```c
/*
 * (C) Copyright 2002
 * Sysgo Real-Time Solutions, GmbH <www.elinos.com>
 * Marius Groeger <mgroeger@sysgo.de>
 *
 * (C) Copyright 2002
 * Sysgo Real-Time Solutions, GmbH <www.elinos.com>
 * Alex Zuepke <azu@sysgo.de>
 *
 * (C) Copyright 2002
 * Gary Jennejohn, DENX Software Engineering, <gj@denx.de>
 *
 * See file CREDITS for list of people who contributed to this
 * project.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston,
 * MA 02111-1307 USA
 */

#include <common.h>
#include <arm920t.h>
#if defined(CONFIG_S3C2400)
#include <s3c2400.h>
#elif defined(CONFIG_S3C2410)
#include <s3c2410.h>
#endif

#include <asm/proc-armv/ptrace.h>

extern void reset_cpu(ulong addr);
int timer_load_val = 0;

/* macro to read the 16 bit timer */
static inline ulong READ_TIMER(void)
{
    S3C24X0_TIMERS * const timers = S3C24X0_GetBase_TIMERS();

    return (timers->TCNTO4 & 0xffff);
}

#ifdef CONFIG_USE_IRQ // undefined in smdk2410.h, ghcstop
/* enable IRQ interrupts */
void enable_interrupts (void)
{
    unsigned long temp;
    __asm__ __volatile__("mrs %0, cpsr\n"
                 "bic %0, %0, #0x80\n"
                 "msr cpsr_c, %0"
                 : "=r" (temp)
                 :
                 : "memory");
}


/*
 * disable IRQ/FIQ interrupts
 * returns true if interrupts had been enabled before we disabled them
 */
int disable_interrupts (void)
{
    unsigned long old,temp;
    __asm__ __volatile__("mrs %0, cpsr\n"
                 "orr %1, %0, #0xc0\n"
                 "msr cpsr_c, %1"
                 : "=r" (old), "=r" (temp)
                 :
                 : "memory");
    return (old & 0x80) == 0;
}
#else
```

```
 83  void enable_interrupts (void)
 84  {
 85      return;
 86  }
 87  int disable_interrupts (void)
 88  {
 89      return 0;
 90  }
 91  #endif
 92
 93
 94
 95  void bad_mode (void)
 96  {
 97      panic ("Resetting CPU ...\n");
 98      reset_cpu (0);
 99  }
100
101  void show_regs (struct pt_regs *regs)
102  {
103      unsigned long flags;
104      const char *processor_modes[] = {
105      "USER_26",  "FIQ_26",   "IRQ_26",   "SVC_26",
106      "UK4_26",   "UK5_26",   "UK6_26",   "UK7_26",
107      "UK8_26",   "UK9_26",   "UK10_26",  "UK11_26",
108      "UK12_26",  "UK13_26",  "UK14_26",  "UK15_26",
109      "USER_32",  "FIQ_32",   "IRQ_32",   "SVC_32",
110      "UK4_32",   "UK5_32",   "UK6_32",   "ABT_32",
111      "UK8_32",   "UK9_32",   "UK10_32",  "UND_32",
112      "UK12_32",  "UK13_32",  "UK14_32",  "SYS_32",
113      };
114
115      flags = condition_codes (regs);
116
117      printf ("pc : [<%08lx>]    lr : [<%08lx>]\n"
118          "sp : %08lx  ip : %08lx  fp : %08lx\n",
119          instruction_pointer (regs),
120          regs->ARM_lr, regs->ARM_sp, regs->ARM_ip, regs->ARM_fp);
121      printf ("r10: %08lx  r9 : %08lx  r8 : %08lx\n",
122          regs->ARM_r10, regs->ARM_r9, regs->ARM_r8);
123      printf ("r7 : %08lx  r6 : %08lx  r5 : %08lx  r4 : %08lx\n",
124          regs->ARM_r7, regs->ARM_r6, regs->ARM_r5, regs->ARM_r4);
125      printf ("r3 : %08lx  r2 : %08lx  r1 : %08lx  r0 : %08lx\n",
126          regs->ARM_r3, regs->ARM_r2, regs->ARM_r1, regs->ARM_r0);
127      printf ("Flags: %c%c%c%c",
128          flags & CC_N_BIT ? 'N' : 'n',
129          flags & CC_Z_BIT ? 'Z' : 'z',
130          flags & CC_C_BIT ? 'C' : 'c', flags & CC_V_BIT ? 'V' : 'v');
131      printf ("  IRQs %s  FIQs %s  Mode %s%s\n",
132          interrupts_enabled (regs) ? "on" : "off",
133          fast_interrupts_enabled (regs) ? "on" : "off",
134          processor_modes[processor_mode (regs)],
135          thumb_mode (regs) ? " (T)" : "");
136  }
137
138  void do_undefined_instruction (struct pt_regs *pt_regs)
139  {
140      printf ("undefined instruction\n");
141      show_regs (pt_regs);
142      bad_mode ();
143  }
144
145  void do_software_interrupt (struct pt_regs *pt_regs)
146  {
147      printf ("software interrupt\n");
148      show_regs (pt_regs);
149      bad_mode ();
150  }
151
152  void do_prefetch_abort (struct pt_regs *pt_regs)
153  {
154      printf ("prefetch abort\n");
155      show_regs (pt_regs);
156      bad_mode ();
157  }
158
159  void do_data_abort (struct pt_regs *pt_regs)
160  {
161      printf ("data abort\n");
162      show_regs (pt_regs);
163      bad_mode ();
164  }
```

```
165
166 void do_not_used (struct pt_regs *pt_regs)
167 {
168     printf ("not used\n");
169     show_regs (pt_regs);
170     bad_mode ();
171 }
172
173 void do_fiq (struct pt_regs *pt_regs)
174 {
175     printf ("fast interrupt request\n");
176     show_regs (pt_regs);
177     bad_mode ();
178 }
179
180 void do_irq (struct pt_regs *pt_regs)
181 {
182     printf ("interrupt request\n");
183     show_regs (pt_regs);
184     bad_mode ();
185 }
186
187 static ulong timestamp;
188 static ulong lastdec;
189
190 /* ghcstop */
191 int interrupt_init (void)
192 {
193     /*
194         // include/s3c24x0.h
195
196         // PWM TIMER (see manual chapter 10)
197     typedef struct {
198             S3C24X0_REG32   TCNTB;
199             S3C24X0_REG32   TCMPB;
200             S3C24X0_REG32   TCNTO;
201     } S3C24X0_TIMER;
202
203     typedef struct {
204             S3C24X0_REG32   TCFG0;
205             S3C24X0_REG32   TCFG1;
206             S3C24X0_REG32   TCON;
207             S3C24X0_TIMER   ch[4];
208             S3C24X0_REG32   TCNTB4;
209             S3C24X0_REG32   TCNTO4;
210     }  S3C24X0_TIMERS;
211         */
212     S3C24X0_TIMERS * const timers = S3C24X0_GetBase_TIMERS(); // include/s3c2410.h, timer register b
213
214     /*
215          * use PWM Timer 4 because it has no output
216          * : PWM(Pulse Width Modulation) timer 4는 아무런 output도 없기 때문에 이것을 사용한다.
217            ==> output pin이 없으므로 내부용으로만 쓰이고, TOUT보다는 interrupt interval timer
218            로서 쓰는것 같다.
219          */
220     /*
221          * prescaler for Timer 4 is 16
222          * : timer 4에 prescaler 16을 사용한다.
223          *   여기의 [8:15]비트는 timer 2,3,4 에 관한 prescaler설정이다. prescaler를
224          *   15(0x0f)로 설정을 하나 메뉴얼을 보면 실제로 사용되는 값은 prescaler+1의
225          *   값이 사용이 되기 때문에 16으로 얘기하는 것 같다.
226          *
227          *   메뉴얼 10-11
228          */
229     timers->TCFG0 = 0x0f00;
230
231     // timer 4의 TCNTBn으로 로드할 값을 설정하는 부분
232     if (timer_load_val == 0) // 설정이 안되어 있다면
233     {
234         /*
235          * for 10 ms clock period @ PCLK with 4 bit divider = 1/2
236          * (default) and prescaler = 16. Should be 10390
237          * @33.25MHz and 15625 @ 50 MHz
238          : 4bit divider가 1/2(디폴트 값, TCFG1 [19:16]이 0b0000)이고
239            prescaler가 16일 경우 10ms 클럭 주기를 가지려면 TCNTBn값이
240            10390@33.25MHz이거나 15625 @ 50 MHz이 되야한다.
241
242          ※ smdk2410.h의 CFG_HZ세팅도 변화를 줘야한다. 여기를 바꿀 경우
243          거기에 있는 것과 여기의 세팅을 맞춰줘야 함.
244
245          ※ divider value는 TCFG1에서 하면 된다.여기서는 그냥 레지스터 디폴트
246          값으로 사용을 하고, 만일 디바이더값을 바꾸고 싶으면 TCFG1레지스터
```

```
247           의 값을 변경시켜줄 것.
248
249           아무래도 divider값은 sw로 넣어주면 알아서 설정하는거 같은데
250           왜냐하면 PCLK랑 prescaler는 설정해주는 레지스터가 있기 때문이다.
251
252           메뉴얼을 보면 설정은
253           Timer input clock freq = PCLK/(prescaler+1)/(divider value)
254           여기서 prescaler = 0~255고, divider value는 = 2,4,8,16 이다.
255
256           10ms = 10 x 0.001 = 0.01초
257
258                   10ms (interrupt)period일 경우 실제 clock은
259                   T(주기) = 1/F(주파수) 이므로
260                   주파수는 1/T이 되므로 주파수는 1/0.01 = 100이 된다.
261                   즉, 타이머 입력 클럭이 100번 쳐야 인터럽트가 발생한다는 얘기이다.
262
263                   이렇게 되면 실제 클럭의 100이 10ms 타이머의 1 클럭이 된다는 얘기이다.
264                   즉, timer interrupt clock이라고 봐도 무방하고, 이것은 타이머 입력
265                   클럭의 1/100 이다. divider가 100이 된거겠지용....
266
267                   1: 100(실제 clock) = x : timer input clock
268
269                   x = timer input clock / 100
270
271                   x = PCLK/16/2/100 = PCLK /(16 * 2 * 200)
272
273                   이렇게 해서 timer4에 인터럽트 주기로 넣어줘야할 count값이 결정된다.
274           */
275           timer_load_val = get_PCLK()/(2 * 16 * 100);
276       }
277
278       /* load value for 10 ms timeout */
279       lastdec = timers->TCNTB4 = timer_load_val;
280       /* auto load, manual update of Timer 4 */
281       timers->TCON = (timers->TCON & ~0x0700000) | 0x600000;
282       /* auto load, start Timer 4 */
283       timers->TCON = (timers->TCON & ~0x0700000) | 0x500000;
284       timestamp = 0;
285
286       return (0);
287   }
288
289   /*
290    * timer without interrupts
291    */
292
293   void reset_timer (void)
294   {
295       reset_timer_masked ();
296   }
297
298   ulong get_timer (ulong base)
299   {
300       return get_timer_masked () - base;
301   }
302
303   void set_timer (ulong t)
304   {
305       timestamp = t;
306   }
307
308   void udelay (unsigned long usec)
309   {
310       ulong tmo;
311
312       tmo = usec / 1000;
313       tmo *= (timer_load_val * 100);
314       tmo /= 1000;
315
316       tmo += get_timer (0);
317
318       while (get_timer_masked () < tmo)
319           /*NOP*/;
320   }
321
322   void reset_timer_masked (void)
323   {
324       /* reset time */
325       lastdec = READ_TIMER();
326       timestamp = 0;
327   }
328
```

```
329 ulong get_timer_masked (void)
330 {
331     ulong now = READ_TIMER();
332
333     if (lastdec >= now) {
334         /* normal mode */
335         timestamp += lastdec - now;
336     } else {
337         /* we have an overflow ... */
338         timestamp += lastdec + timer_load_val - now;
339     }
340     lastdec = now;
341
342     return timestamp;
343 }
344
345 void udelay_masked (unsigned long usec)
346 {
347     ulong tmo;
348
349     tmo = usec / 1000;
350     tmo *= (timer_load_val * 100);
351     tmo /= 1000;
352
353     reset_timer_masked ();
354
355     while (get_timer_masked () < tmo)
356         /*NOP*/;
357 }
358
359 /*
360  * This function is derived from PowerPC code (read timebase as long long).
361  * On ARM it just returns the timer value.
362  */
363 unsigned long long get_ticks(void)
364 {
365     return get_timer(0);
366 }
367
368 /* ghcstop */
369 /*
370  * This function is derived from PowerPC code (timebase clock frequency).
371  * On ARM it returns the number of timer ticks per second.
372  : 이 함수는 PowerPC 코드(timebase clock frequency)에서 가져온 것이다.
373    ARM에서는 초당 타이머 tick 갯수를 리턴한다.
374    ==> 아마도 ppcboot와 armboot를 합쳐서 u-boot로 만드는 과정에서 일관성을
375    유지하려고 만든것 같다....실제로 이런코드들이 많다....ex> dram_init()
376  */
377 ulong get_tbclk (void)
378 {
379     ulong tbclk;
380
381 #if defined(CONFIG_SMDK2400) || defined(CONFIG_TRAB)
382     tbclk = timer_load_val * 100;
383 #elif defined(CONFIG_SMDK2410) || defined(CONFIG_VCMA9)
384     tbclk = CFG_HZ; /*=> smdk2410.h에 정의된 값, 위의 interrupt_init()함수에서의
385                        세팅등을 참조할 것*/
386 #else
387 #   error "tbclk not configured"
388 #endif
389
390     return tbclk;
391 }
392
```