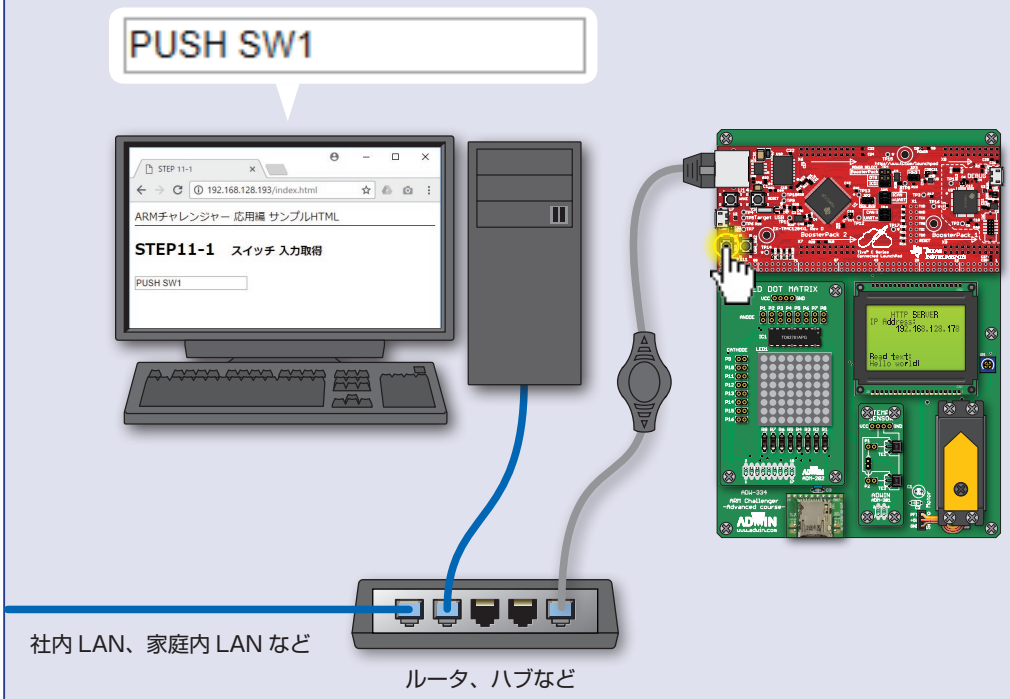
マイコンからブラウザへのメッセージ送信

学習内容

マイコンからパソコンのブラウザへメッセージ送信を行います。
マイコンボードのスイッチの状態をブラウザに表示させましょう。

課題 11-1

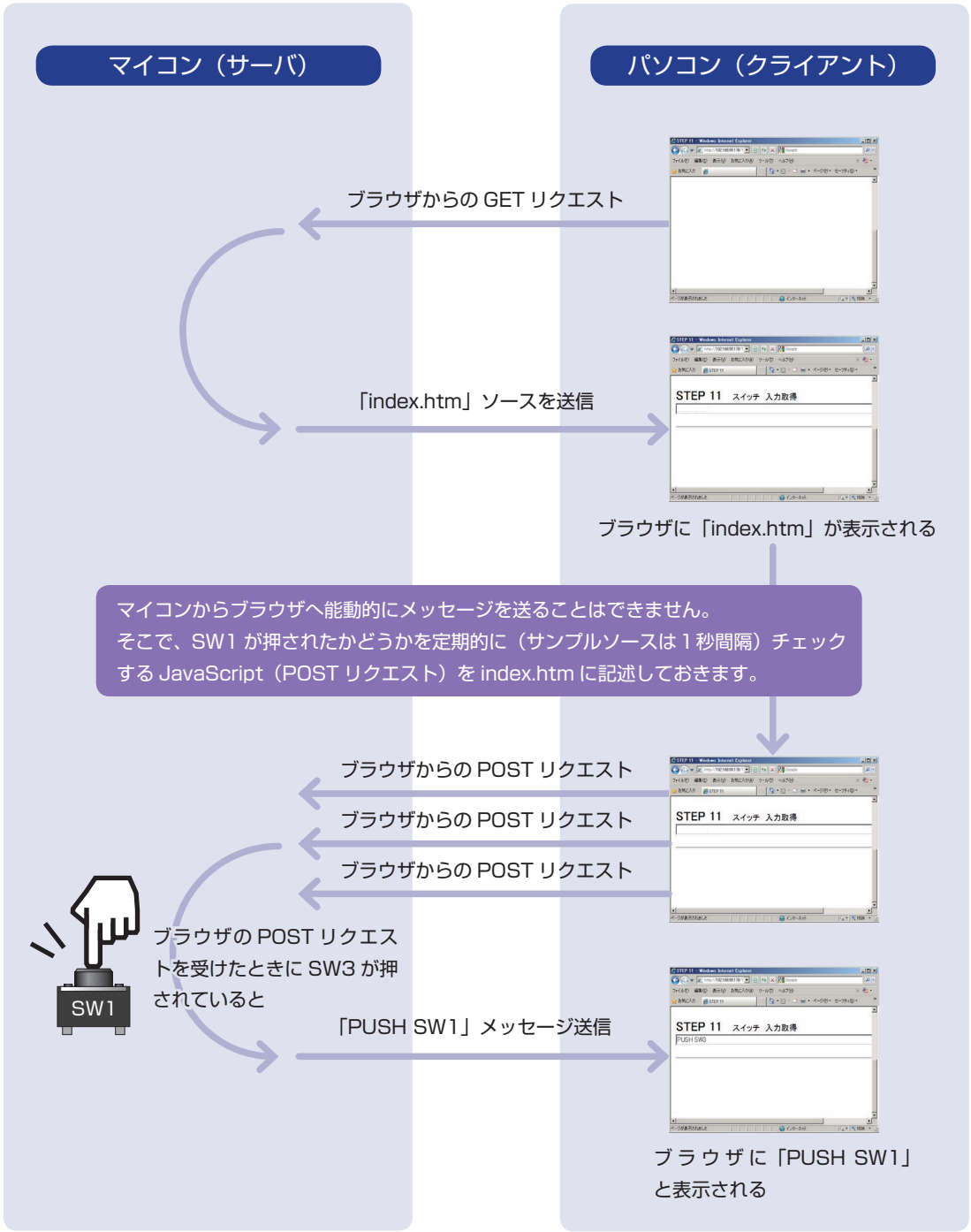
マイコンボードの「SW1」スイッチを押すと、ブラウザに「PUSH SW1」と表示されるようにしましょう。



⚠ SW はブラウザにメッセージが表示されるまで少し長めに押し続けてください。

マイコンからブラウザへのメッセージ送信

課題を整理してみましょう。 以下のフローに沿ってパソコンとマイコンでやり取りを行うよう設計します。



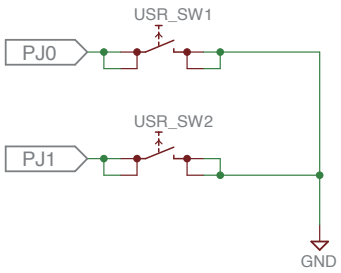
マイコンからブラウザへのメッセージ送信

配線 11-1

ユーザースイッチ（SW1 および SW2）は、マイコ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	
PJ0	USR_SW1
PJ1	USR_SW2

スイッチ OFF で High、スイッチ ON で Low になるアクティブローで配線されています。



マイコンからブラウザへのメッセージ送信

スイッチ入力でブラウザにメッセージを表示させる方法

マイコンボードのスイッチ入力をブラウザの表示に反映させるには、「サーバからのデータ取得」、「ブラウザの表示内容の変更」、「処理（関数等）の定期的な実行」が必要になります。以下では、JavaScipt でこれらを実現する方法を見ていきましょう。

サーバからのデータ取得

XMLHttpRequest によってサーバから取得したデータは、「responseText」に格納されています。以下は、取得したデータを alert() によってアラートダイアログに表示する例です。

```
var request = new XMLHttpRequest();
request.open("POST", "target.cgi", false);
request.send("get_sw");
alert(request.responseText);
```

ブラウザの表示内容の変更

JavaScript で取得したデータをブラウザの表示に反映させるには、「DOM」（Document Object Model）という仕組みを使って、HTML の内容を書き換える必要があります。

例えば、HTML ソースコード中に以下のようなテキストボックスがあったとします。

```
<input id="object1" type="text" value="sample text">
```

このソースコードをブラウザで表示すると、テキストボックスは「sample text」という文字列が書き込まれた形で表示されますが、DOM 操作ではこの文字列を別の文字列にすることが可能です。

以下は、上記テキストボックスの文字列を DOM 操作によって変更する例です。ここでは、対象の要素を id で取得しています。

```
var obj1 = document.getElementById("object1");  // id によって要素を取得
obj1.value = "changed text";    // 取得した要素の value に新しい値を設定
```

関数の定期的な実行

JavaScript でブラウザから定期的にサーバのデータを取得するには、いくつかの方法が考えられますが、ここでは例として setTimeout を用いて関数を定期的に実行する方法を見てみます。

以下は、setTimeout による繰り返し処理の実現例です。

```
function my_func() {
        /* 行いたい処理 */
        setTimeout("my_func()", 1000); // 1 秒後に my_func() を繰り返す
}
```

ここで setTimeout は、第二引数で指定した時間（ミリ秒）後に、第一引数で指定した関数を実行します。上記の例の場合、my_func() を実行して行いたい処理が完了すると、1000 ミリ秒後（1 秒後）に再度 my_func() が実行され、以降それの繰り返しになります。

なお、上記の方法を用いる場合、最初の一回については別途実行する必要があります。このような場合、HTML の body タグに onload 要素で実行したい関数を指定すると、ページが表示された時にその関数が実行されます。以下は指定例です。

```
<body onload="my_func()">
```

プレーンテキスト用の HTTP レスポンス

HTTP レスポンス中では、Web サーバが返信するデータの種類を「Content-type」で指定しました。Content-type は HTML の場合は「text/html」でしたが、HTML ではない単なる平文（プレーンテキスト）の場合は「text/plain」になります※。以下は、プレーンテキストを返信する場合の HTTP レスポンス例です。

```
HTTP/1.0 200 OK
Connection: close
Content-type: text/plain

text data・・・
```

※ ただし、今回の目的に関しては Content-type の設定の影響はなく、Content-type が「text/html」であっても動作には問題ありません。

マイコンからブラウザへのメッセージ送信

コーディング index.htm 11-1

以下は、index.htm 10-3 を元にしたコーディング例です。■は index.htm 10-3 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、「status.cgi」にコマンド名「get_sw」を POST メソッドの本文で送ることによりスイッチを取得することとします。

index.htm

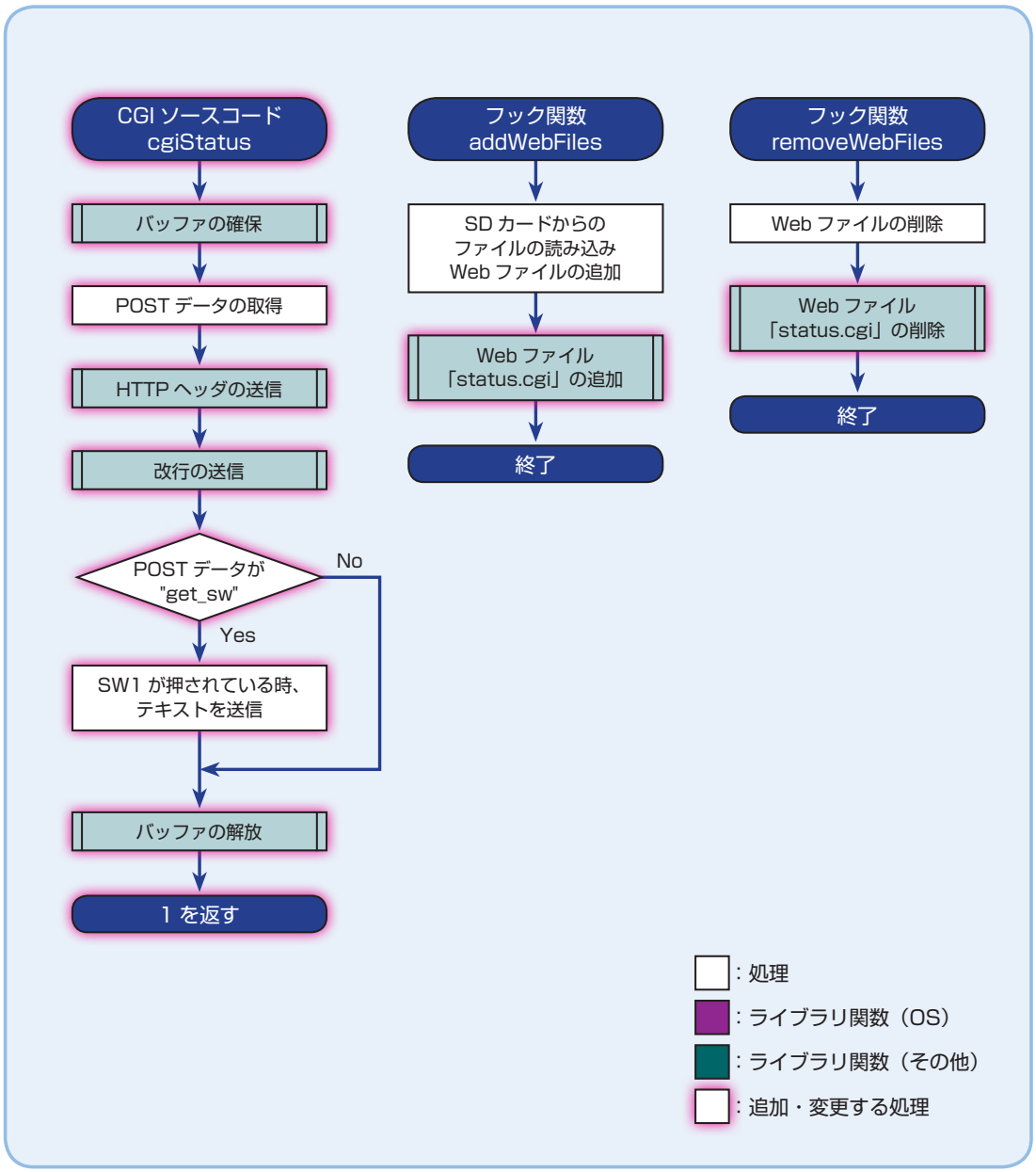
```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 11-1</title>
<style>
h1 {
        font-size: medium;
        font-weight: normal;
        border-bottom: 1px solid;
}
span.exp {
        margin-left: 1em;
        font-size: 70%;
}
</style>
<script>
        // 周期的に実行する関数
        function do_periodic() {
                Get_SW();
                // 関数の繰り返し
                setTimeout("do_periodic()", 1000);
        }
        // マイコンボードの制御
        function control_onclick(cmd) {
                var request = new XMLHttpRequest();
                request.open("POST", "control.cgi", false);
                request.send(cmd);
        }
        // マイコンボードのスイッチの取得
        function Get_SW() {
                // 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_sw");

                // 取得したテキストをテキストボックスに表示
                var text_sw = document.getElementById("text_sw");
                text_sw.value = request.responseText;
        }
</script>
</head>
```

```
43
44  <body onload="do_periodic()">
45          <h1>ARM チャレンジャー 応用編 サンプル HTML</h1>
46
47          <h2> STEP10-3 <span class="exp"> ドットマトリックス LED 文字 A/B 変更 </span> </h2>
48          <input type="button" value=" トグル LED" onclick='control_onclick("toggle_LDM")'>
49          <input type="button" value=" 文字変更 " onclick='control_onclick("changeChar_LDM")'>
50
51          <h2> STEP11-1 <span class="exp"> スイッチ 入力取得 </span> </h2>
52          <input id="text_sw" type="text">
53  </body>
54
55  </html>
```

マイコンからブラウザへのメッセージ送信

フローチャート 11-1

以下は、課題 11-1 を実現するためのフローチャート例です。その他の関数・タスクは「フローチャート 10-3」と同じです。



マイコンからブラウザへのメッセージ送信

ライブラリ関数 11-1

課題 11-1 で使用する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。

ピンの読み込み　unsigned int GPIO_read(unsigned int index)

ピンの値を読み込み、High であれば 1 を、Low であれば 0 を返します。

```
if(GPIO_read(Board_BUTTON0)==0) { /* USR_SW1 が押されている場合の処理 */ }
```

引数の設定例

- Board_BUTTON0：マイコンボードの USR_SW1 を指すマクロ。STEP10 の課題 10-1 も参照。

詳細は CCS の「Help」→「Help Contents」から「TI-RTOS for TivaC *」→「Documentation Links」→「Drivers Documents」→「TI-RTOS Drivers Runtime APIs (doxygen)」→「GPIO.h」をご覧ください。

マイコンからブラウザへのメッセージ送信

コーディング main.c 11-1

以下は、main.c 10-3 のソースコードを元にしたコーディング例です。■は main.c 10-3 から追加した部分です（■はコメント等の変更です）。

main.c

これ以前の行に変更はありません

```
74 /*
75  * CGI ソースコード（control.cgi）
76  */
77 static int cgiControl(SOCKET s, int ContentLength, char *pArgs)
78 {
79     char *buffer;
80     char len;
81
82     CONSOLE("CGI（コントロール）を開始 \n");
```

この間の行に変更はありません

```
144 /*
145  * CGI ソースコード（status.cgi）
146  */
147 static int cgiStatus(SOCKET s, int ContentLength, char *pArgs)
148 {
149     char *buffer;
150     char len;
151
152     CONSOLE("CGI（状態取得）を開始 \n");
153
154     // バッファを確保
155     buffer = (char*) mmBulkAlloc(ContentLength + 1);
156     if (!buffer)
157     {
158         CONSOLE(" バッファの確保に失敗しました！ \n");
159         goto CGI_ERROR;
160     }
161
162     // POST データを取得
163     len = recv(s, buffer, ContentLength, MSG_WAITALL);
164     if (len < ContentLength)
165     {
166         CONSOLE("POST データの取得に失敗しました！ \n");
167         goto CGI_ERROR;
168     }
169     buffer[ContentLength] = '\0';
170
171     CONSOLE("POST データ :\n%s\n\n", buffer);
```

```
172
173     // HTTP ヘッダの送信
174     httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
175     httpSendClientStr(s, CRLF);
176
177     // POST データが "get_sw" の時、押されているスイッチを返信
178     if (strcmp(buffer, "get_sw") == 0)
179     {
180         // SW1 が押されている時のテキストを送信
181         if (GPIO_read(Board_BUTTON0) == 0)
182         {
183             httpSendClientStr(s, "PUSH SW1");
184         }
185     }
186
187     CGI_ERROR:
188     // バッファを解放
189     if (buffer) mmBulkFree(buffer);
190
191     // 1 を返す
192     return 1;
193 }
194
195 /*
196  * ユーザータスク
197  */
```

この間の行に変更はありません

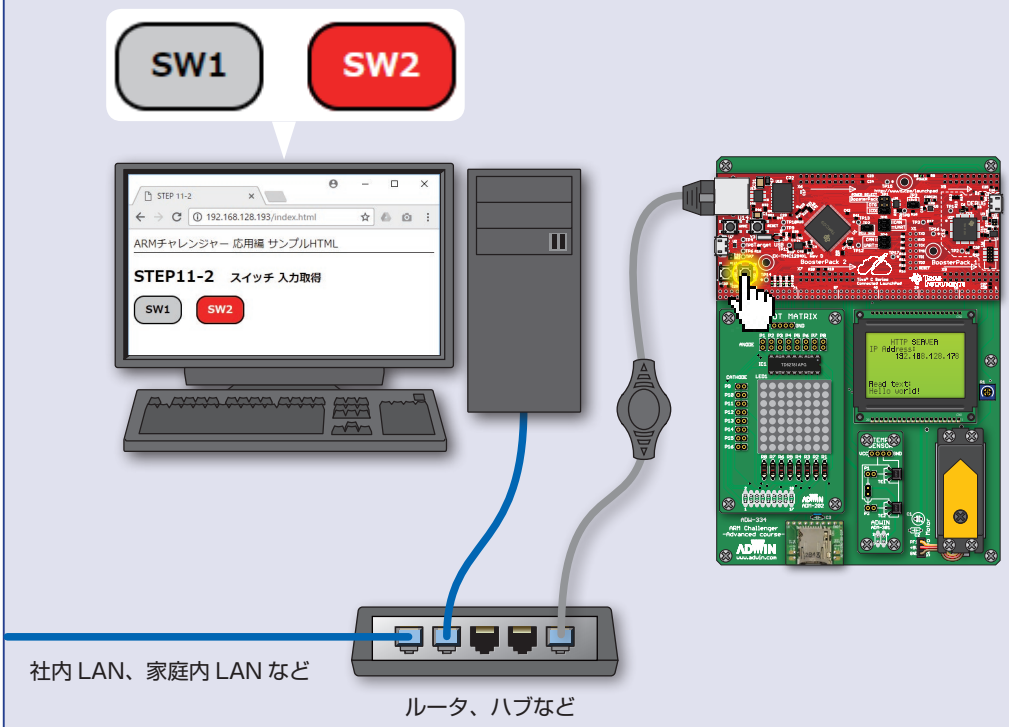
```
459 Void addWebFiles()
460 {
  .       .
  .       .
  .       .
515     /* Web ファイルの追加 */
516     efs_createfile("index.html", strlen(page), (UINT8 *) page);
517     efs_createfile("control.cgi", 0, (UINT8 *) cgiControl);
518     efs_createfile("status.cgi", 0, (UINT8 *) cgiStatus);
519 }
520
521 Void removeWebFiles()
522 {
523     /* Web ファイルの削除 */
524     efs_destroyfile("index.html");
525     efs_destroyfile("control.cgi");
526     efs_destroyfile("status.cgi");
527 }
```

これ以降の行に変更はありません

マイコンからブラウザへのメッセージ送信

課題 11-2

マイコンボードの「SW1」スイッチまたは「SW2」スイッチを押すと、ブラウザ上でスイッチの押された様子がグラフィカルに分かるようにしましょう。



マイコンからブラウザへのメッセージ送信

複数の情報を含むデータの解析

ここでは、2 つのスイッチ SW1 と SW2 が同時に押されている場合に、「スイッチが 2 つ押されている」という情報をやりとりすることを考えてみましょう。

これを実現する方法のひとつとしては、「スイッチが 2 つ押されている」という状態を新たに定義し、それをやりとりするということが考えられます。もう一つの方法としては、「スイッチ SW1 が押されている」という情報と「スイッチ SW2 が押されている」という情報を単純につなぎ合わせて送信し、受信側ではひとまとまりに送られてきたその情報を分割して元の 2 つの情報を取得するということが考えられます。

ここでは、後者の方法でデータをやりとりすることを考えてみます。この方法であれば、例えばスイッチが 3 つ以上になった場合など、より複雑な情報をやりとりしたい場合にも容易に対応することができます。以下は、JavaScript での実現例です。

文字列の分割

JavaScript では、文字列の分割は split メソッドで行うことができます。以下は、split メソッドの使用例です（text1 で設定しているのは、「スイッチ SW1 が押されている」という情報と「スイッチ SW2 が押されている」という情報を単純につなぎ合わせた文字列の例です）。

```
var text1 = "SW1,SW2,";
var array1 = text1.split(",");
```

この例では文字列 text1 は「,」で分割され、array1 には以下のような配列が代入されることになります。

```
array1 = ["SW1", "SW2", ""];
```

マイコンからブラウザへのメッセージ送信

複数の情報を含むデータの解析

配列の各要素へのアクセス

配列の各要素にアクセスするには、for 文でアクセスする方法の他に、「forEach()」というメソッドを使う方法もあります。以下は、forEach メソッドの使用例です。

```
var array1 = [ "SW1", "SW2", "" ];
array1.forEach(function(val) {
        /* 各要素を取得して処理する内容。各要素は val に代入されている。 */
});
```

文字列を分割して各要素を取得したい場合、以下のようにひとつにまとめて書くこともできます。

```
var text1 = "SW1,SW2,";
text1.split(",").forEach(function(val) {
        /* 各要素を取得して処理する内容。各要素は val に代入されている。 */
});
```

スタイルの動的な変更

HTML 文章の見た目はスタイルシートで設定しますが、このスタイルシートを JavaScript で変更し、見た目を動的に変更することも可能です。

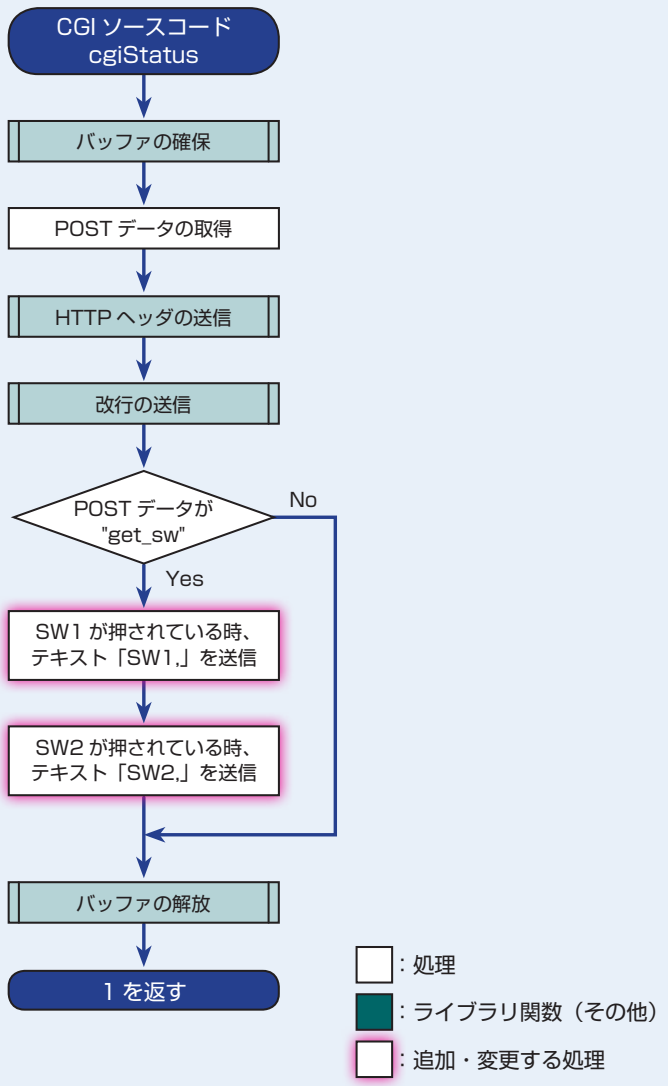
スタイルの動的な変更は、対象の要素に設定されているスタイルを直接編集する他、スタイルをあらかじめクラスで設定しておき、要素のクラスを変更するという方法も考えられます。以下は、後者によるスタイル変更例です。

```
var target1 = document.getElementById("targetId");
if(/* 条件 */) {
        target1.className = "style1"; // class を "style1" に設定
} else {
        target1.className = "style2"; // class を "style1" に設定
}
```

マイコンからブラウザへのメッセージ送信

フローチャート 11-2

以下は、課題 11-2 を実現するためのフローチャート例です。ここでは、SW1 が押されている時に文字列「SW1,」を、SW2 が押されている時に文字列「SW2,」を送信し、複数のスイッチが押されている時は文字列を「,」で分割すれば押されているスイッチが判別できるようにしています。その他のタスクは「フローチャート 10-2」と同じです。その他のタスクは「フローチャート 11-1」と同じです。



マイコンからブラウザへのメッセージ送信

コーディング index.htm 11-2

以下は、index.htm 11-1 を元にしたコーディング例です。ここでは、スイッチの入力情報をボタン状に配置し、スタイル（色および背景）を変えることでスイッチの状態をグラフィカルに示すようにしています。■は index.htm 11-1 から追加・変更した部分です（■は課題名等の変更です）。

index.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset='Shift_JIS'>
<title>STEP 11-2</title>
<style>
h1 {
        font-size: medium;
        font-weight: normal;
        border-bottom: 1px solid;
}

span.exp {
        margin-left: 1em;
        font-size: 70%;
}

span.sw {
        font-weight: bold;
        border: 2px solid black;
        border-radius: 1em;
        padding: 0.5em 1em;
        background-color: lightgray;
        margin-right: 1em;
}

span.sw_on {
        color: white;
        background-color: red;
}
</style>
<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);
        }
        // マイコンボードの制御
        function control_onclick(cmd) {
                var request = new XMLHttpRequest();
                request.open("POST", "control.cgi", false);
                request.send(cmd);
        }
```

```
45          // マイコンボードのスイッチの取得
46          function Get_SW() {
47                  // サーバからテキストを取得
48                  var request = new XMLHttpRequest();
49                  request.open("POST", "status.cgi", false);
50                  request.send("get_sw");
51
52                  // 取得したテキストをテキストボックスに表示
53                  var text_sw = document.getElementById("text_sw");
54                  text_sw.value = request.responseText;
55
56                  // 取得したテキストを解析し、スイッチの表示に反映
57                  var sw1_class = "sw";
58                  var sw2_class = "sw";
59                  request.responseText.split(",").forEach(function(val) {
60                          if (val === "SW1") sw1_class += " sw_on";
61                          if (val === "SW2") sw2_class += " sw_on";
62                  });
63                  var sw1 = document.getElementById("sw1");
64                  var sw2 = document.getElementById("sw2");
65                  sw1.className = sw1_class;
66                  sw2.className = sw2_class;
67          }
68  </script>
69  </head>
70  <body onload="do_periodic()">
71          <h1>ARM チャレンジャー 応用編 サンプル HTML</h1>
72
73          <h2> TEP10-3 <span class="exp"> ドットマトリックス LED 文字 A/B 変更 </span> </h2>
74          <input type="button" value=" トグル LED" onclick='control_onclick("toggle_LDM")'>
75          <input type="button" value=" 文字変更 " onclick='control_onclick("changeChar_LDM")'>
76
77          <h2> STEP11-1 <span class="exp"> スイッチ 入力取得 </span> </h2>
78          <input id="text_sw" type="text">
79
80          <h2> STEP11-2 <span class="exp"> スイッチ 入力取得 </span> </h2>
81          <span id="sw1" class="sw">SW1</span>
82          <span id="sw2" class="sw">SW2</span>
83
84  </body>
85  </html>
```

マイコンからブラウザへのメッセージ送信

コーディング main.c 11-2

以下は、main.c 11-2 のソースコードを元にしたコーディング例です。■は main.c 11-1 から追加・変更した部分です。SD カードから読み込む HTML ファイルが大きくなってきており、今までの設定（2048 バイト）では足りないので、ページデータの読み込みバッファも大きくしています。

main.c

これ以前の行に変更はありません

```
144 /*
145  * CGI ソースコード（status.cgi）
146  */
147 static int cgiStatus(SOCKET s, int ContentLength, char *pArgs)
148 {
 .       .
 .       .
 .       .
177     // POST データが "get_sw" の時、押されているスイッチを返信
178     if (strcmp(buffer, "get_sw") == 0)
179     {
180         // SW1 が押されている時のテキストを送信
181         if (GPIO_read(Board_BUTTON0) == 0)
182         {
183             httpSendClientStr(s, "SW1,");
184         }
185         // SW2 が押されている時のテキストを送信
186         if(GPIO_read(Board_BUTTON1)==0)
187         {
188             httpSendClientStr(s, "SW2,");
189         }
190     }
```

この間の行に変更はありません

```
464 Void addWebFiles()
465 {
466     SDSPI_Handle sdspiHandle; // SD カードマウントのハンドラ
467     SDSPI_Params sdspiParams; // SD カードマウントのパラメータ
468
469     FILE *fd; // ファイルディスクリプタ
470
471     static char page[4096]; // ページデータの読み込みバッファ
472     unsigned int bytesRead; // ファイル読み込みサイズ
```

これ以降の行に変更はありません