Fan Controller Datasheet FanController V 1.00

Resources	PSoC® Blocks			API Memory (Bytes)		Pins
	Digital	Analog CT	Analog SC	Flash	RAM	
Supported Devices: CY8C23x33, CY8C24x23, CY8C24x33, CY8C24x94, CY8C27x43, CY8C28x23, CY8C28x33, CY8C28x43, CY8C28x45, CY8C28x52, CY8C29x66						
PWM8 (Open Loop)	2+b	1	–	1850	25+(4xb)	b+f
PWM10(Open loop)	2+(2xb)	1	–	1900	25+(6xb)	b+f
PWM8 (Closed loop)	2+f	1	–	2600	25+(4xf)	2xf
PWM10(Closed loop)	2+(2xf)	1	–	2700	30+(6xf)	2xf
b = Number of fan banks and f = Number of fans used.The total number of fans in Closed-Loop and Fan banks in Open-Loop Control Methods is limited by the total of number digital blocks available in a particular family of devices.The total number of fans in Open-Loop Control Methods is limited by the total of pins that can be connected to an Analog Continuous Block.						

Features and Overview

- Individual or banked pulse width modulator (PWM) outputs with tachometer input
- Supports 24- or 48-kHz PWM frequencies
- Supports 8- or 10-bit PWM resolutions
- Supports fan speeds ranging from 450 to 25,000 rotations per minute (RPM)
- Supports fan stall/rotor lock detection on all fans
- Supports automatic and manual speed regulation
- Selectable tolerance and damping factor parameter

The Fan Controller User Module adjusts fan speeds by changing the duty cycle of the PWM signal applied to the speed control input of the 4-wire brushless DC fans.

Functional Description

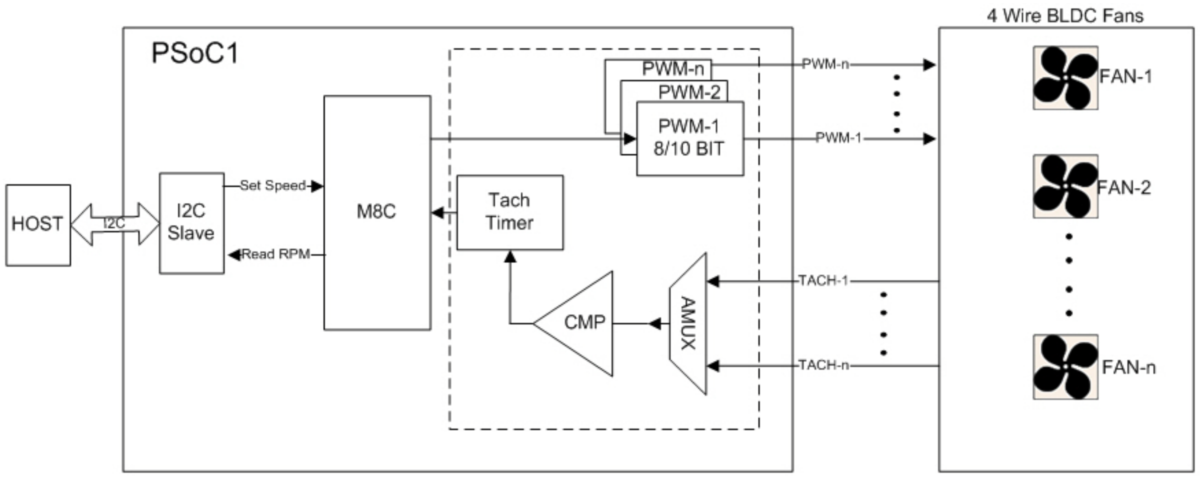
The Fan Controller User Module combines the 8- and 10-bit PWM blocks (implemented on a 16-bit PWM), required to generate PWM, to control the speed of the fan. It also connects the PWMs with an analog multiplexer, hysteresis comparator, and a Timer 16 block to read the TACH pulses from multiple fans for calculating the fan speed. The fans can operate in both open-loop and closed-loop modes. In the open-loop mode, the Fan Controller sets the duty cycle of the PWM, as directed by the host, and reads the TACH input and sends the actual speed of the fan when asked. The host decides how the fan should be controlled based on other system-level inputs. In the closed-loop mode, the host tells the target RPM with the allowed tolerance and the fan controller ensures that the fans run at the given RPM.

The user module adjusts fan speeds by changing the duty cycle of the PWM signal applied to the speed control input of the 4-wire brushless DC fans.

This user module supports the concept of fan “banks” where multiple fans share the same PWM drive signal but all of the individual tachometer feedback signals are connected on individual terminals to enable speed measurement of all fans. Banking is not supported when closed-loop control is enabled because that logic can only support a 1:1 mapping of the PWM drive to tachometer feedback.

The open-loop fan control mode represents the least complex implementation (see Figure 1). In this mode, each fan has a duty cycle provided through APIs that control the PWMs directly. The user module measures fan speed through tachometer input timing and reports those speeds in the RPM again through APIs. Firmware in main.c is then responsible for acting as a pass-through from an external host (over an I^2C interface, for example).

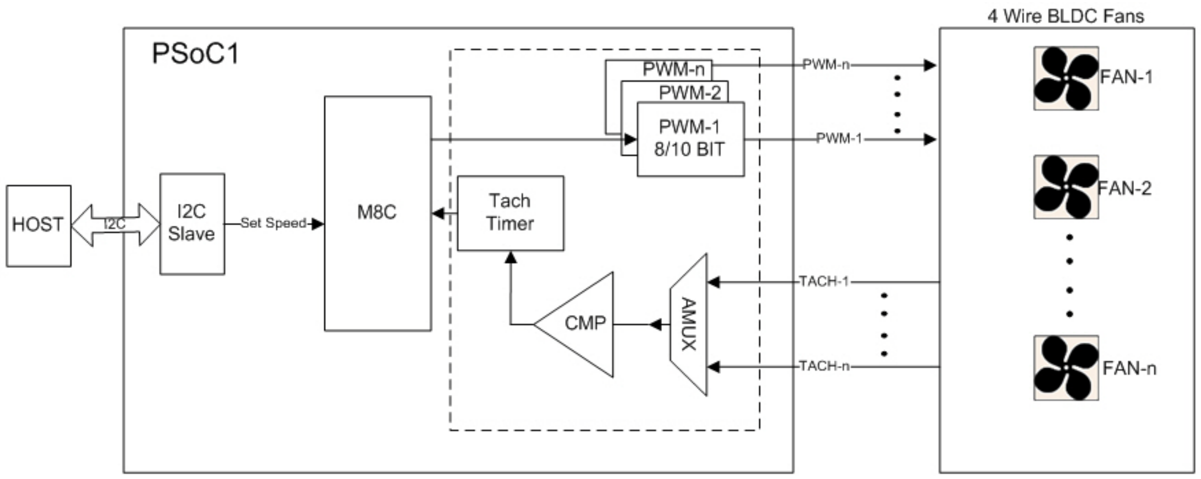
Figure 1. Fan Controller Hardware Block Diagram: Open-Loop Mode



The Closed-Loop Fan Control Method provides a self-regulation capability in terms of fan speed, controlled completely by PSoC® 1 (see Figure 2). In this mode, the user module is given a desired speed for each fan through APIs. The user module is then responsible for adjusting duty cycles and monitoring tachometer inputs independently for each fan until the speed is regulated. This is a ‘set and forget’ mode of operation that frees the host (external processor) from the burden of managing the fans. Actual speeds for each fan are also available through APIs. Failure to achieve regulation optionally generates an interrupt for custom exception handling.

In the closed-loop fan control mode, you can override the control loop and revert to the open-loop fan control mode, if needed. You can switch back and forth between both the modes.

Figure 2. Fan Controller Hardware Block Diagram: Closed-Loop Mode



The user module has three fundamental blocks:

1. Speed control through duty-cycle modulated PWMs
2. Speed measurement through a custom-capture timer connected to the tachometer feedback signals
3. Alert generation in response to fault conditions

Speed Control

In the open-loop and closed-loop fan control modes, standard PWMs are used to control fan speeds. You can select a 8- or 10-bit PWM in the wizard based on your need. The 10-bit PWM is based on the 16-bit PWM. If the selected PWM resolution is 10 bits, then both the VC1 and VC2 clocks are available. If the selected PWM resolution is 8 bits, then the VC1 clock is the PWM clock and only the VC2 clock is available. The set of PWMs clocks formed based on System Clocks are shown in Table 1.

Table 1. PWM Clock sources vs. PWM Resolution and PWM Output

PWM Clock Source	PWM Clock (MHz)	PWM Resolution (bit)	PWM Period	PWM Output (kHz)
VC1 = SysClk/2	12	8	250	48
VC1 = SysClk/4	6	8	250	24
SysClk*2	48	10	1000	48
SysClk	24	10	1000	24

Speed Measurement

In both the Fan Control modes, a custom-capture timer is implemented to measure the period of the tachometer input signals. This can, in turn, be converted to a measurement of fan rotational speed in RPM. Depending on the number of physical poles used in the motor construction, the number of high-low pulses on the tachometer that are measured to correspond to one physical rotation of the fan motor varies. The most common 4-wire brushless DC motor generates two high-low pulses for each rotation. After you enter the number of pulses for every rotation from the Fan datasheet, the wizard calculates the RPM based on that entry.

The capture timer has a 16-bit resolution and is capable to detect fan speeds ranging from 25,000 RPM (1200 counts) down to 450 RPM (counter overflow).

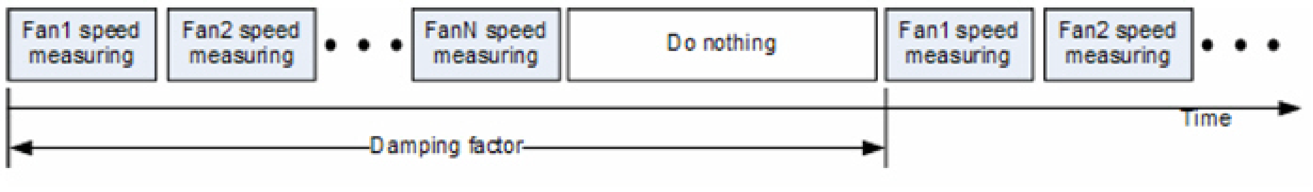
The capture timer is clocked by the VC3 Clock. The VC3 Clock divider is dependent on the tach pulses count generated by the brushless DC motor during each rotation. Dependence of VC3 Clock on the ‘Tach pulses per rotation’ parameter is shown in the Table 2. The IMO Clock = 24 MHz; VC3 Source= SysClk/1.

Table 2. Dependence of Timer Clock (VC3 Divider) on the TACH Pulses per Rotation Parameter of FAN

VC3 Divider	TimerClock, MHz	TACH Pulses per rotation	RPM = 460	RPM = 25,000	Timer overflow,msec
48	0.5	1	65200	1200	131.0
24	1	2	65200	1200	65.5
16	1.5	3	65200	1200	43.7
12	2	4	65200	1200	32.7

In the closed-loop fan control mode, the control loop has a programmable damping factor that controls the “stiffness” or aggressiveness to make duty cycle changes in response to measured speed changes. In applications where the fans themselves have rapid dynamic response, a higher damping factor prevents unwanted oscillations in the control loop around the desired target speed. This is implemented as a simple time delay between successive tachometer queue measurements (see Figure 3). This time delay (between 0 and 2000 ms per fan tachometer measurement) provides natural damping by slowing down the rate at which the new tachometer queues are measured. For production implementation, the damping factor is implemented such that there is a simpler damping impact for a setting of the damping factor, regardless of the number of fans in the system. Because of this, the damping factor is determined based on the type of fan and not the number of fans. The damping factor time is measured by the capture timer.

Figure 3. Damping Factor Definition



Alert Generation

Fan stall faults (a condition when the fan does not rotate) are detected by the speed measurement block when tachometer measurements are timed out. The user sets the fan fault threshold in RPM and the duration for which that RPM has been measured to trigger a fan fault alert.

In the closed-loop fan control mode, speed regulation faults can occur if the CPU does not get the fans to reach the desired speed. This can occur if the PWM drive is already set at 100% but the actual speed is below the desired speed. This can also occur if the PWM drive is already set to 0% but the actual speed is above the desired speed. In a real-world application, this means that there is some mechanical problem in the fan and it cannot rotate at rated speeds or it indicates an error in the thermal management algorithm (implemented either inside or outside PSoC 1) and a desired speed outside the normal operating range is requested. Before generating a speed regulation failure, the condition must occur for 16 successive duty cycle updates. This prevents false alert generation as the fan speed is adjusted near the limits.

DC and AC Electrical Characteristics

Table 3. Fan Controller DC and AC Electrical Characteristics

Parameter	Max	Conditions and Notes
PWM drive frequency	50 kHz	PWM 8/10-bit resolution
PWM drive resolution	10-bit	PWM Frequency – 48 kHz
PWM drive accuracy(this is mainly based on IMO accuracy)	± 4% or ± 5%	4% for devices with IMO variation of 2.5% and 5% for the devices with IMO variation of 4%.

Parameter	Max	Conditions and Notes
Fan RPM supported	25000	With 4 pulses per rotation fans
RPM measurement error	4%, 5%	For max RPM – 25000 / min RPM – 1000
Tach pin input current/voltage	Refer to the DC GPIO Specifications section of the selected device's datasheet	
PWM pin output current	Refer to the DC GPIO Specifications section of the selected device's datasheet	

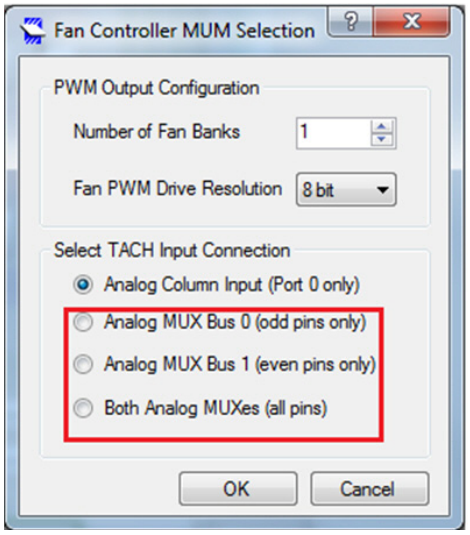
Placement

FanController is a Multi User Module (MUM). The blocks for the user module are automatically placed when the user module is instantiated and user selections are entered; alternate placements are available in certain configurations only. Only one instance of the user module can be placed in the project.

The following sections guide you through the placement and configuration of this user module.

Multi User Module Wizard

1. Select and place the FanController User Module from the Thermal management category in the user module catalog. The MUM wizard (see the following screenshot), which helps you to choose the configuration, pops up. There are many configurations depending on the selected device.


Fan Controller MUM Selection

PWM Output Configuration

Number of Fan Banks: 1

Fan PWM Drive Resolution: 8 bit

Select TACH Input Connection

[✓ Analog Column Input (Port 0 only)]

Analog MUX Bus 0 (odd pins only)

Analog MUX Bus 1 (even pins only)

Both Analog MUXes (all pins)

- **Number of Fan Banks**: The number is equal to the count of used PWMs. The parameter value is limited by the count of PWMs that can be placed simultaneously. The parameter operation depends on the Fan Control Method selected. In the closed-loop method the count of fans is identical to the count of Fan Banks. In the open-loop method, the count of used fans is defined in the wizard.
- **Fan PWM Drive Resolution**: This parameter allows selecting between 8- or 10-bit PWMs.
- **Tach Input Connections**: This item is present only for devices that have an AMUX bus, that is the CY8C28xxx and CY8C24x94 family of devices. This selects how the Fan Tachometer connection is connected to the Fan Controller.

Select the analog column input if you want to use the analog column input for connecting the fan tachometer to the Fan Controller User Module. This helps to free up AMUX bus for other purposes. Select between AMUX BUS0, AMUXBUS1, or choose both depending on your need.

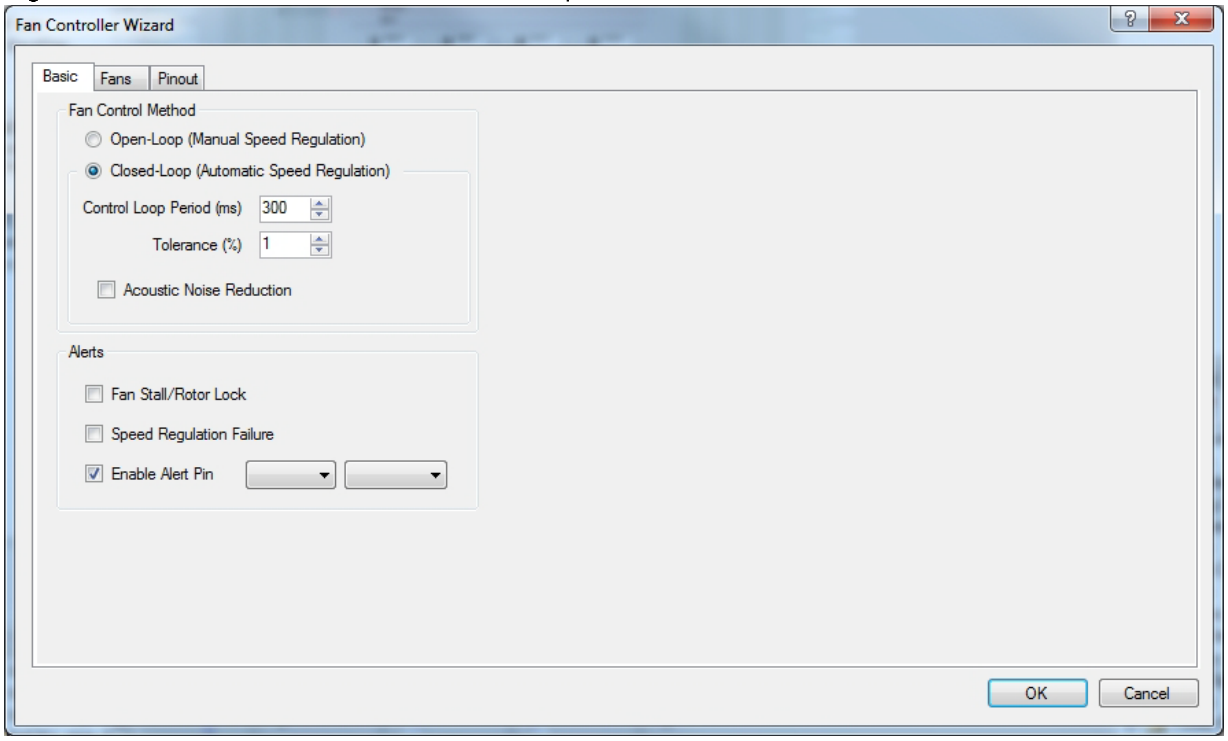
Devices without AMUX bus use Analog Column Input (Port 0 only) to connect TACH inputs by default. The otther three options are grayed out.

- Click **OK** to place the Fan Controller User Module.

FanController Wizard

Right-click the Fan Controller User Module in **Workspace Explorer** or right-click on the FanController User Module in the **Interconnect View** to access the Fan Controller Wizard. The Wizard GUI adapts to all the settings of the Fan Controller User Module entered earlier.

Figure 4. FanController Wizard – Closed Loop Mode



Fan Control Method
- Open-Loop (Manual Speed Regulation)
- [✓ Closed-Loop (Automatic Speed Regulation)]

Control Loop Period (ms) 300

Tolerance (%) 1

☐ Acoustic Noise Reduction

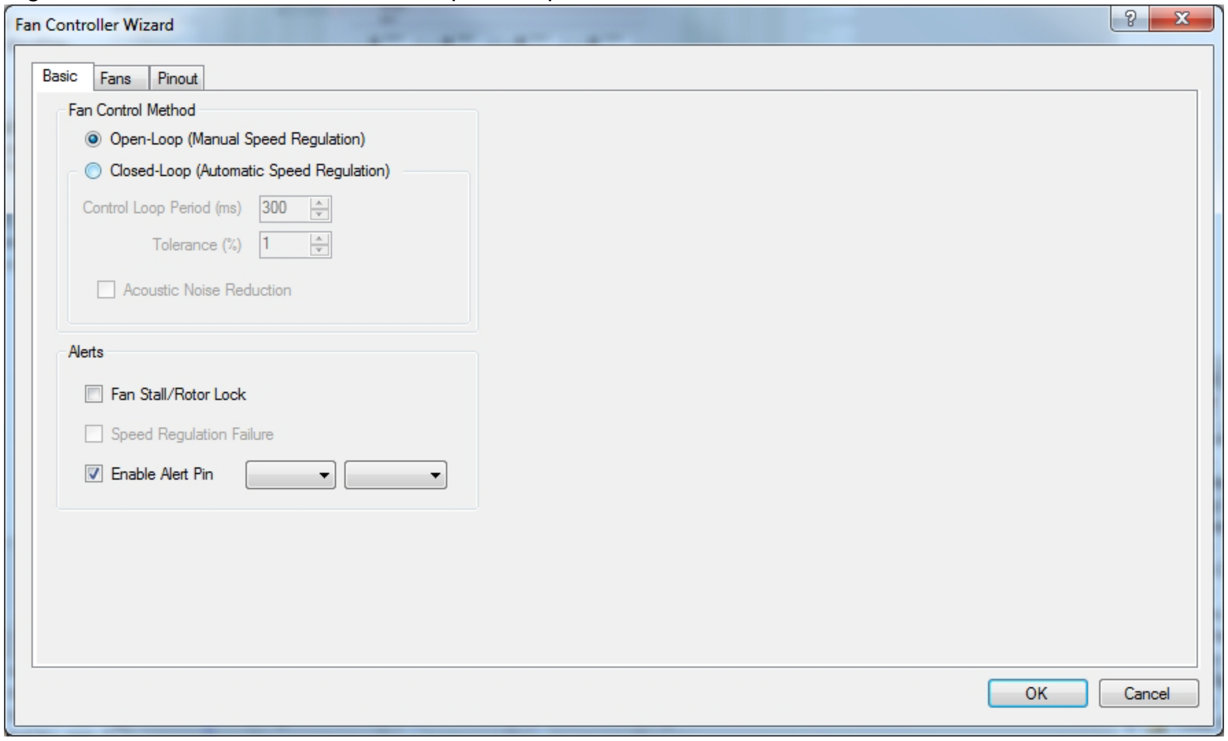
Alerts

☐ Fan Stall/Rotor Lock

☐ Speed Regulation Failure

☑ Enable Alert Pin

Figure 5. FanController Wizard – Open Loop Mode



Fan Control Method
- [✓ Open-Loop (Manual Speed Regulation)]
- Closed-Loop (Automatic Speed Regulation)

Control Loop Period (ms) 300

Tolerance (%) 1

☐ Acoustic Noise Reduction

Alerts

☐ Fan Stall/Rotor Lock

☐ Speed Regulation Failure

☑ Enable Alert Pin

The Wizard has Basic, Fans, and Pins tabs; each of the tabs along with its parameters are explained in the following sections.

Parameters and Resources

Basic Tab

Fan Control Method

Use this to choose between closed- and open-loop fan speed control methods.

- In the closed-loop method, you can set the target RPM and the Fan Controller makes sure that the fans run at the given RPM with the allowed tolerance.
- In the open-loop method, the Fan Controller sets the duty cycle of the PWM as directed in the firmware, and reads the actual speed of the fan when asked. Based on other system level inputs, the host/firmware decides how the fan should be controlled.

In the open-loop method, control options which are not applicable are grayed out. See Figure 5 for open-loop controls

Enable Alert Pin (For Open and Closed loop)

This parameter enables/disables the possibility to use a pin to display Alerts. The pin can be selected from the drop-down menu. If the parameter is unchecked, then the FanController_EnableAlert and FanController_DisableAlert APIs are not operable.

Fan Stall / Rotor Lock (For Open and Closed loop)

This parameter enables/disables the Fan Stall alert from the user module. The alert arises when a fan speed is less than 460 RPM. After assertion, the alert remains asserted until it is not raided or cleared by the firmware. The parameter value can be overwritten by the FanController_SetAlertMode API.

The following parameters are only for the closed loop.

Acoustic Noise Reduction (For Closed loop only)

This parameter enables/disables acoustic noise reduction during fan speed control. If the parameter is enabled, then DutyCycle change steps are smaller, and you need more time to get the desired speed.

Tolerance (For Closed loop only)

This parameter sets the tolerance of speed regulation. Tolerance can be chosen between 1% and 10% with 1% step.

Control Loop Period (For Closed loop only)

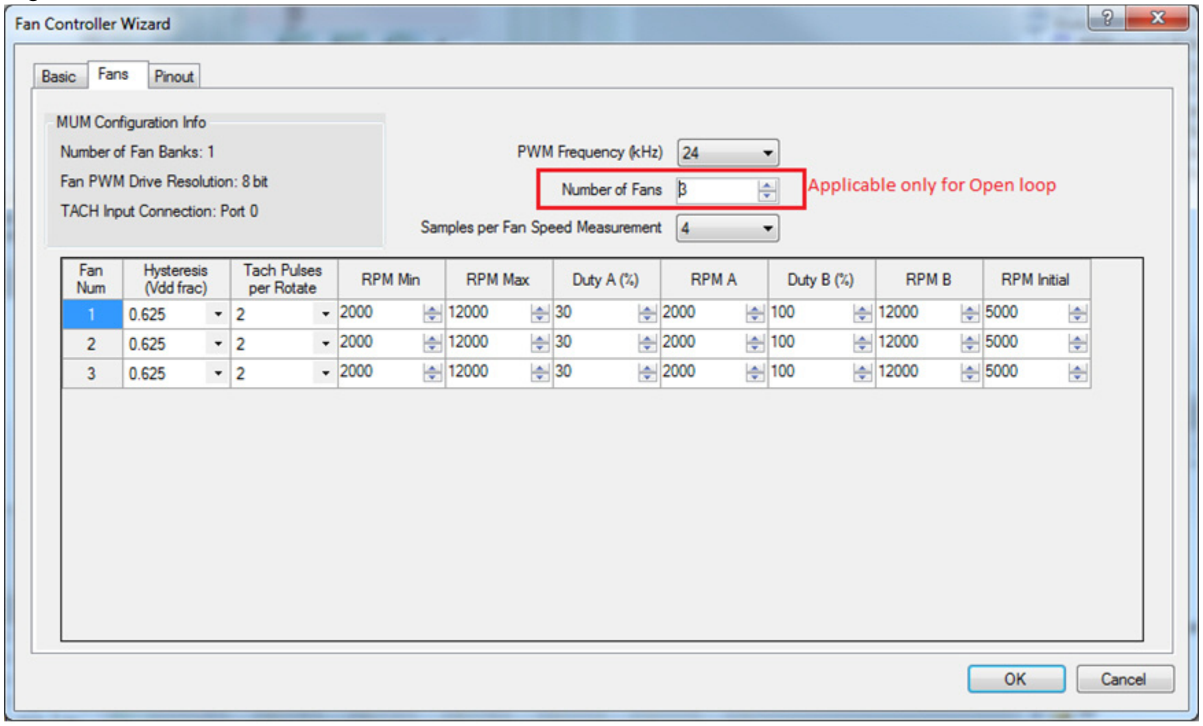
This parameter controls the dynamic response time of the Closed-Control Loop. This parameter controls how frequently the PWM Duty Cycle for each fan is adjusted. If only a few fans are used, a higher parameter value ensures that the Control Loop can regulate the speed to the desired value without oscillations around it. If many fans are controlled, a lower parameter value ensures adequate response time to changes in the fan speed. This parameter enables fine tuning of the Closed-Loop Control to match the electromechanical characteristics of the selected fans. Control Loop Period can be chosen between 0 to 2 seconds with 100 ms step.

Speed Regulation Failure

This parameter enables/disables the Speed Regulation Failure alert. The alert occurs in two cases: (1) if the desired fan speed exceeds the current actual fan speed but the fan's DutyCycle is already 100%; (2) if the desired fan speed is below the current actual fan speed but the fan's DutyCycle0 is already 0%. After it is asserted, the alert remains asserted until it is not raided/cleared by firmware. The parameter value can be overwritten by the FanController_SetAlertMode API.

Fans Tab

Figure 6. FanController Wizard Fans Tab



MUM Configuration Info

Number of Fan Banks: 1

Fan PWM Drive Resolution: 8 bit

TACH Input Connection: Port 0

PWM Frequency (kHz) 24

Number of Fans 3 Applicable only for Open loop

Samples per Fan Speed Measurement 4

Fan Num	Hysteresis (Vdd frac)	Tach Pulses per Rotate	RPM Min	RPM Max	Duty A (%)	RPM A	Duty B (%)	RPM B	RPM Initial
1	0.625	2	2000	12000	30	2000	100	12000	5000
2	0.625	2	2000	12000	30	2000	100	12000	5000
3	0.625	2	2000	12000	30	2000	100	12000	5000

Parameters common to both open and closed parameters

PWM Frequency

This parameter selects between 24 kHz or 48 kHz PWM frequency. Default value is 24 kHz

Samples per Fan Speed Measurement

This parameter sets count of TACH pulses needed for speed calculation. More samples increase accuracy of the speed measurement but take more time

Hysteresis

This parameter sets the hysteresis of the comparator used for filtering the input TACH signal. The options for hysteresis are 0.062, 0.125, 0.188, 0.250, 0.312, 0.375, 0.437, 0.500, 0.562, 0.625, 0.688, 0.750, 0.812, 0.875 and 0.937.The Hysteresis voltage is set by resistive divider tap at increments of Vdd *(1/16, 2/16, ..., n/16, ..., 15/16). The Hysteresis amount is a function of the power supply setting (that is, 3.3 V, 5 V, 2.7 V, and so on).

Tach Pulses per Rotate

This parameter selects the number of tach pulses generated by brushless DC motor per rotation. The options for this parameter are 1, 2, and 4.

RPM Min

This parameter sets the RPM of the fan when the PWM Duty Cycle is 0%. The value can be selected from 460 to 25000 with 100 steps.

RPM Max

This parameter sets the RPM of the fan when the PWM DutyCycle is 100%. The value can be selected from 461 to 25000 in steps of 100.

Duty A (%)

This parameter sets the first Duty Cycle data point from the Fan datasheet. The value can be selected from 0 to 100 in steps of 1%.

RPM A (%)

This parameter sets the first RPM data point from the Fan datasheet. The value can be selected from 460 to 25000 in steps of 100.

Duty B (%)

This parameter sets the second Duty Cycle data point from the Fan datasheet. The value can be selected from 0 to 100 in steps of 1%.

RPM B (%)

This parameter sets the second RPM data point from the Fan datasheet. The value can be selected from 460 to 25000 in steps of 100

RPM-Initial

This parameter sets the initial RPM of the fan when the UM is enabled. The value can selected from 460 to 25000 in steps of 100.

Parameters for Open loop only

Number of Fans

This parameter specifies the number of fans used in the system. This parameter is available only in the open-loop control method. In the closed-loop control method, the number of fans is equal to the number of PWM banks.

Pins Tab

User module parameters in the Pins Tab control the pin assignment of the FanController User Module. The tab is different for Open and Closed Loop Methods.

For the selected part number, the Pin assignment wizard displays the available graphics of the part with available pins and allowed pins for a particular function. There is a drag-and-drop mechanism to easily configure the Tach and PWM pins. Follow these steps to configure this wizard:

1. In an open-loop control, a particular fan can be assigned to any fan bank by checking it against that bank. All fans in a particular bank have the same PWM drive pin (refer to Figure 8).
2. Left-click a fan's PWM/Tach connection and note that the available pins get highlighted with a red box, Then drag it to any available pin. The port pin is green after selection and is no longer available. To change the pin assignments, drag the new Tach/PWM to the assigned pin. The pins are displayed in the Table Assignment view.
3. Repeat this procedure for the remaining independent PWM/Tach of other fans.

4. Click **OK** to complete.

Figure 7. Fan Controller Pinout Tab Wizard GUI – Closed-Loop Method

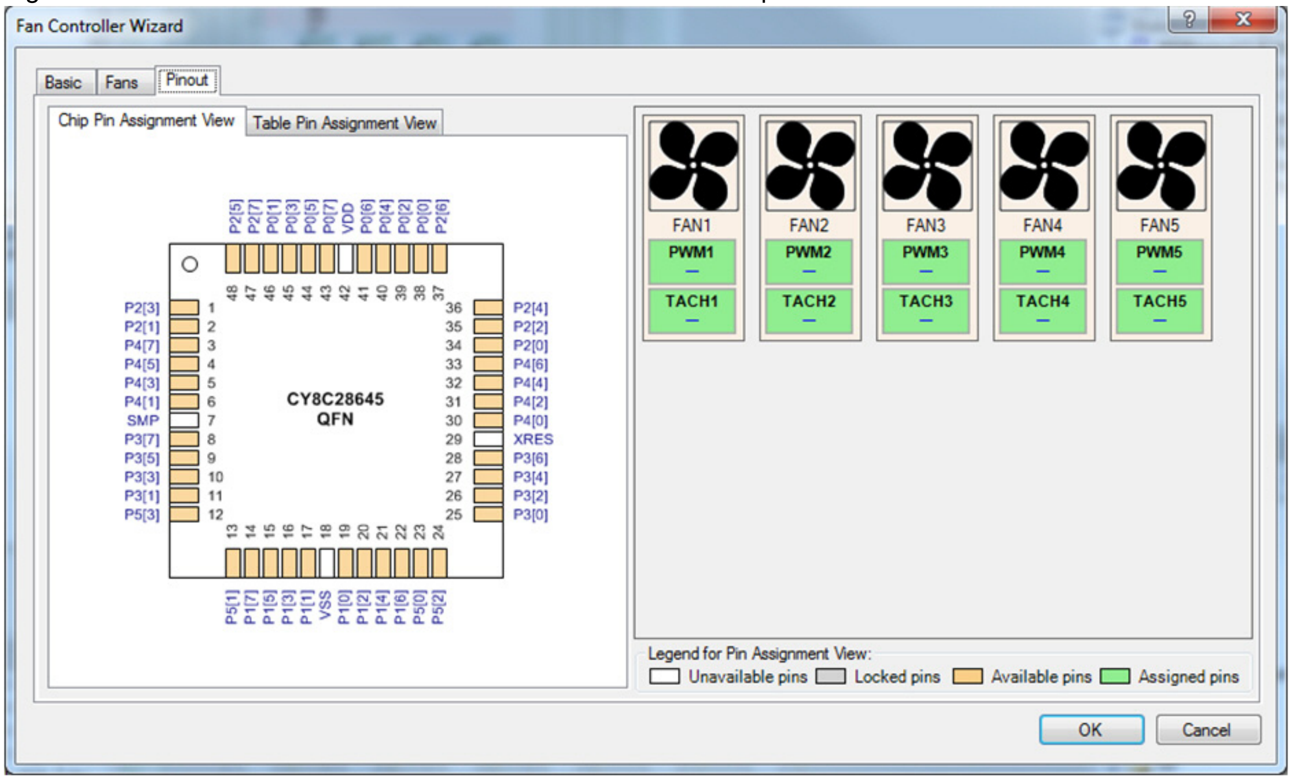
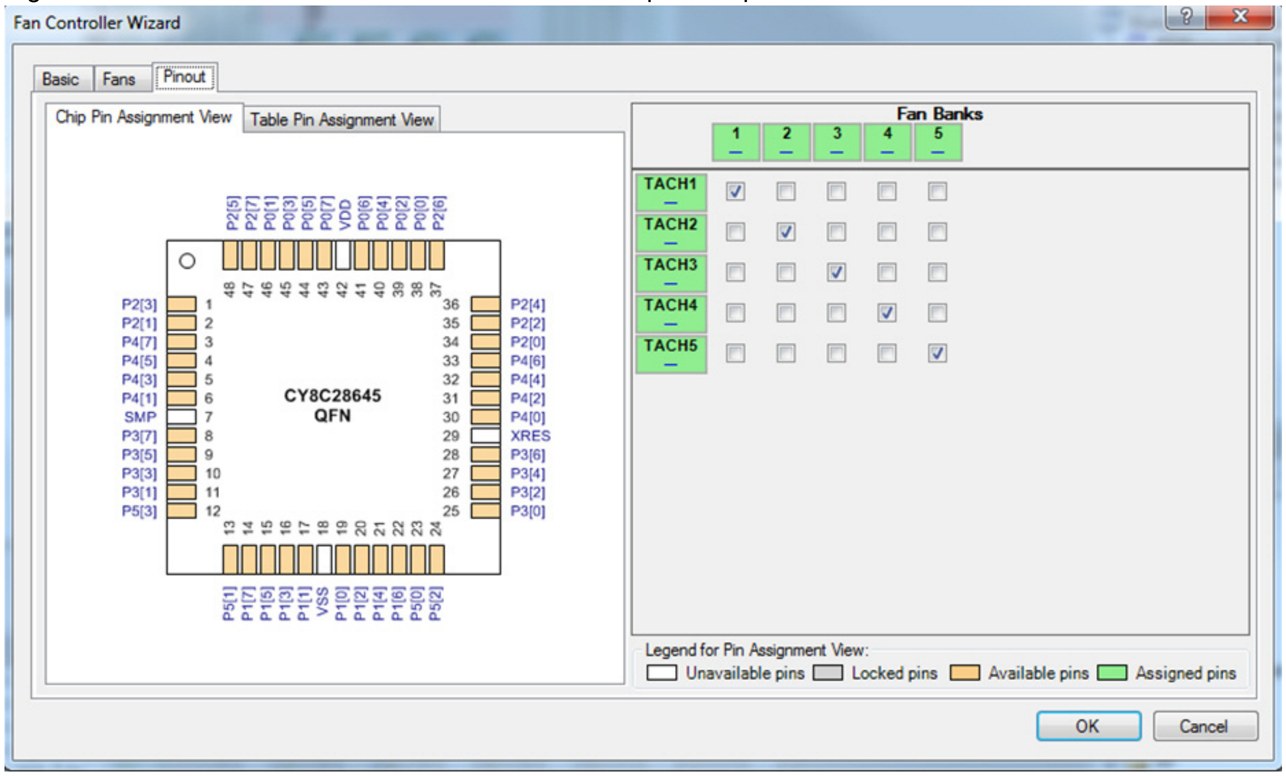

Fan Controller Wizard
Basic
Fans
Pinout
Chip Pin Assignment View
Table Pin Assignment View
CY8C28645
QFN
P2[5] P2[7] P0[1] P0[3] P0[5] P0[7] VDD P0[6] P0[4] P0[2] P0[0] P2[6]
P2[3] P2[1] P4[7] P4[5] P4[3] P4[1] SMP P3[7] P3[5] P3[3] P3[1] P5[3]
P2[4] P2[2] P2[0] P4[6] P4[4] P4[2] P4[0] XRES P3[6] P3[4] P3[2] P3[0]
P5[1] P1[7] P1[5] P1[3] P1[1] VSS P1[0] P1[2] P1[4] P1[6] P5[0] P5[2]
FAN1 PWM1 TACH1
FAN2 PWM2 TACH2
FAN3 PWM3 TACH3
FAN4 PWM4 TACH4
FAN5 PWM5 TACH5
Legend for Pin Assignment View:
Unavailable pins
Locked pins
Available pins
Assigned pins
OK
Cancel

Figure 8. Fan Controller Pinout Tab Wizard GUI – Open Loop Method


Fan Controller Wizard
Basic
Fans
Pinout
Chip Pin Assignment View
Table Pin Assignment View
CY8C28645
QFN
P2[5] P2[7] P0[1] P0[3] P0[5] P0[7] VDD P0[6] P0[4] P0[2] P0[0] P2[6]
P2[3] P2[1] P4[7] P4[5] P4[3] P4[1] SMP P3[7] P3[5] P3[3] P3[1] P5[3]
P2[4] P2[2] P2[0] P4[6] P4[4] P4[2] P4[0] XRES P3[6] P3[4] P3[2] P3[0]
P5[1] P1[7] P1[5] P1[3] P1[1] VSS P1[0] P1[2] P1[4] P1[6] P5[0] P5[2]
Legend for Pin Assignment View:
Unavailable pins
Locked pins
Available pins
Assigned pins
OK
Cancel

Fan Banks	1	2	3	4	5
TACH1	✓				
TACH2		✓			
TACH3			✓		
TACH4				✓	
TACH5					✓

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This sections specifies the interface to each function together with related constants provided by the “include” files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API functions may leave A and X unchanged, there is no guarantee they will do so in the future.

Functions available in both open- and closed-loop methods

FanController_Start

Description:

Initializes the FanController User Module. Starts the Hysteresis Comparator, TACH Timer, and PWMs. Global Interrupts must be enabled for UM operation.

C Prototype:

```
void FanController_Start(void)
```

Assembly:

```
lcall FanController_Start
```

Parameters:

None

Return Value:

None

FanController_Stop

Description:

Stops the FanController User Module. Shuts down the Hysteresis Comparator and TACH Timer. Restores the TACH input pins to the default High-Z Analog drive mode. Sets the DutyCycles of all PWMs to 100%.

C Prototype:

```
void FanController_Stop(void)
```

Assembly:

```
lcall FanController_Stop
```

Parameters:

None

Return Value:

None

FanController_EnableAlert

Description:

Enables the pin alert assertion when pending alerts are enabled. The alert sources that need to be enabled are configured using the FanController_SetAlertMode() and the FanController_SetAlerts() APIs.

C Prototype:

```
void FanController_EnableAlert(void)
```

Assembly:

```
lcall FanController_EnableAlert
```

Parameters:

None

Return Value:

None

Notes:

The API is operable only if the Alert Pin is selected in the Wizard. By default, the Alert Pin in the Wizard is selected.

FanController_DisableAlert

Description:

Disables the assertion of the pin alert.

C Prototype:

```
void  FanController_DisableAlert(void);
```

Assembly:

```
lcall  FanController_DisableAlert
```

Parameters:

None

Return Value:

None

Note:

Alert pin is de-asserted. The API is operable only if the Alert Pin is selected in Wizard. By default, the Alert Pin in Wizard is selected.

FanController_SetFanState

Description:

Defines whether the Fan Tach output is present in the Tach frequency measurement queue. In the Closed-Loop Method, it only allows to enable/disable the appropriate Fan PWM.

C Prototype:

```
void FanController_SetFanState(BYTE bFanNumber, BYTE bEnable)
```

Assembly:

```
mov X, bEnable
mov A, bFanNumber
lcall FanController_SetFanState
```

Parameters:

BYTE bFanNumber: number of Fan for which a state should be changed.

BYTE bEnable: defines the fan state.

Symbolic Name	Value	Description
FanController_FAN_ENABLED	1	Includes Fan Tach output to Tach frequency measurement queue. Closed-loop method only: Enables appropriate Fan PWM.
FanController_FAN_DISABLED	0	Excludes Fan Tach output to Tach frequency measurement queue. Closed-loop method only: Disables appropriate Fan PWM.

Return Value:

None

Note:

By default all Fan Tach outputs are present in the Tach frequency measurement queue and all PWMs are enabled. In the open-loop method all fan PWMs are always enabled.

FanController_GetFanState

Description:

Returns status of selected Fan, irrespective of whether or not its Tach output is present in the Tach frequency measurement queue.

C Prototype:

```
BYTE FanController_GetFanState(BYTE bFanNumber)
```

Assembly:

```
mov A, bFanNumber
lcall FanController_GetFanState
```

Parameters:

BYTE bFanNumber: number of selected Fan.

Symbolic Name	Value	Description
FanController_FAN_ENABLED	1	Includes Fan Tach output to Tach frequency measurement queue. Closed-loop method only: Enables appropriate Fan PWM.
FanController_FAN_DISABLED	0	Excludes Fan Tach output to Tach frequency measurement queue. Closed-loop method only: Disables appropriate Fan PWM.

Note:

By default all Fan Tach outputs are present in the Tach frequency measurement queue and all PWMs are enabled. In the open-loop method, all Fan PWMs are always enabled.

FanController_SetAlertMode

Description:

Configures alert sources from the user module. Two alert sources are available: Fan Stall or Rotor Lock, and Speed Regulation Failure in the Closed-Loop method.

C Prototype:

```
void FanController_SetAlertMode(BYTE bAlertMode)
```

Assembly:

```
mov A, bAlertMode
lcall  FanController_SetAlertMode
```

Parameters:

BYTE bAlertMode - determines Alert source.

Symbolic Name	Value	Description
FanController_STALL_ALERTS	0X01	Enable Fan stall/rotor lock alerts
FanController_SPEED_ALERTS	0X02	Enable speed regulation failure alerts(Closed Loop mode only))

Return Value:

None.

Note:

The API overrides the Alerts settings that are set in the Wizard.

FanController_GetAlertMode

Description:

Returns enabled alert sources from the user module.

C Prototype:

```
BYTE FanController_GetAlertMode(void)
```

Assembly:

```
lcall  FanController_GetAlertMode
```

Parameters:

None.

Return Value:

Enabled alert sources.

Symbolic Name	Value	Description
FanController_STALL_ALERTS	0X01	Fan stall / rotor lock alerts are enabled
FanController_SPEED_ALERTS	0X02	Speed regulation failure alerts are enabled(Closed-Loop method only)

FanController_SetAlerts

Description:

Enables or disables alerts from selected fan. The operation applies to both fan stall alerts and speed regulation failure alerts. By default all fans present have their alert masks enabled.

C Prototype:

```
void FanController_SetAlerts(BYTE bFanNumber, BYTE bAlertEnable)
```

Assembly:

```
mov X, bAlertEnable
mov A, bFanNumber
lcall FanController_SetAlerts
```

Parameters:

BYTE bFanNumber: number of selected Fan.

BYTE bAlertEnable: defines whether alerts from selected fan are enabled or disabled

Symbolic Name	Value	Description
FanController_ALERTS_ENABLED	1	Enables alerts from the selected fan
FanController_ALERTS_DISABLED	0	Disables alerts from the selected fan

Return Value

None

FanController_GetAlerts

Description:

Determines if alerts are enabled/disabled for the selected fan. The operation applies to both fan stall alerts and speed regulation failure alerts.

C Prototype:

```
BYTE FanController_GetAlerts(BYTE bFanNumber)
```

Assembly:

```
mov A, bFanNumber
lcall FanController_GetAlerts
```

Parameters:

BYTE bFanNumber: number of selected Fan

Return Value:

Alerts status of the selected fan.

Symbolic Name	Value	Description
FanController_ALERTS_ENABLED	1	Alerts from the selected fan are enabled
FanController_ALERTS_DISABLED	0	Alerts from the selected fan are disabled

FanController_GetAlertSource

Description:

Returns pending alert sources from the user module. This API can be used to poll the alert status. If this API returns a non-zero value, the FanController_GetFanStallStatus() and FanController_GetFanSpeedStatus() APIs can provide further information on which fans have a fault.

C Prototype:

```
BYTE FanController_GetAlertSource(void)
```

Assembly:

```
lcall FanController_GetAlertSource
```

Parameters:

None

Return Value:

Pending alert sources.

Symbolic Name	Value	Description
FanController_STALL_ALERTS	0X01	Fan stall / rotor lock alerts are pending
FanController_SPEED_ALERTS	0X02	Speed regulation failure alerts are pending (Closed-Loop method only)

FanController_GetFanStallStatus

Description:

Returns the stall / rotor lock status of the selected fan and clears it pending stall alert. Fan stall/ rotor lock status is set when the current fan speed is less than 460 RPM.

C Prototype:

```
BYTE FanController_GetFanStallStatus(BYTE bFanNumber)
```

Assembly:

```
mov A, bFanNumber
lcall FanController_GetFanStallStatus
```

Parameters:

BYTE bFanNumber: number of selected Fan.

Return Value:

Returns the stall/rotor lock status of selected fan.

Symbolic Name	Value	Description
FanController_STATUS_OK	0	No stall/rotor lock status of selected fan
FanController_STATUS_STALL	1	Stall/rotor lock status of selected fan

Notes:

The API clears pending stall / rotor lock alert of selected fan only. To clear all pending stall / rotor lock alerts the FanController_ClearPendingStallAlerts API should be called.

FanController_ClearPendingStallAlerts

Description:

Clears the stall/rotor lock alerts of all fans and clears the flags of pending stall alerts.

C Prototype:

```
void FanController_ClearPendingStallAlerts(void)
```

Assembly:

```
lcall FanController_ClearPendingStallAlerts
```

Parameters:

None.

Return Value:

None.

FanController_SetDutyCycle

Description:

Sets the PWM duty cycle of the selected fan or fan bank in tenth of a percent.

C Prototype:

```
void FanController_SetDutyCycle(BYTE bFanOrBankNumber,WORD wDutyCycle)
```

Assembly:

```
mov A, >wDutyCycle
push A
mov A, <wDutyCycle
push A
mov A, bFanOrBankNumber
push A
lcall FanController_GetFanSpeedStatus
add SP, -3
```

Parameters:

BYTE bFanOrBankNumber. Number of selected fan or bank. The value should not exceed the number of fans or banks in the system WORD wDutyCycle. Duty cycle in tenths of a percent. For example, 50% duty cycle = 500. The valid range is 0..1000.

Return Value:

None.

FanController_GetDutyCycle

Description:

Returns the current PWM duty cycle of the selected fan or fan bank in tenths of a percent.

C Prototype:

```
WORD FanController_GetDutyCycle(BYTE bFanOrBankNumber)
```

Assembly:

```
mov A, bFanOrBankNumber
lcall FanController_GetDutyCycle
```

Parameters:

BYTE bFanOrBankNumber. Number of selected fan or bank. The value should not exceed the number of fans or banks in the system.

Return Value:

Returns duty cycle in tenths of a percent. For example, 50% duty cycle = 500. The valid range is 0..1000.

FanController_SetDesiredSpeed

Description:

Sets the desired speed of the specified fan in RPM. In the closed-loop fan control mode, the wRPM parameter is passed to the control loop as the new target fan speed for regulation. In the open-loop method of fan control, the wRPM parameter is converted to a duty cycle based on the fan parameters entered into the Fans tab of the Wizard 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.

C Prototype:

```
void FanController_SetDesiredSpeed(BYTE bFanNumber, WORD wRpm)
```

Assembly:

```
mov A, >wRpm
push A
mov A, <wRpm
push A
mov A, bFanNumber
push A
lcall FanController_SetDesiredSpeed
add SP, -3
```

Parameters:

BYTE bFanNumber. Number of selected fan.

WORD wRpm:

The valid range is 500..25,000 but should not exceed the maximum RPM and should not go below the minimum RPM that the fan is capable of running at. Doing so causes a speed regulation failure.

Return Value:

None.

FanController_GetActualSpeed

Description:

Returns the current actual speed of the specified fan in RPM.

C Prototype:

```
WORD FanController_GetActualSpeed(BYTE bFanNumber))
```

Assembly:

```
mov A, bFanNumber
lcall FanController_GetActualSpeed
```

Parameters:

BYTE bFanNumber. Number of selected fan.

Return Value:

Returns the currently desired speed for the selected fan in RPM.

Some functions available in the closed-loop control method only.

FanController_GetFanSpeedStatus

Description:

Returns the speed regulation status of selected fan and clears its pending speed regulation alert. 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%.

C Prototype:

```
BYTE FanController_GetFanSpeedStatus(BYTE bFanNumber)
```

Assembly:

```
mov A, bFanNumber
```

```
lcall FanController_GetFanSpeedStatus
```

Parameters:

BYTE bFanNumber: Number of selected Fan.

Return Value:

Returns the speed regulation status of selected.

Symbolic Name	Value	Description
FanController_STATUS_OK	0	No speed regulation fail status of selected fan
FanController_STATUS_SPEED_FAIL	1	Speed regulation fail status of selected fan

Note:

The API clears the pending speed regulation alert of only the selected fan. To clear all pending speed regulation alerts, the baFanSpeedStatus array should be cleared manually and the API with any allowed parameter should be called.

FanController_ClearPendingSpeedAlerts

Description:

Clears the speed alerts of all fans and clears flag of pending speed alerts.

C Prototype:

```
void FanController_ClearPendingSpeedAlerts(void)
```

Assembly:

```
lcall FanController_ClearPendingSpeedAlerts
```

Parameters:

None.

Return Value:

None.

FanController_ClearAllPendingAlerts

Description:

Clears both the fan stall/rotor lock alerts and speed regulation alerts of all fans and clears the flags of the pending alerts.

C Prototype:

```
void FanController_ClearAllPendingAlerts(void)
```

Assembly:

```
lcall FanController_ClearAllPendingAlerts
```

Parameters:

None.

Return Value:

None.

FanController_GetDesiredSpeed

Description:

Returns the currently desired RPM of the specified fan.

C Prototype:

```
WORD FanController_GetDesiredSpeed(BYTE bFanNumber)
```

Assembly:

```
mov A, bFanNumber
lcall FanController_GetDesiredSpeed
```

Parameters:

BYTE bFanNumber: Number of selected Fan.

Return Value:

Currently desired speed for the selected fan in RPM.

FanController_OverrideClosedLoopControl

Description:

Allows firmware to take over fan control in the closed-loop fan control mode. Note that this API cannot be called in the open-loop fan control mode.

C Prototype:

```
void FanController_OverrideClosedLoopControl(BYTE bOverride)
```

Assembly:

```
mov A, bOverride
lcall FanController_OverrideClosedLoopControl
```

Parameters:

BYTEbOverride.

Symbolic Name	Value	Description
FanController_MODE_CLOSED_LOOP	0	Closed-loop control method of fans
FanController_MODE_OPEN_LOOP	1	Open-loop control method of fans

Return Value:

None.

Sample Firmware Source Code

```
//----------------------------------------------------------------------------
// C main line
//----------------------------------------------------------------------------
```

```
#include <m8c.h>          // part specific constants and macros
#include "PSoCAPI.h"      // PSoC API definitions for all User Modules

void ProceedStallAlert(BYTE b)
{
}

void main(void)
{
    BYTE bFanNumber;
    M8C_EnableGInt;

 FanController_Start(); // Enable User Module Operations
    // Enable Fan Stall Failure alerts
    FanController_SetAlertMode(FanController_STALL_ALERTS);
    // Enable pin assertion when pending alerts are present
    FanController_EnableAlert();

    // Settings separate speed for each fan
    FanController_SetDesiredSpeed(1, 4000);
    FanController_SetDesiredSpeed(2, 8000);
    FanController_SetDesiredSpeed(3, 5000);
    FanController_SetDesiredSpeed(4, 9900);

    while (1)
    {
        if (FanController_GetAlertSource() & FanController_STALL_ALERTS)
        { // fan fault happened
            // loop to find stopped fans
            for (bFanNumber = 1; bFanNumber <= FanController_TOTAL_FAN_COUNT;
            bFanNumber++)
        {
            // read and clear stall status of the fan
            if (FanController_GetFanStallStatus(bFanNumber) &
            FanController_STATUS_STALL)
            {
                // exclude stopped fan from speed measurement and regulate queues
                FanController_SetFanState(bFanNumber, FanController_FAN_DISABLED);

                // User defined function to process the Fan Stall fault
                ProceedStallAlert(bFanNumber);
            }
        }
        // When stall statuses of all fans are cleared, the flag of pending
        // stall alerts is cleared too
        }
    }
}
```

The similar code in Assembly is:

```
;-----------------------------------------------------------------------------
; Assembly main line
;-----------------------------------------------------------------------------
```

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

area bss(RAM, REL)
bFanNumber:   blk 1

area text(ROM, REL)
_main:
    M8C_EnableGInt
    ;  Enable User Module Operations
    lcall FanController_Start
    ;Enable Fan Stall Failure alerts
    mov   A, FanController_STALL_ALERTS
    lcall FanController_SetAlertMode
    ; Enable pin assertion when pending alerts are present
    lcall FanController_EnableAlert
    ; Settings speed for 1st Fan
        mov   A, >4000
        push  A
        mov   A, <4000
        push  A
        mov   A, 1
        push  A
    lcall FanController_SetDesiredSpeed
loop:
        lcall FanController_GetAlertSource
        and   A, FanController_STALL_ALERTS
    jz loop
        ; fan fault happened
        ; loop to find stopped fans

        mov   [bFanNumber], FanController_TOTAL_FAN_COUNT
CheckNextFan:
        ; read and clear stall status of the fan
        mov   A, [bFanNumber]
        lcall FanController_GetFanStallStatus
        and   A, FanController_STATUS_STALL
        jz    CheckNextFan
        ; exclude stopped fan from speed measurement and regulate queues
        mov   A, [bFanNumber]
        mov   X, FanController_FAN_DISABLED
        lcall FanController_SetFanState
        mov   A, [bFanNumber]
        ; User defined function to process the Fan Stall fault
    lcall ProceedStallAlert
        dec   [bFanNumber]
        jnz   CheckNextFan
    ; when stall statuses of all fans are cleared, the flag of pending
    ; stall alerts is cleared too
        jmp   loop
ProceedStallAlert:
    ; insert code to process the Fan Stall fault
    ret
```

Version History

Version	Originator	Description
1.00	DHA	Initial release
1.00.b	MYKZ	Renamed "Damping Factor (ms)" in the Wizard to "Control Loop Period (ms)".

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.