Angle and Speed Calculation MSS Software Implementation

User Guide

Microsemi

Table of Contents

Introduction

Angle Calculation

In sensorless motor control applications, the back EMF of the motor is used to calculate the rotor angle. In this implementation, the back EMF is calculated using the Luenberger State Observer system, as shown in Figure 1. The Luenberger State Observer system takes voltage and currents represented in stationary orthogonal two-phase reference frame along with feedback signals from the state observer as inputs. The feedback signals are estimated voltage and currents in stationary orthogonal two-phase (α, β axis) reference frames. The estimated voltage signals from the state observer are given to a phase lock loop, which gives the rotor speed and rotor angle as output.

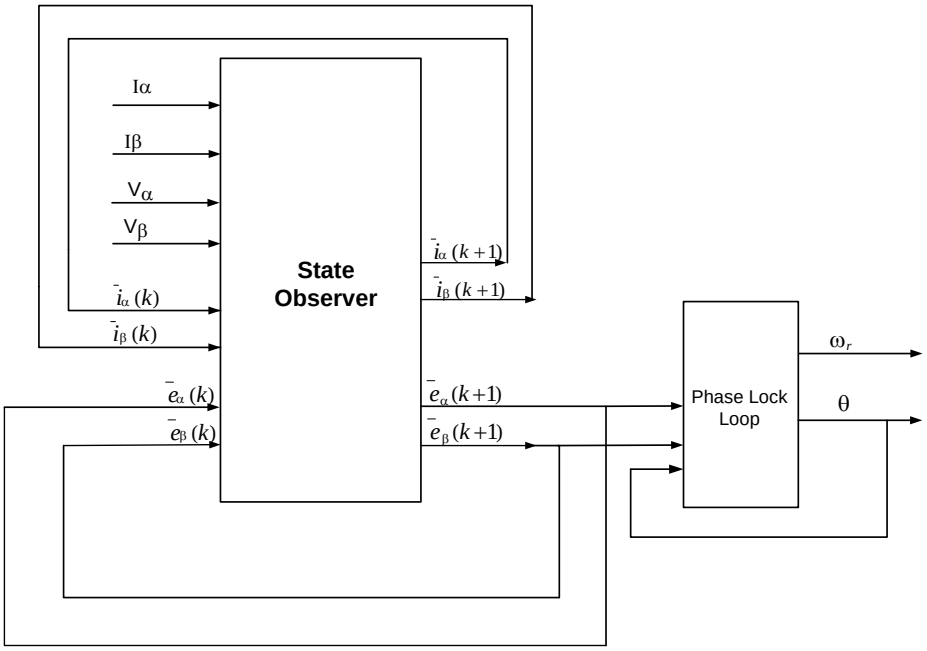

Figure 1 · Angle Calculation Block Diagram using Luenberger State Observer

Speed Calculation

The angle calculated from the above implementation is used to calculate the rotor speed. The angle is given to a high pass filter (differentiator). The output of the filter would give the value of the rotor speed (since dθ/dt = ω). The high pass filter is implemented by choosing the filter coefficients accordingly.

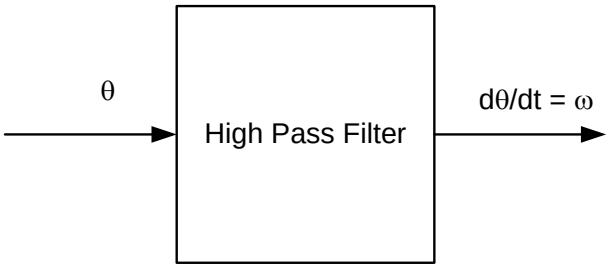

Figure 2 · Speed Calculation

API Type Definitions

This section describes the type definitions required to implement the MSS software libraries of angle and speed calculations.

Angle Calculation Type Definitions

current_type

Table 1 gives the type definition of current_type.

Table 1 · current_type

Name	current_type	
Type	typedef struct { int32_t actual_ialpha; int32_t actual_ibeta; int32_t estimated_ialpha; int32_t estimated_ibeta; int32_t max_limit; int32_t min_limit; } current_type;	
File	Lib.h	
Range	int32_t actual_ialpha	The actual current component in stationary orthogonal reference frame on alpha axis
	int32_t actual_ibeta	The actual current component in stationary orthogonal reference frame on beta axis
	int32_t estimated_ialpha	The estimated current component in stationary orthogonal reference frame on alpha axis
	int32_t estimated_ibeta	The estimated current component in stationary orthogonal reference frame on beta axis
	int32_t max_limit	The limit value of estimated_ialpha and estimated_ibeta, if these values exceed the max_limit
	int32_t min_limit	The limit value of estimated_ialpha and estimated_ibeta, if these values are less than the min_limit

voltage_type

Table 2 gives the type definition of voltage_type.

Table 2 · voltage_type

Name	voltage_type	
Type	typedef struct { int32_t actual_valpha; int32_t actual_vbeta; } voltage_type;	
File	Lib.h	
Range	int32_t actual_valpha	The actual voltage component in stationary orthogonal reference frame on alpha axis
	int32_t actual_vbeta	The actual voltage component in stationary orthogonal reference frame on beta axis

bemf_type

Table 3 gives the type definition of bemf_type.

Table 3 · bemf_type

Name	bemf_type	
Type	typedef struct { int32_t estimated_ealpha; int32_t estimated_ebeta; int32_t max_limit; int32_t min limit; } bemf_type;	
File	Lib.h	
Range	int32_t estimated_ealpha	The estimated back emf voltage component in stationary orthogonal reference frame on alpha axis
	int32_t estimated_ebeta	The estimated back emf voltage component in stationary orthogonal reference frame on beta axis
	int32_t max_limit	The limit value of estimated_ealpha and estimated_ebeta, if these values exceed the max_limit
	Int32_t min_limit	The limit value of estimated_ealpha and estimated_ebeta, if these values are less than the min_limit

stateobserver_type

Table 4 gives the type definition of stateobserver_type.

Table 4 · stateobserver_type

Name	stateobserver_type	
Type	typedef struct { current_type current; voltage_type voltage; bemf_type bemf; int32_t estimatorfactor1; int32_t estimatorfactor2; int32_t estimatorfactor3; int32_t estimatorfactor4; int32_t estimatorfactor5; } stateobserver_type;	
File	Lib.h	
Range	current_type current	The actual and estimated currents on alpha axis and beta axis
	voltage_type voltage	The actual and estimated voltages on alpha axis and beta axis
	bemf_type bemf	The estimated backemf on alpha and beta axis
	int32_t estimatorfactor1	This value is calculated using the equation: R * Ts/Ls where, Ls - Phase inductance R - Phase Resistance Ts - Sampling time
	int32_t estimatorfactor2	This value is calculated by Ts/Ls
	int32_t estimatorfactor3	This value is calculated as K1 * Ts Where , K1 - observer gain Ts - Sampling time
	int32_t estimatorfactor4	This value is calculated as K2 * Ts Where, K2 - observer gain Ts - Sampling time

	int32_t estimatorfactor5	This value is calculated as p * Ts Where , p - number of pole pairs of motor Ts - Sampling time

angle_correction_type

Table 5 gives the type definition of angle_correction_type.

Table 5 · angle_correction_type

Name	angle_correction_type	
Type	typedef struct { picontrollertype pi_anglecorrection; int32_t rotor_electrical_anlge; int32_t rotor_speed; int32_t sinvalue; int32_t cosvalue; int32_t direction; int32_t posoffset; } angle_correction_type;	
File	Lib.h	
Range	picontrollertype pi_anglecorrection	The PI controller of the angle correction
	int32_t rotor_electrical_anlge	The rotor electrical angle
	int32_t rotor_speed	The speed of the rotor
	int32_t sinvalue	The sine value of the previous rotor electrical angle
	int32_t cosvalue	The cosine value of the previous rotor electrical angle
	int32_t direction	The direction of rotor.
	int32_t posoffset	The position offset

angle_calculation_type

Table 6 gives the type definition of angle_calculation_type.

Table 6 · angle_calculation_type

Name	angle_calculation_type
Type	typedef struct { stateobserver_type stateobserver; angle_correction_type angle_values; } angle_calculation_type;

File	Lib.h	
Range	stateobserver_type stateobserver	The state observer block
	angle_correction_type angle_values	The angle correction block

Speed Calculation Type Definitions

filter_type

Table 7 gives the type definition of filter_type.

Table 7 · filter_type

Name	filter_type	
Type	typedef struct { int32_t coeff_a; int32_t coeff_b; int32_t y1; } filter_type;	
File	Lib.h	
Range	int32_t coeff_a	The coefficient value A
	int32_t coeff_b	The coefficient value B
	int32_t y1	The filter output value

speedcalculation_type

Table 8 gives the type definition of speedcalculation_type:

Table 8 · speedcalculation_type

Name	speedcalculation_type	
Type	typedef struct { int32_t electrical_anlge; int32_t speed; int32_t direction; uint32_t calc_count; filter_type filter; } speedcalculation_type;	
File	Lib.h	
Range	int32_t electrical_anlge	The rotor electrical angle
	int32_t speed	The current speed of the motor
	int32_t direction	The direction of the motor. This value shall hold the numerical representation of clockwise or counter clock wise.
	uint32_t calc_count	The speed control rate
	filter_type filter	The filter structure that contains filter coefficient and output

API Functions Description

This section describes the functions required to implement the various tasks of angle and speed calculations.

Angle Calculation Functions Description

AngleCalc_Init

Table 9 describes the AngleCalc_Init function that is used to initialize all the values present in the angle calculation structure to zero.

Table 9 · Specification of API AngleCalc_Init

Syntax	void AngleCalc_Init(angle_calculation_type *anglecalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to the input structure
Parameters (Output)	anglecalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	This function initializes the angle calculation structure to zero.

AngleCalc_SetGainAndFactor

Table 10 describes the AngleCalc_SetGainAndFactor function that is used to set the angle calculation structure to the configured value.

Table 10 · Specification of API AngleCalc_SetGainAndFactor

Syntax	void AngleCalc_SetGainAndFactor(angle_calculation_type *anglecalc_ptr, int32_t estimatorfactor1, int32_t estimatorfactor2, int32_t estimatorfactor3, int32_t estimatorfactor4, int32_t estimatorfactor5,)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to angle calculation structure estimatorfactor1: calculated as R*Ts/Ls estimatorfactor2: calculated as Ts/Ls estimatorfactor3: calculated as K1*Ts estimatorfactor4: calculated as K2*Ts estimatorfactor5: calculated as P*Ts
Parameters (Output)	None
Return	None
Algorithm Description	This function sets the angle calculation structure to configured value.

AngleCalc_RotorAngle

Table 11 describes the AngleCalc_RotorAngle function that is used to calculate the rotor angle by following the algorithm mentioned in Figure 3 and Figure 4.

Table 11 · Specification of API AngleCalc_RotorAngle

Syntax	static inline AngleCalc_RotorAngle(angle_calculation_type *anglecalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to the input structure
Parameters (Output)	anglecalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	The flow of rotor angle calculation is represented in Figure 3 and Figure 4.

AngleCalc_RotorSpeed

Table 12 describes the AngleCalc_RotorSpeed function that is used to calculate the speed of the rotor by following the algorithm discussed in Figure 5.

Table 12 · Specification of API AngleCalc_RotorSpeed

Syntax	static inline AngleCalc_RotorSpeed(angle_calculation_type *anglecalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to the input structure
Parameters (Output)	anglecalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	The flow of rotor angle calculation is represented in Figure 5.

AngleCalc_SetEstBemfMinMax

Table 13 describes the AngleCalc_SetEstBemfMinMax function that is used to set the maximum and minimum limits of the estimated voltage value.

Table 13 · Specification of API AngleCalc_SetEstBemfMinMax

Syntax	void AngleCalc_SetEstVoltageMinMax(angle_calculation_type *anglecalc_ptr int32_t minvalue,int32_t maxvalue)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to the input structure minvalue: Lower limit value of the estimated voltage maxvalue: Upper limit value of the estimated voltage
Parameters (Output)	anglecalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	anglecalc_ptr ->voltage.estimated_min_limit = minvalue anglecalc_ptr ->voltage.estimated_max_limit = maxvalue

Figure 3 · Rotor Angle Calculation Flow Chart: Part 1

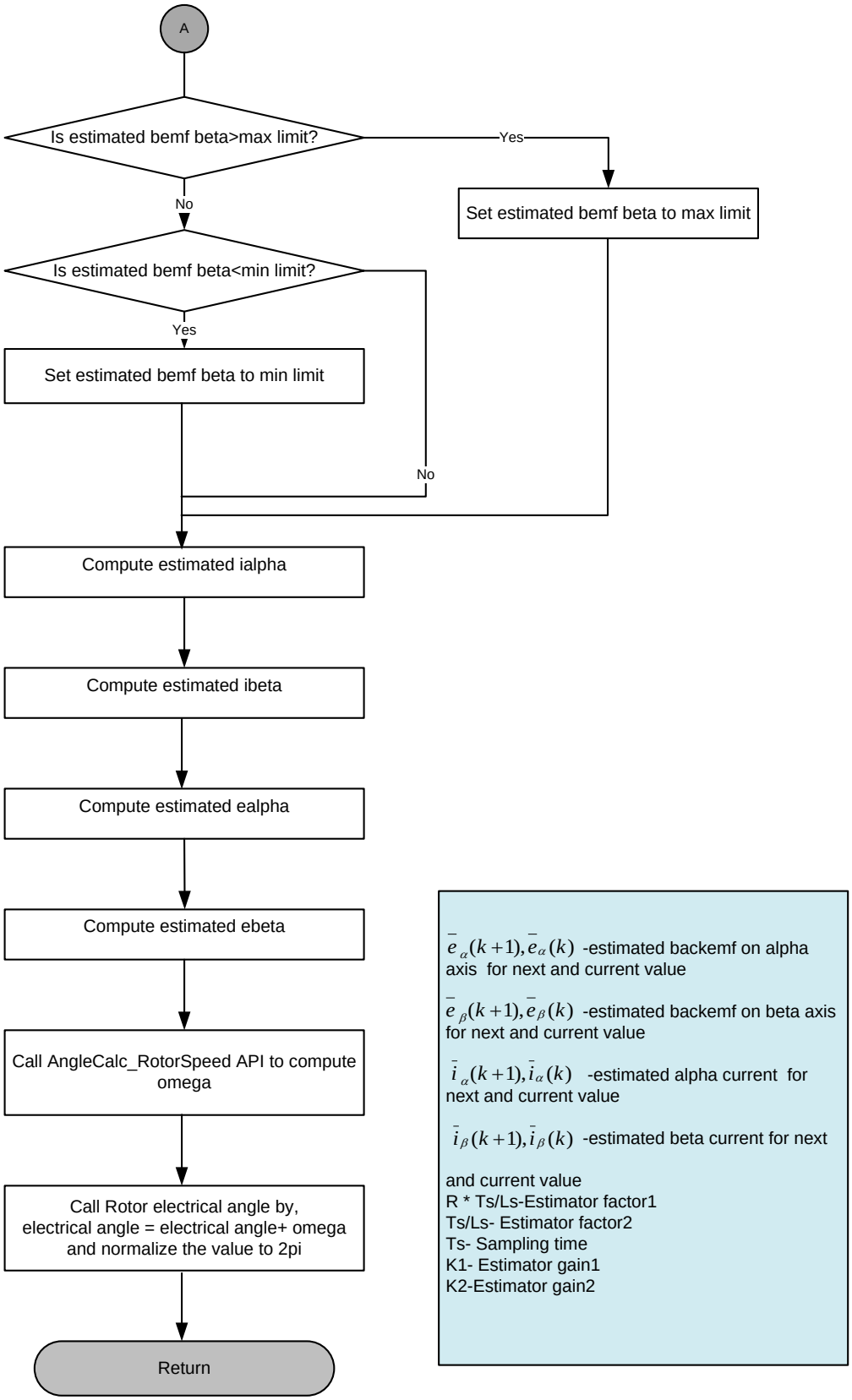

Figure 4 · Rotor Angle Calculation Flow Chart: Part 2

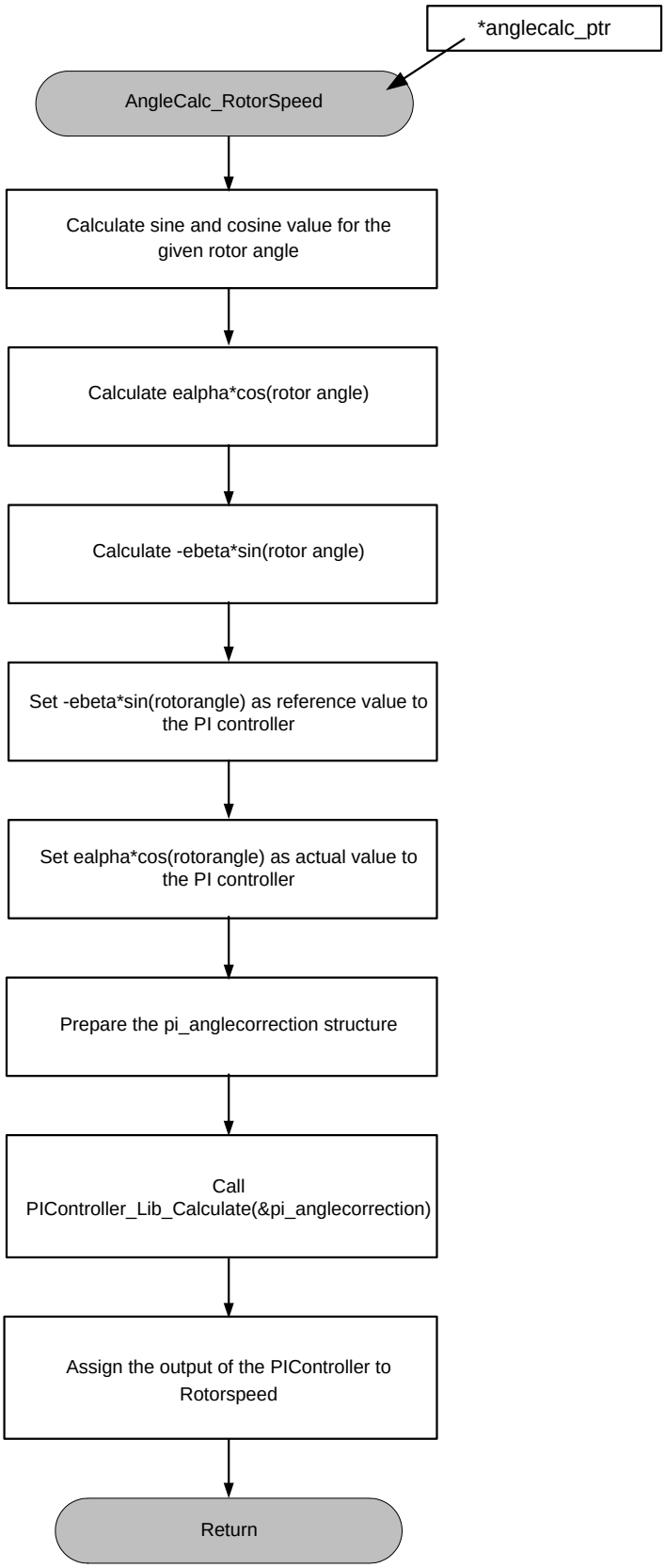

Figure 5 · Rotor Speed Calculation

AngleCalc_SetEstCurrentMinMax

Table 14 describes the AngleCalc_SetEstCurrentMinMax function that is used to set the minimum and maximum limits of the estimated current value.

Table 14 · Specification of API AngleCalc_SetEstCurrentMinMax

Syntax	void AngleCalc_SetEstCurrentMinMax(angle_calculation_type *anglecalc_ptr int32_t minvalue,int32_t maxvalue)
Re-entrancy	Re-entrant
Parameters (Inputs)	anglecalc_ptr: Pointer to the input structure minvalue: Lower limit value of the estimated current maxvalue: Upper limit value of the estimated current
Parameters (Output)	anglecalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	anglecalc_ptr -> current.estimated_min_limit = minvalue anglecalc_ptr -> current.estimated_max_limit = maxvalue

Speed Calculation Functions Description

SpeedCalc_Init

Table 15 describes the SpeedCalc_Init function that is used to initialize the speed calculation structure to zero.

Table 15 · Specification of API SpeedCalc_Init

Syntax	void SpeedCalc_Init(speedcalculation_type *speedcalc_ptr)
Re-entrancy	Non Re-entrant
Parameters (Inputs)	speedcalc_ptr: Pointer to the input structure
Parameters (Output)	speedcalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	This function initializes the speed calculation structure to zero

SpeedCalc_Set_FilterConst

Table 16 describes the SpeedCalc_Set_FilterConst function that is used to set the filter coefficients, ‘a’ and ‘b’ in speed calculation structure.

Table 16 · Specification of API SpeedCalc_Set_FilterConst

Syntax	void SpeedCalc_Set_FilterConst(speedcalculation_type *speedcalc_ptr, int32_t coeff_a,int32_t coeff_b)
Re-entrancy	Re-entrant
Parameters (Inputs)	speedcalc_ptr : Pointer to speed calculation structure coeff_a: filter coefficient A coeff_b: filter coefficient B
Parameters (Output)	None

Return	None
Algorithm Description	This function sets the speed calculation structure coefficient ‘a’ and ‘b‘ as mentioned below: speedcalc_ptr ->filter.coeff_a = coeff_a speedcalc_ptr ->filter.coeff_b = coeff_b

SpeedCalc_Filter

Table 17 describes the SpeedCalc_Filter function that is used for filter implementation.

Table 17 · Specification of API SpeedCalc_Filter

Syntax	void SpeedCalc_Filter(speedcalculation_type *speedcalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	speedcalc_ptr: Pointer to the input structure
Parameters (Output)	speedcalc_ptr: Pointer to the output structure
Return	None
Algorithm Description	The filter is implemented as below: speedcalc_ptr->filter.y1 = ((speedcalc_ptr->filter.coeff_a * input) - (speedcalc_ptr->filter.coeff_b * speedcalc_ptr->filter.y1) + speedcalc_ptr->filter.y1)

SpeedCalc_GetSpeed

Table 18 describes the SpeedCalc_GetSpeed function that is used to calculate the speed by differentiating the rotor angle (since dθ/dt = ωr).This functions returns the value of the speed of the rotor.

Table 18 · Specification of API SpeedCalc_GetSpeed

Syntax	int32_t SpeedCalc_GetSpeed(const speedcalculation_type *speedcalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	speedcalc_ptr: Pointer to the input structure
Parameters (Output)	speedcalc_ptr: Pointer to the output structure
Return	Speed
Algorithm Description	This API returns the speedcalc_ptr->filter.y1 value as speed.

SpeedCalc_GetDirection

Table 19 describes the SpeedCalc_GetDirection function, which is used to return a value that indicates the direction of rotation of the rotor.

Table 19 · Specification of API SpeedCalc_GetDirection

Syntax	int32_t SpeedCalc_GetDirection(const speedcalculation_type *speedcalc_ptr)
Re-entrancy	Re-entrant
Parameters (Inputs)	speedcalc_ptr: Pointer to the input structure
Parameters (Output)	speedcalc_ptr: Pointer to the output structure
Return	Direction
Algorithm Description	This API returns the speedcalc_ptr->direction value as direction.

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **408.643.6913**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Microsemi SoC Products Group Customer Support website for more information and support (http://www.microsemi.com/soc/support/search/default.aspx). Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

50200363-0/02.14