SMART ARM-based Microcontrollers

AT03246: SAM D/R/L/C External Interrupt (EXTINT) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of external interrupts generated by the physical device pins, including edge detection. The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripheral is used by this module:

- EIC (External Interrupt Controller)

The following devices can use this module:

- Atmel | SMART SAM D20/D21
- Atmel | SMART SAM R21
- Atmel | SMART SAM D09/D10/D11
- Atmel | SMART SAM L21/L22
- Atmel | SMART SAM DA1
- Atmel | SMART SAM C20/C21

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

The External Interrupt (EXTINT) module provides a method of asynchronously detecting rising edge, falling edge, or specific level detection on individual I/O pins of a device. This detection can then be used to trigger a software interrupt or event, or polled for later use if required. External interrupts can also optionally be used to automatically wake up the device from sleep mode, allowing the device to conserve power while still being able to react to an external stimulus in a timely manner.

3.1. Logical Channels

The External Interrupt module contains a number of logical channels, each of which is capable of being individually configured for a given pin routing, detection mode, and filtering/wake up characteristics.

Each individual logical external interrupt channel may be routed to a single physical device I/O pin in order to detect a particular edge or level of the incoming signal.

3.2. NMI Channels

One or more Non Maskable Interrupt (NMI) channels are provided within each physical External Interrupt Controller module, allowing a single physical pin of the device to fire a single NMI interrupt in response to a particular edge or level stimulus. An NMI cannot, as the name suggests, be disabled in firmware and will take precedence over any in-progress interrupt sources.

NMIs can be used to implement critical device features such as forced software reset or other functionality where the action should be executed in preference to all other running code with a minimum amount of latency.

3.3. Input Filtering and Detection

To reduce the possibility of noise or other transient signals causing unwanted device wake-ups, interrupts, and/or events via an external interrupt channel. A hardware signal filter can be enabled on individual channels. This filter provides a Majority-of-Three voter filter on the incoming signal, so that the input state is considered to be the majority vote of three subsequent samples of the pin input buffer. The possible sampled input and resulting filtered output when the filter is enabled is shown in Table 3-1 Sampled Input and Resulting Filtered Output on page 6.

Table 3-1. Sampled Input and Resulting Filtered Output

Input Sample 1	Input Sample 2	Input Sample 3	Filtered Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0

Input Sample 1	Input Sample 2	Input Sample 3	Filtered Output
1	0	1	1
1	1	0	1
1	1	1	1

3.4. Events and Interrupts

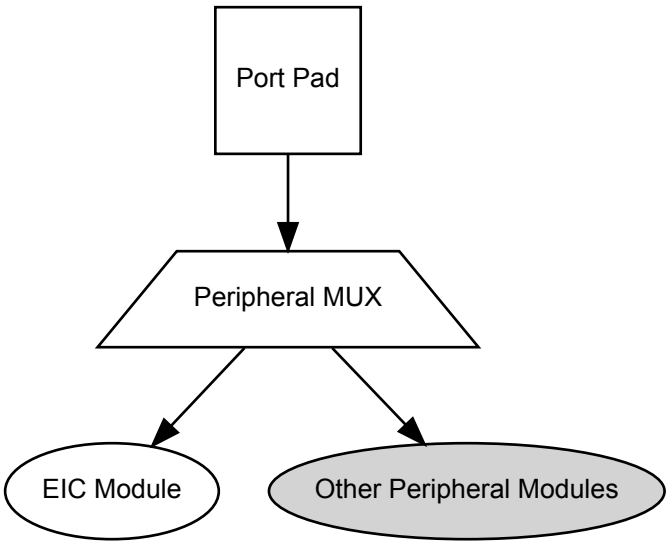
Channel detection states may be polled inside the application for synchronous detection, or events and interrupts may be used for asynchronous behavior. Each channel can be configured to give an asynchronous hardware event (which may in turn trigger actions in other hardware modules) or an asynchronous software interrupt.

Note: The connection of events between modules requires the use of the SAM Event System Driver (EVENTS) to route output event of one module to the input event of another. For more information on event routing, refer to the event driver documentation.

3.5. Physical Connection

Figure 3-1 Physical Connection on page 7 shows how this module is interconnected within the device.

Figure 3-1. Physical Connection



4. Special Considerations

Not all devices support disabling of the NMI channel(s) detection mode - see your device datasheet.

5. Extra Information

For extra information, see Extra Information for EXTINT Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for EXTINT Driver.

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type extint_callback_t

```
typedef void(* extint_callback_t )(void)
```

Type definition for an EXTINT module callback function

7.2. Structure Definitions

7.2.1. Struct extint_chan_conf

Configuration structure for the edge detection mode of an external interrupt channel.

Table 7-1. Members

Type	Name	Description
enum extint_detect	detection_criteria	Edge detection mode to use
bool	filter_input_signal	Filter the raw input signal to prevent noise from triggering an interrupt accidentally, using a three sample majority filter
uint32_t	gpio_pin	GPIO pin the NMI should be connected to
uint32_t	gpio_pin_mux	MUX position the GPIO pin should be configured to
enum extint_pull	gpio_pin_pull	Internal pull to enable on the input pin
bool	wake_if_sleeping	Wake up the device if the channel interrupt fires during sleep mode

7.2.2. Struct extint_events

Event flags for the extint_enable_events() and extint_disable_events().

Table 7-2. Members

Type	Name	Description
bool	generate_event_on_detect[]	If `true`, an event will be generated when an external interrupt channel detection state changes

7.2.3. Struct extint_nmi_conf

Configuration structure for the edge detection mode of an external interrupt NMI channel.

Table 7-3. Members

Type	Name	Description
enum extint_detect	detection_criteria	Edge detection mode to use. Not all devices support all possible detection modes for NMIs.
bool	filter_input_signal	Filter the raw input signal to prevent noise from triggering an interrupt accidentally, using a three sample majority filter
uint32_t	gpio_pin	GPIO pin the NMI should be connected to
uint32_t	gpio_pin_mux	MUX position the GPIO pin should be configured to
enum extint_pull	gpio_pin_pull	Internal pull to enable on the input pin

7.3. Macro Definitions

7.3.1. Macro EIC_NUMBER_OF_INTERRUPTS

```
#define EIC_NUMBER_OF_INTERRUPTS
```

7.3.2. Macro EXTINT_CLK_GCLK

```
#define EXTINT_CLK_GCLK
```

The EIC is clocked by GCLK_EIC.

7.3.3. Macro EXTINT_CLK_ULP32K

```
#define EXTINT_CLK_ULP32K
```

The EIC is clocked by CLK_ULP32K.

7.4. Function Definitions

7.4.1. Event Management

7.4.1.1. Function extint_enable_events()

Enables an External Interrupt event output.

```
void extint_enable_events(
        struct extint_events *const events)
```

Enables one or more output events from the External Interrupt module. See here for a list of events this module supports.

Note: Events cannot be altered while the module is enabled.

Table 7-4. Parameters

Data direction	Parameter name	Description
[in]	events	Struct containing flags of events to enable

7.4.1.2. Function extint_disable_events()

Disables an External Interrupt event output.

```
void extint_disable_events(
        struct extint_events *const events)
```

Disables one or more output events from the External Interrupt module. See here for a list of events this module supports.

Note: Events cannot be altered while the module is enabled.

Table 7-5. Parameters

Data direction	Parameter name	Description
[in]	events	Struct containing flags of events to disable

7.4.2. Configuration and Initialization (Channel)

7.4.2.1. Function extint_chan_get_config_defaults()

Initializes an External Interrupt channel configuration structure to defaults.

```
void extint_chan_get_config_defaults(
        struct extint_chan_conf *const config)
```

Initializes a given External Interrupt channel configuration structure to a set of known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- Wake the device if an edge detection occurs whilst in sleep
- Input filtering disabled
- Internal pull-up enabled
- Detect falling edges of a signal

Table 7-6. Parameters

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

7.4.2.2. Function extint_chan_set_config()

Writes an External Interrupt channel configuration to the hardware module.

```
void extint_chan_set_config(
        const uint8_t channel,
        const struct extint_chan_conf *const config)
```

Writes out a given configuration of an External Interrupt channel configuration to the hardware module. If the channel is already configured, the new configuration will replace the existing one.

Table 7-7. Parameters

Data direction	Parameter name	Description
[in]	channel	External Interrupt channel to configure
[in]	config	Configuration settings for the channel

7.4.3. Configuration and Initialization (NMI)

7.4.3.1. Function extint_nmi_get_config_defaults()

Initializes an External Interrupt NMI channel configuration structure to defaults.

```
void extint_nmi_get_config_defaults(
        struct extint_nmi_conf *const config)
```

Initializes a given External Interrupt NMI channel configuration structure to a set of known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- Input filtering disabled
- Detect falling edges of a signal
- Asynchronous edge detection is disabled

Table 7-8. Parameters

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

7.4.3.2. Function extint_nmi_set_config()

Writes an External Interrupt NMI channel configuration to the hardware module.

```
enum status_code extint_nmi_set_config(
        const uint8_t nmi_channel,
        const struct extint_nmi_conf *const config)
```

Writes out a given configuration of an External Interrupt NMI channel configuration to the hardware module. If the channel is already configured, the new configuration will replace the existing one.

Table 7-9. Parameters

Data direction	Parameter name	Description
[in]	nmi_channel	External Interrupt NMI channel to configure
[in]	config	Configuration settings for the channel

Returns

Status code indicating the success or failure of the request.

Table 7-10. Return Values

Return value	Description
STATUS_OK	Configuration succeeded
STATUS_ERR_PIN_MUX_INVALID	An invalid pinmux value was supplied
STATUS_ERR_BAD_FORMAT	An invalid detection mode was requested

7.4.4. Detection testing and clearing (channel)

7.4.4.1. Function extint_chan_is_detected()

Retrieves the edge detection state of a configured channel.

```
bool extint_chan_is_detected(
        const uint8_t channel)
```

Reads the current state of a configured channel, and determines if the detection criteria of the channel has been met.

Table 7-11. Parameters

Data direction	Parameter name	Description
[in]	channel	External Interrupt channel index to check

Returns

Status of the requested channel's edge detection state.

Table 7-12. Return Values

Return value	Description
true	If the channel's edge/level detection criteria was met
false	If the channel has not detected its configured criteria

7.4.4.2. Function extint_chan_clear_detected()

Clears the edge detection state of a configured channel.

```
void extint_chan_clear_detected(
        const uint8_t channel)
```

Clears the current state of a configured channel, readying it for the next level or edge detection.

Table 7-13. Parameters

Data direction	Parameter name	Description
[in]	channel	External Interrupt channel index to check

7.4.5. Detection Testing and Clearing (NMI)

7.4.5.1. Function extint_nmi_is_detected()

Retrieves the edge detection state of a configured NMI channel.

```
bool extint_nmi_is_detected(
        const uint8_t nmi_channel)
```

Reads the current state of a configured NMI channel, and determines if the detection criteria of the NMI channel has been met.

Table 7-14. Parameters

Data direction	Parameter name	Description
[in]	nmi_channel	External Interrupt NMI channel index to check

Returns

Status of the requested NMI channel's edge detection state.

Table 7-15. Return Values

Return value	Description
true	If the NMI channel's edge/level detection criteria was met
false	If the NMI channel has not detected its configured criteria

7.4.5.2. Function extint_nmi_clear_detected()

Clears the edge detection state of a configured NMI channel.

```
void extint_nmi_clear_detected(
        const uint8_t nmi_channel)
```

Clears the current state of a configured NMI channel, readying it for the next level or edge detection.

Table 7-16. Parameters

Data direction	Parameter name	Description
[in]	nmi_channel	External Interrupt NMI channel index to check

7.4.6. Callback Configuration and Initialization

7.4.6.1. Function extint_register_callback()

Registers an asynchronous callback function with the driver.

```
enum status_code extint_register_callback(
        const extint_callback_t callback,
        const uint8_t channel,
        const enum extint_callback_type type)
```

Registers an asynchronous callback with the EXTINT driver, fired when a channel detects the configured channel detection criteria (e.g. edge or level). Callbacks are fired once for each detected channel.

Note: NMI channel callbacks cannot be registered via this function; the device's NMI interrupt should be hooked directly in the user application and the NMI flags manually cleared via extint_nmi_clear_detected().

Table 7-17. Parameters

Data direction	Parameter name	Description
[in]	callback	Pointer to the callback function to register
[in]	channel	Logical channel to register callback for
[in]	type	Type of callback function to register

Returns
Status of the registration operation.

Table 7-18. Return Values

Return value	Description
STATUS_OK	The callback was registered successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied
STATUS_ERR_ALREADY_INITIALIZED	Callback function has been registered, need unregister first

7.4.6.2. Function extint_unregister_callback()

Unregisters an asynchronous callback function with the driver.

```
enum status_code extint_unregister_callback(
        const extint_callback_t callback,
        const uint8_t channel,
        const enum extint_callback_type type)
```

Unregisters an asynchronous callback with the EXTINT driver, removing it from the internal callback registration table.

Table 7-19. Parameters

Data direction	Parameter name	Description
[in]	callback	Pointer to the callback function to unregister
[in]	channel	Logical channel to unregister callback for
[in]	type	Type of callback function to unregister

Returns
Status of the de-registration operation.

Table 7-20. Return Values

Return value	Description
STATUS_OK	The callback was unregistered successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied
STATUS_ERR_BAD_ADDRESS	No matching entry was found in the registration table

7.4.6.3. Function extint_get_current_channel()

Find what channel caused the callback.

```
uint8_t extint_get_current_channel( void )
```

Can be used in an EXTINT callback function to find what channel caused the callback in case the same callback is used by multiple channels.

Returns
Channel number.

7.4.7. Callback Enabling and Disabling (Channel)

7.4.7.1. Function extint_chan_enable_callback()

Enables asynchronous callback generation for a given channel and type.

```
enum status_code extint_chan_enable_callback(
        const uint8_t channel,
        const enum extint_callback_type type)
```

Enables asynchronous callbacks for a given logical external interrupt channel and type. This must be called before an external interrupt channel will generate callback events.

Table 7-21. Parameters

Data direction	Parameter name	Description
[in]	channel	Logical channel to enable callback generation for
[in]	type	Type of callback function callbacks to enable

Returns
Status of the callback enable operation.

Table 7-22. Return Values

Return value	Description
STATUS_OK	The callback was enabled successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.4.7.2. Function extint_chan_disable_callback()

Disables asynchronous callback generation for a given channel and type.

```
enum status_code extint_chan_disable_callback(
        const uint8_t channel,
        const enum extint_callback_type type)
```

Disables asynchronous callbacks for a given logical external interrupt channel and type.

Table 7-23. Parameters

Data direction	Parameter name	Description
[in]	channel	Logical channel to disable callback generation for
[in]	type	Type of callback function callbacks to disable

Returns
Status of the callback disable operation.

Table 7-24. Return Values

Return value	Description
STATUS_OK	The callback was disabled successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.5. Enumeration Definitions

7.5.1. Callback Configuration and Initialization

7.5.1.1. Enum extint_callback_type

Enum for the possible callback types for the EXTINT module.

Table 7-25. Members

Enum value	Description
EXTINT_CALLBACK_TYPE_DETECT	Callback type for when an external interrupt detects the configured channel criteria (i.e. edge or level detection)

7.5.2. Enum extint_detect

Enum for the possible signal edge detection modes of the External Interrupt Controller module.

Table 7-26. Members

Enum value	Description
EXTINT_DETECT_NONE	No edge detection. Not allowed as a NMI detection mode on some devices.
EXTINT_DETECT_RISING	Detect rising signal edges

Enum value	Description
EXTINT_DETECT_FALLING	Detect falling signal edges
EXTINT_DETECT_BOTH	Detect both signal edges
EXTINT_DETECT_HIGH	Detect high signal levels
EXTINT_DETECT_LOW	Detect low signal levels

7.5.3. Enum extint_pull

Enum for the possible pin internal pull configurations.

Note: Disabling the internal pull resistor is not recommended if the driver is used in interrupt (callback) mode, due the possibility of floating inputs generating continuous interrupts.

Table 7-27. Members

Enum value	Description
EXTINT_PULL_UP	Internal pull-up resistor is enabled on the pin
EXTINT_PULL_DOWN	Internal pull-down resistor is enabled on the pin
EXTINT_PULL_NONE	Internal pull resistor is disconnected from the pin

8. Extra Information for EXTINT Driver

8.1. Acronyms

The table below presents the acronyms used in this module:

Acronym	Description
EIC	External Interrupt Controller
MUX	Multiplexer
NMI	Non-Maskable Interrupt

8.2. Dependencies

This driver has the following dependencies:

- System Pin Multiplexer Driver

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
• Driver updated to follow driver type convention • Removed `extint_reset`(), `extint_disable`() and `extint_enable()` functions. Added internal function `_system_extint_init`(). • Added configuration EXTINT_CLOCK_SOURCE in conf_extint.h • Removed configuration EXTINT_CALLBACKS_MAX in conf_extint.h, and added channel parameter in the register functions `extint_register_callback`() and `extint_unregister_callback`()
Updated interrupt handler to clear interrupt flag before calling callback function
Updated initialization function to also enable the digital interface clock to the module if it is disabled
Initial Release

9. Examples for EXTINT Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM External Interrupt (EXTINT) 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 EXTINT - Basic
- Quick Start Guide for EXTINT - Callback

9.1. Quick Start Guide for EXTINT - Basic

The supported board list:

- SAM D20 Xplained Pro
- SAM D21 Xplained Pro
- SAM R21 Xplained Pro
- SAM L21 Xplained Pro
- SAM L22 Xplained Pro
- SAM DA1 Xplained Pro
- SAM C21 Xplained Pro

In this use case, the EXTINT module is configured for:

- External interrupt channel connected to the board LED is used
- External interrupt channel is configured to detect both input signal edges

This use case configures a physical I/O pin of the device so that it is routed to a logical External Interrupt Controller channel to detect rising and falling edges of the incoming signal.

When the board button is pressed, the board LED will light up. When the board button is released, the LED will turn off.

9.1.1. Setup

9.1.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.1.1.2. Code

Copy-paste the following setup code to your user application:

```
void configure_extint_channel(void)
{
    struct extint_chan_conf config_extint_chan;
    extint_chan_get_config_defaults(&config_extint_chan);

    config_extint_chan.gpio_pin           = BUTTON_0_EIC_PIN;
    config_extint_chan.gpio_pin_mux       = BUTTON_0_EIC_MUX;
    config_extint_chan.gpio_pin_pull      = EXTINT_PULL_UP;
    config_extint_chan.detection_criteria = EXTINT_DETECT_BOTH;
    extint_chan_set_config(BUTTON_0_EIC_LINE, &config_extint_chan);
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_extint_channel();
```

9.1.1.3. Workflow

1. Create an EXTINT module channel configuration struct, which can be filled out to adjust the configuration of a single external interrupt channel.

   ```
   struct extint_chan_conf config_extint_chan;
   ```

2. Initialize the channel configuration struct with the module's default values.

   ```
   extint_chan_get_config_defaults(&config_extint_chan);
   ```

   Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Adjust the configuration struct to configure the pin MUX (to route the desired physical pin to the logical channel) to the board button, and to configure the channel to detect both rising and falling edges.

   ```
   config_extint_chan.gpio_pin           = BUTTON_0_EIC_PIN;
   config_extint_chan.gpio_pin_mux       = BUTTON_0_EIC_MUX;
   config_extint_chan.gpio_pin_pull      = EXTINT_PULL_UP;
   config_extint_chan.detection_criteria = EXTINT_DETECT_BOTH;
   ```

4. Configure external interrupt channel with the desired channel settings.

   ```
   extint_chan_set_config(BUTTON_0_EIC_LINE, &config_extint_chan);
   ```

9.1.2. Use Case

9.1.2.1. Code

Copy-paste the following code to your user application:

```
while (true) {
    if (extint_chan_is_detected(BUTTON_0_EIC_LINE)) {

        // Do something in response to EXTINT edge detection
        bool button_pin_state = port_pin_get_input_level(BUTTON_0_PIN);
        port_pin_set_output_level(LED_0_PIN, button_pin_state);

        extint_chan_clear_detected(BUTTON_0_EIC_LINE);
    }
}
```

9.1.2.2. Workflow

1. Read in the current external interrupt channel state to see if an edge has been detected.

   ```
   if (extint_chan_is_detected(BUTTON_0_EIC_LINE)) {
   ```

2. Read in the new physical button state and mirror it on the board LED.

   ```
   // Do something in response to EXTINT edge detection
   bool button_pin_state = port_pin_get_input_level(BUTTON_0_PIN);
   port_pin_set_output_level(LED_0_PIN, button_pin_state);
   ```

3. Clear the detection state of the external interrupt channel so that it is ready to detect a future falling edge.

   ```
   extint_chan_clear_detected(BUTTON_0_EIC_LINE);
   ```

9.2. Quick Start Guide for EXTINT - Callback

The supported board list:

- SAM D20 Xplained Pro
- SAM D21 Xplained Pro
- SAM R21 Xplained Pro
- SAM L21 Xplained Pro
- SAM L22 Xplained Pro
- SAM DA1 Xplained Pro
- SAM C21 Xplained Pro

In this use case, the EXTINT module is configured for:

- External interrupt channel connected to the board LED is used
- External interrupt channel is configured to detect both input signal edges
- Callbacks are used to handle detections from the External Interrupt

This use case configures a physical I/O pin of the device so that it is routed to a logical External Interrupt Controller channel to detect rising and falling edges of the incoming signal. A callback function is used to handle detection events from the External Interrupt module asynchronously.

When the board button is pressed, the board LED will light up. When the board button is released, the LED will turn off.

9.2.1. Setup

9.2.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.2.1.2. Code

Copy-paste the following setup code to your user application:

```
void configure_extint_channel(void)
{
    struct extint_chan_conf config_extint_chan;
    extint_chan_get_config_defaults(&config_extint_chan);

    config_extint_chan.gpio_pin           = BUTTON_0_EIC_PIN;
    config_extint_chan.gpio_pin_mux       = BUTTON_0_EIC_MUX;
    config_extint_chan.gpio_pin_pull      = EXTINT_PULL_UP;
    config_extint_chan.detection_criteria = EXTINT_DETECT_BOTH;
    extint_chan_set_config(BUTTON_0_EIC_LINE, &config_extint_chan);
}

void configure_extint_callbacks(void)
{
    extint_register_callback(extint_detection_callback,
            BUTTON_0_EIC_LINE,
            EXTINT_CALLBACK_TYPE_DETECT);
    extint_chan_enable_callback(BUTTON_0_EIC_LINE,
            EXTINT_CALLBACK_TYPE_DETECT);
}

void extint_detection_callback(void)
{
    bool pin_state = port_pin_get_input_level(BUTTON_0_PIN);
```

```
    port_pin_set_output_level(LED_0_PIN, pin_state);
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_extint_channel();
configure_extint_callbacks();

system_interrupt_enable_global();
```

9.2.1.3. Workflow

1. Create an EXTINT module channel configuration struct, which can be filled out to adjust the configuration of a single external interrupt channel.

   ```
   struct extint_chan_conf config_extint_chan;
   ```

2. Initialize the channel configuration struct with the module's default values.

   ```
   extint_chan_get_config_defaults(&config_extint_chan);
   ```

   Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Adjust the configuration struct to configure the pin MUX (to route the desired physical pin to the logical channel) to the board button, and to configure the channel to detect both rising and falling edges.

   ```
   config_extint_chan.gpio_pin           = BUTTON_0_EIC_PIN;
   config_extint_chan.gpio_pin_mux       = BUTTON_0_EIC_MUX;
   config_extint_chan.gpio_pin_pull      = EXTINT_PULL_UP;
   config_extint_chan.detection_criteria = EXTINT_DETECT_BOTH;
   ```

4. Configure external interrupt channel with the desired channel settings.

   ```
   extint_chan_set_config(BUTTON_0_EIC_LINE, &config_extint_chan);
   ```

5. Register a callback function `extint_handler()` to handle detections from the External Interrupt controller.

   ```
   extint_register_callback(extint_detection_callback,
           BUTTON_0_EIC_LINE,
           EXTINT_CALLBACK_TYPE_DETECT);
   ```

6. Enable the registered callback function for the configured External Interrupt channel, so that it will be called by the module when the channel detects an edge.

   ```
   extint_chan_enable_callback(BUTTON_0_EIC_LINE,
           EXTINT_CALLBACK_TYPE_DETECT);
   ```

7. Define the EXTINT callback that will be fired when a detection event occurs. For this example, a LED will mirror the new button state on each detection edge.

   ```
   void extint_detection_callback(void)
   {
       bool pin_state = port_pin_get_input_level(BUTTON_0_PIN);
       port_pin_set_output_level(LED_0_PIN, pin_state);
   }
   ```

9.2.2. Use Case

9.2.2.1. Code

Copy-paste the following code to your user application:

```
while (true) {
    /* Do nothing - EXTINT will fire callback asynchronously */
}
```

9.2.2.2. Workflow

1. External interrupt events from the driver are detected asynchronously; no special application `main()` code is required.

10. Document Revision History

Doc. Rev.	Date	Comments
42112E	12/2015	Added support for SAM L21/L22, SAM C21, SAM D09, and SAM DA1
42112D	12/2014	Added support for SAM R21 and SAM D10/D11
42112C	01/2014	Added support for SAM D21
42112B	06/2013	Added additional documentation on the event system. Corrected documentation typos.
42112A	06/2013	Initial release

Atmel | Enabling Unlimited Possibilities®

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