三轴加速度传感器在跌倒检测中的应用

作者：Ning Jia

前言

人们在跌倒后会面临双重危险。显而易见的是跌倒本身可能对人体产生伤害；另外，如果跌倒后不能得到及时的救助，可能会使结果更加恶化。例如，许多老年人由于其身体比较虚弱，自理能力和自我保护能力下降，常常会发生意外跌倒，如果得不到及时的救助，这种跌倒可能会导致非常严重的后果。有资料显示，很多严重的后果并不是由于跌倒直接造成的，而是由于跌倒后，未得到及时的处理和救护。当出现跌倒情况时，如果能够及时地通知到救助人员，将会大大地减轻由于跌倒而造成的危害。

不仅是对老人，在很多其他情况下，跌倒的报警也是非常有帮助的，尤其是从比较高的地方跌倒下来的时候。比如人们在登山，建筑，擦窗户，刷油漆和修理屋顶的时候。

这促使人们越来越热衷于对跌倒检测以及跌倒预报仪器的研制。近年来，随着 MEMS 加速度传感器技术的发展，使得设计基于三轴加速度传感器的跌倒检测器成为可能。这种跌倒检测器的基本原理是通过测量佩戴该仪器的个体在运动过程中的三个正交方向的加速度变化来感知其身体姿态的变化，并通过算法分析判断该个体是否发生跌倒情况。当个体发生跌倒时，仪器能够配合 GPS 模块以及无线发送模块对这一情况进行定位及报警，以便获得相应的救助。而跌倒检测器的核心部分就是判断跌倒情况是否发生的检测原理及算法。

ADXL345[1] 是ADI公司的一款 3 轴、数字输出的加速度传感器。本文将在研究跌倒检测原理的基础上，提出一种基于ADXL345的新型跌倒检测解决方案。

iMEMS 加速度传感器 ADXL345

iMEMS半导体技术把微型机械结构与电子电路集成在同一颗芯片上。iMEMS加速度传感器就是利用这种技术，实现对单轴、双轴甚至三轴加速度进行测量并产生模拟或数字输出的传感器。根据不同的应用，加速度传感器的测量范围从几g到几十g不等。数字输出的加速度传感器还会集成多种中断模式。这些特性可以为用户提供更加方便灵活的解决方案。

ADXL345 是 ADI 公司最近推出的基于 iMEMS 技术的 3 轴、数字输出加速度传感器。ADXL345 具有+/-2，+/-4，+/-8，+/-16g 可变的测量范围；最高 13bit 分辨率；固定的 4mg/LSB 灵敏度；3mm*5mm*1mm 超小封装；40-145uA 超低功耗；标准的 I2C 或 SPI 数字接口；32 级 FIFO 存储；以及内部多种运动状态检测和灵活的中断方式等特性。所有这些特性，使得 ADXL345 有助于大大简化跌倒检测算法，使其成为一款非常适合用于跌倒检测器应用的加速度传感器。本文给出的跌倒检测解决方案，完全基于ADXL345内部的运动状态检测功能和中断功能，甚至不需要对加速度的具体数值进行实时读取和复杂的计算操作，可以使算法的复杂度降至最低。

中断系统

图1给出了ADXL345的系统框图及管脚定义。

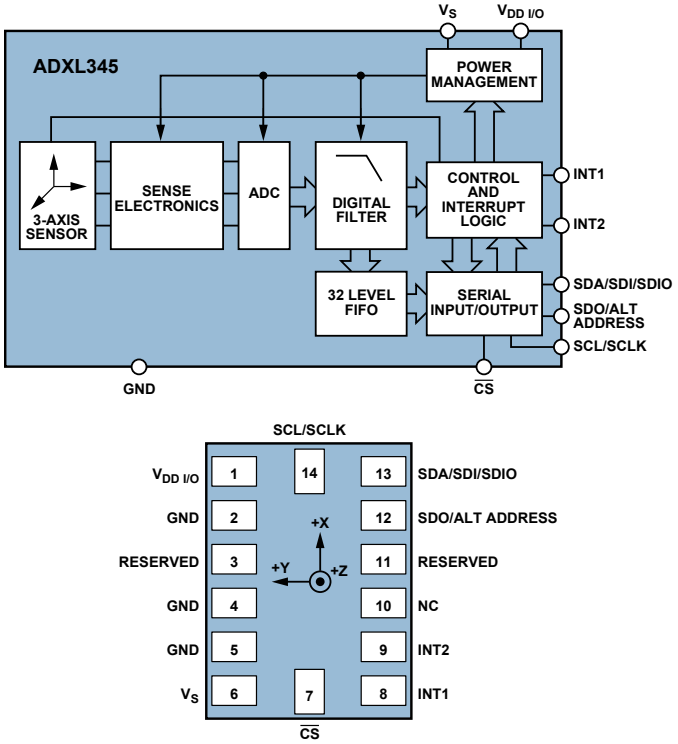


图 1. ADXL345 系统框图及管脚定义

ADXL345具有两个可编程的中断管脚：Int1和Int2。以及Data_Ready、Single_Tap、Double_Tap、Activity、Inactivity、Free_Fall、Watermark、Overrun，共计8个中断源。每个中断源可以独立地使能或禁用，还可以灵活地选择是否映射到Int1或Int2中断管脚。所有的功能都可以同时使用，只是某些功能可能需要共用中断管脚。中断功能通过INT_ENABLE寄存器的相应位来选择使能或禁用，通过INT_MAP寄存器的相应位来选择映射到Int1管脚或Int2管脚。

中断功能的具体定义如下：

1、Data_Ready - 当有新的数据产生时，Data_Ready中断置位；当没有新的数据时，Data_Ready中断清除。

2、Single_Tap - 当加速度值超过一定门限（THRESH_TAP）并且持续时间小于一定时间范围（DUR）的时候，Single_Tap中断置位。

3、Double_Tap - 当第一次Single_Tap事件发生后，在一定时间（LATENT）之后，并在一定时间（WINDOW）之内，又发生第二次Single_Tap事件时，Double _Tap中断置位。

图2给出了有效的Single_Tap中断和Double _Tap中断的示意图。

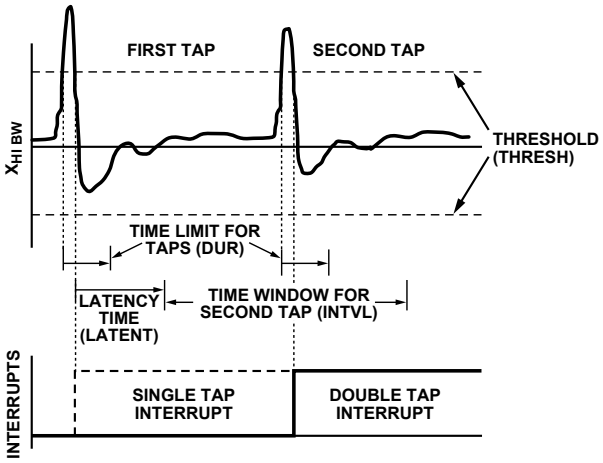


图 2. Single_Tap 和 Double _Tap 中断示意

4、Activity - 当加速度值超过一定门限（THRESH_ACT）时，Activity中断置位。

5、Inactivity - 当加速度值低于一定门限（THRESH_INACT）并且持续超过一定时间（TIME_INACT）时，Inactivity中断置位。TIME_INACT可以设定的最长时间为255s。

需要指出的是，对于Activity和Inactivity中断，用户可以针对X、Y、Z轴来分别进行使能或禁用。比如，可以只使能X轴的Activity中断，而禁用Y轴和Z轴的Activity中断。

另外，对于Activity和Inactivity中断，用户还可以自由选择DC coupled工作方式或者AC coupled工作方式。其区别在于，DC coupled工作方式下，每个采样点的加速度值将直接与门限（THRESH_ACT或THRESH_INACT）进行比较，来判断是否发生中断；而AC coupled工作方式下，新的采样点将以之前的某个采样点为参考，用两个采样点的差值与门限（THRESH_ACT或THRESH_INACT）进行比较，来判断是否发生中断。AC coupled工作方式下的Activity检测，是选择检测开始时的那一个采样点作为参考，以后每个采样点的加速度值都与参考点进行比较。如果它们的差值超过门限（THRESH_ACT），则Activity中断置位。AC coupled工作方式下的Inctivity检测，同样要选择一个参考点。如果新采样点与参考点的加速度差值超过门限（THRESH_INACT），参考点会被该采样点更新。如果新采样点与参考点的加速度差值小于门限（THRESH_INACT），并且持续超过一定时间（TIME_INACT），则Inctivity置位。

6、Free_Fall - 当加速度值低于一定门限（THRESH_FF）并且持续超过一定时间（TIME_FF）时，Free_Fall中断置位。与Inactivity中断的区别在于，Free_Fall中断主要用于对自由落体运动的检测。因此， X、Y、Z轴总是同时被使能或禁用；其时间设定也比Inactivity中断中要小很多，TIME_FF可以设定的最大值为1.28s；而且Free_Fall中断只能是DC coupled工作方式。

7、Watermark - 当FIFO里所存的采样点超过一定点数（SAMPLES）时，Watermark中断置位。当FIFO里的采样点被读取，使得其中保存的采样点数小于该数值（SAMPLES）时，Watermark中断自动清除。

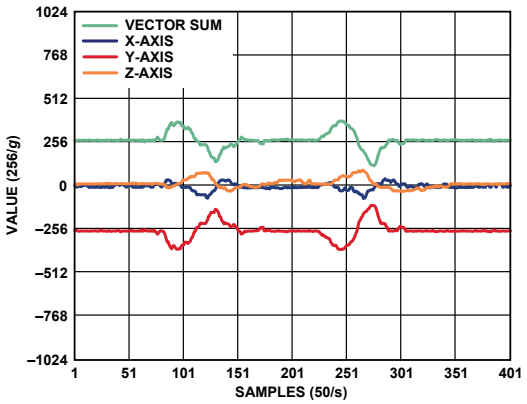
需要指出的是，ADXL345的FIFO最多可以存储32个采样点（X、Y、Z三轴数值），且具有Bypass模式、普通FIFO模式、Stream模式和Trigger模式，一共4种工作模式。FIFO功能也是ADXL345的一个重要且十分有用的功能。但是本文后面给出的解决方案中，并没有使用到FIFO功能，所以，在此不做详细介绍。

8、Overrun - 当有新采样点更新了未被读取得前次采样点时，Overrun中断置位。Overrun功能与FIFO的工作模式有关，当FIFO工作在Bypass模式下，如果有新采样点更新了DATAX、DATAY和DATAZ寄存器里的数值，则Overrun中断置位。当FIFO工作在其他三种模式下，只有FIFO被存满32点时，Overrun中断才会置位。FIFO里的采样点被读取后，Overrun中断自动清除。

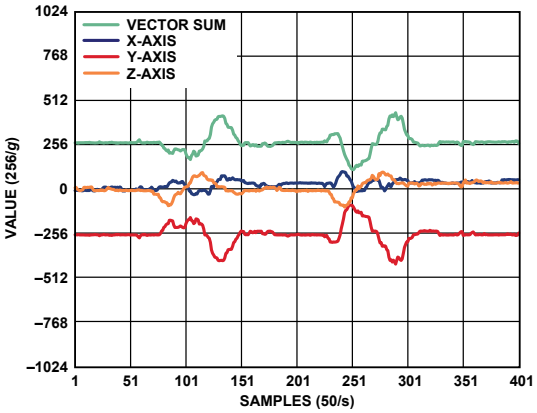
跌倒过程中的加速度变化特征

对跌倒检测原理的研究主要是找到人体在跌倒过程中的加速度变化特征。

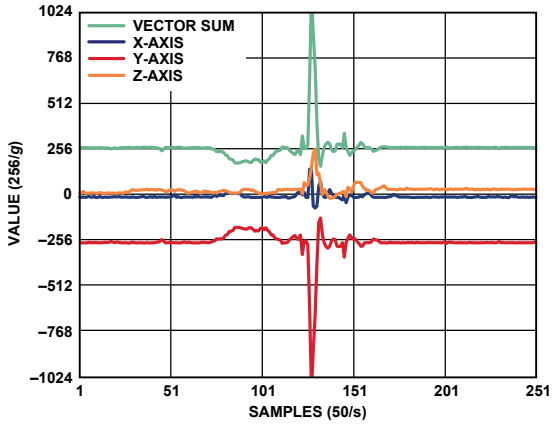
图3给出的是加速度在不同运动过程中的变化曲线，包括(a)步行上楼、(b)步行下楼、(c)坐下、(d)起立。假设跌倒检测器被固定在被测的人体上。其中红色的曲线是Y轴（垂直方向）的加速度曲线，其正常静止状态下应该为-1g；黑色和黄色的曲线分别是X轴（前后方向）和Z轴（左右方向）的加速度曲线，其正常静止状态下应该为0g；绿色的曲线是三轴加速度的矢量和，其正常静止状态下应该为+1g。



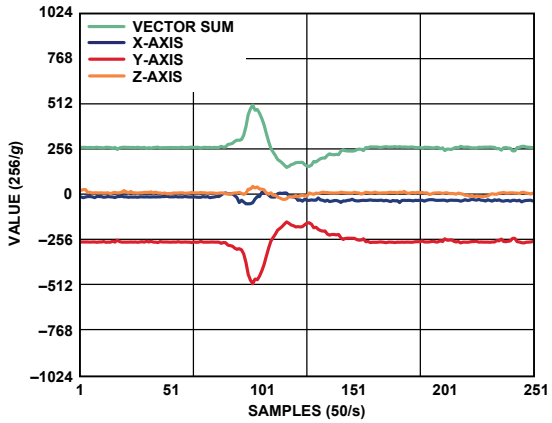
a. 步行上楼



b. 步行下楼



c. 坐下



d. 起立

图 3. 不同运动过程中的加速度变化曲线

由于老年人的运动相对比较慢，所以在普通的步行过程中，加速度变化不会很大。最明显的加速度变化就是在坐下动作中 Y 轴加速度（和加速度矢量和）上有一个超过 3g 的尖峰，这个尖峰是由于身体与椅子接触而产生的。

而跌倒过程中的加速度变化则完全不同。图 4 给出的是意外跌倒过程中的加速度变化曲线。通过图 4 和图 3 的比较，可以发现跌倒过程中的加速度变化有 4 个主要特征，这可以作为跌倒检测的准则。这 4 个特这在图 4 中以红色的方框标注，下面将对其逐一进行详细介绍。

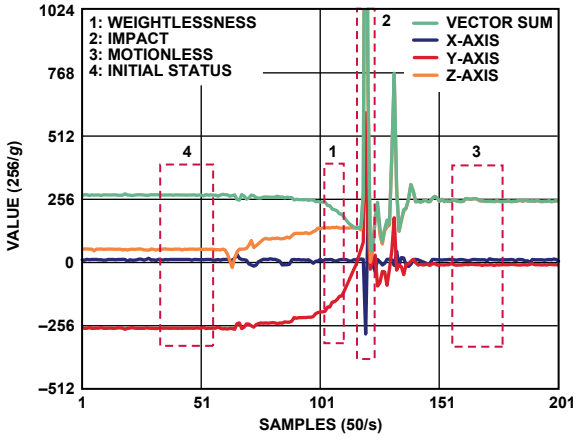


图 6. 自动功率控制回路的功能框图

1. 失重：在跌倒的开始都会发生一定的失重现象。在自由落体的下降过程，这个现象会更加明显，加速度的矢量和会降低到接近0g，持续时间与自由落体的高度有关。对于一般的跌倒，失重现象虽然不会有像自由落体那么明显，但也会发生合加速度小于1g的情况（通常情况下合加速度应大于1g）。因此，这可以作为跌倒状态的第一个判断依据。可以由ADXL345的Free_Fall中断来检测。
2. 撞击：失重之后，人体发生跌倒的时候会与地面或其他物体发生撞击，在加速度曲线中会产生一个很大的冲击。这个冲击可以通过ADXL345的Activity中断来检测。因此，Free_Fall中断之后，紧接着产生Activity中断是跌倒状态的第二个判断依据。
3. 静止：通常，人体在跌倒后，也就是撞击发生之后，不可能马上起来，会有短暂的静止状态（如果人因为跌倒而导致昏迷，甚至可能是较长时间的静止）。表现在加速度曲线上就是会有一段时间的平稳。这可以通过ADXL345的Inactivity中断来检测。因此，Activity中断之后的Inactivity中断是跌倒状态的第三个判断依据。
4. 与初始状态比较：跌倒之后，人体会发生翻转，因此人体的方向会与原先静止站立的姿态（初始状态）不同。这使得跌倒之后的静止状态下的三轴加速度数值与初始状态下的三轴加速度不同（见图4）。假设跌倒检测器固定在被测人体上的某个部位，这样初始状态下的三轴加速度数值可以认为是已知的（本例中，初始状态为：X轴0g，Y轴-1g，Z轴0g）。读取Inactivity中断之后的三轴加速度数据，并与初始状态进行比较。如图4所示，重力加速度方向由Y轴上的-1g变为了Z轴上的1g，这说明人体发生了侧向跌倒。因此，跌倒检测的第四个依据就是跌倒后的静止状态下加速度值与初始状态发生变化，且矢量变化超过一定的门限值（比如0.7g）。

这四个判断依据综合在一起，构成了整个的跌倒检测算法，可以对跌倒状态给出报警。当然，还要注意各个中断之间的时间间隔要在合理的范围之内。比如，除非是从很高的楼顶掉下来，否则 Free_Fall 中断（失重）和 Activity 中断（撞击）之间的时间间隔不会很长。同样，通常情况下， Activity 中断（撞击）和 Inactivity 中断（静止）之间的时间间隔也不会很长。本文接下来会通过一个具体实例给出一组合理的取值。当然，相关中断的检测门限以及时间参数也可以根据需要而灵活设置。

另外，如果跌倒造成了严重的后果，比如，导致了人的昏迷。那么人体会在更常的一段时间内都保持静止。这个状态仍然可以通过 Inactivity 中断来检测。也就是说，如果发现在跌倒之后的很长时间内都保持 Inactivity 状态，可以再次给出一个严重报警。

典型电路连接

ADXL345 和微控制器之间的电路连接非常简单。本文中的测试平台由ADXL345 和微控制器ADuC7026[2] 组成。图 5 给出了ADXL345 和ADuC7026 之间的典型电路连接。ADXL345

的\CS 管脚接高电平，表示 ADXL345 工作在 I2C 模式。SDA 和 SCL 是 I2C 总线的数据线和时钟线，分别连接到 ADuC7026 相应的 I2C 总线管脚。ADuC7026 的一个 GPIO 管脚连接到 ADXL345 的 ALT 管脚，用来选择 ADXL345 的 I2C 地址。ADXL345 的 INT1 管脚连接到 ADuC7026 的 IRQ 输入用来产生中断信号。

其他的单片机或者处理器都可以采用与图 5 类似的电路与 ADXL345 进行连接。ADXL345 还可以工作在 SPI 模式以获得更高的数据传输速率。关于 SPI 工作模式的具体描述，请参考 ADXL345 数据手册。

利用 ADXL345 简化跌倒检测算法

本节将给出以上解决方案的具体算法实现。表 1 中简要说明了每个寄存器的作用以及在本算法中的设置值。对于各个寄存中每一位的具体含义，请参考 ADXL345 的数据手册。

需要指出的是，表 1 给出的设置值中，某些寄存器会给出两个数值，这说明在算法中会切换使用这两个数值，来达到不同的检测目的。算法的流程图如图 6 所示。

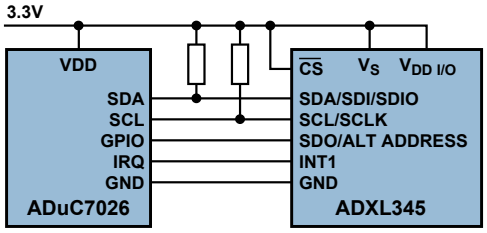


图 5. ADXL345 与微控制器之间的典型电路连接

表 1. ADXL345 寄存器功能说明

地址	寄存器名称	类型	默认值	说明	设置值	设置的功能
0	DEVID	只读	0xE5	器件 ID	只读	
1-1C	Reserved			保留，不要操作	保留	
1D	THRESH_TAP	读/写	0x00	Tap 的门限	不使用	
1E	OFSX	读/写	0x00	X 轴失调	0x06	补偿 X 轴失调，通过初始化校正获得
1F	OFSY	读/写	0x00	Y 轴失调	0xF9	补偿 Y 轴失调，通过初始化校正获得
20	OFSZ	读/写	0x00	Z 轴失调	0xFC	补偿 Z 轴失调，通过初始化校正获得
21	DUR	读/写	0x00	Tap 的持续时间	不使用	
22	LATENT	读/写	0x00	Tap 的延迟时间	不使用	
23	WINDOW	读/写	0x00	Tap 的时间窗	不使用	
24	THRESH_ACT	读/写	0x00	Activity 的门限	0x20/0x08	设置 Activity 的门限为 2g 或 0.5g
25	THRESH_INACT	读/写	0x00	Inactivity 的门限	0x03	设置 Inactivity 的门限为 0.1875g
26	TIME_INACT	读/写	0x00	Inactivity 的时间	0x02/0x0A	设置 Inactivity 的时间为 2s 或 10s
27	ACT_INACT_CTL	读/写	0x00	Activity/Inactivity 使能控制	0x7F/0xFF	使能 X、Y、Z 三轴的 Activity 和 Inactivity 功能，其中 Inactivity 为 AC coupled 模式，Activity 为 DC coupled 或 AC coupled 模式
28	THRESH_FF	读/写	0x00	Free-Fall 的门限	0x0C	设置 Free-Fall 的门限为 0.75g
29	TIME_FF	读/写	0x00	Free-Fall 的时间	0x06	设置 Free-Fall 的时间为 30ms
2A	TAP_AXES	读/写	0x00	Tap/Double Tap 使能控制	不使用	
2B	ACT_TAP_STATUS	只读	0x00	Activity/Tap 中断轴指示	只读	
2C	BW_RATE	读/写	0x0A	采样率和功耗模式控制	0x0A	设置采样率为 100Hz
2D	POWER_CTL	读/写	0x00	工作模式控制	0x00	设置为正常工作模式
2E	INT_ENABLE	读/写	0x00	中断使能控制	0x1C	使能 Activity、Inactivity、Free-Fall 中断
2F	INT_MAP	读/写	0x00	中断影射控制	0x00	所有中断影射到 Int1 管脚
30	INT_SOURCE	只读	0x00	中断源指示	只读	
31	DATA_FORMAT	读/写	0x00	数据格式控制	0x0B	设置为+/-16g 测量范围，13bit 右对齐模式，中断为高电平触发，使用 I2C 数据接口
32	DATAX0	只读	0x00	X 轴数据	只读	
33	DATAX1	只读	0x00		只读	
34	DATAY0	只读	0x00	Y 轴数据	只读	
35	DATAY1	只读	0x00		只读	
36	DATAZ0	只读	0x00	Z 轴数据	只读	
37	DATAZ1	只读	0x00		只读	
38	FIFO_CTL	读/写	0x00	FIFO 控制	不使用	
39	FIFO_STATUS	只读	0x00	FIFO 状态	只读	

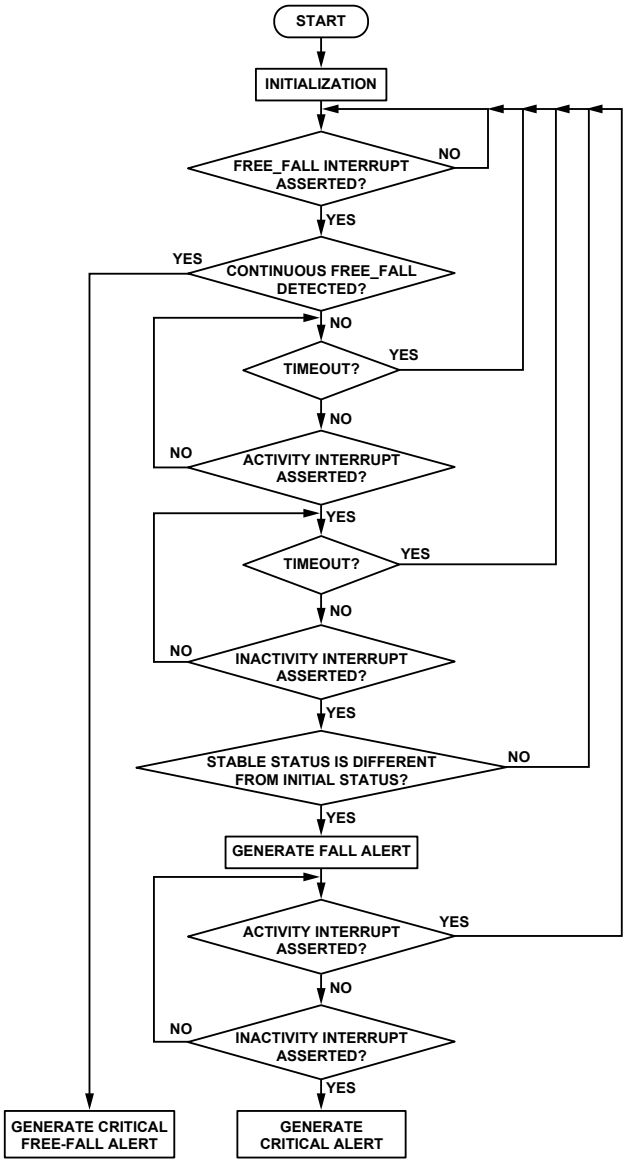


图 6. 算法流程图

算法中，关于各种中断的门限以及时间参数的设置如下所述。

1. 初始化后，系统等待Free_Fall中断（失重），这里把THRESH_FF设为0.75g，把TIME_FF设为30ms。
2. Free_Fall中断产生之后，系统开始等待Activity中断（撞击），这里把THRESH_ACT设为2g，Activity中断为DC coupled工作模式。
3. Free_Fall中断（失重）与Activity中断（撞击）之间的时间间隔设置为200ms。如果超过200ms，则认为无效。200ms计时需要通过MCU中的定时器来实现。
4. Activity中断产生之后，系统开始等待Inactivity中断（撞击后的静止），这里把THRESH_INACT设为0.1875g，把TIME_INACT设为2s，Inactivity中断为AC coupled工作模式。
5. 在Activity中断产（撞击）生之后的3.5s时间之内，应该有Inactivity中断（撞击后的静止）产生。如果超时，则认为无效。3.5s计时需要通过MCU中的定时器来实现。
6. 如果Inactivity中断之后的加速度值与初始状态（假设已知）下数值的矢量差超过0.7g，则说明检测到一次有效的跌倒，系统会给出一个报警。
7. 当检测到跌倒状态之后，为了判断是否在跌倒之后人体有长时间的静止不动。需要继续检测Activity中断和Inactivity中断。这里把THRESH_ACT设为0.5g，Activity中断为AC coupled工作模式。把THRESH_INACT设为0.1875g，把TIME_INACT设为10s，Inactivity中断为AC coupled工作模式。也就是说，如果在10s之内，人体一直没有任何动作，则会产生Inactivity中断，使系统给出一个严重报警。而在此期间一旦人体有所动作，则会产生Activity中断，从而结束整个判断过程。
8. 本算法还可以检测出人体从较高的地方跌落。如果Free_Fall中断连续产生且之间的间隔小于100ms，可以认为，人体处于连续的跌落状态。如果Free_Fall中断（失重）连续发生300ms，则说明人体是从超过0.45m的高度跌落，系统会给出一个跌落的报警。

$$S = \frac{1}{2}gt^2 = \frac{1}{2} * 10 * 0.3^2 = 0.45\,\mathrm{m}$$

本算法已在 ADuC7026 微控制器中以 C 语言实现（见附录）。本文设计了一个实验方案对算法进行验证。实验对向前跌倒，向后跌倒，向左、右两侧跌倒等不同跌倒姿势以及跌倒后是否有长时间静止状态的情况分别进行了 10 次测试，表 2 中给出的是相关测试结果。

表 2 测试结果

跌倒姿势	跌倒后长时间静止	1	2	3	4	5	6	7	8	9	10
向前跌倒	否	√	√	√	√	√	√	√	√	√	√
	是	√*	√*	√*	√*	√*	√*	√*	√*	√*	√*
向后跌倒	否	√	√	√	√	√	√	√	√	√	√
	是	√*	√*	√*	√*	√*	√*	√*	√*	√*	√*
向左侧跌倒	否	√	√	√	√	√	√	√	√	√	√
	是	√*	√*	√*	√*	√*	√*	√*	√*	√*	√*
向右侧跌倒	否	√	√	√	√	√	√	√	√	√	√
	是	√*	√*	√*	√*	√*	√*	√*	√*	√*	√*

注：符号√表示检测到跌倒，符号*表示检测到跌倒后的长时间静止。

从这个实验中可以看出基于 ADXL345 的解决方案能够有效地对跌倒状态进行检测。当然，这里只是一个简单的实验方案，仍需要进行更加全面、有效和长期的实验来验证该解决方案的可靠性。

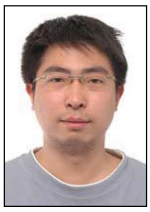
结论

ADXL345 是 ADI 公司的一款功能强大的加速度传感器产品。本文利用 ADXL345 内部的多种运动状态检测功能和灵活的中断功能，提出一种新的跌倒检测解决方案。经验证，该解决方案具有算法复杂度低，检测准确度高的优点。

附录

本算法的基于 ADXL345 和 ADuC7026 的测试平台实现。通过 Keil UV3 编译，工程中共有 4 个头文件和一个 c 文件。下面详细给出了 c 文件中源代码。

作者简介



Ning Jia [ning.jia@analog.com]是中国技术支持中心现场应用工程师，在ADI有两年的工作经历，负责向中国用户就ADI的模拟产品提供技术支持。Ning于 2007 年获北京邮电大学信号和信息处理专业硕士学位。

参考资料

[1] ADXL345 数据手册（www.analog.com,搜索 ADXL345）

[2] ADuC7026 数据手册（www.analog.com,搜索 ADuC7026）

```
// Include header files
#include "FallDetection.h"

void IRQ_Handler() __irq            // IRQ interrupt
{
   unsigned char i;

   if((IRQSTA & GP_TIMER_BIT)==GP_TIMER_BIT)//TIMER1 Interrupt, interval 20ms
      {
      T1CLRI = 0;                   // Clear Timer1 interrupt
      if(DetectionStatus==0xF2)     // Strike after weightlessness is detected, waiting for stable
      {
            TimerWaitForStable++;
            if(TimerWaitForStable>=STABLE_WINDOW)// Time out, restart
            {
                        IRQCLR  = GP_TIMER_BIT;// Disable ADuC7026's Timer1 interrupt
                        DetectionStatus=0xF0;
                        putchar(DetectionStatus);
                        ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                        ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                        ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
            ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE  | XL345_INACT_Y_ENABLE
                                                                        | XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                        | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
                                                                        | XL345_ACT_X_ENABLE | XL345_ACT_DC;
      xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
            }
      }
      else if(DetectionStatus==0xF1)// Weightlessness is detected, waiting for strike
      {
            TimerWaitForStrike++;
            if(TimerWaitForStrike>=STRIKE_WINDOW)// Time out, restart
            {
                        IRQCLR  = GP_TIMER_BIT;// Disable ADuC7026's Timer1 interrupt
                        DetectionStatus=0xF0;
                        putchar(DetectionStatus);
                        ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                        ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                        ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
            ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE  | XL345_INACT_Y_ENABLE
                                                                        | XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                        | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
                                                                        | XL345_ACT_X_ENABLE | XL345_ACT_DC;
xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
            }
      }
   }
   if((IRQSTA&SPM4_IO_BIT)==SPM4_IO_BIT)// External interrupt form ADXL345 INT0
   {
      IRQCLR = SPM4_IO_BIT;         // Disable ADuC7026's external interrupt
      xl345Read(1, XL345_INT_SOURCE, &ADXL345Registers[XL345_INT_SOURCE]);
      if((ADXL345Registers[XL345_INT_SOURCE]&XL345_ACTIVITY)==XL345_ACTIVITY)// Activity interrupt asserted
      {
            if(DetectionStatus==0xF1)// Waiting for strike, and now strike is detected
            {
                        DetectionStatus=0xF2;// Go to Status "F2"
                        putchar(DetectionStatus);
                        ADXL345Registers[XL345_THRESH_ACT]=STABLE_THRESHOLD;
                        ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                        ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
            ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE  | XL345_INACT_Y_ENABLE
                                                                        | XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                        | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
                                                                        | XL345_ACT_X_ENABLE | XL345_ACT_AC;
      xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                        IRQEN|=GP_TIMER_BIT;// Enable ADuC7026's Timer1 interrupt
                        TimerWaitForStable=0;
            }
            else if(DetectionStatus==0xF4)// Waiting for long time motionless, but a movement is detected
            {
                        DetectionStatus=0xF0;// Go to Status "F0", restart
                        putchar(DetectionStatus);
                        ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                        ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                        ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
            ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE  | XL345_INACT_Y_ENABLE
                                                                        | XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                        | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
                                                                        | XL345_ACT_X_ENABLE | XL345_ACT_DC;
   xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
            }
      }
      else if((ADXL345Registers[XL345_INT_SOURCE]&XL345_INACTIVITY)==XL345_INACTIVITY) // Inactivity interrupt asserted
      {
            if(DetectionStatus==0xF2)           // Waiting for stable, and now stable is detected
            {
                        DetectionStatus=0xF3;// Go to Status "F3"
                        IRQCLR  = GP_TIMER_BIT;
                        putchar(DetectionStatus);

                        xl345Read(6, XL345_DATAX0, &ADXL345Registers[XL345_DATAX0]);

                        DeltaVectorSum=0;
                        for(i=0;i<3; i++)
                        {
                                    Acceleration[i]=ADXL345Registers[XL345_DATAX1+i*2]&0x1F;
                                    Acceleration[i]=(Acceleration[i]<<8)|ADXL345Registers[XL345_DATAX0+i*2];
```

```
                                    if(Acceleration[i]<0x1000)
                                    {
                                              Acceleration[i]=Acceleration[i]+0x1000;
                                    }
                                    else //if(Acceleration[i]>=4096)
                                    {
                                              Acceleration[i]=Acceleration[i]-0x1000;
                                    }

                                    if(Acceleration[i]>InitialStatus[i])
                                    {
                                              DeltaAcceleration[i]=Acceleration[i]-InitialStatus[i];
                                    }
                                    else
                                    {
                                              DeltaAcceleration[i]=InitialStatus[i]-Acceleration[i];
                                    }

                                    DeltaVectorSum=DeltaVectorSum+DeltaAcceleration[i]*DeltaAcceleration[i];
                          }

                          if(DeltaVectorSum>DELTA_VECTOR_SUM_THRESHOLD)   // The stable status is different from the initial status
                          {
                                    DetectionStatus=0xF4;      // Valid  fall detection
                                    putchar(DetectionStatus);

                                    ADXL345Registers[XL345_THRESH_ACT]=STABLE_THRESHOLD;
                                    ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                                    ADXL345Registers[XL345_TIME_INACT]=NOMOVEMENT_TIME;
               ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE | XL345_INACT_Y_ENABLE
| XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                                                                  | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
| XL345_ACT_X_ENABLE | XL345_ACT_AC;
                                    xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                          }
                          else        // Delta vector sum is not exceed the threshold
                          {
                                    DetectionStatus=0xF0;      // Go to Status "F0", restart
                                    putchar(DetectionStatus);
                                    ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                                    ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                                    ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
               ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE | XL345_INACT_Y_ENABLE
| XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                                                                  | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
| XL345_ACT_X_ENABLE | XL345_ACT_DC;
                                    xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                          }
               }
               else if(DetectionStatus==0xF4) // Wait for long time motionless and now it is detected
               {
                          DetectionStatus=0xF5;      // Valid critical fall detection
                          putchar(DetectionStatus);

                          ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                          ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                          ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
               ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE | XL345_INACT_Y_ENABLE
| XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                                                  | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
| XL345_ACT_X_ENABLE | XL345_ACT_DC;
                          xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                          DetectionStatus=0xF0;              // Go to Status "F0", restart
                          putchar(DetectionStatus);
               }
          }
          else if((ADXL345Registers[XL345_INT_SOURCE]&XL345_FREEFALL)==XL345_FREEFALL) // Free Fall interrupt asserted
          {
               if(DetectionStatus==0xF0)           // Waiting for weightless, and now it is detected
               {
                          DetectionStatus=0xF1;   // Go to Status "F1"
                          putchar(DetectionStatus);
                          ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                          ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                          ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
               ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE | XL345_INACT_Y_ENABLE
| XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                                  | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
| XL345_ACT_X_ENABLE | XL345_ACT_DC;
                          xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                          IRQEN|=GP_TIMER_BIT;    // Enable ADuC7026's Timer1 interrupt
                          TimerWaitForStrike=0;
                          TimerFreeFall=0;
               }
               else if(DetectionStatus==0xF1) // Waiting for strike after weightless, and now a new free fall is detected
               {
                          if(TimerWaitForStrike<FREE_FALL_INTERVAL)       // if the Free Fall interrupt is continuously assert within the time of
"FREE_FALL_INTERVAL",
                          {           // then it is consider as a continuous free fall
                                    TimerFreeFall=TimerFreeFall+TimerWaitForStrike;
                          }
                          else        / Not a continuous free fall
                          {
                                    TimerFreeFall=0;
                          }
                          TimerWaitForStrike=0;
                          if(TimerFreeFall>=FREE_FALL_OVERTIME)                     // if the continuous time of free fall is longer than
"FREE_FALL_OVERTIME"
                          {           // consider that a free fall from high place is detected
                                    DetectionStatus=0xFF;
                                    putchar(DetectionStatus);
                                    ADXL345Registers[XL345_THRESH_ACT]=STRIKE_THRESHOLD;
                                    ADXL345Registers[XL345_THRESH_INACT]=NOMOVEMENT_THRESHOLD;
                                    ADXL345Registers[XL345_TIME_INACT]=STABLE_TIME;
               ADXL345Registers[XL345_ACT_INACT_CTL]=XL345_INACT_Z_ENABLE | XL345_INACT_Y_ENABLE
| XL345_INACT_X_ENABLE | XL345_INACT_AC
                                                                                  | XL345_ACT_Z_ENABLE | XL345_ACT_Y_ENABLE
| XL345_ACT_X_ENABLE | XL345_ACT_DC;
                                    xl345Write(4, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
                                    DetectionStatus=0xF0;
                                    putchar(DetectionStatus);
                          }
               }
               else
               {
                          TimerFreeFall=0;
               }
          }
          IRQEN  |=SPM4_IO_BIT;         // Enable ADuC7026's external interrupt
     }
}

void  main(void)
{
    ADuC7026_Initiate();                       // ADuC7026 initialization
    ADXL345_Initiate();                        // ADXL345 initialization
    DetectionStatus=0xF0;                      // Clear detection status, Start
    InitialStatus[0]=0x1000;         // X axis=0g, unsigned short int, 13 bit resolution, 0x1000 = 4096 = 0g, +/-0xFF = +/-256 = +/-1g
    InitialStatus[1]=0x0F00;         // Y axis=-1g
    InitialStatus[2]=0x1000;         // Z axis=0g
    IRQEN =SPM4_IO_BIT;                        // Enable ADuC7026's external interrupt, to receive the interrupt from ADXL345 INT0
    while(1)// Endless loop, wait for interrupts
    {
       ;
    }
}
```