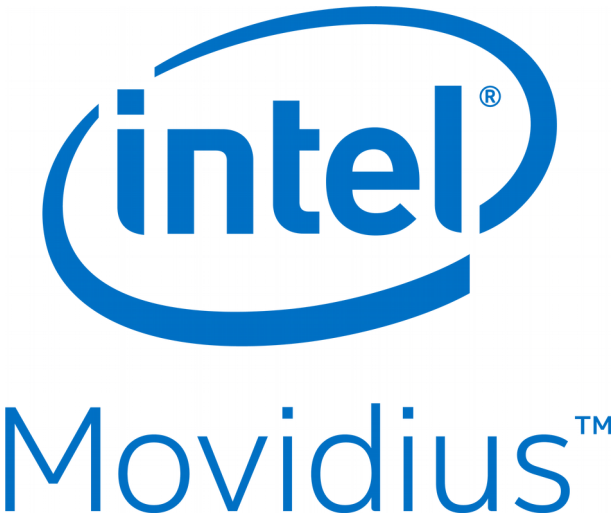


MA2x5x Sensor Integration Guide

v1.0 / August 2018

Copyright and Proprietary Information Notice

Intel® Movidius™
2200 Mission College Blvd
M/S SC11-201
Santa Clara, CA 95054
http://www.movidius.com/

MM Solutions AD.
15 - 17 Tintyava Street
Sofia 1113, Bulgaria
http://www.mm-sol.com/

Table of Contents

1 Introduction

1.1 About this document

The purpose of this document is to give a general view of the Camera SDK component used in Myriad 2 Camera Development Kit (CDK). The Camera SDK is the application interface that is available to the user to add camera specific information.

1.2 Scope

This document describes the steps required to add a new sensor to the CDK.

It describes the code modifications and additions required to integrate a new sensor in the CDK.

1.3 Target Audience

This document is targeted at users of the CDK framework wishing to build camera or image processing solutions running on the Movidius Myriad 2 silicon.

The audience of this document includes CDK system integrators, SW Team Leads and SW developers.

1.4 Related Documents and Resources

Documents relating to the CDK can be obtained from http://www.movidius.org. If you do not have access to the documents below, you can request them. Relevant documents include:

1. CDK Release package (download from Movidius.org)
2. Myriad 2 Platform Datasheet
3. Myriad 2 Instruction Set Reference Manual
4. Camera Interface Specifications, MIPI Alliance, http://www.mipi.org/specifications/camera-interface#CSI2

Ref.	Document	Edition/Date/Link
A1	Myriad 2 Platform Databook, MA2xxx-DS-0.06	0.06/2014-01-23
A2	Myriad 2 Camera System Requirements	v0.03/2013-08-28
R1	Myriad 2 Camera System Architecture Overview	v0.01/2014-01-22
R2	Myriad Streaming Image Processing Pipeline (SIPP) User Guide	v0.06/
R3	Ma2x5x CDK User Guide	Latest version
R4	IPIPE Architecture Specification	Latest version

1.5 Terms and definitions

Term	Document
CDK	Camera Development Kit
3AFW	3A Framework
API	Application Programmer Interface
CD	Camera Driver
DTP	Dynamic Tuning Parameters (DTP) Server Component
GUZZI	Camera control application running on Myriad 2 LeonOS processor
HIF	Host Interface
NVM	Non-volatile Memory

2 Camera SDK Component Description

2.1 Component Role

The Camera SDK is a CDK module component that includes all low level sensor HW controls, all HW configuration and DTP data related to the sensor.

The Camera SDK also includes other hardware modules associated with the sensor such as lens, flash, torch and memories.

In addition, sensor integration is localized to a single module so no other part of the software is affected by adding sensors. The benefits of this architecture (compared with Common Universal Sensor Driver based on SMIA specification) are many, but the main one is that when a new sensor is integrated, it will not break the functionality of other sensors and it will also not break the functionality of common part of the sensor driver.

The only parts that higher level software (e.g. algorithms) are required to do is to load the proper tuning parameters tables for the new sensor. The generation of the tuning tables will be covered under the topic of DTP Tuning Profiles.

2.2 Environment Diagram

As shown in Figure 1, the basic SDK components include: Drivers, DTP data and Platform.

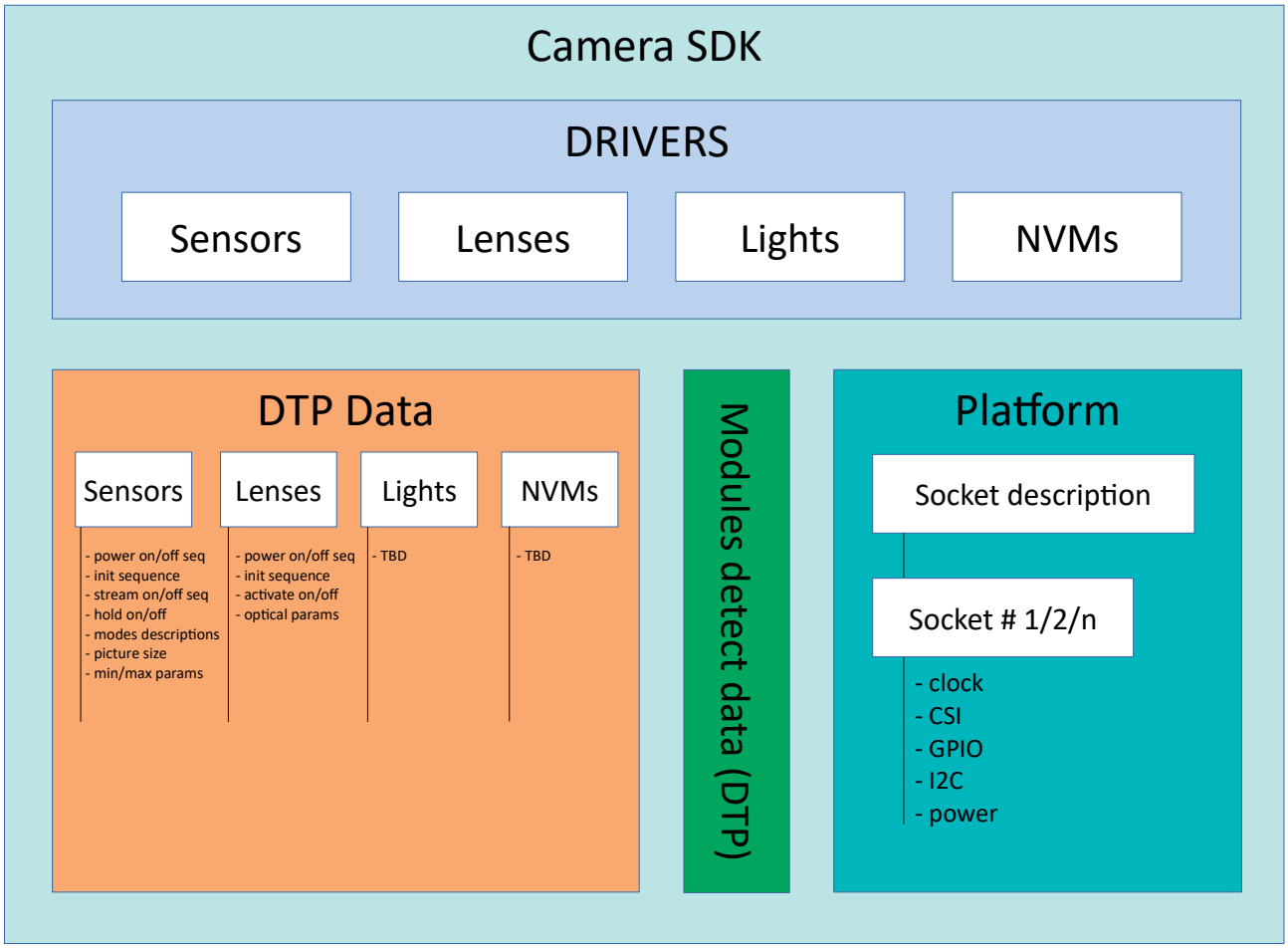


Figure 1: SDK Environment

3 Component Design

The role of the Camera SDK is to collect all source code related to the camera module in a single component. This includes code relating to the camera socket, the platform external drivers and APIs.

The purpose of the camera sub module includes:

- Provide low level communication.
- Execute time critical actions e.g. power on/off, start/stop, suspend.
- Allow reconfiguration the sensor parameters e.g. the lens driver position for sensor exposure gain.

The module contains only low level control functions; no high level algorithms are performed in this module.

Every sub-module exports a standard handle and the handle has a specific set of functions defined e.g. for the sensor driver this is a power on/off, set mode, change exposure gain. And for the Lens driver this is power on/off the lens and a move the lens.

In many instances, the sensor module is also associated with lens and flash module and those are accounted for but the Camera SDK module to form a complete solution.

4 New Sensor Integration

Sensor integration is a process to adding new camera module to the CDK platform.

The integration process is dependent on the following items:

- System hardware design: relevant I2C, GPIO and clocks.
- GPIO pin-muxing.
- Low-level software driver components for GPIO, I2C and Clock.

Because of the SW and HW specifics of the sensors, every integration goes through several steps.

- Acquiring the needed HW and datasheets.
- Acquiring the low level SW drivers on which the integration will be done.
- Integration of sensor/lens drivers in the SW.
- Booting up and debug.

There are different approaches to the sequence of this steps. If the HW is available, low level driver integration and bring-up can start first, then, the SW integration begins. The integration process can start from the SW steps and later add the HW specific steps. The second approach is preferred when some hardware or drivers are not available (sensor modules, boards, …).

Using the “MySensor” sensor as an example the following sections describe the step required to add a new sensor.

4.1 Create sensor DTP ID

DTP Ids file location: camera_sdk/modules/dtp_ids.h

Camera ID

```
44      DTP_CAM_ID_OV13860,        // 5
45      DTP_CAM_ID_MySensor,       // 6
46  } dtp_cam_module_id_t;
```

Sensor ID

```
56      DTP_SEN_DUAL_OV13860,      // 6
57      DTP_SEN_MySensor,          // 7
58  } dtp_sensors_id_t;
```

NOTE: The ID for “DTP_CAM_ID_MySensor” (id is 6 in the example) should match the sensor ID of the same sensor in the database.bin generated by the IQ Tuning Tool. In other case the IQ tuning will not apply.

The database.bin file location is:

~/mdk/projects/Ipipe2/components/lib/dtp_db/database.bin

4.2 Adding sensor “MySensor” in sensor detect list

File location: camera_sdk/modules/camera_module_dtp/cameras/dtp_cm_db.h

This file contains the sensor detect script sequence:

```
117 #define W BE16
118 static const hat_cm_socket_command_entry_t MySensor_cm_det_seq[] = {
119     CMD_ENTRY_DUMMY(),
120     SENS_INTENT_ACTION(SEN_RSRC_POWER_1, HAT_HW_RES_ON),
121     SENS_INTENT_ACTION(SEN_RSRC_CLOCK, HAT_HW_RES_ON),
122     SENS_DELAY(1000),
123     SENS_INTENT_ACTION(SEN_RSRC_XSHTDWN, HAT_HW_RES_OFF),
124     SENS_DELAY(1000),
125     SENS_INTENT_ACTION(SEN_RSRC_XSHTDWN, HAT_HW_RES_ON),
126     SENS_DELAY(1000),
127     SENS_INTENT_CMD_CCI_I2C_RD_CHECK(0x0200 , W(0x1234)),
128     SENS_INTENT_ACTION(SEN_RSRC_POWER_1, HAT_HW_RES_OFF),
129     SENS_INTENT_ACTION(SEN_RSRC_XSHTDWN, HAT_HW_RES_OFF),
130     SENS_INTENT_ACTION(SEN_RSRC_CLOCK, HAT_HW_RES_OFF),
131     CMD_ENTRY_END(),
132 };
133 #undef W
```

The settings used in the macro:

```
SENS_INTENT_CMD_CCI_I2C_RD_CHECK(0x200, W(0x1234))
```

are the register ID address and expected value. These are sensor specific and should be provided in the sensor datasheet.

In the above example these values are:

- **0x0200** - Sensor ID register address
- **0x1234** - is expected value

4.3 Add sensor sensor detect script in detected sensors list:

```
453         },
454     }, // MySensor camera module
455     { // MySensor camera module
456         .cm_dtp_id      = DTP_CAM_ID_MySensor,
457         .cm_name        = "MySensor camera module 0x20",
458         .sen_dtp_id     = DTP_SEN_MySensor,
459         .lens_dtp_id    = DTP_LENS_DUMMY,
460         .lights_dtp_id  = DTP_LIGHTS_DUMMY,
461         .nvm_dtp_id     = DTP_NVM_DUMMY,
462         .socket_mask    = 1 << PH_SOCKET_2,
463         .detect_seq = { // hat_cm_socket_command_entry_t
464             ARR_SIZE(ov9714_cm_det_seq),
465             ov9714_cm_det_seq,
466         },
467         .cm_features = {
468             .cp = {
469                 .sensor = {
470                     .address = 0x20>>1,
471                     .register_size = 2,
472                     .slave_addr_bit_size = 7,
473                 },
474                 .lens = {
475                     .address = 0,
476                     .register_size = 2,
477                     .slave_addr_bit_size = 7,
478                 },
479                 .light = {
480                     .address = 0,
481                     .register_size = 2,
482                     .slave_addr_bit_size = 7,
483                 },
484                 .nvm = {
485                     .address = 0,
486                     .register_size = 2,
487                     .slave_addr_bit_size = 7,
488                 },
489             },
490         },
491     }, // MySensor camera module
```

4.4 Add socket mask per sensor

This mask can assign sensor to specific socket. For all other sockets, this sensor can not be detected.

```
455     { // MySensor camera module
456         .cm_dtp_id      = DTP_CAM_ID_MySensor,
457         .cm_name        = "MySensor camera module 0x20",
458         .sen_dtp_id     = DTP_SEN_MySensor,
459         .lens_dtp_id    = DTP_LENS_DUMMY,
460         .lights_dtp_id  = DTP_LIGHTS_DUMMY,
461         .nvm_dtp_id     = DTP_NVM_DUMMY,
462         .socket_mask    = 1 << PH_SOCKET_2,
463         .detect_seq = { // hat_cm_socket_command_entry_t
```

To enable sensor for all sockets, the value of this mask should be 0.

```
462         .socket_mask    = 0,
463         .detect_seq = { // hat_cm_socket_command_entry_t
```

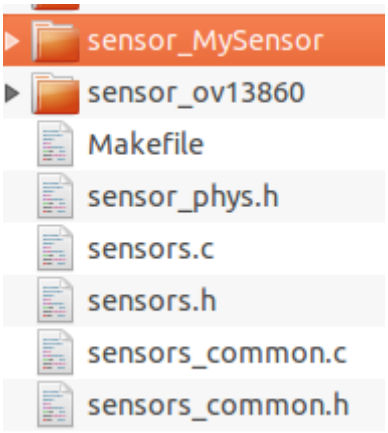
Example for sensor enabling on sockets #2 and #3:

```
462         .socket_mask    = 1 << PH_SOCKET_2 | 1 << PH_SOCKET_3,
```

4.5 Add sensor “MySensor” low level driver

Create a folder for new sensor low level drivers:

File location: camera_sdk/modules/hal/modules/hal_camera_module/sensors/



You can copy one of the existing sensors to use as a template.

The key data is the MySensoor_sens_obj, which contains function pointers to sensor specific functions or to default functions provided by the framework.

```
const hat_cm_ph_sen_obj_t mySensor_sens_obj = {
    .sen_drv_id          = DTP_SEN_MySensor,
    .power_off           = sensors_common_power_off,
```

```
    .power_on            = sensors_common_power_on,
    .init                = sensors_common_init,
    .stream_off          = sensors_common_stream_off,
    .stream_on           = sensors_common_stream_on,
    .set_grped_prm_hold =sensors_common_set_grped_prm_hold,
    .set_mode            = sensors_common_set_mode,
    .set_lpfr            = sensors_common_set_lpfr,

    .set_exp             = mySensor_set_exp,
    .set_gain            = mySensor_set_gain,
    .get_temperature     = mySensor_get_temp,
};
```

In most cases, the "common" or default functions, `sensors_common_XXX`, are sufficient. For gain and exposure, the integrator may wish to write custom functions.

4.6 Include sensor "MySensor" in sensor list and build

Add sensor to supported sensor list:

File location: camera_sdk/modules/hal/modules/hal_camera_module/sensors/sensor.c

```
41 #include "sensor_MySensor/inc/MySensor.h"
42
43 #include <utils/mms_debug.h>
44
45 mmsdbg_declare_variable(hal_cm_driver);
46 #define MMSDEBUGLEVEL mmsdbg_use_variable(hal_cm_driver)
47
48 #ifndef ARR_SIZE
49 #define ARR_SIZE(a) (sizeof(a)/sizeof(a[0]))
50 #endif
51
52 const hat_cm_ph_sen_handle_t sensor_handle_list[] = {
53     (hat_cm_ph_sen_handle_t)&sen_dummy_sens_obj,     // 0
54     (hat_cm_ph_sen_handle_t)&ar1330_sens_obj,        // 1
55     (hat_cm_ph_sen_handle_t)&imx214_sens_obj,        // 2
56     (hat_cm_ph_sen_handle_t)&imx208_sens_obj,        // 3
57     (hat_cm_ph_sen_handle_t)&ov13860_sens_obj,       // 4
58     (hat_cm_ph_sen_handle_t)&MySensor_sens_obj,      // 5
59 };
```

Include the sensor in the build:

File location: camera_sdk/modules/hal/modules/hal_camera_module/sensors/Makefile

```
 9 src += sensor_MySensor/src/MySensor.c
10 src += sensors_common.c
```

4.7 Add sensor DTP data

File location: camera_sdk/modules/camera_module_dtp/sensors/dtp_sensor_MySensor_db/

This folder contains all sensor specific data such as sensor modes, sensor sequences, min/max parameters, array size etc.

```
modules/camera_module_dtp/sensors/Makefile
modules/camera_module_dtp/sensors/dtp_sensor_MySensor_db/dtp_sensor_MySensor.c
modules/camera_module_dtp/sensors/dtp_sensor_MySensor_db/dtp_sensor_MySensor.h
modules/camera_module_dtp/sensors/dtp_sensor_MySensor_db/dtp_sensor_MySensor_db.h
modules/camera_module_dtp/sensors/dtp_sensor_MySensor_db/dtp_sensor_MySensor_mode_640x480_100fps_db.h
```

These are the files used to program the sensor specific registers e.g for different modes. Each mode or operation is programmed via a table of I2C write commands:

```
SENS_INTENT_CMD_CCI_I2C_WR(0x0103, 0x01)
…
```

The specific command sequence is sensor specific and is provided by the sensor manufacturer.

4.8 Add sensor “MySensor” DTP data to DTP list

Add the sensor to the list of supported sensors in DTP.

File location: camera_sdk/modules/camera_module_dtp/sensors/dtp_sensors_db.c

```
42 #include "dtp_sensor_ov13860_db/dtp_sensor_ov13860.h"
43 #include "dtp_sensor_MySensor_db/dtp_sensor_MySensor.h"
44
45 static const dtp_sensor_db_t dtp_sensors_db[] = {
46     ADD_SEN_DTP(dummy_sensor),
47     ADD_SEN_DTP(dual_imx214),
48     ADD_SEN_DTP(imx208),
49     ADD_SEN_DTP(imx214),
50     ADD_SEN_DTP(ar1330),
51     ADD_SEN_DTP(ov13860),
52     ADD_SEN_DTP(MySensor),
53 };
```

And include the sensor dtp in the build:

File location: camera_sdk/modules/camera_module_dtp/sensors/Makefile

```
 9         dtp_sensor_ov13860_db/dtp_sensor_ov13860.c\
10         dtp_sensor_MySensor_db/dtp_sensor_MySensor.c\
```

4.9 Creating a new platform

The easiest way to create a custom platform (e.g. for a new hardware board) is to copy one of the existing ones and use it as a template.

For example, if we have a new platform, e.g. “my_platform”.

1. Create *camera_sdk/configs/my_platform* from *camera_sdk/configs/rtems_mv182*.
2. In *camera_sdk/configs/my_platform/Makefile.maps* change the package name to **pkg_board_platform := modules/platform/my_platform**.
3. Create *camera_sdk/modules/platform/my_platform* from *camera_sdk/modules/platform/mv182*.
4. In *camera_sdk/modules/platform/my_platform/Makefile* change the name of the platform file to **src/plat_my_platform.c**
5. In all source inside *camera_sdk/modules/platform/my_platform* files change the include paths to contain

   ```
   #include <platform/my_platform/
   ```

   Also change the names of the functions to better reflect which platform they are related to.
6. Create (modify) socket files to describe the new HW for each camera interface (socket):
   - camera_sdk/modules/platform/my_platform/src/plat_socket1.c
   - camera_sdk/modules/platform/my_platform/src/plat_socket2.c
   - camera_sdk/modules/platform/my_platform/src/plat_socket3.c
7. List and describe the HW resources that are used:
   - I2C buses,
   - GPIOs,
   - Mipi,
   - power supplies,
   - clocks
   - etc.

4.10 Rebuild CDK

1. Build camera_SDK libs to pick up the new sensor;

   At the CDK root folder:

   ```
   make configs/rtems_sensor_sdk && make
   ```
2. Rebuild application for Myriad 2

   ```
   cd mdk/mdk/projects/Ipipe/apps/GuzziSpi
      make
   ```

Download the Myriad 2 application binary image as per instructions in the CDK User Guide.