

```

*****
811 Sat Feb 7 18:57:48 2015
new/usr/src/uts/armv6/bcm2835/ml/locore.s
armv6: bcm2835 & qvpb have nearly identical locore_start
It makes sense to common-size _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  * |-----+ <---- I/O base 0x20000000 (corresponds to 0x7E000000)
39  * |
40  * |   I/O
41  * |   Peripherals
42  * |
43  * |-----+ <---- Top of SDRAM
44  * |
45  * |   Main
46  * |   Memory
47  * |
48  * |-----+ <---- VC
49  * |
50  * |   SDRAM
51  * |
52  * |-----+ <---- Split determined by bootloader config
53  * |
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)

```

```

*****
4958 Sat Feb 7 18:57:48 2015
new/usr/src/uts/armv6/ml/glocore.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 (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 L1 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 /*
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 mov r4, #1
90 lsl r4, r4, #13
91 orr r3, r3, r4
92 mcr p15, 0, r3, c1, c0, 0

94 /* invoke machine specific setup */
95 bl _mach_start

97 bl _fakebop_start
98 SET_SIZE(_start)

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

104 /* ARGUSED */
105 void
106 _locore_start(struct boot_syscalls *sysp, struct bootops *bop)
107 {}

109 #else /* __lint */

111 /*
112  * We got here from _kobj_init() via exitto(). We have a few different
113  * tasks that we need to take care of before we hop into mlsetup and
114  * then main. We're never going back so we shouldn't feel compelled to
115  * preserve any registers.
116  *
117  * o Enable our I/D-caches
118  * o Save the boot syscalls and bootops for later
119  * o Set up our stack to be the real stack of t0stack.

```

```

120  * o Save t0 as curthread
121  * o Set up a struct REGS for mlsetup
122  * o Make sure that we're 8 byte aligned for the call
123  */
125  ENTRY(_locore_start)

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

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

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

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

167  /*
168  * Go ahead now and enable the L1 I/D caches.
169  */
170  mrc    p15, 0, r0, c1, c0, 0
171  orr    r0, #0x04 /* D-cache */
172  orr    r0, #0x1000 /* I-cache */
173  mcr    p15, 0, r0, c1, c0, 0

175  /*
176  * mlsetup() takes the struct regs as an argument. main doesn't take
177  * any and should never return. Currently, we have an 8-byte aligned
178  * stack. We want to push a zero frame pointer to terminate any
179  * stack walking, but that would cause us to end up with only a
180  * 4-byte aligned stack. So, to keep things nice and correct, we
181  * push a zero value twice - it's similar to a typical function
182  * entry:
183  *   push { r9, lr }
184  */
185  mov    r9, #0

```

```

186  push  { r9 } /* link register */
187  push  { r9 } /* frame pointer */
188  mov   r0, sp
189  bl   mlsetup
190  bl   main
191  /* NOTREACHED */
192  ldr   r0, =__return_from_main
193  ldr   r0, [r0]
194  bl   panic
195  SET_SIZE(_locore_start)

197  __return_from_main:
198  .string "main() returned"
199  #endif /* ! __lint */

201  ENTRY(arm_reg_read)
202  ldr r0, [r0]
203  bx lr
204  SET_SIZE(arm_reg_read)

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

```

```

*****
697 Sat Feb 7 18:57:48 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)

```