EAS2A01 Datasheet

- Cal. capability: 9.2K DMIPS + 1800 GMAC
- Data transmission interfaces:
  - 1 Gigabit Ethernet interface
  - 1 HDMI output interface
  - 8 GMSL interfaces
  - 2 CAN interfaces
  - 2 RS-232 interfaces
- Operating voltage: DC 9-16V
- Operation memory: 2GB
- Storage memory: 16GB
- Operating temperature: -40 to 85 ℃
- Humidity: 0 - 95%, no condensation
- Storage temperature: -40 to 85 ℃
- Dimensions: 284×143×42mm
- Weight: less than 1500g

Revision History

Time	Version	Detail	Reviser
Apr. 29, 2019	V1.0	First version	David Wang
Sep. 20, 2019	V1.1	3.3.3 updated	David Wang
Feb. 11, 2020	V1.2	EcoCoder-AV, address updated	David Wang
Feb. 20, 2020	V1.3	EcoCoder-AI update	David Wang
May 11, 2020	V1.4	Contact info update	Zack Li

Contact us

Web: http://www.ecotrons.com

Email: info@ecotrons.com

ev-support@ecotrons.com

Address: 13115 Barton Rd, Ste H

Whittier, CA 90605 USA

Telephone: +1 562-758-3039

+1 562-713-1105

Content

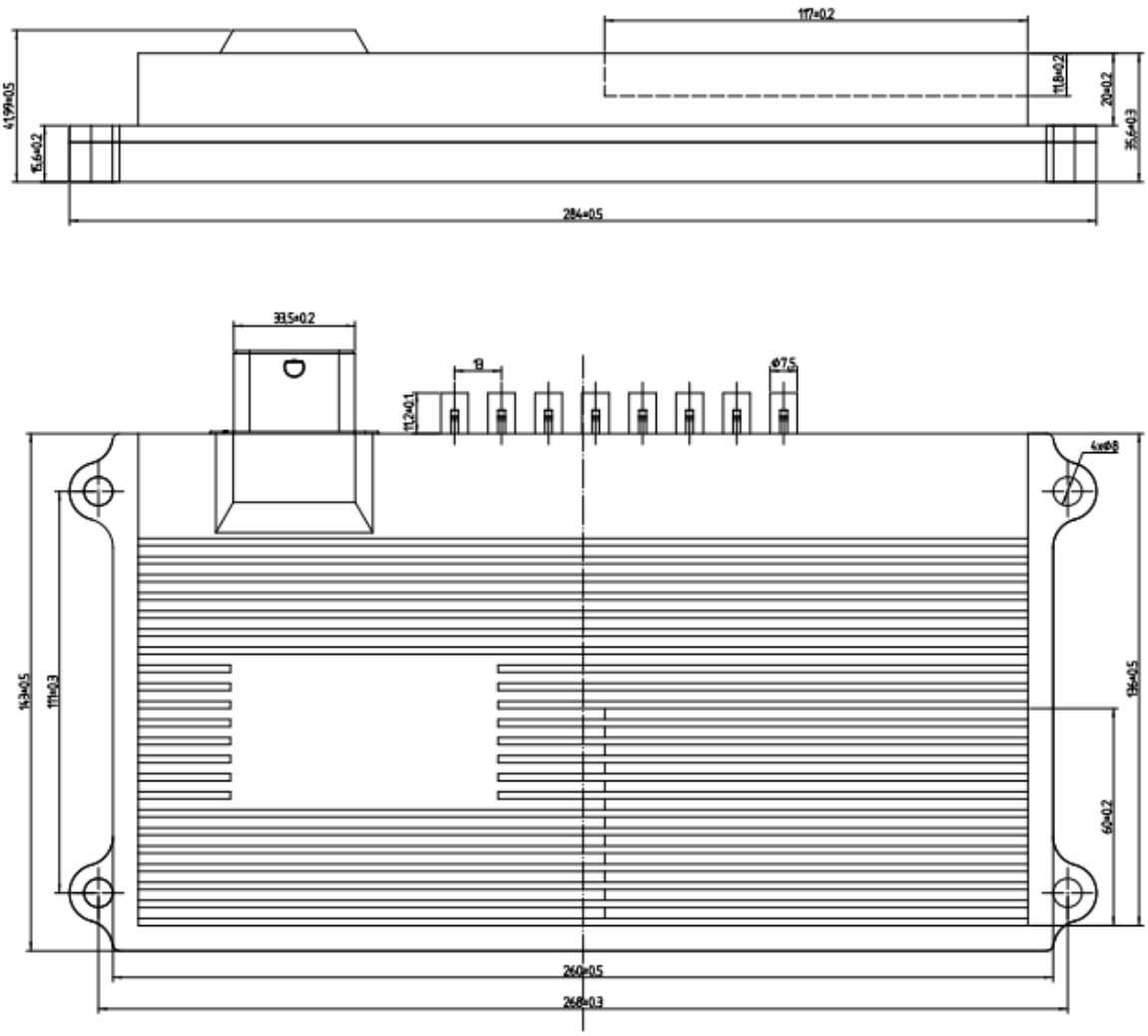
Chapter 1 Summery

EAS2A01 is a central computing platform developed by Ecotrons LLC for autopilot systems with car-level chips. The main computing chip of EAS2A01 is NXP's S32V234 which is designed specifically for the data processing of the autonomous system. S32V234 contains a parallel computing module that enables hardware-level data operation acceleration. The software system of EAS2A01 is also customized for autonomous driving. It contains a real-time optimized Linux kernel, a high-performance runtime framework ROS, EcoCyber and so on. Moreover, we provide development tools on PC, enabling developers to safely, conveniently and efficiently build an automatic driving system that meets the L3 level autonomous driving system.

Chapter 2 Mechanics

2.1 Dimensions

The size of housing is 284 × 143 × 42 mm (connectors excluded). The color is silver, and the material is aluminum.



2.2 Connector

EAS2A01 uses the 32-pin water-proof car-level connector from Tyco shown as below. The interface of the camera is FAKRA.

No.	Name	Type	Supplier
1	PCB needle	64334-0100	TE
2	32P sheath	64319-3211	TE
4	Terminal	64323-1029	TE
5	Terminal	64323-1039	TE
6	32P back	64319-1201	TE

Chapter 3 Quick Start

3.1 Preparation

Before using this device, please prepare the following items:

- Stable power supply, 12V DC / 2 A minimum
- USB to RS-232 adapter
- Network cable with RJ-45 connector
- Wireless router
- Computer with ubuntu 14.04 and minicom installed

3.2 Basic Knowledge

If you are a Linux beginner, it's helpful to learn some quick tutorials about Linux command line tools. Please click this: http://www.ee.surrey.ac.uk/Teaching/Unix/

3.3 Using the device

3.3.1 Connect all the components

Connect the positive and negative terminals of the device to DC power supply, and connect the RS232-1 of the device to the computer through USB to RS-232 adapter, ensuring that the computer can use the serial device normally. Connect the device to the router through the network cable, and connect the computer to the router via WIFI, so that the computer and the device are in the same local area network.

3.3.2 Configure

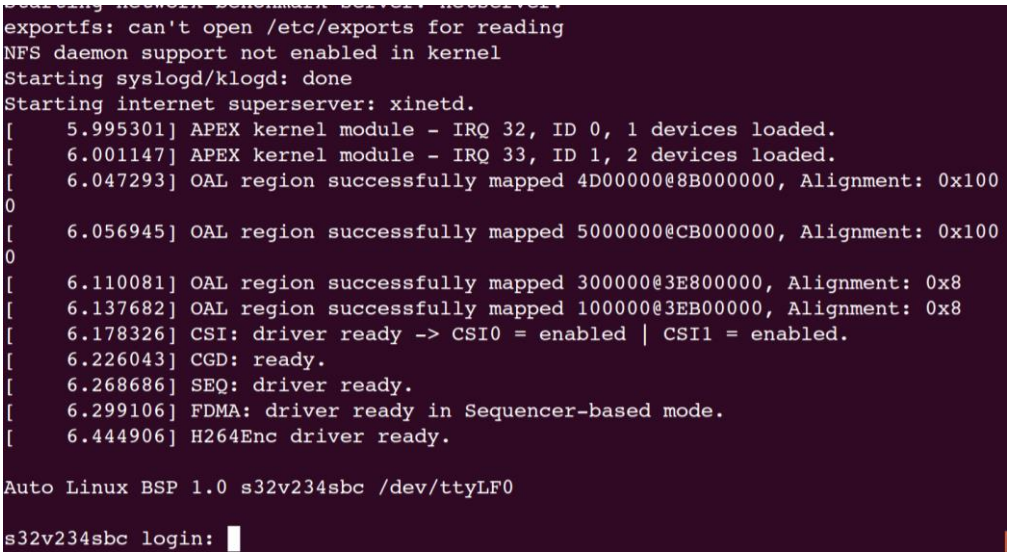
Serial Port Configuration: baud rate = 115200, data bits = 8, parity bit = no, stop bit = 1.

Router Configuration: DHCP service = enabled.

3.3.3 Boot

Turn on the device's KeyOn switch and start the device power. The device shall start U-Boot and then run the Linux system. If you see the following information in the serial terminal window,

the system starts normally. The username is “root” and there is no password. To learn more about the software information of the system, please read the software description of this document.



3.3.4 Test

Go to /home/root/testfile, you can see the test instructions and test programs for various interfaces in the directory.

```
# cd /home/root/testfile
```

Test HDMI interface

Connect the display to the HDMI interface of the device through HDMI cable. In the serial terminal, go to hdmi_test directory and go through hdmi_test.txt. Work as the file says, you will see the image information on the display.

```
# cd /home/root/testfile/hdmi_test/
```

```
# cat hdmi_test.txt
```

Test FAKRA interface

To test FAKRA interface, you need to prepare a camera which is compatible with the software. If you need help with camera selection, please contact us. Here we are going to take camera

module MAXCAMOV10640 as an example. The sensor of the camera module is OV10640, and the serializer is MAX96705. Connect the camera module to FAKRA interface, in the serial terminal, go to directory cam_test, and execute isp_ov10640_quad_slotX.elf (X in isp_ov10640_quad_slotX.elf stands for the No. of FAKRA interface), and you can see the image information on the display. If you don't see any image in the display, you can try pressing 'X', and then keep pressing '+' or '-'.

```
# cd /home/root/testfile/cam_test/

# ./isp_ov10640_quad_slot3.elf
```

Test CAN interface

Connect CANA and CANB of the device to the same CAN bus through a CAN cable, in the serial terminal, go to can_test directory, view at can_test.txt and work as the instruction says. If CANB can receive the data frame sent by CANA, CANA can also receive the data frame sent by CANB, both CAN channels can work normally.

```
# cd /home/root/testfile/can_test/

# cat can_test.txt
```

Test network interface

Type ifconfig in the serial terminal and remember the IP address of the device. Type ifconfig in the computer terminal and remember the IP address of the computer. After you confirm they are in the same local area network, you can use ping command to test the connection state of the internet. For example, if the computer's IP is 192.168.1.112, the command should look like this:

```
# ifconfig

# ping 192.168.1.112
```

Test serial port

If the serial terminal can interact with the device, the RS232-1 hardware is in good state and we only need to test RS232-2. Disconnect the USB to RS232 adapter from RS232-1 of the device, instead, connect with RS232-2 interface of the device. After confirming that there is no problem with the network connection between the device and the computer, you can enter the telnet command in the computer terminal and replace <> to log in to the device through the network. Go to the uart_test directory and check the uart_test.txt file. Follow the instructions, you can the test PC sending to the device and the device sending to PC.

```
$ telnet  <target_ip>
```

```
# cd /home/root/testfile/uart_test/
```

```
# cat uart_test.txt
```

Test ROS

Work in the serial terminal: go to ros_test directory and view ros_test.txt file. Work as the instruction says, you will be able to see that the most basic publisher **talker** and subscriber **listener** in the ROS system can communicate normally.

```
# cd /home/root/testfile/ros_test/
```

```
# cat ros_test.txt
```

3.3.5 Develop

Develop the application based on the device using the development tool provided in this document, and transfer the compiled executable program to the device through the SCP command. Then you can start the application.

Chapter 4 Hardware

The hardware circuit of the device is designed according to the application requirements of the automatic driving system. The electrical parameters meet the requirements of the vehicle standard. It has various data transmission interfaces, which makes it easy to do multi-sensor fusion of the automatic driving system. The main chip contains a variety of high-performance computing units to adapt to the calculation characteristics of automatic driving sequence calculation and parallel computing.

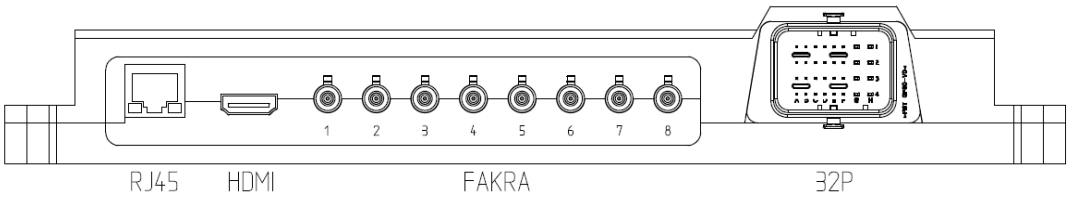
- Operating voltage: DC 9-16V
- Operation memory: 2GB
- Storage memory: 16GB
- Calculation capability: 9.2K DMIPS + 1800 GMAC
- Data transmission interfaces:
    - 1 Gigabit Ethernet interface
    - 1 HDMI output interface
    - 8 GMSL interfaces
    - 2 CAN interfaces
    - 2 RS-232 interfaces

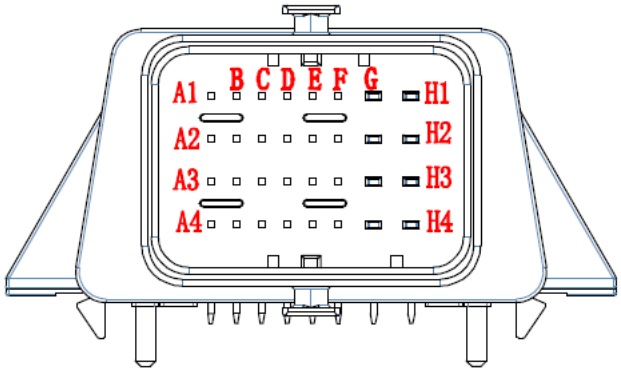
4.1 Specifications

Item	Detail
Operating voltage	DC 9-16V
SDRAM	2GB
eMMC	16GB
Operating temperature	-40 to 85 ℃
Operating humidity	0 – 95%, no condensation
Storage temperature	-40 to 85 ℃
Dimensions	284×143×42mm
Weight	≤1500g

4.2 Device Ports

The naming rules for the device ports are shown below:





You can look up the port definition in the table below.

Port/Pin	Function	Usage example
RJ45	Gigabit Ethernet	LiDAR
HDMI	High-Definition Multimedia Interface Output	
FAKRA1	Gigabit Multimedia Serial Link (GMSL) Input 1	Camera
FAKRA2	Gigabit Multimedia Serial Link (GMSL) Input 2	Camera
FAKRA3	Gigabit Multimedia Serial Link (GMSL) Input 3	Camera
FAKRA4	Gigabit Multimedia Serial Link (GMSL) Input 4	Camera
FAKRA5	Gigabit Multimedia Serial Link (GMSL) Input 5	Camera
FAKRA6	Gigabit Multimedia Serial Link (GMSL) Input 6	Camera
FAKRA7	Gigabit Multimedia Serial Link (GMSL) Input 7	Camera
FAKRA8	Gigabit Multimedia Serial Link (GMSL) Input 8	Camera
32P-H1	Power Positive	DC 9-16V
32P-H2	Power Positive	DC 9-16V
32P-H3	Power Positive	DC 9-16V

32P-H4	Power Positive	DC 9-16V
32P-G1	Power GND	
32P-G2	Power GND	
32P-G3	Power GND	
32P-G4	Power GND	
32P-F2	KeyOn Switch 1	
32P-E2	KeyOn Switch 2	
32P-C4	CANA H	Millimeter wave radar
32P-D4	CANA L	
32P-A4	CANB H	Vehicle CAN
32P-B4	CANB L	
32P-A3	CANA Shield Ground	
32P-A2	CANB Shield Ground	
32P-B3	RS232-1 Receiver	Debug
32P-C3	RS232-1 Transmitter	
32P-C2	RS232-2 Receiver	IMU
32P-D2	RS232-2 Transmitter	
32P-C3	Signal GND	
32P-D3	Signal GND	
32P-E3	Signal GND	
32P-E4	Signal GND	
32P-F3	Signal GND	
32P-F4	Signal GND	

Notes:

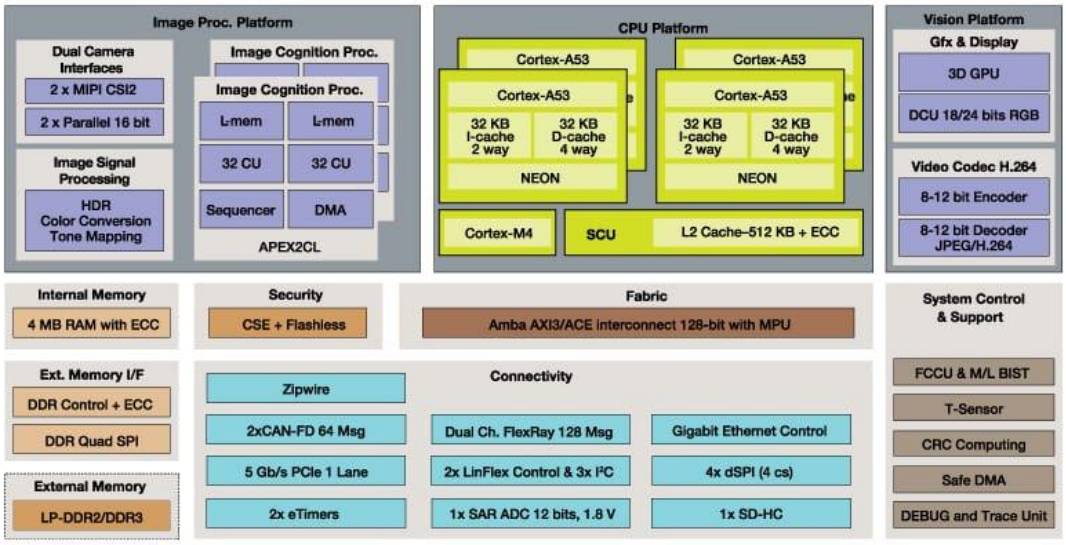
The output voltage of the FAKRA interface is hardware configurable. There are two voltage levels to choose from: DC 5V and power supply voltage. Please refer to the EAS2A01 Product Test Report for specific output voltage or contact the equipment supplier.

Default terminal resistance of CAN interface is 120Ω.

4.3 System Main Chip

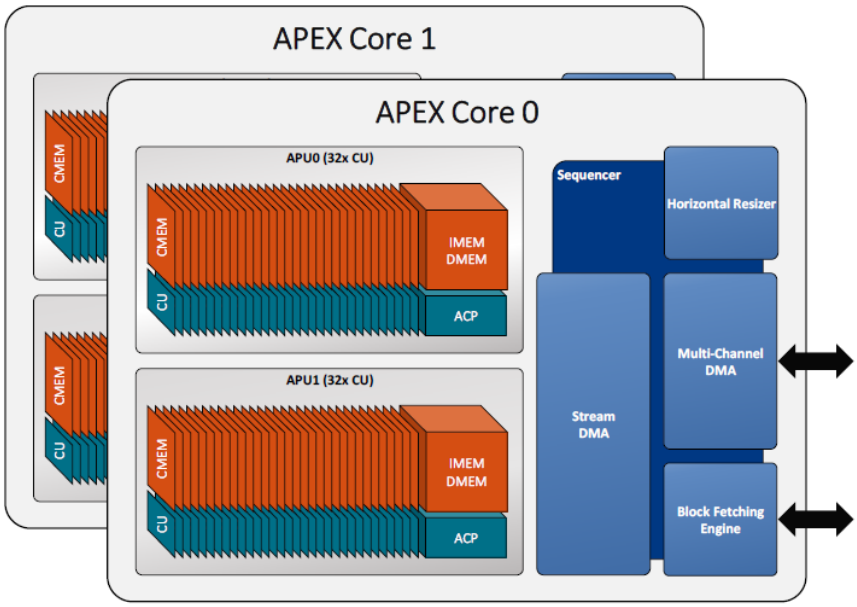
The main chip of EAS2A01 is NXP S32V234, which is NXP's second-generation vision processor designed to support computationally intensive applications such as image processing. It is designed for automotive-grade reliability, functional safety and safety measures. It supports ISO 26262 functional safety, and is suitable for ASIL-C, IEC 61508 and DO 178 applications. It can be used for ADAS, NCAP front-view cameras, foreign object detection and recognition, surround vision, machine learning and sensor fusion applications. The processor's computing unit consists of a quad-core Arm® Cortex®-A53 application processor, a single-core ARM Cortex-M4 real-time processing unit, a dual-core APEX-2 vision accelerator, an image processing unit ISP and a 3D GPU. Applied to autonomous driving, its specific calculation tasks can be assigned as follows:

- ARM A53: task scheduling, hardware management
- APEX: parallel computing, such as image recognition, neural networks, etc.
- ISP: image processing, color conversion, tone mapping, etc.
- GPU: display functions, such as image rendering or overlay generation



The APEX core is a programmable, high-performance, energy-efficient vision accelerator core. It is a massively parallel hybrid processor that is ideal for processing large amounts of data. Each APEX core contains two array processor units (APUs), an advanced direct memory access (DMA) engine and other hardware blocks. The APU is a scalar vector hybrid processor with 32 16-bit

computing units (CUs), whose local dedicated computation memory (CMEM) is for vectors and tightly coupled to the 32-bit scalar RISC processor. Please refer to S32V234 Manual for more information.



The APEX core provides flexible configuration at runtime. By default, APEX is configured as a single APU with 64-CU. When using the APEX Core Framework library, you can change the configuration at runtime and have two separate APUs, each with 32 CUs. For information on how to use APEX to program, please download Vision SDK Software and refer to the documentation, or you can use the S32 Design Studio for Vision IDE.

We compare the computational efficiency of Arm® Cortex®-A53 and APEX with the demo apex_add.elf in the VSDK package provided by NXP. The Vision Software Development Kit (VSDK) provides a comprehensive development environment for the S32V234 vision processor family. The VSDK can be used to develop applications for the Arm® Cortex®-A53. With the graphical tools in the S32 Design Studio for Vision IDE, you can also develop computationally intensive image processing application based on Image Signal Processing (ISP) and Parallel Computing (APEX). At the same time, NXP provides the APEXCV library, which uses APEX to implement basic operations in OpenCV, which is convenient for application developers to port

applications based on CPU version OpenCV to APEX for operation acceleration. APEX supports application developers flexible programming experience based on APEX.

Test project: Perform matrix addition of 2048x1024

Test process: matrix addition operation is performed on ARM, single APEX acceleration unit, and two APEX acceleration units respectively, and the time spent by different operation units is measured at the same time, and broadcasted through the console.

Test result:

Round	ARM (Unit: ms)	APEX Single (Unit: ms)	Acceleration Factor	APEX Dual (Unit: ms)	Acceleration Factor
1	65.924599	2.192400	30.069604	1.673000	39.405020
2	65.961395	2.069500	31.873107	1.608200	41.015667
3	65.849403	2.066400	31.866726	1.607300	40.968956
4	65.765999	2.072700	31.729628	1.602900	41.029384
5	66.085007	2.073000	31.878923	1.610700	41.028750
Average	65.917281	2.094800	31.483597	1.620420	40.689555

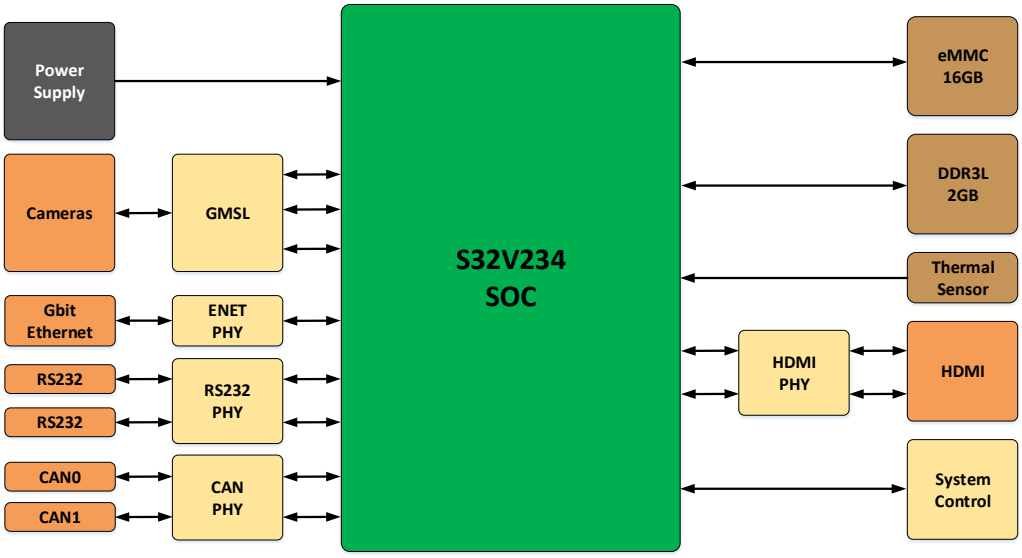
Test conclusion:

A single APEX acceleration unit can accelerate about 30 times.

Two APEX acceleration units can accelerate about 40 times.

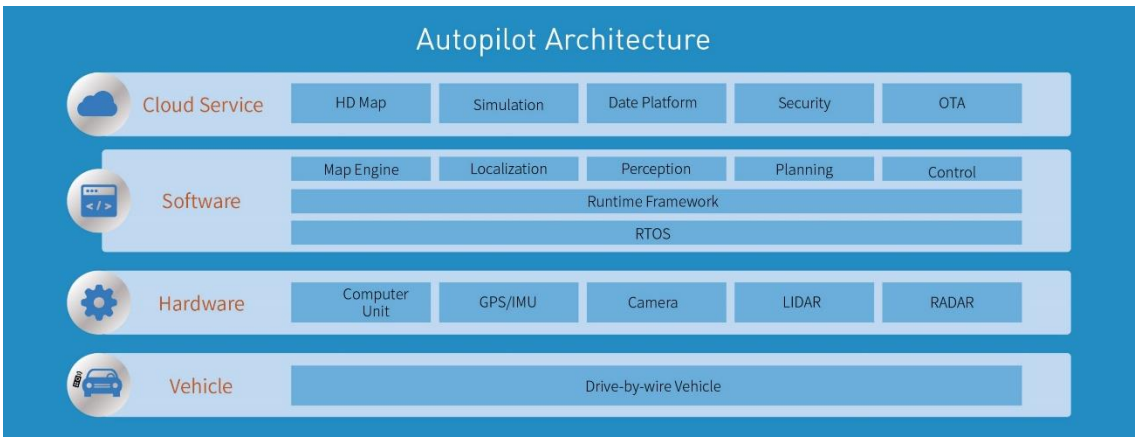
4.4 Circuit Structure

The internal circuit structure is shown below:

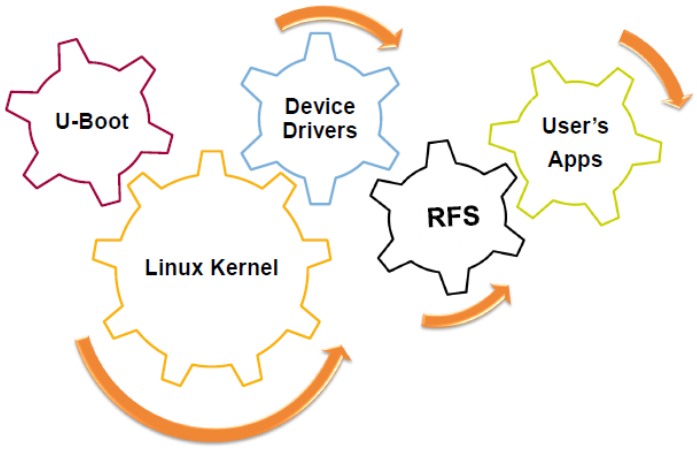


Chapter 5 Software

The software system of EAS2A01 is customized for the autonomous driving system. The following figure is a typical block diagram of the autonomous driving system. The software system of EAS2A01 consists of RTOS, Runtime Framework and so on. The RTOS is a real-time optimized Linux operating system. The Runtime Framework has two software frameworks to choose from: one is the Indigo version of ROS (Robot Operating System), and the other is EcoCyber based on Apollo Cyber RT developed by Ecotrons.



We will introduce U-Boot, Linux kernel, root file system (RFS), ROS and EcoCyber to you. Among them, U-Boot, Linux kernel, and root file system (RFS) constitute the Linux operating system, as shown in the following figure.

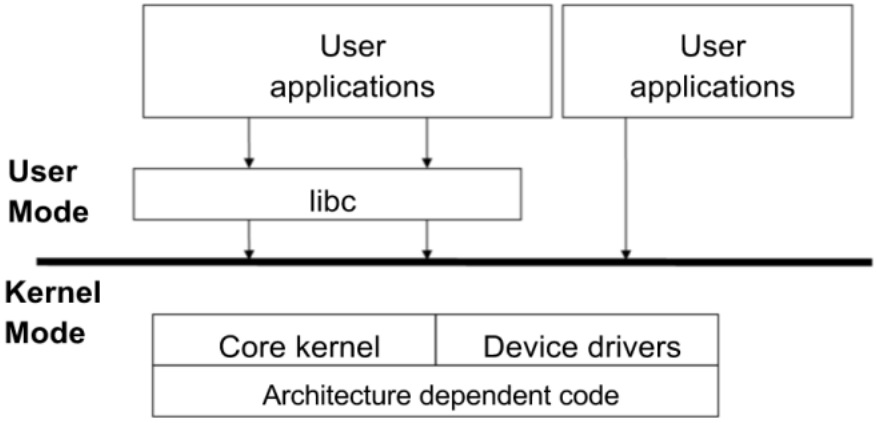


5.1 U-Boot

U-Boot is a general-purpose bootloader widely used in the embedded field. U-Boot can work in boot load mode and download mode. The boot load mode is the normal working mode of the bootloader. In this mode, the bootloader automatically loads the operating system image from the FLASH into the RAM and boots it to start running. The download mode is that the bootloader downloads the kernel image and the root file system image from the PC to the FLASH of the target board through some communication means. Users can use the command interface provided by the Bootloader to complete the operation they want. For details, please refer to the Wiki area for cooperating on U-Boot development.

5.2 Linux Kernel

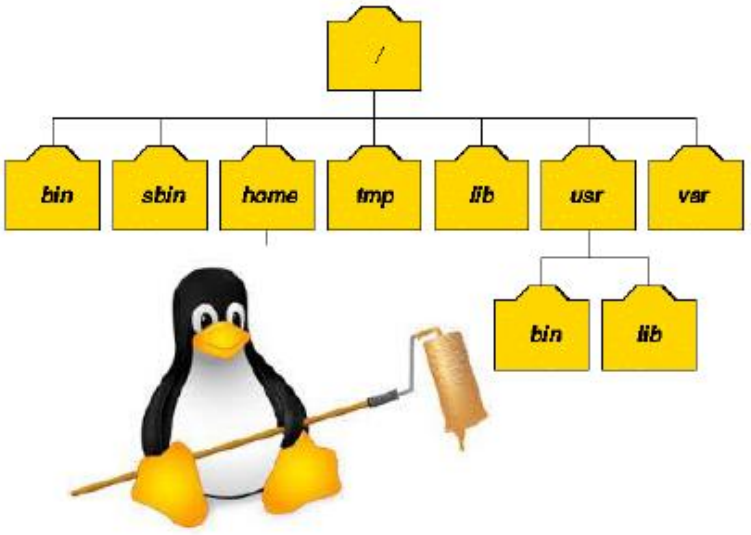
The Linux kernel is an open source computer operating system kernel. It is a Unix-like operating system written in C language and conforms to the POSIX standard. It mainly implements process management, memory management, device management, virtual file system, etc., to achieve isolation between kernel space and user space, as shown in the following figure. Please refer to https://www.kernel.org/ for details.



5.3 Root File System

The root file system is the first file system mounted when the kernel is booted. The operating system loads some of the basic initialization scripts or services and other executable programs into memory after the root file system is mounted. The Linux root file system of this device conforms to the specifications of the Linux system. Generally, there are directories: dev, proc,

sbin, bin, usr, tmp, lib, home, etc., as shown below. For the standard directory structure of Linux file system, please refer to Filesystem Hierarchy Standard.



By default, the file system contains some basic software libraries, including: NXP-VSDK, boost, libc, Python2.7, Python3.5, OpenCV, PCL, Perl and so on. Application developers can develop applications based on these software libraries and add software libraries as needed. For details, refer to the EcoSDK-S2 Instruction Manual.

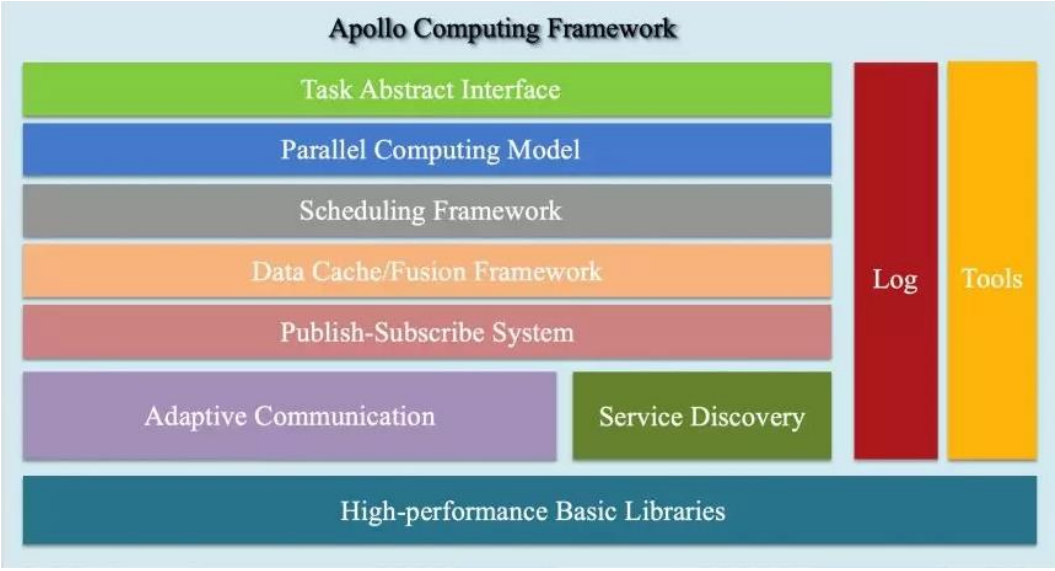
5.4 ROS

ROS (Robot Operating System) is a robot software platform that provides operating system-like functions for heterogeneous computer clusters. The predecessor of ROS was the Switchyard project established by the Stanford Artificial Intelligence Laboratory to support the Stanford Intelligent Robot STAIR. By 2008, the development of the project was continued mainly by Willow Garage. ROS provides some standard operating system services such as hardware abstraction, underlying device control, common function implementation, interprocess messages and packet management. ROS is based on a graph-like architecture whereby processes at different nodes can accept, publish and aggregate various information (such as sensing, control, state, planning, etc.). ROS can be divided into two layers, the lower layer is the operating system layer described above, and the upper layer is the various software packages

that the different user groups contribute to realize different functions, such as positioning drawing, action planning, sensing, simulation and so on. For details, please refer to the ROS website.

5.5 EcoCyber

EcoCyber is a high-performance runtime framework developed by Ecotrons LLC based on Apollo Cyber RT, designed for autonomous driving scenarios. Based on a centralized computing model, it is optimized for high concurrency, low latency and high throughput in autonomous driving to meet the requirements of autonomous driving solutions. EcoCyber is built on the concept of components. The result of component development is the base library and common components. The principle is high reuse and low coupling, resulting in the refined components for different functions. The specific functions depend on the extracted components, and the components themselves may have dependencies, but generally not much. Each component is a building block of the EcoCyber framework that includes a specific algorithm module that processes a set of input numbers and produces a set of output numbers.



EcoCyber's structure is shown above. The bottom layer is the basic library, and above it are communication-related modules, including service discovery and the Publish-Subscribe System. EcoCyber supports cross-process and cross-machine communication. The upper layer does not need to care about the implementation details, because the communication layer will

automatically select the corresponding communication mechanism according to the deployment of the algorithm module. Above the communication layer is the data cache/fusion layer. Data needs to be fused between multiple sensors, and the algorithm needs to cache certain data. For example, typical simulation applications require a data bridge between different algorithm modules, and the data layer acts as a bridge for communication between the modules. Further up is the computating model, which includes scheduling and tasks. Above the computing model is the interface provided to the developer. EcoCyber provides a Component Class for developers, which the developer only needs to inherit and implement the Proc interface. At the same time, EcoCyber is also based on coroutines, providing developers with parallel computing-related interfaces. There are also tools for development and debugging, recording and playback, and other performance debugging tools. For instructions on how to create a new component using Cyber RT, please refer to How to Create a New Component Using Cyber RT.

Chapter 6 Interface

In a Linux system, all devices are divided into three categories: character devices, block devices, and network interfaces.

A character device refers to a device that can read and write only one byte after another, or in order. Character devices are stream-oriented devices, so it cannot read data in the device memory randomly. Common character devices are mouse, keyboard, serial port, console, and LED devices. The console (/dev/console) and the serial port (/dev/ttyS0) are examples of character. Character devices are accessed through file system nodes, such as /dev/tty1 and /dev/lp0. The only relevant difference between a character device and a regular file is that you can often move around in a normal file, but most character devices are just data channels, and you can only access them sequentially. However, there are character devices that look like data areas, which you can move around. For example, frame grabber is often the case, where applications can use mmap or lseek to access the entire requested image.

A block device is a device that can read a certain length of data from any location of the device. Block devices include hard disks, USB flash drives and SD cards. Block devices are accessed through file system nodes located in the /dev directory. A block device (such as a disk) should be able to host a file system. Linux allows an application to read and write a block device like a character device, which allows any number of bytes to be transferred at a time. As a result, the difference between block and character devices is only in the kernel and in the way that data is managed internally, and therefore differs in the software interface of the kernel/driver. Like a character device, each block device is accessed through a file system node, and the difference between them is transparent to the user. The block driver is completely different in terms of the kernel driver from the character driver.

A network interface is a device that can exchange data with other hosts. Usually, an interface is a hardware device, but it can also be a pure software device, such as a loopback interface. A network interface is responsible for sending and receiving data messages. Under the driver of the kernel network subsystem, it is not necessary to know how a single transaction is mapped to the actual transmitted message. Many network connections (especially those using TCP) are

stream-oriented, but network devices are often designed to handle the sending and receiving of messages. A network driver knows nothing about a single connection; it only processes messages. Since it is not a stream-oriented device, a network interface is not as easy to map to a node of the file system as /dev/tty1. The way Unix provides access to interfaces is still by assigning a name to them, e.g. eth0, but the name does not have a corresponding entry in the file system. The communication between the kernel and the network device driver is completely different from that used for character and block device drivers. Instead of using read and write command, the kernel calls the socket function associated with the message passing.

For EAS2A01, different interfaces correspond to different types of devices in a Linux system. Among them, RS232 and HDMI belong to the character type device, Ethernet interface and CAN belong to the network interface, and the camera corresponding to the GMSL interface is not a standard device of the Linux system. For these devices, you can use the programming language C to write programs that call the driver access functions of these devices to access the interface. For details, please refer to the "EcoSDK-S2 Instruction Manual". We will explain in detail how to configure interface parameters through the console to access the interface in this chapter.

6.1 RS232

RS232, an asynchronous transmission standard interface developed by the Electronic Industries Association (EIA), is one of the communication interfaces on personal computers. To learn more about the RS232 interface, you can refer to RS232-Wikipedia. The RS232 interface is mapped to character device files in the Linux operating system, and RS232-1 and RS232-2 correspond to /dev/ttyLF0 and dev/ttyLF1, respectively.

To view the parameters of RS232, use this command:

```
# stty -F /dev/ttyLF1 -a
```

To set the baud rate of RS232-2 as 115200 and 8 data bits, use command below. If it can’t display properly, you may type in sty --help to see other configuration setup.

```
# stty -F /dev/ttyLF1 ispeed 115200 ospeed 115200 cs8
```

To print the data from RS232, use this command:

```
# cat /dev/ttyLF1
```

To send data by RS232, you can use this command:

```
# echo “hello world” > dev/ttyLF1
```

6.2 CAN

CAN is the abbreviation of Controller Area Network (CAN). It was developed by German BOSCH company, which is famous for developing and producing automotive electronics products, and eventually became an international standard (ISO 11898). It is one of the most widely used fieldbuses in the world. In North America and Western Europe, the CAN bus protocol has become the standard bus for automotive computer control systems and embedded industrial control LANs. To learn more about the CAN bus, you can refer to CANbus-Wikipedia. The basic software in the EAS2A01 device maps the CAN interface of the device to the SocketCAN in the operating system, which is managed as a network device by the system.

To view CAN device, you can use the command as below, whereas eth0 is ethernet interface, can0 and can1 are CANA and CANB respectively.

```
# ifconfig –a
```

```
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:30
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 00:03:00:00:02:41
          inet addr:192.168.1.238  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:146 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10061 (9.8 KiB)  TX bytes:5447 (5.3 KiB)
          Interrupt:21 Base address:0x4000
```

You can use the command below to configure the CAN baud rate as 500Kbps:

```
# ip link set can0 type can bitrate 500000
```

To view parameters of CAN, you can use this command:

```
# ip -details link show can0
```

To enable CANA, use this command:

```
# ifconfig can0 up
```

To disenable CANA, type in command like this:

```
# ifconfig can0 down
```

You can send a CAN frame with ID as 0x5A0 and data as 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, use this command:

```
 # cansend can0 -i 0x5a0 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
```

To receive data from CAN, use the command below:

```
# candump can0
```

6.3 Ethernet

Ethernet is the most widely used LAN communication method and a protocol. The Ethernet protocol defines a set of software and hardware standards that connect different computer devices together. The basic elements of Ethernet (Ethernet) device networking are switches, routers, hubs, fiber and common network cables, and Ethernet protocols and communication rules. The port for network data connection in Ethernet is the Ethernet interface. To learn more about Ethernet, you can refer to the Ethernet-Wikipedia. The basic software in the EAS2A01 device maps the Ethernet interface of the device to eth0 in the operating system.

To view Ethernet interface, you can use the command below, whereas eth0 is Ethernet interface, and can0 and can1 are CANA and CANB respectively.

```
# ifconfig –a
```

```
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:30
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 00:03:00:00:02:41
          inet addr:192.168.1.238  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:146 errors:0 dropped:0 overruns:0 frame:0
          TX packets:85 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10061 (9.8 KiB)  TX bytes:5447 (5.3 KiB)
          Interrupt:21 Base address:0x4000
```

You can set the IP address to be obtained automatically. Use the vi editor to open the network port configuration file and edit the file as shown below, and then restart the network port device. If you need help with how to use vi editor, please refer to A Beginner's Guide to Vim.

```
# vi /etc/network/interfaces
```

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
dhclient_opts -n

auto can0
iface can0 inet manual
```

```
# /etc/init.d/networking restart
```

You can set the IP address fixed. Use Vi editor to open the Ethernet configuration and edit the file as follows, then restart the Ethernet device.

```
# vi /etc/network/interfaces
```

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 10.0.0.100
        netmask 255.255.255.0
        gateway 10.0.0.1

auto can0
```

```
# /etc/init.d/networking restart
```

6.4 HDMI

High Definition Multimedia Interface is an all-digital video and audio transmission interface that provides uncompressed audio and video signals. HDMI can be used in set-top boxes, DVD players, personal computers, video game consoles, integrated amplifiers, digital audio and televisions. HDMI can transmit audio and video signals. Since the audio and video signals use the same cable, the installation of the system wiring is greatly simplified. To learn more about HDMI, see HDMI-Wikipedia. In the EAS2A01 device, the HDMI interface is mapped to the frame buffer device in the Linux operating system, and the video display device is driven in a memory buffer containing the complete frame data, and the corresponding device file is /dev/fb0. If the system has multiple graphics cards, multiple frame buffer devices can be supported under Linux, up to 32.

You can get a blurred screen on the monitor connected to HDMI.

```
# cat /dev/urandom > /dev/fb0
```

You can get a black screen on the monitor connected to HDMI.

```
# cat /dev/zero > /dev/fb0
```

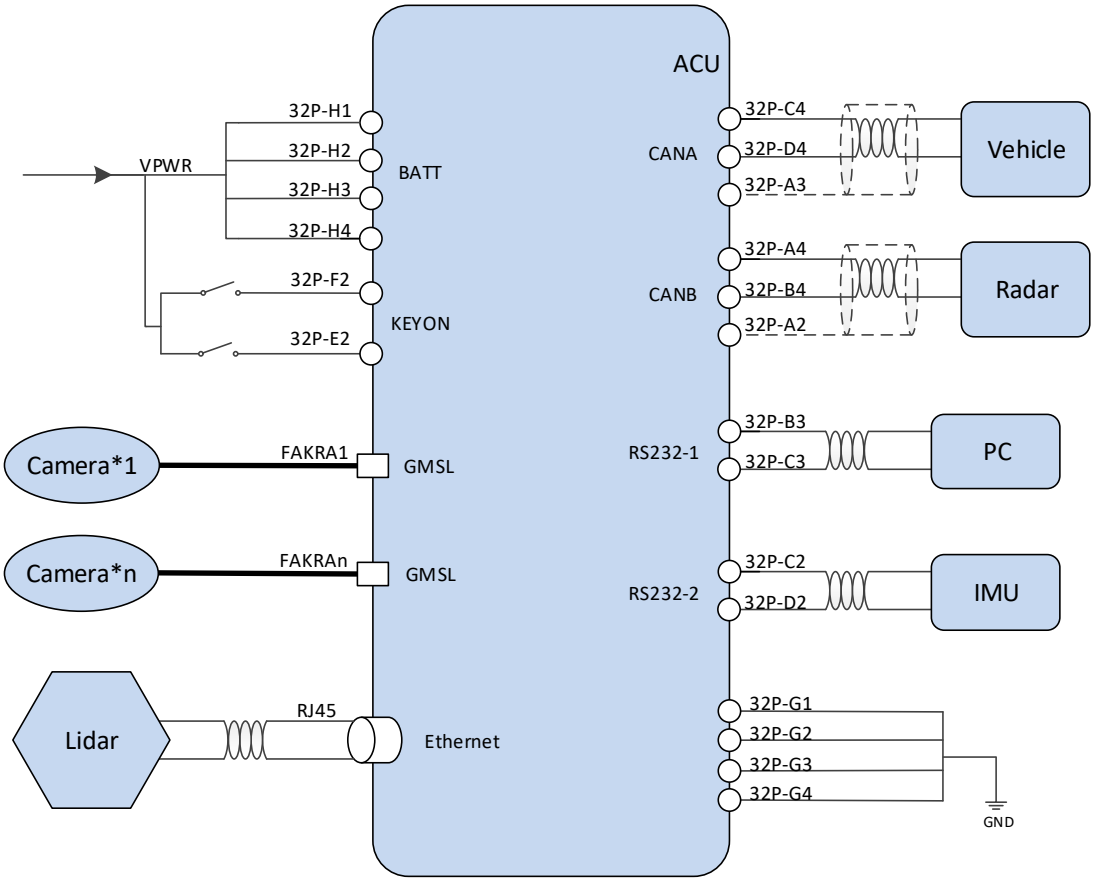
6.5 GMSL

The Gigabit Multimedia Serial Link (GMSL) fully supports the broadband, complex interconnection and data integrity required by future automotive infotainment systems and advanced driver assistance systems (ADAS). From the ultra-low power requirements of the camera to the broadband requirements for sensor data aggregation, GMSL SerDes can meet all the needs of future systems. Advanced link integrity and diagnostics provide reliable link performance monitoring, which is critical to the design of automotive safety systems. The GMSL serializer and deserializer support up to 15m of shielded twisted pair (STP) or coaxial cable transmission to meet the automotive industry's most demanding electromagnetic compatibility (EMC) requirements. Both the serializer and deserializer ICs have built-in spread spectrum capability to improve the electromagnetic compatibility (EMC) of the link without the need for an external spread spectrum clock. Interoperability between the serializer and deserializer families allows different interfaces to be used on both sides of the link. In addition to driving high-resolution center/rear display and dashboards, GMSL SerDes is also capable of megapixel camera system design. Based on the many performances of modern automotive video interconnects, GMSL technology provides broadband, feature-rich, design flexibility and other

advantages to support future automotive design requirements such as driverless systems. Different camera models need to be compatible with different driver packages and cannot be accessed directly through the console.

Chapter 7 Demo Application

The figure below is a demonstration of autonomous driving hardware platform, which consists of EAS2A01 and sensors.

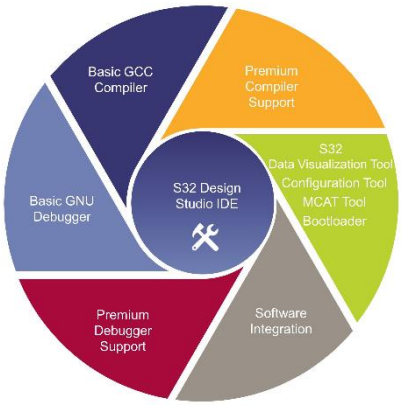


Chapter 8 Development Tool

Devices consisting of hardware, operating system stacks, and runtime environments are not capable of autopilot, and we need to continue to develop software packages that implement specific functionality, build them, and deploy them to the devices. Therefore, we offer three development tools that users can use to develop applications for specific scenarios.

8.1 S32 Design Studio for Vision IDE

S32 Design Studio for Vision IDE is an integrated development environment for the S32V234 processor, which is capable of code editing, code compiling and debugging. It is based on open source software such as the Eclipse IDE, the GNU Compiler Suite (GCC), and the GNU Debugger (GDB). S32 Design Studio for Vision IDE provides designers with an intuitive development tool with no code size limitations and visual programming tools, making visual accelerometer programming easier. NXP's other software packages, along with S32 Design Studio, provide a comprehensive support environment that can reduce development time. Please refer to the S32 Design Studio for Vision IDE for details.



8.2 VSDK

The Vision Software Development Kit (VSDK) provides a comprehensive development environment for the S32V234 vision processor family. The VSDK can be used to develop applications for the Arm® Cortex®-A53. With the graphical tools in the S32 Design Studio for Vision IDE, you can also develop computationally intensive image processing application based

on Image Signal Processing (ISP) and Parallel Computing (APEX). The VSDK also includes a demo application that runs on Arm® Cortex®-A53, Image Signal Processing (ISP), Parallel Computing (APEX) and provides complete source code. Please download and view the Vision SDK Software for details.

8.3 EcoSDK-S2

EcoSDK-S2 provides users with a complete application development environment, including:

- Cross-development toolchain: consists of a cross-compiler, cross-connector, cross-debugger, and a set of other tools for application development.
- System root: EcoSDK-S2 contains 2 system roots: one for the development host, which contains the cross-development toolchain and other tools; the other for the target, which is the full root file system for the target, contains development kits consisting of header files and libraries.
- Environment settings: The script provided by the EcoSDK-S2 package allows you to configure an environment for cross-development on the development host.
- Analysis tools: A variety of user space tools for analyzing your application on your target system.

The components in EcoSDK-S2 give application developers all the tools necessary to write applications based on the Linux operating system, ROS, and EcoCyber. For details, please refer to EcoSDK-S2 Instruction Manual.

8.4 EcoCoder-AI

EcoCoder-AI is a powerful automatic code generation library based on Matlab / Simulink that links directly to the target controller. EcoCoder-AI integrates code generation, compilation and one-click generation of executable files. It is possible to directly convert the control model based on Simulink into an EcoCyber-based executable program for the target controller and download it to the target controller. For details, please refer to EcoCoder-AI Instruction Manual.