3 軸加速度センサーADXL345 による転倒検出アプリケーション

著者：Ning Jia

はじめに

高齢になると注意力や身を守る能力が低下することから、転倒事故を起こす可能性が高くなります。救援が遅れると、このような事故から深刻な事態を招くことがあります。統計によれば、深刻な事態の大多数は、転倒の直接的な結果というよりも、転倒後の支援や処置の遅れが原因になっています。転倒したときに、救助の人への連絡が遅れなければ、転倒後の危険性を大きく減少させることができます。このような観点から、転倒の検出と予測のために、さまざまな機器が開発されています。

近年、MEMS 加速度センサー技術の向上によって、3 軸加速度センサーをベースにした転倒検出器の設計が可能になりました。これらの転倒検出器は、センサーを装着した人の加速度の変化を直交する 3 つの方向で追跡することによって、移動しているときの身体の位置の変化を検出するという原理で動作します。検出されたデータをアルゴリズムにより解析することで、その人が転倒したかどうかを判定します。転倒した場合、この機器は、GPS モジュールやワイヤレス・トランスミッタ・モジュールを使用して、その場所を特定し、支援を求める警報を発します。転倒検出器の中心になるものは、検出の原理と転倒という緊急事態を判断するためのアルゴリズムになります。

アナログ・デバイセズの 3 軸デジタル出力加速度センサー ADXL345 は、転倒検出器アプリケーションに最適です。このアプリケーション・ノートでは、身体の転倒検出に関する原理の研究に基づいて、転倒事態を検出するための ADXL345 によるソリューションを提案します。

ADXL345 MEMS加速度センサー

マイクロ・エレクトロ・メカニカル・システム（MEMS）とは、機械構造と電気回路をシリコン・チップに組み込む半導体技術です。MEMS 加速度センサーは、この技術をベースにしたセンサーで、1 軸、2 軸、または 3 軸の加速度を検出します。数 g～数十 g のさまざまな範囲のデジタル／アナログ出力の製品があり、複数の割込みモードを実装している製品もあります。

ADXL345 は、アナログ・デバイセズのデジタル出力 MEMS 3 軸加速度センサーです。ADXL345 の特長は、±2 *g*、±4 *g*、±8 *g*、±16 *g* の計測範囲を選択できること、最大 13 ビットの分解能、約 4 mg/LSB の感度、3 mm × 5 mm × 1 mm の超小型パッケージ、40～145 µA の超低消費電力、標準的な I^2C と SPI のデジタル・インターフェース、32 レベルの FIFO、さまざまな動作状態検出アルゴリズムを内蔵していること、柔軟な割込みシステムがあることです。これらの特長によって転倒検出のアルゴリズムが大幅に簡素化するため、ADXL345 は転倒検出器アプリケーションに最適な加速度センサーと言えます。このアプリケーション・ノートで提案する転倒検出ソリューションは、ADXL345 に内蔵されている動作状態検出と割込みシステムの機能を生かし、アルゴリズムの複雑さを最大限軽減します。実際の加速度値を取得したり、その他の計算をしたりする必要はほとんどありません。

ADXL345 の割込みシステムについては、「割込み」で説明します。ADXL345 の詳細な仕様については、データシートをご覧になるか、www.analog.com にアクセスしてください。図 1 に ADXL345 のシステム・ブロック図、図 2 にピン定義を示します。

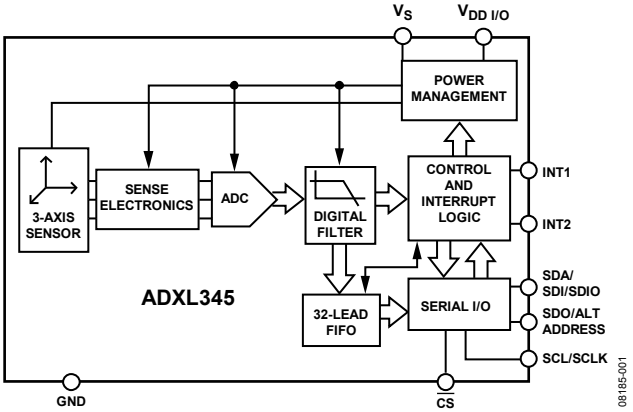

図 1. ADXL345 のシステム・ブロック図

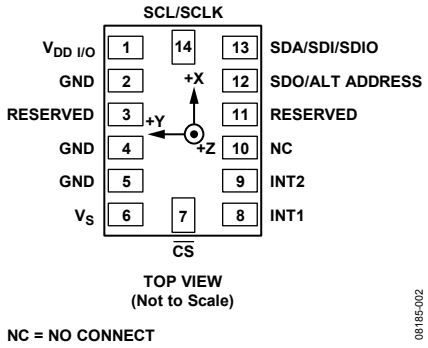

図 2. ADXL345 のピン配置

Rev. 0

アナログ・デバイセズ株式会社

本　　社／〒105-6891　東京都港区海岸 1-16-1　ニューピア竹芝サウスタワービル
電話 03（5402）8200
大阪営業所／〒532-0003　大阪府大阪市淀川区宮原 3-5-36　新大阪トラストタワー
電話 06（6350）6868

目次

割込み

ADXL345には、2本の割込みピンINT1とINT2があり、合計8つの割込み（DATA_READY、SINGLE_TAP、DOUBLE_TAP、アクティブ、インアクティブ、FREE_FALL、ウォーターマーク、オーバーラン）を各割り込みピンに任意にマッピングできます。各割込みを個別にイネーブルまたはディスエーブルにするには、INT_ENABLEレジスタの適切なビットをセットします。

DATA_READY

DATA_READY ビットは、新しいデータがレジスタにアップデートされた時にセットされ、新しいデータを読み込むとクリアされます。

SINGLE_TAP

SINGLE_TAP ビットは、THRESH_TAP レジスタの値を上回る加速度が、DUR レジスタで指定された値よりも短いパルス幅で発生した場合にセットされます。

DOUBLE_TAP

DOUBLE_TAP ビットは、THRESH_TAP レジスタの値を上回る加速度が、DUR レジスタで指定した値よりも短いパルス幅で発生し、2 番目のタップ信号が Latent レジスタによって指定した時間よりも後、Window レジスタで指定した時間以内に検出された場合にセットされます。図 3 に、有効な SINGLE_TAP 割込みと DOUBLE_TAP 割込みを示します。

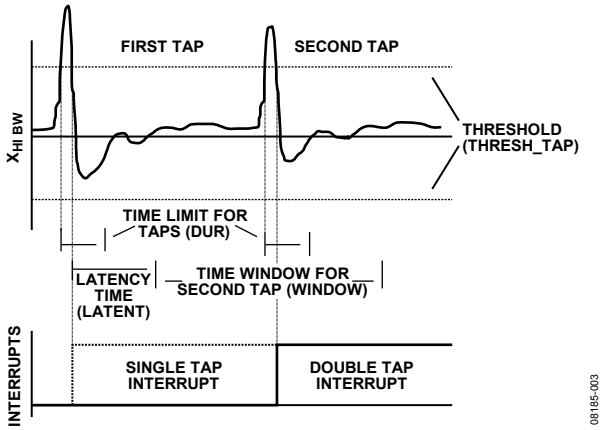

図 3. SINGLE_TAP 割込みと DOUBLE_TAP 割込み

アクティブ

Activity ビットは、THRESH_ACT レジスタで指定した値より大きい加速度が生じた場合にセットされます。

インアクティブ

Inactivity ビットは、THRESH_INACT レジスタで指定した値より小さい加速度が、TIME_INACT レジスタで指定した値よりも長い時間発生した場合にセットされます。TIME_INACT の最大値は 255 秒です。アクティブ割込みとインアクティブ割込みの場合、x、y、z の各軸をユーザが個別にイネーブルまたはディスエーブルにすることができます。たとえば、x 軸のアクティブ割込みをイネーブルにしながら、y 軸と z 軸の割込みをディセーブルにすることができます。さらに、アクティブ割込みとインアクティブ割込みについて、DC/AC カップリングの動作モードを選択できます。DC カップリング動作では、現在の加速度の大きさを THRESH_ACT および THRESH_INACT と直接比較して、アクティブ／インアクティブの検出を判定します。アクティブ検出のための AC カップリング動作では、アクティブ検出の開始時の加速度値を基準値とします。測定された加速度値を基準値と比較し、その差が THRESH_ACT 値を上回ると、デバイスがアクティブ割込みをトリガします。同様に、インアクティブ検出のための AC カップリング動作でも、比較のために基準値を使用し、デバイスがインアクティブ閾値を上回ると基準値が更新されます。基準値が選択されると、デバイスは、基準値と現在の加速度との差を THRESH_INACT と比較します。差が THRESH_INACT の値を下回る状態が TIME_INACT の合計時間以上続くと、デバイスはインアクティブ状態と見なされ、インアクティブ割込みがトリガされます。

FREE_FALL

FREE_FALL ビットは、THRESH_FF レジスタで指定した値より小さい加速度が、TIME_FF レジスタで指定した値よりも長い時間発生した場合にセットされます。FREE_FALL 割込みは、主に自由落下運動の検出に使用されます。このため、FREE_FALL 割込みでは、インアクティブ割込みと異なり、必ず全軸の値を検出に使用し、またタイマ期間がきわめて短く（最大 1.28 秒）、つねに DC カップリングになります。

ウォーターマーク

Watermark ビットは、FIFO 内のサンプル数が Samples レジスタで指定されている値になるとセットされます。FIFO が読み出され、FIFO 内のサンプル数が Samples レジスタで指定した値よりも小さくなると、Watermark ビットはクリアされます。ADXL345 の FIFO には、4 つの動作モード（バイパス・モード、FIFO モード、ストリーム・モード、トリガ・モード）があり、最大 32 のサンプル（x、y、z の 3 軸）を保存できます。FIFO 機能は ADXL345 にとって重要できわめて便利な機能ですが、転倒検出用に提案するソリューションでは FIFO 機能を使用しません。したがって、このアプリケーション・ノートではこれ以上説明しません。FIFO 機能の詳細については、ADXL345 のデータシートまたは AN-1025 を参照してください。

オーバーラン

Overrun ビットは、新しいデータが未読のまま更に新しいデータがアップデートされたときにセットされます。オーバーラン機能の厳密な動作は、FIFO モードに依存します。バイパス・モードでは、DATAX、DATAY、DATAZ の各レジスタで新しいデータが未読のデータに入れ替わったときに Overrun ビットがセットされます。それ以外のモードでは、FIFO 内のサンプル数が 32 になったときに Overrun ビットがセットされます。Overrun ビットは、FIFO の内容が読み出されるとクリアされるので、データの読出し時に自動的にクリアされます。

転倒プロセスにおける加速度の変化特性

転倒検出の原理に関する主な研究は、身体が転倒するプロセスにおける加速度の変化の特性を調べています。図4～図7に、階段を下りる、階段を上がる、椅子に座る、椅子から立ち上がるという動作における加速度の変化を表す曲線を示します（転倒検出器はその人の体にベルトで装着しています）。

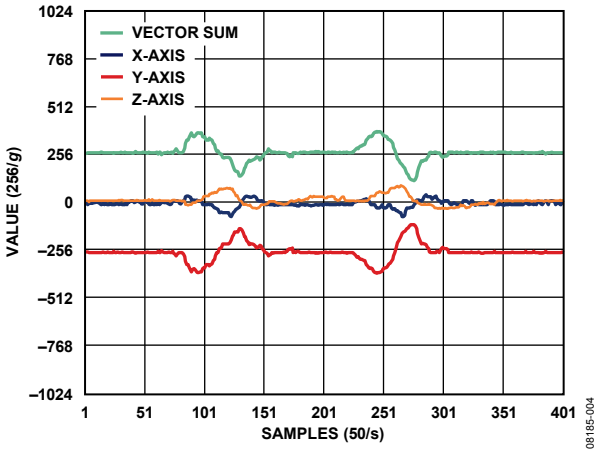

図 4.　加速度変化曲線（階段を下りるとき）

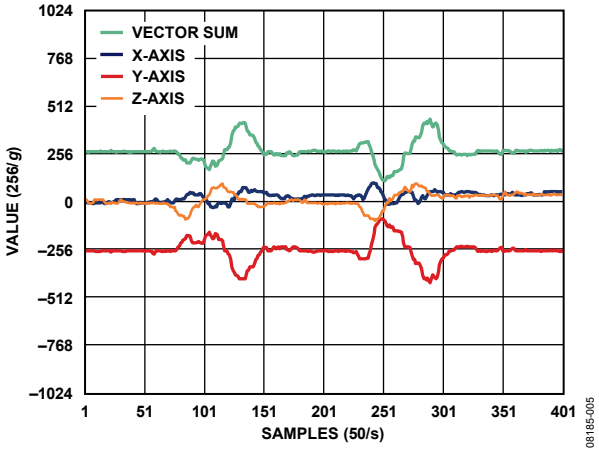

図 5.　加速度変化曲線（階段を上がるとき）

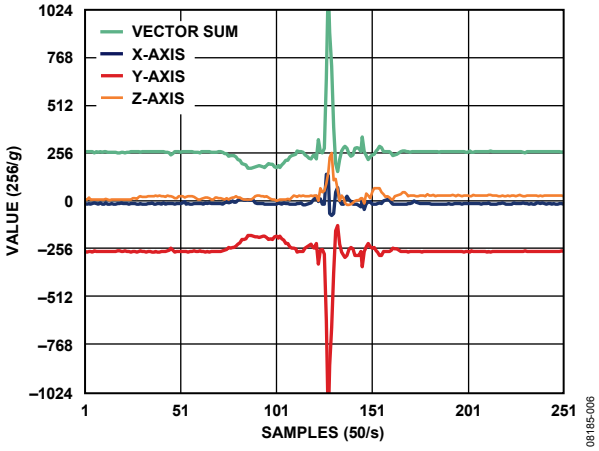

図 6.　加速度変化曲線（座るとき）

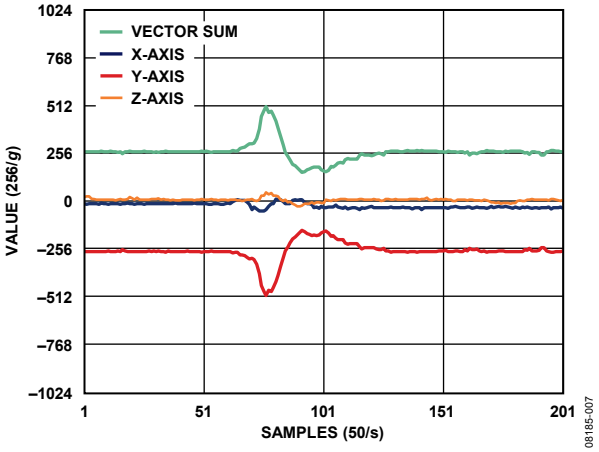

図 7.　加速度変化曲線（立ち上がるとき）

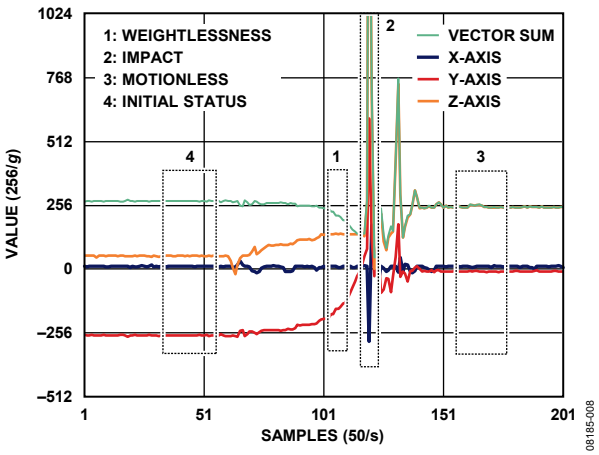

図 8.　加速度変化曲線（転倒したとき）

高齢者の動きは比較的遅いため、図4と図5の歩行動作中の加速度の変化はそれほど大きくありません。図8は、転倒したときの加速度の変化曲線です。図8を図4～図7と比較すると、転倒イベントには4つの重要な特性があることがわかります。これらの4つの特性を転倒検出の判定基準に使用することができます。これらの特性は図8の枠で示していますが、以下に詳しく説明しましょう。

WEIGHTLESSNESS

無重力状態は、転倒の最初に必ず発生します。この現象は自由落下時と同じ状態で、自由落下時には加速度のベクトル和はほぼ0 gまで減少します。継続時間は自由落下の高さによります。通常の転倒における無重力状態は自由落下の場合ほどではありませんが、加速度のベクトル和はやはり 1 g を下回ります（一般に、通常の状態では重力の影響があるため、加速度のベクトル和は1 gを上回ります）。つまり、これは転倒状態を判定するための最初の基準と考えることができ、ADXL345 の FREE_FALL 割込みによって検出することができます。

IMPACT

無重力状態の後、身体が地面に衝突します。図 8 の加速度曲線では、大きな衝撃であることが示されています。ADXL345 のアクティブ割込みが、この衝撃を検出します。つまり、転倒を判定するための 2 番目の基準は、FREE_FALL 割込み直後のアクティブ割込みです。

MOTIONLESS

一般に、転倒して衝撃を受けた身体は、すぐに起き上がることができず、しばらく静止状態のままにとどまります。これは、図 8 で加速度曲線の平らな線分として示されており、ADXL345 のインアクティブ割込みが検出します。つまり、転倒状態を判定するための 3 番目の基準は、アクティブ割込み後のインアクティブ割込みです。

INITIAL STATUS

転倒の後、身体の向きが変わるため、3 軸の加速度は転倒前の初期状態と異なるものになります。転倒検出器をベルトで身体に装着していて加速度の初期状態を取得していれば、インアクティブ割込みの後の 3 軸の加速度データを読み取り、そのサンプリング・データを初期状態と比較することができます。つまり、サンプリング・データと初期状態の差が特定の閾値（たとえば、0.7 g）を上回っていれば転倒と判定するという 4 番目の基準になります。

これらの 4 つの判定基準を組み合わせて、転倒検出アルゴリズムを開発し、これによって転倒状態の場合に警報を発するシステムを作ることができます。割込みと次の割込みの間の時間間隔は、妥当な範囲内でなければなりません。非常に高い位置からの落下でない限り、通常、FREE_FALL 割込み（WEIGHTLESSNESS）とアクティブ割込み（IMPACT）の間の時間間隔はそれほど長くなりません。同様に、アクティブ割込み（IMPCAT）とインアクティブ割込み（MOTIONLESS）の間の時間間隔もあまり長くしないでください。実用的な例と妥当な値については、「ADXL345 による転倒検出アルゴリズム」を参照してください。関連する割込み検出閾値と時間パラメータは、要求仕様に応じて最適化します。さらに、転倒によって昏睡状態などの深刻な結果になった場合、身体はもっと長い間静止状態のままになります。この状態もインアクティブ割込みによって検出することができます。つまり、転倒後に一定の長時間にわたりインアクティブ状態が続いていることが検出された場合、再び緊急警報を発することができます。

システムの代表的な回路接続

ADXL345 と MCU との回路接続はきわめて簡単です。このアプリケーション・ノートでは、ADXL345 と ADuC7026 マイクロコントローラを使用してテスト・プラットフォームを作成しました。図 9 に、ADXL345 と ADuC7026 との代表的な接続を示します。ADXL345 の $\overline{CS}$ ピンをハイレベルに接続すると、ADXL345 は I^2C モードで動作します。SDA と SCL は、I^2C バスのデータとクロックであり、ADuC7026 の対応するピンに接続します。ADuC7026 の GPIO を ADXL345 の SDO/ALT ADDRESS ピンに接続して、ADXL345 の I^2C アドレスを選択します。ADXL345 の INT1 ピンを ADuC7026 の IRQ 入力に接続して、割込み信号を生成します。図 9 に示すような回路接続を使用すれば、ほぼあらゆる MCU やプロセッサから ADXL345 にアクセスすることができます。ADXL345 を SPI モードにすれば、I^2C 接続時よりも高い速度での通信も可能です。SPI 接続の回路例については、ADXL345 のデータシートを参照してください。

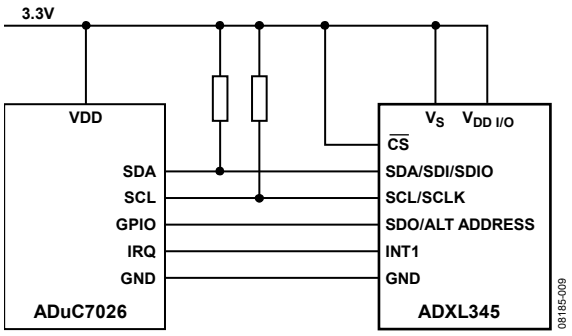

図 9. ADXL345 と MCU との代表的な回路接続

ADXL345 による転倒検出アルゴリズム

ここでは、前述のソリューションをもとにしたアルゴリズムを説明しましょう。

表 1 は、各レジスタの機能と本アルゴリズムで使用する値を示しています。各レジスタ・ビットの詳細な定義については、ADXL345 データシートを参照してください。

なお、表 1 に示すレジスタの一部には、2 つのアルゴリズム設定値が記載されています。これは、検出目的によって、2 つの値のどちらかを選択できることを示しています。アルゴリズムのフローチャートを図 10 に示します。

表 1. ADXL345 のレジスタの機能の説明

Hex Address	Dec Address	Register Name	Type	Reset Value	Description	Settings in Algorithm	Function of the Settings in Algorithm
0x00	0	DEVID	Read-only	0xE5	Device ID	Read-only	
0x01 to 0x1C	1 to 28	Reserved	Reserved		Reserved, do not access	Reserved	
0x1D	29	THRESH_TAP	Read/write	0x00	Tap threshold	Not used	
0x1E	30	OFSX	Read/write	0x00	X-axis offset	0x06	X-axis offset compensation, obtain from initialization calibration
0x1F	31	OFSY	Read/write	0x00	Y-axis offset	0xF9	Y-axis offset compensation, obtain from initialization calibration
0x20	32	OFSZ	Read/write	0x00	Z-axis offset	0xFC	Z-axis offset compensation, obtain from initialization calibration
0x21	33	DUR	Read/write	0x00	Tap duration	Not used	
0x22	34	Latent	Read/write	0x00	Tap latency	Not used	
0x23	35	Window	Read/write	0x00	Tap window	Not used	
0x24	36	THRESH_ACT	Read/write	0x00	Activity threshold	0x20/0x08	Set activity threshold as 2 *g*/0.5 *g*
0x25	37	THRESH_INACT	Read/write	0x00	Inactivity threshold	0x03	Set inactivity threshold as 0.1875 *g*
0x26	38	TIME_INACT	Read/write	0x00	Inactivity time	0x02/0x0A	Set inactivity time as 2 sec or 10 sec
0x27	39	ACT_INACT_CTL	Read/write	0x00	Axis enable control for activity/inactivity	0x7F/0xFF	Enable activity and inactivity of x-, y-, z-axis, wherein inactivity is ac-coupled mode, activity is dc-coupled/ ac-coupled mode
0x28	40	THRESH_FF	Read/write	0x00	Free-fall threshold	0x0C	Set free-fall threshold as 0.75 *g*
0x29	41	TIME_FF	Read/write	0x00	Free-fall time	0x06	Set free-fall time as 30 ms
0x2A	42	TAP_AXES	Read/write	0x00	Axis control for tap/double tap	Not used	
0x2B	43	ACT_TAP_STATUS	Read-only	0x00	Source of activity/tap	Read-only	
0x2C	44	BW_RATE	Read/write	0x0A	Data rate and power mode control	0x0A	Set sample rate as 100 Hz
0x2D	45	POWER_CTL	Read/write	0x00	Power save features control	0x00	Set as normal working mode
0x2E	46	INT_ENABLE	Read/write	0x00	Interrupt enable control	0x1C	Enable activity, inactivity, free-fall interrupts
0x2F	47	INT_MAP	Read/write	0x00	Interrupt mapping control	0x00	Map all interrupts to INT1 pin
0x30	48	INT_SOURCE	Read-only	0x00	Source of interrupts	Read-only	
0x31	49	DATA_FORMAT	Read/write	0x00	Data format control	0x0B	Set as ±16 *g* measurement range, 13-bit right alignment, high level interrupt trigger, I^2C interface
0x32	50	DATAX0	Read-only	0x00	X-Axis Data 0	Read-only	
0x33	51	DATAX1	Read-only	0x00	X-Axis Data 1	Read-only	
0x34	52	DATAY0	Read-only	0x00	Y-Axis Data 0	Read-only	
0x35	53	DATAY1	Read-only	0x00	Y-Axis Data 1	Read-only	
0x36	54	DATAZ0	Read-only	0x00	Z-Axis Data 0	Read-only	
0x37	55	DATAZ1	Read-only	0x00	Z-Axis Data 1	Read-only	
0x38	56	FIFO_CTL	Read/write	0x00	FIFO control	Not used	
0x39	57	FIFO_STATUS	Read-only	0x00	FIFO status	Not used	

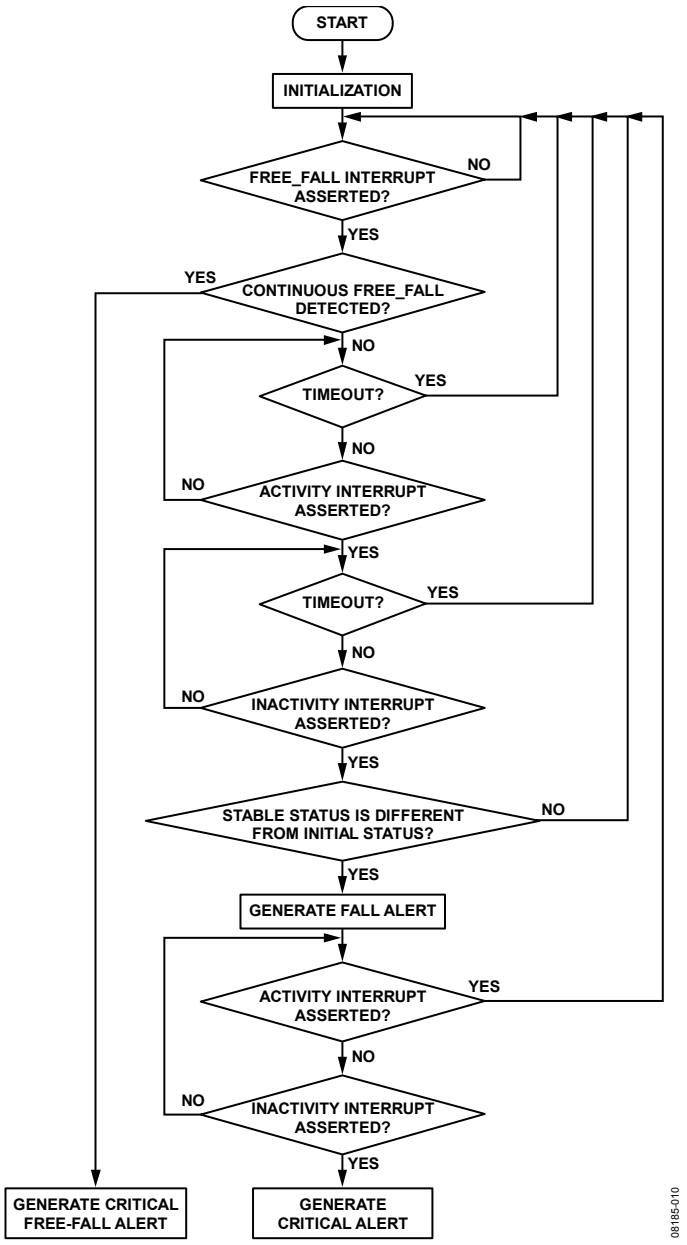

図 10.　アルゴリズムのフローチャート

アルゴリズムの各割込み閾値と関連する時間パラメータは、以下の通りです。

1. 初期化の後、システムは FREE_FALL 割込み（WEIGHTLESSNESS）を待ちます。THRESH_FF の設定は 0.75 g、TIME_FF は 30 ms です。
2. FREE_FALL 割込みがアサートされると、システムはアクティブ割込み（IMPACT）を待ちます。THRESH_ACT の設定は 2 g で、アクティブ割込みは DC カップリング・モードです。
3. FREE_FALL 割込み（WEIGHTLESSNESS）とアクティブ割込み（衝撃）との時間間隔は 200 ms です。2 つの割込みの間の時間が 200 ms を上回ると、状態は有効になりません。200 ms のカウントは、MCU のタイマで実行します。
4. アクティブ割込みがアサートされると、システムはインアクティブ割込み（MOTIONLESS）を待ちます。THRESH_INACT の設定は 0.1875 g で、TIME_INACT は 2 秒です。インアクティブ割込みは AC カップリング・モードです。
5. インアクティブ割込み（MOTIONLESS）は、アクティブ割込み（IMPACT）から 3.5 秒以内にアサートされる必要があります。さもなければ、結果は無効となりアルゴリズムは初期状態に戻ります。3.5 秒のカウントは、MCU のタイマで実行します。
6. MOTIONLESS と INITIAL STATUS との加速度の差が 0.7 g の以上であると、有効な転倒として認識され、システムは転倒警報を発します。
7. 転倒を検出した後、アクティブ割込みとインアクティブ割込みを連続的に監視し、転倒後に長い静止時間があるかどうか判断する必要があります。THRESH_ACT の設定は 0.5 g で、アクティブ割込みは AC カップリング・モードです。THRESH_INACT の設定は 0.1875 g で、TIME_INACT は 10 秒、インアクティブ割込みは AC カップリング・モードです。つまり、身体の静止状態が 10 秒間続くと、インアクティブ割込みがアサートされ、システムが緊急警報を発します。身体が動くと、アクティブ割込みが生成され、すべてのシーケンスが完了します。
8. このアルゴリズムでは、身体が高い位置から自由落下した場合も検出できます。2つのFREE_FALL割込みの間隔が100 msを下回ると、連続していると見なされます。FREE_FALL割込み（無重力状態）が300 msの間連続的にアサートされた場合、緊急の自由落下警報を発します。次式に自由落下の高さと落下時間の関係を示します。300 msの落下時間は0.45 mからの自由落下に相当します。

$$S = \frac{1}{2}gt^2 = \frac{1}{2} \times 10 \times 0.3^2 = 0.45 \text{ m}$$

このアルゴリズムは、ADuC7026 マイクロコントローラで実行するために C 言語で作成されています。提案したソリューションは、アルゴリズムを検証するためにテストを行い動作を確認しています。前方への転倒、後方への転倒、右側への転倒、左側への転倒などの各位置を 20 回テストします。最初の 10 回のテストは、転倒後に長い静止時間がない代表的な転倒です。次の 10 回のテストは、転倒後に長い静止時間がある代表的な転倒です。表 2 にテスト結果を示します。

この実験により、提案した ADXL345 ベースのソリューションによって転倒状態が的確に検出できることがわかります。ただし、これは簡単な実験にすぎません。提案したソリューションの信頼性を検証するには、より包括的で効果的な長期間の実験が必要です。

表 2. テスト結果

	Test Condition		Test Result	
Trial No.	Falling Position	With Prolonged Motionless Period After Fall	Fall Detected (No. of Times)	Prolonged Motionless Detected (No. of Times)
1 to 10	Falling forward	No	10	0
11 to 20	Falling forward	Yes	10	10
21 to 30	Falling backward	No	10	0
31 to 40	Falling backward	Yes	10	10
41 to 50	Falling to the left	No	10	0
51 to 60	Falling to the left	Yes	10	10
61 to 70	Falling to the right	No	10	0
71 to 80	Falling to the right	Yes	10	10

コード例

ここでは、提案したソリューションによる ADXL345 と ADuC7026 のプラットフォームの C コード例を示します。このプロジェクトには 4 つの.h ファイルと 1 つの.c ファイルがあり、Keil UV3 によってコンパイルされます。FallDetection.c ファイルには転倒検出アルゴリズムが入っています。FallDetection.h には、転倒検出アルゴリズムに使用する定義と変数、ADXL345 の読出し／書込み関数の実装、ADXL345 の初期化の詳細があります。ADuC7026Driver.h には、ADuC7026 GPIO 制御関数、I^2C のマスタ読出し／書込み関数、ADuC7026 の初期化があります。xl345.h ファイルには、ADXL345 のレジスタとビットの定義があります。xl345_io.h ファイルには、I^2C と SPI の両方での ADXL345 のバースト読出し／書込みのための関数が入っています。

FallDetection.c

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

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]>= 0x1000)
                              {
                                    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 does not exceed the threshold
                         {
                                DetectionStatus=0xF0;  // Go to Status "F0", restar
                                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 considered 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
        {
               ;
        }
}
```

FallDetection.h

```
#include "ADuC7026Driver.h"
#include "xl345.h"
#include "xl345_io.h"

// Definitions used for Fall Detection Algorithm
#define        STRIKE_THRESHOLD                    0x20    //62.5mg/LSB, 0x20=2g
#define        STRIKE_WINDOW                       0x0A    //20ms/LSB, 0x0A=10=200ms
#define        STABLE_THRESHOLD                    0x08    //62.5mg/LSB, 0x10=0.5g
#define        STABLE_TIME                         0x02    //1s/LSB, 0x02=2s
#define        STABLE_WINDOW                       0xAF    //20ms/LSB, 0xAF=175=3.5s
#define        NOMOVEMENT_THRESHOLD                0x03    //62.5mg/LSB, 0x03=0.1875g
#define        NOMOVEMENT_TIME                     0x0A    //1s/LSB, 0x0A=10s
#define        FREE_FALL_THRESHOLD                 0x0C    //62.5mg/LSB, 0x0C=0.75g
#define        FREE_FALL_TIME                      0x06    //5ms/LSB, 0x06=30ms
#define        FREE_FALL_OVERTIME                  0x0F    //20ms/LSB, 0x0F=15=300ms
#define        FREE_FALL_INTERVAL                  0x05    //20ms/LSB, 0x05=100ms
#define        DELTA_VECTOR_SUM_THRESHOLD          0x7D70  //1g=0xFF, 0x7D70=0.7g^2

// Variables used for Fall Detection Algorithm
unsigned char    DetectionStatus;    // Detection status:
                                     //    0xF0:    Start
                                     //    0xF1:    Weightlessness
                                     //    0xF2:    Strike after weightlessness
                                     //    0xF3:    Stable after strike, valid fall detection
                                     //    0xF4:    Long time motionless, valid critical fall detection
                                     //    0xFF:    Continuous free fall, free fall from a high place
unsigned char    TimerWaitForStable;  // Counter of time that wait for stable after strike
unsigned char    TimerWaitForStrike;  // Counter of time that wait for strike after weightless
unsigned char    TimerFreeFall;       // Counter of continuous time for free fall

unsigned short int    InitialStatus[3];     // Initial status for X-, Y-, Z- axis
unsigned short int    Acceleration[3];      // Acceleration for X-, Y-, Z- axis
unsigned long int     DeltaAcceleration[3]; // Acceleration[] - Initial_Status[]
unsigned long int     DeltaVectorSum;       // Vector sum of the DeltaAcceleration[]

BYTE ADXL345Registers[57];    // ADXL345 registers array, total 57 registers in ADXL345

// Implementation of the read function based ADuC7026
void xl345Read(unsigned char count, unsigned char regaddr, unsigned char *buf)
{
       BYTE r;
       WORD RegisterAddress;
       for (r=0;r<count;r++)  // Read the register
       {
              RegisterAddress = regaddr+r;
              WriteData[0] = RegisterAddress;
              ReadViaI2C(XL345_ALT_ADDR, 0, 1);
              buf[r] = ReadData[0];
       }
}
```

```
// Implementation of the write function based ADuC7026
void xl345Write(unsigned char count, unsigned char regaddr, unsigned char *buf)
{
	BYTE r;
	WORD RegisterAddress;
	for (r=0;r<count;r++)  // Write the register
	{
		RegisterAddress = regaddr+r;
		WriteData[0] = RegisterAddress;
		WriteData[1] = buf[r];
		WriteViaI2C(XL345_ALT_ADDR, 0, 1);
	}
}

void ADXL345_Initiate()    // ADXL345 initialization, refer to ADXL345 data sheet
{
	xl345Read(1, XL345_DEVID, &ADXL345Registers[XL345_DEVID]);
	//putchar(ADXL345Registers[XL345_DEVID]);    //byte
	ADXL345Registers[XL345_OFSX]=0xFF;
	ADXL345Registers[XL345_OFSY]=0x05;
	ADXL345Registers[XL345_OFSZ]=0xFF;
	xl345Write(3, XL345_OFSX, &ADXL345Registers[XL345_OFSX]);
	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;
	ADXL345Registers[XL345_THRESH_FF]=FREE_FALL_THRESHOLD;
	ADXL345Registers[XL345_TIME_FF]=FREE_FALL_TIME;
	xl345Write(6, XL345_THRESH_ACT, &ADXL345Registers[XL345_THRESH_ACT]);
	ADXL345Registers[XL345_BW_RATE]=XL345_RATE_100;
	ADXL345Registers[XL345_POWER_CTL]=XL345_STANDBY;
	ADXL345Registers[XL345_INT_ENABLE]=XL345_ACTIVITY | XL345_INACTIVITY | XL345_FREEFALL;
	ADXL345Registers[XL345_INT_MAP]=0x00;
	xl345Write(4, XL345_BW_RATE, &ADXL345Registers[XL345_BW_RATE]);
	ADXL345Registers[XL345_DATA_FORMAT]=XL345_FULL_RESOLUTION | XL345_DATA_JUST_RIGHT | XL345_RANGE_16G;
	xl345Write(1, XL345_DATA_FORMAT, &ADXL345Registers[XL345_DATA_FORMAT]);
	ADXL345Registers[XL345_POWER_CTL]=XL345_MEASURE;
	xl345Write(1, XL345_POWER_CTL, &ADXL345Registers[XL345_POWER_CTL]);
	xl345Read(1, XL345_INT_SOURCE, &ADXL345Registers[XL345_INT_SOURCE]);
}
```

ADuC7026Driver.h

```
#include <ADuC7026.h>

// Definitions of data type
#define BYTE    unsigned char        // 8_bits
#define WORD    unsigned short int   // 16_bits
#define DWORD   unsigned long int    // 32_bits

#define ADXL345_I2C_ADDRESS_SELECT    0x40    // GPIO:P4.0, to select the ADXL345's I2C address

// Variables for I2C operation, to implement burst read/write based ADuC7026, maximum number to burst read/write
is 8 bytes
BYTE Steps, Status;
BYTE ReadData[8], WriteData[9];

// Rewrite the putchar() function, send one byte data via UART
int putchar(int ch)
{
       COMTX=ch;
       while(!(0x020==(COMSTA0 & 0x020)))
       {;}
       return ch;
}

//GPIO Control functions
void OutputBit(BYTE GPIONum, BYTE Data)    // Write the pin of "GPIONum" with "Data" (0 or 1)
{
       DWORD Temp;
       Temp=1<<(GPIONum&0x0F);
       switch(GPIONum>>4)
       {
              case 0:
                     GP0DAT|=(Temp<<24);
                     if(Data==0)
                     {
                            GP0CLR=(Temp<<16);
                     }
                     else
                     {
                            GP0SET=(Temp<<16);
                     }
                     break;
              case 1:
                     GP1DAT|=(Temp<<24);
                     if(Data==0)
                     {
                            GP1CLR=(Temp<<16);
                     }
                     else
                     {
                            GP1SET=(Temp<<16);
                     }
```

```
                break;
        case 2:
                GP2DAT|=(Temp<<24);
                if(Data==0)
                {
                        GP2CLR=(Temp<<16);
                }
                else
                {
                        GP2SET=(Temp<<16);
                }
                break;
        case 3:
                GP3DAT|=(Temp<<24);
                if(Data==0)
                {
                        GP3CLR=(Temp<<16);
                }
                else
                {
                        GP3SET=(Temp<<16);
                }
                break;
        case 4:
                GP4DAT|=(Temp<<24);
                if(Data==0)
                {
                        GP4CLR=(Temp<<16);
                }
                else
                {
                        GP4SET=(Temp<<16);
                }
                break;
    }
}

// ADuC7026 initialization
void UART_Initiate()  // ADuC7026 UART initialization, initiate the UART Port to 115200bps
{
    POWKEY1 = 0x01;         // Start PLL setting,changeless
    POWCON=0x00;
    POWKEY2 = 0xF4;         // Finish PLL setting,changeless
    GP1CON = 0x2211;        // I2C on P1.2 and P1.3. Setup tx & rx pins on P1.0 and P1.1 for UART
    COMCON0 = 0x80;         // Setting DLAB
    COMDIV0 = 0x0B;         // Setting DIV0 and DIV1 to DL calculated
    COMDIV1 = 0x00;
    COMCON0 = 0x07;         // Clearing DLAB
    COMDIV2 = 0x883E;       // Fractional divider
                                    // M=1
                                    // N=01101010101=853
                                    // M+N/2048=1.4165
                                    // 41.78MHz/(16*2*2^CD*DL*(M+N/2048))  //CD=0  DL=0B=11
```

```
                                        // 115.2Kbps  M+N/2048 =1.0303  M=1, N=62=0x3EH=000 0011 1110
                                        //comdiv2=0x883E
}

void I2C1_Initiate()   // ADuC7026 I2C1 initialization, initiate the I2C1 Port to 100kbps
{
       GP1CON = 0x2211;         // I2C on P1.2 and P1.3. Setup tx & rx pins on P1.0 and P1.1 for UART
       I2C1CFG = 0x82;          // Master Enable & Enable Generation of Master Clock
       I2C1DIV = 0x3232;        // 0x3232 = 400kHz
                                // 0xCFCF = 100kHz
       FIQEN |= SM_MASTER1_BIT;    //Enable I2C1 Master Interupt
}

void Timer1_Initiate()  // ADuC7026 Timer1 initialization, Interval = 20ms
{
       T1LD = 0xCC010;
       T1CON = 0xC0;
}

void ADuC7026_Initiate(void)  // ADuC7026 initialization, initiate the UART, I2C1, Timer1, and GPIOs
{
       UART_Initiate();
       I2C1_Initiate() ;
       Timer1_Initiate();
       OutputBit(ADXL345_I2C_ADDRESS_SELECT,0); //Grounding the SDO (p4.0), I2C address for writing and reading
is 0xA6 and 0xA7
}

// ADuC7026 I2C1 Master, implement burst read/write based ADuC7026, maximum number to burst read/write is 8 bytes
// support 1 byte address and dual byte address
// enable I2C1 interrupt as FIQ interrupt, burst read/write is realized in the FIQ interrupt

void WriteViaI2C(BYTE DeviceAddr, BYTE AddrType, BYTE NumberOfWriteBytes)
// Write "NumberOfWriteBytes" data to "DeviceAddr" address
// AddrType=0, single-byte address;  AddrType=1, dual byte address
// Data to write is saved in "WriteData[]"
{
       Status=0;
       Steps=NumberOfWriteBytes+AddrType+1;
       I2C1ADR = DeviceAddr<<1;
       I2C1CNT=NumberOfWriteBytes+AddrType-1;
       I2C1MTX = WriteData[Status];
       while(Steps != Status)
       {
              ;
       }
}

void ReadViaI2C(BYTE DeviceAddr, BYTE AddrType, BYTE NumberOfReadBytes)
// Read "NumberOfWriteBytes" data from "DeviceAddr" address
// AddrType=0, single byte address;   AddrType=1, dual byte address
// Readback data is saved in "ReadData[]"
{
```

```
        Status=0;
        Steps=AddrType+1;
        I2C1ADR = DeviceAddr<<1;
        I2C1MTX = WriteData[Status];
        while(Steps != Status)
        {
               ;
        }
        Status=0;
        Steps=NumberOfReadBytes;
        I2C1CNT=NumberOfReadBytes-1;
        I2C1ADR = (DeviceAddr<<1)+1;
        while(Steps != Status)
        {
               ;
        }
}

void FIQ_Handler() __fiq    // FIQ interrupt
{
        // ADuC7026 Transmit
        if(((I2C1MSTA & 0x4) == 0x4) && (Status < (Steps-1)) )
        {
               Status++;
               I2C1MTX = WriteData[Status];
        }
        else if(((I2C1MSTA & 0x4) == 0x4) && (Status == (Steps-1)))
        {
               Status ++;
        }
        // ADuC7026 Receive
        else if (((I2C1MSTA & 0x8) == 0x8) && (Status <= (Steps-1)))
        {
               ReadData[Status] =  I2C1MRX;
               Status ++;
        }
}
```

xl345.h

```
/*---------------------------------------------------------------------
  The present firmware, which is for guidance only, aims at providing
  customers with coding information regarding their products in order
  for them to save time.  As a result, Analog Devices shall not be
  held liable for any direct, indirect, or consequential damages with
  respect to any claims arising from the content of such firmware and/or
  the use made by customers of the coding information contained herein
  in connection with their products.
---------------------------------------------------------------------*/

#ifndef __XL345_H
#define __XL345_H

/* --- I2C addresses --- */
/* The primary slave address is used when the SDO pin is tied or pulled
   high.  The alternate address is selected when the SDO pin is tied or
   pulled low.  When building the hardware, if you intend to use I2C,
   the state of the SDO pin must be set.  The SDO pin is also used for
   SPI communication.  To save system power, there is no internal pull-up
   or pull-down.                                                      */
#define XL345_SLAVE_ADDR    0x1d
#define XL345_ALT_ADDR      0x53
/* additional I2C defines for communications functions that need the
   address shifted with the read/write bit appended                   */
#define XL345_SLAVE_READ    XL345_SLAVE_ADDR << 1 | 0x01
#define XL345_SLAVE_WRITE   XL345_SLAVE_ADDR << 1 | 0x00
#define XL345_ALT_READ      XL345_ALT_ADDR << 1 | 0x01
#define XL345_ALT_WRITE     XL345_ALT_ADDR << 1 | 0x00

/* ------- Register names ------- */
#define XL345_DEVID           0x00
#define XL345_RESERVED1       0x01
#define XL345_THRESH_TAP      0x1d
#define XL345_OFSX            0x1e
#define XL345_OFSY            0x1f
#define XL345_OFSZ            0x20
#define XL345_DUR             0x21
#define XL345_LATENT          0x22
#define XL345_WINDOW          0x23
#define XL345_THRESH_ACT      0x24
#define XL345_THRESH_INACT    0x25
#define XL345_TIME_INACT      0x26
#define XL345_ACT_INACT_CTL   0x27
#define XL345_THRESH_FF       0x28
#define XL345_TIME_FF         0x29
#define XL345_TAP_AXES        0x2a
#define XL345_ACT_TAP_STATUS  0x2b
#define XL345_BW_RATE         0x2c
#define XL345_POWER_CTL       0x2d
#define XL345_INT_ENABLE      0x2e
#define XL345_INT_MAP         0x2f
#define XL345_INT_SOURCE      0x30
```

```
#define XL345_DATA_FORMAT     0x31
#define XL345_DATAX0          0x32
#define XL345_DATAX1          0x33
#define XL345_DATAY0          0x34
#define XL345_DATAY1          0x35
#define XL345_DATAZ0          0x36
#define XL345_DATAZ1          0x37
#define XL345_FIFO_CTL        0x38
#define XL345_FIFO_STATUS     0x39

/*---------------------------------------------------------------------
  Bit field definitions and register values
  ---------------------------------------------------------------------*/
//#define XL345_
/* register values for DEVID                                          */
/* The device ID should always read this value, The customer does not
  need to use this value but it can be read to check that the
  device can communicate                                             */

#define XL345_ID              0xe5

/* Reserved soft reset value                                          */
#define XL345_SOFT_RESET      0x52

/* Registers THRESH_TAP through TIME_INACT take only 8-bit values
   There are no specific bit fields in these registers               */

/* Bit values in ACT_INACT_CTL                                        */
#define XL345_INACT_Z_ENABLE  0x01
#define XL345_INACT_Z_DISABLE 0x00
#define XL345_INACT_Y_ENABLE  0x02
#define XL345_INACT_Y_DISABLE 0x00
#define XL345_INACT_X_ENABLE  0x04
#define XL345_INACT_X_DISABLE 0x00
#define XL345_INACT_AC        0x08
#define XL345_INACT_DC        0x00
#define XL345_ACT_Z_ENABLE    0x10
#define XL345_ACT_Z_DISABLE   0x00
#define XL345_ACT_Y_ENABLE    0x20
#define XL345_ACT_Y_DISABLE   0x00
#define XL345_ACT_X_ENABLE    0x40
#define XL345_ACT_X_DISABLE   0x00
#define XL345_ACT_AC          0x80
#define XL345_ACT_DC          0x00

/* Registers THRESH_FF and TIME_FF take only 8-bit values
   There are no specific bit fields in these registers               */

/* Bit values in TAP_AXES                                             */
#define XL345_TAP_Z_ENABLE    0x01
#define XL345_TAP_Z_DISABLE   0x00
#define XL345_TAP_Y_ENABLE    0x02
#define XL345_TAP_Y_DISABLE   0x00
```

```
#define XL345_TAP_X_ENABLE    0x04
#define XL345_TAP_X_DISABLE   0x00
#define XL345_TAP_SUPPRESS    0x08

/* Bit values in ACT_TAP_STATUS                                        */
#define XL345_TAP_Z_SOURCE    0x01
#define XL345_TAP_Y_SOURCE    0x02
#define XL345_TAP_X_SOURCE    0x04
#define XL345_STAT_ASLEEP     0x08
#define XL345_ACT_Z_SOURCE    0x10
#define XL345_ACT_Y_SOURCE    0x20
#define XL345_ACT_X_SOURCE    0x40

/* Bit values in BW_RATE                                               */
/* Expresed as output data rate */
#define XL345_RATE_3200       0x0f
#define XL345_RATE_1600       0x0e
#define XL345_RATE_800        0x0d
#define XL345_RATE_400        0x0c
#define XL345_RATE_200        0x0b
#define XL345_RATE_100        0x0a
#define XL345_RATE_50         0x09
#define XL345_RATE_25         0x08
#define XL345_RATE_12_5       0x07
#define XL345_RATE_6_25       0x06
#define XL345_RATE_3_125      0x05
#define XL345_RATE_1_563      0x04
#define XL345_RATE__782       0x03
#define XL345_RATE__39        0x02
#define XL345_RATE__195       0x01
#define XL345_RATE__098       0x00

/* Expressed as output bandwidth */
/* Use either the bandwidth or rate code,
   whichever is more appropriate for your application */
#define XL345_BW_1600         0x0f
#define XL345_BW_800          0x0e
#define XL345_BW_400          0x0d
#define XL345_BW_200          0x0c
#define XL345_BW_100          0x0b
#define XL345_BW_50           0x0a
#define XL345_BW_25           0x09
#define XL345_BW_12_5         0x08
#define XL345_BW_6_25         0x07
#define XL345_BW_3_125        0x06
#define XL345_BW_1_563        0x05
#define XL345_BW__782         0x04
#define XL345_BW__39          0x03
#define XL345_BW__195         0x02
#define XL345_BW__098         0x01
#define XL345_BW__048         0x00

#define XL345_LOW_POWER       0x08
```

```
#define XL345_LOW_NOISE       0x00
/* Bit values in POWER_CTL                                          */
#define XL345_WAKEUP_8HZ           0x00
#define XL345_WAKEUP_4HZ           0x01
#define XL345_WAKEUP_2HZ           0x02
#define XL345_WAKEUP_1HZ           0x03
#define XL345_SLEEP                0x04
#define XL345_MEASURE              0x08
#define XL345_STANDBY              0x00
#define XL345_AUTO_SLEEP           0x10
#define XL345_ACT_INACT_SERIAL     0x20
#define XL345_ACT_INACT_CONCURRENT 0x00

/* Bit values in INT_ENABLE, INT_MAP, and INT_SOURCE are identical.
   Use these bit values to read or write any of these registers.        */
#define XL345_OVERRUN              0x01
#define XL345_WATERMARK            0x02
#define XL345_FREEFALL             0x04
#define XL345_INACTIVITY           0x08
#define XL345_ACTIVITY             0x10
#define XL345_DOUBLETAP            0x20
#define XL345_SINGLETAP            0x40
#define XL345_DATAREADY            0x80

/* Bit values in DATA_FORMAT                                        */

/* Register values read in DATAX0 through DATAZ1 are dependent on the
   value specified in data format.  Customer code will need to interpret
   the data as desired.                                             */
#define XL345_RANGE_2G             0x00
#define XL345_RANGE_4G             0x01
#define XL345_RANGE_8G             0x02
#define XL345_RANGE_16G            0x03
#define XL345_DATA_JUST_RIGHT      0x00
#define XL345_DATA_JUST_LEFT       0x04
#define XL345_10BIT                0x00
#define XL345_FULL_RESOLUTION      0x08
#define XL345_INT_LOW              0x20
#define XL345_INT_HIGH             0x00
#define XL345_SPI3WIRE             0x40
#define XL345_SPI4WIRE             0x00
#define XL345_SELFTEST             0x80

/* Bit values in FIFO_CTL                                           */
/* The low bits are a value 0 to 31 used for the watermark or the number
   of pre-trigger samples when in triggered mode                    */
#define XL345_TRIGGER_INT1         0x00
#define XL345_TRIGGER_INT2         0x20
#define XL345_FIFO_MODE_BYPASS     0x00
#define XL345_FIFO_RESET           0x00
#define XL345_FIFO_MODE_FIFO       0x40
#define XL345_FIFO_MODE_STREAM     0x80
#define XL345_FIFO_MODE_TRIGGER    0xc0
```

```
/* Bit values in FIFO_STATUS                                        */
/* The low bits are a value 0 to 32 showing the number of entries
   currently available in the FIFO buffer                           */

#define XL345_FIFO_TRIGGERED       0x80


#endif /* __XL345_H */
```

xl345_io.h

```
/*---------------------------------------------------------------------
  The present firmware, which is for guidance only, aims at providing
  customers with coding information regarding their products in order
  for them to save time.  As a result, Analog Devices shall not be
  held liable for any direct, indirect, or consequential damages with
  respect to any claims arising from the content of such firmware and/or
  the use made by customers of the coding information contained herein
  in connection with their products.
---------------------------------------------------------------------*/
#ifndef __XL345_IO_H
#define __XL345_IO_H
#include "XL345.h"
/* Wrapper functions for reading and writing bursts to / from the ADXL345
   These can use I2C or SPI.  Will need to be modified for your hardware
*/

/*
  The read function takes a byte count, a register address, and a
  pointer to the buffer where to return the data.  When the read
  function runs in I2C as an example, it goes through the following
  sequence:
    1) I2C start
    2) Send the correct I2C slave address + write
    3) Send the register address
    4) I2C stop
    6) I2C start
    7) Send the correct I2C slave address + read
    8) I2C read for each byte but the last one + ACK
    9) I2C read for the last byte + NACK
   10) I2C stop
*/
void xl345Read(unsigned char count, unsigned char regaddr, unsigned char *buf);
/*
  The write function takes a byte count and a pointer to the buffer
  with the data.  The first byte of the data should be the start
  register address, the remaining bytes will be written starting at
  that register.  The minimum byte count that should be passed is 2,
  one byte of address, followed by a byte of data.  Multiple
  sequential registers can be written with longer byte counts. When
  the write function runs in I2C as an example, it goes through the
  following sequence:
    1) I2C start
    2) Send the correct I2C slave address + write
    3) Send the number of bytes requested form the buffer
    4) I2C stop
*/
void xl345Write(unsigned char count, unsigned char regaddr, unsigned char *buf);
#endif
```

結論

ADXL345 は、アナログ・デバイセズの高性能加速度センサーです。このアプリケーション・ノートでは、内蔵されているさまざまな動作状態検出機能と柔軟な割込みを利用して、転倒検出のための新しいソリューションを提案しています。このソリューションは、ADXL345 のハードウェア割込みを有効に活用するもので、簡単なソフトウェアで実装でき、高い検出精度を備えていることがテストされています。

参考資料

ADXL345 のデータシート（アナログ・デバイセズ、2009 年）

AN08185-0-6/09(0)-J