RX66T グループ　e-AI モータ異常検知サンプルソフト
アプリケーションノート

要旨

本アプリケーションノートは e-AI (embedded-Artificial Intelligence)の使用例として、RX66T を使用したモータ制御システムにモータの異常を検知する機能を追加したアプリケーション例について説明します。

サンプルソフトウェアには学習済 DNN を同梱しています。本書記載のハードウェアを入手することで、すぐに e-AI システムの動作確認が出来ます。

本アプリケーションノートの対象ソフトウェアはあくまで参考用途であり、弊社がこの動作を保証するものではありません。本アプリケーションノート対象ソフトウェアを使用する場合、適切な環境で十分な評価をしたうえで御使用下さい。

動作確認デバイス

RX66T(R5F566TEADFP)

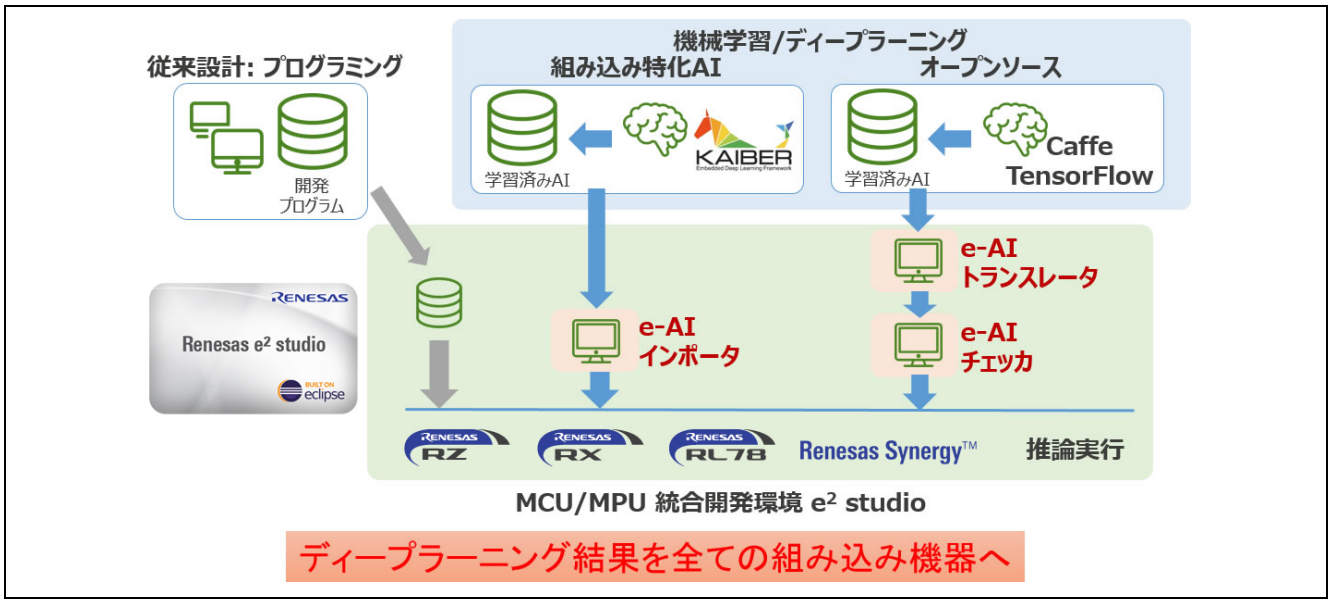
目次

1.　はじめに

ルネサスは Endpoint intelligence からはじまるイノベーションにより、クラウドやエッジでのビッグデータ活用だけでは解決できない領域で、より安全で健やかな暮らしを支える、環境に優しいスマート社会の実現に貢献していきます。

バークレーが開発した Caffe や、Google の開発した TensorFlow といったフレームワークを利用することにより、誰もが比較的容易に AI（Artificial Intelligence）を利用することが可能になりました。 AI の得意分野は使用するアルゴリズムによって様々ですが、碁や画像識別で一躍有名となった AI には、多層のネットワークを積層した DNN（Deep Neural Network）が利用されています。教師データと呼ばれるラベル付きの情報を入力し、出力に現れる推定結果が一致するように学習を行うアルゴリズムで、多層化、および特徴抽出の自動化の技術的ブレークスルーのおかげで飛躍的に推定精度が向上しました。DNN は、学習と推論実行とで必要な計算量に大きな差があり、推論時には少ない計算パワーで実行できることが大きな特徴です。この計算パワーの非対称性に着目して、組み込み機器において主に推論実行を行うことに対し、e-AI (embedded-Artificial Intelligence)　と名づけました。

この学習済 DNN を MCU/MPU に実装するのに有効なツールが、e-AI 開発環境です。e-AI 開発環境では、e2 studio の C/C++プロジェクトに適合する形で、学習済み DNN を MCU/MPU に実装することが可能です。



2.　概説

図 2.1 にシステムブロック図を示します。本例はブラシレス DC モータ制御を行う MCU のソフトウェアに学習済 DNN を追加した e-AI によるモータ状態（異常値）表示システムです。AI 推論の結果は PC のソフトウェアで表示します。

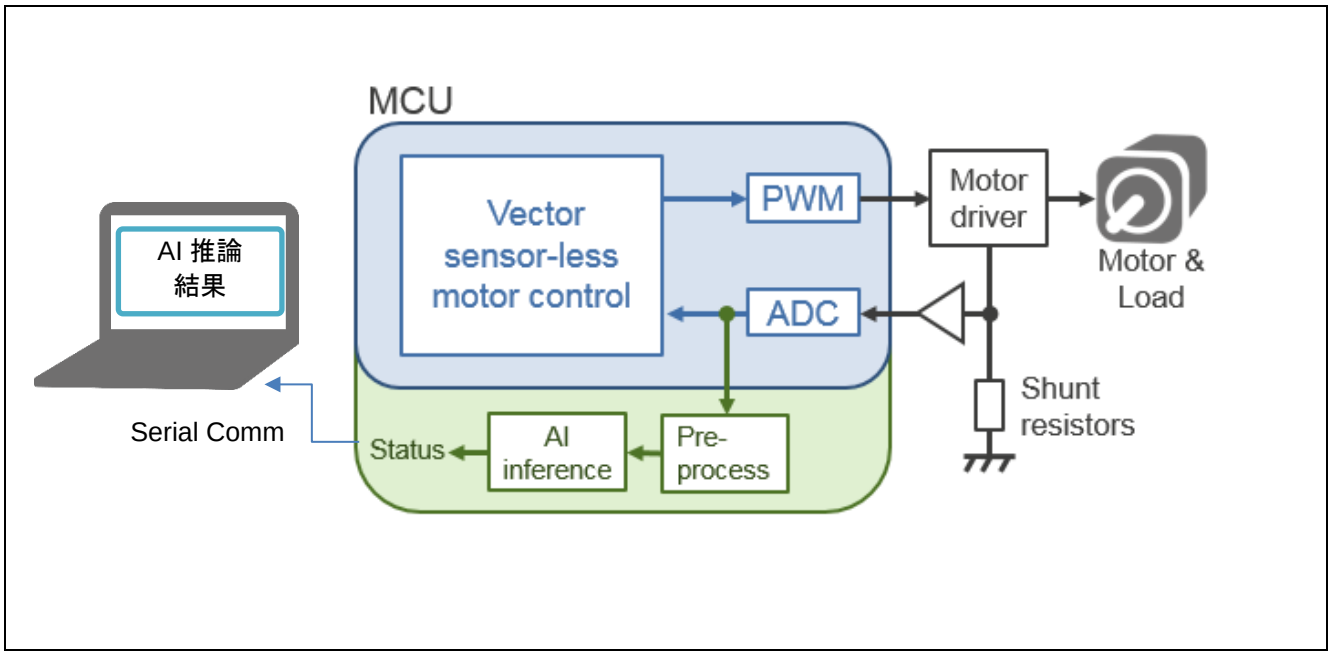


図 2.1　システムブロック図

本例の AI 推論処理は下記の動作を行います。

1. 3 相電流の A/D 変換値を蓄積し FFT のフレームを生成
2. 学習済 DNN の入力データ前処理
   i. データフレームを FFT（周波数スペクトル生成）
   ii. 周波数スペクトルの特徴点抽出（学習済 DNN の入力データ生成）
3. AI 推論

本システムのブラシレス DC モータ制御にはセンサレスベクトル制御方式を用いており、制御のため 3 相電流を A/D コンバータでモニタリングしています。本システムでは 3 相電流の波形がモータの状態により変化することに着目し、この 3 相電流を学習済 DNN の入力として使用します。A/D 変換値は一定時間蓄積し、時間軸の波形データを得ます。

入力データの前処理では AI が 3 相電流波形の特徴点を検出しやすいよう FFT により周波数スペクトルを生成します。FFT 処理では、FFT のフレームの切れ目の変化を検出できるように、2 つのフレームバッファをオーバーラップさせています。また記憶領域の限られる e-AI システムでは DNN のネットワーク層削減もポイントであり、入力データの特徴点周辺を抽出して使用する工夫もしています。

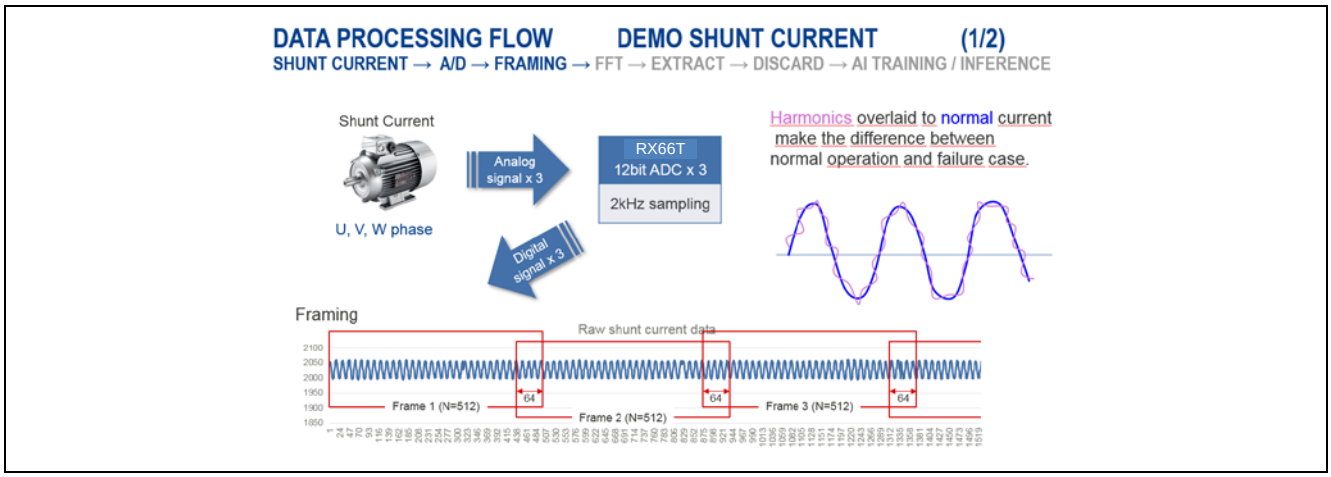


図 2.2　データ処理フロー図(1/2)

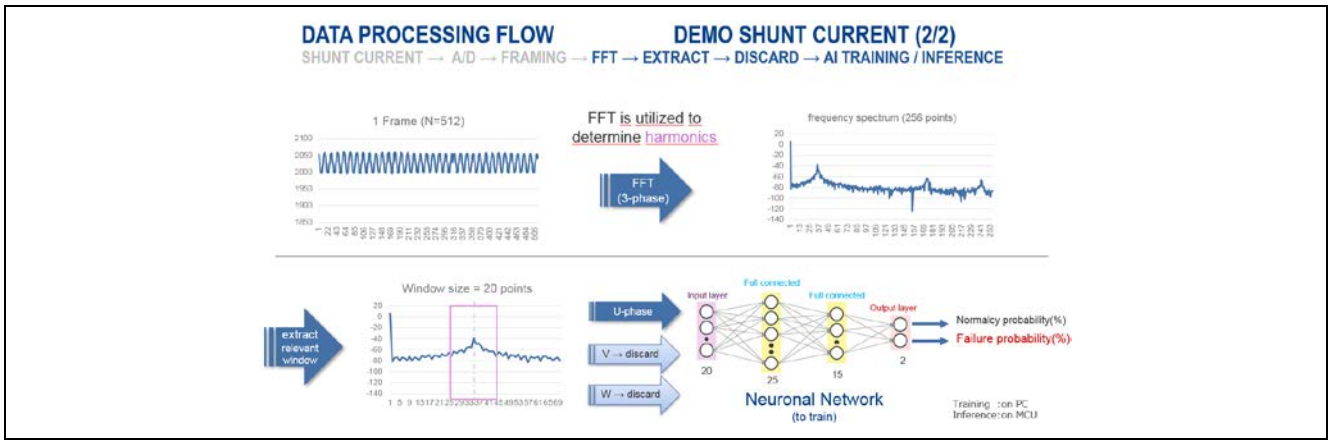


図 2.3　データ処理フロー図(2/2)

AI 推論結果は同梱の PC のソフトウェア（DataCollectionTool）で表示します。本例は推論結果の他に 3 相電流の A/D 変換値およびスペクトル波形を表示出来ます。また波形データのログ機能を持っているため、波形表示しながら学習用データを収集できます。DataCollectionTool の詳細は「5. 波形モニターツール」を参照してください。

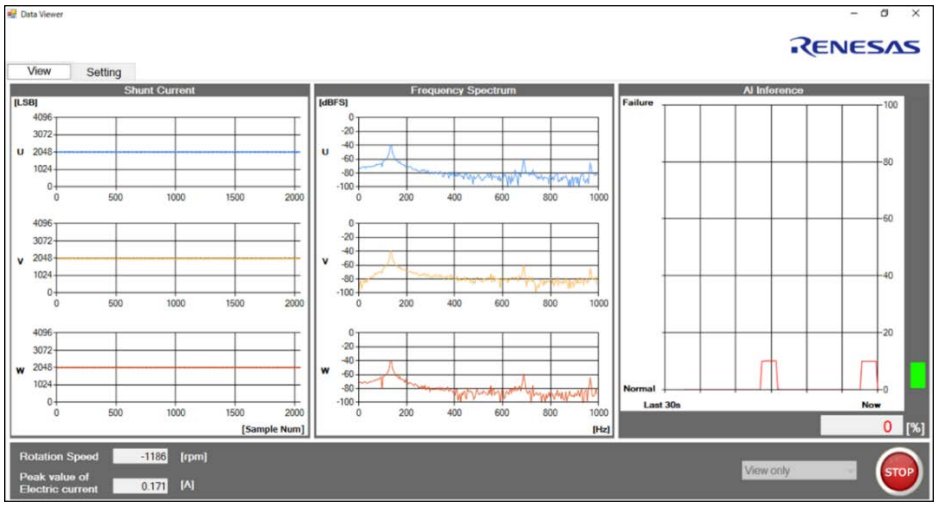


図 2.4　DataCollectionTool 概観

3.　仕様

3.1　動作確認条件

表 3.1 に動作確認条件を示します。

表 3.1　動作確認条件

項目	内容
使用マイコン	RX66T（R5F566TEADFP）
動作周波数	メインクロック：8MHz 水晶発振子 CPU クロック（ICLK）：160MHz 周辺モジュールクロック A（PCLKA）：80MHz 周辺モジュールクロック B（PCLKB）：40MHz 周辺モジュールクロック C（PCLKC）：160MHz 周辺モジュールクロック D（PCLKD）：40MHz FlashIF クロック（FCLK）：40MHz
動作電圧	5.0V（RX66T）
動作モード	シングルチップモード
エンディアン	リトルエンディアン
統合開発環境	ルネサス エレクトロニクス製　e^{2}studio V7.2.0
C コンパイラ	ルネサス エレクトロニクス製　CC-RX: V3.00.00 • コンパイラオプション：-isa=rxv3 -fpu -save_acc
自動生成ツール	ルネサス エレクトロニクス製　RX スマート・コンフィグレータ V7.2.0
エミュレータ	ルネサス エレクトロニクス製　E2 Lite emulator
ミドルウェア	ルネサス エレクトロニクス製　RX ファミリ RX DSP ライブラリ Version 5.0
e-AI 開発環境	ルネサス エレクトロニクス製　e-AI トランスレータ V1.0.2 ルネサス エレクトロニクス製　e-AI チェッカ V1.0.2
評価ボード	ルネサス エレクトロニクス製 24V Motor Control Evaluation System for RX23T（RTK0EM0006S01212BJ） • 24V 系インバータボード ルネサス エレクトロニクス製　RX66CPU カード（RTK0EMX870C00000BJ）
モータ	ツカサ電子工業株式会社製　TG55-L（RTK0EM0006S01212BJ 同梱品） 簡易モータベンチ【注】

注：詳細は付録の「簡易モータベンチの組立」を参照してください。

3.2　ハードウェアブロック図

図 3.1 にハードウェアブロック図を示します。本例は参考資料[2]「永久磁石同期モータのセンサレスベクトル制御 RX66T 実装編」に記載されたハードウェア構成をベースとし、通信機能と駆動モータの負荷デバイスを追加した構成となっています。ベースのハードウェア構成詳細は参考資料[2]を参照してください。

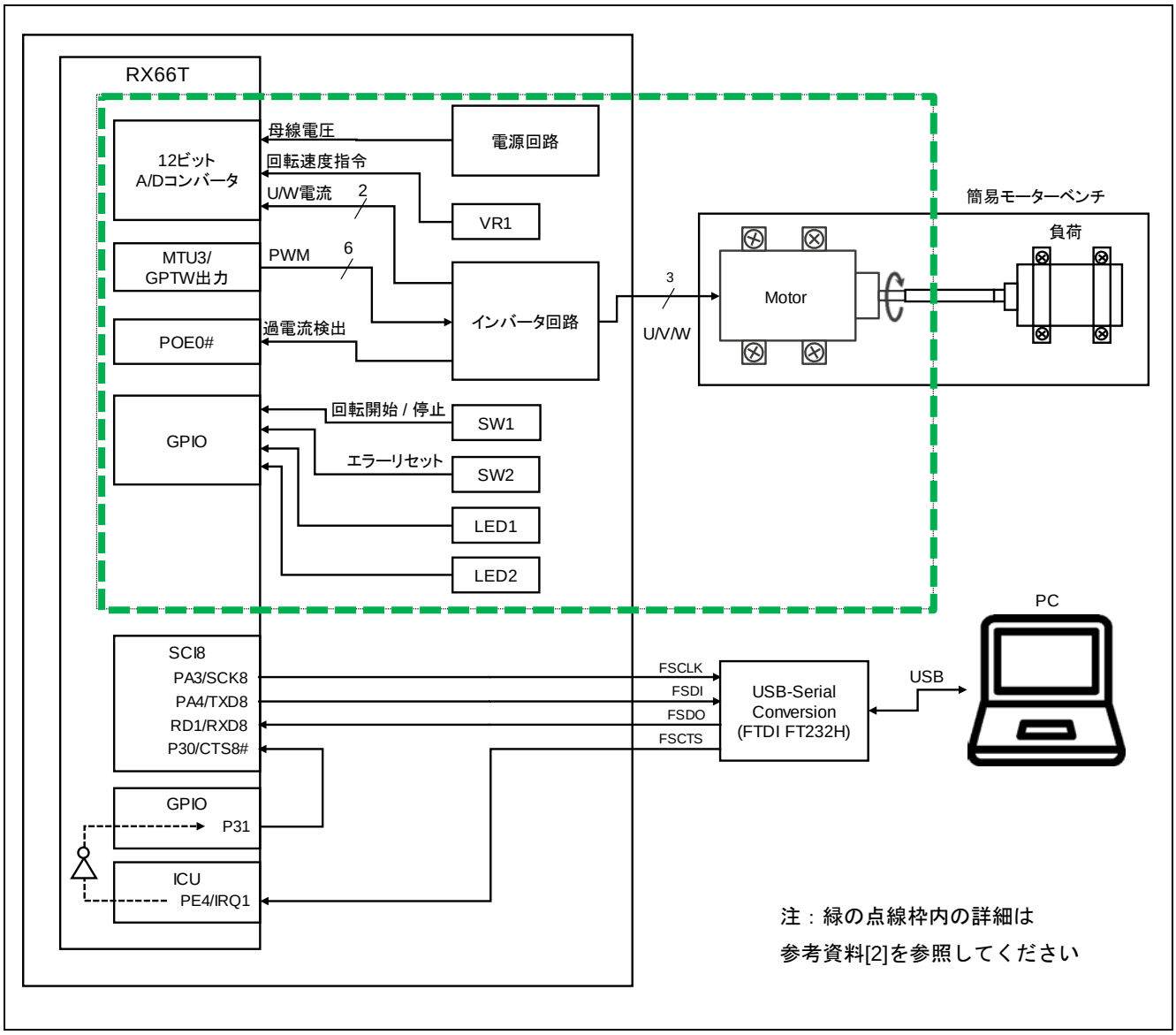


図 3.1　ハードウェアブロック図

3.3　動作概要

図 3.2 にシステム動作フローを示します。下記に動作概要を示します。

①　モータのセンサレスベクトル制御を実行

24V 系インバータボードに電源を投入すると RX66TCPU ボードにも電源が投入され、モータ駆動動作を開始します。処理の詳細は参考資料[2]、ボード操作の詳細は参考資料[3]を参照してください。

②　モータ駆動電流データの前処理を実行、e-AI 推論で異常を判別

1. A/D 変換値の蓄積
   CMT1 により 2kHz のサンプリング周波数を生成し、モータ 3 相電流の A/D 変換値を取得します。3 相電流は 12 ビット A/D コンバータ（S12ADH）に入力されます。A/D 変換値は FFT のため 512 サンプル（1 フレーム）蓄積します。次のフレーム以降は直前のフレームの後端 64 サンプルをオーバーラップさせ A/D 変換値を蓄積します。
2. データ前処理
   RX DSP Library を使用し FFT 処理を行います。FFT 結果の周波数スペクトルは dBFS 単位に変換します。本例では 0dB = S12ADH のフルスケール値 = 4095LSB と定義しています。次に周波数スペクトルの特徴点抽出のため、DC 成分を除いた周波数スペクトルのピーク値と前後 10 サンプルを切り出します。
3. AI 推論
   抽出した特徴点を学習済 DNN に入力し、推論により 2 つのクラス（正常と異常）の確率を出力します。本例では異常の確率を異常の度合いとしています。

③　シリアル通信で PC と通信

SCI8 を調歩同期式モード（クロック出力有り、フロー制御有り）で使用し、USB-シリアル変換ケーブルを用いて各データを PC に転送します。SCI8 の送信データレジスタ（TDR）へのデータ転送には DMAC0 を使用します。

④　異常の度合いや電流波形データをツールに表示

PC 上で動作する GUI ツール「Data Collection Tool」により受信データを数値およびグラフ表示します。

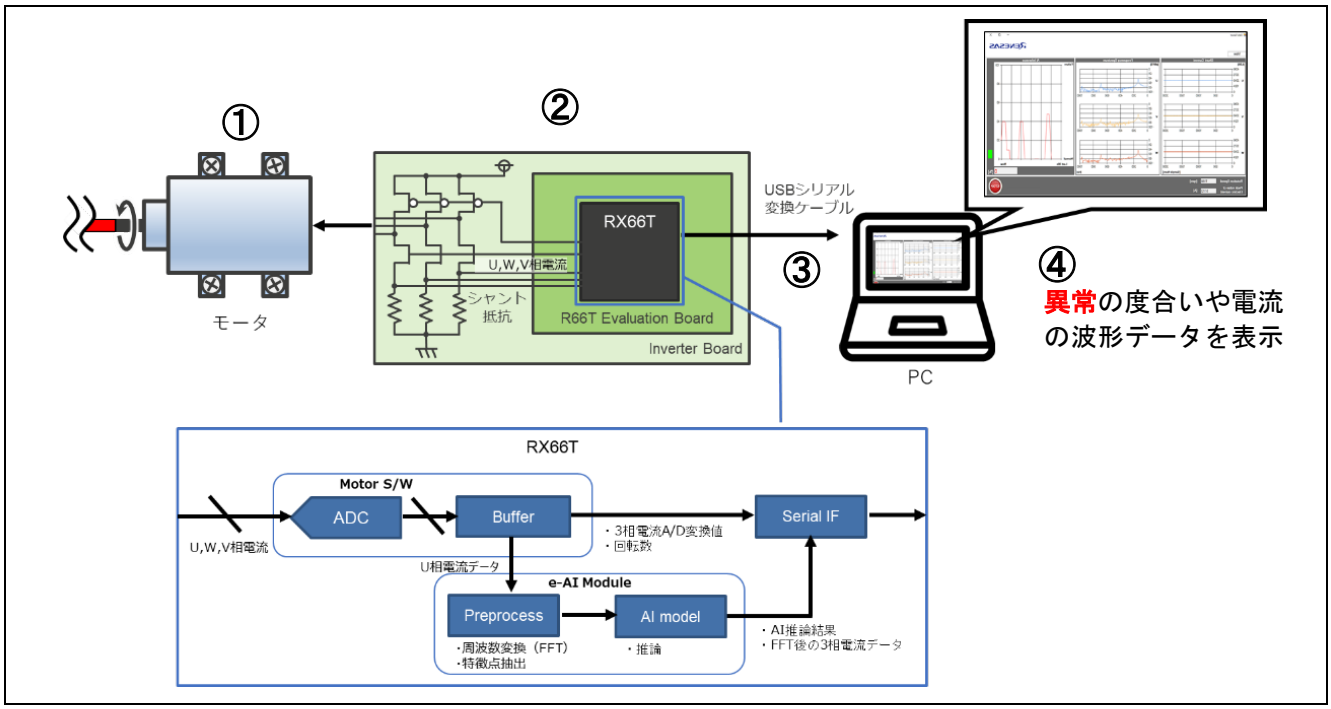


図 3.2　システム動作フロー

図 3.3 に正常の状態と異常の状態のイメージ図を示します。駆動モータと負荷モータのシャフトが同一線上にある状態を正常とし、軸がずれた状態を異常と定義しています。本例では駆動モータと負荷モータのシャフトをチューブでカップリングした簡易モータベンチを使用してこの状態を再現します。

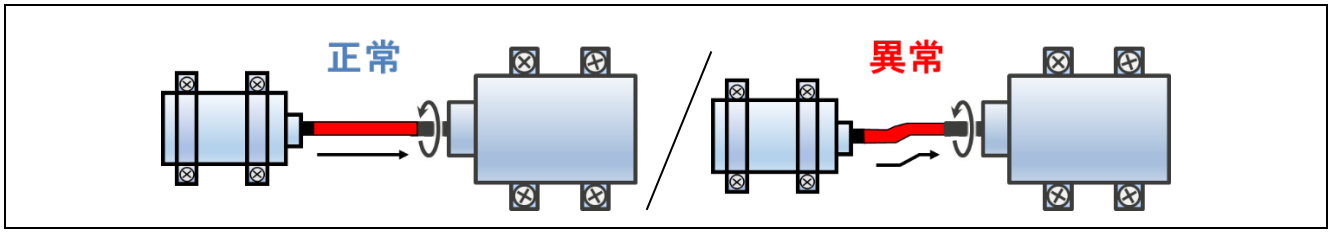


図 3.3　正常と異常の状態のイメージ図

3.4　状態遷移

図 3.4 に本例の状態遷移図を示します。本章で示す SW1、SW2、VR1、LED1 および LED2 は 24V Motor Control Evaluation System for RX23T の 24V 系インバータボードに実装されたデバイスを指します。RESET ボタンは RX66CPU カードに実装されたデバイスを指します。

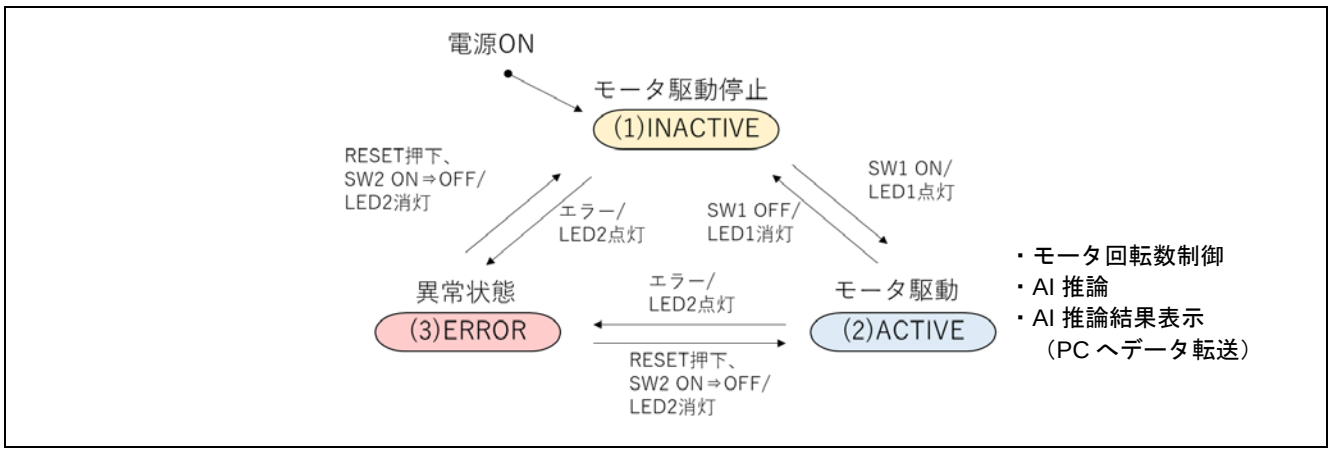


図 3.4　状態遷移図

(1) INACTIVE

電源投入直後の状態を示します。この状態ではモータは駆動しません。SW1 を ON にすると ACTIVE 状態に遷移します。INACTIVE 状態でエラーを検知すると ERROR 状態に遷移します。

(2) ACTIVE

ACTIVE 状態では LED1 が点灯しモータを駆動することができます。ACTIVE 状態のとき、以下の処理を行います。

- モータ回転数制御
  VR1 でモータの回転数を制御します。
- e-AI 推論
  モータの異常度合いを推論します。
- PC へデータ転送
  モータの駆動電流のデータや、推論結果を PC に送信します。

ACTIVE 状態でエラーが発生すると ERROR 状態に遷移します。

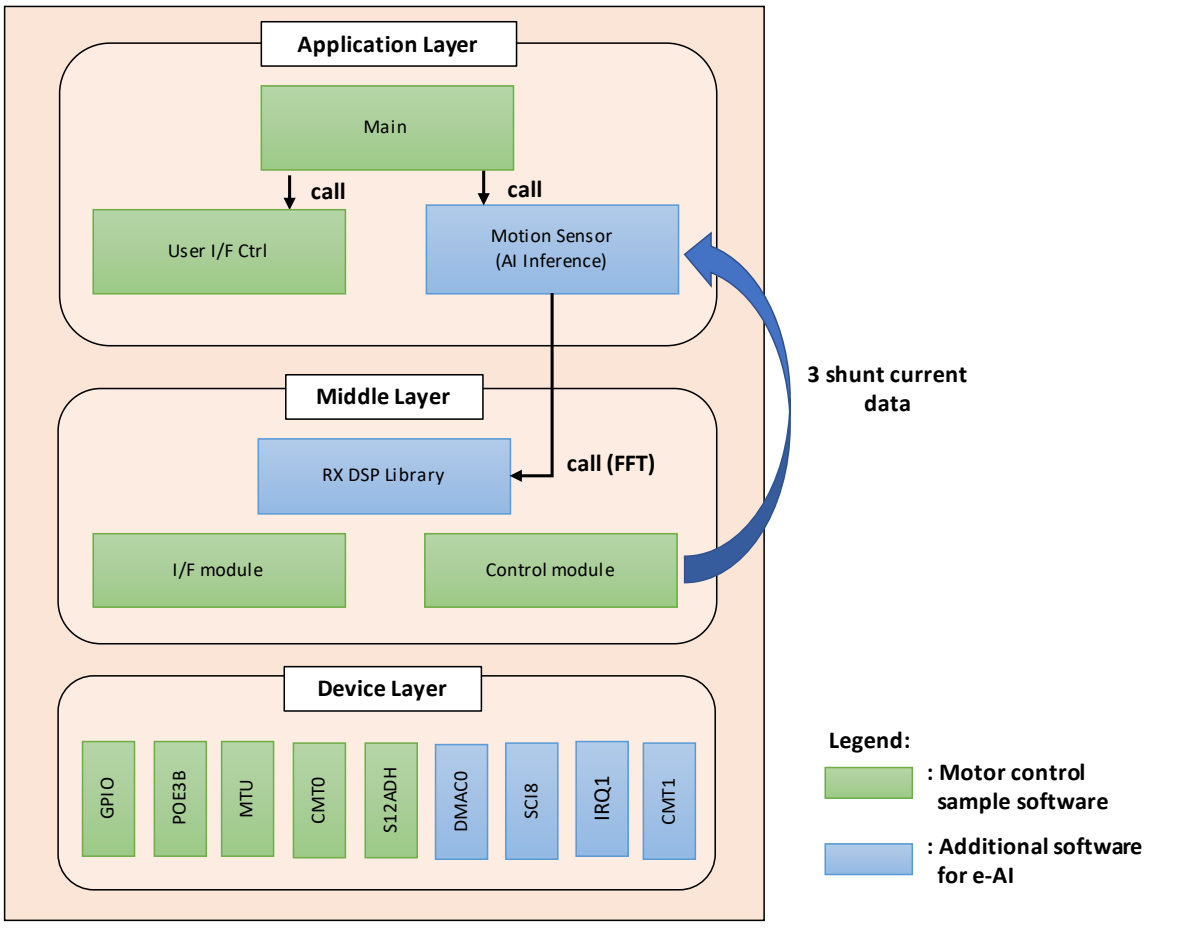
(3) ERROR

モータの過電流などエラーを検知すると LED2 が点灯し ERROR 状態に遷移します。ERROR 状態は RESET ボタンを押下するか、SW2 を ON→OFF と操作することでエラーを解除し INACTIVE/ACTIVE 状態に遷移します。

4.　MCU ソフトウェア説明

4.1　ソフトウェア構造

図 4.1 にソフトウェア構造を示します。本例は参考資料[2]「永久磁石同期モータのセンサレスベクトル制御 (RX66T 実装編)」のサンプルソフトウェアに AI 推論処理、関連ドライバおよび RX ファミリ RX DSP ライブラリを追加した構造となっています。サンプルソフトウェアに関する詳細は参考資料[2]を参照してください。



注：青枠内が新規追加部です。それ以外の詳細は参考資料[2]を参照してください。

図 4.1　ソフトウェア構造図

4.2　ファイル構成

図 4.2 にファイル構成を示します。

```
rx66t_eAI_motor_inference_sample
│
├──application
│   ├──eAI_src
│   │   │   MotionSensor.c
│   │   │   MotionSensor.h
│   │   │   r_fft_window_coefficient.h
│   │   │
│   │   └──Translator
│   │           dnn_compute.c
│   │           input_image_0.h
│   │           layer_graph.h
│   │           layer_shapes.h
│   │           network.c
│   │           Typedef.h
│   │           weights.h
│   │
│   ├──main
│   │       main.c
│   │       main.h
│   │
│   └──user_interface
│       ├──board
│       │       r_mtr_board.c
│       │       r_mtr_board.h
│       │
│       └──ics
│               ICS2_RX66T.h
│               ICS2_RX66T.lib
│               rx66t_eAI_motor_inference_sample.map
│               RX66T_MRSSK_SPM_LESS_FOC_E2S_RV100.map
│               RX66T_MRSSK_SPM_LESS_FOC_E2S_RV100.rmt
│               r_mtr_ics.c
│               r_mtr_ics.h
│
├──config
│       r_mtr_config.h
│       r_mtr_control_parameter.h
│       r_mtr_inverter_parameter.h
│       r_mtr_motor_parameter.h
│
├──driver
│   ├──inverter
│   │       r_mtr_ctrl_mrssk.c
│   │       r_mtr_ctrl_mrssk.h
│   │
│   ├──mcu
│   │   │   r_mtr_ctrl_mcu.h
│   │   │   r_mtr_ctrl_rx66t.c
│   │   │   r_mtr_ctrl_rx66t.h
│   │   │   r_mtr_interrupt.c
│   │   │
│   │   └──auto_generation
│   │           dbsct.c
│   │           hwsetup.c
│   │           intprg.c
│   │           iodefine.h
│   │           resetprg.c
│   │           sbrk.c
│   │           sbrk.h
│   │           stacksct.h
│   │           typedefine.h
│   │           vect.h
│   │           vecttbl.c
│   │
│   └──smc_gen
│       ├──Config_CMT1
│       │       Config_CMT1.c
│       │       Config_CMT1.h
│       │       Config_CMT1_user.c
│       │
│       ├──Config_DMAC0
│       │       Config_DMAC0.c
│       │       Config_DMAC0.h
│       │       Config_DMAC0_user.c
│       │
│       ├──Config_ICU
│       │       Config_ICU.c
│       │       Config_ICU.h
│       │       Config_ICU_user.c
│       │
│       ├──Config_SCI8
│       │       Config_SCI8.c
│       │       Config_SCI8.h
│       │       Config_SCI8_user.c
│       │
│       ├──general
│       │       r_cg_cmt.h
│       │       r_cg_dmac.h
│       │       r_cg_dmac_user.c
│       │       r_cg_hardware_setup.c
│       │       r_cg_icu.h
│       │       r_cg_macrodriver.h
│       │       r_cg_sci.h
│       │       r_cg_userdefine.h
│       │       r_smc_cgc.c
│       │       r_smc_cgc.h
│       │       r_smc_cgc_user.c
│       │       r_smc_entry.h
│       │       r_smc_interrupt.c
│       │       r_smc_interrupt.h
│       │
│       └──r_bsp
│           │   platform.h
│           │
│           ├──board
│           │   └──user
│           │           r_bsp.h
│           │
│           └──mcu
│               ├──all
│               └──rx66t
│                   └──register_access
│                           iodefine.h
│
└──middle
    ├──common
    │       r_mtr_common.h
    │       r_mtr_filter.c
    │       r_mtr_filter.h
    │       r_mtr_fluxwkn.h
    │       r_mtr_fluxwkn.obj
    │       r_mtr_mod.c
    │       r_mtr_mod.h
    │       r_mtr_pi_control.c
    │       r_mtr_pi_control.h
    │       r_mtr_statemachine.c
    │       r_mtr_statemachine.h
    │       r_mtr_transform.c
    │       r_mtr_transform.h
    │       r_mtr_volt_err_comp.h
    │       r_mtr_volt_err_comp.obj
    │
    ├──control
    │       r_mtr_bemf_observer.h
    │       r_mtr_bemf_observer.obj
    │       r_mtr_ctrl_gain_calc.obj
    │       r_mtr_foc_action.c
    │       r_mtr_foc_control_less_speed.c
    │       r_mtr_foc_control_less_speed.h
    │       r_mtr_foc_current.c
    │       r_mtr_foc_current.h
    │       r_mtr_foc_speed.c
    │       r_mtr_foc_speed.h
    │       r_mtr_interrupt_carrier.c
    │       r_mtr_interrupt_timer.c
    │       r_mtr_opl2less.h
    │       r_mtr_opl2less.obj
    │       r_mtr_opl_damp_ctrl.h
    │       r_mtr_opl_damp_ctrl.obj
    │       r_mtr_parameter.h
    │
    ├──dsplib-rxv3
    │       RX_DSP_FPU_LE.lib
    │       r_dsp_complex.h
    │       r_dsp_statistical.h
    │       r_dsp_transform.h
    │       r_dsp_typedefs.h
    │       r_dsp_types.h
    │       r_dsp_ver_info.h
    │
    └──interface
            r_mtr_driver_access.c
            r_mtr_driver_access.h
```

図 4.2　ファイル構成

4.3　使用リソース

4.3.1　使用リソース一覧

表 4.1 に使用リソース一覧を示します。表の太字は本例で追加した周辺機能です。太字以外の周辺機能の詳細は参考資料[2]を参照してください。

表 4.1　使用リソース一覧

項目	説明
クロック発生回路	外部発振子から動作クロックを生成
S12ADH	・3 相電流測定 ・インバータ母線電圧測定 ・回転速度指令値入力
CMT0	500 [μs]インターバルタイマ
MTU3	相補 PWM 出力
POE3B	過電流検出、出力短絡検出
GPIO	・スイッチ（SW1、SW2）入力 ・LED（LED1、LED2）点灯/消灯制御 ・シリアル通信のフロー制御信号反転出力
CMT1	**3 相電流値サンプリング周期生成 (2kHz)**
SCI8	**シリアル通信**
DMAC0	**SCI8 の転送要求発生により、送信データレジスタへ転送**
ICU（IRQ1）	**端子レベルの両エッジで割り込み発生**
I/O ポート（P31）	**IRQ1 端子のレベルを反転して出力**

4.3.2　割り込み

表 4.2 に割り込み処理一覧を示します。表の太字は本例で追加した割り込み処理です。太字以外の割り込み処理の詳細は参考資料[2]を参照してください。

表 4.2　割り込み処理一覧

割り込み 要求発生元 (周辺モジュール)	名称 (周辺モジュールの 名称)	割り込み 優先レベル （IPR）	内容
グループ割り込み 1 （POE3） **（SCI8）**	GROUPBL1 （OEI1） **（TEI8、ERI8）**	15（最高）	・ モータ割り込み処理（過電流検出） **・ PC とのデータ送受信** **- SCI8 送信完了割り込み（TEI8）** **- SCI8 受信エラー割り込み（ERI8）**
PERIA（MTU3）	INTA209（TCIV4）	12	モータ割り込み処理(500μs)
ICU	**IRQ1**	**12**	**シリアル通信のフロー制御信号レベル監視**
SCI8	**RXI8**	**12**	**PC からのデータ受信**
CMT0	CMI0	11	モータ割り込み処理(50μs)
SCI8	**TXI8**	**8**	**PC へのデータ送信（DMAC 転送のトリガ）**
DMAC0	**DMACI0**	**8**	**転送完了フラグクリア**
CMT1	**CMI1**	**5**	**3 相電流サンプリング周期（2kHz）**

以下に本例での割り込み処理変更点を示します。

① ファイル名：intprg.c
下記のコードをコメントアウトしています。

```
void Excep_CMT1_CMI1(void){ }
void Excep_SCI8_RXI8(void){ }
void Excep_SCI8_TXI8(void){ }
void Excep_DMAC_DMAC0I(void){ }
void Excep_ICU_IRQ1(void){ }
```

② ファイル名：r_mtr_interrupt.c
外部変数・外部関数の宣言、過電流検出割り込み（グループ 1 割り込み）およびモータ割り込み処理（50 μs）に処理を追加しています。

```
/* Used for e-AI inference processing */
extern float g_current = 0;
extern void r_Config_SCI8_transmitend_interrupt(void);
extern void r_Config_SCI8_receiveerror_interrupt(void);

------------------------------------------------------------------
#pragma interrupt (mtr_over_current_interrupt(vect = VECT_ICU_GROUPBL1))
static void mtr_over_current_interrupt(void)
{
  /* Overcurrent current detection */
  if (1 == ICU.GRPBL1.BIT.IS9){
     mtr_over_current(&g_st_foc);
  }

  /* SCI8 TEI8 (transmit end) */
  if (1 == ICU.GRPBL1.BIT.IS24){
    r_Config_SCI8_transmitend_interrupt();
  }

  /* SCI8 ERI8 (recieve error) */
  if (1 == ICU.GRPBL1.BIT.IS25){
    r_Config_SCI8_receiveerror_interrupt();
  }
} /* End of function mtr_over_current_interrupt */

------------------------------------------------------------------
#pragma interrupt (mtr_tciv4_interrupt(vect = VECT_PERIA_INTA208))
static void mtr_tciv4_interrupt(void)
{
   mtr_foc_interrupt_carrier(&g_st_foc);
   mtr_ics_interrupt_process();

   /* get peak current for e-AI process */
  if(g_current < g_st_foc.f4_iu_ad)
  {
   g_current = g_st_foc.f4_iu_ad;
  }
}
```

4.4　メイン処理

図 4.3 にメイン処理のフローチャートを示します。メイン処理とモータ制御処理の関係が分かるようモータ制御の割り込み処理も併記しています。

メイン処理ではシステムの初期設定を行い、ユーザインタフェース処理と AI 推論処理をループ実行します。ユーザインタフェース処理は 24V 系インバータボードの入力デバイス（SW1、SW2、VR1）を用いてシステム制御を行う処理で、入力デバイスの情報をシステム状態やモータ回転速度指令値に変換しモータ制御処理に渡します。

モータの駆動制御は 2 種類のモータ割り込み処理で実行され、システム状態が ACTIVE のときにモータ駆動することが出来ます。

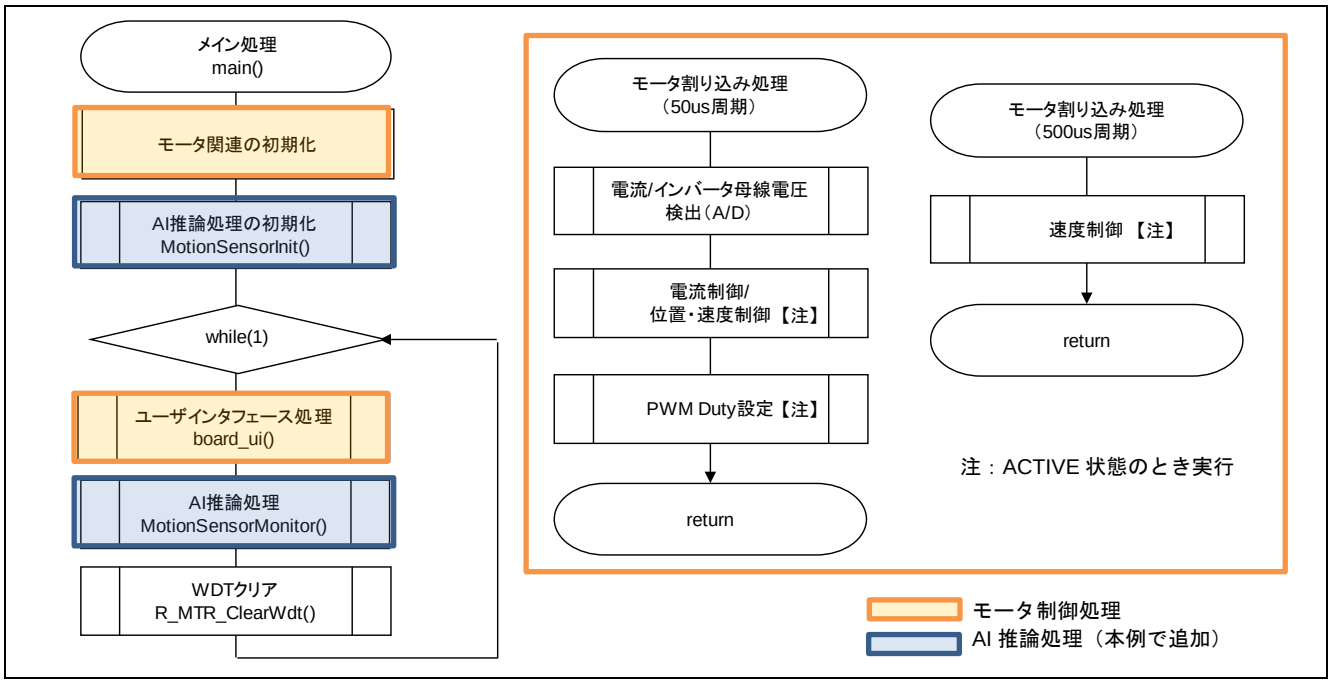


図 4.3　メイン処理のフローチャート

4.5　モータ制御処理

本節では参考資料[2]「永久磁石同期モータのセンサレスベクトル制御 (RX66T 実装編)」のサンプルソフトウェアから変更した部分を説明します。記載のないモータ制御処理の詳細は参考資料[2]を参照してください。

本例は 24V 系インバータボードのユーザインタフェースを使用するため定義を書き換えています。

対象ファイル：r_mtr_config.h

変更前：

```
#define  CONFIG_DEFAULT_UI  (ICS_UI)
```

変更後：

```
#define  CONFIG_DEFAULT_UI  (BOARD_UI)
```

本例はモータ回転速度指令値制限のためユーザインタフェース処理（board_ui 関数）のコードを書き換えています。

表 4.3 回転速度指令値の変換比

項目		変換比（指令値：A/D 変換値）
回転速度指令値	CW	1000[rpm]~2000[rpm]：08C8H~0FFFH

対象ファイル：main.c

対象関数：board_ui()

変更前：

```
/*============================*/
/*     Set speed reference    */
/*============================*/
u2_temp_vr1_signal = get_vr1();
s2_temp = (u2_temp_vr1_signal - ADJUST_OFFSET) * VR1_SCALING; /* Read speed
reference from VR1 */

s2_temp = R_MTR_LimitAbs(s2_temp, g_u2_max_speed_rpm);
R_MTR_SetSpeed(s2_temp); /* Set speed reference */
```

変更後：

```
/*============================*/
/*     Set speed reference    */
/*============================*/
u2_temp_vr1_signal = get_vr1();
s2_temp = u2_temp_vr1_signal - ADJUST_OFFSET;   /* Read speed reference from
VR1 */

/* Change VR1 A/D value to rotation speed command value of +1000 to +2000rpm
*/
/* The area near the VR1 center is the dead zone.  */
if(-200 >= s2_temp)
{
   s2_temp = (int16_t)(((0.542f * (float)s2_temp) - 891) * (-1));
}
else if(200 <= s2_temp)
{
   s2_temp = (int16_t)(0.542f * (float)s2_temp) + 891;
}
else
{
   s2_temp = 0;
}

s2_temp = R_MTR_LimitAbs(s2_temp, g_u2_max_speed_rpm);
R_MTR_SetSpeed(s2_temp); /* Set speed reference */
```

4.6　FFT 処理

3 相電流のデータを FFT 処理するため RX ファミリ用 DSP ライブラリを使用します。表 4.4 に DSP ライブラリで使用する関数を示します。

表 4.4　DSP ライブラリ使用関数

関数名	説明
R_DSP_FFT_Init_i16ci32()	実数 FFT 演算のハンドル初期化
R_DSP_FFT_i16ci32()	実数 FFT 変換(FFT 結果を複素数で出力)
R_DSP_VecCplxMag_ci32i32()	複素数振幅値取得

本例の RX DSP ライブラリを使用した FFT のソースコードを示します。

```
#include <r_dsp_transform.h>

static void fExecuteFFT(int16_t *pInBuf, float *pOutBuf)
{
    uint32_t i;
    uint16_t SamplingHalf = gv_SamplingConditions.m_SamplingCount / 2;

    gs_fft_time.data = pInBuf;

    /* Real FFT transformation (output FFT result as a complex number)  */
    R_DSP_FFT_i16ci32( &gs_fft_handle, &gs_fft_time, &gs_fft_freq);

    /*
     * Obtain complex magnitude value
     * Analysis conditions when FFT length is 512 and the sampling frequency is 2kHz
     *  - frequency ticks: 2000Hz / 512 = 3.90625Hz)
     *  - Measurable frequency: 3.90625Hz * 255 = 996.09375Hz
     */
    R_DSP_VecCplxMag_ci32i32( &gs_fft_freq, &gs_fft_mag, SamplingHalf);

    /*
     * Replace 0th element of calculation result with gs_fft_buf[0].re
     * This process is necessary to remove the influence of real number half of the FFT
length.
     */
    *(int32_t *) gs_fft_mag.data = gs_fft_buf[0].re;

    /*
     * Convert to dBFS(decibels below full scale)
     * Step1：Convert fixed point number (1Q15) to floating point number.
            Divide FFT library output result (stored in int32 gs_fft_VecCplxMagi32[]) by
(2^15-1).
            It is half the amplitude of the A/D full scale value.
     * Step2：Double the result of step1 to get full amplitude of A/D full scale value
     * Step3：Convert the result of step 2 to voltage-ratio(dB).
     * Step4：  Subtract 12bit A/D full scale voltage-ratio (72.247199 dB) to convert to
dBFS.
     * Note : The maximum magnitude that can be used for dBFS is 0dBFS.
             If a magnitude lower than this is used, a negative number will be displayed.
     */
    for (i = 0; i < SamplingHalf; i++)
    {
        if (0 != gs_fft_VecCplxMagi32[i])
        {
            pOutBuf[i] = (20.0 * log10f((float) gs_fft_VecCplxMagi32[i] / 16383)) -
gv_DecibelsBelowFullScale;
        }
        /* When the complex absolute value is 0 */
        else
        {
            pOutBuf[i] = -gv_DecibelsBelowFullScale;
        }
    }
}
```

FFT 初期化のソースコードを示します。本例ではスペクトル解析のため、窓関数（ハニング窓）を使用しています。

```
static void fInitFft (void)
{
    uint16_t Count;

    /* Initialize FFT handle data */
    gs_fft_handle.n = gv_SamplingConditions.m_SamplingCount;
    gs_fft_handle.work = NULL;
    gs_fft_handle.options = (R_DSP_FFT_OPT_SCALE | R_DSP_FFT_BIT_REVERSAL_DEFAULT |
R_DSP_FFT_OPT_TWIDDLE_DEFAULT);
    gs_fft_handle.bitrev = gs_fft_bitrev;
    gs_fft_handle.twiddles = gs_fft_twiddles;
    gs_fft_handle.window = NULL;

    gs_fft_time.n = gv_SamplingConditions.m_SamplingCount;
    gs_fft_freq.n = gv_SamplingConditions.m_SamplingCount / 2;
    gs_fft_mag.n = gv_SamplingConditions.m_SamplingCount / 2;

    /* Set window coefficient according to FFT length */
      switch(gv_SamplingConditions.m_SamplingCount)
    {
        case DEF_SamplingCount1024:
            gs_fft_handle.window = (void *) i16_hanning_fft1024;
            break;
        case DEF_SamplingCount512:
            gs_fft_handle.window = (void *) i16_hanning_fft512;
            break;
        case DEF_SamplingCount256:
            gs_fft_handle.window = (void *) i16_hanning_fft256;
            break;
        case DEF_SamplingCount128:
            gs_fft_handle.window = (void *) i16_hanning_fft128;
            break;
        default:
            gs_fft_handle.window = NULL;
            break;
    }

    /* Initialize FFT */
    R_DSP_FFT_Init_i16ci32( &gs_fft_handle);

    /* Sampling size */
    gv_SamplingSize = (gv_SamplingConditions.m_SamplingCount * 2) -
gv_SamplingConditions.m_SamplingOverLap;

    /* Overlap area of sampling buffer #2 and sampling buffer #1 start areas  */
    gv_Buffer2to1Overlap = gv_SamplingSize - gv_SamplingConditions.m_SamplingOverLap;

    /* End position of #1 frame  */
    gv_Frame1End = gv_SamplingConditions.m_SamplingCount - 1;

    /* End position of #2 frame  */
    gv_Frame2End = gv_SamplingSize - 1;

    /* (Sampling count/2)**2    */
    gv_SquareSamplingHalf = (float) ((gv_SamplingConditions.m_SamplingCount / 2)
           * (gv_SamplingConditions.m_SamplingCount / 2));

    /* Sampling frequency */
    Count = (uint16_t) (DEF_CMT_CLK8 / (float)
(gv_SamplingConditions.m_SamplingFrequency)) - 1;

    /* A/D conversion trigger timer setting */
    CMT1.CMCOR = Count;
}
```

4.7　AI 推論処理

4.7.1　フローチャート

図 4.4 に AI 推論処理のフローチャートを示します。

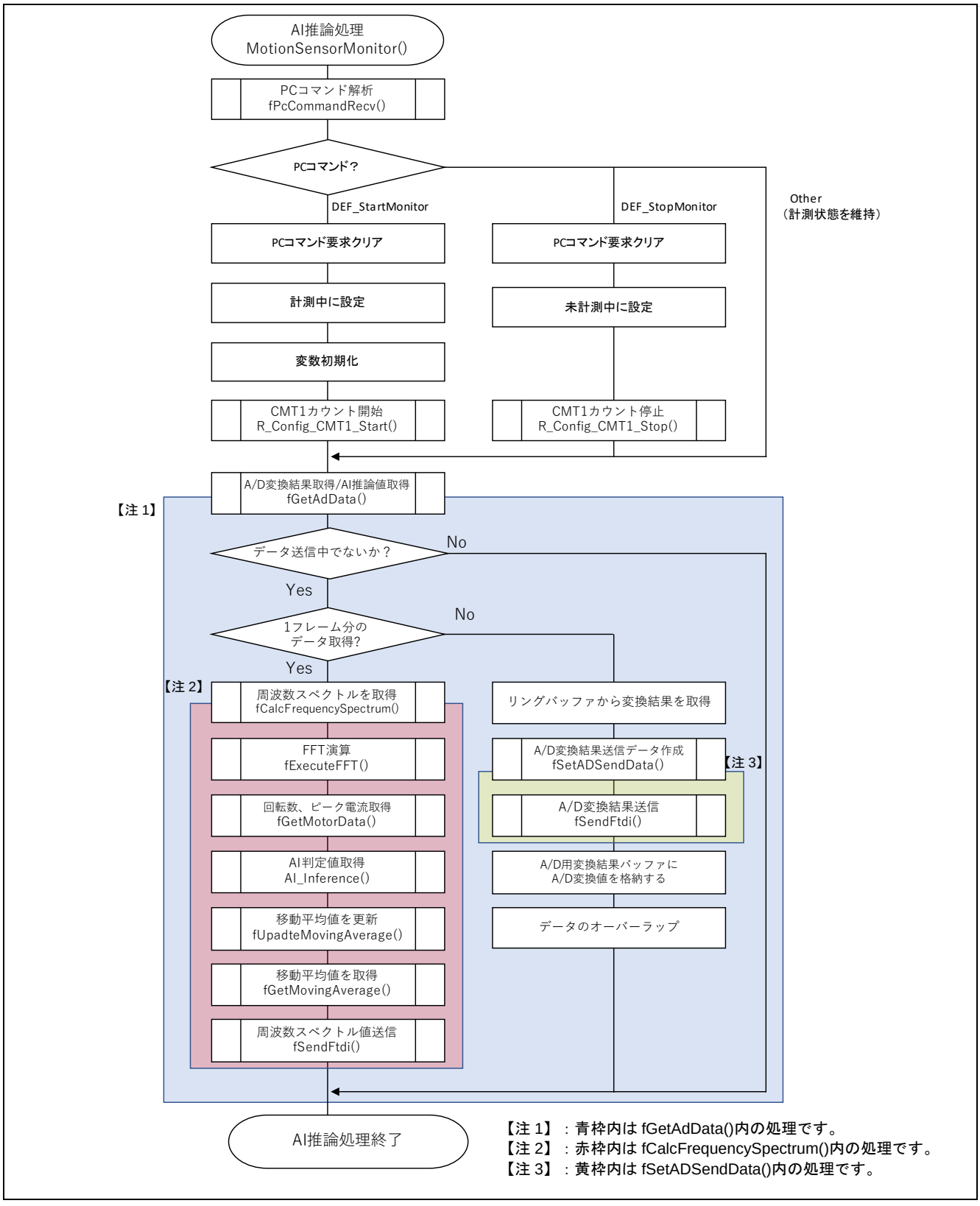


図 4.4　AI 推論処理フローチャート

図 4.5 にモータ電流値の A/D 変換結果取得フローチャートを示します。本処理は CMI1 割り込みで行います。

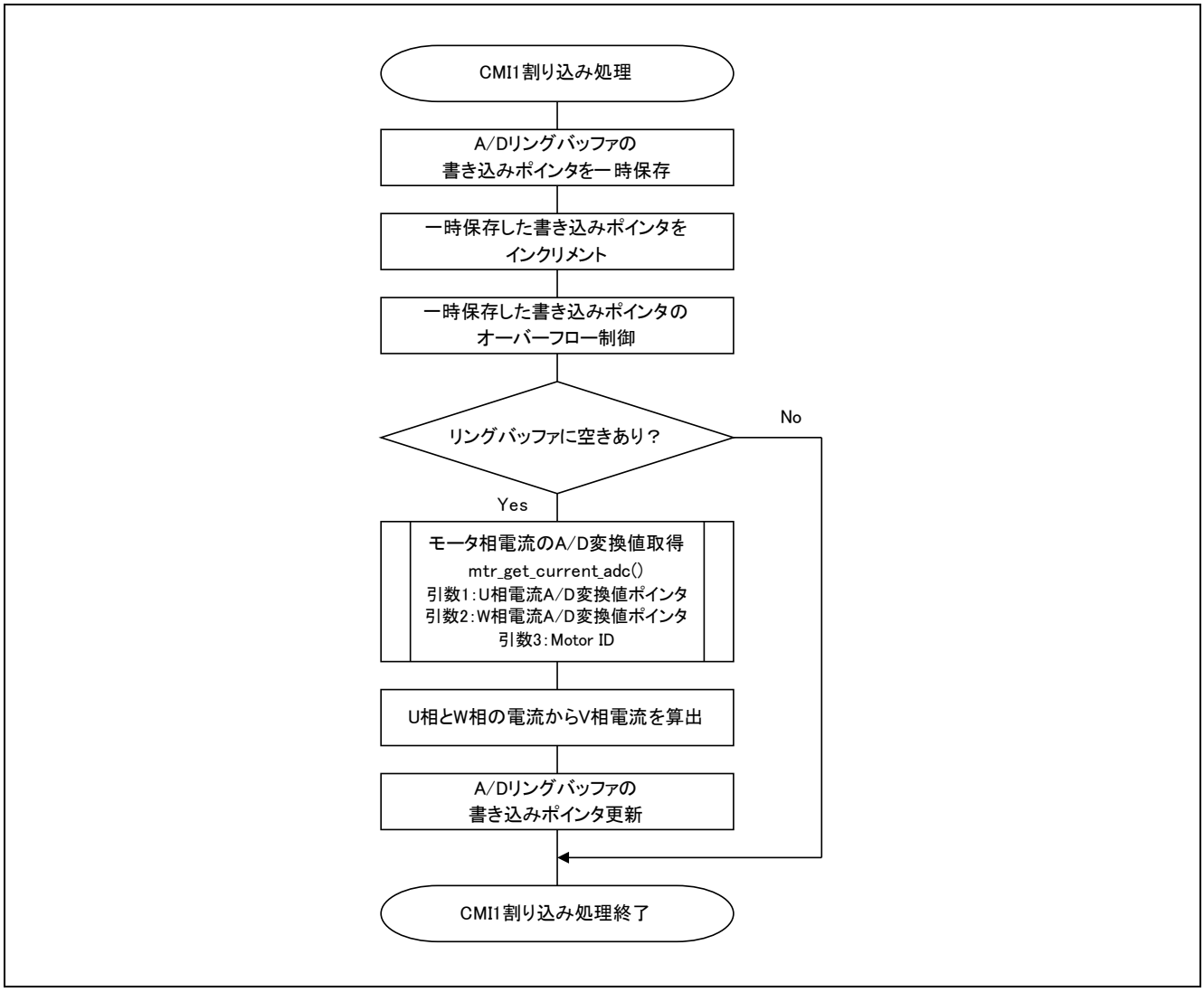


図 4.5　モータ電流値の A/D 変換結果取得フローチャート

4.7.2　データフロー

図 4.6 に AI 推論処理のデータフローを示します。

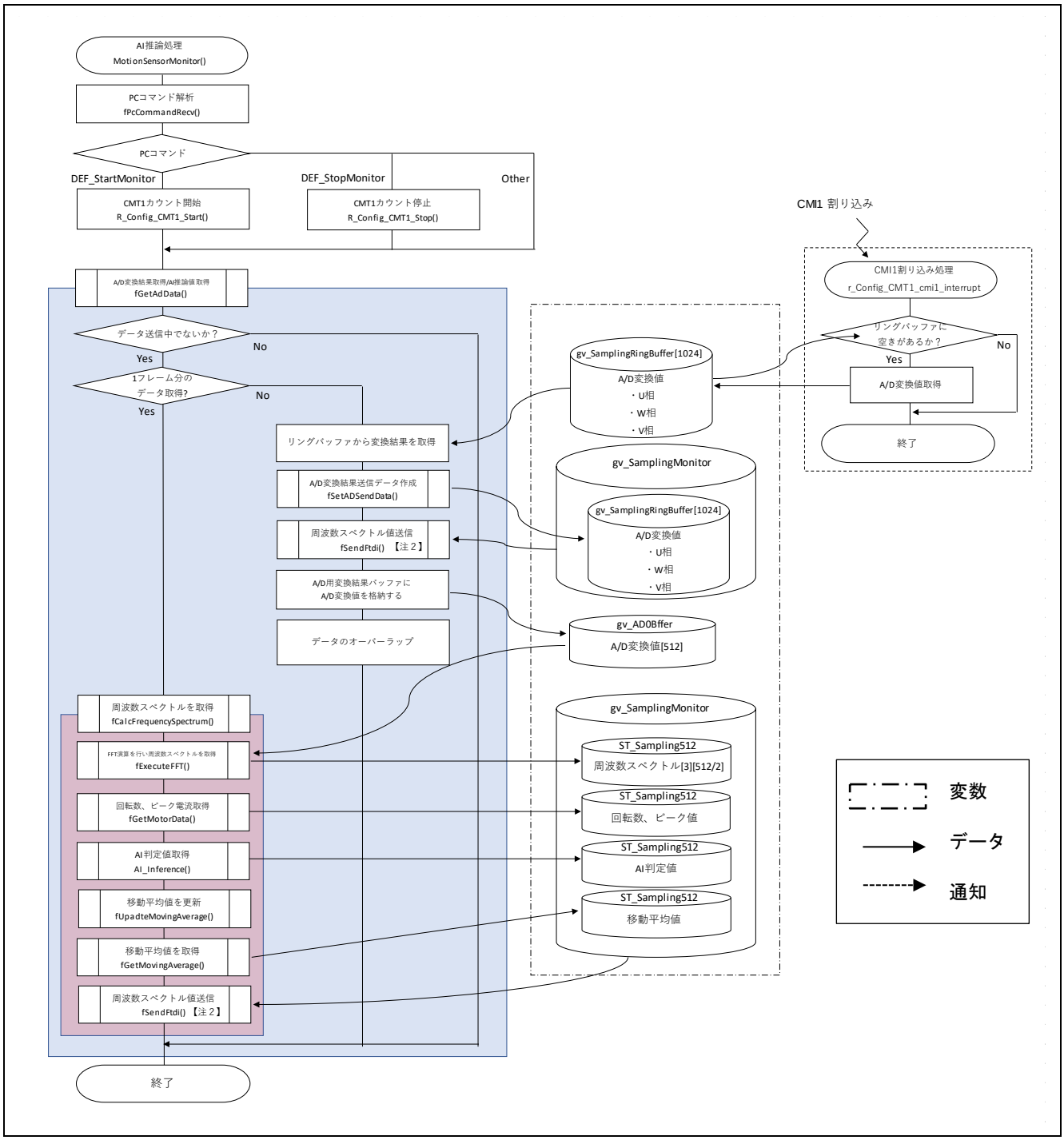


図 4.6　AI 推論処理データフロー

4.7.3　ネットワーク構造

図 4.7 に本例のネットワーク構造を示します。

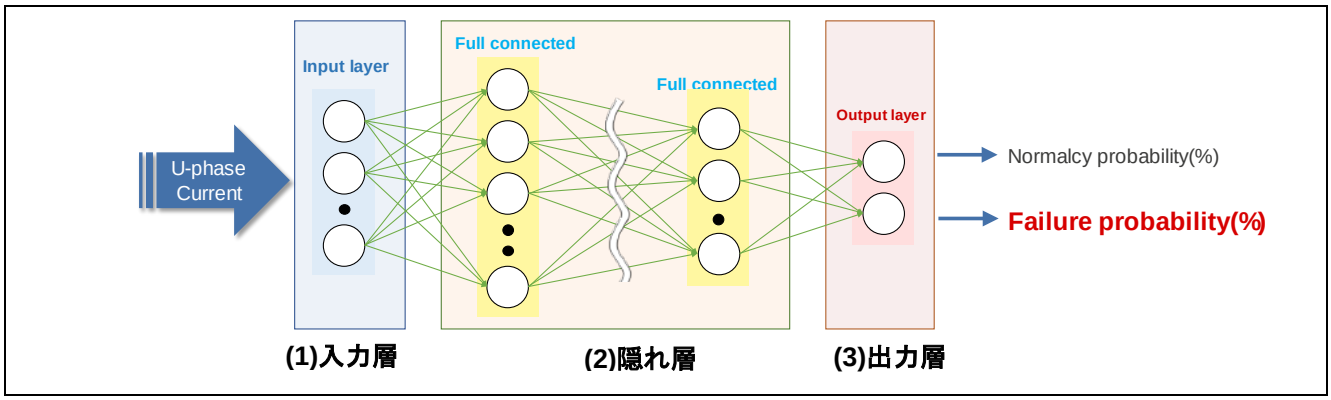


図 4.7　AI モデル構造

(1) 入力層 (Input layer)： FFT 処理後の電流データを入力します。
(2) 隠れ層 (Hidden layer)： Full Connect 層を使用しています。
(3) 出力層 (Output layer)：正常の可能性と異常の可能性の確率を出力します。

本例では図 4.8 の状態を正常、図 4.9 の状態を異常として、発生する電流波形の違いから e-AI によって異常の度合いを推論させます。本ソフトでは軸ずれを異常と定義しています。

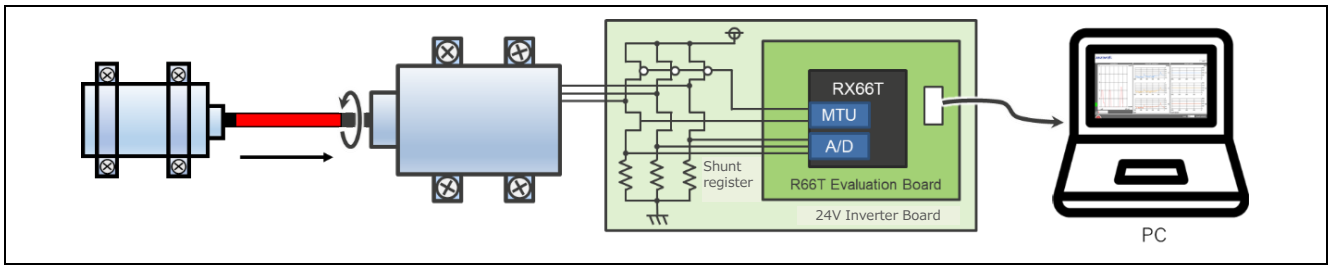


図 4.8　正常な状態

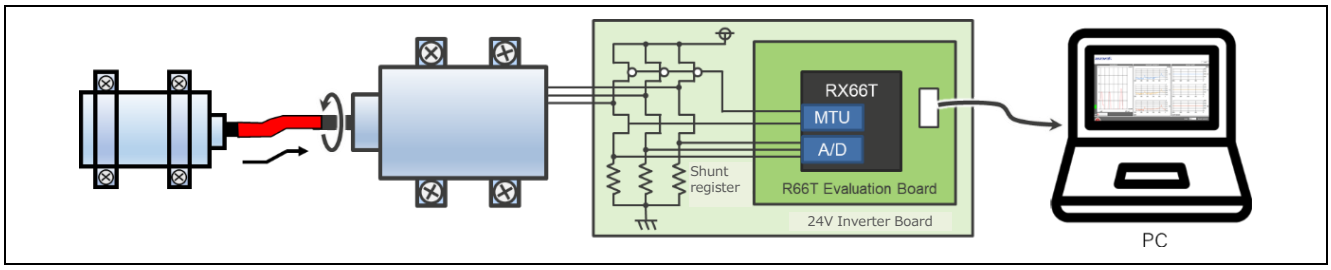


図 4.9　異常な状態

4.7.4　学習済みモデルの更新方法

学習済みモデルを更新する場合、AI フレームワークを使用して学習済みモデルを作成し、e-AI Translator を用いてソースファイルに変換してください。e-AI Translator で学習済みモデルを変換する際は図 4.2 で示す e-AI_src フォルダを変換結果出力先フォルダに指定してください。詳細手順は e-AI Translator のユーザーズマニュアルを参照してください。

e-AI Translator が出力する推論処理の API（dnn_conpute 関数）は、本例の AI_Inference 関数でコールしています。また推論処理の前に周波数スペクトルに対して AI_INPUT_SIZE で定義された範囲を抽出する前処理を行っています。AI 推論処理および前処理に関するコードはシステムに応じて変更してください。

```
static int8_t AI_Inference (uint16_t SamplingCount, float *pSpectrumX)
{
    TPrecision *p_prediction;
    int8_t AIresult;
    TsInt i;
    static float data_in[AI_INPUT_SIZE];

    /* The pre-processing function and dnn_compute function are called */
    for (i=0; i < AI_INPUT_SIZE; i++)
        {
            data_in[i] = pSpectrumX[i+1];
    }
    /* Call e-AI inference function */
    p_prediction = dnn_compute(data_in);
    AIresult = (int8_t) ((p_prediction[1] * 100) + 0.5);

    return AIresult;
}
```

5.　波形モニターツール

5.1　動作概要

　波形モニターツールは RX66T からシリアル通信で 3 相電流データや AI 推論値を送信し、その結果を表示するツールです。インストールは不要です。下記に波形モニターツールの動作概要を示します。

・表示の開始/停止の制御
・RX66T 側から送信されてきたデータの表示
　- 3 相電流の波形データ
　- 3 相電流データの FFT 後の波形データ
　- AI 推論結果
　　①移動平均波形
　　②実値バー(0~100%を 10 刻みで表示)
　　③数値
　- ピーク電流値
　- モータ回転数

5.2　機能説明

　波形モニターツールの各機能を説明します。本ツールは各種情報表示を行う「View タブ」と動作設定を行う「Setting タブ」があります。

5.2.1　View タブ

　図 5.1 に View タブの表示仕様を示します。図中の番号は後述の機能説明の番号に対応します。

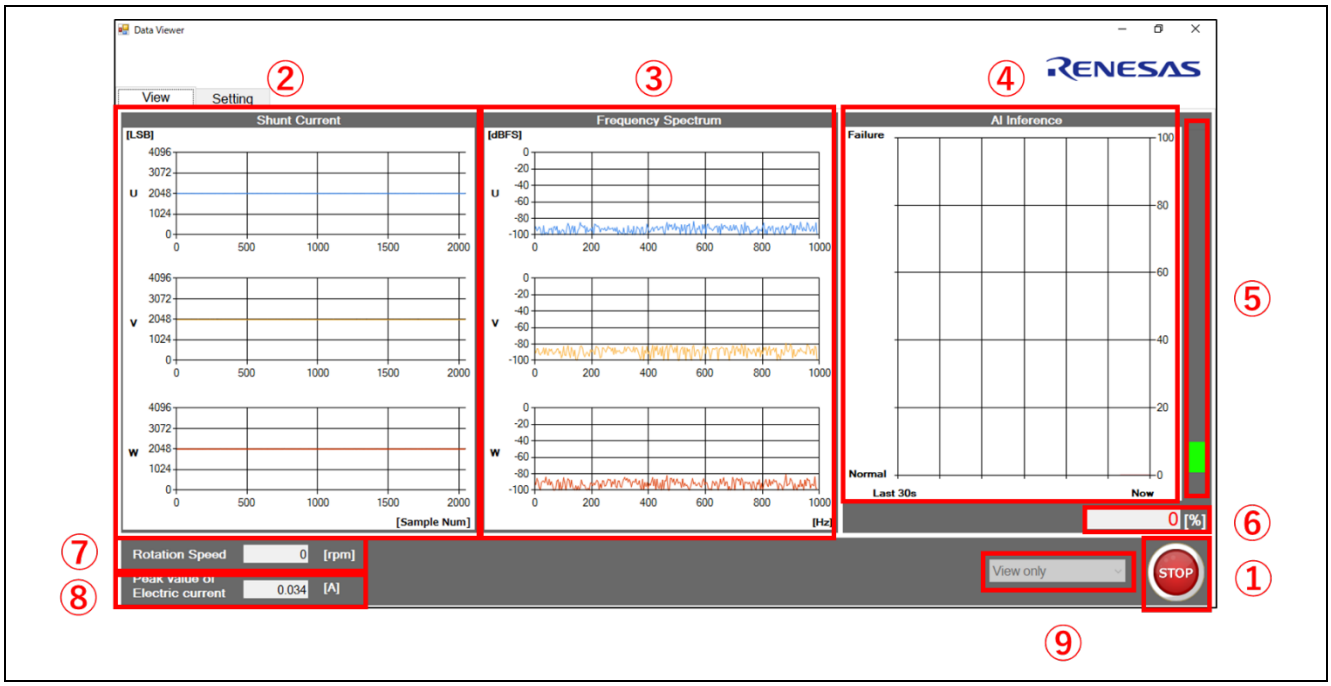


図 5.1　View タブの表示仕様

①データ取得 START／STOP ボタン

GUI ソフト起動直後は START ボタンが表示されます。それぞれの機能について下記に示します。

・START ボタン押下時
  - PC 側から RX66T 側に送信要求コマンドが送信され、データが RX66T 側から PC 側に送信されます
  - 受信したデータをリアルタイム表示します
・STOP ボタン押下時
  - PC 側から RX66T 側に送信停止コマンドが送信され、データ取得を終了します

②3 相電流の波形データ

3 相電流のサンプリングデータを U,V,W ごとにグラフで描画します。

③周波数特性

②の 3 相電流の波形データを FFT により周波数スペクトル変換し、dBFS 単位に変換したグラフを描画します。

④AI 推論結果の移動平均値波形

AI 推論で出力された異常の確率を移動平均し、波形グラフで描画します。

⑤AI 推論結果の実値バー

AI 推論で出力された異常の確率を 10%刻みの積み上げグラフで描画します。

⑥AI 推論結果の数値

AI 推論で出力された異常の確率を数値で表示します。

⑦回転数の数値

モータの回転数を数値で表示します。

⑧電流のピーク値の数値

3 相電流のピーク電流値を数値で表示します。本例では U 相電流のピーク値を表示します。

⑨ログ機能選択

ドロップダウンリストからログ（CSV ファイル）を出力するか選択します。CSV ファイルは初期設定で C ドライブ直下の「CSV Location」フォルダに格納されます。

・View only
  - 各種データのモニターのみ行います。
・Save to CSV (divided)
  - 各種データのモニターを行い、ログも出力します。本設定はデータラベル毎にファイルを出力します。データは水平方向に追記し、FFT の 1 フレーム毎に改行されます。
・Save to CSV (combined)
  - 各種データのモニターを行い、ログも出力します。本設定は各列にデータラベルが付与された単独のファイルを出力します。データは取得終了まで垂直方向に追記されます。

5.2.2　Setting タブ

図 5.2 に Setting タブの表示仕様を示します。図中の番号は後述の機能説明の番号に対応します。

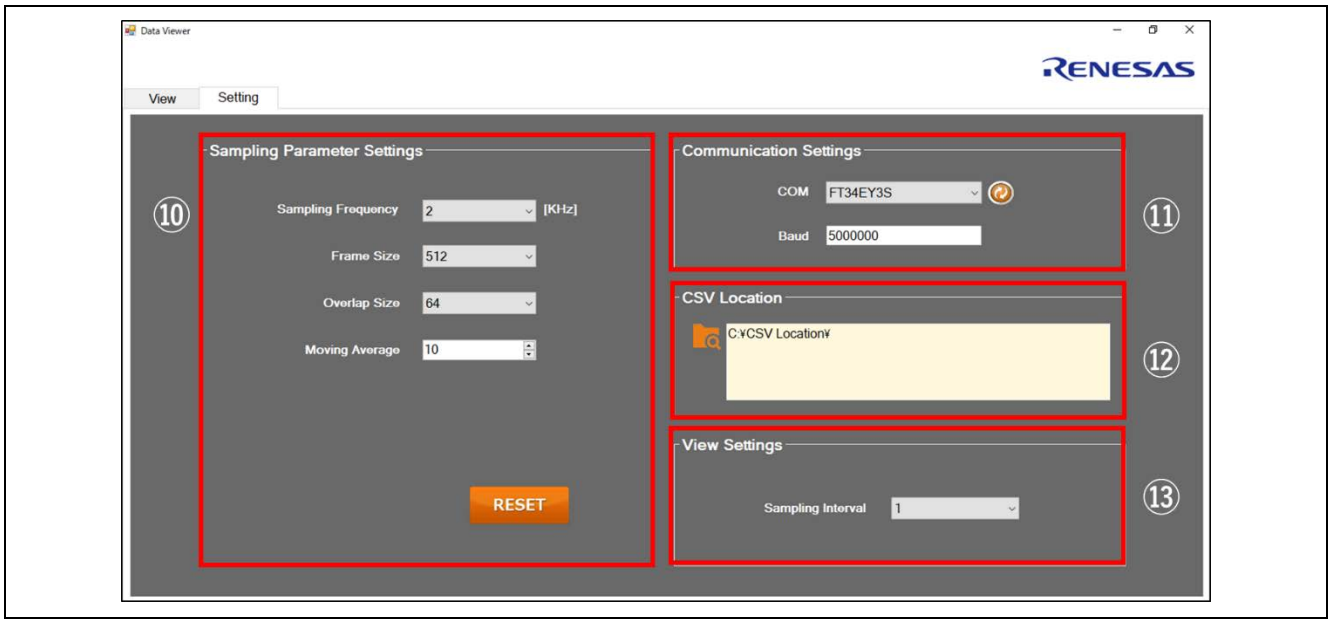


図 5.2　Setting タブの表示仕様

⑩サンプリングパラメータ設定

本例の学習済 DNN は Moving Average を除き、デフォルト設定に最適化されています。

1. Sampling Frequency
   サンプリング周波数を指定します。(1/2/4/8[kHz]、デフォルト：2[kHz])
2. Frame Size
   FFTのフレームサイズを指定します。(128/256/512/1024、デフォルト：512)
3. Overlap Size
   FFTフレームのオーバーラップ数を指定します。(16/32/64/128、デフォルト：64)
4. Moving Average
   AI推論結果のグラフ描画の移動平均長を指定します。(指定範囲：1～100回、デフォルト：10)

⑪通信設定

1. COM
   PC接続されているFTDIデバイス名を表示します。
2. Baud
   MCUとPC間の通信ボーレートを指定します。(範囲：9600～5000000、デフォルト：5000000)

⑫CSV 保存先設定

View タブでログ出力する設定にした場合の CSV ファイル出力先を指定します。

⑬描画設定

描画間引きレートを選択します。(1/2/4/8/16/32/64、デフォルト：1)

5.3　操作方法

①起動

PC に USB-シリアル変換ケーブルを接続し、図 5.3 の DataCollectionTool.exe を実行します。

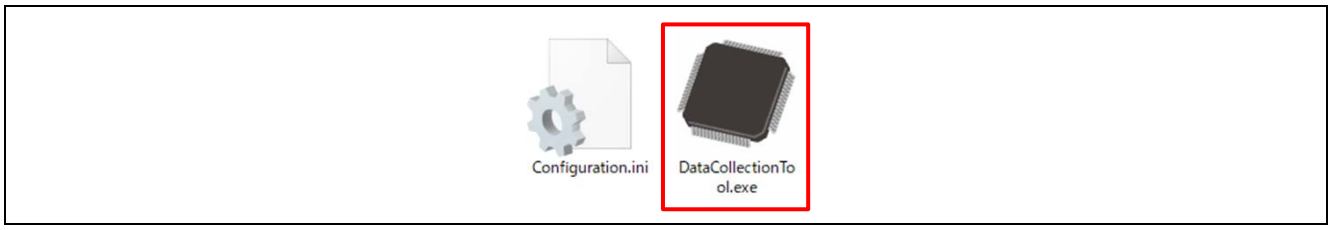


図 5.3　”DataCollectionTool.exe”ファイル

PC と USB-シリアル変換ケーブルを接続する前に exe ファイルを開くと、図 5.4 のようなエラーが表示されます。

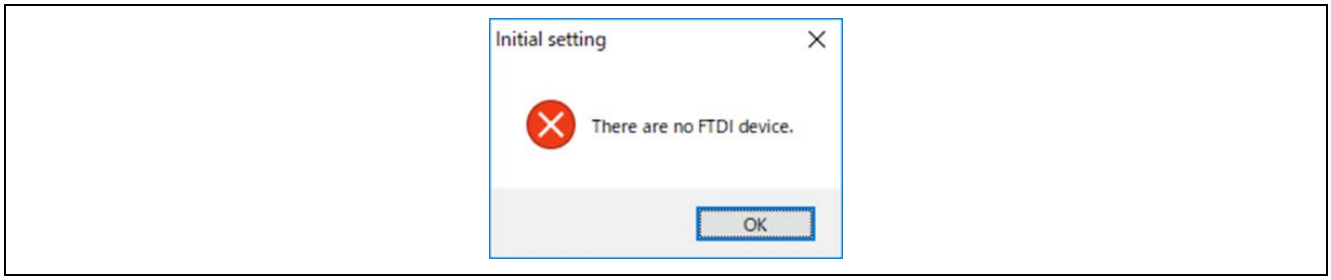


図 5.4　エラー表示

exe ファイルを実行すると図 5.5 のウインドウが表示されます。

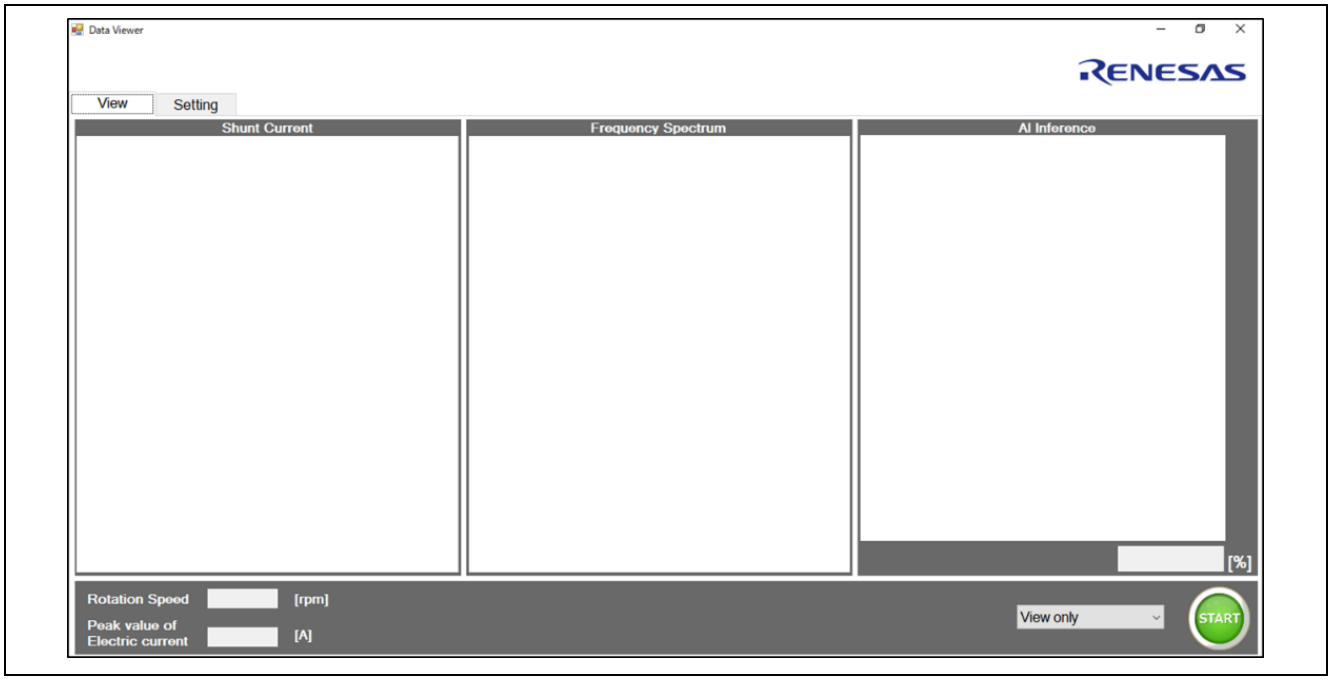


図 5.5　初期起動時画面

②データ取得開始

ウインドウ右下にある図 5.6 の START ボタンを押下するとデータ取得を開始し各データを表示します。

図 5.6　データ取得スタートボタン

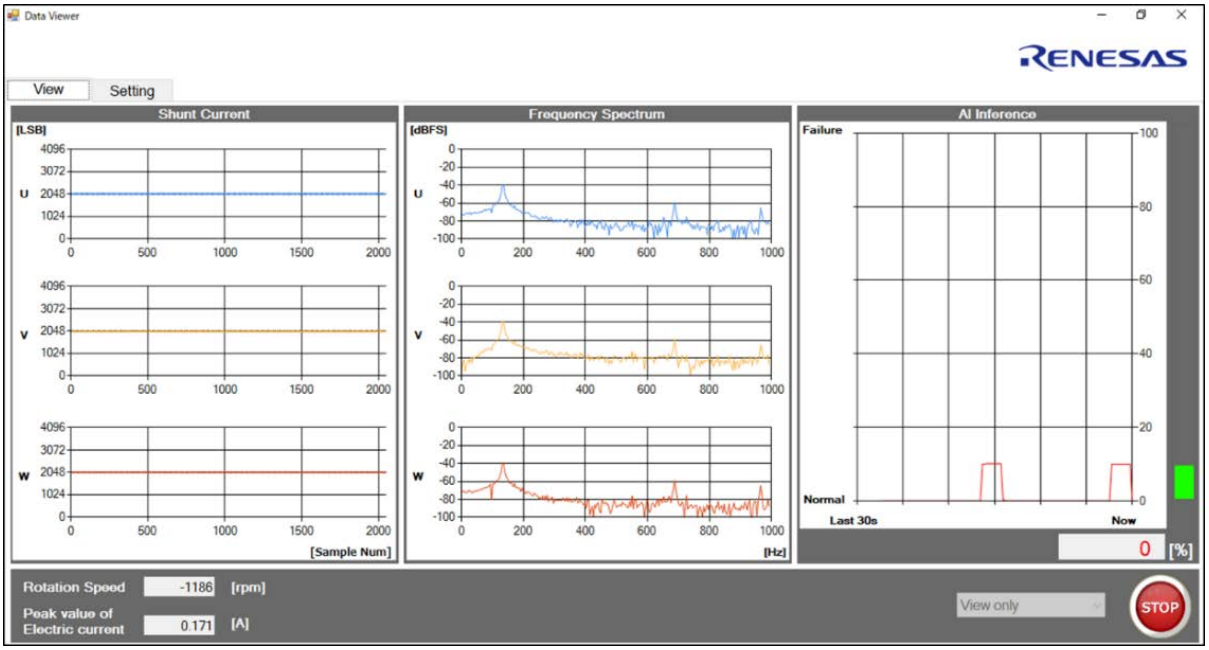


図 5.7　データ取得中の画面

③データ取得終了

ウインドウ右下にある図 5.8 の STOP ボタンを押下するとデータ取得を終了します。

図 5.8　データ取得時の画面

データ取得停止中に「5.2.1 View タブ」のドロップダウンリストで”Save to CSV(divided)”を選択すると、データ表示しながら、各軸のデータと回転数/ピーク電流についてのデータの CSV ファイル出力を行います。図 5.9 に生成されるファイルを示します

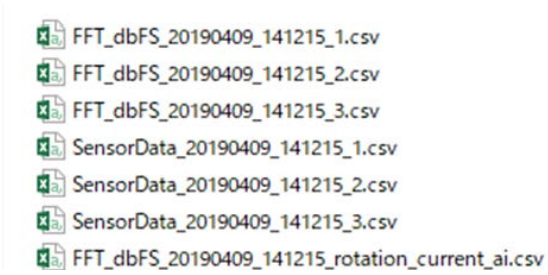


図 5.9　Save to CSV(divided)選択時に生成されるファイル

同様に”Save to CSV(combined)”を選択すると、データ表示しながら、各軸のデータを一つにまとめた CSV ファイル出力を行います。図 5.10 に生成されるファイルを示します。

図 5.10　Save to CSV(combined)選択時に生成されるファイル

6.　参考資料

[1] RX66T グループ ユーザーズマニュアル ハードウェア編 (R01UH0749)
[2] 永久磁石同期モータのセンサレスベクトル制御 RX66T 実装編(R01AN4244)
[3] Renesas Solution Starter Kit 24V Motor Control Evaluation System for RX23T(Motor RSSK) 取扱説明書 (R20UT369)
[4] RX66T CPU カード取扱説明書(R12UZ0028)
[5] RX ファミリ RX DSP ライブラリ Version 5.0 (R01AN4359)
[6] RX ファミリ アナログ入力信号を FFT するサンプルプログラム (R01AN4015)
[7] e-AI トランスレータ V1.0.2 ユーザーズマニュアル (R20UT4135)
[8] 統合開発環境 e2 studio ユーザーズマニュアル 入門ガイド (R20UT4374)
[9] Renesas Flash Programmer V3.05 フラッシュ書き込みソフトウェア ユーザーズマニュアル (R20UT4307)

付録 1.　動作確認方法

1.　ご注意事項

本章で説明する評価機器（簡易モータベンチ）は構造上長期間の動作や評価には適しません。またモータ駆動時の振動による各部の緩みや部品の離脱等の事故の恐れがあります。安全に十分配慮した上で動作確認を行うようお願い致します。

2.　評価環境

表 1 に本例開発時のハードウェア環境を示します。

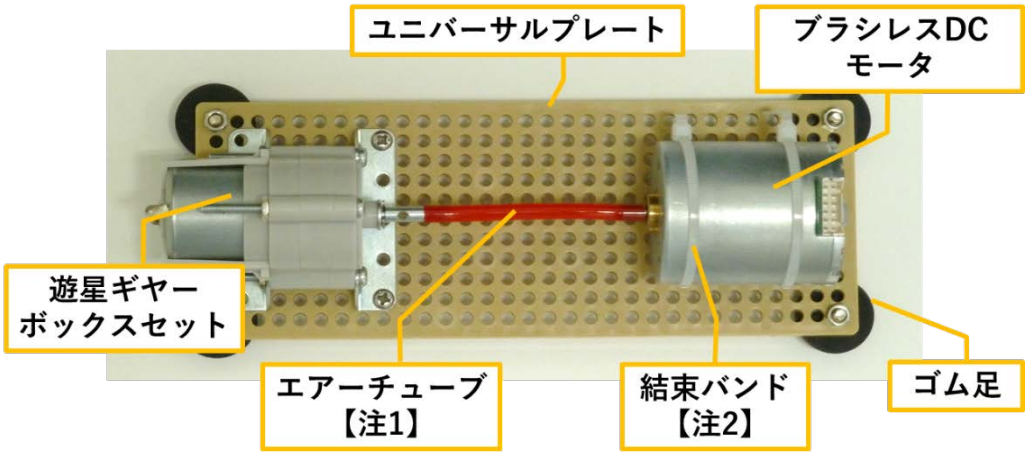
表 1.　ハードウェア環境

項目	名称	メーカ	仕様、型番等	数量
評価ボード	RX66TCPU カード	ルネサス	RTK0EMX870C00000BJ	1
	24V 系インバータ基板	ルネサス	RTK0EM0006S01212BJ	1
	24V AC アダプタ	（汎用）	出力：DC24V－2A 以上 プラグ形状：外径 5.5mm/内径 2.1mm センタープラス	1
	USB シリアル変換ケーブル【注】	FTDI	C232HM-EDHSL-0 USB Hi-Speed to MPSSE Cable	1
	ピンヘッダ	（汎用）	2.54mm ピッチ、36 極 2 列のもの	1
駆動モータ	ブラシレス DC モータ	ツカサ電工株式会社	TG-55L （RTK0EM0006S01212BJ 付属品）	1
簡易モータベンチ	ユニバーサルプレート	株式会社タミヤ	Item No:70098	1
	遊星ギヤーボックスセット	株式会社タミヤ	Item No:72001 4:1 のギア（茶色）を 1 つ使用	1
	ゴム足	（汎用）	ー	4
	エアーチューブ	（汎用）	外径 4mm、内径 2.5mm のもの 長さ 52mm にカット	1
	結束バンド	（汎用）	幅 2mm のもの	4
	ユニバーサル基板	（汎用）	2.54mm ピッチのもの	1
表示用 PC	ー	（汎用）	OS：Windows® 10 プロセッサ：1.6GHz 以上 メインメモリ：1G Byte 以上 インタフェース: USB2.0 (E2Lite および USB-シリアルケーブル接続用)	1

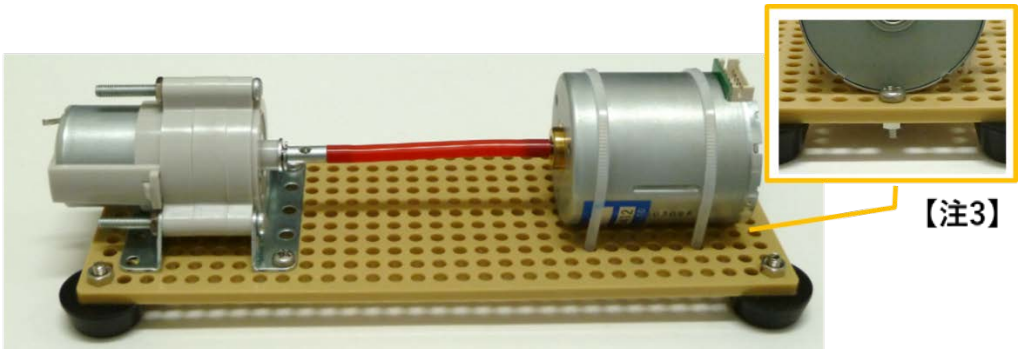
注：使用環境によっては転送データにノイズが重畳することがあります。その場合は周辺環境を整えるかケーブルを短くする等対策を行ってください。

3.　簡易モータベンチの組立て

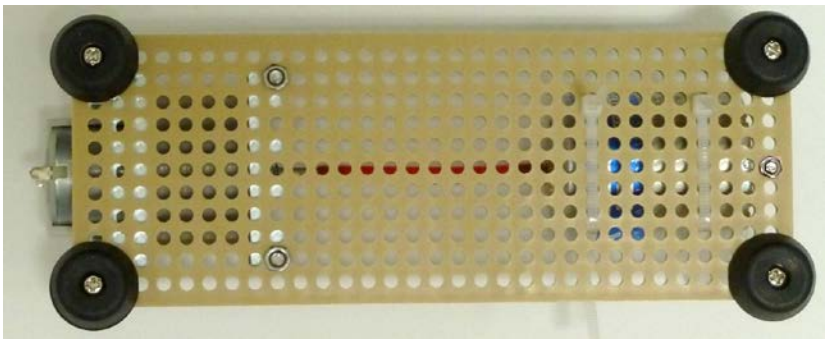
駆動モータと簡易モータベンチの組み立て方法を説明します。図 1 に完成図を示します。



a.上面



b.側面



c.底面

注 1：エアーチューブは駆動/負荷モータの根元までしっかり差し込んでください。
　　また、遊星ギヤーボックスセットを固定する前に遊星ギヤーボックスセットのシャフトにエアーチューブを付けてください。
注 2：ブラシレス DC モータが固定されるようにきつく縛ってください。
注 3：ブラシレス DC モータが動かないように、ねじをストッパーにします。

図 1.　簡易モータベンチ概観

図 2 に簡易モータベンチの部品固定位置について示します。

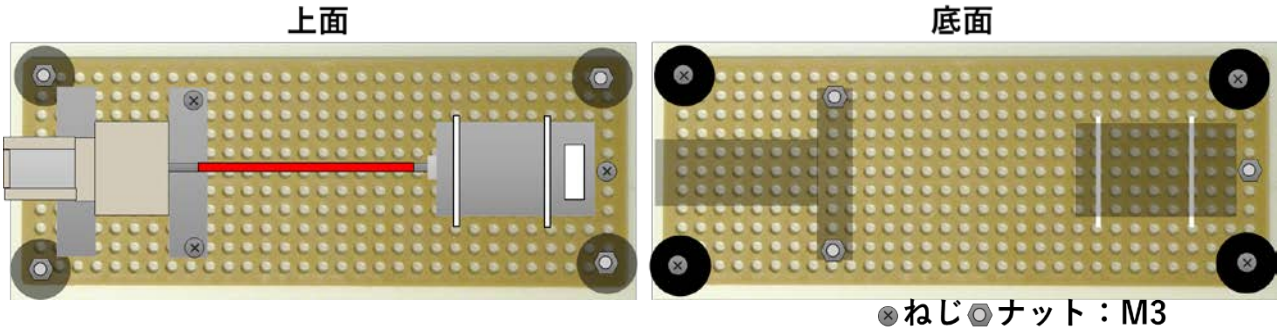


図 2.　簡易モータベンチの部品固定位置

4.　USB-シリアル変換ケーブル

RX66T と PC 間の通信には FTDI (Future Technology Devices International) 社製の USB-シリアル変換ケーブル C232HM-EDHSL-0 を使用します。

i.　ケーブル接続方法

- RX66TCPU ボードの CNC 端子にピンヘッダ（36 極、2 列）を実装してください。
- 表 2 を参考に C232HM-EDHSL-0 を CNC 端子のピンヘッダに接続してください。CNC 端子の 27 ピンと 28 ピンはジャンパブロックでショートしてください。

表 2.　USB シリアル変換ケーブルピン接続表

C232HM-EDHSL-0		CNC 端子ピン番号	
Function	Wire color	Pin No.	Function
FSCLK	Yellow	29	SCK8
FSDI	Orange	15	TXD8
FSDO	Green	35	RXD8
FSCTS	Brown	33	IRQ1
-	-	28 - 27 (Jumper short)	P31 -> CTS8#
GND	Black	31	GND

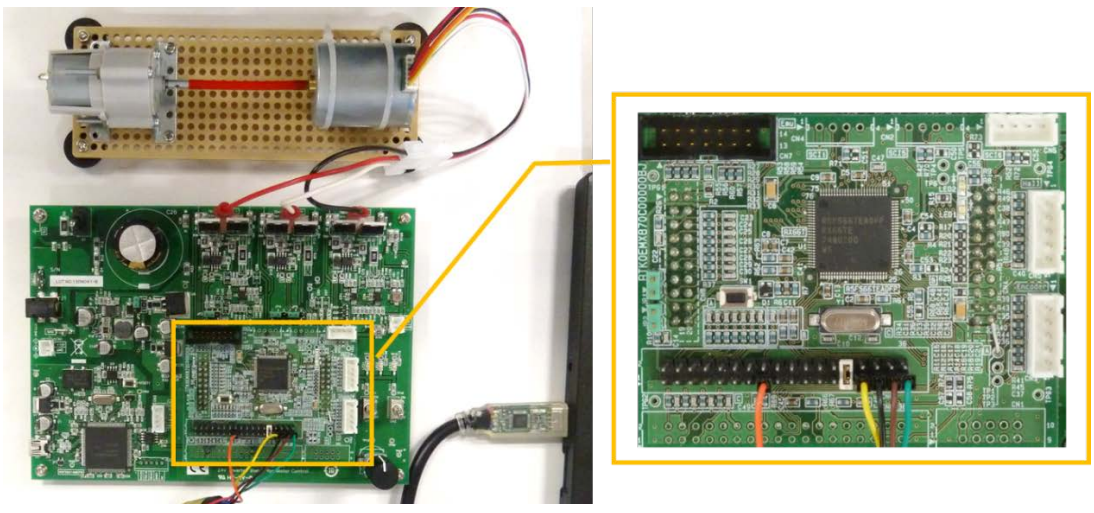


図 3.　USB シリアル変換ケーブル接続図

ii.　デバイス設定

USB シリアル変換ケーブルを使用する前に FT232HL の動作モードを変更する必要があります。以下の手順に従って動作モードを変更してください。

① FTDI 社のサイトから FT_Prog のソフトウェアをダウンロードし、インストールします。
http://www.ftdichip.com/Support/Utilities.htm#FT_Prog

② FT Prog の動作には.NET Framework 4.0 が必要である為、ダウンロードしインストールします。
https://msdn.microsoft.com/ja-jp/vstudio/ff687189.aspx

③ ダウンロード・インストールが完了したら FT_PROG.exe を実行すると図 4 の画面が表示されます。赤丸で示した虫眼鏡アイコンをクリックすると、接続されている FTDI のデバイスがリストアップされます。 ここで USB シリアル変換ケーブルを接続して、虫眼鏡アイコンを押してください。すると接続したデバイスが表示されます

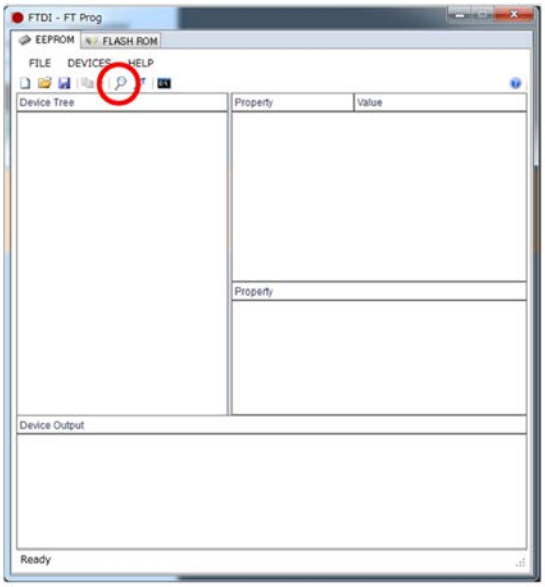


図 4.　初期起動時画面

④ 図 5 のように動作モードを[Hardware]→OPTO isolate、[Driver]→D2XX に変更してください。

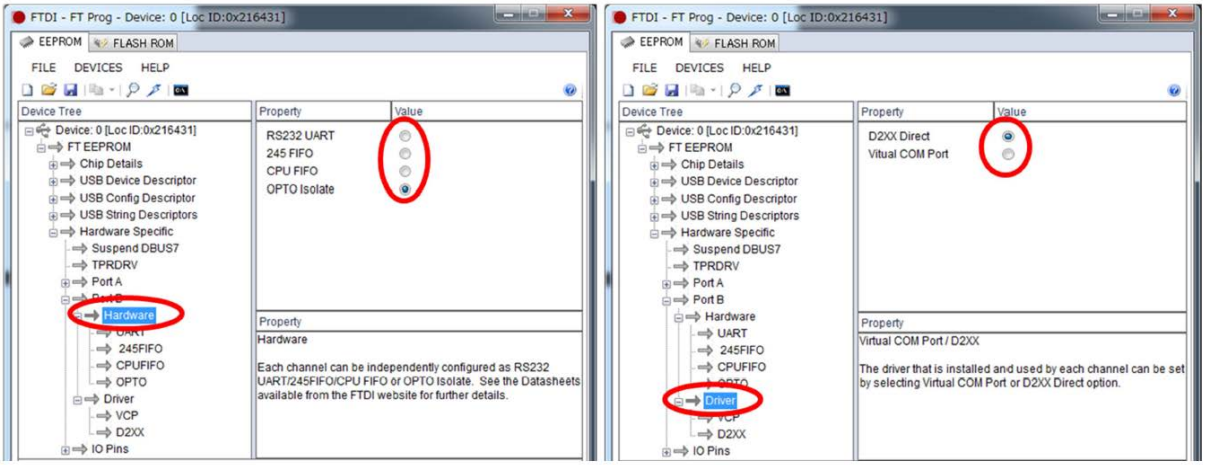


図 5.　”Hardware””Driver”設定画面

⑤　図 6 の赤丸で示した稲妻アイコンをクリックします。EEPROM 書き込みダイアログが開きます。

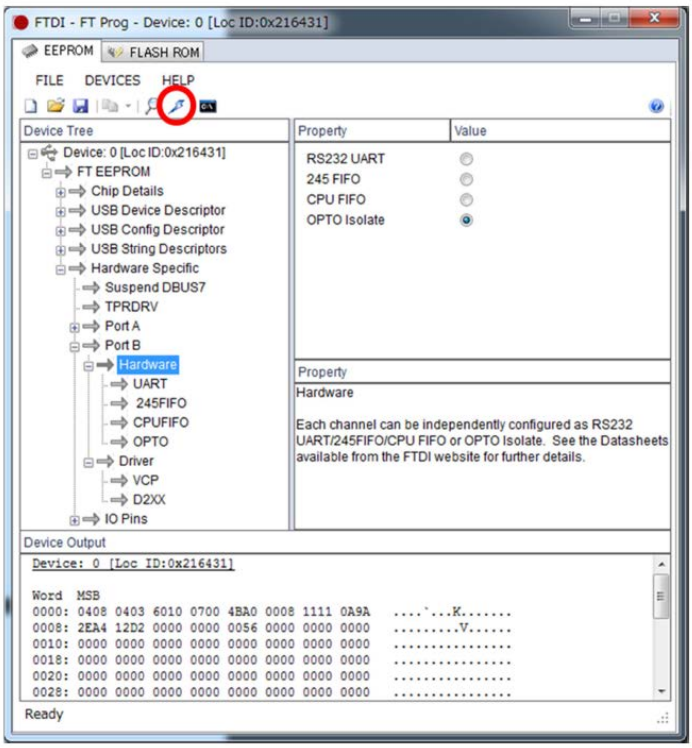


図 6.　設定画面

⑥　変更した内容を有効にするには図 7 の赤丸で示した Program を押してください。数秒で基板上の EEPROM に書き込みが行われます。

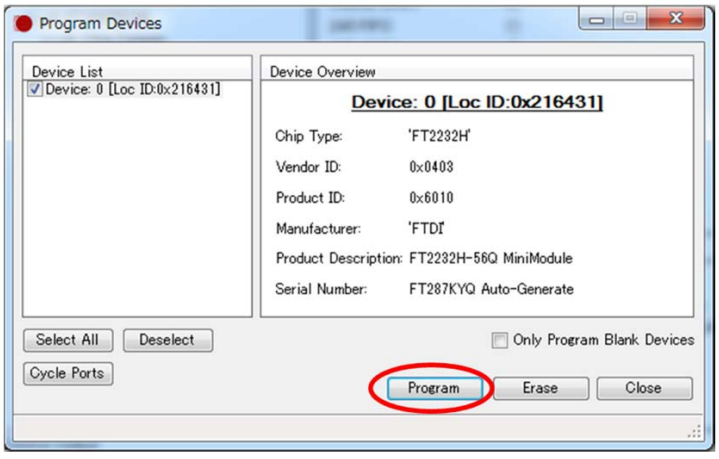


図 7.　FTDI デバイス書込画面

⑦　書き込みが終わったら、USB シリアル変換ケーブルを PC から取り外してください。再接続すると、それ以降新しい動作モードで動作します。

5.　MCU への ROM ファイル書き込み

MCU への ROM ファイル書き込み方について説明します。MCU に書き込む際には E2 Lite を使用します。E2 Lite を接続するエミュレータコネクタの位置については参考資料[4] RX66T CPU カード 取扱説明書を参照してください。

①e²studio

サンプルプロジェクトを e²studio にインポートして E2Lite 経由で書込む方法については参考資料[8] e²studio 統合開発環境ユーザーズマニュアル 入門ガイドを参照してください。

②Renesas Flash Programmer

Renesas Flash Programmer で mot ファイルを書き込む方法については参考資料[9] Renesas Flash Programmer V3.05 フラッシュ書き込みソフトウェアユーザーズマニュアルを参照してください。

MCU に書き込みを行うプログラムファイルのパスには、サンプルソフト内の DefaultBuild/[プロジェクト名].mot を指定してください。

6.　操作方法

電源投入方法とモータ制御方法、電源遮断方法については参考資料[3]を参照してください。

VR1 によるモータ回転数の制御方法を下記に示します。数字と英字は図 8 に対応します。

・回転速度

- ⓪：停止
- MIN：1000rpm
- MAX：2000rpm

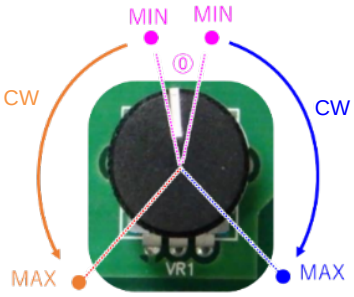


図 8.　モータ回転数の制御

下図に正常の状態と異常の状態を示します。本サンプルソフトでは軸ずれを異常と定義しています。土台に力を加え、エアーチューブが少し曲がることで軸ずれを再現しています。力が強すぎるとモータが停止してしまう可能性があるので図 10 のようにエアーチューブが少したわむように指で軽く力を加え、動作させてください。目安としてはモータが 1000rpm の時にピーク値電流が 0.3A 程に達する程度です。

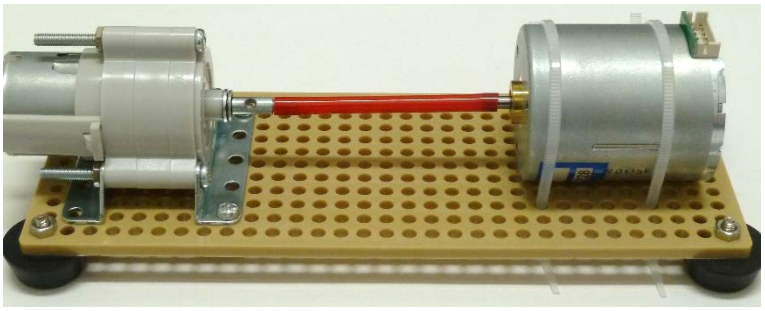


図 9.　正常の状態

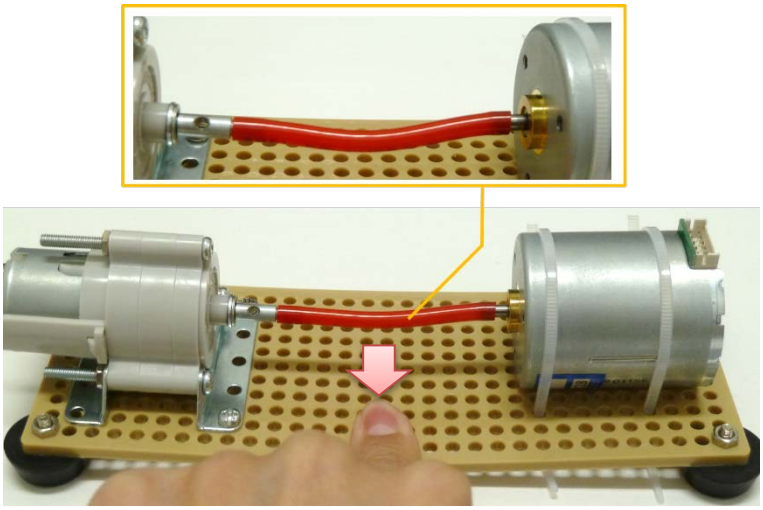


図 10.　異常の状態

7.　参考情報

簡易モータベンチ評価時の参考情報を記載します。想定の AI 推論値が得られない場合、簡易モータベンチの負荷が適正でない可能性があります。下表のピーク値電流を参考に調整してください。ピーク値電流と AI 推論値は波形モニターツールのログから 10 秒間の平均値を算出した値です。

モータ回転数(*)	シャフト負荷状態	簡易ベンチ A		簡易ベンチ B		簡易ベンチ C	
		ピーク値電流 (A)	AI 推論値 (%)	ピーク値電流 (A)	AI 推論値 (%)	ピーク値電流 (A)	AI 推論値 (%)
1000rpm	正常	0.121	0	0.130	15	0.113	13
	異常	0.306	84	0.407	100	0.285	93
1500rpm	正常	0.124	1	0.141	0	0.124	0
	異常	0.318	98	0.537	100	0.327	88
2000rpm	正常	0.135	5	0.156	6	0.131	1
	異常	0.285	95	0.531	96	0.290	96

*：波形モニターツールの表示値

付録 2.　MCU ソフトウェア詳細情報

1.　メモリ使用量

表 1.　メモリ使用量

項目	合計サイズ	詳細
RAM	40.02KB	ー
ROM	41.87KB	データセクション　　：11.36 KB プログラムセクション：30.51 KB

2.　CPU 負荷

表 2.　CPU 負荷

項目	処理時間	CPU 負荷率
モータ割り込み処理(50μs)	12.5μs	25.00%
モータ割り込み処理(500μs)	2.4μs	0.48%
FFT 処理（fExecuteFFT 関数）（注）	2128.7μs	0.95%
e-AI 推論処理（AI_Inference 関数）（注）	316.7μs	0.14%

注：割り込み処理時間を除く

3. スマート・コンフィグレータ設定一覧

i. クロック設定

表 3 にクロック設定一覧を示します。

表 3. クロック設定一覧

項目	説明
VCC	5V
メインクロック	チェックを ON
発振源	発振子
周波数	8MHz
PLL 回路　分周比	x1
PLL 回路　逓倍比	x20.0
SCKCR(FCLK[3:0])	x1/4　（FCLK=40MHz）
SCKCR(ICLK[3:0])	x1　（ICLK=160MHz）
SCKCR(PCLKA[3:0])	x1/2　（PCLKA=80MHz）
SCKCR(PCLKB[3:0])	x1/4　（PCLKB=40MHz）
SCKCR(PCLKC[3:0])	x1　（PCLKC=160MHz）
SCKCR(PCLKD[3:0])	x1/4　（PCLKD=40MHz）
SCKCR(BCK[3:0])	x1/4　（BCLK=40MHz）

注：未記載の設定は初期値を使用します。

ii. CMT1

モータの 3 相電流の A/D サンプリング周期（2kHz）を生成します。

表 4 に CMT1 の設定一覧を示します。

表 4. CMT1 の設定一覧

項目	内容
クロック設定	PCLK/8
インターバル時間	500μs
コンペアマッチ割り込みを許可（CMI1）	有効
割り込み優先順位	11

注：未記載の設定は初期値を使用します。

iii. SCI8

RX66T と PC 間でシリアル通信を行うため SCI8 を使用します。シリアル通信インタフェースには FTDI 製の USB-シリアル変換ケーブル（C232HM-EDHSL-0）を使用し、MPSSE(*1)モードで転送を行います。MPSSE モードで転送するには SCI8 は調歩同期モードに設定し、SCK8 端子からクロックを出力します。またフロー制御に CTS8#端子を使用します。

*1：Multi-Protocol Synchronous Serial Engine

表 5 に SCI8 の設定一覧を示します。

表 5. SCI8 の設定一覧

項目	内容
通信方式	SCI 調歩同期式モード
スタートビット検出	RXD8 端子の Low レベル
データビット長	9bit
パリティ	none
ストップビット	1bit
データ転送方向	LSB ファースト
転送クロック	内部クロック（PCLKB）
ビットレート	5,000,000
ビットモジュレーション機能	有効
SCK8 端子機能	クロック出力
ハードウェアフロー制御	CTS8#
送信データ処理	DMAC で処理する
受信データ処理	割り込みサービスルーチンで処理する
TXI8 優先順位	12
RXI8 優先順位	8
受信エラー割り込み許可（ERI8）	有効
TEI8, ERI8 優先順位（グループ BL1）	レベル 15 【注 2】
コールバック機能設定	送信完了、受信完了、受信エラー
使用端子	TXD8：PA4/TXD8（37 ピン） RXD8：PD1/RXD8（24 ピン） SCK8：PA3/SCK8（38 ピン） CTS8：P30/CTS8#（63 ピン）

注 1：未記載の設定は初期値を使用します。
注 2：参考資料[2]のサンプルコードで定義済みのため変更しないでください。

送信データ初期値の 9 ビット目を'0'にするため、下記のコードを追加しています。

対象ファイル：Config_SCI8_user.c（スマート・コンフィグレータで出力）

対象関数：R_Config_SCI8_Create_UserInit 関数（スマート・コンフィグレータで生成）

```
void R_Config_SCI8_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Set 9th bit to “0” */
    SCI8.TDRHL.BYTE.TDRH = 0xFE;
    /* End user code. Do not edit comment generated here */
}
```

iv. DMAC0

表 6 に DMAC0 の設定一覧を示します。

表 6. DMAC0 の設定一覧

項目	設定値
起動要因	SCI8（TXI8）に設定
起動要因フラグ制御	転送開始時に起動要因となった割り込みフラグをクリアする
転送モード	ノーマルモード
転送データサイズ	8 ビット
転送回数	1 （アプリケーション内で転送時に設定）
転送元アドレス	0x00000000 （アプリケーション内で転送時に設定）
転送元アドレス更新方法	インクリメント
転送先アドレス	0x00000000 （別途ユーザ初期設定部にて設定する）
転送先アドレス更新方法	アドレス固定
割り込み設定（DMAC0I）	転送終了割り込みを許可する
割り込み優先順位	8

注：未記載の設定は初期値を使用します。

DMAC0 の転送先アドレスに SCI8 の送信レジスタを設定するため、下記のコードを追加しています。

対象ファイル：Config_DMAC0_user.c（スマート・コンフィグレータで出力）

対象関数：R_ConfigCMAC0_Create_UserInit 関数（スマート・コンフィグレータで生成）

```
void R_Config_DMAC0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    DMAC0.DMDAR = (void *)&SCI8.TDRHL.BYTE.TDRL;
    /* End user code. Do not edit comment generated here */
}
```

v.　IRQ1

表 7 に IRQ1 の設定一覧を示します。

表 7.　IRQ1 の設定一覧

項目	設定値
検出タイプ	両エッジ
デジタルフィルタ	無効
優先順位	レベル 12
使用端子	PE4/IRQ1（8 ピン）

注：未記載の設定は初期値を使用します。

IRQ1 端子のレベルを反転して I/O ポートから出力するため下記のコードを追加しています。

対象ファイル：Config_ICU_user.c（スマート・コンフィグレータで出力）

①対象関数：R_ConfigCMAC0_Create_UserInit 関数（スマート・コンフィグレータで生成）

```
void R_Config_ICU_Create_UserInit (void)
{
    /* Start user code for user init. Do not edit comment generated here */

    /* Initial setting of GPIO used for FSCTS signal inversion */
    /* PE4/IRQ1(IN) -> ~P31(OUT) -> P30/CTS8#(IN) */
    PORT3.PODR.BIT.B1 = ~PORTE.PIDR.BIT.B4;

    /* Set P31 to the output port */
    PORT3.PDR.BIT.B1 = 1;

    /* End user code. Do not edit comment generated here */
}
```

②対象関数：r_Config_ICU_irq1_interrupt 関数（スマート・コンフィグレータで生成）

```
static void r_Config_ICU_irq1_interrupt(void)
{
    /* Start user code for r_Config_ICU_irq1_interrupt. Do not edit comment
generated here */

    /* Invert the FSCTS signal */
    PORT3.PODR.BIT.B1 = ~PORTE.PIDR.BIT.B4;

    /* End user code. Do not edit comment generated here */
}
```

4. 関数一覧

表 8 にメイン処理使用関数一覧を示します。太字は本例にて追加または変更した関数です。

表 8. メイン処理使用関数一覧

関数名	説明
clrpsw_i()	割り込み禁止
R_MTR_InitHardware()	モータ周辺機能の初期化
R_MTR_InitBoardUi()	ユーザインタフェースの初期化
software_init()	変数の初期化
R_MTR_InitControl()	三相ベクトルモータ制御を初期化
ics2_init()	" Renesas Motor Workbench"ツールを使用するための初期化
R_MTR_ExecEvent()	FOC 制御のイベント実行
R_Systeminit()	センサ周辺機能の初期化
setpsw_i()	割り込み許可
R_MTR_ChargeCapacitor()	母線電圧の安定待ち
MotionSensorInit()	**AI 推論処理の初期化**
board_ui()	**モータ制御処理**
ics_ui()	" Renesas Motor Workbench"ツールを使用する処理
MotionSensorMonitor()	**AI 推論処理**
R_MTR_ClearWdt()	WDT クリア

表 9 にモータ制御処理使用関数一覧を示します。

表 9. モータ制御処理使用関数一覧

関数名	説明
R_MTR_GetStatus()	モータの状態を取得
get_sw1()	SW1 の状態を取得
get_vr1()	VR1 の状態を取得
R_MTR_SetSpeed()	回転速度指令値の設定
get_sw2()	SW2 の状態を取得

表 10 に AI 推論処理使用関数一覧を示します。

表 10. AI 推論処理使用関数

関数名	説明
fGetAdData()	A/D 変換値取得
fSetADSendData()	変換結果送信データ作成
fCalcFrequencySpectrum()	FFT 演算を行い、周波数スペクトルを取得
fExecuteFFT()	FFT 演算の実行
fInitFft()	FFT 演算の初期化
fGetMotorData()	回転数、ピーク電流取得
fSendFtdi()	データを PC に送信
fPcCommandRecv()	PC コマンドの解析
fPcGetData()	PC コマンドの取得
fUpadteMovingAverage()	AI 推論値移動平均データ更新
fGetMovingAverage()	AI 推論値移動平均データ取得
fGetSamplingMode	サンプリングモード受信処理

5. 定数一覧

表 11 に AI 推論処理に用いる定数一覧を示します。青色の網掛けの箇所は DSP ライブラリを使用した FFT 処理に使用する定数です。

表 11. 定数一覧(MotionSensor.h)

MotionSensor.h		
定数名	設定値	説明
DEF_PCLKB	40000000.0	PCLKB の設定値 (単位：MHｚ)
DEF_CMT_CLK8	DEF_PCLKB / 8.0	CMT1 の動作クロック
DEF_DefaultSamplingFrequency	2000	サンプリング周波数の初期値 （単位：Hz）
DEF_MinSamplingFrequency	1000	最小サンプリング周波数 （単位：Hz）
DEF_MaxSamplingFrequency	8000	最大サンプリング周波数 （単位：Hz）
DEF_DefaultSamplingCount	512	サンプリング数初期値
DEF_SamplingCount128	128	サンプリング数(128)
DEF_SamplingCount256	256	サンプリング数(256)
DEF_SamplingCount512	512	サンプリング数(512)
DEF_SamplingCount1024	1024	サンプリング数(1024)
DEF_MaxSamplingCount	DEF_SamplingCount1024	最大サンプリング数
DEF_MinSamplingCount	DEF_SamplingCount128	最小サンプリング数
DEF_DefaultOverlapSampling	64	オーバーラップ数の初期値
DEF_MinOverlapSampling	16	最小オーバーラップ数
DEF_MaxOverlapSampling	128	最大オーバーラップ数
DEF_FFT_NUM_BITREV	240	ビット反転テーブルの要素数
DEF_SamplingRingBuffer	1024	サンプリング用リングバッファのサイズ
DEF_HeaderSize	7	送信データのヘッダ部のバイト数
DEF_SendAdMonitor	1	“A/D 変換値”送信コマンド
DEF_SendFrequencyMonitor	2	“周波数スペクトル”送信コマンド
DEF_StartMonitor	1	“モニター開始”受信コマンド
DEF_StopMonitor	2	“モニター停止”受信コマンド
DEF_SetUpSampling	3	“サンプリング数設定”受信コマンド
DEF_PcRecvBuffSize	128	PC からの受信バッファサイズ
DEF_PcRecvTimeOut	100	PC からの受信タイムアウト時間 （単位：10msec）
DEF_Chattering	20	SW 用チャタリング除去時間(200ms)
DEF_MovingAverageMinCount	1	AI 推論値移動平均最小数
DEF_MovingAverageMaxCount	100	AI 推論値移動平均最大数
DEF_DefaultMovingAverageMaxCount	20	AI 推論値移動平均回数の初期値
DEF_DispInterval	50	7SEGLED 表示間隔(500ms)
DEF_ShuntCurrentSampling	1	電流モード

6. 変数一覧

表 12 に AI 推論処理に用いるグローバル変数一覧を示します。青色の網掛けの箇所は DSP ライブラリを使用した FFT 処理に使用する変数です。

表 12. グローバル変数一覧(MotionSensor.c)

MotionSensor.c		
型名	変数名	説明
ST_SamplingRingBuffer	gv_SamplingRingBuffer	A/D サンプリング用リングバッファ
uint16_t	gv_SamplingCounter	A/D サンプリングカウンタ
int16_t	gv_FrameSamplingEnd	A/D サンプリングバッファの 1 フレーム分取得終了位置通知
int16_t	gv_AD0Bffer[(DEF_MaxSamplingCount * 2) - DEF_MinOverlapSampling]	A/D 変換結果(U 相)バッファ
int16_t	gv_AD1Bffer[(DEF_MaxSamplingCount * 2) - DEF_MinOverlapSampling]	A/D 変換結果(V 相)バッファ
int16_t	gv_AD2Bffer[(DEF_MaxSamplingCount * 2) - DEF_MinOverlapSampling]	A/D 変換結果(W 相)バッファ
int16_t	gv_RspiSendStatus	FTDI 送信ステータス
int32_t	gs_fft_VecCplxMagi32[DEF_MaxSamplingCount/2]	複素数振幅値格納バッファ
cplxi32_t	gs_fft_buf[(DEF_MaxSamplingCount/2)]	"R_DSP_FFT_i16ci32"関数の演算結果を格納する領域(サンプリング数/2)
vector_t	gs_fft_time = {DEF_MaxSamplingCount, NULL}	"R_DSP_FFT_i16ci32"関数の引数(入力バッファを示す)
vector_t	gs_fft_freq = {(DEF_MaxSamplingCount/2), gs_fft_buf}	"R_DSP_FFT_i16ci32"関数の引数(出力バッファを示す)
vector_t	gs_fft_mag = {(DEF_MaxSamplingCount/2), gs_fft_VecCplxMagi32}	"R_DSP_VecCplxMag_ci32i32"関数の引数(出力バッファを示す)
r_dsp_fft_t	gs_fft_handle = {DEF_MaxSamplingCount, 0};	FFT 関数のハンドル
int16_t	gs_fft_twiddles[(DEF_MaxSamplingCount+(DEF_MaxSamplingCount/2))]	回転因子バッファ
Int32_t	gs_fft_bitrev[DEF_FFT_NUM_BITREV]	ビットリバースバッファ
float	gv_DecibelsBelowFullScale	A/D コンバータフルスケール値（単位：dB） 周波数変換計算の作業用バッファ
float	gv_SquareSamplingHalf	(サンプリング数/2)^2 の値 周波数変換計算の作業用バッファ
int16_t	gv_MonitorStatus	計測開始/停止の状態
ST_SamplingMonitor	gv_SamplingMonitor	モニター送信データ(周波数スペクトルなどのデータを格納)
uint8_t	gv_RecvData	受信データ(初期化の際に受信できるか確認するためデータ)

ST_PcRecvRingBuffer	gv_PcRecvRingBuffer	PC からの受信リングバッファ
uint16_t	gv_PcRecvStatus	PC からの受信ステータス
uint8_t	gv_PcCommand	PC からの受信コマンド
uint32_t	gv_PcRecvTimer	PC からの受信監視タイマ
ST_SamplingConditions	gv_SamplingConditions	サンプリング条件
uint16_t	gv_SamplingSize	サンプリングバッファサイズ
uint16_t	gv_Buffer2to1Overlap	サンプリングバッファのオーバーラップサンプル数を格納
uint16_t	gv_Frame1End	偶数フレーム終了位置を格納
uint16_t	gv_Frame2End	奇数フレーム終了位置を格納
ST_MovingAverage	gv_MovingAverage	AI 推論値移動平均算出の作業用バッファ

下記に本サンプルソフトで流用する e-AI サンプルコードで使用する構造体一覧を示します

```
/* A/D sampling buffer structures */
typedef struct {
  int16_t             m_Ad0Value; /* A/D conversion result (U phase) buffer */
  int16_t             m_Ad1Value; /* A/D conversion result (V phase) buffer */
  int16_t             m_Ad2Value; /* A/D conversion result (W phase) buffer */
} ST_SamplingRingData;

/* A/D ring buffer structures */
typedef struct {
  uint16_t               m_Write; /* Ring buffer write position */
  uint16_t               m_Read;     /* Ring buffer read position */
  ST_SamplingRingData       m_SamplingRingData[DEF_SamplingRingBuffer]; /* Ring
buffer */
} ST_SamplingRingBuffer;

/* Information structures (frame length 128 samples) */
typedef struct {
  float                  m_FrequencySpectrum[3][DEF_SamplingCount128/2]; /*
Stores frequency spectrum */
  float                  m_RotationSpeed;  /* Stores rotation speed */
  float                  m_PeekCurrent;    /* Stores peak current value */
  int8_t                 m_AiResult;       /* Stores AI inference value (%)*/
  float                  m_AiMovingAverage;/* Stores AI inference value
(%)moving average  */
} ST_Sampling128;

/* Information structures (frame length 256 samples) */
typedef struct {
  float                  m_FrequencySpectrum[3][DEF_SamplingCount256/2];
  float                  m_RotationSpeed;
  float                  m_PeekCurrent;
  int8_t                 m_AiResult;
  float                  m_AiMovingAverage;
} ST_Sampling256;

Information structures (frame length 512 samples)
typedef struct {
  float                  m_FrequencySpectrum[3][DEF_SamplingCount512/2];
  float                  m_RotationSpeed;
  float                  m_PeekCurrent;
  int8_t                 m_AiResult;
  float                  m_AiMovingAverage;
} ST_Sampling512;

Information structures (frame length 1024 samples)
typedef struct {
  float                  m_FrequencySpectrum[3][DEF_SamplingCount1024/2];
  float                  m_RotationSpeed;
  float                  m_PeekCurrent;
  int8_t                 m_AiResult;
  float                  m_AiMovingAverage;
} ST_Sampling1024;;
```

```
/* Serial communication buffer structures and unions */
typedef struct {
   uint8_t              m_Header[DEF_HeaderSize]; /* Stores communication
header */
   uint8_t              m_Cmd; /* Communication control command, 01: A/D
conversion value, 02: FFT processed value */
   union {
      ST_SamplingRingData  m_SamplingData; /* Entity of 3 shunt current buffer
structure */
      ST_Sampling128    m_Sampling128; /* Entity of data structures */
      ST_Sampling256    m_Sampling256;
      ST_Sampling512    m_Sampling512;
      ST_Sampling1024   m_Sampling1024;
   } UN_Sampling;
} ST_SamplingMonitor;

/* Serial communication ring buffer structure */
typedef struct {
  uint16_t              m_Write; /* Ring buffer write position */
  uint16_t              m_Read; /* Ring buffer read position */
  uint8_t            m_RingData[DEF_PcRecvBuffSize]; /* Ring buffer */
} ST_PcRecvRingBuffer;

/* Moving average operation buffer structure */
typedef struct {
  uint16_t              m_MaxCount; /* Acquired data count */
  uint16_t              m_BufferFull;  /* Buffer full flag */
  uint16_t              m_Count; /* Moving average length */
  uint32_t              m_Sum;/* Moving average sum buffer */
  uint16_t              m_AiInference[DEF_MovingAverageMaxCount]; /* AI
inference value buffer */
} ST_MovingAverage;

/* Sampling conditions structure  */
typedef struct {
  uint16_t              m_SamplingFrequency; /* Sampling frequency */
  uint16_t              m_SamplingCount;  /* Sampling count */
  uint16_t              m_SamplingOverLap;   /* Overlap size */
  uint16_t              m_MovingAverageMaxCount;   /* Moving average maximum
count  */
  uint16_t              m_SamplingMode;   /* Smampling mode */
  uint32_t              m_CheckSum;    /* Checksum */
} ST_SamplingConditions;
```

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019.06.20	ー	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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合せください。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、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。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、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。