cadence

Active Safety Features Driver User Guide

Product Version 1.0.0

January 2024

Active Safety Features Driver User Guide

Cadence Design Systems

Hardware Identifier:4841fe58761d944bbc50d000c3b25159

Table of Contents

List of Figures

List of Tables

Chapter 1. Acronyms

API	Application Programming Interface.
ASF	Active Safety Feature.
BIST	Built-In Self Test
Core Driver	Cadence driver component that provides IP programming abstraction.
CPS	Cadence Platform Services. A set of basic platform specific access functions acting as hardware abstraction layer for the Core Driver to operate.
ECC	Error-Correcting Code
PHY	Physical Layer
SECDED	Single Error Correcting and Double Error Detecting
SRAM	Static Random Access Memory - memory that retains its state while powered, used for low-density, low latency memories

Chapter 2. Document Purpose

This document is intended to serve as an overview and a reference for customers looking to use the Cadence ASF Core Driver for the Cadence IP Core Controllers supporting ASF.

Chapter 3. Description

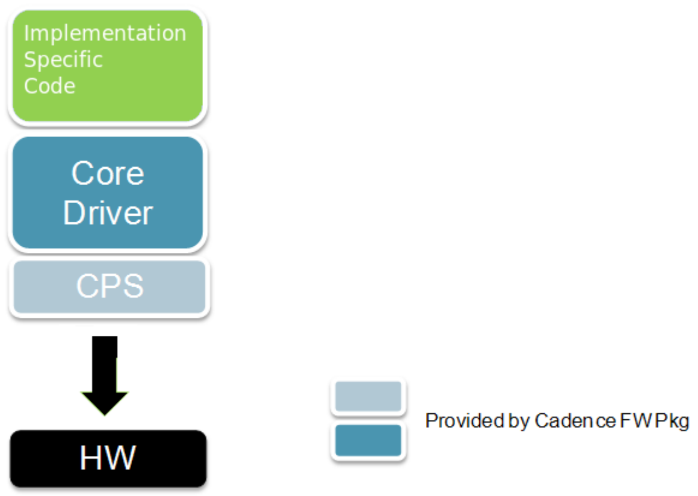
The FW provided by Cadence is :

1. Production Code: Code built to production standards. i.e. it is fully verified with positive and negative testing.
2. Reference Code: This is example code for the platform specific portions of the customers' code base. Reference code is only provided as an example to help the customer with their environment. Ideally code snippets from this will be referred to, and the customer will create their own versions of the reference code that is specific to their environment.

   Reference code is divided into two directories . The first one located in sample_src directory contains example that shows in simple way how ASF Core Driver can be used. The second directory named tests contains source code that can be used for testing purpose. It consist of set of functions that cover all Core Driver API function.

Overview of the Driver Components.

Figure 3.1. Driver Component Diagram



Here, the Core Driver is provided as Production Code. This is meant to be used in the final product, and has been rigorously tested.

The remaining blue box represents the reference code provided to help the customer integrate the ASF extension hardware into his environment. CPS (Cadence Platform Services) respresents a simple hardware abstraction layer allowing access to the hardware.

The portion in green represents the code specific to the implementation which should be written by the customer, and which makes use of the core driver to access the hardware. This can be based on the provided sample code which makes use of the driver to perform typical configuration and operations on the hardware.

3.1. Core Driver Features

Supports all features of the ASF IP Core:

1. IP Fault Logging and Reporting.
2. Transaction Timeout Monitoring.
3. IP Protocol Checking.
4. IP Integrity Checking.
5. SRAM Protection.
6. Data and Address Path Parity Protection
7. Configuration and Status Register Protection.

ASF Core Driver is intended to be reused across the different Cadence drivers. It can be used as internal part of other drivers or as separate driver working in parallel.

Each IP core can supports different ASF features and only subset of them can be supported. ASF driver detects which features are supported by controller associated with it.

It presents a common and consistent way of handling ASF extension. Each IP core supporting ASF must have created separate instance of ASF Driver.

Although ASF IP Core support two separate interrupt line, one for fatal and one for non fatal errors. The ASF Driver assumes that only one interrupt source can be associated with single instance,

Core Driver also implements simple mechanism for gathering statistic about detected faults.

Core Driver driver is compliance with MISRA and HIS.

Core driver is designed to support multiple instance of IP cores.

Core driver is designed to provide APIs allowing access to read/write the ASF core registers

3.2. Reference Code Features

Show how core driver is initialized and API function can be used.

Contain the set of tests that can be used for testing all API function.

3.3. Naming conventions

The ASF Core Driver API will use the prefix ASF_ for public top-level names of functions, constants and data objects definitions.

Chapter 4. Core Driver Usage Model

4.1. Description

ASF Core Driver provide two API interfaces option:

1. Object-like API.

2. Function Prototypes API.

For more details describing difference between API options, please refer to “api_usage_guide.pdf” document.

A public header is provided to define the API and the data structures used. The file is contained in the include directory of the distribution as *asf_if.h and asf_obj_if.h*.

The API requires a private data pointer that provides the Driver with instance information. It is critical that the upper software layer abides by the programming model.

Every API call apart from *probe/ASF_Probe* requires the private data address as the first parameter, to identify the Driver instance and provide access to the internal state of that instance.

There is no resource protection mechanism provided in the ASF Core Driver. It is expected that the higher level software stack will be responsible for protecting calls to the Core Driver where appropriate. In critical areas such as the calling of the *isr/ASF_Isr* function, usage is expected to be single threaded.

If preferred is object-like API then the client code must obtain the object pointer to the core driver. It can be retrieved by issuing *ASF_GetInstance()* function. For more details please refer to api_usage_guide.pdf document.

4.2. Driver Source Structure

A sample makefile is included in the core_driver directory, this may be used with GNU make in order to build the driver on your system. Please see the TODO comments in this file for the necessary modifications.

The Core Driver source is provided in two directories: include/ and src/. The include/ directory contains the public definitions which should be used by higher level software when accessing the driver. Private include files are provided in the src/ directory, and should not be used by higher level software that calls the Core Driver.

However, if the Core Driver is bypassed in order to access hardware registers directly, these include files may be of use since they define structs mapping the hardware registers,

4.3. Probing the Hardware

For object-like API client code must obtain the object pointer to the Core Driver. It will be returned as a result of invoking the *ASF_GetInstance()* function. This object provides access to all of the API functions, but initially only *probe* may be called.

If case of function prototypes API client code has direct access to all API function.

Before initializing the Core Driver, the client must first invoke the *probe/ASF_Probe* function. The configuration information is passed to *probe/ASF_Probe* in an *ASF_Config* struct, although at this stage only those fields which affect the requirements need to be set, as indicated in the struct description. The client may repeat *probe/ASF_Probe* calls to customize configuration depending on available system resources.

```
...
static ASF_Config config = {
  .regBase = (ASF_Regs*)(EXAMPLE_ASF_REGS_BASE),
  .controllerName = {"Example CORE with ASF"},
  .controllerVersion = {"1.0.0"},
  .transactionTimeoutValue = 100
}

/*For function prototypes API*/
uint32_t startExample() {
    ASF_SysReq memReq;
    uint32_t ret = CDN_EOK;
    uint32_t supported =0;
    .......
    memset(&memReq,0,sizeof(memReq));

    if ((ret= ASF_Probe(&config, &memReq)))  {
        return ret;
    }
    printf("Probing of driver successfully completed.\n");
    .....
}

/*For object-like API*/
uint32_t startExample() {
    ASF_SysReq memReq;
    uint32_t ret = CDN_EOK;
    uint32_t supported =0;
    ASF_OBJ * obj = ASF_GetInstance();
    .......
    memset(&memReq,0,sizeof(memReq));

    if ((ret= obj->probe(&config, &memReq)))  {
        return ret;
    }
    printf("Probing of driver successfully completed.\n");
    .....
}
```

4.4. Initialization

After probing stage, the client code must allocate the memory no smaller than what is requested in the return structure provided by the *probe/ASF_Probe* function and use that as the private data for the subsequent initialization and other function calls. If memory allocation is not possible (e.g. if running on a bare-metal platform, statically allocated memory may be used, but the client code should ensure that the resource requirements are met.

The Driver can now be initialized with an *init/ASF_Init* call, which will:

- initialize the Driver instance
- configure the hardware as specified in the *config* parameter fields

e.g. *init/ASF_Init* example, following on from *probe/ASF_Probe* example above:

```
static ASF_PrivateData privateData;
```

```
/*For Function Prototypes API*/
uint32_t startExample() {
    ....
    /*Clear privateData memory*/
    memset(&privateData,0,sizeof(ASF_PrivateData));

    //set all parameters:
    ret = ASF_Init(&privateData, &config, &callback);
    if(ret != CDN_EOK) {
        return ret;
    }
    ....
}

/*For object-like API*/
uint32_t startExample() {
    ....
    /*Clear privateData memory*/
    memset(&privateData,0,sizeof(ASF_PrivateData));

    //set all parameters:
    ret = obj->init(&privateData, &config, &callback);
    if(ret != CDN_EOK) {
        return ret;
    }
    ....
}
```

4.5. Initialization Sequence - selecting ASF features.

ASF driver is responsible for reporting detected error condition to upper layer. For this reason the error conditions, that should be reported have to be selected before starting the driver. Of course client code can change setting after starting but in most case it will be doing before invoking *start/ASF_Start* function.

To activate or deactivate reporting of fault conditions API provides the following functions:

1. enableEvent/ASF_EnableEvent
2. disableEvent/ASF_DisableEvent
3. enableAllEvent/ASF_EnableAllEvents
4. disableAllEvent/ASF_DisableAllEvents
5. enableProtocolEventByMask/ASF_EnableProtocolEventByMask.
6. enableProtocolEventByID/ASF_EnableProtocolEventByID
7. disableProtocolEventByMask/ASF_DisableProtocolEventByMask
8. disableProtocolEventByID/ASF_DisableProtocolEventByID
9. enableTimeoutEventByID/ASF_EnableTimeoutEventByID
10. disableTimeoutEventByMask/ASF_DisableTimeoutEventByMask
11. disableTimeoutEventByID/ASF_DisableTimeoutEventByID

Client code also has the possibility to select which event will be reported as fatal/non fatal interrupt. For this purpose Core Driver API provide two functions:

1. setEventAsNonFatal/ASF_SetEventAsNonFatal

2. setEventAsFatal/ASF_SetEventAsFatal

For more details about these functions please refer to "Driver API" chapter.

By default after initialization all error conditions are disabled, and all are set as fatal.

Hardware controllers don't have to implement all ASF features, therefor API has some additional functions which allow to check supported features:

1. checkIfASFSupported/ASF_CheckIfASFSupported

2. getSupportedASF/ASF_GetSupportedASF

3. getSupportedTimeoutErrors/ASF_GetSupportedTimeoutErrors

4. getSupportedProtocolErrors/ASF_GetSupportedProtocolErrors

Example of usage without probing and initialization section according with object-like API.

```
uint32_t startExample() {
     ....
    /**Set all events as fatal*/
    for(i = 0; i <= ASF_INTEGRITY; i++) {
        ret = ASF_SetEventAsFatal(&privateData, (ASF_EventErrorType)i);
        if(ret != CDN_EOK) {
            return CDN_EINVAL;
        }
    }

   /** Set ASF_SRAM_CORRECTABLE event as non fatal.*/
   ret = ASF_SetEventAsNonFatal(&privateData, ASF_SRAM_CORRECTABLE);
   if(ret != CDN_EOK) {
      return CDN_EINVAL;
   }

   /** Enable handling of all supported events.*/
   if(ASF_EnableAllEvents(&privateData) != CDN_EOK) {
      return CDN_EINVAL;
   }

   /** Get supported protocol errrors mask. */
   if(ret = ASF_GetSupportedProtocolErrors(&privateData, &supported)) {
       return ret;
   }

   /** Enable detection of protocol errors.*/
   if(ASF_EnableProtocolEventByMask(&privateData, supported)) {
        return CDN_EINVAL;
   }

   /** Get supported protocol errrors mask. */
   if(ret = ASF_GetSupportedTimeoutErrors(&privateData, &supported)) {
       return ret;
   }

   /** Enable detection of protocol errors.*/
   if(ASF_EnableTimeoutEventByMask(&privateData, supported)) {
        return CDN_EINVAL;
   }
}
```

4.6. Registering Error Reporting Callback as part of Initialization

Driver provides seven error callback functions. One for each group of error conditions. An Error callbacks may be registered (optional) if the upper layer software needs to be notified when certain error conditions are noticed by the Core Driver. The callback addresses are maintained by the Core Driver once initialized and the Core Driver may invoke the callbacks when it encounters errors as reported by the hardware.

The Callback functions must be declared and defined by the upper layer software and a valid callback function pointers should be passed down to the Core Driver. Callbacks function are passed to Core Driver in ASF_Callbacks object in *init/ASF_Init* function

The code below registers a callbacks as part of the "init" process.

```
static void sramCorrectableEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) {}
static void sramUncorrectableEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }
static void dataAdressParityEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }
static void configStatusRegiseterEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }
static void transactionTimeoutEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }
static void protocolEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }
static void integrityEvent(ASF_PrivateData* privetData, ASF_EventInfo* eventInfo) { }

ASF_Callbacks callback = {
    .sramCorrectableEvent = sramCorrectableEvent,
    .sramUncorrectableEvent = sramUncorrectableEvent,
    .dataAdressParityEvent = dataAdressParityEvent,
    .configStatusRegiseterEvent = configStatusRegiseterEvent,
    .transactionTimeoutEvent = transactionTimeoutEvent,
    .protocolEvent = protocolEvent,
    .integrityEvent = integrityEvent
};

uint32_t startExample() {
    ...
    /** Set all parameters.*/
    ret = ASF_Init(&privateData, &config, &callback);
    if(ret != CDN_EOK) {
        return ret;
    }
 ...
}
```

4.7. Collaboration with main Core Driver

ASF driver is specific kind of Cadence driver because it cannot exists alone and is an extension to other Core Drivers. As a result, client code using IP controller with ASF should initialize base Core Driver and ASF Core Driver. Each driver should created a separate instance of ASF Core Driver. It is responsibility of the upper layer to bind base Core Driver with ASF Core Driver

4.8. Interrupt Handling

The *isr/ASF_Isr* function may be invoked by upper layer software as part of the Interrupt processing being handled by the upper layer software. The invocation of the *isr/ASF_Isr* function of the Core Driver will result in a callback to the function registered as part of the *init/ASF_Init* process IF there is an fault condition detected by ASF controller.

It is the responsibility of the upper layer software to deal with any faults condition that are reported by the Core driver. The *isr/ASF_Isr* routine is expected to be called by the Upper layer software code as part of its interrupt handler.

Although the ASF controller support two separate interrupt line dedicated for handlin fata and non-fatal fault conditions, the Core Driver API support only one *isr/ASF_Isr* function. The kind of detected interrupt will be recognized internally by driver and will be reported to upper layer in ASF_EventInfo object, which in turn is the parameter passed to callback functions. To more information about initializing callback function, please refer to "Registering Error Reporting Callback as part of Initialization" section

Example of using isr/ASF_Isr:

```
/**Function Prototypes API*/
void sampleIsr() {
    ASF_Isr(&privateData);
}

/**Object-like API*/
void sampleIsr() {
    drv->isr(&privateData);
}
```

sampleIsr function should be registered in interrupt handling system which is a platform specific code.

Chapter 5. Configuration and Hardware Operation Information

5.1. Introduction

The following definitions specify the driver operation environment that is defined by hardware configuration or client code.

These defines are located in the header file of the core driver.

Table 5.1. Configuration and Hardware Operation Information

`#define`	value	description
`ASF_CONTROLLER_NAME_LEN`	(100U)	Length of array holding the controller name associated with this ASF instance.
`ASF_CONTROLLER_VERSION_LEN`	(16U)	Length of array holding the version of controller associated with this ASF instance.
`ASF_MAX_NUMBER_EXTENDED_EVENTS`	(32U)	This number is used by statistics module. The available range is 0 to 32. The safest solution is to leave this parameter unchanged. As extended events Core Driver understand the maximum available number of protocol or timeout fault errors. ASF has two IP specific group of faults - protocol errors and transaction timeout errors. They are reported as interrupt by ASF core. Each of them has additional separate register asf_trans_to_fault_mask and asf_protocol_fault_mask and each bit in these registers represent different fault event. So every protocol (ASF_PROTOCOL) and transaction timeout (ASF_TRANSACTION_TIMEOUT) fault event can means 1 of 32 different faults.
`ASF_LAST_SRAM_ERROR_ARRAY_SIZE`	(10U)	Size of array used by statistics module holding last detected SRAM errors.

Chapter 6. Dynamic Data Structures

6.1. Introduction

This section defines the data structures used by the driver to provide hardware information, modification and dynamic operation of the driver.

These data structures are defined in the header file of the core driver and utilized by the API.

6.2. ASF_Config_s

Type: struct

Description

Configuration of device.

Object of this type is used for probe and init functions.

Fields

`regBase`	Base address of device controller registers.
`controllerName`	Name of the controller related to this ASF instance. This field is optional and can be helpful during debugging complex project.
`controllerVersion`	Version of controller related to this ASF instance. This field is optional and can be helpful during debugging complex project.
`transactionTimeoutValue`	Timer value to use for transaction timeout monitor. Driver assumes that this value will be set only once during initialization. This parameter shall be specified in milliseconds.

6.3. ASF_SysReq_s

Type: struct

Description

System requirements returned by probe.

Fields

`privDataSize`	Size of memory required for driver's private data.

6.4. ASF_sramEventInfo_s

Type: struct

Description

Structure holds information describing SRAM fault event.

Fields

`sramInstanceId`	ID of SRAM instance that generated fault Event.
`sramAddress`	Address of SRAM instance that generated fault Event.

6.5. ASF_sramInfo_s

Type: struct

Description

Structure holds information related to all SRAM instances.

Fields

`sramUncorrectableError`	Field used only for ASF_SRAM_UNCORRECTABLE and ASF_SRAM_CORRECTABLE error events. This filed will be set only if ASF was not able to correct detected SRAM fault.
`sramFaultInfo`	Field used only for ASF_SRAM_UNCORRECTABLE and ASF_SRAM_CORRECTABLE error events. Each controller can support many SRAM block. This field indicate for which SRAM instance ID and SRAM address fault occurred.
`uncorrectableErroCounter`	Field contains number of detected uncorrectable errors. It is updated only on ASF_SRAM_UNCORRECTABLE event.
`correctableErrorCounter`	Field contains number of detected correctable errors. This filed is updated only on ASF_SRAM_CORRECTABLE event.

6.6. ASF_EventInfo_s

Type: struct

Description

Instance of this object is passed as parameter to all functions defined in ASF_Callbacks object.

Core Driver used this structure to describe all kind of detected events The supported events are described by ASF_EventErrorType data type. The type of detected and reported event is stored in eventErrorCode field. Information included in this object can be collected by upper layer for diagnostic or statistics purpose. Additionally to facilitate identification of IP controller associated with this event it has name and version fields describing name and version of IP Controller.

Fields

`name`	Name of controller associated with ASF module.
`version`	Version of controller associated with ASF module.
`eventErrorCode`	Interrupt event type detected by ASF.
`fatalEvent`	Indicates whether reported fault is fatal or non-fatal.
`sramInfo`	Field holds all information related to SRAM.
`extendedErrorIdx`	Fault number for protocol and timeout events. ASF specification say that this timeout and protocol fault are IP specific, Each IP controller can contains different numbers of this events. extendedErrorIdx field defines the index of fault event. Basic fault type can be read from eventErrorCode field.

6.7. ASF_StatInfo_s

Type: struct

Description

Structure holds statistic data gathered by ASF Core Driver.

Only enabled ASF features will be gathered. For protection of gathered data Core Driver internally holds ale collected data and sends to upper layer only copy of this data. To retrieved actual ASF_statInfo object getStatistic API function should be called. This statistic data can be used for debug and diagnostic purposes. Core Driver has implemented three API functions that can be used for handling statistic:

Fields

`name`	Name of controller associated with ASF module.
`version`	Version of controller associated with ASF module.
`integrityErrCounter`	Number of detected integrity errors.
`configStatusErrCounter`	Number of detected configuration and status registers errors.
`dataAdressParityErrCounter`	Number of detected data and address paths parity errors.
`sramUncorrectableErrCounter`	Number of detected SRAM uncorrectable errors.
`lastSramCorrectableErr`	Last ASF_LAST_SRAM_ERROR_ARRAY_SIZE detected correctable SRAM error events.
`lastSramUncorrectableErr`	Last ASF_LAST_SRAM_ERROR_ARRAY_SIZE detected uncorrectable SRAM error events.
`sramCorrectableErrCounter`	

	Number of detected SRAM correctable errors.
`protocolErrCounter`	Number of detected protocol errors. Each element of array holds information about other protocol error. Max number of possible protocol errors supported by ASF is 32. The number of supported by controller timeout errors associated with this instance of ASF can be read from protocolErrCount.
`allProtocolErrCounter`	Number of all detected protocol errors.
`timeoutErrCounter`	Number of detected timeout errors. Each element of array holds information about other timeout error. Max number of possible timeout errors supported by ASF is 32. The number of supported by controller timeout errors associated with this instance of ASF can be read from timeoutErrCount,
`allTimeoutErrCounter`	Number of all detected timeout errors.

6.8. ASF_Callbacks_s

Type: struct

Description

struct containing function pointers for communication ASF driver with higher layers.

Each call passes the driver's privateData pointer for instance identification if necessary, and also pass data related to the event.

Fields

`sramCorrectableEvent`

`sramUncorrectableEvent`

`dataAdressParityEvent`

`configStatusRegiseterEvent`

`transactionTimeoutEvent`

`protocolEvent`

`integrityEvent`

6.9. ASF_EventErrorType

Type: enum

Description

Type defines all available events that can be reported by ASF.

Values

`ASF_SRAM_CORRECTABLE`	Value: = 0U
`ASF_SRAM_UNCORRECTABLE`	Value: = 1U
`ASF_DATA_PARITY`	Value: = 2U
`ASF_CONFIGURATION`	Value: = 3U
`ASF_TRANSACTION_TIMEOUT`	Value: = 4U
`ASF_PROTOCOL`	Value: = 5U
`ASF_INTEGRITY`	Value: = 6U

6.10. ASF_Config

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct ASF_Config_s` **`ASF_Config`**

6.11. ASF_SysReq

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct ASF_SysReq_s` **`ASF_SysReq`**

6.12. ASF_sramEventInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

`struct ASF_sramEventInfo_s` **`ASF_sramEventInfo`**

6.13. ASF_sramInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
struct ASF_sramInfo_s ASF_sramInfo
```

6.14. ASF_EventInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
struct ASF_EventInfo_s ASF_EventInfo
```

6.15. ASF_StatInfo

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
struct ASF_StatInfo_s ASF_StatInfo
```

6.16. ASF_Callbacks

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
struct ASF_Callbacks_s ASF_Callbacks
```

6.17. ASF_PrivateData

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
struct ASF_PrivateData_s ASF_PrivateData
```

6.18. ASF_eventErrorDetected

Type: typedef

This is a callback event type. A function with matching prototype needs to be registered with the `callbacks` structure at `init` to enable event signaling.

Function Declaration

```
void(* ASF_eventErrorDetected )(ASF_PrivateData *privetData, ASF_EventInfo
*eventInfo)
```

Description

Reports all errors detected by ASF.

Params: privetData - driver state info specific to this instance. eventInfo - information fully describing detected event.

Chapter 7. Driver Function API

7.1. Introduction

Prototypes for the driver API functions.

The user application can link statically to the necessary API functions and call them directly.

7.2. ASF_Probe

Function Declaration

`uint32_t ASF_Probe(`*`config`*, *`sysReq`*`);`

`const ASF_Config *config`

`ASF_SysReq *sysReq`

Parameter List

`config` [in]

Driver/hardware configuration required.

`sysReq` [out]

Holds information about the size of memory allocations required.

Description

Checks configuration object.

Return Value

CDN_EOK on success (requirements structure filled).

CDN_ENOTSUP if configuration cannot be supported due to driver/hardware constraints.

7.3. ASF_Init

Function Declaration

`uint32_t ASF_Init(`*`privateData`*, *`config`*, *`callbacks`*`);`

`ASF_PrivateData *privateData`

`const ASF_Config *config`

`const ASF_Callbacks *callbacks`

Parameter List

`privateData` [in]

Driver state info specific to this instance.

`config`	[in]
	Specifies driver/hardware configuration.
`callbacks`	[in]
	Client-supplied callback functions.

Description

Initializes the driver instance as specified in the config.

Return Value

CDN_EOK on success

CDN_EINVAL if illegal/inconsistent values in 'config'.

CDN_ENOTSUP if hardware has an inconsistent configuration or doesn't support feature(s) required by 'config' parameters.

7.4. ASF_Destroy

Function Declaration

`void ASF_Destroy(`*`privateData`*`);`

`ASF_PrivateData *privateData`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.

Description

Destroy the driver (automatically performs a stop).

7.5. ASF_Start

Function Declaration

`void ASF_Start(`*`privateData`*`);`

`ASF_PrivateData *privateData`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.

Description

Start the ASF driver.

7.6. ASF_Stop

Function Declaration

`void ASF_Stop(`*`privateData`*`);`

```
ASF_PrivateData *privateData
```

Parameter List

`privateData` [in]

Driver state info specific to this instance.

Description

Stop the driver.

This should disable the hardware, including its interrupt at the source.

7.7. ASF_Isr

Function Declaration

`void ASF_Isr(`*`privateData`*`);`

```
ASF_PrivateData *privateData
```

Parameter List

`privateData` [in]

Driver state info specific to this instance.

Description

Driver ISR.

Platform-specific code is responsible for ensuring this gets called when the corresponding hardware's interrupt is asserted. Registering the ISR should be done after calling init, and before calling start. The driver's ISR will not attempt to lock any locks, but will perform client callbacks. If the client wishes to defer processing to non-interrupt time, it is responsible for doing so.

7.8. ASF_CheckIfASFSupported

Function Declaration

`uint32_t ASF_CheckIfASFSupported(`*`privateData`*`, `*`asfFeature`*`);`

```
const ASF_PrivateData *privateData
```

`ASF_EventErrorType asfFeature`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`asfFeature`	[in]
	Feature required for checking.

Description

Function checks if controller support selected ASF features.

Return Value

CDN_EOK on success - required ASF feature is supported by IP core.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if required ASF features is not supported.

7.9. ASF_GetSupportedASF

Function Declaration

`uint32_t ASF_GetSupportedASF(`*`privateData`*, *`asf_flag`*`);`

`const ASF_PrivateData *privateData`

`uint32_t *asf_flag`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`asf_flag`	[out]
	Bit flags specifies supported ASF features.

Description

Function by means of asf_flag returns all supported ASF features.

Features are returned in the form of bit mask (e.g 1 << ASF_INTEGRITY | 1 << ASF_PROTOCOL | 1 << ASF_DATA_PARITY).

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.10. ASF_GetSupportedTimeoutErrors

Function Declaration

`uint32_t ASF_GetSupportedTimeoutErrors(`*`privateData`*, *`flag`*`);`

`const ASF_PrivateData *privateData`

`uint32_t *flag`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`flag`	[out]
	Bit flags specifies supported timeout errors.

Description

Function gets supported timeout faults.

The number and meaning of timeout faults are IP specific and can vary depending on the IP controller. Core Driver is able to detect how many and which faults are supported but doesn't have knowledge what they mean. For this reason the interpretation of these faults should be done by upper layer. Supported timeout faults are returned in the form of bit mask, where each bit corresponds to different type of timeout fault

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.11. ASF_GetSupportedProtocolErrors

Function Declaration

`uint32_t ASF_GetSupportedProtocolErrors(`*`privateData`*, *`flag`*`);`

`const ASF_PrivateData *privateData`

`uint32_t *flag`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`flag`	[out]
	Bit flags specifies supported protocol errors.

Description

Function gets supported timeout faults.

The number and meaning of timeout faults are IP specific and can vary depending on the IP controller. Core Driver is able to detect how many and which faults are supported but doesn't have knowledge what they mean. For this reason the interpretation of these faults should be done by upper layer. Supported timeout faults are returned in the form of bit mask, where each bit corresponds to different type of timeout fault

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.12. ASF_SelfTest

Function Declaration

`uint32_t ASF_SelfTest(`*`privateData`*`);`

`ASF_PrivateData *privateData`

Parameter List

`privateData` [in]

Driver state info specific to this instance.

Description

Run self test to checks if all events are generated for supported ASF features.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_EPROTO if one of available fault events was not generated.

7.13. ASF_TestEvent

Function Declaration

`uint32_t ASF_TestEvent(`*`privateData`*`, `*`eventType`*`);`

`ASF_PrivateData *privateData`

`ASF_EventErrorType eventType`

Parameter List

`privateData` [in]

	Driver state info specific to this instance.
`eventType`	[in]
	Indicating which test will be started.

Description

Function runs single test for a given type of fault event (see ASF_EventErrorType).

Test forces generate selected type's event and checks if that event has occurred.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_EPROTO if selected test doesn't generate event.

CDN_ENOTSUP if selected ASF feature is not implemented.

7.14. ASF_EnableEvent

Function Declaration

`uint32_t ASF_EnableEvent(`*`privateData`*, *`eventType`*);

`ASF_PrivateData *privateData`

`ASF_EventErrorType eventType`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`eventType`	[in]
	Fault event that will be enabled..

Description

Enable generation one of 7 available fault events.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if selected ASF feature is not implemented.

7.15. ASF_DisableEvent

Function Declaration

`uint32_t ASF_DisableEvent(`*`privateData`*, *`eventType`*);

`ASF_PrivateData *privateData`

`ASF_EventErrorType eventType`

Parameter List

`privateData`	[in] Driver state info specific to this instance.
`eventType`	[in] Fault event that will be disabled.

Description

Disable generation one of 7 available fault events.

Return Value

CDN_EOK on success

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if selected ASF feature is not implemented.

7.16. ASF_EnableAllEvents

Function Declaration

`uint32_t ASF_EnableAllEvents(`*`privateData`*);

`ASF_PrivateData *privateData`

Parameter List

`privateData`	[in] Driver state info specific to this instance.

Description

Enable generation all seven available fault events, provided that they are implemented in IP controller.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.17. ASF_DisableAllEvents

Function Declaration

`uint32_t ASF_DisableAllEvents(`*`privateData`*`);`

`ASF_PrivateData *privateData`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.

Description

Disable generation all available fault events.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.18. ASF_SetEventAsNonFatal

Function Declaration

`uint32_t ASF_SetEventAsNonFatal(`*`privateData`*`, `*`eventType`*`);`

`ASF_PrivateData *privateData`

`ASF_EventErrorType eventType`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`eventType`	[in]
	Fault event that will be treated as non-fatal.

Description

Sets selected fault events as non-fatal.

By default after initialization all faults events are treated as fatal.

Return Value

CDN_EOK on success

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if selected ASF feature is not implemented.

7.19. ASF_SetEventAsFatal

Function Declaration

`uint32_t ASF_SetEventAsFatal(`*`privateData`*, *`eventType`*`);`

`ASF_PrivateData *privateData`

`ASF_EventErrorType eventType`

Parameter List

`privateData`	[in] Driver state info specific to this instance.
`eventType`	[in] Fault event that will be treated as fatal.

Description

Sets selected fault events as fatal.

By default after initialization all faults events are treated as fatal.

Return Value

CDN_EOK on success

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if selected ASF feature is not implemented.

7.20. ASF_EnableProtocolEventByMask

Function Declaration

`uint32_t ASF_EnableProtocolEventByMask(`*`privateData`*, *`mask`*`);`

`ASF_PrivateData *privateData`

`uint32_t mask`

Parameter List

`privateData`	[in] Driver state info specific to this instance.
`mask`	[in] Bit mask indicating which fault events will be enabled.

Description

Enables generation selected by mask protocol fault events.

The types of protocol fault events are IP specific. All protocol faults for which bits are set will be enabled. This function doesn't disable any enabled fault events. All not supported protocol fault event will be ignored.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.21. ASF_EnableProtocolEventByID

Function Declaration

`uint32_t ASF_EnableProtocolEventByID(`*`privateData`*, *`id`*`);`

`ASF_PrivateData *privateData`

`uint8_t id`

Parameter List

`privateData`	[in] Driver state info specific to this instance.
`id`	[in] Index of protocol fault event that will be enabled.

Description

Enables generation selected by ID protocol fault events.

The types of protocol fault events are IP specific.This function allows to enable handling selected by index fault event. Index is the bit number in appropriate ASF register and it is numbered starting from 1 to max 32. The number of available protocol fault events is IP specific.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if required protocol fault event is not supported.

7.22. ASF_DisableProtocolEventByMask

Function Declaration

`uint32_t ASF_DisableProtocolEventByMask(`*`privateData`*, *`mask`*`);`

`ASF_PrivateData *privateData`

`uint32_t mask`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`mask`	[in]
	Bit mask indicating which fault events will be disabled.

Description

Disables generation of timeout fault events selected by mask.

The types of protocol fault events are IP specific. All protocol faults for which bits are set will be disabled. This function doesn't enable any disabled fault events. All not supported protocol fault event will be ignored.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.23. ASF_DisableProtocolEventByID

Function Declaration

`uint32_t ASF_DisableProtocolEventByID(`*`privateData`*, *`id`*`);`

`ASF_PrivateData *privateData`

`uint8_t id`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`id`	[in]
	Index of protocol fault event that will be enabled.

Description

Disables generation of timeout fault events selected by ID.

The types of protocol fault events are IP specific.This function allows to disable handling selected by index fault event. Index is the bit number in appropriate ASF register and it is numbered starting from 1 to max 32. The number of available protocol fault events is IP specific.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if required protocol fault event is not supported.

7.24. ASF_EnableTimeoutEventByMask

Function Declaration

`uint32_t ASF_EnableTimeoutEventByMask(`*`privateData`*, *`mask`*`);`

`ASF_PrivateData *privateData`

`uint32_t mask`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`mask`	[in]
	Bit mask indicating which timeout events will be enabled.

Description

Enables generation selected by mask timeout fault events.

The types of timeout fault events are IP specific. All timeout faults for which bits are set will be enabled. This function doesn't disable any enabled timeout events. All not supported timeout fault event will be ignored.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.25. ASF_EnableTimeoutEventByID

Function Declaration

`uint32_t ASF_EnableTimeoutEventByID(`*`privateData`*, *`id`*`);`

`ASF_PrivateData *privateData`

`uint8_t id`

Parameter List

`privateData`	[in]

	Driver state info specific to this instance.
`id`	[in]
	Index of timeout fault event that will be enabled.

Description

Enables generation selected by ID timeout fault events.

The types of timeout fault events are IP specific.This function allows to enable handling selected by index timeout event. Index is the bit number in appropriate ASF register and it is numbered starting from 1 to max 32. The number of available timeout fault events is IP specific.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP If id exceeds the number of supported timeout fault events.

7.26. ASF_DisableTimeoutEventByMask

Function Declaration

`uint32_t ASF_DisableTimeoutEventByMask(`*`privateData`*, *`mask`*);

`ASF_PrivateData *privateData`

`uint32_t mask`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`mask`	[in]
	Bit mask indicating which fault events will be disabled.

Description

Disables generation of timeout fault events selected by mask.

The types of timeout fault events are IP specific. All timeout faults for which bits are set will be disabled. This function doesn't enable any disabled fault events. All not supported timeout fault event will be ignored.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.27. ASF_DisableTimeoutEventByID

Function Declaration

`uint32_t ASF_DisableTimeoutEventByID(`*`privateData`*`, `*`id`*`);`

`ASF_PrivateData *privateData`

`uint8_t id`

Parameter List

`privateData` [in]

Driver state info specific to this instance.

`id` [in]

Index of timeout fault event that will be enabled.

Description

Disables generation of timeout fault events selected by ID.

The types of timeout fault events are IP specific.This function allows to disable handling selected by index fault event. Index is the bit number in appropriate ASF register and it is numbered starting from 1 to max 32. The number of available timeout fault events is IP specific.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

CDN_ENOTSUP if required timeout fault event is not supported.

7.28. ASF_GetStatistic

Function Declaration

`uint32_t ASF_GetStatistic(`*`privateData`*`, `*`stats`*`);`

`const ASF_PrivateData *privateData`

`ASF_StatInfo *stats`

Parameter List

`privateData` [in]

Driver state info specific to this instance.

`stats` [out]

Pointer to object that will be filled with statistic data.

Description

Function fills the ASF_statistisc object passed by stats parameter with all collected fault events.

For the protection of data, Core Driver returns to requester only copies of gathered data.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.29. ASF_ClearStatistic

Function Declaration

`uint32_t ASF_ClearStatistic(`*`privateData`*`);`

`ASF_PrivateData *privateData`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.

Description

Function clears the statistics gathered by ASF Core Driver.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid.

7.30. ASF_RestoreStatistic

Function Declaration

`uint32_t ASF_RestoreStatistic(`*`privateData`*`, `*`stats`*`);`

`ASF_PrivateData *privateData`

`const ASF_StatInfo *stats`

Parameter List

`privateData`	[in]
	Driver state info specific to this instance.
`stats`	[in]

Pointer to object with data being restored.

Description

Function restores the statistics from saved copies.

After restoring of data driver will continue collection of data on restored object. Before this operation driver has to be stopped. Function can be used to restore data after rebooting whole system.

Return Value

CDN_EOK on success.

CDN_EINVAL if one of the parameter are invalid