ECOTRON

EcoSDK-S2

User Manual

Revision History

Time	Version	Detail	Reviser
Apr. 30, 2019	V1.0	First version	David Wang
Sep. 20, 2019	V1.1	First page updated	David Wang
Feb. 11, 2020	V1.2	Logo, address updated	David Wang
Feb. 20, 2020	V1.3	Content 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

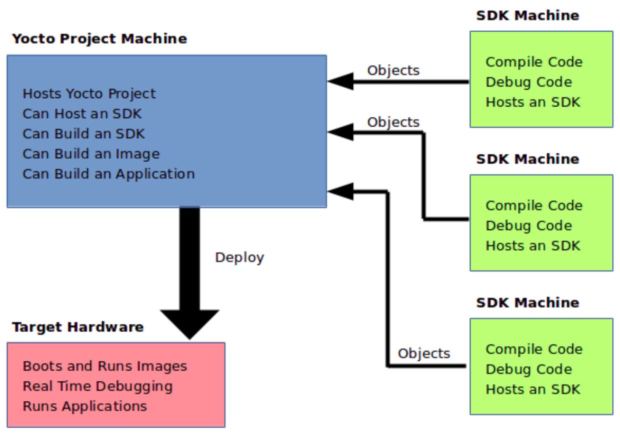
Chapter 1 Summary

EcoSDK-S2 is a PC-side cross-development tool developed by Ecotrons LLC for its computing platform EAS2A01. For detailed information about EAS2A01, please refer to EAS2A01 Datasheet. 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 the development kit with header file and libraries in it.
- Environment settings: The script provided by 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 software package in EcoSDK-S2 provides application developers with all the necessary tools for developing applications based on Linux and ROS (Robot Operating System). These applications can be based on GNU Make, GNU Autotool, or CMake. After initializing the cross-development environment, the user can cross-compile the target application on the development host.

Compared to S32 Design Studio for Vision IDE from NXP, EcoSDK-S2 provides application developers with the ability to cross-compile ROS node programs. The application developer only needs to execute the environment configuration script of the software, then he can use the same compile instruction as the PC side to compile and execute the executable program file that can be run on the EAS2A01 device according to the ROS node program code developed by the PC.


Yocto Project Machine
Hosts Yocto Project
Can Host an SDK
Can Build an SDK
Can Build an Image
Can Build an Application
Deploy
Target Hardware
Boots and Runs Images
Real Time Debugging
Runs Applications
Objects
Objects
Objects
SDK Machine
Compile Code
Debug Code
Hosts an SDK
SDK Machine
Compile Code
Debug Code
Hosts an SDK
SDK Machine
Compile Code
Debug Code
Hosts an SDK

Chapter 2 Basic Knowledge

EcoSDK-S2 is a cross-development tool for developing Linux system user space applications and ROS system applications. Here are some materials you may find helpful.

If you are a Linux beginner, maybe it's helpful to learn some quick tutorials about Linux command line tools. Here is the link: http://www.ee.surrey.ac.uk/Teaching/Unix/. If you want to learn C or C++ programming, you can check out the C language tutorial, or the C++ tutorial. If you want to program based on Linux, try this. If you want to use the Makefile to manage your software code, this tutorial might be able to help you. If you want to use CMake to manage your software code, you can go to CMake Homepage. If you want to write an application based on ROS (Robot Operating System), you can start from the wiki of ROS. If you want to learn about the GDB debugger, check out the GDB website.

Chapter 3 Software Installation

3.1 Computer Configuration

Requirement	Minimum Configuration
Operating System	Ubuntu-14.04-amd64
CPU	Intel(R) Core(TM)2 Duo CPU E4600 @2.40GHz
RAM	2G
Hard Disk	10G available space
Requirement	Recommended Configuration
Operating System	Ubuntu-14.04-amd64
System Language	English
CPU	Intel(R) Core(TM) i3-2120 CPU@3.30GHz
RAM	4G
Hard Disk	10G available space

3.2 Install EcoSDK-S2

3.2.1 Installation

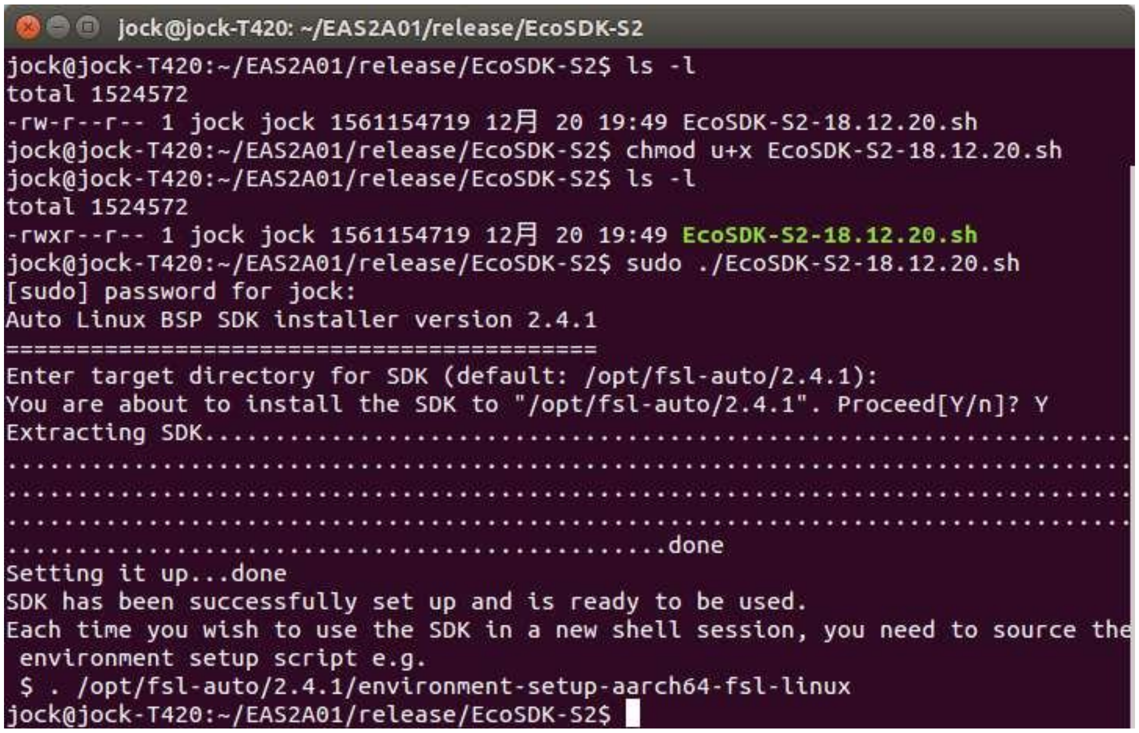
Open the command terminal and go to the directory where EcoSDK-S2 is located and check whether the file is executable. If it's not executable, use the chmod command to change the file's attributes. Run the setup file and the installation will start. During the installation, the user will be prompted to enter the installation directory. If you choose the default installation directory, just press enter. Installation steps are shown in the figure below.

```
$ ls -l
```

```
$ chmod  u+x  EcoSDK-S2-18.12.20.sh
```

```
$ ls -l
```

```
$ sudo  ./EcoSDK-S2-18.12.20.sh
```



```
jock@jock-T420: ~/EAS2A01/release/EcoSDK-S2
jock@jock-T420:~/EAS2A01/release/EcoSDK-S2$ ls -l
total 1524572
-rw-r--r-- 1 jock jock 1561154719 12月 20 19:49 EcoSDK-S2-18.12.20.sh
jock@jock-T420:~/EAS2A01/release/EcoSDK-S2$ chmod u+x EcoSDK-S2-18.12.20.sh
jock@jock-T420:~/EAS2A01/release/EcoSDK-S2$ ls -l
total 1524572
-rwxr--r-- 1 jock jock 1561154719 12月 20 19:49 EcoSDK-S2-18.12.20.sh
jock@jock-T420:~/EAS2A01/release/EcoSDK-S2$ sudo ./EcoSDK-S2-18.12.20.sh
[sudo] password for jock:
Auto Linux BSP SDK installer version 2.4.1
==========================================
Enter target directory for SDK (default: /opt/fsl-auto/2.4.1):
You are about to install the SDK to "/opt/fsl-auto/2.4.1". Proceed[Y/n]? Y
Extracting SDK...............................................................
.............................................................................
.............................................................................
.............................................................................
...............................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
 environment setup script e.g.
 $ . /opt/fsl-auto/2.4.1/environment-setup-aarch64-fsl-linux
jock@jock-T420:~/EAS2A01/release/EcoSDK-S2$
```

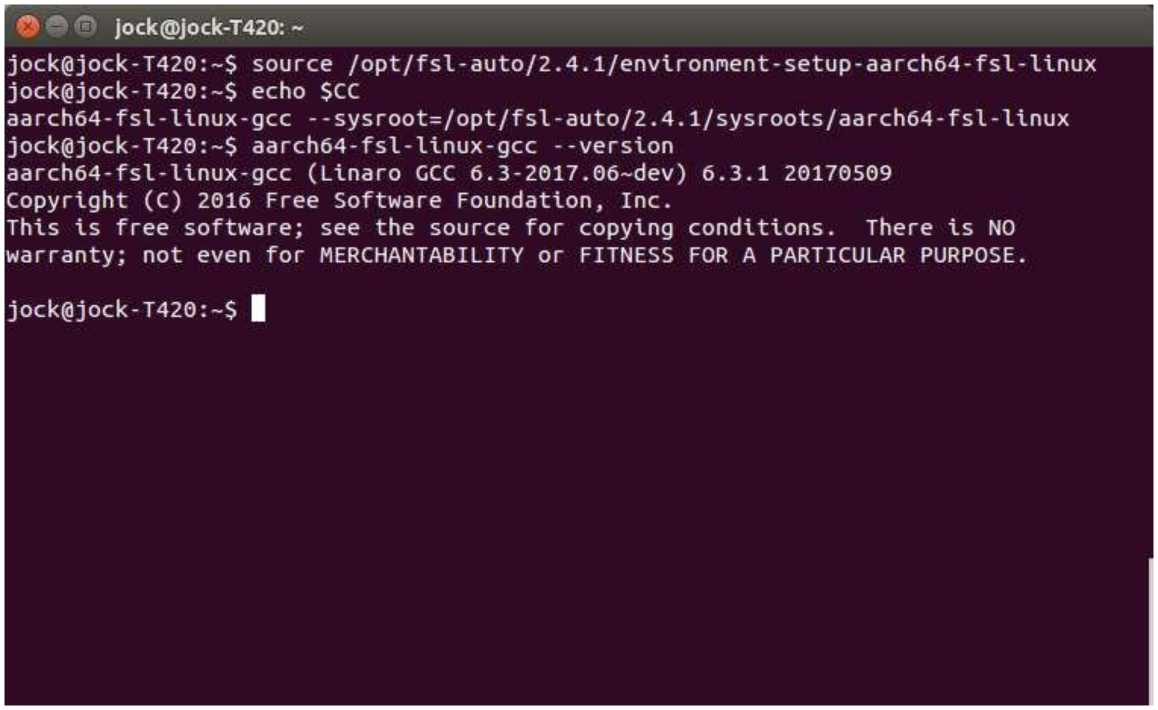
3.2.2 Verification

In the command terminal, enter the command "source <install path>/environment-setup-aarch64-fsl-linux" to configure the cross-development environment, where <install path> is the installation directory selected during installation. If you select the default installation path, <install Path> is "/opt/fsl-auto/2.4.1". Then type in the commands "echo $CC" and "aarch64-fsl-linux-gcc --version" respectively. If you see the messages as shown below, EcoSDK-S2 is installed successfully.

```
$ source <install path>/environment-setup-aarch64-fsl-linux
```

```
$ echo $CC
```

```
$ aarch64-fsl-linux-gcc --version
```


jock@jock-T420: ~

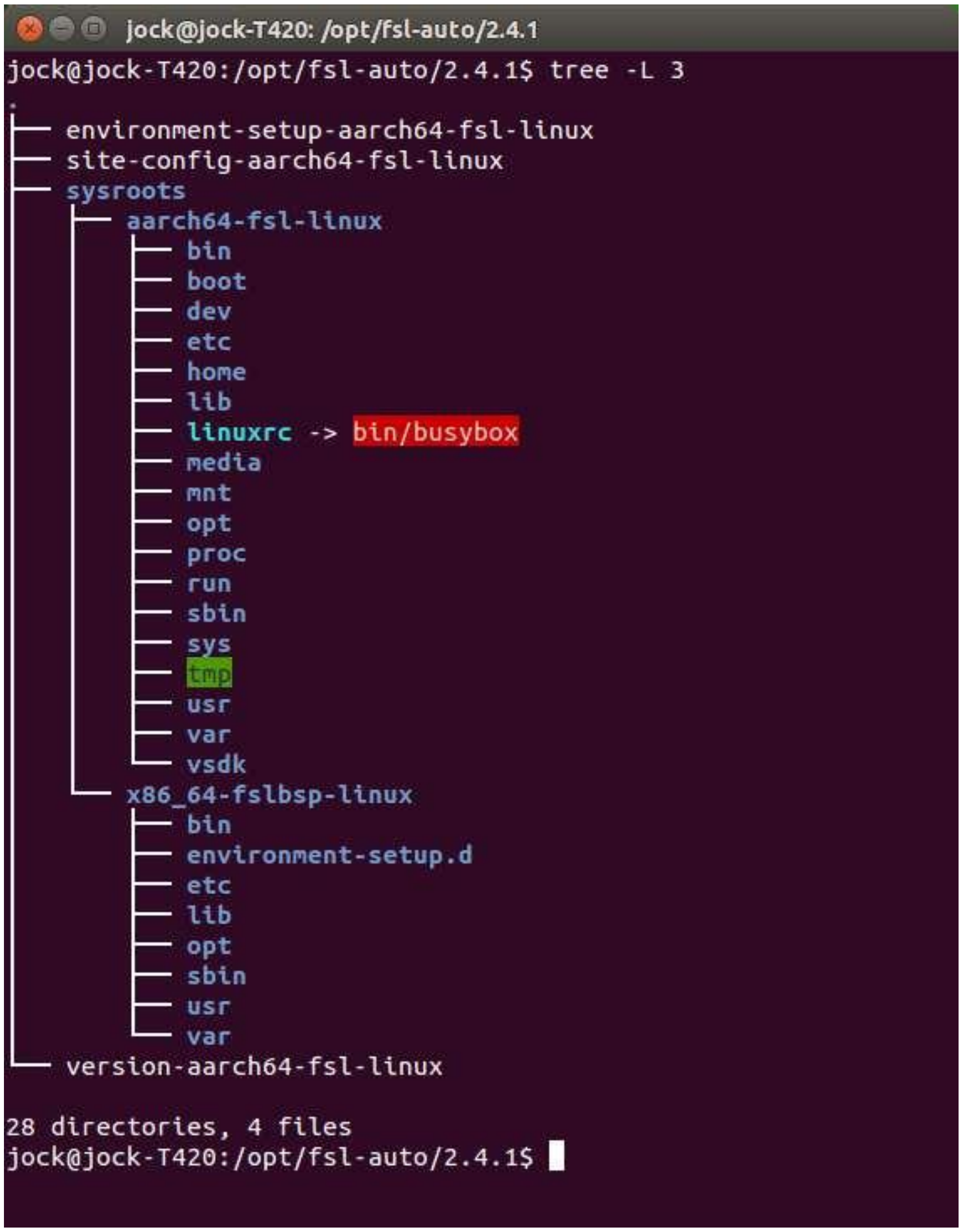
```
jock@jock-T420:~$ source /opt/fsl-auto/2.4.1/environment-setup-aarch64-fsl-linux
jock@jock-T420:~$ echo $CC
aarch64-fsl-linux-gcc --sysroot=/opt/fsl-auto/2.4.1/sysroots/aarch64-fsl-linux
jock@jock-T420:~$ aarch64-fsl-linux-gcc --version
aarch64-fsl-linux-gcc (Linaro GCC 6.3-2017.06~dev) 6.3.1 20170509
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

jock@jock-T420:~$ 
```

Chapter 4 Using EcoSDK-S2

4.1 Package structure

You can use command ‘tree’ to see the structure of EcoSDK-S2.



```
jock@jock-T420: /opt/fsl-auto/2.4.1
jock@jock-T420:/opt/fsl-auto/2.4.1$ tree -L 3
.
├── environment-setup-aarch64-fsl-linux
├── site-config-aarch64-fsl-linux
├── sysroots
│   ├── aarch64-fsl-linux
│   │   ├── bin
│   │   ├── boot
│   │   ├── dev
│   │   ├── etc
│   │   ├── home
│   │   ├── lib
│   │   ├── linuxrc -> bin/busybox
│   │   ├── media
│   │   ├── mnt
│   │   ├── opt
│   │   ├── proc
│   │   ├── run
│   │   ├── sbin
│   │   ├── sys
│   │   ├── tmp
│   │   ├── usr
│   │   ├── var
│   │   └── vsdk
│   └── x86_64-fslbsp-linux
│       ├── bin
│       ├── environment-setup.d
│       ├── etc
│       ├── lib
│       ├── opt
│       ├── sbin
│       ├── usr
│       └── var
└── version-aarch64-fsl-linux

28 directories, 4 files
jock@jock-T420:/opt/fsl-auto/2.4.1$
```

There are several types of files and subdirectories:

- Environment settings: The environment-setup-aarch64-fsl-linux script is used to initialize the environment variables needed to cross-compile the target program on the host.
- Site configuration: When you are using GNU Autotools development package, the site configuration file site-config-* contains configuration settings.
- System root: The sysroot subdirectory contains a system root subdirectory for the target machine and a system root subdirectory for the host. The subdirectory x86_64-fslbsp-linux is the system root for the host and contains the cross toolchain. The subdirectory aarch64-fsl-linux is the root file system for the target EAS2A01, and contains executable programs, software libraries, configuration files, etc. that can run on the target EAS2A01.
- Version file: The version-* file contains version information about the toolchain version.

Notes: By default, the created toolchain only builds dynamically linked binaries. If you want to build a statically linked binary, you need to make sure that you can the package containing the static library in your system root.

4.2 Compile an Application

Before using EcoSDK-S2 to develop applications, initialize the development environment with the corresponding script. Execute the environment configuration script environment-setup-aarch64-fsl-linux in the installation directory. If you open the script, you will see a set of environment variables related to cross-compilation. The following sections explain how to use EcoSDK-S2 for cross-compilation for direct compilation projects, Autotools-based projects, Makefile-based projects and ROS-based projects.

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.2.1 Use EcoSDK-S2 for Direct Compilation Project

We'll take a simple program with only one C file as an example to show how to use the cross compiler in EcoSDK-S2 to create an executable program that can run on EAS2A01.

4.2.1.1 Create Your Working Directory and Write a Program

Create an empty directory and set it as your working directory.

```
$ mkdir  $HOME/helloworld
```

```
$ cd  $HOME/helloworld
```

Create a file named hello-world.c, and fill the content as below:

hello-world.c:

```
#include <stdio.h>
int main()
{
     printf("Hello, World! \n");
     return 0;
}
```

4.2.1.2 Initialize Cross-Development Environment

When you install EcoSDK-S2, it will create cross-toolchain environment setup script in the installation directory automatically. Before you develop applications using these tools, you should run the script first. The script's name starts like "environment-setup".

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.2.1.3 Cross-Compile the Application

You can use this command to cross-compile the project you work on.

```
$ $CC  hello-world.c  -o  hello-world
```

Then you can verify the project with an easy command shown below. It will print out the file structure of the executable binaries.

```
$ file  hello-world
```

4.2.1.4 Execute the programs

You can use command scp to copy your executable hello-world into your target system through network. Please replace the <target_ip> with your target EAS2A01's IP address.

```
$ scp  hello-world  root@<target_ip>:/home/root
```

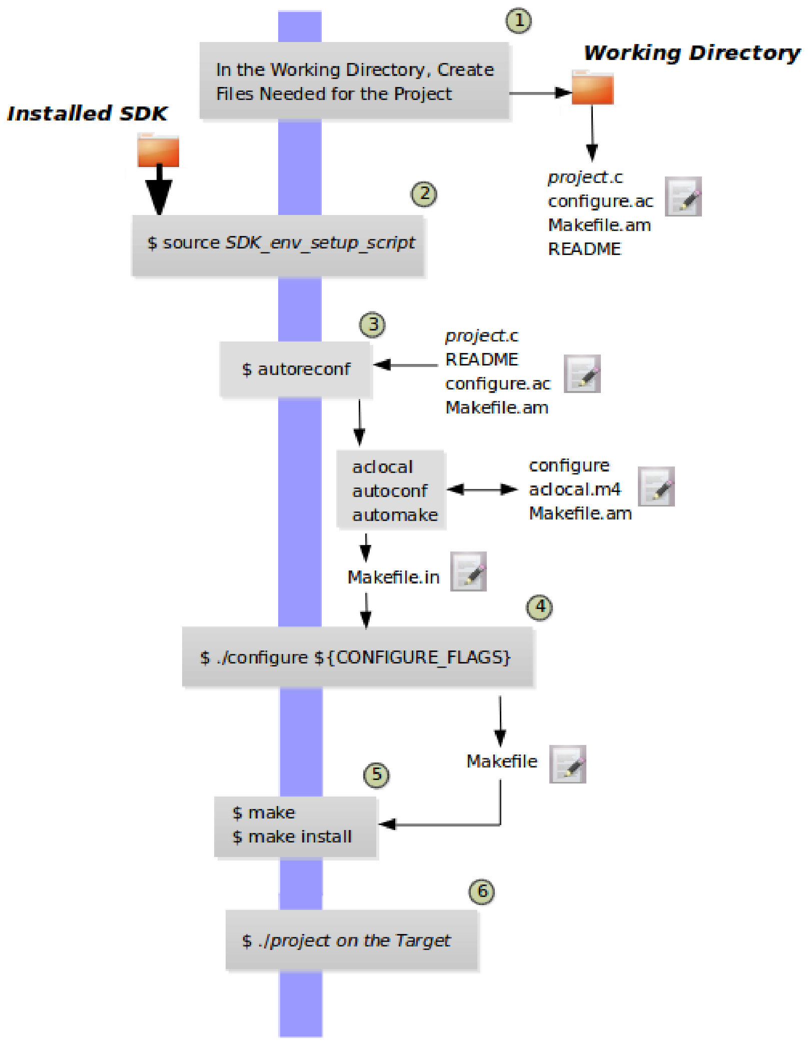
Set this file as an executable file through serial port terminal and execute the program.

```
# chmod  u+x  hello-world
# ./hello-world
hello,world!
```

Now it’s done!

4.2.2 Use EcoSDK-S2 for Autotools-based Project

After you install EcoSDK-S2, you can easily use the workflow cross development application based on GNU. Here is a simple Autotools workflow.



You can follow the instructions below to create a simple Hello World program with Autotools. For detailed information of GNU Autotools workflow, please refer to the same demo on GNOME Developer.

4.2.2.1 Create Your Working Directory and Write a Program

Create an empty directory and set it as your working directory.

```
$ mkdir  $HOME/helloworld
```

```
$ cd  $HOME/helloworld
```

Create and edit the required files. A source file, a configuration file, a Makefile.am file and a README file are needed: hello.c, configure.ac, Makefile.am, and README.

Create an empty README file with the following command, which is required by the GNU coding standard:

```
$ touch README
```

The other three files should look like this:

hello.c:

```
#include <stdio.h>
int main()
{
     printf("Hello, World! \n");
     return 0;
}
```

configure.ac:

```
AC_INIT(hello,0.1)
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CC
AC_CONFIG_FILES(Makefile)
AC_OUTPUT
```

Makefile.am:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

4.2.2.2 Initialize Cross-Development Environment

When you install EcoSDK-S2, it will create cross-toolchain environment setup script in the installation directory automatically. Before you develop applications using these tools, you should run the script first. The script's name starts like "environment-setup".

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.2.2.3 Create Configure Script

Use autoreconf command to create a configure script.

```
$ autoreconf
```

4.2.2.4 Cross-Compile

Use the command below to cross-compile the project, and it will automatically use the cross-compiler. The environment variable CONFIGURE_FLAGS will provide GNU configure with smallest parameters.

```
$ ./configure $ {CONFIGURE_FLAGS}
```

4.2.2.5 Build and Install

Use these two commands to generate the project and install them into the target directory.

```
$ make
```

```
$ make install DESTDIR=./tmp
```

You can verify the project with an easy command shown below. It will print out the file structure of the executable binaries.

```
$ file ./tmp/usr/local/bin/hello
```

4.2.2.6 Execute

You can run the project in EAS2A01. You can use command scp to copy your executable hello program into your target system through network. Please replace the <target_ip> with your target EAS2A01's IP address.

```
$ scp  ./tmp/usr/local/bin/hello root@<target_ip>:/home/root
```

Set this file as an executable file through serial port terminal and execute the program.

```
# chmod  u+x  hello
```

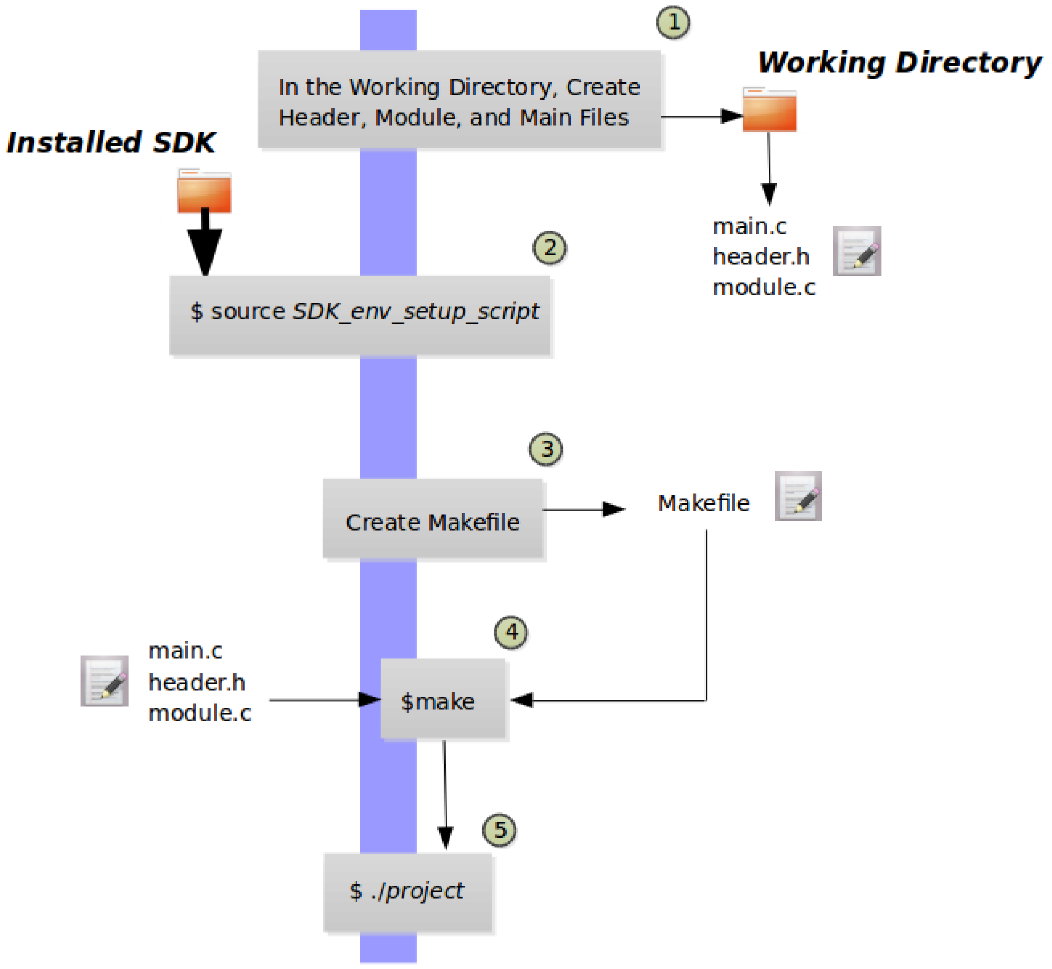
```
# ./hello
```

```
hello,world!
```

Now it's finished!

4.2.3 Use EcoSDK-S2 for Makefile-based Project

Running the EcoSDK-S2 cross-development environment configuration script will set up a series of environment variables related to cross-compilation. Makefile-based projects use and interact with these environment variables. These environment variables are subject to general rules. In this section we will introduce a simple Makefile-based development process and provide an example of how to use cross-compiled environment variables and Makefile variables during development. The figure below shows the workflow of a simple Makefile-based project.



4.2.3.1 Create Your Working Directory and Write a program

Create an empty directory and set it as your working directory.

```
$ mkdir  $HOME/helloworld
```

```
$ cd  $HOME/helloworld
```

Create and Edit the required files. We need to create three files: we will call functions from main.c, declare functions in module.h, and define functions in module.c.

main.c:

```
#include <module.h>
int main()
{
	sample_func();
	return 0;
}
```

module.h:

```
#ifdef MODULE_H

void sample_func()

#endif
```

module.c:

```
#include <stdio.h>
void sample_func()
{
	printf("Hello, World! \n");
}
```

4.2.3.2 Initialize Cross-Development Environment

When you install EcoSDK-S2, it will create cross-toolchain environment setup script in the installation directory automatically. Before you develop applications using these tools, you should run the script first. The script's name starts like "environment-setup".

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.2.3.3 Create Makefile

Makefile

```
all: main.o module.o
	${CC} main.o module.o -o target_bin
main.o: main.c module.h
	${CC} -I . -c main.c
module.o: module.c module.h
	${CC} -I . -c module.c
clean:
	rm -rf *.o
	rm target_bin
```

4.2.3.4 Build the project

Use command make to generate binary output file, the value of CC is set when you run SDK environment configuration.

```
$ make
```

You will see the cross-compiler it's using is the one you define CC as when you run installation script. You can verify the project with an easy command shown below. It will print out the file structure of the executable binaries.

```
$ file ./target_bin
```

4.2.3.5 Execute

You can run the project in EAS2A01. Use command scp to copy your executable hello program into your target system through network. Please replace the <target_ip> with your target EAS2A01's IP address.

```
$ scp  ./target_bin root@<target_ip>:/home/root
```

Set this file as an executable file through serial port terminal and execute the program.

```
# chmod  u+x  target_bin
```

```
# ./target_bin
```

```
hello,world!
```

Now it's done!

4.2.4 Use EcoSDK-S2 for ROS-based Project

ROS-based projects use the catkin instructions to build executable programs, and catkin will call CMake. Running EcoSDK-S2 cross-development environment configuration script will set up a series of environment variables related to cross-compilation. After that, running the catkin command will call the cross-compiler to compile the project. In this section, we will introduce a cross-compilation process for a simple ROS-based project.

4.2.4.1 Create Your Working Directory and Write a Program

Create a temporary directory to store ROS demo program fetched from the net.

```
$ mkdir  $HOME/tmp
```

```
$ cd $HOME/tmp
```

```
$ git clone https://github.com/ros/catkin_tutorials.git
```

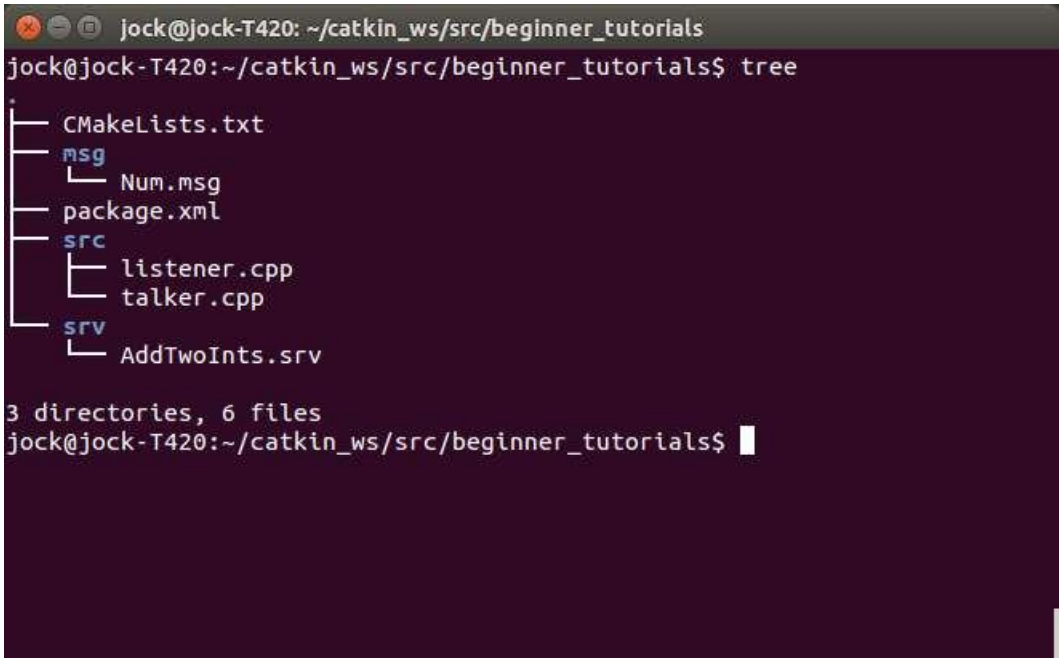
Create a directory as your working directory, and copy a software package from demo program into your working directory.

```
$ mkdir  -p  $HOME/catkin_ws/src
```

```
$ cd  $HOME/catkin_ws
```

```
$ cp -a $HOME/tmp/catkin_tutorials/create_package_pubsub/catkin_ws/src/beginner_tutorials src/
```

You can use command tree to view the files in directory /beginner_tutorials.



```
jock@jock-T420: ~/catkin_ws/src/beginner_tutorials
jock@jock-T420:~/catkin_ws/src/beginner_tutorials$ tree
.
├── CMakeLists.txt
├── msg
│   └── Num.msg
├── package.xml
├── src
│   ├── listener.cpp
│   └── talker.cpp
└── srv
    └── AddTwoInts.srv

3 directories, 6 files
jock@jock-T420:~/catkin_ws/src/beginner_tutorials$
```

4.2.4.2 Initialize Cross-Development Environment

When you install EcoSDK-S2, it will create cross-toolchain environment setup script in the installation directory automatically. Before you develop applications using these tools, you should run the script first. The script's name starts like "environment-setup".

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.2.4.3 Build the Project

Use command catkin_make to generate binary output file.

```
$ catkin_make
```

You will see the compiler it's using is cross-compiler. You can verify the project with the commands shown below. It will print out the file structure of the executable binaries.

```
$ file devel/lib/beginner_tutorials/talker
```

```
$ file devel/lib/beginner_tutorials/listener
```

4.2.4.4 Execute

You can run the project in EAS2A01. Use command scp to copy your executable talker and listener files of your program into your target system through network. Please replace the <target_ip> with your target EAS2A01's IP address.

```
$ scp  devel/lib/beginner_tutorials/talker  root@<target_ip>:/home/root
```

Set this file as an executable file through serial port terminal and execute the program.

```
# chmod  u+x  talker
```

```
# chmod  u+x  listener
```

```
# source  /opt/ros/indigo/setup.sh &
```

```
# roscore  &
```

```
# ./listener &
```

```
# ./talker &
```

Now it’s done!

4.3 Debug the Application

The gdbserver is pre-installed inside the EAS2A01 device, and the application developer can debug the application running on the target machine on the development host through the network. We will introduce the debug process of the hello-world program compiled directly in the above.

4.3.1 Initialize Cross-Development Environment

When you install EcoSDK-S2, it will create cross-toolchain environment setup script in the installation directory automatically. Before you develop applications using these tools, you should run the script first. The script’s name starts like “environment-setup”.

```
$ source  /<install path>/environment-setup-aarch64-fsl-linux
```

4.3.2 Recompile

If you want to use gbd debugger to debug the application, you need to compile the information you need to debug into the executable file.

```
$ $CC  -g  hello-world.c  -o  hello-world
```

4.3.3 Execute the Program

Use command scp to copy your executable hello-world program into your target system through network. Please replace the <target_ip> with your target EAS2A01’s IP address.

```
$ scp  hello-world  root@<target_ip>:/home/root
```

Use gdbserver to run the application in EAS2A01 through serial port terminal. <host_ip> is the development host’s IP, and <gdb_port> is a private port.

```
# gdbserver  <host_ip>:<gdb_port>  ./hello-world
```

4.3.4 Link Object Programs

Use the cross debugger to load the executable program on the development host and establish a link to the target. Replace <target_ip> with the IP address of the development host, replace <gdb_port> with the one just entered in the target machine. After that, you can debug the program with gdb just like it’s a local program. For more information on the gdb debugger, you can refer to the GDB official website.

```
$ $GDB  hello-world

(gdb) target remote <target_ip>:<gdb_port>

(gdb)
```

EcoSDK-S2 is a cross-development toolkit for developing applications generated by the Yocto Project. For more details, refer to Yocto Project Application Development and the Extensible Software Development Kit (eSDK).

Chapter 5 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 EAS2A01 Datasheet. We will explain in detail how to call the device driver functions in C programming language to access the interface in this chapter.

5.1 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.

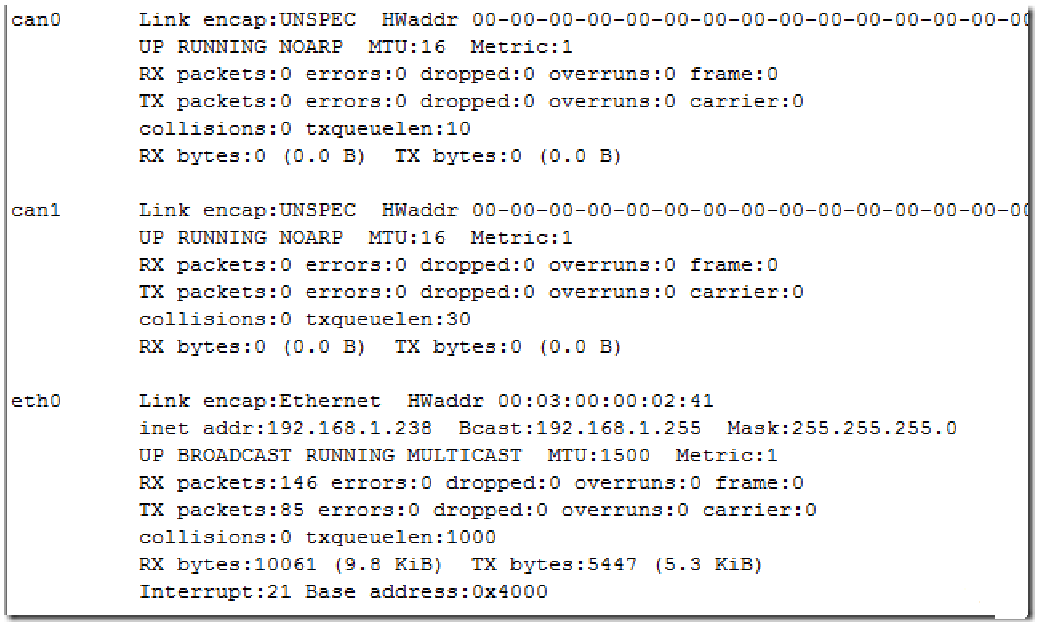
If you want to receive and transmit data through serial port, try this:
https://digilander.libero.it/robang/rubrica/serial.htm#CONTENTS

5.2 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 or https://elinux.org/CAN_Bus. 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. If you want to know more about SocketCAN, please read: SocketCAN - Wikipedia.

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
```

If you want to use C programming language to realize CAN data receive and transmit, you can refer to SocketCAN Documentation (Linux Kernel). If you want to know more about SocketCAN-based applications in Linux, refer to https://github.com/linux-can.

5.3 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 Network socket, which is eth0, in the operating system. If you want to know more about Network socket, refer to Network socket - Wikipedia. If you want to use C programming language to realize data transmission based on Socket, please refer to https://www.tutorialspoint.com/unix_sockets/index.htm or https://beej.us/guide/bgnet/.

5.4 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. For detailed information, refer to https://en.wikipedia.org/wiki/Framebuffer.

If you want to use C programming language to control display device based on framebuffer, refer to https://cmcenroe.me/2018/01/30/fbclock.html or https://gist.github.com/FredEckert/3425429.

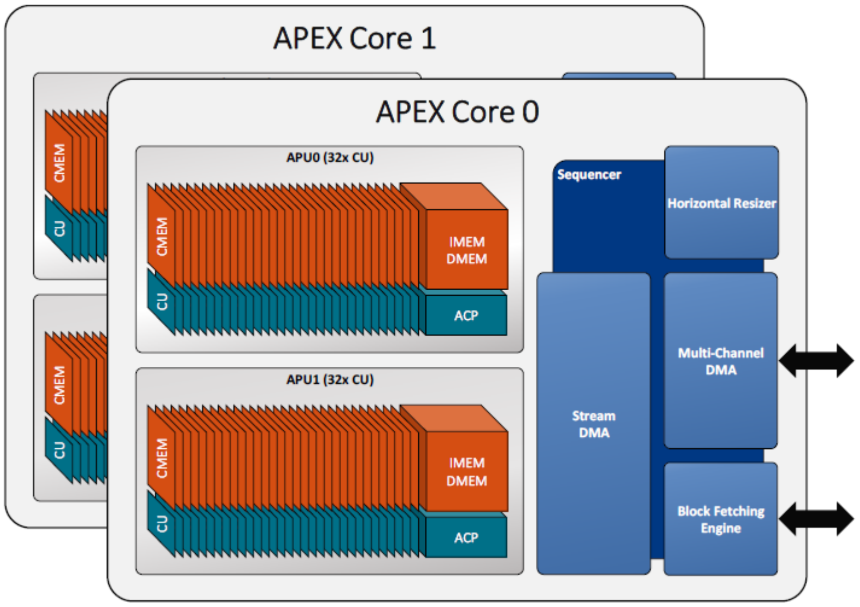
5.5 GMSL

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.

If you want to use GMSL to read video signal, you need to use Image Signal Process Unit (ISP) in S32V234. If you want to develop ISP applications, you need to use S32 Design Studio for Vision IDE and Vision SDK Software.

Chapter 6 Using APEX

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.