```
--- X:/src/SSP-Shulink-RX62N/ssp/task.c             水 6 11 18:48:58 2014
+++ X:/src/WAIT-SSP/ssp/task.c          日 6 15 08:13:44 2014
@@ -38,6 +38,8 @@
 *  わない.
 *
 */
+#include <setjmp.h>
+
 #include <stddef.h>
 #include <limits.h>

@@ -45,9 +47,9 @@
 #include "kernel_impl.h"
 #include "task.h"

+#include "kernel_cfg.h"


-
 /*
 *  トレースログマクロのデフォルト定義
 */
@@ -71,6 +73,12 @@ extern const uint_t        tinib_epriority[];      /*

 #ifdef TOPPERS_tskini

+/*-----------------------------------------
+ *  ディスパッチャーのコンテキスト
+ */
+jmp_buf disp_ctx;       //  ディスパッチャコンテキスト
+
+
 /*
 *  実行状態タスクの起動時優先度
 */
@@ -91,6 +99,12 @@ bool_t reqflg;


+
+/*
+ *  前回　実行task
+ */
+intptr_t last_ipri;
+
 /*
 *  read_primap の初期値
 */
@@ -266,15 +280,32 @@ test_dormant(uint_t ipri)

 #ifdef TOPPERS_tskini

+
+/*-----------------------------------------------------------
+ *  初期状態が起動のタスクについて
+ *  make_active() でタスクをready状態にする       takahashi
+ */
 void
 initialize_task(void)
 {
+    intptr_t ipri;
     /*
     *  レディキューのビットマップ初期化
     */
     ready_primap = init_rdypmap;

     /*  実行時優先度の初期化 */
     runtsk_ipri = IPRI_NULL;
+    last_ipri = IPRI_NULL;    //  ありえない値にする

+    /*  タスクコンテキストの設定
```

```
70  +   for(ipri = 0;ipri <  TNUM_TSKID;ipri++)
71  +   {
72  +       if (primap_test(ipri))
73  +       {
74  +           make_ctx(ipri);
75  +       }
76  +       task_wait[ipri] = 0;
77  +   }
78  +
79      /*  割込み禁止フラグの初期化*/
80      disdsp = false;
81    }
82  @@ -281,6 +312,48 @@ initialize_task(void)
83
84  #endif /* TOPPERS_tskini */
85
86  +/*-----------------------------------------------------
87  + *   コンテキストの準備をしておく                takahashi
88  + */
89  +static jmp_buf jmpp;        //  このルーチンのセーブ用
90  +
91  +void make_ctx(uint_t ipri)
92  +{
93  +   t_lock_cpu();
94  +//      printf("make_ctx ipri= %d\n",ipri);
95  +   if (setjmp(jmpp) == 0)  //  ここに戻り用
96  +   {
97  +       //  続き
98  +       //   タスクスタックに切り替える
99  +       set_task_stack(TOPPERS_TASKSTKPT(ipri ));
100 +       if (setjmp(task_ctx[ipri]) == 0)
101 +       {
102 +           /* 登録した場合*/
103 +           longjmp(jmpp,1);//  戻る
104 +       }
105 +       else
106 +       {
107 +           /*  タスク起動時*/
108 +           t_unlock_cpu();
109 +           /*  タスクに来ました*/
110 +           /*  タスク実行開始*/
111 +           (*((TASK)(tinib_task[ipri])))(tinib_exinf[ipri]);
112 +           disdsp = false;
113 +           /*  ビットマップクリア*/
114 +           primap_clear(ipri);
115 +
116 +           //タスクが終わった場合どうするのか？--> このあとは  dispacher()に行く
117 +           longjmp(disp_ctx,1);        //sta_ker  の続きに行く
118 +       }
119 +   }
120 +   else
121 +   {
122 +       //  登録終了
123 +       t_unlock_cpu();
124 +   }
125 +}
126 +
127 +
128 /*
129  *  ipri :  起動対象タスクの起動時優先度(内部表現)
130  */
131 @@ -301,6 +374,8 @@ make_active(uint_t ipri)
132           dsp = false;
133       }
134
135 +   make_ctx(ipri);
136 +
137       return dsp;
138   }
```

```
139
140 @@ -310,9 +385,9 @@ make_active(uint_t ipri)
141 /*          primap_clear(ipri);
142  *  apri :   実行開始タスクの起動時優先度
143  *    呼び出し条件：CPUロック
144 + *    このルーチンは廃止                takahashi
145  */
146
147 -#define TOPPERS_tskrun
148
149 #ifdef TOPPERS_tskrun
150
151 @@ -319,68 +394,22 @@ make_active(uint_t ipri)
152 void
153 run_task(uint_t ipri)
154  {
155 -   uint_t next_pri;    /*  次に実行開始するタスクの起動時優先度 */
156 -   uint_t saved_pri;  /*  呼び出し元タスクの起動時優先度*/
157 -
158 -   next_pri = ipri;
159 -   saved_pri = runtsk_ipri;
160 -
161 -   do {
162 -       runtsk_ipri = tinib_epriority[next_pri];
163 -
164 -       /* CPU ロック解除*/
165 -       t_unlock_cpu();
166 -
167 -       /*  タスク実行開始*/
168 -       (*((TASK)(tinib_task[next_pri])))(tinib_exinf[next_pri]);
169 -
170 -       if (t_sense_lock()) {
171 -           /*
172 -            *  CPU ロック状態でext_tskが呼ばれた場合は，CPUロックを解除し
173 -            *  てからタスクを終了する．実装上は，サービスコール内でのCPU
174 -            *  ロックを省略すればよいだけ．
175 -            */
176 -       }
177 -       else {
178 -           /*
179 -            *  このt_lock_cpuをこの下のdisdspの設定のようにしないのは，
180 -            *  CPU ロック中に再度_lock_cpuを呼ばないためである．
181 -            */
182 -           t_lock_cpu();
183 -       }
184 -
185 -       /*  割込み優先度マスクは全解除状態のはずなので，何もしない*/
186 -
187 -       /*
188 -        *  ディスパッチ禁止状態でext_tskが呼ばれた場合は，ディスパッ
189 -        *  チ許可状態にしてからタスクを終了する．
190 -        *
191 -        *  本来は以下のように記述すべきであるが，いずれにせよdisdspを
192 -        *  falseにすればいいため，単にfalseに設定する．
193 -        *
194 -        *      if (disdsp) {
195 -        *          disdsp = false;
196 -        *      }
197 -        */
198 -       disdsp = false;
199 -
200 -       /*  ビットマップクリア*/
201 -       primap_clear(next_pri);
202 -
203 -     /*  戻り先タスクの実行時優先度より高い起動時優先度をもつタスクが起動されたか*/
204 -   } while((!primap_empty()) && (saved_pri > (next_pri = search_schedtsk())));
205 -
206 -   runtsk_ipri = saved_pri;
207  }
```

```
208 -
209  #endif /* TOPPERS_tskrun */
210
211  /*
212   *    この関数は全割込みロック状態と同等の状態でsta_ker から呼ばれる
213  + *     このルーチンは廃止しようと思ったがdispach のみそのまま                takahashi
214   */
215
216  #define TOPPERS_tsk_dsp
217  #ifdef TOPPERS_tsk_dsp
218  +void dispatch(intptr_t ipri)
219  +{
220  +   last_ipri = ipri;
221  +   runtsk_ipri = ipri;
222  +   longjmp(task_ctx[ipri],1);
223  +}
224
225  void
226  dispatcher(void)
227  @@ -388,12 +417,99 @@ dispatcher(void)
228      do {
229          if(!primap_empty()) {
230              /*   タスクの開始*/
231  -           run_task(search_schedtsk());
232  +           //run_task(search_schedtsk());
233  +           dispatch(search_schedtsk());          //  これからは帰ってこない
234          }
235          else {
236  +           /*  実行時優先度の初期化 */
237  +           runtsk_ipri = IPRI_NULL;
238  +           last_ipri = IPRI_NULL;    //  ありえない値にする
239              idle_loop();
240          }
241      } while(true);
242   }
243
244  +ER
245  +wai_tsk(void)
246  +{
247  +   ER          ercd;
248  +   uint_t      tskpri;
249  +
250  +//     LOG_ACT_TSK_ENTER(tskid);
251  +//     CHECK_TSKCTX_UNL();
252  +//     CHECK_TSKID_SELF(tskid);
253  +
254  +   tskpri = get_ipri_self(TSK_SELF);
255  +   //tskpri = runtsk_ipri;
256  +   t_lock_cpu();
257  +   task_wait[tskpri] = 1;        //wait  状態
258  +   primap_clear(tskpri);         //  レディQから削除
259  +
260  +   //  ここのコンテキクストを登録
261  +   if (setjmp(task_ctx[tskpri]) == 0)
262  +   {
263  +       /* 登録した場合*/
264  +       longjmp(disp_ctx,1);          //sta_ker   の続きに行く
265  +   }
266  +   else
267  +   {
268  +       //   タスク復帰した場合
269  +       t_unlock_cpu();
270  +       return(ercd);
271  +   }
272  +
273  +   t_unlock_cpu();
274  +
275  +  error_exit:
276  +
```

```
277  +   return(ercd);
278  +}
279  +
280  +ER
281  +go_tsk(ID tskid)
282  +{
283  +   ER          ercd;
284  +   uint_t      self_tskpri; //  呼び出しタスク
285  +   uint_t      ipri;           //go   するタスク
286  +
287  +
288  +//      LOG_ACT_TSK_ENTER(tskid);
289  +//      CHECK_TSKCTX_UNL();
290  +//      CHECK_TSKID_SELF(tskid);
291  +
292  +   ipri = get_ipri(tskid);
293  +   self_tskpri = get_ipri_self(TSK_SELF);
294  +   t_lock_cpu();
295  +   if (task_wait[ipri] == 0)
296  +   {
297  +       t_unlock_cpu();
298  +       return(E_OBJ);
299  +   }
300  +
301  +   task_wait[ipri] = 0;        //wait  状態解除
302  +   primap_set(ipri);       //  レディQ追加
303  +
304  +
305  +
306  +   //  ここのコンテキクストを登録
307  +   if (setjmp(task_ctx[self_tskpri]) == 0)
308  +   {
309  +       /* 登録した場合*/
310  +       longjmp(disp_ctx,1);          //sta_ker   の続きに行く
311  +   }
312  +   else
313  +   {
314  +       //   タスク復帰した場合
315  +       t_unlock_cpu();
316  +       return(ercd);
317  +   }
318  +
319  +   t_unlock_cpu();
320  +
321  +  error_exit:
322  +
323  +   return(ercd);
324  +}
325  +
326  +
327   #endif /* TOPPERS_tsk_dsp */
328
```