AF Driver porting introduction

Agenda

- AF Driver Porting Guide
  - File list
  - Lens configuration
  - I2C bus number configuration
  - Kernel configuration
  - VCM Power configuration
  - HAL configuration
  - Selinux policy configuration
- Log Analysis
  - Init log
  - Uninit log
- Debug Check Points

Applicable Version

- Android P
- Android Q

AF Driver Porting Guide

- ❑ File list
- ❑ Lens configuration
- ❑ I2C bus number configuration
- ❑ Kernel configuration
- ❑ VCM Power configuration
- ❑ HAL configuration
- ❑ Selinux policy configure

File list

```
device\mediatek\<ProjectName>\ProjectConfig.mk
kernel-4.X\arch\arm64\configs\XX_defconfig
```

Config

```
Kernel-4.X\drivers\misc\mediatek\dws\[$platform]\[$project].dws
```

dws

```
kernel-4.X\drivers\misc\mediatek\lens\main:
kernel-4.X\drivers\misc\mediatek\lens\main2:
kernel-4.X\drivers\misc\mediatek\lens\main3:
kernel-4.X\drivers\misc\mediatek\lens\sub:
kernel-4.X\drivers\misc\mediatek\lens\sub2:

kernel-4.X\drivers\misc\mediatek\lens\main\inc\lens_info.h
kernel-4.X\drivers\misc\mediatek\lens\main\inc\lens_list.h
kernel-4.X\drivers\misc\mediatek\lens\main\main_lens.c
kernel-4.X\drivers\misc\mediatek\lens\main\common\xxxxxxaf\xxxxxxaf.c
```

Kernel-4.X
4.9 or 4.14

hal

```
vendor\mediatek\proprietary\custom \[$Platform]\ hal\inc\camera_custom_lens.h
vendor\mediatek\proprietary\custom \[$Platform]\ hal\lens\src\lenslist.cpp
vendor\mediatek\proprietary\custom\[$Platform]\hal\lens\xxxxxxaf\lens_para_xxxxxxAF.cpp
or
Vendo\mediatek\proprietary\custom\[$Platform]\hal\imgsensor\ver1\xxxx_mipi_raw\Scene_Capture\AF.cpp
```

File list

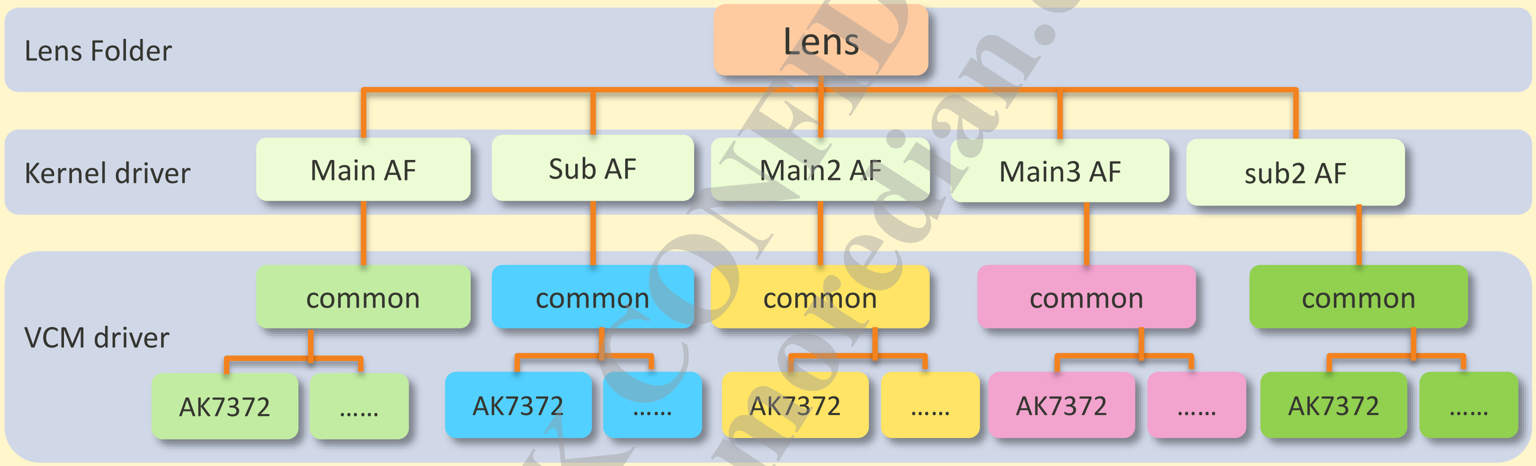
```
alps/device/mediatek/[$platform]/init.[$platform].rc
alps/device/mediatek/sepolicy/basic/non_plat/device.te
alps/device/mediatek/sepolicy/basic/non_plat/mediaserver.te
alps/device/mediatek/sepolicy/basic/non_plat/file_contexts
alps/device/mediatek/sepolicy/basic/non_plat/factory.te
alps/device/mediatek/sepolicy/basic/non_plat/atci_service.te
alps/device/mediatek/sepolicy/basic/non_plat/mtk_hal_camera.te
alps/device/mediatek/sepolicy/basic/non_plat/cameraserver.te
alps/device/mediatekprojects/[$project]/init.project.rc
```

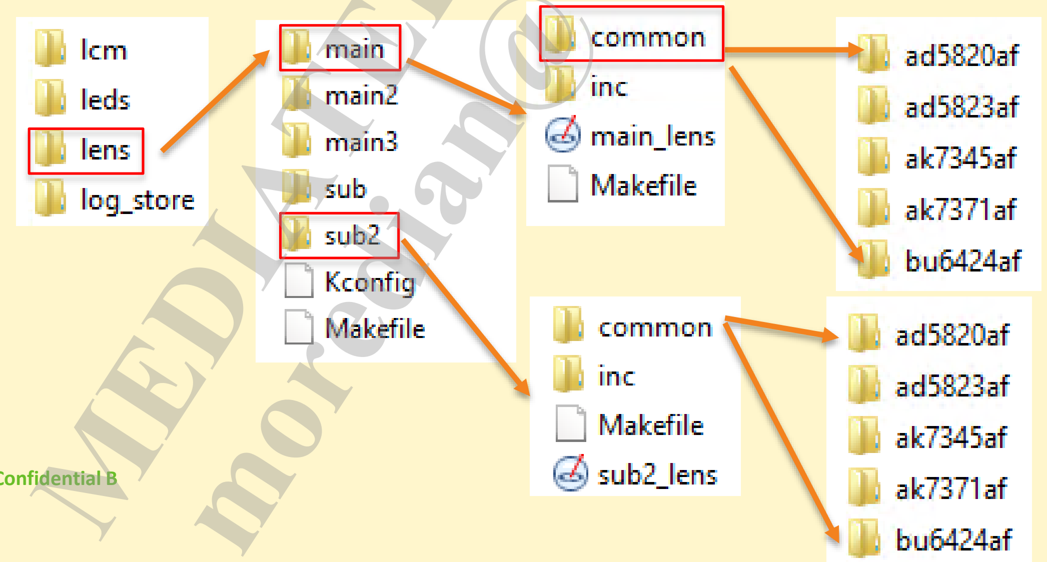
selinux policy configure

```
alps/kernel-4.XX/arch/arm64/boot/dts/mediatek/[$Platform].dts
alps/kernel-4.14/arch/arm64/boot/dts/mediatek/mtXXXX.dtsi
alps/kernel-4.14/arch/arm64/boot/dts/mediatek/[$Project].dts
```

dts

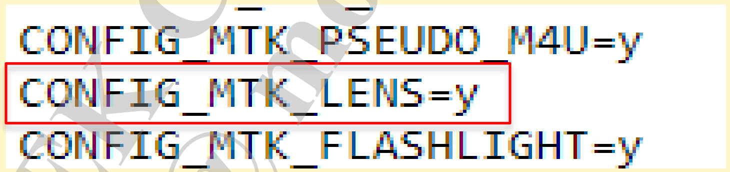
Architecture





Step 1
Lens configuration

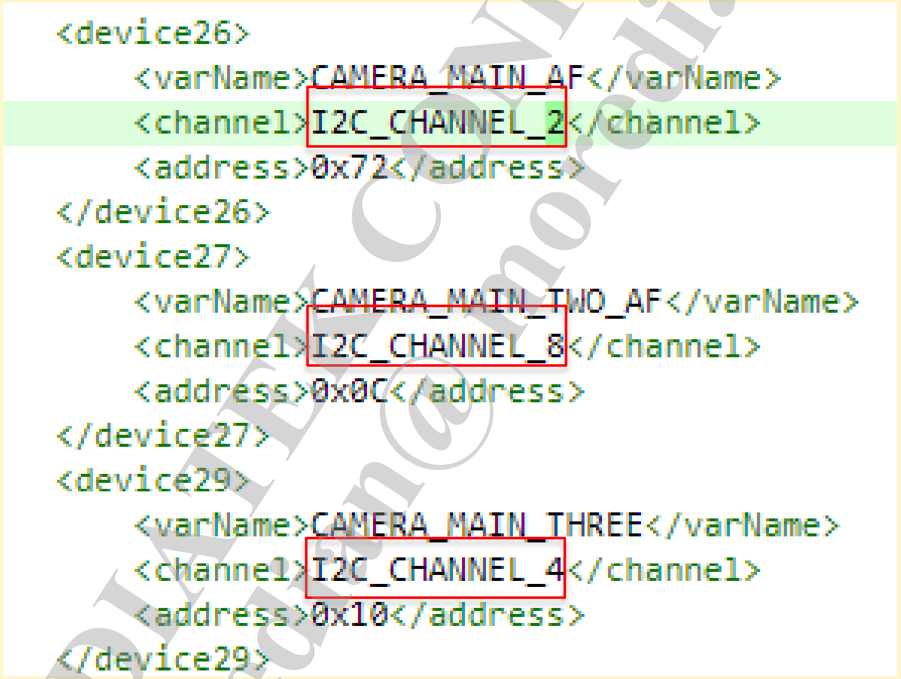
- It’s no need to config the lens’s name in ProjectConfig.mk file
- Config MTK_Lens in file of [$Project_ name]_defconfig (/kernel-4.14/arch/arm64/configs/)



```
CONFIG_MTK_PSEUDO_M4U=y
CONFIG_MTK_LENS=y
CONFIG_MTK_FLASHLIGHT=y
```

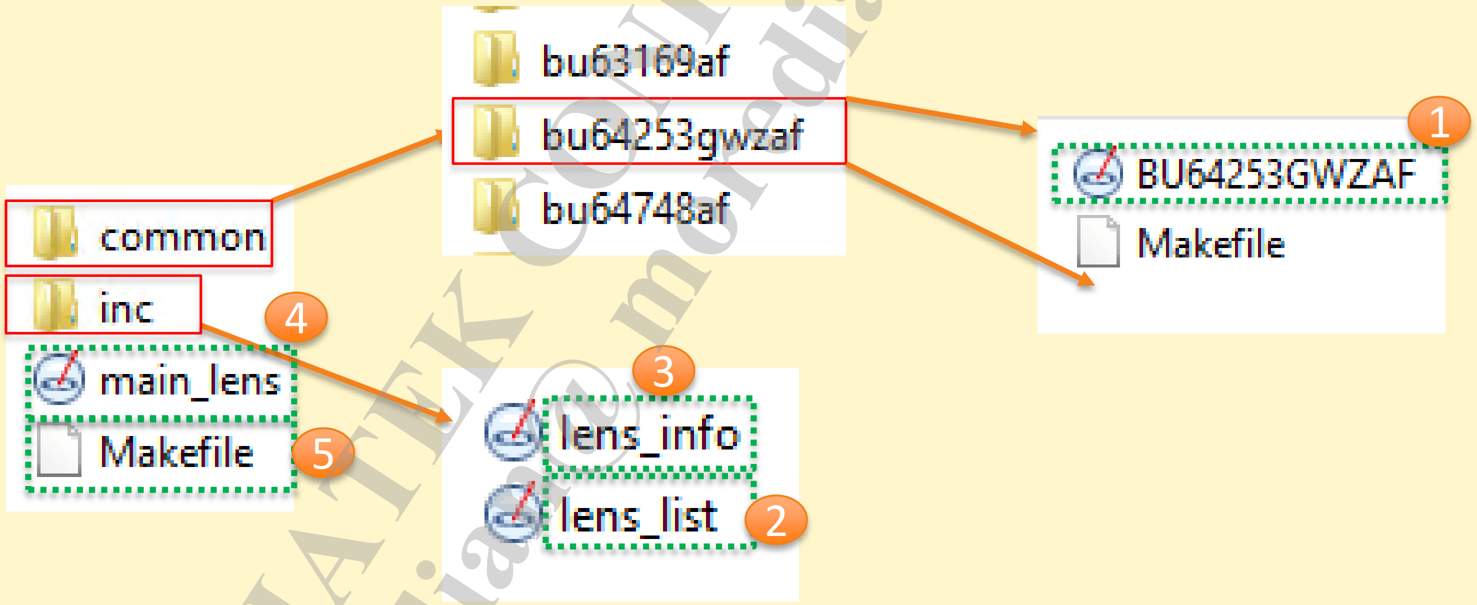
Step 2
I2C bus number configuration

- Configure I2C bus number in [$project].dws



```
<device26>
    <varName>CAMERA_MAIN_AF</varName>
    <channel>I2C_CHANNEL_2</channel>
    <address>0x72</address>
</device26>
<device27>
    <varName>CAMERA_MAIN_TWO_AF</varName>
    <channel>I2C_CHANNEL_8</channel>
    <address>0x0C</address>
</device27>
<device29>
    <varName>CAMERA_MAIN_THREE</varName>
    <channel>I2C_CHANNEL_4</channel>
    <address>0x10</address>
</device29>
```

Step 3
Kernel configuration

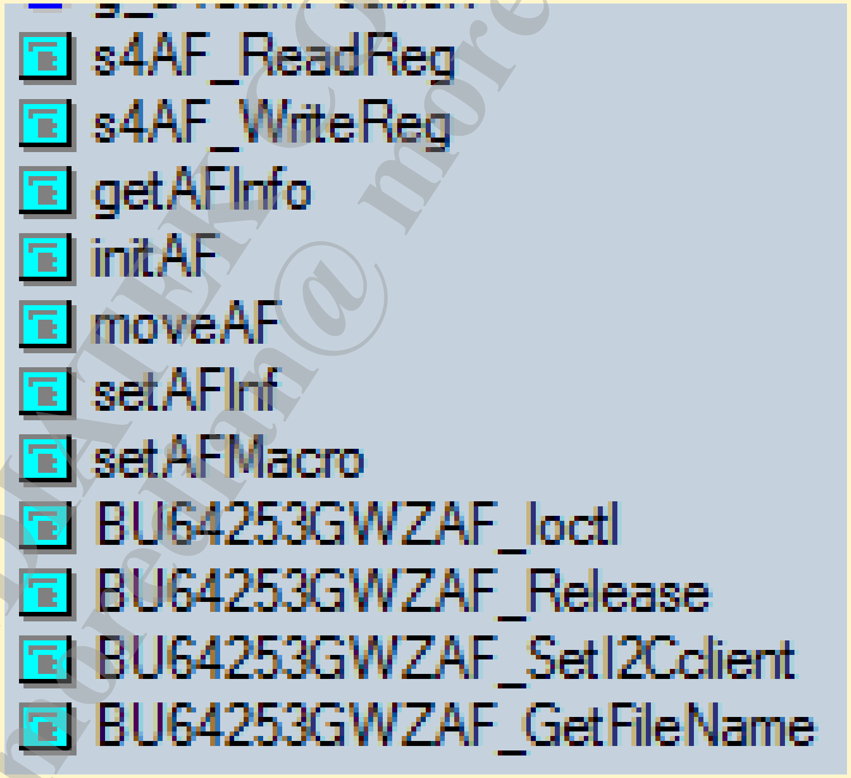


Step3-1
Implement VCM driver

BU64253WZAF.c

- Modify the lens driver

BU64253WZAF.c can be cloned from other lens driver

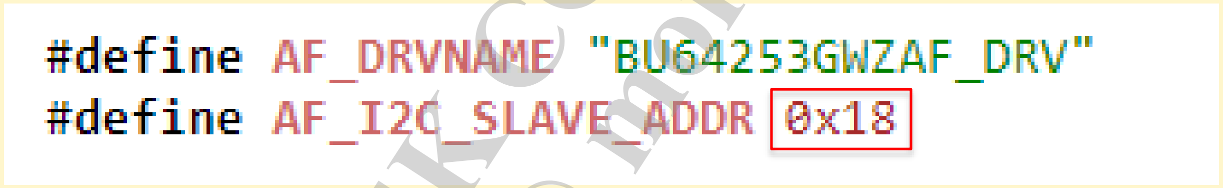


s4AF_ReadReg
s4AF_WriteReg
getAFInfo
initAF
moveAF
setAFInf
setAFMacro
BU64253GWZAF_Ioctl
BU64253GWZAF_Release
BU64253GWZAF_SetI2Cclient
BU64253GWZAF_GetFileName

Step3-1-1
Implement VCM driver

BU64253WZAF.c

- configure I2C address



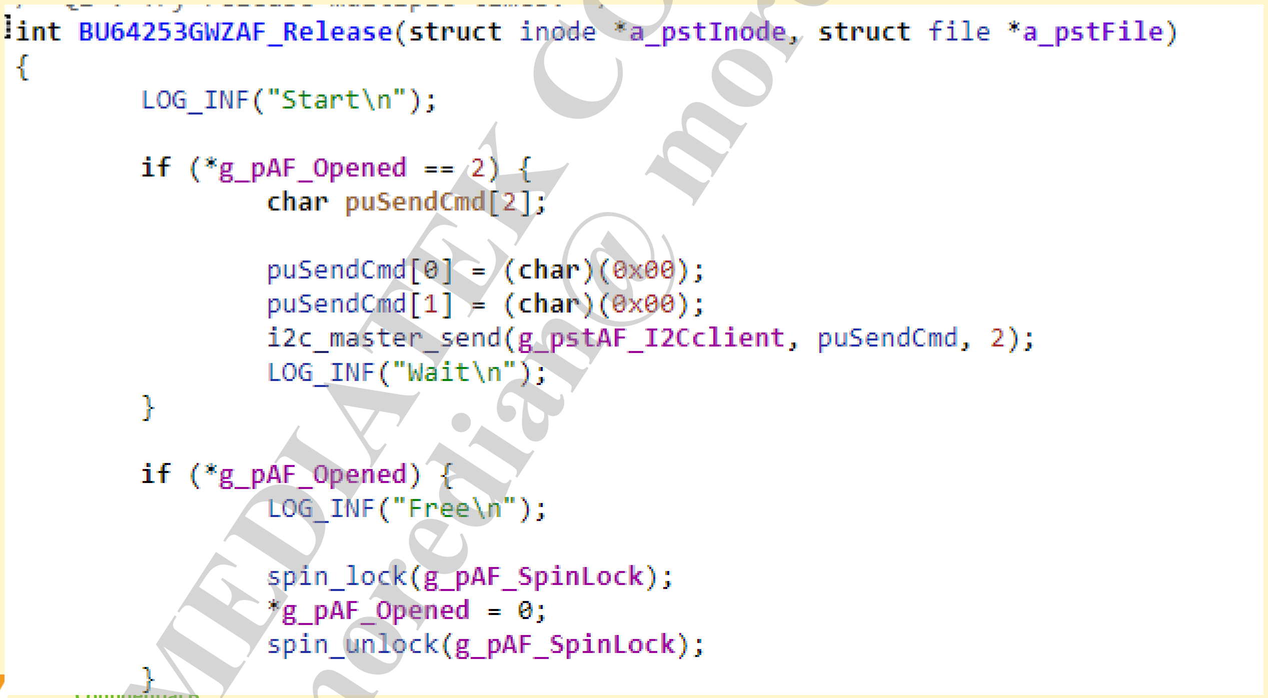
```
#define AF_DRVNAME "BU64253GWZAF_DRV"
#define AF_I2C_SLAVE_ADDR 0x18
```

Step3-1-2
Implement VCM driver

BU64253WZAF.c

- BU64253GWZAF_Release

uninit AF, Will be called when unint AF



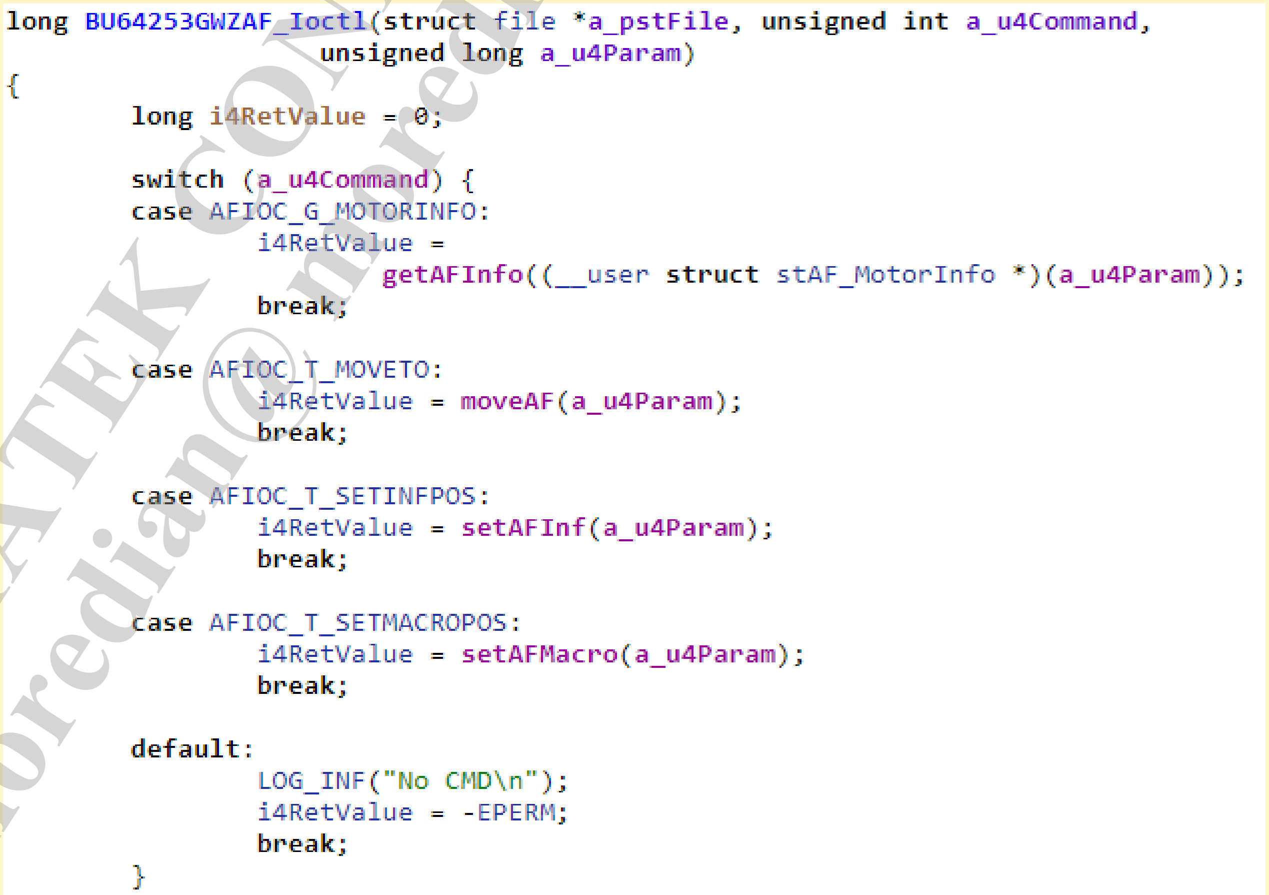
```
int BU64253GWZAF_Release(struct inode *a_pstInode, struct file *a_pstFile)
{
        LOG_INF("Start\n");

        if (*g_pAF_Opened == 2) {
                char puSendCmd[2];

                puSendCmd[0] = (char)(0x00);
                puSendCmd[1] = (char)(0x00);
                i2c_master_send(g_pstAF_I2Cclient, puSendCmd, 2);
                LOG_INF("Wait\n");
        }

        if (*g_pAF_Opened) {
                LOG_INF("Free\n");

                spin_lock(g_pAF_SpinLock);
                *g_pAF_Opened = 0;
                spin_unlock(g_pAF_SpinLock);
        }
```

Step3-1-3
Implement VCM driver

BU64253WZAF.c

- BU64253GWZAF_Ioctl

Ioctrl function



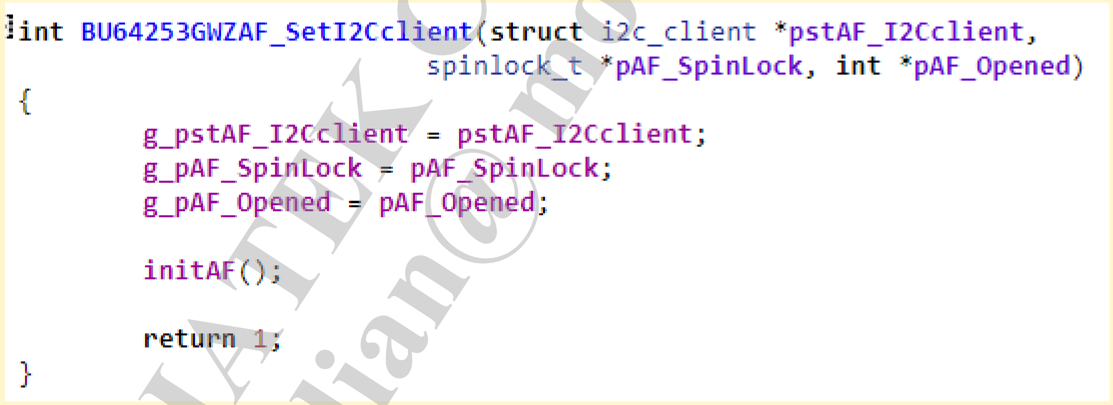
```
long BU64253GWZAF_Ioctl(struct file *a_pstFile, unsigned int a_u4Command,
                         unsigned long a_u4Param)
{
        long i4RetValue = 0;

        switch (a_u4Command) {
        case AFIOC_G_MOTORINFO:
                i4RetValue =
                        getAFInfo((__user struct stAF_MotorInfo *)(a_u4Param));
                break;

        case AFIOC_T_MOVETO:
                i4RetValue = moveAF(a_u4Param);
                break;

        case AFIOC_T_SETINFPOS:
                i4RetValue = setAFInf(a_u4Param);
                break;

        case AFIOC_T_SETMACROPOS:
                i4RetValue = setAFMacro(a_u4Param);
                break;

        default:
                LOG_INF("No CMD\n");
                i4RetValue = -EPERM;
                break;
        }
```

Step3-1-4
Implement VCM driver

BU64253WZAF.c

- BU64253GWZAF_SetI2Cclient

set I2C client , and initAF will be called when init AF



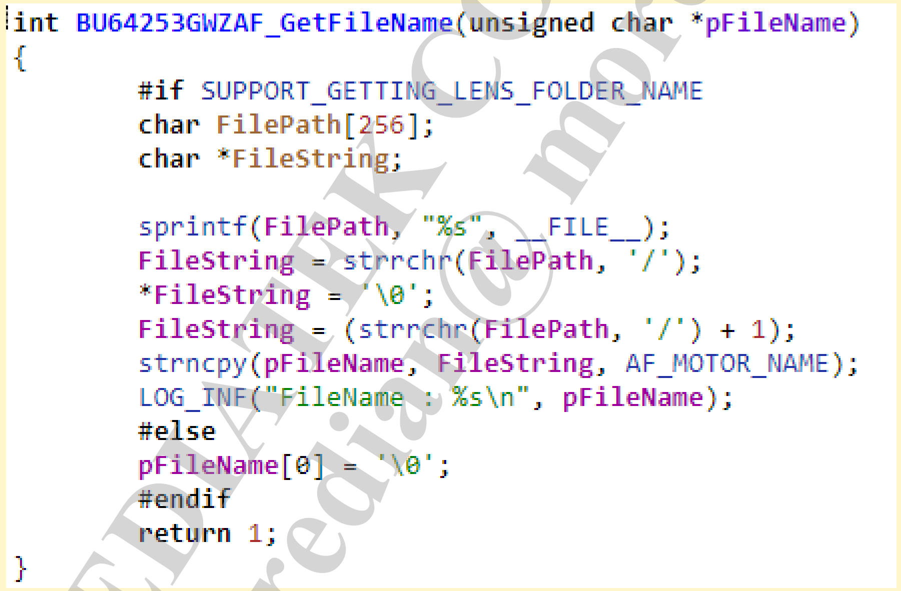
```
int BU64253GWZAF_SetI2Cclient(struct i2c_client *pstAF_I2Cclient,
                              spinlock_t *pAF_SpinLock, int *pAF_Opened)
{
        g_pstAF_I2Cclient = pstAF_I2Cclient;
        g_pAF_SpinLock = pAF_SpinLock;
        g_pAF_Opened = pAF_Opened;

        initAF();

        return 1;
}
```

Step3-1-5
Implement VCM driver

BU64253WZAF.c

- BU64253GWZAF_GetFileName

get file name



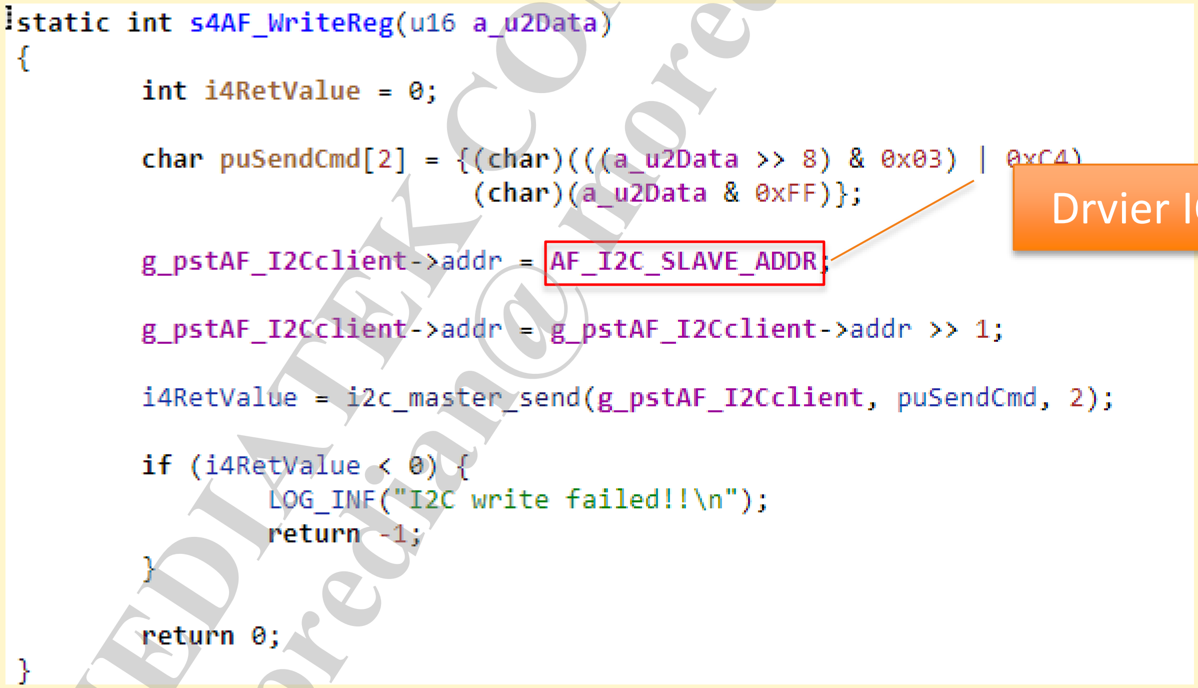
```
int BU64253GWZAF_GetFileName(unsigned char *pFileName)
{
        #if SUPPORT_GETTING_LENS_FOLDER_NAME
        char FilePath[256];
        char *FileString;

        sprintf(FilePath, "%s", __FILE__);
        FileString = strrchr(FilePath, '/');
        *FileString = '\0';
        FileString = (strrchr(FilePath, '/') + 1);
        strncpy(pFileName, FileString, AF_MOTOR_NAME);
        LOG_INF("FileName : %s\n", pFileName);
        #else
        pFileName[0] = '\0';
        #endif
        return 1;
}
```

Step3-1-6
Implement VCM driver

BU64253WZAF.c

- s4AF_WriteReg

  write register API



```
static int s4AF_WriteReg(u16 a_u2Data)
{
        int i4RetValue = 0;

        char puSendCmd[2] = {(char)(((a_u2Data >> 8) & 0x03) | 0xC4)
                             (char)(a_u2Data & 0xFF)};

        g_pstAF_I2Cclient->addr = AF_I2C_SLAVE_ADDR;

        g_pstAF_I2Cclient->addr = g_pstAF_I2Cclient->addr >> 1;

        i4RetValue = i2c_master_send(g_pstAF_I2Cclient, puSendCmd, 2);

        if (i4RetValue < 0) {
                LOG_INF("I2C write failed!!\n");
                return -1;
        }

        return 0;
}
```

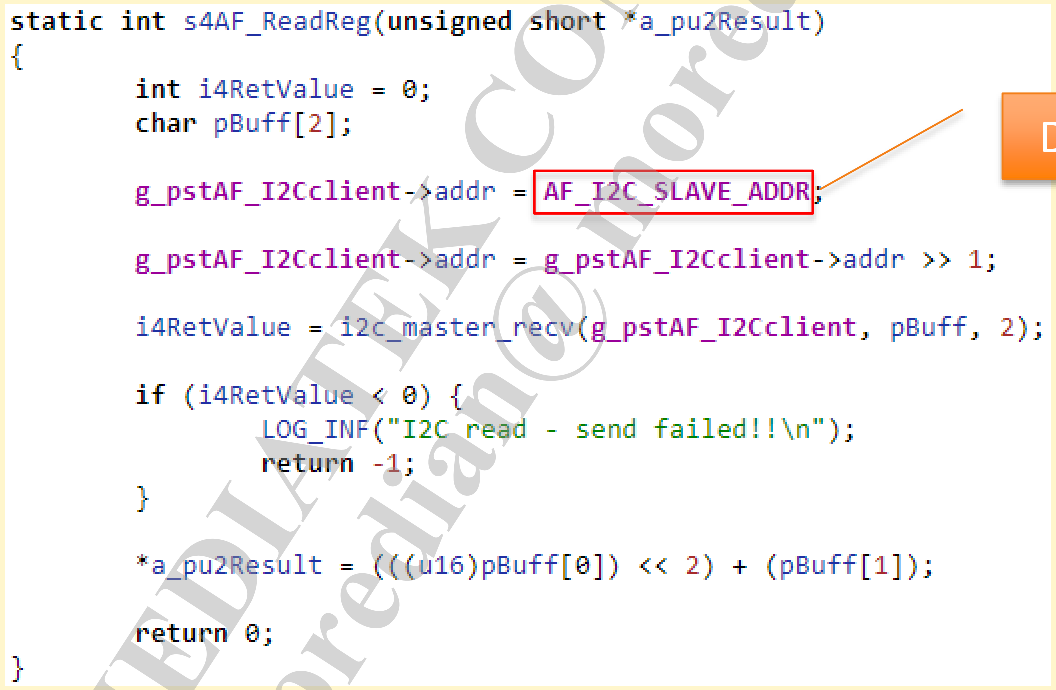
Drvier IC I2C address

Step3-1-7
Implement VCM driver

BU64253WZAF.c

- s4AF_ReadReg

  read register API



```
static int s4AF_ReadReg(unsigned short *a_pu2Result)
{
        int i4RetValue = 0;
        char pBuff[2];

        g_pstAF_I2Cclient->addr = AF_I2C_SLAVE_ADDR;

        g_pstAF_I2Cclient->addr = g_pstAF_I2Cclient->addr >> 1;

        i4RetValue = i2c_master_recv(g_pstAF_I2Cclient, pBuff, 2);

        if (i4RetValue < 0) {
                LOG_INF("I2C read - send failed!!\n");
                return -1;
        }

        *a_pu2Result = (((u16)pBuff[0]) << 2) + (pBuff[1]);

        return 0;
}
```

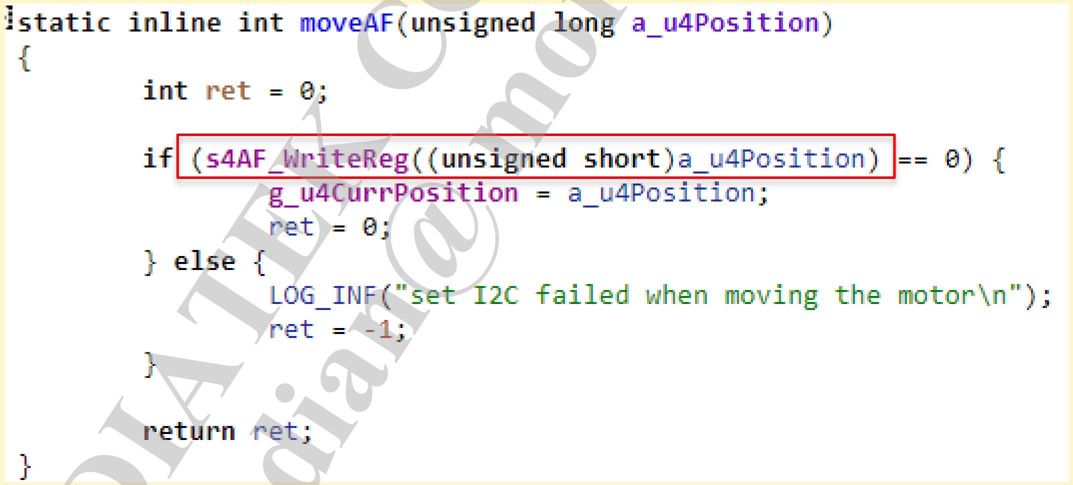
Drvier IC I2C address

Step3-1-8
Implement VCM driver

BU64253WZAF.c

- moveAF

write lens pos to driver IC



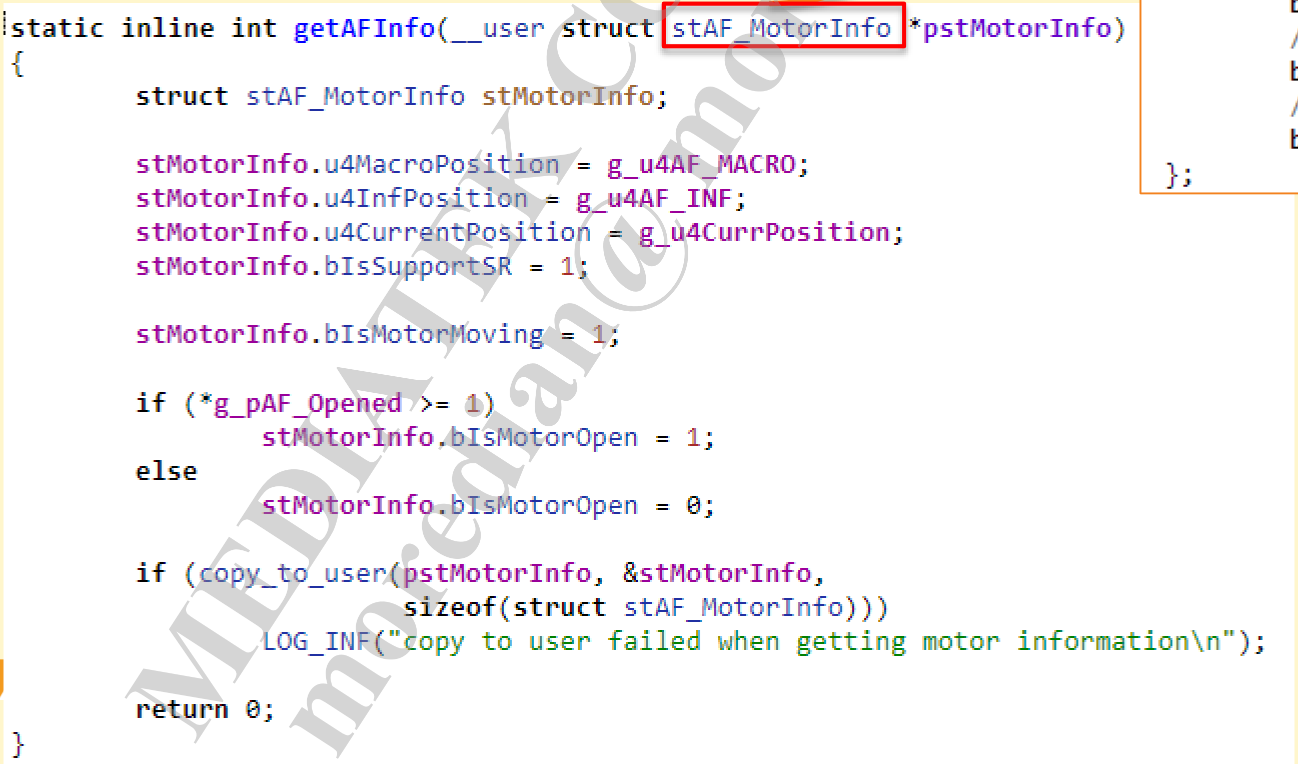
```
static inline int moveAF(unsigned long a_u4Position)
{
        int ret = 0;

        if (s4AF_WriteReg((unsigned short)a_u4Position) == 0) {
                g_u4CurrPosition = a_u4Position;
                ret = 0;
        } else {
                LOG_INF("set I2C failed when moving the motor\n");
                ret = -1;
        }

        return ret;
}
```

Step3-1-9
Implement VCM driver

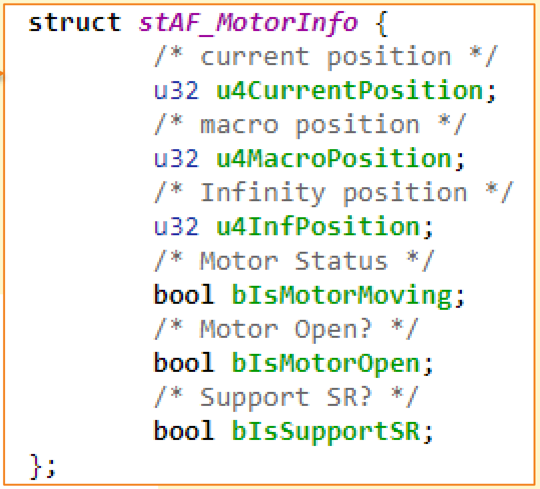
- getAFInfo

Get AF information

BU64253WZAF.c



```
static inline int getAFInfo(__user struct stAF_MotorInfo *pstMotorInfo)
{
        struct stAF_MotorInfo stMotorInfo;

        stMotorInfo.u4MacroPosition = g_u4AF_MACRO;
        stMotorInfo.u4InfPosition = g_u4AF_INF;
        stMotorInfo.u4CurrentPosition = g_u4CurrPosition;
        stMotorInfo.bIsSupportSR = 1;

        stMotorInfo.bIsMotorMoving = 1;

        if (*g_pAF_Opened >= 1)
                stMotorInfo.bIsMotorOpen = 1;
        else
                stMotorInfo.bIsMotorOpen = 0;

        if (copy_to_user(pstMotorInfo, &stMotorInfo,
                     sizeof(struct stAF_MotorInfo)))
                LOG_INF("copy to user failed when getting motor information\n");

        return 0;
}
```



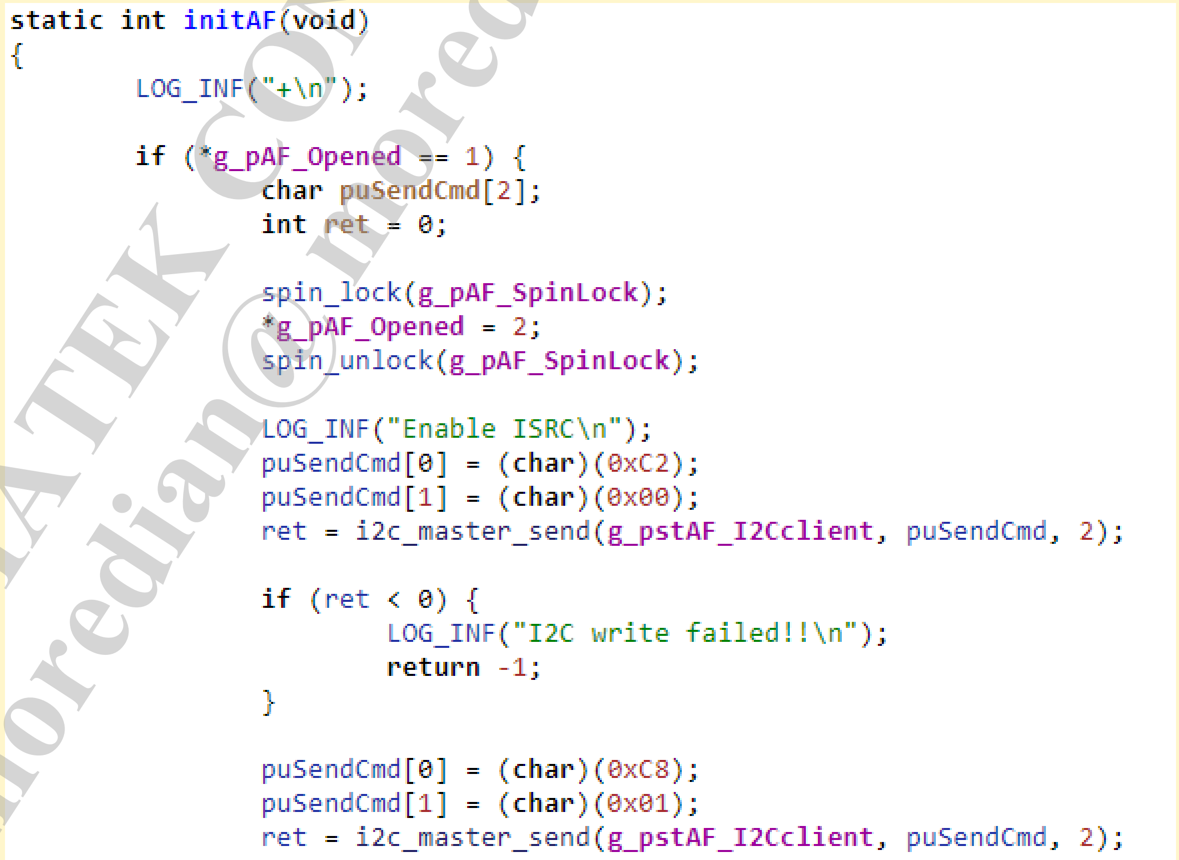
```
struct stAF_MotorInfo {
        /* current position */
        u32 u4CurrentPosition;
        /* macro position */
        u32 u4MacroPosition;
        /* Infinity position */
        u32 u4InfPosition;
        /* Motor Status */
        bool bIsMotorMoving;
        /* Motor Open? */
        bool bIsMotorOpen;
        /* Support SR? */
        bool bIsSupportSR;
};
```

Step3-1-10
Implement VCM driver

BU64253WZAF.c

- initAF

init driver IC



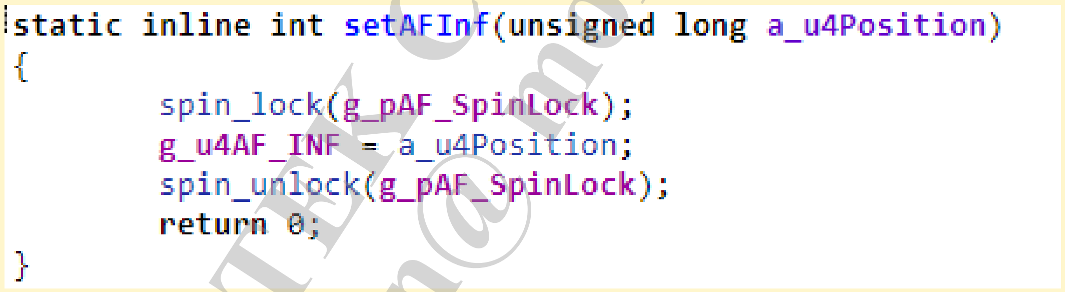
```
static int initAF(void)
{
	LOG_INF("+\n");

	if (*g_pAF_Opened == 1) {
		char puSendCmd[2];
		int ret = 0;

		spin_lock(g_pAF_SpinLock);
		*g_pAF_Opened = 2;
		spin_unlock(g_pAF_SpinLock);

		LOG_INF("Enable ISRC\n");
		puSendCmd[0] = (char)(0xC2);
		puSendCmd[1] = (char)(0x00);
		ret = i2c_master_send(g_pstAF_I2Cclient, puSendCmd, 2);

		if (ret < 0) {
			LOG_INF("I2C write failed!!\n");
			return -1;
		}

		puSendCmd[0] = (char)(0xC8);
		puSendCmd[1] = (char)(0x01);
		ret = i2c_master_send(g_pstAF_I2Cclient, puSendCmd, 2);
```

Step3-1-11
Implement VCM driver

BU64253WZAF.c

- setAFInf

set infinity pos to lens driver



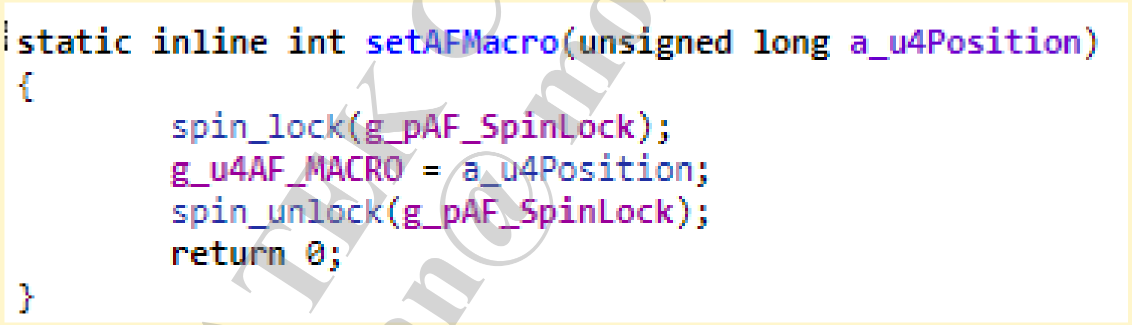
```
static inline int setAFInf(unsigned long a_u4Position)
{
        spin_lock(g_pAF_SpinLock);
        g_u4AF_INF = a_u4Position;
        spin_unlock(g_pAF_SpinLock);
        return 0;
}
```

Step3-1-12
Implement VCM driver

BU64253WZAF.c

- setAFMacro

set macro position to lens driver



```
static inline int setAFMacro(unsigned long a_u4Position)
{
        spin_lock(g_pAF_SpinLock);
        g_u4AF_MACRO = a_u4Position;
        spin_unlock(g_pAF_SpinLock);
        return 0;
}
```

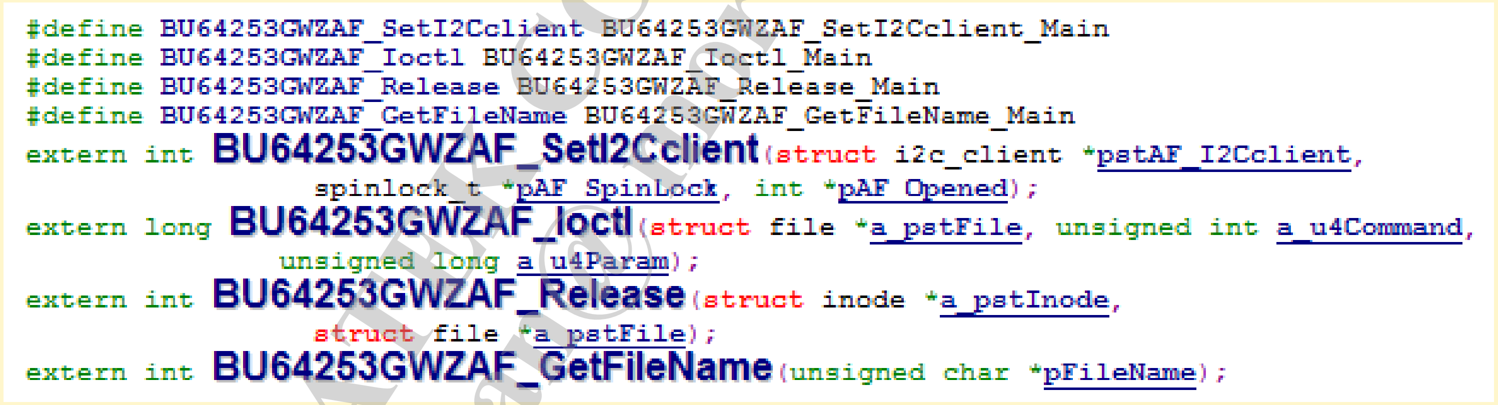
MEDIATEK

Confidential B

Step3-2
Implement VCM driver

lens_list.h

- Add driver’s extern function

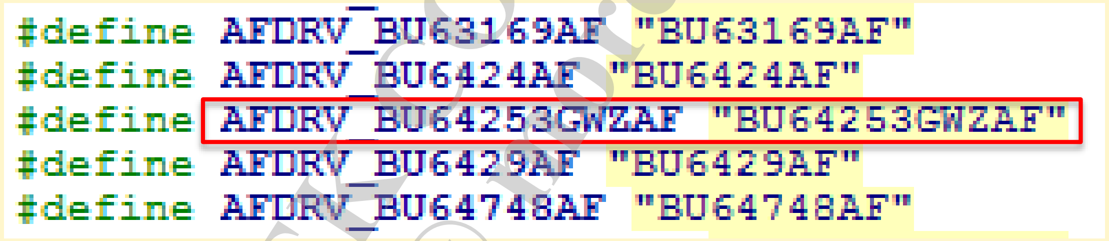


```
#define BU64253GWZAF_SetI2Cclient BU64253GWZAF_SetI2Cclient_Main
#define BU64253GWZAF_Ioctl BU64253GWZAF_Ioctl_Main
#define BU64253GWZAF_Release BU64253GWZAF_Release_Main
#define BU64253GWZAF_GetFileName BU64253GWZAF_GetFileName_Main
extern int BU64253GWZAF_SetI2Cclient(struct i2c_client *pstAF_I2Cclient,
                spinlock_t *pAF_SpinLock, int *pAF_Opened);
extern long BU64253GWZAF_Ioctl(struct file *a_pstFile, unsigned int a_u4Command,
               unsigned long a_u4Param);
extern int BU64253GWZAF_Release(struct inode *a_pstInode,
                struct file *a_pstFile);
extern int BU64253GWZAF_GetFileName(unsigned char *pFileName);
```

Step3-3
Implement VCM driver

lens_info.h

- Add AF driver name

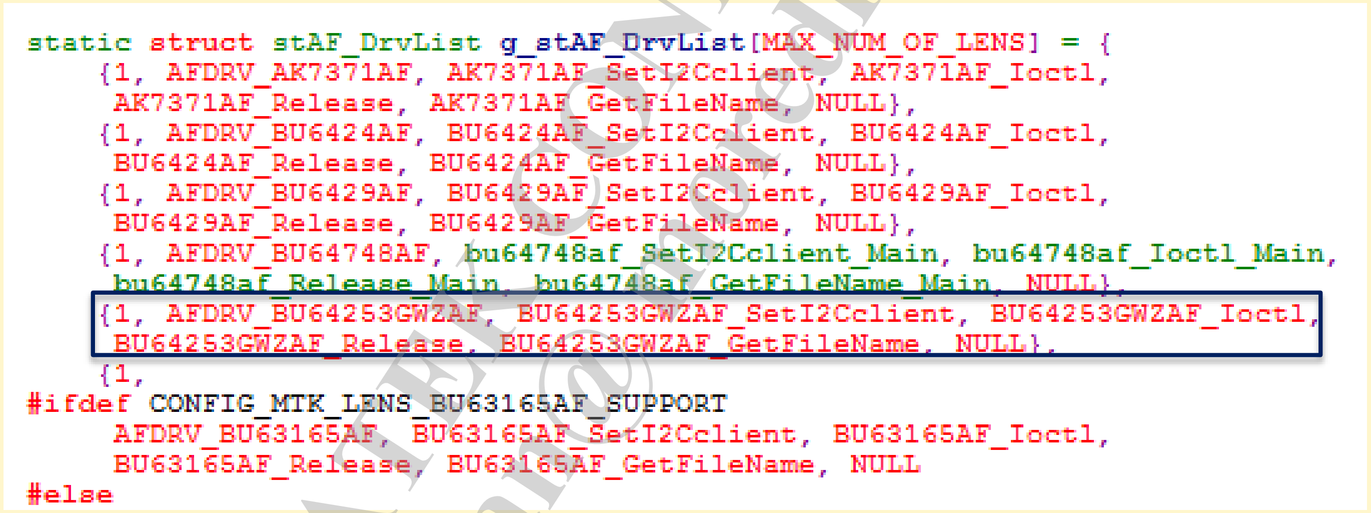


```
#define AFDRV_BU63169AF "BU63169AF"
#define AFDRV_BU6424AF "BU6424AF"
#define AFDRV_BU64253GWZAF "BU64253GWZAF"
#define AFDRV_BU6429AF "BU6429AF"
#define AFDRV_BU64748AF "BU64748AF"
```

Step3-4
Implement VCM driver

main_lens.c

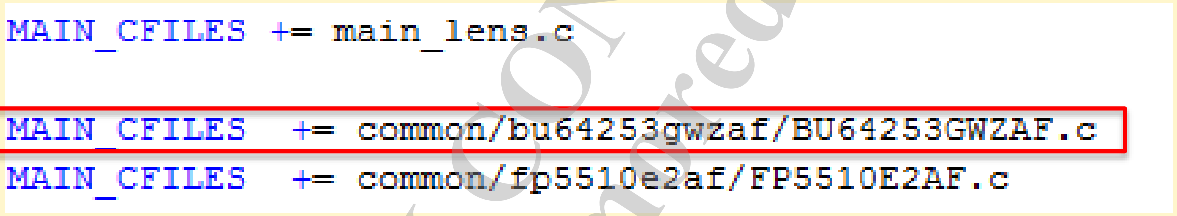
- Add Driver to DrvList



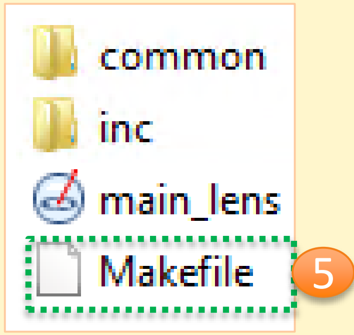
```
static struct stAF_DrvList g_stAF_DrvList[MAX_NUM_OF_LENS] = {
    {1, AFDRV_AK7371AF, AK7371AF_SetI2Cclient, AK7371AF_Ioctl,
     AK7371AF_Release, AK7371AF_GetFileName, NULL},
    {1, AFDRV_BU6424AF, BU6424AF_SetI2Cclient, BU6424AF_Ioctl,
     BU6424AF_Release, BU6424AF_GetFileName, NULL},
    {1, AFDRV_BU6429AF, BU6429AF_SetI2Cclient, BU6429AF_Ioctl,
     BU6429AF_Release, BU6429AF_GetFileName, NULL},
    {1, AFDRV_BU64748AF, bu64748af_SetI2Cclient_Main, bu64748af_Ioctl_Main,
     bu64748af_Release_Main, bu64748af_GetFileName_Main, NULL},
    {1, AFDRV_BU64253GWZAF, BU64253GWZAF_SetI2Cclient, BU64253GWZAF_Ioctl,
     BU64253GWZAF_Release, BU64253GWZAF_GetFileName, NULL},
    {1,
#ifdef CONFIG_MTK_LENS_BU63165AF_SUPPORT
     AFDRV_BU63165AF, BU63165AF_SetI2Cclient, BU63165AF_Ioctl,
     BU63165AF_Release, BU63165AF_GetFileName, NULL
#else
```

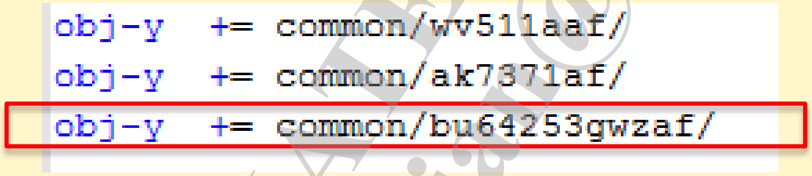
Step3-5
Implement VCM driver

- Add the makefile setting of lens driver



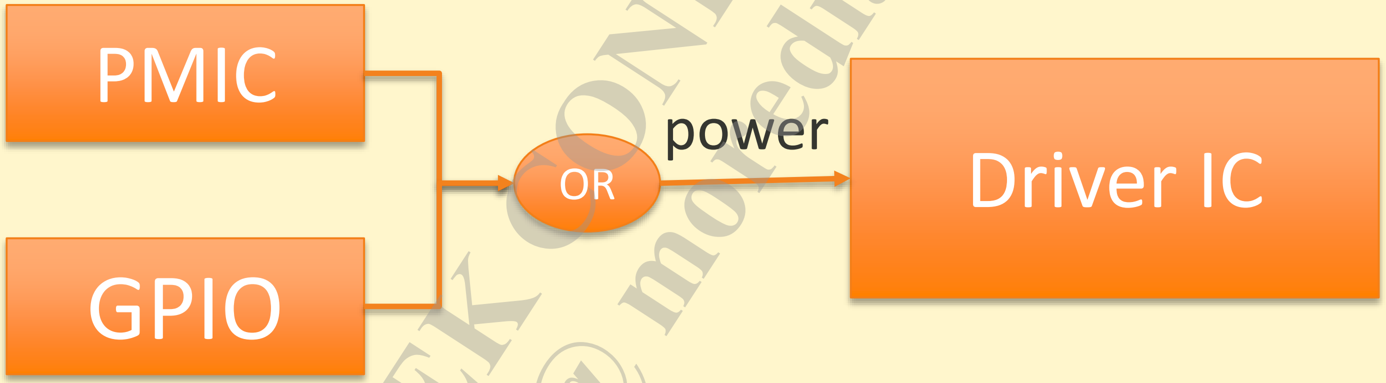
```
MAIN_CFILES += main_lens.c

MAIN_CFILES   += common/bu64253gwzaf/BU64253GWZAF.c
MAIN_CFILES   += common/fp5510e2af/FP5510E2AF.c
```





```
obj-y   += common/wv511aaf/
obj-y   += common/ak7371af/
obj-y   += common/bu64253gwzaf/
```

Step 4
VCM Power configuration

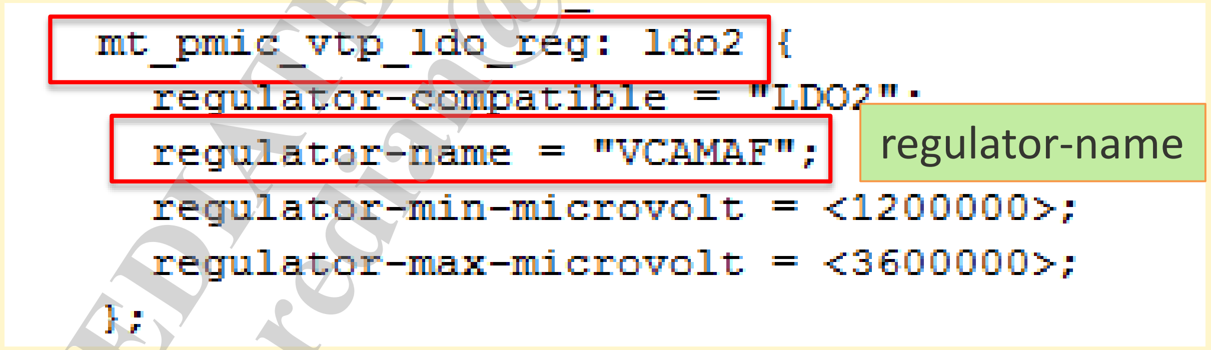


Step 4-1-1
PMIC supply power (Kernel-4.9/4.4)

- Confirm PMIC pin

  eg: AF voltage is supplied by ldo2 of mt6360(PMIC)

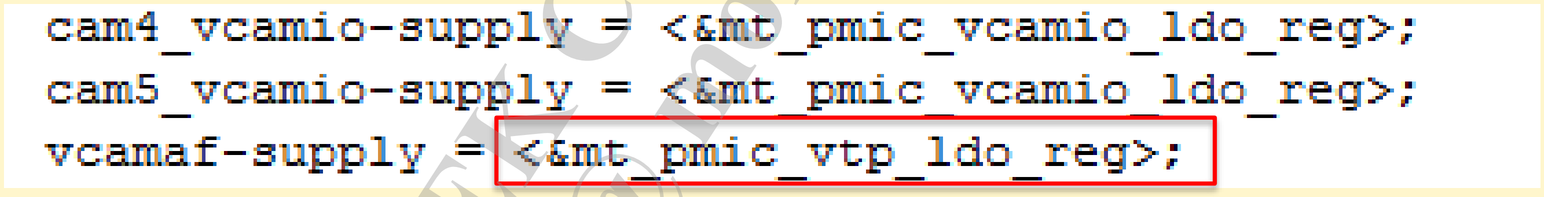
- Configure regulator name in mt6360.dtsi



```
mt_pmic_vtp_ldo_reg: ldo2 {
    regulator-compatible = "LDO2";
    regulator-name = "VCAMAF";
    regulator-min-microvolt = <1200000>;
    regulator-max-microvolt = <3600000>;
};
```

Step 4-1-2
PMIC supply power (Kernel-4.9/4.4)

- Configure power supply with ldo2 in [$project].dts

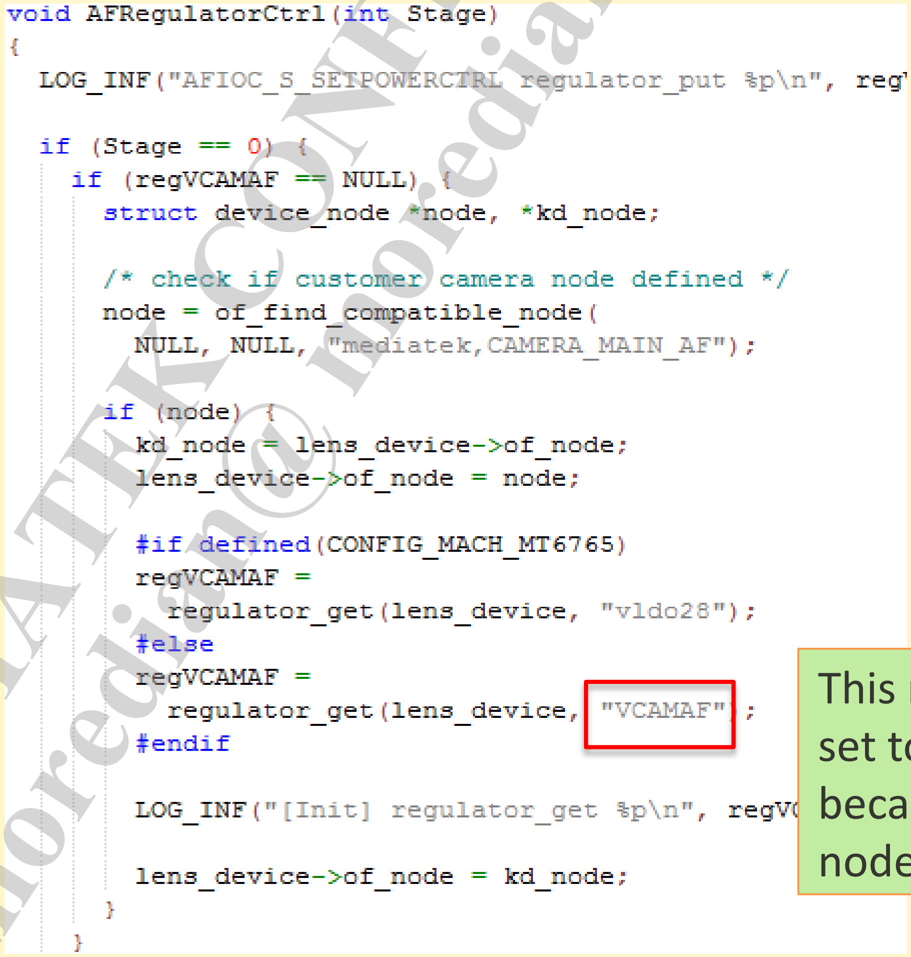


```
cam4_vcamio-supply = <&mt_pmic_vcamio_ldo_reg>;
cam5_vcamio-supply = <&mt_pmic_vcamio_ldo_reg>;
vcamaf-supply = <&mt_pmic_vtp_ldo_reg>;
```

Step 4-1-3
PMIC supply power (Kernel-4.9/4.4)

main_lens.c

- Configure AFRegulatorCtrl



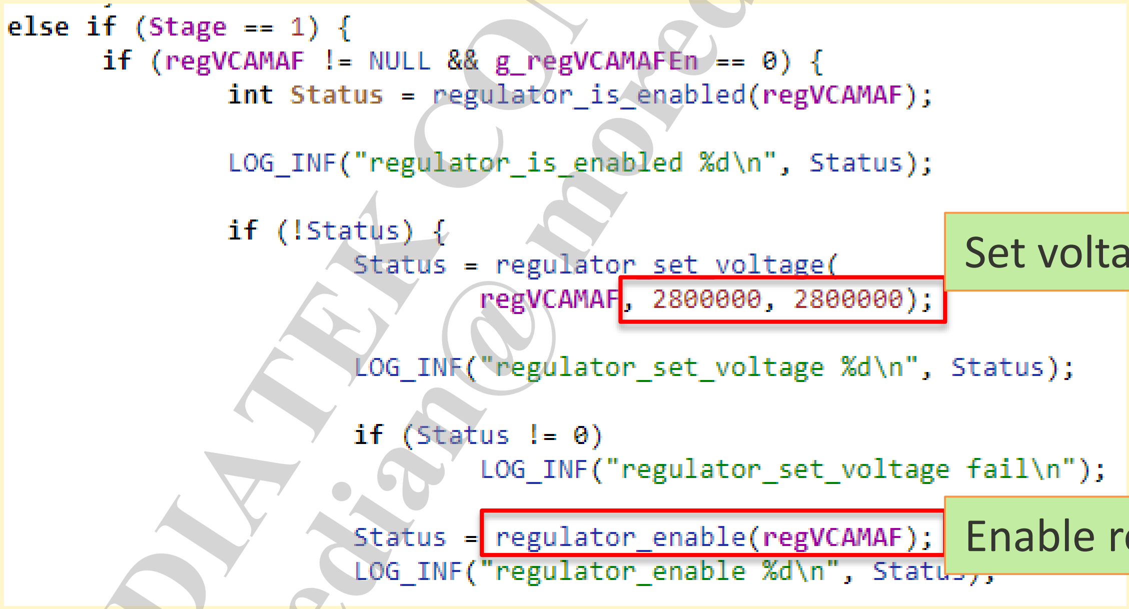
```
void AFRegulatorCtrl(int Stage)
{
  LOG_INF("AFIOC_S_SETPOWERCTRL regulator_put %p\n", reg

  if (Stage == 0) {
    if (regVCAMAF == NULL) {
      struct device_node *node, *kd_node;

      /* check if customer camera node defined */
      node = of_find_compatible_node(
        NULL, NULL, "mediatek,CAMERA_MAIN_AF");

      if (node) {
        kd_node = lens_device->of_node;
        lens_device->of_node = node;

        #if defined(CONFIG_MACH_MT6765)
        regVCAMAF =
          regulator_get(lens_device, "vldo28");
        #else
        regVCAMAF =
          regulator_get(lens_device, "VCAMAF");
        #endif

        LOG_INF("[Init] regulator_get %p\n", regV

        lens_device->of_node = kd_node;
      }
    }
```

This name can be only set to regulator-name because no AF dts node is configured

Step 4-1-4
PMIC supply power (Kernel-4.9/4.4)

main_lens.c

- Set voltage



```
else if (Stage == 1) {
        if (regVCAMAF != NULL && g_regVCAMAFEn == 0) {
                int Status = regulator_is_enabled(regVCAMAF);

                LOG_INF("regulator_is_enabled %d\n", Status);

                if (!Status) {
                        Status = regulator_set_voltage(
                                regVCAMAF, 2800000, 2800000);

                        LOG_INF("regulator_set_voltage %d\n", Status);

                        if (Status != 0)
                                LOG_INF("regulator_set_voltage fail\n");

                        Status = regulator_enable(regVCAMAF);
                        LOG_INF("regulator_enable %d\n", Status);
```

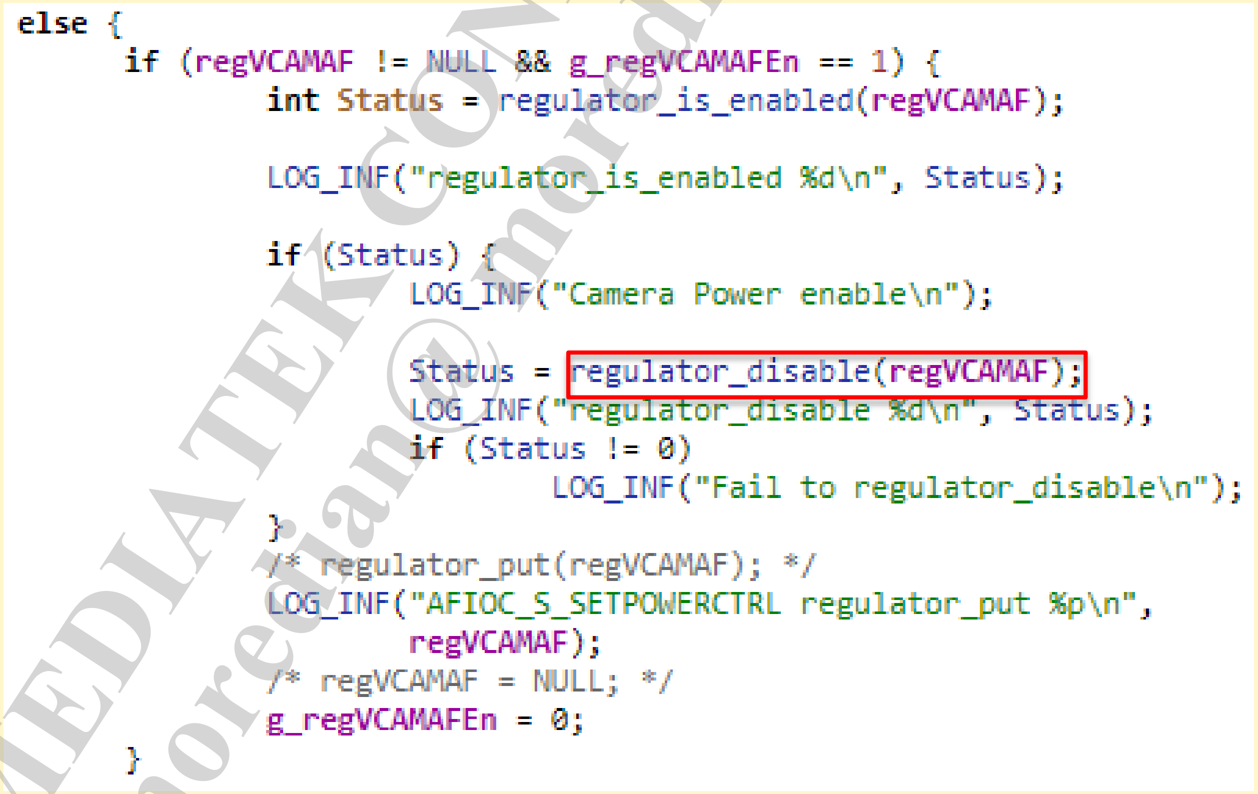
Set voltage to 2.8V

Enable regulator

Step 4-1-4
PMIC supply power (Kernel-4.9/4.4)

main_lens.c

- Disable regulator

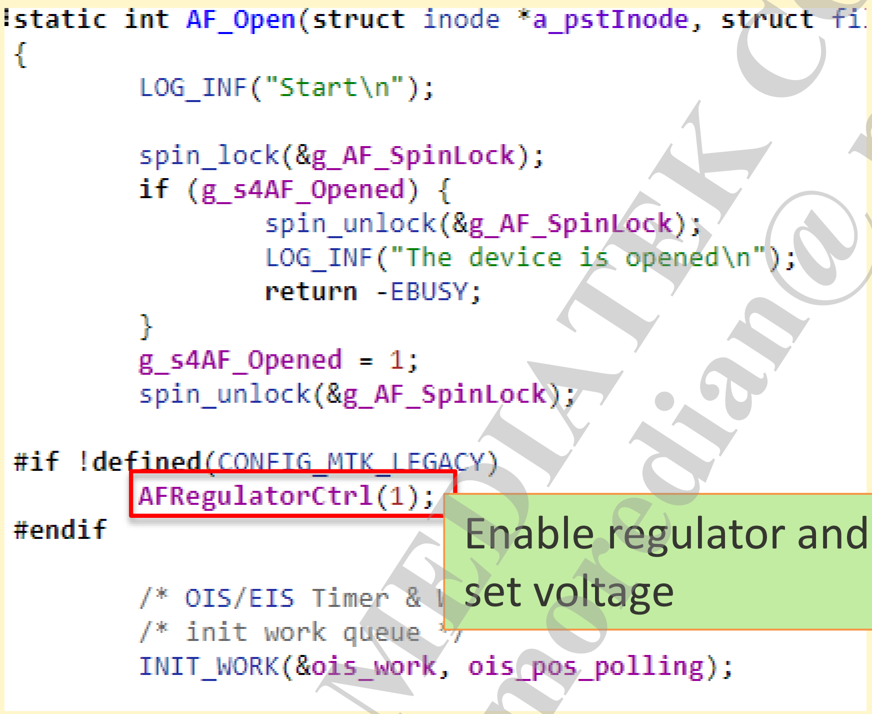


```
else {
	if (regVCAMAF != NULL && g_regVCAMAFEn == 1) {
		int Status = regulator_is_enabled(regVCAMAF);

		LOG_INF("regulator_is_enabled %d\n", Status);

		if (Status) {
			LOG_INF("Camera Power enable\n");

			Status = regulator_disable(regVCAMAF);
			LOG_INF("regulator_disable %d\n", Status);
			if (Status != 0)
				LOG_INF("Fail to regulator_disable\n");
		}
		/* regulator_put(regVCAMAF); */
		LOG_INF("AFIOC_S_SETPOWERCTRL regulator_put %p\n",
			regVCAMAF);
		/* regVCAMAF = NULL; */
		g_regVCAMAFEn = 0;
	}
```

Step 4-1-5
PMIC supply power (Kernel-4.9/4.4)

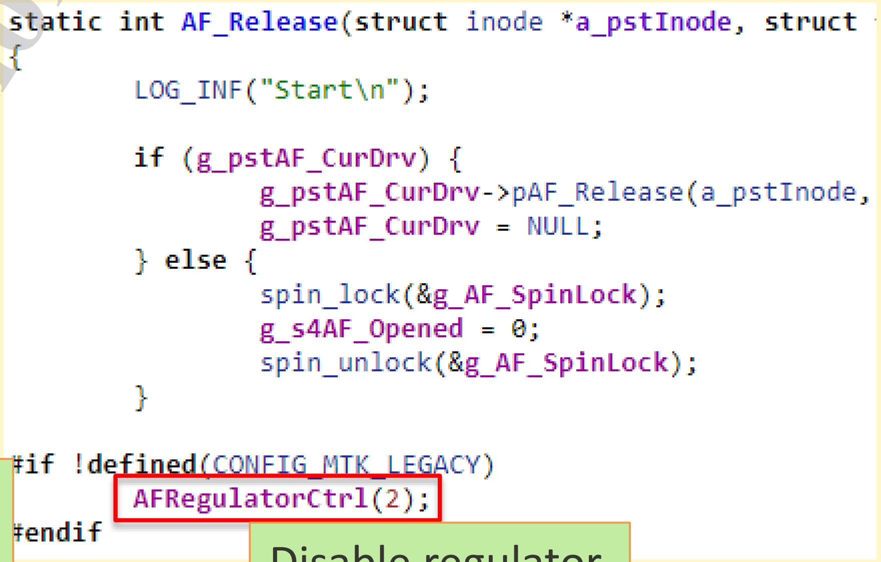
main_lens.c

- Enable regulator when AF Open and disable regulator when AF Release



```
static int AF_Open(struct inode *a_pstInode, struct fi
{
        LOG_INF("Start\n");

        spin_lock(&g_AF_SpinLock);
        if (g_s4AF_Opened) {
                spin_unlock(&g_AF_SpinLock);
                LOG_INF("The device is opened\n");
                return -EBUSY;
        }
        g_s4AF_Opened = 1;
        spin_unlock(&g_AF_SpinLock);

#if !defined(CONFIG_MTK_LEGACY)
        AFRegulatorCtrl(1);
#endif

        /* OIS/EIS Timer & w
        /* init work queue */
        INIT_WORK(&ois_work, ois_pos_polling);
```

Enable regulator and set voltage



```
static int AF_Release(struct inode *a_pstInode, struct
{
        LOG_INF("Start\n");

        if (g_pstAF_CurDrv) {
                g_pstAF_CurDrv->pAF_Release(a_pstInode,
                g_pstAF_CurDrv = NULL;
        } else {
                spin_lock(&g_AF_SpinLock);
                g_s4AF_Opened = 0;
                spin_unlock(&g_AF_SpinLock);
        }

#if !defined(CONFIG_MTK_LEGACY)
        AFRegulatorCtrl(2);
#endif
```

Disable regulator

Step 4-1-6
GPIO supply power (Kernel-4.9/4.4)

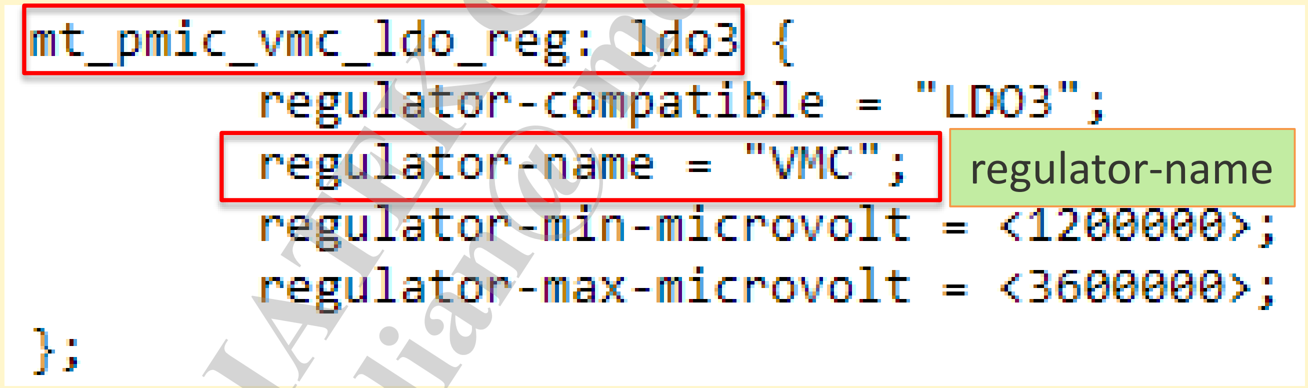
- Kernel-4.9 doesn't support GPIO pin control in lens driver, you can enable GPIO in camera driver. Otherwise, you must submit a e-service to get a patch.

Step 4-2-1
GPIO/PMIC supply power (Kernel-4.14)

- Confirm PMIC or GPIO pin

  eg: AF voltage is supplied by ldo3 of pmic mt6360

  eg: AF voltage is supplied by GPIO30
- Configure regulator name in mt6360.dtsi

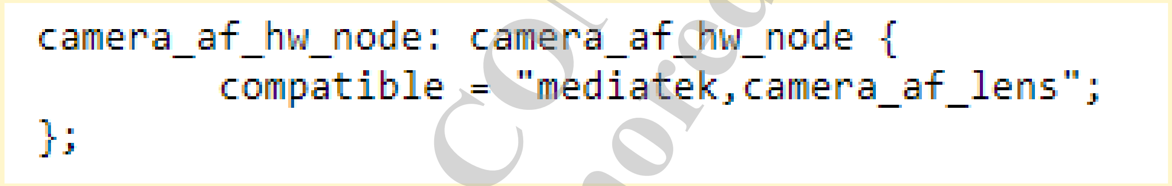


```
mt_pmic_vmc_ldo_reg: ldo3 {
        regulator-compatible = "LDO3";
        regulator-name = "VMC";
        regulator-min-microvolt = <1200000>;
        regulator-max-microvolt = <3600000>;
};
```

Step 4-2-2
GPIO/PMIC supply power (Kernel-4.14)

- Configure AF hw dts node in [$platform].dts

  if the node is exist , there is nothing to do

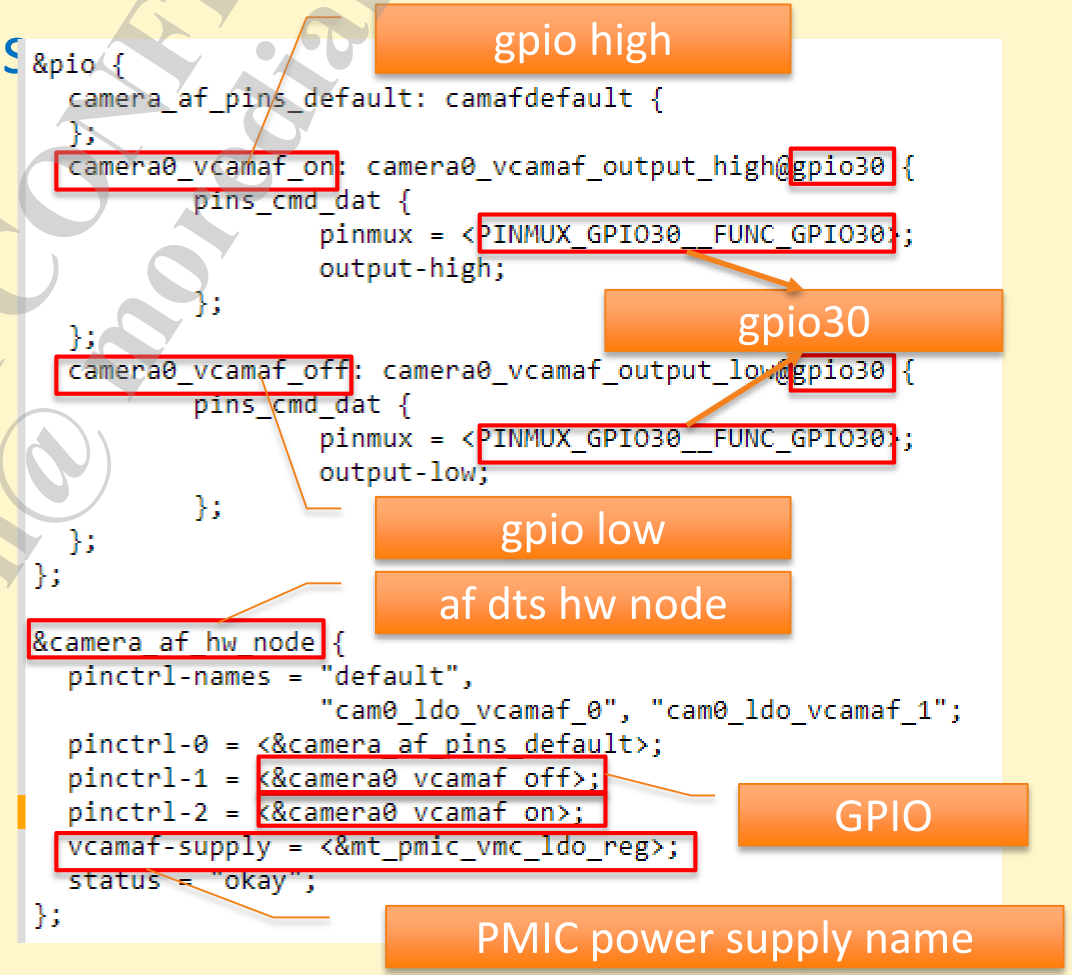


```
camera_af_hw_node: camera_af_hw_node {
        compatible = "mediatek,camera_af_lens";
};
```

Step 4-2-3
GPIO/PMIC supply power (Kernel-4.14)

- Configure power supply with ldo3 and pinctrl with GPIO in [$project].dts

GPIO and PMIC are both included in af hw dts node

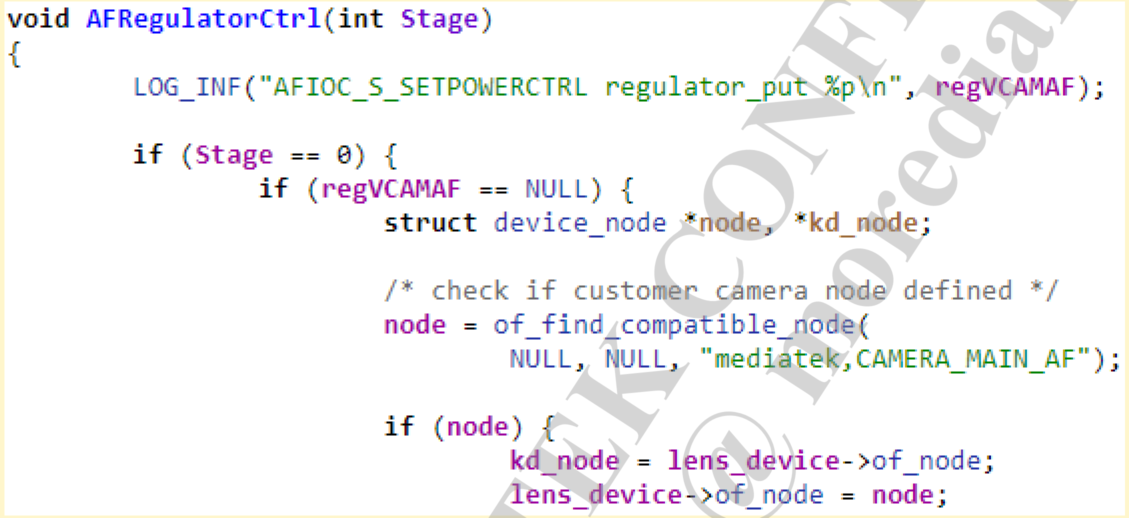


```
&pio {
  camera_af_pins_default: camafdefault {
  };
  camera0_vcamaf_on: camera0_vcamaf_output_high@gpio30 {
          pins_cmd_dat {
                  pinmux = <PINMUX_GPIO30__FUNC_GPIO30>;
                  output-high;
          };
  };
  camera0_vcamaf_off: camera0_vcamaf_output_low@gpio30 {
          pins_cmd_dat {
                  pinmux = <PINMUX_GPIO30__FUNC_GPIO30>;
                  output-low;
          };
  };
};

&camera_af_hw_node {
  pinctrl-names = "default",
                  "cam0_ldo_vcamaf_0", "cam0_ldo_vcamaf_1";
  pinctrl-0 = <&camera_af_pins_default>;
  pinctrl-1 = <&camera0_vcamaf_off>;
  pinctrl-2 = <&camera0_vcamaf_on>;
  vcamaf-supply = <&mt_pmic_vmc_ldo_reg>;
  status = "okay";
};
```

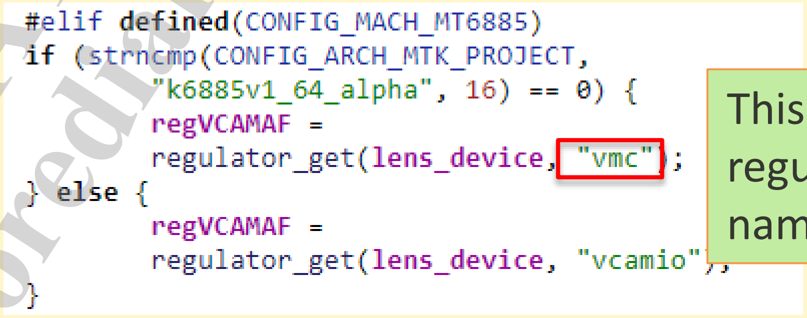
Step4-3-4
GPIO/PMIC supply power (Kernel-4.14)

main_lens.c

- Configure AFRegulatorCtrl



```
void AFRegulatorCtrl(int Stage)
{
        LOG_INF("AFIOC_S_SETPOWERCTRL regulator_put %p\n", regVCAMAF);

        if (Stage == 0) {
                if (regVCAMAF == NULL) {
                        struct device_node *node, *kd_node;

                        /* check if customer camera node defined */
                        node = of_find_compatible_node(
                                NULL, NULL, "mediatek,CAMERA_MAIN_AF");

                        if (node) {
                                kd_node = lens_device->of_node;
                                lens_device->of_node = node;
```



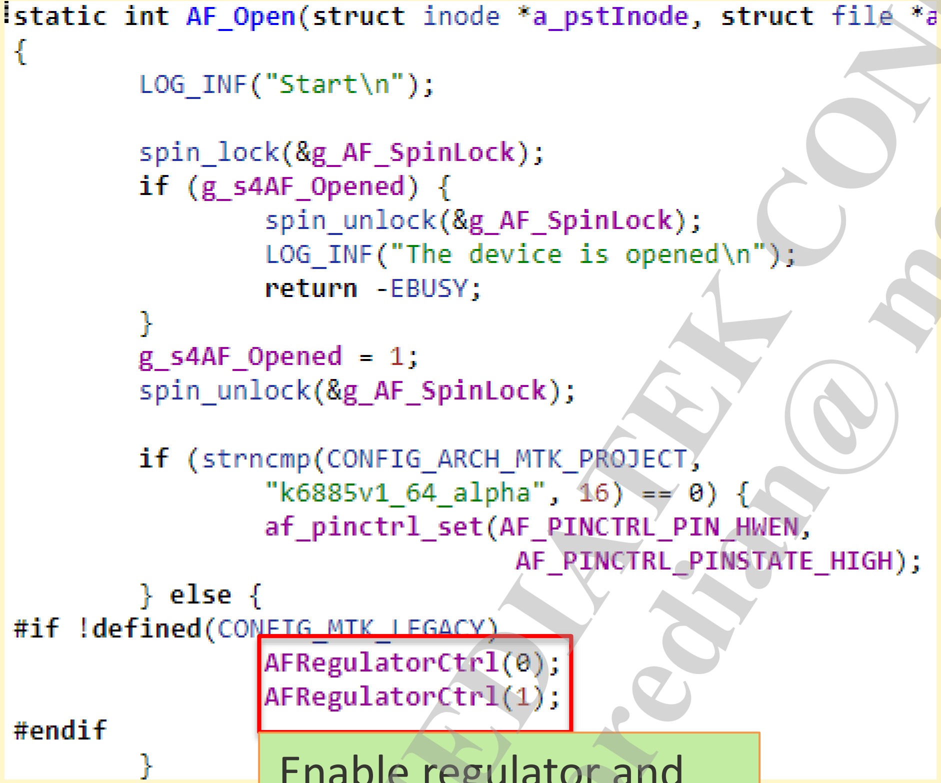
```
#elif defined(CONFIG_MACH_MT6885)
if (strncmp(CONFIG_ARCH_MTK_PROJECT,
        "k6885v1_64_alpha", 16) == 0) {
        regVCAMAF =
        regulator_get(lens_device, "vmc");
} else {
        regVCAMAF =
        regulator_get(lens_device, "vcamio");
}
```

This name can be set to regulator-name or supply name in dts hw node

Step 4-4-5
GPIO/PMIC supply power (Kernel-4.14)

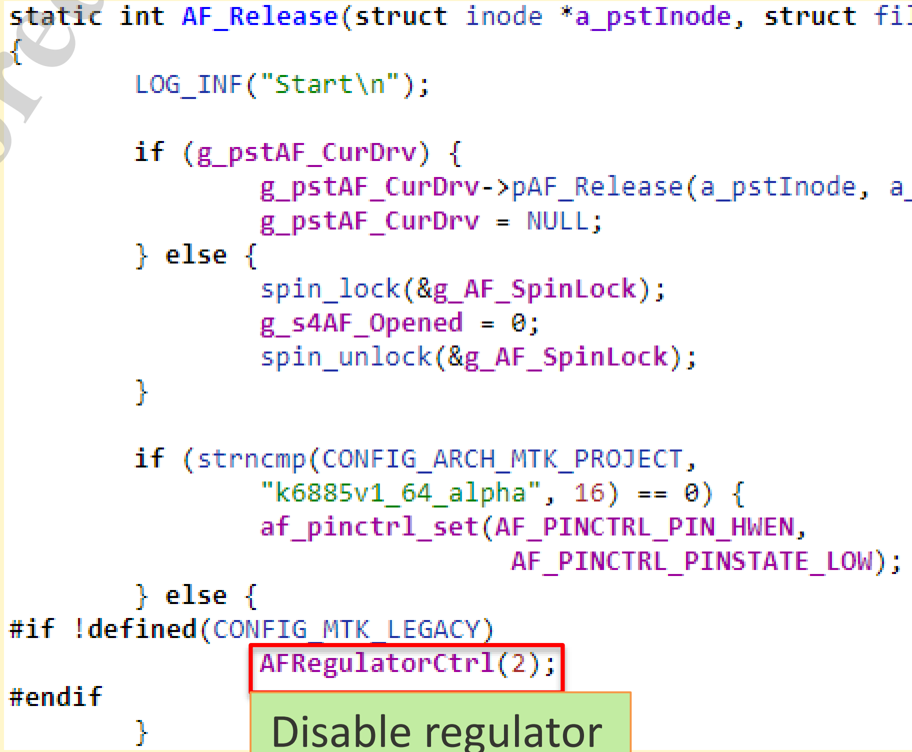
main_lens.c

- Enable regulator when AF Open and disable regulator when AF Release



```
static int AF_Open(struct inode *a_pstInode, struct file *a
{
	LOG_INF("Start\n");

	spin_lock(&g_AF_SpinLock);
	if (g_s4AF_Opened) {
		spin_unlock(&g_AF_SpinLock);
		LOG_INF("The device is opened\n");
		return -EBUSY;
	}
	g_s4AF_Opened = 1;
	spin_unlock(&g_AF_SpinLock);

	if (strncmp(CONFIG_ARCH_MTK_PROJECT,
		"k6885v1_64_alpha", 16) == 0) {
		af_pinctrl_set(AF_PINCTRL_PIN_HWEN,
			AF_PINCTRL_PINSTATE_HIGH);
	} else {
#if !defined(CONFIG_MTK_LEGACY)
		AFRegulatorCtrl(0);
		AFRegulatorCtrl(1);

#endif
	}
```

Enable regulator and set voltage



```
static int AF_Release(struct inode *a_pstInode, struct fi
{
	LOG_INF("Start\n");

	if (g_pstAF_CurDrv) {
		g_pstAF_CurDrv->pAF_Release(a_pstInode, a
		g_pstAF_CurDrv = NULL;
	} else {
		spin_lock(&g_AF_SpinLock);
		g_s4AF_Opened = 0;
		spin_unlock(&g_AF_SpinLock);
	}

	if (strncmp(CONFIG_ARCH_MTK_PROJECT,
		"k6885v1_64_alpha", 16) == 0) {
		af_pinctrl_set(AF_PINCTRL_PIN_HWEN,
			AF_PINCTRL_PINSTATE_LOW);
	} else {
#if !defined(CONFIG_MTK_LEGACY)
		AFRegulatorCtrl(2);

#endif
	}
```

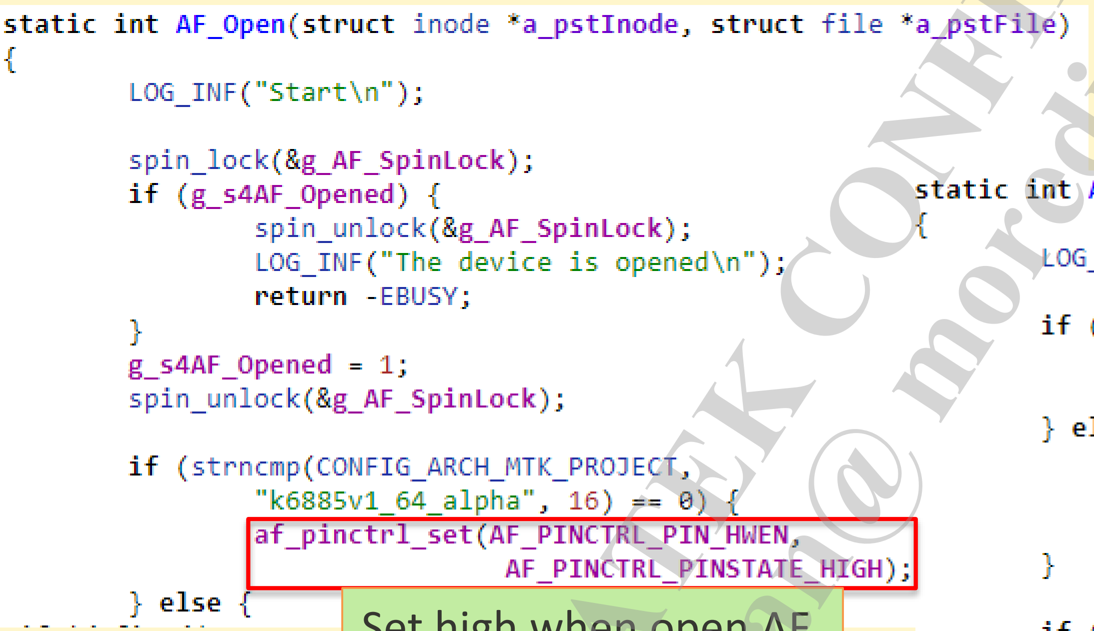
Disable regulator

MEDIATEK

Step 4-5-6
GPIO/PMIC supply power (Kernel-4.14)

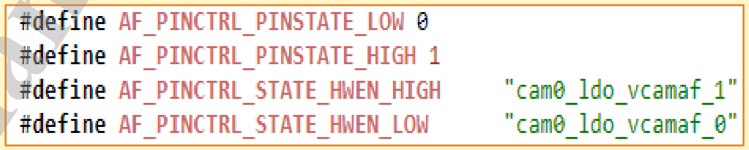
main_lens.c

- Configure gpio control

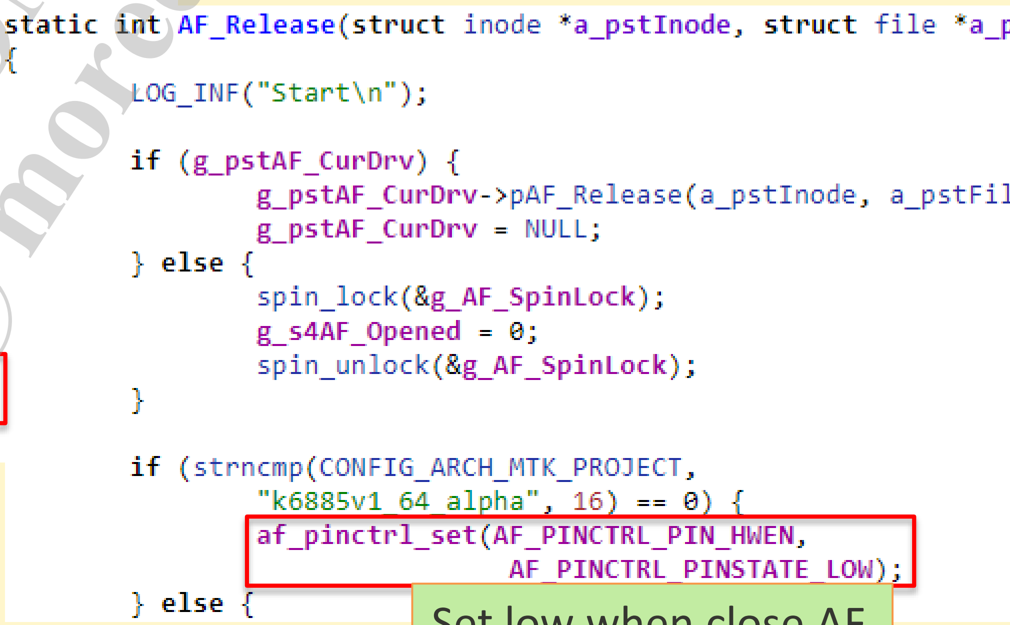


```
static int AF_Open(struct inode *a_pstInode, struct file *a_pstFile)
{
	LOG_INF("Start\n");

	spin_lock(&g_AF_SpinLock);
	if (g_s4AF_Opened) {
		spin_unlock(&g_AF_SpinLock);
		LOG_INF("The device is opened\n");
		return -EBUSY;
	}
	g_s4AF_Opened = 1;
	spin_unlock(&g_AF_SpinLock);

	if (strncmp(CONFIG_ARCH_MTK_PROJECT,
		"k6885v1_64_alpha", 16) == 0) {
		af_pinctrl_set(AF_PINCTRL_PIN_HWEN,
			AF_PINCTRL_PINSTATE_HIGH);
	} else {
```

Set high when open AF



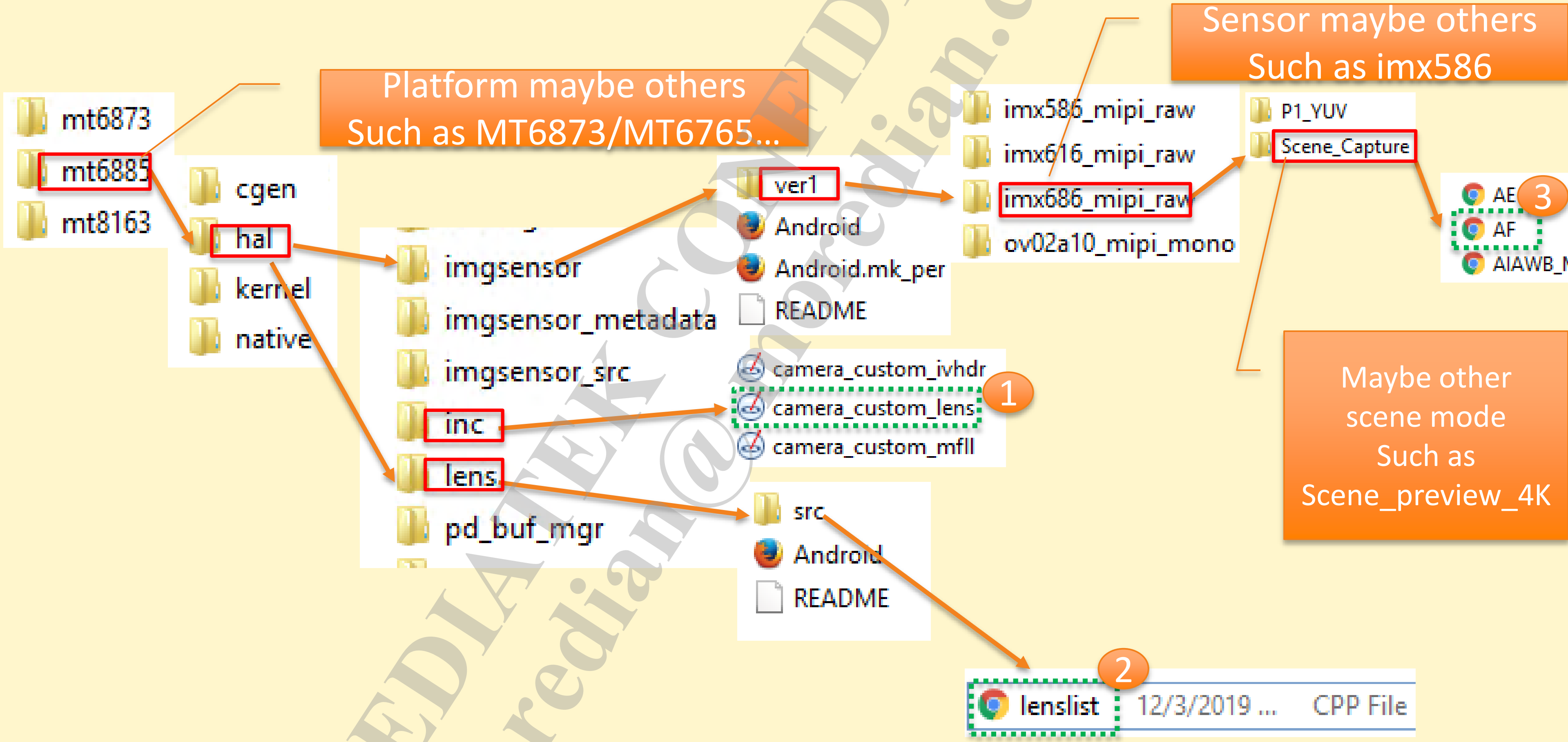
```
#define AF_PINCTRL_PINSTATE_LOW 0
#define AF_PINCTRL_PINSTATE_HIGH 1
#define AF_PINCTRL_STATE_HWEN_HIGH	"cam0_ldo_vcamaf_1"
#define AF_PINCTRL_STATE_HWEN_LOW	"cam0_ldo_vcamaf_0"
```



```
static int AF_Release(struct inode *a_pstInode, struct file *a_p
{
	LOG_INF("Start\n");

	if (g_pstAF_CurDrv) {
		g_pstAF_CurDrv->pAF_Release(a_pstInode, a_pstFil
		g_pstAF_CurDrv = NULL;
	} else {
		spin_lock(&g_AF_SpinLock);
		g_s4AF_Opened = 0;
		spin_unlock(&g_AF_SpinLock);
	}

	if (strncmp(CONFIG_ARCH_MTK_PROJECT,
		"k6885v1_64_alpha", 16) == 0) {
		af_pinctrl_set(AF_PINCTRL_PIN_HWEN,
			AF_PINCTRL_PINSTATE_LOW);
	} else {
```

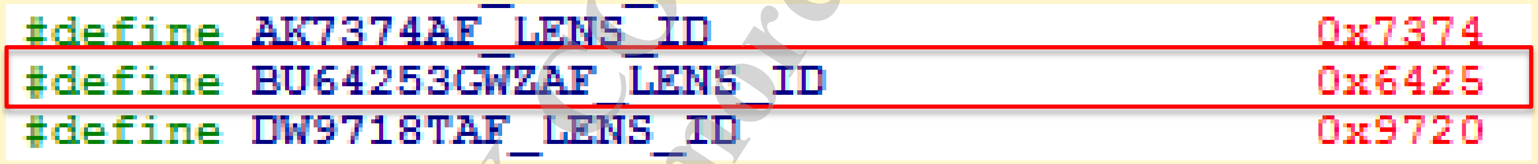
Set low when close AF

Step5
HAL configuration



Step5-1
Add lens ID

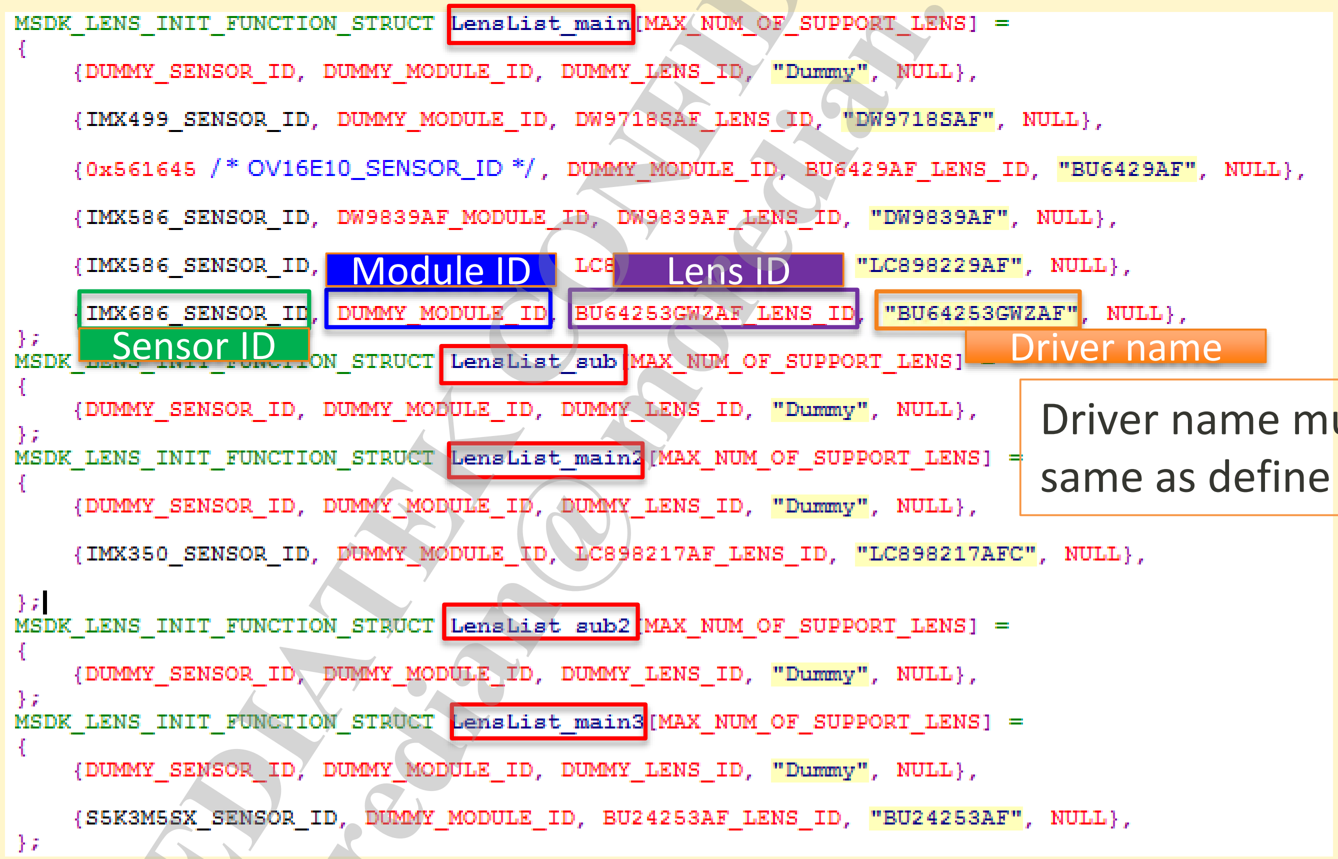
camera_custom_lens.h



```
#define AK7374AF_LENS_ID                                      0x7374
#define BU64253GWZAF_LENS_ID                                  0x6425
#define DW9718TAF_LENS_ID                                     0x9720
```

Step5-2
Add lens list

lenslist.cpp

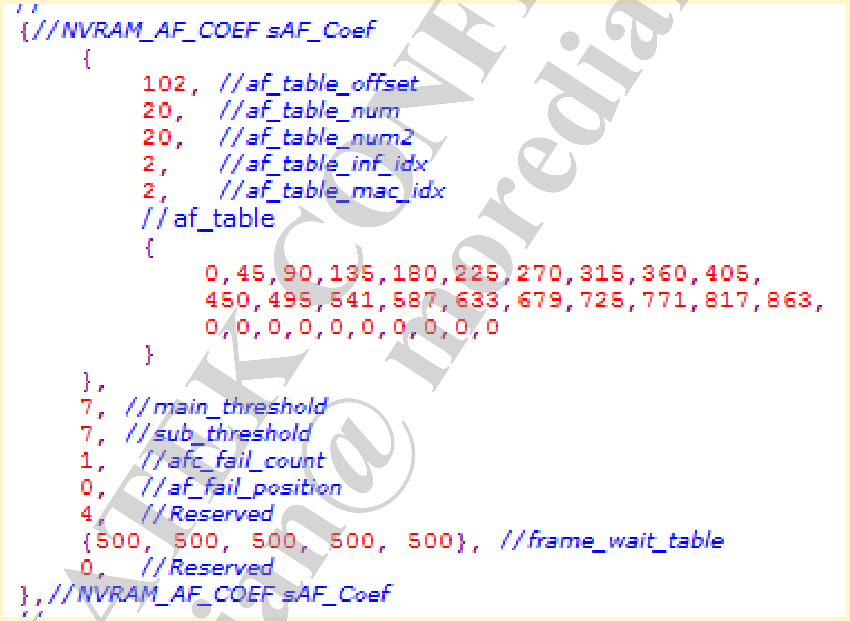


```
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_main[MAX_NUM_OF_SUPPORT_LENS] =
{
    {DUMMY_SENSOR_ID, DUMMY_MODULE_ID, DUMMY_LENS_ID, "Dummy", NULL},

    {IMX499_SENSOR_ID, DUMMY_MODULE_ID, DW9718SAF_LENS_ID, "DW9718SAF", NULL},

    {0x561645 /* OV16E10_SENSOR_ID */, DUMMY_MODULE_ID, BU6429AF_LENS_ID, "BU6429AF", NULL},

    {IMX586_SENSOR_ID, DW9839AF_MODULE_ID, DW9839AF_LENS_ID, "DW9839AF", NULL},

    {IMX586_SENSOR_ID,                   LC8                  "LC898229AF", NULL},

    {IMX686_SENSOR_ID, DUMMY_MODULE_ID, BU64253GWZAF_LENS_ID, "BU64253GWZAF", NULL},
};
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_sub[MAX_NUM_OF_SUPPORT_LENS] =
{
    {DUMMY_SENSOR_ID, DUMMY_MODULE_ID, DUMMY_LENS_ID, "Dummy", NULL},
};
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_main2[MAX_NUM_OF_SUPPORT_LENS] =
{
    {DUMMY_SENSOR_ID, DUMMY_MODULE_ID, DUMMY_LENS_ID, "Dummy", NULL},

    {IMX350_SENSOR_ID, DUMMY_MODULE_ID, LC898217AF_LENS_ID, "LC898217AFC", NULL},

};
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_sub2[MAX_NUM_OF_SUPPORT_LENS] =
{
    {DUMMY_SENSOR_ID, DUMMY_MODULE_ID, DUMMY_LENS_ID, "Dummy", NULL},
};
MSDK_LENS_INIT_FUNCTION_STRUCT LensList_main3[MAX_NUM_OF_SUPPORT_LENS] =
{
    {DUMMY_SENSOR_ID, DUMMY_MODULE_ID, DUMMY_LENS_ID, "Dummy", NULL},

    {S5K3M5SX_SENSOR_ID, DUMMY_MODULE_ID, BU24253AF_LENS_ID, "BU24253AF", NULL},
};
```

Driver name must be the same as define in lens_info.h

Step 5-3
Modify AF table in AF.cpp

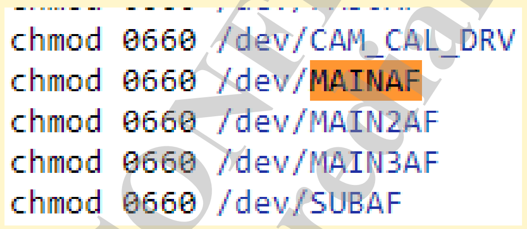
Copy the parameter of the project has been MP
Then check the AF table is suitable with the tuning owner



```
//
{//NVRAM_AF_COEF sAF_Coef
    {
        102, //af_table_offset
        20,  //af_table_num
        20,  //af_table_num2
        2,   //af_table_inf_idx
        2,   //af_table_mac_idx
        //af_table
        {
            0,45,90,135,180,225,270,315,360,405,
            450,495,541,587,633,679,725,771,817,863,
            0,0,0,0,0,0,0,0,0,0
        }
    },
    7, //main_threshold
    7, //sub_threshold
    1,  //afc_fail_count
    0,  //af_fail_position
    4,  //Reserved
    {500, 500, 500, 500, 500}, //frame_wait_table
    0,  //Reserved
},//NVRAM_AF_COEF sAF_Coef
//
```

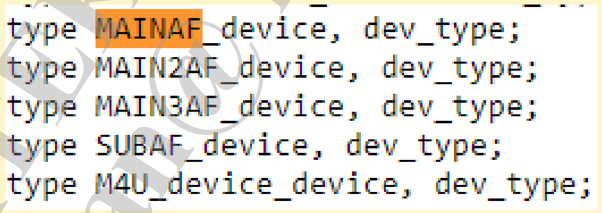
Step 6
Selinux policy configuration

- alps/device/mediatek/[$platform]/init.[$platform].rc



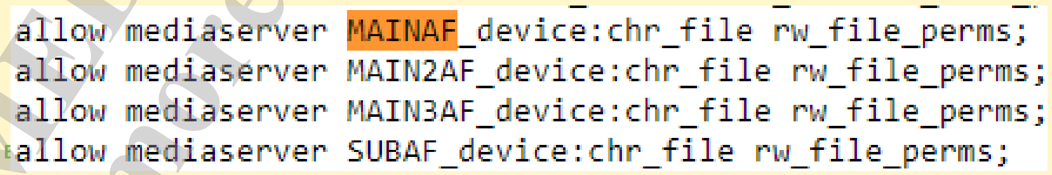
```
chmod 0660 /dev/CAM_CAL_DRV
chmod 0660 /dev/MAINAF
chmod 0660 /dev/MAIN2AF
chmod 0660 /dev/MAIN3AF
chmod 0660 /dev/SUBAF
```

- alps/device/mediatek/sepolicy/basic/non_plat/device.te



```
type MAINAF_device, dev_type;
type MAIN2AF_device, dev_type;
type MAIN3AF_device, dev_type;
type SUBAF_device, dev_type;
type M4U_device_device, dev_type;
```

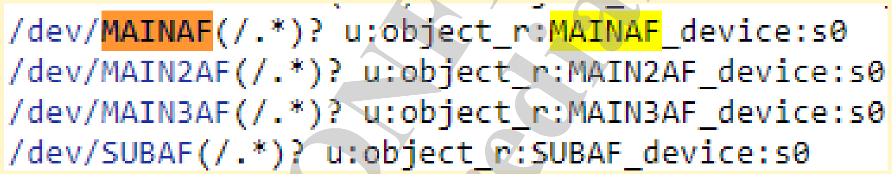
- alps/device/mediatek/sepolicy/basic/non_plat/mediaserver.te



```
allow mediaserver MAINAF_device:chr_file rw_file_perms;
allow mediaserver MAIN2AF_device:chr_file rw_file_perms;
allow mediaserver MAIN3AF_device:chr_file rw_file_perms;
allow mediaserver SUBAF_device:chr_file rw_file_perms;
```

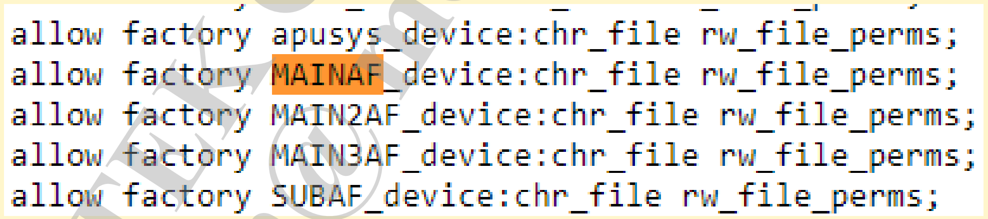
Step 6
Selinux policy configuration

- alps/device/mediatek/sepolicy/basic/non_plat/file_contexts



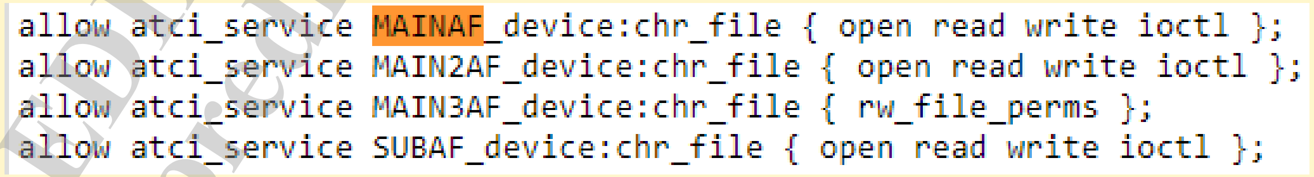
```
/dev/MAINAF(/.*)? u:object_r:MAINAF_device:s0
/dev/MAIN2AF(/.*)? u:object_r:MAIN2AF_device:s0
/dev/MAIN3AF(/.*)? u:object_r:MAIN3AF_device:s0
/dev/SUBAF(/.*)? u:object_r:SUBAF_device:s0
```

- alps/device/mediatek/sepolicy/basic/non_plat/factory.te



```
allow factory apusys_device:chr_file rw_file_perms;
allow factory MAINAF_device:chr_file rw_file_perms;
allow factory MAIN2AF_device:chr_file rw_file_perms;
allow factory MAIN3AF_device:chr_file rw_file_perms;
allow factory SUBAF_device:chr_file rw_file_perms;
```

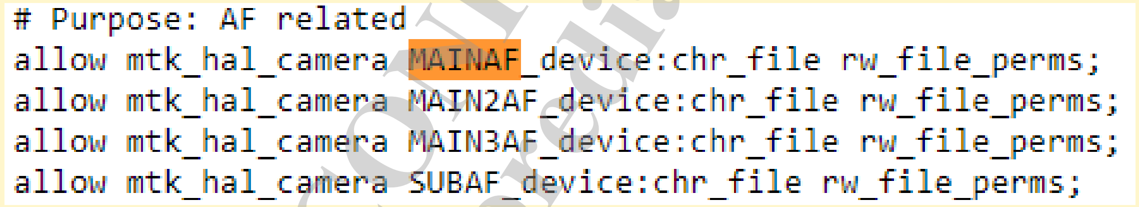
- alps/device/mediatek/sepolicy/basic/non_plat/atci_service.te



```
allow atci_service MAINAF_device:chr_file { open read write ioctl };
allow atci_service MAIN2AF_device:chr_file { open read write ioctl };
allow atci_service MAIN3AF_device:chr_file { rw_file_perms };
allow atci_service SUBAF_device:chr_file { open read write ioctl };
```

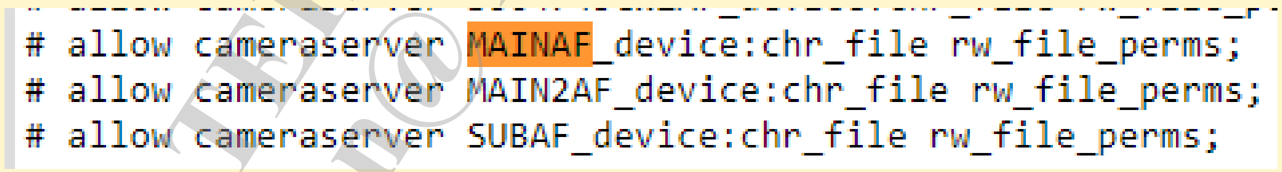
Step 6
Selinux policy configuration

- alps/device/mediatek/sepolicy/basic/non_plat/mtk_hal_camera.te



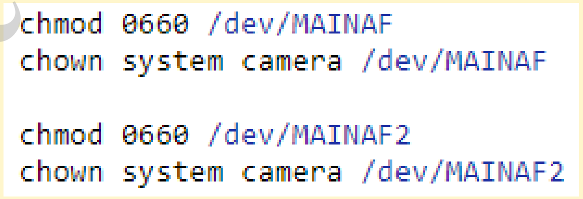
```
# Purpose: AF related
allow mtk_hal_camera MAINAF_device:chr_file rw_file_perms;
allow mtk_hal_camera MAIN2AF_device:chr_file rw_file_perms;
allow mtk_hal_camera MAIN3AF_device:chr_file rw_file_perms;
allow mtk_hal_camera SUBAF_device:chr_file rw_file_perms;
```

- alps/device/mediatek/sepolicy/basic/non_plat/cameraserver.te



```
# allow cameraserver MAINAF_device:chr_file rw_file_perms;
# allow cameraserver MAIN2AF_device:chr_file rw_file_perms;
# allow cameraserver SUBAF_device:chr_file rw_file_perms;
```

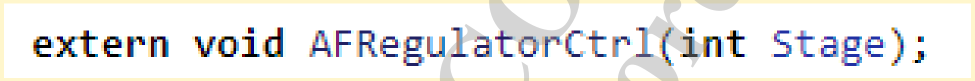
- alps/device/mediatekprojects/[$project]/init.project.rc



```
chmod 0660 /dev/MAINAF
chown system camera /dev/MAINAF

chmod 0660 /dev/MAINAF2
chown system camera /dev/MAINAF2
```

Summary

- If porting main2af or subaf, you can use main2_lens.c or sub_lens.c to replace man_lens.c
- APIs of controlling power only exist in main_lens, other dev must extern the API in lens_list.h



```
extern void AFRegulatorCtrl(int Stage);
```

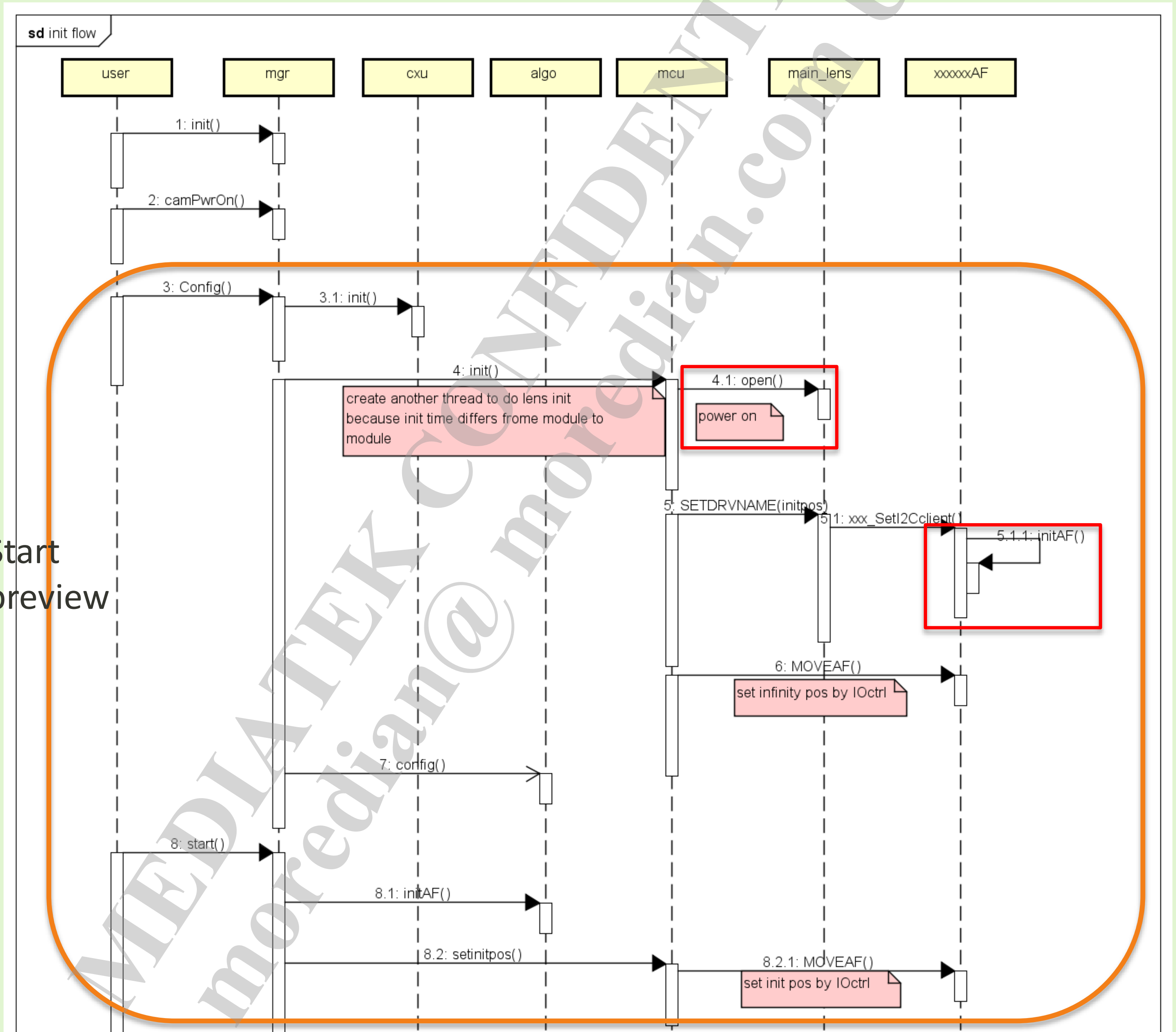
- If add a new dev, such as MAIN4AF, you can clone a driver from main2AF , and then configure it

MEDIATEK

Log Analysis

- ❑ Init(Power on) log
- ❑ Uninit(Power off) log

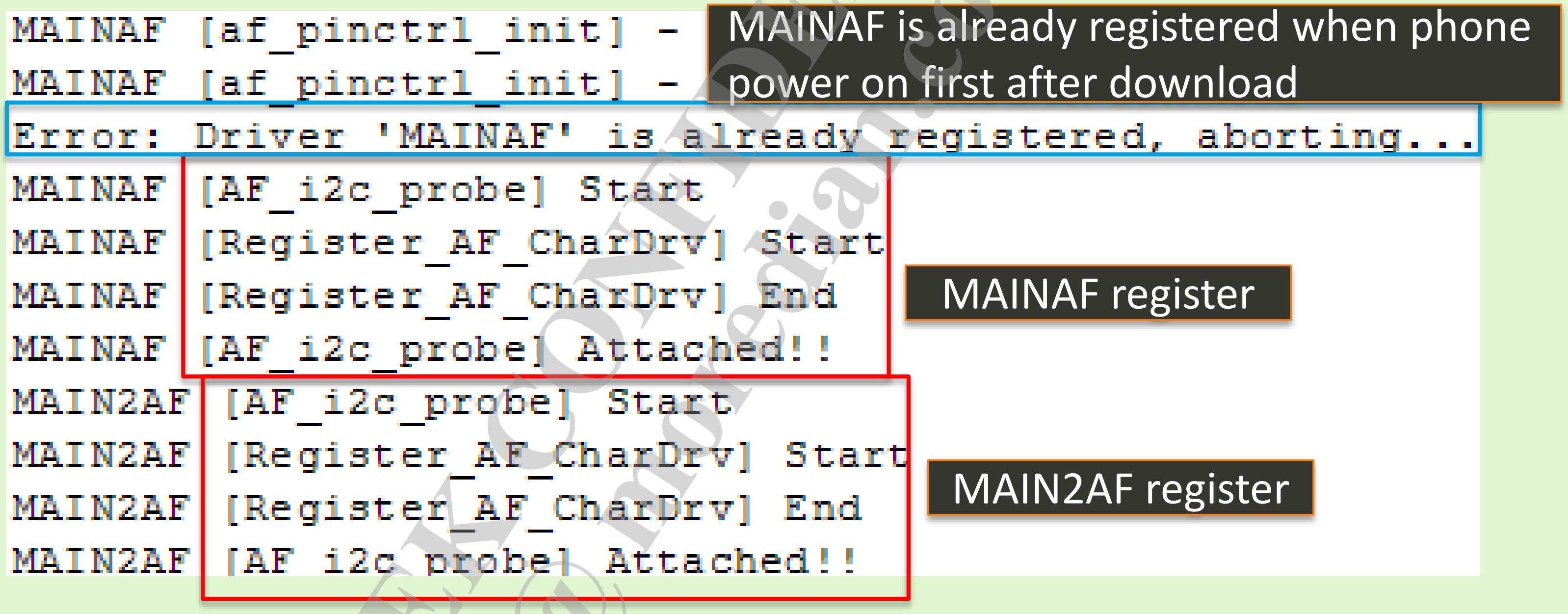
Init Flow



Start preview

MEDIATEK

Device Register log when phone power on



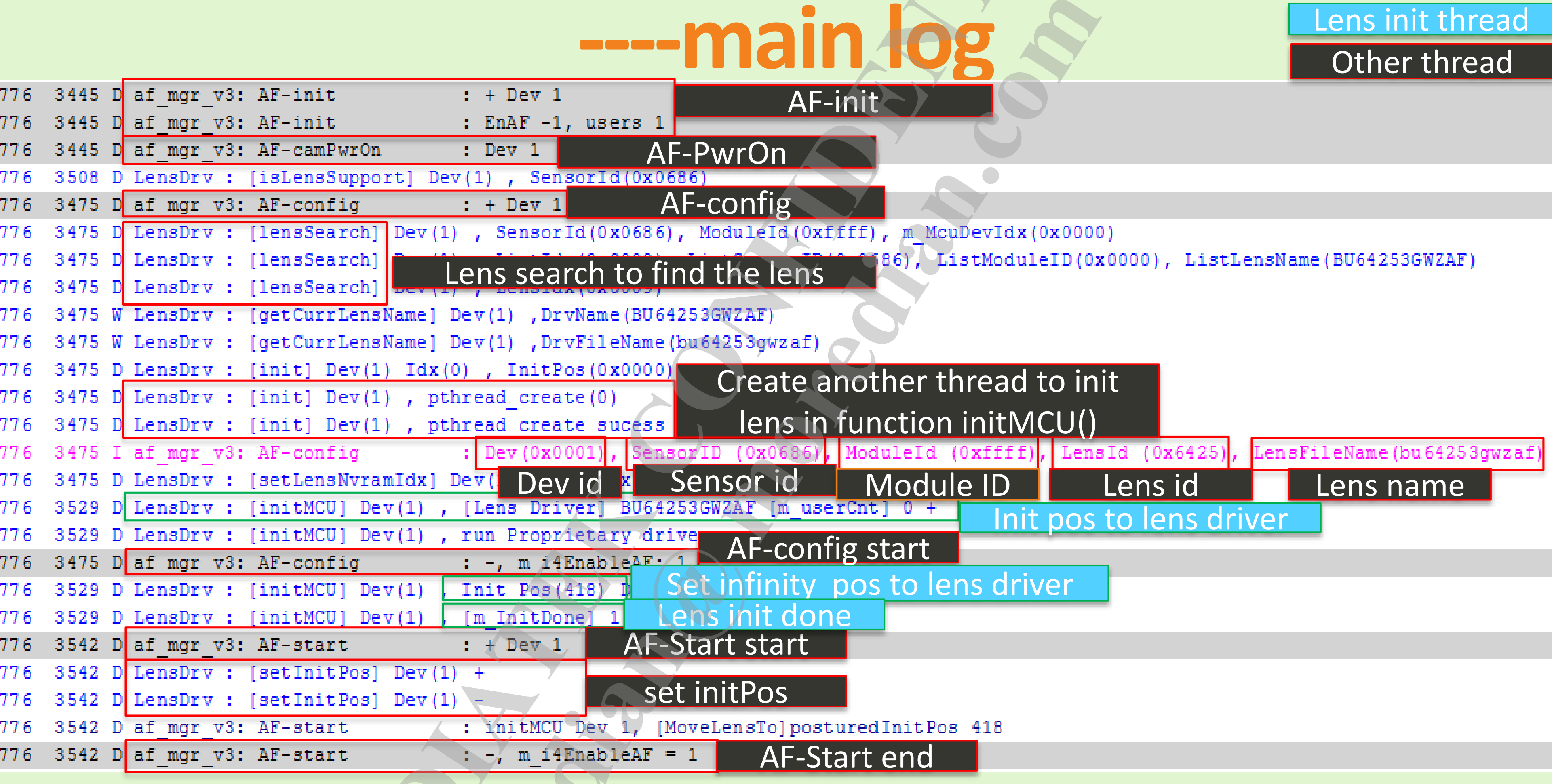
```
MAINAF [af_pinctrl_init] -
MAINAF [af_pinctrl_init] -
Error: Driver 'MAINAF' is already registered, aborting...
MAINAF [AF_i2c_probe] Start
MAINAF [Register_AF_CharDrv] Start
MAINAF [Register_AF_CharDrv] End
MAINAF [AF_i2c_probe] Attached!!
MAIN2AF [AF_i2c_probe] Start
MAIN2AF [Register_AF_CharDrv] Start
MAIN2AF [Register_AF_CharDrv] End
MAIN2AF [AF_i2c_probe] Attached!!
```

MAINAF is already registered when phone power on first after download

MAINAF register

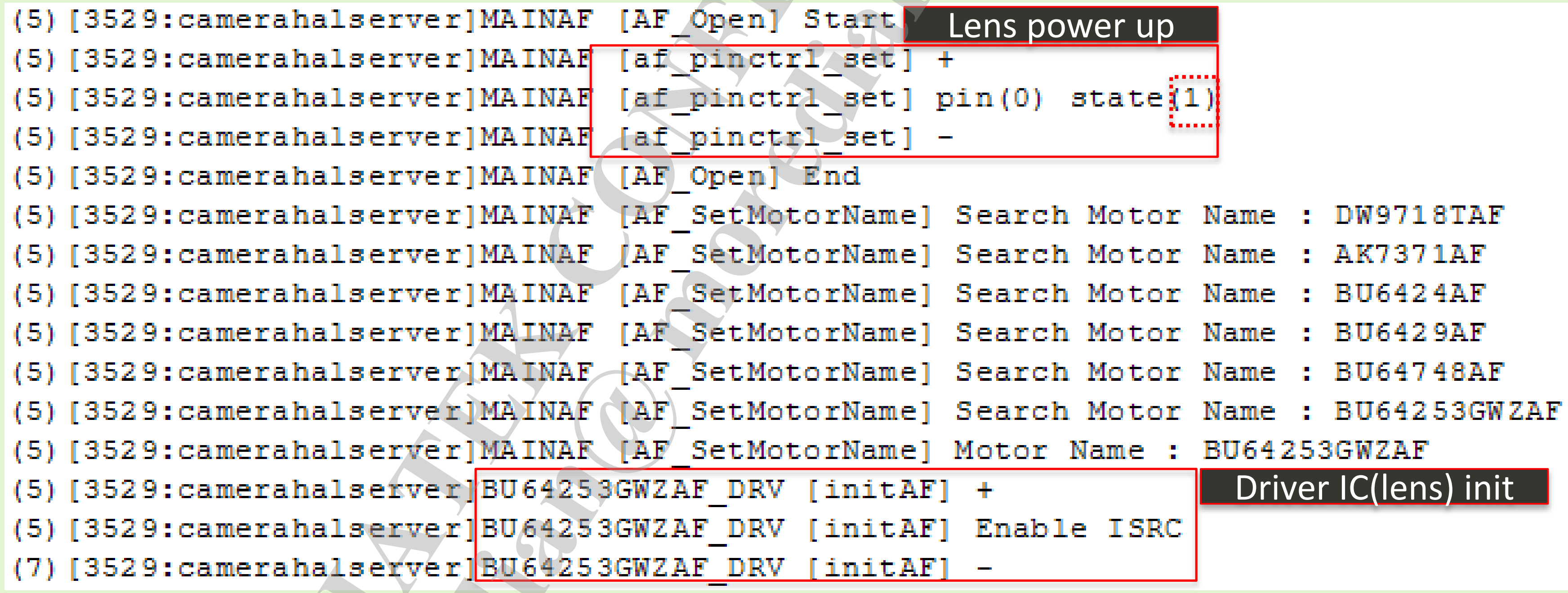
MAIN2AF register

AF init when camera power on ----main log



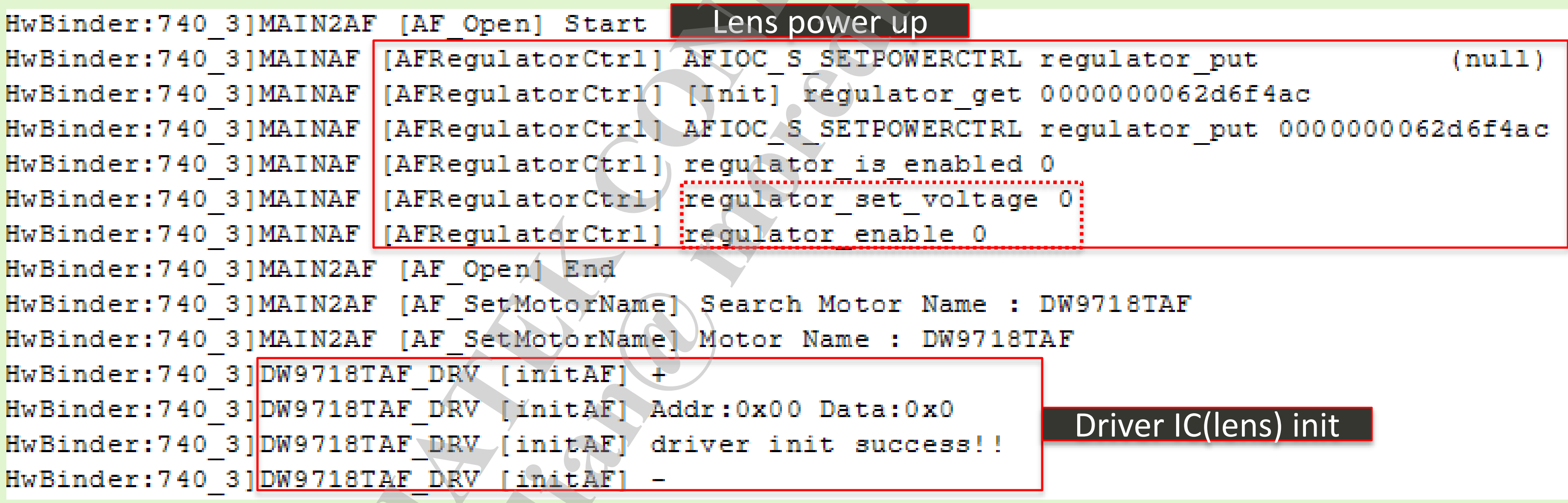
```
776   3445 D af_mgr_v3: AF-init               : + Dev 1
776   3445 D af_mgr_v3: AF-init               : EnAF -1, users 1
776   3445 D af_mgr_v3: AF-camPwrOn           : Dev 1
776   3508 D LensDrv : [isLensSupport] Dev(1) , SensorId(0x0686)
776   3475 D af mgr v3: AF-config             : + Dev 1
776   3475 D LensDrv : [lensSearch] Dev(1) , SensorId(0x0686), ModuleId(0xffff), m_McuDevIdx(0x0000)
776   3475 D LensDrv : [lensSearch]                                    86), ListModuleID(0x0000), ListLensName(BU64253GWZAF)
776   3475 D LensDrv : [lensSearch]
776   3475 W LensDrv : [getCurrLensName] Dev(1) ,DrvName(BU64253GWZAF)
776   3475 W LensDrv : [getCurrLensName] Dev(1) ,DrvFileName(bu64253gwzaf)
776   3475 D LensDrv : [init] Dev(1) Idx(0) , InitPos(0x0000)
776   3475 D LensDrv : [init] Dev(1) , pthread_create(0)
776   3475 D LensDrv : [init] Dev(1) , pthread create sucess
776   3475 I af_mgr_v3: AF-config             : Dev(0x0001), SensorID (0x0686), ModuleId (0xffff), LensId (0x6425), LensFileName(bu64253gwzaf)
776   3475 D LensDrv : [setLensNvramIdx] Dev(
776   3529 D LensDrv : [initMCU] Dev(1) , [Lens Driver] BU64253GWZAF [m_userCnt] 0 +
776   3529 D LensDrv : [initMCU] Dev(1) , run Proprietary drive
776   3475 D af mgr v3: AF-config             : -, m i4EnableAF: 1
776   3529 D LensDrv : [initMCU] Dev(1) , Init Pos(418) D
776   3529 D LensDrv : [initMCU] Dev(1) , [m_InitDone] 1
776   3542 D af_mgr_v3: AF-start              : + Dev 1
776   3542 D LensDrv : [setInitPos] Dev(1) +
776   3542 D LensDrv : [setInitPos] Dev(1) -
776   3542 D af_mgr_v3: AF-start              : initMCU Dev 1, [MoveLensTo]posturedInitPos 418
776   3542 D af_mgr_v3: AF-start              : -, m_i4EnableAF = 1
```

AF init when camera power on ----Kernel log(GPIO pinctrl)



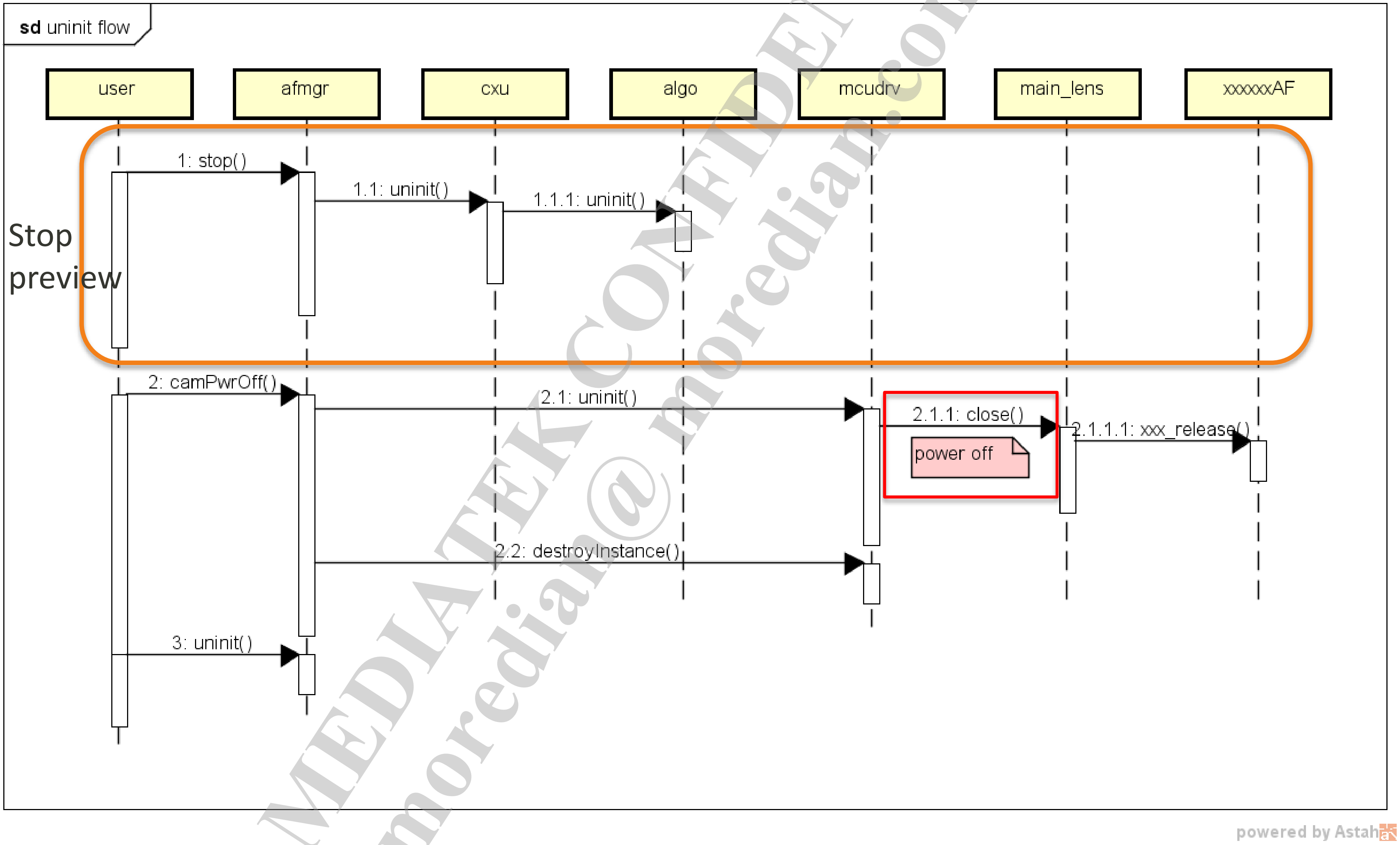
```
(5) [3529:camerahalserver]MAINAF [AF_Open] Start
(5) [3529:camerahalserver]MAINAF [af_pinctrl_set] +
(5) [3529:camerahalserver]MAINAF [af_pinctrl_set] pin(0) state(1)
(5) [3529:camerahalserver]MAINAF [af_pinctrl_set] -
(5) [3529:camerahalserver]MAINAF [AF_Open] End
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : DW9718TAF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : AK7371AF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : BU6424AF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : BU6429AF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : BU64748AF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Search Motor Name : BU64253GWZAF
(5) [3529:camerahalserver]MAINAF [AF_SetMotorName] Motor Name : BU64253GWZAF
(5) [3529:camerahalserver]BU64253GWZAF_DRV [initAF] +
(5) [3529:camerahalserver]BU64253GWZAF_DRV [initAF] Enable ISRC
(7) [3529:camerahalserver]BU64253GWZAF_DRV [initAF] -
```

AF init when camera power on ----Kernel log(PMIC)

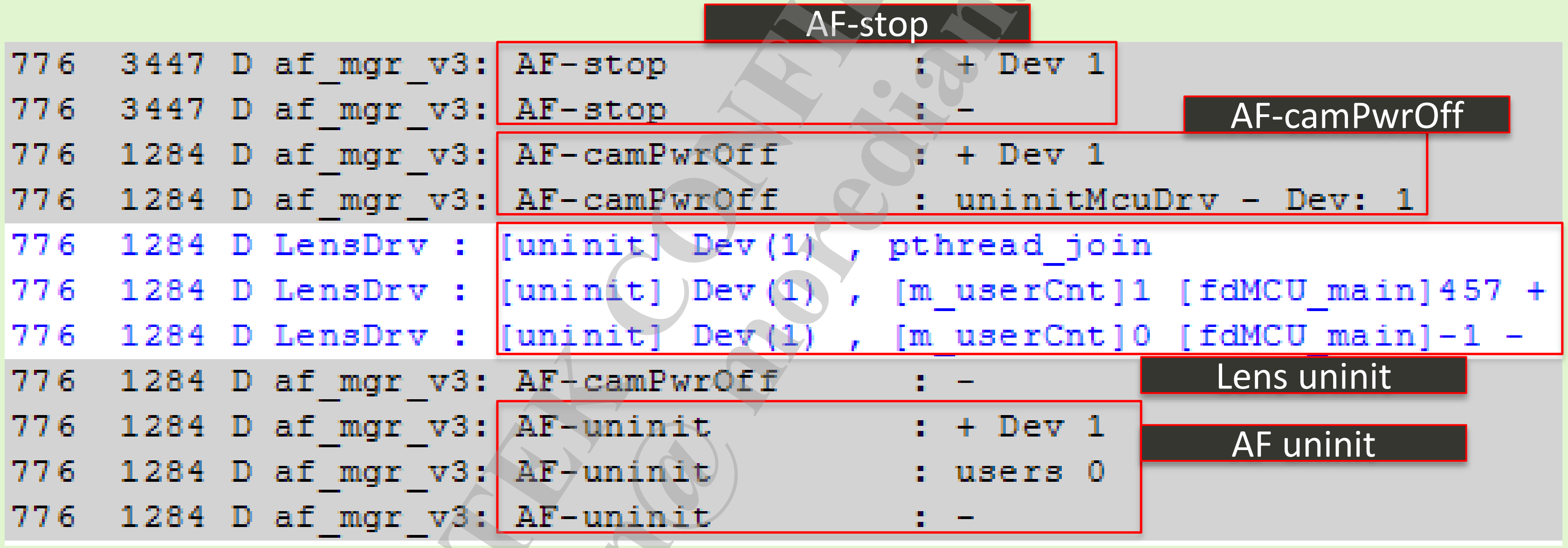


```
HwBinder:740_3]MAIN2AF [AF_Open] Start
HwBinder:740_3]MAINAF [AFRegulatorCtrl] AFIOC_S_SETPOWERCTRL regulator_put          (null)
HwBinder:740_3]MAINAF [AFRegulatorCtrl] [Init] regulator_get 000000062d6f4ac
HwBinder:740_3]MAINAF [AFRegulatorCtrl] AFIOC_S_SETPOWERCTRL regulator_put 000000062d6f4ac
HwBinder:740_3]MAINAF [AFRegulatorCtrl] regulator_is_enabled 0
HwBinder:740_3]MAINAF [AFRegulatorCtrl] regulator_set_voltage 0
HwBinder:740_3]MAINAF [AFRegulatorCtrl] regulator_enable 0
HwBinder:740_3]MAIN2AF [AF_Open] End
HwBinder:740_3]MAIN2AF [AF_SetMotorName] Search Motor Name : DW9718TAF
HwBinder:740_3]MAIN2AF [AF_SetMotorName] Motor Name : DW9718TAF
HwBinder:740_3]DW9718TAF_DRV [initAF] +
HwBinder:740_3]DW9718TAF_DRV [initAF] Addr:0x00 Data:0x0
HwBinder:740_3]DW9718TAF_DRV [initAF] driver init success!!
HwBinder:740_3]DW9718TAF_DRV [initAF] -
```

Uninit Flow

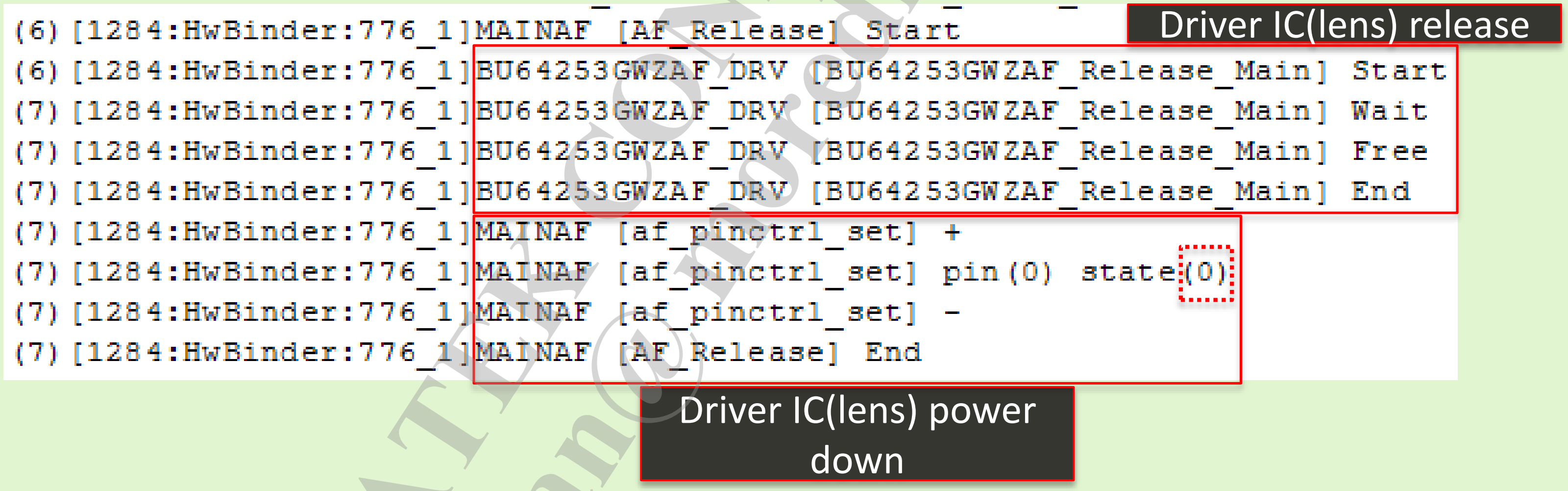

sd uninit flow
user
afmgr
cxu
algo
mcudrv
main_lens
xxxxxxAF
Stop preview
1: stop()
1.1: uninit()
1.1.1: uninit()
2: camPwrOff()
2.1: uninit()
2.1.1: close()
power off
2.1.1.1: xxx_release()
2.2: destroyInstance()
3: uninit()
powered by Astah

AF unint when camera power off ----main log



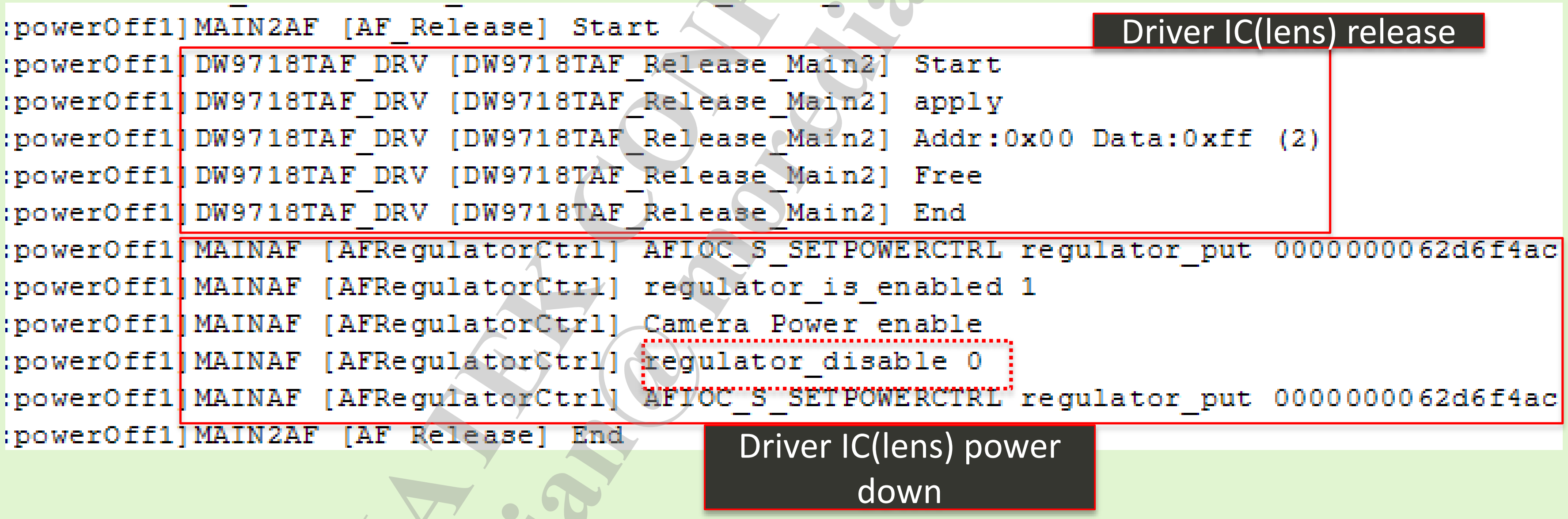
```
776   3447 D af_mgr_v3: AF-stop             : + Dev 1
776   3447 D af_mgr_v3: AF-stop             : -
776   1284 D af_mgr_v3: AF-camPwrOff        : + Dev 1
776   1284 D af_mgr_v3: AF-camPwrOff        : uninitMcuDrv - Dev: 1
776   1284 D LensDrv  : [uninit] Dev(1) , pthread_join
776   1284 D LensDrv  : [uninit] Dev(1) , [m_userCnt]1 [fdMCU_main]457 +
776   1284 D LensDrv  : [uninit] Dev(1) , [m_userCnt]0 [fdMCU_main]-1 -
776   1284 D af_mgr_v3: AF-camPwrOff        : -
776   1284 D af_mgr_v3: AF-uninit           : + Dev 1
776   1284 D af_mgr_v3: AF-uninit           : users 0
776   1284 D af_mgr_v3: AF-uninit           : -
```

AF uninit when camera power off ----Kernel log(GPIO pinctrl)



```
(6) [1284:HwBinder:776_1]MAINAF [AF_Release] Start
(6) [1284:HwBinder:776_1]BU64253GWZAF_DRV [BU64253GWZAF_Release_Main] Start
(7) [1284:HwBinder:776_1]BU64253GWZAF_DRV [BU64253GWZAF_Release_Main] Wait
(7) [1284:HwBinder:776_1]BU64253GWZAF_DRV [BU64253GWZAF_Release_Main] Free
(7) [1284:HwBinder:776_1]BU64253GWZAF_DRV [BU64253GWZAF_Release_Main] End
(7) [1284:HwBinder:776_1]MAINAF [af_pinctrl_set] +
(7) [1284:HwBinder:776_1]MAINAF [af_pinctrl_set] pin(0) state(0)
(7) [1284:HwBinder:776_1]MAINAF [af_pinctrl_set] -
(7) [1284:HwBinder:776_1]MAINAF [AF_Release] End
```

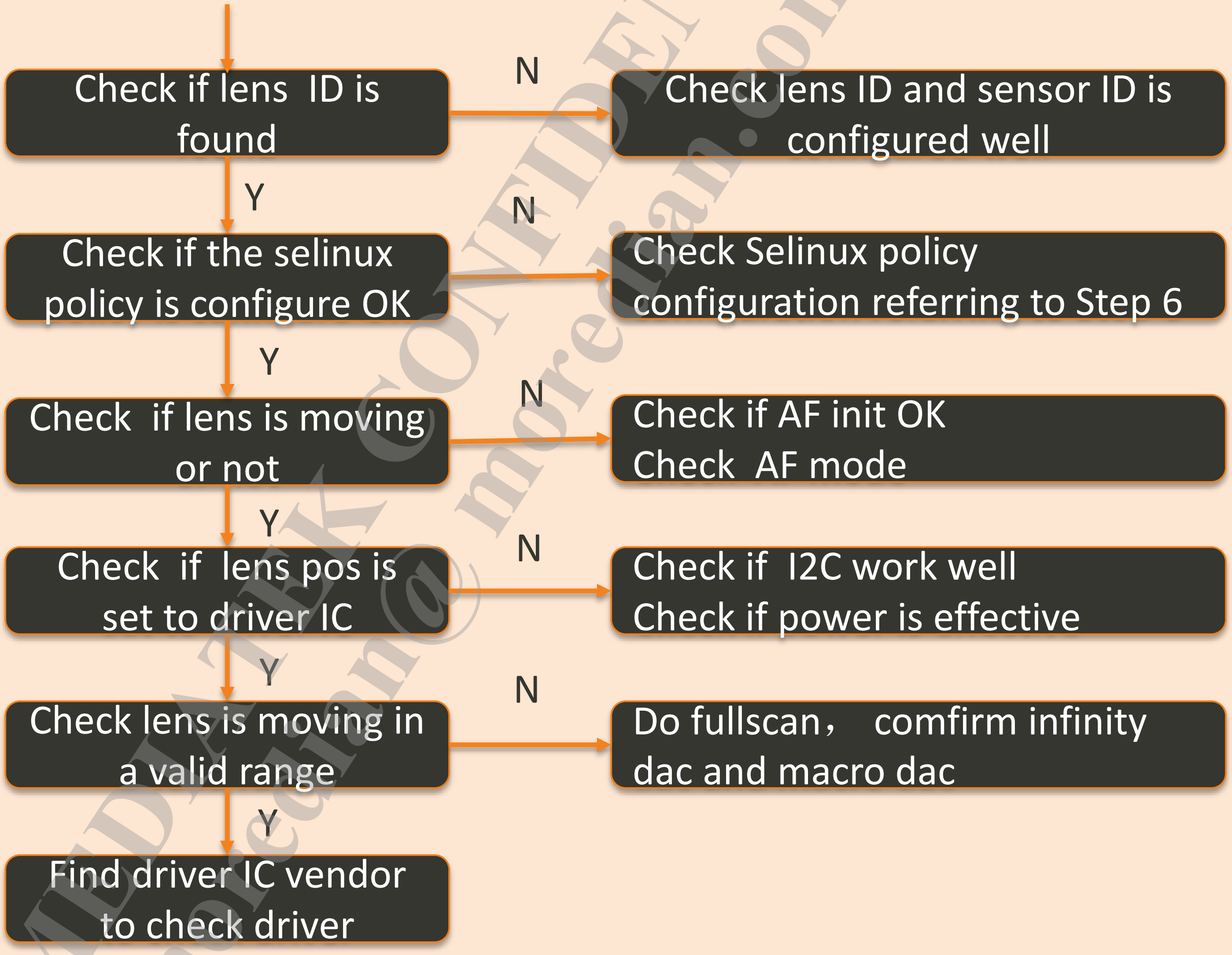
AF uninit when camera power off ----Kernel(PMIC)



```
:powerOff1]MAIN2AF [AF_Release] Start
:powerOff1]DW9718TAF_DRV [DW9718TAF_Release_Main2] Start
:powerOff1]DW9718TAF_DRV [DW9718TAF_Release_Main2] apply
:powerOff1]DW9718TAF_DRV [DW9718TAF_Release_Main2] Addr:0x00 Data:0xff (2)
:powerOff1]DW9718TAF_DRV [DW9718TAF_Release_Main2] Free
:powerOff1]DW9718TAF_DRV [DW9718TAF_Release_Main2] End
:powerOff1]MAINAF [AFRegulatorCtrl] AFIOC_S_SETPOWERCTRL regulator_put 000000062d6f4ac
:powerOff1]MAINAF [AFRegulatorCtrl] regulator_is_enabled 1
:powerOff1]MAINAF [AFRegulatorCtrl] Camera Power enable
:powerOff1]MAINAF [AFRegulatorCtrl] regulator_disable 0
:powerOff1]MAINAF [AFRegulatorCtrl] AFIOC_S_SETPOWERCTRL regulator_put 000000062d6f4ac
:powerOff1]MAIN2AF [AF_Release] End
```

Debug Check Points

Debug Check Points


Check if lens ID is found
N
Check lens ID and sensor ID is configured well
Y
Check if the selinux policy is configure OK
N
Check Selinux policy configuration referring to Step 6
Y
Check if lens is moving or not
N
Check if AF init OK
Check AF mode
Y
Check if lens pos is set to driver IC
N
Check if I2C work well
Check if power is effective
Y
Check lens is moving in a valid range
N
Do fullscan， comfirm infinity dac and macro dac
Y
Find driver IC vendor to check driver

Debug Check Point-1
Check if lens ID is found

- NG log

```
af_mgr_v3: AF-config        : Dev(0x0001), SensorID (0x0586), ModuleId (0xffff), LensId (0xffff), LensFileName(dummy)
```

If lens id is 0xffff, it means sensor ID 0x0586 has not a matched lens

- OK log

```
af_mgr_v3: AF-config        : Dev(0x0001), SensorID (0x0686), ModuleId (0xffff), LensId (0x6425), LensFileName(bu64253gwzaf)
```

- debug method

  Check if Lenslist are configured referring to Step5-1 and Step5-2

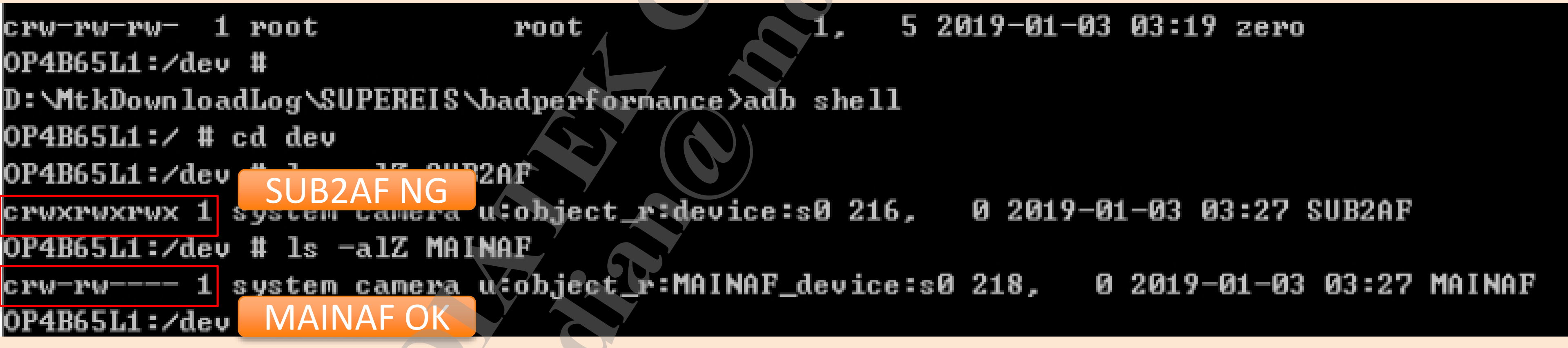
Debug Check Point-2
Check if the selinux policy is configure OK

- How to check

adb shell -> cd dev -> input the following command

MAIN AF can be MAIN2AF or MAIN3AF or other

Comand: ls -alZ MAINAF



```
crw-rw-rw-   1 root           root              1,    5 2019-01-03 03:19 zero
OP4B65L1:/dev #
D:\MtkDownloadLog\SUPEREIS\badperformance>adb shell
OP4B65L1:/ # cd dev
OP4B65L1:/dev # ls -alZ SUB2AF
crwxrwxrwx 1 system camera u:object_r:device:s0 216,   0 2019-01-03 03:27 SUB2AF
OP4B65L1:/dev # ls -alZ MAINAF
crw-rw---- 1 system camera u:object_r:MAINAF_device:s0 218,   0 2019-01-03 03:27 MAINAF
OP4B65L1:/dev
```

Debug Check Point-2 selinux policy is set OK

- NG log

```
20228 01-02 11:53:30.026822 16898 17161 W LensDrv : [initMCU] Dev(8) ,
Lens error opening (Permission denied)
```

- Debug method

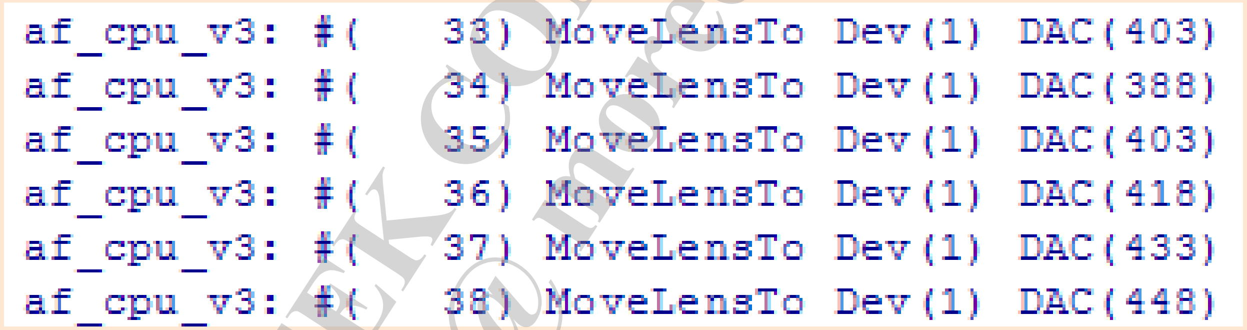
Check Selinux policy configuration referring to Step 6

Debug Check Point-3
Check if lens is moving or not

- Find “movelensto”

If there has not any logs, lens can’t move

OK log:



```
af_cpu_v3: #(    33) MoveLensTo Dev(1) DAC(403)
af_cpu_v3: #(    34) MoveLensTo Dev(1) DAC(388)
af_cpu_v3: #(    35) MoveLensTo Dev(1) DAC(403)
af_cpu_v3: #(    36) MoveLensTo Dev(1) DAC(418)
af_cpu_v3: #(    37) MoveLensTo Dev(1) DAC(433)
af_cpu_v3: #(    38) MoveLensTo Dev(1) DAC(448)
```

Debug Check Point-3
Check if lens is moving or not

- Debug Method
  - Check AF is init OK, check following log

NG log：

```
13140 05-29 14:19:54.570537   776  3475 D af_mgr_v3: AF-config        : -, m_i4EnableAF: -1
14039 05-29 14:19:54.607960   776  3542 D af_mgr_v3: AF-start        : -, m_i4EnableAF = -1
```

OK log:

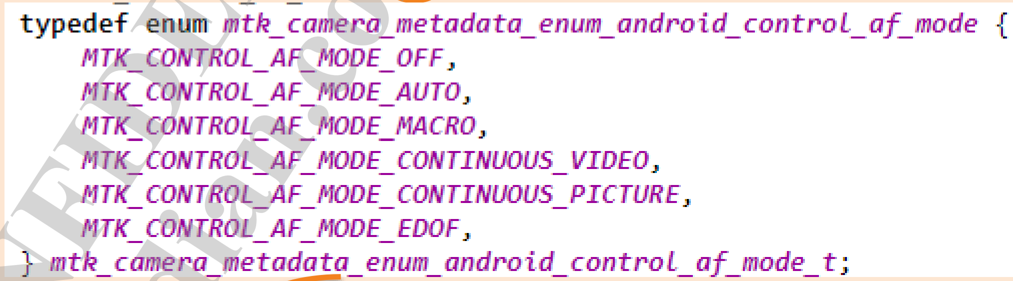
```
13140 05-29 14:19:54.570537   776  3475 D af_mgr_v3: AF-config        : -, m_i4EnableAF: 1
14039 05-29 14:19:54.607960   776  3542 D af_mgr_v3: AF-start        : -, m_i4EnableAF = 1
```

If NG: there will be some error log output in function start() and config(), you can debug it according the error log

Debug Check Point-3
Check if lens is moving or not

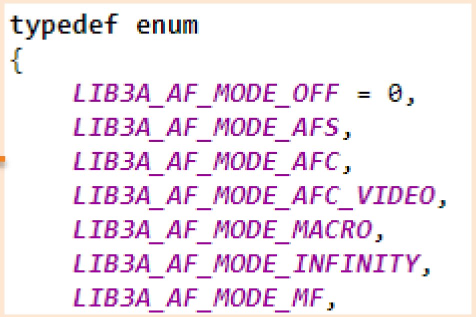
- Debug Method
  - Check AF mode

OK log:



```
typedef enum mtk_camera_metadata_enum_android_control_af_mode {
    MTK_CONTROL_AF_MODE_OFF,
    MTK_CONTROL_AF_MODE_AUTO,
    MTK_CONTROL_AF_MODE_MACRO,
    MTK_CONTROL_AF_MODE_CONTINUOUS_VIDEO,
    MTK_CONTROL_AF_MODE_CONTINUOUS_PICTURE,
    MTK_CONTROL_AF_MODE_EDOF,
} mtk_camera_metadata_enum_android_control_af_mode_t;
```

```
Hal3Av3 : [parseMeta] MTK_CONTROL_AF_MODE(5 -> 4)
af_mgr_v3: #(   4,   1) cmd-setAFMode Dev(1):ctl_afmode(4)
af_mgr_v3: #(   4,   1) cmd-setAFMode Dev(1):lib_afmode 0->2
AfAlgoSt: [Cmd_setAFMode][Mode]2
```



```
typedef enum
{
    LIB3A_AF_MODE_OFF = 0,
    LIB3A_AF_MODE_AFS,
    LIB3A_AF_MODE_AFC,
    LIB3A_AF_MODE_AFC_VIDEO,
    LIB3A_AF_MODE_MACRO,
    LIB3A_AF_MODE_INFINITY,
    LIB3A_AF_MODE_MF,
```

NG log: AF will move only one time

```
Hal3Av3 : [parseMeta] MTK_CONTROL_AF_MODE(5 -> 0)
af_mgr_v3: #(   4,   1) cmd-setAFMode Dev(1):ctl_afmode(6)
af_mgr_v3: #(   4,   1) cmd-setAFMode Dev(1):lib_afmode 0->6
AfAlgoSt: [Cmd_setAFMode][Mode]6
```

If AF mode is not expected, you can ask AP owner of the customer to check

AF mode 3A	AF mode AF lib	Note
MTK_CONTROL_AF_MODE_OFF	LIB3A_AF_MODE_MF	manual AF, lens only moves to target position
MTK_CONTROL_AF_MODE_AUTO	LIB3A_AF_MODE_AFS	single trigger, such as touch AF
MTK_CONTROL_AF_MODE_MACRO	LIB3A_AF_MODE_MACRO	closer single trigger, such as touch AF
MTK_CONTROL_AF_MODE_CONTINUOUS_VIDEO	LIB3A_AF_MODE_AFC_VIDEO	more smooth continuous auto focus
MTK_CONTROL_AF_MODE_CONTINUOUS_PICTURE	LIB3A_AF_MODE_AFC	continuous auto focus, AF is triggered by algo
MTK_CONTROL_AF_MODE_EDOF	LIB3A_AF_MODE_OFF	lens will not move

Debug Check Point-4
Check if lens pos is set to driver IC

- Add debug log in MOVEAF of driver IC(lens) driver
  - Add code to print the a_u4Position
  - Add code to read the register of position and compare if the value is equal to a_u4Position

If the value is not equal to a_u4Position , you need to check kernel log to confirm if I2C works well

For P90 , you must enter the following adb command to catch debug log

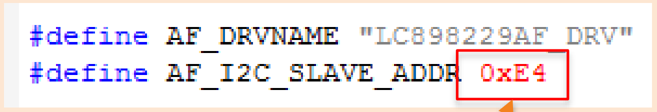
```
adb shell setprop vendor.debug.forceCPUAF.enable 1
```

Debug Check Point-4
Check if lens pos is set to driver IC

- Check kernel log to confirm if I2C work well
  - find "transfer timeout" or "ACK error", then find the log related according to the i2c address, if find the following abnormal log, it means I2C doesn't work well
- Transfer timeout
  - NG log:

```
[24289:AFthread]i2c i2c-2: addr:0x72,transfer timeout
[24289:AFthread]i2c i2c-2: timeout:start=0x1,ch_err=0x0
[24289:AFthread]i2c_dump_info +++++++++++++++++++
[24289:AFthread]I2C structure:
```

0x72→I2C address = 0x72*2= 0xE4



```
#define AF_DRVNAME "LC898229AF_DRV"
#define AF_I2C_SLAVE_ADDR 0xE4
```

  - Debug method

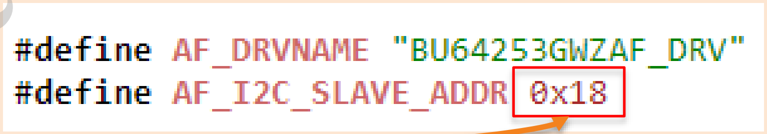
Check if I2C bus number is also used by other devices, and measure the SDA or Sclk to make sure it is OK or abnormal

Some times, Sclk data is always pulled high by other device

Debug Check Point-4
Check if lens pos is set to driver IC

- ACK error
  - NG log

```
SLAVE_ADDR=18,INTR_STAT=3,CONTROL=28,FIFO_STAT=1,DEBUGSTAT=300, tmo=500
[3954:AFthread]i2c i2c-2: i2c_gpio_dump_info +++++++++++++++++++
[3954:AFthread]i2c i2c-2: I2C gpio structure:
[3954:AFthread]i2c i2c-2: addr:0xc,ACK error
[3954:AFthread]i2c i2c-2: trans done with error
```



```
#define AF_DRVNAME "BU64253GWZAF_DRV"
#define AF_I2C_SLAVE_ADDR 0x18
```

0xC ➔ I2C address = 0xC*2 =0x18

  - Debug method

1. Check if the I2C address is matched the driver IC. If not , configure it referring to step3-1-1

   for example： check if 0x18 is equal to the AF_I2C_SLAVE_ADDR in driver IC driver,
2. Check if the I2C bus number is configured as HW connecting. If not, configure it referring to step2
3. Measure if the power of driver IC is effective. If not , configure it referring to step4
4. Check if the driver I2C has any other enable pin
5. Ask driver IC vendor for help

Debug Check Point-5
Check lens is moving in a valid range

- Do fullscan at infinity distance and macro distance such as 5m and 10cm

Get the infinity_dac and macro_dac, compare them with the lens pos ,

if the lens pos is between them, please ask driver IC vendor for help

if not , fine tune the AF table in AF parameter file referring to step5-3

Thanks !!!