104

```
/*
*
* 3780i.c -- helper routines for the 3780i DSP
*
*
* Written By: Mike Sullivan IBM Corporation
*
* Copyright (C) 1999 IBM Corporation
*
* 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.
*
* NO WARRANTY
* THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT
* LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT,
* MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is
* solely responsible for determining the appropriateness of using and
* distributing the Program and assumes all risks associated with its
* exercise of rights under this Agreement, including but not limited to
* the risks and costs of program errors, damage to or loss of data,
* programs or equipment, and unavailability or interruption of operations.
*
* DISCLAIMER OF LIABILITY
* NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
* TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED
* HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES
*
* 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
*
*
* 10/23/2000 - Alpha Release
*       First release to the public
*/

#include <linux/version.h>
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/unistd.h>
#include <linux/delay.h>
#include <linux/ioport.h>
#include <linux/init.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/system.h>
#include <asm/irq.h>
#include <asm/bitops.h>
#include "smapi.h"
#include "mwavedd.h"
#include "3780i.h"

static spinlock_t dsp_lock = SPIN_LOCK_UNLOCKED;
static unsigned long flags;


static void PaceMsaAccess(unsigned short usDspBaseIO)
{
        if(current->need_resched)
                schedule();
        udelay(100);
        if(current->need_resched)
                schedule();
}

unsigned short dsp3780I_ReadMsaCfg(unsigned short usDspBaseIO,
                                   unsigned long ulMsaAddr)
{
        unsigned short val;

        PRINTK_3(TRACE_3780I,
                "3780i::dsp3780I_ReadMsaCfg entry usDspBaseIO %x ulMsaAddr %lx\n",
                usDspBaseIO, ulMsaAddr);

        spin_lock_irqsave(&dsp_lock, flags);
        OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulMsaAddr);
        OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulMsaAddr >> 16));
        val = InWordDsp(DSP_MsaDataDSISHigh);
```

```
91          spin_unlock_irqrestore(&dsp_lock, flags);
92
93          PRINTK_2(TRACE_3780I, "3780i::dsp3780I_ReadMsaCfg exit val %x\n", val);
94
95          return val;
96  }
97
98  void dsp3780I_WriteMsaCfg(unsigned short usDspBaseIO,
99                            unsigned long ulMsaAddr, unsigned short usValue)
100 {
101
102         PRINTK_4(TRACE_3780I,
103                 "3780i::dsp3780i_WriteMsaCfg entry usDspBaseIO %x ulMsaAddr %lx usValue %x\n",
104                 usDspBaseIO, ulMsaAddr, usValue);
105
106         spin_lock_irqsave(&dsp_lock, flags);
107         OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulMsaAddr);
108         OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulMsaAddr >> 16));
109         OutWordDsp(DSP_MsaDataDSISHigh, usValue);
110         spin_unlock_irqrestore(&dsp_lock, flags);
111 }
112
113 void dsp3780I_WriteGenCfg(unsigned short usDspBaseIO, unsigned uIndex,
114                           unsigned char ucValue)
115 {
116         DSP_ISA_SLAVE_CONTROL rSlaveControl;
117         DSP_ISA_SLAVE_CONTROL rSlaveControl_Save;
118
119
120         PRINTK_4(TRACE_3780I,
121                 "3780i::dsp3780i_WriteGenCfg entry usDspBaseIO %x uIndex %x ucValue %x\n",
122                 usDspBaseIO, uIndex, ucValue);
123
124         MKBYTE(rSlaveControl) = InByteDsp(DSP_IsaSlaveControl);
125
126         PRINTK_2(TRACE_3780I,
127                 "3780i::dsp3780i_WriteGenCfg rSlaveControl %x\n",
128                 MKBYTE(rSlaveControl));
129
130         rSlaveControl_Save = rSlaveControl;
131         rSlaveControl.ConfigMode = TRUE;
132
133         PRINTK_2(TRACE_3780I,
134                 "3780i::dsp3780i_WriteGenCfg entry rSlaveControl+ConfigMode %x\n",
135                 MKBYTE(rSlaveControl));
136
137         OutByteDsp(DSP_IsaSlaveControl, MKBYTE(rSlaveControl));
138         OutByteDsp(DSP_ConfigAddress, (unsigned char) uIndex);
139         OutByteDsp(DSP_ConfigData, ucValue);
140         OutByteDsp(DSP_IsaSlaveControl, MKBYTE(rSlaveControl_Save));
141
142         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_WriteGenCfg exit\n");
143
144
145 }
146
147 unsigned char dsp3780I_ReadGenCfg(unsigned short usDspBaseIO,
148                                   unsigned uIndex)
149 {
150         DSP_ISA_SLAVE_CONTROL rSlaveControl;
151         DSP_ISA_SLAVE_CONTROL rSlaveControl_Save;
152         unsigned char ucValue;
153
154
155         PRINTK_3(TRACE_3780I,
156                 "3780i::dsp3780i_ReadGenCfg entry usDspBaseIO %x uIndex %x\n",
157                 usDspBaseIO, uIndex);
158
159         MKBYTE(rSlaveControl) = InByteDsp(DSP_IsaSlaveControl);
160         rSlaveControl_Save = rSlaveControl;
161         rSlaveControl.ConfigMode = TRUE;
162         OutByteDsp(DSP_IsaSlaveControl, MKBYTE(rSlaveControl));
163         OutByteDsp(DSP_ConfigAddress, (unsigned char) uIndex);
164         ucValue = InByteDsp(DSP_ConfigData);
165         OutByteDsp(DSP_IsaSlaveControl, MKBYTE(rSlaveControl_Save));
166
167         PRINTK_2(TRACE_3780I,
168                 "3780i::dsp3780i_ReadGenCfg exit ucValue %x\n", ucValue);
169
170
171         return ucValue;
172 }
173
174 int dsp3780I_EnableDSP(DSP_3780I_CONFIG_SETTINGS * pSettings,
175                        unsigned short *pIrqMap,
176                        unsigned short *pDmaMap)
177 {
178         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
179         int i;
180         DSP_UART_CFG_1 rUartCfg1;
```

```
181         DSP_UART_CFG_2 rUartCfg2;
182         DSP_HBRIDGE_CFG_1 rHBridgeCfg1;
183         DSP_HBRIDGE_CFG_2 rHBridgeCfg2;
184         DSP_BUSMASTER_CFG_1 rBusmasterCfg1;
185         DSP_BUSMASTER_CFG_2 rBusmasterCfg2;
186         DSP_ISA_PROT_CFG rIsaProtCfg;
187         DSP_POWER_MGMT_CFG rPowerMgmtCfg;
188         DSP_HBUS_TIMER_CFG rHBusTimerCfg;
189         DSP_LBUS_TIMEOUT_DISABLE rLBusTimeoutDisable;
190         DSP_CHIP_RESET rChipReset;
191         DSP_CLOCK_CONTROL_1 rClockControl1;
192         DSP_CLOCK_CONTROL_2 rClockControl2;
193         DSP_ISA_SLAVE_CONTROL rSlaveControl;
194         DSP_HBRIDGE_CONTROL rHBridgeControl;
195         unsigned short ChipID = 0;
196         unsigned short tval;
197
198
199         PRINTK_2(TRACE_3780I,
200                 "3780i::dsp3780I_EnableDSP entry pSettings->bDSPEnabled %x\n",
201                 pSettings->bDSPEnabled);
202
203
204         if (!pSettings->bDSPEnabled) {
205                 PRINTK_ERROR( KERN_ERR "3780i::dsp3780I_EnableDSP: Error: DSP not enabled. Aborting.\n" );
206                 return -EIO;
207         }
208
209
210         PRINTK_2(TRACE_3780I,
211                 "3780i::dsp3780i_EnableDSP entry pSettings->bModemEnabled %x\n",
212                 pSettings->bModemEnabled);
213
214         if (pSettings->bModemEnabled) {
215                 rUartCfg1.Reserved = rUartCfg2.Reserved = 0;
216                 rUartCfg1.IrqActiveLow = pSettings->bUartIrqActiveLow;
217                 rUartCfg1.IrqPulse = pSettings->bUartIrqPulse;
218                 rUartCfg1.Irq =
219                         (unsigned char) pIrqMap[pSettings->usUartIrq];
220                 switch (pSettings->usUartBaseIO) {
221                 case 0x03F8:
222                         rUartCfg1.BaseIO = 0;
223                         break;
224                 case 0x02F8:
225                         rUartCfg1.BaseIO = 1;
226                         break;
227                 case 0x03E8:
228                         rUartCfg1.BaseIO = 2;
229                         break;
230                 case 0x02E8:
231                         rUartCfg1.BaseIO = 3;
232                         break;
233                 }
234                 rUartCfg2.Enable = TRUE;
235         }
236
237         rHBridgeCfg1.Reserved = rHBridgeCfg2.Reserved = 0;
238         rHBridgeCfg1.IrqActiveLow = pSettings->bDspIrqActiveLow;
239         rHBridgeCfg1.IrqPulse = pSettings->bDspIrqPulse;
240         rHBridgeCfg1.Irq = (unsigned char) pIrqMap[pSettings->usDspIrq];
241         rHBridgeCfg1.AccessMode = 1;
242         rHBridgeCfg2.Enable = TRUE;
243
244
245         rBusmasterCfg2.Reserved = 0;
246         rBusmasterCfg1.Dma = (unsigned char) pDmaMap[pSettings->usDspDma];
247         rBusmasterCfg1.NumTransfers =
248                 (unsigned char) pSettings->usNumTransfers;
249         rBusmasterCfg1.ReRequest = (unsigned char) pSettings->usReRequest;
250         rBusmasterCfg1.MEMCS16 = pSettings->bEnableMEMCS16;
251         rBusmasterCfg2.IsaMemCmdWidth =
252                 (unsigned char) pSettings->usIsaMemCmdWidth;
253
254
255         rIsaProtCfg.Reserved = 0;
256         rIsaProtCfg.GateIOCHRDY = pSettings->bGateIOCHRDY;
257
258         rPowerMgmtCfg.Reserved = 0;
259         rPowerMgmtCfg.Enable = pSettings->bEnablePwrMgmt;
260
261         rHBusTimerCfg.LoadValue =
262                 (unsigned char) pSettings->usHBusTimerLoadValue;
263
264         rLBusTimeoutDisable.Reserved = 0;
265         rLBusTimeoutDisable.DisableTimeout =
266                 pSettings->bDisableLBusTimeout;
267
268         MKWORD(rChipReset) = ~pSettings->usChipletEnable;
269
270         rClockControl1.Reserved1 = rClockControl1.Reserved2 = 0;
```

```
271         rClockControl1.N_Divisor = pSettings->usN_Divisor;
272         rClockControl1.M_Multiplier = pSettings->usM_Multiplier;
273
274         rClockControl2.Reserved = 0;
275         rClockControl2.PllBypass = pSettings->bPllBypass;
276
277         /* Issue a soft reset to the chip */
278         /* Note: Since we may be coming in with 3780i clocks suspended, we must keep
279         * soft-reset active for 10ms.
280         */
281         rSlaveControl.ClockControl = 0;
282         rSlaveControl.SoftReset = TRUE;
283         rSlaveControl.ConfigMode = FALSE;
284         rSlaveControl.Reserved = 0;
285
286         PRINTK_4(TRACE_3780I,
287                 "3780i::dsp3780i_EnableDSP usDspBaseIO %x index %x taddr %x\n",
288                 usDspBaseIO, DSP_IsaSlaveControl,
289                 usDspBaseIO + DSP_IsaSlaveControl);
290
291         PRINTK_2(TRACE_3780I,
292                 "3780i::dsp3780i_EnableDSP rSlaveContrl %x\n",
293                 MKWORD(rSlaveControl));
294
295         spin_lock_irqsave(&dsp_lock, flags);
296         OutWordDsp(DSP_IsaSlaveControl, MKWORD(rSlaveControl));
297         MKWORD(tval) = InWordDsp(DSP_IsaSlaveControl);
298
299         PRINTK_2(TRACE_3780I,
300                 "3780i::dsp3780i_EnableDSP rSlaveControl 2 %x\n", tval);
301
302
303         for (i = 0; i < 11; i++)
304                 udelay(2000);
305
306         rSlaveControl.SoftReset = FALSE;
307         OutWordDsp(DSP_IsaSlaveControl, MKWORD(rSlaveControl));
308
309         MKWORD(tval) = InWordDsp(DSP_IsaSlaveControl);
310
311         PRINTK_2(TRACE_3780I,
312                 "3780i::dsp3780i_EnableDSP rSlaveControl 3 %x\n", tval);
313
314
315         /* Program our general configuration registers */
316         WriteGenCfg(DSP_HBridgeCfg1Index, MKBYTE(rHBridgeCfg1));
317         WriteGenCfg(DSP_HBridgeCfg2Index, MKBYTE(rHBridgeCfg2));
318         WriteGenCfg(DSP_BusMasterCfg1Index, MKBYTE(rBusmasterCfg1));
319         WriteGenCfg(DSP_BusMasterCfg2Index, MKBYTE(rBusmasterCfg2));
320         WriteGenCfg(DSP_IsaProtCfgIndex, MKBYTE(rIsaProtCfg));
321         WriteGenCfg(DSP_PowerMgCfgIndex, MKBYTE(rPowerMgmtCfg));
322         WriteGenCfg(DSP_HBusTimerCfgIndex, MKBYTE(rHBusTimerCfg));
323
324         if (pSettings->bModemEnabled) {
325                 WriteGenCfg(DSP_UartCfg1Index, MKBYTE(rUartCfg1));
326                 WriteGenCfg(DSP_UartCfg2Index, MKBYTE(rUartCfg2));
327         }
328
329
330         rHBridgeControl.EnableDspInt = FALSE;
331         rHBridgeControl.MemAutoInc = TRUE;
332         rHBridgeControl.IoAutoInc = FALSE;
333         rHBridgeControl.DiagnosticMode = FALSE;
334
335         PRINTK_3(TRACE_3780I,
336                 "3780i::dsp3780i_EnableDSP DSP_HBridgeControl %x rHBridgeControl %x\n",
337                 DSP_HBridgeControl, MKWORD(rHBridgeControl));
338
339         OutWordDsp(DSP_HBridgeControl, MKWORD(rHBridgeControl));
340         spin_unlock_irqrestore(&dsp_lock, flags);
341         WriteMsaCfg(DSP_LBusTimeoutDisable, MKWORD(rLBusTimeoutDisable));
342         WriteMsaCfg(DSP_ClockControl_1, MKWORD(rClockControl1));
343         WriteMsaCfg(DSP_ClockControl_2, MKWORD(rClockControl2));
344         WriteMsaCfg(DSP_ChipReset, MKWORD(rChipReset));
345
346         ChipID = ReadMsaCfg(DSP_ChipID);
347
348         PRINTK_2(TRACE_3780I,
349                 "3780i::dsp3780I_EnableDSP exiting bRC=TRUE, ChipID %x\n",
350                 ChipID);
351
352         return 0;
353 }
354
355 int dsp3780I_DisableDSP(DSP_3780I_CONFIG_SETTINGS * pSettings)
356 {
357         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
358         DSP_ISA_SLAVE_CONTROL rSlaveControl;
359
360
```

```
361         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_DisableDSP entry\n");
362
363         rSlaveControl.ClockControl = 0;
364         rSlaveControl.SoftReset = TRUE;
365         rSlaveControl.ConfigMode = FALSE;
366         rSlaveControl.Reserved = 0;
367         spin_lock_irqsave(&dsp_lock, flags);
368         OutWordDsp(DSP_IsaSlaveControl, MKWORD(rSlaveControl));
369
370         udelay(5);
371
372         rSlaveControl.ClockControl = 1;
373         OutWordDsp(DSP_IsaSlaveControl, MKWORD(rSlaveControl));
374         spin_unlock_irqrestore(&dsp_lock, flags);
375
376         udelay(5);
377
378
379         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_DisableDSP exit\n");
380
381         return 0;
382 }
383
384 int dsp3780I_Reset(DSP_3780I_CONFIG_SETTINGS * pSettings)
385 {
386         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
387         DSP_BOOT_DOMAIN rBootDomain;
388         DSP_HBRIDGE_CONTROL rHBridgeControl;
389
390
391         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_Reset entry\n");
392
393         spin_lock_irqsave(&dsp_lock, flags);
394         /* Mask DSP to PC interrupt */
395         MKWORD(rHBridgeControl) = InWordDsp(DSP_HBridgeControl);
396
397         PRINTK_2(TRACE_3780I, "3780i::dsp3780i_Reset rHBridgeControl %x\n",
398                 MKWORD(rHBridgeControl));
399
400         rHBridgeControl.EnableDspInt = FALSE;
401         OutWordDsp(DSP_HBridgeControl, MKWORD(rHBridgeControl));
402         spin_unlock_irqrestore(&dsp_lock, flags);
403
404         /* Reset the core via the boot domain register */
405         rBootDomain.ResetCore = TRUE;
406         rBootDomain.Halt = TRUE;
407         rBootDomain.NMI = TRUE;
408         rBootDomain.Reserved = 0;
409
410         PRINTK_2(TRACE_3780I, "3780i::dsp3780i_Reset rBootDomain %x\n",
411                 MKWORD(rBootDomain));
412
413         WriteMsaCfg(DSP_MspBootDomain, MKWORD(rBootDomain));
414
415         /* Reset all the chiplets and then reactivate them */
416         WriteMsaCfg(DSP_ChipReset, 0xFFFF);
417         udelay(5);
418         WriteMsaCfg(DSP_ChipReset,
419                         (unsigned short) (~pSettings->usChipletEnable));
420
421
422         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_Reset exit bRC=0\n");
423
424         return 0;
425 }
426
427
428 int dsp3780I_Run(DSP_3780I_CONFIG_SETTINGS * pSettings)
429 {
430         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
431         DSP_BOOT_DOMAIN rBootDomain;
432         DSP_HBRIDGE_CONTROL rHBridgeControl;
433
434
435         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_Run entry\n");
436
437
438         /* Transition the core to a running state */
439         rBootDomain.ResetCore = TRUE;
440         rBootDomain.Halt = FALSE;
441         rBootDomain.NMI = TRUE;
442         rBootDomain.Reserved = 0;
443         WriteMsaCfg(DSP_MspBootDomain, MKWORD(rBootDomain));
444
445         udelay(5);
446
447         rBootDomain.ResetCore = FALSE;
448         WriteMsaCfg(DSP_MspBootDomain, MKWORD(rBootDomain));
449         udelay(5);
450
```

```
451         rBootDomain.NMI = FALSE;
452         WriteMsaCfg(DSP_MspBootDomain, MKWORD(rBootDomain));
453         udelay(5);
454
455         /* Enable DSP to PC interrupt */
456         spin_lock_irqsave(&dsp_lock, flags);
457         MKWORD(rHBridgeControl) = InWordDsp(DSP_HBridgeControl);
458         rHBridgeControl.EnableDspInt = TRUE;
459
460         PRINTK_2(TRACE_3780I, "3780i::dsp3780i_Run rHBridgeControl %x\n",
461                 MKWORD(rHBridgeControl));
462
463         OutWordDsp(DSP_HBridgeControl, MKWORD(rHBridgeControl));
464         spin_unlock_irqrestore(&dsp_lock, flags);
465
466
467         PRINTK_1(TRACE_3780I, "3780i::dsp3780i_Run exit bRC=TRUE\n");
468
469         return 0;
470 }
471
472
473 int dsp3780I_ReadDStore(unsigned short usDspBaseIO, void *pvBuffer,
474                         unsigned uCount, unsigned long ulDSPAddr)
475 {
476         unsigned short *pusBuffer = pvBuffer;
477         unsigned short val;
478
479
480         PRINTK_5(TRACE_3780I,
481                 "3780i::dsp3780I_ReadDStore entry usDspBaseIO %x, pusBuffer %p, uCount %x, ulDSPAddr %lx\n",
482                 usDspBaseIO, pusBuffer, uCount, ulDSPAddr);
483
484
485         /* Set the initial MSA address. No adjustments need to be made to data store addresses */
486         spin_lock_irqsave(&dsp_lock, flags);
487         OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulDSPAddr);
488         OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulDSPAddr >> 16));
489         spin_unlock_irqrestore(&dsp_lock, flags);
490
491         /* Transfer the memory block */
492         while (uCount-- != 0) {
493                 spin_lock_irqsave(&dsp_lock, flags);
494                 val = InWordDsp(DSP_MsaDataDSISHigh);
495                 spin_unlock_irqrestore(&dsp_lock, flags);
496                 if(put_user(val, pusBuffer++))
497                         return -EFAULT;
498
499                 PRINTK_3(TRACE_3780I,
500                         "3780I::dsp3780I_ReadDStore uCount %x val %x\n",
501                         uCount, val);
502
503                 PaceMsaAccess(usDspBaseIO);
504         }
505
506
507         PRINTK_1(TRACE_3780I,
508                 "3780I::dsp3780I_ReadDStore exit bRC=TRUE\n");
509
510         return 0;
511 }
512
513 int dsp3780I_ReadAndClearDStore(unsigned short usDspBaseIO,
514                                 void *pvBuffer, unsigned uCount,
515                                 unsigned long ulDSPAddr)
516 {
517         unsigned short *pusBuffer = pvBuffer;
518         unsigned short val;
519
520
521         PRINTK_5(TRACE_3780I,
522                 "3780i::dsp3780I_ReadAndDStore entry usDspBaseIO %x, pusBuffer %p, uCount %x, ulDSPAddr %lx\n",
523                 usDspBaseIO, pusBuffer, uCount, ulDSPAddr);
524
525
526         /* Set the initial MSA address. No adjustments need to be made to data store addresses */
527         spin_lock_irqsave(&dsp_lock, flags);
528         OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulDSPAddr);
529         OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulDSPAddr >> 16));
530         spin_unlock_irqrestore(&dsp_lock, flags);
531
532         /* Transfer the memory block */
533         while (uCount-- != 0) {
534                 spin_lock_irqsave(&dsp_lock, flags);
535                 val = InWordDsp(DSP_ReadAndClear);
536                 spin_unlock_irqrestore(&dsp_lock, flags);
537                 if(put_user(val, pusBuffer++))
538                         return -EFAULT;
539
540                 PRINTK_3(TRACE_3780I,
```

```
                        "3780I::dsp3780I_ReadAndCleanDStore uCount %x val %x\n",
                        uCount, val);

                PaceMsaAccess(usDspBaseIO);
        }


        PRINTK_1(TRACE_3780I,
                "3780I::dsp3780I_ReadAndClearDStore exit bRC=TRUE\n");

        return 0;
}


int dsp3780I_WriteDStore(unsigned short usDspBaseIO, void *pvBuffer,
                         unsigned uCount, unsigned long ulDSPAddr)
{
        unsigned short *pusBuffer = pvBuffer;


        PRINTK_5(TRACE_3780I,
                "3780i::dsp3780D_WriteDStore entry usDspBaseIO %x, pusBuffer %p, uCount %x, ulDSPAddr %lx\n",
                usDspBaseIO, pusBuffer, uCount, ulDSPAddr);


        /* Set the initial MSA address. No adjustments need to be made to data store addresses */
        spin_lock_irqsave(&dsp_lock, flags);
        OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulDSPAddr);
        OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulDSPAddr >> 16));
        spin_unlock_irqrestore(&dsp_lock, flags);

        /* Transfer the memory block */
        while (uCount-- != 0) {
                unsigned short val;
                if(get_user(val, pusBuffer++))
                        return -EFAULT;
                spin_lock_irqsave(&dsp_lock, flags);
                OutWordDsp(DSP_MsaDataDSISHigh, val);
                spin_unlock_irqrestore(&dsp_lock, flags);

                PRINTK_3(TRACE_3780I,
                        "3780I::dsp3780I_WriteDStore uCount %x val %x\n",
                        uCount, val);

                PaceMsaAccess(usDspBaseIO);
        }


        PRINTK_1(TRACE_3780I,
                "3780I::dsp3780D_WriteDStore exit bRC=TRUE\n");

        return 0;
}


int dsp3780I_ReadIStore(unsigned short usDspBaseIO, void *pvBuffer,
                        unsigned uCount, unsigned long ulDSPAddr)
{
        unsigned short *pusBuffer = pvBuffer;

        PRINTK_5(TRACE_3780I,
                "3780i::dsp3780I_ReadIStore entry usDspBaseIO %x, pusBuffer %p, uCount %x, ulDSPAddr %lx\n",
                usDspBaseIO, pusBuffer, uCount, ulDSPAddr);

        /*
        * Set the initial MSA address. To convert from an instruction store
        * address to an MSA address
        * shift the address two bits to the left and set bit 22
        */
        ulDSPAddr = (ulDSPAddr << 2) | (1 << 22);
        spin_lock_irqsave(&dsp_lock, flags);
        OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulDSPAddr);
        OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulDSPAddr >> 16));
        spin_unlock_irqrestore(&dsp_lock, flags);

        /* Transfer the memory block */
        while (uCount-- != 0) {
                unsigned short val_lo, val_hi;
                spin_lock_irqsave(&dsp_lock, flags);
                val_lo = InWordDsp(DSP_MsaDataISLow);
                val_hi = InWordDsp(DSP_MsaDataDSISHigh);
                spin_unlock_irqrestore(&dsp_lock, flags);
                if(put_user(val_lo, pusBuffer++))
                        return -EFAULT;
                if(put_user(val_hi, pusBuffer++))
                        return -EFAULT;

                PRINTK_4(TRACE_3780I,
                        "3780I::dsp3780I_ReadIStore uCount %x val_lo %x val_hi %x\n",
                        uCount, val_lo, val_hi);
```

```
631
632                 PaceMsaAccess(usDspBaseIO);
633
634         }
635
636         PRINTK_1(TRACE_3780I,
637                 "3780I::dsp3780I_ReadIStore exit bRC=TRUE\n");
638
639         return 0;
640 }
641
642
643 int dsp3780I_WriteIStore(unsigned short usDspBaseIO, void *pvBuffer,
644                          unsigned uCount, unsigned long ulDSPAddr)
645 {
646         unsigned short *pusBuffer = pvBuffer;
647
648         PRINTK_5(TRACE_3780I,
649                 "3780i::dsp3780I_WriteIStore entry usDspBaseIO %x, pusBuffer %p, uCount %x, ulDSPAddr %lx\n",
650                 usDspBaseIO, pusBuffer, uCount, ulDSPAddr);
651
652
653         /*
654         * Set the initial MSA address. To convert from an instruction store
655         * address to an MSA address
656         * shift the address two bits to the left and set bit 22
657         */
658         ulDSPAddr = (ulDSPAddr << 2) | (1 << 22);
659         spin_lock_irqsave(&dsp_lock, flags);
660         OutWordDsp(DSP_MsaAddrLow, (unsigned short) ulDSPAddr);
661         OutWordDsp(DSP_MsaAddrHigh, (unsigned short) (ulDSPAddr >> 16));
662         spin_unlock_irqrestore(&dsp_lock, flags);
663
664         /* Transfer the memory block */
665         while (uCount-- != 0) {
666                 unsigned short val_lo, val_hi;
667                 if(get_user(val_lo, pusBuffer++))
668                         return -EFAULT;
669                 if(get_user(val_hi, pusBuffer++))
670                         return -EFAULT;
671                 spin_lock_irqsave(&dsp_lock, flags);
672                 OutWordDsp(DSP_MsaDataISLow, val_lo);
673                 OutWordDsp(DSP_MsaDataDSISHigh, val_hi);
674                 spin_unlock_irqrestore(&dsp_lock, flags);
675
676                 PRINTK_4(TRACE_3780I,
677                         "3780I::dsp3780I_WriteIStore uCount %x val_lo %x val_hi %x\n",
678                         uCount, val_lo, val_hi);
679
680                 PaceMsaAccess(usDspBaseIO);
681
682         }
683
684         PRINTK_1(TRACE_3780I,
685                 "3780I::dsp3780I_WriteIStore exit bRC=TRUE\n");
686
687         return 0;
688 }
689
690
691 int dsp3780I_GetIPCSource(unsigned short usDspBaseIO,
692                           unsigned short *pusIPCSource)
693 {
694         DSP_HBRIDGE_CONTROL rHBridgeControl;
695         unsigned short temp;
696
697
698         PRINTK_3(TRACE_3780I,
699                 "3780i::dsp3780I_GetIPCSource entry usDspBaseIO %x pusIPCSource %p\n",
700                 usDspBaseIO, pusIPCSource);
701
702         /*
703         * Disable DSP to PC interrupts, read the interupt register,
704         * clear the pending IPC bits, and reenable DSP to PC interrupts
705         */
706         spin_lock_irqsave(&dsp_lock, flags);
707         MKWORD(rHBridgeControl) = InWordDsp(DSP_HBridgeControl);
708         rHBridgeControl.EnableDspInt = FALSE;
709         OutWordDsp(DSP_HBridgeControl, MKWORD(rHBridgeControl));
710
711         *pusIPCSource = InWordDsp(DSP_Interrupt);
712         temp = (unsigned short) ~(*pusIPCSource);
713
714         PRINTK_3(TRACE_3780I,
715                 "3780i::dsp3780I_GetIPCSource, usIPCSource %x ~%x\n ",
716                 *pusIPCSource, temp);
717
718         OutWordDsp(DSP_Interrupt, (unsigned short) ~(*pusIPCSource));
719
720         rHBridgeControl.EnableDspInt = TRUE;
```

```
721             OutWordDsp(DSP_HBridgeControl, MKWORD(rHBridgeControl));
722             spin_unlock_irqrestore(&dsp_lock, flags);
723
724
725             PRINTK_2(TRACE_3780I,
726                     "3780i::dsp3780I_GetIPCSource exit usIPCSource %x\n",
727                     *pusIPCSource);
728
729             return 0;
730     }
```

```
/*
*
* mwavedd.c -- mwave device driver
*
*
* Written By: Mike Sullivan IBM Corporation
*
* Copyright (C) 1999 IBM Corporation
*
* 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.
*
* NO WARRANTY
* THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT
* LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT,
* MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is
* solely responsible for determining the appropriateness of using and
* distributing the Program and assumes all risks associated with its
* exercise of rights under this Agreement, including but not limited to
* the risks and costs of program errors, damage to or loss of data,
* programs or equipment, and unavailability or interruption of operations.
*
* DISCLAIMER OF LIABILITY
* NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
* TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED
* HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES
*
* 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
*
*
* 10/23/2000 - Alpha Release
*       First release to the public
*/

#include <linux/version.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/major.h>
#include <linux/miscdevice.h>
#include <linux/proc_fs.h>
#include <linux/serial.h>
#include <linux/sched.h>
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
#include <linux/spinlock.h>
#else
#include <asm/spinlock.h>
#endif
#include <linux/delay.h>
#include "smapi.h"
#include "mwavedd.h"
#include "3780i.h"
#include "tp3780i.h"

#ifndef __exit
#define __exit
#endif

MODULE_DESCRIPTION("3780i Advanced Communications Processor (Mwave) driver");
MODULE_AUTHOR("Mike Sullivan and Paul Schroeder");
MODULE_LICENSE("GPL");

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
static int mwave_get_info(char *buf, char **start, off_t offset, int len);
#else
static int mwave_read_proc(char *buf, char **start, off_t offset, int xlen, int unused);
static struct proc_dir_entry mwave_proc = {
        0,                      /* unsigned short low_ino */
        5,                      /* unsigned short namelen */
        "mwave",                /* const char *name */
        S_IFREG | S_IRUGO,      /* mode_t mode */
        1,                      /* nlink_t nlink */
        0,                      /* uid_t uid */
        0,                      /* gid_t gid */
        0,                      /* unsigned long size */
```

```
91          NULL,                   /* struct inode_operations *ops */
92          &mwave_read_proc        /* int (*get_info) (...) */
93  };
94  #endif
95
96  /*
97  * These parameters support the setting of MWave resources. Note that no
98  * checks are made against other devices (ie. superio) for conflicts.
99  * We'll depend on users using the tpctl utility to do that for now
100 */
101 int mwave_debug = 0;
102 int mwave_3780i_irq = 0;
103 int mwave_3780i_io = 0;
104 int mwave_uart_irq = 0;
105 int mwave_uart_io = 0;
106 MODULE_PARM(mwave_debug, "i");
107 MODULE_PARM(mwave_3780i_irq, "i");
108 MODULE_PARM(mwave_3780i_io, "i");
109 MODULE_PARM(mwave_uart_irq, "i");
110 MODULE_PARM(mwave_uart_io, "i");
111
112 static int mwave_open(struct inode *inode, struct file *file);
113 static int mwave_close(struct inode *inode, struct file *file);
114 static int mwave_ioctl(struct inode *inode, struct file *filp,
115                        unsigned int iocmd, unsigned long ioarg);
116
117 MWAVE_DEVICE_DATA mwave_s_mdd;
118
119 static int mwave_open(struct inode *inode, struct file *file)
120 {
121         unsigned int retval = 0;
122
123         PRINTK_3(TRACE_MWAVE,
124                 "mwavedd::mwave_open, entry inode %x file %x\n",
125                 (int) inode, (int) file);
126         PRINTK_2(TRACE_MWAVE,
127                 "mwavedd::mwave_open, exit return retval %x\n", retval);
128
129         MOD_INC_USE_COUNT;
130         return retval;
131 }
132
133 static int mwave_close(struct inode *inode, struct file *file)
134 {
135         unsigned int retval = 0;
136
137         PRINTK_3(TRACE_MWAVE,
138                 "mwavedd::mwave_close, entry inode %x file %x\n",
139                 (int) inode, (int) file);
140
141         PRINTK_2(TRACE_MWAVE, "mwavedd::mwave_close, exit retval %x\n",
142                 retval);
143
144         MOD_DEC_USE_COUNT;
145         return retval;
146 }
147
148 static int mwave_ioctl(struct inode *inode, struct file *file,
149                        unsigned int iocmd, unsigned long ioarg)
150 {
151         unsigned int retval = 0;
152         pMWAVE_DEVICE_DATA pDrvData = &mwave_s_mdd;
153
154         PRINTK_5(TRACE_MWAVE,
155                 "mwavedd::mwave_ioctl, entry inode %x file %x cmd %x arg %x\n",
156                 (int) inode, (int) file, iocmd, (int) ioarg);
157
158         switch (iocmd) {
159
160                 case IOCTL_MW_RESET:
161                         PRINTK_1(TRACE_MWAVE,
162                                 "mwavedd::mwave_ioctl, IOCTL_MW_RESET calling tp3780I_ResetDSP\n");
163                         retval = tp3780I_ResetDSP(&pDrvData->rBDData);
164                         PRINTK_2(TRACE_MWAVE,
165                                 "mwavedd::mwave_ioctl, IOCTL_MW_RESET retval %x from tp3780I_ResetDSP\n",
166                                 retval);
167                         break;
168
169                 case IOCTL_MW_RUN:
170                         PRINTK_1(TRACE_MWAVE,
171                                 "mwavedd::mwave_ioctl, IOCTL_MW_RUN calling tp3780I_StartDSP\n");
172                         retval = tp3780I_StartDSP(&pDrvData->rBDData);
173                         PRINTK_2(TRACE_MWAVE,
174                                 "mwavedd::mwave_ioctl, IOCTL_MW_RUN retval %x from tp3780I_StartDSP\n",
175                                 retval);
176                         break;
177
178                 case IOCTL_MW_DSP_ABILITIES: {
179                         MW_ABILITIES rAbilities;
180
```

```
181                     PRINTK_1(TRACE_MWAVE,
182                             "mwavedd::mwave_ioctl, IOCTL_MW_DSP_ABILITIES calling tp3780I_QueryAbilities\n");
183                     retval = tp3780I_QueryAbilities(&pDrvData->rBDData, &rAbilities);
184                     PRINTK_2(TRACE_MWAVE,
185                             "mwavedd::mwave_ioctl, IOCTL_MW_DSP_ABILITIES retval %x from tp3780I_QueryAbilities\n",
186                             retval);
187                     if (retval == 0) {
188                             if( copy_to_user((char *) ioarg, (char *) &rAbilities, sizeof(MW_ABILITIES)) )
189                                     return -EFAULT;
190                     }
191                     PRINTK_2(TRACE_MWAVE,
192                             "mwavedd::mwave_ioctl, IOCTL_MW_DSP_ABILITIES exit retval %x\n",
193                             retval);
194             }
195                     break;
196
197             case IOCTL_MW_READ_DATA:
198             case IOCTL_MW_READCLEAR_DATA: {
199                     MW_READWRITE rReadData;
200                     unsigned short *pusBuffer = 0;
201
202                     if( copy_from_user((char *) &rReadData, (char *) ioarg, sizeof(MW_READWRITE)) )
203                             return -EFAULT;
204                     pusBuffer = (unsigned short *) (rReadData.pBuf);
205
206                     PRINTK_4(TRACE_MWAVE,
207                             "mwavedd::mwave_ioctl IOCTL_MW_READ_DATA, size %lx, ioarg %lx pusBuffer %p\n",
208                             rReadData.ulDataLength, ioarg, pusBuffer);
209                     retval = tp3780I_ReadWriteDspDStore(&pDrvData->rBDData, iocmd,
210                             (void *) pusBuffer, rReadData.ulDataLength, rReadData.usDspAddress);
211             }
212                     break;
213
214             case IOCTL_MW_READ_INST: {
215                     MW_READWRITE rReadData;
216                     unsigned short *pusBuffer = 0;
217
218                     if( copy_from_user((char *) &rReadData, (char *) ioarg, sizeof(MW_READWRITE)) )
219                             return -EFAULT;
220                     pusBuffer = (unsigned short *) (rReadData.pBuf);
221
222                     PRINTK_4(TRACE_MWAVE,
223                             "mwavedd::mwave_ioctl IOCTL_MW_READ_INST, size %lx, ioarg %lx pusBuffer %p\n",
224                             rReadData.ulDataLength / 2, ioarg,
225                             pusBuffer);
226                     retval = tp3780I_ReadWriteDspDStore(&pDrvData->rBDData,
227                             iocmd, pusBuffer,
228                             rReadData.ulDataLength / 2,
229                             rReadData.usDspAddress);
230             }
231                     break;
232
233             case IOCTL_MW_WRITE_DATA: {
234                     MW_READWRITE rWriteData;
235                     unsigned short *pusBuffer = 0;
236
237                     if( copy_from_user((char *) &rWriteData, (char *) ioarg, sizeof(MW_READWRITE)) )
238                             return -EFAULT;
239                     pusBuffer = (unsigned short *) (rWriteData.pBuf);
240
241                     PRINTK_4(TRACE_MWAVE,
242                             "mwavedd::mwave_ioctl IOCTL_MW_WRITE_DATA, size %lx, ioarg %lx pusBuffer %p\n",
243                             rWriteData.ulDataLength, ioarg,
244                             pusBuffer);
245                     retval = tp3780I_ReadWriteDspDStore(&pDrvData->rBDData, iocmd,
246                             pusBuffer, rWriteData.ulDataLength, rWriteData.usDspAddress);
247             }
248                     break;
249
250             case IOCTL_MW_WRITE_INST: {
251                     MW_READWRITE rWriteData;
252                     unsigned short *pusBuffer = 0;
253
254                     if( copy_from_user((char *) &rWriteData, (char *) ioarg, sizeof(MW_READWRITE)) )
255                             return -EFAULT;
256                     pusBuffer = (unsigned short *) (rWriteData.pBuf);
257
258                     PRINTK_4(TRACE_MWAVE,
259                             "mwavedd::mwave_ioctl IOCTL_MW_WRITE_INST, size %lx, ioarg %lx pusBuffer %p\n",
260                             rWriteData.ulDataLength, ioarg,
261                             pusBuffer);
262                     retval = tp3780I_ReadWriteDspIStore(&pDrvData->rBDData, iocmd,
263                                     pusBuffer, rWriteData.ulDataLength, rWriteData.usDspAddress);
264             }
265                     break;
266
267             case IOCTL_MW_REGISTER_IPC: {
268                     unsigned int ipcnum = (unsigned int) ioarg;
269
270                     PRINTK_3(TRACE_MWAVE,
```

```
271                                     "mwavedd::mwave_ioctl IOCTL_MW_REGISTER_IPC ipcnum %x entry usIntCount %x\n",
272                                     ipcnum,
273                                     pDrvData->IPCs[ipcnum].usIntCount);
274
275                             if (ipcnum > 16) {
276                                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_ioctl: IOCTL_MW_REGISTER_IPC: Error: Invalid ipcnum %x\n", ipcnum);
277                                     return -EINVAL;
278                             }
279                             pDrvData->IPCs[ipcnum].bIsHere = FALSE;
280                             pDrvData->IPCs[ipcnum].bIsEnabled = TRUE;
281     #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
282     #else
283                             current->priority = 0x28;       /* boost to provide priority timing */
284     #endif
285
286                             PRINTK_2(TRACE_MWAVE,
287                                     "mwavedd::mwave_ioctl IOCTL_MW_REGISTER_IPC ipcnum %x exit\n",
288                                     ipcnum);
289                     }
290                             break;
291
292                     case IOCTL_MW_GET_IPC: {
293                             unsigned int ipcnum = (unsigned int) ioarg;
294                             spinlock_t ipc_lock = SPIN_LOCK_UNLOCKED;
295                             unsigned long flags;
296
297                             PRINTK_3(TRACE_MWAVE,
298                                     "mwavedd::mwave_ioctl IOCTL_MW_GET_IPC ipcnum %x, usIntCount %x\n",
299                                     ipcnum,
300                                     pDrvData->IPCs[ipcnum].usIntCount);
301                             if (ipcnum > 16) {
302                                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_ioctl: IOCTL_MW_GET_IPC: Error: Invalid ipcnum %x\n", ipcnum);
303                                     return -EINVAL;
304                             }
305
306                             if (pDrvData->IPCs[ipcnum].bIsEnabled == TRUE) {
307                                     PRINTK_2(TRACE_MWAVE,
308                                             "mwavedd::mwave_ioctl, thread for ipc %x going to sleep\n",
309                                             ipcnum);
310
311                                     spin_lock_irqsave(&ipc_lock, flags);
312                                     /* check whether an event was signalled by */
313                                     /* the interrupt handler while we were gone */
314                                     if (pDrvData->IPCs[ipcnum].usIntCount == 1) {   /* first int has occurred (race condition) */
315                                             pDrvData->IPCs[ipcnum].usIntCount = 2;  /* first int has been handled */
316                                             spin_unlock_irqrestore(&ipc_lock, flags);
317                                             PRINTK_2(TRACE_MWAVE,
318                                                     "mwavedd::mwave_ioctl IOCTL_MW_GET_IPC ipcnum %x handling first int\n",
319                                                     ipcnum);
320                                     } else {        /* either 1st int has not yet occurred, or we have already handled the first int */
321                                             pDrvData->IPCs[ipcnum].bIsHere = TRUE;
322                                             interruptible_sleep_on(&pDrvData->IPCs[ipcnum].ipc_wait_queue);
323                                             pDrvData->IPCs[ipcnum].bIsHere = FALSE;
324                                             if (pDrvData->IPCs[ipcnum].usIntCount == 1) {
325                                                     pDrvData->IPCs[ipcnum].
326                                                     usIntCount = 2;
327                                             }
328                                             spin_unlock_irqrestore(&ipc_lock, flags);
329                                             PRINTK_2(TRACE_MWAVE,
330                                                     "mwavedd::mwave_ioctl IOCTL_MW_GET_IPC ipcnum %x woke up and returning to application\n",
331                                                     ipcnum);
332                                     }
333                                     PRINTK_2(TRACE_MWAVE,
334                                             "mwavedd::mwave_ioctl IOCTL_MW_GET_IPC, returning thread for ipc %x processing\n",
335                                             ipcnum);
336                             }
337                     }
338                             break;
339
340                     case IOCTL_MW_UNREGISTER_IPC: {
341                             unsigned int ipcnum = (unsigned int) ioarg;
342
343                             PRINTK_2(TRACE_MWAVE,
344                                     "mwavedd::mwave_ioctl IOCTL_MW_UNREGISTER_IPC ipcnum %x\n",
345                                     ipcnum);
346                             if (ipcnum > 16) {
347                                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_ioctl: IOCTL_MW_UNREGISTER_IPC: Error: Invalid ipcnum %x\n", ipcnum);
348                                     return -EINVAL;
349                             }
350                             if (pDrvData->IPCs[ipcnum].bIsEnabled == TRUE) {
351                                     pDrvData->IPCs[ipcnum].bIsEnabled = FALSE;
352                                     if (pDrvData->IPCs[ipcnum].bIsHere == TRUE) {
353                                             wake_up_interruptible(&pDrvData->IPCs[ipcnum].ipc_wait_queue);
354                                     }
```

```
355                         }
356                 }
357                         break;
358
359                 default:
360                         PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_ioctl: Error: Unrecognized iocmd %x\n", iocmd);
361                         return -ENOTTY;
362                         break;
363         } /* switch */
364
365         PRINTK_2(TRACE_MWAVE, "mwavedd::mwave_ioctl, exit retval %x\n", retval);
366
367         return retval;
368 }
369
370
371 static ssize_t mwave_read(struct file *file, char *buf, size_t count,
372                           loff_t * ppos)
373 {
374         PRINTK_5(TRACE_MWAVE,
375                 "mwavedd::mwave_read entry file %p, buf %p, count %x ppos %p\n",
376                 file, buf, count, ppos);
377
378         return -EINVAL;
379 }
380
381
382 static ssize_t mwave_write(struct file *file, const char *buf,
383                            size_t count, loff_t * ppos)
384 {
385         PRINTK_5(TRACE_MWAVE,
386                 "mwavedd::mwave_write entry file %p, buf %p, count %x ppos %p\n",
387                 file, buf, count, ppos);
388
389         return -EINVAL;
390 }
391
392
393 static int register_serial_portandirq(unsigned int port, int irq)
394 {
395         struct serial_struct serial;
396
397         switch ( port ) {
398                 case 0x3f8:
399                 case 0x2f8:
400                 case 0x3e8:
401                 case 0x2e8:
402                         /* OK */
403                         break;
404                 default:
405                         PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::register_serial_portandirq: Error: Illegal port %x\n", port );
406                         return -1;
407         } /* switch */
408         /* port is okay */
409
410         switch ( irq ) {
411                 case 3:
412                 case 4:
413                 case 5:
414                 case 7:
415                         /* OK */
416                         break;
417                 default:
418                         PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::register_serial_portandirq: Error: Illegal irq %x\n", irq );
419                         return -1;
420         } /* switch */
421         /* irq is okay */
422
423         memset(&serial, 0, sizeof(serial));
424         serial.port = port;
425         serial.irq = irq;
426         serial.flags = ASYNC_SHARE_IRQ;
427
428         return register_serial(&serial);
429 }
430
431
432 #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
433 static struct file_operations mwave_fops = {
434         owner:THIS_MODULE,
435         read:mwave_read,
436         write:mwave_write,
437         ioctl:mwave_ioctl,
438         open:mwave_open,
439         release:mwave_close
440 };
441 #else
442 static struct file_operations mwave_fops = {
443         NULL,                   /* lseek */
444         mwave_read,             /* read */
```

```
445         mwave_write,            /* write */
446         NULL,                   /* readdir */
447         NULL,                   /* poll */
448         mwave_ioctl,            /* ioctl */
449         NULL,                   /* mmap */
450         mwave_open,             /* open */
451         NULL,                   /* flush */
452         mwave_close             /* release */
453 };
454 #endif
455
456 static struct miscdevice mwave_misc_dev = { MWAVE_MINOR, "mwave", &mwave_fops };
457
458 /*
459 * mwave_init is called on module load
460 *
461 * mwave_exit is called on module unload
462 * mwave_exit is also used to clean up after an aborted mwave_init
463 */
464 static void mwave_exit(void)
465 {
466         pMWAVE_DEVICE_DATA pDrvData = &mwave_s_mdd;
467
468         PRINTK_1(TRACE_MWAVE, "mwavedd::mwave_exit entry\n");
469
470         if (pDrvData->bProcEntryCreated) {
471 #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
472                 remove_proc_entry("mwave", NULL);
473 #else
474                 proc_unregister(&proc_root, mwave_proc.low_ino);
475 #endif
476         }
477         if ( pDrvData->sLine >= 0 ) {
478                 unregister_serial(pDrvData->sLine);
479         }
480         if (pDrvData->bMwaveDevRegistered) {
481                 misc_deregister(&mwave_misc_dev);
482         }
483         if (pDrvData->bDSPEnabled) {
484                 tp3780I_DisableDSP(&pDrvData->rBDData);
485         }
486         if (pDrvData->bResourcesClaimed) {
487                 tp3780I_ReleaseResources(&pDrvData->rBDData);
488         }
489         if (pDrvData->bBDInitialized) {
490                 tp3780I_Cleanup(&pDrvData->rBDData);
491         }
492
493         PRINTK_1(TRACE_MWAVE, "mwavedd::mwave_exit exit\n");
494 }
495
496 module_exit(mwave_exit);
497
498 static int __init mwave_init(void)
499 {
500         int i;
501         int retval = 0;
502         unsigned int resultMiscRegister;
503         pMWAVE_DEVICE_DATA pDrvData = &mwave_s_mdd;
504
505         memset(&mwave_s_mdd, 0, sizeof(MWAVE_DEVICE_DATA));
506
507         PRINTK_1(TRACE_MWAVE, "mwavedd::mwave_init entry\n");
508
509         pDrvData->bBDInitialized = FALSE;
510         pDrvData->bResourcesClaimed = FALSE;
511         pDrvData->bDSPEnabled = FALSE;
512         pDrvData->bDSPReset = FALSE;
513         pDrvData->bMwaveDevRegistered = FALSE;
514         pDrvData->sLine = -1;
515         pDrvData->bProcEntryCreated = FALSE;
516
517         for (i = 0; i < 16; i++) {
518                 pDrvData->IPCs[i].bIsEnabled = FALSE;
519                 pDrvData->IPCs[i].bIsHere = FALSE;
520                 pDrvData->IPCs[i].usIntCount = 0;       /* no ints received yet */
521 #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
522                 init_waitqueue_head(&pDrvData->IPCs[i].ipc_wait_queue);
523 #endif
524         }
525
526         retval = tp3780I_InitializeBoardData(&pDrvData->rBDData);
527         PRINTK_2(TRACE_MWAVE,
528                 "mwavedd::mwave_init, return from tp3780I_InitializeBoardData retval %x\n",
529                 retval);
530         if (retval) {
531                 PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_init: Error: Failed to initialize board data\n");
532                 goto cleanup_error;
533         }
534         pDrvData->bBDInitialized = TRUE;
```

```
535
536             retval = tp3780I_CalcResources(&pDrvData->rBDData);
537             PRINTK_2(TRACE_MWAVE,
538                     "mwavedd::mwave_init, return from tp3780I_CalcResources retval %x\n",
539                     retval);
540             if (retval) {
541                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd:mwave_init: Error: Failed to calculate resources\n");
542                     goto cleanup_error;
543             }
544
545             retval = tp3780I_ClaimResources(&pDrvData->rBDData);
546             PRINTK_2(TRACE_MWAVE,
547                     "mwavedd::mwave_init, return from tp3780I_ClaimResources retval %x\n",
548                     retval);
549             if (retval) {
550                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd:mwave_init: Error: Failed to claim resources\n");
551                     goto cleanup_error;
552             }
553             pDrvData->bResourcesClaimed = TRUE;
554
555             retval = tp3780I_EnableDSP(&pDrvData->rBDData);
556             PRINTK_2(TRACE_MWAVE,
557                     "mwavedd::mwave_init, return from tp3780I_EnableDSP retval %x\n",
558                     retval);
559             if (retval) {
560                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd:mwave_init: Error: Failed to enable DSP\n");
561                     goto cleanup_error;
562             }
563             pDrvData->bDSPEnabled = TRUE;
564
565             resultMiscRegister = misc_register(&mwave_misc_dev);
566             if (resultMiscRegister < 0) {
567                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd:mwave_init: Error: Failed to register misc device\n");
568                     goto cleanup_error;
569             }
570             pDrvData->bMwaveDevRegistered = TRUE;
571
572             pDrvData->sLine = register_serial_portandirq(
573                     pDrvData->rBDData.rDspSettings.usUartBaseIO,
574                     pDrvData->rBDData.rDspSettings.usUartIrq
575             );
576             if (pDrvData->sLine < 0) {
577                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd:mwave_init: Error: Failed to register serial driver\n");
578                     goto cleanup_error;
579             }
580             /* uart is registered */
581
582             if (
583     #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
584                     !create_proc_info_entry("mwave", 0, NULL, mwave_get_info)
585     #else
586                     proc_register(&proc_root, &mwave_proc)
587     #endif
588             ) {
589                     PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_init: Error: Failed to register /proc/mwave\n");
590                     goto cleanup_error;
591             }
592             pDrvData->bProcEntryCreated = TRUE;
593
594             /* SUCCESS! */
595             return 0;
596
597             cleanup_error:
598             PRINTK_ERROR(KERN_ERR_MWAVE "mwavedd::mwave_init: Error: Failed to initialize\n");
599             mwave_exit(); /* clean up */
600
601             return -EIO;
602     }
603
604     module_init(mwave_init);
605
606
607     /*
608     * proc entry stuff added by Ian Pilcher <pilcher@us.ibm.com>
609     */
610
611     #if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
612     static int mwave_get_info(char *buf, char **start, off_t offset, int len)
613     {
614             DSP_3780I_CONFIG_SETTINGS *pSettings = &mwave_s_mdd.rBDData.rDspSettings;
615
616             char *out = buf;
617
618             out += sprintf(out, "3780i_IRQ %i\n", pSettings->usDspIrq);
619             out += sprintf(out, "3780i_DMA %i\n", pSettings->usDspDma);
620             out += sprintf(out, "3780i_IO  %#.4x\n", pSettings->usDspBaseIO);
621             out += sprintf(out, "UART_IRQ %i\n", pSettings->usUartIrq);
622             out += sprintf(out, "UART_IO  %#.4x\n", pSettings->usUartBaseIO);
623
624             return out - buf;
```

```
625 }
626 #else /* kernel version < 2.4.0 */
627 static int mwave_read_proc(char *buf, char **start, off_t offset,
628                            int xlen, int unused)
629 {
630         DSP_3780I_CONFIG_SETTINGS *pSettings = &mwave_s_mdd.rBDData.rDspSettings;
631         int len;
632
633         len = sprintf(buf,        "3780i_IRQ %i\n", pSettings->usDspIrq);
634         len += sprintf(&buf[len], "3780i_DMA %i\n", pSettings->usDspDma);
635         len += sprintf(&buf[len], "3780i_IO  %#.4x\n", pSettings->usDspBaseIO);
636         len += sprintf(&buf[len], "UART_IRQ  %i\n", pSettings->usUartIrq);
637         len += sprintf(&buf[len], "UART_IO   %#.4x\n", pSettings->usUartBaseIO);
638
639         return len;
640 }
641 #endif
```

```
/*
*
* smapi.c -- SMAPI interface routines
*
*
* Written By: Mike Sullivan IBM Corporation
*
* Copyright (C) 1999 IBM Corporation
*
* 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.
*
* NO WARRANTY
* THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT
* LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT,
* MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is
* solely responsible for determining the appropriateness of using and
* distributing the Program and assumes all risks associated with its
* exercise of rights under this Agreement, including but not limited to
* the risks and costs of program errors, damage to or loss of data,
* programs or equipment, and unavailability or interruption of operations.
*
* DISCLAIMER OF LIABILITY
* NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
* TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED
* HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES
*
* 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
*
*
* 10/23/2000 - Alpha Release
*       First release to the public
*/

#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/mc146818rtc.h>  /* CMOS defines */
#include "smapi.h"
#include "mwavedd.h"

static unsigned short g_usSmapiPort = 0;


int smapi_request(unsigned short inBX, unsigned short inCX,
                  unsigned short inDI, unsigned short inSI,
                  unsigned short *outAX, unsigned short *outBX,
                  unsigned short *outCX, unsigned short *outDX,
                  unsigned short *outDI, unsigned short *outSI)
{
        unsigned short myoutAX = 2, *pmyoutAX = &myoutAX;
        unsigned short myoutBX = 3, *pmyoutBX = &myoutBX;
        unsigned short myoutCX = 4, *pmyoutCX = &myoutCX;
        unsigned short myoutDX = 5, *pmyoutDX = &myoutDX;
        unsigned short myoutDI = 6, *pmyoutDI = &myoutDI;
        unsigned short myoutSI = 7, *pmyoutSI = &myoutSI;
        unsigned short usSmapiOK = -EIO, *pusSmapiOK = &usSmapiOK;
        unsigned int inBXCX = (inBX << 16) | inCX;
        unsigned int inDISI = (inDI << 16) | inSI;
        int retval = 0;

        PRINTK_5(TRACE_SMAPI, "inBX %x inCX %x inDI %x inSI %x\n",
                inBX, inCX, inDI, inSI);

        __asm__ __volatile__("movw  $0x5380,%%ax\n\t"
                             "movl  %7,%%ebx\n\t"
                             "shrl  $16, %%ebx\n\t"
                             "movw  %7,%%cx\n\t"
                             "movl  %8,%%edi\n\t"
                             "shrl  $16,%%edi\n\t"
                             "movw  %8,%%si\n\t"
                             "movw  %9,%%dx\n\t"
                             "out   %%al,%%dx\n\t"
                             "out   %%al,$0x4F\n\t"
                             "cmpb  $0x53,%%ah\n\t"
                             "je    2f\n\t"
                             "1:\n\t"
```

```
                                "orb  %%ah,%%ah\n\t"
                                "jnz  2f\n\t"
                                "movw  %%ax,%0\n\t"
                                "movw  %%bx,%1\n\t"
                                "movw  %%cx,%2\n\t"
                                "movw  %%dx,%3\n\t"
                                "movw  %%di,%4\n\t"
                                "movw  %%si,%5\n\t"
                                "movw  $1,%6\n\t"
                                "2:\n\t":"=m"(*(unsigned short *) pmyoutAX),
                                "=m"(*(unsigned short *) pmyoutBX),
                                "=m"(*(unsigned short *) pmyoutCX),
                                "=m"(*(unsigned short *) pmyoutDX),
                                "=m"(*(unsigned short *) pmyoutDI),
                                "=m"(*(unsigned short *) pmyoutSI),
                                "=m"(*(unsigned short *) pusSmapiOK)
                                :"m"(inBXCX), "m"(inDISI), "m"(g_usSmapiPort)
                                :"%eax", "%ebx", "%ecx", "%edx", "%edi",
                                "%esi");

        PRINTK_8(TRACE_SMAPI,
                "myoutAX %x myoutBX %x myoutCX %x myoutDX %x myoutDI %x myoutSI %x usSmapiOK %x\n",
                myoutAX, myoutBX, myoutCX, myoutDX, myoutDI, myoutSI,
                usSmapiOK);
        *outAX = myoutAX;
        *outBX = myoutBX;
        *outCX = myoutCX;
        *outDX = myoutDX;
        *outDI = myoutDI;
        *outSI = myoutSI;

        retval = (usSmapiOK == 1) ? 0 : -EIO;
        PRINTK_2(TRACE_SMAPI, "smapi::smapi_request exit retval %x\n", retval);
        return retval;
}


int smapi_query_DSP_cfg(SMAPI_DSP_SETTINGS * pSettings)
{
        int bRC = -EIO;
        unsigned short usAX, usBX, usCX, usDX, usDI, usSI;
        unsigned short ausDspBases[] = { 0x0030, 0x4E30, 0x8E30, 0xCE30, 0x0130, 0x0350, 0x0070, 0x0DB0 };
        unsigned short ausUartBases[] = { 0x03F8, 0x02F8, 0x03E8, 0x02E8 };
        unsigned short numDspBases = 8;
        unsigned short numUartBases = 4;

        PRINTK_1(TRACE_SMAPI, "smapi::smapi_query_DSP_cfg entry\n");

        bRC = smapi_request(0x1802, 0x0000, 0, 0,
                &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
        if (bRC) {
                PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_query_DSP_cfg: Error: Could not get DSP Settings. Aborting.\n");
                return bRC;
        }

        PRINTK_1(TRACE_SMAPI, "smapi::smapi_query_DSP_cfg, smapi_request OK\n");

        pSettings->bDSPPresent = ((usBX & 0x0100) != 0);
        pSettings->bDSPEnabled = ((usCX & 0x0001) != 0);
        pSettings->usDspIRQ = usSI & 0x00FF;
        pSettings->usDspDMA = (usSI & 0xFF00) >> 8;
        if ((usDI & 0x00FF) < numDspBases) {
                pSettings->usDspBaseIO = ausDspBases[usDI & 0x00FF];
        } else {
                pSettings->usDspBaseIO = 0;
        }
        PRINTK_6(TRACE_SMAPI,
                "smapi::smapi_query_DSP_cfg get DSP Settings bDSPPresent %x bDSPEnabled %x usDspIRQ %x usDspDMA %x usDspBaseIO %x\n",
                pSettings->bDSPPresent, pSettings->bDSPEnabled,
                pSettings->usDspIRQ, pSettings->usDspDMA,
                pSettings->usDspBaseIO);

        /* check for illegal values */
        if ( pSettings->usDspBaseIO == 0 )
                PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_query_DSP_cfg: Worry: DSP base I/O address is 0\n");
        if ( pSettings->usDspIRQ == 0 )
                PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_query_DSP_cfg: Worry: DSP IRQ line is 0\n");

        bRC = smapi_request(0x1804, 0x0000, 0, 0,
                &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
        if (bRC) {
                PRINTK_ERROR("smapi::smapi_query_DSP_cfg: Error: Could not get DSP modem settings. Aborting.\n");
                return bRC;
        }

        PRINTK_1(TRACE_SMAPI, "smapi::smapi_query_DSP_cfg, smapi_request OK\n");

        pSettings->bModemEnabled = ((usCX & 0x0001) != 0);
        pSettings->usUartIRQ = usSI & 0x000F;
        if (((usSI & 0xFF00) >> 8) < numUartBases) {
```

```
181                     pSettings->usUartBaseIO = ausUartBases[(usSI & 0xFF00) >> 8];
182             } else {
183                     pSettings->usUartBaseIO = 0;
184             }
185
186             PRINTK_4(TRACE_SMAPI,
187                     "smapi::smapi_query_DSP_cfg get DSP modem settings bModemEnabled %x usUartIRQ %x usUartBaseIO %x\n",
188                     pSettings->bModemEnabled,
189                     pSettings->usUartIRQ,
190                     pSettings->usUartBaseIO);
191
192             /* check for illegal values */
193             if ( pSettings->usUartBaseIO == 0 )
194                     PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_query_DSP_cfg: Worry: UART base I/O address is 0\n");
195             if ( pSettings->usUartIRQ == 0 )
196                     PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_query_DSP_cfg: Worry: UART IRQ line is 0\n");
197
198             PRINTK_2(TRACE_SMAPI, "smapi::smapi_query_DSP_cfg exit bRC %x\n", bRC);
199
200             return bRC;
201     }
202
203
204     int smapi_set_DSP_cfg(void)
205     {
206             int bRC = -EIO;
207             int i;
208             unsigned short usAX, usBX, usCX, usDX, usDI, usSI;
209             unsigned short ausDspBases[] = { 0x0030, 0x4E30, 0x8E30, 0xCE30, 0x0130, 0x0350, 0x0070, 0x0DB0 };
210             unsigned short ausUartBases[] = { 0x03F8, 0x02F8, 0x03E8, 0x02E8 };
211             unsigned short ausDspIrqs[] = { 5, 7, 10, 11, 15 };
212             unsigned short ausUartIrqs[] = { 3, 4 };
213
214             unsigned short numDspBases = 8;
215             unsigned short numUartBases = 4;
216             unsigned short numDspIrqs = 5;
217             unsigned short numUartIrqs = 2;
218             unsigned short dspio_index = 0, uartio_index = 0;
219
220             PRINTK_5(TRACE_SMAPI,
221                     "smapi::smapi_set_DSP_cfg entry mwave_3780i_irq %x mwave_3780i_io %x mwave_uart_irq %x mwave_uart_io %x\n",
222                     mwave_3780i_irq, mwave_3780i_io, mwave_uart_irq, mwave_uart_io);
223
224             if (mwave_3780i_io) {
225                     for (i = 0; i < numDspBases; i++) {
226                             if (mwave_3780i_io == ausDspBases[i])
227                                     break;
228                     }
229                     if (i == numDspBases) {
230                             PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_set_DSP_cfg: Error: Invalid mwave_3780i_io address %x. Aborting.\n",
     mwave_3780i_io);
231                             return bRC;
232                     }
233                     dspio_index = i;
234             }
235
236             if (mwave_3780i_irq) {
237                     for (i = 0; i < numDspIrqs; i++) {
238                             if (mwave_3780i_irq == ausDspIrqs[i])
239                                     break;
240                     }
241                     if (i == numDspIrqs) {
242                             PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_set_DSP_cfg: Error: Invalid mwave_3780i_irq %x. Aborting.\n", mwav
    e_3780i_irq);
243                             return bRC;
244                     }
245             }
246
247             if (mwave_uart_io) {
248                     for (i = 0; i < numUartBases; i++) {
249                             if (mwave_uart_io == ausUartBases[i])
250                                     break;
251                     }
252                     if (i == numUartBases) {
253                             PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_set_DSP_cfg: Error: Invalid mwave_uart_io address %x. Aborting.\n", m
    wave_uart_io);
254                             return bRC;
255                     }
256                     uartio_index = i;
257             }
258
259
260             if (mwave_uart_irq) {
261                     for (i = 0; i < numUartIrqs; i++) {
262                             if (mwave_uart_irq == ausUartIrqs[i])
263                                     break;
264                     }
265                     if (i == numUartIrqs) {
266                             PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_set_DSP_cfg: Error: Invalid mwave_uart_irq %x. Aborting.\n", mwave_
    uart_irq);
```

```
267                         return bRC;
268                 }
269         }
270
271         if (mwave_uart_irq || mwave_uart_io) {
272
273                 /* Check serial port A */
274                 bRC = smapi_request(0x1402, 0x0000, 0, 0,
275                         &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
276                 if (bRC) goto exit_smapi_request_error;
277                 /* bRC == 0 */
278                 if (usBX & 0x0100) {    /* serial port A is present */
279                         if (usCX & 1) { /* serial port is enabled */
280                                 if ((usSI & 0xFF) == mwave_uart_irq) {
281 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
282                                         PRINTK_ERROR(KERN_ERR_MWAVE
283 #else
284                                         PRINTK_3(TRACE_SMAPI,
285 #endif
286                                                 "smapi::smapi_set_DSP_cfg: Serial port A irq %x conflicts with mwave_uart_irq %x\n", usS
    I, mwave_uart_irq);
287 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
288                                         PRINTK_1(TRACE_SMAPI,
289                                                 "smapi::smapi_set_DSP_cfg Disabling conflicting serial port\n");
290                                         bRC = smapi_request(0x1403, 0x0100, 0, usSI,
291                                                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
292                                         if (bRC) goto exit_smapi_request_error;
293                                         bRC = smapi_request(0x1402, 0x0000, 0, 0,
294                                                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
295                                         if (bRC) goto exit_smapi_request_error;
296 #else
297                                         goto exit_conflict;
298 #endif
299                                 } else {
300                                         if ((usSI >> 8) == uartio_index) {
301 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
302                                                 PRINTK_ERROR(KERN_ERR_MWAVE
303 #else
304                                                 PRINTK_3(TRACE_SMAPI,
305 #endif
306                                                         "smapi::smapi_set_DSP_cfg: Serial port A base I/O address index %x conflicts with
    uartio_index %x\n", usSI >> 8, uartio_index);
307 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
308                                                 PRINTK_1(TRACE_SMAPI,
309                                                         "smapi::smapi_set_DSP_cfg Disabling conflicting serial port\n");
310                                                 bRC = smapi_request (0x1403, 0x0100, 0, usSI,
311                                                         &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
312                                                 if (bRC) goto exit_smapi_request_error;
313                                                 bRC = smapi_request (0x1402, 0x0000, 0, 0,
314                                                         &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
315                                                 if (bRC) goto exit_smapi_request_error;
316 #else
317                                                 goto exit_conflict;
318 #endif
319                                         }
320                                 }
321                         }
322                 }
323
324                 /* Check serial port B */
325                 bRC = smapi_request(0x1404, 0x0000, 0, 0,
326                         &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
327                 if (bRC) goto exit_smapi_request_error;
328                 /* bRC == 0 */
329                 if (usBX & 0x0100) {    /* serial port B is present */
330                         if (usCX & 1) { /* serial port is enabled */
331                                 if ((usSI & 0xFF) == mwave_uart_irq) {
332 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
333                                         PRINTK_ERROR(KERN_ERR_MWAVE
334 #else
335                                         PRINTK_3(TRACE_SMAPI,
336 #endif
337                                                 "smapi::smapi_set_DSP_cfg: Serial port B irq %x conflicts with mwave_uart_irq %x\n", usS
    I, mwave_uart_irq);
338 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
339                                         PRINTK_1(TRACE_SMAPI,
340                                                 "smapi::smapi_set_DSP_cfg Disabling conflicting serial port\n");
341                                         bRC = smapi_request(0x1405, 0x0100, 0, usSI,
342                                                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
343                                         if (bRC) goto exit_smapi_request_error;
344                                         bRC = smapi_request(0x1404, 0x0000, 0, 0,
345                                                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
346                                         if (bRC) goto exit_smapi_request_error;
347 #else
348                                         goto exit_conflict;
349 #endif
350                                 } else {
351                                         if ((usSI >> 8) == uartio_index) {
352 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
353                                                 PRINTK_ERROR(KERN_ERR_MWAVE
```

```
354 #else
355                                                     PRINTK_3(TRACE_SMAPI,
356 #endif
357                                                              "smapi::smapi_set_DSP_cfg: Serial port B base I/O address index %x conflicts with
    uartio_index %x\n", usSI >> 8, uartio_index);
358 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
359                                                     PRINTK_1 (TRACE_SMAPI,
360                                                         "smapi::smapi_set_DSP_cfg Disabling conflicting serial port\n");
361                                                     bRC = smapi_request (0x1405, 0x0100, 0, usSI,
362                                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
363                                                     if (bRC) goto exit_smapi_request_error;
364                                                     bRC = smapi_request (0x1404, 0x0000, 0, 0,
365                                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
366                                                     if (bRC) goto exit_smapi_request_error;
367 #else
368                                                     goto exit_conflict;
369 #endif
370                                             }
371                                     }
372                             }
373                     }
374
375                     /* Check IR port */
376                     bRC = smapi_request(0x1700, 0x0000, 0, 0,
377                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
378                     if (bRC) goto exit_smapi_request_error;
379                     bRC = smapi_request(0x1704, 0x0000, 0, 0,
380                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
381                     if (bRC) goto exit_smapi_request_error;
382                     /* bRC == 0 */
383                     if ((usCX & 0xff) == mwave_uart_irq) {  /* serial port is enabled */
384 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
385                             PRINTK_ERROR(KERN_ERR_MWAVE
386 #else
387                             PRINTK_3(TRACE_SMAPI,
388 #endif
389                                     "smapi::smapi_set_DSP_cfg: IR port irq %x conflicts with mwave_uart_irq %x\n", usSI, mwave_uart_irq);
390 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
391                             PRINTK_1(TRACE_SMAPI,
392                                     "smapi::smapi_set_DSP_cfg Disabling conflicting IR port\n");
393                             bRC = smapi_request(0x1701, 0x0100, 0, 0,
394                                     &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
395                             if (bRC) goto exit_smapi_request_error;
396                             bRC = smapi_request(0x1700, 0, 0, 0,
397                                     &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
398                             if (bRC) goto exit_smapi_request_error;
399                             bRC = smapi_request(0x1705, 0x01ff, 0, usSI,
400                                     &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
401                             if (bRC) goto exit_smapi_request_error;
402                             bRC = smapi_request(0x1704, 0x0000, 0, 0,
403                                     &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
404                             if (bRC) goto exit_smapi_request_error;
405 #else
406                             goto exit_conflict;
407 #endif
408                     } else {
409                             if ((usSI & 0xff) == uartio_index) {
410 #ifndef MWAVE_FUTZ_WITH_OTHER_DEVICES
411                                     PRINTK_ERROR(KERN_ERR_MWAVE
412 #else
413                                     PRINTK_3(TRACE_SMAPI,
414 #endif
415                                             "smapi::smapi_set_DSP_cfg: IR port base I/O address index %x conflicts with uartio_index %x\n", usSI
     & 0xff, uartio_index);
416 #ifdef MWAVE_FUTZ_WITH_OTHER_DEVICES
417                                     PRINTK_1(TRACE_SMAPI,
418                                             "smapi::smapi_set_DSP_cfg Disabling conflicting IR port\n");
419                                     bRC = smapi_request(0x1701, 0x0100, 0, 0,
420                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
421                                     if (bRC) goto exit_smapi_request_error;
422                                     bRC = smapi_request(0x1700, 0, 0, 0,
423                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
424                                     if (bRC) goto exit_smapi_request_error;
425                                     bRC = smapi_request(0x1705, 0x01ff, 0, usSI,
426                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
427                                     if (bRC) goto exit_smapi_request_error;
428                                     bRC = smapi_request(0x1704, 0x0000, 0, 0,
429                                             &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
430                                     if (bRC) goto exit_smapi_request_error;
431 #else
432                                     goto exit_conflict;
433 #endif
434                             }
435                     }
436             }
437
438             bRC = smapi_request(0x1802, 0x0000, 0, 0,
439                     &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
440             if (bRC) goto exit_smapi_request_error;
441
```

```
442         if (mwave_3780i_io) {
443                 usDI = dspio_index;;
444         }
445         if (mwave_3780i_irq) {
446                 usSI = (usSI & 0xff00) | mwave_3780i_irq;
447         }
448
449         bRC = smapi_request(0x1803, 0x0101, usDI, usSI,
450                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
451         if (bRC) goto exit_smapi_request_error;
452
453         bRC = smapi_request(0x1804, 0x0000, 0, 0,
454                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
455         if (bRC) goto exit_smapi_request_error;
456
457         if (mwave_uart_io) {
458                 usSI = (usSI & 0x00ff) | (uartio_index << 8);
459         }
460         if (mwave_uart_irq) {
461                 usSI = (usSI & 0xff00) | mwave_uart_irq;
462         }
463         bRC = smapi_request(0x1805, 0x0101, 0, usSI,
464                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
465         if (bRC) goto exit_smapi_request_error;
466
467         bRC = smapi_request(0x1802, 0x0000, 0, 0,
468                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
469         if (bRC) goto exit_smapi_request_error;
470
471         bRC = smapi_request(0x1804, 0x0000, 0, 0,
472                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
473         if (bRC) goto exit_smapi_request_error;
474
475 /* normal exit: */
476         PRINTK_1(TRACE_SMAPI, "smapi::smapi_set_DSP_cfg exit\n");
477         return 0;
478
479 exit_conflict:
480         /* Message has already been printed */
481         return -EIO;
482
483 exit_smapi_request_error:
484         PRINTK_ERROR(KERN_ERR_MWAVE "smapi::smapi_set_DSP_cfg exit on smapi_request error bRC %x\n", bRC);
485         return bRC;
486 }
487
488
489 int smapi_set_DSP_power_state(BOOLEAN bOn)
490 {
491         int bRC = -EIO;
492         unsigned short usAX, usBX, usCX, usDX, usDI, usSI;
493         unsigned short usPowerFunction;
494
495         PRINTK_2(TRACE_SMAPI, "smapi::smapi_set_DSP_power_state entry bOn %x\n", bOn);
496
497         usPowerFunction = (bOn) ? 1 : 0;
498
499         bRC = smapi_request(0x4901, 0x0000, 0, usPowerFunction,
500                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
501
502         PRINTK_2(TRACE_SMAPI, "smapi::smapi_set_DSP_power_state exit bRC %x\n", bRC);
503
504         return bRC;
505 }
506
507
508 int SmapiQuerySystemID(void)
509 {
510         int bRC = -EIO;
511         unsigned short usAX = 0xffff, usBX = 0xffff, usCX = 0xffff,
512                 usDX = 0xffff, usDI = 0xffff, usSI = 0xffff;
513
514         printk("smapi::SmapiQUerySystemID entry\n");
515         bRC = smapi_request(0x0000, 0, 0, 0,
516                 &usAX, &usBX, &usCX, &usDX, &usDI, &usSI);
517
518         if (bRC == 0) {
519                 printk("AX=%x, BX=%x, CX=%x, DX=%x, DI=%x, SI=%x\n",
520                         usAX, usBX, usCX, usDX, usDI, usSI);
521         } else {
522                 printk("smapi::SmapiQuerySystemID smapi_request error\n");
523         }
524
525         return bRC;
526 }
527
528
529 int smapi_init(void)
530 {
531         int retval = -EIO;
```

```
532         unsigned short usSmapiID = 0;
533         unsigned long flags;
534
535         PRINTK_1(TRACE_SMAPI, "smapi::smapi_init entry\n");
536
537         spin_lock_irqsave(&rtc_lock, flags);
538         usSmapiID = CMOS_READ(0x7C);
539         usSmapiID |= (CMOS_READ(0x7D) << 8);
540         spin_unlock_irqrestore(&rtc_lock, flags);
541         PRINTK_2(TRACE_SMAPI, "smapi::smapi_init usSmapiID %x\n", usSmapiID);
542
543         if (usSmapiID == 0x5349) {
544                 spin_lock_irqsave(&rtc_lock, flags);
545                 g_usSmapiPort = CMOS_READ(0x7E);
546                 g_usSmapiPort |= (CMOS_READ(0x7F) << 8);
547                 spin_unlock_irqrestore(&rtc_lock, flags);
548                 if (g_usSmapiPort == 0) {
549                         PRINTK_ERROR("smapi::smapi_init, ERROR unable to read from SMAPI port\n");
550                 } else {
551                         PRINTK_2(TRACE_SMAPI,
552                                 "smapi::smapi_init, exit TRUE g_usSmapiPort %x\n",
553                                 g_usSmapiPort);
554                         retval = 0;
555                         //SmapiQuerySystemID();
556                 }
557         } else {
558                 PRINTK_ERROR("smapi::smapi_init, ERROR invalid usSmapiID\n");
559                 retval = -ENXIO;
560         }
561
562         return retval;
563 }
```

```
/*
*
* tp3780i.c −− board driver for 3780i on ThinkPads
*
*
* Written By: Mike Sullivan IBM Corporation
*
* Copyright (C) 1999 IBM Corporation
*
* 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.
*
* NO WARRANTY
* THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT
* LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON−INFRINGEMENT,
* MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is
* solely responsible for determining the appropriateness of using and
* distributing the Program and assumes all risks associated with its
* exercise of rights under this Agreement, including but not limited to
* the risks and costs of program errors, damage to or loss of data,
* programs or equipment, and unavailability or interruption of operations.
*
* DISCLAIMER OF LIABILITY
* NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
* TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
* USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED
* HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES
*
* 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
*
*
* 10/23/2000 − Alpha Release
*       First release to the public
*/

#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/ptrace.h>
#include <linux/ioport.h>
#include <asm/io.h>
#include "smapi.h"
#include "mwavedd.h"
#include "tp3780i.h"
#include "3780i.h"
#include "mwavepub.h"

extern MWAVE_DEVICE_DATA mwave_s_mdd;

static unsigned short s_ausThinkpadIrqToField[16] =
        { 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0x0002, 0x0003, 0xFFFF, 0x0004,
        0xFFFF, 0xFFFF, 0x0005, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007 };
static unsigned short s_ausThinkpadDmaToField[8] =
        { 0x0001, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0x0004 };
static unsigned short s_numIrqs = 16, s_numDmas = 8;


static void EnableSRAM(THINKPAD_BD_DATA * pBDData)
{
        DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData−>rDspSettings;
        unsigned short usDspBaseIO = pSettings−>usDspBaseIO;
        DSP_GPIO_OUTPUT_DATA_15_8 rGpioOutputData;
        DSP_GPIO_DRIVER_ENABLE_15_8 rGpioDriverEnable;
        DSP_GPIO_MODE_15_8 rGpioMode;

        PRINTK_1(TRACE_TP3780I, "tp3780i::EnableSRAM, entry\n");

        MKWORD(rGpioMode) = ReadMsaCfg(DSP_GpioModeControl_15_8);
        rGpioMode.GpioMode10 = 0;
        WriteMsaCfg(DSP_GpioModeControl_15_8, MKWORD(rGpioMode));

        MKWORD(rGpioDriverEnable) = 0;
        rGpioDriverEnable.Enable10 = TRUE;
        rGpioDriverEnable.Mask10 = TRUE;
        WriteMsaCfg(DSP_GpioDriverEnable_15_8, MKWORD(rGpioDriverEnable));

        MKWORD(rGpioOutputData) = 0;
        rGpioOutputData.Latch10 = 0;
```

```
91          rGpioOutputData.Mask10 = TRUE;
92          WriteMsaCfg(DSP_GpioOutputData_15_8, MKWORD(rGpioOutputData));
93
94          PRINTK_1(TRACE_TP3780I, "tp3780i::EnableSRAM exit\n");
95  }
96
97
98  static void UartInterrupt(int irq, void *dev_id, struct pt_regs *regs)
99  {
100         PRINTK_3(TRACE_TP3780I,
101                 "tp3780i::UartInterrupt entry irq %x dev_id %x\n", irq, (int) dev_id);
102 }
103
104 static void DspInterrupt(int irq, void *dev_id, struct pt_regs *regs)
105 {
106         pMWAVE_DEVICE_DATA pDrvData = &mwave_s_mdd;
107         DSP_3780I_CONFIG_SETTINGS *pSettings = &pDrvData->rBDData.rDspSettings;
108         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
109         unsigned short usIPCSource = 0, usIsolationMask, usPCNum;
110
111         PRINTK_3(TRACE_TP3780I,
112                 "tp3780i::DspInterrupt entry irq %x dev_id %x\n", irq, (int) dev_id);
113
114         if (dsp3780I_GetIPCSource(usDspBaseIO, &usIPCSource) == 0) {
115                 PRINTK_2(TRACE_TP3780I,
116                         "tp3780i::DspInterrupt, return from dsp3780i_GetIPCSource, usIPCSource %x\n",
117                         usIPCSource);
118                 usIsolationMask = 1;
119                 for (usPCNum = 1; usPCNum <= 16; usPCNum++) {
120                         if (usIPCSource & usIsolationMask) {
121                                 usIPCSource &= ~usIsolationMask;
122                                 PRINTK_3(TRACE_TP3780I,
123                                         "tp3780i::DspInterrupt usPCNum %x usIPCSource %x\n",
124                                         usPCNum, usIPCSource);
125                                 if (pDrvData->IPCs[usPCNum - 1].usIntCount == 0) {
126                                         pDrvData->IPCs[usPCNum - 1].usIntCount = 1;
127                                 }
128                                 PRINTK_2(TRACE_TP3780I,
129                                         "tp3780i::DspInterrupt usIntCount %x\n",
130                                         pDrvData->IPCs[usPCNum - 1].usIntCount);
131                                 if (pDrvData->IPCs[usPCNum - 1].bIsEnabled == TRUE) {
132                                         PRINTK_2(TRACE_TP3780I,
133                                                 "tp3780i::DspInterrupt, waking up usPCNum %x\n",
134                                                 usPCNum - 1);
135                                         wake_up_interruptible(&pDrvData->IPCs[usPCNum - 1].ipc_wait_queue);
136                                 } else {
137                                         PRINTK_2(TRACE_TP3780I,
138                                                 "tp3780i::DspInterrupt, no one waiting for IPC %x\n",
139                                                 usPCNum - 1);
140                                 }
141                         }
142                         if (usIPCSource == 0)
143                                 break;
144                         /* try next IPC */
145                         usIsolationMask = usIsolationMask << 1;
146                 }
147         } else {
148                 PRINTK_1(TRACE_TP3780I,
149                         "tp3780i::DspInterrupt, return false from dsp3780i_GetIPCSource\n");
150         }
151         PRINTK_1(TRACE_TP3780I, "tp3780i::DspInterrupt exit\n");
152 }
153
154
155 int tp3780I_InitializeBoardData(THINKPAD_BD_DATA * pBDData)
156 {
157         int retval = 0;
158         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
159
160
161         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_InitializeBoardData entry pBDData %p\n", pBDData);
162
163         pBDData->bDSPEnabled = FALSE;
164         pSettings->bInterruptClaimed = FALSE;
165
166         retval = smapi_init();
167         if (retval) {
168                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_InitializeBoardData: Error: SMAPI is not available on this machine\n");
169         } else {
170                 if (mwave_3780i_irq || mwave_3780i_io || mwave_uart_irq || mwave_uart_io) {
171                         retval = smapi_set_DSP_cfg();
172                 }
173         }
174
175         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_InitializeBoardData exit retval %x\n", retval);
176
177         return retval;
178 }
179
180 int tp3780I_Cleanup(THINKPAD_BD_DATA * pBDData)
```

```
{
        int retval = 0;

        PRINTK_2(TRACE_TP3780I,
                "tp3780i::tp3780I_Cleanup entry and exit pBDData %p\n", pBDData);

        return retval;
}

int tp3780I_CalcResources(THINKPAD_BD_DATA * pBDData)
{
        SMAPI_DSP_SETTINGS rSmapiInfo;
        DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;

        PRINTK_2(TRACE_TP3780I,
                "tp3780i::tp3780I_CalcResources entry pBDData %p\n", pBDData);

        if (smapi_query_DSP_cfg(&rSmapiInfo)) {
                PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_CalcResources: Error: Could not query DSP config. Aborting.\n");
                return -EIO;
        }

        /* Sanity check */
        if (
                ( rSmapiInfo.usDspIRQ == 0 )
                || ( rSmapiInfo.usDspBaseIO ==  0 )
                || ( rSmapiInfo.usUartIRQ ==  0 )
                || ( rSmapiInfo.usUartBaseIO ==  0 )
        ) {
                PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_CalcResources: Error: Illegal resource setting. Aborting.\n");
                return -EIO;
        }

        pSettings->bDSPEnabled = (rSmapiInfo.bDSPEnabled && rSmapiInfo.bDSPPresent);
        pSettings->bModemEnabled = rSmapiInfo.bModemEnabled;
        pSettings->usDspIrq = rSmapiInfo.usDspIRQ;
        pSettings->usDspDma = rSmapiInfo.usDspDMA;
        pSettings->usDspBaseIO = rSmapiInfo.usDspBaseIO;
        pSettings->usUartIrq = rSmapiInfo.usUartIRQ;
        pSettings->usUartBaseIO = rSmapiInfo.usUartBaseIO;

        pSettings->uDStoreSize = TP_ABILITIES_DATA_SIZE;
        pSettings->uIStoreSize = TP_ABILITIES_INST_SIZE;
        pSettings->uIps = TP_ABILITIES_INTS_PER_SEC;

        if (pSettings->bDSPEnabled && pSettings->bModemEnabled && pSettings->usDspIrq == pSettings->usUartIrq) {
                pBDData->bShareDspIrq = pBDData->bShareUartIrq = 1;
        } else {
                pBDData->bShareDspIrq = pBDData->bShareUartIrq = 0;
        }

        PRINTK_1(TRACE_TP3780I, "tp3780i::tp3780I_CalcResources exit\n");

        return 0;
}


int tp3780I_ClaimResources(THINKPAD_BD_DATA * pBDData)
{
        int retval = 0;
        DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
        struct resource *pres;
#endif

        PRINTK_2(TRACE_TP3780I,
                "tp3780i::tp3780I_ClaimResources entry pBDData %p\n", pBDData);

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
        pres = request_region(pSettings->usDspBaseIO, 16, "mwave_3780i");
        if ( pres == NULL ) retval = -EIO;
#else
        retval = check_region(pSettings->usDspBaseIO, 16);
        if (!retval) request_region(pSettings->usDspBaseIO, 16, "mwave_3780i");
#endif
        if (retval) {
                PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_ClaimResources: Error: Could not claim I/O region starting at %x\n", pSettings->usDspBaseIO);
                retval = -EIO;
        }

        PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_ClaimResources exit retval %x\n", retval);

        return retval;
}

int tp3780I_ReleaseResources(THINKPAD_BD_DATA * pBDData)
{
        int retval = 0;
        DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
```

```
270
271         PRINTK_2(TRACE_TP3780I,
272                 "tp3780i::tp3780I_ReleaseResources entry pBDData %p\n", pBDData);
273
274         release_region(pSettings->usDspBaseIO & (~3), 16);
275
276         if (pSettings->bInterruptClaimed) {
277                 free_irq(pSettings->usDspIrq, 0);
278                 pSettings->bInterruptClaimed = FALSE;
279         }
280
281         PRINTK_2(TRACE_TP3780I,
282                 "tp3780i::tp3780I_ReleaseResources exit retval %x\n", retval);
283
284         return retval;
285 }
286
287
288
289 int tp3780I_EnableDSP(THINKPAD_BD_DATA * pBDData)
290 {
291         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
292         BOOLEAN bDSPPoweredUp = FALSE, bDSPEnabled = FALSE, bInterruptAllocated = FALSE;
293
294         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_EnableDSP entry pBDData %p\n", pBDData);
295
296         if (pBDData->bDSPEnabled) {
297                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: DSP already enabled!\n");
298                 goto exit_cleanup;
299         }
300
301         if (!pSettings->bDSPEnabled) {
302                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780::tp3780I_EnableDSP: Error: pSettings->bDSPEnabled not set\n");
303                 goto exit_cleanup;
304         }
305
306         if (
307                 (pSettings->usDspIrq >= s_numIrqs)
308                 || (pSettings->usDspDma >= s_numDmas)
309                 || (s_ausThinkpadIrqToField[pSettings->usDspIrq] == 0xFFFF)
310                 || (s_ausThinkpadDmaToField[pSettings->usDspDma] == 0xFFFF)
311         ) {
312                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: invalid irq %x\n", pSettings->usDspIrq);
313                 goto exit_cleanup;
314         }
315
316         if (
317                 ((pSettings->usDspBaseIO & 0xF00F) != 0)
318                 || (pSettings->usDspBaseIO & 0x0FF0) == 0
319         ) {
320                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: Invalid DSP base I/O address %x\n", pSettings->usDsp
    BaseIO);
321                 goto exit_cleanup;
322         }
323
324         if (pSettings->bModemEnabled) {
325                 if (
326                         pSettings->usUartIrq >= s_numIrqs
327                         || s_ausThinkpadIrqToField[pSettings->usUartIrq] == 0xFFFF
328                 ) {
329                         PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: Invalid UART IRQ %x\n", pSettings->usUa
    rtIrq);
330                         goto exit_cleanup;
331                 }
332                 switch (pSettings->usUartBaseIO) {
333                         case 0x03F8:
334                         case 0x02F8:
335                         case 0x03E8:
336                         case 0x02E8:
337                                 break;
338
339                         default:
340                                 PRINTK_ERROR("tp3780i::tp3780I_EnableDSP: Error: Invalid UART base I/O address %x\n", pSettings->us
    UartBaseIO);
341                                 goto exit_cleanup;
342                 }
343         }
344
345         pSettings->bDspIrqActiveLow = pSettings->bDspIrqPulse = TRUE;
346         pSettings->bUartIrqActiveLow = pSettings->bUartIrqPulse = TRUE;
347
348         if (pBDData->bShareDspIrq) {
349                 pSettings->bDspIrqActiveLow = FALSE;
350         }
351         if (pBDData->bShareUartIrq) {
352                 pSettings->bUartIrqActiveLow = FALSE;
353         }
354
355         pSettings->usNumTransfers = TP_CFG_NumTransfers;
356         pSettings->usReRequest = TP_CFG_RerequestTimer;
```

```
357         pSettings->bEnableMEMCS16 = TP_CFG_MEMCS16;
358         pSettings->usIsaMemCmdWidth = TP_CFG_IsaMemCmdWidth;
359         pSettings->bGateIOCHRDY = TP_CFG_GateIOCHRDY;
360         pSettings->bEnablePwrMgmt = TP_CFG_EnablePwrMgmt;
361         pSettings->usHBusTimerLoadValue = TP_CFG_HBusTimerValue;
362         pSettings->bDisableLBusTimeout = TP_CFG_DisableLBusTimeout;
363         pSettings->usN_Divisor = TP_CFG_N_Divisor;
364         pSettings->usM_Multiplier = TP_CFG_M_Multiplier;
365         pSettings->bPllBypass = TP_CFG_PllBypass;
366         pSettings->usChipletEnable = TP_CFG_ChipletEnable;
367
368         if (request_irq(pSettings->usUartIrq, &UartInterrupt, 0, "mwave_uart", 0)) {
369                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: Could not get UART IRQ %x\n", pSettings->usUartI
    rq);
370                 goto exit_cleanup;
371         } else {                /* no conflict just release */
372                 free_irq(pSettings->usUartIrq, 0);
373         }
374
375         if (request_irq(pSettings->usDspIrq, &DspInterrupt, 0, "mwave_3780i", 0)) {
376                 PRINTK_ERROR("tp3780i::tp3780I_EnableDSP: Error: Could not get 3780i IRQ %x\n", pSettings->usDspIrq);
377                 goto exit_cleanup;
378         } else {
379                 PRINTK_3(TRACE_TP3780I,
380                         "tp3780i::tp3780I_EnableDSP, got interrupt %x bShareDspIrq %x\n",
381                         pSettings->usDspIrq, pBDData->bShareDspIrq);
382                 bInterruptAllocated = TRUE;
383                 pSettings->bInterruptClaimed = TRUE;
384         }
385
386         smapi_set_DSP_power_state(FALSE);
387         if (smapi_set_DSP_power_state(TRUE)) {
388                 PRINTK_ERROR(KERN_ERR_MWAVE "tp3780i::tp3780I_EnableDSP: Error: smapi_set_DSP_power_state(TRUE) failed\n");
389                 goto exit_cleanup;
390         } else {
391                 bDSPPoweredUp = TRUE;
392         }
393
394         if (dsp3780I_EnableDSP(pSettings, s_ausThinkpadIrqToField, s_ausThinkpadDmaToField)) {
395                 PRINTK_ERROR("tp3780i::tp3780I_EnableDSP: Error: dsp7880I_EnableDSP() failed\n");
396                 goto exit_cleanup;
397         } else {
398                 bDSPEnabled = TRUE;
399         }
400
401         EnableSRAM(pBDData);
402
403         pBDData->bDSPEnabled = TRUE;
404
405         PRINTK_1(TRACE_TP3780I, "tp3780i::tp3780I_EnableDSP exit\n");
406
407         return 0;
408
409 exit_cleanup:
410         PRINTK_ERROR("tp3780i::tp3780I_EnableDSP: Cleaning up\n");
411         if (bDSPEnabled)
412                 dsp3780I_DisableDSP(pSettings);
413         if (bDSPPoweredUp)
414                 smapi_set_DSP_power_state(FALSE);
415         if (bInterruptAllocated) {
416                 free_irq(pSettings->usDspIrq, 0);
417                 pSettings->bInterruptClaimed = FALSE;
418         }
419         return -EIO;
420 }
421
422
423 int tp3780I_DisableDSP(THINKPAD_BD_DATA * pBDData)
424 {
425         int retval = 0;
426         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
427
428         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_DisableDSP entry pBDData %p\n", pBDData);
429
430         if (pBDData->bDSPEnabled) {
431                 dsp3780I_DisableDSP(&pBDData->rDspSettings);
432                 if (pSettings->bInterruptClaimed) {
433                         free_irq(pSettings->usDspIrq, 0);
434                         pSettings->bInterruptClaimed = FALSE;
435                 }
436                 smapi_set_DSP_power_state(FALSE);
437                 pBDData->bDSPEnabled = FALSE;
438         }
439
440         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_DisableDSP exit retval %x\n", retval);
441
442         return retval;
443 }
444
445
```

```
446 int tp3780I_ResetDSP(THINKPAD_BD_DATA * pBDData)
447 {
448         int retval = 0;
449         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
450
451         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_ResetDSP entry pBDData %p\n",
452                 pBDData);
453
454         if (dsp3780I_Reset(pSettings) == 0) {
455                 EnableSRAM(pBDData);
456         } else {
457                 retval = -EIO;
458         }
459
460         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_ResetDSP exit retval %x\n", retval);
461
462         return retval;
463 }
464
465
466 int tp3780I_StartDSP(THINKPAD_BD_DATA * pBDData)
467 {
468         int retval = 0;
469         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
470
471         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_StartDSP entry pBDData %p\n", pBDData);
472
473         if (dsp3780I_Run(pSettings) == 0) {
474                 // @BUG @TBD EnableSRAM(pBDData);
475         } else {
476                 retval = -EIO;
477         }
478
479         PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_StartDSP exit retval %x\n", retval);
480
481         return retval;
482 }
483
484
485 int tp3780I_QueryAbilities(THINKPAD_BD_DATA * pBDData, MW_ABILITIES * pAbilities)
486 {
487         int retval = 0;
488
489         PRINTK_2(TRACE_TP3780I,
490                 "tp3780i::tp3780I_QueryAbilities entry pBDData %p\n", pBDData);
491
492         /* fill out standard constant fields */
493         pAbilities->instr_per_sec = pBDData->rDspSettings.uIps;
494         pAbilities->data_size = pBDData->rDspSettings.uDStoreSize;
495         pAbilities->inst_size = pBDData->rDspSettings.uIStoreSize;
496         pAbilities->bus_dma_bw = pBDData->rDspSettings.uDmaBandwidth;
497
498         /* fill out dynamically determined fields */
499         pAbilities->component_list[0] = 0x00010000 | MW_ADC_MASK;
500         pAbilities->component_list[1] = 0x00010000 | MW_ACI_MASK;
501         pAbilities->component_list[2] = 0x00010000 | MW_AIC1_MASK;
502         pAbilities->component_list[3] = 0x00010000 | MW_AIC2_MASK;
503         pAbilities->component_list[4] = 0x00010000 | MW_CDDAC_MASK;
504         pAbilities->component_list[5] = 0x00010000 | MW_MIDI_MASK;
505         pAbilities->component_list[6] = 0x00010000 | MW_UART_MASK;
506         pAbilities->component_count = 7;
507
508         /* Fill out Mwave OS and BIOS task names */
509
510         memcpy(pAbilities->mwave_os_name, TP_ABILITIES_MWAVEOS_NAME,
511                 sizeof(TP_ABILITIES_MWAVEOS_NAME));
512         memcpy(pAbilities->bios_task_name, TP_ABILITIES_BIOSTASK_NAME,
513                 sizeof(TP_ABILITIES_BIOSTASK_NAME));
514
515         PRINTK_1(TRACE_TP3780I,
516                 "tp3780i::tp3780I_QueryAbilities exit retval=SUCCESSFUL\n");
517
518         return retval;
519 }
520
521 int tp3780I_ReadWriteDspDStore(THINKPAD_BD_DATA * pBDData, unsigned int uOpcode,
522                                void *pvBuffer, unsigned int uCount,
523                                unsigned long ulDSPAddr)
524 {
525         int retval = 0;
526         DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
527         unsigned short usDspBaseIO = pSettings->usDspBaseIO;
528         BOOLEAN bRC = 0;
529
530         PRINTK_6(TRACE_TP3780I,
531                 "tp3780i::tp3780I_ReadWriteDspDStore entry pBDData %p, uOpcode %x, pvBuffer %p, uCount %x, ulDSPAddr %lx\n",
532                 pBDData, uOpcode, pvBuffer, uCount, ulDSPAddr);
533
534         if (pBDData->bDSPEnabled) {
535                 switch (uOpcode) {
```

```
536                     case IOCTL_MW_READ_DATA:
537                             bRC = dsp3780I_ReadDStore(usDspBaseIO, pvBuffer, uCount, ulDSPAddr);
538                             break;
539
540                     case IOCTL_MW_READCLEAR_DATA:
541                             bRC = dsp3780I_ReadAndClearDStore(usDspBaseIO, pvBuffer, uCount, ulDSPAddr);
542                             break;
543
544                     case IOCTL_MW_WRITE_DATA:
545                             bRC = dsp3780I_WriteDStore(usDspBaseIO, pvBuffer, uCount, ulDSPAddr);
546                             break;
547                     }
548             }
549
550             retval = (bRC) ? -EIO : 0;
551             PRINTK_2(TRACE_TP3780I, "tp3780i::tp3780I_ReadWriteDspDStore exit retval %x\n", retval);
552
553             return retval;
554     }
555
556
557     int tp3780I_ReadWriteDspIStore(THINKPAD_BD_DATA * pBDData, unsigned int uOpcode,
558                                    void *pvBuffer, unsigned int uCount,
559                                    unsigned long ulDSPAddr)
560     {
561             int retval = 0;
562             DSP_3780I_CONFIG_SETTINGS *pSettings = &pBDData->rDspSettings;
563             unsigned short usDspBaseIO = pSettings->usDspBaseIO;
564             BOOLEAN bRC = 0;
565
566             PRINTK_6(TRACE_TP3780I,
567                     "tp3780i::tp3780I_ReadWriteDspIStore entry pBDData %p, uOpcode %x, pvBuffer %p, uCount %x, ulDSPAddr %lx\n",
568                     pBDData, uOpcode, pvBuffer, uCount, ulDSPAddr);
569
570             if (pBDData->bDSPEnabled) {
571                     switch (uOpcode) {
572                     case IOCTL_MW_READ_INST:
573                             bRC = dsp3780I_ReadIStore(usDspBaseIO, pvBuffer, uCount, ulDSPAddr);
574                             break;
575
576                     case IOCTL_MW_WRITE_INST:
577                             bRC = dsp3780I_WriteIStore(usDspBaseIO, pvBuffer, uCount, ulDSPAddr);
578                             break;
579                     }
580             }
581
582             retval = (bRC) ? -EIO : 0;
583
584             PRINTK_2(TRACE_TP3780I,
585                     "tp3780i::tp3780I_ReadWriteDspIStore exit retval %x\n", retval);
586
587             return retval;
588     }
589
```

# Table of Content