Renesas Synergy™
GUIX™ 基本アプリケーション

R30AN0320JJ0100
Rev.1.00
2018.06.05

要旨

本アプリケーションノートでは、Synergy Software Package (SSP)に含まれている GUIX™ の様々な機能を使った基本的なアプリケーションについて説明します。また、GUIX のアプリケーション開発に参考となる情報についても説明します。GUIX のアプリケーション開発を始めるにあたり、本アプリケーションノートを参照することで、GUIX を使ったアプリケーション開発に必要な情報の大枠を把握することができます。

動作確認デバイス

SK-S7G2, PK-S5D9

使用機能

以下の SSP モジュール、および、MCU 機能を使用しています。

種別	モジュール名／機能名
Framework	gx
	sf_touch_panel_i2c
	sf_message
HAL Driver	r_icu
	r_gpt
	r_sci_spi

目次

1. はじめに

1.1 概要

Renesas Synergy™ の Synergy Software Package (SSP)には GUI を使ったアプリケーション作成に有用な GUX™ が含まれています。本アプリケーションノートでは、この GUIX の様々な機能を使い、Renesas Synergy™ の MCU を搭載したボード上で動作する一つのシンプルなアプリケーションを紹介します。GUIX のアプリケーションとして組み込まれることの多い機能をシンプルな形で設計しているため、初めて GUIX のアプリケーションを開発する方がその概要を把握するのに適したものになっています。

尚、GUIX の設計ツールである GUIX Studio™ のインストールフォルダ下の examples フォルダには、各種のサンプルアプリケーションが用意されています。これらには、Microsoft Visual C++でビルド可能なプロジェクトが用意されており、Windows PC 上で動作を確認できます。GUIX のアプリケーション開発においては、これらも参考となります。

1.2 評価環境

本アプリケーションノートは、以下の環境で動作確認をしています。

SSP (Synergy Software Package)	v1.3.3
e² studio	v5.4.0.023
IAR EW for Synergy	V7.71.3
GUIX Studio™	V5.4.0.0
Board	SK-S7G2 PK-S5D9

1.3 本アプリケーションプロジェクトのビルド方法

以下の手順でビルドします。ただし、IAR EW for Synergy は、PK-S5D9 の QSPI へのダウンロードに対応していないため、プロジェクトは用意されていません。

1. アプリケーションプロジェクトを利用する開発環境にインポートします。インポート方法は、Renesas Synergy™ Platform Importing a Renesas Synergy Project (R11AN0023EU0118)を参照してください。
2. 開発環境のコンフィギュレータにおいて Generate Project Content を実行します。
3. guix_studio¥guix_basic_GNU.gxp、または、guix_studio¥guix_basic_IAR.gxp を GUIX Studio でオープンします。
4. GUIX Studio で Project → Generate All Output Files を選択します。これにより、src¥gui¥GNU、または、src¥gui¥IAR に GUIX に関するファイルが出力されます。
5. GUIX Studio を終了します。
6. src¥gui¥GNU、または、src¥gui¥IAR のどちらか一方のみがビルド対象となっているか確認します。両方がビルド対象となっている場合は、異なる開発環境のものをビルド対象外に設定します。
7. ビルドします。

2. アプリケーションの機能

2.1 機能一覧

本アプリケーションでは、以下の機能を利用しています。

- Radio Button
- Vertical List
- Drop List
- Slider
- Modal Dialog
- Button
- Progress Bar
- Circular Gauge
- 表示言語の切り替え
- ユーザー定義イベント
- ドラッグイベント
- スワイプ操作
- リソースの外部メモリ配置
- タッチパネル補正

2.2 機能概要

2.2.1 タッチパネル補正

アプリケーション起動直後、３点のタップ入力の結果をユーザーに促します。その結果からタッチパネル座標と GUI 画面座標の差分の補正を行うための係数を計算します。以降のタッチパネル入力については、得られた係数を使って座標の補正を行います。完了後は、メインメニューを表示します。

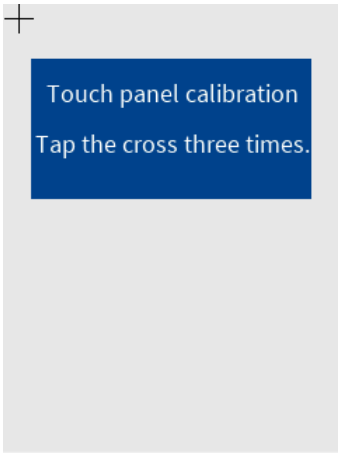

図 1 タッチパネル補正画面

2.2.2 メインメニュー

各動作項目へ遷移するためのボタンを表示し、ボタンが押された場合は、対応した項目の画面表示に切り替えます。画面上部の？マークをクリックすると、Modal Dialog で本アプリケーションノートの情報が表示されます。

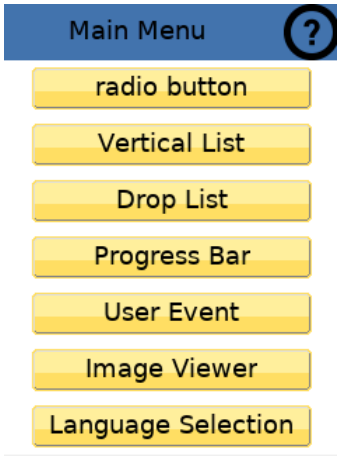

図 2　メインメニュー画面

2.2.3　Modal Dialog

本アプリケーションの情報をモーダルダイアログで表示します。このモーダルダイアログ以外へのタップのイベントは無効となり、同ダイアログ内の OK ボタンを押すとダイアログ表示を終了し、メインメニュー表示に戻ります。

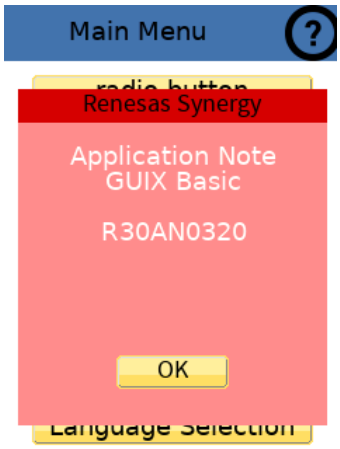

図 3　Modal Dialog 画面

2.2.4　Radio Button

３つのグループのラジオボタンを表示し、それぞれの操作によりボード上の３つの LED の点滅動作の ON/OFF を制御します。画面上部の家型のマークをクリックすることで、メインメニュー表示に戻ります。これは、以降の画面についても同様です。

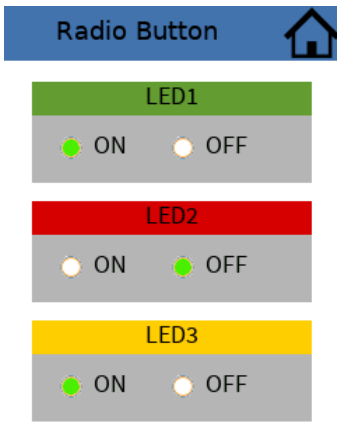

図 4　ラジオボタン画面

2.2.5 Vertical List と Slider

３つの LED に対する８つの制御パターンを選択する Vertical List を表示します。そこで選択された制御パターンで LED の点滅動作を切り替えます。また、LED の点滅動作の時間間隔を制御する水平方向の Slider を表示します。この Slider で設定された値で、点滅間隔を設定します。

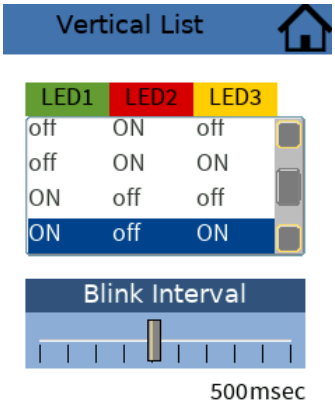

図 5 Vertical List と Slider

2.2.6 Drop List

LED の点灯パターンを選択するドロップリストを表示します。

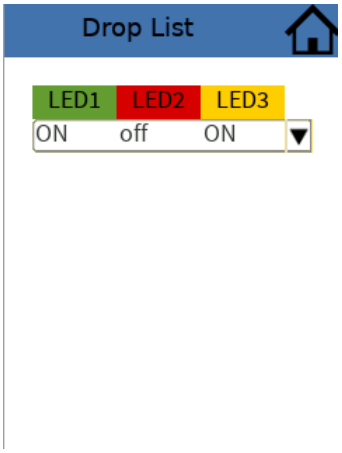

図 6 Drop List 画面

2.2.7 Progress Bar と Circular Gauge

プログレスバーと半円形のインジケーターを表示します。各ウィジェット上を水平方向にドラッグすることで、それぞれの値を 0%から 100%まで変更することができます。プログレスバーの色は、値によって赤、黄、緑に変化します。

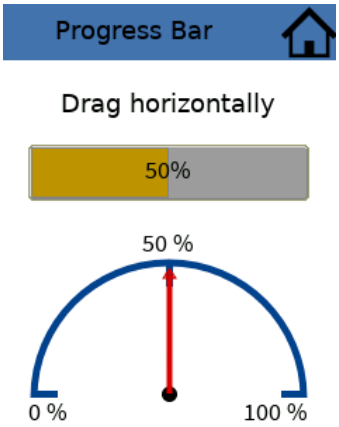

図 7　Progress Bar と Circular Gauge 画面

2.2.8　　カウンタ

00 から 99 までの数字を表示するカウンタを表示します。画面上の 2 つの▲型ボタンで値を変更することができます。また、ボード上の S4, S5 ボタンでも値を変更することができます。どちらのボタンも長押しすることで、オートリピート入力動作を行います。

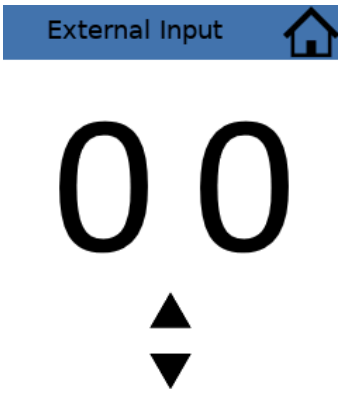

図 8　カウンタ画面

2.2.9　　画像ビューア

アプリケーションに組み込まれている画像を表示します。画像は画面を水平方向にスワイプすることで、切り替えることができます。画像を画面の横方向の半分以上ドラッグすると次の画像に切り替わりますが、それ以外の場合は元の画像表示に戻ります。スワイプ操作に対して、画像は循環して表示されます。画像データは QSPI メモリに保存されています。

図 9　画像ビューア画面

2.2.10　言語選択

画面の表示言語を切り替えます。英語と日本語が選択可能です。日本語を選択した場合、日本語表示が設定されている文字列は日本語に切り替わります。

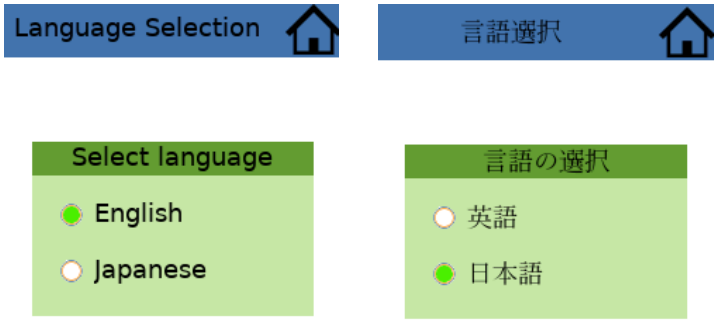

図 10　言語選択画面

3. 設計概要

3.1 Blinky Thread

LED の点滅動作を制御します。SSP でテンプレートとして用意されている Blinky with ThreadX プロジェクトで生成される blinky_thread_entry.c をベースに、機能を変更して使用しています。以下に一部の関数について説明します。

blinky_thread_entry	Blinky Thread の処理を行います。Messaging Framework でメッセージ受信を行い、受信したメッセージが Blinky Event Class であれば、その内容に従って点滅させる LED と点滅間隔を変更します。Blinky Event Class のメッセージが指定された時間内に受信できない場合は、点滅動作が有効な LED の点灯状態をトグルさせます。
blinky_update	Messaging Framework を使い、点滅させる LED と点滅間隔の情報を Blinky Event Class のメッセージとして送信します。GUI の操作などにより、LED の点滅動作を変更する場合に使用します。
blinky_create_pattern_row	LED の点滅動作パターンについての prompt ウィジェットを生成します。Drop List や Vertical List で利用します。

3.2 GUI Thread

GUI Thread は、GUIX の初期化処理をした後、Messaging Framework でイベント待ちのループ処理に入ります。Touch Event Class のメッセージを受信した場合は、それを GUIX のイベントに変換した後、gx_system_event_send 関数で GUIX にイベントとして通知します。

3.2.1 GUIX 初期化処理

以下の手順で GUIX の初期化処理を行います。

1. gx_system_initialize 関数で GUIX の初期化を行います。
2. SF_EL_GX の open 関数をコールし、SF_EL_GX の初期化を行います。
3. gx_studio_display_configure 関数をコールします。これにより、SF_EL_GX の setup 関数がコールされ、ディスプレイドライバの設定を行います。
4. SF_EL_GX の canvasInit 関数で GUIX キャンバスのメモリアドレスを設定します。ディスプレイドライバは、このアドレスにイメージを描画し、また、このアドレスのイメージを表示します。
5. gx_studio_named_widget_create 関数で GUIX ウィジェットを生成します。
6. gx_widget_show 関数で最初に表示するウィジェットを設定します。
7. gx_system_start 関数で GUIX を起動します。

また、必要に応じて以下で述べる GUIX 用メモリアロケート関数とメモリ解放関数の登録を行います。

3.2.2 GUIX 用メモリアロケート関数とメモリ解放関数

gx_system_memory_allocator_set 関数で、GUIX 内で利用されるメモリアロケート関数、および、メモリ解放関数を登録しています。GUIX では、処理に必要なメモリ領域を動的に確保し、その処理が終了した後はそのメモリ領域を開放することが必要となる処理があります。対象となる処理については、GUIX™ User Guide の gx_system_memory_allocator_set 関数の説明を参照してください。

本アプリケーションノートでは、Circular Gauge での回転処理に必要となるため、メモリアロケート関数、および、メモリ解放関数の登録を行います。ここでは、メモリアロケート用の領域 mem_guix_pool を確保し、その領域を tx_byte_pool_create 関数で ThreadX の byte pool 領域として登録します。メモリアロケートを行う guix_malloc 関数では、tx_byte_allocate 関数をコールし、指定されたサイズのメモリを同 byte pool から確保します。メモリ解放を行う guix_free 関数では、指定された領域を tx_byte_release 関数で解放します。

3.3 タッチパネル補正

本アプリケーションで使用するボードにおいては、タッチパネルから得られる座標と GUI 上の座標の差異があり、小さいウィジェットへの操作に支障があります。このため、起動直後にこの差異を軽減させるためのキャリブレーションを行います。画面上の 3 点に＋マークを 1 つずつ順次表示し、GUI 上の座標(X_{rn}, Y_{rn})とそれに対して得られたタッチパネル座標(X_{an}, Y_{an})を得ます（n は座標番号 1～3）。得られた 3 つの組の結果（(X_{r1}, Y_{r1}), (X_{a1}, Y_{r1})の組、(X_{r2}, Y_{r2}), (X_{a2}, Y_{r2})の組、(X_{r3}, Y_{r3}), (X_{a3}, Y_{r3})の組）から、タッチパネル座標を GUI 座標に以下の式で補正するための係数 Ax, Bx, Cx, Ay, By, Cy を計算します。

$$X = AxXa + BxYa + Cx$$

$$Y = AyXa + ByYa + Cy$$

補正に関する説明は省略します。

得られた上記の係数は、gui_thread_entry.c 内の ssp_touch_to_guix 関数において、Touch Panel Framework から得られたイベントを GUIX のイベントに渡す前の処理で使用されます。キャリブレーション前の初期状態では、Ax=1、 By=1、これら以外は 0 とすることで、補正なしとしています。

関連する項目は以下になります。

GUIX Studio	window_calibration ウィジェット
ソースコード	src/calibration.c src/calibration.h src/gui_thread_entry.c src/window_calibration.c

3.3.1 calibration.c

以下の 2 関数が実装されています。

calibration_update_factor	得られた 3 つの組の結果（(Xr1, Yr1), (Xa1, Yr1)の組、(Xr2, Yr2), (Xa2, Yr2)の組、(Xr3, Yr3), (Xa3, Yr3)の組）からタッチパネル座標を GUI 座標に以下の式で補正するための係数 Ax, Bx, Cx, Ay, By, Cy を計算する。 window_calibration ウィジェットにおいて、3 つ目の＋マークに対するタップが行われた後、この関数をコールする。
calibration_get_coordinate	入力されたタッチパネル座標を、補正係数 Ax, Bx, Cx, Ay, By, Cy で GUI 座標に変換する。

3.4 共通機能

3.4.1 window_base テンプレート

window 型のウィジェットであり、画面上部のタイトル表示用 prompt ウィジェット、ホームアイコン表示用の pixelmap_button ウィジェット、？マークアイコン表示用の pixelmap_button ウィジェット、背景画面用の window ウィジェットを持ちます。window_base テンプレートとして設定され、以降で説明する各 window ウィジェットで共通して使用されます。

関連する項目は以下になります。

GUIX Studio	window_base ウィジェット
ソースコード	src/window_base.c

3.4.2 window_base.c

共通して使用される以下の機能が実装されています。

window_base_handler	ホームアイコンがクリックされた場合、メインメニューの表示に切り替える。？マークアイコンがクリックされた場合、アプリケーションの情報を表示するモーダルウィンドウを表示する。
toggle_window	第一引数で指定された window を表示し、第二引数で指定された window を detach する。
update_window_top	第一引数で指定されたウィジェット内のタイトル表示を、第二引数で指定されたリソースの表示に切り替える。また、同ウィジェットのアイコン表示を第三引数で指定されたリソースの表示に切り替える。

3.4.3 モーダルウィンドウ表示

？マークアイコンがクリックされた場合、window_aboutme ウィジェットをモーダルウィンドウとして表示します。この表示は gx_window_execute 関数で行います。これにより、同ウィジェット外に対するクリックなどのイベントは無視されます。同ウィジェットに対するイベントは window_aboutme_handler で処理し、OK ボタンがクリックされた場合は、gx_window_close 関数をコールして、モーダルウィンドウ表示を終了します。

3.5 メインメニュー

関連する項目は以下になります。

GUIX Studio	window_main_menu ウィジェット
ソースコード	src/window_main_menu.c

3.5.1 window_main_menu ウィジェット

window_base 型のウィジェットであり、子ウィジェットとして、各機能へ遷移するための text_button ウィジェットを持ちます。Event Function として、window_main_menu_handler を登録します（図 11）。

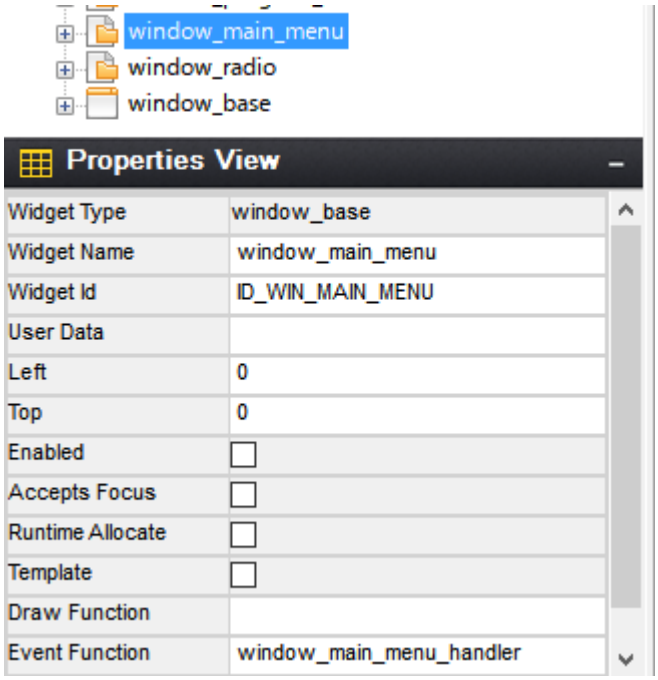

図 11 Event Function の登録

3.5.2 window_main_menu.c

window_main_menu_handler の処理が実装されます。同ハンドラでは、以下のように発生するイベントに対応した処理を行います。

表 1 window_main_menu_handler の処理

GX_EVENT_SHOW
gx_window_event_process をコール。
画面上部のタイトル表示をメインメニューに変更。
HOME アイコンを非表示にし、？マークアイコンを表示。
ID_TXB_MENU_???
ToggleScreen をコールし、対応する機能画面に遷移。

3.6 ラジオボタン

関連する項目は以下になります。

GUIX Studio	window_radio ウィジェット
ソースコード	src/window_radio_menu.c

3.6.1 window_radio ウィジェット

window_base 型のウィジェットであり、子ウィジェットとして、各 LED 制御のラジオボタンをグループ化する３つの window ウィジェットを持ちます。３つの window ウィジェットは、それぞれ、タイトル表示用の prompt ウィジェット、LED 制御 ON/OFF の radio_button ウィジェットを２つ持ちます。

Event Function として、window_radio_handler を登録します。

3.6.2 window_radio.c

window_radio_handler の処理が実装されます。

表 2 window_radio_handler の処理

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
現在の LED 制御設定に合わせてラジオボタンの状態を設定する。
GX_SIGNAL(ID_RDB_LEDn_ON, GX_EVENT_RADIO_SELECT)
LEDn の点滅動作を ON にする。
GX_SIGNAL(ID_RDB_LEDn_OFF,GX_EVENT_RADIO_SELECT)
LEDn の点滅動作を OFF にする。

3.7 垂直リスト

関連する項目は以下になります。

GUIX Studio	window_vlist ウィジェット
ソースコード	src/window_vertical_list.c

3.7.1 window_vlist ウィジェット

window_base 型のウィジェットであり、主な子ウィジェットとして、LED 制御パターンを選択する vertical_list_led_pattern ウィジェットと、LED の点滅間隔を設定する slider_led_interval ウィジェット、slider_led_interval で設定された値を表示する prompt_inverval_value ウィジェットを持ちます。

Event Function として、window_vertical_list_handler を登録します。

3.7.2 vertical_list_led_pattern ウィジェット

vertical_list 型のウィジェットであり、子ウィジェットとして、リストのスクロール操作を行う vscroll 型ウィジェットの vertical_scroll を持ちます。リスト内のアイテムである LED 制御パターンは、ランタイムで本ウィジェットに登録します。

3.7.3 slider_led_interval ウィジェット

slider 型のウィジェットであり、LED の点滅間隔を 100msec から 1000msec の間で設定します。

3.7.4 window_vertical_list.c

window_vlist ウィジェットの Event Function である window_vertical_list_handler 関数、および、vertical_list_led_pattern に登録するアイテムを初期登録する vertical_list_create 関数が実装されています。

表 3 window_vertical_list_handler

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
現在の LED 制御設定に合わせ、ID_VTL_LED_PATTERN リスト内の該当インデックスを選択状態にする。
現在の LED 点滅間隔の設定に合わせ、slider_led_interval ウィジェットのスライダーの位置を設定する。
現在の LED 点滅間隔を ID_PRP_INTERVAL_VALUE ウィジェットに表示する。
GX_SIGNAL (ID_VTL_LED_PATTERN, GX_EVENT_LIST_SELECT)
ID_VTL_LED_PATTERN で選択されたアイテムを取得し、その設定に LED 制御を変更する。
GX_SIGNAL (ID_SLI_LED_INTERVAL, GX_EVENT_SLIDER_VALUE)
ID_SLI_LED_INTERVAL ウィジェットの値を取得し、その値で LED の点滅間隔を変更する。
上記の LED 点滅間隔を ID_PRP_INTERVAL_VALUE ウィジェットに表示する。

vertical_list_create 関数は、vertical_list_led ウィジェットに登録する項目を追加する処理です。gui_thread_entry.c の gui_thread_entry 関数において、window_vertical_list ウィジェット生成後に使用します。

3.8 ドロップリスト

関連する項目は以下になります。

GUIX Studio	window_drop_list ウィジェット
ソースコード	src/window_drop_list.c

3.8.1 window_drop_list ウィジェット

window_base 型のウィジェットであり、主な子ウィジェットとして、LED 制御パターンを選択する drop_list_led ウィジェットを持ちます。drop_list_led ウィジェットには、ドロップリストを開閉する子ウィジェットを追加します。この子ウィジェットでドロップリストを開閉させるためには、このウィジェットの Widget ID を ID_DROP_LIST_BUTTON で定義する必要があります。GUIX™ User Guide の gx_drop_list_create 関数の記述を参照してください。

Event Function として、window_drop_list_handler を登録します。

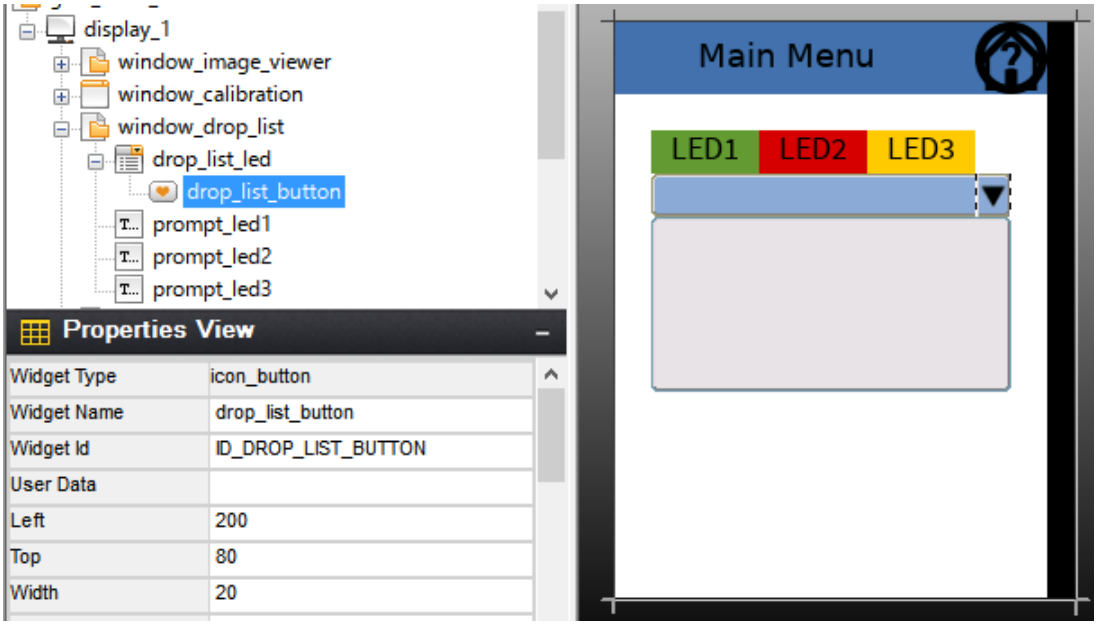

図 12 ドロップリストボタンのウィジェット ID

3.8.2 window_drop_list.c

window_drop_list ウィジェットの Event Function である window_drop_list_handler 関数、および、drop_list_led に登録するアイテムを初期登録する drop_list_create 関数が実装されています。

表 4 window_drop_list_hander

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
現在の LED 制御設定に合わせて、ID_DRL_LED_PATTERN 内の該当インデックスを選択状態にする。
GX_SIGNAL(ID_DRL_LED_PATTERN, GX_EVENT_LIST_SELECT)
ID_DRL_LED_PATTERN で選択されたアイテムを取得し、得られた設定に LED 制御を変更する。

drop_list_create 関数は、drop_list_led ウィジェットに登録する項目を追加する処理です。gui_thread_entry.c の gui_thread_entry 関数において、window_drop_list ウィジェット生成後に使用します。

3.9 プログレスバー

関連する項目は以下になります。

GUIX Studio	window_progress_bar ウィジェット
ソースコード	src/window_progress_bar.c

3.9.1 window_progress_bar ウィジェット

window_base 型のウィジェットであり、主な子ウィジェットとして、progress_bar ウィジェット、および、circular gauge ウィジェットを持ち、それぞれ 0～100%の値を示します。両ウィジェットをタップし、そのまま水平方向にドラッグすると、ドラッグされた方向に従って、プログレスバーと半円型のメーターの値が変化します。

3.9.2 window_progress_bar.c

window_progress_bar ウィジェットの Event Function である window_progress_bar_handler 関数、および、progress_bar ウィジェットにおいて、値によってバーの色を変更するための Draw Function である draw_progress_bar 関数が実装されています。

表 5 window_progress_bar_hander

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
現在のパーセンテージ値に合わせて、ID_PRG_PERCENTAGE 内の値を設定する。
現在のパーセンテージ値に合わせて、ID_GAG_PERCENTAGE 内の値を設定する。
GX_EVENT_PEN_DOWN
タップされた X 座標と現在のパーセンテージ値を保存する。
GX_EVENT_PEN_UP
タップされた X 座標を無効値に設定する。
GX_EVENT_PEN_DRAG
PEN_DOWN 時に保存した X 座標と現在の DRAG における X 座標の差を求め、その値を元に増減値を得る。
その増減値で現在のパーセンテージ値を変更し、gx_progress_bar_value_set 関数で ID_PRG_PERCENTAGE の値を更新し、また、gx_circular_gauge_angle_set 関数で ID_GAG_PERCENTAGE のニードルの回転表示を更新する。

3.9.3 Draw Function のオーバーライド

draw_progress_bar 関数は、progress_bar_percentage ウィジェットの Draw Function プロパティにおいて設定します（図 13）。これにより、同ウィジェットの Draw Function は progress_bar 型ウィジェットのデフォルト関数である gx_progress_bar_draw 関数から draw_progress_bar 関数にオーバーライドされます。このオーバーライドされた draw_progress_bar 関数では、数値に応じてウィジェットの fill color を赤、黄、緑に変更するよう gx_widget_fill_color_set 関数をします。その後、元の gx_progress_bar_draw 関数をコールします。

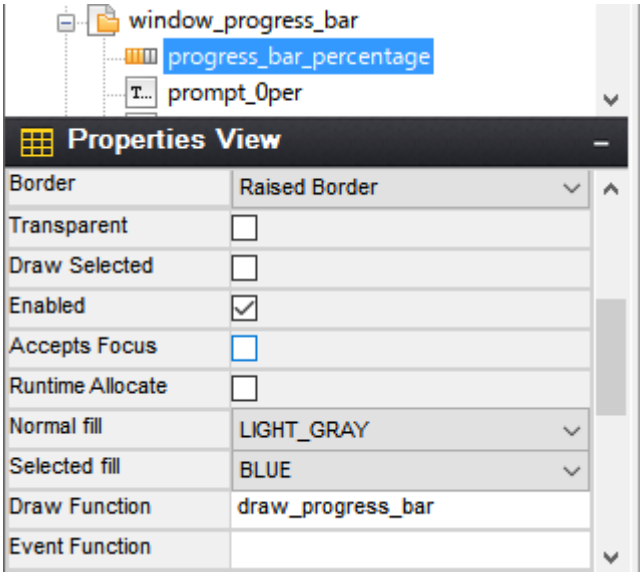

図 13　Draw Function のオーバーライド

3.10 ユーザー定義イベント

関連する項目は以下になります。

GUIX Studio	window_counter ウィジェット
ソースコード	src/button1_thread_entry.c src/button2_thread_entry.c src/window_counter.c

3.10.1 ユーザー定義イベントの実装

本アプリケーションでは、ボード上の２つのプッシュボタンへの操作に対して、ユーザー定義のイベントを GUIX に送信しています。ユーザー定義のイベントに対するイベント ID は、gx_api.h で定義されている GX_FIRST_APP_EVENT (0x40000000) 以降の値に割り当てます。本アプリケーションでは、２つのボタンに対して、それぞれ、USER_EVENT_BUTTON_1 (= GX_FIRST_APP_EVENT + 0)、USER_EVENT_BUTTON_2 (= GX_FIRST_APP_EVENT + 1)を設定しています。

各ボタンに対して Button1 Thread、Button2 Thread を作成し、ボタンの割り込みに対しては External IRQ Driver on r_icu を、ボタンのオートリピート入力の処理用に Timer Driver on r_gpt を、それぞれについて使用しています。最初のボタン押下を検知すると 1000msec のタイマを起動し、このタイマが満了すると USER_EVENT_BUTTON_1/2 のイベントを gx_system_event_send 関数で GUIX に送信します。最初のタイマ満了以降は、タイマの設定を 100msec に設定します。これにより、ボタンを押下したままにすると、最初のボタン押下の 1000msec 後は、100msec 毎にボタン入力イベントを繰り返して GUIX に送信します。

3.10.2 window_counter ウィジェット

window_base 型のウィジェットであり、主な子ウィジェットとして、二桁の数値の十の位の数字を表示する window_tens_place ウィジェットと、一の位の数字を表示する window_ones_place ウィジェット、数値をカウントアップするボタンの pixelmap_button_up ウィジェット、カウントダウンするボタンの pixelmap_button_down ウィジェットを持ちます。

3.10.3 window_counter.c

window_counter ウィジェットの Event Function である window_counter_handler 関数が実装されています。

表 6 window_counter_handler

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
GX_SIGNAL(ID_PXB_UP, GX_EVENT_CLICKED)
USER_EVENT_BUTTON_2
カウンタ値を１増やし、数字表示を更新する。
GX_SIGNAL(ID_PXB_DOWN, GX_EVENT_CLICKED)
USER_EVENT_BUTTON_1
カウンタ値を１減らし、数字表示を更新する。

表 6に示すように、ID_PXB_UP のクリックイベントと USER_EVENT_BUTTON_2 イベントの両方においてカウンタ値を 1 加算します。同様に、ID_PXB_DOWN のクリックイベントと USER_EVENT_BUTTON_1 イベントの両方においてカウンタ値を 1 減算します。

数値表示の更新は update_counter 関数で、window_tens_place ウィジェット、および、window_ones_place ウィジェットの各 window ウィジェットの wallpaper を gx_window_wallpaper_set 関数で対応する数字の pixelmap に変更することで行います。

3.10.4 オートリピート入力

GUIX のボタンウィジェットについては、そのプロパティにおいて Auto Repeat を ON に設定します(図 14)。これにより、同ボタンウィジェットをタップ後に押下したままにすると、同ウィジェットへの GX_EVENT_CLICKED イベントが一定間隔で発生します。3.10.1 で述べたように、USER_EVENT_BUTTON_1/2 についてもタイマを使ってオートリピート入力となるようにイベントを発生させています。

GUIX ボタンウィジェットと外部ボタンによるオートリピート入力動作は、表 7のようにその動作が異なっています。GUIX ボタンウィジェットについては、SSP で提供されている GUIX ライブラリ libgx.a のボタンリピート設定に従って動作しています。これは synergy/ssp/inc/framework/el/gx_api.h にある図 15に示す設定に対応します。GUIX のシステムタイマの初期値は 20msec 周期であるので(4.3 GUIXシステムタイマを参照)、オートリピート開始判定時間は `20msec * GX_REPEAT_BUTTON_INITIAL_TICS = 200msec`、オートリピート時間間隔は `20msec * GX_REPEAT_BUTTON_REPEAT_TICS = 60msec` となります。この GUIX ボタンウィジェットのオートリピート入力時間をユーザー定義イベントと同じ設定にする場合は、4.1 GUIXソースコード、および、4.2 GUIXのカスタマイズに示すように、プロジェクトに GUIX Source を追加し、`GX_REPEAT_BUTTON_INITIAL_TICS = 50`、`GX_REPEAT_BUTTON_REPEAT_TICS= 5` の定義を C プリプロセッサに設定してビルドします。

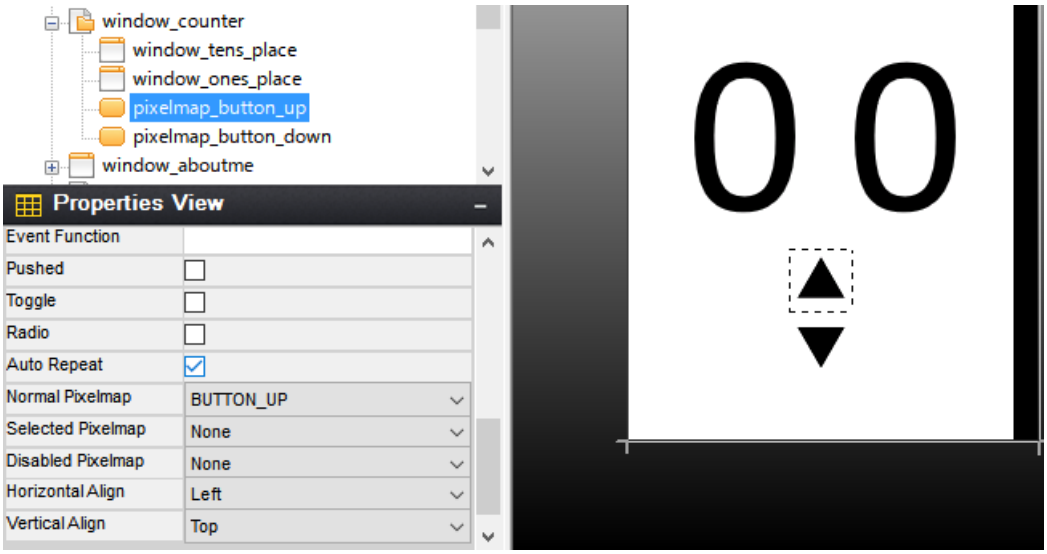

図 14 Button ウィジェットの Auto Repeat 設定

表 7 オートリピート入力時間（GUI ボタンと外部ボタン）

	オートリピート開始判定時間	オートリピート時間間隔
GUIX ボタウィジェット	200 msec	60 msec
外部ボタン	1000 msec	100 msec

```
#ifndef GX_REPEAT_BUTTON_INITIAL_TICS
#define GX_REPEAT_BUTTON_INITIAL_TICS      10
#endif

#ifndef GX_REPEAT_BUTTON_REPEAT_TICS
#define GX_REPEAT_BUTTON_REPEAT_TICS       3
#endif
```

図 15 GUIX のボタンのリピート動作設定 (gx_api.h)

3.11 画像ビューア

関連する項目は以下になります。

GUIX Studio	window_image_viewer ウィジェット
ソースコード	src/window_image_viewer.c

3.11.1 window_image_viewer ウィジェット

window_base 型のウィジェットであり、画像を表示するウィンドウと、左右のスワイプ操作時にそのウィンドウの左右に表示する２つのウィンドウの計３つの window ウィジェットを持ちます。

3.11.2 window_image_viewer.c

window_image_viewer の Event Function である window_image_viewer_handler 関数が実装されています。

表 8 window_image_viewer_handler

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
最初の同イベント発生時に、最初に表示する window ウィジェットのみ GX_STATUS_VISIBLE を設定し、残り２つの window ウィジェットは GX_STATUS_VISIBLE を外す。
３つの window ウィジェットに対して、表示する画像とその左右に表示する画像を設定する。
gx_animation_drag_enable 関数をコールし、ウィジェットの Event Function をスクリーンドラッグアニメーションの Event Function に置き換える。
GX_EVENT_HIDE
gx_animation_drag_disable 関数をコールし、gx_animation_drag_enable 関数コール時に置き換えたウィジェットの Event Function に戻す。
GX_EVENT_ANIMATION_COMPLETE
アニメーション完了時に表示状態になった画像のインデックスを調べる。
３つの window ウィジェットに対して、表示する画像とその左右に表示する画像を設定する。

また、スワイプ操作時の画像の切り替えアニメーションに関する初期化を行う image_viewer_initialize 関数が実装されています。同関数では GX_ANIMATION_INFO 型の変数の初期化を行い、スクリーンのドラッグ操作に対して、３つの window ウィジェットのリストを循環的に表示するアニメーションの設定を行います。

各 window ウィジェットに表示する画像の更新処理は、update_images 関数において gx_window_wallpaper_set 関数で行います。

同ファイル中のその他の関数については、本アプリケーションノートの本題から外れるため、説明を省略します。

3.11.3 画像データの QSPI メモリへの配置

本アプリケーションノートでは、GUIX Studio のプロジェクトに８つの JPEG ファイルを登録し、それらを GUIX Pixelmap フォーマットに変換したデータとして利用しています。これらの画像データは、ボードに搭載されている QSPI シリアルフラッシュメモリ上に保存します。

特定のデータを QSPI 領域に割り当てるには、それらを QSPI のセクションに割り当てるように設定する必要があります。本アプリケーションノートでは、GUIX Studio において、以下のようにして画像データのリソースデータを１つのファイルに出力しています。

1. GUIX Studio において、各画像データのデータ保存先ファイル名を Specify Output File に設定します。（図 35を参照。本アプリケーションノートでは、IMAGE1～IMAGE5 において image_data.c に設定。）
2. GUIX Studio において、Project -> Generate All Output Files を選択すると、画像データのリソースのみが上で指定したファイルに保存されます。

この image_data.c 内のデータを QSPI のメモリ領域に割り当てる方法として以下の２つがあります。

(1) 画像データを.qspi_flash セクションに割り当て

SSP に用意されているサンプルプロジェクト Blinky with ThreadX に含まれるリンカスクリプトを流用する場合、image_data.c に含まれる.rodata は、そのままでは.text セクションに割り当てられるため、Code Flash 領域に配置されます。しかし、同リンカスクリプトでは、.qspi_flash セクションに割り当てることで QSPI の領域に配置されるようになっています。よって、image_data.c の各データを.qspi_flash セクションに割り当てるように指定することで、画像データを QSPI 領域に割り当てることができます。

以下のように BSP_PLACE_IN_SECTION マクロを使うことで、データを.qspi_flash に割り当てるように指定できます。これを image_data.c に含まれる全データについて行います。この方法は GNU, IAR の両方で共通です。

```
static GX_CONST USHORT DISPLAY_1_THEME_1_IMAGE1_pixelmap_data[66054]
BSP_PLACE_IN_SECTION(".qspi_flash") =
```

(2) 画像データの.rodata を QSPI 領域に割り当て

image_data.c は画像データを変更する度に GUIX Studio によって更新されます。このため、image_data..c の編集は行わず、リンカスクリプトで image_data.c 内の.rodata を QSPI 領域に配置させる方法もあります。

GNU の場合、リンカスクリプト（S7G2.ld など）を図 16のようにすることで、image_data.o の.rodata を QSPI のメモリに割り当てることができます。この方法では、image_data.o 以外に含まれる.rodata を.text セクションに割り当て、残りの.rodata を.qspi_flash セクションに割り当てます。

```
.text :
{
   （略）
        *(EXCLUDE_FILE (*image_data.o) .rodata*)    /* Replaced *(.rodata*) */
   （略）
}

 （略）

.qspi_flash :
{
   （略）
        *(.rodata*)    /* Added */
   （略）
}
```

図 16 QSPI 領域への配置（GNU）

IAR の場合、リンカスクリプト（S7G2.icf など）を図 17のようにすることで、image_data.o の.rodata を QSPI のメモリに割り当てることができます。この方法では以下のようにしています。

- image_data.o 内の.rodata を指す block QSPI_GUIX を定義
- 上記以外の.rodata 指す block RODATA を定義
- block RODATA を FLASH_region に追加
- block QSPI_GUIX を QSPI_region に追加

```
（略）
define block QSPI_GUIX                    { section .rodata object image_data.o };
define block RODATA                       { section .rodata };
```

```
（略）
place in FLASH_region   { block LOCK_LOOKUP,
                        ro,
                        block RODATA,
                        block QSPI_NON_RETENTIVE_INIT_BLOCK,
                        block RAM_INIT_CODE,
                        block USB_DEV_DESC_BLK };
（略）
place in QSPI_region    { block QSPI_GUIX,
                        block QSPI_NON_RETENTIVE_BLOCK,
                        section .qspi_flash  };
```

図 17　QSPI 領域への配置（IAR）

3.12 言語切り替え

関連する項目は以下になります。

GUIX Studio	ウィジェット	window_lang_selection
	Theme	theme_2
	Fonts	MultiLang
ソースコード	src/window_lang_selection.c	

3.12.1 言語の追加

Strings リソースに日本語表示用の文字列を追加するため、Language を 1 つ追加し、Language ID は Japanese [ja]を選択します。言語の追加操作については、4.5 言語の追加を参照してください。

3.12.2 Theme の追加

本アプリケーションでは、英語表示と日本語表示の切り替えを font table を変更することで行います。このため、Theme を 1 つ追加して、Theme_1 を英語表示時の font table、Theme_2 を日本語表示時の font table とし、両言語用の font table が生成されるようにします。Theme の追加については、4.4 Themeの追加を参照してください。

3.12.3 フォントの追加

上述の Theme の追加により Theme の数が 2 つ（Theme_1、Theme_2）となりますが、これらに、多言語表示用の Fonts リソースを追加します。Theme_1 については GUIX Studio の fonts フォルダにある verasans¥Vera.ttf を、Theme_2 については同フォルダにある kochi-mincho-subst.ttf を”MultiLang”という名前で Fonts リソースに追加します。この時、”Include character set defined by String Table”にチェックを入れ、生成するフォントデータを Strings リソースで使用される文字のみにすることで、フォントのリソースデータのサイズを最小限となるようにしています。フォントの追加については、4.6 フォントを参照してください。

これにより、guix_basic_GNU_resources.c、または、guix_basic_IAR_resources.c において、英語表示用の font table である display_1_theme_1_font_table、日本語表示用の font table である display_1_theme_2_font_table が生成されます。

3.12.4 日本語文字列の追加

Strings リソースにおいて、日本語として表示するものについて、日本語の文字列を設定します。Strings リソースへの日本語文字列の設定の操作については、4.7 追加言語の文字列設定を参照してください。

3.12.5 window_lang_selection ウィジェット

window_lang_select ウィジェットの Event Function である window_lang_select_handler 関数が実装されています。

表 9 window_lang_select_hander

GX_EVENT_SHOW
共通処理（表 1の GX_EVENT_SHOW）
GX_SIGNAL(ID_RDB_ENGLISH, GX_EVENT_RADIO_SELECT)
gx_display_font_table_set 関数で英語表示用の font table を設定。 gx_system_active_language_set 関数で言語を英語に設定。
GX_SIGNAL(ID_RDB_JAPANESE, GX_EVENT_RADIO_SELECT)
gx_display_font_table_set 関数で日本語表示用の font table を設定。 gx_system_active_language_set 関数で言語を日本語に設定。

4. 参考情報

以下では、GUIX に関する各種手順、ノウハウ、および、関連する情報について示します。

4.1 GUIX ソースコード

プロジェクトに GUIX on gx を追加した場合、通常は gx ライブラリ libgx.a が使用されます。デバッグ時に GUIX の内部の動作をモニタする場合や、GUIX の設定を変更する場合は、GUIX source をプロジェクトに追加し、GUIX をソースコードからビルドする必要があります。図 18のように、GUIX on gx に下にある”Add GUIX Source”をクリックし、”New”→”GUIX Source”を選択して GUIX Source を追加します。追加すると図 19のように”GUIX Source”が赤字で表示されます。これは、プロジェクトで libgx.a と GUIX Source が 2 つ存在するため、リンク時に二重定義エラーが発生する可能性があることを警告表示しています。プロジェクトをビルドした時に同エラーが発生する場合は、図 20、および、図 21のようにプロジェクトのリンカの設定から libgx.a を外してください。リンクに問題がないことが確認でき、図 19の警告表示を抑制したい場合は、図 22に示すように GUIX Source のプロパティ”Show linkage waning”を Disabled に設定します。

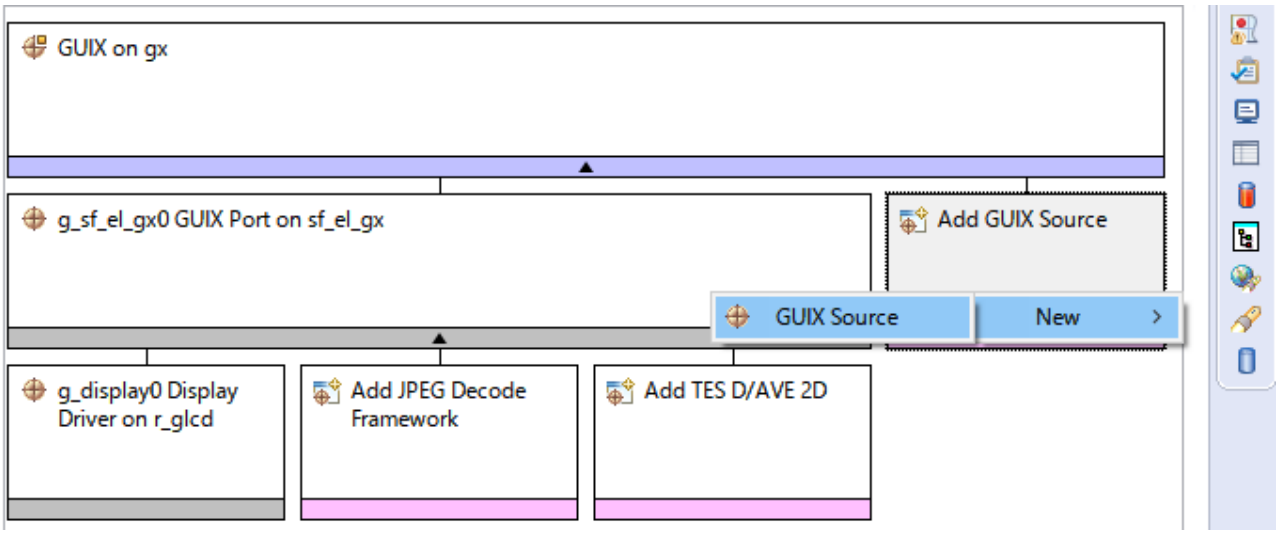

図 18 GUIX ソースコードの追加

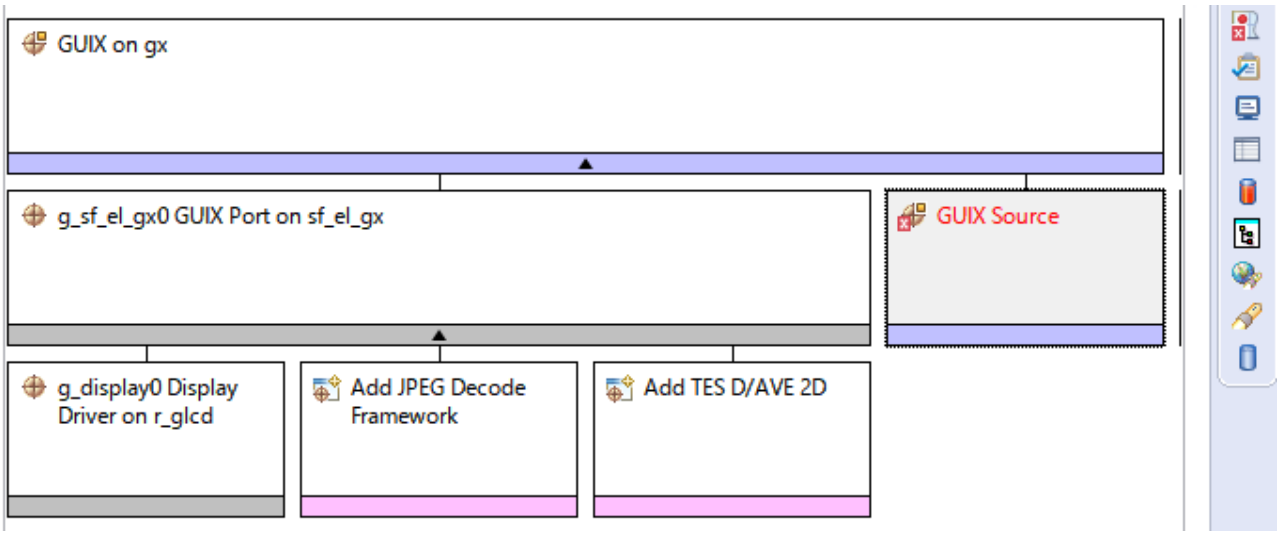

図 19 GUIX ソースコード追加の警告表示

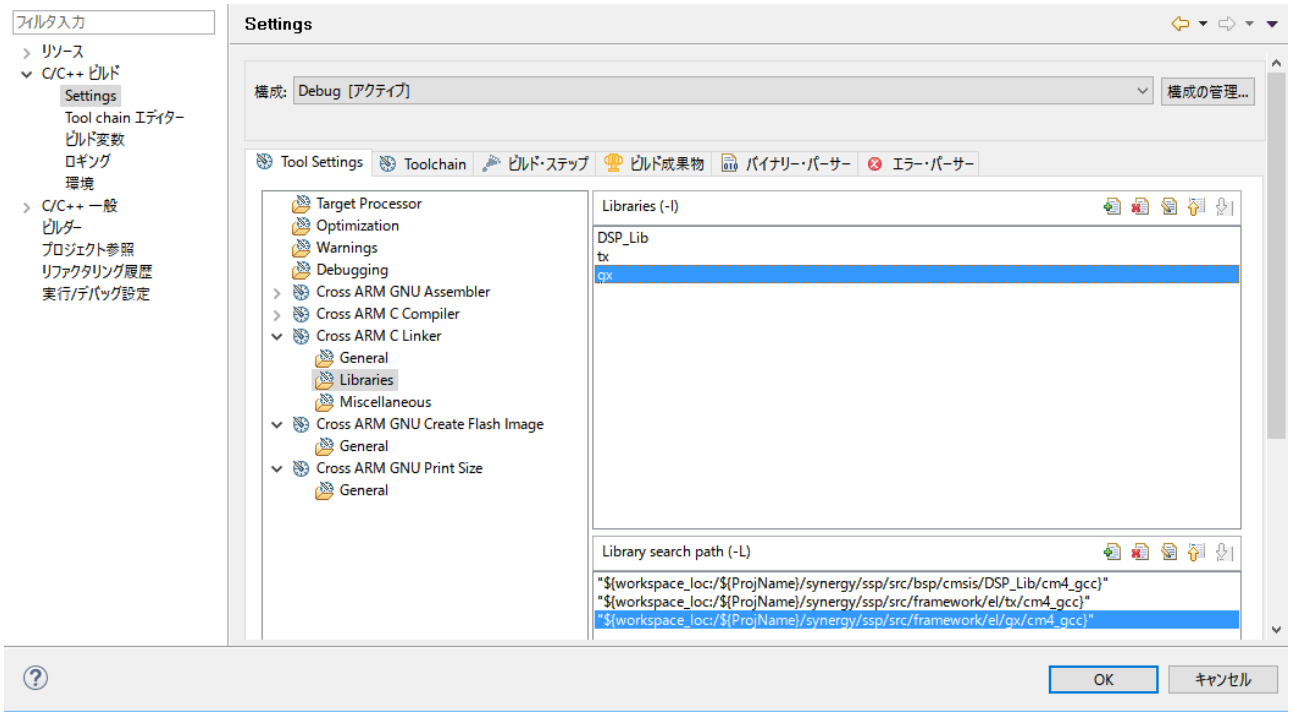

図 20 GUIX ライブラリの除外（GNU 環境）

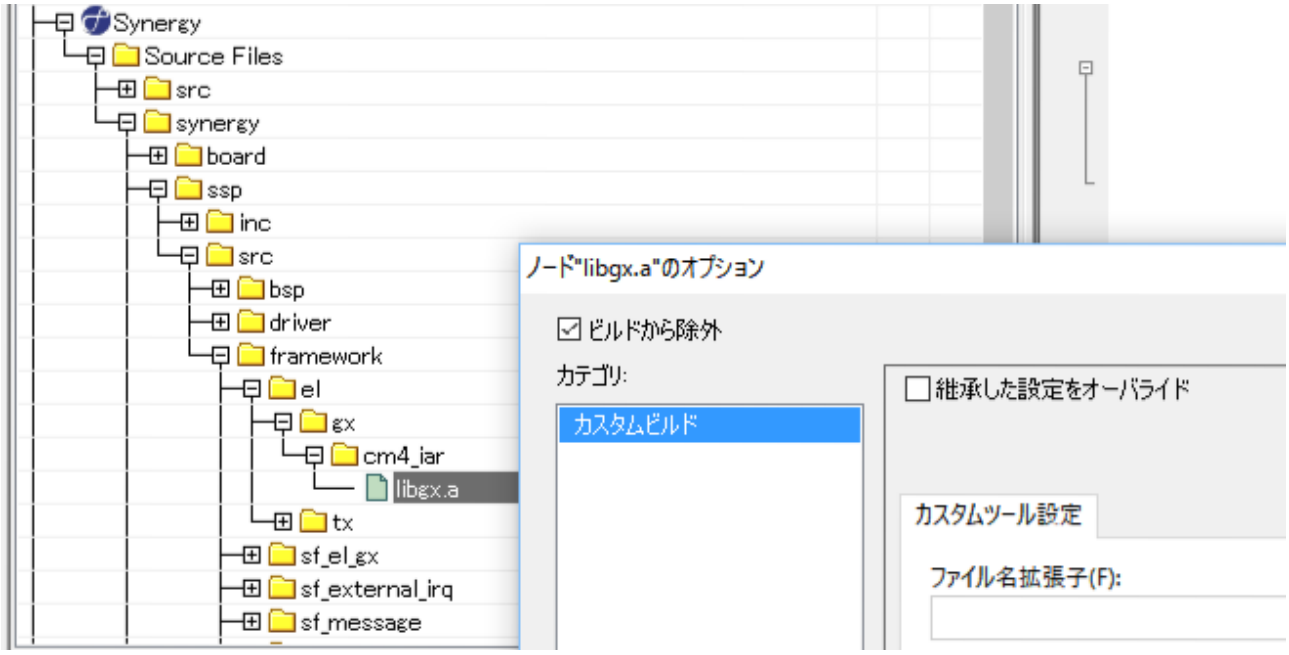

図 21 GUIX ライブラリの除外（IAR 環境）

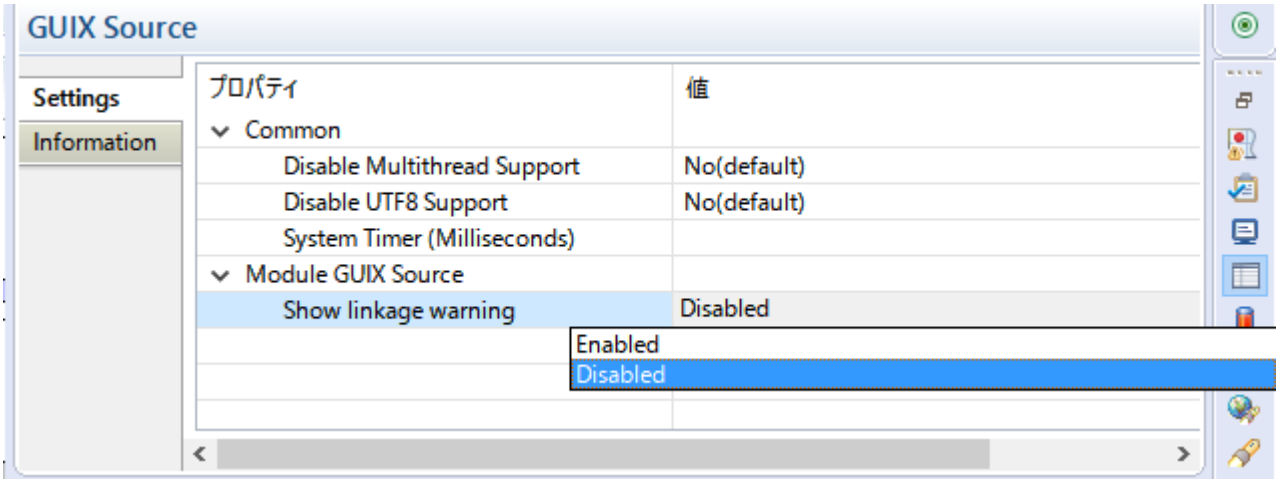

図 22　GUIX ライブラリのリンク警告の抑制

4.2 GUIX のカスタマイズ

4.1に示すように GUIX Source をプロジェクトに追加し、GUIX をソースコードからビルドするようにした場合、GUIX の各種設定を変更することができます。この時、GUIX Source のプロパティにある項目（図 22 参照）以外については、synergy¥ssp¥inc¥framework¥el¥gx_api.h で行います。例えば、同ファイルでは、ボタンのオートリピート入力に関する GX_REPEAT_BUTTON_INITIAL_TICS、および、GX_REPEAT_BUTTON_REPEAT_TICS が定義されていますが、設定されているこれらの初期値を変更する場合、各開発環境のプリプロセッサの設定において定義を行います（図 23、図 24）。

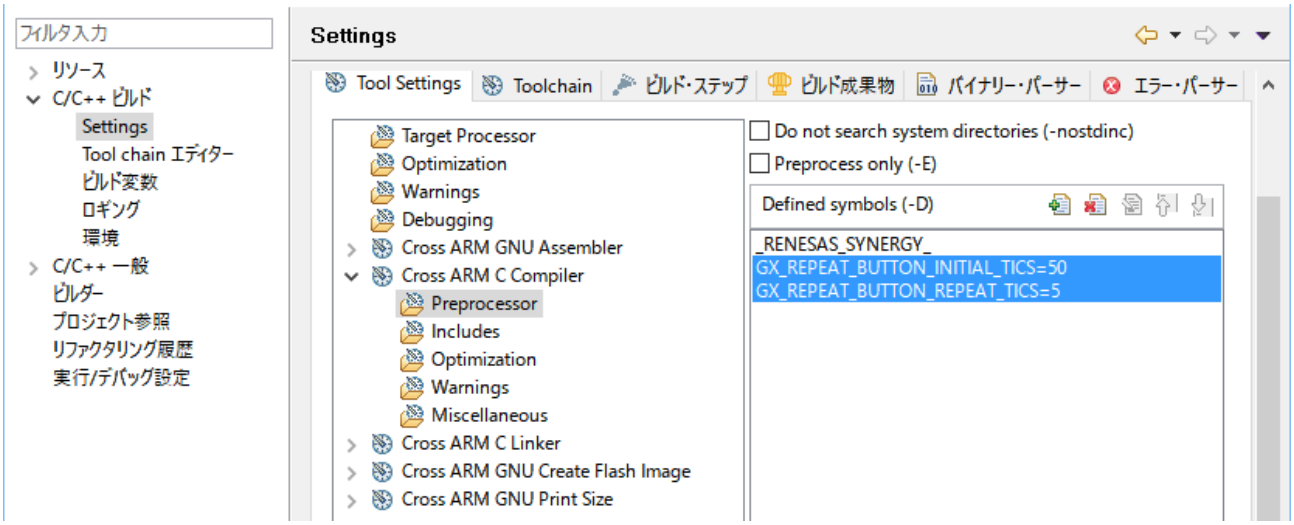

図 23 GUIX のカスタマイズ（GNU）

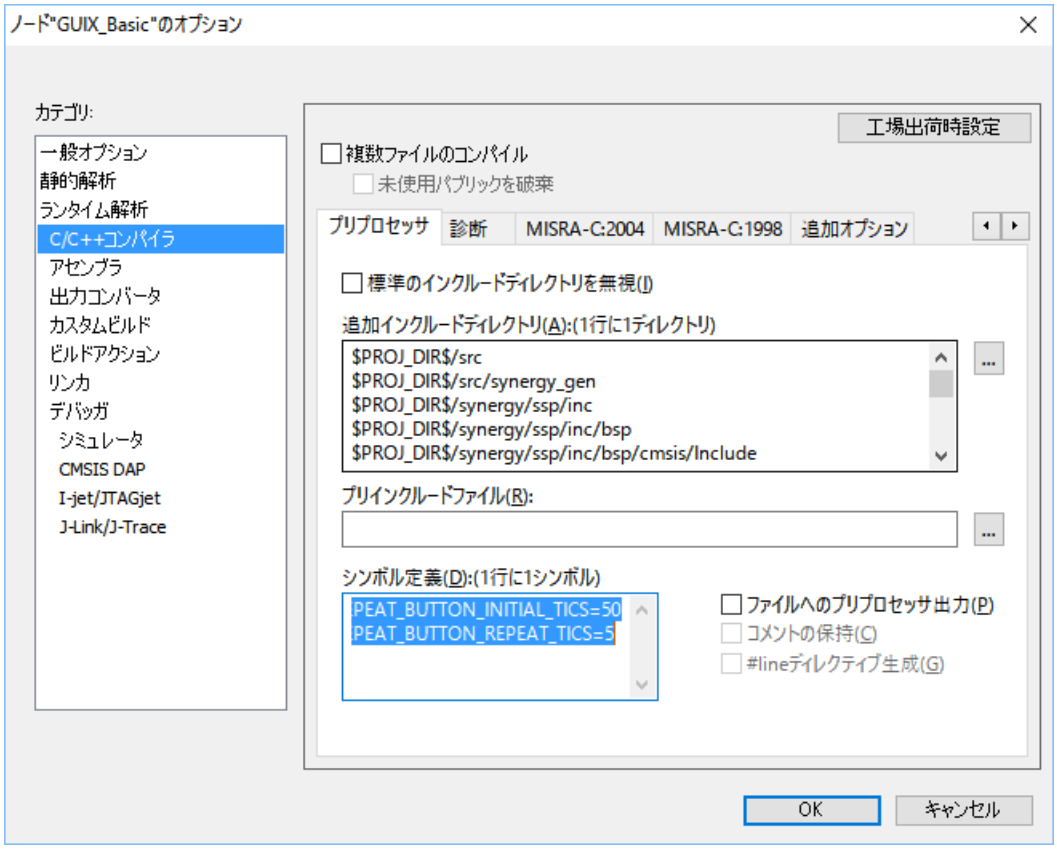

図 24 GUIX のカスタマイズ（IAR）

4.3 GUIX システムタイマ

GUIX はシステム起動時の gx_system_initailize 関数の処理において ThreadX の timer を作成し、GUIX におけるタイマに関する処理はこれを利用しています。初期値は図 25のように 20msec となっています。これは、GUIX Source のプロパティにある System Timer で変更できます（図 22）。

```
/* Default 20ms GUIX system timer.  */
#ifndef GX_SYSTEM_TIMER_MS
#define GX_SYSTEM_TIMER_MS       20
#endif

#if defined(GX_THREADX_BINDING)

/* Set default ThreadX timer tick frequency 100Hz (10ms timer). */
#ifndef TX_TIMER_TICKS_PER_SECOND
#define TX_TIMER_TICKS_PER_SECOND ((ULONG)100)
#endif

/* Derive GX_SYSTEM_TIMER_TICKS based on GX_SYSTEM_TIMER_MS value. */
#ifndef GX_SYSTEM_TIMER_TICKS
#define GX_SYSTEM_TIMER_TICKS     ((GX_SYSTEM_TIMER_MS * TX_TIMER_TICKS_PER_SECOND) / 1000)
#endif

#endif /* GX_THREADX_BINDING */
```

図 25　GUIX システムタイマの初期値 (gx_api.h)

4.4 Theme の追加

Theme を追加する場合、GUIX Studio において、Configure -> Themes を選択し、Configure Themes ダイアログを表示します。同ダイアログにおいて、Number of Themes for display にある▲ボタンを押すと、Theme が 1 つ増えます。次に、Theme Index を追加された Index に設定し、Theme Name にその Theme の名称を設定し、Save ボタンを押します。

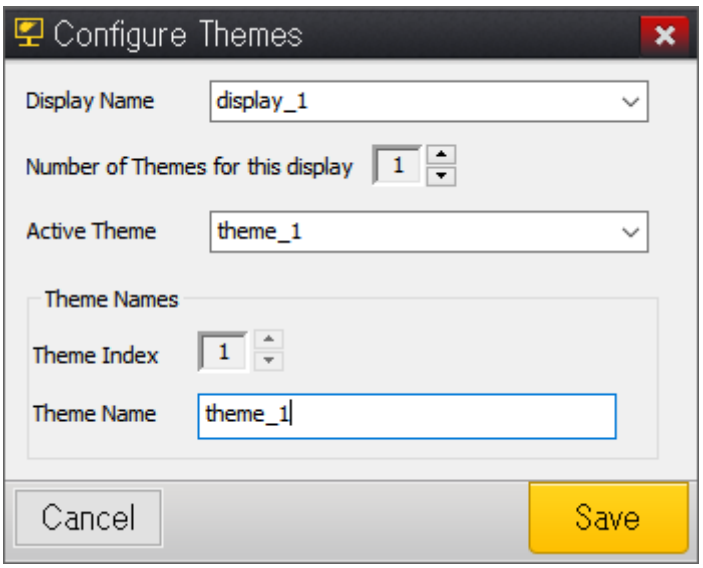

図 26 Theme の追加

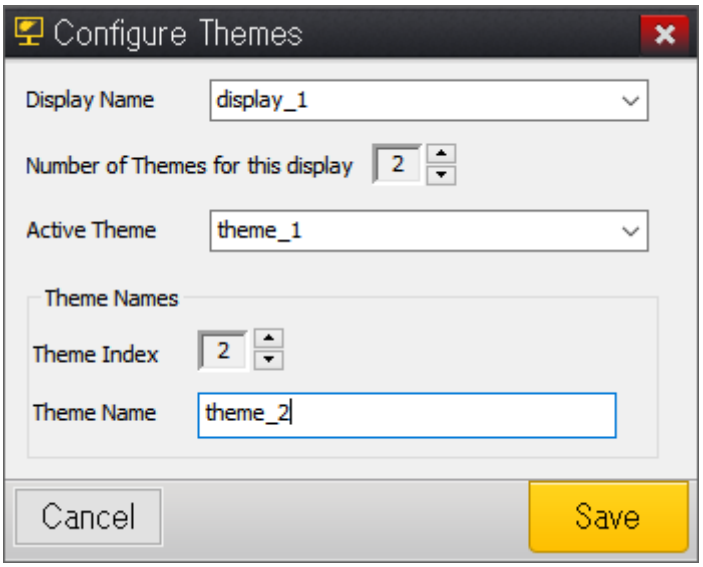

図 27 Theme Name の設定

4.5 言語の追加

言語を追加する場合、GUIX Studio において、Configure -> Languages を選択し、Configure Languages ダイアログを表示します。同ダイアログにおいて、Add Language ボタンを押すと、Number of Languages が 1 つ増えます。次に、Language Index を追加された Index に設定し、設定したい言語の Language ID を設定します。

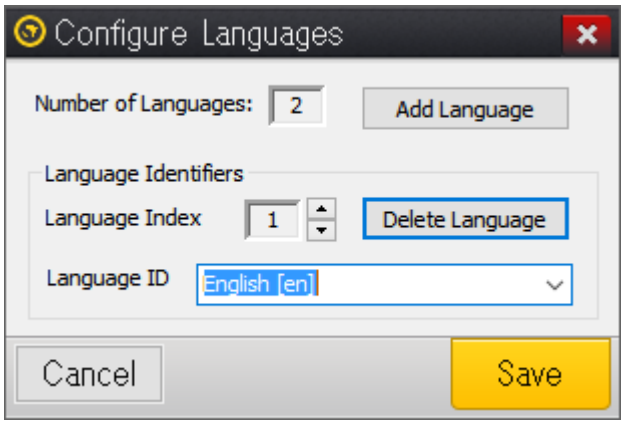

図 28 言語の追加

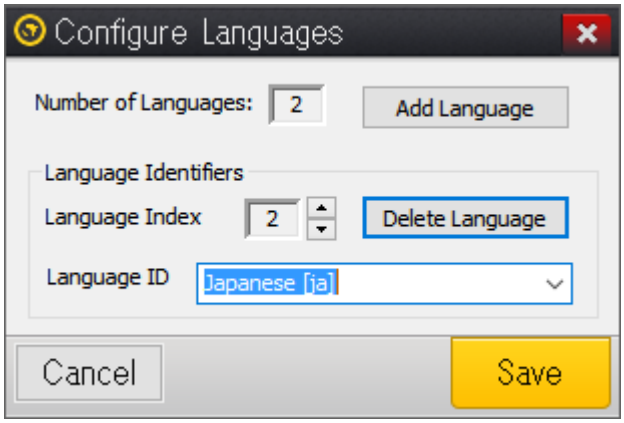

図 29 追加言語の指定

4.6 フォント

4.6.1 フォントの追加

GUIX Studio において、Fonts の下部にある Add New Font をクリックします（図 30）。表示される Edit Font ダイアログ（図 31）において、Browse ボタンを押し、フォントファイルを指定します。指定できるフォントファイルは True Type Font です。<proj_dir>の左の▼ボタンを押すと、フォントファイルのパスを、Project Relative Path, Studio Relative Path, Absolute Path から指定できます。その他、Font Name、Font Height、Font Format を適当に設定します。

フォントデータとして出力する文字を指定する設定は、”Include character set defined by String Table”、”Support Extended Character Range”、および、”Include the following character ranges”の下の各文字種別のチェックボックスとそれらの文字コード範囲を指定する先頭の値と末尾の値で行います。“Include character set defined by String Table” を有効にすると、Strings で定義された文字に関するフォントデータを生成します。”Support Extended Character Range”を有効にすると、 “Include character set defined by String Table” の末尾に文字範囲を指定する欄が４つ追加され、それらでは Private use character range として、0x0000 から 0x10ffff まで指定可能になります。ただし、GX_EXTENDED_UNICODE_SUPPORT を有効にして GUIX をビルドする必要があります。”Include the following character ranges”に列挙されている各文字種別のチェックボックスを有効にすると、それらの右にある先頭の値と末尾の値の範囲の全文字についてのフォントデータを生成します。この２つの数値は変更が可能となっているので、表示する文字に合わせて数値を調整することができます。例えば、日本語の”、”(U+3001)、や、”。”(U+3002)は、初期設定では含まれていません。表示されるようにする場合は、Hiragana の範囲をそれらが含まれる値に設定する必要があります。

“Include character set defined by String Table”、”Include the following character ranges”の両方を指定した場合は両方の選択が有効となり、結合された全選択文字についてのフォントデータが生成されます。

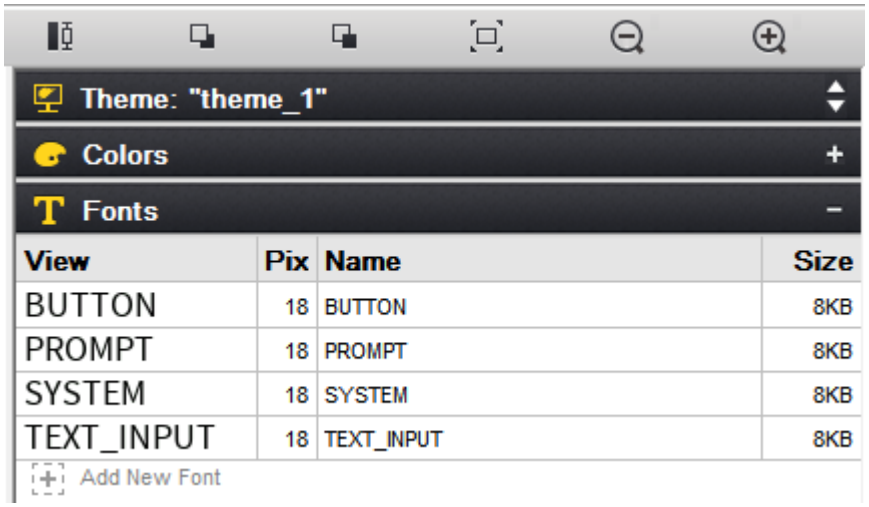

図 30 フォントの追加

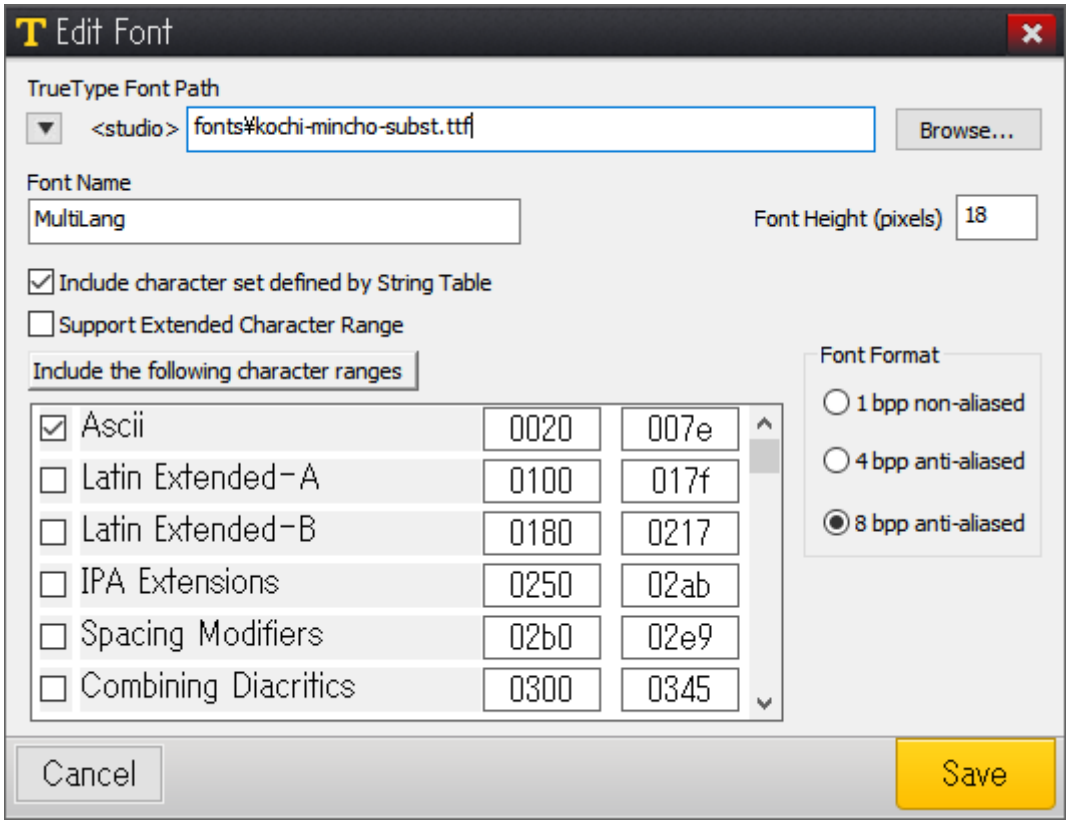

図 31　フォントの設定

4.6.2　フォントリソースのデータサイズ

日本語フォントを使用する場合、そのデータサイズの大きさが問題になることがあります。システムとして、表示する文字が GUIX Studio の Strings で定義したものに限られる場合は、フォントの設定において、“Include character set defined by String Table” にチェックを入れ、"Include the following character ranges"にある各チェックを外すことで、Strings で定義された文字に関するフォントデータだけが生成されるので、データサイズを小さくすることができます。また、Font Height を小さくすることや、Font Format にあるアンチエイリアス設定を 1bpp non-aliased に変更することもデータサイズ抑制につながります。

外部から入力される文字列を表示するシステムの場合、表示する可能性のある全ての文字についてフォントデータを生成する必要があります。この場合は、”Include the following character range”にある各文字種別のチェックボックスを有効にして、２つの数値の間の全文字コードについてのフォントデータを生成するようにします。

4.7 追加言語の文字列設定

Strings に登録されている各文字列について、追加言語の文字列を追加します。GUIX Studio の Strings を右クリックし、”View / Edit Strings...”を選択します。表示された String Table Editor において、図 32の赤枠で囲んだマークをクリックして、目的の言語の欄を表示します（図 33）。Strings Text の上段にある文字列に対応する該当言語の文字列を下段のボックスに入力します。

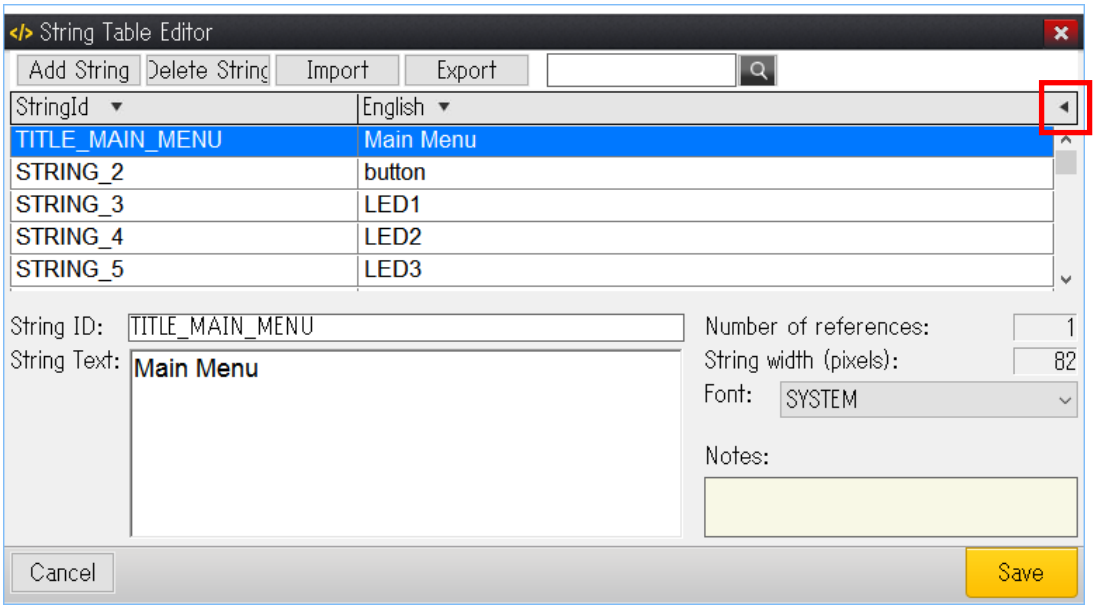

図 32 追加言語の文字列設定（String Table Editor のオープン）

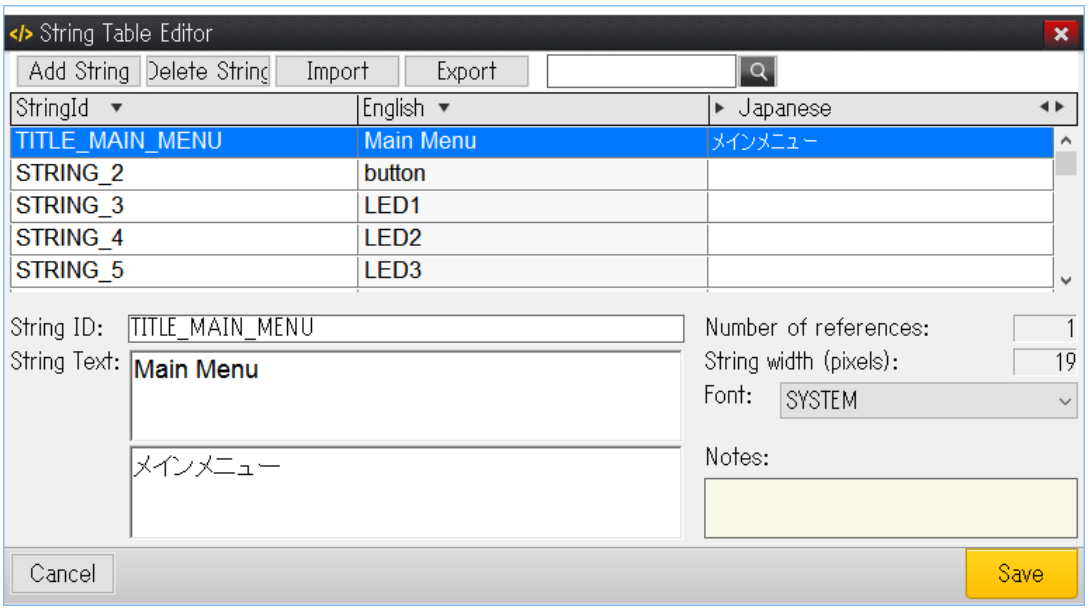

図 33 追加言語の文字列設定（文字列入力）

4.8 リソースデータの出力

GUIX のリソースデータを出力する場合、図 34に示すように各種のリソースをファイル指定して出力することができます。同図の場合は pixelmap のリソースデータが pixelmap_data.c, pixelmap_data.h に生成されます。また、図 35のように Pixelmap 毎に出力ファイルを指定することもできます。

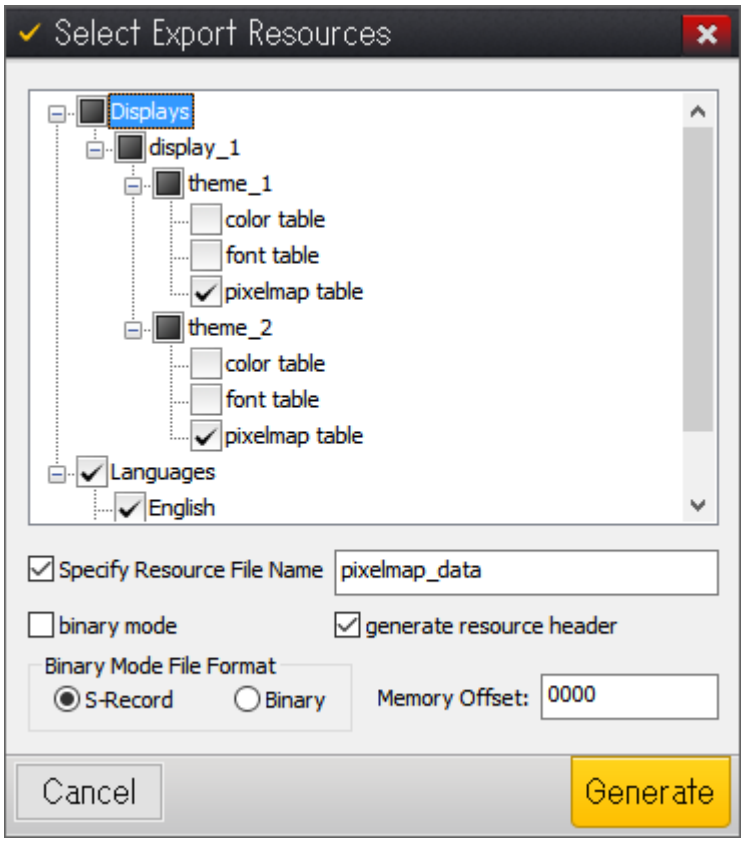

図 34 リソースデータの個別生成

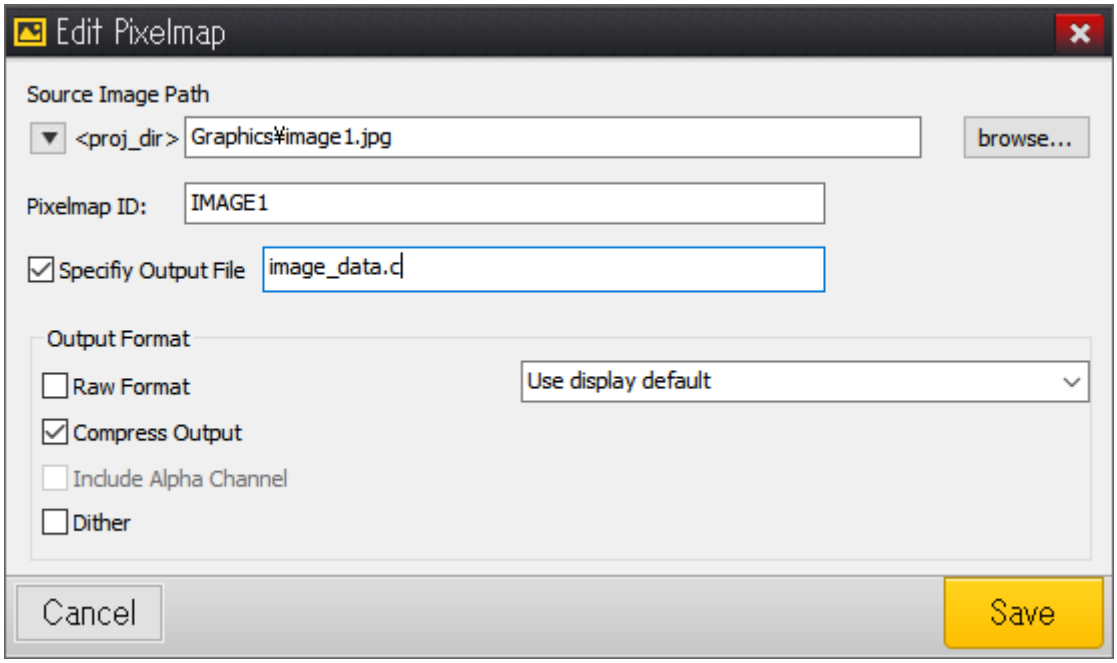

図 35 Pixelmap データ出力ファイルの指定

4.9 フレームバッファのデータ確認

Display Driver on r_glcd のプロパティにおいてフレームバッファを設定しますが、e² studio では、このフレームバッファのデータを画像イメージで表示させることができます。デバッガのメモリビューにおいて、フレームバッファのアドレスを入力します（図 36）。追加されたモニターアドレスにおいて、新規レンダリングの追加を行い、レンダリングとして RAW イメージを選択します（図 37）。フレームバッファのフォーマットに合わせて RAW イメージフォーマットを設定します（図 38）。プログラム実行を一時停止すると、現在のフレームバッファのイメージがメモリビューに表示されます（図 39）。

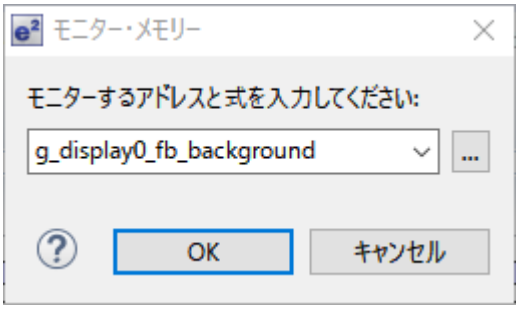

図 36 フレームバッファのデータ確認（フレームバッファのアドレス設定）

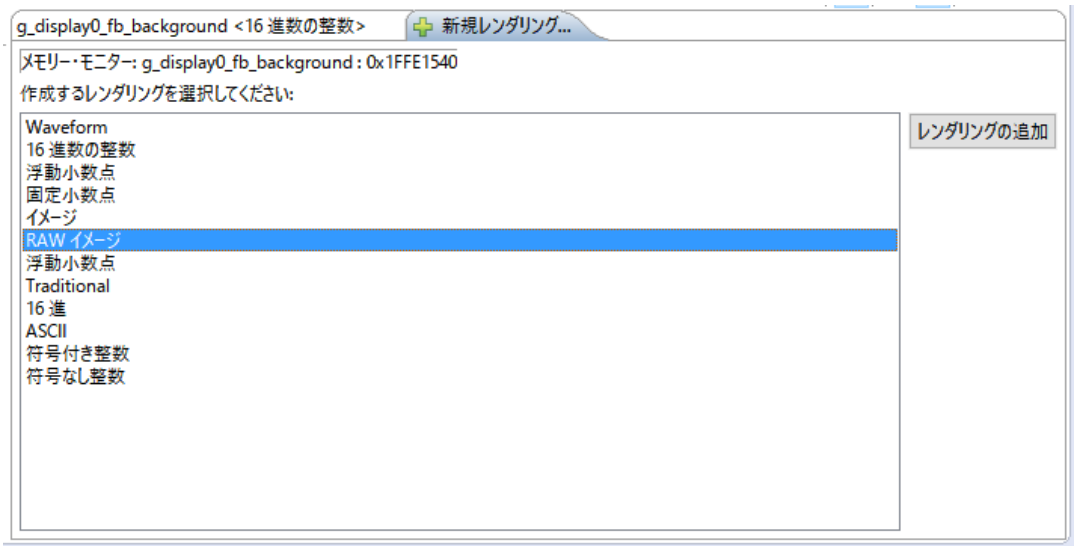

図 37 フレームバッファの確認（レンダリングの設定）

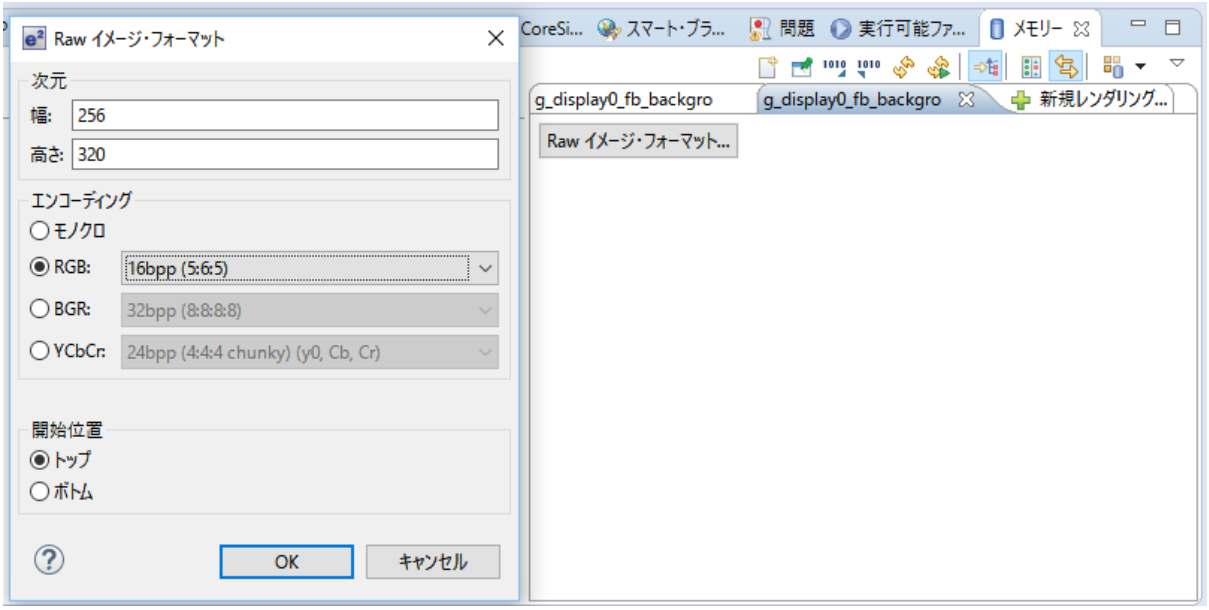

図 38　フレームバッファの確認（RAW イメージフォーマットの設定）

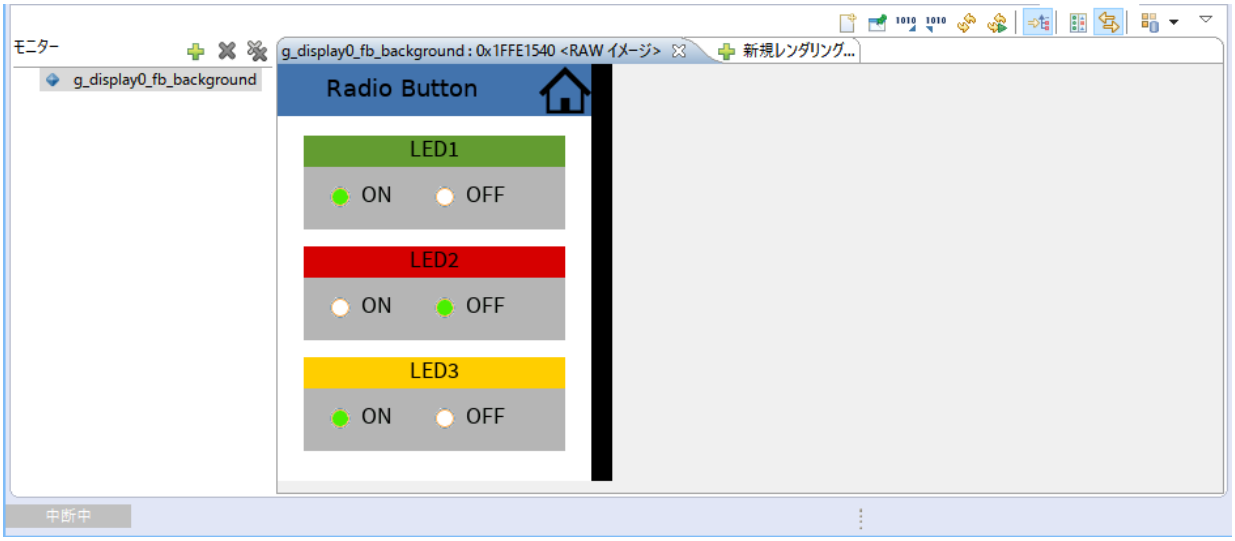

図 39　フレームバッファの確認（イメージ表示）

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018-06-05	-	初版

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部ROM、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、ノイズ輻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、その他の不適切に使用しないでください。かかる改造、改変、複製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、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、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できません。たとえ、意図し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。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   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. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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その総株主の議決権の過半数を直接または間接に保有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.3.0-1　2016.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/