ARM Exception System Introduction (HXD8 example)

Alec Tsai

ARM Exceptions

- Exceptions are generated by internal and external sources to cause the processor to handle an event, such as an externally generated interrupt or an attempt to execute an undefined instruction.
- The processor state just before handling the exception must be preserved so that the original program can be resumed when the exception routine has completed.

ARM Exception Processing Modes

Exception type	Mode	Normal address	High vector address
Reset	Supervisor	0x00000000	0xFFFF0000
Undefined instructions	Undefined	0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor	0x00000008	0xFFFF0008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C	0xFFFF000C
Data Abort (data access memory abort)	Abort	0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (fast interrupt)	FIQ	0x0000001C	0xFFFF001C

ARM Interrupt Software Flow in U-boot

```
.globl _start
_start:     b       reset
            ldr         pc, _undefined_instruction
            ldr         pc, _software_interrupt
            ldr         pc, _prefetch_abort
            ldr         pc, _data_abort
            ldr         pc, _not_used
            ldr         pc, _irq
            ldr         pc, _fiq

_undefined_instruction:     .word undefined_instruction
_software_interrupt:        .word software_interrupt
_prefetch_abort:            .word prefetch_abort
_data_abort:                .word data_abort
_not_used:                  .word not_used
_irq:                       .word irq
_fiq:                       .word fiq
```

```
            .align      5
irq:
            get_irq_stack
            irq_save_user_regs
            bl          do_irq
            irq_restore_user_regs
```

```
void do_irq (struct pt_regs *pt_regs)
{
#if defined (CONFIG_USE_IRQ)
#if defined (ARM920_IRQ_CALLBACK)
            ARM920_IRQ_CALLBACK();
            return;
#elif defined (CONFIG_ARCH_INTEGRATOR)
            /* ASSUMED to be a timer interrupt  */
            /* Just clear it - count handled in */
            /* integratorap.c                   */
            *(volatile ulong *)(CFG_TIMERBASE + 0x0C) = 0;
#endif /* ARCH_INTEGRATOR */
#else
            printf ("interrupt request\n");
            show_regs (pt_regs);
            bad_mode ();
#endif
}
```

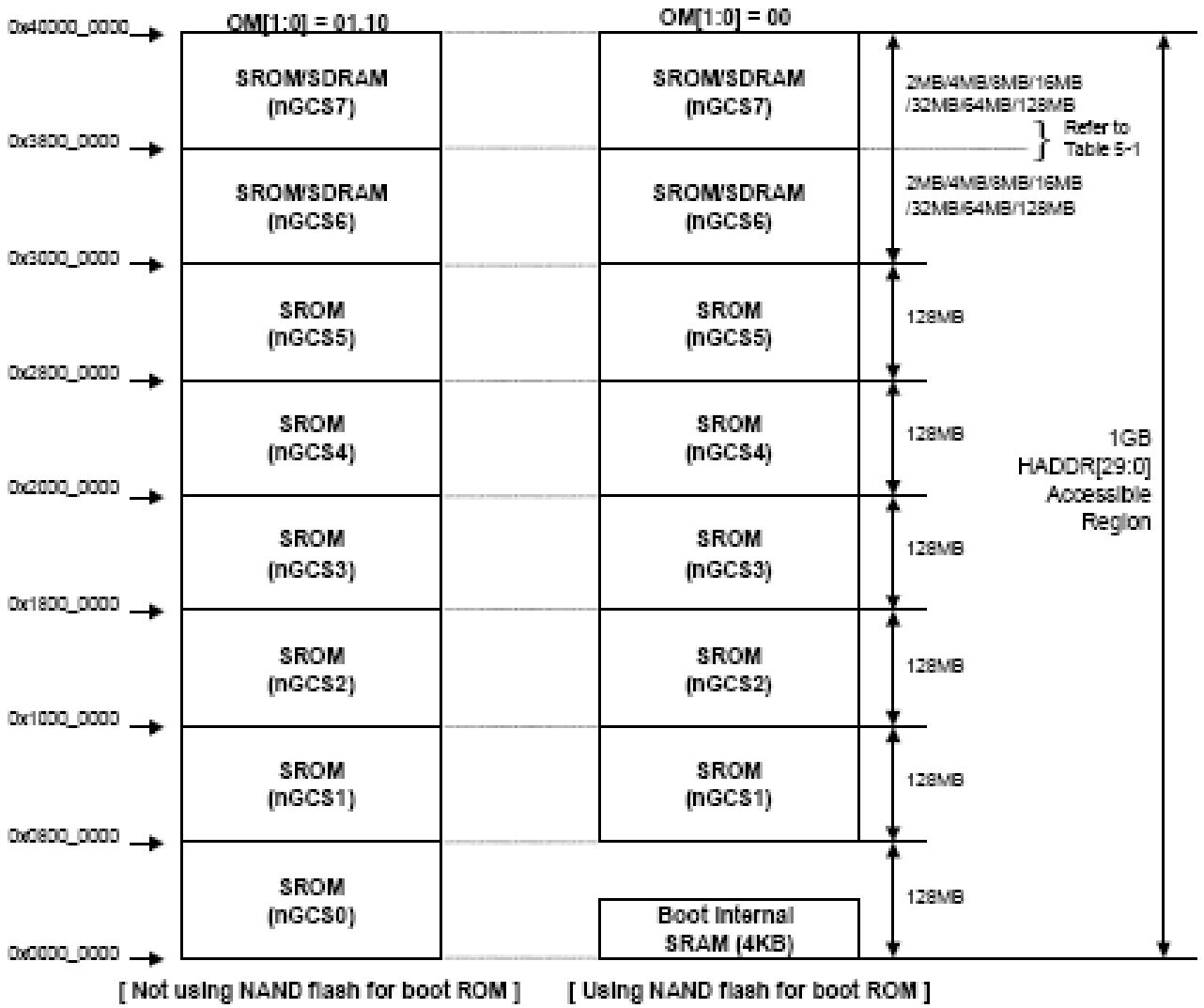
```
#define ARM920_IRQ_CALLBACK s3c2410_irq
```

```
#ifdef CONFIG_USE_IRQ
void s3c2410_irq(void)
{
            S3C24X0_INTERRUPT * irq = S3C24X0_GetBase_INTERRUPT();
            u_int32_t intpnd = irq->INTPND;

#ifdef CONFIG_USB_DEVICE
            if (intpnd & BIT_USBD) {
                        s3c2410_udc_irq();
                        irq->SRCPND = BIT_USBD;
                        irq->INTPND = BIT_USBD;
            }
#endif /* USB_DEVICE */
            else if (intpnd & BIT_ADC) {
                        Adc_or_TsAuto();
                        irq->SRCPND = BIT_ADC;
                        irq->INTPND = BIT_ADC;
            }

}
#endif /* USE_IRQ */
```

ARM Reset

- When the nRESET signal goes LOW, ARM920T abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.
- When nRESET goes HIGH again, ARM920T:
  - Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
  - Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
  - Forces the PC to fetch the next instruction from address 0x00.
  - Execution resumes in ARM state.

S3C2440A Memory Map after Reset



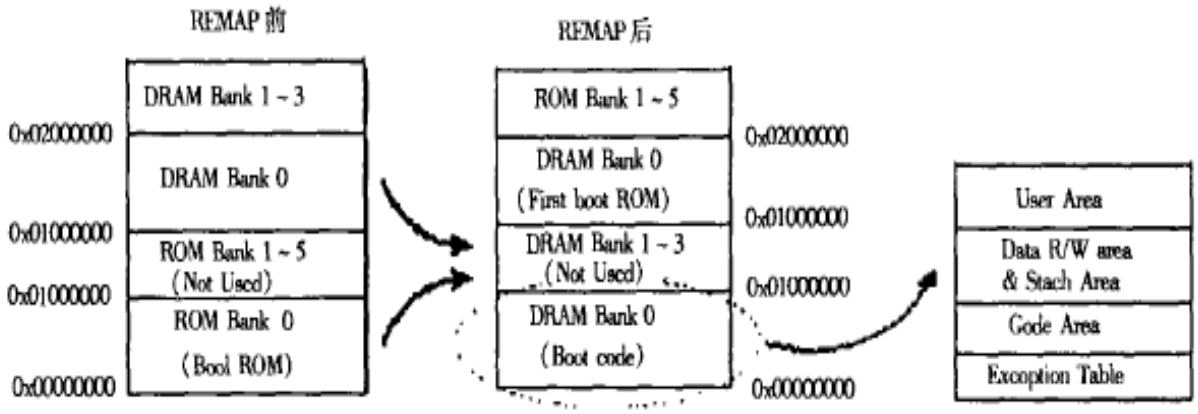
ARM System Memory Map

- Remap
  - Setup Remap bit, ex: s3c2510

Table 4-1. The Base Address of Remapped Memory

	Before remap	After remap
Memory bank0	0x00000000	0x80000000
Memory bank1	0x01000000	0x81000000
Memory bank2	0x02000000	0x82000000
Memory bank3	0x03000000	0x83000000
Memory bank4	0x04000000	0x84000000
Memory bank5	0x05000000	0x85000000
Memory bank6	0x06000000	0x86000000
Memory bank7	0x07000000	0x87000000
SDRAM bank0	0x40000000	0x00000000
SDRAM bank1	0x80000000	0x40000000

  - Setup ROM/SRAM/Flash Control Register, ex: s3c4510



ARM System Memory Map

- Relocate
  - Linking script? GCC ld?, ex s3c24xx
    - u-boot/board/hxd8/config.mk (vs. u-boot/board/hxd8/u-boot.lds)
      - TEXT_BASE = 0x33F80000
    - ldr pc, _start_armboot
    - _start_armboot:.word start_armboot
      - System.map
        - 33f80294 t _start_armboot
        - 33f82e24 T start_armboot

```
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
        . = 0x00000000;
        . = ALIGN(4);
        .text      :
        {
          cpu/arm920t/start.o      (.text)
          cpu/arm920t/s3c24x0/nand_read.o (.text)
          *(.text)
        }
        …
}
```

```
-DTEXT_BASE=$(TEXT_BASE)
arm-linux-gcc –g … -D__KERNEL__ -DTEXT_BASE=0x 0x33F80000 …
```

VMA vs. LMA

- The Gnu ld documentation has the following explanation: "Every loadable or allocatable output section has two addresses. The first is the VMA, or virtual memory address. This is the address the section will have when the output file is run. The second is the LMA, or load memory address. This is the address at which the section will be loaded. In most cases the two addresses will be the same. An example of when they might be different is when a data section is loaded into ROM, and then copied into RAM when the program starts up (this technique is often used to initialize global variables in a ROM based system). In this case the ROM address would be the LMA, and the RAM address would be the VMA. "

U-boot Sections

h2@hiro: ~/my_work/u-boot - Shell - Konsole

Session Edit View Bookmarks Settings Help

```
h2@hiro:~/my_work/u-boot$ objdump -h u-boot

u-boot:     file format elf32-little

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000238a4  33f80000  33f80000  00008000  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00000e58  33fa38a4  33fa38a4  0002b8a4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .rodata.str1.4 000084d8  33fa46fc  33fa46fc  0002c6fc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .data         00005978  33facbd4  33facbd4  00034bd4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  4 .got.plt      0000000c  33fb254c  33fb254c  0003a54c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .got          00000004  33fb2558  33fb2558  0003a558  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .u_boot_cmd   000006c0  33fb255c  33fb255c  0003a55c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .bss          000534b8  33fb2c1c  33fb2c1c  0003ac1c  2**2
                  ALLOC
  8 .debug_line   000099d8  00000000  00000000  0003ac1c  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .debug_info   000304c6  00000000  00000000  000445f4  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .debug_abbrev 0000a163  00000000  00000000  00074aba  2**0
                  CONTENTS, READONLY, DEBUGGING
 11 .debug_aranges 00000bc0  00000000  00000000  0007ec20  2**3
                  CONTENTS, READONLY, DEBUGGING
 12 .debug_frame  0000456c  00000000  00000000  0007f7e0  2**2
                  CONTENTS, READONLY, DEBUGGING
 13 .debug_loc    00019c3e  00000000  00000000  00083d4c  2**0
                  CONTENTS, READONLY, DEBUGGING
 14 .debug_pubnames 00002ee2  00000000  00000000  0009d98a  2**0
                  CONTENTS, READONLY, DEBUGGING
 15 .debug_ranges 00002140  00000000  00000000  000a086c  2**0
                  CONTENTS, READONLY, DEBUGGING
 16 .debug_str    00007c99  00000000  00000000  000a29ac  2**0
                  CONTENTS, READONLY, DEBUGGING
 17 .comment      00000678  00000000  00000000  000aa645  2**0
                  CONTENTS, READONLY
h2@hiro:~/my_work/u-boot$
```

Shell