in g2-r4/v2014.01+gitAUTOINC+XXXXXXX/git directory

marvell_6220_board_r4-debug.elf (= tx.elf)

marvell_6220_r4_lowpower-debug.elf (= lpp.exe)

tx.exe => tx.bin

Question: tx.elf, tx.bin怎么生成的？

Answer:

in r4/oem/marvell/build/project_marvell_6220_r4_targets.mk

```
================================================================================================
raw: $(TARGET:.elf=_stripped.elf) raw_clean
	@echo Generating $(APPTARGET)$(RAW).bin
ifdef HSR_SRAM_START
	@echo Halt Self Refresh routine moved to $(HSR_SRAM_START)
else
	@echo Halt Self Refresh routine in default location LCM BASE
endif
	@$(ELFSTRIP) -S --strip-unneeded -Rbss -R.comment -R.stack -R.wtm_secure -o bin.tmp $<
	@$(OBJCOPY) -O binary bin.tmp $(TARGET:.elf=.bin)
	@-$(RM) bin.tmp
# Now lets refactor names to be descriptive.
	@-$(MV) $(APPTARGET).bin $(APPTARGET)$(RAW).bin
	@-$(MV) $(APPTARGET)_stripped.elf $(APPTARGET)$(RAW)_stripped.elf
#leave for generic debug	@-$(MV) $(APPTARGET).elf $(APPTARGET)$(RAW).elf
	@-$(MV) $(APPTARGET).elf.map $(APPTARGET)$(RAW).elf.map
	@-$(CP) $(APPTARGET)$(RAW).bin tx.bin
	@-$(CP) $(APPTARGET).elf tx.elf
	@echo option 1: Program SD on local:    sudo oem/marvell/$(PRODUCT_DIR)/buildtools/r4local.sh tx.bin /dev/sd--X
```

```
	@echo option 2: Copy R4 tool to target: scp oem/marvell/88pa6270_r4/buildtools/r4burn.sh  root@10.71.130.141:/usr/bin
	@echo "		Copy R4.bin to target:  scp tx.bin root@10.71.130.141:~/tx.bin"
	@echo "		Program SD on target:   r4burn.sh"
	@echo done.
```

==========

从上面的makefile，可看到如下tips:

把tx.bin写入到sd卡上

$ sudo r4local.sh tx.bin /dev/sdX (运行在host上)

而在target上

r4burn.sh (必须有tx.bin存在)

==========

```
walterzh$ /opt/arm-marvell-gcc/bin/arm-marvell-linux-gnueabi-readelf -l marvell_6220_board_r4-debug.elf

Elf file type is EXEC (Executable file)
Entry point 0x6000000
There are 3 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  EXIDX          0x0b18d0 0x060b17d0 0x060b17d0 0x02910 0x02910 R   0x4
  LOAD           0x000100 0x06000000 0x06000000 0xdf158 0x118990 RWE 0x100
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00     .ARM.exidx
   01     vectors text .test .ARM.extab .ARM.exidx data rodata bss icache_align
```

02

walterzh$ /opt/arm-marvell-gcc/bin/arm-marvell-linux-gnueabi-readelf -S marvell_6220_board_r4-debug.elf

There are 25 section headers, starting at offset 0x211f2c:

Section Headers:

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] vectors           PROGBITS        06000000 000100 000060 00  AX  0   0  4
  [ 2] text              PROGBITS        06000100 000200 0ad100 00  AX  0   0 256
  [ 3] .test             PROGBITS        060ad200 0ad300 000090 00  WA  0   0  4
  [ 4] .ARM.extab        PROGBITS        060ad290 0ad390 004540 00   A  0   0  4
  [ 5] .ARM.exidx        ARM_EXIDX       060b17d0 0b18d0 002910 00  AL  2   0  4
  [ 6] data              PROGBITS        060b40e0 0b41e0 002144 00  WA  0   0 32
  [ 7] rodata            PROGBITS        060b6224 0b6324 028f10 00   A  0   0 32
  [ 8] bss               NOBITS          060df158 0df258 039838 00  WA  0   0 32
  [ 9] .stack            PROGBITS        06118990 0df258 002110 00      0   0  4
  [10] .debug_info       PROGBITS        00000000 0e1368 0b0242 00      0   0  1
  [11] .debug_abbrev     PROGBITS        00000000 1915aa 017783 00      0   0  1
  [12] .debug_aranges    PROGBITS        00000000 1a8d30 0027c0 00      0   0  8
  [13] .debug_line       PROGBITS        00000000 1ab4f0 02bd24 00      0   0  1
  [14] .debug_str        PROGBITS        00000000 1d7214 023e06 00      0   0  1
  [15] .comment          PROGBITS        00000000 1fb01a 000469 00      0   0  1
  [16] .ARM.attributes   ARM_ATTRIBUTES  00000000 1fb483 000033 00      0   0  1
  [17] .debug_frame      PROGBITS        00000000 1fb4b8 0107d0 00      0   0  4
  [18] icache_align      PROGBITS        060df140 0df240 000018 00  AX  0   0  4
  [19] .debug_ranges     PROGBITS        00000000 20bc88 000140 00      0   0  8
  [20] .debug_loc        PROGBITS        00000000 20bdc8 0050c2 00      0   0  1
  [21] .debug_pubnames   PROGBITS        00000000 210e8a 000fae 00      0   0  1
  [22] .shstrtab         STRTAB          00000000 211e38 0000f2 00      0   0  1
  [23] .symtab           SYMTAB          00000000 212314 01f220 10     24 6207  4
  [24] .strtab           STRTAB          00000000 231534 00dc7b 00      0   0  1
```

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)

tx.bin运行在0x06000000开始的address space（physical address space）中。

in oem/marvell/build/project_marvell_6220_r4.mk

```
# where should the code live in memory
RTOS_MEM_START ?= 0x06000000  (96M)
RTOS_MEM_SIZE  ?= 0x00400000     (4M)
```

r4/oem/marvell/88pa6220_r4/buildtools/memory_map.ld

```
MEMORY
{
   /* NOTE - the ram line here is rewritten by the makefile before the linker
           script is used.  If you want to move the code, specify
           RTOS_MEM_START=... and RTOS_MEM_SIZE=... on the makeline */
   ram (rwx): ORIGIN = 0x0, LENGTH = 0x0
   squ0 (rw ) : ORIGIN = 0xD1000000, LENGTH = 32K  /* SQU0 32K - TBD*/
   squ1 (rw ) : ORIGIN = 0xD1008000, LENGTH = 32K  /* SQU1 32K - TBD*/
   squ2 (rw ) : ORIGIN = 0xD1010000, LENGTH = 32K  /* SQU2 32K - TBD*/
}
```

From R4 view, it will occupy the following physical address space.

[0x06000000, 0x06400000) --- from 96M to 100M

[0xD1000000, 0xD1018000) --- from 3344M to 3344.09375M

==========

From 0xD1000000 to 0xD1018000的96K SRAM是R4运行的场所（physical address），必须在Linux管理的空间中挖掉这一块。

在Linux中是通过“mmio-sram” driver来实现这个挖掉[0xD1000000, 0xD1018000)这个功能的，这样Linux下的code(无论是kernel code还是application)就不会访问这块空间了。

in pegmatite.dtsi

```
        squ: squ@d1000000 {
                compatible = "mmio-sram";
                reg = <0 0xd1000000 0 0x18000>;
                clocks = <&apbus_squ_clkgate>;

                #address-cells = <1>;
                #size-cells = <1>;
                ranges = <0 0 0xd1000000 0x18000>;

                smpboot-sram@0 {
                        compatible = "marvell,pegmatite-smpboot-sram";
                        /*
                         * Three words are required. Round up to 32 to keep
                         * alignment with mmio-sram chunk size. Perhaps mmio-sram
                         * should do this internally.
                         */
                        reg = <0x0 0x20>;
                };
        };

```

"marvell,pegmatite-smpboot-sram" is for arch/arm/mach-pegmatite/platsmp.c

CONFIG_SRAM=y

will enable drivers/misc/sram.c driver.

about "sram" driver, reference:

https://lwn.net/Articles/537728/

Documentation/devicetree/bindings/misc/sram.c

==========

[0x06000000, 0x06400000) --- from 96M to 100M

Question: 在Linux中怎么reserve呢？

Answer: Linux在初始化时接收到可用RAM就是缺少从96M到100M这4M空间的。

具体见《memblock allocator》note。

==========

lpp.exe的physical address space

in oem/marvell/88pa6220_r4/lowpower_v1/buildtools/memory_map_r4.ld

```
MEMORY
{
/*
 * Rev A LCM 96KB @ 0xD0FE_8000, aliased every 128KB. such to butt up against SQU memory.  keep this base 0xD0FE_8000.
 * Our vectors MUST be aligned on 64KB boundary.
 *     On rev A             data/bss/stack/rodata will be 32KB @ 0xD0FE_8000
 *     On rev A vectors/text                    will be 64KB @ 0xD0FF_0000
 *     D0FE0000     D0FE8000     D0FF0000     D0FF8000     D1000000
 *     A            B            C            D
 * GsA     XXXXXXXXXXXXXXXXDDDDDDDDDDDDDDDDVTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
 * V = vectors, X = not available, T = text, D = vars(data+bss+rodata+stack)
 */
   vars (rw)  : ORIGIN = 0xD0FE8000, LENGTH = 32K  /* LCM first 32 of 96K total */
   ram  (rwx) : ORIGIN = 0xD0FF0000, LENGTH = 64K  /* LCM last 64 of 96K total */
   squ0 (rw ) : ORIGIN = 0xD1000000, LENGTH = 32K  /* SQU0 32K - TBD*/
```

```
    squ1 (rw ) : ORIGIN = 0xD1008000, LENGTH = 32K  /* SQU1 32K - TBD*/
    squ2 (rw ) : ORIGIN = 0xD1010000, LENGTH = 32K  /* SQU2 32K - TBD*/
}
```

```
walterzh$ /opt/arm-marvell-gcc/bin/arm-marvell-linux-gnueabi-readelf -l marvell_6220_r4_lowpower-debug.elf

Elf file type is EXEC (Executable file)
Entry point 0xd0ff0000
There are 3 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x0000a0 0xd0fe8000 0xd0fe8000 0x02cd0 0x04c80 RW  0x20
  LOAD           0x002d80 0xd0ff0000 0xd0ff0000 0x0db28 0x0db28 R E 0x20
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00     data rodata bss
   01     vectors text
   02
```

IPS::LCM::SRAM0	0xD0FE8000	IPS::LCM::SRAM0
IPS::LCM::SRAM1	0xD0FF0000	IPS::LCM::SRAM1
IPS::LCM::SRAM2	0xD0FF8000	IPS::LCM::SRAM2

LCM::SRAM0作为lpp.exe运行的data segment (0xd0fe8000, 0xd0fe8000 + 32K)

LCM::SRAM1 and LCM::SRAM2作为lpp.exe运行的code segment (0xd0ff0000, 0xd0ff0000 + 64K)

这段地址空间在Linux中怎么reserve呢？