AT10732: SAM L True Random Number Generator (TRNG) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's True Random Number Generator functionality.

The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripheral is used by this module:

- TRNG (True Random Number Generator)

The following devices can use this module:

- Atmel | SMART SAM L21/L22

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

This driver provides an interface for the TRNG functions on the device.

As soon as the TRNG is enabled, the module provides a new 32-bit random data, for every 84 CLK_TRNG_APB clock cycles.

4. Special Considerations

There are no special considerations for this module.

5. Extra Information

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

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

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

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type trng_callback_t

```
typedef void(* trng_callback_t )(struct trng_module *const module_inst)
```

Type definition for a TRNG module callback function.

7.1.2. Variable _trng_instance

```
struct trng_module * _trng_instance
```

7.2. Structure Definitions

7.2.1. Struct trng_config

Configuration structure for a True Random Number Generator.

Table 7-1. Members

Type	Name	Description
bool	run_in_standby	If `true`, the True Random Number Generator will not be stopped in standby sleep mode

7.2.2. Struct trng_events

Event flags for the TRNG module. This is used to enable and disable events via trng_enable_events() and trng_disable_events().

Table 7-2. Members

Type	Name	Description
bool	generate_event_on_data_ready	Enable event generation on random data ready

7.2.3. Struct trng_module

TRNG software instance structure, used to retain software state information of an associated hardware module instance.

Note: The fields of this structure should not be altered by the user application; they are reserved for module-internal use only.

7.3. Function Definitions

7.3.1. Driver Initialization and Configuration

7.3.1.1. Function trng_init()

Initializes a hardware TRNG module instance.

```
enum status_code trng_init(
        struct trng_module *const module_inst,
        Trng *const hw,
        struct trng_config *const config)
```

Enables the clock and initializes the TRNG module, based on the given configuration values.

Table 7-3. Parameters

Data direction	Parameter name	Description
[in, out]	module_inst	Pointer to the software module instance struct
[in]	hw	Pointer to the TRNG hardware module
[in]	config	Pointer to the TRNG configuration options struct

Returns
Status of the initialization procedure.

Table 7-4. Return Values

Return value	Description
STATUS_OK	The module was initialized successfully

7.3.1.2. Function trng_get_config_defaults()

Initializes all members of a TRNG configuration structure to safe defaults.

```
void trng_get_config_defaults(
        struct trng_config *const config)
```

Initializes all members of a given True Random Number Generator configuration structure to safe 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:

- True Random Number Generator will not be stopped in standby sleep mode

Table 7-5. Parameters

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

7.3.1.3. Function trng_enable()

Enables a TRNG that was previously configured.

```
void trng_enable(
        struct trng_module *const module_inst)
```

Enables True Random Number Generator that was previously configured via a call to trng_init().

Table 7-6. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the True Random Number Generator peripheral

7.3.1.4. Function trng_disable()

Disables a TRNG that was previously enabled.

```
void trng_disable(
        struct trng_module *const module_inst)
```

Disables True Random Number Generator that was previously started via a call to trng_enable().

Table 7-7. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the True Random Number Generator peripheral

7.3.1.5. Function trng_enable_events()

Enables a TRNG event output.

```
void trng_enable_events(
        struct trng_module *const module_inst,
        struct trng_events *const events)
```

Enables output events from the True Random Number Generator module. See Section Struct trng_events for a list of events this module supports.

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

Table 7-8. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the TRNG peripheral
[in]	events	Struct containing flags of events to enable

7.3.1.6. Function trng_disable_events()

Disables a TRNG event output.

```
void trng_disable_events(
        struct trng_module *const module_inst,
        struct trng_events *const events)
```

Disables output events from the True Random Number Generator module. See Section Struct trng_events for a list of events this module supports.

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

Table 7-9. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the TRNG peripheral
[in]	events	Struct containing flags of events to disable

7.3.2. Read TRNG Result

7.3.2.1. Function trng_read()

Read the random data result.

```
enum status_code trng_read(
        struct trng_module *const module_inst,
        uint32_t * result)
```

Reads the random data result.

Table 7-10. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the TRNG software instance struct
[out]	result	Pointer to store the result value in

Returns
Status of the TRNG read request.

Table 7-11. Return Values

Return value	Description
STATUS_OK	The result was retrieved successfully
STATUS_BUSY	A random result was not ready

7.3.3. Callback Management

7.3.3.1. Function trng_register_callback()

Registers a callback.

```
enum status_code trng_register_callback(
        struct trng_module *const module,
        trng_callback_t callback_func,
        enum trng_callback callback_type)
```

Registers a callback function which is implemented by the user.

Note: The callback must be enabled by trng_enable_callback, in order for the interrupt handler to call it when the conditions for the callback type is met.

Table 7-12. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to TC software instance struct
[in]	callback_func	Pointer to callback function
[in]	callback_type	Callback type given by an enum

Table 7-13. Return Values

Return value	Description
STATUS_OK	The function exited successfully

7.3.3.2. Function trng_unregister_callback()

Unregisters a callback.

```
enum status_code trng_unregister_callback(
        struct trng_module * module,
        enum trng_callback callback_type)
```

Unregisters a callback function implemented by the user. The callback should be disabled before it is unregistered.

Table 7-14. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to TC software instance struct
[in]	callback_type	Callback type given by an enum

Table 7-15. Return Values

Return value	Description
STATUS_OK	The function exited successfully

7.3.3.3. Function trng_enable_callback()

Enables callback.

```
void trng_enable_callback(
        struct trng_module *const module,
        enum trng_callback callback_type)
```

Enables the callback function registered by trng_register_callback. The callback function will be called from the interrupt handler when the conditions for the callback type are met.

Table 7-16. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to TRNG software instance struct
[in]	callback_type	Callback type given by an enum

7.3.3.4. Function trng_disable_callback()

Disables callback.

```
void trng_disable_callback(
        struct trng_module *const module,
        enum trng_callback callback_type)
```

Disables the callback function registered by the trng_register_callback.

Table 7-17. Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to TRNG software instance struct
[in]	callback_type	Callback type given by an enum

7.3.4. Job Management

7.3.4.1. Function trng_read_buffer_job()

Read multiple random data from TRNG.

```
enum status_code trng_read_buffer_job(
        struct trng_module *const module_inst,
        uint32_t * buffer,
        uint32_t number)
```

As soon as the TRNG is enabled, the module provides a new 32-bits random data for every 84 CLK_TRNG_APB clock cycles.

Table 7-18. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the TRNG software instance struct
[in]	number	Number of random data to get
[out]	buffer	Buffer to store the random data

Returns
Status of the job start.

Table 7-19. Return Values

Return value	Description
STATUS_OK	The read job was started successfully and is in progress
STATUS_BUSY	The TRNG is already busy with another job

7.3.4.2. Function trng_get_job_status()

Gets the status of a job.

```
enum status_code trng_get_job_status(
        struct trng_module * module_inst,
        enum trng_job_type type)
```

Gets the status of an ongoing or the last job.

Table 7-20. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the TRNG software instance struct
[in]	type	Type of job to abort

Returns
Status of the job.

7.3.4.3. Function trng_abort_job()

Aborts an ongoing job.

```
void trng_abort_job(
        struct trng_module * module_inst,
        enum trng_job_type type)
```

Table 7-21. Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the TRNG software instance struct
[in]	type	Type of job to abort

7.4. Enumeration Definitions

7.4.1. Enum trng_callback

Enum for possible callback types for the TRNG module.

Table 7-22. Members

Enum value	Description
TRNG_CALLBACK_READ_BUFFER	Callback for specific number of random data ready

7.4.2. Enum trng_job_type

Enum for the possible types of TRNG asynchronous jobs that may be issued to the driver.

Table 7-23. Members

Enum value	Description
TRNG_JOB_READ_BUFFER	Asynchronous TRNG read into a user provided buffer

8. Extra Information for TRNG Driver

8.1. Acronyms

Acronym	Description
TRNG	True Random Number Generator

8.2. Dependencies

This driver has no dependencies.

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 Release

9. Examples for TRNG Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM True Random Number Generator (TRNG) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that QSGs can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for TRNG - Basic
- Quick Start Guide for TRNG - Callback

9.1. Quick Start Guide for TRNG - Basic

In this use case, the True Random Number Generator (TRNG) module is configured for:

- The TRNG peripheral will not be stopped in standby sleep mode

This use case will read random data in polling mode repeatedly. After reading a data, the board LED will be toggled.

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:

```
/* TRNG module software instance (must not go out of scope while in use) */
static struct trng_module trng_instance;

void configure_trng(void)
{
    /* Create a new configuration structure for the TRNG settings
     * and fill with the default module settings. */
    struct trng_config config_trng;
    trng_get_config_defaults(&config_trng);

    /* Alter any TRNG configuration settings here if required */

    /* Initialize TRNG with the user settings */
    trng_init(&trng_instance, TRNG, &config_trng);
}
```

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

```
system_init();
configure_trng();
trng_enable(&trng_instance);
```

9.1.1.3. Workflow

1. Create a TRNG device instance struct, which will be associated with a TRNG peripheral hardware instance.

   ```
   static struct trng_module trng_instance;
   ```

   Note: Device instance structures shall **never** go out of scope when in use.

2. Create a new function `configure_trng()`, which will be used to configure the overall TRNG peripheral.

```
void configure_trng(void)
```

3. Create a TRNG peripheral configuration structure that will be filled out to set the module configuration.

```
struct trng_config config_trng;
```

4. Fill the TRNG peripheral configuration structure with the default module configuration values.

```
trng_get_config_defaults(&config_trng);
```

5. Initialize the TRNG peripheral and associate it with the software instance structure that was defined previously.

```
trng_init(&trng_instance, TRNG, &config_trng);
```

6. Enable the now initialized TRNG peripheral.

```
trng_enable(&trng_instance);
```

9.1.2. Implementation

9.1.2.1. Code

Copy-paste the following code to your user application:

```
uint32_t random_result;

while (true) {
    while (trng_read(&trng_instance, &random_result) != STATUS_OK) {
    }
    port_pin_toggle_output_level(LED_0_PIN);
    /* Add a short delay to see LED toggle */
    volatile uint32_t delay = 50000;
    while(delay--) {
    }
}
```

9.1.2.2. Workflow

1. Make the application loop infinitely.

```
while (true) {
```

2. Start to read a random data from TRNG until success.

```
while (trng_read(&trng_instance, &random_result) != STATUS_OK) {
}
```

3. Toggle the board LED to indicate a random data is read.

```
port_pin_toggle_output_level(LED_0_PIN);
/* Add a short delay to see LED toggle */
volatile uint32_t delay = 50000;
while(delay--) {
}
```

9.2. Quick Start Guide for TRNG - Callback

In this use case, the True Random Number Generator (TRNG) module is configured for:

- The TRNG peripheral will not be stopped in standby sleep mode

This use case will read random data in interrupt mode repeatly. After reading specific size of buffer data, the board LED will be toggled.

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:

```
bool volatile trng_read_done = false;

void configure_trng(void);
void configure_trng_callback(void);
void trng_complete_callback(struct trng_module *const module_inst);

/* TRNG module software instance (must not go out of scope while in use) */
static struct trng_module trng_instance;

void configure_trng(void)
{
    /* Create a new configuration structure for the TRNG settings
     * and fill with the default module settings. */
    struct trng_config config_trng;
    trng_get_config_defaults(&config_trng);

    /* Alter any TRNG configuration settings here if required */

    /* Initialize TRNG with the user settings */
    trng_init(&trng_instance, TRNG, &config_trng);
}

void trng_complete_callback(struct trng_module *const module_inst)
{
    trng_read_done = true;
}

void configure_trng_callback(void)
{
    trng_register_callback(&trng_instance, trng_complete_callback,
            TRNG_CALLBACK_READ_BUFFER);
    trng_enable_callback(&trng_instance, TRNG_CALLBACK_READ_BUFFER);
}
```

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

```
system_init();
configure_trng();
configure_trng_callback();

trng_enable(&trng_instance);
```

9.2.1.3. Workflow

1. Create a TRNG device instance struct, which will be associated with a TRNG peripheral hardware instance.

   ```
   static struct trng_module trng_instance;
   ```

Note: Device instance structures shall **never** go out of scope when in use.

2. Create a new function `configure_trng()`, which will be used to configure the overall TRNG peripheral.

```
void configure_trng(void)
```

3. Create a TRNG peripheral configuration structure that will be filled out to set the module configuration.

```
struct trng_config config_trng;
```

4. Fill the TRNG peripheral configuration structure with the default module configuration values.

```
trng_get_config_defaults(&config_trng);
```

5. Initialize the TRNG peripheral and associate it with the software instance structure that was defined previously.

```
trng_init(&trng_instance, TRNG, &config_trng);
```

6. Create a new callback function.

```
void trng_complete_callback(struct trng_module *const module_inst)
{
    trng_read_done = true;
}
```

7. Create a callback status software flag.

```
bool volatile trng_read_done = false;
```

8. Let the callback function set the flag to true when read job done.

```
trng_read_done = true;
```

9. Create a new function `configure_trng_callback()`, which will be used to configure the callbacks.

```
void configure_trng_callback(void)
{
    trng_register_callback(&trng_instance, trng_complete_callback,
            TRNG_CALLBACK_READ_BUFFER);
    trng_enable_callback(&trng_instance, TRNG_CALLBACK_READ_BUFFER);
}
```

10. Register callback function.

```
trng_register_callback(&trng_instance, trng_complete_callback,
        TRNG_CALLBACK_READ_BUFFER);
```

11. Enable the callbacks.

```
trng_enable_callback(&trng_instance, TRNG_CALLBACK_READ_BUFFER);
```

12. Enable the now initialized TRNG peripheral.

```
trng_enable(&trng_instance);
```

Note: This should not be done until after the TRNG is set up and ready to be used.

9.2.2. Implementation

9.2.2.1. Code

Copy-paste the following code to your user application:

```
uint32_t random_buffer[5];

while (true) {
    trng_read_buffer_job(&trng_instance, random_buffer, 5);
    while (!trng_read_done) {
    }
    trng_read_done = false;
    port_pin_toggle_output_level(LED_0_PIN);
    /* Add a short delay to see LED toggle */
    volatile uint32_t delay = 50000;
    while(delay--) {
    }
}
```

9.2.2.2. Workflow

1. Make the application loop infinitely.

   ```
   while (true) {
   ```

2. Start an asynchronous TRNG read job, to store random data into the global buffer and generate a callback when complete.

   ```
   trng_read_buffer_job(&trng_instance, random_buffer, 5);
   ```

3. Wait until the asynchronous read job is complete.

   ```
   while (!trng_read_done) {
   }
   trng_read_done = false;
   ```

4. Toggle the board LED to indicate specific size of random data were read.

   ```
   port_pin_toggle_output_level(LED_0_PIN);
   /* Add a short delay to see LED toggle */
   volatile uint32_t delay = 50000;
   while(delay--) {
   }
   ```

10. Document Revision History

Doc. Rev.	Date	Comments
42444B	01/2016	Added support for SAM L22
42444A	06/2015	Initial document 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**