SafeXcel IP™
SafeXcel-IP-201

Driver Library v3.0

User Guide

Document Revision: F
Document Date: 2013-03-04
Document Number: 007-201300-307
Document Status: Accepted

INSIDE Secure B.V.
Boxtelseweg 26A
5261 NE Vught
The Netherlands
Phone: +31-73-6581900
Fax: +31-73-6581999
http://www.insidesecure.com/

For further information contact: ESSEmbeddedHW-Support@insidesecure.com

Revision History

Doc Rev	Page(s) Section(s)	Date	Author	Purpose of Revision
A	All	2009-04-17	RWI	• First release
B	Several	2010-01-27	RWI	• Updated for Driver Framework v4.0 • Finalized for EIP-201 Driver Library v2.0
C	Several	2010-03-12	JBO, EV	• Update for version 3.0
D	All	2010-09-03	MHO	• Applied document template • Minor updates
E	All	2011-08-23	AB	• Applied new document template • Minor style updates
F		2013-03-04	FvdM	• Applied new document template

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Purpose

This document is intended for software developers who want to use the services of the *SafeXcel-IP-201 HW1.1 and HW1.2 Advanced Interrupt Controller* via the *SafeXcel-IP-201 Driver Library*. This document explains in detail how to use the *SafeXcel-IP-201 Driver Library* to configure and use the *SafeXcel-IP-201 Advanced Interrupt Controller* in an optimal way.

Chapter 2 contains a high-level overview of the software and hardware.

Chapter 3 describes each service in detail.

Throughout this document, the *SafeXcel-IP-201 Advanced Interrupt Controller* is also referred to as the *AIC*. Further, the terms 'Host' and 'CPU' are used interchangeably to mean the Central Processing Unit on which the driver software runs.

Note: *This document does not list all the parameters in detail. The reader is advised to refer to the API header file `eip201.h` [4] for the parameter details of each function.*

1.2 Related Documents

The following documents are part of the documentation set.

Ref.	Document Name	Document Number
[1]	Driver Framework Porting Guide	007-900420-304
[2]	SafeXcel-IP-201 HW1.1 Programmer Manual	009658-001
[3]	SafeXcel-IP-201 HW1.2 Programmer Manual	Internal document
[4]	EIP-201 Driver Library source code, the `eip201.h` API header file	n/a
[5]	SafeXcel-IP-201 Driver Library User Guide (this document)	007-201300-307

This information is correct at the time of document release. INSIDE Secure reserves the right to update the related documents without updating this document. Please contact INSIDE Secure for the latest document revisions.

For more information or support, please go to https://essoemsupport.insidesecure.com/ for our online support system. In case you do not have an account for this system, please ask one of your colleagues who already has an account to create one for you or send an e-mail to ESSEmbeddedHW-Support@insidesecure.com.

1.3 Abbreviations and Definitions

AIC	Advanced Interrupt Controller
API	Application Programming Interface
LSB	Least Significant Byte
LSW	Least Significant Word
MSB	Most Significant Byte
MSW	Most Significant Word
OS	Operating System

2 Functional Overview

This chapter provides a high-level overview of the software and hardware architecture.

2.1 Architectural Overview

The *AIC* combines many interrupt sources into one interrupt signal. The *EIP-201 Driver Library* configures the *AIC* and acknowledges interrupts.

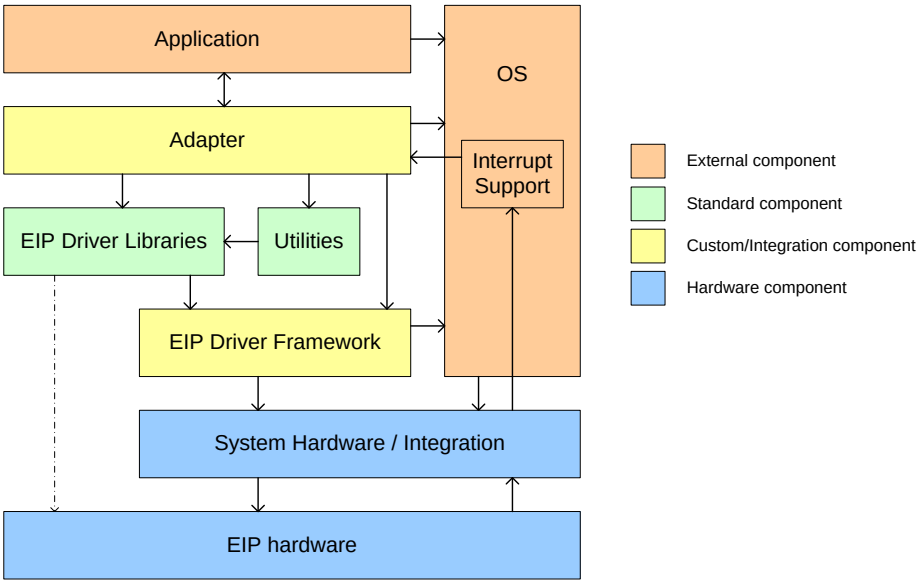

Figure 1 Architectural System Overview

The *Adapter* layer is aware of the constraints set by the *EIP-201 Driver Library* and by other parts of the hardware. The *Adapter* layer has the following tasks:

- Interrupt hooking and handling
- Request serialization
- Application interfacing
- Virtual/logical memory handling (if needed)
- Cache coherency handling (if needed)
- Concurrent context synchronization (if needed)
- Dynamic memory allocation (if needed).

Note: *The* Adapter *layer is not part of the driver library and must be custom-made for each environment, together with the EIP Driver Framework. Refer to section 2.2.*

The *EIP-201 Driver Library* is a building block that is clean from all the typical system issues listed above. It abstracts the *AIC*, thereby removing the need to understand the HW registers and commands/responses in higher SW layers.

2.2 EIP Driver Framework

This software block provides decoupling of integration issues (CPU, compiler, integration hardware) for the *EIP-201 Driver Library*. This allows the *EIP-201 Driver Library* to be a standard, reusable component. The *EIP Driver Framework* is a set of APIs/interfaces (see [1]) for the *EIP-201 Driver Library* to use which are classified as follows:

- Basic Definitions API: Types with guaranteed storage size and a few support macros.
- C library API: Guarantees the existence of a subset of the C library API.
- Device API: Access to EIP registers and embedded memories.
- DMA Resource API: Provides functionality for dynamic resources such as DMA buffers, namely address translation, data coherency and read/write operations, DMA-safe buffer allocation.

The *EIP-201 Driver Library* requires an implementation of the Basic Definitions API and Device API for the specific environment it will be used in.

2.3 Execution Contexts

The *EIP-201 Driver Library* always executes in the context of the caller and does not require a separate schedulable entity (process or thread). This gives the *Adapter* layer complete control over context switches.

2.4 Interrupt Sources and Bitmasks

The SafeXcel-IP-201 Advanced Interrupt Controller supports up to 32 interrupt sources numbered 0 through 31.

In the *EIP-201 Driver Library* API, the types `EIP201_Source_t` and `EIP201_SourceBitmap_t` are used to represent a single interrupt source and multiple interrupt sources respectively. Both types have been declared as a `uint32_t` and require one or more bits to be set to represent the interrupt source(s). The Basic Definition macros `BIT_0` up to `BIT_31` can be used for this.

Examples:

```
EIP201_Source_t SingleInt = BIT_5;
EIP201_SourceBitmap_t ManyInts = BIT_12 + BIT_5 + BIT_0;
```

EIP-201 Driver Library API functions that operate on an individual interrupt source use the type `EIP201_Source_t`, while functions that operate on many interrupt sources at once have a parameter typed `EIP201_SourceBitmask_t`.

2.5 Device Handle

The *EIP-201 Driver Library* API requires a parameter named Device, typed `Device_Handle_t`. This value is returned by the EIP Driver Framework Device API function `Device_Find()` and represents the *AIC* device instance. Using this value the *EIP-201 Driver Library* can access the *AIC* registers.

In a system with more than one *SafeXcel-IP-201 Advanced Interrupt Controller*, the same driver code can control all *AIC* instances by using different Device parameter value. The EIP Driver Framework implementation of the Device API must route the register read/write requests to the correct *AIC* instance, based on this Device parameter.

2.6 Driver Configuration

The file `cs_eip201.h` contains all the Configuration Switches for the *EIP-201 Driver Library*. This file defines concrete values for these switches at a product level. It is to be created at the Application level and customized according to its needs. The configuration options are described in Chapter 3.5.

2.7 Synchronization

The *EIP-201 Driver Library* API does not contain any mandatory state, which means the Adapter layer can call the functions in any order. A typical driver would not rely on the values in the registers when the software is loaded. The `EIP201_Initialize` service brings the hardware to a known state by setting the configuration and interrupt-enable masks, but this is not mandated.

A few functions cannot execute concurrently, as shown in Table 1.

Table 1 API function non-reentrance and synchronization groups

API	Re-entrant?	Sync Group
`EIP201_Initialize`	No	1
`EIP201_Config_Change`	No	1
`EIP201_SourceMask_EnableSource`	Yes	-
`EIP201_SourceMask_DisableSource`	Yes	-
`EIP201_Acknowledge`	Yes	-
`EIP201_Config_Read`	Yes	-
`EIP201_SourceMask_SourceIsEnabled`	Yes	-
`EIP201_SourceMask_ReadAll`	Yes	-
`EIP201_SourceStatus_IsEnabledSourcePending`	Yes	-
`EIP201_SourceStatus_IsRawSourcePending`	Yes	-
`EIP201_SourceStatus_ReadAllEnabled`	Yes	-
`EIP201_SourceStatus_ReadAllRaw`	Yes	-

Functions in the same ‘Sync Group’ must not execute concurrently. The Adapter layer must ensure this cannot happen by employing a synchronization mechanism.

2.8 Error Handling

Almost all of the *EIP-201 Driver Library* functions return the status `EIP201_Status_t`, which may contain following error codes:

- `EIP201_STATUS_UNSUPPORTED_IRQ`
  This error code is returned only when `EIP201_STRICT_ARGS` is enabled in `cs_eip201.h` (see Chapter 3.5).
- `EIP201_STATUS_UNSUPPORTED_HARDWARE_VERSION`
  This error code might be returned by `EIP201_Initialize` when for the supplied instance no suitable hardware is detected.

2.9 Endianness

All register I/O operations are 32 bits wide and operate through the Device API, which must handle potential endianness differences in the system.

2.10 Interrupt Primer

Interrupt signals can be level-based, which means they maintain a specific level (high or low) while inactive and change state for the duration they are active.

In order to combine active-high and active-low interrupt signals into a single interrupt output, which is the task of the *AIC*, some of the interrupts must simply be inverted.

Level-based interrupts have the side effect that the interrupt can go inactive before the CPU has had a chance to service it. To avoid this, the *AIC* supports edge-detection, to detect and remember the change of state in the interrupt signal. The CPU will always see the remembered state, even if the interrupt source has already gone inactive again. The CPU is required to 'acknowledge' the interrupt with the *AIC*, allowing it to detect a new edge.

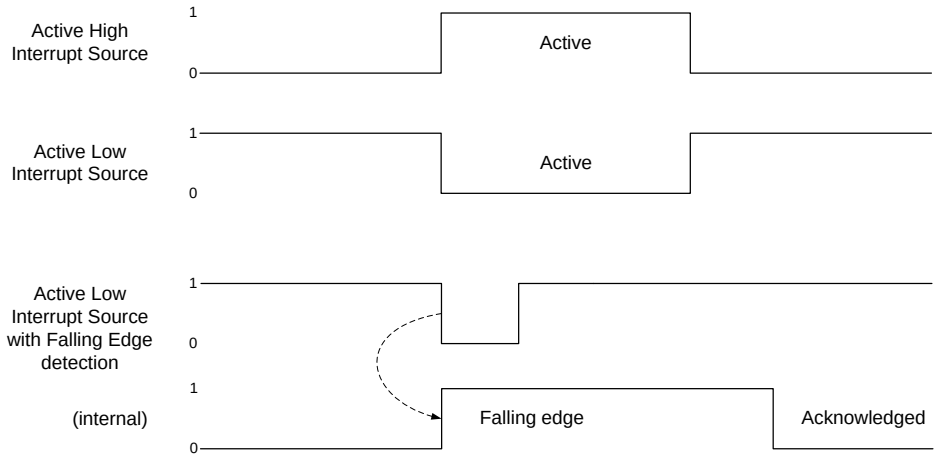

Figure 2 Level-based interrupts and Edge detection

Note: *An edge-detected interrupt can have very short active-periods, as shown in the drawing above. If an interrupt source has many short pulses, the edge-detector remembers the first edge only. All further edges have no effect until the host has acknowledged the detected edge.*

2.11 AIC Functional Blocks

Figure 3 shows the functional blocks of the *AIC*. The interrupt sources are on the left and the combined output from the *AIC* is on the right.

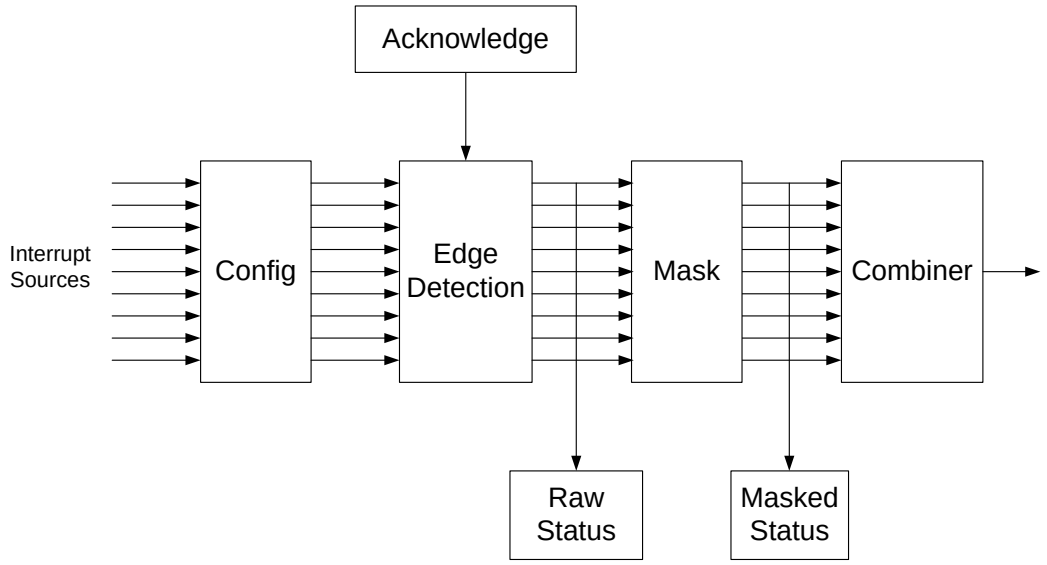

Figure 3 AIC Functional blocks

The ‘Config’ block deals with active-high and active-low differences.

The ‘Edge Detection’ block detects and remembers the edges for individual interrupt sources configured for this. The ‘Acknowledge’ block resets the detected interrupts.

The output of the ‘Edge Detection’ block is the status of pending interrupts. This status is available as the ‘Raw Status’.

The ‘Mask’ block stops unwanted interrupts from being included in the combined output. The ‘Masked Status’ only shows pending interrupts that are not disabled by the mask.

Note: *Acknowledge has no effect on level-based interrupt sources. For these, the source of the interrupt must be removed (in the respective EIP device) for the interrupt to go inactive.*

3 Provided Services

This chapter describes how to use each of the available services. The reader is referred to the header file `eip201.h` for details on each function parameter.

3.1 Interrupt Configuration

The interrupt configuration service allows configuring of each individual interrupt source based on one of the following four types as introduced in section 2.10:

- Active Low
- Active High
- Rising edge
- Falling edge

Summary of the related API functions:

API	Description
EIP201_Config_Change	Replace the configuration for one or more interrupt sources.
EIP201_Config_Read	Read the current configuration for one interrupt source.
EIP201_Initialize	Set the configuration and mask for all interrupt sources.

3.2 Interrupt Masking

The functions in this category operate on the mask block (see Figure 3), providing the basic service to ‘enable’ and ‘disable’ interrupts.

API	Description
EIP201_SourceMask_EnableSource	Enable one or more interrupt sources in the mask.
EIP201_SourceMask_DisableSource	Disable one or more interrupt sources in the mask.
EIP201_SourceMask_SourceIsEnabled	Return the mask status for one interrupt source.
EIP201_SourceMask_ReadAll	Return the mask status for all interrupt sources.
EIP201_Initialize	Set the configuration and mask for all interrupt sources.

3.3 Interrupt Status

This group of functions allows the status of each interrupt source to be checked. Refer to Figure 3. An interrupt is ‘pending’ when it is active. Edge-detected interrupts are pending until acknowledged. Section 2.10 contains more details on this.

API	Description
EIP201_SourceStatus_IsEnabledSourcePending	Check if an interrupt is pending, after the mask. Masked (disabled) interrupts are never pending.
EIP201_SourceStatus_IsRawSourcePending	Check if interrupt is pending before the mask.
EIP201_SourceStatus_ReadAllEnabled	Read pending status for all interrupts, after the mask.
EIP201_SourceStatus_ReadAllRaw	Read pending status for all interrupts, before the mask.

3.4 Interrupt Handling

When the *AIC* interrupt output (on the right side of Figure 3) interrupts the CPU, the Adapter layer asks the *AIC* which source(s) require service. The interrupts to be serviced must be acknowledged.

API	Description
EIP201_SourceStatus_IsEnabledSourcePending	Check for one specific interrupt source if it is pending.
EIP201_SourceStatus_ReadAllEnabled	Return all interrupt sources that are pending.
EIP201_Acknowledge	Acknowledge one or more edge-detected interrupt sources, starting a new detection.

Example:

```
void
MyInterruptHandler(void)
{
    uint32_t ActiveSources;

    do
    {
        ActiveSources = EIP201_SourceStatus_ReadAllEnabled(Device);

        EIP201_Acknowledge(Device, ActiveSources);

        DecodeAndHandleActiveInterrupts(ActiveSources);

    }
    while(ActiveSources);
}
```

3.5 Driver Library Configuration Options

This section describes the configuration options of the *EIP-201 Driver Library*. The operations can be set in the `cs_eip201.h` file. The following options can be set in this configuration file:

- `EIP201_STRICT_ARGS`
  When this option is defined, a number of parameter-testing macros expand to extra error checking code inside the *EIP-201 Driver Library*. Use this option to debug the *Adapter* layer since it allows simple debugging of the invalid parameters.
  When this option is not defined (i.e. when it is disabled), these same macros expand to 'nothing', thereby saving on footprint and increasing performance.
- `EIP201_STRICT_ARGS_MAX_NUM_OF_INTERRUPTS`
  This option is required for strict-argument checking of the selected interrupt source(s). If the system for example has 10 interrupts, only bits 0 through 9 may be set. The strict-argument implementation checks if any of the higher bits [31:10] are set, and if so, returns `EIP201_STATUS_UNSUPPORTED_IRQ`.

3.6 EIP201_REMOVE_*

These 12 options allow compile-time removal of specific services from the *EIP-201 Driver Library*, thereby reducing footprint.

```
EIP201_REMOVE_INITIALIZE
EIP201_REMOVE_CONFIG_CHANGE
EIP201_REMOVE_CONFIG_READ
EIP201_REMOVE_SOURCEMASK_ENABLESOURCE
EIP201_REMOVE_SOURCEMASK_DISABLESOURCE
EIP201_REMOVE_SOURCEMASK_SOURCEISENABLED
EIP201_REMOVE_SOURCEMASK_READALL
EIP201_REMOVE_SOURCESTATUS_ISENABLEDSOURCEPENDING
EIP201_REMOVE_SOURCESTATUS_ISRAWSOURCEPENDING
EIP201_REMOVE_SOURCESTATUS_READALLENABLED
EIP201_REMOVE_SOURCESTATUS_READALLRAW
EIP201_REMOVE_ACKNOWLEDGE
```

(End of Document)