CHERRY

JAGUAR SBC-RK3588-AMR

Single Board Computer for Autonomous Mobile Robots
featuring the Rockchip **RK3588** cutting-edge processor

USER MANUAL

Document revision:	v1.1.0
Issue date:	Apr 12, 2024

Contents

1 Introduction

Congratulations for acquiring CHERRY Embedded Solutions new product, combining best-in-class performance with a rich set of peripherals.

Note: The latest version of this manual and related resources can always be found on our website at the following address:

https://embedded.cherry.de/product/jaguar-sbc-rk3588/

1.1 Device Overview

Cutting-edge performance for autonomous mobile robots (AMR)

Due to its top performance CPU in parallel with GPU and NPU, algorithms can be based on rules, statistical methods, and neural learning. The Rockchip RK3588 is a low-power octa-core processor for Internet of Things (IoT) devices with artificial intelligence (AI). It features four ARM Cortex-A76 and four ARM Cortex-A55 cores, a Mali G610 GPU, and an NPU with up to 6 TOPs. The 64-bit-capable ARMv8 cores support both ARM Cryptographic Extension (for wire-rate AES cryptography) and AdvSIMD vector processing. The JAGUAR SBC-RK3588-AMR uses up to 16 GB LPDDR4X memory and up to 256 GB eMMC storage on board, and optionally external storage media such as NVMe/SSD, and SD card.

Native support of four high-resolution cameras

The JAGUAR SBC-RK3588-AMR module provides MIPI-CSI interfaces for four high-resolution cameras. Using a MIPI-CSI interface not only reduces the costs of camera modules, but also provides a continuous stream of raw video data into the processor – regardless of USB or Ethernet protocol. The four cameras can be grouped into two pairs that are synchronized (vsync), enabling their data to be easily combined into an accurate 3-dimensional point cloud. In addition to the video input signal, the two camera ports can also receive the signals of Inertial Measurement Units (IMU) integrated in the cameras, allowing to track accelerations and turns.

Simple integration into the design of your robot

The JAGUAR SBC-RK3588-AMR comes with common ports for the various interfaces, which provides both simple interfacing to the robotic device and low connector costs for the overall product. Additional interfaces can be utilized by an optional extension board that plugs into a specific connector called a "mezzanine" connector because it adds a second floor. The JAGUAR SBC-RK3588-AMR operates from a single power input with a voltage range of 12 to 24 V. While its consumption under load is a moderate 18 W, it provides up to 35 W for devices attached via PCIe and USB. It also routes the voltage input to an optional extension board on a mezzanine connector. Last but not least, the Rockchip RK3588 is located on the bottom of the PCBA as the highest element on this side, which means that, heat can be easily dissipated by simply interfacing the Rockchip RK3588 to the chassis with a thermal conductor.

State-of-the-art security for your assets

The JAGUAR SBC-RK3588-AMR features a secure element in addition to the capability to enable a Secure Boot mechanism. This secure element is based on the GlobalPlatform 2.2.1 compliant JavaCard environment. Secure boot guarantees that only signed images can run on the device. Enjoy the peace of mind that comes with a government-grade security solution for all identification, key-storage and asset-protection requirements. The Common Criteria (EAL6+) certified security module ensures that you never have to sacrifice security for performance again.

Note: In favor of readability in this document JAGUAR SBC-RK3588-AMR will be referenced by its code name Jaguar

1.2 Precautions

Warning: ESD Sensitive Device

Electronic boards and their components are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections of this product, in order to ensure product integrity at all times.

Do not handle this product out of its protective enclosure while it is not used for operational purposes unless it is otherwise protected. Whenever possible, unpack or pack this product only at EOS/ESD safe work stations. Where a safe work station is not guaranteed, it is important for the user to be electrically discharged before touching the product with hands or tools.

2 Interfaces

Jaguar provides a wide variety of interfaces.

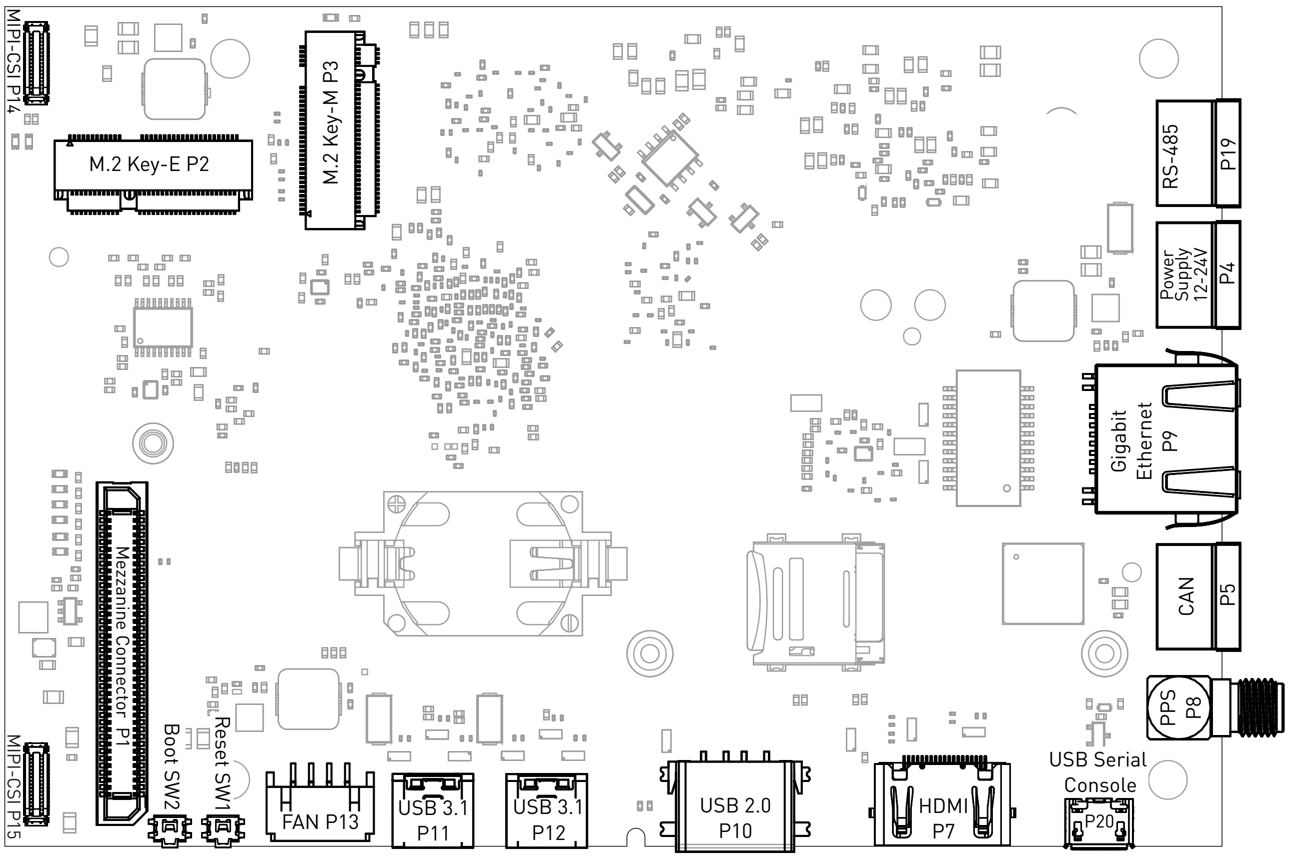

Fig. 2.1: Jaguar interfaces overview

2.1 Power Supply

In order to power the board, connect the appropriate cable to the highlighted connector shown in the figure below. The Jaguar power supply voltage is 12-24V.

Note: Be careful when connecting the power cable since all three terminal block connectors are of the same type.

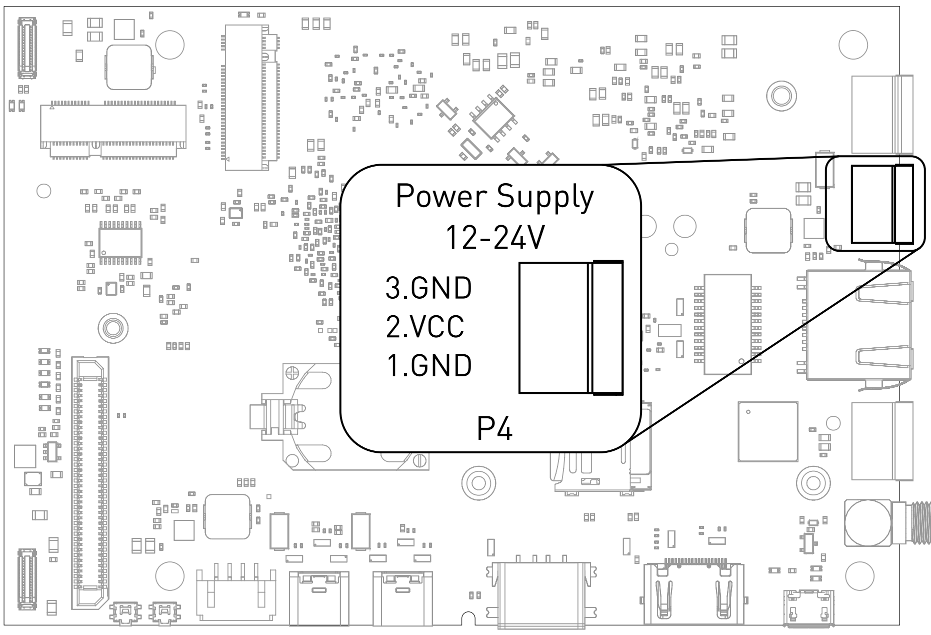

Fig. 2.2: Power connector

Table 2.1: Compatible mating connectors

Manufacturer	Partnumber	Description
Würth	691361100003	Vertical
Würth	691363110003	Horizontal with hook on wire Side
Würth	691366110003	Horizontal with hook on back Side
Würth	691304100003	Screwless Plug Vertical Entry Low Profile

2.2 USB Serial Console

Jaguar contains an on-board Silicon Labs CP2102N USB-serial converter. Connect a Micro-USB cable to the Micro-USB as highlighted below:

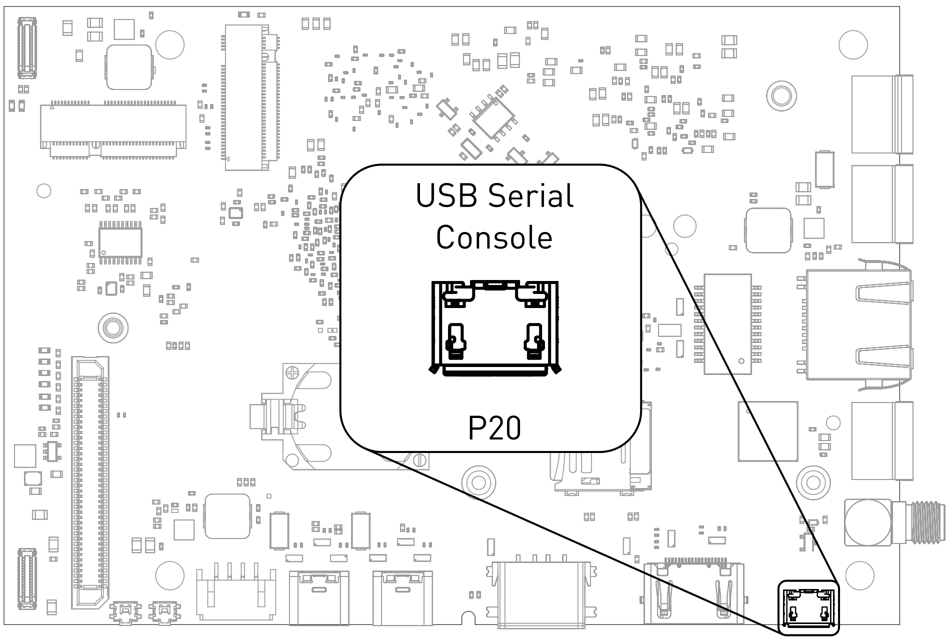

Fig. 2.3: USB UART

For macOS and Windows, drivers are available from Silicon Labs: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

A terminal emulation program is required to access the serial console.

Table 2.2: Terminal emulators recommendations

Operation System	Terminal Emulator	URL	Example Commandline
Microsoft Windows	PuTTY	https://www.chiark.greenend.org.uk/~sgtatham/putty/	
	MobaXterm	https://mobaxterm.mobatek.net/	
	Tera Term	https://ttssh2.osdn.jp/	
macOS	cu		`cu -s 115200 -l /dev/cu.SLAB_USBtoUART`
Linux	picocom	https://github.com/npat-efault/picocom/	`picocom -b 115200 /dev/ttyUSB0`
	minicom	https://salsa.debian.org/minicom-team/minicom/	`minicom -b 115200 -D /dev/ttyUSB0`
	GNU Screen	https://www.gnu.org/software/screen/	`screen /dev/ttyUSB0 115200`

Note: Make sure to disable software flow-control (XON/XOFF). Otherwise, serial input may not be recognized.

After system boot-up with the Jaguar Debian development image, the login console appears on the terminal:

```
jaguar login:
```

You can log with one of the following credentials:

Table 2.3: Default User

Username	Password
root	root
user	123123

2.3 Buttons

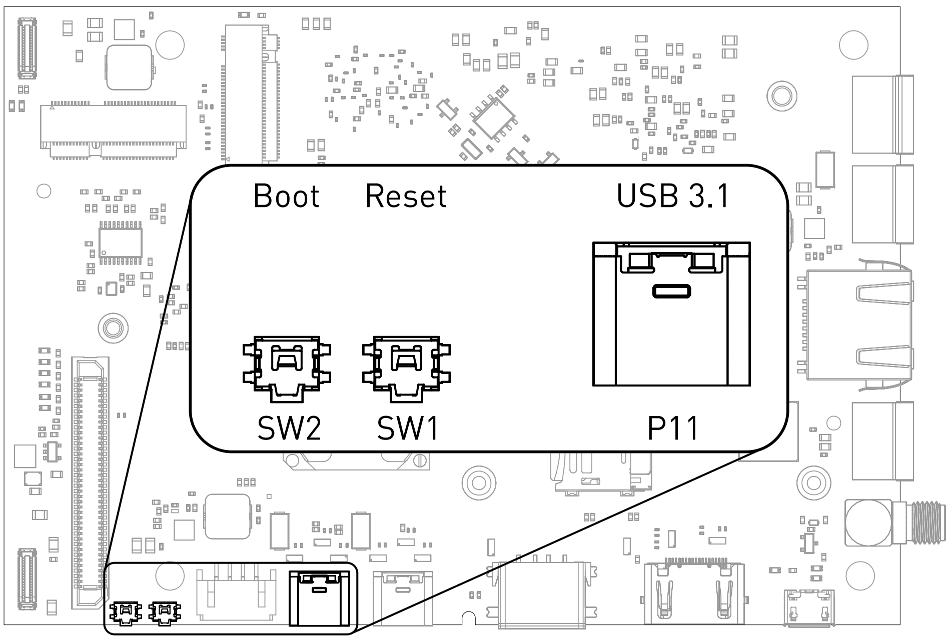

Fig. 2.4: Buttons and Download USB Type-C port

The control buttons provide the following functionality:

- `Reset` triggers a board reset.
- `BOOT SW` forces alternate boot sequence.

2.3.1 Boot Order

The used boot order of the Jaguar board depends on the state of the `BOOT SW` switch.

	Default	`BOOT SW` pressed
1	eMMC storage	SD card
2	SD card	USB loader
3	USB loader	

If no bootloader is found on any storage device, Jaguar will go into USB loader mode, showing up as a USB device with VID:PID `2207:350b` on the USB port P11 marked **Download**.

Once booted into Linux, presses on the `BOOT SW` button will trigger a `KEY_VENDOR` input event on the `/dev/input/by-path/platform-adc-keys-event` input device:

```
$ evtest /dev/input/by-path/platform-adc-keys-event
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "adc-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 360 (KEY_VENDOR)
Properties:
Testing ... (interrupt to exit)
Event: time 1695722632.280609, type 1 (EV_KEY), code 360 (KEY_VENDOR), value 1
Event: time 1695722632.280609, -------------- SYN_REPORT ------------
```

(continues on next page)

(continued from previous page)

```
Event: time 1695722632.383952, type 1 (EV_KEY), code 360 (KEY_VENDOR), value 0
Event: time 1695722632.383952, -------------- SYN_REPORT ------------
```

2.4 FAN

A PWM controlled fan with tacho signal can be connected. The supply voltage can be selected by changing a 0-Ohm resistor, the default supply is 12 V.

Table 2.4: Fan supply (bold default)

Resistor	Fan supply
R314	Main supply voltage
R315	**12 V**
R316	5 V

Table 2.5: Fan mating connector

Manufacturer	Partnumber
JST	PHR-4

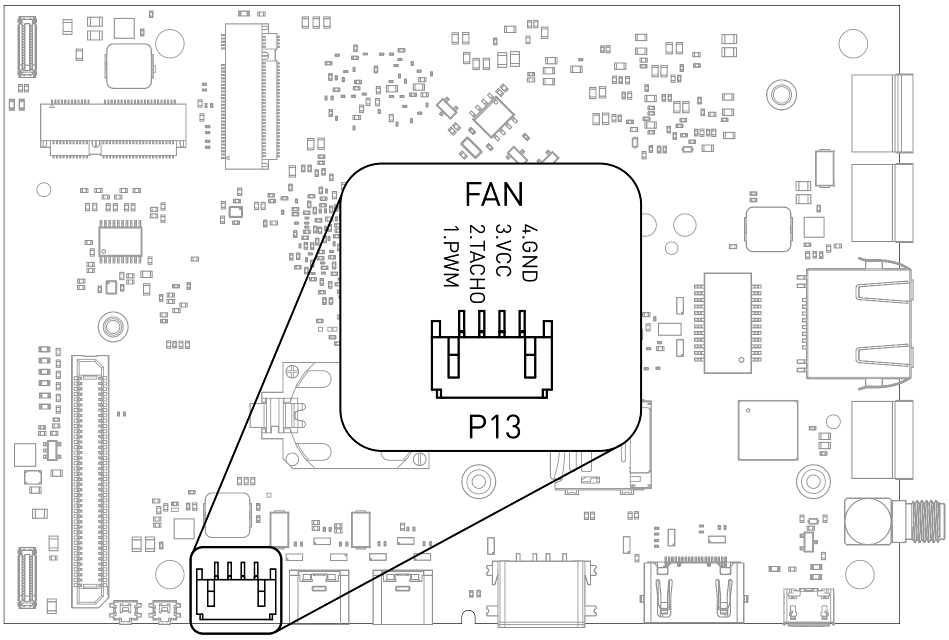

Fig. 2.5: FAN connection

2.5 CAN

Jaguar supports up to three CAN busses. CAN0 has an on-board transceiver and supports up to 1 MBaud data rate. CAN1 and CAN2 are available on the Mezzanine connector and require a transceiver on the Mezzanine board.

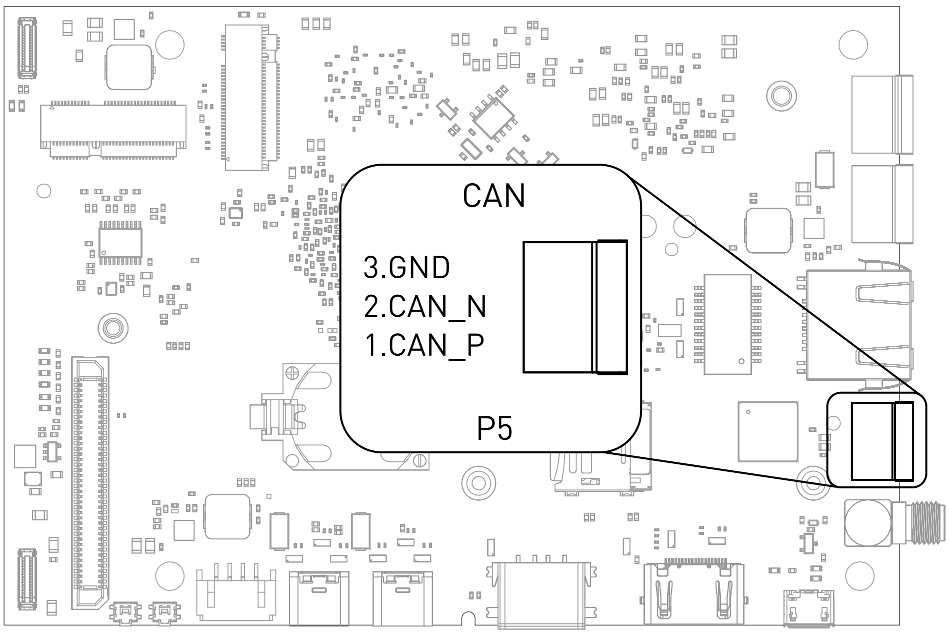

Fig. 2.6: CAN connector

Table 2.6: Compatible mating connectors

Manufacturer	Partnumber	Description
Würth	691361100003	Vertical
Würth	691363110003	Horizontal with hook on wire Side
Würth	691366110003	Horizontal with hook on back Side
Würth	691304100003	Screwless Plug Vertical Entry Low Profile

2.6 RS-485

Jaguar supports half-duplex RS-485.

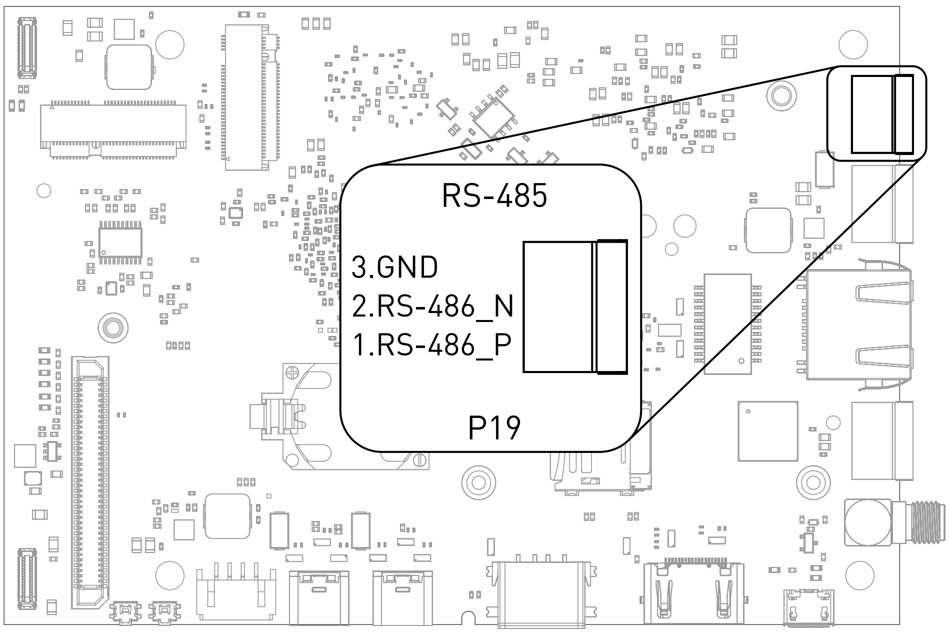

Fig. 2.7: RS-485 connector

Table 2.7: Compatible mating connectors

Manufacturer	Partnumber	Description
Würth	691361100003	Vertical
Würth	691363110003	Horizontal with hook on wire Side
Würth	691366110003	Horizontal with hook on back Side
Würth	691304100003	Screwless Plug Vertical Entry Low Profile

2.7 Battery

A CR2032 coin cell can be used to supply the on-board real-time-clock. The coin cell is only used when Jaguar is not supplied power from an external source.

2.8 Mezzanine Connector

Jaguar has an 80-pin connector for extensions board. Most pins have multiple functions that can be selected via software configuration. See Table 2.10 Mezzanine multiplex functions.

Table 2.8: Compatible mating connectors

Manufacturer	Partnumber	Description
Hirose	ER8-80P-0.8SV-2H	2mm height plug
Hirose	ER8-80P-0.8SV-5H	5mm height plug

Table 2.9: Mezzanine connector pinout

Pin	Function	Pin	Function
1	CP2102_POWER_EN	41	GND
2	CAM0_STROBE	42	GPIO3_C0
3	ADC_IN2	43	CAM2_CLK0_P
4	CAM1_STROBE	44	GPIO3_C1
5	WDTRIG#	45	CAM2_CLK0_N
6	I2C1_SCL	46	GPIO3_C2
7	GND	47	GND
8	I2C1_SDA	48	GPIO3_C3
9	CAM3_D1_P	49	CAM2_MCLK
10	GND	50	GPIO3_C4
11	CAM3_D1_N	51	CAM2_RST
12	X	52	GPIO3_C5
13	GND	53	PCIE20_0_RX_P
14	GPIO3_A0	54	GPIO3_C6
15	CAM3_D0_P	55	PCIE20_0_RX_N
16	GPIO3_A1	56	GPIO3_C7
17	CAM3_D0_N	57	GND
18	GPIO3_A2	58	GPIO3_D0
19	GND	59	PCIE20_0_TX_P
20	GPIO3_A3	60	GPIO3_D1
21	CAM3_CLK0_P	61	PCIE20_0_TX_N
22	GPIO3_A4	62	GPIO3_D2
23	CAM3_CLK0_N	63	GND
24	GPIO3_A5	64	GPIO3_D3
25	GND	65	PCIE20_0_CLK_P
26	GPIO3_A6	66	GPIO3_D4
27	CAM3_MLCK	67	PCIE20_0_CLK_N

continues on next page

Table 2.9 – continued from previous page

Pin	Function	Pin	Function
28	GPIO3_B1	68	GPIO3_D5
29	CAM3_RST	69	GND
30	GPIO3_B2	70	GND
31	CAM2_D1_P	71	VCC_1V8
32	GPIO3_B3	72	VCC_3V3
33	CAM2_D1_N	73	VCC_1V8
34	GPIO3_B4	74	VCC_3V3
35	GND	75	GND
36	GPIO3_B5	76	GND
37	CAM2_D0_P	77	VCC_IN
38	GPIO3_B6	78	VCC_5V0
39	CAM2_D0_N	79	VCC_IN
40	GPIO3_B7	80	VCC_5V0

Table 2.10: Mezzanine multiplex functions

Pin Number	GPIO Name	PWM	SDIO	I2S	I2C
14	GPIO3_A0	PWM10_M0	SDIO_D0_M1	I2S3_MCLK	I2C6_SDA_M4
16	GPIO3_A1	PWM11_IR_M0	SDIO_D1_M1	I2S3_SCLK	I2C6_SCL_M4
18	GPIO3_A2		SDIO_D2_M1	I2S3_LRCK	
20	GPIO3_A3		SDIO_D3_M1	I2S3_SDO	
22	GPIO3_A4		SDIO_CMD_M1	I2S3_SDI	
24	GPIO3_A5		SDIO_CLK_M1		I2C4_SDA_M0
26	GPIO3_A6				I2C4_SCL_M0
49	GPIO3_A7	PWM8_M0			
27	GPIO3_B0	PWM9_M0			
28	GPIO3_B1	PWM2_M1			
30	GPIO3_B2	PWM3_IR_M1		I2S2_SDI_M1	
32	GPIO3_B3			I2S2_SDO_M1	
34	GPIO3_B4			I2S2_MCLK_M1	
36	GPIO3_B5	PWM12_M0		I2S2_SCLK_M1	
38	GPIO3_B6	PWM13_M0		I2S2_LRCK_M1	
40	GPIO3_B7				
42	GPIO3_C0				
44	GPIO3_C1				
46	GPIO3_C2	PWM14_M0			
48	GPIO3_C3	PWM15_IR_M0			
50	GPIO3_C4				
52	GPIO3_C5				
54	GPIO3_C6				
56	GPIO3_C7				
58	GPIO3_D0	PWM8_M2			
60	GPIO3_D1	PWM9_M2			
62	GPIO3_D2				
64	GPIO3_D3	PWM10_M2			
66	GPIO3_D4				
68	GPIO3_D5	PWM11_IR_M3			

Table 2.11: Mezzanine multiplex functions (cont.)

Pin Number	GPIO Name	CAN	PCIE	UART
14	GPIO3_A0			
16	GPIO3_A1			
18	GPIO3_A2			UART8_TX_M1
20	GPIO3_A3			UART8_RX_M1

continues on next page

Table 2.11 – continued from previous page

Pin Number	GPIO Name	CAN	PCIE	UART
22	GPIO3_A4			UART8_RSTN_M1
24	GPIO3_A5			UART8_CTSN_M1
26	GPIO3_A6			
49	GPIO3_A7			
27	GPIO3_B0			
28	GPIO3_B1			
30	GPIO3_B2			
32	GPIO3_B3			
34	GPIO3_B4			
36	GPIO3_B5	CAN1_RX_M0		
38	GPIO3_B6	CAN1_TX_M0		
40	GPIO3_B7			
42	GPIO3_C0			
44	GPIO3_C1		PCIE30X2_BUTTON_RSTN_M1	
46	GPIO3_C2			
48	GPIO3_C3			
50	GPIO3_C4	CAN2_RX_M0		UART5_TX_M1
52	GPIO3_C5	CAN2_TX_M0	PCIE30X4_WAKEN_M2	UART5_RX_M1
54	GPIO3_C6			
56	GPIO3_C7		PCIE20X1_2_CLKREQN_M0	
58	GPIO3_D0		PCIE20X1_2_WAKEN_M0	UART4_RX_M1
60	GPIO3_D1		PCIE20X1_2_PERSTN_M0	UART4_TX_M1
62	GPIO3_D2		PCIE30X2_CLKREQN_M2	UART9_RTSN_M2
64	GPIO3_D3		PCIE30X2_WAKEN_M2	UART9_CTSN_M2
66	GPIO3_D4		PCIE30X2_PERSTN_M2	UART9_RX_M2
68	GPIO3_D5		PCIE30X4_BUTTON_RSTN	UART9_TX_M2

Table 2.12: Mezzanine multiplex functions (cont.)

Pin Number	GPIO Name	ETH	MIPI_CAMERA_CLK	SPI
14	GPIO3_A0	GMAC1_TXD2		SPI4_MISO_M1
16	GPIO3_A1	GMAC1_TXD3		SPI4_MOSI_M1
18	GPIO3_A2	GMAC1_RXD2		SPI4_CLK_M1
20	GPIO3_A3	GMAC1_RXD3		SPI4_CS0_M1
22	GPIO3_A4	GMAC1_TXCLK		SPI4_CS1_M1
24	GPIO3_A5	GMAC1_RXCLK		
26	GPIO3_A6	ETH1_REFCLKO_25M		
49	GPIO3_A7	GMAC1_RXD0	MIPI_CAMERA2_CLK_M1	
27	GPIO3_B0	GMAC1_RXD1	MIPI_CAMERA3_CLK_M1	
28	GPIO3_B1	GMAC1_RXDV_CRS	MIPI_CAMERA4_CLK_M1	
30	GPIO3_B2	GMAC1_TXER		
32	GPIO3_B3	GMAC1_TXD0		
34	GPIO3_B4	GMAC1_TXD1		
36	GPIO3_B5	GMAC1_TXEN		
38	GPIO3_B6	GMAC1_MCLKINOUT		
40	GPIO3_B7	GMAC1_PTP_REF_CLK		SPI1_MOSI_M1
42	GPIO3_C0	GMAC_PPSTRIG		SPI1_MISO_M1
44	GPIO3_C1	GMAC1_PPSCLK		SPI1_CLK_M1
46	GPIO3_C2	GMAC1_MDC		SPI1_CS0_M1
48	GPIO3_C3	GMAC1_MDIO		SPI1_CS1_M1
50	GPIO3_C4			SPI3_CS0_M3
52	GPIO3_C5			SPI3_CS1_M3
54	GPIO3_C6			SPI3_MISO_M3
56	GPIO3_C7			SPI3_MOSI_M3
58	GPIO3_D0			SPI3_CLK_M3
60	GPIO3_D1			

continues on next page

Table 2.12 – continued from previous page

Pin Number	GPIO Name	ETH	MIPI_CAMERA_CLK	SPI
62	GPIO3_D2			
64	GPIO3_D3			
66	GPIO3_D4			
68	GPIO3_D5			

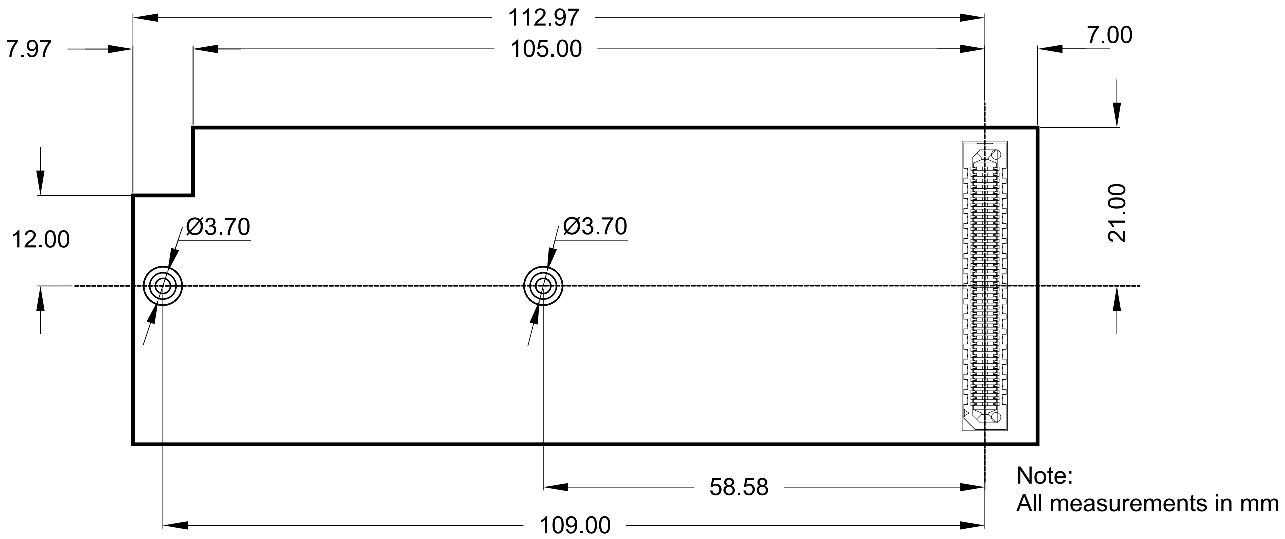

Fig. 2.8: Mezzanine board dimensions (bottom view)

3 Mechanical Specification

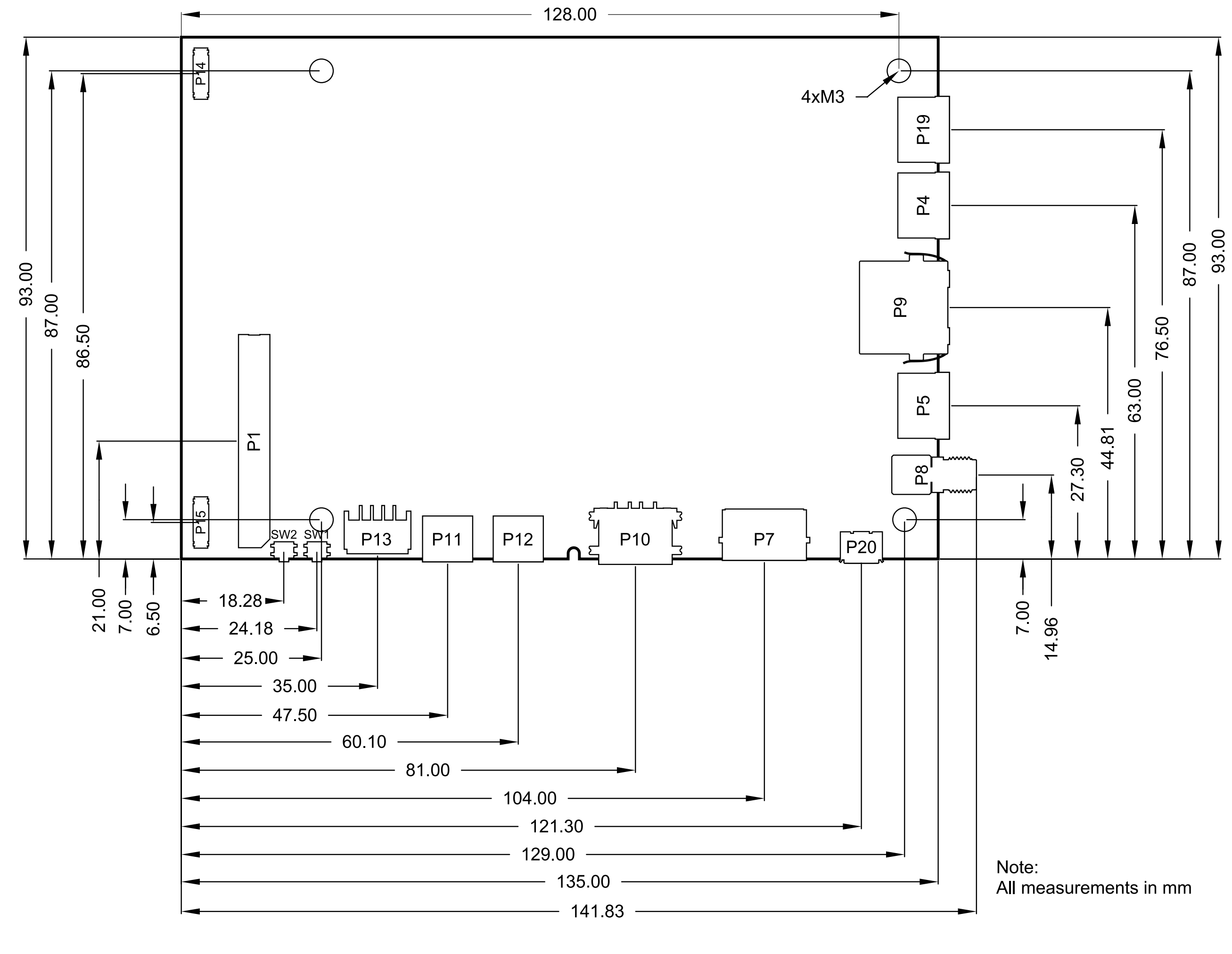

Fig. 3.1: Mechanical dimensions

4 Software Overview

This chapter provides instructions for compiling and deploying the BSP (Board Support Package) software to the board.

4.1 Supported Distributions

The following chapters describe how to build a disk image for:

- Debian: Section 5 Debian image guide
- Yocto: Section 6 Building a Yocto image

4.2 Compiling Linux Applications

The easiest option is to compile your applications directly on a board running Debian. Install the `gcc` package and related utilities and you are good to go:

```
sudo apt-get install build-essential
```

The second option is to cross-compile your applications on a host PC. The compiler that is installed in Section 5.1 Prepare the host PC is suitable.

4.3 Known issues

1. The SD card cannot currently be automatically detected at runtime, so please insert it before booting the Jaguar.

   It is however possible to force a detection at runtime by running the following commands:

   ```
   echo "fe2c0000.mmc" > /sys/bus/platform/drivers/dwmmc_rockchip/unbind
   echo "fe2c0000.mmc" > /sys/bus/platform/drivers/dwmmc_rockchip/bind
   ```

5 Debian image guide

As opposed to Yocto, Debian does not provide a completely integrated build experience by itself. Linux kernel and U-Boot have to be compiled manually and copied to the appropriate directory to be picked up by Debian build system.

This chapter will go through all neccessary steps, finally building a complete image using the debos Debian image builder. The result will be a fully-functional Debian system.

Alternatively, prebuilt images can be downloaded from https://downloads.embedded.cherry.de/jaguar .

At the time of writing this document, the following Debian image variants are available for Jaguar:

- Debian 12 Bookworm

Note: While Debian is a great tool for fast prototyping of your product, it is highly recommended to use a distribution/image tailored to your need. This can be achieved by Yocto or Buildroot for example.

5.1 Prepare the host PC

The debos Debian OS Builder is only available for Debian and Debian-based distributions (like Ubuntu). This chapter assumes you use Debian or a Debian-based distribution as the host PC.

Install packages for compiling the parts and the complete image:

```
sudo apt-get -y install debos git build-essential gcc-aarch64-linux-gnu make bison bc flex \
  libssl-dev device-tree-compiler python3-dev python3-pkg-resources swig fdisk \
  bmap-tools python-is-python3 python3-setuptools python3-pyelftools
```

As debos internally uses kvm virtualization, your user must be a member of the `kvm` group:

```
sudo adduser "$(id -un)" kvm
```

Log out and back for the change to take affect. Then verify that `kvm` is listed in your groups:

```
id -Gn
```

Note: If you are not using Debian distribution on your host PC you need to use podman to build the debos image:

```
sudo apt-get install podman
```

5.2 Get the ATF

Get the Arm Trusted Firmware as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://github.com/rockchip-linux/rkbin
cd rkbin || return
```

(continues on next page)

(continued from previous page)

```
# Tag  linux-5.10-gen-rkr4.1
git checkout "1356c978"
export RKBIN_FOLDER="$PWD"
export BL31="$RKBIN_FOLDER/bin/rk35/rk3588_bl31_v1.38.elf"
export ROCKCHIP_TPL="$RKBIN_FOLDER/bin/rk35/rk3588_ddr_lp4_2112MHz_lp5_2736MHz_v1.11.bin"
# shellcheck disable=SC2103  # we want to export variables, not possible within subshell
cd ..

# Make the baudrate match our U-Boot
sed -i 's/uart baudrate=/uart baudrate=115200/' rkbin/tools/ddrbin_param.txt
rkbin/tools/ddrbin_tool rkbin/tools/ddrbin_param.txt "$ROCKCHIP_TPL"
```

This step should take under 1 minute total.

5.3 Compile U-Boot

Note: Variables `BL31` and `ROCKCHIP_TPL` must be already set as described in Section 5.2 Get the ATF .

Get the source code and compile the U-Boot bootloader as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://git.embedded.cherry.de/jaguar-u-boot.git

(
cd jaguar-u-boot || return

# Load u-boot config
make jaguar-rk3588_defconfig

# Build U-Boot
make -j"$(nproc)"
)

# Make the resulting file available to later steps
export JAGUAR_UBOOT_DIR="$PWD/jaguar-u-boot"
```

This step should take about 1 minute total.

5.4 Compile the Linux kernel

Get the source code and compile the Linux kernel as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://git.embedded.cherry.de/jaguar-linux.git

(
cd jaguar-linux || return

# Compile
make jaguar-rk3588_defconfig
```

(continues on next page)

(continued from previous page)

```
make -j"$(nproc)"
## Make sure there are no modules from older builds, otherwise may pollute rootfs
## if using debos-recipes instructions.
rm --recursive --force overlay/
make -j"$(nproc)" INSTALL_MOD_PATH=overlay modules_install
)

# Make the resulting files available to later steps
export JAGUAR_LINUX_DIR="$PWD/jaguar-linux"
```

The time required for this step heavily depends on your internet connection and CPU power. On a quad-core 2.9GHz machine with an 1Gb/s internet connection, it takes about 20 minutes total.

Warning: It is essential the kernel modules installed on the system are built from the exact same sources as the kernel Image itself or the modules will fail to be detected by the kernel.

Note: One can install new modules without needing to recompile the debos image entirely by running the following command:

```
export IP=10.11.12.13 # set to the IP address of the device
rsync --delete --recursive overlay/lib/modules/ root@"$IP":/lib/modules
```

Update the kernel Image if there was some change made to it so that it will find the new modules upon reboot.

Reboot for the new modules to be loaded.

5.5 Building the debos image

5.5.1 Prepare required components

Note: The variables `JAGUAR_UBOOT_DIR` and `JAGUAR_LINUX_DIR` must be already set as described in Section 5.3 Compile U-Boot and Section 5.4 Compile the Linux kernel, respectively.

Get the source code for the debos recipe and copy the necessary components that were built in the previous steps:

```
# Download the source code
git clone https://git.embedded.cherry.de/debos-recipes.git
cd debos-recipes || return

# Copy Linux binaries into the ``jaguar`` folder
cp "$JAGUAR_LINUX_DIR"/arch/arm64/boot/Image jaguar/overlay/boot/
## Match dtb and dtbo
cp "$JAGUAR_LINUX_DIR"/arch/arm64/boot/dts/rockchip/rk3588-jaguar*.dtb* jaguar/overlay/boot/
rm --recursive --force jaguar/overlay/lib/modules
mkdir --parents jaguar/overlay/lib/modules
cp --archive "$JAGUAR_LINUX_DIR"/overlay/lib/modules/ jaguar/overlay/lib/
## Remove known problematic symlinks as debos would dereference them
rm jaguar/overlay/lib/modules/*/build
rm jaguar/overlay/lib/modules/*/source

# Copy U-Boot binaries into the ``jaguar`` folder
cp "$JAGUAR_UBOOT_DIR"/u-boot-rockchip.bin jaguar/
```

5.5.2 Build a complete image

Depending on your host PC and internet connection, this step should complete in about 5-10 minutes.

The resulting image is a file called `sdcard-jaguar-debos-bookworm.XXX.YYY.img` and, for convenience, the symlink `sdcard-jaguar-debos-bookworm.img` that always points to the latest version.

Debian 12 Bookworm

```
# Build the image using debos
build_board=jaguar ./build.sh

# Or: Build the image using podman (for host PCs not using Debian)
# build_board=jaguar debos_host=podman ./build.sh

# Make the resulting image available to later steps
export SDCARD_IMG="$PWD/sdcard-jaguar-debos-bookworm.img"
```

Note: When running inside a virtual machine that does not support nesting, you may get an error like this:

```
open /dev/kvm: no such file or directory
```

In this case, prepend `debos_host=chroot` to the `build.sh` command, resulting in:

```
debos_host=chroot build_board=jaguar ./build.sh
```

The `debos_host=chroot` mode uses `sudo` internally as it requires root permissions.

6 Building a Yocto image

The Yocto Project is an open-source project that helps building Linux-based distributions, mainly for embeded-ded products. CHERRY provides a minimal BSP layer to allow building Yocto images for the company's modules. An extended layer is also provided for a less bare experience, see instructions in Section 6.3 Extended meta layer. Upon request, access can be given to a more featureful "demonstration" layer which provides hardware and software validation scripts as well as demo applications.

This user guide does not aim at getting the user familiar with development with the Yocto Project but rather help them setup their build environment to create a basic Yocto image that can be used on one of CHERRY modules.

The Yocto project provides an open source Linux build framework, which allows to create customized build environments for embedded systems.

Yocto consists of the following parts:

- The Yocto Project tools,
- Reference Linux distribution (Poky),
- Build system (co-maintained with OpenEmbedded),

There exists extensive documentation for the Yocto Project and BitBake.

The Yocto Project releases a new version twice a year and some versions are maintained for a longer time when marked as LTS (Long-Term Support). Such is the case of Kirkstone (4.0), supported until at least April 2024. CHERRY highly recommend to use LTS versions and update to a newer version once its support has reached end-of-life, to benefit from bug fixes, security fixes, miscellaneous improvements and additional features.

6.1 Prerequisites

While the Yocto Project supports many different build systems, CHERRY currently only tests building on Debian 11 (Bullseye).

The required packages for Debian are listed in the documentation and can be installed with the following command:

```
sudo apt-get install -y --no-install-recommends gawk wget git diffstat unzip \
 texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-venv \
 python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 \
 libegl1-mesa libsdl1.2-dev xterm python3-subunit mesa-common-dev zstd \
 liblz4-tool file
```

6.2 BSP meta layer

The Yocto Project BSP meta layer can be found at https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-bsp.git/.

It contains the minimal configuration and recipe append files (bbappend) necessary to build a minimal working image. It is meant to be a base upon which to build and thus many tools are purposefully missing.

6.2.1 Initial setup

Clone the BSP meta layer and its dependencies from a new directory called `yocto`:

```
mkdir yocto
cd yocto || return
git clone https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-bsp.git \
        -b kirkstone
git clone https://git.yoctoproject.org/poky -b kirkstone-4.0.10
git clone https://git.yoctoproject.org/meta-arm -b yocto-4.0.2
git clone https://git.yoctoproject.org/meta-rockchip -b kirkstone
git clone https://git.openembedded.org/meta-openembedded -b kirkstone
```

The following directory layout should be observed:

```
$ tree -L 1 yocto/
yocto
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
└── poky
```

Note: It is essential that the Yocto layers are checked out on a branch that supports the same release as the others, otherwise there may be some unexpected issues. With the aforementioned instructions, the layers have been checked out to a branch supporting the Yocto Project Kirkstone (4.0) release.

One can check if a branch supports a release by looking into `conf/layer.conf` and look for the `LAYERSERIES_COMPAT_*` variable. All layers should have the same one in common, here "kirkstone".

6.2.2 Initializing build environment

Once the layers have been properly cloned in their appropriate branch, the build environment needs to be initialized. This can be done by running the following command:

```
# shellcheck disable=SC3046,SC1091
source poky/oe-init-build-env build
```

This will initialize the build environment by making the `bitbake` build tool available in the current shell and creating a `build` directory where temporary and final build artifacts will be stored.

The following directory layout should be observed:

```
$ tree -L 1 yocto/
yocto
├── build
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
└── poky
```

The first time the command is run, it'll create a new build directory called `build` and add the appropriate configuration files. On the later runs, if the directory still exists, the command will only configure the terminal environment and not change anything in the build directory. This makes it perfectly safe to run the command multiple times, from different terminals for example.

Note: Once the current terminal is closed or a new one is opened, this command should be re-executed to be able to interact again with the Yocto Project tools.

The Yocto Project then needs to be configured to include layers to find new recipes or configuration files, which is essential to build new pieces of software or compile for a specific hardware target system.

This can be done with the `bitbake-layers` tool:

```
bitbake-layers add-layer ../meta-arm/meta-arm-toolchain
bitbake-layers add-layer ../meta-arm/meta-arm
bitbake-layers add-layer ../meta-rockchip
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-theobroma-systems-bsp
```

6.2.3 Building a minimal image

To build a bootable artifact, BitBake will be called with the specified machine and target image:

```
MACHINE="jaguar" bitbake core-image-minimal
```

Note: Technically speaking, the `MACHINE` variable could be set in `build/conf/local.conf` file once and for all. If possible, CHERRY recommends passing the variable explicitly in the command directly as this makes it more visible to the user and also allows to easily build for multiple machines without modifying a file in-between.

The build process can take several hours depending on the capabilities of the build machine and the user's Internet connection.

Note: If the Bitbake process needs to be stopped for any reason, a `SIGINT` (`Ctrl` + `c`) signal can be sent **once**. Bitbake will gracefully close down upon reception of this signal. This graceful shutdown can take a lot of time depending on the tasks that are currently being executed. It is **highly** recommended to not send this signal more than once, failing to do so may hinder next Bitbake commands.

The artifacts can be found after some time in `build/tmp/deploy/images/jaguar/` directory. A flashable image is one whose extension is `.wic`, e.g. `core-image-minimal-jaguar-20221021134027.rootfs.wic`.

Make the resulting image available for later steps:

```
export SDCARD_IMG="$PWD/build/tmp/deploy/images/jaguar/core-image-minimal-jaguar.wic"
```

6.2.4 Building with kas

kas is a setup tool for Bitbake-based projects, such as the Yocto Project, which aims to replace the commands listed above for a simpler, more automated, setup and creation of images.

CHERRY provides a kas configuration file `kas-theobroma.yml` in the BSP meta layer for convenience.

kas can be installed on the build machine with the following command:

```
sudo apt-get install -y --no-install-recommends kas
```

Note: It is also available as a Python package and installable with:

```
python3 -m venv venv
# shellcheck disable=SC3046,SC1091
source venv/bin/activate
python3 -m pip install kas==4.0
```

The Section 6.2.1 Initial setup and Section 6.2.2 Initializing build environment can then be replaced by the following two commands:

```
mkdir yocto
cd yocto || return
git clone https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-bsp.git \
        -b kirkstone
kas checkout meta-theobroma-systems-bsp/kas-theobroma.yml
```

The Section 6.2.3 Building a minimal image can now be replaced with:

```
KAS_MACHINE="jaguar" kas build meta-theobroma-systems-bsp/kas-theobroma.yml
```

Note: kas is also available in an OCI container form on GitHub container registry.

It is still recommended to install kas through pip but then use its kas-container wrapper script to start the container properly. E.g. to replace the last command to build an image with kas one can call this instead:

```
python3 -m venv venv
# shellcheck disable=SC3046,SC1091
source venv/bin/activate
python3 -m pip install kas==4.0
export KAS_IMAGE_VERSION="4.0"
export KAS_MACHINE="jaguar"
kas-container build meta-theobroma-systems-bsp/kas-theobroma.yml
```

6.3 Extended meta layer

The Yocto Project extended layer can be found at https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-extended.git/.

In addition to the minimal features, this layer includes the network manager, and many more features will be added soon.

6.3.1 Initial setup

Clone the Extended layer and its dependencies from a new directory called yocto:

```
mkdir yocto
cd yocto || return
git clone https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-extended.git \
        -b kirkstone
git clone https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-bsp.git \
        -b kirkstone
git clone https://git.yoctoproject.org/poky -b kirkstone-4.0.10
git clone https://git.yoctoproject.org/meta-arm -b yocto-4.0.2
git clone https://git.yoctoproject.org/meta-rockchip -b kirkstone
git clone https://git.openembedded.org/meta-openembedded -b kirkstone
```

The following directory layout should be observed:

```
$ tree -L 1 yocto/
yocto
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
├── meta-theobroma-systems-extended
└── poky
```

Note: It is essential that the Yocto layers are checked out on a branch that supports the same release as the others, otherwise there may be some unexpected issues. With the aforementioned instructions, the layers have been checked out to a branch supporting the Yocto Project Kirkstone (4.0) release.

One can check if a branch supports a release by looking into `conf/layer.conf` and look for the `LAYERSERIES_COMPAT_*` variable. All layers should have the same one in common, here "kirkstone".

6.3.2 Initializing build environment

Once the layers have been properly cloned in their appropriate branch, the build environment needs to be initialized. This can be done by running the following command:

```
# shellcheck disable=SC3046,SC1091
source poky/oe-init-build-env build
```

This will initialize the build environment by making the `bitbake` build tool available in the current shell and creating a `build` directory where temporary and final build artifacts will be stored.

The following directory layout should be observed:

```
$ tree -L 1 yocto/
yocto
├── build
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
├── meta-theobroma-systems-extended
└── poky
```

The first time the command is run, it'll create a new build directory called `build` and add the appropriate configuration files. On the later runs, if the directory still exists, the command will only configure the terminal environment and not change anything in the build directory. This makes it perfectly safe to run the command multiple times, from different terminals for example.

Note: Once the current terminal is closed or a new one is opened, this command should be re-executed to be able to interact again with the Yocto Project tools.

The Yocto Project then needs to be configured to include layers to find new recipes or configuration files, which is essential to build new pieces of software or compile for a specific hardware target system.

This can be done with the `bitbake-layers` tool:

```
bitbake-layers add-layer ../meta-arm/meta-arm-toolchain
bitbake-layers add-layer ../meta-arm/meta-arm
bitbake-layers add-layer ../meta-rockchip
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-openembedded/meta-networking
bitbake-layers add-layer ../meta-theobroma-systems-bsp
bitbake-layers add-layer ../meta-theobroma-systems-extended
```

6.3.3 Building an image

To build a bootable artifact, BitBake will be called with the specified machine and target image:

```
MACHINE="jaguar" bitbake theobroma-extended-image
```

Note: Technically speaking, the `MACHINE` variable could be set in `build/conf/local.conf` file once and for all. If possible, CHERRY recommends passing the variable explicitly in the command directly as this makes it more visible to the user and also allows to easily build for multiple machines without modifying a file in-between.

The build process can take several hours depending on the capabilities of the build machine and the user's Internet connection.

Note: If the Bitbake process needs to be stopped for any reason, a `SIGINT` (`Ctrl` + `c`) signal can be sent **once**. Bitbake will gracefully close down upon reception of this signal. This graceful shutdown can take a lot of time depending on the tasks that are currently being executed. It is **highly** recommended to not send this signal more than once, failing to do so may hinder next Bitbake commands.

The artifacts can be found after some time in `build/tmp/deploy/images/jaguar/` directory. A flashable image is one whose extension is `.wic`, e.g. `theobroma-extended-image-jaguar-20221021134027.rootfs.wic`.

Make the resulting image available for later steps:

```
export SDCARD_IMG="$PWD/build/tmp/deploy/images/jaguar/theobroma-extended-image-jaguar.wic"
```

6.3.4 Building with kas

kas is a setup tool for Bitbake-based projects, such as the Yocto Project, which aims to replace the commands listed above for a simpler, more automated, setup and creation of images.

CHERRY provides a `kas` configuration file `kas-theobroma.yml` in the BSP meta layer for convenience.

`kas` can be installed on the build machine with the following command:

```
sudo apt-get install -y --no-install-recommends kas
```

Note: It is also available as a Python package and installable with:

```
python3 -m venv venv
# shellcheck disable=SC3046,SC1091
source venv/bin/activate
python3 -m pip install kas==4.0
```

The Section 6.3.1 Initial setup and Section 6.3.2 Initializing build environment can then be replaced by the following two commands:

```
mkdir yocto
cd yocto || return
git clone https://git.embedded.cherry.de/yocto-layers/meta-theobroma-systems-extended.git \
        -b kirkstone
kas checkout meta-theobroma-systems-extended/kas-theobroma.yml
```

The Section 6.3.3 Building an image can now be replaced with:

```
KAS_MACHINE="jaguar" kas build meta-theobroma-systems-extended/kas-theobroma.yml
```

Note: kas is also available in an OCI container form on GitHub container registry.

It is still recommended to install `kas` through `pip` but then use its `kas-container` wrapper script to start the container properly. E.g. to replace the last command to build an image with `kas` one can call this instead:

```
python3 -m venv venv
# shellcheck disable=SC3046,SC1091
source venv/bin/activate
python3 -m pip install kas==4.0
export KAS_IMAGE_VERSION="4.0"
export KAS_MACHINE="jaguar"
kas-container build meta-theobroma-systems-extended/kas-theobroma.yml
```

---

7 Deploy a disk image

This chapter describes how to write a disk image of the Debian 12 bookworm variant as generated in the previous chapter.

Note: The variable `SDCARD_IMG` must be already set as described in respective chapter.

Warning: Avoid having the disk image on both the SD Card and the internal eMMC of the module.

As the Linux kernel on the module uses `PARTLABEL` and `PARTUUID` to identify partitions to mount, it will be unpredictable whether the SD Card or the internal eMMC is used.

7.1 Deploy on SD Card

Insert an SD card into the host PC and check `dmesg -w` to find out the device name that was used.

To flash the image on an SD card, bmaptool can be used, it is both faster and safer than a traditional `dd`. For that, the `.bmap` companion file, automatically built by the Yocto Project or `build.sh debos-recipes` wrapper script, should be in the same directory as the `SDCARD_IMG` artifact.

Then run the following command, with `/dev/sdX` replaced by the block device representing the user's SD card:

```
sudo bmaptool copy "$SDCARD_IMG" /dev/sdX
```

7.2 Deploy on internal eMMC

7.2.1 Compile rkdeveloptool

To write the image directly onto the on-board eMMC, the flashing tool rkdeveloptool is used, and it must be compiled on the host PC:

```
# Install compile dependencies
sudo apt-get -y install git libudev-dev libusb-1.0-0-dev dh-autoreconf pkg-config \
        build-essential

# Download rkdeveloptool source code
git clone https://github.com/rockchip-linux/rkdeveloptool.git
cd rkdeveloptool || return

# Compile rkdeveloptool
autoreconf -i
CPPFLAGS=-Wno-format-truncation ./configure
make

# Download miniloaders used for flashing
git clone https://github.com/rockchip-linux/rkbin.git tools/rk_tools

# Build miniloader binaries
(
cd tools/rk_tools/ || return
# Tag linux-5.10-gen-rkr4.1
git checkout "1356c978"
./tools/boot_merger RKBOOT/RK3588MINIALL.ini
```

(continues on next page)

(continued from previous page)

```
)

# Make the resulting files available to later steps
export RKDEVELOPTOOL_DIR="$PWD/tools/rk_tools/"
```

This step should take about 1 minute total.

7.2.2 Enter USB flashing mode

Connect a USB-C cable between the `USB P11 Download port` (see Fig. 2.4 Buttons and Download USB Type-C port) and a USB port of your host PC.

Make sure there is no SD card inserted into the board.

Push the `BOOT SW` button (see Fig. 2.4 Buttons and Download USB Type-C port).

Push the `Reset` button and release the `BOOT SW` button.

The `lsusb` command on your host PC should return the following:

```
$ lsusb -d 2207:350b
Bus xxx Device 0xx: ID 2207:350b Fuzhou Rockchip Electronics Company
```

7.2.3 Flash the eMMC

To write the image file path stored in the variable `SDCARD_IMG` to the on-board eMMC, run:

```
cd "$RKDEVELOPTOOL_DIR" || return
sudo ./rkdeveloptool db tools/rk_tools/rk3588_spl_loader_v* && sleep 1
sudo ./rkdeveloptool wl 0 "$SDCARD_IMG"
sudo ./rkdeveloptool rd
```

This step should take about 1 minute for the Debian image.

8 Companion controller features

This chapter describes the companion controller (Mule ATtiny) features.

8.1 How to flash Mule-ATtiny

The ATtiny can be flashed through the UPDI lines, from the running system on Jaguar (No additional hardware required). For convenience, `mule-attiny.sh` tool is available for flashing the Mule ATtiny microcontroller. The tool is available here: https://git.embedded.cherry.de/som-tools.git/tree/mule-attiny.

8.1.1 Requirements

- avrdude tool (minimum v7.1)

8.1.2 Install avrdude

```
apt-get install avrdude
```

8.1.3 Flashing Mule ATtiny

```
MULE_FIRMWARE="/path/to/mule-ATtiny816-xxxxxxx.hex"
./mule-attiny.sh --flash "$MULE_FIRMWARE"
```

Note: The above commands should be run with root privileges.

Note: It is highly recommended that one reboots the main SoC interacting with the companion microcontroller after flashing to make sure device drivers are properly initialized.

9 Serial Number

Each Jaguar has a unique serial number that can be read by software.

In U-Boot, the serial number is contained in the environment variable `serial#`. You can print it using the command:

```
printenv serial#
```

Under Linux, it is represented by a simple text file in `/sys`:

```
cat /sys/firmware/devicetree/base/serial-number
```

The serial number is fixed in hardware (derived from the SoC CPU ID) and cannot be modified.

10 Contact

Cherry Embedded Solutions GmbH
Seestadtstraße 27
1220 Vienna
Austria

Inquiries: sales-es@cherry.de
Technical Support: support-es@cherry.de

11 Revision History

Date	Revision	Major changes
Nov 27, 2023	v1.0.0	First release
Dec 18, 2023	v1.0.1	recalled version
Feb 14, 2024	v1.0.2	Updated flashing instructions for companion microcontroller to use wrapper shell script Replaced dd flashing instructions with bmaptool Updated pip instructions for Bookworm Added yocto directory creation Updated kas container instructions Rephrased layer version requirement Update instructions for U-Boot v2024.01
Apr 02, 2024	v1.1.0	Add instructions on how to use kernel modules in debos Changed shell code snippets to pass shellcheck Theobroma Systems is now CHERRY Embedded Solutions