SMART ARM-based Microcontrollers

AT07898: SAM3/4S/4L/4E/4N/4CM/4C/G Timer Counter (TC) Driver

APPLICATION NOTE

Introduction

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

The Timer Counter (TC) includes several identical 16-bit or 32-bit Timer Counter channels. Each channel can be independently programmed to perform a wide range of functions that includes frequency measurement, event counting, interval measurement, pulse generation, delay timing, and pulse width modulation.

Devices from the following series can use this module:

- Atmel | SMART SAM3
- Atmel | SMART SAM4S
- Atmel | SMART SAM4L
- Atmel | SMART SAM4E
- Atmel | SMART SAM4N
- Atmel | SMART SAM4CM
- Atmel | SMART SAM4C
- Atmel | SMART SAMG

The outline of this documentation is as follows:

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

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The Timer Counter (TC) includes several identical 16-bit or 32-bit Timer Counter channels. The number of TC channels is device specific, refer to the device-specific datasheet for more information.

Each channel can be independently programmed to perform a wide range of functions that includes frequency measurement, event counting, interval measurement, pulse generation, delay timing, and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs, and two multi-purpose input/ output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The Timer Counter (TC) embeds a quadrature decoder logic connected in front of the timers. When enabled, the quadrature decoder performs the input line filtering, decoding of quadrature signals and connects to the timers/counters in order to read the position and speed of the motor.

4. Special Considerations

4.1. External Clock

In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock (MCLK) period. The external clock frequency must be at least 2.5 times lower than the master clock.

4.2. External Trigger

If an external trigger is used, the duration of its pulses must be longer than the master clock (MCLK) period in order to be detected.

5. Extra Information

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

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

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

7. API Overview

7.1. Function Definitions

7.1.1. Function tc_disable_interrupt()

Disable TC interrupts on the specified channel.

```
void tc_disable_interrupt(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_sources)
```

Table 7-1 Parameters

Data direction	Parameter name	Description
[in, out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure
[in]	ul_sources	A bitmask of Interrupt sources

Where the input parameter *ul_sources* can be one or more of the following:

Parameter Value	Description
TC_IDR_COVFS	Disables the Counter Overflow Interrupt
TC_IDR_LOVRS	Disables the Load Overrun Interrupt
TC_IDR_CPAS	Disables the RA Compare Interrupt
TC_IDR_CPBS	Disables the RB Compare Interrupt
TC_IDR_CPCS	Disables the RC Compare Interrupt
TC_IDR_LDRAS	Disables the RA Load Interrupt
TC_IDR_LDRBS	Disables the RB Load Interrupt
TC_IDR_ETRGS	Disables the External Trigger Interrupt

7.1.2. Function tc_disable_qdec_interrupt()

Disable TC QDEC interrupts.

```
void tc_disable_qdec_interrupt(
        Tc * p_tc,
        uint32_t ul_sources)
```

Note: This function is not available on SAM4L or SAMG devices.

Table 7-2 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_sources	A bitmask of QDEC interrupts to be disabled

Where the input parameter *ul_sources* can be one or more of the following:

Parameter Value	Description
TC_QIDR_IDX	Disable the rising edge detected on IDX input interrupt
TC_QIDR_DIRCHG	Disable the change in rotation direction detected interrupt
TC_QIDR_QERR	Disable the quadrature error detected on PHA/PHB interrupt

7.1.3. Function tc_enable_interrupt()

Enable the TC interrupts on the specified channel.

```
void tc_enable_interrupt(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_sources)
```

Table 7-3 Parameters

Data direction	Parameter name	Description
[in, out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure
[in]	ul_sources	Bitmask of interrupt sources

Where the input parameter *ul_sources* can be one or more of the following:

Parameter Value	Description
TC_IER_COVFS	Enables the Counter Overflow Interrupt
TC_IER_LOVRS	Enables the Load Overrun Interrupt
TC_IER_CPAS	Enables the RA Compare Interrupt
TC_IER_CPBS	Enables the RB Compare Interrupt
TC_IER_CPCS	Enables the RC Compare Interrupt
TC_IER_LDRAS	Enables the RA Load Interrupt
TC_IER_LDRBS	Enables the RB Load Interrupt
TC_IER_ETRGS	Enables the External Trigger Interrupt

7.1.4. Function tc_enable_qdec_interrupt()

Enable TC QDEC interrupts.

```
void tc_enable_qdec_interrupt(
        Tc * p_tc,
        uint32_t ul_sources)
```

Note: This function is not available on SAM4L or SAMG devices.

Table 7-4 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_sources	A bitmask of QDEC interrupts to be enabled

Where the input parameter *ul_sources* can be one or more of the following:

Parameter Value	Description
TC_QIER_IDX	Enable the rising edge detected on IDX input interrupt
TC_QIER_DIRCHG	Enable the change in rotation direction detected interrupt
TC_QIER_QERR	Enable the quadrature error detected on PHA/PHB interrupt

7.1.5. Function tc_find_mck_divisor()

Find the best PBA/MCK divisor.

```
uint32_t tc_find_mck_divisor(
        uint32_t ul_freq,
        uint32_t ul_mck,
        uint32_t * p_uldiv,
        uint32_t * ul_tcclks,
        uint32_t ul_boardmck)
```

For SAM4L devices: Finds the best PBA divisor given the timer frequency and PBA clock. The result is guaranteed to satisfy the following equation:

```
(ul_pbaclk / (2* DIV * 65536)) <= freq <= (ul_pbaclk / (2* DIV))
```

with DIV being the lowest possible value, to maximize timing adjust resolution.

For non SAM4L devices: Finds the best MCK divisor given the timer frequency and MCK. The result is guaranteed to satisfy the following equation:

```
(MCK / (DIV * 65536)) <= freq <= (MCK / DIV)
```

with DIV being the lowest possible value, to maximize timing adjust resolution.

Table 7-5 Parameters

Data direction	Parameter name	Description
[in]	ul_freq	Desired timer frequency
[in]	ul_mck	PBA clock frequency
[out]	p_uldiv	Divisor value
[out]	p_ultcclks	TCCLKS field value for divisor
[in]	ul_boardmck	Board clock frequency (set to 0 for SAM4L devices)

Returns
The divisor found status.

Table 7-6 Return Values

Return value	Description
0	No suitable divisor was found
1	A divisor was found

7.1.6. Function tc_get_feature()

Indicate TC features.

```
uint32_t tc_get_feature(
        Tc * p_tc)
```

Note: This function is only available on SAM4L devices.

Table 7-7 Parameters

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

Returns
The TC FEATURES register contents.

7.1.7. Function tc_get_interrupt_mask()

Read the TC interrupt mask for the specified channel.

```
uint32_t tc_get_interrupt_mask(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-8 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to read

Returns
The TC interrupt mask value.

7.1.8. Function tc_get_qdec_interrupt_mask()

Read TC QDEC interrupt mask.

```
uint32_t tc_get_qdec_interrupt_mask(
        Tc * p_tc)
```

Note: This function is not available on SAM4L or SAMG devices.

Table 7-9 Parameters

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

Returns
The QDEC interrupt mask value.

7.1.9. Function tc_get_qdec_interrupt_status()

Get current TC QDEC interrupt status.

```
uint32_t tc_get_qdec_interrupt_status(
        Tc * p_tc)
```

Note: This function is not available on SAM4L or SAMG devices.

Table 7-10 Parameters

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

Returns
The TC QDEC interrupt status.

7.1.10. Function tc_get_status()

Get the current status for the specified TC channel.

```
uint32_t tc_get_status(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-11 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel number

Returns
The current TC status.

7.1.11. Function tc_get_version()

Indicate TC version.

```
uint32_t tc_get_version(
        Tc * p_tc)
```

Note: This function is only available on SAM4L devices.

Table 7-12 Parameters

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

Returns
The TC VERSION register contents.

7.1.12. Function tc_init()

Configure TC for timer, waveform generation, or capture.

```
void tc_init(
        Tc * p_tc,
        uint32_t ul_Channel,
        uint32_t ul_Mode)
```

Table 7-13 Parameters

Data direction	Parameter name	Description
[in, out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure
[in]	ul_mode	Control mode register bitmask value to set

Note: For more information regarding *ul_mode* configuration refer to the section entitled "Channel Mode Register: Capture Mode" and/or section "Waveform Operating Mode" in the device-specific datasheet.

Note: If the TC is configured for waveform generation then the external event selection (EEVT) should only be set to TC_CMR_EEVT_TIOB, or the equivalent value of 0, if it really is the intention to use TIOB as an external event trigger. This is because this setting forces TIOB to be an input, even if the external event trigger has not been enabled with TC_CMR_ENETRG, and thus prevents normal operation of TIOB.

7.1.13. Function tc_init_2bit_gray()

Configure TC for 2-bit Gray Counter for Stepper Motor.

```
uint32_t tc_init_2bit_gray(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_steppermode)
```

Note: The function tc_init() must be called prior to this one.

Note: This function is not available on SAM3U devices.

Table 7-14 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure
[in]	ul_steppermode	Stepper motor mode register value to set

Returns
0 for OK.

7.1.14. Function tc_read_cv()

Read the counter value on the specified channel.

```
uint32_t tc_read_cv(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-15 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to read

Returns
The counter value.

7.1.15. Function tc_read_ra()

Read TC Register A (RA) on the specified channel.

```
uint32_t tc_read_ra(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-16 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to read

Returns
The TC Register A (RA) value.

7.1.16. Function tc_read_rb()

Read TC Register B (RB) on the specified channel.

```
uint32_t tc_read_rb(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-17 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to read

Returns
The TC Register B (RB) value.

7.1.17. Function tc_read_rc()

Read TC Register C (RC) on the specified channel.

```
uint32_t tc_read_rc(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-18 Parameters

Data direction	Parameter name	Description
[in]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to read

Returns
The Register C (RC) value.

7.1.18. Function tc_set_block_mode()

Configure the TC Block mode.

```
void tc_set_block_mode(
        Tc * p_tc,
        uint32_t ul_blockmode)
```

Note: The function tc_init() must be called prior to this one.

Table 7-19 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_blockmode	Block mode register value to set

Note: For more information regarding *ul_blockmode* configuration refer to the section entitled "TC Block Mode Register" in the device-specific datasheet.

7.1.19. Function tc_set_writeprotect()

Enable or disable write protection of TC registers.

```
void tc_set_writeprotect(
        Tc * p_tc,
        uint32_t ul_enable)
```

Note: This function is not available on SAM3U devices.

Table 7-20 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_enable	1 to enable, 0 to disable

7.1.20. Function tc_start()

Start the TC clock on the specified channel.

```
void tc_start(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-21 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure

7.1.21. Function tc_stop()

Stop the TC clock on the specified channel.

```
void tc_stop(
        Tc * p_tc,
        uint32_t ul_channel)
```

Table 7-22 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to configure

7.1.22. Function tc_sync_trigger()

Asserts a SYNC signal to generate a software trigger on all channels.

```
void tc_sync_trigger(
        Tc * p_tc)
```

Table 7-23 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer

7.1.23. Function tc_write_ra()

Write to TC Register A (RA) on the specified channel.

```
void tc_write_ra(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_value)
```

Table 7-24 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to write
[in]	ul_value	Value to write

7.1.24. Function tc_write_rb()

Write to TC Register B (RB) on the specified channel.

```
void tc_write_rb(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_value)
```

Table 7-25 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to write
[in]	ul_value	Value to write

7.1.25. Function tc_write_rc()

Write to TC Register C (RC) on the selected channel.

```
void tc_write_rc(
        Tc * p_tc,
        uint32_t ul_channel,
        uint32_t ul_value)
```

Table 7-26 Parameters

Data direction	Parameter name	Description
[out]	p_tc	Module hardware register base address pointer
[in]	ul_channel	Channel to write
[in]	ul_value	Value to write

8. Extra Information for Timer Counter Driver

8.1. Acronyms

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

Acronym	Definition
MCK	Master Clock
PBA	Peripheral Bus A clock
PHA	Quadrature Decoder input signal Phase A
PHB	Quadrature Decoder input signal Phase B
QDEC	Quadrature Decoder
QSG	Quick Start Guide
RA	Register A
RB	Register B
RC	Register C
TIOB	Timer Input Output B

8.2. Dependencies

This driver has the following dependencies:

- System Clock Management (sysclock)
- General Purpose I/O (GPIO) driver
- Power Manager Controller (PMC) 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
Initial document release

9. Examples for Timer Counter

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM3/4S/4L/4E/4N/4CM/4C/G Timer Counter (TC) 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 TC driver
- Timer Counter Capture Waveform Example

9.1. Quick Start Guide for the TC driver

This is the quick start guide for the SAM3/4S/4L/4E/4N/4CM/4C/G Timer Counter (TC) 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

- TC Capture Mode Basic Usage
- TC Waveform Mode Basic Usage

9.1.2. TC Capture Mode Basic Usage

This use case will demonstrate how to initialize the TC module to operate in capture mode using interrupts. Note, that the macros used to select the TC channel are device specific. Refer to the appropriate device-specific datasheet for more information.

9.1.3. Setup Steps

9.1.3.1. Prerequisites

This module requires the following services:

- System Clock Management (sysclock)
- General Purpose I/O (GPIO) driver

9.1.3.2. Setup Code

Add these macros to the top of your main application C-file:

```
/* Use TC Peripheral 0. */
#define TC             TC0
#define TC_PERIPHERAL  0
```

```
/* Configure TC0 channel 2 as capture input. */
#define TC_CHANNEL_CAPTURE 2
#define ID_TC_CAPTURE ID_TC2
#define PIN_TC_CAPTURE PIN_TC0_TIOA2
#define PIN_TC_CAPTURE_MUX PIN_TC0_TIOA2_MUX
```

```
/* Use TC2_Handler for TC capture interrupt. */
#define TC_Handler  TC2_Handler
#define TC_IRQn     TC2_IRQn
```

Add this macro and functions to your main application C-file:

```
#define TC_CAPTURE_TIMER_SELECTION TC_CMR_TCCLKS_TIMER_CLOCK3
```

```
static void tc_capture_initialize(void)
{
    /* Configure the PMC to enable the TC module */
    sysclk_enable_peripheral_clock(ID_TC_CAPTURE);
#if SAMG55
    /* Enable PCK output */
    pmc_disable_pck(PMC_PCK_3);
    pmc_switch_pck_to_mck(PMC_PCK_3, PMC_PCK_PRES_CLK_1);
    pmc_enable_pck(PMC_PCK_3);
#endif

    /* Init TC to capture mode. */
    tc_init(TC, TC_CHANNEL_CAPTURE,
            TC_CAPTURE_TIMER_SELECTION /* Clock Selection */
            | TC_CMR_LDRA_RISING /* RA Loading: rising edge of TIOA */
            | TC_CMR_LDRB_FALLING /* RB Loading: falling edge of TIOA */
            | TC_CMR_ABETRG /* External Trigger: TIOA */
            | TC_CMR_ETRGEDG_FALLING /* External Trigger Edge: Falling
edge */
    );
}
```

```
void TC_Handler(void)
{
```

```
}
```

9.1.3.3. Workflow

1. Enable the TC module's capture pin:

   ```
   ioport_set_pin_mode(PIN_TC_CAPTURE, PIN_TC_CAPTURE_MUX);
   ioport_disable_pin(PIN_TC_CAPTURE);
   ```

2. Initialize the capture channel to the following:
   - Load RA on the rising edge of TIOA
   - Load RB on the falling edge of TIOA
   - Set the external trigger to TIOA
   - Set the external trigger to falling edge

     ```
     tc_capture_initialize();
     ```

3. Enable the TC interrupt using NVIC:

   ```
   NVIC_DisableIRQ(TC_IRQn);
   NVIC_ClearPendingIRQ(TC_IRQn);
   NVIC_SetPriority(TC_IRQn, 0);
   NVIC_EnableIRQ(TC_IRQn);
   ```

4. Enable the capture channel interrupt:

   ```
   tc_enable_interrupt(TC, TC_CHANNEL_CAPTURE, TC_IER_LDRBS);
   ```

5. In the TC_Handler() function, the load. RB interrupt can be checked by:

```
if ((tc_get_status(TC, TC_CHANNEL_CAPTURE) & TC_SR_LDRBS) ==
TC_SR_LDRBS) {
```

```
}
```

6. In the TC_Handler() function, the RA value. can be read by:

```
uint32_t gs_ul_captured_ra;
```

```
gs_ul_captured_ra = tc_read_ra(TC, TC_CHANNEL_CAPTURE);
```

7. In the TC_Handler() function, the RB value. can be read by:

```
uint32_t gs_ul_captured_rb;
```

```
gs_ul_captured_rb = tc_read_rb(TC, TC_CHANNEL_CAPTURE);
```

9.1.4. TC Waveform Mode Basic Usage

This use case will demonstrate how to initialize the TC module to operate in waveform mode. Note, that the macros used to select the TC channel are device specific. Refer to the appropriate device-specific datasheet for more information.

9.1.5. Setup Steps

9.1.5.1. Prerequisites

This module requires the following services:

- System Clock Management (sysclock)
- General Purpose I/O (GPIO) driver

9.1.5.2. Setup Code

Add these macros to the top of your main application C-file:

```
/* Use TC Peripheral 0. */
#define TC              TC0
#define TC_PERIPHERAL   0
```

```
/* Configure TC0 channel 1 as waveform output. */
#define TC_CHANNEL_WAVEFORM 1
#define ID_TC_WAVEFORM      ID_TC1
#define PIN_TC_WAVEFORM     PIN_TC0_TIOA1
#define PIN_TC_WAVEFORM_MUX PIN_TC0_TIOA1_MUX
```

Add these macros and function to your main application C-file:

```
#define TC_WAVEFORM_TIMER_SELECTION TC_CMR_TCCLKS_TIMER_CLOCK4
```

```
#define TC_WAVEFORM_DIVISOR     128
```

```
#define TC_WAVEFORM_FREQUENCY   178
```

```
#define TC_WAVEFORM_DUTY_CYCLE  30
```

```
* static void tc_waveform_initialize(void)
* {
*   uint32_t ra, rc;
*
*   // Configure the PMC to enable the TC module.
*   sysclk_enable_peripheral_clock(ID_TC_WAVEFORM);
*
*   // Init TC to waveform mode.
*   tc_init(TC, TC_CHANNEL_WAVEFORM,
*           TC_WAVEFORM_TIMER_SELECTION // Waveform Clock Selection
*           | TC_CMR_WAVE       // Waveform mode is enabled
*           | TC_CMR_ACPA_SET   // RA Compare Effect: set
*           | TC_CMR_ACPC_CLEAR // RC Compare Effect: clear
*           | TC_CMR_CPCTRG     // UP mode with automatic trigger on RC
Compare
*   );
*
*   // Configure waveform frequency and duty cycle.
*   rc = (sysclk_get_peripheral_bus_hz(TC) /
*           TC_WAVEFORM_DIVISOR /
*           TC_WAVEFORM_FREQUENCY;
*   tc_write_rc(TC, TC_CHANNEL_WAVEFORM, rc);
*   ra = (100 - TC_WAVEFORM_FREQUENCY_DUTY_CYCLE * rc / 100;
*   tc_write_ra(TC, TC_CHANNEL_WAVEFORM, ra);
*
*   // Enable TC TC_CHANNEL_WAVEFORM.
*   tc_start(TC, TC_CHANNEL_WAVEFORM);
* }
```

9.1.5.3. Workflow

1. Enable the TC module's waveform pin:

   ```
   ioport_set_pin_mode(PIN_TC_WAVEFORM, PIN_TC_WAVEFORM_MUX);
   ioport_disable_pin(PIN_TC_WAVEFORM);
   ```

2. Initialize the waveform channel to the following:
   - Output frequency of 178Hz, with a duty-cycle of 30%
   - Use TC_CMR_TCCLKS_TIMER_CLOCK4, with a divisor of 128

     ```
     tc_waveform_initialize();
     ```

9.2. Timer Counter Capture Waveform Example

9.2.1. Purpose

This example indicates how to use the Timer Counter in capture mode and waveform mode in order to measure the pulse frequency and count the total pulse number of an external signal injected into the device's TIOA pin.

9.2.2. Requirements

This package can be used with SAM4 evaluation kits such as the SAM4L EK, the SAM4L Xplained Pro, and other evaluation kits. Refer to the list of kits available for the actual device on http://www.atmel.com.

It generates a waveform on the Timer Counter pin PIN_TC_WAVEFORM, and it captures a waveform from pin PIN_TC_CAPTURE. Refer to the respective device evaluation kit's board file (conf_board.h) in order to examine these #defines.

Connect PIN_TC_WAVEFORM to PIN_TC_CAPTURE on the evaluation kit.

9.2.3. Description

This example shows how to configure the Timer Counter in both waveform and capture operating modes. In capture mode, a pulse signal (output from PIN_TC_WAVEFORM) is connected to PIN_TC_CAPTURE, and Register A (RA) and Register B (RB) will be loaded when their programmed event occurs.

When the Timer Counter interrupt occurs, its interrupt handler reads the RA and RB register values (for computing pulse frequency) and also increases the total pulse count number. The current pulse frequency and total pulse count number are both output via the debug UART.

The code can be roughly broken down as follows:

- Select pre-defined waveform frequency and duty cycle to be generated
- Configure TC_CHANNEL_WAVEFORM as waveform output
- Configure TC_CHANNEL_CAPTURE as a capture input
- Configure capture Register A (RA) to be loaded when a rising edge on TIOA occurs
- Configure capture Register B (RB) to be loaded when a falling edge on TIOA occurs
- Configure a Timer Counter interrupt and enable the RB load interrupt
- Pressing 'c' in the terminal window, starts capture
- Pressing 's' in the terminal window, stops capture and dumps the information captured so far

9.2.4. Main Files

- tc.c: Timer Counter driver
- tc.h: Timer Counter driver header file
- tc_capture_waveform_example.c: Timer Counter 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:

```
*     -- TC capture waveform example  xxx --
*     -- xxxxxx-xx
*     -- Compiled: xxx xx xxxx xx:xx:xx --
```

5. Choose the item from the following menu to test:

```
*     Menu :
*     ------
*       Output waveform property:
*       0: Set Frequency =  178 Hz, Duty Cycle = 30%
*       1: Set Frequency =  375 Hz, Duty Cycle = 50%
*       2: Set Frequency =  800 Hz, Duty Cycle = 75%
*       3: Set Frequency = 1000 Hz, Duty Cycle = 80%
*       4: Set Frequency = 4000 Hz, Duty Cycle = 55%
*       -------------------------------------------
*       c: Capture waveform from TC(TC_PERIPHERAL)
channel(TC_CHANNEL_CAPTURE)
*       s: Stop capture and display captured information
*       h: Display menu
*     ------
```

10. Document Revision History

Doc. Rev.	Date	Comments
42301B	07/2015	Updated title of application note and added list of supported devices
42301A	05/2014	Initial document release