μITRON 仕様リアルタイム OS シミュレータ

NORTi Simulator

ユーザーズガイド

2016 年 12 月版

MiSPO

株式会社ミスポ

はじめに

NORTi Simulator はターゲット CPU の命令セットをシミュレーションする「クロスシミュレータ」ではなく、x86 の命令セットで実行される「ネイティブコードシミュレータ」です。ターゲットに搭載するためのμITRON ベースのプログラムを、Microsoft Visual Studio でコンパイルし、NORTi Simulator と組み合わせることで、Windows 上で実行可能なプログラムとなります。

NORTi Simulator は、パソコン(PC)の Ethernet ポート(LAN ポート)や COM ポートを使えることが従来のシミュレータにはなかった大きな特長で、実際の通信が行える極めてターゲットに近いシミュレーションが可能です。I/O のシミュレーションについては、画面上にスイッチボックスを表示するコンポーネントがサンプルとして添付されています。これをユーザーがターゲットに合わせて拡張することにより、個々の入出力レベルのシミュレーション、さらにはシステム全体の総合的なシミュレーションが可能となります。

デバッグには Visual Studio のデバッガを使用することができます。そして、NORTi Simulator に付属のタスクステートウォッチャを併用することにより、カーネルが管理する各資源の状態をグラフィカルに観測することができます。

NORTi の TCP/IP プロトコルスタックも含めてのシミュレーションが可能ですので、特にネットワーク対応機器の開発では必須のツールです。

本書について

本書では、NORTi Simulator の概要と、導入方法や操作方法について説明しています。

NORTi Simulator 上で動作するリアルタイム OS「NORTi」そのものに関しては、次の共通のユーザーズガイドをご覧ください。

- no4guid.pdf … NORTi Version 4 ユーザーズガイド カーネル編
- n4nguid.pdf … NORTi Version 4 ユーザーズガイド TCP/IP 編（※）
- n4update.pdf … NORTi Version 4 ユーザーズガイド 補足説明書

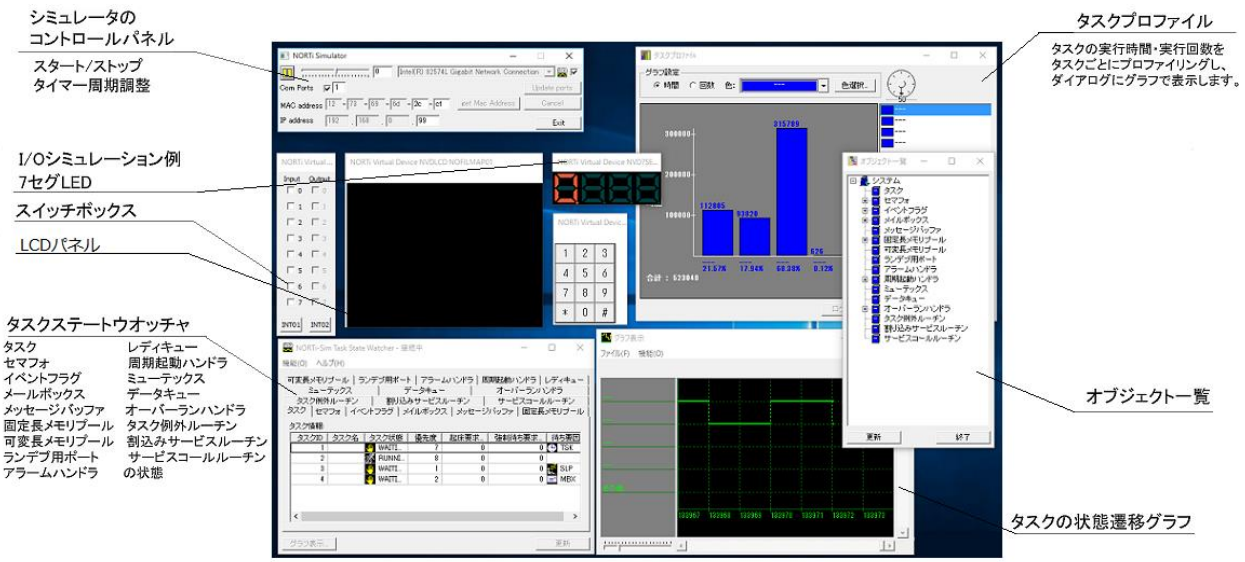
（※）第 5 版までは、「TCP/IP 編」でなく「ネットワーク編」という名称です。

目次

第１章 導入

1.1 特長

・μITRON のタスクを Windows のスレッドに割り当て、μITRON3.0/4.0 仕様のシステムコールをシミュレーション
・x86 系 CPU のネイティブコードで実行されるため、インストラクションセットシミュレータより高速
・ITRON TCP/IP の API 仕様と互換の API で、PC の LAN ポートを利用可能
・NORTi のシリアル入出力ドライバと互換の API で、PC の COM ポートを利用可能
・Windows の API も利用可能で、デバッグやシミュレーション用の入出力を自在に追加可能
・タスクステートウォッチャが装備され、各資源の状態とタスクの状態遷移をモニタ可能



1.2 基本的な仕組み

NORTi Simulator は、μITRON ベースのプログラムを Windows 上で実行、デバッグするためのライブラリとユーティリティから構成されています。
ユーザープログラムは DLL としてコンパイルされ、シミュレータ本体である nortisim.exe からこの DLL を呼び出して実行することでシミュレーションを行います。
NORTi Simulator は、μITRON のタスクを Windows のスレッドに割り当てることで、μITRON 仕様 OS 向けに書かれたプログラムを Windows 上でシミュレーションしています。
ユーザープログラムによってタスクが生成されると、シミュレータエンジンは Windows のスレッドを生成し、μITRON のタスクを割り当てます。各システムコールのシミュレーション実現の

ためには、NORTi カーネルと同等の管理ブロックおよび制御用コードがシミュレータエンジン内に実装されており、NORTi カーネルと同等のスケジューリングアルゴリズムによって実行すべきスレッドが選択されます。

NORTi Simulator の TCP/IP プロトコルスタックには、Windows のプロトコルスタックではなく、NORTi に付属のプロトコルスタックを使用しております。NORTi と完全互換の TCP/IP 通信を実現するため、Windows 用の LAN ドライバではなく、WinPcap のパケットドライバを使用しています。

タスクステートウォッチャはシミュレータ本体とは別プロセスとして実装されており、シミュレータエンジンとプロセス間通信を行うことによって、タスクやイベントフラグやメールボックス等のオブジェクトの状態表示と、タスクの状態遷移グラフ表示を実現しています。

タスクは Windows のスレッドのひとつですので、μITRON のシステムコールだけでなく Windows の API を使うことができ、それを活用することによって周辺のシミュレーション機能を実現することもできます。

1.3 NORTi Simulator 使用上の制限

1.3.1 対応する Windows のバージョン

NORTi Simulator の実行環境としては、Windows 7、Windows 8.1、または Windows 10 を推奨します。それぞれ 32 ビット、64 ビットいずれのバージョンでも利用可能ですが、Visual Studio 2015 を快適に利用できる 64 ビット OS の利用を推奨いたします。
Windows Vista 以前、ならびに各種サーバ OS はサポート対象外です。

1.3.2 対応するハードウェア

上記 Windows を実行できる PC であればハードウェアに関して特に制限はありませんが、Visual Studio をストレスなく実行するため、4GB 以上の RAM を搭載した PC を推奨します。また、シリアル通信を実行するプログラムのために COM ポートならびに LAN ポートが装備されている必要があります。

1.3.3 対応コンパイラ

Visual Studio 2015(Community 版を含む)が使用できます。Visual Studio 2015 をインストールする際は、インストールするタイプで Custom を選択し、Programming Languages にチェックをつけてください。これによって C/C++言語で書かれたソースをコンパイルすることができるようになります。

1.3.4 16 ビットマイコン(H8, M16C 等)でご利用のお客様へ

NORTi では int 型のサイズが 16 ビットの CPU もサポートしていますが、NORTi Simulator では int 型が 32 ビットとなります。ITRON で再定義してある INT、UINT 等も全てカーネル編ユーザーズガイドの「データタイプ(32 ビット CPU の場合)」が適用されますので、int 型のサイズに依存しないようにプログラムを記述してください。

1.3.5 NORTi3 Standard をご使用のお客様へ

NORTi Simulator は、NORTi Version 4 をベースとしており、NORTi Version 4 と同様に NORTi3 Extended 互換のシステムコールもサポートしています。しかし、そのサブセットである NORTi3 Standard とは完全に互換ではありません。NORTi3 Standard と NORTi3 Extended との違いをご理解の上、NORTi Simulator をご利用ください。

1.3.6 NORTi Oceans をご使用のお客様へ

同様に、NORTi Simulator は、NORTi Version 4 のサブセットである NORTi Oceans とも完全に

互換ではありません。NORTi Oceans と NORTi Version 4 との違いをご理解の上、NORTi Simulator をご利用ください。

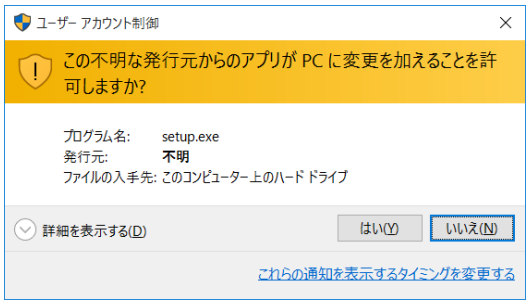
1.3.7 その他の制限事項

NORTi Simulator は、実際のターゲット CPU やクロスコンパイラを使った場合と異なる実行結果となる場合があります。具体的には、「第 3 章 プログラミング上の注意事項」を参照してください。

1.4 インストール

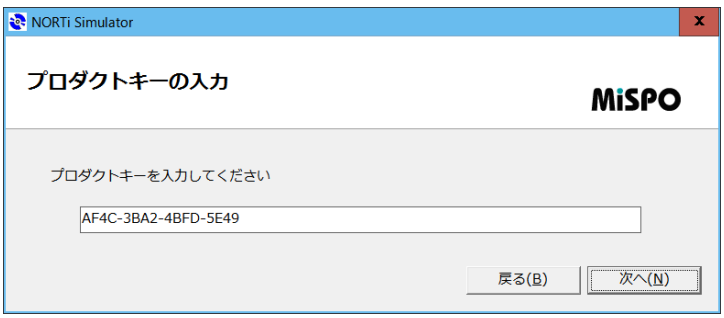
1.4.1 NORTi Simulator のインストール

管理者権限のあるユーザーアカウントで Windows にログインし、CD-ROM からインストーラ Setup.exe を起動してください。起動後は下記のダイアログが順に表示されますので、それに従ってインストールを進めてください。

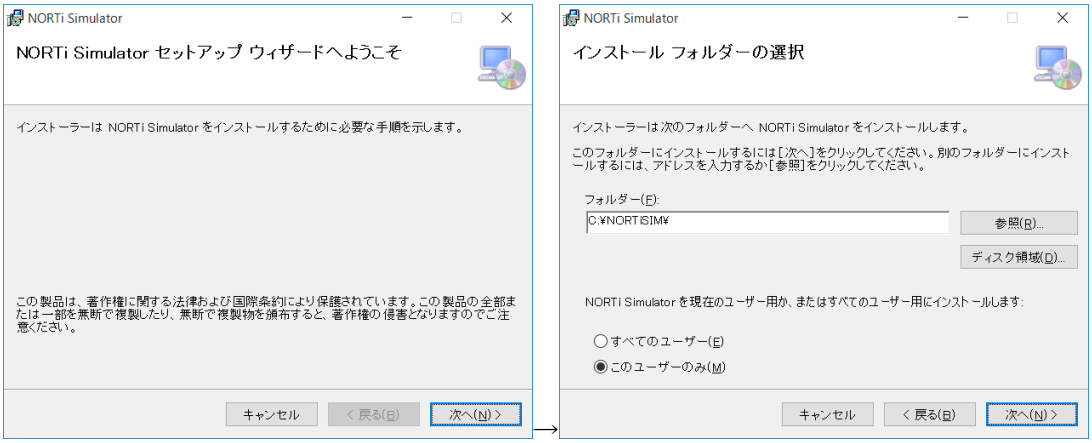


上記の警告画面が出た場合は「はい」を選んでください。

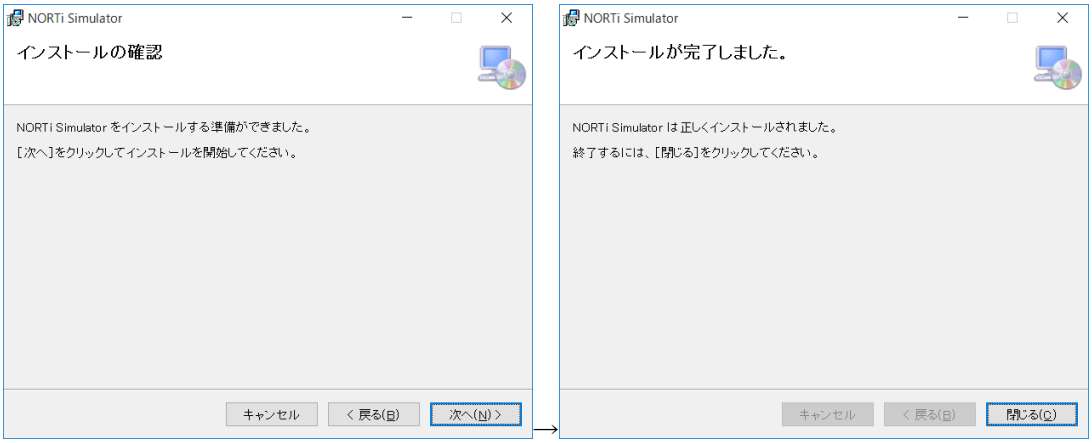
選択できる WinPcap のインストールについては、1.4.2 を参照してください。
VC++ 2008(x86)ランタイムライブラリのインストールついては、1.4.3 を参照してください。



プロダクトキーは自動的に入力されますので、「次へ」を押してください。



デフォルトのインストール先フォルダは、C:¥NORTiSIM です。本書ではこのフォルダへインストールされることを前提に解説します。



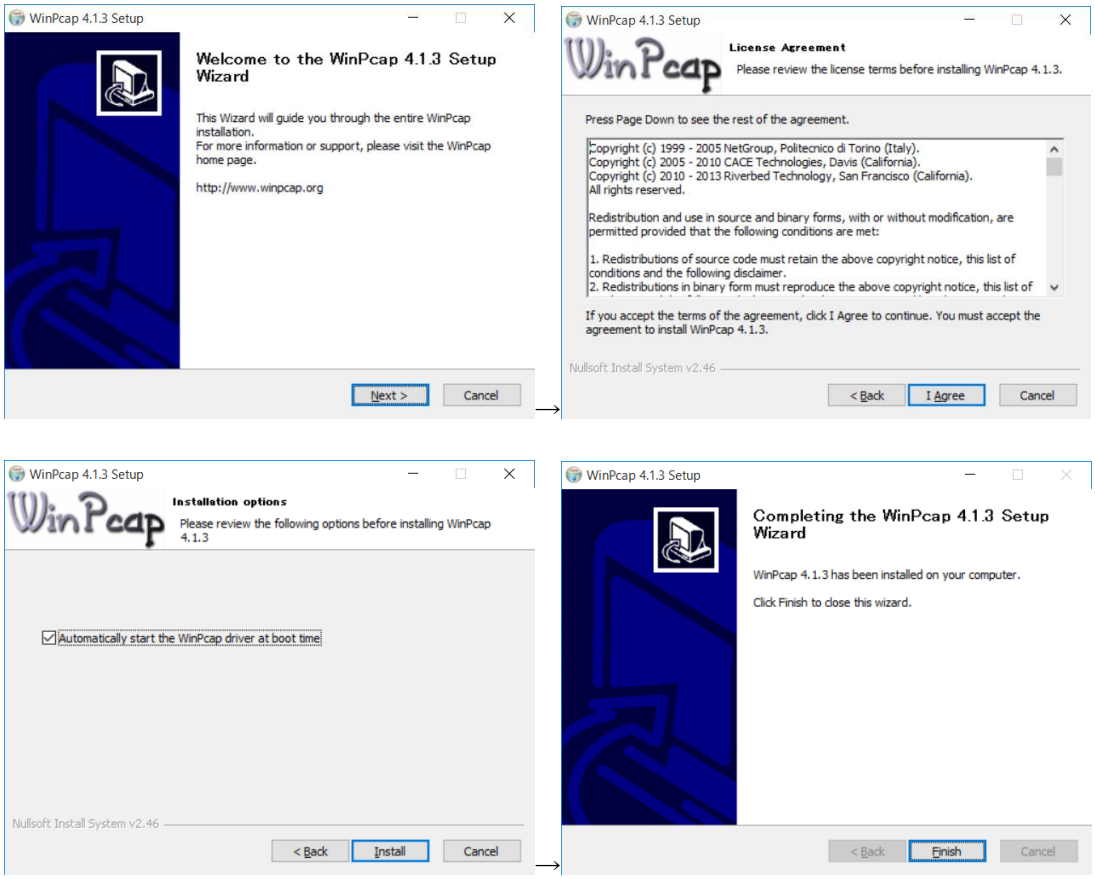
以上で NORTi Simulator のインストールは完了です。

1.4.2 パケットキャプチャライブラリのインストール

NORTi Simulator では、Windows の TCP/IP スタックを介さずに、LAN で送受信するパケットを直接扱いますが、そのためには WinPcap パケットキャプチャライブラリバージョン 4 以上が必要です。

すでに Wireshark 等を使用されていて、最新バージョンの WinPcap がインストールされている場合にはインストールの必要はありません。

デフォルトでは NORTi Simulator 本体のインストールの次に WinPcap のインストーラが起動しますので、画面の指示に従ってインストールを進めてください。

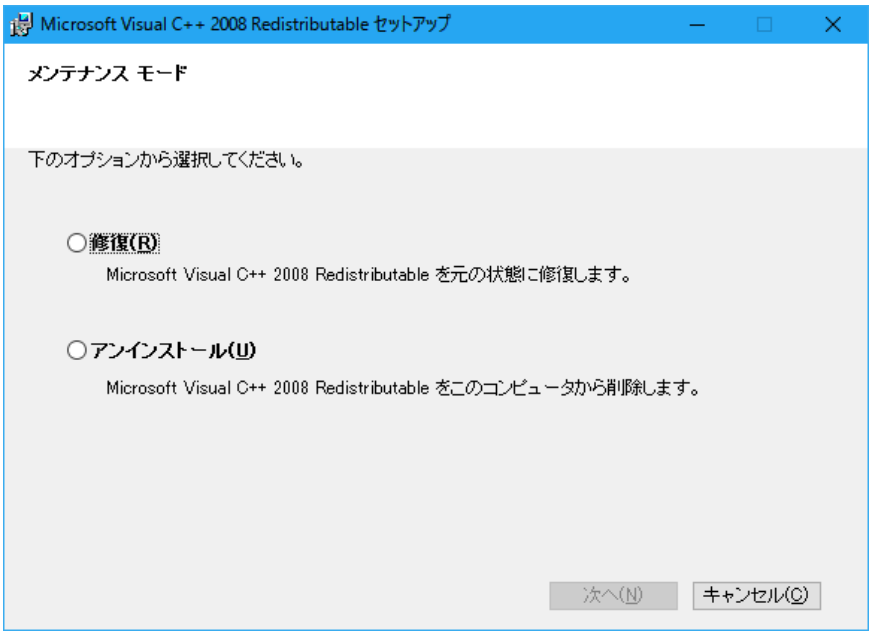


WinPcap パケットキャプチャライブラリに関する詳細は下記 URL を参照してください。

http://www.winpcap.org/

1.4.3 VC++ランタイムライブラリのインストール

NORTi Simulator を起動するのに必要な VC++ 2008(x86)のランタイムライブラリが、Windows 7 ではインストールされていない場合があります。そのため、Windows 7 でご利用の場合は、「VC++ 2008(x86)ランタイムライブラリのインストール」を選択してください。
すでにランタイムライブラリがインストールされている場合には下記の画面が現れますので、ここでキャンセルしてください。

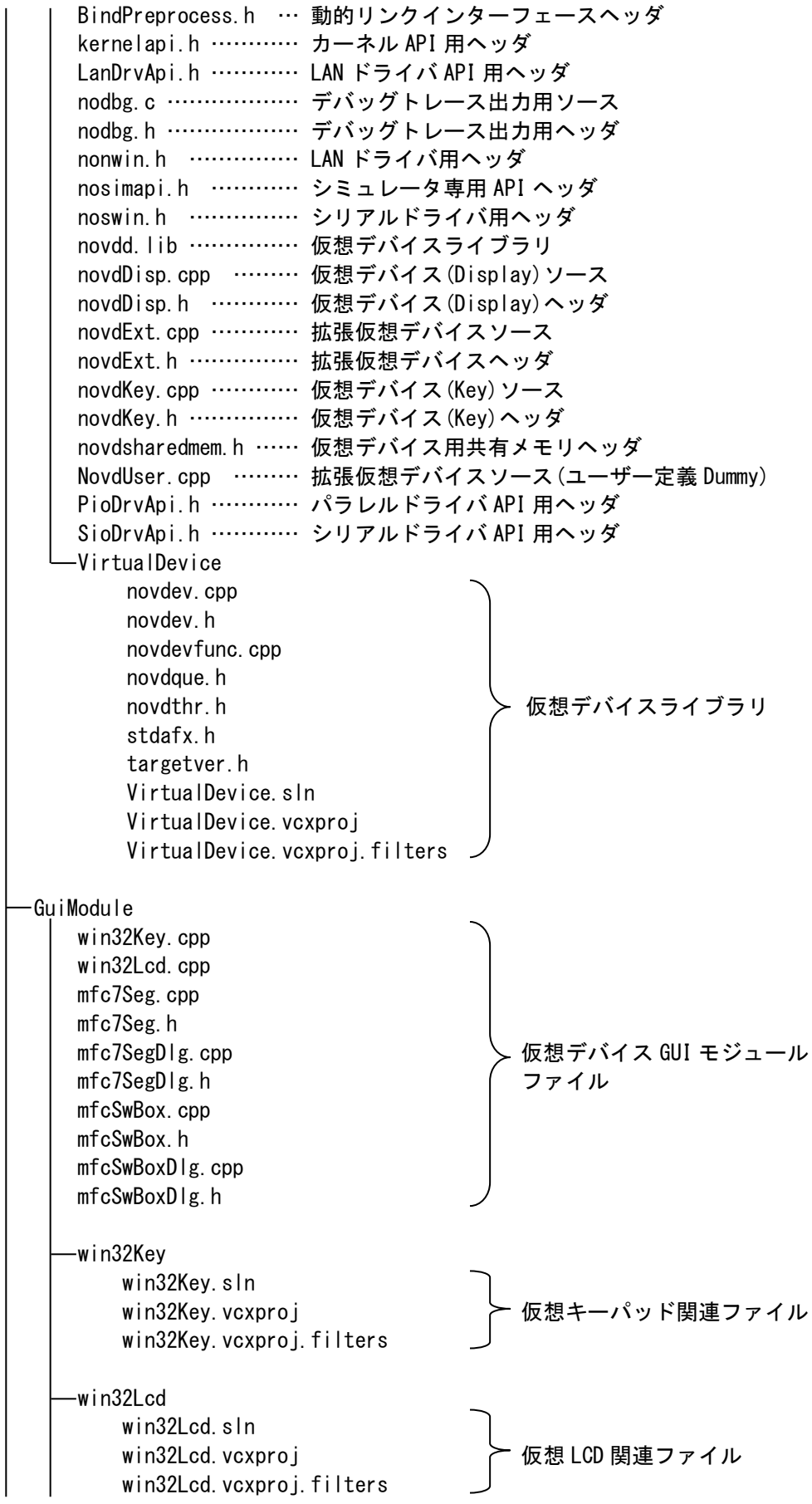


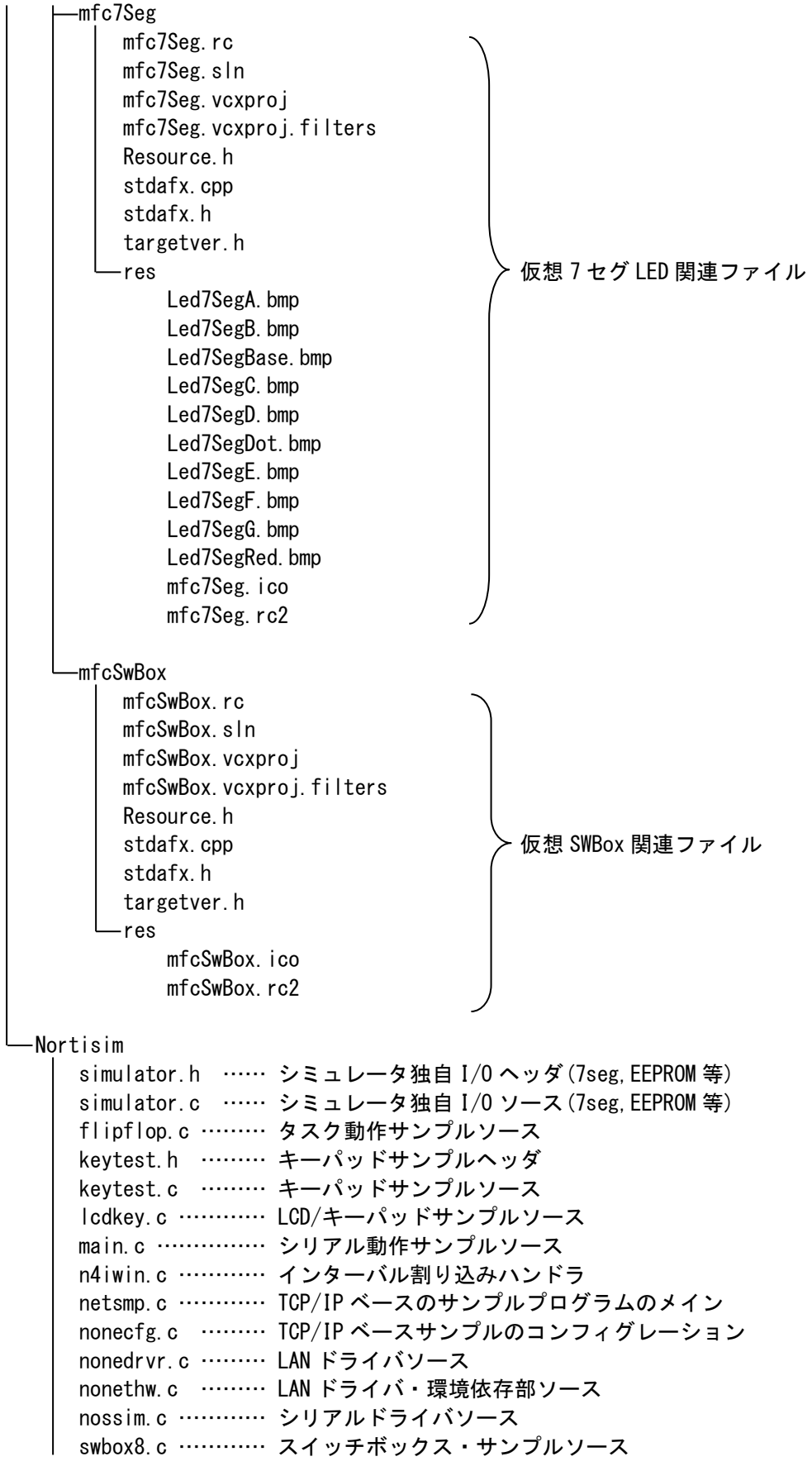
1.5 ファイル構成

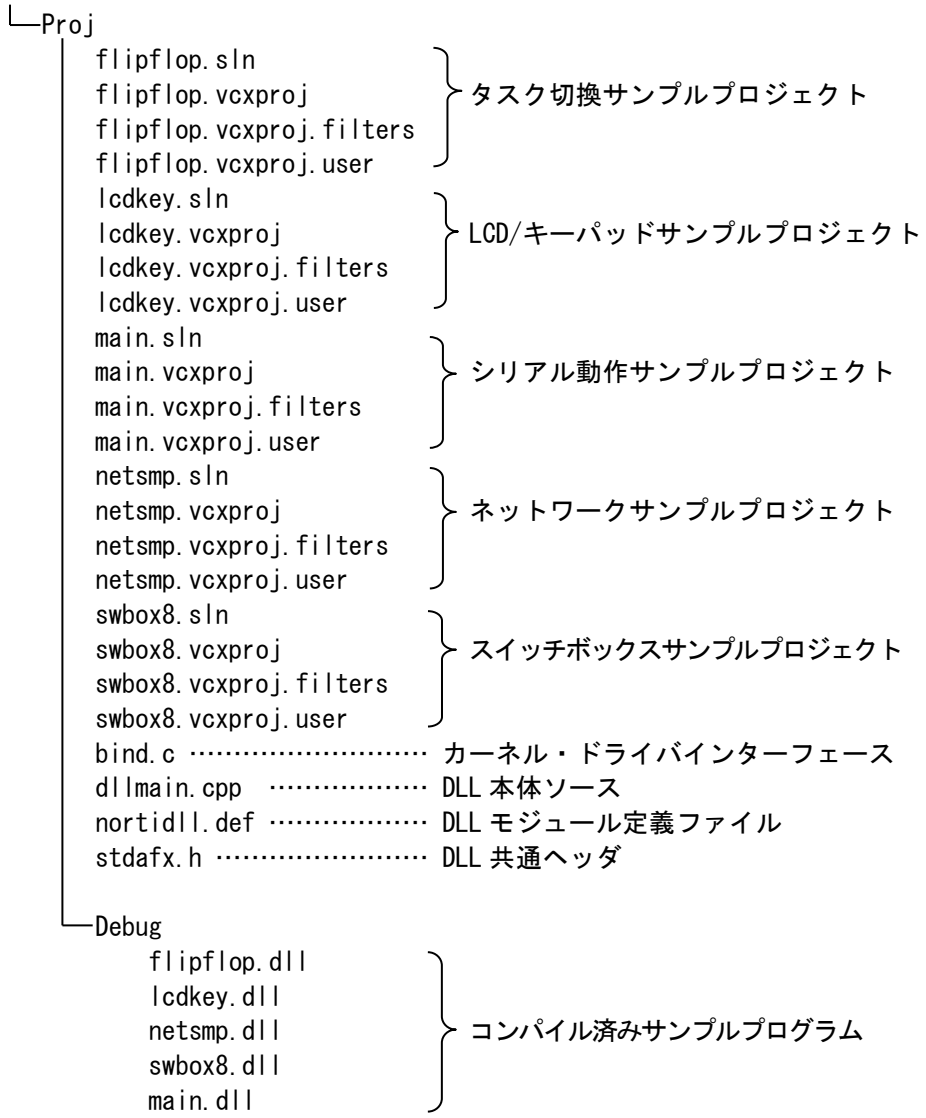
```
NORTiSIM
 ├─BIN (DLL と実行ファイル)
 │      MFC71.dll  ····· MFC ランタイム
 │      msvcr71.dll  ····· Visual C++ ランタイム
 │      nagctl43.dll ····· カーネルシミュレータエンジン 1/2
 │      nagitr43.dll ····· カーネルシミュレータエンジン 2/2
 │      NagTSWN4.exe ····· タスクステートウォッチャ(TSW)
 │      NagTSWN4.tlb ····· TSW 用タイプライブラリ
 │      nortisim.exe ····· シミュレータ本体(アプリ DLL 起動用 EXE)
 │      nvd7seg.exe  ····· 仮想デバイス(7segLED)
 │      nvdkey.exe ····· 仮想デバイス(キーパッド)
 │      nvdlcd.exe ····· 仮想デバイス(LCD)
 │      nvdswbox.exe ····· 仮想デバイス(SW ボックス)
 │
 ├─DOC (ドキュメント)
 │      no4sim.pdf ····· NORTi Simulator ユーザーズガイド(本書)
 │      NORTi4_SIM_4??.txt ····· NORTi Simulator リリースノート
 │
 ├─INC (カーネルと TCP/IP スタックのヘッダファイル)
 │      itron.h  ····· ITRON 標準ヘッダ
 │      kernel.h ····· カーネル標準ヘッダ
 │      kernels.h  ····· シミュレータカーネルヘッダ
 │      nosys4.h ····· NORTi4 システム内部定義ヘッダ
 │      nocfg4.h ····· コンフィグレーションヘッダ
 │      n4rsim.h ····· CPU 差異定義ヘッダ(シミュレータ)
 │      nosio.h  ····· シリアル入出力関数ヘッダ
 │      nonet.h  ····· NORTi TCP/IP 標準ヘッダ
 │      nonetc.h ····· NORTi TCP/IP コンフィグレーションヘッダ
 │      nonets.h ····· NORTi TCP/IP 内部定義ヘッダ
 │      nonelan.h  ····· NORTi TCP/IP マルチチャンネル LAN ドライバヘッダ
 │      nonitod.h  ····· 数値/文字列変換関数ヘッダ
 │      nonloop.h  ····· ループバックドライバヘッダ
 │      norti3.h ····· NORTi3 互換用カーネル標準ヘッダ
 │      nosys3.h ····· NORTi3 互換用システム内部定義ヘッダ
 │      nocfg3.h ····· コンフィグレーションヘッダ
 │
 ├─SRC (カーネルと TCP/IP スタックのソースファイル)
 │      noknl4s.c  ····· シミュレータカーネルインターフェースのソース
 │      nosio.c  ····· シリアル入出力関数のソース
 │      nonet.c  ····· NORTi TCP/IP サービスコールのソース
 │      nonetsim.c ····· TCP/IP シミュレータ依存部のソース
 │      nontcp1.c  ····· TCP モジュールのソース 1/2
 │      nontcp2.c  ····· TCP モジュールのソース 2/2
 │      noneudp.c  ····· UDP モジュールのソース
 │      nonicmp.c  ····· ICMP モジュールのソース
 │      nonearp.c  ····· ARP モジュールのソース
 │      nonetip.c  ····· IP モジュールのソース
 │      noneipf.c  ····· IP フラグメントモジュールのソース
 │      nonigmp.c  ····· IGMP モジュールのソース
 │      nonigmp3.c ····· IGMPv3 モジュールのソース
```

```
        noneraw.c  ····· RAW モジュールのソース
        nonelan.c  ····· ネットワークドライバインターフェースのソース
        nonigmp3.c ····· IGMPv3 モジュールのソース
        nonloop.c  ····· ループバックドライバのソース
        nonslib.c  ····· C 標準ライブラリに不足の関数のソース

├─LIB (ライブラリ)
│   └─SIM
│       │  n4dsim.lib  ····· TCP/IP スタックのライブラリ
│       └─PROJ  ····· 同ライブラリ生成用のプロジェクトフォルダ
│              n4dsim.sln
│              n4dsim.vcxproj
│
├─NETSMP  (TCP/IP アプリケーションサンプル)
│   ├─INC
│   │      noncons.h ····· コンソール入出力関数のヘッダ
│   │      nonteln.h ····· Telnet クライアント/サーバのヘッダ
│   │      nonftp.h  ····· FTP クライアント/サーバのヘッダ
│   │      nontftp.h ····· TFTP サーバのヘッダ
│   │      nonecho.h ····· Echo クライアント/サーバのヘッダ
│   │      nondhcp.h ····· DHCP クライアントのヘッダ
│   │      nonfile.h ····· 簡易ファイルシステムのヘッダ
│   │      nonsntp.h ····· SNTP クライアントのヘッダ
│   │      sntptime.h  ····· 時刻変換関数のヘッダ
│   │      sntpcfg.h ····· SNTP コンフィグレーションヘッダ
│   │      nonttcp.h ····· ttcp コマンド利用 TCP 性能測定プログラムのヘッダ
│   └─SRC
│          nonshel.c ····· コンソールと Telnet クライアントのシェルのソース
│          noncons.c ····· コンソール入出力関数のソース
│          nontelc.c ····· Telnet クライアントのソース
│          nonteld.c ····· Telnet サーバのソース
│          nonftpc.c ····· FTP クライアントのソース
│          nonftpd.c ····· FTP サーバのソース
│          nontftp.c ····· TFTP サーバのソース
│          nonechc.c ····· Echo クライアントのソース
│          nonecho.c ····· Echo サーバのソース
│          nondhcp.c ····· DHCP クライアントのソース
│          nonedns.c ····· DNS クライアントのソース
│          nonping.c ····· ping コマンドのソース
│          nondump.c ····· LAN パケットダンプ用のソース
│          nonfile.c ····· 簡易ファイルシステムのソース
│          nonramd.c ····· RAM ディスクのソース
│          nonsntp.c ····· SNTP クライアントのソース
│          sntptime.c  ····· 時刻変換関数のソース
│          nonttcp.c ····· ttcp コマンド利用 TCP 性能測定プログラムのソース
│
└─SMP (サンプルプログラム)
    └─SIM
        ├─Common
        │    │  BindFuncDef.h ····· 動的リンクインターフェースヘッダ
        │    │  BindPostprocess.h ··· 動的リンクインターフェースヘッダ
```


BindPreprocess.h … 動的リンクインターフェースヘッダ
kernelapi.h …… カーネル API 用ヘッダ
LanDrvApi.h …… LAN ドライバ API 用ヘッダ
nodbg.c …… デバッグトレース出力用ソース
nodbg.h …… デバッグトレース出力用ヘッダ
nonwin.h …… LAN ドライバ用ヘッダ
nosimapi.h …… シミュレータ専用 API ヘッダ
noswin.h …… シリアルドライバ用ヘッダ
novdd.lib …… 仮想デバイスライブラリ
novdDisp.cpp …… 仮想デバイス(Display)ソース
novdDisp.h …… 仮想デバイス(Display)ヘッダ
novdExt.cpp …… 拡張仮想デバイスソース
novdExt.h …… 拡張仮想デバイスヘッダ
novdKey.cpp …… 仮想デバイス(Key)ソース
novdKey.h …… 仮想デバイス(Key)ヘッダ
novdsharedmem.h …… 仮想デバイス用共有メモリヘッダ
NovdUser.cpp …… 拡張仮想デバイスソース(ユーザー定義 Dummy)
PioDrvApi.h …… パラレルドライバ API 用ヘッダ
SioDrvApi.h …… シリアルドライバ API 用ヘッダ
VirtualDevice
novdev.cpp
novdev.h
novdevfunc.cpp
novdque.h
novdthr.h
stdafx.h
targetver.h
VirtualDevice.sln
VirtualDevice.vcxproj
VirtualDevice.vcxproj.filters
仮想デバイスライブラリ
GuiModule
win32Key.cpp
win32Lcd.cpp
mfc7Seg.cpp
mfc7Seg.h
mfc7SegDlg.cpp
mfc7SegDlg.h
mfcSwBox.cpp
mfcSwBox.h
mfcSwBoxDlg.cpp
mfcSwBoxDlg.h
仮想デバイス GUI モジュール
ファイル
win32Key
win32Key.sln
win32Key.vcxproj
win32Key.vcxproj.filters
仮想キーパッド関連ファイル
win32Lcd
win32Lcd.sln
win32Lcd.vcxproj
win32Lcd.vcxproj.filters
仮想 LCD 関連ファイル


mfc7Seg
mfc7Seg.rc
mfc7Seg.sln
mfc7Seg.vcxproj
mfc7Seg.vcxproj.filters
Resource.h
stdafx.cpp
stdafx.h
targetver.h
res
Led7SegA.bmp
Led7SegB.bmp
Led7SegBase.bmp
Led7SegC.bmp
Led7SegD.bmp
Led7SegDot.bmp
Led7SegE.bmp
Led7SegF.bmp
Led7SegG.bmp
Led7SegRed.bmp
mfc7Seg.ico
mfc7Seg.rc2
仮想 7 セグ LED 関連ファイル
mfcSwBox
mfcSwBox.rc
mfcSwBox.sln
mfcSwBox.vcxproj
mfcSwBox.vcxproj.filters
Resource.h
stdafx.cpp
stdafx.h
targetver.h
res
mfcSwBox.ico
mfcSwBox.rc2
仮想 SWBox 関連ファイル
Nortisim
simulator.h …… シミュレータ独自 I/O ヘッダ(7seg,EEPROM 等)
simulator.c …… シミュレータ独自 I/O ソース(7seg,EEPROM 等)
flipflop.c …… タスク動作サンプルソース
keytest.h …… キーパッドサンプルヘッダ
keytest.c …… キーパッドサンプルソース
lcdkey.c …… LCD/キーパッドサンプルソース
main.c …… シリアル動作サンプルソース
n4iwin.c …… インターバル割り込みハンドラ
netsmp.c …… TCP/IP ベースのサンプルプログラムのメイン
nonecfg.c …… TCP/IP ベースサンプルのコンフィグレーション
nonedrvr.c …… LAN ドライバソース
nonethw.c …… LAN ドライバ・環境依存部ソース
nossim.c …… シリアルドライバソース
swbox8.c …… スイッチボックス・サンプルソース


Proj
flipflop.sln
flipflop.vcxproj
flipflop.vcxproj.filters
flipflop.vcxproj.user
タスク切換サンプルプロジェクト
lcdkey.sln
lcdkey.vcxproj
lcdkey.vcxproj.filters
lcdkey.vcxproj.user
LCD/キーパッドサンプルプロジェクト
main.sln
main.vcxproj
main.vcxproj.filters
main.vcxproj.user
シリアル動作サンプルプロジェクト
netsmp.sln
netsmp.vcxproj
netsmp.vcxproj.filters
netsmp.vcxproj.user
ネットワークサンプルプロジェクト
swbox8.sln
swbox8.vcxproj
swbox8.vcxproj.filters
swbox8.vcxproj.user
スイッチボックスサンプルプロジェクト
bind.c ····· カーネル・ドライバインターフェース
dllmain.cpp ····· DLL 本体ソース
nortidll.def ····· DLL モジュール定義ファイル
stdafx.h ····· DLL 共通ヘッダ
Debug
flipflop.dll
lcdkey.dll
netsmp.dll
swbox8.dll
main.dll
コンパイル済みサンプルプログラム

第 2 章 NORTi Simulator の操作

2.1 サンプルプログラムのビルドとデバッグ

この章では例として Windows 10 Pro 64 ビット、並びに Microsoft Visual Studio 2015 Community がインストールされた PC で、SMP¥SIM フォルダのサンプルプログラムをビルド、デバッグする手順を解説します。

なお、netsmp.dll と main.dll は、COM ポートの装備されている PC で実行してください。COM ポートがない場合は、USB シリアル変換ケーブルで USB ポートを仮想 COM ポートとしてください。

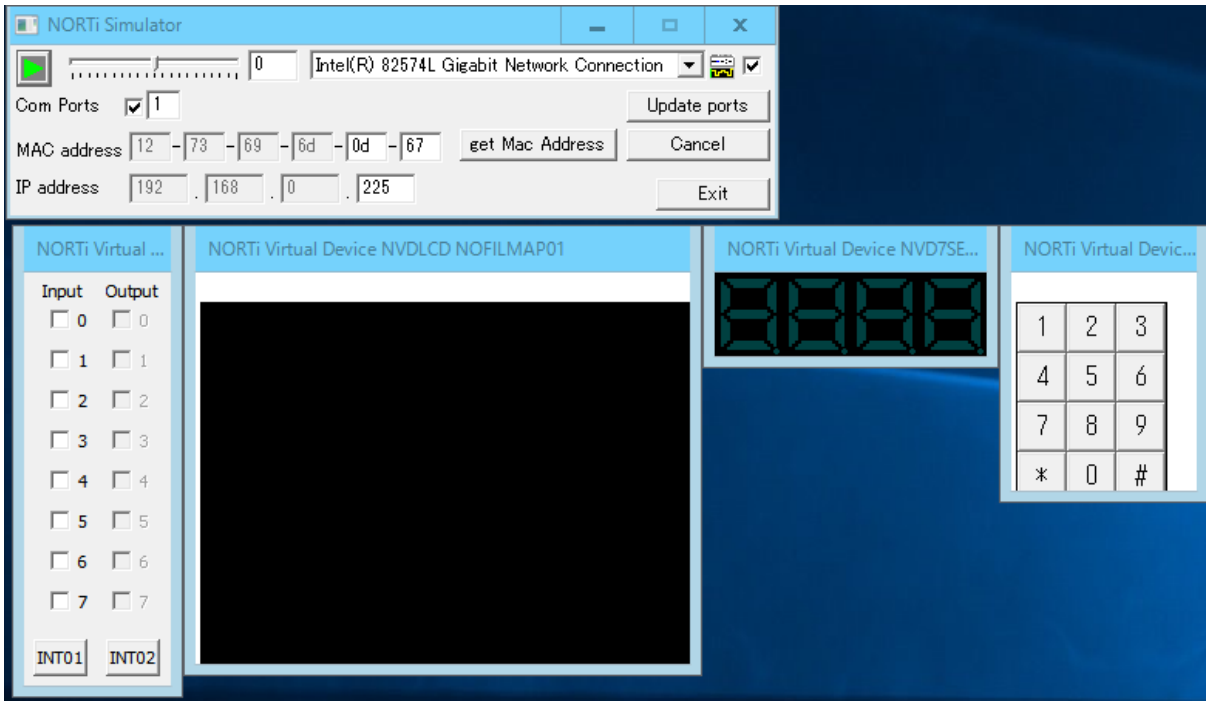
また、Visual Studio を起動せずにサンプルプログラムの動作だけを確認することも可能ですので、その場合には「2.1.2 コンパイル済みサンプルプログラムの実行」をお読みください。

2.1.1 ソリューションを開く

複数のサンプルプログラムが収録されていますが、ここでは lcdkey.dll を例に挙げて説明します。Visual Studio 2015 を起動したら、[ファイル]>[開く]>[プロジェクト/ソリューション]とメニューをたどり、C:¥NORTiSIM¥SMP¥SIM¥nortisim¥proj¥lcdkey.sln を選択して開いてください。次に[デバッグ]>[デバッグの開始]と選択することでコンパイルが行われ、続いて nortisim.exe が起動し、デバッグ実行が開始されます。この時「このプロジェクトは変更されています、ビルドしますか？」というダイアログが現れた場合は「はい」を選んでください。

ここで実行するシミュレーション対象のプログラムは lcdkey.dll という win32 の DLL です。DLL は単独で実行できないため、呼び出し元の EXE ファイルとして、シミュレータの本体である nortisim.exe が、ソリューションのプロパティで指定されています。

次の図は NORTi Simulator が開始され、サンプルプログラムの DLL が正常にロードされた状態を示します。



スタート/ストップボタン(2.2.1 の図を参照)をクリックするとサンプルプログラムのタスクが起動します。初期化が正常に完了すると、7 セグ LED に数字が表示され、増加し始めます。

2.1.2 コンパイル済みサンプルプログラムの実行

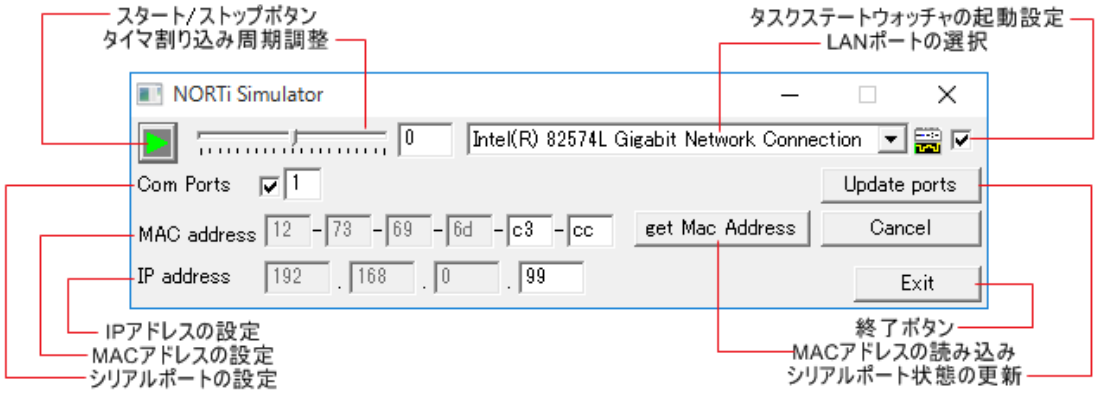
Visual Studio 2015 がインストールされている必要がありますが、その起動とビルドの操作を省いて、まずサンプルプログラムの動作を確認することができます。

シミュレータ本体である C:¥NORTiSIM¥BIN¥nortisim.exe を起動すると、DLL を選ぶダイアログが表示されますので、C:¥NORTiSIM¥SMP¥SIM¥NORTiSIM¥PROJ¥Debug フォルダにある、例えば lcdkey.dll を選び、スタート/ストップボタン(2.2.1 の図を参照)を押すと、サンプルプログラムが動作を開始します。

2.2 操作

2.2.1 コントロールパネル各部の名称

nortisim.exe が起動すると次の図に示すコントロールパネルが開きます。



2.2.2 スタート/ストップボタン

このボタンをクリックすることにより、プログラムが起動します。再度クリックするとプログラムが停止します。

2.2.3 タイマ割り込み周期調整

スライダを動かして周期タイマ割り込みの周期を変更できます。スライダの右側に、数値を表示しています。 この数値とタイマ割り込み周期の関係は以下の通りです。(MSEC=10 ミリ秒)

数値	周期計算式	割り込み周期	時間経過
3	MSEC × (3 + 1)	4 倍	遅い
2	MSEC × (2 + 1)	3 倍	↑
1	MSEC × (1 + 1)	2 倍	↑
0	MSEC × (0 + 1)	等倍	通常
-1	MSEC ÷ (1 + 1)	1/2 倍	↓
-2	MSEC ÷ (2 + 1)	1/3 倍	↓
-3	MSEC ÷ (3 + 1)	1/4 倍	早い

割り込みの周期を変化させると、システムクロックに依存する部分の時間の流れがあわせて変化します。

2.2.4 タスクステートウォッチャの起動設定

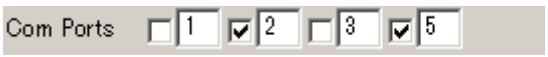
このチェックボックスをチェックして、スタート/ストップボタンをクリックし、プログラムを起動すると、タスクステートウォッチャが起動します。
チェックを外してスタートした場合は、タスクステートウォッチャは起動しません。チェックボックスの状態は Windows のレジストリに保持されるので、次回起動時からは同じ設定で使用できます。

2.2.5 LAN ポートの選択

TCP/IP プロトコルスタックのシミュレーションを行う場合に使用する PC の LAN ポートを、ここで選択できます。
ここでは、NORTi Simulator が起動したときに Windows のデフォルトゲートウェイで設定されている LAN ポートが自動で選択されます。
もし、他の LAN ポートを使用する時には手動で選択してください。
TCP/IP プロトコルスタックを使用しない場合は、何が選択されていても構いません。

2.2.6 シリアルポートの設定

NORTi Simulator の起動時に COM1 から COM8 の間で使えるシリアルポートの番号がここに表示されます。チェックボックスが付いているポート番号が左から、論理チャネル 0 に該当します。
例えば、次のようになっている時には、チャネル 0 は COM2、チャネル 1 は COM5 を使う事になります。



使えるシリアルポートが無い場合は空欄となっていて、プログラムでシリアルポートの操作を行うと、ランタイムエラーとなってプログラムが中断されてしまいますのでご注意ください。
他のプロセスでシリアルポートがすでに使用中であると、ここに表示されなくなります。
そのシリアルポートを NORTi Simulator で使いたい時には、シリアルポートを使用しているプロセスを終了してから、「Update ports」ボタンを押すと、ポート番号が表示されるので、NORTi Simulator で使える状態になります。

2.2.7 IP アドレスの設定

IP アドレスが 4 つのボックスに表示されています。一番右を除いて選択した LAN ポートの IP アドレスが表示されています。一番右は自由に設定してください。
ユーザープログラムが eep_get 関数でアドレスを取得する場合は、ネットワークサンプルはこのボックスの値を IP アドレスとして動作します。DHCP を使用した場合や、eep_get 関数を使用せずに直接アドレスをコードで設定した場合は、IP アドレスが設定された時点で表示が更新されます。この値はレジストリに保存され、次回起動時に現在の設定値が適用されます。

2.2.8 MAC アドレスの設定

ここには、NORTi Simulator のデフォルトの MAC アドレスが表示されます。
"Get MAC address"ボタンを押すと、選択されている LAN ポートの MAC アドレスの後 2 バイトが、一番右の 2 個のボックスに表示されます。任意の 16 進数 2 桁の値に変更しても構いません。
また、「Cancel」ボタンをクリックした場合、以前レジストリに保存した値が存在するならば、それが表示されます。
ユーザープログラムが eep_get 関数で MAC アドレスを取得する場合は、ネットワークサンプルはこのボックスの値を MAC アドレスとして動作します。 eep_get 関数を使用せずに直接 MAC アドレスをコードで設定した場合は、プログラム内で MAC アドレスが設定された時点で表示が更新されます。
この値は、レジストリに保存され、次回起動時は現在の設定値が使用できます。

2.2.9 終了ボタン

このボタンを押すと NORTi Simulator が終了します。
ウィンドウの[×]ボタンと同じ機能です。

2.2.10 仮想デバイス GUI モジュール

仮想デバイス GUI モジュールは独立したウィンドウ応用プログラムで、別プロセスで動作します。NORTi Simulator のアプリケーションが自動で立ち上げ、コントロールパネルが終了するときに一緒に終了させています。
例外の発生等で終了した時には、GUI モジュールが動作たまま残るので、手動で終了させてください。
サンプルの GUI モジュールは Alt-F4 キーで終了できます。

2.3 タスクステートウォッチャ

タスクステートウォッチャでは以下の情報を表示することが可能です。

μITRON3.0 仕様，μITRON4.0 仕様共通	μITRON4.0 仕様のみ
• タスク • セマフォ • イベントフラグ • メールボックス • メッセージバッファ • 固定長メモリプール • 可変長メモリプール • ランデブ用ポート • アラームハンドラ • 周期起動ハンドラ • レディキュー	• ミューテックス • データキュー • オーバーランハンドラ • タスク例外ルーチン • 割り込みサービスルーチン • サービスコールルーチン

2.3.1 起動・終了

NORTi Simulator コントロールパネルによりシミュレーションをスタートするときに、タスクステートウォッチャの起動設定がチェックされていると、タスクステートウォッチャが起動します。

NORTi Simulator コントロールパネルによりシミュレーションを終了しても、タスクステートウォッチャは、終了しません。デバック終了時にタスクステートウォッチャを閉じてください。

2.3.2 更新

自動でステータス情報を更新させる場合には「機能」メニューの「ステータス自動更新」を有効にしてください。

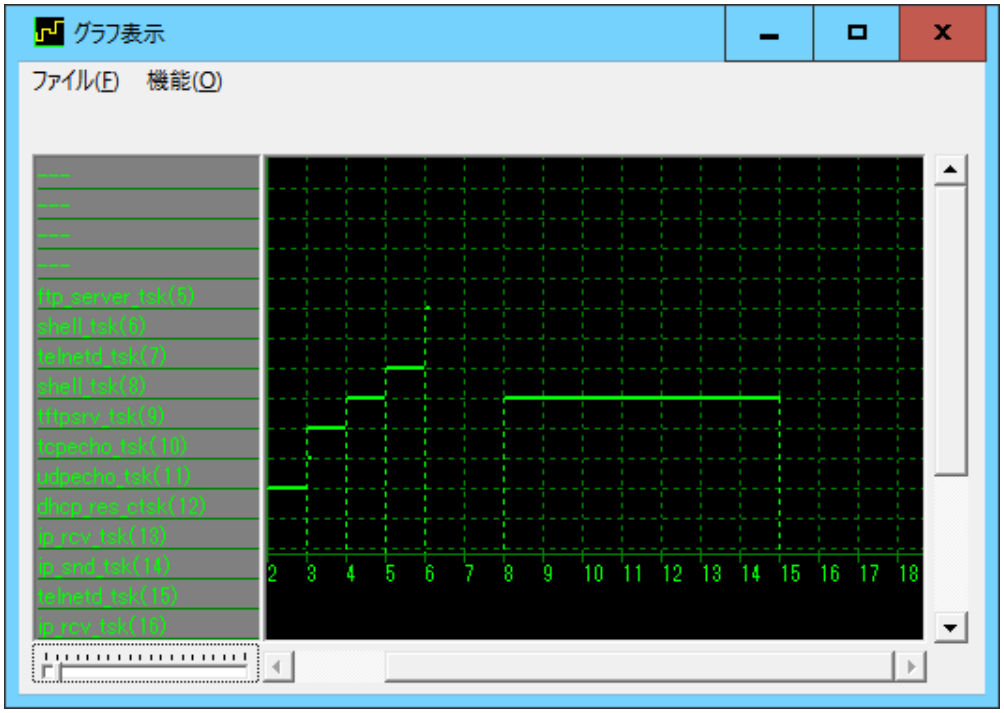
リスト上に表示されたステータス情報の更新はタスクが切り替わった時に行われます。

シミュレーション実行中は、「更新」ボタンが使用できませんのでデバッグ中に更新させる場合にはタブを切り替えて更新してください。

2.3.3 グラフ表示

本機能により、時間軸とタスク ID 軸のタスク状態のグラフを表示することが可能です。

「グラフ表示」ボタンを押すとグラフが表示されます。



また、グラフデータ一覧機能では更に詳しい情報をリストで調査する事ができます。

グラフ表示の「機能」メニューの「グラフデータ一覧」命令で次のようにグラフデータ一覧が表示されます。

データ番号	タスクID	タスク名	待ち要因	開始時間	終了時間	時間	トータル時間
17	18	MainTask	TSK	4967.000000	4975.000000	8.000000	325.000000
18	17	ip_snd_tsk	MBX	4975.000000	5010.000000	35.000000	333.000000
19	16	ip_rcv_tsk	SLP	5010.000000	5042.000000	32.000000	368.000000
20	14	ip_snd_tsk	MBX	5042.000000	5042.000000	0.000000	400.000000
21	16	ip_rcv_tsk	SLP	5042.000000	5043.000000	1.000000	400.000000
22	18	MainTask	TSK	5043.000000	5076.000000	33.000000	401.000000
23	16	ip_rcv_tsk	SLP	5076.000000	5078.000000	2.000000	434.000000
24	17	ip_snd_tsk	MBX	5078.000000	5110.000000	32.000000	436.000000
25	14	ip_snd_tsk	MBX	5110.000000	5143.000000	33.000000	468.000000
26	18	MainTask	TSK	5143.000000	5176.000000	33.000000	501.000000
27	17	ip_snd_tsk	MBX	5176.000000	5210.000000	34.000000	534.000000
28	16	ip_rcv_tsk	SLP	5210.000000	5242.000000	32.000000	568.000000
29	14	ip_snd_tsk	MBX	5242.000000	5244.000000	2.000000	600.000000
30	18	MainTask	TSK	5244.000000	5276.000000	32.000000	602.000000
31	16	ip_rcv_tsk	SLP	5276.000000	5303.000000	27.000000	634.000000

「ログ出力...」ボタンでテキストファイルに保存することができます。

2.3.4 ヘルプ

「ヘルプ」-「バージョン表示」命令でバージョン情報が表示されます。

2.3.5 各タブの説明

2.3.5.1 タスクタブ

タスク ID	タスクの ID 番号です。
タスク名	タスク生成時に指定された関数の名前です。タスクが生成されていない場合は表示されません。
タスク状態	タスクの状態が表示されます。未登録(タスクが未生成、または ID 範囲外)の場合は空欄となります。
優先度	タスクの現在の優先度が 10 進数で表示されます。 未登録と DORMANT 状態のタスクでは空欄となります。
起床要求カウント	起床要求キューイング数が 10 進数で表示されます。 未登録と DORMANT 状態のタスクでは空欄となります。
強制待ち要求カウント	強制待ち要求ネスト数が 10 進数で表示されます。 未登録と DORMANT 状態のタスクでは空欄となります。
待ち要因	タスクが WAIT 状態の場合、何を待っているかが次の略号で表示さ

	れます。 タスクが WAIT 状態でない場合は空欄となります。 • SLP …… 起床待ち • TSK …… 時間待ち • POR …… ランデブ終了待ち • FLG …… イベントフラグ成立待ち • SMB …… メッセージバッファでの送信待ち • CAL …… ランデブ呼び出し待ち • ACP …… ランデブ受け付け待ち • SEM …… セマフォ獲得待ち • MBX …… メールボックスメッセージ受信待ち • MBF …… メッセージバッファでの受信待ち • MPL …… 可変長メモリブロック獲得待ち • MPF …… 固定長メモリブロック獲得待ち • SDT …… データキュー送信待ち • RDT …… データキュー受信待ち • MTX …… ミューテックス資源獲得待ち

2.3.5.2 セマフォタブ

セマフォ ID	セマフォの ID 番号です。
セマフォカウント	現在のセマフォカウント値が 10 進数で表示されます。
待ちタスク	このセマフォを待っている先頭タスクの名前と ID 番号が表示されます。 ID 番号は 10 進数で表示されます。 セマフォ ID 範囲外や待ちタスクがない場合は空欄となります。 複数の待ちタスクがある場合はその並びが表示されます。 セマフォカウントが 1 以上の場合、待ちタスクはあり得ません。

2.3.5.3 イベントフラグタブ

イベントフラグ ID	イベントフラグの ID 番号です。
フラグパターン	現在のイベントフラグ値が 16 進数で表示されます。 イベントフラグ ID 範囲外の場合は空欄となります。
待ちタスク	このイベントフラグの成立を待っている先頭タスクの名前と ID 番号が表示されます。 ID 番号は 10 進数で表示されます。 イベントフラグ ID 範囲外や待ちタスクがない場合は空欄となります。 複数の待ちタスクがある場合はその並びが表示されます。

2.3.5.4 メールボックスタブ

メールボックス ID	メールボックスの ID 番号です。
メッセージアドレス	メールボックスの先頭にあるメッセージのアドレスが 16 進数で表示されます。 メールボックス ID 範囲外やメッセージがない場合は空欄となります。
待ちタスク	このメールボックスでメッセージ受信を待っている先頭タスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 メールボックス ID 範囲外や待ちタスクがない場合は空欄となります。 複数の待ちタスクがある場合はその並びが表示されます。

2.3.5.5 メッセージバッファタブ

メッセージバッファ ID	メッセージバッファの ID 番号です。
メッセージ	メッセージバッファの先頭にあるメッセージの内容が先頭から最大 4 バイト 16 進数で表示されます。 メッセージバッファ ID 範囲外やメッセージがない場合は空欄となります。 バッファの先頭に入っているメッセージの長さを表わすヘッダのサイズがデフォルトの値 4 バイトと異なる場合、正しくメッセージデータを表示することができません。この場合にはオプションダイアログで正しいヘッダサイズを設定してください。
送信待ちタスク	送信待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 送信待ちタスクがない場合は空欄となります。 複数の送信待ちタスクがある場合はその並びが表示されます。
受信待ちタスク	受信待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 受信待ちタスクがない場合は空欄となります。 複数の送信待ちタスクがある場合はその並びが表示されます。

2.3.5.6 固定長メモリプールタブ

メモリプール ID	メモリプールの ID 番号です。
空きメモリブロック数	メモリプールにある空きメモリブロックの個数が 10 進数で表示されます。
送信待ちタスク	送信待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 送信待ちタスクがない場合は空欄となります。 複数の送信待ちタスクがある場合はその並びが表示されます。

2.3.5.7 可変長メモリプールタブ

メモリプール ID	メモリプールの ID 番号です。
総空きメモリサイズ	メモリプールにある空きメモリの総サイズが 10 進数で表示されます。
最大連続空きメモリサイズ	メモリプールにある最大の連続空きメモリサイズが 10 進数で表示されます。
待ちタスク	このメモリプールでメモリ獲得を待っている先頭タスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 メモリプール ID範囲外や待ちタスクがない場合は空欄となります。 複数の待ちタスクがある場合はその並びが表示されます。

2.3.5.8 ランデブ用ポートタブ

ランデブ用ポート ID	ランデブ用ポートの ID 番号です。
呼び出し待ちタスク	現在のセマフォカウント値が 10 進数で表示されます。
待ちタスク	呼び出し側の待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 呼び出し待ちタスクがない場合は空欄となります。 複数の呼び出し待ちタスクがある場合はその並びが表示されます。
受け付け待ちタスク	受け付け側の待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 受け付け待ちタスクがない場合は空欄となります。 複数の受け付け待ちタスクがある場合はその並びが表示されます。

2.3.5.9 アラームハンドラタブ

ハンドラ番号	アラームハンドラの番号です。
起動時間	ハンドラ起動時間(シミュレーション上での時間)が、上位：下位の形式の 16 進数で表示されます。 ハンドラ番号範囲外やアラームハンドラが定義されていない場合は空欄となります。

2.3.5.10 周期起動ハンドラタブ

ハンドラ番号	周期起動ハンドラの番号です。
活性状態	周期起動ハンドラの活性状態が「ON」または「OFF」で表示されます。 ハンドラ番号範囲外や周期起動ハンドラが定義されていない場合は空欄となります。
起動時間	次の起動時間(シミュレーション上での時間)が、上位：下位の形式の 16 進数で表示されます。 ハンドラ番号範囲外や周期起動ハンドラが定義されていない場合は空欄となります。

2.3.5.11 レディキュータブ

優先度	レディキューの各優先度です。 優先度 1 はレディーキューテーブルのトップ(RDQ[0])を指します。
実行待ちタスク	この優先度で実行待っているタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 優先度範囲外や実行待ちタスクがない場合は空欄となります。 複数の実行待ちタスクがある場合はその並びが表示されます。

2.3.5.12 ミューテックスタブ

ミューテックス ID	ミューテックス ID 番号が表示されます。
ロック待ちタスク	ロック待ちのタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 優先度範囲外や実行待ちタスクがない場合は空欄となります。 複数の実行待ちタスクがある場合はその並びが表示されます。
ロックしているタスク	ロックしているタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 優先度範囲外や実行待ちタスクがない場合は空欄となります。 複数の実行待ちタスクがある場合はその並びが表示されます。

2.3.5.13 データキュータブ

データキューID	データキューの ID 番号が表示されます。
データ数	データ数が表示されます。
受信待ちタスク	受信待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 優先度範囲外や実行待ちタスクがない場合は空欄となります。 複数の実行待ちタスクがある場合はその並びが表示されます。
送信待ちタスク	送信待ちタスクの名前と ID 番号が表示されます。ID 番号は 10 進数で表示されます。 優先度範囲外や実行待ちタスクがない場合は空欄となります。 複数の実行待ちタスクがある場合はその並びが表示されます。

2.3.5.14 オーバーランハンドラタブ

タスク ID	オーバーランハンドラが設定されているタスク IDが表示されます。
活性状態	オーバーランハンドラの状態が表示されます。 動作中 ・・・ オーバーランハンドラ動作中 停止中 ・・・ オーバーランハンドラ起動中
残り時間	オーバーランハンドラ起動までの残り時間が表示されます。

2.3.5.15 タスク例外ルーチンタブ

タスク ID	タスク例外ルーチンが設定されているタスク ID が表示されます。
ステータス	ステータスが表示されます。 動作中 ・・・ 例外ルーチン動作中 停止中 ・・・ 例外ルーチン起動中
例外要因	例外要因が表示されます。

2.3.5.16 割り込みサービスルーチンタブ

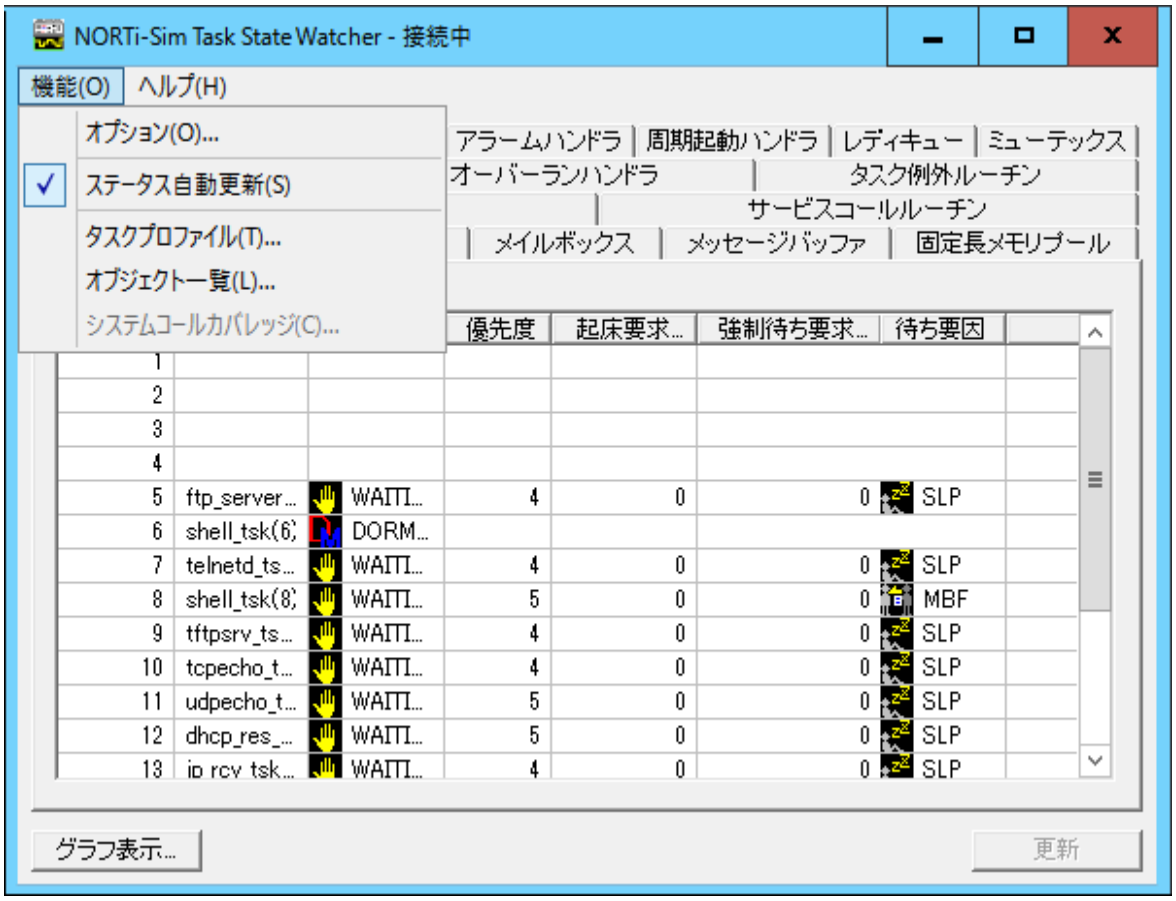
割り込みサービスルーチンID	割り込みサービスルーチン ID 番号が表示されます。
割り込み番号	設定されている割り込み番号が表示されます。
関数(ルーチン)	関数のアドレスが表示されます。

2.3.5.17 サービスコールルーチンタブ

サービスコールルーチン ID	サービスコールルーチン ID 番号が表示されます。
パラメータ数	サービスコールルーチンのパラメータ数が表示されます。
関数(ルーチン)	サービスコール関数のアドレスが表示されます。

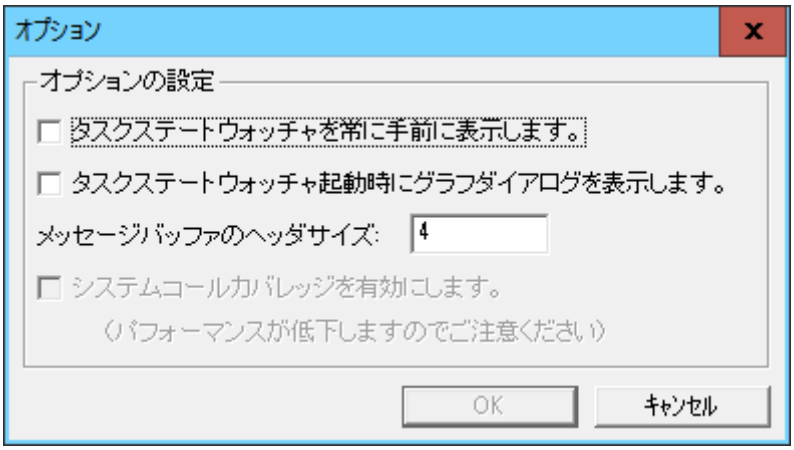
2.3.6 機能

「機能」メニューで各設定項目が表示されます。



2.3.6.1 オプション

タスクステートウォッチャの補助的な設定をおこないます。オプションダイアログは、[機能]メニューの[オプション…]から起動することができます。



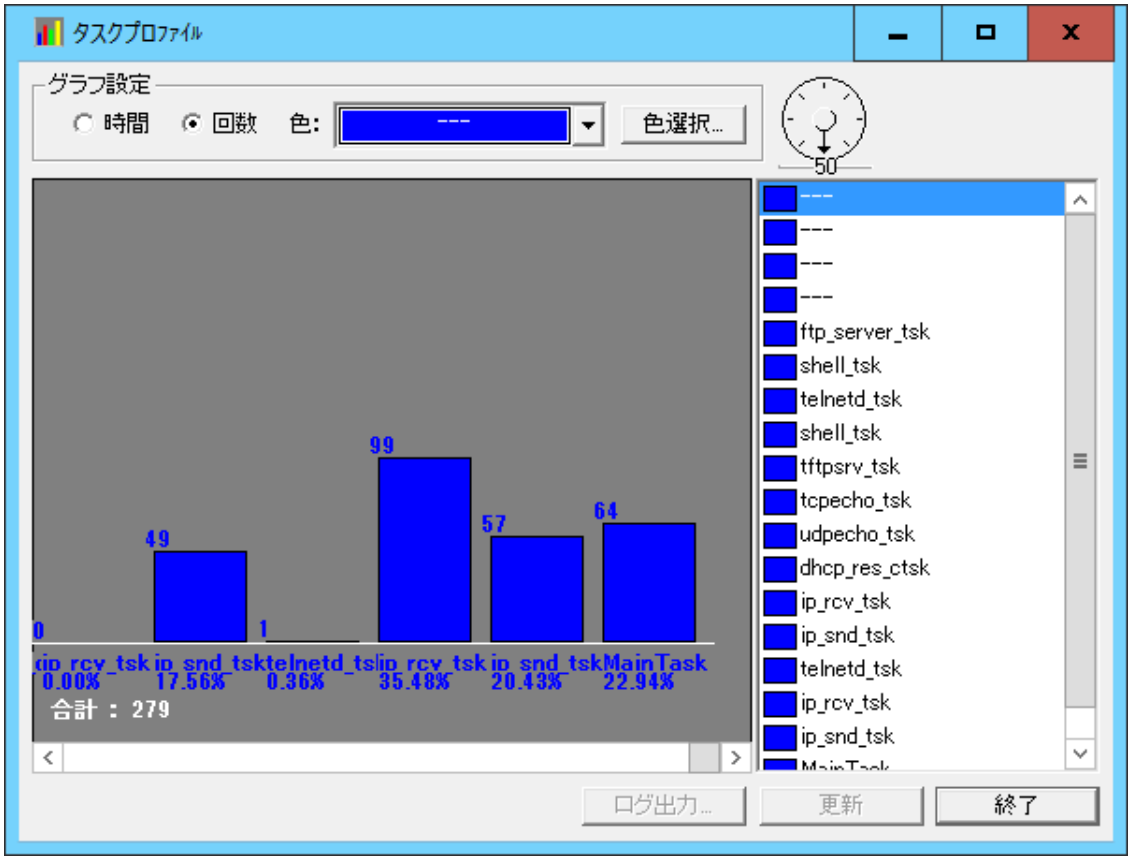
タスクステートウォッチャを常に手前に表示します。	この項目を設定するとタスクステートウォッチャのウィンドウがほかのウィンドウに覆われることなく常に最上位に表示されるようになります。
タスクステートウォッチャ起動時にグラフダイアログを表示します。	設定しない場合にはテキストベースの表形式の表示となります。
メッセージバッファのヘッダサイズ	メッセージバッファの先頭に設定されるメッセージの長さが何バイトで設定されているかを指定します。

2.3.6.2 ステータスの自動更新

[ステータスの自動更新]をクリックするとチェックマークが表示されステータスが自動更新されます。再度クリックするとチェックマークが消え自動更新が停止します。
ステータスの自動更新は[機能]メニューで選択できます。

2.3.6.3 タスクプロファイル

タスクの実行時間・実行回数をタスクごとにプロファイリングし、ダイアログにグラフで表示します。タスクプロファイルダイアログはタスクステートウォッチャの[機能]メニューから[タスクプロファイル]を選択して起動することができます。



時間	タスク実行時間のプロファイルを表示します。
回数	タスク実行回数のプロファイルを表示します。
色	各タスクのグラフ表示色を設定します。 プルダウンメニューでタスクを選択し色選択ボタンで設定します。
終了	タスクプロファイルを終了します。
グラフエリア	タスクのプロファイル情報を表示します。
タスク名リスト	タスク名の一覧をリスト表示します。 タスク名を選択すると選択されたタスクのグラフがグラフエリア内に表示されます。 全タスクがグラフエリア内に表示されている場合には機能しません。
ダイヤル	各タスクの棒グラフ幅を調節します。

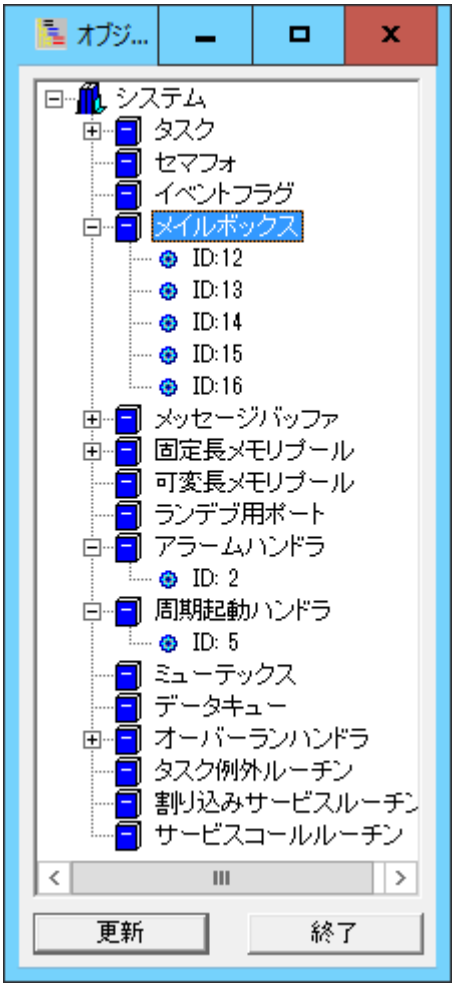
※　ディスパッチの状態によってはタスク実行時間(シミュレータエンジン内のカウンタ)が０の場合があります。この場合、タスクプロファイルではデータとして記録されませんので内部的に１が設定されます。よって、タスクステートウォッチャのタスク遷移グラフ上との時間に誤差が生じる場合がありますのでご了承ください。

※　ダイアログを表示させた状態で実行を行いますとパフォーマンスが落ちますのでご注意ください。

※　システムの途中で一旦削除されたタスクのプロファイルデータは保存されていますが、次に同じ ID で別タスクが起動した場合、以前のプロファイルデータに加算されます。削除されたタスクの名前は表示されません。

2.3.6.4 オブジェクト一覧

オブジェクトが一覧表示されます。選択した情報がタスクステータスダイアログに表示されます。（タスクステータスダイアログと同じ動きをします。）



第 3 章 仮想デバイス

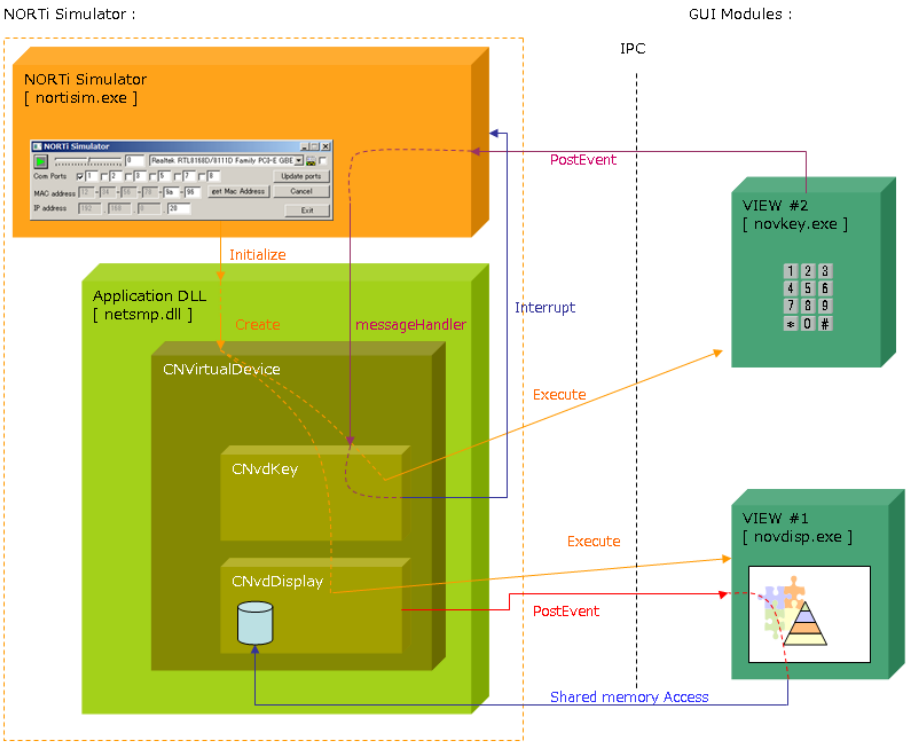
3.1 概要

NORTi Simulator には仮想デバイス機能があり、サンプルとしてスイッチボックスを表示する仮想デバイス、7 セグメント LED の仮想デバイス、仮想 LCD、仮想キーパッドなどを添付しています。

仮想デバイスインターフェースクラスを拡張することで、新規仮想デバイスの追加が可能です。

ここでは、仮想デバイスの構造、新規デバイスの追加に関して解説します。

3.2 構成



仮想デバイスは、SimDeviceDesc、インターフェース(CNVirtualDevice)と GUI モジュールで構成されます。

SimDeviceDesc はデバイスの情報を記述します。

インターフェースは、クラスライブラリと C ラッパー関数で構成されており、NORTi Simulator アプリケーションとのインターフェースを担当します。

GUI モジュールは独立した exe で、NORTi Simulator の仮想デバイスインターフェースと Win32

IPC(Interprocess communication)で通信して、デバイスの見た目、ユーザー入力を担当します。
以上の各要素で、仮想デバイスがシミュレーションする基本的な機能は次の通りです。

装置メモリ
UI 表示
UI 入力
イベント発生
イベント通知

3.2.1 SimDeviceDesc

仮想デバイスの情報を格納するための構造体として、SimDeviceDesc を宣言しています。

```
typedef struct _SimDeviceDesc {
    UINT cls;               クラス ID
    UINT id;                デバイス ID
    UINT irq;               IRQ 番号
    UINT pri;               割り込み優先度
    UINT vadd;              仮想アドレス
    UINT range;             仮想アドレスの範囲(長さ)
    UINT misc;              付加情報
    const char *exename;    GUI モジュールの実行ファイル名
    const char *cmdarg;     GUI モジュールのコマンドライン引数
} SimDeviceDesc;
```

3.2.1.1 クラス ID

デバイスの種類を特定するために、属性と番号からなるクラス ID を使います。
クラス ID 関連のマクロは「nosimapi.h」で定義されています。
クラス番号と主番号でインターフェースの Class 型が特定できます。

属性(16 ビット)	主番号(8 ビット)	副番号(8 ビット)

属性は次の各属性をビット OR 演算して重複指定できます。

VDEVATTR_EXT　　: 拡張デバイス、内蔵デバイス以外
VDEVATTR_GUI　　: GUI モジュール使用
VDEVATTR_INPUT　: 入力機能使用
VDEVATTR_OUTPUT : 出力機能使用
VDEVATTR_INT　　: 割り込み使用

主番号はクラス属性に対し、0～255 の番号が指定できます。

副番号は 0～255 の番号が指定できます。
クラス IDはフィールドごとに意味を持ちますが、全体として 32 ビットの 1 つの IDになります。

3.2.1.2 id

同じクラス IDであっても、コマンドライン引数などが異なる場合の区別をするのに使用します。

3.2.1.3 irq、pri

irq は割り込み番号です。0～31 のいずれかが使用できます。
pri は割り込み優先度です。0～31 のいずれかが使用できます。

3.2.1.4 vadd、range

vadd は仮想アドレス番号です。sim_output, sim_input 関数でアクセスするためのアドレスです。
SIMADDR_USER 未満の番地は内蔵デバイス用に予約されているので、ユーザープログラムでは SIMADDR_USER 以上の番地を使ってください。
仮想アドレスを通したアクセスをしない場合は、0 を使ってください。
range は仮想アドレスの長さ(バイト単位)です。
仮想アドレスを通したアクセスをしない場合は、0 を使ってください。

3.2.1.5 misc

misc は各インターフェース固有の追加情報です。
インターフェースクラスを作成する時に、自由に使ってください。

3.2.1.6 exename、cmdarg

exename は GUI モジュールの実行ファイル名です。NORTi Simulator の作業ディレクトリは、NORTiSIM/BIN ですので、相対パスか絶対パスで記述してください。
cmdarg は GUI モジュールを実行する時のコマンドライン引数です。インターフェースのなかでも参照できますが、GUI モジュールに伝えるのが本来の目的です。
GUI モジュールと同じ情報を維持する必要がある場合は、インターフェース内部で、この引数を使うのを推薦します。インターフェース内部では、初期化のあと、argc と argv のメンバー変数で参照できます。
exename と cmdarg を変えることで既存のインターフェースを使って新しいデバイスを生成することができます。

3.2.1.7 例示

サンプルのキーパッドの場合、次のようにSimDeviceDescを定義しています。

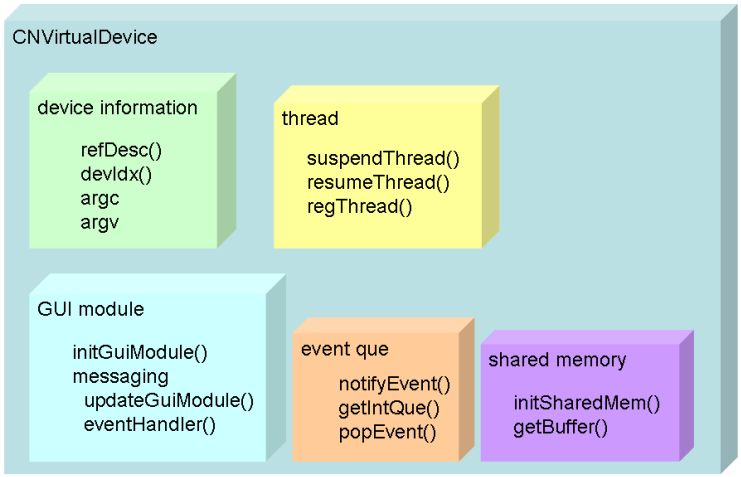
```
const SimDeviceDesc dev_keypad = {
    VDEVCLASS_KEYPAD, 0,
    28, 8,
    0, 0,
    0,
    "nvdkey.exe",
    "3 4 48 2 1,2,3,4,5,6,7,8,9,*,0,#"
};
```

3.2.2 CNVirtualDevice

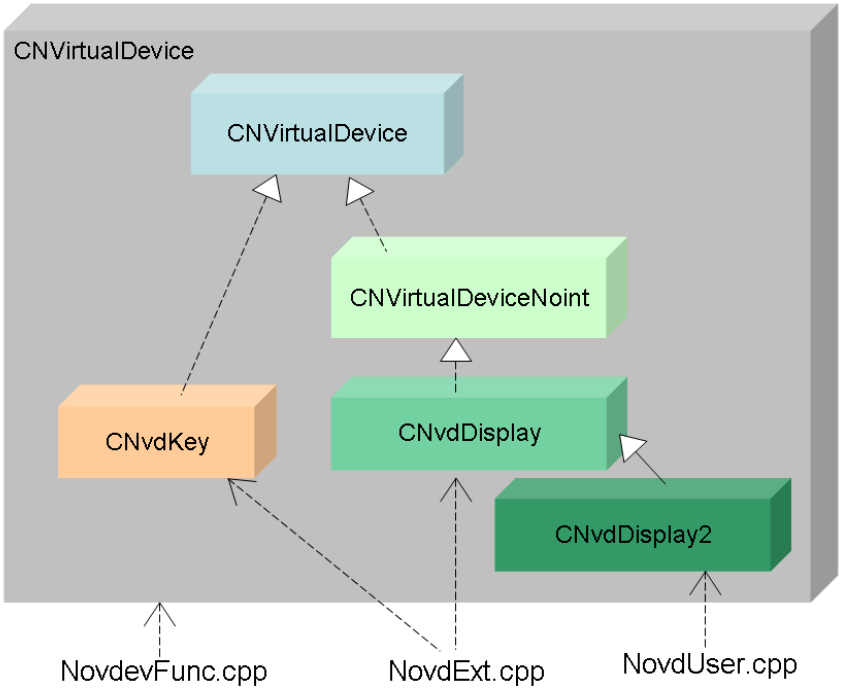
CNVirtualDeviceは、インターフェースのC++クラスです。
クラスの継承を通じてコードを再使用するために、クラスで製作していますが、C コードではクラスメンバー関数を呼び出しできないので、ラッパー関数を提供しています。
CNVirtualDeviceは、「novdev.h」に定義されています。
ラッパー関数は、「novdevfunc.cpp」で実装しています。
インターフェースは、ライブラリとして提供しますが、継承した派生クラスはサンプルソースファイルとして提供しています。
派生クラスも同様に、ラッパー関数が必要ですが、SMP/SIM/common/novdext.cppのようにソースでビルドするようになっています。
仮想デバイスを追加する場合にも、別のソースを作成して実装してください。

3.2.2.1 機能図

仮想デバイスインターフェースクラスの機能は、次のように区分して説明します。



3.2.2.2 階層図



CNVirtualDevice は抽象クラスなので、インターフェースとして、継承して実装しなければなりません。

NORTi Simulator では、サンプルとして、CNvdDisplay、CNvdKey クラスを提供します。
さらにサンプルのクラスを継承して、新しいユーザーデバイスを作成することもできます。

3.2.2.3 コンストラクタ・デストラクタ

CNVirtualDevice(const SimDeviceDesc *desc, UINT clsid);
virtual ~CNVirtualDevice(void);
コンストラクタには、

- メンバー変数の初期化
- SimDeviceDesc の保持
- SimDeviceDesc の cls を検証
- argc, argv の解析
- gpDev[],devCnt の処理 (3.2.2.4 参照)

等の機能があります。
コンストラクタでは SimDeviceDesc の cls 情報と、クラスメンバーのクラス ID の一致検査を行います。実装クラスのコンストラクタの引数でも同じく自分のクラス ID をデフォルト引数に入れてください。不一致の場合には例外処理をします。
デストラクタでは、メンバー変数の終了処理、GUI モジュールの終了処理などを行います。

3.2.2.4 デバイス情報関連

static CNVirtualDevice *gpDev[MAXVDEV]
デバイスのインスタンスを保持する配列

static int devCnt
デバイスが生成されるたびにカウントアップされる変数。生成されたデバイスは、gpDev[0]から gpDev[devCnt-1]で参照できる

int devIdx(void)
デバイスが生成された時の devCnt の値(増加前)を得る
すなわち gpDev[devIdx()]は this を参照することになる
C 関数側では、生成時にこの値を保存することで、デバイスを参照することができる

SimDeviceDesc *refDesc(void)
生成時の SimDeviceDesc のコピーへのポインタ

UINT classid, id, irq, pri
生成時の SimDeviceDesc の各情報のコピー

char *argv[16]、int argc

生成時の SimDeviceDesc の GUI モジュールの実行情報

3.2.2.5 GUI モジュール、メッセージ関連

protected: int initGuiModule(void)

SimDeviceDesc 情報を元に生成されたインスタンスを初期化する

成功時 0 をリターン

protected: int updateGuiModule(void)

GUI モジュールに WM_PAINT メッセージを送る

成功時 0 をリターン

LRESULT messageHandler(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam) = 0

純粋仮想関数

GUI モジュールからのメッセージを処理する

Win32 の標準の PostMessage と SendMessage を処理するので、その時の引数がそのまま渡される

HWND hwnd ： NORTi Simulator コントロールパネルのウィンドウハンドル

UINT msg ： GUI モジュールからのメッセージ

WPARAM wparam ： メッセージの WPARAM

LPARAM lparam ： メッセージの LPARAM

このハンドラは、NORTi Simulator のユーザープログラムのタスクとは別のスレッドで動くため、ユーザープログラム側のリソースを操作する場合は注意が必要

protected: HWND getHwnd(void)

GUI モジュールのウィンドウハンドル

共有メモリに接続している GUI モジュールのウィンドウハンドルを獲得する

protected: TCHAR filename[1024]

GUI モジュールの実行ファイル名

protected: TCHAR cmdarg[2048]

GUI モジュールの実行引数

protected: TCHAR guiTitle[128]

GUI モジュールのタイトル

getHwind()でウィンドウハンドルを得てこのタイトルを適用する

int activateGuiModule(void)

GUI モジュールをアクティブ化する

NORTi Simulator コントロールパネルがアクティブウィンドウになった時に NORTi Simulator コントロールパネルから simvdev_control()を通じて呼ばれる

3.2.2.6 共有メモリ関連

protected: char smName[128]

共有メモリの名前

コンストラクタで設定される

protected: int initSharedMem(UINT pagesize, UINT npages)

pagesize 分のページを npages 個分生成する

UINT pagesize : ページの長さ(バイト)

UINT npages : ページ個数

成功時 0 をリターン

pagesize は内部で、4 バイトアライン(拡大)で整列される

unsigned char *getBuffer(int idx)

ページバッファのポインタ

int idx : ページ番号

protected: SharedmemHeader *smHdr

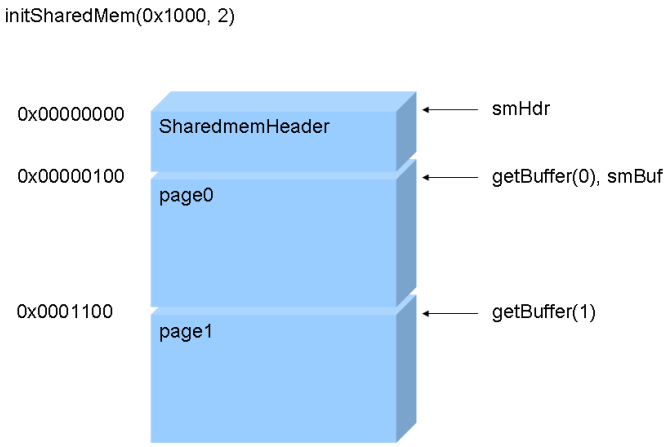
共有メモリヘッダへのポインタ

protected: void *smBuf

共有メモリのページバッファへのポインタ

protected: HANDLE hMapFile

共有メモリのハンドル



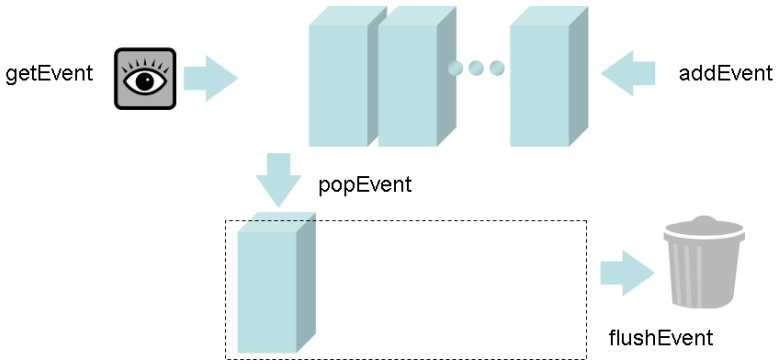
3.2.2.7 イベント関連

NORTi Simulator では、Windows のメッセージなどを使って、スレッドから仮想割り込みを発生させます。

スレッドセーフではない方法でデバイスと NORTi Simulator 間データ交換を行うと予期しない不具合の原因になりますので注意してください。

割り込みの処理中は、スレッド関連の問題を予防するために、専用のイベントキューを使ってください。

ここではイベントキューに関して説明します。



protected: void addEvent(void *e) = 0

イベントを追加

protected: void flushEvent(void) = 0

削除予約されたイベントを削除する

void *getEvent(void) = 0

イベントを参照

void popEvent(void) = 0

イベントを取り出して削除予約

以上の関数はすべて純粋仮想関数ですので、実装クラスでの実装が必要です。
実装は

```
CNvdPtrQue eventQue;
virtual void popEvent(void) {eventQue.pop();}
virtual void *getEvent(void) {return eventQue.front();}
virtual void addEvent(void *e) {if (e) eventQue.push(e);}
virtual void flushEvent(void) {eventQue.flush();}
```

のようにしてください。
CNvdPtrQue クラスは「novdque.h」に定義しています。
CnvdPtrQue 型のオブジェクトに addEvent でイベントを登録するときには、引数に汎用ポインタ void * を取っていますので、任意の型のデータを登録できます。なお、イベントを削除する際に内部で delete を行うので、破棄されてもよいデータを渡してください。
固定長のイベントを登録する時は、イベントをコピーして登録することもできます。

```
CNvdQue<int> eventQue;
virtual void popEvent(void) {eventQue.pop();}
virtual void *getEvent(void) {return eventQue.front();}
virtual void addEvent(void *e) {if (e) eventQue.push(e);}
virtual void flushEvent(void) {eventQue.flush();}
```

CNvdQue クラスに、C++テンプレートで使う型を指定すると、使うデータ型を自由に選ぶことができます。これを使う時には、オート変数のポインタでも登録することもできます。用途によって、選択して実装してください。
他にも、popEvent()などの関数を、一から実装することもできますが、そのときには、スレッドセーフになるよう注意してください。特に、getEvent()と popEvent()は NORTi Simulator のタスクで使う関数なので、この中で Win32API を使うと不具合の原因になります。
提供している CNvdQue クラスでは、popEvent()の中では、キューからすぐ削除せずに削除予約だけしておいて、flushEvent ()関数にて削除処理をすることになっています。

3.2.2.8 スレッド関連

仮想デバイスを実装するにあたり、デバイス内部でポーリングする場合等は新たにスレッドを生成して使うことができますが、NORTi Simualtor のタスクの中でスレッドを生成することはできません。
スレッドを生成できるのは、プログラムが立ち上がって、main()が始まる前の仮想デバイスの初期化の時だけです。
CNvdThread クラスは、NORTi Simulator コントロールパネルでスタート/ストップボタンを押す時にサスペンド・レジューム処理を行います。

protected: int regThread(int tid, CNvdThread * thr)
スレッドを登録する
int tid : スレッド ID(0～7)
CNvdThread * thr : スレッドのインスタンス、new CNvdThreadXXX()などで入力する
成功時に 0 をリターンする
デストラクタでスレッドを終了処理し、失敗時にもその場でデリートするので、new したポインタを保存する必要はない

protected: int suspend(void)
有効なスレッドを全部サスペンドする
NORTi Simulator が simvdev_control()を通じて呼び出す

protected: int resume(void)
有効なスレッドを全部レジュームする
NORTi Simulator が simvdev_control()を通じて呼び出す

protected: void suspendThread(int tid)
tid のスレッドをサスペンドする

protected: void resumeThread(int tid)
tid のスレッドをレジュームする

CNvdThread クラスは抽象クラスで、「novdthr.h」で宣言しています。
仮想デバイスの実装でスレッドの生成が必要な場合には、CNvdThread から派生したクラスを定義して使用してください。

CNvdThread::CNvdThread(const char *name = NULL, UINT msCycleTime = 0)
コンストラクタ
const char *name : スレッドの名前、デバッグのために指定するので、NULL でも構わない
UINT msCycleTime : スレッド実行の反復周期(ミリ秒)、この周期で operate()を実行する

CNvdThread::~CNvdThread(void)

デストラクタ

protected: UINT CNvdThread::nCycleTime

スレッド実行周期

コンストラクタの引数で初期化される

operate()は nCycleTime だけ待ってから最初の一回目を実行する前の待機時間である

protected: UINT CNvdThread::nReturn

スレッド本体の戻り値

0 に初期化される

void CNvdThread::execute(void)

nCycleTime の待機時間の前に即時実行される

コンストラクタを呼んだスレッドで 1 回実行され、その時にスレッドを生成する

protected: void CNvdThread::operate(void) = 0

純粋仮想関数で、スレッドのメインの処理はここで行う

スレッドが生成されると nCycleTime の待機後呼び出され、そのあと nCycleTime ごとに繰り返し実行される

virtual void CNvdThread::resume(void)

スレッドをレジュームする

virtual void CNvdThread::suspend(void)

スレッドをサスペンドする

virtual void CNvdThread::exit(void)

スレッドを終了する

protected: void CNvdThread::run(void)

nCycleTime 待ってから operate()を実行する

3.2.2.9 その他

virtual void setFlag(UINT id, BOOL value)

割り込み禁止フラグの状態を設定する

UINT id : フラグ ID

BOOL value : フラグ状態

デフォルトで全てのフラグはクリア状態

virtual BOOL getFlag(UINT id)

割り込み禁止フラグの状態を返す

```
typedef enum _FlagId {
    FI_INTDISABLE = 0,
    FI_COMMONMAX,
    FI_MAX = 256
} FlagId;
```

フラグ ID

FI_INTDISABLE は割り込み禁止状態で使う

FI_COMMONMAX は CNVirtualDevice で使うフラグの個数

派生クラスでは ID として、FI_COMMONMAX から FI_MAX-1 までの数を使える

void enableInterrupt(void)

割り込みを許可する(= 割り込み禁止フラグをクリアする)

デフォルト状態はクリアなので初期化されたら割り込みは有効状態である

void disableInterrupt(void)

割り込みを禁止する

禁止状態では、notifyEvent を実行しても addEvent と sim_interrupt が動作しないので、NORTi Simulator の割り込みは発生しない

BOOL isInterruptEnabled(void)

割り込みが許可状態なら TRUE をリターンする

3.2.3 GUI モジュール

NORTi Simulator では GUI モジュールサンプルとして、7 セグメント LED、スイッチボックス、LCD、キーパッドを提供します。

GUI モジュールは COM インターフェースを通してデバイスの外観の描画や入出力の動作をします。処理系には全く制限がありませんので、ユーザーが使い慣れた開発ツールを使って開発することができます。

ここでは GUI モジュールを使用するための COM インターフェースの実装方法を説明します。

3.2.3.1 初期化処理

NORTi Simulator のデバイス初期化で、GUI モジュールを実行する時には、コマンドライン引数で初期化に必要な情報を渡します。

初期化では、コマンドライン引数の解析と共有メモリの獲得をしなければなりません。

コマンドライン引数の解析

コマンドライン引数には GUI モジュールを動作させるための必要な実行情報が入っています。

SimDeviceDesc の cmdarg が次のように設定された場合に、

"320 240 1280 32 2" /* [width] [height] [byte line length] [bit per pixel] [pages] */

GUI モジュールには次のように実行情報が追加された引数が渡されるので、argv[2]から SimDeviceDesc の cmdarg の情報が入ることになっています。

[実行ファイル名] [classid],[id],[devIdx],[hwnd],[smName] 320 240 1280 32 2
[classid] : SimDeviceDesc の cls(16 進数)
[id] : SimDeviceDesc の id(16 進数)
[devIdx] : 生成したインターフェースの devIdx()値(16 進数)
[hwnd] : NORTi Simulator コントロールパネルのウィンドウハンドル(16 進数)
[smName] : 生成したインターフェースの共有メモリの名前(文字列)

argv[1]では[classid],[id],[devIdx],[hwnd],[smName]の情報が入るので、次のようなコードで解析できます。

```
{
    int i = 0;
    char *cmdarg = strtok(argv[2], ",");
    while (cmdarg != NULL) {
        if (i == 0)
            g_classid = strtoul(cmdarg, NULL, 16);
        else if (i == 1)
            g_id = strtoul(cmdarg, NULL, 16);
        else if (i == 2)
            g_idx = strtoul(cmdarg, NULL, 16);
        else if (i == 3)
            g_hwnd = (HWND)strtoul(cmdarg, NULL, 16);
        else if (i == 4)
            strcpy(g_mapfilename, cmdarg);
        cmdarg = strtok(NULL, ",");
        i++;
    }
}
```

arg[1]から得た情報は、共有メモリの獲得、メッセージ通信などで使ってください。
arg[2]から入っている情報で共有メモリの使用に必要な情報を読み取ってください。この情報はインターフェースクラスごとに意味が異なりますから、それぞれの状況に応じて対応を変えてください。

共有メモリの獲得

共有メモリを獲得するためには、名前が必要になるので、先にコマンドライン引数から情報を受けることが必要です。
もし、共有メモリを使う必要がない場合でも、インターフェースを使った通信のために、SharedmemHeader 分の共有メモリを使わなければなりませんので、共有メモリの初期化は必須です。
次のようなコードで共有メモリの獲得ができます。

```
{
```

```
    g_hMapFile = OpenFileMappingA(FILE_MAP_READ | FILE_MAP_WRITE, FALSE, g_mapfilename);
    if (g_hMapFile == NULL)
        MessageBoxA(NULL, g_mapfilename, "Error", MB_OK);
        bufsm = (unsigned char *)MapViewOfFile(g_hMapFile, FILE_MAP_READ | FILE_MAP_WRITE,
                                               0, 0, sm_size);
        if (bufsm == NULL) {
        MessageBoxA(NULL, "Could not map view of file.¥n", "Error", MB_OK);
    } else {
        bufhdr = (DWORD *)bufsm;
        SharedmemHeader *smhdr = (SharedmemHeader *)bufhdr;
        smhdr->hwnd = hwnd;
    }
}
```

sm_size は使う共有メモリの大きさです。コマンドライン引数の arg[2]から、使用する共有メモリの大きさを計算して使ってください。

NORTi Simulator との通信のために、共有メモリの先頭に GUI モジュールの自分のウィンドウハンドルを書く必要があります。そのあと NORTi Simulator がアクセスして、GUI モジュールのタイトルを次のように変更します。

NORTi Virtual Device [ファイル名] [共有メモリ名]

3.2.3.2 出力処理

NORTi Simulator から GUI モジュールへの出力の処理を説明します。

NORTi Simulator のインターフェースの実装によって違ってくるのですが、CNvdDisplay を使用する場合には、WM_PAINT が発生した時、共有メモリの特定位置に出力するデータがあるので、そのデータを読み取って必要な形に描画することで、出力機能を実装できます。

その他、ユーザーがインターフェースを追加すると、PostMessage()のイベント、WPARAM、LPARAM の情報を使って自由に出力することができます。

詳しい説明は Windows プログラミング一般になるので、各サンプルのソースと、Windows API や MFC の SDK を参考にしてください。

3.2.3.3 入力処理

GUI モジュールを使った NORTi Simulator の入力方法に関して説明します。

出力と同じく、NORTi Simulator のインターフェースの実装によっていろいろな動作を実現します。

例えば CnvdKey で実現しているキー入力のように、データの量が少ない場合は WM_USER イベントの引数を使って NORTi Simulator 側にデータを渡せます。

```
PostMessage(g_hwnd, WM_USER + g_idx, 0, id);
```

のように簡単に入力を通知できます。

仮想デバイスを生成するときに決める devIdx()の値を WM_USER に渡すことで、自分のインター

フェースにメッセージを通知してください。なお、必要な情報は WPARAM と LPARAM だけで渡してください。
入力する情報が大きい場合は、共有メモリを使う必要があります。WPARAM と LPARAM に共有メモリのオフセットなどの情報を渡して、NORTi Simulator のインターフェースで、受信処理を行うことを想定していますが、実装はユーザーで行ってください。

3.3 仮想デバイスサンプル

ここでは、サンプルデバイスとそのビルドに関して説明します。

3.3.1 インターフェースクラス

3.3.1.1 インターフェースクラスライブラリ

インターフェースクラスの CNVirtualDevice はビルドしなおす必要はありません。
もし、サポートなどの理由でビルドしなおす時には、

SMP¥SIM¥common¥VirtualDevice¥VirtualDevice.sln

を使ってビルドしてください。

3.3.1.2 派生クラスの追加

派生クラスの CNvdKey と CNvdDisplay などは、プログラムと一緒にソースからビルドされます。
元のソースファイルを変更しても構いませんが、元のソースを維持するために、派生クラスを追加作成する方法を推奨します。
派生クラスを作成するときは、インターフェースを区別するために、

```
#define VDEVCLASS_KEYP      (0x01 | VDEVCLASS_KEY)
```

のようにクラス ID を追加で宣言して使用してください。
派生クラスのコンストラクタのデフォルト引数には、宣言したクラス ID を設定してください。

```
explicit CNvdKeyPolling(const SimDeviceDesc *desc, UINT clsid = VDEVCLASS_KEYP);
```

あとは、変更したいメンバー関数だけ実装してください。
また、C ラッパー関数のため、別途のソースファイルを追加して、NovdUser.cpp の代わりに使ってください。その中で、作成した派生クラスのコンストラクタを呼び出す new 文を使って、create_user_device()関数を実装してください。

3.3.2 サンプル GUI モジュールの構成

デバイス名	ソリューションファイル	バイナリ	環境
7 セグメント LED	SMP¥SIM¥GuiModule¥mfc7Seg¥mfc7seg.sln	BIN¥nvd7seg.exe	MFC
スイッチボックス	SMP¥SIM¥GuiModule¥mfcSwBox¥mfcSwBox.sln	BIN¥nvdSwbox.exe	MFC
LCD	SMP¥SIM¥GuiModule¥win32Lcd¥win32Lcd.sln	BIN¥nvdlcd.exe	Win32
キーパッド	SMP¥SIM¥GuiModule¥win32¥win32Key.sln	BIN¥nvdkey.exe	Win32

NORTi Simulator をインストールすると、表の位置にサンプルのバイナリが配置されます。
変更して使う場合など、再ビルドが必要な場合は以下を参考にしてビルドしてください。
ビルドした GUI モジュール(.exe ファイル)を使う時には、NORTiSIM¥BIN フォルダに置いて SimDeviceDesc にファイル名だけを書き込んで使います。あるいは他のフォルダに置いて、絶対パスでファイル名を指定してください。
全てのサンプルプロジェクトは、Visual Studio 2015 でビルドできます。

3.3.3 ディスプレイ装置

サンプルデバイスの 7 セグメント LED と仮想 LCD は、共に CNvdDisplay インターフェース(novdDisp.cpp)を使っています。

```
#define VDEVCLASS_7SEG          (0x0001 | VDEVCLASS_DISP)
#define VDEVCLASS_LCD           (0x0002 | VDEVCLASS_DISP)

const SimDeviceDesc dev_7seg = {
    VDEVCLASS_7SEG, 0,
    0, 0,
    0, 0,
    0,
    "nvd7seg.exe",
    "4 1 4 8 2"  /* [width] [height] [byte line length] [bit per pixel] [pages] */
};
const SimDeviceDesc dev_lcd = {
    VDEVCLASS_LCD, 0,
    0, 0,
    0, 0,
    0,
    "nvdlcd.exe",
    "320 240 1280 32 2" /* [width] [height] [byte line length] [bit per pixel] [pages] */
};
void out_rawled(int led, unsigned char d)
{
    buf_7seg[led] = d;
    simvdev_disp_update(gdid7Seg);
}
void ini_cpu()
{
     :
    gdidLcd = simvdev_create(&dev_lcd);
    gdid7Seg = simvdev_create(&dev_7seg);
     :
```

```
    simvdev_get_buffer(gdid7Seg, 0, &buf_7seg);
    :
}
```

CNvdDisplay を使用する場合、NORTi Simulator のユーザープログラム側では上記のような簡単なコードを書くだけで、新たなディスプレイ装置を作成することができます。
nvdlcd.exe の GUI モジュールは cmdarg の情報に沿って画面の大きさを変えているので、cmdarg の情報を変更して必要な大きさの画面をシミュレートすることができます。
CnvdDisplay を使うには、まず cmdarg の縦、横、横線のバイト数、バッファ個数情報に沿って生成した共有メモリの画面バッファに描画内容を書き込み、WM_PAINT メッセージで GUI モジュールに通知すると描画が行われます。
サンプル GUI モジュールの「win32Lcd.cpp」は、Win32 の 32 ビットフォーマットに対応していますが、一例として、RGB565 フォーマットから Win32 の 32 ビットフォーマットに変換する transRGB565_BGRA8888()関数を用意しています。このように実際のハードウェアで使うフォーマットの変換関数を作成すれば、実際のディスプレイ装置をシミュレーションすることができます。
ユーザープログラムでは次のようにバッファの位置を示すポインタを受けて、バッファを操作してから simvdev_disp_update()で WM_PAINT を GUI モジュールへ通知します。

```
simvdev_disp_lock_buffer(gdidLcd, &buf);
…バッファの操作…
simvdev_disp_update(gdidLcd);
simvdev_disp_unlock_buffer(gdidLcd);
```

lock/unlock 関数を使っていますが、サンプルのインターフェースクラスでは、ロック動作は実装されていません。
ロック処理が必要な場合には、派生インターフェースクラスを追加して以下の関数を実装しなおしてください。ただし、NORTi Simulator のタスクから呼ぶ関数なので、同期しないで失敗してリターンするように実装して、待ちはタスク部のプログラムで実装してください。

```
unsigned char * CNvdDisplay::lockBuffer(void)
{
    return getBuffer(0);
}

void CNvdDisplay::unlockBuffer(void)
{
}
```

一般的なディスプレイ装置の場合は、GUI モジュールを作り直すだけで実装できます。
7 セグメント LED のサンプルの「mfc7Seg.cpp」場合も、同じ CNvdDisplay を使用して 8 ビットフォーマットの画素を使い、1 つの画素を 1 つのセグメント LED で表現するという方法を採用しています。

3.3.4 入力装置

サンプルデバイスのスイッチボックスと仮想キーパッドは、共に CNvdKey インターフェース(novdKey.cpp)を使っています。

```
#define VDEVCLASS_SWBOX         (0x0001 | VDEVCLASS_KEY)
#define VDEVCLASS_KEYPAD        (0x0002 | VDEVCLASS_KEY)

const SimDeviceDesc dev_swbox = {
    VDEVCLASS_SWBOX, 0,
    1, 9,
    0, 0,
    0,
    "nvdSwbox.exe",
    "2 8 64 2" /* [width] [height] [buffer size] [pages] */
};
const SimDeviceDesc dev_keypad = {
    VDEVCLASS_KEYPAD, 0,
    28, 8,
    0, 0,
    0,
    "nvdkey.exe",
    "3 4 48 2 1,2,3,4,5,6,7,8,9,*,0,#" /* [width] [height] [buffer size] [pages] [key layout] */
};
void out_rawled(int led, unsigned char d)
{
    buf_7seg[led] = d;
    simvdev_disp_update(gdid7Seg);
}
void ini_cpu()
{
    gdidSwbox = simvdev_create(&dev_swbox);
     :
    gdidKey = simvdev_create(&dev_keypad);

    simvdev_get_buffer(gdidSwbox, 0, &buf_swbox_in);
    simvdev_get_buffer(gdidSwbox, 1, &buf_swbox_out);
     :
}
```

CNvdKey と nvdkey.exe を使用する場合に、ユーザープログラム側では上記のような簡単なコードで、必要なキーパッド装置をシミュレートすることができます。
nvdkey.exe は cmdarg の横、縦の分のキーを生成し、レイアウトに入ったラベルを付けます。
長方形配列のようなレイアウト以外のキー配置をする場合には、nvdkey.exe を変更して使用するか、作り直してください。
CNvdKey の場合、入力されたキーは割り込みキューに入りますが、デバイスに文字キーを内蔵しているので、文字キー配列から直接読み込むこともできます。

```
int CNvdKey::readChar(char *c, int len)
```

もし、文字キューを使わない場合でも、周期的に readChar を呼び出して捨ててください。
CNvdKey を使っても cmdarg の情報に沿って、共有メモリにバッファを生成するので、出力機能

で使用することもできます。
スイッチボックスの nvdSwbox.exe では、共有メモリを 2 ページ生成して、各ページを入力、出力で使用します。入力の検知にはポーリングを使っています。
次のように simulator.c にて入出力を実装しています。

```
BOOL input(int n)
{
    unsigned char *buf = &buf_swbox_in[(n << 2) + 3];
    if (*buf & 0x80)
        return TRUE;
    else
        return FALSE;
}

void output(int n, BOOL b)
{
    unsigned char *buf = &buf_swbox_out[(n << 2) + 3];
    if (b)
        *buf |= 0x80;
    else
        *buf &= 0x7f;
    simvdev_update(gdidSwbox);
}
```

ボタンは割り込みを使っていますが、割り込みキューにはどのボタンから発生した割り込みかの情報を入れています。

3.3.5 割り込み

サンプルプログラムの swBox8.c の inh_body()の割り込みハンドラでは、次のように、インターフェースで使う割り込みキューのポインタを参照し、同じ形の構造体で割り込み情報を処理します。

```
void inh_body(void)
{
    struct QueType {
        unsigned char st;
        unsigned char key;
    } *data;
    while (data = simvdev_intque(gdidSwbox)) {
        if (data->st == 0) {
            if (data->key == 1) {
                wup_tsk(TSK_CNT1);                  /* タスク起床 */
            } else {
                wup_tsk(TSK_CNT2);                  /* タスク起床 */
            }
        }
        simvdev_pop_int(gdidSwbox);
    }
}
```

keytest.c では、キー入力の通知だけ割り込みを使って、simvdev_key_read_char()でキーバッ

ファを読んでいますが、割り込みキューにもデータが入っているので、使わなくても削除する必要があります。デバイスの割り込みキューがあふれないように simvdev_pop_int()で使わないデータを読み捨ててください。

次は keytest.c の割り込みハンドラの例です。

```
static void keypad_int(int id)
{
    int tid;
    while (simvdev_intque(id)) {
        simvdev_pop_int(id);
    }
    if ((tid = g_tid[id]) != 0)
        iwup_tsk(tid);
}
```

第 4 章 シミュレータ固有の関数

Windows 上で動く NORTi Simulator と、実際のターゲットで動く NORTi とは、完全に同じコードからビルドされた共通のカーネルを使用します。また、LAN ポート、シリアルポートのドライバはサンプルで用意しているものをそのまま使用できます。
しかしながらドライバ関連と割り込み処理はハードウェアによって変える必要がありますので、ここではサンプルドライバとその他のハード依存部を実装するために使う API を紹介します。ハードウェア依存部を作成し、シミュレータでの開発をする場合に参考にしてください。

4.1 simulator.c

simulator.c は、サンプルプログラムで使用するハードウェアに依存するコードなどをまとめて実装しています。
プログラムで main()関数が実行される前に呼ばれる cpu_ini()関数や、仮想デバイスの初期化関連の関数もこのファイルで定義しています。
このファイルはコピーして、必要な場合に変更して使用してください。
simulator.c では、仮想デバイスの関数、シミュレータ専用 API を使っているので、下記の仮想デバイス関連の関数、シミュレータ専用 API についての解説を参照してください。

4.2 仮想デバイス関数

ここで紹介する関数は、novdevfunc.cpp の関数で、CNVirtualDevice インターフェースのラッパー関数なので、仮想デバイスに共通する基本機能だけを紹介します。
シミュレートするハードウェアに固有の機能は､追加で実装した関数(クラスメンバーまたはラッパー関数)で使用してください。
関数の内部の詳細は第 3 章 仮想デバイスを参照してください。

4.2.1 生成関連

int simvdev_create(const SimDeviceDesc *desc)
デバイスを生成する
SimDeviceDesc *desc : デバイス記述子
デバイスの ID をリターン
失敗時には例外発生

SimDeviceDesc *simvdev_ref(int id)
生成時使用したデバイス記述子の写本のポインタ
int id : デバイス ID

simvdev_create()の中では、create_device()関数、またその中で、create_user_device()を呼び出します。追加した仮想デバイスインターフェースもcreate_user_device()関数を実装することで、既存のインターフェースと同じく、simvdev_create()関数で生成することができます。

4.2.2 割り込み関連

int simvdev_ena_int(int id)

割り込みを許可

int id　　：デバイスID

成功時0をリターン

int simvdev_dis_int(int id)

割り込みを禁止

int id　　：デバイスID

成功時0をリターン

void *simvdev_intque(int id)

割り込みキューの先頭データのポインタを取得

int id　　：デバイスID

void simvdev_pop_int(int id)

割り込みキューの先頭データを削除

int id　　：デバイスID

4.2.3 GUI、バッファ関連

void simvdev_update(int id)

GUIモジュールへ画面更新を要求する

int id　　：デバイスID

int simvdev_get_buffer(int id, int page, void **buffer)

共有メモリバッファのポインタを取得

int id　　：デバイスID

int page　　：バッファページ番号

void **buffer　　：ポインタ

成功時0をリターン

4.2.4 デバイス制御関連

ここの関数はNORTi Simulatorから実行するハンドラ関数です。ユーザーが直接呼び出す必要はありません。

int simvdev_control(int code)

仮想デバイスを制御

int code　　：制御コード

DLLCTRL_EXIT　　：終了

DLLCTRL_SUSP　　：一時停止

DLLCTRL_RESM　　：再開

DLLCTRL_ACTIVE　　：GUI モジュールのウインドウをアクティブにする

LRESULT simvdev_message(HWND hwnd, UINT msg, WPARAM wparam, LPARAM lparam)

GUI モジュールなど、NORTi Simulator 外部からのメッセージハンドラ

hwnd　　：NORTi Simulator コントロールパネルのウィンドウハンドル

UINT msg　　：メッセージ番号

WPARAM wparam　　：WPARAM 引数

LPARAM lparam　　：LPARAM 引数

4.3 NORTi Simulator 専用 API

4.3.1 LAN 関連関数

次に列挙した関数は LAN ポートドライバで使う API で、関数の本体は NORTi Simulator の BIN¥nortisim.exe で実装しています。そのためソースは非公開になっています。
NORTi Simulator の LAN ドライバの nonedrvr.c でのみ使用できる API なので、詳しい説明は省略します。

```
UINT nic_get_rx_count(void)
UINT nic_ext_rx_pac(void)
UB* nic_get_rx_buf(void)
void nic_rel_rx_pac(void)
void nic_set_tx_len(UINT n)
UB*  nic_get_tx_buf(void)
void nic_req_tx_pac(void)
BOOL nic_inq_tx_reqed(void)
void nic_set_lan_isr(void *pv_hr_isr_lan)
UINT nic_get_isr_flag(void)
void nic_clr_isr_flag(UINT flag)
void nic_set_mac_addr(const UB *macaddr)
BOOL nic_sta(void)
void nic_end(void)
```

4.3.2 シリアル関連関数

下記の関数はシリアルポートドライバ関数で使う API で、関数の本体は NORTi Simulator の BIN¥nortisim.exe で実装しています。ソースは非公開になっています。
NORTi Simulator の LAN ドライバの nossim.c だけで使用できる API なので、詳しい説明は省略します。

```
BOOL com_setup(UINT ch)
void com_cleanup(UINT ch)
DCB* com_inq_dcb(UINT ch)
void com_mod_dcb(UINT ch)
void com_ena_int(UINT ch)
void com_dis_int(UINT ch)
UINT com_get_isr_flag(UINT ch)
void com_clr_isr_flag(UINT ch, UINT flag)
```

UINT com_get_imr_flag(UINT ch)

void com_set_imr_flag(UINT ch, UINT flag)

void com_clr_imr_flag(UINT ch, UINT flag)

UINT com_read_char(UINT ch, BYTE *py)

BOOL com_write_char(UINT ch, BYTE y)

void com_control_rts(INT ch, BOOL be)

void com_control_dtr(INT ch, BOOL be)

void com_purge(INT ch, UINT nf)

void com_break_tx(INT ch, BOOL be)

const COMMPROP* com_inq_properties(UINT ch)

void com_ren_status(UINT ch)

UINT com_get_errors(UINT ch)

const COMSTAT* com_ref_status(UINT ch)

4.3.3 その他の関数

4.3.3.1 割り込み機能

int sim_interrupt(UINT irq, UINT pri)

仮想割り込みを発生させる

UINT irq　　：割り込み番号(0～31)

IUNT pri　　：割り込み優先度(0～15)

成功時に 0 をリターン

仮想割り込み機能は、仮想デバイスの notifyEvent()を通じて使用してください。

4.3.3.2 初期化機能

int sim_init_kertim(UINT irq, UINT pri, UINT msec)

タイマ割り込みを初期化する

UINT irq　　：タイマ割り込み番号(0～31)

IUNT pri　　：タイマ割り込み優先度(0～15)

IUNT msec　　：タイマ割り込み周期(MSEC)

成功時に 0 をリターン

int sim_init_ethernet(UINT irq, UINT pri)

LAN ポートを初期化する

UINT irq　　：LAN 割り込み番号(0～31)

IUNT pri　　：LAN 割り込み優先度(0～15)

成功時に 0 をリターン

int sim_init_serial(UINT irq, UINT pri)

シリアルポートを初期化する

UINT irq　　：シリアルポート割り込み番号(0～31)

IUNT pri　　：シリアルポート割り込み優先度(0～15)

成功時に 0 をリターン

4.3.3.3 仮想 I/O 機能

int sim_input(UW addr, UW *data)

仮想入出力の入力機能

UW addr　　：アドレス

UW data　　：入力する data をコピーするポインタ

成功時に 0 をリターン

int sim_output(UW addr, UW *data)

仮想入出力の出力機能

UW addr　　：アドレス

UW data　　：出力する data のポインタ

成功時に 0 をリターン

仮想 I/O で使用できるアドレスは次の通りです。

マクロ	アドレス	
SIMADDR_SYSTICKDIFF	0	
SIMADDR_MACADDR	160	MAC アドレス情報
SIMADDR_IPADDR	320	IP アドレス情報
SIMADDR_USER	4096	仮想デバイスで予約

SIMADDR_USER から上位のアドレスにアクセスすると、「dllmain.cpp」の devIO()関数を通して、また、「simulator.c」の dev_output()と dev_input()が呼び出されます。もし、仮想デバイスの共有メモリなどを仮想 IO アドレスに割り振る場合は、dev_output()と dev_input()の中身を実装してください。

4.3.3.4 デバッグ出力機能

int sim_trace_opt(UINT lvl, UW opt)

デバッグ出力機能の設定

UINT lvl　　：出力レベル、設定したレベル以上のエラーレベルだけ出力される

NDBG_ERR_NONE（全部出力）

NDBG_ERR_INFO

NDBG_ERR_WARN

NDBG_ERR_ERROR

NDBG_ERR_PANIC

NDBG_ERR_MAX（出力しない）

UW opt　　：出力フォーマットの設定

0、NDBGOPT_NO_FILE、NDBGOPT_TO_CONSOLE、NDBGOPT_NO_TIME の組み合わせ

0 をリターン

int sim_trace_out(UINT type, const char * str1, const char * str2, UINT line, UINT lvl, UINT ercd)

デバッグ出力

UINT type　　：出力タイプ

DOSTYPE_TRACE - 改行、時間、関数名などの情報を付けて str1 を出力

DOSTYPE_RAW - str1 だけを出力

const char *str1　：出力する文字列

const char *str2　：関数名または NULL(オプション)

UINT line　　：行数

UINT lvl　　：出力レベル

UINT ercd　　：予約(0)

0 をリターン

DOSTYPE_TRACE の出力形式は、次のようになります。

[改行]

[時間]|[レベル]|[関数名]|[行数] [文字列]

(例)

18:45:33|LVLINFO|MainTask()|258 test string

DOSTYPE_RAW を使うと、改行とフォーマットなしで出力しますので、文字列を続けて出す際に使用してください。

以上のデバッグ出力 API を使用するときには、SMP¥SIM¥common¥nodbg.h をインクルードしてください。

直接 API 関数を呼び出しても構いませんが、nodbg.h の TRACELVL と TRACECNT マクロを用意しています。マクロを使うと、TRACEOUT マクロの条件でソースコードの変更なしで全部のデバッグ出力を無効にすることができます。

また、次のように引数を簡単に書くことができます。

TRACELVL(level, string)

TRACECNT(level, string)

4.3.4 API の実体

NORTi Simulator は、コントロールパネルの nortisim.exe 実行ファイルとユーザープログラムの DLL で構成されます。DLL が初期化されたら、関数ポインタで nortisim.exe 側の API を呼び出せます。

以上の API は C 関数へのポインタになっているので、呼び出すときには必ず関数ポインタを宣言してから呼び出さなければなりません。

NORTi Simulator では、シミュレータ専用のヘッダである「kernels.h」のなかで「KernelApi.h」、「LanDrvApi.h」、「noswin.h」、「SioDrvApi.h」、「PioDrvApi.h」などのヘッダファイルで関数ポインタ宣言がされていますから、「kernel.h」をインクルードすることで API を使うことができます。

もし、kernel.h のインクルードなしで、関数ポインタを普通の C 関数として呼び出してしまうと例外が発生しますのでご注意ください。

第 5 章 プログラミング上の注意事項

5.1 使用上の注意事項

5.1.1 二重起動

NORTi Simulator の二重起動はできません。

5.1.2 メモリリーク

デバッガ終了時に CwinThread、threadcore のメモリリークが発生する場合がありますが、問題はありません。

5.1.3 ディスパッチの誤差

NORTi Simulator のタスクは Windows のスレッドによって実現されています。スレッドのディスパッチタイミングは Windows の負荷等によって影響を受けるため、リアルタイム OS の応答性能は期待できません。
タスクステートウォッチャのグラフ上でも誤差が生じます。

5.1.4 デバッグメッセージ

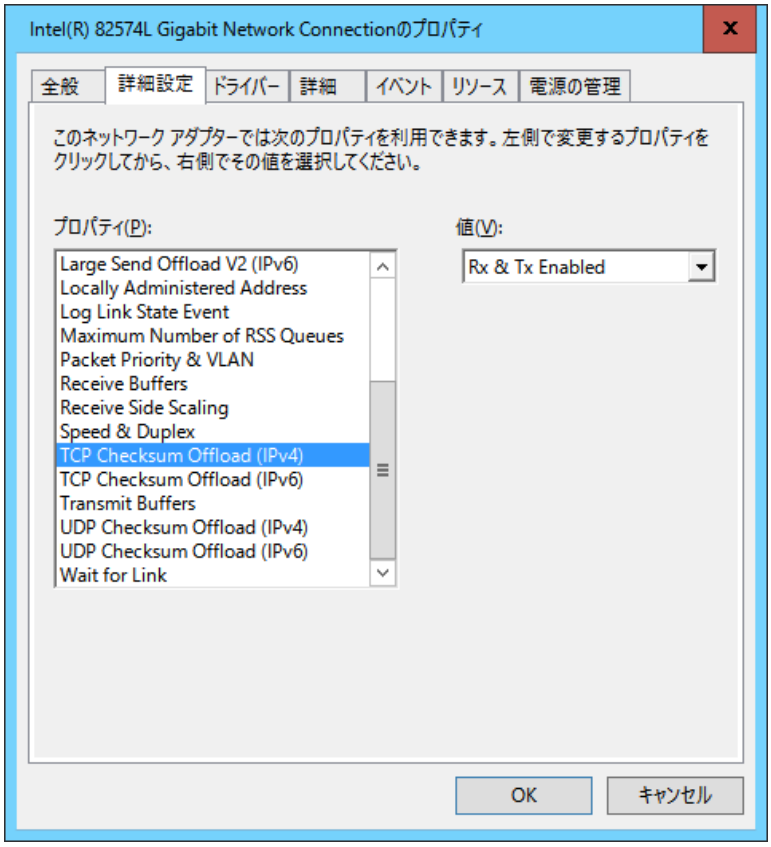
デバッグの開始時点で次のようなメッセージが表示されることがありますが､問題ありません。

'***.DLL' をロードしました、合致するシンボル情報は見つかりませんでした。

5.1.5 Windows のオフロード処理

Windows では、通信速度の向上や、CPU の負担軽減のために、各種のオフロード機能をサポートしています。ハードウェア技術の発展で、オフロードをサポートする LAN アダプタの増加に伴い、オフロード機能がデフォルトで有効になっている PC が増えています。
NORTi Simulator を 1 台の PC で Windows アプリと通信テストを行う際には、各種オフロード機能が動作に影響があります。
ハードウェアで処理される前のパケットがそのままキャプチャされることで、実際物理回線から受信したパケットとは少し違ってきます。
チェクサムオフロード機能に対しては、欠けたチェクサムを計算して入れることで、回避していますが、他のオフロード機能に対しては、対応していません。
回線を通した外部との通信には問題ないが、同一 PC 内同士で正常に通信できない場合、速度が非常に遅くなる場合には、各オフロード機能を無効にしてください。

オフロード機能を無効にするには、Windows の「デバイス マネージャー」を使います。
使用するネットワークアダプターを右クリックし、プロパティを選択すると、プロパティ画面が表示されます。



「詳細設定」で、

IP チェクサムオフロード
送信 IP チェクサムオフロード
TCP 送信チェクサムオフロード
UDP 送信チェクサムオフロード
一括送信オフロード
Offload Transmit IP Checksum
Offload Transmit TCP Checksum
TCP Segmentation Offload
Large Send Offload
Offload TCP Large Send
Offload TCP Segmentation

などの設定を無効にしてください。
ドライバによってサポートするオフロード機能が異なる場合があり、または同じ機能で表現が異なることがありますので、動作に異常のある場合にはオフロード関係のオプションを全部無

効にしてみてください。

5.2 コーディングの注意事項

5.2.1 Windows API による待ち

タスクの待ちは ITRON システムコールのみによって行ってください。
getc、Sleep、WaitForSingleObject 等による待ちを行うと、シミュレータエンジンのタスクのディスパッチが正常に行われなくなって不具合の原因になります。

5.2.2 エンディアン

NORTi Simulator は x86 系 CPU のネイティブコードで実行されるため、リトルエンディアンとなります。実際のターゲットがビッグエンディアンの場合はコーディングに注意してください。

5.2.3 型のサイズ

NORTi Simulator は x86 系 CPU のネイティブコードで実行されるため、char は 1 バイト、short は 2 バイト、int および long は 4 バイトとなります。実際のターゲットとサイズが異なる場合は注意してください。
また、Visual Studio のデフォルトの time_t 型は 64 ビットですが、NORTi Simulator では 32 ビットを前提としているため、コンパイラオプションで _USE_32BIT_TIME_T マクロの定義を追加する必要があります。

5.2.4 メモリサイズ

NORTi Simulator 上ではメモリサイズの制限が緩いため、実際のターゲットでメモリ不足にならないよう注意が必要です。

5.2.5 タスクスタックサイズ

NORTi Simulator のタスクは Windows のスレッドによって実現されているので、スタックはスレッドのものが使われます。タスク生成時のスタックサイズパラメータは、タスク情報内には保持されますが、実際には Visual C++のデフォルトサイズ(1MB)で生成されます。そのためパラメータで指定したサイズを超えてスタックを消費してもエラーを起こさずに動作する場合があります。
同じソースコードを実際のターゲットで使うときには、スタックサイズに注意してください。

5.2.6 カーネルタイマ

カーネルタイマのソースとして Windows のタイマを使用しており、その精度は Windows に依存

しています。タイマについてはだいたいの目安と考えてください。

5.2.7 メモリの直接参照

ビットフィールドやアドレスの直接指定によるメモリ参照を行うと Windows のページ違反が起きます。

5.2.8 割り込みハンドラ

割り込みハンドラ実行中は割り込み禁止状態となります。
Windows スレッドのディスパッチによって割り込みハンドラが処理されるので、応答速度が遅くなります。
また、割り込み発生が失敗した時には遅延処理しますので、実際のターゲットの割り込み応答速度は期待できません。

5.2.9 タスクやハンドラ以外でのシステムコール

初期化時を除いて原則的に使用できません。

5.2.10 スレッド内ループ

スレッドを作成して無限ループする場合、ループの最後に Sleep(0)を挿入して、スレッドの切り替わるタイミングを作ってください。

5.2.11 名前つきハンドル

CreateEvent、CreateMutex などでハンドルを作成する場合、プロセス ID 等を付加したマシン内でユニークな名前付ハンドルとすることを推奨します。

NORTi Simulator ユーザーズガイド

2016 年 12 月版

株式会社ミスポ http://www.mispo.co.jp/
〒222-0033　神奈川県横浜市港北区新横浜3-20-8
BENEX S-3 12F
TEL 044-829-3381 FAX 044-829-3382

一般的なお問い合せ sales@mispo.co.jp
技術サポートご依頼 norti@mispo.co.jp