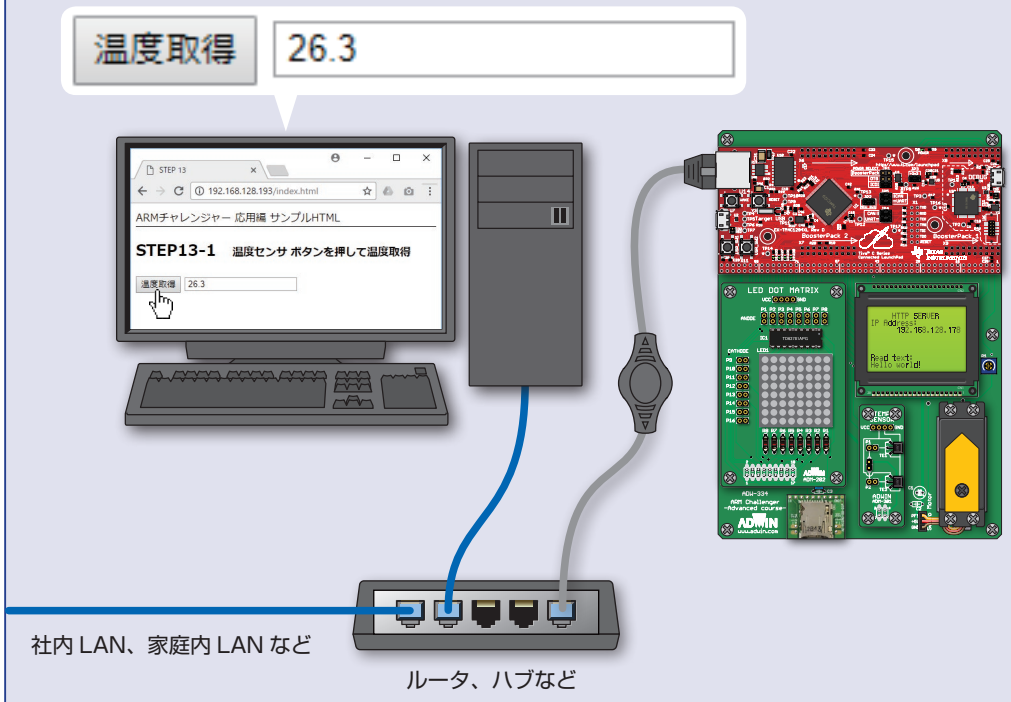
温度のブラウザへの表示

学習内容

マイコンからのパソコンのブラウザにメッセージを送信します。
ブラウザとマイコンのやり取りは STEP11 と同じです。
温度センサの入力電圧を A/D 変換し温度を求める方法を学習します。

課題 13-1

ブラウザ内の「温度取得」ボタンをクリックすると、ブラウザに温度が表示されるようにしましょう。

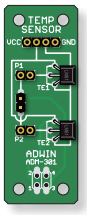


温度のブラウザへの表示

配線 13-1

マイコンボードと温度センサボードは、メインボード上で以下のように配線されています。

マイコンボード		温度センサボード
PE4	⟷	P1
PE5	⟷	P2
+3.3V	⟷	VCC
GND		GND



ブラウザに現在の気温を表示する方法

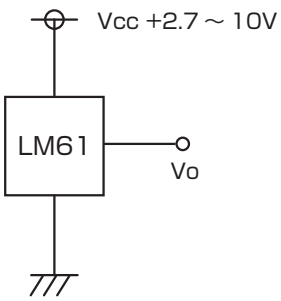
STEP13 では、ブラウザに表示されているボタンを押すことで現在の気温を取得し表示します。

基本的な考え方は STEP11 と同様で、JavaScript でサーバからデータを取得し、ブラウザの表示内容を変更します。

詳しくは STEP11 をご覧ください。

温度センサについて

今回使用している温度センサ LM61 は、周囲の温度に比例した電圧を出力するものです。
温度センサからの出力電圧を ARM マイコンの A/D 変換機能を使用して取り込み、特性式から現在の気温を換算して求めることができます。



温度 T（℃）	Vo（mV）
+100	1600
+85	1450
+25	850
0	600
−25	350

Vo = (10 × T) + 600
T = (Vo − 600) ÷ 10

ADC についての注意事項

TM4C192x シリーズの ADC

本コースで用いている TM4C1294NCPDTI を含む TM4C192x シリーズのマイコンには、
・ADC の最初の 2 回の値が不正確となる可能性がある
・ADC を行う際に PE3 ピンの電圧に異常が生じる可能性がある
というハードウェア的なエラーが存在しています※。そのため、プログラミングを行う際にはこれらのエラーへの対策も行う必要があります。また、後者に関連して、PE3 ピンの出力を H に設定している時は ADC が適切に行われない場合があるようです。このため、ADC を適切に行うには、ADC の実行中は PE3 ピンを L にしておく必要があります。

なお、本コースでは PE3 ピンは LED ドットマトリックスの P4 に使われています。ADC の開始前に PE3 ピンを L に設定しておくとともに、ADC の実行中は LDM タスクが PE3 ピンを H に設定しないようにしておきましょう。

TI-RTOS における ADC の使用方法

TI-RTOS では、ADC を扱うドライバは今のところ提供されておらず、ADC 関係の処理は TivaWare のライブラリ関数等で直接書く必要があります。

※　マイコンのハードウェア的なエラーについては、「View」→「Product Explorer」より、「Software」→「TM4C ARM Crtex-M4F MCU」→「Devices」→「TM4C」→「TM4C129X」→「Errata」をご覧ください。

温度のブラウザへの表示

コーディング index.htm 13-1

以下は、index.htm 12 を元にしたコーディング例です。■は index.htm 12 から追加した部分です（■は課題名等の変更です）。ここでは、「status.cgi」にコマンド名「get_temp」を POST メソッドの本文で送ることにより温度を取得することとします。

index.htm

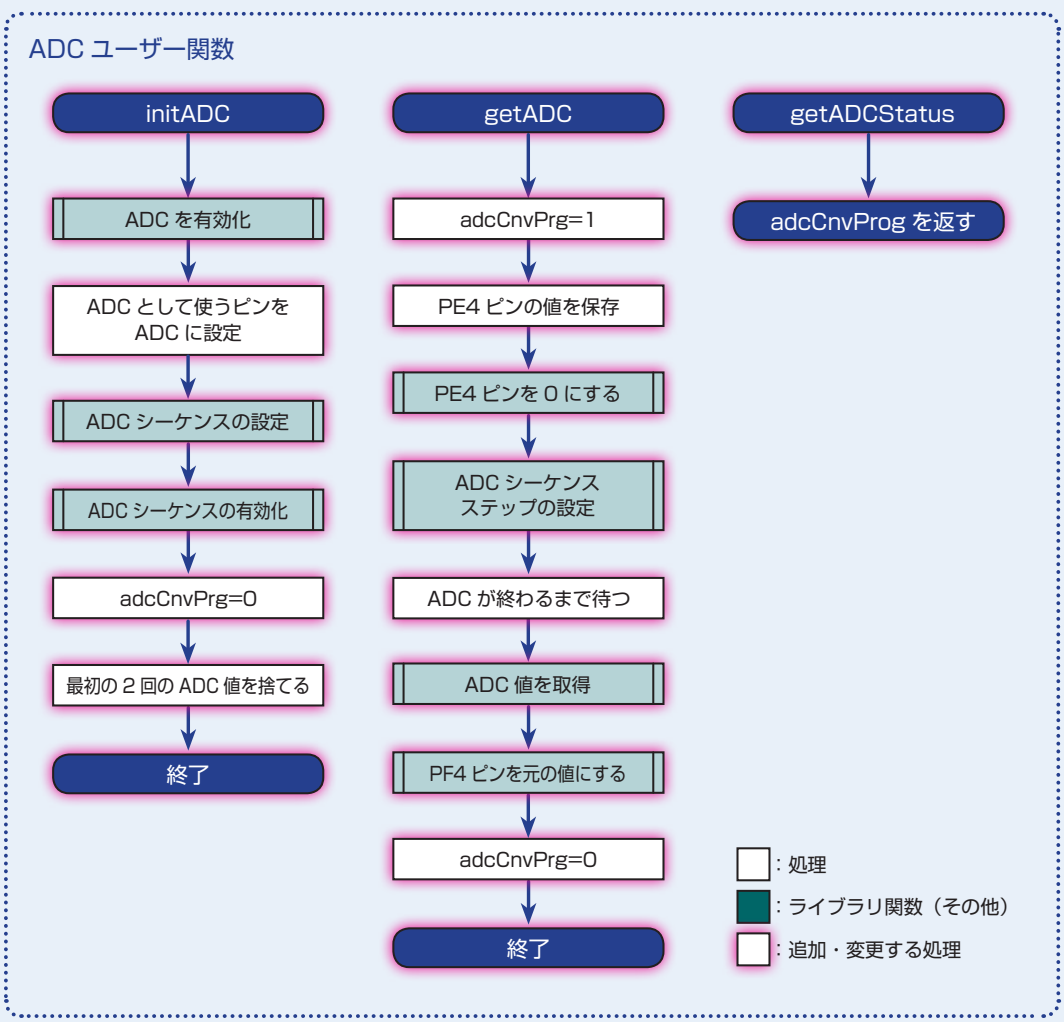
```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 13</title>
```

この間の行に変更はありません

```
<script>
        // 周期的に実行する関数
        function do_periodic() {
                Get_SW();
                // 関数の繰り返し
                setTimeout("do_periodic()", 1000);
        }
        ・
        ・
        ・
        // マイコンボードから温度の取得
        function Get_Temp() {
                // サーバからテキストを取得
                var request = new XMLHttpRequest();
                request.open("POST", "status.cgi", false);
                request.send("get_temp");
                // 取得したテキストをテキストボックスに表示
                var text_temp = document.getElementById("text_temp");
                text_temp.value = request.responseText;
        }
</script>
</head>
<body onload="do_periodic()">
        <h1>ARM チャレンジャー 応用編 サンプル HTML</h1>
        ・
        ・
        ・
        <h2> STEP13-1 <span class="exp"> 温度センサ ボタンを押して温度取得 </span> </h2>
        <input type="button" value=" 温度取得 " onclick="Get_Temp()">
        <input id="text_temp" type="text">
</body>
</html>
```

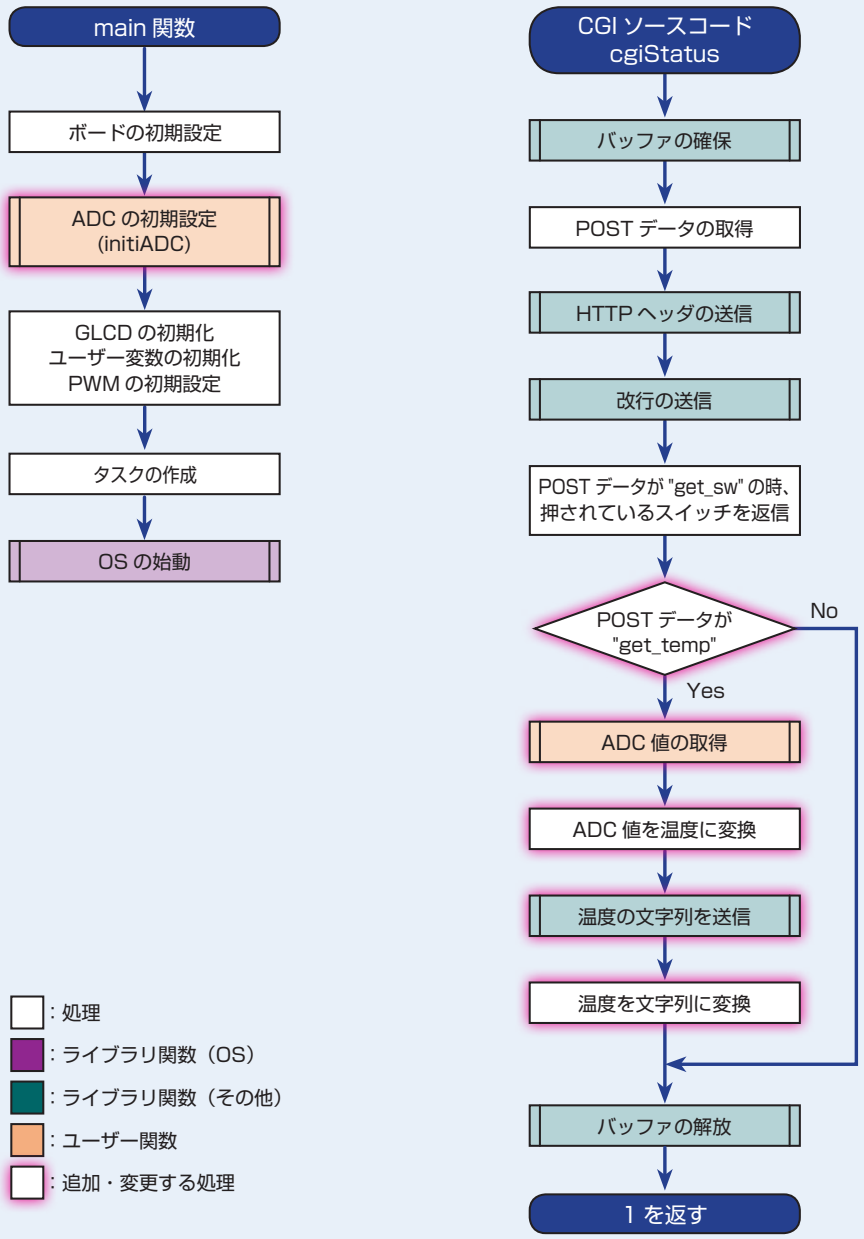
フローチャート 13-1

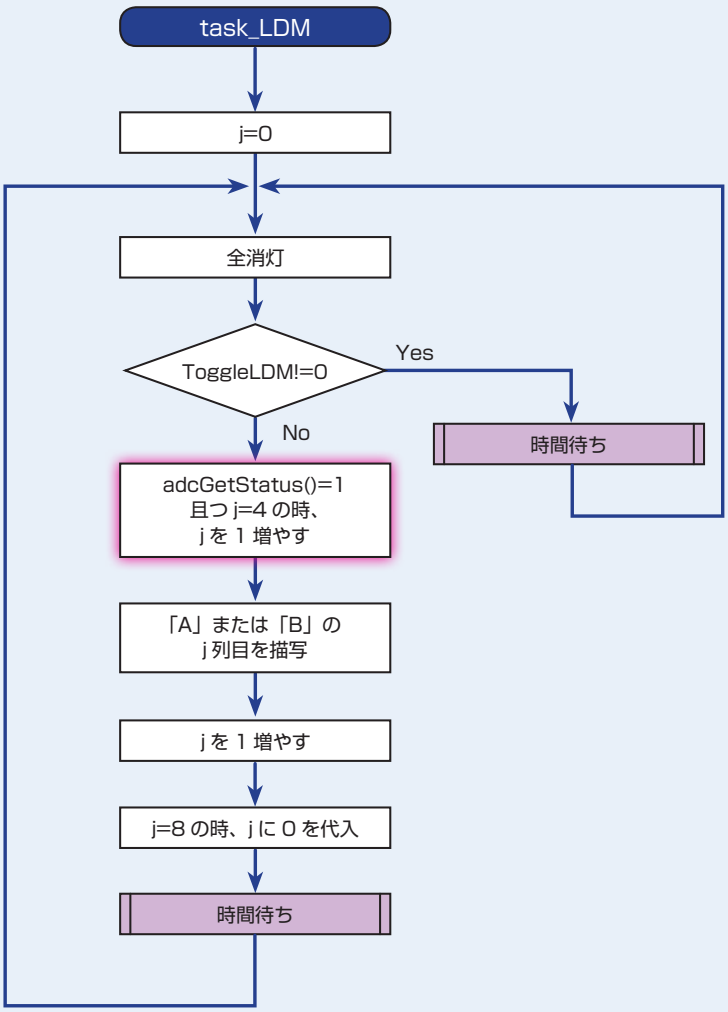
以下は、課題 13-1 を実現するためのフローチャート例です。ここでは、ADC を扱うための関数として
・ADC 関連の初期設定を行う「initADC」
・ADC を実行し ADC 値を取得する「getADC」
・ADC が実行中か否かを取得する「getADCStatus」
の３つの関数を準備することにしています。また、ADC を実行中か否かは変数「adcCnvPrg」で判断するようにしています。これらの関数はファイル「ADC.c」内に作成することにします。その他の関数・タスクは「フローチャート 12」と同じです。



温度のブラウザへの表示

フローチャート 13-1




task_LDM
j=0
全消灯
ToggleLDM!=0
Yes
No
時間待ち
adcGetStatus()=1
且つ j=4 の時、
j を 1 増やす
「A」または「B」の
j 列目を描写
j を 1 増やす
j=8 の時、j に 0 を代入
時間待ち

温度のブラウザへの表示

ライブラリ関数 13-1

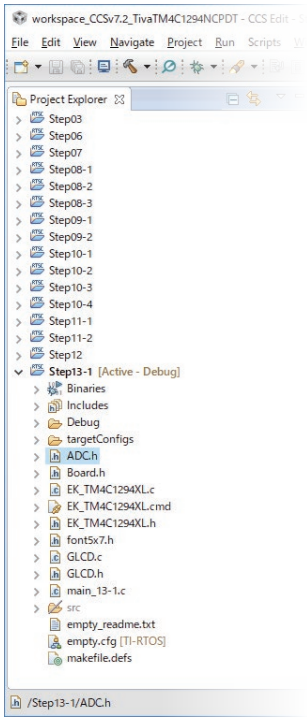
本 STEP では、ADC を使用するために以下のライブラリ関数を使っています。

- GPIOPinTypeADC()
- ADCSequenceConfigure()
- ADCSequenceEnable()
- ADCSequenceStepConfigure()
- ADCProcessorTrigger()
- ADCIntStatus()
- ADCSequenceDataGet()

これらの関数に関する詳細は、「No.07 ARM チャレンジャー入門編」をご覧ください。または、Project Explorer からプロジェクトディレクトリ→「Includes」→「*/tirexcontent/tirtos_tivac_*/products/TivaWare_C_Series-*」→「driverlib」内の「adc.h」、「adc.c」、「gpio.h」、「gpio.c」をご覧ください。

コーディング ADC.h 13-1

ADC に必要な関数・定数の宣言は、プロジェクトディレクトリ直下にヘッダファイル「ADC.h」を作成し、その中に記述するものとします。以下は ADC.h のコーディング例です。

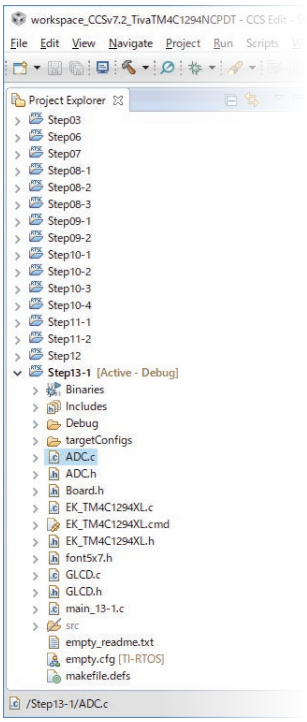


ADC.h　ADC.c　mian.c

```
1  #ifndef ADC_H_
2  #define ADC_H_
3
4  /* ADC ピンの名前 */
5  typedef enum ADC_Name
6  {
7      ADC_P1 = 0,
8      ADC_P2,
9      ADC_PCOUNT
10 } ADC_Name;
11
12 /* ADC の初期化 */
13 void initADC(void);
14
15 /* ADC 値の取得 */
16 unsigned int getADC(ADC_Name adcPinName);
17
18 /* ADC の状態（ADC を実行中か否か）の取得 */
19 int getADCStatus(void);
20
21 #endif /* ADC_H_ */
22
```

コーディング ADC.c 13-1

ADC に必要な関数のソースコードは、プロジェクトディレクトリ直下にソースファイル「ADC.c」を作成し、その中に記述するものとします。以下は ADC.c のコーディング例です。



ADC.h | ADC.c | mian.c

```c
#include <stdbool.h>
#include <stdint.h>

#include <driverlib/gpio.h>
#include <driverlib/adc.h>
#include <driverlib/sysctl.h>
#include <inc/hw_memmap.h>

#include "ADC.h"

/* ADC の状態（ADC を実行中か否か）*/
volatile int adcCnvPrg;

/* ADC 設定用構造体 */
typedef struct ADCPinConfig
{
    unsigned int port;
    unsigned int pin;
    unsigned int config;
} ADCPinConfig;

/* ADC の設定値 */
ADCPinConfig adcPinConfig[ADC_PCOUNT] = {
    {
     .port = GPIO_PORTE_BASE, .pin = GPIO_PIN_4,
     .config = ADC_CTL_CH9 | ADC_CTL_IE | ADC_CTL_END
    },
    {
     .port = GPIO_PORTE_BASE,
     .pin = GPIO_PIN_5,
     .config = ADC_CTL_CH8 | ADC_CTL_IE | ADC_CTL_END
    }
};

/* ADC の初期化 */
void initADC(void)
{
    int i; // 汎用カウンタ

    // ADC を有効化
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    // ADC として使うピンを ADC に設定
```

温度のブラウザへの表示

コーディング ADC.c 13-1

```
44      for (i = 0; i < ADC_PCOUNT; i++)
45      {
46          GPIOPinTypeADC(adcPinConfig[i].port, adcPinConfig[i].pin);
47      }
48
49      // ADC シーケンスの設定
50      ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
51
52      // ADC シーケンスの有効化
53      ADCSequenceEnable(ADC0_BASE, 3);
54
55      // adcCnvPrg の初期化
56      adcCnvPrg = 0;
57
58      // 最初の 2 回の ADC 値は捨てる
59      for(i=0;i<ADC_PCOUNT;i++)
60      {
61          getADC((ADC_Name)i);
62          getADC((ADC_Name)i);
63      }
64  }
65
66  /* ADC 値の取得 */
67  unsigned int getADC(ADC_Name adcPinName)
68  {
69      uint32_t data;
70      uint32_t pinVal;
71
72      // ADC 値取得処理開始
73      adcCnvPrg = 1;
74
75      // PE4 ピンの値を保存
76      pinVal = GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_3);
77
78      // PE4 ピンを L にする
79      GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_3, 0);
80
81      // ADC シーケンスステップの設定
82      ADCSequenceStepConfigure(ADC0_BASE, 3, 0, adcPinConfig[adcPinName].config);
83
84      // ADC 開始
85      ADCProcessorTrigger(ADC0_BASE, 3);
86
87      // ADC が終わるまで待つ
88      while (!ADCIntStatus(ADC0_BASE, 3, false))
89          ;
90
91      // ADC 値を取得
92      ADCSequenceDataGet(ADC0_BASE, 3, &data);
93
```

```
94      // PE4 ピンを元の値に戻す
95      GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_3, pinVal);
96
97      // ADC 値取得処理終了
98      adcCnvPrg = 0;
99      return data;
100 }
101
102 /* ADC の状態（ADC を実行中か否か）の取得 */
103 int getADCStatus(void)
104 {
105     return adcCnvPrg;
106 }
107
```

温度のブラウザへの表示

コーディング main.c 13-1

以下は、main.c 12 のソースコードを元にしたコーディング例です。■は main.c 12 から追加した部分です。

ADC.h | ADC.c | mian.c

これ以前の行に変更はありません

```
22 /* ボードヘッダファイル */
23 #include "Board.h"
24
25 /* ADC ヘッダファイル */
26 #include "ADC.h"
27
28 /* GLCD ヘッダファイル */
29 #include "GLCD.h"
```

この間の行に変更はありません

```
165 /*
166  * CGI ソースコード (control.cgi)
167  */
168 static int cgiControl(SOCKET s, int ContentLength, char *pArgs)
169 {
 .       .
 .       .
 .       .
198     // POST データが "get_sw" の時、押されているスイッチを返信
199     if (strcmp(buffer, "get_sw") == 0)
200     {
201         // SW1 が押されている時のテキストを送信
202         if (GPIO_read(Board_BUTTON0) == 0)
203         {
204             httpSendClientStr(s, "SW1,");
205         }
206         // SW2 が押されている時のテキストを送信
207         if(GPIO_read(Board_BUTTON1)==0) {
208             httpSendClientStr(s, "SW2,");
209         }
210     }
211
212     // POST データが "get_temp" の時、温度データを取得
213     if (strcmp(buffer, "get_temp") == 0)
214     {
215         unsigned int data; // ADC データ格納用
216         char Temp[6];      // 格納用
217         int temp;          // 温度の絶対値
218         char minus;        // 温度の符号
```

```
219
220         // 温度センサ P1 の ADC 値を取得
221         data = getADC(ADC_P1);
222         CONSOLE("ADC: %d\n", data);
223
224         // ADC 値を温度（0.1℃単位）に変換
225         temp = (data * 3.3 * 1000 / 4096 - 600);
226
227         // 温度の絶対値と符号を取得
228         minus = 0;
229         if (temp < 0)
230         {
231             temp = -temp;
232             minus = 1;
233         }
234
235         // 絶対値が 999 以上の場合は 999（± 99.9℃）とする
236         if (temp > 999)
237             temp = 999;
238         CONSOLE("Temp: %d\n", temp);
239
240         // 温度の文字列を作成
241         Temp[4] = '0' + temp % 10;          // 小数点第 1 位
242         Temp[3] = '.';                      //「.」
243         Temp[2] = '0' + (temp / 10) % 10;   // 1 の位
244         if (temp / 100 > 1)                 // 10 の位および符号
245         {
246             Temp[1] = '0' + (temp / 100) % 10;
247             Temp[0] = ' ';
248             if (minus == 1)
249                 Temp[0] = '-';
250         }
251         else
252         {
253             Temp[1] = ' ';
254             if (minus == 1)
255                 Temp[1] = '-';
256             Temp[0] = ' ';
257         }
258         Temp[5] = '\0';
259
260         // 温度の文字列を送信
261         httpSendClientStr(s, Temp);
262     }
263
264     CGI_ERROR:
265     // バッファを解放
266     if (buffer) mmBulkFree(buffer);
267
268     // 1 を返す
```

温度のブラウザへの表示

コーディング main.c 13-1

```
269     return 1;
270 }
```

この間の行に変更はありません

```
399 /*
400  * LED ドットマトリクスタスク
401  */
402 Void task_LDM(UArg arg0, UArg arg1)
403 {
 .          .
 .          .
 .          .
423         // LED ドットマトリクス描写の休止
424         if (toggleLDM != 1)
425         {
426             SLEEP(30);
427             continue;
428         }
429
430         // ADC 実行中は LDM_P4 の列を飛ばす
431         if(getADCStatus()==1 && LDM_A[j]==LDM_P4) j++;
432
433         if (charLDM == 'A')
434         {
```

この間の行に変更はありません

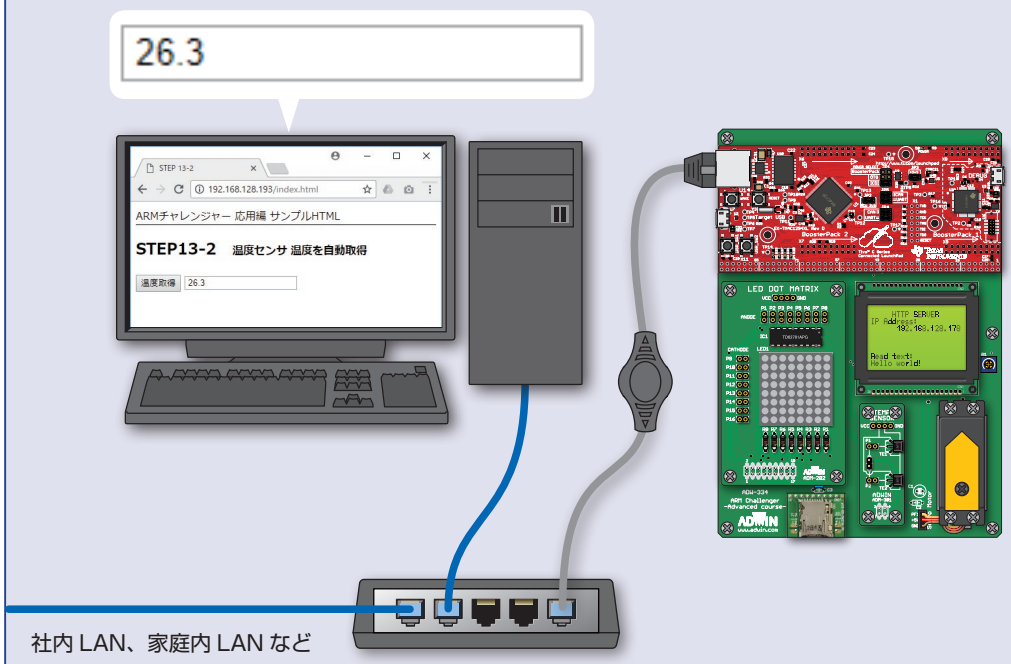
```
609 /*
610  * メイン
611  */
612
613 int main(void)
614 {
615     Task_Params taskParams;
616     PWM_Params pwmParams;
617
618     // ボードの初期設定
619     CONSOLE(" ボードの初期設定 \n");
620     Board_initGeneral();
621     Board_initGPIO();
622     Board_initPWM();
623     Board_initEMAC();
624     Board_initSDSPI();
625
626     // ADC の初期設定
627     initADC();
```

これ以降の行に変更はありません

温度のブラウザへの表示

課題 13-2

ブラウザに表示される温度が、1 秒毎に更新されるようにしましょう。



温度のブラウザへの表示

コーディング index.htm 13-2

この課題を実現するためには、マイコン側は課題 13-1 から変更する必要はありません。ブラウザ側（HTML ファイル内の JavaScript）から温度取得を定期的に行うようにすれば課題を実現できます。

以下は、index.htm 13-1 のソースコードを元にしたコーディング例です。■は index.htm 13-1 から追加した部分です（■は課題名等の変更です）。

index.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 13-2</title>
    ・
    ・
    ・
<script>
        // 周期的に実行する関数
        function do_periodic() {
                Get_SW();
                Get_Temp();
                // 関数の繰り返し
                setTimeout("do_periodic()", 1000);
        }
    ・
    ・
    ・
        <h2> STEP12 <span class="exp"> サーボモータ 位置制御 </span> </h2>
        <input type="button" value=" 左 90° " onclick='control_onclick("servo_position=-90")'>
        <input type="button" value=" 中央 0° " onclick='control_onclick("servo_position=0")'>
        <input type="button" value=" 右 90° " onclick='control_onclick("servo_position=90")'>

        <h2> STEP13-2 <span class="exp">温度センサ 温度を自動取得 </span> </h2>
        <input type="button" value=" 温度取得 " onclick="Get_Temp()">
        <input id="text_temp" type="text">
</body>
</html>
```