AT07942: SAM4L Asynchronous Timer (AST) Driver

APPLICATION NOTE

Introduction

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

The Asynchronous Timer can generate periodic interrupts and peripheral events from output from the prescaler, as well as alarm interrupts and peripheral events, which can trigger at any counter value.

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

The AST module in the SAM4L devices is a 32-bit counter, with a 32-bit programmable prescaler. Typically, the AST clock is run continuously, including in the device's low-power sleep modes, to track the current time and date information. The AST can also wake-up the system from backup mode using either the alarm wakeup, periodic wakeup, or overflow wake-up mechanisms.

The AST has been designed to meet the system tick and Real-Time Clock requirements of most embedded operating systems.

In this driver, the AST is operated in Calendar Mode. This allows for an easy integration of a Real-Time Clock and calendar into a user application to track the passing of time and/or perform scheduled tasks.

Whilst operating in Calendar Mode, the AST features:

- Time tracking in seconds, minutes, and hours in 24 hour format
- Date tracking in day, month, and year
- Automatic leap year correction

3.1. Alarm Interrupt

The AST has a device dependent number of independent hardware alarms that can be configured by the user application. These alarms will be triggered on a match with the current clock value, and can be set up to trigger an interrupt, event, or both. The AST can also be configured to clear the clock value on alarm match which will generate an overflow interrupt.

Note:
Refer to module configuration at the end of the AST section of the device datasheet for the number of alarms supported.

Whilst in Calendar Mode and using a nominal 1Hz input clock frequency, a register overflow will occur after 64 years.

3.2. Periodic Events

The AST can generate events at periodic intervals, allowing for direct peripheral actions without CPU intervention. The periodic events can be generated on an AST prescaler match, and will be generated on the rising edge transition of the specified bit. The resulting periodic frequency can be calculated by the following formula:

$$f_{PERIODIC} = \frac{f_{ASY}}{2^{n+1}}$$

Where
f_{ASY}
refers to the *asynchronous* clock set up in the AST module configuration.

Note: The connection of events between modules requires the use of the SAM4L Peripheral Event Controller (PEVC) to route an output event of one module to the the input event of another. For more information on event routing, refer to the event driver documentation.

3.3. Digital Tuner

The AST module contains Digital Tuner logic to compensate for inaccurate source clock frequencies, which would otherwise result in skewed time measurements.

4. Special Considerations

4.1. Crystal Selection

The external crystal selection used by the AST module in the final system design must take into account:

- Current consumption to achieve the best power savings in low power operating modes
- Frequency drift (due to temperature effects on the circuit) for the best time accuracy

4.2. Year Limit

The AST module has a year range of 63 years from the starting year configured when the module is initialized. Dates outside the start to end year range described below will need software adjustment:

$$[YEAR_{START}, YEAR_{START} + 64]$$

5. Extra Information

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

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

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

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type ast_callback_t

```
typedef void(* ast_callback_t )(void)
```

AST interrupt callback.

7.1.2. Type ast_event_source_t

```
typedef enum ast_event_source ast_event_source_t
```

AST event source.

7.1.3. Type ast_interrupt_source_t

```
typedef enum ast_interrupt_source ast_interrupt_source_t
```

AST interrupt source.

7.1.4. Type ast_mode_t

```
typedef enum ast_mode ast_mode_t
```

AST Calendar Mode.

7.1.5. Type ast_oscillator_type_t

```
typedef enum ast_oscillator_type ast_oscillator_type_t
```

AST Oscillator type.

7.1.6. Type ast_wakeup_source_t

```
typedef enum ast_wakeup_source ast_wakeup_source_t
```

AST wake-up source.

7.2. Structure Definitions

7.2.1. Struct ast_calendar

Table 7-1 Members

Type	Name	Description
union ast_calendar.@1	@1	

7.2.2. Union ast_calendar.__unnamed__

Table 7-2 Members

Type	Name	Description
uint32_t	field	
struct ast_calv	FIELD	Calendar

7.2.3. Struct ast_calv

Description for Calendar Field.

Table 7-3 Members

Type	Name	Description
uint32_t	day	Day in the range 1 to 31
uint32_t	hour	Hours in the range 0 to 23
uint32_t	min	Minutes in the range range 0 to 59
uint32_t	month	Month in the range 1 to 12
uint32_t	sec	Seconds in the range 0 to 59
uint32_t	year	Year in the range 0 to 63

7.2.4. Struct ast_config

AST configuration.

Table 7-4 Members

Type	Name	Description
struct ast_calendar	calendar	Initial calendar Value
uint32_t	counter	Initial counter Value
ast_mode_t	mode	Mode: Calendar Mode: AST_CALENDAR_MODE or Counter Mode: AST_COUNTER_MODE
ast_oscillator_type_t	osc_type	Oscillator type
uint8_t	psel	Prescaler Value

7.3. Macro Definitions

7.3.1. Predefined PSEL Values

7.3.1.1. Macro AST_PSEL_32KHZ_1HZ

```
#define AST_PSEL_32KHZ_1HZ
```

7.3.1.2. Macro AST_PSEL_RC_1_76HZ

```
#define AST_PSEL_RC_1_76HZ
```

7.3.2. Macro AST_POLL_TIMEOUT

```
#define AST_POLL_TIMEOUT
```

Timeout to prevent code hang in clock initialization.

7.4. Function Definitions

7.4.1. Function ast_clear_interrupt_flag()

Clear an AST interrupt status flag.

```
void ast_clear_interrupt_flag(
        Ast * ast,
        ast_interrupt_source_t source)
```

Table 7-5 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST interrupt source for which the status is to be cleared

7.4.2. Function ast_clear_prescalar()

Clear the AST periodic prescaler counter to zero.

```
void ast_clear_prescalar(
        Ast * ast)
```

Table 7-6 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

7.4.3. Function ast_configure_digital_tuner()

Tune the AST prescaler frequency to the desired frequency.

```
uint32_t ast_configure_digital_tuner(
        Ast * ast,
```

```
        uint32_t input_freq,
        uint32_t tuned_freq)
```

Note:
Refer to the section entitled "digital tuner" in the ast chapter of the device-specific datasheet for more information.

Table 7-7 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	input_freq	Prescaled AST Clock Frequency
[in]	tuned_freq	Desired AST frequency

7.4.4. Function ast_disable()

Disable the AST counter and module.

```
void ast_disable(
        Ast * ast)
```

Table 7-8 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

7.4.5. Function ast_disable_digital_tuner()

Disable the AST digital tuner.

```
void ast_disable_digital_tuner(
        Ast * ast)
```

Table 7-9 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

7.4.6. Function ast_disable_event()

Disable an AST event.

```
void ast_disable_event(
        Ast * ast,
        ast_event_source_t source)
```

Table 7-10 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST event source to be disabled

7.4.7. Function ast_disable_interrupt()

Disable an AST interrupt.

```
void ast_disable_interrupt(
        Ast * ast,
        ast_interrupt_source_t source)
```

Table 7-11 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST interrupt source to be disabled

7.4.8. Function ast_disable_wakeup()

Disable an AST asynchronous wake-up source.

```
void ast_disable_wakeup(
        Ast * ast,
        ast_wakeup_source_t source)
```

Table 7-12 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST wake-up source to be disabled

7.4.9. Function ast_enable()

Enable the AST module.

```
void ast_enable(
        Ast * ast)
```

Table 7-13 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

7.4.10. Function ast_enable_counter_clear_on_alarm()

Enable the option to clear the counter on an AST alarm.

```
void ast_enable_counter_clear_on_alarm(
        Ast * ast,
        uint8_t alarm_channel)
```

Table 7-14 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	alarm_channel	AST Alarm Channel

7.4.11. Function ast_enable_event()

Enable an AST event.

```
void ast_enable_event(
        Ast * ast,
        ast_event_source_t source)
```

Table 7-15 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST event source to be enabled

7.4.12. Function ast_enable_interrupt()

Enable an AST interrupt.

```
void ast_enable_interrupt(
        Ast * ast,
        ast_interrupt_source_t source)
```

Table 7-16 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST interrupt source to be enabled

7.4.13. Function ast_enable_wakeup()

Enable an AST asynchronous wake-up source.

```
void ast_enable_wakeup(
        Ast * ast,
        ast_wakeup_source_t source)
```

Table 7-17 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST wake-up source to be enabled

7.4.14. Function ast_init_digital_tuner()

Initialize the AST digital tuner.

```
void ast_init_digital_tuner(
        Ast * ast,
        bool add,
        uint8_t value,
        uint8_t exp)
```

Table 7-18 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	add	Set to true if the frequency needs to be increased, false if it has to be decreased.
[in]	value	Parameter used in the formula
[in]	exp	Parameter used in the formula

7.4.15. Function ast_is_busy()

Check the busy status of AST.

```
bool ast_is_busy(
        Ast * ast)
```

Table 7-19 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
AST busy status.

Table 7-20 Return Values

Return value	Description
false	AST is not busy
true	AST is busy

7.4.16. Function ast_is_clkbusy()

Check the busy status of the AST clock.

```
bool ast_is_clkbusy(
        Ast * ast)
```

Table 7-21 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
AST clock busy status.

Table 7-22 Return Values

Return value	Description
false	AST clock is not busy
true	AST clock is busy

7.4.17. Function ast_is_enabled()

Check the status of the AST module.

```
bool ast_is_enabled(
        Ast * ast)
```

Table 7-23 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

Table 7-24 Return Values

Return value	Description
false	AST is not enabled
true	AST is enabled

7.4.18. Function ast_read_calendar_value()

Get the AST current calendar value.

```
struct ast_calendar ast_read_calendar_value(
        Ast * ast)
```

Note: There maybe a compiler warning about returning a structure type value, however it is safe because ast_calendar is actually uint32_t type.

Table 7-25 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
The AST current calendar value.

7.4.19. Function ast_read_counter_value()

Get the AST current counter value.

```
uint32_t ast_read_counter_value(
        Ast * ast)
```

Table 7-26 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
AST current counter value.

7.4.20. Function ast_read_interrupt_mask()

Get the AST interrupt mask value.

```
uint32_t ast_read_interrupt_mask(
        Ast * ast)
```

Table 7-27 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
AST Interrupt mask value.

7.4.21. Function ast_read_status()

Get the status of AST.

```
uint32_t ast_read_status(
        Ast * ast)
```

Table 7-28 Parameters

Data direction	Parameter name	Description
[in]	ast	Module hardware register base address pointer

Returns
AST status.

7.4.22. Function ast_set_callback()

Set callback for AST interrupts.

```
void ast_set_callback(
        Ast * ast,
        ast_interrupt_source_t source,
        ast_callback_t callback,
```

```
        uint8_t irq_line,
        uint8_t irq_level)
```

Table 7-29 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	source	AST interrupt source
[in]	callback	Callback function pointer
[in]	irq_line	Interrupt line
[in]	irq_level	Interrupt level

7.4.23. Function ast_set_config()

Initialize and enable the AST module in Calendar Mode or Counter Mode.

```
uint32_t ast_set_config(
        Ast * ast,
        struct ast_config * ast_conf)
```

Note: If you use the 32kHz oscillator it will be enabled by this function.

Table 7-30 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	ast_conf	AST configuration structure pointer

Returns
Initialization result.

Table 7-31 Return Values

Return value	Description
0	Initialization failed due to a software timeout
1	Initialization succeeded

7.4.24. Function ast_start()

Start the AST counter.

```
void ast_start(
        Ast * ast)
```

Table 7-32 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

7.4.25. Function ast_stop()

Stop the AST counter.

```
void ast_stop(
        Ast * ast)
```

Table 7-33 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer

7.4.26. Function ast_write_alarm0_value()

Set the AST alarm0 value.

```
void ast_write_alarm0_value(
        Ast * ast,
        uint32_t alarm_value)
```

Table 7-34 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	alarm_value	AST alarm0 value

7.4.27. Function ast_write_calendar_value()

Set the AST current calendar value.

```
void ast_write_calendar_value(
        Ast * ast,
        struct ast_calendar calendar)
```

Table 7-35 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	calendar	Startup date structure

7.4.28. Function ast_write_counter_value()

Set the AST current counter value.

```
void ast_write_counter_value(
        Ast * ast,
        uint32_t ast_counter)
```

Table 7-36 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	ast_counter	Startup counter value

7.4.29. Function ast_write_periodic0_value()

Set the AST periodic0 value.

```
void ast_write_periodic0_value(
        Ast * ast,
        uint32_t pir)
```

Table 7-37 Parameters

Data direction	Parameter name	Description
[in, out]	ast	Module hardware register base address pointer
[in]	pir	AST periodic0

7.5. Enumeration Definitions

7.5.1. Enum ast_event_source

AST event source.

Table 7-38 Members

Enum value	Description
AST_EVENT_ALARM	Alarm event generation
AST_EVENT_PER	Peripheral event generation
AST_EVENT_OVF	Counter overflow event generation

7.5.2. Enum ast_interrupt_source

AST interrupt source.

Table 7-39 Members

Enum value	Description
AST_INTERRUPT_ALARM	Alarm interrupt
AST_INTERRUPT_PER	Periodic interrupt
AST_INTERRUPT_OVF	Overflow interrupt
AST_INTERRUPT_READY	Synchronization complete interrupt
AST_INTERRUPT_CLKREADY	Clock synchronization complete interrupt

7.5.3. Enum ast_mode

AST Calendar Mode.

Table 7-40 Members

Enum value	Description
AST_COUNTER_MODE	Counter Mode
AST_CALENDAR_MODE	Calendar Mode

7.5.4. Enum ast_oscillator_type

AST Oscillator type.

Table 7-41 Members

Enum value	Description
AST_OSC_RC	System RC oscillator (RCSYS)
AST_OSC_32KHZ	32kHz oscillator (OSC32 or RC32)
AST_OSC_PB	APB clock
AST_OSC_GCLK	Generic clock (GCLK)
AST_OSC_1KHZ	1kHz clock from the 32kHz oscillator or 32kHz RC oscillator (CLK_1K)

7.5.5. Enum ast_wakeup_source

AST wake-up source.

Table 7-42 Members

Enum value	Description
AST_WAKEUP_ALARM	Alarm wake-up source
AST_WAKEUP_PER	Peripheral interrupt wake-up source
AST_WAKEUP_OVF	Counter overflow wake-up source

8. Extra Information for Asynchronous Timer Driver

8.1. Acronyms

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

Acronym	Definition
BPM	Backup Power Manager
QSG	Quick Start Guide

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 Asynchronous Timer Driver

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4L Asynchronous Timer (AST) 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 AST driver
- Asynchronous Timer (AST) - Example 1 Calendar Mode
- Asynchronous Timer (AST) - Example 2 Alarm Wakeup

9.1. Quick Start Guide for the AST driver

This is the quick start guide for the SAM4L Asynchronous Timer (AST) Driver, with step-by-step instructions on how to configure and use the driver for a specific use case.The code examples can be copied into the main application loop or any other function that will need to control the AST module.

9.1.1. Use Cases

- AST Basic Usage

9.1.2. AST Basic Usage

This use case will demonstrate how to initialize the AST module to calendar or counter mode.

9.1.3. Setup Steps

9.1.3.1. Prerequisites

This module requires the following service:

- System Clock Management (sysclock)

9.1.3.2. Setup Code

Add this to the main loop or a setup function:

```
osc_priv_enable_osc32();
```

9.1.3.3. Workflow

1. Enable the AST module:

   ```
   ast_enable(AST);
   ```

2. Initialize the AST to counter mode:

   ```
   * struct ast_config ast_conf;
   * ast_conf.mode = AST_COUNTER_MODE;
   * ast_conf.osc_type = AST_OSC_32KHZ;
   * ast_conf.psel = AST_PSEL_32KHZ_1HZ;
   * ast_conf.counter = 0;
   * ast_set_config(AST, &ast_conf);
   *
   ```

3. Or initialize the AST to calendar mode:

   ```
   * struct ast_calendar calendar;
   * struct ast_config ast_conf;
   ```

```
* calendar.FIELD.sec = 0;
* calendar.FIELD.min = 15;
* calendar.FIELD.hour = 12;
* calendar.FIELD.day = 20;
* calendar.FIELD.month = 9;
* calendar.FIELD.year = 12;
* struct ast_config ast_conf;
* ast_conf.mode = AST_CALENDAR_MODE;
* ast_conf.osc_type = AST_OSC_32KHZ;
* ast_conf.psel = AST_PSEL_32KHZ_1HZ;
* ast_conf.calendar = calendar;
* ast_set_config(AST, &ast_conf)
```

Note:
We need to set the clock after prescaler to 1Hz.

9.1.4. Usage Steps

9.1.4.1. Usage Code

We can get the calendar value by:

```
calendar = ast_read_calendar_value(AST);
```

Or we can get the counter value by:

```
ast_counter = ast_read_counter_value(AST);
```

We can set the alarm interrupt by:

```
* ast_write_alarm0_value(AST, calendar.field + 1);
* ast_set_callback(AST, ast_interrupt_alarm, ast_alarm_callback,
*    AST_ALARM_IRQn, 1);
```

And we can set the periodic interrupt by:

```
* ast_write_periodic0_value(AST, AST_PSEL_32KHZ_1HZ - 4);
* ast_set_callback(AST, ast_interrupt_per, ast_per_callback,
*    AST_PER_IRQn, 1);
```

9.2. Asynchronous Timer (AST) - Example 1 Calendar Mode

9.2.1. Purpose

This example demonstrates how to use the AST driver with the 32kHz oscillator crystal.

9.2.2. Requirements

This example can be used with SAM4L evaluation kits.

9.2.3. Description

Upon startup, the program uses the USART driver to display a real time counter incremented every second. The display is triggered by the interrupt controller module.

9.2.4. Main Files

- ast.c: Asynchronous Timer driver

- ast.h: Asynchronous Timer driver header file
- ast_example1.c: Asynchronous Timer example application

9.2.5. Compilation Information

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

9.2.6. Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. Start the application.
4. In the terminal window, the following text should appear:

```
*      -- AST Example 1 in Calendar Mode --
*      -- xxxxxx-xx
*      -- Compiled: xxx xx xxxx xx:xx:xx --
*
*      Config AST with 32 KHz oscillator.
*
*      Calendar: Year:XX Month:XX Day:XX, XXhXXmXXs
```

5. Approximately every second the time should increment.

9.3. Asynchronous Timer (AST) - Example 2 Alarm Wakeup

9.3.1. Purpose

This example demonstrates how to use the AST driver with the 32kHz oscillator crystal and to also setup an alarm to wake-up the device from its low power modes.

9.3.2. Requirements

This example can be used with SAM4L evaluation kits.

9.3.3. Description

Upon startup, the program uses the USART driver to display a menu that lets the user select the low power mode from which the device will wake-up when the AST Alarm triggers.

9.3.4. Main Files

- ast.c: Asynchronous Timer driver
- ast.h: Asynchronous Timer driver header file
- ast_example2.c: Asynchronous Timer example application using AST wakeup and BPM sleep function.

9.3.5. Compilation Information

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

9.3.6. Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 baud
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. Start the application.
4. In the terminal window, the following text should appear:

```
*     -- -- AST Example 2 in Counter Mode --
*     -- xxxxxx-xx
*     -- Compiled: xxx xx xxxx xx:xx:xx --
*
*      Config AST with 32kHz oscillator.
*     Use alarm0 to wakeup from low power mode.
*
*       Menu :
*             -- Select low power mode:
*             1: Sleep mode 0
*             2: Sleep mode 1
*             3: Sleep mode 2
*             4: Sleep mode 3
*             5: Wait mode
*             6: Retention mode
*             7: Backup mode
```

10. Document Revision History

Doc. Rev.	Date	Comments
42283B	07/2015	Updated title of application note and added list of supported devices
42283A	05/2014	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**