AES-S32V-NXP-G EVK Software Guide

Version 1.1

LIT# AES-S32V-NXP-G-Software-Guide-rev-1.1

Contents

1. Introduction

This document provides software startup user guide for Avnet S32v234 ADAS devlopment kit. It includes detail instructions of installing development environment, the Avnet AES-S32V-NXP-G EVK boot up, DDR test, and application demos.

1.1.Glossary

Term	Definition
SOM	System-on-Module
GPU	Graphics Processing Unit
ISP	Image Sensor Processor
DMS	Driver Monitoring System
IPU	Image Processing Unit
VIU	Video-In-Lite module
2D-ACE	Two Dimensional Animation and Compositing Engine
DCU	Display Control Unit, as a token of “2D-ACE”
ADAS	Advanced Driver Assistance System
APEX	APEX-CL Image Cognition Processor
RM	Reference Manual
BSP	Board Support Package
VSKD	Vision Software Development Kit

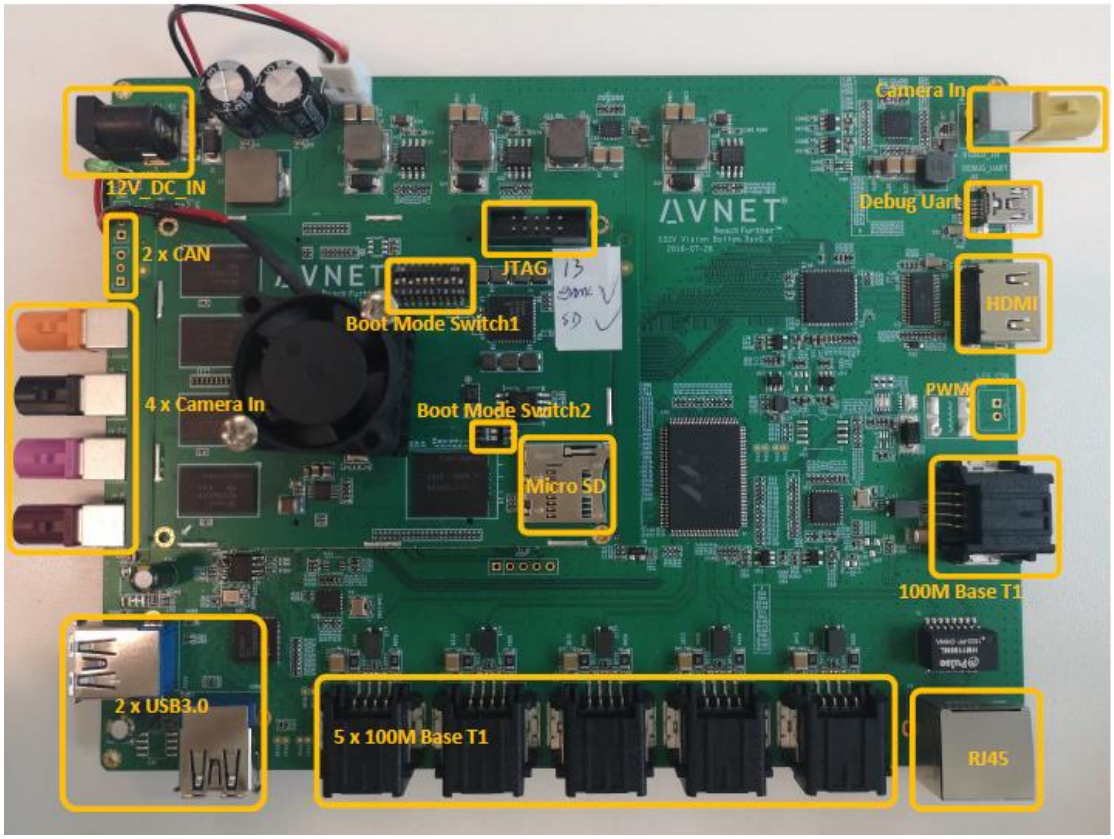
1.2.Additional Documentation

1.2.1. Additional information and documentation on NXP’s S32V234 MPU can be found at https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/vision-processor-for-front-and-surround-view-camera-machine-learning-and-sensor-fusion:S32V234

1.2.2. Additional information and documentation on AES-S32V-NXP-G (Core board) or AES-S32V-NXP-G (Carrier board) can be found at www.to/be/referenced under the appropriate product page.

1.2.3. Please Chapter 8, Getting Help and Support for further links.

2. AES-S32V-NXP-G EVK Introduction



Picture 1 – AES-S32V-NXP-G EVK

The AES-S32V-NXP-G EVK is a cost-competitive evaluation board and development platform engineered for high-performance, safe computation-intensive front vision, surround vision, and sensor fusion applications.

Based on the 32-bit Arm®Cortex®-A53 based S32V processors, the AES-S32V-NXP-G EVK has an efficient form factor while covering most of the uses cases available for the S32V234. The AES-S32V-NXP-G EVK is the recommended starting evaluation board for S32V.

Target Applications:

- Automotive Vision Systems
- Driver Monitoring Systems (DMS) and Occupant Monitoring Systems
- Fron View Camera
- Smart Rear View Camera
- Surround View & Sense Park Assist System

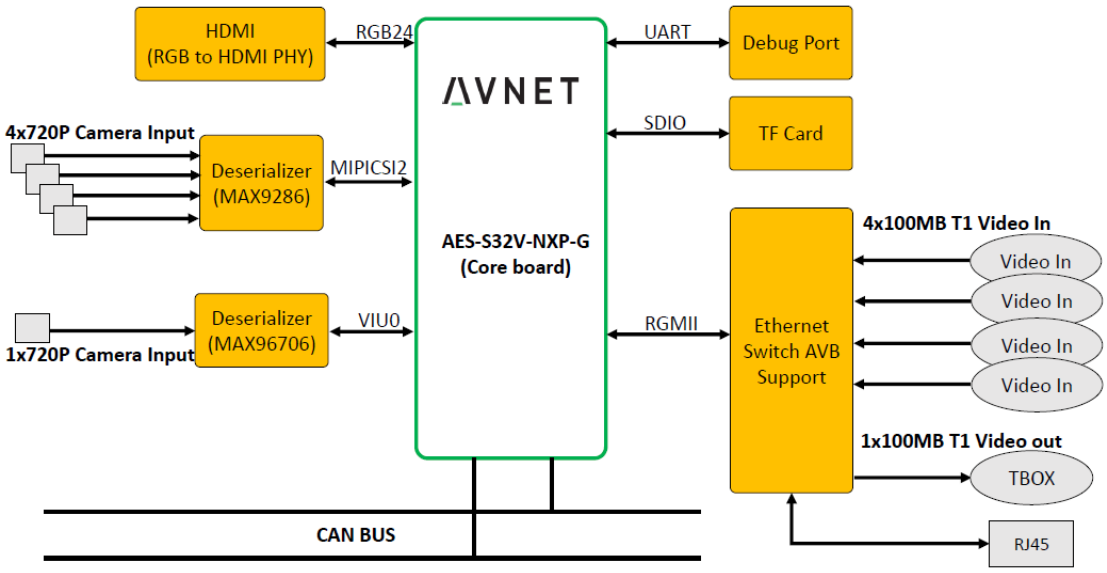


Figure 1 – AES-S32V-NXP-G EVK Function Block Diagram

2.1. AES-S32V-NXP-G EVK Delivery Package List:

✓ AES-S32V-NXP-G (Core board) & AES-S32V-NXP-G (Carrier board)

✓ 12V-DC power adapter

✓ 16GB TF Card (including Linux_BSP_18.0 images and demos)

✓ USB to Uart debug cable

Addiction list:

- HD Ethernet camera, 1280 x 720 @30fps, H.264 encode, support AVB
- HD Surround view camera, for use with MAX9286
- DMS camera, for use with MAX96706

2.2. Software Development Environment

- BSP and VSDK Version:
  Linux_BSP_18.0
  VSKD_1.2.0

- Host Development Environment:
  Ubuntu16.04
  S32DS_Vision_V2.0

3. Build the Linux BSP

3.1. BSP Introduction

The Linux BSP is a collection of source code that can be used to create U-Boot boot loader, Linux kernel image and a root filesystem for the supported boards.

All the steps described below have been run and validated on Ubuntu-16.04 LTS (native or through a virtual machine). It is then recommended to install a 16.04 LTS (or later) version of Ubuntu before going through the following sections.

Note: The latest BSP version port to AES-S32V-NXP-G EVK Board is BSP18.0.

Following sections introduce setup and build NXP official BSP18.0. If you want full functions support on AES-S32V-NXP-G EVK, please refer to chapter 8 *Getting Help and Support* for AES-S32V-NXP-G EVK BSP18.0 patches support.

3.1.1. U-Boot Bootloader

The Linux BSP delivery package contains the following U-Boot bootloader binary:
auto_linux_bsp18.0/<board>-/u-boot-<board>.s32
This bootloader supports SD/MMC/eMMC (u-boot.s32) and QSPI (u-boot.s32.qspi).
The u-boot.s32.qspi is generated by following steps in QSPI section in “Building the u-boot bootloader”.

3.1.2. Linux Kernel Image

This Linux BSP contains a pre-built kernel image based on the v4.14.34 of the Linux kernel. The kernel image and dtb file is located at the following path:
auto_linux_bsp18.0/<board>

3.1.3. Root File System

The package contains the following rootfs file system:
auto_linux_bsp18.0/<board>/fsl-image-[auto-]<soc>-<board>.tar.gz. The rootfs.tar.gz file system includes NXP specific libraries. Its contents can be mounted as a NFS share or can be stored on a boot media such as Secure Digital (SD) card.

3.2. Building the Linux BSP using Yocto

The Yocto Project(r) is a Linux Foundation collaborative open source project whose goal is to produce tools and processes that enable the creation of Linux distributions for embedded and IoT software that are independent of the underlying architecture of the embedded hardware.

3.2.1. Building default YOCTO

All the steps described below have been run and validated on Ubuntu-16.04 LTS (native or through a virtual machine). It is then recommended to install a 16.04 LTS (or later) version of Ubuntu before going through the following sections.
To get the BSP you need to have repo installed and its prerequisites. This only needs to be done once.
Install dependencies:

- python 2.x - 2.6 or newer:

```
sudo apt install python
```

- git 1.8.3 or newer:

```
sudo apt install git
```

- curl:

```
sudo apt install curl
```

To get the BSP you need to have repo installed. Please install it using the following commands (this only needs to be done once):

```
mkdir ~/bin
```

```
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo >
~/bin/repo
chmod a+x ~/bin/repo
export PATH=${PATH}:~/bin
```

Next, download the Yocto Project Environment into your directory:

```
mkdir fsl-auto-yocto-bsp
cd fsl-auto-yocto-bsp
repo init -u https://source.codeaurora.org/external/autobsps32/auto_yocto_bsp -b
hotfix/bsp18.0.hf1
repo sync
```

This will download the sources for the latest NXP Auto Linux BSP (from the branch alb/master), structured on top of the Yocto rocko release and upstream NXP QorIQ SDK.

The repository provides more manifest files, particularized for different use cases. The desired manifest file is selected by specifying it in the repo init command, using parameter -m <manifest file>, e.g:

```
repo init -u https://source.codeaurora.org/external/autobsps32/auto_yocto_bsp -b
hotfix/bsp18.0.hf1 -m <manifest file>.xml
```

The manifests are delivered in two formats: <manifest>.xml and <manifest>-bitbucket.xml. The second would allow fetching internal bitbucket development repositories.

Enter the directory fsl-auto-yocto-bsp and follow below instructions (available in the README file).

Note:

A Yocto build needs at least 50GB of free space and takes a lot of time (2 to 10 hours, depending on the system configuration). It is recommended to use a powerful system with many cores and a fast storage media (SSD is recommended).

If the machine chosen was s32v234evb, after the successful finalization of the build, the results will be placed in the build_s32v234evb/tmp/deploy/images/s32v234evb directory. Results for other machines will be placed in similar directories, but named according to the machine name.

This release includes support for:

- Machines: s32g275sim, s32v234bbmini, s32v234evb28899, s32v234evb, s32v234pcie, s32v234tmdp, s32v344sim, s32r45xsim, s32v234evbubuntu, s32v234sbc, s32v234ccpb
- Images: fsl-image-base, fsl-image-auto, fsl-image-ubuntu, fsl-image-ubuntu-base

To build the Linux BSP, please follow the steps:

1) First time setup

   ```
   ./sources/meta-alb/scripts/host-prepare.sh
   ```

2) Create a build directory in the SDK root with:

   ```
   source nxp-setup-alb.sh -m <machine>
   ```

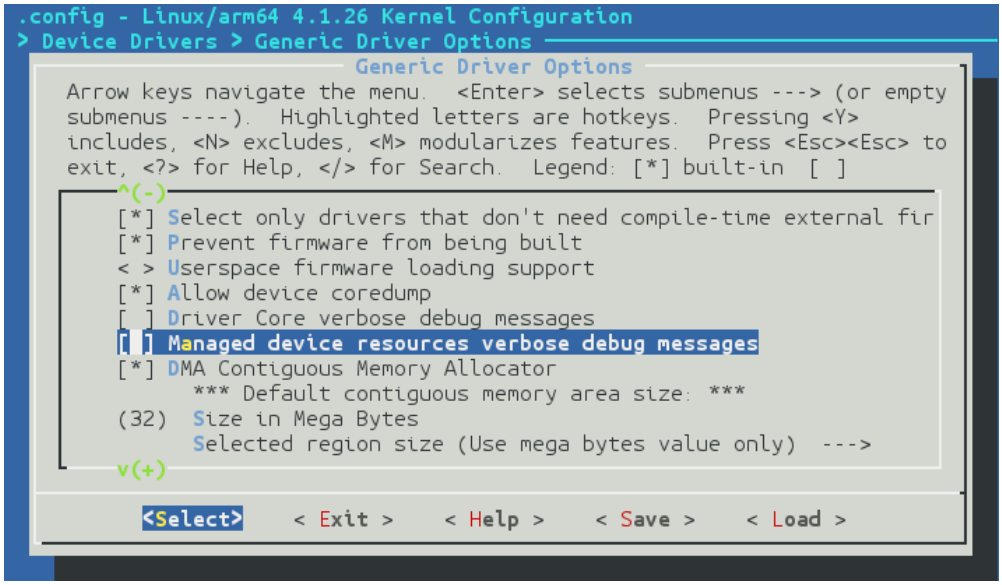
   For example, machine is s32v234evb:

```
source nxp-setup-alb.sh -m s32v234evb
```

3) [Optional] If you want to add GPU drivers (see chapter 'Setting Up the Graphics Drivers') you should set CMA (Contiguous Memory Allocator) size to 32MB using

```
bitbake virtual/kernel -c menuconfig
```

And from Device Drivers -> Generic Driver Options -> [*] DMA Contiguous Memory Allocator -> set (32) Size in Mega Bytes instead of 16 (default)



Save to .config and Exit.

4) When this is done, a "bitbake <imagename>", e.g.

```
bitbake fsl-image-auto
```

Would be enough to completely build u-boot, kernel, modules, and a rootfs ready to be deployed. Look for a build result in <builddirectory>/tmp/deploy/images/.

3.2.2. Additional Instructions for Developers

The following document assumes you are in <Yocto folder>/build_<machine>- build directory.

Locate the source for any packages, use the command bellow run from the build directory:

```
bitbake -e <package> | grep "^S="
```

This command will go through the package environment and find out information about the location of the source.

Example:

U-boot source:

```
bitbake -e u-boot | grep "^S="
S="<builddirectory>/tmp/work/s32v234evb-fsl-linux/u-boot-s32/2016.01-r0/git"
```

Linux source:

```
bitbake -e virtual/kernel | grep "^S="
S="<builddirectory>/tmp/work/s32v234evb-fsl-linux/linux-s32/4.14-r0/git"
```

Note: More additional Instructions for developer refer to capcher 2.1.2 of Auto_Linux_BSP_18.0_HF1_User_Manual.pdf
(https://nxp.flexnetoperations.com/control/frse/download?agree=Accept&element=10373557)

3.3. Manually building Linux BSP components

3.3.1. Obtaining a GCC toolchain for ARM 64-bit

This Linux BSP has been built and tested using an NXP patched version of the Linaro GCC 6.3.1 toolchain.
Starting from BSP16.0, the Linux BSP is validated using an NXP patched version of the Linaro GCC 6.3.1
2017.05 toolchain. The patched version of gcc is available on:
http://www.nxp.com/webapp/swlicensing/sso/downloadSoftware.sp?catid=S32-DS-3.0-EAR2
Also, the sources are available on CAF:
https://source.codeaurora.org/external/s32ds/compiler/gcc/
The unmodified version of gcc is available on:
https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz
Once you have downloaded the toolchain package, in order to install it, you just need to untar it in a directory of your choice.

3.3.2. Setting up and building U-boot bootloader

1) Downloading the u-boot bootloader source code

There are two ways of obtaining the source for this component, each described below.

- Cloning the GIT repository:
  In a Linux terminal window, type in the following commands, the <branch_name> is decided by yourself

```
git clone https://source.codeaurora.org/external/autobsps32/u-boot
cd u-boot
git checkout -b <branch_name> v2016.01_bsp18.0
ls
```

The contents of the u-boot source code should appear here.

- If you already have a clone of the repository, you can run the following commands in the root directory of the existing repository:

```
git fetch origin v2016.01_bsp18.0
git checkout -b <branch_name> v2016.01_bsp18.0
ls
```

The contents of the u-boot source code should appear here.

2) Building the u-boot bootloader

In the same Linux terminal window as above, type in the following commands:

```
make ARCH=aarch64
CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-linux-gnu-
s32v234evb_defconfig
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-linux-
gnu-
```

This command should generate the u-boot image with IVT header and Program data (u-boot.s32) that can be written onto the SD.

3.3.3. Setting up and building the Linux kernel

1) Downloading the Linux kernel source code

There are two ways of obtaining the source for this component, each described below.

- Cloning the GIT repository:

In a Linux terminal window, type in the following commands, the <branch_name> is decided by yourself

```
git clone https://source.codeaurora.org/external/autobsps32/linux
cd linux
git checkout -b <branch_name> v4.14.34_bsp18.0
ls
```

The contents of the u-boot source code should appear here.

- If you already have a clone of the repository, you can run the following commands in the root directory of the existing repository:

```
git fetch origin v4.14.78_bsp20.0
git checkout -b <branch_name> v4.14.34_bsp18.0
ls
```

The contents of the u-boot source code should appear here.

2) Building the Linux Kernel

In the same Linux terminal window as above, type in the following commands:

```
make ARCH=arm64
CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-linux-gnu-
s32v234_defconfig
make ARCH=arm64
CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-linux-gnu-
```

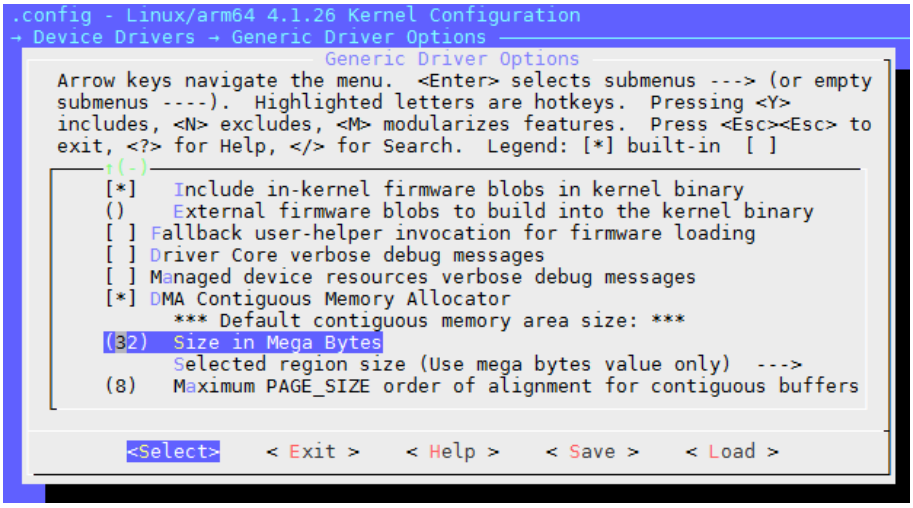
This command should generate the kernel binary (*Image*) in arch/arm64/boot and the board device tree blobs (e.g. *s32v234-evb.dtb, s32v234-pcie.dtb, s32v234-tmdp.dtb, s32v344-simulator.dtb, s32g275-simulator.dtb, s32r45x-simulator.dtb*...) in arch/arm64/boot/dts/freescale/.

Note:

- If you want to add GPU drivers (see chapter '*Setting Up the Graphics Drivers*' of Auto_Linux_BSP_18.0_HF1_User_Manual.pdf) you should set CMA (Contiguous Memory Allocator) size to 32MB using:

```
make ARCH=arm64
CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-
linux-gnu- s32v234_defconfig menuconfig
```

And from Device Drivers -> Generic Driver Options -> [*] DMA Contiguous Memory Allocator -> set (32) Size in Mega Bytes instead of 16 (default)



- Linux Kernel considers that its RAM starts from Kernel's ENTRY_POINT (0x80080000) ignoring the DDR below.

3.3.4. Setting up and building a root file system

Please use Yocto to generate it or use one the prebuilt binary rootfs-es delivered in the binary archive binaries_auto_linux_bsp18.0.tgz. Or use precompiled binaries from https://freescaleesd.flexnetoperations.com/337170/477/13422477/binaries_auto_linux_bsp18.0.tgz?ftpRequestID=7065323917&server=freescaleesd.flexnetoperations.com&dtm=DTM20190515025916MjU5ODM4ODY=&authparam=1557914356_be5f784de3dcb10000646041aa1ad60c&ext=.tgz

4. How to boot

This chapter describes the steps to prepare an SD/MMC card and boot up an Avnet AES-S32V-NXP-G EVK. The boot modes of the Avnet AES-S32V-NXP-G EVK are controlled by the boot configuration switches on the board.

4.1. Boot from TF card

The support SD class for SD are 4 and 10. An SD/MMC card reader is required. It will be used to transfer the boot loader and kernel images to initialize the partition table and copy the root file system.

The Linux kernel running on the Linux host will assign a device node to the SD/MMC card reader. The kernel might decide the device node name.

In this example it is assumed that the device assigned is /dev/sdx.

Note:

Make sure the device node is correct for the SD/MMC card! Otherwise, it may damage your operating system or data on your computer!

4.1.1. Setup a TF card

There are two ways to prepare a bootable SD Card:

1) To use Image Built from Yocto

   After successfully building Yocto, look for build result in <builddirectory>/tmp/deploy/images/s32v234evb

   The .sdcard format creates an image with all necessary partitions and loads the bootloader, kernel and rootfs to this image.

   Ensure that any partitions on the card are properly unmounted before writing the card image, or you may have a corrupted card image in the end. Also ensure to properly “sync” the filesystem before ejecting the card to ensure all data has been written.

   You can just low level copy the data on this file to the SD card device using dd as on the following command example:

   ```
   sudo dd if=<image name>.sdcard of=/dev/sdx bs=1M && sync
   ```

2) To manually copy BSP binaries into the SD Card

   a) Create partitions on the TF card

   ```
   sudo fdisk /dev/sdx
   ```

   Then type the following parameters (each followed by <ENTER>):

   ```
   d [repeat this until no partition is reported by the 'p' command ]
   n [create a new partition]
   p [create a primary partition]
   1 [the first partition]
   <enter>[using the default value will create a partition that starts at offset 2048]
   +255M [size of the actual partition = 255 MB]
   n [create a new partition]
   p [create a primary partition]
   2 [the second partition]
   <enter>[using the default value will create a partition that starts at offset 67584]
   <enter>[using the default value will create a partition that uses the remaining space on
   the card]
   t [set partition type]
   1 [partition #1]
   c [FAT32]
   t [set partition type]
   2 [partition #2]
   83 [Linux]
   w [ this writes the partition table to the medium and fdisk exits]
   ```

These newly created partitions need to be formatted to be usable. Firstly, remove and put TF Card back into its slot, then run the following commands to format the partitions:

```
sudo mkfs.vfat -n boot /dev/sdx1
sudo mkfs.ext3 -L rootfs /dev/sdx2
```

b) Copy the BSP binaries to TF Card

After format the partitions of TF card, replug card reader, there will be two partitions appeared:

- /media/<username>/boot
- /media/<username>/rootfs.

```
sudo dd if=u-boot.s32 of=/dev/sdx bs=512 seek=8 conv=fsync
sudo cp Image /media/<username>/boot/
sudo cp <dtb_file> /media/<username>//boot/<dtb_file>
sudo tar xf <rootfs> -C /media/<username>//rootfs
sync
```

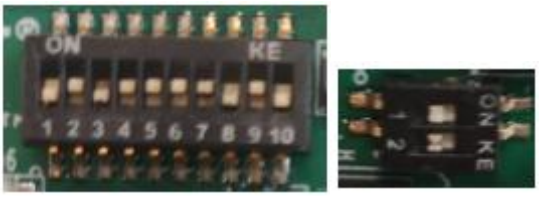
Note: For manual component building, the component names and paths should be updated accordingly to the corresponding build path. For example, for Linux use dtb file *<linux src>/arch/arm64/boot/dts/freescale/s32v234-evb.dtb* and image file *<linux src>/arch/arm64/boot/Image*

4.1.2. Setup Boot Configuration switches for TF Card Boot

There are two groups of DIP switches for boot mode configuration, Boot Mode Switch1 and Boot Mode Switch2, on the Avnet AES-S32V-NXP-G (Core board). Please set the DIP switch as following table for TF card booting:

BOOT MODE SWITCH1										BOOT MODE SWITCH2	
OFF	ON	OFF	ON	ON	ON	ON	OFF	ON	OFF	OFF	OFF
1	2	3	4	5	6	7	8	9	10	1	2

Table 1 – Boot Mode Switches Configuration for TF card Boot



Pitcture 2 – Boot Mode Switches Configuration for TF Card Boot

4.2. Boot from eMMC memory

AES-S32V-NXP-G EVK Board provides a 32GB eMMC memory as one of booting devices.

4.2.1. Setup eMMC memory with Avnet eMMC Setup Package

The Avnet eMMC Setup Package is used to burn images to eMMC memory by TF card. AES-S32V-NXP-G EVK boot into u-boot via TF Card first. The u-boot (u-boot-

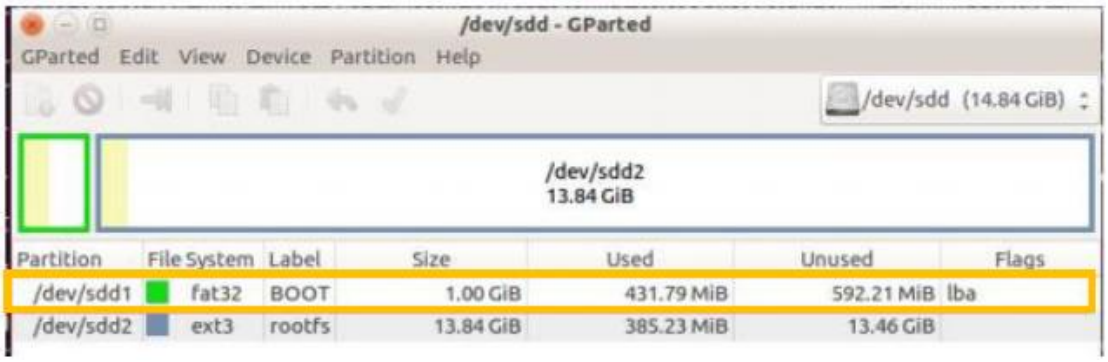
install.s32) burned on TF card is a specific version contains some commands for eMMC setup, so that we can create and format partitions on eMMC, then deliver linux BSP images to eMMC memory.

Notes:

- **The images for eMMC boot is the same images with SD booting.**
- **TF card is needed in setting up eMMC memory for:**
  - **a) To boot into u-boot via TF card**
  - **b) To store eMMC images, e.g. u-boot.s32, Image, s32v234.dtb, rootfs.ext3 in TF card**
- **Since S32v234 SoC contains only one SDHC controller, the TF card and eMMC cannot be access at the same time.**

Following sections show the steps to setup eMMC with Avnet eMMC Setup Package.

1) Setup TF Card for eMMC Burn
   a) Create a VFAT partition, at least 600M, and should be the first partition



   b) Copy imag, rootfs.ext4, s32v234-avnet2g.dtb, u-boot.s32, vsdk_runtime.tar.bz2 to VFAT partitions

```
cd bin/
cp image rootfs.ext4 s32v234-avnet2g.dtb u-boot.s32
vsdk_runtime.tar.bz2 /your/path/to/sd/vfat
```

   c) Burn u-boot-install.s32 to TF card

```
sudo dd if=./u-boot-install.s32 of=/dev/sdx bs=512 seek=8
conv=fsync
```

2) Set Boot Configuration Switches to TF Card Boot

There are two groups of DIP switches for boot mode configuration, Boot Mode Switch1 and Boot Mode Switch2, on AES-S32V-NXP-G (Core board). Please set the DIP switch as following table for TF card booting:

BOOT MODE SWITCH1										BOOT MODE SWITCH2	
OFF	ON	OFF	ON	ON	ON	ON	OFF	ON	OFF	OFF	OFF
1	2	3	4	5	6	7	8	9	10	1	2

Table 2 – Boot Mode Switches Configuration for TF card Boot

3) Boot AES-S32V-NXP-G EVK from TF card, press any key in 3 seconds downcounting to enter u-boot shell, then excute following commands:

```
emmcsetup 1
```

```
fdisk -c 0 1024 4096 4096
fatformat mmc 0:1
sdcardsetup 1
sdtoemmcboot mmc 0:1 0xC0000000 u-boot.s32
sdtoemmckernel mmc 0:1 0xC0000000 image
sdtoemmckernel mmc 0:1 0xC0000000 s32v234-avnet2g.dtb
sdtoemmckernel mmc 0:1 0xC0000000 vsdk_runtime.tar.bz2
sdtoemmcrootfs mmc 0:1 0xC0000000 rootfs.ext4
```

Debug log as following screenshot:

```
=> emmcsetup 1
SD is disabled. eMMC is active and can be used
=> fdisk -c 0 1024 4096 4096
fdisk is completed

partion #    size(MB)     block start #    block count    partition_Id (block s)
   1          1024           16384          2097152          0x0E
   2          4096          2113536         8388608          0x83
   3          4096         10502144         8388608          0x83
   4          5796         18890752        11870208          0x83
=> fatformat mmc 0:1
Start format MMC0 partition1 ...
Partition1: Start Address(0x4000), Size(0x200000)
size checking ...
Under 8G
write FAT info: 32
Fat size : 0x800
Erase FAT region.........................
Partition1 format complete.
=> sdcardsetup 1
eMMC is disabled. SD is active and can be used
=> sdtoemmcboot mmc 0:1 0xC0000000 u-boot.s32
reading u-boot.s32
368640 bytes read in 45 ms (7.8 MiB/s)
SD is disabled. eMMC is active and can be used

MMC write: dev # 0, block # 8, count 720 ... 720 blocks written: OK
eMMC is disabled. SD is active and can be used
do_sdtoemmcboot is success and file size is 720 bytes
=> sdtoemmckernel mmc 0:1 0xC0000000 image
reading image
8153096 bytes read in 694 ms (11.2 MiB/s)
SD is disabled. eMMC is active and can be used
writing image
8153096 bytes written
eMMC is disabled. SD is active and can be used
do_sdtoemmckernel is success and file size is 8153096 bytes
=> sdtoemmckernel mmc 0:1 0xC0000000 s32v234-avnet2g.dtb
reading s32v234-avnet2g.dtb
23705 bytes read in 16 ms (1.4 MiB/s)
SD is disabled. eMMC is active and can be used
writing s32v234-avnet2g.dtb
23705 bytes written
eMMC is disabled. SD is active and can be used
do_sdtoemmckernel is success and file size is 23705 bytes
=> sdtoemmckernel mmc 0:1 0xC0000000 vsdk_runtime.tar.bz2
reading vsdk_runtime.tar.bz2
22655780 bytes read in 1910 ms (11.3 MiB/s)
SD is disabled. eMMC is active and can be used
writing vsdk_runtime.tar.bz2
22655780 bytes written
eMMC is disabled. SD is active and can be used
do_sdtoemmckernel is success and file size is 22655780 bytes
=> sdtoemmcrootfs mmc 0:1 0xC0000000 rootfs.ext4
reading rootfs.ext4
```

```
419430400 bytes read in 35128 ms (11.4 MiB/s)
SD is disabled. eMMC is active and can be used

MMC write: dev # 0, block # 2113536, count 819200 ... 819200 blocks written: OK
eMMC is disabled. SD is active and can be used
do_sdtoemmcrootfs is success and file size is 819200 bytes
=>
```

Tips:

a) Switch between eMMC and TF card access in u-boot

- To access eMMC

```
emmcsetup 1
```

- To access TF card

```
sdcardsetup 1
```

b) List files in VFAT partition

```
fatls mmc 0:1 /
```

c) List files in ext4 partition

```
ext4ls mmc 0:2 /
```

4) Set Boot Mode Switches to eMMC Boot

BOOT MODE SWITCH1										BOOT MODE SWITCH2	
OFF	ON	OFF	ON	ON	ON	ON	ON	ON	OFF	OFF	ON
1	2	3	4	5	6	7	8	9	10	1	2

Table 3 – Boot Mode Switches Configuration for eMMC Boot

5) Boot from eMMC, menually install vsdk_runtime package

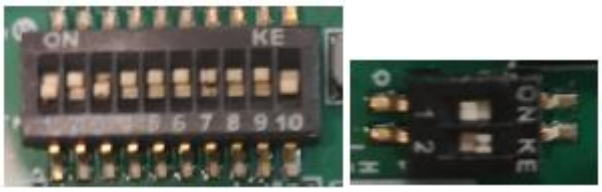
```
mount /dev/mmcblk0p1 /media/
cd /media
tar -jxf vsdk_runtime.tar.bz2 -C /
sync
cd /home/root/
umount /dev/mmcblk0p1
```

4.2.2. Setup Boot Configurarion switch for eMMC Boot

There are two groups of DIP switches for boot mode configuration, Boot Mode Switch1 and Boot Mode Switch2, on AES-S32V-NXP-G (Core board). Please set the DIP switch as following table for eMMC booting:

BOOT MODE SWITCH1										BOOT MODE SWITCH2	
OFF	ON	OFF	ON	ON	ON	ON	ON	ON	OFF	OFF	ON
1	2	3	4	5	6	7	8	9	10	1	2

Table 4 – Boot Mode Switches Configuration for eMMC Boot



Pitcture 3 – Boot Mode Switches Configuration for eMMC Boot

5. Vision SDK & Setup Environment

This chapter will introduce the Vision SDK of NXP and setup development environment of VSDK

5.1. Vision SDK

The Vision Software Development Kit (VisionSDK) for S32V2 provides a comprehensive SW enablement environment for NXP/AMPs 2nd generation of vision processors, S32V2xx. The S32V2xx family of devices is designed to support computation intensive applications for image processing.

The VisionSDK provides comprehensive abstraction of the powerfull accelerators for image signal processing (ISP) and massive parallel computing (APEX). Well documented APIs help to fully exploit the HW capabilities and create applications easily.

Users of the VisionSDK can be inspired by the broad offering of demo applications bundled with the package coming with full source code.

The VisionSDK is topped with its sophisticated build system which makes it easy to create new projects. For convenient code development NXP offers the eclipse based Design Studio for Vision.

The latest version of VSDK is SW32V23-VSDK001-RTM-1.3.0 which can be download from NXP official website:

https://nxp.flexnetoperations.com/control/frse/download?element=10595057

More detail information please refer to *Vision SDK 1.3 User Guide.pdf*

This VisionSDK is compatible with Avnet S32v234-EVK, but additional patches are required if specific cameras need to be support. More technical supports please refer to chapter 8.

5.2.Suggested Develop Enviroment & Tools

- Host operation system: PC windows10, for S32 Design Studio for Vision
- VirtualBox: Ubuntu16.04, for compilation of Linux BSP and vsdk drivers
- Debug tool: Putty, for UART/SSH console debug
- File transmission tool: WinSCP, for file transmission between host and AES-S32V-NXP-G EVK Board

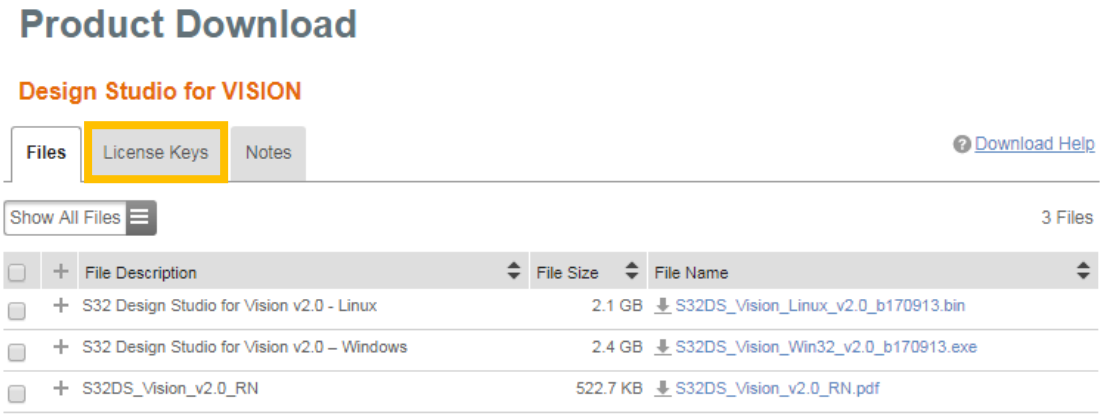
Picture 4 – AES-S32V-NXP-G EVK Board Develop Interfaces

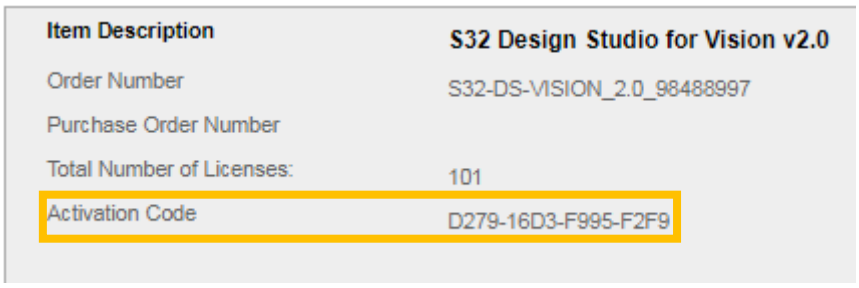
5.3. S32 Design Studio for Vision (S32DS)

The S32 Design Studio for Vision IDE is a complimentary integrated development environment for S32V234 processor which is a high-performance automotive processor designed to support safe computation-intensive applications in the area of vision and sensor fusion. S32 Design Studio for Vision enables editing, compiling and debugging of designs. Based on, open-source software including Eclipse IDE, GNU Compiler Collection (GCC) and GNU Debugger (GDB). S32 Design Studio for Vision IDE offers designers a straightforward development tool with no code-size limitations and visual programming tools that make vision accelerators programming easy. NXP software, along with the S32 Design Studio IDE provides a comprehensive enablement environment that reduces development time.

5.3.1. Installation of S32DS

1) Download S32 Design Studio for Vision v2.0 from NXP webside (need to register and login): https://nxp.flexnetoperations.com/control/frse/download?agree=Accept&element=9231187
2) Install follow instructions in *S32 Design Studio for Vision v2.0 Installation Manual* (https://www.nxp.com/docs/en/user-guide/S32DS_VISION_2.0_UM.pdf)
3) Note:
    - Choose the online verification method
    - The activation code can be found here:

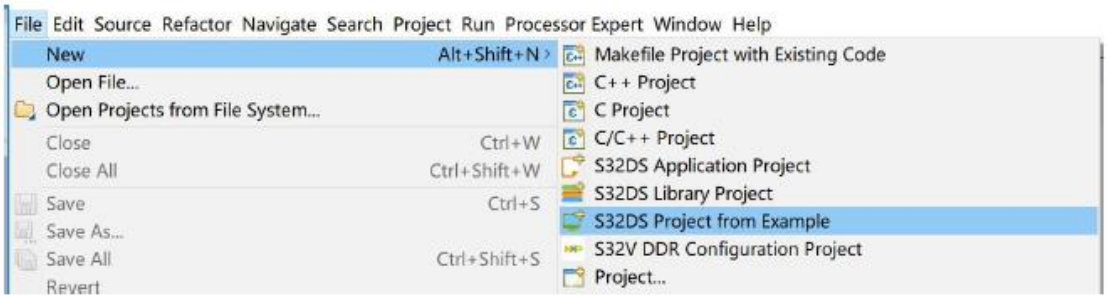




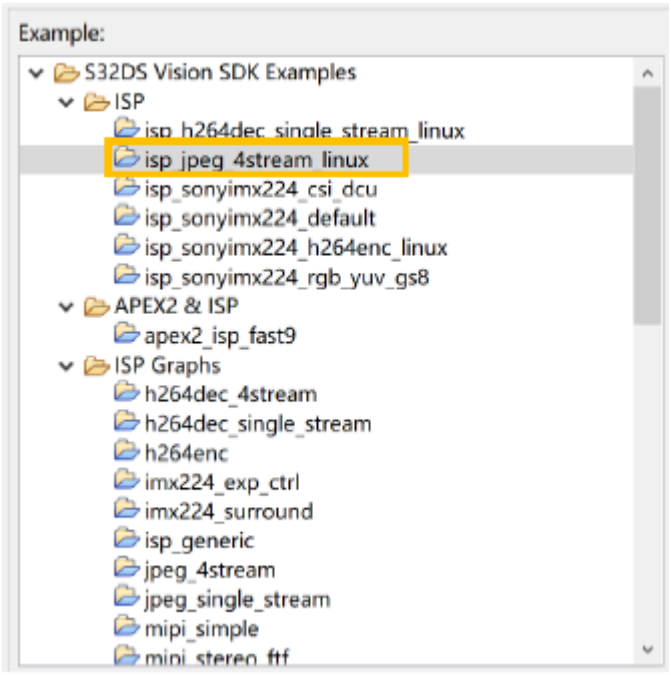
5.3.2. Run Example Project in S32DS

To help developpers learn S32DS quickly, there are few example projects in S32DS. Follow the steps below to create, compile and run examples.

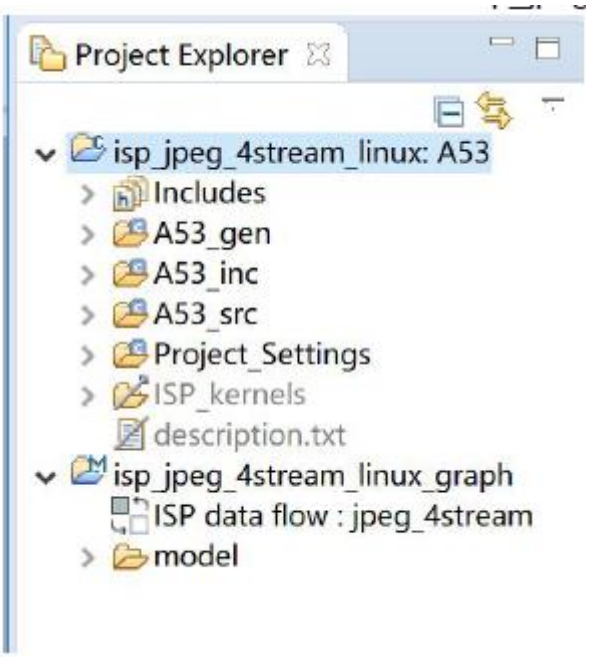
1) File -> New -> S32DS Project from Example



2) Choose Example, e.g. *isp_jpeg_4stream_linux*

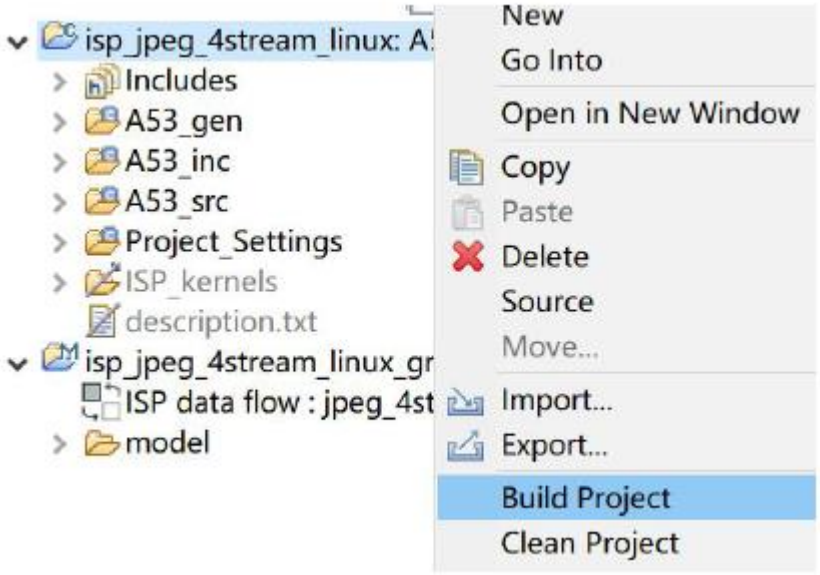


After creating *isp_jpeg_4stream_linux,* two projects, *isp_jpeg_4stream_linux_graph* and *isp_jpeg_4stream_linux,* will appear in Project Explorer

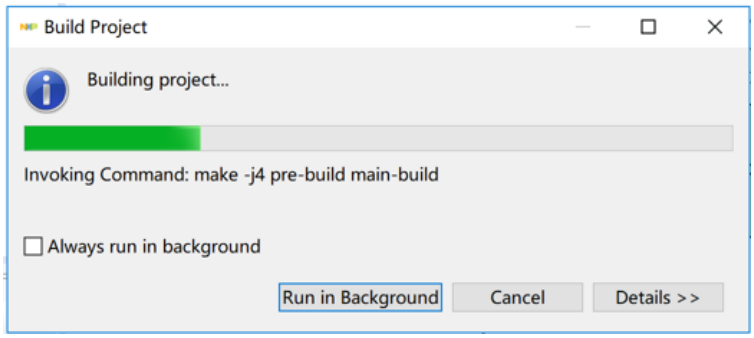


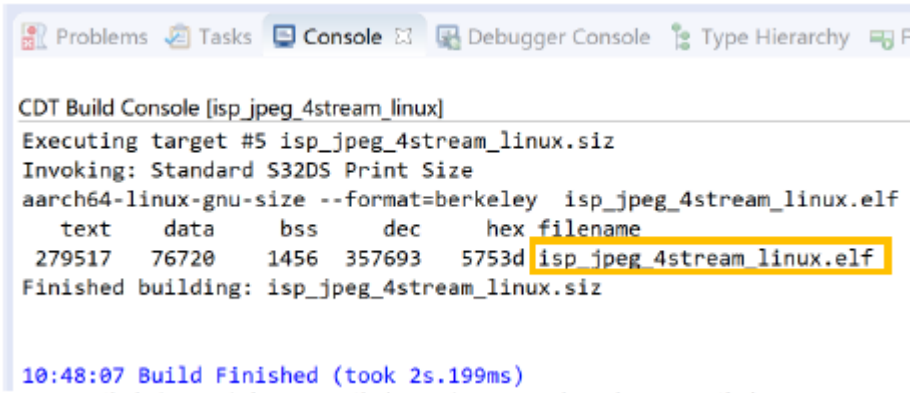
3) Compile *isp_jpeg_4stream_linux*

Right click isp_jpeg_4stream_linux -> Build Project



Check the console, build success if showing the same as following screenshot:





4) Run example on AES-S32V-NXP-G EVK Board

   a. Copy test dependencies used by example to AES-S32V-NXP-G EVK Board's /home/root directory, the test dependencies are in the following path:

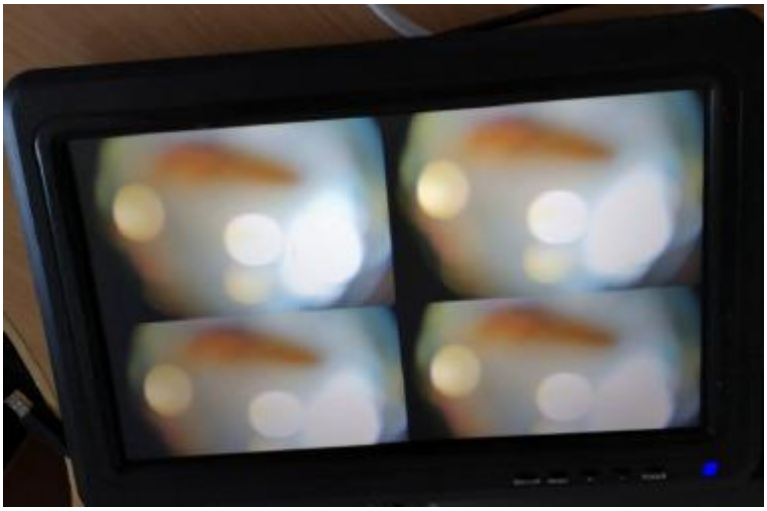
\path\to\your\S32DS\install\directory\S32DS\s32v234_sdk\demos\data

   b. Copy .elf binary file generated from step 3) to AES-S32V-NXP-G EVK Board's /home/root directory

   c. Change permission and run the binary

```
chmod +x ./isp_jpeg_4stream_linux.elf
./isp_jpeg_4stream_linux.elf
```

   d. If successfully executed, monitor will show the following picture:



Pitcture 5 – isp_jpeg_4stream_linux demo output

5) Some notes about isp_jpeg_4stream_linux_graph project

isp_jpeg_4stream_linux_graph is an ISP graph project, developpers edit the ISP processing flow block, and use Visual Graph Tool (VGT) in S32DS to generate code to isp_jpeg_4stream_linux/A53_gen/. More details on how to use VGT, please refer to:

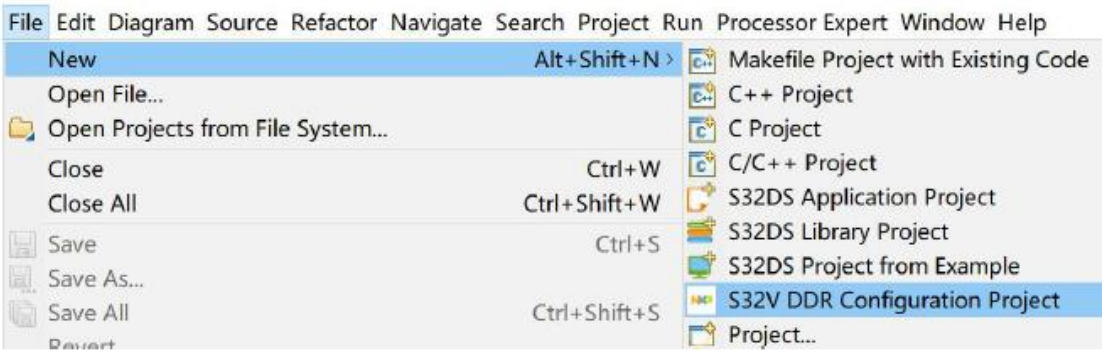
\path\to\your\S32DS\install\directory\S32DS\help\resources\howto\ISP_graph_tool_in_depth_tutorial_091217.pdf

6. DDR Configuration and Validation

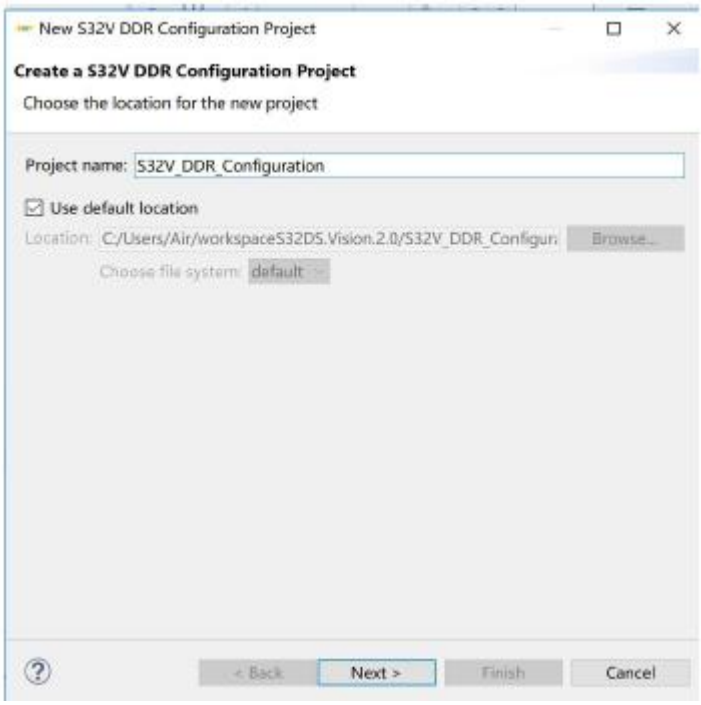
The S32 Design Studio for Vision integrates DDR Configuration and Validation Tool (DDR CVT). This chapter will introduce how to use S32DS DDR CVT to create a DDR configuration and validation project for AES-S32V-NXP-G EVK Board. More details about DDR CVT please refer to *\path\to\your\S32DS\install\directory\S32DS\help\pdf\pdf_ddr.pdf*

6.1. Create DDR Configuration project

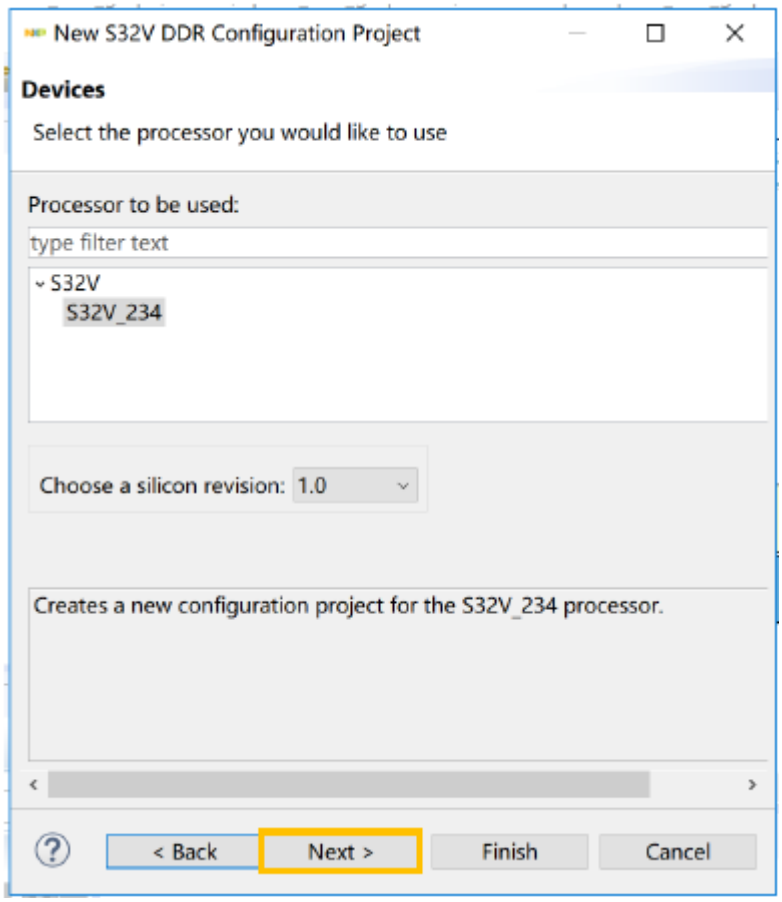
1) File -> New -> S32V DDR Configuration Project:



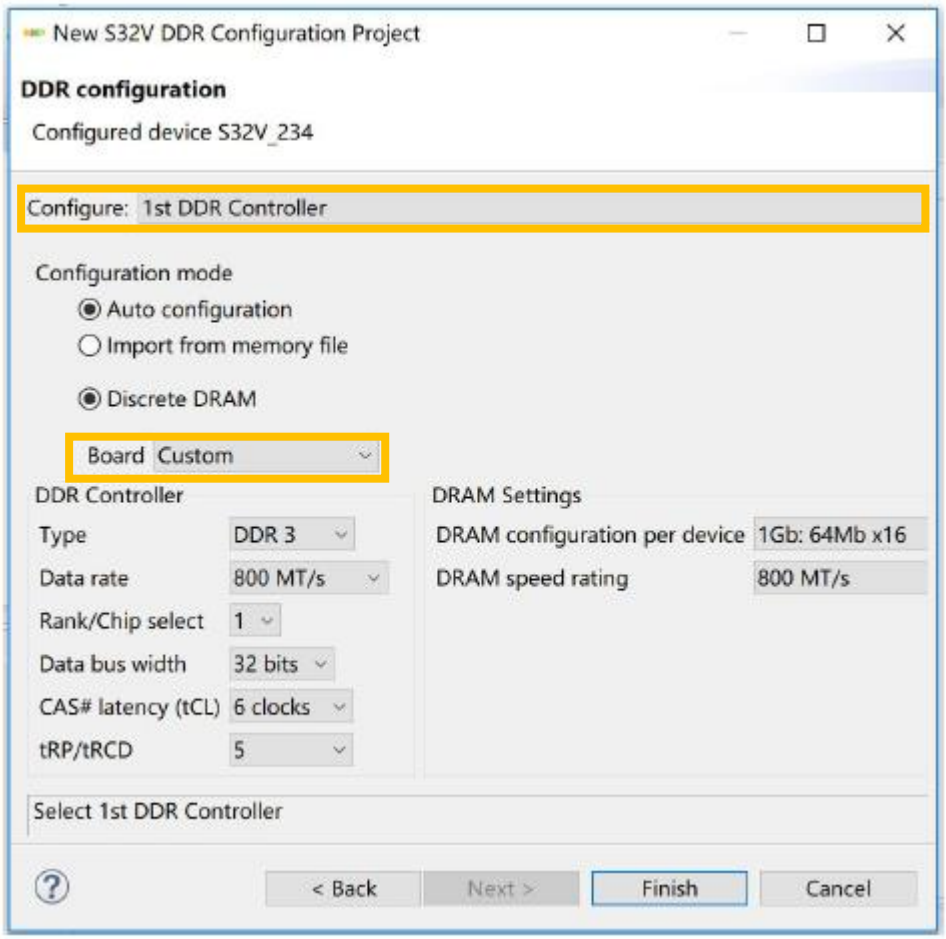
2) Enter the project name



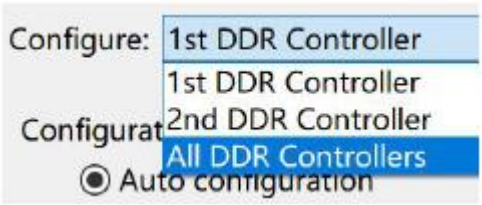
3) Choose processor part number, S32v_234 -> Next:



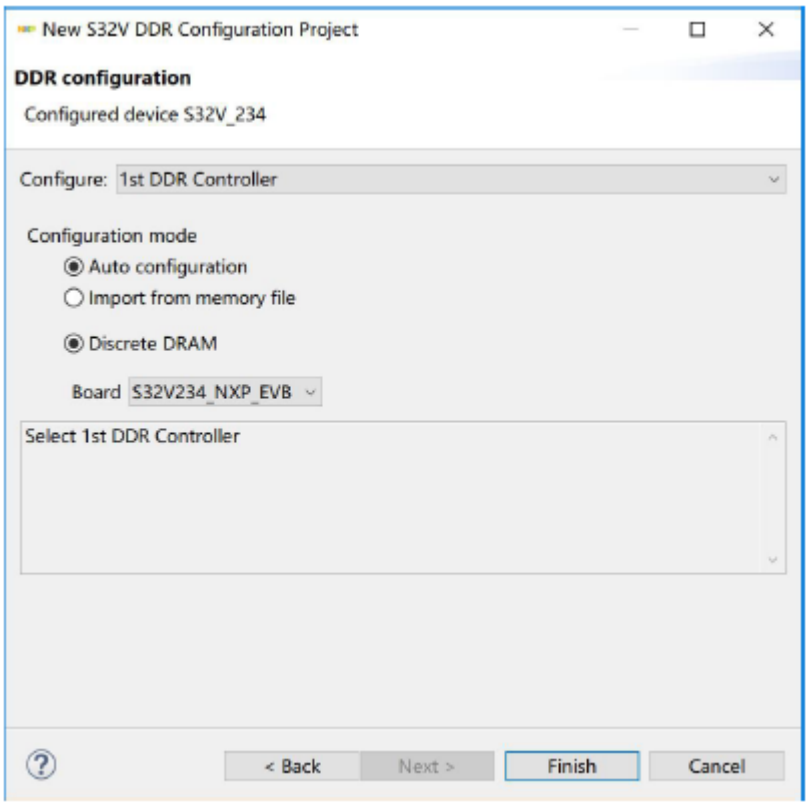
4) Click “Next”, showing the following window:



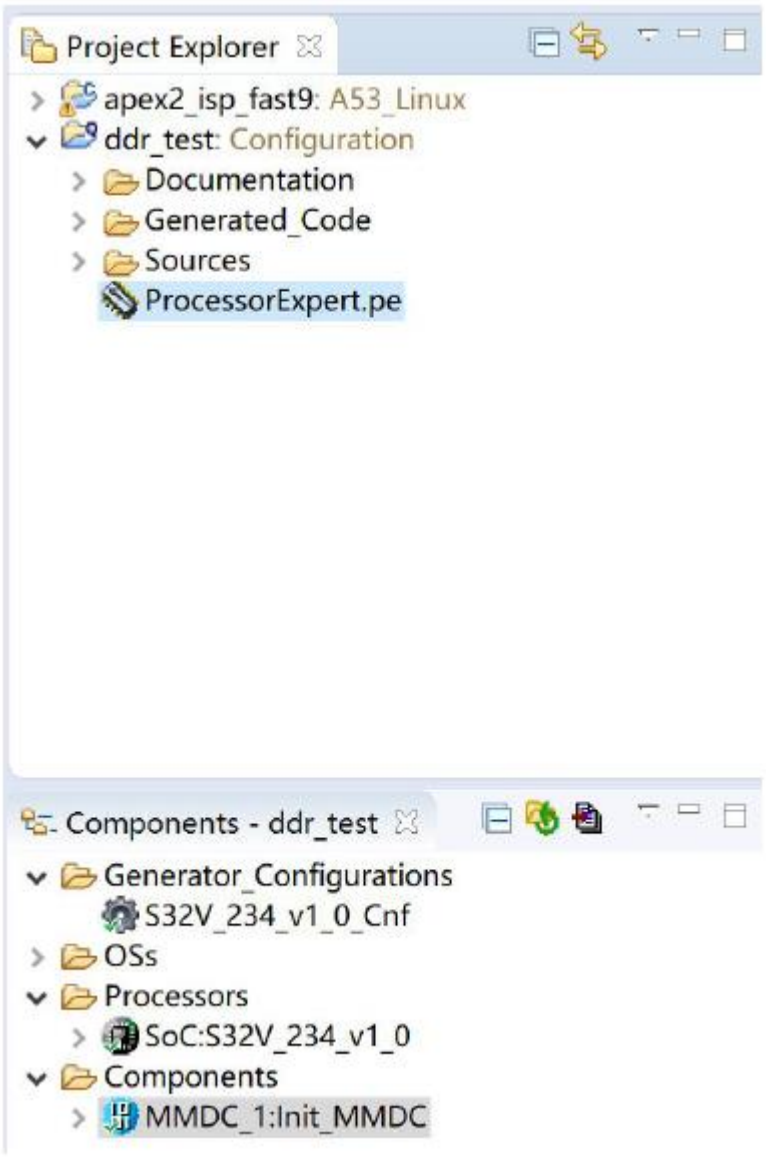
- Choose the DDR controller to configure, MMDC_0 is the 1st DDR Controller, MMDC_1 is the 2nd DDR Controller, from “Configure” drop-down menu:



- Choose the target board from the “Board” drop-down menu, the configuration of AES-S32V-NXP-G EVK Board is the same with *S32v234_NXP_EVB*, so we just select *S32v234_NXP_EVB*. If developers need to define a new configuration, just select “*Custom*”:



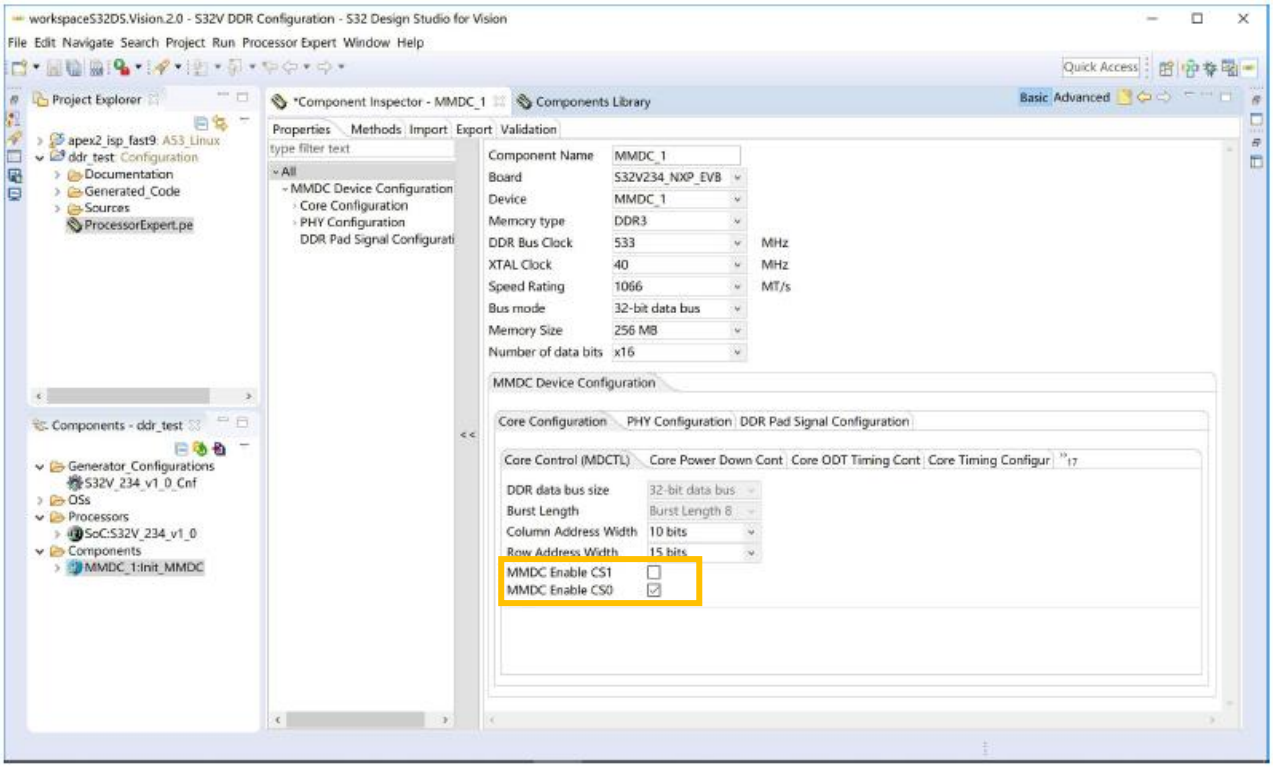
5) Click “*Finish*” -> double click “*ProcessorExpert.pe*”, showing the “*Components*” window:



6.2. Configure DDR Controller

Here we take MMDC_0 as an example, the configuration method is the same on MMDC_1.

Double click "*MMDC_0*", showing the "*Component Inspector*" window, then configure the MMDC_0 according to your DDR's datasheet. AES-S32V-NXP-G EVK Board uses the default configuration of S32v234_NXP_EVB.



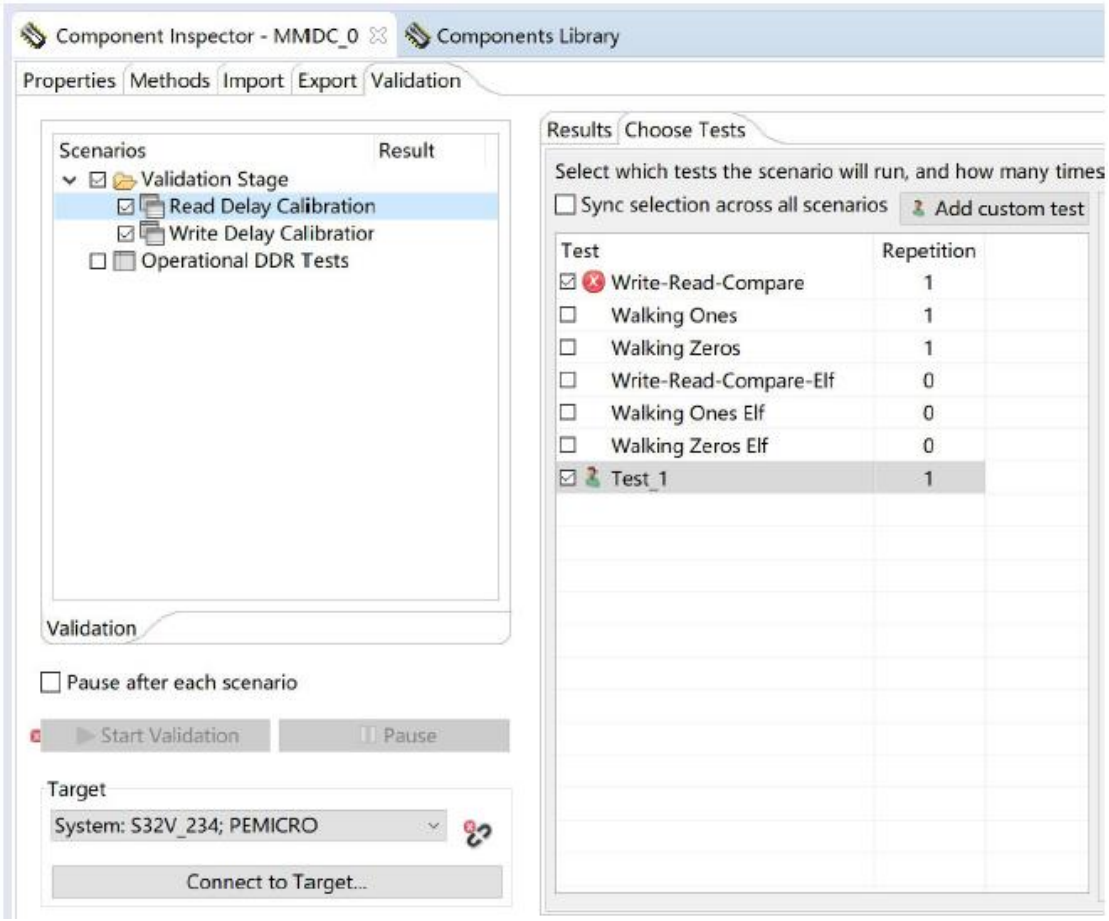
Note:

1) If you choose "*Custom*" Configure, the *MMDC Enable CSx* is unchoosed in *MDCTL* tab, we need to select one menually, or this would cause the "*Start Validation*" button unclickable after success connected:

2) When "*Memory Size*" is modified, the "*Row Address Width*" in "*MDCTL*" tab will changed
3) Actually, all the value of initial regsiters can be configure or check in the "Configuration Registers" tab:
   Window -> Show View -> Others -> Processor Expert -> Configuration Registers

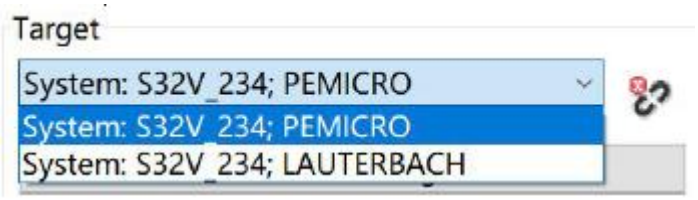
6.3. DDR Validation

6) Choose "Validation" tab:



a. Choose test case in "*Scenarios*", configure the test method and repeat times in "Choose Tests"

b. Choose target test tool:



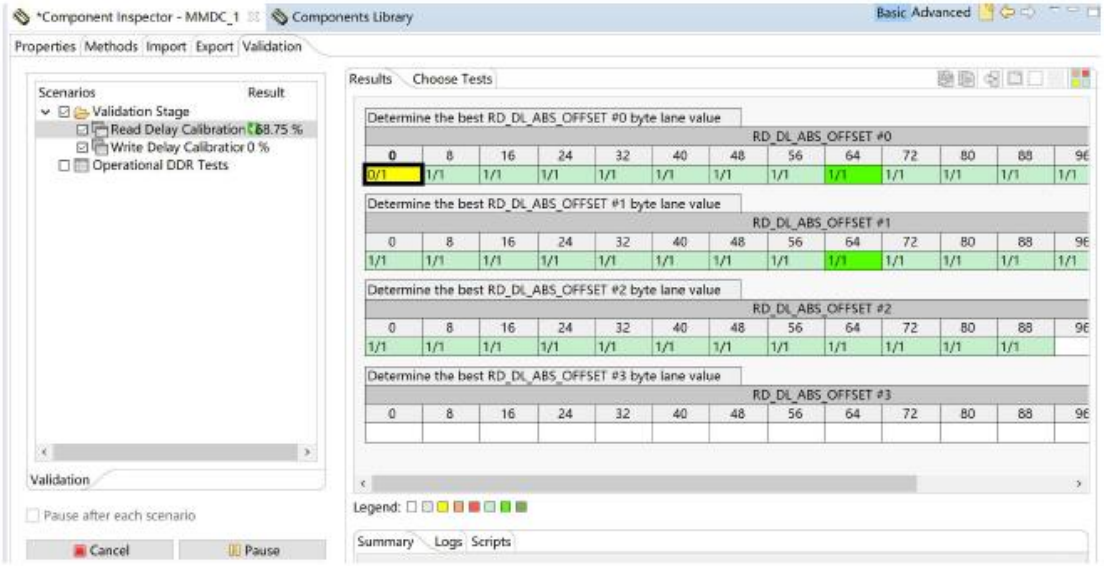
Note: The USB Multilink Universal is an easy-to-use debug and programming interface which allows the PC to communicate with a target processor through a USB port. More information please refer to https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/universal-multilink-development-interface:UMultilink?lang=en&lang_cd=en&

Picture 6 – The USB Multilink Universal

c. Click “Connect to target”

Note: ignore the disconnect log in console, it is connected successfully if the connect icon’s Red Cross disappeared.

d. Click “Start Validation”, the “Result” tab shows validation process:



Note:

i. The yellow and green color of test result in each byte lane mean that validation executed success. Green means that current testing parameter is good, yellow means that current testing parameter is wrong. If red color appear on the byte lane, that means connection error or validation error, check the “Log” to find error position.

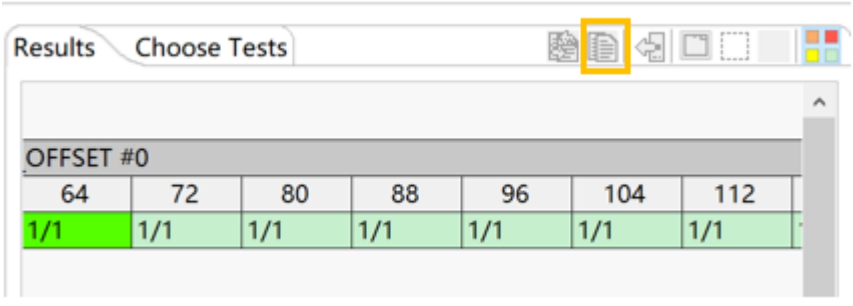
ii. The Configuration Registers are changed in the validation process

iii. Execute MMDC_0 validation first, then execute MMDC_1 validation. Each validation needs more then one hour.

6.4. Export Result

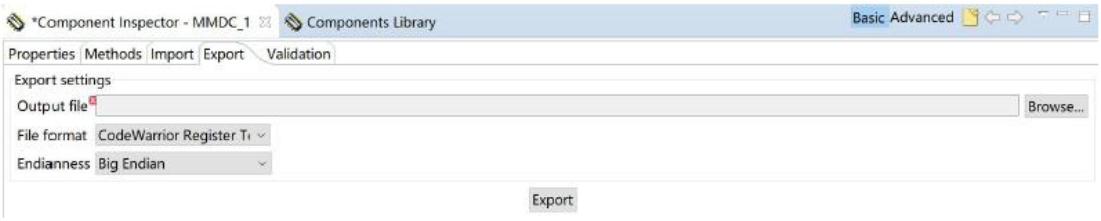
After validation, the init value in Configuration Registers is optimized register parameter, here has three methods to export the results:

Method 1. Click the small icon in red block, the results can be exported in html/CSV/Excel format. This report shows detail information of each “WR/RD_DL_ABS_OFFSET#”



Method 2. In the “Export” tab, some of the register parameters can be exported.

The report only contains register parameters of 0x40036xxx and 0x4006Cxxx



Method 3. Right click “ProcessorExport.pe” -> Generate Processor Expert Code. The register_defines_MMDC_x.h and SetupDDR3_x.c will be generated in “Generated_Code” folder, the .h file contains all result value of MMDC and configuration registers.

6.5. Modify u-boot with validation results

According to register_defines_MMDC_x.h, modify following files in u-boot:
u-boot/arch/arm/cpu/armv8/s32v234/ddr3.c;
u-boot/arch/arm/include/asm/arch-s32v234/ddr3.h

```
/* Set the amount of DRAM */
/* Set DQS settings based on board type */
#if defined(CONFIG_S32V234EVB_29288) || defined(CONFIG_S32V234BBMINI_29406)
#define MMDC_MDASP_MODULE0_VALUE          0x0000007F  /* 1 GB memory */
#define MMDC_MPRDDLCTL_MODULE0_VALUE      0x44434442  /* Read delay line offsets */
#define MMDC_MPWRDLCTL_MODULE0_VALUE      0x42434041  /* Write delay line offsets */
#define MMDC_MPDGCTRL0_MODULE0_VALUE      0x41470138  /* Read DQS gating control 0 */
#define MMDC_MPDGCTRL1_MODULE0_VALUE      0x012E0135  /* Read DQS gating control 1 */

#define MMDC_MDASP_MODULE1_VALUE          0x0000007F  /* 1 GB memory */
#define MMDC_MPRDDLCTL_MODULE1_VALUE      0x44434442  /* Read delay line offsets */
#define MMDC_MPWRDLCTL_MODULE1_VALUE      0x3F3F4242  /* Write delay line offsets */
#define MMDC_MPDGCTRL0_MODULE1_VALUE      0x414B0138  /* Read DQS gating control 0 */
#define MMDC_MPDGCTRL1_MODULE1_VALUE      0x01320142  /* Read DQS gating control 1 */
```

```
/* Set the amount of DRAM */
/* Set DQS settings based on board type */
switch(module) {
case MMDC0:
        writel(MMDC_MDASP_MODULE0_VALUE, mmdc_addr + MMDC_MDASP);
        writel(MMDC_MPRDDLCTL_MODULE0_VALUE, mmdc_addr + MMDC_MPRDDLCTL);
        writel(MMDC_MPWRDLCTL_MODULE0_VALUE, mmdc_addr + MMDC_MPWRDLCTL);
        writel(MMDC_MPDGCTRL0_MODULE0_VALUE, mmdc_addr + MMDC_MPDGCTRL0);
        writel(MMDC_MPDGCTRL1_MODULE0_VALUE, mmdc_addr + MMDC_MPDGCTRL1);
break;
case MMDC1:
        writel(MMDC_MDASP_MODULE1_VALUE, mmdc_addr + MMDC_MDASP);
        writel(MMDC_MPRDDLCTL_MODULE1_VALUE, mmdc_addr + MMDC_MPRDDLCTL);
        writel(MMDC_MPWRDLCTL_MODULE1_VALUE, mmdc_addr + MMDC_MPWRDLCTL);
        writel(MMDC_MPDGCTRL0_MODULE1_VALUE, mmdc_addr + MMDC_MPDGCTRL0);
        writel(MMDC_MPDGCTRL1_MODULE1_VALUE, mmdc_addr + MMDC_MPDGCTRL1);
        break;
}

writel(MMDC_MDRWD_VALUE, mmdc_addr + MMDC_MDRWD);
writel(MMDC_MDPDC_VALUE, mmdc_addr + MMDC_MDPDC);
writel(MMDC_MDREF_VALUE, mmdc_addr + MMDC_MDREF);
writel(MMDC_MPODTCTRL_VALUE, mmdc_addr + MMDC_MPODTCTRL);
writel(MMDC_MDSCR_RESET_VALUE, mmdc_addr + MMDC_MDSCR);
```

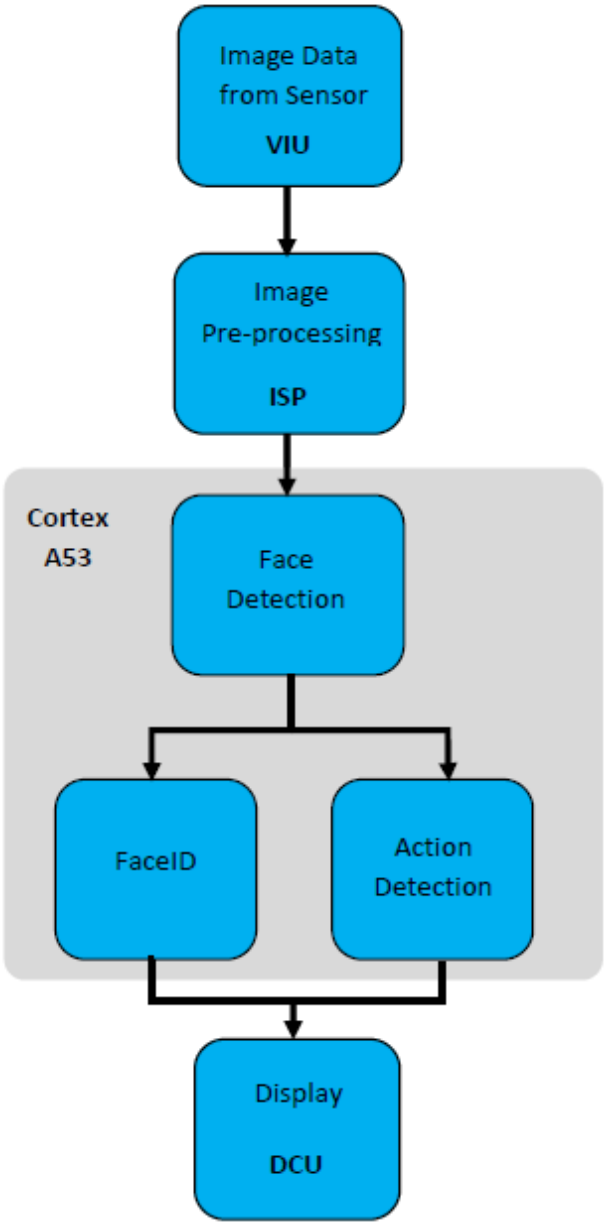
7. AES-S32V-NXP-G ADAS Demos

7.1. DMS

The Driver Monitoring System (DMS), also known as Driver Attention Monitor, is a vehicle safety system. The system used a camera place on the steering column which is capable of eye tracking, via infrared LED detectors. If the driver is not paying attention to the road ahead, or taking some actions, such as drinking, smoking or calling, the system will warn the driver by flashing lights, warning sounds.

The Avnet's DMS demo features:

1) Face Detection
2) Face ID
3) Action Detection
    - Drink
    - Smoke
    - Phone Call
    - Eye Closed
    - Attention (head Orientation)
    - Yawn
4) Display information through HDMI in English/Chinese
5) Transmit information through CAN/UART



Feature 2 – AES-S32V-NXP-G DMS Demo Block Diagram

Picture 7 – AES-S32V-NXP-G DMS demo output

7.2.Surround View

TODO

7.3. AVB

TODO

8. Getting Help and Support

If additional support is required, Avnet has many avenues to search depending on your needs. For general question regarding AES-S32V-NXP-G (Core board) and Carrier Card or accessories, please visit our website at www.to/be/referenced. Here you can find documentation, technical specifications, videos and tutorials, reference designs and other support.

Detailed questions regarding AES-S32V-NXP-G EVK hardware design, software application development, using NXP tools, training and other topics can be posted on the AES-S32V-NXP-G Support Forums at www.to/be/referenced. Avnet's technical support team monitors the forum during normal business hours.

Those interested in customer-specific options on AES-S32V-NXP-G EVK can send inquiries to to-be-referenced@avnet.com.

9. Release Note

Version	Release date	Author	Note
Rev-1.0	16-May-2019	Air.Xu	First version
Rev-1.1	20-Sept-2019	Air.Xu	Changed part number