Vision SDK

(v03.xx)

ISS Sensor Framework

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards ought to be provided by the customer so as to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is neither responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

TABLE OF CONTENTS

1 Introduction

Vision Software Development Kit (Vision SDK) is a multi-processor software development package for TI's family of ADAS SoCs. The software framework allows users to create different ADAS application data flows involving video capture, video pre-processing, video analytics algorithms, and video display. The framework has sample ADAS data flows which exercises different CPUs and HW accelerators in the ADAS SoC and shows how to effectively use different sub-systems in the SoC. Frame work is generic enough to plug in application specific algorithms in the system.

ISS usecase on TDA3xx captures the RAW data from the external sensors using ISS capture module, process them using ISP modules and display the processed output using display device. ISS usecase uses sensors with the different type of interface and different features, some sensors supports CSI2 interface whereas some supports parallel interface. Some supports Linear output (single exposure), WDR output (Multi-exposure merged output) where some support line interleaved WDR output (Multi-exposure output).

The sensors used in the ISS usecases are all based on the ISS sensor framework. This framework supports easy way to control and configure the external sensor and also easy way to add a new sensor or to remove existing sensor. Below are main advantages/features of the sensor framework.

1. Scalable
2. Easy to add new sensor or remove existing supported sensor
3. Easy to add new feature of the sensor
4. No sensor specific hard coding
5. No multiple sensor Ids
6. Instead of sensor Id, uses, easy to understand, sensor name

This document explains in detail ISS sensor frame.

2 Design

There are two layers in this framework, core layer and sensor layer. Core layer implements framework API and provides application interface API to configure/control sensor. Sensor layer is just one c file for each sensor, which provides basic information about sensor and also configures the sensor.

This frame work provides two types of SW API interfaces, one is internal interface, which is used by the sensors to register itself to the frame work and to provide based features. The other SW interface is external interface, which is used by the application to control and configure the sensors.

2.1 Internal SW Interface

This interface is used by the individual sensor files. Using this interface, sensors register themselves to sensor frame work and provide sensor specific information.

1. Sensor Interfacing information
2. DCC information
3. The maximum number of channels supported by the sensor
4. AEWB Mode information
5. Few optional callback pointer

2.1.1 Register API

This API is used for registering sensors to the frame work. This API takes sensor information structure as an argument. Sensor information structure explains how sensor is connected to the ISS and provides other information.

```
Int32 IssSensorIf_RegisterSensor(IssSensorIf_Params *pSensorPrms);
```

Structures

Sensor information structure provides information about how sensor is connected to the ISS. This structure is used at the time of registering sensor to the framework.

```
struct IssSensorIf_Params_t {
    char                            name[ISS_SENSORS_MAX_NAME];
    /**< Name of the sensor, using which it is registered to this
framework */

    IssSensor_Info                  info;
    /**< Sensor information, which does not change */

    UInt32                          dccId;
    /**< DCC Id of the sensor,
         typically sensor driver provides dcc id, but if sensor driver
         is not opened and it is required to flash dcc profile in qspi,
         this id will be used.
         Note: it should be same as the id in the driver */

    IssSensorIf_Start               start;
    /**< Function to configure and start the sensor,
         If it is null, start will return error */
    IssSensorIf_Stop                stop;
    /**< Function to stop streaming in sensor,
         If it is null, start will return error */
    IssSensorIf_GetExpParams        getExpParams;
    /**< Function to get the sensor exposure parameters */
    IssSensorIf_SetAeParams         setAeParams;
    /**< Function to set AE parameters like exposure and analog gain */
    IssSensorIf_GetDccParams        getDccParams;
    /**< Function to get DCC params */
    IssSensor_InitAewbConfig        initAewbConfig;
    /**< Function to initialize AEWB configuration */
    IssSensor_GetIspConfig          getIspConfig;
    /**< Function to get default ISP configuration */
    IssSensor_ReadWriteRegister     readWriteReg;
```

```
    /**< Function to read/write sensor register */
};

typedef struct
{
    UInt32                              width;
    /**< output width of the sensor  */
    UInt32                              height;
    /**< output height of the sensor  */
    System_VideoDataFormat              dataFormat;
    /**< dataformat of the sensor  */
    System_BitsPerPixel                 bpp;
    /**< Bits per pixel*/

    UInt32                              features;
    /**< Bitwise list of feature supported by the sensor */

    UInt32                              aewbMode;
    /**< AEWB mode */

    UInt32                              ramOffset;
    /**< Offset where DCC profile for this sensor is stored,
         Currently used for specifying QSPI offset */

    UInt32                              maxExp;
    /**< Max Exposure supported for WDR mode, used in #frmInfo */

    IssSensor_LineInterleavedExpFrmInfo lnIntrExpFrmInfo;
    /**< Frame Information for line interleaved output frame */
} IssSensor_Info;
```

2.2 External SW Interface

This interface is used by the application to control and configure the sensor. There are mainly three APIs supported in this external interface

2.2.1 Create API

This API is used for creating a new sensor driver. It uses pointer to the IssSensor_CreateParams as an argument.

```
Ptr IssSensor_Create(IssSensor_CreateParams *pCreatePrms);
```

Create Parameter structure

```
typedef struct
{
    char                            name[ISS_SENSORS_MAX_NAME];
    /**< Name of the sensor */
    UInt32
enableFeatures[ISS_SENSORS_MAX_CHANNEL];
    /**< Bit mask of the features to be enabled in the sensor */
    UInt32                          fps[ISS_SENSORS_MAX_CHANNEL];
    /**< Sensor output fps */
    System_VideoIfWidth
videoIfWidth[ISS_SENSORS_MAX_CHANNEL];
    /**< Video Interface Width for each channel
         used for specifying number of lanes in CSI2 capture */
    UInt32                          numChan;
    /**< Number of channel in which sensor is to be opened */
    UInt32                          i2cInstId;
    /**<  Instance  id  of  the  I2c  on  which  this  sensor  is  to  be
configured */
    UInt8                           i2cAddr[ISS_SENSORS_MAX_CHANNEL];
    /**< I2c address of the each sensor channel */
} IssSensor_CreateParams;
```

Below are important steps that create API performs

1. It searches the registered sensor based on the sensor name given in the create Params.
2. If the sensor is found
    a. It checks for the requested features vs supported features and returns error if a requested feature is not supported
    b. If there is no other error, it returns handle to the sensor.
    c.
    d. It uses the BSP board module to configure board. This includes setting up the pinmux, configuring IO expander to connect sensor to ISS input port and powering up the sensor. Board module also provides sensors i2c information like i2c instance and i2c address of the sensor.
    e. If the board module is not supported by the sensor driver, i2c information of the sensor is provided at the registration time.
    f. It opens BSP sensor driver using the i2c information from the board module or from the registration information.
    g. It gets the features supported by the sensor driver. These features help in configuring and controlling the sensor driver
    h. ISP operation mode in the create parameters is used to configure sensor in the WDR mode. Based on this mode, sensor is either configured in non-wdr, two pass wdr or single pass wdr mode.

i. Once the WDR mode is configured in the sensor, it gets the DCC profile from the sensor driver.

j. It calls the getDefaultConfig callback API to get the default resolution from the sensor. Sensor driver could be supporting multiple resolution, but the usecase runs with the resolution supported in the detDefaultConfig callback.

k. It sets the default resolution in the sensor driver.

2.2.2 Start/Stop API

This APIs are mainly used for configuring and starting/stopping sensor. The core layer internally calls sensor specific start/stop callback and sensor layer configures sensor and then starts/stops the sensor.

This API takes handle to the sensor and channel id of the sensor in case if sensor is opened in multi-channel mode.

```
Int32 IssSensor_Start(Ptr handle, UInt32 chId);
Int32 IssSensor_Stop(Ptr handle, UInt32 chId);
```

2.2.3 Control API

This API is mainly used for sending control command. Below table lists the supported control command.

Idx	Command	Argument	Description
1	Get Default ISP configuration	Pointer to the ISP configuration IssIspConfigurationParameters	This ioctl is used for getting default ISP configuration. Typically, used to get the H3A and GLBCE configuration, the modules for which DCC is not supported. Also used for the sensor for which DCC is not available.
2	Initialize AEWB Create Params	Pointer to AlgorithmLink_IssAewbCreateParams	Initialize AEWB Create parameters AEWB create parameters are sensor specific, this command is used for initializing create args ofr AEWB algorithm.
3	SET AE Params	Pointer to IssSensor_AeParams	When AEWB algorithm is running, it requires to change the AE and analog gain in the sensor runtime. This command is used for setting AE and analog

			gain in the sensor.
4	GET Exposure Params	Pointer to IssSensor_ExposureParams	When WDR is enabled in the sensor, this command is used to get the exposure ratio
5	Read/Write Register	Pointer to IssSensor_ReadWriteReg	Command to read/write sensor register
6	Get DCC Config	Pointer to IssSensor_DccParams	Returns size and pointer to array containing DCC profile

2.2.4 Get Sensor Information

This API provides sensor specific information, it provides sensor resolution, sensor connection information, DCC information etc..

```
Int32 IssSensor_GetSensorInfo (
    char name[],IssSensor_Info *pSensorInfo)
```

Sensor Information Structure

```
typedef struct
{
    UInt32                              width;
    /**< output width of the sensor  */
    UInt32                              height;
    /**< output height of the sensor  */
    System_VideoDataFormat              dataFormat;
    /**< dataformat of the sensor  */
    System_BitsPerPixel                 bpp;
    /**< Bits per pixel*/

    UInt32                              features;
    /**< Bitwise list of feature supported by the sensor */

    UInt32                              aewbMode;
    /**< AEWB mode */

    UInt32                              ramOffset;
    /**< Offset where DCC profile for this sensor is stored,
         Currently used for specifying QSPI offset */

    UInt32                              maxExp;
    /**< Max Exposure supported for WDR mode, used in #frmInfo */
```

```
    IssSensor_LineInterleavedExpFrmInfo lnIntrExpFrmInfo;
    /**< Frame Information for line interleaved output frame */
} IssSensor_Info;
```

3 Steps for adding new sensor

This chapter provided steps for adding new sensor in the ISS sensor framework.

1. Each sensor is implemented in a unique file under `vision_sdk\apps\src\rtos\iss\src\sensor` folder. For the given new sensor, add a new C file.
2. Create a global instance of the structure `IssSensorIf_Params` in this file.
3. Implement the init function in this file. This init function should initialize instances of `IssSensorIf_Params` and registers the new sensor to the sensor frame work using `IssSensorIf_RegisterSensor` API.
4. Implement start and stop callbacks API. These APIs are used for configuring the sensor as well as starting/stopping sensor streaming.
5. Implement `IssGetDefaultISPConfig` API. This api is used to get the default ISP configuration for this sensor. If the sensor does not support DCC, this default configuration will be used for configuring ISP. Also for H3A and GLBCE modules, the configuration comes from this file. DCC profile does not have configuration for H3A and GLBCE modules
6. Implement `IssGetAewbConfig` API. If the usecase uses AEWB algorithm, this API is used for initializing AEWB create parameters. Based on the H3A configuration, it initializes AEWB create parameters, AE dynamic params and AWB calibration data. It also initializes the DCC information in AEWB create parameters.
7. Call the implemented init function from `IssSensor_Init`, available in the file `vision_sdk/apps/src/rtos/iss/src/sensor/iss_sensors.c`
8. Add this new sensor file in `vision_sdk/apps/src/rtos/iss/src/sensor/SRC_FILES.MK` make file so that it gets build along with the other files.

4 Revision History

Version	Date	Revision History
1.0	20th January 2016	Initial Version
2.0	04th July, 2017	Updated as per new sensor framework in Vision SDK 3.0

«««« § »»»»