MMA9553L Intelligent Pedometer Platform Software Reference Manual

Devices Supported:

MMA9553L

Document Number: MMA9553LSWRM
Rev. 2.2, 6/2015

Energy Efficient Solutions by Freescale™

freescale™

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MMA9553LSWRM
Rev. 2.2

freescale™
Sensors

Contents

Chapter 1 About This Document

Chapter 2 Pedometer Application

Chapter 1 About This Document

1.1 Overview

1.1.1 Purpose

This reference manual describes the features, architecture, usage examples, and programming model of the MMA9553L intelligent pedometer platform.

1.1.2 Audience

This document is primarily for system architects and software application developers who are using or considering use of the MMA9553L platform in a system.

1.2 Terms and acronyms

AFE	Analog Front End
APP_ID	Application Identifier
application table	Publishes the location of the pedometer's top-level functions and the sizes of the external data structures to the Scheduler
API	Application Programming Interface
BSL	Base Stride Length
CC	Command Complete
CI	Command Interpreter
CMD	Command
COCO	Conversion Complete
consumers	Persons who use devices developed by original equipment manufacturers who incorporate Freescale technology. (See "users.")
CSR	ColdFire Configuration Status Register
DFC	Data Format Code
DM	Background Debug Module
DTAP	Double tap (n.)
FIFO	First In First Out, a data storage and retrieval method
FOPT	Flash Options register
GPIO	General-Purpose Input/Output, a microcontroller pin that can be programmed by software
hash	A deterministic, cryptographic function that converts an arbitrary block of data into a fixed-size bit string—the (cryptographic) hash value—such that an accidental or intentional change to the data will change that hash value
HG	High g

IFR	Flash Information Block, a partition of flash memory reserved for Freescale use
JTAG	Joint Test Action Group (JTAG), the common name for the IEEE 1149.1 standard *Standard Test Access Port and Boundary-Scan Architecture*, for test-access ports
legacy mode	The lower mailbox registers continue to operate in the command/response mode and the upper registers operate in the Quick-Read mode. The data in the Quick-Read registers is automatically updated, so a read-request command is not required before reading the data from the upper mailboxes. Complete information can be found in the MMA955xL Software Reference Manual.
LG	Low g
LL	Landscape left
LR	Landscape right
MAC	Multiply-accumulate unit
MBOX	Mailbox
MCU	Microcontroller
MTIMOV	Module Timer Overflow Module
PC	Program Counter
PD	Portrait down
PDB	Program Delay Block
PL	Portrait/Landscape
POR	Power-on Reset
Postcondition	Conditions that hold after the actions performed by the module/function.
Precondition	Conditions that must hold for a module/function to execute.
PU	Portrait up
SFD	Start Frame Digital
Shared secret	Encrypted data known only to the parties involved in a secure communication. This data can include a password, a pass phrase, a big number, or an array of randomly chosen bytes.
SSP	Supervisor Stack Pointer
TPM	Timer Program Module
users	Developers who incorporate Freescale technology into their devices. (See “consumers.”)
VBR	Vector Base Register, a register in the ColdFire memory map that controls the location of the exception vector table

1.3 Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles also are italicized.
0x0	Prefix of 0x to denote a hexadecimal number
0b	Suffix of b to denote a binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base-address register.
nibble	A four-bit data unit
byte	An eight-bit data unit
word	A 16-bit data unit
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a “do not care.”
n	Used to express an undefined numerical value.
~	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Field concatenation operator
$\overline{\text{OVERLINE}}$	Indicates that a signal is active-low.

1.4 Register figure conventions

This document uses the following conventions for the register reset values:

—	The bit is undefined at reset.
u	The bit is unaffected by reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

Read / Write	0 /	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 0.
Read / Write	1 /	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 1.
Read / Write	FIELDNAME	Indicates a read/write bit.
Read / Write	FIELDNAME /	Indicates a read-only bit field in a memory-mapped register.
Read / Write	/ FIELDNAME	Indicates a write-only bit field in a memory-mapped register.
Read / Write	FIELDNAME / w1c	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
Read / Write	0 / FIELDNAME	Indicates a self-clearing bit.

1.5 References

1. For a list of related reference manuals, application notes, and other documents, visit the *Documentation* tab of the MMA955xL product page on freescale.com.
2. IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std. 1149.1™-2001 (R2008)
3. I^2C-Bus Specification Version 2.1, January 2000, Philips Semiconductors
4. I^2C-Bus Specification and User Manual, NXP Semiconductors Document UM10204, Rev. 03 - 19 June 2007
5. *ColdFire Family Programmer's Reference Manual (CFPRM)* Rev. 3, 03/2005, Freescale Semiconductor, Inc.
6. Wikipedia entry for "Semaphore": http://en.wikipedia.org/wiki/Semaphore_%28programming%29
7. *ITU-T V.41 Recommendation: Code-Independent Error Control System,* available at http://www.itu.int/rec/T-REC-V.41-198811-I/en.
8. *ITU-T X.25 Recommendation: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit,* available at http://www.itu.int/rec/T-REC-X.25-198811-S/en.
9. *ITU-T T.30 Recommendation: Procedures for document facsimile transmission in the general switched telephone network,* available at http://www.itu.int/rec/T-REC-T.30-200509-I/en.
10. *Motion and Freefall Detection Using the MMA8450Q (AN3917),* available at http://www.freescale.com/files/sensors/doc/app_note/AN3917.pdf.

Chapter 2 Pedometer Application

Application ID	0x15
Default speed	30 Hz
Configuration registers	Start on page 17.
Status registers	Start on page 22.

2.1 Background and overview

The MMA9553L device augments the MMA9550L with calculations for step-counting, speed, distance, activity monitoring, and calorie-counting, as well as autonomous sleep functionality to minimize current consumption. As a member of the MMA955xL family, the MMA9553L utilizes a common command interface to access status and configuration data in addition to a common task scheduler to execute Freescale-provided tasks and supplemental, user tasks.

2.2 Functional description

The pedometer is implemented as an application compatible with the MMA955xL Scheduler and Command Interpreter. This enables the pedometer to leverage the existing system services of MMA955xL to execute callback functions in response to accelerometer data becoming available at a selected rate.

In addition to the Scheduler and Interpreter, the platform takes advantage of an established slave-port command interpreter, GPIO mapping, and quick-read mailbox mapping.

The following figure illustrates the hardware and software components and interactions in the MMA9553L platform.

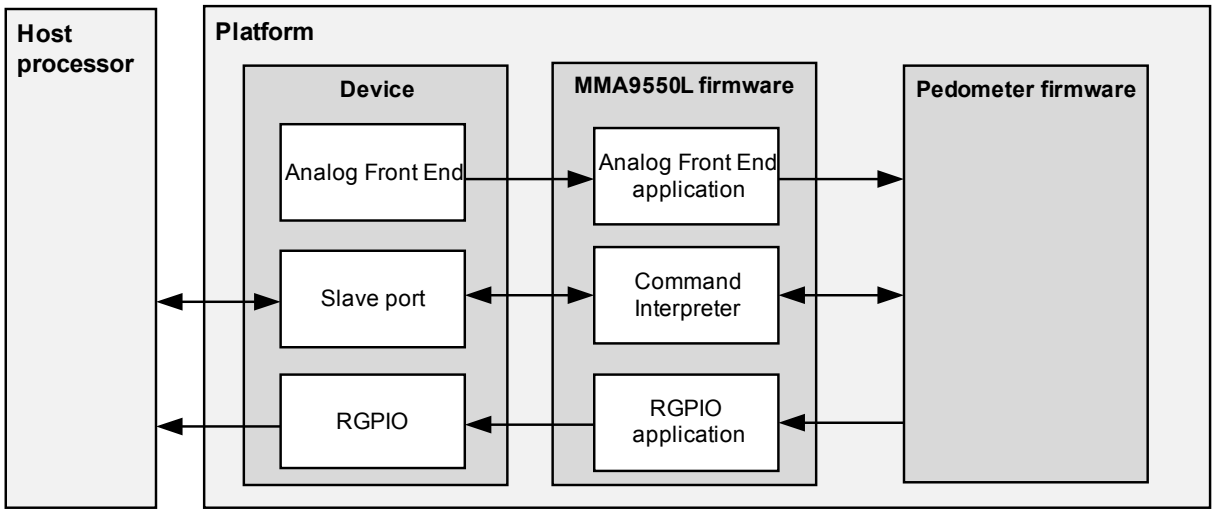

Figure 2-1. Pedometer Data Flow Diagram

The pedometer consists of an application table and up to four top-level callback functions, as required for all MMA955xL applications. The application table is placed at the beginning of a 512-byte flash page and publishes to the Scheduler the location of the pedometer's top-level functions and the sizes of the external data structures.

The top-level functions for the MMA9553L platform are:

- **`pedometer_init()`**: Initialization function that executes exactly once after reset. It is used to configure an application's task-scheduling attributes and request dynamic RAM allocation.
- **`pedometer_reset()`**: This function can be executed multiple times and is used to reset the pedometer to its default state without requiring a hardware reset. Resets an application's internal and external variables to their default states.
- **`pedometer_main()`**: The main function executed after every occurrence of a selected event. In the case of the pedometer, this function executes after each new accelerometer sample is acquired and invokes the main pedometer function to process the sample.
- **`pedometer_clear()`**: Clear an application's external status variables, leaving internal variables unchanged. The pedometer does not need this function and therefore places a null pointer in the respective application table entry.

The MMA955xL platform requires user applications to dynamically request RAM for data structures that interface with the slave-port Command Interpreter. These data structures must be organized such that the status variables are immediately followed by configuration variables, with no padding bytes in between. Private variables may follow the configuration variables, if needed.

2.2.1 Step detection

Step detection is based solely on detecting step impact, without taking into consideration the consumer's height, weight, or gender.

The algorithm operates by keeping track of momentary acceleration, defined as:

$$A = \sqrt{X^2 + Y^2 + Z^2}$$

where X, Y, and Z represent a single accelerometer reading, normalized by dividing by 1g.

The values of A are accumulated over a fixed period of time (0.19 seconds). At every reading, the average A for that period is calculated and saved. The algorithm detects steps by analyzing the spread of the accumulated average A values. The spread is the difference between minimum and maximum of the calculated values.

For a step to be reported, the spread in the buffer of average values of A has to exceed a fixed threshold (0.13 g) and stay above that threshold for at least the fixed value of 0.07 seconds. If the spread falls below the threshold sooner than 0.07 seconds, the motion is ignored.

The STEPCNT variable contains the number of steps detected since the last reset. That count is updated every time a step is detected.

2.2.2 Distance estimation

The distance covered by an individual step is calculated using Base Stride Length (BSL), the estimated "normal" stride length for this consumer.

The BSL is calculated as follows:

$$BSL = Height\ (centimeters) \times GenderFactor \times 1.1$$

where Height is the consumer's height and GenderFactor is 0.415 for males or 0.413 for females

For more information about Height, see "Height/Weight register" on page 20. For more information on GenderFactor, see "Filter register" on page 20.

If no consumer information is provided, BSL is set to 0, and the resulting distance values are 0. If there is a need to update the BSL without consumer information, the configuration structure can be used to set a fixed stride length.

The overall distance is calculated as the sum of estimated stride lengths for all steps detected since the last reset. The stride length for a particular step is calculated as follows:

$$Stride = BSL \times StepRateFactor$$

The StepRateFactor values are shown in the following table:

Table 2-1. StepRateFactor calculation

Step rate, S (steps/sec)	StepRateFactor
$S < 1.6$ (very slow)	0.88
$1.6 \leq S < 1.8$ (slow)	0.95
$1.8 \leq S < 2.35$ (normal)	1.00
$2.35 \leq S < 2.8$ (fast)	1.30
$S \geq 2.8$ (very fast)	2.30

The DISTANCE variable contains the value of overall distance and is updated every time a step is detected.

2.2.3 Speed calculation

Speed is calculated over a sliding time window as:

$$Speed = \frac{Distance\ (meters)}{Time}$$

where Distance is the total distance covered by all steps detected within the time window. Time is the length of the window and can be configured by the SPDPRD variable.

For more information, see:

- Speed: "Speed register" on page 24
- Distance: "Distance register" on page 24
- Time: "Filter register" on page 20
- SPDPRD variable: "Speed Period/Step Coalesce register" on page 21

The SPEED variable contains the current speed value and is updated every time a step is detected or once a second if there are no steps. If there are no steps, the speed may not necessarily fall to zero even if the

activity level falls to rest. The activity level is reset to rest if there are no steps for a certain amount of time. The speed calculation does not include a similar reset when there are no steps. Therefore, it may conflict with the activity level, which is described in the following section. In this scenario, the user should disregard the speed if the activity is rest.

2.2.4 Activity-level calculation

The activity-level calculation is based on the speed value. The activity-level value is assigned according to the following table.

Table 2-2. Activity-level calculation

Latest speed, S (Km/h)	Activity level
$S \geq 10.5$	Running
$6.5 \leq S < 10.5$	Jogging
$1.0 \leq S < 6.5$	Walking
$S < 1.0$	Rest

The ACTIVITY variable contains the current activity level and is updated every time a step is detected or once a second if there are no steps. Additionally, if no steps are detected for the previous 2.5 seconds, the activity level is reset to Rest.

2.2.5 Calorie-expenditure calculation

The estimated amount of calories burned by a single step is calculated as:

$$\text{Calories} = \frac{\text{MetabolicFactor} \times 0.00029}{\text{StepRate}} \times \text{Weight}$$

where StepRate is calculated as described in "Distance estimation" on page 10, Weight is the consumer's weight in kilograms, and MetabolicFactor is calculated according to the following table:

Table 2-3. MetabolicFactor calculation

Step Rate, S (steps/sec)	MetabolicFactor
$S < 1.6$ (very slow)	2.0
$1.6 \leq S < 1.8$ (slow)	2.5
$1.8 \leq S < 2.35$ (normal)	3.8
$2.35 \leq S < 2.8$ (fast)	8.0
$S \geq 2.8$ (very fast)	12.5

For more information, see:

— StepRate: "Filter register" on page 20

— Weight: "Height/Weight register" on page 20

The CALS variable contains the total amount of calories burned since the last reset. The value is updated every time a step is detected.

2.2.6 Debounce count

The debounce_count() function implements a debounce counter as defined in *Motion and Freefall Detection Using the MMA8450Q* (AN3917), an application note that is accessible from the platform documentation link in "References" on page 7.

If the input condition is satisfied, the count is incremented by one up to the threshold. Otherwise, the count is decremented or cleared depending on the debounce counter mode.

The debounce counter's behavior is shown in Figure 2-2.

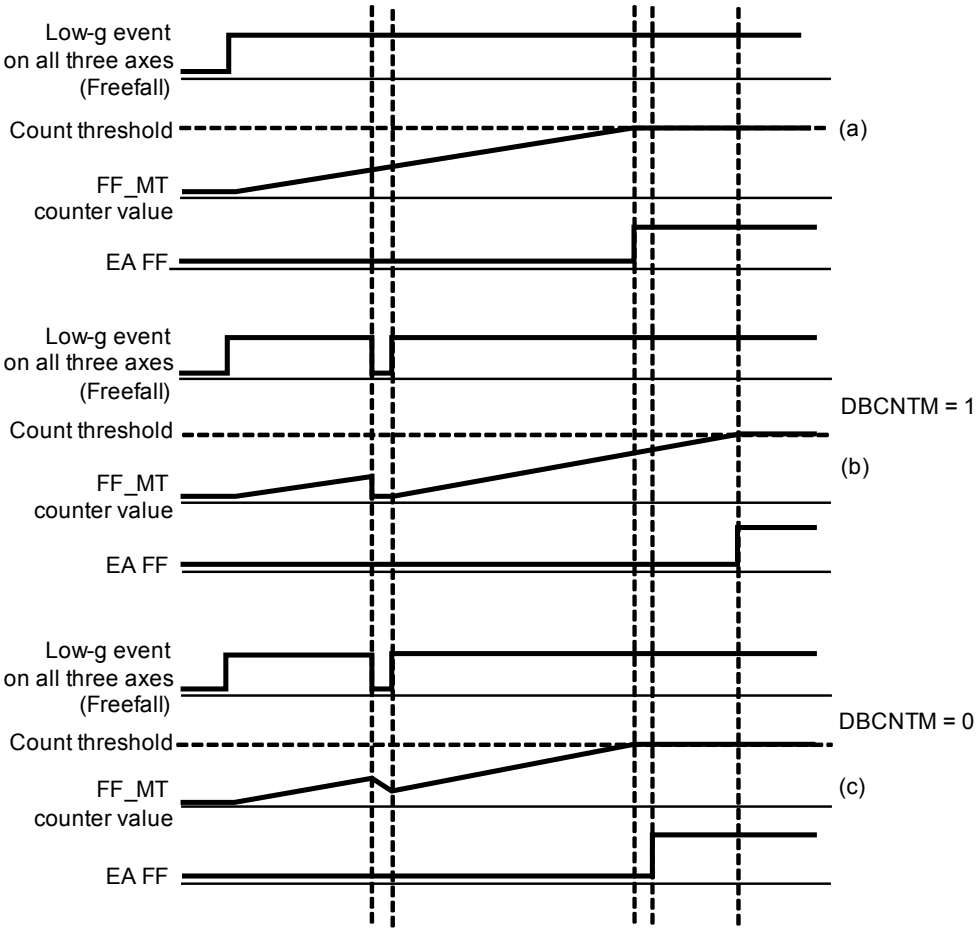

Figure 2-2. Debounce counter behavior

2.2.7 Autonomous suspend

The pedometer uses the acceleration vector magnitude squared, $(X^2 + Y^2 + Z^2)$, to determine if the device is stationary. It is designed to suspend the pedometer conservatively and wake the pedometer aggressively to avoid missing any steps.

The autonomous-suspend function compares the acceleration vector magnitude to the configurable minimum and maximum thresholds (SLEEPMIN and SLEEPMAX) and passes the boolean result to a debounce counter. If the thresholds are satisfied for at least SLEEPTHD samples, the pedometer autonomously suspends. The thresholds are satisfied if the output of the debounce counter is asserted.

If the thresholds are not satisfied for at least SLEEPTHD samples, the pedometer executes normally.

The parameters SLEEPMIN, SLEEPMAX, SLEEPTHD, and SLP_DBCNTM configure the behavior. The SLEEPMAX parameter's reset value disables the autonomous suspend function by default.

For more information, see:

— SLEEPMIN: "Sleep Minimum register" on page 17

— SLEEPMAX: "Sleep Maximum register" on page 18

— SLEEPTHD: "Sleep Count Threshold register" on page 18

— SLP_DBCNTM: "Configuration/Step Length register" on page 19

If custom sleep functionality is desired, a user may disable the pedometer's autonomous-suspend functionality and instead use the MMA955xL's Reset/Suspend/Clear application to enable or disable the pedometer. For an example, see "Enable/disable the Pedometer application" on page 29.

2.2.8 Usage of data types and data structures

2.2.8.1 Basic data types

The following application identifiers are used by the Pedometer application and must not be reused by another application.

```
#define RESERVED_APPID  20 /* APP_ID reserved for pedometer internal functions */
#define PEDOMETER_APPID 21 /* APP_ID for pedometer wrapper interface */
```

2.2.8.2 Structure data types

The pedometer uses a compound data structure for its public-status and configuration variables. The status variables precede the configuration variables as required for all user applications compatible with the MMA955xL platform.

This data structure aligns all elements to their natural boundaries. For example, words are word-aligned. The structure also limits the length such that all status variables or all configuration variables can be read or written with a single, slave-port command.

In Legacy mode, a slave-port command can read up to 16 status bytes or 16 configuration bytes. Normal and Legacy modes are defined in the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual*.

2.3 Memory-maps and register descriptions

The Pedometer Application running in the MMA9553L device has eight configuration registers and six status or data registers. The configuration registers allow the user to customize and control the behavior of the pedometer application. The status registers report back the measured and calculated data.

All status registers are shown as 16 bits wide. They are byte-accessible, but should be read 16 bits (two bytes) or more at a time with a single command, if the user wishes to read them atomically.

Similarly, configuration registers are shown as 16 bits wide but are also byte-accessible. Most fields defined within the configuration registers are 8-bits or less and are byte-aligned, so they can be written one byte at a time if desired. All bytes should be written using a single command if the user wishes to modify them atomically.

For more information on reading /writing the Mailbox registers, please see Chapter 5 of the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955xLSWRM), accessible from "References" on page 7.

2.3.1 Pedometer memory maps

Table 2-4. Configuration registers

Offset address	Register	Access	Reset	Details
0x0	Sleep Minimum register	R/W	0x0000	"Sleep Minimum register" on page 17
0x2	Sleep Maximum register	R/W	0x0000	"Sleep Maximum register" on page 18
0x4	Sleep Count Threshold register	R/W	0x0001	"Sleep Count Threshold register" on page 18
0x6	Config/Step Length register	R/W	0x0000	"Configuration/Step Length register" on page 19
0x8	Height/Weight register	R/W	0xAF50	"Height/Weight register" on page 20
0xA	Filter register	R/W	0x0403	"Filter register" on page 20
0xC	Speed Period register	R/W	0x0501	"Speed Period/Step Coalesce register" on page 21
0xE	Activity Count Threshold register	R/W	0x0000	"Activity Count Threshold register" on page 21

Table 2-5. Status registers

Offset address	Register	Access	Reset	Details
0x0	Status register	R	0x0001	"Status register" on page 22
0x2	Step count register	R	0x0000	"Step Count register" on page 23
0x4	Distance register	R	0x0000	"Distance register" on page 24
0x6	Speed register	R	0x0000	"Speed register" on page 24
0x8	Calories register	R	0x0000	"Calories register" on page 25
0xA	Sleep Count register	R	0x0000	"Sleep Count register" on page 25

2.3.2 Pedometer application configuration example

To write all the pedometer configuration registers, send the following command packet from the host to the device mailboxes. The most significant byte of a register (MSB) is written in the lower numbered mailbox.

NOTE

The question marks represent placeholders for the application specific values. Please replace the question marks with your own values for the application.

MB0 = 0x15: Set the Pedometer Application Identifier (0x15)

MB1 = 0x20: Set the Command: Write Config command, with zero offset (0x20)

MB2 = 0x00: Set the Offset to point to the first configuration register

MB3 = 0x10: Set the Count field to declare writing 16 bytes

MB4–5 = 0x????: Value for Sleep Min Register

MB6–7 = 0x????: Value for Sleep Max Register

MB8–9 = 0x????: Value for Sleep Count Threshold

MB10–11 = 0x????: Value for Config / Step Length Register

MB12–13 = 0x????: Value for Height / Weight

MB14–15 = 0x????: Set the Filter Register

MB16–17 = 0x????: Set the Speed Period

MB18–19 = 0x????: Set the Activity Count

Bytes to send: 0x15, 0x20, 0x00, 0x10, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??,

0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??, 0x??

To read all the pedometer configuration registers, send the following command packet from the host to the device mailboxes. This can be used as a device identification command, allowing a host to differentiate the MMA9553L from the MMA9550L.

MB0: 0x15 = pedometer application ID

MB1: 0x10 = opcode to read configuration

MB2: 0x00 = offset into pedometer configuration register map

MB3: 0x10 = number of bytes to read

The MMA9553L response will be:

MB0: 0x15 = pedometer application ID

MB1: 0x80 = COCO=1, error code=0

MB2: 0x10 = actual number of bytes read

MB3: 0x10 = requested number of bytes to read

MB4: sleep minimum MSB

MB5: sleep minimum LSB

MB6: sleep maximum MSB

MB7: sleep maximum LSB

MB8: sleep count threshold MSB

MB9: sleep count threshold LSB

MB10: config

MB11: step length

MB12: height

MB13: weight

MB14: filter step

MB15: male, filter time

MB16: step period

MB17: step coalesce

MB18: activity count threshold MSB

MB19: activity count threshold LSB

The MMA9550 response will be:

MB0: 0x15 = pedometer application ID

MB1: 0x84 = COCO=1, error code=4 (MCI _ERROR_PARAM)

MB2: 0x00 = actual number of bytes read

MB3: 0x10 = requested number of bytes to read

2.3.3 Pedometer application read example

To read all the pedometer status registers, send the following command packet from the host to the device mailboxes:

MB0 = 0x15: Set the Pedometer Application Identifier (0x15)

MB1 = 0x30: Set the Command: Read Status command, with zero offset (0x30)

MB2 = 0x00: Set the Offset (0x00) to point to the first status register

MB3 = 0x0C: Set the Count field to (12) to declare reading 12 bytes

Read back the mailboxes, when the COCO (Command Complete) bit is set the status data will be in the mailbox registers.

Bytes to send: 0x15, 0x30, 0x00, 0x0C

The response to this command will be:

MB0: 0x15 = pedometer application ID

MB1: 0x80 = COCO=1, error code=0

MB2: 0x0C = actual number of bytes read

MB3: 0x0C = requested number of bytes to read

MB4: pedometer status register MSB

MB5: pedometer status register LSB

MB6: step count MSB

MB7: step count LSB

MB8: distance MSB

MB9: distance LSB

MB10: speed MSB

MB11: speed LSB

MB12: calories MSB

MB13: calories LSB

MB14: sleep count MSB

MB15: sleep count LSB

2.3.4 Pedometer configuration-register descriptions

2.3.4.1 Sleep Minimum register

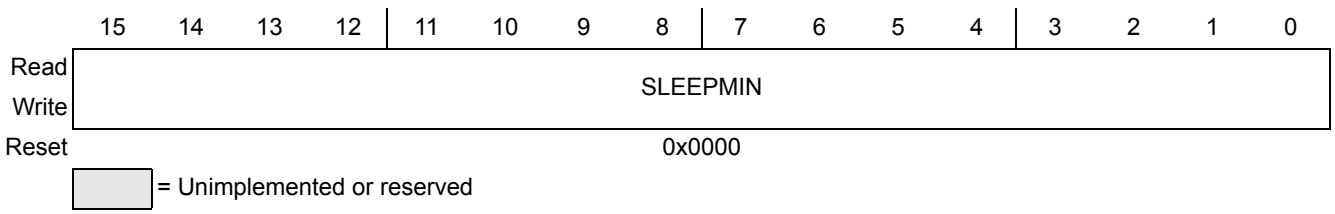

Figure 2-3. Sleep Minimum register

Table 2-6. Sleep Minimum register field descriptions

Bit(s)	Field	Description
15:0	SLEEPMIN	Minimum acceleration vector magnitude for autonomous suspend. The acceleration vector magnitude must be greater than SLEEPMIN and less than SLEEPMAX to satisfy the autonomous suspend condition(1). At rest, the acceleration vector magnitude measures approximately 1g. Therefore, SLEEPMIN and SLEEPMAX are expected to be set to values near 1g (4096 at 0.244 mg/LSB resolution). Valid range: 0x0000:0xFFFF (uint16). Units: 0.244 mg/LSB

1. This condition must be satisfied for SLEEPTHD samples for the pedometer to autonomously suspend. See Table 2-8 on page 19.

2.3.4.2 Sleep Maximum register

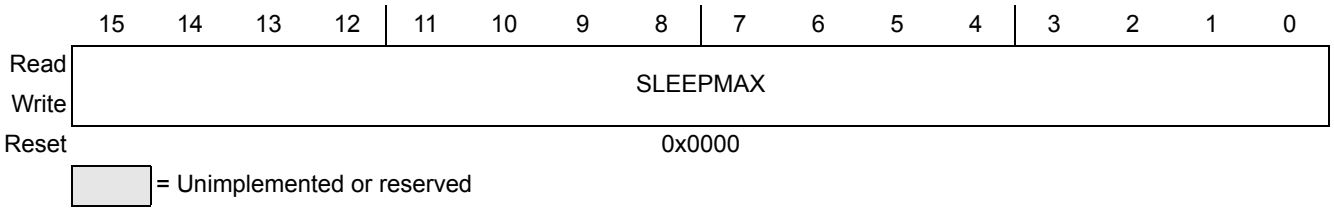

Figure 2-4. Sleep Maximum register

Table 2-7. Sleep Maximum register field descriptions

Bit(s)	Field	Description
15:0	SLEEPMAX	Maximum acceleration vector magnitude for autonomous suspend. The acceleration vector magnitude must be greater than SLEEPMIN and less than SLEEPMAX to satisfy the autonomous suspend condition(1). At rest, the acceleration vector magnitude measures approximately 1g. Therefore, SLEEPMIN and SLEEPMAX are expected to be set to values near 1g (4096 at 0.244 mg/LSB resolution). Set to SLEEPMAX 0 to disable autonomous suspend. Valid range: 0x0000:0xFFFF (uint16). Units: 0.244 mg/LSB

1. This condition must be satisfied for SLEEPTHD samples for the pedometer to autonomously suspend. See Table 2-8 on page 19.

2.3.4.3 Sleep Count Threshold register

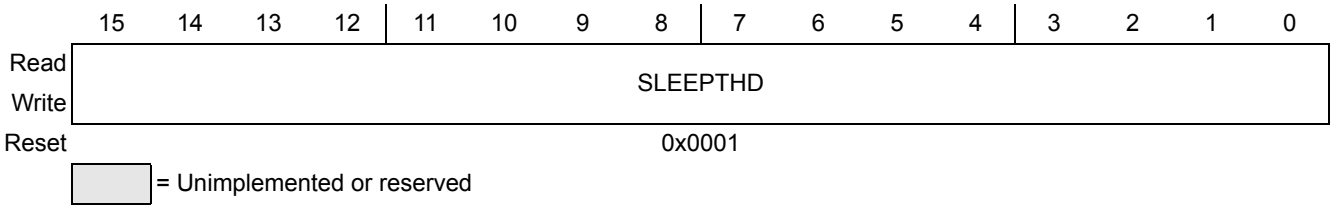

Figure 2-5. Sleep Count Threshold register

Table 2-8. Sleep Count Threshold register field descriptions

Bit(s)	Field	Description
15:0	SLEEPTHD	Autonomous suspend debounce count threshold. The autonomous suspend condition(1) must be satisfied for SLEEPTHD samples for the pedometer to autonomously suspend. Valid range: 0x0000:0xFFFF (uint16).

1. The acceleration vector magnitude must be greater than SLEEPMIN and less than SLEEPMAX to satisfy this condition. For more information on SLEEPMIN and SLEEPMAX, see "Sleep Minimum register" and "Sleep Maximum register".

2.3.4.4 Configuration/Step Length register

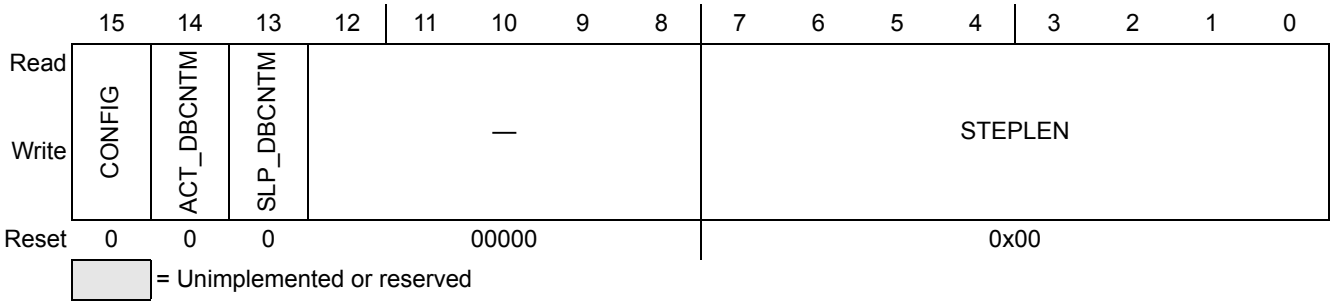

Figure 2-6. Configuration/Step Length register

Table 2-9. Configuration/Step Length register field descriptions

Bit(s)	Field	Description
15	CONFIG	(Re)initializes the pedometer with current configuration values. Modifications to other pedometer configuration registers will not take effect until this bit is set. It is automatically cleared after the (re)initialization completes. 0 Do not (re)initialize the pedometer 1 (Re)initialize the pedometer with current configuration values
14	ACT_DBCNTM	Activity debounce counter mode. 0 Decrement the count when the activity level changes 1 Clear the count when the activity level changes
13	SLP_DBCNTM	Autonomous suspend debounce counter mode. 0 Decrement the count when the device is in motion 1 Clear the count when the device is in motion
12:8	—	Reserved. Set to 0.
7:0	STEPLEN	Step length in centimeters. Set to 0 to automatically estimate the consumer's step length based on gender and height. Valid range: 0x00:0xFF (uint8)

2.3.4.5 Height/Weight register

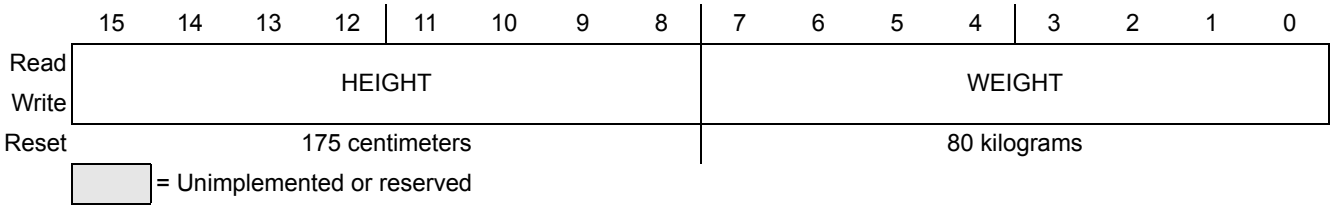

Figure 2-7. Height/Weight register

Table 2-10. Height/Weight register field descriptions

Bit(s)	Field	Description
15:8	HEIGHT	Height in centimeters. Used to estimate step length, if STEPLEN = 0. Valid range: 0x00:0xFF (uint8) Default = 175.
7:0	WEIGHT	Weight in kilograms. Used to estimate step length, if STEPLEN = 0. Valid range: 0x00:0xFF (uint8) Default = 80.

2.3.4.6 Filter register

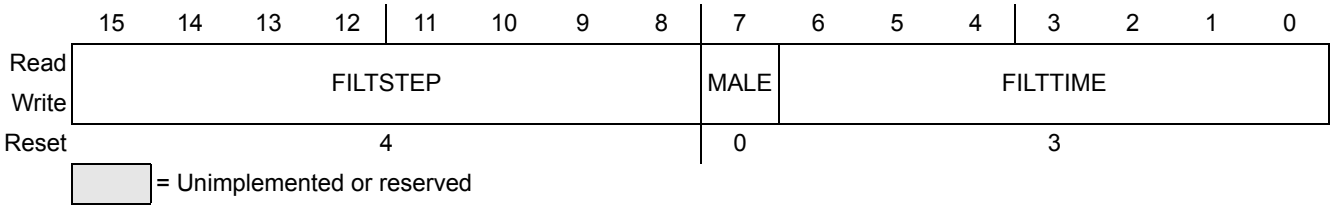

Figure 2-8. Filter register

Table 2-11. Filter register field descriptions

Bit(s)	Field	Description
15:8	FILTSTEP	Number of steps that must occur within FILTTIME for the pedometer to decide the consumer is making steps. Set to 0 to disable step filtering. If the value specified is greater than 6, then 6 will be used. Valid range: 0x00:0x06 (uint8). Default = 4.
7	MALE	Gender 0 Female 1 Male
6:0	FILTTIME	Number of seconds in which filter steps must occur. Set to 0 to disable step filtering. Valid range: 0x00:0x7F (uint8) Default = 3.

2.3.4.7 Speed Period/Step Coalesce register

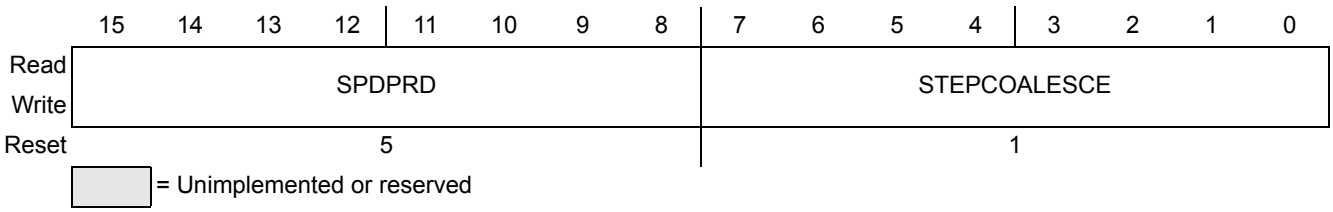

Figure 2-9. Speed Period/Step Coalesce register

Table 2-12. Speed Period/Step Coalesce register field descriptions

Bit(s)	Field	Description
15:8	SPDPRD	Number of seconds in which to compute speed. If set to a value greater than 5, then 5 will be used. Valid range: 0x02:0x05. **Warning:** Do not set SPDPRD to 0 or 1 as this may cause undesirable behavior.
7:0	STEPCOALESCE	Number of steps to coalesce before asserting STEPCHG. 0 Disables STEPCHG. 1 Asserts STEPCHG after every step. The default. Valid range: 0x00:0xFF (uint8).

2.3.4.8 Activity Count Threshold register

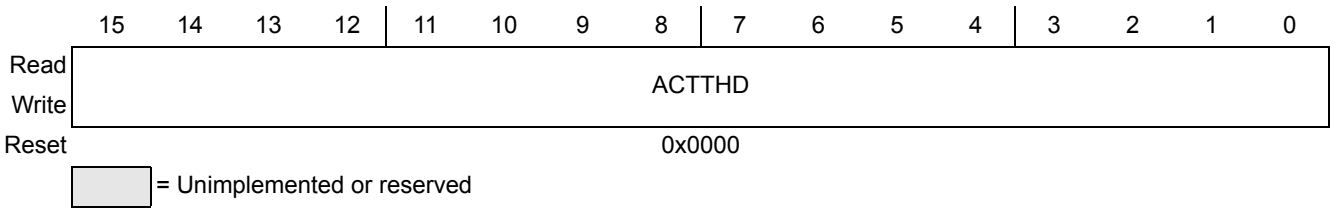

Figure 2-10. Activity Count Threshold register

Table 2-13. Activity Count Threshold register field descriptions

Bit(s)	Field	Description
15:0	ACTTHD	Activity debounce count threshold. The internal activity level must be stable for ACTTHD samples before ACTIVITY is updated. Valid range: 0x0000:0xFFFF (uint16) 0 The activity debouncer is effectively bypassed.(1) 1 The current internal activity level must equal the previous internal activity level in order to update ACTIVITY.

1. For more information on the activity debouncer, see Table 2-9 on page 19.

2.3.5 Pedometer status-register descriptions

2.3.5.1 Status register

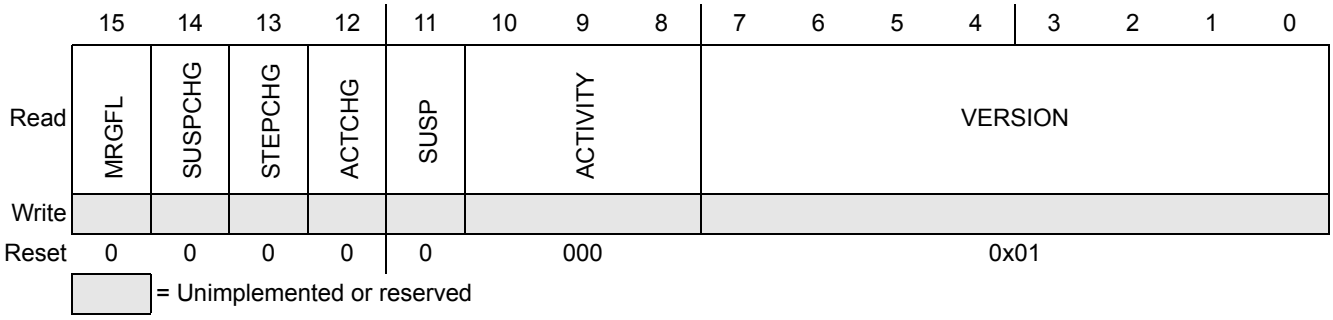

Figure 2-11. Status register

Table 2-14. Status register field descriptions

Bit(s)	Field	Description
15	MRGFL	Merged status change flags. This bit is the logical OR of the SUSPCHG, STEPCHG, and ACTCHG flags. It can be routed to a pin to enable a single, merged-output interrupt using the MMA955xL GPIO application(1). The host can trigger an interrupt on rising edges to receive notification when at least one of the status change flags is asserted. The host is responsible for resolving the source if desired. That can be done by comparing the STEPCNT to a previous value to determine that a STEPCNT change caused the MRGFL assertion. 0 None of the status change flags are asserted 1 At least one of the status change flags (SUSPCHG, STEPCHG, ACTCHG) is asserted
14	SUSPCHG	Indicates a change in the SUSP bit. This bit is transient and only asserts during frames in which the SUSP bit changes from the previous frame. A frame is one 30-Hz period. This bit can be routed to a pin to enable output interrupts using the MMA955xL GPIO application(1). The host can trigger an interrupt on rising edges to receive notification when the pedometer suspends or resumes. 0 No change in the SUSP bit since the last pedometer call. 1 The SUSP bit changed since the last pedometer call.
13	STEPCHG	Indicates a change in STEPCNT by STEPCOALSCE steps. This bit is transient and only asserts during frames in which STEPCNT changed from the previous frame. A frame is one 30-Hz period. This bit can be routed to a pin to enable output interrupts using the MMA955xL GPIO application(1). The host can trigger an interrupt on rising edges to receive notification after every step. 0 The step count has not been incremented by STEPCOALESCE steps since the last STEPCHG assertion or the pedometer was last initialized. 1 The step count has been incremented by STEPCOALESCE steps since the last STEPCHG assertion or the pedometer was last initialized.
12	ACTCHG	Indicates a change in activity level. This bit is transient and only asserts during frames in which ACTIVITY changed from the previous frame. A frame is one 30-Hz period. This bit can be routed to a pin to enable output interrupts using the MMA955xL GPIO application(1). The host can trigger an interrupt on rising edges to receive notification when the activity level is changed. 0 No change in activity level since last pedometer call 1 New activity level since last pedometer call

Table 2-14. Status register field descriptions (Continued)

Bit(s)	Field	Description
11	SUSP	Indicates whether the pedometer is active or has been autonomously suspended. This bit can be routed to a pin to enable output interrupts using the MMA955xL GPIO application(1). The host can trigger an interrupt on rising edges to receive notification when the pedometer is autonomously suspended and trigger an interrupt on falling edges to receive notification when the pedometer resumes. 0 Pedometer is active 1 Pedometer is suspended
10:8	ACTIVITY	Activity level: 000 Unknown 001 Rest 010 Walking 011 Jogging 100 Running 101-111 Reserved
7:0	VERSION	Version number of the pedometer application, incremented by one for each new release. 0 R1.32 1 R1.33

1. For more information on the GPIO application, see chapter 5 of the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955xLSWRM), accessible from "References" on page 7.

2.3.5.2 Step Count register

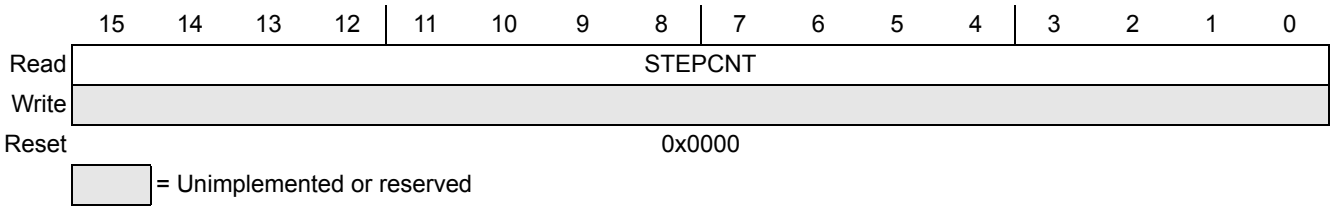

Figure 2-12. Step Count register

Table 2-15. Step Count register field descriptions

Bit(s)	Field	Description
15:0	STEPCNT	The total step count since the pedometer was last reset. Valid range: 0x0000:0xFFFF (uint16)

2.3.5.3 Distance register

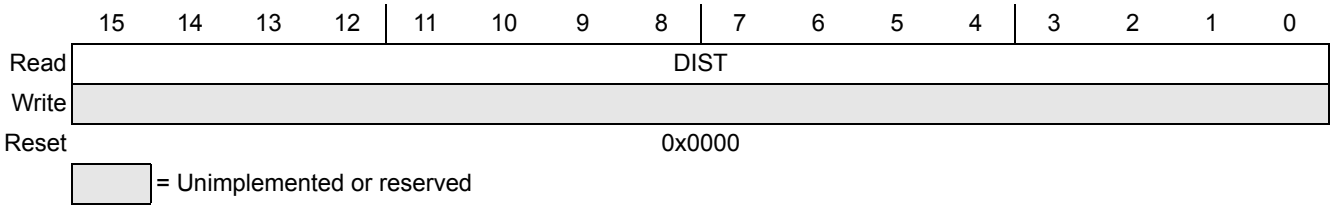

Figure 2-13. Distance register

Table 2-16. Distance register field descriptions

Bit(s)	Field	Description
15:0	DIST	The total distance in meters since the pedometer was last reset. Valid range: 0x0000:0xFFFF (uint16).

2.3.5.4 Speed register

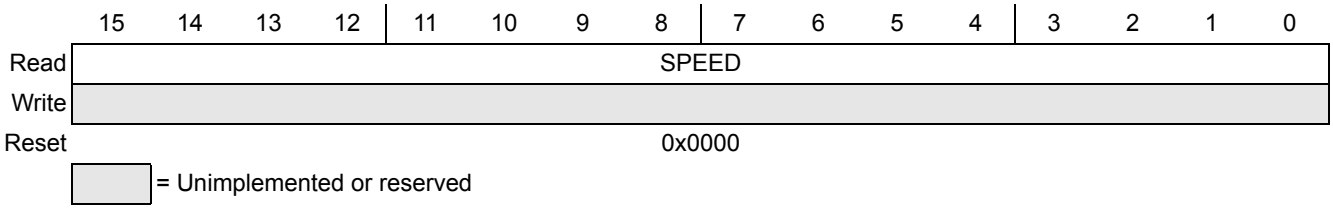

Figure 2-14. Speed register

Table 2-17. Speed register field descriptions

Bit(s)	Field	Description
15:0	SPEED	Average speed in meters per hour over SPDPRD(1). Valid range: 0x0000:0xFFFF (uint16)

1. For information on SPDPRD, see Table 2-12 on page 21.

NOTE

If there are no steps, the speed may not necessarily fall to zero even if the activity level falls to rest. See 2.2.3, “Speed calculation” on page 10 for more information.

2.3.5.5 Calories register

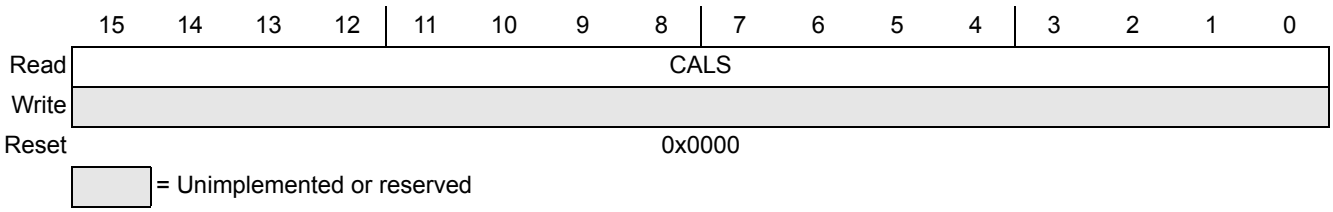

Figure 2-15. Calories register

Table 2-18. Calories register field descriptions

Bit(s)	Field	Description
15:0	CALS	Total calorie count since the pedometer was last reset. Valid range: 0x0000:0xFFFF (uint16)

2.3.5.6 Sleep Count register

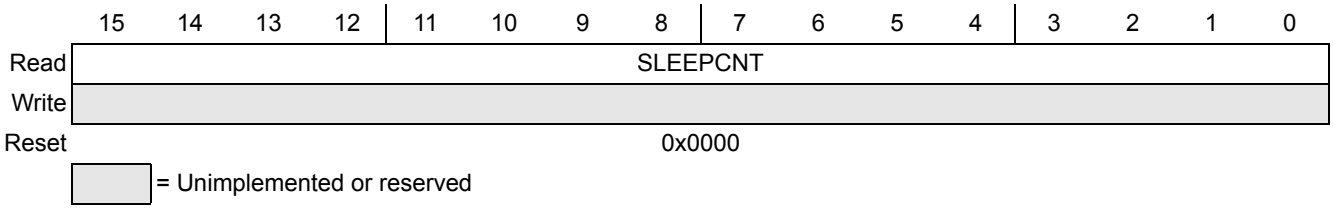

Figure 2-16. Sleep Count register

Table 2-19. Sleep Count register field descriptions

Bit(s)	Field	Description
15:0	SLEEPCNT	Current value of the autonomous suspend debounce counter. Valid range: 0x0000:0xFFFF (uint16)

2.4 Error reporting

The error reporting philosophy is to check only for errors that could cause system instability. Error conditions that do not cause system instability are not checked or reported.

The MMA955xL Command Interpreter includes error-checking and error-reporting when accessing the pedometer's status and configuration variables. For example, the Command Interpreter does not allow a slave-port command to access more bytes than are present in an application’s public data structure.

For more details, see the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955xLSWRM), accessible from “References” on page 7.

2.5 Functions

2.5.1 pedometer_init()

This function initializes the pedometer once after a hardware reset. It performs the following operations:

1. Requests dynamic RAM allocation for the pedometer's status, configuration, and private variables.
2. Configures the scheduling attributes for the Pedometer application.
   — Priority = TASK30HZ.
   This instructs the scheduler to execute **`pedometer_main()`** at 30 Hz.
   — Activity = ALWAYS.
   This is the *scheduler-defined sleep/wake* activity level, not the *pedometer* activity level. This setting instructs the scheduler to run **`pedometer_main()`** regardless of the MMA9553L's sleep/wake state. The pedometer performs its own computations to determine its sleep condition.
3. Configures the Analog Front End (AFE) application to sample the accelerometer at 30 Hz in 4g mode.
   All other AFE application characteristics are left at their default values.
4. Configure the AFE, GPIO, MBOX, FIFO, EVENT_QUEUE, STATUS_REG, CI, RST_CLR_SUSP, LONG_SHORT_INT applications to execute in the 30 Hz task.

Precondition

None.

Postcondition

User RAM is allocated to the pedometer, the AFE is configured to sample accelerometer data in 4g mode, and the scheduler is configured to execute the Pedometer application at 30 Hz.

Prototype

```
void pedometer_init(void);
```

Parameters

None.

Return values

None.

2.5.2 pedometer_reset()

This function resets all pedometer variables to their default states. It may be called more than once after a hardware reset.

Precondition

None.

Postcondition

All pedometer variables are set to their default states.

Prototype

```
void pedometer_reset(void);
```

Parameters

None.

Return values

None.

2.5.3 pedometer_main()

This function executes after each new accelerometer sample (30 Hz) is completed and computes the step count, speed, and other status pedometer variables. This function performs the following operations:

1. Uses the new accelerometer sample to determine if the device is stationary.
2. Reinitializes, if necessary.

   A reinitialization occurs when the host processor—or another MMA9553L user application—sets the CONFIG bit, indicating that new configuration parameters (such as height or weight) need to be passed to the pedometer.
3. Does one of the following:
   a) If the device is not autonomously suspended: Processes the new accelerometer sample (XYZ) to compute step count, speed, and other status variables.
   b) If the device *is* autonomously suspended: Skips this step.
4. Updates the pedometer status variables.

   This occurs within a critical section to ensure that all status variables are updated atomically.

By default, the MMA955xL firmware samples the accelerometer and executes all system tasks at 30 Hz. System tasks include activities such as AFE-filtering, GPIO update, and quick-read update.

The AFE application provides several output taps with different sample rates and bandwidths. Of these, the pedometer uses the stage 0 AFE output, nominally sampled at 488 Hz with a 100 Hz bandwidth. The MMA9553L overrides the default behavior of the MMA955xL such that the stage 0 AFE output is sampled at 30.5 Hz with 6.25 Hz bandwidth. This output is always scaled to 8g resolution, regardless of the hardware AFE configuration (2g/4g/8g).

For the pedometer, the g-mode setting only affects the saturation level and noise, not the scaling. Although it violates Nyquist at 30 Hz, this AFE output is used to simplify the pedometer implementation. This is the most-intuitive option, if a user chooses to reconfigure the hardware AFE sampling rate.

Precondition

The **`pedometer_init()`** function must be executed before calling this function.

Postcondition

The pedometer's status variables are updated.

Prototype

```
void pedometer_main(void);
```

Parameters

None.

Return values

None.

2.5.4 pedometer_clear()

This function is not implemented.

Precondition

None.

Postcondition

N/A

Prototype

```
void pedometer_clear(void);
```

Parameters

None.

Return values

None.

2.6 Sample operations

This section provides sample slave-port commands to read and modify application variables.

NOTE

All commands are in the hexadecimal format. For more details on the command protocol and format, see Chapter 4 of the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955xLSWRM), accessible from "References" on page 7.

2.6.1 Read status variables

This command reads all 12 bytes of the status variables from the Pedometer application (APP_ID=0x15), starting at byte offset 0.

```
15 30 00 0C
```

2.6.2 Read configuration variables

This command reads all 16 bytes of the configuration variables from the Pedometer application (APP_ID=0x15), starting at byte offset 0.

```
15 10 00 10
```

2.6.3 Write configuration variables

This command writes all 16 bytes of the configuration variables to the Pedometer application (APP_ID=0x15), starting at byte offset 0.

```
15 20 00 10 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
```

2.6.4 Reset configuration variables to their defaults

This command writes one byte of configuration variable to the Reset/Suspend/Clear application (APP_ID=0x17) starting at byte offset 1.

This causes the scheduler to invoke the pedometer's reset function, **`pedometer_reset()`**.

```
17 20 01 01 20
```

2.6.5 Enable/disable the Pedometer application

This command writes one byte of configuration variable to the Reset/Suspend/Clear application (APP_ID=0x17) starting at byte offset 5.

This causes the scheduler to suspend the Pedometer application, even if the device is not stationary.

```
17 20 05 01 20 (disable)
17 20 05 01 00 (enable)
```

2.6.6 Configure the AFE range

This command writes one byte of the configuration variable to the AFE application (APP_ID=0x06), starting at byte offset 0. Since the pedometer uses a normalized acceleration output provided by the AFE application, the g mode only affects the saturation level and noise.

Regardless of what g mode the AFE is configured to, for or by other applications, the resolution seen by the Pedometer application is always 8g.

```
06 20 00 01 40 (2g mode)
06 20 00 01 80 (4g mode)
06 20 00 01 00 (8g mode)
```

For more information on configuring the AFE range, see the "AFE configuration registers" section of the MMA955xL *Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955xLSWRM). That document is accessible from the product documentation link in "References" on page 7.

2.6.7 Configure output interrupt: Activity change on GPIO6

This command writes two bytes of the configuration variable to the GPIO application (APP_ID=0x03), starting at byte offset 0.

This causes the GPIO application to copy the pedometer's ACTCHG status bit to GPIO6 after every new accelerometer sample. The host processor can trigger an interrupt on rising edges of this pin to receive notification when the activity level changes.

```
03 20 00 02 15 04
```

2.6.8 Configure output interrupt: Step change on GPIO7

This command writes two bytes of the configuration variable to the GPIO application (APP_ID=0x03), starting at byte offset 2.

This causes the GPIO application to copy the pedometer's STEPCHG status bit to GPIO7 after every new accelerometer sample. The host processor can trigger an interrupt on rising edges of this pin to receive notification when the step count changes.

```
03 20 02 02 15 05
```

2.6.9 Configure output interrupt: Suspend change on GPIO8

This command writes two bytes of the configuration variable to the GPIO application (APP_ID=0x03) starting at byte offset 4.

This causes the GPIO application to copy the pedometer's SUSPCHG status bit to GPIO8 after every new accelerometer sample. The host processor can trigger an interrupt on rising edges of this pin to receive notification when the pedometer autonomously suspends or resumes.

```
03 20 04 02 15 06
```

2.6.10 Configure output interrupt: Merged flags on GPIO6

This command writes two bytes of the configuration variable to the GPIO application (APP_ID=0x03) starting at byte offset 0.

This causes the GPIO application to copy the pedometer's MRGFL status bit to GPIO6 after every new accelerometer sample. The host processor can trigger an interrupt on rising edges of this pin to receive notification the activity-level, step-count, or suspend-state changes.

```
03 20 00 02 15 07
```

2.6.11 Configure output interrupt: Every 10 steps on GPIO7

The first command writes one byte of configuration variable to the Pedometer application (APP_ID = 0x15), starting at byte offset 13. This causes the pedometer to coalesce steps and assert STEPCHG once every 10 steps.

The second command writes two bytes of configuration variables to the GPIO application (APP_ID = 0x03), starting at byte offset 2. This causes the GPIO application to copy the pedometer's STEPCHG status bit to GPIO7 after every new accelerometer sample.

```
15 20 0D 01 0A
03 20 02 02 15 05
```

2.6.12 Wake up from Deep Sleep (Stop No Clock mode)

By default, the MMA955xL initializes all applications after a hardware reset and then falls into Deep Sleep mode where all system clocks are stopped. A slave-port command can be used to wake up the clocks, enable the accelerometer, and execute the pedometer.

This command writes one byte of configuration variable to the sleep/wake application (APP_ID=0x12) starting at byte offset 6.

```
12 20 06 01 00
```

Revision History

Revision	Date	Description
1.0	04/2013	• Initial Release of the Software Reference Manual
2.0	07/2013	• Updates for release R1.33
2.1	10/2013	• Section 2.2.3 Speed calculation, added disclaimer re: no steps • Table 2-5 Status registers, Offset address 0x0, Reset was 0x0000 • Table 2-12 Speed Period/Step Coalesce register field descriptions, Bit 15:8: Deleted “If set to 0, then 1 will be used.” Valid range was 0x01:0x05 Added warning • Speed register field descriptions table, added note
2.2	06/2015	• Section 2.2.2 StepRateFactor value changed to 1.30 • Section 2.2.5 Calorie-expenditure equation corrected to match the calculation.