

```

*****
80637 Sat Feb 7 19:01:07 2015
new/usr/src/lib/libdisasm/common/dis_arm.c
libdisasm: disassembly of strex may cause SIGSEGV
To make things confusing enough, the names of bitfields between ldrex and
strex are not totally consistent so this commit makes it easier on future
readers.
*****
_____unchanged_portion_omitted_____

1272 /*
1273  * Handle LDREX and STREX out of the extra loads/stores extensions.
1274  */
1275 static int
1276 arm_dis_lsexcl(uint32_t in, char *buf, size_t buflen)
1277 {
1278     arm_cond_code_t cc;
1279     arm_reg_t rx, ry, rz;
1280     arm_reg_t rn, rd, rm;
1281     int lbit;
1282     size_t len;

1283     /*
1284      * To make things confusing enough, the names of bitfields between
1285      * ldrex and strex are not totally consistent. Specifically,
1286      * STREX rd, rt, [rn]
1287      *     rn = 19:16
1288      *     rd = 15:12
1289      *     rt = 3:0
1290      *
1291      * LDREX rt, [rn]
1292      *     rn = 19:16
1293      *     rt = 15:12
1294      *
1295      * To avoid having to do too many mental gymnastics, let's just
1296      * think of the bitfields as:
1297      *
1298      *     rx = 19:16
1299      *     ry = 15:12
1300      *     rz = 3:0
1301      *
1302      * And so we print the instructions as:
1303      * STREX ry, rz, [rx]
1304      * LDREX ry, [rx]
1305      */

1307 #endif /* ! codereview */
1308     cc = (in & ARM_CC_MASK) >> ARM_CC_SHIFT;
1309     rx = (in & ARM_ELS_RN_MASK) >> ARM_ELS_RN_SHIFT;
1310     ry = (in & ARM_ELS_RD_MASK) >> ARM_ELS_RD_SHIFT;
1311     rz = (in & ARM_ELS_LOW_AM_MASK);
1283     rn = (in & ARM_ELS_RN_MASK) >> ARM_ELS_RN_SHIFT;
1284     rd = (in & ARM_ELS_RD_MASK) >> ARM_ELS_RD_SHIFT;
1285     rm = in & ARM_ELS_RN_MASK;
1312     lbit = in & ARM_ELS_LBIT_MASK;

1314     len = snprintf(buf, buflen, "%s%sex %s, ",
1315                  lbit != 0 ? "ldr" : "str",
1316                  arm_cond_names[cc], arm_reg_names[ry]);
1290     arm_cond_names[cc], arm_reg_names[rd]);
1317     if (len >= buflen)
1318         return (-1);

1320     if (lbit)
1321         len += snprintf(buf + len, buflen - len, "[%s]",
1322                        arm_reg_names[rx]);

```

```

1296         arm_reg_names[rn]);
1323     else
1324         len += snprintf(buf + len, buflen - len, "%s, [%s]",
1325                        arm_reg_names[rz], arm_reg_names[rx]);
1299     arm_reg_names[rm], arm_reg_names[rn]);
1326     return (len >= buflen ? -1 : 0);
1327 }
_____unchanged_portion_omitted_____

```

```
*****
```

```
10566 Sat Feb 7 19:01:07 2015
```

```
new/usr/src/uts/arm/misc/fake_stubs.c
```

```
arm: implement getfp
```

```
*****
```

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright (c) 2014 Joyent, Inc. All rights reserved.
14 */

16 /*
17  * Fake stubs that we need to advance
18 */
19 extern void bop_panic(const char *);

21 #define STUB(x) void x(void) { bop_panic(#x); }

23 STUB(bcmap)
24 STUB(kdi_flush_caches)
25 STUB(kobj_text_alloc)
26 STUB(kdi_range_is_nontoxic)
27 STUB(dcache_flushall)
28 STUB(SHALFinal)
29 STUB(kdi_pwrite)
30 STUB(SHALUpdate)
31 STUB(hat_unload)
32 STUB(stubs_base)
33 STUB(rw_exit)
34 STUB(hat_getpfnum)
35 STUB(hat_devload)
36 STUB(kdi_vtop)
37 STUB(stubs_end)
38 STUB(SHALInit)
39 STUB(rw_enter)
40 STUB(kobj_texthole_free)
41 STUB(kdi_pread)
42 STUB(kobj_vmem_init)
43 STUB(cpr)
44 STUB(acct)
45 STUB(bind)
46 STUB(recv)
47 STUB(send)
48 STUB(spl0)
49 STUB(spl7)
50 STUB(spl8)
51 STUB(splx)
52 STUB(audit_init_module)
53 STUB(i_ddi_intr_ops)
54 STUB(dcopy_cmd_alloc)
55 STUB(impl_acc_hdl_free)
56 STUB(plat_hold_page)
57 STUB(check_status)
58 STUB(audit_symlink_create)
59 STUB(hat_page_clrattr)
60 STUB(copyinstr_noerr)
61 STUB(audit)
```

```
62 STUB(cas32)
63 STUB(cas64)
64 STUB(copyoutstr)
65 STUB(getfp)
65 STUB(htonl)
66 STUB(htons)
67 STUB(indir)
68 STUB(idmap_reg_dh)
69 STUB(kcopy)
70 STUB(kzero)
71 STUB(nosys)
72 STUB(ntohl)
73 STUB(ntohs)
74 STUB(fifo_vfastoff)
75 STUB(splhi)
76 STUB(ucopy)
77 STUB(uzero)
78 STUB(i_ddi_acc_clr_fault)
79 STUB(audit_setf)
80 STUB(audit_priv)
81 STUB(audit_exec)
82 STUB(audit_exit)
83 STUB(hat_leave_region)
84 STUB(door_ki_upcall)
85 STUB(unset_idle_cpu)
86 STUB(thread_onproc)
87 STUB(set_proc_ast)
88 STUB(ddi_rep_put64)
89 STUB(ddi_rep_put32)
90 STUB(ddi_rep_put16)
91 STUB(boot_virt_alloc)
92 STUB(ddi_rep_get64)
93 STUB(ddi_rep_get32)
94 STUB(ddi_rep_get16)
95 STUB(i_ddi_map_fault)
96 STUB(lwp_stk_cache_init)
97 STUB(dtrace_interrupt_enable)
98 STUB(ftrace_interrupt_enable)
99 STUB(xcopyout_nta)
100 STUB(hat_pagesync)
101 STUB(console_enter)
102 STUB(spec_snode_walk)
103 STUB(audit_chdirec)
104 STUB(prinvaldate)
105 STUB(lock_clear)
106 STUB(ka_init)
107 STUB(loadable_syscall)
108 STUB(sockconfig)
109 STUB(fuword8_noerr)
110 STUB(lwp_detach_brand_hdlrs)
111 STUB(valid_va_range_aligned)
112 STUB(lwp_forkregs)
113 STUB(devfs_devpolicy)
114 STUB(hat_stats_disable)
115 STUB(pr_free_watched_pages)
116 STUB(install_trap)
117 STUB(dtrace_membar_consumer)
118 STUB(socket_sendmblk)
119 STUB(audit_symlink)
120 STUB(i_ddi_apply_range)
121 STUB(lock_clear_splx)
122 STUB(audit_strputmsg)
123 STUB(i_ddi_alloc_intr_phdl)
124 STUB(i_ddi_acc_set_fault)
125 STUB(clconf_get_nodeid)
126 STUB(e_ddi_copyfromdev)
```

```

127 STUB(impl_acc_hdl_alloc)
128 STUB(sdev_devstate_change)
129 STUB(translate_devid)
130 STUB(impl_keep_instance)
131 STUB(hat_stats_enable)
132 STUB(hr_clock_unlock)
133 STUB(audit_closet)
134 STUB(hat_join_srd)
135 STUB(hat_softlock)
136 STUB(spec_is_clone)
137 STUB(audit_fdsend)
138 STUB(audit_fdrecv)
139 STUB(random_get_bytes)
140 STUB(audit_finish)
141 STUB(pf_is_memory)
142 STUB(peekpoke_mem)
143 STUB(fastboot_update_config)
144 STUB(audit_savepath)
145 STUB(hat_get_mapped_size)
146 STUB(thread_stk_init)
147 STUB(hat_free_start)
148 STUB(impl_ddi_sunbus_initchild)
149 STUB(lwp_rtt)
150 STUB(prlwpfree)
151 STUB(prlwpexit)
152 STUB(hat_memload)
153 STUB(console_exit)
154 STUB(map_addr_vacalign_check)
155 STUB(hat_pageunload)
156 STUB(spec_fence_snode)
157 STUB(copyout_noerr)
158 STUB(audit_vncreate_finish)
159 STUB(on_fault)
160 STUB(door_ki_lookup)
161 STUB(lbolt_softint_post)
162 STUB(door_revoke_all)
163 STUB(spec_is_selfclone)
164 STUB(prefetch_write_many)
165 STUB(dump_plat_addr)
166 STUB(dump_plat_data)
167 STUB(au_to_arg32)
168 STUB(random_get_pseudo_bytes)
169 STUB(num_phys_pages)
170 STUB(cmp_set_nosteal_interval)
171 STUB(no_fault)
172 STUB(sync_icache)
173 STUB(lock_try)
174 STUB(lock_set)
175 STUB(sock_getmsg)
176 STUB(getsetcontext)
177 STUB(i_ddi_rnumber_to_regspec)
178 STUB(lock_spin_try)
179 STUB(_insque)
180 STUB(sock_putmsg)
181 STUB(pr_isself)
182 STUB(save_syscall_args)
183 STUB(getsockname)
184 STUB(fss_allocbuf)
185 STUB(poke_cpu)
186 STUB(lbolt_softint_add)
187 STUB(on_trap)
188 STUB(ip_ocsum)
189 STUB(audit_vncreate_start)
190 STUB(i_ddi_free_intr_phdl)
191 STUB(accept)
192 STUB(kcopy_nta)

```

```

193 STUB(audit_devpolicy)
194 STUB(page_mem_avail)
195 STUB(door_exit)
196 STUB(door_fork)
197 STUB(door_slam)
198 STUB(_remque)
199 STUB(valid_usr_range)
200 STUB(i_ddi_bus_map)
201 STUB(caller)
202 STUB(casptr)
203 STUB(snf_segmap)
204 STUB(so_socket)
205 STUB(copyin)
206 STUB(socket_setsockopt)
207 STUB(getpstack)
208 STUB(va_to_pfn)
209 STUB(specfind)
210 STUB(gethrestime_sec)
211 STUB(hat_unlock)
212 STUB(ovbcopy)
213 STUB(au_uwrite)
214 STUB(spec_segmap)
215 STUB(exec_set_sp)
216 STUB(copyin_noerr)
217 STUB(audit_setppriv)
218 STUB(listen)
219 STUB(lowbit)
220 STUB(mdbboot)
221 STUB(door_ki_rele)
222 STUB(door_ki_hold)
223 STUB(door_ki_info)
224 STUB(door_ki_open)
225 STUB(i_ddi_add_softint)
226 STUB(prexit)
227 STUB(prfree)
228 STUB(prstep)
229 STUB(kpreempt)
230 STUB(mdpreamble)
231 STUB(resume)
232 STUB(hr_clock_lock)
233 STUB(prrelvm)
234 STUB(sendto)
235 STUB(sir_on)
236 STUB(subyte)
237 STUB(idmap_get_door)
238 STUB(vpanic)
239 STUB(pagezero)
240 STUB(i_ddi_remove_softint)
241 STUB(dcopy_free)
242 STUB(copyinstr)
243 STUB(thread_load)
244 STUB(makectty)
245 STUB(set_all_zone_usr_proc_sys)
246 STUB(hat_flush_range)
247 STUB(impl_assign_instance)
248 STUB(randtick)
249 STUB(copyoutstr_noerr)
250 STUB(hat_memload_region)
251 STUB(map_addr)
252 STUB(map_pgsz)
253 STUB(devi_stillreferenced)
254 STUB(i_ddi_cacheattr_to_hatacc)
255 STUB(spec_ufence_snode)
256 STUB(i_ddi_devacc_to_hatacc)
257 STUB(prbarrier)
258 STUB(audit_setfsat_path)

```

```

259 STUB(hat_dump)
260 STUB(hat_exit)
261 STUB(hat_sync)
262 STUB(gethrestime)
263 STUB(suword8_noerr)
264 STUB(recvmsg)
265 STUB(suword16_noerr)
266 STUB(fuword16_noerr)
267 STUB(au_free_rec)
268 STUB(cpu_intr_swtdch_exit)
269 STUB(clconf_maximum_nodeid)
270 STUB(devfs_clean)
271 STUB(sysdc_thread_enter)
272 STUB(dump_plat_pfn)
273 STUB(hat_chgprot)
274 STUB(hat_chgattr)
275 STUB(syscall_ap)
276 STUB(tnf_opaque_array_1)
277 STUB(map_pgszvec)
278 STUB(lwp_setrval)
279 STUB(semexit)
280 STUB(sendmsg)
281 STUB(setregs)
282 STUB(resume_from_zombie)
283 STUB(shmexit)
284 STUB(shmfork)
285 STUB(i_ddi_mem_alloc)
286 STUB(hat_supported)
287 STUB(spec_assoc_vp_with_devi)
288 STUB(dcopy_cmd_post)
289 STUB(dcopy_cmd_poll)
290 STUB(dcopy_cmd_free)
291 STUB(i_ddi_intr_redist_all_cpus)
292 STUB(impl_fix_props)
293 STUB(dld_autopush)
294 STUB(cladmin)
295 STUB(resume_from_intr)
296 STUB(pr_isobject)
297 STUB(spec_devi_open_count)
298 STUB(lwp_rtt_initial)
299 STUB(hat_clrattr)
300 STUB(hat_alloc)
301 STUB(hat_enter)
302 STUB(set_errno)
303 STUB(setsockopt)
304 STUB(getsockopt)
305 STUB(connect)
306 STUB(hat_probe)
307 STUB(copyout)
308 STUB(copystr)
309 STUB(ucopystr)
310 STUB(hat_share)
311 STUB(hat_setup)
312 STUB(splhigh)
313 STUB(hat_page_getshare)
314 STUB(hat_unlock_region)
315 STUB(hat_swapout)
316 STUB(sulword)
317 STUB(fastboot_update_and_load)
318 STUB(suword8)
319 STUB(ddi_get8)
320 STUB(ddi_put8)
321 STUB(gethrtime)
322 STUB(fifo_getinfo)
323 STUB(auditdoor)
324 STUB(ddi_rep_put8)

```

```

325 STUB(ddi_rep_get8)
326 STUB(hat_page_checkshare)
327 STUB(impl_ddi_prop_int_from_prom)
328 STUB(tod_get)
329 STUB(tod_set)
330 STUB(au_doormsg)
331 STUB(nl7c_sendfilev)
332 STUB(scalehrtime)
333 STUB(so_socketpair)
334 STUB(getpeername)
335 STUB(hat_page_getattr)
336 STUB(recvfrom)
337 STUB(i_ddi_check_cache_attr)
338 STUB(hat_memload_array)
339 STUB(getuserpc)
340 STUB(prexecstart)
341 STUB(hat_unload_callback)
342 STUB(door_ki_upcall_limited)
343 STUB(hat_kpm_page2va)
344 STUB(gethrtime_waitfree)
345 STUB(hat_unshare)
346 STUB(i_ddi_set_softint_pri)
347 STUB(makespecvp)
348 STUB(common_specvp)
349 STUB(suword32_noerr)
350 STUB(fuword32_noerr)
351 STUB(plat_tod_fault)
352 STUB(suword32)
353 STUB(suword16)
354 STUB(fuword16)
355 STUB(fuword32)
356 STUB(hat_join_region)
357 STUB(kidmap_getsidbygid)
358 STUB(prexecend)
359 STUB(kidmap_getgidbysid)
360 STUB(kidmap_getuidbysid)
361 STUB(kidmap_getsidbyuid)
362 STUB(impl_acc_hdl_get)
363 STUB(i_ddi_trigger_softint)
364 STUB(exec_get_spslew)
365 STUB(debug_enter)
366 STUB(pr_allstopping)
367 STUB(zfs_prop_to_name)
368 STUB(dtrace_membar_producer)
369 STUB(idmap_purge_cache)
370 STUB(dtrace_gethrtime)
371 STUB(sosendfile64)
372 STUB(prefetch_smap_w)
373 STUB(hat_getpagesize)
374 STUB(cpu_intr_swtdch_enter)
375 STUB(devfs_walk)
376 STUB(hat_getattr)
377 STUB(prefetch_page_r)
378 STUB(fulword)
379 STUB(fuword8)
380 STUB(fss_freebuf)
381 STUB(hat_memload_array_region)
382 STUB(hat_kpm_mapin)
383 STUB(spec_getvnodeops)
384 STUB(hat_thread_exit)
385 STUB(hat_dup_region)
386 STUB(fss_changepset)
387 STUB(fss_changeproj)
388 STUB(lwp_stk_init)
389 STUB(lwp_stk_fini)
390 STUB(hat_free_end)

```

```

391 STUB(lwp_pcb_exit)
392 STUB(lwp_load)
393 STUB(hat_dup)
394 STUB(hat_map)
395 STUB(hat_kpm_mapout)
396 STUB(set_proc_post_sys)
397 STUB(e_ddi_copytodev)
398 STUB(idmap_unreg_dh)
399 STUB(set_idle_cpu)
400 STUB(gethrestime_lasttick)
401 STUB(lock_set_spl)
402 STUB(highbit)
403 STUB(cl_flk_state_transition_notify)
404 STUB(drv_usecwait)
405 STUB(set_base_spl)
406 STUB(fttrace_interrupt_disable)
407 STUB(impl_free_instance)
408 STUB(intr_passivate)
409 STUB(dcopy_alloc)
410 STUB(valid_va_range)
411 STUB(ddi_get64)
412 STUB(ddi_get32)
413 STUB(ddi_get16)
414 STUB(ddi_put64)
415 STUB(ddi_put32)
416 STUB(ddi_put16)
417 STUB(sock_getfasync)
418 STUB(dtrace_interrupt_disable)
419 STUB(lwp_freeregs)
420 STUB(xcopyin_nta)
421 STUB(i_ddi_mem_free)
422 STUB(hat_page_setattr)
423 STUB(impl_setup_ddi)
424 STUB(shutdown)
425 STUB(audit_anchorpath)
426 STUB(i_convert_boot_device_name)
427 STUB(dsl_prop_get)
428 STUB(__aeabi_llsr)
429 STUB(__aeabi_llsl)
430 STUB(siron)
431 STUB(panic_saveregs)
432 STUB(panic_savetrap)
433 STUB(panic_quiesce_hw)
434 STUB(panic_stopcpus)
435 STUB(mp_cpu_poweroff)
436 STUB(cpu_create_intrstat)
437 STUB(mp_cpu_faulted_enter)
438 STUB(pg_plat_hw_shared)
439 STUB(cpupm_plat_domain_id)
440 STUB(bp_color)
441 STUB(pg_plat_cmt_policy)
442 STUB(siron_poke_cpu)
443 STUB(getpil)
444 STUB(panic_showtrap)
445 STUB(cpu_disable_intr)
446 STUB(setjmp)
447 STUB(traceregs)
448 STUB(unscalehrtime)
449 STUB(cpupm_plat_state_enumerate)
450 STUB(mp_cpu_stop)
451 STUB(pg_plat_cpus_share)
452 STUB(pg_plat_hw_rank)
453 STUB(cpu_enable_intr)
454 STUB(mp_cpu_faulted_exit)
455 STUB(mp_cpu_unconfigure)
456 STUB(pg_plat_get_core_id)

```

```

457 STUB(get_cpu_mstate)
458 STUB(elfexec)
459 STUB(pg_plat_hw_instance_id)
460 STUB(mapexec_brand)
461 STUB(panic_trigger)
462 STUB(cpu_delete_intrstat)
463 STUB(panic_dump_hw)
464 STUB(panic_enter_hw)
465 STUB(cpupm_plat_change_state)
466 STUB(mp_cpu_start)
467 STUB(mp_cpu_configure)
468 STUB(mach_cpu_pause)
469 STUB(kdi_siron)
470 STUB(ld_ib_prop)
471 STUB(mp_cpu_poweron)
472 STUB(strplumb)
473 STUB(consconfig)
474 STUB(release_bootstrap)
475 STUB(cluster)
476 STUB(reset_syscall_args)
477 STUB(halt)
478 STUB(cbe_init_pre)
479 STUB(cbe_init)
480 STUB(post_startup)
481 STUB(start_other_cpus)
482 STUB(dtrace_safe_synchronous_signal)
483 STUB(prstop)
484 STUB(prnotify)
485 STUB(prnstep)
486 STUB(sendsig)
487 STUB(audit_core_start)
488 STUB(dtrace_safe_defer_signal)
489 STUB(audit_core_finish)
490 STUB(reset)
491 STUB(prom_enter_mon)
492 STUB(mutex_gettick)
493 STUB(splr)
494 STUB(ulock_clear)
495 STUB(cu_plat_cpc_init)
496 STUB(kcpc_hw_load_pcbe)
497 STUB(spl_xcall)
498 STUB(devfs_reset_perm)
499 STUB(clboot_modload)
500 STUB(devfs_remdrv_cleanup)
501 STUB(sdev_modctl_readdir_free)
502 STUB(prom_panic)
503 STUB(hat_kpm_mseghash_clear)
504 STUB(add_physmem_cb)
505 STUB(sdev_modctl_readdir)
506 STUB(ulock_try)
507 STUB(ppmapout)
508 STUB(ppmapin)
509 STUB(ppcopy)
510 STUB(cpu_call)
511 STUB(hat_reserve)
512 STUB(sdev_modctl_devexists)
513 STUB(devname_profile_update)
514 STUB(spa_boot_init)
515 STUB(clboot_rootconf)
516 STUB(sync_data_memory)
517 STUB(pagescrub)
518 STUB(clboot_mountroot)
519 STUB(devname_filename_register)
520 STUB(hat_kpm_mseghash_update)
521 STUB(hat_page_demote)
522 STUB(strplumb_get_netdev_path)

```

`new/usr/src/uts/arm/misc/fake_stubs.c`

9

523 STUB(arm_gettick)

new/usr/src/uts/arm/ml/arm_subr.s

1

1237 Sat Feb 7 19:01:07 2015

new/usr/src/uts/arm/ml/arm_subr.s

arm: implement getfp

_____unchanged_portion_omitted_____

63 #endif /* __lint */

65 ENTRY(getfp)

66 mov r0, r9

67 bx lr

68 SET_SIZE(getfp)

69 #endif /* !codereview */

new/usr/src/uts/armv6/bcm2835/Makefile.files

1

659 Sat Feb 7 19:01:07 2015

new/usr/src/uts/armv6/bcm2835/Makefile.files

bcm2835: use the real uart instead of the mini-uart

The real uart is more capable. We'll want to use it for the real console eventually anyway, so let's bite the bullet now when no one will really notice. (For comparison, the Linux kernel uses the real uart and totally lacks a driver for the miniuart.)

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License ("CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #
```

```
12 #
13 # Copyright (c) 2013, Joyent, Inc. All rights reserved.
14 # Copyright (c) 2015, Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 #
```

```
17 BCM2835_OBJS = \
18     bcm2835_bsmdep.o \
19     bcm2835_uart.o \
20 #endif /* ! codereview */
21     boot_console.o \
22     locore.o
19     locore.o \
20     miniuart.o
```

```
24 BCM2835_LOADER_OBJS = \
25     bcm2835_ldep.o
```



```

*****
4202 Sat Feb 7 19:01:08 2015
new/usr/src/uts/armv6/bcm2835/loader/bcm2835_ldep.c
bcm2835: use the real uart instead of the mini-uart
The real uart is more capable. We'll want to use it for the real console
eventually anyway, so let's bite the bullet now when no one will really
notice. (For comparison, the Linux kernel uses the real uart and totally
lacks a driver for the miniuart.)
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright (c) 2013 Joyent, Inc. All rights reserved.
14  * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15  */

17 #include <sys/elf.h>
18 #include <sys/ata.h>

20 /*
21  * The primary serial console that we end up using is the normal UART, not
22  * the mini-uart that shares interrupts and registers with the SPI masters
23  * as well.
24  * The primary serial console that we end up using is not in fact a normal UART,
25  * but is instead actually a mini-uart that shares interrupts and registers with
26  * the SPI masters as well. While the RPi also supports another more traditional
27  * UART, that isn't what we are actually hooking up to generally with the
28  * adafruit cable. We already wasted our time having to figure that out. --
29  */

26 #define UART_BASE          0x20201000
27 #define UART_DR             0x0
28 #define UART_FR             0x18
29 #define UART_IBRD           0x24
30 #define UART_FBRD           0x28
31 #define UART_LCRH           0x2c
32 #define UART_CR             0x30
33 #define UART_ICR            0x44

35 #define UART_FR_RXFE        0x10      /* RX fifo empty */
36 #define UART_FR_TXFF        0x20      /* TX fifo full */

38 #define UART_LCRH_FEN        0x00000010 /* fifo enable */
39 #define UART_LCRH_WLEN_8    0x00000060 /* 8 bits */

41 #define UART_CR_UARTEN      0x001      /* uart enable */
42 #define UART_CR_TXE         0x100      /* TX enable */
43 #define UART_CR_RXE         0x200      /* RX enable */

28 #define AUX_BASE           0x20215000
29 #define AUX_ENABLES         0x4
30 #define AUX_MU_IO_REG       0x40
31 #define AUX_MU_IER_REG      0x44
32 #define AUX_MU_IIR_REG      0x48
33 #define AUX_MU_LCR_REG      0x4C
34 #define AUX_MU_MCR_REG      0x50
35 #define AUX_MU_LSR_REG      0x54
36 #define AUX_MU_CNTL_REG     0x60

```

```

37 #define AUX_MU_BAUD         0x68

39 #define AUX_MU_RX_READY     0x01
40 #define AUX_MU_TX_READY     0x20

46 /*
47  * All we care about are pins 14 and 15 for the UART. Specifically, alt0
48  * for GPIO14 is TXD0 and GPIO15 is RXD0. Those are controlled by FSEL1.
49  * For the mini UART, all we care about are pins 14 and 15 for the UART.
50  * Specifically, alt5 for GPIO14 is TXD1 and GPIO15 is RXD1. Those are
51  * controlled by FSEL1.
52  */
49 /*
50 #define GPIO_BASE          0x20200000
51 #define GPIO_FSEL1         0x4
52 #define GPIO_PUD           0x94
53 #define GPIO_PUDCLK0       0x98

55 #define GPIO_SEL_ALT0       0x4
56 #define GPIO_SEL_ALT5       0x2
57 #define GPIO_UART_MASK     0xfffc0fff
58 #define GPIO_UART_TX_SHIFT  12
59 #define GPIO_UART_RX_SHIFT  15

60 #define GPIO_PUD_DISABLE    0x0
61 #define GPIO_PUDCLK_UART   0x0000c000

63 static __GNU_INLINE uint32_t arm_reg_read(uint32_t reg)
64 {
65     volatile uint32_t *ptr = (volatile uint32_t *)reg;

67     return *ptr;
68 }
_____unchanged_portion_omitted_____

77 /*
78  * A simple nop
79  */
80 static void
81 uart_nop(void)
82 {
83     __asm__ volatile("mov r0, r0\n" : : :);
84 }
_____unchanged_portion_omitted_____

97 void
98 fakeload_backend_init(void)
99 {
100     uint32_t v;
101     int i;

103     /* disable UART */
104     arm_reg_write(UART_BASE + UART_CR, 0);
105     /* Enable the mini UAT */
106     arm_reg_write(AUX_BASE + AUX_ENABLES, 0x1);

103     /* Disable interrupts */
104     arm_reg_write(AUX_BASE + AUX_MU_IER_REG, 0x0);

106     /* Disable the RX and TX */
107     arm_reg_write(AUX_BASE + AUX_MU_CNTL_REG, 0x0);

109     /*
110      * Enable 8-bit word length. External sources tell us the PRM is buggy
111      * here and that even though bit 1 is reserved, we need to actually set
112      * it to get 8-bit words.

```

```

113  */
114  arm_reg_write(AUX_BASE + AUX_MU_LCR_REG, 0x3);

116  /* Set RTS high */
117  arm_reg_write(AUX_BASE + AUX_MU_MCR_REG, 0x0);

119  /* Disable interrupts */
120  arm_reg_write(AUX_BASE + AUX_MU_IER_REG, 0x0);

122  /* Set baud rate */
123  arm_reg_write(AUX_BASE + AUX_MU_IIR_REG, 0xc6);
124  arm_reg_write(AUX_BASE + AUX_MU_BAUD, 0x10e);

106  /* TODO: Factor out the gpio bits */
107  v = arm_reg_read(GPIO_BASE + GPIO_FSEL1);
108  v &= GPIO_UART_MASK;
109  v |= GPIO_SEL_ALT0 << GPIO_UART_RX_SHIFT;
110  v |= GPIO_SEL_ALT0 << GPIO_UART_TX_SHIFT;
129  v |= GPIO_SEL_ALT5 << GPIO_UART_RX_SHIFT;
130  v |= GPIO_SEL_ALT5 << GPIO_UART_TX_SHIFT;
111  arm_reg_write(GPIO_BASE + GPIO_FSEL1, v);

113  arm_reg_write(GPIO_BASE + GPIO_PUD, GPIO_PUD_DISABLE);
114  for (i = 0; i < 150; i++)
115      uart_nop();
135  bcm2835_minuart_nop();
116  arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, GPIO_PUDCLK_UART);
117  for (i = 0; i < 150; i++)
118      uart_nop();
138  bcm2835_minuart_nop();
139  // XXX: GPIO_PUD_DISABLE again?
119  arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, 0);

121  /* clear all interrupts */
122  arm_reg_write(UART_BASE + UART_ICR, 0x7ff);

124  /* set the baud rate */
125  arm_reg_write(UART_BASE + UART_IBRD, 1);
126  arm_reg_write(UART_BASE + UART_FBRD, 40);

128  /* select 8-bit, enable FIFOs */
129  arm_reg_write(UART_BASE + UART_LCRH, UART_LCRH_WLEN_8 | UART_LCRH_FEN);

131  /* enable UART */
132  arm_reg_write(UART_BASE + UART_CR, UART_CR_UARTEN | UART_CR_TXE |
133              UART_CR_RXE);
142  /* Finally, go back and enable RX and TX */
143  arm_reg_write(AUX_BASE + AUX_MU_CNTL_REG, 0x3);
134  }

136 void
137 fakeload_backend_putc(int c)
138 {
139     if (c == '\n')
140         fakeload_backend_putc('\r');

142     while (arm_reg_read(UART_BASE + UART_FR) & UART_FR_TXFF)
143         ;
144     arm_reg_write(UART_BASE + UART_DR, c & 0x7f);
145     if (c == '\n')
146         fakeload_backend_putc('\r');
152     for (;;) {
153         if (arm_reg_read(AUX_BASE + AUX_MU_LSR_REG) & AUX_MU_TX_READY)
154             break;
155     }
156     arm_reg_write(AUX_BASE + AUX_MU_IO_REG, c & 0x7f);

```

```

147 }

149 /*
150  * Add a map for the uart.
151  */
152 void
153 fakeload_backend_addmaps(atag_header_t *chain)
154 {
155     atag_illumos_mapping_t aim;

157     aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
158     aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
159     aim.aim_paddr = GPIO_BASE;
160     aim.aim_vaddr = GPIO_BASE;
161     aim.aim_vlen = 0x1000;
162     aim.aim_plen = 0x1000;
163     aim.aim_mapflags = PF_R | PF_W | PF_NORELOC | PF_DEVICE;
164     atag_append(chain, &aim.aim_header);

166     aim.aim_header.ah_size = ATAG_ILLUMOS_MAPPING_SIZE;
167     aim.aim_header.ah_tag = ATAG_ILLUMOS_MAPPING;
168     aim.aim_paddr = UART_BASE;
169     aim.aim_vaddr = UART_BASE;
178     aim.aim_paddr = AUX_BASE;
179     aim.aim_vaddr = AUX_BASE;
170     aim.aim_vlen = 0x1000;
171     aim.aim_plen = 0x1000;
172     aim.aim_mapflags = PF_R | PF_W | PF_NORELOC | PF_DEVICE;
173     atag_append(chain, &aim.aim_header);
174 }

```

unchanged_portion_omitted

```

*****
811 Sat Feb 7 19:01:08 2015
new/usr/src/uts/armv6/bcm2835/ml/locore.s
armv6: bcm2835 & qvpb have nearly identical locore_start
It makes sense to common-ize _start for all armv6 machines. They will all
have to do the same basic setup. If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 */

17 #include <sys/asm_linkage.h>
18 #include <sys/machparam.h>
19 #include <sys/cpu_asm.h>

21     ENTRY(_mach_start)
22 /*
23  * Every story needs a beginning. This is ours.
24 */

25 /*
26  * We are in a primordial world here. The BMC2835 is going to come along and
27  * boot us at _start. Normally we would go ahead and use a main() function, but
28  * for now, we'll do that ourselves. As we've started the world, we also need to
29  * set up a few things about us, for example our stack pointer. To help us out,
30  * it's useful to remember the rough memory map. Remember, this is for physcial
31  * addresses. There is no virtual memory here. These sizes are often manipulated
32  * by the 'configuration' in the bootloader.
33  */
34  * +-----+ <---- Max physical memory
35  * |
36  * |
37  * |
38  * |
39  * |
40  * |   I/O
41  * |   Peripherals
42  * |
43  * |-----+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
44  * |
45  * |   Main
46  * |   Memory
47  * |
48  * |-----+ <---- Top of SDRAM
49  * |
50  * |   VC
51  * |   SDRAM
52  * |
53  * |-----+ <---- Split determined by bootloader config
54  * |
55  * |   ARM
56  * |   SDRAM
57  * |

```

```

58  * +-----+ <---- Bottom of physical memory 0x00000000
59  *
60  * With the Raspberry Pi Model B, we have 512 MB of SDRAM. That means we have a
61  * range of addresses from [0, 0x20000000). If we assume that the minimum amount
62  * of DRAM is given to the GPU - 32 MB, that means we really have the following
63  * range: [0, 0x1e000000).
64  *
65  * By default, this binary will be loaded into 0x8000. For now, that means we
66  * will set our initial stack to 0x10000000.
67 */

69 /*
70  * Recall that _start is the traditional entry point for an ELF binary.
71 */
72     ENTRY(_start)
73     ldr sp, =t0stack
74     ldr r4, =DEFAULTSTKSZ
75     add sp, r4
76     bic sp, sp, #0xff

78 /*
79  * establish bogus stacks for exceptional CPU states, our exception
80  * code should never make use of these, and we want loud and violent
81  * failure should we accidentally try.
82  */
83     cps #(CPU_MODE_UND)
84     mov sp, #-1
85     cps #(CPU_MODE_ABT)
86     mov sp, #-1
87     cps #(CPU_MODE_FIQ)
88     mov sp, #-1
89     cps #(CPU_MODE_IRQ)
90     mov sp, #-1
91     cps #(CPU_MODE_SVC)

93     /* Enable highvecs (moves the base of the exception vector) */
94     mrc     p15, 0, r3, c1, c0, 0
95     mov     r4, #1
96     lsl     r4, r4, #13
97     orr     r3, r3, r4
98     mcr     p15, 0, r3, c1, c0, 0

22     /* Enable access to p10 and p11 (privileged mode only) */
23     mrc     p15, 0, r0, c1, c0, 2
24     orr     r0, #0x00500000
25     mcr     p15, 0, r0, c1, c0, 2

27     bx     r14
28     SET_SIZE(_mach_start)
105     bl _fakebop_start
106     SET_SIZE(_start)

108     ENTRY(arm_reg_read)
109     ldr r0, [r0]
110     bx lr
111     SET_SIZE(arm_reg_read)

113     ENTRY(arm_reg_write)
114     str r1, [r0]
115     bx lr
116     SET_SIZE(arm_reg_write)

```

```

*****
3422 Sat Feb 7 19:01:08 2015
new/usr/src/uts/armv6/bcm2835/os/bcm2835_uart.c
bcm2835: use the real uart instead of the mini-uart
The real uart is more capable. We'll want to use it for the real console
eventually anyway, so let's bite the bullet now when no one will really
notice. (For comparison, the Linux kernel uses the real uart and totally
lacks a driver for the miniuart.)
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 */

17 /*
18  * A simple uart driver for the RPi.
19 */
20 #include <sys/types.h>

22 #include "bcm2835_uart.h"

24 extern uint32_t arm_reg_read(uint32_t);
25 extern void arm_reg_write(uint32_t, uint32_t);

27 /*
28  * The primary serial console that we end up using is the normal UART, not
29  * the mini-uart that shares interrupts and registers with the SPI masters
30  * as well.
31 */

33 #define UART_BASE          0x20201000
34 #define UART_DR            0x0
35 #define UART_FR            0x18
36 #define UART_IBRD          0x24
37 #define UART_FBRD          0x28
38 #define UART_LCRH          0x2c
39 #define UART_CR            0x30
40 #define UART_ICR           0x44

42 #define UART_FR_RXFE       0x10      /* RX fifo empty */
43 #define UART_FR_TXFF       0x20      /* TX fifo full */

45 #define UART_LCRH_FEN      0x00000010 /* fifo enable */
46 #define UART_LCRH_WLEN_8   0x00000060 /* 8 bits */

48 #define UART_CR_UARTEN     0x001      /* uart enable */
49 #define UART_CR_TXE        0x100      /* TX enable */
50 #define UART_CR_RXE        0x200      /* RX enable */

53 /*
54  * All we care about are pins 14 and 15 for the UART. Specifically, alt0
55  * for GPIO14 is TXD0 and GPIO15 is RXD0. Those are controlled by FSEL1.
56  */
57 #define GPIO_BASE          0x20200000

```

```

58 #define GPIO_FSEL1        0x4
59 #define GPIO_PUD          0x94
60 #define GPIO_PUDCLK0     0x98

62 #define GPIO_SEL_ALT0     0x4
63 #define GPIO_UART_MASK   0xffc0fff
64 #define GPIO_UART_TX_SHIFT 12
65 #define GPIO_UART_RX_SHIFT 15

67 #define GPIO_PUD_DISABLE  0x0
68 #define GPIO_PUDCLK_UART 0x0000c000

70 /*
71  * A simple nop
72 */
73 static void
74 bcm2835_uart_nop(void)
75 {
76     __asm__ volatile("mov r0, r0\n" : : :);
77 }

79 void
80 bcm2835_uart_init(void)
81 {
82     uint32_t v;
83     int i;

85     /* disable UART */
86     arm_reg_write(UART_BASE + UART_CR, 0);

88     /* TODO: Factor out the gpio bits */
89     v = arm_reg_read(GPIO_BASE + GPIO_FSEL1);
90     v &= GPIO_UART_MASK;
91     v |= GPIO_SEL_ALT0 << GPIO_UART_RX_SHIFT;
92     v |= GPIO_SEL_ALT0 << GPIO_UART_TX_SHIFT;
93     arm_reg_write(GPIO_BASE + GPIO_FSEL1, v);

95     arm_reg_write(GPIO_BASE + GPIO_PUD, GPIO_PUD_DISABLE);
96     for (i = 0; i < 150; i++)
97         bcm2835_uart_nop();
98     arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, GPIO_PUDCLK_UART);
99     for (i = 0; i < 150; i++)
100         bcm2835_uart_nop();
101     arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, 0);

103     /* clear all interrupts */
104     arm_reg_write(UART_BASE + UART_ICR, 0x7ff);

106     /* set the baud rate */
107     arm_reg_write(UART_BASE + UART_IBRD, 1);
108     arm_reg_write(UART_BASE + UART_FBRD, 40);

110     /* select 8-bit, enable FIFOs */
111     arm_reg_write(UART_BASE + UART_LCRH, UART_LCRH_WLEN_8 | UART_LCRH_FEN);

113     /* enable UART */
114     arm_reg_write(UART_BASE + UART_CR, UART_CR_UARTEN | UART_CR_TXE |
115                 UART_CR_RXE);
116 }

118 void
119 bcm2835_uart_putc(uint8_t c)
120 {
121     while (arm_reg_read(UART_BASE + UART_FR) & UART_FR_TXFF)
122         ;
123     arm_reg_write(UART_BASE + UART_DR, c & 0x7f);

```

```
124     if (c == '\n')
125         bcm2835_uart_putc('\r');
126 }

128 uint8_t
129 bcm2835_uart_getc(void)
130 {
131     while (arm_reg_read(UART_BASE + UART_FR) & UART_FR_RXFE)
132         ;
133     return (arm_reg_read(UART_BASE + UART_DR) & 0x7f);
134 }

136 int
137 bcm2835_uart_isc(void)
138 {
139     return ((arm_reg_read(UART_BASE + UART_FR) & UART_FR_RXFE) == 0);
140 }
141 #endif /* ! codereview */
```

new/usr/src/uts/armv6/bcm2835/os/bcm2835_uart.h

1

884 Sat Feb 7 19:01:08 2015

new/usr/src/uts/armv6/bcm2835/os/bcm2835_uart.h

bcm2835: use the real uart instead of the mini-uart

The real uart is more capable. We'll want to use it for the real console eventually anyway, so let's bite the bullet now when no one will really notice. (For comparison, the Linux kernel uses the real uart and totally lacks a driver for the minuart.)

```
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */
11
12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15  */
16
17 #ifndef _BCM2835_UART_H
18 #define _BCM2835_UART_H
19
20 /*
21  * Interface to the BCM2835's uart.
22  */
23
24 #ifdef __cplusplus
25 extern "C" {
26 #endif
27
28 #include <sys/types.h>
29
30 void bcm2835_uart_init(void);
31 void bcm2835_uart_putc(uint8_t);
32 uint8_t bcm2835_uart_getc(void);
33 int bcm2835_uart_isc(void);
34
35 #ifdef __cplusplus
36 }
37 #endif
38
39 #endif /* _BCM2835_UART_H */
40 #endif /* ! codereview */
```

```
*****
1609 Sat Feb  7 19:01:08 2015
new/usr/src/uts/armv6/bcm2835/os/boot_console.c
bcm2835: use the real uart instead of the mini-uart
The real uart is more capable. We'll want to use it for the real console
eventually anyway, so let's bite the bullet now when no one will really
notice. (For comparison, the Linux kernel uses the real uart and totally
lacks a driver for the miniuart.)
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright (c) 2013 Joyent, Inc. All rights reserved.
14 */

16 /*
17  * bcm2835 boot console implementation
18 */

20 #include "bcm2835_uart.h"
20 #include "miniuart.h"

22 /*
23  * There are a few different potential boot consoles that we could have on the
24  * bcm2835. There is both a mini uart and a full functioning uart. Generally,
25  * people will use one of them, but we want to support both. As such we have a
25  * people will use the mini uart, but we want to support both. As such we have a
26  * single global ops vector that we set once during bcons_init and never again.
27 */
28 #define BMC2835_CONSNAME_MAX 24
29 typedef struct bcm2835_consops {
30     char bco_name[BMC2835_CONSNAME_MAX];
31     void (*bco_putc)(uint8_t);
32     uint8_t (*bco_getc)(void);
33     int (*bco_isc)(void);
34 } bcm2835_consops_t;

36 static bcm2835_consops_t consops;

38 /*
39  * For now, we only support the real uart.
39  * For now, we only support the mini uart.
40 */
41 void
42 bcons_init(char *bstr)
43 {
44     bcm2835_uart_init();
45     consops.bco_putc = bcm2835_uart_putc;
46     consops.bco_getc = bcm2835_uart_getc;
47     consops.bco_isc = bcm2835_uart_isc;
48     bcm2835_miniuart_init();
49     consops.bco_putc = bcm2835_miniuart_putc;
50     consops.bco_getc = bcm2835_miniuart_getc;
51     consops.bco_isc = bcm2835_miniuart_isc;
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }
1001 }
1002 }
1003 }
1004 }
1005 }
1006 }
1007 }
1008 }
1009 }
1010 }
1011 }
1012 }
1013 }
1014 }
1015 }
1016 }
1017 }
1018 }
1019 }
1020 }
1021 }
1022 }
1023 }
1024 }
1025 }
1026 }
1027 }
1028 }
1029 }
1030 }
1031 }
1032 }
1033 }
1034 }
1035 }
1036 }
1037 }
1038 }
1039 }
1040 }
1041 }
1042 }
1043 }
1044 }
1045 }
1046 }
1047 }
1048 }
1049 }
1050 }
1051 }
1052 }
1053 }
1054 }
1055 }
1056 }
1057 }
1058 }
1059 }
1060 }
1061 }
1062 }
1063 }
1064 }
1065 }
1066 }
1067 }
1068 }
1069 }
1070 }
1071 }
1072 }
1073 }
1074 }
1075 }
1076 }
1077 }
1078 }
1079 }
1080 }
1081 }
1082 }
1083 }
1084 }
1085 }
1086 }
1087 }
1088 }
1089 }
1090 }
1091 }
1092 }
1093 }
1094 }
1095 }
1096 }
1097 }
1098 }
1099 }
1100 }
1101 }
1102 }
1103 }
1104 }
1105 }
1106 }
1107 }
1108 }
1109 }
1110 }
1111 }
1112 }
1113 }
1114 }
1115 }
1116 }
1117 }
1118 }
1119 }
1120 }
1121 }
1122 }
1123 }
1124 }
1125 }
1126 }
1127 }
1128 }
1129 }
1130 }
1131 }
1132 }
1133 }
1134 }
1135 }
1136 }
1137 }
1138 }
1139 }
1140 }
1141 }
1142 }
1143 }
1144 }
1145 }
1146 }
1147 }
1148 }
1149 }
1150 }
1151 }
1152 }
1153 }
1154 }
1155 }
1156 }
1157 }
1158 }
1159 }
1160 }
1161 }
1162 }
1163 }
1164 }
1165 }
1166 }
1167 }
1168 }
1169 }
1170 }
1171 }
1172 }
1173 }
1174 }
1175 }
1176 }
1177 }
1178 }
1179 }
1180 }
1181 }
1182 }
1183 }
1184 }
1185 }
1186 }
1187 }
1188 }
1189 }
1190 }
1191 }
1192 }
1193 }
1194 }
1195 }
1196 }
1197 }
1198 }
1199 }
1200 }
1201 }
1202 }
1203 }
1204 }
1205 }
1206 }
1207 }
1208 }
1209 }
1210 }
1211 }
1212 }
1213 }
1214 }
1215 }
1216 }
1217 }
1218 }
1219 }
1220 }
1221 }
1222 }
1223 }
1224 }
1225 }
1226 }
1227 }
1228 }
1229 }
1230 }
1231 }
1232 }
1233 }
1234 }
1235 }
1236 }
1237 }
1238 }
1239 }
1240 }
1241 }
1242 }
1243 }
1244 }
1245 }
1246 }
1247 }
1248 }
1249 }
1250 }
1251 }
1252 }
1253 }
1254 }
1255 }
1256 }
1257 }
1258 }
1259 }
1260 }
1261 }
1262 }
1263 }
1264 }
1265 }
1266 }
1267 }
1268 }
1269 }
1270 }
1271 }
1272 }
1273 }
1274 }
1275 }
1276 }
1277 }
1278 }
1279 }
1280 }
1281 }
1282 }
1283 }
1284 }
1285 }
1286 }
1287 }
1288 }
1289 }
1290 }
1291 }
1292 }
1293 }
1294 }
1295 }
1296 }
1297 }
1298 }
1299 }
1300 }
1301 }
1302 }
1303 }
1304 }
1305 }
1306 }
1307 }
1308 }
1309 }
1310 }
1311 }
1312 }
1313 }
1314 }
1315 }
1316 }
1317 }
1318 }
1319 }
1320 }
1321 }
1322 }
1323 }
1324 }
1325 }
1326 }
1327 }
1328 }
1329 }
1330 }
1331 }
1332 }
1333 }
1334 }
1335 }
1336 }
1337 }
1338 }
1339 }
1340 }
1341 }
1342 }
1343 }
1344 }
1345 }
1346 }
1347 }
1348 }
1349 }
1350 }
1351 }
1352 }
1353 }
1354 }
1355 }
1356 }
1357 }
1358 }
1359 }
1360 }
1361 }
1362 }
1363 }
1364 }
1365 }
1366 }
1367 }
1368 }
1369 }
1370 }
1371 }
1372 }
1373 }
1374 }
1375 }
1376 }
1377 }
1378 }
1379 }
1380 }
1381 }
1382 }
1383 }
1384 }
1385 }
1386 }
1387 }
1388 }
1389 }
1390 }
1391 }
1392 }
1393 }
1394 }
1395 }
1396 }
1397 }
1398 }
1399 }
1400 }
1401 }
1402 }
1403 }
1404 }
1405 }
1406 }
1407 }
1408 }
1409 }
1410 }
1411 }
1412 }
1413 }
1414 }
1415 }
1416 }
1417 }
1418 }
1419 }
1420 }
1421 }
1422 }
1423 }
1424 }
1425 }
1426 }
1427 }
1428 }
1429 }
1430 }
1431 }
1432 }
1433 }
1434 }
1435 }
1436 }
1437 }
1438 }
1439 }
1440 }
1441 }
1442 }
1443 }
1444 }
1445 }
1446 }
1447 }
1448 }
1449 }
1450 }
1451 }
1452 }
1453 }
1454 }
1455 }
1456 }
1457 }
1458 }
1459 }
1460 }
1461 }
1462 }
1463 }
1464 }
1465 }
1466 }
1467 }
1468 }
1469 }
1470 }
1471 }
1472 }
1473 }
1474 }
1475 }
1476 }
1477 }
1478 }
1479 }
1480 }
1481 }
1482 }
1483 }
1484 }
1485 }
1486 }
1487 }
1488 }
1489 }
1490 }
1491 }
1492 }
1493 }
1494 }
1495 }
1496 }
1497 }
1498 }
1499 }
1500 }
1501 }
1502 }
1503 }
1504 }
1505 }
1506 }
1507 }
1508 }
1509 }
1510 }
1511 }
1512 }
1513 }
1514 }
1515 }
1516 }
1517 }
1518 }
1519 }
1520 }
1521 }
1522 }
1523 }
1524 }
1525 }
1526 }
1527 }
1528 }
1529 }
1530 }
1531 }
1532 }
1533 }
1534 }
1535 }
1536 }
1537 }
1538 }
1539 }
1540 }
1541 }
1542 }
1543 }
1544 }
1545 }
1546 }
1547 }
1548 }
1549 }
1550 }
1551 }
1552 }
1553 }
1554 }
1555 }
1556 }
1557 }
1558 }
1559 }
1560 }
1561 }
1562 }
1563 }
1564 }
1565 }
1566 }
1567 }
1568 }
1569 }
1570 }
1571 }
1572 }
1573 }
1574 }
1575 }
1576 }
1577 }
1578 }
1579 }
1580 }
1581 }
1582 }
1583 }
1584 }
1585 }
1586 }
1587 }
1588 }
1589 }
1590 }
1591 }
1592 }
1593 }
1594 }
1595 }
1596 }
1597 }
1598 }
1599 }
1600 }
1601 }
1602 }
1603 }
1604 }
1605 }
1606 }
1607 }
1608 }
1609 }
1610 }
1611 }
1612 }
1613 }
1614 }
1615 }
1616 }
1617 }
1618 }
1619 }
1620 }
1621 }
1622 }
1623 }
1624 }
1625 }
1626 }
1627 }
1628 }
1629 }
1630 }
1631 }
1632 }
1633 }
1634 }
1635 }
1636 }
1637 }
1638 }
1639 }
1640 }
1641 }
1642 }
1643 }
1644 }
1645 }
1646 }
1647 }
1648 }
1649 }
1650 }
1651 }
1652 }
1653 }
1654 }
1655 }
1656 }
1657 }
1658 }
1659 }
1660 }
1661 }
1662 }
1663 }
1664 }
1665 }
1666 }
1667 }
1668 }
1669 }
1670 }
1671 }
1672 }
1673 }
1674 }
1675 }
1676 }
1677 }
1678 }
1679 }
1680 }
1681 }
1682 }
1683 }
1684 }
1685 }
1686 }
1687 }
1688 }
1689 }
1690 }
1691 }
1692 }
1693 }
1694 }
1695 }
1696 }
1697 }
1698 }
1699 }
1700 }
1701 }
1702 }
1703 }
1704 }
1705 }
1706 }
1707 }
1708 }
1709 }
1710 }
1711 }
1712 }
1713 }
1714 }
1715 }
1716 }
1717 }
1718 }
1719 }
1720 }
1721 }
1722 }
1723 }
1724 }
1725 }
1726 }
1727 }
1728 }
1729 }
1730 }
1731 }
1732 }
1733 }
1734 }
1735 }
1736 }
1737 }
1738 }
1739 }
1740 }
1741 }
1742 }
1743 }
1744 }
1745 }
1746 }
1747 }
1748 }
1749 }
1750 }
1751 }
1752 }
1753 }
1754 }
1755 }
1756 }
1757 }
1758 }
1759 }
1760 }
1761 }
1762 }
1763 }
1764 }
1765 }
1766 }
1767 }
1768 }
1769 }
1770 }
1771 }
1772 }
1773 }
1774 }
1775 }
1776 }
1777 }
1778 }
1779 }
1780 }
1781 }
1782 }
1783 }
1784 }
1785 }
1786 }
1787 }
1788 }
1789 }
1790 }
1791 }
1792 }
1793 }
1794 }
1795 }
1796 }
1797 }
1798 }
1799 }
1800 }
1801 }
1802 }
1803 }
1804 }
1805 }
1806 }
1807 }
1808 }
1809 }
1810 }
1811 }
1812 }
1813 }
1814 }
1815 }
1816 }
1817 }
1818 }
1819 }
1820 }
1821 }
1822 }
1823 }
1824 }
1825 }
1826 }
1827 }
1828 }
1829 }
1830 }
1831 }
1832 }
1833 }
1834 }
1835 }
1836 }
1837 }
1838 }
1839 }
1840 }
1841 }
1842 }
1843 }
1844 }
1845 }
1846 }
1847 }
1848 }
1849 }
1850 }
1851 }
1852 }
1853 }
1854 }
1855 }
1856 }
1857 }
1858 }
1859 }
1860 }
1861 }
1862 }
1863 }
1864 }
1865 }
1866 }
1867 }
1868 }
1869 }
1870 }
1871 }
1872 }
1873 }
1874 }
1875 }
1876 }
1877 }
1878 }
1879 }
1880 }
1881 }
1882 }
1883 }
1884 }
1885 }
1886 }
1887 }
1888 }
1889 }
1890 }
1891 }
1892 }
1893 }
1894 }
1895 }
1896 }
1897 }
1898 }
1899 }
1900 }
1901 }
1902 }
1903 }
1904 }
1905 }
1906 }
1907 }
1908 }
1909 }
1910 }
1911 }
1912 }
1913 }
1914 }
1915 }
1916 }
1917 }
1918 }
1919 }
1920 }
1921 }
1922 }
1923 }
1924 }
1925 }
1926 }
1927 }
1928 }
1929 }
1930 }
1931 }
1932 }
1933 }
1934 }
1935 }
1936 }
1937 }
1938 }
1939 }
1940 }
1941 }
1942 }
1943 }
1944 }
1945 }
1946 }
1947 }
1948 }
1949 }
1950 }
1951 }
1952 }
1953 }
1954 }
1955 }
1956 }
1957 }
1958 }
1959 }
1960 }
1961 }
1962 }
1963 }
1964 }
1965 }
1966 }
1967 }
1968 }
1969 }
1970 }
1971 }
1972 }
1973 }
1974 }
1975 }
1976 }
1977 }
1978 }
1979 }
1980 }
1981 }
1982 }
1983 }
1984 }
1985 }
1986 }
1987 }
1988 }
1989 }
1990 }
1991 }
1992 }
1993 }
1994 }
1995 }
1996 }
1997 }
1998 }
1999 }
2000 }
2001 }
2002 }
2003 }
2004 }
2005 }
2006 }
2007 }
2008 }
2009 }
2010 }
2011 }
2012 }
2013 }
2014 }
2015 }
2016 }
2017 }
2018 }
2019 }
2020 }
2021 }
2022 }
2023 }
2024 }
2025 }
2026 }
2027 }
2028 }
2029 }
2030 }
2031 }
2032 }
2033 }
2034 }
2035 }
2036 }
2037 }
2038 }
2039 }
2040 }
2041 }
2042 }
2043 }
2044 }
2045 }
2046 }
2047 }
2048 }
2049 }
2050 }
2051 }
2052 }
2053 }
2054 }
2055 }
2056 }
2057 }
2058 }
2059 }
2060 }
2061 }
2062 }
2063 }
2064 }
2065 }
2066 }
2067 }
2068 }
2069 }
2070 }
2071 }
2072 }
2073 }
2074 }
2075 }
2076 }
2077 }
2078 }
2079 }
2080 }
2081 }
2082 }
2083 }
2084 }
2085 }
2086 }
2087 }
2088 }
2089 }
2090 }
2091 }
2092 }
2093 }
2094 }
2095 }
2096 }
2097 }
2098 }
2099 }
2100 }
2101 }
2102 }
2103 }
2104 }
2105 }
2106 }
2107 }
2108 }
2109 }
2110 }
2111 }
2112 }
2113 }
2114 }
2115 }
2116 }
2117 }
2118 }
2119 }
2120 }
2121 }
2122 }
2123 }
2124 }
2125 }
2126 }
2127 }
2128 }
2129 }
2
```

new/usr/src/uts/armv6/ml/cache.s

1

5396 Sat Feb 7 19:01:09 2015

new/usr/src/uts/armv6/ml/cache.s

armv6: bit 2 (0x4) enables the dcache

This fixes a pretty simple typo. Sadly, this still isn't enough to get
bcm2835 past mutex_enter.

unchanged portion omitted

```
173     ENTRY(armv6_dcache_enable)
174     mrc     p15, 0, r0, c1, c0, 0
175     orr    r0, #0x4
175     orr    r0, #0x2
176     mcr     p15, 0, r0, c1, c0, 0
177     SET_SIZE(armv6_dcache_enable)
```

unchanged portion omitted

```
185     ENTRY(armv6_dcache_disable)
186     mrc     p15, 0, r0, c1, c0, 0
187     bic    r0, #0x4
187     bic    r0, #0x2
188     mcr     p15, 0, r0, c1, c0, 0
189     SET_SIZE(armv6_dcache_disable)
```

unchanged portion omitted

new/usr/src/uts/armv6/ml/glocore.s

1

```
*****
4937 Sat Feb 7 19:01:09 2015
new/usr/src/uts/armv6/ml/glocore.s
armv6: simplify highvecs enabling code
Use the barrel shifter, Luke.
armv6: bcm2835 & qvpb have nearly identical locore_start
It makes sense to common-ize _start for all armv6 machines. They will all
have to do the same basic setup. If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
armv6: bit 2 (0x4) enables the dcache
This fixes a pretty simple typo. Sadly, this still isn't enough to get
bcm2835 past mutex_enter.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */
12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 */
17 #include <sys/asm_linkage.h>
18 #include <sys/machparam.h>
19 #include <sys/cpu_asm.h>
21 #include "assym.h"
23 /*
24  * Every story needs a beginning. This is ours.
25 */
23 #if defined(__lint)
27 /*
28  * Each of the different machines has its own locore.s to take care of getting
29  * the machine specific setup done. Just before jumping into fakebop the
30  * first time, we call this machine specific code.
31 */
25 #endif
33 /*
34  * We are in a primordial world here. The loader is going to come along and
35  * boot us at _start. As we've started the world, we also need to set up a
36  * few things about us, for example our stack pointer. To help us out, it's
37  * useful to remember what the loader set up for us:
38  *
39  * - unaligned access are allowed (A = 0, U = 1)
40  * - virtual memory is enabled
41  * - we (unix) are mapped right were we want to be
42  * - a UART has been enabled & any memory mapped registers have been 1:1
43  *   mapped
44  * - ATAGs have been updated to tell us what the mappings are
45  * - I/D LI caches have been enabled
28  * Each of the different machines has its own locore.s to take care of getting
29  * us into fakebop for the first time. After that, they all return here to a
30  * generic locore to take us into mlsetup and then to main forever more.
46 */
48 /*
```

new/usr/src/uts/armv6/ml/glocore.s

2

```
49  * External globals
50  */
51 .globl _locore_start
52 .globl mlsetup
53 .globl sysp
54 .globl bootops
55 .globl bootopsp
56 .globl t0
58 .data
59 .comm t0stack, DEFAULTSTKSZ, 32
60 .comm t0, 4094, 32
63 /*
64  * Recall that _start is the traditional entry point for an ELF binary.
65 */
66 ENTRY(_start)
67     ldr    sp, =t0stack
68     ldr    r4, =DEFAULTSTKSZ
69     add    sp, r4
70     bic    sp, sp, #0xff
72 /*
73  * establish bogus stacks for exceptional CPU states, our exception
74  * code should never make use of these, and we want loud and violent
75  * failure should we accidentally try.
76 */
77     cps    #(CPU_MODE_UND)
78     mov    sp, #-1
79     cps    #(CPU_MODE_ABT)
80     mov    sp, #-1
81     cps    #(CPU_MODE_FIQ)
82     mov    sp, #-1
83     cps    #(CPU_MODE_IRQ)
84     mov    sp, #-1
85     cps    #(CPU_MODE_SVC)
87 /* Enable highvecs (moves the base of the exception vector) */
88     mrc    p15, 0, r3, c1, c0, 0
89     orr    r3, r3, #(1 << 13)
90     mcr    p15, 0, r3, c1, c0, 0
92 /* invoke machine specific setup */
93     bl    _mach_start
95     bl    _fakebop_start
96     SET_SIZE(_start)
99 #endif /* ! codereview */
100 #if defined(__lint)
102 /* ARGSUSED */
103 void
104 _locore_start(struct boot_syscalls *sysp, struct bootops *bop)
105 {}
107 #else /* __lint */
109 /*
110  * We got here from _kobj_init() via exitto(). We have a few different
111  * tasks that we need to take care of before we hop into mlsetup and
112  * then main. We're never going back so we shouldn't feel compelled to
113  * preserve any registers.
114  */
```

```

47  * o Enable unaligned access
115 * o Enable our I/D-caches
116 * o Save the boot syscalls and bootops for later
117 * o Set up our stack to be the real stack of t0stack.
118 * o Save t0 as curthread
119 * o Set up a struct REGS for mlsetup
120 * o Make sure that we're 8 byte aligned for the call
121 */

123 ENTRY(_locore_start)

126 /*
127 * We've been running in t0stack anyway, up to this point, but
128 * _locore_start represents what is in effect a fresh start in the
129 * real kernel -- We'll never return back through here.
130 *
131 * So reclaim those few bytes
132 */
133 ldr    sp, =t0stack
134 ldr    r4, =(DEFAULTSTKSZ - REGSIZE)
135 add    sp, r4
136 bic    sp, sp, #0xff

138 /*
139 * Save flags and arguments for potential debugging
140 */
141 str    r0, [sp, #REGOFF_R0]
142 str    r1, [sp, #REGOFF_R1]
143 str    r2, [sp, #REGOFF_R2]
144 str    r3, [sp, #REGOFF_R3]
145 mrs    r4, CPSR
146 str    r4, [sp, #REGOFF_CPSR]

148 /*
149 * Save back the bootops and boot_syscalls.
150 */
151 ldr    r2, =sysp
152 str    r0, [r2]
153 ldr    r2, =bootops
154 str    r1, [r2]
155 ldr    r2, =bootopsp
156 ldr    r2, [r2]
157 str    r1, [r2]

159 /*
160 * Set up our curthread pointer
161 */
162 ldr    r0, =t0
163 mcr    p15, 0, r0, c13, c0, 4

165 /*
166 * Go ahead now and enable the L1 I/D caches.
99  * Go ahead now and enable unaligned access, the L1 I/D caches.
100 *
101 * Bit 2 is for the D cache
102 * Bit 12 is for the I cache
103 * Bit 22 is for unaligned access
167 */
168 mrc    p15, 0, r0, c1, c0, 0
169 orr    r0, #0x04 /* D-cache */
170 orr    r0, #0x1000 /* I-cache */
106 orr    r0, #0x02
107 orr    r0, #0x1000
108 orr    r0, #0x400000
171 mcr    p15, 0, r0, c1, c0, 0

```

```

173 /*
174 * mlsetup() takes the struct regs as an argument. main doesn't take
175 * any and should never return. Currently, we have an 8-byte aligned
176 * stack. We want to push a zero frame pointer to terminate any
177 * stack walking, but that would cause us to end up with only a
178 * 4-byte aligned stack. So, to keep things nice and correct, we
179 * push a zero value twice - it's similar to a typical function
180 * entry:
181 *     push { r9, lr }
182 */
183 mov    r9, #0
184 push  { r9 } /* link register */
185 push  { r9 } /* frame pointer */
186 mov    r0, sp
187 bl    mlsetup
188 bl    main
189 /* NOTREACHED */
190 ldr    r0, =__return_from_main
191 ldr    r0, [r0]
192 bl    panic
193 SET_SIZE(_locore_start)

195 __return_from_main:
196 .string "main() returned"
197 #endif /* __lint */

199 ENTRY(arm_reg_read)
200 ldr    r0, [r0]
201 bx    lr
202 SET_SIZE(arm_reg_read)

204 ENTRY(arm_reg_write)
205 str    r1, [r0]
206 bx    lr
207 SET_SIZE(arm_reg_write)
208 #endif /* ! codereview */

```

```

*****
25287 Sat Feb 7 19:01:09 2015
new/usr/src/uts/armv6/os/fakebop.c
armv6: bop_panic should hexdump the stack
It's the little things that make debugging easier.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright (c) 2014 Joyent, Inc. All rights reserved.
14  * Copyright (c) 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 */

17 /*
18  * Just like in i86pc, we too get the joys of mimicking the SPARC boot system.
19 */

21 #include <sys/types.h>
22 #include <sys/param.h>
23 #include <sys/bootconf.h>
24 #include <sys/bootsvcs.h>
25 #include <sys/boot_console.h>
26 #include <sys/atag.h>
27 #include <sys/varargs.h>
28 #include <sys/cmn_err.h>
29 #include <sys/sysmacros.h>
30 #include <sys/system.h>
31 #include <sys/ctype.h>
32 #include <sys/bootstat.h>
33 #include <sys/privregs.h>
34 #include <sys/cpu_asm.h>
35 #include <sys/boot_mmu.h>
36 #include <sys/elf.h>
37 #include <sys/archsystem.h>
38 #endif /* ! codereview */

40 static bootops_t bootop;

42 /*
43  * Debugging help
44  */
45 static int fakebop_prop_debug = 0;
46 static int fakebop_alloc_debug = 0;
47 static int fakebop_atag_debug = 0;

49 static uint_t kbm_debug = 1;
50 #define DBG_MSG(x)    { if (kbm_debug) bcons_puts(x); bcons_puts("\n"); }
51 #define BUFFERSIZE   256
52 static char buffer[BUFFERSIZE];

54 /*
55  * fakebop memory allocations scheme
56  *
57  * It's a virtual world out there. The loader thankfully tells us all the areas
58  * that it has mapped for us and it also tells us about the page table arena --
59  * a set of addresses that have already been set aside for us. We have two
60  * different kinds of allocations to worry about:

```

```

61  *
62  *   o Those that specify a particular vaddr
63  *   o Those that do not specify a particular vaddr
64  *
65  * Those that do not specify a particular vaddr will come out of our scratch
66  * space which is a fixed size arena of 16 MB (FAKEBOP_ALLOC_SIZE) that we set
67  * aside at the beginning of the allocator. If we end up running out of that
68  * then we'll go ahead and figure out a slightly larger area to worry about.
69  *
70  * Now, for those that do specify a particular vaddr we'll allocate more
71  * physical address space for it. The loader set aside enough L2 page tables for
72  * us that we'll go ahead and use the next 4k aligned address.
73  */
74 #define FAKEBOP_ALLOC_SIZE    (16 * 1024 * 1024)

76 static size_t bop_alloc_scratch_size;
77 static uintptr_t bop_alloc_scratch_next;      /* Next scratch address */
78 static uintptr_t bop_alloc_scratch_last;     /* Last scratch address */

80 static uintptr_t bop_alloc_pnext;           /* Next paddr */
81 static uintptr_t bop_alloc_plast;         /* cross this paddr and panic */

83 #define BI_HAS_RAMDISK    0x1

85 /*
86  * TODO Generalize this
87  * This is the set of information tha we want to gather from the various atag
88  * headers. This is simple and naive and will need to evolve as we have
89  * additional boards beyond just the RPi.
90  */
91 typedef struct bootinfo {
92     uint_t      bi_flags;
93     char        *bi_cmdline;
94     uint32_t    bi_ramdisk;
95     uint32_t    bi_ramsize;
96 } bootinfo_t;

98 static bootinfo_t bootinfo;      /* Simple set of boot information */

100 static struct boot_syscalls bop_syp = {
101     bcons_getchar,
102     bcons_putchar,
103     bcons_ischar,
104 };

106 /*
107  * stuff to store/report/manipulate boot property settings.
108  */
109 typedef struct bootprop {
110     struct bootprop *bp_next;
111     char *bp_name;
112     uint_t bp_vlen;
113     char *bp_value;
114 } bootprop_t;

116 static bootprop_t *bprops = NULL;

118 static void
119 hexdump_stack()
120 {
121     extern char t0stack[];

123     uint8_t *start = (uint8_t *)t0stack;
124     uint8_t *end = start + DEFAULTSTKSZ;
125     uint8_t *ptr;
126     int i;

```

```
128     bop_printf(NULL, "stack (fp = %x):\n", getfp());
130     ptr = (uint8_t *) (getfp() & ~0xf);
131     if (ptr <= start || ptr >= end)
132         ptr = start;
134     while (ptr < end) {
135         uint32_t *tmp = (uint32_t *)ptr;
137         bop_printf(NULL, "%p: %08x %08x %08x %08x\n", ptr,
138             tmp[0], tmp[1], tmp[2], tmp[3]);
140         ptr += 16;
141     }
143 }
145 #endif /* ! codereview */
146 void
147 bop_panic(const char *msg)
148 {
149     bop_printf(NULL, "ARM bop_panic:\n%s\n", msg);
151     hexdump_stack();
153     bop_printf(NULL, "Spinning Forever...", msg);
154     bop_printf(NULL, "ARM bop_panic:\n%s\nSpinning Forever...", msg);
155     for (;;)
156     }
    unchanged_portion_omitted
```

```

*****
697 Sat Feb 7 19:01:09 2015
new/usr/src/uts/armv6/qvpb/ml/locore.s
armv6: bcm2835 & qvpb have nearly identical locore_start
It makes sense to common-ize _start for all armv6 machines. They will all
have to do the same basic setup. If there is any machine specific setup
they need to do, they can do so in the new _mach_start function.
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2013 (c) Joyent, Inc. All rights reserved.
14  * Copyright 2015 (c) Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
15 #endif /* !codereview */
16 */

18 #include <sys/asm_linkage.h>
19 #include <sys/machparam.h>
20 #include <sys/cpu_asm.h>

22     ENTRY(_mach_start)
23     /* nothing to do */
24     bx    r14
25     SET_SIZE(_mach_start)
14 /*
15  * Every story needs a beginning. This is ours.
16 */

18 /*
19  * We are in a primordial world here. The BMC2835 is going to come along and
20  * boot us at _start. Normally we would go ahead and use a main() function, but
21  * for now, we'll do that ourselves. As we've started the world, we also need to
22  * set up a few things about us, for example our stack pointer. To help us out,
23  * it's useful to remember the rough memory map. Remember, this is for physcial
24  * addresses. There is no virtual memory here. These sizes are often manipulated
25  * by the 'configuration' in the bootloader.
26  *
27  * +-----+ <---- Max physical memory
28  * |       |
29  * |       |
30  * |       |
31  * +-----+
32  * |       |
33  * |   I/O  |
34  * | Peripherals |
35  * |       |
36  * +-----+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
37  * |       |
38  * |   Main |
39  * | Memory |
40  * |       |
41  * +-----+ <---- Top of SDRAM
42  * |       |
43  * |   VC   |
44  * | SDRAM  |
45  * |       |
46  * +-----+ <---- Split determined by bootloader config

```

```

47 * |
48 * |         ARM         |
49 * |         SDRAM      |
50 * | |
51 * | +-----+ <---- Bottom of physical memory 0x00000000
52 *
53 * With the Raspberry Pi Model B, we have 512 MB of SDRAM. That means we have a
54 * range of addresses from [0, 0x20000000). If we assume that the minimum amount
55 * of DRAM is given to the GPU - 32 MB, that means we really have the following
56 * range: [0, 0x1e000000).
57 *
58 * By default, this binary will be loaded into 0x8000. For now, that means we
59 * will set our initial stack to 0x10000000.
60 */

62 /*
63  * Recall that _start is the traditional entry point for an ELF binary.
64 */
65     ENTRY(_start)
66     ldr sp, =t0stack
67     ldr r4, =DEFAULTSTKSZ
68     add sp, r4
69     bic sp, sp, #0xff

71     /*
72     * establish bogus stacks for exceptional CPU states, our exception
73     * code should never make use of these, and we want loud and violent
74     * failure should we accidentally try.
75     */
76     cps #(CPU_MODE_UND)
77     mov sp, #-1
78     cps #(CPU_MODE_ABT)
79     mov sp, #-1
80     cps #(CPU_MODE_FIQ)
81     mov sp, #-1
82     cps #(CPU_MODE_IRQ)
83     mov sp, #-1
84     cps #(CPU_MODE_SVC)

86     /* Enable highvecs (moves the base of the exception vector) */
87     mrc    p15, 0, r3, c1, c0, 0
88     mov    r4, #1
89     lsl   r4, r4, #13
90     orr   r3, r3, r4
91     mcr   p15, 0, r3, c1, c0, 0

93     bl _fakebop_start
94     SET_SIZE(_start)

96     ENTRY(arm_reg_read)
97     ldr r0, [r0]
98     bx lr
99     SET_SIZE(arm_reg_read)

101    ENTRY(arm_reg_write)
102    str r1, [r0]
103    bx lr
104    SET_SIZE(arm_reg_write)

```