EAXVA01 Datasheet

- Operating voltage: DC 9-36V
- Calculation capability: 30TOPS
- Data transmission interfaces:
  - 1 Gigabit Ethernet interface
  - 1 HDMI output interface
  - 6 FPDLink III input interface
  - 2 CAN interfaces
  - 2 RS-232 interfaces
- Main chip: Nvidia Xavier
- Operating Temperature: -25 to 80 ℃
- Humidity: 0 - 95%, no condensation
- Storage temperature: -25 to 80 ℃
- Dimensions: 285×143×42mm
- Weight: less than 1500g
- Operation memory: 16GB
- Storage memory: 32GB

Revision History

Time	Version	Detail	Reviser
Apr. 29, 2019	V1.0	First version	David Wang
Sep. 20, 2019	V1.1	First page updated	David Wang
Feb. 11, 2020	V1.2	Logo, address, EcoCoder-AV 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

Contents

Chapter 1 Summery 5

Chapter 2 Mechanics 6

2.1 Dimensions 6

2.2 Connector 6

Chapter 3 Quick Start 8

3.1 Preparation 8

3.2 Basic Knowledge 8

3.3 Using the device 8

3.3.1 Connect all the components 8

3.3.2 Configure 8

3.3.3 Boot 8

3.3.4 Test 8

3.3.5 Develop 10

Chapter 4 Hardware 11

4.1 Specifications 11

4.2 Device Ports 12

4.3 System Main Chip 13

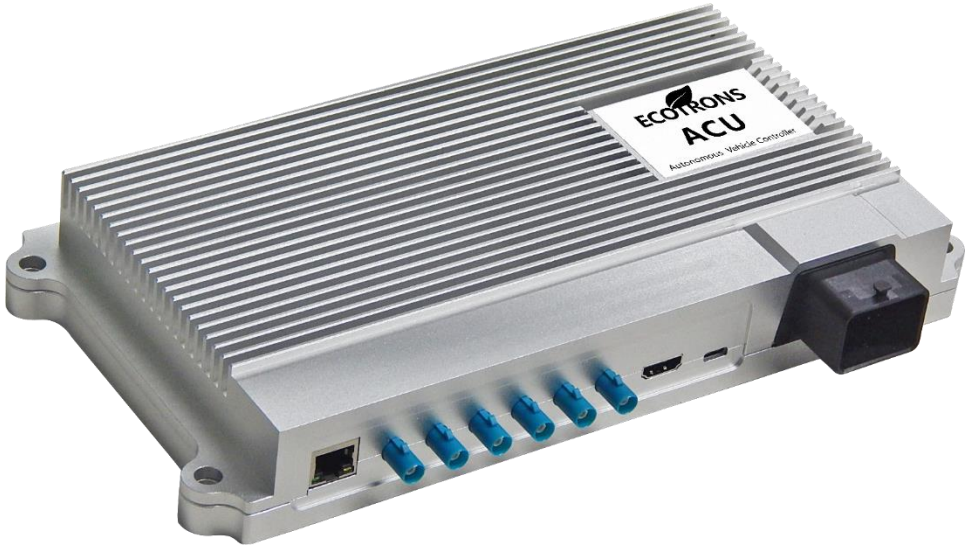
4.3.1 Volta GPU 15

4.3.2 Operation Test 15

4.4 Circuit Structure 16
Chapter 5 Software 18
5.1 Real-Time Operating System 18
5.2 ROS 19
5.3 EcoCyber 19
Chapter 6 Interface 22
6.1 RS232 23
6.2 CAN 24
6.3 Ethernet 25
6.4 FPDLink-III 28
Chapter 7 Demo Application 29
Chapter 8 Development Tool 30
8.1 Local Development Tool Kit 30
8.2 EcoSDK-XV 30
8.3 EcoCoder-AI 30

Chapter 1 Summery

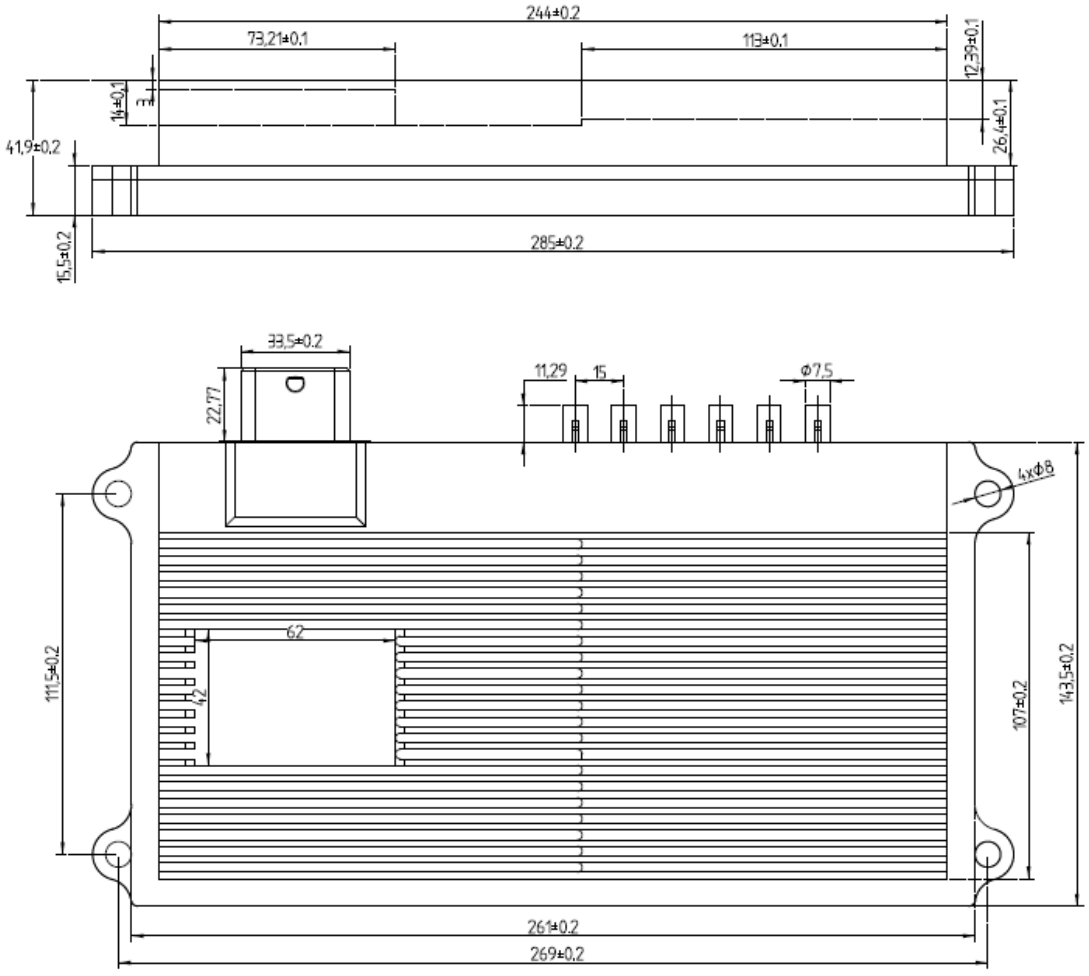
EAXVA01 is a central computing platform developed by Ecotrons LLC for autopilot systems. The internal main computing chip is NVIDIA's Xavier. Xavier is designed by NVIDIA for embedded intelligent systems including automated driving systems. With more than 9 billion transistors, Xavier can perform 30 trillion operations per second, but with a power of only 30 watts, 20 times faster than existing TX2 platforms. Xavier has six different processors that enable it to process dozens of algorithms simultaneously and in real-time for sensor processing, ranging, positioning and mapping, vision and perception, path planning, and vehicle control. The software system is also customized for the autopilot system, including a real-time optimized Linux operating system, high-performance runtime framework ROS, EcoCyber, etc., enabling developers to build automated driving system that meets the L4 requirements, safely, conveniently and efficiently.



Chapter 2 Mechanics

2.1 Dimensions

The size of housing is 285 × 143 × 42 mm (connectors excluded). The color is silver, and the material is aluminum.



2.2 Connector

EAXVA01 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, 9-36V DC / 2 A minimum
- USB to RS-232 adapter
- Network cable with RJ-45 connector
- Computer with ubuntu 18.04 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 computer through network cable to the same local area network as the device is in.

3.3.2 Configure

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

3.3.3 Boot

Turn on the device's KeyOn switch and start the device power, the Linux will start. The username is ‘nvidia’, and the password is ‘nvidia’. 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 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 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. Follow the instructions, 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.

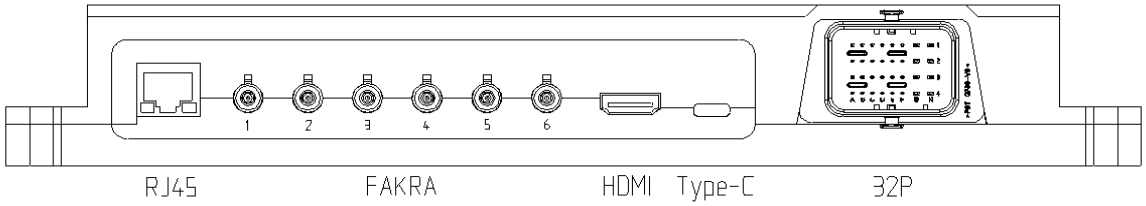
- Operating voltage: DC 9-36V
- Operation memory: 16GB
- Storage memory: 32GB
- Calculation capability: 30TOPS
- Data transmission interfaces:
    - 1 Gigabit Ethernet interface
    - 1 HDMI output interface
    - 6 FPDLink III input interface
    - 2 CAN interfaces
    - 2 RS-232 interfaces

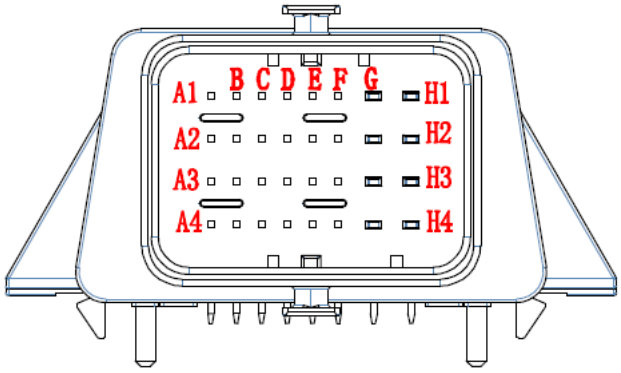
4.1 Specifications

Item	Detail
Operating voltage	DC 9-36V
Operation memory	16GB
Storage memory	32GB
Operating temperature	-25 to 80 ℃
Operating humidity	0 - 95%, no condensation
Storage temperature	-25 to 80 ℃
Dimensions	285×143×42mm
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
FAKRA1	FPD-Link III Input 1	Camera
FAKRA2	FPD-Link III Input 2	Camera
FAKRA3	FPD-Link III Input 3	Camera
FAKRA4	FPD-Link III Input 4	Camera
FAKRA5	FPD-Link III Input 5	Camera
FAKRA6	FPD-Link III Input 6	Camera
HDMI	High-Definition Multimedia Interface Output	
Type-C	USB-C interface	
32P-H1	Power Positive	DC 9-36V
32P-H2	Power Positive	DC 9-36V
32P-H3	Power Positive	DC 9-36V

32P-H4	Power Positive	DC 9-36V
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:

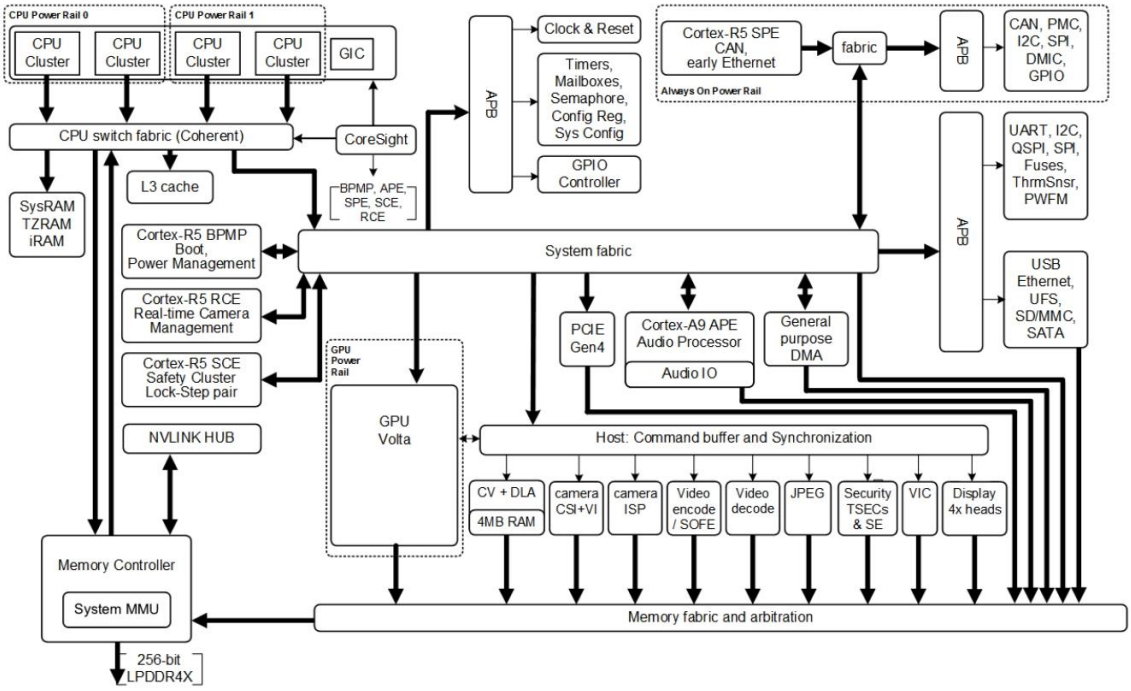
The output voltage of the FAKRA interface is hardware configurable. There are two voltage levels to choose from: DC 5V and DC 12V. Please refer to the EAXVA01 Product Test Report for specific output voltage or contact the equipment supplier.

Default terminal resistance of CAN interface is 120Ω.

4.3 System Main Chip

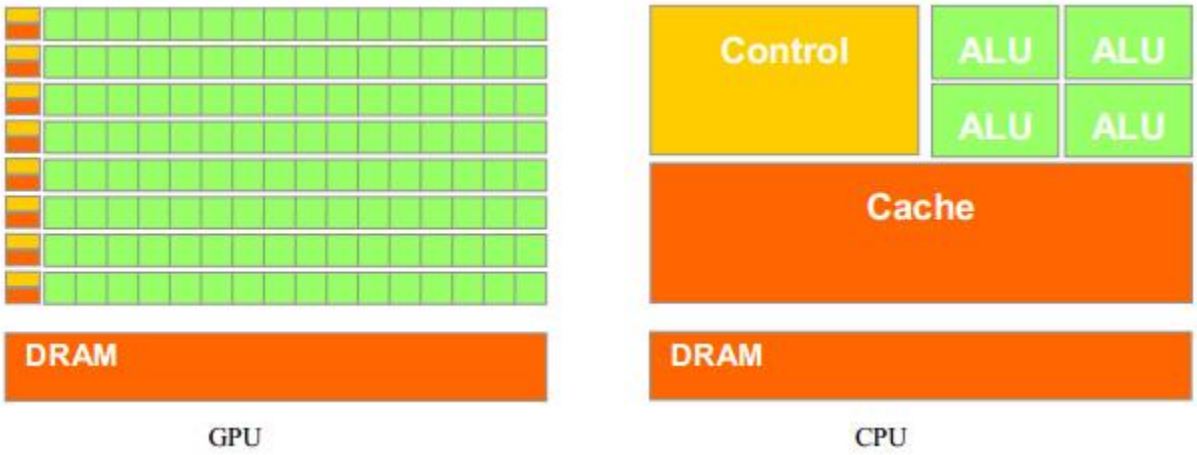
The main chip of EAXVA01 is NVIDIA's Xavier which is designed for embedded intelligent systems including autopilot systems. Xavier has six different processors: Valta Tensor Core GPU, eight-core ARM64 CPU, dual NVDLA deep learning accelerator, image processor, vision processor and video processor. These processors enable dozens of algorithms to be processed simultaneously and in real-time for sensor processing, ranging, positioning and mapping, vision and perception, and path planning. This level of performance is critical, allowing robots to take input from sensors, locate themselves, sense their environment, identify and predict motion of nearby objects, reasonably act and perform safely. With more than 9 billion transistors, Jetson Xavier can perform 30 trillion operations per second, with the same processing power as a workstation with a $10,000 GPU, but with a power of only 30 watts. Xavier is 20 times faster than the existing Jetson TX2 platform. The computational performance of different internal processors and internal structure are show below.

- Eight-core CPU: Eight-core "Carmel" CPU based on ARMv8 ISA
- Deep Learning Accelerator (DLA): 5 TOPS (FP16) | 10 TOPS (INT8)
- Volta GPU: 512 CUDA cores | 20 TOPS (INT8) | 1.3 TFLOPS (FP32)
- Vision Processor: 1.6 TOPS
- Stereo and Optical Flow Engine (SOFE): 6 TOPS
- Image Signal Processor (ISP): 1.5 Giga Pixels/s
- Video Encoder: 1.2 GPix/s
- Video Decoder: 1.8 GPix/s



4.3.1 Volta GPU

Because GPUs have a highly parallel structure, GPUs have higher efficiency than CPUs in processing graphics data and complex algorithms. Most of the CPU area is occupied by controllers and registers. In contrast, GPUs have more ALUs (Arithmetic Logic Units) for data processing than data cache and flow control, which is targeted at parallel computation of computationally intensive data.



4.3.2 Operation Test

We use the performance test routines in the system software installation package provided by NVIDIA to draw Mandelbrot fractal graphs separately using ARMv8 CPU and Volta GPU to compare the time it takes to draw a picture.

Test project: Compare the time required to draw a Mandelbrot fractal graph with an ARMv8 CPU and a Volta GPU

Test process: By inputting instructions, the control application switches the operation between the CPU and the GPU, records the time it takes to draw a Mandelbrot fractal graph.

Test result:

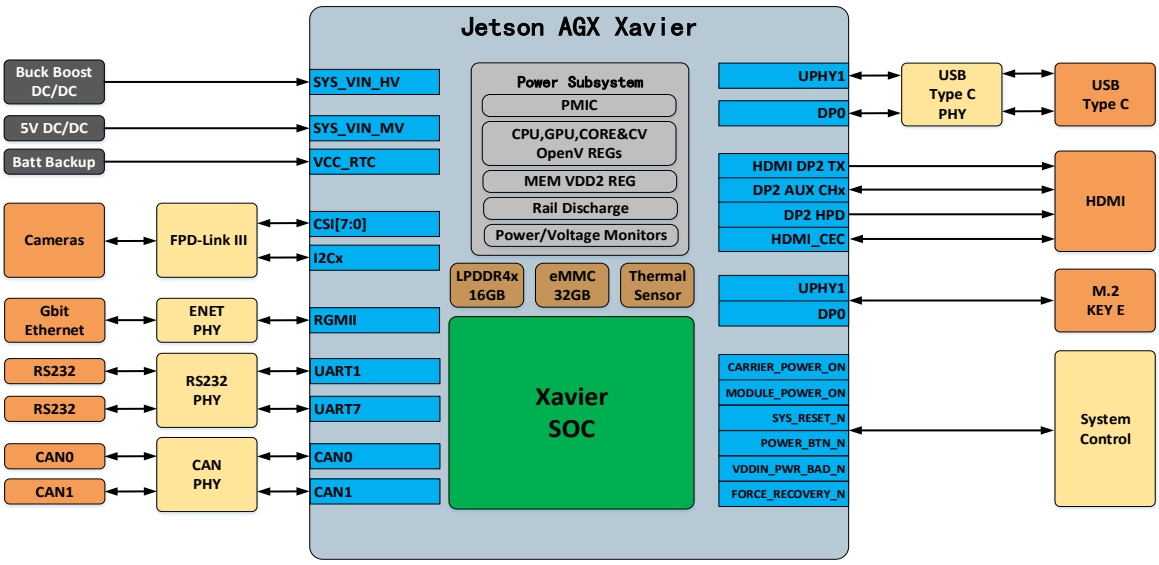
Round	CPU (Unit: s)	GPU (Unit: s)	Acceleration Factor
1	0.71118802	0.00064200	1107.77
2	0.70679700	0.00065700	1075.79
3	0.71068001	0.00063600	1117.42
4	0.72342801	0.00049100	1473.38
5	0.70813304	0.00061800	1145.85
6	0.70578402	0.00055500	1271.68
7	0.71340507	0.00054700	1304.21
8	0.70767504	0.00049400	1432.54
9	0.70830107	0.00070400	1006.11
10	0.70901704	0.00060800	1166.15
Average	0.71044083	0.00059520	1193.62

Test conclusion:

Volta GPU Acceleration Unit accelerates approximately 1193 times compared to ARMv8 CPU.

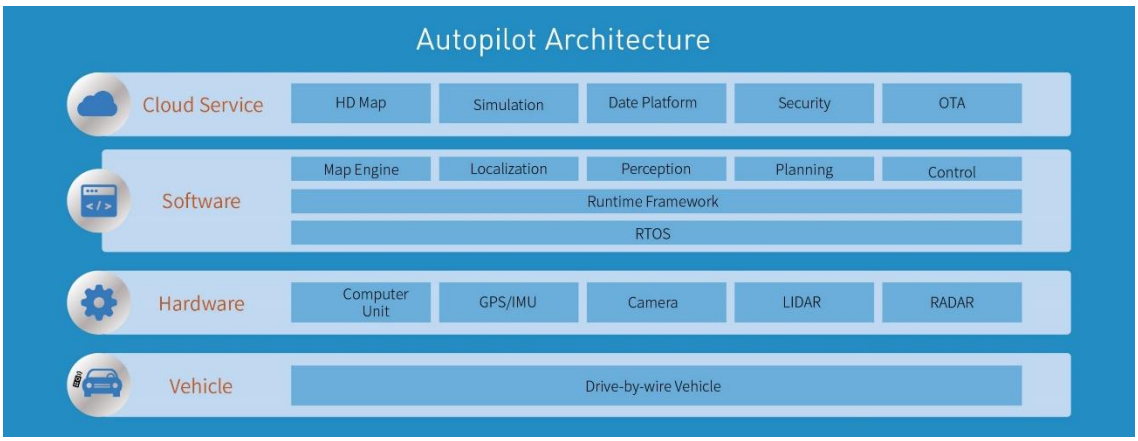
4.4 Circuit Structure

The internal circuit structure is shown below:



Chapter 5 Software

The software system of EAXVA01 is customized for the autonomous driving system. The following figure is a typical block diagram of the autonomous driving system. The software system of EAXVA01 consists of RTOS, Runtime Framework and so on. The RTOS is a real-time optimized RT-Linux operating system based on Ubuntu 18.04. The Runtime Framework has two software frameworks to choose from: one is the Melodic version of ROS (Robot Operating System), and the other is EcoCyber based on Apollo Cyber RT developed by Ecotrons.



5.1 Real-Time Operating System

Linux is a free-to-use and freely distributed Unix-like operating system. It is a multi-user, multi-tasking, multi-threaded and multi-CPU based operating system based on POSIX and UNIX. It runs major UNIX utilities, applications and network protocols. It supports both 32-bit and 64-bit hardware. We are using the RT-Linux operating system based on the Ubuntu 18.04 version optimized for real-time performance. Real-time operating system (RTOS) means that when external events or data are generated, the system can be accept and process them at a fast enough speed, and the processing result can control the production process or make a quick response to the processing system within a specified time. The RTOS will schedule all available resources to complete real-time tasks and coordinate all the real-time tasks. Its main feature is to provide timely response and high reliability. For non-real-time operating systems, when opening multiple applications, to ensure the user experience, the system must respond them all, so all the applications will share the computing resources at the same time. As the result, every

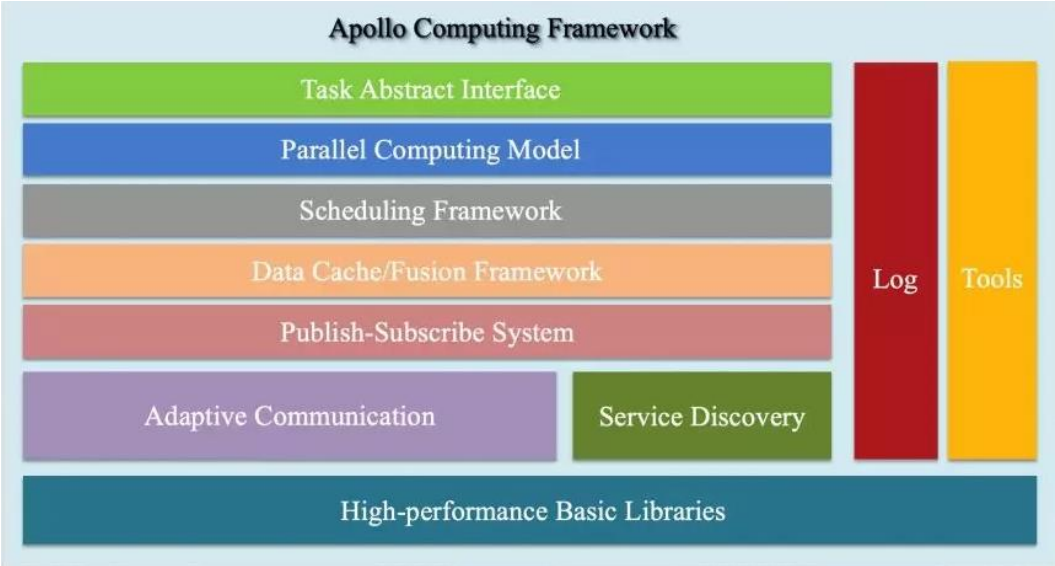
application can be executed, but none of them is running smoothly. For real-time operating systems, it is characterized by the fact that if a task needs to be executed, it will be executed immediately within a short delay, rather than pursuing multiple tasks simultaneously. This feature has its own advantages in autonomous driving: the priority of different commands can be set in advance and high-priority tasks can be executed immediately. For critical applications, no matter what state other applications are in, the system must ensure adequate storage and CPU resources for it, so that critical applications can work well under all circumstances. Please refer to Ubuntu for details.

5.2 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 http://www.ros.org/.

5.3 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 EAXVA01, different interfaces correspond to different types of devices in a Linux system. Among them, RS232 belong to the character type device, Ethernet interface and CAN belong to the network interface. 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-XV 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 EAXVA01 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
```

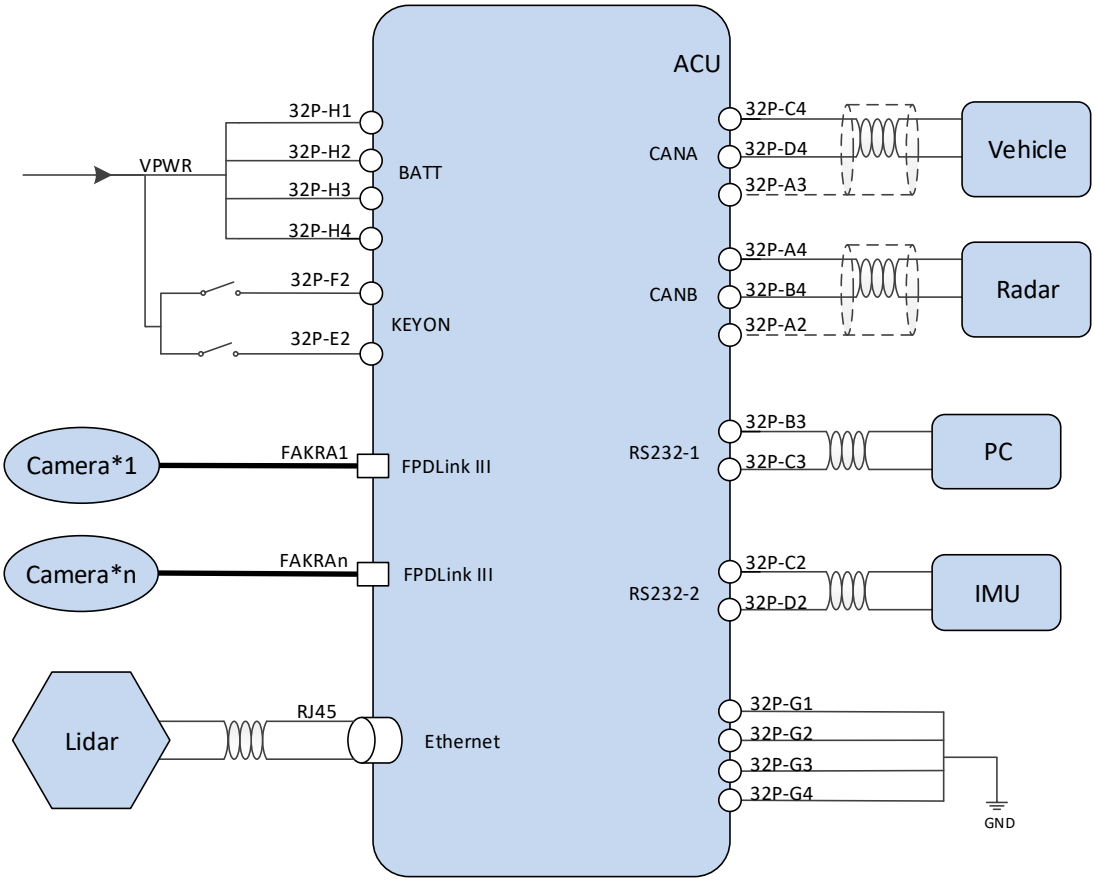
6.4 FPDLink-III

The FPD-Link III serial bus solution supports full-duplex control of high-speed video data transmission and bidirectional control communication over a single differential link. Integrating video data and control with a single differential pair reduces interconnect size and weight while eliminating bias issues and simplifying system design. Initially, the FPD-LINK application was used for video display on laptops. TI developed the FPD-Link III serializer/deserializer in conjunction with the automotive entertainment information system application environment, which was adopted in infotainment system displays and ADAS camera applications. An STP or coaxial cable carries video, audio, control data and power at the same time. To learn more about FPD-Link, you can refer to FPD-Link-Wikipedia.

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 EAXVA01 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 and deploy them to the devices. Therefore, we offer three development tools that users can use to develop applications for specific scenarios.

8.1 Local Development Tool Kit

EAXVA01 has a set of local development tools installed, including gcc, make, CMake, catkin, Bazel and gdb debugger. Application developers can develop user space applications directly on the EAXVA01 platform.

8.2 EcoSDK-XV

EcoSDK-XV 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-XV 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-XV 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-XV 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-XV Instruction Manual.

8.3 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 or ROS-based executable program for the target controller and download it to the target controller. For details, please refer to EcoCoder-AI Instruction Manual.