```
**********************************************************
   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;
1279         arm_reg_t rn, rd, rm;
1280         int lbit;
1281         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
**********************************************************
```

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

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

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

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

STUB(bcmp)
STUB(kdi_flush_caches)
STUB(kobj_text_alloc)
STUB(kdi_range_is_nontoxic)
STUB(dcache_flushall)
STUB(SHA1Final)
STUB(kdi_pwrite)
STUB(SHA1Update)
STUB(hat_unload)
STUB(stubs_base)
STUB(rw_exit)
STUB(hat_getpfnum)
STUB(hat_devload)
STUB(kdi_vtop)
STUB(stubs_end)
STUB(SHA1Init)
STUB(rw_enter)
STUB(kobj_texthole_free)
STUB(kdi_pread)
STUB(kobj_vmem_init)
STUB(cpr)
STUB(acct)
STUB(bind)
STUB(recv)
STUB(send)
STUB(spl0)
STUB(spl7)
STUB(spl8)
STUB(splx)
STUB(audit_init_module)
STUB(i_ddi_intr_ops)
STUB(dcopy_cmd_alloc)
STUB(impl_acc_hdl_free)
STUB(plat_hold_page)
STUB(check_status)
STUB(audit_symlink_create)
STUB(hat_page_clrattr)
STUB(copyinstr_noerr)
STUB(audit)
```



```
STUB(cas32)
STUB(cas64)
STUB(copyoutstr)
STUB(getfp)
STUB(htonl)
STUB(htons)
STUB(indir)
STUB(idmap_reg_dh)
STUB(kcopy)
STUB(kzero)
STUB(nosys)
STUB(ntohl)
STUB(ntohs)
STUB(fifo_vfastoff)
STUB(splhi)
STUB(ucopy)
STUB(uzero)
STUB(i_ddi_acc_clr_fault)
STUB(audit_setf)
STUB(audit_priv)
STUB(audit_exec)
STUB(audit_exit)
STUB(hat_leave_region)
STUB(door_ki_upcall)
STUB(unset_idle_cpu)
STUB(thread_onproc)
STUB(set_proc_ast)
STUB(ddi_rep_put64)
STUB(ddi_rep_put32)
STUB(ddi_rep_put16)
STUB(boot_virt_alloc)
STUB(ddi_rep_get64)
STUB(ddi_rep_get32)
STUB(ddi_rep_get16)
STUB(i_ddi_map_fault)
STUB(lwp_stk_cache_init)
STUB(dtrace_interrupt_enable)
STUB(ftrace_interrupt_enable)
STUB(xcopyout_nta)
STUB(hat_pagesync)
STUB(console_enter)
STUB(spec_snode_walk)
STUB(audit_chdirec)
STUB(prinvalidate)
STUB(lock_clear)
STUB(ka_init)
STUB(loadable_syscall)
STUB(sockconfig)
STUB(fuword8_noerr)
STUB(lwp_detach_brand_hdlrs)
STUB(valid_va_range_aligned)
STUB(lwp_forkregs)
STUB(devfs_devpolicy)
STUB(hat_stats_disable)
STUB(pr_free_watched_pages)
STUB(install_utrap)
STUB(dtrace_membar_consumer)
STUB(socket_sendmblk)
STUB(audit_symlink)
STUB(i_ddi_apply_range)
STUB(lock_clear_splx)
STUB(audit_strputmsg)
STUB(i_ddi_alloc_intr_phdl)
STUB(i_ddi_acc_set_fault)
STUB(clconf_get_nodeid)
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_closef)
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(getpcstack)
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(mdboot)
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(mdpreboot)
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_unfence_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_swtch_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_pgszcvec)
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_allstopped)
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_swtch_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(ftrace_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(prnostep)
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)
```

```
 523 STUB(arm_gettick)
```

****
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 */
```

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

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

BCM2835_OBJS =                  \
        bcm2835_bsmdep.o        \
        bcm2835_uart.o          \
#endif /* ! codereview */
        boot_console.o          \
        locore.o
        locore.o                \
        miniuart.o

BCM2835_LOADER_OBJS = \
        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.)
**********************************************************
```

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

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

#include <sys/elf.h>
#include <sys/atag.h>

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

#define UART_BASE               0x20201000
#define UART_DR                 0x0
#define UART_FR                 0x18
#define UART_IBRD               0x24
#define UART_FBRD               0x28
#define UART_LCRH               0x2c
#define UART_CR                 0x30
#define UART_ICR                0x44

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

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

#define UART_CR_UARTEN          0x001           /* uart enable */
#define UART_CR_TXE             0x100           /* TX enable */
#define UART_CR_RXE             0x200           /* RX enable */
#define AUX_BASE        0x20215000
#define AUX_ENABLES     0x4
#define AUX_MU_IO_REG   0x40
#define AUX_MU_IER_REG  0x44
#define AUX_MU_IIR_REG  0x48
#define AUX_MU_LCR_REG  0x4C
#define AUX_MU_MCR_REG  0x50
#define AUX_MU_LSR_REG  0x54
#define AUX_MU_CNTL_REG 0x60
```



```
#define AUX_MU_BAUD     0x68

#define AUX_MU_RX_READY 0x01
#define AUX_MU_TX_READY 0x20

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

#define GPIO_SEL_ALT0   0x4
#define GPIO_SEL_ALT5   0x2
#define GPIO_UART_MASK  0xfffc0fff
#define GPIO_UART_TX_SHIFT      12
#define GPIO_UART_RX_SHIFT      15

#define GPIO_PUD_DISABLE        0x0
#define GPIO_PUDCLK_UART        0x0000c000

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

        return *ptr;
}
______unchanged_portion_omitted_

/*
 * A simple nop
 */
static void
uart_nop(void)
bcm2835_miniuart_nop(void)
{
        __asm__ volatile("mov r0, r0\n" : : :);
}
______unchanged_portion_omitted_

void
fakeload_backend_init(void)
{
        uint32_t v;
        int i;

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

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

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

        /*
         * Enable 8-bit word length. External sources tell us the PRM is buggy
         * here and that even though bit 1 is reserved, we need to actually set
         * 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_miniuart_nop();
116         arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, GPIO_PUDCLK_UART);
117         for (i = 0; i < 150; i++)
118                 uart_nop();
138                 bcm2835_miniuart_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.
**********************************************************
```

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

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

#include <sys/asm_linkage.h>
#include <sys/machparam.h>
#include <sys/cpu_asm.h>

        ENTRY(_mach_start)
/*
 * Every story needs a beginning. This is ours.
 */

/*
 * We are in a primordial world here. The BMC2835 is going to come along and
 * boot us at _start. Normally we would go ahead and use a main() function, but
 * for now, we'll do that ourselves. As we've started the world, we also need to
 * set up a few things about us, for example our stack pointer. To help us out,
 * it's useful to remember the rough memory map. Remember, this is for physcial
 * addresses. There is no virtual memory here. These sizes are often manipulated
 * by the 'configuration' in the bootloader.
 *
 * +----------------+ <---- Max physical memory
 * |                |
 * |                |
 * |                |
 * +----------------+
 * |                |
 * |      I/O       |
 * |  Peripherals   |
 * |                |
 * +----------------+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
 * |                |
 * |     Main       |
 * |    Memory      |
 * |                |
 * +----------------+ <---- Top of SDRAM
 * |                |
 * |       VC       |
 * |     SDRAM      |
 * |                |
 * +----------------+ <---- Split determined by bootloader config
 * |                |
 * |      ARM       |
 * |     SDRAM      |
 * |                |
```



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

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr sp, =t0stack
        ldr r4, =DEFAULTSTKSZ
        add sp, r4
        bic sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps #(CPU_MODE_UND)
        mov sp, #-1
        cps #(CPU_MODE_ABT)
        mov sp, #-1
        cps #(CPU_MODE_FIQ)
        mov sp, #-1
        cps #(CPU_MODE_IRQ)
        mov sp, #-1
        cps #(CPU_MODE_SVC)

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

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

        bx      r14
        SET_SIZE(_mach_start)
        bl _fakebop_start
        SET_SIZE(_start)

        ENTRY(arm_reg_read)
        ldr r0, [r0]
        bx lr
        SET_SIZE(arm_reg_read)

        ENTRY(arm_reg_write)
        str r1, [r0]
        bx lr
        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.)
**********************************************************
```

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

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

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

#include "bcm2835_uart.h"

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

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

#define UART_BASE               0x20201000
#define UART_DR                 0x0
#define UART_FR                 0x18
#define UART_IBRD               0x24
#define UART_FBRD               0x28
#define UART_LCRH               0x2c
#define UART_CR                 0x30
#define UART_ICR                0x44

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

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

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


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



```
#define GPIO_FSEL1      0x4
#define GPIO_PUD        0x94
#define GPIO_PUDCLK0    0x98

#define GPIO_SEL_ALT0   0x4
#define GPIO_UART_MASK  0xfffc0fff
#define GPIO_UART_TX_SHIFT      12
#define GPIO_UART_RX_SHIFT      15

#define GPIO_PUD_DISABLE        0x0
#define GPIO_PUDCLK_UART        0x0000c000

/*
 * A simple nop
 */
static void
bcm2835_uart_nop(void)
{
        __asm__ volatile("mov r0, r0\n" : : :);
}

void
bcm2835_uart_init(void)
{
        uint32_t v;
        int i;

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

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

        arm_reg_write(GPIO_BASE + GPIO_PUD, GPIO_PUD_DISABLE);
        for (i = 0; i < 150; i++)
                bcm2835_uart_nop();
        arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, GPIO_PUDCLK_UART);
        for (i = 0; i < 150; i++)
                bcm2835_uart_nop();
        arm_reg_write(GPIO_BASE + GPIO_PUDCLK0, 0);

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

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

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

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

void
bcm2835_uart_putc(uint8_t c)
{
        while (arm_reg_read(UART_BASE + UART_FR) & UART_FR_TXFF)
                ;
        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 */
```

****
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 miniuart.)
****

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

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

#ifndef _BCM2835_UART_H
#define _BCM2835_UART_H

/*
 * Interface to the BCM2835's uart.
 */

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>

void bcm2835_uart_init(void);
void bcm2835_uart_putc(uint8_t);
uint8_t bcm2835_uart_getc(void);
int bcm2835_uart_isc(void);

#ifdef __cplusplus
}
#endif

#endif /* _BCM2835_UART_H */
#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.)
```
**********************************************************
```

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

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

/*
 * bcm2835 boot console implementation
 */

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

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

static bcm2835_consops_t consops;

/*
 * For now, we only support the real uart.
 * For now, we only support the mini uart.
 */
void
bcons_init(char *bstr)
{
        bcm2835_uart_init();
        consops.bco_putc = bcm2835_uart_putc;
        consops.bco_getc = bcm2835_uart_getc;
        consops.bco_isc = bcm2835_uart_isc;
        bcm2835_miniuart_init();
        consops.bco_putc = bcm2835_miniuart_putc;
        consops.bco_getc = bcm2835_miniuart_getc;
        consops.bco_isc = bcm2835_miniuart_isc;
}
```
_____***unchanged_portion_omitted_***

```
**********************************************************
    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_
```

```
**********************************************************
    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.
**********************************************************
```

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

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

#include <sys/asm_linkage.h>
#include <sys/machparam.h>
#include <sys/cpu_asm.h>

#include "assym.h"

/*
 * Every story needs a beginning. This is ours.
 */
#if defined(__lint)

/*
 * Each of the different machines has its own locore.s to take care of getting
 * the machine specific setup done.  Just before jumping into fakebop the
 * first time, we call this machine specific code.
 */
#endif

/*
 * We are in a primordial world here. The loader is going to come along and
 * boot us at _start. As we've started the world, we also need to set up a
 * few things about us, for example our stack pointer. To help us out, it's
 * useful to remember what the loader set up for us:
 *
 * - unaligned access are allowed (A = 0, U = 1)
 * - virtual memory is enabled
 *   - we (unix) are mapped right were we want to be
 *   - a UART has been enabled & any memory mapped registers have been 1:1
 *     mapped
 *   - ATAGs have been updated to tell us what the mappings are
 * - I/D L1 caches have been enabled
 * Each of the different machines has its own locore.s to take care of getting
 * us into fakebop for the first time. After that, they all return here to a
 * generic locore to take us into mlsetup and then to main forever more.
 */

        /*
```



```
         * External globals
         */
        .globl  _locore_start
        .globl  mlsetup
        .globl  sysp
        .globl  bootops
        .globl  bootopsp
        .globl  t0

        .data
        .comm   t0stack, DEFAULTSTKSZ, 32
        .comm   t0, 4094, 32

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr     sp, =t0stack
        ldr     r4, =DEFAULTSTKSZ
        add     sp, r4
        bic     sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps     #(CPU_MODE_UND)
        mov     sp, #-1
        cps     #(CPU_MODE_ABT)
        mov     sp, #-1
        cps     #(CPU_MODE_FIQ)
        mov     sp, #-1
        cps     #(CPU_MODE_IRQ)
        mov     sp, #-1
        cps     #(CPU_MODE_SVC)

        /* Enable highvecs (moves the base of the exception vector) */
        mrc     p15, 0, r3, c1, c0, 0
        orr     r3, r3, #(1 << 13)
        mcr     p15, 0, r3, c1, c0, 0

        /* invoke machine specific setup */
        bl      _mach_start

        bl      _fakebop_start
        SET_SIZE(_start)


#endif /* ! codereview */
#if defined(__lint)

/* ARGSUSED */
void
_locore_start(struct boot_syscalls *sysp, struct bootops *bop)
{}

#else   /* __lint */

        /*
         * We got here from _kobj_init() via exitto().  We have a few different
         * tasks that we need to take care of before we hop into mlsetup and
         * then main. We're never going back so we shouldn't feel compelled to
         * preserve any registers.
         *
```

```
 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.
**********************************************************
```

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

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

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

#include <sys/types.h>
#include <sys/param.h>
#include <sys/bootconf.h>
#include <sys/bootsvcs.h>
#include <sys/boot_console.h>
#include <sys/atag.h>
#include <sys/varargs.h>
#include <sys/cmn_err.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/ctype.h>
#include <sys/bootstat.h>
#include <sys/privregs.h>
#include <sys/cpu_asm.h>
#include <sys/boot_mmu.h>
#include <sys/elf.h>
#include <sys/archsystm.h>
#endif /* ! codereview */

static bootops_t bootop;

/*
 * Debugging help
 */
static int fakebop_prop_debug = 0;
static int fakebop_alloc_debug = 0;
static int fakebop_atag_debug = 0;

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

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



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

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

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

#define BI_HAS_RAMDISK  0x1

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

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

static struct boot_syscalls bop_sysp = {
        bcons_getchar,
        bcons_putchar,
        bcons_ischar,
};

/*
 * stuff to store/report/manipulate boot property settings.
 */
typedef struct bootprop {
        struct bootprop *bp_next;
        char *bp_name;
        uint_t bp_vlen;
        char *bp_value;
} bootprop_t;

static bootprop_t *bprops = NULL;

static void
hexdump_stack()
{
        extern char t0stack[];

        uint8_t *start = (uint8_t *)t0stack;
        uint8_t *end = start + DEFAULTSTKSZ;
        uint8_t *ptr;
        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);
 37         bop_printf(NULL, "ARM bop_panic:\n%s\nSpinning Forever...", msg);
154         for (;;)
155                 ;
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.
**********************************************************
```

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

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

#include <sys/asm_linkage.h>
#include <sys/machparam.h>
#include <sys/cpu_asm.h>

        ENTRY(_mach_start)
        /* nothing to do */
        bx      r14
        SET_SIZE(_mach_start)
/*
 * Every story needs a beginning. This is ours.
 */

/*
 * We are in a primordial world here. The BMC2835 is going to come along and
 * boot us at _start. Normally we would go ahead and use a main() function, but
 * for now, we'll do that ourselves. As we've started the world, we also need to
 * set up a few things about us, for example our stack pointer. To help us out,
 * it's useful to remember the rough memory map. Remember, this is for physcial
 * addresses. There is no virtual memory here. These sizes are often manipulated
 * by the 'configuration' in the bootloader.
 *
 * +----------------+ <---- Max physical memory
 * |                |
 * |                |
 * |                |
 * +----------------+
 * |                |
 * |      I/O       |
 * |  Peripherals   |
 * |                |
 * +----------------+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
 * |                |
 * |     Main       |
 * |    Memory      |
 * |                |
 * +----------------+ <---- Top of SDRAM
 * |                |
 * |       VC       |
 * |     SDRAM      |
 * |                |
 * +----------------+ <---- Split determined by bootloader config
```



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

/*
 * Recall that _start is the traditional entry point for an ELF binary.
 */
        ENTRY(_start)
        ldr sp, =t0stack
        ldr r4, =DEFAULTSTKSZ
        add sp, r4
        bic sp, sp, #0xff

        /*
         * establish bogus stacks for exceptional CPU states, our exception
         * code should never make use of these, and we want loud and violent
         * failure should we accidentally try.
         */
        cps #(CPU_MODE_UND)
        mov sp, #-1
        cps #(CPU_MODE_ABT)
        mov sp, #-1
        cps #(CPU_MODE_FIQ)
        mov sp, #-1
        cps #(CPU_MODE_IRQ)
        mov sp, #-1
        cps #(CPU_MODE_SVC)

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

        bl _fakebop_start
        SET_SIZE(_start)

        ENTRY(arm_reg_read)
        ldr r0, [r0]
        bx lr
        SET_SIZE(arm_reg_read)

        ENTRY(arm_reg_write)
        str r1, [r0]
        bx lr
        SET_SIZE(arm_reg_write)
```