

メモリ用 自己診断プログラム アプリケーションノート

概要

本アプリケーションノートは、メモリの自己診断プログラムを説明した資料です。

本資料で説明されている自己診断プログラムは、メモリの読み出しと書き込みを確認するためのライブラリです。

TXZ シリーズのペリフェラルドライバが同梱されたサンプルプログラムに対応しています。

ライブラリはサンプルプログラムに上書きして使用します。

目次

概要	1
目次	2
1. はじめに.....	4
2. 自己診断テストライブラリ概要	5
2.1. 自己診断サンプルプロジェクト	5
3. 自己診断テストライブラリ詳細	6
3.1. RAM テスト	6
3.2. FLASH テスト.....	7
3.3. CRC 計算テスト設定.....	11
4. CRC32/CRC16 計算のためのリンカ設定ファイルの変更	12
5. 使用ドライバー一覧.....	13
6. 参考資料.....	14
7. 改訂履歴.....	15
製品取り扱い上のお願い.....	16

Arm、Cortex および Keil は、Arm Limited（またはその子会社）の米国およびその他の国における登録商標です。

この資料に記載されている社名・商品名・サービス名などは、それぞれ各社が商標として使用している場合があります。

1. はじめに

本アプリケーションノートは、メモリ用自己診断プログラムの説明資料です。
本ライブラリコードは、公開中のサンプルプログラムに追加(上書き)して使用します。

本資料の説明は、TMPM4K グループ(1)をベースに記載しております。
各製品への展開は、製品仕様に合わせて変更をお願いします。

本サンプルプログラムは、「自己診断プログラムアプリケーションノート 基本設定」の動作確認環境の条件、TMPM4KxA v1.0.0 のサンプルプログラム、2019年2月時点のリファレンスマニュアルを使用して開発/評価を実施しています。

2. 自己診断テストライブラリ概要

本自己診断テストライブラリは、Evaluation Board 上で動作確認を行っています。

以下のセルフテスト機能を提供します。

名称	テスト内容
RAM テスト	チェッカパターン 0x55555555 および 0xAAAAAAAA を同一メモリアドレスに書き込み、正常に読み出し可能なことを確認します。 これを指定されたメモリ範囲 (最大 18KB) に対して確認します。
FLASH テスト	CRC32 値は、指定されたアドレス範囲の読み取りデータに対して計算され、FLASH メモリにプログラムされた値が正しく読み取られることを確認します (最大 128 KB)。 プログラムビルド時にあらかじめ計算されていた CRC32 の値と比較します。 (※本テストは IAR Embedded Workbench のみ対応しています)
CRC 計算テスト	CRC 計算ハードウェア(CRC-B)を利用して、指定したメモリ範囲の CRC32 値を計算します。

2.1. 自己診断サンプルプロジェクト

Project¥Examples¥Safety フォルダ以下には、以下の自己診断ライブラリ用サンプルプロジェクトが配置されます。

プロジェクト名	動作概要	利用する自己診断ライブラリ関数
MEMORY_Sample	RAM テストおよび FLASH テストを行い、結果を LED 表示します。	safety_RAM() safety_FLASH()

3. 自己診断テストライブラリ詳細

自己診断テストライブラリ機能の詳細を記載します。
本サンプルプログラムは、メモリ用自己診断プログラムです。
以下のテストは、TPPM4K グループ(1)を使用した場合の設定例です。

自己診断ライブラリのソースコード(.c ファイル)は Libraries¥Safety¥src フォルダに、ヘッダファイル(.h ファイル)は Libraries¥Safety¥inc に存在します。

3.1. RAM テスト

チェックパターン 0x55555555 および 0xAAAAAAAA を同一メモリアドレスに書き込み、正常に読み出し可能なことを確認します。これを指定されたメモリ範囲 (最大 18KB) に対して実行します。

エラー検出時にはそのアドレスを結果のメモリバッファに書き込みます。

各メモリアドレスの値はテスト後にはテスト前の値を戻しますが、テスト失敗時の値は不定です。

テスト範囲に、割り込み処理時に変更される RAM 領域(スタック領域や割り込み処理中に書き込む変数領域)が含まれている場合、予期しない変更が発生する場合があります。このような場合はテスト領域を分割し、割り込み禁止で問題の RAM 領域のテストを実行することを推奨します。

safety_RAM 関数は割り込みを使用しません。

ソースファイル: safety_ram.c
ヘッダファイル: safety_ram.h
使用ライブラリ: txz_gpio.c/h

関数名	
bool safety_RAM(uint32_t start, uint32_t length, uint32_t *result_word)	
入力パラメータ	
uint32_t start	テストを開始する RAM アドレス 4 の倍数の値に丸められます。
uint32_t length	テスト範囲(バイト単位) 4 の倍数の値に丸められます。
出力パラメータ	
uint32_t *result_word	結果を格納するメモリバッファ 異常が検出されたアドレスを書き込みます。 (正常の場合は何もしません)
戻り値	
bool	テスト結果(true: 成功、false: 失敗)

※戻り値が数秒間確認できない場合は、上記関数以外の要因によりテストが正常に機能していないことが考えられるのでテスト失敗として処理してください。
ただし、ターミナル I/O 出力表示を使用しているときに表示終了まで数秒かかりますので、テスト失敗判断は表示を見ながら判定してください。

3.2. FLASH テスト

指定されたアドレス範囲の CRC32 の値を計算し、FLASH メモリ領域へ書き込んだ内容とは異なる値が読み出されないことを確認します。

この関数ではテストを行う範囲のチェックは行われず、設定の範囲は任意に指定可能です。適切な範囲で呼び出されているかは呼び出し側で正しく指定する必要があります。

このライブラリ関数は割り込みを使用しません。

ソースファイル: `safety_flash.c`
ヘッダファイル: `safety_flash.h`
使用ライブラリ: `txz_gpio.c/h`

関数名	
<code>bool safety_FLASH(uint32_t start, uint32_t length, uint32_t crc32_value)</code>	
入力パラメータ	
<code>uint32_t start</code>	テストを開始する FLASH アドレス 1 バイト単位の値を指定可能
<code>uint32_t length</code>	テスト範囲(バイト) 1 バイト単位の値を指定可能
<code>uint32_t crc32_value</code>	あらかじめ計算しておいたテスト範囲の CRC32 値
出力パラメータ	
なし	-
戻り値	
<code>bool</code>	テスト結果(true: 成功、false: 失敗)

※戻り値が数秒間確認できない場合は、上記関数以外の要因によりテストが正常に機能していないことが考えられるのでテスト失敗として処理してください。
ただし、ターミナル I/O 出力表示を使用しているときに表示終了まで数秒かかりますので、テスト失敗判断は表示を見ながら判定してください。

<IAR Embedded Workbench の場合>

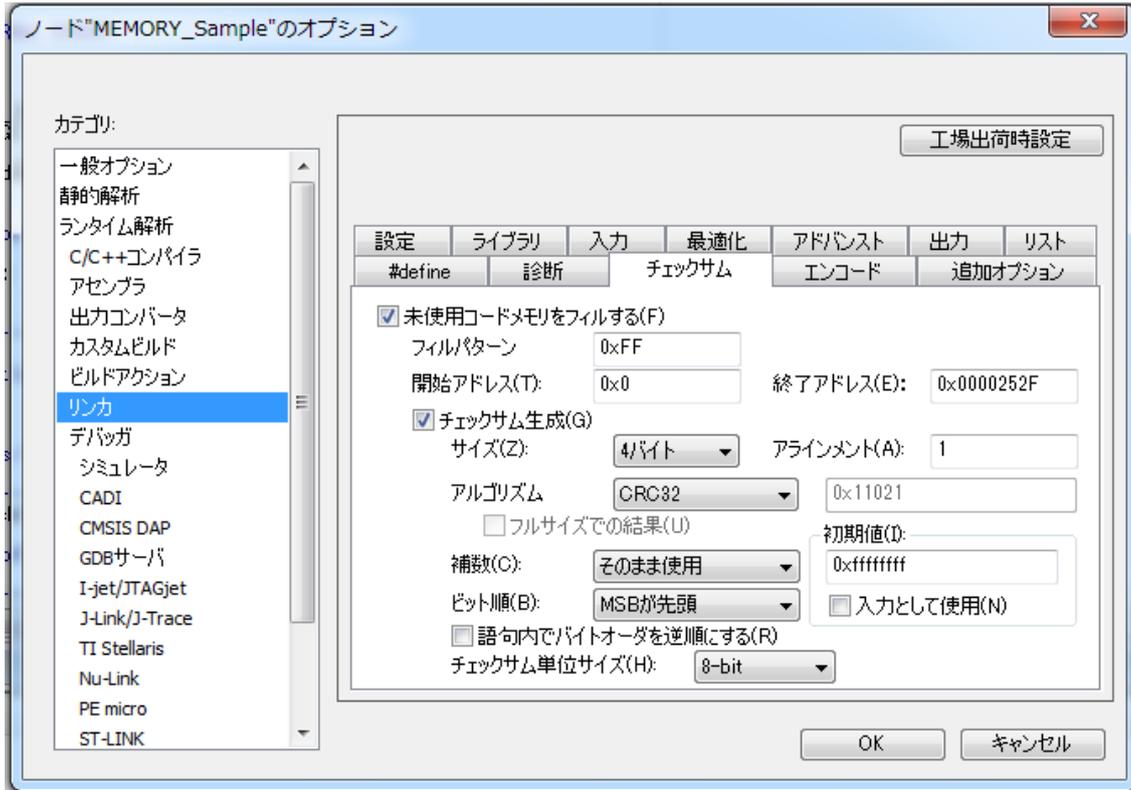
このライブラリ関数を利用するには、FLASH メモリ範囲の CRC32 の値を事前に求めておくことが必要です。これを IAR Embedded Workbench のリンカオプションを利用して行うには、以下のように設定します。

(CRC16 の場合も同様の手順で準備を行います)

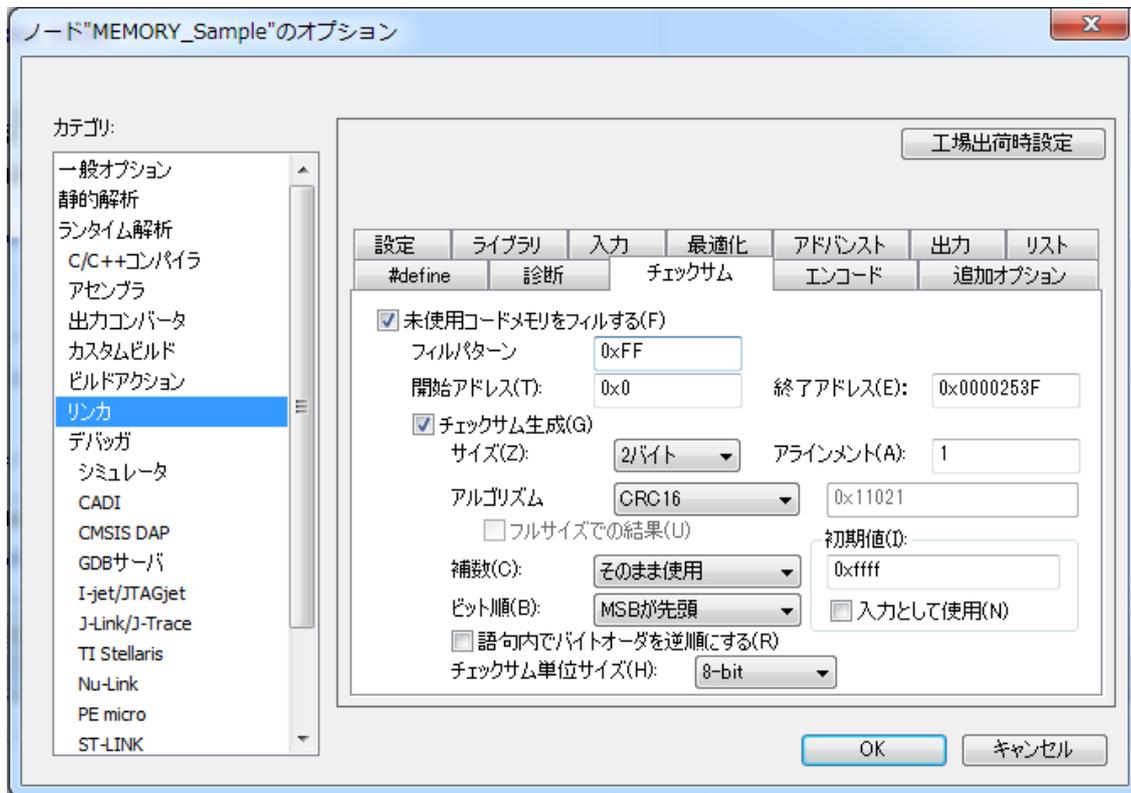
(例として MEMORY_Sample プロジェクトを利用して説明します)

1. 「CRC32/CRC16 計算のためのリンカ設定ファイルの変更」を参照し、リンカ設定ファイルを変更します。
2. 対象プロジェクトのオプション設定で、リンカオプションの「チェックサム」タブで下図のように設定します。

CRC32 の場合:



CRC16 の場合 :



このうち、「開始アドレス」と「終了アドレス」はプロジェクトごとに変更します。

一般に開始アドレスは FLASH メモリの開始(TMPM4K4A の場合は 0x0)、終了アドレスは FLASH メモリの最後の使用アドレスに設定しますが、CRC32 および CRC16 の値を保存してある領域は含めないようにする必要がありますので、何らかの方法でこのアドレスを知る必要があります。

3. いったん終了アドレスを適当な値 (0x1000 など) に設定してビルドをしてみます。
4. MAP ファイルから __checksum 変数のアドレスを確認します。

確認方法は、「CRC32 計算のためのリンカ設定ファイルの変更」に記述しています。

MEMORY_Sample.map CRC32 の例 (抜粋)

```
.rodata      const      0x000024a8    0x0  zero_init3.o [5]
.rodata      const      0x000024a8    0x0  packbits_init_single.o [5]
Initializer bytes  const      0x000024a8    0x85 <for P2-1>
CHECKSUM      0x00002530    0x4  <Block>
.checksum     const      0x00002530    0x4  Place holder __checksum
              - 0x00002534  0x2390
```

これは、リンカの算出したチェックサム結果を格納する __checksum 変数が 0x2530 に割り付けられたことを示しています。1. でリンカの .icf 設定を変更して __checksum が FLASH 上の ReadOnly メモリの最後に来るようにしてあるので、FLASH メモリでチェックすべき範囲はその前のアドレス、つまり 0x252F までということになります。

5. 再びリンカ設定の「チェックサム」タブを開いて、「終了アドレス」に 0x252F を設定して再度ビルドをします。すると、チェックサム計算範囲とその値がビルド時のメッセージとして表示されます。
6. これで準備ができました。あとはプログラム中から以下の変数を利用することで、FLASH メモリ範囲と CRC32 および CRC16 値を利用することができます。

```
// use IAR Linker defined symbol "__checksum"
extern uint32_t __checksum;

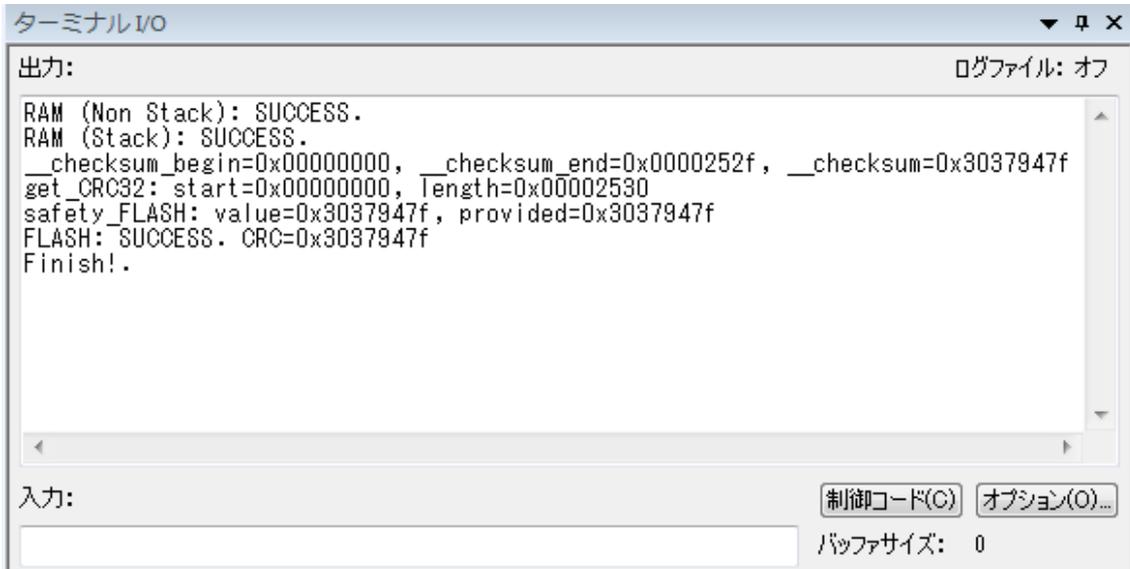
// use IAR Linker defined symbol "__checksum_begin", "__checksum_end"
// following symbols are defined as assembler LABEL
extern uint8_t __checksum_begin;      // FLASH start
extern uint8_t __checksum_end;      // FLASH end except __checksum

// in IAR we can use Linker Symbol so use it
#define FLASH_TEST_START      (uint32_t)&__checksum_begin
#define FLASH_TEST_SIZE      (&__checksum_end - &__checksum_begin + 1)
#define FLASH_TEST_CRC32VALUE (__checksum)

(中略)

result = safety_FLASH(FLASH_TEST_START, FLASH_TEST_SIZE,
FLASH_TEST_CRC32VALUE);
```

このように設定した MEMORY_Sample で DEBUGMSG を有効にするには、一般オプション/ライブラリ設定・低レベルインタフェースのライブラリ実装を「セミホスティング」、stdout/stderr を「セミホスティング経由」に設定し、C/C++コンパイル/プリプロセッサ・シンボル定義から NDEBUG を削除してビルドをします。この場合、以下のようにターミナル I/O に出力されます。



The screenshot shows a terminal window titled "ターミナル I/O" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains the following text:

```
出力: ログファイル: オフ
RAM (Non Stack): SUCCESS.
RAM (Stack): SUCCESS.
__checksum_begin=0x00000000, __checksum_end=0x0000252f, __checksum=0x3037947f
get_CRC32: start=0x00000000, length=0x00002530
safety_FLASH: value=0x3037947f, provided=0x3037947f
FLASH: SUCCESS. CRC=0x3037947f
Finish!.
```

At the bottom of the window, there is an input field labeled "入力:" and two buttons: "制御コード(C)" and "オプション(O)...". To the right of the input field, it says "バッファサイズ: 0".

3.3. CRC 計算テスト設定

FLASH テストでは CRC 計算機能(CRC-B)を利用して、指定したメモリ範囲の CRC32 値および CRC16 値を計算します。このコードを参考に CRC-B を利用して CRC 計算を行うことができます。
(Libraries¥Safety¥src¥safety_flash.c を参照)

CRC32、CRC16 を動作させるために、以下の設定を切り替える必要があります。

Libraries¥Safety¥inc¥ safety_flash.h より

<CRC32 の場合>

#define CRC32	0
#define CRC16	1
#define CRC	CRC32

<CRC16 の場合>

#define CRC32	0
#define CRC16	1
#define CRC	CRC16

4. CRC32/CRC16 計算のためのリンカ設定ファイルの変更

MEMORY_Sampleプロジェクトでは、CRC32-B回路を利用してFLASHメモリアドレス範囲のCRC32およびCRC16値を計算します。このとき、計算した値がコンパイラの出力したバイナリファイルのCRC32およびCRC16値と一致するか比較するために、リンカの機能を利用してあらかじめFLASHメモリ範囲の値を計算させます。

これを行うために、TMPM4K4A用に用意された共通のリンカ設定ファイルTMPM4K4FYAUG.icf(注1)から以下のように変更します。

TMPM4K4FYAUG.icfはIAR Embedded Workbenchのみ使用可能なリンカ設定ファイルとなります。Arm® Keil®ではリンカ設定ファイルが使用できません。

<IAR Embedded Workbenchの場合>

変更前の記述

```
place in ROM_region { readonly};
```



変更後の記述

```
define block CHECKSUM with alignment = 4 { ro section .checksum };  
place in ROM_region { ro, last block CHECKSUM};
```

変更したリンカ設定ファイルTMPM4K4FYAUG_checksum.icfをLibraries¥Safetyフォルダに用意しました。

この変更により、リンカが自動的に設定するメモリ範囲のCRC32値の保存領域である、__checksum変数がReadonlyデータの最後に配置されます。(チェックサム出力用のリンカオプションを設定せず、__checksum変数が使用されない場合は、このリンカ設定ファイル変更の影響はありません)

__checksum変数が最終的にどこに配置されたかは、プロジェクトビルド時に生成される.mapファイルで確認可能です。MEMORY_Sampleでは、Project¥Examples¥Safety¥MEMORY_Sample¥Debug¥Listに.mapファイルが生成されます。

MEMORY_Sample.map の例 (抜粋)

```
.text          ro code  0x00002450   0x38  packbits_init_single.o [5]  
.iar.init_table  const   0x00002488   0x20  - Linker created -  
.rodata        const   0x000024a8   0x0   zero_init3.o [5]  
.rodata        const   0x000024a8   0x0   packbits_init_single.o [5]  
Initializer bytes  const   0x000024a8   0x85  <for P2-1>  
CHECKSUM       const   0x00002530   0x4   <Block>  
.checksum      const   0x00002530   0x4   Place holder __checksum  
              - 0x00002534 0x2390
```

上記の例では、0x2530に__checksum変数が割り当てられています。

注 1: C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\linker\Toshibaフォルダに保存されています。

5. 使用ドライバー一覧

このテストライブラリではTPM4KxA v1.0.0バージョンのドライバおよびプロジェクト内のコードを利用しています。

CMSIS ライブラリ

カテゴリ	ソースファイル名
スタートアップ	startup_TPM4K4A.s
システム(クロック設定など)	system_TPM4KxA.c

Periph_driver

カテゴリ	ソースファイル名
GPIO	txz_gpio.c

Project Examples 内

カテゴリ	ソースファイル名
BSP (評価ボードサポート)	bsp.c
LED 出力	bsp_led.c

6. 参考資料

以下の資料を使用して開発を実施しています。

- ・ データシート
- ・ リファレンスマニュアル
- ・ 自己診断プログラムアプリケーションノート基本設定
- ・ ARM® Cortex®-M4 Processor technical Reference Manual
- ・ ARMv7-M Architecture Reference Manual

7. 改訂履歴

Revision	Date	Description
1.0	2019-08-29	初版

製品取り扱い上のお願い

株式会社東芝およびその子会社ならびに関係会社を以下「当社」といいます。

本資料に掲載されているハードウェア、ソフトウェアおよびシステムを以下「本製品」といいます。

- 本製品に関する情報等、本資料の掲載内容は、技術の進歩などにより予告なしに変更されることがあります。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。また、文書による当社の事前の承諾を得て本資料を転載複製する場合でも、記載内容に一切変更を加えたり、削除したりしないでください。
- 当社は品質、信頼性の向上に努めていますが、半導体・ストレージ製品は一般に誤作動または故障する場合があります。本製品をご使用頂く場合は、本製品の誤作動や故障により生命・身体・財産が侵害されることのないように、お客様の責任において、お客様のハードウェア・ソフトウェア・システムに必要な安全設計を行うことをお願いします。なお、設計および使用に際しては、本製品に関する最新の情報（本資料、仕様書、データシート、アプリケーションノート、半導体信頼性ハンドブックなど）および本製品が使用される機器の取扱説明書、操作説明書などをご確認の上、これに従ってください。また、上記資料などに記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を使用する場合は、お客様の製品単独およびシステム全体で十分に評価し、お客様の責任において適用可否を判断してください。
- 本製品は、特別に高い品質・信頼性が要求され、またはその故障や誤作動が生命・身体に危害を及ぼす恐れ、膨大な財産損害を引き起こす恐れ、もしくは社会に深刻な影響を及ぼす恐れのある機器（以下“特定用途”という）に使用されることは意図されていませんし、保証もされていません。特定用途には原子力関連機器、航空・宇宙機器、医療機器（ヘルスケア除く）、車載・輸送機器、列車・船舶機器、交通信号機器、燃焼・爆発制御機器、各種安全関連機器、昇降機器、発電関連機器などが含まれますが、本資料に個別に記載する用途は除きます。特定用途に使用された場合には、当社は一切の責任を負いません。なお、詳細は当社営業窓口まで、または当社 Web サイトのお問い合わせフォームからお問い合わせください。
- 本製品を分解、解析、リバースエンジニアリング、改造、改変、翻案、複製等しないでください。
- 本製品を、国内外の法令、規則及び命令により、製造、使用、販売を禁止されている製品に使用することはできません。
- 本資料に掲載してある技術情報は、製品の代表的動作・応用を説明するためのもので、その使用に際して当社及び第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。
- 別途、書面による契約またはお客様と当社が合意した仕様書がない限り、当社は、本製品および技術情報に関して、明示的にも黙示的にも一切の保証（機能動作の保証、商品性の保証、特定目的への合致の保証、情報の正確性の保証、第三者の権利の非侵害保証を含むがこれに限らない。）をしておりません。
- 本製品、または本資料に掲載されている技術情報を、大量破壊兵器の開発等の目的、軍事利用の目的、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替及び外国貿易法」、「米国輸出管理規則」等、適用ある輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。
- 本製品の RoHS 適合性など、詳細につきましては製品個別に必ず当社営業窓口までお問い合わせください。本製品のご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用ある環境関連法令を十分調査の上、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は一切の責任を負いかねます。