HB0113
Handbook
CorePWM v4.5

a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

50200113. 9.0 9/20

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 9.0

Added PolarFire® SoC support.

1.2 Revision 8.0

The following is a summary of the changes made in this revision.

- Added RTG4 information in Supported Familiesand Utilization and Performance Tables.
- Updated core version to 4.4 in Core Version.

1.3 Revision 7.0

The following is a summary of the changes made in this revision.

- Updated Table 10, page 12 (SAR 63930).
- Updated Tool Flows, page 23 (SAR 64045).

1.4 Revision 6.0

The following is a summary of the changes made in this revision.

- Added RTG4 information in Supported Families, page 3 and Utilization and Performance Tables.
- Updated core version to 4.3 in Core Version.

1.5 Revision 5.0

The following is a summary of the changes made in this revision.

- Added SmartFusion, SmartFusion2, and IGLOO2 families in Supported Families, page 3 (SAR 57616).
- Changed core version from 4.1 to 4.2 in “Core Version” section (SAR 57616).

1.6 Revision 4.0

Updated Figure 2 and Table 10 (SAR 57175).

1.7 Revision 3.0

Updated core version to v4.2.

1.8 Revision 2.0

Updated core version to v4.1.

1.9 Revision 1.0

The following is a summary of the changes made in this revision.

- Updated Utilization and Performance tables.
- Added low-cost TACHOMETER solution with up to 16 digital inputs.
- Added center-aligned PWM support.
- Updated tool flow to support Libero IDE v8.5 and SmartDesign.
- Updated Figure 3, Figure 5, and Figure 6 in the Configuration Example, page 20.
- Updated Figure 14.
- Updated the Software Driver section.

2 Introduction

2.1 Core Overview

2.1.1 Intended Use

CorePWM is a general purpose, multi-channel pulse width modulator (PWM) module for motor control, tone generation, battery charging, heating elements, and more.

In General Purpose PWM mode, duty cycle updates can be performed asynchronously or synchronously, based on parameter selection. In synchronous mode, all channels are updated at the beginning of the PWM period, which is useful for motor control and can be used to keep a constant dead band space between channel waveforms. Asynchronous mode is relevant to applications such as LED control, where synchronous updates are not required. Asynchronous mode lowers the area size, reducing shadow register requirements.

In addition to the general purpose PWM modes, there is a “Low Ripple DAC” mode that creates a minimum period pulse train whose High/Low average is that of the chosen duty cycle. When used with a low-pass filter (such as a simple RC circuit), a DAC can be created with far better bandwidth and ripple performance than a standard PWM algorithm can achieve. This type of DAC is ideally suited for fine tuning of power supply output levels.

CorePWM also provides support for tachometer monitoring of 3- and 4-wire fans. Incoming tachometer data is read by the firmware through the APB interface to calculate fan speed.

2.1.2 Key Features

- Configuration updates for all channels can be synchronized to the beginning of the PWM period, allowing precise updates and maintaining phase alignments between channels
- Configurable resolution based on the APB bus width
- Low-cost PWM solution with up to 16 separate PWM digital outputs, configurable via a register interface
- For DAC applications: Optional, per-channel Low Ripple DAC mode, allowing for greater resolution output of a given filter
- Low-cost TACHOMETER solution with up to 16 separate TACH digital inputs, configurable via a register interface
- All PWM outputs are double-edge-controlled
- Per-channel fixed register option for lower tile count
- Edge control based on a configurable PWM period with prescaler value and 0% to 100% duty cycle capability
- Set High, set Low, and Toggle Edge-Control modes
- Can be programmed on-the-fly from a Microcontroller, such as Core8051s, CoreABC, or the Fusion backbone
- Can be used to perform open or closed-loop margining of power supplies

2.2 Supported Families

- PolarFire® SoC
- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2
- IGLOO®/e
- ProASIC®3/E/L
- Fusion
- ProASICPLUS®
- Axcelerator®
- RTAX-S
- RTAX-DSP
- SmartFusion®

2.3 Core Version

This handbook supports CorePWM version 4.5.

2.4 Supported Interfaces

CorePWM is available with an APB interface, which is described in the APB Interface Timing, page 22.

2.5 Utilization and Performance

CorePWM has been implemented in several of Microsemi's device families. A summary of various implementation data is listed in the following tables (using standard speed grades).

As shown in Table 1 through Table 8, it is recommended to fix all registers that are not used, via parameters, to ensure optimal synthesis tile reduction.

Table 1 • **CorePWM Device Utilization and Performance (one 8-bit DAC channel configuration)**

	Logical Elements			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	20	96	116	AGLE600	1.0%	72
ProASIC3/E	20	76	96	A3P250	2.0%	96
Fusion	20	76	96	AFS600	1.0%	101
ProASICPLUS	20	102	122	APA300	1.0%	101
Axcelerator	20	31	51	AX250	1.0%	252
RTAX-S	20	31	51	RTAX250S	1.0%	223
SmartFusion	20	78	98	A2F500M3G	0.85%	180
SmartFusion2	20	30	50	M2S150T	0.03%	250
IGLOO2	20	30	50	M2GL150T	0.03%	250
RTG4	20	40	60	RT4G150	0.04%	250
PolarFire	20	30	50	PA5M500	0.012%	400

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 1, APB_DWIDTH = 8; DAC_MODE1 = 1 (DAC mode), FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 0, FIXED_PERIOD_EN = 1, FIXED_PERIOD = 1, SHADOW_REG_EN1 = 0.

Table 2 • **CorePWM Device Utilization and Performance (one 16-bit DAC channel configuration)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	54	147	201	AGLE600V2	2.0%	59
ProASIC3/E	54	111	165	A3P250	3.0%	85
Fusion	54	111	165	AFS600	1.0%	94
ProASICPLUS	55	209	264	APA200	3.0%	74
Axcelerator	57	53	110	AX250	2.0%	210
RTAX-S	57	53	110	RTAX250S	1.0%	176
SmartFusion	86	140	226	A2F500M3G	2.0%	132
SmartFusion2	53	68	121	M2S150T	0.09%	245
IGLOO2	53	68	121	M2GL150T	0.09%	245
RTG4	69	75	144	RT4G150	0.1%	227
PolarFire	54	48	102	PA5M500	0.025%	400

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 1, APB_DWIDTH = 16; DAC_MODE1 = 1 (DAC mode), FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 0, FIXED_PERIOD_EN = 1, FIXED_PERIOD = 1, SHADOW_REG_EN1 = 1.

Table 3 • **CorePWM Device Utilization and Performance (one 8-bit general purpose PWM channel configuration)**

	Tiles			Utilization		Performance
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**(MHz)**
IGLOO/e	15	55	70	AGLE600V2	1.0%	82
ProASIC3/E	15	55	70	A3P250	1.0%	130
Fusion	15	55	70	AFS600	1.0%	144
ProASICPLUS	15	58	73	APA300	1.0%	141
Axcelerator	16	40	56	AX250	1.0%	181
RTAX-S	16	40	56	RTAX250S	1.0%	187
SmartFusion	15	60	75	A2F500M3G	0.65%	211
SmartFusion2	23	45	68	M2S150T	0.05%	250
IGLOO2	23	45	68	M2GL150T	0.05%	250
RTG4	21	58	79	RT4G150	0.05%	238
PolarFire	15	36	51	PA5M500	0.012%	400

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 1, APB_DWIDTH = 8; DAC_MODE1 = 0 (general purpose PWM mode), FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 0, FIXED_PERIOD_EN = 1, FIXED_PERIOD = 8, FIXED_PWM_POS_EN1 = 1,

FIXED_PWM_POSEDGE1 = 0, FIXED_PWM_NEG_EN1 = 0, FIXED_PWM_NEGEDGE1 = 0, SHADOW_REG_EN1 = 0.

.

Table 4 • **CorePWM Device Utilization and Performance (one 16-bit general purpose PWM channel configuration)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	91	272	363	AGLE600V2	3.0%	46
ProASIC3/E	91	275	366	A3P250	6.0%	72
Fusion	91	275	366	AFS600	3.0%	79
ProASICPLUS	127	482	609	APA300	6.0%	59
Axcelerator	93	145	238	AX250	6.0%	110
RTAX-S	93	145	238	RTAX250S	6.0%	88
SmartFusion	90	324	414	A2F500M3G	3.6%	110
SmartFusion2	100	150	350	M2S150T	0.17%	170
IGLOO2	100	150	350	M2GL150T	0.17%	170
RTG4	134	166	300	RT4G150	0.2%	182
PolarFire	91	136	227	PA5M500	0.052%	350

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 1, APB_DWIDTH = 16; DAC_MODE1 = 0 (General Purpose PWM mode); SHADOW_REG_EN1 = 1, FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 64.

Table 5 • **CorePWM Device Utilization and Performance (8-bit multiple-output configuration example: 3 DAC mode outputs without shadow update register)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	58	208	266	AGLE600V2	2.0%	76
ProASIC3/E	58	150	208	A3P250	3.0%	101
Fusion	58	150	208	AFS600	2.0%	109
ProASICPLUS	58	280	338	APA300	4.0%	101
Axcelerator	58	66	124	AX250	2.0%	250
RTAX-S	58	66	124	RTAX250S	2.0%	217
SmartFusion	66	159	225	A2F500M3G	2.0%	157
SmartFusion2	66	72	138	M2S150T	0.1%	250
IGLOO2	66	72	138	M2GL150T	0.1%	250
RTG4	66	77	143	RT4G150	0.09%	250
PolarFire	58	64	112	PA5M500	0.03%	400

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 3, APB_DWIDTH = 8; DAC_MODE1, DAC_MODE2, and DAC_MODE3 = 1 (DAC Mode) FIXED_PERIOD_EN = 1, FIXED_PERIOD = 1, FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 0, SHADOW_REG_EN1 = 0, SHADOW_REG_EN2 = 0, SHADOW_REG_EN3 = 0.

Table 6 • **CorePWM Device Utilization and Performance (12-bit multiple-output configuration example: 3 DAC mode outputs, 3 general purpose PWM mode outputs)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	212	723	935	AGLE600V2	7.0%	45
AGLE600V2	212	694	906	A3P250	15.0%	45
ProASIC3/E	212	694	906	A3P250	15.0%	74
Fusion	212	694	906	AFS600	7.0%	82
ProASICPLUS	229	1054	1,283	APA300	16.0%	67
Axcelerator	216	307	523	AX250	12.0%	103
RTAX-S	216	307	523	RTAX250S	12.0%	87
SmartFusion	148	396	552	A2F500M3G	4.8%	132
SmartFusion2	176	274	450	M2S150T	0.31%	215
IGLOO2	176	274	450	M2GL150T	0.31%	215
RTG4	254	339	593	RT4G150	0.39%	114
PolarFire	212	309	521	PA5M500	0.1%	350

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: PWM_NUM = 6, APB_DWIDTH = 16; DAC_MODE1, DAC_MODE2, and DAC_MODE3 = 1 (DAC mode),
DAC_MODE4, DAC_MODE5, and DAC_MODE6 = 0 (general purpose PWM mode),
FIXED_PRESCALE_EN = 1, FIXED_PRESCALE = 8, SHADOW_REG_EN1 = 0,
SHADOW_REG_EN2 = 0, SHADOW_REG_EN3 = 0,
SHADOW_REG_EN3 = 0, SHADOW_REG_EN4 = 0,
SHADOW_REG_EN5 = 0, SHADOW_REG_EN6 = 0.

Table 7 • **CorePWM Device Utilization and Performance (one 16-bit general purpose PWM channel and one TACH input configuration)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	314	768	082	AGLE600V2	8.0%	47
ProASIC3/E	314	768	082	A3P250	18.0%	75
Fusion	314	768	1,082	AF600	8.0%	83
ProASICPLUS	328	1,043	1,371	APA300	17.0%	70
Axcelerator	319	483	802	AX250	19.0%	103
RTAX-S	319	483	802	RTAX250S	19.0%	87
SmartFusion	197	613	810	A2F500M3G	7.1%	115
SmartFusion2	252	416	668	M2S150T	0.45%	137
IGLOO2	252	416	668	M2GL150T	0.45%	137
RTG4	329	436	765	RT4G150	0.51%	164
PolarFire	314	497	811	PA5M500	0.15%	340

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: CONFIG_MODE = 1, PWM_NUM=3, TACH_NUM=3,APB_DWIDTH=16; DAC_MODE1=0 (General Purpose PWM mode), FIXED_PRESCALE_EN=1, FIXED_PRESCALE=8, FIXED_PERIOD_EN=0, FIXED_PWM_POS_EN1=1, FIXED_PWM_POSEDGE1=0, FIXED_PWM_NEG_EN1=0, FIXED_PWM_NEGEDGE1=0, SHADOW_REG_EN1=0, FIXED_PWM_POS_EN2=1, FIXED_PWM_POSEDGE2=0, FIXED_PWM_NEG_EN2=0, FIXED_PWM_NEGEDGE2=0, SHADOW_REG_EN2=0, FIXED_PWM_POS_EN3=1, FIXED_PWM_POSEDGE3=0, FIXED_PWM_NEG_EN3=0, FIXED_PWM_NEGEDGE3=0, SHADOW_REG_EN3=0.

Table 8 • **CorePWM Device Utilization and Performance (one TACH input configuration with 16-bit APB data width)**

	Tiles			Utilization		
Family	**Sequential**	**Combinatorial**	**Total**	**Device**	**Total**	**Performance (MHz)**
IGLOO/e	207	389	596	AGLE600V2	4.0%	52
ProASIC3/E	207	383	590	A3P250	10.0%	89
Fusion	207	383	590	AF600	4.0%	98
ProASICPLUS	207	506	713	APA300	9.0%	92
Axcelerator	209	269	478	AX250	5%	114
RTAX-S	209	269	478	RTAX250S	11%	101
SmartFusion	204	469	673	A2F500M3G	5.8%	132
SmartFusion2	156	248	404	M2S150T	0.28%	210
IGLOO2	156	248	404	M2GL150T	0.28%	210
RTG4	204	243	447	RT4G150	0.29%	172
PolarFire	204	289	493	PA5M500	0.09%	400

Note: FPGA resources and performance data for the PolarFire SoC family is similar to PolarFire family.

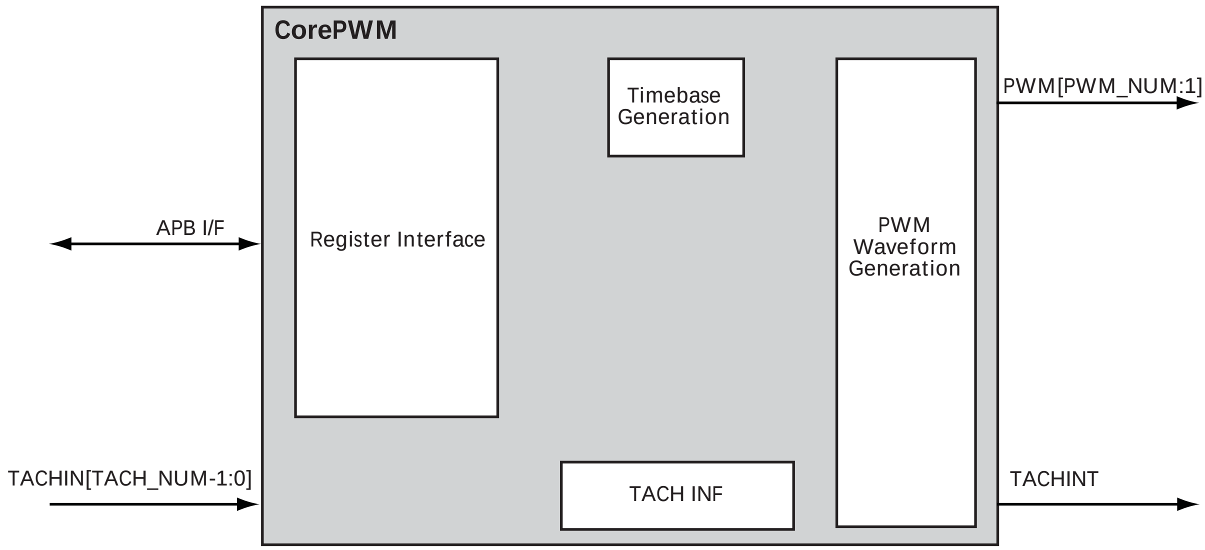
Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics were set as follows: CONFIG_MODE = 2, TACH_NUM=3, APB_DWIDTH=16.

3 Design Description

3.1 Functional Blocks

The CorePWM (pulse width modulation) macro generates up to 16 general purpose PWM signals, as shown in Figure 1. CorePWM includes a Register Interface block, Timebase Generation block, TACK INF block, and PWM Generation block.

Figure 1 • **CorePWM Block Diagram**



The **Register Interface** block connects to an APB bus for PWM register configuration and updating. Descriptions for all registers are given in Table 11, page 14. A Shadow Register may be used so that PWM waveform updates occur only at the beginning of a PWM period. A shadow register holds all values and writes them when the SYNC_UPDATE register is set to 1. In other words, for all channel synchronous updates, write a “1” to the SYNC_UPDATE register after writing to all the channel registers.

The **Timebase Generatio**n block accepts PRESCALE and PERIOD register values and produces a PERIOD count. The number of system clocks between PERIOD counts is equal to the PRESCALE value.

The **PWM Waveform Generation** block has two modes:

- **General Purpose PWM mode** takes input Period_cnt counter values and compares them with the register values for all the PWM positive and negative edge locations. When a comparison is met, each respective output waveform is set to the correct high/low/toggle value. An example General Purpose PWM waveform configuration is demonstrated in Figure 3, page 20. The example explains the relationship between the Prescale and Period register values, and how to configure the PWM waveforms with a given Prescale/Period timebase.
- **Low Ripple DAC mode** is intended to drive a low-pass filter, typically a single-pole RC filter. Narrow pulses of constant width are spread evenly over time such that the average voltage is equal to the duty cycle. The output of the filter is then a DC voltage directly proportional to the duty cycle. This type of pulse train allows for much lower ripple at the output of the filter, and benefits from either higher bandwidth and/or smaller R and C values.

In the Tach interface module, the width of the decrementing counter is configured to 16 bits. The Tach interface module is used to measure the period of the TACHIN[x] signal by measuring between two successive positive or negative edges of TACHIN[x]. The measured value will be stored in the corresponding input's TACHPULSEDUR register. The measured value will be read by the firmware through the APB interface. The access to the control and status registers of the Tach interface module is through the APB interface. The stored value in TACHPULSEDUR will correspond to the count for half of a revolution of a four pole fan. When determining the speed for other than four pole fans, the algorithm that converts the counter value to RPMs must be adjusted by the firmware. TACH INF supports 16- and 32-bit APB interface, but it does not support 8-bit interfaces.

To accurately measure the speed of 3-wire fans, you must turn on the fan periodically and long enough to get a complete tach measurement, often referred to as PWM pulse stretching. The PWM_STRETCH register allows you to set the desired PWMx signals to the level specified by PWM_STRETCH_VALUE. The following algorithm can be used to measure the speed of 3-wire fans. This algorithm assumes that the TACHMODEy bit is set to '1' (one-shot mode):

- Software enables pulse stretching by writing a '1' to PWM_STRETCHx, which forces PWMx to PWM_STRETCH_VALUEx. This requirement is not enforced by the hardware.
- Software can add a delay to ensure the fan tachometer circuitry is operational before enabling fan speed measurement.
- Software clears the TACHSTATUSy bit, enabling a one-time Tach measurement on the input signal TACHINy corresponding to one of the fans controlled by PWMx
- Software receives an interrupt and verifies that the Tach measurement for TACHINy has been completed (via TACHSTATUSy bit)
- Software disables pulse stretching by writing a '0' to PWM_STRETCHx.

3.2 I/O Signals

The port signals for the CorePWM macro are defined in Table 9, page 11 and illustrated in Figure 2. All signals are either Input (input only) or Output (output only).

Figure 2 • **CorePWM I/O Signal Diagram**

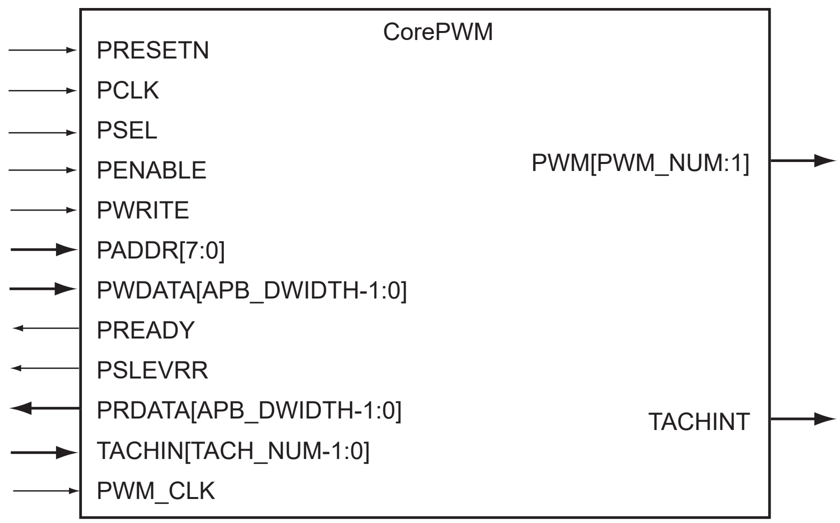


Table 9 • **CorePWM I/O Signal Descriptions**

Name	Type	Description
System Signals		
PRESETN	Input	Active low asynchronous reset
PCLK	Input	System clock - all operations and status shall be synchronous to the rising edge of this clock signal
Microcontroller Signals		
PSEL	Input	Select line for CorePWM
PENABLE	Input	Read output enable
PWRITE	Input	Write enable
PADDR[7:0]	Input	Register address
PWDATA[APB_DWIDTH-1:0]	Input	Write address/data input
PREADY	Output	Ready signal, tied High
PSLVERR	Output	Transfer error signal, tied Low
PRDATA[APB_DWIDTH-1:0]	Output	Read data output
PWM Signals		
PWM[PWM_NUM:1]	Output	Pulse width modulation output
TACH Signals		
TACHIN[TACH_NUM -1:0]	Input	TACH input
TACHINT	Output	Interrupt output for the tachometer. This signal indicates a TACHSTATUS register bit has been set to one. The polarity of this signal is controlled by the TACHINT_ACT_LEVEL configurable option.
PWM_CLK	Input	PWM clock - This clock used if the required frequency for PWM generation is greater/less than PCLK. This signal is only enabled when SEPERATE_PWM_CLK = 1

Note: All signals active-High (logic 1) unless otherwise noted.

3.3 Verilog/VHDL Parameters

CorePWM has parameters (Verilog) and generics (VHDL) for configuring the RTL code, described in Table 10. All parameters and generics are integer types.

Table 10 • **CorePWM Parameters/Generics Descriptions**

Name	Description
FAMILY	Selects the Target family. Must be set to match the supported FPGA family: 11 - Axcelerator 11 - Axcelerator 12 - RTAX-S 14 - ProASICPLUS 15 - ProASIC3 16 - ProASIC3E 17 - Fusion 18 - SmartFusion 19 - SmartFusion2 20 - IGLOO 21 - IGLOOe 22 - ProASIC3L 24 - IGLOO2 25 - RTG4 26 - PolarFire 27 - PolarFire SoC
CONFIG_MODE	When 0, supports PWM only (legacy with dead banding support) When 1, supports both PWM and TACH When 2, supports TACH only
PWM_NUM	Number of PWM outputs from 1 to 16. This parameter is used only when CONFIG_MODE is set to 0 or 1.
APB_DWIDTH	PWM resolution and APB bus width from 8 to 32. This parameter must be set to either 16 or 32 when CONFIG_MODE is either 1 or 2.
FIXED_PRESCALE_EN	Fixed Prescale Enable. FIXED_PRESCALE_EN hardwires the register, disallowing APB write-access, and reducing tile count. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PRESCALE	Hardwired PRESCALE[APB_DWIDTH -1:0] register value. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PERIOD_EN	Fixed Period Enable. FIXED_PERIOD_EN hardwires the register, disallowing APB write-access, and reducing tile count. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PERIOD	Hardwired PERIOD[APB_DWIDTH -1:0] register value. This parameter is used only when CONFIG_MODE set to 0 or 1.
SHADOW_REG_ENx	Shadow Register Enable. Synchronizes all register modification changes to the beginning of the PWM cycle; that is, when PERIOD Count = 0. This parameter is used only when CONFIG_MODE set to 0 or 1.
DAC_MODEx	DAC mode. 1 = Low ripple DAC mode; 0 = General purpose PWM mode. **Note:** Note: x refers to each channel, from 1 to 16. This parameter is used only when CONFIG_MODE set to 0 or 1.

Table 10 • CorePWM Parameters/Generics Descriptions (continued)

Name	Description
FIXED_PWM_POS_ENx	Fixed per channel Positive Edge Enable. **Note:** x refers to each channel, from 1 to 16. FIXED_PWM_POS_ENx hardwires the register, disallowing APB write-access, and reducing tile count. In a typical PWM application, either the FIXED_PWM_POS_ENx or the FIXED_PWM_NEG_ENx could be set if one of those edges do not need to be software controlled with APB write-accesses. Fixing both edges would result in static output. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PWM_POSEDGEx	Hardwired POSEDGE[APB_DWIDTH -1:0] register value. **Note:** Note: x refers to each channel, from 1 to 16. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PWM_NEG_ENx FIXED_DAC_OUT_ENx	Fixed per channel Negative Edge Enable. **Note:** x refers to each channel, from 1 to 16. FIXED_PWM_NEG_ENx hardwires the register, disallowing APB write-access, and reducing tile count. In a typical PWM application, either the FIXED_PWM_NEG_ENx or the FIXED_PWM_POS_ENx could be set if one of those edges do not need to be software-controlled with APB write-accesses. Fixing both edges would result in static output. For DAC applications, the FIXED_PWM_POS_ENx value is unconnected while the FIXED_DAC_OUT_ENx value would typically be disabled, as using it would result in static output. This parameter is used only when CONFIG_MODE set to 0 or 1.
FIXED_PWM_NEGEDGEx FIXED_DAC_LEVELOUTx	Hardwired NEGEDGE[APB_DWIDTH -1:0] register value. When in DAC Mode, this parameter also fixes DACx_LEVELOUT, which is typically not fixed in DAC applications, as it would only create a static duty cycle output. Note: x refers to each channel, from 1 to 16. This parameter is used only when CONFIG_MODE set to 0 or 1.
PWM_STRETCH_VALUEx	Defines PWMx level when PWM_STRETCHx is set to 1. When 0, PWMx is set to 0 if PWM_STRETCHx is set to 1. When 1, PWMx is set to 1 if PWM_STRETCHx is set to 1 (default). This parameter is used only when CONFIG_MODE set to 1.
TACH_NUM	Number of Tachometer inputs from 1 to 16. This parameter is used only when CONFIG_MODE set to 1 or 2.
TACH_EDGEy	Fixed per Tachometer input edge select. Selects the edge used to capture the counter value for the TACH[x] input signals. 0, capture counter value on falling edge of TACH[x] (default); 1, capture counter value on rising edge of TACH[x]. This parameter is used only when CONFIG_MODE set to 1 or 2.
TACHINT_ACT_LEVEL	Selects active Low or active High TACHINT interrupt: 0, active Low interrupt (default); 1, active High interrupt. This parameter is used only when CONFIG_MODE set to 1 or 2.
SEPERATE_PWM_CLK	Separate PWM Clock - If the clock frequency required for PWM generation is greater/less than PCLK this signal should be enabled. When enabled (1) input PWM_CLK can be used for PWM_CLK generation. If PCLK is at the desired clock frequency this parameter/generic should be disabled (0).

Note: APB_DWIDTH must always be greater than or equal to PWM_NUM for all APB read and write operations to be successful.

3.4 Register Map

All registers are based on APB width parameter selection; default is 8 bits.

Table 11 • **CorePWM Register Definitions**

Register Name	Paddr[7:0]	Description	Type	Default
PRESCALE	0x00	PWM MODE: The system clock cycle is multiplied with the PRESCALE value resulting in the minimum PERIOD count timebase. DAC MODE: The Prescale and Period Registers could be used in conjunction with the shadow register to synchronize DAC LEVELOUT.	R/W	0X08
PERIOD	0x04	PWM MODE: The PRESCALE value is multiplied with the PERIOD value yielding the PWM waveform cycle.	R/W	0x08
PWM_ENABLE_0_7	0x08	Bitwise channel enables for PWM/DAC channels 1 through 8.	R/W	0x00
PWM_ENABLE_8_15	0x0C	Bitwise channel enables for PWM/DAC channels 9 through 16.	R/W	0x00
SYNC_UPDATE	0xE4	SYNC_UPDATE: When this bit is set to “1” and SHADOW_REG_EN is selected, all POSEDGE and NEGEDGE registers are updated synchronously. Synchronous updates to the PWM waveform occur only when SHADOW_REG_EN is asserted and SYNC_UPDATE is set to "1". When this bit is set to "0", all the POSEDGE and NEGEDGE registers are updated asynchronously.	R/W	0x00
PWM1_POSEDGE	0x10	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution. When APB writes to this register, all the channels are updated.	R/W	0x00
PWM1_NEGEDGE DAC1_LEVELOUT	0x14	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM2_POSEDGE	0x18	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM2_NEGEDGE DAC2_LEVELOUT	0x1C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM3_POSEDGE	0x20	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM3_NEGEDGE DAC3_LEVELOUT	0x24	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%	R/W	0x00
PWM4_POSEDGE	0x28	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00

Table 11 • CorePWM Register Definitions (continued)

Register Name	Paddr[7:0]	Description	Type	Default
PWM4_NEGEDGE DAC4_LEVELOUT	0x2C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM5_POSEDGE	0x30	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM5_NEGEDGE DAC5_LEVELOUT	0x34	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM6_POSEDGE	0x38	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM6_NEGEDGE DAC6_LEVELOUT	0x3C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM7_POSEDGE	0x40	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM7_NEGEDGE DAC7_LEVELOUT	0x44	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM8_POSEDGE	0x48	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM8_NEGEDGE DAC8_LEVELOUT	0x4C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM9_POSEDGE	0x50	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM9_NEGEDGE DAC9_LEVELOUT	0x54	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM10_POSEDGE	0x58	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM10_NEGEDGE DAC10_LEVELOUT	0x5C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM11_POSEDGE	0x60	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM11_NEGEDGE DAC11_LEVELOUT	0x64	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM12_POSEDGE	0x68	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00

Table 11 • **CorePWM Register Definitions *(continued)***

Register Name	Paddr[7:0]	Description	Type	Default
PWM12_NEGEDGE DAC12_LEVELOUT	0x6C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM13_POSEDGE	0x70	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM13_NEGEDGE DAC13_LEVELOUT	0x74	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM14_POSEDGE	0x78	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM14_NEGEDGE DAC14_LEVELOUT	0x7C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM15_POSEDGE	0x80	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM15_NEGEDGE DAC15_LEVELOUT	0x84	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM16_POSEDGE	0x88	PWM MODE: Sets the positive edge of the output with respect to the PERIOD resolution.	R/W	0x00
PWM16_NEGEDGE DAC16_LEVELOUT	0x8C	PWM MODE: Sets the negative edge of the output with respect to the PERIOD resolution. DAC MODE: Sets the desired output level, from 0-100%.	R/W	0x00
PWM_STRETCH	0x90	When 0, the state of PWMx is determined by PWMx_POSEDGE/NEGEDGE register settings. When 1, PWMx is set to PWM_STRETCH_VALUEx.	R/W	0x0000
TACHPRESCALE	0x94	Clock prescale setting. Determines effective clock rate for the counter based on PCLK: 0000 = divide by 1 (default) 0001 = divide by 2 0010 = divide by 4 0011 = divide by 8 0100 = divide by 16 0101 = divide by 32 0110 = divide by 64 0111 = divide by 128 1000 = divide by 256 1001 = divide by 512 1010 = divide by 1,024 1011 = divide by 2,048 Others = divide by 2,048	R/W	0x0

Table 11 • CorePWM Register Definitions (continued)

Register Name	Paddr[7:0]	Description	Type	Default
TACHSTATUS	0x98	TACH status register which contains one bit per TACH input, indicating whether the respective TACHPULSEDUR register has been updated at least once since the bit was cleared. The bits in this register gets cleared by writing "1", "0" does not have any effect.	R/W1 C	0x0000
TACHIRQMASK	0x9C	TACH interrupt mask register with one bit per tachometer signal, indicating whether CorePWM needs to assert an interrupt if the respective bit in TACHSTATUS register is asserted.	R/W	0x0000
TACHMODE	0xA0	TACH Mode. Sets the measurement mode used for each TACH input. When 0: TACH input is continuously measured and stored in the respective TACHPULSEDUR register. When 1: A one-time measurement is performed only if the respective bit in TACHSTATUS register is cleared.	R/W	0x0000
TACHPULSEDUR_0	0xA4	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[0]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_1	0xA8	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[1]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_2	0xAC	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[2]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_3	0xB0	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[3]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_4	0xB4	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[4]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000

Table 11 • CorePWM Register Definitions (continued)

Register Name	Paddr[7:0]	Description	Type	Default
TACHPULSEDUR_5	0xB8	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[5]. The edge to be used is configurable.If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_6	0xBC	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[6]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_7	0xC0	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[7]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_8	0xC4	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[8]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_9	0xC8	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[9]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_10	0xCC	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[10]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_11	0xD0	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[11]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_12	0xD4	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[12]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000

Table 11 • CorePWM Register Definitions (continued)

Register Name	Paddr[7:0]	Description	Type	Default
TACHPULSEDUR_13	0xD8	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[13]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_14	0xDC	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[14]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000
TACHPULSEDUR_15	0xE0	Stores the number of timer ticks between two successive positive (or negative) edges from the TACHIN[15]. The edge to be used is configurable. If the number of timer ticks exceeds the maximum register value, the value of 0 shall be stored instead.	R	0x0000

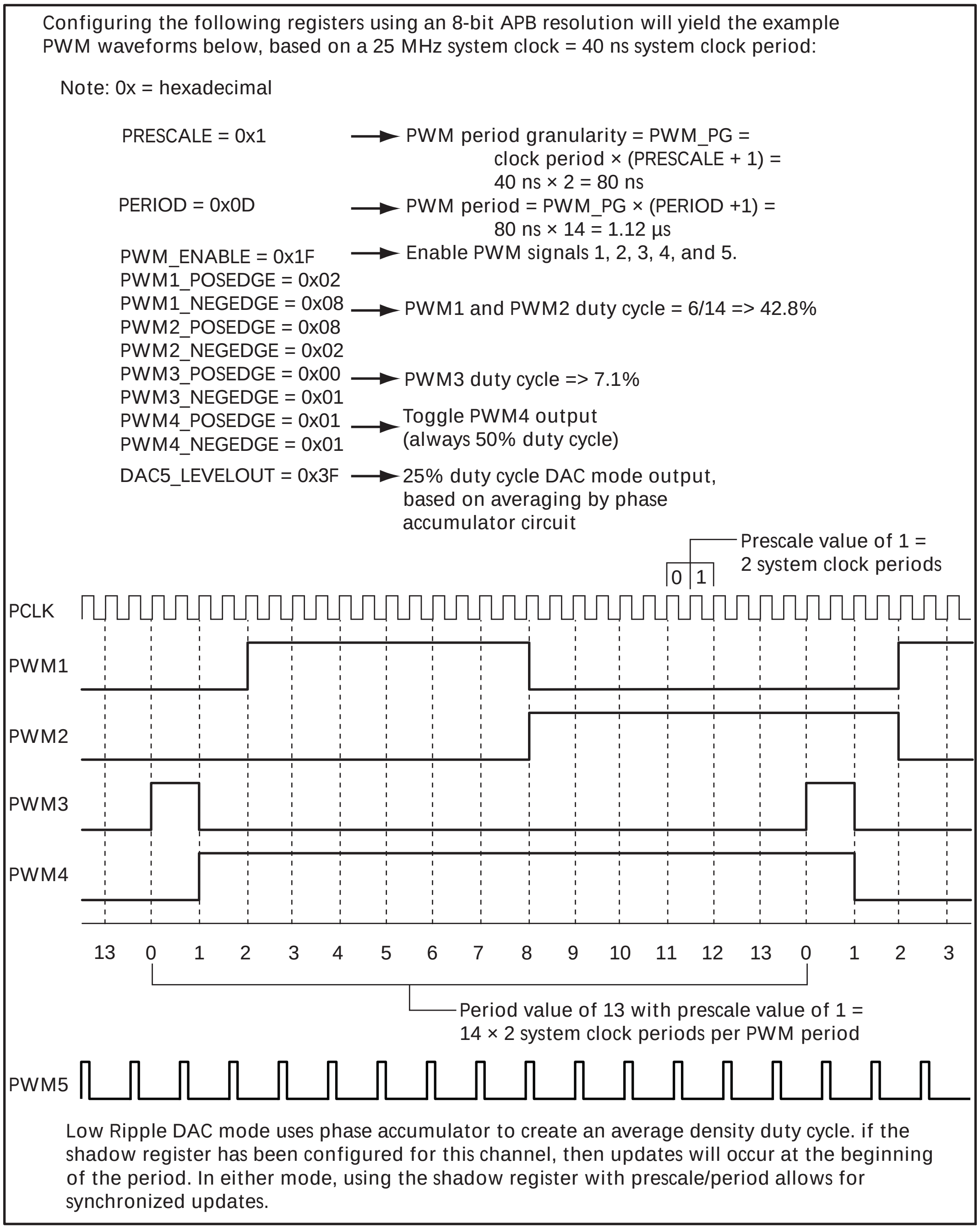
Note:

1. 0d = decimal; 0x = hexidecimal; 0b = binary.
2. If DAC mode for all active channels is configured as “Low ripple DAC mode” (DAC_MODEx = 1) and shadow register are disabled (SHADOW_REG_ENx = 0) for all active channels register PRESCALE and PERIOD become read only because they are not used in Low ripple DAC mode.

3.5 Configuration Example

Figure 3 demonstrates how several register configurations affect General Purpose and Low Ripple DAC PWM output waveform generation.

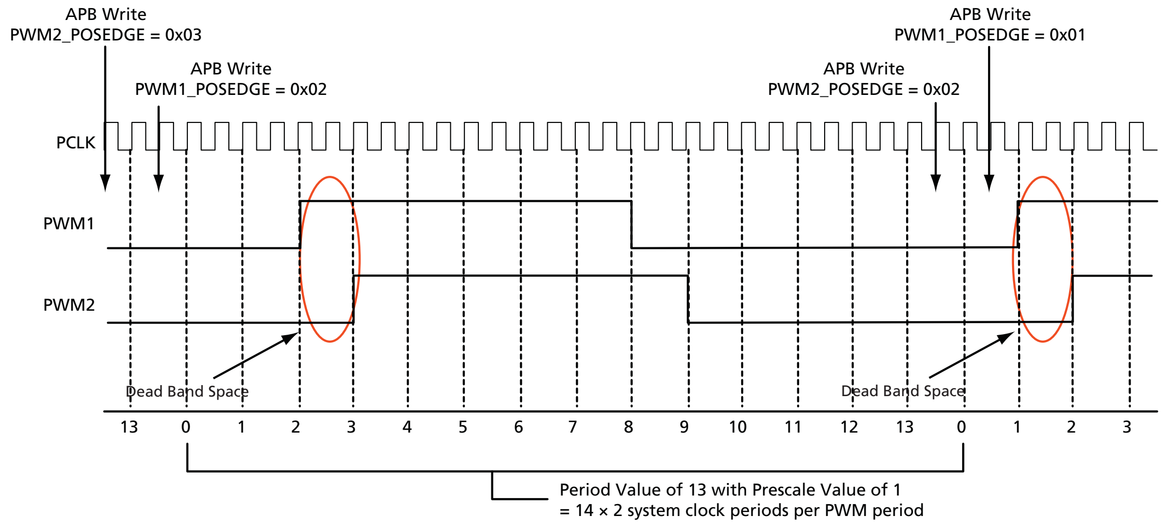
Figure 3 • **CorePWM Waveform Generation Example**



Note: If SEPERATE_PWM_CLK is enabled (1) then replace PCLK with PWM_CLK in above timing diagram.

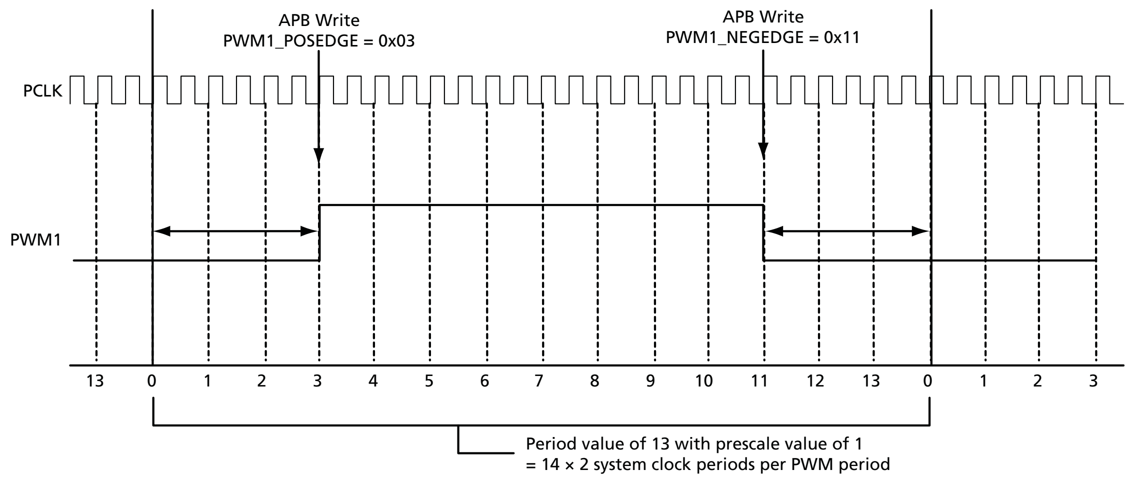
Figure 4 to Figure 7, page 22 demonstrate how to avoid overlapping of the dead banding issue using the register configurations of channel 1 and channel 2 as a pair. Both channel 1(PWM1) and channel 2(PWM2) are updated after writing a “1” to the SYNC_UPDATE configuration register to avoid overlapping of dead band space.

Figure 4 • **Dead Band Space Example**



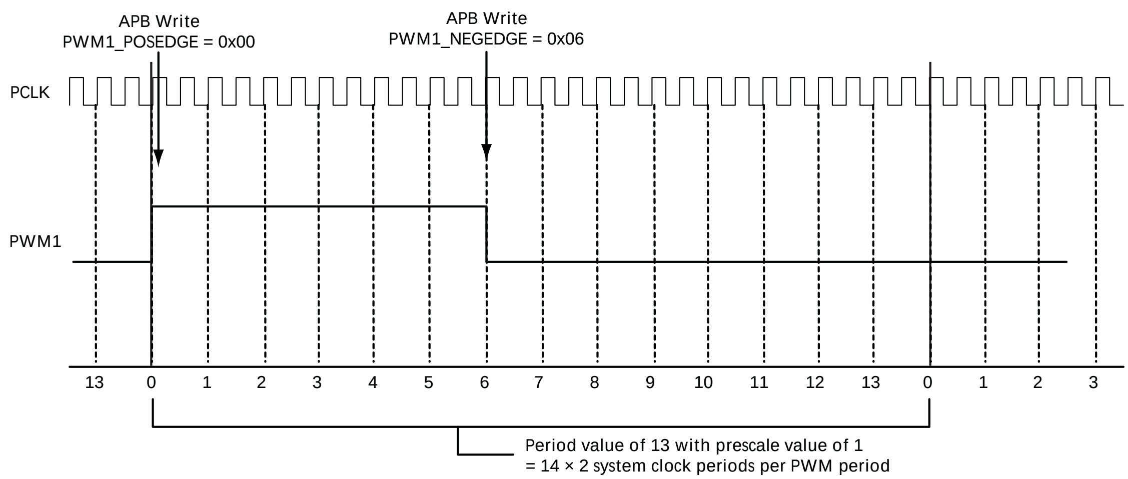
Note: If SEPERATE_PWM_CLK is enabled (1) then replace PCLK with PWM_CLK in above timing diagram.

Figure 5 • **Center-Aligned PWM Waveform Example**



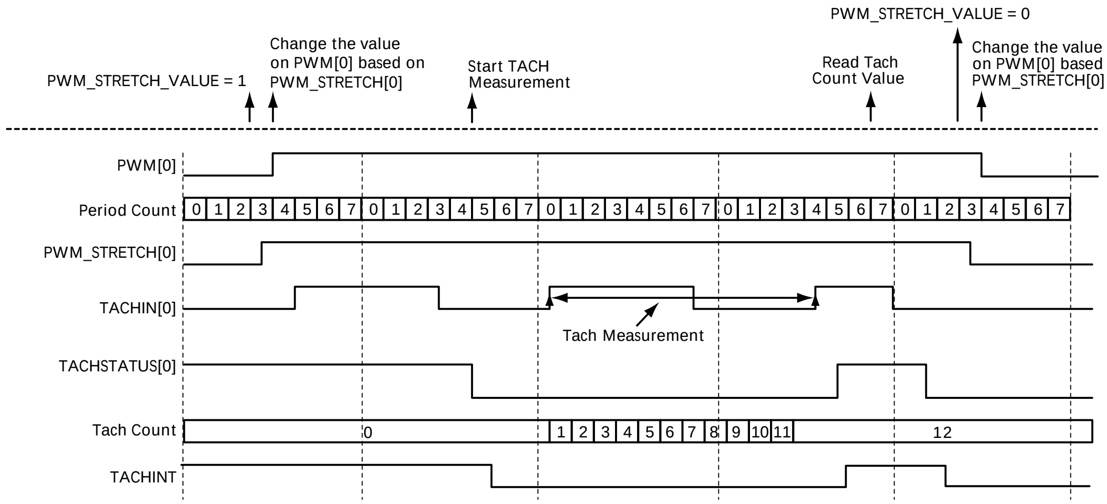
Note: If SEPERATE_PWM_CLK is enabled (1) then replace PCLK with PWM_CLK in above timing diagram.

Figure 6 • **Left-Aligned PWM Waveform Example**



Note: If SEPERATE_PWM_CLK is enabled (1) then replace PCLK with PWM_CLK in above timing diagram.

Figure 7 • **Tach Measurement**



3.6 APB Interface Timing

Figure 8 and Figure 9 depict typical write cycle and read cycle timing relationships relative to the system clock.

Figure 8 • **Data Write Cycle**

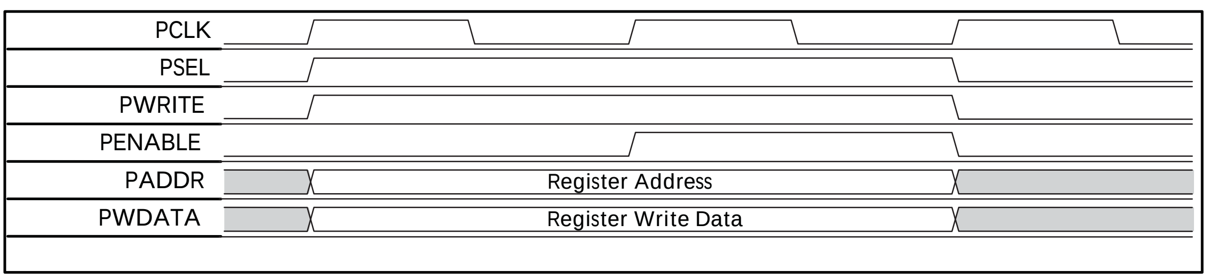
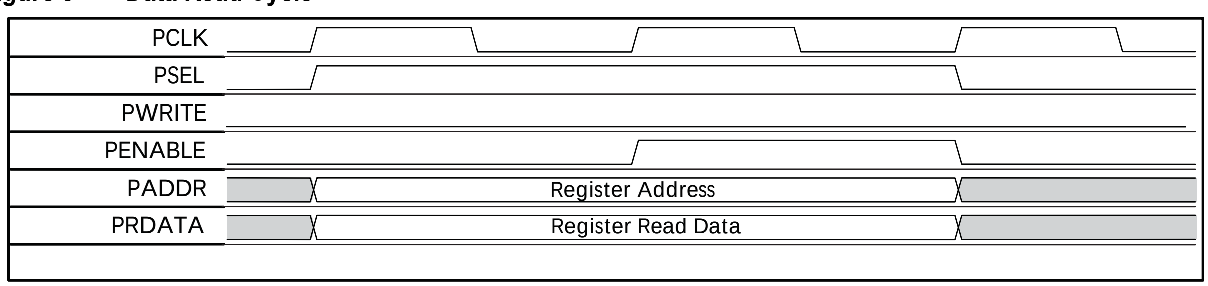


Figure 9 • **Data Read Cycle**



4 Tool Flows

CorePWM is license free.

4.1 RTL

Complete RTL source code is provided for the core and testbenches.

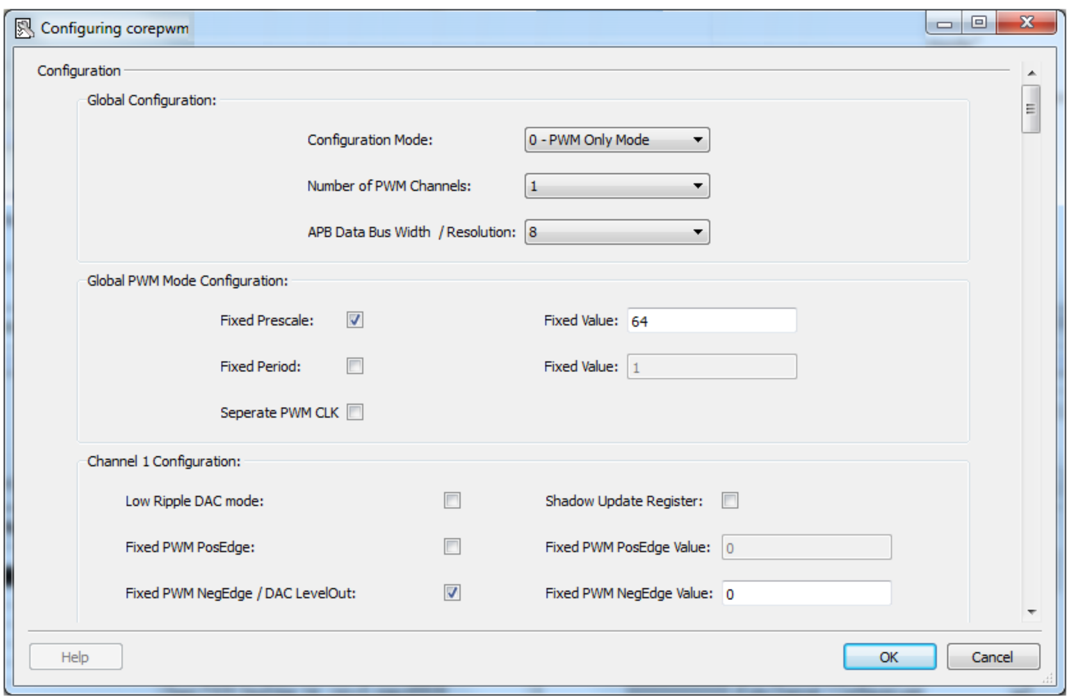
4.2 SmartDesign

The core can be configured using the configuration GUI within SmartDesign. To know how to create SmartDesign project using the IP cores, refer to *Libero SoC documents page* and use the latest SmartDesign user guide.

An example of configuring one channel for PWM mode operation is shown in Figure 10. Note the following in this example:

- Number of PWM Channels is 1.
- APB bit width and corresponding PWM resolution is 8 bits.
- The Prescale value (the number of clock ticks between Period ticks) is selected to be Fixed at 64, reducing the tile count, as no registers are used.
- The Period value is not Fixed and hence software-controlled.
- The Shadow Update Register is enabled, allowing for synchronized PWM updates at the beginning the Period count.
- The Positive edge of the PWM is not Fixed and hence software-controlled.
- The Negative edge of the PWM waveform is Fixed at Period count 0.
- The User Testbench is selected to be generated.

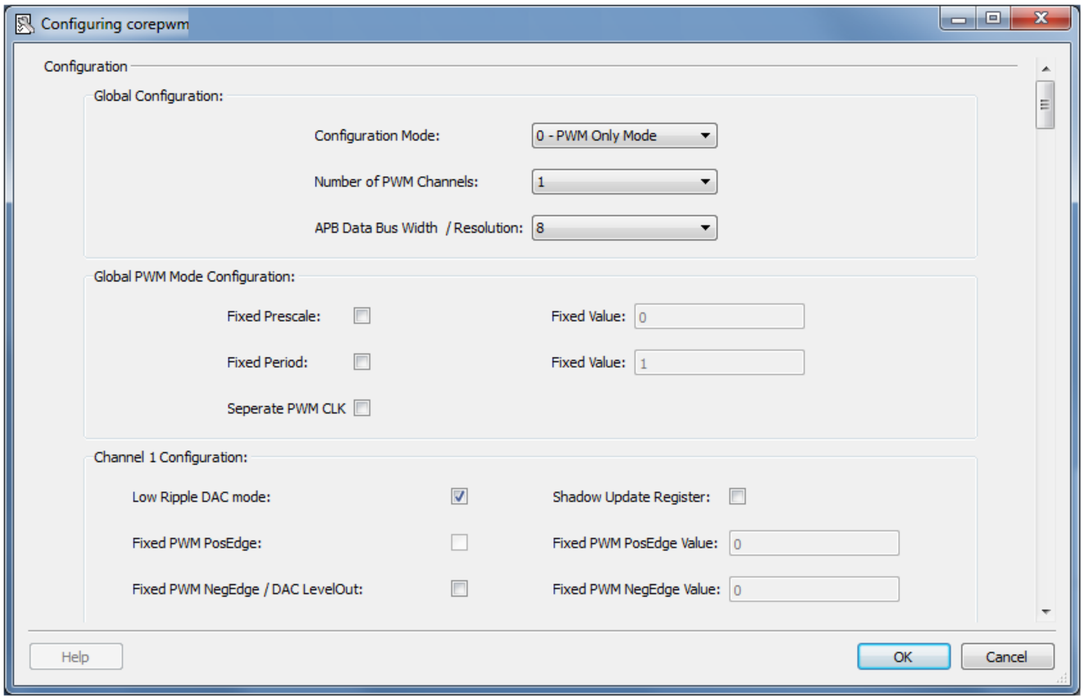
Figure 10 • **CorePWM Configuration within SmartDesign - PWM Mode**



An example of configuring one Channel for low ripple DAC operation is shown in Figure 11. Note the following in this example:

- Number of PWM Channels is 1.
- APB bit width and corresponding PWM/DAC resolution is 8 bits.
- The Prescale and Period Values can be used in conjunction with the Shadow Update Register to update DAC1_LEVELOUT values at a given period. For example, 3 DACs could be updated simultaneously based on the Prescale and Period values if the Shadow Update Register is enabled. In this example, the DAC1_LEVELOUT value is updated whenever the APB bus updates the DAC1_LEVELOUT register.
- The DAC LevelOut value is not Fixed and hence software-controlled. Note the DAC1_LEVELOUT value is synonymous with a Duty Cycle value; i.e., an 8-bit DAC1_LEVELOUT hex value of 7F is equal to an average duty cycle of 50% and will yield half of the full analog value after RC filtering.
- Note the “Fixed PWM PosEdge” value does not apply to DAC mode channel.
- The User Testbench is selected to be generated.

Figure 11 • **CorePWM Configuration within SmartDesign - DAC Mode**



4.3 Importing into Libero IDE

CorePWM is available for download to the SmartDesign IP Catalog, via the Libero IDE web repository. For information on using SmartDesign to instantiate, configure, connect, and generate cores, refer to the Libero IDE online help.

4.4 Importing into Libero SoC

The CorePWM CPZ file must be installed into Libero software. This is done automatically through the Catalog update function in Libero, or the CPZ file can be manually added using the **Add Core** catalog feature. Once the CPZ file is installed in Libero, the core can be configured, generated, and instantiated within SmartDesign for inclusion in the Libero project. For more information, see the *Knowledge Based article*.

To know how to create SmartDesign project using the IP cores, refer to *Libero SoC documents page* and use the latest SmartDesign user guide.

4.5 Simulation Flows in Libero IDE

To run simulations, select the user testbench within the SmartDesign CorePWM configuration GUI, right-click, and select **Generate Design**. When SmartDesign generates the design files, it will install the appropriate testbench files. Set the design root to the CorePWM instantiation in the Libero IDE design hierarchy pane, and click the **Simulation** icon in the Libero IDE Design Flow window. This will invoke ModelSim® and automatically run simulation.

A simplified block diagram of the User Testbench is shown in Figure 12. The user testbench instantiates the CorePWM macro and provides a Register Write Stimulus process, Register Read process, and a PWM output duty cycle check process.

4.6 Simulation Flows in Libero SoC

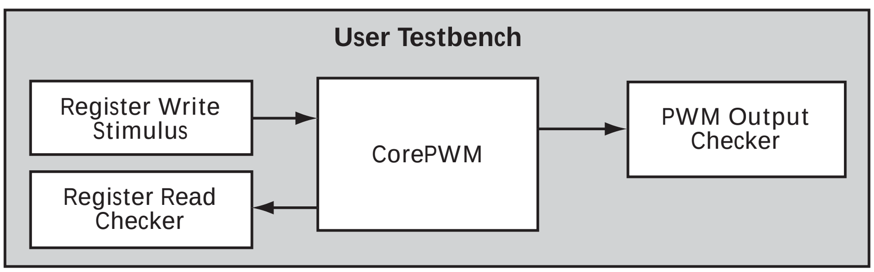
The User Testbench for CorePWM is included in all the releases.

To run simulations, select the **User Testbench** flow in the SmartDesign and click **Save and Generate** on the Generate pane.

The User Testbench is selected through the Core Testbench Configuration GUI. When SmartDesign generates the Libero SoC project, it installs the user testbench files.

To run the user testbench, set the design root to the CorePWM instantiation in the Libero SoC design hierarchy pane and click **Simulation** in the Libero SoC Design Flow window. This invokes ModelSim® and automatically runs the simulation.

Figure 12 • **CorePWM Verification Testbench**



4.7 Synthesis in the Libero IDE

Having set the design route appropriately, click the **Synthesis** icon in Libero IDE. The Synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, select the **Run** icon.

4.8 Place-and-Route in Libero IDE

Having set the design route appropriately and run Synthesis, click on the **Layout** icon in Libero IDE to invoke Designer. CorePWM requires no special place-and-route settings.

4.9 Synthesis in Libero SoC

After setting the design root appropriately for the design, use the following steps to run the Synthesis:

1. Click **Synthesis** in the Libero SoC software. The Synthesis window appears displaying the Synplicity project.
2. Set Synplicity to use the Verilog 2001 standard if Verilog is used.
3. Click **Run** to run the Synthesis.

4.10 Place-and-Route in Libero SoC

After setting the design route appropriately for the design, and running Synthesis, click **Layout** in the Libero SoC software to invoke Designer. CorePWM does not require special place-and-route settings.

5 Example Applications

For General Purpose PWM applications, a duty cycle calculator is available online to assist in calculating the PWM POSEDGE and NEGEDGE register values, given a requested duty cycle. This is provided on the Microsemi website as a downloadable Excel spreadsheet:

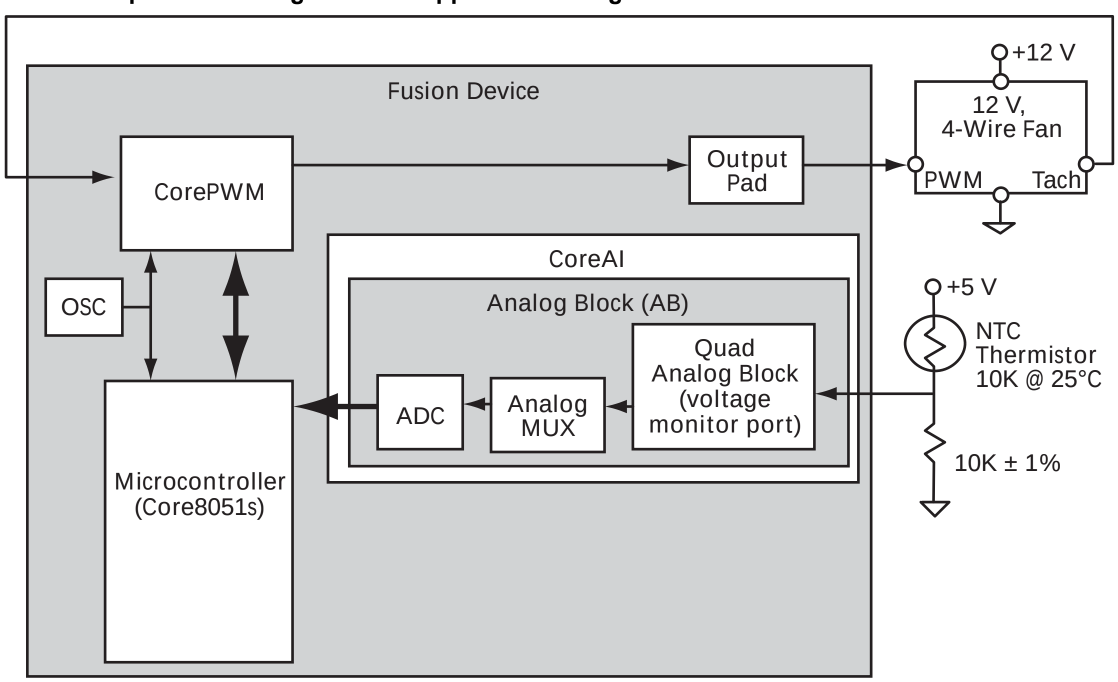
http://www.microsemi.com/documents/duty_cycle_calc.zip

For DAC applications, a low ripple DAC calculator is also available online: http://www.microsemi.com/documents/low_ripple_dac_calc.zip

5.1 General Purpose PWM Application - Temperature Monitor

A typical temperature monitor application using CorePWM is shown in Figure 13. In this example, fan speed is controlled by fluctuations in the NTC thermistor's resistive value. As shown, changes in the input voltage to the voltage monitor port will be converted to a digital value via the ADC and forwarded to an on-chip microcontroller (such as Core8051s). The microcontroller algorithm will periodically configure/reconfigure CorePWM registers based on the thermistor value and/or the fan's tachometer value.

Figure 13 • **Temperature/Voltage Monitor Application Using CorePWM in a Fusion Device**

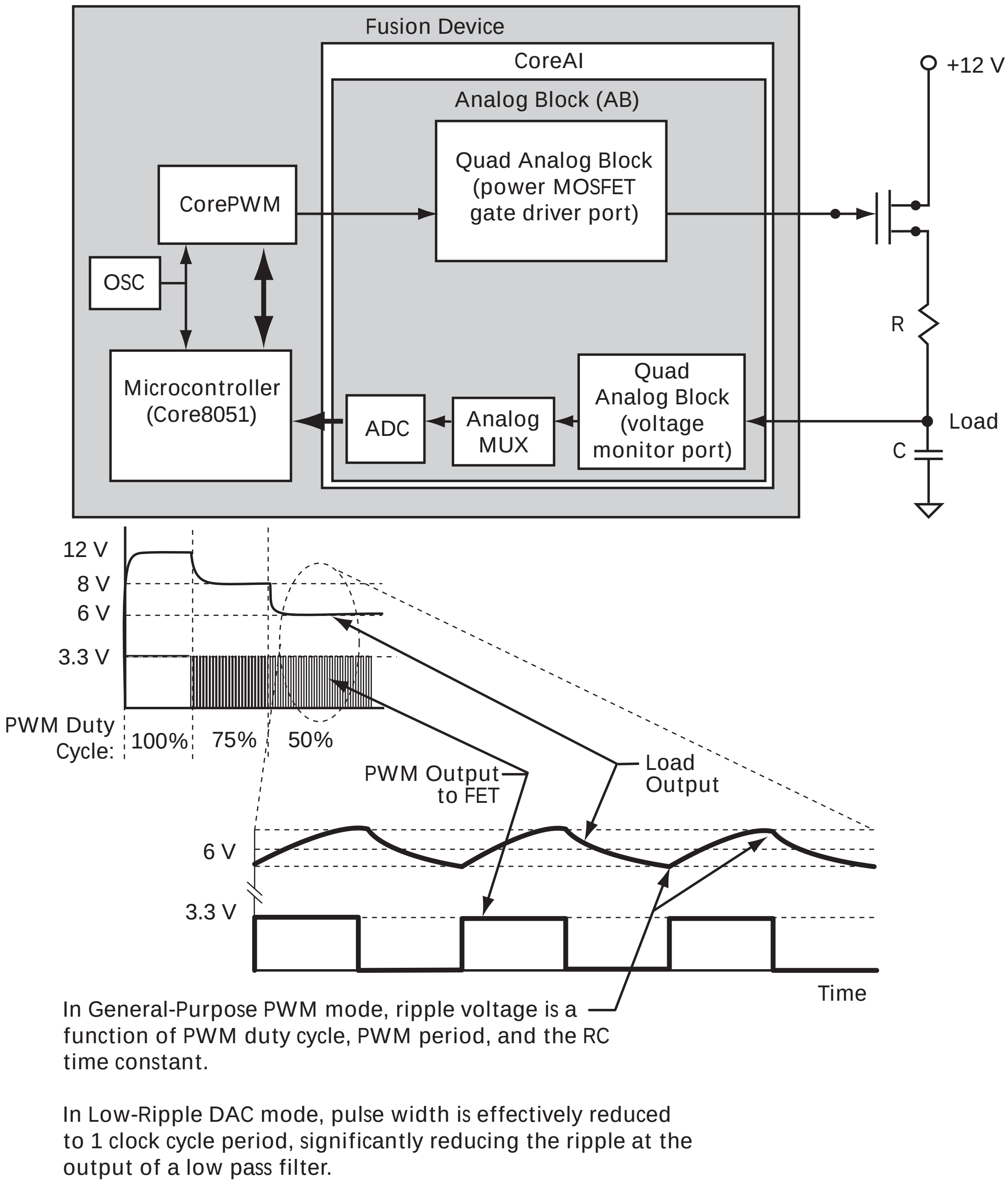


5.2 DAC

A typical DAC application using CorePWM is shown in Figure 14. In this example, PWM output is averaged to a varying DC voltage. At reset, the PWM duty cycle, or level out value, is 100% and the voltage increases to the rail of 12 volts. The PWM duty cycle / level out value changes to 75% and then 50%, and the output of the RC filter follows this by dropping to 8 volts and then 6 volts. The generated ripple voltage is a function of the RC circuit values, the APB system clock period, and the PWM duty cycle.

As shown, a field-effect transistor (FET) is used to increase and decouple the output voltage/current from the Fusion device. The load is monitored and changes to the PWM output are processed via a microcontroller (Core8051s, CoreABC, etc.).

Figure 14 • **DAC Application Using CorePWM in Fusion Device**



The FET, in this case, is used to illustrate the ability to extend the DAC's output to 12 V. For most applications, 3.3 V is sufficient. Higher clock speeds (and therefore lower ripple) can be achieved by driving the RC filter with a general purpose TTL output.

Using Low Ripple DAC mode has the added benefit of requiring a smaller time constant for the filter, which allows for smaller R and C components to be used. A Low Ripple DAC calculator is available to assist in determining the ideal values for R and C.

6 Software Driver

Drivers for CorePWM are available via the Firmware Catalog tool provided with Libero IDE. For more information on the Firmware Catalog, refer to:

https://www.microsemi.com/product-directory/design-tools/4880-firmware-catalog#downloads