APPLICATION NOTE

AT07906: SAM4 Pulse Width Modulation Controller (PWM)

ASF PROGRAMMERS MANUAL

SAM4 Pulse Width Modulation Controller (PWM)

This driver for SAM4E and SAM4S devices provides an interface for the configuration and management of the device's Pulse Width Modulation functionality.

The Pulse Width Modulation Controller has four independently controllable channels. Each channel controls two complementary square-wave outputs. The characteristics of the output waveform such as period, duty-cycle, polarity, and dead-times (also called dead-bands or non-overlapping times) are also configurable.

The outline of this documentation is as follows:

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

Table of Contents

Software License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.

4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

The PWM Controller has four independently controllable channels. Each channel controls two complementary square-wave outputs. The characteristics of the output waveform such as period, duty-cycle, polarity, and dead-times (also called dead-bands or non-overlapping times) are also configurable.

All PWM Controller channels integrate a double-buffering system in order to prevent an unexpected output waveform while modifying the period, the spread spectrum, the duty-cycle, the additional edge register, or the dead-times.

PWM channels can be linked together as synchronous channels, in order to be able to update their duty-cycle or dead-times at the same time. Synchronous channel duty cycle update can be performed by via a Peripheral DMA Controller (PDC) channel, which offers buffer transfer without processor Intervention.

Note The SAM4E PWM Controller includes a spread-spectrum counter to allow a constantly varying period (only for Channel 0). This counter may be useful in minimizing electromagnetic interference or reducing the acoustic noise of a PWM driven motor.

The PWM Controller provides eight independent comparison units, each capable of comparing a programmed value to the counter of the synchronous channels (counter of channel 0). These comparisons can be used to generate software interrupts, to trigger pulses on the two independent event lines (in order to synchronize ADC conversions with a lot of flexibility independently of the PWM outputs) and to trigger PDC transfer requests.

The PWM block provides a fault-protection mechanism with eight fault inputs, capable of detecting fault conditions and overriding the PWM outputs asynchronously (outputs forced to 0, 1, or high impedance).

3. Special Considerations

3.1 I/O Lines

The pins used for interfacing to the PWM are multiplexed with GPIO lines. The user application must first program the GPIO controller, in order to assign the desired PWM pins to their peripheral function. If any PWM I/O lines are not used by the target application, they can be used for other purposes by the GPIO controller.

3.2 Power Management

The PWM is not continuously clocked. The user application must first enable the PWM clock in the Power Management Controller (PMC) before using the PWM. Also, if the application does not require PWM operations, the PWM clock can be stopped whenever it is not needed and can be restarted at a later time. In this case, the PWM will resume its operations from where it left off.

3.3 Interrupt Sources

The PWM interrupt line is connected to one of the internal sources of the Nested Vectored Interrupt Controller (NVIC). Using the PWM interrupt requires that the NVIC be configured first.

Note It is recommended that the PWM interrupt line **not** be used in edge sensitive mode.

4. Extra Information

For extra information, see Extra Information for Pulse Width Modulation Controller. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

5. Examples

For a list of examples related to this driver, see Examples for SAM Pulse Width Modulation Controller (PWM).

6. API Overview

6.1 Variable and Type Definitions

6.1.1 Type pwm_ch_t

```
typedef enum _pwm_ch_t pwm_ch_t
```

PWM channel numbers.

6.2 Structure Definitions

6.2.1 Struct pwm_channel_t

PWM channel mode input parameter configuration.

Table 6-1. Members

Type	Name	Description
pwm_additional_edge_mode_t	additional_edge_mode	Additional Edge Mode (SAM4E devices only).
pwm_align_t	alignment	Channel alignment.
bool	b_deadtime_generator	Enable/disable channel dead-time generator (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
bool	b_pwmh_output_inverted	Enable/disable channel dead-time PWMH output inversion (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
bool	b_pwml_output_inverted	Enable/disable channel dead-time PWML output inversion (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
bool	b_sync_ch	Enable/disable Synchronous Channel (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
uint32_t	channel	Channel number.
pwm_counter_event_t	counter_event	Channel counter event (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
pwm_fault_id_t	fault_id	Channel fault ID (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
pwm_output_t	output_selection	Channel output
pwm_level_t	polarity	Channel initial polarity.
pwm_spread_spectrum_mode_t	spread_spectrum_mode	Spread spectrum mode (SAM4E devices only).
uint32_t	ul_additional_edge	Additional edge value (range 0 to 65535) (SAM4E devices only).

Type	Name	Description
uint32_t	ul_duty	Duty cycle value.
pwm_level_t	ul_fault_output_pwmh	Channel PWMH output level during fault protection (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
pwm_level_t	ul_fault_output_pwml	Channel PWML output level during fault protection (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
uint32_t	ul_period	Period cycle value.
uint32_t	ul_prescaler	Channel prescaler. Refer to the section entitled "CPRE: Channel Pre-scaler" in the device-specific datasheet for more information.
uint32_t	ul_spread	Spread spectrum value (range 0 to 65535) (SAM4E devices only).
uint16_t	us_deadtime_pwmh	PWMH output dead-time value (range 0 to 4095) (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).
uint16_t	us_deadtime_pwml	PWML output dead-time value (range 0 to 4095) (SAM3U/SAM3S/SAM3XA/SAM4S/SAM4E devices only).

6.2.2 Struct pwm_clock_t

Input parameters when initializing PWM.

Table 6-2. Members

Type	Name	Description
uint32_t	ul_clka	Frequency of clock A in Hz (set 0 to switch it off).
uint32_t	ul_clkb	Frequency of clock B in Hz (set 0 to switch it off).
uint32_t	ul_mck	Frequency of master clock in Hz.

6.2.3 Struct pwm_cmp_t

PWM comparison configuration structure.

Table 6-3. Members

Type	Name	Description
bool	b_enable	Enable/disable comparison unit.
bool	b_is_decrementing	Comparison mode.
bool	b_pulse_on_line_0	Enable/disable the match pulse event generation on PWM line 0.

Type	Name	Description
bool	b_pulse_on_line_1	Enable/disable the match pulse event generation on PWM line 1.
uint32_t	ul_period	Comparison period value.
uint32_t	ul_trigger	Comparison trigger value.
uint32_t	ul_update_period	Comparison update period value.
uint32_t	ul_value	Comparison value.
uint32_t	unit	Comparison unit number.

6.2.4 Struct pwm_fault_t

PWM fault input behavior configuration.

Table 6-4. Members

Type	Name	Description
bool	b_clear	Enable/disable fault flag clearing.
bool	b_filtered	Enable/disable fault filtering.
pwm_fault_id_t	fault_id	Fault ID.
pwm_level_t	polarity	Polarity of fault input.

6.2.5 Struct pwm_output_t

PWM channel output configuration.

Table 6-5. Members

Type	Name	Description
bool	b_override_pwmh	Enable/disable the PWMH output override.
bool	b_override_pwml	Enable/disable the PWML output override.
pwm_level_t	override_level_pwmh	The override output level for PWMH.
pwm_level_t	override_level_pwml	The override output level for PWML.

6.2.6 Struct pwm_protect_t

PWM write-protect information configuration.

Table 6-6. Members

Type	Name	Description
uint32_t	ul_hw_status	Bitmask of the PWM register groups for write-protect hardware status.
uint32_t	ul_offset	Offset address of the PWM register to which a write access has been attempted.

Type	Name	Description
uint32_t	ul_sw_status	Bitmask of PWM register groups for write-protect software status.

6.3 Macro Definitions

6.3.1 Macro PWM_INVALID_ARGUMENT

```
#define PWM_INVALID_ARGUMENT
```

Invalid argument error.

6.4 Function Definitions

6.4.1 Function pwm_channel_disable()

Disable the specified PWM channel.

```
void pwm_channel_disable(
  Pwm * p_pwm,
  uint32_t ul_channel)
```

Note A disabled PWM channel can be re-initialized using pwm_channel_init().

Table 6-7. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_channel	PWM channel number

6.4.2 Function pwm_channel_disable_interrupt()

Disable a PWM channel's counter event and fault protection interrupt.

```
void pwm_channel_disable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_event,
  uint32_t ul_fault)
```

Table 6-8. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_event	Channel number on which the counter event interrupt should be disabled.

Data direction	Parameter name	Description
[in]	ul_fault	Channel number on which the fault protection interrupt should be disabled (the parameter value is ignored by SAM3N/SAM4N/SAM4C devices).

6.4.3 Function pwm_channel_enable()

Enable the specified PWM channel.

```
void pwm_channel_enable(
  Pwm * p_pwm,
  uint32_t ul_channel)
```

Note The PWM channel should be initialized by pwm_channel_init() before it is enabled.

Table 6-9. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_channel	PWM channel number

6.4.4 Function pwm_channel_enable_interrupt()

Enable a PWM channel's counter event and fault protection interrupt.

```
void pwm_channel_enable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_event,
  uint32_t ul_fault)
```

Table 6-10. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_event	Channel number on which the counter event interrupt should be enabled.
[in]	ul_fault	Channel number on which the fault protection interrupt should be enabled (the parameter value is ignored by SAM3N/SAM4N/SAM4C devices).

6.4.5 Function pwm_channel_get_counter()

Get a PWM channel counter value.

```
uint32_t pwm_channel_get_counter(
  Pwm * p_pwm,
  pwm_channel_t * p_channel)
```

Table 6-11. Parameters

Data direction	Parameter name	Description
[in]	p_pwm	Module hardware register base address pointer
[in]	p_channel	Channel configuration structure pointer

Returns The PWM channel counter value.

6.4.6 Function pwm_channel_get_interrupt_mask()

Get the PWM channel counter event and fault protection trigger interrupt mask.

```
uint32_t pwm_channel_get_interrupt_mask(
  Pwm * p_pwm)
```

Table 6-12. Parameters

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

Returns The PWM channel counter event and fault protection trigger interrupt mask. Refer to the section called "pwm interrupt mask register" in the device-specific datasheet for more information.

6.4.7 Function pwm_channel_get_interrupt_status()

Get the PWM channel counter event, and fault protection trigger interrupt status.

```
uint32_t pwm_channel_get_interrupt_status(
  Pwm * p_pwm)
```

Table 6-13. Parameters

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

Returns The channel counter event, and fault protection trigger interrupt status. Refer to the section called "PWM Interrupt Status Register" in the device-specific datasheet for more information.

6.4.8 Function pwm_channel_get_status()

Check which PWM channel(s) are enabled.

```
uint32_t pwm_channel_get_status(
  Pwm * p_pwm)
```

Table 6-14. Parameters

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

Returns

The bitmask of enabled PWM channel(s). Refer to the section called "PWM Status Register" in the device-specific datasheet for further information.

6.4.9 Function pwm_channel_init()

Initialize a PWM channel.

```
uint32_t pwm_channel_init(
  Pwm * p_pwm,
  pwm_channel_t * p_channel)
```

Table 6-15. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in]	p_channel	Channel configuration structure pointer

Returns

The PWM channel initialization status.

Table 6-16. Return Values

Return value	Description
0	PWM channel was initialized correctly

6.4.10 Function pwm_channel_update_additional_edge()

Change a PWM channel's additional edge value and mode.

```
void pwm_channel_update_additional_edge(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  uint32_t ul_additional_edge,
  pwm_additional_edge_mode_t additional_edge_mode)
```

Note This function is only available on SAM4E devices.

Table 6-17. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in, out]	p_channel	Channel configuration structure pointer
[in]	ul_additional_edge	Additional edge value
[in]	additional_edge_mode	Additional edge mode

6.4.11 Function pwm_channel_update_dead_time()

Change a PWM channel's dead-time.

```
void pwm_channel_update_dead_time(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  uint16_t us_deadtime_pwmh,
  uint16_t us_deadtime_pwml)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[out]	p_channel	Channel configuration structure pointer
[in]	us_deadtime_pwmh	Dead-time value for PWMH output
[in]	us_deadtime_pwml	Dead-time value for PWML output

6.4.12 Function pwm_channel_update_duty()

Set the duty cycle of a PWM channel.

```
uint32_t pwm_channel_update_duty(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  uint32_t ul_duty)
```

Table 6-19. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in, out]	p_channel	"Channel configuration structure" pointer
[in]	ul_duty	Duty cycle value

Returns An indication of whether the PWM channel duty cycle was successfully changed.

Table 6-20. Return Values

Return value	Description
0	The PWM channel duty cycle was successfully changed
PWM_INVALID_ARGUMENT	Invalid parameter(s) in the channel configuration

6.4.13 Function pwm_channel_update_output()

Change a PWM channel output selection.

```
void pwm_channel_update_output(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  pwm_output_t * p_output,
  bool b_sync)
```

Table 6-21. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[out]	p_channel	Channel configuration structure pointer
[in]	p_output	PWM channel output selection
[in]	b_sync	Set to true to change the output synchronously (at the beginning of the next PWM period). Set to false to change the output asynchronously (at the end of the execution of the function).

6.4.14 Function pwm_channel_update_period()

Set the period of a PWM channel.

```
uint32_t pwm_channel_update_period(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  uint32_t ul_period)
```

Table 6-22. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in, out]	p_channel	Channel configuration structure pointer
[in]	ul_period	Period value

Returns An indication of whether the PWM channel period was successfully changed.

Table 6-23. Return Values

Return value	Description
0	The PWM channel period was successfully changed
PWM_INVALID_ARGUMENT	Invalid parameter(s) in the channel configuration

6.4.15 Function pwm_channel_update_polarity_mode()

Change a PWM channel's polarity mode.

```
void pwm_channel_update_polarity_mode(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  bool polarity_inversion_flag,
  pwm_level_t polarity_value)
```

Note This function is only available on SAM4E devices.

Table 6-24. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in, out]	p_channel	Channel configuration structure pointer
[in]	polarity_inversion_flag	Polarity inversion (true for inversion, false otherwise)
[in]	polarity_value	Polarity value

6.4.16 Function pwm_channel_update_spread()

Set a PWM channel's spread spectrum value.

```
void pwm_channel_update_spread(
  Pwm * p_pwm,
  pwm_channel_t * p_channel,
  uint32_t ul_spread)
```

Note This function is only available on SAM4E devices.

Table 6-25. Parameters

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

Data direction	Parameter name	Description
[out]	p_channel	Channel configuration structure pointer
[in]	ul_spread	Spread spectrum value

6.4.17 Function pwm_cmp_change_setting()

Change the settings for a PWM comparison unit.

```
uint32_t pwm_cmp_change_setting(
  Pwm * p_pwm,
  pwm_cmp_t * p_cmp)
```

Table 6-26. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in]	p_cmp	Comparison unit configuration structure

Returns The PWM comparison unit configuration change status.

Table 6-27. Return Values

Return value	Description
0	PWM comparison unit was configured correctly

6.4.18 Function pwm_cmp_disable_interrupt()

Disable the specified PWM comparison unit interrupt.

```
void pwm_cmp_disable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources,
  pwm_cmp_interrupt_t type)
```

Table 6-28. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Comparison unit number
[in]	type	Select a match or update interrupt

6.4.19 Function pwm_cmp_enable_interrupt()

Enable the specified PWM comparison unit interrupt.

```
void pwm_cmp_enable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources,
  pwm_cmp_interrupt_t type)
```

Table 6-29. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Comparison unit number
[in]	type	Select whether to match or update interrupts

6.4.20 Function pwm_cmp_get_period_counter()

Get the specified PWM comparison unit period counter.

```
uint32_t pwm_cmp_get_period_counter(
  Pwm * p_pwm,
  uint32_t ul_cmp_unit)
```

Table 6-30. Parameters

Data direction	Parameter name	Description
[in]	p_pwm	Module hardware register base address pointer
[in]	ul_cmp_unit	PWM comparison unit number.

Returns The PWM comparison unit period counter.

6.4.21 Function pwm_cmp_get_update_counter()

Get the specified PWM comparison unit update period counter.

```
uint32_t pwm_cmp_get_update_counter(
  Pwm * p_pwm,
  uint32_t ul_cmp_unit)
```

Table 6-31. Parameters

Data direction	Parameter name	Description
[in]	p_pwm	Module hardware register base address pointer
[in]	ul_cmp_unit	PWM comparison unit number

Returns The PWM comparison unit update period counter.

6.4.22 Function pwm_cmp_init()

Initialize a PWM comparison unit.

```
uint32_t pwm_cmp_init(
  Pwm * p_pwm,
  pwm_cmp_t * p_cmp)
```

Table 6-32. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	p_cmp	Comparison unit configuration structure

Returns The PWM comparison unit initialization status.

Table 6-33. Return Values

Return value	Description
0	PWM comparison unit was initialized correctly

6.4.23 Function pwm_disable_protect()

Disable the write protection of the PWM registers.

```
void pwm_disable_protect(
  Pwm * p_pwm,
  uint32_t ul_group)
```

Note Only a hardware reset of the PWM controller (handled by PMC) can disable hardware write protection.

Table 6-34. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_group	Bitmask of PWM register group

6.4.24 Function pwm_enable_protect()

Enable the PWM registers write protect.

```
void pwm_enable_protect(
  Pwm * p_pwm,
  uint32_t ul_group,
  bool b_sw)
```

Table 6-35. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_group	Bitmask of PWM register group
[in]	b_sw	Protection, true for software protection and false for hardware protection.

6.4.25 Function pwm_fault_clear_status()

Clear a PWM fault input.

```
void pwm_fault_clear_status(
  Pwm * p_pwm,
  pwm_fault_id_t id)
```

Table 6-36. Parameters

Data direction	Parameter name	Description
[in]	p_pwm	Module hardware register base address pointer
[in]	id	Fault ID

6.4.26 Function pwm_fault_get_input_level()

Get the PWM fault input level.

```
pwm_level_t pwm_fault_get_input_level(
  Pwm * p_pwm,
  pwm_fault_id_t id)
```

Table 6-37. Parameters

Data direction	Parameter name	Description
[in]	p_pwm	Module hardware register base address pointer
[in]	id	Fault ID

Returns The PWM fault input Level.

6.4.27 Function pwm_fault_get_status()

Get the PWM fault status.

```
uint32_t pwm_fault_get_status(
  Pwm * p_pwm)
```

Table 6-38. Parameters

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

Returns The bitmask of IDs for all currently active PWM faults.

6.4.28 Function pwm_fault_init()

Initialize the PWM fault input behavior.

```
uint32_t pwm_fault_init(
  Pwm * p_pwm,
  pwm_fault_t * p_fault)
```

Table 6-39. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in]	p_fault	Fault configuration structure pointer

Returns The fault input initialization status.

Table 6-40. Return Values

Return value	Description
0	Fault input behavior initialized correctly

6.4.29 Function pwm_get_interrupt_mask()

Get the PDC transfer, synchronous channels and comparison interrupt mask.

```
uint32_t pwm_get_interrupt_mask(
  Pwm * p_pwm)
```

Table 6-41. Parameters

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

Returns The PDC transfer, synchronous channels and comparison interrupt mask. Refer to the section called "PWM Interrupt Mask Register 2" in the device-specific datasheet.

6.4.30 Function pwm_get_interrupt_status()

Get the PDC transfer, synchronous channels and comparison interrupt status.

```
uint32_t pwm_get_interrupt_status(
  Pwm * p_pwm)
```

Table 6-42. Parameters

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

Returns

The PDC transfer, synchronous channels and comparison interrupt status. Refer to the section called "PWM Interrupt Status Register 2" in the device-specific datasheet.

6.4.31 Function pwm_get_protect_status()

Get the PWM write protection status.

```
bool pwm_get_protect_status(
  Pwm * p_pwm,
  pwm_protect_t * p_protect)
```

Table 6-43. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[out]	p_protect	Pointer to a PWM protect structure

Returns

The PWM write protection status.

Table 6-44. Return Values

Return value	Description
false	Write protection is disabled
true	Write protection is enabled

6.4.32 Function pwm_init()

Initialize the PWM source clock (clock A and clock B).

```
uint32_t pwm_init(
  Pwm * p_pwm,
  pwm_clock_t * clock_config)
```

Table 6-45. Parameters

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

Data direction	Parameter name	Description
[in]	clock_config	Pointer to a PWM clock configuration structure

Returns The initialization result.

Table 6-46. Return Values

Return value	Description
0	Initialization successful
PWM_INVALID_ARGUMENT	Invalid parameter(s) in the channel configuration

6.4.33 Function pwm_pdc_disable_interrupt()

Disable a PDC transfer interrupt.

```
void pwm_pdc_disable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources)
```

Table 6-47. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Bitmask of PWM PDC transfer interrupt sources

Where the input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
PWM_IDR2_ENDTX	Disable the PDC End of TX buffer interrupt
PWM_IDR2_TXBUFE	Disable the PDC TX buffer empty interrupt

6.4.34 Function pwm_pdc_enable_interrupt()

Enable a PDC transfer interrupt.

```
void pwm_pdc_enable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources)
```

Table 6-48. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Bitmask of PWM PDC transfer interrupt sources

Where the input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
PWM_IER2_ENDTX	Enable the PDC End of TX buffer interrupt
PWM_IER2_TXBUFE	Enable the PDC TX buffer empty interrupt

6.4.35 Function pwm_pdc_set_request_mode()

Set PDC transfer request mode.

```
void pwm_pdc_set_request_mode(
  Pwm * p_pwm,
  pwm_pdc_request_mode_t request_mode,
  uint32_t ul_cmp_unit)
```

Note

If the synchronous channels' update mode is PWM_SYNC_UPDATE_MODE_0 on page 35 or PWM_SYNC_UPDATE_MODE_1 on page 35, then input parameter *ul_pdc_request* will be ignored and the PDC transfer request will **never** occur.

Table 6-49. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	request_mode	PDC transfer request mode
[in]	ul_cmp_unit	PWM comparison unit number

6.4.36 Function pwm_stepper_motor_init()

Initialize the PWM stepper motor mode.

```
void pwm_stepper_motor_init(
  Pwm * p_pwm,
  pwm_stepper_motor_pair_t pair,
  bool b_enable_gray,
  bool b_down)
```

Table 6-50. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in]	pair	The two PWM channels used by the stepper motor.
[in]	b_enable_gray	Enable/disable gray count
[in]	b_down	Counter direction, true to count down, false to count up

6.4.37 Function pwm_sync_change_period()

Set the time period between each update of the PWM synchronous channels.

```
void pwm_sync_change_period(
  Pwm * p_pwm,
  uint32_t ul_update_period)
```

Table 6-51. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_update_period	Time between each update of the synchronous channels

6.4.38 Function pwm_sync_disable_interrupt()

Disable a synchronous channel interrupt.

```
void pwm_sync_disable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources)
```

Table 6-52. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Bitmask of PWM synchronous channels' interrupt sources

Where the input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
PWM_IDR2_WRDY	Disable the synchronous channels' Write Ready for update interrupt
PWM_IDR2_UNRE	Disable the Synchronous channels' update underrun error interrupt

6.4.39 Function pwm_sync_enable_interrupt()

Enable a PWM synchronous channel interrupt.

```
void pwm_sync_enable_interrupt(
  Pwm * p_pwm,
  uint32_t ul_sources)
```

Table 6-53. Parameters

Data direction	Parameter name	Description
[out]	p_pwm	Module hardware register base address pointer
[in]	ul_sources	Bitmask of PWM synchronous channel interrupt sources

Where the input parameter *ul_sources* is a bitmask containing one or more of the following:

Parameter Value	Description
PWM_IER2_WRDY	Enable the synchronous channels' Write Ready for update interrupt
PWM_IER2_UNRE	Enable the Synchronous channels' update underrun error interrupt

6.4.40 Function pwm_sync_get_period_counter()

Get the PWM synchronization update period counter.

```
uint32_t pwm_sync_get_period_counter(
  Pwm * p_pwm)
```

Table 6-54. Parameters

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

Returns The PWM synchronization update Period Counter.

6.4.41 Function pwm_sync_init()

Initialize the PWM synchronous channels' update mode and period.

```
uint32_t pwm_sync_init(
  Pwm * p_pwm,
  pwm_sync_update_mode_t mode,
  uint32_t ul_update_period)
```

Table 6-55. Parameters

Data direction	Parameter name	Description
[in, out]	p_pwm	Module hardware register base address pointer
[in]	mode	Synchronous channels update mode
[in]	ul_update_period	Time between each update of the synchronous channel

Returns The PWM channel initialization status.

Table 6-56. Return Values

Return value	Description
0	PWM channel was initialized correctly

6.4.42 Function pwm_sync_unlock_update()

Unlock the synchronous channels update.

```
void pwm_sync_unlock_update(
  Pwm * p_pwm)
```

Note After being unlocked the synchronous channels will be updated at the next PWM period.

Table 6-57. Parameters

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

6.5 Enumeration Definitions

6.5.1 Enum _pwm_ch_t

PWM channel numbers.

Table 6-58. Members

Enum value	Description
PWM_CHANNEL_0	PWM channel 0.
PWM_CHANNEL_1	PWM channel 1.
PWM_CHANNEL_2	PWM channel 2.
PWM_CHANNEL_3	PWM channel 3.

6.5.2 Enum pmc_cmp_unit_t

PWM comparison unit.

Table 6-59. Members

Enum value	Description
PWM_CMP_UNIT_0	PWM comparison unit 0.
PWM_CMP_UNIT_1	PWM comparison unit 1.
PWM_CMP_UNIT_2	PWM comparison unit 2.
PWM_CMP_UNIT_3	PWM comparison unit 3.
PWM_CMP_UNIT_4	PWM comparison unit 4.
PWM_CMP_UNIT_5	PWM comparison unit 5.
PWM_CMP_UNIT_6	PWM comparison unit 6.
PWM_CMP_UNIT_7	PWM comparison unit 7.

6.5.3 Enum pwm_additional_edge_mode_t

PWM additional edge (SAM4E devices only).

Table 6-60. Members

Enum value	Description
PWM_ADDITIONAL_EDGE_MODE_INC	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, and the counter of channel x is incrementing.
PWM_ADDITIONAL_EDGE_MODE_DEC	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, and the counter of channel x is incrementing.
PWM_ADDITIONAL_EDGE_MODE_BOTH	The additional edge of the channel x output waveform occurs when CCNTx reaches ADEDGVUP, whether the counter is incrementing or not.

6.5.4 Enum pwm_align_t

PWM channel alignment.

Table 6-61. Members

Enum value	Description
PWM_ALIGN_LEFT	The period is left aligned.
PWM_ALIGN_CENTER	The period is center aligned.

6.5.5 Enum pwm_cmp_interrupt_t

PWM comparison interrupt.

Table 6-62. Members

Enum value	Description
PWM_CMP_MATCH	Comparison unit match interrupt.
PWM_CMP_UPDATE	Comparison unit update interrupt.

6.5.6 Enum pwm_counter_event_t

PWM event.

Table 6-63. Members

Enum value	Description
PWM_EVENT_PERIOD_END	The channel counter event occurs at the end of the PWM period.
PWM_EVENT_PERIOD_HALF_END	The channel counter event occurs halfway through the PWM period.

6.5.7 Enum pwm_fault_id_t

PWM fault input ID.

Table 6-64. Members

Enum value	Description
PWM_FAULT_PWMFI1	
PWM_FAULT_MAINOSC	Main oscillator fault.
PWM_FAULT_ADC	ADC fault.
PWM_FAULT_ACC	Analog Comparator fault.
PWM_FAULT_TIMER_0	Timer 0 fault.
PWM_FAULT_TIMER_1	Timer 1 fault.

6.5.8 Enum pwm_level_t

PWM level.

Table 6-65. Members

Enum value	Description
PWM_LOW	Low level.
PWM_HIGH	High level.
PWM_HIGHZ	High Impedance (SAM4E only).

6.5.9 Enum pwm_pdc_interrupt_t

PWM PDC transfer interrupt.

Table 6-66. Members

Enum value	Description
PWM_PDC_TX_END	PDC transmit end.
PWM_PDC_TX_EMPTY	PDC transmit buffer empty.

6.5.10 Enum pwm_pdc_request_mode_t

PWM PDC transfer request mode.

Table 6-67. Members

Enum value	Description
PWM_PDC_UPDATE_PERIOD_ELAPSED	PDC transfer request is set as soon as the update period elapses.
PWM_PDC_COMPARISON_MATCH	PDC transfer request is set as soon as the selected comparison matches.

6.5.11 Enum pwm_protect_reg_group_t

PWM write-protect register groups.

Table 6-68. Members

Enum value	Description
PWM_GROUP_CLOCK	The write protect clock register.
PWM_GROUP_DISABLE	The write protect disable register.
PWM_GROUP_MODE	The write protect mode registers.
PWM_GROUP_PERIOD	The write protect period registers.
PWM_GROUP_DEAD_TIME	The write protect read time registers.
PWM_GROUP_FAULT	The write protect fault registers.

6.5.12 Enum pwm_spread_spectrum_mode_t

PWM Spread-Spectrum mode (SAM4E devices only).

Table 6-69. Members

Enum value	Description
PWM_SPREAD_SPECTRUM_MODE_TRIANGULAR	The spread-spectrum counter starts to count from -SPRD when the channel 0 is enabled and counts upwards at each PWM period. When it reaches +SPRD, it restarts counting from -SPRD again.
PWM_SPREAD_SPECTRUM_MODE_RANDOM	The spread-spectrum counter is loaded with a new random value at each PWM period. This random value is uniformly distributed, and is between -SPRD and +SPRD.

6.5.13 Enum pwm_stepper_motor_pair_t

PWM channels used by motor stepper.

Table 6-70. Members

Enum value	Description
PWM_STEPPER_MOTOR_CH_0_1	Channels 0 and 1.
PWM_STEPPER_MOTOR_CH_2_3	Channels 2 and 3.

6.5.14 Enum pwm_sync_interrupt_t

PWM synchronous channels interrupt.

Table 6-71. Members

Enum value	Description
PWM_SYNC_WRITE_READY	Write Ready for synchronous channels update.
PWM_SYNC_UNDERRUN	Synchronous channels Update Underrun Error.

6.5.15 Enum pwm_sync_update_mode_t

PWM synchronous channels update mode.

Table 6-72. Members

Enum value	Description
PWM_SYNC_UPDATE_MODE_0	Manual write of double-buffer registers and manual update of synchronous channels.
PWM_SYNC_UPDATE_MODE_1	Manual write of double-buffer registers and automatic update of synchronous channel.
PWM_SYNC_UPDATE_MODE_2	Automatic write of duty-cycle update registers by the PDC and automatic update of synchronous channel.

7. Extra Information for Pulse Width Modulation Controller

7.1 Acronyms

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

Acronym	Definition
ADC	Analog to Digital Converter
GPIO	General Purpose Input Output
ID	Identity
NVIC	Nested Vectored Interrupt Controller
PDC	Peripheral DMA Controller
PMC	Power Manager Controller
PWMH	Pulse Width Modulation High
PWML	Pulse Width Modulation Low
QSG	Quick Start Guide

7.2 Dependencies

This driver has the following dependencies:

- None

7.3 Errata

There are no errata related to this driver.

7.4 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

8. Examples for SAM Pulse Width Modulation Controller (PWM)

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4 Pulse Width Modulation Controller (PWM). 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 SAM PWM module
- Pulse Width Modulator Controller - Example controlling an LED
- Pulse Width Modulator Controller - Example synchronous channel LED control

8.1 Quick Start guide for SAM PWM module

This is the quick start guide for the SAM4 Pulse Width Modulation Controller (PWM), with step-by-step instructions on how to configure and use the drivers in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, for example, the main application function.

- Basic Use Case
- Advanced Use Case

8.1.1 Basic Use Case

In this basic use case, the PWM Controller module is configured as follows:

- Output a square-wave on PWM channel 0
- The frequency of the square-wave is 1kHz with a 50% duty cycle
- Clock A is used as the source clock
- The output wave can be verified on the device's selected output pin

8.1.2 Setup Steps

8.1.2.1 Prerequisites

- Power Management Controller (PMC) driver
- General Purpose I/O (GPIO) driver

8.1.2.2 Example Code

Add the following PWM initialization code segments to the beginning of your main application function:

```
// PWM frequency in Hz.
#define PWM_FREQUENCY      1000
// Period value of PWM output waveform.
#define PERIOD_VALUE       100
```

```
  // Initial duty cycle value.
  #define INIT_DUTY_VALUE    50
```

```
pwm_channel_t g_pwm_channel_led;
```

```
pmc_enable_periph_clk(ID_PWM);
```

```
pwm_channel_disable(PWM, PWM_CHANNEL_0);
```

```
// Set PWM clock A as PWM_FREQUENCY*PERIOD_VALUE (clock B is not used).
pwm_clock_t clock_setting = {
    .ul_clka = PWM_FREQUENCY * PERIOD_VALUE,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);
```

```
g_pwm_channel_led.channel = PWM_CHANNEL_0;
```

```
//  eriod is left-aligned.
g_pwm_channel_led.alignment = PWM_ALIGN_LEFT;
// Output waveform starts at a low level.
g_pwm_channel_led.polarity = PWM_LOW;
// Use PWM clock A as source clock.
g_pwm_channel_led.ul_prescaler = PWM_CMR_CPRE_CLKA;
// Period value of output waveform.
g_pwm_channel_led.ul_period = PERIOD_VALUE;
// Duty cycle value of output waveform.
g_pwm_channel_led.ul_duty = INIT_DUTY_VALUE;
pwm_channel_init(PWM, &g_pwm_channel_led);
```

8.1.2.3 Workflow

1. Define the PWM channel instance, in order to configure channel 0:

```
pwm_channel_t g_pwm_channel_led;
```

2. Enable the module clock for the PWM peripheral:

```
pmc_enable_periph_clk(ID_PWM);
```

3. Disable PWM channel 0:

```
pwm_channel_disable(PWM, PWM_CHANNEL_0);
```

4. Set up the clock for PWM module:

```
// Set PWM clock A as PWM_FREQUENCY*PERIOD_VALUE (clock B is not used).
pwm_clock_t clock_setting = {
    .ul_clka = PWM_FREQUENCY * PERIOD_VALUE,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);
```

Note

- Only clock A is configured (clock B is not used)
- The expected frequency is 1kHz.

5. Initialize the channel instance and configure PWM channel 0, selecting clock A as its source clock and setting the duty cycle at 50%:

```
g_pwm_channel_led.channel = PWM_CHANNEL_0;
```

```
//  eriod is left-aligned.
g_pwm_channel_led.alignment = PWM_ALIGN_LEFT;
// Output waveform starts at a low level.
g_pwm_channel_led.polarity = PWM_LOW;
// Use PWM clock A as source clock.
g_pwm_channel_led.ul_prescaler = PWM_CMR_CPRE_CLKA;
// Period value of output waveform.
g_pwm_channel_led.ul_period = PERIOD_VALUE;
// Duty cycle value of output waveform.
g_pwm_channel_led.ul_duty = INIT_DUTY_VALUE;
pwm_channel_init(PWM, &g_pwm_channel_led);
```

Note

- The period is left-aligned and the output waveform starts at a low level.
- After setting each channel's parameters, the g_pwm_channel_led structure can be reused to configure other PWM channels.

8.1.3 Usage Steps

8.1.3.1 Example Code

Add the following code to, for example, the main loop in your application C-file:

```
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.1.3.2 Workflow

Enable PWM channel 0 and output a square-wave on this channel:

```
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.1.4 Advanced Use Case

In this use case, the PWM Controller module is configured as follows:

- Output a square-wave on PWM channel 0
- The frequency of the square-wave is 1kHz
- The duty cycle is changed in the PWM's ISR
- Clock A is used as the source clock
- The output wave can be verified on the device's selected output pin

8.1.5 Setup Steps

8.1.5.1 Prerequisites

- Power Management Controller (PMC) driver
- General Purpose I/O (GPIO) Management driver

8.1.5.2 Example Code

Add the following code segments to your application C-file:

```
pwm_channel_t pwm_channel_instance; *
```

```
void PWM_Handler(void)
{
    static uint32_t ul_duty = 0;
    uint32_t ul_status;
    static uint8_t uc_countn = 0;
    static uint8_t uc_flag = 1;

    ul_status = pwm_channel_get_interrupt_status(PWM);
    if ((ul_status & PWM_CHANNEL_0) == PWM_CHANNEL_0) {
        uc_count++;
        if (uc_count == 10) {
            if (uc_flag) {
                ul_duty++;
                if (ul_duty == 100) {
                    uc_flag = 0;
                }
            } else {
                ul_duty--;
                if (ul_duty == 0) {
                    uc_flag = 1;
                }
            }
            uc_count = 0;
            pwm_channel_instance.channel = PWM_CHANNEL_0;
            pwm_channel_update_duty(PWM, &pwm_channel_instance, ul_duty);
        }
    }
}
```

```
pmc_enable_periph_clk(ID_PWM);

pwm_channel_disable(PWM, PWM_CHANNEL_0);

pwm_clock_t clock_setting = {
    .ul_clka = 1000 * 100,
    .ul_clkb = 0,
    .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);

pwm_channel_instance.ul_prescaler = PWM_CMR_CPRE_CLKA;
pwm_channel_instance.ul_period = 100;
pwm_channel_instance.ul_duty = 0;
pwm_channel_instance.channel = PWM_CHANNEL_0;
pwm_channel_init(PWM, &pwm_channel_instance);
```

```
    pwm_channel_enable_interrupt(PWM, PWM_CHANNEL_0, 0);
```

8.1.5.3 Workflow

1. Declare the PWM channel instance in order to configure channel 0:

```
pwm_channel_t pwm_channel_instance;
```

2. Declare the PWM interrupt handler function in the application:

```
void PWM_Handler(void);
```

3. In the function PWM_Handler(), get the PWM interrupt status:

```
ul_status = pwm_channel_get_interrupt_status(PWM);
```

4. In the function PWM_Handler(), check if the PWM channel 0 interrupt has occurred:

```
      if ((ul_status & PWM_CHANNEL_0) == PWM_CHANNEL_0) {
      }
```

5. In the function PWM_Handler(), if the PWM channel 0 interrupt has occurred then update the ul_duty value:

```
      uc_count++;
      if (uc_count == 10) {
         if (uc_flag) {
             ul_duty++;
             if (ul_duty >= 100) {
                 uc_flag = 0;
             }
         } else {
             ul_duty--;
             if (ul_duty == 0) {
                 uc_flag = 1;
             }
         }
     }
```

6. In the function PWM_Handler(), if the ul_duty value has been updated then change the square-wave duty:

```
  pwm_channel_instance.channel = PWM_CHANNEL_0;
  pwm_channel_update_duty(PWM, &pwm_channel_instance, ul_duty);
```

7. Enable the PWM clock:

```
pmc_enable_periph_clk(ID_PWM);
```

8. Disable the PWM channel 0:

```
pwm_channel_disable(PWM, PWM_CHANNEL_0);
```

9. Setup the clock for the PWM Controller module:

```
* pwm_clock_t clock_setting = {
```

```
    .ul_clka = 1000 * 100,
   .ul_clkb = 0,
   .ul_mck = sysclk_get_cpu_hz()
};
pwm_init(PWM, &clock_setting);
```

Note

- Only Clock A is configured (clock B is not used).
- The expected output frequency is 1kHz.

10. Initialize the channel structure and configure PWM channel 0, selecting clock A as its source clock and setting the initial duty cycle as 0%:

```
pwm_channel_instance.ul_prescaler = PWM_CMR_CPRE_CLKA;
pwm_channel_instance.ul_period = 100;
pwm_channel_instance.ul_duty = 0;
pwm_channel_instance.channel = PWM_CHANNEL_0;
pwm_channel_init(PWM, &pwm_channel_instance);
```

Note

- The period is left-aligned and the output waveform starts at a low level.
- The g_pwm_channel_instance structure can be reused to configure other PWM channels, after setting each channel's parameters.

11. Enable the PWMchannel 0 interrupt:

```
pwm_channel_enable_interrupt(PWM, PWM_CHANNEL_0, 0);
```

Note

- To enable the PWM interrupt, the NVIC must be configured to enable the PWM Controller interrupt.
- When the channel 0 counter reaches the channel period, the interrupt (counter event) will occur.

8.1.6 Usage Steps

8.1.6.1 Example Code

Add the following code to, for example, the main loop in your application C-file:

```
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.1.6.2 Workflow

Enable PWM channel 0, and output square-wave on this channel:

```
pwm_channel_enable(PWM, PWM_CHANNEL_0);
```

8.2 Pulse Width Modulator Controller - Example controlling an LED

8.2.1 Purpose

This example demonstrates the simple configuration of two PWM channels to generate signals with a variable duty cycle. The brightness of the two LEDs on the evaluation kit will vary repeatedly.

8.2.2 Requirements

This example can be used on any SAM3/4 evaluation kits (except SAM4L).

The two required LEDs need to be connected to PWM output pins, otherwise consider verifying the PWM output signals using an oscilloscope.

8.2.3 Main Files

- pwm.c: Pulse Width Modulator Controller driver
- pwm.h: Pulse Width Modulator Controller driver header file
- pwm_led_example.c: Pulse Width Modulator example application

8.2.4 Compilation Information

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

8.2.5 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:

```
-- PWM LED Example --
-- xxxxxx-xx --
-- Compiled: xxx xx xxxx xx:xx:xx --
```

5. The example code then performs the following:
   - Initialize the system clock and pin setting on the evaluation kit
   - Initialize the PWM clock
   - Configure the PIN_PWM_LED0_CHANNEL
   - Configure the PIN_PWM_LED1_CHANNEL
   - Enable the interrupt on counter event for the PIN_PWM_LED0_CHANNEL and PIN_PWM_LED1_CHANNEL channels
   - Change the duty cycle in the ISR

8.3 Pulse Width Modulator Controller - Example synchronous channel LED control

8.3.1 Purpose

This example demonstrates the simple configuration of two PWM synchronous channels to generate variable duty cycle signals. The duty cycle values are updated automatically by the Peripheral DMA Controller (PDC), which makes two of the on-board LEDs glow repeatedly.

8.3.2 Requirements

This example can be used on any SAM3/4 evaluation kits (except SAM4L).

The two required LEDs need to be connected to PWM output pins, otherwise consider verifying the PWM output signals using an oscilloscope.

8.3.3 Main Files

- pwm.c: Pulse Width Modulator Controller driver
- pwm.h: Pulse Width Modulator Controller driver header file
- pwm_sync_example.c: Pulse Width Modulator example application

8.3.4 Compilation Information

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

8.3.5 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:

```
-- PWM SYNC Example --
-- xxxxxx-xx --
-- Compiled: xxx xx xxxx xx:xx:xx --

===============================================================
Menu: press a key to change the configuration.
===============================================================
  u : Change update period for synchronous channels
  d : Change dead time of PWM outputs
  o : Enable/disable output override
```

5. The example code then performs the following:

- Initialize the system clock and the GPIO pins
- Initialize the PWM clock
- Configure the PIN_PWM_LED0_CHANNEL
- Configure the PIN_PWM_LED1_CHANNEL
- Configure the PDC transfer for PWM duty cycle update
- Enable the PDC TX interrupt and PIN_PWM_LED0_CHANNEL
- Update the synchronous period, dead time and override output via UART console
- Restart the PDC transfer in the ISR

Index

E

F

M

S

T

Document Revision History

Doc. Rev.	Date	Comments
42294A	05/2014	Initial document release

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