Web サーバの構築

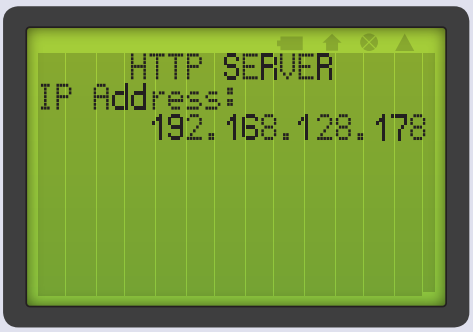
学習内容

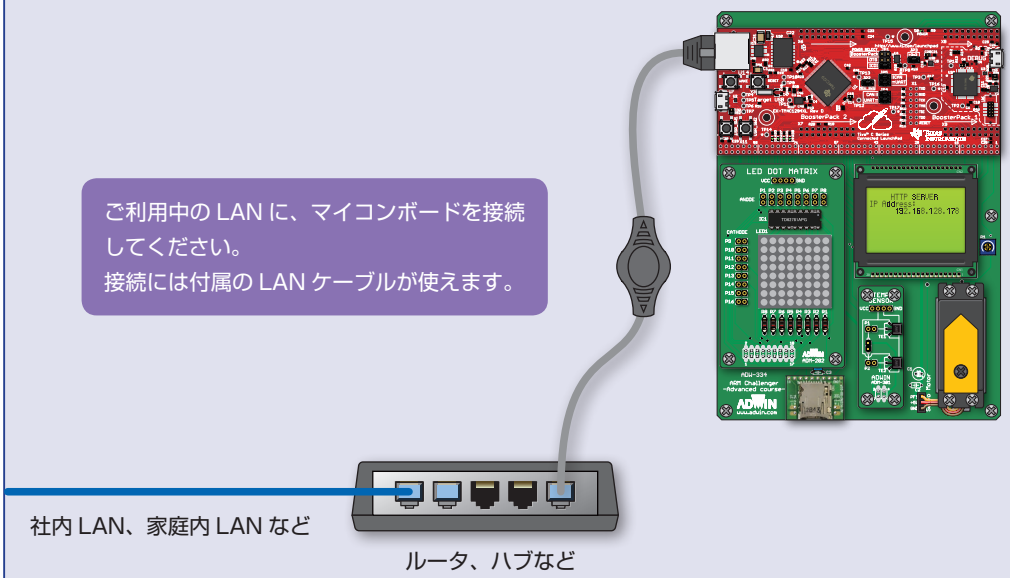
ARM マイコンで Web サーバを構築します。
Web サーバとは、Web ページを提供するサーバです。クライアントからのリクエストに対して、要求された Web ページ（HTML ファイルなど）をクライアントに返します。

課題 08-1

マイコンをネットワークに接続し、GLCD にマイコンの IP アドレスを表示させましょう。

IP アドレスは LAN 内のサーバやルータから DHCP で自動的に割り振られた番号を使います。ご不明な場合は、ネットワーク管理者にお問い合わせください。





Web サーバの構築

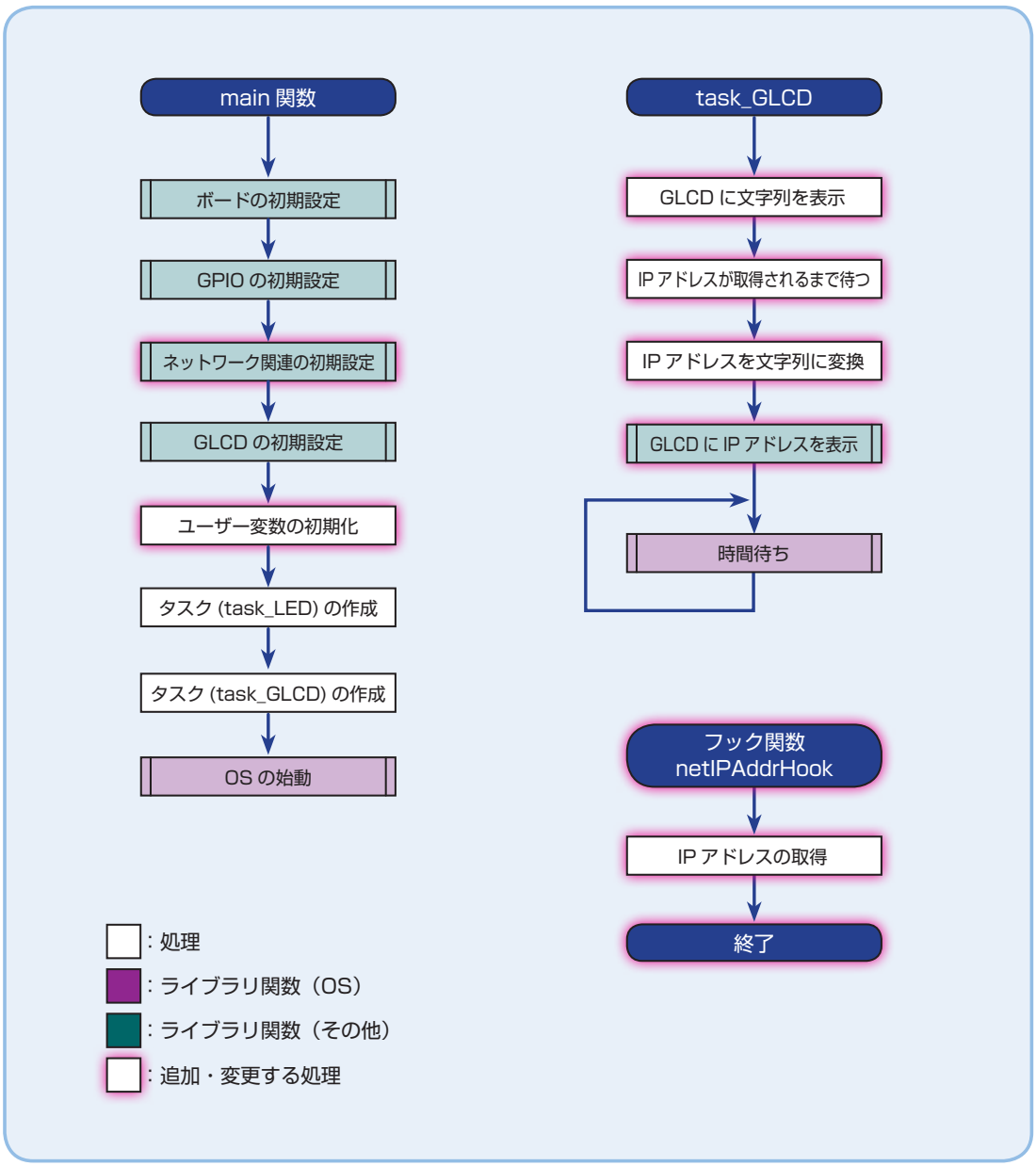
Network Developer's Kit（NDK）【TI-RTOS】

TI-RTOS には、ネットワーク開発に便利な Network Developer's Kit (NDK) という開発プラットフォームが用意されています。TCP/IP ネットワークを扱うプログラミングが可能であるほか、Web サーバを簡単に実現できる HTTP サーバモジュールも用意されています。

なお、NDK についての詳細は、CCS の「Help」→「Help Contents」から「TI-RTOS for TivaC*」→「Documentation Links」→「Networking Documentaion」をご覧ください。

フローチャート 08-1

以下は、課題 08-1 を実現するためのフローチャート例です。IP アドレスは、マイコンが IP アドレスを取得した時に呼び出される「フック関数」によって取得することができます。その他のタスクは「フローチャート 07」と同じです。



Web サーバの構築

初期設定 08-1

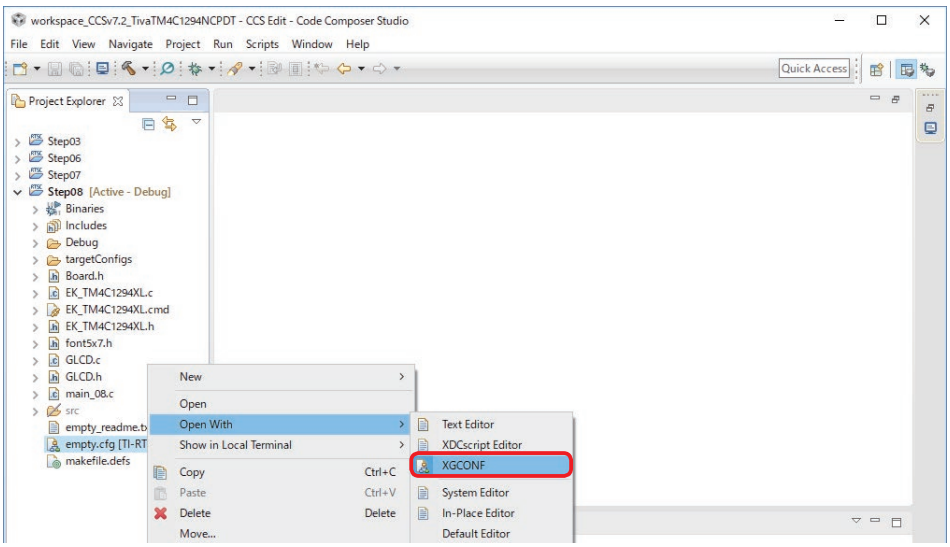
1. cfg ファイルの設定

ネットワークに関するモジュールの有効化等、OS の設定は cfg ファイルで行います。設定は cfg ファイルを平文で編集することでも行えますが、CCS の「XGCONF」を使うとグラフィカルに設定を行うことが可能です。

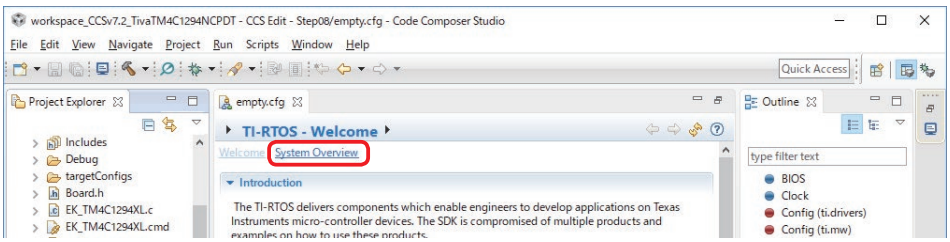
HTTP サーバモジュールの有効化

まずは、HTTP サーバモジュールを有効化します。なお、本コースではプロトコルとして IPv4 を使うものとします。

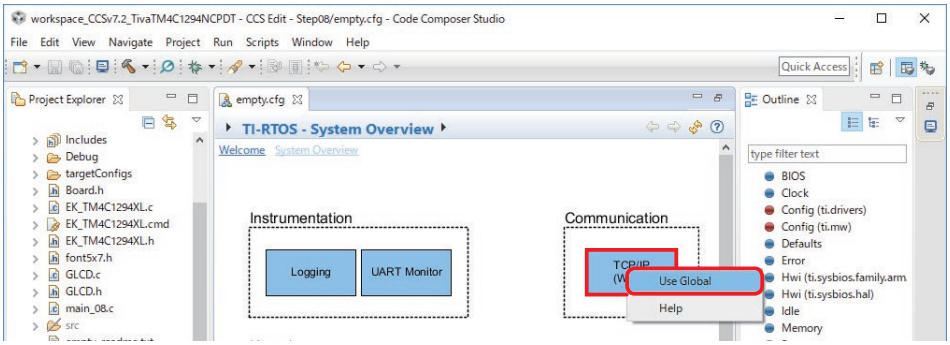
❶「empty.cfg」を右クリックし、「Open With」→「XGCONF」を選択します。



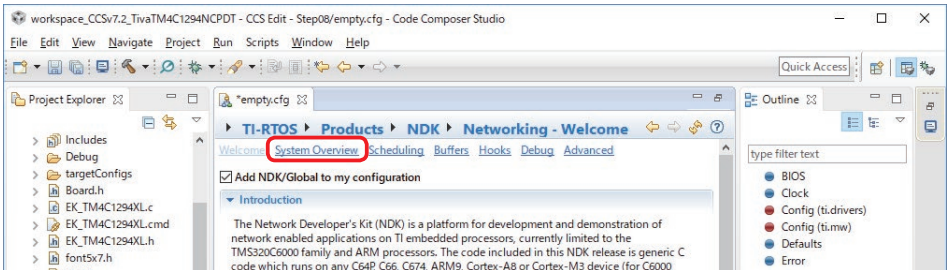
❷「System Overview」を開き、



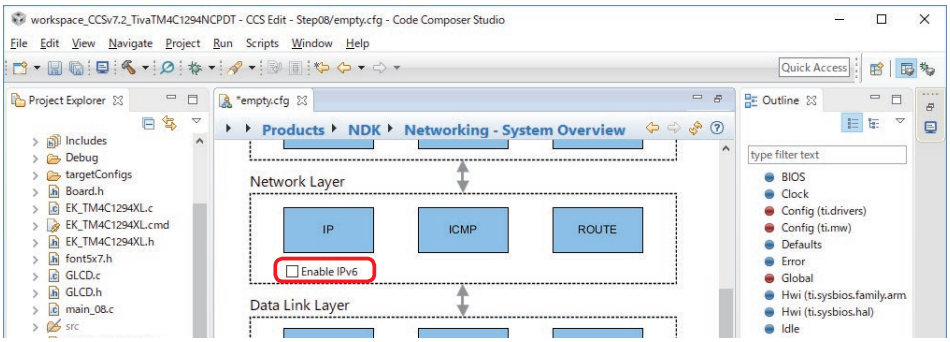
❸「Communication」内の「TCP/IP (Wired)」を右クリックして「Use Global」を選択します。



❹「Networking welcome」が開いたら、ページ内の「System Overview」を開きます。



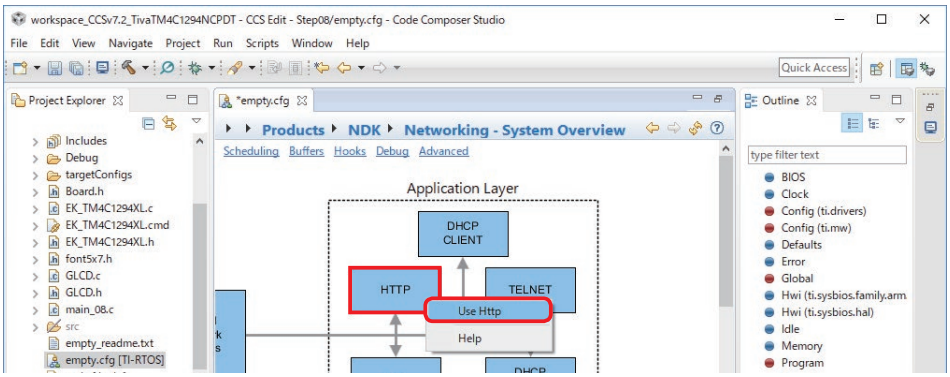
❺「Network Layer」内の「Enable IPv6」のチェックを外します。



Web サーバの構築

初期設定 08-1

❻「Application Layer」内の「HTTP」を右クリックし、「Use Http」を選択します。



以上の操作によって、「empty.cfg」ファイルには以下の行が追加されることになります※1,2。

```
var Global = xdc.useModule('ti.ndk.config.Global');
var Http = xdc.useModule('ti.ndk.config.Http');
Global.IPv6 = false;
```

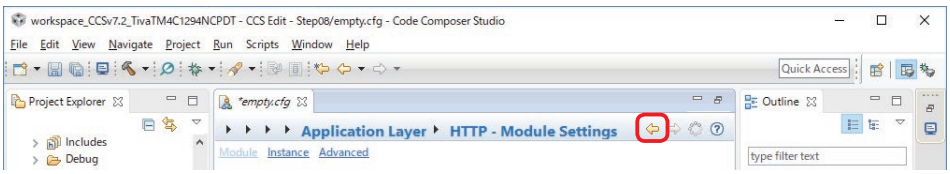
※1　empty.cfg ファイルの中身は、ページ下の「cfg Script」タブから確認することができます。

※2　追加された行は empty.cfg ファイルの途中や末尾に分散して配置される場合がありますが、実際にはファイルのどこに配置してもかまいません。XGCONF では必ずしも適切な場所に行を配置させるわけでもないようなので、気になるようでしたらファイルの途中に追加された行をファイル末尾に移動させてしまってもかまいません。

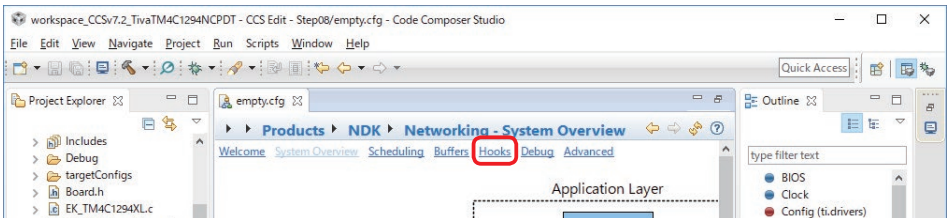
フック関数の追加

NDK では、特定のタイミングで呼び出される「フック関数」を設定することができます。ここでは、IP アドレスが取得された時にフック関数が呼び出されるようにします。

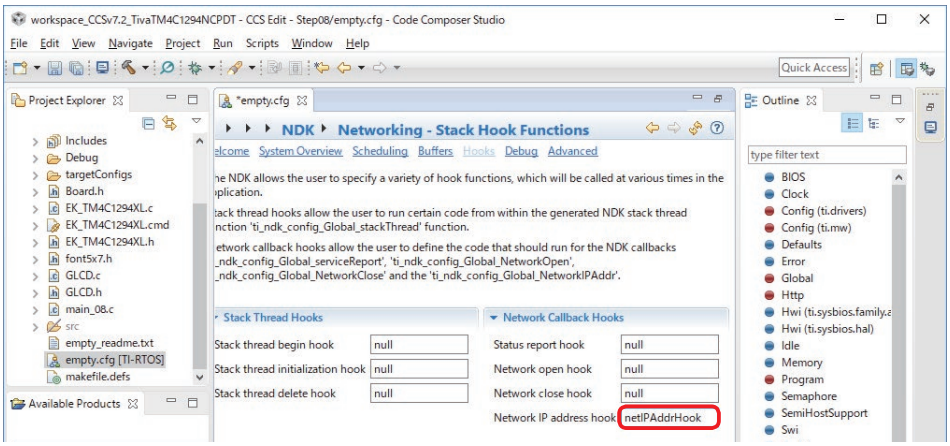
❼ ページ右上の「Back」アイコン をクリックし、「Networking – System Overview」に戻ります。empty.cfg タブを閉じてしまった場合は、１、２、３の操作を再度行ってください。



❽ ページ上の「Hooks」を選択し、「Nertworking – Stack Hook Functions」を開きます。



❾「Network IP address hook」を「netIPAddrHook」と設定します。



以上の操作によって、「empty.cfg」ファイルには以下の行が追加されることになります。

```
Global.networkIPAddrHook = "&netIPAddrHook";
```

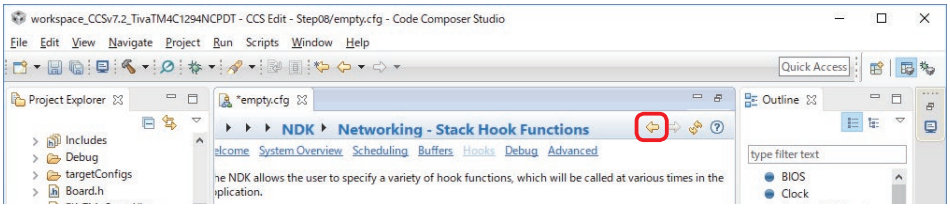
Web サーバの構築

初期設定 08-1

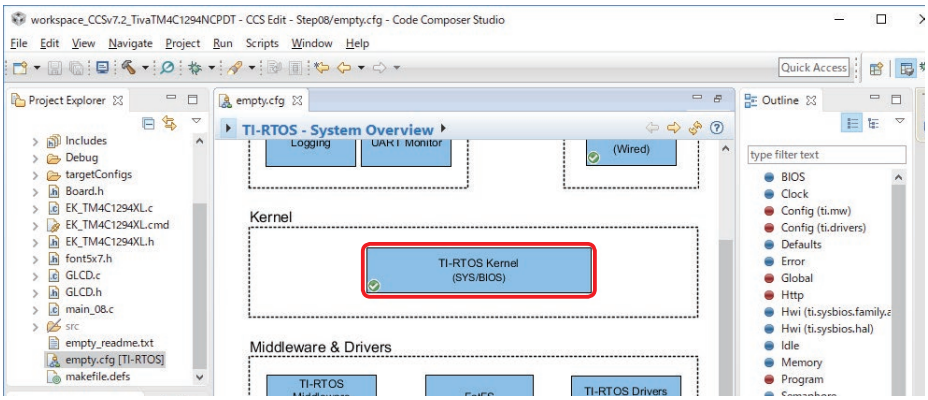
OS のヒープ領域の拡張

NDK を使うには、さらに OS のヒープ領域を拡張する必要があります。

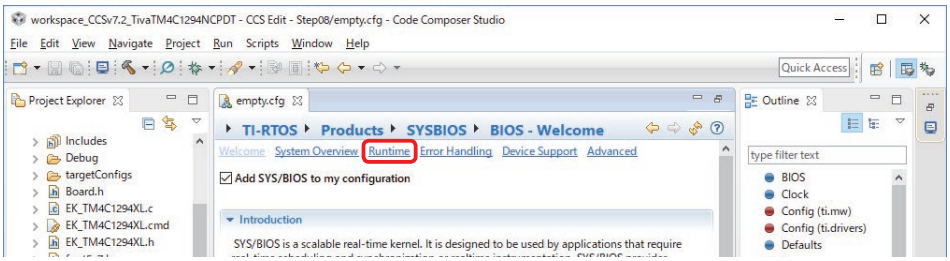
⑩ ページ右上の「Back」アイコン をクリックし、「TI-RTOS – System Overview」に戻ります。



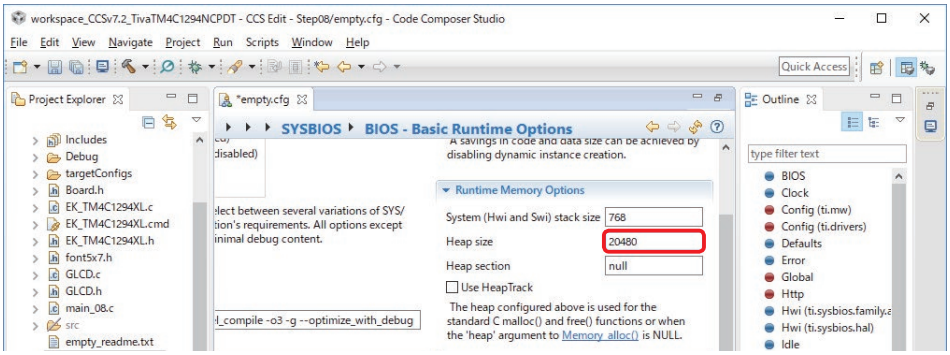
⑪「Kernel」内の「TI-RTOS Kernel (SYS/BIOS)」をクリックします。



⑫「BIOS – Welcome」ページが開いたら、ページ上の「Runtime」を選択します。



⑬「Runtime Memory Options」の「Heap size」を「20480」に設定します。



以上の操作によって、「empty.cfg」ファイルは、OS のヒープ領域に関する設定が以下のように変更されることになります。

```
/*
* Specify default heap size for BIOS.
*/
BIOS.heapSize = 20480;
```

ここまでの設定が完了したら、「Save」アイコン をクリックして cfg ファイルを保存しましょう。

2. フック関数の作成

追加したフック関数は、ソースコード中に実体を記述していないとコンパイルエラーになるため、とりあえずフック関数の実体を作っておきましょう。

まずは、NDK のヘッダファイルをインクルードしておきましょう。以下は記述例です。

```
13 /* NDK ヘッダファイル */
14 #include <ti/ndk/inc/netmain.h>
```

以下はフック関数の記述例です。

```
94 /*
95 * フック関数
96 */
97
98 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
99 {
100
101 }
```

IP アドレスが取得された時に呼び出されるフック関数（ここでは netIPAddrHook）は、上記記述例のように 3 つの引数を持ち、このうち最初の引数（ここでは IPAddr）に IP アドレスが上位バイトから格納されています（ビッグエンディアン）。

リトルエンディアン と ビッグエンディアン

ネットワーク関連の 2 バイト以上のデータは、通常は上位バイトから送信され（ビッグエンディアン）、一方でネットワーク以外の分野では 2 バイト以上のデータは下位バイトから扱われるのが一般的です（リトルエンディアン）。そのため、ソケットプログラミングでは設定値等のバイトオーダーを逆にする必要がしばしばあります。

フック関数の引数に格納されている IP アドレスは、ネットワークのバイトオーダーに合うように上位バイトから格納されており（ビッグエンディアン）、例えば「192.168.0.1」（16 進数で 0xC0A80001）という IP アドレスの場合、引数に格納されている数値は「0x0100A8C0」になります。

3. イーサネットと MAC アドレスの初期設定

main 関数内のボードの初期設定部分に、以下のように Board_initEMAC 関数を追加しましょう。

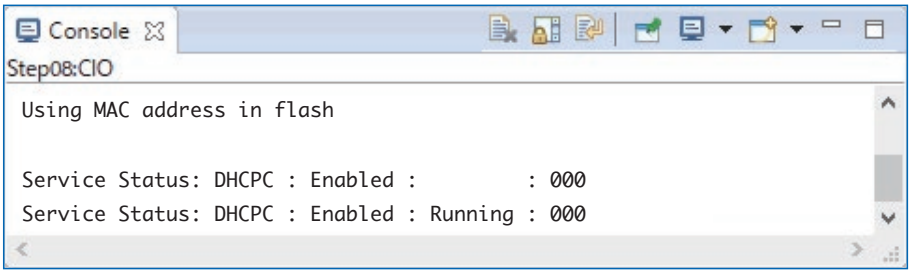
```
125  // ボードの初期設定
126  CONSOLE(" ボードの初期設定 ¥n");
127  Board_initGeneral();
128  Board_initGPIO();
129  Board_initEMAC();
```

この関数は、MAC アドレスの初期設定等を行います。

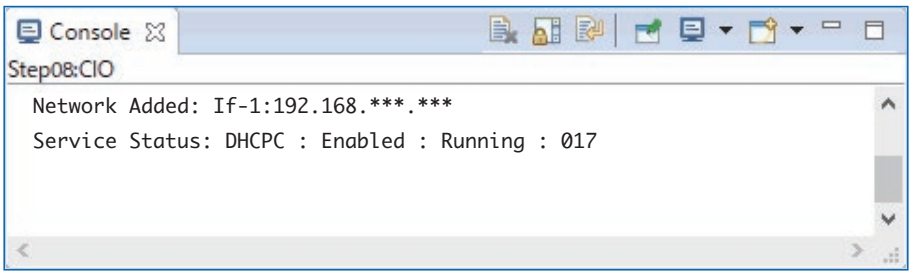
4. ネットワーク接続の確認

以上の初期設定が完了し、コンパイルが完了したら、ネットワーク接続の確認を行ってみましょう。

マイコンボードの LAN ポートに LAN ケーブルを接続し、デバッグを開始しましょう。プログラムを開始すると、コンソールに以下のようなメッセージが表示されます。



その後、しばらくするとネットワークへの接続が完了し、以下のような IP アドレスを含むメッセージが表示されます。なお、ネットワークの状況等によっては、接続までに時間がかかることがあるのでご注意ください。



同じネットワーク内に接続されている PC 等から、ping 等でこのアドレスにアクセスし、反応があることを確認してみましょう。

Web サーバの構築

ping によるネットワーク接続の確認方法【Windows】

ping とは

インターネットなどの TCP/IP ネットワークを診断するプログラムです。
接続されているかどうか調べたい相手の IP アドレスを指定すると、通常 32 バイト程度のデータを送信し、相手から返信があるかどうか、返信がある場合はどのくらい時間がかかっているか、などのデータを元にネットワークの接続状況を診断することができます。

ping の実行方法

ping を実行させるには Windows でコマンドプロンプトを起動し、

　　ping ***.***.***.***

と入力します。このとき、「***.***.***.***」はマイコンに設定した IP アドレスを入力します。きちんと接続されている場合、

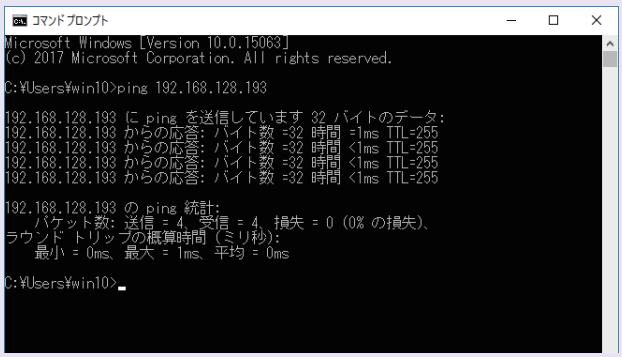
　　Reply from ***.***.***.*** : バイト数 =32　時間 <1ms　TTL=255

のように表示されます。

表示される「時間」や「TTL」の値は接続方法によって異なります。
接続に問題がある場合は

　Request timed out.

と表示されます。このような表示が出た場合、マイコンやパソコンの LAN 接続やマイコンの IP アドレスなどの設定をチェックしてください。



インクルードファイル 08-1

課題 08-1 で使用するインクルードファイルを解説します。

なお、インクルードされているファイルは、テキストカーソルをファイル名部分に合わせて「F3」キーを押すことで開くことができます。

NDK ヘッダファイル　ti/ndk/inc/netmain.h

NDK の型や関数等を使うには、このヘッダファイルをインクルードします。なお、このヘッダファイル自体の内容は他のヘッダファイルのインクルードなので、実際の定義等を見るにはインクルードされているファイルを辿っていってください。

NDK 関連のヘッダファイルおよびソースファイルは、「 C:\ti\tirtostivac_*\products\tidrivers_tivac_*\packages 」内にあります。

型 08-1

課題 08-1 で使用する型を解説します。

なお、テキストカーソルをソースコード中の関数の部分に合わせると、その型・構造体に関する情報がポップアップされます。さらに「F3」キーを押すと、その型・構造体が定義されているファイルを開くことができます。

IPN, uint

これらは NDK のヘッダファイルで定義されており、IPN は IP アドレス格納用の型、uint は符号無し整数型です。実体はいずれも 32 ビット符号無し整数です。

Web サーバの構築

ライブラリ関数 08-1

課題 08-1 で使用する関数を解説します。

なお、テキストカーソルをソースコード中の型や構造体の部分に合わせると、その関数に関する情報がポップアップされます。さらに「F3」キーを押すと、その関数が定義されているファイルを開くことができます。

NDK 関係

IP アドレスの文字列への変換　void NtIPN2Str(IPN IPAddr, char *pStrBuffer)

IPAddr で指定したネットワークフォーマット（ビッグエンディアン）の 32 ビット IP アドレスを、文字列に変換して pStrBuffer に代入します。なお、変換後の文字列はヌル文字を含めて 16 文字以内になります。

```
NtIPN2Str(IP, IP_Address);
```

引数の設定例

- IP　：ネットワークフォーマット（ビッグエンディアン）の IP アドレス。
- IP_Address　：文字列を格納する配列へのポインタ。16 文字以上確保していること。

詳細は CCS の「Help」→「Help Contents」より「TI-RTOS for TivaC *」→「Documentation Links」→「Networking Documenation」→「TI-RTOS Networking API Reference Guide」を開き、「5Network Tools Library - Support Functions」をご覧ください。

イーサネット、MAC アドレスの初期化　void Board_initEMAC(void)

MAC アドレスの設定、ネットワークモニタリング用の LED3(D3)、LED4(D4) の設定、その他ネットワークを利用する上で必要な設定を行います。関数の実体は、「EX_TM4C1294XL.c」中の「EK_TM4C1294XL_initEMAC」に書かれています。

```
Board_initEMAC();
```

コーディング main.c 08-1

以下は、main.c 07 のソースコードを元にしたコーディング例です。■は main.c 07 から追加・変更した部分です。

main.c

```
/* XDCtools ヘッダファイル */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS ヘッダファイル */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>

/* ドライバヘッダファイル */
#include <ti/drivers/GPIO.h>

/* NDK ヘッダファイル */
#include <ti/ndk/inc/netmain.h>

/* ボードヘッダファイル */
#include "Board.h"

/* GLCD ヘッダファイル */
#include "GLCD.h"

/* ユーザー定義マクロ */
#define CONSOLE(...) do { System_printf(__VA_ARGS__); System_flush(); } while(0)
#define SLEEP(X) Task_sleep((X)*1000/Clock_tickPeriod)

/*
 * ユーザータスクの優先度
 */
#define TASK_LED_PRIO 1
#define TASK_GLCD_PRIO 9

/*
 * ユーザースタックのサイズ
 */
#define TASK_LED_STACK 512
#define TASK_GLCD_STACK 512

/*
 * ユーザータスクの構造体
 */
Task_Struct taskLEDStruct;
Task_Struct taskGLCDStruct;

/*
 * ユーザータスクのスタック
 */
Char taskLEDStack[TASK_LED_STACK];
Char taskGLCDStack[TASK_GLCD_STACK];

```

Web サーバの構築

コーディング main.c 08-1

```
/*
 * ユーザー変数
 */
uint32_t IP; // IP アドレス格納用変数

/*
 * ユーザータスク
 */

/*
 * LED 点滅タスク
 */
Void task_LED(UArg arg0, UArg arg1)
{
    CONSOLE("LED 点滅タスクの開始 \n");

    // LED1 を点灯
    GPIO_write(Board_LED0, Board_LED_ON);

    while (1)
    {
        // LED を 0.5 秒おきにトグル
        SLEEP(500);
        GPIO_toggle(Board_LED0);
    }
}

/*
 * GLCD 表示タスク
 */
Void task_GLCD(UArg arg0, UArg arg1)
{
    char IP_Address[16]; // IP アドレス表示用配列

    CONSOLE("GLCD 表\示タスクの開始 \n");

    // 各文字列を GLCD に表示
    GLCD_str(0, 5, "HTTP SERVER");
    GLCD_str(1, 0, "IP Address:");

    // IP アドレスが取得されるまで待つ
    while (IP == 0) SLEEP(100);

    // IP アドレスを文字列に変換
    NtIPN2Str(IP, IP_Address);

    // IP アドレスを GLCD に表示
    GLCD_str(2, 6, IP_Address);

    while (1)
    {
        // 時間待ち
        SLEEP(1000);
    }
}
```

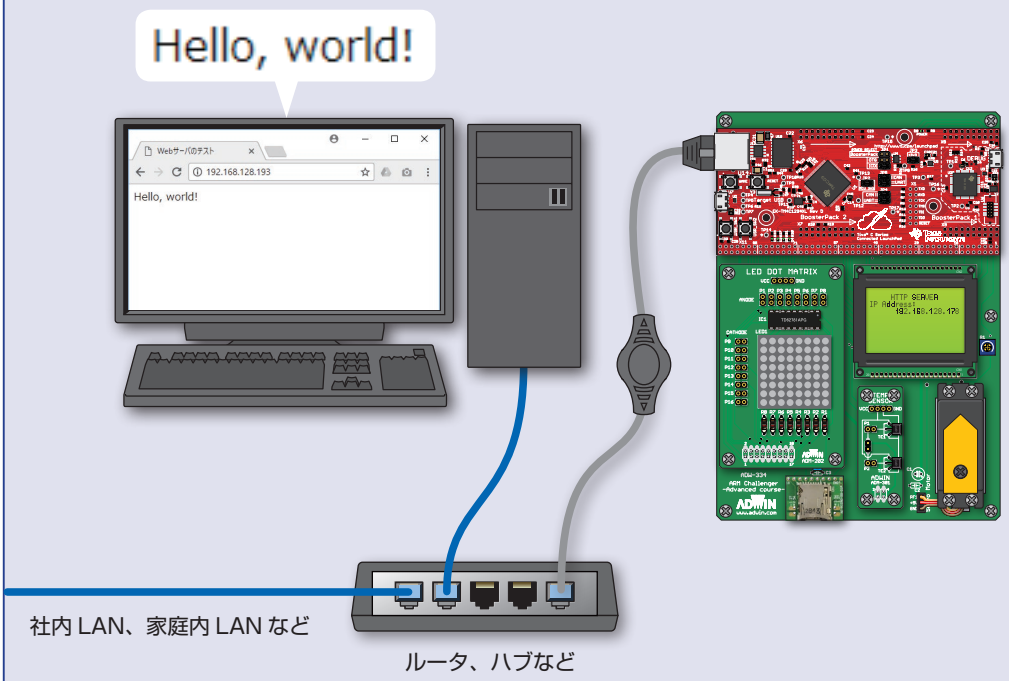
```
105
106 /*
107 * フック関数
108 */
109
110 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
111 {
112     /* IP アドレスの取得 */
113     IP = IPAddr;
114     CONSOLE("IP アドレス : %x\n", IP);
115 }
116
117 /*
118  * メイン
119  */
120
121 int main(void)
122 {
123     Task_Params taskParams;
124
125     // ボードの初期設定
126     CONSOLE(" ボードの初期設定 \n");
127     Board_initGeneral();
128     Board_initGPIO();
129     Board_initEMAC();
130
131     // GLCD の初期化
132     GLCD_init();
133
134     // ユーザー変数の初期値の設定
135     IP = 0;
136
137     // ユーザータスクの作成
138     CONSOLE(" ユーザータスクの作成 \n");
139     Task_Params_init(&taskParams);
140     taskParams.stackSize = TASK_LED_STACK;
141     taskParams.stack = &taskLEDStack;
142     taskParams.priority = TASK_LED_PRIO;
143     Task_construct(&taskLEDStruct, (Task_FuncPtr) task_LED, &taskParams, NULL);
144
145     Task_Params_init(&taskParams);
146     taskParams.stackSize = TASK_GLCD_STACK;
147     taskParams.stack = &taskGLCDStack;
148     taskParams.priority = TASK_GLCD_PRIO;
149     Task_construct(&taskGLCDStruct, (Task_FuncPtr) task_GLCD, &taskParams, NULL);
150
151     // OS の起動
152     CONSOLE("OS の起動 \n");
153     BIOS_start();
154
155     return (0);
156 }
157
```

Web サーバの構築

課題 08-2

1. パソコンのブラウザからマイコンにリクエストを送信しましょう。
   例）http://192.168.128.178/

2. ソースファイル（main.c）内に書かれた HTML 文を返信し、ブラウザに「HELLO WORLD」と表示させましょう。



ソケットプログラミング

Web サーバを構築するには、「ソケットプログラミング」と呼ばれる方法を用います。

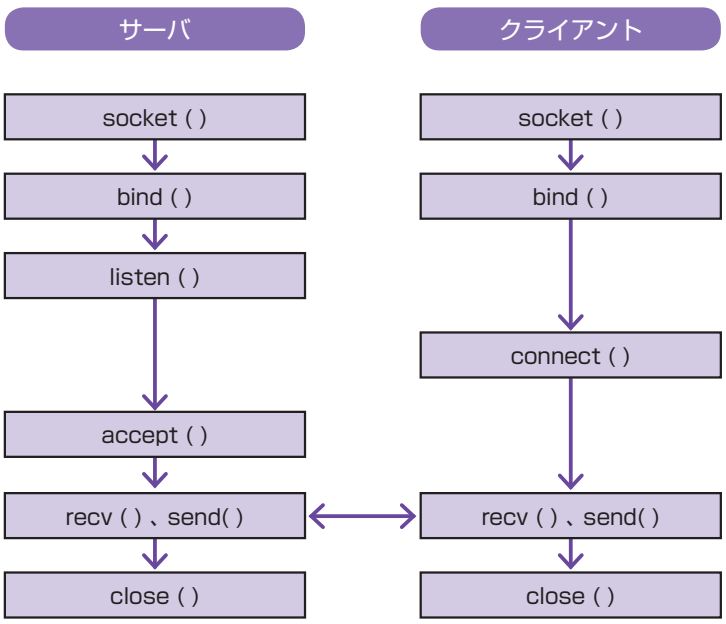
ソケットの概念

ソケット (Socket) とは、ネットワーク上でアドレス指定が可能な通信接続ポイント (端点) のことで、TCP/IP アプリケーションを作成するための抽象化されたインターフェースです。

ソケットの機能

ソケットには定型のイベント・フローがあります。
サーバ (マイコン) はまず、クライアント (今回の例ではブラウザ) がサーバを探せるようにアドレスを確立（バインド）します。アドレスが確立されると、サーバはクライアントからのサービス要求待ちになります。クライアントからの要求がサーバに届くことで、サーバとクライアントの間で接続が確立します。サーバとクライアントの間のデータ交換は、クライアントがソケットを経由してサーバに接続しているときに行われます。このときに、サーバはクライアントの要求を受けて処理を実行し、クライアントに応答を送信します。

以下の図は、ソケットプログラミングの典型的なイベント・フローを表しています。
TI-RTOS の HTTP サーバモジュールは、このイベント・フローが組み込まれており、ユーザーはデータを準備するだけで済みます。クライアント側のソケットプログラミングは Web ブラウザに実装済です。



Web サーバの構築

HTTP 通信

HTTP とは

HTTP とは HyperText Transfer Protocol の略で、Web サーバとブラウザなどとの通信に使用されているプロトコルです。通信には TCP を用いており、テキストベースでメッセージのやり取りを行っています。実際にブラウザが Web サーバにアクセスするときには、以下のようなメッセージをサーバに送っています。

HTTP リクエスト　例）

```
GET / HTTP/1.1
Accept: image/gif,  image/jpeg, image/pjpeg, ・・・
Accept-Language: ja
User-Agent: Mozilla/4 .0 (compatible;  MSIE 8.0; ・・・
Accept-Encoding:  gzip, deflate
Host: xxx.xxx.xxx.xxx
```

これらのメッセージは HTTP と呼ばれており、HTTP 通信におけるさまざまな情報が記されています。
さらに、ブラウザが送信する HTTP を「HTTP リクエスト」と呼び、サーバが送信するメッセージの HTTP を「HTTP レスポンス」と呼びます。

この HTTP リクエストの中で重要なのは、「GET /」です。
「GET」はメソッドと呼ばれる命令の一種で、サーバに対して「/」以降に記されているファイルの取得を要求していることを示します。

サーバはこの「GET」メッセージを受け取ったら「/」以降に記されているファイルを HTTP レスポンスを付けてブラウザに向けて送信しなければなりません。
サーバが送信する HTTP レスポンスの例は以下のようになります。

HTTP レスポンス　例）

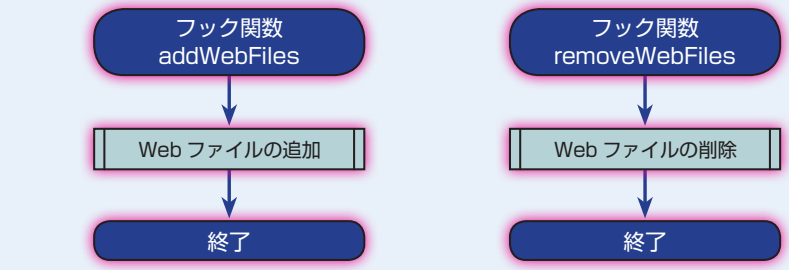
```
HTTP/1.1  200 OK.
Date: Tue, 05 Oct 2010  08:18:54 GMT
Server: Apache/2.2 .3 (Red Hat)
Accept-Ranges: bytes
Content-Length: 50206
Content-Type: text/html
Connection:  keep-alive

<html>\r\n・・・
```

「200 OK.」は HTTP ステータスコードの一つで、クライアントからの要求が正常に受信し処理されたことを示しています。
「Content-Type: text/html」は返信される文字列が html 文であることを示しています。

TI-RTOS の HTTP サーバモジュールは、HTTP リクエストの解析と HTTP レスポンスの送信を自動で行ってくれます。

フローチャート 08-2

ここでは、HTTP サーバモジュールを用いて Web サーバを構築することとします。以下は、課題 08-2 を実現するためのフローチャート例です。フック関数で Web ファイルの追加・削除を行っています。その他のタスクは「フローチャート 08-1」と同じです。



：処理
：ライブラリ関数（その他）
：追加・変更する処理

Web サーバの構築

初期設定 08-2

cfg ファイルの設定

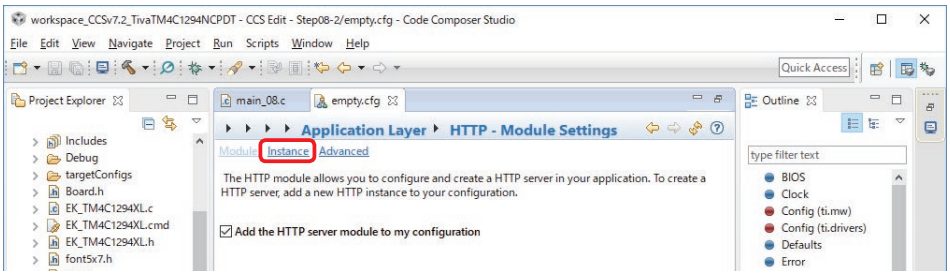
以下は、初期設定 08-1 からの empty.cfg ファイルの設定例です。

HTTP サーバモジュールの有効化

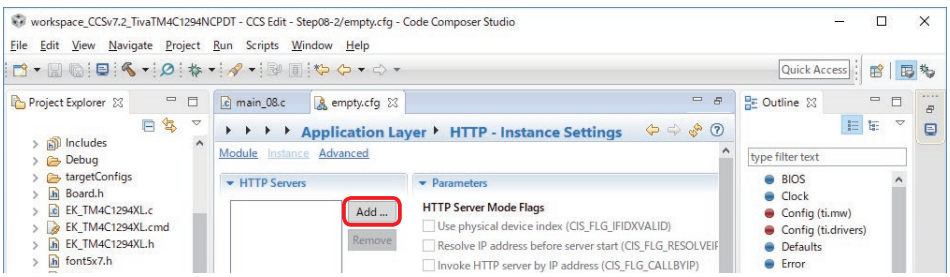
❶「empty.cfg」を開きます※。

❷「System Overview」→「Communication」内の「TCP/IP (Wired)」→「System Overview」→「Application Layer」内の「HTTP」を開きます。

❸「HTTP – Module Settings」が開いたら、左上から「Instance」を選択します。

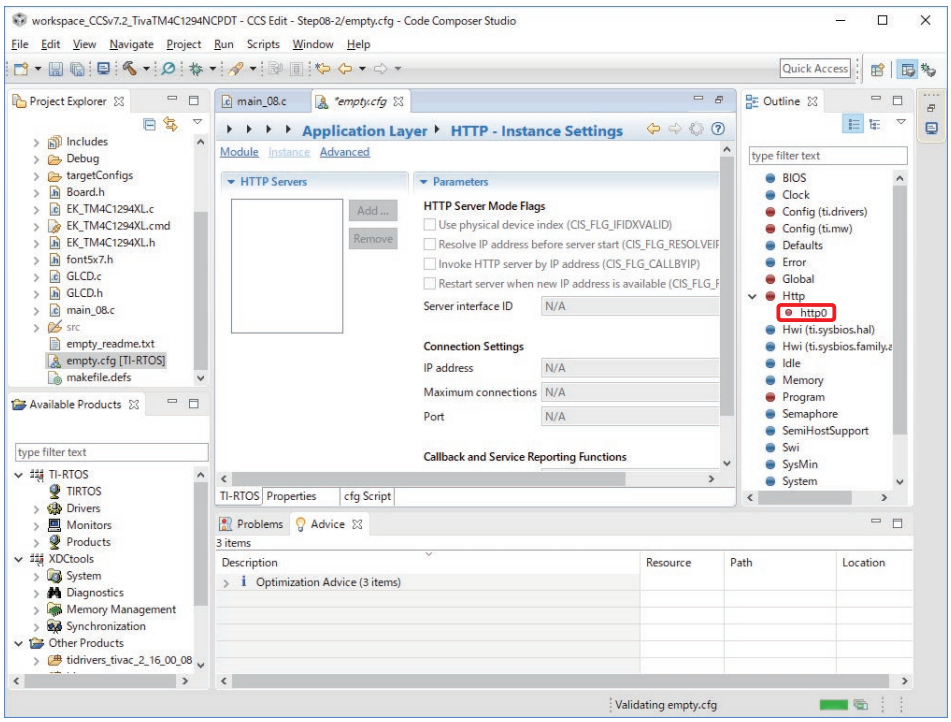


❹「HTTP – Instance Settings」ページが開いたら、「HTTP Servers」の「Add ...」をクリックし、



※ 右クリックで「Open With」→「XGCONF」を選択しても構いませんが、前回「XGCONF」で開いている場合、ダブルクリックするだけで GUI の XGCONF で開くことができます。

❺「HTTP Servers」に「http0」が追加されたことを確認します。



以上の操作によって、「empty.cfg」ファイルには以下の行が追加されることになります。

```
var http0Params = new Http.Params();
var http0 = Http.create(http0Params);
```

Web サーバの構築

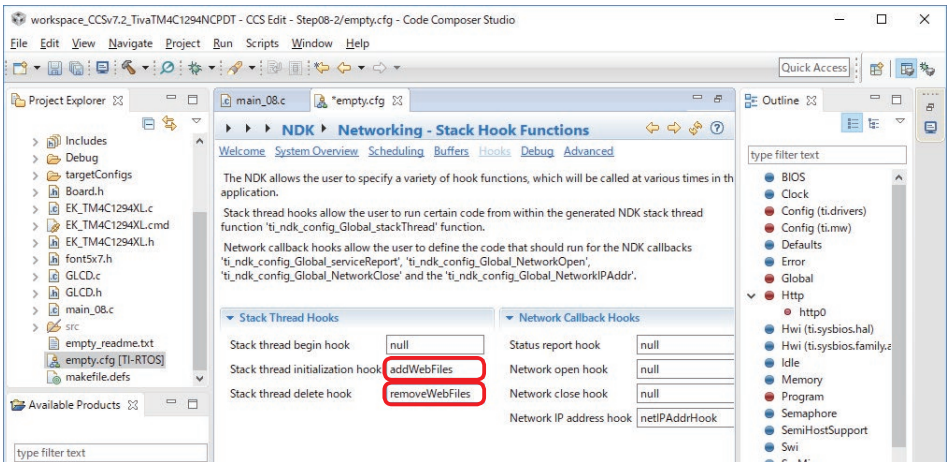
初期設定 08-2

フック関数の追加

❻ページ右上の「Back」アイコン をクリックし、「Networking – System Overview」に戻ります。

❼ ページ上の「Hooks」を選択し、「Nertworking – Stack Hook Functions」を開きます。

❽「Stack Thread Hooks」の「Stack thread initialization hook」に「addWebFiles」を「Stack thread delete hook」に「removeWebFiles」を設定します。



以上の操作によって、「empty.cfg」ファイルには以下の行が追加されることになります。

```
Global.stackInitHook = "&addWebFiles";
Global.stackDeleteHook = "&removeWebFiles";
```

ここまでの設定が完了したら、「Save」アイコン をクリックして cfg ファイルを保存しましょう。

ライブラリ関数 08-2

課題 08-2 で使用する関数を解説します。

なお、テキストカーソルをソースコード中の型や構造体の部分に合わせると、その関数に関する情報がポップアップされます。さらに「F3」キーを押すと、その関数が定義されているファイルを開くことができます。

以下の関数についての詳細は、CCS の「Help」→「Help Contents」より「TI-RTOS for TivaC*」→「Documentation Links」→「Networking Documenation」→「TI-RTOS Networking API Reference Guide」を開き、「2 Operating System Abstraction API」→「2.6 File I/O Support for Embedded Systems」→「2.6.2 EFS Custom API Functions」をご覧ください。

NDK 関係

RAM ベースファイルの作成　void efs_createfile(char *name, INT32 length,UINT8 *pData)

アドレス pData で始まる大きさ length のデータを、name という名前の Web ファイルとして追加します。

```
efs_createfile("index.html", strlen(page), (UINT8 *)page);
```

引数の設定例

- `"index.html"`　：ファイル名。
- `strlen(page)`　：データの大きさ。ここでは文字列 page の長さ。
- `(UINT8 *)page`　：データを格納している配列等（ここでは文字列）のポインタ。

RAM ベースファイルの破棄　void efs_destroyfile(char *name)

efs_createfile関数で作成したWeb ファイル「name」を破棄します。

```
efs_destroyfile("index.html");
```

引数の設定例

- `"index.html"`　：破棄する Web ファイルのファイル名。

Web サーバの構築

コーディング main.c 08-2

以下は、main.c 08-1 のソースコードを元にしたコーディング例です。■は main.c 08-1 から追加・変更した部分です。

main.c

これ以前の行に変更はありません

```
106 /*
107 * フック関数
108 */
109
110 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
111 {
112     /* IP アドレスの取得 */
113     IP = IPAddr;
114     CONSOLE("IP アドレス : %x\n", IP);
115 }
116
117 Void addWebFiles()
118 {
119     static char page[] = "<!DOCTYPE html>\n"
120             "<html>\n"
121             "<head>\n"
122             "<meta charset='Shift_JIS'>\n"
123             "<title>Web サーバのテスト </title>\n"
124             "<body>\n"
125             "Hello, world!\n"
126             "</body>\n"
127             "</html>\n";
128
129     /* Web ファイルの追加 */
130     efs_createfile("index.html", strlen(page), (UINT8 *) page);
131 }
132
133 Void removeWebFiles()
134 {
135     /* Web ファイルの削除 */
136     efs_destroyfile("index.html");
137 }
138
139 /*
140  * メイン
141  */
```

これ以降の行に変更はありません

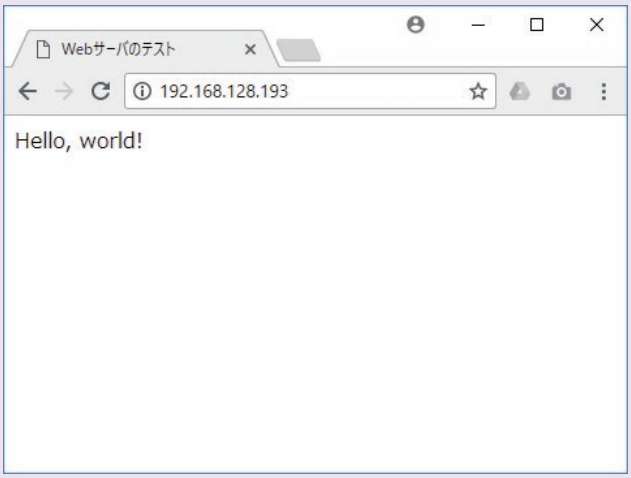
ソケットプログラミングによる Web サーバの構築

動作結果は「課題 08-2」と同じですが、ソケットプログラミングによって Web サーバを構築する方法についても試してみましょう。

ブラウザからの ARM マイコンへのアクセス方法

ブラウザを起動後、アドレス欄に「http://***.***.***.*** (マイコンに設定した IP アドレス)」を入力すれば、マイコンにアクセスすることができます。右画像は Google Chrome の表示例です。

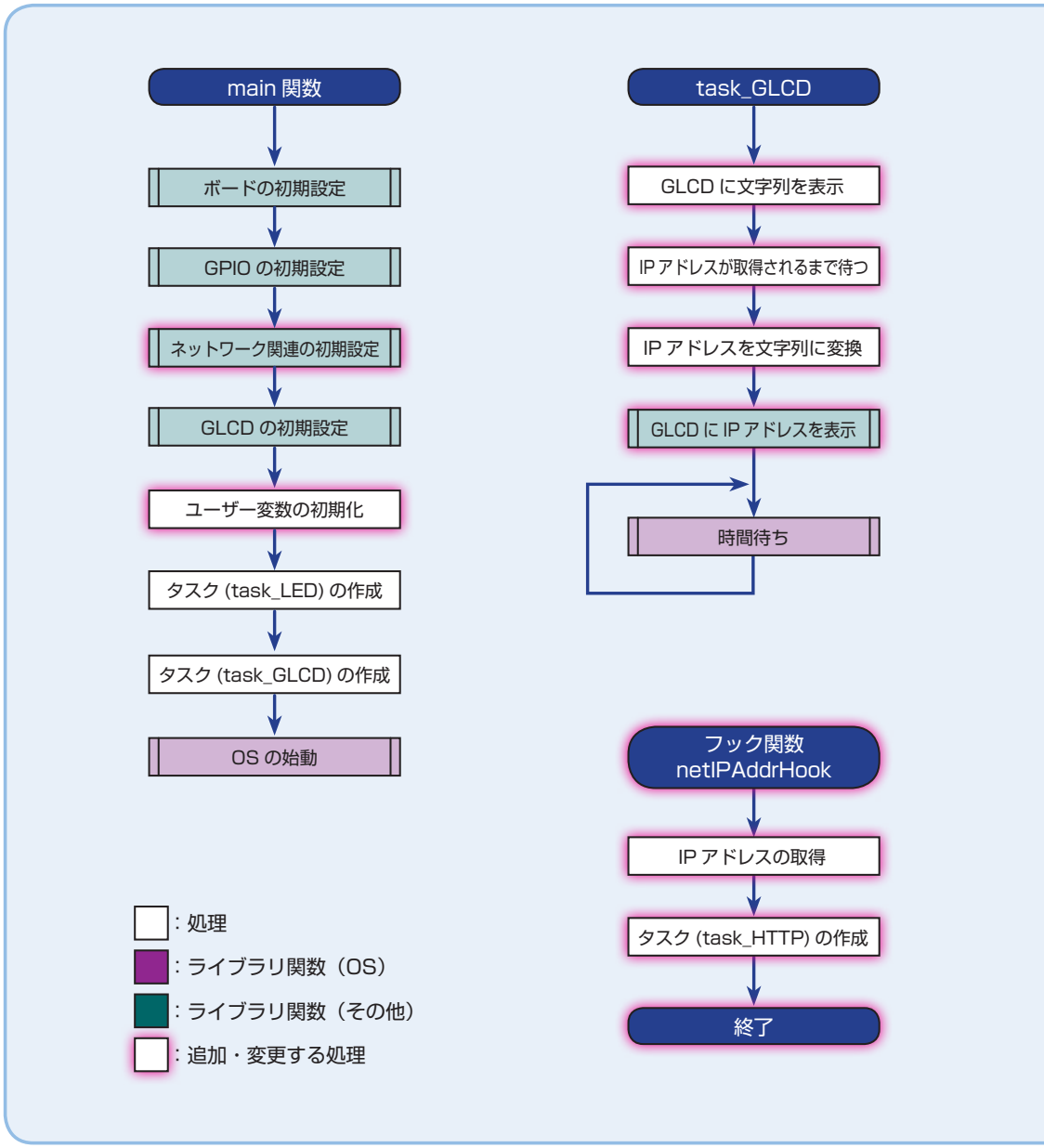
正常に表示されない場合は、ping を使ってネットワーク接続が正常か確認を行ったり、マイコンのプログラムのチェックしてください。

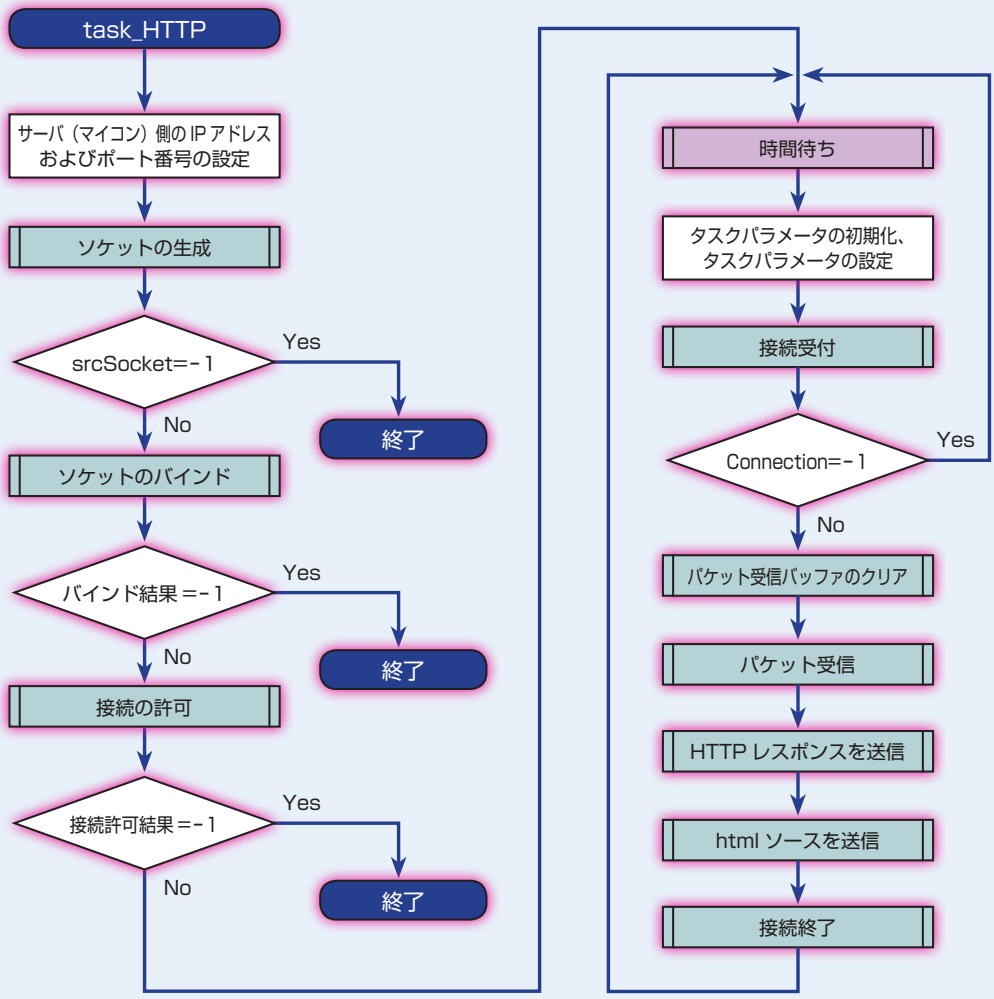


Web サーバの構築

フローチャート 08s

以下は、フローチャート例です。HTTP タスクは、マイコンが IP アドレスを取得した時に呼び出されるフック関数 netIPAddrHook 内で作成しています。その他のタスクは「フローチャート 07」と同じです。




task_HTTP
サーバ（マイコン）側の IP アドレス
およびポート番号の設定
ソケットの生成
srcSocket=－1
Yes
終了
No
ソケットのバインド
バインド結果 =－1
Yes
終了
No
接続の許可
接続許可結果 =－1
Yes
終了
No
時間待ち
タスクパラメータの初期化、
タスクパラメータの設定
接続受付
Connection=－1
Yes
No
パケット受信バッファのクリア
パケット受信
HTTP レスポンスを送信
html ソースを送信
接続終了

Web サーバの構築

初期設定 08s

1. cfg ファイルの設定

以下は、cfg ファイルの初期状態 (Step 07 までの状態) からの設定例です。

TCP モジュールの有効化

まず、TCP モジュールを有効化します。なお、本コースではプロトコルとして IPv4 を使うものとします。

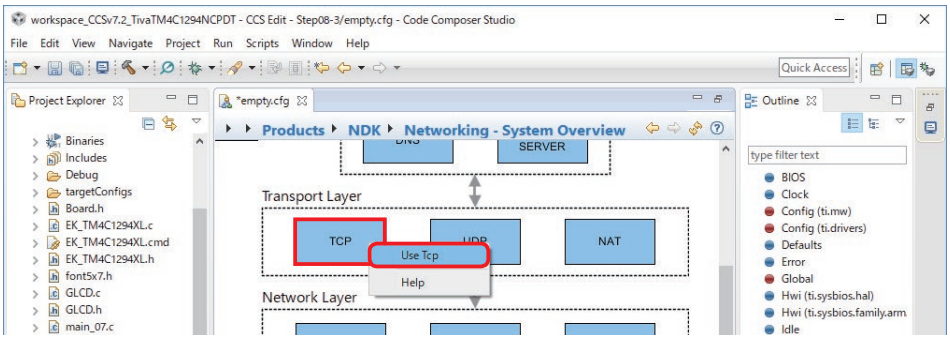
❶「 empty.cfg 」を右クリックし、「 Open With 」→「 XGCONF 」を選択します。

❷「 System Overview 」を開き、「 Communication 」内の「 TCP/IP (Wired) 」を右クリックして「 Use Global 」を選択します。

❸「 Networking welcome 」が開いたら、ページ内の「 System Overview 」を開きます。

❹「 Network Layer 」内の「 Enable IPv6 」のチェックを外します。

❺「 Transport Layer 」内の「 TCP 」を右クリックし、「 Use Tcp 」を選択します。



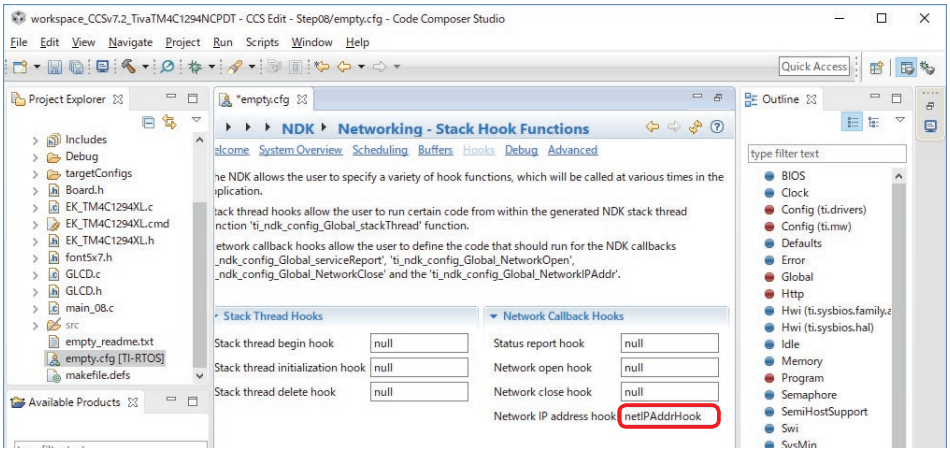
フック関数の追加

ここでは、IP アドレスが取得された時にフック関数が呼び出されるようにします。

❻ ページ右上の「 Back 」アイコン をクリックし、「 Networking – System Overview 」に戻ります。

❼ ページ上の「 Hooks 」を選択し、「 Nertworking – Stack Hook Functions 」を開きます。

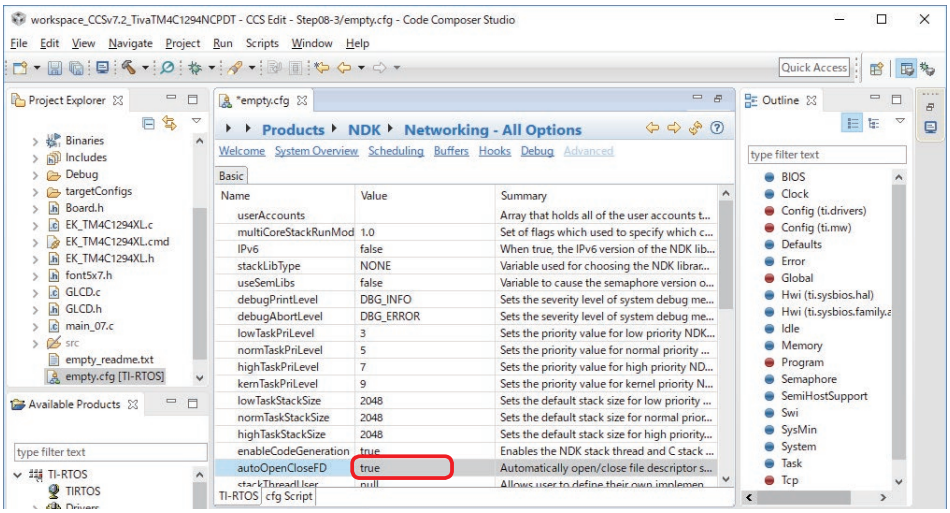
❽「 Network IP address hook 」を「 netIPAddrHook 」と設定します。



autoOpenCloseFD の有効化

ソケットを生成するには、NDK のオプション「 autoOpenCloseFD 」を有効化する必要があります。

❾ ページ上の「 Advanced 」を選択し、「 Networking – All Options 」を開き、「 autoOpenCloseFD 」の Value を「 true 」に設定します。



Web サーバの構築

初期設定 08s

OS のヒープ領域の拡張

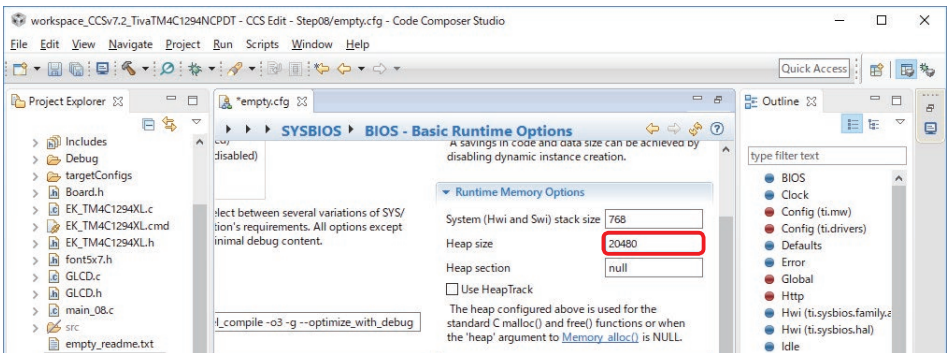
NDK を使うために、OS のヒープ領域を拡張します。

⑩ ページ右上の「 Back 」アイコン をクリックし、「 TI-RTOS – System Overview 」に戻ります。

⑪「 Kernel 」内の「 TI-RTOS Kernel (SYS/BIOS) 」をクリックします。

⑫「 BIOS – Welcome 」ページが開いたら、ページ上の「 Runtime 」を選択します。

⑬「 Runtime Memory Options 」の「 Heap size 」を「 20480 」に設定します。
この欄は、cfg ファイル保存後に「 null 」になる場合がありますが、設定は反映されています。



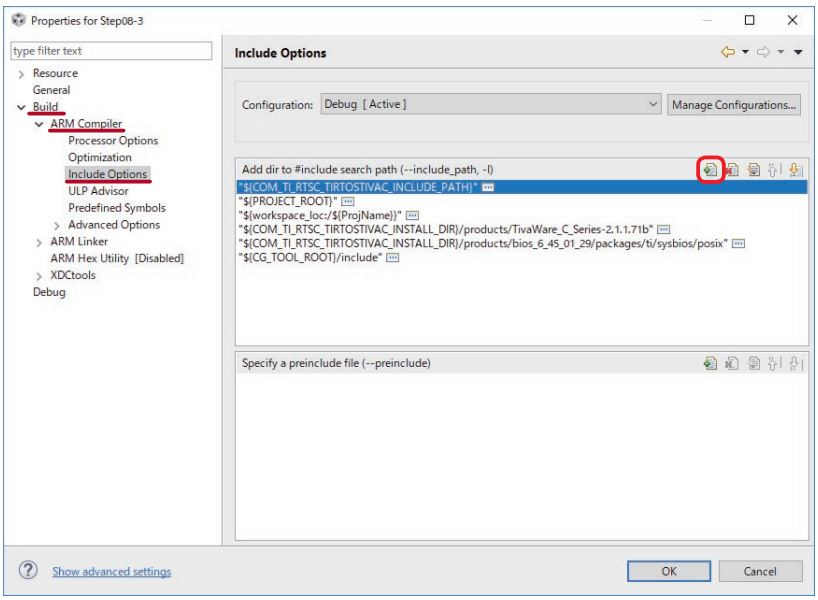
以上の設定が完了したら、「 Save 」アイコン をクリックして cfg ファイルを保存しましょう。

2. インクルードサーチパスの追加

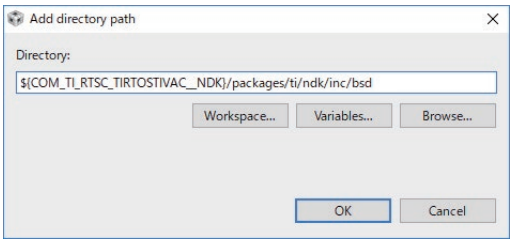
ソケットプログラミングを行うのに必要なパスを追加します。

❶ プロジェクトエクスプローラ上でプロジェクトディレクトリを右クリックし、「Properties」を選択します。

❷ 「Build」→「ARM Compiler」→「Include Options」を選択します。



❸ 「Add dir to #include search path」の「Add ...」アイコン をクリックし、「Add directory path」ウィンドウが開いたら「${COM_TI_RTSC_TIRTOSTIVAC__NDK}/packages/ti/ndk/inc/bsd」と入力して「OK」をクリックします※。



❹ ウィンドウ右下の「OK」をクリックし、Properties ウィンドウを閉じます。

※ 「${COM_TI_RTSC_TIRTOSTIVAC__NDK}」は「Variables...」から選択することができます。

Web サーバの構築

インクルードファイル 08s

ソケットプログラミングを行う場合に必要なインクルードファイルを解説します。

なお、インクルードされているファイルは、テキストカーソルをファイル名部分に合わせて「F3」キーを押すことで開くことができます。

標準ヘッダファイル　string.h

C 言語の標準ヘッダファイルである「string.h」は、主に文字列や配列を操作する関数が定義されていますが、ここでは以下の関数を使うためにインクルードしています。

- memset()
- strlen()

ソケットヘッダファイル　sys/socket.h

ソケットプログラミングに使う関数や型は、「sys/socket.h」をインクルードすることで使うことができるようになります。本コースでは以下のマクロ・構造体・関数を使います。

- htons()
- htonl()
- INADDR_ANY
- struct sockaddr
- struct sockaddr_in
- socket()
- bind()
- listen()
- accept()
- recv()
- send()
- close()

マクロ 08s

ソケットプログラミングで使うマクロを解説します。

なお、テキストカーソルをソースコード中のマクロの部分に合わせると、そのマクロに関する情報がポップアップされます。さらに「F3」キーを押すと、そのマクロが定義されているファイルを開くことができます。

バイトオーダーの反転　htons(a)

2 バイト（16 ビット）のデータ a のバイトオーダーを逆にした値を作ります（上位 8 ビットと下位 8 ビットを入れ替えた値を作ります）。

バイトオーダーの反転　htonl(a)

4 バイト（32 ビット）のデータ a のバイトオーダーを逆にした値を作ります。

未定のアドレス　INADDR_ANY

アドレスが定まっていない、あるいはアドレスを定めていないことを意味する IP アドレス「0.0.0.0」を指します。実体としては整数の「0」です。

Web サーバの構築

型・構造体 08s

ソケットプログラミングで使う構造体を解説します。

なお、テキストカーソルをソースコード中の構造体の部分に合わせると、その構造体に関する情報がポップアップされます。
さらに「F3」キーを押すと、その構造体が定義されているファイルを開くことができます。

struct sockaddr

ockaddr 構造体は、ソケットのアドレス等を指定するためのもので、以下のように定義されています。

```
struct sockaddr {
  sa_family_t sa_family;  /* address family */
  char sa_data[14];       /* socket data */
};
```

このうち、最初のメンバ変数「sa_family」には、ネットワークアドレスの種類を指定します（IPv4 ネットワークの場合は AF_INET を指定します）。また 14 ビットの配列「sa_data」に具体的なアドレス情報が格納されます。

実際のアドレスの指定には、以下の sockaddr_in 構造体を使うほうが便利ですが、関数の引数に取る際は sockaddr 型に型変換（キャスト）する必要があります※。

※　なお、型変換を行わない場合もプログラムの動作自体に影響はありませんが、コンパイル時に警告メッセージが出ます。

struct sockaddr_in

sockaddr_in 構造体は、IPv4 ネットワーク向けのアドレス指定を簡単に行うためのもので、以下のように定義されています。

```
struct sockaddr_in {
  sa_family_t sin_family;      /* address family */
  unsigned short sin_port;     /* port */
  struct in_addr sin_addr;
  char sin_zero[8];            /* fixed length address value */
};
```

また、定義中にある in_addr 構造体は以下のように定義されており、構造体のサイズは 4 バイトです。

```
struct in_addr {
  unsigned int s_addr;         /* 32 bit long IP address, net order */
};
```

sockaddr_in 構造体のメンバ変数は、以下のようになります。

- sin_family　：ネットワークアドレスの種類を指定。sockaddr_in は IPv4 向けの構造体なので、「AF_INET」を指定することになる。
- sin_port　：ポート番号を指定。HTTP の場合、ポート番号は一般に「80」。
- sin_addr.s_addr　：IP アドレスを指定。
- sin_zero　：sockaddr 構造体とサイズを合わせるための配列。特に使われないが、念のため「0」で埋めておく。

なお、ポート番号と IP アドレスは、いずれもネットワーク向けのバイトオーダーで指定する（バイトオーダーを逆にする）必要があります。

sockaddr_in 構造体の sin_port にポート番号を、sin_addr.s_addr にアドレスを指定することは、sockaddr 構造体のメンバ配列 sa_data の最初の 2 バイトにポート番号を、その次の 4 バイトにアドレスを格納するのと同じデータ操作になります。

Web サーバの構築

ライブラリ関数 08s

ソケットプログラミングで使うライブラリ関数を解説します。

なお、テキストカーソルをソースコード中の関数の部分に合わせると、その関数に関する情報がポップアップされます。さらに「F3」キーを押すと、その関数が定義されているファイルを開くことができます。

C 言語標準

メモリブロックのセット　void *memset(void *mem, int ch, size_t length)

string.h で定義されている C 言語標準の関数で、アドレス mem から length バイトを値 ch で埋めます。以下の例では、構造体型変数 servaddr を 0 で埋めています。

```
memset(&servaddr, 0, sizeof(servaddr));
```

引数の設定例

- `&servaddr`　：値を埋める最初のアドレスを指定。
- `0`　：埋める値を指定。例は 0 ＝ヌル文字。
- `sizeof(servaddr)`　：埋める長さ（バイト）を指定 。

文字列の長さの取得　size_t strlen(const char *string)

string.h で定義されている C 言語標準の関数で、文字列 string の長さを返すための関数です（実体としては string の指すアドレスから最初にヌル文字が現れるまでのバイト数を返します）。以下の例では、配列 page に格納されている文字列の長さを返します。

```
strlen(page);
```

引数の設定例

- `page`　：長さを調べたい文字列（char 型配列）の最初のアドレス。

ソケット関係

以下は、ソケットプログラミングで用いる関数です。

これらの関数のドキュメントは CCS の「Help」→「HelpContents」から「TI-RTOS for TivaC *」→「Documentation Links」→「NetworkingDocumentation」→「TI-RTOS Networking API Reference Guide」を開き、「3.3.3 Sockets APIFunctions」をご覧ください。

ソケットの生成　int socket(int domain, int type, int protocol)

通信用の端点（ソケット）を生成し、ソケットを管理するディスクリプタを返します。エラー時には -1 を返します。以下は、TCP/IPv4 用のソケットの生成例です。

```
srcSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

引数の設定例

- AF_INET　：通信を行うドメインを指定する。IPv4 ネットワークの場合は AF_INET を指定。
- SOCK_STREAM　：通信方式を指定する。
  TCP 通信では SOCK_STREAM、UDP 通信では SOCK_DGRAM とする。
  HTTP は TCP 通信を用いるので SOCK_STREAM としている。
- TCP_PROTOCOL　：ソケットが使用するプロトコルを指定する。
  TCP なら TCP_PROTOCOL、UDP なら UDP_PROTOCOL とする。
  HTTP は TCP なので TCP_PROTOCOL としている。

詳細は「3.3.3 Sockets API Functions」の「socket()」をご覧ください。

ソケットのバインド　int bind(int sockfd, const struct sockaddr *my_addr,socklen_t addrlen)

ソケットにアドレス情報を設定します。エラー時には -1 を返します。以下は、ソケット srcCosket にアドレス情報 servaddr を設定する例です（なお、socklen_t 型の実体は整数 (int) 型です）。

```
bind(srcSocket, (struct sockaddr*)&servaddr, sizeof(servaddr));
```

引数の設定例

- srcSocket　：設定するソケットのディスクリプタを指定する。
- (struct sockaddr*)&servaddr　：アドレス情報を設定した sockaddr 構造体のポインタを指定する。
- sizeof(servaddr)　：第二引数で指定した構造体の構造体長を指定する。
  ここでは sizeof() 関数を使用して、構造体長を取得している。

詳細は「3.3.3 Sockets API Functions」の「bind()」をご覧ください。

Web サーバの構築

ライブラリ関数 08s

ソケットを接続待機状態にする　int listen(int sockfd, int backlog)

指定したソケットを接続待機状態にします。これによって、初めてソケットは TCP サーバになります。失敗した場合は -1 を返します。以下は、使用例です。

```
listen(srcSocket, 4);
```

引数の設定例

- srcSocket　：接続するソケットのディスクリプタを指定する。
  ソケットは bind 関数でアドレス情報を設定したものを使用する。
- 4：クライアントを待たせる数を設定する。場合によるが通常は 4 ～ 10 くらいの値をとる。

詳細は「3.3.3 Sockets API Functions」の「listen()」をご覧ください。

クライアントとの接続を確立　int accept(int sockfd, struct sockaddr *addr,socklen_t *addrlen)

ソケットへの接続を受け付け、受け付けたソケットのディスクリプタを返します。TCP サーバにおいては、クライアントとの接続を確立します。エラーの場合は -1 を返します。以下は、使用例です。

```
Connection = accept(srcSocket, (struct sockaddr*)&cliaddr, &addrlen);
```

引数の設定例

- srcSocket　：接続を確立するソケットのディスクリプタを指定する。
  listen 関数で指定したものと同じものを指定する。
- (struct sockaddr*)&cliaddr：接続要求のあったクライアントのアドレス情報を格納するための sockaddr 構造体のポインタを設定する。
- addrlen　：構造体長のポインタを指定する（関数実行時には第二引数で指定した構造体長を渡すのに使用し、関数の実行が完了すると実際のアドレスの大きさが格納される）

詳細は「3.3.3 Sockets API Functions」の「accept()」をご覧ください。

パケットの受信　ssize_t recv(int sockfd, void *buf, size_t len, inf flags)

接続済みの相手からパケットを受信し、受信したバイト数を返します。エラーの場合は -1 を返します。以下は、受信したデータをバッファ Receive_data に格納する例です（なお、ssize_t 型の実体は整数 (int) 型です）。

```
receiveDataSize = recv(Connection, Receive_data, sizeof(Receive_data), 0);
```

引数の設定例

- Connection　：受信する接続済みソケットのディスクリプタ（accept 関数が返したもの）を指定する。
- Receive_data　：受信バッファの先頭ポインタを設定する。
- sizeof(Receive_data)　：受信バッファのサイズを指定する。
- 0　：受信オプションを指定する。通常は 0 とする。

詳細は「3.3.3 Sockets API Functions」の「recv()」をご覧ください。

パケットの送信　ssize_t send(int sockfd, const void *buf, size_t len, int flags)

接続済みの相手にパケットを送信し、送信したバイト数を返します。エラーの場合は -1 を返します。以下は、配列 Response_Header に格納されている文字列を送信する例です。

```
send(Connection, Response_Header, strlen(Response_Header), 0);
```

引数の設定例

- Connection　：送信する接続済みソケットのディスクリプタ（accept 関数が返したもの）を指定する。
- Response_Header　：送信したいデータの先頭ポインタを設定する。
- sizeof(Response_Header)　：送信データのサイズを指定する。
- 0　：送信オプションを指定する。通常は 0 とする。

詳細は「3.3.3 Sockets API Functions」の「send()」をご覧ください。

Web サーバの構築

ライブラリ関数 08s

ソケットを解放する　int close(int sockfd)

ディスクリプターをクローズします。ここでは、接続済みソケットを解放します。エラーの場合は -1 を返します。以下は、ディスクリプタ Connection が示すソケットを解放する例です。

```
close(Connection);
```

引数の設定例

- `Connection` ：解放したい接続済みソケットのディスクリプタ（accept 関数が返したもの）を指定する。

詳細は「3.3.3 Sockets API Functions」の「close()」をご覧ください。

コーディング socket.c 08s

NDK のヘッダファイルはソケットのヘッダファイルと干渉するため、ソケットプログラミング用に新しいソースファイルを作り、その中に HTTP タスクを記述することにします。

1. プロジェクトエクスプローラー上でプロジェクトフォルダを右クリックし、「New」→「Source File」を選択します。

2. 「New Source File」ウィンドウが開いたら、「Source file」に適当なファイル名（例えば「socket.c」）を入力し、右下の「Finish」をクリックします。

以下は、「socket.c」のコーディング例です。

socket.c　main.c

```
/* XDCtools ヘッダファイル */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS ヘッダファイル */
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>

/* C 言語ヘッダファイル */
#include <string.h>
#include <sys/socket.h>

/* ユーザー定義マクロ */
#define CONSOLE(...) do { System_printf(__VA_ARGS__); System_flush(); } while(0)
#define SLEEP(X) Task_sleep((X)*1000/Clock_tickPeriod)

/*
 * HTTP タスク
 */
Void task_HTTP(UArg arg0, UArg arg1)
{
    int i; // 汎用カウンタ
    int Connection; // ブラウザとの接続状態
    char Receive_data[400]; // 受信バッファ
    int receiveDataSize; // 受信データサイズ
    int srcSocket; // ソケット
    struct sockaddr_in servaddr, cliaddr; // ソケットアドレス構造体
    int addrlen; // クライアントのアドレス長

    // HTTP レスポンスを定義
    char Response_Header[] = "HTTP/1.0 200 OK\n"
            "Connection: close\n"
```

Web サーバの構築

コーディング socket.c 08s

```
33              "Content-type: text/html\n"
34              "\n";
35
36      // 表示させる web ページの html ソース
37      char page[] = "<!DOCTYPE html>\n"
38              "<html>\n"
39              "<head>\n"
40              "<meta charset='Shift_JIS'>\n"
41              "<title>Web サーバのテスト </title>\n"
42              "</head>\n"
43              "<body>\n"
44              "Hello, world!\n"
45              "</body>\n"
46              "</html>\n";
47
48      CONSOLE("HTTP タスクの開始 \n");
49
50      // サーバ（マイコン）側の IP アドレスおよびポート番号の設定
51      memset(&servaddr, 0, sizeof(servaddr));
52      servaddr.sin_family = AF_INET;
53      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
54      servaddr.sin_port = htons(80);
55
56      // ソケットの生成
57      srcSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
58      if (srcSocket == -1)
59      {
60          CONSOLE(" ソケットの生成に失敗しました！ \n");
61          return;
62      }
63
64      // ソケットのバインド
65      if (bind(srcSocket, (struct sockaddr*) &servaddr, sizeof(servaddr)) == -1)
66      {
67          CONSOLE(" ソケットのバインドに失敗しました！ \n");
68          return;
69      }
70
71      // 接続の許可
72      if (listen(srcSocket, 4) == -1)
73      {
74          CONSOLE(" 接続の許可に失敗しました！ \n");
75          return;
76      }
77
78      while (1)
79      {
80          // 時間待ち
81          SLEEP(20);
82
```

```
83          // クライアントのアドレス長の初期化
84          addrlen = sizeof(cliaddr);
85
86          // 接続受付
87          Connection = accept(srcSocket, (struct sockaddr*) &cliaddr, &addrlen);
88          if (Connection == -1)
89          {
90              CONSOLE(" 接続受付に失敗しました！ \n");
91              continue;
92          }
93
94          // パケット受信バッファのクリア
95          memset(Receive_data, 0, sizeof(Receive_data));
96
97          // パケット受信
98          receiveDataSize = recv(Connection, Receive_data, sizeof(Receive_data), 0);
99          CONSOLE("%d バイト受信しました。\n", receiveDataSize);
100
101         // 受信したデータをコンソールに表示
102         CONSOLE("======== Receive_data begin ========\n");
103         for (i = 0; i < receiveDataSize; i += 128)
104         {
105             int j = i + 128;
106             if (j < receiveDataSize)
107             {
108                 char c = Receive_data[i + 128];
109                 Receive_data[i + 128] = '\0';
110                 CONSOLE("%s", Receive_data + i);
111                 Receive_data[i + 128] = c;
112             }
113             else
114             {
115                 CONSOLE("%s", Receive_data + i);
116             }
117         }
118         CONSOLE("\n======== Receive_data end ========\n");
119
120         // HTTP レスポンスを送信
121         send(Connection, Response_Header, strlen(Response_Header), 0);
122
123         // html ソースを送信
124         send(Connection, page, strlen(page), 0);
125
126         // 接続終了
127         close(Connection);
128     }
129 }
130
```

Web サーバの構築

コーディング main.c 08s

以下は、main.c 07 のソースコードを元にしたコーディング例です。■は main.c 07 から追加・変更した部分です。

socket.c | main.c

```
1  /* XDCtools ヘッダファイル */
2  #include <xdc/std.h>
3  #include <xdc/runtime/System.h>
4
5  /* BIOS ヘッダファイル */
6  #include <ti/sysbios/BIOS.h>
7  #include <ti/sysbios/knl/Task.h>
8  #include <ti/sysbios/knl/Clock.h>
9
10 /* ドライバヘッダファイル */
11 #include <ti/drivers/GPIO.h>
12
13 /* NDK ヘッダファイル */
14 #include <ti/ndk/inc/netmain.h>
15
16 /* ボードヘッダファイル */
17 #include "Board.h"
18
19 /* GLCD ヘッダファイル */
20 #include "GLCD.h"
21
22 /* ユーザー定義マクロ */
23 #define CONSOLE(...) do { System_printf(__VA_ARGS__); System_flush(); } while(0)
24 #define SLEEP(X) Task_sleep((X)*1000/Clock_tickPeriod)
25
26 /*
27  * ユーザータスクの優先度
28  */
29 #define TASK_LED_PRIO 1
30 #define TASK_GLCD_PRIO 9
31 #define TASK_HTTP_PRIO 5
32
33 /*
34  * ユーザースタックのサイズ
35  */
36 #define TASK_LED_STACK 512
37 #define TASK_GLCD_STACK 512
38 #define TASK_HTTP_STACK 2048
39
40 /*
41  * ユーザータスクの構造体
42  */
43 Task_Struct taskLEDStruct;
44 Task_Struct taskGLCDStruct;
```

```
Task_Struct taskHTTPStruct;

/*
 * ユーザータスクのスタック
 */
Char taskLEDStack[TASK_LED_STACK];
Char taskGLCDStack[TASK_GLCD_STACK];
Char taskHTTPStack[TASK_HTTP_STACK];

/*
 * ユーザー変数
 */
uint32_t IP; // IP アドレス格納用変数

/*
 * ユーザータスク
 */

/*
 * LED 点滅タスク
 */
Void task_LED(UArg arg0, UArg arg1)
{
    CONSOLE("LED 点滅タスクの開始 \n");

    // LED1 を点灯
    GPIO_write(Board_LED0, Board_LED_ON);

    while (1)
    {
        // LED を 0.5 秒おきにトグル
        SLEEP(500);
        GPIO_toggle(Board_LED0);
    }
}

/*
 * GLCD 表示タスク
 */
Void task_GLCD(UArg arg0, UArg arg1)
{
    char IP_Address[16]; // IP アドレス表示用配列

    CONSOLE("GLCD 表\示タスクの開始 \n");

    // 各文字列を GLCD に表示
    GLCD_str(0, 5, "HTTP SERVER");
    GLCD_str(1, 0, "IP Address:");

    // IP アドレスが取得されるまで待つ
```

Web サーバの構築

コーディング main.c 08s

```
95      while (IP == 0) SLEEP(100);
96
97      // IP アドレスを文字列に変換
98      NtIPN2Str(IP, IP_Address);
99
100     // IP アドレスを GLCD に表示
101     GLCD_str(2, 6, IP_Address);
102
103     while (1)
104     {
105         // 時間待ち
106         SLEEP(1000);
107     }
108 }
109
110 /*
111  * HTTP タスク
112  */
113 Void task_HTTP(UArg arg0, UArg arg1);
114
115 /*
116  * フック関数
117  */
118
119 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
120 {
121     Task_Params taskParams;
122
123     /* IP アドレスの取得 */
124     IP = IPAddr;
125
126     CONSOLE("IP アドレス : %x\n", IP);
127
128     // HTTP タスクの作成
129     Task_Params_init(&taskParams);
130     taskParams.stackSize = TASK_HTTP_STACK;
131     taskParams.stack = &taskHTTPStack;
132     taskParams.priority = TASK_HTTP_PRIO;
133     Task_construct(&taskHTTPStruct, (Task_FuncPtr) task_HTTP, &taskParams, NULL);
134 }
135
136 /*
137  * メイン
138  */
139
140 int main(void)
141 {
142     Task_Params taskParams;
143
144     // ボードの初期設定
```

```
145     CONSOLE(" ボードの初期設定 \n");
146     Board_initGeneral();
147     Board_initGPIO();
148     Board_initEMAC();
149
150     // GLCD の初期化
151     GLCD_init();
152
153     // ユーザー変数の初期値の設定
154     IP = 0;
155
156     // ユーザータスクの作成
157     CONSOLE(" ユーザータスクの作成 \n");
158     Task_Params_init(&taskParams);
159     taskParams.stackSize = TASK_LED_STACK;
160     taskParams.stack = &taskLEDStack;
161     taskParams.priority = TASK_LED_PRIO;
162     Task_construct(&taskLEDStruct, (Task_FuncPtr) task_LED, &taskParams, NULL);
163
164     Task_Params_init(&taskParams);
165     taskParams.stackSize = TASK_GLCD_STACK;
166     taskParams.stack = &taskGLCDStack;
167     taskParams.priority = TASK_GLCD_PRIO;
168     Task_construct(&taskGLCDStruct, (Task_FuncPtr) task_GLCD, &taskParams, NULL);
169
170     // OS の起動
171     CONSOLE("OS の起動 \n");
172     BIOS_start();
173
174     return (0);
175 }
176
```