



μ C3/Standard + μ Net3 評価版ガイド

(Armadillo-440 編)

2016 年 03 月 イー・フォース株式会社

1. はじめに

このたびは、μ C3/Standard + μ Net3 評価版をお試しいただきまして、ありがとうございます。本書では評価版パッケージのインストール手順、パッケージの概要、サンプルプログラム実行方法などを説明いたします。

なお、μ C3/Standard (RTOS) 、μ Net3(TCP/IP プロトコルスタック)の詳細については、評価版のインストール後、Document フォルダにインストールされる各ユーザーズガイドを参考にして下さい。

制限事項

本評価版はアットマークテクノ社の Armadillo-440 上でのみ利用可能です。添付しているプログラムを製品評価以外の目的で使用することはできません。本評価版は製品版とは異なり、RTOS、プロトコルスタックのソースコードが含まれないほか、下記の機能制限がなされています。

- RTOS の ID 数の制限

- ・タスクの個数：10（製品版では 255）
- ・その他のオブジェクト個数：それぞれ 10（製品版では 999）

※. 参考情報：μ C3/Standardユーザーズガイド「3.3.1 オブジェクトのID番号上限
のコンフィグレーション情報」

- TCP/IP プロトコルスタックの制限

- ・ソケット最大数：6
- ・登録コンテンツ：3
- ・HTTPセッション数：2
- ・マルチキャスト：使用不可
- ・IP reassembly：使用不可
- ・ネットワークのMTUサイズ：変更不可
- ・ネットワークバッファ数：変更不可
- ・ETHERNETのコンフィグレーション不可
- ・MACアドレス：変更不可

本評価版を使用するにはIARシステムズ社の統合開発環境が別途必要です。なお、統合開発環境の評価版は、IARシステムズ社のホームページから入手が可能です。

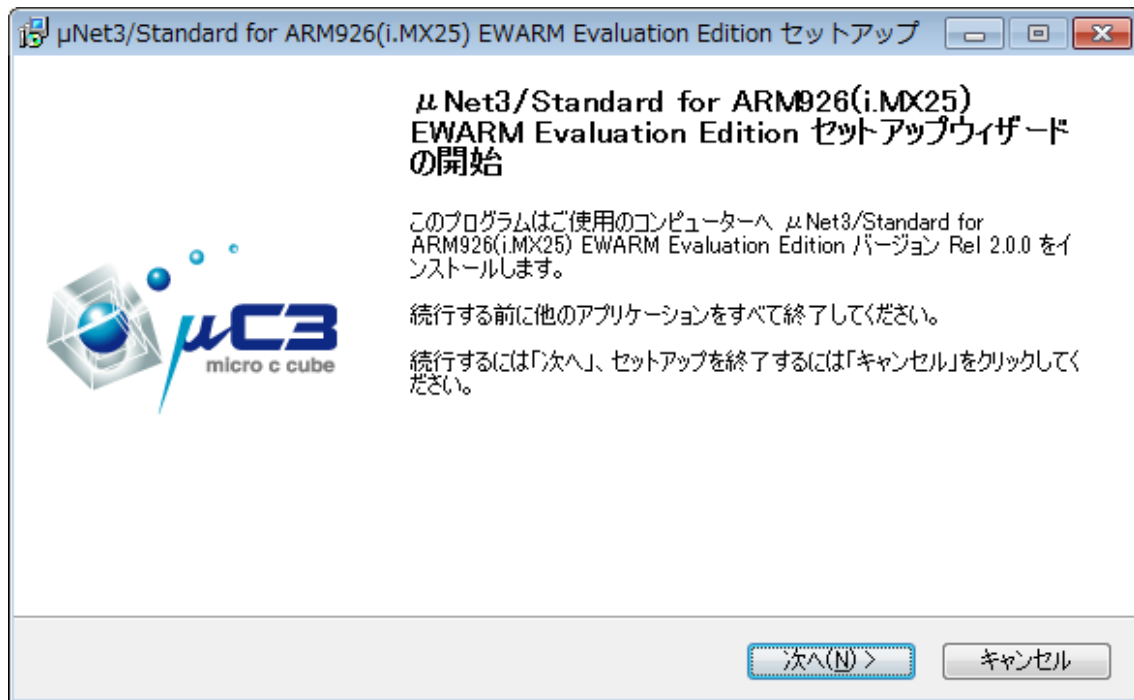
※注意…統合開発環境のコードサイズ制限版では、ネットワークのサンプルプログラム(Armadillo-440.NET)を実行することが出来ません。その場合は、30日間期間限定版を使用してください。

2. パッケージの概要

インストーラ

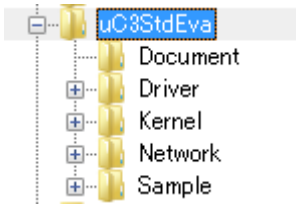
μ C3/Standard + μ Net3 評価版では、インストーラが用意されています。

uNet3s_iMX25_ewarm_eva.exe を起動すると下記インストール画面が表示されるので、インストーラのメッセージに従い、評価版パッケージをインストールしてください。

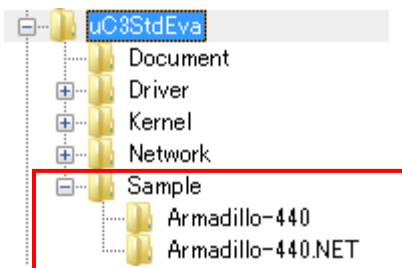


フォルダ構成

以下ストールが完了すると、評価版のパッケージは、以下の構成となります。



サンプルプログラム



サンプルプログラムが上記のフォルダに含まれています。

次ページ以降に、評価版パッケージ内のサンプルプログラムの実行方法を説明します。

- Armadillo-440

μ C3/Standard の RTOS 上で、LED 点滅と UART 通信を実施するプログラムです。
UART 通信では、μ C3/Standard の標準 COM ポートドライバを使用しています。

- Armadillo-440.NET

μ C3/Standard + μ Net3 上で、ネットワーク機能を実施するプログラムです。

3. サンプルプログラム

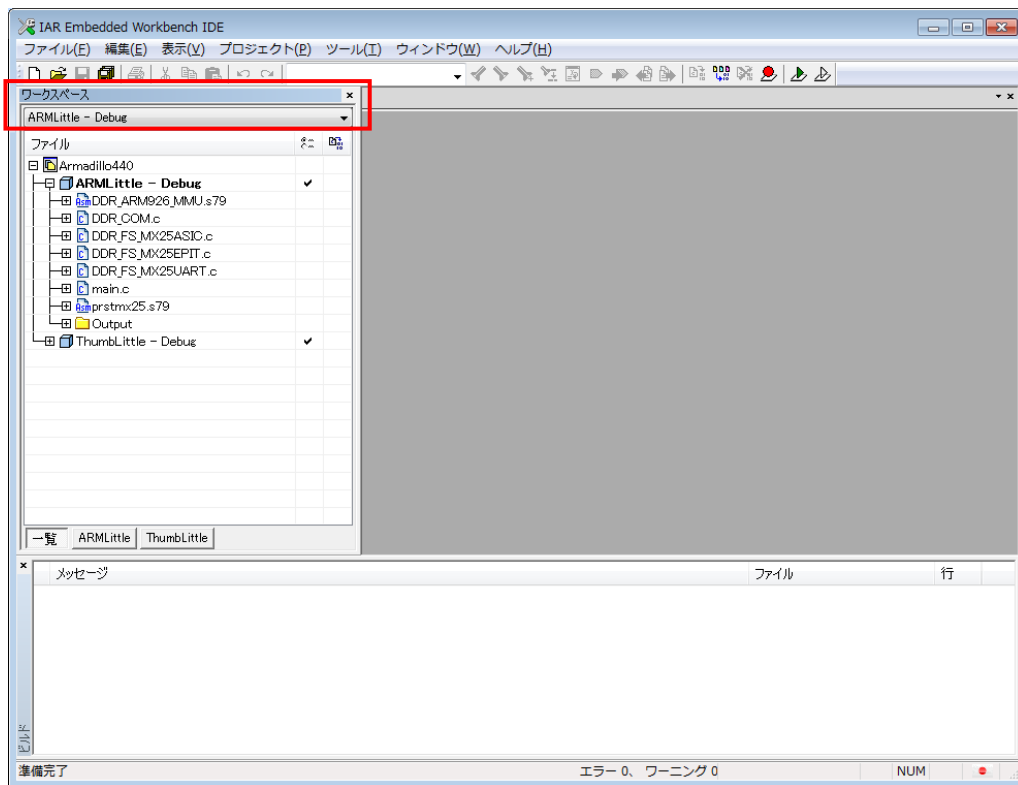
2つのサンプルプログラムの実行例を説明します。

Armadillo-440

フォルダ内には、Readme.txt もありますので、こちらも合わせて参照してください。以下動作例を示します。

まず、EWARMでプロジェクトを開きます。

EWARMを起動し、メニュー 「ファイル」 → 「開く」 → 「ワークスペース」 でフォルダ内の「Armadillo440.eww」を開いてください。

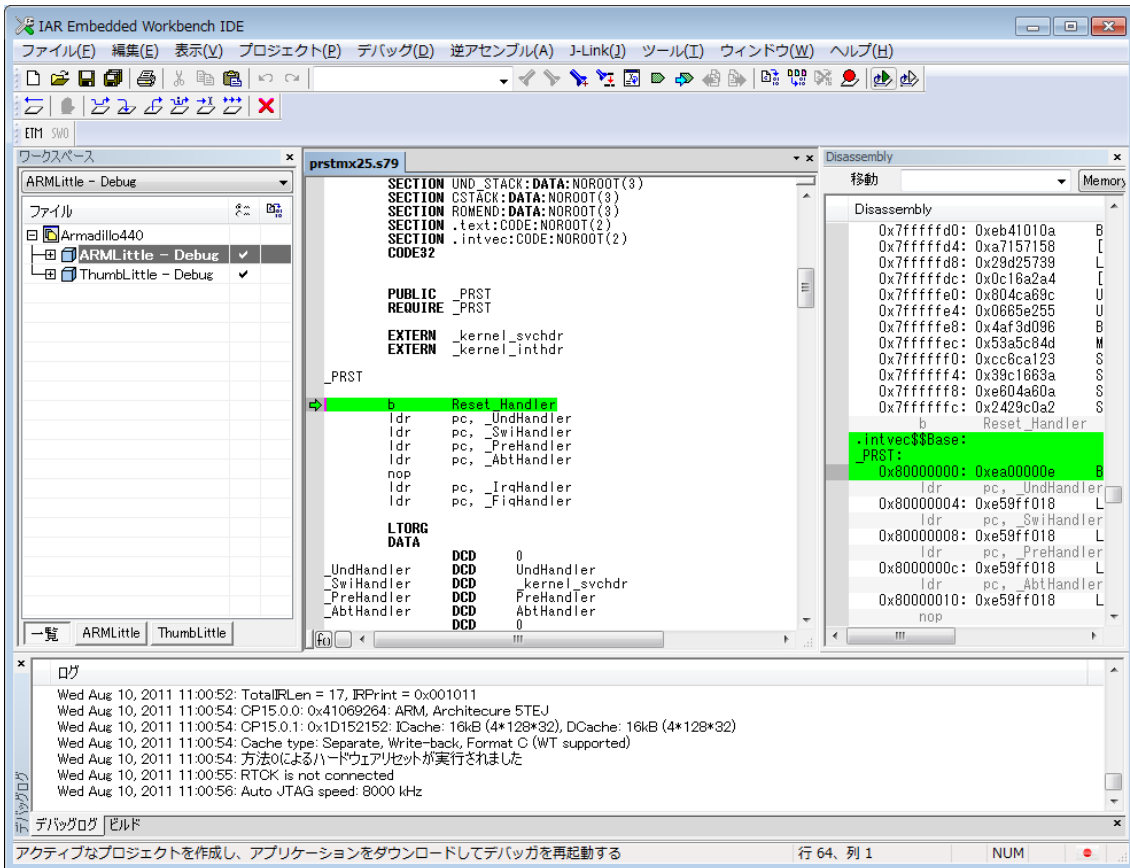



ワークスペースには、デバッグ用に2つのプロジェクトが作成されています。Thumb ステートを使用した"ThumbLittle"と ARM ステートのみの"ARMLittle"です。各のプロジェクトには"Debug"と"Flash"の2つのコンフィグレーションがあります。"Debug"では、生成されたロードモジュールが SRAM にダウンロードされ、"Flash"を選択すると Flash メモリへロードモジュールをダウンロード後、プログラムが実行されます。


実行したいプロジェクトを、上記のプルダウンメニューから選択してください。その後、

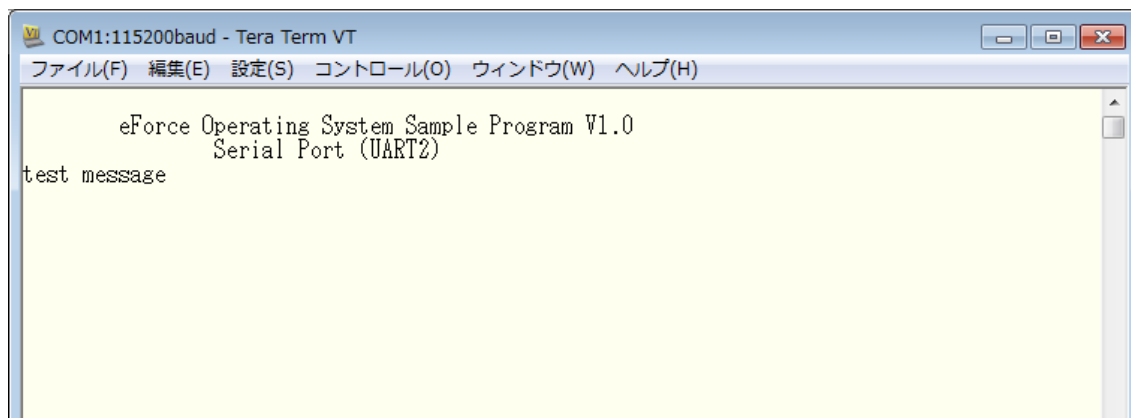
「ビルド」ボタンでプログラムのコンパイルを実行します。





プログラムを動作させるには、まず  「ダウンロードしてデバッグ」 ボタンをクリックすると上記のデバッグ画面に切り替わります。

このサンプルでは、オンボードの LED 点滅と、UART2 からメッセージの出力の後、一文字ずつ受信と送信を繰り返します。そのため、次ページを参考にして、PC のターミナルエミュレータを設定後、  「実行」 ボタンをクリックします。



このサンプルでは、オンボードの LED が点滅し、UART2 からはメッセージ出力の後、一文字ずつ受信と送信を繰り返します。

上記の例では、PC と Armadillo をシリアルケーブル(クロス)で接続し、Tera Term 上にメッセージ表示後、PC のキーボードから「test message」とメッセージを打ち込んだ例を示しています。

通信条件は下記となります。

ボーレート	データ	パリティ	ストップビット	フロー制御
115200	8 bit	なし	1 bit	Xon/Xoff

Armadillo-440.NET

フォルダ内には、Readme.txt もありますので、こちらも合わせて参照してください。サンプルプログラムでは、DHCP クライアント・HTTP サーバ・DNS クライアントの機能を試すことが出来ます。

また、本プログラムでは、DHCP サーバから IP アドレスを取得します。DHCP サーバがない環境で動作させる場合は、net.c と net_cfg.c 内の記述を以下のように変更してください。

変更、その 1

```
#define DHCP_ENB    1    /* Enable DHCP */
```



```
#define DHCP_ENB    0    /* Enable DHCP */
```

変更、その 2

```
T_NET_ADR gNET_ADR[] = {
```

```
    {0x0, 0x0, 0x0, 0x0, 0x0},
```

```
};
```

```
#else
```

```
T_NET_ADR gNET_ADR[] = {
```

```
    {0x0, 0x0, 0xC0A80B23, 0xC0A80001, 0xFFFFFFFF0},
```

```
    /* IP Address 192.168.11.35, Gateway 192.168.11.1, Subnet 255.255.255.0 */
```

```
};
```

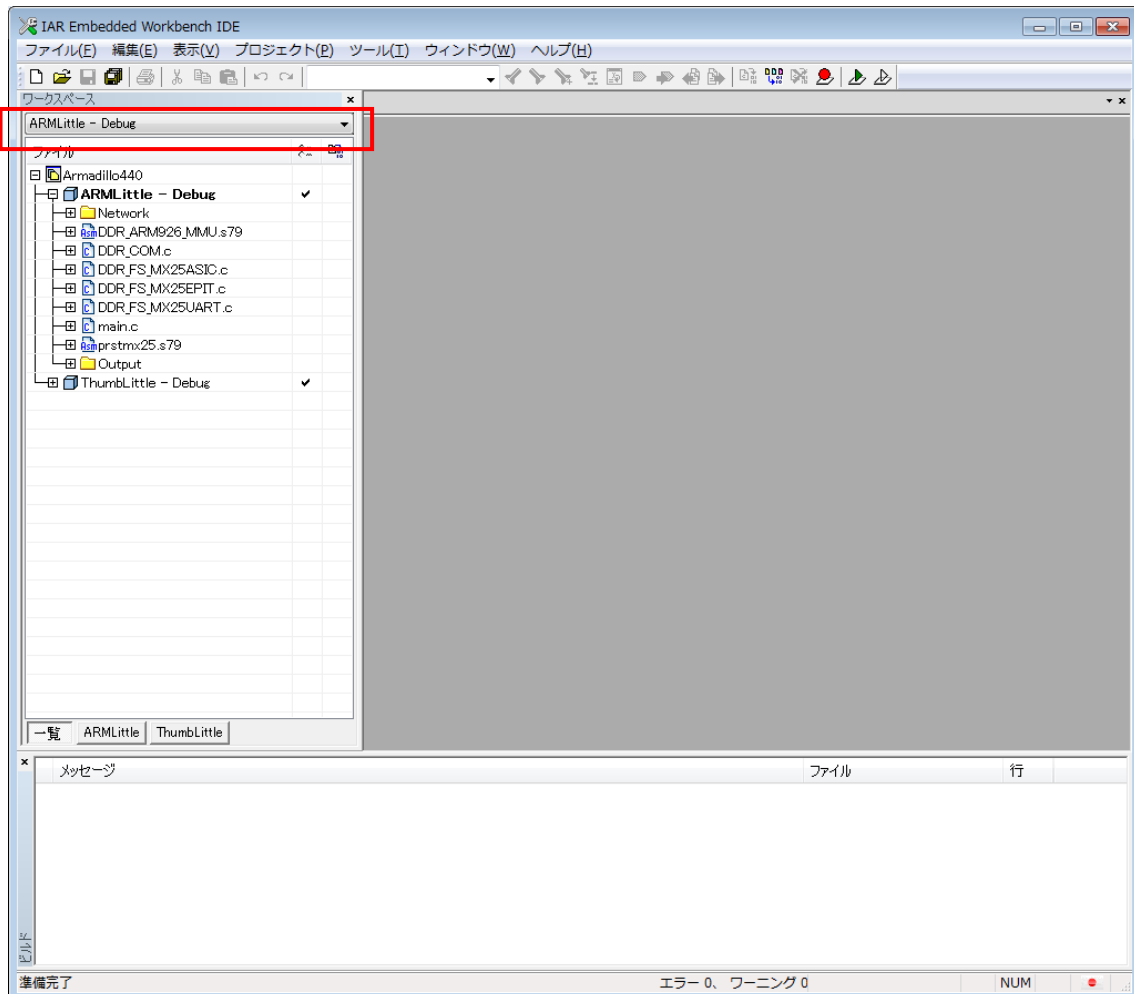
```
#endif
```


デバイスに設定する IP Address, Gateway Address, Subnet Mask の必要な値に変更してください。

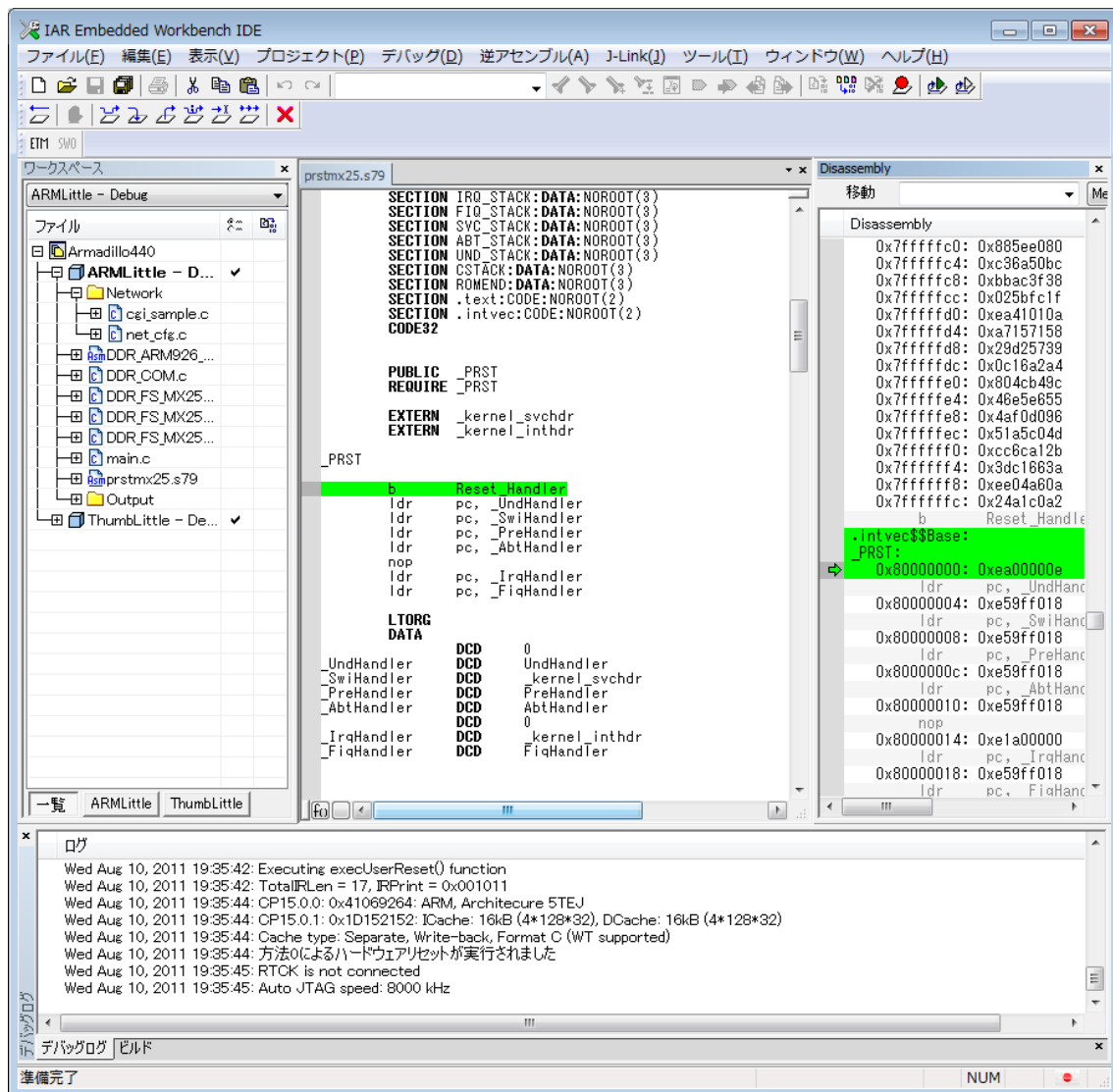
以下動作例を示します。


まず、EWARMでプロジェクトを開きます。

EWARMを起動し、メニュー 「ファイル」 → 「開く」 → 「ワークスペース」 でフォルダ内の「Armadillo440.eww」を開いてください。




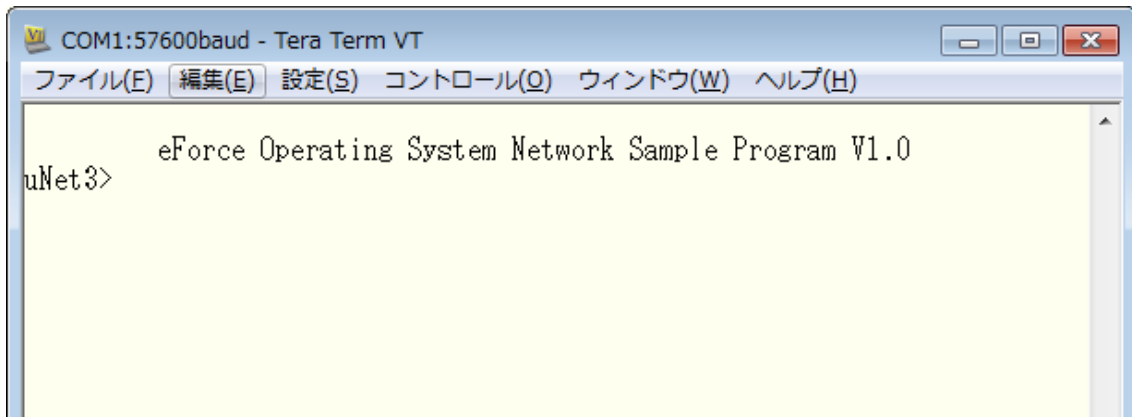
ワークスペースには、デバッグ用に2つのプロジェクトが作成されています。Thumb ステートを使用した"ThumbLittle"と ARM ステートのみの"ARMLittle"です。各のプロジェクトには"Debug"と"Flash"の2つのコンフィグレーションがあります。"Debug"では、生成されたロードモジュールが SRAM にダウンロードされ、"Flash"を選択すると Flash メモリへロードモジュールをダウンロード後、プログラムが実行されます。上記のワークスペース下のプルダウンメニューから選択してください。その後、 「ビルド」ボタンでプログラムをコンパイルします。



プログラムを動作させるには、まず  「ダウンロードしてデバッグ」 ボタンをクリックすると上記の画面に切り替わります。

このサンプルでは、UART を使用してテストメッセージを表示し、UART を使用したモニタ機能も使用しているため、次ページを参考にして、PC のターミナルエミュレータを設定後、

 「実行」 ボタンをクリックします。



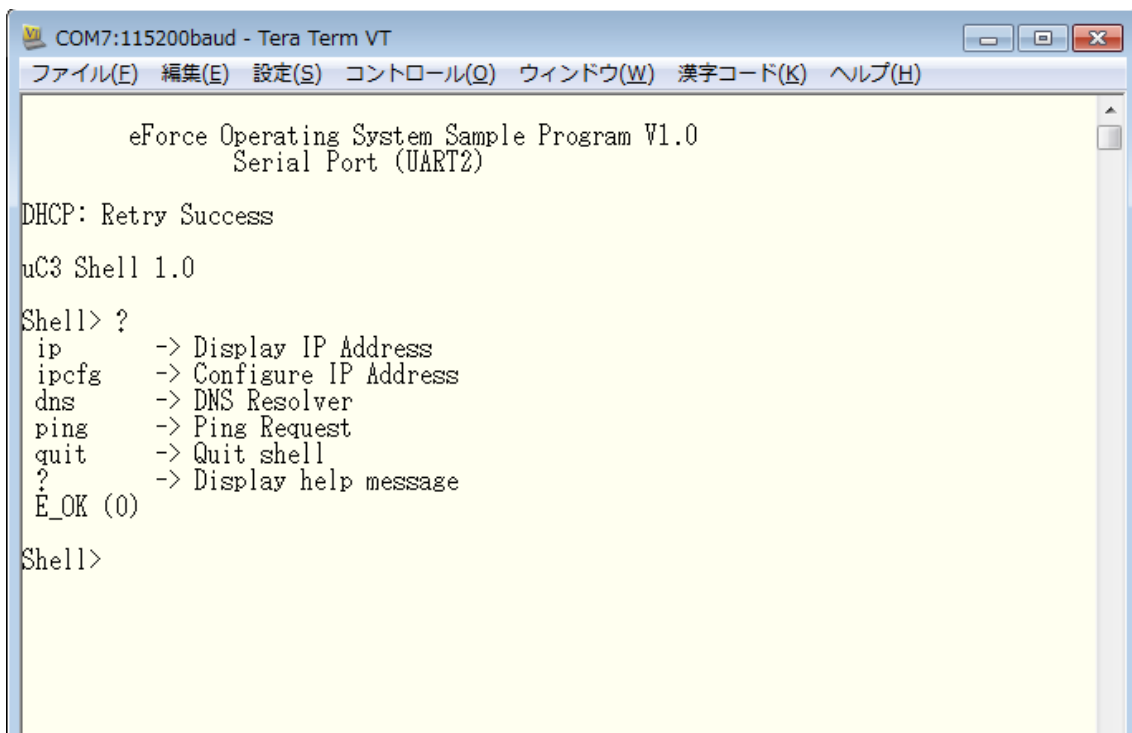
サンプルプログラムが起動すると、オンボードの LED が点滅し、UART2 からは、上記メッセージが表示されます。

テストでは、PC と Armadillo をシリアルケーブル(クロス)で接続し、Tera Term 上にメッセージ表示されています。

この時の通信条件は下記となります。

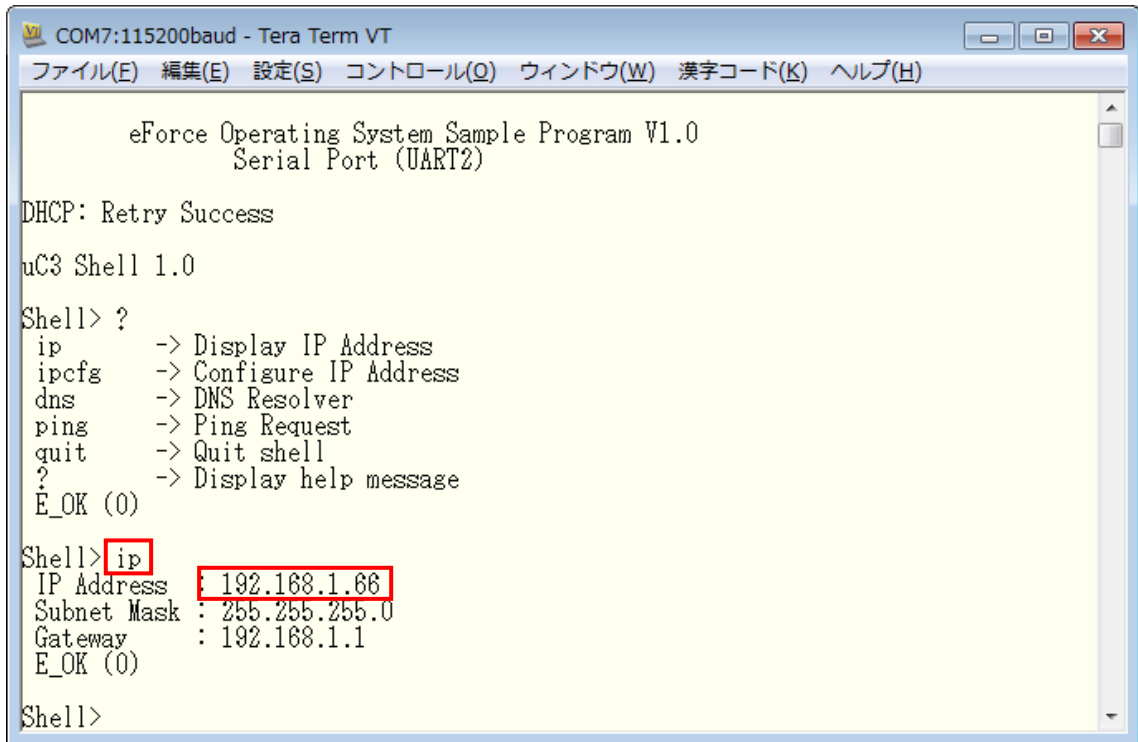
ボーレート	データ	パリティ	ストップビット	フロー制御
115200	8 bit	なし	1 bit	なし

ターミナルエミュレータでは、「?」を入力すると、下記のように使用できるコマンドが表示されます。



① IP アドレスの表示

ターミナルエミュレータで「ip」を入力すると、デバイスに設定された IP アドレスを確認することが出来ます。



```
COM7:115200baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)

eForce Operating System Sample Program V1.0
Serial Port (UART2)

DHCP: Retry Success

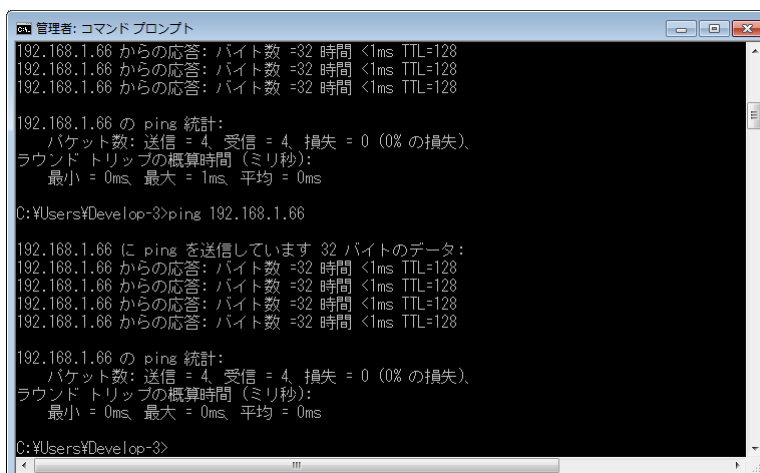
uC3 Shell 1.0

Shell> ?
ip          -> Display IP Address
ipcfg       -> Configure IP Address
dns         -> DNS Resolver
ping        -> Ping Request
quit        -> Quit shell
?           -> Display help message
E_OK (0)

Shell> ip
IP Address  : 192.168.1.66
Subnet Mask : 255.255.255.0
Gateway     : 192.168.1.1
E_OK (0)

Shell>
```

DHCP の機能を使用している場合は、デバイスに割り当てられた IP Address の確認となります。DHCP の機能を使用していない場合、設定した IP Address が表示されます。以降の確認では、このアドレスを使用します。



```
管理者: コマンドプロンプト
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128

192.168.1.66 の ping 統計:
    パケット数: 送信 = 4, 受信 = 4, 損失 = 0 (0% の損失),
ラウンド トリップの概算時間 (ミリ秒):
    最小 = 0ms, 最大 = 1ms, 平均 = 0ms

C:\Users\YDevelop-3>ping 192.168.1.66

192.168.1.66 に ping を送信しています 32 バイトのデータ:
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.1.66 からの応答: バイト数 =32 時間 <1ms TTL=128

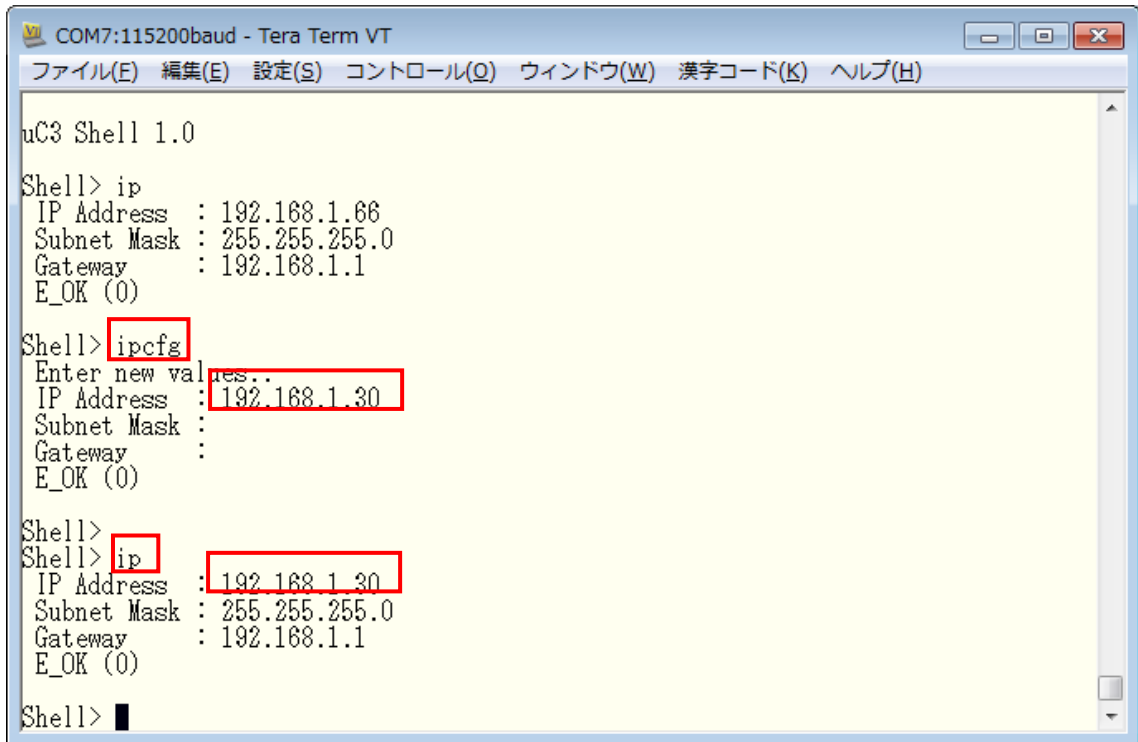
192.168.1.66 の ping 統計:
    パケット数: 送信 = 4, 受信 = 4, 損失 = 0 (0% の損失),
ラウンド トリップの概算時間 (ミリ秒):
    最小 = 0ms, 最大 = 0ms, 平均 = 0ms

C:\Users\YDevelop-3>
```

デバイスとの通信確認は、ping で確認することが出来ます。上記はデバイスと通信確認を実施した例を示します。

次に、デバイスの IP Address をターミナルエミュレータから変更する例を示します。

変更したい IP Address 「192.168.1.30」とします。ターミナルエミュレータで「ipcfg」を入力後、必要な情報を入力します。その後、「ip」と入力した例となります。

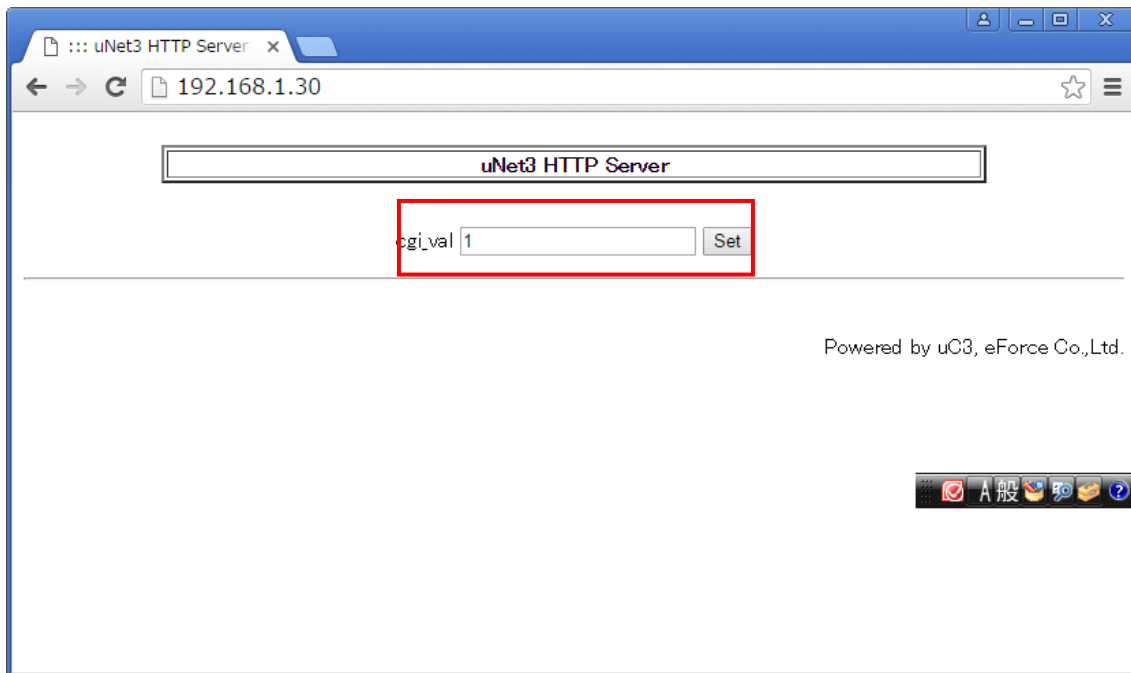


```
COM7:115200baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) 漢字コード(K) ヘルプ(H)

uC3 Shell 1.0
Shell> ip
IP Address : 192.168.1.66
Subnet Mask : 255.255.255.0
Gateway : 192.168.1.1
E_OK (0)
Shell> ipcfg
Enter new values...
IP Address : 192.168.1.30
Subnet Mask :
Gateway :
E_OK (0)
Shell>
Shell> ip
IP Address : 192.168.1.30
Subnet Mask : 255.255.255.0
Gateway : 192.168.1.1
E_OK (0)
Shell> █
```

② HTTP サーバ

ウェブブラウザからデバイスにアクセスすることで、LED の点滅間隔を 100msec 単位で変更することが可能です。アクセスする IP Address は、「①IP アドレスの表示」を参考にして下さい。



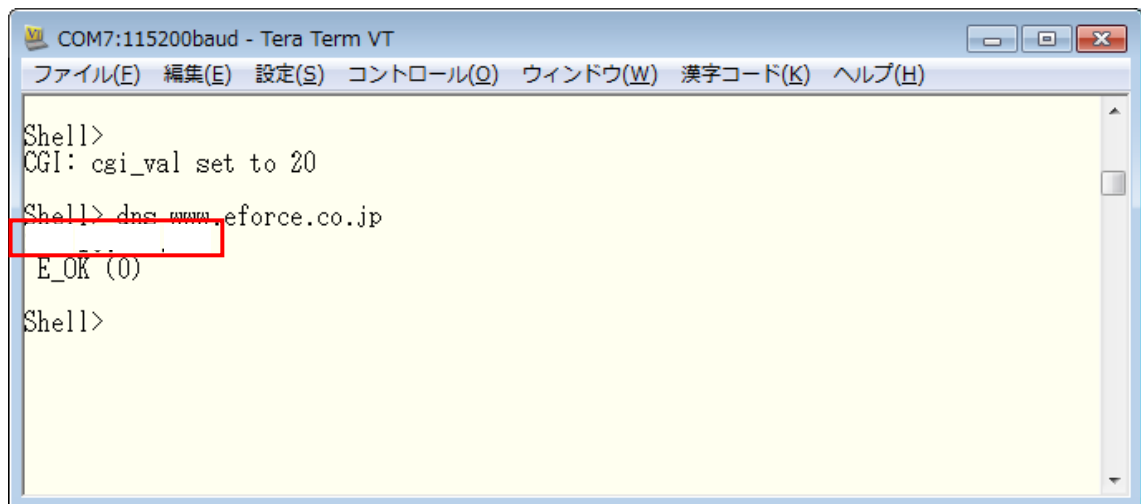
上記は、ウェブブラウザからデバイスにアクセスした例を示します。ここに「1」を入力して「Set」ボタンをクリックすると、LED の点滅周期が 100msec となります。

③ DNS クライアント

FQDN の A クエリを発行して IP アドレスの解決を実施します。

以下に実行例を示します。

ターミナルエミュレータで「dns (ホスト名)」を入力すると、ホスト名の IP Address の解決を実施して、その IP Address を表示します。



```
COM7:115200baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)

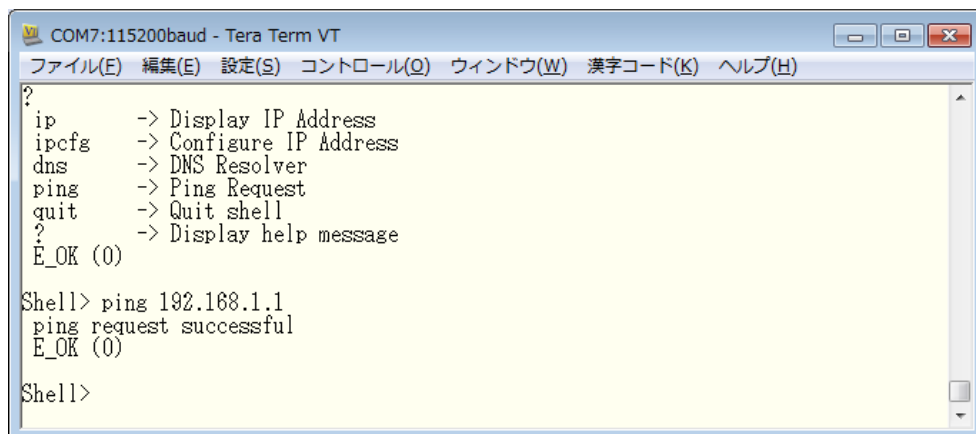
Shell>
CGI: cgi_val set to 20
Shell> dns www.eforce.co.jp
E_OK (0)
Shell>
```

この例では、以下となります。

- ホスト名 : www.eforce.co.jp
- 解決した IP Address : xxx.xx.xxx.xxx

④ Ping の実行

ボードから ping を実行できます。



```
COM7:115200baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)

?
ip      -> Display IP Address
ipcfg   -> Configure IP Address
dns     -> DNS Resolver
ping    -> Ping Request
quit    -> Quit shell
?       -> Display help message
E_OK (0)

Shell> ping 192.168.1.1
ping request successful
E_OK (0)

Shell>
```

上記のように「ping 192.168.1.1」とすると、ボードから ping を実行します。
成功すると、E_OK となります。

4. μC3/Standard のシステムコール

μ C3/Standard は 32 ビットプロセッサが搭載された組込システム向けのリアルタイム OS です。高性能プロセッサが、より高度なリアルタイム制御に耐えられるよう、割込み禁止区間を極力なくし、割込み応答性を重視して設計しています。

μ C3/ Standard は μ ITRON4.0 仕様のフルセットを基本とし使われる可能性の少ないと思われる下記の機能を省いています。

- 静的 API と、これを解釈するコンフィグレータ
- タスク例外処理機能
- サービスコール管理機能

システムコール一覧

○：使用可

×：使用不可

△：プロセッサ依存

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
A) タスク管理機能				
cre_tsk / acre_tsk	○	○	×	×
del_tsk	○	×	×	×
act_tsk / iact_tsk	○	○	○	○
can_act	○	○	○	×
sta_tsk	○	○	○	○
ext_tsk	○	×	×	×
exd_tsk	○	×	×	×
ter_tsk	○	×	×	×
chg_pri	○	○	○	×
get_pri	○	○	○	×
ref_tsk	○	○	○	×
ref_tst	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
B) タスク付属同期				
slp_tsk	○	×	×	×
tslp_tsk	○	×	×	×
wup_tsk/iwup_tsk	○	○	○	○
can_wup	○	○	○	×
rel_wai/irel_wai	○	○	○	○
sus_tsk	○	○	○	×
rsm_tsk	○	○	○	×
frsm_tsk	○	○	○	×
dly_tsk	○	×	×	×
C) タスク例外処理				
def_tex	×	×	×	×
ras_tex	×	×	×	×
iras_tex	×	×	×	×
dis_tex	×	×	×	×
ena_tex	×	×	×	×
sns_tex	×	×	×	×
ref_tex	×	×	×	×
D) 同期・通信 セマフォ				
cre_sem/acre_sem	○	○	×	×
del_sem	○	×	×	×
sig_sem/isig_sem	○	○	○	○
wai_sem	○	×	×	×
pol_sem	○	○	○	×
twai_sem	○	×	×	×
ref_sem	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
E) 同期・通信 イベントフラグ				
cre_flg／acre_flg	○	○	×	×
del_flg	○	×	×	×
set_flg／iset_flg	○	○	○	○
clr_flg	○	○	○	×
wai_flg	○	×	×	×
pol_flg	○	○	○	×
twai_flg	○	×	×	×
ref_flg	○	○	○	×
F) 同期・通信 データキュー				
cre_dtq／acre_dtq	○	○	×	×
del_dtq	○	×	×	×
snd_dtq	○	×	×	×
psnd_dtq／ipsnd_dtq	○	○	○	○
tsnd_dtq	○	×	×	×
fsnd_dtq／ifsnd_dtq	○	○	○	○
rcv_dtq	○	○	○	×
prcv_dtq	○	○	○	×
trcv_dtq	○	×	×	×
ref_dtq	○	○	○	×
G) 同期・通信 メールボックス				
cre_mbx／acre_mbx	○	○	×	×
del_mbx	○	×	×	×
snd_mbx	○	○	○	×
rcv_mbx	○	×	×	×
prcv_mbx	○	○	○	×
trcv_mbx	○	×	×	×
ref_mbx	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
H) 拡張同期・通信 ミューテック				
cre_mtx/acre_mtx	○	○	×	×
del_mtx	○	×	×	×
loc_mtx	○	×	×	×
ploc_mtx	○	×	×	×
tloc_mtx	○	×	×	×
unl_mtx	○	×	×	×
ref_mtx	○	○	○	×
I) 拡張同期・通信 メッセージバッファ				
cre_mbf/acre_mbf	○	○	×	×
del_mbf	○	×	×	×
snd_mbf	○	×	×	×
psnd_mbf	○	○	○	×
tsnd_mbf	○	×	×	×
rcv_mbf	○	×	×	×
prcv_mbf	○	○	○	×
trcv_mbf	○	×	×	×
ref_mbf	○	○	○	×
J) 拡張同期・通信 ランデブ				
cre_por/acre_por	○	○	×	×
del_por	○	×	×	×
cal_por	○	×	×	×
tcal_por	○	×	×	×
acp_por	○	×	×	×
pacp_por	○	×	×	×
tacp_por	○	×	×	×
fwd_por	○	×	×	×
rpl_rdv	○	×	×	×
ref_por	○	○	○	×
ref_rdv	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
K) メモリプール管理 固定長メモリプール				
cre_mpf/acre_mpf	○	○	×	×
del_mpf	○	×	×	×
get_mpf	○	×	×	×
pget_mpf	○	○	○	×
tget_mpf	○	×	×	×
rel_mpf	○	○	○	×
ref_mpf	○	○	○	×
L) メモリプール管理 可変長メモリプール				
cre_mpl/acre_mpl	○	○	×	×
del_mpl	○	×	×	×
get_mpl	○	×	×	×
pget_mpl	○	○	○	×
tget_mpl	○	×	×	×
rel_mpl	○	○	○	×
ref_mpl	○	○	○	×
M) 時間管理システム時刻管理				
set_tim	○	○	○	×
get_tim	○	○	○	×
isig_tim	×	×	×	○
N) 時間管理周期ハンドラ				
cre_cyc/acre_cyc	○	○	×	×
del_cyc	○	×	×	×
sta_cyc	○	○	○	×
stp_cyc	○	○	○	×
ref_cyc	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
O) 時間管理アラームハンドラ				
cre_alm/acre_alm	○	○	×	×
del_alm	○	×	×	×
sta_alm	○	○	○	×
stp_alm	○	○	○	×
ref_alm	○	○	○	×
P) 時間管理オーバランハンドラ				
def_ovr	○	○	×	×
ivsig_ovr	×	×	×	○
sta_ovr	○	○	○	×
stp_ovr	○	○	○	×
ref_ovr	○	○	○	×
Q) システム状態管理				
rot_rdq/irot_rdq	○	○	○	○
get_tid/iget_tid	○	○	○	○
loc_cpu/iloc_cpu	○	○	○	○
unl_cpu/iunl_cpu	○	○	○	○
dis_dsp	○	×	×	×
ena_dsp	○	×	×	×
sns_ctx	○	○	○	○
sns_loc	○	○	○	○
sns_dsp	○	○	○	○
sns_dpn	○	○	○	○
ref_sys	○	○	○	×

システムコール名	タスク	初期化 ハンドラ	タイム イベント ハンドラ	割込み サービス ルーチン
R) 割込み管理				
def_inh	○	○	○	×
cre_isr/acre_isr	○	○	×	×
del_isr	○	×	×	×
ref_isr	○	○	○	×
dis_int	△	△	△	△
ena_int	△	△	△	△
chg_ims	○	○	○	○
get_ims	○	○	○	○
S) サービスコール管理機能				
def_svc	×	×	×	×
cal_svc	×	×	×	×
T) システム構成管理機能				
def_exc	△	△	△	△
ref_cfg	○	○	○	○
ref_ver	○	○	○	○
U) 独自機能・デバイスドライバ管理機能				
vdef_dev	○	○	×	×
vctr_dev	○	×	×	×

5. μNet3 の API

μ Net3 は弊社 リアルタイム、オペレーティング、システム μ C3 向けに実装された TCP/IP プロトコルスタックです。

μ Net3 はワンチップマイコン向けに最適化されたコンパクトな TCP/IP プロトコルスタックとなっており、また、導入を容易にするために、わかりやすい独自 API を採用しています。

主な機能

- IPv4、ARP、ICMP、IGMPv2、UDP、TCP プロトコルをサポート
- DHCP クライアント、DNS クライアント、FTP サーバー、HTTP サーバー機能が利用可能
- コンフィグレータによる TCP/IP の設定が可能(Compact 版)
- TCP 高速再送/高速復帰アルゴリズムサポート
- IP 再構築とフラグメンテーションサポート
- 複数のネットワーク・インタフェースをサポート

API 一覧

API 名	
A) ネットワーク・インタフェース	
net_ini	TCP/IP プロトコルスタックの初期化
net_cfg	ネットワーク・インタフェースのパラメータ設定
net_ref	ネットワーク・インタフェースのパラメータ参照
B) ネットワーク・デバイス制御	
net_dev_ini	ネットワーク・デバイスの初期化
net_dev_cls	ネットワーク・デバイスの解放
net_dev_ctl	ネットワーク・デバイスの制御
net_dev_sts	ネットワーク・デバイスの状態取得

API 名	
C) ソケット	
cre_soc	ソケットの生成(Standard 版のみ)
del_soc	ソケットの削除(Standard 版のみ)
con_soc	ソケットの接続
cls_soc	ソケットの切断
snd_soc	データの送信
rcv_soc	データの受信
cfg_soc	ソケットのパラメータ設定
ref_soc	ソケットのパラメータ参照
abt_soc	ソケット処理の中止
D) ネットワークアプリケーション	
dhcp_client	DHCP クライアントの開始
ftp_server	FTP サーバーの開始
http_server	HTTP サーバーの開始
CgiGetParam	CGI 引数の解析
HttpSendText	テキストコンテンツの送信
dns_get_ipaddr	ホスト名から IP アドレスの取得
dns_get_name	IP アドレスからホスト名の取得
E) その他	
ip_aton	ドット表記の IPv4 アドレス文字列を 32 ビット値に変換
ip_ntoa	32 ビット値の IPv4 アドレスをドット表記の IPv4 アドレス文字列に変換
ip_byte2n	IPv4 アドレスの配列を 32 ビット値に変換
ip_n2byte	IPv4 アドレスの 32 ビット値を配列に変換
htons	16 ビット値をネットワークバイトオーダーへ変換
ntohs	16 ビット値をホストバイトオーダーへ変換
htonl	32 ビット値をネットワークバイトオーダーへ変換
ntohl	32 ビット値をホストバイトオーダーへ変換