アプリケーション ノート AN-80 BridgeSwitch ファミリー

BridgeSwitch FAULT 通信インターフェイス

power integrations™

はじめに

このアプリケーション ノートでは、BridgeSwitch™ FAULT ステータス通信インターフェイス機能のソフトウェアの実装について説明します。具体的には、BridgeSwitch FAULT ステータス通信インターフェイス、受信したステータス アップデートを取り込んで処理するステート マシン、リファレンス コードとそのデータ構造について説明します。また、UART ターミナルでステータス アップデートを表示してソフトウェアをデモンストレーションし、インバータ ボードでの回路保護の実装例も示します。

BridgeSwitch FAULT ステータス通信インターフェイス

BridgeSwitch デバイスは、内部及びシステム レベルの異常を含むステータス アップデートを、オープン ドレインの FAULT ピンを介してシステム MCU に通信できます。通信には、7 ビットのワード パターンの後に奇数パリティ ビットを使用して、ステータス アップデートをレポートします。

以降のセクションでは FAULT バスの仕様について詳しく説明します。

ハードウェア構成

すべての検出されたステータス アップデートをシステム マイクロコントローラに伝達するため、すべての FAULT ピンがシングルワイヤ バスに接続され、システム供給電圧にプルアップされます。図 1 は、3 つの BridgeSwitch デバイスがシングルワイヤ バス構成でシステム MCU に接続される、一般的なインターフェイスを示しています。

デバイス ID ピン接続では、各デバイスが、そのIDピンの接続に応じてデバイス特定のデバイス ID を割り当てることができます。このデバイス ID を使用し、ステータス通信の開始時に対応するデバイス ID 期間 t_{ID} の間、FAULT バスをプルダウンして、異常状態が検出された物理的な場所をシステム マイクロコントローラに伝達します。

テーブル 1 は、デバイス ID、デバイス IDの期間 t_{ID}、及び ID ピン接続を介してそれぞれのIDをプログラムする方法を示します。

デバイス ID	t_{ID}	ID ピン接続
1	40 µs	BPL ピン
2	60 µs	フローティング
3	80 µs	SG ピン

テーブル 1: ID ピンによるデバイス ID 選択

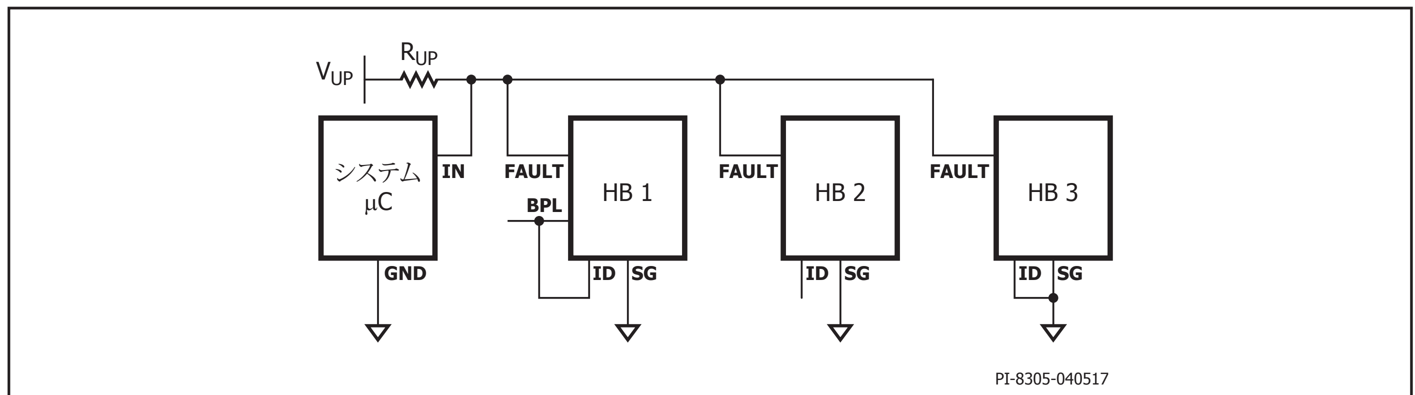

図 1: シングルワイヤ ステータス通信バス 及び デバイスID プログラミング

FAULT ステータス通信バスの仕様

ステータス エンコーディング
7 ビットのワードの後にパリティ ビットを続けて、異常情報をエンコーディングします。テーブル 2 は、デバイスがシステム マイクロコントローラに伝える可能性がある、様々なステータス アップデートのエンコーディングをまとめたものです。ステータス ワードは 5 個のブロックで構成され、同時に発生することができないステータス変更がグループ化されています。そのため、異常の優先順位や異常の報告順番を考慮せずに、複数のステータス アップデートをシステム マイクロコントローラに同時にレポートできます。

最後の行 (7 ビット ワード "000 00 0 0") はデバイスの準備が整っているというステータスをエンコードするもので、起動シーケンスが正常に行われた時、特定の異常が解除された時、異常が存在しない場合にシステム MCUにステータス要求を確認する時に送信されます。

FAULT バスに関する通信は、次のいずれかの理由で開始されます。

- 正常起動の後、ミッション モード通信の準備が整っています。
- いずれかのデバイスによって、FAULT ステータスレジスタのアップデート通信が開始されました。
- システム マイクロコントローラによる問い合わせに応じて、現在のステータスを通信します。

ステータス	ビット **0**	ビット **1**	ビット **2**	ビット **3**	ビット **4**	ビット **5**	ビット **6**
HV バス OV	0	0	1				
HV バス UV 100%	0	1	0				
HV バス UV 85%	0	1	1				
HV バス UV 70%	1	0	0				
HV バス UV 55%	1	0	1				
システムの過熱異常	1	1	0				
LS ドライバ未準備[1]	1	1	1				
LS FET 過熱警告				0	0		
LS FET 過熱シャットダウン				1	0		
HS ドライバ未準備[2]				1	1		
LS FET 過電流						1	
HS FET 過電流							1
デバイス未準完了 (異常なし)	0	0	0	0	0	0	0

注:
1. XL ピンのオープン/ショート異常、IPH ピンと XL ピンのショート、及びトリム ビットの破損が含まれます。
2. HS と LS 間の通信切断、範囲外の V_{BPH} または内部 5 V、及び XH ピンのオープン/ショート異常が含まれます。

テーブル 2. ステータスワードのエンコーディング

図 2 は、これら 3 つの場合すべてに対応する、ステータス通信のフローチャートを示しています。

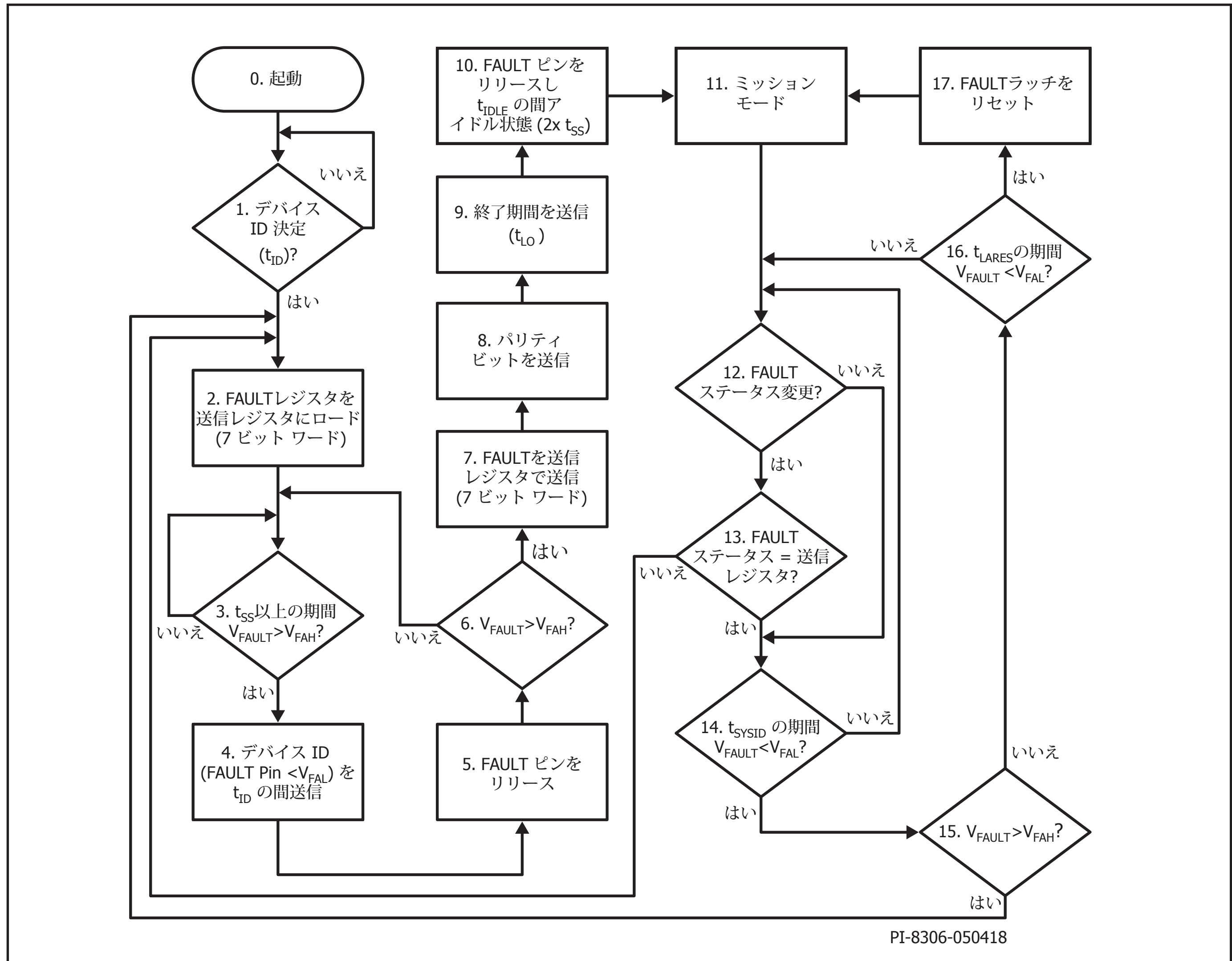

図 2.　ステータス通信のフローチャート

ステータス アップデート通信は常に、通信デバイスによって開始されたバス調停から始まります。バスが 80 µs 以上静止している場合に、FAULT ピンをプルダウンして、対応するデバイス ID 期間 t_{ID} を送信します。通信フローチャート (図 2) に示すように、デバイスがバス調停を獲得した後、現在の FAULT レジスタ (7 ビット ワード) の後に奇数パリティ ビットと送信終了信号を付加して送信されます。

ビット ストリーム タイミング

図 3 は、BridgeSwitch がステータス アップデート通信に使用する、ビット ストリーム タイミングを示しています。FAULT ピンでの異なる期間のHighレベル、及びその後のLowレベル(Typ. 10 µs)の2つの論理状態によってエンコードされます。Highレベルの期間がt_{BIT1}(Typ. 40 µs)の場合は論理値「1」がエンコードされ、Highレベルの期間がt_{BIT0}(Typ. 10 µs)の場合は論理値「0」がエンコードされます。テーブル3は、ビットストリームのタイミングテーブルを示しています。

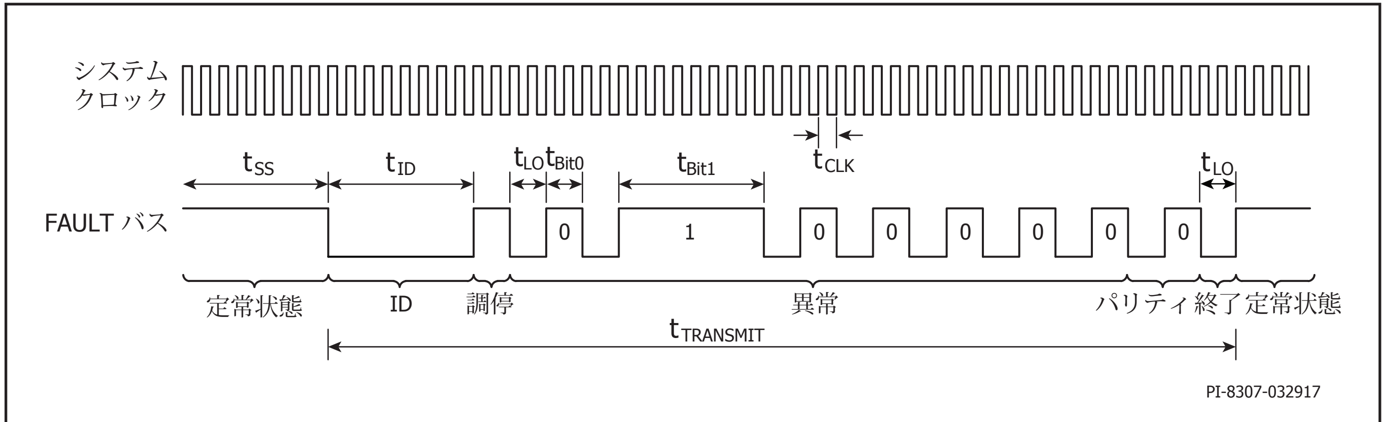

図 3.　ステータス通信のビットストリーム

記号	概要	ロジック状態	期間 (一般的)
t_{ID}	デバイス ID	0	テーブル 1 を参照
t_{LO}	ロータイム	0	10 µs
t_{Bit0}	ロジック 0	1	10 µs
t_{Bit1}	ロジック 1	1	40 µs

テーブル 3. ビットストリームのタイミングテーブル

送信が完了するたびに、t_{IDLE} (Typ. 2 × t_{SS} = 160 µs) の間アイドル状態になり、その後、新しい通信が開始されます。これにより、バス上の他のデバイスは、発生したステータス変更を送信したり、システム マイクロコントローラが送信したステータスの問い合わせに応答することができます。

デバイスは検出されたステータス アップデートごとに 1 回だけ通信します。また、すべてのシステム レベル異常のステータス変更をシステム MCU にレポートします。これには、DC バスの低電圧状態と過電圧状態や、外付けの温度監視における異常が含まれます。さらに、LS パワー FREDFETの過熱保護を除いて、デバイスの内部異常に関するすべてのステータス レベル変更もレポートします。

ミッションモードになったBridgeSwitchデバイスは、FAULT バスを監視して、システムマイクロコントローラから送信される有効なコマンドを検出します。たとえば、マイクロコントローラがバスを t_{SYSID} (Typ. 160 µs) の間プルダウンしてステータス アップデートを問い合わせることがあります (図 2 のステップ 15 を参照)。または、過熱シャットダウン ラッチなどのデバイス ステータス レジスタをリセットし、FAULT バスを期間 t_{LARES} (Typ. 2 × T_{SYSID} = 320 µs) の間プルダウンして、起動シーケンスモードに入る (図 2 のステップ 17 を参照) コマンドの場合もあります。MCU がラッチのリセット コマンドを送信した後は、起動シーケンスを行うことを推奨します。テーブル 4 は、使用可能なシステム マイクロコントローラのコマンドを示しています。

バス プルダウン期間	コマンド
t_{SYSID}	ステータス問い合わせ
$t_{LARES(2xTSYSID)}$	過熱シャットダウンのラッチリセットや起動シーケンス モードを含むステータス レジスタ

テーブル 4. システム MCU のコマンド

ソフトウェア実装

ここでは、前のセクションで説明したステータス通信仕様に基づいて各 BridgeSwitch デバイスからステータス アップデートを取り込んで処理する、ステート マシンの実装について説明します。

ここに示す例では、割り込みベースの実装を使用しています。ユーザーは、モーター制御アルゴリズムやマイクロコントローラ タイプなどの特定のアプリケーション要件に基づいて、割り込みの優先順位を決める必要があります。

システム MCU の周辺機器

FAULT バス通信インターフェイスの動作を確認するため、Cypress PSoC Creator IDE Version 4.1 を使用してリファレンス コードを作成し、Cypress PSoC 4 MCU (CY8CKIT-042 PSoC Pioneer Kit) でテストしました。この MCU ボードでは、USB コネクターを介して PC に通信する、オンボード プログラマー及びデバッガーが提供されます。受信したステータス アップデートを異常検出例のセクションで説明している UART コンソール上に出力して、ステート マシンの動作チェックを行いました。

FAULT ステータス通信バスは、オープン ドレイン ドライブ モードで、単一の双方向 MCU ピンに接続されます。このピンは、信号の立ち上がりエッジと立ち下がりエッジの両方を取り込むタイマーに接続されています。異常シグナルは、2 個の 16 ビット タイマー/カウンター ブロック (Bit_counter_timer 及び ID_counter_timer) を 12 MHz クロックとともに使用する割り込みベースで処理されます。Bit_counter_timer は立ち上がりエッジから立ち下りエッジで信号を取り込み、ID_counter_timer は立ち下りエッジから立ち上がりエッジで信号を取り込みます。これら 2 つのタイマーはカウント値を取り込み、異常信号の立ち下りエッジと立ち上がりエッジをそれぞれ受信すると、割り込みを生成します。FAULT ステート マシン ルーチンは受信した割り込みを処理します。

ソフトウェアの説明

ソフトウェアの動作では、最初に fault_bus_state 変数をアイドル状態 (STEADY_STATE) に初期化します。この fault_bus_state 変数は、割り込みを受信すると異常信号の状態を取り込みます。初期化関数 init_fault_bus_interrupt() は、タイマー/カウンター取り込みポートを初期化し、立ち上がりエッジと立ち下りエッジの両方に対して取り込みの割り込みを有効にします。割り込みサービス ルーチン (ISR) がトリガーされるたびに、fault_detect() 関数が呼び出されます。この関数は FAULT ステート マシン ルーチンで、受信した異常を取り込んで処理します。主なソフトウェア フローの概要を図 5 に示しています。

FAULT ステート マシン ルーチンは、基本的に ISR イベントを処理し、異常処理の現在の状態に基づいて fault_bus_state 変数を更新します。FAULT マシン状態は STEADY_STATE、ID_DET、ARBITRATION、T_LO、及び BIT_DETECT です。FAULT ステート マシンの詳細なソフトウェア フローを、図 6 に示しています。FAULT ステータスの完全なパケットをパリティ エラーなしで受信した場合は、異常処理関数 fault_process() が呼び出されます。そうでない場合は再同期が実行され、fault_bus_state が STEADY_STATE にリセットされます。

この異常処理関数は受信した FAULT ステータスアップデートをデコードし、必要なアクションを呼び出します。たとえば、過電流異常の受信後にインバータをシャットダウンしたり、過熱警告ステータス アップデートの受信後にインバータの出力電力を下げたりします。FAULT ステータスは fault 変数に格納され、この変数は新しいステータス アップデートを受信するたびに更新されます。ユーザーは、受信した FAULT ステータス及びアプリケーション要件に応じて、必要なアクションを実行するか、MCU で実行する処理を決定する必要があります。テーブル 5 は、ステータス アップデートの受信後にマイクロコントローラが実行できるアクションの例を示しています。fault_process() 関数のソフトウェア フロー図を、図 7 に示しています。

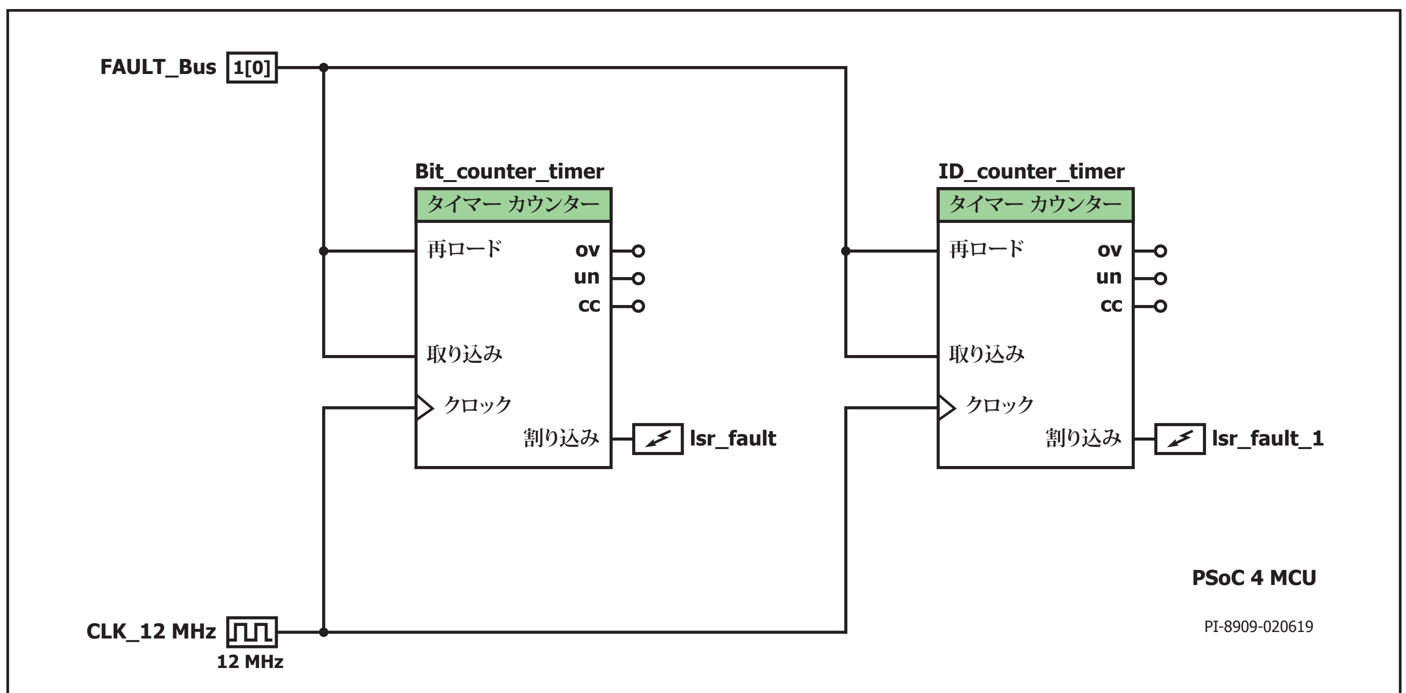

図 4. PSoC 4 MCU を使用した異常信号処理例用のシステムMCUペリフェラル

ステータス問い合わせ及びラッチ リセット コマンド

ステータス問い合わせコマンド及びラッチ リセット コマンドのソフトウェア動作では、次の使用事例に対応します。

ステータス問い合わせコマンド

MCU は、インバータが一定期間オフになった後にインバータのリスタート（PWM信号の送信）が必要になるたびに、ステータス問い合わせを送信することがあります。たとえば、過入力電圧または過電流の異常がレポートされた後などです。ステータス問い合わせの主な目的は、すべてのデバイスの準備が整っているか、または MCU が起動シーケンスを開始する必要があるかを確認することです。図 8 は、ステータス問い合わせの実装例のフローチャートです。

ステータス問い合わせルーチンでは、各 BridgeSwitch デバイスのステータスを確認して、次のどの状態であるかを判断します。

A. 各デバイスが READY ステータス (異常なし) で応答します。MCU は RESTART コマンドを呼び出してインバータをリスタートし、PWM 信号を送信して BridgeSwitch の入力を制御します。
B. 1 つ以上のデバイスが、ハイサイド ドライバの準備が整っていない異常を応答します。MCU は START UP コマンドを呼び出します。これによって起動シーケンスが開始され、ハイサイド ドライバの供給電圧 (HP ピンについては V_{BPH}) が定常レベルに充電されます。起動シーケンスの完了後、MCU は他のステータス問い合わせコマンドに対応します。すべてのデバイスが READY ステータスで応答した場合、MCU は RESTART コマンドを呼び出してインバータをリスタートします。いずれかのデバイスが READY 以外のステータスで応答した場合、インバータはシャットダウン モードのままになります。

ステータス問い合わせコマンドは status_query() 関数によって処理され、FAULT バスが t_{SYSID} = 160 μs の間 (テーブル 4 を参照) プルダウンされます。各デバイスがこのコマンドをフォローして、それぞれのステータスを逐次送信します。システム マイクロコントローラによってステータス問い合わせが送信された後、検出された FAULT ステータスが process_status_query_command() 関数 (FAULT バス ステート マシン関数内に置かれています) によって処理されます。この関数は検出された各デバイス ステータスを格納し、それらを status_query_action() 関数で処理します。status_query_action() 関数は受信した FAULT ステータスを確認し、テーブル 5 に示した条件に基づいてアクションを提供します。

ラッチ リセット コマンド

MCU は、いずれか (またはすべて) のデバイスが過熱異常 (及びラッチ オフ) をレポートした後しばらくしてから、ラッチ リセット コマンドを送信できます。図 9 は、ラッチ リセット コマンド例のフロー チャートです。ラッチ リセット コマンド後は起動シーケンスが推奨されます。それにより、バイパス ハイサイド電圧が通常のレベルになってから、スイッチングが再開されます。

ラッチ リセット コマンドは latch_reset() 関数によって処理され、FAULT バスが t_{LARES} = 320 μs (テーブル 4 を参照) の間プルダウンされて、各デバイス ステータスがリセットされます。システム マイクロコントローラによってラッチ リセット コマンドが送信された後、起動シーケンスが呼び出されます。ラッチ リセット コマンドが送信されるたびに、FAULT 検出関数が無効化される必要があります。

異常	ステータス ワード	ソフトウェア アクション/決定	注
高電圧バス OV	001 xxx x	シャットダウン	一般的には、1 つのデバイスのみが HV バスを監視し、MCU がインバータ全体をシャットダウンします。
高電圧バス UV 100%	010 xxx x	–	MCU はモーター出力を公称電力まで高めることができます (前のステータス アップデートが UV85、UV70、または UV50 であった場合)。
高電圧バス UV 85%	011 xxx x	警告	MCU はモーター出力電力を下げて (速度/トルク) インバータの負荷を軽減できます。
高電圧バス UV 70%	100 xxx x	警告	MCU はモーター出力電力を下げて (速度/トルク) インバータの負荷を軽減できます。
高電圧バス UV 55%	101 xxx x	警告	MCU はモーター出力電力を下げて (速度/トルク) インバータの負荷を軽減できます。
システムの過熱異常	110 xxx x	警告/シャットダウン	温度監視対象となっている外付け部品によって異なります。
LS ドライバ未準備	111 xxx x	シャットダウン	MCU はしばらくしてからインバータのリスタートを試行して、異常が解消されたかどうかを確認できます。
LS FET 過熱警告	xxx 010 x	警告	MCU はモーター出力電力を下げて (速度/トルク) インバータの負荷を軽減するか PCB 温度を制限できます。
LS FET 過熱シャットダウン	xxx 10x x	シャットダウン	ラッチのシャットダウンは 1 つのデバイスでのみ発生することがありますが、MCU はインバータ全体をシャットダウンする必要があります。MCU は、放熱期間の後にインバータのリスタートを試行できます。
LS FET 過電流	xxx xx1 x	シャットダウン	デバイスは対応する FREDFET を自動的にオフにして、モーターがストール状態や過負荷状態にならないようにします。MCU はインバータ全体をシャットダウンします。
HS ドライバ未準備	xxx 11x x	シャットダウン	MCU はしばらくしてから (数秒後) インバータのリスタートを試行して、異常が解消されたかどうかを確認できます。
HS FET 過電流	xxx xxx 1	シャットダウン	デバイスは対応する FREDFET を自動的にオフにして、モーターがストール状態や過負荷状態にならないようにします。MCU はインバータ全体をシャットダウンします。
デバイス準備済 (異常なし)	000 000 0	–	MCU は、前のステータス アップデートが HV バス OV または LS/HS FET 過電流であった後に、インバータをリスタートできます。また、前のステータス アップデートが過熱警告であった場合は、モーターを公称電力まで高めることもできます。

テーブル 5. ステータス アップデートの受信後にマイクロコントローラが実行するアクションの例

ソフトウェア フローチャート

次のフローチャートは、異常信号の処理のためのソフトウェアの概要を示しています。

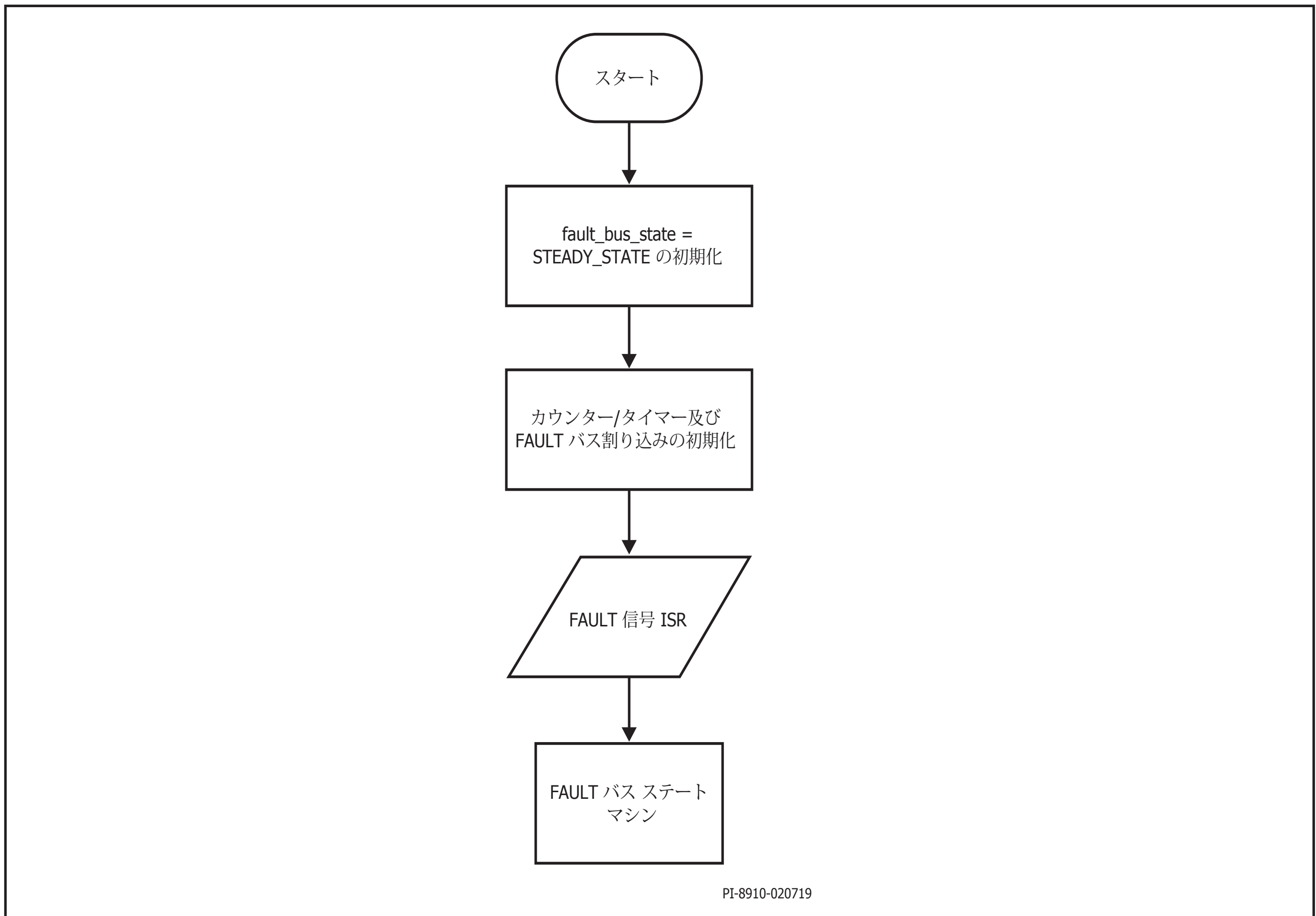

図 5.　ソフトウェアでの FAULT バス実装の概要

FAULT バス ステート マシン

図 6 は、FAULT バス ステート マシンのソフトウェア フローチャートを示しています。

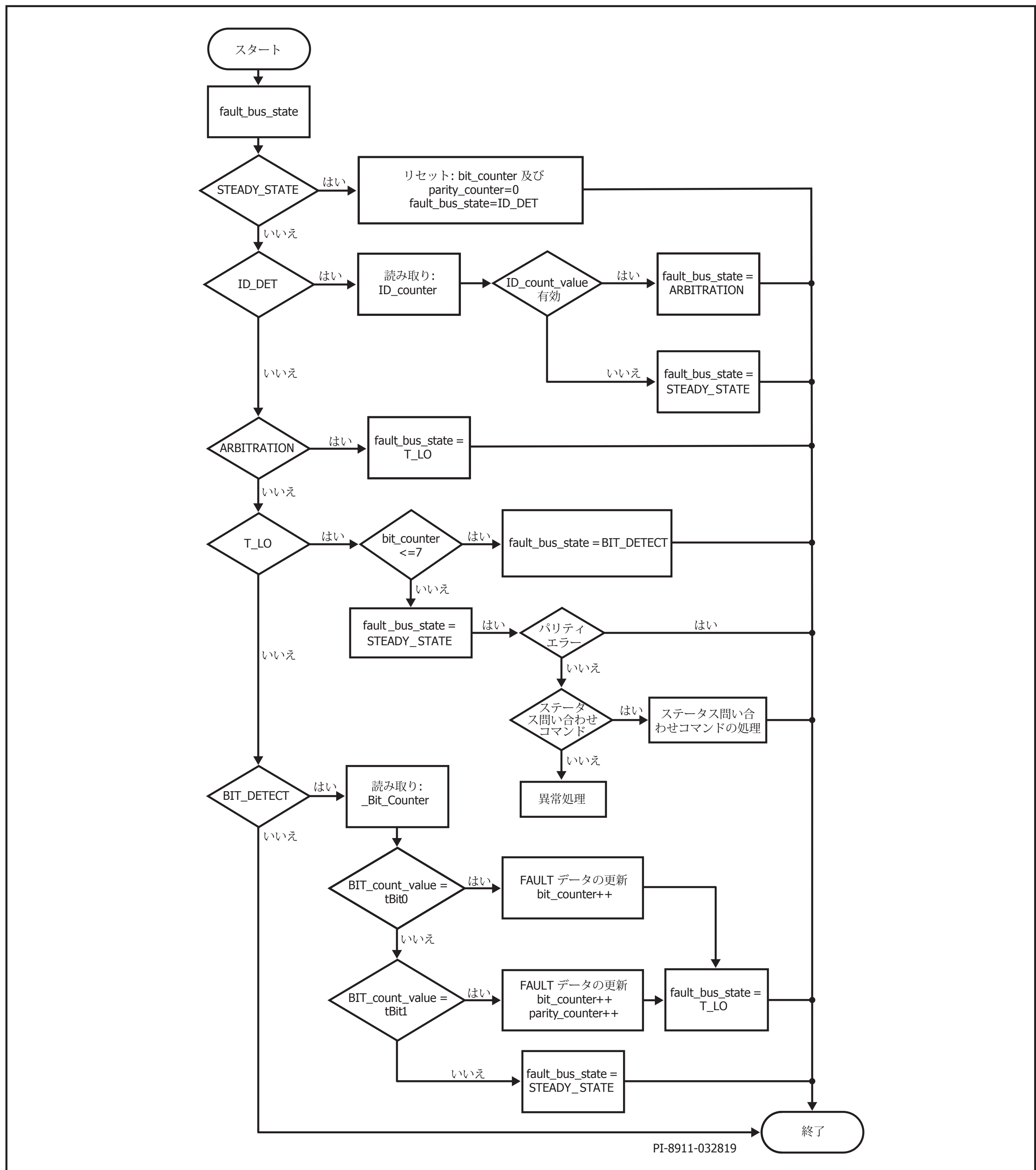

図 6. FAULT バス ステートマシン

図 7 は、異常処理関数のソフトウェア フローチャートを示しています。

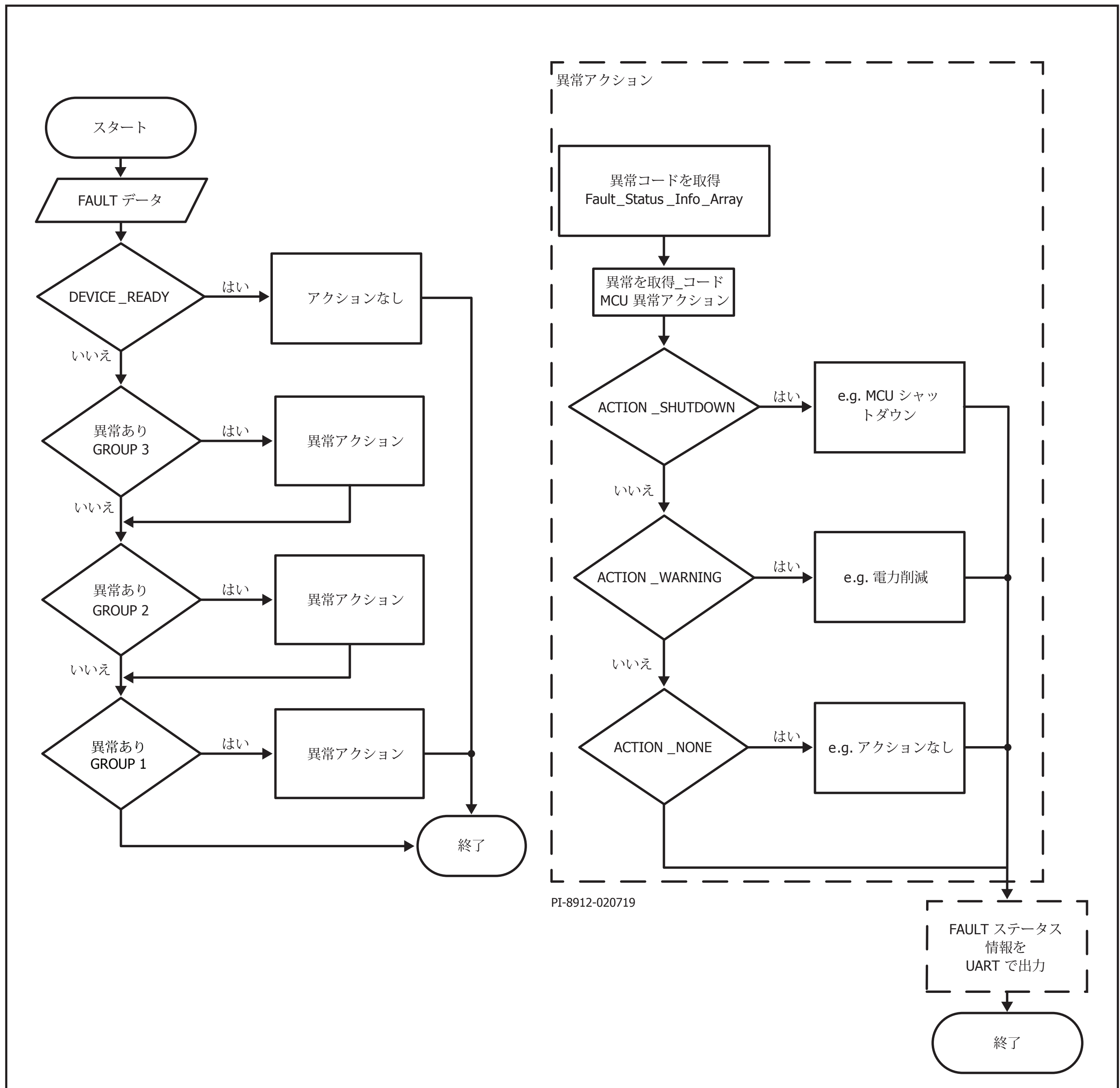

図 7. 異常処理関数

受信した異常データには複数のタイプのステータス アップデートが含まれている場合があり、異常処理関数で各異常タイプを処理できる必要があります。同時に発生できない異常ビットは、異常タイプを判断するためにグループ化されます。テーブル 6 はステータス ワードのグループを示しています。GROUP1、GROUP2、ローサイド FET 過電流、及びハイサイド FET 過電流の異常は、単一のステータス ワード内で同時にレポートできます。

グループ	異常	ビット 0	ビット 1	ビット 2	ビット 3	ビット 4	ビット 5	ビット 6
GROUP1	HV バス OV	0	0	1				
	HV バス UV 100%	0	1	0				
	HV バス UV 85%	0	1	1				
	HV バス UV 70%	1	0	0				
	HV バス UV 55%	1	0	1				
	システムの過熱異常	1	1	0				
	S ドライバ未準備[1]	1	1	1				
GROUP2	LS FET 過熱警告				0	0		
	LS FET 過熱シャットダウン				1	0		
	HS ドライバ未準備[2]				1	1		
LS FET 過電流							1	
HS FET 過電流								1

テーブル 6. FAULT ステータスのグループ

この実装例では、レポートされた異常タイプごとに異常アクション関数が呼び出されます。異常アクション関数は異常コードに対応する異常アクションをFAULT_STATUS_INFO_ARRAY から選択し、その後で MCU がそれを実行します。考えられるアクションについては、テーブル 5 を参照してください。レポートされたステータス アップデートに基づくアクションは、特定のアプリケーション要件に応じて調整する必要があります。

本書の異常検出例の段落では、UART コンソールで FAULT ステータスを表示することで、ステータス アップデートのデコードをデモンストレーションしています。この段落では MCU が 3 相インバータ ボードで実行するアクションも示しています。

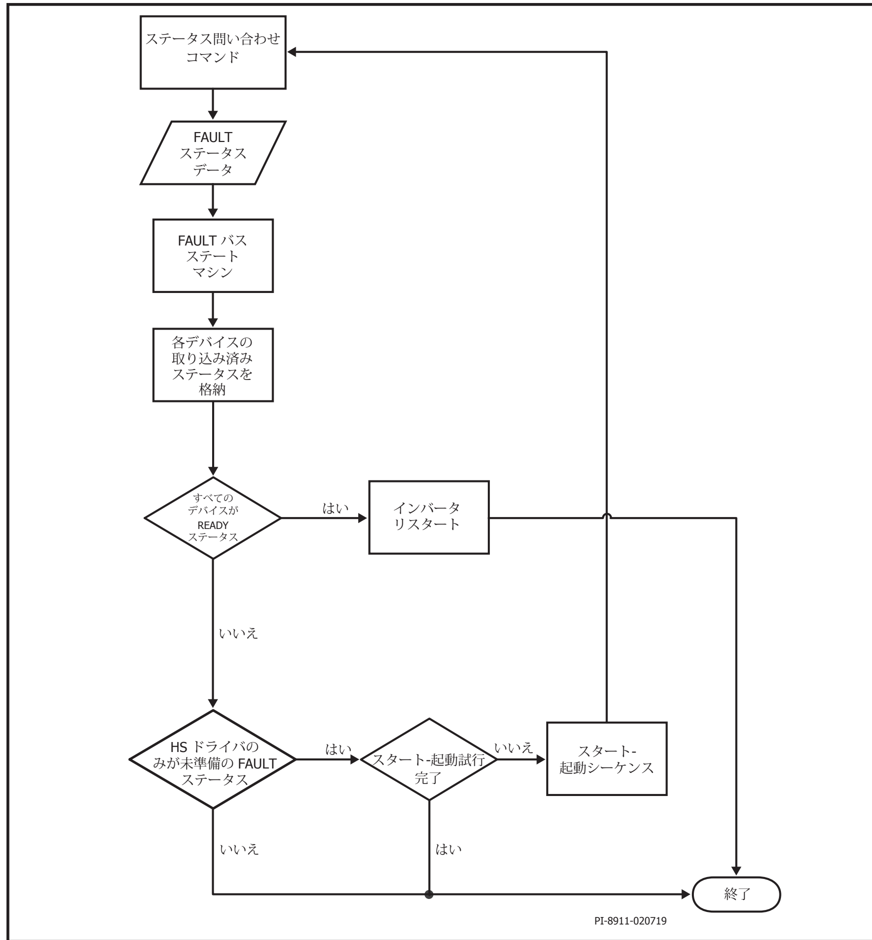

図 8.　ステータス問い合わせコマンドの処理関数

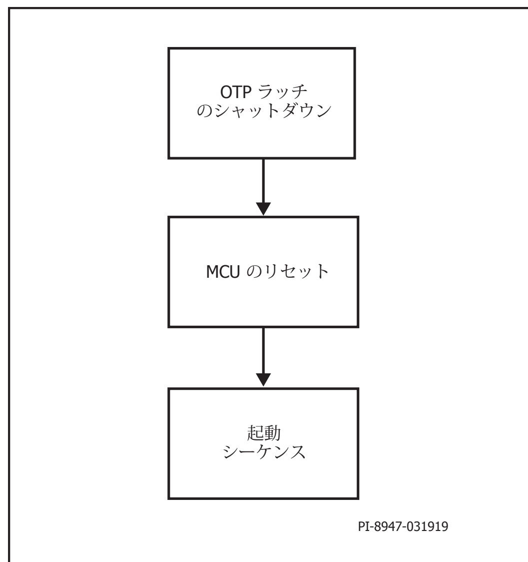

図 9.　ラッチ リセット コマンド関数

リファレンス コードのデータ構造

FAULT ステート マシンの状態
次に示すのは、検出プロセス中に現在の異常信号状態を決定する、FAULT バス状態です。

```
STEADY_STATE =0,
ID_DET,
ARBITRATION,
T_LO,
BIT_DETECT,
```

Fault_Status_Info_Array
この FAULT_STATUS_INFO アレイは、デコードされた FAULT ステータスの更新後のアクションのリストです。

```
{HV_BUS_OV, ACTION_SHUTDOWN},
{HV_BUS_UV_100, ACTION_NONE},
{HV_BUS_UV_85, ACTION_WARNING},
{HV_BUS_UV_70, ACTION_WARNING},
{HV_BUS_UV_55, ACTION_WARNING},
{SYSTEM_THERMAL_FAULT, ACTION_SHUTDOWN},
{LS_DRIVER_FAULT, ACTION_SHUTDOWN},
{LS_FET_THERMAL_WARNING, ACTION_WARNING},
{LS_FET_THERMAL_SHUTDOWN, ACTION_SHUTDOWN},
{HS_DRIVER_FAULT, ACTION_SHUTDOWN},
{LS_FET_OVERCURRENT, ACTION_SHUTDOWN},
{HS_FET_OVERCURRENT, ACTION_SHUTDOWN},
```

たとえば、過電圧のエントリ {HV_BUS_OV, ACTION_SHUTDOWN} は、このエラーが発生した場合に MCU がシステムをシャットダウンする必要があることを示します。

この実装での、FAULT ステータスの更新後のアクションは次のとおりです。
ACTION_SHUTDOWN,
ACTION_WARNING,
ACTION_NONE,

FAULT ステータス コード
発生する可能性のある異常状態は次のとおりです。

```
//GROUP1 FAULTS
HV_BUS_OV = 4u,
HV_BUS_UV_100 = 2u,
HV_BUS_UV_85 = 6u,
HV_BUS_UV_70 = 1u,
HV_BUS_UV_55 = 5u,
SYSTEM_THERMAL_FAULT = 3u,
LS_DRIVER_FAULT = 7u,

//GROUP2 FAULTS
LS_FET_THERMAL_WARNING = 16u,
LS_FET_THERMAL_SHUTDOWN = 8u,
HS_DRIVER_FAULT = 24u,

//LS FET OVERCURRENT
LS_FET_OVERCURRENT = 32u,

//HS FET OVERCURRENT
HS_FET_OVERCURRENT = 64u,

//FAULT CLEAR
DEVICE_READY = 128u,
```

FAULT_STRUCT
この構造には、発生している異常の FAULT コードとデバイス ID が含まれています。

dev_id
fault

power integrations™
www.power.com

リファレンス コード

このコード例は、PSoC Creator IDE Version 4.1 を使用して作成され、DER-654 設計例インバータ ボードを使用して CY8CKIT-042 PSoC Pioneer Kit デバイス上でテストされました。以下のコードは異常信号処理と関連するリファレンス関数を表しています。ここに示されているリファレンス コードには、UART コンソールで FAULT ステータス情報を出力するコード スニペットは含まれていません (詳細については注釈の段落を参照)。使用されているその他の変数の定義については、提供されているコード ファイルを参照してください。

```
/* ============================================================================
 * THE SOFTWARE INCLUDED IN THIS FILE IS FOR GUIDANCE ONLY.
 * Power Integrations SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR
 * CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM USE OF THIS
 * SOFTWARE.
 * ==========================================================================*/

/******************************************************************************
* Function Name: void fault_detect(void)
*******************************************************************************
*
* Summary:
* This function is the state machine for the fault bus.
*
* Parameters: None
*
* Return: None
*****************************************************************************/

void fault_detect(void)
{

 switch(fault_bus_state)
   {
  case STEADY_STATE:  bit_counter = 0;
                      parity_counter = 0;
                      /* change state to ID detect */
                      fault_bus_state = ID_DET;
                      break;

  case ID_DET:        /* change state to ARBITRATION */
                      fault_bus_state = ARBITRATION;
                      /*Read ID_counter_timer capture value */
                      ID_count_value = Read_ID_Counter;

   if((ID_count_value >= ID_40uS_MIN)&&(ID_count_value <= ID_40us_MAX))
                            {
                       //Device 1
                       fault_struct.dev_id = DEVICE_ID_1; }

   else if((ID_count_value >= ID_60uS_MIN)&&(ID_count_value <= ID_60us_MAX))
                            {
                       //Device 2
                       fault_struct.dev_id = DEVICE_ID_2; }
```

```
  else if((ID_count_value >= ID_80uS_MIN)&&(ID_count_value <= ID_80us_MAX))
                           {
                             //Device 3
                             fault_struct.dev_id = DEVICE_ID_3; }


  else {
                             //Re-synchronize fault detection if
                             //invalid ID was received
                            fault_bus_state = STEADY_STATE; }

                       break;

case ARBITRATION:      /* change state to T_LO */

                       fault_bus_state = T_LO;

                       break;

case T_LO:          if(bit_counter <= 7)
                           {
                       /* change state to BIT_DETECT*/
                       fault_bus_state = BIT_DETECT;   }

                       else
                           {
                       /* change state to STEADY_STATE */
                       fault_bus_state = STEADY_STATE;

                       if(!(parity_counter & 1))
                            {

                             //Parity Error
                            }

                        else

                             //Process fault
                            process_fault();
                               }

                           }
                           break;
```


power integrations™
www.power.com

```
case BIT_DETECT:  /* Read Bit_counter_timer capture value*/
                  BIT_count_value = Read_Bit_Counter;

    if((BIT_count_value >= T_BIT0_MIN) && (BIT_count_value <= T_BIT0_MAX))
                        {
               /* change state to T_LO*/
               fault_bus_state = T_LO;

               //update fault status variable
               fault_struct.fault = fault_struct.fault & ~(1 << bit_counter);
               bit_counter++;
                        }
    else if((BIT_count_value >= T_BIT1_MIN)&&(BIT_count_value <= T_BIT1_MAX))
                        {
               /* change state to T_LO*/
               fault_bus_state = T_LO;

               // update fault status variable
               fault_struct.fault = fault_struct.fault | (1 << bit_counter);
               parity_counter++;
               bit_counter++;
                         }
    else      {
              //Re-synchronize fault detection when invalid BIT was received
              fault_bus_state = STEADY_STATE;
                         }

                    break;


        default:
                    break;

    }

}

/**************************end of function ********************************/
```

1.

```
/*****************************************************************************
* Function Name: void process_fault(void)
*****************************************************************************
*
* Summary:
* This function is to process fault after receiving it.
*
* Parameters: None
*
* Return: None
*
*****************************************************************************/
void process_fault(void){

    /*If the received fault is DEVICE_READY*/
    if(fault_struct.fault == DEVICE_READY){

    //user own implementation
    }

    else{

    /*Low-side FET Overcurrent*/
    if((fault_struct.fault & BIT5) != 0){
    tfault = (fault_struct.fault & BIT5);
    action_fault(tfault);
    }

    /*High-side FET Overcurrent*/
    if((fault_struct.fault & BIT6) != 0){
    tfault = (fault_struct.fault & BIT6);
    action_fault(tfault);
    }

    /*Group1 Faults*/
    if((fault_struct.fault & GROUP1) != 0){
    tfault = (fault_struct.fault & GROUP1);
    action_fault(tfault);
    }

    /*Group2 Faults*/
    if((fault_struct.fault & GROUP2) != 0){
    tfault = (fault_struct.fault & GROUP2);
    action_fault(tfault);
    }


   }

 }



/*****************************end of function*****************************/
```

```
/***************************************************************************
*
* Function Name: void fault_action(unit8)
****************************************************************************
*
* Summary:
* This function is to command an action after a fault is received
*
* Parameters: masked fault by group
*
* Return: None
*
***************************************************************************/

void action_fault(uint8 tfault){

/*Look the fault code into the fault_status_info_arr array and the
corresponding MCU action*/

   int loop_count = sizeof(fault_status_info_arr)/sizeof(FAULT_STATUS_INFO);
   for (int i=0; i<=loop_count; i++){

        if(tfault != (fault_status_info_arr[i].fault_code))
        continue;

        switch(fault_status_info_arr[i].fault_action){

                  case ACTION_NONE:
                  /* do nothing */
                  break;

                  case ACTION_WARNING:
                  /* user own implementation */
                  break;

                  case ACTION_SHUTDOWN:
                  /* Shutdown MCU */
                  break;

        }


        /**OPTIONAL -print fault information for debugging purposes only**/
        print_fault_info(tfault);

  }

}


/*****************************end of function*****************************/
```

以下のコードは、本書で説明しているステータス問い合わせコマンド及びラッチ リセット コマンドの実装例に関連するリファレンス関数を表しています。ステータス問い合わせコマンド及びラッチ リセット コマンドの呼び出しは、実際の実装では、各ユーザーの使用事例に応じて個別に処理する必要があります。使用されている変数の定義については、提供されているコード ファイルを参照してください。

```
/***************************************************************************
***
* Function Name: void status_query(void)
****************************************************************************
***
*
* Summary:
* This function is to command a status query
*
* Parameters: None
*
* Return: None
*
****************************************************************************
**/

void status_query(void){


    /*Clear FAULT Bus ISRs*/
    FAULT_Bus_ClearInterrupt();

    /*Pull down the FAULT Bus for 160 uS*/
    FAULT_Bus_Write(0);
    CyDelayUs(160);

    FAULT_Bus_Write(1);

    /*Enable FAULT_Bus ISRs*/
    init_fault_bus_interrupt();

    /*Set status query flag*/
    status_query_state = TRUE;

}
```


power integrations™
www.power.com

```
/**********************************************************************
***
* Function Name: void process_status_query_command(void)
**********************************************************************
***
*
* Summary:
* This function is to process the status query command
*
* Parameters: None
*
* Return: None
*
**********************************************************************
**/

void process_status_query_command(){

    //store each devices fault status
    device_fault_arr[fault_struct.dev_id] = fault_struct.fault;

    //increment device_counter
    device_counter++;

    if(device_counter == DEVICE_COUNT){

    //status_query_action
    status_query_action();

    //reset status query state
    status_query_state = FALSE;

    //reset device counter
    device_counter = 0;

        }

}
```

```
/***********************************************************************
***
* Function Name: void status_query_action(void)
***********************************************************************
***
*
* Summary:
* This function is to process the captured fault status from a status query
* command
* Parameters: None
*
* Return: None
*
***********************************************************************
**/
void status_query_action(void){

    //Function that checks if all devices are READY
    if (device_ready_check()){

    /*All devices are READY, Inverter restart function should be placed here
    *
    */ }

    //Function that checks for only HS driver not ready fault
    else if(hs_driver_not_ready_check()){

    //Command a startup sequence after the first status query command
    if(startup_flag == FALSE){

    /*Startup sequence function should be placed here
    *
    */

    /*Check the status if HS not ready fault/s is/are cleared*/
    status_query();

    //Assert startup_flag after start up sequence
    startup_flag = TRUE;
     }
    else{

    //HS driver not ready fault still exists

    //De-assert startup_flag
    startup_flag = FALSE;

        }
      }
    else{

    //Other faults are present
    startup_flag = FALSE;
       }
      }
```


power integrations™
www.power.com

```
/*********************************************************************
***
* Function Name: boolean device_ready_check(void)
*********************************************************************
***
*
* Summary:
* This function is to check if all devices are ready
*
* Parameters: None
*
* Return: boolean
*
*********************************************************************
**/

boolean device_ready_check(void){

  uint8 tfault_status =0;

  //Check if all devices are READY
  for(uint8 i=0; i<sizeof(device_fault_arr); i++){

   tfault_status  |= device_fault_arr[i];

  }

  //If all devices are READY
  if(tfault_status == DEVICE_READY){

    //return TRUE
    return TRUE;

    }else{

    //return FALSE
    return FALSE;
    }


}
```

```
/**********************************************************************
***
* Function Name: boolean hs_driver_not_ready_check(void)
**********************************************************************
***
*
* Summary:
* This function is to check if all devices are READY
*
* Parameters: None
*
* Return: boolean
*
**********************************************************************
**/

boolean hs_driver_not_ready_check(void){

 //Default hs_driver_fault_flag
 hs_fault_flag = FALSE;

 for(uint8 i=0; i<sizeof(device_fault_arr); i++){

    if((device_fault_arr[i] == DEVICE_READY) || (device_fault_arr[i] ==
HS_DRIVER_NOT_READY_FAULT)){

        if(device_fault_arr[i] == HS_DRIVER_NOT_READY_FAULT){

        //Assert hs_driver_fault flag
        hs_fault_flag = TRUE;

        continue;
        }

    }else{

        //Other fault/s is/are present
        return FALSE;
        }

    }

    return hs_fault_flag;

}
```

```
/*****************************************************************************
***
* Function Name: void latch_reset(void)
*****************************************************************************
***
*
* Summary:
* This function is to command latch reset
*
* Parameters: None
*
* Return: None
*
*****************************************************************************
**/
void latch_reset(void){

    /*Disable FAULT Bus ISRs*/
    FAULT_Bus_ClearInterrupt();

    /*Pull down the FAULT Bus for 320 uS*/
    FAULT_Bus_Write(0);
    CyDelayUs(320);

    FAULT_Bus_Write(1);
    }
/*****************************************************************************
***
* Function Name: void mcu_latch_reset(void)
*****************************************************************************
***
*
* Summary:
* This function is to command latch_reset followed by a power up sequence
*
* Parameters: None
*
* Return: None
*
*****************************************************************************
**/
void mcu_latch_reset(void){

    //latch reset command
    latch_reset();

    /*Power up sequence function should be placed here
    *
    *
    */

    //Enable FAULT Bus ISRs
    init_fault_bus_interrupt();

   }
```

異常検出の例

ここに示されている異常検出例及びマイクロコントローラによる決定は、前のセクションに記載されているリファレンス コードを使用し、テーブル 5 にリストされているアクション例に従ったものです。

UART ターミナルは FAULT ステータス情報を表示して、FAULT ステート マシンを示します。表示される情報の形式は、[デバイス ID、異常、アクション] です。たとえば、W、STS、及び S という UART メッセージは、ステータス アップデートがデバイス W (デバイス 1、2、及び 3 をそれぞれ U、V、及び W で指定) からのもので、FAULT ステータスはシステム過熱保護 (STS)、MCU アクションはシャットダウン (S) であることを表しています。

図 10 は、レポートされたシステムの過熱異常と、インバータをシャットダウンするアクション例を示しています (図 11 の UART ターミナル出力を参照)。

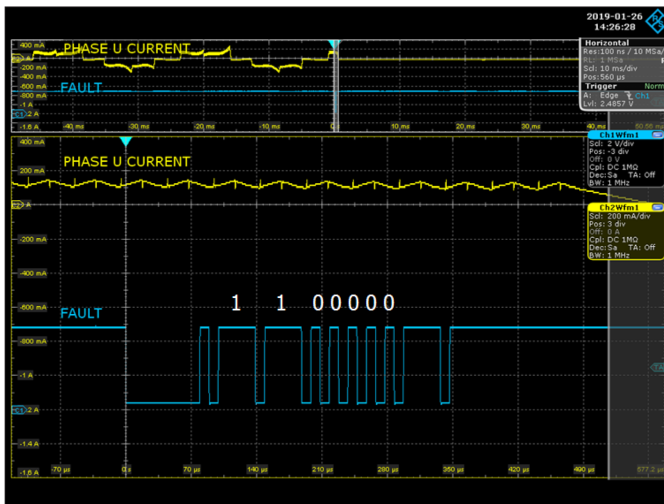

図 10.　システム温度ステータスが異常になった後のインバータ シャットダウンの例

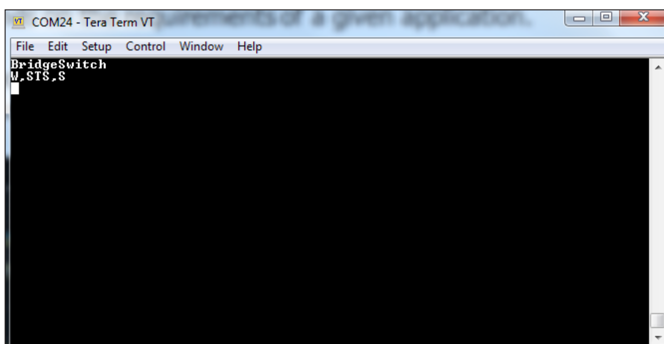

図 11.　システム過熱異常を受信した後の UART ターミナル出力

図 12 は、レポートされたローサイド過電流異常と、インバータをシャットダウンするアクション例を示しています (図 13 の UART ターミナル出力を参照)

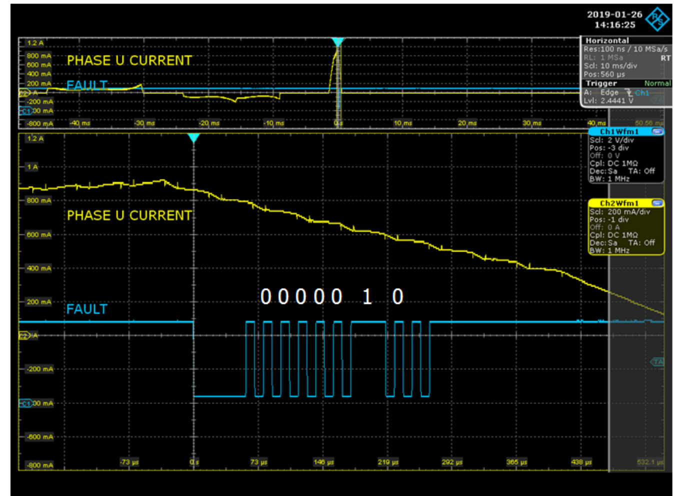

図 12.　ローサイド過電流異常を受信した後のインバータ シャットダウンの例

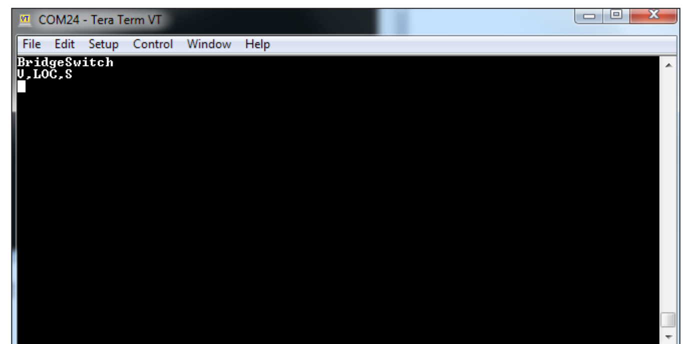

図 13.　ローサイド過電流異常を受信した後の UART ターミナル出力

図 14 は、警告ステータスでレポートされた高電圧バス UV85 異常を示しています。この実装例では、MCU は特定のアクションを実行せず、警告ステータスが表示されるだけです。図 15 の UART ターミナルを参照してください。

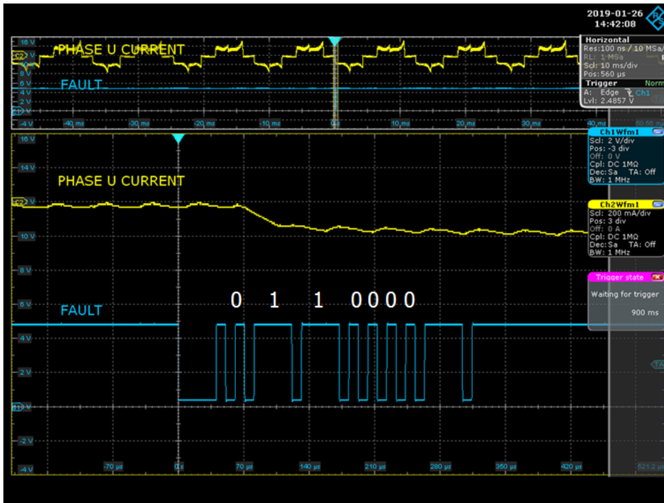

図 14.　高電圧バス UV85 を受信した後の警告ステータス

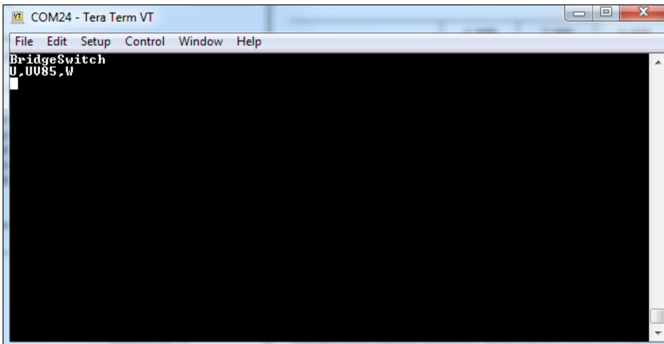

図 15.　高電圧バス UV85 を受信した後の UART ターミナル出力

図 16 は、レポートされた高電圧バス過電圧と、インバータをシャットダウンするアクション例を示しています (図 17 の UART ターミナル出力を参照)。

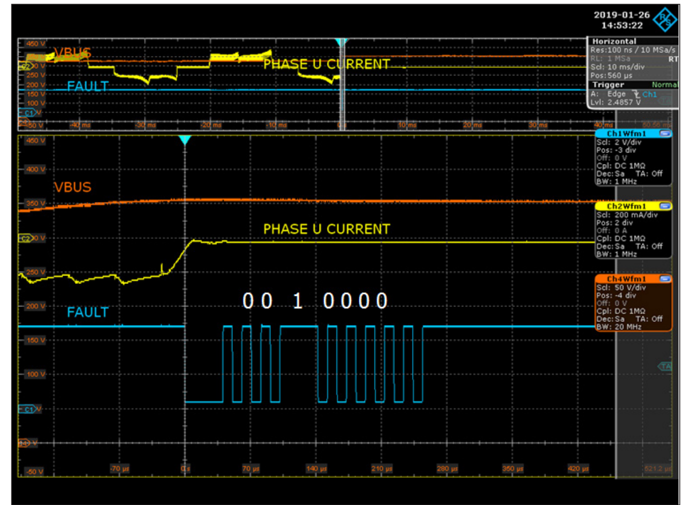

図 16.　高電圧バス過電圧を受信した後のインバータ シャットダウンの例

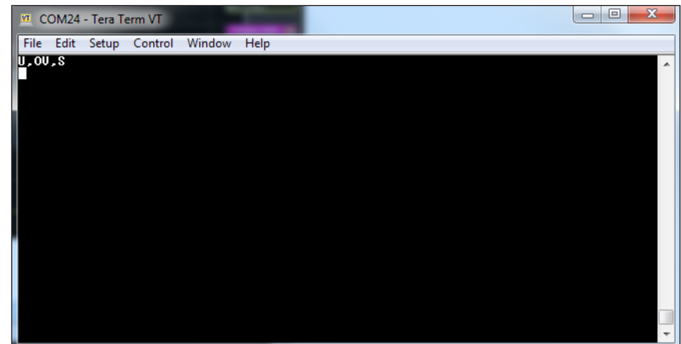

図 17.　高電圧バス過電圧を受信した後の UART ターミナル出力

MCU コマンドの例

図 18 は、入力の過電圧異常が原因でシャットダウンされた後の、ステータス問い合わせコマンド後のインバータ リスタートを示しています。

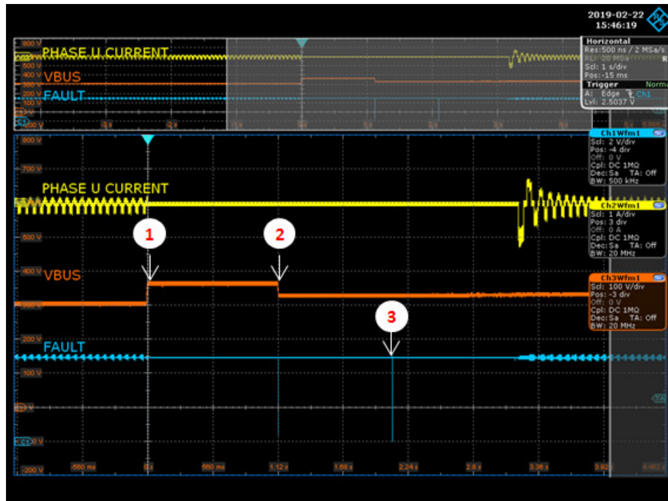

図 18.　入力過電圧後のステータス問い合わせコマンド

(1) 入力過電圧が発生し、図 19 に示されているように OV ステータスでインバータがシャットダウンされます。(2) 過電圧状態が解消され、(3) ではシステム マイクロコントローラがステータス問い合わせコマンドを送信してデバイスのステータスを確認します。図 20 は、ステータス問い合わせコマンドと、3 つのデバイスすべてからのステータス レポートを示しています。すべてのデバイスが READY ステータスをレポートし、システム マイクロコントローラがインバータをリスタートします。

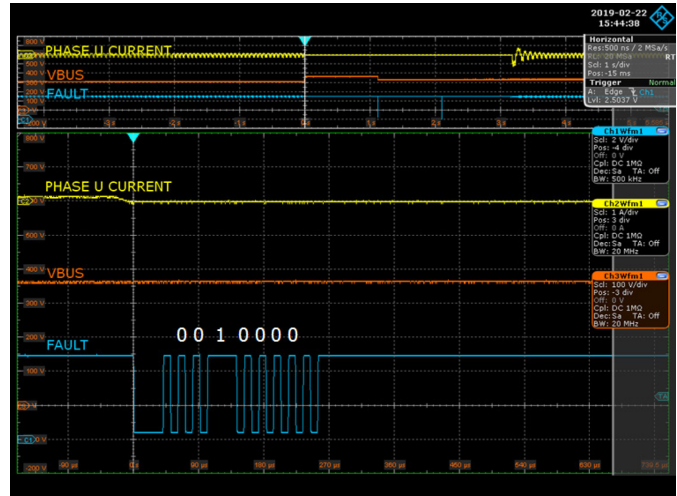

図 19.　入力過電圧後のインバータ シャットダウン

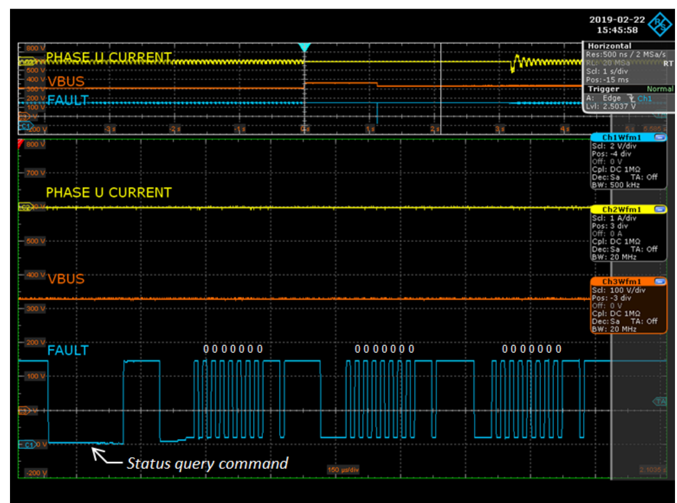

図 20.　入力過電圧後のステータス問い合わせコマンド (すべてのデバイスが READY 状態をレポート)

図 21 は、ハイサイド ドライバ未準備異常が原因でシャットダウンされた (1) 後の、ステータス問い合わせコマンド (2) 後のインバータ リスタートを示しています。

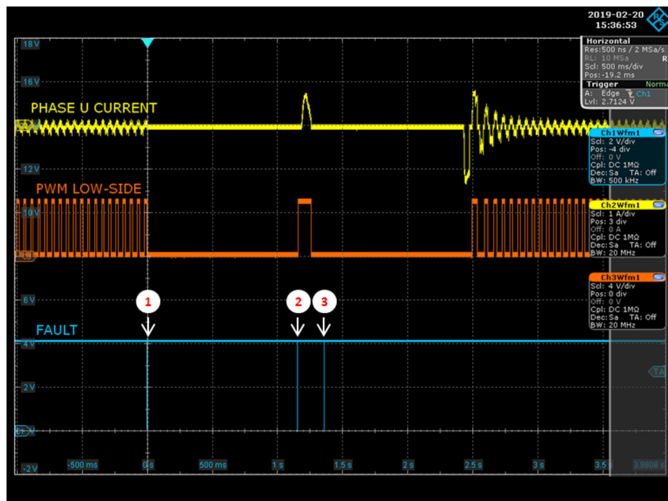

図 21.　ハイサイド ドライバ未準備異常の後の、ステータス問い合わせコマンド

この例では、レポートされたステータス アップデートはハイサイド ドライバ未準備異常だけです。MCU は起動ルーチンを要求します (100 ms の間、ローサイド PWM 入力 INL に High ロジックを適用)。(3) では、MCU が別のステータス問い合わせコマンドを送信して、すべてのデバイスが READY ステータスであるかを確認します。この例では、すべての異常が解消され、すべてのデバイスが READY ステータスです。MCU はインバータ リスタートを開始し、PWM 信号を BridgeSwitch 制御入力 INL 及び /INH に送信します。

図 22 は、対応するハイサイド ドライバの未準備の FAULT ステータスを示しています。図 23 は、ステータス問い合わせコマンドと、起動シーケンス試行後の 3 つのデバイスすべてからのステータス レポートを示しています。すべてのデバイスが READY ステータスをレポートし、システム マイクロコントローラがインバータをリスタートします。

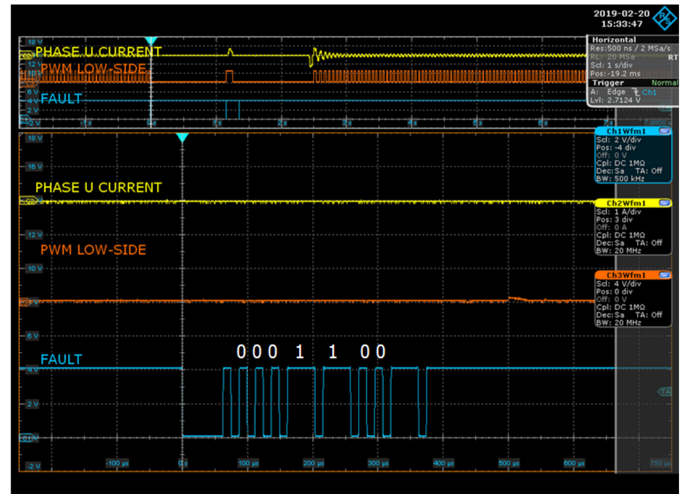

図 22.　ハイサイド ドライバ未準備異常の後の、インバータ シャットダウン

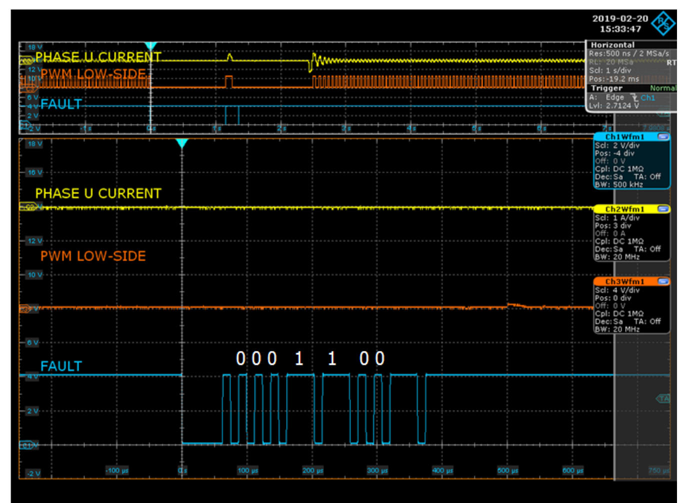

図 23.　起動シーケンスの後のステータス問い合わせコマンド

図 24 は、過熱異常が原因でのラッチ シャットダウンの後の、ラッチ リセット コマンドを示しています。MCU は、ラッチ リセット コマンドの送信後に、フル起動シーケンスを適用します。

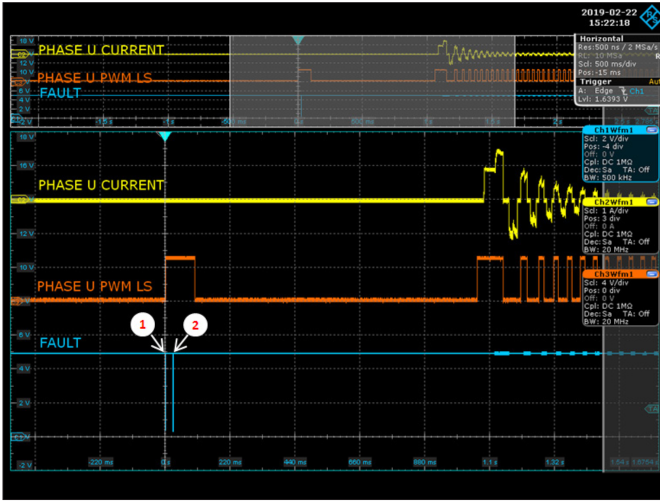

図 24.　ラッチ過熱保護の後のラッチ リセット コマンド及び起動シーケンス

図 25 は、ラッチ リセット コマンド (1) と、ハイサイド未準備のデフォルト ステータス レポートを示しています (ラッチ リセット コマンドの呼び出し時には異常検出が無効になります)。MCU は、ラッチ リセット コマンドの送信後に、起動シーケンスを適用します。図 26 では、すべてのデバイスが READY 状態をレポートし (2) インバータが起動します。

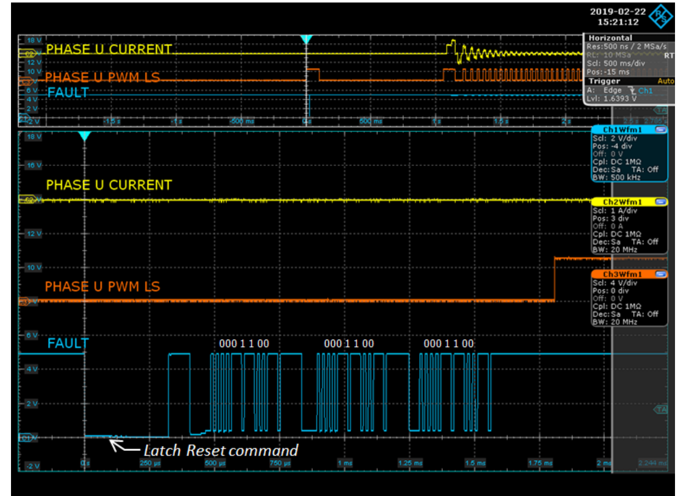

図 25.　ラッチ OTP 後の、ラッチ リセット コマンド及び起動シーケンス

図 26 では、起動シーケンスが正常に完了し、すべてのデバイスが READY ステータスをレポートしています。

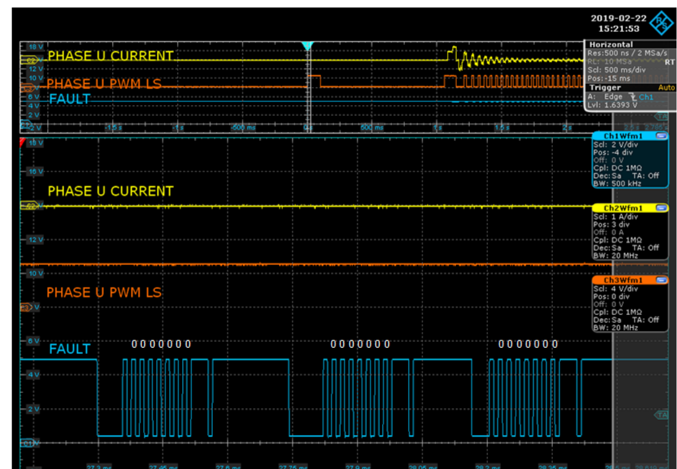

図 26.　起動シーケンスの後のデバイス ステータス

power integrations™
www.power.com

コード例ライブラリ

次のリンクを使用して、コード例ライブラリを BridgeSwitch の製品ページ (www.power.com) からダウンロードできます。

https://motor-driver.power.com/products/bridgeswitch-family/bridgeswitch/

注

このアプリケーション ノートでは、デバッグを目的とした、UART コンソールでの異常情報の表示について説明しています。この表示は、MCU にかかる負荷を制限するために、ポーリング方式で実行する必要があります。表示する情報の量を最小限に抑えることで、負荷も軽減されます。

改訂	注	日付
A	初回リリース	04/19

最新の情報については、弊社ウェブサイトを参照してください。www.power.com
Power Integrations は、信頼性や生産性を向上するために、いつでも製品を変更する権利を保有します。Power Integrations は、ここに記載した機器または回路を使用したことから生じる事柄について責任を一切負いません。Power Integrations は、ここでは何らの保証もせず、商品性、特定目的に対する適合性、及び第三者の権利の非侵害性の黙示の保証などが含まれますがこれに限定されず、すべての保証を明確に否認します。

特許情報
ここで例示した製品及びアプリケーション (製品の外付けトランス構造と回路も含む) は、米国及び他国の特許の対象である場合があります。また、Power Integrations に譲渡された米国及び他国の出願中特許の対象である可能性があります。Power Integrations が保有する特許の全リストは、www.power.com に掲載されています。Power Integrations は、www.power.com/ip.htm に定めるところに従って、特定の特許権に基づくライセンスを顧客に許諾します。

生命維持に関する方針
Power Integrations の社長の書面による明示的な承認なく、Power Integrations の製品を生命維持装置またはシステムの重要な構成要素として使用することは認められていません。ここで使用した用語は次の意味を持つものとします。

1. 「生命維持装置またはシステム」とは、(i) 外科手術による肉体への埋め込みを目的としているか、または (ii) 生命活動を支援または維持するものであり、かつ (iii) 指示に従って適切に使用した時に動作しないと、利用者に深刻な障害または死をもたらすと合理的に予想されるものです。
2. 「重要な構成要素」とは、生命維持装置またはシステムの構成要素のうち、動作しないと生命維持装置またはシステムの故障を引き起こすか、あるいは安全性または効果に影響を及ぼすと合理的に予想される構成要素です。

Power Integrations、Power Integrations ロゴ、CAPZero、ChiPhy、CHY、DPA-Switch、EcoSmart、E-Shield、eSIP、eSOP、HiperPLC、HiperPFS、HiperTFS、InnoSwitch、Innovation in Power Conversion、InSOP、LinkSwitch、LinkZero、LYTSwitch、SENZero、TinySwitch、TOPSwitch、PI、PI Expert、SCALE、SCALE-1、SCALE-2、SCALE-3、及び SCALE-iDriver は Power Integrations, Inc. の商標です。その他の商標は、各社の所有物です。©2019, Power Integrations, Inc.

Power Integrations の世界各国の販売サポート担当

世界本社
5245 Hellyer Avenue
San Jose, CA 95138, USA
代表: +1-408-414-9200
カスタマー サービス:
下記以外の国: +1-65-635-64480
アメリカ: +1-408-414-9621
電子メール: usasales@power.com

中国 (上海)
Rm 2410, Charity Plaza, No. 88
North Caoxi Road
Shanghai, PRC 200030
電話: +86-21-6354-6323
電子メール: chinasales@power.com

中国 (深圳)
17/F, Hivac Building, No. 2, Keji Nan
8th Road, Nanshan District,
Shenzhen, China, 518057
電話: +86-755-8672-8689
電子メール: chinasales@power.com

ドイツ (AC-DC/LED 販売)
Einsteinring 24
85609 Dornach/Aschheim
Germany
電話: +49-89-5527-39100
電子メール: eurosales@power.com

ドイツ (ゲート ドライバ販売)
HellwegForum 1
59469 Ense
Germany
電話: +49-2938-64-39990
電子メール:
igbt-driver.sales@power.com

インド
#1, 14th Main Road
Vasanthanagar
Bangalore-560052 India
電話: +91-80-4113-8020
電子メール: indiasales@power.com

イタリア
Via Milanese 20, 3rd.Fl.
20099 Sesto San Giovanni (MI) Italy
電話: +39-024-550-8701
電子メール: eurosales@power.com

日本
〒222-0033
神奈川県横浜市
港北区新横浜 1-7-9
友泉新横浜一丁目ビル
電話: +81-45-471-1021
電子メール: japansales@power.com

韓国
RM 602, 6FL
Korea City Air Terminal B/D, 159-6
Samsung-Dong, Kangnam-Gu,
Seoul, 135-728, Korea
電話: +82-2-2016-6610
電子メール: koreasales@power.com

シンガポール
51 Newton Road
#19-01/05 Goldhill Plaza
Singapore, 308900
電話: +65-6358-2160
電子メール:
singaporesales@power.com

台湾
5F, No. 318, Nei Hu Rd., Sec.1
Nei Hu Dist.
Taipei 11493, Taiwan R.O.C.
電話: +886-2-2659-4570
電子メール:
taiwansales@power.com

英国
Building 5, Suite 21
The Westbrook Centre
Milton Road
Cambridge
CB4 1YG
電話: +44 (0) 7823-557484
電子メール: eurosales@power.com