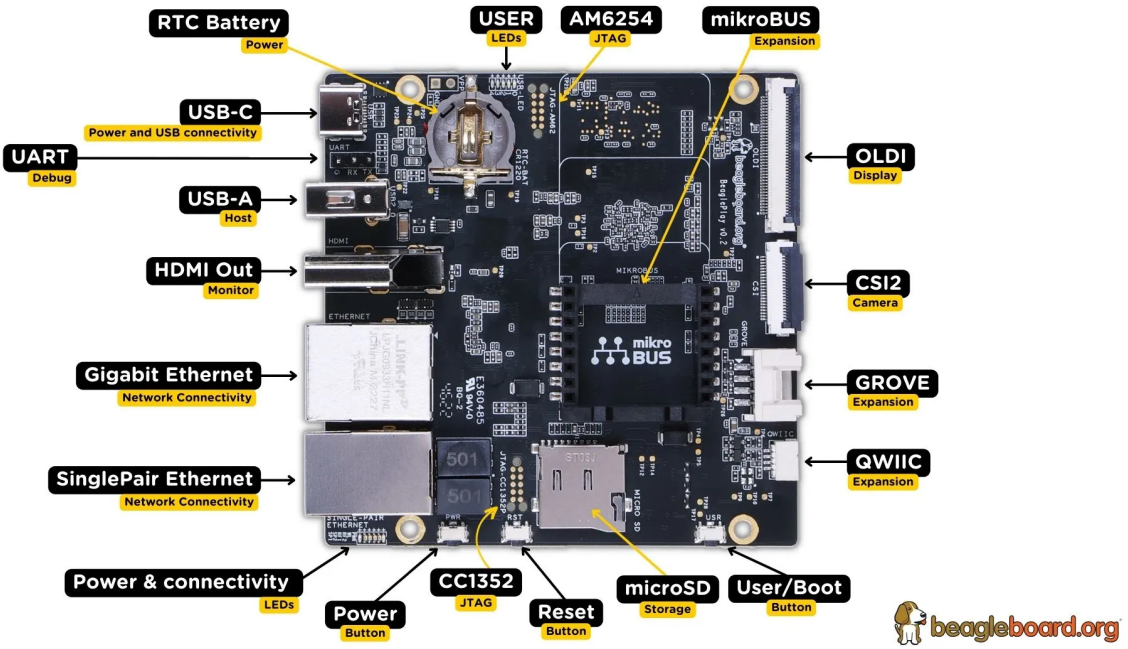


Fig. 1.1: BeaglePlay board front components location

Table 1.2: BeaglePlay board front components location

Feature	Description
RTC Battery	BQ32002 Real Time Clock (RTC) Battery holder takes CR1220 3V battery
User LEDs	Five user LEDs, board-power-and-boot section provides more details. These LEDs are connect to the AM6254 SoC
JTAG (AM62)	AM6254 SoC JTAG debug port
mikroBUS	mikroBUS for MikroE Click boards or any compliant add-on
OLDI	AM6254 OpenLDI(OLDI) display port
CSI	AM6254 Camera Serial Interface (MIPI CSI-2)
Grove	SeeedStudio Grove modules connection port
QWIIC	SparkFun QWIIC / Adafruit STEMMA-QT port for I2C modules connectivity
User Button	Programmable user button, also servers as boot mode slect button (SD Card/eMMC). Press down to select SD Card as boot medium
SD Card	Use to expand storage, boot linux image or flash latest image on eMMC
Reset button	Press to reset BeaglePlay board (AM6254 SoC)
JTAG (CC1352)	JTAG debug port for CC1352P7
Power button	Press to shut-down (OFF), hold down to boot (ON)
Power & Connectivity LEDs	Indicator LEDs for Power ON, CC1352 RF, and Single-pair connectivity
Single-pair Ethernet	Single-pair Ethernet connectivity port with power over data line
GigaBit Ethernet	1Gb/s Wired internet connectivity
HDMI Output	Full size HDMI port for connecting to external display monitors
USB-A host port	Port to connect USB devices like cameras, keyboard & mouse combos, etc
USB-C port	Power and Device data role port

Back

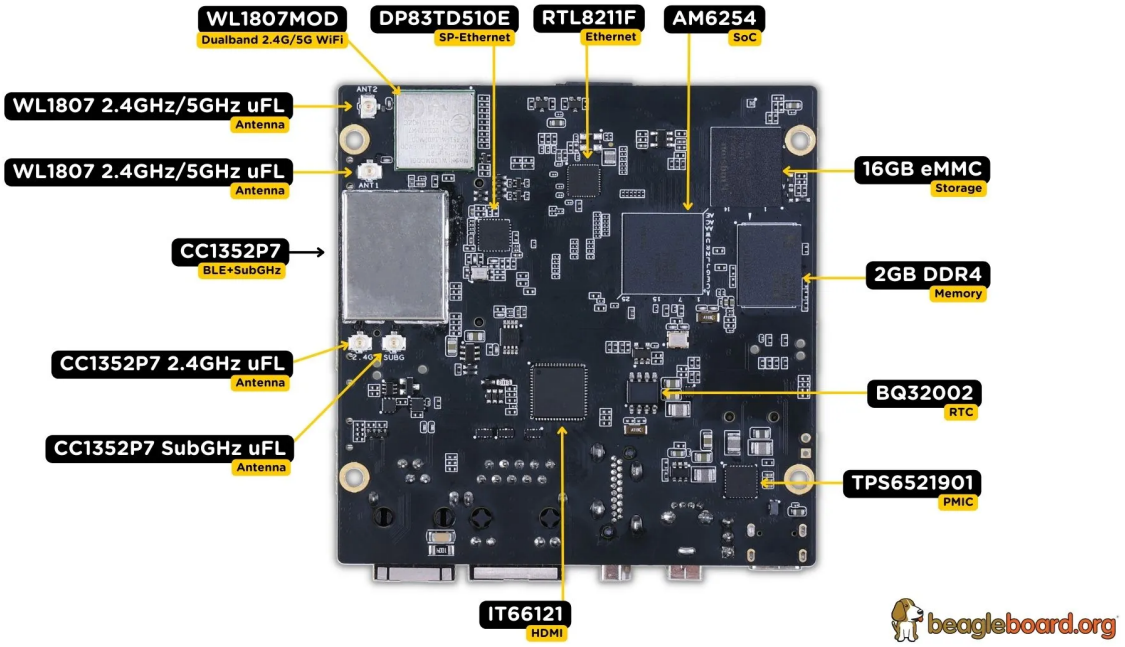


Fig. 1.2: BeaglePlay board back components location

Table 1.3: BeaglePlay board back components location

Feature	Description
CC1352P7	2.4GHz BLE + SubG IEEE 802.15.4 with 1 x 2.4GHz + 1 x SubG uFL antenna
WL1807MOD	Dual band (2.4GHz & 5GHz) WiFi module with 2 x uFL antennas
DP83TD510E	Single-pair IEEE 802.3cg 10BASE-T1L Ethernet PHY
RTL8211F	Gigabit IEEE 802.11 Ethernet PHY
AM6254	Main SoC
16GB eMMC	Flash storage
2GB DDR4	RAM / Memory
BQ32002	Real Time Clock (RTC)
TPS6521901	Power Management IC
IT66121	HDMI Transmitter

Chapter 2

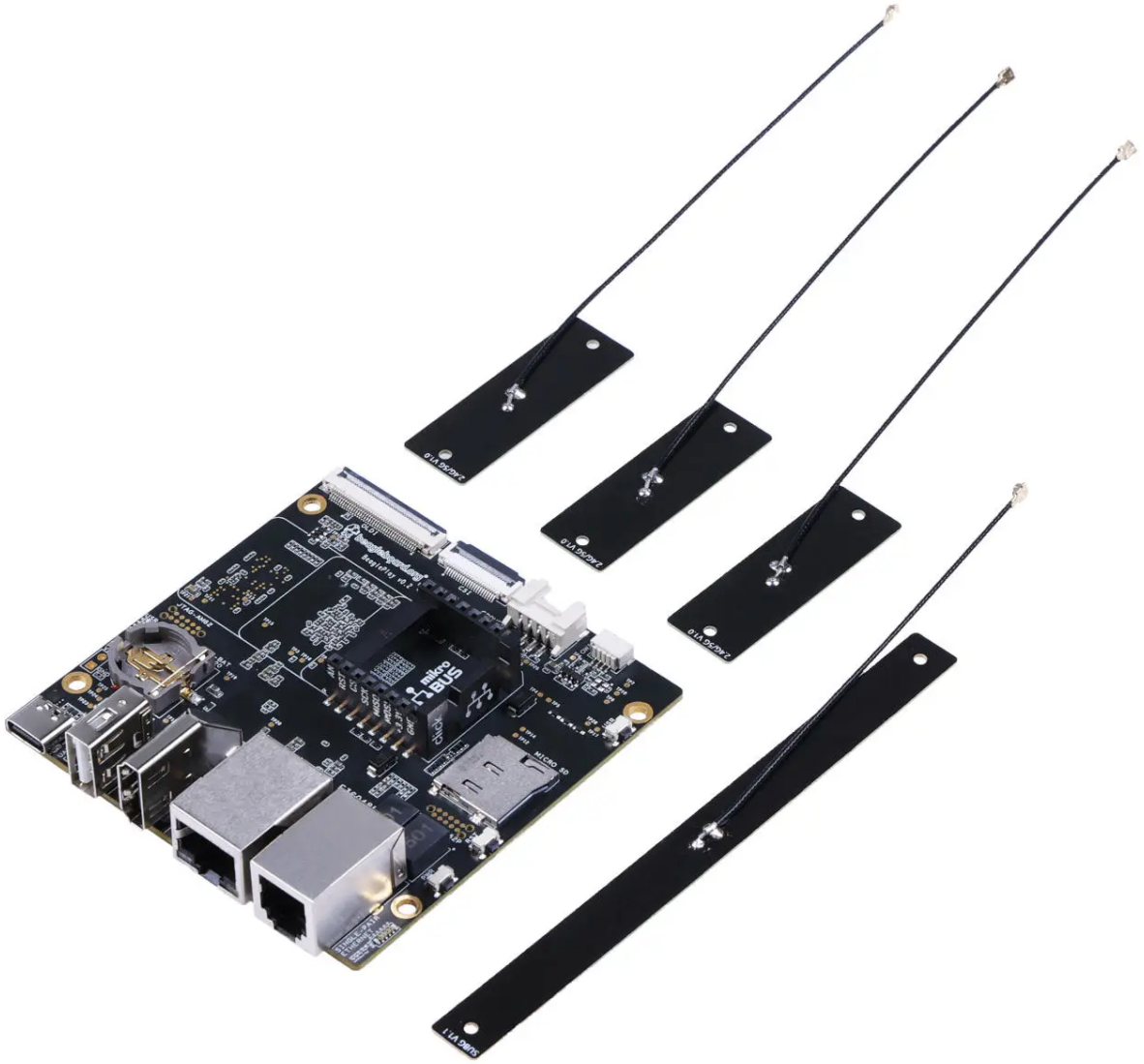
Quick Start Guide

2.1 What's included in the box?

When you purchase a brand new BeaglePlay, In the box you'll get:

1. BeaglePlay board
2. One (1) sub-GHz antenna
3. Three (3) 2.4GHz/5GHz antennas
4. Plastic standoff hardware
5. Quick-start card

Tip: For board files, 3D model, and more, you can checkout BeaglePlay repository on OpenBeagle.



2.2 Attaching antennas

You can watch this video to see how to attach the antennas.

2.3 Tethering to PC

Tip: Checkout beagleboard-getting-started for,

1. Updating to latest software.
2. Power and Boot.
3. Network connection.
4. Browsing to your Beagle.
5. Troubleshooting.

For tethering to your PC you'll need a USB-C data cable.

Fig. 2.1: https://youtu.be/8zeIVd-JRc0

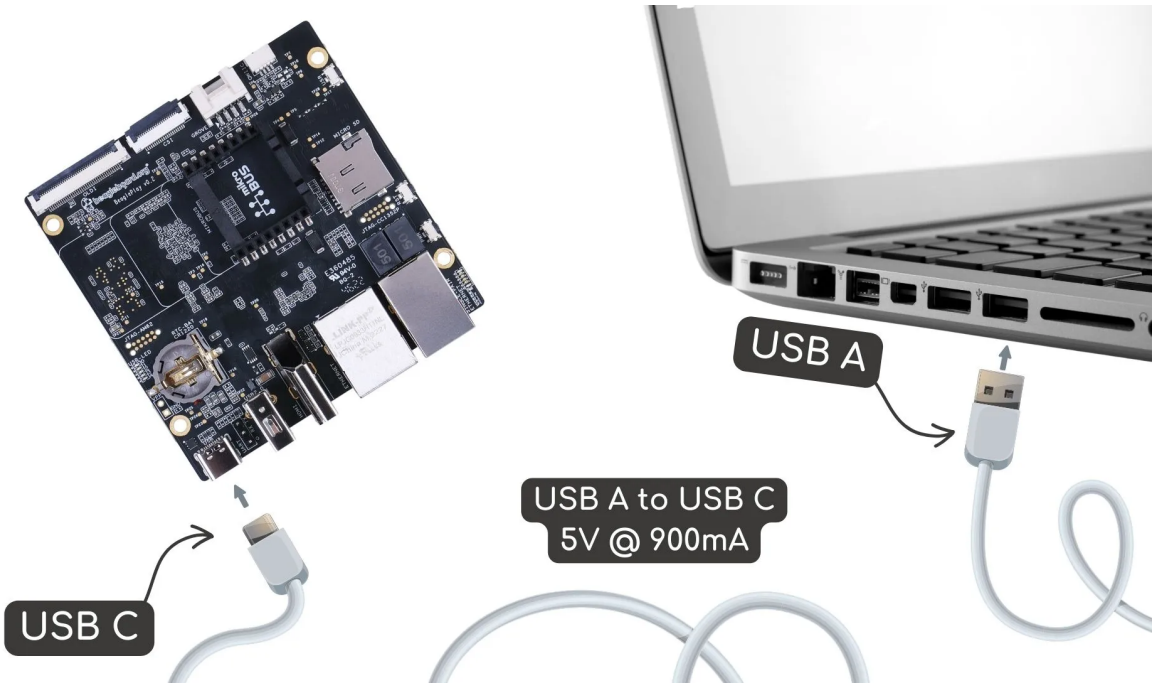


Fig. 2.2: Tethering BeaglePlay to PC

2.4 Access VSCode

You can access VSCode in two ways:

1. *USB*
2. *Access Point*

2.4.1 USB

Once connected, you can browse to 192.168.7.2:3000 to access the VSCode IDE to browse documents and start programming your BeaglePlay!

2.4.2 Access Point

By default BeaglePlay Access Point is enabled, You can connect to `BeaglePlay-XXXX` Access Point with the password `BeaglePlay` and then browse to 192.168.7.2:3000 to access the VSCode IDE.

Note: You may get a warning about an invalid or self-signed certificate. This is a limitation of not having a public URL for your board. If you have any questions about this, please as on https://forum.beagleboard.org/tag/play.

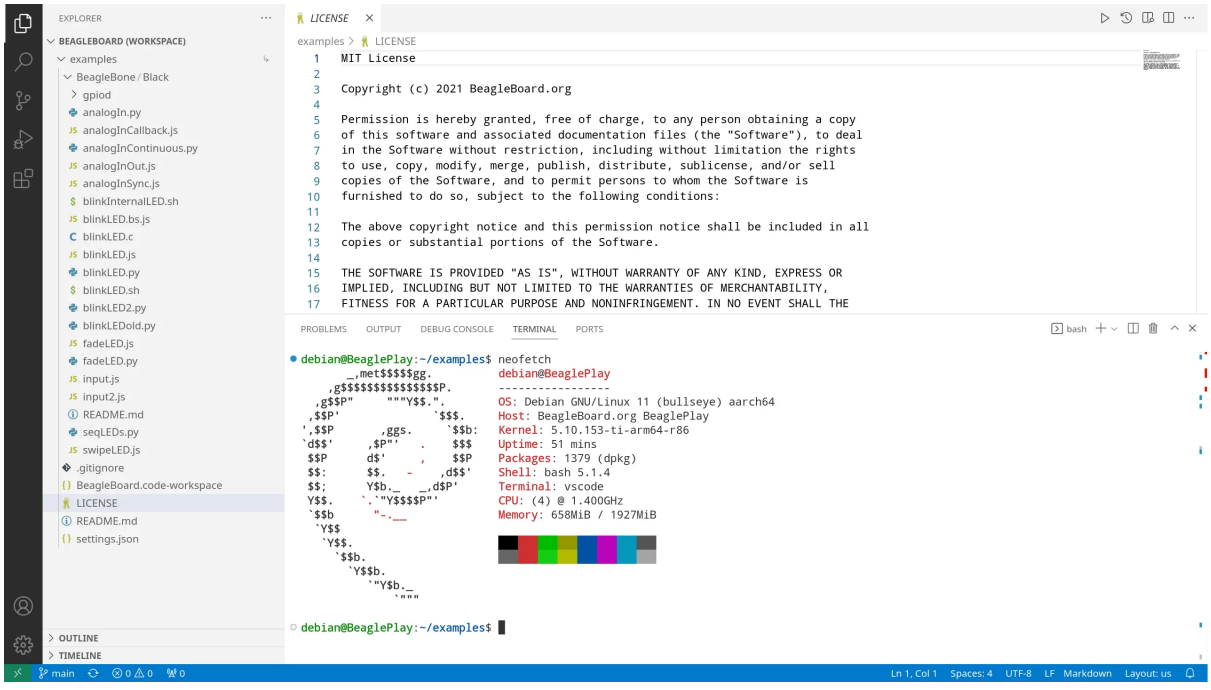


Fig. 2.3: BeaglePlay VSCode IDE (192.168.7.2:3000)

Tip: For more Wifi and Access Point related info go to *Connect WiFi*

2.5 Demos and Tutorials

- *Using Serial Console*

- *Connect WiFi*
- *Using QWIIC*
- *Using Grove*
- *Using mikroBUS*
- *Using OLDI Displays*
- *Using CSI Cameras*
- *Wireless MCU Zephyr Development*
- *BeaglePlay Kernel Development*
- *Understanding Boot*

Chapter 3

Design and specifications

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

Tip: You can download BeaglePlay schematic to have clear view of all the elements that makes up the BeaglePlay hardware.

BeaglePlay design repository

3.1 Block diagram

The block diagram below shows all the parts that makes up your BeaglePlay board. BeaglePlay as mentioned in previous chapters is based on AM6254 SoC which is shown in the middle. Connection of other parts like power supply, memory, storage, wifi, ethernet, and others is also clearly shown in the block diagram. This block diagram shows the high level specifications of the BeaglePlay hardware and the sections below this are going to show you the individual part in more detail with schematic diagrams.

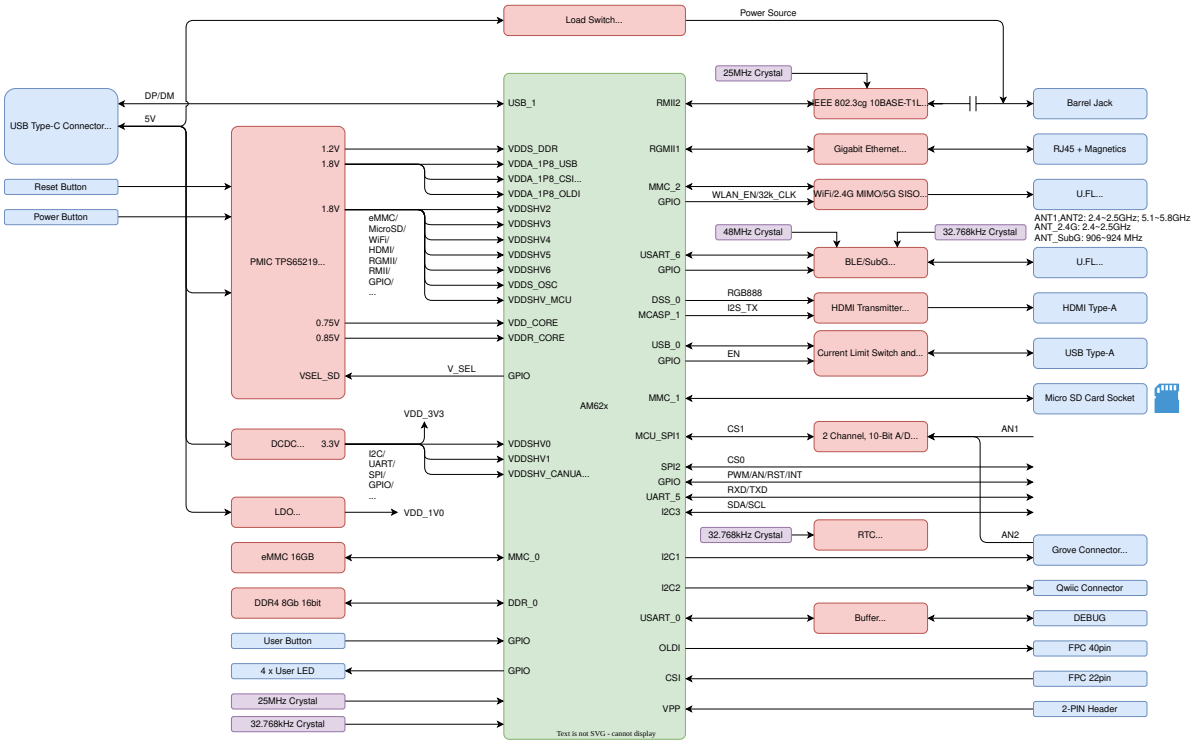
3.2 System on Chip (SoC)

AM62x Sitara™ Processors from Texas Instruments are Human-machine-interaction SoC with Arm® Cortex®-A53-based edge AI and full-HD dual display. AM6254 which is on your BeaglePlay board has a multi core design with Quad 64-bit Arm® Cortex®-A53 microprocessor subsystem at up to 1.4 GHz, Single-core Arm® Cortex®-M4F MCU at up to 400MHz, and Dedicated Device/Power Manager. Talking about the multimedia capabilities of the processor you can connect upto two display monitors with 1920x1080 @ 60fps each, additionally there is a OLDI/LVDS (4 lanes - 2x) and 24-bit RGB parallel interface for connecting external display panels. One 4 Lane CSI camera interface is also available which has support for 1,2,3 or 4 data lane mode up to 2.5Gbps speed. The list of features is very long and if you are interested to know more about the AM62x SoC you may take a look at AM62x Sitara™ Processors datasheet.

3.3 Power management

Different parts of the board requires different voltages to operate and to fulfill requirements of all the chips on BeaglePlay we have Low Drop Out (LDO) voltage regulators for fixed voltage output and Power Management Integrated Circuit (PMIC) that interface with SoC to generate software programable voltages. 2 x LDOs and 1 x PMIC used on BeaglePlay are shown below.

BeaglePlay System Block Diagram



3.3.1 1.0V LDO

TLV75801 is an adjustable 500-mA low-dropout (LDO) regulator. Consumes very low quiescent current and provides fast line and load transient performance. The TLV758P features an ultra-low dropout of 130 mV at 500 mA that can help improve the power efficiency of the system. The TLV758P is stable with small ceramic output capacitors, allowing for a small overall solution size. A precision band-gap and error amplifier provides high accuracy of 0.7% (max) at 25°C and 1% (max) over temperature (85°C). This device includes integrated thermal shutdown, current limit, and undervoltage lockout (UVLO) features. The TLV758P has an internal foldback current limit that helps reduce the thermal dissipation during short-circuit events.

TLV75801 provides 1.0V required by the single-pair Ethernet PHY (U13 - DP83TD510ERHBR). It was decided this was less likely to be needed than the other rails coming off of the primary PMIC and therefore was given its own regulator when running low on power rails.

Note: The voltage drop from 1.8V to 1.0V is rated up to 0.3A (240mW), but the typical current from the DP83TD51E data sheet (SNLS656C) is stated at 3.5mA (2.8mW) and the maximum is 7.5mA (6mW). This isn't overly significant on a board typically consuming 400mA at 5V (2W). However, this is an area where some power optimization could be performed if concerned about sleep modes.

3.3.2 3.3V DCDC buck

TLV62595 is a high-frequency synchronous step-down converter optimized for compact solution size and high efficiency. The device integrates switches capable of delivering an output current up to 4 A. At medium to heavy loads, the converter operates in pulse width modulation (PWM) mode with typical 2.2-MHz switching frequency. At light load, the device automatically enters Power Save Mode (PSM) to maintain high efficiency over the entire load current range with a quiescent current as low as 10 µA.

This provides 3.3V for the vast majority of 3.3V I/Os on the board, off-board 3.3V power to microSD, mikroBUS, QWIIC and Grove connectors, as well as to the PMIC LDO to provide power for the 1.8V on-board I/Os, DDR4, and

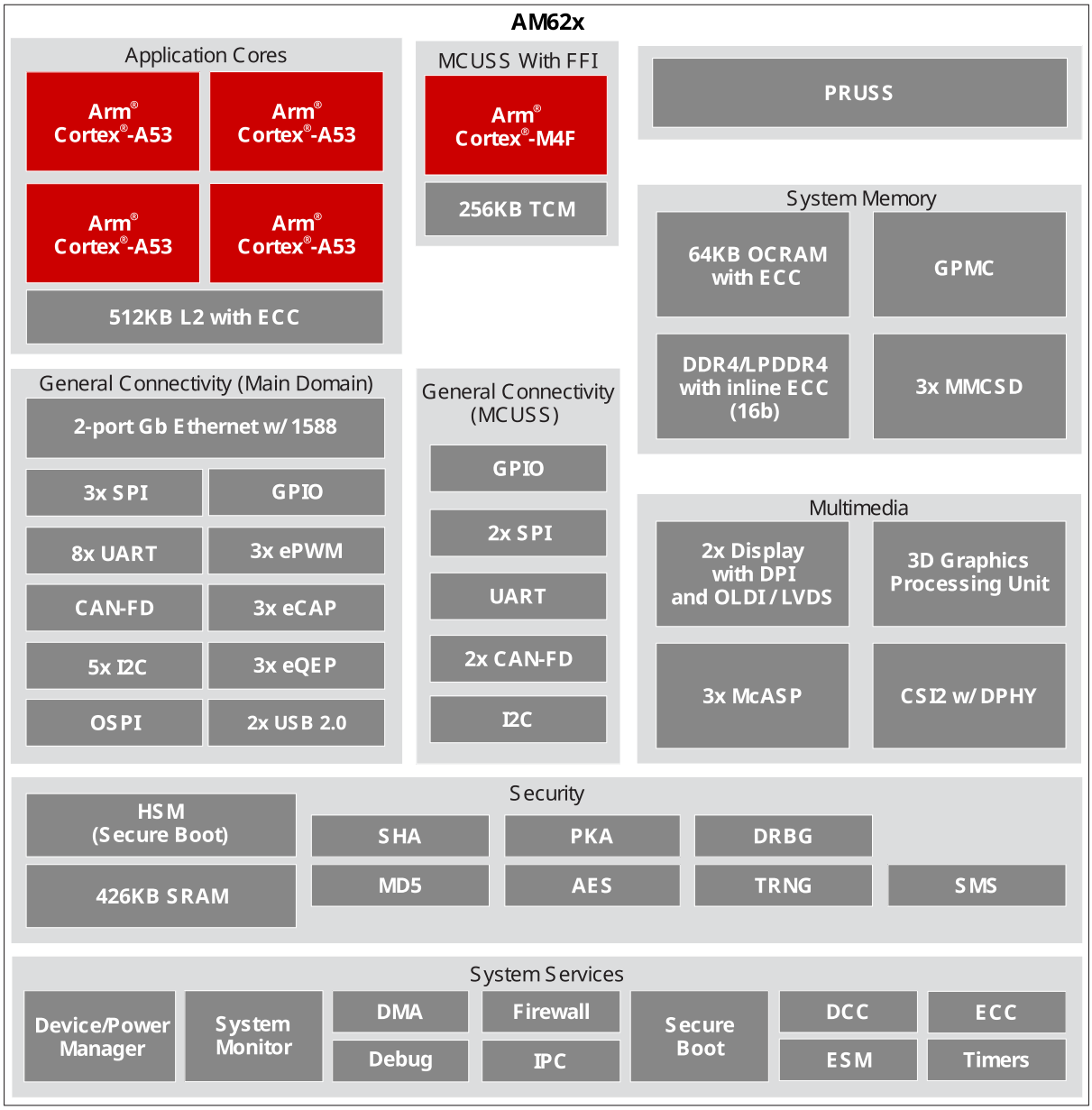
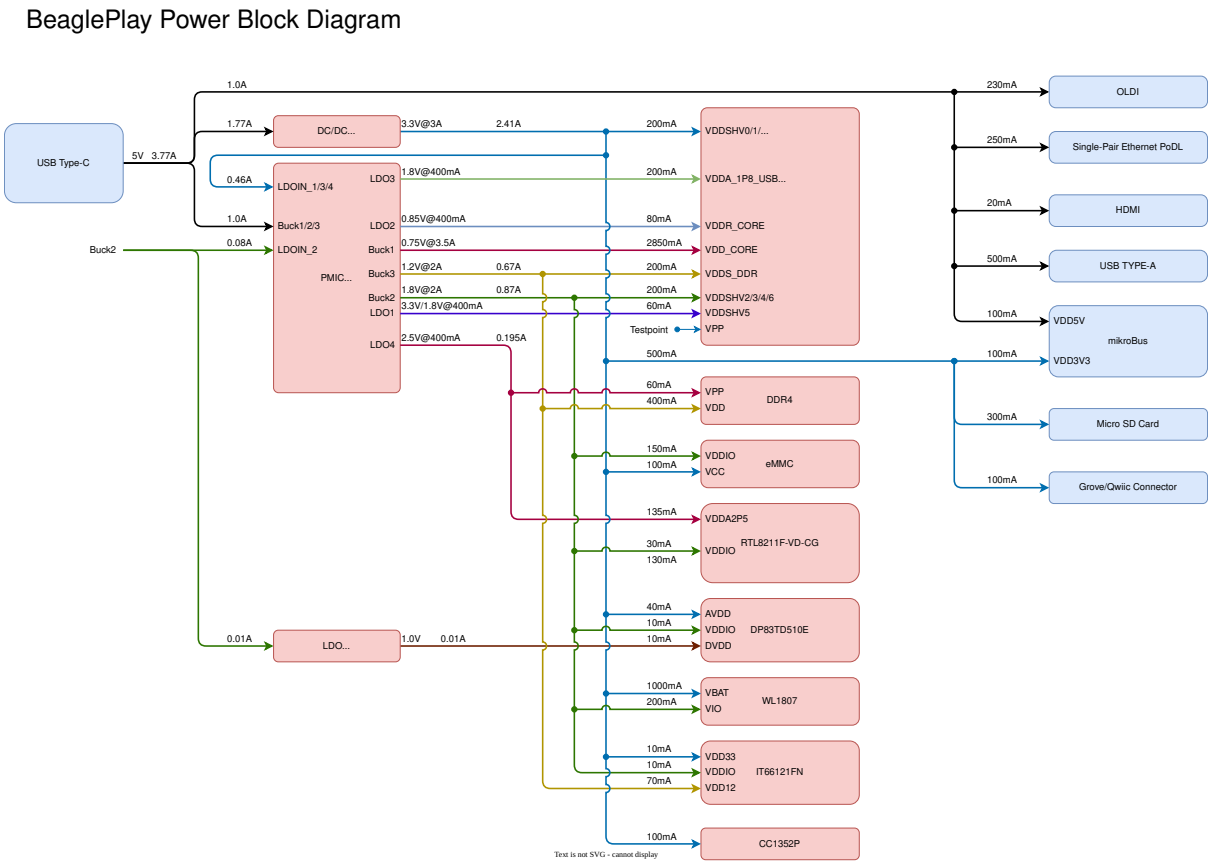


Fig. 3.1: AM6254 SoC block diagram



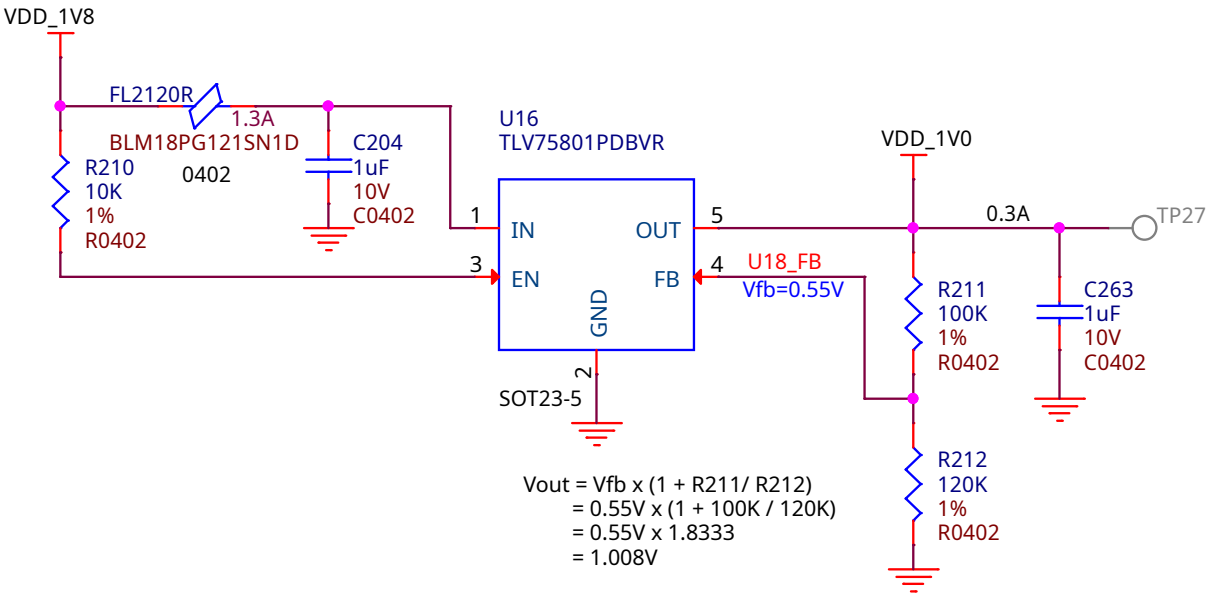


Fig. 3.2: TLV75801PDBVR LDO schematic for 1V0 output

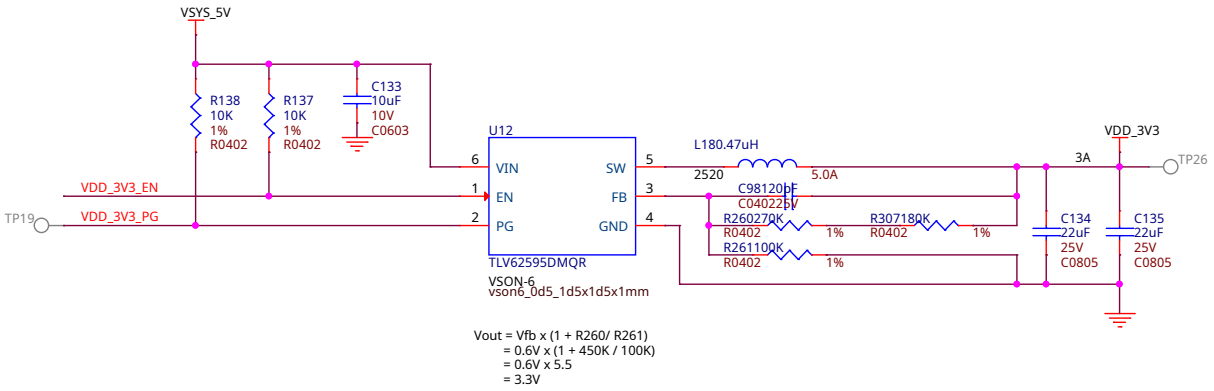


Fig. 3.3: TLV62595DMQ step-down regulator schematic for 3V3 output

gigabit Ethernet PHY. Due to the relatively high current rating (3A), a highly efficient (up to 97%) was chosen.

Note: The primary TPS65219 PMIC firmware uses GPO2 to provide the enable signal (VDD_3V3_EN). The power-good signal (VDD_3V3_PG) is available at TP19 and is unused on the rest of the board.

3.3.3 PMIC

The TPS65219 is a Power Management IC (PMIC) designed to supply a wide range of SoCs in both portable and stationary applications. The DC-DC converters are capable of 1x 3.5 A and 2x 2 A. The converters require a small 470 nH inductor, 4.7 µF input capacitance, and a minimum 10 µF output capacitance per rail. Two of the LDOs support output currents of 400 mA at an output voltage range of 0.6 V to 3.4 V. These LDOs support bypass mode, acting as a load- switch, and allow voltage-changes during operation. The other two LDOs support output currents of 300 mA at an output voltage range of 1.2 V to 3.3 V. The LDOs also support load-switch mode. The I2C-interface, IOs, GPIOs and multi-function-pins (MFP) allow a seamless interface to a wide range of SoCs.

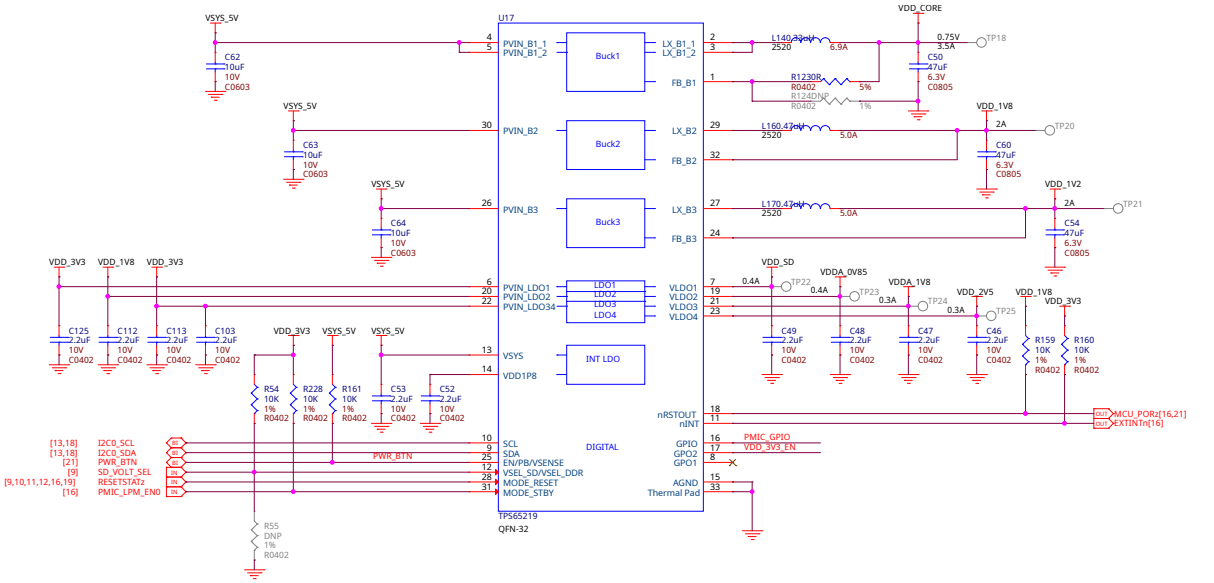


Fig. 3.4: TPS65219 Power Management Integrated Circuit (PMIC) schematic

This is the primary power management integrated circuit (PMIC) for the design. It coordinates the power sequencing and provides numerous power rails required for the core of the system, including dynamic voltages

for the processor core and microSD card. The TPS6521903 variant is used for this DDR4-based system. The 03 at the end indicates the sequencing programmed into the device and is covered in the TPS6521903 Technical Reference Manual SLVUCJ2.

Todo: Add specific power-up/down sequence notes here as well a highlight any limitations and known issues.

3.4 General Connectivity and Expansion

One of the main advantage of using a Single Board Computer (SBC) is having direct accessibility of general purpose input & output (GPIO) pins and other interfaces like I2C, SPI, ADC, PWM. Your BeaglePlay board shines in this domain as well with mikroBUS connector that can take 1000s of click board from MikroElektronika, Grove connector allows to connect hundreds of Grove modules from Seeed Studio, and QWIIC connector allows to connect I2C modules like QWIIC modules from SparkFun or STEMMA QT modules from Adafruit. Note that you also get one USB-A port and one USB-C port. BeaglePlay's USB-A port with host support enables you to connect any USB device like your keyboard & mouse. The USB-C connector allows you to power the board and to connect the board to a PC. You can then connect via SSH or use the pre-installed VisualStudio Code editor by putting the address `192.168.7.2:3000` in your web browser.

3.4.1 USB A & USB C

Below is the schematic of full size USB A for peripheral connection and USB C for device power & tethering.

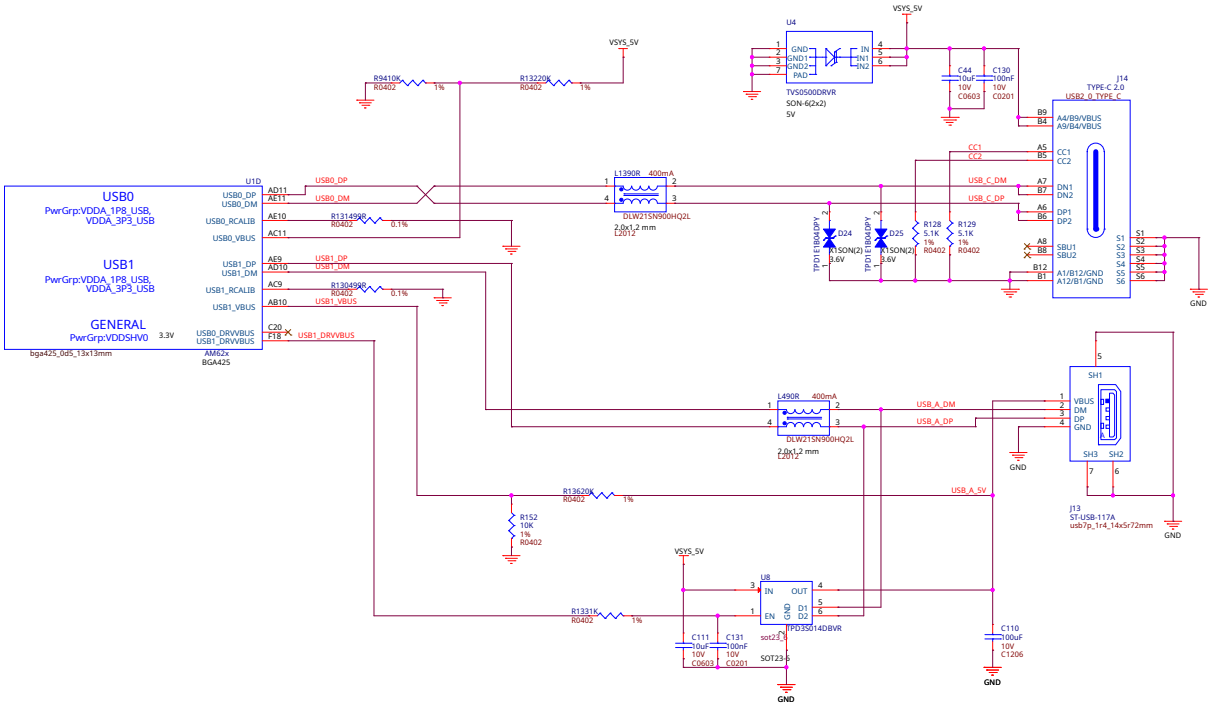


Fig. 3.5: USB-A and USB-C

3.4.2 2ch 10bit ADC

The ADC102S051 is a low-power, two-channel CMOS 10-bit analog-to-digital converter with a high- speed serial interface. Unlike the conventional practice of specifying performance at a single sample rate only, the ADC102S051 is fully specified over a sample rate range of 200 ksps to 500 ksps. The converter is based on a

successive-approximation register architecture with an internal track-and-hold circuit. It can be configured to accept one or two input signals at inputs IN1 and IN2. The output serial data is straight binary, and is compatible with several standards, such as SPI, QSPI, MICROWIRE, and many common DSP serial interfaces. We ar using it over SPI. The ADC102S051 operates with a single supply that can range from +2.7V to +5.25V. Normal power consumption using a +3V or +5V supply is 2.7 mW and 8.6 mW, respectively. The power-down feature reduces the power consumption to just 0.12 µW using a +3V supply, or 0.47 µW using a +5V supply.

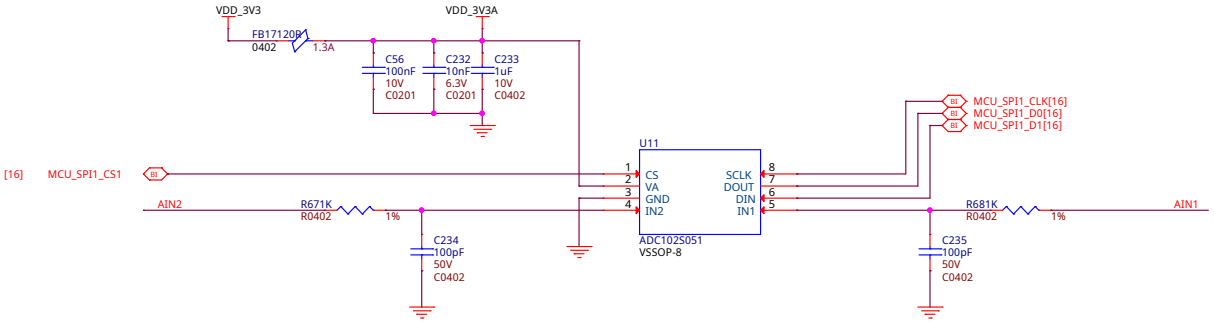


Fig. 3.6: ADC102S051 - 12bit Analog to Digital Converter (ADC)

3.4.3 mikroBUS

mikroBUS is a standard specification by MikroElektronika that can be freely used by anyone following the guidelines. It includes SPI, I2C, UART, PWM, ADC, reset, interrupt, and power (3.3V and 5V) connections to common embedded peripherals.

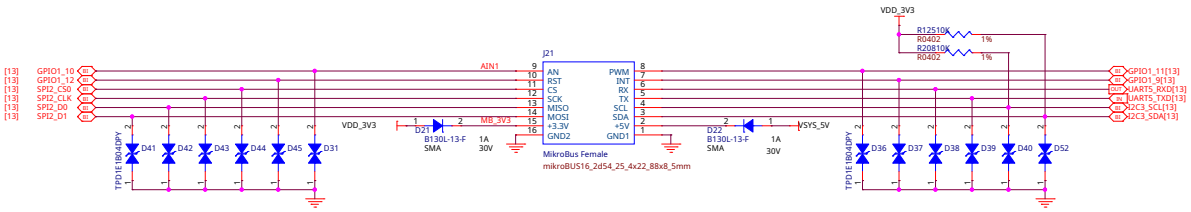


Fig. 3.7: mikroBUS connector schematic

3.4.4 Grove

Seeed Studio Grove System is a modular, standardized connector prototyping ecosystem. The Grove System takes a building block approach to assembling electronics. Compared to the jumper or solder based system, it is easier to connect devices to an application, simplifying the learning system

3.4.5 QWIIC

Qwiic, or STEMMA QT are 4pin JST SH 1.00 connectors for easy I2C connection.

3.5 Buttons and LEDs

To interact with the Single Board Computers we use buttons for input and LEDs for visual feedback. On your BeaglePlay board you will find 3 buttons each with a specific purpose: power, reset, and user. For visual feedback you will find 5 user LEDs near USB-C port and 6 more indicator LEDs near your BeaglePlay's Single Pair ethernet port. Schematic diagrams below show how these buttons and LEDs are wired.

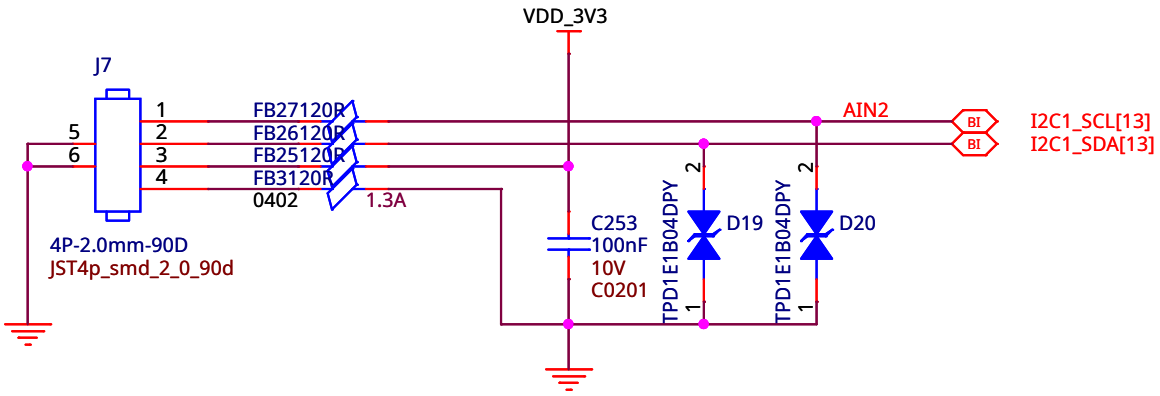


Fig. 3.8: Grove connector schematic

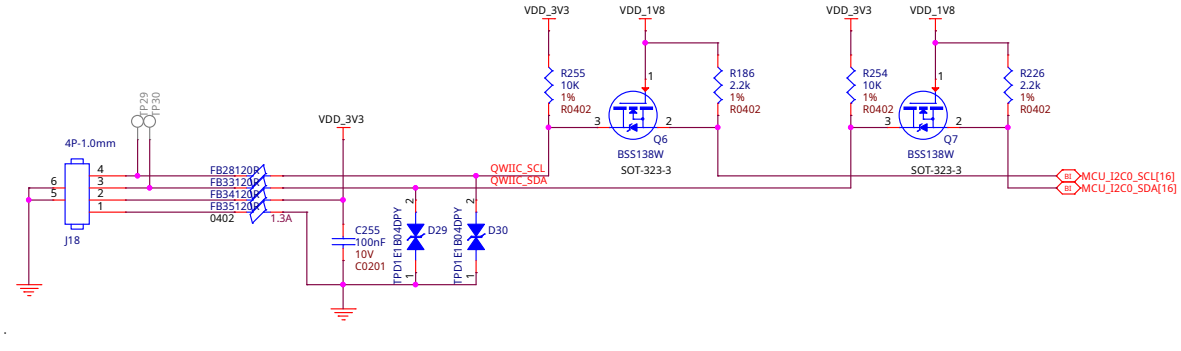
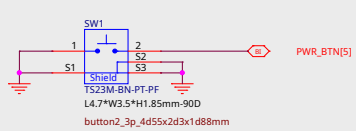
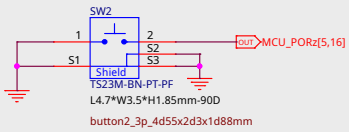
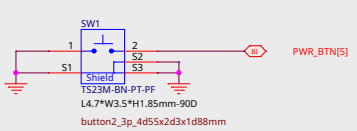


Fig. 3.9: QWIIC connector for I2C modules

3.5.1 Buttons

Power, Reset and User buttons for turning board ON/OFF, resetting board, and boot selection or user assigned control.

Table 3.1: BeaglePlay buttons

Power	Reset	User
		

3.5.2 LEDs

Power and user LEDs for status and general purpose usage.

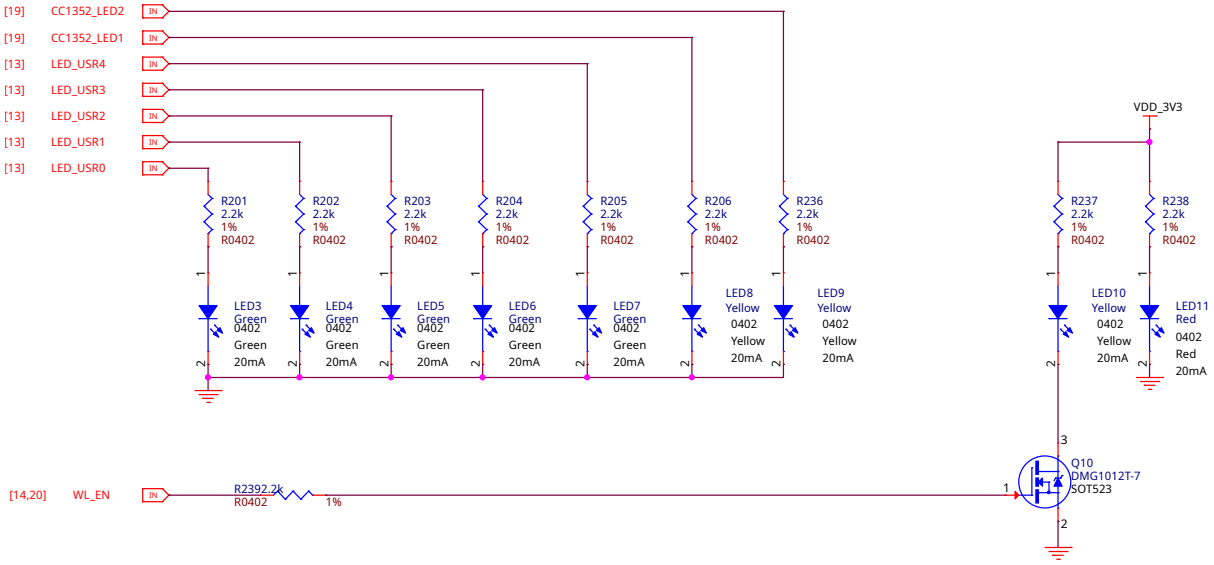


Fig. 3.10: BeaglePlay LEDs

3.6 Wired and wireless connectivity

For internet connection or general connectivity between BeaglePlay and other devices.

3.6.1 Gigabit ethernet

The Realtek RTL8211F-CG is a highly integrated Ethernet transceiver that is compatible with 10Base-T, 100Base-TX, and 1000Base-T IEEE 802.3 standards. It provides all the necessary physical layer functions to transmit and receive Ethernet packets over CAT.5 UTP cable. The RTL8211F(I)-CG uses state-of-the-art DSP technology and an Analog Front End (AFE) to enable high-speed data transmission and reception over UTP cable. Functions such as Crossover Detection & Auto-Correction, polarity correction, adaptive equalization, cross-talk cancellation, echo cancellation, timing recovery, and error correction are implemented in the RTL8211F(I)-CG to provide robust transmission and reception capabilities at 10Mbps, 100Mbps, or 1000Mbps.

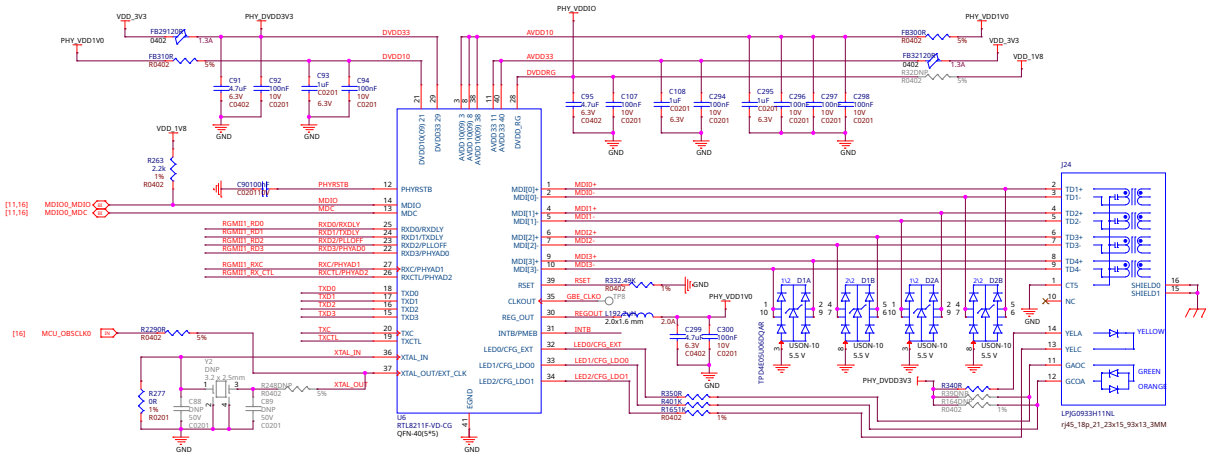


Fig. 3.11: Gigabit ethernet

3.6.2 Single pair ethernet

The DP83TD510E is an ultra-low power Ethernet physical layer transceiver compliant with the IEEE 802.3cg 10Base-T1L specification. The PHY has very low noise coupled receiver architecture enabling long cable reach and very low power dissipation. The DP83TD510E has external MDI termination to support intrinsic safety requirements. It interfaces with MAC layer through MII, Reduced MII (RMII) , RGMII, and RMII low power 5-MHz master mode. It also supports RMII back-to-back mode for applications that require cable reach extension beyond 2000 meters. It supports a 25MHz reference clock output to clock other modules on the system. The DP83TD510E offers integrated cable diagnostic tools; built-in self- test, and loopback capabilities for ease of design or debug

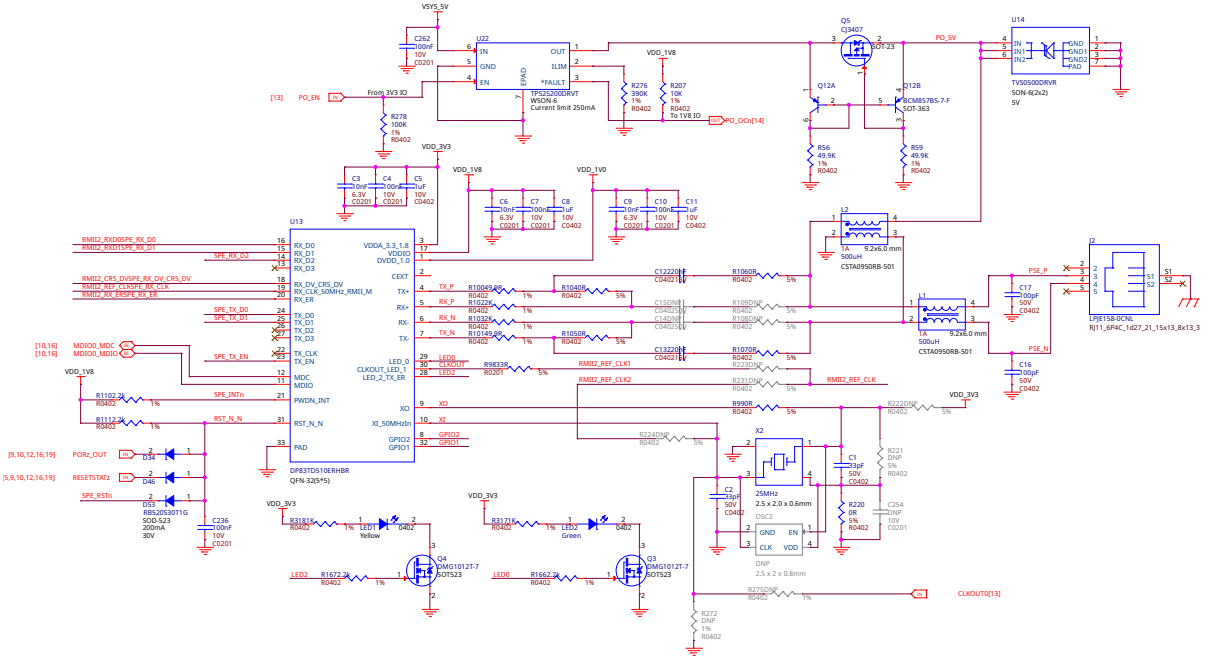


Fig. 3.12: Single pair ethernet

3.6.3 WiFi 2.4G/5G

The WL18x7MOD is a Wi-Fi, dual-band, 2.4- and 5-GHz module solution with two antennas supporting industrial temperature grade. The device is FCC, IC, ETSI/CE, and TELEC certified for AP (with DFS support) and client. TI offers drivers for high-level operating systems, such as Linux® and Android™. Additional drivers, such as WinCE and RTOS, which includes QNX, Nucleus, ThreadX, and FreeRTOS, are supported through third parties.

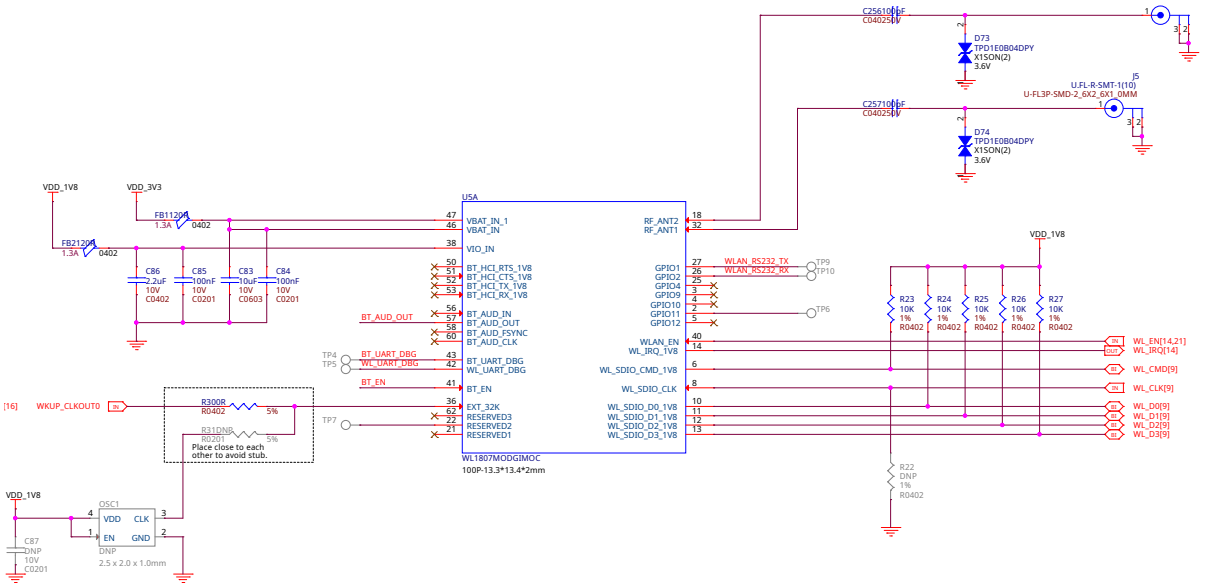


Fig. 3.13: WL1807MOD dual-band (2.4G/5G) WiFi

3.6.4 BLE & SubGHz

The SimpleLink™ CC1352P7 device is a multiprotocol and multi-band Sub-1 GHz and 2.4-GHz wireless microcontroller (MCU) supporting Thread, Zigbee®, Bluetooth® 5.2 Low Energy, IEEE 802.15.4g, IPv6-enabled smart objects (6LoWPAN), mioty®, Wi-SUN®, proprietary systems, including the TI 15.4-Stack (Sub-1 GHz and 2.4 GHz), and concurrent multiprotocol through a Dynamic Multiprotocol Manager (DMM) driver. The CC1352P7 is based on an Arm® Cortex® M4F main processor and optimized for low-power wireless communication and advanced sensing in grid infrastructure, building automation, retail automation, personal electronics and medical applications.

3.7 Memory, Media and Data storage

3.7.1 DDR4

3.7.2 eMMC/SD

3.7.3 microSD Card

3.7.4 Board EEPROM

3.8 Multimedia I/O

3.8.1 HDMI

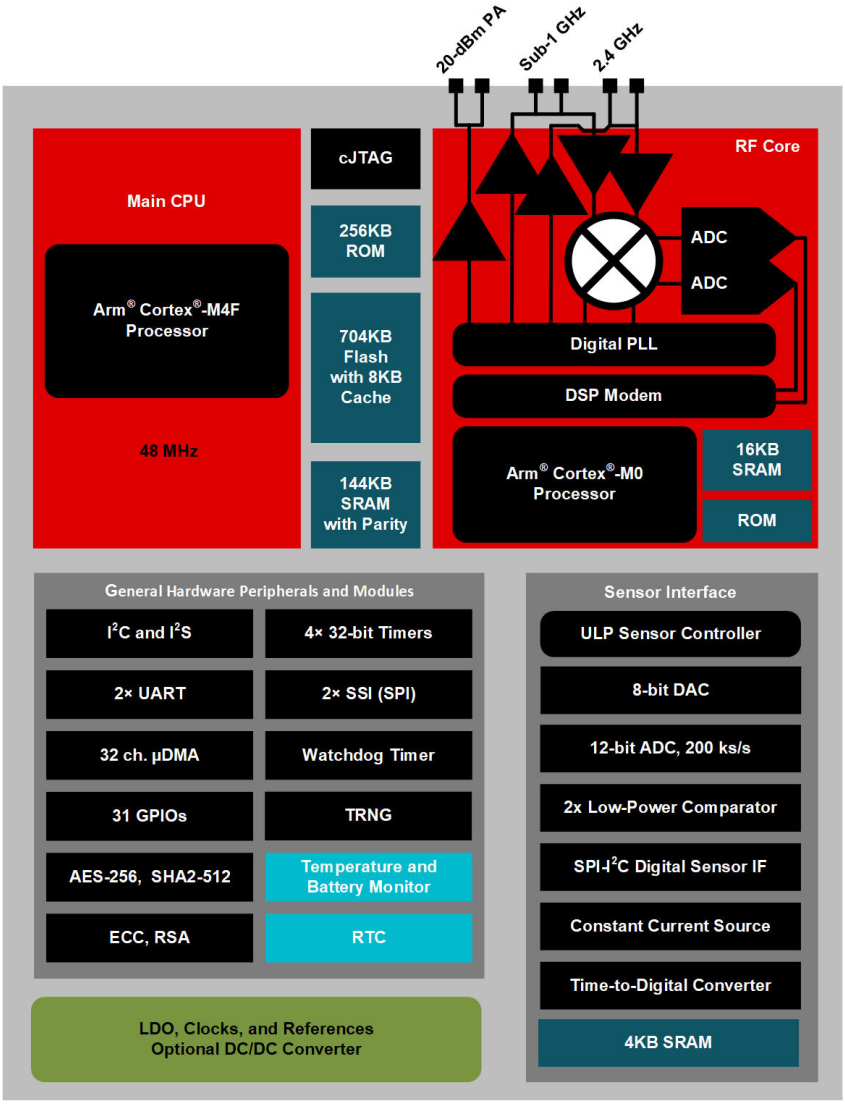


Fig. 3.14: CC1352P7 block diagram

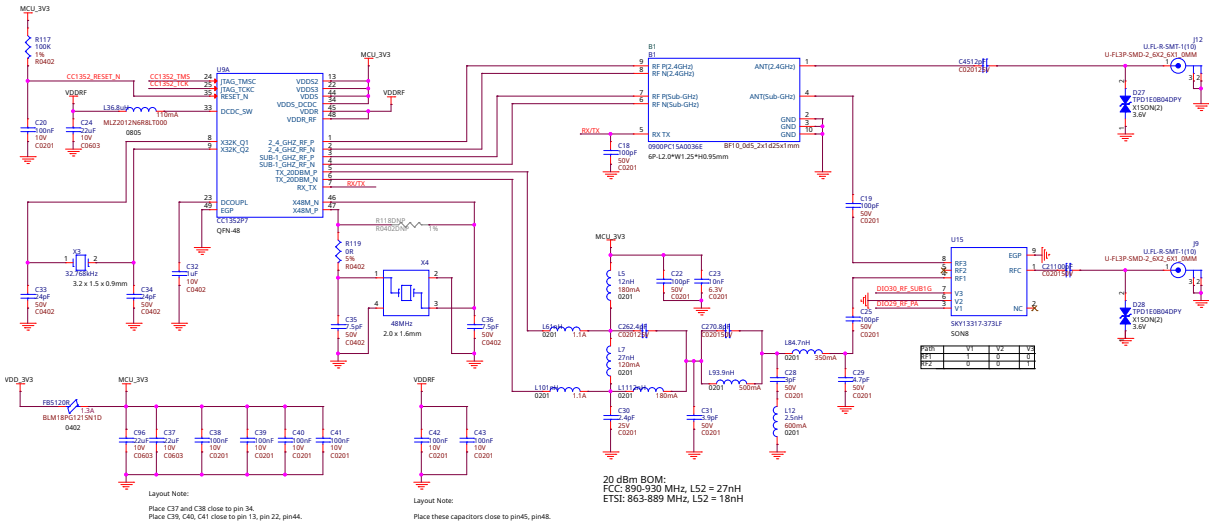


Fig. 3.15: CC1352P7 Bluetooth Low Energy (BLW) and SubGHz connectivity

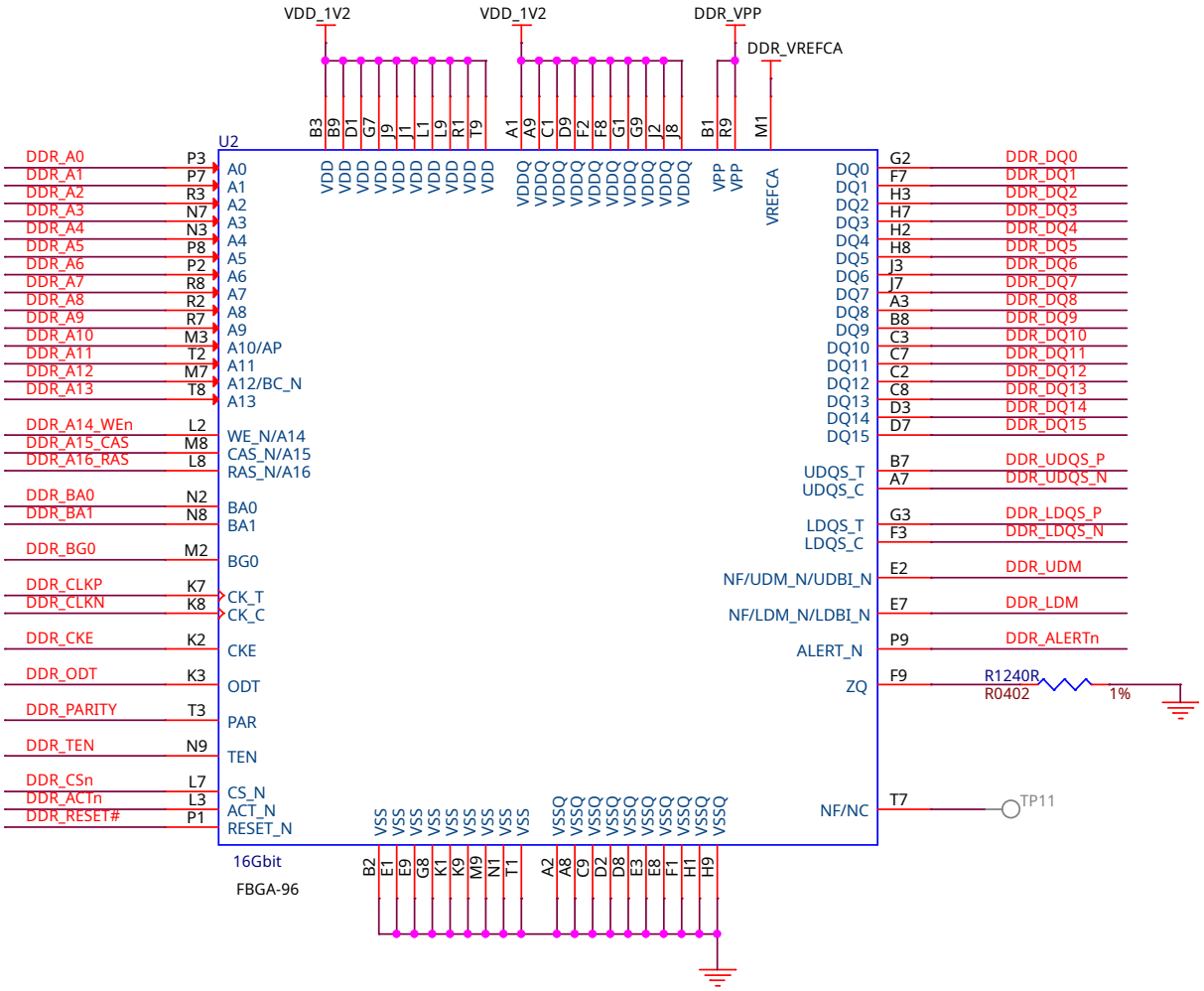


Fig. 3.16: DDR4 Memory

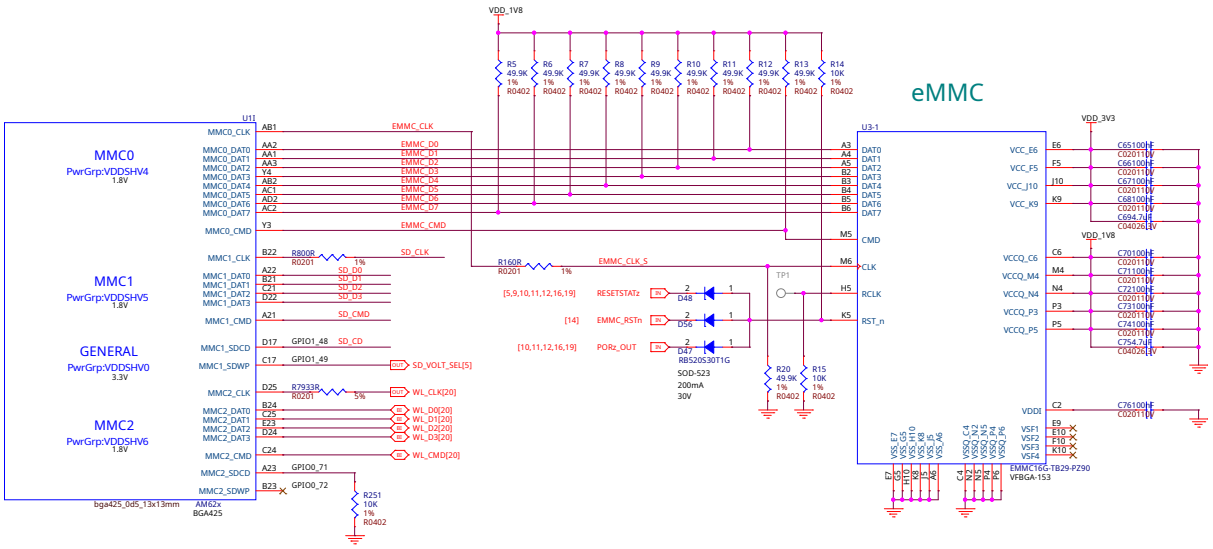


Fig. 3.17: eMMC/SD storage

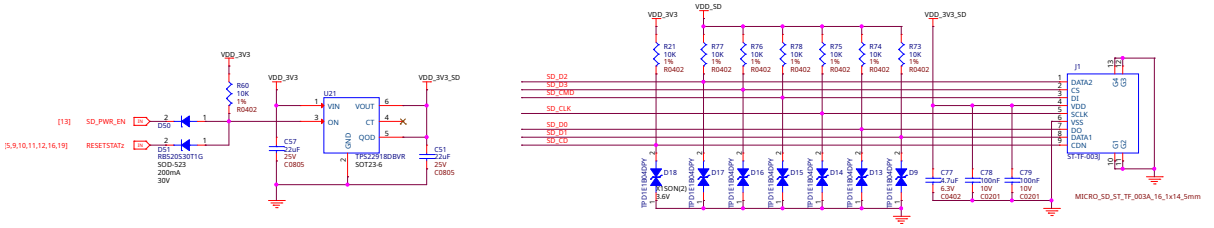


Fig. 3.18: microSD Card storage slot

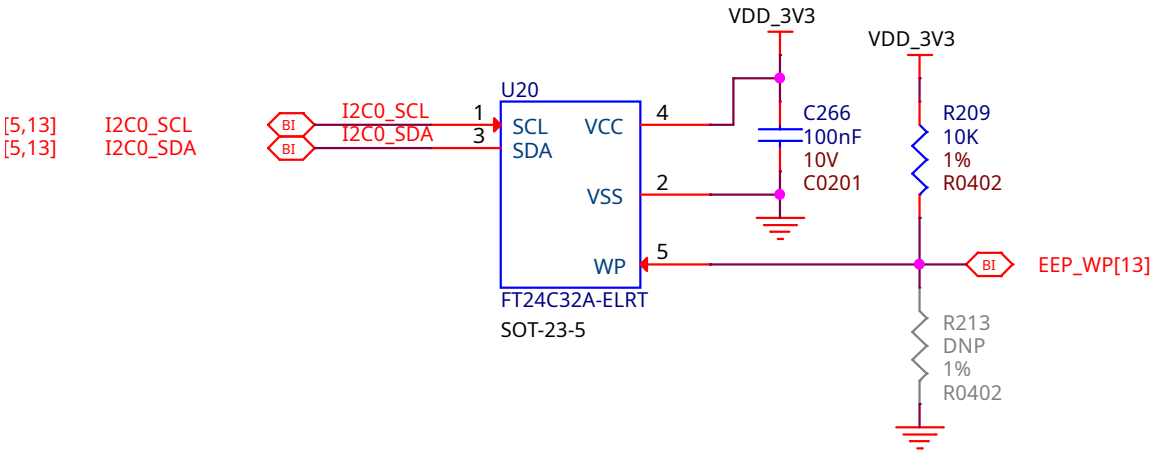


Fig. 3.19: Board EEPROM ID

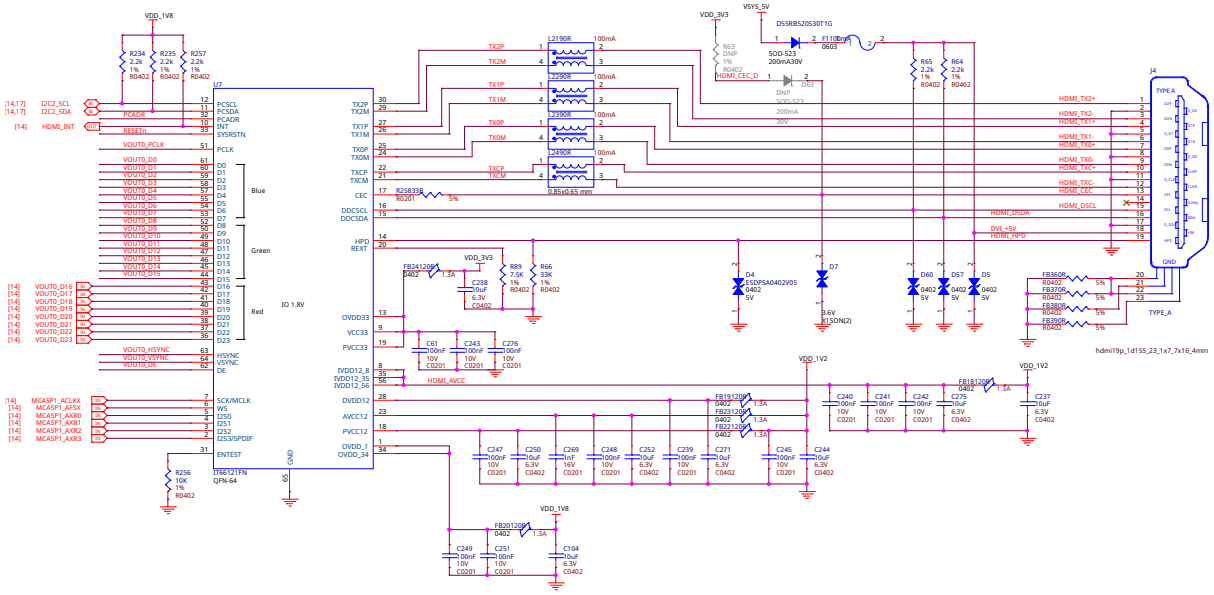


Fig. 3.20: HDMI output

3.8.2 OLDI

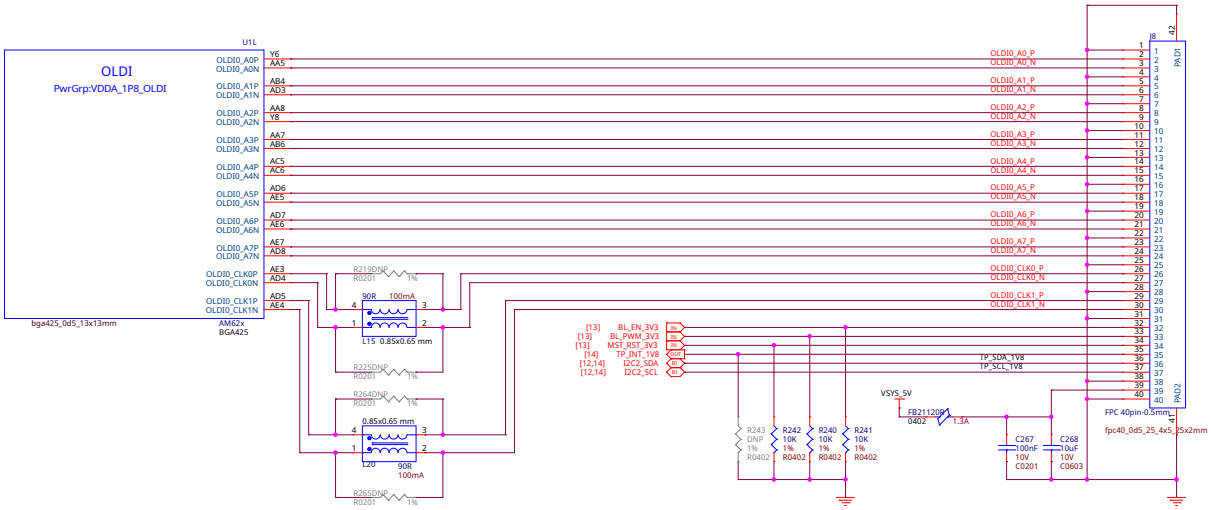


Fig. 3.21: OLDI display interface

3.8.3 CSI

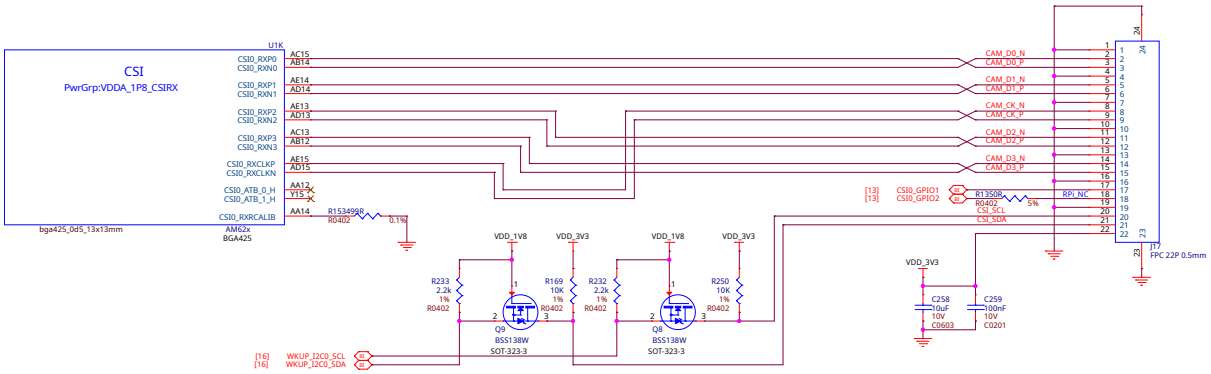


Fig. 3.22: CSI camera interface

3.9 RTC & Debug

3.9.1 RTC

3.9.2 UART Debug Port

3.9.3 AM62x JTAG & TagConnect

3.9.4 CC1352 JTAG & TagConnect

3.10 Mechanical Specifications

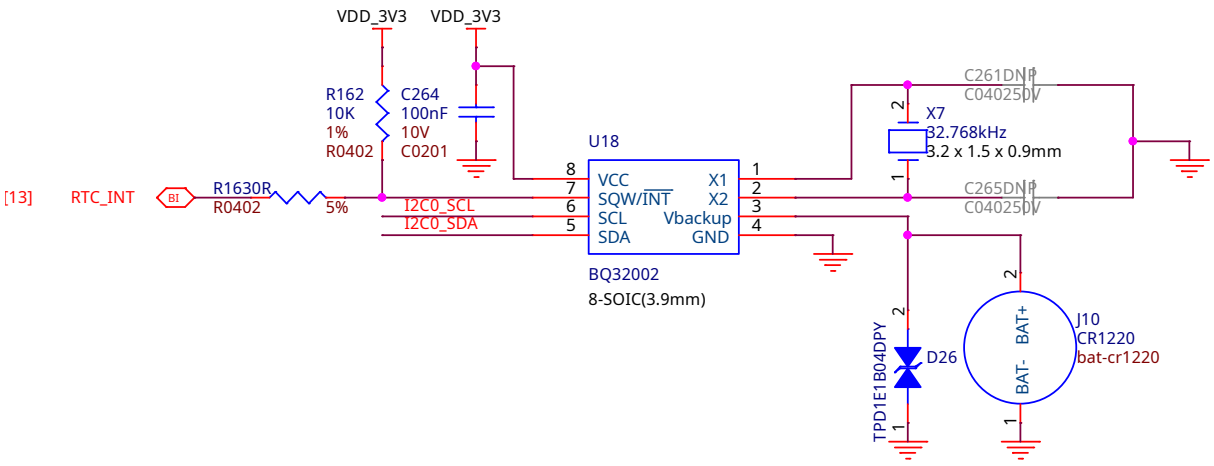


Fig. 3.23: Real Time Clock (RTC)

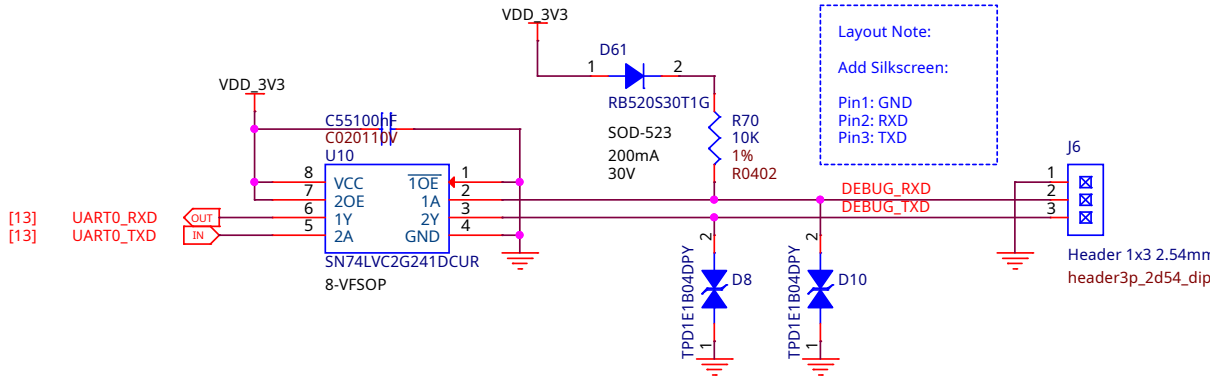


Fig. 3.24: UART debug port

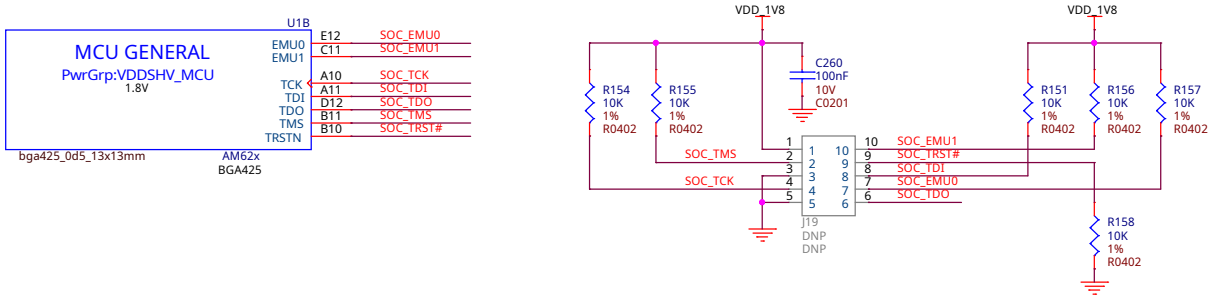


Fig. 3.25: AM62 JTAG debug port and TagConnect interface

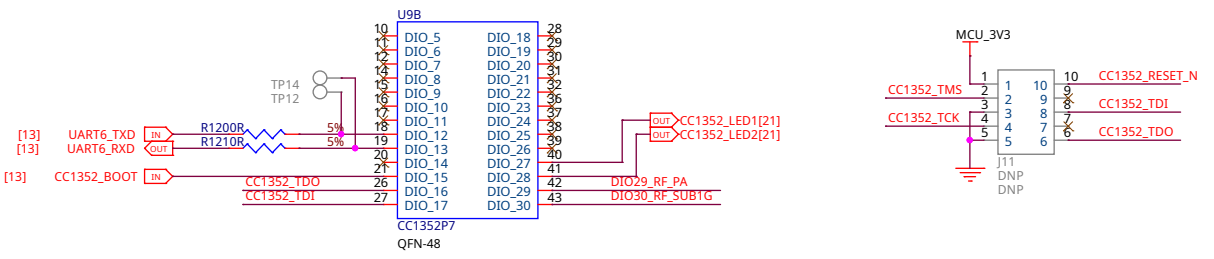


Fig. 3.26: CC1352 JTAG debug port and TagConnect interface

3.10.1 Dimensions & Weight

Table 3.2: Dimensions & weight

Parameter	Value
Size	82.5x80x20mm
Max heigh	20mm
PCB Size	80x80mm
PCB Layers	8 layers
PCB Thickness	1.6mm
RoHS compliant	Yes
Weight	55.3g

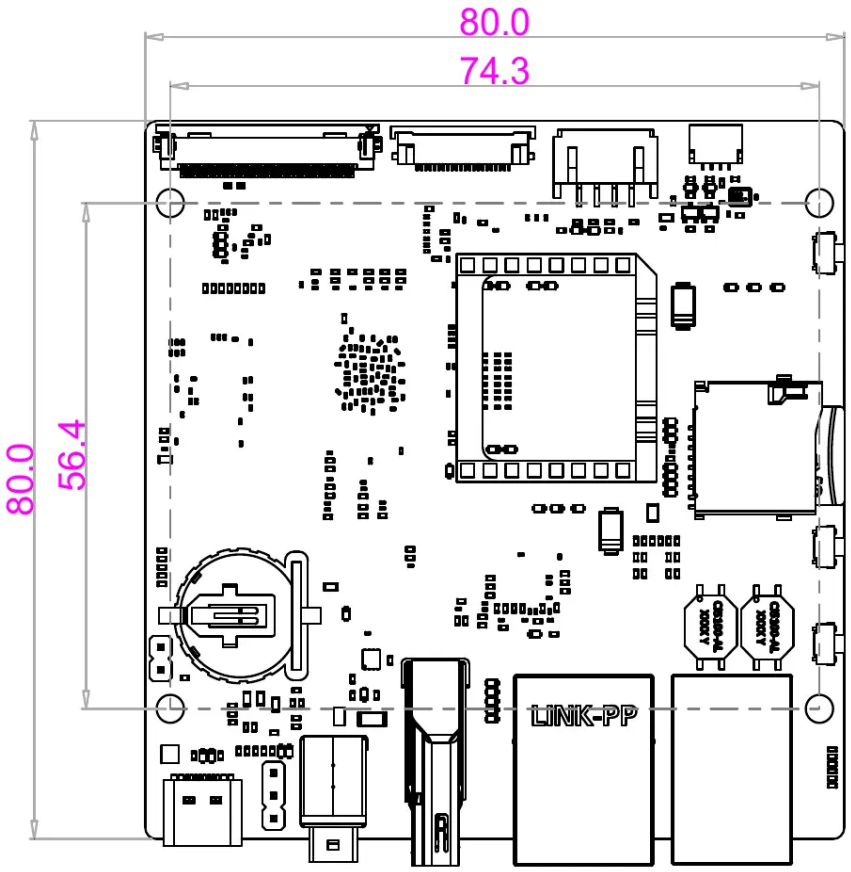


Fig. 3.27: BeaglePlay board dimensions

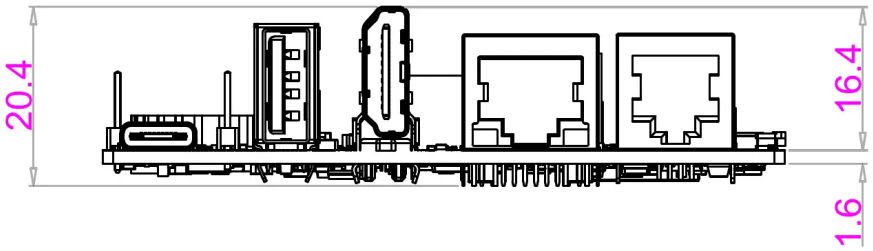


Fig. 3.28: BeaglePlay board side dimensions

Chapter 4

Expansion

Todo: Add information on building expansion hardware for BeaglePlay.

4.1 mikroBUS

The mikroBUS header provides several GPIO pins as well as UART, I2C, SPI, PWM and an Analog Input.

By default, the port is controlled by a mikroBUS driver that helps with auto-detecting MikroE Click Board that feature ClickID. This does however mean that if you want to manually control the port, you may need to first disable the driver.

To disable the driver, do the following - TODO

4.2 Grove

The Grove port on BeaglePlay exposes one of the SoC I2C Ports as well as an analog input.

It maps directly in linux as /dev/I2C-TODO or as the following alias /dev/play/grove

4.3 QWIIC

The QWIIC port on BeaglePlay exposes one of the SoC I2C Ports.

It maps directly in linux as */dev/I2C-2* or as the following alias */dev/play/qwiic*

4.4 CSI

The AM62x SoC (and by extension BeaglePlay) does not feature on-board ISP (Image Signal Processor) hardware, and as such, Raw-Bayer CSI Sensors must be pre-processed into normal images by the A53 cores.

To avoid performance penalties related to the approach above, it is recommended to use a sensor with a built-in ISP, such as the OV5640 which is supported out of box.

The PCam5C from Digilent is one CSI camera that features this sensor.

Note: Since BeaglePlay uses a 22-pin CSI connector, a 15 pin to 22 pin CSI adapter may also be required such as this one

Once installed, there are some software changes required to load the device driver at boot for the OV5640.

We will need to modify the following file: /boot/firmware/extlinux/extlinux.conf

We will add the following line to load the OV5640 DTBO:

```
fdtoverlays /overlays/k3-am625-beagleplay-csi2-ov5640.dtbo
```

Then you can reboot: sudo reboot

Camera should now work, you can use mplayer to test.

```
sudo apt-get install -y mplayer

mplayer tv: // -tv driver=v4l2:device=/dev/
↪video0:width=640:height=480:fps=30:outfmt=yuy2
```

4.5 OLDI

BeaglePlay brings out two OLDI (LVDS) channels, each with up to four data lanes and one clock lane to support 21/28-bit serialized RGB pixel data and synchronization transmissions. The first port, OLDI0, consists of OLDI0_A0-3/CLK0 and corresponds to odd pixels, while the second port, OLDI1, consists of OLDI0_A4-7/CLK1 and corresponds to even pixels.

It is pin compatible with the following two displays from Lincoln Technology Solutions:

Both displays have the following features and only differ in bezzle type:

- **Resolution** - 1920x1200 (16:10)
- **LCD Size (diagonal)** - 10.1”
- **Refresh Rate** - 60Hz
- **Brightness** - 1000nit
- **Pannel Type** - Edge-lit IPS
- **Touch Enabled** - Yes, Capacitive
- **Connector** - 40 pin FFC ribbon cable

A “Flush Coverglass” Version A “Oversized Cover Glass” Version - similar in style to a Tablet Display

To enable OLDI display support, modify the following file: /boot/firmware/extlinux/extlinux.conf

Then, add the following line to load the Lincoln LCD185 OLDI DTBO:

```
fdtoverlays /overlays/k3-am625-beagleplay-lt-lcd185.dtbo
```

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

```
label Linux eMMC
   kernel /Image
   append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.ifnames=0␣
↪systemd.unified_cgroup_hierarchy=false quiet
   fdtdir /
   fdtoverlays /overlays/k3-am625-beagleplay-lt-lcd185.dtbo
   initrd /initrd.img
```

Chapter 5

Demos and tutorials

5.1 Using Serial Console

To see the board boot log and access your BeaglePlay's console you can connect a USB-UART cable as depicted in the image below and use applications like `tio` to access the console.

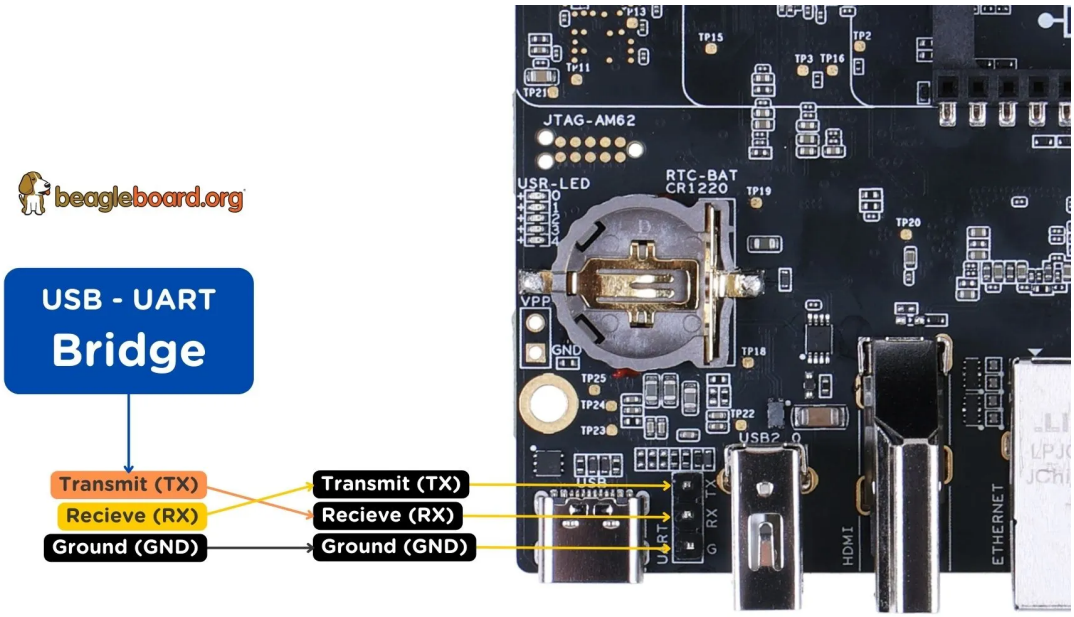


Fig. 5.1: Serial debug (USB-UART) cable connection.

If you are using Linux your USB to UART converter may appear as `/dev/ttyUSB`. It will be different for Mac and Windows operating systems.

```
[lorforlinux@fedora ~] $ tio /dev/ttyUSB0
tio v2.5
Press ctrl-t q to quit
Connected
```

Tip: For more information on `USB to UART` cables, you can checkout serial-debug-cables section.

5.2 Connect WiFi

Note: A common issue experienced by users when connecting to Wireless networks are network names that include special characters such as spaces, apostrophes etc, this may make connecting to your network more difficult. It is thus recommended to rename your Wireless AP to something simpler. For Example - renaming "Boris's Wireless Network" to "BorisNet". This avoids having to add special "escape" characters in the name. This shows up especially if you try connecting to iPhone/iOS HotSpots, where the network name is the device name, which by default is something like "Dan's iPhone". Also see this potential solution..

If you have a monitor and keyboard/mouse combo connected, the easiest way is to use *wpa_gui*.

Alternatively, you can use *wpa_cli instructions* over a shell connection through:

- The *serial console*,
- VSCode or `ssh` over a USB network connection,
- VSCode or `ssh` over an Ethernet connection,
- VSCode or `ssh` over *BeaglePlay WiFi access point*, or
- *A local Terminal Emulator session*.

Once you have a shell connection, follow the *wpa_cli* instructions.

5.2.1 BeaglePlay WiFi Access Point

Running the default image, your BeaglePlay should be hosting a WiFi access point with the SSID "BeaglePlay-XXXX", where *XXXX* is selected based on a hardware identifier on your board to try to increase the chances it will be unique.

Tip: The "XXXX" will be a combination of numbers and the letters A through F.

Note: At some point, we plan to introduce a captive portal design that will enable using your smartphone to provide BeaglePlay local WiFi login information. For now, you'll need to use a computer and

Step 1. Connect to BeaglePlay-XXXX

Tip: The password is either "BeaglePlay" or "BeagleBone" and the IP address will be 192.168.8.1.

Whatever your computer provides as a mechanism for searching for WiFi access points and connecting to them, just use that. You will want to have DHCP enabled, but that is the typical default. Connect to the "BeaglePlay-XXXX" access point and use the password "BeaglePlay" or "BeagleBone".

Note: The configuration for the access point is in the file system at `/etc/hostapd/hostapd.conf`.

Once your are connected to the access point, BeaglePlay should provide your computer an IP address and use 192.168.8.1 for itself. It should also be broadcasting the mDNS name "beagleplay.local".

Step 2. Browse to 192.168.8.1

Once you have connected to the access point, you can simply open VSCode by browsing to https://192.168.8.1:3000.

Within VSCode, you can press “CTRL-‘” to open a terminal session to get access to a shell connection.

You could also choose to *ssh* into your board via *ssh debian@192.168.8.1* and use the password *temppwd*.

Important: Once logged in, you should change the default password using the *passwd* command.

5.2.2 wpa_gui

Simplest way to connect to WiFi is to use `wpa_gui` tool pre-installed on your BeaglePlay. Follow simple steps below to connect to any WiFi access point.

Step 1: Starting wpa_gui

You can start `wpa_gui` either from `Applications > Internet > wpa_gui` or double click on the `wpa_gui` desktop application shortcut.

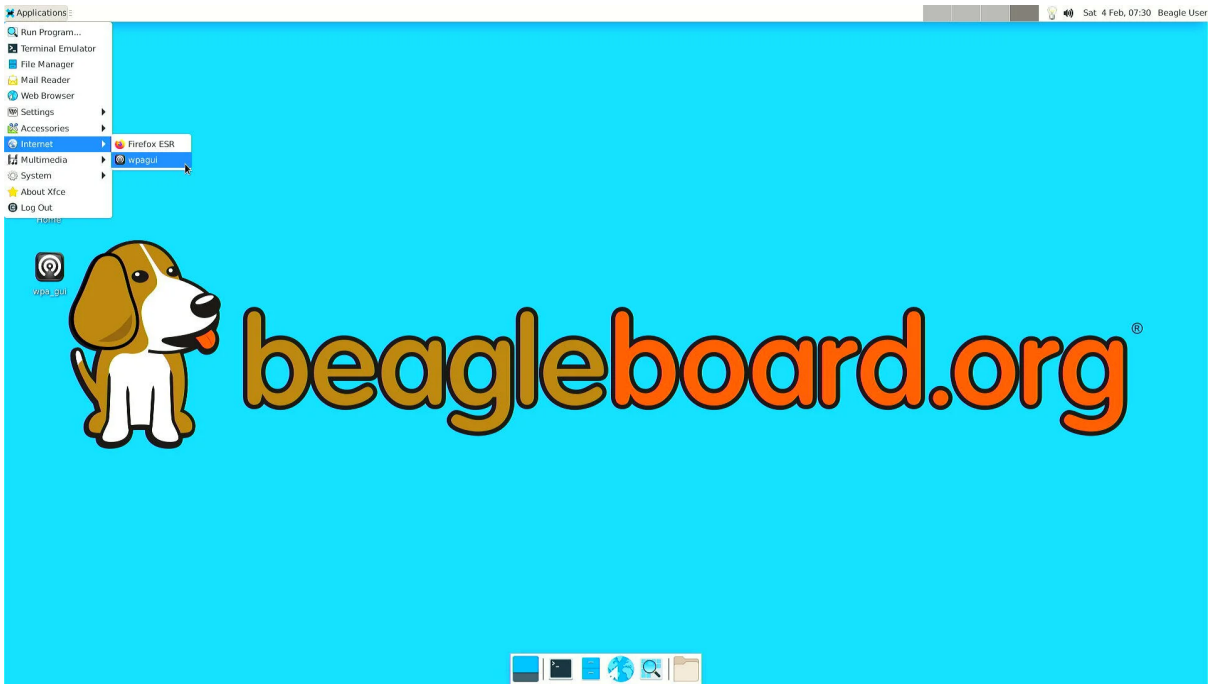


Fig. 5.2: Starting wpa_gui from Applications > Internet > wpa_gui

Step 2: Understanding wpa_gui interface

Let’s see the `wpa_gui` interface in detail,

1. `Adapter` is the WiFi interface device, it should be `wlan0` (on-board WiFi) by default.
2. `Network` shows the WiFi access point `SSID` if you are connected to that network.
3. **`Current Status` tab shows you network information if you are connected to any network.**

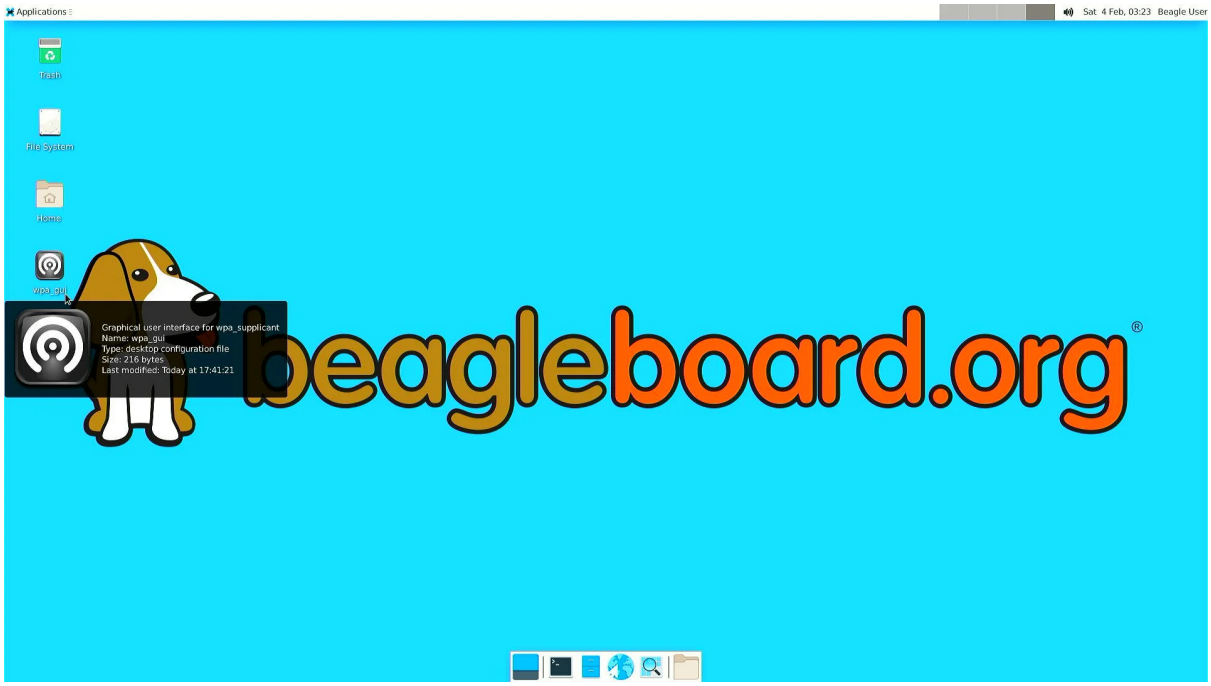


Fig. 5.3: Starting wpa_gui from Desktop application shortcut

- Click on `Connect` to connect if not automatically done.
- Click on `Disconnect` to disconnect/reset the connection.
- Click on `Scan` to scan nearby WiFi access points.

4. `Manage Network` tab shows you all the saved networks and options to manage those.

Step 3: Scanning & Connecting to WiFi access points

To scan the WiFi access points around you, just click on `Scan` button available under `wpa_gui > Current Status > Scan`.

A new window will open up with,

1. SSID (WiFi name)
2. BSSID
3. Frequency
4. Signal strength
5. flags

Now, you just have to double click on the Network you want to connect to as shown below.

Note: SSIDs and BSSIDs are not fully visible in screenshot below but you can change the column length to see the WiFi names better.

Final step is to type your WiFi access point password under `PSK` input field and click on `Add` (as shown in screenshot below) which will automatically connect your board to WiFi (if password is correct).

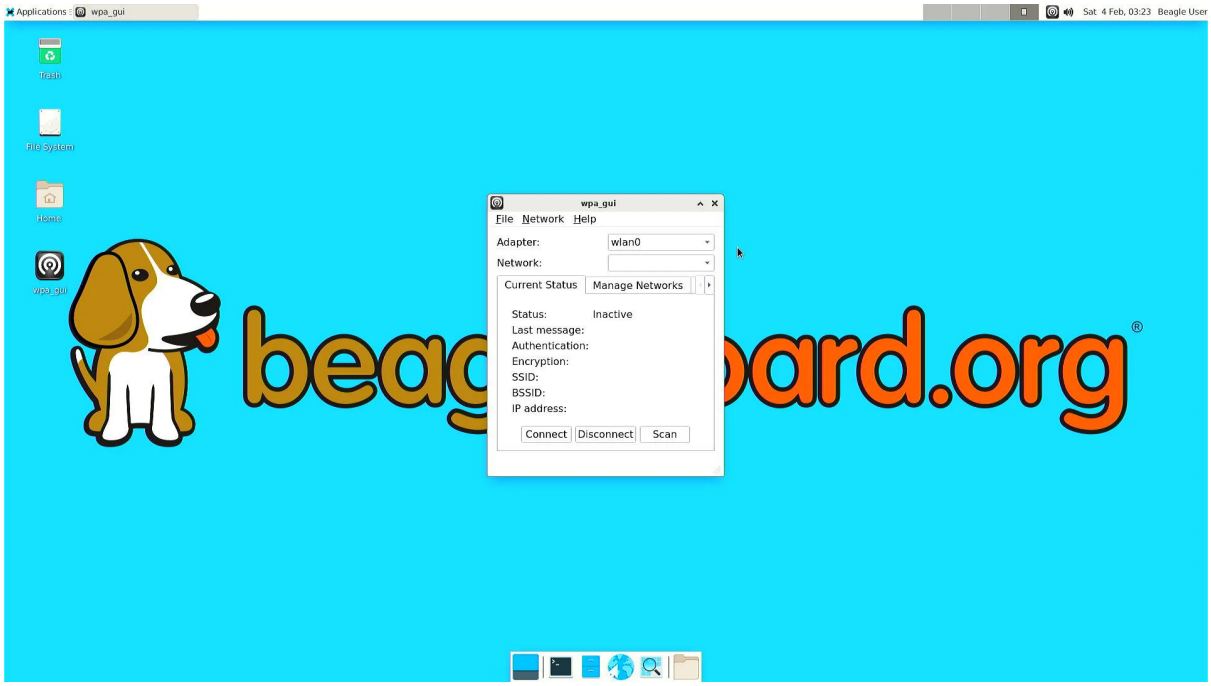


Fig. 5.4: wpa_gui interface

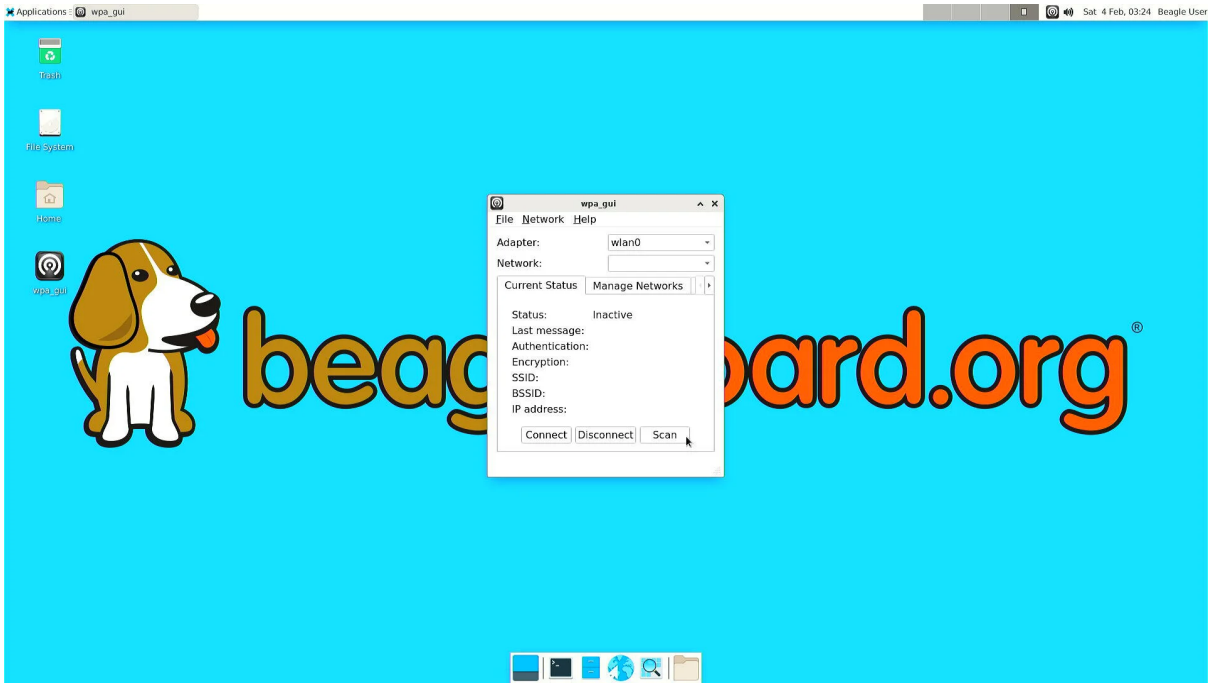


Fig. 5.5: Scanning WiFi access points

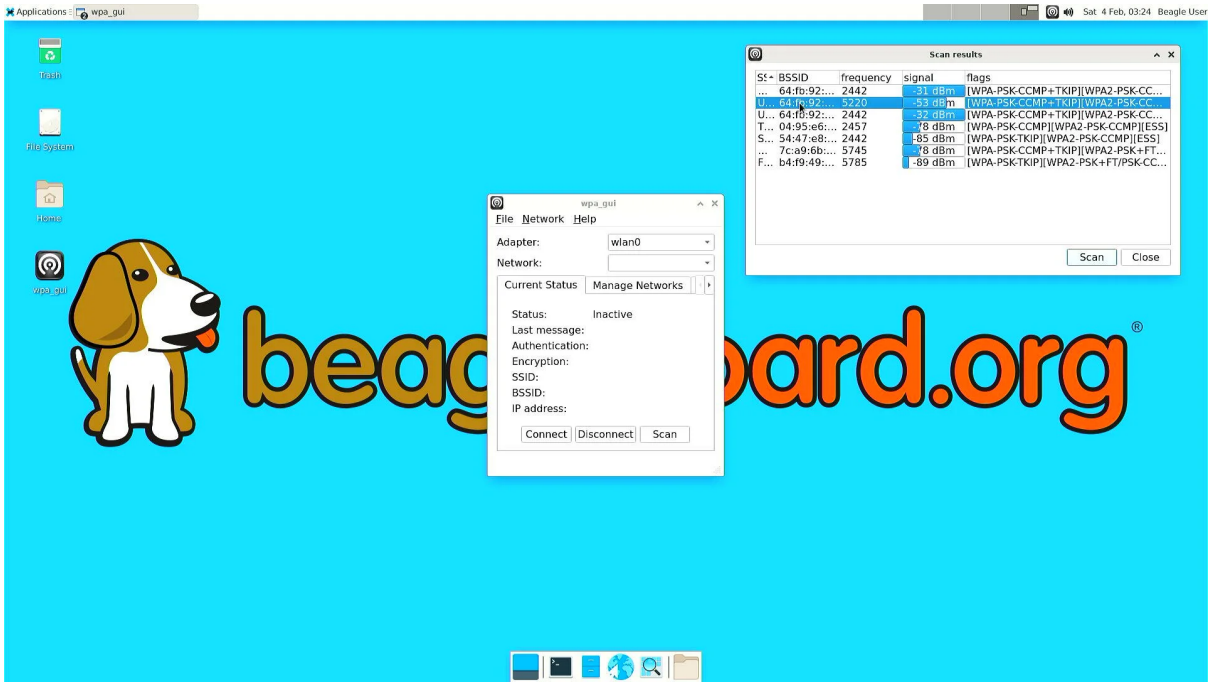


Fig. 5.6: Selecting WiFi access point

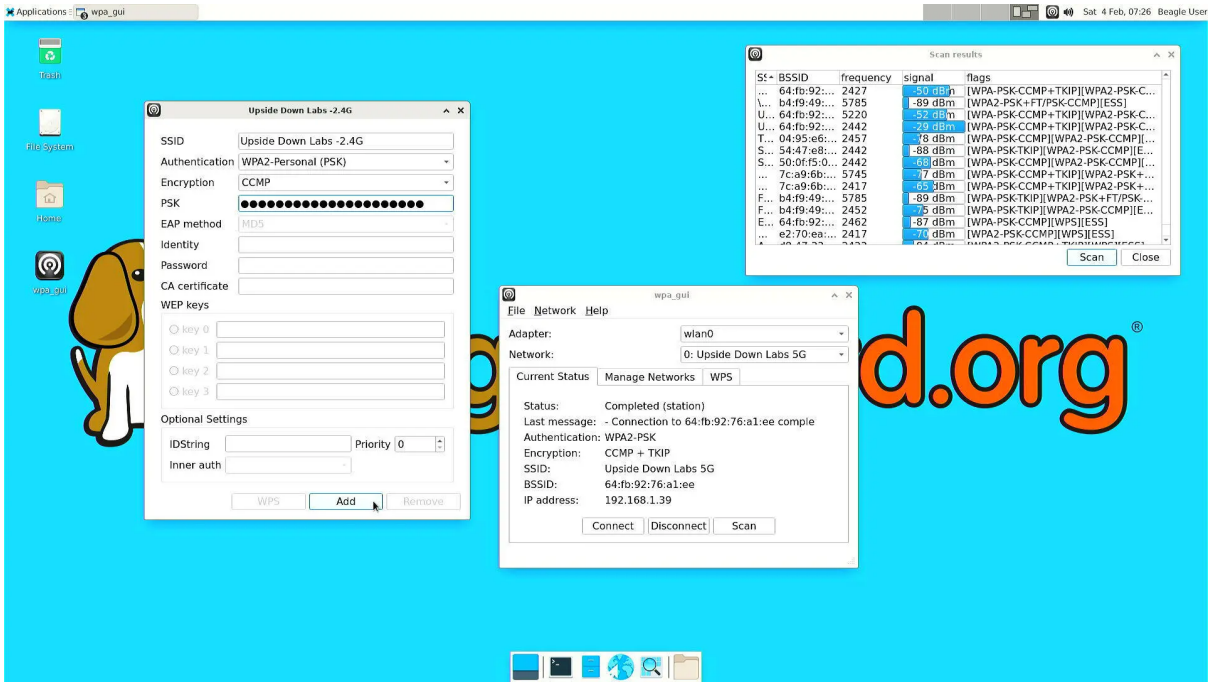


Fig. 5.7: Connecting to WiFi access point

5.2.3 wpa_cli (shell)

In commands shown below, swap out “68:ff:7b:03:0a:8a” and “mypassword” with your network BSSID and password, respectively.

```
debian@BeaglePlay:~$ wpa_cli scan
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli scan_results
Selected interface 'wlan0'
bssid / frequency / signal level / flags / ssid
68:ff:7b:03:0a:8a    5805    -49     [WPA2-PSK-CCMP][WPS][ESS]       mywifi
debian@BeaglePlay:~$ wpa_cli add_network
Selected interface 'wlan0'
1
debian@BeaglePlay:~$ wpa_cli set_network 1 bssid 68:ff:7b:03:0a:8a
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli set_network 1 psk '"mypassword"'
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ wpa_cli enable_network 1
Selected interface 'wlan0'
OK
debian@BeaglePlay:~$ ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.245  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::6e30:2aff:fe29:757d  prefixlen 64  scopeid 0x20<link>
        inet6 2601:408:c083:b6c0::e074  prefixlen 128  scopeid 0x0<global>
        ether 6c:30:2a:29:75:7d  txqueuelen 1000  (Ethernet)
        RX packets 985  bytes 144667 (141.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 10826 (10.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Important: The single quotes around the double quotes are needed to make sure the double quotes are given to *wpa_cli instructions*. It expects to see them.

Note: For more information about *wpa_cli instructions*, see https://w1.fi/wpa_supplicant/

To make these changes persistent, you need to edit */etc/wpa_supplicant/wpa_supplicant-wlan0.conf*. This is described in *wpa_cli (XFCE)* section.

5.2.4 wpa_cli (XFCE)

Another way of connecting to a WiFi access point is to edit the `wpa_supplicant` configuration file.

Step 1: Open up terminal

Open up a terminal window either from `Applications > Terminal Emulator` Or from Task Manager.

Step 2: Setup credentials

To setup credentials of your WiFi access point follow these steps,

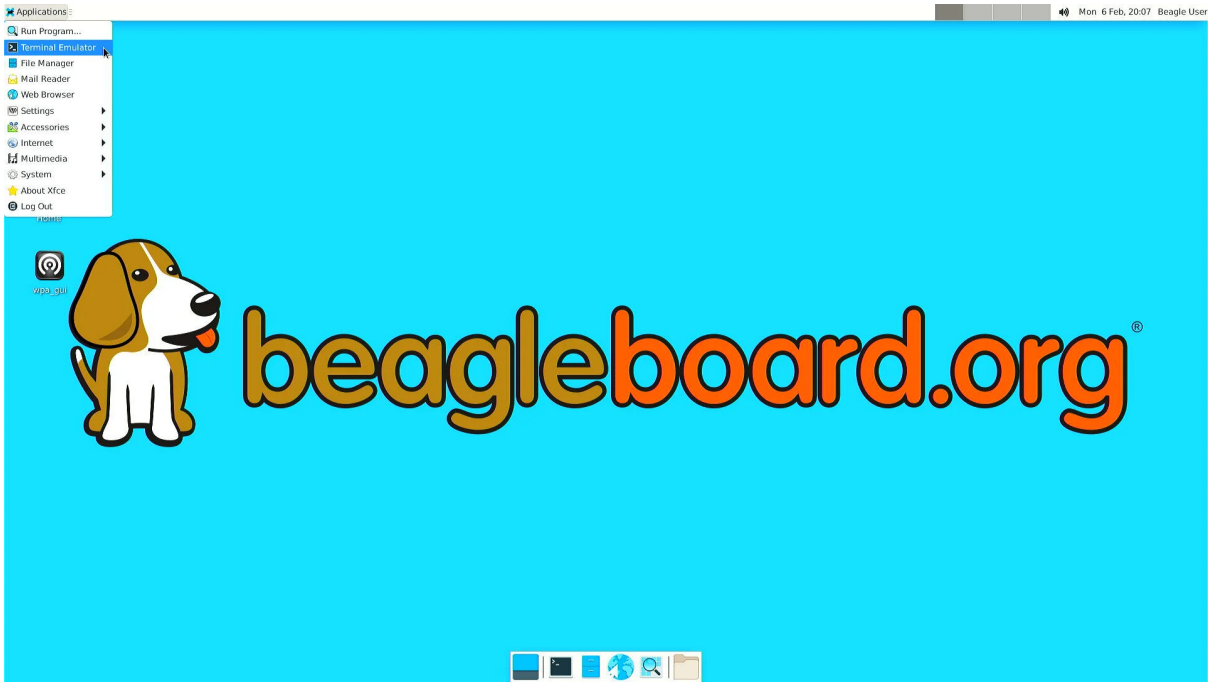


Fig. 5.8: Open terminal from Applications > Terminal Emulator

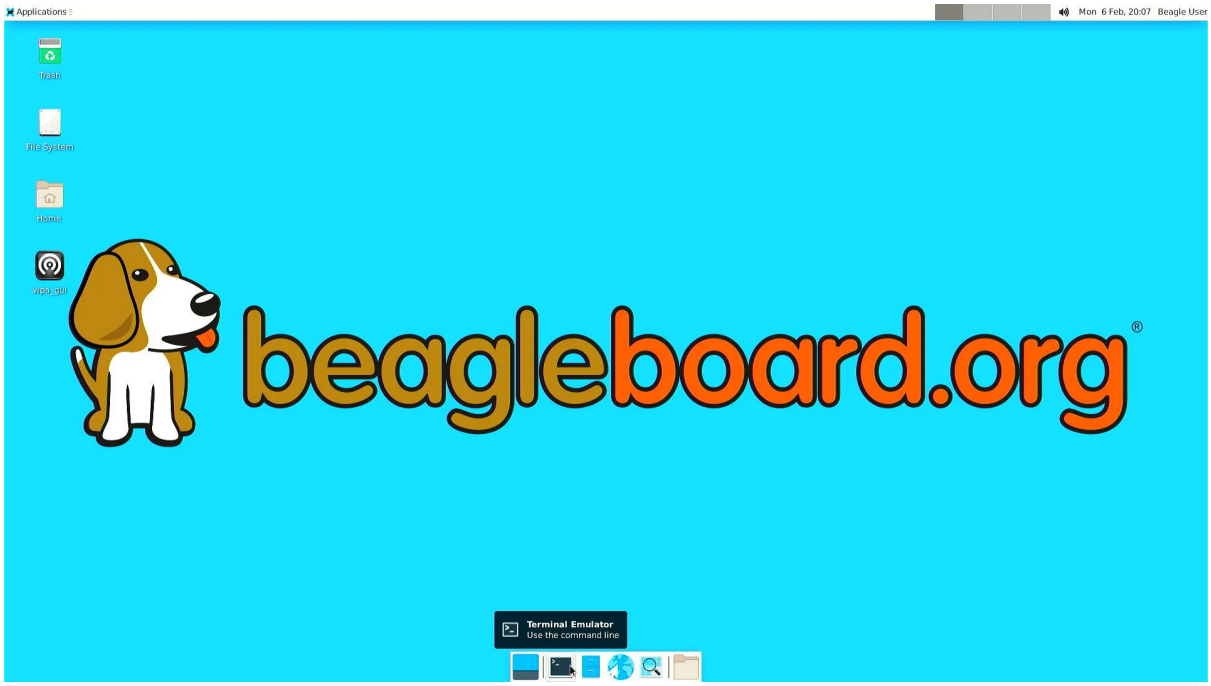


Fig. 5.9: Open terminal from Task Manager

1. Execute `sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf`, which will open up `wpa_supplicant-wlan0.conf` inside `nano` (terminal based) text editor. 1. Edit `wpa_supplicant-wlan0.conf` to add SSID (WiFi name) & PSK (WiFi password) of your WiFi access point.

```
....
network={
        ssid="WiFi Name"
        psk="WiFi Password"
        ....
}
```

1. Now save the details using `ctrl + O` then enter.
2. To exit out of the `nano` text editor use `ctrl + X`.

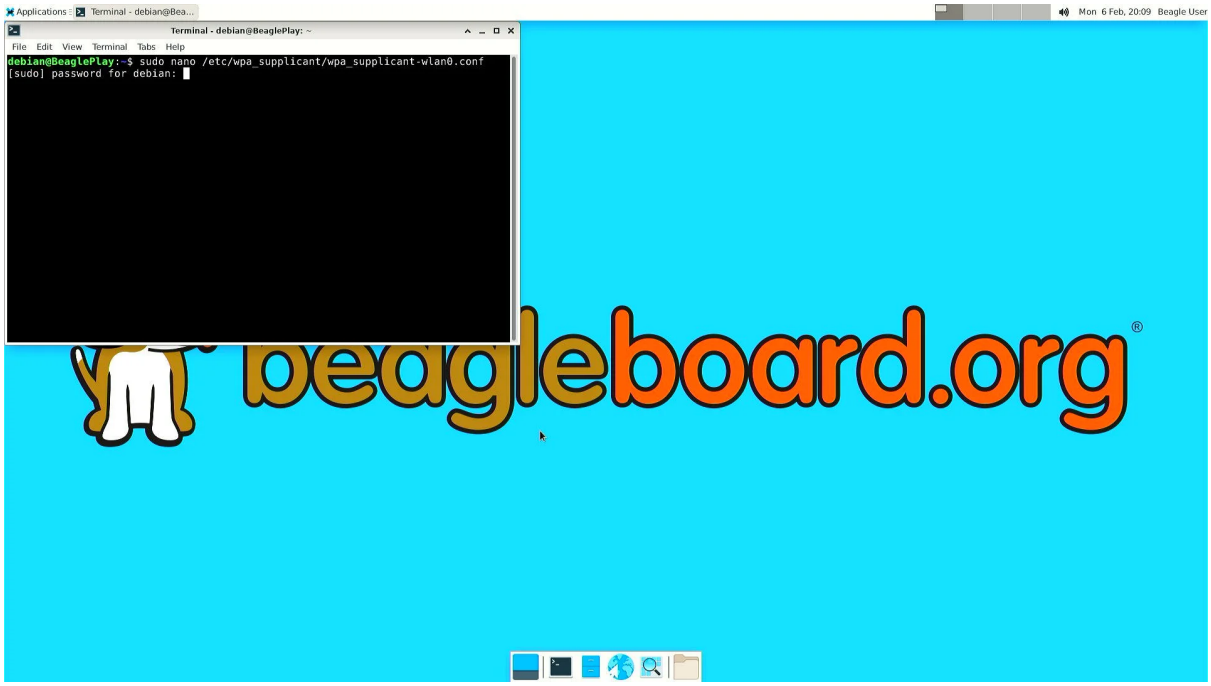


Fig. 5.10: Run: $ sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf

Step 3: Reconfigure wlan0

The WiFi doesn't automatically connect to your WiFi access point after you add the credentials to `wpa_supplicant-wlan0.conf`.

1. To connect you can either execute `sudo wpa_cli -i wlan0 reconfigure`
2. Or Reboot your device by executing `reboot` inside your terminal window.
3. Execute `ping 8.8.8.8` to check your connection. Use `ctrl + C` to quit.

```
debian@BeaglePlay:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=5.83 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=7.27 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=5.30 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=5.28 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=9.04 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=118 time=7.52 ms
```

(continues on next page)

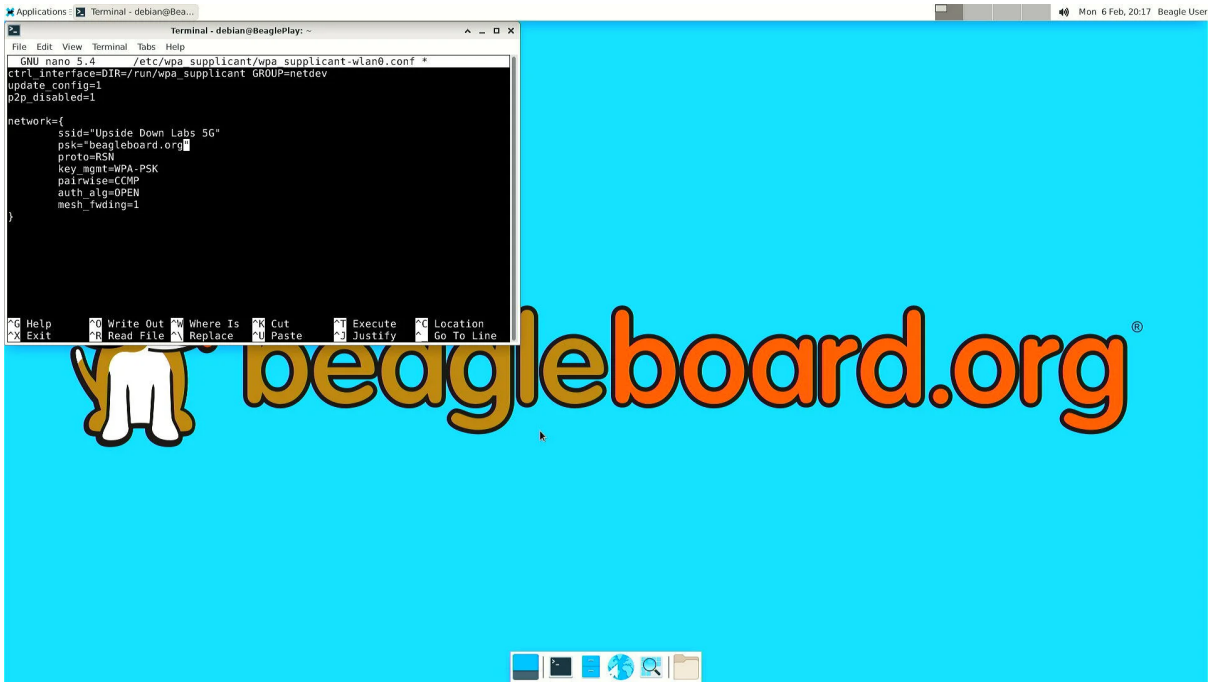


Fig. 5.11: Add SSID and PSK

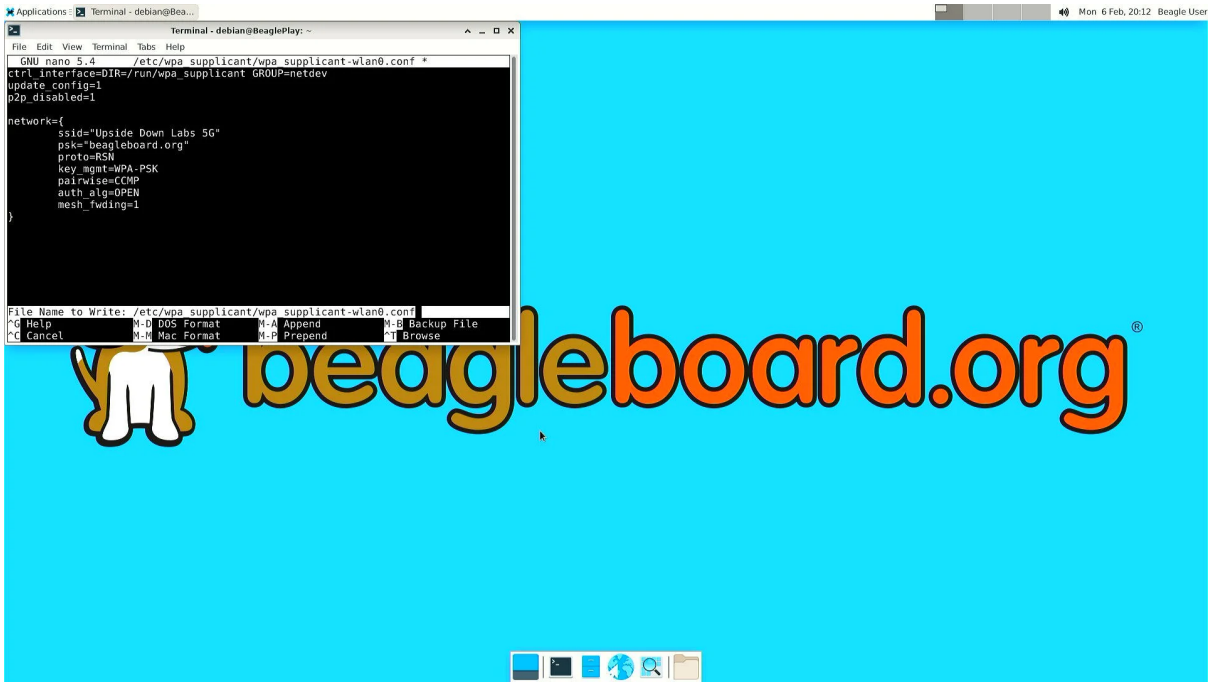


Fig. 5.12: Save credentials (ctrl + O) and Exit (ctrl + X)

(continued from previous page)

```
64 bytes from 8.8.8.8: icmp_seq=7 ttl=118 time=5.39 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=118 time=5.94 ms
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7008ms
rtt min/avg/max/mdev = 5.281/6.445/9.043/1.274 ms
```

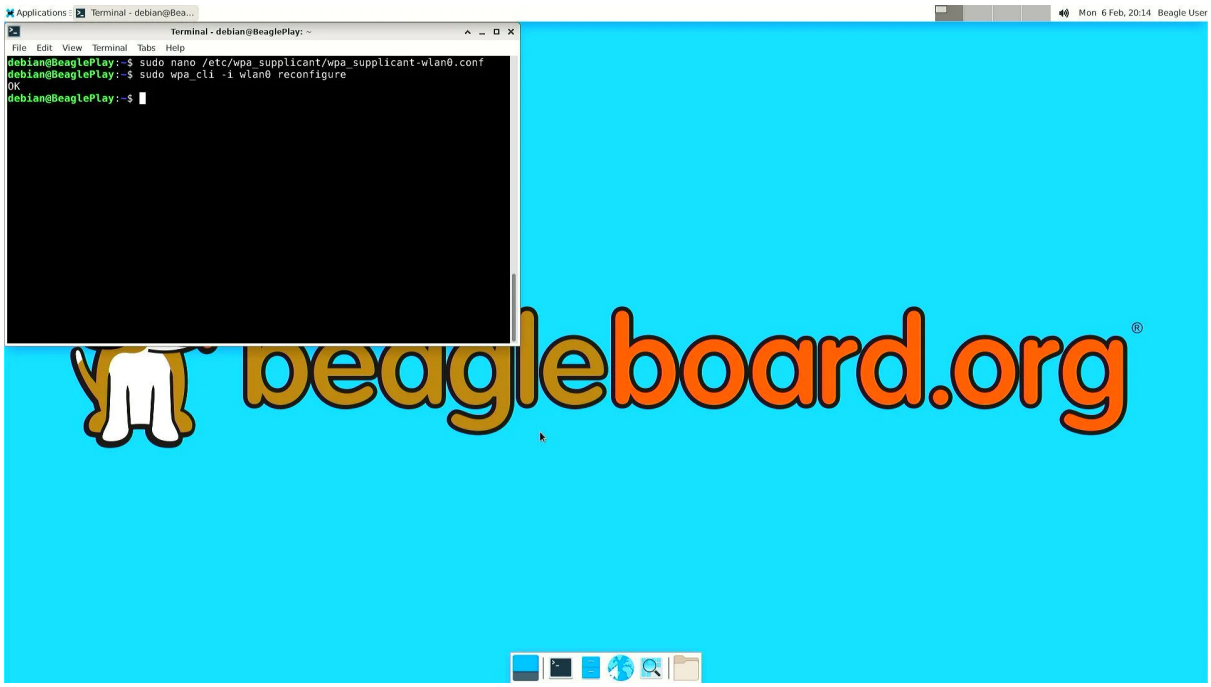


Fig. 5.13: Connect to WiFi by running $ sudo wpa_cli -i wlan0 reconfigure

5.2.5 Disabling the WIFI Access Point

In certain situations, such as running HomeAssistant, you may chose to connect your BeaglePlay to the internet via Ethernet. In this case, it may be desirable to disable it's Wifi access point so that users outside the local network aren't able to connect to it.

The Wifi Access Point that BeaglePlay provides is started using uDev rules. created by the *bb-wlan0-defaults* package

You can simply remove the *bb-wlan0-defaults* package:

```
sudo apt remove bb-wlan0-defaults
```

Now just reboot and the Wifi Access point should no longer start.

You can also disable it by removing the two following udev rule files:

```
rm /etc/udev/rules.d/81-add-SoftAp0-interface.rules
rm /etc/udev/rules.d/82-SoftAp0-start-hostpad.rules
```

The issue with doing this latter option is that if you later update your OS, the bb-wlan0-defaults may get updated as well and re-add the rules.

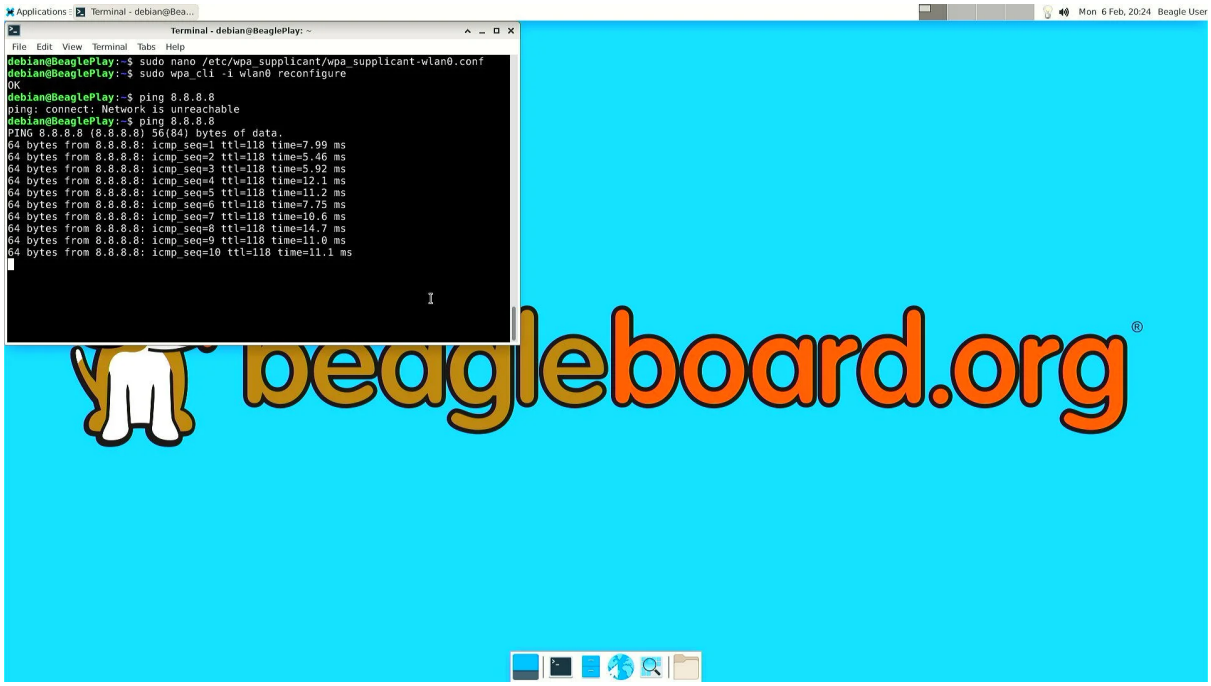


Fig. 5.14: To check connection try running $ ping 8.8.8.8

5.2.6 Re-Enabling the WIFI Access Point

Conversely, you can re-enable the access point by re-installing the *bb-wlan0-default* package.

```
sudo apt install bb-wlan0-defaults --reinstall
```

Now just reboot.

Todo: Add notes on changing SSID/Password

5.3 Using Grove

See qwiic_stemma_grove_addons.

A link to the appropriate I2C controller can be found at `/dev/play/grove/i2c`.

5.4 Using mikroBUS

Steps:

1. Identify if mikroBUS add-on includes a ClickID with `manifest`. If not, `manifest` must be supplied.
2. Identify if mikroBUS add-on is supported by the kernel. If not, kernel module must be added.
3. Identify how driver exposes the data: IIO, net, etc.
4. Connect and power
5. Verify and utilize

5.4.1 Using boards with ClickID

What is mikroBUS?

mikroBUS is an open standard for add-on boards for sensors, connectivity, displays, storage and more with over 1,400 available from just a single source, MikroE. With the flexibility of all of the most common embedded serial busses, UART, I2C and SPI, along with ADC, PWM and GPIO functions, it is a great solution for connecting all sorts of electronics.

Note: Learn more at https://www.mikroe.com/mikrobus

What is ClickID?

ClickID enables mikroBUS add-on boards to be identified along with the configuration required to use it with the mikroBUS Linux driver. The configuration portion is called a `manifest`.

Note: Learn more at https://github.com/MikroElektronika/click_id

BeaglePlay's Linux kernel is patched with a mikrobus driver that automatically reads the ClickID and loads a driver, greatly simplifying usage.

Does my add-on have ClickID?

Look for the board's `'D` (ID) logo. It's near the PWM pin on the upper right-hand side in the illustration below.

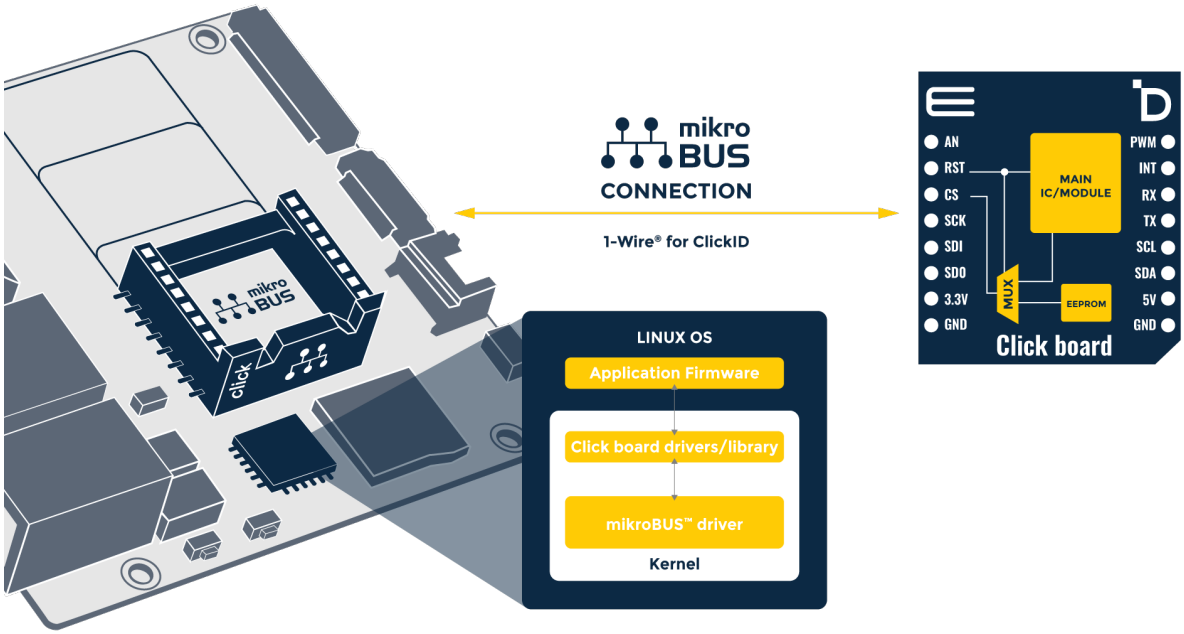


Fig. 5.15: mikroBUS clickID - BeaglePlay connection

If your add-on has ClickID, simply connect it while BeaglePlay is powered off and then apply power.

Example of examining boot log to see a ClickID was detected.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.096254] mikrobus:mikrobus_port_register: registering port mikrobus-0
[    2.096325] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
[    2.663698] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 1, driver=opt3001, protocol=3, reg=44
[    2.663711] mikrobus_manifest:mikrobus_manifest_parse:  Ambient 2 Click␣
↪manifest parsed with 1 devices
[    2.663783] mikrobus mikrobus-0: registering device : opt3001
```

Note: Not all Click boards with ClickID have valid `manifest` entries. Then you can follow *What if my add-on has invalid manifest entries?* to make your add-on detected.

To use the add-on, see *Accel Click Board Example*.

What if my add-on doesn't have ClickID?

If add-on doesn't have clickID then it can not be detected directly.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.123994] mikrobus:mikrobus_port_register: registering port mikrobus-0
[    2.124059] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
```

Available `manifest` can be installed that has been created for your add-on as we have created over 100 of them. You can install the existing manifest files onto your BeaglePlay. First, make sure you have the latest manifests installed in your system.

```
sudo apt update
sudo apt install bbb.io-clickid-manifests
```

Take a look at the list of `manifest` files to see if the Click or other mikrobus add-on board `manifest` is installed.

```
debian@BeaglePlay:~$ ls /lib/firmware/mikrobus/
10DOF-CLICK.mnfb          COMPASS-2-CLICK.mnfb       I2C-2-SPI-CLICK.mnfb   ␣
↪    PWM-CLICK.mnfb
13DOF-2-CLICK.mnfb        COMPASS-CLICK.mnfb         I2C-MUX-CLICK.mnfb     ␣
↪    RFID-CLICK.mnfb
3D-HALL-3-CLICK.mnfb      CURRENT-CLICK.mnfb         ILLUMINANCE-CLICK.mnfb ␣
↪    RF-METER-CLICK.mnfb
3D-HALL-6-CLICK.mnfb      DAC-7-CLICK.mnfb           IR-GESTURE-CLICK.mnfb  ␣
↪    RMS-TO-DC-CLICK.mnfb
6DOF-IMU-2-CLICK.mnfb     DAC-CLICK.mnfb             IR-THERMO-2-CLICK.mnfb ␣
↪    RTC-6-CLICK.mnfb
6DOF-IMU-4-CLICK.mnfb     DIGIPOT-3-CLICK.mnfb       LED-DRIVER-7-CLICK.mnfb␣
↪    SHT1x-CLICK.mnfb
6DOF-IMU-6-CLICK.mnfb     DIGIPOT-CLICK.mnfb         LIGHTRANGER-2-CLICK.
↪mnfb    SHT-CLICK.mnfb
6DOF-IMU-8-CLICK.mnfb     EEPROM-2-CLICK.mnfb        LIGHTRANGER-3-CLICK.
↪mnfb    SMOKE-CLICK.mnfb
9DOF-CLICK.mnfb           EEPROM-3-CLICK.mnfb        LIGHTRANGER-CLICK.mnfb ␣
↪    TEMP-HUM-11-CLICK.mnfb
ACCEL-3-CLICK.mnfb        EEPROM-CLICK.mnfb          LPS22HB-CLICK.mnfb     ␣
↪    TEMP-HUM-12-CLICK.mnfb
ACCEL-5-CLICK.mnfb        ENVIRONMENT-CLICK.mnfb     LSM303AGR-CLICK.mnfb   ␣
↪    TEMP-HUM-3-CLICK.mnfb
ACCEL-6-CLICK.mnfb        ETH-CLICK.mnfb             LSM6DSL-CLICK.mnfb     ␣
↪    TEMP-HUM-4-CLICK.mnfb
```

(continues on next page)

(continued from previous page)

```
ACCEL-8-CLICK.mnfb         ETH-WIZ-CLICK.mnfb          MAGNETIC-LINEAR-CLICK.
↪mnfb  TEMP-HUM-7-CLICK.mnfb
ACCEL-CLICK.mnfb           FLASH-2-CLICK.mnfb          MAGNETIC-ROTARY-CLICK.
↪mnfb  TEMP-HUM-9-CLICK.mnfb
ADC-2-CLICK.mnfb           FLASH-CLICK.mnfb            MICROSD-CLICK.mnfb      ␣
↪    TEMP-HUM-CLICK.mnfb
ADC-3-CLICK.mnfb           GENERIC-SPI-CLICK.mnfb      MPU-9DOF-CLICK.mnfb     ␣
↪    TEMP-LOG-3-CLICK.mnfb
ADC-5-CLICK.mnfb           GEOMAGNETIC-CLICK.mnfb      MPU-IMU-CLICK.mnfb      ␣
↪    TEMP-LOG-4-CLICK.mnfb
ADC-8-CLICK.mnfb           GNSS-4-CLICK.mnfb           NO2-2-CLICK.mnfb        ␣
↪    TEMP-LOG-6-CLICK.mnfb
ADC-CLICK.mnfb             GNSS-7-CLICK.mnfb           NO2-CLICK.mnfb          ␣
↪    THERMO-12-CLICK.mnfb
AIR-QUALITY-2-CLICK.mnfb   GNSS-ZOE-CLICK.mnfb         OLEDB-CLICK.mnfb        ␣
↪    THERMO-15-CLICK.mnfb
AIR-QUALITY-3-CLICK.mnfb   GSR-CLICK.mnfb              OLEDC-CLICK.mnfb        ␣
↪    THERMO-17-CLICK.mnfb
AIR-QUALITY-5-CLICK.mnfb   GYRO-2-CLICK.mnfb           OLEDW-CLICK.mnfb        ␣
↪    THERMO-3-CLICK.mnfb
ALCOHOL-2-CLICK.mnfb       GYRO-CLICK.mnfb             OZONE-2-CLICK.mnfb      ␣
↪    THERMO-4-CLICK.mnfb
ALCOHOL-3-CLICK.mnfb       HALL-CURRENT-2-CLICK.mnfb   PRESSURE-11-CLICK.mnfb ␣
↪    THERMO-7-CLICK.mnfb
ALTITUDE-3-CLICK.mnfb      HALL-CURRENT-3-CLICK.mnfb   PRESSURE-3-CLICK.mnfb   ␣
↪    THERMO-8-CLICK.mnfb
ALTITUDE-CLICK.mnfb        HALL-CURRENT-4-CLICK.mnfb   PRESSURE-4-CLICK.mnfb   ␣
↪    THERMO-CLICK.mnfb
AMBIENT-2-CLICK.mnfb       HDC1000-CLICK.mnfb          PRESSURE-CLICK.mnfb     ␣
↪    THERMOSTAT-3-CLICK.mnfb
AMBIENT-4-CLICK.mnfb       HEART-RATE-3-CLICK.mnfb     PROXIMITY-10-CLICK.mnfb␣
↪    UV-3-CLICK.mnfb
AMBIENT-5-CLICK.mnfb       HEART-RATE-4-CLICK.mnfb     PROXIMITY-2-CLICK.mnfb  ␣
↪    VACUUM-CLICK.mnfb
AMMETER-CLICK.mnfb         HEART-RATE-5-CLICK.mnfb     PROXIMITY-5-CLICK.mnfb  ␣
↪    VOLTMETER-CLICK.mnfb
COLOR-2-CLICK.mnfb         HEART-RATE-7-CLICK.mnfb     PROXIMITY-9-CLICK.mnfb  ␣
↪    WAVEFORM-CLICK.mnfb
COLOR-7-CLICK.mnfb         HEART-RATE-CLICK.mnfb       PROXIMITY-CLICK.mnfb    ␣
↪    WEATHER-CLICK.mnfb
```

Below command to grant root privileges of the intended user and then enter passsword. This will take you to the different shell.

```
sudo su
```

Then, load the appropriate `manifest` using the `mikrobus` bus driver. For example, with the Ambient 2 Click, you can write that `manifest` to the `mikrobus-0` `new_device` entry.

```
cat /lib/firmware/mikrobus/AMBIENT-2-CLICK.mnfb > /sys/bus/mikrobus/devices/
↪mikrobus-0/new_device
```

You can now exit this shell.

```
exit
```

Once done, you can check it using command `dmesg | grep mikrobus` which shows that add-on is now detected.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.096254] mikrobus:mikrobus_port_register: registering port mikrobus-0
```

(continues on next page)

(continued from previous page)

```
[    2.096325] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
[    2.663698] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 1, driver=opt3001, protocol=3, reg=44
[    2.663711] mikrobus_manifest:mikrobus_manifest_parse:  Ambient 2 Click␣
↪manifest parsed with 1 devices
[    2.663783] mikrobus mikrobus-0: registering device : opt3001
```

Note: It'll forget on reboot... need to have a boot service.

Todo: To make it stick, ...

What if my add-on has invalid manifest entries?

Not all Click boards with ClickID have valid manifest entries. If your add-on has clickID but shows the command output like below.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.119771] mikrobus:mikrobus_port_register: registering port mikrobus-0
[    2.119842] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
[    2.261113] mikrobus_manifest:mikrobus_manifest_header_validate: manifest␣
↪version too new (150.189 > 0.3)
[    2.261130] mikrobus mikrobus-0: invalid manifest size -22
```

There are some available manifest that can be used to write in the eeprom of clickID board. Once you `sudo apt update` and `sudo apt install bbb.io-clickid-manifests` then you can see the list of manifests using command `ls /lib/firmware/mikrobus/`. Let's take the `Accel Click - ClickID` Board with invalid manifest entries, To get the valid manifest we need to write `ACCEL-CLICK.mnfb` to eeprom of ClickID board using the following commands.

First check the file name for the add-on device. It can be in the form of `w1_bus_master1-xx-xxxxxxx`.

```
debian@BeaglePlay:~$ ls /sys/bus/w1/devices/
w1_bus_master1  w1_bus_master1-xx-xxxxxxx
```

Then in the following command, `/lib/firmware/mikrobus/ACCEL-CLICK.mnfb` is the path of manifest file and `/sys/bus/w1/devices/w1_bus_master1-xx-xxxxxxx/mikrobus_manifest` is path for one wire eeprom clickID board. You must replace the the file name `w1_bus_master1-xx-xxxxxxx` with your clickID board file in the below command.

```
debian@BeaglePlay:~$ sudo dd if=/lib/firmware/mikrobus/ACCEL-CLICK.mnfb of=/
↪sys/bus/w1/devices/w1_bus_master1-xx-xxxxxxx/mikrobus_manifest
0+1 records in
0+1 records out
132 bytes copied, 0.0144496 s, 9.1 kB/s
```

Now, Reboot your BeaglePlay. After rebooting, the add-on has been detected with valid manifest entries.

```
debian@BeaglePlay:~$ dmesg | grep mikrobus
[    2.126654] mikrobus:mikrobus_port_register: registering port mikrobus-0
[    2.126727] mikrobus mikrobus-0: mikrobus port 0 eeprom empty probing␣
↪default eeprom
[    2.797179] mikrobus_manifest:mikrobus_manifest_attach_device: parsed␣
↪device 1, driver=adxl345, protocol=3, reg=1d
[    2.797191] mikrobus_manifest:mikrobus_manifest_parse:  Accel Click␣
```

(continues on next page)

(continued from previous page)

```
↪manifest parsed with 1 devices
[    2.797267] mikrobus mikrobus-0: registering device : adxl345
```

Note: The updation has done in the eeprom of clickID board. It will not forget after reboot.

Note: We will be adding a link to the `mikrobus-0` device at `/dev/play/mikrobus` in the near future, but you can find it for now at `/sys/bus/mikrobus/devices/mikrobus-0`. If you need to supply an ID (manifest), this is the directory where you will do it.

Manifesto: https://git.beagleboard.org/beagleconnect/manifesto

Patched Linux with out-of-tree Mikrobus driver: https://git.beagleboard.org/beagleboard/linux

To use the add-on, see *Accel Click Board Example*.

Accel Click Board Example

Next, let's explore how to read raw sensor values using the Accel Click board. This step will help us understand the basics of sensor data retrieval and processing.

First, let's check the IIO devices available.

```
debian@BeaglePlay:~$ ls /sys/bus/iio/devices/
iio:device0  iio:device1
```

Considering the device `iio:device0` is the MikroBUS click ID connected to the BeaglePlay board. Depending on your specific setup and device configuration, you might need to adjust the path or device number (device0) accordingly. In this case device0 corresponds to our Accel Click, let's check its name.

```
debian@BeaglePlay:~$ cat /sys/bus/iio/devices/iio\:device0/name
adxl345
```

The file corresponding to the IIO device, including raw values, can be viewed using the following command:

```
debian@BeaglePlay:~$ ls /sys/bus/iio/devices/iio\:device0
dev                          in_accel_scale        in_accel_x_raw        in_
↪accel_y_raw        in_accel_z_raw  power                          subsystem
in_accel_sampling_frequency  in_accel_x_calibbias  in_accel_y_calibbias  in_
↪accel_z_calibbias  name            sampling_frequency_available  uevent
```

To view the raw values from the accel click (assuming `iio:device0` is configured correctly for your MikroBUS click ID on the BeaglePlay board), you can use the following command:

```
debian@BeaglePlay:~$ cat /sys/bus/iio/devices/iio\:device0/in_accel_x_raw
3
```

This command reads and displays the raw X-axis accelerometer data from `iio:device0`. You can replace `in_accel_x_raw` with `in_accel_y_raw` or `in_accel_z_raw` to view raw data from the Y-axis or Z-axis accelerometer channels respectively, depending on your requirements.

To create a script displays accelerometer raw data values from `iio:device0` use `nano accelclick.sh` command. Copy the below script and paste it to the `accelclick.sh` file. It reads the raw X, Y, and Z axis values from `/sys/bus/iio/devices/iio:device0/in_accel_x_raw`, `/sys/bus/iio/devices/iio:device0/in_accel_y_raw`, and `/sys/bus/iio/devices/iio:device0/in_accel_z_raw` respectively.

```
X=$(cat /sys/bus/iio/devices/iio\:device0/in_accel_x_raw)
Y=$(cat /sys/bus/iio/devices/iio\:device0/in_accel_y_raw)
Z=$(cat /sys/bus/iio/devices/iio\:device0/in_accel_z_raw)
echo "X = ${X}        Y = ${Y}      Z= ${Z}"
```

Note: Adjust the device path `iio:device0` according to your setup. Also, ensure that your system and hardware configuration are correctly set up to provide live accelerometer data through these paths.

To make the script file executable, use the following command:

```
debian@BeaglePlay:~$ chmod +x accelclick.sh
```

When you run `watch -n 0.5 ./accelclick.sh`, the watch command will execute `./accelclick.sh` every 0.5 seconds and display its output in the terminal.

```
debian@BeaglePlay:~$ watch -n 0.5 ./accelclick.sh
```

This is the output of your accelclick.sh script. It shows the current values of your accelerometer's X, Y, and Z axis in raw form.

```
Every 0.5s: ./accelclick.sh

X = 3        Y = 11      Z= 284
```

5.4.2 Using boards with Linux drivers

Depending on the type of mikrobus add-on board, the Linux driver could be of various different types. For sensors, the most common is *IIO driver*.

IIO driver

Per https://docs.kernel.org/driver-api/iio/intro.html,

> The main purpose of the Industrial I/O subsystem (IIO) is to provide support for devices that in some sense perform either analog-to-digital conversion (ADC) or digital-to-analog conversion (DAC) or both. The aim is to fill the gap between the somewhat similar hwmon and input subsystems. Hwmon is directed at low sample rate sensors used to monitor and control the system itself, like fan speed control or temperature measurement. Input is, as its name suggests, focused on human interaction input devices (keyboard, mouse, touchscreen). In some cases there is considerable overlap between these and IIO.
>
> Devices that fall into this category include:
>
> - analog to digital converters (ADCs)
> - accelerometers
> - capacitance to digital converters (CDCs)
> - digital to analog converters (DACs)
> - gyroscopes
> - inertial measurement units (IMUs)
> - color and light sensors
> - magnetometers
> - pressure sensors

- proximity sensors
- temperature sensors

See also https://wiki.analog.com/software/linux/docs/iio/iio.

To discover IIO driver enabled devices, use the `iio_info` command.

```
debian@BeaglePlay:~$ iio_info
Library version: 0.24 (git tag: v0.24)
Compiled with backends: local xml ip usb
IIO context created with local backend.
Backend version: 0.24 (git tag: v0.24)
Backend description string: Linux BeaglePlay 5.10.168-ti-arm64-r104
↪#1bullseye SMP Thu Jun 8 23:07:22 UTC 2023 aarch64
IIO context has 2 attributes:
        local,kernel: 5.10.168-ti-arm64-r104
        uri: local:
IIO context has 2 devices:
        iio:device0: opt3001
                1 channels found:
                    illuminance:  (input)
                        2 channel-specific attributes found:
                                attr  0: input value: 163.680000
                                attr  1: integration_time value: 0.800000
                2 device-specific attributes found:
                                attr  0: current_timestamp_clock value:␣
↪realtime
                                attr  1: integration_time_available value: 0.
↪1 0.8
                No trigger on this device
        iio:device1: adc102s051
                2 channels found:
                        voltage1:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 4084
                                attr  1: scale value: 0.805664062
                        voltage0:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 2440
                                attr  1: scale value: 0.805664062
                No trigger on this device
```

Note that the units are standardized for the IIO interface based on the device type. If raw values are provided, a scale must be applied to get to the standardized units.

Storage driver

Network driver

5.4.3 How does ClickID work?

5.4.4 Disabling the mikroBUS driver

If you'd like to use other means to control the mikroBUS connector, you might want to disable the mikroBUS driver. This is most easily done by enabling a device tree overlay at boot.

Todo: Document kernel version that integrates this overlay and where to get update instructions.

Note: To utilize the overlay with these instructions, make sure to have TBD version of kernel, modules and firmware installed. Use *uname -a* to determine the currently running kernel version. See TBD for information on how to update.

Apply overlay to disable mikrobus0 instance.

```
echo "    fdtoverlays /overlays/k3-am625-beagleplay-release-mikrobus.dtbo" |␣
↪sudo tee -a /boot/firmware/extlinux/extlinux.conf
sudo shutdown -r now
```

Log back in after reboot and verify the device driver did not capture the busses.

```
debian@BeaglePlay:~$ ls /dev/play
grove  mikrobus  qwiic
debian@BeaglePlay:~$ ls /dev/play/mikrobus/
i2c
debian@BeaglePlay:~$ ls /sys/bus/mikrobus/devices/
debian@BeaglePlay:~$ ls /proc/device-tree/chosen/overlays/
k3-am625-beagleplay-release-mikrobus  name
debian@BeaglePlay:~$
```

To re-enable.

```
sudo sed -e '/release-mikrobus/ s/^#*/#/' -i /boot/firmware/extlinux/
↪extlinux.conf
sudo shutdown -r now
```

Verify driver is enabled again.

```
debian@BeaglePlay:~$ ls /sys/bus/mikrobus/devices/
mikrobus-0
debian@BeaglePlay:~$ ls /proc/device-tree/chosen/overlays/
ls: cannot access '/proc/device-tree/chosen/overlays/': No such file or␣
↪directory
debian@BeaglePlay:~$
```

Todo:

- How do turn off the driver?
- How do turn on spidev?
- How do I enable GPIO?
- How do a provide a manifest?

Todo:

- Needs udev
- Needs live description

5.5 Using QWIIC

See qwiic_stemma_grove_addons.

A link to the appropriate I2C controller can be found at `/dev/play/qwiic/i2c`.

5.5.1 OLED Display using QWIIC

Let's see a simple way to use an I2C QWIIC OLED from Sparkfun with only minor modifications to the source code. (They will probably have this working by default in the future)

The Sparkfun Qwiic OLED Display Library Comes in 3 Parts:

- QWIIC_I2C_Py - We will need to modify this
- QWIIC-OLED-Base
- QWIIC-OLED-Display

The reason we need to modify Qwiic_I2C_Py is that by default, the library expects only one I2C Bus to be present for something like a Raspberry Pi, but our Beagle has many! Specifically, we want to use I2C-5 which is the bus connected to the QWIIC header.

5.5.2 Wiring/Connection

Make the connection as shown below.

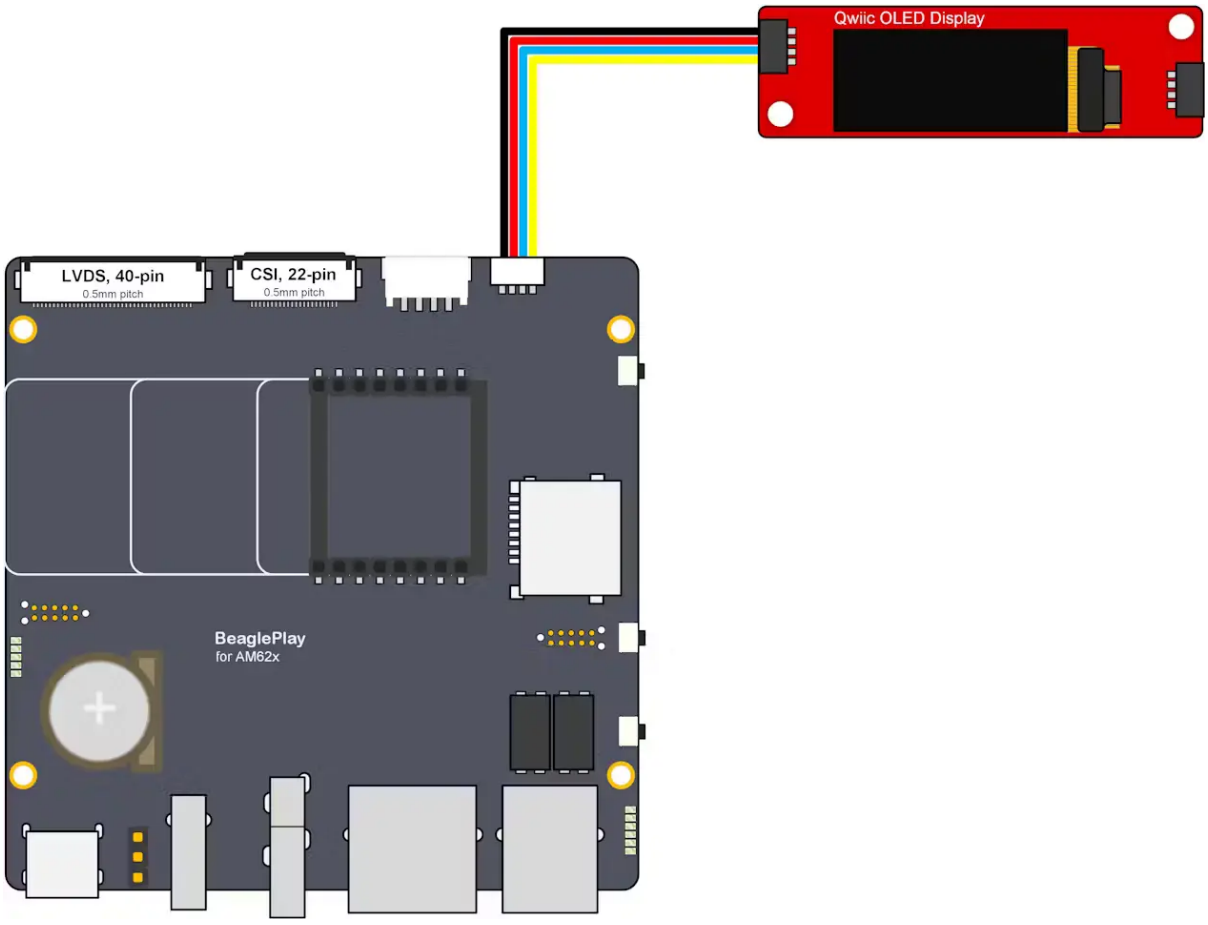


Fig. 5.16: BeaglePlay QWIIC OLED Connection

You can check what bus a device is connected to by scanning it. First lets see what buses are available.

```
debian@BeaglePlay:~$ ls /dev/ | grep "i2c"
i2c-0
i2c-1
i2c-2
```

(continues on next page)

(continued from previous page)

```
i2c-3
i2c-5
```

You can now scan each bus as follows:

```
i2cdetect -y -r 0
```

The `0` corresponds to `i2c-0`. we can then replace `0` with each bus until we find the oled, in this case, we know we are looking for a device at address `0x3C`.

```
debian@BeaglePlay:~$ i2cdetect -y -r 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Note that when we see a `UU`, this indicates that there is a device which is currently being used by another linux process.This is most likely another I2C device that the Beagle uses, such as the EEPROM. You can safely ignore this, but it's helpful to know what you're looking at.

Moving on, let's see Bus 5 (Hint, I2C-5 is the QWIIC connector):

```
debian@BeaglePlay:~$ i2cdetect -y -r 5
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

5.5.3 Using Python libraries to display on OLED.

let's install sparkfun Qwiic_I2C_Py Library.

```
git clone https://github.com/virtualRadish/Qwiic_I2C_Py_LC
```

Change directory to `Qwiic_I2C_Py_LC`.

```
cd Qwiic_I2C_Py_LC/
```

Install `setup.py`.

```
sudo python setup.py install
```

Install python libraries for OLED Displays.

```
sudo pip install sparkfun-qwiic-oled-base
sudo pip install sparkfun-qwiic-oled-display
```

Let's create a file `HelloWorld.py` to display some text on display.

```
nano HelloWorld.py
```

Now copy paste the text below, then press CTRL+O and ENTER to save, CTRL+X to exit.

```
from __future__ import print_function
import qwiic_oled_display
import sys
import time
def runExample():
    #  These three lines of code are all you need to initialize the
    #  OLED and print the splash screen.
    #  Before you can start using the OLED, call begin() to init
    #  all of the pins and configure the OLED.
    print("\nSparkFun OLED Display - Hello World Example\n")
    #  Create instance with parameters for Qwiic OLED Display
    myOLED = qwiic_oled_display.QwiicOledDisplay(0x3C)
    if not myOLED.connected:
        print("The Qwiic OLED Display isn't connected to the system. Please␣
↪check your connection", \
            file=sys.stderr)
        return
    myOLED.begin()

#  clear(ALL) will clear out the OLED's graphic memory.
    myOLED.clear(myOLED.ALL) #  Clear the display's memory (gets rid of␣
↪artifacts)
#  To actually draw anything on the display, you must call the display()␣
↪function.
    myOLED.display()  #  Display buffer contents
    time.sleep(1)
#  clear(PAGE) will clear the SBC display buffer.
    myOLED.clear(myOLED.PAGE)  #  Clear the display's buffer
#  Display buffer contents
    myOLED.display()
    time.sleep(1)
#  Print "Hello World"
#  ------------------------------------------------------------------------
↪-
#  Add text
    myOLED.print("Hello World!")
    myOLED.set_cursor(0, 10) # Set cursor to top-left
    myOLED.print("I'm BeaglePlay!")
#  Display buffer contents
    myOLED.display()

if __name__ == '__main__':
    try:
        runExample()
    except (KeyboardInterrupt, SystemExit) as exErr:
        print("\nEnding OLED Hello Example")
        sys.exit(0)
```

Now run it. After executing following command, “Hello World!” in first line and “I’m BeaglePlay!” in second line will be printed on OLED display.

```
python HelloWorld.py
```

Now, lets display our current IP Address.

Shout out out to this StackOverflow one-liner which gets our IP Address cleanly so we can display it as a string:

```
ipAddr = ((([ip for ip in socket.gethostbyname_ex(socket.gethostname())[2]␣
```

(continues on next page)

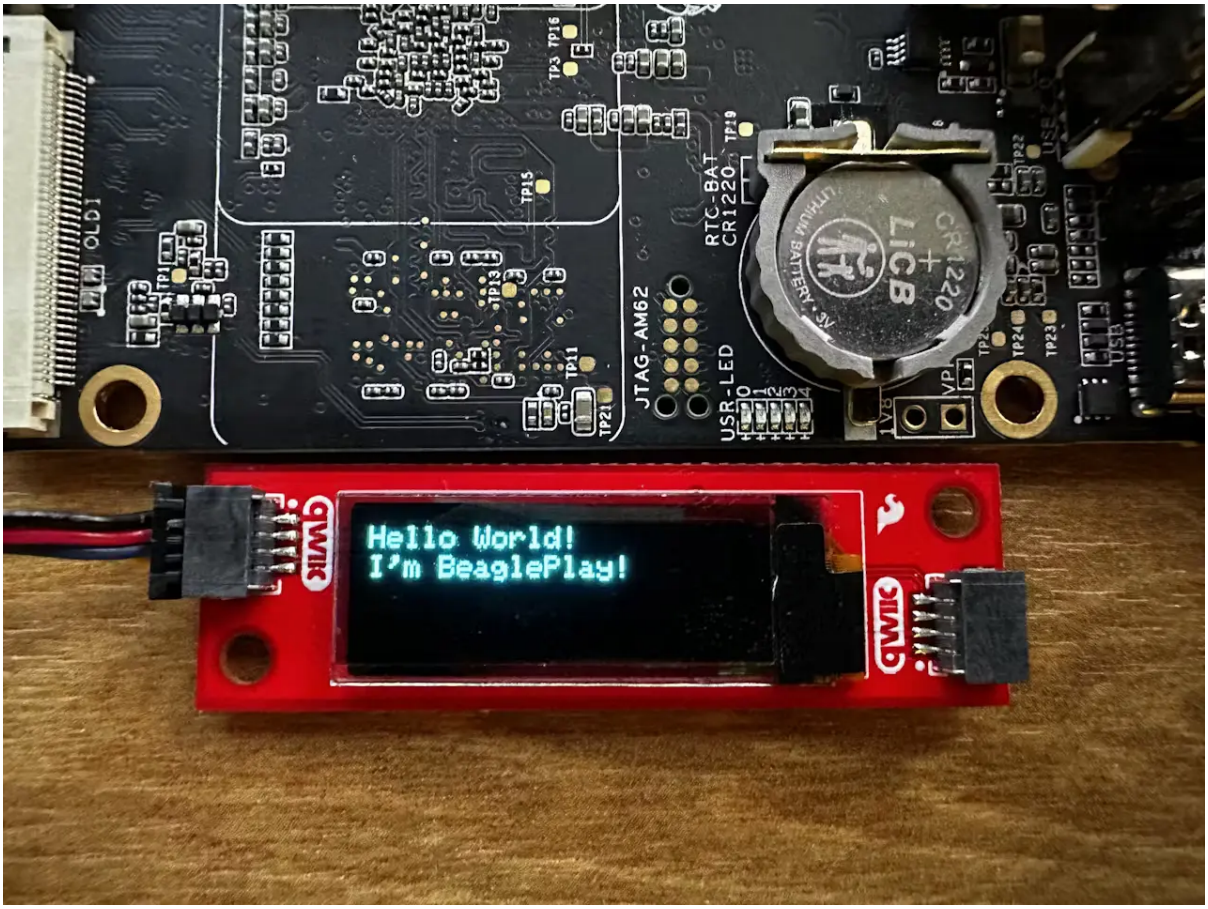


Fig. 5.17: BeaglePlay QWIIC OLED HelloWorld.py Output

(continued from previous page)

```
↪if not ip.startswith("127.")] or [[(s.connect(("8.8.8.8", 53)), s.
↪getsockname()[0], s.close()) for s in [socket.socket(socket.AF_INET,␣
↪socket.SOCK_DGRAM)]][0]
[1]]) + ["no IP found"])[0])
```

Additionally in above text we can display our current IP Address using below script. You can create a new file then copy paste it and run.

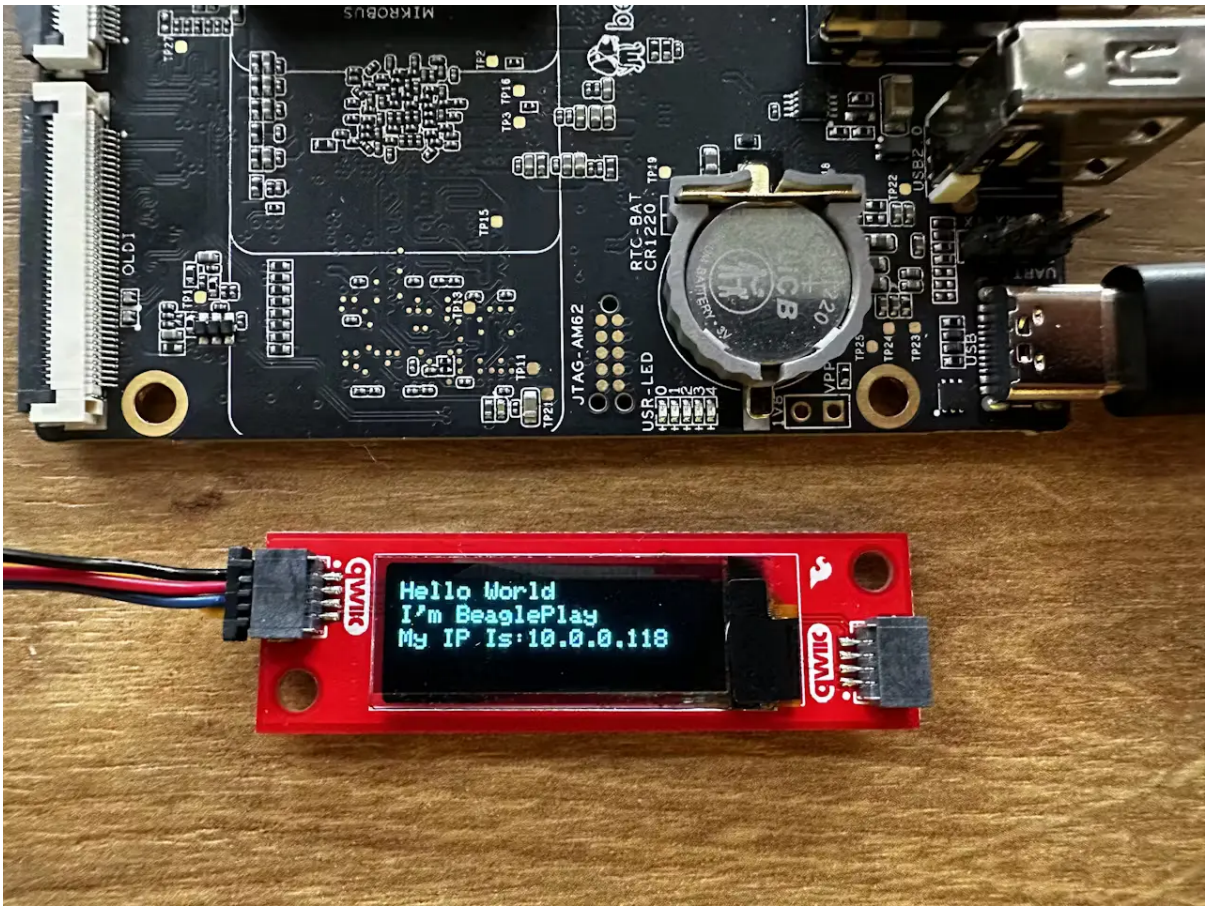
```
from __future__ import print_function
import qwiic_oled_display
import sys
import time
import socket

def runExample():

    IPAddr=(([ip for ip in socket.gethostbyname_ex(socket.gethostname())[2]␣
↪if not ip.startswith("127.")] or [[(s.connect(("8.8.8.8", 53)), s.
↪getsockname()[0], s.close()) for s in [socket.socket(socket.AF_INET,␣
↪socket.SOCK_DGRAM)]][0][1]]) + ["no IP found"])[0]

    #  These three lines of code are all you need to initialize the
    #  OLED and print the splash screen.
    #  Before you can start using the OLED, call begin() to init
    #  all of the pins and configure the OLED.
    print("\nSparkFun OLED Display - Hello World Example\n")
    #  Create instance with parameters for Qwiic OLED Display
    myOLED = qwiic_oled_display.QwiicOledDisplay(0x3C)
    if not myOLED.connected:
        print("The Qwiic OLED Display isn't connected to the system. Please␣
↪check your connection", \
              file=sys.stderr)
        return
    myOLED.begin()

#  clear(ALL) will clear out the OLED's graphic memory.
    myOLED.clear(myOLED.ALL) #  Clear the display's memory (gets rid of␣
↪artifacts)
#  To actually draw anything on the display, you must call the display()␣
↪function.
    myOLED.display()  #  Display buffer contents
    time.sleep(1)
#  clear(PAGE) will clear the SBC display buffer.
    myOLED.clear(myOLED.PAGE)  #  Clear the display's buffer
#  Display buffer contents
    myOLED.display()
    time.sleep(1)
#  Print "Hello World"
#  ------------------------------------------------------------------------
↪-
#  Add text
    myOLED.print("Hello World!")
    myOLED.set_cursor(0, 10) # Set cursor to top-left
    myOLED.print("I'm BeaglePlay!")
    myOLED.set_cursor(0, 25) # Set cursor to top-left
    myOLED.print("My IP Is:")
    myOLED.print(IPAddr)
#  Display buffer contents
    myOLED.display()

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    try:
        runExample()
    except (KeyboardInterrupt, SystemExit) as exErr:
        print("\nEnding OLED Hello Example")
        sys.exit(0)
```

You will now see current IP Address as well on OLED display.



Credits: Andrei Aldea, Nishka Rao, Brian Berner

5.6 Using RTC

BeaglePlay has an onboard Real Time Clock (BQ32002). If you have installed a CR1220 battery, you can use this to keep the time even when the device has been powered off, or has no Internet access to get time using ntp servers.

5.6.1 Understanding multiple rtc devices

On the BeaglePlay there's 2 separate RTC devices. One is the inbuilt one - which by default shows up as `/dev/rtc0`. And the other is BQ32002 - which is it's own discrete chip - and shows up as `/dev/rtc1` by default.

You can find out the time set in both these clocks with the `hwclock` command.

```
debian@BeaglePlay:~$ sudo hwclock -r --rtc /dev/rtc0
2023-12-21 12:43:52.007564+05:30
```

(continues on next page)

(continued from previous page)

```
debian@BeaglePlay:~$ sudo hwclock -r --rtc /dev/rtc1
1970-01-01 05:33:28.877722+05:30
```

Note that the time in `rtc0` has been set after booting up using ntp servers automatically.

5.6.2 Get the current time, timezone, and other settings

```
debian@BeaglePlay:~$ timedatectl
               Local time: Thu 2023-12-21 00:20:19 EST
           Universal time: Thu 2023-12-21 05:20:19 UTC
                 RTC time: Thu 2023-12-21 05:20:20
                Time zone: America/New_York (EST, -0500)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

The above command shows the time set on BeaglePlay, the universal time, the time set on the RTC, the timezone, and more. From the above we can see that the time in UTC is 5:20hrs and the time as per the timezone is 00:20hrs.

5.6.3 Setting the timezone

You can see the available timezones using the following command -

```
debian@BeaglePlay:~$ timedatectl list-timezones
```

You can quit viewing the list by pressing the q character on your keyboard.

Once you have selected your timezone, you can set it as follows

```
debian@BeaglePlay:~$ sudo timedatectl set-timezone America/New_York
```

5.6.4 Enable ntp

We can set the time using ntp servers. This requires us to be connected to the Internet.

```
debian@BeaglePlay:~$ sudo timedatectl set-ntp true
```

5.6.5 Setting the time manually

You might want to set the time manually on your BeaglePlay. In this case you need to first disable the ntp synchronization.

```
debian@BeaglePlay:~$ sudo timedatectl set-ntp false
debian@BeaglePlay:~$ sudo timedatectl set-time "2023-12-21 03:00:00"
```

Using the above command we have set the time to 0300hrs on 21st December 2023.

5.6.6 Using `rtcwake` to sleep

If you would like to put your BeaglePlay to sleep for a predetermined period of time, you can use the `rtcwake` command

```
debian@BeaglePlay:~$ sudo sudo rtcwake -m disk --seconds 120 -d /dev/rtc1 -v
Using UTC time.
     delta   = 0
     tzone   = 0
     tzname  = UTC
     systime = 1703147162, (UTC) Thu Dec 21 08:26:02 2023
     rtctime = 1703147162, (UTC) Thu Dec 21 08:26:02 2023
alarm 0, sys_time 1703147162, rtc_time 1703147162, seconds 120
rtcwake: wakeup from "disk" using /dev/rtc1 at Thu Dec 21 08:28:03 2023
suspend mode: disk; suspending system
```

The above command puts your BeaglePlay to sleep for 120 seconds, by writing the contents of your memory to disk. You can find what are the different modes that are supported similar to `disk` by running the `--list-modes` subcommand.

```
debian@BeaglePlay:~$ rtcwake --list-modes
freeze mem disk off no on disable show
```

5.7 Using OLDI Displays

5.8 Using CSI Cameras

5.9 Wireless MCU Zephyr Development

BeaglePlay includes a Texas Instruments CC1352P7 wireless microcontroller (MCU) that can be programmed using the Linux Foundation Zephyr RTOS.

Developing directly in Zephyr will not be ultimately required for end-users who won't touch the firmware running on the CC1352 on BeaglePlay™ and will instead use the provided wireless functionality. However, it is important for early adopters as well as people looking to extend the functionality of the open source design. If you are one of those people, this is a good place to get started.

Further, BeaglePlay is a reasonable development platform for creating Zephyr-based applications for beagleconnect_freedom_home. The same Zephyr development environment setup here is also described for targeting applications on that board.

5.9.1 Install the latest software image for BeaglePlay

Note: These instructions should be generic for BeaglePlay and other boards and only the specifics of which image was used to test these instructions need be included here moving forward and the detailed instructions can be referenced elsewhere.

You may want to download and install the latest Debian Linux operating system image for BeaglePlay.

Note: These instructions were validated with the BeagleBoard.org Debian image BeaglePlay Debian 11.6 Flasher 2023-03-10.

---

1. Load this image to a microSD card using a tool like Etcher.
2. Insert the microSD card into BeaglePlay.
3. Power BeaglePlay via the USB-C connector.
4. Wait for the LEDs to start blinking, then turn off.

5. Remove power from BeaglePlay.
6. *IMPORTANT* Remove microSD card from BeaglePlay.
7. Apply power to BeaglePlay.

Note: This will flash the CC1352 as well as the eMMC flash on BeaglePlay.

Todo: Describe how to know it is working

5.9.2 Log into BeaglePlay

Please either plug in a keyboard, monitor and mouse or `ssh` into the board. We can point somewhere else for instructions on this. You can also point your web browser to the board to log into the Visual Studio Code IDE environment.

Todo: A big part of what is missing here is to put your BeaglePlay on the Internet such that we can download things in later steps. That has been initially brushed over.

5.9.3 Flash existing IEEE 802.15.4 radio bridge (WPANUSB) firmware

If you've recieved a board fresh from the factory, this is already done and not necessary, unless you want to restore the contents back to the factory condition.

Background

This *WPANUSB* application was originally developed for radio devices with a USB interface. The CC1352P7 does not have a USB device, so the application was modified to communicate over a UART serial interface.

For the beagleconnect_freedom_home, a USB-to-UART bridge device was used and the USB endpoints were made compatible with the WPANUSB linux driver which we augmented to support this board. To utilize the existing *WPANUSB* Zephyr application and this Linux driver, we chose to encode our UART traffic with HDLC. This has the advantage of enabing a serial console interface to the Zephyr shell while WPANUSB-specific traffic is directed to other USB endpoints.

For BeaglePlay, the USB-to-UART bridge is not used, but we largely kept the same *WPANUSB* application, including the HDLC encoding.

Note: Now you know why this WPAN bridge application is called *WPANUSB*, even though USB isn't used!

Steps

1. Ensure the *bcfserial* driver isn't blocking the serial port.

```
echo "    fdtoverlays /overlays/k3-am625-beagleplay-bcfserial-no-
↪firmware.dtbo" | sudo tee -a /boot/firmware/extlinux/extlinux.
↪conf
sudo shutdown -r now
```

Note: The default password is *temppwd*.

2. Download and flash the *WPANUSB* Zephyr application firmware onto the CC1352P7 on BeaglePlay from the releases on git.beagleboard.org or distros on www.beagleboard.org/distros.

```
debian@BeaglePlay:~$ wget https://files.beagle.cc/file/
↪beagleboard-public-2021/images/zephyr-beagle-cc1352-0.2.2.zip
debian@BeaglePlay:~$ unzip zephyr-beagle-cc1352-0.2.2.zip
debian@BeaglePlay:~$ build/play/cc2538-bsl.py build/play/wpanusb
```

3. Ensure the *bcfserial* driver is set to load.

```
sudo sed -e '/bcfserial-no-firmware/ s/^#*/#/' -i /boot/firmware/
↪extlinux/extlinux.conf
sudo shutdown -r now
```

4. Verify the the 6LoWPAN network is up.

```
debian@BeaglePlay:~$ lsmod | grep bcfserial
bcfserial              24576  0 ①
mac802154              77824  2 wpanusb,bcfserial
debian@BeaglePlay:~$ ifconfig
SoftAp0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.8.1  netmask 255.255.255.0  broadcast 192.
↪168.8.255
        inet6 fe80::3ee4:b0ff:fe7e:b5f7  prefixlen 64  scopeid␣
↪0x20<link>
        ether 3c:e4:b0:7e:b5:f7  txqueuelen 1000  (Ethernet)
        RX packets 4046  bytes 576780 (563.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4953  bytes 5116336 (4.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.
↪255.255
        ether 02:42:f8:29:41:69  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether f4:84:4c:fc:5d:13  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 246239  bytes 19948296 (19.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 246239  bytes 19948296 (19.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0
```

(continues on next page)

(continued from previous page)

```
lowpan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1280 ②
        inet6 fe80::200:0:0:0  prefixlen 64  scopeid 0x20<link> ③
        inet6 2001:db8::2  prefixlen 64  scopeid 0x0<global> ④
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 ␣
↪txqueuelen 1000  (UNSPEC)
        RX packets 107947  bytes 6629290 (6.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2882  bytes 179511 (175.3 KiB) ⑤
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.7.2  netmask 255.255.255.0  broadcast 192.
↪168.7.255
        inet6 fe80::1eba:8cff:fea2:ed6b  prefixlen 64  scopeid␣
↪0x20<link>
        ether 1c:ba:8c:a2:ed:6b  txqueuelen 1000  (Ethernet)
        RX packets 9858  bytes 2638440 (2.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4155  bytes 1454082 (1.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.6.2  netmask 255.255.255.0  broadcast 192.
↪168.6.255
        inet6 fe80::1eba:8cff:fea2:ed6d  prefixlen 64  scopeid␣
↪0x20<link>
        ether 1c:ba:8c:a2:ed:6d  txqueuelen 1000  (Ethernet)
        RX packets 469614  bytes 35385636 (33.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 365548  bytes 66523708 (63.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.161  netmask 255.255.255.0  broadcast 192.
↪168.0.255
        inet6 fe80::3ee4:b0ff:fe7e:b5f6  prefixlen 64  scopeid␣
↪0x20<link>
        inet6 2601:408:c083:b6c0::d00d  prefixlen 128  scopeid␣
↪0x0<global>
        ether 3c:e4:b0:7e:b5:f6  txqueuelen 1000  (Ethernet)
        RX packets 3188898  bytes 678154090 (646.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1162074  bytes 293237366 (279.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0

wpan0: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 123 ⑥
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 ␣
↪txqueuelen 300  (UNSPEC)
        RX packets 108495  bytes 2539160 (2.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2888  bytes 140523 (137.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions␣
↪0
```

① You'll want to see that the *bcfserial* driver has been loaded.

② There should be a *lowpan0* interface.

③ There should be a link-local address for *lowpan0*.

④ There should be a global address for *lowpan0*.

⑤ Seeing some packets have been transmitted can give you some confidence.

⑥ The *wpan0* interface should be there, but we have a 6LoWPAN adapter on top of it.

Note: You may find Linux-WPAN.org useful.

5.9.4 Setup Zephyr development on BeaglePlay

1. Download and setup Zephyr for BeaglePlay

```
cd
sudo apt update
sudo apt install --no-install-recommends -y \
    gperf \
    ccache dfu-util \
    libsdl2-dev \
    libxml2-dev libxslt1-dev libssl-dev libjpeg62-turbo-dev␣
↪libmagic1 \
    libtool-bin autoconf automake libusb-1.0-0-dev \
    python3-tk python3-virtualenv
wget https://github.com/zephyrproject-rtos/sdk-ng/releases/
↪download/v0.15.1/zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
tar xf zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
rm zephyr-sdk-0.15.1_linux-aarch64_minimal.tar.gz
./zephyr-sdk-0.15.1/setup.sh -t arm-zephyr-eabi -c
west init -m https://git.beagleboard.org/beagleconnect/zephyr/
↪zephyr --mr sdk zephyr-beagle-cc1352-sdk
cd $HOME/zephyr-beagle-cc1352-sdk
python3 -m virtualenv zephyr-beagle-cc1352-env
echo "export ZEPHYR_TOOLCHAIN_VARIANT=zephyr" >> $HOME/zephyr-
↪beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
echo "export ZEPHYR_SDK_INSTALL_DIR=$HOME/zephyr-sdk-0.15.1" >>
↪$HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo "export ZEPHYR_BASE=$HOME/zephyr-beagle-cc1352-sdk/zephyr" >
↪> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo 'export PATH=$HOME/zephyr-beagle-cc1352-sdk/zephyr/scripts:
↪$PATH' >> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-
↪env/bin/activate
echo "export BOARD=beagleplay" >> $HOME/zephyr-beagle-cc1352-sdk/
↪zephyr-beagle-cc1352-env/bin/activate
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
west update
west zephyr-export
pip3 install -r zephyr/scripts/requirements-base.txt
```

2. Activate the Zephyr build environment

   If you exit and come back, you'll need to reactivate your Zephyr build environment.

```
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

3. Verify Zephyr setup for BeaglePlay

```
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/
↪cmake).
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ python3 --version
Python 3.9.2
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ dtc --version
Version: DTC 1.6.0
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ west --version
West version: v0.14.0
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ ./zephyr-sdk-0.15.1/
↪arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc --version
arm-zephyr-eabi-gcc (Zephyr SDK 0.15.1) 12.1.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. ␣
↪There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A␣
↪PARTICULAR PURPOSE.
```

5.9.5 Build applications for BeaglePlay CC1352

Now you can build various Zephyr applications

1. Build and flash Blinky example

```
cd $HOME/zephyr-beagle-cc1352-sdk/zephyr
west build -d build/play_blinky samples/basic/blinky
west flash -d build/play_blinky
```

2. Try out Micropython

```
cd
git clone -b beagleplay-cc1352 https://git.beagleboard.org/
↪beagleplay/micropython
cd micropython
west build -d play ports/zephyr
west flash -d play
tio /dev/ttyS4
```

5.9.6 Build applications for BeagleConnect Freedom

1. Build and flash Blinky example

```
cd $HOME/zephyr-beagle-cc1352-sdk/zephyr
west build -d build/freedom_blinky -b beagleconnect_freedom␣
↪samples/basic/blinky
west flash -d build/freedom_blinky
```

2. Try out Micropython

```
cd
git clone -b beagleplay-cc1352 https://git.beagleboard.org/
↪beagleplay/micropython
cd micropython
west build -d freedom -b beagleconnect_freedom ports/zephyr
west flash -d freedom
tio /dev/ttyACM0
```

Important: Nothing below here is tested

Todo:

```
west build -d build/sensortest zephyr/samples/boards/beagle_bcf/sensortest --
↪ -DOVERLAY_CONFIG=overlay-subghz.conf
```

```
west build -d build/wpanusb modules/lib/wpanusb_bc -- -DOVERLAY_
↪CONFIG=overlay-subghz.conf
```

```
west build -d build/bcfserial modules/lib/wpanusb_bc -- -DOVERLAY_
↪CONFIG=overlay-bcfserial.conf -DDTC_OVERLAY_FILE=bcfserial.overlay
```

```
west build -d build/greybus modules/lib/greybus/samples/subsys/greybus/net --
↪ -DOVERLAY_CONFIG=overlay-802154-subg.conf
```

Flash applications to BeagleConnect Freedom

And then you can flash the BeagleConnect Freedom boards over USB

1. **Make sure you are in Zephyr directory**

   ```
   cd $HOME/bcf-zephyr
   ```

2. **Flash Blinky**

   ```
   cc2538-bsl.py build/blinky
   ```

Debug applications over the serial terminal

Todo: Describe how to handle the serial connection

5.10 BeaglePlay Kernel Development

This guide is for all those who want to kick start their kernel development journey on the TI AM625x SoC Based BeaglePlay.

5.10.1 Getting the Kernel Source Code

The Linux kernel is hosted on a number of servers around the world. The main repository is hosted on the kernel.org website, but there are also mirrors hosted by other organizations, such as GitHub and Bootlin.

The Linux Torvalds tree is the most up-to-date source of the Linux kernel. It is used by Linux distributions and other projects to build their own kernels. The tree is also a popular destination for kernel developers who want to contribute to the kernel.

Kernel sources can directly be fetched using `git`:

```
git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

A big advantage of using `git` to fetch the kernel sources is that you'll easily be able to manage your changes, keeping track of what you might edit. If you are looking for a quicker way to download a single version of the Linux kernel sources to get started, you might consider fetching a "tarball" using `wget`.

```
wget https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/
↪snapshot/linux-6.6.tar.gz
tar xf linux-6.6.tar.gz
```

Note: While fetching a tarball with `wget` might be faster than fetching the full history with `git`, the ability to track changes with `git` is significant.

For more information on using `git`, see beagleboard-git-usage.

5.10.2 Preparing to Build

These instructions should be valid on any Debian-based system, but were tested on a BeaglePlay itself.

```
sudo apt update
sudo apt install -y fakeroot build-essential libncurses-dev xz-utils libssl-
↪dev flex libelf-dev bison debhelper
```

5.10.3 Configuring the Kernel

The easiest way to configure the kernel is to start with a configuration known to work. A running BeaglePlay is a great source for that configuration, as it gets compiled into the running kernel.

Note: If you don't have a BeaglePlay booted, you can copy a known good kernel configuration from the BeagleBoard.org Linux git repository at https://git.beagleboard.org/beagleboard/linux. On each release branch, the last commit typically contains a `bb.org_defconfig` file. For BeaglePlay, you should look for an `arm64` branch.

Example: https://git.beagleboard.org/beagleboard/linux/-/blob/f47f74d11b19d8ae2f146de92c258f40e0930d86/arch/arm64/configs/bb.org_defconfig

Running on a BeaglePlay, you can configure your kernel using `/proc/config.gz`. You'll also want to `make olddefconfig` to update your config for the newer kernel. If you want to look at configuration options that haven't previously been configured, then use `make oldconfig` instead. Once you've got an initial configuration, you can edit the configuration various ways including `make menuconfig`.

```
cd linux-6.6
zcat /proc/config.gz > .config
make olddefconfig
```

You can also take advantage of the running system to provide the WiFi regulatory database (regulatory.db). This is needed such that your kernel sets the WiFi signals appropriately for compliance with regional restrictions.

For more information, see Linux wireless regulatory documentation and the signed database images at https://git.kernel.org/pub/scm/linux/kernel/git/sforshee/wireless-regdb.git/tree/.

```
mkdir -p firmware
cp /lib/firmware/regulatory.db* firmware/
```

5.10.4 Building the Kernel

Once you're set on your configuration, you'll want to build the kernel and build any external modules. To make things simpler to install, we'll create a Debian package of the kernel.

Note: Building the kernel on BeaglePlay might take a while. For me, it took about an hour.

```
cd ..
make -C ./linux-6.6 -j4 KDEB_PKGVERSION=1xross bindeb-pkg
```

5.10.5 Installing and Booting the Kernel

Important: In case your new kernel fails, you'll want to be prepared to either reflash the board or to use a serial cable to halt u-boot and request loading a working kernel still available on the board.

See *Using Serial Console* to setup access over the debug serial port.

Listing 5.1: Install 6.6.0 kernel and reboot

```
    sudo dpkg -i linux-image-6.6.0_1xross_arm64.deb
    sudo shutdown -r now
```

As long as the kernel you built has no significant issues, you'll boot back into a running system.

If there was a boot or connectivity failure, you can try an alternate connectivity method, such as the *Using Serial Console* or Ethernet, or you can reflash the board and try again from a known good kernel source.

For me, the linux-6.6 kernel booted fine, but the beagleplay.local (mDNS/Avahi broadcast) address did not show up right away. I was able to find the BeaglePlay hosted WiFi access point, the connection to my local WiFi network, connect over Ethernet and connect over USB network. The */dev/play* directory did not exist, but the */dev/bone* directory did, so this gives me a good starting point for generating some patches to update the mainline kernel. :-D

See beagleboard-linux-upstream for more next steps by providing updates you make to the kernel to the upstream repository for everyone to benefit and for you to benefit from on future kernel versions.

5.10.6 Kernel Debug

Consider reading the kernel documentation on debugging via gdb.

Also, consider the the TI Linux Board Porting Series, specifically the module on debugging with JTAG in CCS.

5.10.7 References

- To understand more about booting code on BeaglePlay, see *Understanding Boot*.
- For more details on the Linux kernel build system, see The kernel build system on kernel.org.
- For additional guidance, see the official TI-SDK documentation for AM62X

5.11 BeagleConnect™ Greybus demo using BeagleConnect™ Freedom and BeaglePlay

5.11.1 BeaglePlay CC1352 Firmware

Build (Download and Setup Zephyr for BeaglePlay)

1. Install prerequisites

```
cd
sudo apt update
sudo apt install --no-install-recommends -y \
    gperf \
    ccache dfu-util \
    libsdl2-dev \
    libxml2-dev libxslt1-dev libssl-dev libjpeg62-turbo-dev␣
↪libmagic1 \
    libtool-bin autoconf automake libusb-1.0-0-dev \
    python3-tk python3-virtualenv
```

2. Download the latest Zephyr Release, extract it and cleanup

```
sudo wget https://github.com/zephyrproject-rtos/sdk-ng/releases/
↪download/v0.16.3/zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
tar xf zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
rm zephyr-sdk-0.16.3_linux-aarch64_minimal.tar.xz
```

3. Run the Zephyr SDK Setup Script

```
./zephyr-sdk-0.16.3/setup.sh -t arm-zephyr-eabi -c
```

4. Download and Initialize West. (Zephyr's meta-tool)

Note: You may want to add */home/debian/.local/bin* to your *.bashrc* file to make the West command available after a reboot

---

```
pip3 install --user -U west
export PATH="/home/debian/.local/bin:$PATH"
west init -m https://git.beagleboard.org/beagleconnect/zephyr/
↪zephyr --mr sdk-next zephyr-beagle-cc1352-sdk
cd $HOME/zephyr-beagle-cc1352-sdk
```

5. Setup a Python Virtual Environment and add our PATH Variables

```
virtualenv zephyr-beagle-cc1352-env
echo "export ZEPHYR_TOOLCHAIN_VARIANT=zephyr" >> $HOME/zephyr-
↪beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
echo "export ZEPHYR_SDK_INSTALL_DIR=$HOME/zephyr-sdk-0.16.3" >>
↪$HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo "export ZEPHYR_BASE=$HOME/zephyr-beagle-cc1352-sdk/zephyr" >
↪> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/bin/
↪activate
echo 'export PATH=$HOME/zephyr-beagle-cc1352-sdk/zephyr/scripts:
↪$PATH' >> $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-
↪env/bin/activate
echo "export BOARD=beagleplay_cc1352" >> $HOME/zephyr-beagle-
↪cc1352-sdk/zephyr-beagle-cc1352-env/bin/activate
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

6. Update West

```
west update
west zephyr-export
```

7. Install Python Prerequisites

```
pip3 install -r zephyr/scripts/requirements-base.txt
```

8. Activate the Zephyr build environment

   NOTE - If you exit and come back, you'll need to reactivate your Zephyr build environment.

```
source $HOME/zephyr-beagle-cc1352-sdk/zephyr-beagle-cc1352-env/
↪bin/activate
```

9. Verify Zephyr setup for BeaglePlay

```
(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/
↪cmake).

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ python3 --version
Python 3.9.2

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ dtc --version
Version: DTC 1.6.0

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ west --version
West version: v0.14.0

(zephyr-beagle-cc1352-env) debian@BeaglePlay:~$ ./zephyr-sdk-0.16.3/
↪arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc --version
arm-zephyr-eabi-gcc (Zephyr SDK 0.16.3) 12.1.0

Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. ␣
↪There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A␣
↪PARTICULAR PURPOSE.
```

10. Clone CC1352 Firmware at top level: https://git.beagleboard.org/gsoc/greybus/cc1352-firmware

```
cd ~
git clone https://git.beagleboard.org/gsoc/greybus/cc1352-
↪firmware
```

11. Build the Firmware

```
west build -b beagleplay_cc1352 -p always cc1352-firmware
```

12. You can now find the built firmware at *build/zephyr/zephyr.bin*

Flash

1. Ensure the *gb-beagleplay* driver isn't blocking the serial port.

```
debian@BeaglePlay:~$ echo "    fdtoverlays /overlays/k3-am625-
↪beagleplay-bcfserial-no-firmware.dtbo" | sudo tee -a /boot/
↪firmware/extlinux/extlinux.conf
debian@BeaglePlay:~$ sudo shutdown -r now
```

Note: The default password is *temppwd*.

2. Clone cc1352-flasher

```
cd
git clone https://git.beagleboard.org/beagleconnect/cc1352-
↪flasher.git
```

3. Flash Firmware

```
python $HOME/cc1352-flasher --beagleplay $HOME/zephyr-beagle-
↪cc1352-sdk/build/zephyr/zephyr.bin
```

4. Ensure the *gb-beagleplay* driver is set to load.

```
sudo sed -e '/bcfserial-no-firmware/ s/^#*/#/' -i /boot/firmware/
↪extlinux/extlinux.conf
sudo shutdown -r now
```

5.11.2 Building gb-beagleplay Kernel Module

Note: *gb-beagleplay* is still not merged upstream and thus needs to be built seperately. This should not be required in the future.

1. Disable bcfserial driver. Add `quiet module_blacklist=bcfserial` to kernel parameters at `/boot/firmware/extlinux/extlinux.conf` (line 2) as shown below.

```
label Linux eMMC
    kernel /Image
    append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.
↪ifnames=0 quiet module_blacklist=bcfserial ①
    fdtdir /
    #fdtoverlays /overlays/<file>.dtbo
    #fdtoverlays /overlays/k3-am625-beagleplay-bcfserial-no-firmware.
↪dtbo
    fdtoverlays /overlays/k3-am625-beagleplay-release-mikrobus.dtbo
    initrd /initrd.img
```

① `quiet module_blacklist=bcfserial` has been added to this line

1. Reboot

```
debian@BeaglePlay:~$ sudo shutdown -r now
```

2. Download the upstream module

```
debian@BeaglePlay:~$ git clone https://git.beagleboard.org/gsoc/
↪greybus/beagleplay-greybus-driver.git
debian@BeaglePlay:~$ cd beagleplay-greybus-driver
```

3. Install dependencies

```
debian@BeaglePlay:~$ sudo apt install linux-headers-$(uname -r)
```

4. Build Kernel moudle

```
debian@BeaglePlay:~/beagleplay-greybus-driver$ make
make -C /lib/modules/5.10.168-ti-arm64-r111/build M=/home/debian/
↪beagleplay-greybus-driver modules
make[1]: Entering directory '/usr/src/linux-headers-5.10.168-ti-
↪arm64-r111'
  CC [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.o
  MODPOST /home/debian/beagleplay-greybus-driver/Module.symvers
  CC [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.
↪mod.o
  LD [M]  /home/debian/beagleplay-greybus-driver/gb-beagleplay.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.10.168-ti-
↪arm64-r111'
```

5.11.3 Flashing BeagleConnect Freedom Greybus Firmware

1. Connect BeagleConnect Freedom to BeaglePlay
2. Build the BeagleConnect Freedom firmware

```
west build -b beagleconnect_freedom modules/greybus/samples/
↪subsys/greybus/net/ -p -- -DOVERLAY_CONFIG=overlay-802154-subg.
↪conf
```

3. Flash the BeagleConnect Freedom

```
west flash
```

5.11.4 Run the Demo

1. Connect BeagleConnect Freedom.
2. See shell output using *tio*

```
tio /dev/ACM0
```

3. Press the Reset button on BeagleConnect Freedom
4. Verify that greybus is working by checking the *tio* output. It should look as follows:

```
[00:00:00.000,976] <dbg> greybus_platform_bus: greybus_init:␣
↪probed greybus: 0 major: 0 minor: 1
[00:00:00.001,068] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 4: hdc2010
[00:00:00.001,129] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 3: opt3001
[00:00:00.001,190] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 2: Greybus Service Sample␣
↪Application
[00:00:00.001,251] <dbg> greybus_platform_string: greybus_string_
↪init: probed greybus string 1: Zephyr Project RTOS
[00:00:00.001,251] <dbg> greybus_platform_interface: greybus_
↪interface_init: probed greybus interface 0
[00:00:00.001,281] <dbg> greybus_platform_bundle: greybus_bundle_
↪init: probed greybus bundle 1: class: 10
[00:00:00.001,312] <dbg> greybus_platform_bundle: greybus_bundle_
↪init: probed greybus bundle 0: class: 0
[00:00:00.001,342] <dbg> greybus_platform_control: greybus_
↪control_init: probed cport 0: bundle: 0 protocol: 0
[00:00:00.001,434] <dbg> greybus_platform: gb_add_cport_device_
↪mapping: added mapping between cport 1 and device gpio@40022000
```

(continues on next page)

(continued from previous page)

```
[00:00:00.001,464] <dbg> greybus_platform_gpio_control: greybus_
↪gpio_control_init: probed cport 1: bundle: 1 protocol: 2
[00:00:00.001,556] <dbg> greybus_platform: gb_add_cport_device_
↪mapping: added mapping between cport 2 and device sensor-switch
[00:00:00.001,556] <dbg> greybus_platform_i2c_control: greybus_
↪i2c_control_init: probed cport 2: bundle: 1 protocol: 3
*** Booting Zephyr OS build bcf-sdk-0.2.1-3384-ge76584f824c8 ***
[00:00:00.009,704] <dbg> greybus_service: greybus_service_init:␣
↪Greybus initializing..
[00:00:00.009,765] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 0
[00:00:00.009,796] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 1
[00:00:00.009,826] <dbg> greybus_manifest: identify_descriptor:␣
↪cport_id = 2
[00:00:00.009,857] <dbg> greybus_transport_tcpip: gb_transport_
↪backend_init: Greybus TCP/IP Transport initializing..
[00:00:00.010,101] <inf> greybus_transport_tcpip: CPort 0 mapped␣
↪to TCP/IP port 4242
[00:00:00.014,709] <inf> greybus_transport_tcpip: CPort 1 mapped␣
↪to TCP/IP port 4243
[00:00:00.014,953] <inf> greybus_transport_tcpip: CPort 2 mapped␣
↪to TCP/IP port 4244
[00:00:00.015,075] <inf> greybus_transport_tcpip: Greybus TCP/IP␣
↪Transport initialized
[00:00:00.015,136] <inf> greybus_manifest: Registering CONTROL␣
↪greybus driver.
[00:00:00.015,167] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP0
[00:00:00.015,411] <inf> greybus_manifest: Registering GPIO␣
↪greybus driver.
[00:00:00.015,411] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP1
[00:00:00.015,625] <inf> greybus_manifest: Registering I2C␣
↪greybus driver.
[00:00:00.015,625] <dbg> greybus: _gb_register_driver:␣
↪Registering Greybus driver on CP2
[00:00:00.015,777] <inf> greybus_service: Greybus is active
```

5. Load gb-beagleplay

```
debian@BeaglePlay:~$ sudo insmod $HOME/beagleplay-greybus-driver/
↪gb-beagleplay.ko
```

6. Check *iio_device* to verify that greybus node has been detected:

```
debian@BeaglePlay:~$ iio_info
Library version: 0.24 (git tag: v0.24)
Compiled with backends: local xml ip usb
IIO context created with local backend.
Backend version: 0.24 (git tag: v0.24)
Backend description string: Linux BeaglePlay 5.10.168-ti-arm64-
↪r111 #1bullseye SMP Tue Sep 26 14:22:20 UTC 2023 aarch64
IIO context has 2 attributes:
        local,kernel: 5.10.168-ti-arm64-r111
        uri: local:
IIO context has 2 devices:
        iio:device0: adc102s051
                2 channels found:
                        voltage1:  (input)
                        2 channel-specific attributes found:
```

(continues on next page)

(continued from previous page)

```
                                attr  0: raw value: 4068
                                attr  1: scale value: 0.805664062
                        voltage0:  (input)
                        2 channel-specific attributes found:
                                attr  0: raw value: 0
                                attr  1: scale value: 0.805664062
                No trigger on this device
        iio:device1: hdc2010
                3 channels found:
                        temp:  (input)
                        4 channel-specific attributes found:
                                attr  0: offset value: -15887.
↪515151
                                attr  1: peak_raw value: 28928
                                attr  2: raw value: 28990
                                attr  3: scale value: 2.517700195
                        humidityrelative:  (input)
                        3 channel-specific attributes found:
                                attr  0: peak_raw value: 43264
                                attr  1: raw value: 41892
                                attr  2: scale value: 1.525878906
                        current:  (output)
                        2 channel-specific attributes found:
                                attr  0: heater_raw value: 0
                                attr  1: heater_raw_available␣
↪value: 0 1
                No trigger on this device
```

5.12 Understanding Boot

There are several phases to BeaglePlay boot. The simplest place to take control of the system is using *Distro Boot*. It is simplest because it is very generic, not at all specific to BeaglePlay or AM62, and was included in the earliest BeagleBoard.org Debian images shipping pre-installed in the on-board flash.

5.12.1 Distro Boot

For some background on distro boot, see the u-boot documentation on distro boot. There is also BeaglePlay specific u-boot documentation.

In *Typical /boot/firmware/extlinux/extlinux.conf file*, you can see line 1 provides a label and subsequent indented lines provide parameters for that boot option.

Listing 5.2: Typical /boot/firmware/extlinux/extlinux.conf file

```
1 label Linux eMMC
2     kernel /Image
3     append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.ifnames=0␣
  ↪quiet
4     fdtdir /
5     #fdtoverlays /overlays/<file>.dtbo
6     initrd /initrd.img
```

It is important to note that this file is not on the root file system of BeaglePlay. It is sitting on a separate FAT32 partition that is mounted at `/boot/firmware`. You can see the mounted file systems and their formats in *List of mounted file systems*.

The FAT32 partition in this setup is often referred to as the boot file system.

Listing 5.3: List of mounted file systems

```
debian@BeaglePlay:~$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
udev              903276        0    903276   0% /dev
tmpfs             197324     1524    195800   1% /run
/dev/mmcblk0p2  14833640 12144024   1914296  87% /
tmpfs             986608        0    986608   0% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
/dev/mmcblk0p1    130798    53214     77584  41% /boot/firmware
tmpfs             197320       32    197288   1% /run/user/1000
debian@BeaglePlay:~$ lsblk
NAME         MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0      179:0    0 14.6G  0 disk
├─mmcblk0p1  179:1    0  128M  0 part /boot/firmware
└─mmcblk0p2  179:2    0 14.5G  0 part /
mmcblk0boot0 179:256  0    4M  1 disk
mmcblk0boot1 179:512  0    4M  1 disk
debian@BeaglePlay:~$ sudo sfdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 14.6 GiB, 15678308352 bytes, 30621696 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xba67172a

Device         Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1 *      2048   264191   262144  128M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      264192 30621695 30357504 14.5G 83 Linux
```

To better understand BeaglePlay's U-Boot Distro Boot, let's install a Linux kernel and initramfs from the Buildroot project. There is a pre-built image release at https://git.beagleboard.org/beagleboard/buildroot/-/releases/2023.11-beagle1.

Currently, the Linux kernel image needs to be uncompressed and stored in the FAT32 file system. An initramfs image is a simple way to provide a starting root file system. When running Linux, some kind of root file system is required.

An initramfs image is utilized on Debian systems to make sure any kernel modules needed are available and to provide a bit of a recovery opportunity in case the root file system is corrupted. You can learn more about initramfs and initrd on *the Debian Initrd Wiki page <https://wiki.debian.org/Initrd>* and *the Linux kernel documentation admin guide initrd entry <https://docs.kernel.org/admin-guide/initrd.html>*.

In the case of utilizing Buildroot, the entire Linux distribution is incorporated into the initramfs root file system image.

The contents of the initrd can be read using `lsinitramfs /boot/firmware/initrd.img`.

Listing 5.4: Copy kernel to FAT32 filesystem

```
debian@BeaglePlay:~$ wget https://git.beagleboard.org/beagleboard/buildroot/-/
↪jobs/19194/artifacts/raw/public/beagleplay/images/Image
--2023-12-19 22:17:54--  https://git.beagleboard.org/beagleboard/buildroot/-/
↪jobs/19194/artifacts/raw/public/beagleplay/images/Image
Resolving git.beagleboard.org (git.beagleboard.org)... 44.226.162.25
Connecting to git.beagleboard.org (git.beagleboard.org)|44.226.162.25|:443...
↪ connected.
HTTP request sent, awaiting response... 200 OK
Length: 32172544 (31M) [application/octet-stream]
Saving to: 'Image'

Image               100%[===================>]  30.68M  1.78MB/s    in 18s
```

(continues on next page)

(continued from previous page)

```
2023-12-19 22:18:13 (1.74 MB/s) - 'Image' saved [32172544/32172544]

debian@BeaglePlay:~$ sudo cp Image /boot/firmware/Image-buildroot
[sudo] password for debian:
debian@BeaglePlay:~$ wget https://git.beagleboard.org/beagleboard/buildroot/-/
↪jobs/19194/artifacts/raw/public/beagleplay/images/rootfs.cpio.gz
--2023-12-19 22:16:44--  https://git.beagleboard.org/beagleboard/buildroot/-/
↪jobs/19194/artifacts/raw/public/beagleplay/images/rootfs.cpio.gz
Resolving git.beagleboard.org (git.beagleboard.org)... 44.226.162.25
Connecting to git.beagleboard.org (git.beagleboard.org)|44.226.162.25|:443...
↪ connected.
HTTP request sent, awaiting response... 200 OK
Length: 30111086 (29M) [application/octet-stream]
Saving to: 'rootfs.cpio.gz'

rootfs.cpio.gz      100%[===================>]  28.72M  21.5MB/s    in 1.3s

2023-12-19 22:16:46 (21.5 MB/s) - 'rootfs.cpio.gz' saved [30111086/30111086]

debian@BeaglePlay:~$ sudo cp rootfs.cpio.gz /boot/firmware/rootfs.cpio.gz-
↪buildroot
```

Listing 5.5: Modified /boot/firmware/extlinux/extlinux.conf file

```
menu title Select image to boot
timeout 10
default Buildroot

label Debian
    kernel /Image
    append root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait net.ifnames=0␣
↪quiet
    fdtdir /
    #fdtoverlays /overlays/<file>.dtbo
    initrd /initrd.img

label Buildroot
    kernel /Image-buildroot
    append rootwait net.ifnames=0 quiet
    fdtdir /
    initrd /rootfs.cpio.gz-buildroot
```

Listing 5.6: Reboot into modified kernel and rootfs

```
debian@BeaglePlay:~$ sudo shutdown -r now
Connection to 192.168.0.117 closed by remote host.
Connection to 192.168.0.117 closed.
jkridner@slotcar:~$ sudo nmap -n -p 22 192.168.0.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-19 17:32 EST
...
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 50:3E:AA:AD:78:06 (TP-Link Technologies)

Nmap scan report for 192.168.0.112
Host is up (0.00020s latency).
...
jkridner@slotcar:~$ ssh root@192.168.0.112
The authenticity of host '192.168.0.112 (192.168.0.112)' can't be␣
↪established.
ED25519 key fingerprint is␣
```

(continues on next page)

(continued from previous page)

```
↪SHA256:EZdvLkCNMyhy4RhvseUSC5EwaJR5Kgpk8JZG9kF+pmk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.112' (ED25519) to the list of known␣
↪hosts.
root@192.168.0.112's password:
# uname -a
Linux BeaglePlay 6.6.3 #1 SMP Tue Dec 19 21:32:06 UTC 2023 aarch64 GNU/Linux
# cat /etc/os-release
NAME=Buildroot
VERSION=2023.11-beagle1
ID=buildroot
VERSION_ID=2023.11
PRETTY_NAME="Buildroot 2023.11"
```

5.12.2 Booting U-Boot

Listing 5.7: Install bootloader to eMMC

```
#!/bin/bash

if ! id | grep -q root; then
        echo "must be run as root"
        exit
fi

wdir="/opt/u-boot/bb-u-boot-beagleplay"

if [ -b /dev/mmcblk0 ] ; then
        #mmc extcsd read /dev/mmcblk0
        mmc bootpart enable 1 2 /dev/mmcblk0
        mmc bootbus set single_backward x1 x8 /dev/mmcblk0
        mmc hwreset enable /dev/mmcblk0

        echo "Clearing eMMC boot0"

        echo '0' >> /sys/class/block/mmcblk0boot0/force_ro

        echo "dd if=/dev/zero of=/dev/mmcblk0boot0 count=32 bs=128k"
        dd if=/dev/zero of=/dev/mmcblk0boot0 count=32 bs=128k

        echo "dd if=${wdir}/tiboot3.bin of=/dev/mmcblk0boot0 bs=128k"
        dd if=${wdir}/tiboot3.bin of=/dev/mmcblk0boot0 bs=128k
fi
```

install-emmc.sh

5.13 Smart energy efficient video doorbell

1. Intelligent camera streaming and recording at 640x480 resolution and 30 FPS with power saving.
2. Detect user activity using an external button/sensor and configure it as a wake-up source
3. Application should start streaming on wakeup event, pause on system suspend and resume back seamlessly thus saving power while system was in suspended state.

Give options to enable below functionalities:

- Stream Live camera feed when visitor activity is detected

Fig. 5.18: https://youtu.be/4jbOXl_o4uo

- On-the-fly recording of live camera feed with a timeout to record visitor activity
- On-the-fly streaming of live camera feed to remote server for post processing/storage or display.

5.13.1 About deep sleep

Deep Sleep AKA Suspend-to-RAM is a low-power mode that allows an embedded device to retain its state in RAM while the processor is turned off. This can save a significant amount of power, especially in devices that are battery-powered.

The benefits of using deep sleep in embedded devices are faster wake-up time and better efficiency.

Tip: Checkout kernel docs on power states

5.13.2 Hardware requirements

1. BeaglePlay board
2. A CSI MIPI camera like TEVI-OV5640 or a USB web-cam
3. HDMI monitor & HDMI cable
4. Ethernet cable and a laptop/desktop with an Ethernet port
5. A Grove PIR sensor or a Grove button

5.13.3 Software requirements

First, make sure that you have the latest U-Boot which packages the right firmwares to make deep sleep work on beagleplay. You will also need to use ti-linux-kernel for basic suspend-to-RAM because the patches are yet to make it into upstream.

You can always use Robert Nelson's latest default debian images which should come with the right uboot and kernel required.

On debian, you may also need to make sure you have gstreamer installed, refer to https://gstreamer.freedesktop.org/documentation/installing/on-linux.html?gi-language=c for further details on how to install gstreamer.

5.13.4 Devicetree changes

You will need to tell Linux what your wakeup source is going to be, it can be a simple button or even a PIR sensor. To do this you'll need to make the following changes to the k3-am625-beagleplay.dts:

```
diff --git a/arch/arm64/boot/dts/ti/k3-am625-beagleplay.dts b/arch/arm64/boot/
↪dts/ti/k3-am625-beagleplay.dts
index b3328ae24b5f..9a83102e3604 100644
--- a/arch/arm64/boot/dts/ti/k3-am625-beagleplay.dts
+++ b/arch/arm64/boot/dts/ti/k3-am625-beagleplay.dts
@@ -166,6 +166,20 @@ vdd_sd_dv: regulator-5 {
                        <3300000 0x1>;
        };

+       motion_gpio_key {
+               compatible = "gpio-keys";
+               autorepeat;
+               pinctrl-names = "default";
+               pinctrl-0 = <&grove_pins_default>;
+               switch {
+                       label = "senseGPIO";
+                       linux,code = <KEY_WAKEUP>;
+                       interrupts-extended = <&main_gpio1 28 IRQ_TYPE_EDGE_
↪RISING>,
+                                             <&main_pmx0 0x1e8>;
+                       interrupt-names = "irq", "wakeup";
+               };
+       };
+
        leds {
                compatible = "gpio-leds";
```

The above will help us configure the grove connector's GPIO to act as a wakeup source from Deep Sleep.

If using the CSI MIPI camera like tevi-ov5640 then, be sure to also apply the respective overlay, for tevi-ov5640 apply `k3-am625-beagleplay-csi2-tevi-ov5640.dtbo` overlay.

The Technexion TEVI-OV5640 module supports Suspend-to-RAM but may fail to set the sensor registers in time when built as a module. You can fix this by making it a part of the kernel image: Find further details in the TI-SDK Documentation

Todo: Add the below changes to the beagle defconfig

```
diff --git a/arch/arm64/configs/defconfig b/arch/arm64/configs/defconfig
index 1f402994efed..0f081e5f96c1 100644
--- a/arch/arm64/configs/defconfig
+++ b/arch/arm64/configs/defconfig
@@ -739,14 +739,14 @@ CONFIG_RC_DECODERS=y
 CONFIG_RC_DEVICES=y
 CONFIG_IR_MESON=m
 CONFIG_IR_SUNXI=m
-CONFIG_MEDIA_SUPPORT=m
+CONFIG_MEDIA_SUPPORT=y
```

(continues on next page)

(continued from previous page)

```
 # CONFIG_DVB_NET is not set
 CONFIG_MEDIA_USB_SUPPORT=y
 CONFIG_USB_VIDEO_CLASS=m
 CONFIG_V4L_PLATFORM_DRIVERS=y
 CONFIG_SDR_PLATFORM_DRIVERS=y
 CONFIG_V4L_MEM2MEM_DRIVERS=y
-CONFIG_VIDEO_CADENCE_CSI2RX=m
+CONFIG_VIDEO_CADENCE_CSI2RX=y
 CONFIG_VIDEO_WAVE_VPU=m
 CONFIG_VIDEO_IMG_VXD_DEC=m
 CONFIG_VIDEO_IMG_VXE_ENC=m
@@ -764,12 +764,12 @@ CONFIG_VIDEO_SAMSUNG_EXYNOS_GSC=m
 CONFIG_VIDEO_SAMSUNG_S5P_JPEG=m
 CONFIG_VIDEO_SAMSUNG_S5P_MFC=m
 CONFIG_VIDEO_SUN6I_CSI=m
-CONFIG_VIDEO_TI_J721E_CSI2RX=m
+CONFIG_VIDEO_TI_J721E_CSI2RX=y
 CONFIG_VIDEO_HANTRO=m
 CONFIG_VIDEO_IMX219=m
 CONFIG_VIDEO_IMX390=m
 CONFIG_VIDEO_OV2312=m
-CONFIG_VIDEO_OV5640=m
+CONFIG_VIDEO_OV5640=y
 CONFIG_VIDEO_OV5645=m
 CONFIG_VIDEO_DS90UB953=m
 CONFIG_VIDEO_DS90UB960=m
@@ -1309,8 +1309,8 @@ CONFIG_PHY_XGENE=y
 CONFIG_PHY_CAN_TRANSCEIVER=m
 CONFIG_PHY_SUN4I_USB=y
 CONFIG_PHY_CADENCE_TORRENT=y
-CONFIG_PHY_CADENCE_DPHY=m
-CONFIG_PHY_CADENCE_DPHY_RX=m
+CONFIG_PHY_CADENCE_DPHY=y
+CONFIG_PHY_CADENCE_DPHY_RX=y
 CONFIG_PHY_CADENCE_SIERRA=y
 CONFIG_PHY_MIXEL_MIPI_DPHY=m
 CONFIG_PHY_FSL_IMX8M_PCIE=y
```

5.13.5 Linux commands

Once your hardware, software and devicetree changes are all set, and you boot till linux prompt we can finally start with the final bit. The below section describes various gstreamer pipelines created using sample gst-launch-1.0 gstreamer application. You can create your own gstreamer application too with some dynamic features, customized options taking referece from these pipelines.

Note: If using CSI based TEVI-OV5640 module, you need to also set the mediagraph prior to using camera. You can set set the media graph and sanity test the camera using below command which uses cam tool from libcamera:

```
cam -c1 --stream width=640,height=480,pixelformat=UYVY -C20
```

Additionally, if using a different camera, you can check the supported resolutions and video formats using below command:

```
v4l2-ctl --all -d /dev/videoX (where X is your video node number e.g. /dev/
↪video0)
```

There are two sets of gstreamer pipelines that get run in this demo one at server side i.e. on beagleplay board

directly which captures and displays the camera feed and streams it to the remote or client side, and the other at client side itself which receives the camera feed, records it, decodes it and displays it using the remote host machine.

Server side gstreamer pipeline (on beagleplay board):

Here, you can run either of the below two sets of gstreamer pipeline depending upon your requirement :

Display live camera feed

Pipeline topology

```
v4l2src --> kmssink
```

Gstreamer Pipeline

```
#Stop weston if using kmssink
systemctl stop weston.service
gst-launch-1.0 -v v4l2src io-mode=dmabuf device="/dev/video0" ! video/x-raw,␣
↪width=640, height=480, format=YUY2 ! kmssink       driver-name=tidss force-
↪modesetting=true sync=false async=false
```

Note: Change the video format to UYVY if using CSI based ov5640 camera

Description The Linux kernel uses V4L2 based driver for Camera and DRM/KMS based driver for Display, Gstreamer has v4l2src element to communicate with V4l2's based driver and kmssink element to talk with display driver and using above command, we can create a media pipeline which shares video buffers from camera to display using DMA to transfer the buffer. This is specified using io-mode property of v4l2src. By default display server i.e weston is in charge of controlling the display, so it needs to be disabled if we want to control the display directly. We also use kmssink's force-modesetting property to set the display mode to the camera resolution and have a full screen display. If using a graphics server involving GPU, one can use waylandsink (which uses weston as display server), glimagesink (which uses opengl API) or ximagesink (which uses Xorg as display server) depending upon the display server.

Display live camera feed + Stream out to remote server

Pipeline topology

```
                  .-->kmssink
v4l2src --> tee--|
                  .-->x264enc-->rtph264pay-->udpsink
```

Gstreamer pipeline

```
#Stop weston if using kmssink
systemctl stop weston.service
gst-launch-1.0 -v v4l2src io-mode=dmabuf device="/dev/video0" ! video/x-raw,␣
↪width=640, height=480, format=YUY2 ! queue ! tee name=t t. ! queue !␣
↪kmssink driver-name=tidss force-modesetting=true sync=false async=false t.␣
↪! queue ! ticolorconvert ! queue ! x264enc speed-preset=superfast  key-int-
↪max=30 tune=zerolatency bitrate=25000 ! queue ! rtph264pay config-
↪interval=30 ! udpsink sync=false port=5000 host=192.168.0.2 async=false &
```

Note: Change the video format to UYVY if using CSI based ov5640 camera

Description Here we use gstreamer's tee element to split the media pipeline graph into two arms, one arm is used to display the camera feed on-screen (which is same as the one described in previous section) and other arm is used to encode the camera feed and stream it to remote server. We use libx264 based x264enc element to encode the raw video to H.264 based access units. However x264enc does not support YUY2 video format, so we use ticolorconvert element to convert the video format to the one supported by the encoder, this element is CPU based but it uses ARM neon based instructions underneath for faster conversion. The x264enc element also offers different parameters to fine tune the encoding. We use superfast speed preset along with zerolatency tuning as we want to strem in realtime with minimal latency. We set IDR or key frame interval to 30 frames using key-int-max property. The IDR frame is important from streaming point of view as it marks arrival of fresh group of pictures without any dependencies to previous frames so that decoding at client side can resume back seamlessly even if there were packet losses in between due to network issues. However the value needs to be carefully chosen as the trade-off with higher frequency of IDR frames though is the increase in size of rtp payload which may consume more bandwidth. The video quality of encoded stream is controleld by bitrate parameter which specifies number of Kbits used for encoding video for 1s. Higher value for bitrate will increase the video quality albeit at the cost of increased size. The encoded frame is then packetized into RTP packets using rtph264pay element and transmitted over network using UDP protocol using udpsink element. The ip address and port number of remote host are specified using "host" and "port" property of udpsink element respectively.

This gstreamer pipeline is useful for prototyping use-case where you not just want to display the camera feed outside the door when some visitor comes, but also want to stream out to a remote server (for e.g. security control rool or to your mobile device) for more safety.

Display live camera feed + Stream out to remote server+ Record camera feed

Pipeline topology

```
                  .-->kmssink
v4l2src --> tee--|                  .--filesink
                  .-->x264enc-->tee--|
                                     .--rtph264pay-->udpsink
```

Gstreamer pipeline

```
#Stop weston if using kmssink
systemctl stop weston.service
gst-launch-1.0 -v v4l2src io-mode=dmabuf device="/dev/video0" ! video/x-raw,␣
↪width=640, height=480, format=YUY2 ! queue ! tee name=t t. ! queue !␣
↪kmssink driver-name=tidss force-modesetting=true sync=false async=false t.␣
↪! queue ! ticolorconvert ! x264enc speed-preset=superfast key-int-max=30␣
↪bitrate=5000 ! queue ! tee name=t1 t1. ! queue ! rtph264pay config-
↪interval=60 ! udpsink port=5000 host=192.168.0.2 sync=false async=false t1.
↪ ! queue ! filesink location="op.h264"
```

Note: Change the video format to UYVY if using CSI based ov5640 camera

Description In addition to the media topology described in previous section, one more tee element is added here to save the encoded file over user-specified storage media. This could be helpful to have the camera feed of all the visitors (or potential intruders :)) recorded at the device end itself for future reference/analysis or as a blackbox recording. However care needs to be taken to constantly backup the recorded stream so that storage media does not run out of space.

Client side gstreamer pipeline (runs on remote host):

The previous section described how the camera feed is displayed and streamed out to remote server using RTP and UDP protocols. Here we will discuss about how we can receive the transmitted stream and display it or record it. We use X86_64 based Ubuntu machine as remote host here.

Display camera feed received over network

Pipeline topology

```
udpsrc --> rtpjitterbuffer-->rtph264depay-->h264parse-->avdec_h264-->
↪xvimagesink
```

Gstreamer pipeline

```
# This is the IP address of the remote host which is specified in the server␣
↪pipelien running on beagleplay
sudo ifconfig enp2s0 192.168.0.2
gst-launch-1.0 udpsrc port=5000 caps = "application/x-rtp,␣
↪media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264,␣
↪payload=(int)96" ! rtpjitterbuffer latency=50 ! rtph264depay ! h264parse !␣
↪avdec_h264 ! queue ! fpsdisplaysink text-overlay=false name=fpssink video-
↪sink="xvimagesink sync=false" sync=false -v
```

Description The above gstreamer pipeline uses udpsrc element which reads UDP packets from the network, listening on the specified port (5000) and provide RTP packets to downstream element. rtpjitterbuffer element is used to buffer the incoming RTP packets to help reduce the impact of network jitter on smoothness of video. The bufferring is set to 50ms using latency property of rtpjitterbuffer, the value should be chosen optimally as tradeoff of choosing higher value is protection against network jitter maintaining the smoothness of pipeline but a higher value also increases the glass-to-glass latency. rtph264depay element is used to depacketize the H264 payload from RTP packets and feed send it to h264parse which parses it and provides access unit-by-access unit byte-stream to avdec_h264 which is a libav based software decoding element to decode H264 stream to raw video. fpsdisplaysink element is used along with xvimagesink (X11 backend) as video-sink to display overall frame-rate of the pipeline. If using weston as display server then waylandsink should be used as video-sink instead.

Display camera feed received over network + record incoming stream

Pipeline topology

```
                                                                    .-->avdec_h264--
↪>xvimagesink
udpsrc --> rtpjitterbuffer-->rtph264depay-->h264parse-->tee--|
                                                                    .-->filesink
```

Gstreamer pipeline

```
# This is the IP address of the remote host which is specified in the server␣
↪pipelien running on beagleplay
sudo ifconfig enp2s0 192.168.0.2
gst-launch-1.0 udpsrc port=5000 caps = "application/x-rtp,␣
↪media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264,␣
↪payload=(int)96" ! rtpjitterbuffer latency=50 ! rtph264depay ! h264parse !␣
↪video/x-h264, stream-format=byte-stream ! tee name=t t. ! queue ! filesink␣
↪location="op.h264"  t. ! queue ! avdec_h264 ! queue ! fpsdisplaysink text-
↪overlay=false name=fpssink video-sink="xvimagesink sync=false" sync=false -
↪v
```

Description This is same as pipeline described in previous section albeit with the extra addition of tee element which adds another arm to save the decoded video over a file on the host machine.

2. Let the above pipelines run in the background and then to suspend the device (beagleplay):

```
echo mem > /sys/power/state
```

3. Then, if you press the button/ trigger PIR sensor with some movement it should bring the device back up and you will see the video resume almost instantly on both the server side and client side. This is because underlying software stack also involving video and display related drivers support system suspend/resume, thus helping the application to resume seamlessly.

4. Additionally, you can enable auto suspend for the device by using a simple systemd service. Follow the guide here to see how to create and enable a script as a systemd service. The script that I used was as follows:

```
#!/bin/bash

while true
do
 sleep 15       # Adjust this time to whatever delay you prefer the device␣
↪stays on after resume
 echo "Entering Suspend to RAM..."
 echo mem > /sys/power/state
done
```

5.13.6 Resources

1. https://software-dl.ti.com/processor-sdk-linux/esd/AM62X/09_02_01_09/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/Power_Management/pm_low_power_modes.html#deep-sleep

2. https://software-dl.ti.com/processor-sdk-linux/esd/AM62X/09_02_01_09/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/Camera/CSI2RX.html

Chapter 6

Support

All support for BeaglePlay design is through BeagleBoard.org community at BeagleBoard.org forum.

6.1 Production board boot media

- BeaglePlay Rev A2

6.2 Certifications and export control

6.2.1 Export designations

- HS: 8471504090
- US HS: 8473301180
- EU HS: 8471707000

6.2.2 Size and weight

- Bare board dimensions: 82.5 x 80 x 20 mm
- Bare board weight: 55.3 g
- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 125.3 g

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 BeaglePlay design repository.

6.3.2 Software docs

For BeaglePlay specific software projects you can checkout all the BeaglePlay 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/play

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://git.beagleboard.org/beagleplay/beagleplay. Versions released into production are noted below.

Table 6.1: BeaglePlay board change history

Rev	Changes	Date	By
A2	Initial production version	2023-03-08	JK