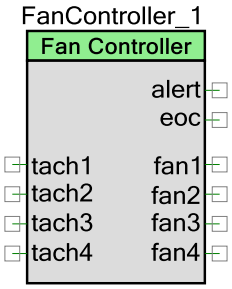
Fan Controller

4.0



Features

- Support for up to 16 PWM-controlled, 4-wire brushless DC fans for PSoC 3/PSoC 5LP devices and up to 6 fans for PSoC 4
- Individual or banked PWM outputs with tachometer inputs
- Supports 25 kHz, 50 kHz or user-specified PWM frequencies
- Supports fan speeds up to 25,000 RPM
- Supports 4-pole and 6-pole motors
- Supports fan stall / rotor lock detection on all fans
- Supports firmware controlled or hardware controlled fan speed regulation for PSoC 3/PSoC 5LP
- Supports firmware-controlled fan speed regulation for PSoC 4
- Customizable alert pin for fan fault reporting

General Description

The Fan Controller Component enables designers to quickly and easily develop fan controller solutions using PSoC. The Component is a system-level solution that encapsulates all necessary hardware blocks including PWMs, or TCPWMs for PSoC 4, tachometer input capture timer, control registers, status registers and a DMA controller (ISR for PSoC 4) reducing development time and effort.

The Component is customizable through a graphical user interface enabling designers to enter fan electromechanical parameters such as duty cycle-to-RPM mapping and physical fan bank organization. Performance parameters including PWM frequency and resolution as well as open or closed loop control methodology can be configured through the same user interface. Once the system parameters are entered, the Component delivers the most optimal implementation saving resources within PSoC to enable integration of other thermal management and system management functionality. Easy-to-use APIs are provided to enable firmware developers to get up and running quickly.

Note Designs using the Fan Controller Component should not use PSoC's low power sleep or hibernate modes. Entering those modes prevents the Fan Controller Component from controlling and monitoring the fans.

When to Use a Fan Controller

The Fan Controller Component should be used in any thermal management application that needs to drive and monitor 4-wire, PWM based DC cooling fans. If the application requires more than 16 fans, the Fan Controller Component can be instantiated multiple times. Similarly, if the fans in the application are organized into banks, designers have the option of instantiating one Fan Controller Component per bank or instantiating one Component that handles all the banks.

Input/Output Connections

This section describes the various input and output connections for the Fan Controller. An asterisk (*) in the list of I/Os states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input *

An input for a user-defined clock source for the fan control PWMs. It is present only when the External Clock option is selected in the Component customizer.

tach1-16 – Input *

Tachometer signal from each fan that enables the Fan Controller to measure fan rotational speeds. The Component is designed to work with 4-pole DC fans that produce 2 high-low pulse trains per rotation on their tachometer output or 6-pole DC fans that produce 3 high-low pulse trains. tach2..16 inputs are optional.

Note For PSoC 4 devices, the maximum number of tach inputs is limited to six because of the limited digital resources.

fan1-16 – Output *

PWM output with variable duty cycle to control the speed of the fans. These output terminals are replaced by the bank1..8 outputs if fan banking is enabled.

Note For Automatic Hardware (UDB) mode, the maximum number of fan outputs (and associated tach inputs) is limited to 12 to minimize digital resource utilization.

Note For PSoC 4 devices the maximum number of fan outputs is limited to four because of the limited digital resources.

bank1-8 – Output *

PWM output with variable duty cycle to control the speed of the fan banks. These outputs appear only when banking is enabled.

alert – Output

Active high output terminal asserted when fan faults are detected (if enabled). Fan faults can be either a fan stall/rotor lock or speed regulation failure. The signal is synchronous to the “eoc” signal, and the “alert” signal asserts high right after the “eoc”. This signal is a “sticky” signal. That is, once the signal is asserted high for a fault condition, it remains high until the GetAlertSource() API is called in firmware. If the fault condition is not located and fixed, then the “alert” signal will be set again after the “eoc” signal pulse. In the case of a fan stall failure, the “alert” signal will be set right after the next “eoc” pulse. In the case of a speed regulation failure the alert will be generated after 16 fault conditions in a row occur.

eoc – Output

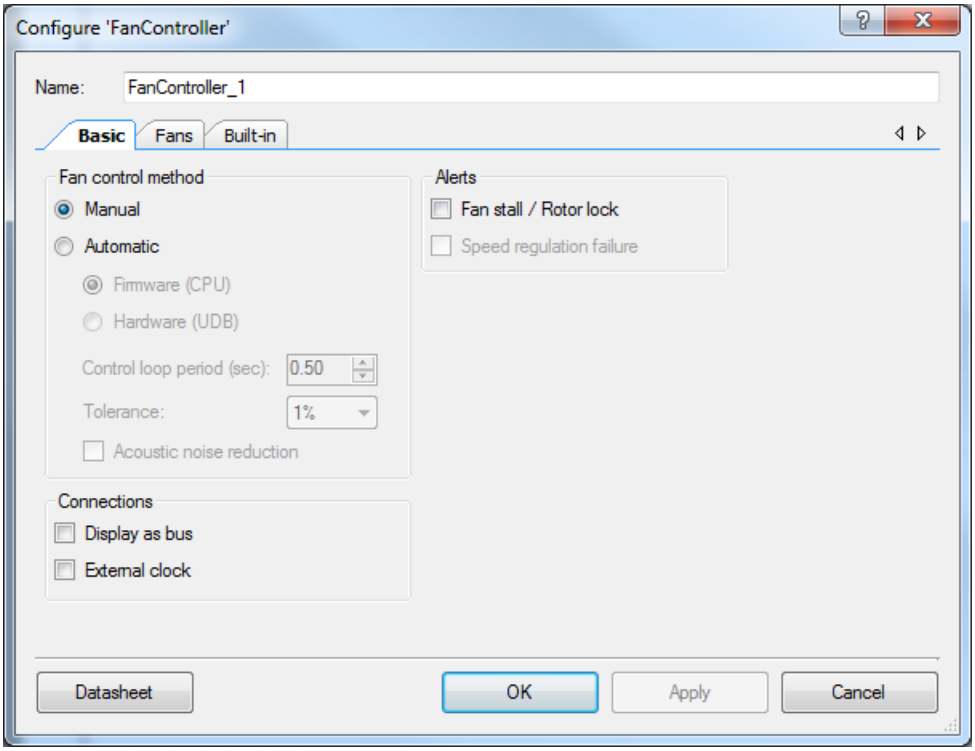
End-of-Cycle output is pulsed high each time the tachometer block has measured the speed of all fans in the system. This can be used to synchronize firmware algorithms to the Fan Controller hardware by connecting the terminal to a Status Register Component or to an Interrupt Component.

Component Parameters

Drag a Fan Controller onto your design and double click it to open the Configure dialog.

Basic Tab

This tab is used to configure the fundamental operating parameters of the Component.



This tab contains the following parameters:

Fan control method – Manual/Automatic

This parameter determines how the fan speeds are controlled. The possible options for selection include:

- Manual
- Automatic
  - Firmware (CPU)
  - Hardware (UDB)

Manual

Manual speed control is an open loop fan control mode and represents the lowest complexity implementation. In this mode, the SetDutyCycle() API is used to set the PWM output duty cycle. The actual fan speed is not used or taken into account. The assumption is that the fan will run at the desired speed. However, fan faults (stalled or locked rotor) are detected and reported.

Manual speed control setting should be selected in cases where:

1. A complex or custom fan control algorithm needs to be implemented in firmware within PSoC.
2. An external host controller is responsible for managing the fan speed algorithm and the Fan Controller Component is simply being used as the hardware interface to the fans.
3. Multiple fans are organized into banks sharing a common PWM drive signal.

Automatic Firmware (CPU)

Automatic Firmware (CPU) mode utilizes a firmware PID algorithm that is buried in the Component to control speed regulation. When this mode is selected, an additional tab, **PID Control**, displays in the Component Configure dialog. This tab provides an interface to enter PID algorithm parameters. Based on the parameters, the PID algorithm, which is implemented in ISR, analyzes the desired and actual fan's RPM and sets the proper duty cycle for a fan currently controlled.

This mode should be used to control multiple fans independently using an optimized PID control loop running in firmware.

Automatic Hardware (UDB)

Automatic Hardware (UDB) mode dictates that hardware blocks inside the PSoC control fan speed regulation automatically without any CPU intervention. Firmware sets the desired speed for each fan and the hardware automatically adjusts the PWM duty cycle to achieve and maintain the desired speed within specified tolerance limits.

The Automatic Hardware (UDB) setting should be selected to control multiple fans with minimal firmware development.

Note For PSoC 4 devices, the Automatic Hardware (UDB) option is disabled because PSoC 4 has limited UDB resources.

Control loop period

In Automatic controlled fan mode, this parameter controls the dynamic response time of the Automatic Hardware or Firmware control loop (closed loop). This parameter controls how frequently the internal algorithm adjusts the PWM duty cycles for each fan. It enables fine tuning of the hardware control logic to match the selected fan's electromechanical characteristics. The valid range for this parameter is 0 to 2.55 (specified in seconds). The default setting is 0.5.

Tolerance

The option is only present in Automatic Hardware (UDB) control fan mode. This parameter sets the acceptable tolerance when specifying desired fan speed targets. The tolerance is specified as a percentage relative to the desired speed setting. This parameter enables fine tuning of the hardware control logic to match the selected fan's electromechanical characteristics.

The valid range for this parameter is 1 to 10%. The default setting is 1%. If 8-bit PWM resolution is selected in the **Fan Controller Fans** tab, a Tolerance setting of 5% is recommended.

Acoustic Noise Reduction

In Automatic Hardware (UDB) control fan mode, this parameter limits audible noise from fans by limiting the positive rate of change of speed. If enabled, and firmware requests an increase in desired fan speed, the PWM duty cycle applied to the fan increases gradually to the new setting rather than applying a sudden step change. This eliminates noisy fan whir from sudden speed increases. This option is selected by default. This option is disabled for Automatic Firmware (CPU) mode because the PID algorithm has this option built in.

Connections

- **Display as bus –** If selected, the tach inputs and fan/bank outputs will be displayed as buses. Otherwise, they are displayed as individual terminals.
- **External clock –** If selected, you may connect an external clock source to the Component clock input. If not selected, then an internal clock source will be used.

Alerts

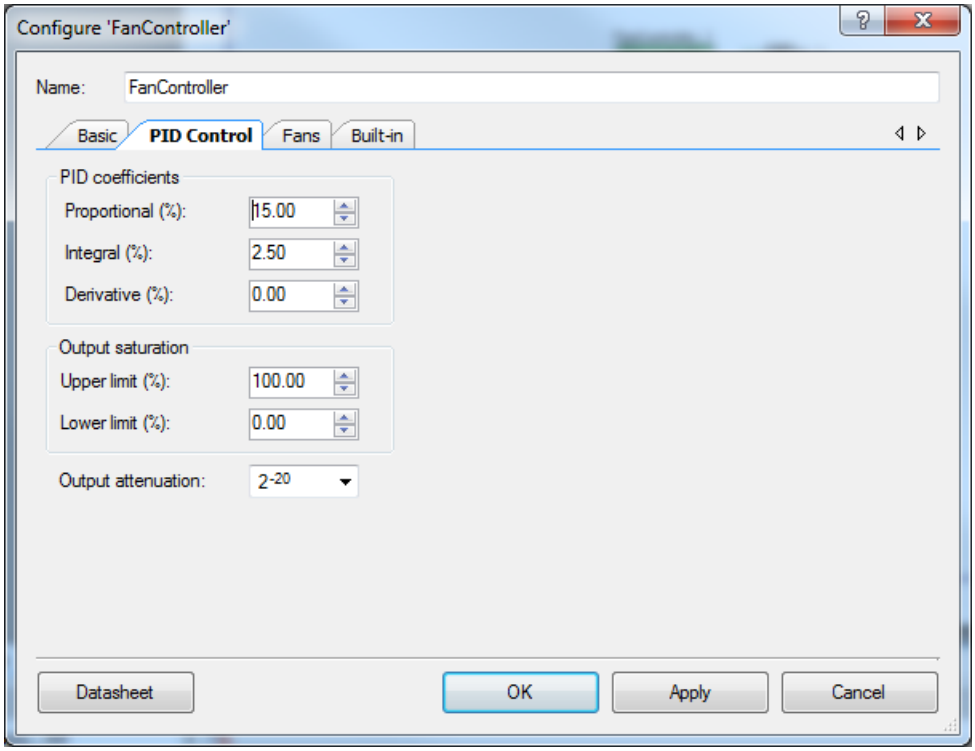
- **Fan stall / Rotor lock –** The Fan Controller can be configured to generate an active-high alert signal when a fan stalls or stops rotating due to a mechanical obstruction. The fan stall / rotor lock alert also will be generated when a fans speed is less the 500 RPM. This option is selected by default.
- **Speed regulation failure –** The Fan Controller can be configured to generate an active-high alert signal in Automatic control fan mode when the Automatic control loop is not able to achieve the desired speed. This option is not selected by default.

  Speed regulation faults occur if the automatic control algorithm is not able to get the fans to reach the desired speed. This could occur if the PWM driving the fan is already set at

100% but the actual speed is still below the desired speed. This could also occur if the PWM is already set to 0% but the actual speed is still above the desired speed. To avoid false alerts, the speed regulation fault is generated if the fault condition occurs during 16 consecutive speed measurement periods.

PID Control Tab

This tab is used to configure the coefficients for Firmware CPU Control mode.



This tab contains the following parameters:

PID Coefficients

These three gain parameters: **Proportional gain (K_P), Integral gain (K_I), and Derivative gain (K_D)** are used in the PID Control algorithm. Each term is presented as a ratio from 0.00% to 100.00%. The default terms are set to yield slow and stable performance for most systems (that is, low bandwidth PI control). These coefficients are applied to each fan in the configuration at startup and can be changed later using the FanController_SetPID() API. The values can be changed individually for each fan.

See the PID Algorithm section of this datasheet for details on PID Coefficients.

Output saturation

- **Upper limit –** This parameter defines the initial upper boundary for the integrator. It is presented as a ratio from 0.00% to 100.00%. This parameter can be used to set the upper

limit for the fan; it also prevents integrator windup commonly seen in unbounded control systems. The value is applied to each fan in the configuration at start and can be changed later using the FanController_SetSaturation() API. The value can be changed individually for each fan.

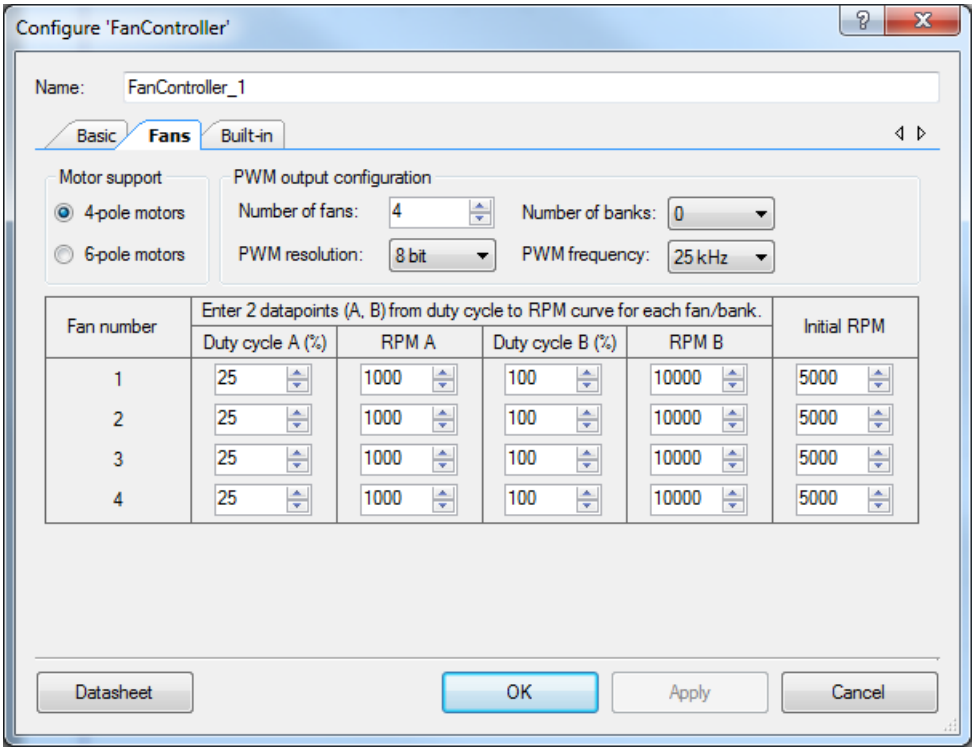
- **Lower limit –** This parameter defines the initial lower boundary for the integrator. It is presented as a ratio from 0.00% to 100.00%. This parameter can be used to set the lower limit for the fan; it also prevents integrator windup commonly seen in unbounded control systems. The value is applied to each fan in the configuration at start and can be changed later using the FanController_SetSaturation() API. This parameter should always be set lower than the upper saturation limit. The value can be changed individually for each fan.

Output attenuation

This is the attenuation defined in powers of 2. The range of the exponent is from 2^{-8} to 2^{-23}. The default gain is set to yield slow and stable performance for most systems (that is, low bandwidth PI control).

Fans Tab

This tab is used to configure fan-specific parameters.



This tab contains the following parameters:

Motor support

This parameter specifies the number of high – low pulses that appear on the fan's tachometer output per revolution. The 4-pole option means there are two high and two low pulses per fan revolution. The 6-pole option means that there are three high and three low pulses per fan revolution.

PWM output configuration

- **Number of fans –** This parameter specifies how many fans are in the system. The valid range for this parameter is 1 to 16. The default setting is 4.

  Note For Automatic Hardware (UDB) mode, the number of fans is limited to 12. For PSoC 4 devices, the number of fans is limited to four in Automatic Firmware (CPU) mode or six (using the banking feature) in manual mode.

- **Number of banks –** This parameter is only visible in firmware control mode. It specifies how many fan banks are in the system. It is assumed that when fans are organized into banks, each bank has the same number of fans. Therefore, the number of fans must be divisible by the number of banks. The value 0 indicates that the fans are not banked. For banked operation, the valid range for this parameter is from 1 to (Number of fans/2). The default setting is 0.

- **PWM resolution –** This parameter specifies the resolution of the duty cycle for the modulated PWM signal that drives the fans to control rotational speed. Valid options for this parameter are 8-bit or 10-bit. The default setting is 8-bit.

- **PWM frequency –** This parameter specifies the frequency of the modulated PWM signal that drives the fans. Valid options for this parameter are 25 kHz or 50 kHz when the internal clock is used. The default setting is 25 kHz. When an external clock is used, this parameter is grayed out since the PWM frequency depends on the input clock source. See the Functional Description section for more details.

Fan specifications

- **Duty cycle A (%), RPM A –** These parameters together specify one data point on the duty cycle-to-RPM transfer function for the selected fan or bank of fans. The RPM A parameter specifies the speed at which the fan nominally runs when driven by a PWM with a duty cycle of Duty A (%). This information is available from the fan manufacturer's datasheet. It should be noted that the Fan Controller Component may drive PWM duty cycles down to 0% even if Duty A (%) is set to a non-zero value.
  - The valid range for the Duty A (%) parameter is 0-99. The default setting is 25.
  - The valid range for the RPM A parameter is 500 to 24,999. The default setting is 1,000.

- **Duty cycle B (%), RPM B –** These parameters together specify a 2nd data point on the duty cycle-to-RPM transfer function for the selected fan or bank of fans. The RPM B parameter specifies the speed at which the fan nominally runs when driven by a PWM with a duty cycle of Duty B (%). This information is available from the fan manufacturer's datasheet. It should be noted that the Fan Controller Component may drive PWM duty cycles up to 100% even if Duty B (%) is set below 100%.
  - The valid range for the Duty B (%) parameter is 1-100. The default setting is 100.
  - The valid range for the RPM B parameter is 501 to 25,000. The default setting is 10,000.
- **Initial RPM –** This parameter specifies the initial RPM of an individual fan. The value of Initial RPM is converted into a duty cycle and set as the initial duty cycle for an individual fan. The Initial RPM parameter may be set lower than the RPM A parameter.

Clock Selection

The Component uses several clock tree sources for its operation. These clocks are: Bus Clock (not used for PSoC 4), Tachometer Clock (500 kHz), and PWM Clock (6, 12 or 24 MHz depending on the configuration). The Component has an option to connect an external clock source instead of the PWM Clock to create desirable PWM output frequency.

Note For the Component to operate with 10-bit PWM resolution in PSoC 4 devices, the IMO should be set to 48 MHz. due to clock restrictions.

Application Programming Interface

Application Programming Interface (API) routines allow you to interact with the Component using firmware. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name FanController_1 to the first instance of a Component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is FanController.

Functions

Function	Description
FanController_Start()	Start the Component
FanController_Stop()	Stop the Component and disable hardware blocks
FanController_Init()	Initializes the Component

Function	Description
FanController_Enable()	Enables hardware blocks inside the Component
FanController_EnableAlert()	Enables alerts from the Component
FanController_DisableAlert()	Disables alerts from the Component
FanController_SetAlertMode()	Configures alert sources
FanController_GetAlertMode()	Returns currently enabled alert sources
FanController_SetAlertMask()	Enables masking of alerts from each fan
FanController_GetAlertMask()	Returns alert masking status of each fan
FanController_GetAlertSource()	Returns pending alert source(s)
FanController_GetFanStallStatus()	Returns a bit mask representing the stall status of each fan
FanController_GetFanSpeedStatus()	Returns a bit mask representing the speed regulation status of each fan in hardware control mode
FanController_SetDutyCycle()	Sets the PWM duty cycle for the specified fan or fan bank
FanController_GetDutyCycle()	Returns the PWM duty cycle for the specified fan or fan bank
FanController_SetDesiredSpeed()	Sets the desired fans speed for the specified fan in hardware control mode
FanController_GetDesiredSpeed()	Returns the desired fans speed for the specified fan in hardware control mode
FanController_GetActualSpeed()	Returns the actual speed for the specified fan
FanController_OverrideAutomaticControl()	Enables firmware to override Automatic fan control
FanController_SetSaturation()	Changes the PID controller output saturation.
FanController_SetPID()	Changes the PID controller coefficients for the controlled fan.

void FanController_Start(void)

Description: Enables the Component. Calls the Init() API if the Component has not been initialized before. Calls the Enable() API.

Parameters: None

Return Value: None

Side Effects: None

void FanController_Stop(void)

Description: Disables the Component. All PWM outputs will be driven to 100% duty cycle to ensure cooling continues while the Component is not operational.

Note Due to PSoC 4 resource limitations, the PWM outputs will be cleared to low.

Parameters: None

Return Value: None

Side Effects: Alert pin is deasserted.

void FanController_Init(void)

Description: Initializes the Component.

Parameters: None

Return Value: None

Side Effects: None

void FanController_Enable(void)

Description: Enables hardware blocks within the Component.

Parameters: None

Return Value: None

Side Effects: None

void FanController_EnableAlert(void)

Description: Enables generation of the alert signal. Specifically which alert sources are enabled is configured using the FanController_SetAlertMode() and the FanController_SetAlertMask() APIs.

Parameters: None

Return Value: None

Side Effects: None

void FanController_DisableAlert(void)

Description: Disables generation of the alert signal.

Parameters: None

Return Value: None

Side Effects: Alert pin is de-asserted.

void FanController_SetAlertMode(uint8 alertMode)

Description: Configures alert sources from the Component. Two alert sources are available: 1) Fan Stall or Rotor Lock, 2) Hardware control mode speed regulation failure.

Parameters: uint8 alertMode

Bit Field	Enabled Alert Source
FanController_STALL_ALERT	1=Enable fan stall / rotor lock alert
FanController_SPEED_ALERT	1=Enable Automatic control speed regulation failure alert

Return Value: None

Side Effects: None

uint8 FanController_GetAlertMode(void)

Description: Returns the alert sources that are enabled.

Parameters: None

Return Value: uint8 alertMode

Bit Field	Enabled Alert Source
FanController_STALL_ALERT	1=Enable fan stall / rotor lock alert
FanController_SPEED_ALERT	1=Enable Closed Loop speed regulation failure alert

Side Effects: None

void FanController_SetAlertMask(uint16 alertMask)

Description: Enables or disables alerts from each fan through a mask. Masking applies to both fan stall alerts and speed regulation failure alerts.

Parameters: uint16 alertMask

Bit Field	Enabled Alert Source
bit0	1=Enable alerts for Fan1
bit1	1=Enable alerts for Fan2
…	…
bit15	1=Enable alerts for Fan16

Return Value: None

Side Effects: None

uint16 FanController_GetAlertMask(void)

Description: Returns alert mask status from each fan. Masking applies to both fan stall alerts and speed regulation failure alerts.

Parameters: None

Return Value: uint16 alertMask

Bit Field	Enabled Alert Source
bit0	1=Enable alerts for Fan1
bit1	1=Enable alerts for Fan2
…	…
bit15	1=Enable alerts for Fan16

Side Effects: None

uint8 FanController_GetAlertSource(void)

Description: Returns pending alert sources from the Component. This API can be used to poll the alert status of the Component. Alternatively, if the alert pin is used to generate interrupts to PSoC's CPU core, the interrupt service routine can use this API to determine the source of the alert. In either case, when this API returns a non-zero value, the FanController_GetFanStallStatus() and FanController_GetFanSpeedStatus() APIs can provide further information on which fan(s) has(have) a fault.

Parameters: uint8 alertMode

Bit Field	Pending Alert
FanController_STALL_ALERT	1=Fan stall / rotor lock alert pending
FanController_SPEED_ALERT	1=Closed Loop speed regulation failure alert pending

Return Value: None

Side Effects: Calling this API de-asserts the alert pin. If any alerts are pending, the alert pin is re-asserted right after the next end-of-cycle (eoc) pulse.

uint16 FanController_GetFanStallStatus(void)

Description: Returns the stall / rotor lock status of all fans.

Parameters: None

Return Value: uint16 stallStatus

Bit Field	Status
bit0	Fan1 stall status (1=stall, 0=OK)
bit1	Fan2 stall status
…	…
bit15	Fan16 stall status

Side Effects: Calling this API clears all pending fan stall alerts.

uint16 FanController_GetFanSpeedStatus(void)

Description: Returns the hardware fan control mode speed regulation status of all fans. Speed regulation failures occur in two cases: 1) if the desired fan speed exceeds the current actual fan speed but the fan's duty cycle is already at 100%, 2) if the desired fan speed is below the current actual fan speed, but the fan's duty cycle is already at 0%.

Parameters: None

Return Value: uint16 speedStatus

Bit Field	Status
bit0	Fan1 speed regulation status (1=failure, 0=OK)
bit1	Fan2 speed regulation status
…	…
bit15	Fan16 speed regulation status

Side Effects: Calling this API clears all pending speed regulation alerts.

void FanController_SetDutyCycle(uint8 fanOrBankNumber, uint16 dutyCycle)

Description: Sets the PWM duty cycle of the selected fan or fan bank in hundredths of a percent. In hardware fan control mode, if manual duty cycle control is desirable, call the FanController_OverrideHardwareControl() API prior to calling this API.

Note Due to PSoC 4 resource limitations, setting 100% duty cycle will not generate a continuous high signal on respective "fan" output.

Parameters: uint8 fanOrBankNumber

Fan or bank number. Valid range is 1..16 but should not exceed the number of fans or banks in the system.

uint16 dutyCycle

Duty cycle in hundredths of a percent. For example, 50% duty cycle = 5000. Valid range is 0..10000.

Return Value: None

Side Effects: None

uint16 FanController_GetDutyCycle(uint8 fanOrBankNumber)

Description: Returns the current PWM duty cycle of the selected fan or fan bank in hundredths of a percent.

Parameters: uint8 fanOrBankNumber

Fan or bank number. Valid range is 1..16 but should not exceed the number of fans or banks in the system.

Return Value: Duty cycle in hundredths of a percent. For example, 50% duty cycle = 5000.

Side Effects: None

void FanController_SetDesiredSpeed(uint8 fanNumber, uint16 rpm)

Description: Sets the desired speed of the specified fan in revolutions per minute (RPM). In hardware fan control mode, the RPM parameter is passed to the control loop hardware as the new target fan speed for regulation. In firmware fan control mode, the RPM parameter is converted to a duty cycle based on the fan parameters entered into the Fans tab of the customizer and written to the appropriate PWM. This provides firmware with a method for initiating coarse level speed control. Fine level firmware speed control can then be achieved using the FanController_SetDutyCycle() API.

Parameters: uint8 fanNumber

Fan or bank number. Valid range is 1..16 but should not exceed the number of fans in the system.

uint16 rpm

Valid range is 500..25,000 but should not exceed the maximum RPM that the fan is capable of running at. Doing so will cause a speed regulation failure.

Return Value: None

Side Effects: None

uint16 FanController_GetDesiredSpeed(uint8 fanNumber)

Description: Returns the currently desired speed for the selected fan.

Parameters: uint8 fanNumber

Fan or bank number. Valid range is 1..16 but should not exceed the number of fans in the system.

Return Value: Currently desired speed for the selected fan in RPM

Side Effects: None

uint16 FanController_GetActualSpeed(uint8 fanNumber)

Description: Returns the current actual speed for the selected fan.

Parameters: uint8 fanNumber

Fan number. Valid range is 1..16 but should not exceed the number of fans in the system.

Return Value: Current actual speed for the selected fan in RPM

Side Effects: None

void FanController_OverrideAutomaticControl(uint8 override)

Description: Allows firmware to take over fan control in hardware fan control mode. Note that this API cannot be called in firmware fan control mode.

Parameters: uint8 override

0 = hardware assumes control of fans

1 = firmware assumes control of fans

Valid range is 0..1. Default is 0

Return Value: None

Side Effects: None

void FanController_SetSaturation(uint8 fanNum, uint16 satH, uint16 satL)

Description: Changes the PID controller output saturation. This bounds the output PWM to the fan and prevents what is known as integrator windup.

Parameters: uint8 fanNum

Fan number. Valid range is 1..16 but should not exceed the number of fans in the system.

uint16 satH

The upper threshold for saturation. Valid range is 0 to 65535. A value of 0 represents 0% of the duty cycle. A value of 65535 represents 100% duty cycle.

uint16 satL

The lower threshold for saturation. Valid range is 0 to 65535. A value of 0 represents 0% of the duty cycle. A value of 65535 represents 100% duty cycle.

Return Value: None

Side Effects: None

void FanController_SetPID (uint8 fanNum, uint16 kp, uint16 ki, uint16 kd)

Description: Changes the PID controller coefficients for the controlled fan. The coefficients are integers that are proportional to the gain.

Parameters: uint8 fanNum: Fan number. Valid range is 1..16 but should not exceed the number of fans in the system.

uint16 kp: Proportional gain. Valid range is 0 to 65535. A value of 0 represents 0% gain. A value of 65535 represents 100% gain.

uint16 ki: Integral gain. Valid range is 0 to 65535. A value of 0 represents 0% gain. A value of 65535 represents 100% gain.

uint16 kd: Derivative gain. Valid range is 0 to 65535. A value of 0represents 0% gain. A value of 65535 represents 100% gain.

Return Value: None

Side Effects: None

Global Variables

Variable	Description
FanController_initVar	The initVar variable is used to indicate initial configuration of this Component. This variable is pre-pended with the Component name. The variable is initialized to zero and set to 1 the first time FanController_Start() is called. This allows for Component initialization without re-initialization in all subsequent calls to the FanController_Start() routine. If re-initialization of the Component is required the FanController_Stop() routine should be called followed by the FanController_Init() and FanController_Enable().

Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the Component's generated source files, perform the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*). This will "uncomment" the function call from the Component's source code.
- Write the function declaration (in *cyapicallbacks.h*). This will make this function visible by all the project files.
- Write the function implementation (in any user file).

Callback Function [1]	Associated Macro	Description
FanController_DATA_SEND_ISR_EntryCallback	FanController_DATA_SEND_ISR_ENTRY_CALLBACK	Used at the beginning of the FanController_DATA_SEND_ISR_() interrupt handler to perform additional application-specific actions.
FanController_DATA_SEND_ISR_ExitCallback	FanController_DATA_SEND_ISR_EXIT_CALLBACK	Used at the end of the FanController_DATA_SEND_ISR_() interrupt handler to perform additional application-specific actions.
FanController_PID_CONTROL_ISR_EntryCallback	FanController_PID_CONTROL_ISR_ENTRY_CALLBACK	Used at the beginning of the FanController_PID_CONTROL_ISR() interrupt handler to perform additional application-specific actions.
FanController_PID_CONTROL_ISR_ExitCallback	FanController_PID_CONTROL_ISR_EXIT_CALLBACK	Used at the end of the FanController_ISR4() interrupt handler to perform additional application-specific actions.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Fan Controller Component has the following specific deviations:

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
10.3	R	A cast should not be performed between a pointer to object type and a different pointer to object type.	When Automatic Firmware mode is used the PID algorithm casts signed integer variables to unsigned.

[1] The callback function name is formed by Component function name optionally appended by short explanation and “Callback” suffix.

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
13.7	R	Boolean operations whose results are invariant shall not be permitted .	Depending on the Component configuration there can be condition checks which conditions are invariant.
19.7	A	A function should be used in preference to a function-like macro.	A function-like macro FanController_OverrideHardwareControl() was added for support of existing designs as the function FanController_OverrideHardwareControl() was renamed to FanController_OverrideAutomaticControl()
21.1	R	Minimization of run-time failures shall be ensured by the use of at least one of the following: a) Static analysis tools/ techniques b) Dynamic analysis tools/ techniques c) Explicit coding of checks to handle run-time faults	Depending on the Component configuration there can be condition checks which conditions are invariant as the result some part of code can be redundant.

This Component has the following embedded Components: DMA, ISR. Refer to the corresponding Component datasheet for information on their MISRA compliance and specific deviations.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For Component-specific examples, open the dialog from the Component Catalog or an instance of the Component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

Interrupt Service Routine

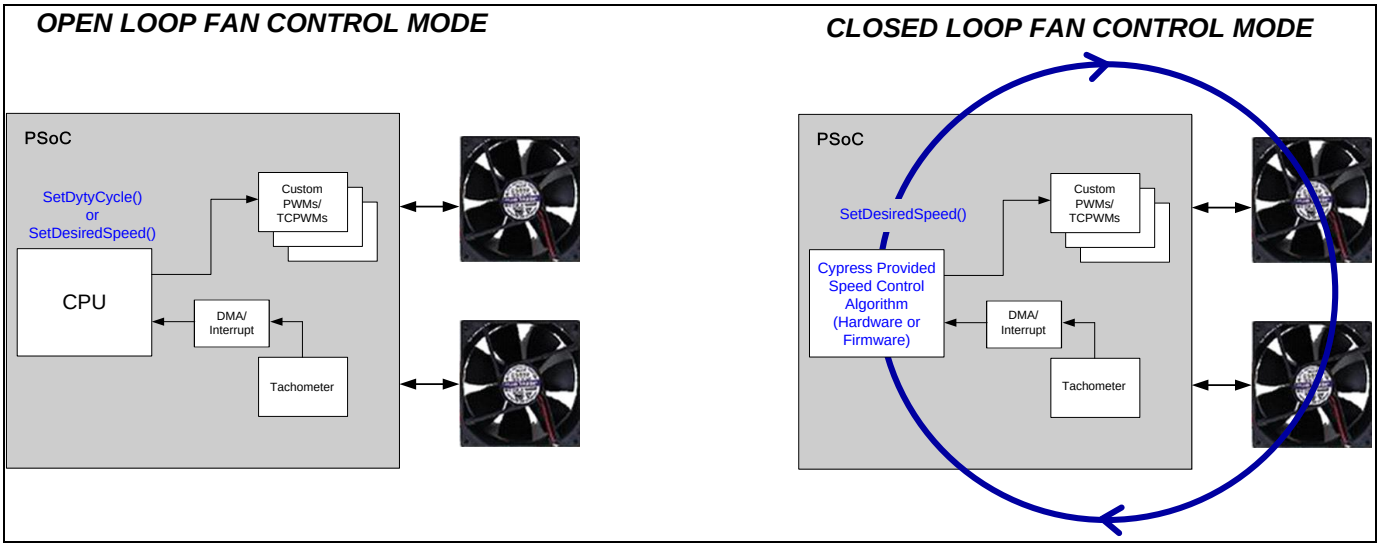
The Fan Controller Component can utilize two interrupts depending on the mode it is configured with or the device which is used. When it is configured in Automatic Firmware (CPU) mode the Component uses *FanController_PID_ISR* interrupt. This interrupt implements a PID Control algorithm. An addition interrupt - *FanController_DataSend* is used when Component is utilized in PSoC4 design. This interrupt is responsible for transferring measured RPM speed from tachometer to RAM.

Note For PSoC 4, if there are any other interrupts used in the design except Fan Controller interrupts it is important that *FanController_DataSend* interrupt have highest priority than other interrupts. This is because of the small time window during which the *FanController_DataSend* should read the current active fan address and locate read value of actual speed to a proper RAM location.

Functional Description

Block Diagram and Configuration

The schematic shown in Figure below shows high level block diagrams of the two fundamental modes of the Component: 1) open loop control mode (Manual) and 2) closed loop control mode (Automatic).



Manual (Open Loop) Control mode is straightforward and it was described in Component Parameters section.

Automatic (Closed Loop) Control Mode

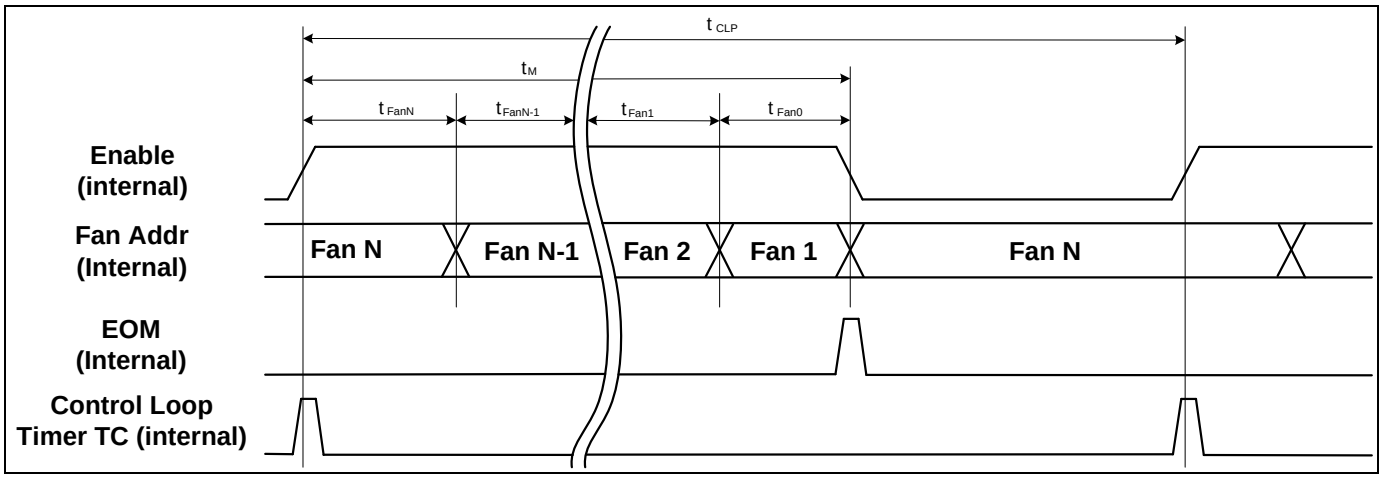
In Automatic Control mode the controlling algorithm is hidden from the user but the Component provides a set of parameters for configuring the algorithm selected. Although this algorithm doesn't require almost any firmware development it does require some basic knowledge on Component operation to be able to configure it properly.

Control Loop Period

Control Loop Period provides a programmable delay between fans speeds adjustments. The main purpose on this delay is to give fans time to stabilize which prevents them from oscillating. The Control Loop Period is optional for Automatic Hardware (UDB) mode as in case when Component is configured for many fans (eg. 12 or more) the measurement of that number of

fans introduces a natural delay so the Control Loop Period may not be needed. For Automatic Firmware (CPU) mode the Control Loop Period (t_{CLP}) is required and it should be always be greater than time consumed for speed measure of all fans (t_M), while in Automatic Hardware (UDB) t_{CLP} can be smaller than t_M.

The following timing diagram illustrates the Control Loop Period generation.



On the diagram above the internal enable signal is set to high when terminal count of internal control loop timer is set high. This enable signal starts a sequence of speed measurement for all fans. The sequence is stopped, with asserting enable signal to low, when internal End-Of-Measurement (EOM) signal pulses.

Control Loop Period Calculation

For Automatic Hardware (UDB) mode the control loop period make sense only if it is greater than time for measurement of all fans. For Automatic Firmware (CPU) mode it is vital that $t_{CLP} > t_M$ as each terminal count of control loop period timer the PID ISR is triggered to run the PID algorithm and it that moment all the measured actual speed values for all fans must be available.

The t_M can be calculated from the formula:

$$t_M = t_{Fan1} + t_{Fan2} + .. + t_{FanN} ; \quad (1)$$

Where t_{Fan1} ... t_{FanN} is time periods required to measure speed for Fan1 .. FanN. The time that is required for measuring speed for the fan is defined by the following formula:

$$t_{Fan} = 1.75 * (60/RPM_{Fan_min}); \quad (2)$$

In this formula coefficient 1.75 dictated by the hardware as 1.75 of fan's rotation is required to measure the speed. RPM_{Fan_min} is the minimal speed for the specific fan.

Example. For configuration of 4 fans that all have RPM_{Fan_min} = 1000 the result will be following:

$$t_M = 4 * 1.75 * (60/1000) = 0.42 \text{ sec}$$

But $t_{CLP} > t_M$ so the minimal value of the Control Loop Period for this configuration is 0.43 sec.

Custom Clock

The Component has a feature that allows the connection of an external clock. The frequency of the clock source determines the PWM output frequency, and the relation between the input clock frequency (f_{CLK}) and output PWM frequency (f_{PWM}) is shown in the following equations:

$$f_{PWM} = f_{CLK} / P_{PWM}; \quad (3)$$

Where P_{PWM} is the PWM period. For 8 bit resolution P_{PWM} equals to 240 and for P_{PWM} it equals to 960.

PID Algorithm

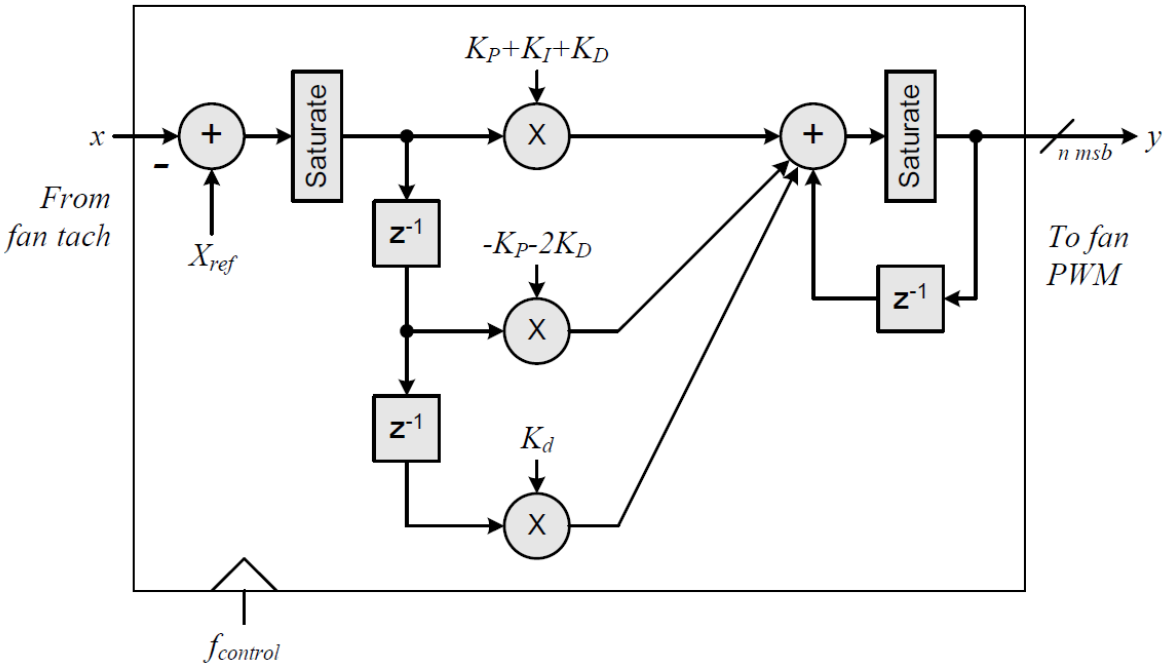
The Automatic Firmware CPU control mode uses proportional-integral-derivative (PID) algorithm for controlling fans. The algorithm is defined in the following equation:

$$G_{PID}[z] = K_P + K_I * (1 / (1 - z^{-1})) + K_D * (1 - z^{-1}); \quad (4)$$

Although Equation 4 could be implemented in code directly, the result would yield code that is quite abusive on PSoC's processing and memory resources. However, Equation 4 can be algebraically simplified to a form that is far more efficient for implementation and is shown in Equation 5:

$$G_{PID}[z] = ((K_P + K_I + K_D) + (- K_P + 2 * K_D) * z^{-1} + * (K_D) * z^{-2}) / (1 - z^{-1}); \quad (5)$$

This transfer function represents the desired PID algorithm for the fan controller. Note, the derivative term is likely unnecessary for most fan control applications, but it is maintained in the event that a yet to be known fan system requires the extra compensation. Figure below shows the corresponding signal flow diagram for the transfer function presented in Equation 5.



To help understand, the following table contains a brief summary of the inputs and outputs of the PID control algorithm.

Signal	Direction	Note
PID_refference (X_{ref})	Input	The reference the control regulates in case of Fan Controller it is the desired speed in RPM.
PID_in (x)	Input	The signal being controlled, in this case the measured fan speed in RPM.
PID_error_saturation_high	Input	The saturation boundary for the input error signal. These are internal constants hardcoded to 4096(high) and -4096(low). These values are reasonably large to account for a wide error input and small enough to fit well within the controller computational limits.
PID_error_saturation_low		
PID_A1	Input	The controller coefficients where A1, A2, and A3 are functions of KP, KI, and KD. These coefficients should are available for changing at run-time through FanController_SetPID() function for the situation where compensation may need tweaking for a very non-linear fan response. However, in most cases these parameters are expected to be constant. The coefficients are signed integers limited to < 16 bits.
PID_A2		
PID_A3		
PID_output_saturation_high	Input	The PID output should be limited to the dynamic range of the system being controlled, the PWM driving the fan in this case. These set the upper and lower boundaries for the integrator (prevents integrator windup). These parameters are available to the user for changing at runtime through FanController_SetSaturation() function. Note that the saturation is scaled by the PID_POST_GAIN term.
PID_output_saturation_low		
PID_POST_GAIN (G_O)	Input	The output gain for the controller. Used in computation of output saturation high and low limits. This parameter is inversely proportional to output attenuation.
PID_out (y)	output	The output of the PID sent to the PWM that drives the fan.

The coefficients used in the algorithm are defined by Equation 6, 7, and 8.

$$PID_A1 = (K_P + K_I + K_D) * 2^{12} ; \quad (6)$$

$$PID_A2 = -(K_P + 2 * K_D) * 2^{12} ; \quad (7)$$

$$PID_A3 = (K_D) * 2^{12} ; \quad (8)$$

Note The equations impose coefficient results in the range of 12 to 13 bits with a maximum no higher than 13 bits. Thus the combined results from the feed-forward terms are not sufficient to cause an overflow in the controller integration stage, certainly not with small error.

The maximum and minimum saturation levels for the output saturation are set by the customizer generated upper limit (YH), lower limit (YL), the PWM period mentioned in previous section (PPWM), and the output gain (GO).

$$PID_output_saturation_high = (Y_H * P_{PWM})/G_O; \quad (9)$$

$$PID_output_saturation_low = (Y_L * P_{PWM})/G_O; \quad (10)$$

Registers

The Fan Controller has several control and status registers that are used by the firmware APIs to control operation and monitor status. None of these registers should be directly accessible by user firmware.

Component Debug Window

The Fan Controller Component supports the PSoC Creator Component debug window. The following registers are displayed in the Fan Controller Component debug window.

FanController_GLBL_CTRL

This is Component's Global Control register.

FanController_ALERT_STATUS

This is Component's Alert status register.

FanController_STALL_STATUS_LSB

This register holds status of stall alerts for fans 1-8.

FanController_MSB_STALL_STATUS_MSB

This register holds status of stall alerts for fans 9-16.

FanController_ALRT_MASK_LSB

This register holds status of speed alerts for fans 1-8.

FanController_MSB_ALRT_MASK_MSB

This register holds status of stall speed for fans 9-16.

Resources

The Fan Controller Component is placed throughout the UDB array. The following table shows UDB resources t utilized by the Component.

PSoC 3/PSoC 5LP

Configuration[2][3]	Resource Type					
	Datapath Cells	Macrocells[4]	Status Cells	Control Cells	DMA Channels	Interrupts
Manual mode, 4 Fans	3 (7)	32	3	3	1	–
Manual mode, 8 Fans	7 (11)	40	3	3	1	–
Manual mode, 12 Fans	9 (15)	49	4	4	1	–
Manual mode, 16 Fans	11 (19)	59	4	4	1	–
Auto Firmware mode, 4 Fans	4 (8)	37	3	3	1	–
Auto Firmware mode, 8 Fans	8 (12)	45	3	3	1	–
Auto Firmware mode, 12 Fans	10 (16)	54	4	4	1	–
Auto Firmware mode, 16 Fans	12 (20)	64	4	4	1	–
Auto Hardware mode, 4 Fans	7 (11)	61	4	7	2	–
Auto Hardware mode, 8 Fans	11 (19)	97	4	11	2	–
Auto Hardware mode, 12 Fans	15	135	6	16	2	–

PSoC 4

Configuration	Resource Type						
	Datapath Cells	Macrocells	Status Cells	Control Cells	DMA Channels	TCPWM Blocks	Interrupts
Manual mode, 4 Fans	3	19	2	3	–	4	1
Auto Firmware mode, 4 Fans	4	25	2	3	–	4	2

2 Table demonstrates resource usage for 8-bit resolution. Resources that are occupied by 10-bit configuration if they differ are shown in parentheses.

3 For the Hardware mode, resources used by the Control Loop Period feature are not included.

4 The numbers are shown for PSoC 3 device. For PSoC 5LP the Component utilizes 3 macrocells less than for PSoC 3. It's because of the fix for simultaneous CPU and DMA multi-byte access issue.

API Memory Usage

The Component memory usage varies significantly, depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration[5][6]	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Manual mode, 4 Fans	1689 (1718)	82	1440	65	792 (816)	70
Manual mode, 8 Fans	1705 (1734)	162	N/A	N/A	804 (820)	138
Manual mode, 12 Fans	1738 (1767)	242	N/A	N/A	872 (896)	206
Manual mode, 16 Fans	1754 (1783)	322	N/A	N/A	900 (932)	274
Auto Firmware mode, 4 Fans	3368 (3483)	186	2088	169	1368 (1384)	174
Auto Firmware mode, 8 Fans	3441 (3499)	366	N/A	N/A	1380 (1400)	342
Auto Firmware mode, 12 Fans	3454 (3485)	546	N/A	N/A	1456 (1480)	510
Auto Firmware mode, 16 Fans	3500 (3679)	726	N/A	N/A	1488 (1504)	678
Auto Hardware mode, 4 Fans	2567 (2581)	115	N/A	N/A	1120 (1132)	103
Auto Hardware mode, 8 Fans	2599 (2613)	227	N/A	N/A	1188 (1196)	203
Auto Hardware mode, 12 Fans	2652	339	N/A	N/A	1316	303

[5] Table demonstrates resource usage for 8-bit resolution. Resources that are occupied by 10 bit configuration if they differ are shown in parentheses.

[6] For the Hardware mode, resources used by the Control Loop Period feature are not included.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Characteristics

PSoC 3/PSoC 5LP (PWM clock – 6 MHz, BUS_CLK – 24 MHz)

Parameter	Description	Min	Typ[7]	Max	Unit
Idd	Component current consumption(Manual mode, 4 Fans)				
	8-bit resolution	–	108	–	µA
	10-bit resolution	–	139	–	µA
	Component current consumption(Manual mode, 8 Fans)				
	8-bit resolution	–	194	–	µA
	10-bit resolution	–	260	–	µA
	Component current consumption(Manual mode, 12 Fans)				
	8-bit resolution	–	307	–	µA
	10-bit resolution	–	395	–	µA
	Component current consumption(Manual mode, 16 Fans)				
	8-bit resolution	–	392	–	µA
	10-bit resolution	–	505	–	µA
	Component current consumption(Automatic Firmware mode, 4 Fans)				
	8-bit resolution	–	180	–	µA
	10-bit resolution	–	221	–	µA
	Component current consumption(Automatic Firmware mode, 8 Fans)				
	8-bit resolution	–	264	–	µA
	10-bit resolution	–	319	–	µA
	Component current consumption(Automatic Firmware mode, 12 Fans)				
	8-bit resolution	–	329	–	µA
	10-bit resolution	–	428	–	µA

7 Device I/O and clock distribution current not included. Other factors such as routing conditions, frequency of tach inputs and temperature also have an impact on the current consumption. The values are at 25 °C.

Parameter	Description	Min	Typ[7]	Max	Unit
Idd (cont.)	Component current consumption(Automatic Firmware mode, 16 Fans)				
	8-bit resolution	–	385	–	µA
	10-bit resolution	–	508	–	µA
	Component current consumption(Automatic Hardware mode, 4 Fans)				
	8-bit resolution	–	225	–	µA
	10-bit resolution	–	269	–	µA
	Component current consumption(Automatic Hardware mode, 8 Fans)				
	8-bit resolution	–	417	–	µA
	10-bit resolution	–	518	–	µA
	Component current consumption(Automatic Hardware mode, 12 Fans)				
	8-bit resolution	–	636	–	µA

PSoC 3/PSoC 5LP (PWM clock – 12 MHz, BUS_CLK – 24 MHz)

Parameter	Description	Min	Typ[8]	Max	Unit
Idd	Component current consumption(Manual mode, 4 Fans)				
	8-bit resolution	–	169	–	µA
	10-bit resolution	–	233	–	µA
	Component current consumption(Manual mode, 8 Fans)				
	8-bit resolution	–	310	–	µA
	10-bit resolution	–	439	–	µA
	Component current consumption(Manual mode, 12 Fans)				
	8-bit resolution	–	498	–	µA
	10-bit resolution	–	677	–	µA
	Component current consumption(Manual mode, 16 Fans)				
	8-bit resolution	–	641	–	µA
	10-bit resolution	–	871	–	µA

8 Device I/O and clock distribution current not included. Other factors such as routing conditions, frequency of tach inputs and temperature also have an impact on the current consumption. The values are at 25 °C.

Parameter	Description	Min	Typ[8]	Max	Unit
Idd (cont.)	Component current consumption(Automatic Firmware mode, 4 Fans)				
	8-bit resolution	–	246	–	µA
	10-bit resolution	–	307	–	µA
	Component current consumption(Automatic Firmware mode, 8 Fans)				
	8-bit resolution	–	360	–	µA
	10-bit resolution	–	486	–	µA
	Component current consumption(Automatic Firmware mode, 12 Fans)				
	8-bit resolution	–	512	–	µA
	10-bit resolution	–	695	–	µA
	Component current consumption(Automatic Firmware mode, 16 Fans)				
	8-bit resolution	–	618	–	µA
	10-bit resolution	–	870	–	µA
	Component current consumption(Automatic Hardware mode, 4 Fans)				
	8-bit resolution	–	403	–	µA
	10-bit resolution	–	492	–	µA
	Component current consumption(Automatic Hardware mode, 8 Fans)				
	8-bit resolution	–	756	–	µA
	10-bit resolution	–	951	–	µA
	Component current consumption(Automatic Hardware mode, 12 Fans)				
	8-bit resolution	–	1162	–	µA

PSoC 3/PSoC 5LP (PWM clock – 24 MHz, BUS_CLK – 24 MHz)

Parameter	Description	Min	Typ[9]	Max	Unit
Idd	Component current consumption(Manual mode, 4 Fans)				
	8-bit resolution	–	290	–	µA
	10-bit resolution	–	417	–	µA
Idd (cont.)	Component current consumption(Manual mode, 8 Fans)				
	8-bit resolution	–	545	–	µA
	10-bit resolution	–	798	–	µA
	Component current consumption(Manual mode, 12 Fans)				
	8-bit resolution	–	887	–	µA
	10-bit resolution	–	1243	–	µA
	Component current consumption(Manual mode, 16 Fans)				
	8-bit resolution	–	1150	–	µA
	10-bit resolution	–	1607	–	µA
	Component current consumption(Automatic Firmware mode, 4 Fans)				
	8-bit resolution	–	354	–	µA
	10-bit resolution	–	494	–	µA
	Component current consumption(Automatic Firmware mode, 8 Fans)				
	8-bit resolution	–	614	–	µA
	10-bit resolution	–	906	–	µA
	Component current consumption(Automatic Firmware mode, 12 Fans)				
	8-bit resolution	–	897	–	µA
	10-bit resolution	–	1246	–	µA
	Component current consumption(Automatic Firmware mode, 16 Fans)				
	8-bit resolution	–	1089	–	µA
	10-bit resolution	–	1671	–	µA
	Component current consumption(Automatic Hardware mode, 4 Fans)				
	8-bit resolution	–	753	–	µA
	10-bit resolution	–	938	–	µA

9 Device I/O and clock distribution current not included. Other factors such as routing conditions, frequency of tach inputs and temperature also have an impact on the current consumption. The values are at 25 °C.

Parameter	Description	Min	Typ[9]	Max	Unit
	Component current consumption(Automatic Hardware mode, 8 Fans)				
	8-bit resolution	–	1446	–	µA
	10-bit resolution	–	1826	–	µA
	Component current consumption(Automatic Hardware mode, 12 Fans)				
	8-bit resolution	–	2220	–	µA

PSoC 4 (PWM clock – 6 MHz, BUS_CLK – 24 MHz)

Parameter	Description	Min	Typ	Max	Unit
Idd	Component current consumption (Manual mode, 4 Fans)	–	605	–	µA

PSoC 4 (PWM clock – 12 MHz, BUS_CLK – 24 MHz)

Parameter	Description	Min	Typ	Max	Unit
Idd	Component current consumption	–	900	–	µA

PSoC 4 (PWM clock – 6 MHz, BUS_CLK – 48 MHz)

Parameter	Description	Min	Typ	Max	Unit
Idd	Component current consumption	–	670	–	µA

PSoC 4 (PWM clock – 12 MHz, BUS_CLK – 48 MHz)

Parameter	Description	Min	Typ	Max	Unit
Idd	Component current consumption	–	1010	–	µA

PSoC 4 (PWM clock – 24 MHz, BUS_CLK – 48 MHz)

Parameter	Description	Min	Typ	Max	Unit
Idd	Component current consumption	–	1200	–	µA

AC Specifications

PSoC 3/PSoC 5LP

Parameter	Description	Min	Typ	Max	Unit
f_{pwm_clk}	Input clock frequency[10]				
	Manual mode, 8-bit, 4 Fans	–	–	51	MHz
	Manual mode, 8-bit, 8 Fans	–	–	45	MHz
	Manual mode, 8-bit, 12 Fans	–	–	51	MHz
	Manual mode, 8-bit, 16 Fans	–	–	47	MHz
	Manual mode, 10-bit, 4 Fans	–	–	44	MHz
	Manual mode, 10-bit, 8 Fans	–	–	43	MHz
	Manual mode, 10-bit, 12 Fans	–	–	43	MHz
	Manual mode, 10-bit, 16 Fans	–	–	43	MHz
	Automatic Firmware mode, 8-bit, 4 Fans	–	–	51	MHz
	Automatic Firmware mode, 8-bit, 8 Fans	–	–	46	MHz
	Automatic Firmware mode, 8-bit, 12 Fans	–	–	46	MHz
	Automatic Firmware mode, 8-bit, 16 Fans	–	–	56	MHz
	Automatic Firmware mode, 10-bit, 4 Fans	–	–	44	MHz
	Automatic Firmware mode, 10-bit, 8 Fans	–	–	44	MHz
	Automatic Firmware mode, 10-bit, 12 Fans	–	–	44	MHz
	Automatic Firmware mode, 10-bit, 16 Fans	–	–	44	MHz
	Automatic Hardware mode, 8-bit, 4 Fans	–	–	55	MHz
	Automatic Hardware mode, 8-bit, 8 Fans	–	–	44	MHz
	Automatic Hardware mode, 8-bit, 12 Fans	–	–	46	MHz
	Automatic Hardware mode, 10-bit, 4 Fans	–	–	50	MHz
	Automatic Hardware mode, 10-bit, 8 Fans	–	–	50	MHz
f_{tach_clk}	Tachometer clock frequency	–	500	–	kHz
Tach Resolution	Tachometer counter resolution	–	16	–	Bits

[10] The values provide a maximum safe operating PWM frequency for the Fans. The Component may run at higher clock frequencies, at which point you will need to validate the timing requirements with Static Timing Analysis results.

Parameter	Description	Min	Typ	Max	Unit
f_{tach}	Tachometer Speed	500[11]	–	25000	RPM

PSoC 4

Parameter	Description	Min	Typ	Max	Unit
f_{pwm_clk}	Input clock frequency[12]	–	–	48	MHz

Component Errata

This section lists known problems with the Component.

Cypress ID	Component Version	Problem	Workaround
169327	4.0	On PSoC 4, if there are any other interrupts used in the design except FanController_DataSend interrupt or CyEnterCriticalSection() is called, the glitches in speed measurements can be observed. This is because of the small time window (equal to SYSCLK / 500 kHz CPU cycles) during which the FanController_DataSend should read the current active fan address and store the read value of the actual speed to a proper RAM location.	FanController_DataSend interrupt must always be enabled and has the highest priority. Ensure that the SYSCLK is sufficiently high for the CPU to respond during the specified time window.

11 Speed less than the minimum will be interpreted as halted.

12 The values provide a maximum safe operating PWM frequency for the Fans. The Component may run at higher clock frequencies, at which point you will need to validate the timing requirements with Static Timing Analysis results.

Cypress ID	Component Version	Problem	Workaround
266959	4.0	When the Component is used in multi-fan configuration, and when one or more “tach” inputs are tied to LOW or physically unconnected, the Component may measure an incorrect speed value for a neighbor fan with a lower index. For example, for a four fan configuration when “tach4” is physically unconnected, an incorrect speed value can be measured for fan#3. The speed for the other fans will be measured correctly. The issue is caused by the flaw in the tachometer state machine, which is a part of the Component implementation.	The “tach” inputs should never be physically unconnected; if they are, they should be tied to high. The issue might also be happen when some “tach” inputs are stalled to LOW, or a corresponding fan rotates with a speed lower than the stall threshold (500 RPM). To prevent incorrect speed measurements, use the banking option with extra “tach” inputs. For example, when controlling two fans select number of fans = 4 and number of banks = 2. Connect “tach1” and “tach3” inputs to the tachometer outputs of the controlled fans. Connect “tach2” and “tach4” to VDD or the logic HIGH signal. Connect “bank1” and “bank2” to the “drive” inputs of the controlled fans.

Component Changes

This section lists the major changes in the Component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
4.0.f	Updated datasheet.	Added errata item 266959.
4.0.e	Updated datasheet.	Added notes about limitations on PSoC 4 devices.
4.0.d	Updated datasheet.	Removed note that the data for the PSoC 4200L device is preliminary.
4.0.c	Updated datasheet.	Added note that the data for the PSoC 4200L device is preliminary.
4.0.b	Updated datasheet.	Added Macro Callbacks section.
4.0.a	Edited datasheet.	Added Errata section.
4.0	Added support for PSoC 4200 BLE devices.	Updates to support PSoC 4200 BLE devices.
3.10	Corrected the Component changes made in PSoC Creator 3.0 SP1.	See Knowledge Base Article KBA94159 (www.cypress.com/go/kba94159).
3.0.b	Component version 3.0 is obsolete and can no longer be used.	This Component version was modified without a version number change in PSoC Creator 3.0 SP1.

Version	Description of Changes	Reason for Changes / Impact
3.0.a	Updated datasheet descriptions for alert output, fan stall/rotor lock, and speed regulation sections.	To add more details about how the alerts work.
3.0	Updated Resources, API Usage, Functional Description, DC and AC Electrical Characteristics sections.	
	Fixed the FanController_GetActualSpeed() API.	The function would sometimes return incorrect measured speed values.
	Removed a note about the FanController_GetActualSpeed() being called within a specific time window after EOC pulse on PSoC 3.	The reason for this note is related to simultaneous multi-byte data access by the CPU and DMA. The current version of the Component fixes this issue.
	Added a description of control loop period.	
	Added two new API functions: FanController_SetSaturation() and FanController_PID().	These functions are required to support new Component's mode.
	Renamed FanController_OverrideHardwareControl() to FanController_OverrideAutomaticControl().	The old name became obsolete with introduction of new mode.
	Removed PSoC 5 support.	
	Added support of PSoC 4 family devices.	To provide Fan Controlling capability to PSoC 4.
	Firmware (CPU) mode was renamed to Manual.	This mode was not performing actual fan control and instead a task of implementing fan control algorithm was relied on the user. The new mode - Automatic Firmware (CPU) uses internal PID Control algorithm to control fans.
	New control mode - Automatic Firmware (CPU) was added to the Component. This mode is based on the PID control algorithm.	New feature.
2.30.a	Label "Damping factor (sec)" renamed to "Control loop period (sec)".	Since a corresponding parameter is a time quantity, term "Control loop period" is better.
2.30	Added MISRA Compliance section.	The Component was not verified for MISRA compliance.
	Updated Fan Controller with the latest version of the clock and DMA Components	
2.20	Updated Side Effects section in description of FanController_GetFanSpeedStatus(), FanController_GetFanStallStatus() and FanController_GetAlertSource() functions.	

Version	Description of Changes	Reason for Changes / Impact
	Fixed an issue related to the alert pin, which did not behave as described in the datasheet.	The alert signal was asserting high for a duration of one input clock cycle at each "eoc"; it was expected to go up until it was cleared by the FanController_GetFanSpeedStatus() or FanController_GetFanStallStatus() functions. Now, it has been adjusted to go up until it is cleared by the FanController_GetAlertSource().
	Fixed a bug related to the Component operation in Firmware Mode in which the Component was configured to support 6-pole motors.	The Component didn't work for this configuration because of an error in the implementation of the Component's state machine.
	Fixed a bug due to the customizer validating unused (hidden) Component settings.	Now, only visible row values in the fan table are checked for correctness.
	Fixed a bug due to an incorrect value for the initial duty cycle which was generated by the customizer.	
2.10	Updated Component characterization data.	
	Added PSoC 5LP support	
2.0	Added Component characterization data.	
	Changed Damping Factor behavior.	In previous versions Damping Factor specified a delay between speed measurements of each fan and the value of delay wasn't specified in any units. In this version, Damping Factor specifies a delay between start-to-start of speed measurements of all fans. In addition, this delay is now specified in seconds.
	Added new parameter Initial RPM to every fan in the configuration.	This parameter specifies approximate speed of rotation of the specific fan at Component start. In previous versions all fans were initialized with maximum speed.
	Added new feature of selecting the type of motors that are supported by the Component.	This parameter specifies the number of high – low pulses per fan revolution. 2 high-low pulses for4 pole motors, 3 high-low pulses - 6 pole motors.
	Component symbol was extended with a functionality of adding external clock source to drive internal PWMs. This option is accessible in customizer's GUI.	New Component feature. Allows connection to an external clock. Based on the value of the source clock and the resolution of PWMs in the configuration it is possible to regulate the PWM Output Frequency.
	Added new feature, that is selectable in the customizer's GUI and used to display Components inputs and output as a bus.	Now the sets of tach1-tachN, fan1-fanN and bank1-bankN connections can be shown as buses. This allows space to be saved on the schematic as this option reduces the size of Component symbol.

Version	Description of Changes	Reason for Changes / Impact
	Added Keil function reentrancy support to the APIs.	Add the capability for customers to specify any individual generated functions as reentrant.
	Removed obsolete function name - FanController_OverrideClosedLoop(), which was a simple #define of FanController_OverrideHardwareControl().	As this was marked obsolete in previous versions, it is removed in this version.
1.20	1. Updated for compatibility with PSoC Creator v2.0 2. Re-classified as “Concept” Component 3. SetDesiredSpeed() API modified for improved accuracy	
1.10	1. SetDesiredSpeed() API corrected 2. Glitch filter added to tachometer inputs 3. Fan control terminology changed from Open/Closed loop to Firmware/Hardware control method 4. Symbol colors and size updated 5. Resource utilization updated (reduced) 6. References to power management APIs removed (not supported)	
1.0	First release	

© Cypress Semiconductor Corporation, 2014-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.