JPEG decode driver user guide

TRAVEO™ T2G cluster series

About this document

Scope and purpose

This user guide is intended to enable users to integrate the JPEG decode driver for the TRAVEO™ T2G cluster of 32-bit microcontrollers.

This document describes the responsibilities of the integrator in charge of integrating the JPEG decode driver.

Intended audience

This document is intended for anyone using the JPEG decode driver.

This document is intended for users with a basic knowledge of the following:

- Embedded systems
- The C programming language
- The ISO/IEC10918-1 standard
- The target hardware architecture

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus and sub-menus
Italics	Denotes variable(s) and reference(s)
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
>	Indicates that a cascading sub-menu opens when you select a menu item

Abbreviations and definitions

Table 2 Abbreviations

Abbreviation	Definition
API	Application programming interface
HW	Hardware
IRQ	Interrupt request
MCU	Minimum coded unit
MCU	Microcontroller unit
JPEGDEC	MXS40-compliant JPEG decode subsystem
OS	Operating system

About this document

Abbreviation	Definition
SW	Software

Reference documents

This user guide should be read in conjunction with the following documents:

- ISO/IEC10918-1 standard
- TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) (TRAVEO™ TVII-C-2D-6M) (Doc No. 002-25923)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) (TRAVEO™ TVII-C-2D-6M-DDR) (Doc No. 002-32087)

Table of contents

Table of contents

1 Generic information

1.1 Application integration

This section describes the typical workflow required to integrate the JPEG decode driver with the user environment.

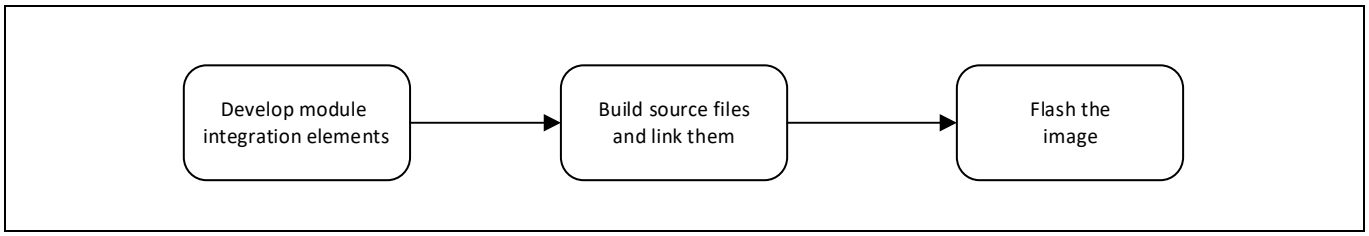

Figure 1 Application integration flow

1.1.1 Package installation and project creation

This section describes how to install the modules and create the projects.

1.1.1.1 Installing JPEG decode driver package

Extract the JPEG decode driver package to a directory, and then do the following:

1. Run the installer(*SW-TVII-JPEG_V[Version number].exe*).
2. The **Setup** dialog box displays. Follow the screen prompts to proceed with the installation process.
3. Enter license key of the installer and click **Next**.

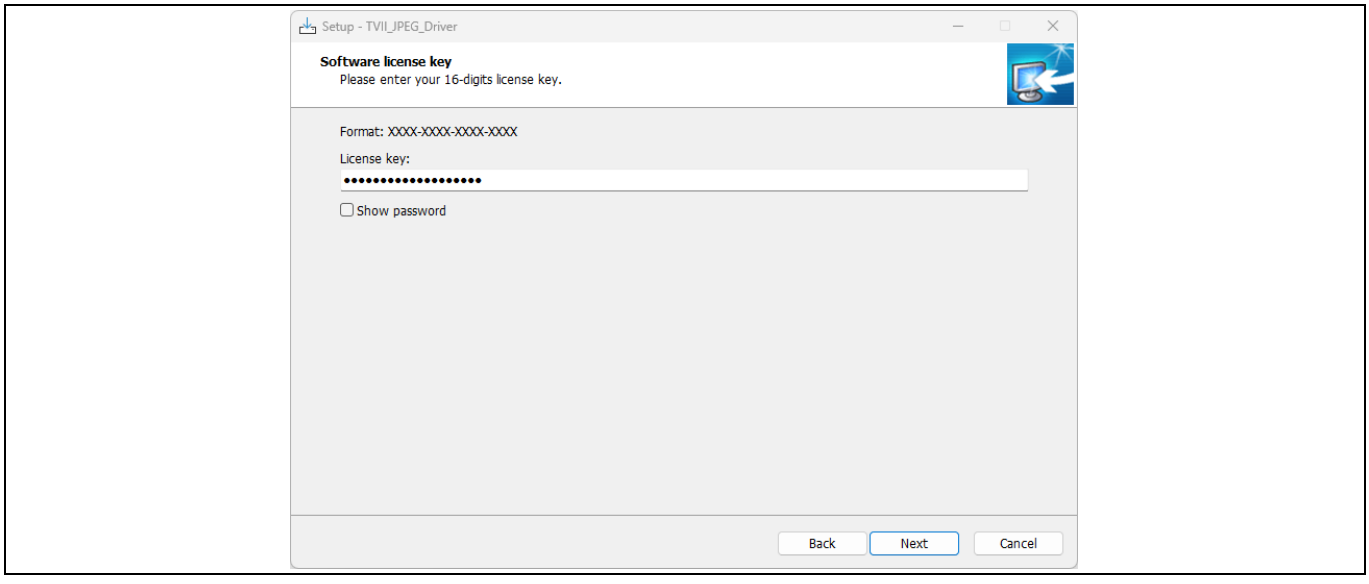

Figure 2 Entering the license key

4. Read the license agreement. If Royalty license, a special EULA will be displayed. Select **I accept the agreement** and click **Next**.

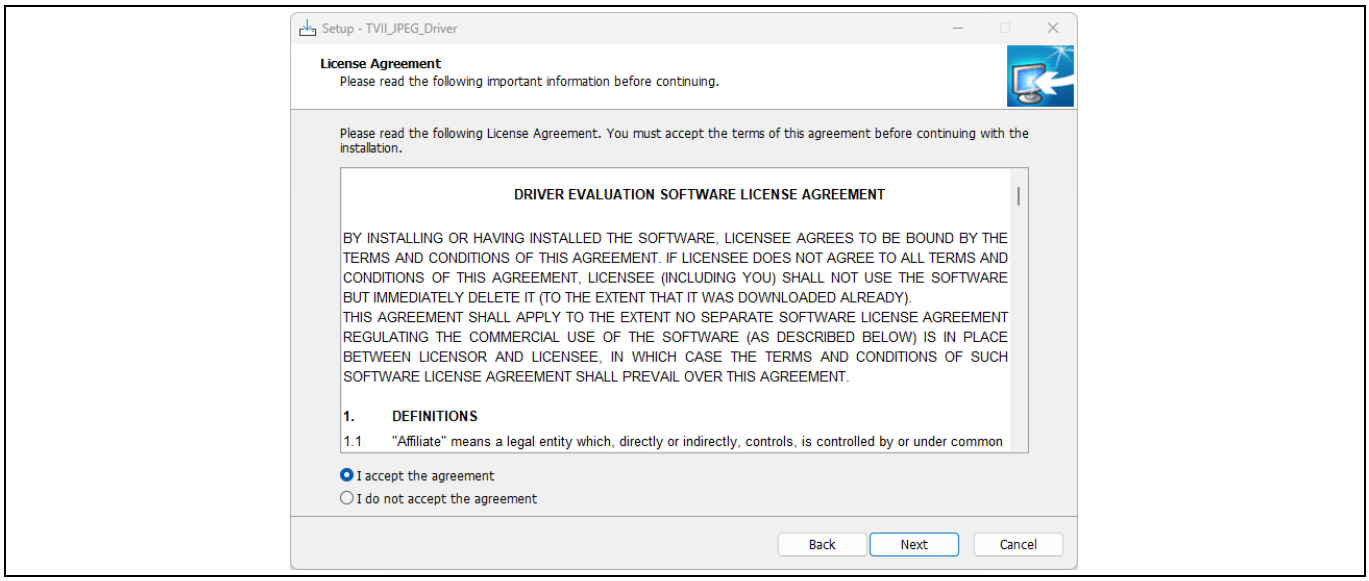

Figure 3 Accepting the license agreement

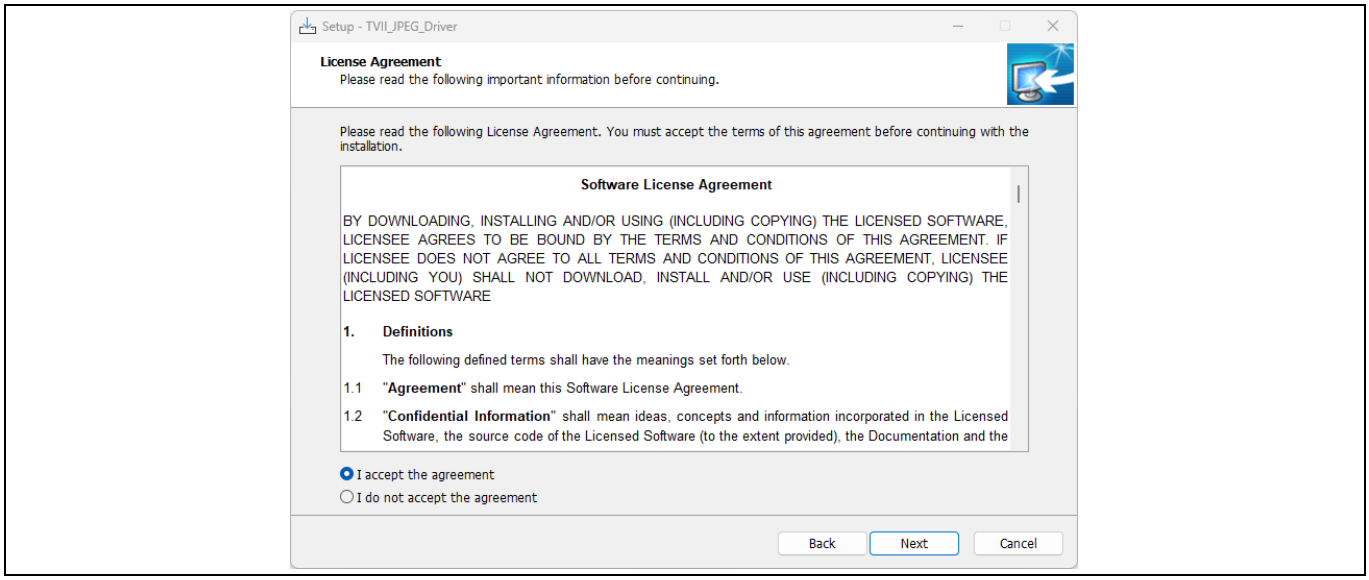

Figure 4 Accepting the license agreement for Royalty license

5. Once installation is complete, click **Finish**.

1.1.2 Building and linking to generate the final application binary

This section describes how to build and link with JPEG decode driver to generate the ELF/HEX files.

1.1.2.1 Compilation folder structure

The following diagram shows the organization of the folders.

```
<Installed folder>
 |
 +---Build
 |    |---GHS
 |    |     +---jpeg_driver
 |    |           \---prj
 |    |                 |  jpeg_driver.gpj
 |    |                 |  libjd.gpj
 |    |                 |
 |    |                 +---config
 |    |                 |     \---production
 |    |                 |          production.opt
 |    |                 \---subprj
 |    |                       intjpg_JpegDrv.ld
 |    |                       source.gpj
 |    \---IAR
 |           \---prj
 |                 jpeg_driver.custom_argvars
 |                 jpeg_driver.ewp
 |                 jpeg_driver.eww
 |
 +---doc
 |        TVII-JPEGDecodeDriver_ReleaseNotes.pdf
 |        TVII-JPEGDecodeDriver_UserGuide.pdf
 |
  \---Source
        \---jpgdec
             +---include
             |     cyjpg.h
             |     cyjpg_basetypes.h
             |     cyjpg_def.h
             |     cyjpg_typedef.h
             |
             +---lib_include
             |     intjpg_ctrl.h
             |     intjpg_def.h
             |     intjpg_typedef.h
             |
             +---lib_src
                   intjpg_ctrl.c
                   intjpg_drvif.c
```

Figure 5 Build folder organization

1.1.2.2 Makefile adaptations

Create the JPEG decode driver library with production configuration by using build environment for GHS or IAR compiler as above. To integrate your application with the JPEG decode driver, add the library files of the JPEG decode driver to the application's Makefile. The integrated application includes the JPEG decode driver.

1.2 Safety objectives

The JPEG decode driver is developed as QM for functional safety.

1.3 Debugging support

The JPEG decode driver does not support debugging.

2 JPEG decode driver

2.1 User information

2.1.1 Description

The JPEG decode driver for TRAVEO™ T2G cluster series is a firmware for TRAVEO™ T2G cluster microcontroller units, which provides an interface to control the JPEG decode function implemented in TRAVEO™ T2G cluster series MCUs. It runs embedded on the target hardware and provides an application interface for the JPEG decode application.

Figure 6 is an example of the detailed system configuration of the JPEG decode driver. The driver provides the interface to control the JPEGDEC HW. The driver enables you to create JPEG decode applications for TRAVEO™ T2G cluster devices that contain a JPEGDEC HW.

The main JPEGDEC HW features depend on the respective TRAVEO™ T2G device. The full featured device includes the following:

- Video performance up to 2880x1080@60Hz for TRAVEO™ TVII-C-2D-6M-DDR
  Video performance up to 1600x800@60Hz for TRAVEO™ TVII-C-2D-6M
- Packed destination buffers for YUV 4:4:4, gray and RGB (8 and 24bpp)
- Semi-planar destination buffers for YUV 4:2:2, 4:1:1 and 4:2:0 (8 and 16 bpp)
- Chroma up-conversion to 4:4:4 for all formats by sample replication
- MMIO registers and interrupt for SW control
- Corrupt data and bus error detection

2.1.2 Hardware-to-software mapping

This section describes the system view of the driver and peripherals administered by it. This document covers the JPEG decode driver, which is shown with red highlights in **Figure 6**, and the interface.

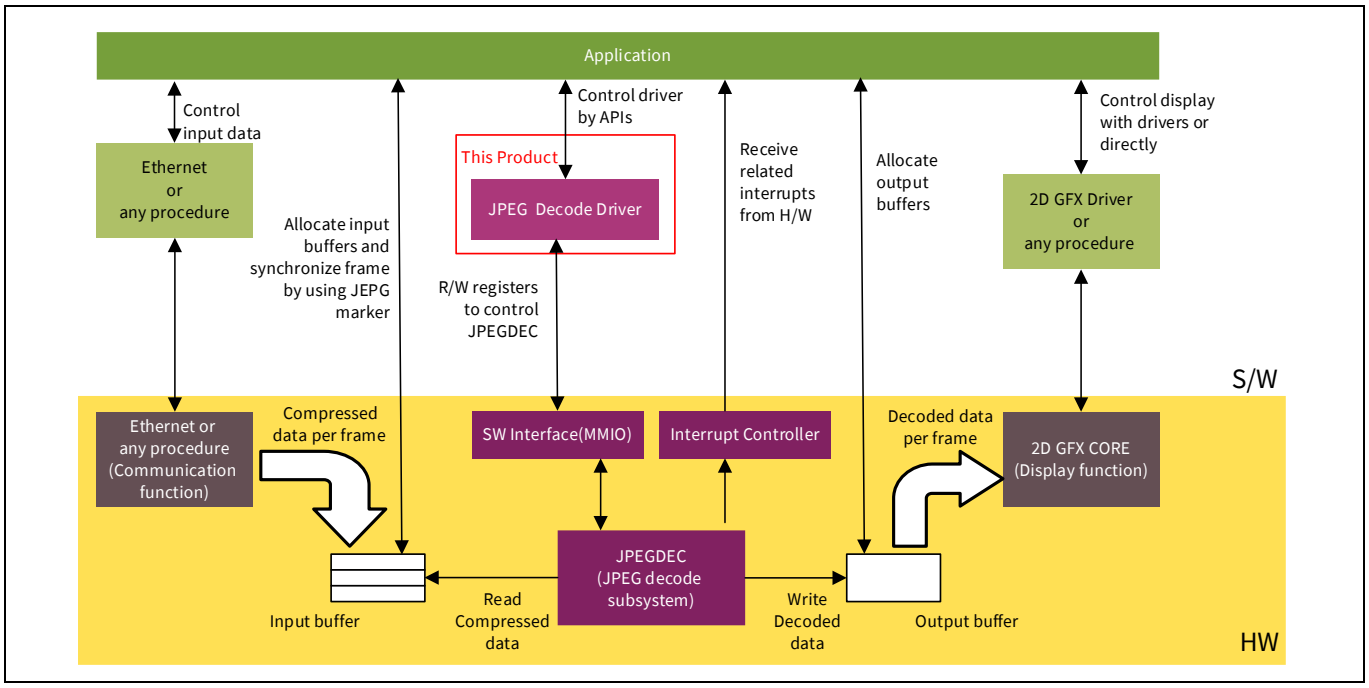

Figure 6 Mapping of hardware-software interfaces

2.1.3 File structure

The following diagram explains the file structure of the JPEG decode driver after the library file created by GHS or IAR build environment.

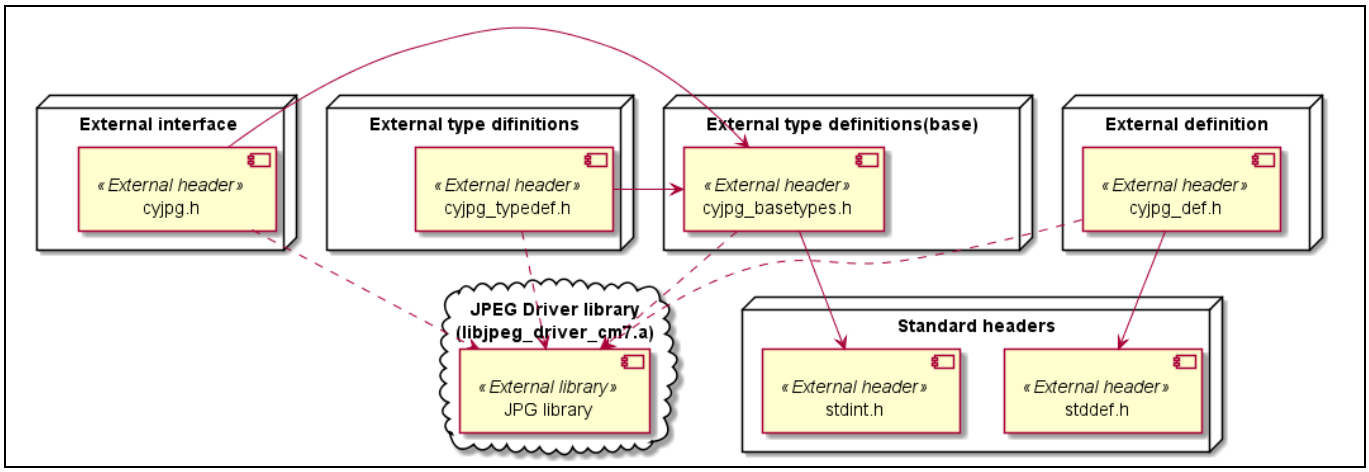

Figure 7 Driver file organization

Table 3 Driver library layout

File name	Description
cyjpg.h	Defines API prototypes
cyjpg_basetypes.h	Defines the base type
cyjpg_def.h	Defines macros for JPEG decode driver
cyjpg_typedef.h	Defines data types and structures
Stddef.h	C standard header file; defines data types
Stdint.h	C standard header file; defines data types
libjpeg_driver_cm7.a	Library file for the JPEG decode driver created by GHS or IAR build environment

2.1.4 System integration

2.1.4.1 Clock and power integration

Not applicable.

2.1.4.2 Input / output integration

Not applicable.

2.1.4.3 Interrupt integration

Not applicable.

2.1.5 Memory mapping

The *cyjpg_def.h* file defines the physical address for JPEG decode HW registers. It is entirely up to the integrator to define a linking scheme to access the JPEG decode HW registers at desired addresses in the processor memory map.

In the *cyjpg_def.h* file, define a physical address for JPEG decode HW as follows:

Code Listing 1 Register address in memory map definition

```
/* Memory map (physical address) */
#define CYJPG_JPEGDEC_BASE          (CYJPG_U32)0x40B10000  /* JPEGDEC Base Address */
#define CYJPG_JPEGDEC_CTL_OFFSET        (CYJPG_U32)0x0000  /* CTL register offset */
#define CYJPG_JPEGDEC_HF_STRUCT_OFFSET  (CYJPG_U32)0x1000  /* HF_STRUCT register offset */
#define CYJPG_JPEGDEC_DEC_STRUCT_OFFSET (CYJPG_U32)0x1800  /* DEC_STRUCT register offset */
```

`CYJPG_JPEGDEC_BASE` is a physical address for JPEG decode HW registers. `CYJPG_JPEGDEC_CTL_OFFSET` is an offset value from `CYJPG_JPEGDEC_BASE` to the CTL register. `CYJPG_JPEGDEC_HF_STRUCT_OFFSET` is an offset value from `CYJPG_JPEGDEC_BASE` to the HF_STRUCT register. `CYJPG_JPEGDEC_DEC_STRUCT_OFFSET` is an offset value from `CYJPG_JPEGDEC_BASE` to the DEC_STRUCT register.

Note: Modify the definitions to fit the memory map if required.

The following sections are defined in the linker directives file for the Green Hills MULTI (GHS) environment.

Code Listing 2 GHS linker directives file for JPEG decode HW

```
SECTIONS {
    .reg_jpeg_ctl           CYJPG_JPEGDEC_BASE    ALIGN(4) :
    .reg_jpeg_hf_struct     (CYJPG_JPEGDEC_BASE + CYJPG_JPEGDEC_HF_STRUCT_OFFSET)
ALIGN(4) :
    .reg_jpeg_dec_struct    (CYJPG_JPEGDEC_BASE + CYJPG_JPEGDEC_DEC_STRUCT_OFFSET)
ALIGN(4) :
}
```

Note: For the IAR environment, no sections are defined in the linker configuration file because they are allocated directly.

2.2 Reference information

2.2.1 Functions – type definitions

This section describes the type definitions used by APIs.

2.2.1.1 CYJPG_ERRIRQ_TYPE_E

Table 4 CYJPG_ERRIRQ_TYPE_E type definition

<Syntax>	`CYJPG_ERRIRQ_TYPE_E`
<Type>	Enumeration
<File>	*cyjpg_typedef.h*
<Range>	`CYJPG_ERROR_NONE`: There is no error interrupt (success). `CYJPG_FETCH_ERROR`: There is an error in the on-going AXI transaction on the fetch function. `CYJPG_STORE_ERROR`: There is an error in the on-going AXI transaction on the store function.
<Description>	This type defines the values for the error interrupt. It describes the type of error interrupt and is used in the `CyJpg_SetErrorInfo` API as an argument.

2.2.1.2 CYJPG_OPERATIONMODE_E

Table 5 CYJPG_OPERATIONMODE_E type definition

<Syntax>	`CYJPG_OPERATIONMODE_E`
<Type>	Enumeration
<File>	*cyjpg_typedef.h*
<Range>	`CYJPG_CONFIG`: The JPEG core is in reset/disabled/configuration mode. `CYJPG_RESERVE`: This value is reserved. `CYJPG_DECODING`: The JPEG core is in decoding mode. `CYJPG_SUSPEND`: The JPEG core is in suspend mode.
<Description>	This type defines the JPEGDEC core status that comes from the OPERATIONMODE field of the JPEGDEC_HF_OPERATINGSTATUS register. It describes the operation mode and can be obtained by calling the `CyJpg_GetHwStatus` API.

2.2.1.3 CYJPG_DECODEINFO_S

Table 6 CYJPG_DECODEINFO_S type definition

<Syntax>	`CYJPG_DECODEINFO_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`subsample`: Frame format. Sub-sampling mode defined in SOF marker segment. 000 YUV 4:4:4 001 YUV 4:2:2 110 YUV 4:1:1 010 YUV 4:2:0 011 Gray Scale 100 CMYK (**Note:** This format is not supported.) 111 Invalid `dri`: The value downloaded from the DRI marker segment, from 0 to 0xFFFF. `sizey`: Y size value downloaded from the SOF marker segment, from 0 to 0xFFFF. `sizex`: X size value downloaded from the SOF marker segment, from 0 to 0xFFFF. `cropstarty`: Y crop start position, from 0 to 0xFFFF. `cropstartx`: X crop start position, from 0 to 0xFFFF. `cropsizey`: Crop size for Y direction, from 0 to 0xFFFF. `cropsizex`: Crop size for X direction, from 0 to 0xFFFF.
<Description>	This type defines a structure for decoding information of the JPEG image. It describes the decoded information and can be obtained by calling the `CyJpg_GetDecInfo` API.

2.2.1.4 CYJPG_JPGINFO_S

Table 7 CYJPG_JPGINFO_S type definition

<Syntax>	`CYJPG_JPGINFO_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`fetchAddress`: Start address of the JPEG image in bytes, from 1 to 0xFFFFFFFF. `fetchLength`: JPEG image length in bytes, from 1 to 0x3FFFFFFF. `storeAddress0`: Start address of the destination buffer 0 in multiples of 64 bytes, from 64 to 0xFFFFFFC0. `storeStride0`: Line stride of the destination buffer 0 in multiples of 64 bytes from 0 to 0xFFFFFFC0. `storeLength0`: Destination buffer 0 length in multiples of 64 bytes from 64 to 0xFFFFFFC0. `storeAddress1`: Start address of the destination buffer 1 in multiples of 64 bytes, from 0 to 0xFFFFFFC0. `storeStride1`: Line stride of the destination buffer 1 in multiples of 64 bytes, from 0 to 0xFFFFFFC0.

	`storeLength1`: Destination buffer 1 length in multiples of 64 bytes, from 0 to 0xFFFFFFC0. `upsamplingEn`: This parameter specifies that decoded pixels are up-sampled to YUV 4:4:4. 0: disable 1: enable `markerskipEn`: This parameter enables the markerskip function. 0: disable 1: enable `cropEn`: This parameter enables the crop function. 0: disable 1: enable `correctionEn`: This bit enables the correction function. 0: disable 1: enable `*decodeinfo_s`: Address pointer for the CYJPG_DECODEINFO_S structure.
<Description>	This type defines a structure for the JPEG information. It describes the JPEG information to decode for calling the following APIs: `CyJpg_Init`, `CyJpg_SetJpegInfo`, `CyJpg_StartDecoding`, `CyJpg_ResumeDecoding`, `CyJpg_ContinueDecoding`

Table 8 Minimum length required for store stride parameters

Color format	`upsamplingEn`	`storeStride0`	`storeStride1`
RGB	-	>= (image width * 3)	0 (not used)
Gray scale	-	>= (image width)	0 (not used)
YUV 4:4:4	-	>= (image width * 3)	0 (not used)
YUV 4:2:2	0	>= (image width)	>= (image width)
	1	>= (image width * 3)	0 (not used)
YUV 4:1:1	0	>= (image width)	>= (image width / 2)
	1	>= (image width * 3)	0 (not used)
YUV 4:2:0	0	>= (image width)	>= (image width)
	1	>= (image width * 3)	0 (not used)

64-byte alignment is also required for `storeStride0` and `storeStride1` values.

2.2.1.5 CYJPG_AXICTL_S

Table 9 CYJPG_AXICTL_S type definition

<Syntax>	`CYJPG_AXICTL_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*

<Range>	`arcache`: ARCACHE value of AXI transactions on the fetch unit, from 0 to 0xF. `awcache`: AWCACHE value of AXI transactions on the store unit, from 0 to 0xF. `fetchburstlength`: Maximum burst length for the fetch unit. 0: Length is 4 1: Length is 8 `storeburstlength`: Maximum burst length for the store unit. 0: Length is 4 1: Length is 8
<Description>	This type defines a structure for the AXI bus control information. It describes the AXI bus information and can be set by calling the `CyJpg_Init` API and `CyJpg_SetAxiInfo` API.

2.2.1.6 CYJPG_IRQCTL_S

Table 10 CYJPG_IRQCTL_S type definition

<Syntax>	`CYJPG_IRQCTL_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`decodeintrmask`: Set the value for DECODEINTRMASK. Setting this parameter to '1' propagates the decoding completion status (INTR_DEC.NORMALEND/INTR_DEC.CORRECTEDEND) to INTR.JPEG_CORE. `frameCompleteClear`: Set the value to clear INTR.FRAME_COMPLETE. `jpegCoreClear`: Set the value to clear INTR.JPEG_CORE. `fetchErrorClear`: Set the value to clear INTR.FETCH_ERROR. `storeErrorClear`: Set the value to clear INTR.STORE_ERROR. `frameCompleteMask`: Set the value for INTR_MASK.FRAME_COMPLETE_MASK. `jpegCoreMask`: Set the value for INTR_MASK.JPEG_CORE_MASK `fetchErrorMask`: Set the value for INTR_MASK.FETCH_ERROR_MASK `storeErrorMask`: Set the value for INTR_MASK.STORE_ERROR_MASK. `appMarkerEn`: Set the value for INTR_DEC_EN.APPMARKER. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `comMarkerEn`: Set the value for INTR_DEC_EN.COMMARKER. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `unknownMarkerEn`: Set the value for INTR_DEC_EN.UNKNOWNMARKER. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `sizeAvailableEn`: Set the value for INTR_DEC_EN.SIZEAVAILABLE. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `errorIntervalEn`: Set the value for INTR_DEC_EN.ERRORINTERVAL. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `errorTotalDataEn`: Set the value for INTR_DEC_EN.ERRORTOTALDATA. Setting this parameter to '1' propagates the corresponding INTR_DEC bit to INTR.JPEG_CORE. `intMaskEn`: This field is a global mask for the propagation of INTR_DEC bits to INTR.JPEG_CORE. This field has following polarity:

	0: enable propagation 1: disable propagation
<Description>	This type defines a structure for interrupt control information. It describes the interrupt control information and can be set by calling the `CyJpg_Init` API and `CyJpg_SetIrqInfo` API.

2.2.1.7 CYJPG_INIT_S

Table 11 CYJPG_INIT_S type definition

<Syntax>	`CYJPG_INIT_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`CYJPG_AXICTL_S *axictl_s`: Address pointer for the AXI bus control structure. `CYJPG_IRQCTL_S *irqctl_s`: Address pointer for the interrupt control structure. `CYJPG_JPGINFO_S *jpginfo_s`: Address pointer for the JPEG information structure.
<Description>	This type defines a structure for initialization. It describes the AXI bus, interrupt information, and JPEG information and can be set by calling the `CyJpg_Init` API.

2.2.1.8 CYJPG_IRQSTATUS_S

Table 12 CYJPG_IRQSTATUS_S type definition

<Syntax>	`CYJPG_IRQSTATUS_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`intrFrameComplete`: Interrupt status of INTR.FRAME_COMPLETE. `intrJpegCore`: Interrupt status of INTR.JPEG_CORE. `intrFetchError`: Interrupt status of INTR.FETCH_ERROR. `intrStoreError`: Interrupt status of INTR.STORE_ERROR. `frameCompleteMasked`: Interrupt status of INTR_MASKED. FRAME_COMPLETE_MASKED. `jpegCoreMasked`: Interrupt status of INTR_MASKED.JPEG_CORE_MASKED. `fetchErrorMasked`: Interrupt status of INTR_MASKED.FETCH_ERROR_MASKED. `storeErrorMasked`: Interrupt status of INTR_MASKED.STORE_ERROR_MASKED. `appMarker`: Interrupt status of INTR_DEC.APPMARKER. `comMarker`: Interrupt status of INTR_DEC.COMMARKER. `unknownMarker`: Interrupt status of INTR_DEC.UNKNOWNMARKER. `sizeAvailable`: Interrupt status of INTR_DEC.SIZEAVAILABLE. `errorInterval`: Interrupt status of INTR_DEC.ERRORINTERVAL. `errorTotalData`: Interrupt status of INTR_DEC.ERRORTOTALDATA. `errorMarker`: Interrupt status of INTR_DEC.ERRORMARKER. `correctedEnd`: Interrupt status of INTR_DEC.CORRECTEDEND. `normalEnd`: Interrupt status of INTR_DEC.NORMALEND. `interval`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.INTERVAL. `totalSize`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.TOTALSIZE. `cropEnd`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.CROPEND.

	`cropSizeError`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.CROPSIZEERROR. `mcuUnitError`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.MCUUNITERROR. `noDri`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.NODRI. `noRst`: Correction mode and crop decoding status of CORRECTIONCROPSTATUS.NORST. `errorCodeVal`: Value of JPEGDEC_DEC_ERRORCODE.
<Description>	This type defines a structure for interrupt status and related status. It describes the interrupt status and related status and can be obtained by calling the `CyJpg_GetJpgIrqCause` API.

2.2.1.9 CYJPG_CORESTATUS_S

Table 13 CYJPG_CORESTATUS_S type definition

<Syntax>	`CYJPG_CORESTATUS_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`operationMode`: Value of the OPERATIONMODE field of the JPEGDEC_HF_OPERATINGSTATUS register. It is defined by the emulation CYJPG_OPERATIONMODE_E. `fetchUnitActive`: Value of the FETCHUNITACTIVE field of the JPEGDEC_HF_OPERATINGSTATUS. `fetchAxiActive`: Value of the FETCHAXIACTIVE field of the JPEGDEC_HF_OPERATINGSTATUS. `storeUnitActive`: Value of the STOREUNITACTIVE field of the JPEGDEC_HF_OPERATINGSTATUS. `storeAxiActive`: Value of the STOREAXIACTIVE field of the JPEGDEC_HF_OPERATINGSTATUS. `fetchstatus`: Value of the JPEGDEC_HF_FETCHSTATUS . `storestatus`: Value of the JPEGDEC_HF_STORESTATUS . `decodingstatus`: Value of the JPEGDEC_DEC_DECODINGSTATUS . `suspend`: Value of the JPEGDEC_DEC_SUSPEND.
<Description>	This type defines a structure for the CORESTATUS information. It describes the JPEG core status information and can be obtained by calling the `CyJpg_GetHwStatus` API.

2.2.1.10 CYJPG_HWERRINFO_S

Table 14 CYJPG_HWERRINFO_S type definition

<Syntax>	`CYJPG_HWERRINFO_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`errorStatusNum`: Index of the latest buffer for HW error status. `errorStatus [10]`: Buffer storing the HW error status. `correctionCropNum`: Index of the latest buffer for correction crop status. `correctionCropStatus [10]`: Buffer storing the correction crop status. `errorIrqNum`: Index of the latest buffer for errorIrq. `errorIrq[2][10]`: Buffer storing the error interrupt information. `errorIrq[0][]` stores the JPEGDEC_DEC_INTR_DEC register value. `errorIrq[1][]` stores the JPEGDEC_DEC_ERRORCODE register value.
<Description>	This type defines a structure for HW error information. It describes the HW error information and related information. It can be obtained by calling the `CyJpg_GetHwErrorInfo` API.

2.2.1.11 CYJPG_SWERRINFO_S

Table 15 CYJPG_SWERRINFO_S type definition

<Syntax>	`CYJPG_SWERRINFO_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`swErrorNum`: Index of the latest buffer for SW error status. `swError [10]`: Buffer storing the SW error status.
<Description>	This type defines a structure for SW error information. It describes the SW error information and can be obtained by calling the `CyJpg_GetSwErrorInfo` API.

2.2.1.12 CYJPG_VERSIONINFO_S

Table 16 CYJPG_VERSIONINFO_S type definition

<Syntax>	`CYJPG_VERSIONINFO_S`
<Type>	Structure
<File>	*cyjpg_typedef.h*
<Range>	`mainVer`: Main version. `minorVer`: Minor version. `patchVer`: Patch version.
<Description>	This type defines a structure for the SW version information. It describes the SW version information and can be obtained by calling the `CyJpg_GetVersionInfo` API.

2.2.2 Functions - APIs

This section lists the APIs provided along with a short description of the functionality.

2.2.2.1 CyJpg_ClearFrameCompIrq

Table 17 CyJpg_ClearFrameCompIrq API specification

<Syntax>	`CYJPG_ERROR CyJpg_ClearFrameCompIrq`
	`(`
	`  void`
	`)`		

<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	-	-	None
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized. `CYJPG_ERR_NO_INTERRUPT`: No interrupt has occurred.
<Description>	This global function indicates to clear cause of the frame complete interrupt.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. A frame complete interrupt has occurred and this function is called from the interrupt handler.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call `CyJpg_Init` API first, and then call this function again. If the return value is `CYJPG_ERR_NO_INTERRUPT`, wait for the frame complete interrupt, and then call this function again.		

2.2.2.2 CyJpg_ClearJpegCoreIrq

Table 18 CyJpg_ClearJpegCoreIrq API specification

<Syntax>	`CYJPG_ERROR CyJpg_ClearJpegCoreIrq ( void )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	-	-	None
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized. `CYJPG_ERR_NO_INTERRUPT`: No interrupt has occurred.
<Description>	This global function indicates to clear cause of the JPEG core interrupt.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. A JPEG core interrupt has occurred and this function is called from the interrupt handler. The `CyJpg_GetJpgIrqCause` API is called.		
<Timing constraints>	None		
<Caveats>	If this API is called before the `CyJpg_GetJpgIrqCause` API, the behavior of this driver is not guaranteed.		
<Error handling>	If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again. If the return value is `CYJPG_ERR_NO_INTERRUPT`, wait for the frame complete interrupt, and then call this function again.		

2.2.2.3 CyJpg_ContinueDecoding

Table 19 CyJpg_ContinueDecoding API specification

<Syntax>	`CYJPG_ERROR CyJpg_ContinueDecoding ( CYJPG_JPGINFO_S const *tmpInfo_s )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	`tmpInfo_s`	`CYJPG_JPGINFO_S const *`	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success

		CYJPG_ERR_NULL: Argument is a NULL pointer. CYJPG_ERR_PARAM: Parameter is invalid. CYJPG_ERR_HW_STATUS: HW status is not expected. CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.
<Description>	This global function indicates to continue decoding with the indicated parameters. This API checks and sets the parameters that are related to fetch/store buffers and up-sampling mode.	
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK. The JPEG core status is in configuration mode.	
<Timing constraints>	None	
<Caveats>	None	
<Error handling>	If the return value is CYJPG_ERR_NULL, correct the pointer for the struct and call this function again. If the return value is CYJPG_ERR_PARAM, correct the parameter and call this function again. If the return value is CYJPG_ERR_HW_STATUS, wait until HW status is as expected., and then call this function again. If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again.	

2.2.2.4 CyJpg_DeInit

Table 20 CyJpg_DeInit API specification

<Syntax>	CYJPG_ERROR CyJpg_DeInit (void)		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	-	-	None
<Parameters (out)>	-	-	None
<Return value>	CYJPG_ERROR		Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NOT_ENABLE: HW is not enabled.
<Description>	This global function deactivates HW and SW.		
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is CYJPG_ERR_NOT_ENABLE, call the CyJpg_Init API first, and then call this function again.		

2.2.2.5 CyJpg_GetDecInfo

Table 21 CyJpg_GetDecInfo API specification

<Syntax>	`CYJPG_ERROR CyJpg_GetDecInfo ( CYJPG_DECODEINFO_S *tmpPtr )`		
<Sync/Async>	Sync		
<Re-entrancy>	True		
<Parameters (in)>	-	-	None
<Parameters (out)>	`tmpPtr`	`CYJPG_DECODEINFO_S *`	Pointer to store the information.
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized.
<Description>	This global function indicates how to get the decode information struct.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. The latest decode information is available after a frame complete interrupt or size and sub-sampling available interrupt occurs.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again. If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again.		

2.2.2.6 CyJpg_GetJpgIrqCause

Table 22 CyJpg_GetJpgIrqCause API specification

<Syntax>	`CYJPG_ERROR CyJpg_GetJpgIrqCause ( CYJPG_IRQSTATUS_S *tmpPtr )`		
<Sync/Async>	Sync		
<Re-entrancy>	True		
<Parameters (in)>	-	-	None
<Parameters (out)>	`tmpPtr`	`CYJPG_IRQSTATUS_S *`	Pointer to store the information.
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success

		CYJPG_ERR_NULL: Argument is a NULL pointer CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.
<Description>	This global function indicates to get JPEGDEC core interrupt cause. When JPEG decoder core interrupt occurs, the CyJpg_GetJpgIrqCause API should be called first. Then, the CyJpg_ClearJpegCoreIrq API needs to be called to clear the interrupts. Otherwise, unintentional interrupts might be cleared.	
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK.	
<Timing constraints>	None	
<Caveats>	None	
<Error handling>	If the return value is CYJPG_ERR_NULL, correct the pointer for the struct and call this function again. If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again.	

2.2.2.7 CyJpg_GetHwErrorInfo

Table 23 CyJpg_GetHwErrorInfo API specification

<Syntax>	CYJPG_ERROR CyJpg_GetHwErrorInfo (CYJPG_HWERRINFO_S *tmpPtr)		
<Sync/Async>	Sync		
<Re-entrancy>	True		
<Parameters (in)>	-	-	None
<Parameters (out)>	tmpPtr	CYJPG_HWERRINFO_S *	Pointer to store the information.
<Return value>	CYJPG_ERROR		Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NULL: Argument is a NULL pointer
<Description>	This global function indicates how to get the HW error information.		
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is CYJPG_ERR_NULL, correct the pointer for the struct and call this function again.		

2.2.2.8 CyJpg_GetHwStatus

Table 24 CyJpg_GetHwStatus API specification

<Syntax>	`CYJPG_ERROR CyJpg_GetHwStatus ( CYJPG_CORESTATUS_S*tmpPtr )`		
<Sync/Async>	Sync		
<Re-entrancy>	True		
<Parameters (in)>	-	-	None
<Parameters (out)>	`tmpPtr`	`CYJPG_CORESTATUS_S *`	Pointer to store the information.
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized.
<Description>	This global function indicates to get HW status.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again. If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again.		

2.2.2.9 CyJpg_GetSwErrorInfo

Table 25 CyJpg_GetSwErrorInfo API specification

<Syntax>	`CYJPG_ERROR CyJpg_GetSwErrorInfo ( CYJPG_SWERRINFO_S *tmpPtr )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	-	-	None
<Parameters (out)>	`tmpPtr`	`CYJPG_SWERRINFO_S *`	Pointer to store the information.
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer

<Description>	This global function indicates how to get the SW error information. This API initializes the stored SW error information.
<Preconditions>	The `CyJpg_Init` API is already called.
<Timing constraints>	None
<Caveats>	None
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again.

2.2.2.10 CyJpg_GetVersionInfo

Table 26 CyJpg_GetVersionInfo API specification

<Syntax>	`CYJPG_ERROR CyJpg_GetVersionInfo`
	`(`
	`  CYJPG_VERSIONINFO_S *tmpPtr`
	`)`		

<Sync/Async>	Sync		
<Re-entrancy>	True		
<Parameters (in)>	-	-	None
<Parameters (out)>	`tmpPtr`	`CYJPG_VERSIONINFO_S *`	Pointer to store the information.
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer
<Description>	This global function indicates to get version information.		
<Preconditions>	None		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again.		

2.2.2.11 CyJpg_Init

Table 27 CyJpg_Init API specification

<Syntax>	`CYJPG_ERROR CyJpg_Init ( CYJPG_INIT_S const *tmpInit_s )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	`tmpInit_s`	`CYJPG_ INIT _S const *`	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer. `CYJPG_ERR_PARAM`: Parameter is invalid. `CYJPG_ERR_NOT_DISABLE`: HW is not disabled.
<Description>	This global function sets each parameter to HW and SW. If the parameters are set out of range or the following parameter/related register value combination is set, the API returns `CYJPG_ERR_PARAM`. Case 1: More than one of following parameters are set to `CYJPG_ENABLE`: • `tmpInit_s->jpginfo_s->markerskipEn` • `tmpInit_s->jpginfo_s->cropEn` • `tmpInit_s->jpginfo_s->correctionEn` Case 2: More than one of following parameters are set to `CYJPG_ENABLE`: • `tmpInit_s->irqctl_s->errorIntervalEn` • `tmpInit_s->irqctl_s->errorTotalDataEn`		
<Preconditions>	JPEG core is disabled.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, check the correct the pointer for the struct, and then call this function again. If the return value is `CYJPG_ERR_PARAM`, correct the parameter, and then call this function again. If the return value is `CYJPG_ERR_NOT_DISABLE`, call the `CyJpg_DeInit` API first, and then call this function again.		

2.2.2.12 CyJpg_ResumeDecoding

Table 28 CyJpg_ResumeDecoding API specification

<Syntax>	`CYJPG_ERROR CyJpg_ResumeDecoding`
	`(`
	`  CYJPG_JPGINFO_S const *tmpInfo_s`
	`)`		

<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	`tmpInfo_s`	`CYJPG_JPGINFO_S const *`	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer. `CYJPG_ERR_PARAM`: Parameter is invalid. `CYJPG_ERR_HW_STATUS`: HW status is not expected. `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized.
<Description>	This global function indicates how to resume decoding with the indicated parameters. This API checks and sets the parameters that are related to store buffer and up-sampling mode.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. The JPEG core status is in suspended mode.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again. If the return value is `CYJPG_ERR_PARAM`, correct the parameter and call this function again. If the return value is `CYJPG_ERR_HW_STATUS`, wait until the HW status is as expected, and then call this function again. If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again.		

2.2.2.13 CyJpg_SetAxiInfo

Table 29 CyJpg_SetAxiInfo API specification

<Syntax>	`CYJPG_ERROR CyJpg_SetAxiInfo ( CYJPG_AXICTL_S const *tmpAxictl )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	`tmpAxictl`	`CYJPG_AXICTL_S const *`	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	`CYJPG_ERROR`		Return the result as a SW error code. `CYJPG_OK`: Success `CYJPG_ERR_NULL`: Argument is a NULL pointer. `CYJPG_ERR_PARAM`: Parameter is invalid. `CYJPG_ERR_NOT_INITIALIZED`: SW is not initialized.
<Description>	This global function sets AXI bus parameters to HW.		
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. The JPEG core status must be in configuration mode.		
<Timing constraints>	None		
<Caveats>	None		
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again. If the return value is `CYJPG_ERR_PARAM`, correct the parameter and call this function again. If the return value is `CYJPG_ERR_NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again.		

2.2.2.14 CyJpg_SetErrorInfo

Table 30 CyJpg_SetErrorInfo API specification

<Syntax>	`CYJPG_ERROR CyJpg_SetErrorInfo ( CYJPG_ERRIRQ_TYPE_E const tmpVal )`		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	`tmpVal`	`CYJPG_ERRIRQ_TYPE_E const`	Error interrupt type
<Parameters (out)>	-	-	None

<Return value>	CYJPG_ERROR	Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_PARAM: Parameter is invalid. CYJPG_ERR_NOT_INITIALIZED: SW is not initialized. CYJPG_ERR_NO_INTERRUPT: No interrupt has occurred.
<Description>	This global function indicates how to set the error interrupt information. This API clears the register that shows the cause of the error interrupt.	
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK. An error interrupt has occurred and this function is called from the interrupt handler.	
<Timing constraints>	None	
<Caveats>	None	
<Error handling>	If the return value is CYJPG_ERR_PARAM, correct the parameter and call this function again. If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again. If the return value is CYJPG_ERR_NO_INTERRUPT, wait for an error interrupt, and then call this function again.	

2.2.2.15 CyJpg_SetIrqInfo

Table 31 CyJpg_SetIrqInfo API specification

<Syntax>	CYJPG_ERROR CyJpg_SetIrqInfo (CYJPG_IRQCTL_S const *tmpIrqctl)		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	tmpIrqctl	CYJPG_IRQCTL_S const *	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	CYJPG_ERROR		Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NULL: Argument is a NULL pointer. CYJPG_ERR_PARAM: Parameter is invalid. CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.
<Description>	This global function sets interrupt parameters to HW. If parameter(s) are set out of range or the following parameters/related register values combination is set, the API returns CYJPG_ERR_PARAM. Case 1: More than one of following parameters are set to CYJPG_ENABLE: • tmpIrqctl->errorIntervalEn		

	• tmpIrqctl->errorTotalDataEn Case 2: If the register is set and the markerskip function is enabled, more than one of the following parameters are set CYJPG_ENABLE: • tmpIrqctl->appMarkerEn • tmpIrqctl->comMarkerEn • tmpIrqctl->unknownMarkerEn • tmpIrqctl->errorIntervalEn • tmpIrqctl->errorTotalDataEn Case 3: The CROP bit in the JPEGDEC_DEC_DECODINGOPTION register is set to enable, tmpIrqctl->errorIntervalEn or/and tmpIrqctl->errorTotalDataEn is set to enable.
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK. The JPEG core status must be in configuration mode.
<Timing constraints>	None
<Caveats>	None
<Error handling>	If the return value is CYJPG_ERR_NULL, correct the pointer for the struct and call this function again. If the return value is CYJPG_ERR_PARAM, correct the parameter and call this function again. If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again.

2.2.2.16 CyJpg_SetJpegInfo

Table 32 CyJpg_SetJpegInfo API specification

<Syntax>	CYJPG_ERROR CyJpg_SetJpegInfo (CYJPG_JPGINFO_S const *tmpJpginfo)		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	tmpJpginfo	CYJPG_JPGINFO_S const *	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	CYJPG_ERROR		Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NULL: Argument is a NULL pointer. CYJPG_ERR_PARAM: Parameter is invalid. CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.
<Description>	This global function sets JPEG parameters to HW. If parameter(s) are sets out of range or the following parameters/related register values combination is set, the API returns CYJPG_ERR_PARAM. Case 1: More than one of following parameters are set to CYJPG_ENABLE:		

	• tmpJpginfo->markerskipEn • tmpJpginfo->cropEn • tmpJpginfo->correctionEn Case 2: if tmpJpginfo->markerskipEn is set to CYJPG_ENABLE, more than one of following bit are set to enable: • JPEGDEC_DEC_INTR_DEC_EN register APPMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register COMMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register UNKNOWNMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register ERRORINTERVAL bit • JPEGDEC_DEC_INTR_DEC_EN register ERRORTOTALDATA bit Case 3: If tmpJpginfo->cropEn is set to CYJPG_ENABLE, the JPEGDEC_DEC_INTR_DEC_EN register ERRORINTERVAL bit or/and ERRORTOTALDATA bit is set to enable.
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK. The JPEG core status must be in configuration mode.
<Timing constraints>	None
<Caveats>	None
<Error handling>	If the return value is CYJPG_ERR_NULL, correct the pointer for the struct and call this function again. If the return value is CYJPG_ERR_PARAM, correct the parameter and call this function again. If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again.

2.2.2.17 CyJpg_StartDecoding

Table 33 CyJpg_StartDecoding API specification

<Syntax>	CYJPG_ERROR CyJpg_StartDecoding (CYJPG_JPGINFO_S const *tmpInfo_s)		
<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	tmpInfo_s	CYJPG_JPGINFO_S const *	Pointer for parameters to be set up
<Parameters (out)>	-	-	None
<Return value>	CYJPG_ERROR		Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NULL: Argument is NULL pointer. CYJPG_ERR_PARAM: Parameter is invalid. CYJPG_ERR_HW_STATUS: HW status is not expected. CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.

<Description>	This global function indicates how to start decoding with the indicated parameters. This API initializes the stored HW error information. If parameter(s) are set out of range or the following parameters/related register values combination is set, the API returns `CYJPG_ERR_PARAM`. Case 1: More than one of following parameters are set to `CYJPG_ENABLE`: • `tmpInfo_s->markerskipEn` • `tmpInfo_s->cropEn` • `tmpInfo_s->correctionEn` Case 2: If `tmpinfo_s->markerskipEn` is set to `CYJPG_ENABLE`, more than one of following bit are set to enable: • JPEGDEC_DEC_INTR_DEC_EN register APPMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register COMMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register UNKNOWNMARKER bit • JPEGDEC_DEC_INTR_DEC_EN register ERRORINTERVAL bit • JPEGDEC_DEC_INTR_DEC_EN register ERRORTOTALDATA bit Case 3: If `tmpInfo_s->cropEn` is set to `CYJPG_ENABLE`, the JPEGDEC_DEC_INTR_DEC_EN register ERRORINTERVAL bit or/and ERRORTOTALDATA bit is set to enable.
<Preconditions>	The `CyJpg_Init` API is already called and returns `CYJPG_OK`. The JPEG core status is in configuration mode.
<Timing constraints>	None
<Caveats>	None
<Error handling>	If the return value is `CYJPG_ERR_NULL`, correct the pointer for the struct and call this function again. If the return value is `CYJPG_ERR_PARAM`, correct the parameter and call this function again. If the return value is `CYJPG_ERR_ HW_STATUS`, wait until the HW status is as expected. Then, call this function again. If the return value is `CYJPG_ERR_ NOT_INITIALIZED`, call the `CyJpg_Init` API first, and then call this function again.

2.2.2.18 CyJpg_SwReset

Table 34 CyJpg_SwReset API specification

<Syntax>	`CYJPG_ERROR CyJpg_SwReset`
	`(`
	`  void`
	`)`		

<Sync/Async>	Sync		
<Re-entrancy>	False		
<Parameters (in)>	-	-	None
<Parameters (out)>	-	-	None

<Return value>	CYJPG_ERROR	Return the result as a SW error code. CYJPG_OK: Success CYJPG_ERR_NOT_INITIALIZED: SW is not initialized.
<Description>	This global function indicates a SW reset.	
<Preconditions>	The CyJpg_Init API is already called and returns CYJPG_OK.	
<Timing constraints>	None	
<Caveats>	None	
<Error handling>	If the return value is CYJPG_ERR_NOT_INITIALIZED, call the CyJpg_Init API first, and then call this function again.	

2.2.3 Error codes classification

This section explains various error types and their corresponding source APIs. The following errors are reported by the JPEG decode driver.

Table 35 Description of production errors reported

Description	Error code and value	Applicable APIs
Parameter failure (Parameter is NULL pointer)	CYJPG_ERR_NULL(0x21000001)	CyJpg_SetAxiInfo CyJpg_SetIrqInfo CyJpg_SetJpegInfo CyJpg_Init CyJpg_StartDecoding CyJpg_ResumeDecoding CyJpg_ContinueDecoding CyJpg_GetVersionInfo CyJpg_GetSwErrorInfo CyJpg_GetHwStatus CyJpg_GetHwErrorInfo CyJpg_GetJpgIrqCause CyJpg_GetDecInfo
Parameter failure (Parameter includes illegal value.)	CYJPG_ERR_PARAM(0x21000002)	CyJpg_SetAxiInfo CyJpg_SetIrqInfo CyJpg_SetJpegInfo CyJpg_Init CyJpg_StartDecoding CyJpg_ResumeDecoding CyJpg_ContinueDecoding CyJpg_SetErrorInfo
JPEG core status failure (JPEG core is not enabled.)	CYJPG_ERR_NOT_ENABLE (0x21000003)	CyJpg_DeInit
JPEG core status failure (JPEG core is not disabled.)	CYJPG_ERR_NOT_DISABLE(0x21000004)	CyJpg_Init

Description	Error code and value	Applicable APIs
JPEG decoding status failure (JPEG core decoding status is not as expected.)	CYJPG_ERR_HW_STATUS (0x21000005)	CyJpg_StartDecoding CyJpg_ResumeDecoding CyJpg_ContinueDecoding
SW status failure (SW status is not as initialized.)	CYJPG_ERR_NOT_INITIALIZED (0x21000006)	CyJpg_SetAxiInfo CyJpg_SetIrqInfo CyJpg_SetJpegInfo CyJpg_StartDecoding CyJpg_ResumeDecoding CyJpg_ContinueDecoding CyJpg_SwReset CyJpg_SetErrorInfo CyJpg_ClearFrameCompIrq CyJpg_GetHwStatus CyJpg_GetJpgIrqCause CyJpg_GetDecInfo CyJpg_ClearJpegCoreIrq
Interrupt status failure (Interrupt status is not as expected.)	CYJPG_ERR_NO_INTERRUPT (0x21000007)	CyJpg_SetErrorInfo CyJpg_ClearFrameCompIrq CyJpg_ClearJpegCoreIrq
Any another errors	CYJPG_KERR (0x1FFFFFFF): Reserved error code. CYJPG_ERR (0x2FFFFFFF): Reserved error code.	None (Do not return from any APIs)

2.2.4 Sample sequence

The sample sequences in the following sections provide examples on how to implement the JPEG decode driver.

2.2.4.1 Initialization and JPEG decoding

The following figure is an example sequence for using the driver for JPEG decoding.

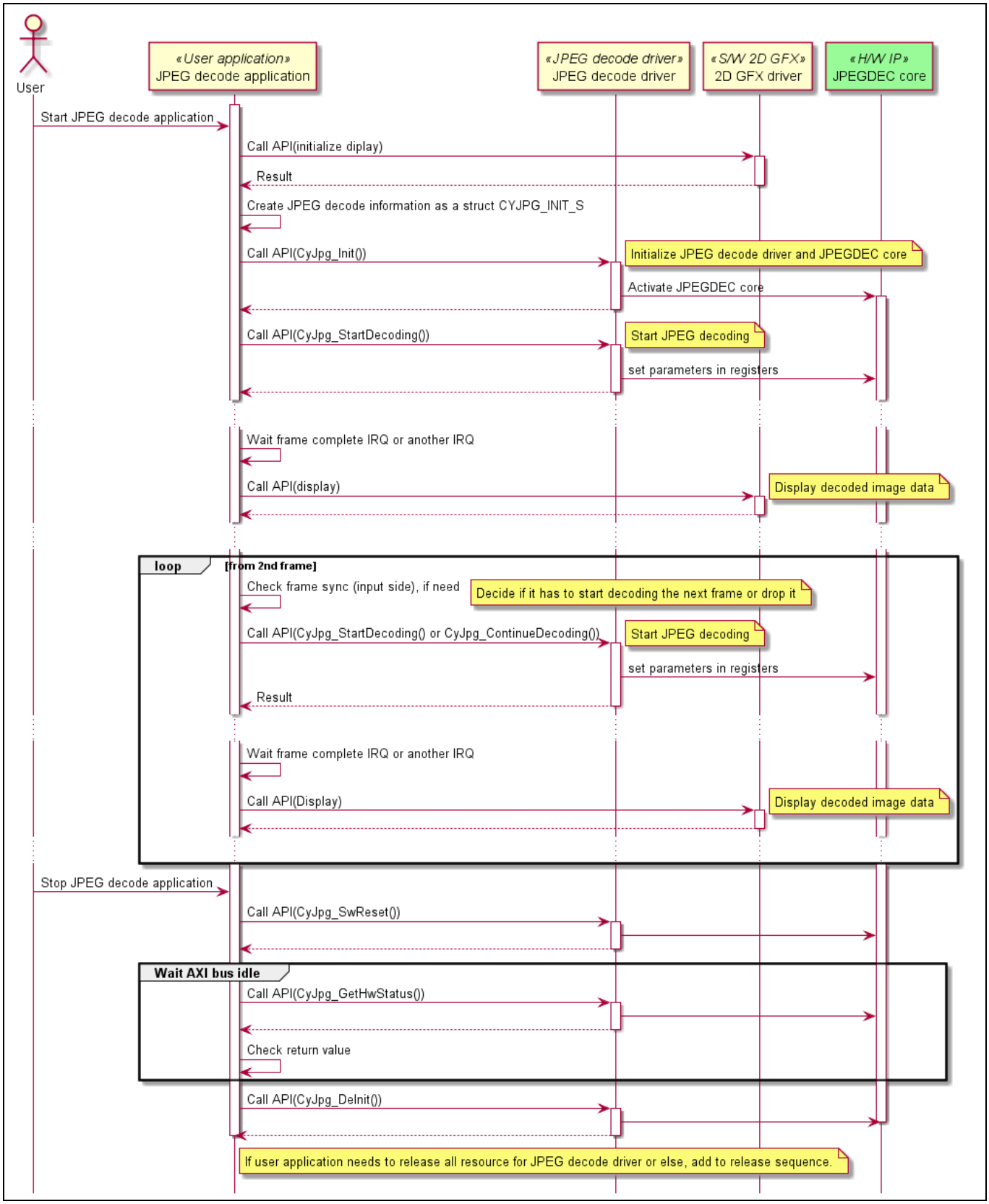

Figure 8 An example of usage for JPEG decode driver

2.2.4.2 JPEG decoding with markerskip function

The following figure is an example sequence of the markerskip function that shows how to use this driver for JPEG decoding.

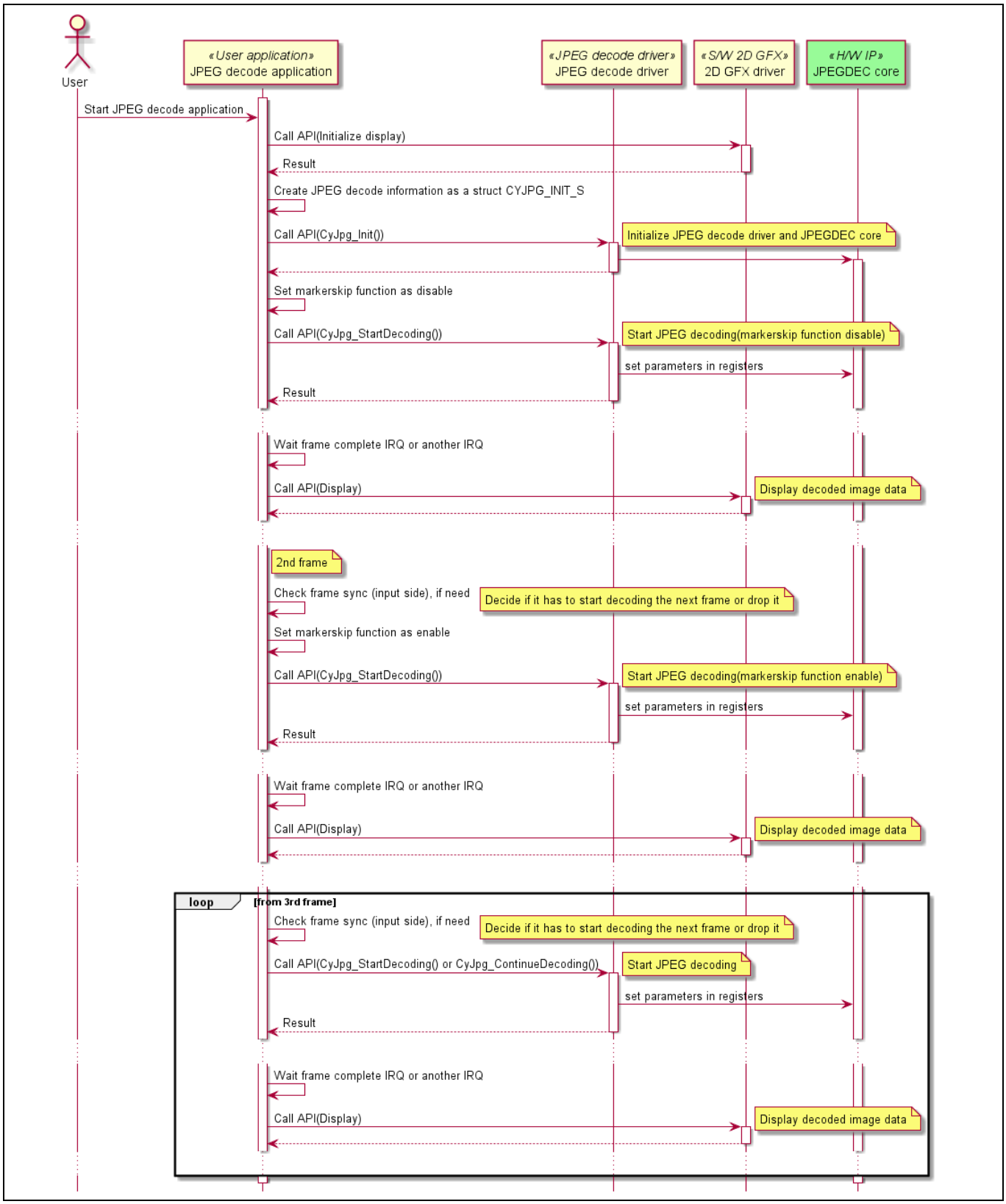

Figure 9 Sample usage with markerskip function

2.2.4.3 Interrupt handling of JPEG decode driver

The following figure is an example sequence of how to handle interrupts for this driver.

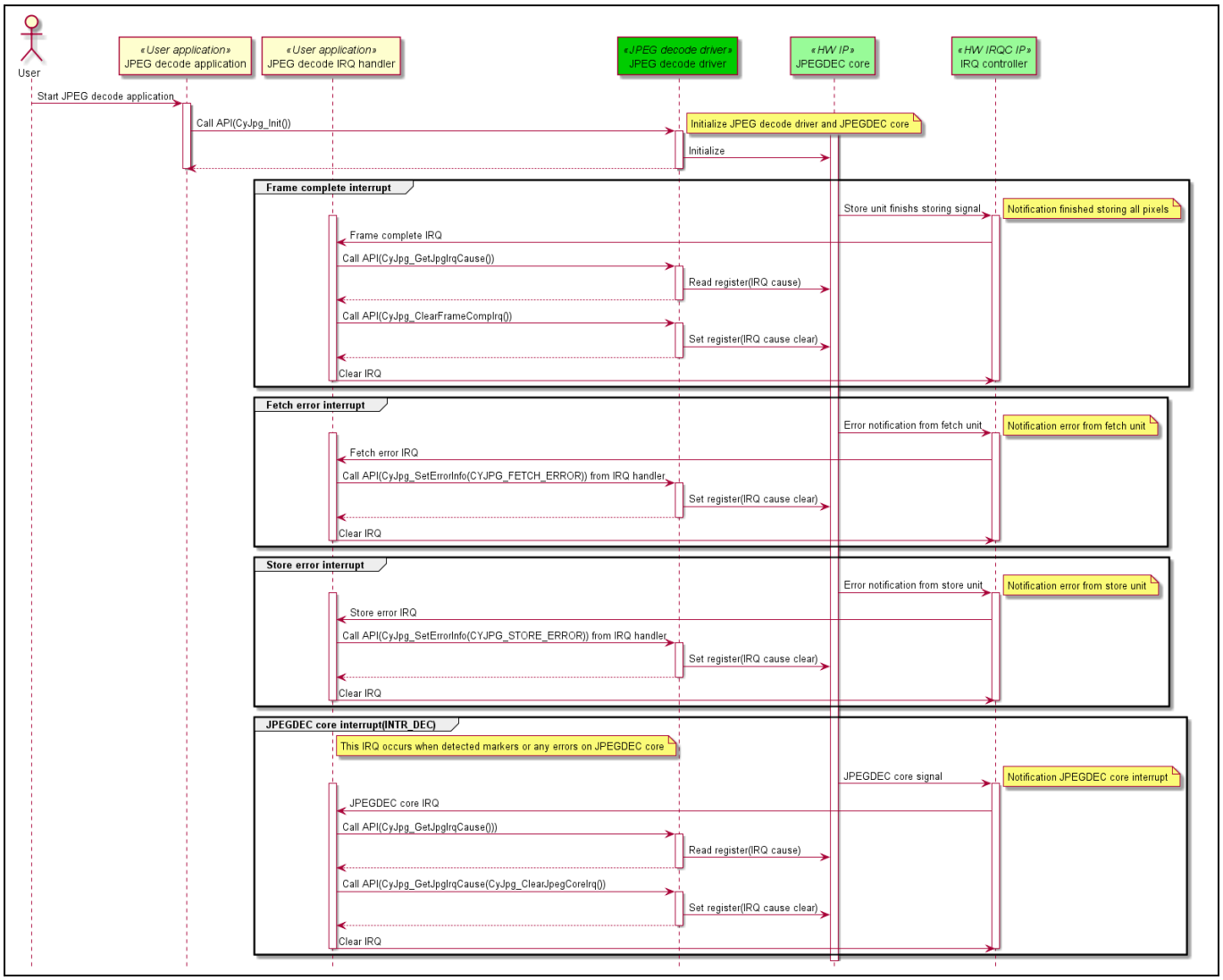

Figure 10 Sample sequence of IRQ handling for JPEG decode driver

2.3 Assumptions, deviations, and limitations

2.3.1 Assumptions

The following table summarizes the assumptions considered for the development of the JPEG decode driver software specification.

Table 36 JPEG decode driver assumptions

Assumption	Description
Native run-time environment ("No OS" and single core)	The driver shall be able to run as part of a native Arm® application with the following characteristics: 1. Allow single-thread / single-process applications 2. No direct dependencies, requirements, or interfaces to an operating system.
Clocks for JPEG decode HW	The clocks for HW are set correctly by the application. If the clock rate requirement is not satisfied, the driver's behavior is not guaranteed.
JPEG stream is as expectedly.	The JPEG stream is stored in the memories as expected, and the application must correctly control these memories. If they are not satisfied, the driver's behavior is not guaranteed.
Interrupt control	Interrupts are controlled correctly by the application. If it is not satisfied, the driver's behavior is not guaranteed.

2.3.2 Deviations

The following table summarizes the deviations from the JPEG decode driver specification.

Table 37 JPEG decode driver deviations

Deviation	Description
Memory control support	The fetch and store memory controls are out of the scope of the JPEG decode driver. The application must control them correctly.
Check absolute address of fetch and store memory in the parameters.	Checking the absolute address of the fetch and store memory locations is out of the scope of the JPEG decode driver. The application must correctly set those addresses to relevant parameters.
HW and SW don't support several functions.	SW does not support the following functions because the HW does not support them. • Progressive and hierarchical coding modes • Sample size 10 bits and 12 bits • Color format conversion or chroma re-sampling • Linear filter • CMYK format
Does not analyze the JPEG stream	The driver does not analyze the JPEG stream. The application should set the JPEG stream information to the driver.

2.3.3 Limitations

The section describes the limitations of the JPEG decode driver specification.

- The driver does not check the upper two bits of the following register. The application must handle these bits.

  JPEGDEC_HF_FETCHBUF1 (FETCHLENGTH): Bits 31:30
- The application should call the `CyJpg_Init` initialization function before any function of the driver.
- The AXI bus setting API functions (`CyJpg_SetAxiInfo` and `CyJpg_Init`) must be called in the AXI bus idle condition.
- The interrupt setting API functions (`CyJpg_SetIrqInfo` and `CyJpg_Init`) must be called when interrupts do not occur.
- The application must set the fetch address and store address to the parameters of the driver as matching the memory map.
- The signal for energy profiling is an output signal from the HW. The JPEG decode driver provides an API to obtain the HW status instead of the signal.
- The crop function has HW limitations. For optimal use, see the HW specification.

2.3.4 Unsupported hardware features

This section lists the hardware features that are not supported by the driver.

There are no unsupported hardware features.

Revision history

Revision	Issue date	Description of change
**	2021-03-02	New Spec
*A	2021-08-12	Updated **Figure 5** in **Compilation folder structure** Added explanation for library creation in **Makefile adaptations** Removed cyjpg_cfg.h file in **File structure**
*B	2021-12-08	Added memory mapping in Memory mapping Updated **Figure 5** in **Compilation folder structure** Updated precondition of CyJpg_GetDecInfo
*C	2022-06-28	Added registers TRM of TRAVEO™ TVII-C-2D-6M-DDR in reference documents
*D	2023-12-08	Web release. No content updates.
*E	2025-04-21	Updated installing instructions in **Installing JPEG decode driver package** Added **Table 8** in **CYJPG_JPGINFO_S**

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-04-21

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-32059 Rev.*E

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.