SMART ARM-based Microcontrollers

AT06863: SAM4L Peripheral Event Controller (PEVC) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides a unified interface for the configuration and management of the Event Channels.

The peripheral event generators and users are interconnected by a network known as the Peripheral Event System.

The Peripheral Event System allows low latency peripheral-to-peripheral signaling without CPU intervention, and without consuming system resources such as bus or RAM bandwidth. This offloads the CPU and system resources compared to a traditional interrupt-based software driven system.

Devices from the following series can use this module:

- Atmel | SMART SAM4L

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

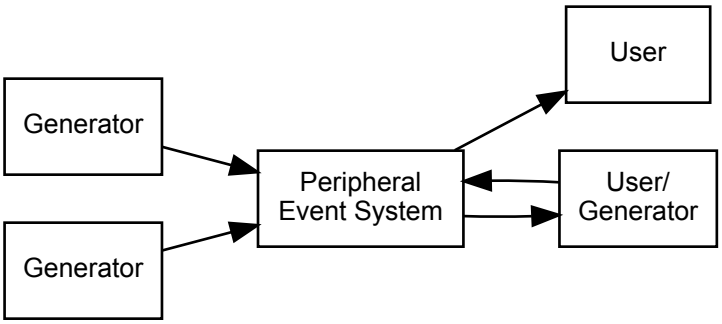
There are no prerequisites for this module.

3. Module Overview

Peripherals within the SAM4L device are capable of generating two types of actions in response to a given stimulus; they can set a register flag for later intervention by the CPU (using interrupt or polling methods), or they can generate event signals which can be internally routed directly to other peripherals within the device. The use of events allows for direct actions to be performed in one peripheral in response to a stimulus in another without CPU intervention. This can lower the overall power consumption of the system if the CPU is able to remain in sleep modes for longer periods, and lowers the latency of the system response.

The Peripheral Event System is comprised of a number of freely configurable Event Channels, plus a number of fixed Event Users. Each Event Channel can be configured to select the input peripheral that will generate the events on the channel, as well as the Event Shaper (EVS) and Input Glitch Filter (IGF) operating modes. The fixed-function Event Users, connected to peripherals within the device, can then subscribe to an Event Channel in a one-to-many relationship in order to receive events as they are generated. An overview of the event system chain is shown in Figure 3-1 Module Overview on page 6.

Figure 3-1 Module Overview



There are many different events that can be routed in the device, which can then trigger many different actions. For example, an Analog Comparator module could be configured to generate an event when the input signal rises above the compare threshold, which then triggers a Timer Counter module to capture the current count value for later use.

3.1. Event Channels

The Peripheral Event Controller module in the SAM4L device consists of several channels, which can be freely linked to an Event Generator (i.e. a peripheral within the device that is capable of generating events). Each channel can be individually configured to select the generator peripheral, signal path, Event Shaper (EVS), and Input Glitch Filter (IGF) applied to the input event signal, before being passed to any event user(s).

Event Channels can support multiple users within the device in a standardized manner; when an Event User is linked to an Event Channel, the channel will automatically handshake with all attached users to ensure that all modules correctly receive and acknowledge the event.

3.2. Event Users

Event Users are able to subscribe to an Event Channel, once it has been configured. Each Event User consists of a fixed connection to one of the peripherals within the device (for example, an ADC module or Timer module) and is capable of being connected to a single Event Channel.

3.3. Event Shaper (EVS)

The Peripheral Event Controller module contains Event Shapers (EVS) for external inputs and general purpose waveforms (i.e. timer outputs or Generic Clocks) that require synchronisation and/or edge detection prior to peripheral event propagation.

Each Event Shaper is responsible for shaping one generator input prior to it going through an Event Channel.

Refer to the module configuration section at the end of the Peripheral Event Controller (PEVC) section in the device datasheet for the specific configuration of Event Shapers and Input Glitch Filters.

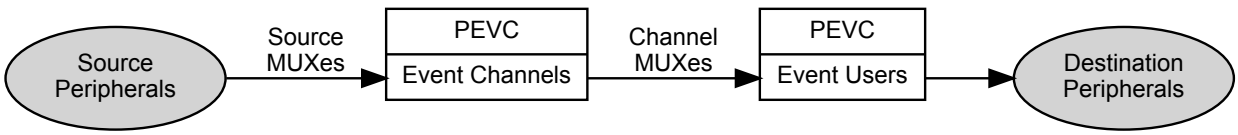
3.3.1. Input Glitch Filter (IGF)

The Peripheral Event Controller module contains Input Glitch Filters (IGF) specifically to allow I/O inputs to be sampled periodically. Input Glitch Filtering can be turned on or off in the Event Shaper associated with the Event Channel.

3.4. Physical Connection

Figure 3-2 Physical Connection on page 7 shows how this module is interconnected within the device.

Figure 3-2 Physical Connection



3.5. Configuring Events

Several steps are required to properly configure an event chain, so that hardware peripherals can respond to events generated by each other, listed below.

3.5.1. Source Peripheral

1. The source peripheral (that will generate events) must be configured and enabled.
2. The source peripheral (that will generate events) must have an output event enabled.

3.5.2. Event System

1. The event system channel must be configured and enabled, with the correct source peripheral selected as the channel's Event Generator.
2. The event system user must be configured and enabled, with the correct source Event Channel selected as the source.

3.5.3. Destination Peripheral

1. The destination peripheral (that will receive events) must be configured and enabled.
2. The destination peripheral (that will receive events) must have an input event enabled.

4. Special Considerations

There are no special considerations for this module.

5. Extra Information

For extra information, see Extra Information for Peripheral Event Controller Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for Peripheral Event Controller Driver.

7. API Overview

7.1. Structure Definitions

7.1.1. Struct events_ch_conf

Configuration structure for an Event Channel.

Table 7-1 Members

Type	Name	Description
uint32_t	channel_id	Channel to configure (user)
uint32_t	generator_id	Event generator to connect to the channel
enum events_igf_edge	igf_edge	Edge detection for Event Channels
bool	shaper_enable	Enable Event Shaper (EVS) or not

7.1.2. Struct events_conf

Configuration structure for event module.

Table 7-2 Members

Type	Name	Description
enum events_igf_divider	igf_divider	Input Glitch Filter divider

7.2. Macro Definitions

7.2.1. Macro EVENT_CHANNEL_N

```
#define EVENT_CHANNEL_N
```

Maximum number for Event Channels (users).

7.2.2. Macro EVENT_GENERATOR_N

```
#define EVENT_GENERATOR_N
```

Maximum number for event generator.

7.3. Function Definitions

7.3.1. Function events_ch_clear_overrun_status()

Clear the overrun status of an Event Channel.

```
void events_ch_clear_overrun_status(
        uint32_t channel_id)
```

Table 7-3 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.2. Function events_ch_clear_trigger_status()

Clear the trigger status of an Event Channel.

```
void events_ch_clear_trigger_status(
        uint32_t channel_id)
```

Table 7-4 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.3. Function events_ch_configure()

Configure an Event Channel.

```
void events_ch_configure(
        struct events_ch_conf *const config)
```

Table 7-5 Parameters

Data direction	Parameter name	Description
[in, out]	config	Configuration settings for the Event Channel

7.3.4. Function events_ch_disable()

Disable an Event Channel.

```
void events_ch_disable(
        uint32_t channel_id)
```

Table 7-6 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.5. Function events_ch_disable_software_trigger()

Disable the software trigger for an Event Channel.

```
void events_ch_disable_software_trigger(
        uint32_t channel_id)
```

Table 7-7 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.6. Function events_ch_enable()

Enable an Event Channel.

```
void events_ch_enable(
        uint32_t channel_id)
```

Table 7-8 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.7. Function events_ch_enable_software_trigger()

Enable the software trigger for an Event Channel.

```
void events_ch_enable_software_trigger(
        uint32_t channel_id)
```

Table 7-9 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.8. Function events_ch_get_config_defaults()

Initialize an Event Channel configuration structure to defaults.

```
void events_ch_get_config_defaults(
        struct events_ch_conf *const config)
```

The default configuration is as follows:

- Channel ID is initialized to invalid number
- Generator ID is initialized to invalid number
- Event shaper is disabled
- Event Input Glitch Filter is disabled

Table 7-10 Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.3.9. Function events_ch_is_enabled()

Get the status (enabled or disabled) of an Event Channel.

```
bool events_ch_is_enabled(
        uint32_t channel_id)
```

Table 7-11 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

Returns
The Event Channel enabled/disabled status.

Table 7-12 Return Values

Return value	Description
true	Event Channel is enabled
false	Event Channel is disabled

7.3.10. Function events_ch_is_overrun()

Get the overrun status of an Event Channel.

```
bool events_ch_is_overrun(
        uint32_t channel_id)
```

Table 7-13 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

Returns
The Event Channel overrun status.

Table 7-14 Return Values

Return value	Description
true	A channel overrun event has occurred
false	A channel overrun event has not occurred

7.3.11. Function events_ch_is_ready()

Get the busy status of an Event Channel.

```
bool events_ch_is_ready(
        uint32_t channel_id)
```

Table 7-15 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

Returns
The Event Channel busy status.

Table 7-16 Return Values

Return value	Description
true	If the Event Channel is ready to be used
false	If the Event Channel is currently busy

7.3.12. Function events_ch_is_triggered()

Get the trigger status of an Event Channel.

```
bool events_ch_is_triggered(
        uint32_t channel_id)
```

Table 7-17 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

Returns
The Event Channel trigger status.

Table 7-18 Return Values

Return value	Description
true	A channel event has occurred
false	A channel event has not occurred

7.3.13. Function events_ch_software_trigger()

Trigger a Software Event for the corresponding Event Channel.

```
void events_ch_software_trigger(
        uint32_t channel_id)
```

Table 7-19 Parameters

Data direction	Parameter name	Description
[in]	channel_id	Event Channel ID

7.3.14. Function events_disable()

Disable the events module.

```
void events_disable( void )
```

7.3.15. Function events_enable()

Enable the events module.

```
void events_enable( void )
```

7.3.16. Function events_get_config_defaults()

Initialize an events configuration structure to defaults.

```
void events_get_config_defaults(
        struct events_conf *const config)
```

The default configuration is as follows:

- Input Glitch Filter Divider is set to EVENT_IGF_DIVIDER_1024

Table 7-20 Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.3.17. Function events_init()

Initialize the events module.

```
void events_init(
        struct events_conf *const config)
```

Table 7-21 Parameters

Data direction	Parameter name	Description
[in]	config	Configuration structure to initialize to default values

7.3.18. Function events_set_igf_divider()

Set the Input Glitch Filter Divider.

```
void events_set_igf_divider(
        enum events_igf_divider divider)
```

Table 7-22 Parameters

Data direction	Parameter name	Description
[in]	divider	Input Glitch Filter divider

Note: As stated in the datasheet, there is one divider value for all Event Shaper (EVS) instances.

7.4. Enumeration Definitions

7.4.1. Enum events_igf_divider

Enumerate for the possible division ratios of an Input Glitch Filter.

Table 7-23 Members

Enum value	Description
EVENT_IGF_DIVIDER_1	Select a prescaler division ratio of 1
EVENT_IGF_DIVIDER_2	Select a prescaler division ratio of 2
EVENT_IGF_DIVIDER_4	Select a prescaler division ratio of 4
EVENT_IGF_DIVIDER_8	Select a prescaler division ratio of 8
EVENT_IGF_DIVIDER_16	Select a prescaler division ratio of 16
EVENT_IGF_DIVIDER_32	Select a prescaler division ratio of 32
EVENT_IGF_DIVIDER_64	Select a prescaler division ratio of 64
EVENT_IGF_DIVIDER_128	Select a prescaler division ratio of 128
EVENT_IGF_DIVIDER_256	Select a prescaler division ratio of 256
EVENT_IGF_DIVIDER_512	Select a prescaler division ratio of 512
EVENT_IGF_DIVIDER_1024	Select a prescaler division ratio of 1024
EVENT_IGF_DIVIDER_2048	Select a prescaler division ratio of 2048
EVENT_IGF_DIVIDER_4096	Select a prescaler division ratio of 4096
EVENT_IGF_DIVIDER_8192	Select a prescaler division ratio of 8192
EVENT_IGF_DIVIDER_16384	Select a prescaler division ratio of 16384
EVENT_IGF_DIVIDER_32768	Select a prescaler division ratio of 32768

7.4.2. Enum events_igf_edge

Table 7-24 Members

Enum value	Description
EVENT_IGF_EDGE_NONE	Input Glitch Filter is disabled
EVENT_IGF_EDGE_RISING	Event detection through Input Glitch Filter on rising edge
EVENT_IGF_EDGE_FALLING	Event detection through Input Glitch Filter on falling edge
EVENT_IGF_EDGE_BOTH	Event detection through Input Glitch Filter on both edges

8. Extra Information for Peripheral Event Controller Driver

8.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
ADC	Analog to Digital Converter
AST	Asynchronous Timer
EVS	Event Shaper
IGF	Input Glitch Filter
PDCA	Peripheral DMA Controller

8.2. Dependencies

This driver has the following dependencies:

- None

8.3. Errata

There are no errata related to this driver.

8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial document release

9. Examples for Peripheral Event Controller Driver

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4L Peripheral Event Controller (PEVC) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for the Peripheral Event Controller Driver
- Example for the Peripheral Event System - AST/PDCA
- Example for the Peripheral Event System - GPIO/PDCA

9.1. Quick Start Guide for the Peripheral Event Controller Driver

This is the quick start guide for the SAM4L Peripheral Event Controller (PEVC) Driver, with step-by-step instructions on how to configure and use the driver for a specific use case.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, e.g., the main application function.

9.1.1. Use Cases

- Basic Use Case

9.1.2. Basic Use Case

This use case will demonstrate how to use the Peripheral Event Controller on SAM4L_EK. In this use case, one Event Channel is configured as:

- Configure AST periodic event 0 as a generator
- Configure PDCA channel 0 as a user to transfer one word
- Enable the event shaper for the generator

9.1.3. Setup Steps

9.1.3.1. Prerequisites

This module requires the following service:

- Clock Management (Sysclock)

9.1.3.2. Code Example

Copy-paste the following setup code to your application:

```
static void init_events(void)
{
    struct events_conf    events_config;
    struct events_ch_conf ch_config;

    /* Initialize event module */
    events_get_config_defaults(&events_config);
    events_init(&events_config);
    events_enable();

    /*
     * Configure an event channel
```

```
     * - AST periodic event 0 --- Generator
     * - PDCA channel 0       --- User
     */
    events_ch_get_config_defaults(&ch_config);
    ch_config.channel_id = PEVC_ID_USER_PDCA_0;
    ch_config.generator_id = PEVC_ID_GEN_AST_2;
    ch_config.shaper_enable = true;
    ch_config.igf_edge = EVENT_IGF_EDGE_NONE;
    events_ch_configure(&ch_config);

    /* Enable the channel */
    events_ch_enable(PEVC_ID_USER_PDCA_0);
}
```

Add this to the main loop or a setup function:

```
/* Initialize AST as event generator. */
init_ast();

/* Initialise events for this example. */
init_events();

/* Initialize the PDCA as event user */
init_pdca();
```

9.1.3.3. Workflow

1. Initialize AST to generate periodic event 0. see sam/drivers/events/example1 for more detail:
   ```
   init_ast();
   ```
2. Initialize the event module and enable it:
   ```
   init_events();
   ```
3. Initialize PDCA channel 0 to transfer data to USART. see sam/drivers/events/example1 for more detail:
   ```
   init_pdca();
   ```

9.1.4. Basic Usage

After the channel is configured correctly and enabled, each time a new event from AST occurs, a character is sent to the USART via PDCA without the use of the CPU.

9.2. Example for the Peripheral Event System - AST/PDCA

9.2.1. Introduction

This example shows how to use the Peripheral Event Controller. In the example, the AST generates a periodic event which is transmitted to the PDCA. Each time a new event occurs, a character is sent to the USART without the use of the CPU. The main loop of the function is a delay 500ms and toggle a LED continuously to show CPU activity.

9.2.2. Main Files

- events.c: Events driver
- events.h: Events driver header file

- events_example1.c: Events example 1 application

9.2.3. Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench® for Atmel®. Other compilers may or may not work.

9.2.4. Device Information

SAM4L device can be used.

9.2.5. Configuration Information

This example has been tested with the following configuration:

- PC terminal settings:
  - 115200 baud
  - 8 data bits
  - no parity bit
  - 1 stop bit
  - no flow control

9.3. Example for the Peripheral Event System - GPIO/PDCA

9.3.1. Introduction

This example shows how to use the Peripheral Event Controller. In the example, an I/O pin is configured to trigger a GPIO event when detecting a falling edge. Each time a new event occurs, it will trigger the PDCA to send a character to the USART without CPU usage.

9.3.2. Main Files

- events.c: Events driver
- events.h: Events driver header file
- events_example2.c: Events example 2 application

9.3.3. Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench for Atmel. Other compilers may or may not work.

9.3.4. Device Information

SAM4L device can be used.

9.3.5. Configuration Information

This example has been tested with the following configuration:

- PC terminal settings:
  - 115200 baud
  - 8 data bits
  - no parity bit
  - 1 stop bit

- no flow control

10. Document Revision History

Doc. Rev.	Date	Comments
42312B	07/2015	Updated title of application note and added list of supported devices
42312A	05/2014	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**