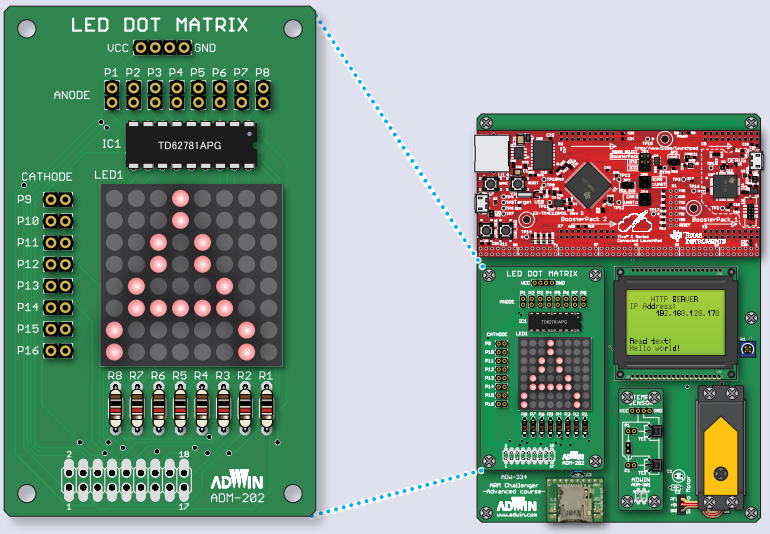
ブラウザからの LED 制御

学習内容

パソコンのブラウザからマイコンの I/O 制御を行います。
ドットマトリクス LED を点灯 / 消灯させてみましょう。

課題 10-1

LED ドットマトリクスに「A」を表示してみましょう。

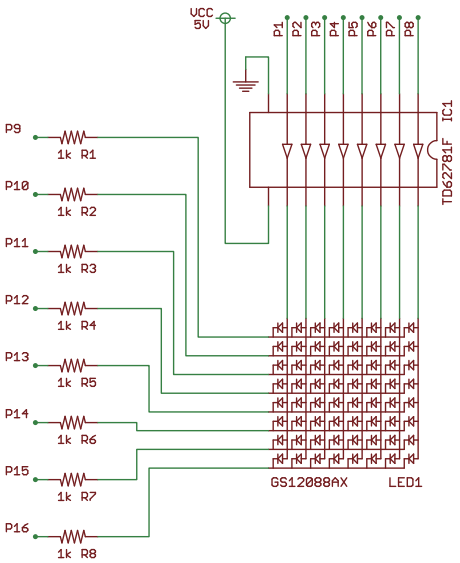


ブラウザからの LED 制御

配線 10-1

今回は LED ドットマトリクスボードを追加で使います。マイコンボードと LED ドットマトリクスボードは、ベースボード上で以下左図のように配線されています。なお、右図は LED ドットマトリクスボードの回路図です

マイコンボード		LED ドットマトリクス
+5V		+5V
GND	⟷	GND
PE0	⟷	P1 (ANODE)
PE1	⟷	P2 (ANODE)
PE2	⟷	P3 (ANODE)
PE3	⟷	P4 (ANODE)
PC4	⟷	P5 (ANODE)
PC5		P6 (ANODE)
PC6	⟷	P7 (ANODE)
PB2		P8 (ANODE)
PL0		P9 (CATHODE)
PL1		P10 (CATHODE)
PL2		P11 (CATHODE)
PL3		P12 (CATHODE)
PL4		P13 (CATHODE)
PL5		P14 (CATHODE)
PF3		P15 (CATHODE)
PG1		P16 (CATHODE)

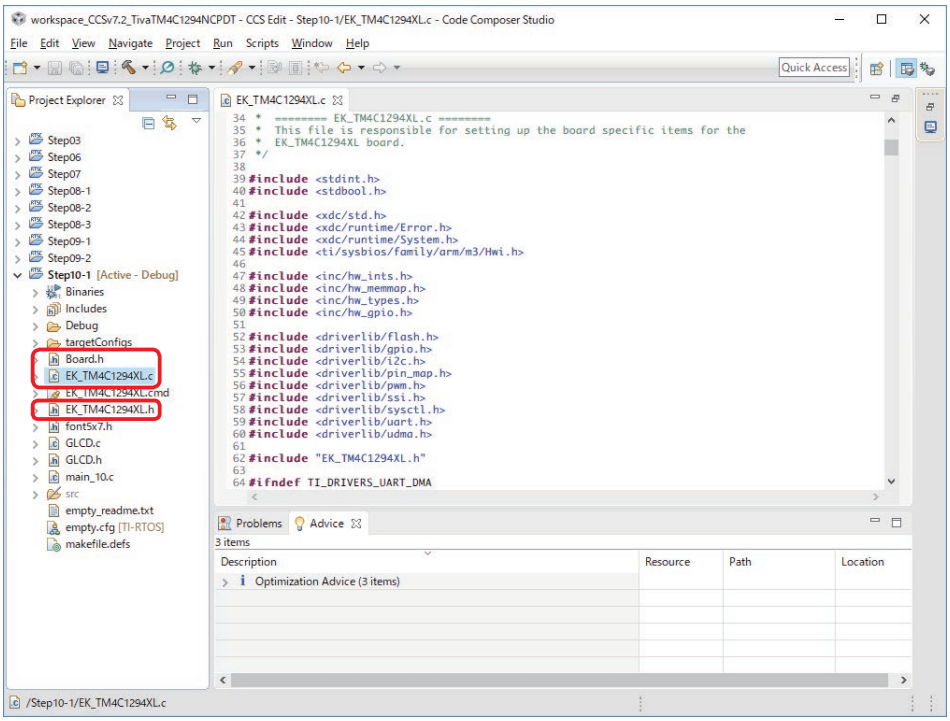


回路図

ボードの設定

サンプルプログラムの初期設定では、LED1,2 などのマイコンボード上で配線済みの GPIO しか TI-RTOS の GPIO ドライバ（GPIO_write 等の GPIO.h で定義されている関数）で扱えません。ここではまず、LED ドットマトリックスの制御に必要な GPIO ピンも GPIO ドライバで扱えるようにしておきます。

必要なピンをドライバで扱えるようにするためには、プロジェクトディレクトリ直下にある「EK_TM4C1294XL.c」を編集する必要があります。また、必要に応じて「EK_TM4C1294XL.h」や「Board.h」も編集すると良いでしょう。



ブラウザからの LED 制御

コーディング EK_TM4C1294XL.c 10-1

出力ピンの設定は、EK_TM4C1294XL.c 内の配列 gpioPinConfigs[] で行われています。既に出力ピンとして設定されている PN1（D1 制御用）、PN0（D2 制御用）を参考に、LED ドットマトリクス用のピン設定を追加しましょう。

以下は、EK_TM4C1294XL.c の設定例です。■が元々のファイルから追加した部分です。

EK_TM4C1294XL.c | EK_TM4C1294XL.h | Board.h | main.c

これ以前の行に変更はありません

```
288 GPIO_PinConfig gpioPinConfigs[] = {
289     /* Input pins */
290     /* EK_TM4C1294XL_USR_SW1 */
291     GPIOTiva_PJ_0 | GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_RISING,
292     /* EK_TM4C1294XL_USR_SW2 */
293     GPIOTiva_PJ_1 | GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_RISING,
294
295     /* Output pins */
296     /* EK_TM4C1294XL_USR_D1 */
297     GPIOTiva_PN_1 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
298     /* EK_TM4C1294XL_USR_D2 */
299     GPIOTiva_PN_0 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
300
301     /* LDM Pins (Outputs) */
302     GPIOTiva_PE_0 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
303     GPIOTiva_PE_1 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
304     GPIOTiva_PE_2 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
305     GPIOTiva_PE_3 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
306     GPIOTiva_PC_4 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
307     GPIOTiva_PC_5 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
308     GPIOTiva_PC_6 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
309     GPIOTiva_PB_2 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
310     GPIOTiva_PL_0 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
311     GPIOTiva_PL_1 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
312     GPIOTiva_PL_2 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
313     GPIOTiva_PL_3 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
314     GPIOTiva_PL_4 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
315     GPIOTiva_PL_5 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
316     GPIOTiva_PF_3 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
317     GPIOTiva_PG_0 | GPIO_CFG_OUT_STD | GPIO_CFG_OUT_STR_HIGH | GPIO_CFG_OUT_LOW,
318 };
```

これ以降の行に変更はありません

なお、出力ピンに設定されているピン番号以外の設定値は、以下のとおりです。

- GPIO_CFG_OUT_STD　　：GPIO ピンを通常の出力ピンに設定
- GPIO_CFG_OUT_STR_HIGH　：出力ピンの強さを high に設定
- GPIO_CFG_OUT_LOW　　：出力値（初期値）を 0 に設定

コーディング EK_TM4C1294XL.h 10-1

出力ピンには、EK_TM4C1294XL.c 内の列挙型 EK_TM4C1294XL_GPIOName で名前をつけることができます。先ほどの配列 gpioPinConfigs[] と同じ順番になるように、ピン名を追加しましょう。以下は、EK_TM4C1294XL.c の設定例です。■が元々のファイルから追加した部分です。

EK_TM4C1294XL.c | EK_TM4C1294XL.h | Board.h | main.c

これ以前の行に変更はありません

```
67  /*!
68   *  @def    EK_TM4C1294XL_GPIOName
69   *  @brief  Enum of LED names on the EK_TM4C1294XL dev board
70   */
71  typedef enum EK_TM4C1294XL_GPIOName {
72      EK_TM4C1294XL_USR_SW1 = 0,
73      EK_TM4C1294XL_USR_SW2,
74      EK_TM4C1294XL_D1,
75      EK_TM4C1294XL_D2,
76      LDM_P1,
77      LDM_P2,
78      LDM_P3,
79      LDM_P4,
80      LDM_P5,
81      LDM_P6,
82      LDM_P7,
83      LDM_P8,
84      LDM_P9,
85      LDM_P10,
86      LDM_P11,
87      LDM_P12,
88      LDM_P13,
89      LDM_P14,
90      LDM_P15,
91      LDM_P16,
92
93      EK_TM4C1294XL_GPIOCOUNT
94  } EK_TM4C1294XL_GPIOName;
```

これ以降の行に変更はありません

ブラウザからの LED 制御

コーディング Board.h 10-1

以上の EK_TM4C1294XL.c および EK_TM4C1294XL.h への追加で、プログラムを書いていく上で必要な設定は完了しています。しかし、Board.h にも関連する追記を行っておくと、第三者が見た場合にわかりやすく、自身への備忘録にもなります。

以下は、Board.h への追記例です。■が元々のファイルから追加した部分です。追加した #define では、ある文字列を同じ文字列に置き換えているだけ、つまり何もしていないのと同じですが、「LDM_P*」というものを定義していることが Board.h を見るだけで分かります。

EK_TM4C1294XL.c | EK_TM4C1294XL.h | **Board.h** | main.c

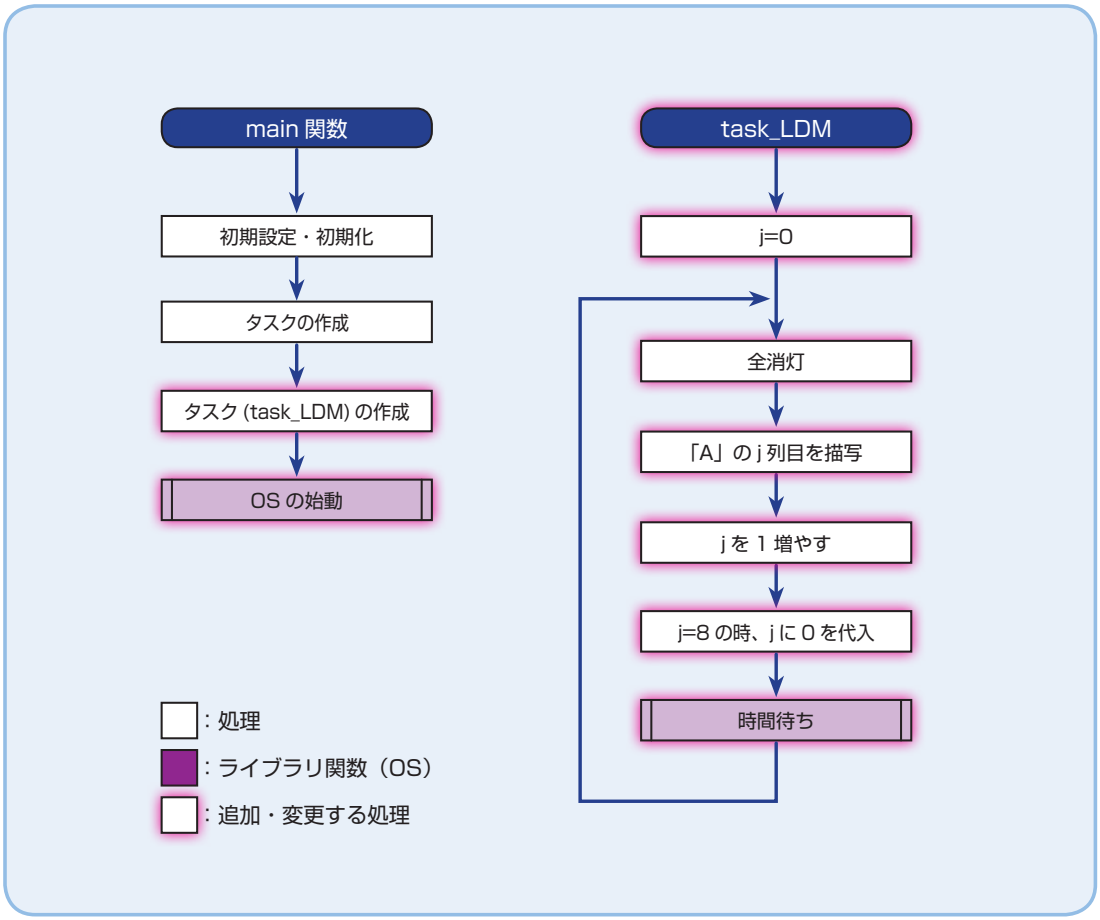
これ以前の行に変更はありません

```
55 #define Board_LED_ON         EK_TM4C1294XL_LED_ON
56 #define Board_LED_OFF        EK_TM4C1294XL_LED_OFF
57 #define Board_LED0           EK_TM4C1294XL_D1
58 #define Board_LED1           EK_TM4C1294XL_D2
59 #define Board_LED2           EK_TM4C1294XL_D2
60 #define Board_BUTTON0        EK_TM4C1294XL_USR_SW1
61 #define Board_BUTTON1        EK_TM4C1294XL_USR_SW2
62 #define LDM_P1               LDM_P1
63 #define LDM_P2               LDM_P2
64 #define LDM_P3               LDM_P3
65 #define LDM_P4               LDM_P4
66 #define LDM_P5               LDM_P5
67 #define LDM_P6               LDM_P6
68 #define LDM_P7               LDM_P7
69 #define LDM_P8               LDM_P8
70 #define LDM_P9               LDM_P9
71 #define LDM_P10              LDM_P10
72 #define LDM_P11              LDM_P11
73 #define LDM_P12              LDM_P12
74 #define LDM_P13              LDM_P13
75 #define LDM_P14              LDM_P14
76 #define LDM_P15              LDM_P15
77 #define LDM_P16              LDM_P16
78
79 #define Board_I2C0           EK_TM4C1294XL_I2C7
80 #define Board_I2C1           EK_TM4C1294XL_I2C8
81 #define Board_I2C_TMP        EK_TM4C1294XL_I2C7
82 #define Board_I2C_NFC        EK_TM4C1294XL_I2C7
83 #define Board_I2C_TPL0401    EK_TM4C1294XL_I2C7
```

これ以降の行に変更はありません

フローチャート 10-1

以下は、課題 10-1 を実現するためのフローチャート例です。その他のタスクは「フローチャート 09-3」と同じです。



ブラウザからの LED 制御

コーディング main.c 10-1

以下は、main.c 09-2 のソースコードを元にしたコーディング例です。■は main.c 09-2 から追加した部分です。

EK_TM4C1294XL.c | EK_TM4C1294XL.h | Board.h | main.c

これ以前の行に変更はありません

```
30 /*
31  * ユーザータスクの優先度
32  */
33 #define TASK_LED_PRIO 1
34 #define TASK_GLCD_PRIO 9
35 #define TASK_FATFS_PRIO 10
36 #define TASK_LDM_PRIO 11
37
38 /*
39  * ユーザースタックのサイズ
40  */
41 #define TASK_LED_STACK 512
42 #define TASK_GLCD_STACK 512
43 #define TASK_FATFS_STACK 1024
44 #define TASK_LDM_STACK 512
45
46 /*
47  * ユーザータスクの構造体
48  */
49 Task_Struct taskLEDStruct;
50 Task_Struct taskGLCDStruct;
51 Task_Struct taskFATFSStruct;
52 Task_Struct taskLDMStruct;
53
54 /*
55  * ユーザータスクのスタック
56  */
57 Char taskLEDStack[TASK_LED_STACK];
58 Char taskGLCDStack[TASK_GLCD_STACK];
59 Char taskFATFSStack[TASK_FATFS_STACK];
60 Char taskLDMStack[TASK_LDM_STACK];
```

この間の行に変更はありません

```
195 /*
196  * LED ドットマトリクスタスク
197  */
198 Void task_LDM(UArg arg0, UArg arg1)
199 {
200     int i, j; // 汎用カウンタ
201
```

```
202     // LDM 用ピン（アノード）
203     uint32_t LDM_A[] = { LDM_P1, LDM_P2, LDM_P3, LDM_P4, LDM_P5,LDM_P6, LDM_P7, LDM_P8 };
204
205     // LDM 用ピン（カソード）
206     uint32_t LDM_C[] = { LDM_P9, LDM_P10, LDM_P11, LDM_P12, LDM_P13,LDM_P14, LDM_P15, LDM_P16 };
207
208     // 列用カウンタ j を 0 に設定
209     j = 0;
210     while (1)
211     {
212         // 全消灯
213         for (i = 0; i < 8; i++)
214         {
215             GPIO_write(LDM_A[i], 0);
216             GPIO_write(LDM_C[i], 1);
217         }
218
219         //「A」の文字を描写
220         GPIO_write(LDM_A[j], 1);
221         switch (j)
222         {
223         case 0:
224             GPIO_write(LDM_C[6], 0);
225             GPIO_write(LDM_C[7], 0);
226             break;
227         case 1:
228             GPIO_write(LDM_C[4], 0);
229             GPIO_write(LDM_C[5], 0);
230             break;
231         case 2:
232             GPIO_write(LDM_C[2], 0);
233             GPIO_write(LDM_C[3], 0);
234             GPIO_write(LDM_C[5], 0);
235             break;
236         case 3:
237             GPIO_write(LDM_C[0], 0);
238             GPIO_write(LDM_C[1], 0);
239             GPIO_write(LDM_C[5], 0);
240             break;
241         case 4:
242             GPIO_write(LDM_C[2], 0);
243             GPIO_write(LDM_C[3], 0);
244             GPIO_write(LDM_C[5], 0);
245             break;
246         case 5:
247             GPIO_write(LDM_C[4], 0);
248             GPIO_write(LDM_C[5], 0);
249             break;
250         case 6:
251             GPIO_write(LDM_C[6], 0);
```

ブラウザからの LED 制御

コーディング main.c 10-1

```
252             GPIO_write(LDM_C[7], 0);
253             break;
254         case 7:
255             break;
256         }
257
258         j++;
259         if (j > 8) j = 0;
260
261         SLEEP(1);
262     }
263 }
264
265 /*
266  * フック関数
267  */
```

この間の行に変更はありません

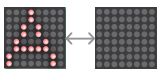
```
342 /*
343  * メイン
344  */
345
346 int main(void)
347 {
348     Task_Params taskParams;
349
350     // ボードの初期設定
351     CONSOLE(" ボードの初期設定 \n");
352     Board_initGeneral();
353     Board_initGPIO();
354     Board_initEMAC();
355     Board_initSDSPI();
356
357     // GLCD の初期化
358     GLCD_init();
359
360     // ユーザー変数の初期値の設定
361     IP = 0;
362     lockSD = 0;
363
364     // ユーザータスクの作成
365     CONSOLE(" ユーザータスクの作成 \n");
366     Task_Params_init(&taskParams);
367     taskParams.stackSize = TASK_LED_STACK;
368     taskParams.stack = &taskLEDStack;
369     taskParams.priority = TASK_LED_PRIO;
370     Task_construct(&taskLEDStruct, (Task_FuncPtr) task_LED, &taskParams, NULL);
371
372     Task_Params_init(&taskParams);
```

```
373     taskParams.stackSize = TASK_GLCD_STACK;
374     taskParams.stack = &taskGLCDStack;
375     taskParams.priority = TASK_GLCD_PRIO;
376     Task_construct(&taskGLCDStruct, (Task_FuncPtr) task_GLCD, &taskParams, NULL);
377
378     Task_Params_init(&taskParams);
379     taskParams.stackSize = TASK_FATFS_STACK;
380     taskParams.stack = &taskFATFSStack;
381     taskParams.priority = TASK_FATFS_PRIO;
382     Task_construct(&taskFATFSStruct, (Task_FuncPtr) task_FATFS, &taskParams, NULL);
383
384     Task_Params_init(&taskParams);
385     taskParams.stackSize = TASK_LDM_STACK;
386     taskParams.stack = &taskLDMStack;
387     taskParams.priority = TASK_LDM_PRIO;
388     Task_construct(&taskLDMStruct, (Task_FuncPtr) task_LDM, &taskParams, NULL);
389
390     // OS の起動
391     CONSOLE("OS の起動 \n");
392     BIOS_start();
393
394     return (0);
395 }
396
```

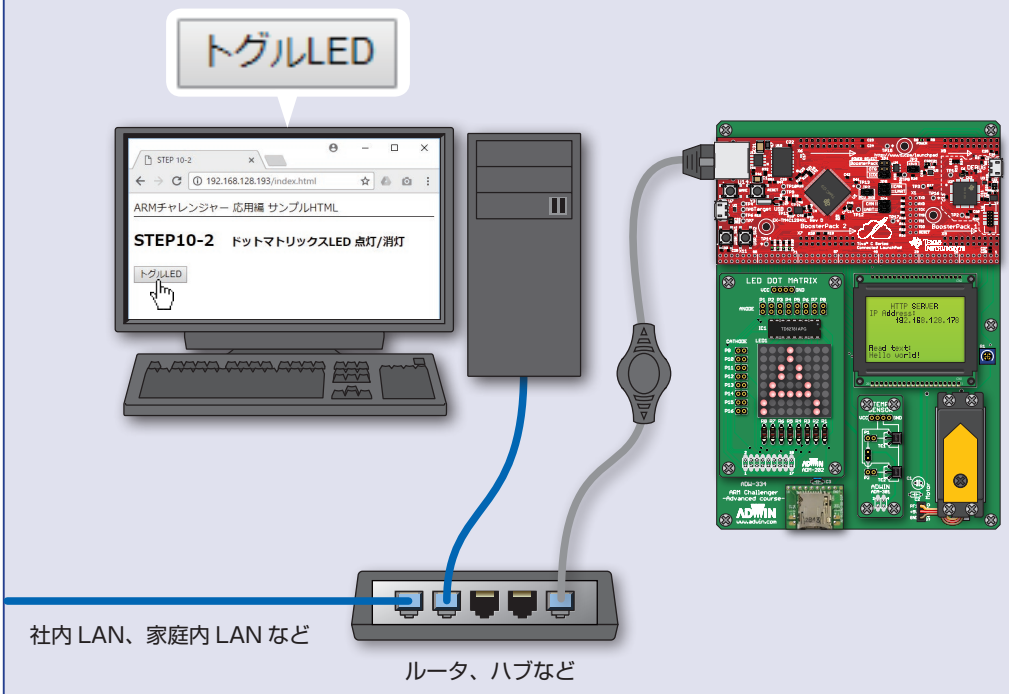
ブラウザからの LED 制御

課題 10-2

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す。
   例）　http://192.168.128.178/toggleLED.cgi
2. 上記のリクエストを送信する度に、ドットマトリクス LED の点灯／消灯が切り替わるようにします。



3. 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す。
   例）　http://192.168.128.178/
4. 上記のリクエストで表示されるページ上のボタンから、ドットマトリクス LED の点灯／消灯を切り替えられるようにします。



CGI によるブラウザからの LED 制御

ブラウザからのマイコンの制御は、ブラウザからのリクエストに応じてサーバ上でプログラムを動作させることで行うことができます。このような仕組みを CGI（Common Gateway Interface）と言います。

TI-RTOS の HTTP サーバモジュールでは、Web ファイルを作成する際に、表示するデータとして文字列ではなく関数を指定することで CGI を実現することができます。

なお、ソケットプログラミングで CGI を実現したい場合は、送られてきたリクエストの内容を解析し、CGI に関するリクエストの場合は対応する処理を行うようにすれば良いことになります。

TI-RTOS の HTTP サーバモジュールでの CGI

以下は、TI-RTOS の HTTP サーバモジュールでの CGI の実現例です。

```
// CGI ソースコード
static int cgiSample(SOCKET s, int ContentLength, char *pArgs)
{
    /* 行いたい処理 */

    // 1 を返す
    return 1;
}

Void addWebFiles()
{
    efs_createfile("sample.cgi", 0, (UINT8 *)cgiSample);
}
```

CGI のソースコードは、int 型の静的な関数で、最後に 1 を返す必要があります。引数については課題 10-3 で解説します。

Web ファイルを作成する efs_createfile 関数では、3 番目の引数に CGI ソースコードの関数を設定します。また、最初の引数（Web ファイル名）には拡張子「.cgi」のファイル名を指定する必要があり、2 番目の引数（データの大きさ）には 0 を指定します。なお、ファイル名を拡張子「.cgi」に限定したくない場合については、p.176 〜の補足をご覧ください。

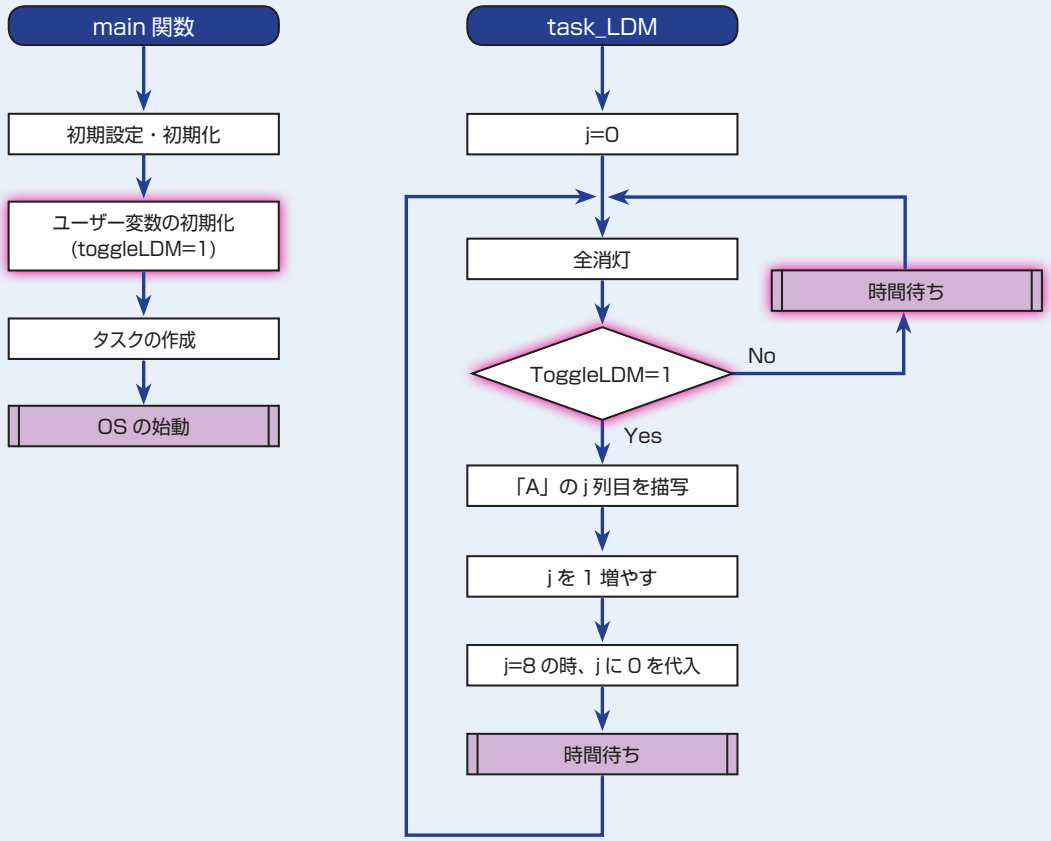
JavaScript を使用した html を作成する場合

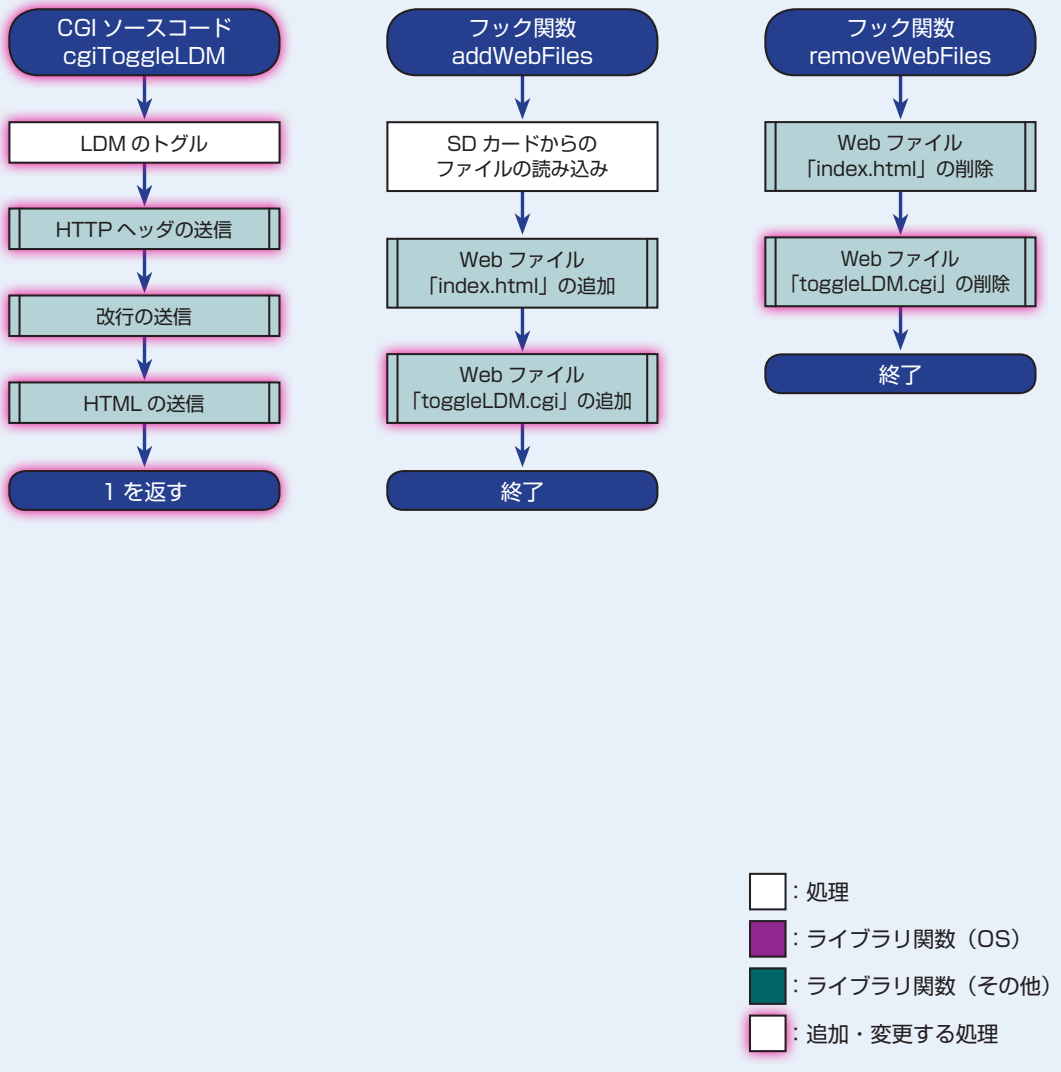
これまでのような、簡単な html ならテキストエディタでも記述できますが、JavaScript のようなプログラミング言語を使う場合は、HTML エディタを使うと開発が容易になります。
コードハイライト（色分け表示）機能があるだけでも見通しが良くなります。

ブラウザからの LED 制御

フローチャート 10-2

以下は、課題 10-2 を実現するためのマイコン側フローチャート例です。ここでは CGI ソースコード「cgiToggleLDM」が実行された時に、ドットマトリクス LED の点灯／消灯を切り替えるようにしています。その他のタスクは「フローチャート 10-1」と同じです。




CGI ソースコード
cgiToggleLDM
LDM のトグル
HTTP ヘッダの送信
改行の送信
HTML の送信
1 を返す
フック関数
addWebFiles
SD カードからの
ファイルの読み込み
Web ファイル
「index.html」の追加
Web ファイル
「toggleLDM.cgi」の追加
終了
フック関数
removeWebFiles
Web ファイル
「index.html」の削除
Web ファイル
「toggleLDM.cgi」の削除
終了
：処理
：ライブラリ関数（OS）
：ライブラリ関数（その他）
：追加・変更する処理

ブラウザからの LED 制御

型 10-2

課題 10-2 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型や構造体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。

SOCKET

ソケットを扱うための変数で、NDK のヘッダファイルで定義されています。実体は void * 型です。

定数・マクロ 10-2

課題 10-2 で使用する型を解説します。以下の定数・マクロは、いずれも NDK のヘッダファイルで定義されています。

なお、テキストカーソルをソースコード中の定数･マクロの部分に合わせると、その定数･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、その定数・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。

HTTP_OK

HTTP のステータスコードのひとつで、リクエストが成功したことを示すステータスコードの整数「200」が定義されています。

CONTENT_TYPE_HTML

コンテンツが HTML であることを示す文字列「"text/html"」が定義されています。

CRLF

改行コードの文字列「"\r\n"」が定義されています。

ライブラリ関数 10-2

課題 10-2 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 forTivaC *」→「Documentation Links」→「Networking Documenation」→「TI-RTOS NetworkingAPI Reference Guide」を開き、「E.5.3 HTTP Server Exported API Functions」をご覧ください。

NDK 関係

クライアントへの HTTP ステータスの送信
void httpSendStatusLine(SOCKET Sock, int StatusCode, char *ContentType)

ソケット Sock のクライアントに、HTTP ステータスコードが「StatusCode」、コンテンツタイプが「ContentType」であるような HTTP レスポンスのヘッダを送信します。

```
httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
```

引数の設定例

- s　：クライアントとのソケットを指定。
- HTTP_OK　：HTTP ステータスコードを指定。例は「200」。
- CONTENT_TYPE_HTML　：コンテンツタイプを指定。例は HTML を示す文字列「"text/html"」。

クライアントへの文字列の送信　void httpSendClientStr(SOCKET Sock, char*Response)

ソケット Sock のクライアントに、文字列「Response」を送信します。

```
httpSendClientStr(s, CRLF);
```

引数の設定例

- s　：クライアントとのソケットを指定。
- CRLF　：文字列を設定。例は改行コードの文字列「"\r\n"」。

なお、HTTP レスポンスのヘッダと本文の間には空行が 1 行ある必要があります。上記の使用例は、この空行を入れるのに使うことができます。

ブラウザからの LED 制御

コーディング main.c 10-2

以下は、main.c 10-1 のソースコードを元にしたコーディング例です。■は main.c 10-1 から追加した部分です。

main.c

これ以前の行に変更はありません

```
62 /*
63  * ユーザー変数
64  */
65 uint32_t IP;        // IP アドレス格納用変数
66 char lockSD;       // SD カード排他制御用変数
67 char toggleLDM;    // LDM のトグル
68
69 /*
70  * ユーザー関数
71  */
72
73 /*
74  * CGI ソースコード
75  */
76 static int cgiToggleLDM(SOCKET s, int ContentLength, char *pArgs)
77 {
78     CONSOLE("CGI を開始 \n");
79
80     // LDM のトグル
81     toggleLDM ^= 1;
82
83     httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
84     httpSendClientStr(s, CRLF);
85     httpSendClientStr(s, "<!DOCTYPE html>\n"
86                          "<html>\n"
87                          "<head>\n"
88                          "<meta charset='Shift_JIS'>\n"
89                          "<title>Web サーバのテスト </title>\n"
90                          "</head>\n"
91                          "<body>\n"
92                          "CGI のテスト \n"
93                          "</body>\n"
94                          "</html>");
95
96     // 1 を返す
97     return 1;
98 }
99
100 /*
101  * ユーザータスク
102  */
```

この間の行に変更はありません

```
227 /*
228  * LED ドットマトリクスタスク
229  */
230 Void task_LDM(UArg arg0, UArg arg1)
231 {
 .          .
 .          .
 .          .
244         // 全消灯
245         for (i = 0; i < 8; i++)
246         {
247             GPIO_write(LDM_A[i], 0);
248             GPIO_write(LDM_C[i], 1);
249         }
250
251         // LED ドットマトリクス描写の休止
252         if (toggleLDM != 1)
253         {
254             SLEEP(30);
255             continue;
256         }
```

この間の行に変更はありません

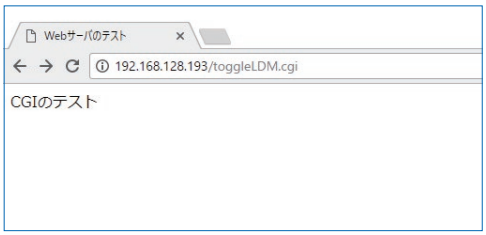
```
304 /*
305  * フック関数
306  */
307
308 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
309 {
310     /* IP アドレスの取得 */
311     IP = IPAddr;
312     CONSOLE("IP アドレス : %x\n", IP);
313 }
314
315 Void addWebFiles()
316 {
 .       .
 .       .
 .       .
371     /* Web ファイルの追加 */
372     efs_createfile("index.html", strlen(page), (UINT8 *) page);
373     efs_createfile("toggleLDM.cgi", 0, (UINT8 *)cgiToggleLDM);
374 }
375
376 Void removeWebFiles()
377 {
378     /* Web ファイルの削除 */
379     efs_destroyfile("index.html");
380     efs_destroyfile("toggleLDM.cgi");
381 }
382
```

ブラウザからの LED 制御

コーディング main.c 10-2

```
383 /*
384  * メイン
385  */
386
387 int main(void)
388 {
389     Task_Params taskParams;
390
391     // ボードの初期設定
392     CONSOLE(" ボードの初期設定 \n");
393     Board_initGeneral();
394     Board_initGPIO();
395     Board_initEMAC();
396     Board_initSDSPI();
397
398     // GLCD の初期化
399     GLCD_init();
400
401     // ユーザー変数の初期値の設定
402     IP = 0;
403     lockSD = 0;
404     toggleLDM = 1;
```

これ以降の行に変更はありません



ブラウザで http://192.168.***.***/toggleLDM.cgi を開き、画面を再読み込みするごとに LED ドットマトリクスが ON/OFF することを確認しましょう。

ブラウザ上のボタンからの制御

ここで、画面の再読込を行うのではなく、ブラウザ上のボタンで LED を操作することを目指しましょう。

ブラウザ上のボタンによる制御は、JavaScript を用いて「toggleLDM.cgi」にアクセスすることで実現することができます。「toggleLDM.cgi」にアクセスする JavaScript は、以下のようになります。

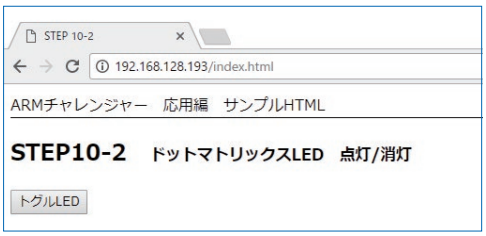
```
var request = new XMLHttpRequest();
request.open("GET", "toggleLDM.cgi", false);
request.send();
```

コーディング index.htm 10-2

以下は、index.htm のコーディング例です。ここでは、HTTP リクエストを POST メソッドで送信し、リクエスト本文が「toggle_LDM」の場合はドットマトリックス LED の点灯／消灯を、「changeChar_LDM」の場合は LED の点灯パターンの切り替えを行うようにします。

index.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 10-2</title>
<style>
h1 {
        font-size: medium;
        font-weight: normal;
        border-bottom: 1px solid;
}

span.exp {
        margin-left: 1em;
        font-size: 70%;
}
</style>
<script>
        function toggleLDM_onclick() {
                var request = new XMLHttpRequest();
                request.open("GET", "toggleLDM.cgi", false);
                request.send();
        }
</script>
</head>
<body>
        <h1>ARM チャレンジャー 応用編 サンプル HTML</h1>

        <h2> STEP10-2 <span class="exp"> ドットマトリックス LED 点灯 / 消灯 </span> </h2>
        <input type="button" value=" トグル LED" onclick="toggleLDM_onclick()">
</body>

</html>
```



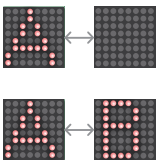
ブラウザで http://192.168.＊.＊/ を開き、画面上のボタンで LED ドットマトリクスが ON/OFF することを確認しましょう。

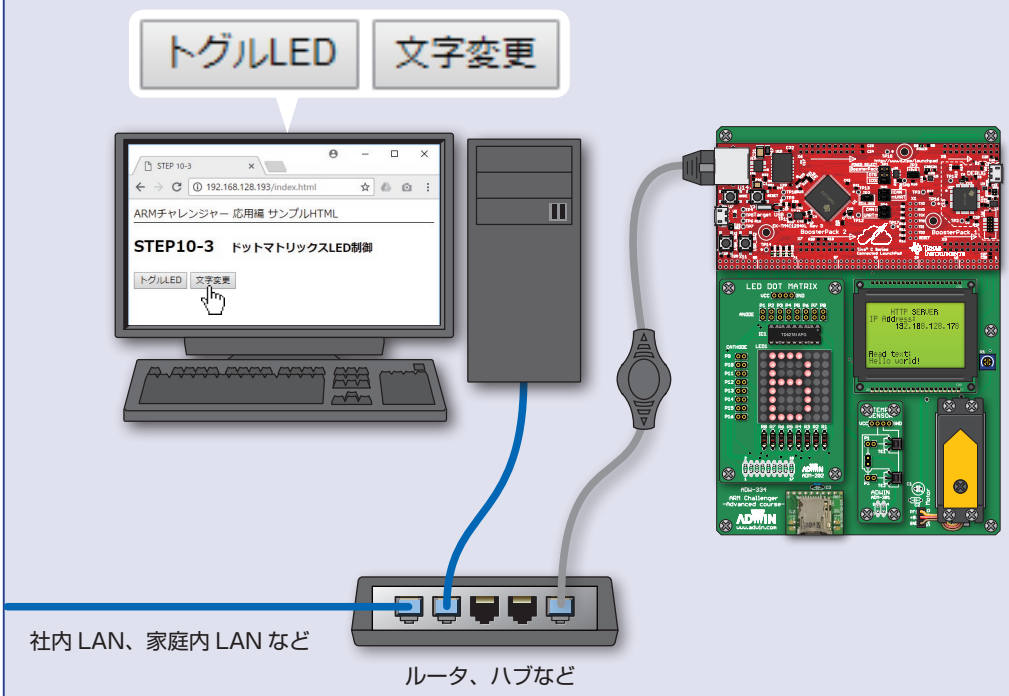
ブラウザからの LED 制御

課題 10-3

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す。
   例）　http://192.168.128.178/

2. 上記のリクエストで開かれたページ内の「トグル LED」ボタンから
   ドットマトリクス LED の点灯／消灯が、「文字変更」ボタンから
   LED の点灯パターンが切り替えられるようにします。





POST メソッドでの LED 制御の方法

ここでは、「LED のトグル」と「LED の点灯パターン変更」を、同じ URL で行うことにします。
この方法による LED 制御は、HTTP リクエストとして POST メソッドを用い、メソッドの本文にあらかじめ決めておいたコマンド名を書くこと実現することができます。

以下は、HTML ファイルに記述する JavaScript の例です。ここでは、コマンドを送るパスを「control.cgi」、POST メソッドの本文（ここではコマンド名）を「toggle_LDM」としています。
POST メソッドの本文は、send() メソッドの引数で指定します。

```
var request = new XMLHttpRequest();
request.open("POST", "control.cgi", false);
request.send("toggle_LDM");
```

マイコン側では、上記の POST メソッドの本文を配列「buffer」に取得したとすると、以下のような処理を行えば良いことになります。ここでは、コマンド名（POST メソッドの本文）が「toggle_LDM」の場合は LED のトグルを行い、「changeChar_LDM」の場合は点灯パターンの変更を行うようにしています。

```
if(strcmp(buffer, "toggle_LDM")==0) {
        // 点灯・消灯処理
}

if(strcmp(buffer, "changeChar_LDM")==0) {
        // 点灯パターン変更処理
}
```

ブラウザからの LED 制御

コーディング index.htm 10-3

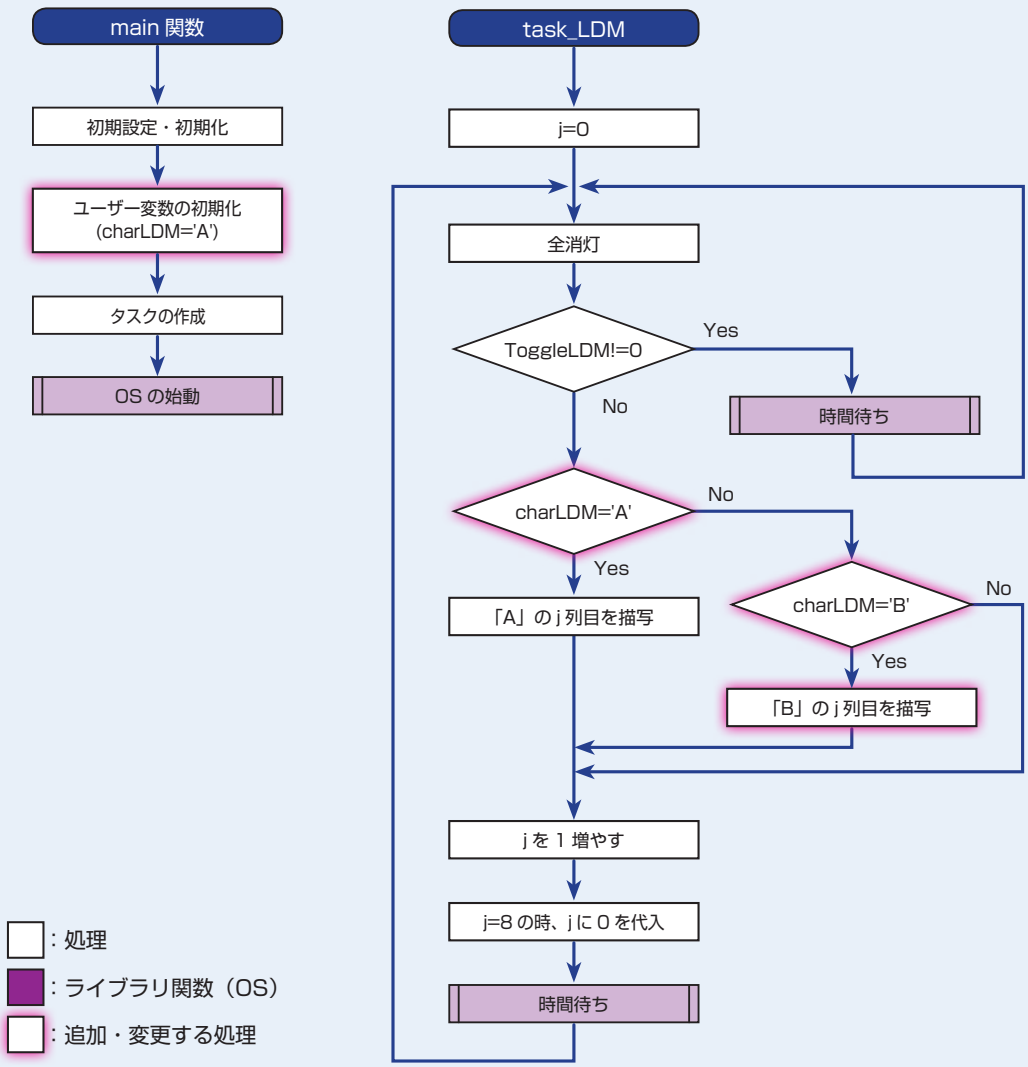
以下は、index.htm 10-2 のソースコードを元にしたコーディング例です。■ は index.htm 10-2 から追加・変更した部分です（■は課題名等の変更です）。ここでは、コマンド名（POST メソッドの本文）は control_onclick 関数の引数で指定するようにしています。また、コマンドを送るパスは「control.cgi」としています。

index.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 10-3</title>
<style>
h1 {
          font-size: medium;
          font-weight: normal;
          border-bottom: 1px solid;
}

span.exp {
          margin-left: 1em;
          font-size: 70%;
}
</style>
<script>
          function control_onclick(cmd) {
                    var request = new XMLHttpRequest();
                    request.open("POST", "control.cgi", false);
                    request.send(cmd);
          }
</script>
</head>
<body>
          <h1>ARM チャレンジャー 応用編 サンプル HTML</h1>

          <h2> STEP10-3 <span class="exp"> ドットマトリックス LED 制御 </span> </h2>
          <input type="button" value=" トグル LED" onclick='control_onclick("toggle_LDM")'>
          <input type="button" value=" 文字変更 " onclick='control_onclick("changeChar_LDM")'>
</body>

</html>
```

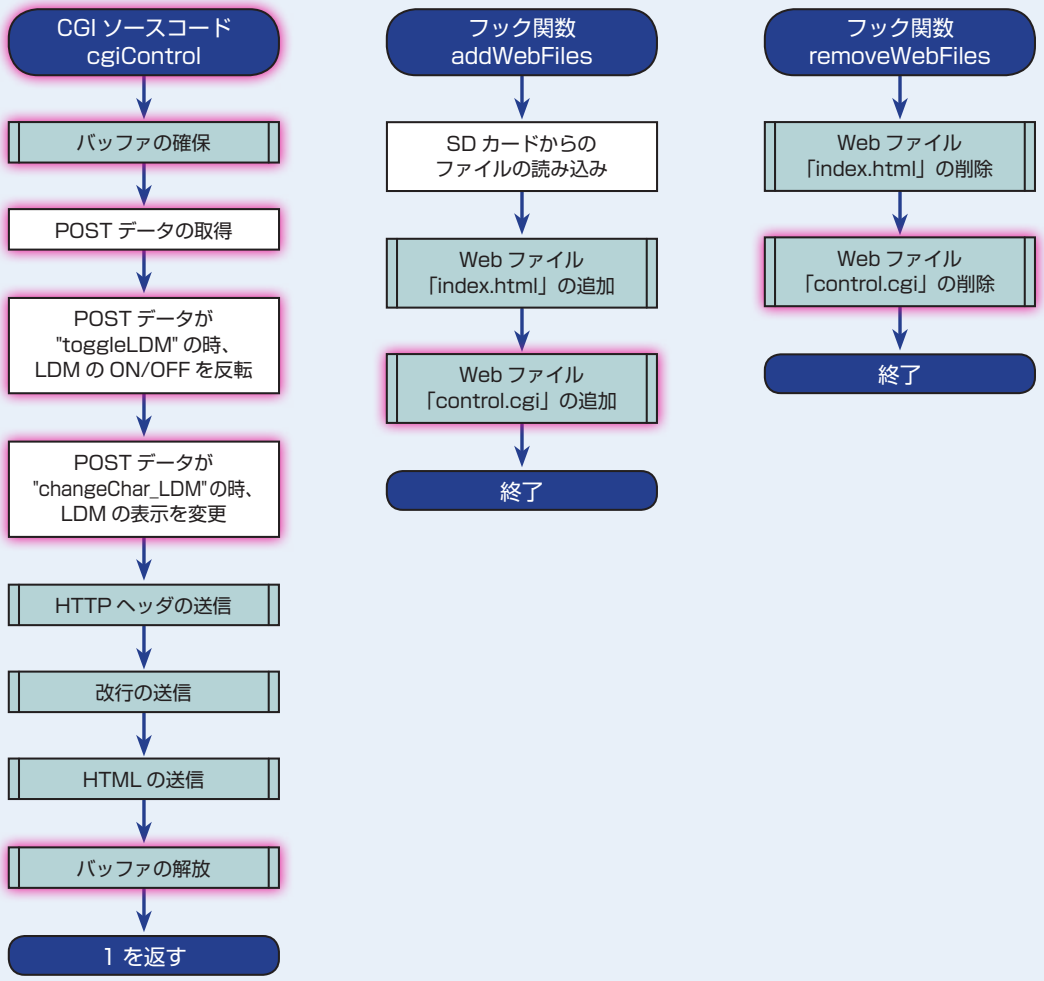
フローチャート 10-3

以下は、課題 10-3 を実現するためのフローチャート例です。ここでは、LED の点灯パターンの切り替え判定を変数「charLDM」で行っています。CGI のファイル名は「control.cgi」とし、CGI の関数名も「フローチャート 10-2」の「cgiToggleLDM」から「cgiControl」に変更しています。その他の関数・タスクは「フローチャート 10-2」と同じです。



ブラウザからの LED 制御

フローチャート 10-3



ライブラリ関数 10-3

課題 10-3 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

文字列の比較　int strcmp(const char *s1, const char *s2)

string.h※で定義されている C 言語標準の関数で、文字列 s1 と文字列 s2 を比較し、同じ文字列の場合は 0 を、そうでない場合は正または負の値を返します。以下は、buffer の示す文字列が「"toggle_LDM"」の場合に処理を実行する例です。

```
if(strcmp(buffer, "toggle_LDM")==0) { /* 処理 */ }
```

※　ヘッダファイル string.h は ti/ndk/inc/netmain.h をインクルードすることで自動的にインクルードされます。

NDK 関係

以下は、本課題で用いる NDK 関係の関数です。

これらの関数のドキュメントは CCS の「Help」→「Help Contents」から「TI-RTOS for TivaC *」→「DocumentationLinks」→「Networking Documentation」→「TI-RTOS Networking API Reference Guide」にあります。

メモリ領域の確保　void *mmBulkAlloc(INT32 Size)

引数 Size で指定した大きさのメモリ領域を確保し、その先頭のポインタを返します。確保に失敗した場合は NULL を返します。

```
buffer = (char*) mmBulkAlloc(ContentLength+1);
```

引数の設定例

- ContentLength+1：ContentLength+1 の大きさのメモリを確保。長さが「ContentLength」の文字列を格納する領域を確保したい場合、最後にヌル文字をつける領域も必要なので「+1」を加えている。

詳細は「TI-RTOS Networking API Reference Guide」の「2.4.2 Memory Allocation API Functions」をご覧ください。

ブラウザからの LED 制御

ライブラリ関数 10-3

パケットの受信　int recv(SOCKET s, void *pbuf, int size, int flags)

接続済みの相手からパケットを最大 size バイト受信して pbuf に格納し、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。

```
len = recv(s, buffer, ContentLength, MSG_WAITALL);
```

引数の設定例

- s　：クライアントとのソケット。
- buffer　：受信したパケットを格納するバッファ。
- ContentLength　：受信するサイズ。
- MSG_WAITALL　：要求した量のデータを受信するまで待つ
  （但し、状況によって要求した量より少なくても返ることがある）。

この関数は、BSD ソケットの recv() 関数（p.111 参照）とほぼ同じ動作をします。詳細は「TIRTOS Networking API Reference Guide」の「F.5 Memory Allocation API Functions」をご覧ください。また、「3.3.3 Sockets API Functions」もご覧ください。

確保したメモリの解放　void mmBulkFree(void *pv)

ポインタ pv で始まるメモリ領域を解放します。なお、メモリ領域は関数 mmBulkAlloc() で確保したものとします。

```
mmBulkFree(buffer);
```

引数の設定例

- buffer　：関数 mmBulkAlloc() の返したポインタを指定。

詳細は「TI-RTOS Networking API Reference Guide」の「2.4.2 Memory Allocation API Functions」をご覧ください。

コーディング main.c 10-3

以下は、main.c 10-2 のソースコードを元にしたコーディング例です。■は main.c 10-2 から追加した部分です。

main.c

これ以前の行に変更はありません

```
62  /*
63   * ユーザー変数
64   */
65  uint32_t IP;       // IP アドレス格納用変数
66  char lockSD;       // SD カード排他制御用変数
67  char toggleLDM;    // LDM のトグル
68  char charLDM;      // LDM に表示する文字
69
70  /*
71   * ユーザー関数
72   */
73
74  /*
75   * CGI ソースコード
76   */
77  static int cgiControl(SOCKET s, int ContentLength, char *pArgs)
78  {
79      char *buffer;
80      char len;
81
82      CONSOLE("CGI を開始 \n");
83
84      // バッファを確保
85      buffer = (char*) mmBulkAlloc(ContentLength + 1);
86      if (!buffer)
87      {
88          CONSOLE(" バッファの確保に失敗しました！ \n");
89          goto CGI_ERROR;
90      }
91
92      // POST データを取得
93      len = recv(s, buffer, ContentLength, MSG_WAITALL);
94      if (len < ContentLength)
95      {
96          CONSOLE("POST データの取得に失敗しました！ \n");
97          goto CGI_ERROR;
98      }
99      buffer[ContentLength] = '\0';
100
101     CONSOLE("POST データ :\n%s\n\n", buffer);
102
103     // POST データが "toggle_LDM" の時、LDM の ON/OFF を反転
104     if (strcmp(buffer, "toggle_LDM") == 0)
105     {
106         toggleLDM ^= 1;
```

ブラウザからの LED 制御

コーディング main.c 10-3

```
107     }
108
109     // POST データが "changeChar_LDM" の時、LDM の表示を変更
110     if (strcmp(buffer, "changeChar_LDM") == 0)
111     {
112         if (charLDM == 'A')
113         {
114             charLDM = 'B';
115         }
116         else
117         {
118             charLDM = 'A';
119         }
120     }
121
122     httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
123     httpSendClientStr(s, CRLF);
124     httpSendClientStr(s, "<!DOCTYPE html>\n"
125                          "<html>\n"
126                          "<head>\n"
127                          "<meta charset='Shift_JIS'>\n"
128                          "<title>Web サーバのテスト </title>\n"
129                          "</head>\n"
130                          "<body>\n"
131                          "CGI のテスト \n"
132                          "</body>\n"
133                          "</html>");
134
135     CGI_ERROR:
136     // バッファを解放
137     if (buffer) mmBulkFree(buffer);
138
139     // 1 を返す
140     return 1;
141 }
```

この間の行に変更はありません

```
270 /*
271  * LED ドットマトリクスタスク
272  */
273 Void task_LDM(UArg arg0, UArg arg1)
274 {
275     int i, j; // 汎用カウンタ
276
277     // LDM 用ピン（アノード）
278     uint32_t LDM_A[] = { LDM_P1, LDM_P2, LDM_P3, LDM_P4, LDM_P5,LDM_P6, LDM_P7, LDM_P8 };
279
280     // LDM 用ピン（カソード）
281     uint32_t LDM_C[] = { LDM_P9, LDM_P10, LDM_P11, LDM_P12, LDM_P13,LDM_P14, LDM_P15, LDM_P16 };
282
283     // 列用カウンタ j を 0 に設定
284     j = 0;
285     while (1)
286     {
287         // 全消灯
```

```
288         for (i = 0; i < 8; i++)
289         {
290             GPIO_write(LDM_A[i], 0);
291             GPIO_write(LDM_C[i], 1);
292         }
293 
294         // LED ドットマトリクス描写の休止
295         if (toggleLDM != 1)
296         {
297             SLEEP(30);
298             continue;
299         }
300 
301         if (charLDM == 'A')
302         {
303             //「A」の文字を描写
304             GPIO_write(LDM_A[j], 1);
305             switch (j)
306             {
307             case 0:
308                 GPIO_write(LDM_C[6], 0);
309                 GPIO_write(LDM_C[7], 0);
310                 break;
311             case 1:
312                 GPIO_write(LDM_C[4], 0);
313                 GPIO_write(LDM_C[5], 0);
314                 break;
315             case 2:
316                 GPIO_write(LDM_C[2], 0);
317                 GPIO_write(LDM_C[3], 0);
318                 GPIO_write(LDM_C[5], 0);
319                 break;
320             case 3:
321                 GPIO_write(LDM_C[0], 0);
322                 GPIO_write(LDM_C[1], 0);
323                 GPIO_write(LDM_C[5], 0);
324                 break;
325             case 4:
326                 GPIO_write(LDM_C[2], 0);
327                 GPIO_write(LDM_C[3], 0);
328                 GPIO_write(LDM_C[5], 0);
329                 break;
330             case 5:
331                 GPIO_write(LDM_C[4], 0);
332                 GPIO_write(LDM_C[5], 0);
333                 break;
334             case 6:
335                 GPIO_write(LDM_C[6], 0);
336                 GPIO_write(LDM_C[7], 0);
337                 break;
338             case 7:
339                 break;
340             }
341         }
342         else if (charLDM == 'B')
343         {
```

ブラウザからの LED 制御

コーディング main.c 10-3

```
344             //「B」の文字を描写
345             GPIO_write(LDM_A[j], 1);
346             switch (j)
347             {
348             case 0:
349                 break;
350             case 1:
351                 GPIO_write(LDM_C[0], 0);
352                 GPIO_write(LDM_C[1], 0);
353                 GPIO_write(LDM_C[2], 0);
354                 GPIO_write(LDM_C[3], 0);
355                 GPIO_write(LDM_C[4], 0);
356                 GPIO_write(LDM_C[5], 0);
357                 GPIO_write(LDM_C[6], 0);
358                 GPIO_write(LDM_C[7], 0);
359                 break;
360             case 2:
361                 GPIO_write(LDM_C[0], 0);
362                 GPIO_write(LDM_C[3], 0);
363                 GPIO_write(LDM_C[7], 0);
364                 break;
365             case 3:
366                 GPIO_write(LDM_C[0], 0);
367                 GPIO_write(LDM_C[3], 0);
368                 GPIO_write(LDM_C[7], 0);
369                 break;
370             case 4:
371                 GPIO_write(LDM_C[0], 0);
372                 GPIO_write(LDM_C[3], 0);
373                 GPIO_write(LDM_C[7], 0);
374                 break;
375             case 5:
376                 GPIO_write(LDM_C[1], 0);
377                 GPIO_write(LDM_C[2], 0);
378                 GPIO_write(LDM_C[4], 0);
379                 GPIO_write(LDM_C[5], 0);
380                 GPIO_write(LDM_C[6], 0);
381                 break;
382             case 6:
383                 break;
384             case 7:
385                 break;
386             }
387         }
388
389         j++;
390         if (j > 8) j = 0;
391
392         SLEEP(1);
393     }
394 }
395
396 /*
397  * フック関数
398  */
399
```

```
400 Void netIPAddrHook(IPN IPAddr, uint IfIdx, uint fAdd)
401 {
402     /* IP アドレスの取得 */
403     IP = IPAddr;
404     CONSOLE("IP アドレス : %x\n", IP);
405 }
406
407 Void addWebFiles()
408 {
 .       .
 .       .
 .       .
463     /* Web ファイルの追加 */
464     efs_createfile("index.html", strlen(page), (UINT8 *) page);
465     efs_createfile("control.cgi", 0, (UINT8 *) cgiControl);
466 }
467
468 Void removeWebFiles()
469 {
470     /* Web ファイルの削除 */
471     efs_destroyfile("index.html");
472     efs_destroyfile("control.cgi");
473 }
474
475 /*
476  * メイン
477  */
478
479 int main(void)
480 {
481     Task_Params taskParams;
482
483     // ボードの初期設定
484     CONSOLE(" ボードの初期設定 \n");
485     Board_initGeneral();
486     Board_initGPIO();
487     Board_initEMAC();
488     Board_initSDSPI();
489
490     // GLCD の初期化
491     GLCD_init();
492
493     // ユーザー変数の初期値の設定
494     IP = 0;
495     lockSD = 0;
496     toggleLDM = 1;
497     charLDM = 'A';
```

これ以降の行に変更はありません

ブラウザからの LED 制御

補足 10-A ファイル名を制限しない CGI

TI-RTOS では、NDK モジュール内の「efs_filecheck()」という関数で Web サーバのファイル名が「.cgi」で終わるものを CGI と判断していますが、この関数の内容を変更することで CGI のファイル名を「*.cgi」に限らないようにすることができます。ここでは、すべてのファイルを CGI として扱う方法を見ていきます。

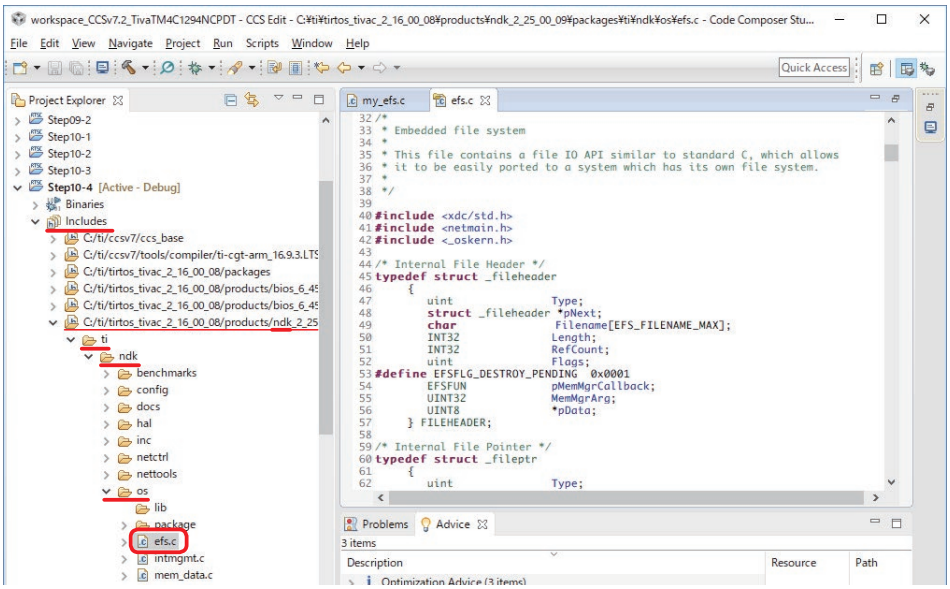
efs.c ファイルのコピー

efs_filecheck() 関数は、NDK モジュール内の efs.c ファイルに書かれています。このファイルを直接編集するのは問題が多いので※1、プロジェクト内にコピーし編集することにします。プロジェクト内に efs_filecheck() 関数が定義されていると、プログラムからは OS の efs_filecheck() 関数ではなくプロジェクト内の（編集した）efs_filecheck() 関数が呼ばれることになります。

1. Project Explorer 内のプロジェクトディレクトリを右クリックし、「New」→「Source File」を選択します。「New Source File」ウィンドウが開いたら、「Source File:」欄に適当なソースファイル名（ここでは、「my_efs.c」とします）を設定し、「Finish」をクリックします。すると、プロジェクトディレクトリ直下に my_efs.c ファイルが作成されます。

2. プロジェクトディレクトリ内の「Include」の「/*/ndk_*/packages」→「ti」→「ndk」→「os」にある efs.c ファイルを開き、ファイルの内容をコピーします※2。

3. コピーした内容を、先ほど作成した my_efs.c ファイルにペーストします※3。



※1　このファイルは OS の一部であって、すべてのプロジェクトに共通のものなので、特定のプロジェクトのために編集すべきではありません。また、編集内容を反映させるには OS の再コンパイルを別途行う必要があり、CCS でプロジェクトをビルドしただけでは変更は反映されません。

※2　ファイル内で「Ctrl+A」を押すと全選択をすることができ、さらに「Ctrl+C」を押すことでコピーができます。

※3　コピーした内容は「Ctrl+V」でペーストできます。

my_efs.c ファイルの編集

my_efs.c ファイルにペーストした内容を、以下のように編集しましょう。■は元の内容からの変更部分です。

my_efs.c　main.c

これ以前の行に変更はありません

```
40 #include <xdc/std.h>
41 #include <ti/ndk/inc/netmain.h>
42 #include <ti/ndk/inc/_oskern.h>
43
44 /* Internal File Header */
```

そのままではパスが通っていないので、ファイル名の前に「ti/ndk/inc 」を追加。

この間の行に変更はありません

```
497 int efs_filecheck(char *name, char *user, char *pass, int *prealm)
498 {
  .     .
  .     .
  .     .

512     /* Next, determine if the file is a "function". */
513
514     /* The method for detecting if a filename represents an executable */
515     /* function is determined by this EFS module. It can use any method */
516     /* suitable to application developer. */
517
518     /* For this implementation, we assume if the file ends in '.cgi', */
519     /* then its a function. */
520
521 //    i = strlen(name);
522 //    if( i>4 && !stricmp( name+i-4, ".cgi" ) )
523         retflags |= EFS_FC_EXECUTE;
```

この２行をコメントアウト。

これ以降の行に変更はありません

以上で、すべてのファイルが CGI として扱われるようになります。なお、すべてのファイルが CGI になるので、通常の HTML ファイルの表示も CGI で行う必要があります。

ブラウザからの LED 制御

コーディング main.c 10-A

以下は、課題 10-2 の CGI ファイル名「toggleLED.cgi」を「toggleLED」に変更する場合のソースコード例です※。■ は main.c 10-2 からの変更部分、■ は main.c 10-2 から削除したコードのある部分です。

※　ここでは、ソースコードは最小限の変更に留めています。より効率的なソースコードを考えてみるのも良いでしょう。

my_efs.c　main.c

これ以前の行に変更はありません

```
73 /*
74  * CGI ソースコード
75  */
76 static int cgiIndexHtml(SOCKET s, int ContentLength, char *pArgs)
77 {
78     SDSPI_Handle sdspiHandle; // SD カードマウントのハンドラ
79     SDSPI_Params sdspiParams; // SD カードマウントのパラメータ
80
81     FILE *fd; // ファイルディスクリプタ
82
83     static char page[2048]; // ページデータの読み込みバッファ
84     unsigned int bytesRead; // ファイル読み込みサイズ
85
86     // 他のタスクが SD カードを使っている場合は使い終わるまで待つ
87     while (lockSD != 0) SLEEP(100);
88
89     // SD カードの使用を開始
90     lockSD = 1;
91
92     // SD カードの設定パラメータの初期化
93     SDSPI_Params_init(&sdspiParams);
94
95     // SD カードをマウント
96     sdspiHandle = SDSPI_open(Board_SDSPI0, 0, &sdspiParams);
97
98     // エラーチェック
99     if (sdspiHandle == NULL)
100    {
101        CONSOLE(" マウントに失敗しました！ \n");
102        // GLCD にエラーメッセージを表示
103        GLCD_str(7, 0, "Mount error");
104    }
105    else
106    {
107        // ファイルを開く
108        fd = fopen("fat:0:index.htm", "r");
109
110        // エラーチェック
111        if (fd == NULL)
112        {
113            CONSOLE(" ファイルを開けませんでした！ \n");
```

addWebFile() 内で行っていた処理をこの関数内に移動。

```
114             // GLCD にエラーメッセージを表示
115             GLCD_str(7, 0, "File open error");
116         }
117         else
118         {
119             // ファイルから文字列を読み込み
120             bytesRead = fread(page, 1, sizeof(page) - 1, fd);
121             page[bytesRead] = '\0';
122             CONSOLE(" 読み込んだバイト数 : %d/%d\n", bytesRead, sizeof(page) - 1);
123 
124             // ファイルを閉じる
125             fclose(fd);
126         }
127 
128         // SD カードを停止
129         SDSPI_close(sdspiHandle);
130     }
131 
132     // SD カードの使用を終了
133     lockSD = 0;
134     httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
135     httpSendClientStr(s, CRLF);
136     httpSendClientStr(s, page);
137 
138     // 1 を返す
139     return 1;
140 }
141 
142 static int cgiToggleLDM(SOCKET s, int ContentLength, char *pArgs)
143 {
```

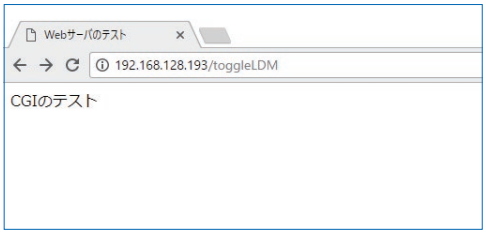
この間の行に変更はありません

```
381 Void addWebFiles()
382 {
383 
384     /* Web ファイルの追加 */
385     efs_createfile("index.html", 0, (UINT8 *)cgiIndexHtml);
386     efs_createfile("toggleLDM", 0, (UINT8 *)cgiToggleLDM);
387 }
388 
389 Void removeWebFiles()
390 {
391     /* Web ファイルの削除 */
392     efs_destroyfile("index.html");
393     efs_destroyfile("toggleLDM");
394 }
```

これ以降の行に変更はありません

ブラウザからの LED 制御

コーディング main.c 10-A



ブラウザで「http://192.168.＊.＊/toggleLDM」を開き、LED ドットマトリクスの点灯 / 消灯ができることを確認しましょう。

また、「http://192.168.＊.＊/」を開いて HTML ファイルが表示されていることも確認しましょう。

なお、課題 10-3 の HTML ファイルが表示されている場合、ブラウザ内のボタンによる LED ドットマトリクスの制御はできないのでご注意ください。