

第 3 版

画像認識ユニット

NVP-Ax230SDK

Software Development Kit

Fine Vision Processor

for Linux

ユーザーズマニュアル

maxell

マクセルシステムテック株式会社

はじめに

このたびはNVP-Ax230シリーズの画像処理ユニット(NVP-Ax230CL/235CL)、および、「NVP-Ax230SDK for Linux」をお買い上げいただきまして、誠にありがとうございます。

本マニュアルはNVP-Ax230シリーズを使用したアプリケーション作成のための基本ソフトウェアである「NVP-Ax230SDK for Linux」について記載します。ハードウェアについては、各ボードのハードウェアマニュアルをご参照ください。

なお、本マニュアルではNVP-Ax230CL/235CLの画像処理ユニットを特に区別する必要がある場合には、NVP-AX230と記載しています。

ご注意

- システムの構築やプログラム作成などの操作を行う前に、本マニュアルの記載内容をよく読み、書かれている指示や注意を十分理解して下さい。誤った操作によりシステムの故障が発生することがあります。
- 本マニュアルの記載内容について理解できない内容、疑問点または不明点がございましたら、弊社営業窓口までお知らせ下さい。また、弊社ホームページのお問い合わせのページからも受け付けてますのでご利用ください。
<http://www.systemtech.maxell.co.jp/solution/vp/>
- お客様の誤った操作に起因する、事故発生や損害につきましては、弊社は責任を負いかねますのでご了承ください。
- 弊社提供のハードウェアおよびソフトウェアを無断で改造しないでください。この場合の品質および安全につきましては、弊社は責任を負いかねますのでご了承ください。
- 本マニュアルの内容について予告なく変更する場合があります。

目次

1.	製品概要.....	1
1.1	NVP-Ax230SDK for Linuxの概要	1
1.1.1	特徴.....	1
1.1.2	前提条件.....	1
1.2	画像処理ユニットNVP-Ax230の概要.....	2
1.2.1	NVP-Ax230CLのハードウェア構成	2
1.2.2	NVP-Ax235CLのハードウェア構成	3
1.3	Linuxアプリケーションの開発	4
2.	Linuxシステム概要	5
2.1	Linuxシステム起動の処理フロー(デフォルト)	5
2.2	Linuxシステム概要	6
2.3	ハードウェア資源とアクセス方法.....	7
3.	パッケージ内容	8
3.1	CD-R	8
3.2	インストール手順.....	8
3.3	パッケージ内容	9
3.3.1	ヘッダファイル	10
3.3.2	共有ライブラリファイル.....	10
3.3.3	デーモン.....	11
3.3.4	サンプルプログラム	11
3.3.5	Linuxツール.....	11
4.	開発環境の準備	12
4.1	VirtualBoxのインストール	12
4.2	Ubuntuのインストール	12
4.3	Ubuntu12.04をVirtualBoxに登録	13
4.4	VirtualBox設定	13
4.5	Ubuntu設定	14
4.6	Sourcery G++ Liteのインストール	15
4.7	Tera Termのインストール	15
4.8	Ax230SDKファームウェア書き換え.....	16
4.8.1	Linux書換え時のSW設定	16
4.8.2	ネットワーク設定.....	16
4.8.3	Linuxフォーマット手順	17
4.8.4	従来SDK(SH動作)変更手順.....	17
5.	Linuxチュートリアル	18
5.1	Linux起動時のSW設定	18
5.2	チュートリアル環境.....	18
5.3	Tera Termの設定	19
5.4	NVP-Ax230の起動	19
5.5	ログイン	20
5.6	ネットワークの設定.....	20
5.7	共有フォルダのマウント.....	21
5.8	telnetの実行	21
5.9	hello.cのコーディング	22

5.10	hello.cのクロスコンパイル	23
5.11	a.outの実行	23
5.12	他ストレージのマウント.....	24
5.12.1	USBメモリのマウント.....	24
5.12.2	SDカードのマウント.....	25
5.12.3	NAS-HDDのマウント	26
5.13	cramfsによる各種設定	27
5.13.1	cramfsイメージの展開	27
5.13.2	各種システム設定の変更	27
5.13.3	cramfsイメージの作成	27
5.13.4	cramfsイメージの作成時の注意事項.....	27
5.13.5	cramfsイメージの書き込み.....	28
5.13.6	cramfsによるシステム設定変更例	30
6.	Linux画像認識アプリケーションの開発	31
6.1	環境変数の設定	31
6.2	impcmdddデーモンの起動	31
6.3	impcmdddデーモン自動起動	32
6.3.1	ライブラリとデーモンをcramfs上に展開.....	32
6.3.2	/etc/init.d/rcSスクリプトの修正	32
6.3.3	cramfsイメージの作成と書き込み.....	33
6.3.4	システム再起動とimpcmdddデーモン確認	33
6.4	Linuxアプリケーションの開発と実行	34
6.4.1	サンプルプログラムのmakeファイルについて.....	34
6.4.2	サンプルプログラムのmakeと実行	35
6.5	注意事項	35
7.	SH画像認識アプリケーションの開発.....	36
7.1	提供ファイル	36
7.2	SHアプリケーションの開発と実行	36
8.	コマンドリファレンス	37
8.1	デバイスID無しリモートコマンド	37
8.1.1	デバイスID無しリモートコマンドの使用開始(StartIP)	37
8.1.2	デバイスID無しリモートコマンドの使用停止(StopIP)	37
8.1.3	デバイスID付きリモートコマンド	38
8.1.4	デバイスID付きリモートコマンドの使用開始(OpenIPDev)	38
8.2	API初期化APIサポートコマンド	38
8.2.1	画像処理ボードのリセット(ResetIP).....	38
8.3	システム制御コマンド	39
8.3.1	バージョン情報の取得(GetIPVersionInfo)	39
8.4	映像表示コマンド	40
8.4.1	映像表示の初期化(du_init).....	40
8.5	構成制御	41
8.5.1	カメラ映像入力構成制御設定(SetConfigCamera)	41
8.5.2	カメラ映像入力プログラムの終了(ExitIPCamera)	41
8.5.3	画像メモリ表示構成制御設定(SetConfigView).....	41
8.5.4	画像メモリ表示プログラム終了(ExitIPView).....	41
8.6	SCIコントロール	42
8.7	タイマ.....	42
8.7.1	時刻の登録(SetIPTime).....	42
8.7.2	時刻の取得(GetIPTime).....	42
8.8	USB-HID制御コマンド	42
8.9	ファイルアクセス.....	43
8.9.1	PC-オンボードディスク間のファイルコピー(fmFileCopy).....	43
8.10	イーサネット通信コマンド	43
8.11	Linuxシステムコールサポート-システム制御.....	44

8.11.1	Linuxコマンドのキャンセル(<i>linux_sigcan</i>).....	44
8.11.2	時刻の設定(<i>linux_setiptime</i>).....	45
8.11.3	時刻の取得(<i>linux_getiptime</i>).....	46
8.12	Linuxシステムコールサポートファイル.....	47
8.12.1	オープン(<i>linux_open</i>).....	47
8.12.2	クローズ(<i>linux_close</i>).....	49
8.12.3	リード(<i>linux_read</i>).....	50
8.12.4	ライト(<i>linux_write</i>).....	51
8.12.5	シーク(<i>linux_lseek</i>).....	52
8.12.6	リンク(<i>linux_link</i>).....	53
8.12.7	シンボリックリンク(<i>linux_symlink</i>).....	54
8.12.8	アンリンク(<i>linux_unlink</i>).....	55
8.12.9	ファイル状態の取得(<i>linux_stat</i>).....	56
8.12.10	ファイル状態の取得(<i>linux_lstat</i>).....	58
8.12.11	ファイルモード変更(<i>linux_chmod</i>).....	60
8.12.12	ファイル所有者の変更(<i>linux_chown</i>).....	61
8.12.13	ディレクトリオープン(<i>linux_opendir</i>).....	62
8.12.14	ディレクトリクローズ(<i>linux_closedir</i>).....	63
8.12.15	ディレクトリリード(<i>linux_readdir</i>).....	64
8.12.16	ファイル名変更(<i>linux_rename</i>).....	65
8.12.17	ファイルコピー(<i>linux_copy</i>).....	66
8.12.18	ディレクトリ作成(<i>linux_mkdir</i>).....	67
8.12.19	ディレクトリ削除(<i>linux_rmdir</i>).....	68
8.12.20	作業ディレクトリの取得(<i>linux_getcwd</i>).....	69
8.12.21	作業ディレクトリの変更(<i>linux_chdir</i>).....	70
8.13	Linuxシステムコールサポートソケット.....	71
8.13.1	ソケットオープン(<i>linux_socket</i>).....	71
8.13.2	ソケットバインド(<i>linux_bind</i>).....	72
8.13.3	ソケットリッスン(<i>linux_listen</i>).....	73
8.13.4	ソケットアクセプト(<i>linux_accept</i>).....	74
8.13.5	ソケットコネクト(<i>linux_connect</i>).....	75
8.13.6	接続相手の名前を取得(<i>linux_getpeername</i>).....	76
8.13.7	ソケット名を取得(<i>linux_getsockname</i>).....	77
8.13.8	ソケットオプションの取得(<i>linux_getsockopt</i>).....	78
8.13.9	ソケットオプションの設定(<i>linux_setsockopt</i>).....	79
8.13.10	データ受信(<i>linux_recv</i>).....	80
8.13.11	データ受信と送信元アドレス取得(<i>linux_recvfrom</i>).....	81
8.13.12	データ送信(<i>linux_send</i>).....	83
8.13.13	データ送信と送信先アドレス設定(<i>linux_sendto</i>).....	84
8.13.14	コネクションの切断(<i>linux_shutdown</i>).....	86
8.13.15	複数ソケットの監視(<i>linux_select</i>).....	87
9.	エラーコード.....	88
9.1	Linux版エラー.....	88
付録A	変更履歴.....	89
付録B	従来SDK(SH動作)変更手順.....	90
B.1	従来SDK(SH動作)書換え時のSW設定.....	90
B.2	ネットワーク設定.....	90
B.3	従来SDK(SH動作)フォーマット手順.....	91
B.4	従来SDK(SH動作)実行時のSW設定.....	92
付録C	トラブルシューティング.....	93

図・表・リスト目次

図1-1 画像処理ユニットNVP-Ax230CL	2
図1-2 画像処理ユニットNVP-Ax235CL	3
図1-3 Linuxアプリケーション 開発環境の例	4
図2-1 Linuxシステム起動時の処理フロー	5
図2-2 Linuxシステム概要	6
図3-1 主要なCD-R内容	8
図3-2 フォルダ構成	9
図4-1 LinuxFormatツール①	17
図5-1 チュートリアル環境	18
図5-2 NAS-HDD接続環境	26
図5-3 LinuxFormatツール②(cramfs)	29
図B-1 LinuxFormatツール③(従来SDK)	91

表1-1 Linux開発環境	1
表1-2 NVP-Ax230 for Linux環境	1
表2-1 ハードウェア資源とアクセス方法	7
表3-1 提供ファイル一覧	10
表3-2 ヘッダファイル一覧	10
表3-3 共有ライブラリ一覧	10
表3-4 デーモン一覧	11
表3-5 サンプルファイル一覧	11
表3-6 Linuxツール一覧	11
表4-1 VirtualBox for Windows hostsのダウンロード	12
表4-2 VirtualBox for Windows hostsのダウンロード	12
表4-3 Sourcery G++ LiteのダウンロードURL	15
表4-4 Sourcery G++ Liteのダウンロードファイル	15
表4-5 Tera Termのダウンロード	15
表4-6 Linux書換え時のSW設定	16
表5-1 Linux起動時のSW設定	18
表5-2 シリアル設定	19
表5-3 ルートユーザ	20
表5-4 ゲストユーザ	21
表5-5 チュートリアルで設定した共有フォルダ纏め	22
表5-6 動作確認済NAS-HDD	26
表5-7 cramfsイメージの提供	27
表5-8 Linux書換え時のSW設定	28
表6-1 環境変数設定一覧	31
表6-2 環境設定ファイル(例)	31
表9-1 Linux版エラー	88
表B-1 従来SDK(SH動作)書換え時のSW設定	90
表B-2 従来SDK(SH動作)実行時のSW設定	92
表C-1 ブートモード時のSW設定	93
表C-2 Linux書換え時のSW設定(+ブートモード)	93

リスト4-1 /etc/rc.local 修正	14
リスト5-1 hello.c	22

リスト5-2 \$HOME/rootfs/cramfs/etc/passwd	30
リスト5-3 \$HOME/rootfs/cramfs/etc/shadow	30
リスト6-1 \$HOME/rootfs/cramfs/etc/init.d/rcS	32
リスト6-2 sample1/Makefile	34

1. 製品概要

1.1 NVP-Ax230SDK for Linux の概要

本 SDK は NVP-Ax230 シリーズのソフトウェア開発キット(NVP-Ax230SDK)のオプション製品で、Linux 環境で NVP-Ax230 シリーズのアプリケーション開発するために必要なライブラリ、ツール、ドキュメントを提供します。

1.1.1 特徴

- ・ Linux 環境 (ARM) でのアプリケーション構築が可能です。
- ・ μ TRON 環境 (SH) でのアプリケーション構築が可能です。
- ・ スタンドアロンシステムのアプリケーション開発が可能です。
- ・ 従来の画像認識アプリケーションが C ソースレベルで容易に移植可能です。

1.1.2 前提条件

本 SDK を利用し、Linux 環境でのアプリケーション開発を行うことができます。

表1-1 Linux開発環境

項目		内容
ハードウェア	PC	Windows が動作するパソコン 10/100Base T/TX × 1 ポート以上
	NVP-Ax230	NVP-Ax230CL NVP-Ax235CL
OS		Microsoft Windows7 32bit 版 (ServicePack1 以上) (64Bit 版は不可)
VM		ORACLE VM VirtualBox 4.2.10r84104 Ubuntu 12.04
クロスコンパイラ		Sourcery G++ Lite 2010q1-202
ソフトウェア開発キット		NVP-Ax230SDK + NVP-Ax230SDK for Linux

表1-2 NVP-Ax230 for Linux環境

項目	内容
ベース Linux	u-boot 2011.03 Linux 2.6.35.9
ルートファイルシステム	cramfs (MontaVista Linux 6 (BusyBox を含む))
NVP-Ax230SDK ライブラリ	libimp.so

1.2 画像処理ユニット NVP-Ax230 の概要

1.2.1 NVP-Ax230CL のハードウェア構成

画像ユニット NVP-Ax230CL は次の図が示す様に

- ・ CPU
- ・ 画像処理プロセッサ
- ・ 画像メモリ及びシステムメモリ（UMA 方式）（2GB）
- ・ フラッシュメモリ
- ・ 映像入力（カメラリンク入力 2CH）
- ・ 映像出力部
- ・ アイサレーション I/O ボード
- ・ イーサネット（100Base-T）
- ・ SCI（2CH + カメラリンクデータ通信用 1CH）

から構成されます。

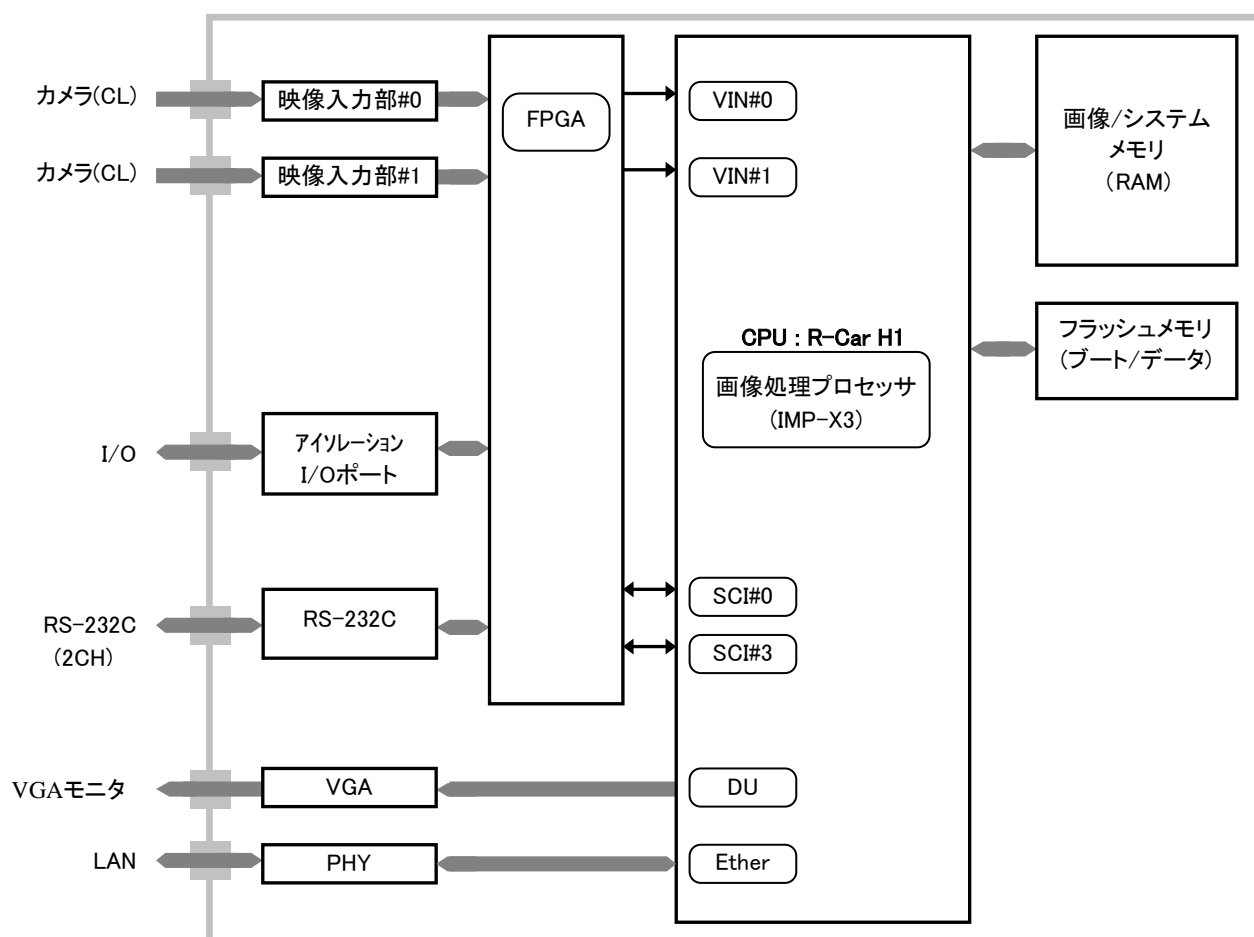


図1-1 画像処理ユニットNVP-Ax230CL

1.2.2 NVP-Ax235CL のハードウェア構成

画像ユニット NVP-Ax235CL は次の図が示す様に

- ・ CPU
- ・ 画像処理プロセッサ
- ・ 画像メモリ及びシステムメモリ（UMA 方式）（2GB）
- ・ フラッシュメモリ
- ・ 映像入力（カメラリンク入力 4CH）
- ・ 映像出力部
- ・ アイサレーション I/O ボード
- ・ イーサネット（100Base-T）
- ・ RTC（バッテリーバックアップ）
- ・ SCI（2CH + カメラリンクデータ通信用 1CH）
- ・ USB ホスト（2CH）
- ・ SD メモリカード
- ・ SCI（2CH + カメラリンクデータ通信用 1CH）

から構成されます。

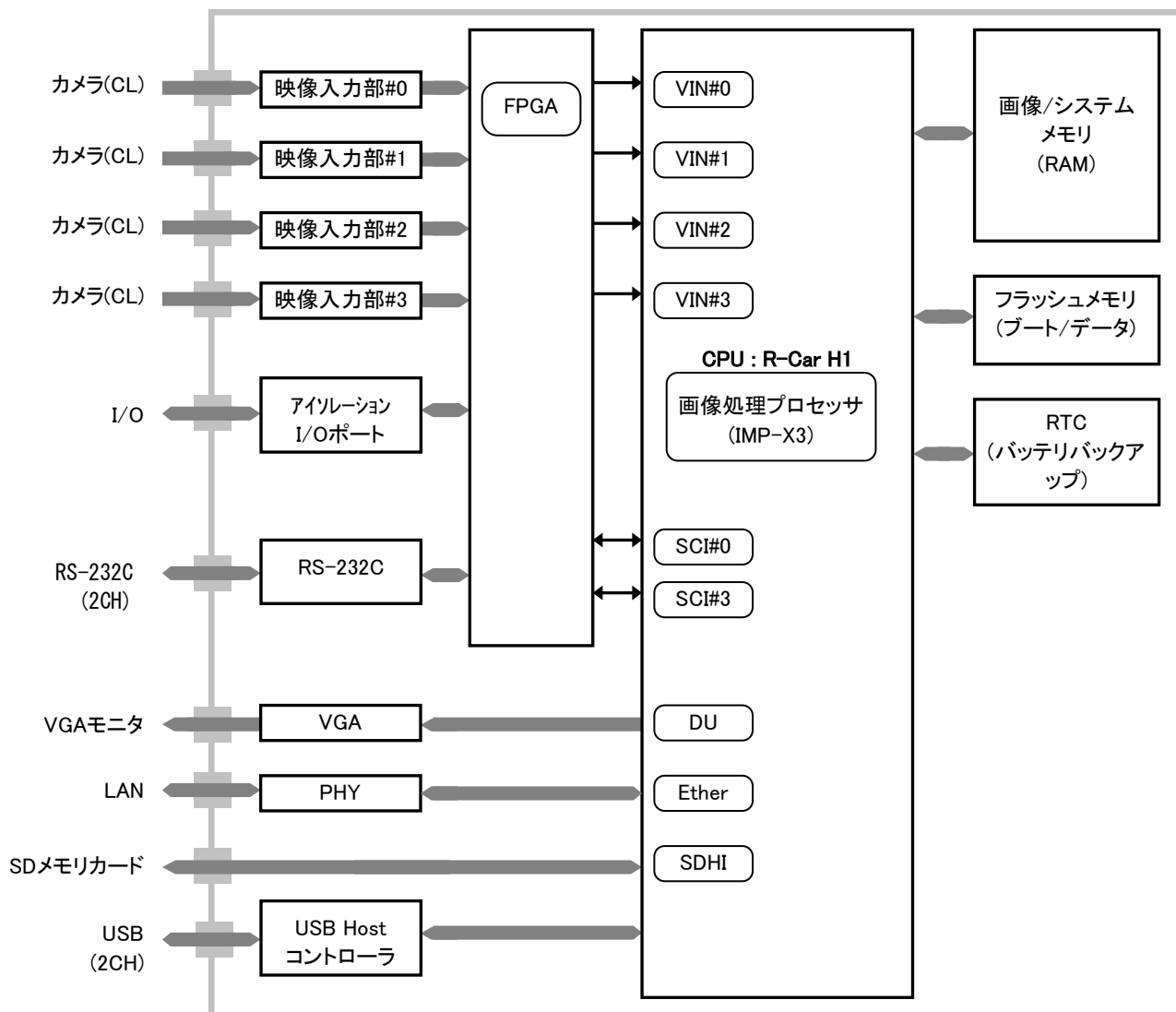


図1-2 画像処理ユニットNVP-Ax235CL

1.3 Linux アプリケーションの開発

Linux アプリケーション 開発環境の例を以下に示します。

以下の例では、Windows 共有フォルダを NVP-Ax230CL にてマウントし、Windows 上で開発したアプリケーション(実行形式ファイル)を実行します。システムコンソールは COM1(シリアル)です。telnet 接続も可能です。

この例では Windows 共有フォルダを利用していますが、NAS-HDD (Network Attached Storage)、NFS (Network File System)、SD メモリカード、USB メモリを使用しても同様の開発を行うことができます。

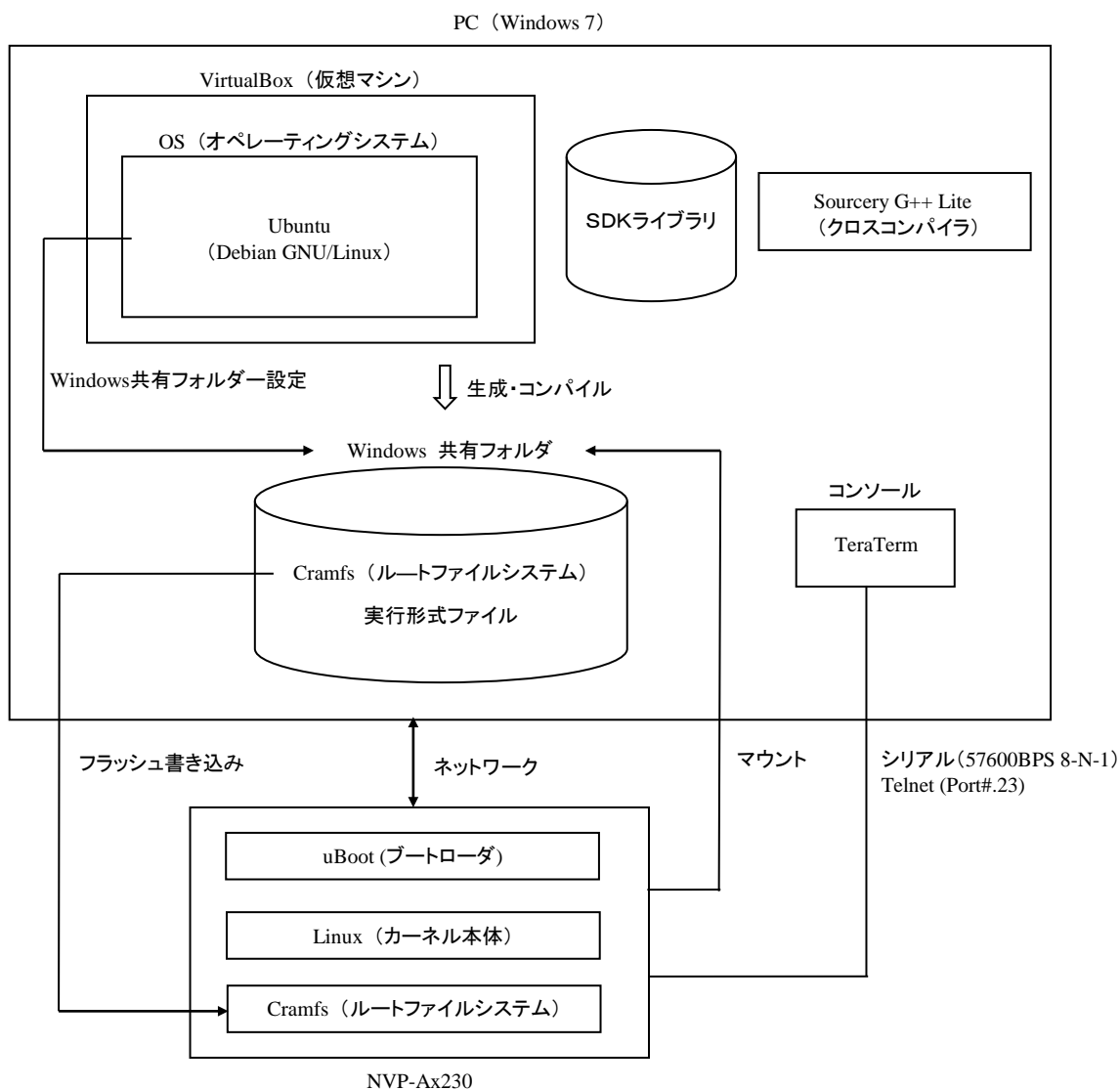
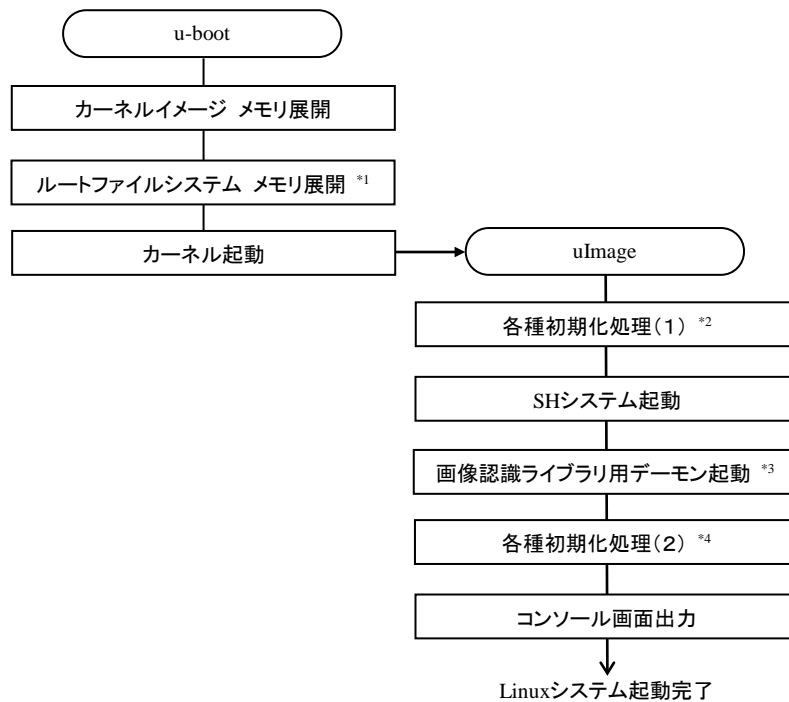


図1-3 Linuxアプリケーション 開発環境の例

2. Linux システム概要

2.1 Linux システム起動の処理フロー（デフォルト）

Linux システム起動時の処理フローの例を以下に示します。



*1 使用するルートファイルシステムやu-boot設定により、メモリ展開が必要ない場合があります。

*2 実行される初期化処理は、カーネル構築やルートファイルシステム設定により異なります。

*3 画像認識ライブラリ用デーモン起動するための設定が必要です。詳細は6.3章を参照ください。

*4 実行される初期化処理は、カーネル構築やルートファイルシステム設定により異なります。ユーザアプリケーションを自動起動する場合、SHシステム起動、画像認識用ライブラリ用デーモン起動が完了後にルートファイルシステム設定を行ってください。

図2-1 Linuxシステム起動時の処理フロー

2.2 Linux システム概要

Linux システム概要を以下に示します。

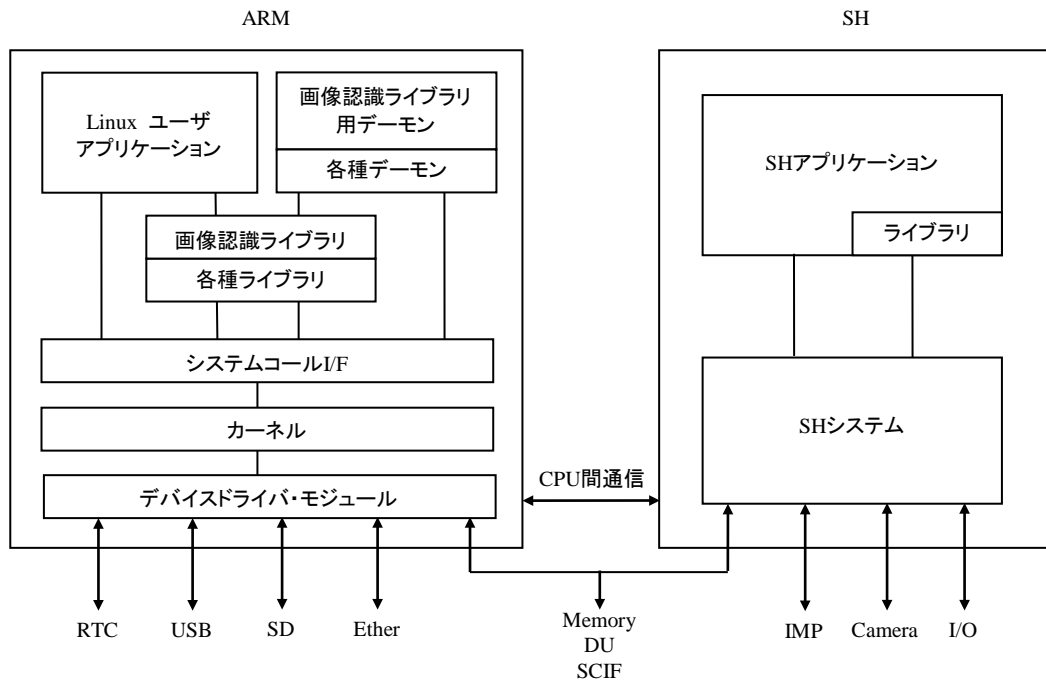


図2-2 Linuxシステム概要

2.3 ハードウェア資源とアクセス方法

ハードウェア資源とアクセス方法を以下に示します。

表2-1 ハードウェア資源とアクセス方法

項目		Linux アプリケーション	SH アプリケーション	備考
メモリ	Flash	標準デバイスドライバ /dev/mtdblock0~5	アクセス可	
	Fdisk	libimp.so(fm コマンド)	fmdisk2.lib(fm コマンド)	
	画像メモリ	libimp.so(AllocImg 等)	ipxcmd2.lib(AllocImg 等)	
	SH アプリ領域	libimp.so(LoadIPDLM 等)	-	
映像入力(Camera)		libimp.so(GetCamera 等)	ipxcmd2.lib(GetCamera 等)	
映像出力(DU)		標準デバイスドライバ /dev/fb0	ipxcmd2.lib(DispImg 等)	排他利用
画像認識(IMP)		libimp.so(各種コマンド)	ipxcmd2.lib(各種コマンド)	
USB		Linux ファイル API /dev/sda1,sdb1,sdc1,sdd1	linuxcmd2.lib(linux_open 等)	
SD カード		Linux ファイル API /dev/mmcblk0p1	linuxcmd2.lib(linux_open 等)	
Ether		Linux ソケット API	linuxcmd2.lib(linux_socket 等)	
SCIF		COM1(システムコンソール)	COM2 ipxctl2.lib(scicom_setup 等)	
RTC		標準デバイスドライバ /dev/rtc0	linuxcmd2.lib(linux_gettime 等)	
I/O		libimp.so(ipport コマンド)	ipxctl2.lib(ipport コマンド)	

3. パッケージ内容

3.1 CD-R

提供する CD-R には以下の内容が含まれます。

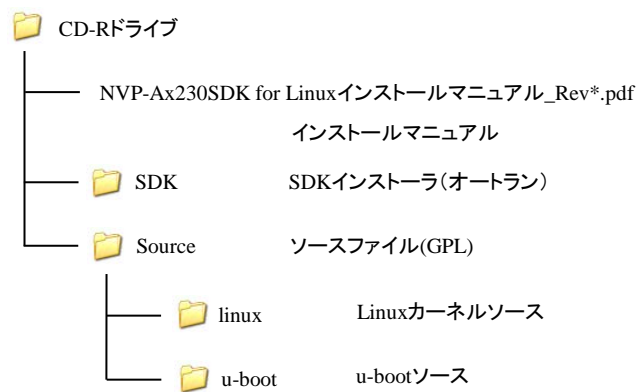


図3-1 主要なCD-R内容

3.2 インストール手順

- (1) 事前に NVP-Ax230SDK をインストールします。
- (2) NVP-Ax230SDK for Linux をインストールします。
(詳細は NVP-Ax230SDK for Linux インストールマニュアルを参照ください)

3.3 パッケージ内容

NVP-Ax230SDK と NVP-Ax230SDK for Linux をインストールすることにより以下の内容が作成されます。

(1) フォルダ構成

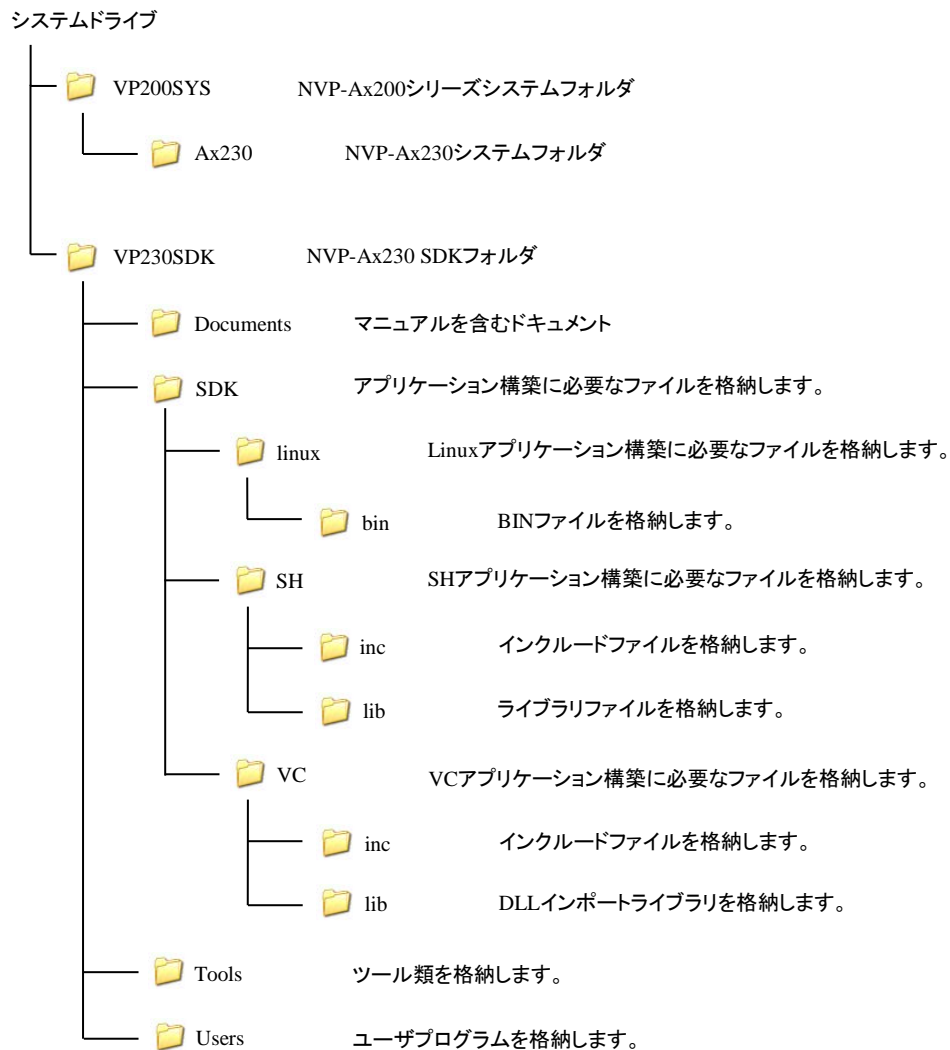


図3-2 フォルダ構成

(2) ファイル構成

本SDKで提供される Linux アプリケーション構築に必要なファイルを以下に示します。SH アプリケーション構築等に関する詳細は「NVP-Ax230SDK ユーザーズマニュアル」を参照ください。

表3-1 提供ファイル一覧

ファイル名		内容
VP230SDK¥SDK¥linux	NVP-Ax230SDK_vVRR.tar.gz	NVP-Ax230SDK for Linux 一式
VP230SDK¥SDK¥linux¥bin	u-boot.bin	bin ファイル ブートローダーイメージ
	uImage.bin	カーネルイメージ
	cramfs.arm.img.bin	cramfs イメージ
	ax230cl_shboot.bin	SH ブートローダーイメージ
	ax2Lcore.bin	SH システムイメージ

VRR : SDK バージョン

NVP-Ax230SDK_vVRR.tar.gz ファイルは tar zxvf <tar.gz ファイル>コマンド実行により展開されます。以下、本マニュアルでは/media/net(共有フォルダ)にて展開することを前提に説明します。

3.3.1 ヘッドファイル

提供する Linux 用ヘッドファイルを以下に示します。

表3-2 ヘッドファイル一覧

ファイル名		内容
/media/net/VP230SDK/inc	cnvcmd.h	画像認識ライブラリ用ヘッドファイル
	ipxdef.h	画像認識ライブラリ用ヘッドファイル
	ipxprot.h	画像認識ライブラリ用ヘッドファイル
	ipxsys.h	画像認識ライブラリ用ヘッドファイル
	vp900.h	画像認識ライブラリ用ヘッドファイル

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

3.3.2 共有ライブラリファイル

提供する Linux 用共有ライブラリを以下に示します。

表3-3 共有ライブラリー一覧

ファイル名		内容
/media/net/VP230SDK/lib	libimp.so	画像認識ライブラリ用ライブラリファイル
	libimp.so.v.r	画像認識ライブラリ用ライブラリファイル

.v.r はライブラリバージョンを示します。

libimp.so は通常、libimp.so.v.r のシンボリックリンクファイルですが、共有フォルダではシンボリックリンクファイルが作成できないため、libimp.so.v.r のコピーです。

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

3.3.3 デーモン

提供する Linux 用デーモンを以下に示します。

表3-4 デーモン一覧

ファイル名		内容
/media/net/VP230SDK/demon	impcmd	画像認識ライブラリ用デーモン

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

3.3.4 サンプルプログラム

提供する Linux 用サンプルプログラムを以下に示します。

表3-5 サンプルファイル一覧

ファイル名		内容
/media/net/VP230SDK/sample	sample1/Makefile	サンプルファイル 1 (デバイス ID 無版)
	sample1/src/main.c	
	sample2/Makefile	サンプルファイル 2 (デバイス ID 有版)
	sample2/src/main.c	

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

3.3.5 Linux ツール

提供する Linux 用ツールを以下に示します。

表3-6 Linuxツール一覧

ファイル名		内容
/media/net/VP230SDK/tools	readme.txt	ツール説明書
	shreset	SH システムリセットコマンド
	xcom	SH システム shell コマンド
	xterm	SH アプリ用 X-Term コマンド

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

4. 開発環境の準備

4.1 VirtualBox のインストール

下記、URL より”VirtualBox for Windows hosts”をダウンロードし、デフォルト設定のままインストールしてください。

表4-1 VirtualBox for Windows hostsのダウンロード

URL	ファイル名
https://www.virtualbox.org/wiki/Downloads	VirtualBox-****-Win.exe (最新)
https://www.virtualbox.org/wiki/Download_Old_Builds_4_2	VirtualBox-4.2.10-84105-Win.exe

※本マニュアルはバージョン 4.2.10 にて以下の説明を行います。

※VirtualBox 最新版は上記 URL からダウンロードできます。ファイル名については HP 確認ください。

4.2 Ubuntu のインストール

下記、URL より”Ubuntu 12.04”の VirtualBox イメージをダウンロードし、ダウンロードした zip ファイルから仮想ハードディスクイメージを展開します。

表4-2 VirtualBox for Windows hostsのダウンロード

URL	ファイル名
https://www.ubuntulinux.jp/download/ja-remix-vhd	ubuntu-ja-12.04-desktop-i386-vhd.zip

4.3 Ubuntu12.04 を VirtualBox に登録

- (1) VirtualBox を起動して、[仮想マシン]-[新規]メニューを選択すると[仮想マシンの作成] ダイアログが表示されます。「名前とオペレーティングシステム」を設定します。下記、設定してから[次に]ボタンをクリックします。

名前	: NVP-Ax230 <任意>
タイプ	: Linux
バージョン	: Ubuntu

- (2) 「メモリーサイズ」を設定します。下記、設定してから[次に]ボタンをクリックします。

メモリーサイズ	: <任意>
---------	--------

- (3) 「ハードドライブ」を設定します。下記、設定してから[作成]ボタンをクリックします。

[既にある仮想ハードドライブファイルを使用する]ラジオボタンを選択
ファイル名に4.2章で取得・展開した vhd ファイルを指定

4.4 VirtualBox 設定

- (1) VirtualBox を起動して、[仮想マシン]-[設定]メニューを選択すると[設定] ダイアログが表示されます。左側の設定リストから[一般]をクリックして、[高度]タブを選択して下記の設定を行います。

クリップボードの共有	: 双方向
------------	-------

- (2) [設定]ダイアログ 左側の設定リストから[共有フォルダー]をクリックします。[フォルダーリスト]を右クリックすると[共有フォルダーの追加]ダイアログが表示されますので、下記設定を行い[OK]ボタンをクリックしてください。

フォルダーのパス	: C:\¥share ^(*)
フォルダー名	: share ^(*)
[読み取り専用]チェックボタン	OFF
[自動マウント]チェックボタン	ON

(*) Windows フォルダ（任意）。事前に本フォルダを作成してください。指定フォルダはゲスト OS から R/W できる様に設定します。

(*) Ubuntu マウント時のディレクトリ名称（ /mnt/share ）

4.5 Ubuntu 設定

(1) VirtualBox の[仮想マシン]-[起動]メニューを選択すると Ubuntu が起動され初期設定が開始されます。

- ① [ようこそ]ダイアログ : 言語選択
- ② [どこに住んでいますか?]ダイアログ : 現在地選択
- ③ [キーボードレイアウト]ダイアログ : キーボード選択
- ④ [あなたの情報を入力してください]ダイアログ
 - : あなたの名前 renesas <任意>
 - コンピュータの名前 nvp-ax230 <任意>
 - ユーザー名の入力 renesas <任意>
 - パスワードの入力 ***** <任意>
 - パスワードの確認 ***** <任意>

(2) Ubuntu 12.04 アップデートを行います。アップデートマネージャを起動して、すべてのアップデートを行い再起動します。

- ① VirtualBox の左側(Lancha)から[Dash ホーム]を選択して、“app”で検索してください。アップデートマネージャが見つかります。

(3) “Guest Additions”のインストールを行います。

- ① VirtualBox の[デバイス]-[Guest Additions のインストール]メニューを選択すると自動実行されます。インストール完了後、再起動します。

(4) Ubuntu にて共有フォルダ 自動マウントの設定を行います。

- ① VirtualBox の左側(Lancha)から[Dash ホーム]を選択して、“term”で検索してください。端末が見つかります。端末を起動してください。

```
ubuntu $ sudo mkdir /mnt/share
ubuntu $ sudo vi /etc/rc.local
```

… 編集内容はリスト 4-1 参照

※ sudo は root 権限で実行するコマンドです(4.5章(1))にて設定したパスワード入力が必要です。
※ 上記例では vi エディタを利用していますが、他エディタ(gedit 等)でも可能です。

リスト4-1 /etc/rc.local 修正

```
mount -t vboxsf share /mnt/share          ## 追加

exit 0
```

(4) Ubuntu に work ディレクトリを作成します。

```
ubuntu $ mkdir $HOME/work
```

4.6 Sourcery G++ Lite のインストール

- (1) 下記、URL にアクセスして、ARM Processors の” Download the GNU/Linux Release”をクリックします。

表4-3 Sourcery G++ LiteのダウンロードURL

URL
http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/

- (2) E-mail 等入力して[Get Lite]ボタンをクリックすると、メール配信されます。配信されたメールに記載される URL にアクセスします。
- (3) ダウンロードリストから “Sourcery G++ Lite 2010q1-202”を選択します。
- (4) “IA32 GNU/Linux TAR”を選択するとダウンロードが開始されます。ダウンロード完了後、c:\¥share フォルダに格納します。(4.4章(2)指定のフォルダ)

表4-4 Sourcery G++ Liteのダウンロードファイル

ファイル名
arm-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

- (5) Ubuntu にてインストールを行います。インストールが完了すると \$HOME/work/tools 下に arm-2010q1 ディレクトリが作成されます。

```
ubuntu $ cd $HOME/work
ubuntu $ mkdir tools
ubuntu $ cd tools
ubuntu $ tar jxvf /mnt/share/arm-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-
gnu.tar.bz2
```

4.7 Tera Term のインストール

下記、URL より” Tera Term”をダウンロードできます(バージョン等は任意)。本アプリケーションはシリアル通信、および、Telnet を使うことを目的としています(相当品で代替可能です)。

表4-5 Tera Termのダウンロード

URL
http://sourceforge.jp/projects/ttssh2/

4.8 Ax230SDK ファームウェア書き換え

4.8.1 Linux 書換え時の SW 設定

Linux 書換え時の SW 設定を以下に示します。

表4-6 Linux書換え時のSW設定

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	予備		SW2	1	OFF	予備	
	2	OFF	予備			2	OFF	予備	
	3	OFF	予備			3	OFF	予備	
	4	OFF	予備			4	OFF	予備	
	5	OFF	予備			5	OFF	Linux 書き換え	
	6	OFF	予備			6	ON		
	7	OFF	予備			7	ON		
	8	OFF	予備			8	OFF	予備	

4.8.2 ネットワーク設定

Linux 書換え時、NVP-Ax230 の IP アドレスは 192.168.0.205(固定)に設定されております。接続する PC のネットワーク設定をあわせてください。

4.8.3 Linux フォーマット手順

NVP-Ax230 に電源投入すると NVP-Ax230 上の ERR-LED=点灯(赤)、RUN-LED=点滅(緑、500ms 間隔)となります。PC の[スタート]-[Ax230SDK]-[Tools]-[LinuxFormat]メニューを選択してください。LinuxFormat が起動します(図は LinuxFormat Version 1.1 以降)。

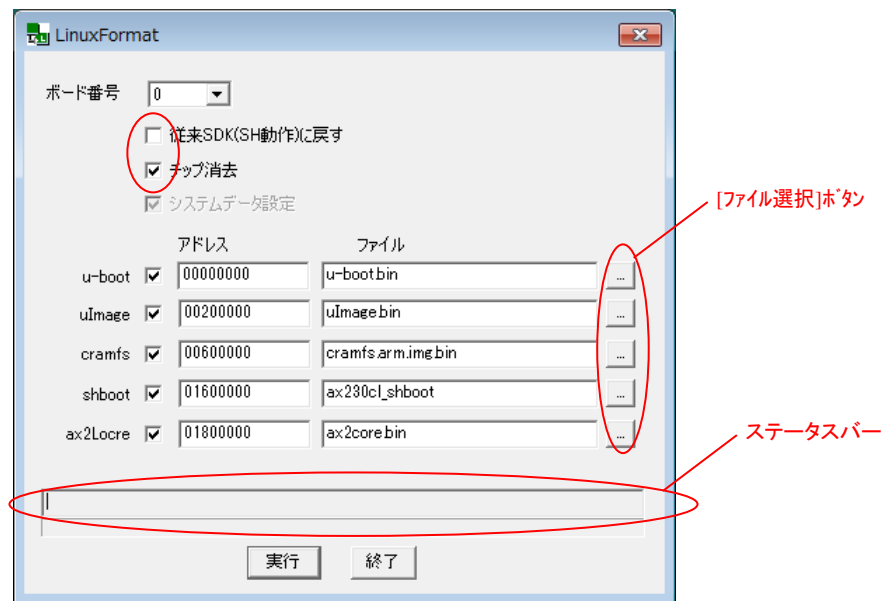


図4-1 LinuxFormatツール①

[従来 SDK(SH 動作)に戻す]チェックボタンを OFF、[チップ消去]チェックボックスを ON にしてください。全てのチェックボタンが自動的に ON になります。

[ファイル選択] ボタンをクリックすると[ファイルを開く]ダイアログが表示されますので、当該 BIN ファイルを選択してください([ファイル]テキストボックスに反映されます)。[アドレス]テキストファイルは初期値のままご使用ください (BIN ファイル詳細は表 3-1を参照ください)。

全項目を設定した後、[実行]ボタンをクリックすると RUN-LED=点滅(緑、100ms 間隔)になり、書き込みを開始します。書き換えは時間がかかります。実行中に電源を切らないでください。書き込み完了後、ステータスバーに”完了...”メッセージが表示されます。

2回目以降は必要な更新に対してのみ、チェックボタンで選択して、必要項目の書き換え可能です。

4.8.4 従来 SDK(SH 動作)変更手順

Linux フォーマット手順により、Linux 版が書き込まれた NVP-Ax230 を従来 SDK(SH 動作)に戻す方法については「付録 B 従来 SDK(SH 動作)変更手順」を参照ください

5. Linux チュートリアル

5.1 Linux 起動時の SW 設定

Linux 起動時の SW 設定を以下に示します。

表5-1 Linux起動時のSW設定

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	予備		SW2	1	OFF	予備	
	2	OFF	予備			2	OFF	予備	
	3	OFF	予備			3	OFF	予備	
	4	OFF	予備			4	OFF	予備	
	5	OFF	予備			5	ON	Linux 起動	
	6	OFF	予備			6	OFF		
	7	OFF	予備			7	OFF		
	8	OFF	予備			8	OFF	予備	

5.2 チュートリアル環境

チュートリアルにて説明する環境を以下に示します。シリアルケーブル、及び、LAN ケーブルにて、PC と NVP-Ax230 間を接続してください。

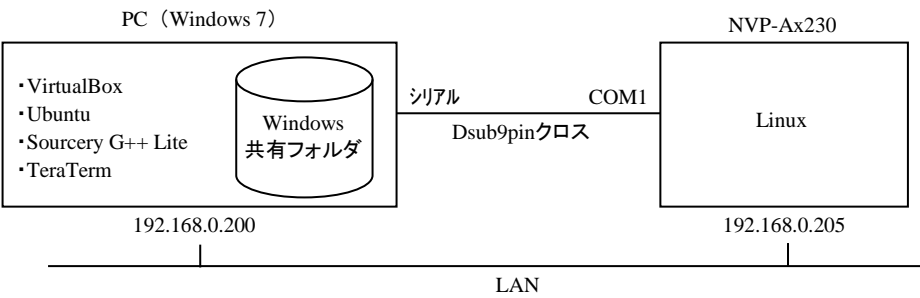


図5-1 チュートリアル環境

5.3 Tera Term の設定

Tera Term を起動して、[設定]-[シリアルポート]メニューを選択すると[シリアルポート設定]ダイアログが表示されます。以下の設定を行ってください。NVP-Ax230 のシステムコンソールとして利用されます。

表5-2 シリアル設定

項目	設定値	備考
ポート	任意	
ボーレート	57600 (BPS)	
データ	8 (bit)	
パリティ	none	
ストップ	1 (bit)	
フロー制御	none	

5.4 NVP-Ax230 の起動

NVP-Ax230 に電源投入すると NVP-Ax230 上の Linux システムが起動します。シリアルコンソール(Tera Term)に起動メッセージが表示されます。

```
U-Boot 2011.03 (Jun 23 2014 - 16:13:46)

CPU : R-CarH1 (md:0xa943)
      [CPU:1000MHz, SHwy:250MHz, DDR:500MHz, EXCLK:62.5MHz]
BOARD: AX320CL
DRAM: 2 GiB
Flash: 64 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:    serial
Err:    serial
WARNING: using static MAC address !!!!
Hit any key to stop autoboot: 0
## Booting kernel from Legacy Image at 40007fc0 ...
   Image Name:   Linux-2.6.35.9
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1979808 Bytes = 1.9 MiB
   Load Address: 40008000
   Entry Point:  40008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

:
:
```

5.5 ログイン

NVP-Ax230 上の Linux システムが起動完了すると、以下のメッセージが表示されます。ルートユーザとしてログインしてください。

```
MontaVista(R) Linux(R) 6.0

MontaVista Linux 6 .dev-snapshot-20120427 NVP-Ax230 ttySC0

NVP-Ax230 login:
```

表5-3 ルートユーザ

項目	設定	備考
ユーザ	root	
パスワード	rsptl	

5.6 ネットワークの設定

ifconfig コマンドにて、NVP-Ax230 の IP アドレスを設定します。

```
NVP-Ax230 $ ifconfig eth0 192.168.0.205
```

ping コマンドにて、PC とネットワーク接続できていることを確認します。

```
NVP-Ax230 $ ping 192.168.0.200
PING 192.168.0.200 (192.168.0.200): 56 data bytes
64 bytes from 192.168.0.200: seq=0 ttl=128 time=1.042 ms
64 bytes from 192.168.0.200: seq=1 ttl=128 time=0.483 ms
64 bytes from 192.168.0.200: seq=2 ttl=128 time=0.604 ms
64 bytes from 192.168.0.200: seq=3 ttl=128 time=0.515 ms
64 bytes from 192.168.0.200: seq=4 ttl=128 time=0.629 ms

--- 192.168.0.200 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.483/0.654/1.042 ms
NVP-Ax230 $
```

<CTRL+C> 入力

- ※ ネットワーク接続が確認できない場合、PC 側にて、Windows ファイアウォールの設定をご確認ください。またその他設定をご確認ください。
- ※ /etc/network/interfaces にて、auto eth0 に関する項目を記述することでシステム起動時にネットワーク設定することが可能です。

5.7 共有フォルダのマウント

mount コマンドにて、共有フォルダをマウントします。PC 側の指定フォルダは事前に Windows 共有の設定を行っておく必要があります。本設定例では PC 側の C:\share を Windows 共有設定にしてください。

```
NVP-Ax230 $ mount -t cifs -o noserverinfo,username=xxxx,password=xxxx,domain=xxxx  
//192.168.0.200/share /media/net/
```

※ username=xxxx, password=xxxx, domain=xxxx は接続する PC 設定をご確認ください。

5.8 telnet の実行

Tera Term を Windows コマンドプロンプトから実行します (telnet 接続)。

```
> ttermpro.exe 192.168.0.205 /P=23 /T=1
```

Tera Term を実行すると以下のログインメッセージが出力されます。guest ユーザとしてログインしてください。

```
MontaVista(R) Linux(R) 6.0  
Linux/armv7l 2.6.35.9  
  
MontaVista Linux 6 .dev-snapshot-20120427 NVP-Ax230  
  
NVP-Ax230 login:
```

表5-4 ゲストユーザ

項目	設定	備考
ユーザ	guest	
パスワード	nvpax230	

5.9 hello.c のコーディング

共有フォルダに hello.c を作成します。

リスト5-1 hello.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("hello world !!¥n");
}
```

hello.c の作成は、Windows 上の任意のエディタ、Ubuntu 上のエディタ、NVP-Ax230 上のエディタ(vi)、いずれでも問題ありません。ここで本マニュアルのチュートリアルにて設定した共有フォルダについて整理します。

表5-5 チュートリアルで設定した共有フォルダ纏め

OS	フォルダ名称	備考
Windows7	C:¥share	4.4章(2) (*1)
Ubuntu	/mnt/share	4.4章(2) 、4.5章(4)
NVP-Ax230	/media/net	5.7章

(*1) Windows 共有フォルダの設定は使用する PC の設定方法を確認ください。

5.10 hello.c のクロスコンパイル

hello.c のクロスコンパイルは ubuntu にて行います。クロスコンパイルにより、a.out（実行形式ファイル）が生成されます。

```
ubuntu $ cd /mnt/share
ubuntu $ ls
hello.c
ubuntu $ $HOME/work/tools/arm-2010q1/bin/arm-none-linux-gnueabi-gcc hello.c
ubuntu $ ls
a.out hello.c
```

5.11 a.out の実行

実行は NVP-Ax230 にて行います。

```
NVP-Ax230 $ cd /media/net
NVP-Ax230 $ ls
a.out hello.c
NVP-Ax230 $ ./a.out
hello world !!
```

5.12 他ストレージのマウント

Ubuntu で開発したプログラムを NVP-Ax230 で実行する方法として、共有フォルダ以外にも USB メモリ、SD カード、NAS-HDD を利用することができます。

5.12.1 USB メモリのマウント

USB メモリをスロット USB1、または、USB2 に挿入すると、システムコンソールに以下のようなメッセージが出力されます。網掛けで示す“角形括弧の文字列+1”がデバイス名となります。この場合、“sda1”がデバイス名です。

```
usb 5-2: new high speed USB device using r8a66597_hcd and address 2
scsi1 : usb-storage 5-2:1.0
usb 5-2: address 2, EndpointAddress 0x02 use DMA FIFO
usb 5-2: address 2, EndpointAddress 0x81 use DMA FIFO
scsi 1:0:0:0: Direct-Access    BUFFALO  ClipDrive          2.00 PQ: 0 ANSI: 2
sd 1:0:0:0: [sda] 503808 512-byte logical blocks: (257 MB/246 MiB)
sd 1:0:0:0: [sda] Write Protect is off
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Attached SCSI removable disk
```

mount コマンドにて USB メモリをマウントします。USB メモリを抜く場合、必ず、umount コマンドによりアンマウントしてください。

```
NVP-Ax230 $ mount -t vfat /dev/sda1 /media/usb1
NVP-Ax230 $ ls -l /media/usb1
:
NVP-Ax230 $ umount /media/usb1
```

5.12.2 SD カードのマウント

SD カードをアダプタに挿入すると、システムコンソールに以下の様なメッセージが出力されます。デバイス名は“mmcblk0p1” 固定です。

```
      :  
  
mmc0: new high speed SDHC card at address e624  
mmcblk0: mmc0:e624 SU32G 29.7 GiB  
mmcblk0: p1  
  
      :
```

SD カード挿入後、mount コマンドにて SD カードをマウントします。SD カードを抜く場合、必ず、umount コマンドによりアンマウントしてください。

```
NVP-Ax230 $ mount -t vfat /dev/mmcblk0p1 /media/sd  
NVP-Ax230 $ ls -l /media/sd  
      :  
  
NVP-Ax230 $ umount /media/sd
```


5.12.3 NAS-HDD のマウント

動作確認済の NAS-HDD を以下に示します。

表5-6 動作確認済NAS-HDD

メーカー	型名	備考
BUFFALO	LS-X1.OTLJ	LS-XL シリーズ
I-O DATA	HDL-CE1.OS	HDL-CE シリーズ

NAS-HDD 接続環境を以下に示します。なお、PC と NAS-HDD の接続は NAS-HDD の資料を参照ください。

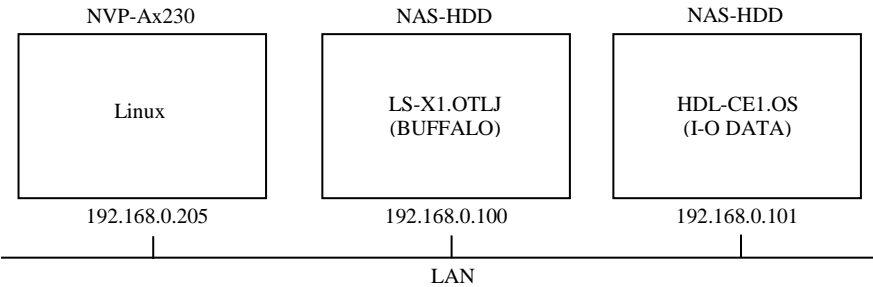


図5-2 NAS-HDD接続環境

(1) LS-X1.OTLJ (BUFFALO)

mount コマンドにて、NAS-HDD をマウントします。

```
NVP-Ax230 $ mount -t cifs -o noserverinfo,username=guest,domain=WORKGROUP  
//192.168.0.100/share /media/net1
```

※ username=guest, domain=WORKGROUP は接続する NAS-HDD 設定をご確認ください。

(2) HDL-CE1.OS (I-O DATA)

mount コマンドにて、NAS-HDD をマウントします。

```
NVP-Ax230 $ mount -t cifs -o noserverinfo,nounix,username=guest,domain=WORKGROUP  
//192.168.0.101/share /media/net2
```

※ username=guest, domain=WORKGROUP は接続する NAS-HDD 設定をご確認ください。

5.13 cramfs による各種設定

ルートファイルシステムには cramfs を採用します。cramfs は空間効率が高く主に組込みシステムで採用される Linux 用読み出し専用ファイルシステムです。

5.13.1 cramfs イメージの展開

- (1) cramfs イメージの提供
- 各種システム設定を行うため、cramfs イメージを提供します(表 3-1参照)。Ubuntu の共有フォルダ (/mnt/share)に格納ください。

表5-7 cramfsイメージの提供

ファイル名
cramfs.arm.img.bin

- (2) cramfs イメージの展開
- cramfsck コマンドにて、cramfs データを ubuntu の \$HOME/work/rootfs 下に展開します。

```
ubuntu $ mkdir $HOME/work/rootfs
ubuntu $ cd $HOME/work/rootfs
ubuntu $ sudo cramfsck -x cramfs /mnt/share/cramfs.arm.img.bin
```

- ※ sudo は root 権限で実行するコマンドです(4.5章(1)にて設定したパスワード入力が必要です)。
- ※ 共有フォルダ(Windows のファイルシステム)を展開先に指定できません。
- ※ 事前に”sudo apt-get install cramfsprogs”コマンドにてパッケージインストールが必要です。

5.13.2 各種システム設定の変更

展開された cramfs データを編集し、各種システム設定を変更します(システム設定詳細は5.13.6章の例を参照ください)。

5.13.3 cramfs イメージの作成

mkcramfs コマンドにて、cramfs イメージを ubuntu の共有フォルダ (/mnt/share)に作成します。

```
ubuntu $ cd $HOME/work/rootfs
ubuntu $ sudo mkcramfs cramfs /mnt/share/cramfs.arm.img.bin
```

- ※ sudo は root 権限で実行するコマンドです(4.5章(1)にて設定したパスワード入力が必要です)。
- ※ 事前に”sudo apt-get install cramfsprogs”コマンドにてパッケージインストールが必要です。

5.13.4 cramfs イメージの作成時の注意事項

cramfs イメージは最大で 16M バイトまで対応可能です。本制限を超えた場合、正常に Linux システムが動作しません。各種システム設定にて 16M バイトを超えない様に注意してください。

5.13.5 cramfs イメージの書き込み

- (1) Linux 書換え時の SW 設定
Linux 書換え時の SW 設定を以下に示します。

表5-8 Linux書換え時のSW設定

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	予備		SW2	1	OFF	予備	
	2	OFF	予備			2	OFF	予備	
	3	OFF	予備			3	OFF	予備	
	4	OFF	予備			4	OFF	予備	
	5	OFF	予備			5	OFF	Linux 書き換え	
	6	OFF	予備			6	ON		
	7	OFF	予備			7	ON		
	8	OFF	予備			8	OFF	予備	

- (2) ネットワークの設定
Linux 書換え時、NVP-Ax230 の IP アドレスは 192.168.0.205 (固定) に設定されています。接続する PC のネットワーク設定をあわせてください。

(3) cramfs 書き換え手順

NVP-Ax230 に電源投入すると NVP-Ax230 上の ERR-LED=点灯(赤)、RUN-LED=点滅(緑、500ms 間隔)となります。PC の[スタート]-[Ax230SDK]-[Tools]-[LinuxFormat]メニューを選択してください。LinuxFormat が起動します(図は LinuxFormat Version 1.1 以降)。

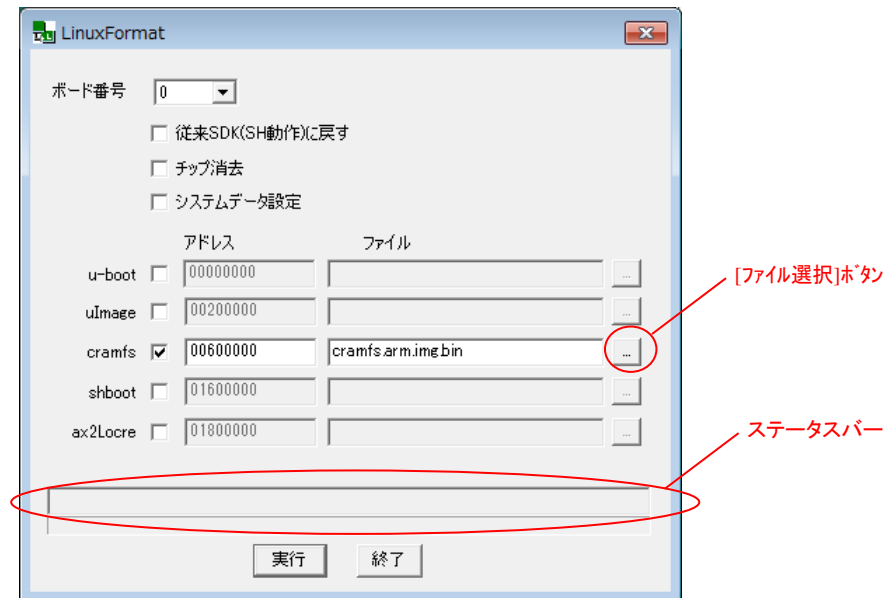


図5-3 LinuxFormatツール②(cramfs)

[cramfs]チェックボックスを ON にして、[ファイル選択] ボタンをクリックすると[ファイルを開く]ダイアログが表示されますので、当該 BIN ファイルを選択してください([ファイル]テキストボックスに反映されます)。[アドレス]テキストファイルは初期値のままご使用ください。(BIN ファイル詳細は表 3-1を参照ください)

[実行]ボタンをクリックすると RUN-LED=点滅(緑、100ms 間隔)になり、書き込みを開始します。書き換えは時間がかかります。実行中に電源を切らないでください。書き込み完了後、ステータスバーに”完了...”メッセージが表示されます。

5.13.6 cramfs によるシステム設定変更例

cramfs によるシステム設定変更例として、guest パスワード変更手順を示します。

(1) /etc/passwd の編集

/etc/passwd を編集し、guest ユーザの第 2 フィールドを x に変更します (シャドーパスワードを使用する)。本内容はデフォルト設定です。

```
ubuntu $ sudo vi $HOME/rootfs/cramfs/etc/passwd
```

※ sudo は root 権限で実行するコマンドです (4.5 章(1)にて設定したパスワード入力が必要です)。

※ 上記例では vi エディタを利用していますが、他エディタ (gedit 等) でも可能です。

リスト5-2 \$HOME/rootfs/cramfs/etc/passwd

```
:
gnats:!:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
guest:x:101:101:guest:/home/guest:/bin/sh
nobody:!:65534:65534:nobody:/nonexistent:/bin/sh
```

(2) シャドーパスワードの生成

mkpasswd コマンドにて、パスワードを生成します。例では MD5 のシャドーパスワードを生成しています。

```
ubuntu $ mkpasswd -S $(head -c 4 /dev/urandom | xxd -p) -m md5
パスワード: *****
$1$009812ad$xqixwBluKqFJQVkpScAJ6.
```

(3) /etc/shadow の編集

/etc/shadow を編集し、gueset ユーザを作成します。(2)で生成したパスワードを設定します。

リスト5-3 \$HOME/rootfs/cramfs/etc/shadow

```
root:$1$74ff5002$r9xoPpriNiq9zoLCCh9rE1:1518:0:99999:7:::
guest:$1$009812ad$xqixwBluKqFJQVkpScAJ6.:1518:0:99999:7:::
```

(4) cramfs イメージの作成

5.13.3 章参照

(5) cramfs イメージの書き込み

5.13.5 章参照

6. Linux 画像認識アプリケーションの開発

6.1 環境変数の設定

共有ライブラリを利用するための環境変数 LD_LIBRARY_PATH の設定を行います。

表6-1 環境変数設定一覧

環境変数設定		内容
LD_LIBRARY_PATH	/media/net/VP230SDK/lib	共有ライブラリ検索パス

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

設定コマンドは以下を以下に示します。

```
NVP-Ax230 $ LD_LIBRARY_PATH=/media/net/VP230SDK/lib
NVP-Ax230 $ export LD_LIBRARY_PATH
```

通常は使用するユーザのシェルの環境設定ファイル(\$HOME/.profile)にて設定します。ファイルがない場合、作成すると便利です。設定例として以下ファイルを提供します。

表6-2 環境設定ファイル(例)

ファイル名		内容
/media/net/VP230SDK/	profile	環境設定ファイル(例)

/media/net/ : 提供 tar.gz ファイルの展開フォルダを示します(ユーザ任意)

6.2 impcmd daemon の起動

NVP-Ax230 の Linux システム起動後、root ユーザにてログインして impcmd daemon を起動します。

```
NVP-Ax230 $ cd /media/net/VP230SDK/demon
NVP-Ax230 $ ./impcmd
```

システム起動時にデモン起動したい場合、/etc/init.d/に記載します。詳細は6.3章を参照ください。
impcmd デモン実行前に環境変数 LD_LIBRARY_PATH の設定が必要です。

6.3 impcmdd デーモン自動起動

impcmdd デーモンを毎回手動で起動する方法に代えて、システム起動時に自動起動することも可能です。以下、設定手順を説明します。

6.3.1 ライブラリとデーモンを cramfs 上に展開

cramfs 上の/var に/media/net/VP230SDK をディレクトリ毎コピーします。

```
ubuntu $ sudo cp -a /mnt/share/VP230SDK $HOME/work/rootfs/cramfs/var/
```

- ※ sudo は root 権限で実行するコマンドです(4.5章(1))にて設定したパスワード入力が必要です。
- ※ VP230SDK ディレクトリのコピー元(Ubuntu 上の/mnt/share/VP230SDK)は NVP-Ax230SDK.tar.gz ファイルが展開されているディレクトリです。
- ※ VP230SDK ディレクトリのコピー先(cramfs 上の /var)はユーザ任意です。

6.3.2 /etc/init.d/rcS スクリプトの修正

cramfs 上の/etc/init.d/rcS はシステム起動時に実行されるスクリプトです。/etc/init.d/rcS を編集し、ライブラリパス設定、および、impcmdd 起動する様に変更します。

```
ubuntu $ sudo vi $HOME/work/rootfs/cramfs/etc/init.d/rcS
```

- ※ sudo は root 権限で実行するコマンドです(4.5章(1))にて設定したパスワード入力が必要です。
- ※ 上記例では vi エディタを利用していますが、他エディタ(gedit 等)でも可能です。

リスト6-1 \$HOME/rootfs/cramfs/etc/init.d/rcS

```
#
# Limit cancellation
#
ulimit -n 8192

#
# NVP-Ax230SDK environment
#
LD_LIBRARY_PATH=/var/VP230SDK/lib
export LD_LIBRARY_PATH

#
# NVP-Ax230SDK impcmdd demon start
#
if [ -x /var/VP230SDK/demon/impcmdd ]
then
    /var/VP230SDK/demon/impcmdd
fi
```

- ※ ulimit -n xxxx により同時オープン可能なファイルディスクリプタの制限を変更できます(デフォルト 1024)。
同時オープン可能なファイルディスクリプタの制限数(yyyy)はユーザ任意です。
- ※ ライブラリパス、および、impcmdd の格納場所はユーザ任意です。
- ※ あらかじめ、本設定はコメントアウトした状態で記載されます。コメントアウトを消してご利用ください。

6.3.3 cramfs イメージの作成と書き込み

(1) cramfs イメージの作成

5.13.3章参照

(2) cramfs イメージの書き込み

5.13.5章参照

6.3.4 システム再起動と impcmd デーモン確認

システム再起動して impcmd デーモンが起動されていることを確認してください。

NVP-Ax230 \$ ps -ef grep impcmd									
root	1237		1	0	07:17	?		00:00:00	/var/VP230SDK/demon/impcmd
root	1655	1313	0	11:06	pts/1			00:00:00	grep impcmd

※ コマンド実行結果は表示例です。

6.4 Linux アプリケーションの開発と実行

6.4.1 サンプルプログラムの make ファイルについて

サンプルプログラム sample1 の make ファイルを確認します。必要に応じて修正ください (sample2 も同様)。

リスト6-2 sample1/Makefile

```
# オブジェクトディレクトリ
SRCDIR      = ./src
OBJDIR      = ./obj

# 最終ターゲット
TARGET      = sample1

# インクルードファイル
INC = /mnt/share/VP230SDK/inc ... ①

# ライブラリファイル
LIB = /mnt/share/VP230SDK/lib ... ②

# コンパイル条件等
CC = /home/renesas/work/tools/arm-2010q1/bin/arm-none-linux-gnueabi-gcc ... ③
CFLAGS = -I$(INC)
LDFLAGS = -L$(LIB)

OBJS      = $(OBJDIR)/main.o
INCS      = $(INC)/ipxdef.h $(INC)/ipxsys.h $(INC)/ipxprot.h $(INC)/cnvcmd.h

#####

# 最終ターゲット
all : $(TARGET)

# 共有ライブラリのビルド
$(TARGET) : $(OBJS) $(LIB)/libimp.so
    @echo "$(CC) $(LDFLAGS) -o $@"
    @$(CC) $(LDFLAGS) -o $@ $^ -lmp

# 各ファイルのビルド
$(OBJDIR)/main.o : $(SRCDIR)/main.c $(INCS) $(OBJDIR)
    $(CC) $(CFLAGS) -o $(OBJDIR)/main.o -c $(SRCDIR)/main.c

$(OBJDIR) :
    mkdir $(OBJDIR)

# 生成ファイルの削除
clean :
    rm $(OBJDIR)/*.o
    @if [ -f $(TARGETSOFI) ]; then rm -rf $(TARGETSOFI); fi
```

① /mnt/share は Ubuntu から見た共有フォルダ名です。展開したヘッダファイル格納フォルダを指定してください。

② /mnt/share は Ubuntu から見た共有フォルダ名です。展開した共有ライブラリ格納フォルダを指定してください。

③ クロスコンパイラのインストール先フォルダを指定してください。

6.4.2 サンプルプログラムの make と実行

サンプルプログラムの make は ubuntu にて行います。make 実行により、sample1 (実行形式ファイル) が生成され、実行できます (sample2 も同様)。

```
ubuntu $ cd /mnt/share/VP230SDK/sample/sample1
ubuntu $ ls
Makefile src
ubuntu $ make
ubuntu $ ls
Makefile obj sample1 src
```

```
NVP-Ax230 $ cd /media/net/VP230SDK/sample/sample1
NVP-Ax230 $ ls
Makefile obj sample1 src
NVP-Ax230 $ ./sample1
```

6.5 注意事項

- (1) Linux アプリケーションはプロセス単位でリモートコマンド使用開始 (StartIP, OpenIPDev)、リモートコマンド使用停止 (StopIP, CloseIPDev) を発行する必要があります。マルチスレッドの場合は同一プロセス内とみなされるので、いずれかひとつのスレッドでリモートコマンド使用開始 (StartIP, OpenIPDev) を発行することで全てのスレッドで画像認識ライブラリコマンドを利用することができます。
- (2) マルチプロセスにて Linux アプリケーション設計する際、リモートコマンド使用開始 (StartIP, OpenIPDev) を、発行してから、子プロセス作成 (fork) すると子プロセス側でリモートコマンド使用開始 (StartIP, OpenIPDev) がエラーとなります (親プロセスの情報が複製されるため、オープン済と判定される)。子プロセス作成 (fork) してからリモートコマンド使用開始 (StartIP, OpenIPDev) を発行してください。
- (3) Linux アプリケーションから、リモートコマンド使用開始 (StartIP, OpenIPDev) を発行した状態で <CTRL+C> 等でアプリケーション終了 (シグナル SIGINT) するとリモートコマンド使用停止 (StopIP, CloseIPDev) を発行されないままとなります (その場合、解放されないリソースが残ります)。Linux の signal システムコールでシグナル受信後の処理を記述し、確実にリモートコマンド使用停止 (StopIP, CloseIPDev) を発行する様にしてください。
- (4) マルチプロセスで Linux アプリケーション設計した場合、各プロセスが発行する画像認識ライブラリコマンドは並行して SH-4A にて受付、実行されますが、マルチスレッドで Linux アプリケーション設計した場合、各スレッドが発行する画像認識ライブラリコマンドは SH-4A にてシーケンシャルに受付・実行されます (画像認識ライブラリコマンドが実行完了するまで次の画像認識ライブラリコマンドの要求は実行されません)。
- (5) Linux システム起動時にアプリケーションを自動起動する場合、SH-4A 起動完了まで画像認識ライブラリコマンドの実行が行われません。リモートコマンド使用開始 (StartIP, OpenIPDev) を発行する前に 2 秒 wait してください。

7. SH 画像認識アプリケーションの開発

7.1 提供ファイル

従来通りです。

7.2 SH アプリケーションの開発と実行

従来通りです。

但し、USB、SD カード、ネットワークは Linux 側で管理するため、従来のシステムコールでそれらの資源を利用できません。新規サポートコマンドでのサポートとなります(8章参照)。

8. コマンドリファレンス

本 SDK にて提供する各種コマンドの詳細は NVP-Ax230SDK リファレンスマニュアルを参照ください。

本章では Linux 版にて、変更のあったコマンドについて変更箇所のみ記載します。

8.1 デバイス ID 無しリモートコマンド

8.1.1 デバイス ID 無しリモートコマンドの使用開始(StartIP)

☐ C 言語 API

```
int ret = StartIP( int BoardNo, int Opt );
```

☐ 機能

Linux 版では各種画像処理コマンド実行前にプロセス単位で StartIP を実行する必要があります。

8.1.2 デバイス ID 無しリモートコマンドの使用停止(StopIP)

☐ C 言語 API

```
int ret = StopIP( int BoardNo );
```

☐ 機能

Linux 版では各種画像処理コマンド実行後にプロセス単位で StopIP を実行する必要があります。

8.1.3 デバイス ID 付きリモートコマンド

8.1.4 デバイス ID 付きリモートコマンドの使用開始(OpenIPDev)

□ C 言語 API

```
int ret = OpenIPDev( int BoardNo, int Opt );
```

□ パラメータ

int	BoardNo	ボード番号(0~15)
int	Opt	オプション。通常は 0 を設定して下さい。
	OPENIP_OPT_RESET_AUTO(0)	ブートしていない場合リセットします。 (SH システムのリセットをしません)
	OPENIP_OPT_RESET_MANUAL(1)	マニュアルリセットします。 (SH システムをリセットします)
	OPENIP_OPT_RESET_POWERON(2)	パワーオンリセットします。 (SH システムをリセットします)
	OPENIP_OPT_RESET_DISABLE(8)	リセットしない。 (SH システムのリセットをしません)

□ 機能

Linux 版では各種画像処理コマンド実行前にプロセス単位で OpenIPDev を実行する必要があります。

8.2 API 初期化 API サポートコマンド

8.2.1 画像処理ボードのリセット(ResetIP)

□ C 言語 API

```
int ret = ResetIP( int BoardNo, int Mode );
```

□ パラメータ

int	BoardNo	ボード番号(0~15)
int	Mode	モード(0または4を指定して下さい)
	(0)	Linux ドライバリセット、SH システムのリセットなし
	(4)	Linux ドライバリセット、SH システムをリセット

8.3 システム制御コマンド

8.3.1 バージョン情報の取得(GetIPVersionInfo)

□ 機能

VERSIONINFO 構造体のメンバ詳細

メンバ名	内容	備考
IPCMD_Env	0 : ボードが動作していない 1 : ROM上のIPLが動作している 2 : RAMにダウンロードされたオンボードシステムが動作している	Linux 版は 2 と設定されます
BOARD_Info	品種 0x20047 : VP-Ax230CL 0x20041 : VP-Ax235CL 0x20147 : VP-Ax230CL-Linux 0x20141 : VP-Ax235CL-Linux	
IPCMD_Version	ax2Lcore 情報	フォーマット変更なし
IPCMD_Update	ax2Lcore 情報	フォーマット変更なし
ROM_Version	shboot 保持(0x016F:FFF0)	フォーマット変更なし
ROM_Update	shboot 保持(0x016F:FFF0)	フォーマット変更なし
PCDRV_Version	Linux Lib 情報	フォーマット変更なし
PCDRV_Update	Linux Lib 情報	フォーマット変更なし
DEVDRV_Version	Linux 版 0x01000000	フォーマット変更なし
DEVDRV_OS	3 : Windows WDM (デバイスドライバ) 4 : TCP DLL 5 : Linux	

8.4 映像表示コマンド

8.4.1 映像表示の初期化(du_init)

☐ C 言語 API

`void du_init(void);`

☐ パラメータ

なし

☐ 対応画面タイプ/画面データタイプ

なし

☐ リターンパラメータ

なし

☐ エラーコード

なし

☐ 機能

映像表示コマンドを従来 SDK 仕様にて使用するための初期化を行います。
初期化前に従来 SDK の映像表示コマンドを実行してもエラーとなりませんので注意してください。

本コマンド実行前は、Linux 側にて映像表示機能を独占します(/dev/fb0)。本コマンド実行後は Linux 側で映像表示機能を利用できなくなりますので注意してください。

8.5 構成制御

8.5.1 カメラ映像入力構成制御設定(SetConfigCamera)

□ パラメータ

enum CfgViewType type		映像入力種別
CFG_VP110_CAMERA	(0)	NVP-Ax230 からのカメラ映像入力 (デフォルト設定)
CFG_VFW_CAMERA	(1)	キャプチャデバイスからのカメラ映像入力
CFG_AVI_FILE	(2)	動画ファイルからのカメラ映像入力
CFG_BMP_LIST	(3)	BMP リストで指定されたファイルからのカメラ映像入力
CFG_NO_OUTPUT	(256)	カメラからの入力をスルー(実行しない)状態にする

□ 機能

Linux 版の場合、CFG_VFW_CAMERA、CFG_AVI_FILE、CFG_BMP_LIST を指定した場合、異常終了(エラーコード 20)となります。

8.5.2 カメラ映像入力プログラムの終了(ExitIPCamera)

□ リターンパラメータ

int ret 異常終了(-1)

□ エラーコード

138 WFWCam2.exe ツール未起動

□ 機能

Linux 版の場合、本関数は必ず異常終了(エラーコード 138)となります。

8.5.3 画像メモリ表示構成制御設定(SetConfigView)

□ パラメータ

enum CfgViewType type		画像メモリ表示種別
CFG_DISP_VP110	(0)	NVP-Ax230 RGB 出力画面に表示する(デフォルト設定)
CFG_DISP_VIEW	(1)	画面番号の画像データをパソコン上の他面に表示する。
CFG_NO_OUTPUT	(256)	画像メモリ表示処理をスルー(実行しない)状態にする

□ 機能

Linux 版の場合、CFG_DISP_VIEW を指定した場合、異常終了(エラーコード 20)となります。

8.5.4 画像メモリ表示プログラム終了(ExitIPView)

□ リターンパラメータ

int ret 異常終了(-1)

□ エラーコード

138 IPView2.exe ツール未起動

□ 機能

Linux 版の場合、本関数は必ず異常終了(エラーコード 138)となります。

8.6 S C I コントロール

チャンネル 0 は Linux 占有のため使用不可となります。チャンネル 1 のみ利用可能です。

8.7 タイマ

8.7.1 時刻の登録(SetIPTime)

□ 機能

本コマンドは SH のみ実装され Linux では実装されません。Linux では Linux 提供のシステムコールにて時刻設定してください。

8.7.2 時刻の取得(GetIPTime)

□ 機能

本コマンドは SH のみ実装され Linux では実装されません。Linux では Linux 提供のシステムコールにて時刻取得してください。

8.8 USB-HID 制御コマンド

USB-HID 制御コマンドは未サポートコマンドとなります。USB は Linux のデバイスファイルとして実装されます。

[対象]

- USB_HID_Start
- USB_HID_Terminate
- USB_HID_GetStatus
- USB_HID_SetReport
- USB_HID_Start

8.9 ファイルアクセス

8.9.1 PC-オンボードディスク間のファイルコピー(fmFileCopy)

本コマンドは未サポートコマンドとなります。

8.10 イーサネット通信コマンド

イーサネット通信コマンドは未サポートコマンドとなります。イーサネット通信は Linux のソケット API として実装されます。

[対象]

- socket
- bind
- listen
- accept
- connect
- getpeername
- getsockname
- recv
- recvfrom
- send
- sendto
- closesocket
- shutdown
- getsockopt
- setsockopt
- selectsocket
- set_blocking_socket
- get_errno
- get_thread_errno
- set_sock_timewait
- get_sock_rcvlen
- set_sock_keepalive

8.11 Linux システムコールサポート-システム制御

8.11.1 Linux コマンドのキャンセル(linux_sigcan)

□ C 言語 API

```
int ret = linux_sigcan(int dstid, int tid);
```

□ パラメータ

int	dstid	0 を指定してください
int	tid	タスク ID

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

-17	タスク ID 不正
-41	状態エラー

□ 機能

Linux システムコールを発行し完了待ちのタスクを指定します。linux_xxx 系コマンドの実行を中断し Waiting 状態が解除されます。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.11.2 時刻の設定(linux_setiptime)

□ C 言語 API

```
int ret = linux_setiptime(int dstid, IP_SYSTEM_TIME *time);
```

□ パラメータ

int	dstid	0 を指定してください
IP_SYSTEM_TIME	*time	時刻

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ パケットの構造

```
typedef struct {  
    short    year;           /* 西暦 */  
    short    month;          /* 月 */  
    short    day_of_week;    /* 曜日 */  
    short    day;            /* 日 */  
    short    hour;           /* 時(24 時間) */  
    short    minute;         /* 分 */  
    short    second;         /* 秒 */  
    short    milliseconds;   /* mS */  
} IP_SYSTEM_TIME;
```

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

Linux システム時刻を設定します。SH版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.11.3 時刻の取得(linux_getiptime)

☐ C 言語 API

```
int ret = linux_getiptime(int dstid, IP_SYSTEM_TIME *time);
```

☐ パラメータ

int	dstid	0 を指定してください
IP_SYSTEM_TIME	*time	時刻

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ パケットの構造

```
typedef struct {  
    short    year;           /* 西暦 */  
    short    month;          /* 月 */  
    short    day_of_week;    /* 曜日 */  
    short    day;            /* 日 */  
    short    hour;           /* 時(24 時間) */  
    short    minute;         /* 分 */  
    short    second;         /* 秒 */  
    short    milliseconds;   /* mS */  
} IP_SYSTEM_TIME;
```

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

Linux システム時刻を取得します。SH版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12 Linux システムコールサポート-ファイル

8.12.1 オープン(linux_open)

□ C 言語 API

```
int fd = linux_open(int dstid, const char *pathname, int flags, int mode);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*pathname	パス名
int	flags	フラグ
int	mode	モード

□ リターンパラメータ

int	fd	ファイルディスクリプタ (≥0) 異常終了 (-1)
-----	----	-------------------------------

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール open 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

flags 設定用

#define LINUX_O_ACCMODE	0x00000003
#define LINUX_O_RDONLY	0x00000000
#define LINUX_O_WRONLY	0x00000001
#define LINUX_O_RDWR	0x00000002
#define LINUX_O_CREAT	0x00000040
#define LINUX_O_EXCL	0x00000080
#define LINUX_O_NOCTTY	0x00000100
#define LINUX_O_TRUNC	0x00000200
#define LINUX_O_APPEND	0x00000400
#define LINUX_O_NONBLOCK	0x00000800
#define LINUX_O_DSYNC	0x00001000
#define LINUX_FASYNC	0x00002000

mode 設定用

#define LINUX_S_IFMT	0xF000
#define LINUX_S_IFSOCK	0xC000
#define LINUX_S_IFLNK	0xA000
#define LINUX_S_IFREG	0x8000
#define LINUX_S_IFBLK	0x6000
#define LINUX_S_IFDIR	0x4000
#define LINUX_S_IFCHR	0x2000
#define LINUX_S_IFIFO	0x1000
#define LINUX_S_ISUID	0x0800
#define LINUX_S_ISGID	0x0400
#define LINUX_S_ISVTX	0x0200
#define LINUX_S_IRWXU	0x01C0
#define LINUX_S_IRUSR	0x0100
#define LINUX_S_IWUSR	0x0080
#define LINUX_S_IXUSR	0x0040
#define LINUX_S_IRWXG	0x0038
#define LINUX_S_IRGRP	0x0020
#define LINUX_S_IWGRP	0x0010
#define LINUX_S_IXGRP	0x0008
#define LINUX_S_IRWXO	0x0007
#define LINUX_S_IROTH	0x0004
#define LINUX_S_IWOTH	0x0002
#define LINUX_S_IXOTH	0x0001

8.12.2 クローズ(`linux_close`)

☐ C 言語 API

```
int ret = linux_close(int dstid, int fd);
```

☐ パラメータ

int	dstid	0 を指定してください
int	fd	ファイルディスクリプタ

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール `errno` に準ずる

☐ 機能

詳細は Linux システムコール `close` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”`linux.h`”ヘッダファイルをインクルードしてください。

8.12.3 リード(`linux_read`)

□ C 言語 API

```
int ret = linux_read(int dstid, int fd, void *buf, size_t count);
```

□ パラメータ

int	dstid	0 を指定してください
int	fd	ファイルディスクリプタ
void	*buf	バッファ
size_t	count	バイト数

□ リターンパラメータ

int	ret	読み込みバイト数 (≥0) 異常終了 (-1)
-----	-----	----------------------------

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `read` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”`linux.h`”ヘッダファイルをインクルードしてください。

8.12.4 ライト(linux_write)

□ C 言語 API

```
int ret = linux_write(int dstid, int fd, void *buf, size_t count);
```

□ パラメータ

int	dstid	0 を指定してください
int	fd	ファイルディスクリプタ
void	*buf	バッファ
size_t	count	バイト数

□ リターンパラメータ

int	ret	書き込みバイト数 (≥0) 異常終了 (-1)
-----	-----	----------------------------

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール write 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.5 シーク (linux_lseek)

□ C 言語 API

```
int ret = linux_lseek(int dstid, int fd, int offset, int whence);
```

□ パラメータ

int	dstid	0 を指定してください
int	fd	ファイルディスクリプタ
void	offset	オフセット
size_t	whence	基準位置

□ リターンパラメータ

int	ret	ファイル先頭からの位置 (≥0) 異常終了 (-1)
-----	-----	-------------------------------

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール lseek 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

whence 設定用

#define LINUX SEEK_SET	0
#define LINUX SEEK_CUR	1
#define LINUX SEEK_END	2

8.12.6 リンク (linux_link)

☐ C 言語 API

```
int ret = linux_link(int dstid, const char *oldpath, const char *newpath);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*oldpath	古いパス名
const char	*newpath	新しいパス名

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール link 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.7 シンボリックリンク (linux_symlink)

☐ C 言語 API

```
int ret = linux_symlink(int dstid, const char *oldpath, const char *newpath);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*oldpath	古いパス名
const char	*newpath	新しいパス名

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール symlink 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.8 アンリンク (linux_unlink)

□ C 言語 API

```
int ret = linux_unlink(int dstid, const char *pathname);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*pathname	パス名

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール unlink 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.9 ファイル状態の取得(linux_stat)

□ C 言語 API

```
int ret = linux_stat(int dstid, const char *path, struct linux_stat *buf);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*path	パス名
struct linux_stat	*buf	ステータス

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ パケットの構造

```
struct linux_stat {  
    unsigned int  st_mode;    /* アクセス保護 */  
    unsigned int  st_nlink;   /* ハードリンクの数 */  
    unsigned int  st_uid;     /* 所有者のユーザ ID */  
    unsigned int  st_gid;     /* 所有者のグループ ID */  
    int           st_size;    /* サイズ */  
    int           st_atime;    /* 最終アクセス時刻 */  
    int           st_mtime;    /* 最終修正時刻 */  
    int           st_ctime;    /* 最終状態変更時刻 */  
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `stat` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

struct linux_stat st_mode 設定用

#define LINUX_S_IFMT	0xF000
#define LINUX_S_IFSOCK	0xC000
#define LINUX_S_IFLNK	0xA000
#define LINUX_S_IFREG	0x8000
#define LINUX_S_IFBLK	0x6000
#define LINUX_S_IFDIR	0x4000
#define LINUX_S_IFCHR	0x2000
#define LINUX_S_IFIFO	0x1000
#define LINUX_S_ISUID	0x0800
#define LINUX_S_ISGID	0x0400
#define LINUX_S_ISVTX	0x0200
#define LINUX_S_IRWXU	0x01C0
#define LINUX_S_IRUSR	0x0100
#define LINUX_S_IWUSR	0x0080
#define LINUX_S_IXUSR	0x0040
#define LINUX_S_IRWXG	0x0038
#define LINUX_S_IRGRP	0x0020
#define LINUX_S_IWGRP	0x0010
#define LINUX_S_IXGRP	0x0008
#define LINUX_S_IRWXO	0x0007
#define LINUX_S_IROTH	0x0004
#define LINUX_S_IWOTH	0x0002
#define LINUX_S_IXOTH	0x0001

8.12.10 ファイル状態の取得(linux_lstat)

□ C 言語 API

```
int ret = linux_lstat(int dstid, const char *path, struct linux_stat *buf);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*path	パス名
struct linux_stat	*buf	ステータス

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ パケットの構造

```
struct linux_stat {  
    unsigned int  st_mode; /* アクセス保護 */  
    unsigned int  st_nlink; /* ハードリンクの数 */  
    unsigned int  st_uid; /* 所有者のユーザ ID */  
    unsigned int  st_gid; /* 所有者のグループ ID */  
    int           st_size; /* サイズ */  
    int           st_atime; /* 最終アクセス時刻 */  
    int           st_mtime; /* 最終修正時刻 */  
    int           st_ctime; /* 最終状態変更時刻 */  
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `lstat` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

struct linux_stat st_mode 設定用

#define LINUX_S_IFMT	0xF000
#define LINUX_S_IFSOCK	0xC000
#define LINUX_S_IFLNK	0xA000
#define LINUX_S_IFREG	0x8000
#define LINUX_S_IFBLK	0x6000
#define LINUX_S_IFDIR	0x4000
#define LINUX_S_IFCHR	0x2000
#define LINUX_S_IFIFO	0x1000
#define LINUX_S_ISUID	0x0800
#define LINUX_S_ISGID	0x0400
#define LINUX_S_ISVTX	0x0200
#define LINUX_S_IRWXU	0x01C0
#define LINUX_S_IRUSR	0x0100
#define LINUX_S_IWUSR	0x0080
#define LINUX_S_IXUSR	0x0040
#define LINUX_S_IRWXG	0x0038
#define LINUX_S_IRGRP	0x0020
#define LINUX_S_IWGRP	0x0010
#define LINUX_S_IXGRP	0x0008
#define LINUX_S_IRWXO	0x0007
#define LINUX_S_IROTH	0x0004
#define LINUX_S_IWOTH	0x0002
#define LINUX_S_IXOTH	0x0001

8.12.11 ファイルモード変更(linux_chmod)

□ C 言語 API

```
int ret = linux_chmod(int dstid, const char *path, unsigned long mode);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*path	パス名
unsigned long	mode	モード

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール chmod 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

mode 設定用

#define LINUX_S_IFMT	0xF000
#define LINUX_S_IFSOCK	0xC000
#define LINUX_S_IFLNK	0xA000
#define LINUX_S_IFREG	0x8000
#define LINUX_S_IFBLK	0x6000
#define LINUX_S_IFDIR	0x4000
#define LINUX_S_IFCHR	0x2000
#define LINUX_S_IFIFO	0x1000
#define LINUX_S_ISUID	0x0800
#define LINUX_S_ISGID	0x0400
#define LINUX_S_ISVTX	0x0200
#define LINUX_S_IRWXU	0x01C0
#define LINUX_S_IRUSR	0x0100
#define LINUX_S_IWUSR	0x0080
#define LINUX_S_IXUSR	0x0040
#define LINUX_S_IRWXG	0x0038
#define LINUX_S_IRGRP	0x0020
#define LINUX_S_IWGRP	0x0010
#define LINUX_S_IXGRP	0x0008
#define LINUX_S_IRWXO	0x0007
#define LINUX_S_IROTH	0x0004
#define LINUX_S_IWOTH	0x0002
#define LINUX_S_IXOTH	0x0001

8.12.12 ファイル所有者の変更(linux_chown)

□ C 言語 API

```
int ret = linux_chown (int dstid, const char *path, unsigned long owner, unsigned long group);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*path	パス名
unsigned long	owner	所有者
unsigned long	group	グループ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール chown 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.13 ディレクトリオープン(linux_opendir)

☐ C 言語 API

```
int ret = linux_opendir (int dstid, const char *name);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*name	ディレクトリ

☐ リターンパラメータ

int	ret	ディレクトリハンドル(ディレクトリストリーム) (NULL 以外) 異常終了 (NULL)
-----	-----	--

☐ エラーコード

Linux システムコール `errno` に準ずる

☐ 機能

詳細は Linux システムコール `opendir` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.14 ディレクトリクローズ(`linux_closedir`)

□ C 言語 API

```
int ret = linux_closedir (int dstid, int dh);
```

□ パラメータ

int	dstid	0 を指定してください
int	dh	ディレクトリハンドル(ディレクトリストリーム)

□ リターンパラメータ

int	ret	ディレクトリストリーム (NULL 以外) 異常終了 (NULL)
-----	-----	--------------------------------------

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `closedir` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”`linux.h`”ヘッダファイルをインクルードしてください。

8.12.15 ディレクトリリード(linux_readdir)

□ C 言語 API

```
struct linux_dirent *ret = linux_closedir (int dstid, int dh, struct linux_dirent *ent);
```

□ パラメータ

int	dstid	0 を指定してください
int	dh	ディレクトリハンドル(ディレクトリストリーム)
struct linux_dirent *	ent	ディレクトリエントリ

□ リターンパラメータ

int	*ret	ディレクトリエントリ (NULL 以外)
		ディレクトリエントリ終端、または。異常終了 (NULL)

□ パケットの構造

```
struct linux_dirent {  
    unsigned char  d_type;          /* ファイル種別 */  
    char           d_name[256];     /* ファイル名 */  
}
```

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール readdir 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_dirent d_type 設定用

#define LINUX_DT_UNKNOWN	0
#define LINUX_DT_FIFO	1
#define LINUX_DT_CHR	2
#define LINUX_DT_DIR	4
#define LINUX_DT_BLK	6
#define LINUX_DT_REG	8
#define LINUX_DT_LNK	10
#define LINUX_DT SOCK	12

8.12.16 ファイル名変更(linux_rename)

☐ C 言語 API

```
int ret = linux_rename (int dstid, const char *oldpath , const char *newpath);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*oldpath	古いパス名
const char	*newpath	新しいパス名

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール rename 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.17 ファイルコピー(linux_copy)

☐ C 言語 API

```
int ret = linux_copy (int dstid, const char *src , const char *dst);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*src	ソースファイル名
const char	*dst	ディストファイル名

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール copy 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.18 ディレクトリ作成(linux_mkdir)

□ C 言語 API

```
int ret = linux_mkdir (int dstid, const char *pathname, unsigned long mode);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*pathname	パス名
unsigned long	mode	モード

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール mkdir 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

mode 設定用

#define LINUX_S_IFMT	0xF000
#define LINUX_S_IFSOCK	0xC000
#define LINUX_S_IFLNK	0xA000
#define LINUX_S_IFREG	0x8000
#define LINUX_S_IFBLK	0x6000
#define LINUX_S_IFDIR	0x4000
#define LINUX_S_IFCHR	0x2000
#define LINUX_S_IFIFO	0x1000
#define LINUX_S_ISUID	0x0800
#define LINUX_S_ISGID	0x0400
#define LINUX_S_ISVTX	0x0200
#define LINUX_S_IRWXU	0x01C0
#define LINUX_S_IRUSR	0x0100
#define LINUX_S_IWUSR	0x0080
#define LINUX_S_IXUSR	0x0040
#define LINUX_S_IRWXG	0x0038
#define LINUX_S_IRGRP	0x0020
#define LINUX_S_IWGRP	0x0010
#define LINUX_S_IXGRP	0x0008
#define LINUX_S_IRWXO	0x0007
#define LINUX_S_IROTH	0x0004
#define LINUX_S_IWOTH	0x0002
#define LINUX_S_IXOTH	0x0001

8.12.19 ディレクトリ削除(`linux_rmdir`)

☐ C 言語 API

```
int ret = linux_rmdir(int dstid, const char *pathname);
```

☐ パラメータ

int	dstid	0 を指定してください
const char	*pathname	パス名

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール `errno` に準ずる

☐ 機能

詳細は Linux システムコール `rmdir` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”`linux.h`”ヘッダファイルをインクルードしてください。

8.12.20 作業ディレクトリの取得(linux_getcwd)

☐ C 言語 API

```
char *ret = linux_getcwd(int dstid, char *buf, size_t len);
```

☐ パラメータ

int	dstid	0 を指定してください
char	*buf	バッファ
size_t	len	バッファ長

☐ リターンパラメータ

char	*ret	正常終了 (buf)
		異常終了 (NULL)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール getcwd 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本サービスコールで取得した作業ディレクトリは impcmd デーモンの作業ディレクトリを意味します。他タスクの作業ディレクトリからの参照・変更に影響するのでご注意ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.12.21 作業ディレクトリの変更(linux_chdir)

□ C 言語 API

```
int ret = linux_chdir(int dstid, const char *path);
```

□ パラメータ

int	dstid	0 を指定してください
const char	*path	パス名

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール chdir 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本サービスコールで取得した作業ディレクトリは impcmd デーモンの作業ディレクトリを意味します。他タスクの作業ディレクトリからの参照・変更に影響するのでご注意ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

8.13 Linux システムコールサポート-ソケット

8.13.1 ソケットオープン(linux_socket)

☐ C 言語 API

```
int sd = linux_socket(int dstid, int domain, int type, int protocol);
```

☐ パラメータ

int	dstid	0 を指定してください
int	domain	ドメイン
int	type	タイプ
int	protocol	プロトコル

☐ リターンパラメータ

int	sd	ソケットディスクリプタ (≥0) 異常終了 (-1)
-----	----	-------------------------------

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール socket 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

domain 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

type 設定用

#define LINUX_SOCKET_STREAM	1
#define LINUX_SOCKET_DGRAM	2

8.13.2 ソケットバインド(linux_bind)

□ C 言語 API

```
int ret = linux_bind(int dstid, int sd, const struct linux_sockaddr *addr, linux_socklen_t addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
const struct linux_sockaddr *	addr	自アドレス情報
linux_socklen_t	addrlen	自アドレス情報の長さ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ パケットの構造

```
struct linux_sockaddr {
    linux_sa_family_t    sa_family;
    char                 sa_data[14];
}
struct linux_in_addr {
    unsigned long        s_addr;
}
struct linux_sockaddr_in {
    linux_sa_family_t    sin_family;
    unsigned short       sin_port;
    struct linux_in_addr sin_addr;
    char                 sin_zero[8];
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `bind` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.3 ソケットリッスン(`linux_listen`)

☐ C 言語 API

```
int ret = linux_listen(int dstid, int sd, int backlog);
```

☐ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
int	backlog	接続キュー最大長

☐ リターンパラメータ

int	ret	正常終了 (0)
int	sd	ソケットディスクリプタ (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール `errno` に準ずる

☐ 機能

詳細は Linux システムコール `listen` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”`linux.h`”ヘッダファイルをインクルードしてください。

8.13.4 ソケットアクセプト(linux_accept)

□ C 言語 API

```
int newsd = linux_accept(int dstid, int sd, const struct linux_sockaddr *addr, linux_socklen_t addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
const struct linux_sockaddr *	addr	接続相手の通信アドレス情報
linux_socklen_t	addrlen	接続相手の通信アドレス情報の長さ

□ リターンパラメータ

int	newsd	ソケットディスクリプタ (≥0) 異常終了 (-1)
-----	-------	-------------------------------

□ パケットの構造

```
struct linux_sockaddr {  
    linux_sa_family_t    sa_family;  
    char                  sa_data[14];  
}  
  
struct linux_in_addr {  
    unsigned long         s_addr;  
}  
  
struct linux_sockaddr_in {  
    linux_sa_family_t     sin_family;  
    unsigned short        sin_port;  
    struct linux_in_addr  sin_addr;  
    char                  sin_zero[8];  
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `accept` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず `"linux.h"` ヘッダファイルをインクルードしてください。

`"linux.h"` ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTONL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.5 ソケットコネクト(linux_connect)

□ C 言語 API

```
int ret = linux_connect(int dstid, int sd, const struct linux_sockaddr *addr, linux_socklen_t addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
const struct linux_sockaddr *	addr	接続相手の通信アドレス情報
linux_socklen_t	addrlen	接続相手の通信アドレス情報の長さ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ パケットの構造

```
struct linux_sockaddr {
    linux_sa_family_t    sa_family;
    char                 sa_data[14];
}
struct linux_in_addr {
    unsigned long        s_addr;
}
struct linux_sockaddr_in {
    linux_sa_family_t    sin_family;
    unsigned short       sin_port;
    struct linux_in_addr sin_addr;
    char                 sin_zero[8];
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `connect` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.6 接続相手の名前を取得(linux_getpeername)

□ C 言語 API

```
int ret = linux_getpeername(int dstid, int sd, struct linux_sockaddr *addr, linux_socklen_t *addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
struct linux_sockaddr *addr		接続相手の通信アドレス情報
linux_socklen_t	addrlen	接続相手の通信アドレス情報の長さ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ パケットの構造

```
struct linux_sockaddr {  
    linux_sa_family_t    sa_family;  
    char                  sa_data[14];  
}  
  
struct linux_in_addr {  
    unsigned long         s_addr;  
}  
  
struct linux_sockaddr_in {  
    linux_sa_family_t     sin_family;  
    unsigned short        sin_port;  
    struct linux_in_addr  sin_addr;  
    char                  sin_zero[8];  
}
```

□ 機能

詳細は Linux システムコール getpeername 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.7 ソケット名を取得(linux_getsockname)

□ C 言語 API

```
int ret = linux_getsockname(int dstid, int sd, struct linux_sockaddr *addr, linux_socklen_t *addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
struct linux_sockaddr *addr		自アドレス情報
linux_socklen_t	addrlen	自アドレス情報の長さ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ パケットの構造

```
struct linux_sockaddr {  
    linux_sa_family_t    sa_family;  
    char                 sa_data[14];  
}  
  
struct linux_in_addr {  
    unsigned long        s_addr;  
}  
  
struct linux_sockaddr_in {  
    linux_sa_family_t    sin_family;  
    unsigned short       sin_port;  
    struct linux_in_addr sin_addr;  
    char                 sin_zero[8];  
}
```

□ 機能

詳細は Linux システムコール getsockname 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.8 ソケットオプションの取得(linux_getsockopt)

□ C 言語 API

```
int ret = linux_getsockopt(int dstid, int sd, int level, int optname, void *optval, linux_socklen_t *optlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
int	level	オプション層
int	optname	オプション名
void	*optval	オプション値
linux_socklen_t	*optlen	オプションサイズ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール getsockopt 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

level 設定用

#define LINUX_SOL_SOCKET	1
#define LINUX_IPPROTO_TCP	5
#define LINUX_IPPROTO_UDP	17

optname 設定用

#define LINUX_SO_DEBUG	1
#define LINUX_SO_REUSEADDR	2
#define LINUX_SO_TYPE	3
#define LINUX_SO_ERROR	4
#define LINUX_SO_DONTROUTE	5
#define LINUX_SO_BROADCAST	6
#define LINUX_SO_SNDBUF	7
#define LINUX_SO_RCVBUF	8
#define LINUX_SO_KEEPALIVE	9
#define LINUX_SO_OOBINLINE	10
#define LINUX_SO_LINGER	13
#define LINUX_SO_SNDLOWAT	19
#define LINUX_SO_RCVLOWAT	18
#define LINUX_SO_SNDTIMEO	21
#define LINUX_SO_RCVTIMEO	20
#define LINUX_TCP_KEEPIDLE	4
#define LINUX_TCP_KEEPINTVL	5
#define LINUX_TCP_KEEPCNT	6

optval 設定用(LINUX_SO_LINGER 用テーブル)

```
struct linux_linger {  
    int    l_onoff;  
    int    l_linger;  
};
```

8.13.9 ソケットオプションの設定(linux_setsockopt)

□ C 言語 API

```
int ret = linux_setsockopt(int dstid, int sd, int level, int optname, void *optval, linux_socklen_t optlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
int	level	オプション層
int	optname	オプション名
void	*optval	オプション値
linux_socklen_t	optlen	オプションサイズ

□ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール setsockopt 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

level 設定用

#define LINUX_SOL_SOCKET	1
#define LINUX_IPPROTO_TCP	5
#define LINUX_IPPROTO_UDP	17

optname 設定用

#define LINUX_SO_DEBUG	1
#define LINUX_SO_REUSEADDR	2
#define LINUX_SO_TYPE	3
#define LINUX_SO_ERROR	4
#define LINUX_SO_DONTROUTE	5
#define LINUX_SO_BROADCAST	6
#define LINUX_SO_SNDBUF	7
#define LINUX_SO_RCVBUF	8
#define LINUX_SO_KEEPALIVE	9
#define LINUX_SO_OOBINLINE	10
#define LINUX_SO_LINGER	13
#define LINUX_SO_SNDLOWAT	19
#define LINUX_SO_RCVLOWAT	18
#define LINUX_SO_SNDTIMEO	21
#define LINUX_SO_RCVTIMEO	20
#define LINUX_TCP_KEEPIDLE	4
#define LINUX_TCP_KEEPINTVL	5
#define LINUX_TCP_KEEPCNT	6

optval 設定用(LINUX_SO_LINGER 用テーブル)

```
struct linux_linger {  
    int    l_onoff;  
    int    l_linger;  
};
```

8.13.10 データ受信(linux_recv)

□ C 言語 API

```
int ret = linux_recv(int dstid, int sd, const void *buf, size_t len, int flags);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
void	*buf	バッファ
size_t	len	バイト数
int	flags	フラグ

□ リターンパラメータ

int	ret	受信バイト数 (≥0) 異常終了 (-1)
-----	-----	--------------------------

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール recv 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

flags 設定用

#define LINUX_MSG_OOB	0x0001
#define LINUX_MSG_DONTROUTE	0x0004
#define LINUX_MSG_DONTWAIT	0x0040
#define LINUX_MSG_EOR	0x0080
#define LINUX_MSG_CONFIRM	0x0800
#define LINUX_MSG_NOSIGNAL	0x4000
#define LINUX_MSG_MORE	0x8000

8.13.11 データ受信と送信元アドレス取得(linux_recvfrom)

□ C 言語 API

```
int ret = linux_recvfrom(int dstid, int sd, const void *buf, size_t len, int flags
                        struct linux_sockaddr *dest_addr, linux_socklen_t addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
void	*buf	バッファ
size_t	len	バイト数
int	flags	フラグ
struct linux_sockaddr *addr		送信元アドレス情報
linux_socklen_t addrlen		送信元アドレス情報の長さ

□ リターンパラメータ

int	ret	受信バイト数 (≥0) 異常終了 (-1)
-----	-----	--------------------------

□ パケットの構造

```
struct linux_sockaddr {
    linux_sa_family_t    sa_family;
    char                 sa_data[14];
}
struct linux_in_addr {
    unsigned long        s_addr;
}
struct linux_sockaddr_in {
    linux_sa_family_t    sin_family;
    unsigned short       sin_port;
    struct linux_in_addr sin_addr;
    char                 sin_zero[8];
}
```

□ エラーコード

Linux システムコール errno に準ずる

□ 機能

詳細は Linux システムコール recvfrom 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.12 データ送信(linux_send)

□ C 言語 API

```
int ret = linux_send(int dstid, int sd, const void *buf, size_t len, int flags);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
void	*buf	バッファ
size_t	len	バイト数
int	flags	フラグ

□ リターンパラメータ

int	ret	送信バイト数 (≥0) 異常終了 (-1)
-----	-----	--------------------------

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `send` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

flags 設定用

#define LINUX_MSG_OOB	0x0001
#define LINUX_MSG_DONTROUTE	0x0004
#define LINUX_MSG_DONTWAIT	0x0040
#define LINUX_MSG_EOR	0x0080
#define LINUX_MSG_CONFIRM	0x0800
#define LINUX_MSG_NOSIGNAL	0x4000
#define LINUX_MSG_MORE	0x8000

8.13.13 データ送信と送信先アドレス設定(linux_sendto)

□ C 言語 API

```
int ret = linux_sendto(int dstid, int sd, const void *buf, size_t len, int flags
                        const struct linux_sockaddr *dest_addr, linux_socklen_t addrlen);
```

□ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
void	*buf	バッファ
size_t	len	バイト数
int	flags	フラグ
const struct linux_sockaddr	*addr	送信先アドレス情報
linux_socklen_t	addrlen	送信先アドレス情報の長さ

□ リターンパラメータ

int	ret	送信バイト数 (≥0) 異常終了 (-1)
-----	-----	--------------------------

□ パケットの構造

```
struct linux_sockaddr {
    linux_sa_family_t    sa_family;
    char                 sa_data[14];
}
struct linux_in_addr {
    unsigned long        s_addr;
}
struct linux_sockaddr_in {
    linux_sa_family_t    sin_family;
    unsigned short       sin_port;
    struct linux_in_addr sin_addr;
    char                 sin_zero[8];
}
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `sendto` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

linux_sockaddr sa_family 設定用

#define LINUX_AF_UNSPEC	0
#define LINUX_AF_INET	2

linux_sockaddr port 設定用

#define HTONS(value)	NTOHS(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(16bit データ)

linux_sockaddr sin_addr 設定用

#define INADDR_ANY	(unsigned long)0x00000000
--------------------	---------------------------

linux_sockaddr sin_addr 設定用

#define HTONL(value)	NTOHL(value)
----------------------	--------------

※ ホストバイトオーダーをネットワークバイトオーダーに変換するマクロ(32bit データ)

8.13.14 コネクションの切断(linux_shutdown)

☐ C 言語 API

```
int ret = linux_shutdown(int dstid, int sd, int how);
```

☐ パラメータ

int	dstid	0 を指定してください
int	sd	ソケットディスクリプタ
int	how	切断方向

☐ リターンパラメータ

int	ret	正常終了 (0)
		異常終了 (-1)

☐ エラーコード

Linux システムコール errno に準ずる

☐ 機能

詳細は Linux システムコール shutdown 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず”linux.h”ヘッダファイルをインクルードしてください。

”linux.h”ヘッダファイルでは以下のシンボル定数が定義されます。

how 設定用

#define LINUX_SHUT_RD	0
#define LINUX_SHUT_WR	1
#define LINUX_SHUT_RDWR	2

8.13.15 複数ソケットの監視(linux_select)

□ C 言語 API

```
int ret = linux_select(int dstid, int nfds,  
                      linux_fd_set *readfds, linux_fd_set *writefds, linux_fd_set *exceptfds, struct linux_timeval *timeout);
```

□ パラメータ

int	dstid	0 を指定してください
int	nfds	ソケットディスクリプタ最大値
linux_fd_set	*readfds	監視受信ソケットディスクリプタ
linux_fd_set	*writefds	監視送信ソケットディスクリプタ
linux_fd_set	*exceptfds	監視 TCP 緊急データ受信ソケットディスクリプタ
struct linux_timeval	*timeout	タイムアウト時間

□ リターンパラメータ

int	ret	更新ソケットディスクリプタ数 (≧0) 異常終了 (-1)
-----	-----	----------------------------------

□ パケットの構造

```
typedef struct {  
    unsigned long    fds_bits[_LINUX_FDSET_LONGS];  
} linux_fd_set;  
  
struct linux_timeval {  
    long             tv_sec;        /* 秒 */  
    long             tv_usec;      /* マイクロ秒 */  
};
```

□ エラーコード

Linux システムコール `errno` に準ずる

□ 機能

詳細は Linux システムコール `select` 仕様を参照ください。SH 版のみ発行可能です。Linux 版では Linux システムコールをご利用ください。

本コマンド発行する場合、必ず `linux.h` ヘッダファイルをインクルードしてください。

`linux.h` ヘッダファイルでは以下のシンボル定数が定義されます。

9. エラーコード

9.1 Linux 版エラー

表9-1 Linux版エラー

エラー番号	エラー内容	備考
-512	SIGINT による中断	
-1024	Lock エラー	
-1025	モジュールオープン	
-1026	メモリ確保エラー	
-1027	メモリコピーエラー	
-1028	タイムアウトエラー	

付録A 変更履歴

【第1版】

初版

【第2版】

4.5 章(4) エディタについての注記を追加
4.8.3 章 LinuxFormat ツールリビジョンアップに関する変更
5.13.5 章 LinuxFormat ツールリビジョンアップに関する変更
5.13.6 章(1) エディタについての注記を追加
6.3.2 章 エディタについての注記を追加
付録 B 従来 SDK(SH 動作)変更手順を追加
付録 C トラブルシューティングを追加

【第3版】

社名を変更しました。

付録B 従来 SDK(SH 動作)変更手順

B.1 従来SDK(SH動作)書換え時のSW設定

従来 SDK(SH 動作)書換え時の SW 設定を以下に示します。

表B-1 従来SDK(SH動作)書換え時のSW設定

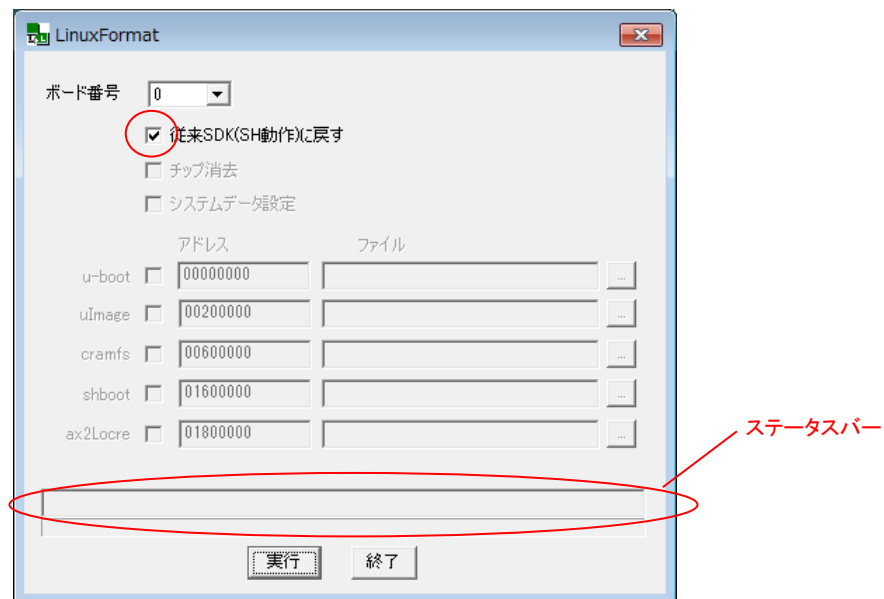
スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	予備		SW2	1	OFF	予備	
	2	OFF	予備			2	OFF	予備	
	3	OFF	予備			3	OFF	予備	
	4	OFF	予備			4	OFF	予備	
	5	OFF	予備			5	OFF	従来 SDK(SH 動作)書き換え	
	6	OFF	予備			6	ON		
	7	OFF	予備			7	ON		
	8	OFF	予備			8	OFF	予備	

B.2 ネットワーク設定

従来 SDK(SH 動作)書換え時、NVP-Ax230 の IP アドレスは 192.168.0.205 (固定) に設定されております。接続する PC のネットワーク設定をあわせてください。

B.3 従来SDK(SH動作)フォーマット手順

NVP-Ax230 に電源投入すると NVP-Ax230 上の ERR-LED=点灯(赤)、RUN-LED=点滅(緑、500ms 間隔)となります。PC の[スタート]-[Ax230SDK]-[Tools]-[LinuxFormat]メニューを選択してください。LinuxFormat が起動します(図は LinuxFormat Version 1.1 以降)。



図B-1 LinuxFormatツール③(従来SDK)

[従来 SDK(SH 動作)に戻す]チェックボタンを ON にしてください。[実行]ボタンをクリックすると RUN-LED=点滅(緑、100ms 間隔)になり、書き込みを開始します。書き換えは時間がかかります。実行中に電源を切らないでください。書き込み完了後、ステータスバーに”完了...”メッセージが表示されます。

B.4 従来SDK(SH動作)実行時のSW設定

従来 SDK(SH 動作)実行時の SW 設定を以下に示します。

表B-2 従来SDK(SH動作)実行時のSW設定

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	予備		SW2	1	OFF	予備	
	2	OFF	予備			2	OFF	予備	
	3	OFF	予備			3	OFF	予備	
	4	OFF	予備			4	OFF	予備	
	5	OFF	予備			5	OFF	従来 SDK(SH 動作)書き換え	
	6	OFF	予備			6	OFF		
	7	OFF	予備			7	OFF		
	8	OFF	予備			8	OFF	予備	

従来 SDK(SH 動作)フォーマット完了後、NVP-Ax230 を電源切断して SW 設定を従来 SDK(SH 動作)実行時に変更してください。NVP-Ax230 に電源投入すると従来 SDK(SH 動作)として起動します。

付録C トラブルシューティング

- (1) SH 側アプリケーションを startup.bat で自動起動したが、システム起動直後のアプリケーション動作が起因して赤 LED が点灯する。そのため、startup.bat の削除ができない。

【Answer】

ブートモードで起動することで startup.bat を適用せずに起動できます。ブートモードの SW 設定を表 C-1 に示します。本 SW 設定後、電源を再投入してください。システムが起動を確認後、startup.bat を削除してください。

表C-1 ブートモード時のSW設定

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	ブートモード		SW2	1	OFF	予備	
	2	OFF				2	OFF	予備	
	3	ON				3	OFF	予備	
	4	OFF				4	OFF	予備	
	5	OFF	予備			5	ON	Linux 起動	
	6	OFF	予備			6	OFF		
	7	OFF	予備			7	OFF		
	8	OFF	予備			8	OFF	予備	

- (2) Linux 書換えの時、NVP-Ax230 に電源投入すると NVP-Ax230 上の ERR-LED=点灯(赤)、RUN-LED=点滅(緑、500ms 間隔)となるが、LinuxFormat ツールの[実行]ボタンを押すと RUN-LED=点滅しなくなる。

【Answer】

SH 側アプリケーションの自動実行(startup.bat)が要因でシステム起動できない可能性があります。ブートモードで Flash 書換えすることで回避可能です。ブートモードの SW 設定を表 C-2 に示します。本 SW 設定後、電源を再投入して LinuxFormat を実行してください。

表C-2 Linux書換え時のSW設定(+ブートモード)

スイッチ設定			説明	備考	スイッチ設定			説明	備考
SW1	1	OFF	ブートモード		SW2	1	OFF	予備	
	2	OFF				2	OFF	予備	
	3	ON				3	OFF	予備	
	4	OFF				4	OFF	予備	
	5	OFF	予備			5	ON	Linux 書き換え	
	6	OFF	予備			6	OFF		
	7	OFF	予備			7	OFF		
	8	OFF	予備			8	OFF	予備	

(3) LinuxFormat 実行後、ステータスバーに”完了...”メッセージの他に下記メッセージが表示される。

- ・ ”WARNNING-Invalid-Serial”
- ・ ”WARNNING-Invalid-MAC”

【Answer】

NVP-Ax230 は工場出荷時にボード固有のシリアル番号、MAC アドレス等が設定されますが、なんらかの原因でシリアル番号、MAC アドレスが不正となっていることを示します。筐体、および、ボードに記載されるシリアル番号を明記の上、弊社技術サポート窓口までご連絡ください。

画像認識ユニット NVP-Ax230SDK for Linux
ユーザーズマニュアル（第3版）

（C）マクセルシステムテック株式会社

開発元

マクセルシステムテック株式会社

設計部 〒992-0021 山形県米沢市花沢3091-6

営業部 〒244-0801 神奈川県横浜市戸塚区信濃町549-2三宅ビル

技術サポート窓口 URL <http://www.systemtech.maxell.co.jp/>
mail : vp-support@maxell.co.jp