ANEXO B: DATASHEET DE LOS SENSORES Y EL MICROCONTROLADOR

INDICE DE CONTENIDOS

3-Axis, ±2 *g*/±4 *g*/±8 *g*/±16 *g* Digital Accelerometer

Data Sheet

ADXL345

FEATURES

- Ultralow power: as low as 23 µA in measurement mode and 0.1 µA in standby mode at V_S = 2.5 V (typical)
- Power consumption scales automatically with bandwidth
- User-selectable resolution
  - Fixed 10-bit resolution
  - Full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 mg/LSB scale factor in all *g* ranges)
- Patent pending, embedded memory management system with FIFO technology minimizes host processor load
- Single tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to V_S
- SPI (3- and 4-wire) and I^2C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (−40°C to +85°C)
- 10,000 *g* shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

- Handsets
- Medical instrumentation
- Gaming and pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive (HDD) protection

GENERAL DESCRIPTION

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I^2C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on any axis with user-set thresholds. Tap sensing detects single and double taps in any direction. Free-fall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated, patent pending memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

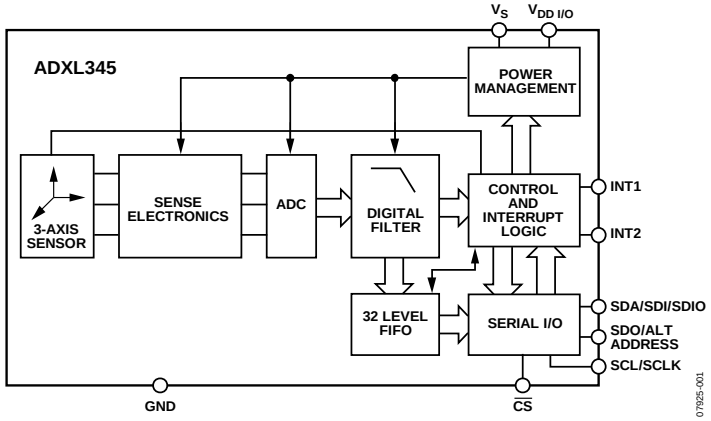


Figure 1.

Rev. D

Document Feedback

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700
Technical Support www.analog.com

TABLE OF CONTENTS

REVISION HISTORY

2/13—Rev. C to Rev. D

Changes to Figure 13, Figure 14, and Figure 15 9
Change to Table 15 22

5/11—Rev. B to Rev. C

Added Preventing Bus Traffic Errors Section 15
Changes to Figure 37, Figure 38, Figure 39 16
Changes to Table 12 19
Changes to Using Self-Test Section 31
Changes to Axes of Acceleration Sensitivity Section 35

11/10—Rev. A to Rev. B

Change to 0 *g* Offset vs. Temperature for Z-Axis Parameter, Table 1 4
Changes to Figure 10 to Figure 15 9
Changes to Ordering Guide 37

4/10—Rev. 0 to Rev. A

Changes to Features Section and General Description Section 1
Changes to Specifications Section 3
Changes to Table 2 and Table 3 5
Added Package Information Section, Figure 2, and Table 4; Renumbered Sequentially 5
Changes to Pin 12 Description, Table 5 6
Added Typical Performance Characteristics Section 7
Changes to Theory of Operation Section and Power Sequencing Section 12
Changes to Powers Savings Section, Table 7, Table 8, Auto Sleep Mode Section, and Standby Mode Section 13
Changes to SPI Section 14
Changes to Figure 36 to Figure 38 15
Changes to Table 9 and Table 10 16
Changes to I²C Section and Table 11 17
Changes to Table 12 18
Changes to Interrupts Section, Activity Section, Inactivity Section, and FREE_FALL Section 19
Added Table 13 19
Changes to FIFO Section 20
Changes to Self-Test Section and Table 15 to Table 18 21
Added Figures 42 and Table 14 21
Changes to Table 19 22
Changes to Register 0x1D—THRESH_TAP (Read/Write) Section, Register 0x1E, Register 0x1F, Register 0x20—OFSX, OFSY, OSXZ (Read/Write) Section, Register 0x21—DUR (Read/Write) Section, Register 0x22—Latent (Read/Write) Section, and Register 0x23—Window (Read/Write) Section ... 23
Changes to ACT_X Enable Bits and INACT_X Enable Bit Section, Register 0x28—THRESH_FF (Read/Write) Section, Register 0x29—TIME_FF (Read/Write) Section, Asleep Bit Section, and AUTO_SLEEP Bit Section 24
Changes to Sleep Bit Section 25
Changes to Power Supply Decoupling Section, Mechanical Considerations for Mounting Section, and Tap Detection Section 27
Changes to Threshold Section 28
Changes to Sleep Mode vs. Low Power Mode Section 29
Added Offset Calibration Section 29
Changes to Using Self-Test Section 30
Added Data Formatting of Upper Data Rates Section, Figure 48, and Figure 49 31
Added Noise Performance Section, Figure 50 to Figure 52, and Operation at Voltages Other Than 2.5 V Section 32
Added Offset Performance at Lowest Data Rates Section and Figure 53 to Figure 55 33

6/09—Revision 0: Initial Version

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 10 µF tantalum, $C_{I/O}$ = 0.1 µF, output data rate (ODR) = 800 Hz, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8, ±16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
OUTPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	230	256	282	LSB/*g*
	±2 *g*, 10-bit resolution	230	256	282	LSB/*g*
	±4 *g*, 10-bit resolution	115	128	141	LSB/*g*
	±8 *g*, 10-bit resolution	57	64	71	LSB/*g*
	±16 *g*, 10-bit resolution	29	32	35	LSB/*g*
Sensitivity Deviation from Ideal	All *g*-ranges		±1.0		%
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	All *g*-ranges, full resolution	3.5	3.9	4.3	mg/LSB
	±2 *g*, 10-bit resolution	3.5	3.9	4.3	mg/LSB
	±4 *g*, 10-bit resolution	7.1	7.8	8.7	mg/LSB
	±8 *g*, 10-bit resolution	14.1	15.6	17.5	mg/LSB
	±16 *g*, 10-bit resolution	28.6	31.2	34.5	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C
0 *g* OFFSET	Each axis				
0 *g* Output for X_{OUT}, Y_{OUT}		−150	0	+150	m*g*
0 *g* Output for Z_{OUT}		−250	0	+250	m*g*
0 *g* Output Deviation from Ideal, X_{OUT}, Y_{OUT}			±35		m*g*
0 *g* Output Deviation from Ideal, Z_{OUT}			±40		m*g*
0 *g* Offset vs. Temperature for X-, Y-Axes			±0.4		mg/°C
0 *g* Offset vs. Temperature for Z-Axis			±1.2		mg/°C
NOISE					
X-, Y-Axes	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		0.75		LSB rms
Z-Axis	ODR = 100 Hz for ±2 *g*, 10-bit resolution or all *g*-ranges, full resolution		1.1		LSB rms
OUTPUT DATA RATE AND BANDWIDTH	User selectable				
Output Data Rate (ODR)[3, 4, 5]		0.1		3200	Hz
SELF-TEST[6]					
Output Change in X-Axis		0.20		2.10	*g*
Output Change in Y-Axis		−2.10		−0.20	*g*
Output Change in Z-Axis		0.30		3.40	*g*
POWER SUPPLY					
Operating Voltage Range (V_S)		2.0	2.5	3.6	V
Interface Voltage Range ($V_{DD\ I/O}$)		1.7	1.8	V_S	V
Supply Current	ODR ≥ 100 Hz		140		µA
	ODR < 10 Hz		30		µA
Standby Mode Leakage Current			0.1		µA
Turn-On and Wake-Up Time[7]	ODR = 3200 Hz		1.4		ms

Parameter	Test Conditions	Min	Typ[1]	Max	Unit
TEMPERATURE					
Operating Temperature Range		−40		+85	°C
WEIGHT					
Device Weight			30		m*g*

[1] The typical specifications shown are for at least 68% of the population of parts and are based on the worst case of mean ±1 σ, except for 0 *g* output and sensitivity, which represents the target value. For 0 *g* offset and sensitivity, the deviation from the ideal describes the worst case of mean ±1 σ.
[2] Cross-axis sensitivity is defined as coupling between any two axes.
[3] Bandwidth is the −3 dB frequency and is half the output data rate, bandwidth = ODR/2.
[4] The output format for the 3200 Hz and 1600 Hz ODRs is different than the output format for the remaining ODRs. This difference is described in the Data Formatting of Upper Data Rates section.
[5] Output data rates below 6.25 Hz exhibit additional offset shift with increased temperature, depending on selected output data rate. Refer to the Offset Performance at Lowest Data Rates section for details.
[6] Self-test change is defined as the output (*g*) when the SELF_TEST bit = 1 (in the DATA_FORMAT register, Address 0x31) minus the output (*g*) when the SELF_TEST bit = 0. Due to device filtering, the output reaches its final value after 4 × τ when enabling or disabling self-test, where τ = 1/(data rate). The part must be in normal power operation (LOW_POWER bit = 0 in the BW_RATE register, Address 0x2C) for self-test to operate correctly.
[7] Turn-on and wake-up times are determined by the user-defined bandwidth. At a 100 Hz data rate, the turn-on and wake-up times are each approximately 11.1 ms. For other data rates, the turn-on and wake-up times are each approximately τ + 1.1 in milliseconds, where τ = 1/(data rate).

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter	Rating
Acceleration	
Any Axis, Unpowered	10,000 *g*
Any Axis, Powered	10,000 *g*
V_S	−0.3 V to +3.9 V
$V_{DD\ I/O}$	−0.3 V to +3.9 V
Digital Pins	−0.3 V to $V_{DD\ I/O}$ + 0.3 V or 3.9 V, whichever is less
All Other Pins	−0.3 V to +3.9 V
Output Short-Circuit Duration (Any Pin to Ground)	Indefinite
Temperature Range	
Powered	−40°C to +105°C
Storage	−40°C to +105°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

THERMAL RESISTANCE

Table 3. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
14-Terminal LGA	150°C/W	85°C/W	30 mg

PACKAGE INFORMATION

The information in Figure 2 and Table 4 provide details about the package branding for the ADXL345. For a complete listing of product availability, see the Ordering Guide section.

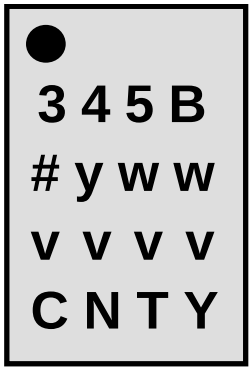


Figure 2. Product Information on Package (Top View)

Table 4. Package Branding Information

Branding Key	Field Description
345B	Part identifier for ADXL345
#	RoHS-compliant designation
yww	Date code
vvvv	Factory lot code
CNTY	Country of origin

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

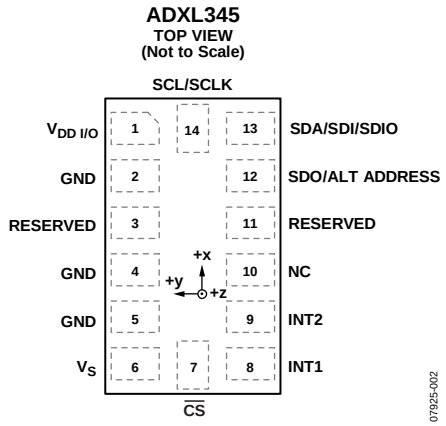


Figure 3. Pin Configuration (Top View)

Table 5. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	This pin must be connected to ground.
3	RESERVED	Reserved. This pin must be connected to V_S or left open.
4	GND	This pin must be connected to ground.
5	GND	This pin must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	RESERVED	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output (SPI 4-Wire)/Alternate I^2C Address Select (I^2C).
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock. SCL is the clock for I^2C, and SCLK is the clock for SPI.

TYPICAL PERFORMANCE CHARACTERISTICS

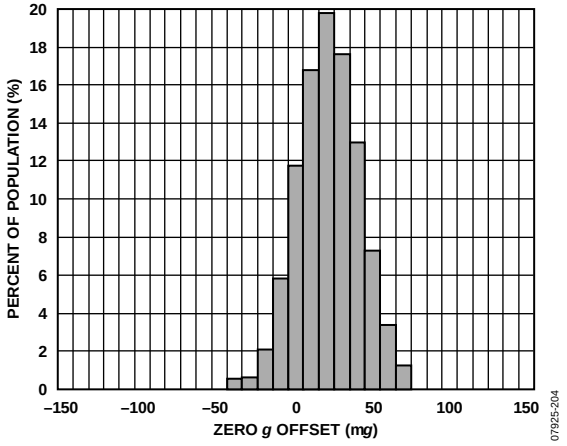


Figure 4. X-Axis Zero g Offset at 25°C, V_S = 2.5 V

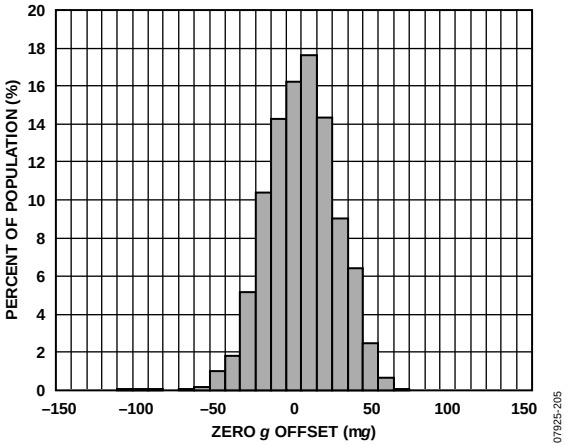


Figure 5. Y-Axis Zero g Offset at 25°C, V_S = 2.5 V

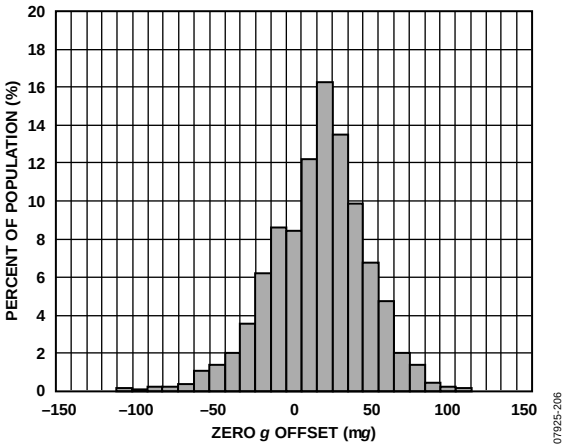


Figure 6. Z-Axis Zero g Offset at 25°C, V_S = 2.5 V

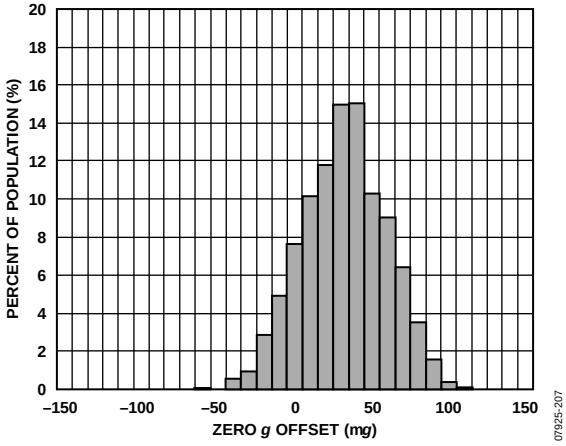


Figure 7. X-Axis Zero g Offset at 25°C, V_S = 3.3 V

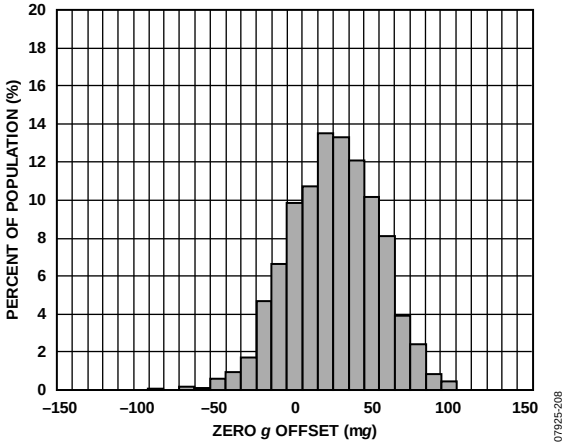


Figure 8. Y-Axis Zero g Offset at 25°C, V_S = 3.3 V

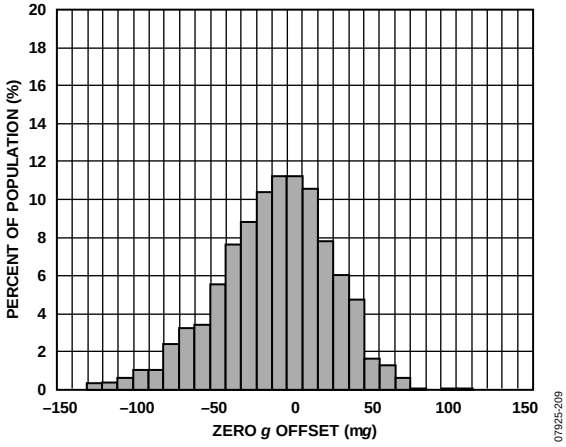


Figure 9. Z-Axis Zero g Offset at 25°C, V_S = 3.3 V

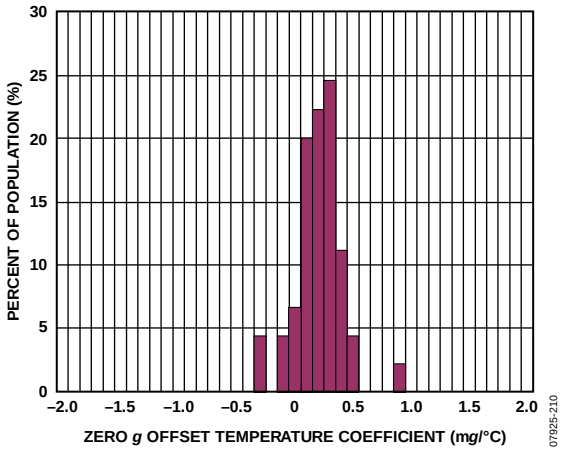


Figure 10. X-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

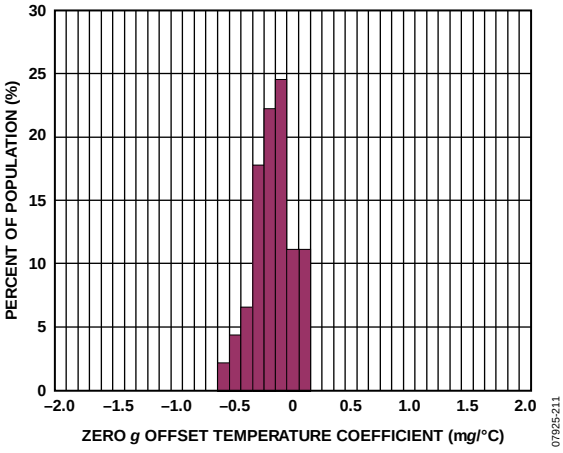


Figure 11. Y-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

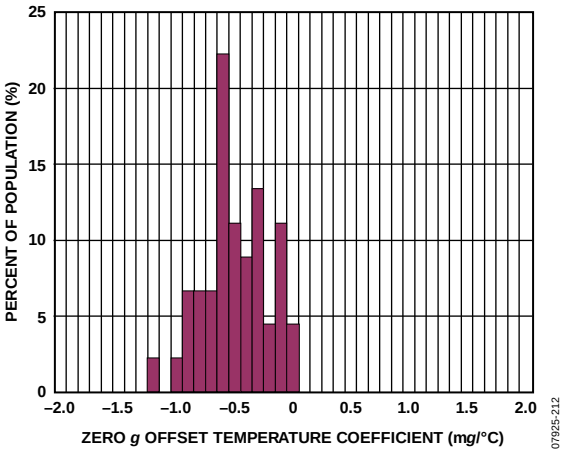


Figure 12. Z-Axis Zero g Offset Temperature Coefficient, V_S = 2.5 V

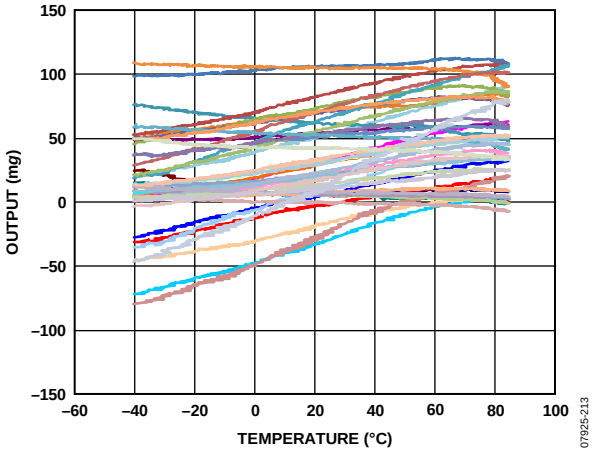


Figure 13. X-Axis Zero g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

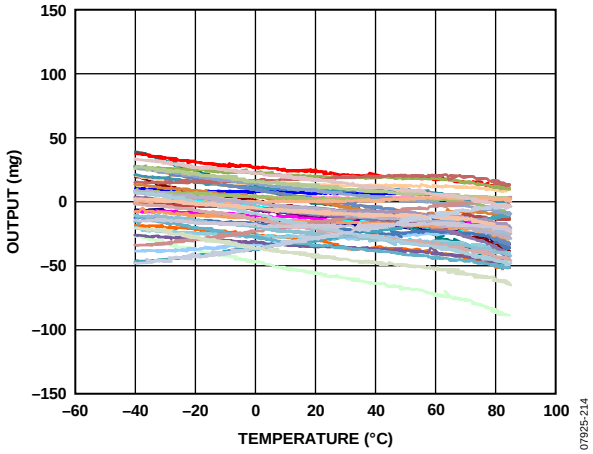


Figure 14. Y-Axis Zero g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

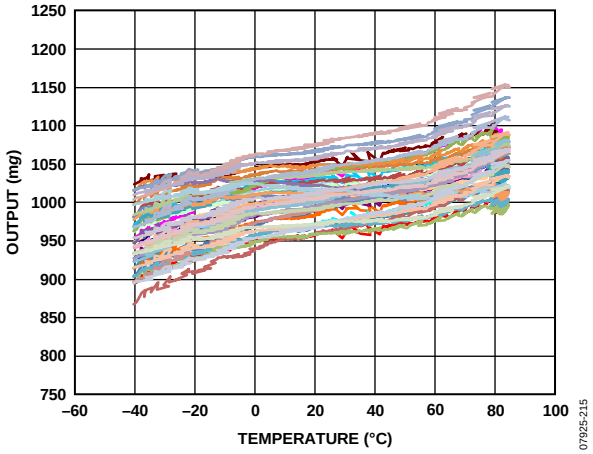


Figure 15. Z-Axis One g Offset vs. Temperature—45 Parts Soldered to PCB, V_S = 2.5 V

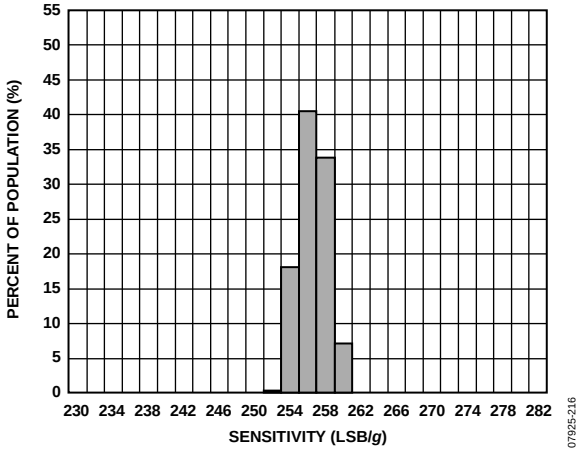


Figure 16. X-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

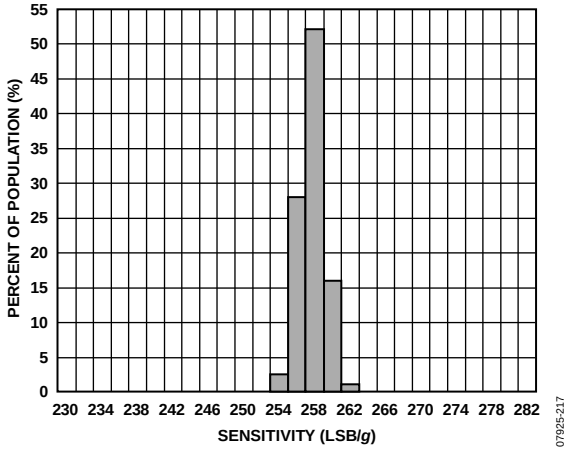


Figure 17. Y-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

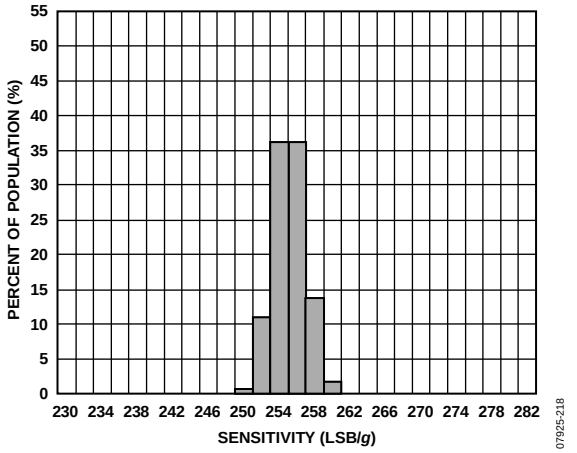


Figure 18. Z-Axis Sensitivity at 25°C, V_S = 2.5 V, Full Resolution

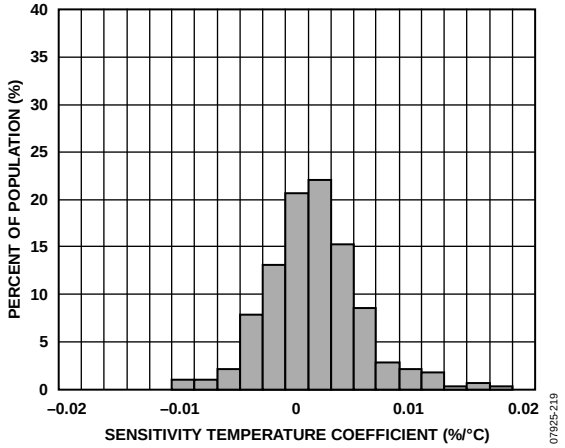


Figure 19. X-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

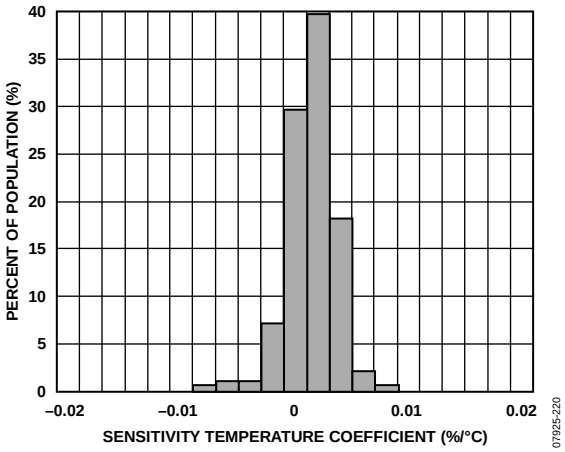


Figure 20. Y-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

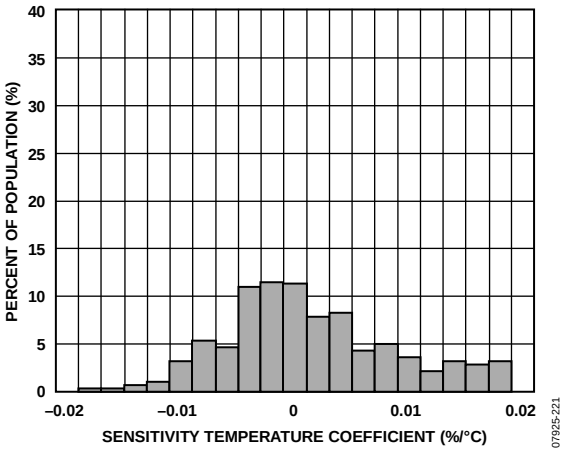


Figure 21. Z-Axis Sensitivity Temperature Coefficient, V_S = 2.5 V

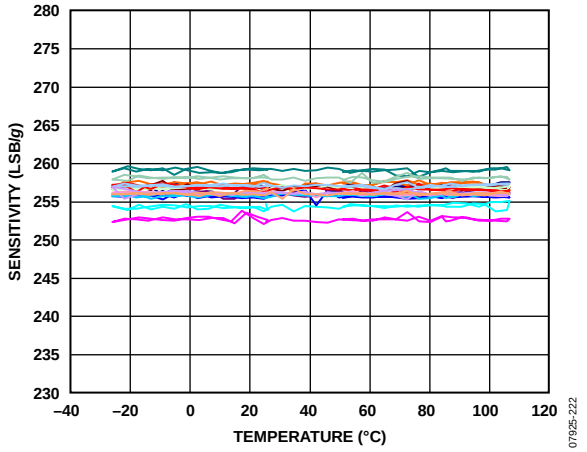


Figure 22. X-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

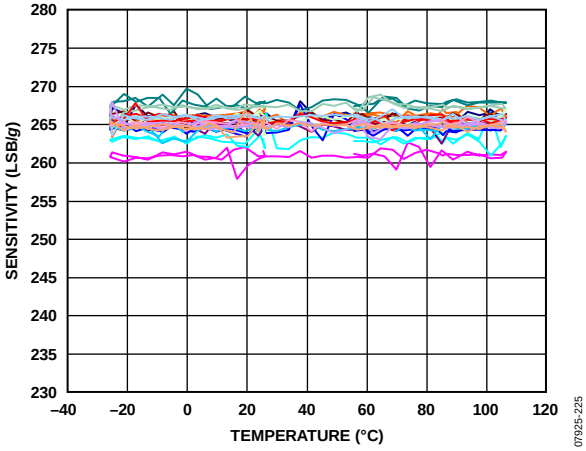


Figure 25. X-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

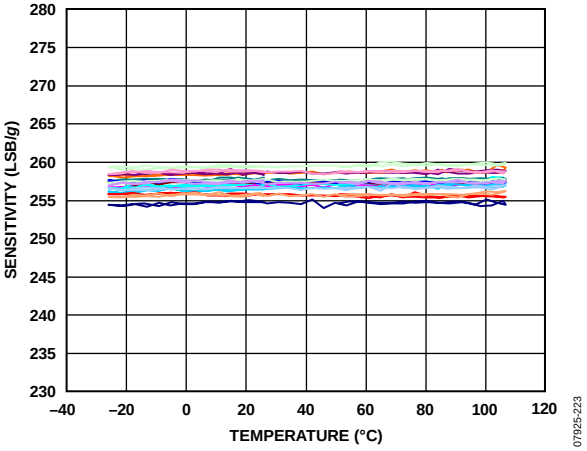


Figure 23. Y-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

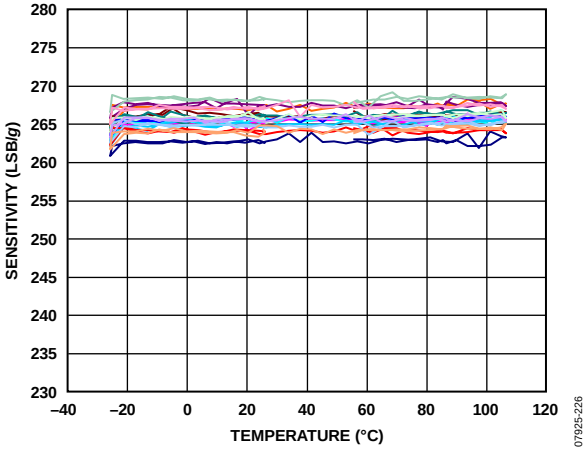


Figure 26. Y-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

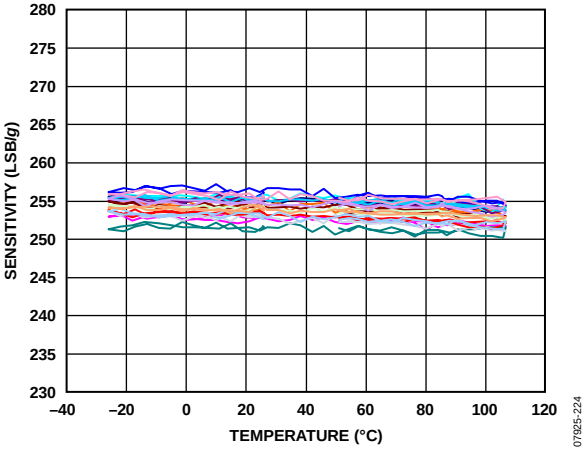


Figure 24. Z-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 2.5 V, Full Resolution

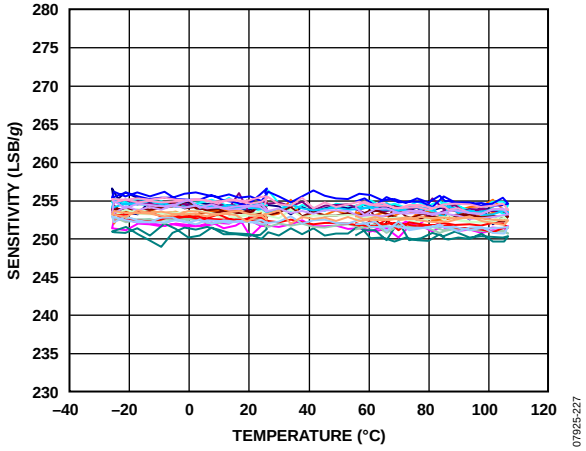


Figure 27. Z-Axis Sensitivity vs. Temperature—
Eight Parts Soldered to PCB, V_S = 3.3 V, Full Resolution

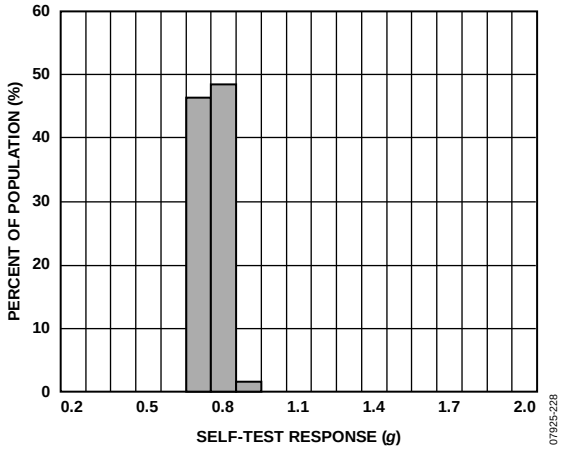


Figure 28. X-Axis Self-Test Response at 25°C, V_S = 2.5 V

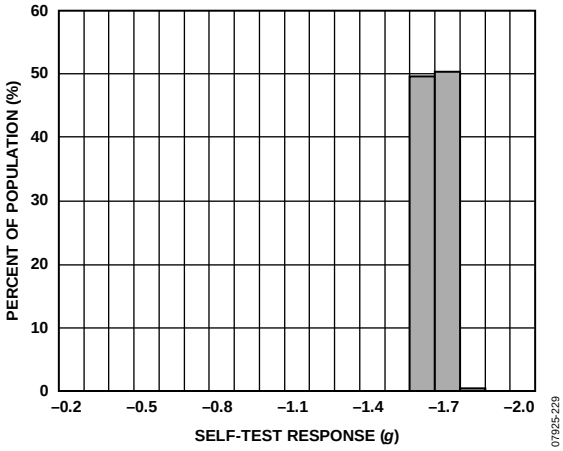


Figure 29. Y-Axis Self-Test Response at 25°C, V_S = 2.5 V

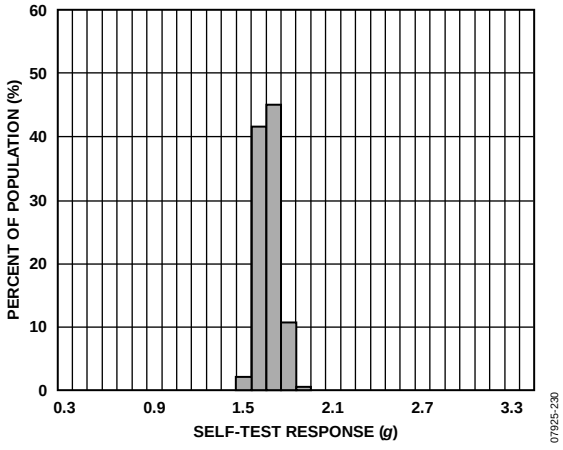


Figure 30. Z-Axis Self-Test Response at 25°C, V_S = 2.5 V

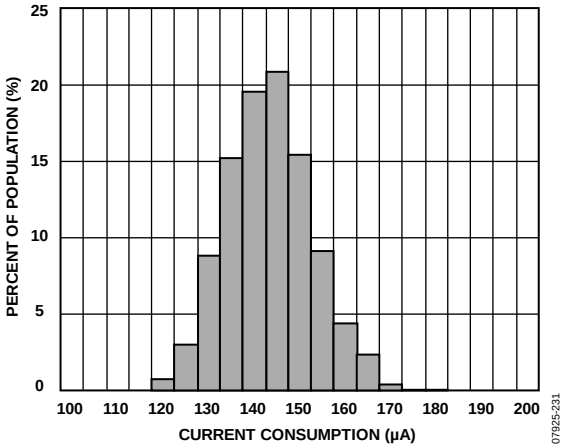


Figure 31. Current Consumption at 25°C, 100 Hz Output Data Rate, V_S = 2.5 V

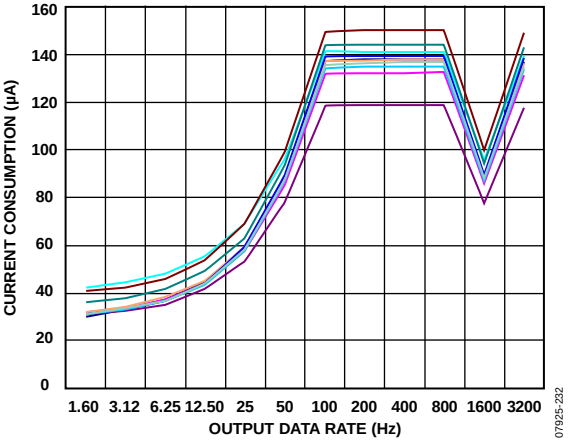


Figure 32. Current Consumption vs. Output Data Rate at 25°C—10 Parts, V_S = 2.5 V

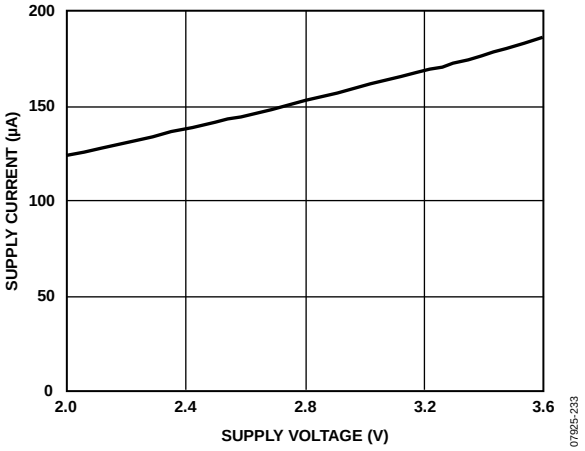


Figure 33. Supply Current vs. Supply Voltage, V_S at 25°C

THEORY OF OPERATION

The ADXL345 is a complete 3-axis acceleration measurement system with a selectable measurement range of ±2 *g*, ±4 *g*, ±8 *g*, or ±16 *g*. It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, that allows the device to be used as a tilt sensor.

The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against forces due to applied acceleration.

Deflection of the structure is measured using differential capacitors that consist of independent fixed plates and plates attached to the moving mass. Acceleration deflects the proof mass and unbalances the differential capacitor, resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation is used to determine the magnitude and polarity of the acceleration.

POWER SEQUENCING

Power can be applied to V_S or $V_{DD\ I/O}$ in any sequence without damaging the ADXL345. All possible power-on modes are summarized in Table 6. The interface voltage level is set with the interface supply voltage, $V_{DD\ I/O}$, which must be present to ensure that the ADXL345 does not create a conflict on the communication bus. For single-supply operation, $V_{DD\ I/O}$ can be the same as the main supply, V_S. In a dual-supply application, however, $V_{DD\ I/O}$ can differ from V_S to accommodate the desired interface voltage, as long as V_S is greater than or equal to $V_{DD\ I/O}$.

After V_S is applied, the device enters standby mode, where power consumption is minimized and the device waits for $V_{DD\ I/O}$ to be applied and for the command to enter measurement mode to be received. (This command can be initiated by setting the measure bit (Bit D3) in the POWER_CTL register (Address 0x2D).) In addition, while the device is in standby mode, any register can be written to or read from to configure the part. It is recommended to configure the device in standby mode and then to enable measurement mode. Clearing the measure bit returns the device to the standby mode.

Table 6. Power Sequencing

Condition	V_S	$V_{DD\ I/O}$	Description
Power Off	Off	Off	The device is completely off, but there is a potential for a communication bus conflict.
Bus Disabled	On	Off	The device is on in standby mode, but communication is unavailable and creates a conflict on the communication bus. The duration of this state should be minimized during power-up to prevent a conflict.
Bus Enabled	Off	On	No functions are available, but the device does not create a conflict on the communication bus.
Standby or Measurement	On	On	At power-up, the device is in standby mode, awaiting a command to enter measurement mode, and all sensor functions are off. After the device is instructed to enter measurement mode, all sensor functions are available.

POWER SAVINGS

Power Modes

The ADXL345 automatically modulates its power consumption in proportion to its output data rate, as outlined in Table 7. If additional power savings is desired, a lower power mode is available. In this mode, the internal sampling rate is reduced, allowing for power savings in the 12.5 Hz to 400 Hz data rate range at the expense of slightly greater noise. To enter low power mode, set the LOW_POWER bit (Bit 4) in the BW_RATE register (Address 0x2C). The current consumption in low power mode is shown in Table 8 for cases where there is an advantage to using low power mode. Use of low power mode for a data rate not shown in Table 8 does not provide any advantage over the same data rate in normal power mode. Therefore, it is recommended that only data rates shown in Table 8 are used in low power mode. The current consumption values shown in Table 7 and Table 8 are for a V_S of 2.5 V.

Table 7. Typical Current Consumption vs. Data Rate (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
3200	1600	1111	140
1600	800	1110	90
800	400	1101	140
400	200	1100	140
200	100	1011	140
100	50	1010	140
50	25	1001	90
25	12.5	1000	60
12.5	6.25	0111	50
6.25	3.13	0110	45
3.13	1.56	0101	40
1.56	0.78	0100	34
0.78	0.39	0011	23
0.39	0.20	0010	23
0.20	0.10	0001	23
0.10	0.05	0000	23

Table 8. Typical Current Consumption vs. Data Rate, Low Power Mode (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
400	200	1100	90
200	100	1011	60
100	50	1010	50
50	25	1001	45
25	12.5	1000	40
12.5	6.25	0111	34

Auto Sleep Mode

Additional power can be saved if the ADXL345 automatically switches to sleep mode during periods of inactivity. To enable this feature, set the THRESH_INACT register (Address 0x25) and the TIME_INACT register (Address 0x26) each to a value that signifies inactivity (the appropriate value depends on the application), and then set the AUTO_SLEEP bit (Bit D4) and the link bit (Bit D5) in the POWER_CTL register (Address 0x2D). Current consumption at the sub-12.5 Hz data rates that are used in this mode is typically 23 μA for a V_S of 2.5 V.

Standby Mode

For even lower power operation, standby mode can be used. In standby mode, current consumption is reduced to 0.1 μA (typical). In this mode, no measurements are made. Standby mode is entered by clearing the measure bit (Bit D3) in the POWER_CTL register (Address 0x2D). Placing the device into standby mode preserves the contents of FIFO.

SERIAL COMMUNICATIONS

I²C and SPI digital communications are available. In both cases, the ADXL345 operates as a slave. I²C mode is enabled if the $\overline{CS}$ pin is tied high to $V_{DD\ I/O}$. The $\overline{CS}$ pin should always be tied high to $V_{DD\ I/O}$ or be driven by an external controller because there is no default mode if the $\overline{CS}$ pin is left unconnected. Therefore, not taking these precautions may result in an inability to communicate with the part. In SPI mode, the $\overline{CS}$ pin is controlled by the bus master. In both SPI and I²C modes of operation, data transmitted from the ADXL345 to the master device should be ignored during writes to the ADXL345.

SPI

For SPI, either 3- or 4-wire configuration is possible, as shown in the connection diagrams in Figure 34 and Figure 35. Clearing the SPI bit (Bit D6) in the DATA_FORMAT register (Address 0x31) selects 4-wire mode, whereas setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz with 100 pF maximum loading, and the timing scheme follows clock polarity (CPOL) = 1 and clock phase (CPHA) = 1. If power is applied to the ADXL345 before the clock polarity and phase of the host processor are configured, the $\overline{CS}$ pin should be brought high before changing the clock polarity and phase. When using 3-wire SPI, it is recommended that the SDO pin be either pulled up to $V_{DD\ I/O}$ or pulled down to GND via a 10 kΩ resistor.

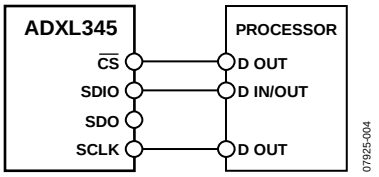


Figure 34. 3-Wire SPI Connection Diagram

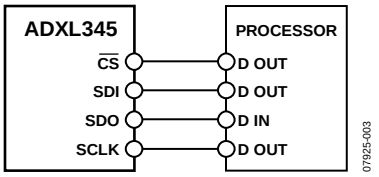


Figure 35. 4-Wire SPI Connection Diagram

$\overline{CS}$ is the serial port enable line and is controlled by the SPI master. This line must go low at the start of a transmission and high at the end of a transmission, as shown in Figure 37. SCLK is the serial port clock and is supplied by the SPI master. SCLK should idle high during a period of no transmission. SDI and SDO are the serial data input and output, respectively. Data is updated on the falling edge of SCLK and should be sampled on the rising edge of SCLK.

To read or write multiple bytes in a single transmission, the multiple-byte bit, located after the R/$\overline{W}$ bit in the first byte transfer (MB in Figure 37 to Figure 39), must be set. After the register addressing and the first byte of data, each subsequent set of clock pulses (eight clock pulses) causes the ADXL345 to point to the next register for a read or write. This shifting continues until the clock pulses cease and $\overline{CS}$ is deasserted. To perform reads or writes on different, nonsequential registers, $\overline{CS}$ must be deasserted between transmissions and the new register must be addressed separately.

The timing diagram for 3-wire SPI reads or writes is shown in Figure 39. The 4-wire equivalents for SPI writes and reads are shown in Figure 37 and Figure 38, respectively. For correct operation of the part, the logic thresholds and timing parameters in Table 9 and Table 10 must be met at all times.

Use of the 3200 Hz and 1600 Hz output data rates is only recommended with SPI communication rates greater than or equal to 2 MHz. The 800 Hz output data rate is recommended only for communication speeds greater than or equal to 400 kHz, and the remaining data rates scale proportionally. For example, the minimum recommended communication speed for a 200 Hz output data rate is 100 kHz. Operation at an output data rate above the recommended maximum may result in undesirable effects on the acceleration data, including missing samples or additional noise.

Preventing Bus Traffic Errors

The ADXL346 $\overline{CS}$ pin is used both for initiating SPI transactions, and for enabling I²C mode. When the ADXL346 is used on a SPI bus with multiple devices, its $\overline{CS}$ pin is held high while the master communicates with the other devices. There may be conditions where a SPI command transmitted to another device looks like a valid I²C command. In this case, the ADXL346 would interpret this as an attempt to communicate in I²C mode, and could interfere with other bus traffic. Unless bus traffic can be adequately controlled to assure such a condition never occurs, it is recommended to add a logic gate in front of the SDI pin as shown in Figure 36. This OR gate will hold the SDA line high when $\overline{CS}$ is high to prevent SPI bus traffic at the ADXL346 from appearing as an I²C start command.

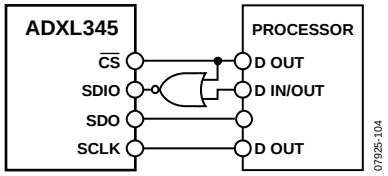


Figure 36. Recommended SPI Connection Diagram when Using Multiple SPI Devices on a Single Bus

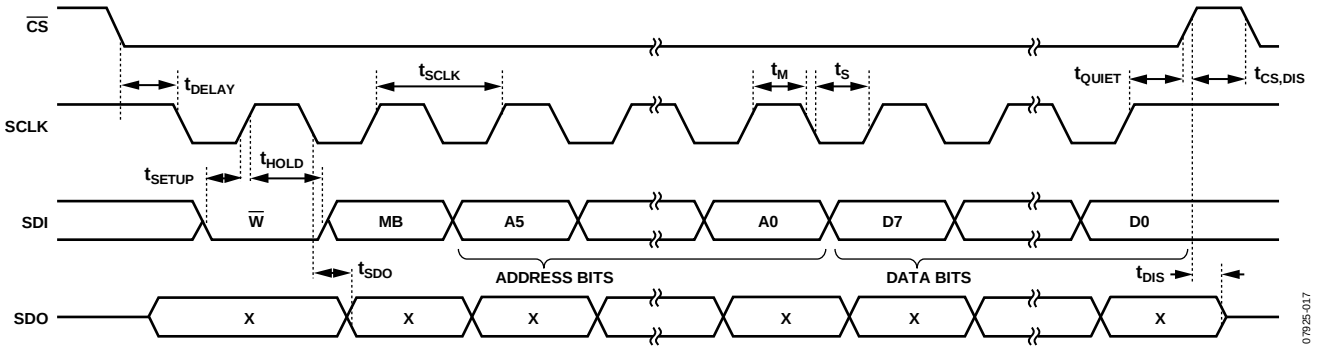


Figure 37. SPI 4-Wire Write

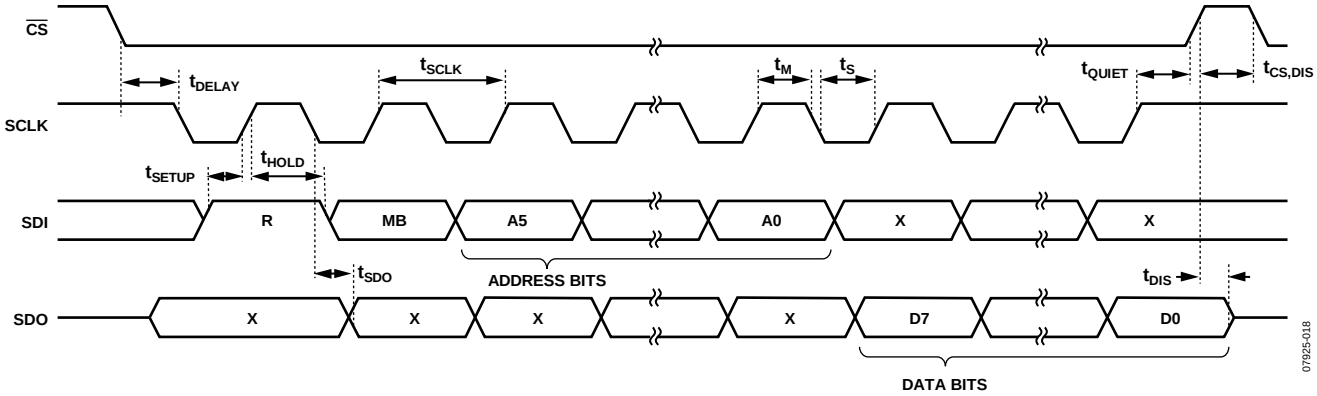


Figure 38. SPI 4-Wire Read

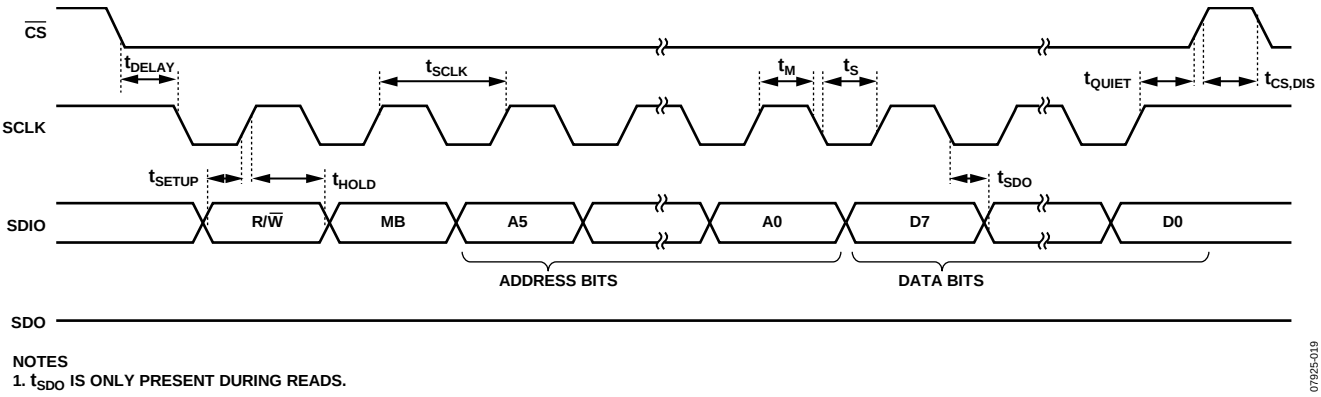


Figure 39. SPI 3-Wire Read/Write

Table 9. SPI Digital Input/Output

Parameter	Test Conditions	Limit[1] Min	Limit[1] Max	Unit
Digital Input				
Low Level Input Voltage (V_{IL})			$0.3 \times V_{DD\ I/O}$	V
High Level Input Voltage (V_{IH})		$0.7 \times V_{DD\ I/O}$		V
Low Level Input Current (I_{IL})	$V_{IN} = V_{DD\ I/O}$		0.1	μA
High Level Input Current (I_{IH})	$V_{IN} = 0$ V	−0.1		μA
Digital Output				
Low Level Output Voltage (V_{OL})	$I_{OL} = 10$ mA		$0.2 \times V_{DD\ I/O}$	V
High Level Output Voltage (V_{OH})	$I_{OH} = -4$ mA	$0.8 \times V_{DD\ I/O}$		V
Low Level Output Current (I_{OL})	$V_{OL} = V_{OL,\ max}$	10		mA
High Level Output Current (I_{OH})	$V_{OH} = V_{OH,\ min}$		−4	mA
Pin Capacitance	$f_{IN} = 1$ MHz, $V_{IN} = 2.5$ V		8	pF

[1] Limits based on characterization results, not production tested.

Table 10. SPI Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)[1]

Parameter	Limit[2, 3] Min	Limit[2, 3] Max	Unit	Description
f_{SCLK}		5	MHz	SPI clock frequency
t_{SCLK}	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
t_{DELAY}	5		ns	$\overline{CS}$ falling edge to SCLK falling edge
t_{QUIET}	5		ns	SCLK rising edge to $\overline{CS}$ rising edge
t_{DIS}		10	ns	$\overline{CS}$ rising edge to SDO disabled
$t_{CS,DIS}$	150		ns	$\overline{CS}$ deassertion between SPI communications
t_S	$0.3 \times t_{SCLK}$		ns	SCLK low pulse width (space)
t_M	$0.3 \times t_{SCLK}$		ns	SCLK high pulse width (mark)
t_{SETUP}	5		ns	SDI valid before SCLK rising edge
t_{HOLD}	5		ns	SDI valid after SCLK rising edge
t_{SDO}		40	ns	SCLK falling edge to SDO/SDIO output transition
t_R[4]		20	ns	SDO/SDIO output high to output low transition
t_F[4]		20	ns	SDO/SDIO output low to output high transition

[1] The $\overline{CS}$, SCLK, SDI, and SDO pins are not internally pulled up or down; they must be driven for proper operation.
[2] Limits based on characterization results, characterized with f_{SCLK} = 5 MHz and bus load capacitance of 100 pF; not production tested.
[3] The timing values are measured corresponding to the input thresholds (V_{IL} and V_{IH}) given in Table 9.
[4] Output rise and fall times measured with capacitive load of 150 pF.

I²C

With $\overline{CS}$ tied high to $V_{DD\ I/O}$, the ADXL345 is in I²C mode, requiring a simple 2-wire connection, as shown in Figure 40. The ADXL345 conforms to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the bus parameters given in Table 11 and Table 12 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 41. With the ALT ADDRESS pin high, the 7-bit I²C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

There are no internal pull-up or pull-down resistors for any unused pins; therefore, there is no known state or default state for the $\overline{CS}$ or ALT ADDRESS pin if left floating or unconnected. It is required that the $\overline{CS}$ pin be connected to $V_{DD\ I/O}$ and that the ALT ADDRESS pin be connected to either $V_{DD\ I/O}$ or GND when using I²C.

Due to communication speed limitations, the maximum output data rate when using 400 kHz I²C is 800 Hz and scales linearly with a change in the I²C communication speed. For example, using I²C at 100 kHz would limit the maximum ODR to 200 Hz. Operation at an output data rate above the recommended maximum may result in undesirable effect on the acceleration data, including missing samples or additional noise.

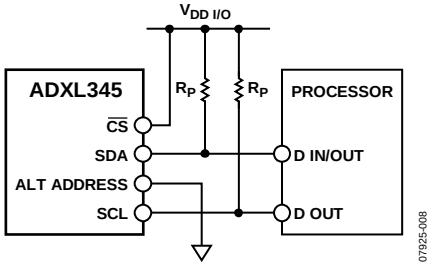


Figure 40. I²C Connection Diagram (Address 0x53)

If other devices are connected to the same I²C bus, the nominal operating voltage level of these other devices cannot exceed $V_{DD\ I/O}$ by more than 0.3 V. External pull-up resistors, R_P, are necessary for proper I²C operation. Refer to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 11. I²C Digital Input/Output

Parameter	Test Conditions	Limit[1] Min	Limit[1] Max	Unit
Digital Input				
Low Level Input Voltage (V_{IL})			$0.3 \times V_{DD\ I/O}$	V
High Level Input Voltage (V_{IH})		$0.7 \times V_{DD\ I/O}$		V
Low Level Input Current (I_{IL})	$V_{IN} = V_{DD\ I/O}$		0.1	μA
High Level Input Current (I_{IH})	$V_{IN} = 0$ V	−0.1		μA
Digital Output				
Low Level Output Voltage (V_{OL})	$V_{DD\ I/O} < 2$ V, $I_{OL} = 3$ mA		$0.2 \times V_{DD\ I/O}$	V
	$V_{DD\ I/O} \geq 2$ V, $I_{OL} = 3$ mA		400	mV
Low Level Output Current (I_{OL})	$V_{OL} = V_{OL,\ max}$	3		mA
Pin Capacitance	$f_{IN} = 1$ MHz, $V_{IN} = 2.5$ V		8	pF

[1] Limits based on characterization results; not production tested.

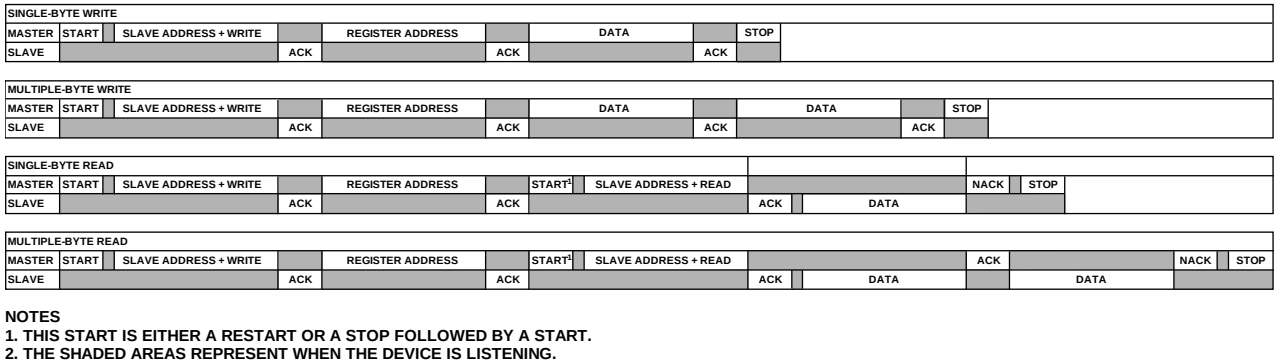


Figure 41. I²C Device Addressing

Table 12. I²C Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

	Limit[1, 2]			
Parameter	**Min**	**Max**	**Unit**	**Description**
f_{SCL}		400	kHz	SCL clock frequency
t_1	2.5		µs	SCL cycle time
t_2	0.6		µs	t_{HIGH}, SCL high time
t_3	1.3		µs	t_{LOW}, SCL low time
t_4	0.6		µs	$t_{HD, STA}$, start/repeated start condition hold time
t_5	100		ns	$t_{SU, DAT}$, data setup time
t_6[3, 4, 5, 6]	0	0.9	µs	$t_{HD, DAT}$, data hold time
t_7	0.6		µs	$t_{SU, STA}$, setup time for repeated start
t_8	0.6		µs	$t_{SU, STO}$, stop condition setup time
t_9	1.3		µs	t_{BUF}, bus-free time between a stop condition and a start condition
t_{10}		300	ns	t_R, rise time of both SCL and SDA when receiving
	0		ns	t_R, rise time of both SCL and SDA when receiving or transmitting
t_{11}		300	ns	t_F, fall time of SDA when receiving
		250	ns	t_F, fall time of both SCL and SDA when transmitting
C_b		400	pF	Capacitive load for each bus line

[1] Limits based on characterization results, with f_{SCL} = 400 kHz and a 3 mA sink current; not production tested.
[2] All values referred to the V_{IH} and the V_{IL} levels given in Table 11.
[3] t_6 is the data hold time that is measured from the falling edge of SCL. It applies to data in transmission and acknowledge.
[4] A transmitting device must internally provide an output hold time of at least 300 ns for the SDA signal (with respect to $V_{IH(min)}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL.
[5] The maximum t_6 value must be met only if the device does not stretch the low period (t_3) of the SCL signal.
[6] The maximum value for t_6 is a function of the clock low time (t_3), the clock rise time (t_{10}), and the minimum data setup time ($t_{5(min)}$). This value is calculated as $t_{6(max)} = t_3 - t_{10} - t_{5(min)}$.

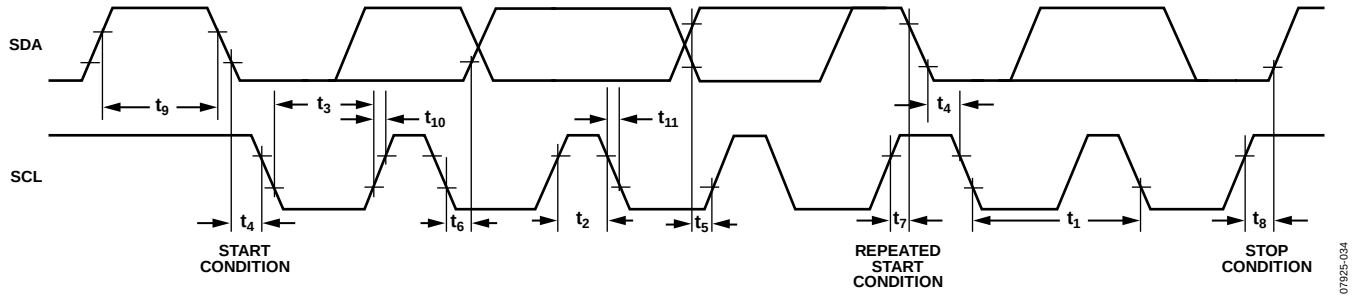


Figure 42. I²C Timing Diagram

INTERRUPTS

The ADXL345 provides two output pins for driving interrupts: INT1 and INT2. Both interrupt pins are push-pull, low impedance pins with output specifications shown in Table 13. The default configuration of the interrupt pins is active high. This can be changed to active low by setting the INT_INVERT bit in the DATA_FORMAT (Address 0x31) register. All functions can be used simultaneously, with the only limiting feature being that some functions may need to share interrupt pins.

Interrupts are enabled by setting the appropriate bit in the INT_ENABLE register (Address 0x2E) and are mapped to either the INT1 pin or the INT2 pin based on the contents of the INT_MAP register (Address 0x2F). When initially configuring the interrupt pins, it is recommended that the functions and interrupt mapping be done before enabling the interrupts. When changing the configuration of an interrupt, it is recommended that the interrupt be disabled first, by clearing the bit corresponding to that function in the INT_ENABLE register, and then the function be reconfigured before enabling the interrupt again. Configuration of the functions while the interrupts are disabled helps to prevent the accidental generation of an interrupt before desired.

The interrupt functions are latched and cleared by either reading the data registers (Address 0x32 to Address 0x37) until the interrupt condition is no longer valid for the data-related interrupts or by reading the INT_SOURCE register (Address 0x30) for the remaining interrupts. This section describes the interrupts that can be set in the INT_ENABLE register and monitored in the INT_SOURCE register.

DATA_READY

The DATA_READY bit is set when new data is available and is cleared when no new data is available.

SINGLE_TAP

The SINGLE_TAP bit is set when a single acceleration event that is greater than the value in the THRESH_TAP register (Address 0x1D) occurs for less time than is specified in the DUR register (Address 0x21).

DOUBLE_TAP

The DOUBLE_TAP bit is set when two acceleration events that are greater than the value in the THRESH_TAP register (Address 0x1D) occur for less time than is specified in the DUR register (Address 0x21), with the second tap starting after the time specified by the latent register (Address 0x22) but within the time specified in the window register (Address 0x23). See the Tap Detection section for more details.

Activity

The activity bit is set when acceleration greater than the value stored in the THRESH_ACT register (Address 0x24) is experienced on any participating axis, set by the ACT_INACT_CTL register (Address 0x27).

Inactivity

The inactivity bit is set when acceleration of less than the value stored in the THRESH_INACT register (Address 0x25) is experienced for more time than is specified in the TIME_INACT register (Address 0x26) on all participating axes, as set by the ACT_INACT_CTL register (Address 0x27). The maximum value for TIME_INACT is 255 sec.

FREE_FALL

The FREE_FALL bit is set when acceleration of less than the value stored in the THRESH_FF register (Address 0x28) is experienced for more time than is specified in the TIME_FF register (Address 0x29) on all axes (logical AND). The FREE_FALL interrupt differs from the inactivity interrupt as follows: all axes always participate and are logically AND'ed, the timer period is much smaller (1.28 sec maximum), and the mode of operation is always dc-coupled.

Watermark

The watermark bit is set when the number of samples in FIFO equals the value stored in the samples bits (Register FIFO_CTL, Address 0x38). The watermark bit is cleared automatically when FIFO is read, and the content returns to a value below the value stored in the samples bits.

Table 13. Interrupt Pin Digital Output

		Limit[1]		
Parameter	**Test Conditions**	**Min**	**Max**	**Unit**
Digital Output				
Low Level Output Voltage (V_{OL})	I_{OL} = 300 µA		$0.2 \times V_{DD\ I/O}$	V
High Level Output Voltage (V_{OH})	I_{OH} = −150 µA	$0.8 \times V_{DD\ I/O}$		V
Low Level Output Current (I_{OL})	$V_{OL} = V_{OL,\ max}$	300		µA
High Level Output Current (I_{OH})	$V_{OH} = V_{OH,\ min}$		−150	µA
Pin Capacitance	f_{IN} = 1 MHz, V_{IN} = 2.5 V		8	pF
Rise/Fall Time				
Rise Time (t_R)[2]	C_{LOAD} = 150 pF		210	ns
Fall Time (t_F)[3]	C_{LOAD} = 150 pF		150	ns

[1] Limits based on characterization results, not production tested.
[2] Rise time is measured as the transition time from $V_{OL,\ max}$ to $V_{OH,\ min}$ of the interrupt pin.
[3] Fall time is measured as the transition time from $V_{OH,\ min}$ to $V_{OL,\ max}$ of the interrupt pin.

Overrun

The overrun bit is set when new data replaces unread data. The precise operation of the overrun function depends on the FIFO mode. In bypass mode, the overrun bit is set when new data replaces unread data in the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). In all other modes, the overrun bit is set when FIFO is filled. The overrun bit is automatically cleared when the contents of FIFO are read.

FIFO

The ADXL345 contains patent pending technology for an embedded memory management system with 32-level FIFO that can be used to minimize host processor burden. This buffer has four modes: bypass, FIFO, stream, and trigger (see FIFO Modes). Each mode is selected by the settings of the FIFO_MODE bits (Bits[D7:D6]) in the FIFO_CTL register (Address 0x38).

Bypass Mode

In bypass mode, FIFO is not operational and, therefore, remains empty.

FIFO Mode

In FIFO mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples until it is full (32 samples from measurements of the x-, y-, and z-axes) and then stops collecting data. After FIFO stops collecting data, the device continues to operate; therefore, features such as tap detection can be used after FIFO is full. The watermark interrupt continues to occur until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Stream Mode

In stream mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples and holds the latest 32 samples from measurements of the x-, y-, and z-axes, discarding older data as new data arrives. The watermark interrupt continues occurring until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Trigger Mode

In trigger mode, FIFO accumulates samples, holding the latest 32 samples from measurements of the x-, y-, and z-axes. After a trigger event occurs and an interrupt is sent to the INT1 or INT2 pin (determined by the trigger bit in the FIFO_CTL register), FIFO keeps the last n samples (where n is the value specified by the samples bits in the FIFO_CTL register) and then operates in FIFO mode, collecting new samples only when FIFO is not full. A delay of at least 5 μs should be present between the trigger event occurring and the start of reading data from the FIFO to allow the FIFO to discard and retain the necessary samples. Additional trigger events cannot be recognized until the trigger mode is reset. To reset the trigger mode, set the device to bypass mode and then set the device back to trigger mode. Note that the FIFO data should be read first because placing the device into bypass mode clears FIFO.

Retrieving Data from FIFO

The FIFO data is read through the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). When the FIFO is in FIFO, stream, or trigger mode, reads to the DATAX, DATAY, and DATAZ registers read data stored in the FIFO. Each time data is read from the FIFO, the oldest x-, y-, and z-axes data are placed into the DATAX, DATAY and DATAZ registers.

If a single-byte read operation is performed, the remaining bytes of data for the current FIFO sample are lost. Therefore, all axes of interest should be read in a burst (or multiple-byte) read operation. To ensure that the FIFO has completely popped (that is, that new data has completely moved into the DATAX, DATAY, and DATAZ registers), there must be at least 5 μs between the end of reading the data registers and the start of a new read of the FIFO or a read of the FIFO_STATUS register (Address 0x39). The end of reading a data register is signified by the transition from Register 0x37 to Register 0x38 or by the $\overline{CS}$ pin going high.

For SPI operation at 1.6 MHz or less, the register addressing portion of the transmission is a sufficient delay to ensure that the FIFO has completely popped. For SPI operation greater than 1.6 MHz, it is necessary to deassert the $\overline{CS}$ pin to ensure a total delay of 5 μs; otherwise, the delay is not sufficient. The total delay necessary for 5 MHz operation is at most 3.4 μs. This is not a concern when using I^2C mode because the communication rate is low enough to ensure a sufficient delay between FIFO reads.

SELF-TEST

The ADXL345 incorporates a self-test feature that effectively tests its mechanical and electronic systems simultaneously. When the self-test function is enabled (via the SELF_TEST bit in the DATA_FORMAT register, Address 0x31), an electrostatic force is exerted on the mechanical sensor. This electrostatic force moves the mechanical sensing element in the same manner as acceleration, and it is additive to the acceleration experienced by the device. This added electrostatic force results in an output change in the x-, y-, and z-axes. Because the electrostatic force is proportional to V_S^2, the output change varies with V_S. This effect is shown in Figure 43. The scale factors shown in Table 14 can be used to adjust the expected self-test output limits for different supply voltages, V_S. The self-test feature of the ADXL345 also exhibits a bimodal behavior. However, the limits shown in Table 1 and Table 15 to Table 18 are valid for both potential self-test values due to bimodality. Use of the self-test feature at data rates less than 100 Hz or at 1600 Hz may yield values outside these limits. Therefore, the part must be in normal power operation (LOW_POWER bit = 0 in BW_RATE register, Address 0x2C) and be placed into a data rate of 100 Hz through 800 Hz or 3200 Hz for the self-test function to operate correctly.

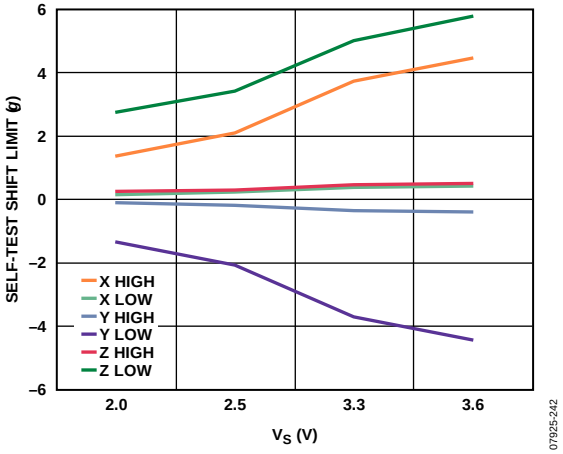


Figure 43. Self-Test Output Change Limits vs. Supply Voltage

Table 14. Self-Test Output Scale Factors for Different Supply Voltages, V_S

Supply Voltage, V_S (V)	X-Axis, Y-Axis	Z-Axis
2.00	0.64	0.8
2.50	1.00	1.00
3.30	1.77	1.47
3.60	2.11	1.69

Table 15. Self-Test Output in LSB for ±2 *g*, 10-Bit or Full Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	50	540	LSB
Y	−540	−50	LSB
Z	75	875	LSB

Table 16. Self-Test Output in LSB for ±4 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	25	270	LSB
Y	−270	−25	LSB
Z	38	438	LSB

Table 17. Self-Test Output in LSB for ±8 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	12	135	LSB
Y	−135	−12	LSB
Z	19	219	LSB

Table 18. Self-Test Output in LSB for ±16 *g*, 10-Bit Resolution (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Axis	Min	Max	Unit
X	6	67	LSB
Y	−67	−6	LSB
Z	10	110	LSB

REGISTER MAP

Table 19.

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

REGISTER DEFINITIONS

Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5 (345 octal).

Register 0x1D—THRESH_TAP (Read/Write)

The THRESH_TAP register is eight bits and holds the threshold value for tap interrupts. The data format is unsigned, therefore, the magnitude of the tap event is compared with the value in THRESH_TAP for normal tap detection. The scale factor is 62.5 mg/LSB (that is, 0xFF = 16 *g*). A value of 0 may result in undesirable behavior if single tap/double tap interrupts are enabled.

Register 0x1E, Register 0x1F, Register 0x20—OFSX, OFSY, OFSZ (Read/Write)

The OFSX, OFSY, and OFSZ registers are each eight bits and offer user-set offset adjustments in twos complement format with a scale factor of 15.6 mg/LSB (that is, 0x7F = 2 *g*). The value stored in the offset registers is automatically added to the acceleration data, and the resulting value is stored in the output data registers. For additional information regarding offset calibration and the use of the offset registers, refer to the Offset Calibration section.

Register 0x21—DUR (Read/Write)

The DUR register is eight bits and contains an unsigned time value representing the maximum time that an event must be above the THRESH_TAP threshold to qualify as a tap event. The scale factor is 625 µs/LSB. A value of 0 disables the single tap/double tap functions.

Register 0x22—Latent (Read/Write)

The latent register is eight bits and contains an unsigned time value representing the wait time from the detection of a tap event to the start of the time window (defined by the window register) during which a possible second tap event can be detected. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x23—Window (Read/Write)

The window register is eight bits and contains an unsigned time value representing the amount of time after the expiration of the latency time (determined by the latent register) during which a second valid tap can begin. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x24—THRESH_ACT (Read/Write)

The THRESH_ACT register is eight bits and holds the threshold value for detecting activity. The data format is unsigned, so the magnitude of the activity event is compared with the value in the THRESH_ACT register. The scale factor is 62.5 mg/LSB. A value of 0 may result in undesirable behavior if the activity interrupt is enabled.

Register 0x25—THRESH_INACT (Read/Write)

The THRESH_INACT register is eight bits and holds the threshold value for detecting inactivity. The data format is unsigned, so the magnitude of the inactivity event is compared with the value in the THRESH_INACT register. The scale factor is 62.5 mg/LSB. A value of 0 may result in undesirable behavior if the inactivity interrupt is enabled.

Register 0x26—TIME_INACT (Read/Write)

The TIME_INACT register is eight bits and contains an unsigned time value representing the amount of time that acceleration must be less than the value in the THRESH_INACT register for inactivity to be declared. The scale factor is 1 sec/LSB. Unlike the other interrupt functions, which use unfiltered data (see the Threshold section), the inactivity function uses filtered output data. At least one output sample must be generated for the inactivity interrupt to be triggered. This results in the function appearing unresponsive if the TIME_INACT register is set to a value less than the time constant of the output data rate. A value of 0 results in an interrupt when the output data is less than the value in the THRESH_INACT register.

Register 0x27—ACT_INACT_CTL (Read/Write)

D7	D6	D5	D4
ACT ac/dc	ACT_X enable	ACT_Y enable	ACT_Z enable
D3	**D2**	**D1**	**D0**
INACT ac/dc	INACT_X enable	INACT_Y enable	INACT_Z enable

ACT AC/DC and INACT AC/DC Bits

A setting of 0 selects dc-coupled operation, and a setting of 1 enables ac-coupled operation. In dc-coupled operation, the current acceleration magnitude is compared directly with THRESH_ACT and THRESH_INACT to determine whether activity or inactivity is detected.

In ac-coupled operation for activity detection, the acceleration value at the start of activity detection is taken as a reference value. New samples of acceleration are then compared to this reference value, and if the magnitude of the difference exceeds the THRESH_ACT value, the device triggers an activity interrupt.

Similarly, in ac-coupled operation for inactivity detection, a reference value is used for comparison and is updated whenever the device exceeds the inactivity threshold. After the reference value is selected, the device compares the magnitude of the difference between the reference value and the current acceleration with THRESH_INACT. If the difference is less than the value in THRESH_INACT for the time in TIME_INACT, the device is considered inactive and the inactivity interrupt is triggered.

ACT_x Enable Bits and INACT_x Enable Bits

A setting of 1 enables x-, y-, or z-axis participation in detecting activity or inactivity. A setting of 0 excludes the selected axis from participation. If all axes are excluded, the function is disabled. For activity detection, all participating axes are logically OR'ed, causing the activity function to trigger when any of the participating axes exceeds the threshold. For inactivity detection, all participating axes are logically AND'ed, causing the inactivity function to trigger only if all participating axes are below the threshold for the specified time.

Register 0x28—THRESH_FF (Read/Write)

The THRESH_FF register is eight bits and holds the threshold value, in unsigned format, for free-fall detection. The acceleration on all axes is compared with the value in THRESH_FF to determine if a free-fall event occurred. The scale factor is 62.5 m*g*/LSB. Note that a value of 0 m*g* may result in undesirable behavior if the free-fall interrupt is enabled. Values between 300 m*g* and 600 m*g* (0x05 to 0x09) are recommended.

Register 0x29—TIME_FF (Read/Write)

The TIME_FF register is eight bits and stores an unsigned time value representing the minimum time that the value of all axes must be less than THRESH_FF to generate a free-fall interrupt. The scale factor is 5 ms/LSB. A value of 0 may result in undesirable behavior if the free-fall interrupt is enabled. Values between 100 ms and 350 ms (0x14 to 0x46) are recommended.

Register 0x2A—TAP_AXES (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	Suppress	TAP_X enable	TAP_Y enable	TAP_Z enable

Suppress Bit

Setting the suppress bit suppresses double tap detection if acceleration greater than the value in THRESH_TAP is present between taps. See the Tap Detection section for more details.

TAP_x Enable Bits

A setting of 1 in the TAP_X enable, TAP_Y enable, or TAP_Z enable bit enables x-, y-, or z-axis participation in tap detection. A setting of 0 excludes the selected axis from participation in tap detection.

Register 0x2B—ACT_TAP_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
0	ACT_X source	ACT_Y source	ACT_Z source	Asleep	TAP_X source	TAP_Y source	TAP_Z source

ACT_x Source and TAP_x Source Bits

These bits indicate the first axis involved in a tap or activity event. A setting of 1 corresponds to involvement in the event, and a setting of 0 corresponds to no involvement. When new data is available, these bits are not cleared but are overwritten by the new data. The ACT_TAP_STATUS register should be read before clearing the interrupt. Disabling an axis from participation clears the corresponding source bit when the next activity or single tap/double tap event occurs.

Asleep Bit

A setting of 1 in the asleep bit indicates that the part is asleep, and a setting of 0 indicates that the part is not asleep. This bit toggles only if the device is configured for auto sleep. See the AUTO_SLEEP Bit section for more information on autosleep mode.

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

LOW_POWER Bit

A setting of 0 in the LOW_POWER bit selects normal operation, and a setting of 1 selects reduced power operation, which has somewhat higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 7 and Table 8 for details). The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Link Bit

A setting of 1 in the link bit with both the activity and inactivity functions enabled delays the start of the activity function until inactivity is detected. After activity is detected, inactivity detection begins, preventing the detection of activity. This bit serially links the activity and inactivity functions. When this bit is set to 0, the inactivity and activity functions are concurrent. Additional information can be found in the Link Mode section.

When clearing the link bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the link bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

AUTO_SLEEP Bit

If the link bit is set, a setting of 1 in the AUTO_SLEEP bit enables the auto-sleep functionality. In this mode, the ADXL345 automatically switches to sleep mode if the inactivity function is enabled and inactivity is detected (that is, when acceleration is below the THRESH_INACT value for at least the time indicated by TIME_INACT). If activity is also enabled, the ADXL345 automatically wakes up from sleep after detecting activity and returns to operation at the output data rate set in the BW_RATE register. A setting of 0 in the AUTO_SLEEP bit disables automatic switching to sleep mode. See the description of the Sleep Bit in this section for more information on sleep mode.

If the link bit is not set, the AUTO_SLEEP feature is disabled and setting the AUTO_SLEEP bit does not have an impact on device operation. Refer to the Link Bit section or the Link Mode section for more information on utilization of the link feature.

When clearing the AUTO_SLEEP bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the AUTO_SLEEP bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

Sleep Bit

A setting of 0 in the sleep bit puts the part into the normal mode of operation, and a setting of 1 places the part into sleep mode. Sleep mode suppresses DATA_READY, stops transmission of data to FIFO, and switches the sampling rate to one specified by the wakeup bits. In sleep mode, only the activity function can be used. When the DATA_READY interrupt is suppressed, the output data registers (Register 0x32 to Register 0x37) are still updated at the sampling rate set by the wakeup bits (D1:D0).

When clearing the sleep bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the sleep bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Wakeup Bits

These bits control the frequency of readings in sleep mode as described in Table 20.

Table 20. Frequency of Readings in Sleep Mode

Setting		
D1	**D0**	**Frequency (Hz)**
0	0	8
0	1	4
1	0	2
1	1	1

Register 0x2E—INT_ENABLE (Read/Write)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Setting bits in this register to a value of 1 enables their respective functions to generate interrupts, whereas a value of 0 prevents the functions from generating interrupts. The DATA_READY, watermark, and overrun bits enable only the interrupt output; the functions are always enabled. It is recommended that interrupts be configured before enabling their outputs.

Register 0x2F—INT_MAP (R/$\overline{W}$)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Any bits set to 0 in this register send their respective interrupts to the INT1 pin, whereas bits set to 1 send their respective interrupts to the INT2 pin. All selected interrupts for a given pin are OR'ed.

Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Register 0x31—DATA_FORMAT (Read/Write)

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

INT_INVERT Bit

A value of 0 in the INT_INVERT bit sets the interrupts to active high, and a value of 1 sets the interrupts to active low.

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor.

Justify Bit

A setting of 1 in the justify bit selects left-justified (MSB) mode, and a setting of 0 selects right-justified mode with sign extension.

Range Bits

These bits set the *g* range as described in Table 21.

Table 21. *g* Range Setting

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)

These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

Register 0x38—FIFO_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_MODE		Trigger	Samples				

FIFO_MODE Bits

These bits set the FIFO mode, as described in Table 22.

Table 22. FIFO Modes

Setting			
D7	**D6**	**Mode**	**Function**
0	0	Bypass	FIFO is bypassed.
0	1	FIFO	FIFO collects up to 32 values and then stops collecting data, collecting new data only when FIFO is not full.
1	0	Stream	FIFO holds the last 32 data values. When FIFO is full, the oldest data is overwritten with newer data.
1	1	Trigger	When triggered by the trigger bit, FIFO holds the last data samples before the trigger event and then continues to collect data until full. New data is collected only when FIFO is not full.

Trigger Bit

A value of 0 in the trigger bit links the trigger event of trigger mode to INT1, and a value of 1 links the trigger event to INT2.

Samples Bits

The function of these bits depends on the FIFO mode selected (see Table 23). Entering a value of 0 in the samples bits immediately sets the watermark status bit in the INT_SOURCE register, regardless of which FIFO mode is selected. Undesirable operation may occur if a value of 0 is used for the samples bits when trigger mode is used.

Table 23. Samples Bits Functions

FIFO Mode	Samples Bits Function
Bypass	None.
FIFO	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Stream	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Trigger	Specifies how many FIFO samples are retained in the FIFO buffer before a trigger event.

0x39—FIFO_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_TRIG	0	Entries					

FIFO_TRIG Bit

A 1 in the FIFO_TRIG bit corresponds to a trigger event occurring, and a 0 means that a FIFO trigger event has not occurred.

Entries Bits

These bits report how many data values are stored in FIFO. Access to collect the data from FIFO is provided through the DATAX, DATAY, and DATAZ registers. FIFO reads must be done in burst or multiple-byte mode because each FIFO level is cleared after any read (single- or multiple-byte) of FIFO. FIFO stores a maximum of 32 entries, which equates to a maximum of 33 entries available at any given time because an additional entry is available at the output filter of the device.

APPLICATIONS INFORMATION

POWER SUPPLY DECOUPLING

A 1 µF tantalum capacitor (C_S) at V_S and a 0.1 µF ceramic capacitor ($C_{I/O}$) at $V_{DD\ I/O}$ placed close to the ADXL345 supply pins is recommended to adequately decouple the accelerometer from noise on the power supply. If additional decoupling is necessary, a resistor or ferrite bead, no larger than 100 Ω, in series with V_S may be helpful. Additionally, increasing the bypass capacitance on V_S to a 10 µF tantalum capacitor in parallel with a 0.1 µF ceramic capacitor may also improve noise.

Care should be taken to ensure that the connection from the ADXL345 ground to the power supply ground has low impedance because noise transmitted through ground has an effect similar to noise transmitted through V_S. It is recommended that V_S and $V_{DD\ I/O}$ be separate supplies to minimize digital clocking noise on the V_S supply. If this is not possible, additional filtering of the supplies, as previously mentioned, may be necessary.

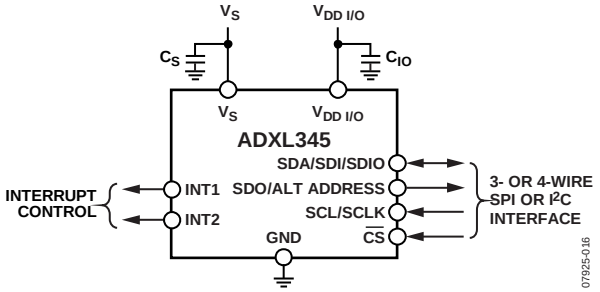


Figure 44. Application Diagram

MECHANICAL CONSIDERATIONS FOR MOUNTING

The ADXL345 should be mounted on the PCB in a location close to a hard mounting point of the PCB to the case. Mounting the ADXL345 at an unsupported PCB location, as shown in Figure 45, may result in large, apparent measurement errors due to undampened PCB vibration. Locating the accelerometer near a hard mounting point ensures that any PCB vibration at the accelerometer is above the accelerometer's mechanical sensor resonant frequency and, therefore, effectively invisible to the accelerometer. Multiple mounting points, close to the sensor, and/or a thicker PCB also help to reduce the effect of system resonance on the performance of the sensor.

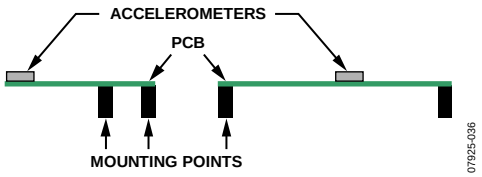


Figure 45. Incorrectly Placed Accelerometers

TAP DETECTION

The tap interrupt function is capable of detecting either single or double taps. The following parameters are shown in Figure 46 for a valid single and valid double tap event:

- The tap detection threshold is defined by the THRESH_TAP register (Address 0x1D).
- The maximum tap duration time is defined by the DUR register (Address 0x21).
- The tap latency time is defined by the latent register (Address 0x22) and is the waiting period from the end of the first tap until the start of the time window, when a second tap can be detected, which is determined by the value in the window register (Address 0x23).
- The interval after the latency time (set by the latent register) is defined by the window register. Although a second tap must begin after the latency time has expired, it need not finish before the end of the time defined by the window register.

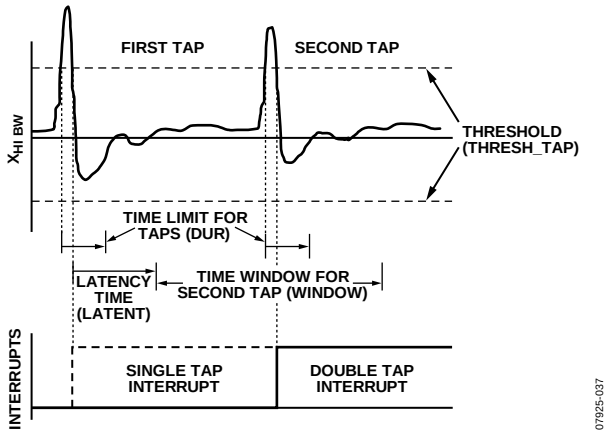


Figure 46. Tap Interrupt Function with Valid Single and Double Taps

If only the single tap function is in use, the single tap interrupt is triggered when the acceleration goes below the threshold, as long as DUR has not been exceeded. If both single and double tap functions are in use, the single tap interrupt is triggered when the double tap event has been either validated or invalidated.

Several events can occur to invalidate the second tap of a double tap event. First, if the suppress bit in the TAP_AXES register (Address 0x2A) is set, any acceleration spike above the threshold during the latency time (set by the latent register) invalidates the double tap detection, as shown in Figure 47.

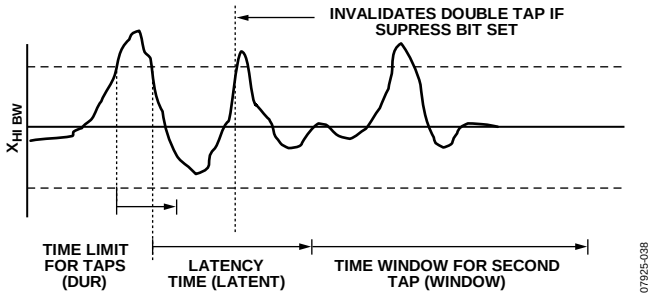


Figure 47. Double Tap Event Invalid Due to High g Event When the Suppress Bit Is Set

A double tap event can also be invalidated if acceleration above the threshold is detected at the start of the time window for the second tap (set by the window register). This results in an invalid double tap at the start of this window, as shown in Figure 48. Additionally, a double tap event can be invalidated if an acceleration exceeds the time limit for taps (set by the DUR register), resulting in an invalid double tap at the end of the DUR time limit for the second tap event, also shown in Figure 48.

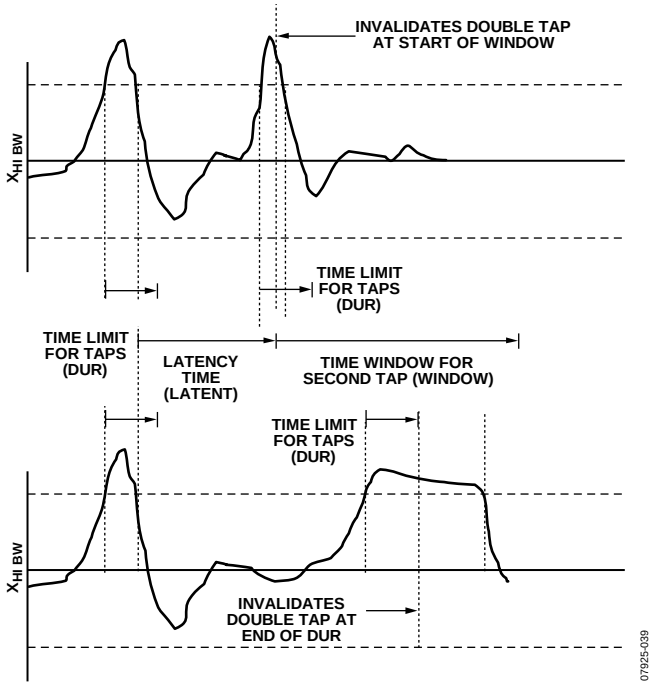


Figure 48. Tap Interrupt Function with Invalid Double Taps

Single taps, double taps, or both can be detected by setting the respective bits in the INT_ENABLE register (Address 0x2E). Control over participation of each of the three axes in single tap/double tap detection is exerted by setting the appropriate bits in the TAP_AXES register (Address 0x2A). For the double tap function to operate, both the latent and window registers must be set to a nonzero value.

Every mechanical system has somewhat different single tap/double tap responses based on the mechanical characteristics of the system. Therefore, some experimentation with values for the DUR, latent, window, and THRESH_TAP registers is required. In general, a good starting point is to set the DUR register to a value greater than 0x10 (10 ms), the latent register to a value greater than 0x10 (20 ms), the window register to a value greater than 0x40 (80 ms), and the THRESH_TAP register to a value greater than 0x30 (3 *g*). Setting a very low value in the latent, window, or THRESH_TAP register may result in an unpredictable response due to the accelerometer picking up echoes of the tap inputs.

After a tap interrupt has been received, the first axis to exceed the THRESH_TAP level is reported in the ACT_TAP_STATUS register (Address 0x2B). This register is never cleared but is overwritten with new data.

THRESHOLD

The lower output data rates are achieved by decimating a common sampling frequency inside the device. The activity, free-fall, and single tap/double tap detection functions without improved tap enabled are performed using undecimated data. Because the bandwidth of the output data varies with the data rate and is lower than the bandwidth of the undecimated data, the high frequency and high *g* data that is used to determine activity, free-fall, and single tap/double tap events may not be present if the output of the accelerometer is examined. This may result in functions triggering when acceleration data does not appear to meet the conditions set by the user for the corresponding function.

LINK MODE

The function of the link bit is to reduce the number of activity interrupts that the processor must service by setting the device to look for activity only after inactivity. For proper operation of this feature, the processor must still respond to the activity and inactivity interrupts by reading the INT_SOURCE register (Address 0x30) and, therefore, clearing the interrupts. If an activity interrupt is not cleared, the part cannot go into autosleep mode. The asleep bit in the ACT_TAP_STATUS register (Address 0x2B) indicates if the part is asleep.

SLEEP MODE VS. LOW POWER MODE

In applications where a low data rate and low power consumption is desired (at the expense of noise performance), it is recommended that low power mode be used. The use of low power mode preserves the functionality of the DATA_READY interrupt and the FIFO for postprocessing of the acceleration data. Sleep mode, while offering a low data rate and power consumption, is not intended for data acquisition.

However, when sleep mode is used in conjunction with the AUTO_SLEEP mode and the link mode, the part can automatically switch to a low power, low sampling rate mode when inactivity is detected. To prevent the generation of redundant inactivity interrupts, the inactivity interrupt is automatically disabled and activity is enabled. When the ADXL345 is in sleep mode, the host processor can also be placed into sleep mode or low power mode to save significant system power. When activity is detected, the accelerometer automatically switches back to the original data rate of the application and provides an activity interrupt that can be used to wake up the host processor. Similar to when inactivity occurs, detection of activity events is disabled and inactivity is enabled.

OFFSET CALIBRATION

Accelerometers are mechanical structures containing elements that are free to move. These moving parts can be very sensitive to mechanical stresses, much more so than solid-state electronics. The 0 *g* bias or offset is an important accelerometer metric because it defines the baseline for measuring acceleration. Additional stresses can be applied during assembly of a system containing an accelerometer. These stresses can come from, but are not limited to, component soldering, board stress during mounting, and application of any compounds on or over the component. If calibration is deemed necessary, it is recommended that calibration be performed after system assembly to compensate for these effects.

A simple method of calibration is to measure the offset while assuming that the sensitivity of the ADXL345 is as specified in Table 1. The offset can then be automatically accounted for by using the built-in offset registers. This results in the data acquired from the DATA registers already compensating for any offset.

In a no-turn or single-point calibration scheme, the part is oriented such that one axis, typically the z-axis, is in the 1 *g* field of gravity and the remaining axes, typically the x- and y-axis, are in a 0 *g* field. The output is then measured by taking the average of a series of samples. The number of samples averaged is a choice of the system designer, but a recommended starting point is 0.1 sec worth of data for data rates of 100 Hz or greater. This corresponds to 10 samples at the 100 Hz data rate. For data rates less than 100 Hz, it is recommended that at least 10 samples be averaged together. These values are stored as X_{0g}, Y_{0g}, and Z_{+1g} for the 0 *g* measurements on the x- and y-axis and the 1 *g* measurement on the z-axis, respectively.

The values measured for X_{0g} and Y_{0g} correspond to the x- and y-axis offset, and compensation is done by subtracting those values from the output of the accelerometer to obtain the actual acceleration:

$$X_{ACTUAL} = X_{MEAS} - X_{0g}$$

$$Y_{ACTUAL} = Y_{MEAS} - Y_{0g}$$

Because the z-axis measurement was done in a +1 *g* field, a no-turn or single-point calibration scheme assumes an ideal sensitivity, S_Z for the z-axis. This is subtracted from Z_{+1g} to attain the z-axis offset, which is then subtracted from future measured values to obtain the actual value:

$$Z_{0g} = Z_{+1g} - S_Z$$

$$Z_{ACTUAL} = Z_{MEAS} - Z_{0g}$$

The ADXL345 can automatically compensate the output for offset by using the offset registers (Register 0x1E, Register 0x1F, and Register 0x20). These registers contain an 8-bit, twos complement value that is automatically added to all measured acceleration values, and the result is then placed into the DATA registers. Because the value placed in an offset register is additive, a negative value is placed into the register to eliminate a positive offset and vice versa for a negative offset. The register has a scale factor of 15.6 m*g*/LSB and is independent of the selected *g*-range.

As an example, assume that the ADXL345 is placed into full-resolution mode with a sensitivity of typically 256 LSB/*g*. The part is oriented such that the z-axis is in the field of gravity and x-, y-, and z-axis outputs are measured as +10 LSB, −13 LSB, and +9 LSB, respectively. Using the previous equations, X_{0g} is +10 LSB, Y_{0g} is −13 LSB, and Z_{0g} is +9 LSB. Each LSB of output in full-resolution is 3.9 m*g* or one-quarter of an LSB of the offset register. Because the offset register is additive, the 0 *g* values are negated and rounded to the nearest LSB of the offset register:

$$X_{OFFSET} = -Round(10/4) = -3 \text{ LSB}$$

$$Y_{OFFSET} = -Round(-13/4) = 3 \text{ LSB}$$

$$Z_{OFFSET} = -Round(9/4) = -2 \text{ LSB}$$

These values are programmed into the OFSX, OFSY, and OFXZ registers, respectively, as 0xFD, 0x03 and 0xFE. As with all registers in the ADXL345, the offset registers do not retain the value written into them when power is removed from the part. Power-cycling the ADXL345 returns the offset registers to their default value of 0x00.

Because the no-turn or single-point calibration method assumes an ideal sensitivity in the z-axis, any error in the sensitivity results in offset error. For instance, if the actual sensitivity was 250 LSB/*g* in the previous example, the offset would be 15 LSB, not 9 LSB. To help minimize this error, an additional measurement point can be used with the z-axis in a 0 *g* field and the 0 *g* measurement can be used in the Z_{ACTUAL} equation.

USING SELF-TEST

The self-test change is defined as the difference between the acceleration output of an axis with self-test enabled and the acceleration output of the same axis with self-test disabled (see Endnote 4 of Table 1). This definition assumes that the sensor does not move between these two measurements, because if the sensor moves, a non–self-test related shift corrupts the test.

Proper configuration of the ADXL345 is also necessary for an accurate self-test measurement. The part should be set with a data rate of 100 Hz through 800 Hz, or 3200 Hz. This is done by ensuring that a value of 0x0A through 0x0D, or 0x0F is written into the rate bits (Bit D3 through Bit D0) in the BW_RATE register (Address 0x2C). The part also must be placed into normal power operation by ensuring the LOW_POWER bit in the BW_RATE register is cleared (LOW_POWER bit = 0) for accurate self-test measurements. It is recommended that the part be set to full-resolution, 16 *g* mode to ensure that there is sufficient dynamic range for the entire self-test shift. This is done by setting Bit D3 of the DATA_FORMAT register (Address 0x31) and writing a value of 0x03 to the range bits (Bit D1 and Bit D0) of the DATA_FORMAT register (Address 0x31). This results in a high dynamic range for measurement and a 3.9 m*g*/LSB scale factor.

After the part is configured for accurate self-test measurement, several samples of x-, y-, and z-axis acceleration data should be retrieved from the sensor and averaged together. The number of samples averaged is a choice of the system designer, but a recommended starting point is 0.1 sec worth of data for data rates of 100 Hz or greater. This corresponds to 10 samples at the 100 Hz data rate. For data rates less than 100 Hz, it is recommended that at least 10 samples be averaged together. The averaged values should be stored and labeled appropriately as the self-test disabled data, that is, X_{ST_OFF}, Y_{ST_OFF}, and Z_{ST_OFF}.

Next, self-test should be enabled by setting Bit D7 (SELF_TEST) of the DATA_FORMAT register (Address 0x31). The output needs some time (about four samples) to settle after enabling self-test. After allowing the output to settle, several samples of the x-, y-, and z-axis acceleration data should be taken again and averaged. It is recommended that the same number of samples be taken for this average as was previously taken. These averaged values should again be stored and labeled appropriately as the value with self-test enabled, that is, X_{ST_ON}, Y_{ST_ON}, and Z_{ST_ON}. Self-test can then be disabled by clearing Bit D7 (SELF_TEST) of the DATA_FORMAT register (Address 0x31).

With the stored values for self-test enabled and disabled, the self-test change is as follows:

$$X_{ST} = X_{ST\_ON} - X_{ST\_OFF}$$

$$Y_{ST} = Y_{ST\_ON} - Y_{ST\_OFF}$$

$$Z_{ST} = Z_{ST\_ON} - Z_{ST\_OFF}$$

Because the measured output for each axis is expressed in LSBs, X_{ST}, Y_{ST}, and Z_{ST} are also expressed in LSBs. These values can be converted to *g*'s of acceleration by multiplying each value by the 3.9 m*g*/LSB scale factor, if configured for full-resolution mode. Additionally, Table 15 through Table 18 correspond to the self-test range converted to LSBs and can be compared with the measured self-test change when operating at a V_S of 2.5 V. For other voltages, the minimum and maximum self-test output values should be adjusted based on (multiplied by) the scale factors shown in Table 14. If the part was placed into ±2 *g*, 10-bit or full-resolution mode, the values listed in Table 15 should be used. Although the fixed 10-bit mode or a range other than 16 *g* can be used, a different set of values, as indicated in Table 16 through Table 18, would need to be used. Using a range below 8 *g* may result in insufficient dynamic range and should be considered when selecting the range of operation for measuring self-test.

If the self-test change is within the valid range, the test is considered successful. Generally, a part is considered to pass if the minimum magnitude of change is achieved. However, a part that changes by more than the maximum magnitude is not necessarily a failure.

Another effective method for using the self-test to verify accelerometer functionality is to toggle the self test at a certain rate and then perform an FFT on the output. The FFT should have a corresponding tone at the frequency the self-test was toggled. Using an FFT like this removes the dependency of the test on supply voltage and on self-test magnitude, which can vary within a rather wide range.

DATA FORMATTING OF UPPER DATA RATES

Formatting of output data at the 3200 Hz and 1600 Hz output data rates changes depending on the mode of operation (full-resolution or fixed 10-bit) and the selected output range.

When using the 3200 Hz or 1600 Hz output data rates in full-resolution or ±2 g, 10-bit operation, the LSB of the output data-word is always 0. When data is right justified, this corresponds to Bit D0 of the DATAx0 register, as shown in Figure 49. When data is left justified and the part is operating in ±2 g, 10-bit mode, the LSB of the output data-word is Bit D6 of the DATAx0 register. In full-resolution operation when data is left justified, the location of the LSB changes according to the selected output range.

For a range of ±2 g, the LSB is Bit D6 of the DATAx0 register; for ±4 g, Bit D5 of the DATAx0 register; for ±8 g, Bit D4 of the DATAx0 register; and for ±16 g, Bit D3 of the DATAx0 register. This is shown in Figure 50.

The use of 3200 Hz and 1600 Hz output data rates for fixed 10-bit operation in the ±4 g, ±8 g, and ±16 g output ranges provides an LSB that is valid and that changes according to the applied acceleration. Therefore, in these modes of operation, Bit D0 is not always 0 when output data is right justified and Bit D6 is not always 0 when output data is left justified. Operation at any data rate of 800 Hz or lower also provides a valid LSB in all ranges and modes that changes according to the applied acceleration.

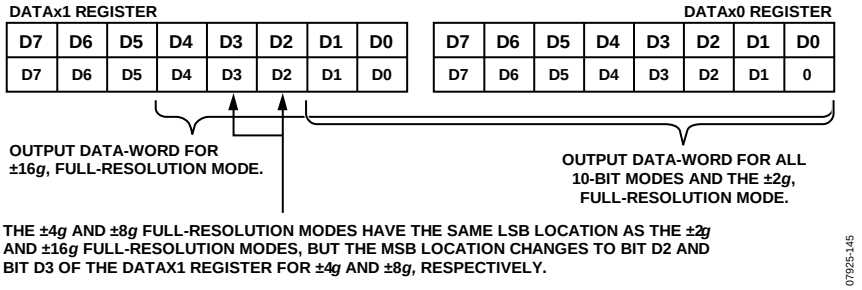


Figure 49. Data Formatting of Full-Resolution and ±2 g, 10-Bit Modes of Operation When Output Data Is Right Justified

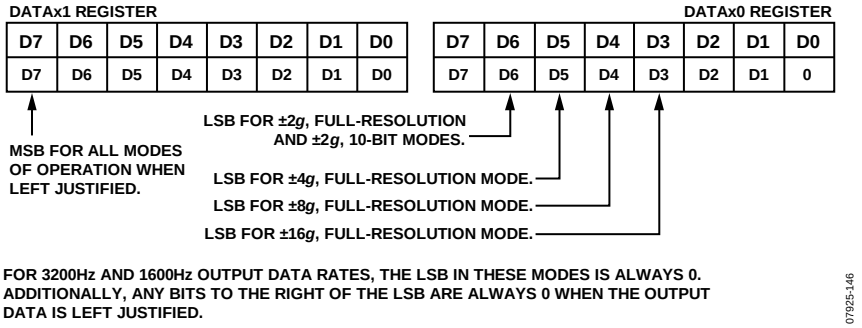


Figure 50. Data Formatting of Full-Resolution and ±2 g, 10-Bit Modes of Operation When Output Data Is Left Justified

NOISE PERFORMANCE

The specification of noise shown in Table 1 corresponds to the typical noise performance of the ADXL345 in normal power operation with an output data rate of 100 Hz (LOW_POWER bit (D4) = 0, rate bits (D3:D0) = 0xA in the BW_RATE register, Address 0x2C). For normal power operation at data rates below 100 Hz, the noise of the ADXL345 is equivalent to the noise at 100 Hz ODR in LSBs. For data rates greater than 100 Hz, the noise increases roughly by a factor of √2 per doubling of the data rate. For example, at 400 Hz ODR, the noise on the x- and y-axes is typically less than 1.5 LSB rms, and the noise on the z-axis is typically less than 2.2 LSB rms.

For low power operation (LOW_POWER bit (D4) = 1 in the BW_RATE register, Address 0x2C), the noise of the ADXL345 is constant for all valid data rates shown in Table 8. This value is typically less than 1.8 LSB rms for the x- and y-axes and typically less than 2.6LSB rms for the z-axis.

The trend of noise performance for both normal power and low power modes of operation of the ADXL345 is shown in Figure 51.

Figure 52 shows the typical Allan deviation for the ADXL345. The 1/f corner of the device, as shown in this figure, is very low, allowing absolute resolution of approximately 100 µg (assuming that there is sufficient integration time). Figure 52 also shows that the noise density is 290 µg/√Hz for the x-axis and y-axis and 430 µg/√Hz for the z-axis.

Figure 53 shows the typical noise performance trend of the ADXL345 over supply voltage. The performance is normalized to the tested and specified supply voltage, V_S = 2.5 V. In general, noise decreases as supply voltage is increased. It should be noted, as shown in Figure 51, that the noise on the z-axis is typically higher than on the x-axis and y-axis; therefore, while they change roughly the same in percentage over supply voltage, the magnitude of change on the z-axis is greater than the magnitude of change on the x-axis and y-axis.

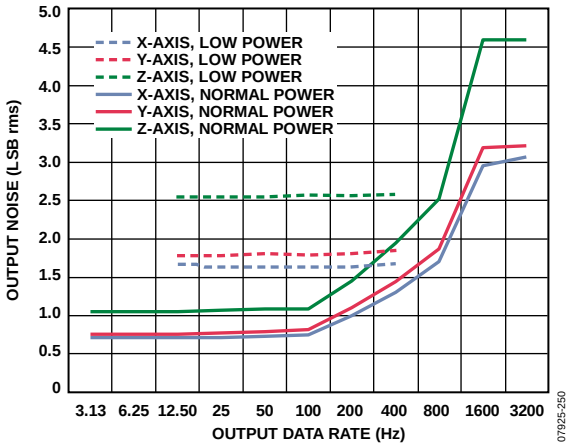


Figure 51. Noise vs. Output Data Rate for Normal and Low Power Modes, Full-Resolution (256 LSB/g)

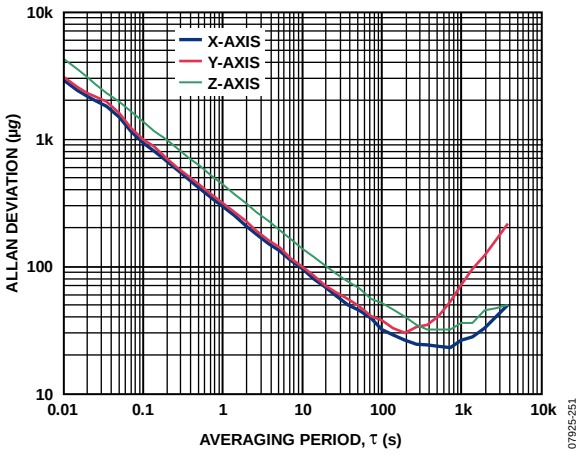


Figure 52. Root Allan Deviation

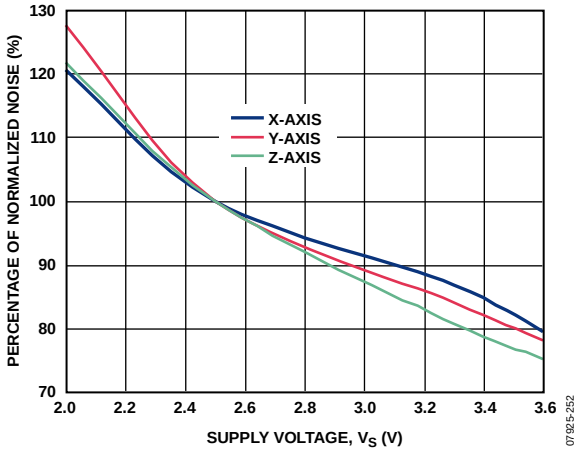


Figure 53. Normalized Noise vs. Supply Voltage, V_S

OPERATION AT VOLTAGES OTHER THAN 2.5 V

The ADXL345 is tested and specified at a supply voltage of V_S = 2.5 V; however, it can be powered with V_S as high as 3.6 V or as low as 2.0 V. Some performance parameters change as the supply voltage changes: offset, sensitivity, noise, self-test, and supply current.

Due to slight changes in the electrostatic forces as supply voltage is varied, the offset and sensitivity change slightly. When operating at a supply voltage of V_S = 3.3 V, the x- and y-axis offset is typically 25 *mg* higher than at Vs = 2.5 V operation. The z-axis is typically 20 *mg* lower when operating at a supply voltage of 3.3 V than when operating at V_S = 2.5 V. Sensitivity on the x- and y-axes typically shifts from a nominal 256 LSB/*g* (full-resolution or ±2 *g*, 10-bit operation) at V_S = 2.5 V operation to 265 LSB/*g* when operating with a supply voltage of 3.3 V. The z-axis sensitivity is unaffected by a change in supply voltage and is the same at V_S = 3.3 V operation as it is at V_S = 2.5 V operation. Simple linear interpolation can be used to determine typical shifts in offset and sensitivity at other supply voltages.

Changes in noise performance, self-test response, and supply current are discussed elsewhere throughout the data sheet. For noise performance, the Noise Performance section should be reviewed. The Using Self-Test section discusses both the operation of self-test over voltage, a square relationship with supply voltage, as well as the conversion of the self-test response in *g*'s to LSBs. Finally, Figure 33 shows the impact of supply voltage on typical current consumption at a 100 Hz output data rate, with all other output data rates following the same trend.

OFFSET PERFORMANCE AT LOWEST DATA RATES

The ADXL345 offers a large number of output data rates and bandwidths, designed for a large range of applications. However, at the lowest data rates, described as those data rates below 6.25 Hz, the offset performance over temperature can vary significantly from the remaining data rates. Figure 54, Figure 55, and Figure 56 show the typical offset performance of the ADXL345 over temperature for the data rates of 6.25 Hz and lower. All plots are normalized to the offset at 100 Hz output data rate; therefore, a nonzero value corresponds to additional offset shift due to temperature for that data rate.

When using the lowest data rates, it is recommended that the operating temperature range of the device be limited to provide minimal offset shift across the operating temperature range. Due to variability between parts, it is also recommended that calibration over temperature be performed if any data rates below 6.25 Hz are in use.

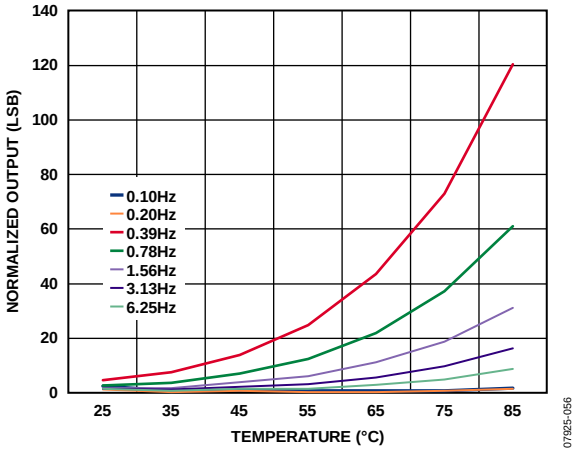


Figure 54. Typical X-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

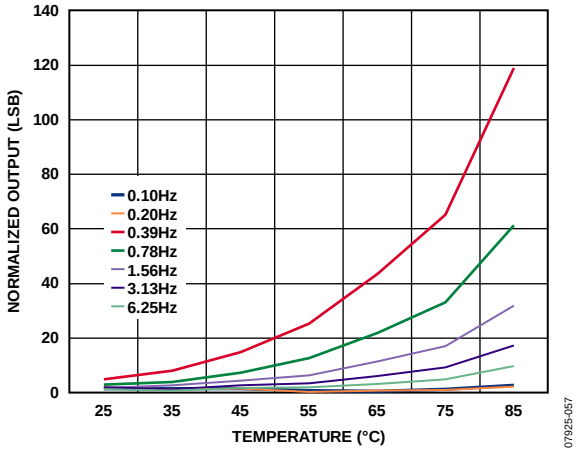


Figure 55. Typical Y-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

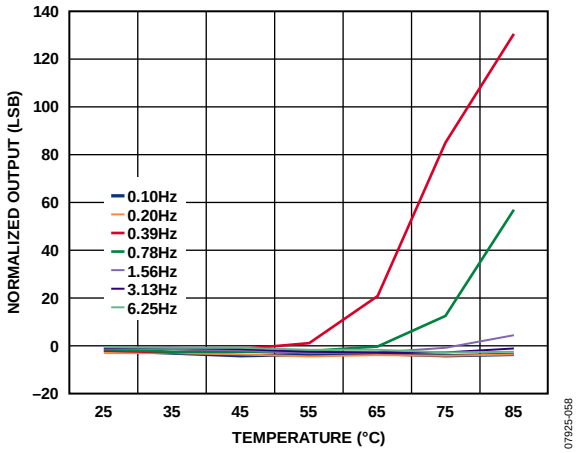


Figure 56. Typical Z-Axis Output vs. Temperature at Lower Data Rates, Normalized to 100 Hz Output Data Rate, V_S = 2.5 V

AXES OF ACCELERATION SENSITIVITY

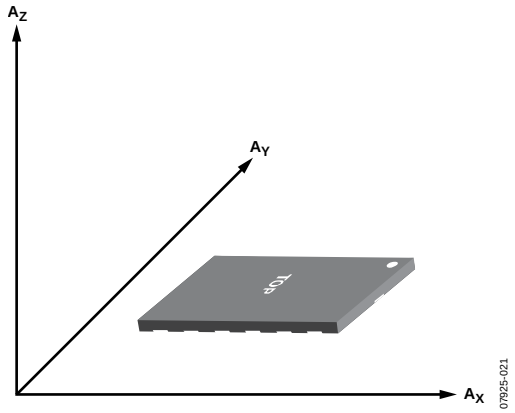


Figure 57. Axes of Acceleration Sensitivity (Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis)

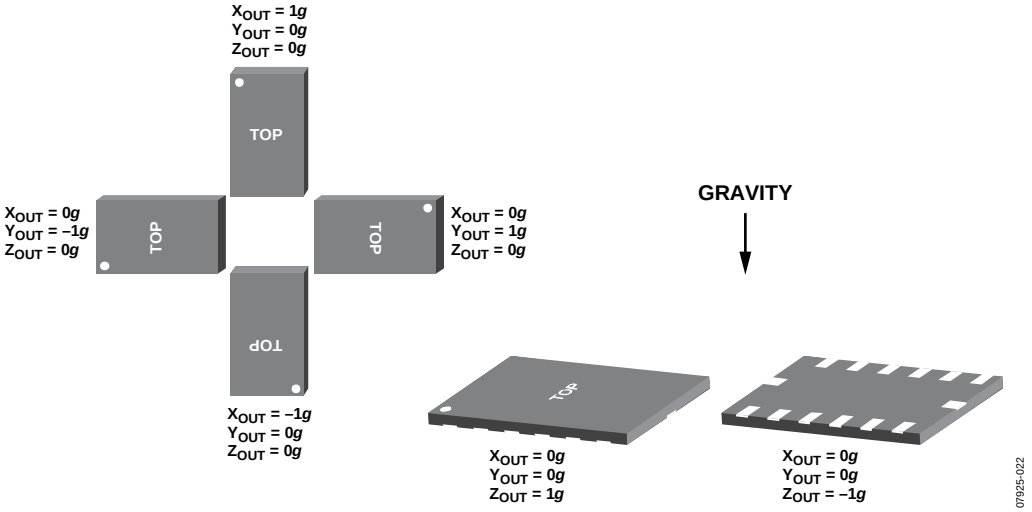


Figure 58. Output Response vs. Orientation to Gravity

LAYOUT AND DESIGN RECOMMENDATIONS

Figure 59 shows the recommended printed wiring board land pattern. Figure 60and Table 24 provide details about the recommended soldering profile.

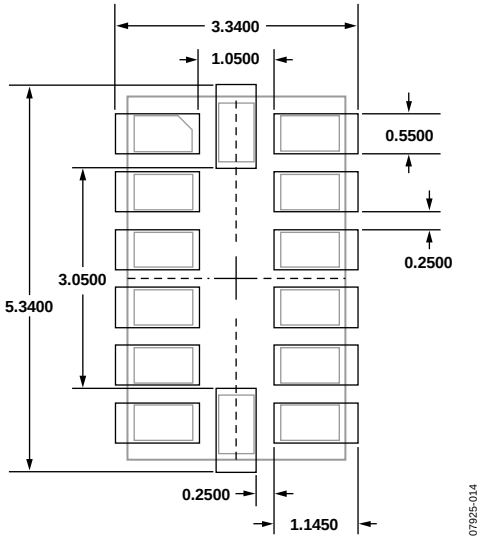


Figure 59. Recommended Printed Wiring Board Land Pattern (Dimensions shown in millimeters)

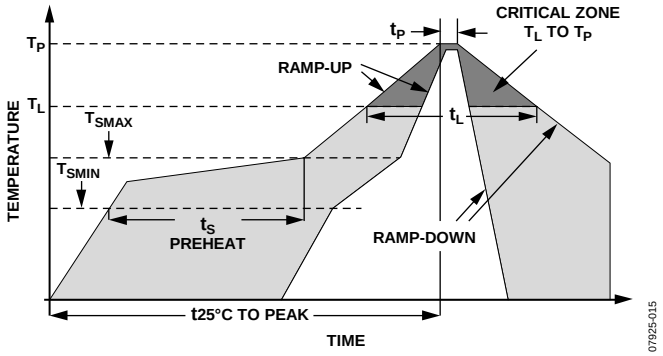


Figure 60. Recommended Soldering Profile

Table 24. Recommended Soldering Profile[1, 2]

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-Free**
Average Ramp Rate from Liquid Temperature (T_L) to Peak Temperature (T_P)	3°C/sec maximum	3°C/sec maximum
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time from T_{SMIN} to T_{SMAX} (t_S)	60 sec to 120 sec	60 sec to 180 sec
T_{SMAX} to T_L Ramp-Up Rate	3°C/sec maximum	3°C/sec maximum
Liquid Temperature (T_L)	183°C	217°C
Time Maintained Above T_L (t_L)	60 sec to 150 sec	60 sec to 150 sec
Peak Temperature (T_P)	240 + 0/−5°C	260 + 0/−5°C
Time of Actual T_P − 5°C (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec maximum	6°C/sec maximum
Time 25°C to Peak Temperature	6 minutes maximum	8 minutes maximum

[1] Based on JEDEC Standard J-STD-020D.1.
[2] For best results, the soldering profile should be in accordance with the recommendations of the manufacturer of the solder paste used.

OUTLINE DIMENSIONS

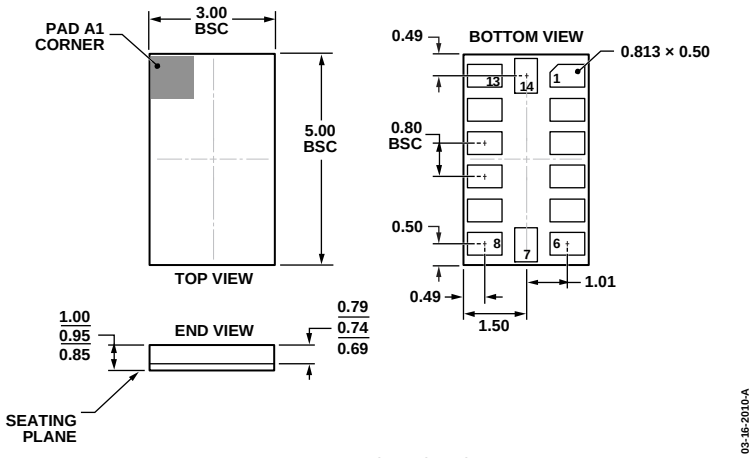


Figure 61. 14-Terminal Land Grid Array [LGA]
(CC-14-1)
Solder Terminations Finish Is Au over Ni
Dimensions shown in millimeters

ORDERING GUIDE

Model[1]	Measurement Range (*g*)	Specified Voltage (V)	Temperature Range	Package Description	Package Option
ADXL345BCCZ	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL7	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
EVAL-ADXL345Z				Evaluation Board	
EVAL-ADXL345Z-DB				Evaluation Board	
EVAL-ADXL345Z-M				Analog Devices Inertial Sensor Evaluation System, Includes ADXL345 Satellite	
EVAL-ADXL345Z-S				ADXL345 Satellite, Standalone	

[1] Z = RoHS Compliant Part.

NOTES

NOTES

NOTES

I²C refers to a communications protocol originally developed by Philips Semiconductors (now NXP Semiconductors).

D07925-0-2/13(D)

www.analog.com

InvenSense Inc.
1197 Borregas Ave, Sunnyvale, CA 94089 U.S.A.
Tel: +1 (408) 988-7339 Fax: +1 (408) 988-8104
Website: www.invensense.com

Document Number: PS-ITG-3200A-00-01.4
Revision: 1.4
Release Date: 03/30/2010

ITG-3200
Product Specification
Revision 1.4

CONTENTS

1 Document Information

1.1 Revision History

Revision Date	Revision	Description
10/23/09	1.0	Initial Release
10/28/09	1.1	Edits for readability
02/12/2010	1.2	• Changed full-scale range and sensitivity scale factor (Sections 2, 3.1, 5.3, and 8.3) • Changed sensitivity scale factor variation over temperature (Section 3.1) • Changed total RMS noise spec (Section 3.1) • Added range for temperature sensor (Section 3.1) • Updated VDD Power-Supply Ramp Rate specification (Sections 3.2 and 4.4) • Added VLOGIC Voltage Range condition (Section 3.2) • Added VLOGIC Reference Voltage Ramp Rate specification (Sections 3.2 and 4.4) • Updated Start-Up Time for Register Read/Write specification (Section 3.2) • Updated Input logic levels for AD0 and CLKIN (Section 3.2) • Updated Level I_{OL} specifications for the I^2C interface (Section 3.3) • Updated Frequency Variation Over Temperature specification for internal clock source (Section 3.4) • Updated VLOGIC conditions for I^2C Characterization (Section 3.5) • Updated ESD specification (Section 3.6) • Added termination requirements for CLKIN if unused (Section 4.1) • Added recommended power-on procedure diagram (Section 4.4) • Changed DLPF_CFG setting 7 to reserved (Section 8.3) • Changed Reflow Specification description (Section 9.12) • Removed errata specifications
03/05/2010	1.3	• Updated temperature sensor linearity spec (Section 3.1) • Updated VDD Power-Supply Ramp Rate timing figure (Sections 3.2 and 4.4) • Updated VLOGIC Reference Voltage timing figure (Section 4.4) • Added default values to registers (all of Section 8) • Updated FS_SEL description (Section 8.3) • Updated package outline drawing and dimensions (Section 9.2) • Updated Reliability (Section 10.1 and 10.2) • Removed Environmental Compliance (Section 11)
03/30/2010	1.4	• Removed confidentiality mark

1.2 Purpose and Scope

This document is a preliminary product specification, providing a description, specifications, and design related information for the ITG-3200™. Electrical characteristics are based upon simulation results and limited characterization data of advanced samples only. Specifications are subject to change without notice. Final specifications will be updated based upon characterization of final silicon.

1.3 Product Overview

The ITG-3200 is the world's first single-chip, digital-output, 3-axis MEMS gyro IC optimized for gaming, 3D mice, and 3D remote control applications. The part features enhanced bias and sensitivity temperature stability, reducing the need for user calibration. Low frequency noise is lower than previous generation devices, simplifying application development and making for more-responsive remote controls.

The ITG-3200 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I^2C (400kHz) interface. Additional features include an embedded temperature sensor and a 2% accurate internal oscillator. This breakthrough in gyroscope technology provides a dramatic 67% package size reduction, delivers a 50% power reduction, and has inherent cost advantages compared to competing multi-chip gyro solutions.

By leveraging its patented and volume-proven Nasiri-Fabrication platform, which integrates MEMS wafers with companion CMOS electronics through wafer-level bonding, InvenSense has driven the ITG-3200 package size down to a revolutionary footprint of 4x4x0.9mm (QFN), while providing the highest performance, lowest noise, and the lowest cost semiconductor packaging required for handheld consumer electronic devices. The part features a robust 10,000*g* shock tolerance, as required by portable consumer equipment.

For power supply flexibility, the ITG-3200 has a separate VLOGIC reference pin, in addition to its analog supply pin, VDD, which sets the logic levels of its I^2C interface. The VLOGIC voltage may be anywhere from 1.71V min to VDD max.

1.4 Applications

- Motion-enabled game controllers
- Motion-based portable gaming
- Motion-based 3D mice and 3D remote controls
- "No Touch" UI
- Health and sports monitoring

2 Features

The ITG-3200 triple-axis MEMS gyroscope includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit with a sensitivity of 14.375 LSBs per °/sec and a full-scale range of ±2000°/sec
- Three integrated 16-bit ADCs provide simultaneous sampling of gyros while requiring no external multiplexer
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Low frequency noise lower than previous generation devices, simplifying application development and making for more-responsive motion processing
- Digitally-programmable low-pass filter
- Low 6.5mA operating current consumption for long battery life 
- Wide VDD supply voltage range of 2.1V to 3.6V
- Flexible VLOGIC reference voltage allows for I^2C interface voltages from 1.71V to VDD
- Standby current: 5µA
- Smallest and thinnest package for portable devices (4x4x0.9mm QFN)
- No high pass filter needed
- Turn on time: 50ms
- Digital-output temperature sensor
- Factory calibrated scale factor
- 10,000 *g* shock tolerant
- Fast Mode I^2C (400kHz) serial interface
- On-chip timing generator clock frequency is accurate to +/-2% over full temperature range
- Optional external clock inputs of 32.768kHz or 19.2MHz to synchronize with system clock
- MEMS structure hermetically sealed and bonded at wafer level
- RoHS and Green compliant

3 Electrical Characteristics

3.1 Sensor Specifications

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameter	Conditions	Min	Typical	Max	Unit	Note
GYRO SENSITIVITY						
Full-Scale Range	FS_SEL=3		±2000		º/s	4
Gyro ADC Word Length			16		Bits	3
Sensitivity Scale Factor	FS_SEL=3		14.375		LSB/(º/s)	3
Sensitivity Scale Factor Tolerance	25°C	-6		+6	%	1
Sensitivity Scale Factor Variation Over Temperature			±10		%	2
Nonlinearity	Best fit straight line; 25°C		0.2		%	6
Cross-Axis Sensitivity			2		%	6
GYRO ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance			±40		º/s	1
ZRO Variation Over Temperature	-40°C to +85°C		±40		º/s	2
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.2V		0.2		º/s	5
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.2V		0.2		º/s	5
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.2V		4		º/s	5
Linear Acceleration Sensitivity	Static		0.1		º/s/*g*	6
GYRO NOISE PERFORMANCE	**FS_SEL=3**					
Total RMS noise	100Hz LPF (DLPFCFG=2)		0.38		º/s-rms	1
Rate Noise Spectral Density	At 10Hz		0.03		º/s/√Hz	2
GYRO MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	1
Y-Axis		27	30	33	kHz	1
Z-Axis		24	27	30	kHz	1
Frequency Separation	Between any two axes	1.7			kHz	1
GYRO START-UP TIME	**DLPFCFG=0**					
ZRO Settling	to ±1º/s of Final		50		ms	6
TEMPERATURE SENSOR						
Range			-30 to +85		ºC	2
Sensitivity			280		LSB/ºC	2
Temperature Offset	35ºC		-13,200		LSB	1
Initial Accuracy	35ºC		TBD		°C	
Linearity	Best fit straight line (-30°C to +85°C)		±1		°C	2, 5
TEMPERATURE RANGE						
Specified Temperature Range		-40		85	ºC	

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
3. Based on design, through modeling and simulation across PVT
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature
6. Tested on 5 parts at room temperature

3.2 Electrical Specifications

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
VDD POWER SUPPLY						
Operating Voltage Range		2.1		3.6	V	2
Power-Supply Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value (see Figure in Section 4.4)	0		5	ms	2
Normal Operating Current			6.5		mA	1
Sleep Mode Current			5		µA	5
VLOGIC REFERENCE VOLTAGE						
Voltage Range	VLOGIC must be ≤VDD at all times	1.71		VDD	V	
VLOGIC Ramp Rate	Monotonic ramp. Ramp rate is 10% to 90% of the final value (see Figure in Section 4.4)			1	ms	6
Normal Operating Current			100		µA	
START-UP TIME FOR REGISTER READ/WRITE			20		ms	5
I^2C ADDRESS	AD0 = 0		1101000			6
	AD0 = 1		1101001			6
DIGITAL INPUTS (AD0, CLKIN)						
V_{IH}, High Level Input Voltage		0.9*VLOGIC			V	5
V_{IL}, Low Level Input Voltage				0.1*VLOGIC	V	5
C_I, Input Capacitance				5	pF	7
DIGITAL OUTPUT (INT)						
V_{OH}, High Level Output Voltage	OPEN=0, Rload=1MΩ	0.9*VLOGIC			V	2
V_{OL}, Low Level Output Voltage	OPEN=0, Rload=1MΩ			0.1*VLOGIC	V	2
$V_{OL.INT1}$, INT Low-Level Output Voltage	OPEN=1, 0.3mA sink current			0.1	V	2
Output Leakage Current	OPEN=1		100		nA	4
t_{INT}, INT Pulse Width	LATCH_INT_EN=0		50		µs	4

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature
6. Guaranteed by design

3.3 Electrical Specifications, continued

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Typical	Units	Notes
I^2C I/O (SCL, SDA)				
V_{IL}, LOW-Level Input Voltage		-0.5 to 0.3*VLOGIC	V	2
V_{IH}, HIGH-Level Input Voltage		0.7*VLOGIC to VLOGIC + 0.5V	V	2
V_{hys}, Hysteresis		0.1*VLOGIC	V	2
V_{OL1}, LOW-Level Output Voltage	3mA sink current	0 to 0.4	V	2
I_{OL}, LOW-Level Output Current	V_{OL} = 0.4V V_{OL} = 0.6V	3 6	mA mA	2 2
Output Leakage Current		100	nA	4
t_{of}, Output Fall Time from V_{IHmax} to V_{ILmax}	Cb bus cap. in pF	20+0.1Cb to 250	ns	2
C_I, Capacitance for Each I/O pin		10	pF	5

Notes:

2. Based on characterization of 5 pieces over temperature.
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Guaranteed by design

3.4 Electrical Specifications, continued

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.71V to VDD, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
INTERNAL CLOCK SOURCE	**CLKSEL=0, 1, 2, or 3**					
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	4
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	4
Clock Frequency Initial Tolerance	CLKSEL=0, 25°C	-2		+2	%	1
	CLKSEL=1,2,3; 25°C	-1		+1	%	1
Frequency Variation over Temperature	CLKSEL=0		-15 to +10		%	2
	CLKSEL=1,2,3		+/-1		%	2
PLL Settling Time	CLKSEL=1,2,3		1		ms	3
EXTERNAL 32.768kHz CLOCK	**CLKSEL=4**					
External Clock Frequency			32.768		kHz	3
External Clock Jitter	Cycle-to-cycle rms		1 to 2		µs	3
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8.192		kHz	3
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1.024		kHz	3
PLL Settling Time			1		ms	3
EXTERNAL 19.2MHz CLOCK	**CLKSEL=5**					
External Clock Frequency			19.2		MHz	3
Sample Rate, Fast	DLPFCFG=0 SAMPLERATEDIV = 0		8		kHz	3
Sample Rate, Slow	DLPFCFG=1,2,3,4,5, or 6 SAMPLERATEDIV = 0		1		kHz	3
PLL Settling Time			1		ms	3
Charge Pump Clock Frequency						
Frequency	1st Stage, 25°C		8.5		MHz	5
	2nd Stage, 25°C		68		MHz	5
	Over temperature		+/-15		%	5

Notes:

1. Tested in production
2. Based on characterization of 30 pieces over temperature on evaluation board or in socket
3. Based on design, through modeling and simulation across PVT
4. Typical. Randomly selected part measured at room temperature on evaluation board or in socket
5. Based on characterization of 5 pieces over temperature.

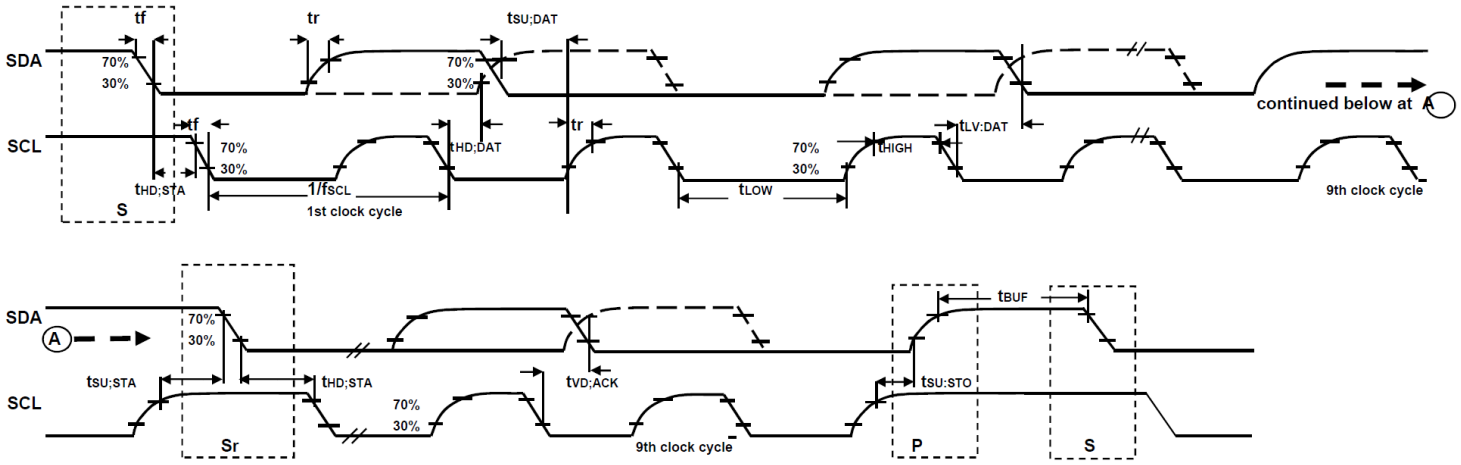
3.5 I²C Timing Characterization

Typical Operating Circuit of Section 4.2, VDD = 2.5V, VLOGIC = 1.8V±5%, 2.5V±5%, 3.0V±5%, or 3.3V±5%, T_A=25°C.

Parameters	Conditions	Min	Typical	Max	Units	Notes
I²C TIMING	**I²C FAST-MODE**					
f_{SCL}, SCL Clock Frequency		0		400	kHz	1
$t_{HD.STA}$, (Repeated) START Condition Hold Time		0.6			us	1
t_{LOW}, SCL Low Period		1.3			us	1
t_{HIGH}, SCL High Period		0.6			us	1
$t_{SU.STA}$, Repeated START Condition Setup Time		0.6			us	1
$t_{HD.DAT}$, SDA Data Hold Time		0			us	1
$t_{SU.DAT}$, SDA Data Setup Time		100			ns	1
t_r, SDA and SCL Rise Time	Cb bus cap. from 10 to 400pF	20+0.1Cb		300	ns	1
t_f, SDA and SCL Fall Time	Cb bus cap. from 10 to 400pF	20+0.1Cb		300	ns	1
$t_{SU.STO}$, STOP Condition Setup Time		0.6			us	1
t_{BUF}, Bus Free Time Between STOP and START Condition		1.3			us	1
C_b, Capacitive Load for each Bus Line				400	pF	2
$t_{VD.DAT}$, Data Valid Time				0.9	us	1
$t_{VD.ACK}$, Data Valid Acknowledge Time				0.9	us	1

Notes:

1. Based on characterization of 5 pieces over temperature on evaluation board or in socket
2. Guaranteed by design



I²C Bus Timing Diagram

3.6 Absolute Maximum Ratings

Stresses above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

Absolute Maximum Ratings

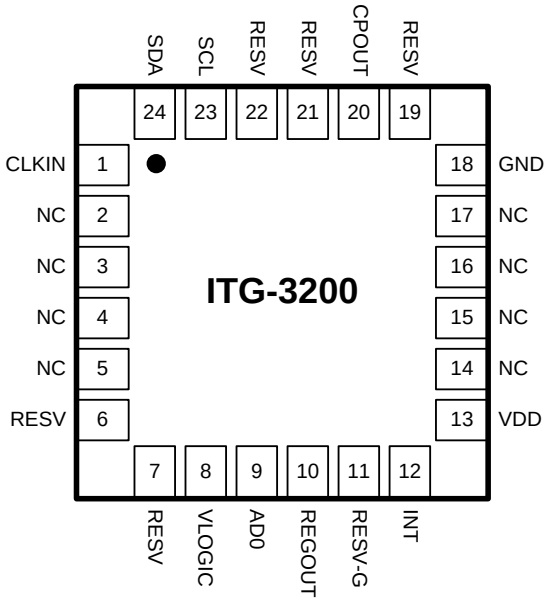
Parameter	Rating
Supply Voltage, VDD	-0.5V to +6V
VLOGIC Input Voltage Level	-0.5V to VDD + 0.5V
REGOUT	-0.5V to 2V
Input Voltage Level (CLKIN, AD0)	-0.5V to VDD + 0.5V
SCL, SDA, INT	-0.5V to VLOGIC + 0.5V
CPOUT (2.1V ≤ VDD ≤ 3.6V)	-0.5V to 30V
Acceleration (Any Axis, unpowered)	10,000*g* for 0.3ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Electrostatic Discharge (ESD) Protection	1.5kV (HBM); 200V (MM)

4 Applications Information

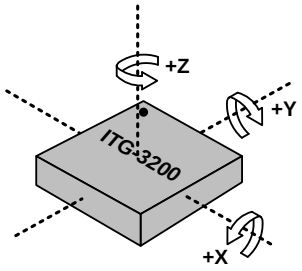
4.1 Pin Out and Signal Description

Number	Pin	Pin Description
1	CLKIN	Optional external reference clock input. Connect to GND if unused.
8	VLOGIC	Digital IO supply voltage. VLOGIC must be ≤ VDD at all times.
9	AD0	I^2C Slave Address LSB
10	REGOUT	Regulator filter capacitor connection
12	INT	Interrupt digital output (totem pole or open-drain)
13	VDD	Power supply voltage
18	GND	Power supply ground
11	RESV-G	Reserved - Connect to ground.
6, 7, 19, 21, 22	RESV	Reserved. Do not connect.
20	CPOUT	Charge pump capacitor connection
23	SCL	I^2C serial clock
24	SDA	I^2C serial data
2, 3, 4, 5, 14, 15, 16, 17	NC	Not internally connected. May be used for PCB trace routing.

Top View

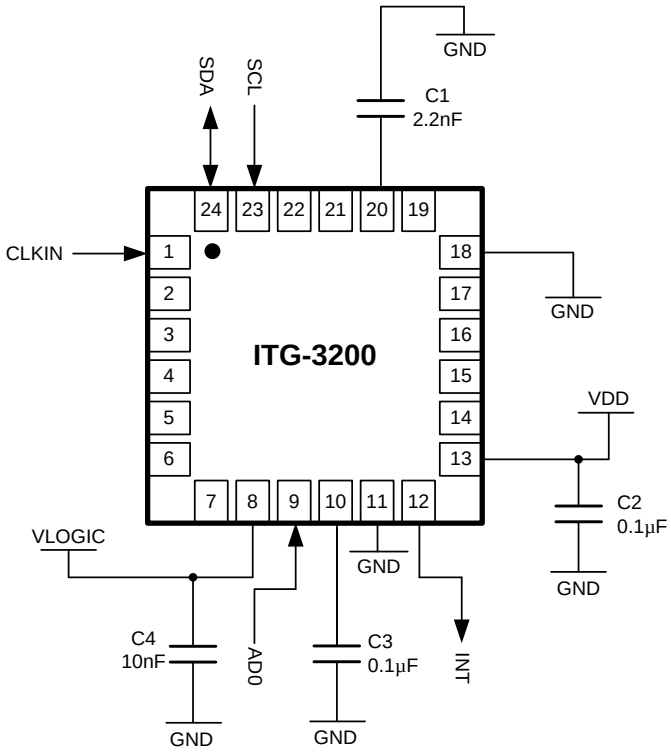


QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

4.2 Typical Operating Circuit

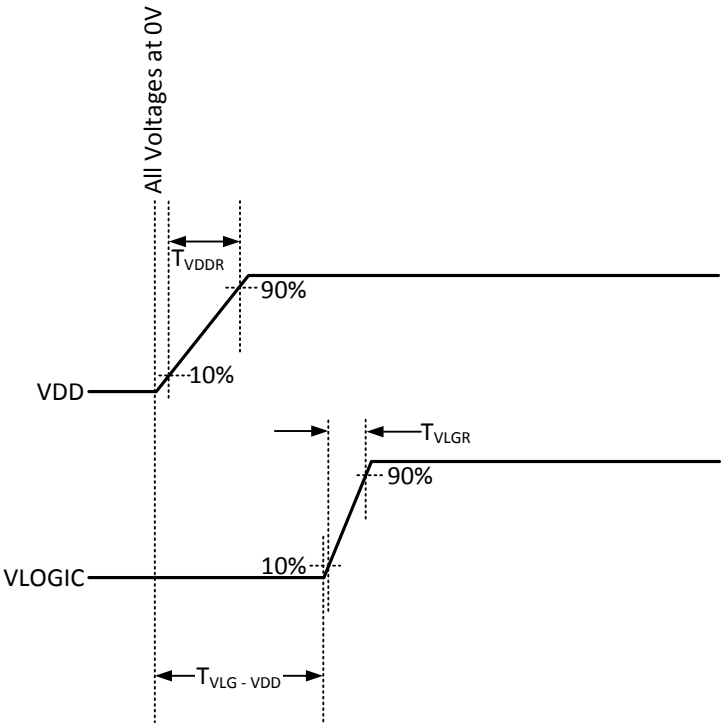


Typical Operating Circuit

4.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Charge Pump Capacitor	C1	Ceramic, X7R, 2.2nF ±10%, 50V	1
VDD Bypass Capacitor	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Regulator Filter Capacitor	C3	Ceramic, X7R, 0.1µF ±10%, 2V	1
VLOGIC Bypass Capacitor	C4	Ceramic, X7R, 10nF ±10%, 4V	1

4.4 Recommended Power-On Procedure

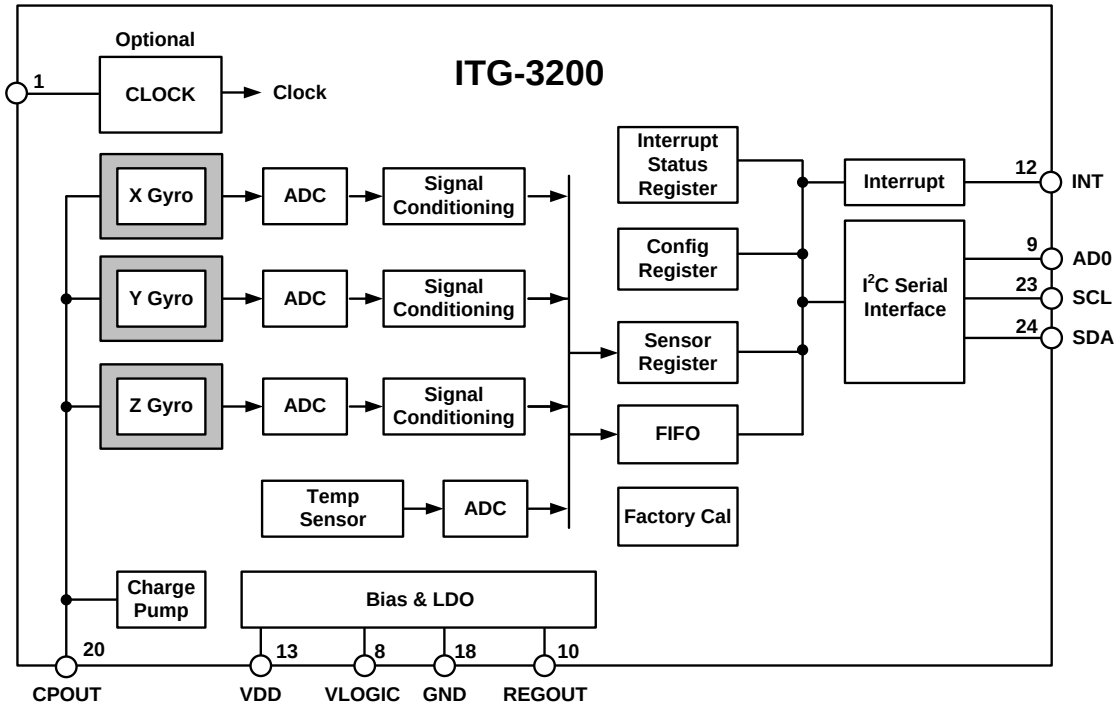


Power-Up Sequencing

1. T_{VDDR} is VDD rise time: Time for VDD to rise from 10% to 90% of its final value
2. T_{VDDR} is ≤5msec
3. T_{VLGR} is VLOGIC rise time: Time for VLOGIC to rise from 10% to 90% of its final value
4. T_{VLGR} is ≤1msec
5. $T_{VLG-VDD}$ is the delay from the start of VDD ramp to the start of VLOGIC rise
6. $T_{VLG-VDD}$ is 0 to 20msec but VLOGIC amplitude must always be ≤VDD amplitude
7. VDD and VLOGIC must be monotonic ramps

5 Functional Overview

5.1 Block Diagram



5.2 Overview

The ITG-3200 consists of the following key blocks and functions:

- Three-axis MEMS rate gyroscope sensors with individual 16-bit ADCs and signal conditioning
- I^2C serial communications interface
- Clocking
- Sensor Data Registers
- Interrupts
- Digital-Output Temperature Sensor
- Bias and LDO
- Charge Pump

5.3 Three-Axis MEMS Gyroscope with 16-bit ADCs and Signal Conditioning

The ITG-3200 consists of three independent vibratory MEMS gyroscopes, which detect rotational rate about the X (roll), Y (pitch), and Z (yaw) axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a deflection that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis.

The full-scale range of the gyro sensors is preset to ±2000 degrees per second (°/s). The ADC output rate is programmable up to a maximum of 8,000 samples per second down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

5.4 I²C Serial Communications Interface

The ITG-3200 communicates to a system processor using the I²C serial interface, and the device always acts as a slave when communicating to the system processor. **The logic level for communications to the master is set by the voltage on the VLOGIC pin.** The LSB of the of the I²C slave address is set by pin 9 (AD0).

5.5 Clocking

The ITG-3200 has a flexible clocking scheme, allowing for a variety of internal or external clock sources for the internal synchronous circuitry. This synchronous circuitry includes the signal conditioning, ADCs, and various control circuits and registers. An on-chip PLL provides flexibility in the allowable inputs for generating this clock.

Allowable internal sources for generating the internal clock are:

- An internal relaxation oscillator (less accurate)
- Any of the X, Y, or Z gyros' MEMS oscillators (with an accuracy of ±2% over temperature)

Allowable external clocking sources are:

- 32.768kHz square wave
- 19.2MHz square wave

Which source to select for generating the internal synchronous clock depends on the availability of external sources and the requirements for clock accuracy. There are also start-up conditions to consider. When the ITG-3200 first starts up, the device operates off of its internal clock until programmed to operate from another source. This allows the user, for example, to wait for the MEMS oscillators to stabilize before they are selected as the clock source.

5.6 Sensor Data Registers

The sensor data registers contain the latest gyro and temperature data. They are read-only registers, and are accessed via the Serial Interface. Data from these registers may be read at any time, however, the interrupt function may be used to determine when new data is available.

5.7 Interrupts

Interrupt functionality is configured via the Interrupt Configuration register. Items that are configurable include the INT pin configuration, the interrupt latching and clearing method, and triggers for the interrupt. Items that can trigger an interrupt are (1) Clock generator locked to new reference oscillator (used when switching clock sources); and (2) new data is available to be read from the Data registers. The interrupt status can be read from the Interrupt Status register.

5.8 Digital-Output Temperature Sensor

An on-chip temperature sensor and ADC are used to measure the ITG-3200 die temperature. The readings from the ADC can be read from the Sensor Data registers.

5.9 Bias and LDO

The bias and LDO sections take in an unregulated VDD supply from 2.1V to 3.6V and generate the internal supply and the references voltages and currents required by the ITG-3200. The LDO output is bypassed by a capacitor at REGOUT. Additionally, the part has a VLOGIC reference voltage which sets the logic levels for its I²C interface.

5.10 Charge Pump

An on-board charge pump generates the high voltage (25V) required to drive the MEMS oscillators. Its output is bypassed by a capacitor at CPOUT.

6 Digital Interface

6.1 I²C Serial Interface

The internal registers and memory of the ITG-3200 can be accessed using I²C at up to 400kHz.

Serial Interface

Pin Number	Pin Name	Pin Description
8	VLOGIC	Digital IO supply voltage. VLOGIC must be ≤ VDD at all times.
9	AD0	I²C Slave Address LSB
23	SCL	I²C serial clock
24	SDA	I²C serial data

6.1.1 I²C Interface

I²C is a two wire interface comprised of the signals serial data (SDA) and serial clock (SCL). In general, the lines are open-drain and bi-directional. In a generalized I²C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master.

The ITG-3200 always operates as a slave device when communicating to the system processor, which thus acts as the master. SDA and SCL lines typically need pull-up resistors to VDD. The maximum bus speed is 400kHz.

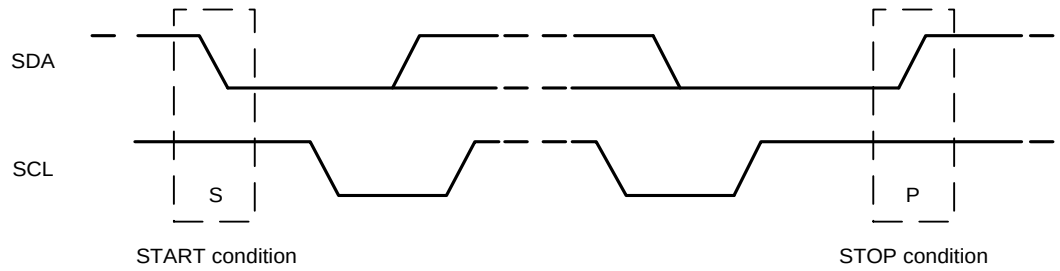
The slave address of the ITG-3200 devices is b110100X which is 7 bits long. The LSB bit of the 7 bit address is determined by the logic level on pin 9. This allows two ITG-3200 devices to be connected to the same I²C bus. When used in this configuration, the address of the one of the devices should be b1101000 (pin 9 is logic low) and the address of the other should be b1101001 (pin 9 is logic high). The I²C address is stored in register 0 (WHO_AM_I register).

I²C Communications Protocol

START (S) and STOP (P) Conditions

Communication on the I²C bus starts when the master puts the START condition (S) on the bus, which is defined as a HIGH-to-LOW transition of the SDA line while SCL line is HIGH (see figure below). The bus is considered to be busy until the master puts a STOP condition (P) on the bus, which is defined as a LOW to HIGH transition on the SDA line while SCL is HIGH (see figure below).

Additionally, the bus remains busy if a repeated START (Sr) is generated instead of a STOP condition.

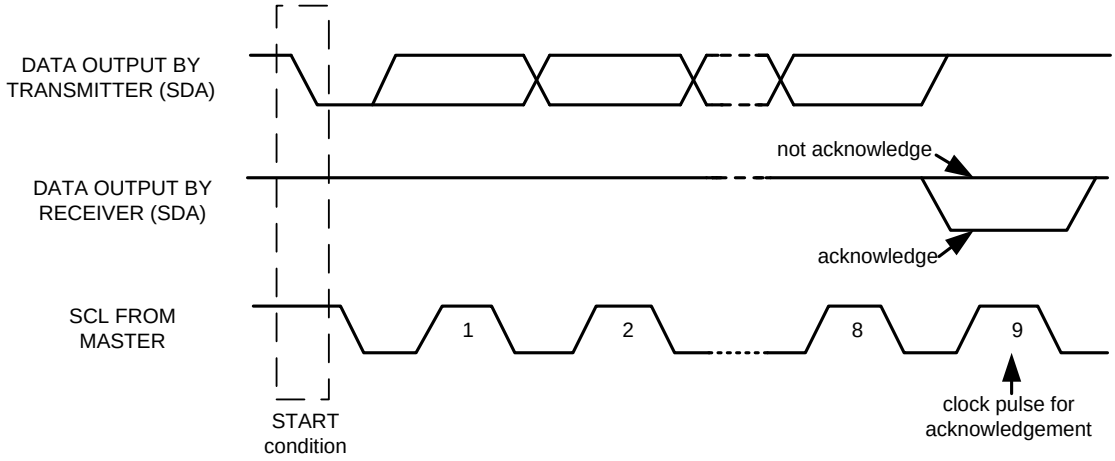


START and STOP Conditions

Data Format / Acknowledge

I²C data bytes are defined to be 8 bits long. There is no restriction to the number of bytes transmitted per data transfer. Each byte transferred must be followed by an acknowledge (ACK) signal. The clock for the acknowledge signal is generated by the master, while the receiver generates the actual acknowledge signal by pulling down SDA and holding it low during the HIGH portion of the acknowledge clock pulse.

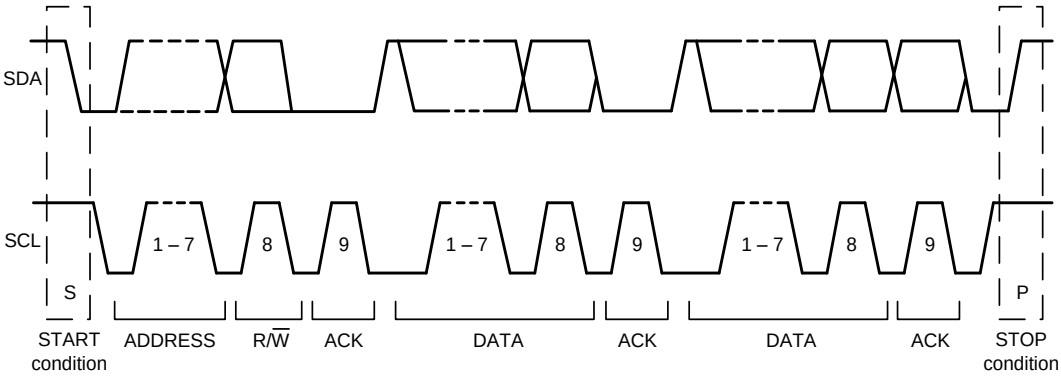
If a slave is busy and cannot transmit or receive another byte of data until some other task has been performed, it can hold SCL LOW, thus forcing the master into a wait state. Normal data transfer resumes when the slave is ready, and releases the clock line (see figure below).



Acknowledge on the I²C Bus

Communications

After beginning communications with the START condition (S), the master sends a 7-bit slave address followed by an 8th bit, the read/write bit. The read/write bit indicates whether the master is receiving data from or is writing to the slave device. Then, the master releases the SDA line and waits for the acknowledge signal (ACK) from the slave device. Each byte transferred must be followed by an acknowledge bit. To acknowledge, the slave device pulls the SDA line LOW and keeps it LOW for the high period of the SCL line. Data transmission is always terminated by the master with a STOP condition (P), thus freeing the communications line. However, the master can generate a repeated START condition (Sr), and address another slave without first generating a STOP condition (P). A LOW to HIGH transition on the SDA line while SCL is HIGH defines the stop condition. All SDA changes should take place when SCL is low, with the exception of start and stop conditions.



Complete I²C Data Transfer

To write the internal ITG-3200 device registers, the master transmits the start condition (S), followed by the I^2C address and the write bit (0). At the 9th clock cycle (when the clock is high), the ITG-3200 device acknowledges the transfer. Then the master puts the register address (RA) on the bus. After the ITG-3200 acknowledges the reception of the register address, the master puts the register data onto the bus. This is followed by the ACK signal, and data transfer may be concluded by the stop condition (P). To write multiple bytes after the last ACK signal, the master can continue outputting data rather than transmitting a stop signal. In this case, the ITG-3200 device automatically increments the register address and loads the data to the appropriate register. The following figures show single and two-byte write sequences.

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

To read the internal ITG-3200 device registers, the master first transmits the start condition (S), followed by the I^2C address and the write bit (0). At the 9th clock cycle (when clock is high), the ITG acknowledges the transfer. The master then writes the register address that is going to be read. Upon receiving the ACK signal from the ITG-3200, the master transmits a start signal followed by the slave address and read bit. As a result, the ITG-3200 sends an ACK signal and the data. The communication ends with a not acknowledge (NACK) signal and a stop bit from master. The NACK condition is defined such that the SDA line remains high at the 9th clock cycle. To read multiple bytes of data, the master can output an acknowledge signal (ACK) instead of a not acknowledge (NACK) signal. In this case, the ITG-3200 automatically increments the register address and outputs data from the appropriate register. The following figures show single and two-byte read sequences.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

I²C Terms

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I²C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9th clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9th clock cycle
RA	ITG-3200 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

7 Register Map

Addr Hex	Addr Decimal	Register Name	R/W	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	WHO_AM_I	R/W	-	ID						-
15	21	SMPLRT_DIV	R/W	SMPLRT_DIV							
16	22	DLPF_FS	R/W	-	-	-	FS_SEL		DLPF_CFG		
17	23	INT_CFG	R/W	ACTL	OPEN	LATCH_ INT_EN	INT_ ANYRD_ 2CLEAR	-	ITG_RDY _EN	-	RAW_ RDY_ EN
1A	26	INT_STATUS	R	-	-	-	-	-	ITG_RDY	-	RAW_ DATA_ RDY
1B	27	TEMP_OUT_H	R	TEMP_OUT_H							
1C	28	TEMP_OUT_L	R	TEMP_OUT_L							
1D	29	GYRO_XOUT_H	R	GYRO_XOUT_H							
1E	30	GYRO_XOUT_L	R	GYRO_XOUT_L							
1F	31	GYRO_YOUT_H	R	GYRO_YOUT_H							
20	32	GYRO_YOUT_L	R	GYRO_YOUT_L							
21	33	GYRO_ZOUT_H	R	GYRO_ZOUT_H							
22	34	GYRO_ZOUT_L	R	GYRO_ZOUT_L							
3E	62	PWR_MGM	R/W	H_RESET	SLEEP	STBY_XG	STBY_YG	STBY_ZG	CLK_SEL		

8 Register Description

This section details each register within the InvenSense ITG-3200 gyroscope. Note that any bit that is not defined should be set to zero in order to be compatible with future InvenSense devices.

The register space allows single-byte reads and writes, as well as burst reads and writes. When performing burst reads or writes, the memory pointer will increment until either (1) reading or writing is terminated by the master, or (2) the memory pointer reaches certain reserved registers between registers 33 and 60.

8.1 Register 0 – Who Am I

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	-	ID						-

Description:
This register is used to verify the identity of the device.
Parameters:
ID Contains the I^2C address of the device, which can also be changed by writing to this register.

The Power-On-Reset value of Bit6: Bit1 is 110 100.

8.2 Register 21 – Sample Rate Divider

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
15	21	SMPLRT_DIV								00h

Description:
This register determines the sample rate of the ITG-3200 gyros. The gyros outputs are sampled internally at either 1kHz or 8kHz, determined by the *DLPF_CFG* setting (see register 22). This sampling is then filtered digitally and delivered into the sensor registers after the number of cycles determined by this register. The sample rate is given by the following formula:

$F_{sample} = F_{internal} / (divider+1)$, where $F_{internal}$ is either 1kHz or 8kHz

As an example, if the internal sampling is at 1kHz, then setting this register to 7 would give the following:

$F_{sample} = 1kHz / (7 + 1) = 125Hz$, or 8ms per sample

Parameters:
SMPLRT_DIV Sample rate divider: 0 to 255

8.3 Register 22 – DLPF, Full Scale

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
16	22	-			FS_SEL		DLPF_CFG			00h

Description:

This register configures several parameters related to the sensor acquisition.

The *FS_SEL* parameter allows setting the full-scale range of the gyro sensors, as described in the table below. The power-on-reset value of *FS_SEL* is 00h. **Set to 03h for proper operation.**

FS_SEL

FS_SEL	Gyro Full-Scale Range
0	Reserved
1	Reserved
2	Reserved
3	±2000°/sec

The *DLPF_CFG* parameter sets the digital low pass filter configuration. It also determines the internal sampling rate used by the device as shown in the table below.

DLPF_CFG

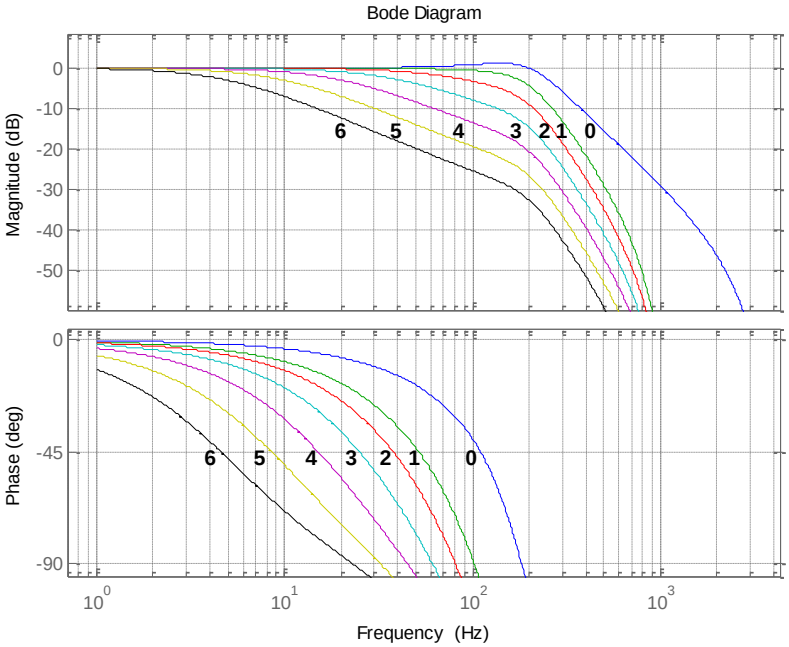
DLPF_CFG	Low Pass Filter Bandwidth	Internal Sample Rate
0	256Hz	8kHz
1	188Hz	1kHz
2	98Hz	1kHz
3	42Hz	1kHz
4	20Hz	1kHz
5	10Hz	1kHz
6	5Hz	1kHz
7	Reserved	Reserved

Parameters:

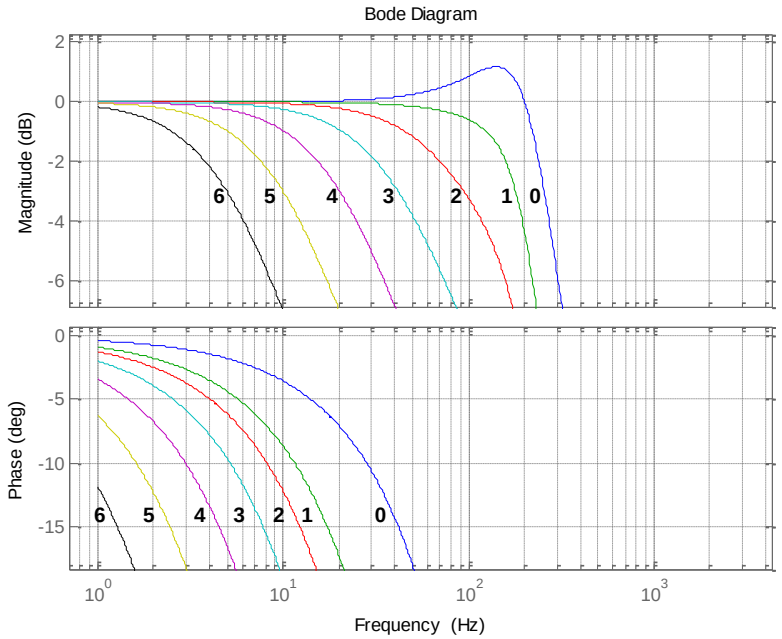
FS_SEL Full scale selection for gyro sensor data

DLPF_CFG Digital low pass filter configuration and internal sampling rate configuration

DLPF Characteristics: The gain and phase responses of the digital low pass filter settings (*DLPF_CFG*) are shown below:



Gain and Phase vs. Digital Filter Setting



Gain and Phase vs. Digital Filter Setting, Showing Passband Details

8.4 Register 23 – Interrupt Configuration

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
17	23	ACTL	OPEN	LATCH_ INT_EN	INT_ ANYRD_ 2CLEAR	0	ITG_RDY_ EN	0	RAW_ RDY_ EN	00h

Description:

This register configures the interrupt operation of the device. The interrupt output pin (INT) configuration can be set, the interrupt latching/clearing method can be set, and the triggers for the interrupt can be set.

Note that if the application requires reading every sample of data from the ITG-3200 part, it is best to enable the raw data ready interrupt (*RAW_RDY_EN*). This allows the application to know when new sample data is available.

Parameters:

ACTL	Logic level for INT output pin – 1=active low, 0=active high
OPEN	Drive type for INT output pin – 1=open drain, 0=push-pull
LATCH_INT_EN	Latch mode – 1=latch until interrupt is cleared, 0=50us pulse
INT_ANYRD_2CLEAR	Latch clear method – 1=any register read, 0=status register read only
ITG_RDY_EN	Enable interrupt when device is ready (PLL ready after changing clock source)
RAW_RDY_EN	Enable interrupt when data is available
0	Load zeros into Bits 1 and 3 of the Interrupt Configuration register.

8.5 Register 26 – Interrupt Status

Type: Read only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
1A	26	-	-	-	-	-	ITG_RDY	-	RAW_ DATA_ RDY	00h

Description:

This register is used to determine the status of the ITG-3200 interrupts. Whenever one of the interrupt sources is triggered, the corresponding bit will be set. The polarity of the interrupt pin (active high/low) and the latch type (pulse or latch) has no affect on these status bits.

Use the Interrupt Configuration register (23) to enable the interrupt triggers. If the interrupt is not enabled, the associated status bit will not get set.

In normal use, the *RAW_DATA_RDY* interrupt is used to determine when new sensor data is available in either the sensor registers (27 to 32).

Interrupt Status bits get cleared as determined by INT_ANYRD_2CLEAR in the interrupt configuration register (23).

Parameters:

ITG_RDY	PLL ready
RAW_DATA_RDY	Raw data is ready

8.6 Registers 27 to 34 – Sensor Registers

Type: Read only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	TEMP_OUT_H							
1C	28	TEMP_OUT_L							
1D	29	GYRO_XOUT_H							
1E	30	GYRO_XOUT_L							
1F	31	GYRO_YOUT_H							
20	32	GYRO_YOUT_L							
21	33	GYRO_ZOUT_H							
22	34	GYRO_ZOUT_L							

Description:
These registers contain the gyro and temperature sensor data for the ITG-3200 parts. At any time, these values can be read from the device; however it is best to use the interrupt function to determine when new data is available.

Parameters:
TEMP_OUT_H/L 16-bit temperature data (2's complement format)
GYRO_XOUT_H/L 16-bit X gyro output data (2's complement format)
GYRO_YOUT_H/L 16-bit Y gyro output data (2's complement format)
GYRO_ZOUT_H/L 16-bit Y gyro output data (2's complement format)

8.7 Register 62 – Power Management

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
3E	62	H_RESET	SLEEP	STBY _XG	STBY _YG	STBY _ZG	CLK_SEL			00h

Description:
This register is used to manage the power control, select the clock source, and to issue a master reset to the device.

Setting the *SLEEP* bit in the register puts the device into very low power sleep mode. In this mode, only the serial interface and internal registers remain active, allowing for a very low standby current. Clearing this bit puts the device back into normal mode. To save power, the individual standby selections for each of the gyros should be used if any gyro axis is not used by the application.

The *CLK_SEL* setting determines the device clock source as follows:

CLK_SEL

CLK_SEL	Clock Source
0	Internal oscillator
1	PLL with X Gyro reference
2	PLL with Y Gyro reference
3	PLL with Z Gyro reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Reserved

On power up, the ITG-3200 defaults to the internal oscillator. It is highly recommended that the device is configured to use one of the gyros (or an external clock) as the clock reference, due to the improved stability.

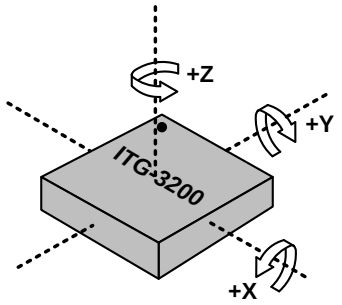
Parameters:

H_RESET	Reset device and internal registers to the power-up-default settings
SLEEP	Enable low power sleep mode
STBY_XG	Put gyro X in standby mode (1=standby, 0=normal)
STBY_YG	Put gyro Y in standby mode (1=standby, 0=normal)
STBY_ZG	Put gyro Z in standby mode (1=standby, 0=normal)
CLK_SEL	Select device clock source

9 Assembly

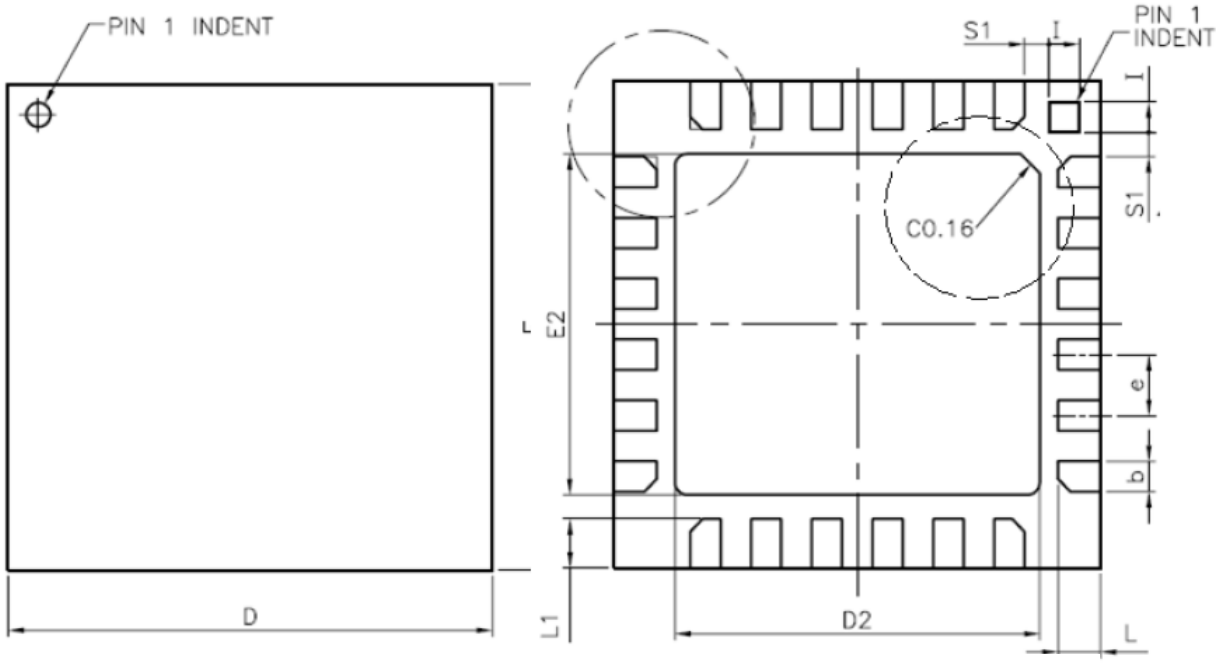
9.1 Orientation

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation.



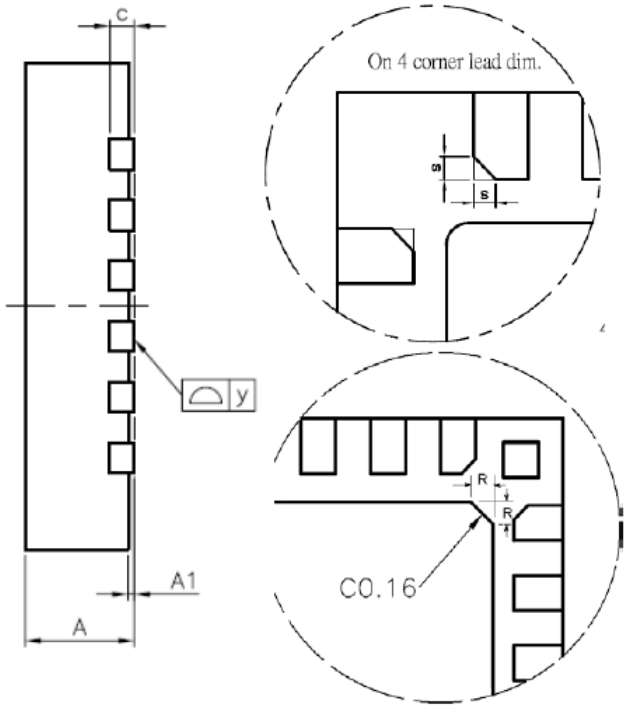
Orientation of Axes of Sensitivity and Polarity of Rotation

9.2 Package Dimensions



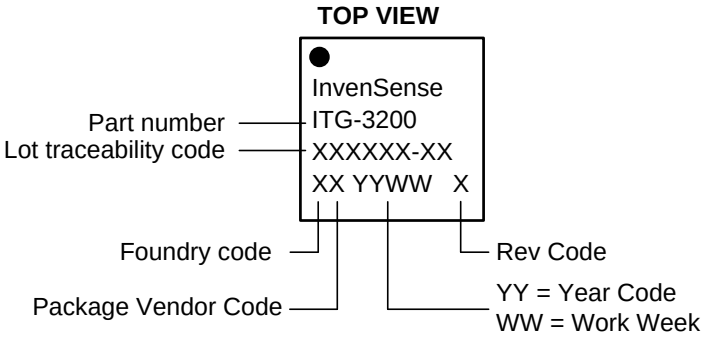
Top View **Bottom View**

SYMBOLS	DIMENSIONS IN MILLIMETERS		
	MIN	NOM	MAX
A	0.85	0.90	0.95
A1	0.00	0.02	0.05
b	0.18	0.25	0.30
c	---	0.20 REF.	---
D	3.90	4.00	4.10
D2	2.95	3.00	3.05
E	3.90	4.00	4.10
E2	2.75	2.80	2.85
e	---	0.50	---
L	0.30	0.35	0.40
L1	0.35	0.40	0.45
I	0.20	0.25	0.30
R	0.085	---	0.145
s	0.05	---	0.15
S1	0.15	0.20	0.25
y	0.00	---	0.075



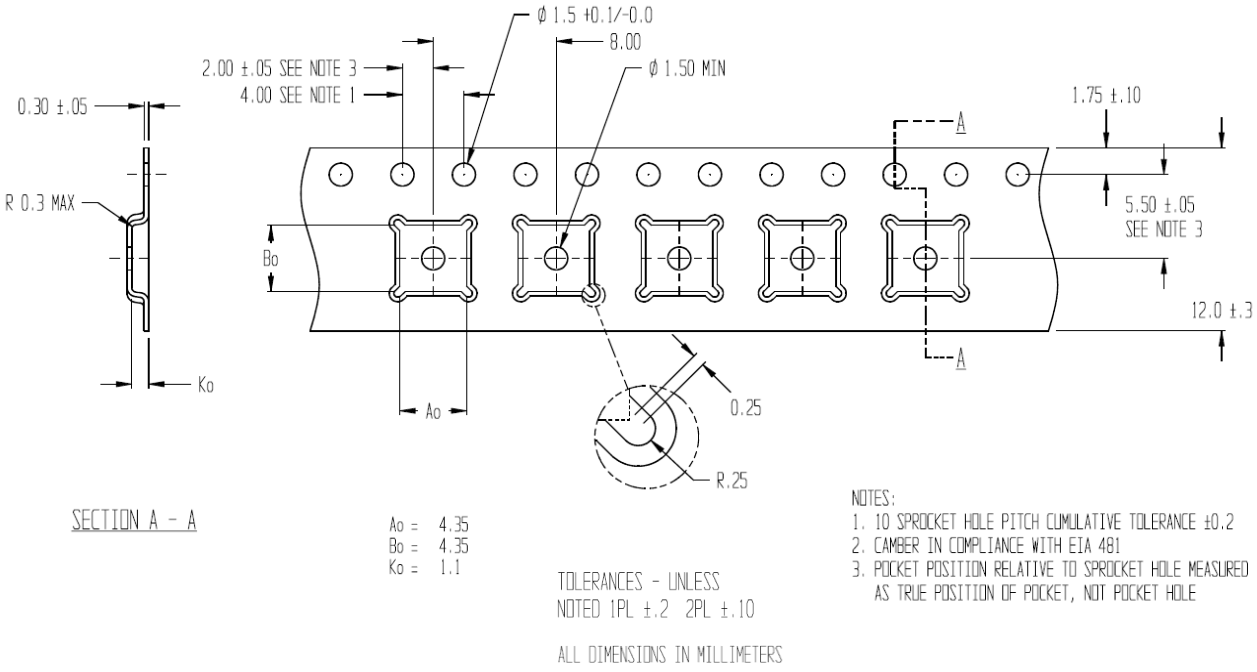
Package Dimensions

9.3 Package Marking Specification

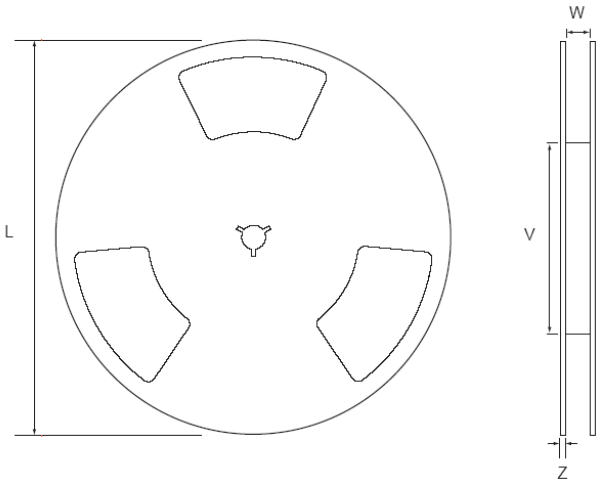


Package Marking Specification

9.4 Tape & Reel Specification



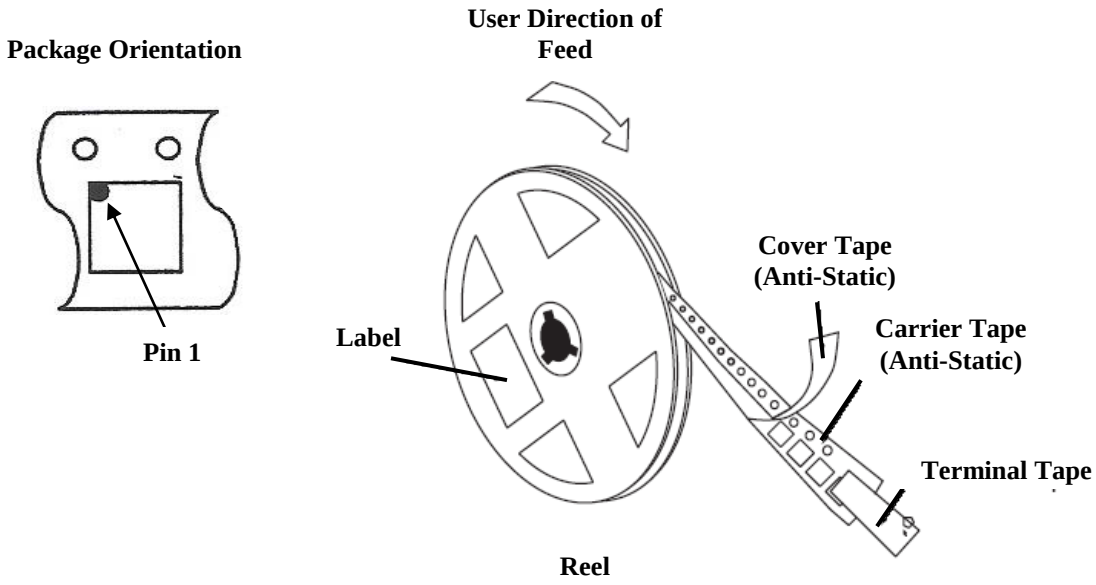
Tape Dimensions



Reel Outline Drawing

Reel Dimensions and Package Size

PKG SIZE	REEL (mm)			
	L	V	W	Z
4x4	330	100	13.2	2.2

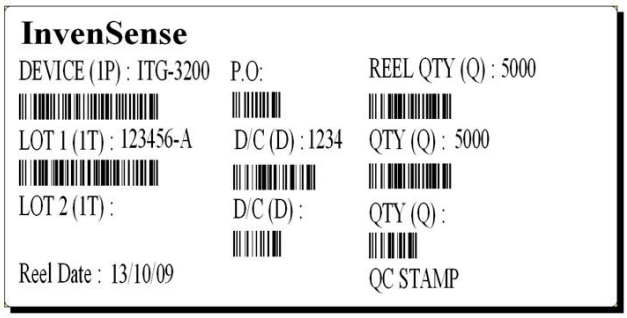


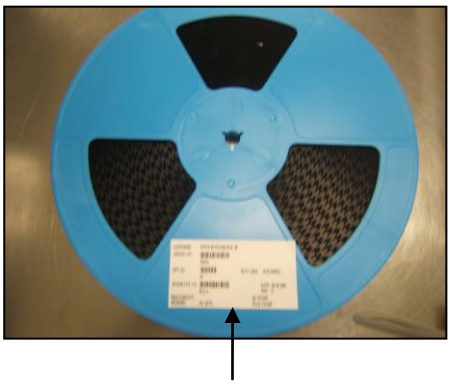
Tape and Reel Specification

Reel Specifications

Quantity Per Reel	5,000
Reels per Box	1
Boxes Per Carton (max)	3 full pizza boxes packed in the center of the carton, buffered by two empty pizza boxes (front and back).
Pcs/Carton (max)	15,000

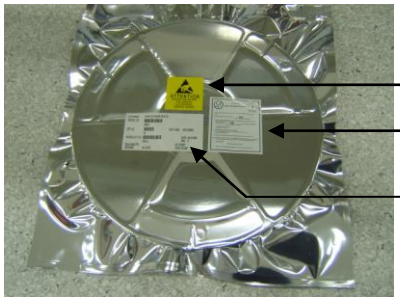
9.5 Label



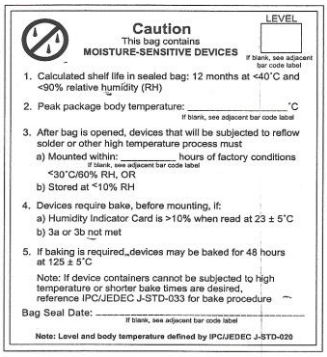


Location of Label

9.6 Packaging



Moisture Barrier Bag With Labels



Moisture-Sensitive Caution Label

Reel in Box

Box with Tape & Reel Label

9.7 Soldering Exposed Die Pad

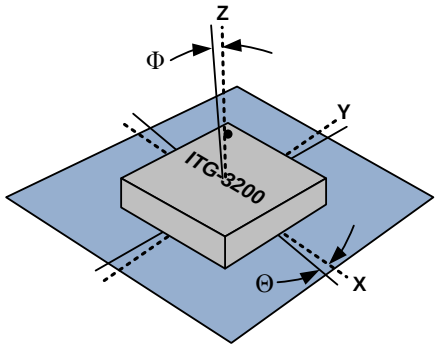
The ITG-3200 has very low active and standby current consumption. The exposed die pad is not required for heat sinking, and should not be soldered to the PCB since soldering to it contributes to performance changes due to package thermo-mechanical stress.

9.8 Component Placement

Testing indicates that there are no specific design considerations other than generally accepted industry design practices for component placement near the ITG-3200 multi-axis gyroscope to prevent noise coupling, and thermo-mechanical stress.

9.9 PCB Mounting and Cross-Axis Sensitivity

Orientation errors of the gyroscope mounted to the printed circuit board can cause cross-axis sensitivity in which one gyro responds to rotation about another axis, for example, the X-axis gyroscope responding to rotation about the Y or Z axes. The orientation mounting errors are illustrated in the figure below.



Package Gyro Axes (·····) Relative to PCB Axes (——) with Orientation Errors (Θ and Φ)

The table below shows the cross-axis sensitivity as a percentage of the specified gyroscope's sensitivity for a given orientation error.

Cross-Axis Sensitivity vs. Orientation Error

Orientation Error (θ or Φ)	Cross-Axis Sensitivity (sinθ or sinΦ)
0º	0%
0.5º	0.87%
1º	1.75%

The specification for cross-axis sensitivity in Section 3 includes the effect of the die orientation error with respect to the package.

9.10 MEMS Handling Instructions

MEMS (Micro Electro-Mechanical Systems) are a time-proven, robust technology used in hundreds of millions of consumer, automotive and industrial products. MEMS devices consist of microscopic moving mechanical structures. They differ from conventional IC products even though they can be found in similar packages. Therefore, MEMS devices require different handling precautions than conventional ICs prior to mounting onto printed circuit boards (PCBs).

The ITG-3200 gyroscope has a shock tolerance of 10,000*g*. InvenSense packages its gyroscopes as it deems proper for protection against normal handling and shipping. It recommends the following handling precautions to prevent potential damage.

- Individually packaged or trays of gyroscopes should not be dropped onto hard surfaces. Components placed in trays could be subject to *g*-forces in excess of 10,000*g* if dropped.
- Printed circuit boards that incorporate mounted gyroscopes should not be separated by manually snapping apart. This could also create *g*-forces in excess of 10,000*g*.

9.11 Gyroscope Surface Mount Guidelines

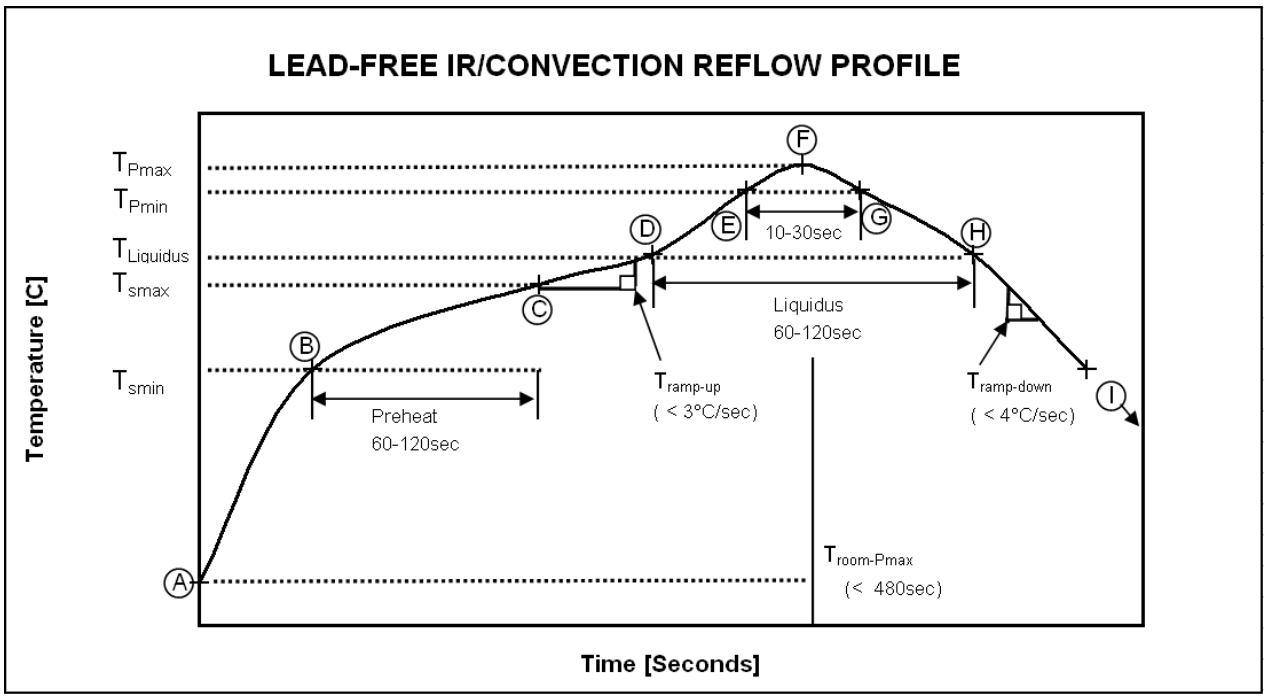
Any material used in the surface mount assembly process of the MEMS gyroscope should be free of restricted RoHS elements or compounds. Pb-free solders should be used for assembly.

In order to assure gyroscope performance, several industry standard guidelines need to be considered for surface mounting. These guidelines are for both printed circuit board (PCB) design and surface mount assembly and are available from packaging and assembly houses.

When using MEMS gyroscope components in plastic packages, package stress due to PCB mounting and assembly could affect the output offset and its value over a wide range of temperatures. This is caused by the mismatch between the Coefficient Temperature Expansion (CTE) of the package material and the PCB. Care must be taken to avoid package stress due to mounting.

9.12 Reflow Specification

The approved solder reflow curve shown in the figure below conforms to IPC/JEDEC J-STD-020D.01 (Moisture/Reflow Sensitivity Classification for Nonhermetic Solid State Surface Mount Devices) with a maximum peak temperature (Tc = 260°C). This is specified for component-supplier reliability qualification testing using lead-free solder for package thicknesses less than 1.6 mm. The reliability qualification pre-conditioning used by InvenSense incorporates three of these conforming reflow cycles. All temperatures refer to the topside of the QFN package, as measured on the package body surface. Customer solder-reflow processes should use the solder manufacturer's recommendations, making sure to never exceed the constraints listed in the table and figure below, as these represent the maximum tolerable ratings for the device. For optimum results, production solder reflow processes should use lower temperatures, reduced exposure times to high temperatures, and lower ramp-up and ramp-down rates than those listed below.



Approved IR/Convection Solder Reflow Curve

Temperature Set Points for IR / Convection Reflow Corresponding to Figure Above

Step	Setting	CONSTRAINTS		
		Temp (°C)	Time (sec)	Rate (°C/sec)
A	T_{room}	25		
B	T_{Smin}	150		
C	T_{Smax}	200	$60 < t_{BC} < 120$	
D	$T_{Liquidus}$	217		$r_{(TLiquidus-TPmax)} < 3$
E	T_{Pmin} [< TPmax-5°C, 250°C]	255		$r_{(TLiquidus-TPmax)} < 3$
F	T_{Pmax} [< TPmax, 260°C]	260	$t_{AF} < 480$	$r_{(TLiquidus-TPmax)} < 3$
G	T_{Pmin} [< TPmax-5°C, 250°C]	255	$t_{EG} < 30$	$r_{(TPmax-TLiquidus)} < 4$
H	$T_{Liquidus}$	217	$60 < t_{DH} < 120$	
I	T_{room}	25		

9.13 Storage Specifications

The storage specification of the ITG-3200 gyroscope conforms to IPC/JEDEC J-STD-020C Moisture Sensitivity Level (MSL) 3.

Storage Specifications for ITG-3200

Calculated shelf-life in moisture-sealed bag	12 months -- Storage conditions: <40°C and <90% RH
After opening moisture-sealed bag	168 hours -- Storage conditions: ambient ≤30°C at 60% RH

10 Reliability

10.1 Qualification Test Policy

InvenSense's products complete a Qualification Test Plan before being released to production. The Qualification Test Plan follows the JEDEC 47D Standards, "Stress-Test-Driven Qualification of Integrated Circuits," with the individual tests described below.

10.2 Qualification Test Plan

Accelerated Life Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
High Temperature Operating Life (HTOL/LFR)	JEDEC JESD22-A108C, Dynamic, 3.63V biased, Tj>125°C [read-points 168, 500, 1000 hours]	3	77	(0/1)
Steady-State Temperature Humidity Bias Life (1)	JEDEC JESD22-A101C, 85°C/85%RH [read-points 168, 500 hours], Information Only 1000 hours]	3	77	(0/1)
High Temperature Storage Life	JEDEC JESD22-A103C, Cond. A, 125°C Non-Bias Bake [read-points 168, 500, 1000 hours]	3	77	(0/1)

Device Component Level Tests

TEST	Method/Condition	Lot Quantity	Sample / Lot	Acc / Reject Criteria
ESD-HBM	JEDEC JESD22-A114F, Class 2 (1.5KV)	1	3	(0/1)
ESD-MM	JEDEC JESD22-A115-A, Class B (200V)	1	3	(0/1)
Latch Up	JEDEC JESD78B Level 2, 125C, +/- 100mA	1	6	(0/1)
Mechanical Shock	JEDEC JESD22-B104C, Mil-Std-883, method 2002, Cond. D, 10,000*g's*, 0.3ms, ±X,Y,Z – 6 directions, 5 times/direction	3	5	(0/1)
Vibration	JEDEC JESD22-B103B, Variable Frequency (random), Cond. B, 5-500Hz, X,Y,Z – 4 times/direction	3	5	(0/1)
Temperature Cycling (1)	JEDEC JESD22-A104D Condition N, -40°C to +85°C, Soak Mode 2, 100 cycles	3	77	(0/1)

Board Level Tests

TEST	Method/Condition/	Lot Quantity	Sample / Lot	Acc / Reject Criteria
Board Mechanical Shock	JEDEC JESD22-B104C,Mil-Std-883, method 2002, Cond. D, 10,000*g's*, 0.3ms, +-X,Y,Z – 6 directions, 5 times/direction	1	5	(0/1)
Board T/C	JEDEC JESD22-A104D Condition N, -40°C to +85°C, Soak Mode 2, 100 cycles	1	40	(0/1)

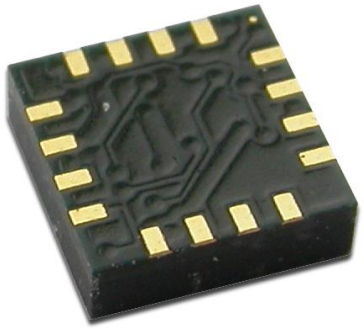
(1) – Tests are preceded by MSL3 Preconditioning in accordance with JEDEC JESD22-A113F

3-Axis Digital Compass IC HMC5883L

Honeywell

Advanced Information

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometry. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I^2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.



The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

FEATURES	BENEFITS
3-Axis Magnetoresistive Sensors and ASIC in a 3.0x3.0x0.9mm LCC Surface Mount Package	Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly
12-Bit ADC Coupled with Low Noise AMR Sensors Achieves 2 milli-gauss Field Resolution in ±8 Gauss Fields	Enables 1° to 2° Degree Compass Heading Accuracy
Built-In Self Test	Enables Low-Cost Functionality Test after Assembly in Production
Low Voltage Operations (2.16 to 3.6V) and Low Power Consumption (100 µA)	Compatible for Battery Powered Applications
Built-In Strap Drive Circuits	Set/Reset and Offset Strap Drivers for Degaussing, Self Test, and Offset Compensation
I^2C Digital Interface	Popular Two-Wire Serial Data Interface for Consumer Electronics
Lead Free Package Construction	RoHS Compliance
Wide Magnetic Field Range (+/-8 Oe)	Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy
Software and Algorithm Support Available	Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available
Fast 160 Hz Maximum Output Rate	Enables Pedestrian Navigation and LBS Applications

SPECIFICATIONS (* Tested at 25°C except stated otherwise.)

Characteristics	Conditions*	Min	Typ	Max	Units
Power Supply					
Supply Voltage	VDD Referenced to AGND VDDIO Referenced to DGND	2.16 1.71	2.5 1.8	3.6 VDD+0.1	Volts Volts
Average Current Draw	Idle Mode Measurement Mode (7.5 Hz ODR; No measurement average, MA1:MA0 = 00) VDD = 2.5V, VDDIO = 1.8V (Dual Supply) VDD = VDDIO = 2.5V (Single Supply)	- -	2 100	- -	µA µA
Performance					
Field Range	Full scale (FS)	-8		+8	gauss
Mag Dynamic Range	3-bit gain control	±1		±8	gauss
Sensitivity (Gain)	VDD=3.0V, GN=0 to 7, 12-bit ADC	230		1370	LSb/gauss
Digital Resolution	VDD=3.0V, GN=0 to 7, 1-LSb, 12-bit ADC	0.73		4.35	milli-gauss
Noise Floor (Field Resolution)	VDD=3.0V, GN=0, No measurement average, Standard Deviation 100 samples (See typical performance graphs below)		2		milli-gauss
Linearity	±2.0 gauss input range			0.1	±% FS
Hysteresis	±2.0 gauss input range		±25		ppm
Cross-Axis Sensitivity	Test Conditions: Cross field = 0.5 gauss, Happlied = ±3 gauss		±0.2%		%FS/gauss
Output Rate (ODR)	Continuous Measurment Mode Single Measurement Mode	0.75		75 160	Hz Hz
Measurement Period	From receiving command to data ready		6		ms
Turn-on Time	Ready for I2C commands Analog Circuit Ready for Measurements		200 50		µs ms
Gain Tolerance	All gain/dynamic range settings		±5		%
I^2C Address	8-bit read address 8-bit write address		0x3D 0x3C		hex hex
I^2C Rate	Controlled by I^2C Master			400	kHz
I^2C Hysteresis	Hysteresis of Schmitt trigger inputs on SCL and SDA - Fall (VDDIO=1.8V) Rise (VDDIO=1.8V)		0.2*VDDIO 0.8*VDDIO		Volts Volts
Self Test	X & Y Axes Z Axis		±1.16 ±1.08		gauss
	X & Y & Z Axes (GN=5) Positive Bias X & Y & Z Axes (GN=5) Negative Bias	243 -575		575 -243	LSb
Sensitivity Tempco	T_A = -40 to 125°C, Uncompensated Output		-0.3		%/°C
General					
ESD Voltage	Human Body Model (all pins) Charged Device Model (all pins)			2000 750	Volts
Operating Temperature	Ambient	-30		85	°C
Storage Temperature	Ambient, unbiased	-40		125	°C

Characteristics	Conditions*	Min	Typ	Max	Units
Reflow Classification	MSL 3, 260 °C Peak Temperature				
Package Size	Length and Width	2.85	3.00	3.15	mm
Package Height		0.8	0.9	1.0	mm
Package Weight			18		mg

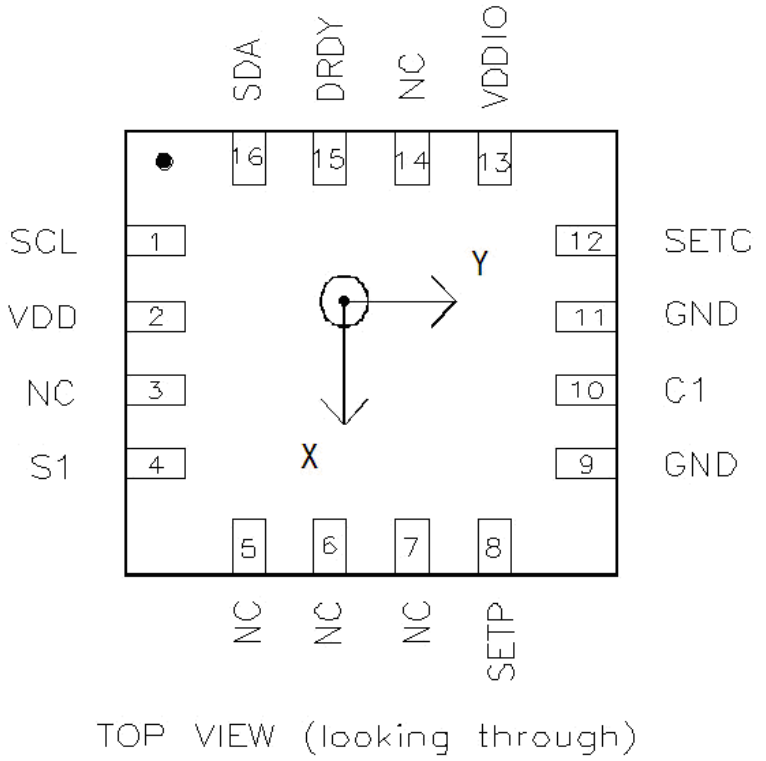
Absolute Maximum Ratings (* Tested at 25°C except stated otherwise.)

Characteristics	Min	Max	Units
Supply Voltage VDD	-0.3	4.8	Volts
Supply Voltage VDDIO	-0.3	4.8	Volts

PIN CONFIGURATIONS

Pin	Name	Description
1	SCL	Serial Clock – I^2C Master/Slave Clock
2	VDD	Power Supply (2.16V to 3.6V)
3	NC	Not to be Connected
4	S1	Tie to VDDIO
5	NC	Not to be Connected
6	NC	Not to be Connected
7	NC	Not to be Connected
8	SETP	Set/Reset Strap Positive – S/R Capacitor (C2) Connection
9	GND	Supply Ground
10	C1	Reservoir Capacitor (C1) Connection
11	GND	Supply Ground
12	SETC	S/R Capacitor (C2) Connection – Driver Side
13	VDDIO	IO Power Supply (1.71V to VDD)
14	NC	Not to be Connected
15	DRDY	Data Ready, Interrupt Pin. Internally pulled high. Optional connection. Low for 250 µsec when data is placed in the data output registers.
16	SDA	Serial Data – I^2C Master/Slave Data

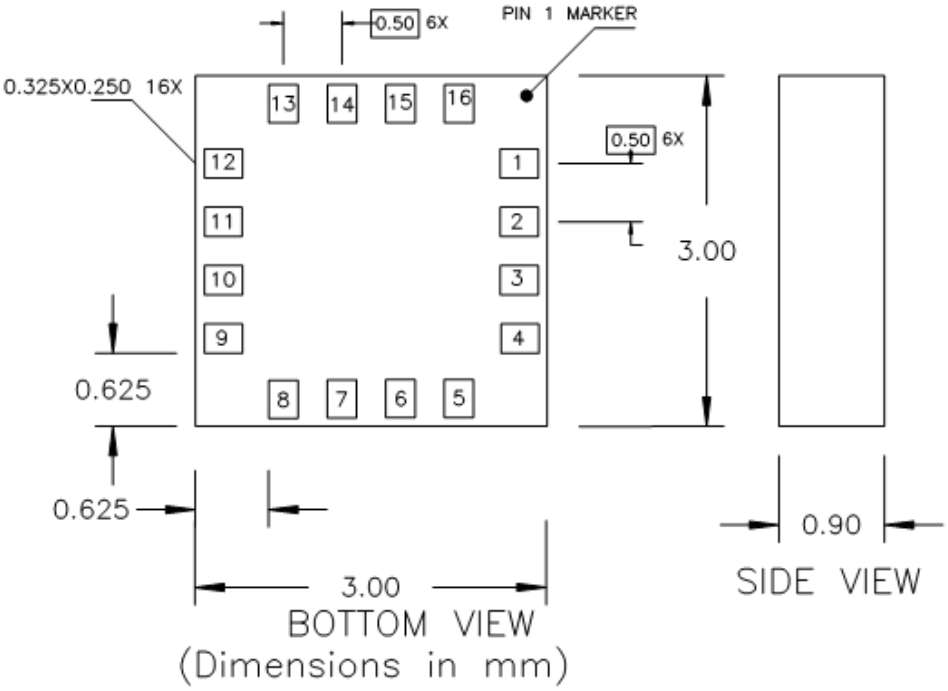
Table 1: Pin Configurations



Arrow indicates direction of magnetic field that generates a positive output reading in Normal Measurement configuration.

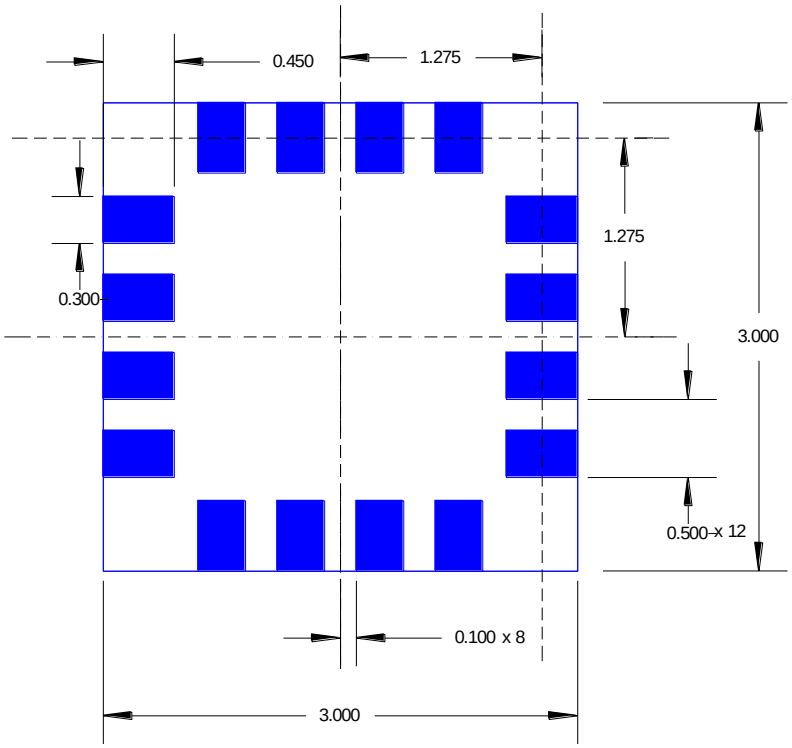
PACKAGE OUTLINES

PACKAGE DRAWING HMC5883L (16-PIN LPCC, dimensions in millimeters)



MOUNTING CONSIDERATIONS

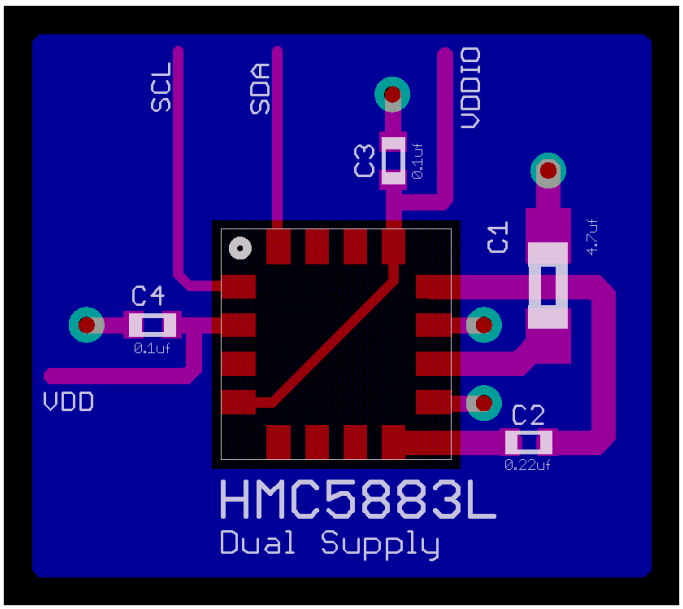
The following is the recommend printed circuit board (PCB) footprint for the HMC5883L.

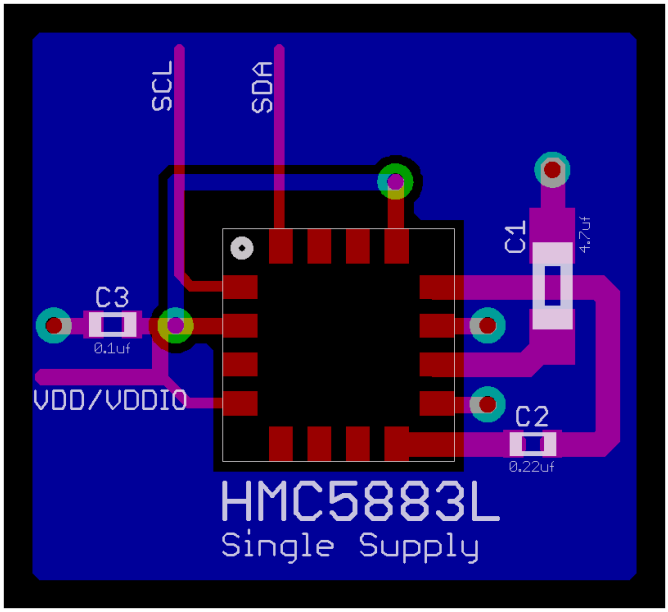


HMC5883 Land Pad Pattern
(All dimensions are in mm)

LAYOUT CONSIDERATIONS

Besides keeping all components that may contain ferrous materials (nickel, etc.) away from the sensor on both sides of the PCB, it is also recommended that there is no conducting copper under/near the sensor in any of the PCB layers. See recommended layout below. Notice that the one trace under the sensor in the dual supply mode is not expected to carry active current since it is for pin 4 pull-up to VDDIO. Power and ground planes are removed under the sensor to minimize possible source of magnetic noise. For best results, use non-ferrous materials for all exposed copper coding.





PCB Pad Definition and Traces

The HMC5883L is a fine pitch LCC package. Refer to previous figure for recommended PCB footprint for proper package centering. Size the traces between the HMC5883L and the external capacitors (C1 and C2) to handle the 1 ampere peak current pulses with low voltage drop on the traces.

Stencil Design and Solder Paste

A 4 mil stencil and 100% paste coverage is recommended for the electrical contact pads.

Reflow Assembly

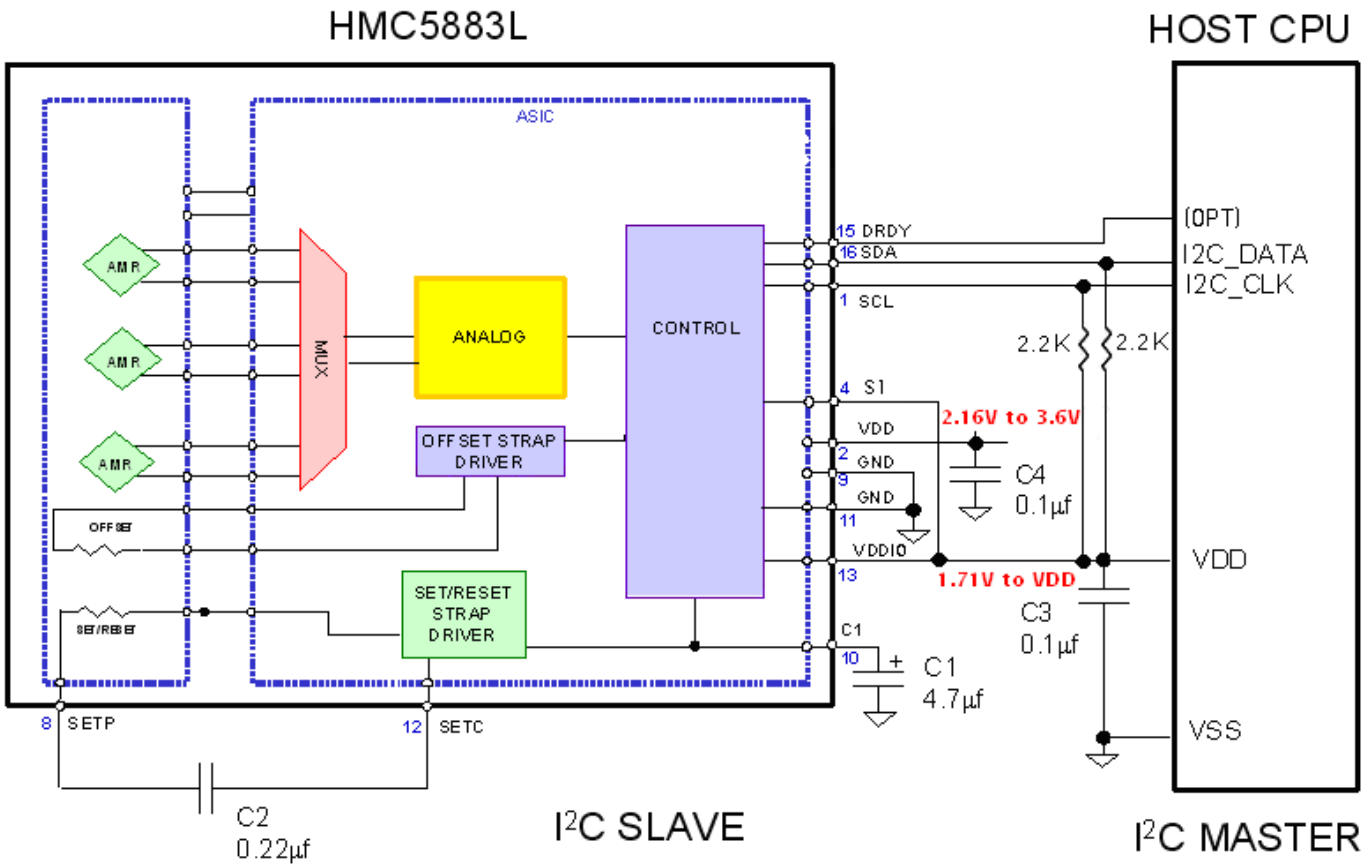
This device is classified as MSL 3 with 260°C peak reflow temperature. A baking process (125°C, 24 hrs) is required if device is not kept continuously in a dry (< 10% RH) environment before assembly. No special reflow profile is required for HMC5883L, which is compatible with lead eutectic and lead-free solder paste reflow profiles. Honeywell recommends adherence to solder paste manufacturer's guidelines. Hand soldering is not recommended. Built-in self test can be used to verify device functionalities after assembly.

External Capacitors

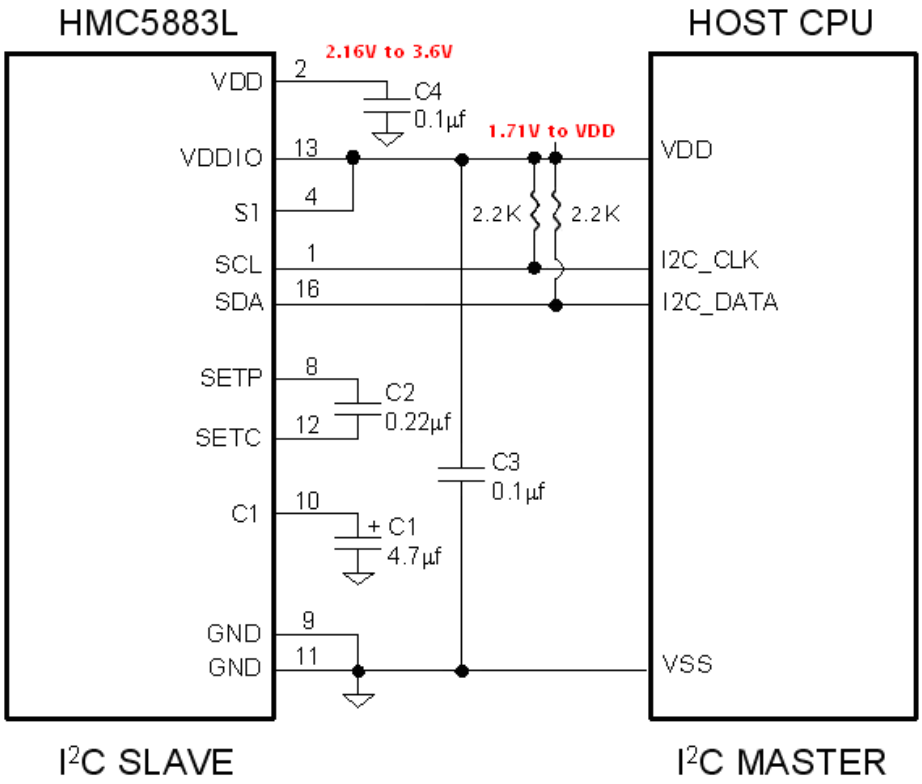
The two external capacitors should be ceramic type construction with low ESR characteristics. The exact ESR values are not critical but values less than 200 milli-ohms are recommended. Reservoir capacitor C1 is nominally 4.7 µF in capacitance, with the set/reset capacitor C2 nominally 0.22 µF in capacitance. Low ESR characteristics may not be in many small SMT ceramic capacitors (0402), so be prepared to up-size the capacitors to gain Low ESR characteristics.

INTERNAL SCHEMATIC DIAGRAM

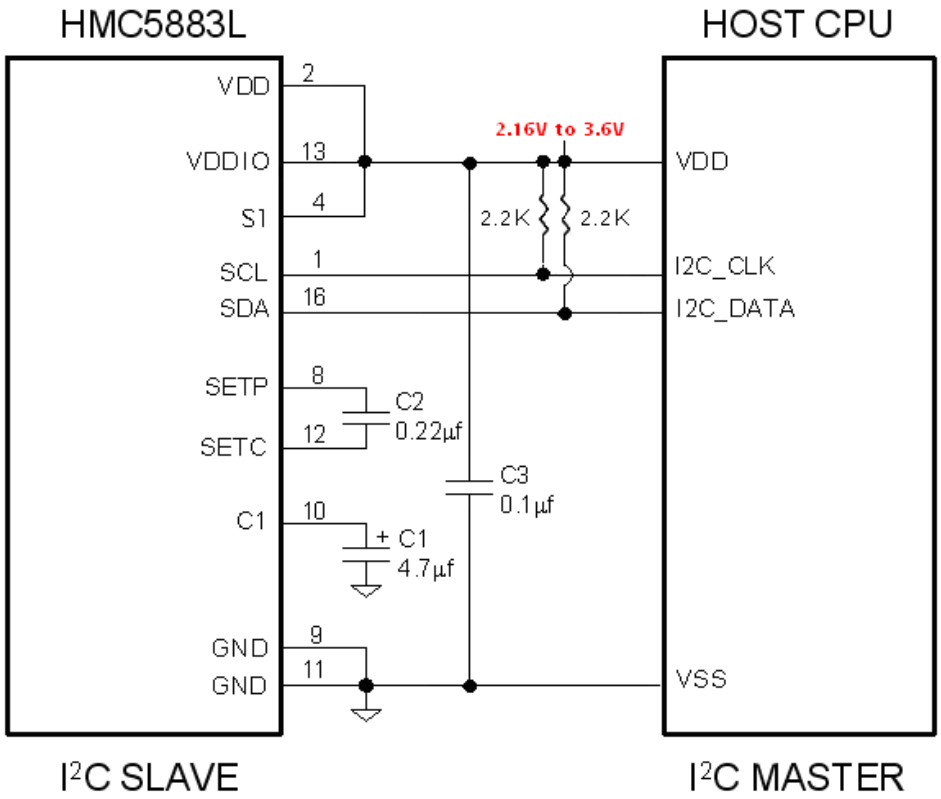
HMC5883L



DUAL SUPPLY REFERENCE DESIGN



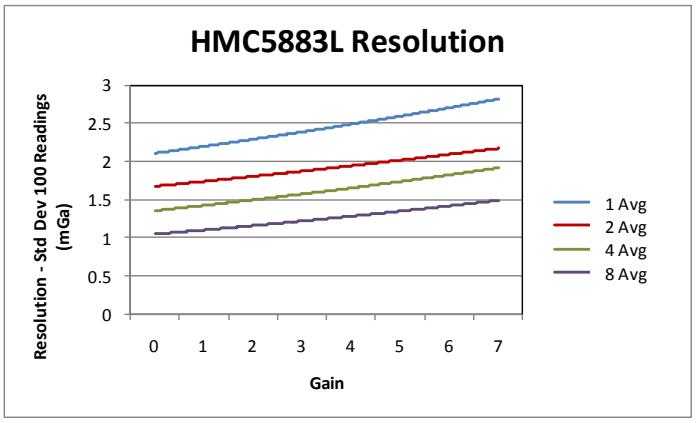
SINGLE SUPPLY REFERENCE DESIGN

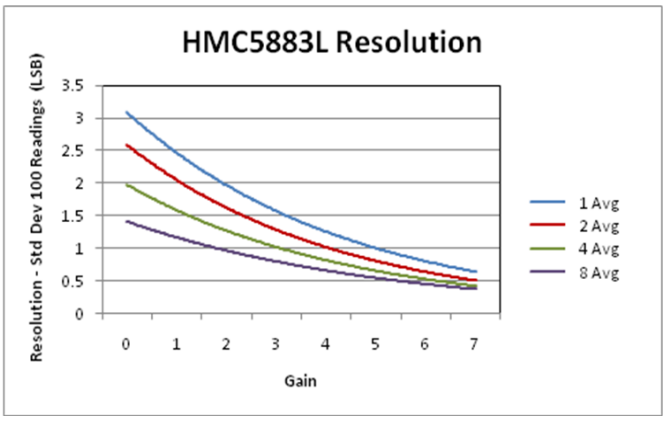


PERFORMANCE

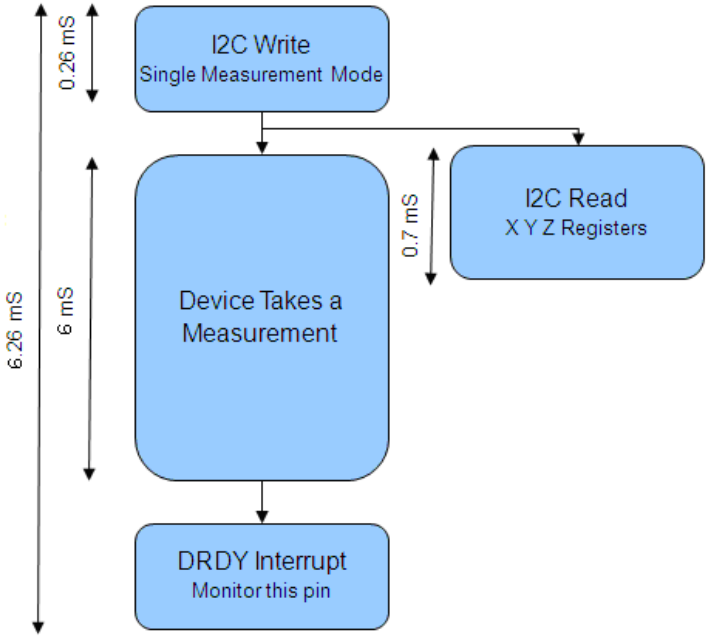
The following graph(s) highlight HMC5883L's performance.

Typical Noise Floor (Field Resolution)





Typical Measurement Period in Single-Measurement Mode



** Monitoring of the DRDY Interrupt pin is only required if maximum output rate is desired.*

BASIC DEVICE OPERATION

Anisotropic Magneto-Resistive Sensors

The Honeywell HMC5883L magnetoresistive sensor circuit is a trio of sensors and application specific support circuits to measure magnetic fields. With power supply applied, the sensor converts any incident magnetic field in the sensitive axis directions to a differential voltage output. The magnetoresistive sensors are made of a nickel-iron (Permalloy) thin-film and patterned as a resistive strip element. In the presence of a magnetic field, a change in the bridge resistive elements causes a corresponding change in voltage across the bridge outputs.

These resistive elements are aligned together to have a common sensitive axis (indicated by arrows in the pinout diagram) that will provide positive voltage change with magnetic fields increasing in the sensitive direction. Because the output is only proportional to the magnetic field component along its axis, additional sensor bridges are placed at orthogonal directions to permit accurate measurement of magnetic field in any orientation.

Self Test

To check the HMC5883L for proper operation, a self test feature in incorporated in which the sensor is internally excited with a nominal magnetic field (in either positive or negative bias configuration). This field is then measured and reported. This function is enabled and the polarity is set by bits MS[n] in the configuration register A. An internal current source generates DC current (about 10 mA) from the VDD supply. This DC current is applied to the offset straps of the magneto-resistive sensor, which creates an artificial magnetic field bias on the sensor. The difference of this measurement and the measurement of the ambient field will be put in the data output register for each of the three axes. By using this built-in function, the manufacturer can quickly verify the sensor's full functionality after the assembly without additional test setup. The self test results can also be used to estimate/compensate the sensor's sensitivity drift due to temperature.

For each "self test measurement", the ASIC:

1. Sends a "Set" pulse
2. Takes one measurement (M1)
3. Sends the (~10 mA) offset current to generate the (~1.1 Gauss) offset field and takes another measurement (M2)
4. Puts the difference of the two measurements in sensor's data output register:

   Output = [M2 – M1] (i.e. output = offset field only)

See SELF TEST OPERATION section later in this datasheet for additional details.

Power Management

This device has two different domains of power supply. The first one is VDD that is the power supply for internal operations and the second one is VDDIO that is dedicated to IO interface. It is possible to work with VDDIO equal to VDD; Single Supply mode, or with VDDIO lower than VDD allowing HMC5883L to be compatible with other devices on board.

I^2C Interface

Control of this device is carried out via the I^2C bus. This device will be connected to this bus as a slave device under the control of a master device, such as the processor.

This device is compliant with *I^2C-Bus Specification*, document number: 9398 393 40011. As an I^2C compatible device, this device has a 7-bit serial address and supports I^2C protocols. This device supports standard and fast modes, 100kHz and 400kHz, respectively, but does not support the high speed mode (Hs). External pull-up resistors are required to support these standard and fast speed modes.

Activities required by the master (register read and write) have priority over internal activities, such as the measurement. The purpose of this priority is to not keep the master waiting and the I^2C bus engaged for longer than necessary.

Internal Clock

The device has an internal clock for internal digital logic functions and timing management. This clock is not available to external usage.

H-Bridge for Set/Reset Strap Drive

The ASIC contains large switching FETs capable of delivering a large but brief pulse to the Set/Reset strap of the sensor. This strap is largely a resistive load. There is no need for an external Set/Reset circuit. The controlling of the Set/Reset function is done automatically by the ASIC for each measurement. One half of the difference from the measurements taken after a set pulse and after a reset pulse will be put in the data output register for each of the three axes. By doing so, the sensor's internal offset and its temperature dependence is removed/cancelled for all measurements. The set/reset pulses also effectively remove the past magnetic history (magnetism) in the sensor, if any.

For each "measurement", the ASIC:

1. Sends a "Set" pulse
2. Takes one measurement (Mset)
3. Sends a "Reset" pulse
4. Takes another measurement (Mreset)
5. Puts the following result in sensor's data output register:

Output = [Mset – Mreset] / 2

Charge Current Limit

The current that reservoir capacitor (C1) can draw when charging is limited for both single supply and dual supply configurations. This prevents drawing down the supply voltage (VDD).

MODES OF OPERATION

This device has several operating modes whose primary purpose is power management and is controlled by the Mode Register. This section describes these modes.

Continuous-Measurement Mode

During continuous-measurement mode, the device continuously makes measurements, at user selectable rate, and places measured data in data output registers. Data can be re-read from the data output registers if necessary; however, if the master does not ensure that the data register is accessed before the completion of the next measurement, the data output registers are updated with the new measurement. To conserve current between measurements, the device is placed in a state similar to idle mode, but the Mode Register is not changed to Idle Mode. That is, MD[n] bits are unchanged. Settings in the Configuration Register A affect the data output rate (bits DO[n]), the measurement configuration (bits MS[n]), when in continuous-measurement mode. All registers maintain values while in continuous-measurement mode. The I^2C bus is enabled for use by other devices on the network in while continuous-measurement mode.

Single-Measurement Mode

This is the default power-up mode. During single-measurement mode, the device makes a single measurement and places the measured data in data output registers. After the measurement is complete and output data registers are updated, the device is placed in idle mode, and the Mode Register *is* changed to idle mode by setting MD[n] bits. Settings in the configuration register affect the measurement configuration (bits MS[n])when in single-measurement mode. All registers maintain values while in single-measurement mode. The I^2C bus is enabled for use by other devices on the network while in single-measurement mode.

Idle Mode

During this mode the device is accessible through the I^2C bus, but major sources of power consumption are disabled, such as, but not limited to, the ADC, the amplifier, and the sensor bias current. All registers maintain values while in idle mode. The I^2C bus is enabled for use by other devices on the network while in idle mode.

REGISTERS

This device is controlled and configured via a number of on-chip registers, which are described in this section. In the following descriptions, *set* implies a logic 1, and *reset* or *clear* implies a logic 0, unless stated otherwise.

Register List
The table below lists the registers and their access. All address locations are 8 bits.

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

Table2: Register List

Register Access

This section describes the process of reading from and writing to this device. The devices uses an address pointer to indicate which register location is to be read from or written to. These pointer locations are sent from the master to this slave device and succeed the 7-bit address (0x1E) plus 1 bit read/write identifier, i.e. 0x3D for read and 0x3C for write.

To minimize the communication between the master and this device, the address pointer updated automatically without master intervention. The register pointer will be incremented by 1 automatically after the current register has been read successfully.

The address pointer value itself cannot be read via the I^2C bus.
Any attempt to read an invalid address location returns 0's, and any write to an invalid address location or an undefined bit within a valid address location is ignored by this device.

To move the address pointer to a random register location, first issue a "write" to that register location with no data byte following the commend. For example, to move the address pointer to register 10, send 0x3C 0x0A.

Configuration Register A

The configuration register is used to configure the device for setting the data output rate and measurement configuration. CRA0 through CRA7 indicate bit locations, with *CRA* denoting the bits that are in the configuration register. CRA7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit.CRA default is 0x10.

CRA7	CRA6	CRA5	CRA4	CRA3	CRA2	CRA1	CRA0
(0)	MA1(0)	MA0(0)	DO2 (1)	DO1 (0)	DO0 (0)	MS1 (0)	MS0 (0)

Table 3: Configuration Register A

Location	Name	Description
CRA7	CRA7	Bit CRA7 is reserved for future function. Set to 0 when configuring CRA.
CRA6 to CRA5	MA1 to MA0	Select number of samples averaged (1 to 8) per measurement output. 00 = 1(Default); 01 = 2; 10 = 4; 11 = 8
CRA4 to CRA2	DO2 to DO0	Data Output Rate Bits. These bits set the rate at which data is written to all three data output registers.
CRA1 to CRA0	MS1 to MS0	Measurement Configuration Bits. These bits define the measurement flow of the device, specifically whether or not to incorporate an applied bias into the measurement.

Table 4: Configuration Register A Bit Designations

The Table below shows all selectable output rates in continuous measurement mode. All three channels shall be measured within a given output rate. Other output rates with maximum rate of 160 Hz can be achieved by monitoring DRDY interrupt pin in single measurement mode.

DO2	DO1	DO0	Typical Data Output Rate (Hz)
0	0	0	0.75
0	0	1	1.5
0	1	0	3
0	1	1	7.5
1	0	0	15 (Default)
1	0	1	30
1	1	0	75
1	1	1	Reserved

Table 5: Data Output Rates

MS1	MS0	Measurement Mode
0	0	Normal measurement configuration (Default). In normal measurement configuration the device follows normal measurement flow. The positive and negative pins of the resistive load are left floating and high impedance.
0	1	Positive bias configuration for X, Y, and Z axes. In this configuration, a positive current is forced across the resistive load for all three axes.
1	0	Negative bias configuration for X, Y and Z axes. In this configuration, a negative current is forced across the resistive load for all three axes..
1	1	This configuration is reserved.

Table 6: Measurement Modes

Configuration Register B

The configuration register B for setting the device gain. CRB0 through CRB7 indicate bit locations, with *CRB* denoting the bits that are in the configuration register. CRB7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit. CRB default is 0x20.

CRB7	CRB6	CRB5	CRB4	CRB3	CRB2	CRB1	CRB0
GN2 (0)	GN1 (0)	GN0 (1)	(0)	(0)	(0)	(0)	(0)

Table 7: Configuration B Register

Location	Name	Description
CRB7 to CRB5	GN2 to GN0	Gain Configuration Bits. These bits configure the gain for the device. The gain configuration is common for all channels.
CRB4 to CRB0	0	These bits must be cleared for correct operation.

Table 8: Configuration Register B Bit Designations

The table below shows nominal gain settings. Use the “Gain” column to convert counts to Gauss. The “Digital Resolution” column is the theoretical value in term of milli-Gauss per count (LSb) which is the inverse of the values in the “Gain” column. The effective resolution of the usable signal also depends on the noise floor of the system, i.e.

Effective Resolution = Max (Digital Resolution, Noise Floor)

Choose a lower gain value (higher GN#) when total field strength causes overflow in one of the data output registers (saturation). Note that the very first measurement after a gain change maintains the same gain as the previous setting. **The new gain setting is effective from the second measurement and on.**

GN2	GN1	GN0	Recommended Sensor Field Range	Gain (LSb/ Gauss)	Digital Resolution (mG/LSb)	Output Range
0	0	0	± 0.88 Ga	1370	0.73	0xF800–0x07FF (-2048–2047)
0	0	1	± 1.3 Ga	1090 (default)	0.92	0xF800–0x07FF (-2048–2047)
0	1	0	± 1.9 Ga	820	1.22	0xF800–0x07FF (-2048–2047)
0	1	1	± 2.5 Ga	660	1.52	0xF800–0x07FF (-2048–2047)
1	0	0	± 4.0 Ga	440	2.27	0xF800–0x07FF (-2048–2047)
1	0	1	± 4.7 Ga	390	2.56	0xF800–0x07FF (-2048–2047)
1	1	0	± 5.6 Ga	330	3.03	0xF800–0x07FF (-2048–2047)
1	1	1	± 8.1 Ga	230	4.35	0xF800–0x07FF (-2048–2047)

Table 9: Gain Settings

Mode Register

The mode register is an 8-bit register from which data can be read or to which data can be written. This register is used to select the operating mode of the device. MR0 through MR7 indicate bit locations, with *MR* denoting the bits that are in the mode register. MR7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit. Mode register default is 0x01.

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
HS(0)	(0)	(0)	(0)	(0)	(0)	MD1 (0)	MD0 (1)

Table 10: Mode Register

Location	Name	Description
MR7 to MR2	HS	Set this pin to enable High Speed I2C, 3400kHz.
MR1 to MR0	MD1 to MD0	Mode Select Bits. These bits select the operation mode of this device.

Table 11: Mode Register Bit Designations

MD1	MD0	Operating Mode
0	0	Continuous-Measurement Mode. In continuous-measurement mode, the device continuously performs measurements and places the result in the data register. RDY goes high when new data is placed in all three registers. After a power-on or a write to the mode or configuration register, the first measurement set is available from all three data output registers after a period of $2/f_{DO}$ and subsequent measurements are available at a frequency of f_{DO}, where f_{DO} is the frequency of data output.
0	1	Single-Measurement Mode (Default). When single-measurement mode is selected, device performs a single measurement, sets RDY high and returned to idle mode. Mode register returns to idle mode bit values. The measurement remains in the data output register and RDY remains high until the data output register is read or another measurement is performed.
1	0	Idle Mode. Device is placed in idle mode.
1	1	Idle Mode. Device is placed in idle mode.

Table 12: Operating Modes

Data Output X Registers A and B

The data output X registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel X. Data output X register A contains the MSB from the measurement result, and data output X register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DXRA0 through DXRA7 and DXRB0 through DXRB7 indicate bit locations, with *DXRA* and *DXRB* denoting the bits that are in the data output X registers. DXRA7 and DXRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DXRA7	**DXRA6**	**DXRA5**	**DXRA4**	**DXRA3**	**DXRA2**	**DXRA1**	**DXRA0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DXRB7	**DXRB6**	**DXRB5**	**DXRB4**	**DXRB3**	**DXRB2**	**DXRB1**	**DXRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 13: Data Output X Registers A and B

Data Output Y Registers A and B

The data output Y registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel Y. Data output Y register A contains the MSB from the measurement result, and data output Y register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DYRA0 through DYRA7 and DYRB0 through DYRB7 indicate bit locations, with *DYRA* and *DYRB* denoting the bits that are in the data output Y registers. DYRA7 and DYRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DYRA7	**DYRA6**	**DYRA5**	**DYRA4**	**DYRA3**	**DYRA2**	**DYRA1**	**DYRA0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DYRB7	**DYRB6**	**DYRB5**	**DYRB4**	**DYRB3**	**DYRB2**	**DYRB1**	**DYRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 14: Data Output Y Registers A and B

Data Output Z Registers A and B

The data output Z registers are two 8-bit registers, data output register A and data output register B. These registers store the measurement result from channel Z. Data output Z register A contains the MSB from the measurement result, and data output Z register B contains the LSB from the measurement result. The value stored in these two registers is a 16-bit value in 2's complement form, whose range is 0xF800 to 0x07FF. DZRA0 through DZRA7 and DZRB0 through DZRB7 indicate bit locations, with *DZRA* and *DZRB* denoting the bits that are in the data output Z registers. DZRA7 and DZRB7 denote the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

In the event the ADC reading overflows or underflows for the given channel, or if there is a math overflow during the bias measurement, this data register will contain the value -4096. This register value will clear when after the next valid measurement is made.

DZRA7	DZRA6	DZRA5	DZRA4	DZRA3	DZRA2	DZRA1	DZRA0
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
DZRB7	**DZRB6**	**DZRB5**	**DZRB4**	**DZRB3**	**DZRB2**	**DZRB1**	**DZRB0**
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

Table 15: Data Output Z Registers A and B

Data Output Register Operation

When one or more of the output registers are read, new data cannot be placed in any of the output data registers until all six data output registers are read. This requirement also impacts DRDY and RDY, which cannot be cleared until new data is placed in all the output registers.

Status Register

The status register is an 8-bit read-only register. This register is used to indicate device status. SR0 through SR7 indicate bit locations, with *SR* denoting the bits that are in the status register. SR7 denotes the first bit of the data stream.

SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
(0)	(0)	(0)	(0)	(0)	(0)	LOCK (0)	RDY(0)

Table 16: Status Register

Location	Name	Description
SR7 to SR2	0	These bits are reserved.
SR1	LOCK	Data output register lock. This bit is set when: 1.some but not all for of the six data output registers have been read, 2. Mode register has been read. When this bit is set, the six data output registers are locked and any new data will not be placed in these register until one of these conditions are met: 1.all six bytes have been read, 2. the mode register is changed, 3. the measurement configuration (CRA) is changed, 4. power is reset.
SR0	RDY	Ready Bit. Set when data is written to all six data registers. Cleared when device initiates a write to the data output registers and after one or more of the data output registers are written to. When RDY bit is clear it shall remain cleared for a 250 μs. DRDY pin can be used as an alternative to the status register for monitoring the device for measurement data.

Table 17: Status Register Bit Designations

Identification Register A

The identification register A is used to identify the device. IRA0 through IRA7 indicate bit locations, with *IRA* denoting the bits that are in the identification register A. IRA7 denotes the first bit of the data stream. The number in parenthesis indicates the default value of that bit.

The identification value for this device is stored in this register. This is a read-only register.
Register values. ASCII value *H*

IRA7	IRA6	IRA5	IRA4	IRA3	IRA2	IRA1	IRA0
0	1	0	0	1	0	0	0

Table 18: Identification Register A Default Values

Identification Register B

The identification register B is used to identify the device. IRB0 through IRB7 indicate bit locations, with *IRB* denoting the bits that are in the identification register A. IRB7 denotes the first bit of the data stream.

Register values. ASCII value *4*

IRB7	IRB6	IRB5	IRB4	IRB3	IRB2	IRB1	IRB0
0	0	1	1	0	1	0	0

Table 19: Identification Register B Default Values

Identification Register C

The identification register C is used to identify the device. IRC0 through IRC7 indicate bit locations, with *IRC* denoting the bits that are in the identification register A. IRC7 denotes the first bit of the data stream.

Register values. ASCII value *3*

IRC7	IRC6	IRC5	IRC4	IRC3	IRC2	IRC1	IRC0
0	0	1	1	0	0	1	1

Table 20: Identification Register C Default Values

I^{2}C COMMUNICATION PROTOCOL

The HMC5883L communicates via a two-wire I^{2}C bus system as a slave device. The HMC5883L uses a simple protocol with the interface protocol defined by the I^{2}C bus specification, and by this document. The data rate is at the standard-mode 100kbps or 400kbps rates as defined in the I^{2}C Bus Specifications. The bus bit format is an 8-bit Data/Address send and a 1-bit acknowledge bit. The format of the data bytes (payload) shall be case sensitive ASCII characters or binary data to the HMC5883L slave, and binary data returned. Negative binary values will be in two's complement form. The default (factory) HMC5883L 8-bit slave address is 0x3C for write operations, or 0x3D for read operations.

The HMC5883L Serial Clock (SCL) and Serial Data (SDA) lines require resistive pull-ups (Rp) between the master device (usually a host microprocessor) and the HMC5883L. Pull-up resistance values of about 2.2K to 10K ohms are recommended with a nominal VDDIO voltage. Other resistor values may be used as defined in the I^{2}C Bus Specifications that can be tied to VDDIO.

The SCL and SDA lines in this bus specification may be connected to multiple devices. The bus can be a single master to multiple slaves, or it can be a multiple master configuration. All data transfers are initiated by the master device, which is responsible for generating the clock signal, and the data transfers are 8 bit long. All devices are addressed by I^{2}C's unique 7-bit address. After each 8-bit transfer, the master device generates a 9th clock pulse, and releases the SDA line. The receiving device (addressed slave) will pull the SDA line low to acknowledge (ACK) the successful transfer or leave the SDA high to negative acknowledge (NACK).

Per the I^2C spec, all transitions in the SDA line must occur when SCL is low. This requirement leads to two unique conditions on the bus associated with the SDA transitions when SCL is high. Master device pulling the SDA line low while the SCL line is high indicates the Start (S) condition, and the Stop (P) condition is when the SDA line is pulled high while the SCL line is high. The I^2C protocol also allows for the Restart condition in which the master device issues a second start condition without issuing a stop.

All bus transactions begin with the master device issuing the start sequence followed by the slave address byte. The address byte contains the slave address; the upper 7 bits (bits7-1), and the Least Significant bit (LSb). The LSb of the address byte designates if the operation is a read (LSb=1) or a write (LSb=0). At the 9th clock pulse, the receiving slave device will issue the ACK (or NACK). Following these bus events, the master will send data bytes for a write operation, or the slave will clock out data with a read operation. All bus transactions are terminated with the master issuing a stop sequence.

I^2C bus control can be implemented with either hardware logic or in software. Typical hardware designs will release the SDA and SCL lines as appropriate to allow the slave device to manipulate these lines. In a software implementation, care must be taken to perform these tasks in code.

OPERATIONAL EXAMPLES

The HMC5883L has a fairly quick stabilization time from no voltage to stable and ready for data retrieval. The nominal 56 milli-seconds with the factory default single measurement mode means that the six bytes of magnetic data registers (DXRA, DXRB, DZRA, DZRB, DYRA, and DYRB) are filled with a valid first measurement.

To change the measurement mode to continuous measurement mode, after the power-up time send the three bytes:

0x3C 0x02 0x00

This writes the 00 into the second register or mode register to switch from single to continuous measurement mode setting. With the data rate at the factory default of 15Hz updates, a 67 milli-second typical delay should be allowed by the I^2C master before querying the HMC5883L data registers for new measurements. To clock out the new data, send:

0x3D, and clock out DXRA, DXRB, DZRA, DZRB, DYRA, and DYRB located in registers 3 through 8. The HMC5883L will automatically re-point back to register 3 for the next 0x3D query. All six data registers must be read properly before new data can be placed in any of these data registers.

Below is an example of a (power-on) initialization process for “continuous-measurement mode”:

1. Write CRA (00) – send **0x3C 0x00 0x70** (8-average, 15 Hz default, normal measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5, or any other desired gain)
3. Write Mode (02) – send **0x3C 0x02 0x00** (Continuous-measurement mode)
4. Wait 6 ms or monitor status register or DRDY hardware interrupt pin
5. Loop
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.
   Send **0x3C 0x03** (point to first data register 03)
   Wait about 67 ms (if 15 Hz rate) or monitor status register or DRDY hardware interrupt pin
   End_loop

Below is an example of a (power-on) initialization process for “single-measurement mode”:

1. Write CRA (00) – send **0x3C 0x00 0x70** (8-average, 15 Hz default or any other rate, normal measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5, or any other desired gain)
3. For each measurement query:
   Write Mode (02) – send **0x3C 0x02 0x01** (Single-measurement mode)
   Wait 6 ms or monitor status register or DRDY hardware interrupt pin
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.

SELF TEST OPERATION

To check the HMC5883L for proper operation, a self test feature in incorporated in which the sensor offset straps are excited to create a nominal field strength (bias field) to be measured. To implement self test, the least significant bits (MS1 and MS0) of configuration register A are changed from 00 to 01 (positive bias) or 10 (negetive bias).

Then, by placing the mode register into single or continuous-measurement mode, two data acquisition cycles will be made on each magnetic vector. The first acquisition will be a set pulse followed shortly by measurement data of the external field. The second acquisition will have the offset strap excited (about 10 mA) in the positive bias mode for X, Y, and Z axes to create about a 1.1 gauss self test field plus the external field. The first acquisition values will be subtracted from the second acquisition, and the net measurement will be placed into the data output registers.

Since self test adds ~1.1 Gauss additional field to the existing field strength, using a reduced gain setting prevents sensor from being saturated and data registers overflowed. For example, if the configuration register B is set to 0xA0 (Gain=5), values around +452 LSb (1.16 Ga * 390 LSb/Ga) will be placed in the X and Y data output registers and around +421 (1.08 Ga * 390 LSb/Ga) will be placed in Z data output register. To leave the self test mode, change MS1 and MS0 bit of the configuration register A back to 00 (Normal Measurement Mode). Acceptable limits of the self test values depend on the gain setting. Limits for Gain=5 is provided in the specification table.

Below is an example of a “positive self test” process using continuous-measurement mode:

1. Write CRA (00) – send **0x3C 0x00 0x71** (8-average, 15 Hz default, positive self test measurement)
2. Write CRB (01) – send **0x3C 0x01 0xA0** (Gain=5)
3. Write Mode (02) – send **0x3C 0x02 0x00** (Continuous-measurement mode)
4. Wait 6 ms or monitor status register or DRDY hardware interrupt pin
5. Loop
   Send **0x3D 0x06** (Read all 6 bytes. If gain is changed then this data set is using previous gain)
   Convert three 16-bit 2’s compliment hex values to decimal values and assign to X, Z, Y, respectively.
   Send **0x3C 0x03** (point to first data register 03)
   Wait about 67 ms (if 15 Hz rate) or monitor status register or DRDY hardware interrupt pin
   End_loop
6. Check limits –
   If all 3 axes (X, Y, and Z) are within reasonable limits (243 to 575 for Gain=5, adjust these limits basing on the gain setting used. See an example below.) Then
   All 3 axes pass positive self test
   Write CRA (00) – send **0x3C 0x00 0x70** (Exit self test mode and this procedure)
   Else
   If Gain<7
   Write CRB (01) – send **0x3C 0x01 0x_0** (Increase gain setting and retry, skip the next data set)
   Else
   At least one axis did not pass positive self test
   Write CRA (00) – send **0x3C 0x00 0x70** (Exit self test mode and this procedure)
   End If

Below is an example of how to adjust the “positive self” test limits basing on the gain setting:

1. If Gain = 6, self test limits are:
   Low Limit = 243 * 330/390 = 206
   High Limit = 575 * 330/390 = 487

2. If Gain = 7, self test limits are:
   Low Limit = 243 * 230/390 = 143
   High Limit = 575 * 230/390 = 339

SCALE FACTOR TEMPERATURE COMPENSATION

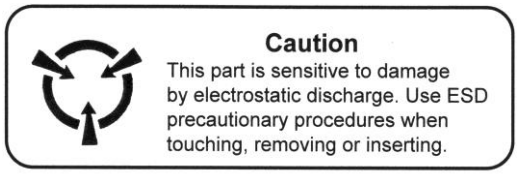
The built-in self test can also be used to periodically compensate the scaling errors due to temperature variations. A compensation factor can be found by comparing the self test outputs with the ones obtained at a known temperature. For example, if the self test output is 400 at room temperature and 300 at the current temperature then a compensation factor of (400/300) should be applied to all current magnetic readings. A temperature sensor is not required using this method.

Below is an example of a temperature compensation process using positive self test method:

1. If self test measurement at a temperature “when the last magnetic calibration was done”:
   X_STP = 400
   Y_STP = 410
   Z_STP = 420
2. If self test measurement at a different tmperature:
   X_STP = 300 (Lower than before)
   Y_STP = 310 (Lower than before)
   Z_STP = 320 (Lower than before)
   Then
   X_TempComp = 400/300
   Y_TempComp = 410/310
   Z_TempComp = 420/320
3. Applying to all new measurements:
   X = X * X_TempComp
   Y = Y * Y_TempComp
   Z = Z * Z_TempComp
   Now all 3 axes are temperature compensated, i.e. sensitivity is same as “when the last magnetic calibration was done”; therefore, the calibration coefficients can be applied without modification.
4. Repeat this process periodically or,for every Δt degrees of temperature change measured, if available.

ORDERING INFORMATION

Ordering Number	Product
HMC5883L-T	Cut Tape
HMC5883L-TR	Tape and Reel 4k pieces/reel



CAUTION: ESDS CAT. 1B

FIND OUT MORE

For more information on Honeywell’s Magnetic Sensors visit us online at www.magneticsensors.com or contact us at 1-800-323-8295 (763-954-2474 internationally).

Honeywell
12001 Highway 55
Plymouth, MN 55441
Tel: 800-323-8295
www.magneticsensors.com

Form # 900405 Rev E
February 2013

Honeywell

PCA9685

16-channel, 12-bit PWM Fm+ I²C-bus LED controller

Rev. 3 — 2 September 2010 Product data sheet

1. General description

The PCA9685 is an I²C-bus controlled 16-channel LED controller optimized for LCD Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 40 Hz to 1000 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency.

Each LED output can be off or on (no PWM control), or set at its individual PWM controller value. The LED output driver is programmed to be either open-drain with a 25 mA current sink capability at 5 V or totem pole with a 25 mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum amount of discrete components for larger current or higher voltage LEDs.

The PCA9685 is in the new Fast-mode Plus (Fm+) family. Fm+ devices offer higher frequency (up to 1 MHz) and more densely populated bus operation (up to 4000 pF).

Although the PCA9635 and PCA9685 have many similar features, the PCA9685 has some unique features that make it more suitable for applications such as LCD backlighting and Ambilight:

- The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels. This feature is not available in PCA9635.
- The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).
- When multiple LED controllers are incorporated in a system, the PWM pulse widths between multiple devices may differ if PCA9635s are used. The PCA9685 has a programmable prescaler to adjust the PWM pulse widths of multiple devices.
- The PCA9685 has an external clock input pin that will accept user-supplied clock (50 MHz max.) in place of the internal 25 MHz oscillator. This feature allows synchronization of multiple devices. The PCA9635 does not have external clock input feature.
- Like the PCA9635, PCA9685 also has a built-in oscillator for the PWM control. However, the frequency used for PWM control in the PCA9685 is adjustable from about 40 Hz to 1000 Hz as compared to the typical 97.6 kHz frequency of the PCA9635. This allows the use of PCA9685 with external power supply controllers. All bits are set at the same frequency.
- The Power-On Reset (POR) default state of LEDn output pins is LOW in the case of PCA9685. It is HIGH for PCA9635.

The active LOW Output Enable input pin ($\overline{OE}$) allows asynchronous control of the LED outputs and can be used to set all the outputs to a defined I²C-bus programmable logic state. The $\overline{OE}$ can also be used to externally 'pulse width modulate' the outputs, which is useful when multiple devices need to be dimmed or blinked together using software control.

Software programmable LED All Call and three Sub Call I²C-bus addresses allow all or defined groups of PCA9685 devices to respond to a common I²C-bus address, allowing for example, all red LEDs to be turned on or off at the same time or marquee chasing effect, thus minimizing I²C-bus commands. Six hardware address pins allow up to 62 devices on the same bus.

The Software Reset (SWRST) General Call allows the master to perform a reset of the PCA9685 through the I²C-bus, identical to the Power-On Reset (POR) that initializes the registers to their default state causing the outputs to be set LOW. This allows an easy and quick way to reconfigure all device registers to the same condition via software.

2. Features and benefits

- 16 LED drivers. Each output programmable at:
  - Off
  - On
  - Programmable LED brightness
  - Programmable LED turn-on time to help reduce EMI
- 1 MHz Fast-mode Plus compatible I²C-bus interface with 30 mA high drive capability on SDA output for driving high capacitive buses
- 4096-step (12-bit) linear programmable brightness per LED output varying from fully off (default) to maximum brightness
- LED output frequency (all LEDs) typically varies from 40 Hz to 1000 Hz (Default of 1Eh in PRE_SCALE register results in a 200 Hz refresh rate with oscillator clock of 25 MHz.)
- Sixteen totem pole outputs (sink 25 mA and source 10 mA at 5 V) with software programmable open-drain LED outputs selection (default at totem pole). No input function.
- Output state change programmable on the Acknowledge or the STOP Command to update outputs byte-by-byte or all at the same time (default to 'Change on STOP').
- Active LOW Output Enable ($\overline{OE}$) input pin. LEDn outputs programmable to logic 1, logic 0 (default at power-up) or 'high-impedance' when $\overline{OE}$ is HIGH.
- 6 hardware address pins allow 62 PCA9685 devices to be connected to the same I²C-bus
- Toggling $\overline{OE}$ allows for hardware LED blinking
- 4 software programmable I²C-bus addresses (one LED All Call address and three LED Sub Call addresses) allow groups of devices to be addressed at the same time in any combination (for example, one register used for 'All Call' so that all the PCA9685s on the I²C-bus can be addressed at the same time and the second register used for three different addresses so that $\frac{1}{3}$ of all devices on the bus can be addressed at the same time in a group). Software enable and disable for these I²C-bus address.
- Software Reset feature (SWRST General Call) allows the device to be reset through the I²C-bus

- 25 MHz typical internal oscillator requires no external components
- External 50 MHz (max.) clock input
- Internal power-on reset
- Noise filter on SDA/SCL inputs
- Edge rate control on outputs
- No output glitches on power-up
- Supports hot insertion
- Low standby current
- Operating power supply voltage range of 2.3 V to 5.5 V
- 5.5 V tolerant inputs
- −40 °C to +85 °C operation
- ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per JESD22-A115 and 1000 V CDM per JESD22-C101
- Latch-up testing is done to JEDEC Standard JESD78 which exceeds 100 mA
- Packages offered: TSSOP28, HVQFN28

3. Applications

- RGB or RGBA LED drivers
- LED status information
- LED displays
- LCD backlights
- Keypad backlights for cellular phones or handheld devices

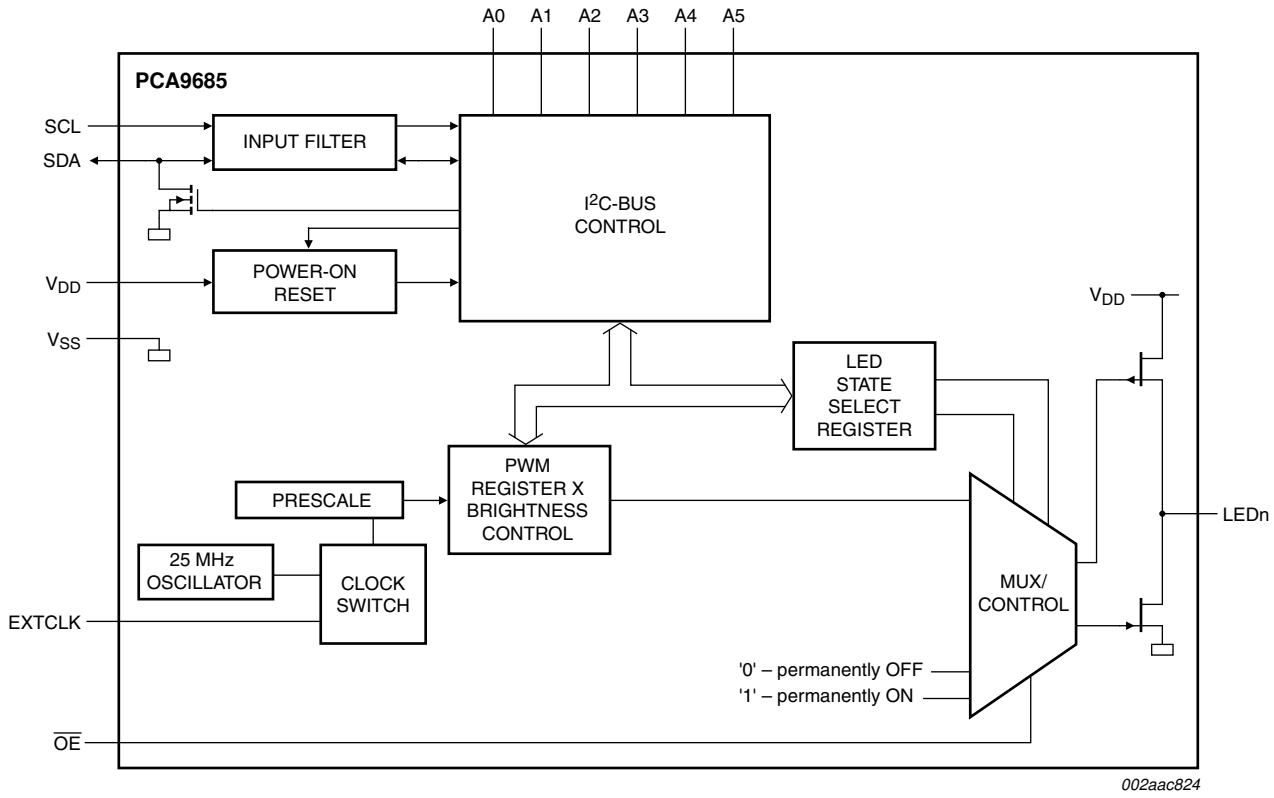
4. Ordering information

Table 1. Ordering information

Type number	Topside mark	Package		
		Name	Description	Version
PCA9685PW	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685PW/Q900[1]	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685BS	P9685	HVQFN28	plastic thermal enhanced very thin quad flat package; no leads; 28 terminals; body 6 × 6 × 0.85 mm	SOT788-1

[1] PCA9685PW/Q900 is AEC-Q100 compliant. Contact **i2c.support@nxp.com** for PPAP.

5. Block diagram



Remark: Only one LED output shown for clarity.

Fig 1. Block diagram of PCA9685

6. Pinning information

6.1 Pinning

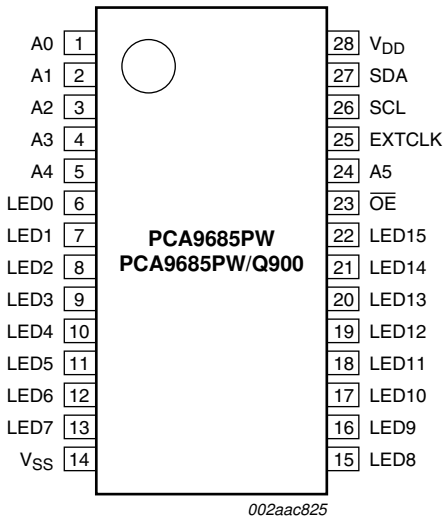


Fig 2. Pin configuration for TSSOP28

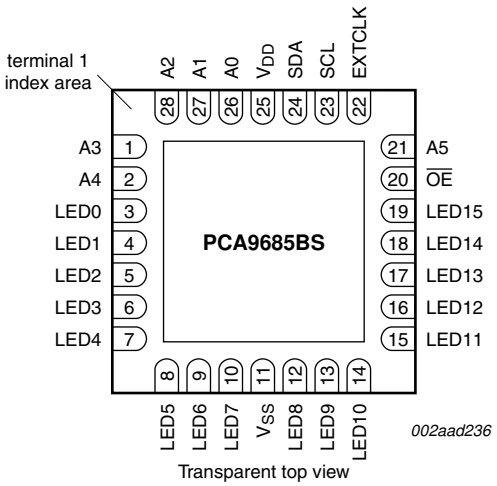


Fig 3. Pin configuration for HVQFN28

6.2 Pin description

Table 2. Pin description

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
A0	1	26	I	address input 0
A1	2	27	I	address input 1
A2	3	28	I	address input 2
A3	4	1	I	address input 3
A4	5	2	I	address input 4
LED0	6	3	O	LED driver 0
LED1	7	4	O	LED driver 1
LED2	8	5	O	LED driver 2
LED3	9	6	O	LED driver 3
LED4	10	7	O	LED driver 4
LED5	11	8	O	LED driver 5
LED6	12	9	O	LED driver 6
LED7	13	10	O	LED driver 7
V_{SS}	14	11[1]	power supply	supply ground
LED8	15	12	O	LED driver 8
LED9	16	13	O	LED driver 9
LED10	17	14	O	LED driver 10
LED11	18	15	O	LED driver 11

Table 2. Pin description *...continued*

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
LED12	19	16	O	LED driver 12
LED13	20	17	O	LED driver 13
LED14	21	18	O	LED driver 14
LED15	22	19	O	LED driver 15
$\overline{OE}$	23	20	I	active LOW output enable
A5	24	21	I	address input 5
EXTCLK	25	22	I	external clock input[2]
SCL	26	23	I	serial clock line
SDA	27	24	I/O	serial data line
V_{DD}	28	25	power supply	supply voltage

[1] HVQFN28 package die supply ground is connected to both V_{SS} pin and exposed center pad. V_{SS} pin must be connected to supply ground for proper device operation. For enhanced thermal, electrical, and board level performance, the exposed pad needs to be soldered to the board using a corresponding thermal pad on the board and for proper heat conduction through the board, thermal vias need to be incorporated in the PCB in the thermal pad region.

[2] This pin must be grounded when this feature is not used.

7. Functional description

Refer to Figure 1 “Block diagram of PCA9685”.

7.1 Device addresses

Following a START condition, the bus master must output the address of the slave it is accessing.

There are a maximum of 64 possible programmable addresses using the 6 hardware address pins. Two of these addresses, Software Reset and LED All Call, cannot be used because their default power-up state is ON, leaving a maximum of 62 addresses. Using other reserved addresses, as well as any other subcall address, will reduce the total number of possible addresses even further.

7.1.1 Regular I²C-bus slave address

The I²C-bus slave address of the PCA9685 is shown in Figure 4. To conserve power, no internal pull-up resistors are incorporated on the hardware selectable address pins and they must be pulled HIGH or LOW.

Remark: Using reserved I²C-bus addresses will interfere with other devices, but only if the devices are on the bus and/or the bus will be open to other I²C-bus systems at some later date. In a closed system where the designer controls the address assignment these addresses can be used since the PCA9685 treats them like any other address. The LED All Call, Software Reset and PCA9564 or PCA9665 slave address (if on the bus) can never be used for individual device addresses.

- PCA9685 LED All Call address (1110 000) and Software Reset (0000 0110) which are active on start-up

- PCA9564 (0000 000) or PCA9665 (1110 000) slave address which is active on start-up
- 'reserved for future use' I²C-bus addresses (0000 011, 1111 1XX)
- slave devices that use the 10-bit addressing scheme (1111 0XX)
- slave devices that are designed to respond to the General Call address (0000 000) which is used as the software reset address
- High-speed mode (Hs-mode) master code (0000 1XX)

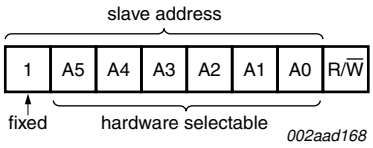


Fig 4. Slave address

The last bit of the address byte defines the operation to be performed. When set to logic 1 a read is selected, while a logic 0 selects a write operation.

7.1.2 LED All Call I²C-bus address

- Default power-up value (ALLCALLADR register): E0h or 1110 000X
- Programmable through I²C-bus (volatile programming)
- At power-up, LED All Call I²C-bus address is enabled. PCA9685 sends an ACK when E0h (R/$\overline{W}$ = 0) or E1h (R/$\overline{W}$ = 1) is sent by the master.

See Section 7.3.7 "ALLCALLADR, LED All Call I²C-bus address" for more detail.

Remark: The default LED All Call I²C-bus address (E0h or 1110 000X) must not be used as a regular I²C-bus slave address since this address is enabled at power-up. All the PCA9685s on the I²C-bus will acknowledge the address if sent by the I²C-bus master.

7.1.3 LED Sub Call I²C-bus addresses

- 3 different I²C-bus addresses can be used
- Default power-up values:
  - SUBADR1 register: E2h or 1110 001X
  - SUBADR2 register: E4h or 1110 010X
  - SUBADR3 register: E8h or 1110 100X
- Programmable through I²C-bus (volatile programming)
- At power-up, Sub Call I²C-bus addresses are disabled. PCA9685 does not send an ACK when E2h (R/$\overline{W}$ = 0) or E3h (R/$\overline{W}$ = 1), E4h (R/$\overline{W}$ = 0) or E5h (R/$\overline{W}$ = 1), or E8h (R/$\overline{W}$ = 0) or E9h (R/$\overline{W}$ = 1) is sent by the master.

See Section 7.3.6 "SUBADR1 to SUBADR3, I²C-bus subaddress 1 to 3" for more detail.

Remark: The default LED Sub Call I²C-bus addresses may be used as regular I²C-bus slave addresses as long as they are disabled.

7.1.4 Software Reset I²C-bus address

The address shown in Figure 5 is used when a reset of the PCA9685 needs to be performed by the master. The Software Reset address (SWRST Call) must be used with R/$\overline{W}$ = logic 0. If R/$\overline{W}$ = logic 1, the PCA9685 does not acknowledge the SWRST. See Section 7.6 “Software reset” for more detail.

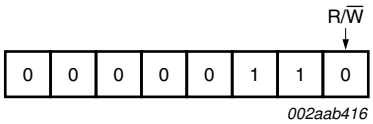


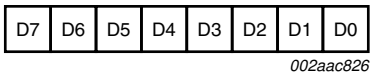
Fig 5. Software Reset address

Remark: The Software Reset I²C-bus address is a reserved address and cannot be used as a regular I²C-bus slave address or as an LED All Call or LED Sub Call address.

7.2 Control register

Following the successful acknowledgement of the slave address, LED All Call address or LED Sub Call address, the bus master will send a byte to the PCA9685, which will be stored in the Control register.

This register is used as a pointer to determine which register will be accessed.



reset state = 00h

Remark: The Control register does not apply to the Software Reset I²C-bus address.

Fig 6. Control register

7.3 Register definitions

Table 3. Register summary

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
0	00	0	0	0	0	0	0	0	0	MODE1	read/write	Mode register 1
1	01	0	0	0	0	0	0	0	1	MODE2	read/write	Mode register 2
2	02	0	0	0	0	0	0	1	0	SUBADR1	read/write	I²C-bus subaddress 1
3	03	0	0	0	0	0	0	1	1	SUBADR2	read/write	I²C-bus subaddress 2
4	04	0	0	0	0	0	1	0	0	SUBADR3	read/write	I²C-bus subaddress 3
5	05	0	0	0	0	0	1	0	1	ALLCALLADR	read/write	LED All Call I²C-bus address
6	06	0	0	0	0	0	1	1	0	LED0_ON_L	read/write	LED0 output and brightness control byte 0
7	07	0	0	0	0	0	1	1	1	LED0_ON_H	read/write	LED0 output and brightness control byte 1
8	08	0	0	0	0	1	0	0	0	LED0_OFF_L	read/write	LED0 output and brightness control byte 2
9	09	0	0	0	0	1	0	0	1	LED0_OFF_H	read/write	LED0 output and brightness control byte 3
10	0A	0	0	0	0	1	0	1	0	LED1_ON_L	read/write	LED1 output and brightness control byte 0
11	0B	0	0	0	0	1	0	1	1	LED1_ON_H	read/write	LED1 output and brightness control byte 1
12	0C	0	0	0	0	1	1	0	0	LED1_OFF_L	read/write	LED1 output and brightness control byte 2
13	0D	0	0	0	0	1	1	0	1	LED1_OFF_H	read/write	LED1 output and brightness control byte 3
14	0E	0	0	0	0	1	1	1	0	LED2_ON_L	read/write	LED2 output and brightness control byte 0
15	0F	0	0	0	0	1	1	1	1	LED2_ON_H	read/write	LED2 output and brightness control byte 1
16	10	0	0	0	1	0	0	0	0	LED2_OFF_L	read/write	LED2 output and brightness control byte 2
17	11	0	0	0	1	0	0	0	1	LED2_OFF_H	read/write	LED2 output and brightness control byte 3
18	12	0	0	0	1	0	0	1	0	LED3_ON_L	read/write	LED3 output and brightness control byte 0
19	13	0	0	0	1	0	0	1	1	LED3_ON_H	read/write	LED3 output and brightness control byte 1
20	14	0	0	0	1	0	1	0	0	LED3_OFF_L	read/write	LED3 output and brightness control byte 2
21	15	0	0	0	1	0	1	0	1	LED3_OFF_H	read/write	LED3 output and brightness control byte 3

Table 3. Register summary *…continued*

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
22	16	0	0	0	1	0	1	1	0	LED4_ON_L	read/write	LED4 output and brightness control byte 0
23	17	0	0	0	1	0	1	1	1	LED4_ON_H	read/write	LED4 output and brightness control byte 1
24	18	0	0	0	1	1	0	0	0	LED4_OFF_L	read/write	LED4 output and brightness control byte 2
25	19	0	0	0	1	1	0	0	1	LED4_OFF_H	read/write	LED4 output and brightness control byte 3
26	1A	0	0	0	1	1	0	1	0	LED5_ON_L	read/write	LED5 output and brightness control byte 0
27	1B	0	0	0	1	1	0	1	1	LED5_ON_H	read/write	LED5 output and brightness control byte 1
28	1C	0	0	0	1	1	1	0	0	LED5_OFF_L	read/write	LED5 output and brightness control byte 2
29	1D	0	0	0	1	1	1	0	1	LED5_OFF_H	read/write	LED5 output and brightness control byte 3
30	1E	0	0	0	1	1	1	1	0	LED6_ON_L	read/write	LED6 output and brightness control byte 0
31	1F	0	0	0	1	1	1	1	1	LED6_ON_H	read/write	LED6 output and brightness control byte 1
32	20	0	0	1	0	0	0	0	0	LED6_OFF_L	read/write	LED6 output and brightness control byte 2
33	21	0	0	1	0	0	0	0	1	LED6_OFF_H	read/write	LED6 output and brightness control byte 3
34	22	0	0	1	0	0	0	1	0	LED7_ON_L	read/write	LED7 output and brightness control byte 0
35	23	0	0	1	0	0	0	1	1	LED7_ON_H	read/write	LED7 output and brightness control byte 1
36	24	0	0	1	0	0	1	0	0	LED7_OFF_L	read/write	LED7 output and brightness control byte 2
37	25	0	0	1	0	0	1	0	1	LED7_OFF_H	read/write	LED7 output and brightness control byte 3
38	26	0	0	1	0	0	1	1	0	LED8_ON_L	read/write	LED8 output and brightness control byte 0
39	27	0	0	1	0	0	1	1	1	LED8_ON_H	read/write	LED8 output and brightness control byte 1
40	28	0	0	1	0	1	0	0	0	LED8_OFF_L	read/write	LED8 output and brightness control byte 2
41	29	0	0	1	0	1	0	0	1	LED8_OFF_H	read/write	LED8 output and brightness control byte 3

Table 3. Register summary *…continued*

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
42	2A	0	0	1	0	1	0	1	0	LED9_ON_L	read/write	LED9 output and brightness control byte 0
43	2B	0	0	1	0	1	0	1	1	LED9_ON_H	read/write	LED9 output and brightness control byte 1
44	2C	0	0	1	0	1	1	0	0	LED9_OFF_L	read/write	LED9 output and brightness control byte 2
45	2D	0	0	1	0	1	1	0	1	LED9_OFF_H	read/write	LED9 output and brightness control byte 3
46	2E	0	0	1	0	1	1	1	0	LED10_ON_L	read/write	LED10 output and brightness control byte 0
47	2F	0	0	1	0	1	1	1	1	LED10_ON_H	read/write	LED10 output and brightness control byte 1
48	30	0	0	1	1	0	0	0	0	LED10_OFF_L	read/write	LED10 output and brightness control byte 2
49	31	0	0	1	1	0	0	0	1	LED10_OFF_H	read/write	LED10 output and brightness control byte 3
50	32	0	0	1	1	0	0	1	0	LED11_ON_L	read/write	LED11 output and brightness control byte 0
51	33	0	0	1	1	0	0	1	1	LED11_ON_H	read/write	LED11 output and brightness control byte 1
52	34	0	0	1	1	0	1	0	0	LED11_OFF_L	read/write	LED11 output and brightness control byte 2
53	35	0	0	1	1	0	1	0	1	LED11_OFF_H	read/write	LED11 output and brightness control byte 3
54	36	0	0	1	1	0	1	1	0	LED12_ON_L	read/write	LED12 output and brightness control byte 0
55	37	0	0	1	1	0	1	1	1	LED12_ON_H	read/write	LED12 output and brightness control byte 1
56	38	0	0	1	1	1	0	0	0	LED12_OFF_L	read/write	LED12 output and brightness control byte 2
57	39	0	0	1	1	1	0	0	1	LED12_OFF_H	read/write	LED12 output and brightness control byte 3
58	3A	0	0	1	1	1	0	1	0	LED13_ON_L	read/write	LED13 output and brightness control byte 0
59	3B	0	0	1	1	1	0	1	1	LED13_ON_H	read/write	LED13 output and brightness control byte 1
60	3C	0	0	1	1	1	1	0	0	LED13_OFF_L	read/write	LED13 output and brightness control byte 2
61	3D	0	0	1	1	1	1	0	1	LED13_OFF_H	read/write	LED13 output and brightness control byte 3

Table 3. Register summary *…continued*

Register# (decimal)	Register# (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
62	3E	0	0	1	1	1	1	1	0	LED14_ON_L	read/write	LED14 output and brightness control byte 0
63	3F	0	0	1	1	1	1	1	1	LED14_ON_H	read/write	LED14 output and brightness control byte 1
64	40	0	1	0	0	0	0	0	0	LED14_OFF_L	read/write	LED14 output and brightness control byte 2
65	41	0	1	0	0	0	0	0	1	LED14_OFF_H	read/write	LED14 output and brightness control byte 3
66	42	0	1	0	0	0	0	1	0	LED15_ON_L	read/write	LED15 output and brightness control byte 0
67	43	0	1	0	0	0	0	1	1	LED15_ON_H	read/write	LED15 output and brightness control byte 1
68	44	0	1	0	0	0	1	0	0	LED15_OFF_L	read/write	LED15 output and brightness control byte 2
69	45	0	1	0	0	0	1	0	1	LED15_OFF_H	read/write	LED15 output and brightness control byte 3
...	reserved for future use											
250	FA	1	1	1	1	1	0	1	0	ALL_LED_ON_L	write/read zero	load all the LEDn_ON registers, byte 0
251	FB	1	1	1	1	1	0	1	1	ALL_LED_ON_H	write/read zero	load all the LEDn_ON registers, byte 1
252	FC	1	1	1	1	1	1	0	0	ALL_LED_OFF_L	write/read zero	load all the LEDn_OFF registers, byte 0
253	FD	1	1	1	1	1	1	0	1	ALL_LED_OFF_H	write/read zero	load all the LEDn_OFF registers, byte 1
254	FE	1	1	1	1	1	1	1	0	PRE_SCALE[1]	read/write	prescaler for output frequency
255	FF	1	1	1	1	1	1	1	1	TestMode[2]	read/write	defines the test mode to be entered
...	All further addresses are reserved for future use; reserved addresses will not be acknowledged.											

[1] Writes to PRE_SCALE register are blocked when SLEEP bit is logic 0 (MODE 1).

[2] Reserved. Writes to this register may cause unpredictable results.

Remark: Auto Increment past register 69 will point to MODE1 register (register 0). Auto Increment also works from register 250 to register 254, then rolls over to register 0.

7.3.1 Mode register 1, MODE1

Table 4. MODE1 - Mode register 1 (address 00h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7	RESTART	R		Shows state of RESTART logic. See Section 7.3.1.1 for detail.
		W		User writes logic 1 to this bit to clear it to logic 0. A user write of logic 0 will have no effect. See Section 7.3.1.1 for detail.
			0*	Restart disabled.
			1	Restart enabled.
6	EXTCLK	R/W		To use the EXTCLK pin, this bit must be set by the following sequence: 1. Set the SLEEP bit in MODE1. This turns off the internal oscillator. 2. Write logic 1s to both the SLEEP and EXTCLK bits in MODE1. The switch is now made. The external clock can be active during the switch because the SLEEP bit is set. This bit is a 'sticky bit', that is, it cannot be cleared by writing a logic 0 to it. The EXTCLK bit can **only** be cleared by a power cycle or software reset. EXTCLK range is DC to 50 MHz. $refresh\_rate = \frac{EXTCLK}{4096 \times (prescale + 1)}$
			0*	Use internal clock.
			1	Use EXTCLK pin clock.
5	AI	R/W	0*	Register Auto-Increment disabled[1].
			1	Register Auto-Increment enabled.
4	SLEEP	R/W	0	Normal mode[2].
			1*	Low power mode. Oscillator off[3][4].
3	SUB1	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 1.
			1	PCA9685 responds to I²C-bus subaddress 1.
2	SUB2	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 2.
			1	PCA9685 responds to I²C-bus subaddress 2.
1	SUB3	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 3.
			1	PCA9685 responds to I²C-bus subaddress 3.
0	ALLCALL	R/W	0	PCA9685 does not respond to LED All Call I²C-bus address.
			1*	PCA9685 responds to LED All Call I²C-bus address.

[1] When the Auto Increment flag is set, AI = 1, the Control register is automatically incremented after a read or write. This allows the user to program the registers sequentially.

[2] It takes 500 μs max. for the oscillator to be up and running once SLEEP bit has been set to logic 0. Timings on LEDn outputs are not guaranteed if PWM control registers are accessed within the 500 μs window. There is no start-up delay required when using the EXTCLK pin as the PWM clock.

[3] No PWM control is possible when the oscillator is off.

[4] When the oscillator is off (Sleep mode) the LEDn outputs cannot be turned on, off or dimmed/blinked.

7.3.1.1 Restart mode

If the PCA9685 is operating and the user decides to put the chip to sleep (setting MODE1 bit 4) without stopping any of the PWM channels, the RESTART bit (MODE1 bit 7) will be set to logic 1 at the end of the PWM refresh cycle. The contents of each PWM register are held valid when the clock is off.

To restart all of the previously active PWM channels with a few I²C-bus cycles do the following steps:

1. Read MODE1 register.
2. Check that bit 7 (RESTART) is a logic 1. If it is, clear bit 4 (SLEEP). Allow time for oscillator to stabilize (500 μs).
3. Write logic 1 to bit 7 of MODE1 register. All PWM channels will restart and the RESTART bit will clear.

Remark: The SLEEP bit **must** be logic 0 for at least 500 μs, before a logic 1 is written into the RESTART bit.

Other actions that will clear the RESTART bit are:

1. Power cycle.
2. I²C Software Reset command.
3. If the MODE2 OCH bit is logic 0, write to any PWM register then issue an I²C-bus STOP.
4. If the MODE2 OCH bit is logic 1, write to all four PWM registers in any PWM channel.

Likewise, if the user does an orderly shutdown[1] of all the PWM channels before setting the SLEEP bit, the RESTART bit will be cleared. If this is done the contents of all PWM registers are invalidated and must be reloaded before reuse.

An example of the use of the RESTART bit would be the restoring of a customer's laptop LCD backlight intensity coming out of Standby to the level it was before going into Standby.

1. Two methods can be used to do an orderly shutdown. The fastest is to write a logic 1 to bit 4 in register ALL_LED_OFF_H. The other method is to write logic 1 to bit 4 in each active PWM channel LEDn_OFF_H register.

7.3.2 Mode register 2, MODE2

Table 5. MODE2 - Mode register 2 (address 01h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7 to 5	-	read only	000*	reserved
4	INVRT[1]	R/W	0*	Output logic state not inverted. Value to use when external driver used. Applicable when $\overline{OE}$ = 0.
			1	Output logic state inverted. Value to use when no external driver used. Applicable when $\overline{OE}$ = 0.
3	OCH	R/W	0*	Outputs change on STOP command[2].
			1	Outputs change on ACK[3].
2	OUTDRV[1]	R/W	0	The 16 LEDn outputs are configured with an open-drain structure.
			1*	The 16 LEDn outputs are configured with a totem pole structure.
1 to 0	OUTNE[1:0][4]	R/W	00*	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = 0.
			01	When $\overline{OE}$ = 1 (output drivers not enabled): LEDn = 1 when OUTDRV = 1 LEDn = high-impedance when OUTDRV = 0 (same as OUTNE[1:0] = 10)
			1X	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = high-impedance.

[1] See Section 7.7 “Using the PCA9685 with and without external drivers” for more details. Normal LEDs can be driven directly in either mode. Some newer LEDs include integrated Zener diodes to limit voltage transients, reduce EMI, protect the LEDs and these must be driven only in the open-drain mode to prevent overheating the IC.

[2] Change of the outputs at the STOP command allows synchronizing outputs of more than one PCA9685. Applicable to registers from 06h (LED0_ON_L) to 45h (LED15_OFF_H) only. 1 or more registers can be written, in any order, before STOP.

[3] Update on ACK requires **all** 4 PWM channel registers to be loaded before outputs will change on the **last** ACK.

[4] See Section 7.4 “Active LOW output enable input” for more details.

7.3.3 LED output and PWM control

The turn-on time of each LED driver output and the duty cycle of PWM can be controlled independently using the LEDn_ON and LEDn_OFF registers.

There will be two 12-bit registers per LED output. These registers will be programmed by the user. Both registers will hold a value from 0 to 4095. One 12-bit register will hold a value for the ON time and the other 12-bit register will hold the value for the OFF time. The ON and OFF times are compared with the value of a 12-bit counter that will be running continuously from 0000h to 0FFFh (0 to 4095 decimal).

Update on ACK requires all 4 PWM channel registers to be loaded before outputs will change on the last ACK.

The ON time, which is programmable, will be the time the LED output will be asserted and the OFF time, which is also programmable, will be the time when the LED output will be negated. In this way, the phase shift becomes completely programmable. The resolution for the phase shift is $^{1}/_{4096}$ of the target frequency. Table 6 lists these registers.

The following two examples illustrate how to calculate values to be loaded into these registers.

Example 1: (assumes that the LED0 output is used and (delay time) + (PWM duty cycle) ≤ 100 %)

Delay time = 10 %; PWM duty cycle = 20 % (LED on time = 20 %; LED off time = 80 %).

Delay time = 10 % = 409.6 ~ 410 counts = 19Ah.

Since the counter starts at 0 and ends at 4095, we will subtract 1, so delay time = 199h counts.

LED0_ON_H = 1h; LED0_ON_L = 99h

LED on time = 20 % = 819.2 ~ 819 counts.

Off time = 4CCh (decimal 410 + 819 − 1 = 1228)

LED0_OFF_H = 4h; LED0_OFF_L = CCh

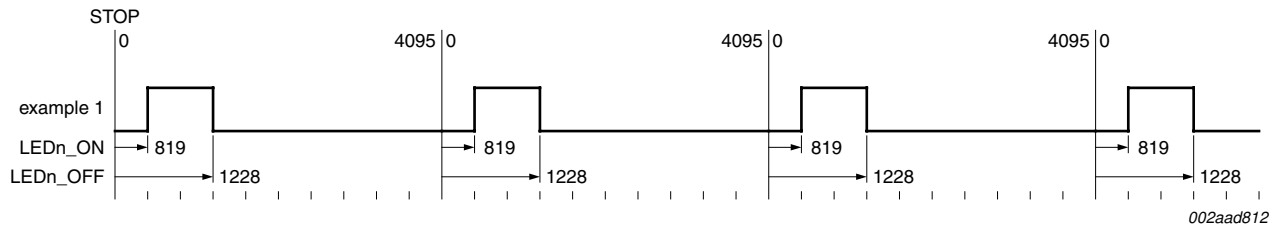


Fig 7. LED output, example 1

Example 2: (assumes that the LED4 output is used and (delay time) + (PWM duty cycle > 100 %)

Delay time = 90 %; PWM duty cycle = 90 % (LED on time = 90 %; LED off time = 10 %).

Delay time = 90 % = 3686.4 ~ 3686 counts − 1 = 3685 = E65h.

LED4_ON_H = Eh; LED4_ON_L = 65h

LED on time = 90 % = 3686 counts.

Since the delay time and LED on period of the duty cycle is greater than 4096 counts, the LEDn_OFF count will occur in the next frame. Therefore, 4096 is subtracted from the LEDn_OFF count to get the correct LEDn_OFF count. See Figure 9, Figure 10 and Figure 11.

Off time = 4CBh (decimal 3685 + 3686 = 7372 − 4096 = 3275)

LED4_OFF_H = 4h; LED4_OFF_L = CBh

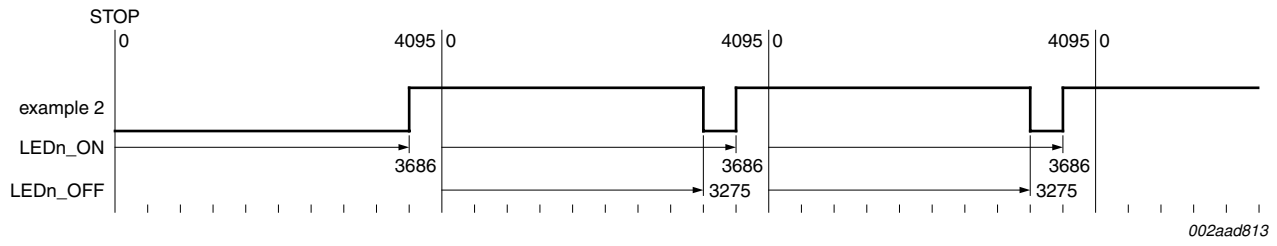
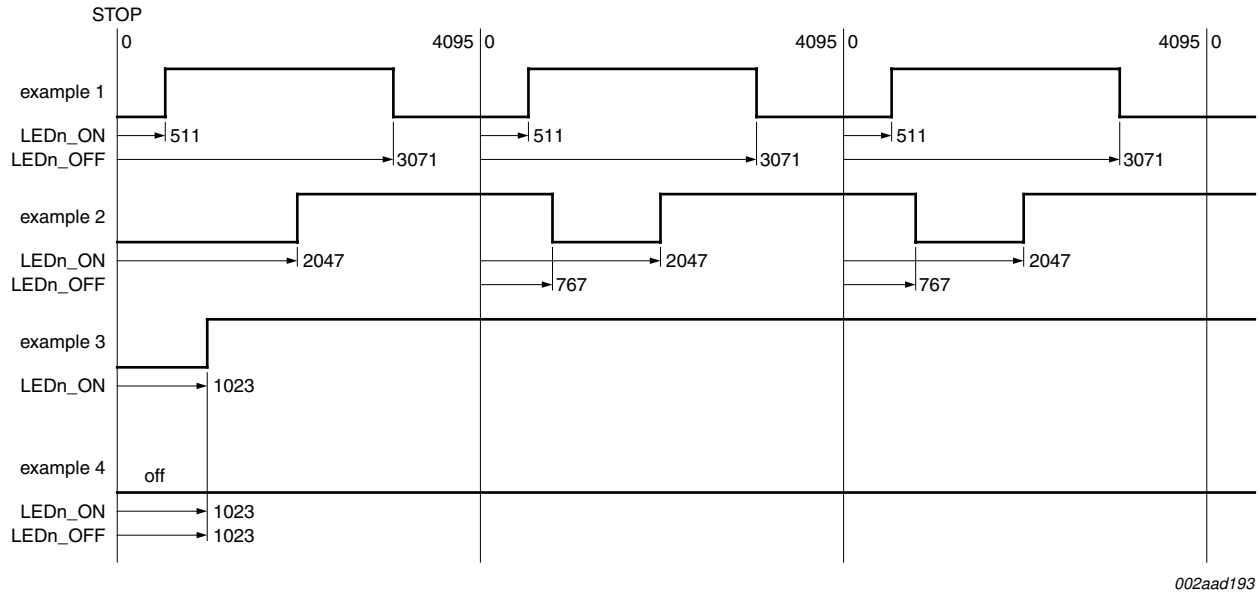


Fig 8. LED output, example 2



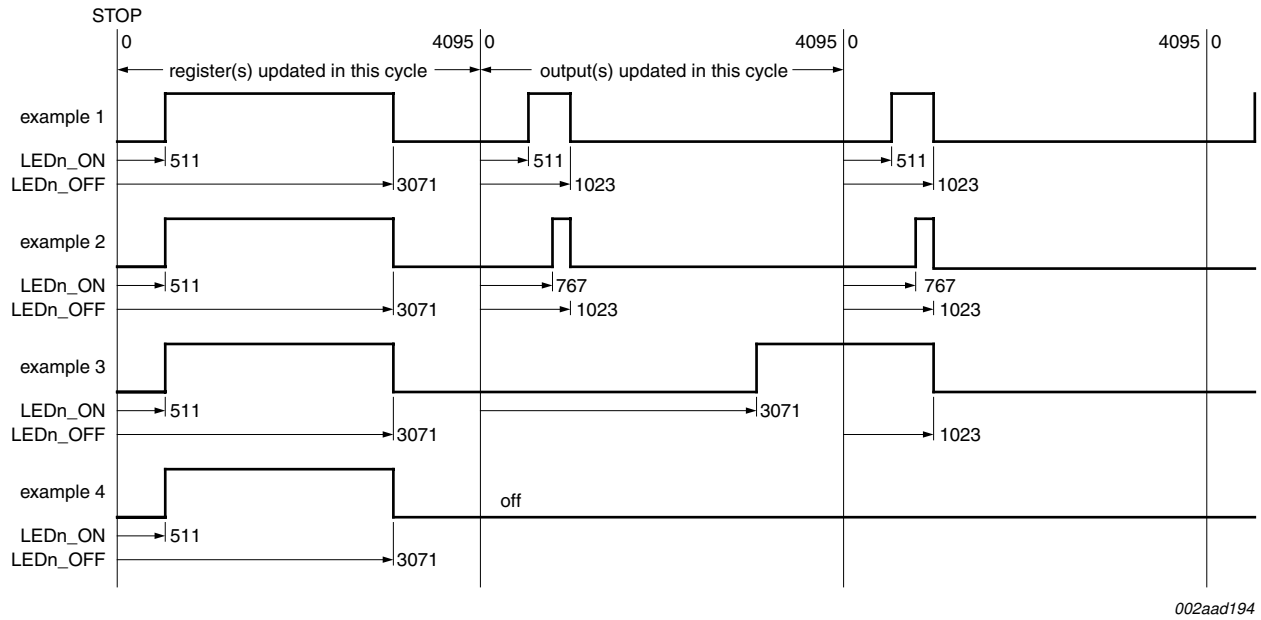
Example 1: LEDn_ON < LEDn_OFF

Example 2: LEDn_ON > LEDn_OFF

Example 3: LEDn_ON[12] = 1; LEDn_ON[11:0] = 1022; LEDn_OFF[12] = 0; LEDn_OFF[11:0] = don't care

Example 4: LEDn_ON[12] = 0; LEDn_OFF[12] = 0; LEDn_ON[11:0] = LEDn_OFF[11:0]

Fig 9. Output example



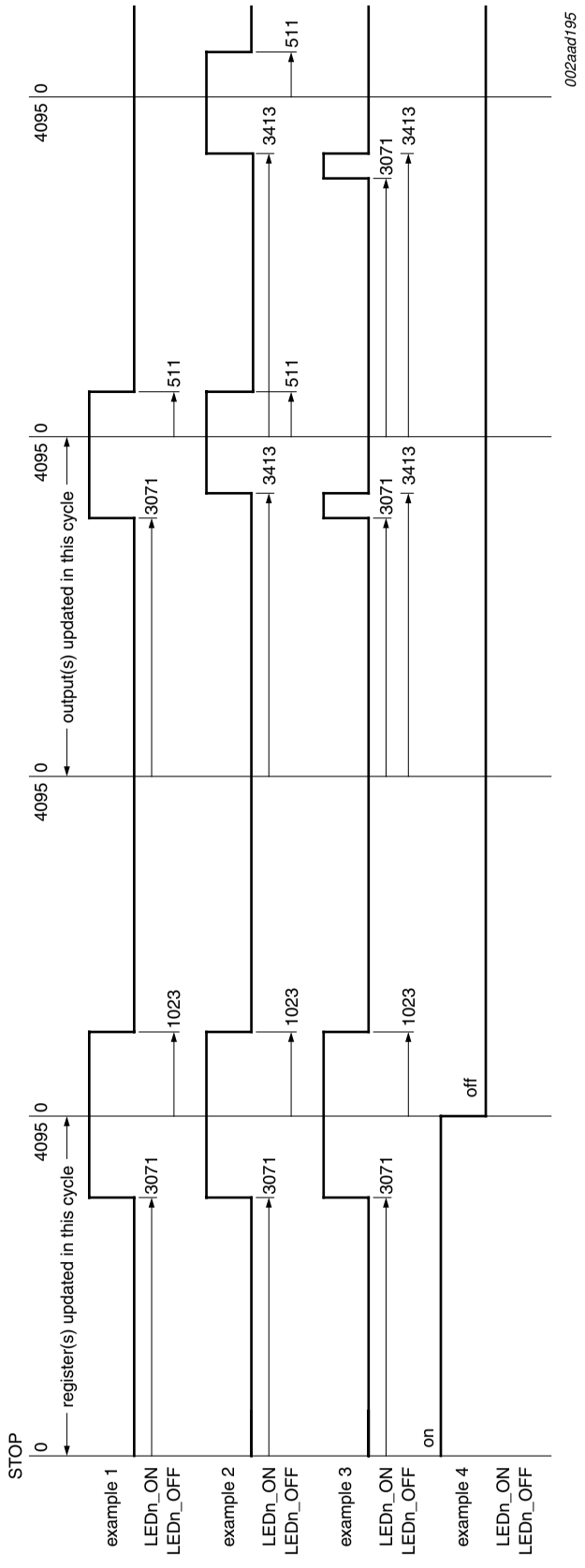
Example 1: LEDn_ON unchanged and LEDn_OFF decreased.

Example 2: LEDn_ON increased and LEDn_OFF decreased.

Example 3: LEDn_ON made > LEDn_OFF.

Example 4: LEDn_OFF[12] set to 1.

Fig 10. Update examples when LEDn_ON < LEDn_OFF



Example 1: LEDn_ON unchanged and LEDn_OFF decreased, but delay still > LEDn_OFF

Example 2: LEDn_ON changed and LEDn_OFF changed, but delay still > LEDn_OFF

Example 3: LEDn_ON unchanged and LEDn_OFF increased where LEDn_ON < LEDn_OFF

Example 4: LEDn_ON[12] = 1 and LEDn_OFF[12] changed from 0 to 1

Fig 11. Update examples when LEDn_ON > LEDn_OFF

Table 6. LED_ON, LED_OFF control registers (address 06h to 45h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
06h	LED0_ON_L	7:0	LED0_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED0, 8 LSBs
07h	LED0_ON_H	7:5	reserved	R	000*	non-writable
		4	LED0_ON_H[4]	R/W	0*	LED0 full ON
		3:0	LED0_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED0, 4 MSBs
08h	LED0_OFF_L	7:0	LED0_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED0, 8 LSBs
09h	LED0_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED0_OFF_H[4]	R/W	1*	LED0 full OFF
		3:0	LED0_OFF_H[3:0]	R/W	0000*	
0Ah	LED1_ON_L	7:0	LED1_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED1, 8 LSBs
0Bh	LED1_ON_H	7:5	reserved	R	000*	non-writable
		4	LED1_ON_H[4]	R/W	0*	LED1 full ON
		3:0	LED1_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED1, 4 MSBs
0Ch	LED1_OFF_L	7:0	LED1_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED1, 8 LSBs
0Dh	LED1_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED1_OFF_H[4]	R/W	1*	LED1 full OFF
		3:0	LED1_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED1, 4 MSBs
0Eh	LED2_ON_L	7:0	LED2_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED2, 8 LSBs
0Fh	LED2_ON_H	7:5	reserved	R	000*	non-writable
		4	LED2_ON_H[4]	R/W	0*	LED2 full ON
		3:0	LED2_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED2, 4 MSBs
10h	LED2_OFF_L	7:0	LED2_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED2, 8 LSBs
11h	LED2_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED2_OFF_H[4]	R/W	1*	LED2 full OFF
		3:0	LED2_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED2, 4 MSBs
12h	LED3_ON_L	7:0	LED3_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED3, 8 LSBs
13h	LED3_ON_H	7:5	reserved	R	000*	non-writable
		4	LED3_ON_H[4]	R/W	0*	LED3 full ON
		3:0	LED3_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED3, 4 MSBs
14h	LED3_OFF_L	7:0	LED3_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED3, 8 LSBs
15h	LED3_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED3_OFF_H[4]	R/W	1*	LED3 full OFF
		3:0	LED3_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED3, 4 MSBs
16h	LED4_ON_L	7:0	LED4_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED4, 8 LSBs
17h	LED4_ON_H	7:5	reserved	R	000*	non-writable
		4	LED4_ON_H[4]	R/W	0*	LED4 full ON
		3:0	LED4_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED4, 4 MSBs

Table 6. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
18h	LED4_OFF_L	7:0	LED4_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED4, 8 LSBs
19h	LED4_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED4_OFF_H[4]	R/W	1*	LED4 full OFF
		3:0	LED4_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED4, 4 MSBs
1Ah	LED5_ON_L	7:0	LED5_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED5, 8 LSBs
1Bh	LED5_ON_H	7:5	reserved	R	000*	non-writable
		4	LED5_ON_H[4]	R/W	0*	LED5 full ON
		3:0	LED5_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED5, 4 MSBs
1Ch	LED5_OFF_L	7:0	LED5_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED5, 8 LSBs
1Dh	LED5_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED5_OFF_H[4]	R/W	1*	LED5 full OFF
		3:0	LED5_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED5, 4 MSBs
1Eh	LED6_ON_L	7:0	LED6_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED6, 8 LSBs
1Fh	LED6_ON_H	7:5	reserved	R	000*	non-writable
		4	LED6_ON_H[4]	R/W	0*	LED6 full ON
		3:0	LED6_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED6, 4 MSBs
20h	LED6_OFF_L	7:0	LED6_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED6, 8 LSBs
21h	LED6_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED6_OFF_H[4]	R/W	1*	LED6 full OFF
		3:0	LED6_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED6, 4 MSBs
22h	LED7_ON_L	7:0	LED7_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED7, 8 LSBs
23h	LED7_ON_H	7:5	reserved	R	000*	non-writable
		4	LED7_ON_H[4]	R/W	0*	LED7 full ON
		3:0	LED7_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED7, 4 MSBs
24h	LED7_OFF_L	7:0	LED7_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED7, 8 LSBs
25h	LED7_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED7_OFF_H[4]	R/W	1*	LED7 full OFF
		3:0	LED7_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED7, 4 MSBs
26h	LED8_ON_L	7:0	LED8_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED8, 8 LSBs
27h	LED8_ON_H	7:5	reserved	R	000*	non-writable
		4	LED8_ON_H[4]	R/W	0*	LED8 full ON
		3:0	LED8_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED8, 4 MSBs
28h	LED8_OFF_L	7:0	LED8_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED8, 8 LSBs
29h	LED8_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED8_OFF_H[4]	R/W	1*	LED8 full OFF
		3:0	LED8_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED8, 4 MSBs

Table 6. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
2Ah	LED9_ON_L	7:0	LED9_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED9, 8 LSBs
2Bh	LED9_ON_H	7:5	reserved	R	000*	non-writable
		4	LED9_ON_H[4]	R/W	0*	LED9 full ON
		3:0	LED9_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED9, 4 MSBs
2Ch	LED9_OFF_L	7:0	LED9_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED9, 8 LSBs
2Dh	LED9_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED9_OFF_H[4]	R/W	1*	LED9 full OFF
		3:0	LED9_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED9, 4 MSBs
2Eh	LED10_ON_L	7:0	LED10_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED10, 8 LSBs
2Fh	LED10_ON_H	7:5	reserved	R	000*	non-writable
		4	LED10_ON_H[4]	R/W	0*	LED10 full ON
		3:0	LED10_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED10, 4 MSBs
30h	LED10_OFF_L	7:0	LED10_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED10, 8 LSBs
31h	LED10_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED10_OFF_H[4]	R/W	1*	LED10 full OFF
		3:0	LED10_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED10, 4 MSBs
32h	LED11_ON_L	7:0	LED11_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED11, 8 LSBs
33h	LED11_ON_H	7:5	reserved	R	000*	non-writable
		4	LED11_ON_H[4]	R/W	0*	LED11 full ON
		3:0	LED11_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED11, 4 MSBs
34h	LED11_OFF_L	7:0	LED11_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED11, 8 LSBs
35h	LED11_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED11_OFF_H[4]	R/W	1*	LED11 full OFF
		3:0	LED11_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED11, 4 MSBs
36h	LED12_ON_L	7:0	LED12_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED12, 8 LSBs
37h	LED12_ON_H	7:5	reserved	R	000*	non-writable
		4	LED12_ON_H[4]	R/W	0*	LED12 full ON
		3:0	LED12_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED12, 4 MSBs
38h	LED12_OFF_L	7:0	LED12_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED12, 8 LSBs
39h	LED12_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED12_OFF_H[4]	R/W	1*	LED12 full OFF
		3:0	LED12_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED12, 4 MSBs
3Ah	LED13_ON_L	7:0	LED13_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED13, 8 LSBs
3Bh	LED13_ON_H	7:5	reserved	R	000*	non-writable
		4	LED13_ON_H[4]	R/W	0*	LED13 full ON
		3:0	LED13_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED13, 4 MSBs

Table 6. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
3Ch	LED13_OFF_L	7:0	LED13_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED13, 8 LSBs
3Dh	LED13_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED13_OFF_H[4]	R/W	1*	LED13 full OFF
		3:0	LED13_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED13, 4 MSBs
3Eh	LED14_ON_L	7:0	LED14_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED14, 8 LSBs
3Fh	LED14_ON_H	7:5	reserved	R	000*	non-writable
		4	LED14_ON_H[4]	R/W	0*	LED14 full ON
		3:0	LED14_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED14, 4 MSBs
40h	LED14_OFF_L	7:0	LED14_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED14, 8 LSBs
41h	LED14_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED14_OFF_H[4]	R/W	1*	LED14 full OFF
		3:0	LED14_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED14, 4 MSBs
42h	LED15_ON_L	7:0	LED15_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED15, 8 LSBs
43h	LED15_ON_H	7:5	reserved	R	000*	non-writable
		4	LED15_ON_H[4]	R/W	0*	LED15 full ON
		3:0	LED15_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED15, 4 MSBs
44h	LED15_OFF_L	7:0	LED15_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED15, 8 LSBs
45h	LED15_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED15_OFF_H[4]	R/W	1*	LED15 full OFF
		3:0	LED15_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED15, 4 MSBs

The LEDn_ON_H output control bit 4, when set to logic 1, causes the output to be always ON. The turning ON of the LED is delayed by the amount in the LEDn_ON registers. LEDn_OFF[11:0] are ignored. When this bit = 0, then the LEDn_ON and LEDn_OFF registers are used according to their normal definition.

The LEDn_OFF_H output control bit 4, when set to logic 1, causes the output to be always OFF. In this case the values in the LEDn_ON registers are ignored.

Remark: When all LED outputs are configured as ‘always OFF’, the prescale counter and all associated PWM cycle timing logic are disabled. If LEDn_ON_H[4] and LEDn_OFF_H[4] are set at the same time, the LEDn_OFF_H[4] function takes precedence.

7.3.4 ALL_LED_ON and ALL_LED_OFF control

The ALL_LED_ON and ALL_LED_OFF registers allow just four I²C-bus write sequences to fill all the ON and OFF registers with the same patterns.

Table 7. ALL_LED_ON and ALL_LED_OFF control registers (address FAh to FEh) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
FAh	ALL_LED_ON_L	7:0	ALL_LED_ON_L[7:0]	W only	0000 0000*	LEDn_ON count for ALL_LED, 8 MSBs
FBh	ALL_LED_ON_H	7:5	reserved	R	000*	non-writable
		4	ALL_LED_ON_H[4]	W only	1*	ALL_LED full ON
		3:0	ALL_LED_ON_H[3:0]	W only	0000*	LEDn_ON count for ALL_LED, 4 MSBs
FCh	ALL_LED_OFF_L	7:0	ALL_LED_OFF_L[7:0]	W only	0000 0000*	LEDn_OFF count for ALL_LED, 8 MSBs
FDh	ALL_LED_OFF_H	7:5	reserved	R	000*	non-writable
		4	ALL_LED_OFF_H[4]	W only	1*	ALL_LED full OFF
		3:0	ALL_LED_OFF_H[3:0]	W only	0000*	LEDn_OFF count for ALL_LED, 4 MSBs
FEh	PRE_SCALE	7:0	PRE_SCALE[7:0]	R/W	0001 1110*	prescaler to program the output frequency

The LEDn_ON and LEDn_OFF counts can vary from 0 to 4095. The LEDn_ON and LEDn_OFF count registers should never be programmed with the same values.

Because the loading of the LEDn_ON and LEDn_OFF registers is via the I²C-bus, and asynchronous to the internal oscillator, we want to ensure that we do not see any visual artifacts of changing the ON and OFF values. This is achieved by updating the changes at the end of the LOW cycle.

7.3.5 PWM frequency PRE_SCALE

The hardware forces a minimum value that can be loaded into the PRE_SCALE register at '3'. The PRE_SCALE register defines the frequency at which the outputs modulate. The prescale value is determined with the formula shown in Equation 1:

$$prescale\ value = round\left(\frac{osc\_clock}{4096 \times update\_rate}\right) - 1 \quad (1)$$

where the update rate is the output modulation frequency required. For example, for an output frequency of 200 Hz with an oscillator clock frequency of 25 MHz:

$$prescale\ value = round\left(\frac{25\ MHz}{4096 \times 200}\right) - 1 = 30 \quad (2)$$

The PRE_SCALE register can only be set when the SLEEP bit of MODE1 register is set to logic 1.

7.3.6 SUBADR1 to SUBADR3, I²C-bus subaddress 1 to 3

Table 8. SUBADR1 to SUBADR3 - I²C-bus subaddress registers 0 to 3 (address 02h to 04h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
02h	SUBADR1	7:1	A1[7:1]	R/W	1110 001*	I²C-bus subaddress 1
		0	A1[0]	R only	0*	reserved
03h	SUBADR2	7:1	A2[7:1]	R/W	1110 010*	I²C-bus subaddress 2
		0	A2[0]	R only	0*	reserved
04h	SUBADR3	7:1	A3[7:1]	R/W	1110 100*	I²C-bus subaddress 3
		0	A3[0]	R only	0*	reserved

Subaddresses are programmable through the I²C-bus. Default power-up values are E2h, E4h, E8h, and the device(s) will not acknowledge these addresses right after power-up (the corresponding SUBx bit in MODE1 register is equal to 0).

Once subaddresses have been programmed to their right values, SUBx bits need to be set to logic 1 in order to have the device acknowledging these addresses (MODE1 register).

Only the 7 MSBs representing the I²C-bus subaddress are valid. The LSB in SUBADRx register is a read-only bit (0).

When SUBx is set to logic 1, the corresponding I²C-bus subaddress can be used during either an I²C-bus read or write sequence.

7.3.7 ALLCALLADR, LED All Call I²C-bus address

Table 9. ALLCALLADR - LED All Call I²C-bus address register (address 05h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
05h	ALLCALLADR	7:1	AC[7:1]	R/W	1110 000*	ALLCALL I²C-bus address register
		0	AC[0]	R only	0*	reserved

The LED All Call I²C-bus address allows all the PCA9685s in the bus to be programmed at the same time (ALLCALL bit in register MODE1 must be equal to 1 (power-up default state)). This address is programmable through the I²C-bus and can be used during either an I²C-bus read or write sequence. The register address can also be programmed as a Sub Call.

Only the 7 MSBs representing the All Call I²C-bus address are valid. The LSB in ALLCALLADR register is a read-only bit (0).

If ALLCALL bit = 0, the device does not acknowledge the address programmed in register ALLCALLADR.

7.4 Active LOW output enable input

The active LOW output enable ($\overline{OE}$) pin, allows to enable or disable all the LED outputs at the same time.

- When a LOW level is applied to $\overline{OE}$ pin, all the LED outputs are enabled and follow the output state defined in the LEDn_ON and LEDn_OFF registers with the polarity defined by INVRT bit (MODE2 register).
- When a HIGH level is applied to $\overline{OE}$ pin, all the LED outputs are programmed to the value that is defined by OUTNE[1:0] in the MODE2 register.

Table 10. LED outputs when $\overline{OE}$ = 1

OUTNE1	OUTNE0	LED outputs
0	0	0
0	1	1 if OUTDRV = 1, high-impedance if OUTDRV = 0
1	0	high-impedance
1	1	high-impedance

The $\overline{OE}$ pin can be used as a synchronization signal to switch on/off several PCA9685 devices at the same time. This requires an external clock reference that provides blinking period and the duty cycle.

The $\overline{OE}$ pin can also be used as an external dimming control signal. The frequency of the external clock must be high enough not to be seen by the human eye, and the duty cycle value determines the brightness of the LEDs.

7.5 Power-on reset

When power is applied to V_{DD}, an internal power-on reset holds the PCA9685 in a reset condition until V_{DD} has reached V_{POR}. At this point, the reset condition is released and the PCA9685 registers and I²C-bus state machine are initialized to their default states. Thereafter, V_{DD} must be lowered below 0.2 V to reset the device.

7.6 Software reset

The Software Reset Call (SWRST Call) allows all the devices in the I²C-bus to be reset to the power-up state value through a specific formatted I²C-bus command. To be performed correctly, it implies that the I²C-bus is functional and that there is no device hanging the bus.

The SWRST Call function is defined as the following:

1. A START command is sent by the I²C-bus master.
2. The reserved SWRST I²C-bus address '0000 000' with the R/$\overline{W}$ bit set to '0' (write) is sent by the I²C-bus master.
3. The PCA9685 device(s) acknowledge(s) after seeing the General Call address '0000 0000' (00h) only. If the R/$\overline{W}$ bit is set to '1' (read), no acknowledge is returned to the I²C-bus master.
4. Once the General Call address has been sent and acknowledged, the master sends 1 byte with 1 specific value (SWRST data byte 1):
   a. Byte 1 = 06h: the PCA9685 acknowledges this value only. If byte 1 is not equal to 06h, the PCA9685 does not acknowledge it.

   If more than 1 byte of data is sent, the PCA9685 does not acknowledge any more.
5. Once the correct byte (SWRST data byte 1) has been sent and correctly acknowledged, the master sends a STOP command to end the SWRST Call: the PCA9685 then resets to the default value (power-up value) and is ready to be addressed again within the specified bus free time (t_{BUF}).

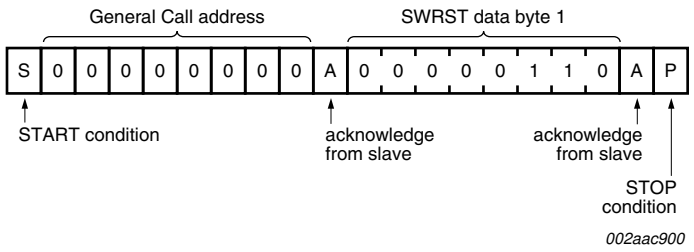


Fig 12. SWRST Call

The I²C-bus master must interpret a non-acknowledge from the PCA9685 (at any time) as a 'SWRST Call Abort'. The PCA9685 does not initiate a reset of its registers. This happens only when the format of the SWRST Call sequence is not correct.

7.7 Using the PCA9685 with and without external drivers

The PCA9685 LED output drivers are 5.5 V only tolerant and can sink up to 25 mA at 5 V.

If the device needs to drive LEDs to a higher voltage and/or higher current, use of an external driver is required.

- INVRT bit (MODE2 register) can be used to keep the LED PWM control firmware the same independently of the type of external driver. This bit allows LED output polarity inversion/non-inversion only when $\overline{OE}$ = 0.
- OUTDRV bit (MODE2 register) allows minimizing the amount of external components required to control the external driver (N-type or P-type device).

Table 11. Use of INVRT and OUTDRV based on connection to the LEDn outputs when $\overline{OE}$ = 0[1]

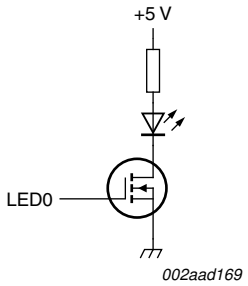
INVRT	OUTDRV	Direct connection to LEDn		External N-type driver		External P-type driver	
		Firmware	External pull-up resistor	Firmware	External pull-up resistor	Firmware	External pull-up resistor
0	0	formulas and LED output state values inverted	LED current limiting R[2]	formulas and LED output state values inverted	required	formulas and LED output state values apply	required
0	1	formulas and LED output state values inverted	LED current limiting R[2]	formulas and LED output state values apply[3]	not required[3]	formulas and LED output state values inverted	not required
1	0	formulas and LED output state values apply[2]	LED current limiting R	formulas and LED output state values apply	required	formulas and LED output state values inverted	required
1	1	formulas and LED output state values apply[2]	LED current limiting R	formulas and LED output state values inverted	not required	formulas and LED output state values apply[4]	not required[4]

[1] When $\overline{OE}$ = 1, LED output state is controlled only by OUTNE[1:0] bits (MODE2 register).

[2] Correct configuration when LEDs directly connected to the LEDn outputs (connection to V_{DD} through current limiting resistor).

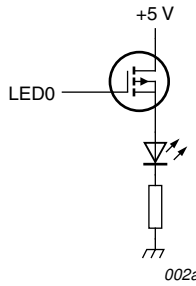
[3] Optimum configuration when external N-type (NPN, NMOS) driver used.

[4] Optimum configuration when external P-type (PNP, PMOS) driver used.



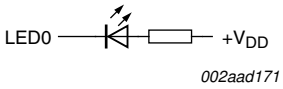
INVRT = 0
OUTDRV = 1

Fig 13. External N-type driver



INVRT = 1
OUTDRV = 1

Fig 14. External P-type driver



INVRT = 1
OUTDRV = 0

Fig 15. Direct LED connection

8. Characteristics of the I²C-bus

The I²C-bus is for 2-way, 2-line communication between different ICs or modules. The two lines are a serial data line (SDA) and a serial clock line (SCL). Both lines must be connected to a positive supply via a pull-up resistor when connected to the output stages of a device. Data transfer may be initiated only when the bus is not busy.

8.1 Bit transfer

One data bit is transferred during each clock pulse. The data on the SDA line must remain stable during the HIGH period of the clock pulse as changes in the data line at this time will be interpreted as control signals (see Figure 16).

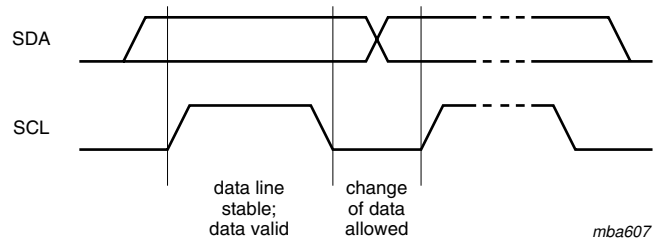


Fig 16. Bit transfer

8.1.1 START and STOP conditions

Both data and clock lines remain HIGH when the bus is not busy. A HIGH-to-LOW transition of the data line while the clock is HIGH is defined as the START condition (S). A LOW-to-HIGH transition of the data line while the clock is HIGH is defined as the STOP condition (P) (see Figure 17).

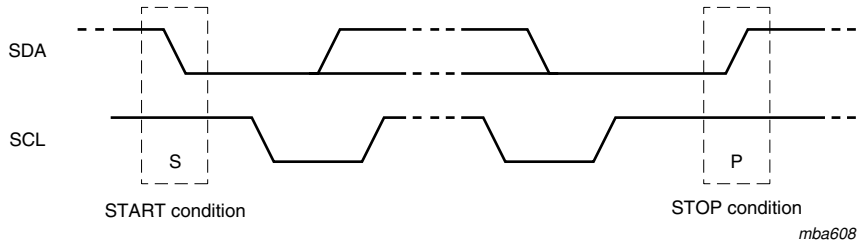


Fig 17. Definition of START and STOP conditions

8.2 System configuration

A device generating a message is a ‘transmitter’; a device receiving is the ‘receiver’. The device that controls the message is the ‘master’ and the devices which are controlled by the master are the ‘slaves’ (see Figure 18).

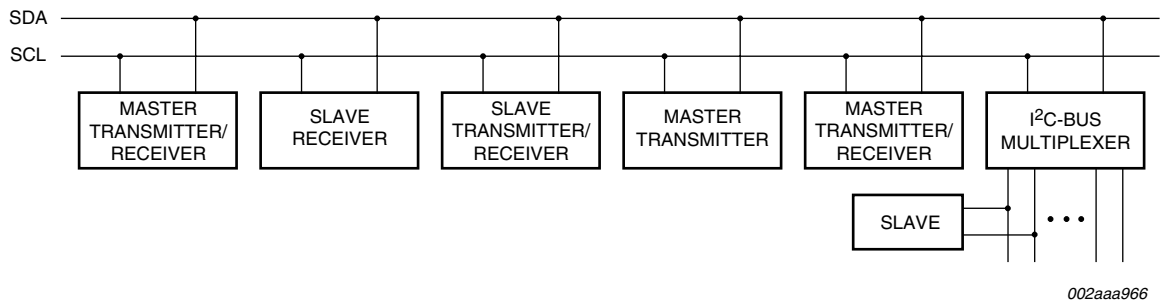


Fig 18. System configuration

8.3 Acknowledge

The number of data bytes transferred between the START and the STOP conditions from transmitter to receiver is not limited. Each byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter, whereas the master generates an extra acknowledge related clock pulse.

A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse; set-up time and hold time must be taken into account.

A master receiver must signal an end of data to the transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave. In this event, the transmitter must leave the data line HIGH to enable the master to generate a STOP condition.

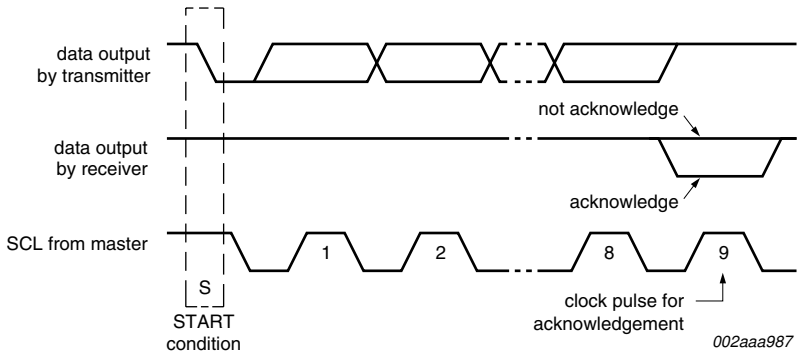
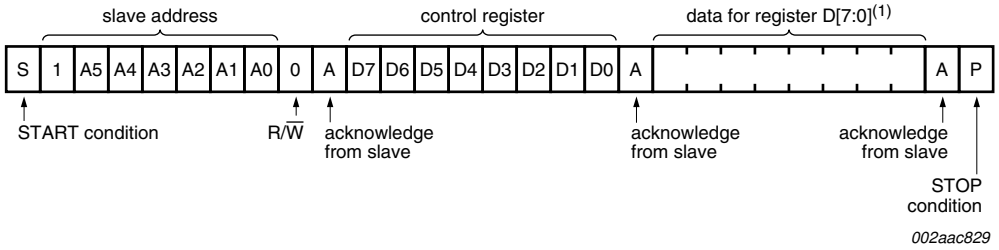


Fig 19. Acknowledgement on the I²C-bus

9. Bus transactions



(1) See Table 3 for register definition.

Fig 20. Write to a specific register

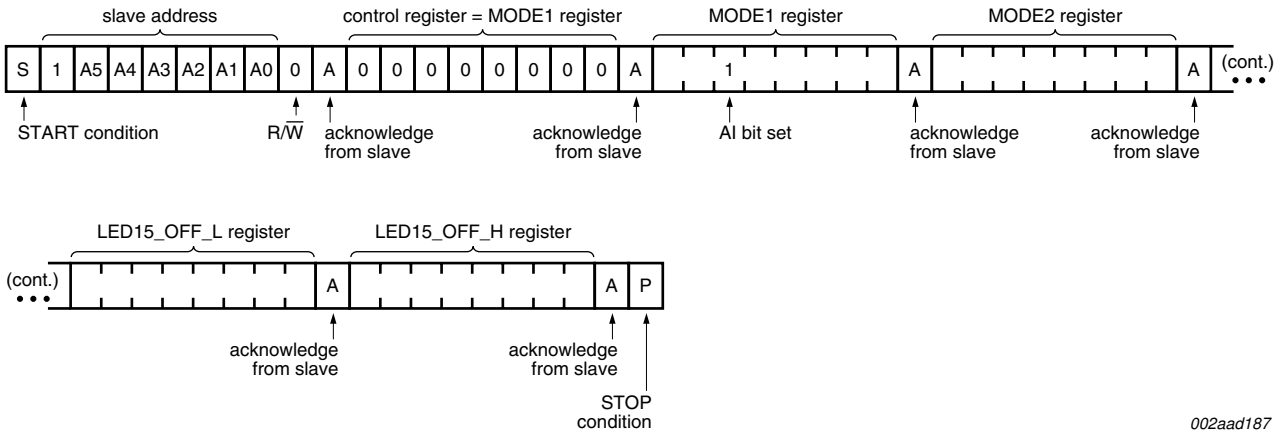


Fig 21. Write to all registers using the Auto-Increment feature; AI initially clear

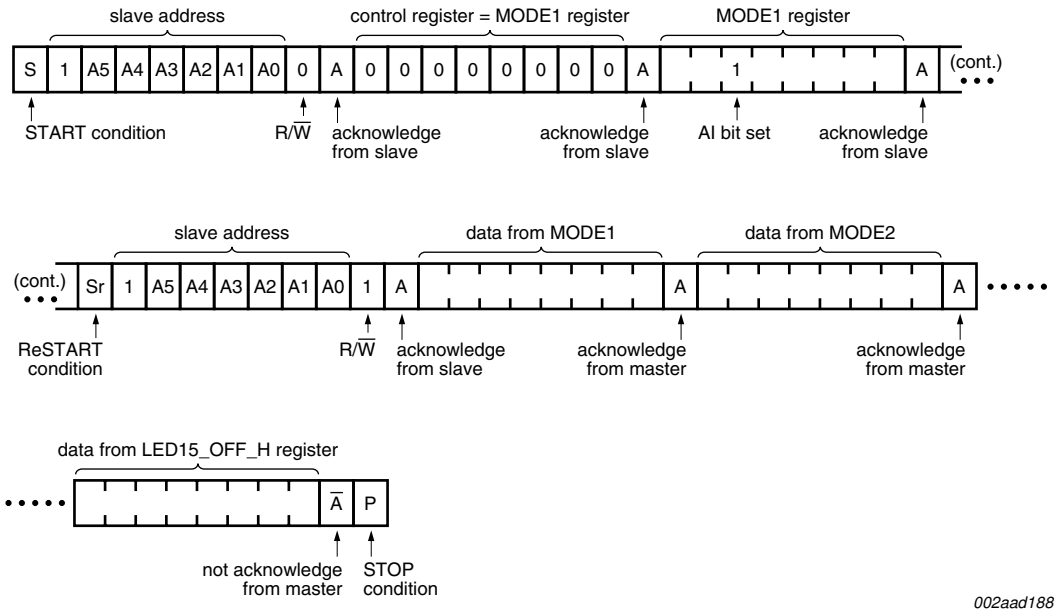


Fig 22. Read all registers using the Auto-Increment feature; AI initially clear

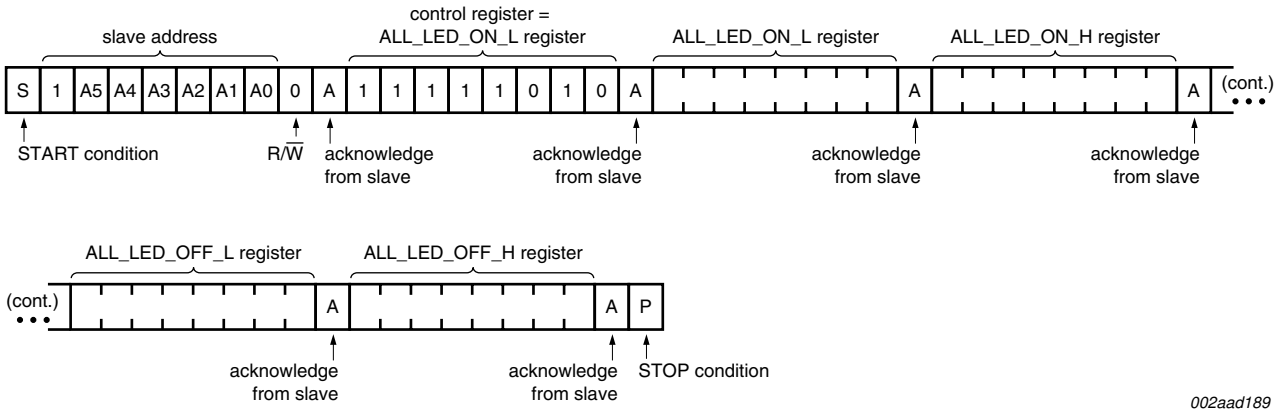


Fig 23. Write to ALL_LED_ON and ALL_LED_OFF registers using the Auto-Increment feature; AI initially set

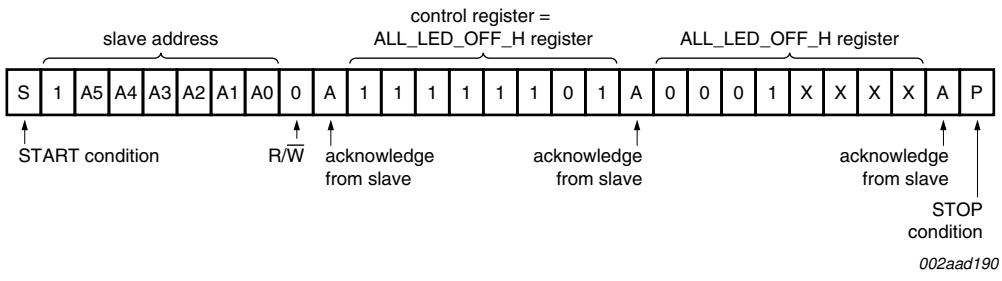
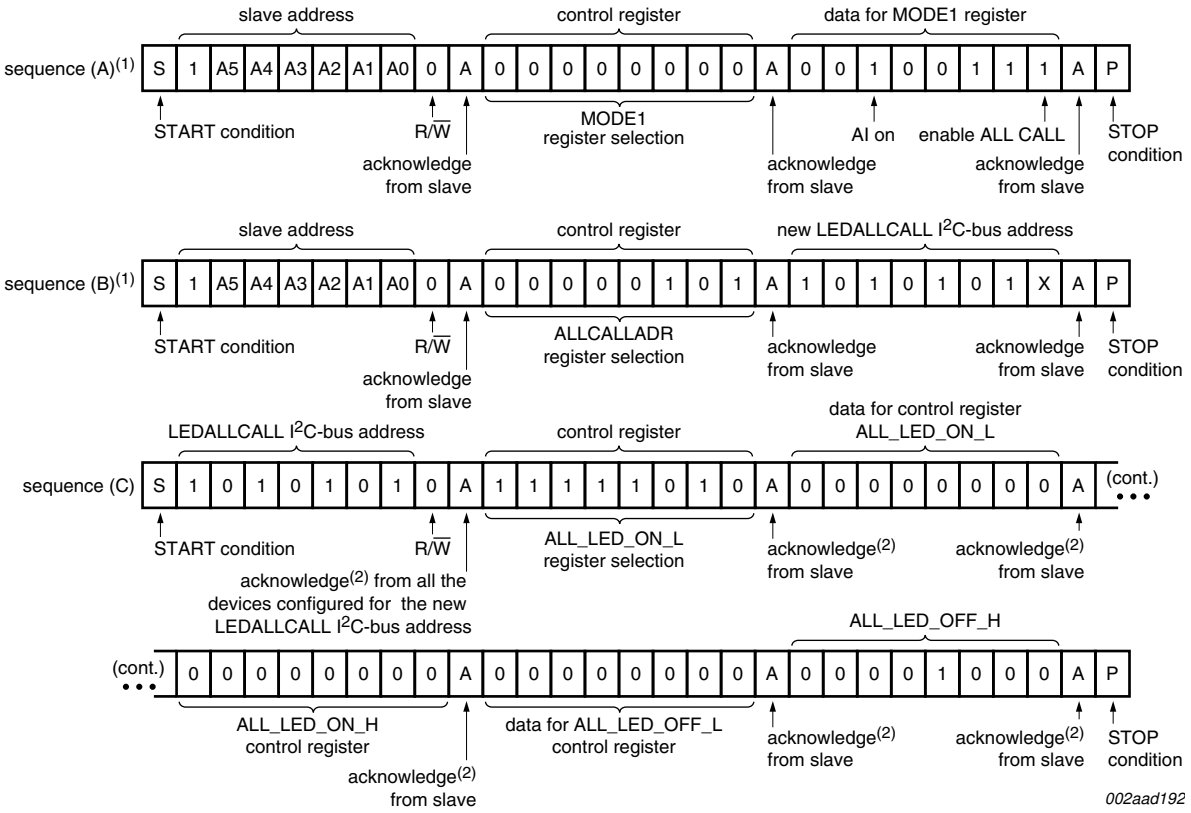


Fig 24. Write to ALL_LED_OFF_H to turn OFF all PWMs

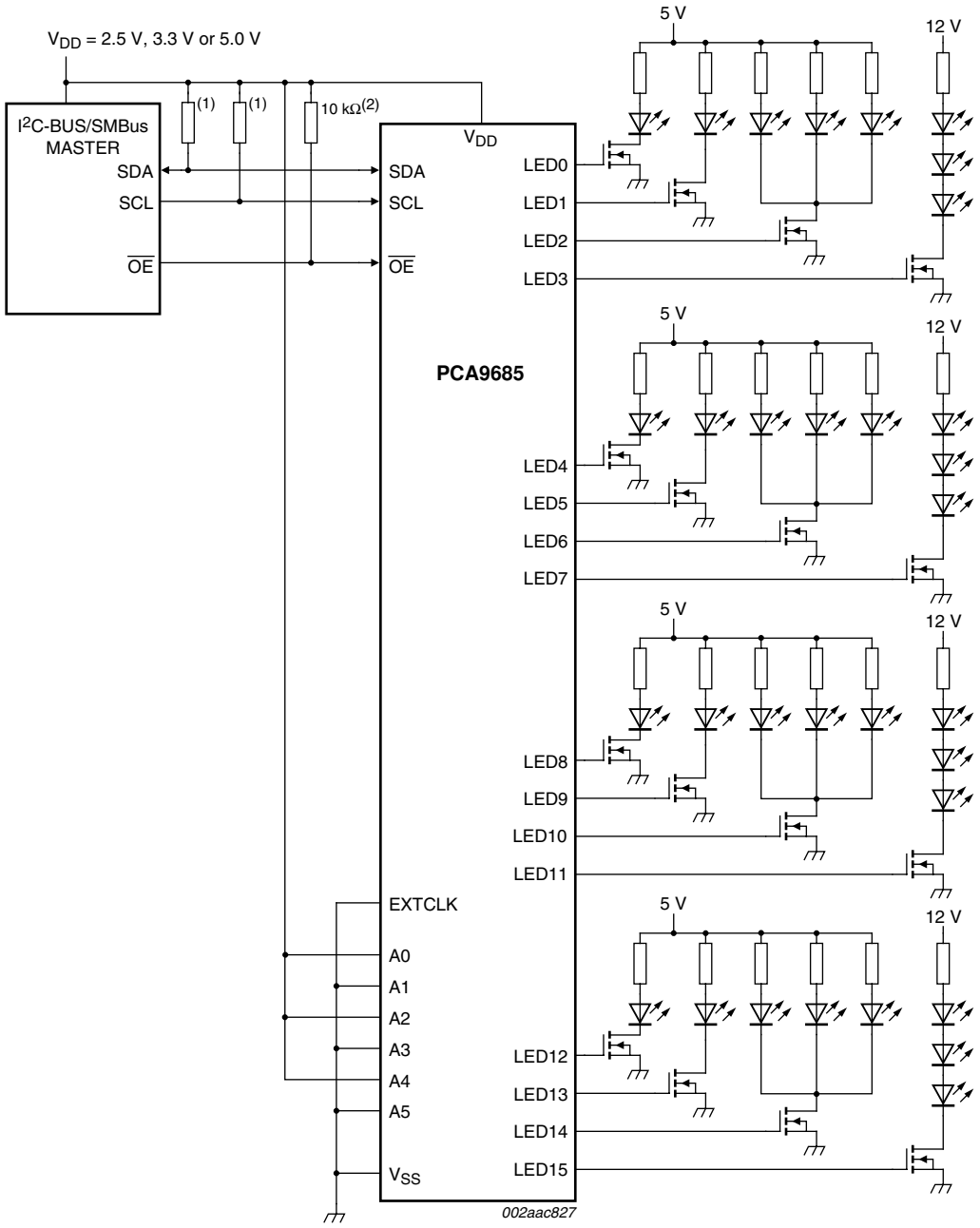


(1) In this example, several PCA9685s are used and the same sequences (A) and (B) above are sent to each of them.

(2) Acknowledge from all the slave devices configured for the new LED All Call I²C-bus address in sequence (B).

Fig 25. LED All Call I²C-bus address programming and LED All Call sequence example

10. Application design-in information



I²C-bus address = 1010 101x.

All 16 of the LEDn outputs configurable as either open-drain or totem pole. Mixing of configuration is not possible.

Remark: Set INVRT = 0, OUTDRV = 1, OUTNE = 01 (MODE2 register bits)

(1) Resistor value should be chosen by referencing section 7 of *UM10204, "I²C-bus specification and user manual"*.

(2) $\overline{OE}$ requires pull-up resistor if control signal from the master is open-drain.

Fig 26. Typical application

Question 1: What kind of edge rate control is there on the outputs?

- The typical edge rates depend on the output configuration, supply voltage, and the applied load. The outputs can be configured as either open-drain NMOS or totem pole outputs. If the customer is using the part to directly drive LEDs, they should be using it in an open-drain NMOS, if they are concerned about the maximum I_{SS} and ground bounce. The edge rate control was designed primarily to slow down the turn-on of the output device; it turns off rather quickly (~1.5 ns). In simulation, the typical turn-on time for the open-drain NMOS was ~14 ns (V_{DD} = 3.6 V; C_L = 50 pF; R_{PU} = 500 Ω).

Question 2: Is ground bounce possible?

- Ground bounce is a possibility, especially if all 16 outputs are changed at full current (25 mA each). There is a fair amount of decoupling capacitance on chip (~50 pF), which is intended to suppress some of the ground bounce. The customer will need to determine if additional decoupling capacitance externally placed as close as physically possible to the device is required.

Question 3: Can I really sink 400 mA through the single ground pin on the package and will this cause any ground bounce problem due to the PWM of the LEDs?

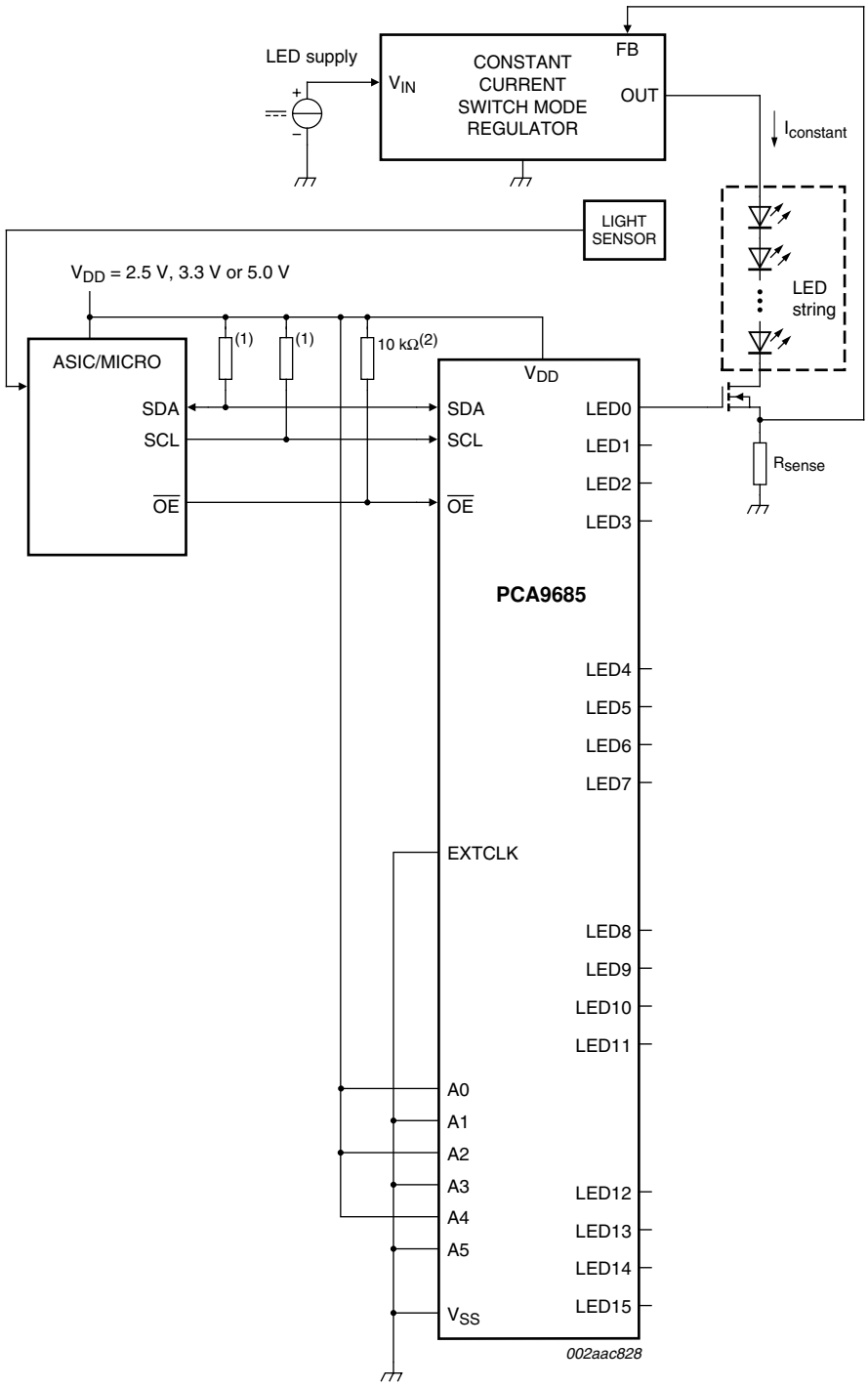
- Yes, you can sink 400 mA through a single ground pin on the **package**. Although the package only has one ground pin, there are two ground pads on the die itself connected to this one pin. Although some ground bounce is likely, it will not disrupt the operation of the part and would be reduced by the external decoupling capacitance.

Question 4: I can't turn the LEDs on or off, but their registers are set properly. Why?

- Check the MODE1 register SLEEP (bit 4) setting. The bit needs to be 0 in order to enable the clocking. If both clock sources (internal osc and EXTCLK) are turned OFF (bit 4 = 1), the LEDs cannot be dimmed or blinked.

Question 5: I'm using LEDs with integrated Zener diodes and the IC is getting very hot. Why?

- The IC outputs can be set to either open-drain or push-pull and default to push-pull outputs. In this application with the Zener diodes, they need to be set to open-drain since in the push-pull architecture there is a low resistance path to GND through the Zener and this is causing the IC to overheat.



I²C-bus address = 1010 101x.

Remark: Set INVRT = 0, OUTDRV = 1, OUTNE = 01 (MODE2 register bits) for this configuration.

(1) Resistor value should be chosen by referencing Section 7 of *UM10204, "I²C-bus specification and user manual"*.

(2) $\overline{OE}$ requires pull-up resistor if control signal from the master is open-drain.

Fig 27. LCD backlighting application

11. Limiting values

Table 12. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134).

Symbol	Parameter	Conditions	Min	Max	Unit
V_{DD}	supply voltage		−0.5	+6.0	V
$V_{I/O}$	voltage on an input/output pin		V_{SS} − 0.5	5.5	V
$I_{O(LEDn)}$	output current on pin LEDn		-	25	mA
I_{SS}	ground supply current		-	400	mA
P_{tot}	total power dissipation		-	400	mW
T_{stg}	storage temperature		−65	+150	°C
T_{amb}	ambient temperature	operating	−40	+85	°C

12. Static characteristics

Table 13. Static characteristics

V_{DD} = 2.3 V to 5.5 V; V_{SS} = 0 V; T_{amb} = −40 °C to +85 °C; unless otherwise specified.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
Supply							
V_{DD}	supply voltage			2.3	-	5.5	V
I_{DD}	supply current	operating mode; no load; f_{SCL} = 1 MHz; V_{DD} = 2.3 V to 5.5 V		-	6	10	mA
I_{stb}	standby current	no load; f_{SCL} = 0 Hz; V_I = V_{DD} or V_{SS}; V_{DD} = 2.3 V to 5.5 V		-	2.2	15.5	µA
V_{POR}	power-on reset voltage	no load; V_I = V_{DD} or V_{SS}	[1]	-	1.70	2.0	V
Input SCL; input/output SDA							
V_{IL}	LOW-level input voltage			−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage			0.7V_{DD}	-	5.5	V
I_{OL}	LOW-level output current	V_{OL} = 0.4 V; V_{DD} = 2.3 V		20	28	-	mA
		V_{OL} = 0.4 V; V_{DD} = 5.0 V		30	40	-	mA
I_L	leakage current	V_I = V_{DD} or V_{SS}		−1	-	+1	µA
C_i	input capacitance	V_I = V_{SS}		-	6	10	pF
LED driver outputs							
I_{OL}	LOW-level output current	V_{OL} = 0.5 V; V_{DD} = 2.3 V to 4.5 V	[2]	12	25	-	mA
$I_{OL(tot)}$	total LOW-level output current	V_{OL} = 0.5 V; V_{DD} = 4.5 V	[2]	-	-	400	mA
I_{OH}	HIGH-level output current	open-drain; V_{OH} = V_{DD}		−10	-	+10	µA
V_{OH}	HIGH-level output voltage	I_{OH} = −10 mA; V_{DD} = 2.3 V		1.6	-	-	V
		I_{OH} = −10 mA; V_{DD} = 3.0 V		2.3	-	-	V
		I_{OH} = −10 mA; V_{DD} = 4.5 V		4.0	-	-	V
I_{OZ}	OFF-state output current	3-state; V_{OH} = V_{DD} or V_{SS}		−10	-	+10	µA
C_o	output capacitance			-	5	8	pF

Table 13. Static characteristics *...continued*

V_{DD} = 2.3 V to 5.5 V; V_{SS} = 0 V; T_{amb} = −40 °C to +85 °C; unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Address inputs; $\overline{OE}$ input; EXTCLK						
V_{IL}	LOW-level input voltage		−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage		0.7V_{DD}	-	5.5	V
I_{LI}	input leakage current		−1	-	+1	μA
C_i	input capacitance		-	3	5	pF

[1] V_{DD} must be lowered to 0.2 V in order to reset part.

[2] Each bit must be limited to a maximum of 25 mA and the total package limited to 400 mA due to internal busing limits.

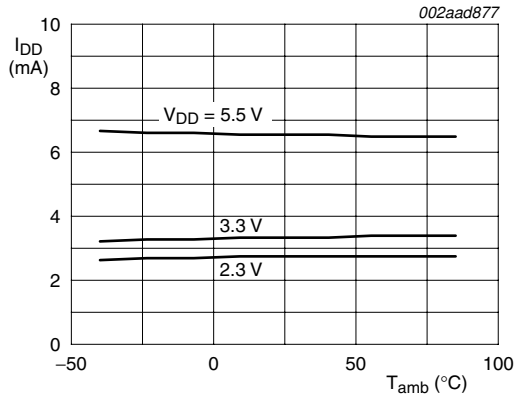


Fig 28. I_{DD} typical values with OSC on and f_{SCL} = 1 MHz versus temperature

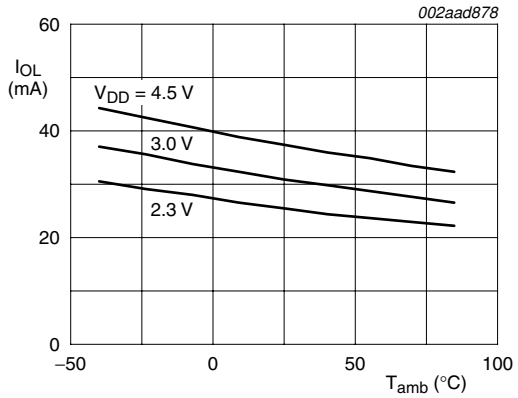


Fig 29. I_{OL} typical drive (LEDn outputs) versus temperature

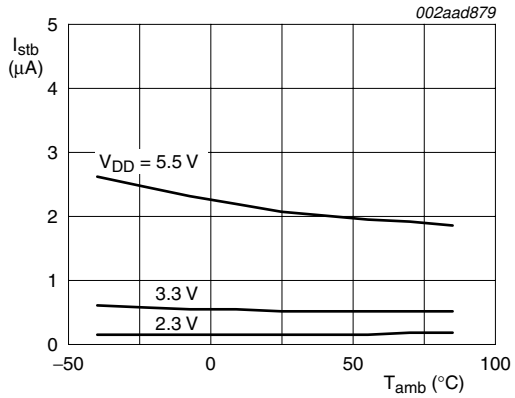


Fig 30. Standby supply current versus temperature

13. Dynamic characteristics

Table 14. Dynamic characteristics

Symbol	Parameter	Conditions		Standard-mode I²C-bus		Fast-mode I²C-bus		Fast-mode Plus I²C-bus		Unit
				Min	Max	Min	Max	Min	Max	
f_{SCL}	SCL clock frequency		[1]	0	100	0	400	0	1000	kHz
f_{EXTCLK}	frequency on pin EXTCLK			DC	50	DC	50	DC	50	MHz
t_{BUF}	bus free time between a STOP and START condition			4.7	-	1.3	-	0.5	-	µs
$t_{HD;STA}$	hold time (repeated) START condition			4.0	-	0.6	-	0.26	-	µs
$t_{SU;STA}$	set-up time for a repeated START condition			4.7	-	0.6	-	0.26	-	µs
$t_{SU;STO}$	set-up time for STOP condition			4.0	-	0.6	-	0.26	-	µs
$t_{HD;DAT}$	data hold time			0	-	0	-	0	-	ns
$t_{VD;ACK}$	data valid acknowledge time		[2]	0.3	3.45	0.1	0.9	0.05	0.45	µs
$t_{VD;DAT}$	data valid time		[3]	0.3	3.45	0.1	0.9	0.05	0.45	µs
$t_{SU;DAT}$	data set-up time			250	-	100	-	50	-	ns
t_{LOW}	LOW period of the SCL clock			4.7	-	1.3	-	0.5	-	µs
t_{HIGH}	HIGH period of the SCL clock			4.0	-	0.6	-	0.26	-	µs
t_f	fall time of both SDA and SCL signals		[4][5]	-	300	$20 + 0.1C_b$ [6]	300	-	120	ns
t_r	rise time of both SDA and SCL signals			-	1000	$20 + 0.1C_b$ [6]	300	-	120	ns
t_{SP}	pulse width of spikes that must be suppressed by the input filter		[7]	-	50	-	50	-	50	ns
t_{PLZ}	LOW to OFF-state propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register		-	40	-	40	-	40	ns
t_{PZL}	OFF-state to LOW propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register		-	60	-	60	-	60	ns
t_{PHZ}	HIGH to OFF-state propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register		-	60	-	60	-	60	ns

Table 14. Dynamic characteristics *…continued*

Symbol	Parameter	Conditions	Standard-mode I²C-bus		Fast-mode I²C-bus		Fast-mode Plus I²C-bus		Unit
			Min	Max	Min	Max	Min	Max	
t_{PZH}	OFF-state to HIGH propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register	-	40	-	40	-	40	ns
t_{PLH}	LOW to HIGH propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 01 in MODE2 register	-	40	-	40	-	40	ns
t_{PHL}	HIGH to LOW propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 00 in MODE2 register	-	60	-	60	-	60	ns

[1] Minimum SCL clock frequency is limited by the bus time-out feature, which resets the serial bus interface if either SDA or SCL is held LOW for a minimum of 25 ms. Disable bus time-out feature for DC operation.

[2] $t_{VD;ACK}$ = time for Acknowledgement signal from SCL LOW to SDA (out) LOW.

[3] $t_{VD;DAT}$ = minimum time for SDA data out to be valid following SCL LOW.

[4] A master device must internally provide a hold time of at least 300 ns for the SDA signal (refer to the V_{IL} of the SCL signal) in order to bridge the undefined region of SCL's falling edge.

[5] The maximum t_f for the SDA and SCL bus lines is specified at 300 ns. The maximum fall time (t_f) for the SDA output stage is specified at 250 ns. This allows series protection resistors to be connected between the SDA and the SCL pins and the SDA/SCL bus lines without exceeding the maximum specified t_f.

[6] C_b = total capacitance of one bus line in pF.

[7] Input filters on the SDA and SCL inputs suppress noise spikes less than 50 ns.

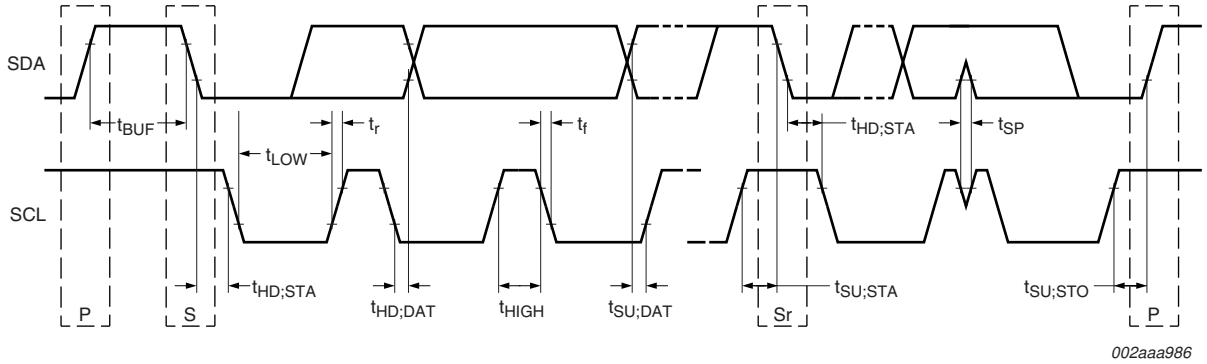
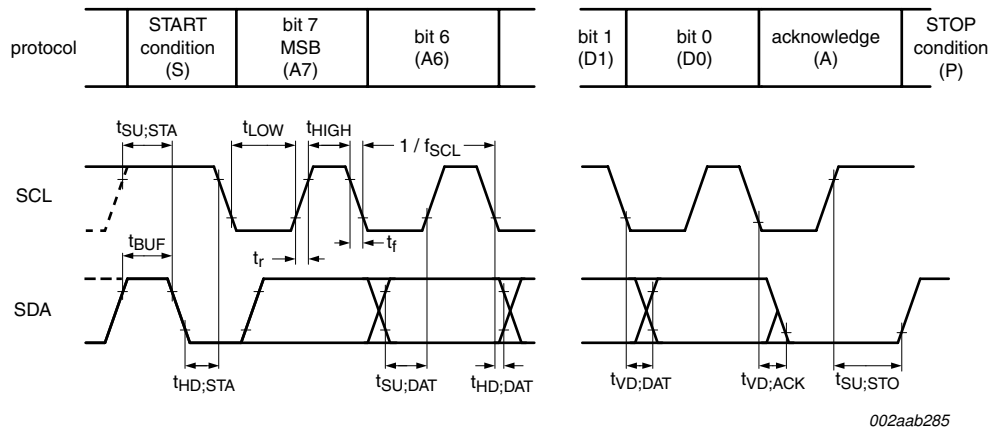


Fig 31. Definition of timing



Rise and fall times refer to V_{IL} and V_{IH}.

Fig 32. I²C-bus timing diagram

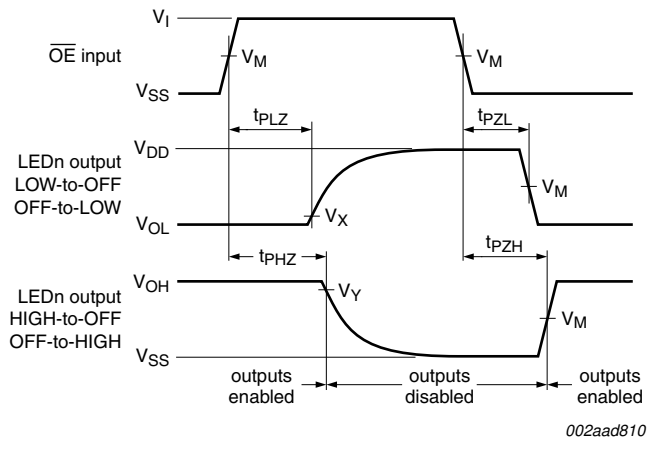
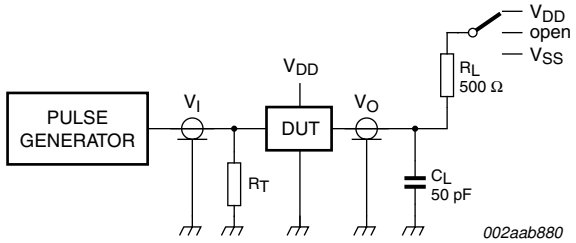


Fig 33. t_{PLZ}, t_{PZL} and t_{PHZ}, t_{PZH} times

14. Test information

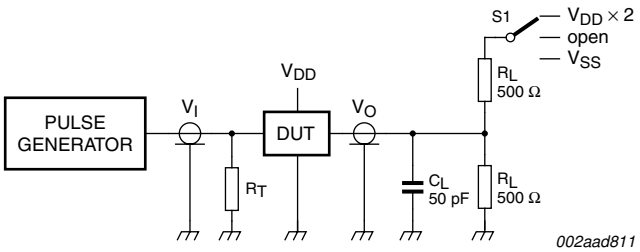


R_L = Load resistor for LEDn.

C_L = Load capacitance includes jig and probe capacitance.

R_T = Termination resistance should be equal to the output impedance Z_o of the pulse generators.

Fig 34. Test circuitry for switching times



R_L = Load resistor for LEDn.

C_L = Load capacitance includes jig and probe capacitance.

R_T = Termination resistance should be equal to the output impedance Z_o of the pulse generators.

Test data are given in Table 15.

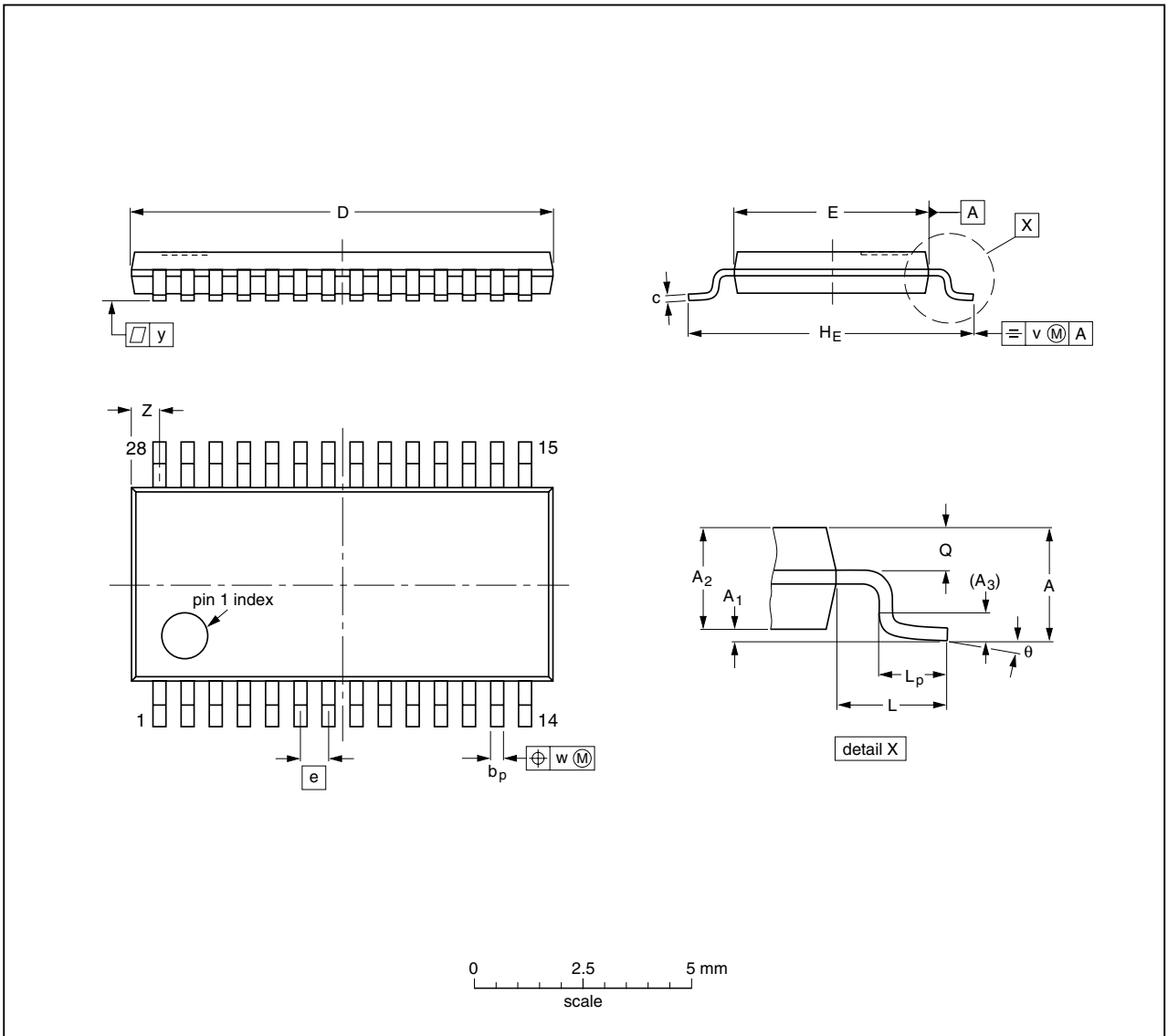
Fig 35. Test circuitry for switching times for enable/disable

Table 15. Test data for enable/disable switching times

Test	Load		Switch
	C_L	R_L	
t_{PD}	50 pF	500 Ω	open
t_{PLZ}, t_{PZL}	50 pF	500 Ω	$V_{DD} \times 2$
t_{PHZ}, t_{PZH}	50 pF	500 Ω	V_{SS}

15. Package outline

TSSOP28: plastic thin shrink small outline package; 28 leads; body width 4.4 mm **SOT361-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	D (1)	E (2)	e	H_E	L	L_p	Q	v	w	y	Z (1)	θ
mm	1.1	0.15 0.05	0.95 0.80	0.25	0.30 0.19	0.2 0.1	9.8 9.6	4.5 4.3	0.65	6.6 6.2	1	0.75 0.50	0.4 0.3	0.2	0.13	0.1	0.8 0.5	8° 0°

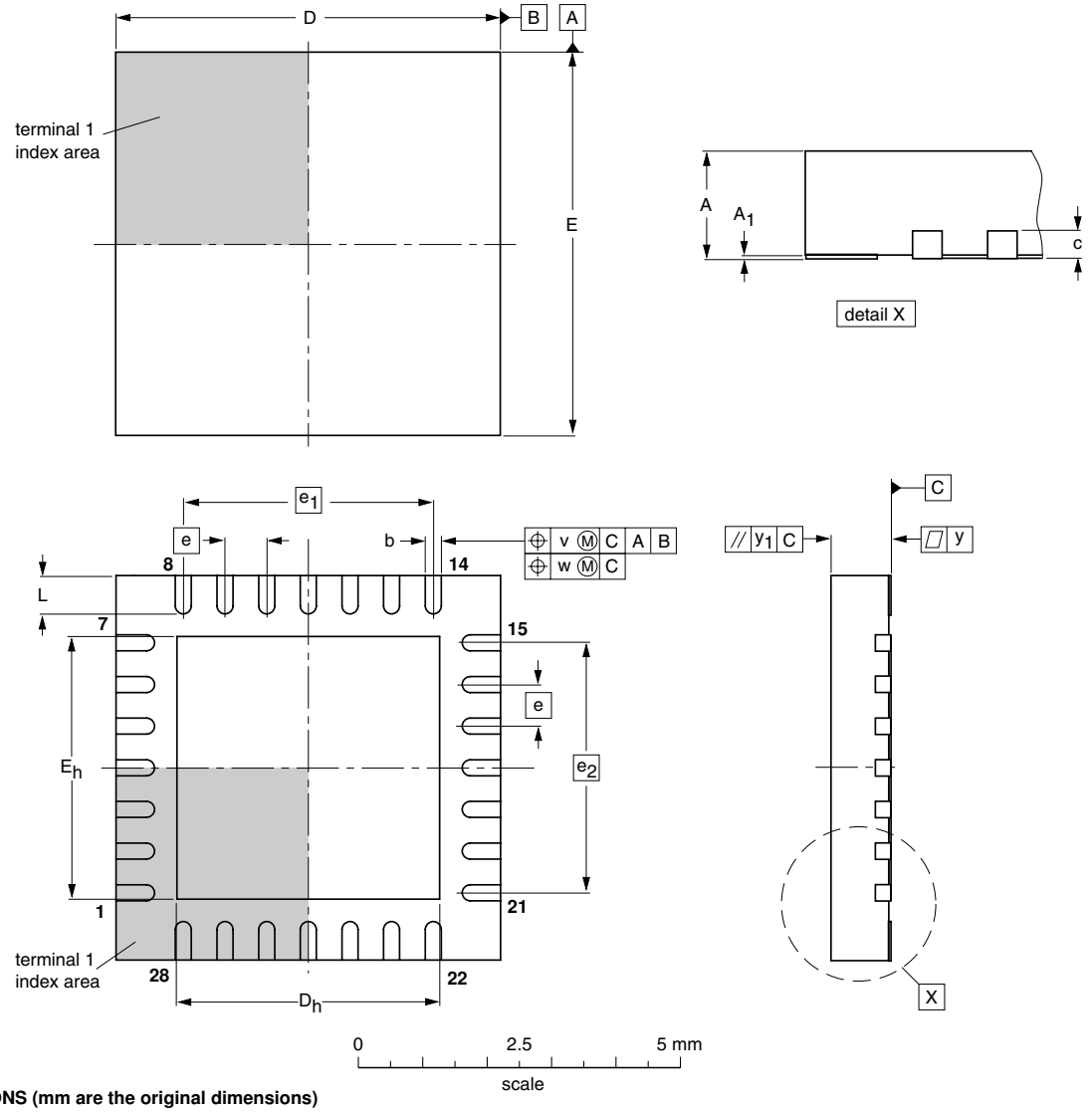
Notes

1. Plastic or metal protrusions of 0.15 mm maximum per side are not included.
2. Plastic interlead protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT361-1		MO-153				~~99-12-27~~ 03-02-19

Fig 36. Package outline SOT361-1 (TSSOP28)

HVQFN28: plastic thermal enhanced very thin quad flat package; no leads; 28 terminals; body 6 x 6 x 0.85 mm **SOT788-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A(1) max.	A_1	b	c	D(1)	D_h	E(1)	E_h	e	e_1	e_2	L	v	w	y	y_1
mm	1	0.05 0.00	0.35 0.25	0.2	6.1 5.9	4.25 3.95	6.1 5.9	4.25 3.95	0.65	3.9	3.9	0.75 0.50	0.1	0.05	0.05	0.1

Note

1. Plastic or metal protrusions of 0.075 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT788-1	- - -	MO-220	- - -			02-10-22

Fig 37. Package outline SOT788-1 (HVQFN28)

16. Handling information

All input and output pins are protected against ElectroStatic Discharge (ESD) under normal handling. When handling ensure that the appropriate precautions are taken as described in *JESD625-A* or equivalent standards.

17. Soldering of SMD packages

This text provides a very brief insight into a complex technology. A more in-depth account of soldering ICs can be found in Application Note *AN10365 "Surface mount reflow soldering description"*.

17.1 Introduction to soldering

Soldering is one of the most common methods through which packages are attached to Printed Circuit Boards (PCBs), to form electrical circuits. The soldered joint provides both the mechanical and the electrical connection. There is no single soldering method that is ideal for all IC packages. Wave soldering is often preferred when through-hole and Surface Mount Devices (SMDs) are mixed on one printed wiring board; however, it is not suitable for fine pitch SMDs. Reflow soldering is ideal for the small pitches and high densities that come with increased miniaturization.

17.2 Wave and reflow soldering

Wave soldering is a joining technology in which the joints are made by solder coming from a standing wave of liquid solder. The wave soldering process is suitable for the following:

- Through-hole components
- Leaded or leadless SMDs, which are glued to the surface of the printed circuit board

Not all SMDs can be wave soldered. Packages with solder balls, and some leadless packages which have solder lands underneath the body, cannot be wave soldered. Also, leaded SMDs with leads having a pitch smaller than ~0.6 mm cannot be wave soldered, due to an increased probability of bridging.

The reflow soldering process involves applying solder paste to a board, followed by component placement and exposure to a temperature profile. Leaded packages, packages with solder balls, and leadless packages are all reflow solderable.

Key characteristics in both wave and reflow soldering are:

- Board specifications, including the board finish, solder masks and vias
- Package footprints, including solder thieves and orientation
- The moisture sensitivity level of the packages
- Package placement
- Inspection and repair
- Lead-free soldering versus SnPb soldering

17.3 Wave soldering

Key characteristics in wave soldering are:

- Process issues, such as application of adhesive and flux, clinching of leads, board transport, the solder wave parameters, and the time during which components are exposed to the wave
- Solder bath specifications, including temperature and impurities

17.4 Reflow soldering

Key characteristics in reflow soldering are:

- Lead-free versus SnPb soldering; note that a lead-free reflow process usually leads to higher minimum peak temperatures (see Figure 38) than a SnPb process, thus reducing the process window
- Solder paste printing issues including smearing, release, and adjusting the process window for a mix of large and small components on one board
- Reflow temperature profile; this profile includes preheat, reflow (in which the board is heated to the peak temperature) and cooling down. It is imperative that the peak temperature is high enough for the solder to make reliable solder joints (a solder paste characteristic). In addition, the peak temperature must be low enough that the packages and/or boards are not damaged. The peak temperature of the package depends on package thickness and volume and is classified in accordance with Table 16 and 17

Table 16. SnPb eutectic process (from J-STD-020C)

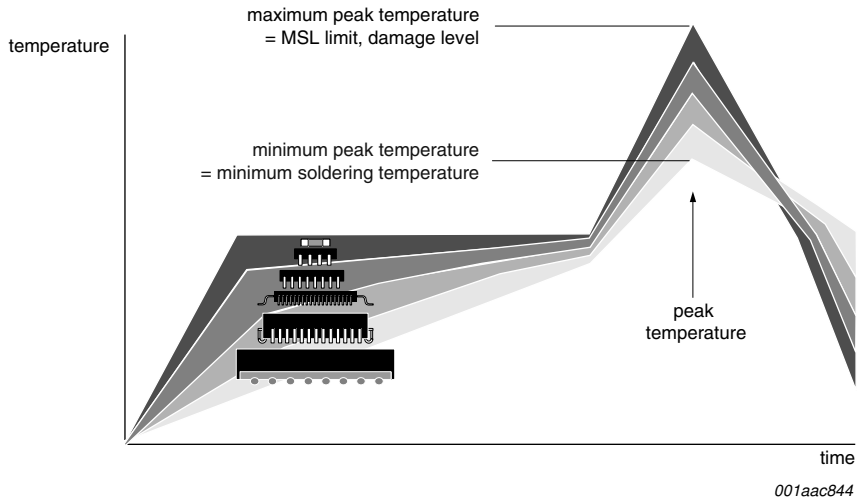
Package thickness (mm)	Package reflow temperature (°C)	
	Volume (mm³)	
	< 350	≥ 350
< 2.5	235	220
≥ 2.5	220	220

Table 17. Lead-free process (from J-STD-020C)

Package thickness (mm)	Package reflow temperature (°C)		
	Volume (mm³)		
	< 350	350 to 2000	> 2000
< 1.6	260	260	260
1.6 to 2.5	260	250	245
> 2.5	250	245	245

Moisture sensitivity precautions, as indicated on the packing, must be respected at all times.

Studies have shown that small packages reach higher temperatures during reflow soldering, see Figure 38.



MSL: Moisture Sensitivity Level

Fig 38. Temperature profiles for large and small components

For further information on temperature profiles, refer to Application Note *AN10365 "Surface mount reflow soldering description"*.

18. Abbreviations

Table 18. Abbreviations

Acronym	Description
CDM	Charged-Device Model
DUT	Device Under Test
EMI	ElectroMagnetic Interference
ESD	ElectroStatic Discharge
HBM	Human Body Model
I²C-bus	Inter-Integrated Circuit bus
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
MM	Machine Model
MSB	Most Significant Bit
NMOS	Negative-channel Metal-Oxide Semiconductor
PCB	Printed-Circuit Board
PMOS	Positive-channel Metal-Oxide Semiconductor
POR	Power-On Reset
PWM	Pulse Width Modulation; Pulse Width Modulator
RGB	Red/Green/Blue
RGBA	Red/Green/Blue/Amber
SMBus	System Management Bus

19. Revision history

Table 19. Revision history

Document ID	Release date	Data sheet status	Change notice	Supersedes
PCA9685 v.3	20100902	Product data sheet	-	PCA9685 v.2
Modifications:	• Table 1 “Ordering information”: Topside mark for PCA9685BS changed from “PCA9685BS” to “P9685”			
PCA9685 v.2	20090716	Product data sheet	-	PCA9685 v.1
PCA9685 v.1	20080724	Product data sheet	-	-

20. Legal information

20.1 Data sheet status

Document status[1][2]	Product status[3]	Definition
Objective [short] data sheet	Development	This document contains data from the objective specification for product development.
Preliminary [short] data sheet	Qualification	This document contains data from the preliminary specification.
Product [short] data sheet	Production	This document contains the product specification.

[1] Please consult the most recently issued document before initiating or completing a design.

[2] The term 'short data sheet' is explained in section "Definitions".

[3] The product status of device(s) described in this document may have changed since this document was published and may differ in case of multiple devices. The latest product status information is available on the Internet at URL http://www.nxp.com.

20.2 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

Short data sheet — A short data sheet is an extract from a full data sheet with the same product type number(s) and title. A short data sheet is intended for quick reference only and should not be relied upon to contain detailed and full information. For detailed and full information see the relevant full data sheet, which is available on request via the local NXP Semiconductors sales office. In case of any inconsistency or conflict with the short data sheet, the full data sheet shall prevail.

Product specification — The information and data provided in a Product data sheet shall define the specification of the product as agreed between NXP Semiconductors and its customer, unless NXP Semiconductors and customer have explicitly agreed otherwise in writing. In no event however, shall an agreement be valid in which the NXP Semiconductors product is deemed to offer functions and qualities beyond those described in the Product data sheet.

20.3 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

Non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

20.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I²C-bus — logo is a trademark of NXP B.V.

21. Contact information

For more information, please visit: **http://www.nxp.com**

For sales office addresses, please send an email to: **salesaddresses@nxp.com**

22. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2 September 2010
Document identifier: PCA9685

ANEXO C: SCRIPT EN MATLAB PARA LA CALIBRACIÓN DE LOS MOTORES

Graficas_potencia.m

```
%%Vectores contienen del 50% al 100% de potencia.

Empuje_M_gris_1 = [210 220 245 261 280 310 340 371 442 470 543];
Empuje_M_negro_1 = [198 212 240 259 272 298 326 360 400 426 480];
Empuje_M_gris_2 = [208 224 245 255 280 300 330 360 399 440 500];
Empuje_M_negro_2 = [196 210 230 248 268 291 320 348 383 430 490];

Empuje_M_gris_1_corr = [210 220 245 261 280 310 340 371 442 470 543]./1.06;
Empuje_M_negro_1_corr = [198 212 240 259 272 298 326 360 400 426 480]./1.02;
Empuje_M_gris_2_corr = [208 224 245 255 280 300 330 360 399 440 500]./1.03;
Empuje_M_negro_2_corr = [196 210 230 248 268 291 320 348 383 430 490];

Ciclo_trabajo = [50 55 60 65 70 75 80 85 90 95 100];

figure, plot(Ciclo_trabajo, Empuje_M_gris_1, 'g')
hold on, plot(Ciclo_trabajo,Empuje_M_negro_1,'r')
hold on, plot(Ciclo_trabajo,Empuje_M_gris_2,'m')
hold on, plot(Ciclo_trabajo,Empuje_M_negro_2,'b')
legend('motor gris 1',' motor negro 1','motor gris 2',' motor negro 2');
xlabel('Ciclo PWM (%)');
ylabel('Empuje (g)');


figure ,plot(Ciclo_trabajo, Empuje_M_gris_1_corr, 'g')
hold on, plot(Ciclo_trabajo,Empuje_M_negro_1_corr,'r')
hold on, plot(Ciclo_trabajo,Empuje_M_gris_2_corr,'m')
hold on, plot(Ciclo_trabajo,Empuje_M_negro_2_corr,'b')
legend('motor gris 1',' motor negro 1','motor gris 2',' motor negro 2');
xlabel('Ciclo PWM (%)');
ylabel('Empuje (g)');


% % syms a b c
% % [a,b,c]=solve('3025*a+55*b+c=220','4900*a+70*b+c=280','9025*a+95*b+c=470')
% %
% % for i=1:11
% %     parabola_ideal(i) = funcion_parabola(Ciclo_trabajo(i));
% %
% % end
% %
% % plot(Ciclo_trabajo, parabola_ideal, 'k')
```

Función_parabola.m

```
function [y] = parabola(x)

y = (9/100)*x^2 - (29/4)*x + 693/2;

end
```

INDICE DEL ANEXO D

i2c.h

```
/*
 *  i2c.h
 *
 *  Variables utilizadas en i2c.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"
#include "runtime.h"

#define ID_i2c "i2c"
#define MAX_i2c 16

struct i2c {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo;
   int t_ciclo_min;
   int t_ciclo_max;

   // variables

   // entradas

   // salidas
   int bus_ok;
   int fd;
};
```

i2c.c

```
/*
 *  i2c.c
 *
 *  Abre el bus I2C.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
```

```
#include <time.h>

#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/i2c-dev.h>

#include "runtime.h"
#include "i2c.h"

#ifdef ID_LISTAS
struct lista instancias_i2c;
#else
#define MAX_i2c 16
struct i2c instancias_i2c[MAX_i2c];
#endif

struct componente clase_i2c;

extern char msgLog[];

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_i2c(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct i2c *este = (struct i2c *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas i2c

  // fin inicialización de propiedades específicas i2c
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_i2c(void *instancia) {
  struct i2c *este = (struct i2c *)instancia;

  char saux[MAX_LONG_NOMBRE];
```

```
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_i2c),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo");
  insertarPropiedad3(este->nombre, &este->t_ciclo_min, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_max, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_max");

  // principio registro de propiedades específicas i2c
  insertarPropiedad3(este->nombre, &este->bus_ok, ID_SAL_INT, PUBLICO |
MODIFICABLE, "bus_ok");
  insertarPropiedad3(este->nombre, &este->fd, ID_SAL_INT, PUBLICO |
MODIFICABLE, "fd");

  // fin registro de propiedades específicas i2c
}

/*****************************************************************
*  void funcionCrea()
*****************************************************************/
int funcion_crea_i2c(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct i2c *este = (struct i2c *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_i2c(instancia, nombre, orden, habilitado);

  if(registra_propiedades_i2c(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_i2c, orden,
                        &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
```

```
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_i2c(void *instancia) {
  struct i2c *este = (struct i2c *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_min = 0x10000000;
  este->t_ciclo_max = 0;
  este->bus_ok = 0;

  // principio

 if ((este->fd = open("/dev/i2c-1", O_RDWR)) < 0){

    #ifdef LOG
    sprintf(msgLog, "%s \"Failed to open i2c bus", ID_ERROR);
    logPrint(msgLog, 1);
    #endif
    exit(1);
  }

  // fin
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_i2c(void *instancia, int t_ciclo) {
  struct i2c *este = (struct i2c *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio funcion normal i2c

  // fin funcion normal i2c

  este->t_ciclo = crononsec(0, &ts);
```

```
  if(este->t_ciclo > este->t_ciclo_max) {
    este->t_ciclo_max = este->t_ciclo;
  }
  else if(este->t_ciclo < este->t_ciclo_min) {
    este->t_ciclo_min = este->t_ciclo;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_i2c(void *instancia, int t_ciclo) {
  struct i2c *este = (struct i2c *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT i2c

  // fin código específico funcion normal noRT i2c

  este->t_ciclo = crononsec(0, &ts);

  if(este->t_ciclo > este->t_ciclo_max) {
    este->t_ciclo_max = este->t_ciclo;
  }
  else if(este->t_ciclo < este->t_ciclo_min) {
    este->t_ciclo_min = este->t_ciclo;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_i2c(void *instancia) {
  struct i2c *este = (struct i2c *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio
//  close(este->fd);
  // fin
}

/*************************************************************
*  void inicilizai2c()
*************************************************************/
```

```
int inicializa_i2c() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_i2c);
  logPrint(msgLog, 3);
#endif

  clase_i2c.funcionCrea = funcion_crea_i2c;
  clase_i2c.funcionInicializa = funcion_inicializa_i2c;
  clase_i2c.funcionNormal = funcion_normal_i2c;
//  clase_i2c.funcionNormalNoRT = funcion_normal_noRT_i2c;
  clase_i2c.funcionFinaliza = funcion_finaliza_i2c;

#ifdef ID_LISTAS
  iniciarLista(&instancias_i2c);
#else
  clase_i2c.n = 0;
  clase_i2c.maxNumComp = MAX_i2c;
#endif

 clase_i2c.instancias = &instancias_i2c;
 clase_i2c.longComponente = sizeof(struct i2c);

 return insertarPropiedad2(ID_COMPONENTE, ID_i2c, &clase_i2c,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

ACELERÓMETRO

acelerometro.h

```
/*
 *  acelerometro.h
 *
 *  Variables ADXL345
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_acelerometro "acelerometro"
#define MAX_acelerometro 16
#define MAX_EJES 3
#define ADXL345_I2C_ADDR 0x53

#define ADXL345_POWER_CTL 0x2d
#define ADXL345_DATA_FORMAT 0x31

struct acelerometro {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   short x,y,z;
   unsigned char buf[8];
   int i;

   /*Niveles de offset del acelerometro*/
   float offset_acc_x;
   float offset_acc_y;
   float offset_acc_z;
   // entradas
   int *bus_ok;
   int *fd;

   // salidas
   float Facc[MAX_EJES]; //Vector datos acelerometro
```

```
};
```

acelerometro.c

```
*
 *  acelerometro.c
 *
 *  Captura de mediciones del sensor ADXL345
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#include <termios.h>
#include "runtime.h"
#include "acelerometro.h"
#include "linux/i2c-dev.h"

#ifdef ID_LISTAS
struct lista instancias_acelerometro;
#else
#define MAX_acelerometro 16
struct acelerometro instancias_acelerometro[MAX_acelerometro];
#endif

struct componente clase_acelerometro;

extern char msgLog[];

int dummy = 0;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_acelerometro(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct acelerometro *este = (struct acelerometro *)instancia;
```

```
  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas acelerometro

  este->fd = &dummy;
  este->bus_ok = &dummy;

  // fin inicialización de propiedades específicas acelerometro
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_acelerometro(void *instancia) {
  struct acelerometro *este = (struct acelerometro *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int i;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_acelerometro), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas acelerometro

  insertarPropiedad3(este->nombre, &este->bus_ok, ID_ENT_INT, PUBLICO
|MODIFICABLE, "bus_ok");
```

```
  insertarPropiedad3(este->nombre, &este->fd, ID_ENT_INT, PUBLICO
|MODIFICABLE, "fd");

  for(i = 0; i < MAX_EJES; i++)
    insertarPropiedad3(este->nombre, &este->Facc[i], ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "Facc[%d]", i);

  insertarPropiedad3(este->nombre, &este->buf[0], ID_VAR_FLOAT, PUBLICO |
NO_MODIFICABLE, "buf[0]");
  insertarPropiedad3(este->nombre, &este->buf[1], ID_VAR_FLOAT, PUBLICO |
NO_MODIFICABLE, "buf[1]");
  insertarPropiedad3(este->nombre, &este->buf[2], ID_VAR_FLOAT, PUBLICO |
NO_MODIFICABLE, "buf[2]");
  insertarPropiedad3(este->nombre, &este->buf[3], ID_VAR_FLOAT, PUBLICO |
NO_MODIFICABLE, "buf[3]");

  insertarPropiedad3(este->nombre, &este->offset_acc_x, ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "offset_acc_x");
  insertarPropiedad3(este->nombre, &este->offset_acc_y, ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "offset_acc_y");
  insertarPropiedad3(este->nombre, &este->offset_acc_z, ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "offset_acc_z");

  // fin registro de propiedades específicas acelerometro
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_acelerometro(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct acelerometro *este = (struct acelerometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_acelerometro(instancia, nombre, orden, habilitado);

  if(registra_propiedades_acelerometro(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_acelerometro, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
```

```
   logPrint(msgLog, 1);
#endif
  }
}

void writetodisp (int fd, int reg, int val){
    char buf[2];
    buf[0] = reg;
    buf[1] = val;
    if (write (fd, buf, 2) != 2){
        fprintf(stderr, "No se puede escribir en el dispositivo i2c\n");
    }
}

/*********************************************************
*  void funcionInicializa()
*********************************************************/
void funcion_inicializa_acelerometro(void *instancia) {
  struct acelerometro *este = (struct acelerometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  este->offset_acc_x = 0.08;
  este->offset_acc_y = 0.04;
  este->offset_acc_z = 0.02;

  // principio código específico funcion inicializa acelerometro

  if (ioctl (*este->fd, I2C_SLAVE, ADXL345_I2C_ADDR) < 0){
     fprintf(stderr, "Acelerometer is not present\n");
  }
   writetodisp(*este->fd, ADXL345_POWER_CTL, 0x00);
   writetodisp(*este->fd, ADXL345_POWER_CTL, 0x08);  //Habilitamos el modo de
medicion continua
   writetodisp(*este->fd, ADXL345_DATA_FORMAT, 0x00);
   writetodisp(*este->fd, ADXL345_DATA_FORMAT, 0x01); // Configuramos el
modo de 10 bits y +-4g

  // fin código específico funcion inicializa acelerometro
}

/*********************************************************
*  void funcionNormal()
*********************************************************/
```

```
void funcion_normal_acelerometro(void *instancia, int t_ciclo) {
  struct acelerometro *este = (struct acelerometro *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal acelerometro


     if (ioctl (*este->fd, I2C_SLAVE, ADXL345_I2C_ADDR) < 0 ){
        fprintf(stderr, "Acelerometer is not present\n");
     }

     este->buf[0] = 0x32;    //Posicion del primer registro.

     if (write(*este->fd, este->buf, 1) != 1) {
        fprintf(stderr, "Error al escribir en el esclavo i2c\n");
     }

     if (read(*este->fd, este->buf, 6) != 6){
        fprintf(stderr, "Unable to read from ADXL345\n");
     }
     else{
        este->x = (este->buf[1] << 8) | este->buf[0];
        este->y = (este->buf[3] << 8) | este->buf[2];
        este->z = (este->buf[5] << 8) | este->buf[4];
        este->Facc[0] = (0.0078 * (float) este->x) - este->offset_acc_x;
        este->Facc[1] = (0.0075 * (float) este->y) - este->offset_acc_y;
        este->Facc[2] = (0.0078 * (float) este->z) - este->offset_acc_z;

        if(fabs(este->Facc[0]) >  0.02){
          este->Facc[0] = este->Facc[0];
        }else{
          este->Facc[0] = 0.00;
        }

        if(fabs(este->Facc[1]) < 0.02){
          este->Facc[1] = 0.00;
        }else{
          este->Facc[1] =  este->Facc[1];
        }

        if (fabs(este->Facc[2]) > 0.98){
          este->Facc[2] = 1.00;
        }else{
          este->Facc[2] =  este->Facc[2];
        }
```

```
    }

  // fin código específico funcion normal acelerometro
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_acelerometro(void *instancia, int t_ciclo) {
  struct acelerometro *este = (struct acelerometro *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT acelerometro

  // fin código específico funcion normal noRT acelerometro
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_acelerometro(void *instancia) {
  struct acelerometro *este = (struct acelerometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza acelerometro

  este->Facc[3] = 0;
```

```
  // fin código específico funcion finaliza acelerometro
}

/*************************************************************
*  void inicilizaacelerometro()
*************************************************************/
int inicializa_acelerometro() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_acelerometro);
  logPrint(msgLog, 3);
#endif

  clase_acelerometro.funcionCrea = funcion_crea_acelerometro;
  clase_acelerometro.funcionInicializa = funcion_inicializa_acelerometro;
  clase_acelerometro.funcionNormal = funcion_normal_acelerometro;
//  clase_acelerometro.funcionNormalNoRT = funcion_normal_noRT_acelerometro;
  clase_acelerometro.funcionFinaliza = funcion_finaliza_acelerometro;

#ifdef ID_LISTAS
  iniciarLista(&instancias_acelerometro);
#else
  clase_acelerometro.n = 0;
  clase_acelerometro.maxNumComp = MAX_acelerometro;
#endif

 clase_acelerometro.instancias = &instancias_acelerometro;
 clase_acelerometro.longComponente = sizeof(struct acelerometro);

 return insertarPropiedad2(ID_COMPONENTE, ID_acelerometro,
&clase_acelerometro, ID_COMPONENTE, NO_MODIFICABLE);
}
```

GIROSCOPO

giroscopo.h

```
/*
 *  giroscopo.h
 *
 *  Variables utilizadas en giroscopo.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_giroscopo "giroscopo"
#define MAX_giroscopo 16

#define ITG3200_I2C_ADDR 0x68
#define ITG3200_PWR_MGM 0x3e
#define ITG3200_SMPLRT_DIV 0x15
#define ITG3200_DLPF_FS 0x16
#define ITG3200_INT_CFG 0x17
#define ITG3200_GYRO_XOUT_H 0x1d
#define ITG3200_GYRO_YOUT_H 0x1f
#define ITG3200_GYRO_ZOUT_H 0x21

#define MAX_ERR_MSG  100
#define MAX_EJES  3
#define LSB 14.375
#define GRADOS_RADIANES 0.017453293
#define ID_SAL_GIRO 0
#define ID_ENT_GIRO 1


struct giroscopo {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   int cont;
   char error_msg_giro[MAX_ERR_MSG];
```

```
    float T, offsetGyro_x, offsetGyro_y, offsetGyro_z;
    short x, y, z;
    unsigned char buf[8], flag[3];
    float CalibGyroX, CalibGyroY, CalibGyroZ;
    float Calib_antX, Calib_antY, Calib_antZ;
    float varX, varY, varZ;
    int flagg;

    // entradas
    int *bus_ok, *fd;

    // salidas
    float Wgyr[MAX_EJES];
    int calibracion_giro;
};
```

giroscopo.c

```
/*
 *  giroscopo.c
 *
 *  Obtencion de medidas del sensor ITG3200
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <sys/stat.h>

#include "local-i2c-dev.h"
#include <linux/types.h>

#include "runtime.h"
#include "giroscopo.h"

#ifdef ID_LISTAS
struct lista instancias_giroscopo;
#else
#define MAX_giroscopo 16
struct giroscopo instancias_giroscopo[MAX_giroscopo];
#endif

struct componente clase_giroscopo;
```

```
extern char msgLog[];

extern int dummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_giroscopo(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {

  struct giroscopo *este = (struct giroscopo *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas giroscopo
  este->fd = &dummy;
  este->bus_ok = &dummy;
  strcpy(este->error_msg_giro, "no error");

  // fin inicialización de propiedades específicas giroscopo
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_giroscopo(void *instancia) {
  struct giroscopo *este = (struct giroscopo *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int i;

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_giroscopo),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
```

```
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->n utilizadas en actualizar_matriz.combre, &este-
>t_ciclo_noRT_min, ID_VAR_INT, PUBLICO | MODIFICABLE,
"t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas giroscopo
  insertarPropiedad3(este->nombre, &este->bus_ok, ID_ENT_INT, PUBLICO
|MODIFICABLE, "bus_ok");
  insertarPropiedad3(este->nombre, &este->fd, ID_ENT_INT, PUBLICO
|MODIFICABLE, "fd");

  for(i = 0; i < MAX_EJES; i++)
    insertarPropiedad3(este->nombre, &este->Wgyr[i], ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "Wgyr[%d]", i);

  insertarPropiedad3(este->nombre, &este->flagg, ID_VAR_INT, PUBLICO |
MODIFICABLE, "flagg");

  insertarPropiedad3(este->nombre, &este->offsetGyro_x, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Cte_calib_giro_X");
  insertarPropiedad3(este->nombre, &este->offsetGyro_y, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Cte_calib_giro_Y");
  insertarPropiedad3(este->nombre, &este->offsetGyro_z, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Cte_calib_giro_Z");

  insertarPropiedad3(este->nombre, &este->error_msg_giro, ID_VAR_TEXT,
PUBLICO | MODIFICABLE, "Msg_error_giro");

  insertarPropiedad3(este->nombre, &este->x, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "x");
  insertarPropiedad3(este->nombre, &este->y, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "y");
  insertarPropiedad3(este->nombre, &este->z, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "z");

  insertarPropiedad3(este->nombre, &este->cont, ID_VAR_INT, PUBLICO |
MODIFICABLE, "Contador");
  insertarPropiedad3(este->nombre, &este->calibracion_giro, ID_SAL_INT, PUBLICO
| MODIFICABLE, "calibracion_giro");

  // fin registro de propiedades específicas giroscopo
}

/*******************************************************
*  void funcionCrea()
```

```
**********************************************************/
int funcion_crea_giroscopo(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct giroscopo *este = (struct giroscopo *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_giroscopo(instancia, nombre, orden, habilitado);

  if(registra_propiedades_giroscopo(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_giroscopo, orden,
                    &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}


 void writeto_giro (int fd, int reg, int val){
     char buf[2];
     buf[0] = reg;
     buf[1] = val;
     if (write (fd, buf, 2) != 2){
         fprintf(stderr, "No se puede escribir en el giroscopo\n");
     }
 }

/**********************************************************
*  void funcionInicializa()
**********************************************************/
void funcion_inicializa_giroscopo(void *instancia) {
  struct giroscopo *este = (struct giroscopo *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;
```

```
  este->CalibGyroX = 0;
  este->CalibGyroY = 0;
  este->CalibGyroZ = 0;

  este->x = 0;
  este->y = 0;
  este->z = 0;

  este->buf[0] = 0;
  este->buf[1] = 0;
  este->buf[2] = 0;
  este->buf[3] = 0;
  este->buf[4] = 0;
  este->buf[5] = 0;

  este->calibracion_giro = 0;
  este->cont = 0;

 // principio código específico funcion inicializa giroscopo

   if (ioctl (*este->fd, I2C_SLAVE, ITG3200_I2C_ADDR) < 0){
      strcpy(este->error_msg_giro,"Giroscopo is not present");
   }
   writeto_giro(*este->fd, ITG3200_PWR_MGM, 0x00);     //No reset.
   writeto_giro(*este->fd, ITG3200_SMPLRT_DIV, 0x00);  //1ms por muestra, de aqui
que T=1ms
   writeto_giro(*este->fd, ITG3200_DLPF_FS, 0x1e);     //f.interna = 8KHz, f.muestreo
= 1000Hz
   writeto_giro(*este->fd, ITG3200_INT_CFG, 0x01);

  // fin código específico funcion inicializa giroscopo
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_giroscopo(void *instancia, int t_ciclo) {
  struct giroscopo *este = (struct giroscopo *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal giroscopo

  if (ioctl (*este->fd, I2C_SLAVE, ITG3200_I2C_ADDR) < 0){
       strcpy(este->error_msg_giro, "Acelerometer is not present");
```

```
 }

 if(este->calibracion_giro == 1){
   if(read(*este->fd, este->flag, 1) != 0){
      if (read(*este->fd, este->buf, 6) != 6){  //La funcion read intenta leer 6$
         strcpy(este->error_msg_giro, "Unable to read from ITG3200");
      }else{
        este->flagg = i2c_smbus_read_byte_data(*este->fd,0x1a);

        if(este->flagg == 1){

           este->x = i2c_smbus_read_word_data(*este-
>fd,ITG3200_GYRO_XOUT_H);
           este->y = i2c_smbus_read_word_data(*este-
>fd,ITG3200_GYRO_YOUT_H);
           este->z = i2c_smbus_read_word_data(*este-
>fd,ITG3200_GYRO_ZOUT_H);

           /*Radianes / segundo*/
                              este->Wgyr[0] = (((float) este->x - offsetGyro_x) / (LSB *
17.8)) * GRADOS_RADIANES;
                              este->Wgyr[1] = (((float) este->y - offsetGyro_y) / (LSB *
17.8)) * GRADOS_RADIANES;
                              este->Wgyr[2] = (((float) este->z - offsetGyro_z) / (LSB *
17.8)) * GRADOS_RADIANES;
         }
         else if(este->flagg == 0){
           strcpy(este->error_msg_giro,"El flagg_fn vale 0");
         }
      }
   }
 }

 if(este->calibracion_giro == 0){
   if(read(*este->fd, este->flag, 1) != 0){
      if (read(*este->fd, este->buf, 6) != 6){  //La funcion read intenta leer 6$
         strcpy(este->error_msg_giro, "Unable to read from ITG3200");
      }else{
        este->flagg = i2c_smbus_read_byte_data(*este->fd,0x1a);
        if(este->flagg == 1){
          //    strcpy(este->error_msg_giro,"El flagg_fn vale 1");
           este->x = i2c_smbus_read_word_data(*este->fd,0x1d);
           este->y = i2c_smbus_read_word_data(*este->fd,0x1f);
           este->z = i2c_smbus_read_word_data(*este->fd,0x21);

             este->Wgyr[0] = (((float) este->x - 3845) / (LSB * 17.8)) *
GRADOS_RADIANES;
                este->Wgyr[1] = (((float) este->y - 784) / (LSB * 17.8)) *
GRADOS_RADIANES;
```

```
                este->Wgyr[2] = (((float) este->z - 281) / (LSB * 17.8)) *
GRADOS_RADIANES;
          }
          if(este->flagg == 0){
            strcpy(este->error_msg_giro,"El flagg_fn vale 0");
          }
         }
        }
        este->cont = este->cont++;

        este->CalibGyroX = este->Wgyr[0];
        este->CalibGyroY = este->Wgyr[1];
        este->CalibGyroZ = este->Wgyr[2];

        este->varX = este->CalibGyroX + este->Calib_antX;
        este->varY = este->CalibGyroY + este->Calib_antY;
        este->varZ = este->CalibGyroZ + este->Calib_antZ;

        este->Calib_antX = este->varX;
        este->Calib_antY = este->varY;
        este->Calib_antZ = este->varZ;

        if(este->cont > 50){
          este->offsetGyro_x = este->Calib_antX / 50.0;
          este->offsetGyro_y = este->Calib_antY / 50.0;
          este->offsetGyro_z = este->Calib_antZ / 50.0;
          este->calibracion_giro = 1;
        }
      }


  // fin código específico funcion normal giroscopo
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*********************************************************
*  void funcionNormalNoRT()
*********************************************************/
void funcion_normal_noRT_giroscopo(void *instancia, int t_ciclo) {
  struct giroscopo *este = (struct giroscopo *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
```

```
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT giroscopo

  // fin código específico funcion normal noRT giroscopo
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_giroscopo(void *instancia) {
  struct giroscopo *este = (struct giroscopo *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza giroscopo

     este->Wgyr[3] = 0;
     este->offsetGyro_x=este->offsetGyro_y=este->offsetGyro_z = 0;
     este->x=este->y=este->z = 0;
     este->CalibGyroX=este->CalibGyroY=este->CalibGyroZ = 0;
     este->Calib_antX=este->Calib_antY=este->Calib_antZ = 0;
     este->varX=este->varY=este->varZ = 0;

  // fin código específico funcion finaliza giroscopo
}

/*************************************************************
*  void inicilizagiroscopo()
*************************************************************/
int inicializa_giroscopo() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_giroscopo);
  logPrint(msgLog, 3);
#endif

  clase_giroscopo.funcionCrea = funcion_crea_giroscopo;
  clase_giroscopo.funcionInicializa = funcion_inicializa_giroscopo;
  clase_giroscopo.funcionNormal = funcion_normal_giroscopo;
  clase_giroscopo.funcionNormalNoRT = funcion_normal_noRT_giroscopo;
```

```
  clase_giroscopo.funcionFinaliza = funcion_finaliza_giroscopo;

#ifdef ID_LISTAS
  iniciarLista(&instancias_giroscopo);
#else
  clase_giroscopo.n = 0;
  clase_giroscopo.maxNumComp = MAX_giroscopo;
#endif

 clase_giroscopo.instancias = &instancias_giroscopo;
 clase_giroscopo.longComponente = sizeof(struct giroscopo);

 return insertarPropiedad2(ID_COMPONENTE, ID_giroscopo, &clase_giroscopo,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

MAGNETOMETRO

magnetometro.h

```
/*
 *  magnetometro.h
 *
 *  Variables utilizadas en magnetometro.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_magnetometro "magnetometro"
#define MAX_magnetometro 16

#define  HMC5883L_I2C_ADDR  0x1e
#define MAX_ERR_MSG  100
#define MAX_EJES  3

#define HMC5883L_READ_ADDR       0x3D
#define HMC5883L_WRITE_ADDR      0x3C
#define Config_Reg_A             0x00
#define Config_Reg_B             0x01
#define Mode_Reg                 0x02
#define X_MSB_Reg                0x03
#define X_LSB_Reg                0x04
#define Z_MSB_Reg                0x05
#define Z_LSB_Reg                0x06
#define Y_MSB_Reg                0x07
#define Y_LSB_Reg                0x08
#define Status_Reg               0x09
#define ID_Reg_A                 0x0A
#define ID_Reg_B                 0x0B
#define ID_Reg_C                 0x0C

#define  RADIANES_GRADOS  57.29577951
#define  GRADOS_RADIANES  0.017453293
#define  pi  3.141592654

#define hmc5883_calib_x 91
#define hmc5883_calib_y 128
#define hmc5883_calib_z -163


struct magnetometro {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
```

```
    byte habilitado;
    byte finalizado;
    int t_ciclo_RT;
    int t_ciclo_RT_min;
    int t_ciclo_RT_max;
    int t_ciclo_noRT;
    int t_ciclo_noRT_min;
    int t_ciclo_noRT_max;

    //variables de Calibracion
    char error_msg_mag[MAX_ERR_MSG];
    short x,y,z;
    unsigned char buf[8];

    float minx, miny, minz, maxx, maxy, maxz;
    float xScale, yScale, zScale;
    float compassXoffset, compassYoffset, compassZoffset;
    float offsetRoll, offsetPitch, offsetYaw;

    // entradas
    int *bus_ok, *fd;

    // salidas
    float Vmag[3];          //Vector datos magnetometro
};
```

magnetometro.c

```
/*
 *  magnetometro.c
 *
 *  Obtencion de medidas del sensor HMC5883L
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <linux/i2c-dev.h>


#include "runtime.h"
#include "magnetometro.h"
```

```
#ifdef ID_LISTAS
struct lista instancias_magnetometro;
#else
#define MAX_magnetometro 16
struct magnetometro instancias_magnetometro[MAX_magnetometro];
#endif

struct componente clase_magnetometro;

extern char msgLog[];

extern int dummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_magnetometro(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {

  struct magnetometro *este = (struct magnetometro *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas magnetometro

  este->fd = &dummy;
  este->bus_ok = &dummy;
  strcpy(este->error_msg_mag, "no error");

  // fin inicialización de propiedades específicas magnetometro
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_magnetometro(void *instancia) {
  struct magnetometro *este = (struct magnetometro *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int z;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_magnetometro), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
```

```
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas magnetometro
  insertarPropiedad3(este->nombre, &este->bus_ok, ID_ENT_INT, PUBLICO
|MODIFICABLE, "bus_ok");
  insertarPropiedad3(este->nombre, &este->fd, ID_ENT_INT, PUBLICO
|MODIFICABLE, "fd");

 for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Vmag[z], ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "Vmag[%d]",z);

  insertarPropiedad3(este->nombre, &este->error_msg_mag, ID_VAR_TEXT,
PUBLICO | MODIFICABLE, "Msg_error_mag");

  insertarPropiedad3(este->nombre, &este->x, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "x");
  insertarPropiedad3(este->nombre, &este->y, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "y");
  insertarPropiedad3(este->nombre, &este->z, ID_VAR_SHORT, PUBLICO |
MODIFICABLE, "z");

  // fin registro de propiedades específicas magnetometro
}

/*********************************************************
*  void funcionCrea()
*********************************************************/
int funcion_crea_magnetometro(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct magnetometro *este = (struct magnetometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
```

```
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_magnetometro(instancia, nombre, orden, habilitado);

  if(registra_propiedades_magnetometro(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_magnetometro, orden,
                    &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}


void writeto_mag (int fd, int reg, int val, void *instancia){  //Funcion creada para escr
 struct magnetometro *este = (struct magnetometro *)instancia;
       char buf[2];     //De -128 a 127
       buf[0] = reg;    //Asocio la primera posicion del vector creado
       buf[1] = val;    //Asocio la segunda posicion del vector creado
       if (write (fd, buf, 2) != 2){
          strcpy(este->error_msg_mag, "No se puede escribir en el magnetometro\n");
       }
}


/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_magnetometro(void *instancia) {
  struct magnetometro *este = (struct magnetometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  este->x = 0;
  este->y = 0;
  este->z = 0;
```

```
  // principio código específico funcion inicializa magnetometro

   if (ioctl (*este->fd, I2C_SLAVE, HMC5883L_I2C_ADDR) < 0){
      strcpy(este->error_msg_mag, "Magnetometer is not present\n");
   }
   //Promedio de 8 muestras cada 13.3 ms
   writeto_mag(*este->fd, Config_Reg_A, 0x78, instancia);
   //Rango campo magnetico +-1.3 Ga
   writeto_mag(*este->fd, Config_Reg_B, 0x20, instancia);
   // Modo de medicion continua
   writeto_mag(*este->fd, Mode_Reg, 0x00, instancia);

  // fin código específico funcion inicializa magnetometro
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_magnetometro(void *instancia, int t_ciclo) {
  struct magnetometro *este = (struct magnetometro *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal magnetometro

     if (ioctl (*este->fd, I2C_SLAVE, HMC5883L_I2C_ADDR) ){
        strcpy(este->error_msg_mag, "Magnetometro NO presente\n");
     }
     este->buf[0] = 0x03;      //Posicion del primer registro de datos

     if ((write(*este->fd, este->buf, 1)) != 1){
        strcpy(este->error_msg_mag, "Error escribiendo en el magnetometro\n");
     }

     if (read(*este->fd, este->buf, 6) != 6){

        strcpy(este->error_msg_mag, "No disponible para leer\n");

     }else{
       este->x = (este->buf[0] << 8) | este->buf[1];
       este->y = (este->buf[4] << 8) | este->buf[5];
       este->z = (este->buf[2] << 8) | este->buf[3];

       este->Vmag[0] = (float) este->x * 0.9229;
       este->Vmag[1] = (float) este->y * 0.9229;
```

```
        este->Vmag[2] = (float) este->z * 0.8593;

        este->Vmag[0] = (1.0 * este->Vmag[0]) + hmc5883_calib_x;
        este->Vmag[1] = (1.07 * este->Vmag[1]) + hmc5883_calib_y;
        este->Vmag[2] = (1.26 * este->Vmag[2]) - hmc5883_calib_z;
      }

  // fin código específico funcion normal magnetometro
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*********************************************************
*  void funcionNormalNoRT()
*********************************************************/
void funcion_normal_noRT_magnetometro(void *instancia, int t_ciclo) {
  struct magnetometro *este = (struct magnetometro *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT magnetometro
  // ...

  // fin código específico funcion normal noRT magnetometro
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*********************************************************
*  void funcionFinaliza()
*********************************************************/
void funcion_finaliza_magnetometro(void *instancia) {
  struct magnetometro *este = (struct magnetometro *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
```

```
#endif

  // principio código específico funcion finaliza magnetometro

   este->offsetRoll = este->offsetPitch = este->offsetYaw = 0.0;
   este->x = este->y = este->z = 0;

  // fin código específico funcion finaliza magnetometro
}

/*********************************************************
*  void inicilizamagnetometro()
*********************************************************/
int inicializa_magnetometro() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_magnetometro);
  logPrint(msgLog, 3);
#endif

  clase_magnetometro.funcionCrea = funcion_crea_magnetometro;
  clase_magnetometro.funcionInicializa = funcion_inicializa_magnetometro;
  clase_magnetometro.funcionNormal = funcion_normal_magnetometro;
  clase_magnetometro.funcionNormalNoRT = funcion_normal_noRT_magnetometro;
  clase_magnetometro.funcionFinaliza = funcion_finaliza_magnetometro;

#ifdef ID_LISTAS
  iniciarLista(&instancias_magnetometro);
#else
  clase_magnetometro.n = 0;
  clase_magnetometro.maxNumComp = MAX_magnetometro;
#endif

 clase_magnetometro.instancias = &instancias_magnetometro;
 clase_magnetometro.longComponente = sizeof(struct magnetometro);

 return insertarPropiedad2(ID_COMPONENTE, ID_magnetometro,
&clase_magnetometro, ID_COMPONENTE, NO_MODIFICABLE);
}
```

CORRECCION DERIVA CON ACELEROMETRO

correcionPI_Acc.h

```
/*
 *  correcionPI_Acc.h
 *
 *  Variables utilizadas en correcionPI_Acc.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#include "giroscopo.h"
#include "ficheroprueba.h"

#define ID_correcionPI_Acc "correcionPI_Acc"
#define MAX_correcionPI_Acc 16


struct correcionPI_Acc {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   float  Kp_ROLLPITCH;   //En adelante estas dos constantes meterlas en variables
para asi poder modificarlas en tiempo real.
   float  Ki_ROLLPITCH;
   float  Kd_ROLLPITCH;

   float Omega_P[MAX_EJES];
   float Omega_I[MAX_EJES];
   float derivative[MAX_EJES];
   float errorRollPitch[MAX_EJES];
   float errorRollPitch_old[MAX_EJES];
   float Scaled_Omega_I[MAX_EJES];

   // entradas
   float *Facc[MAX_EJES];
   float *Wgyr[MAX_EJES];
```

```
    float *dcm_matriz_renorm_1[MAX_EJES][MAX_EJES];
//    int *calibracion_giro;
    // salidas
    float Wgyr_modif[MAX_EJES];
};
```

correcionPI_Acc.c

```
/*
 *  correcionPI_Acc.c
 *
 *  Calcula el error entre el vector de aceleraciones y el vector de los
 *  datos corregidos en el ciclo anterior, este error se utiliza en un
 *  PID.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "correcionPI_Acc.h"

#ifdef ID_LISTAS
struct lista instancias_correcionPI_Acc;
#else
#define MAX_correcionPI_Acc 16
struct correcionPI_Acc instancias_correcionPI_Acc[MAX_correcionPI_Acc];
#endif

struct componente clase_correcionPI_Acc;

extern char msgLog[];

float fdummy = 0;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_correcionPI_Acc(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {

  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;

  strcpy(este->nombre, nombre);
```

```
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas correcionPI_Acc

  este->dcm_matriz_renorm_1[0][0] = &fdummy;
  este->dcm_matriz_renorm_1[0][1] = &fdummy;
  este->dcm_matriz_renorm_1[0][2] = &fdummy;
  este->dcm_matriz_renorm_1[1][0] = &fdummy;
  este->dcm_matriz_renorm_1[1][1] = &fdummy;
  este->dcm_matriz_renorm_1[1][2] = &fdummy;
  este->dcm_matriz_renorm_1[2][0] = &fdummy;
  este->dcm_matriz_renorm_1[2][1] = &fdummy;
  este->dcm_matriz_renorm_1[2][2] = &fdummy;

  este->Facc[0] = &fdummy;
  este->Facc[1] = &fdummy;
  este->Facc[2] = &fdummy;

  este->Wgyr[0] = &fdummy;
  este->Wgyr[1] = &fdummy;
  este->Wgyr[2] = &fdummy;

  // fin inicialización de propiedades específicas correcionPI_Acc
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_correcionPI_Acc(void *instancia) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int z,i,j;

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_correcionPI_Acc), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
```

```
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas correcionPI_Acc

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Facc[z], ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "Facc[%d]", z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr[z], ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "Wgyr[%d]", z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr_modif[z], ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_modif[%d]", z);

  for (i = 0; i < MAX_EJES; i++){
     for (j = 0; j < MAX_EJES; j++){
        insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm_1[i][j],
ID_ENT_FLOAT, PUBLICO | MODIFICABLE,
"dcm_matriz_renorm_1[%d][%d]",i,j);
     }
  }
  insertarPropiedad3(este->nombre, &este->Omega_P[2], ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "Omega_P[2]");
  insertarPropiedad3(este->nombre, &este->Omega_I[2], ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "Omega_I[2]");

  insertarPropiedad3(este->nombre, &este->Kp_ROLLPITCH, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Kp_ROLLPITCH");
  insertarPropiedad3(este->nombre, &este->Ki_ROLLPITCH, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Ki_ROLLPITCH");
  insertarPropiedad3(este->nombre, &este->Kd_ROLLPITCH, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Kd_ROLLPITCH");

  insertarPropiedad3(este->nombre, &este->errorRollPitch[0], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "errorRollPitch[0]");
  insertarPropiedad3(este->nombre, &este->errorRollPitch[1], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "errorRollPitch[1]");
  insertarPropiedad3(este->nombre, &este->errorRollPitch[2], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "errorRollPitch[2]");

  // fin registro de propiedades específicas correcionPI_Acc
```

```
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_correcionPI_Acc(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_correcionPI_Acc(instancia, nombre, orden, habilitado);

  if(registra_propiedades_correcionPI_Acc(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_correcionPI_Acc, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_correcionPI_Acc(void *instancia) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa correcionPI_Acc

  este->Kp_ROLLPITCH = 10;
  este->Ki_ROLLPITCH = 0.4;
```

```
  este->Kd_ROLLPITCH = 0.4;

  este->Omega_P[3] = 0;
  este->Omega_I[3] = 0;

  este->errorRollPitch[0] = 0;
  este->errorRollPitch[1] = 0;
  este->errorRollPitch[2] = 0;

  este->errorRollPitch_old[0] = 0;
  este->errorRollPitch_old[1] = 0;
  este->errorRollPitch_old[2] = 0;

  // fin código específico funcion inicializa correcionPI_Acc
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_correcionPI_Acc(void *instancia, int t_ciclo) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;
  float t_ciclo_preciso_ns = 0;
  int *t_ciclo_rt = (int *)devolverElemento("SISTEMA.tiempo_ciclo.rt");

#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);
  int i;

  // principio código específico funcion normal correcionPI_Acc

  if((*t_ciclo_rt > t_ciclo * 1e6 * 0.95) && (*t_ciclo_rt < t_ciclo * 1e6 * 1.05)){ //Se
filtran los ciclos que no interesan.

    //Calculo del error -- t_ciclo en nanosegundos --
    este->errorRollPitch[0] = *este->Facc[1] * *este->dcm_matriz_renorm_1[2][2] -
*este->Facc[2] * *este->dcm_matriz_renorm_1[2][1];
    este->errorRollPitch[1] = *este->Facc[2] * *este->dcm_matriz_renorm_1[2][0] -
*este->Facc[0] * *este->dcm_matriz_renorm_1[2][2];
    este->errorRollPitch[2] = *este->Facc[0] * *este->dcm_matriz_renorm_1[2][1] -
*este->Facc[1] * *este->dcm_matriz_renorm_1[2][0];

    for( i=0; i<3; i++){
      este->Omega_P[i] = este->errorRollPitch[i] * este->Kp_ROLLPITCH;
    }

    for( i=0; i<3; i++){
```

```
      este->Omega_I[i] = este->Omega_I[i] + este->errorRollPitch[i] *
((float)*t_ciclo_rt/1e9);
    }

    for( i=0; i<3; i++){
      este->derivative[i] = (este->errorRollPitch[i] - este->errorRollPitch_old[i]) /
((float)*t_ciclo_rt/1e9);
    }

    for (i=0 ; i<3 ; i++){
      este->Wgyr_modif[i] = *este->Wgyr[i] +  este->Omega_P[i] + este-
>Ki_ROLLPITCH*este->Omega_I[i] + este->Kd_ROLLPITCH*este->derivative[i];
    }

    for (i=0 ; i<3 ; i++){
     este->errorRollPitch_old[i] = este->errorRollPitch[i];
    }
  }

  // fin código específico funcion normal correcionPI_Acc
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_correcionPI_Acc(void *instancia, int t_ciclo) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT correcionPI_Acc

  // fin código específico funcion normal noRT correcionPI_Acc
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
```

```
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_correcionPI_Acc(void *instancia) {
  struct correcionPI_Acc *este = (struct correcionPI_Acc *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza correcionPI_Acc

  este->Omega_P[3] = 0;
  este->Omega_I[3] = 0;
  este->errorRollPitch[3] = 0;

  // fin código específico funcion finaliza correcionPI_Acc
}
/*************************************************************
*  void inicilizacorrecionPI_Acc()
*************************************************************/
int inicializa_correcionPI_Acc() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_correcionPI_Acc);
  logPrint(msgLog, 3);
#endif

  clase_correcionPI_Acc.funcionCrea = funcion_crea_correcionPI_Acc;
  clase_correcionPI_Acc.funcionInicializa = funcion_inicializa_correcionPI_Acc;
  clase_correcionPI_Acc.funcionNormal = funcion_normal_correcionPI_Acc;
  clase_correcionPI_Acc.funcionNormalNoRT =
funcion_normal_noRT_correcionPI_Acc;
  clase_correcionPI_Acc.funcionFinaliza = funcion_finaliza_correcionPI_Acc;

#ifdef ID_LISTAS
  iniciarLista(&instancias_correcionPI_Acc);
#else
  clase_correcionPI_Acc.n = 0;
  clase_correcionPI_Acc.maxNumComp = MAX_correcionPI_Acc;
#endif

 clase_correcionPI_Acc.instancias = &instancias_correcionPI_Acc;
 clase_correcionPI_Acc.longComponente = sizeof(struct correcionPI_Acc);

 return insertarPropiedad2(ID_COMPONENTE, ID_correcionPI_Acc,
&clase_correcionPI_Acc, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA PROMEDIADORA

promediador.h

```
/*
 *  promediador.h
 *
 *  Variables utilizadas en promediador.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_promediador "promediador"
#define MAX_EJES 3
#define MAX_PROMEDIO 8

struct promediador {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   int primer_dato_pitch;
   float vector_filtro_pitch[MAX_PROMEDIO];
   float vector_filtro_roll[MAX_PROMEDIO];
   float acumula_filtrado_pitch;
   float acumula_filtrado_roll;

   // entradas
   float *Wgyr_modif[MAX_EJES];
   float *pitch;

   // salidas
   float Wgyr_modif_prom[MAX_EJES];
};
```

promediador.c

```
/*
 *  promediador.c
 *
 *  Se promedian los datos corregidos anteriormente.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "promediador.h"

#ifdef ID_LISTAS
struct lista instancias_promediador;
#else
#define MAX_promediador 16
struct promediador instancias_promediador[MAX_promediador];
#endif

struct componente clase_promediador;

extern char msgLog[];

extern float fdummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_promediador(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct promediador *este = (struct promediador *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas promediador

  este->Wgyr_modif[0] = &fdummy;
  este->Wgyr_modif[1] = &fdummy;
```

```
  este->Wgyr_modif[2] = &fdummy;
  este->pitch = &fdummy;

  // fin inicialización de propiedades específicas promediador
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_promediador(void *instancia) {
  struct promediador *este = (struct promediador *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int z;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_promediador), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas promediador

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr_modif_prom[z], ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_modif_prom[%d]",z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr_modif[z], ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_modif[%d]",z);

    insertarPropiedad3(este->nombre, &este->acumula_filtrado_pitch,
ID_VAR_FLOAT, PUBLICO | MODIFICABLE, "acumula_filtrado_pitch");
```

```
    insertarPropiedad3(este->nombre, &este->acumula_filtrado_roll, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "acumula_filtrado_roll");

  // fin registro de propiedades específicas promediador
}

/**********************************************************
*  void funcionCrea()
**********************************************************/
int funcion_crea_promediador(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct promediador *este = (struct promediador *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_promediador(instancia, nombre, orden, habilitado);

  if(registra_propiedades_promediador(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_promediador, orden,
                    &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/**********************************************************
*  void funcionInicializa()
**********************************************************/
void funcion_inicializa_promediador(void *instancia) {
  struct promediador *este = (struct promediador *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa promediador
```

```
  este->primer_dato_pitch = 1;
  este->acumula_filtrado_pitch = 0;
  este->acumula_filtrado_roll = 0;
  este->vector_filtro_pitch[MAX_PROMEDIO] = 0;
  este->vector_filtro_roll[MAX_PROMEDIO] = 0;

  // fin código específico funcion inicializa promediador
}

/*********************************************************
*  void funcionNormal()
*********************************************************/
void funcion_normal_promediador(void *instancia, int t_ciclo) {
  struct promediador *este = (struct promediador *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal promediador

    int n,i;
    if (este->primer_dato_pitch == 1) {
      for (n = 0; n < MAX_PROMEDIO ; n++){    // Llenamos la matriz con el primer
dato recibido
        este->vector_filtro_roll[n] = *este->Wgyr_modif[0];
        este->vector_filtro_pitch[n] = *este->Wgyr_modif[1];
      }
      este->primer_dato_pitch = 0;          //Para que no vuelva a entrar en el bucle anterior
    }

//////////////////// -Promediado Pitch- ////////////////////////////////////
    este->acumula_filtrado_pitch = 0;

      for (i=0; i < (MAX_PROMEDIO - 1); i++){    // Movemos solo los 7 primeros
        este->vector_filtro_pitch[i] = este->vector_filtro_pitch[i+1];
        este->acumula_filtrado_pitch = este->acumula_filtrado_pitch + este-
>vector_filtro_pitch[i];  //Suma los 8 valores que hay en el filtro
      }

      este->vector_filtro_pitch[MAX_PROMEDIO-1] = *este->Wgyr_modif[1];       //
Aniadimos el ULTIMO
      este->acumula_filtrado_pitch =  este->acumula_filtrado_pitch +  este-
>vector_filtro_pitch[MAX_PROMEDIO-1];
      este->acumula_filtrado_pitch =  este->acumula_filtrado_pitch /
MAX_PROMEDIO;
```

```
      este->Wgyr_modif_prom[1] = este->acumula_filtrado_pitch;


//////////////////// -Promediado Roll- ////////////////////////////////////
    este->acumula_filtrado_roll = 0;

    for (i=0; i < (MAX_PROMEDIO - 1); i++){     // Movemos solo los 7 primeros
      este->vector_filtro_roll[i] = este->vector_filtro_roll[i+1];
      este->acumula_filtrado_roll = este->acumula_filtrado_roll + este-
>vector_filtro_roll[i];  //Suma los 8 valores que hay en el filtro
    }

    este->vector_filtro_roll[MAX_PROMEDIO-1] = *este->Wgyr_modif[0];        //
Aniadimos el ULTIMO
    este->acumula_filtrado_roll =  este->acumula_filtrado_roll +  este-
>vector_filtro_roll[MAX_PROMEDIO-1];
    este->acumula_filtrado_roll =  este->acumula_filtrado_roll / MAX_PROMEDIO;

    este->Wgyr_modif_prom[0] = este->acumula_filtrado_roll;


    este->Wgyr_modif_prom[2] = *este->Wgyr_modif[2];  //No se promedia el angulo
yaw

  // fin codigo especifico funcion normal promediador

  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_promediador(void *instancia, int t_ciclo) {
  struct promediador *este = (struct promediador *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT promediador
  // ...
```

```
  // fin código específico funcion normal noRT promediador
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_promediador(void *instancia) {
  struct promediador *este = (struct promediador *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza promediador


  // fin código específico funcion finaliza promediador
}

/*************************************************************
*  void inicilizapromediador()
*************************************************************/
int inicializa_promediador() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_promediador);
  logPrint(msgLog, 3);
#endif

  clase_promediador.funcionCrea = funcion_crea_promediador;
  clase_promediador.funcionInicializa = funcion_inicializa_promediador;
  clase_promediador.funcionNormal = funcion_normal_promediador;
  clase_promediador.funcionNormalNoRT = funcion_normal_noRT_promediador;
  clase_promediador.funcionFinaliza = funcion_finaliza_promediador;

#ifdef ID_LISTAS
  iniciarLista(&instancias_promediador);
#else
  clase_promediador.n = 0;
  clase_promediador.maxNumComp = MAX_promediador;
#endif

 clase_promediador.instancias = &instancias_promediador;
 clase_promediador.longComponente = sizeof(struct promediador);
```

```
 return insertarPropiedad2(ID_COMPONENTE, ID_promediador,
&clase_promediador, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA CORRECTORA CON DATOS DEL MAGNETOMETRO

correccionPI_Mag.h

```
/*
 *  correccionPI_Mag.h
 *
 *  Variables utilizadas en correccionPI_Mag.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_correccionPI_Mag "correccionPI_Mag"
#define MAX_correccionPI_Mag 16
#define MAX_EJES 3
struct correccionPI_Mag {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   float Kp_Yaw;   //En adelante estas dos constantes meterlas en varia$
   float Ki_Yaw;

   float Omega_P_Compass[MAX_EJES];
   float Omega_I_Compass[MAX_EJES];
   float errorYaw[MAX_EJES];
   float Scaled_Omega_I_Compass[MAX_EJES];

   // entradas
   float *Vmag[MAX_EJES];
   float *Wgyr_modif_prom[MAX_EJES];
   float *dcm_matriz_renorm_1[MAX_EJES][MAX_EJES];

   // salidas
   float Wgyr_fin[MAX_EJES];
};
```

correccionPI_Mag.c

```
/*
 *  correccionPI_Mag.c
 *
 *  Calcula el error entre el vector del magnetometro y el vector de los
 *  datos corregidos en el ciclo anterior, este error se utiliza en un
 *  PID para corregir los datos de l giróscopo.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "correccionPI_Mag.h"

#ifdef ID_LISTAS
struct lista instancias_correccionPI_Mag;
#else
#define MAX_correccionPI_Mag 16
struct correccionPI_Mag instancias_correccionPI_Mag[MAX_correccionPI_Mag];
#endif

struct componente clase_correccionPI_Mag;

extern char msgLog[];

extern float fdummy;

/*********************************************************
*  void inicializa_propiedades()
*********************************************************/
void inicializa_propiedades_correccionPI_Mag(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {

  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas correccionPI_Mag

  este->dcm_matriz_renorm_1[0][0] = &fdummy;
  este->dcm_matriz_renorm_1[0][1] = &fdummy;
```

```
  este->dcm_matriz_renorm_1[0][2] = &fdummy;
  este->dcm_matriz_renorm_1[1][0] = &fdummy;
  este->dcm_matriz_renorm_1[1][1] = &fdummy;
  este->dcm_matriz_renorm_1[1][2] = &fdummy;
  este->dcm_matriz_renorm_1[2][0] = &fdummy;
  este->dcm_matriz_renorm_1[2][1] = &fdummy;
  este->dcm_matriz_renorm_1[2][2] = &fdummy;

  este->Vmag[0] = &fdummy;
  este->Vmag[1] = &fdummy;
  este->Vmag[2] = &fdummy;

  este->Wgyr_modif_prom[0] = &fdummy;
  este->Wgyr_modif_prom[1] = &fdummy;
  este->Wgyr_modif_prom[2] = &fdummy;

  // fin inicialización de propiedades específicas correccionPI_Mag
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_correccionPI_Mag(void *instancia) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int z,i,j;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_correccionPI_Mag), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");
```

```
  // principio registro de propiedades específicas correccionPI_Mag

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr_fin[z], ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_fin[%d]",z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Vmag[z], ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "Vmag[%d]",z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->Wgyr_modif_prom[z], ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_modif_prom[%d]",z);

  for (z = 0; z < MAX_EJES; z++)
    insertarPropiedad3(este->nombre, &este->errorYaw[z], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "errorYaw[%d]",z);

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm_1[i][j],
ID_ENT_FLOAT, PUBLICO | MODIFICABLE,
"dcm_matriz_renorm_1[%d][%d]",i,j);
    }
  }

  insertarPropiedad3(este->nombre, &este->Omega_P_Compass[2], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Omega_P_Compass");
  insertarPropiedad3(este->nombre, &este->Omega_I_Compass[2], ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "Omega_I_Compass");

  insertarPropiedad3(este->nombre, &este->Kp_Yaw, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "Kp_Yaw");
  insertarPropiedad3(este->nombre, &este->Ki_Yaw, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "Ki_Yaw");

  // fin registro de propiedades específicas correccionPI_Mag
}

/**********************************************************
*  void funcionCrea()
**********************************************************/
int funcion_crea_correccionPI_Mag(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif
```

```
  inicializa_propiedades_correccionPI_Mag(instancia, nombre, orden, habilitado);

  if(registra_propiedades_correccionPI_Mag(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_correccionPI_Mag, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*****************************************************************
*  void funcionInicializa()
*****************************************************************/
void funcion_inicializa_correccionPI_Mag(void *instancia) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa correccionPI_Mag

  este->Kp_Yaw = 4;
  este->Ki_Yaw = 0.0002;

  este->Omega_P_Compass[3] = 0;
  este->Omega_I_Compass[3] = 0;

  este->errorYaw[3] = 0;

  // fin código específico funcion inicializa correccionPI_Mag
}

/*****************************************************************
*  void funcionNormal()
*****************************************************************/
void funcion_normal_correccionPI_Mag(void *instancia, int t_ciclo) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;
```

```
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal correccionPI_Mag

  int c,i;

  *este->Vmag[0] /= 500;
  *este->Vmag[1] /= 500;
  *este->Vmag[2] /= 500;

  este->errorYaw[0] = *este->Vmag[1] * *este->dcm_matriz_renorm_1[1][2] - *este-
>Vmag[2] * *este->dcm_matriz_renorm_1[1][1];
  este->errorYaw[1] = *este->Vmag[2] * *este->dcm_matriz_renorm_1[1][0] - *este-
>Vmag[0] * *este->dcm_matriz_renorm_1[1][2];
  este->errorYaw[2] = *este->Vmag[0] * *este->dcm_matriz_renorm_1[1][1] - *este-
>Vmag[1] * *este->dcm_matriz_renorm_1[1][0];

  for( c=0; c<3; c++){
     este->Omega_P_Compass[c] = este->errorYaw[c] * este->Kp_Yaw;
  }

  for( c=0; c<3; c++){
     este->Scaled_Omega_I_Compass[c] = este->errorYaw[c] * este->Ki_Yaw;
   }

  for( c=0; c<3; c++){
    este->Omega_I_Compass[c] = este->Omega_I_Compass[c] + este-
>Scaled_Omega_I_Compass[c];
  }

//  for (i=0 ; i<3 ; i++){
      este->Wgyr_fin[2] = *este->Wgyr_modif_prom[2] + este->Omega_I_Compass[2] +
este->Omega_P_Compass[2];
//  }
   este->Wgyr_fin[0] = *este->Wgyr_modif_prom[0];
   este->Wgyr_fin[1] = *este->Wgyr_modif_prom[1];



// fin código específico funcion normal correccionPI_Mag
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
```

```
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*****************************************************
*  void funcionNormalNoRT()
*****************************************************/
void funcion_normal_noRT_correccionPI_Mag(void *instancia, int t_ciclo) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT correccionPI_Mag


  // fin código específico funcion normal noRT correccionPI_Mag
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*****************************************************
*  void funcionFinaliza()
*****************************************************/
void funcion_finaliza_correccionPI_Mag(void *instancia) {
  struct correccionPI_Mag *este = (struct correccionPI_Mag *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza correccionPI_Mag

  este->Omega_P_Compass[3] = 0;
  este->Omega_I_Compass[3] = 0;
  este->errorYaw[3] = 0;

  // fin código específico funcion finaliza correccionPI_Mag
}

/*****************************************************
*  void inicilizacorreccionPI_Mag()
```

```
**********************************************************/
int inicializa_correccionPI_Mag() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_correccionPI_Mag);
  logPrint(msgLog, 3);
#endif

  clase_correccionPI_Mag.funcionCrea = funcion_crea_correccionPI_Mag;
  clase_correccionPI_Mag.funcionInicializa = funcion_inicializa_correccionPI_Mag;
  clase_correccionPI_Mag.funcionNormal = funcion_normal_correccionPI_Mag;
  clase_correccionPI_Mag.funcionNormalNoRT =
funcion_normal_noRT_correccionPI_Mag;
  clase_correccionPI_Mag.funcionFinaliza = funcion_finaliza_correccionPI_Mag;

#ifdef ID_LISTAS
  iniciarLista(&instancias_correccionPI_Mag);
#else
  clase_correccionPI_Mag.n = 0;
  clase_correccionPI_Mag.maxNumComp = MAX_correccionPI_Mag;
#endif

 clase_correccionPI_Mag.instancias = &instancias_correccionPI_Mag;
 clase_correccionPI_Mag.longComponente = sizeof(struct correccionPI_Mag);

 return insertarPropiedad2(ID_COMPONENTE, ID_correccionPI_Mag,
&clase_correccionPI_Mag, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA QUE ACTUALIZA LA MATRIZ DE ROTACIÓN

actualizar_matriz.h

```
/*
 *  actualizar_matriz.h
 *
 *  Variables utilizadas en actualizar_matriz.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_actualizar_matriz "actualizar_matriz"
#define MAX_actualizar_matriz 16
#define MAX_EJES 3

struct actualizar_matriz {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   float Matriz_Cambio[MAX_EJES][MAX_EJES];
   float matriz_temporal[MAX_EJES][MAX_EJES];
   float time_muestreo;
   float acumula[MAX_EJES];

   // entradas
   float *Wgyr_fin[MAX_EJES];
   int *calibracion_giro;
   float *dcm_matriz_renorm_1[MAX_EJES][MAX_EJES];


   // salidas
   float dcm_matriz_1[MAX_EJES][MAX_EJES];
   float DCM_Matriz[MAX_EJES][MAX_EJES];

};
```

actualizar_matriz.c

```
/*
 *  actualizar_matriz.c
 *
 *  A partir de las mediciones del giroscopo en º/s las convierte en º y las almacena en la matriz de rotación.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "actualizar_matriz.h"

#ifdef ID_LISTAS
struct lista instancias_actualizar_matriz;
#else
#define MAX_actualizar_matriz 16
struct actualizar_matriz instancias_actualizar_matriz[MAX_actualizar_matriz];
#endif

struct componente clase_actualizar_matriz;

extern char msgLog[];

extern float fdummy;
extern int dummy;
/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_actualizar_matriz(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {

  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas actualizar_matriz

  este->dcm_matriz_renorm_1[0][0] = &fdummy;
  este->dcm_matriz_renorm_1[0][1] = &fdummy;
  este->dcm_matriz_renorm_1[0][2] = &fdummy;
```

```
  este->dcm_matriz_renorm_1[1][0] = &fdummy;
  este->dcm_matriz_renorm_1[1][1] = &fdummy;
  este->dcm_matriz_renorm_1[1][2] = &fdummy;
  este->dcm_matriz_renorm_1[2][0] = &fdummy;
  este->dcm_matriz_renorm_1[2][1] = &fdummy;
  este->dcm_matriz_renorm_1[2][2] = &fdummy;


  este->Wgyr_fin[0] = &fdummy;
  este->Wgyr_fin[1] = &fdummy;
  este->Wgyr_fin[2] = &fdummy;

  este->calibracion_giro = &dummy;

  // fin inicialización de propiedades específicas actualizar_matriz
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_actualizar_matriz(void *instancia) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int i,j;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_actualizar_matriz), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas actualizar_matriz
```

```
  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_1[i][j], ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "dcm_matriz_1[%d][%d]",i,j);
    }
  }
  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm_1[i][j],
ID_ENT_FLOAT, PUBLICO | MODIFICABLE,
"dcm_matriz_renorm_1[%d][%d]",i,j);
    }
  }

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->DCM_Matriz[i][j], ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "DCM_Matriz[%d][%d]",i,j);
    }
  }

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->Matriz_Cambio[i][j],
ID_VAR_FLOAT, PUBLICO | MODIFICABLE, "Matriz_Cambio[%d][%d]",i,j);
    }
  }

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->matriz_temporal[i][j],
ID_VAR_FLOAT, PUBLICO | MODIFICABLE, "matriz_temporal[%d][%d]",i,j);
    }
  }

 for (i = 0; i < MAX_EJES; i++)
    insertarPropiedad3(este->nombre, &este->Wgyr_fin[i], ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "Wgyr_fin[%d]",i);

 for (i = 0; i < MAX_EJES; i++)
    insertarPropiedad3(este->nombre, &este->acumula[i], ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "acumula[%d]",i);

  insertarPropiedad3(este->nombre, &este->time_muestreo, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "time_muestreo");
  insertarPropiedad3(este->nombre, &este->calibracion_giro, ID_ENT_INT, PUBLICO
| MODIFICABLE, "calibracion_giro");


  // fin registro de propiedades específicas actualizar_matriz
```

```
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_actualizar_matriz(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_actualizar_matriz(instancia, nombre, orden, habilitado);

  if(registra_propiedades_actualizar_matriz(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_actualizar_matriz, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_actualizar_matriz(void *instancia) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa actualizar_matriz

  este->DCM_Matriz[0][0] = 1;
  este->DCM_Matriz[0][1] = 0;
  este->DCM_Matriz[0][2] = 0;
```

```
  este->DCM_Matriz[1][0] = 0;
  este->DCM_Matriz[1][1] = 1;
  este->DCM_Matriz[1][2] = 0;
  este->DCM_Matriz[2][0] = 0;
  este->DCM_Matriz[2][1] = 0;
  este->DCM_Matriz[2][2] = 1;

  este->dcm_matriz_1[0][0] = 1;
  este->dcm_matriz_1[0][1] = 0;
  este->dcm_matriz_1[0][2] = 0;
  este->dcm_matriz_1[1][0] = 0;
  este->dcm_matriz_1[1][1] = 1;
  este->dcm_matriz_1[1][2] = 0;
  este->dcm_matriz_1[2][0] = 0;
  este->dcm_matriz_1[2][1] = 0;
  este->dcm_matriz_1[2][2] = 1;

  // fin código específico funcion inicializa actualizar_matriz
}

/*********************************************************
*  void funcionNormal()
*********************************************************/
void funcion_normal_actualizar_matriz(void *instancia, int t_ciclo) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;
  int *t_ciclo_rt = (int *)devolverElemento("SISTEMA.tiempo_ciclo.rt");
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal actualizar_matriz
  int x, y, w, l, m;

  if ( (*t_ciclo_rt > t_ciclo * 1e6 * 0.95) &&  (*t_ciclo_rt < t_ciclo * 1e6 * 1.05)){

    if(*este->calibracion_giro == 1){
      este->Matriz_Cambio[0][0] = 1;
      este->Matriz_Cambio[0][1] = -((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[2];   // -z
      este->Matriz_Cambio[0][2] = ((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[1];      //y
      este->Matriz_Cambio[1][0] = ((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[2];      //z
      este->Matriz_Cambio[1][1] = 1;
      este->Matriz_Cambio[1][2] = -((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[0];
      este->Matriz_Cambio[2][0] = -((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[1];
      este->Matriz_Cambio[2][1] = ((float)*t_ciclo_rt/1e9) * *este->Wgyr_fin[0];
      este->Matriz_Cambio[2][2] = 1;
    }
```

```
   if ((*este->calibracion_giro == 1 && este->Matriz_Cambio[2][0] < 0.5) && (este-
>Matriz_Cambio[1][2] < 0.5) ){
     for ( x = 0; x < 3; x++)  {
       for ( y = 0; y < 3; y++)    {
         for ( w = 0; w < 3; w++)      {
           este->acumula[w] = *este->dcm_matriz_renorm_1[x][w] * este-
>Matriz_Cambio[w][y];
         }
        este->DCM_Matriz[x][y] = este->acumula[0] + este->acumula[1] + este-
>acumula[2];
       }
     }
   }

   if (*este->calibracion_giro == 1 ){
     for( l = 0; l < MAX_EJES; l++){
       for( m = 0; m < MAX_EJES; m++){
         este->dcm_matriz_1[l][m] = este->DCM_Matriz[l][m];
       }
     }
   }
  }

  // fin código específico funcion normal actualizar_matriz
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
   este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
   este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_actualizar_matriz(void *instancia, int t_ciclo) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT actualizar_matriz
  // fin código específico funcion normal noRT actualizar_matriz
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
```

```
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_actualizar_matriz(void *instancia) {
  struct actualizar_matriz *este = (struct actualizar_matriz *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza actualizar_matriz


  // fin código específico funcion finaliza actualizar_matriz
}
/*************************************************************
*  void inicilizaactualizar_matriz()
*************************************************************/
int inicializa_actualizar_matriz() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_actualizar_matriz);
  logPrint(msgLog, 3);
#endif

  clase_actualizar_matriz.funcionCrea = funcion_crea_actualizar_matriz;
  clase_actualizar_matriz.funcionInicializa = funcion_inicializa_actualizar_matriz;
  clase_actualizar_matriz.funcionNormal = funcion_normal_actualizar_matriz;
  clase_actualizar_matriz.funcionNormalNoRT =
funcion_normal_noRT_actualizar_matriz;
  clase_actualizar_matriz.funcionFinaliza = funcion_finaliza_actualizar_matriz;

#ifdef ID_LISTAS
  iniciarLista(&instancias_actualizar_matriz);
#else
  clase_actualizar_matriz.n = 0;
  clase_actualizar_matriz.maxNumComp = MAX_actualizar_matriz;
#endif

 clase_actualizar_matriz.instancias = &instancias_actualizar_matriz;
 clase_actualizar_matriz.longComponente = sizeof(struct actualizar_matriz);

 return insertarPropiedad2(ID_COMPONENTE, ID_actualizar_matriz,
&clase_actualizar_matriz, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA DE RENORMALIZACIÓN

renormalizar.h

```
/*
 *  renormalizar.h
 *
 *  Variables utilizadas en renormalizar.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_renormalizar "renormalizar"
#define MAX_renormalizar 16
#define MAX_EJES 3

struct renormalizar {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   float error;
   float renorm;

   // entradas
   float *dcm_matriz_1[MAX_EJES][MAX_EJES];
   float *DCM_Matriz[MAX_EJES][MAX_EJES];

   // salidas
   float dcm_matriz_renorm[MAX_EJES][MAX_EJES];
   float dcm_matriz_renorm_1[MAX_EJES][MAX_EJES];

};
```

renormalizar.c

```
/*
 *  renormalizar.c
 *
 *  Corrige la desviación producida por las aproximaciones.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "renormalizar.h"

#ifdef ID_LISTAS
struct lista instancias_renormalizar;
#else
#define MAX_renormalizar 16
struct renormalizar instancias_renormalizar[MAX_renormalizar];
#endif

struct componente clase_renormalizar;

extern char msgLog[];

extern float fdummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_renormalizar(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct renormalizar *este = (struct renormalizar *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas renormalizar

  este->DCM_Matriz[0][0] = &fdummy;
  este->DCM_Matriz[0][1] = &fdummy;
```

```
  este->DCM_Matriz[0][2] = &fdummy;
  este->DCM_Matriz[1][0] = &fdummy;
  este->DCM_Matriz[1][1] = &fdummy;
  este->DCM_Matriz[1][2] = &fdummy;
  este->DCM_Matriz[2][0] = &fdummy;
  este->DCM_Matriz[2][1] = &fdummy;
  este->DCM_Matriz[2][2] = &fdummy;


  este->dcm_matriz_1[0][0] = &fdummy;
  este->dcm_matriz_1[0][1] = &fdummy;
  este->dcm_matriz_1[0][2] = &fdummy;
  este->dcm_matriz_1[1][0] = &fdummy;
  este->dcm_matriz_1[1][1] = &fdummy;
  este->dcm_matriz_1[1][2] = &fdummy;
  este->dcm_matriz_1[2][0] = &fdummy;
  este->dcm_matriz_1[2][1] = &fdummy;
  este->dcm_matriz_1[2][2] = &fdummy;

  // fin inicialización de propiedades específicas renormalizar
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_renormalizar(void *instancia) {
  struct renormalizar *este = (struct renormalizar *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int i,j;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_renormalizar), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
```

```
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas renormalizar

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->DCM_Matriz[i][j], ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "DCM_Matriz[%d][%d]",i,j);
    }
  }
  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_1[i][j], ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "dcm_matriz_1[%d][%d]",i,j);
    }
  }
  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm[i][j],
ID_SAL_FLOAT, PUBLICO | MODIFICABLE, "dcm_matriz_renorm[%d][%d]",i,j);
    }
  }
  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm_1[i][j],
ID_SAL_FLOAT, PUBLICO | MODIFICABLE,
"dcm_matriz_renorm_1[%d][%d]",i,j);
    }
  }
  insertarPropiedad3(este->nombre, &este->renorm, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "renorm");
  insertarPropiedad3(este->nombre, &este->error, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "error");

  // fin registro de propiedades específicas renormalizar
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_renormalizar(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct renormalizar *este = (struct renormalizar *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_renormalizar(instancia, nombre, orden, habilitado);
```

```
  if(registra_propiedades_renormalizar(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_renormalizar, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_renormalizar(void *instancia) {
  struct renormalizar *este = (struct renormalizar *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa renormalizar

  este->error = 0;
  este->renorm = 0;

  // fin código específico funcion inicializa renormalizar
}


/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_renormalizar(void *instancia, int t_ciclo) {
  struct renormalizar *este = (struct renormalizar *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
```

```
 crononsec(1, &ts);

 // principio código específico funcion normal renormalizar

 int x,y,c;
 float acumul = 0;
 for( c = 0; c < MAX_EJES; c++) {
  acumul = acumul + *este->DCM_Matriz[0][c] * *este->DCM_Matriz[1][c];
 }

 este->error = -(acumul) * .5;

 for( c = 0; c < MAX_EJES; c++) {
  *este->dcm_matriz_1[0][c] = *este->DCM_Matriz[1][c] * este->error;
  *este->dcm_matriz_1[1][c] = *este->DCM_Matriz[0][c] * este->error;
 }

 for( c=0; c < MAX_EJES; c++){
  *este->dcm_matriz_1[0][c] = *este->dcm_matriz_1[0][c] + *este-
>DCM_Matriz[0][c];
  *este->dcm_matriz_1[1][c] = *este->dcm_matriz_1[1][c] + *este-
>DCM_Matriz[1][c];
 }

 *este->dcm_matriz_1[2][0] = *este->dcm_matriz_1[0][1] * *este-
>dcm_matriz_1[1][2] - *este->dcm_matriz_1[0][2] * *este->dcm_matriz_1[1][1];
 *este->dcm_matriz_1[2][1] = *este->dcm_matriz_1[0][2] * *este-
>dcm_matriz_1[1][0] - *este->dcm_matriz_1[0][0] * *este->dcm_matriz_1[1][2];
 *este->dcm_matriz_1[2][2] = *este->dcm_matriz_1[0][0] * *este-
>dcm_matriz_1[1][1] - *este->dcm_matriz_1[0][1] * *este->dcm_matriz_1[1][0];


 acumul = 0;
 for( c = 0; c < MAX_EJES; c++) {
  acumul = acumul + *este->dcm_matriz_1[0][c] * *este->dcm_matriz_1[0][c];
 }
 este->renorm = .5 * (3 - acumul);
 for( c=0; c< MAX_EJES; c++) {
  este->dcm_matriz_renorm[0][c] = *este->dcm_matriz_1[0][c] * este->renorm;
 }


 acumul = 0;
 for( c = 0; c < MAX_EJES; c++) {
  acumul = acumul + *este->dcm_matriz_1[1][c] * *este->dcm_matriz_1[1][c];
 }
 este->renorm = .5 * (3 - acumul);
 for( c=0; c< MAX_EJES; c++) {
  este->dcm_matriz_renorm[1][c] = *este->dcm_matriz_1[1][c] * este->renorm;
 }
```

```
  acumul = 0;
  for( c = 0; c < MAX_EJES; c++) {
    acumul = acumul + *este->dcm_matriz_1[2][c] * *este->dcm_matriz_1[2][c];
  }
  este->renorm = .5 * (3 - acumul);
  for( c=0; c < MAX_EJES; c++) {
    este->dcm_matriz_renorm[2][c] = *este->dcm_matriz_1[2][c] * este->renorm;
  }


  for( x = 0; x < MAX_EJES; x++){
    for( y = 0; y < MAX_EJES; y++){
      este->dcm_matriz_renorm_1[x][y] = este->dcm_matriz_renorm[x][y];
    }
  }

  // fin código específico funcion normal renormalizar
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_renormalizar(void *instancia, int t_ciclo) {
  struct renormalizar *este = (struct renormalizar *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT renormalizar

  // fin código específico funcion normal noRT renormalizar
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
```

```
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_renormalizar(void *instancia) {
  struct renormalizar *este = (struct renormalizar *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);

#endif

  // principio código específico funcion finaliza renormalizar


  // fin código específico funcion finaliza renormalizar
}

/*************************************************************
*  void inicilizarenormalizar()
*************************************************************/
int inicializa_renormalizar() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_renormalizar);
  logPrint(msgLog, 3);
#endif

  clase_renormalizar.funcionCrea = funcion_crea_renormalizar;
  clase_renormalizar.funcionInicializa = funcion_inicializa_renormalizar;
  clase_renormalizar.funcionNormal = funcion_normal_renormalizar;
  clase_renormalizar.funcionNormalNoRT = funcion_normal_noRT_renormalizar;
  clase_renormalizar.funcionFinaliza = funcion_finaliza_renormalizar;

#ifdef ID_LISTAS
  iniciarLista(&instancias_renormalizar);
#else
  clase_renormalizar.n = 0;
  clase_renormalizar.maxNumComp = MAX_renormalizar;
#endif

 clase_renormalizar.instancias = &instancias_renormalizar;
 clase_renormalizar.longComponente = sizeof(struct renormalizar);

 return insertarPropiedad2(ID_COMPONENTE, ID_renormalizar,
&clase_renormalizar, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA CONVERSORA DE ANGULOS

conversor_angulos.h

```
/*
 *  conversor_angulos.h
 *
 *  Variables utilizadas en conversor_angulos.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_conversor_angulos "conversor_angulos"
#define MAX_conversor_angulos 16
#define MAX_EJES 3
#define RADIANES_GRADOS 57.29577951

struct conversor_angulos {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   int contador;
   int primerDatoYaw;
   float primerYaw;

   // entradas
   float *dcm_matriz_renorm[MAX_EJES][MAX_EJES];

   // salidas
   float roll;
   float pitch;
   float yaw;
};
```

conversor_angulos.c

```
/*
 *  conversor_angulos.c
 *
 *  Convierte los los angulos en radianes en grados
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "conversor_angulos.h"

#ifdef ID_LISTAS
struct lista instancias_conversor_angulos;
#else
#define MAX_conversor_angulos 16
struct conversor_angulos instancias_conversor_angulos[MAX_conversor_angulos];
#endif

struct componente clase_conversor_angulos;

extern char msgLog[];

extern float fdummy;

/*********************************************************
*  void inicializa_propiedades()
*********************************************************/
void inicializa_propiedades_conversor_angulos(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas conversor_angulos

  este->dcm_matriz_renorm[0][0] = &fdummy;
  este->dcm_matriz_renorm[0][1] = &fdummy;
  este->dcm_matriz_renorm[0][2] = &fdummy;
  este->dcm_matriz_renorm[1][0] = &fdummy;
  este->dcm_matriz_renorm[1][1] = &fdummy;
```

```
  este->dcm_matriz_renorm[1][2] = &fdummy;
  este->dcm_matriz_renorm[2][0] = &fdummy;
  este->dcm_matriz_renorm[2][1] = &fdummy;
  este->dcm_matriz_renorm[2][2] = &fdummy;

  // fin inicialización de propiedades específicas conversor_angulos
}

/*************************************************************
*  void registra_propiedades()
*************************************************************/
int registra_propiedades_conversor_angulos(void *instancia) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;

  char saux[MAX_LONG_NOMBRE];
  int i, j;
  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_conversor_angulos), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas conversor_angulos

  for (i = 0; i < MAX_EJES; i++){
    for (j = 0; j < MAX_EJES; j++){
      insertarPropiedad3(este->nombre, &este->dcm_matriz_renorm[i][j],
ID_ENT_FLOAT, PUBLICO | MODIFICABLE, "dcm_matriz_renorm[%d][%d]",i,j);
    }
  }
  insertarPropiedad3(este->nombre, &este->roll, ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "roll");
```

```
  insertarPropiedad3(este->nombre, &este->pitch, ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "pitch");
  insertarPropiedad3(este->nombre, &este->yaw, ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "yaw");
  insertarPropiedad3(este->nombre, &este->primerDatoYaw, ID_VAR_INT, PUBLICO
| MODIFICABLE, "primerDatoYaw");
  insertarPropiedad3(este->nombre, &este->contador, ID_VAR_INT, PUBLICO |
MODIFICABLE, "contador");

  // fin registro de propiedades específicas conversor_angulos
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_conversor_angulos(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_conversor_angulos(instancia, nombre, orden, habilitado);

  if(registra_propiedades_conversor_angulos(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_conversor_angulos, orden,
                    &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_conversor_angulos(void *instancia) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
```

```
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa conversor_angulos

  este->primerDatoYaw = 1;
  este->contador = 0;
  este->primerYaw = 0;

  // fin código específico funcion inicializa conversor_angulos
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_conversor_angulos(void *instancia, int t_ciclo) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);


  // principio código específico funcion normal conversor_angulos

  este->contador++;
  este->pitch = asin (-(*este->dcm_matriz_renorm[2][0]));
  este->roll = atan2f (*este->dcm_matriz_renorm[2][1], *este-
>dcm_matriz_renorm[2][2]);
  este->pitch = este->pitch * RADIANES_GRADOS;
  este->roll = este->roll * RADIANES_GRADOS;

  // fin código específico funcion normal conversor_angulos
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_conversor_angulos(void *instancia, int t_ciclo) {
```

```
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT conversor_angulos

  // fin código específico funcion normal noRT conversor_angulos
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*********************************************************
*  void funcionFinaliza()
*********************************************************/
void funcion_finaliza_conversor_angulos(void *instancia) {
  struct conversor_angulos *este = (struct conversor_angulos *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza conversor_angulos


  // fin código específico funcion finaliza conversor_angulos
}

/*********************************************************
*  void inicilizaconversor_angulos()
*********************************************************/
int inicializa_conversor_angulos() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_conversor_angulos);
  logPrint(msgLog, 3);
#endif

  clase_conversor_angulos.funcionCrea = funcion_crea_conversor_angulos;
  clase_conversor_angulos.funcionInicializa = funcion_inicializa_conversor_angulos;
  clase_conversor_angulos.funcionNormal = funcion_normal_conversor_angulos;
  clase_conversor_angulos.funcionNormalNoRT =
funcion_normal_noRT_conversor_angulos;
```

```
  clase_conversor_angulos.funcionFinaliza = funcion_finaliza_conversor_angulos;

#ifdef ID_LISTAS
  iniciarLista(&instancias_conversor_angulos);
#else
  clase_conversor_angulos.n = 0;
  clase_conversor_angulos.maxNumComp = MAX_conversor_angulos;
#endif

 clase_conversor_angulos.instancias = &instancias_conversor_angulos;
 clase_conversor_angulos.longComponente = sizeof(struct conversor_angulos);

 return insertarPropiedad2(ID_COMPONENTE, ID_conversor_angulos,
&clase_conversor_angulos, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA DE REFERENCIAS

referencias.h

```
/*
 *  referencias.h
 *
 *  Variables utilizadas en referencias.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_referencias "referencias"
#define MAX_referencias 16

struct referencias {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // salidas
   float reference_pitch;
   float reference_roll;
};
```

referencias.c

```
/*
 *  referencias.c
 *
 *  Componente en el que se definen y se varian las referencias de los angulos pitch y
roll.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
```

```
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "referencias.h"

#ifdef ID_LISTAS
struct lista instancias_referencias;
#else
#define MAX_referencias 16
struct referencias instancias_referencias[MAX_referencias];
#endif

struct componente clase_referencias;

extern char msgLog[];

/*********************************************************
*  void inicializa_propiedades()
*********************************************************/
void inicializa_propiedades_referencias(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct referencias *este = (struct referencias *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas referencias

  // fin inicialización de propiedades específicas referencias
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_referencias(void *instancia) {
  struct referencias *este = (struct referencias *)instancia;

  char saux[MAX_LONG_NOMBRE];

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_referencias),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
```

```
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas referencias

  insertarPropiedad3(este->nombre, &este->reference_pitch, ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "reference_pitch");
  insertarPropiedad3(este->nombre, &este->reference_roll, ID_SAL_FLOAT,
PUBLICO | MODIFICABLE, "reference_roll");

  // fin registro de propiedades específicas referencias
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_referencias(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct referencias *este = (struct referencias *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_referencias(instancia, nombre, orden, habilitado);

  if(registra_propiedades_referencias(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_referencias, orden,
                        &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
```

```
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*****************************************************
*  void funcionInicializa()
*****************************************************/
void funcion_inicializa_referencias(void *instancia) {
  struct referencias *este = (struct referencias *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa referencias

  este->reference_pitch = 0;
  este->reference_roll = 0;

  // fin código específico funcion inicializa referencias
}

/*****************************************************
*  void funcionNormal()
*****************************************************/
void funcion_normal_referencias(void *instancia, int t_ciclo) {
  struct referencias *este = (struct referencias *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal referencias

  //-----

  // fin código específico funcion normal referencias
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
```

```
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*********************************************************
*  void funcionNormalNoRT()
*********************************************************/
void funcion_normal_noRT_referencias(void *instancia, int t_ciclo) {
  struct referencias *este = (struct referencias *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT referencias
  // ...

  // fin código específico funcion normal noRT referencias
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*********************************************************
*  void funcionFinaliza()
*********************************************************/
void funcion_finaliza_referencias(void *instancia) {
  struct referencias *este = (struct referencias *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza referencias


  // fin código específico funcion finaliza referencias
}

/*********************************************************
*  void inicilizareferencias()
*********************************************************/
int inicializa_referencias() {
#ifdef LOG
```

```
  sprintf(msgLog, "Initializing '%s'...", ID_referencias);
  logPrint(msgLog, 3);
#endif

  clase_referencias.funcionCrea = funcion_crea_referencias;
  clase_referencias.funcionInicializa = funcion_inicializa_referencias;
  clase_referencias.funcionNormal = funcion_normal_referencias;
  clase_referencias.funcionNormalNoRT = funcion_normal_noRT_referencias;
  clase_referencias.funcionFinaliza = funcion_finaliza_referencias;

#ifdef ID_LISTAS
  iniciarLista(&instancias_referencias);
#else
  clase_referencias.n = 0;
  clase_referencias.maxNumComp = MAX_referencias;
#endif

 clase_referencias.instancias = &instancias_referencias;
 clase_referencias.longComponente = sizeof(struct referencias);

 return insertarPropiedad2(ID_COMPONENTE, ID_referencias, &clase_referencias,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA PID ANGULO PITCH

pid_pitch.h

```
/*
 *  pid_pitch.h
 *
 *  Variables utilizadas en pid_pitch.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_pid_pitch "pid_pitch"
#define MAX_pid_pitch 16

struct pid_pitch {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   float proportional_pitch;
   float integral_pitch;
   float derivative_pitch;
   float error_pitch;
   float variacion_error_pitch;
   float error_old1_pitch;
   float out_old_pitch;
   float kp_pitch;
   float ki_pitch;
   float kd_pitch;

   // entradas
   float *reference_pitch;
   float *pitch;

   // salidas
   float out_pitch;
};
```

pid_pitch.c

```
/*
 *  pid_pitch.c
 *
 *  Regulador PID para el angulo pitch.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "pid_pitch.h"

#ifdef ID_LISTAS
struct lista instancias_pid_pitch;
#else
#define MAX_pid_pitch 16
struct pid_pitch instancias_pid_pitch[MAX_pid_pitch];
#endif

struct componente clase_pid_pitch;

extern char msgLog[];

extern float fdummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_pid_pitch(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas pid_pitch

  este->reference_pitch = &fdummy;
  este->pitch = &fdummy;

  // fin inicialización de propiedades específicas pid_pitch
}
```

```
/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_pid_pitch(void *instancia) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;

  char saux[MAX_LONG_NOMBRE];

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_pid_pitch),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas pid_pitch

  insertarPropiedad3(este->nombre, &este->proportional_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "proportional_pitch");
  insertarPropiedad3(este->nombre, &este->integral_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "integral_pitch");
  insertarPropiedad3(este->nombre, &este->derivative_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "derivative_pitch");
  insertarPropiedad3(este->nombre, &este->error_pitch, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "error_pitch");
  insertarPropiedad3(este->nombre, &este->variacion_error_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "variacion_error_pitch");
  insertarPropiedad3(este->nombre, &este->error_old1_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "error_old1_pitch");
  insertarPropiedad3(este->nombre, &este->out_old_pitch, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "out_old_pitch");
  insertarPropiedad3(este->nombre, &este->kp_pitch, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "kp_pitch");
```

```
  insertarPropiedad3(este->nombre, &este->ki_pitch, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "ki_pitch");
  insertarPropiedad3(este->nombre, &este->kd_pitch, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "kd_pitch");

  insertarPropiedad3(este->nombre, &este->reference_pitch, ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "reference_pitch");
  insertarPropiedad3(este->nombre, &este->pitch, ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "pitch");

  insertarPropiedad3(este->nombre, &este->out_pitch, ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "out_pitch");

  // fin registro de propiedades específicas pid_pitch
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_pid_pitch(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif
  inicializa_propiedades_pid_pitch(instancia, nombre, orden, habilitado);

  if(registra_propiedades_pid_pitch(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_pid_pitch, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_pid_pitch(void *instancia) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;
#ifdef LOG
```

```
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa pid_pitch

  este->proportional_pitch = 0;
  este->integral_pitch = 0;
  este->derivative_pitch = 0;
  este->error_pitch = 0;
  este->variacion_error_pitch = 0;
  este->error_old1_pitch = 0;
  este->out_old_pitch = 0;

  este->kp_pitch = 0.58;
  este->ki_pitch = 0.0933;
  este->kd_pitch = 0.4725;

  // fin código específico funcion inicializa pid_pitch
}

/*********************************************************
*  void funcionNormal()
*********************************************************/
void funcion_normal_pid_pitch(void *instancia, int t_ciclo) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;
  float t_ciclo_preciso_ns = 0;
  int *t_ciclo_rt = (int *)devolverElemento("SISTEMA.tiempo_ciclo.rt");

#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal pid_pitch

  if ( (*t_ciclo_rt > t_ciclo * 1e6 * 0.95) &&  (*t_ciclo_rt < t_ciclo * 1e6 * 1.05)){

     este->error_pitch = *este->reference_pitch - *este->pitch;

     este->variacion_error_pitch = (este->error_pitch - este->error_old1_pitch);

     este->proportional_pitch = este->kp_pitch * este->error_pitch;
```

```
    este->integral_pitch = este->integral_pitch + este->error_pitch * ((float)*t_ciclo_rt /
1e9);

    if (este->integral_pitch > 150){
      este->integral_pitch = 150;
    }
    else if (este->integral_pitch < -150){
      este->integral_pitch = -150;
    }

    este->derivative_pitch = este->variacion_error_pitch / ((float)*t_ciclo_rt / 1e9);
  }

  este->out_pitch = (0.8 + 0.04*este->error_pitch)*(este->proportional_pitch + este-
>ki_pitch * este->integral_pitch + este->kd_pitch * este->derivative_pitch);

  if (este->out_pitch > 50){
    este->out_pitch = 50;
  }
  else if (este->out_pitch < -50){
    este->out_pitch = -50;
  }

  este->error_old1_pitch = este->error_pitch;
  este->out_old_pitch = este->out_pitch;


  // fin código específico funcion normal pid_pitch
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_pid_pitch(void *instancia, int t_ciclo) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT pid_pitch
```

```
  // fin código específico funcion normal noRT pid_pitch

  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_pid_pitch(void *instancia) {
  struct pid_pitch *este = (struct pid_pitch *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza pid_pitch


  // fin código específico funcion finaliza pid_pitch
}

/*************************************************************
*  void inicilizapid_pitch()
*************************************************************/
int inicializa_pid_pitch() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_pid_pitch);
  logPrint(msgLog, 3);
#endif

  clase_pid_pitch.funcionCrea = funcion_crea_pid_pitch;
  clase_pid_pitch.funcionInicializa = funcion_inicializa_pid_pitch;
  clase_pid_pitch.funcionNormal = funcion_normal_pid_pitch;
  clase_pid_pitch.funcionNormalNoRT = funcion_normal_noRT_pid_pitch;
  clase_pid_pitch.funcionFinaliza = funcion_finaliza_pid_pitch;

#ifdef ID_LISTAS
  iniciarLista(&instancias_pid_pitch);
#else
  clase_pid_pitch.n = 0;
  clase_pid_pitch.maxNumComp = MAX_pid_pitch;
#endif
```

```
 clase_pid_pitch.instancias = &instancias_pid_pitch;
 clase_pid_pitch.longComponente = sizeof(struct pid_pitch);

 return insertarPropiedad2(ID_COMPONENTE, ID_pid_pitch, &clase_pid_pitch,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA REGULACION PID ANGULO ROLL

pid_roll.h

```
/*
 *  pid_roll.h
 *
 *  Variables utilizadas en pid_roll.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */
#include "cosme.h"

#define ID_pid_roll "pid_roll"
#define MAX_pid_roll 16

struct pid_roll {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   float proportional_roll;
   float integral_roll;
   float derivative_roll;
   float error_roll;
   float variacion_error_roll;
   float error_old1_roll;
   float error_old2_roll;
   float error_old3_roll;
   float out_old_roll;
   float kp_roll;
   float ki_roll;
   float kd_roll;

   // entradas
   float *reference_roll;
   float *roll;

   // salidas
   float out_roll;
};
```

pid_roll.c

```
/*
 *  pid_roll.c
 *
 *  Regulador PID para el angulo roll.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "pid_roll.h"

#ifdef ID_LISTAS
struct lista instancias_pid_roll;
#else
#define MAX_pid_roll 16
struct pid_roll instancias_pid_roll[MAX_pid_roll];
#endif

struct componente clase_pid_roll;

extern char msgLog[];

extern float fdummy;

/*************************************************************
*  void inicializa_propiedades()
*************************************************************/
void inicializa_propiedades_pid_roll(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct pid_roll *este = (struct pid_roll *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas pid_roll

  este->reference_roll = &fdummy;
  este->roll = &fdummy;

  // fin inicialización de propiedades específicas pid_roll
}
```

```
/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_pid_roll(void *instancia) {
  struct pid_roll *este = (struct pid_roll *)instancia;

  char saux[MAX_LONG_NOMBRE];

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_pid_roll),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas pid_roll

  insertarPropiedad3(este->nombre, &este->proportional_roll, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "proportional_roll");
  insertarPropiedad3(este->nombre, &este->integral_roll, ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "integral_roll");
  insertarPropiedad3(este->nombre, &este->derivative_roll, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "derivative_roll");
  insertarPropiedad3(este->nombre, &este->error_roll, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "error_roll");
  insertarPropiedad3(este->nombre, &este->variacion_error_roll, ID_VAR_FLOAT,
PUBLICO | MODIFICABLE, "variacion_error_roll");
  insertarPropiedad3(este->nombre, &este->out_old_roll, ID_VAR_FLOAT, PUBLICO
| MODIFICABLE, "out_old_roll");
  insertarPropiedad3(este->nombre, &este->kp_roll, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "kp_roll");
  insertarPropiedad3(este->nombre, &este->ki_roll, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "ki_roll");
```

```
  insertarPropiedad3(este->nombre, &este->kd_roll, ID_VAR_FLOAT, PUBLICO |
MODIFICABLE, "kd_roll");

  insertarPropiedad3(este->nombre, &este->reference_roll, ID_ENT_FLOAT,
PUBLICO | MODIFICABLE, "reference_roll");
  insertarPropiedad3(este->nombre, &este->roll, ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "roll");

  insertarPropiedad3(este->nombre, &este->out_roll, ID_SAL_FLOAT, PUBLICO |
MODIFICABLE, "out_roll");

  // fin registro de propiedades específicas pid_roll
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_pid_roll(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct pid_roll *este = (struct pid_roll *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_pid_roll(instancia, nombre, orden, habilitado);

  if(registra_propiedades_pid_roll(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_pid_roll, orden,
                        &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_pid_roll(void *instancia) {
  struct pid_roll *este = (struct pid_roll *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
```

```
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa pid_roll

  este->proportional_roll = 0;
  este->integral_roll = 0;
  este->derivative_roll = 0;
  este->error_roll = 0;
  este->variacion_error_roll = 0;
  este->error_old1_roll = 0;
  este->out_old_roll = 0;

  este->kp_roll = 0.58;
  este->ki_roll = 0.0933;
  este->kd_roll = 0.4725;

  // fin código específico funcion inicializa pid_roll
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_pid_roll(void *instancia, int t_ciclo) {
  struct pid_roll *este = (struct pid_roll *)instancia;
  int *t_ciclo_rt = (int *)devolverElemento("SISTEMA.tiempo_ciclo.rt");

#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal pid_roll

  if ( (*t_ciclo_rt > t_ciclo * 1e6 * 0.95) &&  (*t_ciclo_rt < t_ciclo * 1e6 * 1.05)){

    este->error_roll = *este->reference_roll - *este->roll;

    este->variacion_error_roll = (este->error_roll - este->error_old1_roll);

    este->proportional_roll = este->kp_roll * este->error_roll;

    este->integral_roll = este->integral_roll + este->error_roll * ((float)*t_ciclo_rt /
1e9);
```

```
    if (este->integral_roll > 150){
      este->integral_roll = 150;
    }
    else if (este->integral_roll < -150){
      este->integral_roll = -150;
    }

    este->derivative_roll = este->variacion_error_roll / ((float)*t_ciclo_rt / 1e9);
  }

  este->out_roll = (0.8 + 0.04*este->error_roll)*(este->proportional_roll + este->ki_roll
* este->integral_roll + este->kd_roll * este->derivative_roll);

  if (este->out_roll > 50){
    este->out_roll = 50;
  }
  else if (este->out_roll < -50){
    este->out_roll = -50;
  }

  este->error_old1_roll = este->error_roll;
  este->out_old_roll = este->out_roll;

  // fin código específico funcion normal pid_roll
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_pid_roll(void *instancia, int t_ciclo) {
  struct pid_roll *este = (struct pid_roll *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT pid_roll

  // fin código específico funcion normal noRT pid_roll
  este->t_ciclo_noRT = crononsec(0, &ts);
```

```
  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*****************************************************************
*  void funcionFinaliza()
*****************************************************************/
void funcion_finaliza_pid_roll(void *instancia) {
  struct pid_roll *este = (struct pid_roll *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza pid_roll


  // fin código específico funcion finaliza pid_roll
}

/*****************************************************************
*  void inicilizapid_roll()
*****************************************************************/
int inicializa_pid_roll() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_pid_roll);
  logPrint(msgLog, 3);
#endif

  clase_pid_roll.funcionCrea = funcion_crea_pid_roll;
  clase_pid_roll.funcionInicializa = funcion_inicializa_pid_roll;
  clase_pid_roll.funcionNormal = funcion_normal_pid_roll;
  clase_pid_roll.funcionNormalNoRT = funcion_normal_noRT_pid_roll;
  clase_pid_roll.funcionFinaliza = funcion_finaliza_pid_roll;

#ifdef ID_LISTAS
  iniciarLista(&instancias_pid_roll);
#else
  clase_pid_roll.n = 0;
  clase_pid_roll.maxNumComp = MAX_pid_roll;
#endif

 clase_pid_roll.instancias = &instancias_pid_roll;
 clase_pid_roll.longComponente = sizeof(struct pid_roll);
```

```
 return insertarPropiedad2(ID_COMPONENTE, ID_pid_roll, &clase_pid_roll,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA DE CAMBIO DE MAGNITUD

cambio_magnitud.h

```
/*
 *  cambio_magnitud.h
 *
 *  Variables utilizadas en cambio_magnitud.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_cambio_magnitud "cambio_magnitud"
#define MAX_cambio_magnitud 16

struct cambio_magnitud {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;
   int potencia;

   // entradas
   float *out_pitch;
   float *out_roll;

   // salidas
   int pwm_motor_gris1;
   int pwm_motor_gris2;
   int pwm_motor_negro1;
   int pwm_motor_negro2;
};
```

cambio_magnitud.c

```
/*
 *  cambio_magnitud.c
 *
 *  Calcula el ciclo PWM para cada motor a partir de los reguladores PID's
```

```
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "runtime.h"
#include "cambio_magnitud.h"

#ifdef ID_LISTAS
struct lista instancias_cambio_magnitud;
#else
#define MAX_cambio_magnitud 16
struct cambio_magnitud instancias_cambio_magnitud[MAX_cambio_magnitud];
#endif

struct componente clase_cambio_magnitud;

extern char msgLog[];

extern float fdummy;

/*********************************************************
*  void inicializa_propiedades()
*********************************************************/
void inicializa_propiedades_cambio_magnitud(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas cambio_magnitu

  este->out_pitch = &fdummy;
  este->out_roll = &fdummy;

  // fin inicialización de propiedades específicas cambio_magnitud
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_cambio_magnitud(void *instancia) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
```

```
  char saux[MAX_LONG_NOMBRE];

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA,
ID_cambio_magnitud), NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas cambio_magnitud

  insertarPropiedad3(este->nombre, &este->out_pitch, ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "out_pitch");
  insertarPropiedad3(este->nombre, &este->out_roll, ID_ENT_FLOAT, PUBLICO |
MODIFICABLE, "out_roll");

  insertarPropiedad3(este->nombre, &este->pwm_motor_gris1, ID_SAL_INT,
PUBLICO | MODIFICABLE, "pwm_motor_gris1");
  insertarPropiedad3(este->nombre, &este->pwm_motor_gris2, ID_SAL_INT,
PUBLICO | MODIFICABLE, "pwm_motor_gris2");
  insertarPropiedad3(este->nombre, &este->pwm_motor_negro1, ID_SAL_INT,
PUBLICO | MODIFICABLE, "pwm_motor_negro1");
  insertarPropiedad3(este->nombre, &este->pwm_motor_negro2, ID_SAL_INT,
PUBLICO | MODIFICABLE, "pwm_motor_negro2");

  insertarPropiedad3(este->nombre, &este->potencia, ID_VAR_INT, PUBLICO |
MODIFICABLE, "potencia");

  // fin registro de propiedades específicas cambio_magnitud
}

/*******************************************************
*  void funcionCrea()
```

```
**************************************************************/
int funcion_crea_cambio_magnitud(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_cambio_magnitud(instancia, nombre, orden, habilitado);

  if(registra_propiedades_cambio_magnitud(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_cambio_magnitud, orden,
                  &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

/*************************************************************
*  void funcionInicializa()
**************************************************************/
void funcion_inicializa_cambio_magnitud(void *instancia) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa cambio_magnitud

  este->pwm_motor_gris1 = 0;
  este->pwm_motor_gris2 = 0;
  este->pwm_motor_negro1 = 0;
  este->pwm_motor_negro2 = 0;

  este->potencia = 1800;
```

```
  // fin código específico funcion inicializa cambio_magnitud
}

/*************************************************************
*  void funcionNormal()
*************************************************************/
void funcion_normal_cambio_magnitud(void *instancia, int t_ciclo) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal cambio_magnitud


  if ((*este->out_pitch > 0.005) || (*este->out_pitch < -0.005)){
    este->pwm_motor_gris1  = floor(este->potencia + (*este->out_pitch * 0.95 * 1) +
0.5);
    este->pwm_motor_gris2  = floor(este->potencia - (*este->out_pitch * 1.0 * 1) + 0.5);
  }
  else{
    este->pwm_motor_gris1 = este->potencia;
    este->pwm_motor_gris2 = este->potencia;
  }


  if ((*este->out_roll > 0.005) || (*este->out_roll < -0.005)){
    este->pwm_motor_negro1  = floor(este->potencia + (*este->out_roll * 0.98 * 1) +
0.5);
    este->pwm_motor_negro2  = floor(este->potencia - (*este->out_roll * 1.02 * 1) +
0.5);
  }
  else{
    este->pwm_motor_negro1 = este->potencia;
    este->pwm_motor_negro2 = este->potencia;
  }



  if (este->pwm_motor_gris1 > 2800){
    este->pwm_motor_gris1 = 2800;
  }
  else if (este->pwm_motor_gris2 > 2800){
    este->pwm_motor_gris2 = 2800;
  }
  else if (este->pwm_motor_gris1 < 1800){
    este->pwm_motor_gris1 = 1800;
```

```
  }
  else if (este->pwm_motor_gris2 < 1800){
    este->pwm_motor_gris2 = 1800;
  }


  if (este->pwm_motor_negro1 > 2800){
    este->pwm_motor_negro1 = 2800;
  }
  else if (este->pwm_motor_negro2 > 2800){
    este->pwm_motor_negro2 = 2800;
  }
  else if (este->pwm_motor_negro1 < 1800){
    este->pwm_motor_negro1 = 1800;
  }
  else if (este->pwm_motor_negro2 < 1800){
    este->pwm_motor_negro2 = 1800;
  }


  // fin código específico funcion normal cambio_magnitud
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_cambio_magnitud(void *instancia, int t_ciclo) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal noRT cambio_magnitud
  // fin código específico funcion normal noRT cambio_magnitud
  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
```

```
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_cambio_magnitud(void *instancia) {
  struct cambio_magnitud *este = (struct cambio_magnitud *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza cambio_magnitud

  este->pwm_motor_gris1 = 0;
  este->pwm_motor_gris2 = 0;
  este->pwm_motor_negro1 = 0;
  este->pwm_motor_negro2 = 0;

  // fin código específico funcion finaliza cambio_magnitud
}

/*************************************************************
*  void inicilizacambio_magnitud()
*************************************************************/
int inicializa_cambio_magnitud() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_cambio_magnitud);
  logPrint(msgLog, 3);
#endif

  clase_cambio_magnitud.funcionCrea = funcion_crea_cambio_magnitud;
  clase_cambio_magnitud.funcionInicializa = funcion_inicializa_cambio_magnitud;
  clase_cambio_magnitud.funcionNormal = funcion_normal_cambio_magnitud;
  clase_cambio_magnitud.funcionNormalNoRT =
funcion_normal_noRT_cambio_magnitud;
  clase_cambio_magnitud.funcionFinaliza = funcion_finaliza_cambio_magnitud;

#ifdef ID_LISTAS
  iniciarLista(&instancias_cambio_magnitud);
#else
  clase_cambio_magnitud.n = 0;
  clase_cambio_magnitud.maxNumComp = MAX_cambio_magnitud;
#endif

 clase_cambio_magnitud.instancias = &instancias_cambio_magnitud;
 clase_cambio_magnitud.longComponente = sizeof(struct cambio_magnitud);
```

```
 return insertarPropiedad2(ID_COMPONENTE, ID_cambio_magnitud,
&clase_cambio_magnitud, ID_COMPONENTE, NO_MODIFICABLE);
}
```

INSTANCIA GENERADORA DE LA SEÑAL PWM

microPWM.h

```
/*
 *  microPWM.h
 *
 *  Variables utilizadas en microPWM.c
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include "cosme.h"

#define ID_microPWM "microPWM"
#define MAX_microPWM 16
#define  PCA9685_I2C_ADDR  0x41

#define SUBADR1            0x02
#define SUBADR2            0x03
#define SUBADR3            0x04
#define MODE1              0x00
#define PRESCALE           0xFE
#define LED0_ON_L          0x06
#define LED0_ON_H          0x07
#define LED0_OFF_L         0x08
#define LED0_OFF_H         0x09
#define ALLLED_ON_L        0xFA
#define ALLLED_ON_H        0xFB
#define ALLLED_OFF_L       0xFC
#define ALLLED_OFF_H       0xFD

#define APAGAR      0
#define MAX_ERR_MSG  100

struct microPWM {
   char nombre[MAX_LONG_NOMBRE];
   int orden;
   byte habilitado;
   byte finalizado;
   int t_ciclo_RT;
   int t_ciclo_RT_min;
   int t_ciclo_RT_max;
   int t_ciclo_noRT;
   int t_ciclo_noRT_min;
   int t_ciclo_noRT_max;

   // variables
   char error_msg_micro_PWM[MAX_ERR_MSG];
```

```
    float prescale;
    float  prescaleval;
    int freq;
    int condicion;
    int ciclo_pwm;

    // entradas
    int *bus_ok;
    int *fd;
    int *pwm_motor_gris1;
    int *pwm_motor_gris2;
    int *pwm_motor_negro1;
    int *pwm_motor_negro2;

    // salidas
};
```

microPWM.c

```
/*
 *  microPWM.c
 *
 *  Se configura el microcontrolador PCA9685 y se escribe en sus registros el valor de
PWM a generar.
 *  11/06/2014
 *  Eugenio Alcalá Baselga
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include "local-i2c-dev.h"
#include <linux/types.h>

#include "runtime.h"
#include "microPWM.h"

#ifdef ID_LISTAS
struct lista instancias_microPWM;
#else
#define MAX_microPWM 16
struct microPWM instancias_microPWM[MAX_microPWM];
#endif
```

```
struct componente clase_microPWM;

extern char msgLog[];

extern int dummy;

/*********************************************************
*  void inicializa_propiedades()
*********************************************************/
void inicializa_propiedades_microPWM(
  void *instancia, char *nombre,
  int orden, unsigned char habilitado) {
  struct microPWM *este = (struct microPWM *)instancia;

  strcpy(este->nombre, nombre);
  este->orden = orden;
  este->habilitado = habilitado;
  este->finalizado = 0;

  // principio inicialización de propiedades específicas microPWM

  este->fd = &dummy;
  este->bus_ok = &dummy;
  este->pwm_motor_gris1 = &dummy;
  este->pwm_motor_gris2 = &dummy;
  este->pwm_motor_negro1 = &dummy;
  este->pwm_motor_negro2 = &dummy;

  // fin inicialización de propiedades específicas microPWM
}

/*********************************************************
*  void registra_propiedades()
*********************************************************/
int registra_propiedades_microPWM(void *instancia) {
  struct microPWM *este = (struct microPWM *)instancia;

  char saux[MAX_LONG_NOMBRE];

  insertarNombre2(este->nombre, este, emStrcpy(saux, ID_SISTEMA, ID_microPWM),
NO_MODIFICABLE);
  insertarPropiedad3(este->nombre, &este->nombre, ID_VAR_TEXT, PUBLICO,
"nombre");
  insertarPropiedad3(este->nombre, &este->orden, ID_VAR_INT, PUBLICO |
MODIFICABLE, "orden");
  insertarPropiedad3(este->nombre, &este->habilitado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "habilitado");
  insertarPropiedad3(este->nombre, &este->finalizado, ID_VAR_BYTE, PUBLICO |
MODIFICABLE, "finalizado");
```

```
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_RT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_min, ID_VAR_INT, PUBLICO
| MODIFICABLE, "t_ciclo_RT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_RT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_RT_max");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT, ID_VAR_INT, PUBLICO |
MODIFICABLE, "t_ciclo_noRT");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_min, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_min");
  insertarPropiedad3(este->nombre, &este->t_ciclo_noRT_max, ID_VAR_INT,
PUBLICO | MODIFICABLE, "t_ciclo_noRT_max");

  // principio registro de propiedades específicas microPWM

  insertarPropiedad3(este->nombre, &este->bus_ok, ID_ENT_INT, PUBLICO
|MODIFICABLE, "bus_ok");
  insertarPropiedad3(este->nombre, &este->fd, ID_ENT_INT, PUBLICO
|MODIFICABLE, "fd");
  insertarPropiedad3(este->nombre, &este->pwm_motor_gris1, ID_ENT_INT,
PUBLICO |MODIFICABLE, "pwm_motor_gris1");
  insertarPropiedad3(este->nombre, &este->pwm_motor_gris2, ID_ENT_INT,
PUBLICO |MODIFICABLE, "pwm_motor_gris2");
  insertarPropiedad3(este->nombre, &este->pwm_motor_negro1, ID_ENT_INT,
PUBLICO |MODIFICABLE, "pwm_motor_negro1");
  insertarPropiedad3(este->nombre, &este->pwm_motor_negro2, ID_ENT_INT,
PUBLICO |MODIFICABLE, "pwm_motor_negro2");
  insertarPropiedad3(este->nombre, &este->condicion, ID_VAR_INT, PUBLICO
|MODIFICABLE, "condicion");
  insertarPropiedad3(este->nombre, &este->ciclo_pwm, ID_VAR_INT, PUBLICO
|MODIFICABLE, "ciclo_pwm");
//  insertarPropiedad3(este->nombre, &este->freq, ID_VAR_INT, PUBLICO
|MODIFICABLE, "frecuencia");

  // fin registro de propiedades específicas microPWM
}

/*************************************************************
*  void funcionCrea()
*************************************************************/
int funcion_crea_microPWM(
  void *instancia, char *nombre, int orden, unsigned char habilitado) {
  struct microPWM *este = (struct microPWM *)instancia;
#ifdef LOG
  sprintf(msgLog, "Creating '%s'...", nombre);
  logPrint(msgLog, 3);
#endif

  inicializa_propiedades_microPWM(instancia, nombre, orden, habilitado);
```

```
  if(registra_propiedades_microPWM(instancia) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering properties of '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }

  if(registrarInstancia(instancia, nombre, &clase_microPWM, orden,
                    &este->habilitado, &este->finalizado) == 0) {
#ifdef LOG
    sprintf(msgLog, "%s \"Registering instance '%s'\"", ID_ERROR, este->nombre);
    logPrint(msgLog, 1);
#endif
  }
}

void writeto_PCA9685 (int fd, int reg, int val){
     char buf[2];
     buf[0] = reg;
     buf[1] = val;
     if (write (fd, buf, 2) != 2){
         fprintf(stderr, "No se puede escribir en el dispositivo i2c\n");
     }
}

void setPWM ( int fd, int channel, int on, int off){
  i2c_smbus_write_byte_data(fd, LED0_ON_L + 4*channel, on & 0xFF);
  i2c_smbus_write_byte_data(fd, LED0_ON_H + 4*channel, on >> 8);
  i2c_smbus_write_byte_data(fd, LED0_OFF_L + 4*channel, off & 0xFF);
  i2c_smbus_write_byte_data(fd, LED0_OFF_H + 4*channel, off >> 8);
}

/*************************************************************
*  void funcionInicializa()
*************************************************************/
void funcion_inicializa_microPWM(void *instancia) {
  struct microPWM *este = (struct microPWM *)instancia;
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif
  este->t_ciclo_RT_min = 0x10000000;
  este->t_ciclo_RT_max = 0;
  este->t_ciclo_noRT_min = 0x10000000;
  este->t_ciclo_noRT_max = 0;

  // principio código específico funcion inicializa microPWM

  este->condicion = 0;
```

```
  if (ioctl (*este->fd, I2C_SLAVE, PCA9685_I2C_ADDR) < 0){
    strcpy(este->error_msg_micro_PWM, "PCA9685 is not present\n");
  }

  writeto_PCA9685(*este->fd, 0x00, 0x00);  //Configuramos el registro MODE1.

  este->freq = 333;

  este->prescaleval = (25000000 / (float)(4096 * este->freq)) - 1;

  if(1){
    printf ("Frecuencia PWM a %i Hz\n",este->freq);
    printf ("Preescalado estimado: %3.2f \n",este->prescaleval);
  }

  este->prescale = floor(este->prescaleval + 0.5);    //floor redondea hacia abajo

  if(1){
    printf ("Preescalado final: %3.2f\n", este->prescale);
  }

  int oldmode = i2c_smbus_read_byte_data(*este->fd, MODE1);
  int newmode = (oldmode & 0x7F) | 0x10;

  i2c_smbus_write_byte_data(*este->fd, MODE1, newmode);

  i2c_smbus_write_byte_data(*este->fd, PRESCALE, (int)floor(este->prescale));

  i2c_smbus_write_byte_data(*este->fd, MODE1, oldmode);

  sleep(0.005);

  i2c_smbus_write_byte_data(*este->fd, MODE1, oldmode | 0x80);

  setPWM(*este->fd,4,0,1710);
  setPWM(*este->fd,2,0,1710);
  setPWM(*este->fd,8,0,1710);
  setPWM(*este->fd,10,0,1710);

// fin código específico funcion inicializa microPWM
}

/*********************************************************
*  void funcionNormal()
*********************************************************/
void funcion_normal_microPWM(void *instancia, int t_ciclo) {
  struct microPWM *este = (struct microPWM *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
```

```
#endif
  struct timespec ts;
  crononsec(1, &ts);

  // principio código específico funcion normal microPWM

  if (ioctl (*este->fd, I2C_SLAVE, PCA9685_I2C_ADDR) < 0){
    strcpy(este->error_msg_micro_PWM, "PCA9685 is not present\n");
  }

  if (este->condicion == 0){
    setPWM(*este->fd,4,0,1710);
    setPWM(*este->fd,2,0,1710);
    setPWM(*este->fd,8,0,1710);
    setPWM(*este->fd,10,0,1710);
  }
  else if (este->condicion == 1){
    setPWM(*este->fd,4,0,*este->pwm_motor_gris1);
    setPWM(*este->fd,2,0,*este->pwm_motor_gris2);
    setPWM(*este->fd,8,0,*este->pwm_motor_negro1);
    setPWM(*este->fd,10,0,*este->pwm_motor_negro2);
  }
  else if (este->condicion == 2){
    setPWM(*este->fd,4,0,este->ciclo_pwm);
    setPWM(*este->fd,2,0,este->ciclo_pwm);
    setPWM(*este->fd,8,0,este->ciclo_pwm);
    setPWM(*este->fd,10,0,este->ciclo_pwm);
  }


  // fin código específico funcion normal microPWM
  este->t_ciclo_RT = crononsec(0, &ts);

  if(este->t_ciclo_RT > este->t_ciclo_RT_max) {
    este->t_ciclo_RT_max = este->t_ciclo_RT;
  }
  else if(este->t_ciclo_RT < este->t_ciclo_RT_min) {
    este->t_ciclo_RT_min = este->t_ciclo_RT;
  }
}
/*************************************************************
*  void funcionNormalNoRT()
*************************************************************/
void funcion_normal_noRT_microPWM(void *instancia, int t_ciclo) {
  struct microPWM *este = (struct microPWM *)instancia;
#ifdef LOG
//  sprintf(msgLog, "Normal function '%s'...", este->nombre);
//  logPrint(msgLog, 6);
#endif
  struct timespec ts;
```

```
  crononsec(1, &ts);

  // principio código específico funcion normal noRT microPWM

  // fin código específico funcion normal noRT microPWM

  este->t_ciclo_noRT = crononsec(0, &ts);

  if(este->t_ciclo_noRT > este->t_ciclo_noRT_max) {
    este->t_ciclo_noRT_max = este->t_ciclo_noRT;
  }
  else if(este->t_ciclo_noRT < este->t_ciclo_noRT_min) {
    este->t_ciclo_noRT_min = este->t_ciclo_noRT;
  }
}
/*************************************************************
*  void funcionFinaliza()
*************************************************************/
void funcion_finaliza_microPWM(void *instancia) {
  struct microPWM *este = (struct microPWM *)instancia;
#ifdef LOG
  sprintf(msgLog, "Finalizing '%s'...", este->nombre);
  logPrint(msgLog, 3);
#endif

  // principio código específico funcion finaliza microPWM
  if (ioctl (*este->fd, I2C_SLAVE, PCA9685_I2C_ADDR) < 0){
    strcpy(este->error_msg_micro_PWM, "PCA9685 is not present\n");
  }

  setPWM(*este->fd,4,0,1710);
  setPWM(*este->fd,2,0,1710);
  setPWM(*este->fd,8,0,1710);
  setPWM(*este->fd,10,0,1710);

  close(*este->fd);

  // fin código específico funcion finaliza microPWM
}

/*************************************************************
*  void inicilizamicroPWM()
*************************************************************/
int inicializa_microPWM() {
#ifdef LOG
  sprintf(msgLog, "Initializing '%s'...", ID_microPWM);
  logPrint(msgLog, 3);
#endif

  clase_microPWM.funcionCrea = funcion_crea_microPWM;
```

```
  clase_microPWM.funcionInicializa = funcion_inicializa_microPWM;
  clase_microPWM.funcionNormal = funcion_normal_microPWM;
  clase_microPWM.funcionNormalNoRT = funcion_normal_noRT_microPWM;
  clase_microPWM.funcionFinaliza = funcion_finaliza_microPWM;

#ifdef ID_LISTAS
  iniciarLista(&instancias_microPWM);
#else
  clase_microPWM.n = 0;
  clase_microPWM.maxNumComp = MAX_microPWM;
#endif

 clase_microPWM.instancias = &instancias_microPWM;
 clase_microPWM.longComponente = sizeof(struct microPWM);

 return insertarPropiedad2(ID_COMPONENTE, ID_microPWM, &clase_microPWM,
ID_COMPONENTE, NO_MODIFICABLE);
}
```

CONEXIONES DE LOS COMPONENTES

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<cosme version="3.2">
  <date/>
  <author/>
  <version/>
  <description/>
  <VERSION_IDE/>
  <cicle_time/>
  <project_constants/>
  <project_variables/>
  <exec_sequences/>
  <layers/>
  <tabs/>
  <instances>
    <instance>
      <instance_name>i2c1</instance_name>
      <component>SISTEMA.COMPONENTE.i2c</component>
      <specific_component/>
      <creation_date/>
      <order>1</order>
      <collapsed/>
      <default_location/>
      <tabs/>
      <enabled>true</enabled>
    </instance>
    <instance>
      <instance_name>acelerometro1</instance_name>
      <component>SISTEMA.COMPONENTE.acelerometro</component>
      <specific_component/>
      <creation_date/>
      <order>2</order>
      <collapsed/>
      <default_location/>
      <tabs/>
      <enabled>true</enabled>
    </instance>
    <instance>
      <instance_name>giroscopo1</instance_name>
      <component>SISTEMA.COMPONENTE.giroscopo</component>
      <specific_component/>
      <creation_date/>
      <order>3</order>
      <collapsed/>
      <default_location/>
      <tabs/>
      <enabled>true</enabled>
    </instance>
    <instance>
```

```
  <instance_name>magnetometro1</instance_name>
  <component>SISTEMA.COMPONENTE.magnetometro</component>
  <specific_component/>
  <creation_date/>
  <order>4</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>correcionPI_Acc1</instance_name>
  <component>SISTEMA.COMPONENTE.correcionPI_Acc</component>
  <specific_component/>
  <creation_date/>
  <order>5</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>promediador1</instance_name>
  <component>SISTEMA.COMPONENTE.promediador</component>
  <specific_component/>
  <creation_date/>
  <order>6</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>correccionPI_Mag1</instance_name>
  <component>SISTEMA.COMPONENTE.correccionPI_Mag</component>
  <specific_component/>
  <creation_date/>
  <order>7</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>actualizar_matriz1</instance_name>
  <component>SISTEMA.COMPONENTE.actualizar_matriz</component>
  <specific_component/>
  <creation_date/>
  <order>8</order>
  <collapsed/>
```

```
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>renormalizar1</instance_name>
  <component>SISTEMA.COMPONENTE.renormalizar</component>
  <specific_component/>
  <creation_date/>
  <order>9</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>conversor_angulos1</instance_name>
  <component>SISTEMA.COMPONENTE.conversor_angulos</component>
  <specific_component/>
  <creation_date/>
  <order>10</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>referencias1</instance_name>
  <component>SISTEMA.COMPONENTE.referencias</component>
  <specific_component/>
  <creation_date/>
  <order>11</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>pid_pitch1</instance_name>
  <component>SISTEMA.COMPONENTE.pid_pitch</component>
  <specific_component/>
  <creation_date/>
  <order>12</order>
  <collapsed/>
  <default_location/>
  <tabs/>
  <enabled>true</enabled>
</instance>
<instance>
  <instance_name>pid_roll1</instance_name>
```

```
    <component>SISTEMA.COMPONENTE.pid_roll</component>
    <specific_component/>
    <creation_date/>
    <order>13</order>
    <collapsed/>
    <default_location/>
    <tabs/>
    <enabled>true</enabled>
   </instance>
   <instance>
    <instance_name>cambio_magnitud1</instance_name>
    <component>SISTEMA.COMPONENTE.cambio_magnitud</component>
    <specific_component/>
    <creation_date/>
    <order>14</order>
    <collapsed/>
    <default_location/>
    <tabs/>
    <enabled>true</enabled>
   </instance>
   <instance>
    <instance_name>microPWM1</instance_name>
    <component>SISTEMA.COMPONENTE.microPWM</component>
    <specific_component/>
    <creation_date/>
    <order>15</order>
    <collapsed/>
    <default_location/>
    <tabs/>
    <enabled>true</enabled>
   </instance>
  </instances>
  <instance_components/>
  <connections>
   <connection>
    <input>acelerometro1.fd</input>
    <input_resending/>
    <input_type>ID_ENT_INT</input_type>
    <input_dimensions/>
    <output_resending/>
    <output>i2c1.fd</output>
    <sequence_name/>
    <connection_type/>
   </connection>
   <connection>
    <input>acelerometro1.bus_ok</input>
    <input_resending/>
    <input_type>ID_ENT_INT</input_type>
    <input_dimensions/>
    <output_resending/>
```

```
  <output>i2c1.bus_ok</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>giroscopo1.fd</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>i2c1.fd</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>giroscopo1.bus_ok</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>i2c1.bus_ok</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>magnetometro1.fd</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>i2c1.fd</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>magnetometro1.bus_ok</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>i2c1.bus_ok</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.Facc[0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>acelerometro1.Facc[0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.Facc[1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>acelerometro1.Facc[1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.Facc[2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>acelerometro1.Facc[2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.Wgyr[0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>giroscopo1.Wgyr[0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.Wgyr[1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>giroscopo1.Wgyr[1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.Wgyr[2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

```
  <output>giroscopo1.Wgyr[2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.dcm_matriz_renorm_1[0][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[0][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.dcm_matriz_renorm_1[0][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[0][1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.dcm_matriz_renorm_1[0][2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[0][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.dcm_matriz_renorm_1[1][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[1][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correcionPI_Acc1.dcm_matriz_renorm_1[1][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>renormalizar1.dcm_matriz_renorm_1[1][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.dcm_matriz_renorm_1[1][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.dcm_matriz_renorm_1[2][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.dcm_matriz_renorm_1[2][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correcionPI_Acc1.dcm_matriz_renorm_1[2][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>promediador1.Wgyr_modif[0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

```
  <output>correcionPI_Acc1.Wgyr_modif[0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>promediador1.Wgyr_modif[1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>correcionPI_Acc1.Wgyr_modif[1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>promediador1.Wgyr_modif[2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>correcionPI_Acc1.Wgyr_modif[2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>promediador1.pitch</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>conversor_angulos1.pitch</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correccionPI_Mag1.Vmag[0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>magnetometro1.Vmag[0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correccionPI_Mag1.Vmag[1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>magnetometro1.Vmag[1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.Vmag[2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>magnetometro1.Vmag[2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[0][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[0][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[0][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[0][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[0][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[0][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[1][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

<output>renormalizar1.dcm_matriz_renorm_1[1][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[1][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[1][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[2][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[2][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>correccionPI_Mag1.dcm_matriz_renorm_1[2][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

```
  <output>renormalizar1.dcm_matriz_renorm_1[2][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correccionPI_Mag1.Wgyr_modif_prom[0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>promediador1.Wgyr_modif_prom[0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correccionPI_Mag1.Wgyr_modif_prom[1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>promediador1.Wgyr_modif_prom[1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>correccionPI_Mag1.Wgyr_modif_prom[2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>promediador1.Wgyr_modif_prom[2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.Wgyr_fin[0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>correccionPI_Mag1.Wgyr_fin[0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.Wgyr_fin[1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

```
  <output>correccionPI_Mag1.Wgyr_fin[1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.Wgyr_fin[2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>correccionPI_Mag1.Wgyr_fin[2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.calibracion_giro</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>giroscopo1.calibracion_giro</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.dcm_matriz_renorm_1[0][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[0][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.dcm_matriz_renorm_1[0][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>renormalizar1.dcm_matriz_renorm_1[0][1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>actualizar_matriz1.dcm_matriz_renorm_1[0][2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>renormalizar1.dcm_matriz_renorm_1[0][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[1][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[1][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[1][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[1][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[2][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[2][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

<output>renormalizar1.dcm_matriz_renorm_1[2][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>actualizar_matriz1.dcm_matriz_renorm_1[2][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm_1[2][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[0][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[0][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[0][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[0][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[0][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[0][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[1][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

<output>actualizar_matriz1.dcm_matriz_1[1][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[1][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[1][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[1][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[1][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[2][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[2][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[2][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>actualizar_matriz1.dcm_matriz_1[2][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>renormalizar1.dcm_matriz_1[2][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

```
  <output>actualizar_matriz1.dcm_matriz_1[2][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[0][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[0][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[0][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[0][1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[0][2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[0][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[1][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[1][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[1][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

```
  <output>actualizar_matriz1.DCM_Matriz[1][1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[1][2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[1][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[2][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[2][0]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[2][1]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[2][1]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>renormalizar1.DCM_Matriz[2][2]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>actualizar_matriz1.DCM_Matriz[2][2]</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>conversor_angulos1.dcm_matriz_renorm[0][0]</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>renormalizar1.dcm_matriz_renorm[0][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[0][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[0][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[0][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[0][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[1][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[1][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[1][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[1][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[1][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

<output>renormalizar1.dcm_matriz_renorm[1][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[2][0]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[2][0]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[2][1]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[2][1]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>conversor_angulos1.dcm_matriz_renorm[2][2]</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>renormalizar1.dcm_matriz_renorm[2][2]</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>pid_pitch1.reference_pitch</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>
<output>referencias1.reference_pitch</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>pid_roll1.reference_roll</input>
<input_resending/>
<input_type>ID_ENT_FLOAT</input_type>
<input_dimensions/>
<output_resending/>

```
  <output>referencias1.reference_roll</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>pid_pitch1.pitch</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>conversor_angulos1.pitch</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>pid_roll1.roll</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>conversor_angulos1.roll</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>cambio_magnitud1.out_pitch</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>pid_pitch1.out_pitch</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>cambio_magnitud1.out_roll</input>
  <input_resending/>
  <input_type>ID_ENT_FLOAT</input_type>
  <input_dimensions/>
  <output_resending/>
  <output>pid_roll1.out_roll</output>
  <sequence_name/>
  <connection_type/>
</connection>
<connection>
  <input>microPWM1.fd</input>
  <input_resending/>
  <input_type>ID_ENT_INT</input_type>
  <input_dimensions/>
  <output_resending/>
```

<output>i2c1.fd</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>microPWM1.bus_ok</input>
<input_resending/>
<input_type>ID_ENT_INT</input_type>
<input_dimensions/>
<output_resending/>
<output>i2c1.bus_ok</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>microPWM1.pwm_motor_gris1</input>
<input_resending/>
<input_type>ID_ENT_INT</input_type>
<input_dimensions/>
<output_resending/>
<output>cambio_magnitud1.pwm_motor_gris1</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>microPWM1.pwm_motor_gris2</input>
<input_resending/>
<input_type>ID_ENT_INT</input_type>
<input_dimensions/>
<output_resending/>
<output>cambio_magnitud1.pwm_motor_gris2</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>microPWM1.pwm_motor_negro1</input>
<input_resending/>
<input_type>ID_ENT_INT</input_type>
<input_dimensions/>
<output_resending/>
<output>cambio_magnitud1.pwm_motor_negro1</output>
<sequence_name/>
<connection_type/>
</connection>
<connection>
<input>microPWM1.pwm_motor_negro2</input>
<input_resending/>
<input_type>ID_ENT_INT</input_type>
<input_dimensions/>
<output_resending/>

```
      <output>cambio_magnitud1.pwm_motor_negro2</output>
      <sequence_name/>
      <connection_type/>
    </connection>
  </connections>
  <grafcets/>
  <registers/>
  <stubs/>
  <resendings/>
</cosme>
```

MAKEFILE

```
CC = gcc
RUNTIME_PATH = $(COSME_HOME)/runtime/
LIB_PATH = $(COSME_HOME)/lib/

all: lib \
i2c.o \
acelerometro.o \
giroscopo.o \
magnetometro.o \
correcionPI_Acc.o \
promediador.o \
correccionPI_Mag.o \
actualizar_matriz.o \
renormalizar.o \
conversor_angulos.o \
pid_pitch.o \
pid_roll.o \
referencias.o \
cambio_magnitud.o \
microPWM.o \
miniCosmeApp

lib:
	make $(LIB_PATH)


i2c.o: i2c.h i2c.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c i2c.c -I$(RUNTIME_PATH) -lm

acelerometro.o: acelerometro.h acelerometro.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c acelerometro.c -I$(RUNTIME_PATH) -lm

giroscopo.o: giroscopo.h giroscopo.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
```

```
	$(CC) -c giroscopo.c -I$(RUNTIME_PATH) -lm

magnetometro.o: magnetometro.h magnetometro.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c magnetometro.c -I$(RUNTIME_PATH) -lm

correcionPI_Acc.o: correcionPI_Acc.h correcionPI_Acc.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c correcionPI_Acc.c -I$(RUNTIME_PATH) -lm

promediador.o: promediador.h promediador.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c promediador.c -I$(RUNTIME_PATH) -lm

correccionPI_Mag.o: correccionPI_Mag.h correccionPI_Mag.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c correccionPI_Mag.c -I$(RUNTIME_PATH) -lm

actualizar_matriz.o: actualizar_matriz.h actualizar_matriz.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c actualizar_matriz.c -I$(RUNTIME_PATH) -lm

renormalizar.o: renormalizar.h renormalizar.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c renormalizar.c -I$(RUNTIME_PATH) -lm
```

```
conversor_angulos.o: conversor_angulos.h conversor_angulos.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c conversor_angulos.c -I$(RUNTIME_PATH) -lm

pid_pitch.o: pid_pitch.h pid_pitch.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c pid_pitch.c -I$(RUNTIME_PATH) -lm

pid_roll.o: pid_roll.h pid_roll.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c pid_roll.c -I$(RUNTIME_PATH) -lm

referencias.o: referencias.h referencias.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c referencias.c -I$(RUNTIME_PATH) -lm

cambio_magnitud.o: cambio_magnitud.h cambio_magnitud.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c cambio_magnitud.c -I$(RUNTIME_PATH) -lm

microPWM.o: microPWM.h microPWM.c \
$(RUNTIME_PATH)utiles.h $(RUNTIME_PATH)utiles.c
$(RUNTIME_PATH)cosme.h \
$(RUNTIME_PATH)runtime.h $(RUNTIME_PATH)runtime.c \
$(RUNTIME_PATH)runtime_esp.h $(RUNTIME_PATH)runtime_esp.c
$(RUNTIME_PATH)raspi.h $(RUNTIME_PATH)raspi.c
	$(CC) -c microPWM.c -I$(RUNTIME_PATH) -lm


miniCosmeApp: miniCosmeApp.c \
```

```
$(RUNTIME_PATH)cosme.h $(RUNTIME_PATH)utiles.o
$(RUNTIME_PATH)listas.o $(RUNTIME_PATH)servidorNombres.o \
$(RUNTIME_PATH)pasarela.o $(RUNTIME_PATH)runtime_esp.o
$(RUNTIME_PATH)raspi.o $(RUNTIME_PATH)runtime.o \
i2c.o \
acelerometro.o \
giroscopo.o \
magnetometro.o \
correcionPI_Acc.o \
promediador.o \
correccionPI_Mag.o \
actualizar_matriz.o \
renormalizar.o \
conversor_angulos.o \
pid_pitch.o \
pid_roll.o \
referencias.o \
cambio_magnitud.o \
microPWM.o
	$(CC) -o miniCosmeApp miniCosmeApp.c -lrt -lpthread -lm -
I$(RUNTIME_PATH) -I$(LIB_PATH) \
	$(RUNTIME_PATH)utiles.o $(RUNTIME_PATH)listas.o
$(RUNTIME_PATH)servidorNombres.o \
	$(RUNTIME_PATH)pasarela.o $(RUNTIME_PATH)runtime_esp.o
$(RUNTIME_PATH)raspi.o \
	$(RUNTIME_PATH)runtime.o \
i2c.o \
acelerometro.o \
giroscopo.o \
magnetometro.o \
correcionPI_Acc.o \
promediador.o \
correccionPI_Mag.o \
actualizar_matriz.o \
renormalizar.o \
conversor_angulos.o \
pid_pitch.o \
pid_roll.o \
referencias.o \
cambio_magnitud.o \
microPWM.o \
```