BeagleBone AI-64

BeagleBoard.org Foundation
Jun 29, 2025

Table of contents

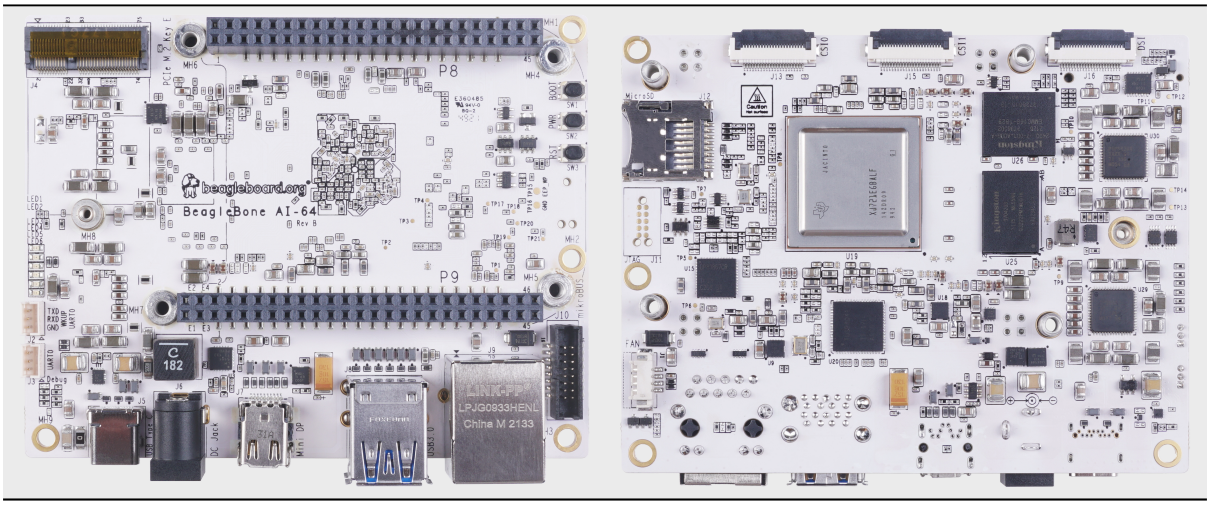
BeagleBone® AI-64 brings a complete system for developing artificial intelligence (AI) and machine learning solutions with the convenience and expandability of the BeagleBone® platform and the peripherals on board to get started right away learning and building applications. With locally hosted, ready-to-use, open-source focused tool chains and development environment, a simple web browser, power source and network connection are all that need to be added to start building performance-optimized embedded applications. Industry-leading expansion possibilities are enabled through familiar BeagleBone® cape headers, with hundreds of open-source hardware examples and dozens of readily available embedded expansion options available off-the-shelf.

Chapter 1

Introduction

BeagleBone AI-64 like its predecessor bbai-home, is designed to address the open-source community, early adopters, and anyone interested in a low cost 64-bit Dual Arm® Cortex®-A72 processor based Single Board Computer (SBC). It also offers access to many of the interfaces and allows for the use of add-on boards called capes, to add many different combinations of features. A user may also develop their own board or add their own circuitry.

Note: AI-64 has been equipped with a minimum set of features to allow the user to experience the power of the processor and is not intended as a full development platform as many of the features and interfaces supplied by the processor are not accessible from BeagleBone AI-64 via onboard support of some interfaces. It is not a complete product designed to do any particular function. It is a foundation for experimentation and learning how to program the processor and to access the peripherals by the creation of your own software and hardware.



1.1 BeagleBone Compatibility

The board is intended to provide functionality well beyond BeagleBone Black or BeagleBone AI, while still providing compatibility with BeagleBone Black's expansion headers as much as possible. There are several significant differences between the three designs.

Table 1.1: Table: BeagleBone Compatibility

Feature	AI-64	AI	Black
SoC	TDA4VM	AM5729	AM3358
Arm CPU	Cortex-A72 (64-bit)	Cortex-A15 (32-bit)	Cortex-A8 (32-bit)
Arm cores/MHz	2x 2GHz	2x 1.5GHz	1x 1GHz
RAM	4GB	1GB	512MB
eMMC flash	16GB	16GB	4GB
Size	4” x 3.1”	3.4” x 2.1”	.4” x 2.1”
Display	miniDP + DSI	microHDMI	microHDMI
USB host (Type-A)	2x 5Gbps	1x 480Mbps	1x 480Mbps
USB dual-role	Type-C 5Gbps	Type-C 5Gbps	mini-AB 480Mbps
Ethernet	10/100/1000M	10/100/1000M	10/100M
M.2	E-key	-	-
WiFi/ Bluetooth	-	AzureWave AW‑CM256SM	-

Todo: add cape compatibility details

1.2 BeagleBone AI-64 Features and Specification

This section covers the specifications and features of the board and provides a high level description of the major components and interfaces that make up the board.

Table 1.2: Table: BeagleBone AI-64 Features and Specification

	Feature
Processor	Texas Instruments TDA4VM
Graphics Engine	PowerVR® Series8XE GE8430
SDRAM Memory	LPDDR4 3.2GHz (4GB) Kingston Q3222PM1WDGTK-U
Onboard Flash	eMMC (16GB) Kingston EMMC16G-TB29-PZ90
PMIC	TPS65941213 and TPS65941111 PMICs regulator and one additional LDO.
Debug Support	**2x 3 pin 3.3V TTL header:** 1. WKUP_UART0: Wake-up domain serial port 2. UART0: Main domain serial port 10-pin JTAG TAG-CONNECT footprint
Power Source	USB C or DC Jack (5V @ >3A)
PCB	4” x 3.1”
Indicators	1x Power & 5x User Controllable LEDs
USB-3.0 Client Port	Access to USB0 SuperSpeed dual-role mode via USB-C (no power output)
USB-3.0 Host Port	TUSB8041 4-port SuperSpeed hub 1x on USB1, 2x Type A Socket up-to 2.8A total depending on power input
Ethernet	Gigabit RJ45 link indicator speed indicator
SD/MMC Connector	microSD (1.8/3.3V)
User Input	1. Reset Button 2. Boot Button 3. Power Button
Video Out	miniDP
Audio	via miniDP (stereo)
Weight	192gm (with heatsink)
Power	Refer to *Main Board Power* section

1.3 Board Component Locations

This section describes the key components on the board. It provides information on their location and function. Familiarize yourself with the various components on the board.

1.3.1 Board components

This section describes the key components on the board, their location and function.

Front components location

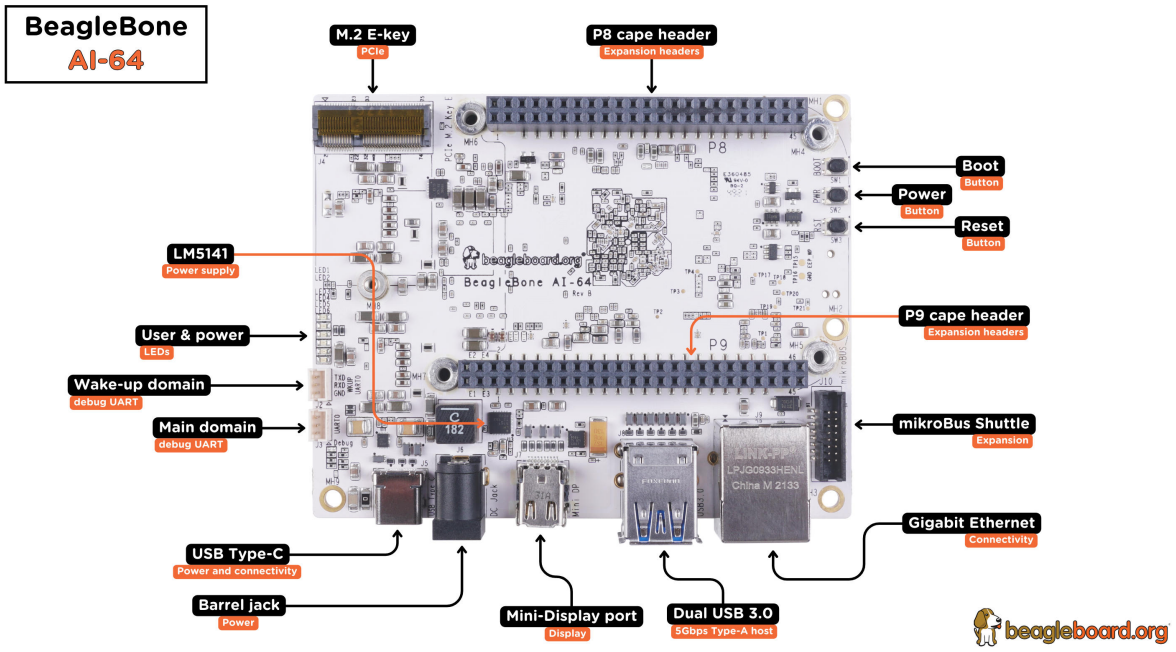


Fig. 1.1: BeagleBone AI-64 board front components location

Table 1.3: BeagleBone AI-64 board front components location

Feature	Description
User & power LEDs	USR0 - USR4 user LEDs & Power (Board ON) LED indicator
UART debug ports	3pin Wake-up domain and Main domain UART debug ports
USB C	Power, connectivity, and board flashing.
Barrel jack	Power input (accepts 5V power)
Mini-Display port	Output for Display/Monitor connection
Dual USB-A	5Gbps USB-A ports for peripherals (Wi-Fi, Bluetooth, Keyboard, etc)
GigaBit Ethernet	1Gb/s Wired internet connectivity
mikroBUS Shuttle	16pin mikroBUS Shuttle connector for interfacing mikroE click boards
P8 & P9 cape header	Expansion headers for BeagleBone capes.
Reset button	Press to reset BeagleBone AI-64 board (TDA4VM SoC)
Power button	Press to shut-down (OFF), hold down to boot (ON)
Boot button	Boot selection button (force to boot from microSD if power is cycled)
M.2 Key E	PCIE M.2 Key E connector

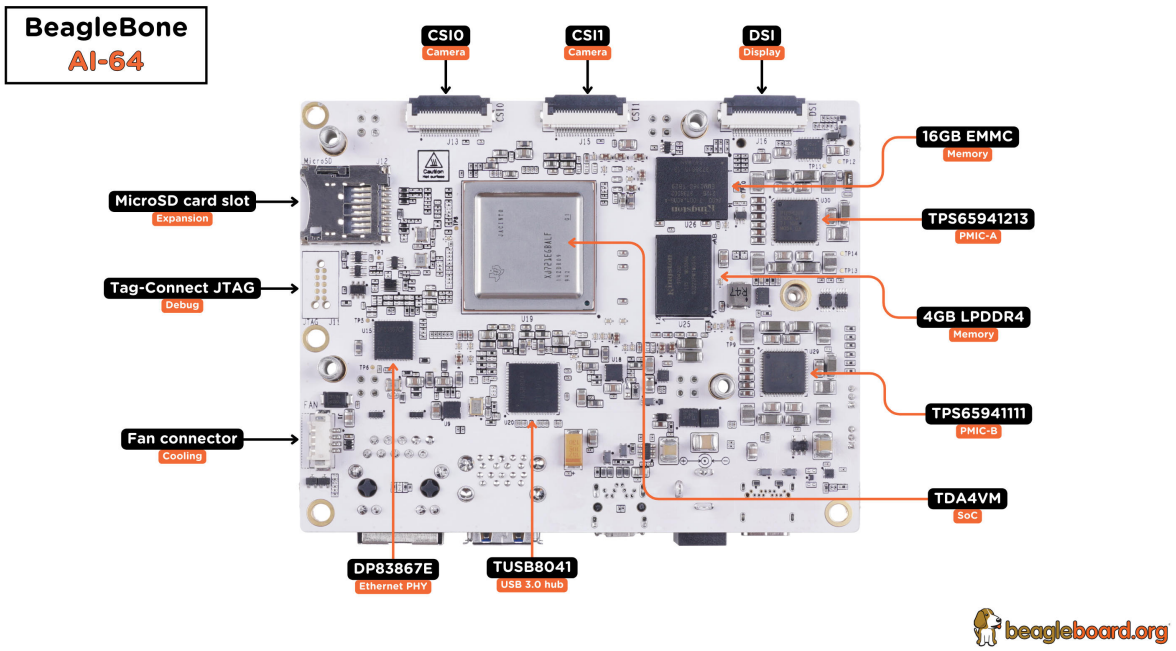


Fig. 1.2: BeagleBone AI-64 board back components location

Back components location

Table 1.4: BeagleBone AI-64 board back components location

Feature	Description
microSD	Micro SD Card holder
JTAG debug port	Tag-Connect JTAG (TDA4Vm) debug port
Fan connector	PWM controllable 4pin fan connector
DP83867E	Ethernet PHY
TUSB8041	USB 3.0 hub IC
TDA4VM	Dual Arm® Cortex®-A72 SoC and C7x DSP with deep-learning, vision and MMA
PMIC	Power management TPS65941213 (PMIC-A) & TPS65941111 (PMIC-B)
16GB eMMC	Flash storage
4GB RAM	4GB LPDDR4 RAM
DSI	MIPI Display connector
CSI0 & CSI1	MIPI Camera connectors

Chapter 2

Quick Start Guide

This section provides instructions on how to hook up your board. This Beagle requires a 5V > 3A (15W) power supply to work properly via either USB Type-C power adapter or a barrel jack power adapter.

Recommended adapters can be found at accessories-power-supplies section. All the *BeagleBone AI-64 connections ports* we will use in this chapter are shown in the figure below.

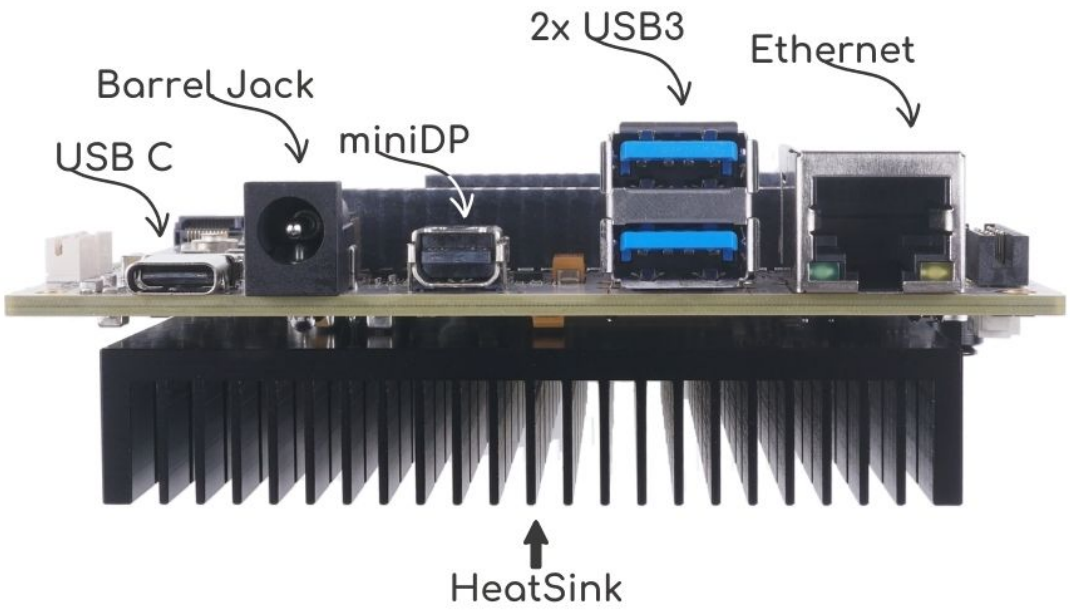


Fig. 2.1: BeagleBone AI-64 connections ports

2.1 What's In the Box

In the box you will find two main items as shown in *BeagleBone AI-64 box content*.

- BeagleBone AI-64
- Instruction card

Note: A USB-C to USB-C cable is not included, but recommended for the tethered scenario and creates a

developer experience where the board can be used immediately with no other equipment needed.

Tip: For board files, 3D model, and more, you can checkout BeagleBona AI-64 repository on OpenBeagle.

Fig. 2.2: BeagleBone AI-64 box content

2.2 Methods of operation

1. Tethered to a PC
2. Standalone development platform in a PC configuration using external peripherals

2.2.1 Main Connection Scenarios

This section describes how to connect and power the board and serves as a slightly more detailed description of the Quick Start Guide included in the box. The board can be configured in several different ways, but we will discuss the two most common scenarios.

- Tethered to a PC via the USB cable
  - `Board is accessed as a storage drive and virtual Ethernet connection.`
- Standalone Desktop
  - `Display`
  - `Keyboard and Mouse`
  - `External 5V > 3A power supply`

Each of these configurations is discussed in general terms in the following sections.

Tethered To A PC

In this configuration, the board is powered by the PC via a single USB cable. The board is accessed either as a USB storage drive or via the browser on the connected PC. You need to use either Firefox or Chrome on the PC, Internet Explorer will not work properly.

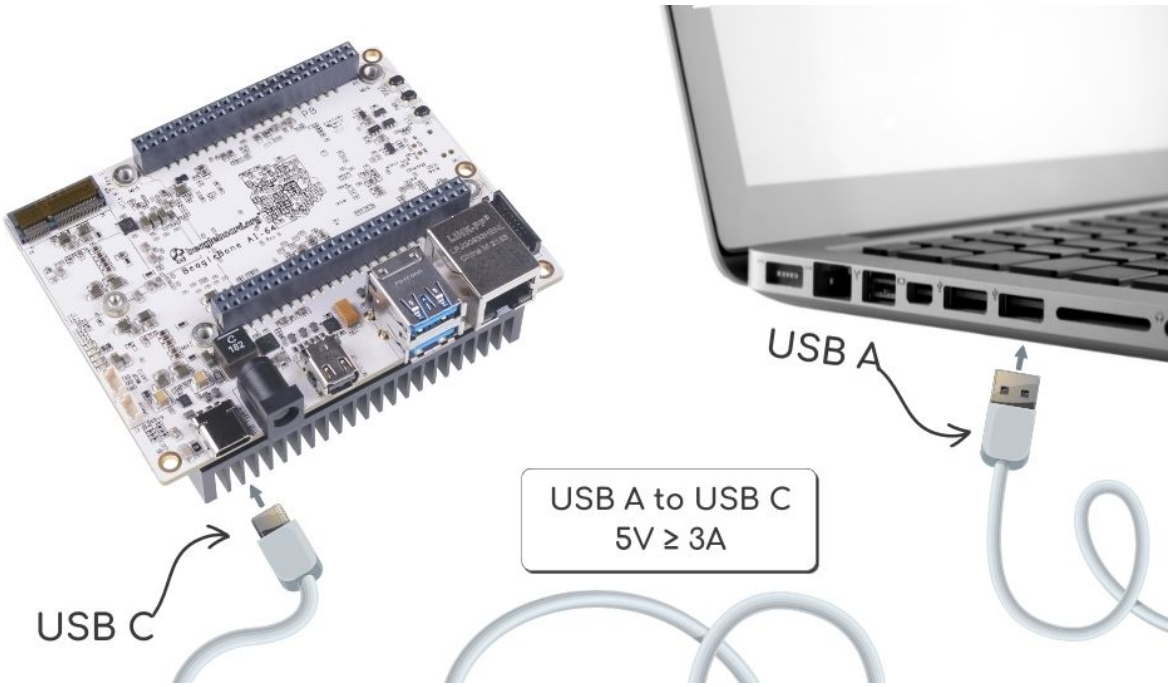


Fig. 2.3: Tethered Configuration

At least 5V @ 3A is required to power the board, In most cases the PC may not be able to supply sufficient power for the board unless the connection is made over a Type-C to Type-C cable. You should always use an external 5V > 3A DC power supply connected to the barrel jack if you are unsure that the system can provide the required power or are otherwise using a USB-A to Type-C cable which will always require power from the DC barrel jack.

Connect the Cable to the Board

1. Connect the type C USB cable to the board as shown in the figure below. The connector is on the top side of the board near barrel jack.
2. Connect the USB-A end of the cable to your PC or laptop USB port as shown in the figure below.
3. The board will power on and the power LED will be on as shown in the figure below.
4. When the board starts to the booting process started by the process of applying power, the LEDs will come on in sequence as shown in the figure below. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it begins to boot the Linux kernel.

Accessing the Board as a Storage Drive

The board will appear around a USB Storage drive on your PC after thekernel has booted, which will take a round 10 seconds. The kernel on the board needs to boot before the port gets enumerated. Once the board appears as a storage drive, do the following:

1. Open the USB Drive folder.
2. Click on the file named **start.htm**
3. The file will be opened by your browser on the PC and you should get a display showing the Quick Start Guide.

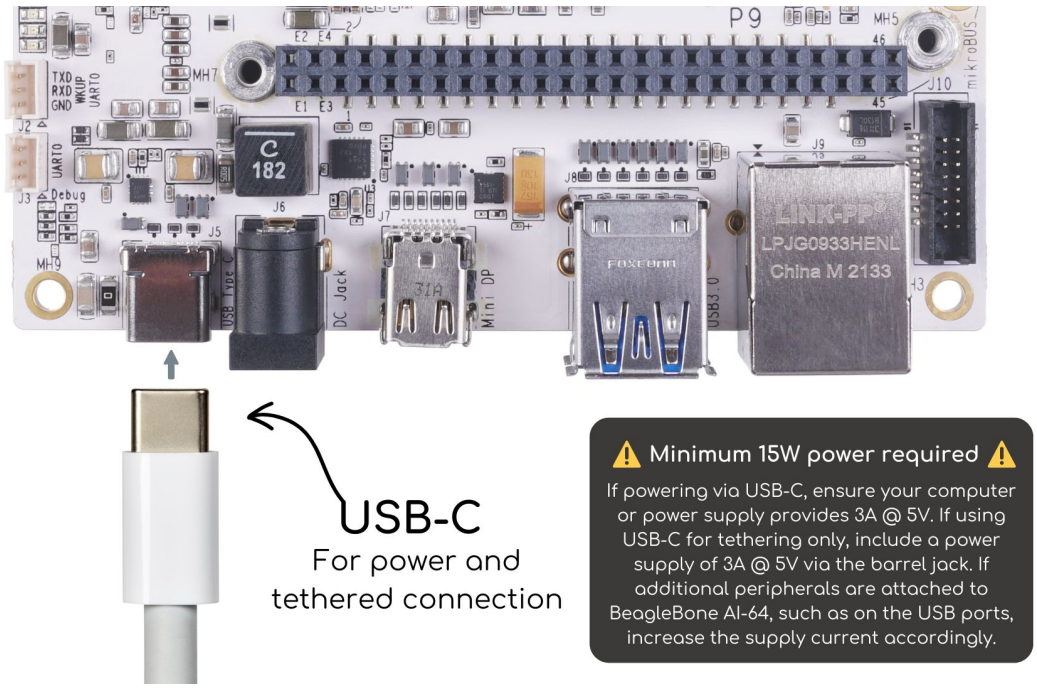


Fig. 2.4: USB Connection to the Board

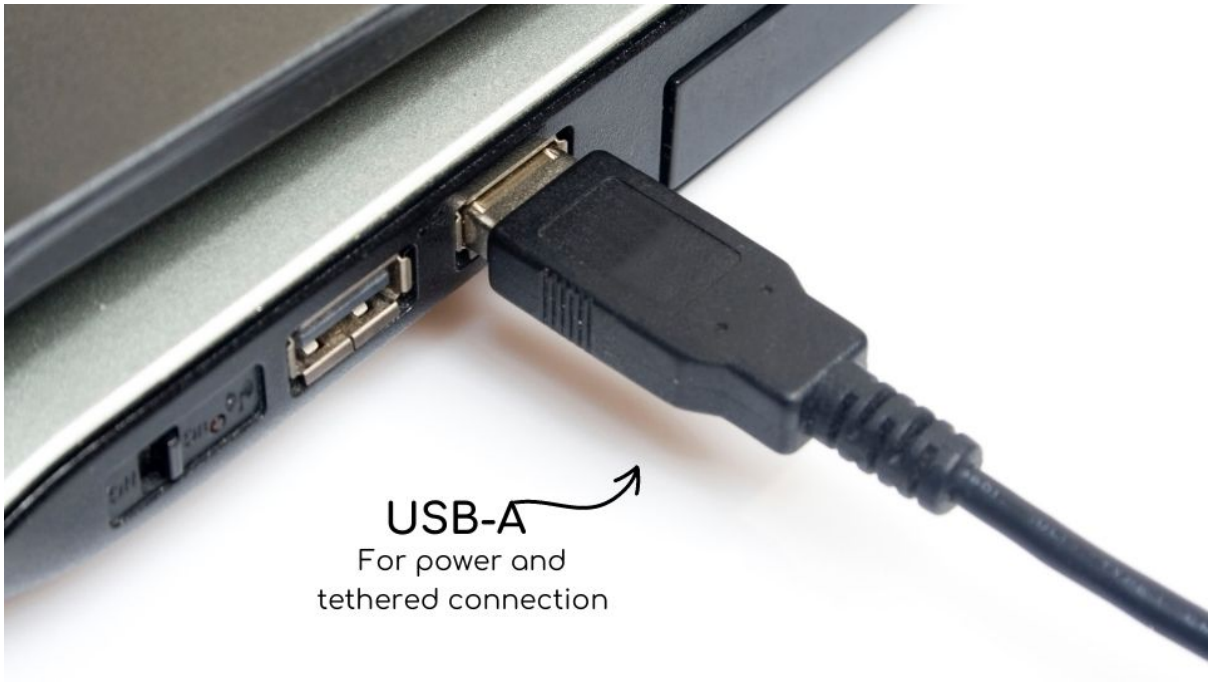


Fig. 2.5: USB Connection to the PC/Laptop

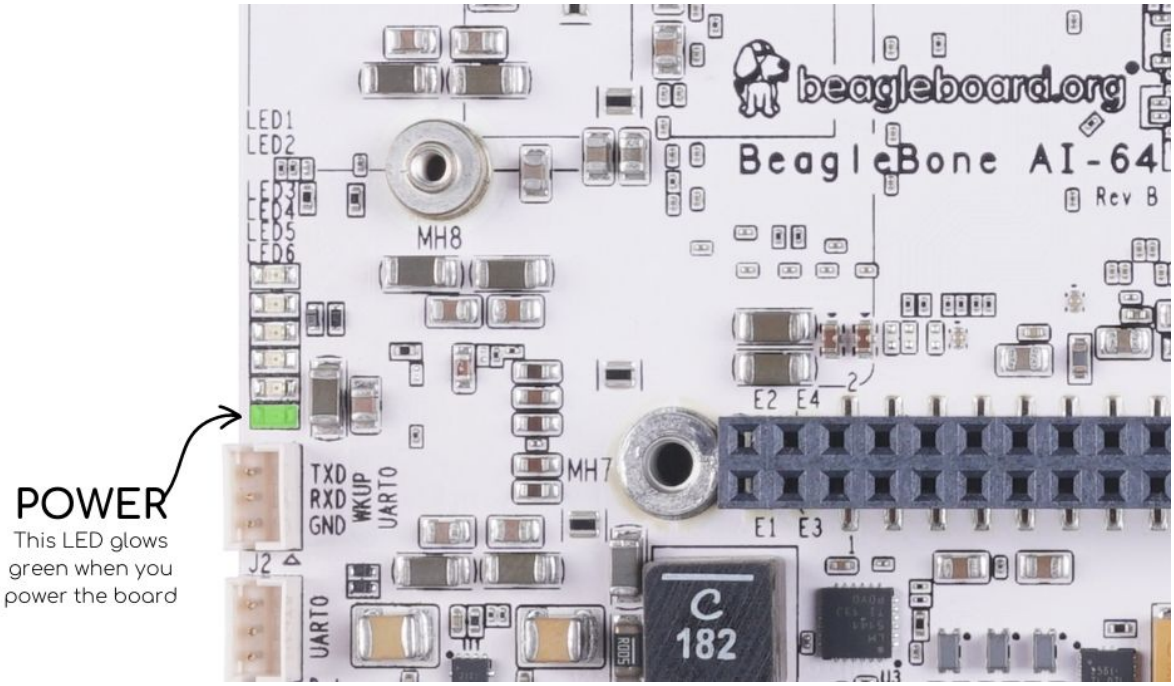


Fig. 2.6: Board Power LED

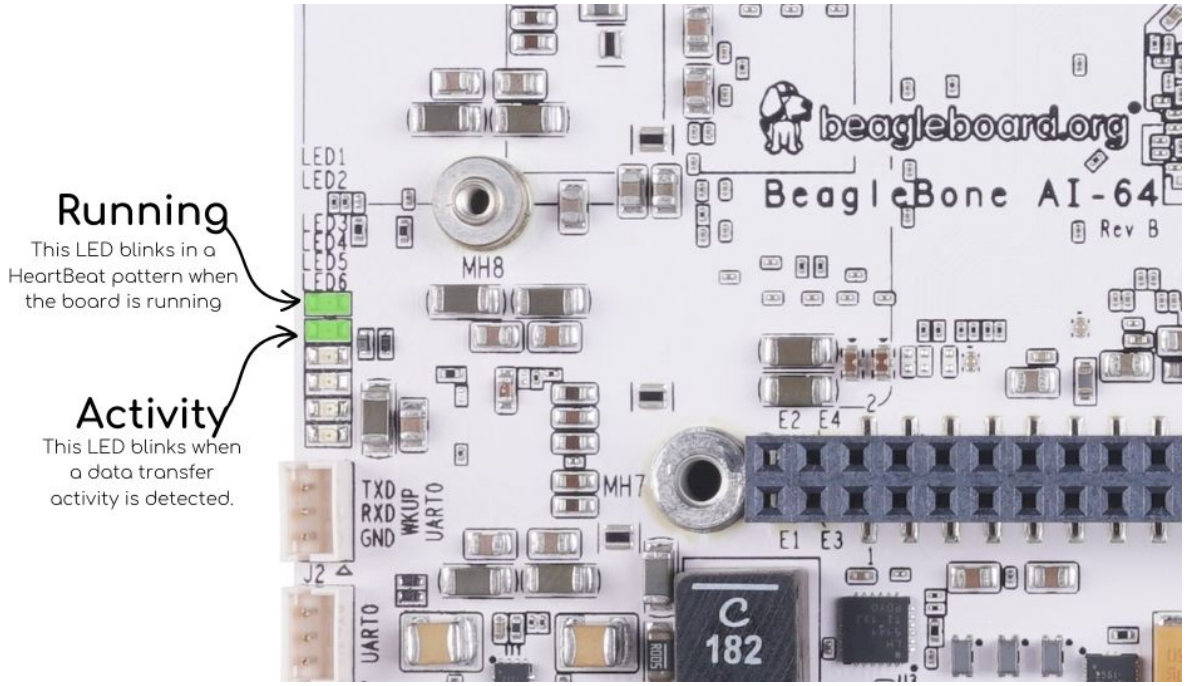


Fig. 2.7: Board Boot Status

4. Your board is now operational! Follow the instructions on your PC screen.

Standalone w/Display and Keyboard/Mouse

In this configuration, the board works more like a PC, totally free from any connection to a PC as shown in the figure below. It allows you to create your code to make the board do whatever you need it to do. It will however require certain common PC accessories. These accessories and instructions are described in the following section.

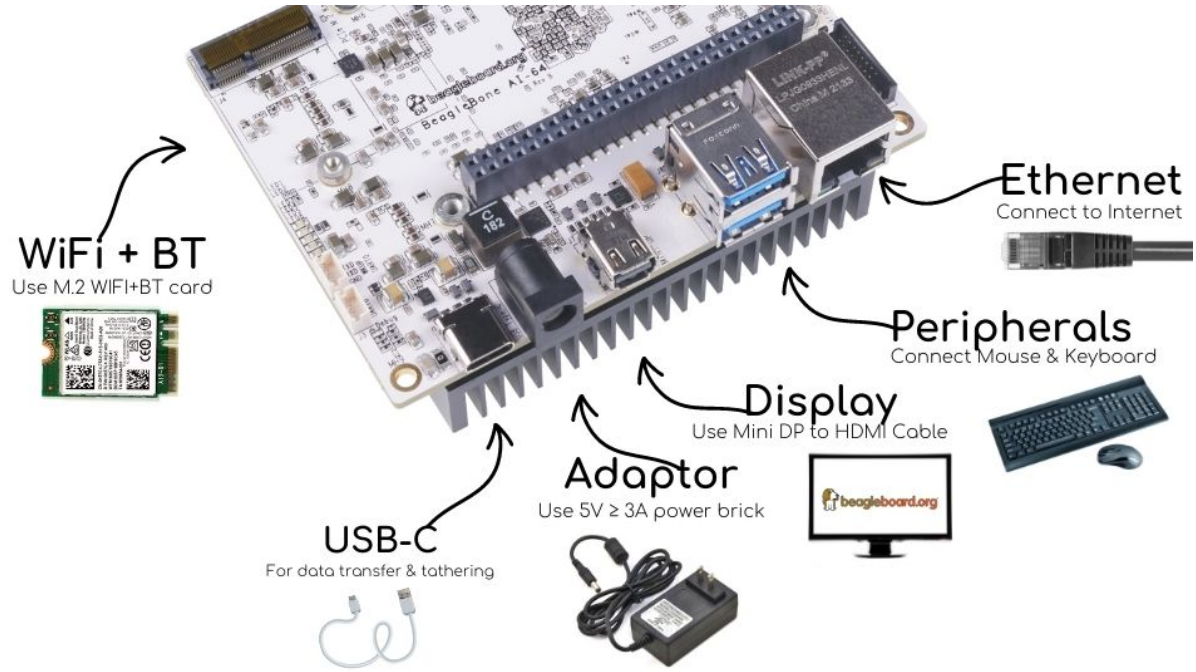


Fig. 2.8: Desktop Configuration

Ethernet cable and M.2 WiFi + Bluetooth card are optional. They can be used if network access required.

Required Accessories

In order to use the board in this configuration, you will need the following accessories:

- 5V > 3A power supply.
- Display Port or HDMI monitor.
- miniDP-DP or active miniDP-HDMI cable.
- USB wired/wireless keyboard and mouse.

Optional Accessories

- Powered USB hub, The board has only two USB Type-A host ports, so you may need to use a powered USB Hub if you wish to add additional USB devices, such as a USB WiFi adapter.
- M.2 Bluetooth & WiFi module, For wireless connections, a USB WiFi adapter or a recommended M.2 WiFi module can provide wireless networking.

Connecting Up the Board

1. Connect the miniDP to DP or active miniDP to HDMI cable from your BeagleBone AI-64 to your monitor.
1. If you have an Display Port or HDMI monitor with HDMI-HDMI or DP-DP cable you can use adapters as shown in the figure below.

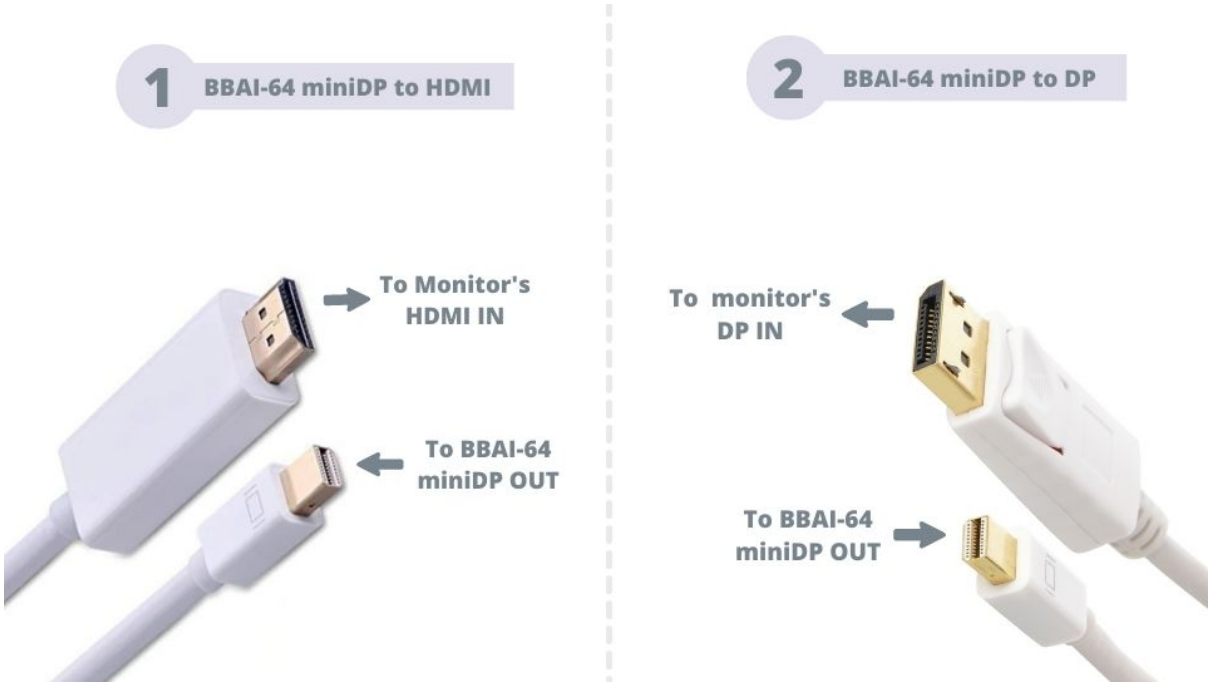


Fig. 2.9: Connect miniDP-DP or active miniDP-HDMI cable to BeagleBone AI-64

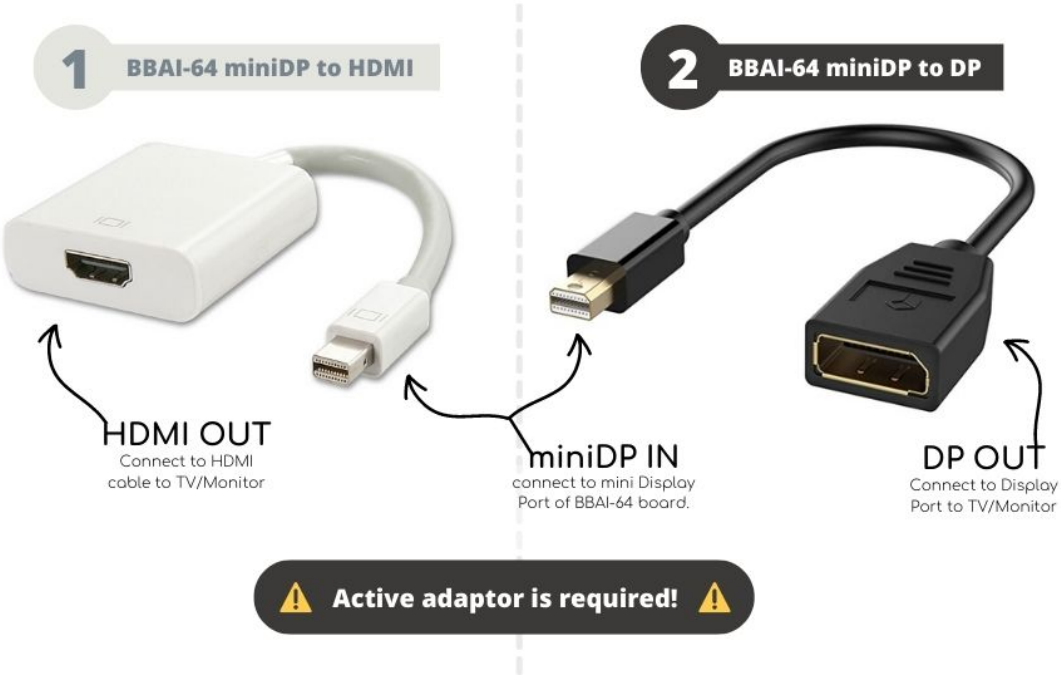


Fig. 2.10: Display adapters

1. If you have wired/wireless USB keyboard and mouse such as seen in the figure below, you need to plug the receiver in the USB host port of the board as shown in the figure below.

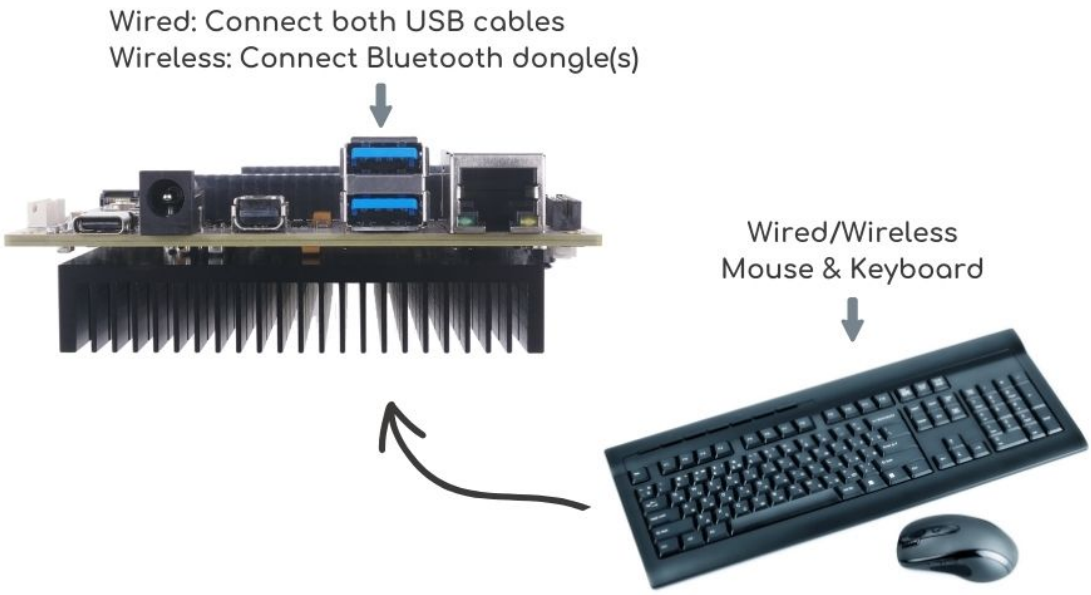


Fig. 2.11: Keyboard and Mouse

1. Connect the Ethernet Cable

If you decide you want to connect to your local area network, an Ethernet cable can be used. Connect the Ethernet Cable to the Ethernet port as shown in the figure below. Any standard 100M Ethernet cable should work.

1. The final step is to plug in the DC power supply to the DC power jack as shown in the figure below.

1. The cable needed to connect to your display is a miniDP-DP or active miniDP-HDMI. Connect the miniDP connector end to the board at this time. The connector is on the top side of the board as shown in the figure below.

The connector is fairly robust, but we suggest that you not use the cable as a leash for your Beagle. Take proper care not to put too much stress on the connector or cable.

1. Booting the Board

As soon as the power is applied to the board, it will start the booting up process. When the board starts to boot the LEDs will come on. It will take a few seconds for the status LEDs to come on, so be patient. The LEDs will be flashing in an erratic manner as it boots the Linux kernel.

While the four user LEDS can be over written and used as desired, they do have specific meanings in the image that is shipped with the board once the Linux kernel has booted.

- **USR0** is the heartbeat indicator from the Linux kernel.
- **USR1** turns on when the microSD card is being accessed
- **USR2** is an activity indicator. It turns on when the kernel is not in the idle loop.
- **USR3** turns on when the onboard eMMC is being accessed.
- **USR4** is an activity indicator for WiFi.

1. A Booted System

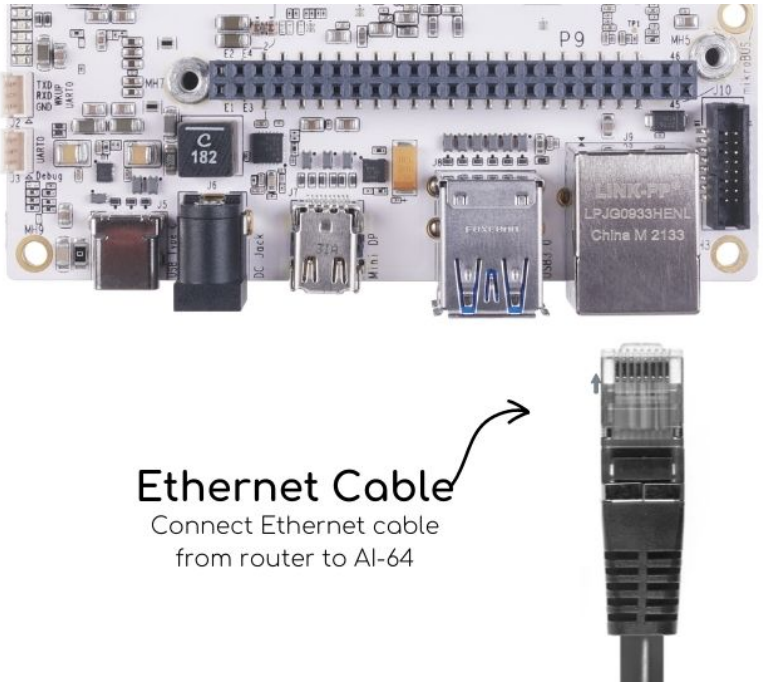


Fig. 2.12: Ethernet Cable Connection

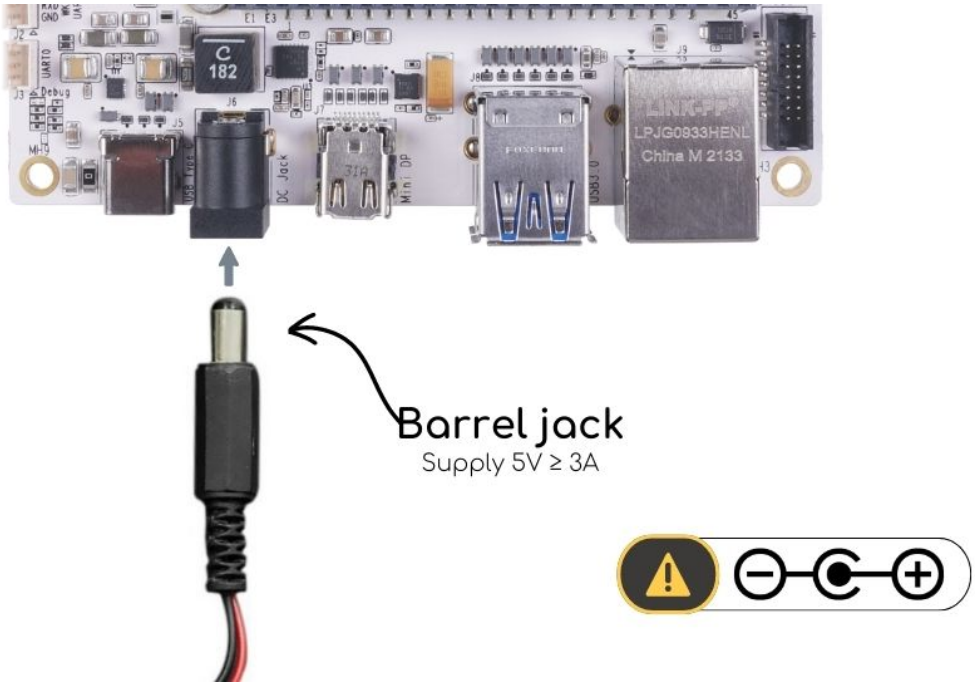


Fig. 2.13: External DC Power

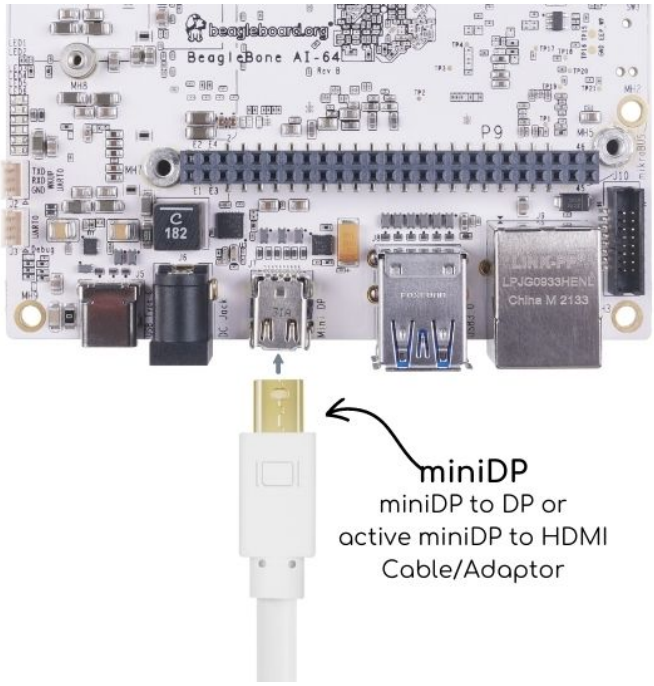


Fig. 2.14: Connect miniDP to DP or active miniDP to HDMI Cable to the Board

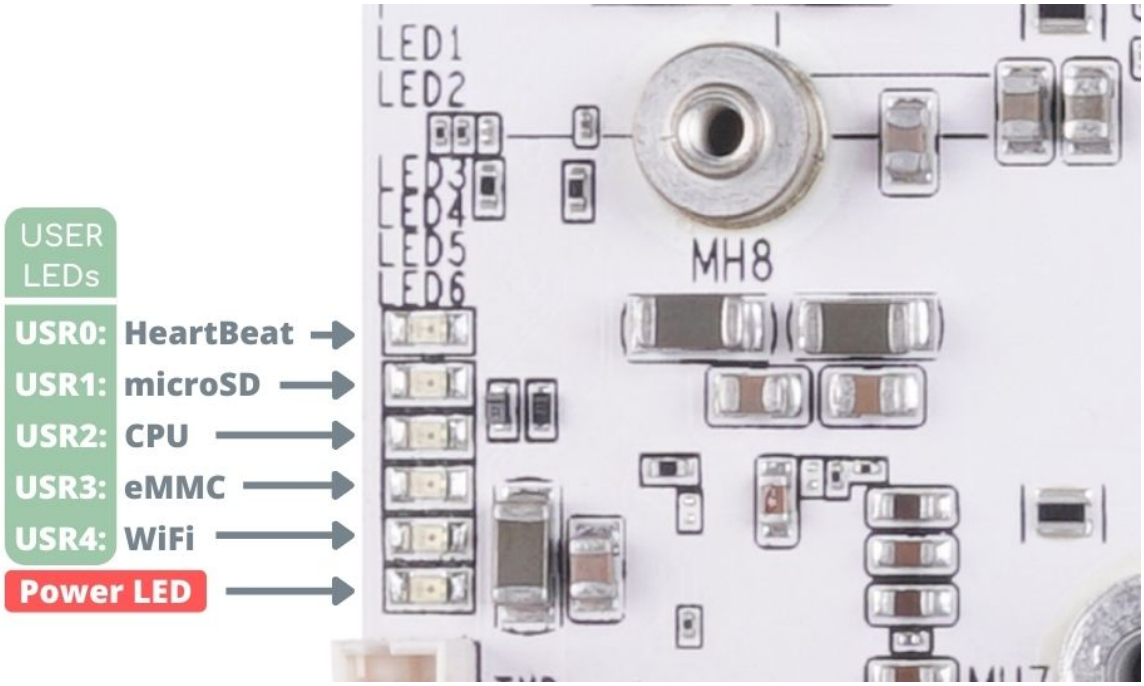


Fig. 2.15: BeagleBone AI-64 LEDs

a. The board will have a mouse pointer appear on the screen as it enters the Linux boot step. You may have to move the physical mouse to get the mouse pointer to appear. The system can come up in the suspend mode with the monitor in a sleep mode.

b. After a minute or two a login screen will appear. You do not have to do anything at this point.

c. After a minute or two the desktop will appear. It should be similar to the one shown in the figure below. HOWEVER, it will change from one release to the next, so do not expect your system to look exactly like the one in the figure, but it will be very similar.

d. And at this point you are ready to go! The figure below shows the desktop after booting.

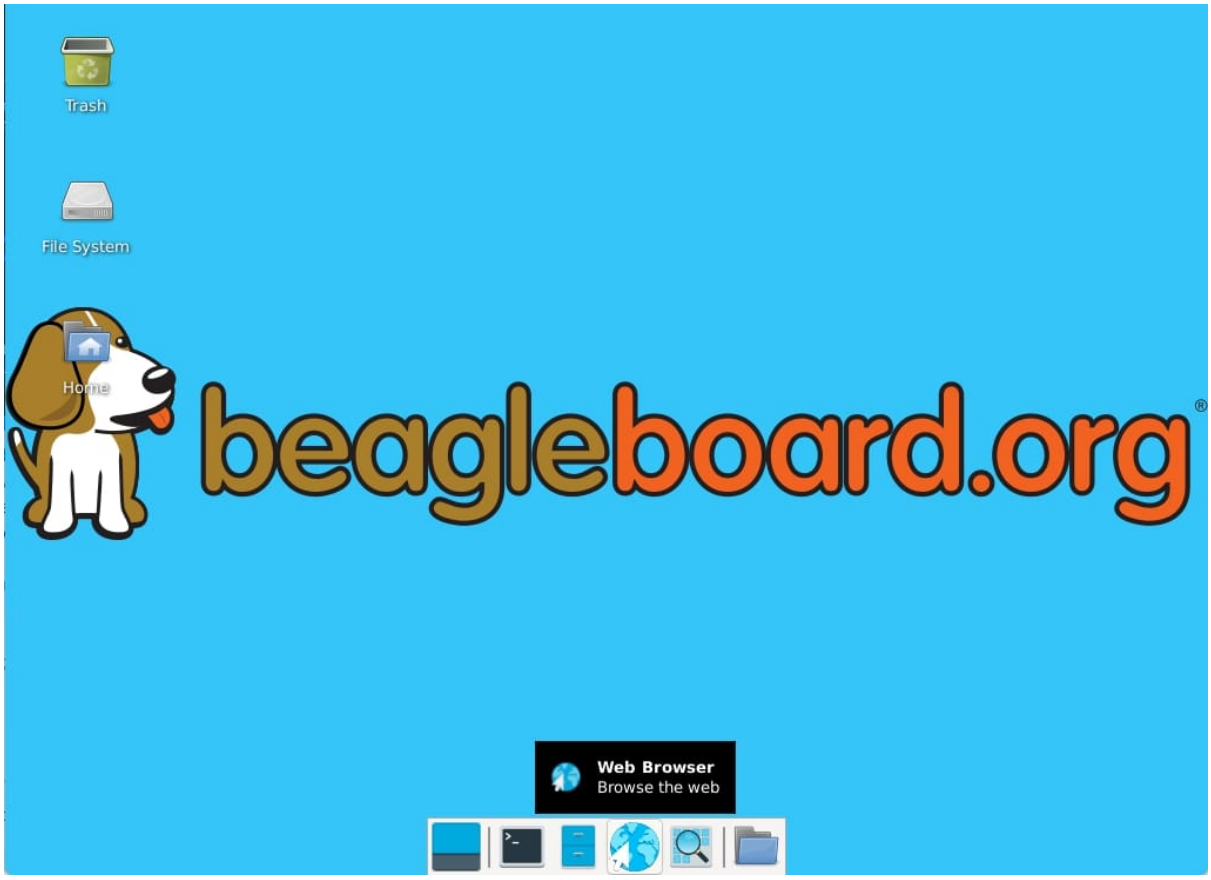


Fig. 2.16: BeagleBone XFCE Desktop Screen

2.3 Update software

Production boards currently ship with the factory-installed 2022-01-14-8GB image. To upgrade from the software image on your BeagleBone AI-64 to the latest, you don't need to completely reflash the board. If you do want to reflash it, visit the flashing instructions on the getting started page. Factory Image update (without reflashing)...

```
sudo apt update
```

```
sudo apt install --only-upgrade bb-j721e-evm-firmware generic-sys-mods
```

```
sudo apt upgrade
```

2.3.1 Update U-Boot:

to ensure only tiboot3.bin is in boot0, the pre-production image we tried to do more in boot0, but failed...

```
sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-emmc.sh
```

```
sudo /opt/u-boot/bb-u-boot-beagleboneai64/install-microsd.sh
```

```
sudo reboot
```

2.3.2 Update Kernel and SGX modules:

```
sudo apt install bbb.io-kernel-5.10-ti-k3-j721e
```

2.3.3 Update xfce:

```
sudo apt install bbb.io-xfce4-desktop
```

2.3.4 Update ti-edge-ai 8.2 examples

```
sudo apt install ti-edgeai-8.2-base ti-vision-apps-8.2 ti-vision-apps-eaik-
↪firmware-8.2
```

2.3.5 Cleanup:

```
sudo apt autoremove --purge
```

2.4 Next steps

- *Edge AI*

Chapter 3

Design and Specifications

If you want to know how BeagleBone AI-64 is designed and the detailed specifications, then this chapter is for you. We are going to attept to provide you a short and crisp overview followed by discussing each hardware design element in detail.

3.1 Block Diagram and Overview

BeagleBone AI-64 key components below shows the high level block diagram of BeagleBone AI-64 board surrounding TDA4VM SoC.

3.2 System on Chip (SoC)

BeagleBone AI-64 uses TI J721E-family TDA4VM system-on-chip (SoC) which is part of the K3 Multicore SoC architecture platform and it is targeted for the reliability and low-latency needs of the automotive market provide for a great general purpose platform suitable for industrial automation, mobile robotics, building automation and numerous hobby projects.

The SoC designed as a low power, high performance and highly integrated device architecture, adding significant enhancement on processing power, graphics capability, video and imaging processing, virtualization and coherent memory support. In addition, these SoCs support state of the art security and functional safety features. For the remaining of this section device, SoC, and processor will be used interchangeably.

Some of the main distinguished characteristics of the device are:

- 64-bit architecture with virtualization and coherent memory support, which leverages full processing capability of 64-bit Arm® Cortex®-A72
- Fully programmable industrial communication subsystems to enable future-proof designs for customers that need to adopt the new Gigabit Time-sensitive Networks (TSN) standards, but still need full support on legacy protocols and continuous system optimization over the product deployment
- Integration of vision hardware processing accelerators to facilitate extensive processing requirements in low power budget for automotive ADAS and machine vision applications
- Integration of a general-purpose microcontroller unit (MCU) with a dual Arm® Cortex®-R5F MCU subsystem, available for general purpose use as two cores or in lockstep, intended to help customers achieve functional safety goals for their end products
- Integration of a next-generation fixed and floating-point C71x Digital Signal Processor (DSP) that significantly boosts power over a broad range of general signal processing tasks for both general applications and automotive functions which also incorporates advanced techniques to improve control code efficiency and ease of programming such as branch prediction, protected pipeline, precise exception and virtual memory management

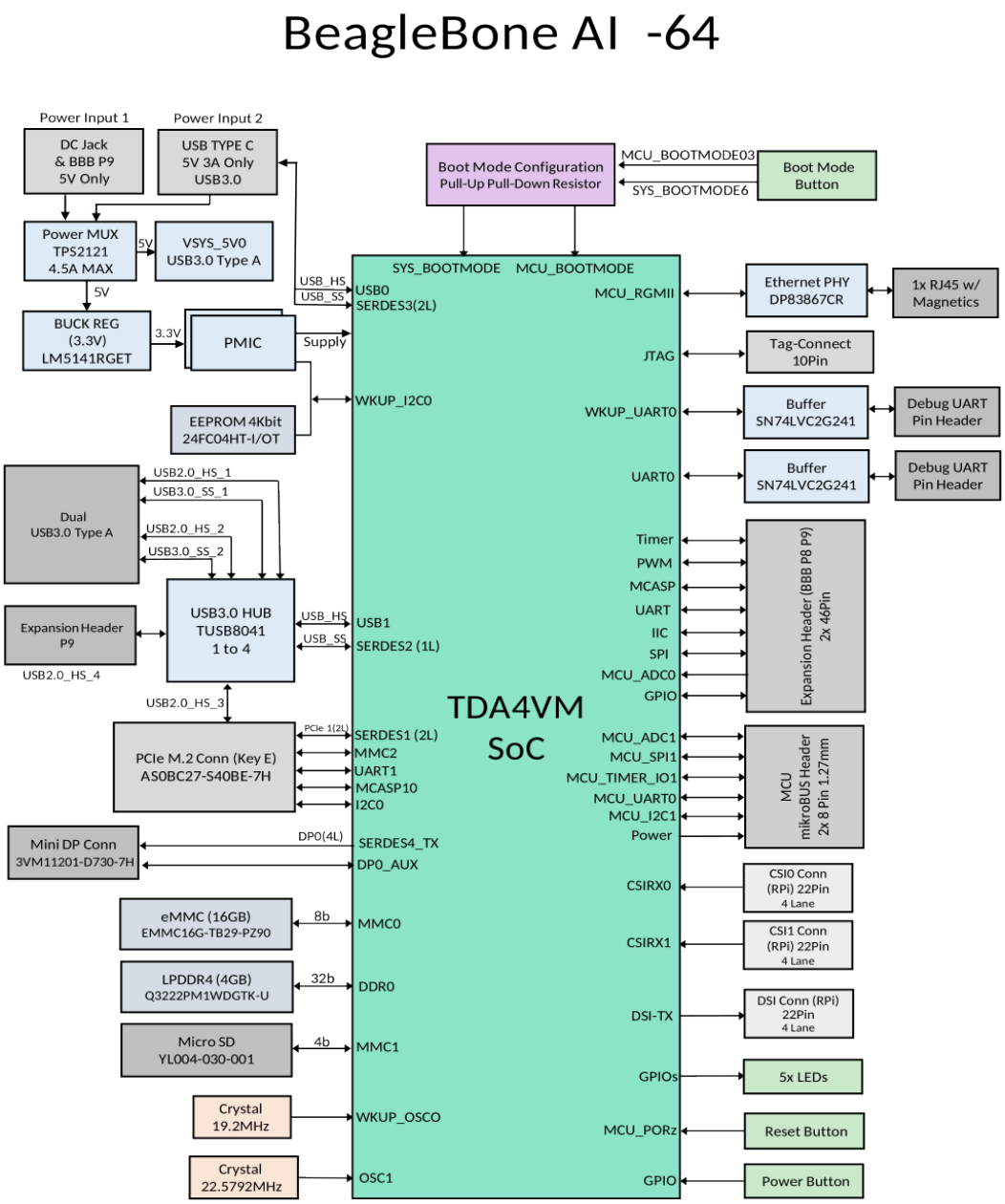


Fig. 3.1: BeagleBone AI-64 key components

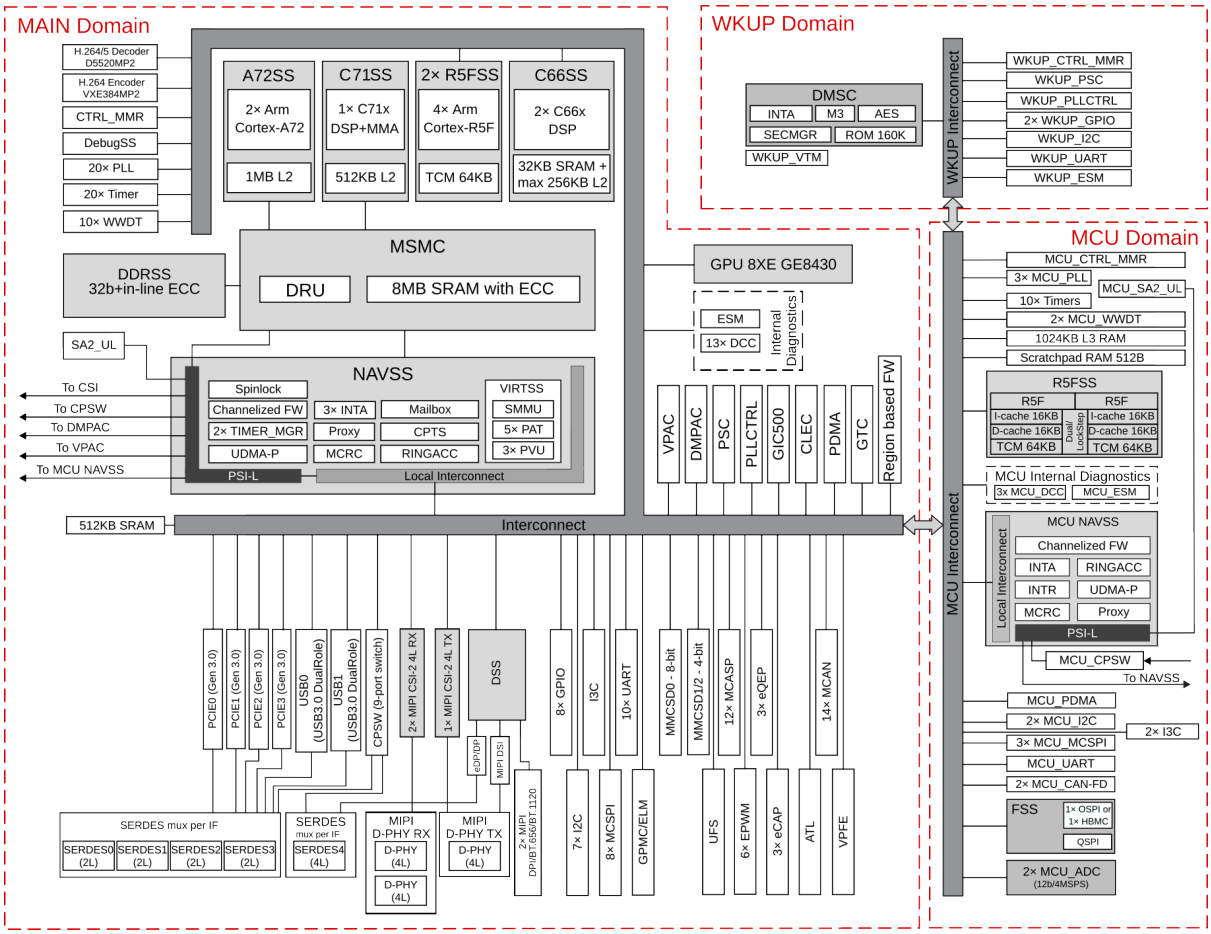


Fig. 3.2: System on Chip (SoC) block diagram

- Tightly coupled Matrix Multiplication Accelerator (MMA) that extends the C71x DSP architecture's scalar and vector facilities enabling deep learning and enhance vision, analytics and wide range of general applications. The achieved total TOPS (Tera Operations Per Second) performance significantly differentiates the device for single board computer in machine vision and deep learning applications
- Key display features including flexibility to interface with different panel types (eDP, DSI, DPI) with multi-layer hardware composition
- Integration of hardware features that help applications to achieve functional safety mechanisms
- Robust security architecture with sandboxed DMSC controller managing all secure configurations with high performance client-server messaging scheme between secure DMSC and all cores
- Simplified solution for power supply management, enabling lower cost system solution (on-die bias LDOs and power good comparators for minimal power sequencing requirements consistent with low cost supply design)

The device is composed of the following main subsystems, across different domains of the SoC, among others:

- One dual-core 64-bit Arm Cortex-A72 microprocessor subsystem at up to 2.0 GHz and up to 24K DMIPS (Dhrystone Million Instructions per Second)
- Up to three Microcontroller Units (MCU), based on dual-core Arm Cortex-R5F processor running at up to 1.0 GHz, up to 12K DMIPS
- Up to two TMS320C66x DSP CorePac modules running at up to 1.35 GHz, up to 40 GFLOPS
- One C71x floating point, vector DSP running at up to 1.0 GHz, up to 80 GFLOPS
- One deep-learning MMA, up to 8 TOPS (8b) at 1.0 GHz
- Up to two gigabit dual-core Programmable Real-Time Unit and Industrial Communication Subsystems (PRU_ICSSG)
- Two Navigator Subsystems (NAVSS) for data movement and control
- One multi-pipeline Display Subsystem (DSS) with one MIPI® Display Serial Interface Controller (DSI) and shared MIPI D-PHY Transmitter (DPHY_TX), one Embedded DisplayPort Transmitter (EDP) with shared Serializer/Deserializer (SERDES), and two MIPI Display Pixel Interface (DPI) ports
- Two Camera Streaming Interface Receivers (CSI_RX_IF) with dedicated MIPI D-PHYs (DPHY_RX)
- One Camera Streaming Interface Transmitter (CSI_TX_IF) with MIPI D-PHY Transmitter (DPHY_TX) shared with DSI
- One Vision Processing Accelerator (VPAC) with image signal processor
- One Depth and Motion Processing Accelerator (DMPAC)
- One dual-core multi-standard HD Video Decoder (DECODER)
- One dual-core multi-standard HD Video Encoder (ENCODER)
- One Graphics Processing Unit (GPU)
- One Device Management and Security Controller (DMSC)

The device provides a rich set of peripherals such as:

- **General connectivity peripherals, including:**
  - Two 12-bit general purpose Analog-to-Digital Converters (ADC)
  - Ten Inter-Integrated Circuit (I2C) interfaces
  - Three Improved Inter-Integrated Circuit (I3C) controllers
  - Eleven master/slave Multichannel Serial Peripheral Interfaces (MCSPI)
  - Twelve configurable Universal Asynchronous Receiver/Transmitter (UART) interfaces
  - Ten General-Purpose Input/Output (GPIO) modules

- **High-speed interfaces, including:**
  - Two Gigabit Ethernet Switch (CPSW) modules
  - Two Dual-Role-Device (DRD) Universal Serial Bus Subsystems (USBSS) with integrated PHY
  - Four Peripheral Component Interconnect express (PCIe) Gen3 subsystems
- **Flash memory interfaces, including:**
  - One Octal SPI (OSPI) interface and one Quad SPI (QSPI) or one QSPI and one HyperBus^TM^
  - One General Purpose Memory Controller (GPMC) with Error Location Module (ELM) and 8- or 16-bit-wide data bus width (supports parallel NOR or NAND FLASH devices)
  - Three Multimedia Card/Secure Digital (MMCSD) controllers
  - One Universal Flash Storage (UFS) interface
- **Industrial and control interfaces, including:**
  - Sixteen Controller Area Network (MCAN) interfaces with flexible data rate support
  - Three Enhanced Capture (ECAP) modules
  - Six Enhanced Pulse-Width Modulation (EPWM) subsystems
  - Three Enhanced Quadrature Encoder Pulse (EQEP) modules
- **Audio peripherals, including:**
  - One Audio Tracking Logic (ATL)
  - Twelve Multichannel Audio Serial Port (MCASP) modules supporting up to 16 channels with independent TX/RX clock/sync domain
- One Video Processing Front End (VPFE) interface module

The device also integrates:

- Power distribution, reset controls and clock management components
- **Power-management techniques for device power consumption minimization:**
  - Adaptive Voltage Scaling (AVS)
  - Dynamic Frequency Scaling (DFS)
  - Gated clocks
  - Multiple voltage domains
  - Independently controlled power domains for major modules
  - Voltage and Temperature Management (VTM) module
  - Power-on Reset Generators (PRG)
  - Power Sleep Controllers (PSC)
- Optimized interconnect (CBASS) architecture to enable latency-critical real time network and IO applications
- **Control modules (CTRL_MMRs) mainly associated with device top-level configurations such as:**
  - IO Pad and pin multiplexing configuration
  - PLL control and associated High-Speed Dividers (HSDIV)
  - Clock selection
  - Analog function controls
- Multicore Shared Memory Controller (MSMC)

- DDR Subsystem (DDRSS) with Error Correcting Code (ECC), supporting LPDDR4
- 1KB RAM with ECC support for C71x boot vectors
- 2KB RAM with ECC support for A72 and R5F boot vectors
- 512KB On-Chip SRAM protected by ECC
- One Global Time Counter (GTC) module
- Thirty 32-bit counter timers with compare and capture modes
- Debug and trace capabilities

The device includes different modules for functional safety requirements support:

- MCU island with dual lock step Arm Cortex-R5F
- Safety enabled interconnect with implemented features to help with Freedom From Interference (FFI)
- Twelve Real Time Interrupt (RTI) modules with Windowed Watchdog Timer (WWDT) functionality to monitor processor cores
- Sixteen Dual-Clock Comparators (DCC) to monitor clocking sources during run-time
- Three Error Signaling Modules (ESM) to enable error monitoring
- Temperature monitoring sensors
- ECC on all critical memories
- Dedicated hardware Memory Cyclic Redundancy Check (MCRC) blocks

The device supports the following main security functionalities among others:

- Secure Boot Management
- Public Key Accelerator (PKA) for large vector math operation
- Cryptographic acceleration (AES, 3DES, MD5, SHA1, SHA2-224, 256, 512 operation)
- Trusted Execution Environment (TEE)
- Secure storage support
- On-the-fly encryption and authentication support for OSPI interface

The device is partitioned into three functional domains as shown in *System on Chip (SoC) block diagram*, each containing specific processing cores and peripherals:

- Wake-up (WKUP) domain
- Microcontroller (MCU) domain with one of the dual Cortex-R5 cluster
- MAIN domain

3.2.1 Boot Modes

There are two boot modes:

- **eMMC Boot:** This is the default boot mode and will allow for the fastest boot time and will enable the board to boot out of the box using the pre-flashed OS image without having to purchase an microSD card or an microSD card writer.
- **SD Boot:** This mode will boot from the microSD slot. This mode can be used to override what is on the eMMC device and can be used to program the eMMC when used in the manufacturing process or for field updates.

Important: This section needs more work and references to greater detail. Other boot modes are possible. Software to support USB and serial boot modes is not provided by beagleboard.org. Please contact TI for support of this feature.

A switch is provided to allow switching between the modes.

- Holding the boot switch down during a removal and reapplication of power without a microSD card inserted will force the boot source to be the USB port and if nothing is detected on the USB client port, it will go to the serial port for download.
- Without holding the switch, the board will boot try to boot from the eMMC. If it is empty, then it will try booting from the microSD slot, followed by the serial port, and then the USB port.
- If you hold the boot switch down during the removal and reapplication of power to the board, and you have a microSD card inserted with a bootable image, the board will boot from the microSD card.

Note: Pressing the RESET button on the board will NOT result in a change of the boot mode. You MUST remove power and reapply power to change the boot mode. The boot pins are sampled during power on reset from the PMIC to the processor.The reset button on the board is a warm reset only and will not force a boot mode change.

Push-button SW1	Primary Boot Mode	Backup Boot Mode	MCU_BOOTMODE05	MCU_BOOTMODE04	MCU_BOOTMODE03	SYS_BOOTMODE6	SYS_BOOTMODE5	SYS_BOOTMODE4
Open	eMMC(MMC0)	SD Card(MMC1)	01	0		0 (Port 0)0 (1.8V)	0 (8-bit)	
Closed	SD Card(MMC1)000	SD Card(MMC1)				1 (Port 1)0 (4-bit)		0 (Filesystem mode)

Primary boot with BOOT normal-open push-button selecting eMMC port 0 (open) or MMC/SD port1 (closed).

Fig. 3.3: Boot config

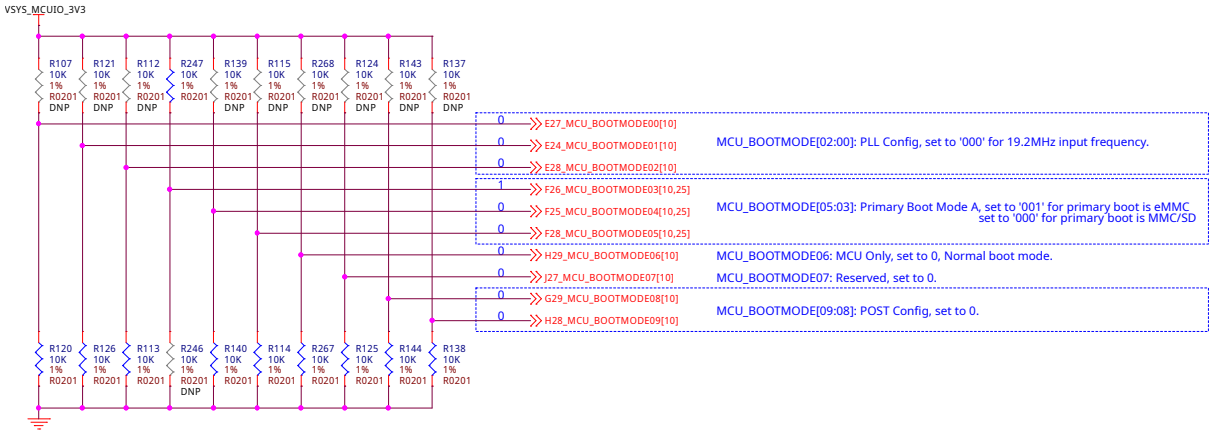


Fig. 3.4: MCU Bootmode

3.2.2 Power Sources

The board can be powered from three different sources:

- 5V > 3A power supply plugged into the barrel jack
- 5V > 3A capable device plugged into the USB Type-C connector
- The cape header pins

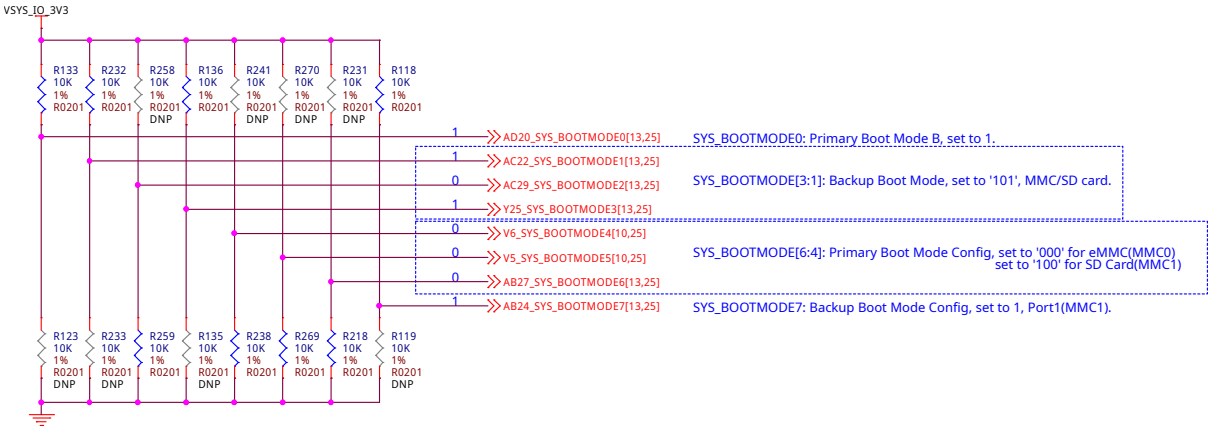


Fig. 3.5: SYS Bootmode

The power supply is not provided with the board but can be easily obtained from numerous sources. A 5V > 3A supply is mandatory to have with the board, but if there is a cape plugged into the board or you have a power hungry device or hub plugged into the host port, then more current may needed from the DC supply.

3.3 Power Management

BeagleBone AI-64 power management invludes a lot of ICs from Texas Instruments,

1. **DC/DC converter:** TPS62813 and LM5141
2. **LDO:** TPS74801
3. **PMICs:** TPS65941213 and TPS65941111
4. **Power Mux:** TPS2121
5. **Power Switch:** TPS22965

3.3.1 1V1 DC/DC

TPS62813 is a 3-A synchronous step-down DC/DC converter with high efficiency and ease of use. The TPS62813 family is based on a peak current mode control topology. The TPS62813 is designed for automotive applications such as infotainment and advanced driver assistance systems. Low resistive switches allow up to 4-A continuous output current at high ambient temperature. The switching frequency is externally adjustable from 1.8 MHz to 4 MHz and can also be synchronized to an external clock in the same frequency range. In PWM/PFM mode, the TPS62813 automatically enter power save mode at light loads to maintain high efficiency across the whole load range. The TPS62813 provide 1% output voltage accuracy in PWM mode which helps design a power supply with high output voltage accuracy. The SS/TR pin allows setting the start-up time or forming tracking of the output voltage to an external source. This feature allows external sequencing of different supply rails and limiting the inrush current during start-up.

3.3.2 1V1 & 2V5 LDO

TPS74801 is a 1.5-A low-VIN (0.8 V) adjustable low-dropout (LDO) voltage regulator with power good and enable. The TPS748 low-dropout (LDO) linear regulator provides an easy-to-use robust power management solution for a wide variety of applications. User- programmable soft-start minimizes stress on the input power source by reducing capacitive inrush current on start-up. The soft-start is monotonic and designed for powering many different types of processors and ASICs. The enable input and power-good output allow easy sequencing with

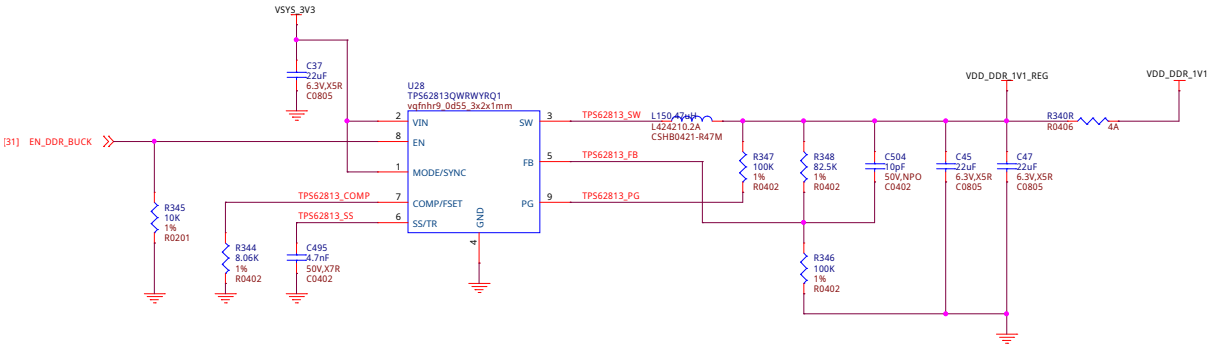


Fig. 3.6: 1V1 @ 1A DDR power supply

external regulators. This complete flexibility allows a solution to be configured that meets the sequencing requirements of FPGAs, DSPs, and other applications with special start-up requirements.

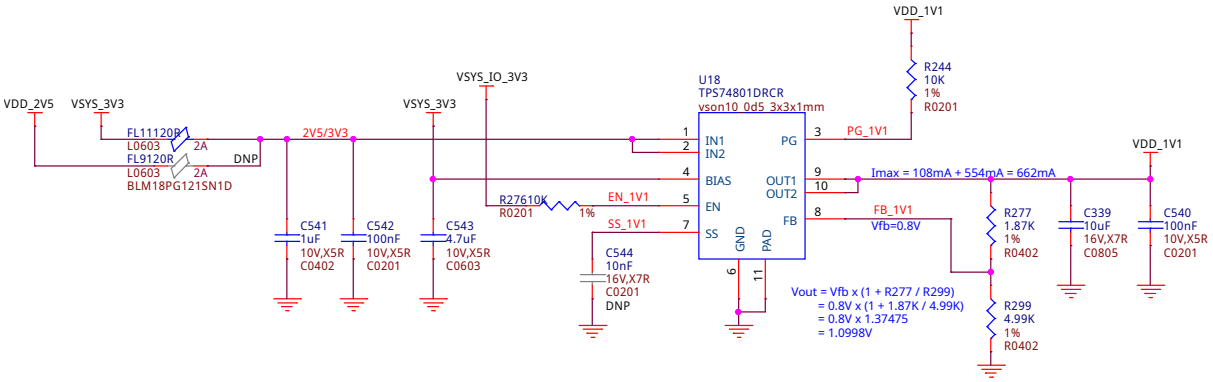


Fig. 3.7: 1V1 USB3 & Ethernet power supply

3.3.3 3V3 DC/DC

The LM5141 is a synchronous buck controller, intended for high voltage wide VIN step-down converter applications. The control method is peak current mode control. Current mode control provides inherent line feed-forward, cycle-by-cycle current limiting, and ease of loop compensation. The LM5141 features slew rate control to simplify the compliance with EMI requirements. The LM5141 has two selectable switching frequencies: 2.2 MHz and 440 kHz. Gate Drivers with Slew Rate Control that can be adjusted to reduce EMI. In light or no-load conditions, the LM5141 operates in skip cycle mode for improved low power efficiency. The LM5141 has a high voltage bias regulator with automatic switch-over to an external bias to reduce the IQ current from VIN. Additional features include frequency synchronization, cycle-by-cycle current limit, hiccup mode fault protection for sustained overload, and power good output.

3.3.4 PMIC

TPS6594-Q1 is a Power Management IC (PMIC) with 5 BUCKs and 4 LDOs for Safety-Relevant Automotive Applications. The TPS6594-Q1 device provides four flexible multi-phase configurable BUCK regulators with 3.5 A output current per phase, and one additional BUCK regulator with 2 A output current. We are using two TPS6594-Q1 ICs TPS65941213 and TPS65941111 as PMIC-A and PMIC-B respectively as shown in AI-64 schematic snippets below.

TPS65941213 (PMIC-A)

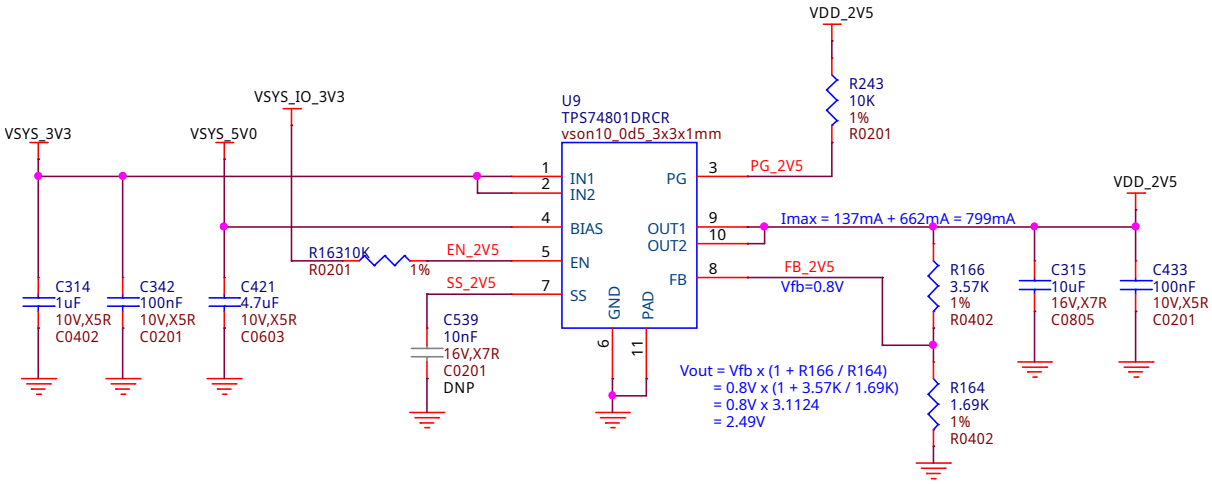


Fig. 3.8: 2V5 Ethernet power supply

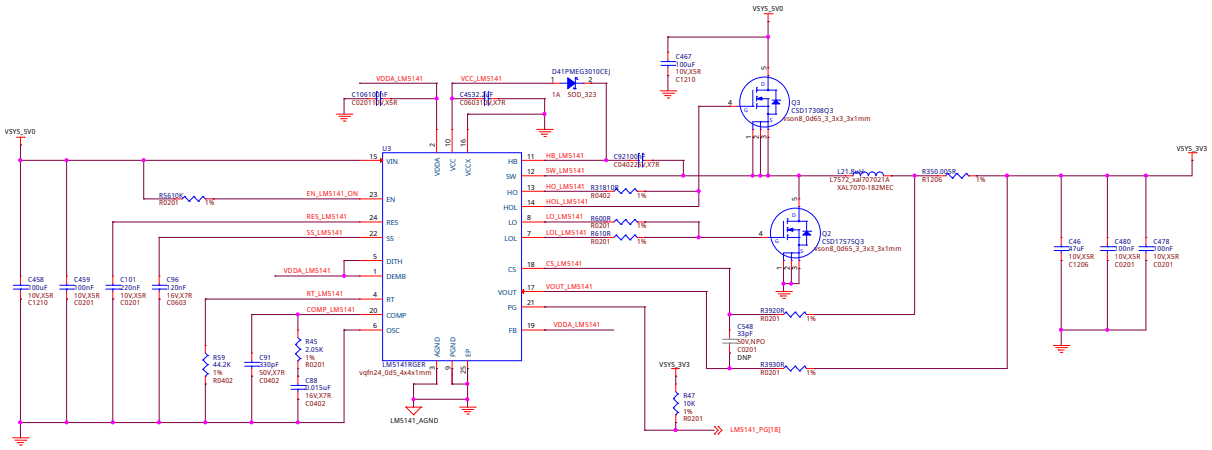


Fig. 3.9: 3V3 power supply

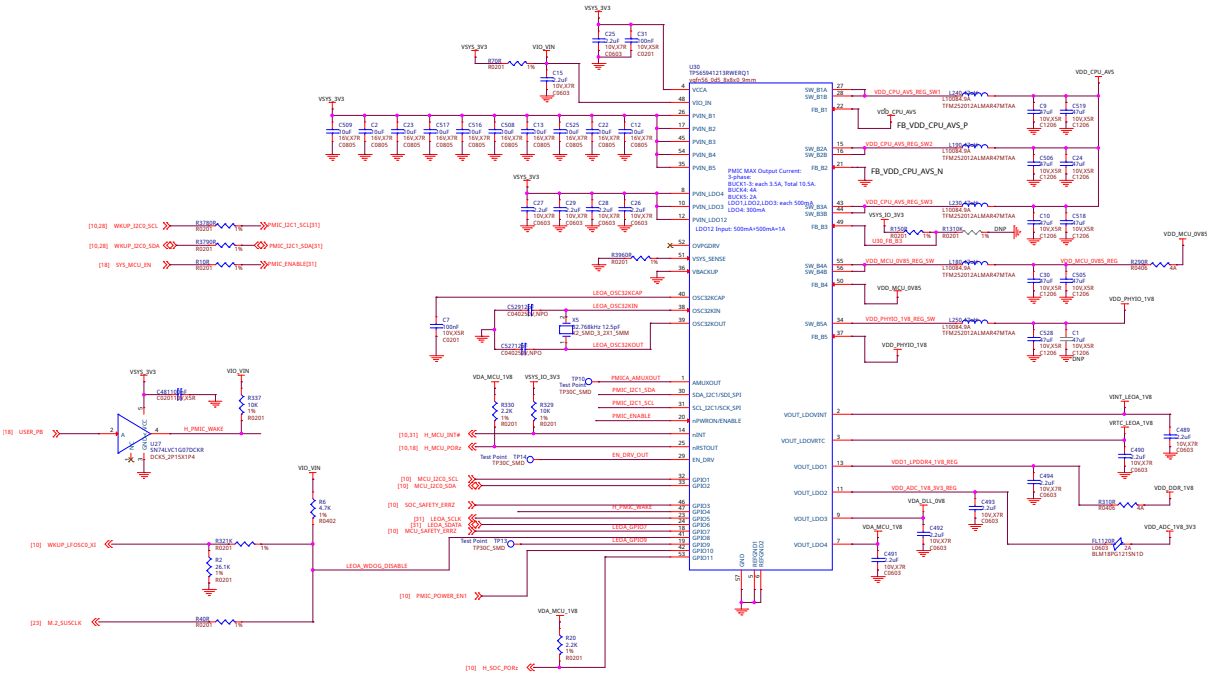


Fig. 3.10: PMIC A

TPS65941111 (PMIC-B)

3.3.5 Power mux

TPS2121 is a 2.7-V to 22-V, 56-mΩ, 4.5-A, power mux with seamless switchover. The TPS212x devices are Dual-Input, Single-Output (DISO) Power Multiplexer (MUX) that are well suited for a variety of systems having multiple power sources. The devices will Automatically Detect, Select, and Seamlessly Transition between available inputs. Priority can be automatically given to the highest input voltage or manually assigned to a lower voltage input to support both ORing and Source Selection operations. A priority voltage supervisor is used to select an input source. An Ideal Diode operation is used to seamlessly transition between input sources. During switchover, the voltage drop is controlled to block reverse current before it happens and provide uninterrupted power to the load with minimal hold-up capacitance. Current limiting is used during startup and switchover to protect against overcurrent events, and also protects the device during normal operation. The output current limit can be adjusted with a single external resistor.

3.3.6 Load switch

TPS22965 is a 5.7-V, 6-A, 16-mΩ load switch with adj. rise time and optional output discharge. The TPS22965 is a single channel load switch that provides configurable rise time to minimize inrush current. The device contains an N-channel MOSFET that can operate over an input voltage range of 0.8 V to 5.7 V and can support a maximum continuous current of 6 A. The switch is controlled by an on and off input (ON), which is capable of interfacing directly with low-voltage control signals. In the TPS22965, a 225-Ω on-chip load resistor is added for quick output discharge when switch is turned off

3.4 General connectivity and expansion

3.4.1 USB type C

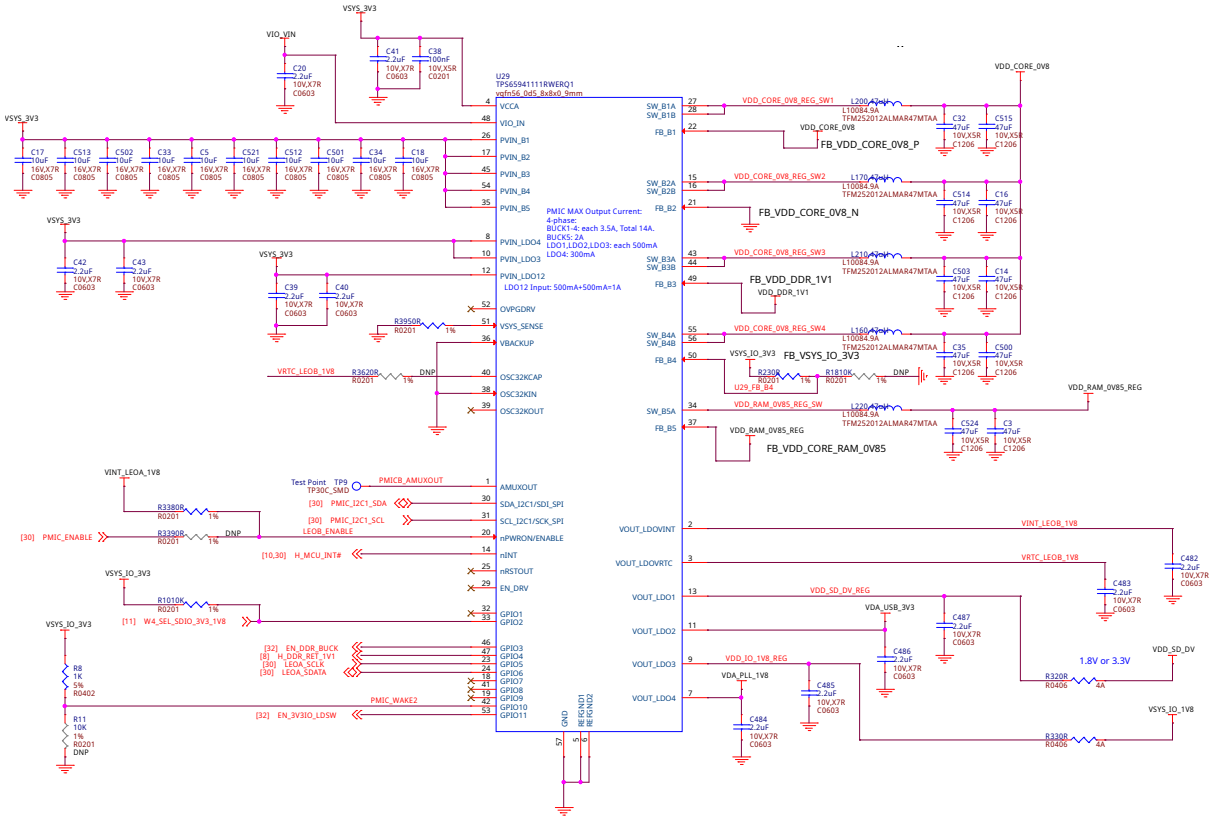


Fig. 3.11: PMIC B

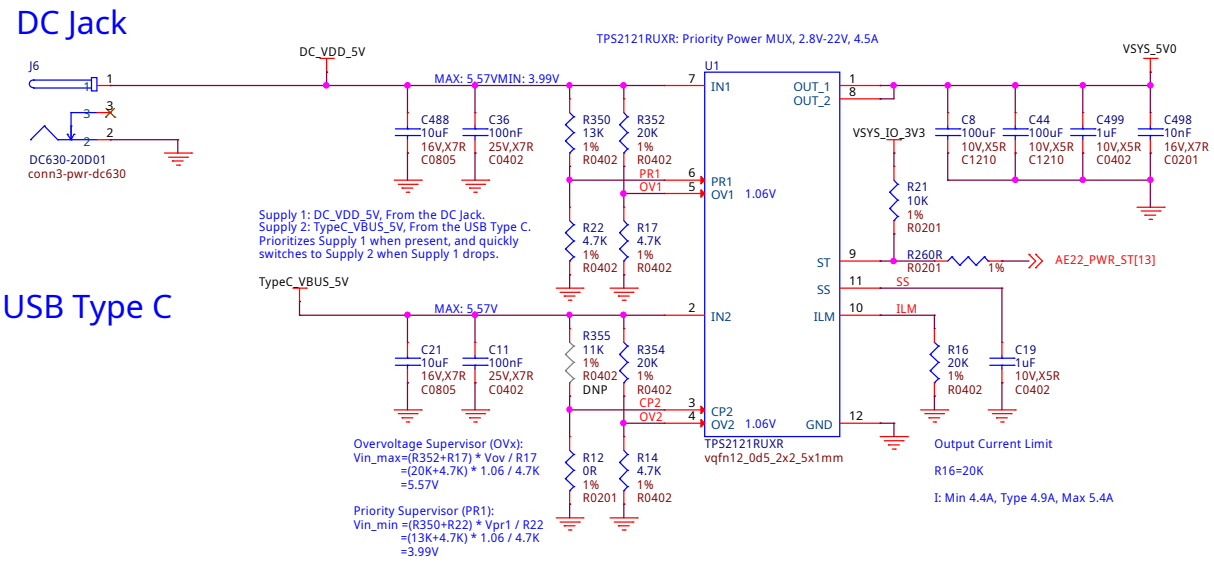


Fig. 3.12: Power mux

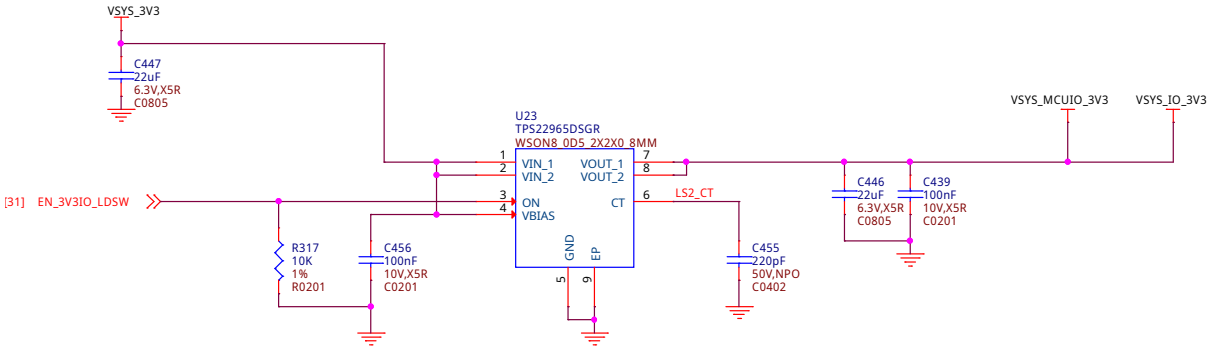


Fig. 3.13: 3V3 load switch

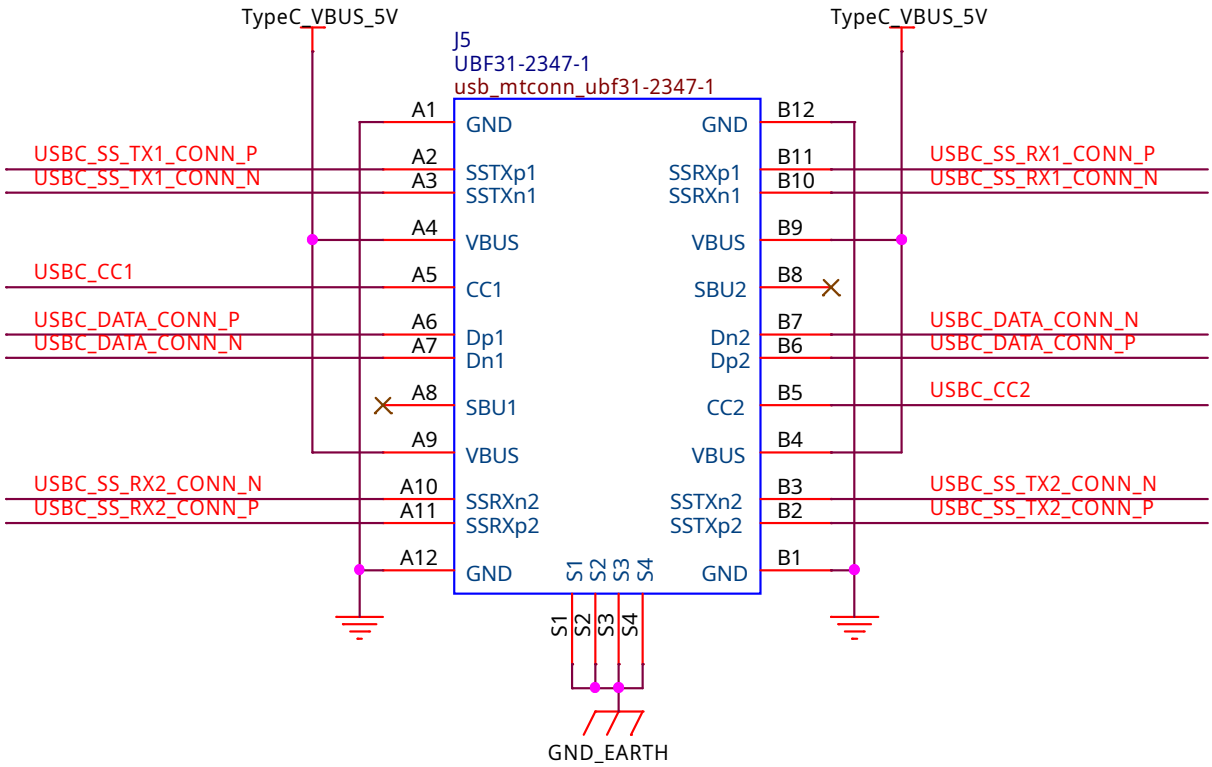


Fig. 3.14: USB type c

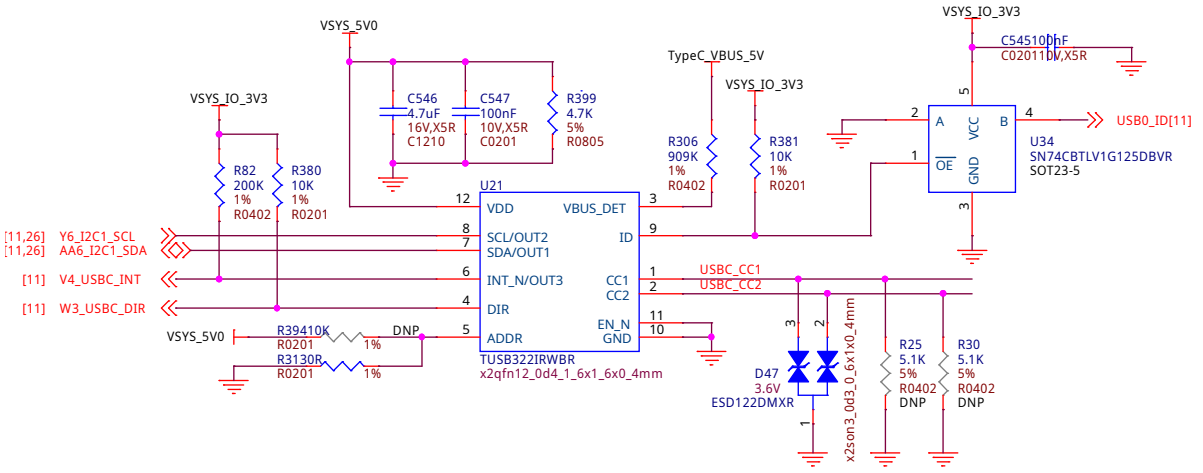


Fig. 3.15: USB type c CC logic

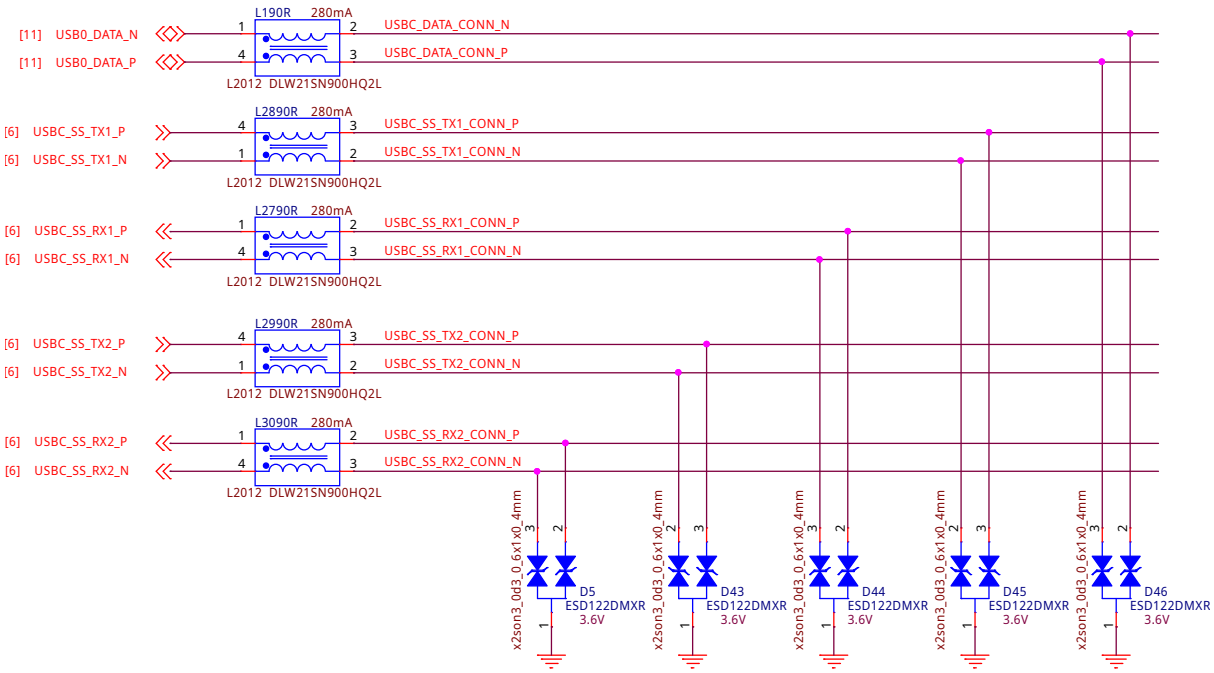


Fig. 3.16: USB type c signals

3.4.2 USB3 Host Ports

On the board is a stacked dual USB 3.0 Type A female connector with full LS/FS/HS/SS host support. The ports can provide power on/off control and up to 1.5A of current at 5V. Under USB power, the board will not be able to supply the full 1.5A.

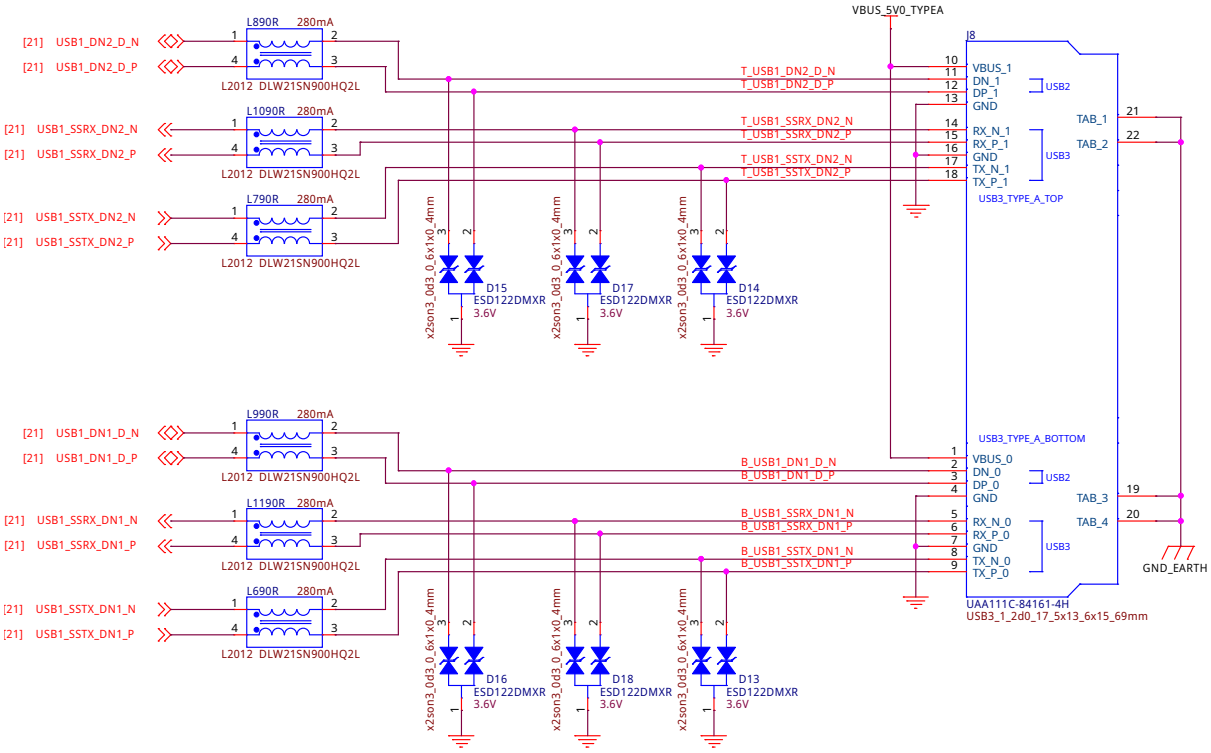


Fig. 3.17: Dual USB3 ports

3.4.3 Cape headers

P8 cape header

P9 cape header

Double pins (shorted)

3.4.4 Fan header

3.4.5 MicroSD Connector

The board is equipped with a single microSD connector to act as the secondary boot source for the board and, if selected as such, can be the primary boot source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board. Booting from MMC0 will be used to flash the eMMC in the production environment or can be used by the user to update the SW as needed.

3.4.6 MikroBus port

3.4.7 PCIe Key E

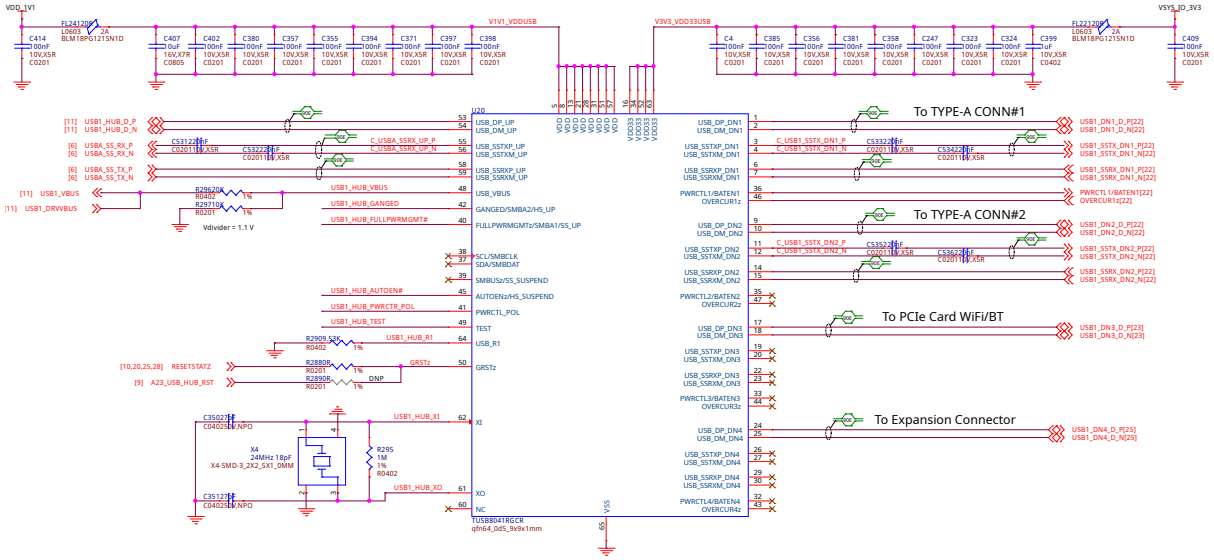


Fig. 3.18: USB3 hub

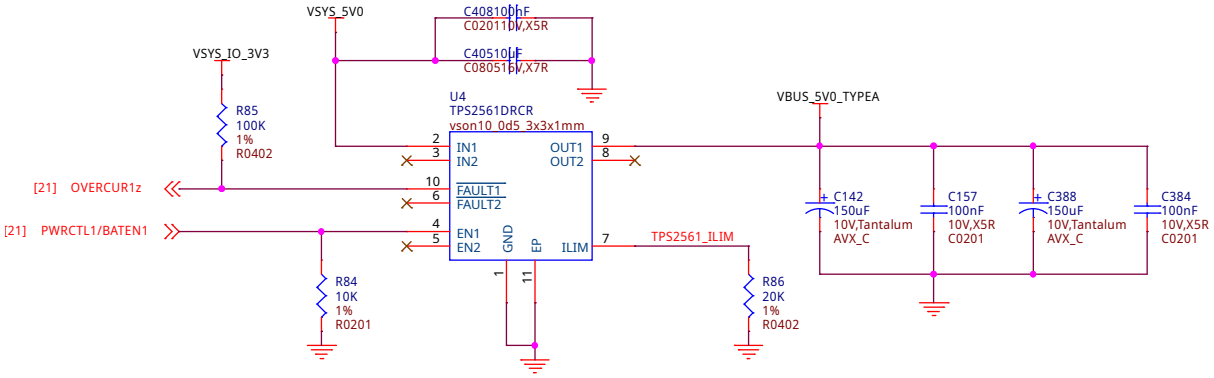


Fig. 3.19: USB3 hub over-current protection

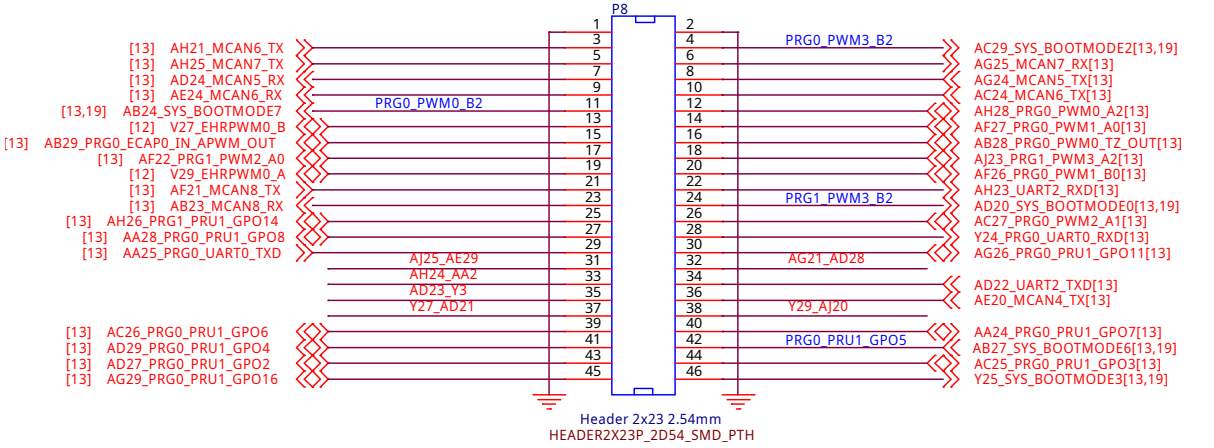


Fig. 3.20: P8 cape header

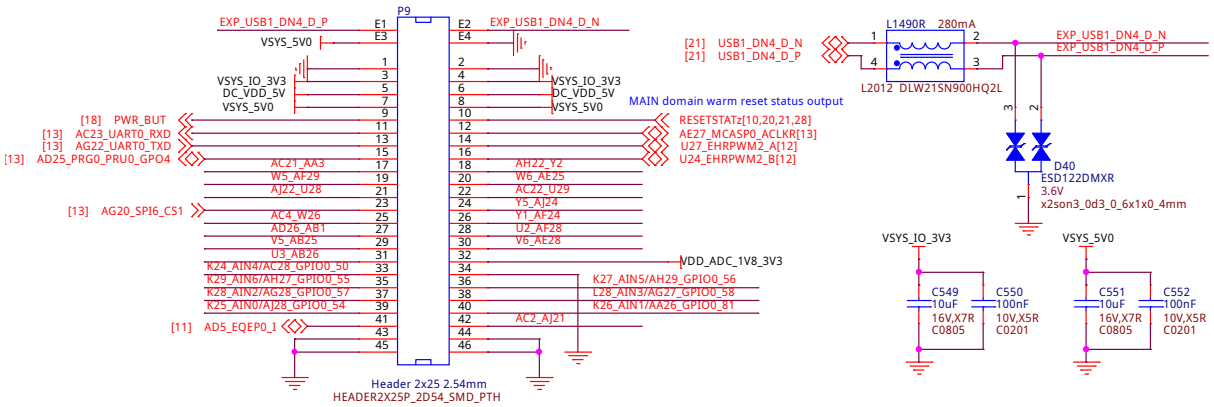


Fig. 3.21: P9 cape header

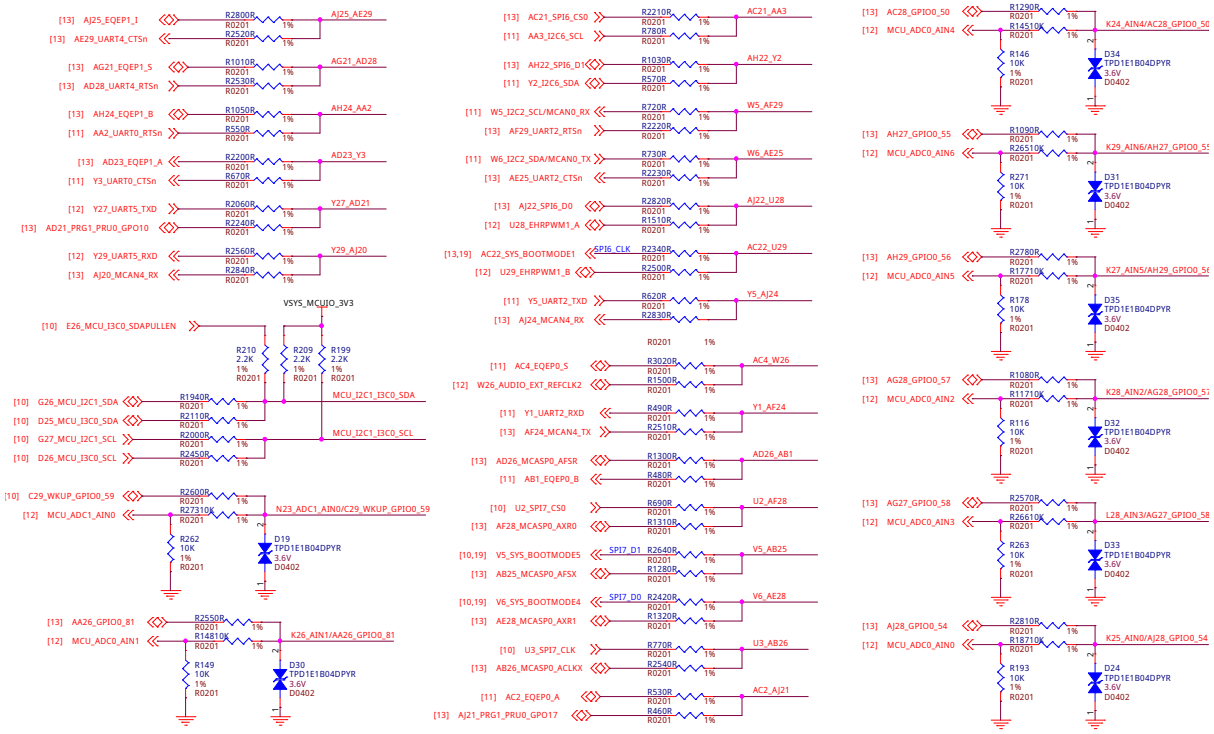


Fig. 3.22: P8 & P9 cape header pins that uses two pins of SoC

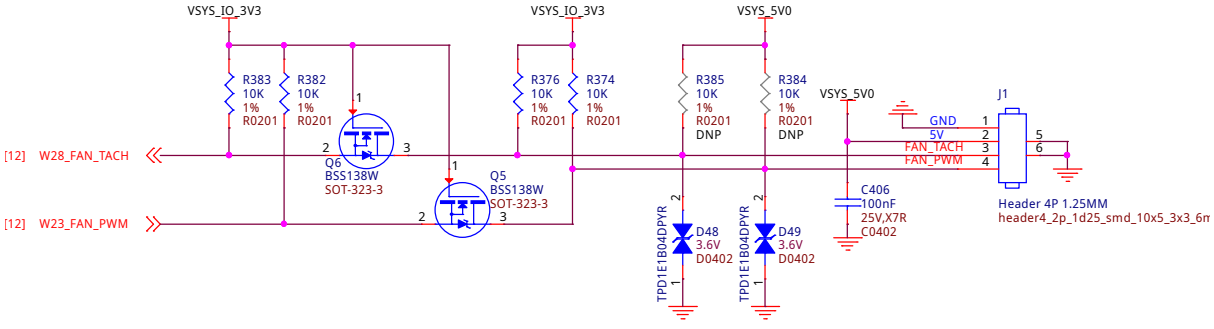


Fig. 3.23: Fan header

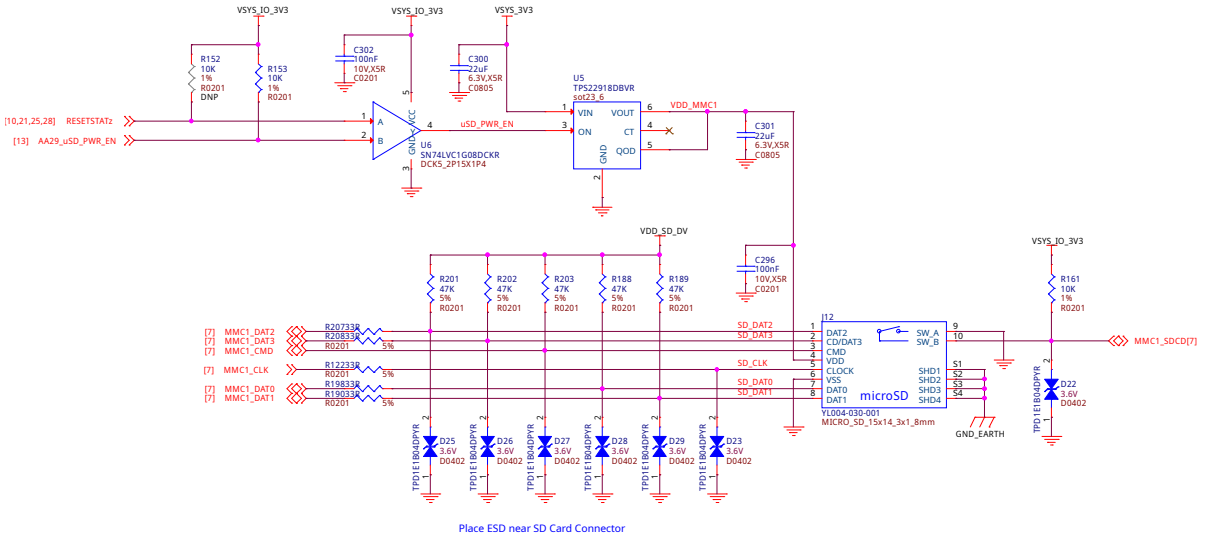


Fig. 3.24: MicroSD card slot

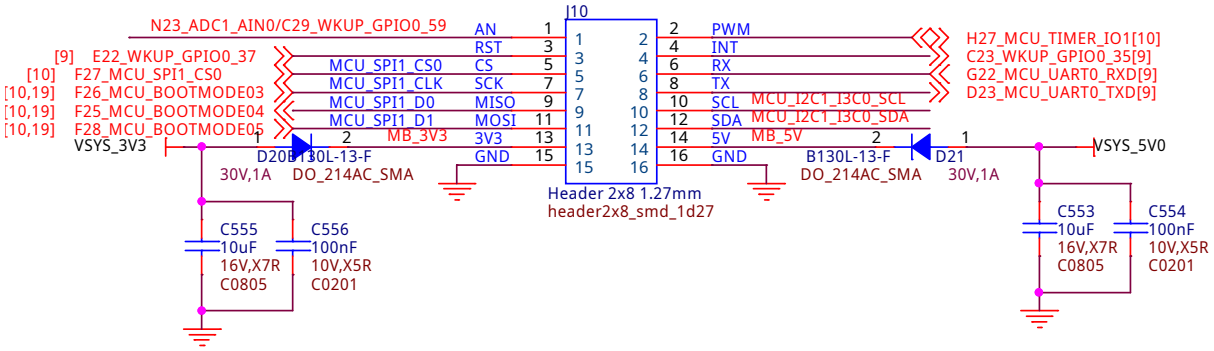


Fig. 3.25: MikroBus port

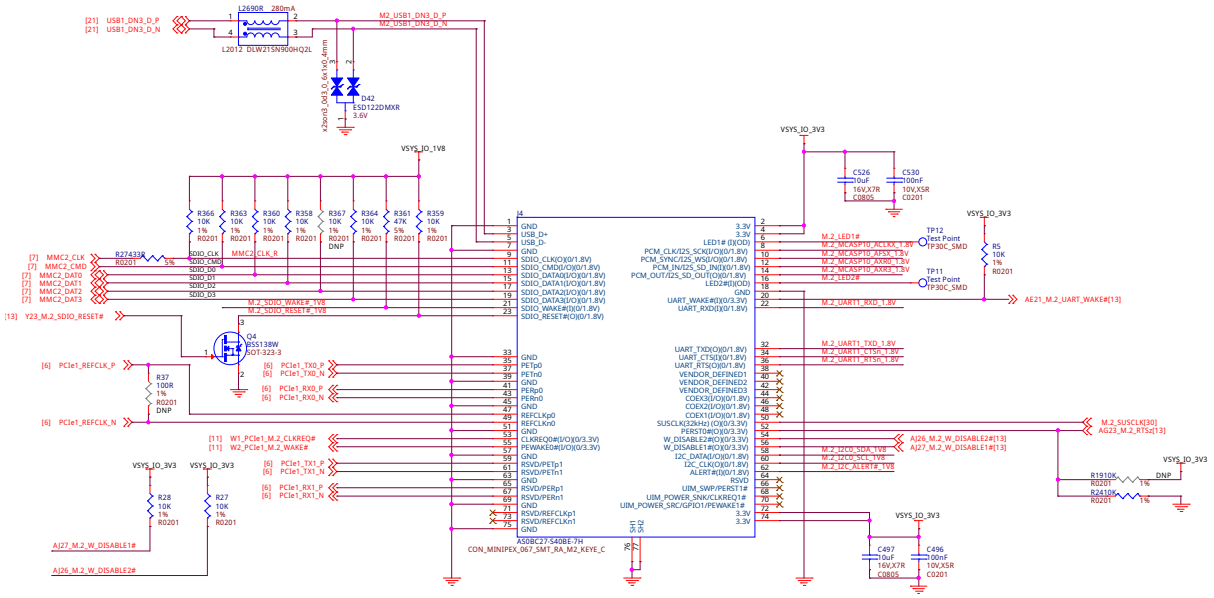


Fig. 3.26: PCIE Key E connector

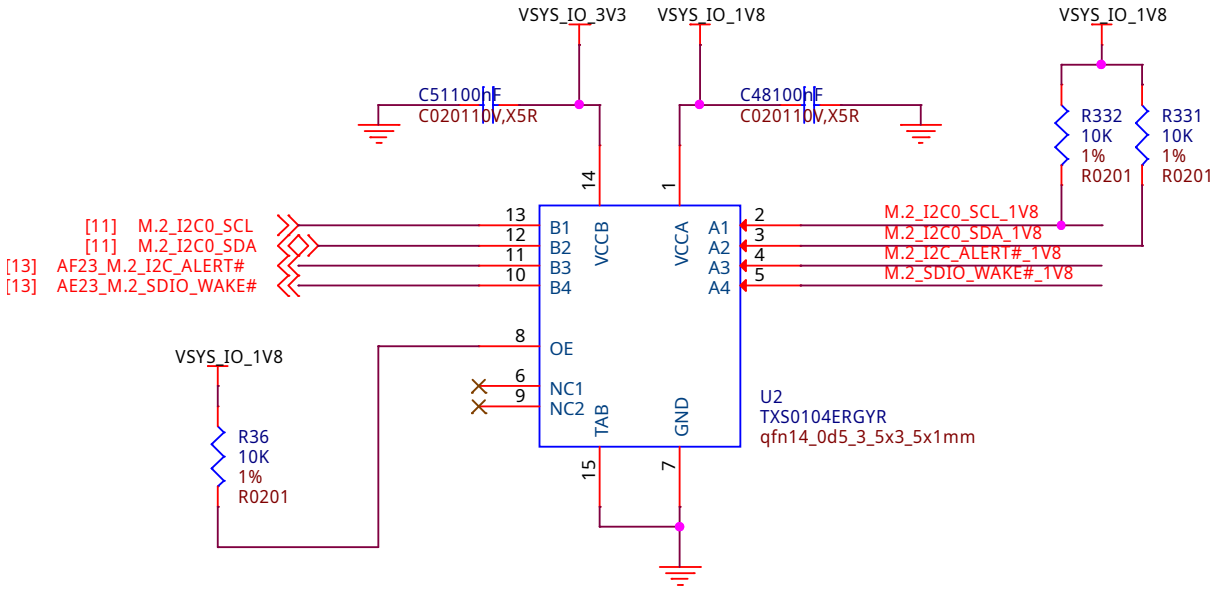


Fig. 3.27: PCIE Key E voltage translator (4ch)

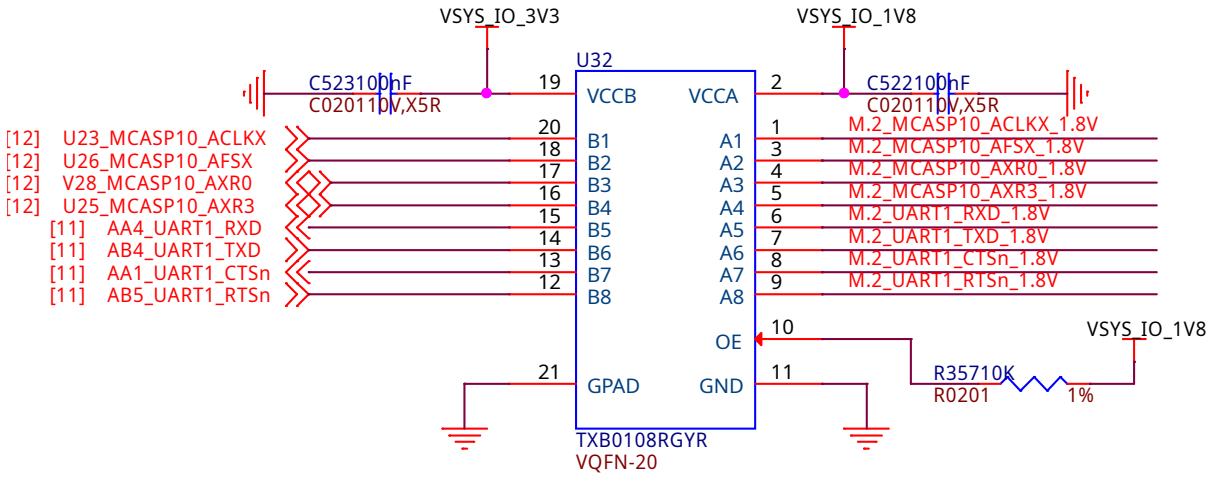


Fig. 3.28: PCIE Key E voltage translator (8ch)

3.5 Buttons & LEDs

3.5.1 Reset & Power Button

1. **Reset button:** When pressed and released, causes a reset of the board.
2. **Power button:** This button takes advantage of the input to the PMIC for power down features.

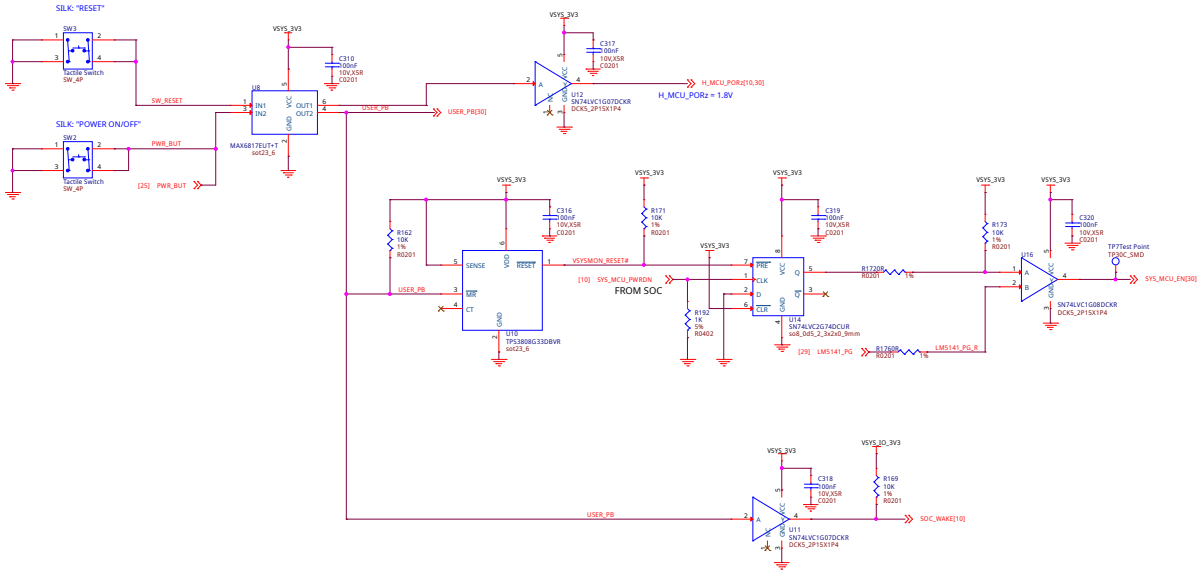


Fig. 3.29: Reset & power button

3.5.2 Boot button

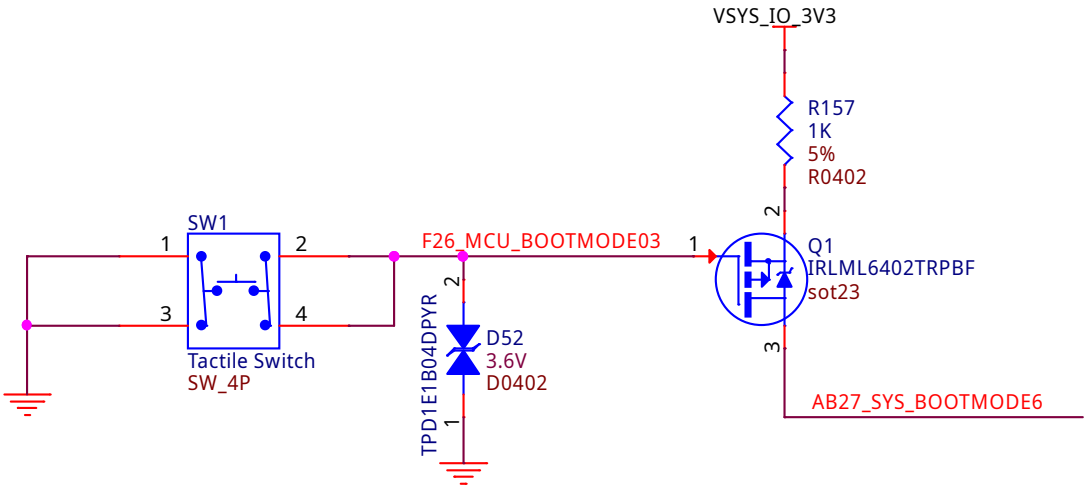


Fig. 3.30: Boot button

3.5.3 LED Indicators

There are a total of six green LEDs on the board.

- One green power LED indicates that power is applied and the power management IC is up.
- Five blue LEDs that can be controlled via the SW by setting GPIO pins.

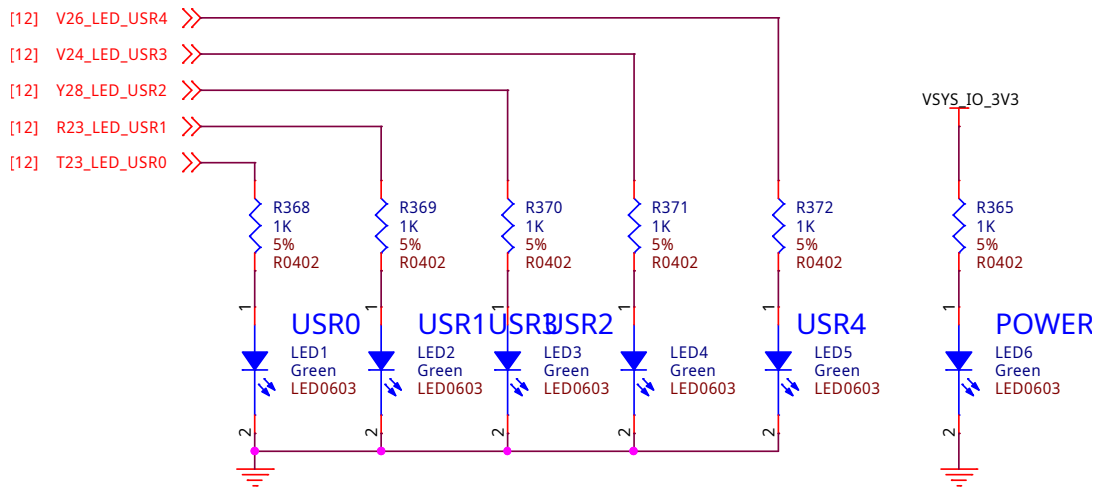


Fig. 3.31: LED indicators

3.6 Gigabit Ethernet

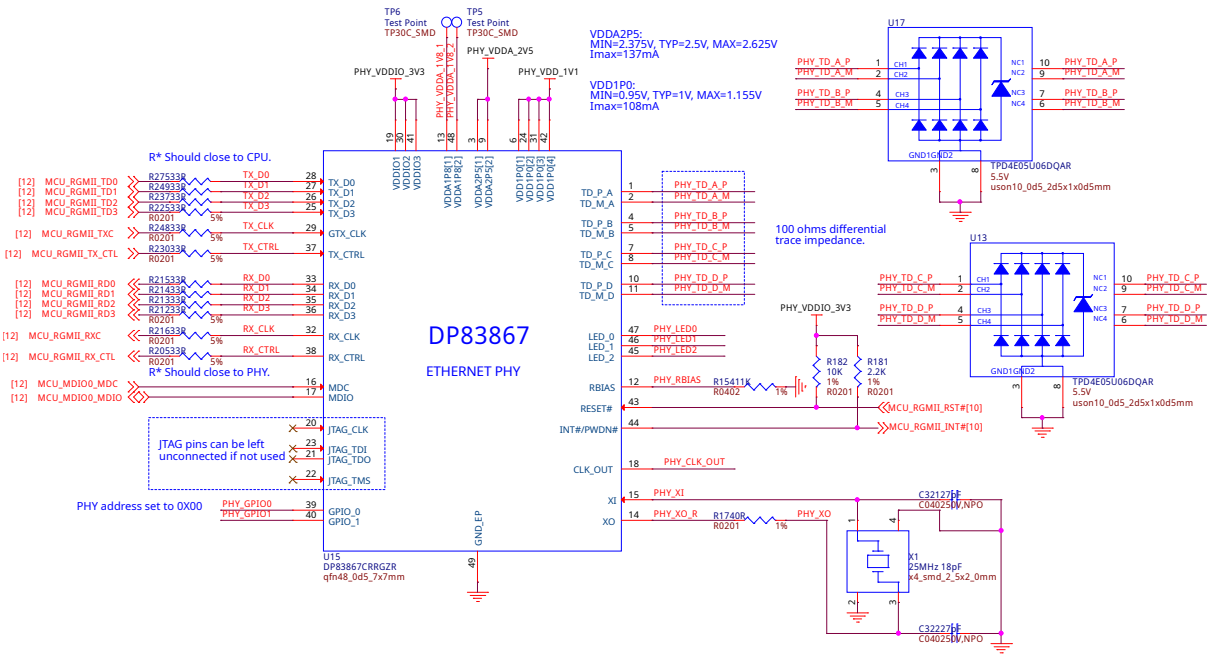


Fig. 3.32: Gigabit ethernet

3.7 Memory, Media, and storage

Described in the following sections are the three memory devices found on the board.

3.7.1 16GB Embedded MMC

A single 16GB embedded MMC (eMMC) device is on the board. The device connects to the MMC1 port of the processor, allowing for 8bit wide access. Default boot mode for the board will be MMC1 with an option to change it to MMC0, the SD card slot, for booting from the SD card as a result of removing and reapplying the

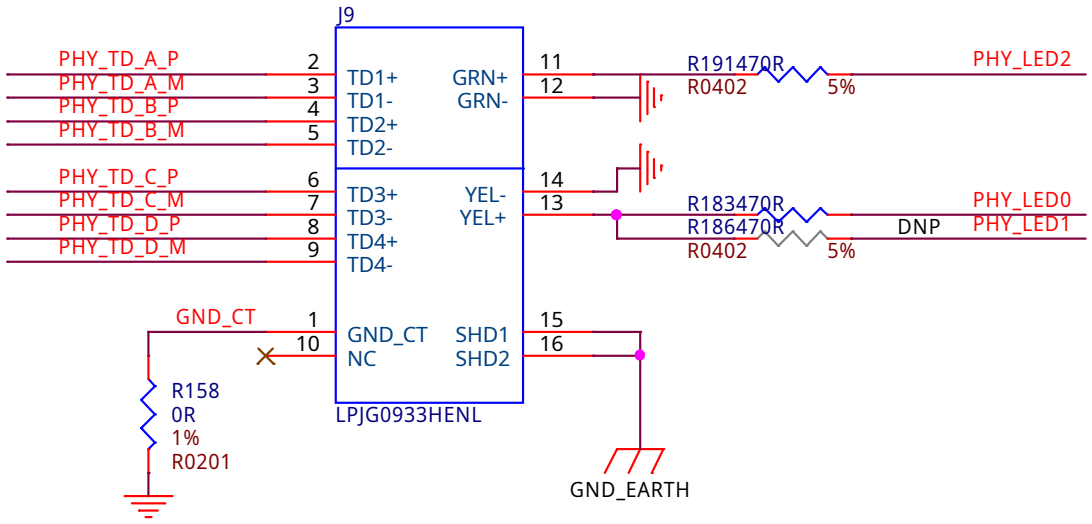


Fig. 3.33: Gigabit ethernet connector

power to the board. Simply pressing the reset button will not change the boot mode. MMC0 cannot be used in 8Bit mode because the lower data pins are located on the pins used by the Ethernet port. This does not interfere with SD card operation but it does make it unsuitable for use as an eMMC port if the 8 bit feature is needed.

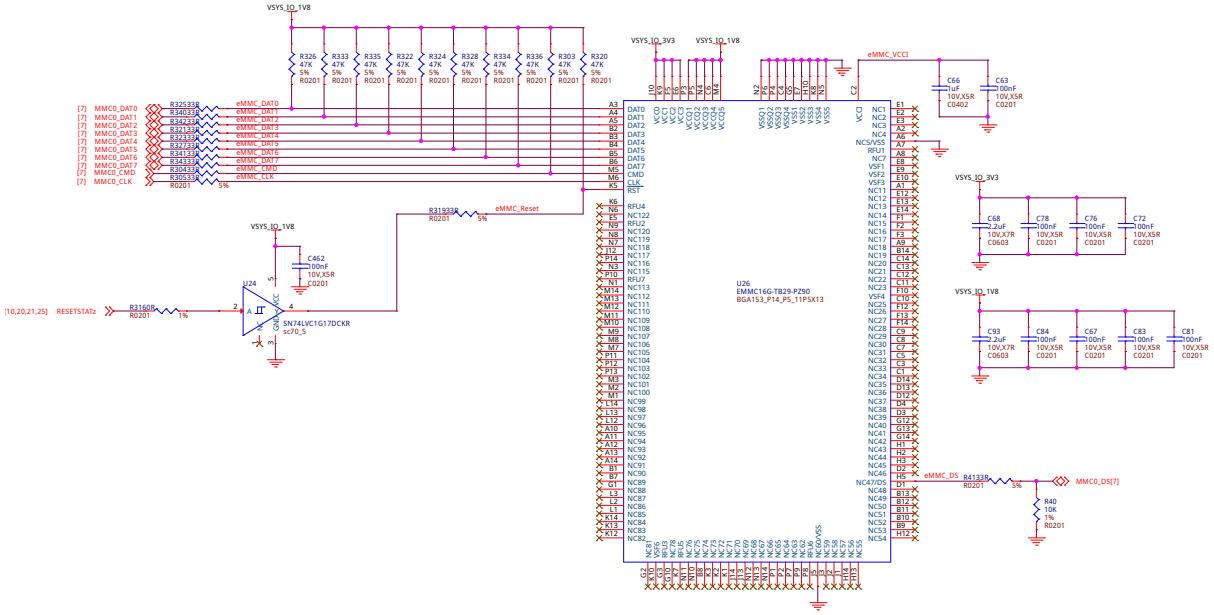


Fig. 3.34: 16GB eMMC storage

3.7.2 4GB LPDDR4

A single (1024M x 16bits x 2channels) LPDDR4 4Gb memory device is used. The memory used is:

- Kingston Q3222PM1WDGTK-U

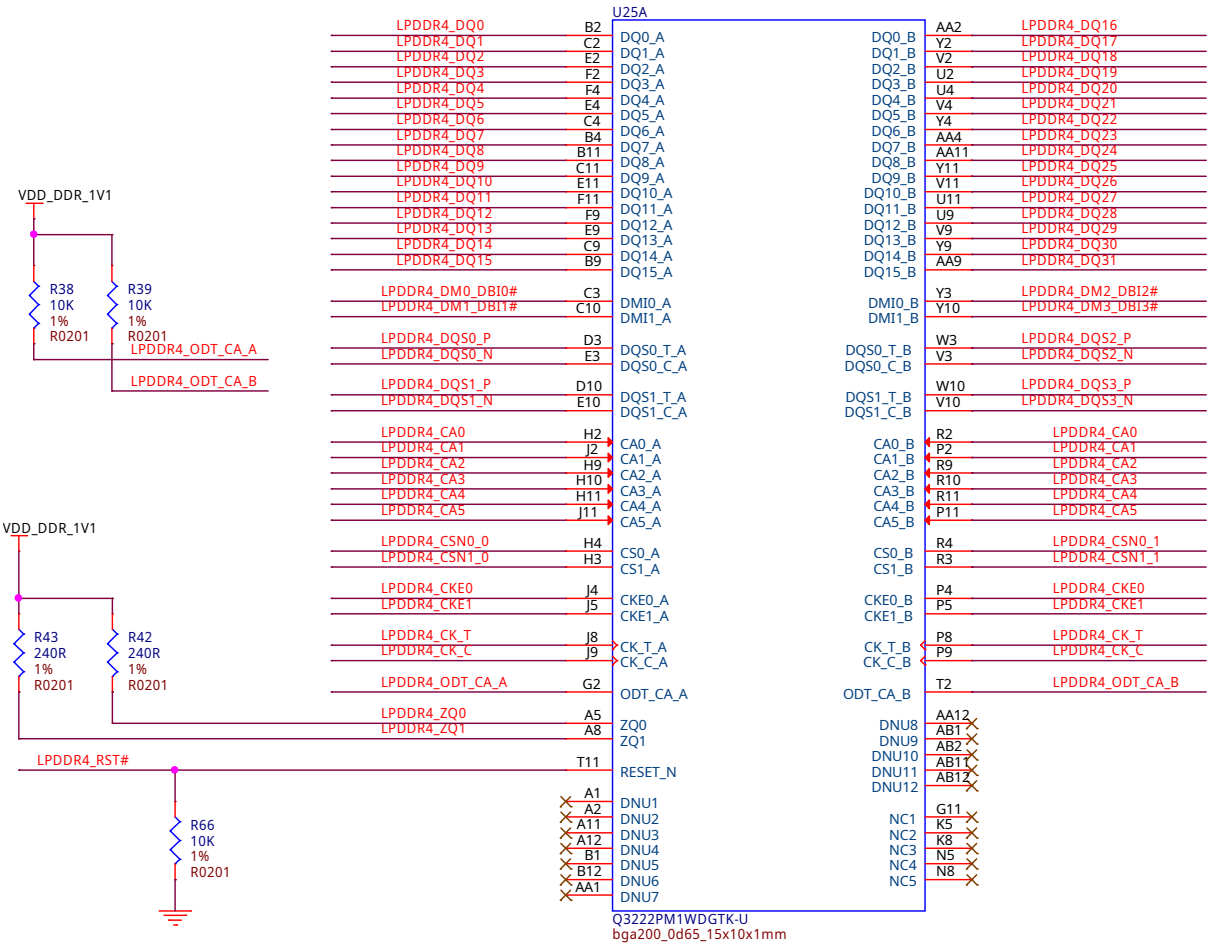


Fig. 3.35: 4GB LPDDR4 RAM

3.7.3 4Kb EEPROM

A single 4Kb EEPROM (24FC04HT-I/OT) is provided on I2C0 that holds the board information. This information includes board name, serial number, and revision information.

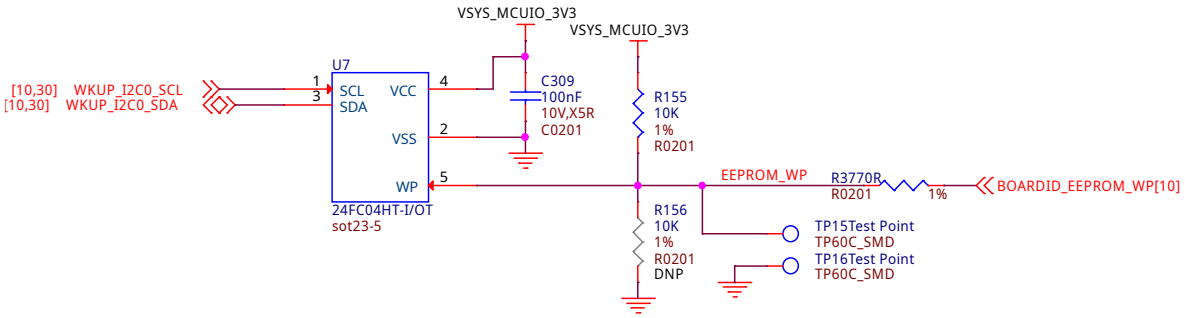


Fig. 3.36: Board ID EEPROM

3.8 Multimedia I/O

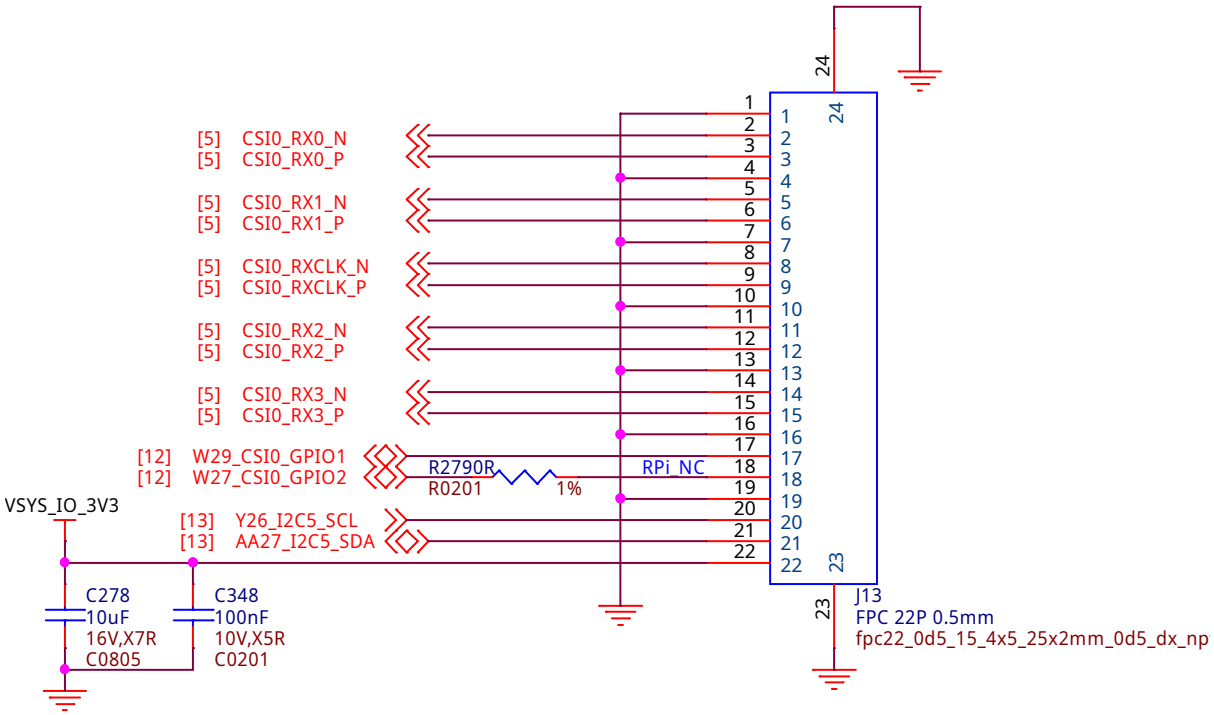


Fig. 3.37: CSI0 MIPI camera0 input

3.9 Debug Ports

3.9.1 Serial debug ports

Two serial debug ports are provided on board via 3pin micro headers,

1. WKUP_UART0: Wake-up domain serial port

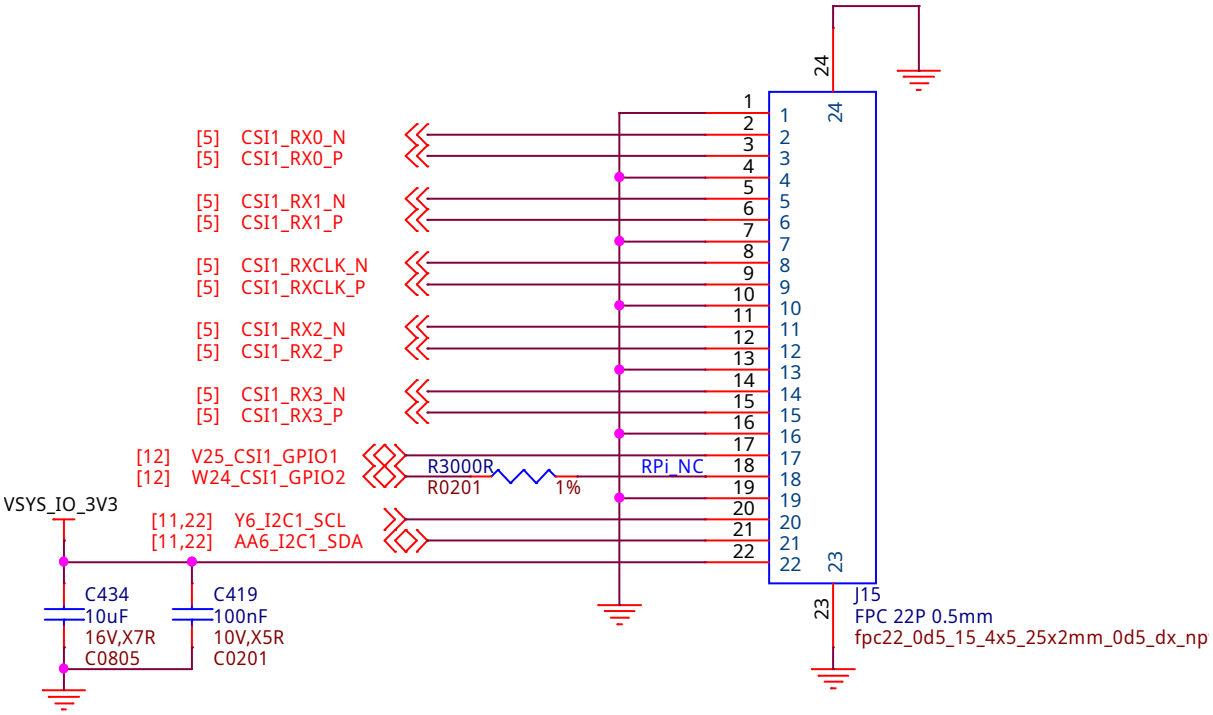


Fig. 3.38: CSI1 MIPI camera1 input

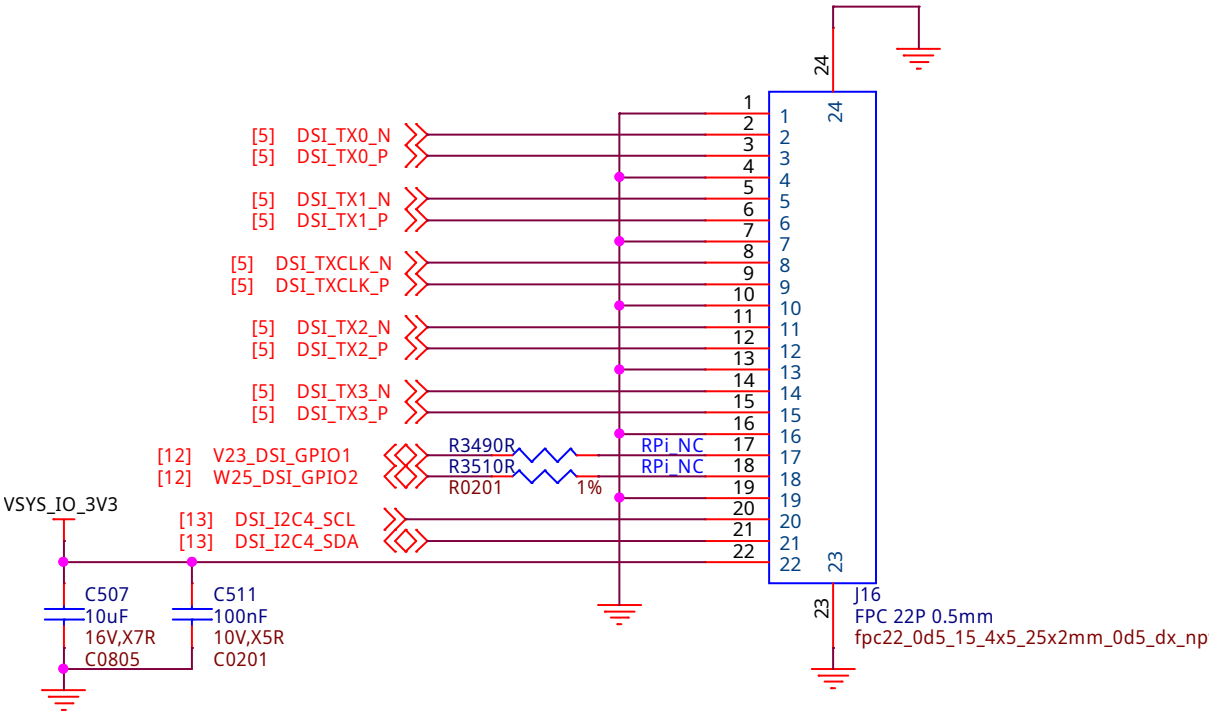


Fig. 3.39: DSI MIPI display output

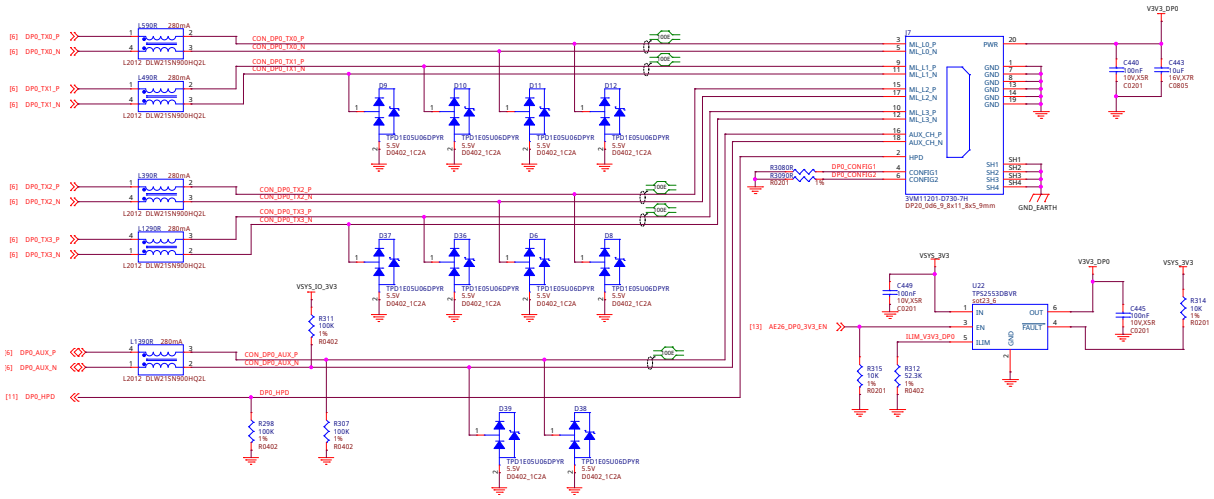


Fig. 3.40: Mini display port output

2. UART0: Main domain serial port

Note: In order to use the interfaces a 3pin micro to 6pin dupont adaptor header is required with a 6 pin USB to TTL adapter. The header is compatible with the one provided by FTDI and can be purchased for about $12 to $20 from various sources. Signals supported are TX and RX. None of the handshake signals are supported.

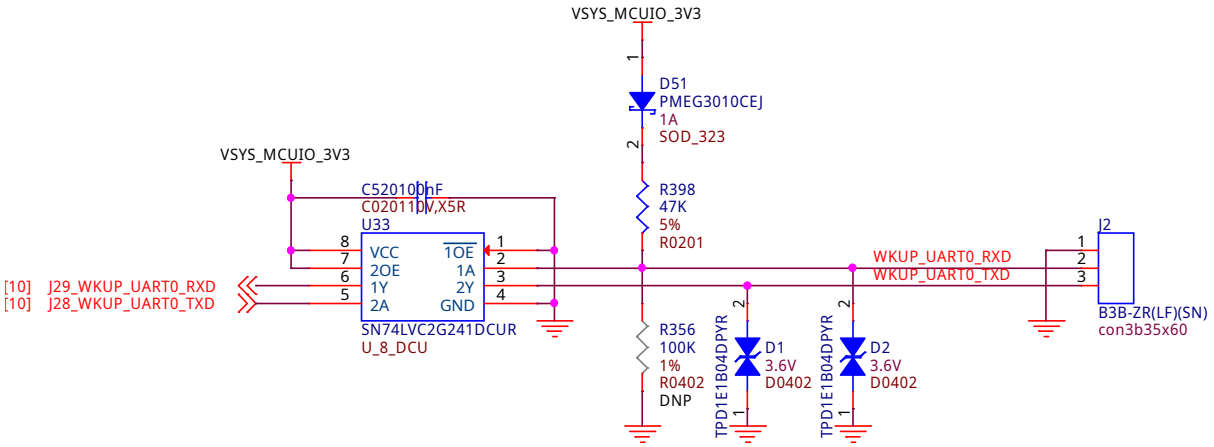


Fig. 3.41: WKUP UART0 debug port

3.9.2 TagConnect

3.10 Mechanical specifications

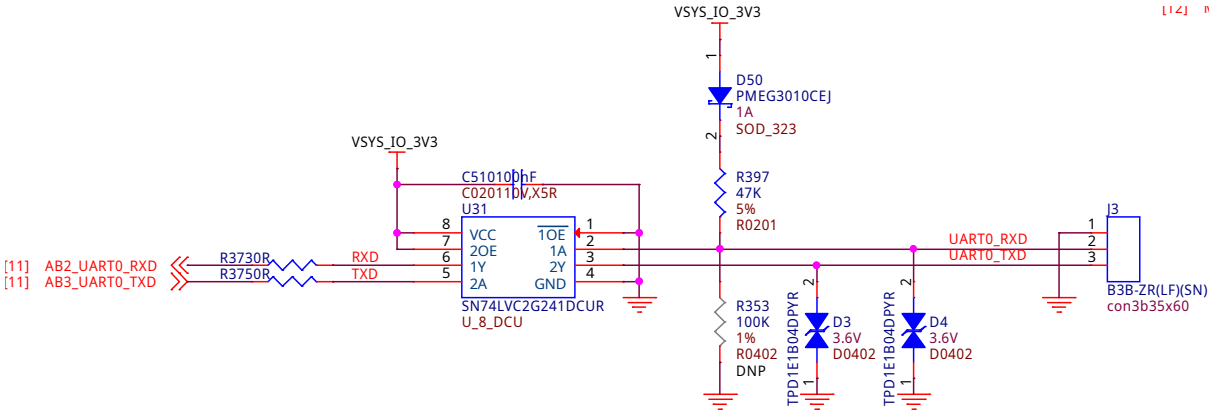


Fig. 3.42: UART0 debug port

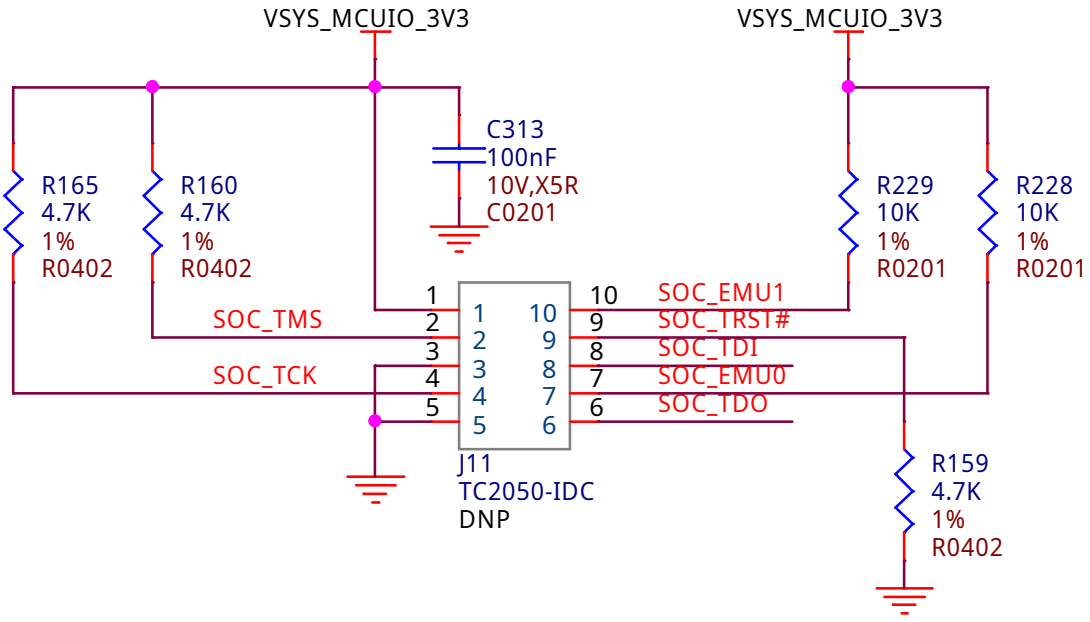


Fig. 3.43: TagConnect JTAG debug port

3.10.1 Dimensions & Weight

Table 3.1: Dimensions & weight

Parameter	Value
Size	104 * 83* 37 mm
Max heigh	23 mm
PCB Size	102.5*80*2.0 mm
PCB Layers	14 layers
PCB Thickness	2.0 mm
RoHS compliant	Yes
Gross Weight	249g
Net Weight	193g

3.10.2 Board Dimensions

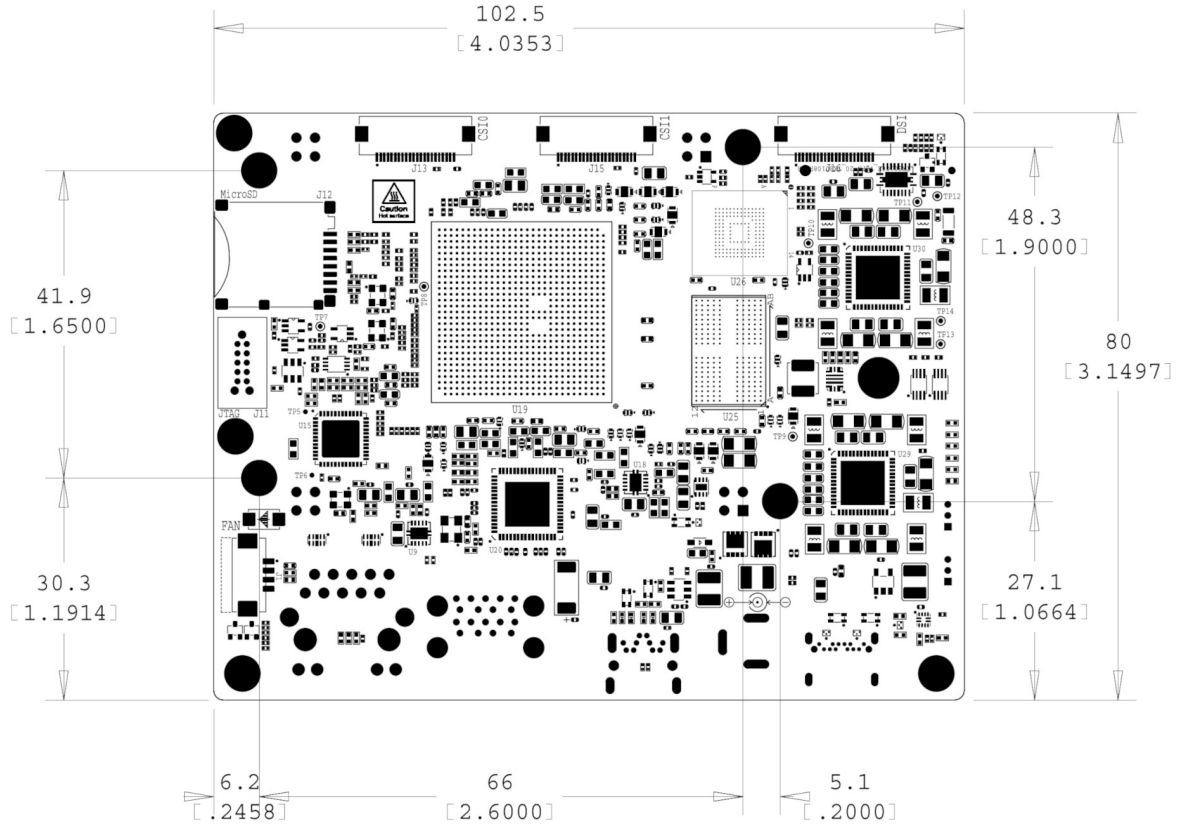
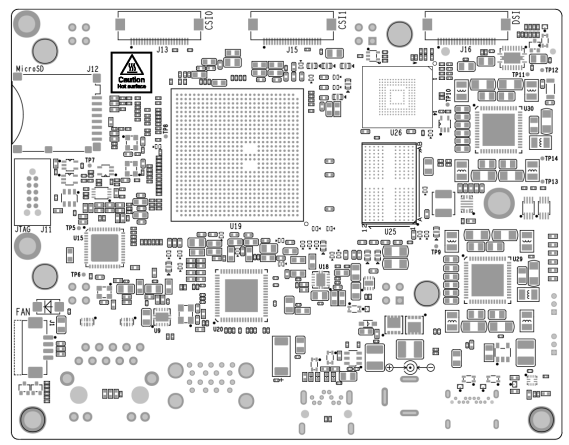
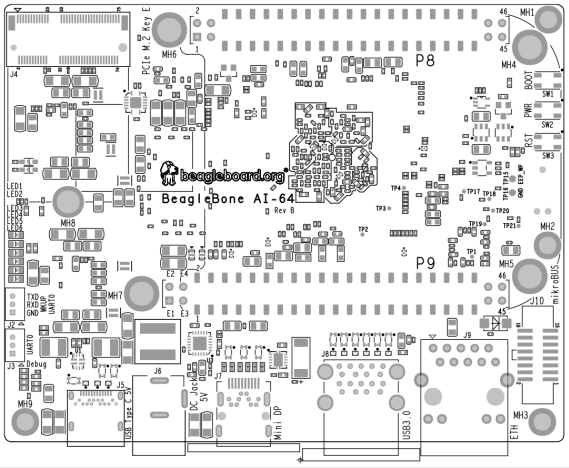


Fig. 3.44: BeagleBone AI-64 board dimensions

3.10.3 PCB silkscreen

Table 3.2: BeagleBone AI-64 silkscreen

Top silkscreen	Bottom silkscreen
	

Chapter 4

Expansion

4.1 Pinout Diagrams

BeagleBone AI-64 P8 & P9 cape headers are designed to be compatible with BeagleBone Black as much as possible. Below pinout diagrams are design to simplify cape header pin usage and cape design process for AI-64. To start using P8 / P9 cape header choose respective pinout diagram tab below.

P8 cape header

P9 cape header

4.2 Cape Header Connectors

Beagle cape expansion interface on the BeagleBone AI-64 like other Beagles is comprised of two headers P8 (46 pin) & P9 (50 pin). All signals on the expansion headers are **3.3V** unless otherwise indicated. **On some of the cape header pins on AI-64 multiple SoC pins are shorted and only one of them should be used at a time.** Information regarding the double/shorted pins is provided in the *Pinout Diagrams* above (simplified) and cape header pin tables below (detailed).

> **Danger:** Do not connect 5V logic level signals to these pins or the board will be damaged.
>
> **NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH. DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.**

4.2.1 Connector P8

The following tables show the pinout of the **P8** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to directly control the GPIO pins with the commands shown below.

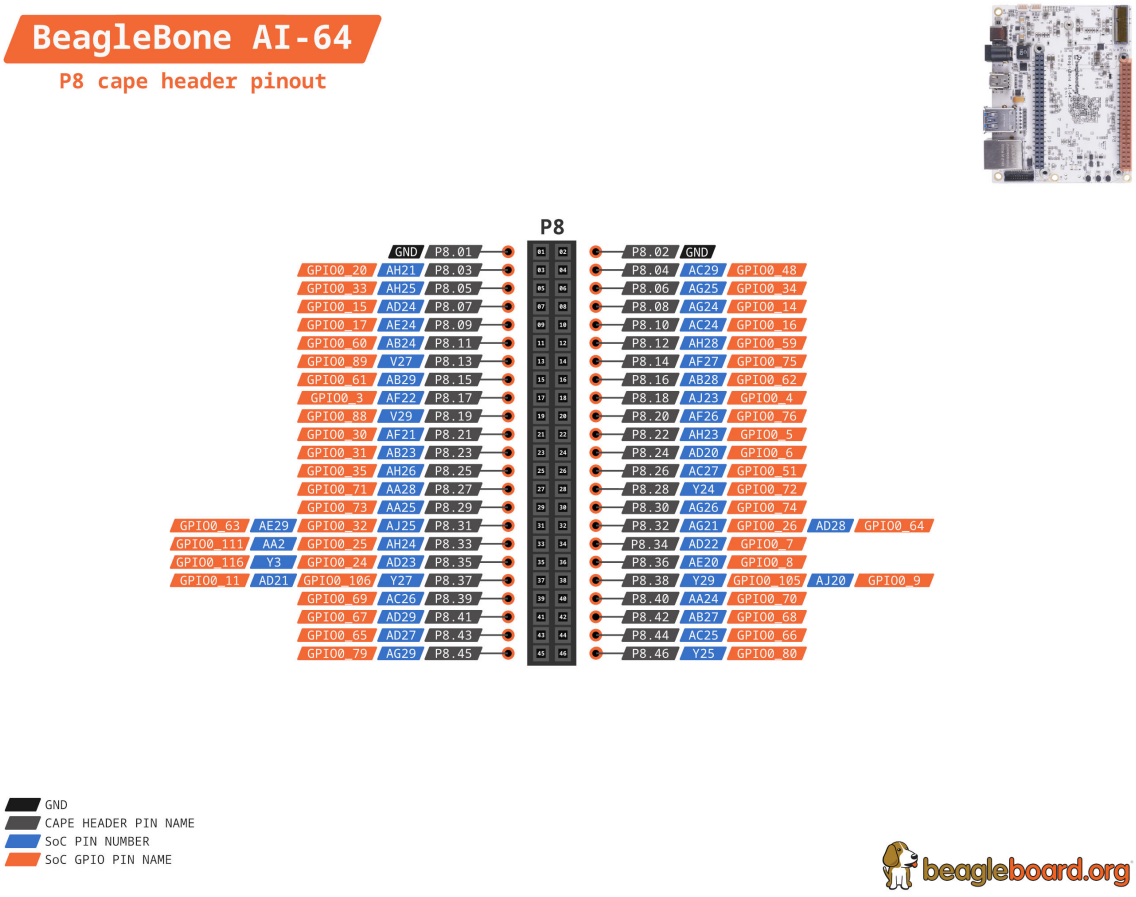


Fig. 4.1: BeagleBone AI-64 P8 cape header pinout

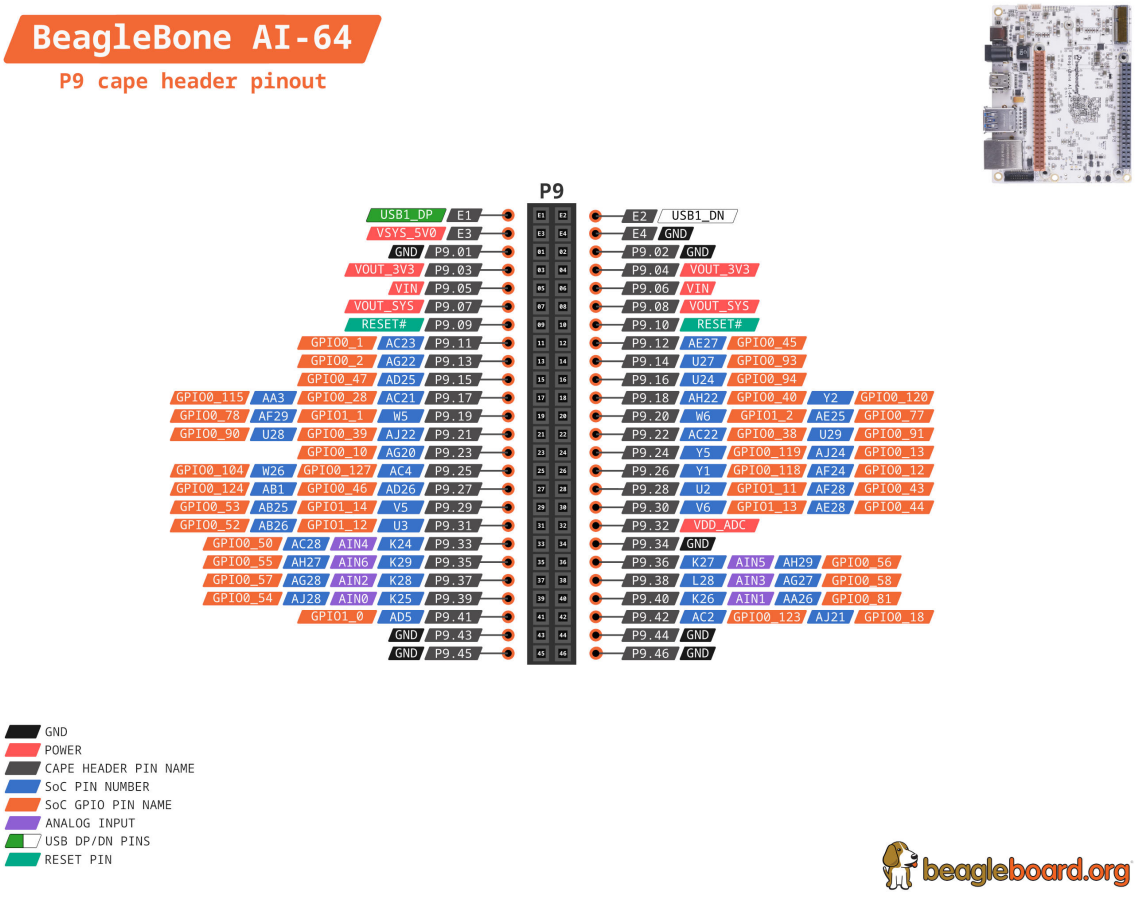


Fig. 4.2: BeagleBone AI-64 P9 cape header pinout

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0

For Example:

+---------+----------+
| Pin     | P8.03    |
+=========+==========+
| GPIO    | 1 20     |
+---------+----------+

Use the commands below for controlling this pin (P8.03) where X = 1 and Y =␣
↪20

# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

Important: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P8.01-P8.02

P8.01	P8.02
GND	GND

P8.03-P8.05

Pin	P8.03	P8.04	P8.05
GPIO	1 20	1 48	1 33
BALL	AH21	AC29	AH25
REG	0x00011C054	0x00011C0C4	0x00011C088
Page	46	30	50
MODE 0	PRG1_PRU0_GPO19	PRG0_PRU0_GPO5	PRG1_PRU1_GPO12
1	PRG1_PRU0_GPI19	PRG0_PRU0_GPI5	PRG1_PRU1_GPI12
2	PRG1_IEP0_EDC_SYNC_OUT0	~	PRG1_RGMII2_TD1
3	PRG1_PWM0_TZ_OUT	PRG0_PWM3_B2	PRG1_PWM1_A0
4	~	~	RGMII2_TD1
5	RMII5_TXD0	RMII3_TXD0	~
6	MCAN6_TX	~	MCAN7_TX
7	GPIO0_20	GPIO0_48	GPIO0_33
8	~	GPMC0_AD0	RGMII8_TD1
9	~	~	~
10	VOUT0_EXTPCLKIN	~	VOUT0_DATA12
11	VPFE0_PCLK	~	~
12	MCASP4_AFSX	MCASP0_AXR3	MCASP9_AFSX
13	~	~	~
14	~	~	~
Bootstrap	~	BOOTMODE2	~

P8.06-P8.09

Pin	P8.06	P8.07	P8.08	P8.09
GPIO	1 34	1 15	1 14	1 17
BALL	AG25	AD24	AG24	AE24
REG	0x00011C08C	0x00011C03C	0x00011C038	0x00011C044
Page	51	44	44	45
MODE 0	PRG1_PRU1_GPO13	PRG1_PRU0_GPO14	PRG1_PRU0_GPO13	PRG1_PRU0_GPO16
1	PRG1_PRU1_GPI13	PRG1_PRU0_GPI14	PRG1_PRU0_GPI13	PRG1_PRU0_GPI16
2	PRG1_RGMII2_TD2	PRG1_RGMII1_TD3	PRG1_RGMII1_TD2	PRG1_RGMII1_TXC
3	PRG1_PWM1_B0	PRG1_PWM0_A1	PRG1_PWM0_B0	PRG1_PWM0_A2
4	RGMII2_TD2	RGMII1_TD3	RGMII1_TD2	RGMII1_TXC
5	~	~	~	~
6	MCAN7_RX	MCAN5_RX	MCAN5_TX	MCAN6_RX
7	GPIO0_34	GPIO0_15	GPIO0_14	GPIO0_17
8	RGMII8_TD2	~	~	~
9	~	RGMII7_TD3	RGMII7_TD2	RGMII7_TXC
10	VOUT0_DATA13	VOUT0_DATA19	VOUT0_DATA18	VOUT0_DATA21
11	VPFE0_DATA8	VPFE0_DATA3	VPFE0_DATA2	VPFE0_DATA5
12	MCASP9_AXR0	MCASP7_AXR1	MCASP7_AXR0	MCASP7_AXR3
13	MCASP4_ACLKR	~	~	MCASP7_AFSR
14	~	~	~	~
Bootstrap	~	~	~	~

P8.10-P8.13

Pin	P8.10	P8.11	P8.12	P8.13
GPIO	1 16	1 60	1 59	1 89
BALL	AC24	AB24	AH28	V27
REG	0x00011C040	0x00011C0F4	0x00011C0F0	0x00011C168
Page	44	33	33	56
MODE 0	PRG1_PRU0_GPO15	PRG0_PRU0_GPO17	PRG0_PRU0_GPO16	RGMII5_TD1
1	PRG1_PRU0_GPI15	PRG0_PRU0_GPI17	PRG0_PRU0_GPI16	RMII7_TXD1
2	PRG1_RGMII1_TX_CTL	PRG0_IEP0_EDC_SYNC_OUT1	PRG0_RGMII1_TXC	I2C3_SCL
3	PRG1_PWM0_B1	PRG0_PWM0_B2	PRG0_PWM0_A2	~
4	RGMII1_TX_CTL	PRG0_ECAP0_SYNC_OUT	RGMII3_TXC	VOUT1_DATA4
5	~	~	~	TRC_DATA2
6	MCAN6_TX	~	~	EHRPWM0_B
7	GPIO0_16	GPIO0_60	GPIO0_59	GPIO0_89
8	~	GPMC0_AD5	~	GPMC0_A5
9	RGMII7_TX_CTL	OBSCLK1	~	~
10	VOUT0_DATA20	~	DSS_FSYNC1	~
11	VPFE0_DATA4	~	~	~
12	MCASP7_AXR2	MCASP0_AXR13	MCASP0_AXR12	MCASP11_ACLKX
13	MCASP7_ACLKR	~	~	~
14	~	~	~	~
Bootstrap	~	BOOTMODE7	~	~

P8.14-P8.16

Pin	P8.14	P8.15	P8.16
GPIO	1 75	1 61	1 62
BALL	AF27	AB29	AB28
REG	0x00011C130	0x00011C0F8	0x00011C0FC
Page	37	33	34
MODE 0	PRG0_PRU1_GPO12	PRG0_PRU0_GPO18	PRG0_PRU0_GPO19
1	PRG0_PRU1_GPI12	PRG0_PRU0_GPI18	PRG0_PRU0_GPI19
2	PRG0_RGMII2_TD1	PRG0_IEP0_EDC_LATCH_IN0	PRG0_IEP0_EDC_SYNC_OUT0
3	PRG0_PWM1_A0	PRG0_PWM0_TZ_IN	PRG0_PWM0_TZ_OUT
4	RGMII4_TD1	PRG0_ECAP0_IN_APWM_OUT	~
5	~	~	~
6	~	~	~
7	GPIO0_75	GPIO0_61	GPIO0_62
8	~	GPMC0_AD6	GPMC0_AD7
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_AXR8	MCASP0_AXR14	MCASP0_AXR15
13	~	~	~
14	UART8_CTSn	~	~
Bootstrap	~	~	~

P8.17-P8.19

Pin	P8.17	P8.18	P8.19
GPIO	1 3	1 4	1 88
BALL	AF22	AJ23	V29
REG	0x00011C00C	0x00011C010	0x00011C164
Page	40	40	57
MODE 0	PRG1_PRU0_GPO2	PRG1_PRU0_GPO3	RGMII5_TD2
1	PRG1_PRU0_GPI2	PRG1_PRU0_GPI3	UART3_TXD
2	PRG1_RGMII1_RD2	PRG1_RGMII1_RD3	~
3	PRG1_PWM2_A0	PRG1_PWM3_A2	SYNC3_OUT
4	RGMII1_RD2	RGMII1_RD3	VOUT1_DATA3
5	RMII1_CRS_DV	RMII1_RX_ER	TRC_DATA1
6	~	~	EHRPWM0_A
7	GPIO0_3	GPIO0_4	GPIO0_88
8	GPMC0_WAIT1	GPMC0_DIR	GPMC0_A4
9	RGMII7_RD2	RGMII7_RD3	~
10	~	~	~
11	~	~	~
12	MCASP6_AXR0	MCASP6_AXR1	MCASP10_AXR1
13	~	~	~
14	UART1_RXD	UART1_TXD	~
Bootstrap	~	~	~

P8.20-P8.22

Pin	P8.20	P8.21	P8.22
GPIO	1 76	1 30	1 5
BALL	AF26	AF21	AH23
REG	0x00011C134	0x00011C07C	0x00011C014
Page	37	49	41
MODE 0	PRG0_PRU1_GPO13	PRG1_PRU1_GPO9	PRG1_PRU0_GPO4
1	PRG0_PRU1_GPI13	PRG1_PRU1_GPI9	PRG1_PRU0_GPI4
2	PRG0_RGMII2_TD2	PRG1_UART0_RXD	PRG1_RGMII1_RX_CTL
3	PRG0_PWM1_B0	~	PRG1_PWM2_B0
4	RGMII4_TD2	SPI6_CS3	RGMII1_RX_CTL
5	~	RMII6_RXD1	RMII1_TXD0
6	~	MCAN8_TX	~
7	GPIO0_76	GPIO0_30	GPIO0_5
8	~	GPMC0_CSn0	GPMC0_CSn2
9	~	PRG1_IEP0_EDIO_DATA_IN_OUT30	RGMII7_RX_CTL
10	~	VOUT0_DATA9	~
11	~	~	~
12	MCASP1_AXR9	MCASP4_AXR3	MCASP6_AXR2
13	~	~	MCASP6_ACLKR
14	UART8_RTSn	~	UART2_RXD
Bootstrap	~	~	~

P8.23-P8.26

Pin	P8.23	P8.24	P8.25	P8.26
GPIO	1 31	1 6	1 35	1 51
BALL	AB23	AD20	AH26	AC27
REG	0x00011C080	0x00011C018	0x00011C090	0x00011C0D0
Page	50	41	51	31
MODE 0	PRG1_PRU1_GPO10	PRG1_PRU0_GPO5	PRG1_PRU1_GPO14	PRG0_PRU0_GPO8
1	PRG1_PRU1_GPI10	PRG1_PRU0_GPI5	PRG1_PRU1_GPI14	PRG0_PRU0_GPI8
2	PRG1_UART0_TXD	~	PRG1_RGMII2_TD3	~
3	PRG1_PWM2_TZ_IN	PRG1_PWM3_B2	PRG1_PWM1_A1	PRG0_PWM2_A1
4	~	~	RGMII2_TD3	~
5	RMII6_CRS_DV	RMII1_TX_EN	~	~
6	MCAN8_RX	~	MCAN8_TX	MCAN9_RX
7	GPIO0_31	GPIO0_6	GPIO0_35	GPIO0_51
8	GPMC0_CLKOUT	GPMC0_WEn	RGMII8_TD3	GPMC0_AD2
9	PRG1_IEP0_EDIO_DATA_IN_OUT31	~	~	~
10	VOUT0_DATA10	~	VOUT0_DATA14	~
11	GPMC0_FCLK_MUX	~	~	~
12	MCASP5_ACLKX	MCASP3_AXR0	MCASP9_AXR1	MCASP0_AXR6
13	~	~	MCASP4_AFSR	~
14	~	~	~	UART6_RXD
Bootstrap	~	BOOTMODE0	~	~

P8.27-P8.29

Pin	P8.27	P8.28	P8.29
GPIO	1 71	1 72	1 73
BALL	AA28	Y24	AA25
REG	0x00011C120	0x00011C124	0x00011C128
Page	36	36	36
MODE 0	PRG0_PRU1_GPO8	PRG0_PRU1_GPO9	PRG0_PRU1_GPO10
1	PRG0_PRU1_GPI8	PRG0_PRU1_GPI9	PRG0_PRU1_GPI10
2	~	PRG0_UART0_RXD	PRG0_UART0_TXD
3	PRG0_PWM2_TZ_OUT	~	PRG0_PWM2_TZ_IN
4	~	SPI3_CS3	~
5	~	~	~
6	MCAN11_RX	PRG0_IEP0_EDIO_DATA_IN_OUT30	PRG0_IEP0_EDIO_DATA_IN_OUT31
7	GPIO0_71	GPIO0_72	GPIO0_73
8	GPMC0_AD10	GPMC0_AD11	GPMC0_AD12
9	~	~	CLKOUT
10	~	DSS_FSYNC3	~
11	~	~	~
12	MCASP1_AFSX	MCASP1_AXR5	MCASP1_AXR6
13	~	~	~
14	~	UART8_RXD	UART8_TXD
Bootstrap	~	~	~

P8.30-P8.32

Pin	P8.30	P8.31	~	P8.32	~
GPIO	1 74	1 32	1 63	1 26	1 64
BALL	AG26	AJ25	AE29	AG21	AD28
REG	0x00011C12C	0x00011C084	0x00011C100	0x00011C06C	0x00011C104
Page	37	50	34	48	34
MODE 0	PRG0_PRU1_GPO11	PRG1_PRU1_GPO11	PRG0_PRU1_GPO0	PRG1_PRU1_GPO5	PRG0_PRU1_GPO1
1	PRG0_PRU1_GPI11	PRG1_PRU1_GPI11	PRG0_PRU1_GPI0	PRG1_PRU1_GPI5	PRG0_PRU1_GPI1
2	PRG0_RGMII2_TD0	PRG1_RGMII2_TD0	PRG0_RGMII2_RD0	~	PRG0_RGMII2_RD1
3	~	~	~	~	~
4	RGMII4_TD0	RGMII2_TD0	RGMII4_RD0	~	RGMII4_RD1
5	RMII4_TX_EN	RMII2_TX_EN	RMII4_RXD0	RMII5_TX_EN	RMII4_RXD1
6	~	~	~	MCAN6_RX	~
7	GPIO0_74	GPIO0_32	GPIO0_63	GPIO0_26	GPIO0_64
8	GPMC0_A26	RGMII8_TD0	UART4_CTSn	GPMC0_WPn	UART4_RTSn
9	~	EQEP1_I	~	EQEP1_S	~
10	~	VOUT0_DATA11	~	VOUT0_DATA5	~
11	~	~	~	~	~
12	MCASP1_AXR7	MCASP9_ACLKX	MCASP1_AXR0	MCASP4_AXR0	MCASP1_AXR1
13	~	~	~	~	~
14	~	~	UART5_RXD	TIMER_IO4	UART5_TXD
Bootstrap	~	~	~	~	~

P8.33-P8.35

Pin	P8.33	~	P8.34	P8.35	~
GPIO	1 25	1 111	1 7	1 24	1 116
BALL	AH24	AA2	AD22	AD23	Y3
REG	0x00011C068	0x00011C1C0	0x00011C01C	0x00011C064	0x00011C1D4
Page	48	67	41	47	67
MODE 0	PRG1_PRU1_GPO4	SPI0_CS0	PRG1_PRU0_GPO6	PRG1_PRU1_GPO3	SPI1_CS0
1	PRG1_PRU1_GPI4	UART0_RTSn	PRG1_PRU0_GPI6	PRG1_PRU1_GPI3	UART0_CTSn
2	PRG1_RGMII2_RX_CTL	~	PRG1_RGMII1_RXC	PRG1_RGMII2_RD3	~
3	PRG1_PWM2_B2	~	PRG1_PWM3_A1	~	UART5_RXD
4	RGMII2_RX_CTL	~	RGMII1_RXC	RGMII2_RD3	~
5	RMII2_TXD0	~	RMII1_TXD1	RMII2_RX_ER	~
6	~	~	AUDIO_EXT_REFCLK0	~	PRG0_IEP0_EDIO_OUTVALID
7	GPIO0_25	GPIO0_111	GPIO0_7	GPIO0_24	GPIO0_116
8	RGMII8_RX_CTL	~	GPMC0_CSn3	RGMII8_RD3	PRG0_IEP0_EDC_LATCH_IN0
9	EQEP1_B	~	RGMII7_RXC	EQEP1_A	~
10	VOUT0_DATA4	~	~	VOUT0_DATA3	~
11	VPFE0_DATA13	~	~	VPFE0_WEN	~
12	MCASP8_AXR2	~	MCASP6_AXR3	MCASP8_AXR1	~
13	MCASP8_ACLKR	~	MCASP6_AFSR	MCASP3_AFSR	~
14	TIMER_IO3	~	UART2_TXD	TIMER_IO2	~
Bootstrap	~	~	~	~	~

P8.36-P8.38

Pin	P8.36	P8.37	~	P8.38	~
GPIO	1 8	1 106	1 11	1 105	1 9
BALL	AE20	Y27	AD21	Y29	AJ20
REG	0x00011C020	0x00011C1AC	0x00011C02C	0x00011C1A8	0x00011C024
Page	42	58	43	58	42
MODE 0	PRG1_PRU0_GPO7	RGMII6_RD2	PRG1_PRU0_GPO10	RGMII6_RD3	PRG1_PRU0_GPO8
1	PRG1_PRU0_GPI7	UART4_RTSn	PRG1_PRU0_GPI10	UART4_CTSn	PRG1_PRU0_GPI8
2	PRG1_IEP0_EDC_LATCH_IN1	~	PRG1_UART0_RTSn	~	~
3	PRG1_PWM3_B1	UART5_TXD	PRG1_PWM2_B1	UART5_RXD	PRG1_PWM2_A1
4	~	~	SPI6_CS2	CLKOUT	~
5	AUDIO_EXT_REFCLK1	TRC_DATA19	RMII5_CRS_DV	TRC_DATA18	RMII5_RXD0
6	MCAN4_TX	EHRPWM5_A	~	EHRPWM_TZn_IN4	MCAN4_RX
7	GPIO0_8	GPIO0_106	GPIO0_11	GPIO0_105	GPIO0_9
8	~	GPMC0_A22	GPMC0_BE0n_CLE	GPMC0_A21	GPMC0_OEn_REn
9	~	~	PRG1_IEP0_EDIO_DATA_IN_OUT29	~	~
10	~	~	OBSCLK2	~	VOUT0_DATA22
11	~	~	~	~	~
12	MCASP3_AXR1	MCASP11_AXR5	MCASP3_AFSX	MCASP11_AXR4	MCASP3_AXR2
13	~	~	~	~	~
14	~	~	~	~	~
Boot-strap	~	~	~	~	~

P8.39-P8.41

Pin	P8.39	P8.40	P8.41
GPIO	1 69	1 70	1 67
BALL	AC26	AA24	AD29
REG	0x00011C118	0x00011C11C	0x00011C110
Page	35	36	35
MODE 0	PRG0_PRU1_GPO6	PRG0_PRU1_GPO7	PRG0_PRU1_GPO4
1	PRG0_PRU1_GPI6	PRG0_PRU1_GPI7	PRG0_PRU1_GPI4
2	PRG0_RGMII2_RXC	PRG0_IEP1_EDC_LATCH_IN1	PRG0_RGMII2_RX_CTL
3	~	~	PRG0_PWM2_B2
4	RGMII4_RXC	SPI3_CS0	RGMII4_RX_CTL
5	RMII4_TXD0	~	RMII4_TXD1
6	~	MCAN11_TX	~
7	GPIO0_69	GPIO0_70	GPIO0_67
8	GPMC0_A25	GPMC0_AD9	GPMC0_A24
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_AXR3	MCASP1_AXR4	MCASP1_AXR2
13	~	~	~
14	~	UART2_TXD	~
Bootstrap	~	~	~

P8.42-P8.44

Pin	P8.42	P8.43	P8.44
GPIO	1 68	1 65	1 66
BALL	AB27	AD27	AC25
REG	0x00011C114	0x00011C108	0x00011C10C
Page	35	34	35
MODE 0	PRG0_PRU1_GPO5	PRG0_PRU1_GPO2	PRG0_PRU1_GPO3
1	PRG0_PRU1_GPI5	PRG0_PRU1_GPI2	PRG0_PRU1_GPI3
2	~	PRG0_RGMII2_RD2	PRG0_RGMII2_RD3
3	~	PRG0_PWM2_A2	~
4	~	RGMII4_RD2	RGMII4_RD3
5	~	RMII4_CRS_DV	RMII4_RX_ER
6	~	~	~
7	GPIO0_68	GPIO0_65	GPIO0_66
8	GPMC0_AD8	GPMC0_A23	~
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP1_ACLKX	MCASP1_ACLKR	MCASP1_AFSR
13	~	MCASP1_AXR10	MCASP1_AXR11
14	~	~	~
Bootstrap	BOOTMODE6	~	~

P8.45-P8.46

Pin	P8.45	P8.46
GPIO	1 79	1 80
BALL	AG29	Y25
REG	0x00011C140	0x00011C144
Page	38	38
MODE 0	PRG0_PRU1_GPO16	PRG0_PRU1_GPO17
1	PRG0_PRU1_GPI16	PRG0_PRU1_GPI17
2	PRG0_RGMII2_TXC	PRG0_IEP1_EDC_SYNC_OUT1
3	PRG0_PWM1_A2	PRG0_PWM1_B2
4	RGMII4_TXC	SPI3_CLK
5	~	~
6	~	~
7	GPIO0_79	GPIO0_80
8	~	GPMC0_AD13
9	~	~
10	~	~
11	~	~
12	MCASP2_AXR2	MCASP2_AXR3
13	~	~
14	~	~
Bootstrap	~	BOOTMODE3

4.2.2 Connector P9

The following tables show the pinout of the **P9** expansion header. The SW is responsible for setting the default function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **GPIO** row is the expected gpio identifier number in the Linux kernel.

Each row includes the gpiochipX and pinY in the format of *X Y*. You can use these values to directly control the GPIO pins with the commands shown below.

```
# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset X Y=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset X Y=0

For Example:

+---------+----------+
| Pin     | P9.11    |
+=========+==========+
| GPIO    | 1 1      |
+---------+----------+

Use the commands below for controlling this pin (P9.11) where X = 1 and Y = 1

# to set the GPIO pin state to HIGH
debian@BeagleBone:~$ gpioset 1 20=1

# to set the GPIO pin state to LOW
debian@BeagleBone:~$ gpioset 1 20=0
```

The **BALL** row is the pin number on the processor.

The **REG** row is the offset of the control register for the processor pin.

The **MODE #** rows are the mode setting for each pin. Setting each mode to align with the mode column will give that function on that pin.

If included, the **2nd BALL** row is the pin number on the processor for a second processor pin connected to the same pin on the expansion header. Similarly, all row headings starting with **2nd** refer to data for this second processor pin.

Important: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

P9.E1-P9.E4

E1	E2	E3	E4
USB1 DP	USB1 DN	VSYS_5V0	GND

P9.01-P9.05

P9.01	P9.02	P9.03	P9.04	P9.05
GND	GND	VOUT_3V3	VOUT_3V3	VIN

P9.06-P9.10

P9.06	P9.07	P9.08	P9.09	P9.10
VIN	VOUT_SYS | VOUT_SYS		RESET#	RESET#

P9.11-P9.13

Pin	P9.11	P9.12	P9.13
GPIO	1 1	1 45	1 2
BALL	AC23	AE27	AG22
REG	0x00011C004	0x00011C0B8	0x00011C008
Page	39	29	40
MODE 0	PRG1_PRU0_GPO0	PRG0_PRU0_GPO2	PRG1_PRU0_GPO1
1	PRG1_PRU0_GPI0	PRG0_PRU0_GPI2	PRG1_PRU0_GPI1
2	PRG1_RGMII1_RD0	PRG0_RGMII1_RD2	PRG1_RGMII1_RD1
3	PRG1_PWM3_A0	PRG0_PWM2_A0	PRG1_PWM3_B0
4	RGMII1_RD0	RGMII3_RD2	RGMII1_RD1
5	RMII1_RXD0	RMII3_CRS_DV	RMII1_RXD1
6	~	~	~
7	GPIO0_1	GPIO0_45	GPIO0_2
8	GPMC0_BE1n	UART3_RXD	GPMC0_WAIT0
9	RGMII7_RD0	~	RGMII7_RD1
10	~	~	~
11	~	~	~
12	MCASP6_ACLKX	MCASP0_ACLKR	MCASP6_AFSX
13	~	~	~
14	UART0_RXD	~	UART0_TXD
Bootstrap	~	~	~

P9.14-P9.16

Pin	P9.14	P9.15	P9.16
GPIO	1 93	1 47	1 94
BALL	U27	AD25	U24
REG	0x00011C178	0x00011C0C0	0x00011C17C
Page	56	30	56
MODE 0	RGMII5_RD3	PRG0_PRU0_GPO4	RGMII5_RD2
1	UART3_CTSn	PRG0_PRU0_GPI4	UART3_RTSn
2	~	PRG0_RGMII1_RX_CTL	~
3	UART6_RXD	PRG0_PWM2_B0	UART6_TXD
4	VOUT1_DATA8	RGMII3_RX_CTL	VOUT1_DATA9
5	TRC_DATA6	RMII3_TXD1	TRC_DATA7
6	EHRPWM2_A	~	EHRPWM2_B
7	GPIO0_93	GPIO0_47	GPIO0_94
8	GPMC0_A9	~	GPMC0_A10
9	~	~	~
10	~	~	~
11	~	~	~
12	MCASP11_AXR0	MCASP0_AXR2	MCASP11_AXR1
13	~	~	~
14	~	~	~
Bootstrap	~	~	~

P9.17-P9.18

Pin	P9.17	~	P9.18	~
GPIO	1 28	1 115	1 40	1 120
BALL	AC21	AA3	AH22	Y2
REG	0x00011C074	0x00011C1D0	0x00011C0A4	0x00011C1E4
Page	49	67	53	68
MODE 0	PRG1_PRU1_GPO7	SPI0_D1	PRG1_PRU1_GPO19	SPI1_D1
1	PRG1_PRU1_GPI7	~	PRG1_PRU1_GPI19	~
2	PRG1_IEP1_EDC_LATCH_IN1	I2C6_SCL	PRG1_IEP1_EDC_SYNC_OUT0	I2C6_SDA
3	~	~	PRG1_PWM1_TZ_OUT	~
4	SPI6_CS0	~	SPI6_D1	~
5	RMII6_RX_ER	~	RMII6_TXD1	~
6	MCAN7_TX	~	PRG1_ECAP0_IN_APWM_OUT	~
7	GPIO0_28	GPIO0_115	GPIO0_40	GPIO0_120
8	~	~	~	PRG0_IEP1_EDC_SYNC_OUT0
9	~	~	~	~
10	VOUT0_DATA7	~	VOUT0_PCLK	~
11	VPFE0_DATA15	~	~	~
12	MCASP4_AXR1	~	MCASP5_AXR1	~
13	~	~	~	~
14	UART3_TXD	~	~	~
Bootstrap	~	~	~	~

P9.19-P9.20

Pin	P9.19	~	P9.20	~
GPIO	2 1	1 78	2 2	1 77
BALL	W5	AF29	W6	AE25
REG	0x00011C208	0x00011C13C	0x00011C20C	0x00011C138
Page	19	38	19	37
MODE 0	MCAN0_RX	PRG0_PRU1_GPO15	MCAN0_TX	PRG0_PRU1_GPO14
1	~	PRG0_PRU1_GPI15	~	PRG0_PRU1_GPI14
2	~	PRG0_RGMII2_TX_CTL	~	PRG0_RGMII2_TD3
3	~	PRG0_PWM1_B1	~	PRG0_PWM1_A1
4	I2C2_SCL	RGMII4_TX_CTL	I2C2_SDA	RGMII4_TD3
5	~	~	~	~
6	~	~	~	~
7	GPIO1_1	GPIO0_78	GPIO1_2	GPIO0_77
8	~	~	~	~
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP2_AXR1	~	MCASP2_AXR0
13	~	~	~	~
14	~	UART2_RTSn	~	UART2_CTSn
Bootstrap	~	~	~	~

P9.21-P9.22

Pin	P9.21	~	P9.22	~
GPIO	1 39	1 90	1 38	1 91
BALL	AJ22	U28	AC22	U29
REG	0x00011C0A0	0x00011C16C	0x00011C09C	0x00011C170
Page	52	56	52	54
MODE 0	PRG1_PRU1_GPO18	RGMII5_TD0	PRG1_PRU1_GPO17	RGMII5_TXC
1	PRG1_PRU1_GPI18	RMII7_TXD0	PRG1_PRU1_GPI17	RMII7_TX_EN
2	PRG1_IEP1_EDC_LATCH_IN0	I2C3_SDA	PRG1_IEP1_EDC_SYNC_OUT1	I2C6_SCL
3	PRG1_PWM1_TZ_IN	~	PRG1_PWM1_B2	~
4	SPI6_D0	VOUT1_DATA5	SPI6_CLK	VOUT1_DATA6
5	RMII6_TXD0	TRC_DATA3	RMII6_TX_EN	TRC_DATA4
6	PRG1_ECAP0_SYNC_IN	EHRPWM1_A	PRG1_ECAP0_SYNC_OUT	EHRPWM1_B
7	GPIO0_39	GPIO0_90	GPIO0_38	GPIO0_91
8	~	GPMC0_A6	~	GPMC0_A7
9	VOUT0_VP2_VSYNC	~	VOUT0_VP2_DE	~
10	VOUT0_VSYNC	~	VOUT0_DE	~
11	~	~	VPFE0_DATA10	~
12	MCASP5_AXR0	MCASP11_AFSX	MCASP5_AFSX	MCASP10_AXR2
13	~	~	~	~
14	VOUT0_VP0_VSYNC	~	VOUT0_VP0_DE	~
Bootstrap	~	~	BOOTMODE1	~

P9.23-P9.25

Pin	P9.23	P9.24	~	P9.25	~
GPIO	1 10	1 119	1 13	1 127	1 104
BALL	AG20	Y5	AJ24	AC4	W26
REG	0x00011C028	0x00011C1E0	0x00011C034	0x00011C200	0x00011C1A4
Page	42	68	43	69	54
MODE 0	PRG1_PRU0_GPO9	SPI1_D0	PRG1_PRU0_GPO12	UART1_CTSn	RGMII6_RXC
1	PRG1_PRU0_GPI9	UART5_RTSn	PRG1_PRU0_GPI12	MCAN3_RX	~
2	PRG1_UART0_CTSn	I2C4_SCL	PRG1_RGMII1_TD1	~	~
3	PRG1_PWM3_TZ_IN	UART2_TXD	PRG1_PWM0_A0	~	AUDIO_EXT_REFCLK2
4	SPI6_CS1	~	RGMII1_TD1	SPI2_D0	VOUT1_DE
5	RMII5_RXD1	~	~	EQEP0_S	TRC_DATA17
6	~	~	MCAN4_RX	~	EHRPWM4_B
7	GPIO0_10	GPIO0_119	GPIO0_13	GPIO0_127	GPIO0_104
8	GPMC0_ADVn_ALE	PRG0_IEP1_EDC_LATCH_IN0	~	~	GPMC0_A20
9	PRG1_IEP0_EDIO_DATA_IN_OUT2	~	RGMII7_TD1	~	VOUT1_VP0_DE
10	VOUT0_DATA23	~	VOUT0_DATA17	~	~
11	~	~	VPFE0_DATA1	~	~
12	MCASP3_ACLKX	~	MCASP7_AFSX	~	MCASP10_AXR7
13	~	~	~	~	~
14	~	~	~	~	~
Bootstrap	~	~	~	~	~

P9.26-P9.27

Pin	P9.26	~	P9.27	~
GPIO	1 118	1 12	1 46	1 124
BALL	Y1	AF24	AD26	AB1
REG	0x00011C1DC	0x00011C030	0x00011C0BC	0x00011C1F4
Page	67	43	30	69
MODE 0	SPI1_CLK	PRG1_PRU0_GPO11	PRG0_PRU0_GPO3	UART0_RTSn
1	UART5_CTSn	PRG1_PRU0_GPI11	PRG0_PRU0_GPI3	TIMER_IO7
2	I2C4_SDA	PRG1_RGMII1_TD0	PRG0_RGMII1_RD3	SPI0_CS3
3	UART2_RXD	PRG1_PWM3_TZ_OUT	PRG0_PWM3_A2	MCAN2_TX
4	~	RGMII1_TD0	RGMII3_RD3	SPI2_CLK
5	~	~	RMII3_RX_ER	EQEP0_B
6	~	MCAN4_TX	~	~
7	GPIO0_118	GPIO0_12	GPIO0_46	GPIO0_124
8	PRG0_IEP0_EDC_SYNC_OUT0	~	UART3_TXD	~
9	~	RGMII7_TD0	~	~
10	~	VOUT0_DATA16	~	~
11	~	VPFE0_DATA0	~	~
12	~	MCASP7_ACLKX	MCASP0_AFSR	~
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.28-P9.29

Pin	P9.28	~	P9.29	~
GPIO	2 11	1 43	2 14	1 53
BALL	U2	AF28	V5	AB25
REG	0x00011C230	0x00011C0B0	0x00011C23C	0x00011C0D8
Page	18	29	68	31
MODE 0	ECAP0_IN_APWM_OUT	PRG0_PRU0_GPO0	TIMER_IO1	PRG0_PRU0_GPO10
1	SYNC0_OUT	PRG0_PRU0_GPI0	ECAP2_IN_APWM_OUT	PRG0_PRU0_GPI10
2	CPTS0_RFT_CLK	PRG0_RGMII1_RD0	OBSCLK0	PRG0_UART0_RTSn
3	~	PRG0_PWM3_A0	~	PRG0_PWM2_B1
4	SPI2_CS3	RGMII3_RD0	~	SPI3_CS2
5	I3C0_SDAPULLEN	RMII3_RXD1	~	PRG0_IEP0_EDIO_DATA_IN_OUT29
6	SPI7_CS0	~	SPI7_D1	MCAN10_RX
7	GPIO1_11	GPIO0_43	GPIO1_14	GPIO0_53
8	~	~	~	GPMC0_AD4
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP0_AXR0	~	MCASP0_AFSX
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	BOOTMODE5	~

P9.30-P9.31

Pin	P9.30	~	P9.31	~
GPIO	2 13	1 44	2 12	1 52
BALL	V6	AE28	U3	AB26
REG	0x00011C238	0x00011C0B4	0x00011C234	0x00011C0D4
Page	68	29	18	31
MODE 0	TIMER_IO0	PRG0_PRU0_GPO1	EXT_REFCLK1	PRG0_PRU0_GPO9
1	ECAP1_IN_APWM_OUT	PRG0_PRU0_GPI1	SYNC1_OUT	PRG0_PRU0_GPI9
2	SYSCLKOUT0	PRG0_RGMII1_RD1	~	PRG0_UART0_CTSn
3	~	PRG0_PWM3_B0	~	PRG0_PWM3_TZ_IN
4	~	RGMII3_RD1	~	SPI3_CS1
5	~	RMII3_RXD0	~	PRG0_IEP0_EDIO_DATA_IN_OUT28
6	SPI7_D0	~	SPI7_CLK	MCAN10_TX
7	GPIO1_13	GPIO0_44	GPIO1_12	GPIO0_52
8	~	~	~	GPMC0_AD3
9	~	~	~	~
10	~	~	~	~
11	~	~	~	~
12	~	MCASP0_AXR1	~	MCASP0_ACLKX
13	~	~	~	~
14	~	~	~	UART6_TXD
Bootstrap	BOOTMODE4	~	~	~

P9.32-P9.35

P9.32	P9.34
VDD_ADC	GND

Pin	P9.33	~	P9.35	~
GPIO	~	1 50	~	1 55
BALL	K24	AC28	K29	AH27
REG	0x00011C140	0x00011C0CC	0x00011C148	0x00011C0E0
Page	20	31	20	32
MODE 0	MCU_ADC0_AIN4	PRG0_PRU0_GPO7	MCU_ADC0_AIN6	PRG0_PRU0_GPO12
1	~	PRG0_PRU0_GPI7	~	PRG0_PRU0_GPI12
2	~	PRG0_IEP0_EDC_LATCH_IN1	~	PRG0_RGMII1_TD1
3	~	PRG0_PWM3_B1	~	PRG0_PWM0_A0
4	~	PRG0_ECAP0_SYNC_IN	~	RGMII3_TD1
5	~	~	~	~
6	~	MCAN9_TX	~	~
7	~	GPIO0_50	~	GPIO0_55
8	~	GPMC0_AD1	~	~
9	~	~	~	~
10	~	~	~	DSS_FSYNC0
11	~	~	~	~
12	~	MCASP0_AXR5	~	MCASP0_AXR8
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.36-P9.37

Pin	P9.36	~	P9.37	~
GPIO	~	1 56	~	1 57
BALL	K27	AH29	K28	AG28
REG	0x00011C144	0x00011C0E4	0x00011C138	0x00011C0E8
Page	20	32	20	32
MODE 0	MCU_ADC0_AIN5	PRG0_PRU0_GPO13	MCU_ADC0_AIN2	PRG0_PRU0_GPO14
1	~	PRG0_PRU0_GPI13	~	PRG0_PRU0_GPI14
2	~	PRG0_RGMII1_TD2	~	PRG0_RGMII1_TD3
3	~	PRG0_PWM0_B0	~	PRG0_PWM0_A1
4	~	RGMII3_TD2	~	RGMII3_TD3
5	~	~	~	~
6	~	~	~	~
7	~	GPIO0_56	~	GPIO0_57
8	~	~	~	UART4_RXD
9	~	~	~	~
10	~	DSS_FSYNC2	~	~
11	~	~	~	~
12	~	MCASP0_AXR9	~	MCASP0_AXR10
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.38-P9.39

Pin	P9.38	~	P9.39	~
GPIO	~	1 58	~	1 54
BALL	L28	AG27	K25	AJ28
REG	0x00011C13C	0x00011C0EC	0x00011C130	0x00011C0DC
Page	~	33	20	32
MODE 0	MCU_ADC0_AIN3	PRG0_PRU0_GPO15	MCU_ADC0_AIN0	PRG0_PRU0_GPO11
1	~	PRG0_PRU0_GPI15	~	PRG0_PRU0_GPI11
2	~	PRG0_RGMII1_TX_CTL	~	PRG0_RGMII1_TD0
3	~	PRG0_PWM0_B1	~	PRG0_PWM3_TZ_OUT
4	~	RGMII3_TX_CTL	~	RGMII3_TD0
5	~	~	~	~
6	~	~	~	~
7	~	GPIO0_58	~	GPIO0_54
8	~	UART4_TXD	~	~
9	~	~	~	CLKOUT
10	~	DSS_FSYNC3	~	~
11	~	~	~	~
12	~	MCASP0_AXR11	~	MCASP0_AXR7
13	~	~	~	~
14	~	~	~	~
Bootstrap	~	~	~	~

P9.40-P9.42

Pin	P9.40	~	P9.41	P9.42	~
GPIO	~	1 81	2 0	1 123	1 18
BALL	K26	AA26	AD5	AC2	AJ21
REG	0x00011C134	0x00011C148	0x00011C204	0x00011C1F0	0x00011C04C
Page	20	38	69	68	45
MODE 0	MCU_ADC0_AIN1	PRG0_PRU1_GPO18	UART1_RTSn	UART0_CTSn	PRG1_PRU0_GPO17
1	~	PRG0_PRU1_GPI18	MCAN3_TX	TIMER_IO6	PRG1_PRU0_GPI17
2	~	PRG0_IEP1_EDC_LATCH_IN0	~	SPI0_CS2	PRG1_IEP0_EDC_SYNC_OUT1
3	~	PRG0_PWM1_TZ_IN	~	MCAN2_RX	PRG1_PWM0_B2
4	~	SPI3_D0	SPI2_D1	SPI2_CS0	~
5	~	~	EQEP0_I	EQEP0_A	RMII5_TXD1
6	~	MCAN12_TX	~	~	MCAN5_TX
7	~	GPIO0_81	GPIO1_0	GPIO0_123	GPIO0_18
8	~	GPMC0_AD14	~	~	~
9	~	~	~	~	~
10	~	~	~	~	~
11	~	~	~	~	VPFE0_DATA6
12	~	MCASP2_AFSX	~	~	MCASP3_AXR3
13	~	~	~	~	~
14	~	UART2_RXD	~	~	~
Bootstrap	~	~	~	~	~

P9.43-P9.46

P9.43	P9.44	P9.45	P9.46
GND	GND	GND	GND

4.2.3 Cape Board Support

BeagleBone AI-64 has the ability to accept up to four EEPROM addressable expansion boards or capes stacked onto the expansion headers. The word cape comes from the shape of the expansion board for BeagleBone boards as it is fitted around the Ethernet connector on the main board. For BeagleBone this notch acts as a key to ensure proper orientation of the cape. On AI-64 you can see a clear silkscreen marking for the cape orientation. Most of BeagleBone capes can be used with your BeagleBone AI-64 also like shown in *BeagleBone AI-64 cape placement* below.

This section describes the rules & guidelines for creating capes to ensure proper operation with BeagleBone AI-64 and proper interoperability with other capes that are intended to coexist with each other. Co-existence is not a requirement and is in itself, something that is impossible to control or administer. But, people will be able to create capes that operate with other capes that are already available based on public information as it pertains to what pins and features each cape uses. This information will be able to be read from the EEPROM on each cape.

For those wanting to create their own capes this should not put limits on the creation of capes and what they can do, but may set a few basic rules that will allow the software to administer their operation with BeagleBone AI-64. For this reason there is a lot of flexibility in the specification that we hope most people will find it liberating in the spirit of Open Source Hardware. On the other hand we are sure that there are others who would like to see tighter control, more details, more rules and much more order to the way capes are handled.

Over time, this specification will change and be updated, so please refer to the latest version of this manual prior to designing your own capes to get the latest information.

Warning: Do not apply voltage to any I/O pin when power is not supplied to the board. It will damage the processor and void the warranty.

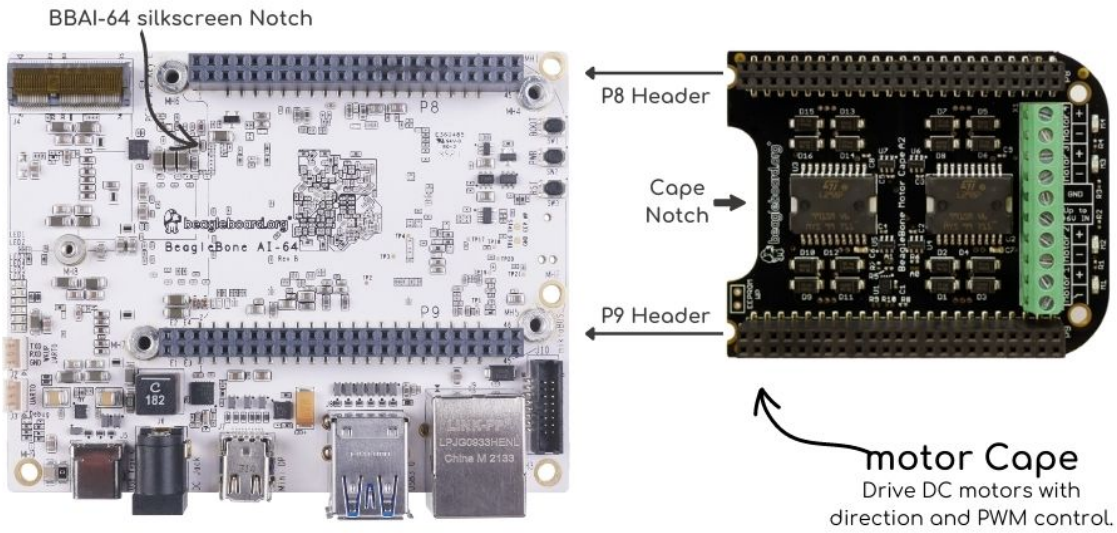


Fig. 4.3: BeagleBone AI-64 cape placement

BeagleBone AI-64 Cape Compatibility

The expansion headers on BeagleBone Black and BeagleBone AI-64 provides similar pin configuration options on P8 and P9 expansion header pins thus provide cape compatibility to a certain extent. Which means most BeagleBone Black capes will also be compatible with BeeagleBone AI-64.

See beaglebone-cape-interface-spec for compatibility information.

EEPROM

Each cape must have its own EEPROM containing information that will allow the software to identify the board and to configure the expansion headers pins during boot as needed. The one exception is proto boards intended for prototyping. They may or may not have an EEPROM on them. An EEPROM is required for all capes sold in order for them operate correctly when plugged into BeagleBone AI-64.

The address of the EEPROM will be set via either jumpers or a dipswitch on each expansion board. *Expansion board EEPROM without write protect* below is the design of the EEPROM circuit.

The addressing of this device requires two bytes for the address which is not used on smaller size EEPROMs, which only require only one byte. Other compatible devices may be used as well. Make sure the device you select supports 16 bit addressing. The part package used is at the discretion of the cape designer.

EEPROM Address In order for each cape to have a unique address, a board ID scheme is used that sets the address to be different depending on the setting of the dipswitch or jumpers on the capes. A two position dipswitch or jumpers is used to set the address pins of the EEPROM.

It is the responsibility of the user to set the proper address for each board and the position in the stack that the board occupies has nothing to do with which board gets first choice on the usage of the expansion bus signals. The process for making that determination and resolving conflicts is left up to the SW and, as of this moment in time, this method is a something of a mystery due to the new Device Tree methodology introduced in the 3.8 kernel.

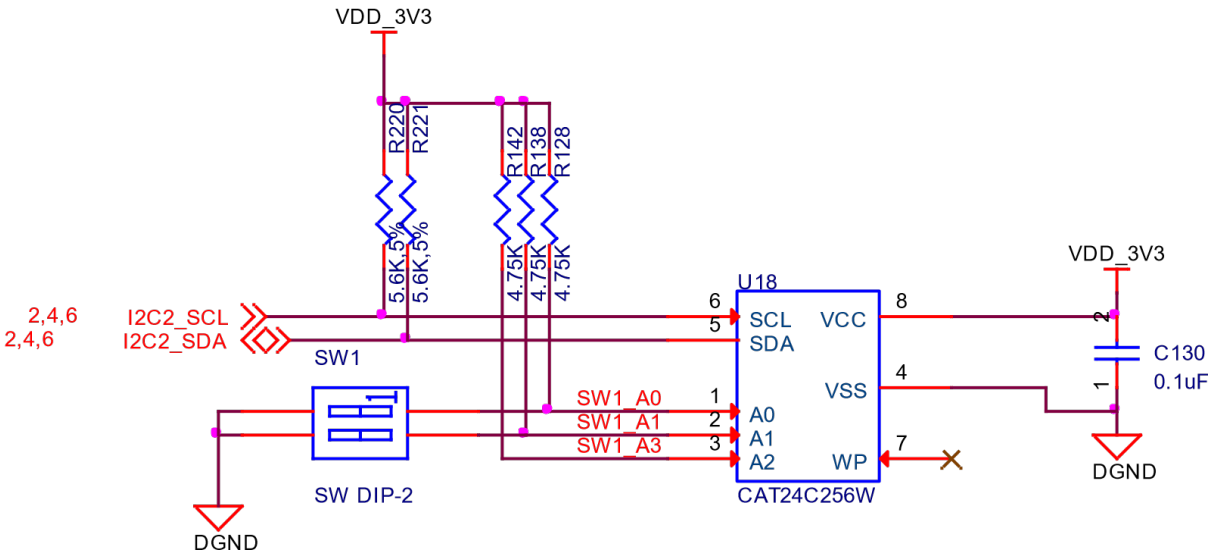


Fig. 4.4: Expansion board EEPROM without write protect

Address line A2 is always tied high. This sets the allowable address range for the expansion cards to *0x54* to**0x57**. All other I2C addresses can be used by the user in the design of their capes. But, these addresses must not be used other than for the board EEPROM information. This also allows for the inclusion of EEPROM devices on the cape if needed without interfering with this EEPROM. It requires that A2 be grounded on the EEPROM not used for cape identification.

I2C Bus The EEPROMs on each expansion board are connected to I2C2 on connector P9 pins 19 and 20. For this reason I2C2 must always be left connected and should not be changed by SW to remove it from the expansion header pin mux settings. If this is done, the system will be unable to detect the capes.

The I2C signals require pullup resistors. Each board must have a 5.6K resistor on these signals. With four capes installed this will result in an effective resistance of 1.4K if all capes were installed and all the resistors used were exactly 5.6K. As more capes are added the resistance is reduced to overcome capacitance added to the signals. When no capes are installed the internal pullup resistors must be activated inside the processor to prevent I2C timeouts on the I2C bus.

The I2C2 bus may also be used by capes for other functions such as I/O expansion or other I2C compatible devices that do not share the same address as the cape EEPROM.

EEPROM Write Protect The design in *Expansion board EEPROM with write protect* has the write protect disabled. If the write protect is not enabled, this does expose the EEPROM to being corrupted if the I2C2 bus is used on the cape and the wrong address written to. It is recommended that a write protection function be implemented and a Test Point be added that when grounded, will allow the EEPROM to be written to. To enable write operation, Pin 7 of the EEPROM must be tied to ground.

When not grounded, the pin is HI via pullup resistor R210 and therefore write protected. Whether or not Write Protect is provided is at the discretion of the cape designer.

Todo:

- Variable & MAC Memory
- VSYS_IO_3V3

EEPROM Data Format *Expansion Board EEPROM* shows the format of the contents of the expansion board EEPROM. Data is stored in Big Endian with the least significant value on the right. All addresses read as a single

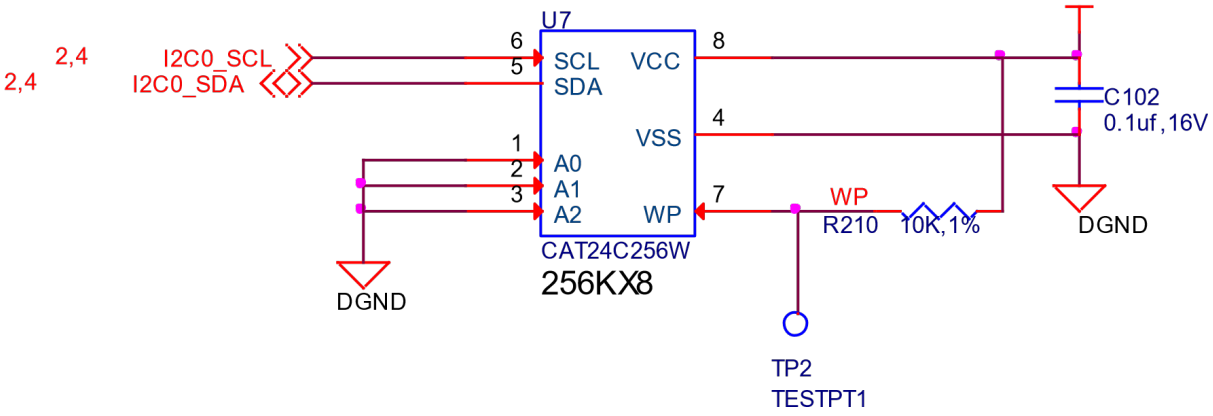


Fig. 4.5: Expansion board EEPROM with write protect

byte data from the EEPROM, but two byte addressing is used. ASCII values are intended to be easily read by the user when the EEPROM contents are dumped.

Todo: *Clean/Update table*

Table 4.1: Expansion Board EEPROM

Name	Offset	Size (bytes	Contents
Heade	0	4	0xAA, 0x55, 0x33, 0xEE
EEPROM Revision	4	2	Revision number of the overall format of this EEPROM in ASCII =A1
Board Name	6	32	Name of board in ASCII so user can read it when the EEPROM is dumped. Up to developer of the board as to what they call the board..
Version	38	4	Hardware version code for board in ASCII.Version format is up to the developer.i.e. 02.1…00A1….10A0
Manufacturer	42	16	ASCII name of the manufacturer. Company or individual's name.
Part Number	58	16	ASCII Characters for the part number. Up to maker of the board.
Number of Pins	74	2	Number of pins used by the daughter board including the power pins used. Decimal value of total pins 92 max, stored in HEX.
Serial Number	76	12	Serial number of the board. This is a 12 character string which is: **WWYY&&&&nnnn** where, WW = 2 digit week of the year of production, YY = 2 digit year of production , &&&&=Assembly code to let the manufacturer document the assembly number or product. A way to quickly tell from reading the serial number what the board is. Up to the developer to determine. nnnn = incrementing board number for that week of production
Pin Usage	88	148	Two bytes for each configurable pins of the 74 pins on the expansion connectors, MSB LSB Bit order: 15..14 ….. 1..0 Bit 15….Pin is used or not…0=Unused by cape 1=Used by cape Bit 14-13…Pin Direction…..1 0=Output 01=Input 11=BDIR Bits 12-7…Reserved……..should be all zeros Bit 6….Slew Rate …….0=Fast 1=Slow Bit 5….Rx Enable…….0=Disabled 1=Enabled Bit 4….Pull Up/Dn Select….0=Pulldown 1=PullUp Bit 3….Pull Up/DN enabled…0=Enabled 1=Disabled Bits 2-0 …Mux Mode Selection…Mode 0-7
VDD_3 Current	236	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
VDD_5 Current	238	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
SYS_5' Current	240	2	Maximum current in milliamps. This is HEX value of the current in decimal 1500mA=0x05 0xDC 325mA=0x01 0x45
DC Supplied	242	2	Indicates whether or not the board is supplying voltage on the VDD_5V rail and the current rating 000=No 1-0xFFFF is the current supplied storing the decimal quivalent in HEX format
Available	244	32543	Available space for other non-volatile codes/data to be used as needed by the manufacturer or SW driver. Could also store presets for use by SW.

Todo: Align with other boards and migrate away from pin usage entries for BeagleBone Black expansion

Pin Usage Consideration

This section covers things to watch for when hooking up to certain pins on the expansion headers.

Expansion Connectors

A combination of male and female headers is used for access to the expansion headers on the main board. There are three possible mounting configurations for the expansion headers:

- **Single** -no board stacking but can be used on the top of the stack.
- **Stacking-up** to four boards can be stacked on top of each other.

- **Stacking with signal stealing-up** to three boards can be stacked on top of each other, but certain boards will not pass on the signals they are using to prevent signal loading or use by other cards in the stack.

The following sections describe how the connectors are to be implemented and used for each of the different configurations.

Non-Stacking Headers-Single Cape For non-stacking capes single configurations or where the cape can be the last board on the stack, the two 46 pin expansion headers use the same connectors. *Single expansion connector* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.

Fig. 4.6: Single expansion connector

The connector is typically mounted on the bottom side of the board as shown in *Single cape expansion connector on BeagleBone Proto Cape with EEPROM from onlogic* . These are very common connectors and should be easily located. You can also use two single row 23 pin headers for each of the dual row headers.

It is allowed to only populate the pins you need. As this is a non-stacking configuration, there is no need for all headers to be populated. This can also reduce the overall cost of the cape. This decision is up to the cape designer.

For convenience listed in *Single Cape Connectors* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into BeagleBone AI-64 connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins or removing those pins that are not used by your particular design. The first item in**Table 18** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on BeagleBone AI-64

Table 4.2: Single Cape Connectors

SUPPLIER	PARTNUMBER	LENGTH(in)	OVERHANG(in)
Major League	TSHC-123-D-03-145-G-LF	.145	.004
Major League	TSHC-123-D-03-240-G-LF	.240	.099
Major League	TSHC-123-D-03-255-G-LF	.255	.114

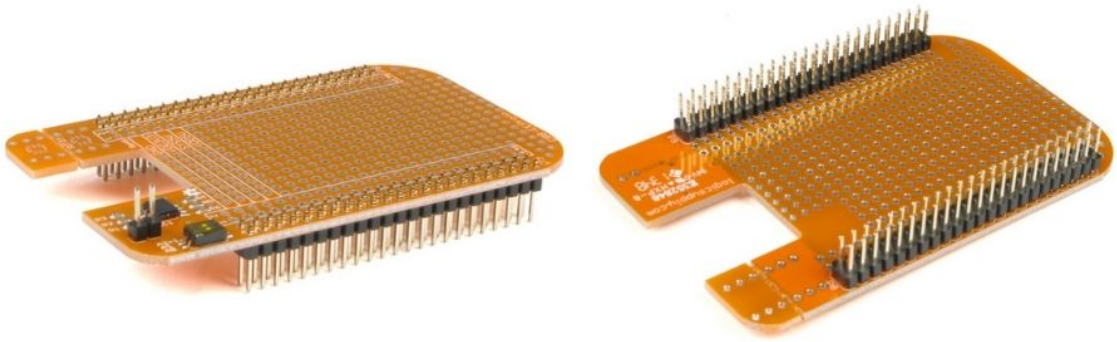


Fig. 4.7: Single cape expansion connector on BeagleBone Proto Cape with EEPROM from onlogic

The G in the part number is a plating option. Other options may be used as well as long as the contact area is gold. Other possible sources are Sullins and Samtec for these connectors. You will need to ensure the depth into the connector is sufficient

Main Expansion Headers-Stacking For stacking configuration, the two 46 pin expansion headers use the same connectors. *Expansion Connector* is a picture of the connector. These are dual row 23 position 2.54mm x 2.54mm connectors.

The connector is mounted on the top side of the board with longer tails to allow insertion into BeagleBone AI-64. *Stacked cape expansion connector* is the connector configuration for the connector.

For convenience listed in *Table 18* are some possible choices for part numbers on this connector. They have varying pin lengths and some may be more suitable than others for your use. It should be noted, that the longer the pin and the further it is inserted into BeagleBone AI-64 connector, the harder it will be to remove due to the tension on 92 pins. This can be minimized by using shorter pins. There are most likely other suppliers out there that will work for this connector as well. If anyone finds other suppliers of compatible connectors that work, let us know and they will be added to this document. The first item in **Table 19** is on the edge and may not be the best solution. Overhang is the amount of the pin that goes past the contact point of the connector on BeagleBone AI-64.

The third part listed in *Stacked Cape Connectors* will have insertion force issues.

Table 4.3: Stacked Cape Connectors

SUPPLIER	PARTNUMBER	TAIL LENGTH(in)	OVERHANG(in)
Major League	SSHQ-123-D-06-G-LF	.190	0.049
Major League	SSHQ-123-D-08-G-LF	.390	0.249
Major League	SSHQ-123-D-10-G-LF	.560	0.419

There are also different plating options on each of the connectors above. Gold plating on the contacts is the minimum requirement. If you choose to use a different part number for plating or availability purposes, make sure you do not select the “LT” option.

Other possible sources are Sullins and Samtec but make sure you select one that has the correct mating depth.

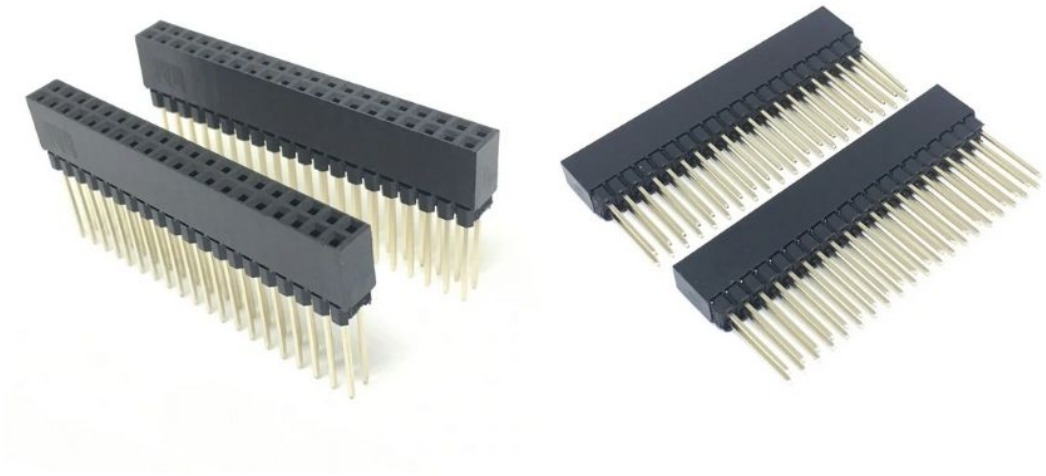


Fig. 4.8: Expansion Connector

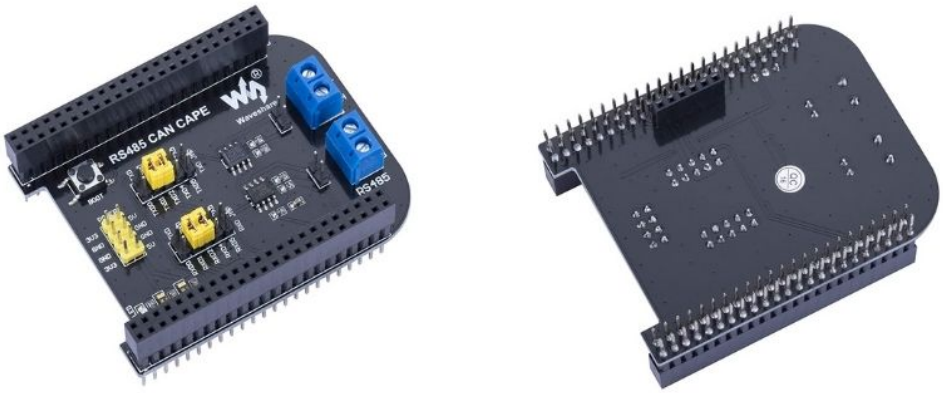


Fig. 4.9: Stacked cape expansion connector

Stacked Capes w/Signal Stealing *Stacked with signal stealing expansion connector figure* is the connector configuration for stackable capes that does not provide all of the signals upwards for use by other boards. This is useful if there is an expectation that other boards could interfere with the operation of your board by exposing those signals for expansion. This configuration consists of a combination of the stacking and nonstacking style connectors.

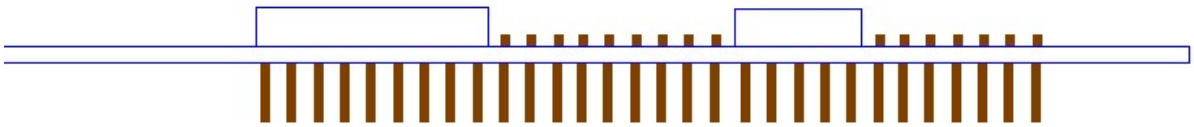


Fig. 4.10: Stacked with signal stealing expansion connector figure

Retention Force The length of the pins on the expansion header has a direct relationship to the amount of force that is used to remove a cape from BeagleBone AI-64. The longer the pins extend into the connector the harder it is to remove. There is no rule that says that if longer pins are used, that the connector pins have to extend all the way into the mating connector on BeagleBone AI-64, but this is controlled by the user and therefore is hard to control. We have also found that if you use gold pins, while more expensive, it makes for a smoother finish which reduces the friction.

This section will attempt to describe the tradeoffs and things to consider when selecting a connector and its pin length.

BeagleBone AI-64 Female Connectors *Connector Pin Insertion Depth* shows the key measurements used in calculating how much the pin extends past the contact point on the connector, what we call overhang.

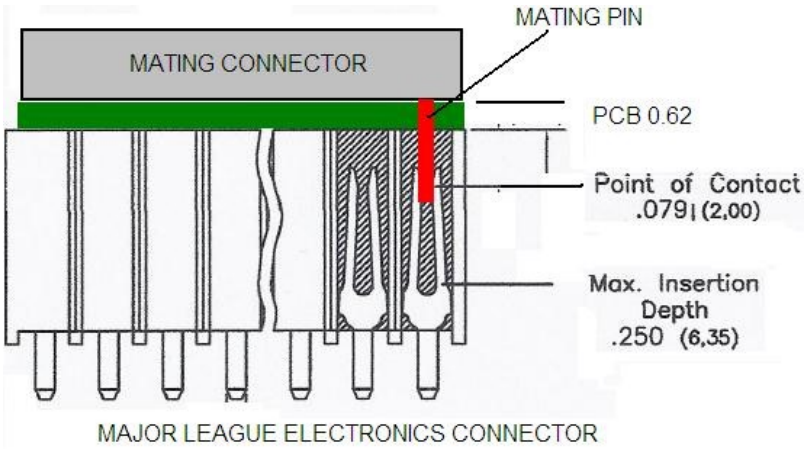


Fig. 4.11: Connector Pin Insertion Depth

To calculate the amount of the pin that extends past the Point of Contact, use the following formula:

Overhang=Total Pin Length- PCB thickness (.062) - contact point (.079)

The longer the pin extends past the contact point, the more force it will take to insert and remove the board. Removal is a greater issue than the insertion.

Signal Usage Based on the pin muxing capabilities of the processor, each expansion pin can be configured for different functions. When in the stacking mode, it will be up to the user to ensure that any conflicts are resolved between multiple stacked cards. When stacked, the first card detected will be used to set the pin muxing of each pin. This will prevent other modes from being supported on stacked cards and may result in them being inoperative.

In *Cape Header Connectors* section of this document, the functions of the pins are defined as well as the pin muxing options. Refer to this section for more information on what each pin is. To simplify things, if you use the default name as the function for each pin and use those functions, it will simplify board design and reduce conflicts with other boards.

Interoperability is up to the board suppliers and the user. This specification does not specify a fixed function on any pin and any pin can be used to the full extent of the functionality of that pin as enabled by the processor.

DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

Cape Power

This section describes the power rails for the capes and their usage.

Main Board Power The *Expansion Voltages* describes the voltages from the main board that are available on the expansion connectors and their ratings. All voltages are supplied by connector**P9**. The current ratings listed are per pin.

Table 4.4: Expansion Voltages

Current	Name	P9	P9	Name	Current
250mA	VDD_3V3B	3	4	VDD_3V3B	250mA
1000mA	VDD_5V	5	6	VDD_5V	1000mA
250mA	SYS_5V	7	8	SYS_5V	250mA

The *VSYS_IO_3V3* rail is supplied by the LDO on BeagleBone AI-64 and is the primary power rail for expansion boards. If the power requirement for the capes exceeds the current rating, then locally generated voltage rail can be used. It is recommended that this rail be used to power any buffers or level translators that may be used.

DC_VDD_5V is the main power supply from the DC input jack. This voltage is not present when the board is powered via USB. The amount of current supplied by this rail is dependent upon the amount of current available. Based on the board design, this rail is limited to 1A per pin from the main board.

The *VSYS_5V0* rail is the main rail for the regulators on the main board. When powered from a DC supply or USB, this rail will be 5V. The available current from this rail depends on the current available from the USB and DC external supplies.

Expansion Board External Power A cape can have a jack or terminals to bring in whatever voltages may be needed by that board. Care should be taken not to let this voltage be fed back into any of the expansion header pins.

It is possible to provide 5V to the main board from an expansion board. By supplying a 5V signal into the *DC_VDD_5V* rail, the main board can be supplied. This voltage must not exceed 5V. You should not supply any voltage into any other pin of the expansion connectors. Based on the board design, this rail is limited to 1A per pin to BeagleBone AI-64.

There are several precautions that need to be taken when working with the expansion headers to prevent damage to the board.

1. *Do not apply any voltages to any I/O pins when the board is not powered on.*
2. *Do not drive any external signals into the I/O pins until after the VSYS_IO_3V3 rail is up.*
3. *Do not apply any voltages that are generated from external sources.*
4. *If voltages are generated from the DC_VDD_5V signal, those supplies must not become active until after the VSYS_IO_3V3 rail is up.*

5. *If you are applying signals from other boards into the expansion headers, make sure you power the board up after you power up the BeagleBone AI-64 or make the connections after power is applied on both boards.*

Powering the processor via its I/O pins can cause damage to the processor.

Todo: Add BeagleBone AI-64 cape mechanical characteristics**

Standard Cape Size

Cape board dimensions shows the outline of the standard cape. The dimensions are in inches.

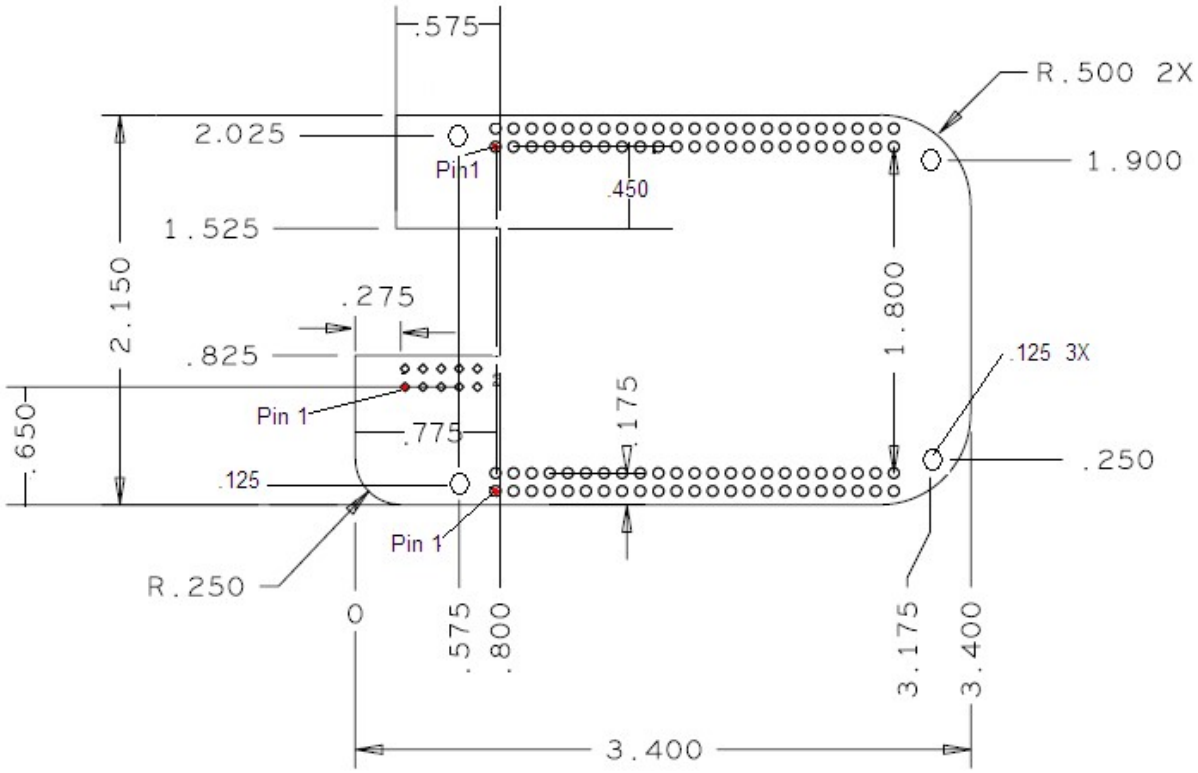


Fig. 4.12: Cape board dimensions

A notch is provided for BeagleBone Ethernet connector to stick up higher than the cape when mounted. This also acts as a key function to ensure that the cape is oriented correctly. Space is also provided to allow access to the user LEDs and reset button on BeagleBone board. On BeagleBone AI-64 board align it with the notch on the board silkscreen.

Extended Cape Size

Capes larger than the standard board size are also allowed. A good example would be the new BeagleBone AI-64 robotics cape. There is no practical limit to the sizes of these types of boards. The notch is also optional, but it is up to the supplier to ensure that the cape is not plugged incorrectly on BeagleBone AI-64 such that damage would be cause to BeagleBone AI-64. Any such damage will be the responsibility of the supplier of such a cape to repair. As with all capes, the EEPROM is required and compliance with the power requirements must be adhered to.

Todo: Update everything taken from BBB chapters to BB AI-64 compatible text.

4.3 RANDOM PRU STUFF THAT MIGHT NEED A HOME

Note: I don't want to blow this information away until I know no work went into it for TDA4VM. It is probably just AM3358 or AM5729 information. :-(

PRU0 and PRU1 Access below shows which PRU-ICSS signals can be accessed on the BeagleBone AI-64 and on which connector and pins they are accessible from. Some signals are accessible on the same pins.

Table 4.

	PIN	PROC	NAME
P8	11	R12	GPIO1_13
	12	T12	GPIO1_12
	15	U13	GPIO1_15
	16	V13	GPIO1_14
	20	V9	GPIO1_31
	21	U9	GPIO1_30
	27	U5	GPIO2_22
	28	V5	GPIO2_24
	29	R5	GPIO2_23
	39	T3	GPIO2_12
	40	T4	GPIO2_13
	41	T1	GPIO2_10
	42	T2	GPIO2_11
	43	R3	GPIO2_8
	44	R4	GPIO2_9
	45	R1	GPIO2_6
	46	R2	GPIO2_7
P9	17	A16	I2C1_SCL
	18	B16	I2C1_SDA
	19	D17	I2C2_SCL
	20	D18	I2C2_SDA
	21	B17	UART2_TXD
	22	A17	UART2_RXD
	24	D15	UART1_TXD
	25	A14	GPIO3_21footnote:[GPIO3_21 is also the 24.576MHZ clock input to the processor to enable HDMI audio.
	26	D16	UART1_RXD
	27	C13	GPIO3_19
	28	C12	SPI1_CS0
	29	B13	SPI1_D0
	30	D12	SPI1_D1
	31	A13	SPI1_SCLK

Chapter 5

Demos and Tutorials

5.1 Edge AI

Todo: Update to latest text from Texas Instruments Edge AI docs

5.1.1 Getting Started

Hardware setup

BeagleBone® AI-64 has TI's TDA4VM SoC which houses dual core A72, high performance vision accelerators, video codec accelerators, latest C71x and C66x DSP, high bandwidth realtime IPs for capture and display, GPU, dedicated safety island security accelerators. The SoC is power optimized to provide best in class performance for perception, sensor fusion, localization and path planning tasks in robotics, industrial and automotive applications.

For more details visit https://www.ti.com/product/TDA4VM

BeagleBone® AI-64 BeagleBone® AI-64 brings a complete system for developing artificial intelligence (AI) and machine learning solutions with the convenience and expandability of the BeagleBone® platform and the peripherals on board to get started right away learning and building applications. With locally hosted, ready-to-use, open-source focused tool chains and development environment, a simple web browser, power source and network connection are all that need to be added to start building performance-optimized embedded applications. Industry-leading expansion possibilities are enabled through familiar BeagleBone® cape headers, with hundreds of open-source hardware examples and dozens of readily available embedded expansion options available off-the-shelf.

To run the demos on BeagleBone® AI-64 you will require,

- BeagleBone® AI-64
- USB camera (Any V4L2 compliant 1MP/2MP camera, Eg. Logitech C270/C920/C922)
- Full HD eDP/HDMI display
- Minimum 16GB high performance SD card
- 100Base-T Ethernet cable connected to internet
- UART cable

- External Power Supply or Power Accessory Requirements
  a. Nominal Output Voltage: 5VDC
  b. Maximum Output Current: 5000 mA

Connect the components to the SK as shown in the image.

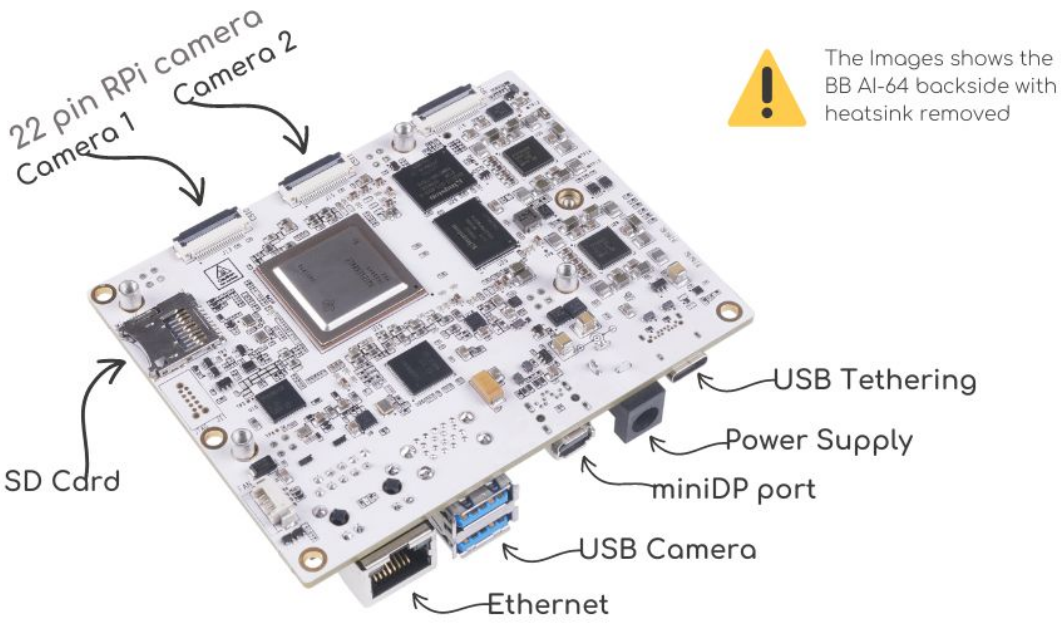


Fig. 5.1: BeagleBone® AI-64 for Edge AI connections

USB Camera UVC (USB video class) compliant USB cameras are supported on the BeagleBone® AI-64. The driver for the same is enabled in linux image. The linux image has been tested with C270/C920/C922 versions of Logitech USB cameras. Please refer to the TI Edge AI SDK FAQ to stream from multiple USB cameras simultaneously.

IMX219 Raw sensor **IMX219 camera module** from **Raspberry pi / Arducam** is supported by BeagleBone® AI-64. It is a 8MP sensor with no ISP, which can transmit raw SRGGB8 frames over CSI lanes at 1080p 60 fps. This camera module can be ordered from https://www.amazon.com/Raspberry-Pi-Camera-Module-Megapixel/dp/B01ER2SKFS The camera can be connected to any of the 2 RPi zero 22 pin camera headers on BB AI-64 as shown below

---

Todo: IMX219 CSI sensor connection with BeagleBone® AI-64 for Edge AI

Note that the headers have to be lifted up to connect the cameras

---

Note: To be updated By default IMX219 is disabled. After connecting the camera you can enable it by specifying the dtb overlay file in `/run/images/mmcblk0p1/uenv.txt` as below,

```
name_overlays=k3-j721e-edgeai-apps.dtbo k3-j721e-sk-rpi-cam-imx219.dtbo
```

Reboot the board after editing and saving the file.

Two RPi cameras can be connected to 2 headers for multi camera use-cases

Please refer *Camera sources (v4l2)* to know how to list all the cameras connected and select which one to use for the demo.

By default imx219 will be configured to capture at 8 bit, but it also supports 10 bit capture in 16 bit container. To use it in 10 bit mode, below steps are required:

- Modify the `/opt/edge_ai_apps/scripts/setup_cameras.sh` to set the format to 10 bit like below

```
CSI_CAM_0_FMT='[fmt:SRGGB8_1X10/1920x1080]'
CSI_CAM_1_FMT='[fmt:SRGGB8_1X10/1920x1080]'
```

- Change the imaging binaries to use 10 bit versions

```
mv /opt/imaging/imx219/dcc_2a.bin /opt/imaging/imx219/dcc_2a_8b.bin
mv /opt/imaging/imx219/dcc_viss.bin /opt/imaging/imx219/dcc_viss_8b.
↪bin
mv /opt/imaging/imx219/dcc_2a_10b.bin /opt/imaging/imx219/dcc_2a.bin
mv /opt/imaging/imx219/dcc_viss_10b.bin /opt/imaging/imx219/dcc_viss.
↪bin
```

- Set the input format in the `/opt/edge_ai_apps/configs/rpiV2_cam_example.yaml` as `rggb10`

Software setup

Preparing SD card image Download the `bullseye-xfce-edgeai-arm64` image from the links below and flash it to SD card using Balena etcher tool.

- To use via SD card: bbai64-debian-11.7-xfce-edgeai-arm64-2023-08-05-10gb.img.xz
- To flash on eMMC: bbai64-emmc-flasher-debian-11.7-xfce-edgeai-arm64-2023-08-05-10gb.img.xz

The Balena etcher tool can be installed either on Windows/Linux. Just download the etcher image and follow the instructions to prepare the SD card.

The etcher image is created for 16 GB SD cards, if you are using larger SD card, it is possible to expand the root filesystem to use the full SD card capacity using below steps

```
#find the SD card device entry using lsblk (Eg: /dev/sdc)
#use the following commands to expand the filesystem
#Make sure you have write permission to SD card or run the commands as root

#Unmount the BOOT and rootfs partition before using parted tool
umount /dev/sdX1
umount /dev/sdX2

#Use parted tool to resize the rootfs partition to use
#the entire remaining space on the SD card
#You might require sudo permissions to execute these steps
parted -s /dev/sdX resizepart 2 '100%'
e2fsck -f /dev/sdX2
resize2fs /dev/sdX2

#replace /dev/sdX in above commands with SD card device entry
```

Power ON and Boot Ensure that the power supply is disconnected before inserting the SD card. Once the SD card is firmly inserted in its slot and the board is powered ON, the board will take less than 20sec to boot and display a wallpaper as shown in the image below.

Todo: BeagleBone® AI-64 wallpaper upon boot

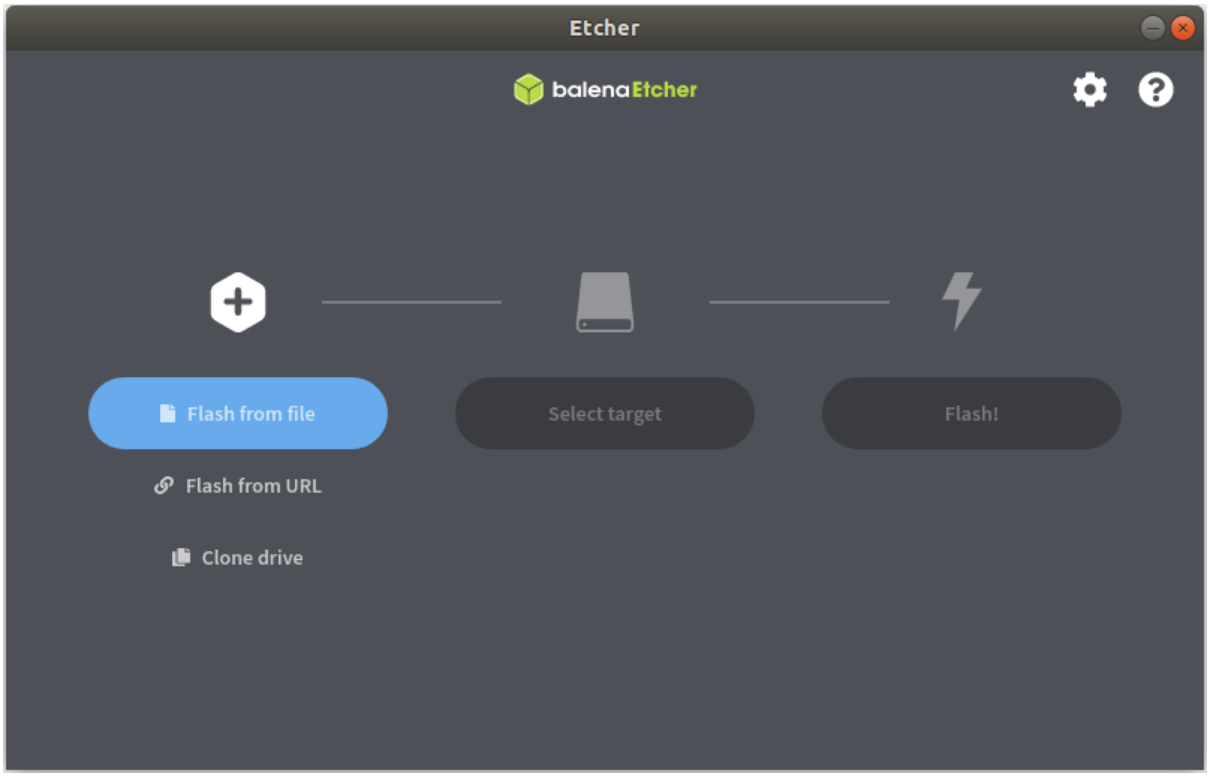


Fig. 5.2: Balena Etcher tool to flash SD card with Processor linux image Linux for Edge AI

You can also view the boot log by connecting the UART cable to your PC and use a serial port communications program.

For **Linux OS minicom** works well. Please refer to the below documentation on 'minicom' for more details.

https://help.ubuntu.com/community/Minicom

When starting minicom, turn on the colors options like below:

```
sudo minicom -D /dev/ttyUSB2 -c on
```

For **Windows OS Tera Term** works well. Please refer to the below documentation on 'TeraTerm' for more details

https://learn.sparkfun.com/tutorials/terminal-basics/tera-term-windows

---

Note: Baud rate should be configured to 115200 bps in serial port communication program. You may not see any log in the UART console if you connect to it after the booting is complete or login prompt may get lost in between boot logs, press ENTER to get login prompt

---

As part of the linux systemd `/opt/edge_ai_apps/init_script.sh` is executed which does the below,

- This kills weston compositor which holds the display pipe. This step will make the wallpaper showing on the display disappear and come back
- The display pipe can now be used by 'kmssink' GStreamer element while running the demo applications.
- The script can also be used to setup proxies if connected behind a firewall.

Once Linux boots login as `root` user with no password.

Connect remotely If you don't prefer the UART console, you can also access the device with the IP address that is shown on the display.

With the IP address one can ssh directly to the board, view the contents and run the demos.

For best experience we recommend using VSCode which can be downloaded from here.

https://code.visualstudio.com/download

You also require the "Remote development extension pack" installed in VSCode as mentioned here:

https://code.visualstudio.com/docs/remote/ssh

Todo: Microsoft Visual Studio Code for connecting to BeagleBone® AI-64 for Edge AI via SSH

5.1.2 Running Simple demos

This chapter describes how to run Python and C++ demo applications in edge_ai_apps with live camera and display.

Note: Please note that the Python demos are useful for quick prototyping while C++ demos are similar by design but tuned for performance.

Running Python based demo applications

Python based demos are simple executable scripts written for image classification, object detection and semantic segmentation. Demos are configured using a YAML file. Details on configuration file parameters can be found in *Demo Configuration file*

Sample configuration files for out of the box demos can be found in `edge_ai_apps/configs` this folder also contains a template config file which has brief info on each configurable parameter `edge_ai_apps/configs/app_config_template.yaml`

Here is how a Python based image classification demo can be run,

```
# go to edge-ai-apps folder
debian@beaglebone:~$ cd /opt/edge_ai_apps/apps_python

# enable root (password: temppwd)
debian@beaglebone:~$ sudo su
[sudo] password for beaglebone:

# use edge-ai-apps
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# sudo ./app_edgeai.py ../
↪configs/image_classification.yaml
```

The demo captures the input frames from connected USB camera and passes through pre-processing, inference and post-processing before sent to display. Sample output for image classification and object detection demos are as below,

To exit the demo press Ctrl+C.

Building and running C++ based demo applications

C++ apps needs to be built directly on target and requires header files of different deep-learning runtime framework and its dependencies which are installed in the setup script. The setup script builds the C++ apps when executed. However one can also follow below steps to clean build C++ apps

```
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# rm -rf build bin lib
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# mkdir build
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# cd build
debian@beaglebone:/opt/edge_ai_apps/apps_cpp/build# cmake ..
debian@beaglebone:/opt/edge_ai_apps/apps_cpp/build# make -j2
```

Run the demo once the application is successfully built

```
debian@beaglebone:/opt/edge_ai_apps/apps_cpp# ./bin/Release/app_edgeai ../
↪configs/image_classification.yaml
```

To exit the demo press Ctrl+C.

Note: Both Python and C++ applications are similar by construction and can accept the same config file and command line arguments

---

Note: The C++ apps built on Yocto Linux may not run in Docker as there could be a mismatch in Glib and other related tools. So its **highly recommended** to rebuild the C++ apps within the Docker environment.

5.1.3 DL models for Edge Inference

Model Downloader Tool

TI Model Zoo is a large collection of deep learning models validated to work on TI processors for edge AI. It hosts several pre-compiled model artifacts for TI hardware.

Use the **Model Downloader Tool** to download more models on target as shown,

```
debian@beaglebone:/opt/edge_ai_apps# ./download_models.sh
```

The script will launch an interactive menu showing the list of available, pre-imported models for download. The downloaded models will be placed under `/opt/model_zoo/` directory

The script can also be used in a non-interactive way as shown below:

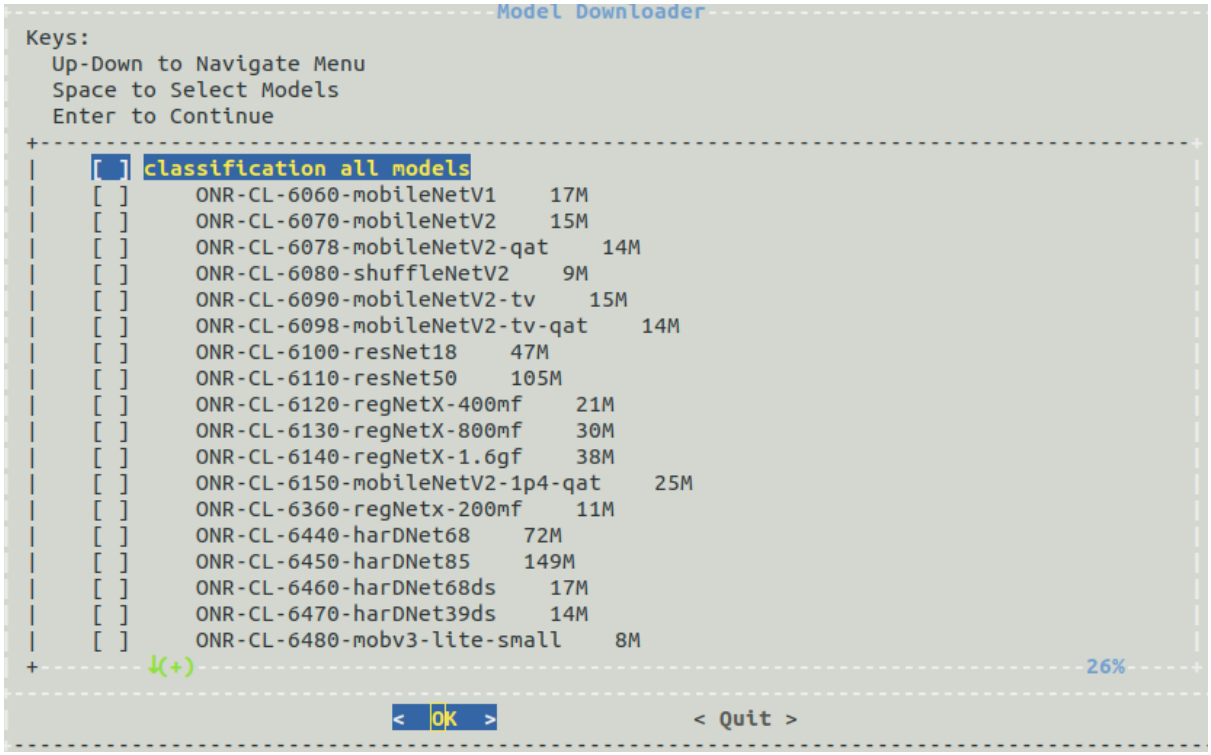


Fig. 5.3: Model downloader tool menu option to download models

```
debian@beaglebone:/opt/edge_ai_apps# ./download_models.sh --help
```

Import Custom Models

The BeagleBone® AI-64 Linux for Edge AI also supports importing pre-trained custom models to run inference on target.

The SDK makes use of pre-compiled DNN (Deep Neural Network) models and performs inference using various OSRT (open source runtime) such as TFLite runtime, ONNX runtime and Neo AI-DLR. In order to infer a DNN, SDK expects the DNN and associated artifacts in the below directory structure.

```
TFL-OD-2010-ssd-mobV2-coco-mlperf-300x300
|
├── param.yaml
|
├── artifacts
|   ├── 264_tidl_io_1.bin
|   ├── 264_tidl_net.bin
|   ├── 264_tidl_net.bin.layer_info.txt
|   ├── 264_tidl_net.bin_netLog.txt
|   ├── 264_tidl_net.bin.svg
|   ├── allowedNode.txt
|   └── runtimes_visualization.svg
|
└── model
    └── ssd_mobilenet_v2_300_float.tflite
```

DNN directory structure Each DNN must have the following 3 components:

1. **model**: This directory contains the DNN being targeted to infer

2. **artifacts**: This directory contains the artifacts generated after the compilation of DNN for SDK, and described in *DNN compilation for SDK – Basic Instructions*
3. **param.yaml**: A configuration file in yaml format to provide basic information about DNN, and associated pre and post processing parameters. More details can be find *Param file format*

Param file format Each DNN has its own pre-process, inference and post-process parameters to get the correct output. This information is typically available in the training software that was used to train the model. In order to convey this information to the SDK in a standardized fashion, we have defined a set of parameters that describe these operations. These parameters are in the param.yaml file.

Please see sample yaml files for various tasks such as image classification, semantic segmentation and object detection in edgeai-benchmark examples. Descriptions of various parameters are also in the yaml files. If users want to bring their own model to the SDK, then they need to prepare this information offline and get to the SDK. In next section we explain how to prepare this information

DNN compilation for SDK – Basic Instructions The BeagleBone® AI-64 Linux for Edge AI supports three different runtimes to infer a DNN, and user can choose a run time depending on the format of DNN. We recommend users to use different run times and compare the performance and select the one which provides best performance. User can find the steps to generate the artifacts directory at Edge AI TIDL Tools

DNN compilation for SDK – Advanced Instructions For beginners who are trying to compile models for the SDK, we recommend the basic instructions given in the previous section. However, DNNs have lot of variety and some models may need a different kind of preprocessing or postprocessing operations. In order to help customers deal with different kinds of models, we have prepared a model zoo in the repository edgeai-modelzoo

For the DNNs which are part of TI's model zoo, one can find the compilation settings and pre-compiled model artifacts in edgeai-benchmark repository. Instructions are also given to compile custom models. When using edgeai-benchmark for model compilation, the yaml file is automatically generated and artifacts are packaged in the way SDK understands. Please follow the instructions in the repository to get started.

5.1.4 Demo Configuration file

The demo config file uses YAML format to define input sources, models, outputs and finally the flows which defines how everything is connected. Config files for out-of-box demos are kept in `edge_ai_apps/configs` folder. The folder contains config files for all the use cases and also multi-input and multi-inference case. The folder also has a template YAML file `app_config_template.yaml` which has detailed explanation of all the parameters supported in the config file.

Config file is divided in 4 sections:

1. Inputs
2. Models
3. Outputs
4. Flows

Inputs

The input section defines a list of supported inputs like camera, filesrc etc. Their properties like shown below.

```
inputs:
    input0:                                          #Camera Input
        source: /dev/video2                          #Device file entry of the␣
↪camera
        format: jpeg                                 #Input data format␣
```

(continues on next page)

(continued from previous page)

```
↪supported by camera
        width: 1280                                     #Width and Height of the␣
↪input
        height: 720
        framerate: 30                                   #Framerate of the source

    input1:                                             #Video Input
        source: ../data/videos/video_0000_h264.mp4      #Video file
        format: h264                                    #File encoding format
        width: 1280
        height: 720
        framerate: 25

    input2:                                             #Image Input
        source: ../data/images/%04d.jpg                 #Sequence of Image files,␣
↪printf style formatting is used
        width: 1280
        height: 720
        index: 0                                        #Starting Index␣
↪(optional)
        framerate: 1
```

All supported inputs are listed in template config file. Below are the details of most commonly used inputs.

Camera sources (v4l2) **v4l2src** GStreamer element is used to capture frames from camera sources which are exposed as v4l2 devices. In Linux, there are many devices which are implemented as v4l2 devices. Not all of them will be camera devices. You need to make sure the correct device is configured for running the demo successfully.

`init_script.sh` is ran as part of systemd, which detects all cameras connected and prints the detail like below in the UART console:

```
debian@beaglebone:/opt/edge_ai_apps# ./init_script.sh
USB Camera detected
    device = /dev/video18
    format = jpeg
CSI Camera 0 detected
    device = /dev/video2
    name = imx219 8-0010
    format = [fmt:SRGGB8_1X8/1920x1080]
    subdev_id = 2
    isp_required = yes
IMX390 Camera 0 detected
    device = /dev/video18
    name = imx390 10-001a
    format = [fmt:SRGGB12_1X12/1936x1100 field: none]
    subdev_id = /dev/v4l-subdev7
    isp_required = yes
    ldc_required = yes
```

script can also be run manually later to get the camera details.

From the above log we can determine that 1 USB camera is connected (/dev/video18), and 1 CSI camera is connected (/dev/video2) which is imx219 raw sensor and needs ISP. IMX390 camera needs both ISP and LDC.

Using this method, you can configure correct device for camera capture in the input section of config file.

```
input0:
    source: /dev/video18  #USB Camera
    format: jpeg          #if connected USB camera supports jpeg
    width: 1280
```

(continues on next page)

(continued from previous page)

```
    height: 720
    framerate: 30

input1:
    source: /dev/video2  #CSI Camera
    format: auto         #let the gstreamer negotiate the format
    width: 1280
    height: 720
    framerate: 30

input2:
    source: /dev/video2  #IMX219 raw sensor that needs ISP
    format: rggb         #ISP will be added in the pipeline
    width: 1920
    height: 1080
    framerate: 30
    subdev-id: 2         #needed by ISP to control sensor params via ioctls

input3:
    source: /dev/video2  #IMX390 raw sensor that needs ISP
    width: 1936
    height: 1100
    format: rggb12       #ISP will be added in the pipeline
    subdev-id: 2         #needed by ISP to control sensor params via ioctls
    framerate: 30
    sen-id: imx390
    ldc: True            #LDC will be added in the pipeline
```

Make sure to configure correct `format` for camera input. `jpeg` for USB camera that supports MJPEG (Ex. C270 logitech USB camera). `auto` for CSI camera to allow gstreamer to negotiate the format. `rggb` for sensor that needs ISP.

Video sources H.264 and H.265 encoded videos can be provided as input sources to the demos. Sample video files are provided under `/opt/edge_ai_apps/data/videos/video_0000_h264.mp4` and `/opt/edge_ai_apps/data/videos/video_000_h265.mp4`

```
input1:
    source: ../data/videos/video_0000_h264.mp4
    format: h264
    width: 1280
    height: 720
    framerate: 25

input2:
    source: ../data/videos/video_0000_h265.mp4
    format: h265
    width: 1280
    height: 720
    framerate: 25
```

Make sure to configure correct `format` for video input as shown above. By default the format is set to `auto` which will then use the GStreamer bin `decodebin` instead.

Image sources JPEG compressed images can be provided as inputs to the demos. A sample set of images are provided under `/opt/edge_ai_apps/data/images`. The names of the files are numbered sequentially and incrementally and the demo plays the files at the fps specified by the user.

```
input2:
    source: ../data/images/%04d.jpg
```

(continues on next page)

(continued from previous page)

```
    width: 1280
    height: 720
    index: 0
    framerate: 1
```

RTSP sources H.264 encoded video streams either coming from a RTSP compliant IP camera or via RTSP server running on a remote PC can be provided as inputs to the demo.

```
input0:
    source: rtsp://172.24.145.220:8554/test # rtsp stream url, replace this␣
↪with correct url
    width: 1280
    height: 720
    framerate: 30
```

Note: Usually video streams from any IP camera will be encrypted and cannot be played back directly without a decryption key. We tested RTSP source by setting up an RTSP server on a Ubuntu 18.04 PC by referring to this writeup, Setting up RTSP server on PC

Models

The model section defines a list of models that are used in the demo. Path to the model directory is a required argument for each model and rest are optional properties specific to given use cases like shown below.

```
models:
    model0:
        model_path: ../models/segmentation/ONR-SS-871-deeplabv3lite-mobv2-
↪cocoseg21-512x512   #Model Directory
        alpha: 0.4                                                          ␣
↪                 #alpha for blending segmentation mask (optional)
    model1:
        model_path: ../models/detection/TFL-OD-202-ssdLite-mobDet-DSP-coco-
↪320x320
        viz_threshold: 0.3                                                  ␣
↪                #Visualization threshold for adding bounding boxes␣
↪(optional)
    model2:
        model_path: ../models/classification/TVM-CL-338-mobileNetV2-qat
        topN: 5                                                             ␣
↪                 #Number of top N classes (optional)
```

Below are some of the use case specific properties:

1. **alpha**: This determines the weight of the mask for blending the semantic segmentation output with the input image `alpha * mask + (1 - alpha) * image`
2. **viz_threshold**: Score threshold to draw the bounding boxes for detected objects in object detection. This can be used to control the number of boxes in the output, increase if there are too many and decrease if there are very few
3. **topN**: Number of most probable classes to overlay on image classification output

The content of the model directory and its structure is discussed in detail in *Import Custom Models*

Outputs

The output section defines a list of supported outputs.

```
outputs:
    output0:                                                    #Display␣
↪Output
        sink: kmssink
        width: 1920                                             #Width and␣
↪Height of the output
        height: 1080
        connector: 39                                           #Connector␣
↪ID for kmssink (optional)

    output1:                                                    #Video Output
        sink: ../data/output/videos/output_video.mkv            #Output␣
↪video file
        width: 1920
        height: 1080

    output2:                                                    #Image Output
        sink: ../data/output/images/output_image_%04d.jpg       #Image file␣
↪name, printf style formatting is used
        width: 1920
        height: 1080
```

All supported outputs are listed in template config file. Below are the details of most commonly used outputs

Display Sink (kmssink) When you have only one display connected to the SK, kmssink will try to use it for displaying the output buffers. In case you have connected multiple display monitors (e.g. Display Port and HDMI), you can select a specific display for kmssink by passing a specific connector ID number. Following command finds out the connected displays available to use.

Note: Run this command outside docker container. The first number in each line is the connector-id which we will use in next step.

```
debian@beaglebone:/opt/edge_ai_apps# modetest -M tidss -c | grep connected
39      38      connected       DP-1            530x300         12      38
48      0       disconnected    HDMI-A-1        0x0             0       47
```

From above output, we can see that connector ID 39 is connected. Configure the connector ID in the output section of the config file.

Video sinks The post-processed outputs can be encoded in H.264 format and stored on disk. Please specify the location of the video file in the configuration file.

```
output1:
    sink: ../data/output/videos/output_video.mkv
    width: 1920
    height: 1080
```

Image sinks The post-processed outputs can be stored as JPEG compressed images. Please specify the location of the image files in the configuration file. The images will be named sequentially and incrementally as shown.

```
output2:
    sink: ../data/output/images/output_image_%04d.jpg
    width: 1920
    height: 1080
```

Flows

The flows section defines how inputs, models and outputs are connected. Multiple flows can be defined to achieve multi input, multi inference like below.

```
flows:
    flow0:                                  #First Flow
        input: input0                       #Input for the Flow
        models: [model1, model2]            #List of models to be used
        outputs: [output0, output0]         #Outputs to be used for each model␣
↪inference output
        mosaic:                             #Positions to place the inference␣
↪outputs in the output frame
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  90
    flow1:                                  #Second Flow
        input: input1
        models: [model0, model3]
        outputs: [output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  540
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  540
```

Each flow should have exactly **1 input**, **n models** to infer the given input and **n outputs** to render the output of each inference. Along with input, models and outputs it is required to define **n mosaics** which are the position of the inference output in the final output plane. This is needed because multiple inference outputs can be rendered to same output (Ex: Display).

Command line arguments Limited set of command line arguments can be provided, run with '-h' or '–help' option to list the supported parameters.

```
usage: Run : ./app_edgeai.py -h for help

positional arguments:
  config           Path to demo config file
                       ex: ./app_edgeai.py ../configs/app_config.yaml

optional arguments:
  -h, --help       show this help message and exit
  -n, --no-curses  Disable curses report
                   default: Disabled
  -v, --verbose    Verbose option to print profile info on stdout
                   default: Disabled
```

5.1.5 Running Advance demos

The same Python and C++ demo applications can be used to run multiple inference models and also work with multiple inputs with just simple changes in the config file.

From a repo of input sources, output sources and models one can define advance dataflows which connect them in various configurations. Details on configuration file parameters can be found in *Demo Configuration file*

Single input multi inference demo

Here is an example of a single-input, multi-inference demo which takes a camera input and run multiple networks on each of them.

```
debian@beaglebone:/opt/edge_ai_apps/apps_python# ./app_edgeai.py ../configs/
↪single_input_multi_infer.yaml
```

Sample output for single input, multi inference demo is as shown below,

Fig. 5.4: Sample output showing single input, mutli-inference output

We can specify the output window location and sizes as shown in the configuration file,

```
flows:
    flow0:
        input: input0
        models: [model0, model1, model2, model3]
        outputs: [output0, output0, output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
```

(continues on next page)

(continued from previous page)

```
            height: 450
            pos_x:  960
            pos_y:  90
        mosaic2:
            width:  800
            height: 450
            pos_x:  160
            pos_y:  540
        mosaic3:
            width:  800
            height: 450
            pos_x:  960
            pos_y:  540
```

Multi input multi inference demo

Here is an example of a multi-input, multi-inference demo which takes a camera input and video input and runs multiple networks on each of them.

```
debian@beaglebone:/opt/edge_ai_apps/apps_python# ./app_edgeai.py ../configs/
↪multi_input_multi_infer.yaml
```

Sample output for multi input, multi inference demo is as shown below,

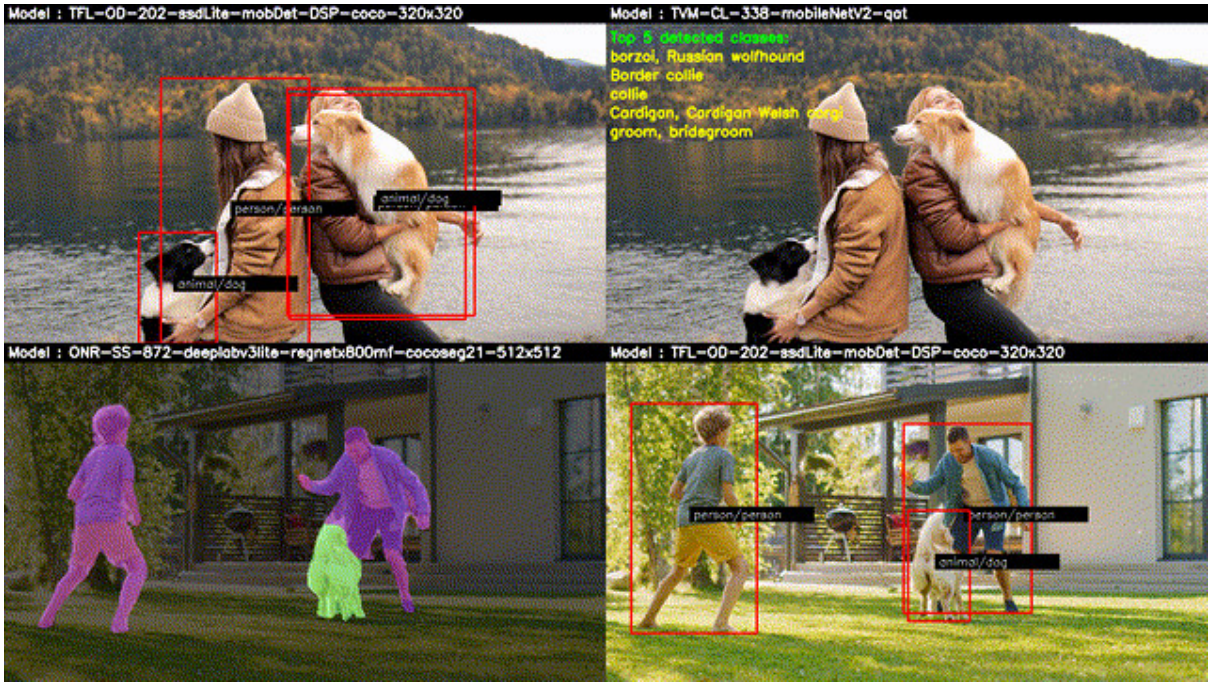


Fig. 5.5: Sample output showing multi-input, mutli-inference output

We can specify the output window location and sizes as shown in the configuration file,

```
flows:
    flow0:
        input: input0
        models: [model1, model2]
        outputs: [output0, output0]
        mosaic:
```

(continues on next page)

(continued from previous page)

```
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  90
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  90
    flow1:
        input: input1
        models: [model0, model3]
        outputs: [output0, output0]
        mosaic:
            mosaic0:
                width:  800
                height: 450
                pos_x:  160
                pos_y:  540
            mosaic1:
                width:  800
                height: 450
                pos_x:  960
                pos_y:  540
```

5.1.6 Docker Environment

Docker is a set of "platform as a service" products that uses the OS-level virtualization to deliver software in packages called containers. Docker container provides a quick start environment to the developer to run the out of box demos and build applications.

The Docker image is based on Ubuntu 20.04.LTS and contains different open source components like OpenCV, GStreamer, Python and pip packages which are required to run the demos. The user can choose to install any additional 3rd party applications and packages as required.

Building Docker image

The *docker/Dockerfile* in the edge_ai_apps repo describes the recipe for creating the Docker container image. Feel free to review and update it to include additional packages before building the image.

Note: Building Docker image on target using the provided Dockerfile will take about 15-20 minutes to complete with good internet connection. Building Docker containers on target can be slow and resource constrained. The Dockerfile provided will build on target without any issues but if you add more packages or build components from source, running out of memory can be a common problem. As an alternative we highly recommend trying QEMU builds for cross-compiling the images for arm64 architecture on a PC and then load the compiled image on target.

Initiate the Docker image build as shown,

```
debian@beaglebone:/opt/edge_ai_apps/docker#./docker_build.sh
```

Running the Docker container

Enter the Docker session as shown,

```
debian@beaglebone:/opt/edge_ai_apps/docker#./docker_run.sh
```

This will start a Ubuntu 20.04.LTS image based Docker container and the prompt will change as below,

```
[docker] debian@beaglebone:/opt/edge_ai_apps#
```

The Docker container has been created in privilege mode, so that it has root capabilities to all devices on the target system like Network etc. The container file system also mounts the target file system of /dev, /opt to access camera, display and other hardware accelerators the SoC has to offer.

Note: It is highly recommended to use the docker_run.sh script to launch the Docker container because this script will take care of saving any changes made to the filesystem. This will make sure that any modifications to the Docker filesystem including new package installation, updates to some files and also command history is saved automatically and is available the next time you launch the container. The container will be committed only if you exit from the container explicitly. If you restart the board without exiting container, any changes done from last saved state will be lost.

Note: After building and running the docker container, one needs to run `setup_script.sh` before running any of the demo applications. Please refer to *Software setup* for more details.

Handling proxy settings

If the board running the Docker container is behind a proxy server, the default settings for downloading files and installing packages via apt-get will not work. If you are running the board from TI network, docker build and run scripts will automatically detect and configure necessary proxy settings

For other cases, you need to modify the script `/usr/bin/setup_proxy.sh` to add the custom proxy settings required for your network.

Additional Docker commands

Note: This section is provided only for additional reference and not required to run out-of-box demos

---

Commit Docker container

Generally, containers have a short life cycle. If the container has any local changes it is good to save the changes on top of the existing Docker image. When re-running the Docker image, the local changes can be restored.

Following commands show how to save the changes made to the last container. Note that this is already done automatically by `docker_run.sh` when you exit the container.

```
cont_id=`docker ps -q -l`
docker commit $cont_id edge_ai_kit
docker container rm $cont_id
```

For more information refer: Commit Docker image

Save Docker Image

Docker image can be saved as tar file by using the command below:

```
docker save --output <pre_built_docker_image.tar>
```

For more information refer here. Save Docker image

Load Docker image

Load a previously saved Docker image using the command below:

```
docker load --input <pre_built_docker_image.tar>
```

For more information refer here. Load Docker image

Remove Docker image

Docker image can be removed by using the command below:

```
Remove selected image:
docker rmi <image_name/ID>

Remove all image:
docker image prune -a
```

For more information refer rmi reference and Image prune reference

Remove Docker container

Docker container can be removed by using the command below:

```
Remove selected container:
docker rm <container_ID>

Remove all container:
docker container prune
```

For more information refer here. rm reference and Container Prune reference

Relocating Docker Root Location

The default location for Docker files is **/var/lib/docker**. Any Docker images created will be stored here. This will be a problem anytime the SD card is updated with a new targetfs. If a secondary storage (SSD or USB based storage) is available, then it is recommended to relocate the default Docker root location so as to preserve any existing Docker images. Once the relocation has been done, the Docker content will not be affected by any future targetfs updates or accidental corruptions of the SD card.

The following steps outline the process for Docker root directory relocation assuming that the current Docker root is not at the desired location. If the current location is the desired location then exit this procedure.

1. Run 'Docker info' command inspect the output. Locate the line with content **Docker Root Dir**. It will list the current location.
2. To preserve any existing images, export them to .tar files for importing later into the new location.
3. Inspect the content under /etc/docker to see if there is a file by name **daemon.json**. If the file is not present then create **/etc/docker/docker.json** and add the following content. Update the 'key:value' pair for the key "graph" to reflect the desired root location. If the file already exists, then make sure that the line with "graph" exists in the file and points to the desired target location.

```
{
  "graph": "/run/images/nvme0n1/docker_root",
  "storage-driver": "overlay",
  "live-restore": true
}
```

In the configuration above, the key/value pair **'"graph": "/run/images/nvme0n1/docker_root"'** defines the root location **'/run/images/nvme0n1/docker_root'.**

4. Once the daemon.json file has been copied and updated, run the following commands

```
$ systemctl restart docker
$ docker info
```

Make sure that the new Docker root appears under **Docker Root Dir** value.

5. If you exported the existing images in step (2) then import them and they will appear under the new Docker root.
6. Anytime the SD card is updated with a new targetfs, steps (1), (3), and (4) need to be followed.

Additional references

https://docs.docker.com/engine/reference/commandline/images/
https://docs.docker.com/engine/reference/commandline/ps/

5.1.7 Data Flows

The **app_edgeai** application at a high level can be split into 3 parts,

- Input pipeline - Grabs a frame from camera, video, image or RTSP source
- Output pipeline - Sends the output to display or a file
- Compute pipeline - Performs pre-processing, inference and post-processing

Here are the data flows for each reference demo and the corresponding GStreamer launch strings that **app_edgeai** application generates. User can interact with the application via the *Demo Configuration file*

Image classification

In this demo, a frame is grabbed from an input source and split into two paths. The "analytics" path resizes the input maintaining the aspect ratio and crops the input to match the resolution required to run the deep learning network. The "visualization" path is provided to the post-processing module which overlays the detected classes. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
```

(continues on next page)

(continued from previous page)

```
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

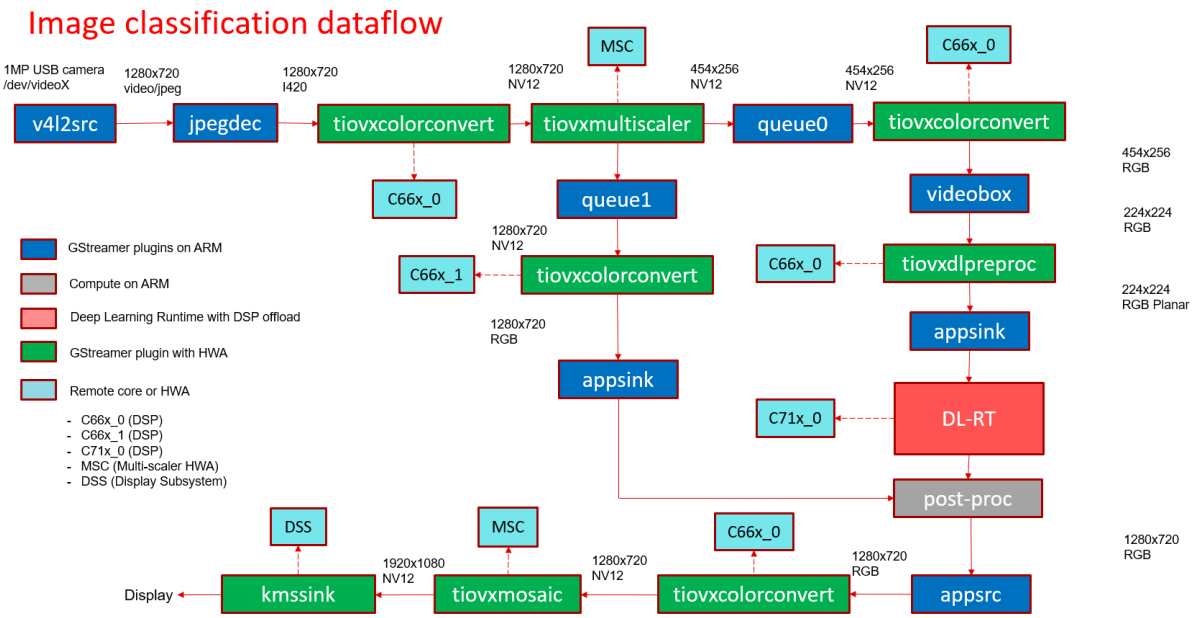


Fig. 5.6: GStreamer based data-flow pipeline for image classification demo with USB camera and display

Object Detection

In this demo, a frame is grabbed from an input source and split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which overlays rectangles around detected objects. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
```

(continues on next page)

(continued from previous page)

```
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

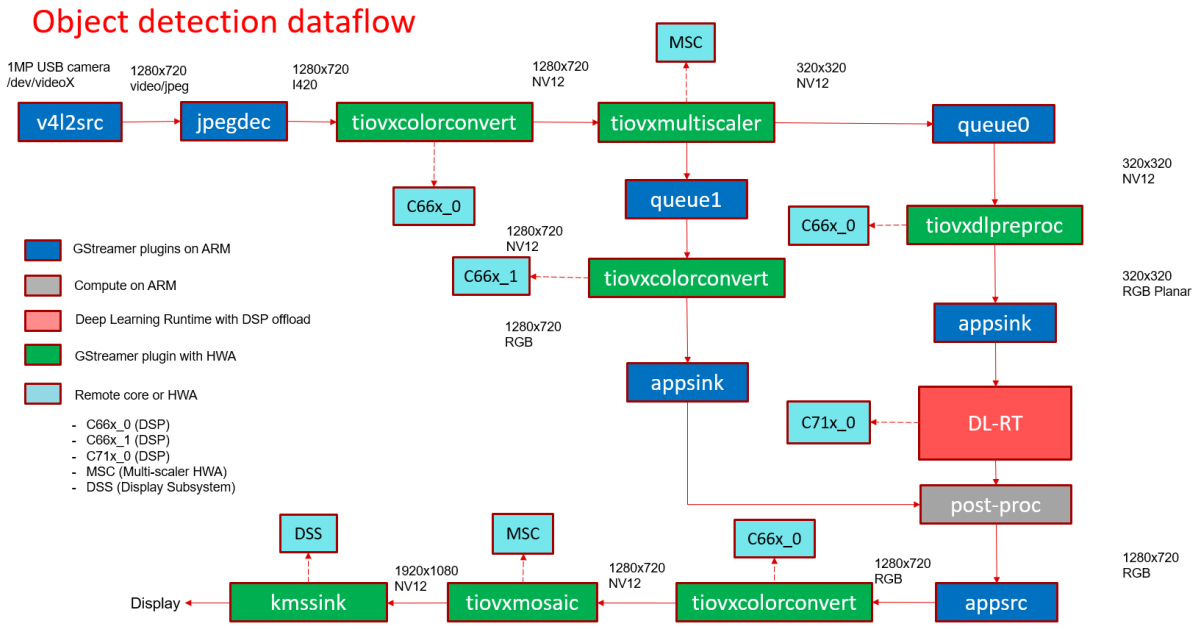


Fig. 5.7: GStreamer based data-flow pipeline for object detection demo with USB camera and display

Semantic Segmentation

In this demo, a frame is grabbed from an input source and split into two paths. The “analytics” path resize the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which blends each segmented pixel to a color map. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
```

(continues on next page)

(continued from previous page)

```
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

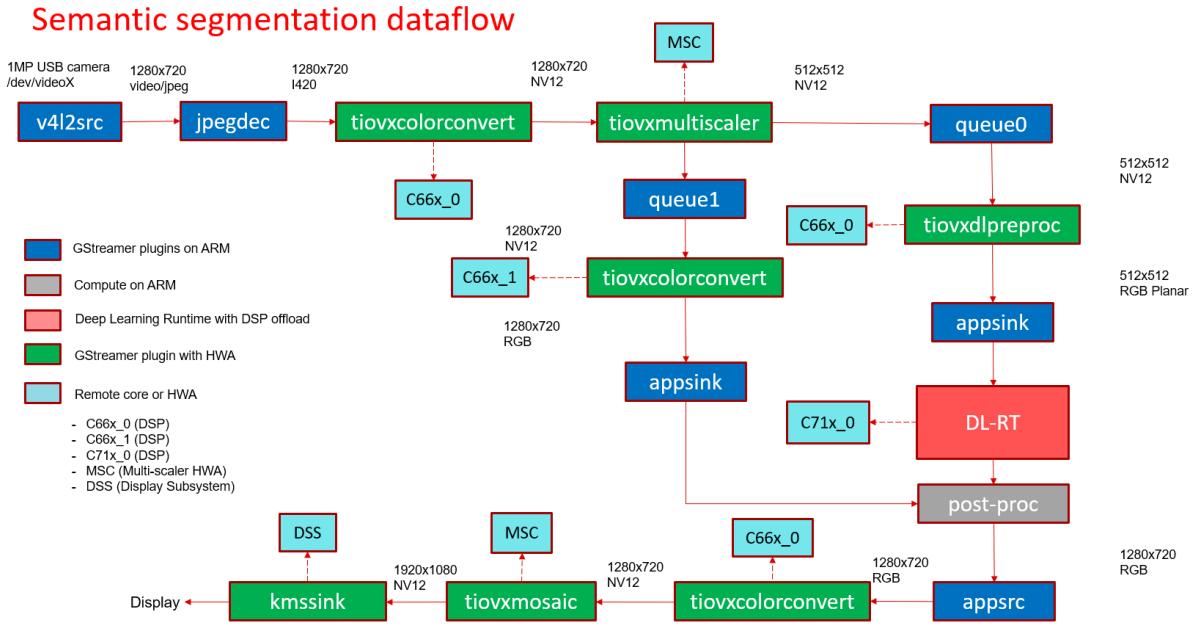


Fig. 5.8: GStreamer based data-flow pipeline for semantic segmentation demo with USB camera and display

Human Pose Estimation

In this demo, a frame is grabbed from an input source and split into two paths. The “analytics” path resize the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which overlays the keypoints and lines to draw the pose. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video2 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=640, height=640 ! tiovxdlpreproc data-
↪type=10 target=0 channel-order=0 mean-0=0.000000 mean-1=0.000000 mean-2=0.
↪000000 scale-0=1.000000 scale-1=1.000000 scale-2=1.000000 tensor-
↪format=bgr out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
GStreamer output pipeline:
```

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
tiovxmosaic name=mosaic_0 background=/tmp/background_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

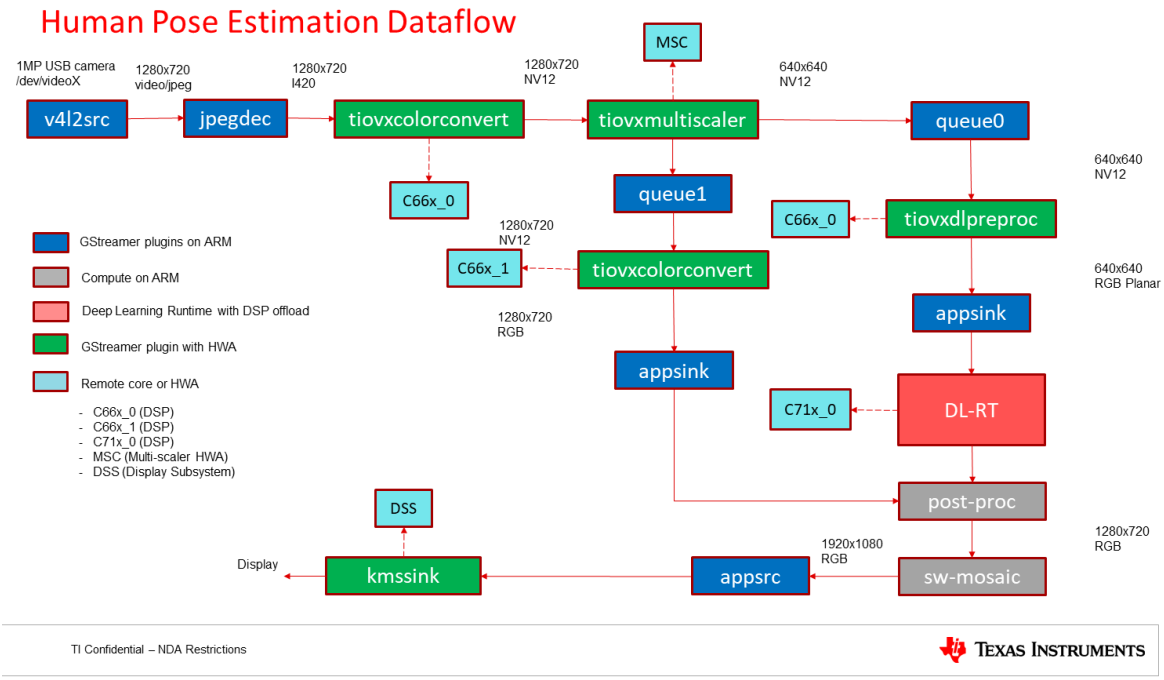


Fig. 5.9: GStreamer based data-flow pipeline for Human Pose Estimation demo with USB camera and display

Video source

In this demo, a video file is read from a known location and passed to a de-muxer to extract audio and video streams, the video stream is parsed and raw encoded information is passed to a HW video decoder. Note that H.264 and H.265 encoded videos are supported, making use of the respective HW decoders. The resulting output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
filesrc location=/opt/edge_ai_apps/data/videos/video_0000_h264.mp4 ! qtdemux␣
↪! h264parse ! v4l2h264dec ! video/x-raw, format=NV12  ! tiovxmultiscaler␣
↪name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

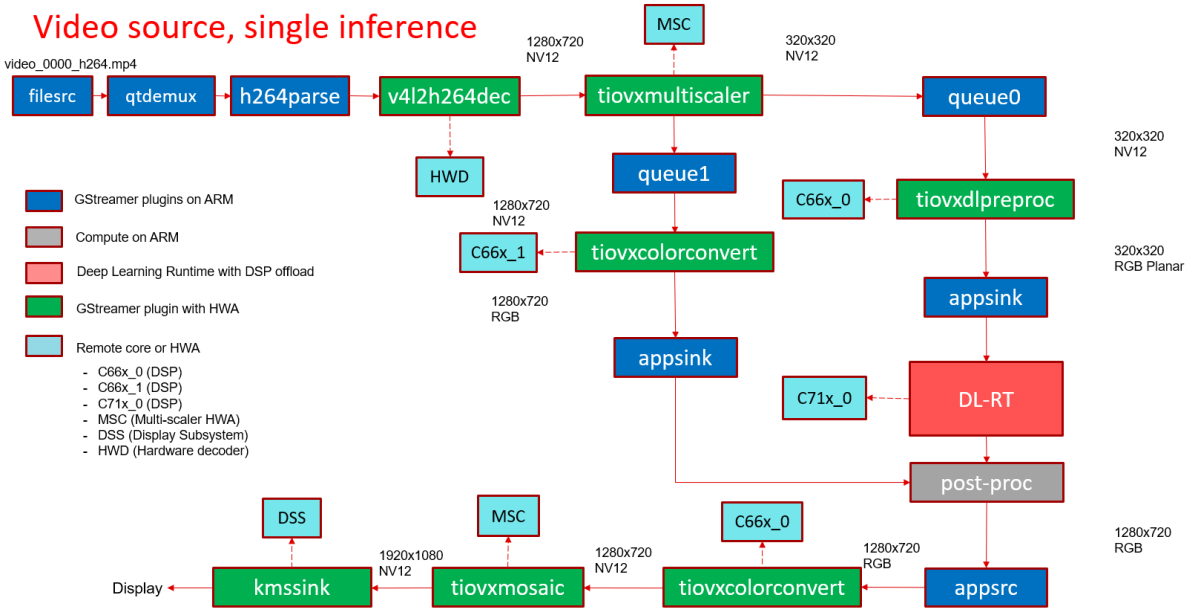


Fig. 5.10: GStreamer based data-flow pipeline with video file input source and display

RTSP source

In this demo, a video file is read from a RTSP source and passed to a de-muxer to extract audio and video streams, the video stream is parsed and raw encoded information is passed to a video decoder and the resulting output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
rtspsrc location=rtsp://172.24.145.220:8554/test latency=0 buffer-mode=auto !
↪ rtph264depay ! h264parse ! v4l2h264dec ! video/x-raw, format=NV12 !
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

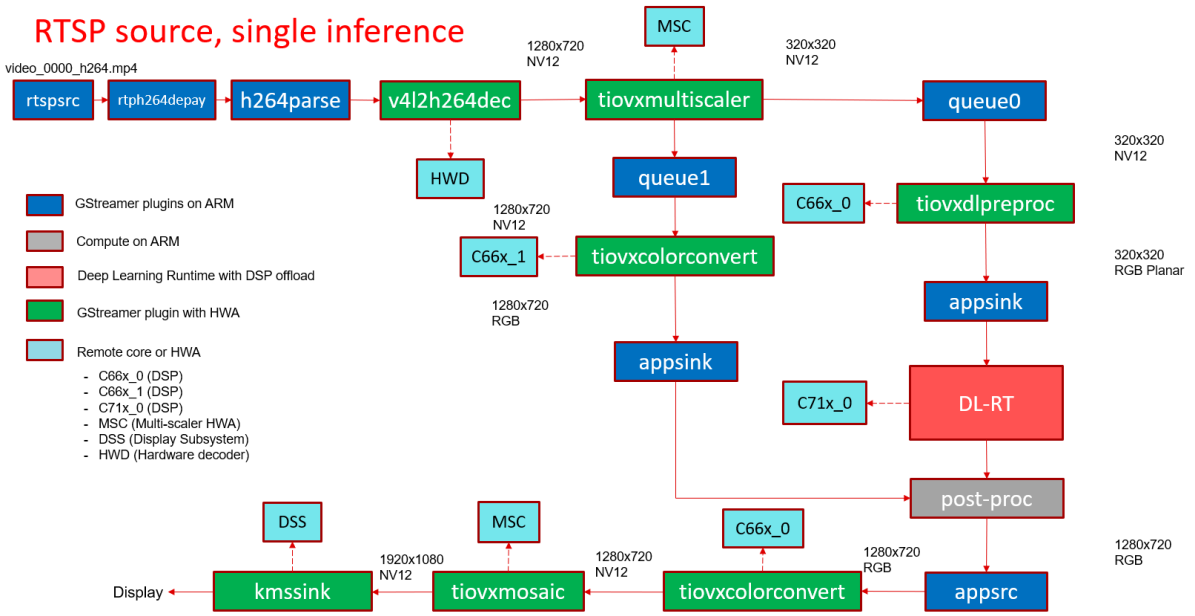


Fig. 5.11: GStreamer based data-flow pipeline with RTSP based video file source and display

RPiV2 Camera Sensor (IMX219)

In this demo, raw frames in SRGGB8 format are captured form RPiV2 (imx219) camera sensor. VISS (Vision Imaging Subsystem) is used to process the raw frames and get the output in NV12, VISS also cotrols the sensor parameters like exposure, gain etc.. via v4l2 ioctls. The NV12 output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video2 io-mode=5 ! video/x-bayer, width=1920,␣
↪height=1080, format=rggb ! tiovxisp device=/dev/v4l-subdev2 dcc-isp-file=/
↪opt/imaging/imx219/dcc_viss.bin dcc-2a-file=/opt/imaging/imx219/dcc_2a.bin␣
↪format-msb=7 ! video/x-raw, format=NV12 ! tiovxmultiscaler ! video/x-raw,␣
↪width=1280, height=720 ! tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
```

(continues on next page)

(continued from previous page)

```
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

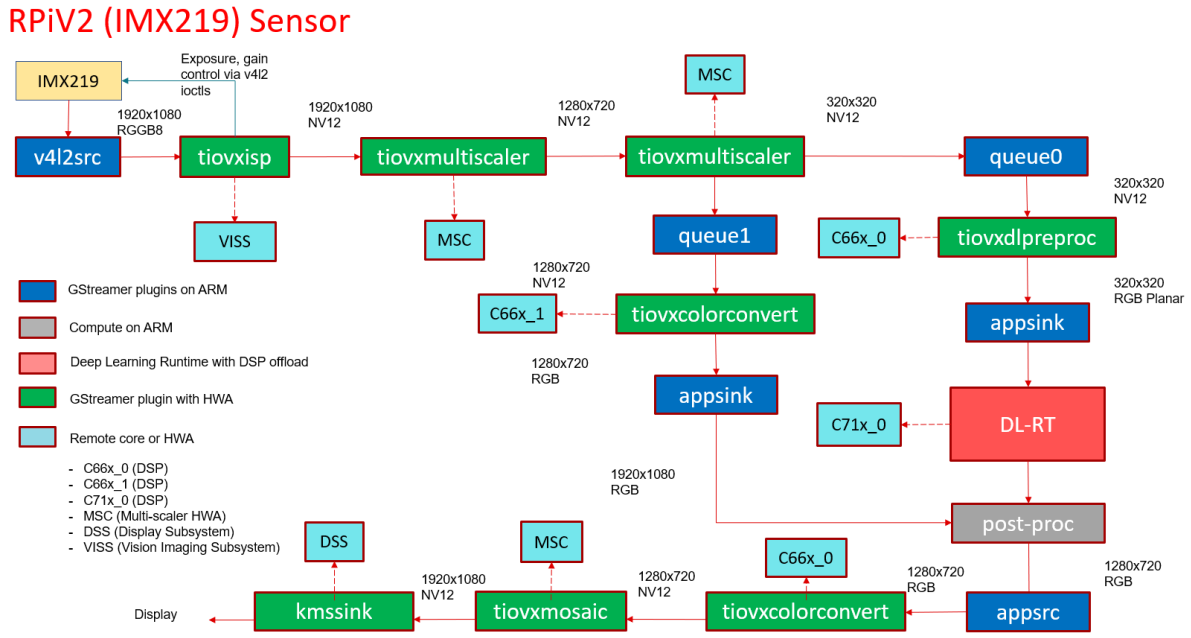


Fig. 5.12: GStreamer based data-flow pipeline with IMX219 sensor, ISP and display

IMX390 Camera Sensor

In this demo, raw frames in SRGGB12 format are captured from IMX390 camera sensor. VISS (Vision Imaging Subsystem) is used to process the raw frames and get the output in NV12, VISS also controls the sensor parameters like exposure, gain etc.. via v4l2 ioctls. This is followed by LDC (Lens Distortion Correction) required due to the fisheye lens. The NV12 output is split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 ! queue leaky=2 ! video/x-bayer, width=1936,␣
↪height=1100, format=rggb12 ! tiovxisp sink_0::device=/dev/v4l-subdev7␣
↪sensor-name=IMX390-UB953_D3 dcc-isp-file=/opt/imaging/imx390/dcc_viss.bin␣
↪sink_0::dcc-2a-file=/opt/imaging/imx390/dcc_2a.bin format-msb=11 ! video/x-
↪raw, format=NV12 ! tiovxldc dcc-file=/opt/imaging/imx390/dcc_ldc.bin␣
↪sensor-name=IMX390-UB953_D3 ! video/x-raw, format=NV12, width=1920,␣
↪height=1080 !tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=3 target=0 channel-order=0 tensor-format=bgr out-pool-size=4 !␣
↪application/x-tensor-tiovx ! appsink name=pre_0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
tiovxmosaic name=mosaic_0 background=/tmp/background_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

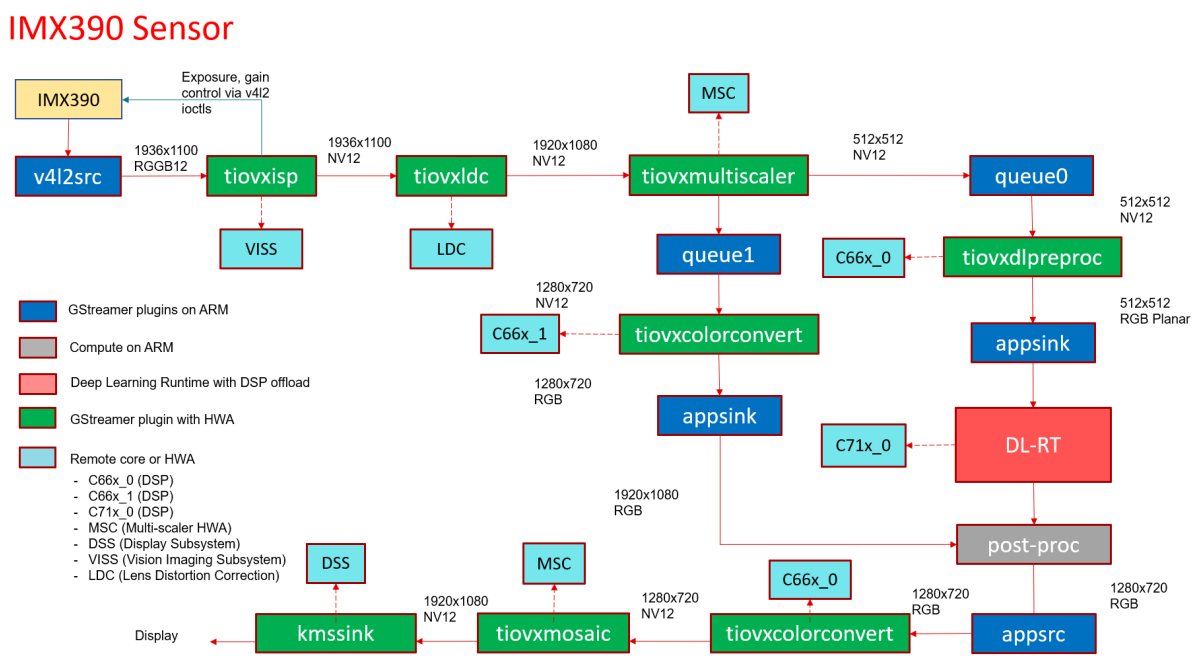


Fig. 5.13: GStreamer based data-flow pipeline with IMX390 sensor, ISP, LDC and display

Video output

In this demo, a frame is grabbed from an input source and split into two paths. The “analytics” path resizes the input to match the resolution required to run the deep learning network. The “visualization” path is provided to the post-processing module which does the required post process required by the model. Post-processed output is given to HW mosaic plugin which positions and resizes the output window on an empty background. Finally the video is encoded using the H.264 HW encoder and written to a video file.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 !␣
↪tiovxmultiscaler name=split_01
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=1280, height=720 !␣
↪tiovxdlcolorconvert target=1 out-pool-size=4 ! video/x-raw, format=RGB !␣
↪appsink name=sen_0 max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1280,␣
↪height=720 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=1280   sink_
↪0::height=720
! video/x-raw,format=NV12, width=1920, height=1080 ! v4l2h264enc␣
↪bitrate=10000000 ! h264parse ! matroskamux !  filesink location=/opt/edge_
↪ai_apps/data/output/videos/output_video.mkv
```

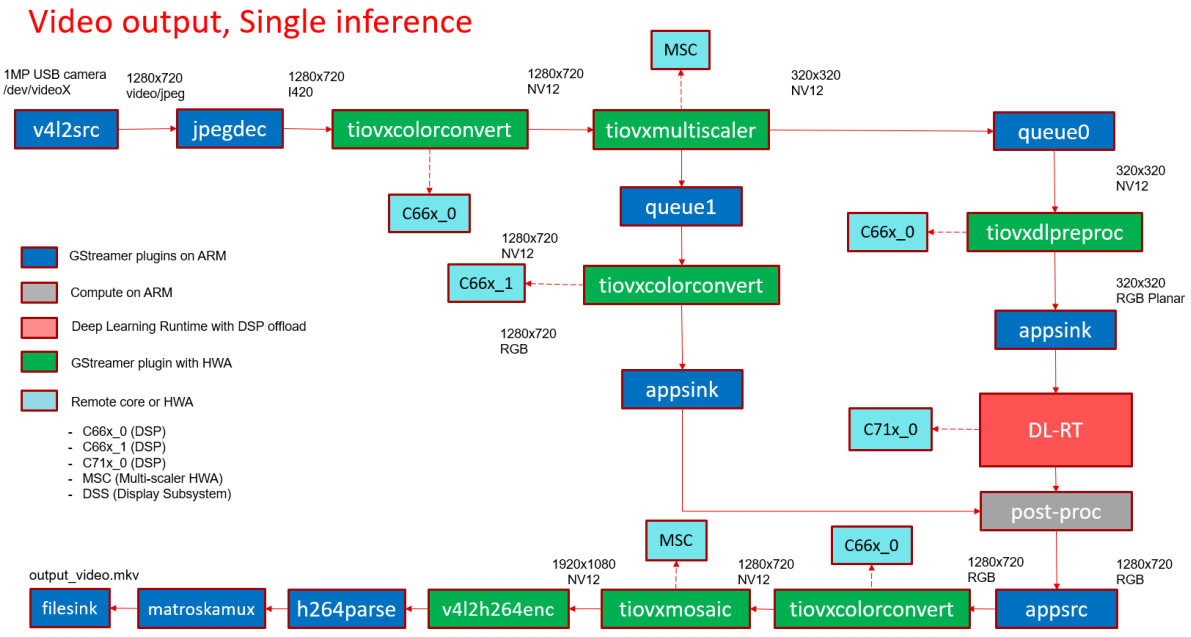


Fig. 5.14: GStreamer based data-flow pipeline with video file input source and display

Single Input Multi inference

In this demo, a frame is grabbed from an input source and split into multiple paths. Each path is further split into two sub-paths one for analytics and another for visualization. Each path can run any type of network, image classification, object detection, semantic segmentation and using any supported run-time.

For example the below GStreamer pipeline splits the input into 4 paths for running 4 deep learning networks. First is a semantic segmentation network, followed by object detection network, followed by two image classifi-

cation networks. If we look at the image classification path, the analytics sub-path resizes the input to maintain the aspect ratio and crops the input to match the resolution required to run the deep learning network. The visualization sub-path is provided to the post-processing module which overlays the detected classes. Post-processed output from all the 4 paths is given to HW mosaic plugin which positions and resizes the output windows on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 ! tee name=tee_
↪split0
tee_split0. ! queue ! tiovxmultiscaler name=split_01
tee_split0. ! queue ! tiovxmultiscaler name=split_02
tee_split0. ! queue ! tiovxmultiscaler name=split_03
tee_split0. ! queue ! tiovxmultiscaler name=split_04
split_01. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0␣
↪max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪1 max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1␣
↪max-buffers=2 drop=true
split_03. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=1 mean-0=128.
↪000000 mean-1=128.000000 mean-2=128.000000 scale-0=0.007812 scale-1=0.
↪007812 scale-2=0.007812 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_2 max-buffers=2 drop=true
split_03. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_2␣
↪max-buffers=2 drop=true
split_04. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_3 max-buffers=2 drop=true
split_04. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_3␣
↪max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_1 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_1
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_2 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_2
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_3 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
```

(continues on next page)

(continued from previous page)

```
↪height=360 ! queue ! mosaic_0.sink_3
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=640   sink_
↪0::height=360
sink_1::startx=960  sink_1::starty=180  sink_1::width=640   sink_
↪1::height=360
sink_2::startx=320  sink_2::starty=560  sink_2::width=640   sink_
↪2::height=360
sink_3::startx=960  sink_3::starty=560  sink_3::width=640   sink_
↪3::height=360
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

Multi Input Multi inference

In this demo, a frame is grabbed from multiple input sources and split into multiple paths. The multiple input sources could be either multiple cameras or a combination of camera, video, image, RTSP source. Each path is further split into two sub-paths one for analytics and another for visualization. Each path can run any type of network, image classification, object detection, semantic segmentation and using any supported run-time.

For example the below GStreamer pipeline splits two inputs into 4 paths for running 2 deep learning networks. First is a object detection network, followed by image classification networks. If we look at the image classification path, the analytics sub-path resizes the input to maintain the aspect ratio and crops the input to match the resolution required to run the deep learning network. The visualization sub-path is provided to the post-processing module which overlays the detected classes. Post-processed output from all the 4 paths is given to HW mosaic plugin which positions and resizes the output windows on an empty background before sending to display.

GStreamer input pipeline:

```
v4l2src device=/dev/video18 io-mode=2 ! image/jpeg, width=1280, height=720 !␣
↪jpegdec ! tiovxdlcolorconvert ! video/x-raw, format=NV12 ! tee name=tee_
↪split0
tee_split0. ! queue ! tiovxmultiscaler name=split_01
tee_split0. ! queue ! tiovxmultiscaler name=split_02
split_01. ! queue ! video/x-raw, width=320, height=320 ! tiovxdlpreproc data-
↪type=10 channel-order=1 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.007812 scale-1=0.007812 scale-2=0.007812 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪0 max-buffers=2 drop=true
split_01. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_0␣
↪max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=1 mean-0=128.
↪000000 mean-1=128.000000 mean-2=128.000000 scale-0=0.007812 scale-1=0.
↪007812 scale-2=0.007812 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_1 max-buffers=2 drop=true
split_02. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_1␣
↪max-buffers=2 drop=true

filesrc location=/opt/edge_ai_apps/data/videos/video_0000_h264.mp4 ! qtdemux␣
↪! h264parse ! v4l2h264dec ! video/x-raw, format=NV12  ! tee name=tee_split1
tee_split1. ! queue ! tiovxmultiscaler name=split_11
tee_split1. ! queue ! tiovxmultiscaler name=split_12
```

(continues on next page)

(continued from previous page)

```
split_11. ! queue ! video/x-raw, width=512, height=512 ! tiovxdlpreproc data-
↪type=10 channel-order=0 mean-0=128.000000 mean-1=128.000000 mean-2=128.
↪000000 scale-0=0.015625 scale-1=0.015625 scale-2=0.015625 tensor-
↪format=rgb out-pool-size=4 ! application/x-tensor-tiovx ! appsink name=pre_
↪2 max-buffers=2 drop=true
split_11. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_2␣
↪max-buffers=2 drop=true
split_12. ! queue ! video/x-raw, width=454, height=256 ! tiovxdlcolorconvert␣
↪out-pool-size=4 ! video/x-raw, format=RGB ! videobox left=115 right=115␣
↪top=16 bottom=16 ! tiovxdlpreproc data-type=10 channel-order=0 mean-0=123.
↪675000 mean-1=116.280000 mean-2=103.530000 scale-0=0.017125 scale-1=0.
↪017507 scale-2=0.017429 tensor-format=rgb out-pool-size=4 ! application/x-
↪tensor-tiovx ! appsink name=pre_3 max-buffers=2 drop=true
split_12. ! queue ! video/x-raw, width=640, height=360 ! tiovxdlcolorconvert␣
↪target=1 out-pool-size=4 ! video/x-raw, format=RGB ! appsink name=sen_3␣
↪max-buffers=2 drop=true
```

GStreamer output pipeline:

```
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_0 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_0
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_1 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_1
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_2 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_2
appsrc format=GST_FORMAT_TIME is-live=true block=true do-timestamp=true␣
↪name=post_3 ! tiovxdlcolorconvert ! video/x-raw,format=NV12, width=640,␣
↪height=360 ! queue ! mosaic_0.sink_3
appsrc format=GST_FORMAT_TIME block=true num-buffers=1 name=background_0 !␣
↪tiovxdlcolorconvert ! video/x-raw,format=NV12, width=1920, height=1080 !␣
↪queue ! mosaic_0.background
tiovxmosaic name=mosaic_0
sink_0::startx=320  sink_0::starty=180  sink_0::width=640   sink_
↪0::height=360
sink_1::startx=960  sink_1::starty=180  sink_1::width=640   sink_
↪1::height=360
sink_2::startx=320  sink_2::starty=560  sink_2::width=640   sink_
↪2::height=360
sink_3::startx=960  sink_3::starty=560  sink_3::width=640   sink_
↪3::height=360
! video/x-raw,format=NV12, width=1920, height=1080 ! kmssink sync=false␣
↪driver-name=tidss
```

5.1.8 Performance Visualization Tool

The performance visualization tool can be used to view all the performance statistics recorded when running the edge AI C++ demo application. This includes the CPU and HWA loading, DDR bandwidth, Junction Temperatures and FPS obtained. Refer to *Available options* for details on the performance metrics available to be plotted.

This tool works as follows:

- Logging: When running the application, the performance statistics can be recorded and stored in log files. This is done automatically when running the C++ application, but the Python application does not generate logs. However a standalone binary executable is provided that can be run in parallel with the Python application, which will generate these performance logs.
- Visualization: There is a Python script which parses these logs and plots graphs, which can be easily

viewed by a visiting a URL in any browser. This script uses Streamlit package to update the graphs in real-time, as the Edge AI application runs in parallel. However, since Streamlit is not supported in the SDK out of box, this script needs to run on docker. Please refer to *Docker Environment* for building and running a docker container.

5.1.9 Generating Performance Logs

Each log file contains real-time values for some performance metrics, averaged over a 2s window. The temperature sensor values are sampled in real time, every 2s. The performance visualization tool then parses these log files one by one based on the modification timestamps.

The edge AI C++ demo will automatically generate log files and store them in the directory `../perf_logs`, that is, one level up from where the C++ app is run. For example, if the app is run from `edge_ai_apps/apps_cpp`, the logs will be stored in `edge_ai_apps/perf_logs`.

Similarly, there is a binary executable that can be compiled that does the same logging standalone. The source for this is available under `edge_ai_apps/scripts/perf_stats/`. The README.md file has simple instructions to build and run this standalone logger binary. After building it, use following command to print the statistics on the terminal as well as save them in log files that can be parsed.

```
debian@beaglebone:/opt/edge_ai_apps/scripts/perf_stats/build# ../bin/Release/
↪ti_perfstats -l
```

5.1.10 Running the Visualization tool

To use this tool, simply start a docker session and then run the command given below. This script expects some log files to be present in the directory `edge_ai_apps/perf_logs` after running any C++ demo. One can also bring up this tool while running the demo but it might affect the performance of the demo itself as it consumes a bit of ARM cycles during launch but stabilizes over a certain duration.

```
[docker] debian@beaglebone:/opt/edge_ai_apps# streamlit run scripts/perf_vis.
↪py --theme.base="light"
```

This script also accepts the log directory as a command line argument as follows:

```
[docker] debian@beaglebone:/opt/edge_ai_apps# streamlit run scripts/perf_vis.
↪py --theme.base="light" -- -D <path/to/logs/directory/>
```

A network URL can be seen in the terminal output. The graphs can be viewed by visiting this URL in any browser. The plotted graphs will keep updating based on the available log files.

To exit press Ctrl+C in the terminal.

Available options

Average frames per second (FPS) recorded by the application is displayed by default. Using the checkboxes in the sidebar, one can select which performance metrics to view. There are 14 metrics available to be plotted, as seen from the above image:

- CPU Load: Total loading for the A72(mpu1_0), R5F(mcu2_0/1), C66x(c6x_1/2) and C71x(c7x_1) DSPs.
- HWA Load: Loading (percentage) for the various available hardware accelerators.
- DDR Bandwidth: Average read, write and total bandwidth recorded in the previous 2s interval.
- Junction Temperatures: The live temperatures recorded at various junctions
- Task Table: A separate graph for each cpu showing the loading due to various tasks running on it.
- Heap Table: A separate graph for each cpu showing the heap memory usage statistics.

Fig. 5.15: Performance visualizer dashboard showing CPU and HWA loading, DDR bandwidth, Junction Temperatures and the FPS obtained

For the first three metrics, there is a choice to view line graphs with a 30s history or bar graphs with only the real-time values. The remaining eleven have real-time bar graphs as the only option.

5.1.11 SDK Components

The BeagleBone® AI-64 Linux for Edge AI can be divided into 3 parts, Applications, BeagleBone® AI-64 Linux and Processor SDK RTOS. Users can get the latest application updates and bug fixes from the public repositories (GitHub and git.ti.com) which aligns with the SDK releases done quarterly. One can also build every component from source by following the steps in the TI Edge AI SDK development flow.

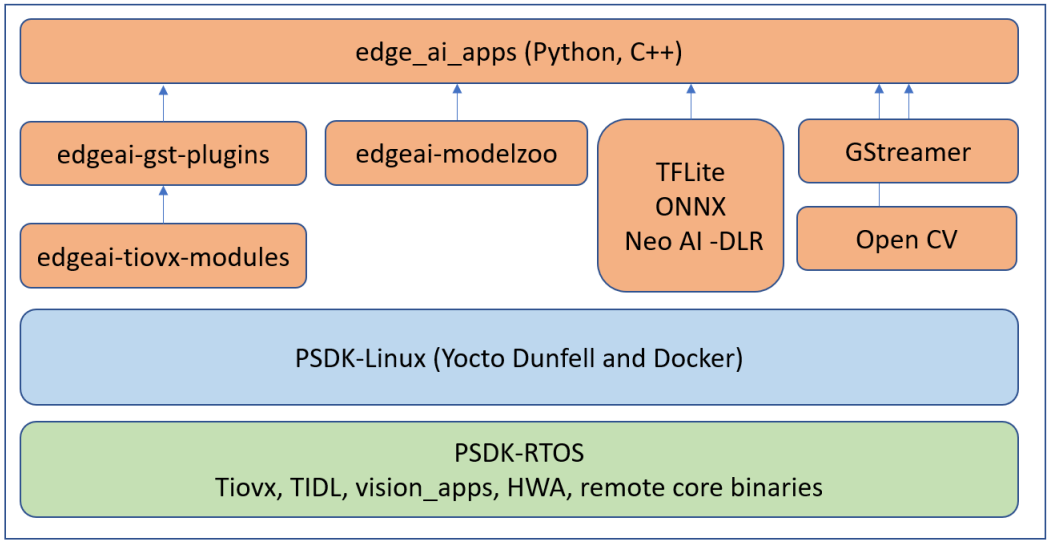


Fig. 5.16: BeagleBone® AI-64 Linux for Edge AI components

Edge AI Applications

The edge AI applications are designed for users to quickly evaluate various Deep Learning networks on TDA4 SoC. The user can run standalone examples and Jupyter notebook applications to evaluate inference models either from TI Edge AI Model Zoo or a custom network. Once a network is finalized for performance and accuracy it can also be easily integrated in a typical capture-inference-display usecase using example GStreamer based applications for rapid prototyping and deployment.

edgeai-tidl-tools This application repository provides standalone Python and C/C++ examples to quickly evaluate inference models using TFLite, ONNX and NeoAI-DLR runtime using file based inputs. It also houses the Jupyter notebooks similar to TI Edge AI Cloud which can be executed right on the TDA4VM Starter Kit.

For more details on using this application repo please refer to the documentation and source code found here: https://github.com/TexasInstruments/edgeai-tidl-tools

edgeai-modelzoo This repo provides collection of example Deep Neural Network (DNN) Models for various computer vision tasks. A few example models are packaged as part of the SDK to run out-of-box demos. More can be downloaded using a download script made available in the edge_ai_apps repo.

For more details on the pre-imported models and related documentation please visit: https://github.com/TexasInstruments/edgeai-modelzoo

edge_ai_apps These are plug-and-play Deep Learning applications which support running open source runtime frameworks such as TFLite, ONNX and NeoAI-DLR with a live camera and display. They help connect realtime camera, video or RTSP sources to DL inference to live display, bitstream or RTSP sinks.

The latest source code with fixes can be pulled from: https://git.ti.com/cgit/edgeai/edge_ai_apps

edgeai-gst-plugins This repo provides the source of custom GStreamer plugins which helps offload tasks to TDA4 hardware accelerators and advanced DSPs with the help of edgeai-tiovx-modules. The repo gets downloaded, built and installed as part of the *Software setup* step.

Source code and documentation: https://github.com/TexasInstruments/edgeai-gst-plugins

edgeai-tiovx-modules This repo provides OpenVx modules which help access underlying hardware accelerators in the TDA4 SoC and serves as a bridge between GStreamer custom elements and underlying OpenVx custom kernels. The repo gets downloaded, built and installed as part of the *Software setup* step.

Source code and documentation: https://github.com/TexasInstruments/edgeai-tiovx-modules

Processor SDK RTOS

The BeagleBone® AI-64 Linux for Edge AI gets all the HWA drivers, optimized libraries, OpenVx framework and more from Processor SDK RTOS

For more information visit Processor SDK RTOS Getting Started Guide.

BeagleBone® AI-64 Linux

The BeagleBone® AI-64 Linux for Edge AI gets all the Linux kernel, filesystem, device-drivers and more from BeagleBone® AI-64 Linux

For more information visit BeagleBone® AI-64 Linux Software Developer's Guide.

5.1.12 Datasheet

This chapter describes the performance measurements of the Edge AI Inference demos.

Performance data of the demos can be auto generated by running following command on target:

```
debian@beaglebone:/opt/edge_ai_apps/tests# ./gen_data_sheet.sh
```

The performance measurements includes the following

1. **FPS** : Effective framerate at which the application runs
2. **Total time** : Average time taken to process each frame, which includes pre-processing, inference and post-processing time
3. **Inference time** : Average time taken to infer each frame
4. **CPU loading** : Loading on different CPU cores present
5. **DDR BW** : DDR read and write BW used
6. **HWA Loading** : Loading on different Hardware accelerators present

Following are the latest performance numbers of the C++ demos:

Source : USB Camera

Capture Framerate : **30 fps** Resolution : **720p** format : **JPEG**

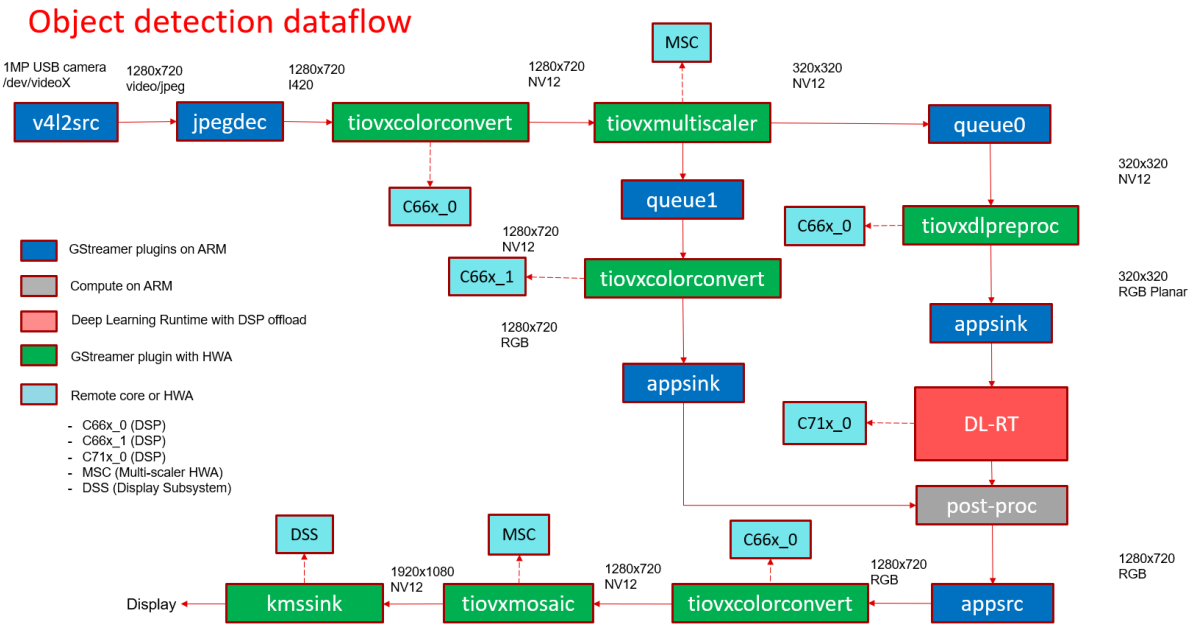


Fig. 5.17: GStreamer based data-flow pipeline with USB camera input and display output

Mod	FPS	To-tal time (ms)	In-fer-ence time (ms)	A72 Load (%)	DDR Reac BW (MB/	DDR Write BW (MB/	DDR To-tal BW (MB/	C71 Loac (%)	C66_ Load (%)	C66_ Load (%)	MCU Load (%)	MCU Load (%)	MSC (%)	MSC (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150 mobi 1p4-qat	30.8(	33.2	3.02	21.6(	1596	619	2215	9.0	20.0	9.0	6.0	1.0	22.1	0	0	0	0	0	0
TFL-CL-0000 mobi mlpe	30.6	33.1	1.04	15.9	1425	563	1988	5.0	22.0	9.0	6.0	1.0	21.9(	0	0	0	0	0	0
TFL-OD-2020 ssdLi mobI DSP-coco-320x	30.6	33.2	5.00	10.2	1534	570	2104	15.0	29.0	9.0	6.0	1.0	22.6	0	0	0	0	0	0
TVM-CL-3410 gluor mxne mob	30.5	33.2	2.02	22.8(	1522	617	2139	6.0	20.0	9.0	6.0	1.0	21.8	0	0	0	0	0	0

Source : Video

Video Framerate : **30 fps** Resolution : **720p** Encoding : **h264**

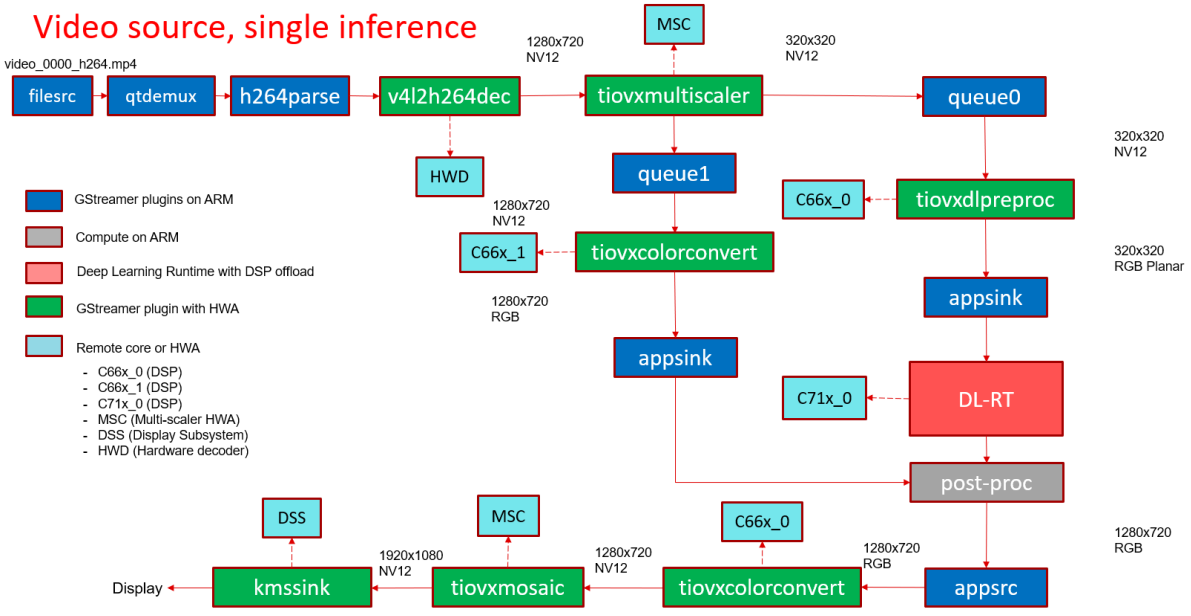


Fig. 5.18: GStreamer based data-flow pipeline with video file input source and display output

Mod	FPS	To-tal time (ms)	In-fer-ence time (ms)	A72 Load (%)	DDR Reac BW (MB/	DDR Write BW (MB/	DDR To-tal BW (MB/	C71 Loac (%)	C66_ Load (%)	C66_ Load (%)	MCU Loac (%)	MCU Loac (%)	MSC (%)	MSC (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150 mobi 1p4-qat	30.5	33.4	3.03	14.2	990	403	1393	2.0	7.0	4.0	1.0	1.0	10.2	0	0	0	0	0	0
TFL-CL-0000 mobi mlpe	30.7	33.4	1.07	30.7	746	97	843	2.0	2.0	1.0	1.0	1.0	15.7	0	0	0	0	0	0
TFL-OD-2020 ssdLi mobI DSP-coco-320x	30.5	33.5	5.06	22.5	736	92	828	2.0	2.0	1.0	1.0	1.0	16.9	0	0	0	0	0	0
TVM-CL-3410 gluor mxne mobv	30.6	33.4	2.01	33.3	712	110	822	1.0	1.0	0.0	1.0	1.0	15.3	0	0	0	0	0	0

Source : CSI Camera (ov5640)

Capture Framerate : **30 fps** Resolution : **720p** format : **YUYV**

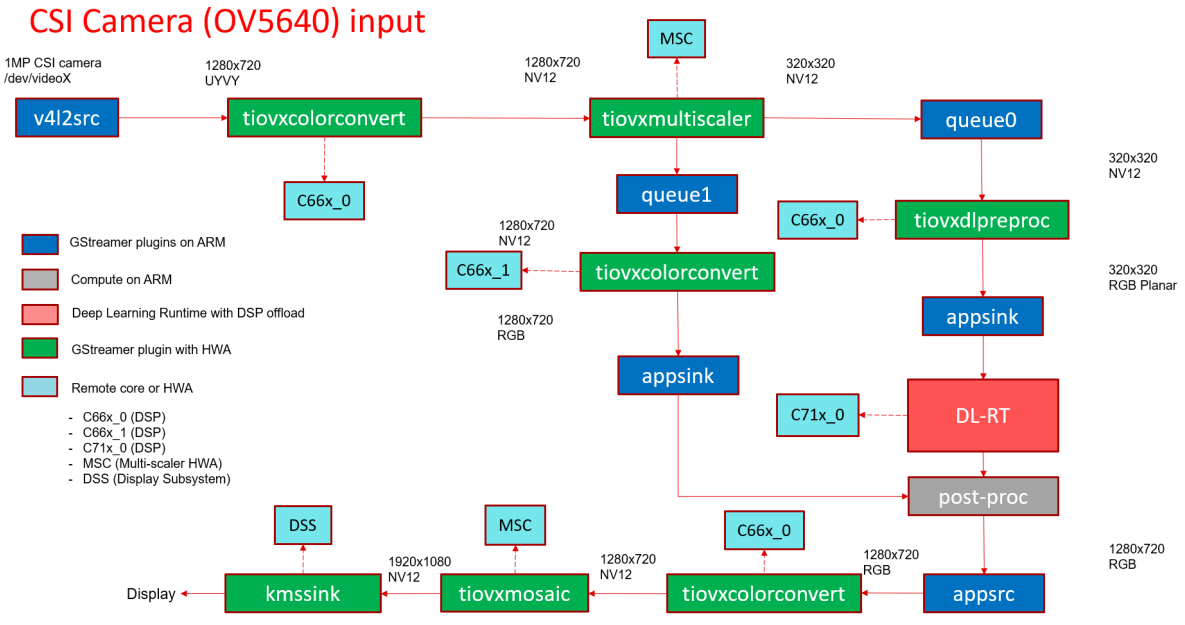


Fig. 5.19: GStreamer based data-flow pipeline for with CSI camera (OV5640) input and display output

Mod	FPS	To-tal time (ms)	In-fer-ence time (ms)	A72 Load (%)	DDR Reac BW (MB/	DDR Write BW (MB/	DDR To-tal BW (MB/	C71 Loac (%)	C66_ Load (%)	C66_ Load (%)	MCU Load (%)	MCU Load (%)	MSC (%)	MSC (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150 mobi 1p4-qat	29.5	34.0	3.02	12.2	1671	699	2370	8.0	45.0	9.0	6.0	1.0	21.3	0	0	0	0	0	0
TFL-CL-0000 mobi mlpe	29.4	34.1	1.01	10.2	1502	645	2147	5.0	47.0	9.0	6.0	1.0	20.9	0	0	0	0	0	0
TFL-OD-2020 ssdLi mobI DSP-coco-320x	29.3	34.6	5.00	10.5	1610	655	2265	14.0	53.0	9.0	6.0	1.0	21.4	0	0	0	0	0	0
TVM-CL-3410 gluor mxne mobv	29.3	34.1	2.01	11.6	1596	698	2294	6.0	45.0	9.0	5.0	1.0	21.1	0	0	0	0	0	0

Source : CSI Camera with VISS (imx219)

Capture Framerate : **30 fps** Resolution : **1080p** format : **SRGGB8**

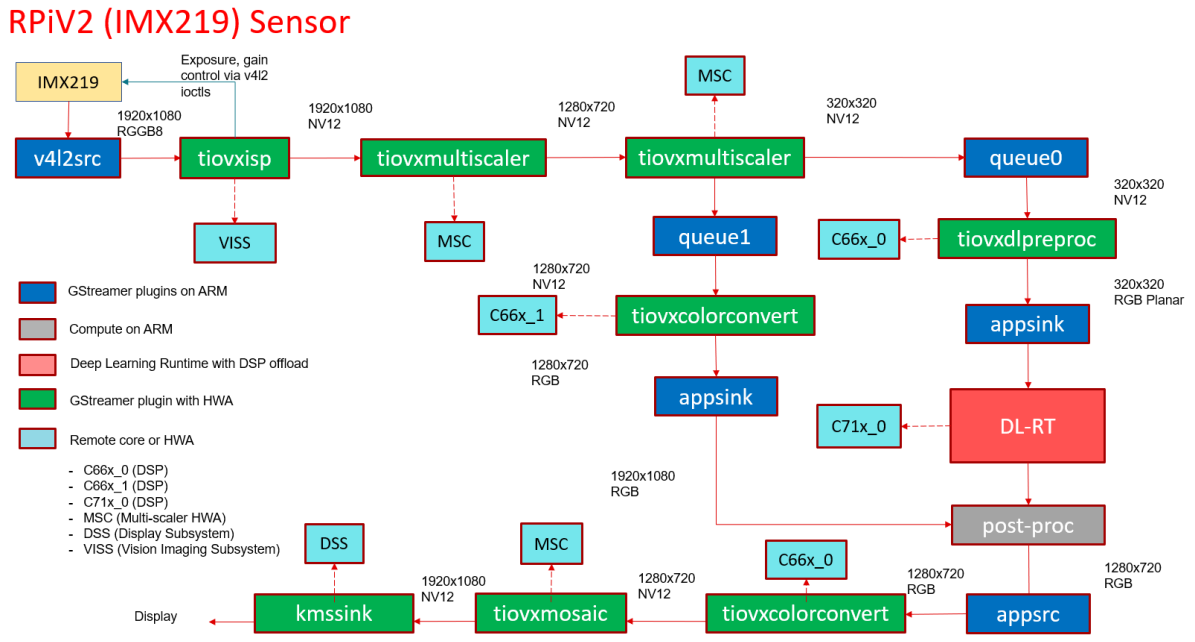


Fig. 5.20: GStreamer based data-flow pipeline with IMX219 sensor, ISP and display

Mod	FPS	To-tal time (ms)	In-fer-ence time (ms)	A72 Load (%)	DDR Reac BW (MB/	DDR Write BW (MB/	DDR To-tal BW (MB/	C71 Loac (%)	C66_ Load (%)	C66_ Load (%)	MCU Load (%)	MCU Load (%)	MSC (%)	MSC (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150 mobi 1p4-qat	30.6	33.1	3.01	15.7	1781	853	2634	9.0	16.0	9.0	13.0	1.0	31.7	0	22.3	0	0	0	0
TFL-CL-0000 mobi mlpe	30.5	33.1	1.04	12.7	1612	798	2410	5.0	18.0	9.0	13.0	1.0	31.6	0	22.3	0	0	0	0
TFL-OD-2020 ssdLi mobI DSP-coco-320x	30.5	33.0	5.00	13.3	1730	809	2539	15.0	25.0	9.0	13.0	1.0	32.6	0	22.1	0	0	0	0
TVM-CL-3410 gluor mxne mob	30.4	33.1	2.01	12.9	1708	852	2560	7.0	16.0	9.0	13.0	1.0	31.8	0	22.2	0	0	0	0

Source : IMX390 over FPD-Link

Capture Framerate : **30 fps** Resolution : **1080p** format : **SRGGB12**

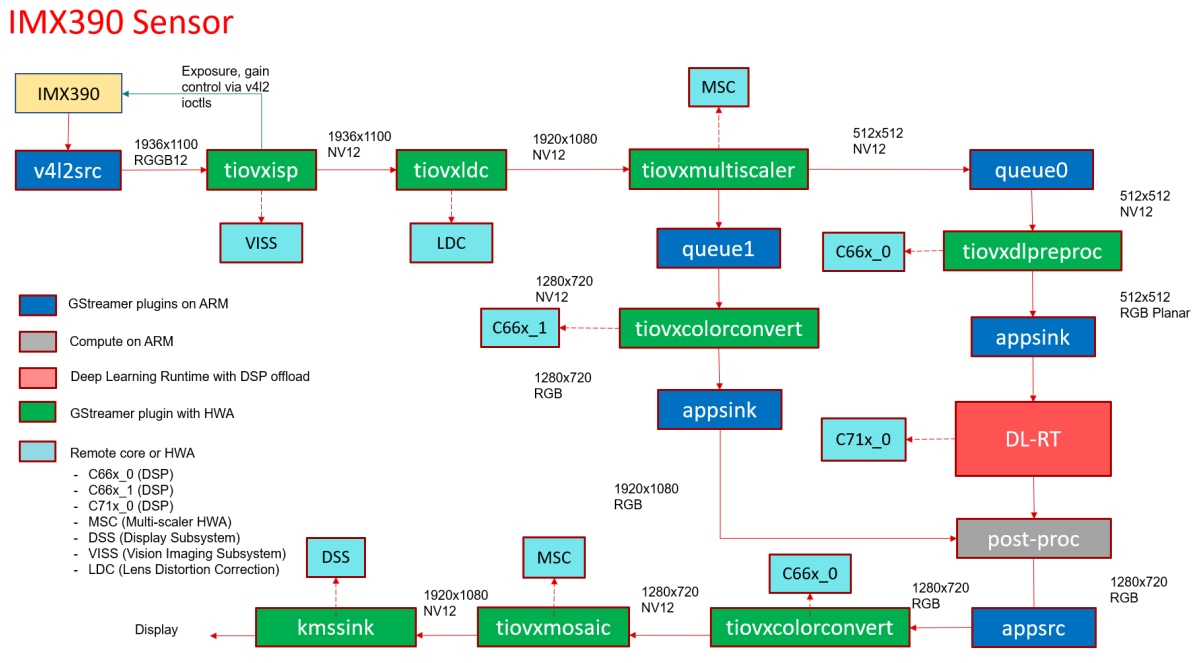


Fig. 5.21: GStreamer based data-flow pipeline with IMX390 sensor, ISP, LDC and display

Mod	FPS	To-tal time (ms)	In-fer-ence time (ms)	A72 Load (%)	DDR Reac BW (MB/	DDR Write BW (MB/	DDR To-tal BW (MB/	C71 Load (%)	C66_ Load (%)	C66_ Load (%)	MCU Load (%)	MCU Load (%)	MSC (%)	MSC (%)	VISS (%)	NF (%)	LDC (%)	SDE (%)	DOF (%)
ONR-CL-6150 mobi 1p4-qat	30.5	33.1	3.09	25.1	2207	1102	3309	10.0	16.0	9.0	14.0	1.0	31.7	0	22.9	0	10.8	0	0
TFL-CL-0000 mobi mlpe	30.5	33.1	1.21	16.2	2019	1040	3059	5.0	18.0	9.0	15.0	1.0	32.8	0	23.3	0	10.1	0	0
TFL-OD-2020 ssdLi mobI DSP-coco-320x	30.4	33.1	5.02	23.7	2201	1067	3268	15.0	25.0	9.0	14.0	1.0	32.8	0	22.8	0	9.95	0	
TVM-CL-3410 gluor mxne mob	30.4	33.1	2.12	21.5	2111	1100	3211	7.0	16.0	9.0	15.0	1.0	32.2	0	22.8	0	10.6	0	0

5.1.13 Test Report

Here is the summary of the sanity tests we ran with both Python and C++ demos. Test cases vary with different inputs, outputs, runtime, models, python/c++ apps.

1. Inputs:
    - Camera (Logitech C270, 1280x720, JPEG)
    - Camera (Omnivision OV5640, 1280x720, YUV)
    - Camera (Rpi v2 Sony IMX219, 1920x1080, RAW)
    - Image files (30 images under edge_ai_apps/data/images)
    - Video file (10s video 1 file under edge_ai_apps/data/videos)
    - RSTP Video Server
2. Outputs:
    - Display (eDP or HDMI)
    - File write to SD card
3. Inference Type:
    - Image classification
    - Object detection
    - Semantic segmentation
4. Runtime/models:
    - DLR
    - TFLite

- ONNX

5. Applications:
    - Python
    - C++
6. Platform:
    - Host OS
    - Docker

Demo Apps test report

Single Input Single Output

Category	# test case	Pass	Fail
Host OS - Python	99	99	0
Host OS - C++	99	99	0

S.No	Models	Input	Output	Host OS-C+
1	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Display	Pass
2	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Video-Filewrite	Fail
3	TVM-CL-3410-gluoncv-mxnet-mobv2	Image	Image-Filewrite	Pass
4	TVM-CL-3410-gluoncv-mxnet-mobv2	Video	Display	Pass
5	TVM-CL-3410-gluoncv-mxnet-mobv2	Video	Video-Filewrite	Pass
6	TVM-CL-3410-gluoncv-mxnet-mobv2	USB Camera	Display	Pass
7	TVM-CL-3410-gluoncv-mxnet-mobv2	USB Camera	Video-Filewrite	Pass
8	TVM-CL-3410-gluoncv-mxnet-mobv2	CSI Camera	Display	Pass
9	TVM-CL-3410-gluoncv-mxnet-mobv2	CSI Camera	Video-Filewrite	Pass
10	TVM-CL-3410-gluoncv-mxnet-mobv2	RPI Camera	Display	Pass
11	TVM-CL-3410-gluoncv-mxnet-mobv2	RPI Camera	Video-Filewrite	Pass
12	TVM-CL-3410-gluoncv-mxnet-mobv2	RTSP - Video	Display	Pass
13	TVM-CL-3410-gluoncv-mxnet-mobv2	RTSP - Video	Video-Filewrite	Pass
14	TFL-CL-0000-mobileNetV1-mlperf	Image	Display	Pass
15	TFL-CL-0000-mobileNetV1-mlperf	Image	Video-Filewrite	Fail
16	TFL-CL-0000-mobileNetV1-mlperf	Image	Image-Filewrite	Pass
17	TFL-CL-0000-mobileNetV1-mlperf	Video	Display	Pass
18	TFL-CL-0000-mobileNetV1-mlperf	Video	Video-Filewrite	Pass
19	TFL-CL-0000-mobileNetV1-mlperf	USB Camera	Display	Pass
20	TFL-CL-0000-mobileNetV1-mlperf	USB Camera	Video-Filewrite	Pass
21	TFL-CL-0000-mobileNetV1-mlperf	CSI Camera	Display	Pass
22	TFL-CL-0000-mobileNetV1-mlperf	CSI Camera	Video-Filewrite	Pass
23	TFL-CL-0000-mobileNetV1-mlperf	RPI Camera	Display	Pass
24	TFL-CL-0000-mobileNetV1-mlperf	RPI Camera	Video-Filewrite	Pass
25	TFL-CL-0000-mobileNetV1-mlperf	RTSP - Video	Display	Pass
26	TFL-CL-0000-mobileNetV1-mlperf	RTSP - Video	Video-Filewrite	Pass
27	ONR-CL-6360-regNetx-200mf	Image	Display	Pass
28	ONR-CL-6360-regNetx-200mf	Image	Video-Filewrite	Fail
29	ONR-CL-6360-regNetx-200mf	Image	Image-Filewrite	Pass
30	ONR-CL-6360-regNetx-200mf	Video	Display	Pass
31	ONR-CL-6360-regNetx-200mf	Video	Video-Filewrite	Pass
32	ONR-CL-6360-regNetx-200mf	USB Camera	Display	Pass
33	ONR-CL-6360-regNetx-200mf	USB Camera	Video-Filewrite	Pass
34	ONR-CL-6360-regNetx-200mf	CSI Camera	Display	Pass
35	ONR-CL-6360-regNetx-200mf	CSI Camera	Video-Filewrite	Pass
36	ONR-CL-6360-regNetx-200mf	RPI Camera	Display	Pass

Table 5.1 – continued from previous page

S.No	Models	Input	Output	Host OS-C+
37	ONR-CL-6360-regNetx-200mf	RPI Camera	Video-Filewrite	Pass
38	ONR-CL-6360-regNetx-200mf	RTSP - Video	Display	Pass
39	ONR-CL-6360-regNetx-200mf	RTSP - Video	Video-Filewrite	Pass
40	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Display	Pass
41	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Video-Filewrite	Fail
42	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Image	Image-Filewrite	Pass
43	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Video	Display	Pass
44	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	Video	Video-Filewrite	Pass
45	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	USB Camera	Display	Pass
46	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	USB Camera	Video-Filewrite	Pass
47	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	CSI Camera	Display	Pass
48	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	CSI Camera	Video-Filewrite	Pass
49	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RPI Camera	Display	Pass
50	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RPI Camera	Video-Filewrite	Pass
51	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RTSP - Video	Display	Pass
52	TVM-OD-5020-yolov3-mobv1-gluon-mxnet-coco-416x416	RTSP - Video	Video-Filewrite	Pass
53	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Display	Pass
54	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Video-Filewrite	Fail
55	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Image	Image-Filewrite	Pass
56	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Video	Display	Pass
57	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	Video	Video-Filewrite	Pass
58	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	USB Camera	Display	Pass
59	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	USB Camera	Video-Filewrite	Pass
60	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	CSI Camera	Display	Pass
61	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	CSI Camera	Video-Filewrite	Pass
62	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RPI Camera	Display	Pass
63	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RPI Camera	Video-Filewrite	Pass
64	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RTSP - Video	Display	Pass
65	TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	RTSP - Video	Video-Filewrite	Pass
66	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Display	Pass
67	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Video-Filewrite	Fail
68	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Image	Image-Filewrite	Pass
69	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Video	Display	Pass
70	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	Video	Video-Filewrite	Pass
71	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	USB Camera	Display	Pass
72	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	USB Camera	Video-Filewrite	Pass
73	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	CSI Camera	Display	Pass
74	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	CSI Camera	Video-Filewrite	Pass
75	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RPI Camera	Display	Pass
76	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RPI Camera	Video-Filewrite	Pass
77	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RTSP - Video	Display	Pass
78	ONR-OD-8050-ssd-lite-regNetX-800mf-fpn-bgr-mmdet-coco-512x512	RTSP - Video	Video-Filewrite	Pass
79	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Display	Pass
80	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Video-Filewrite	Fail
81	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Image	Image-Filewrite	Pass
82	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Video	Display	Pass
83	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	Video	Video-Filewrite	Pass
84	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	USB Camera	Display	Pass
85	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	USB Camera	Video-Filewrite	Pass
86	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	CSI Camera	Display	Pass
87	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	CSI Camera	Video-Filewrite	Pass
88	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RPI Camera	Display	Pass
89	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RPI Camera	Video-Filewrite	Pass
90	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RTSP - Video	Display	Pass
91	TVM-SS-5720-deeplabv3lite-regnetx800mf-cocoseg21-512x512	RTSP - Video	Video-Filewrite	Pass

Table 5.1 – continued from previous page

S.No	Models	Input	Output	Host OS-C+
92	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Display	Pass
93	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Video-Filewrite	Fail
94	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Image	Image-Filewrite	Pass
95	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Video	Display	Pass
96	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	Video	Video-Filewrite	Pass
97	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	USB Camera	Display	Pass
98	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	USB Camera	Video-Filewrite	Pass
99	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	CSI Camera	Display	Pass
100	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	CSI Camera	Video-Filewrite	Pass
101	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RPI Camera	Display	Pass
102	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RPI Camera	Video-Filewrite	Pass
103	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RTSP - Video	Display	Pass
104	TFL-SS-2580-deeplabv3_mobv2-ade20k32-mlperf-512x512	RTSP - Video	Video-Filewrite	Pass
105	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Display	Pass
106	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Video-Filewrite	Fail
107	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Image	Image-Filewrite	Pass
108	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Video	Display	Pass
109	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	Video	Video-Filewrite	Pass
110	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	USB Camera	Display	Pass
111	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	USB Camera	Video-Filewrite	Pass
112	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	CSI Camera	Display	Pass
113	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	CSI Camera	Video-Filewrite	Pass
114	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RPI Camera	Display	Pass
115	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RPI Camera	Video-Filewrite	Pass
116	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RTSP - Video	Display	Pass
117	ONR-SS-8610-deeplabv3lite-mobv2-ade20k32-512x512	RTSP - Video	Video-Filewrite	Pass

Single Input Multi Output

Category	# test case	Pass	Fail
Host OS - Python	15	15	0
docker - Python	15	15	0
Host OS - C++	15	15	0
Docker - C++	15	15	0

S.No	Models	Input	Output	Host OS-C++	Host OS-Python	Docker-C++	Docker-Python	Comments
1	2 Models (TFL-CL, ONR-SS)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
2	3-Models (TVM-CL, TFL-OD, ONR-SS)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
3	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	%04d.jpg	Display	Pass	Pass	Pass	Pass	
4	2 Models (TFL-CL, ONR-SS)	video_0000.m	Display	Pass	Pass	Pass	Pass	
5	3-Models (TVM-CL, TFL-OD, ONR-SS)	video_0000.m	Display	Pass	Pass	Pass	Pass	
6	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	video_0000.m	Display	Pass	Pass	Pass	Pass	
7	2 Models (TFL-CL, ONR-SS)	USB_camera	Display	Pass	Pass	Pass	Pass	
8	3-Models (TVM-CL, TFL-OD, ONR-SS)	USB_camera	Display	Pass	Pass	Pass	Pass	
9	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	USB_camera	Display	Pass	Pass	Pass	Pass	
10	2 Models (TFL-CL, ONR-SS)	CSI_camera	Display	Pass	Pass	Pass	Pass	
11	3-Models (TVM-CL, TFL-OD, ONR-SS)	CSI_camera	Display	Pass	Pass	Pass	Pass	
12	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	CSI_camera	Display	Pass	Pass	Pass	Pass	
13	2 Models (TFL-CL, ONR-SS)	rtsp	Display	Pass	Pass	Pass	Pass	
14	3-Models (TVM-CL, TFL-OD, ONR-SS)	rtsp	Display	Pass	Pass	Pass	Pass	
15	4-Models (TVM-SS, TFL-OD, ONR-SS, ONR-CL)	rtsp	Display	Pass	Pass	Pass	Pass	

Multi Input Multi Output

Category	# test case	Pass	Fail
Host OS - Python	8	8	0
docker - Python	8	8	0
Host OS - C++	8	8	0
Docker - C++	8	8	0

S.No	Models	Input	Output	Host OS-C++	Host OS-Python	Docker-C++	Docker-Python	Comments
1	2 Models (TVM-CL, TFL-OD)	%04d.jpg,video_0000.mp4	Display	Pass	Pass	Pass	Pass	
2	2 Models (TVM-OD, ONR-SS)	%04d.jpg,rtsp	Video-Filewrite	Pass	Pass	Pass	Pass	
3	2 Models (ONR-CL, TVM-SS)	%04d.jpg,USB_camera	Display	Pass	Pass	Pass	Pass	
4	3-Models (TVM-CL, TFL-OD, ONR-SS)	%04d.jpg,CSI_camera,rtsp	Video-Filewrite	Pass	Pass	Pass	Pass	
5	3-Models (TVM-CL, TFL-OD, ONR-SS)	video_0000.mp4,rtsp,%04	Display	Pass	Pass	Pass	Pass	
6	3-Models (TFL-CL, ONR-CL, TVM-SS)	video_0000.mp4,USB_cam	Video-Filewrite	Pass	Pass	Pass	Pass	
7	4-Models (TVM-CL, TFL-SS, ONR-OD, TFL-CL)	USB_camera,CSI_camera	Display	Pass	Pass	Pass	Pass	
8	4-Models (TVM-SS, TFL-SS, ONR-SS, ONR-OD)	USB_camera,video_0000.r	Video-Filewrite	Pass	Pass	Pass	Pass	

Note:

- Video file from RTSP server used for RTSP input test

- Please refer to the TI Edge AI SDK release notes and known issues for more details

Chapter 6

Additional Support Information

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

6.1 Certifications and export control

6.1.1 Export designations

- HS: 8471504090
- US HS: 8543708800
- EU HS: 8471707000

6.2 Hardware Design

You can find all BeagleBone AI-64 hardware files here under the *hw* folder.

6.3 Production board boot media

- BeagleBone AI-64 Rev B1

6.4 Software Updates

Follow instructions below to download the latest image for your BeagleBone AI-64:

1. Go to BeagleBoard.org distro page.
2. *Filter Software Distributions for BeagleBone AI-64* from dropdown and download the image.

Tip: You can follow the flash-latest-image guide for more information on flashing the downloaded image to your board.

To see what SW revision is loaded into the eMMC check */etc/dogtag*. It should look something like as shown below,

```
root@BeagleBone:~# cat /etc/dogtag
BeagleBoard.org Debian Bullseye Xfce Image 2022-01-14
```

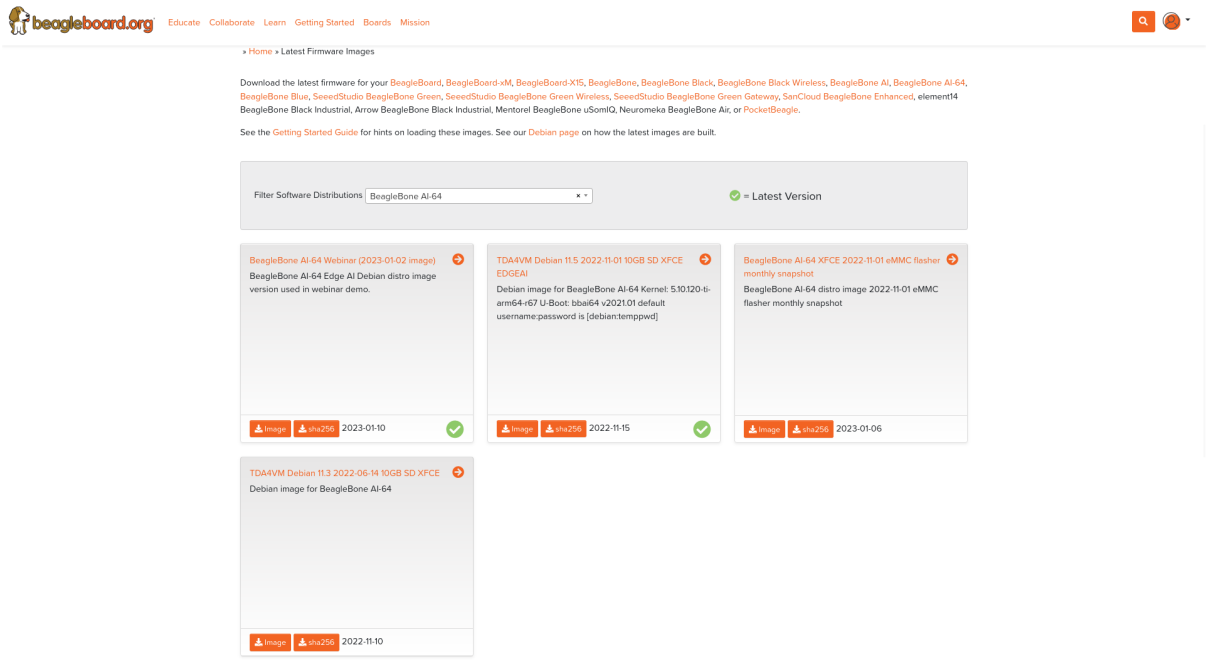


Fig. 6.1: Filter Software Distributions for BeagleBone AI-64

6.5 RMA Support

If you feel your board is defective or has issues, request an Return Merchandise Application (RMA) by filling out the form at http://beagleboard.org/support/rma . You will need the serial number and revision of the board. The serial numbers and revisions keep moving. Different boards can have different locations depending on when they were made. The following figures show the three locations of the serial and revision number.

6.6 Troubleshooting video output issues

Warning: When connecting to an HDMI monitor, make sure your miniDP adapter is *active*. A *passive* adapter will not work. See accessories-cables_minidp_hdmi accessories section for tested cables list.

6.7 Getting Help

If you need some up to date troubleshooting techniques, you can post your queries on link: BeagleBoard.org forum

6.8 Change History

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.

6.8.1 Document Change History

This table seeks to keep track of major revision cycles in the documentation. Moving forward, we'll seek to align these version numbers across all of the various documentation.

Table 6.1: Table 1: Change History

Rev	Changes	Date	By
0.0.1	AI-64 initial prototype	September 2021	James Anderson
0.0.2	AI-64 final prototype	December 2021	James Anderson
0.0.3	AI-64 initial production release	June 9, 2022	Deepak Khatri and Jason Kridner

6.8.2 Board Changes

Be sure to check the board revision history in the schematic file in the BeagleBone AI-64 git repository . Also check the issues list .

Rev B

We are starting with revision B based on this being an update to the BeagleBone Black AI. However, because this board ended up being so different, we've decided to name it BeagleBone AI-64, rather than simply a new revision. This refers to the Seeed release on 21 Dec 2021 of "BeagleBone AI-64_SCH_Rev B_211221". This is the initial production release.

6.9 Mechanical Details

6.9.1 Dimensions and Weight

Table 6.2: Dimensions & weight

Parameter	Value
Size	104 * 83* 37 mm
Max heigh	23 mm
PCB Size	102.5*80*2.0 mm
PCB Layers	14 layers
PCB Thickness	2.0 mm
RoHS compliant	Yes
Gross Weight	249g
Net Weight	193g

6.9.2 Silkscreen and Component Locations

6.10 Pictures

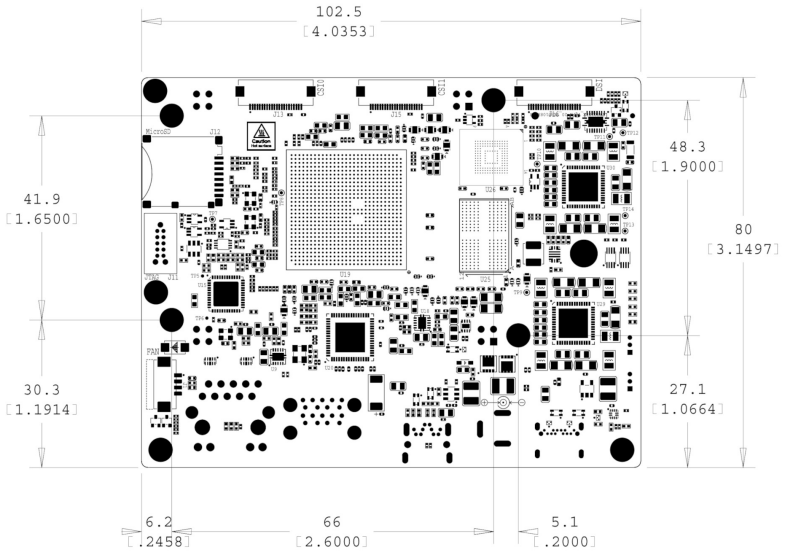


Fig. 6.2: Board Dimensions

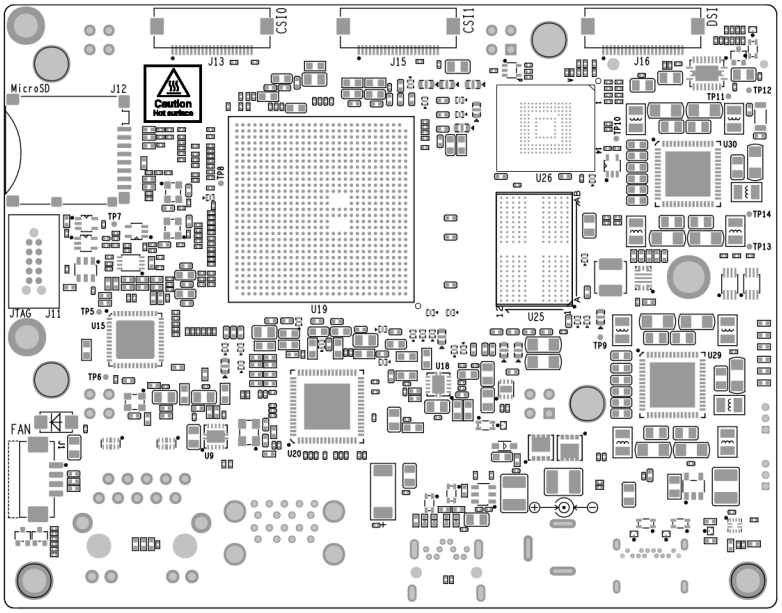


Fig. 6.3: Top silkscreen

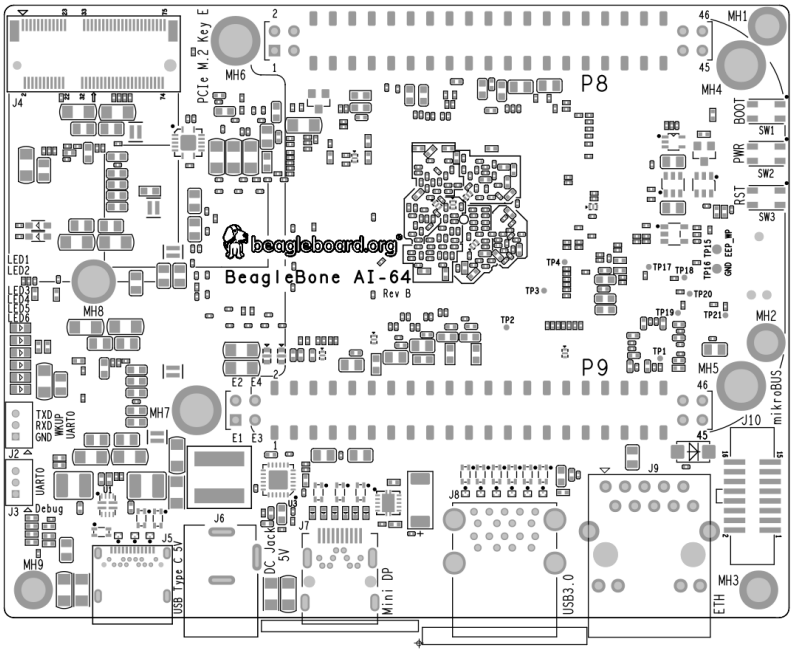


Fig. 6.4: Bottom silkscreen

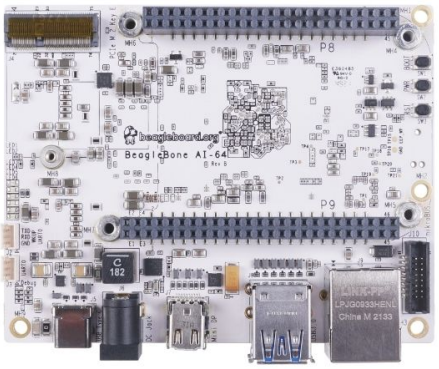


Fig. 6.5: BeagleBone AI-64 front

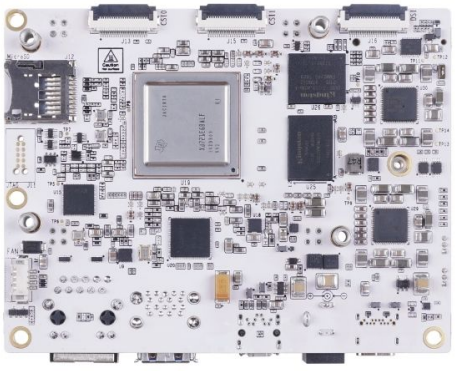


Fig. 6.6: BeagleBone AI-64 back

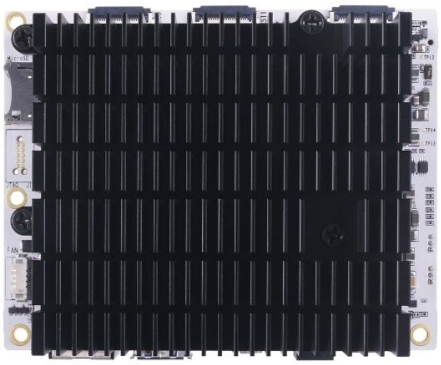


Fig. 6.7: BeagleBone AI-64 back with heatsink

Fig. 6.8: BeagleBone AI-64 front at 45° angle

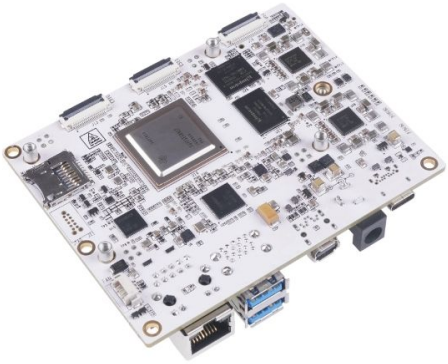


Fig. 6.9: BeagleBone AI-64 back at 45° angle

Fig. 6.10: BeagleBone AI-64 back with heatsink at 45° angle

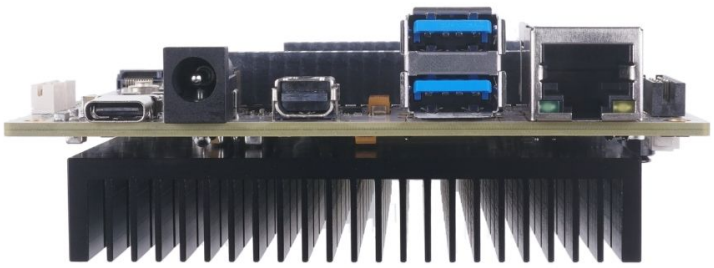


Fig. 6.11: BeagleBone AI-64 ports