SMART ARM-based Microcontrollers

AT13481: SAM Frequency Meter (FREQM) Driver

APPLICATION NOTE

Introduction

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

The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripheral is used by this module:

- FREQM (Frequency Meter)

The following devices can use this module:

- Atmel | SMART SAM L22
- Atmel | SMART SAM C20
- Atmel | SMART SAM C21

The outline of this documentation is as follows:

- Prerequisites
- Module Overview
- Special Considerations
- Extra Information
- Examples
- API Overview

Table of Contents

Introduction......1

1. Software License......4

2. Prerequisites......5

3. Module Overview......6

4. Special Considerations......7

5. Extra Information......8

6. Examples......9

7. API Overview......10

- 7.1. Variable and Type Definitions......10
  - 7.1.1. Type freqm_callback_t......10
  - 7.1.2. Variable _freqm_instance......10
- 7.2. Structure Definitions......10
  - 7.2.1. Struct freqm_config......10
  - 7.2.2. Struct freqm_module......10
- 7.3. Function Definitions......10
  - 7.3.1. Driver Initialization and Configuration......10
  - 7.3.2. Read FREQM Result......12
  - 7.3.3. Callback Configuration and Initialization......13
  - 7.3.4. Callback Enabling and Disabling......14
  - 7.3.5. Function freqm_is_syncing()......15
- 7.4. Enumeration Definitions......15
  - 7.4.1. Enum freqm_callback......15
  - 7.4.2. Enum freqm_callback_type......16
  - 7.4.3. Enum freqm_status......16

8. Extra Information for FREQM Driver......17

- 8.1. Acronyms......17
- 8.2. Dependencies......17
- 8.3. Errata......17
- 8.4. Module History......17

9. Examples for FREQM Driver......18

- 9.1. Quick Start Guide for FREQM - Basic......18
  - 9.1.1. Setup......18
  - 9.1.2. Implementation......19
- 9.2. Quick Start Guide for FREQM - Callback......20
  - 9.2.1. Setup......20
  - 9.2.2. Implementation......22

10. Document Revision History.. 24

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The module accurately measures the frequency of a clock by comparing it to a known reference clock as soon as the FREQM is enabled. Two generic clocks are used by the FREQM. The frequency of the measured clock is:

$$f_{CLK\_MSR} = \frac{VALUE}{REFNUM} \times f_{CLK\_REF}$$

Ratio can be measured with 24-bit accuracy.

The FREQM has one interrupt source, which generates when a frequency measurement is done. It can be used to wake up the device from sleep modes.

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

4. Special Considerations

There are no special considerations for this module.

5. Extra Information

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

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

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

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type freqm_callback_t

```
typedef void(* freqm_callback_t )(void)
```

Type definition for a FREQM module callback function.

7.1.2. Variable _freqm_instance

```
struct freqm_module * _freqm_instance
```

7.2. Structure Definitions

7.2.1. Struct freqm_config

Configuration structure for a Frequency Meter.

Table 7-1 Members

Type	Name	Description
enum gclk_generator	msr_clock_source	GCLK source select for measurement
uint16_t	ref_clock_circles	Measurement duration in number of reference clock cycles. Range 1~255
enum gclk_generator	ref_clock_source	GCLK source select for reference

7.2.2. Struct freqm_module

FREQM 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 freqm_init()

Initializes a hardware FREQM module instance.

```
enum status_code freqm_init(
        struct freqm_module *const module_inst,
        Freqm *const hw,
        struct freqm_config *const config)
```

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

Table 7-2 Parameters

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

Returns
Status of the initialization procedure.

Table 7-3 Return Values

Return value	Description
STATUS_OK	The module was initialized successfully

7.3.1.2. Function freqm_get_config_defaults()

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

```
void freqm_get_config_defaults(
        struct freqm_config *const config)
```

Initializes all members of a given Frequency Meter 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:

- Measurement clock source is GCLK0
- Reference clock source is GCLK1
- Frequency Meter Reference Clock Cycles 127

Table 7-4 Parameters

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

7.3.1.3. Function freqm_enable()

Enables a FREQM that was previously configured.

```
void freqm_enable(
        struct freqm_module *const module_inst)
```

Enables Frequency Meter that was previously configured via a call to freqm_init().

Table 7-5 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Frequency Meter peripheral

7.3.1.4. Function freqm_disable()

Disables a FREQM that was previously enabled.

```
void freqm_disable(
        struct freqm_module *const module_inst)
```

Disables Frequency Meter that was previously started via a call to freqm_enable().

Table 7-6 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Software instance for the Frequency Meter peripheral

7.3.2. Read FREQM Result

7.3.2.1. Function freqm_start_measure()

Start a manual measurement process.

```
void freqm_start_measure(
        struct freqm_module *const module)
```

Table 7-7 Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the FREQM software instance struct

7.3.2.2. Function freqm_clear_overflow()

Clears module overflow flag.

```
void freqm_clear_overflow(
        struct freqm_module *const module)
```

Clears the overflow flag of the module.

Table 7-8 Parameters

Data direction	Parameter name	Description
[in]	module	Pointer to the FREQM software instance struct

7.3.2.3. Function freqm_get_result_value()

Read the measurement data result.

```
enum freqm_status freqm_get_result_value(
        struct freqm_module *const module_inst,
        uint32_t * result)
```

Reads the measurement data result.

Table 7-9 Parameters

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

Note: If overflow occurred, configure faster reference clock or reduce reference clock cycles.

Returns
Status of the FREQM read request.

Table 7-10 Return Values

Return value	Description
FREQM_STATUS_MEASURE_DONE	Measurement result was retrieved successfully
FREQM_STATUS_MEASURE_BUSY	Measurement result was not ready
FREQM_STATUS_CNT_OVERFLOW	Measurement result was overflow

7.3.3. Callback Configuration and Initialization

7.3.3.1. Function freqm_register_callback()

Registers a callback.

```
enum status_code freqm_register_callback(
        struct freqm_module *const module,
        freqm_callback_t callback_func,
        enum freqm_callback callback_type)
```

Registers a callback function which is implemented by the user.

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

Table 7-11 Parameters

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

Table 7-12 Return Values

Return value	Description
STATUS_OK	The function exited successfully

7.3.3.2. Function freqm_unregister_callback()

Unregisters a callback.

```
enum status_code freqm_unregister_callback(
        struct freqm_module * module,
        enum freqm_callback callback_type)
```

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

Table 7-13 Parameters

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

Table 7-14 Return Values

Return value	Description
STATUS_OK	The function exited successfully

7.3.4. Callback Enabling and Disabling

7.3.4.1. Function freqm_enable_callback()

Enable an FREQM callback.

```
enum status_code freqm_enable_callback(
        struct freqm_module *const module,
        const enum freqm_callback_type type)
```

Table 7-15 Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	type	Callback source type

Returns
Status of the callback enable operation.

Table 7-16 Return Values

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

7.3.4.2. Function freqm_disable_callback()

Disable an FREQM callback.

```
enum status_code freqm_disable_callback(
        struct freqm_module *const module,
        const enum freqm_callback_type type)
```

Table 7-17 Parameters

Data direction	Parameter name	Description
[in, out]	module	Pointer to the software instance struct
[in]	type	Callback source type

Returns
Status of the callback enable operation.

Table 7-18 Return Values

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

7.3.5. Function freqm_is_syncing()

Determines if the hardware module(s) are currently synchronizing to the bus.

```
bool freqm_is_syncing( void )
```

Checks to see if the underlying hardware peripheral module(s) are currently synchronizing across multiple clock domains to the hardware bus. This function can be used to delay further operations on a module until such time that it is ready, to prevent blocking delays for synchronization in the user application.

Returns
Synchronization status of the underlying hardware module(s).

Table 7-19 Return Values

Return value	Description
false	If the module has completed synchronization
true	If the module synchronization is ongoing

7.4. Enumeration Definitions

7.4.1. Enum freqm_callback

Enum for possible callback types for the FREQM module.

Table 7-20 Members

Enum value	Description
FREQM_CALLBACK_DONE	Callback for measurement done

7.4.2. Enum freqm_callback_type

FREQM callback type.

Table 7-21 Members

Enum value	Description
FREQM_CALLBACK_MEASURE_DONE	Measurement done callback.

7.4.3. Enum freqm_status

Enum for the possible status types for the FREQM module.

Table 7-22 Members

Enum value	Description
FREQM_STATUS_MEASURE_DONE	FREQM measurement is finish
FREQM_STATUS_MEASURE_BUSY	FREQM measurement is ongoing or not
FREQM_STATUS_CNT_OVERFLOW	FREQM sticky count value overflow

8. Extra Information for FREQM Driver

8.1. Acronyms

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

Acronym	Description
FREQM	Frequency Meter

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 FREQM Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM Frequency Meter (FREQM) 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 FREQM - Basic
- Quick Start Guide for FREQM - Callback

9.1. Quick Start Guide for FREQM - Basic

In this use case, the Frequency Meter (FREQM) module is configured for:

- The FREQM peripheral will not be stopped in standby sleep mode.

This use case will read measurement data in polling mode repeatly. 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:

```
/* FREQM module software instance (must not go out of scope while in use)
*/
static struct freqm_module freqm_instance;

void configure_freqm(void)
{
    /* Create a new configuration structure for the FREQM settings
     * and fill with the default module settings. */
    struct freqm_config config_freqm;
    freqm_get_config_defaults(&config_freqm);

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

    /* Initialize FREQM with the user settings */
    freqm_init(&freqm_instance, FREQM, &config_freqm);
}
```

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

```
system_init();
configure_freqm();
freqm_enable(&freqm_instance);
```

9.1.1.3. Workflow

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

   ```
   static struct freqm_module freqm_instance;
   ```

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

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

```
void configure_freqm(void)
```

3. Create an FREQM peripheral configuration structure that will be filled out to set the module configuration.

```
struct freqm_config config_freqm;
```

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

```
freqm_get_config_defaults(&config_freqm);
```

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

```
freqm_init(&freqm_instance, FREQM, &config_freqm);
```

6. Enable the now initialized FREQM peripheral.

```
freqm_enable(&freqm_instance);
```

9.1.2. Implementation

9.1.2.1. Code

Copy-paste the following code to your user application:

```
uint32_t measure_result;
enum freqm_status status;

freqm_start_measure(&freqm_instance);
while ((status = freqm_get_result_value(&freqm_instance, &measure_result))
        == FREQM_STATUS_MEASURE_BUSY) {
};
switch(status) {
    case FREQM_STATUS_MEASURE_DONE:
        LED_On(LED_0_PIN);
        while (true) {
        }
    case FREQM_STATUS_CNT_OVERFLOW:
        freqm_clear_overflow(&freqm_instance);
        while (true) {
            LED_Toggle(LED_0_PIN);
            volatile uint32_t delay = 50000;
            while(delay--) {
            }
        }
    default:
        Assert(false);
        break;
}
```

9.1.2.2. Workflow

1. Start FREQM measurement and wait until measure done then read result data.

```
freqm_start_measure(&freqm_instance);
while ((status = freqm_get_result_value(&freqm_instance,
&measure_result))
```

```
        == FREQM_STATUS_MEASURE_BUSY) {
};
```

2. The board LED is on to indicate a measurement data is read.

```
case FREQM_STATUS_MEASURE_DONE:
    LED_On(LED_0_PIN);
    while (true) {
    }
```

3. The board LED is toggle to indicate measurement is overflow.

```
case FREQM_STATUS_CNT_OVERFLOW:
    freqm_clear_overflow(&freqm_instance);
    while (true) {
        LED_Toggle(LED_0_PIN);
        volatile uint32_t delay = 50000;
        while(delay--) {
        }
    }
```

9.2. Quick Start Guide for FREQM - Callback

In this use case, the Frequency Meter (FREQM) module is configured for:

- The FREQM peripheral will not be stopped in standby sleep mode.

This use case will read measurement 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 freqm_read_done = false;

void configure_freqm(void);
void configure_freqm_callback(void);
void freqm_complete_callback(void);

/* FREQM module software instance (must not go out of scope while in use)
*/
static struct freqm_module freqm_instance;

void configure_freqm(void)
{
    /* Create a new configuration structure for the FREQM settings
     * and fill with the default module settings. */
    struct freqm_config config_freqm;
    freqm_get_config_defaults(&config_freqm);
    config_freqm.ref_clock_circles = 255;

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

    /* Initialize FREQM with the user settings */
    freqm_init(&freqm_instance, FREQM, &config_freqm);
}

void freqm_complete_callback(void)
{
    freqm_read_done = true;
}

void configure_freqm_callback(void)
{
    freqm_register_callback(&freqm_instance, freqm_complete_callback,
            FREQM_CALLBACK_MEASURE_DONE);
    freqm_enable_callback(&freqm_instance, FREQM_CALLBACK_MEASURE_DONE);
}
```

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

```
system_init();
configure_freqm();
configure_freqm_callback();

freqm_enable(&freqm_instance);
```

9.2.1.3. Workflow

1. Create an FREQM device instance struct, which will be associated with an FREQM peripheral hardware instance.

   ```
   static struct freqm_module freqm_instance;
   ```

   Note: Device instance structures shall **never** go out of scope when in use.
2. Create a new function `configure_freqm()`, which will be used to configure the overall FREQM peripheral.

   ```
   void configure_freqm(void)
   ```
3. Create an FREQM peripheral configuration structure that will be filled out to set the module configuration.

   ```
   struct freqm_config config_freqm;
   ```
4. Fill the FREQM peripheral configuration structure with the default module configuration values.

   ```
   freqm_get_config_defaults(&config_freqm);
   config_freqm.ref_clock_circles = 255;
   ```
5. Initialize the FREQM peripheral and associate it with the software instance structure that was defined previously.

   ```
   freqm_init(&freqm_instance, FREQM, &config_freqm);
   ```
6. Create a new callback function.

   ```
   void freqm_complete_callback(void)
   {
       freqm_read_done = true;
   }
   ```
7. Create a callback status software flag.

   ```
   bool volatile freqm_read_done = false;
   ```

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

```
freqm_read_done = true;
```

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

```
void configure_freqm_callback(void)
{
    freqm_register_callback(&freqm_instance, freqm_complete_callback,
            FREQM_CALLBACK_MEASURE_DONE);
    freqm_enable_callback(&freqm_instance,
FREQM_CALLBACK_MEASURE_DONE);
}
```

10. Register callback function.

```
freqm_register_callback(&freqm_instance, freqm_complete_callback,
        FREQM_CALLBACK_MEASURE_DONE);
```

11. Enable the callbacks.

```
freqm_enable_callback(&freqm_instance, FREQM_CALLBACK_MEASURE_DONE);
```

12. Enable the now initialized FREQM peripheral.

```
freqm_enable(&freqm_instance);
```

Note: This should not be done until after the FREQM is setup 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 measure_result;
enum freqm_status status;
freqm_start_measure(&freqm_instance);

while (!freqm_read_done) {
}
status = freqm_get_result_value(&freqm_instance, &measure_result);
switch(status) {
    case FREQM_STATUS_MEASURE_DONE:
        LED_On(LED_0_PIN);
        while (true) {
        }
    case FREQM_STATUS_CNT_OVERFLOW:
        freqm_clear_overflow(&freqm_instance);
        while (true) {
            LED_Toggle(LED_0_PIN);
            volatile uint32_t delay = 50000;
            while(delay--) {
            }
        }
    default:
        Assert(false);
        break;
}
```

9.2.2.2. Workflow

1. Start an asynchronous FREQM read job, to store measurement data into the global buffer and generate a callback when complete.

   ```
   freqm_start_measure(&freqm_instance);
   ```

2. Wait until the asynchronous read job is complete.

   ```
   while (!freqm_read_done) {
   }
   status = freqm_get_result_value(&freqm_instance, &measure_result);
   ```

3. The board LED on to indicate measurement data read.

   ```
   case FREQM_STATUS_MEASURE_DONE:
       LED_On(LED_0_PIN);
       while (true) {
       }
   ```

4. The board LED toggled to indicate measurement overflow occous.

   ```
   case FREQM_STATUS_CNT_OVERFLOW:
       freqm_clear_overflow(&freqm_instance);
       while (true) {
           LED_Toggle(LED_0_PIN);
           volatile uint32_t delay = 50000;
           while(delay--) {
           }
       }
   ```

10. Document Revision History

Doc. Rev.	Date	Comments
42506A	08/2015	Initial document release