PmodACL2 Library Reference Manual

®

Revision: 10/14/2014

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Overview

The pmodACL2 library provides an interface to an ADXL362 3- axis accelerometer. The library initializes the accelerometer and both reads real time data or can support a FIFO buffer system to get 100 kHz spaced results.

Library Operation

Library Interface

The header file ACL2.h defines all the used register addresses and initialization bytes. The file also holds two classes. The first class is myQueue which is the base for the FIFO buffer using a set integer array. The second class is the ACL2 class. This class is the main interface with the ADXL362 and uses the myQueue class to implement the FIFO buffer for all three axis's with the ability to have a buffer for the temperature data. To instantiate an ACL2 object, include the ACL2 library and instantiate an ACL2 object.

ACL2 Initialization

The ACL2 module is initialized by calling the function begin(). This function sets up the FIFO buffers then calls an initialization function that chooses the following settings.

- Set the freefall detection threshold to 600 mg (g = earth gravities)
- Set the freefall detection time to 30 ms
- Enables the inactivity detect
- Sets the inactivity interrupt to interrupt pin 1
- Sets sensor range to ± 8 g
- Enables measurement

After the accelerometer is initialized, the device will immediately start spitting out real time data at a frequency of 100 kHz.

Used Registers and Their Functions

These are the main registers used in the ACL2 libraries. A more in depth view of these register functions can be found in the ADXL362 datasheet found here. Any of these registers can be accessed by a user by using the readRegister and writeRegister functions.

Address name	Address	Function
PART_ID	0x02	Displays the ACL part ID
X_DATA	0x08	8 bit x-axis data (low power)
Y_DATA	0x09	8 bit y-axis data (low power)
Z_DATA	0x0A	8 bit z-axis data (low power)
STATUS	0x0B	Status Register
FIFO_ENTRIES_L	0x0C	LSBs of the 12 bit value of entries in the FIFO buffer
FIFO_ENTRIES_H	0x0D	MSBs of the 12 bit value of entries in the FIFO buffer
XDATA_L	0x0E	LSBs of the 12 bit x-axis accelerometer data
XDATA_H	0x0F	MSBs of the 12 bit x-axis accelerometer data
YDATA_L	0x10	LSBs of the 12 bit y-axis accelerometer data
YDATA_H	0x11	MSBs of the 12 bit y-axis accelerometer data
ZDATA_L	0x12	LSBs of the 12 bit z-axis accelerometer data
ZDATA_H	0x13	MSBs of the 12 bit z-axis accelerometer data
TEMP_L	0x14	LSBs of the 12 bit temperature data
TEMP_H	0x15	MSBs of the 12 bit temperature data
SOFT_RESET	0x1F	Resets registers to default values
THRESH_INACT_L	0x23	LSBs of the 12 bit threshold inactivity
THRESH_INACT_H	0x24	MSBs of the 12 bit threshold inactivity
FIFO_CONTROL	0x28	Various controlling bits for the FIFO buffer
FIFO_SAMPLES	0x29	Amount of samples before FIFO watermark INT1 fires
INTMAP1	0x2A	Sets which interrupt gets mapped to the INT1 pin
INTMAP2	0x2B	Sets which interrupt gets mapped to the INT2 pin
FILTER_CTL	0x2C	Various controlling bits for the ACL
POWER_CTL	0x2D	Various controlling bits for the ACL

ACL2 Library Functions

myQueue Class

The myQueue class uses a set length integer array. This is possible because the FIFO buffer on the ADXL362 will work on a fill dump basis. Meaning the buffer will fill up until the user needs the data, then the buffer will be completely emptied. This means a set array length can be used and no data is dynamically allocated. Below is a filled myQueue with 510 items

Item0	Item1	Item2	Item3	Item4	……………………		Item510	Item511	Item512
Int0	Int1	Int2	Int3	Int4	………	……….	Int510	-1	-1
↑								↑	↑
Front_ptr and head_ptr								tail_ptr	back_ptr

Public Functions

myQueue()

Parameters:

None

Return Value:

None

Constructor for queue sets the pointers to initial positions and clears the queue by calling resetQueue()

empty()

Parameters:

None

Return Value:

None

Sets all members of the queue to -1 and resets the pointers to the beginning. At a top level it deletes all queue members. Below is the queue after empty.

Item0	Item1	Item2	Item3	Item4	……………………		Item510	Item511	Item512
-1	-1	-1	-1	-1	………	……….	-1	-1	-1
↑									↑
Front_ptr/head_ptr/tail_ptr									back_ptr

size()

Parameters:

None

Return Value:

Int tail_ptr	The number of items currently in the queue

size() returns the tail_ptr. This works out to be the size of the queue in this implementation.

front()

Parameters:

None

Return Value:

Int dataQueue[head_ptr - 1]	The value at the front of the queue

This function returns the first value in the queue without destroying it.

back()

Parameters:

None

Return Value:

Int dataQueue[tail_ptr - 1]	The value at the back of the queue

back() returns the value one less than the tail_ptr. tail_ptr points to the array member after the last valid data. dataQueue[tail_ptr – 1] is the last int in the queue.

push_back(int value)

Parameters:

Int value	The value to push onto the queue

Return Value:

None

Push_back(int value) accepts a value then pushes that value onto the back of the queue

int pop_front()

Parameters:

None

Return Value:

int result	data coming off the queue

pop_front() reads the first member in the queue then returns it and moves the head_ptr to move to the next item in line.

resetQueue()

Parameters:

None

Return Value:

None

Sets pointer values including front and back pointers to start over queue then empties it by calling empty()

getQueue(int* outQueue)

Parameters:

Int* outQueue	int array to copy dataQueue data to

Return Value:

None

Accepts an integer array pointer then empties the dataQueue into the array for use by the user

ACL2 Class

Public Functions

ACL2()

Parameters:

None

Return Value:

None

Constructor for class ACL2.

begin(int CS**)**

Parameters:

int CS	chip select pin for SPI communications

Return Value:

None

This function starts the SPI communication, stores the desired chip select, and sets the accelerometer to the suggested zero values. If you know at startup, the accelerometer will be at rest, a better implementation would be to run setZero() instead of storing these default values into the zero variables. The function then calls reset().

init()

Parameters:

None

Return Value:

None

This function sets the ACL2 up for basic use applying a sensitivity of +- 8g (256 per 1g) and sets up default settings on activity and drop detection by writing to various registers.

int getX()

Parameters:

None

Return Value:

int x	The value of acceleration in the X direction found by using getData()

This function calls getData with the XDATA_H and XDATA_L registers and return the X value for acceleration.

int getY()

Parameters:

None

Return Value:

Int y	The value of acceleration in the Y direction found by using getData()

This function calls getData with the YDATA_H and YDATA_L registers and return the Y value for acceleration.

int getZ()

Parameters:

None

Return Value:

int z	The value of acceleration in the Z direction found by using getData()

This function calls getData with the ZDATA_H and ZDATA_L registers and return the Z value for acceleration.

int getTemp()

Parameters:

None

Return Value:

int temp	The value of temperature on the chip.

This function calls getData with the TEMP_H and TEMP_L registers and return the temp value for on board temperature.

Uint8_t getStatus()

Parameters:

None

Return Value:

uInt8_t status	The 8 bits that occupy the STATUS register

This function reads the status register and returns the 8 bit value.

Uint8_t getRange()

Parameters:

None

Return Value:

uInt8_t range	either 2, 4, or 8g range class item

This function returns the uint8_t range class member which describes the current range of measurement.

Uint8_t readRegister(uint8_t thisRegister)

Parameters:

Uint8_t thisRegister	register to read a byte from

Return Value:

uInt8_t inByte	byte read from register

This function returns the byte located in thisRegister. The function handles all the SPI protocol.

writeRegister(uint8_t thisRegister, uint8_t thisValue)

Parameters:

Uint8_t thisRegister	register to write to
Uint8_t thisValue	byte to write in this register

Return Value:

None

This function writes a byte to a register given by thisRegister's address. The function handles all the SPI protocol.

reset()

Parameters:

None

Return Value:

None

This function writes the byte 'R' to the reset register to initiate a soft reset. Then calls init to set the sensor up for measurement again

updateRange()

Parameters:

None

Return Value:

None

This function reads the filter control register and stores the sensitivity range into the private variable range

setRange(int newRange)

Parameters:

newRange Must be an int value of 2, 4, or 8;

Return Value:

None

This function reads the filter control register (FILTER_CTL) and stores the sensitivity range into the private variable range

setZero()

Parameters:

None

Return Value:

None

This function sets the zeroing variables so that the ACL puts out x= 0, y = 0, z = 1000. The function takes an average over 100 samples since the data can be sporadic.

int getFIFOentries()

Parameters:

None

Return Value:

Int entries Entries in the FIFO buffer

This function reads the FIFO entries registers(FIFO_ENTRIES_H, FIFO_ENTRIES_L) using the getData function to read how many FIFO entries are in the ADXL362 queue to be read out.

fillFIFO()

Parameters:

None

Return Value:

None

This function transfers FIFO data from the ADXL362 into the myQueue elements of the class. The getData functionality had to be recreated since the SPI chip select signal has to stay low during the whole transfer. After this function is called, the xFIFO, yFIFO and zFIFO elements will be populated.

int getData(uint8_t reg1, uint8_t reg2)

Parameters:

reg1 The first register to read from. The high data value which contains the 3 MSBs

reg2 The second register to read from. The low data value which contains the 8 LSB's

Return Value:

None

This function reads data from a register couple and does the masking and shifting to create an int value. This function is used often throughout the program. Changing it might cause many issues.

Private Functions

uint16_t twosToBin(uint16_t input)

Parameters:

input ` an 11 bit twos complement value to be converted to a binary number

Return Value:

return Returns a 16 bit unsigned integer with the positive value of the negative twos compliment

This function converts a negative twos compliment value and performs a bitwise flip and subtracts one to return a positive int value. **This does not return a negative number.**

char getDIR(uint16_t value)

Parameters:

value FIFO raw data to parse direction from

Return Value:

char result axis that the FIFO data represents (x/y/z)

This function takes the raw FIFO data and analyses the 2 MSBs to determine the axis the data represents.

Class members

myQueue xFIFO	myQueue designated for FIFO data on the x-axis
myQueue yFIFO	myQueue designated for FIFO data on the y-axis
myQueue zFIFO	myQueue designated for FIFO data on the z-axis
myQueue tempFIFO	myQueue designated for FIFO data regarding temperature
int chipSelect	The pin being used as the SPI chipSelect
uint8_t range	The current range of the accelerometer (±2g/±4g/±8g)
int xZero	Calibration setting for the x-axis
int yZero	Calibration setting for the y-axis
int zZero	Calibration setting for the z-axis