Embedded System Design
Practice 4

TaeWook Kim & SeokHyun Hong
Hanyang University

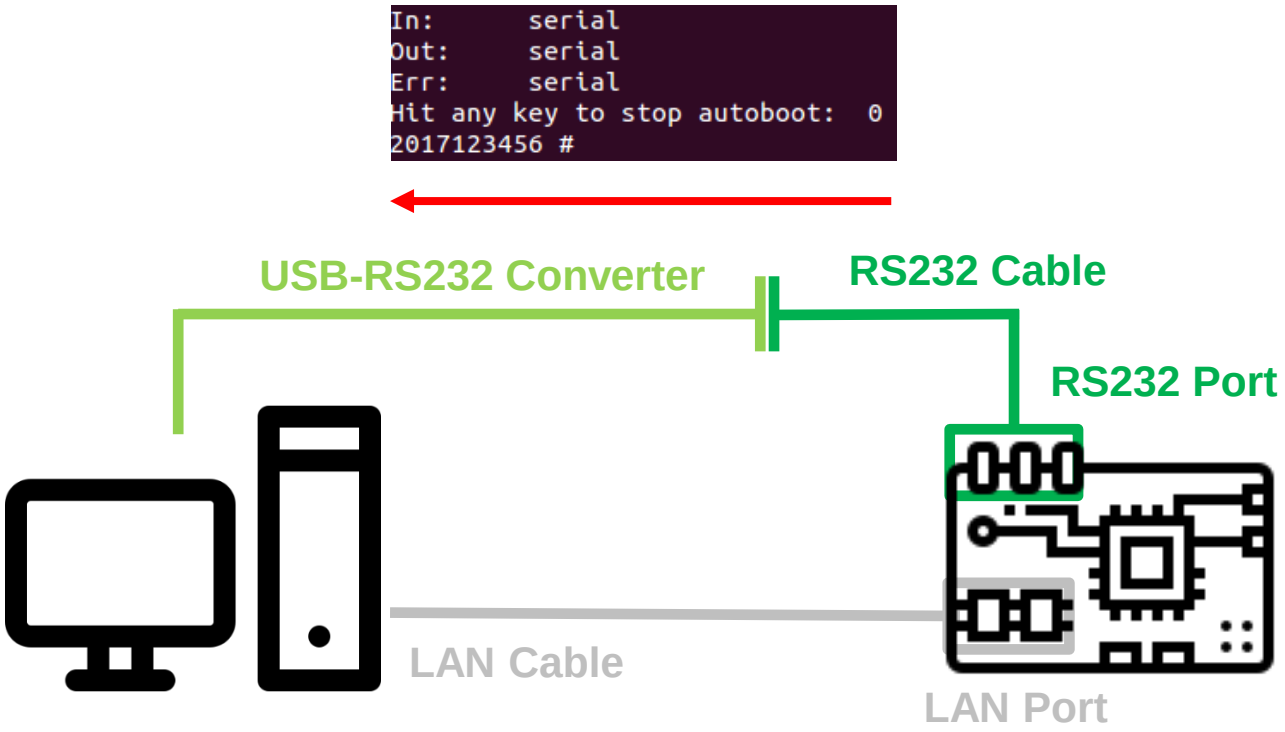
Contents

1. U-Boot & Vpos Porting
2. Hardware Spec
3. VPOS 2.0
4. Data Sheet
5. Startup Code

U-BOOT & VPOS PORTING

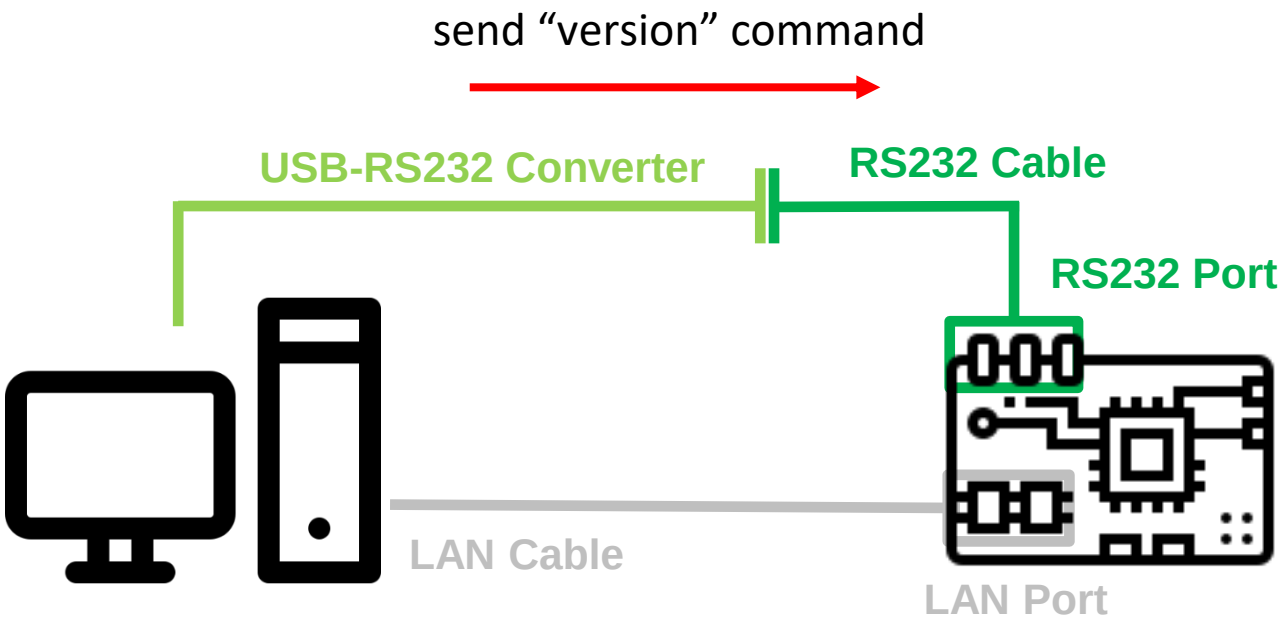
Run U-boot

- When we turn on, the output of the u-boot is transmitted to the minicom through RS232 serial cable



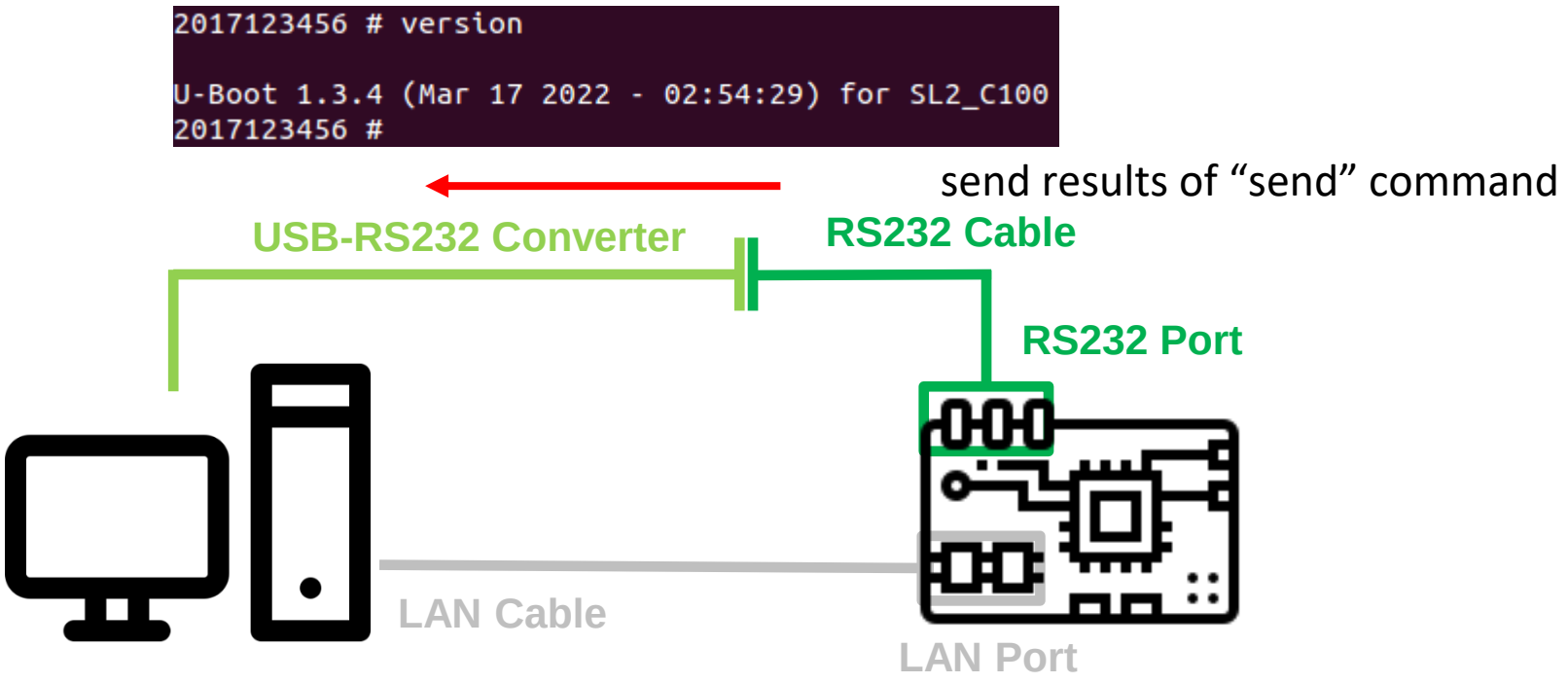
Run U-boot

- And we can also send messages from minicom to the u-boot through RS232 serial cable



Run U-boot

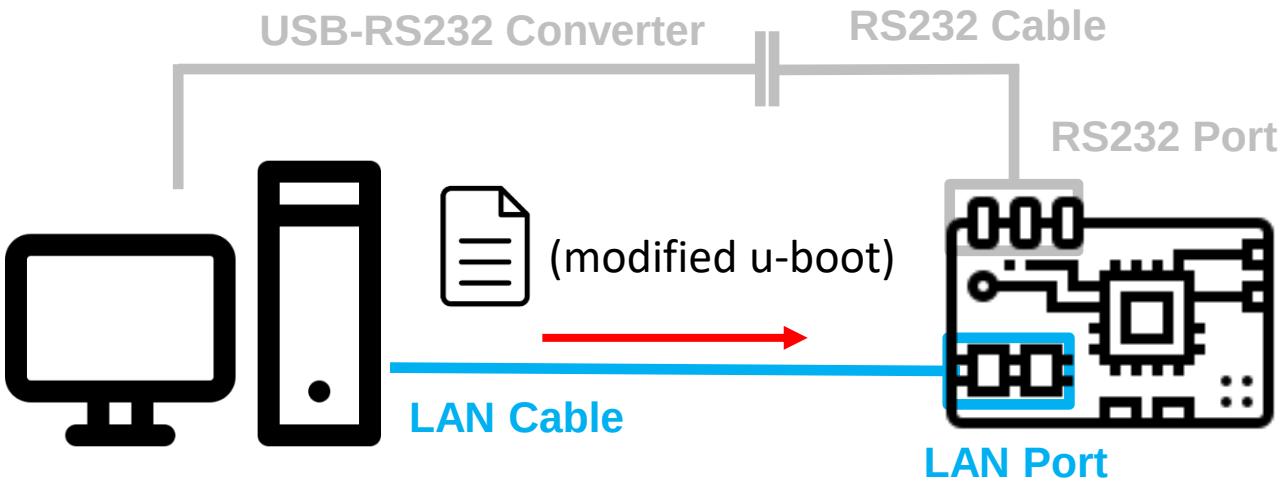
- And we can also send messages from minicom to the u-boot through RS232 serial cable



U-Boot Porting

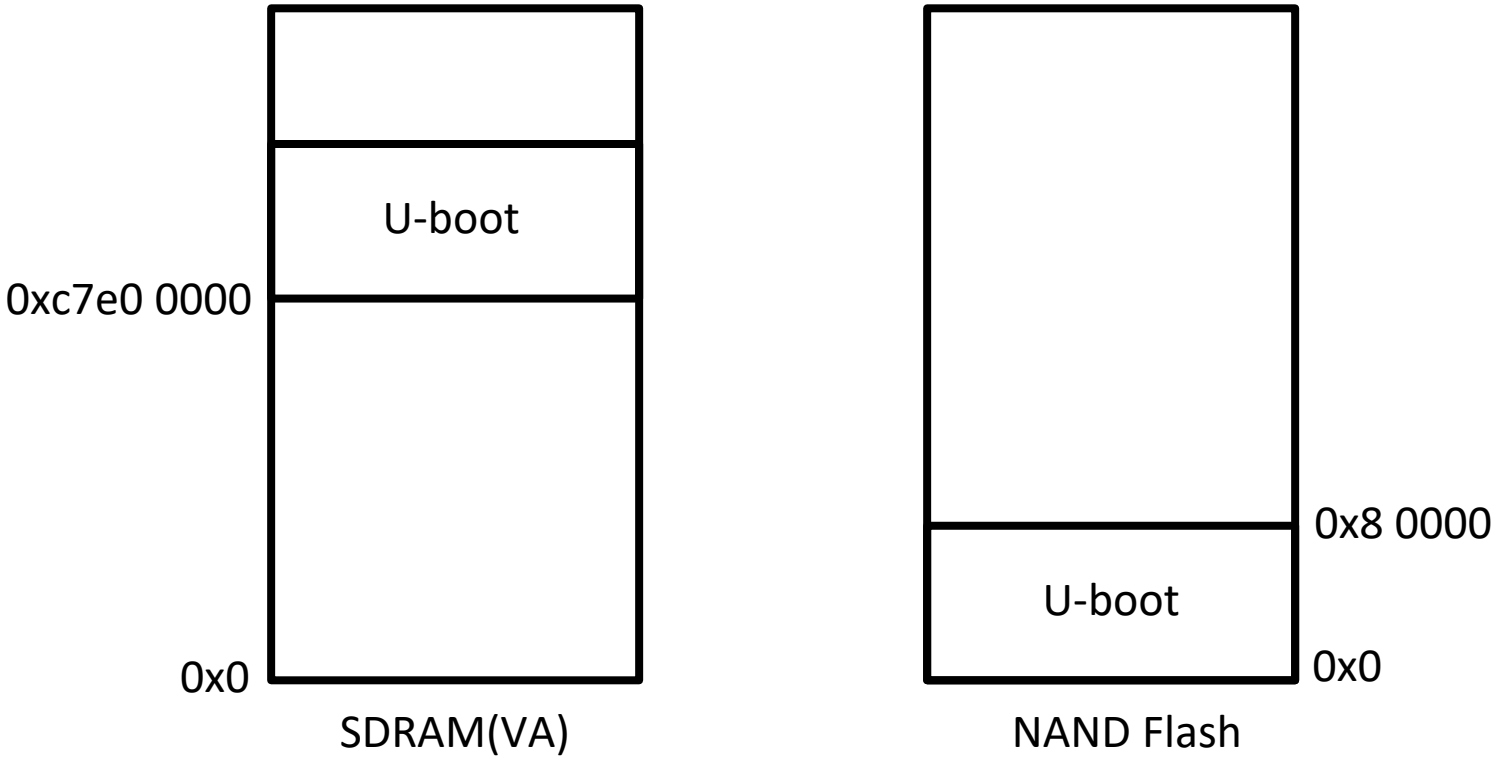
- Through cross-compiler, we created appropriate u-boot for the board

- Then we used the tftp network protocol to send the modified u-boot to the board via LAN cable



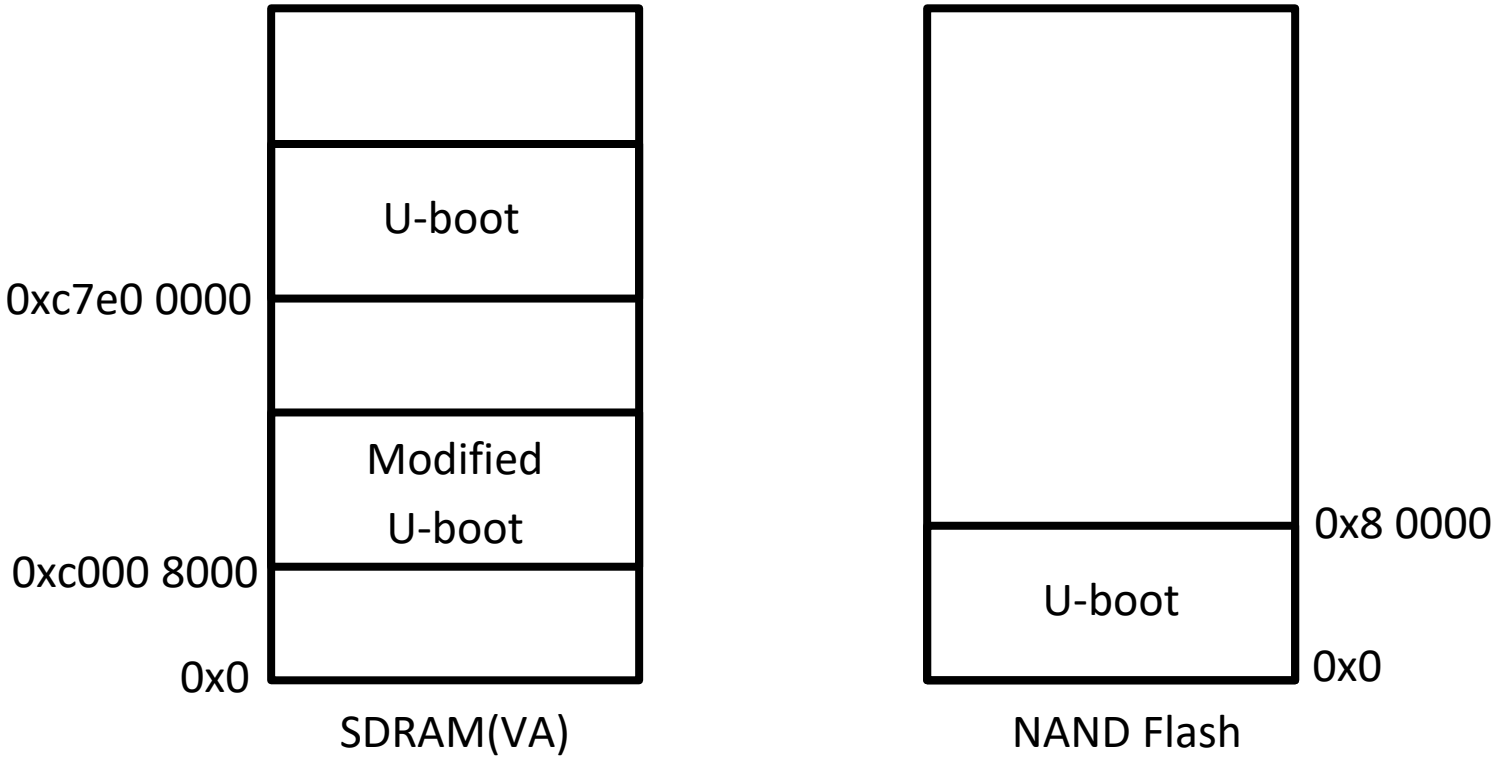
U-Boot Porting

1) When you turn on and boot



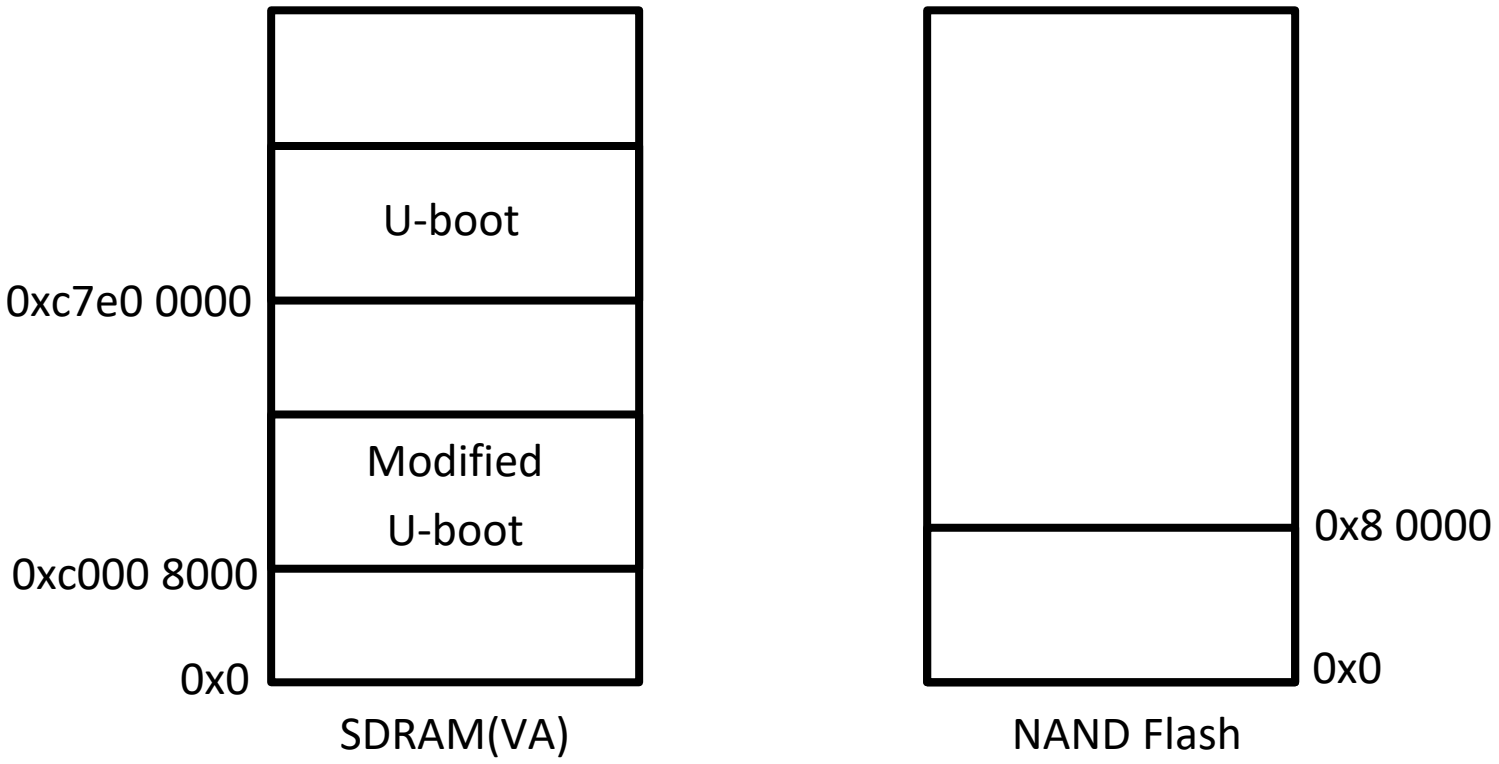
U-Boot Porting

2) tftp c0008000 u-boot.bin



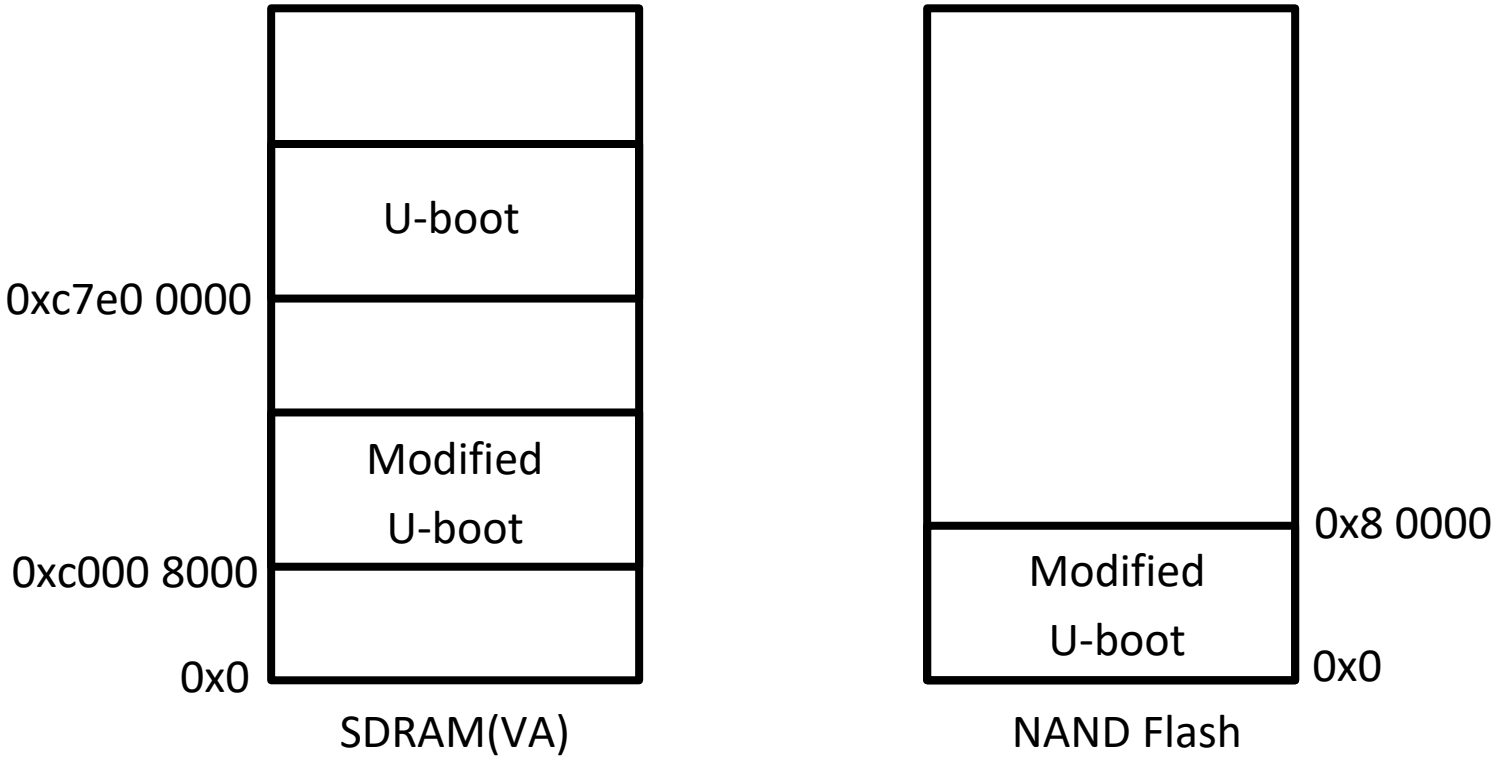
U-Boot Porting

3) nand erase 0 60000



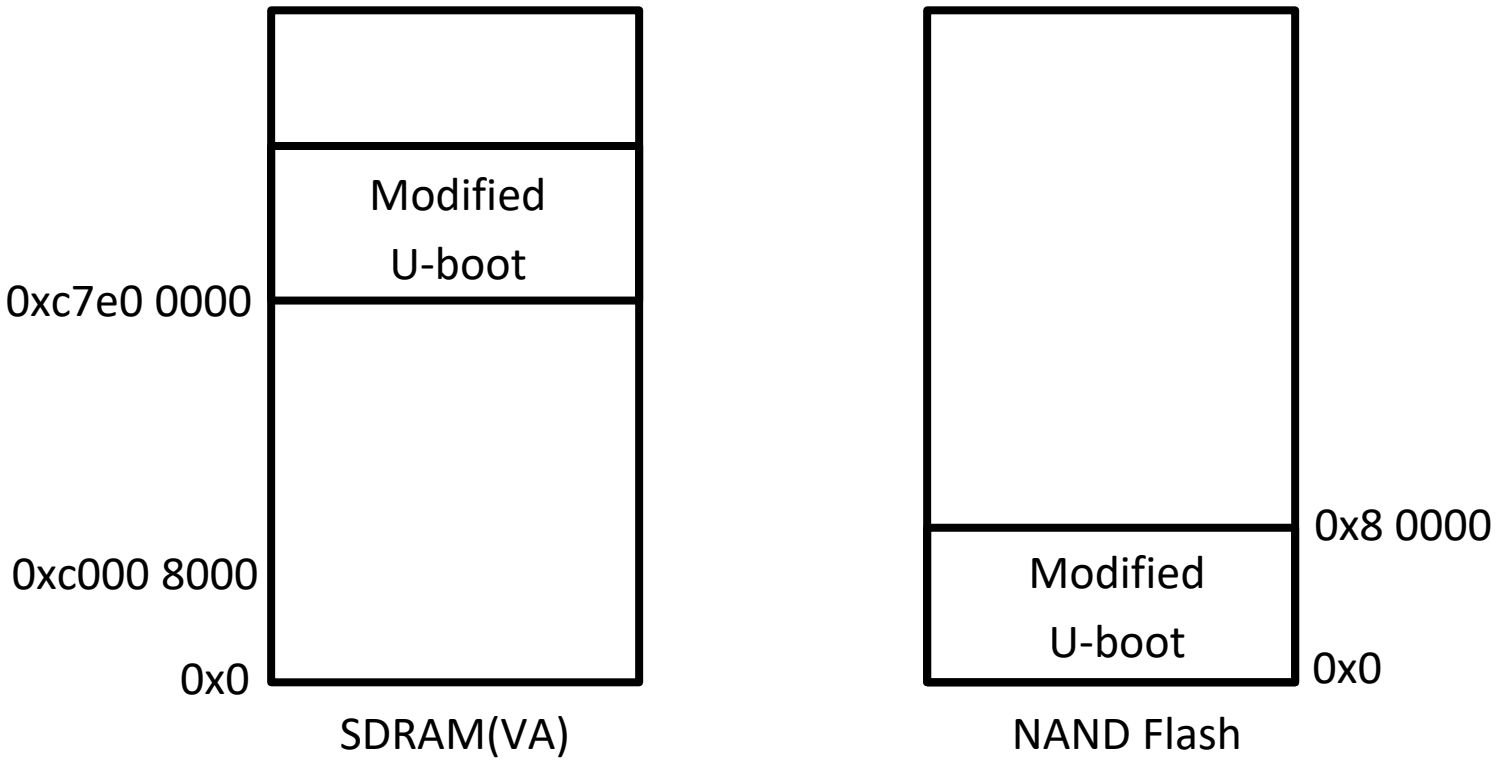
U-Boot Porting

4) nand write c0008000 0 40000



U-Boot Porting

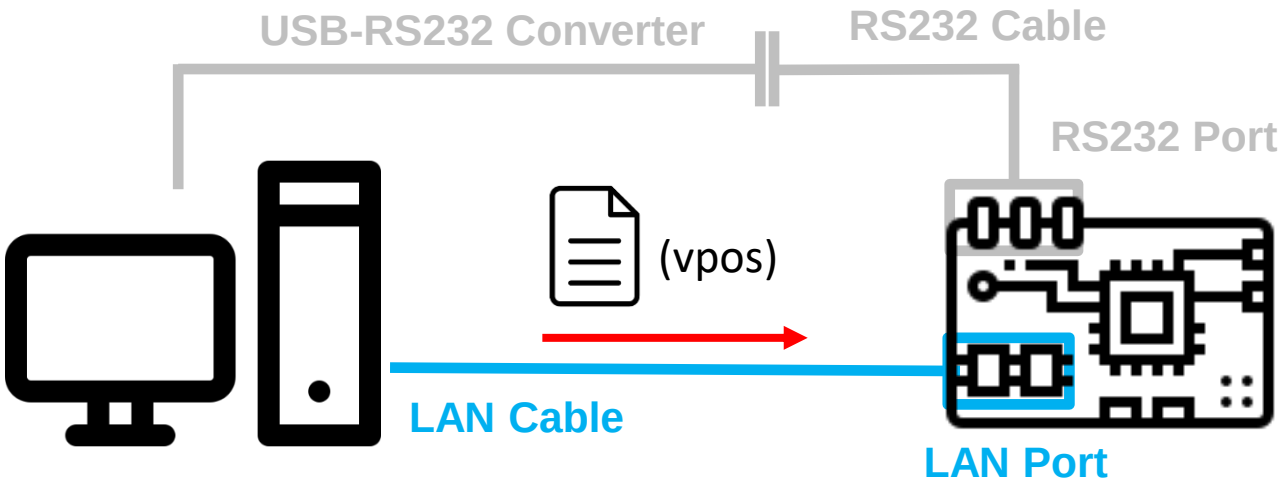
5) turn off and turn on



VPOS Porting

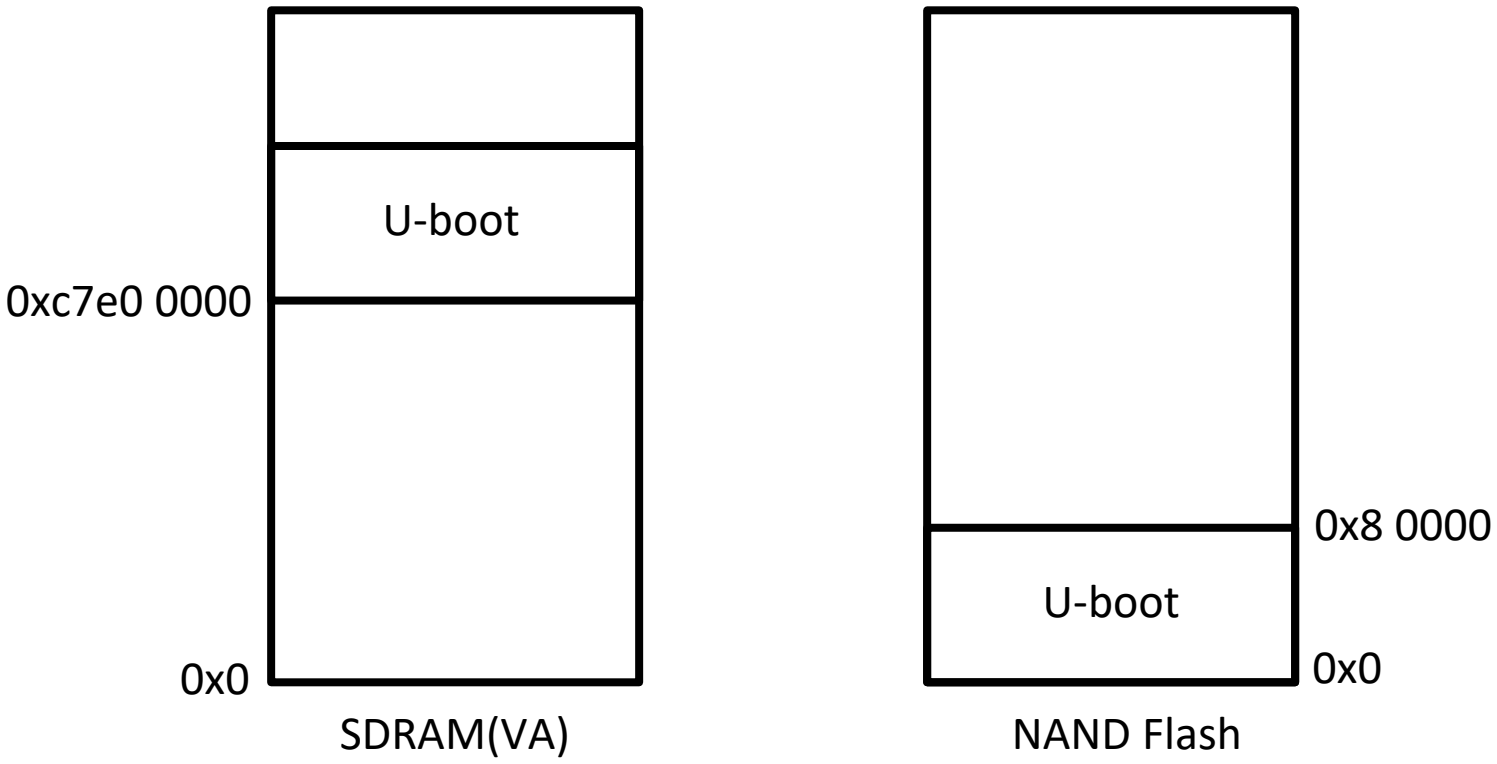
- Through cross-compiler, we created appropriate vpos kernel for the board

- Then we used the tftp network protocol to send the vpos to the board via LAN cable



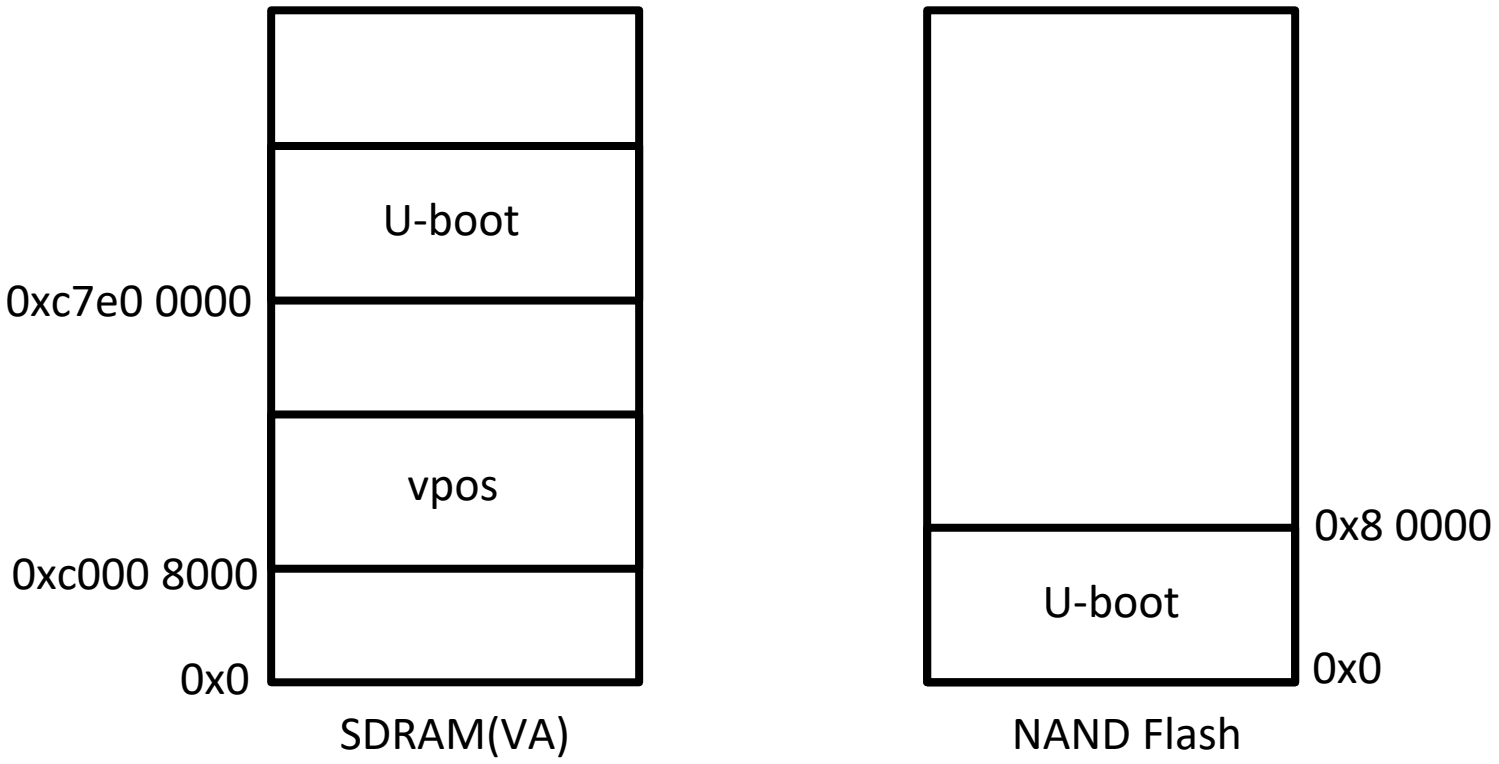
VPOS Porting (not using nand)

1) When you turn on and boot



VPOS Porting (not using nand)

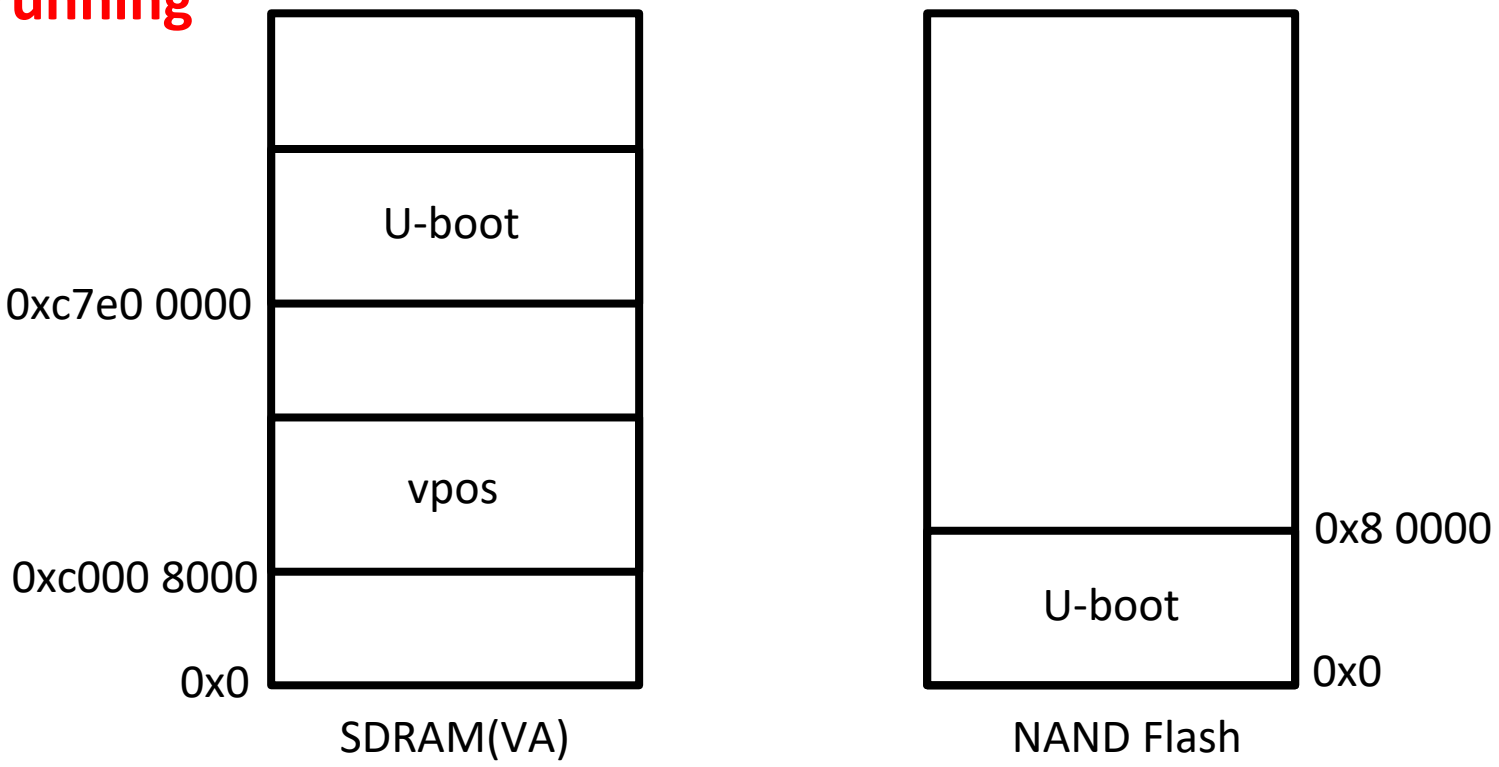
2) tftp c0008000 vpos.bin



VPOS Porting (not using nand)

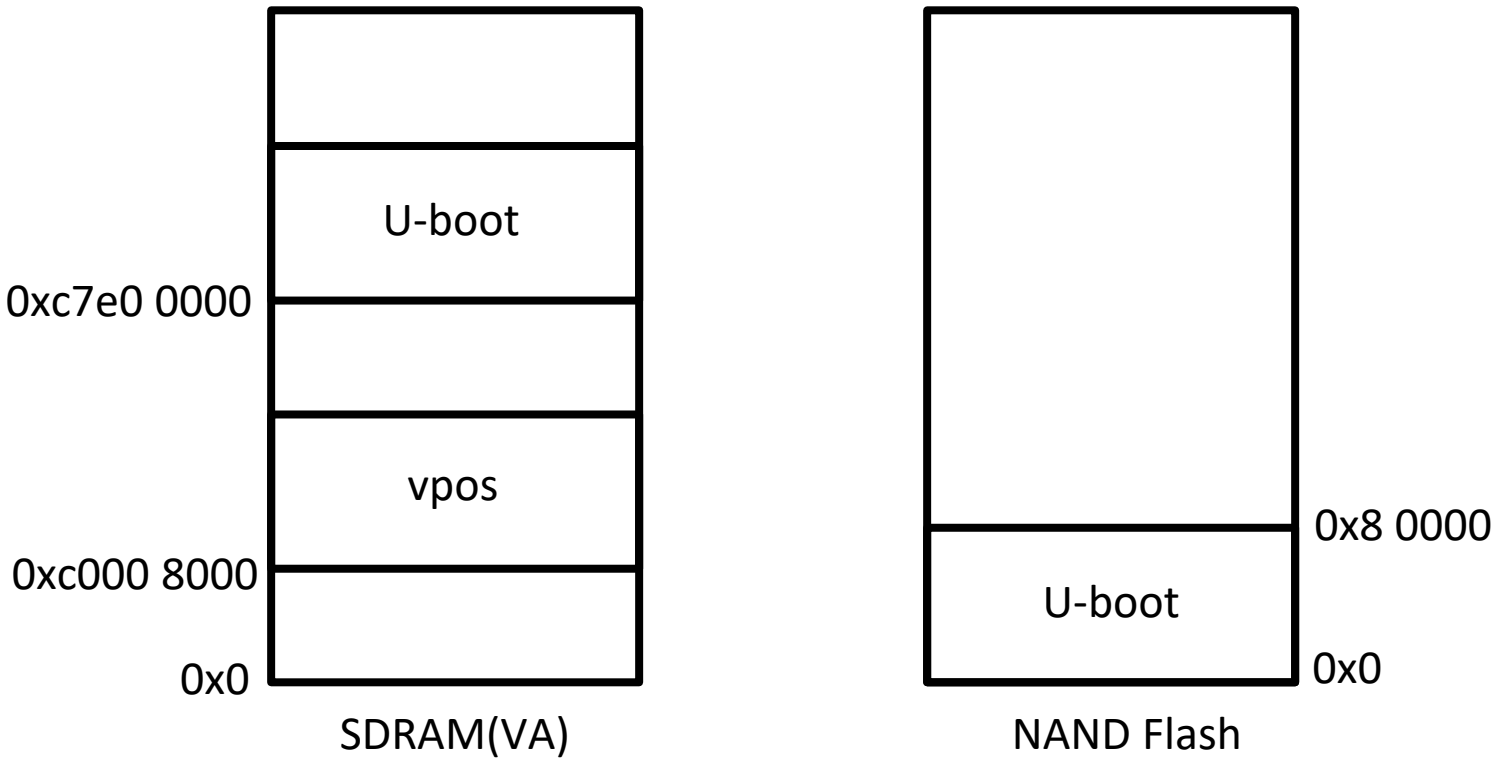
3) bootm c0008000 -> Boot with the image at the SDRAM address

VPOS is running



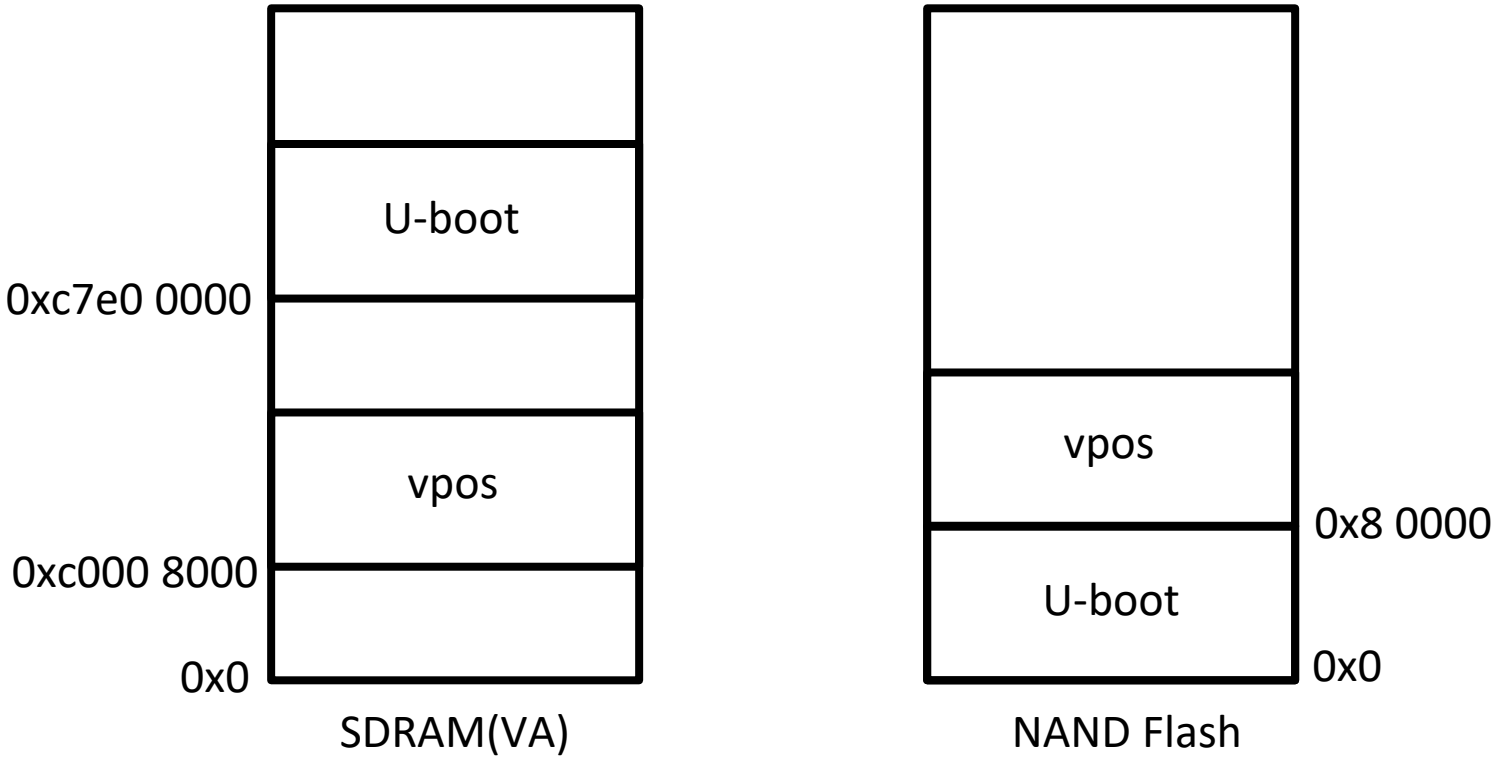
VPOS Porting (using nand)

1) tftp c0008000 vpos.bin



VPOS Porting (using nand)

2) nand erase 8000 400000; nand write c0008000 80000 400000



VPOS Porting (using nand)

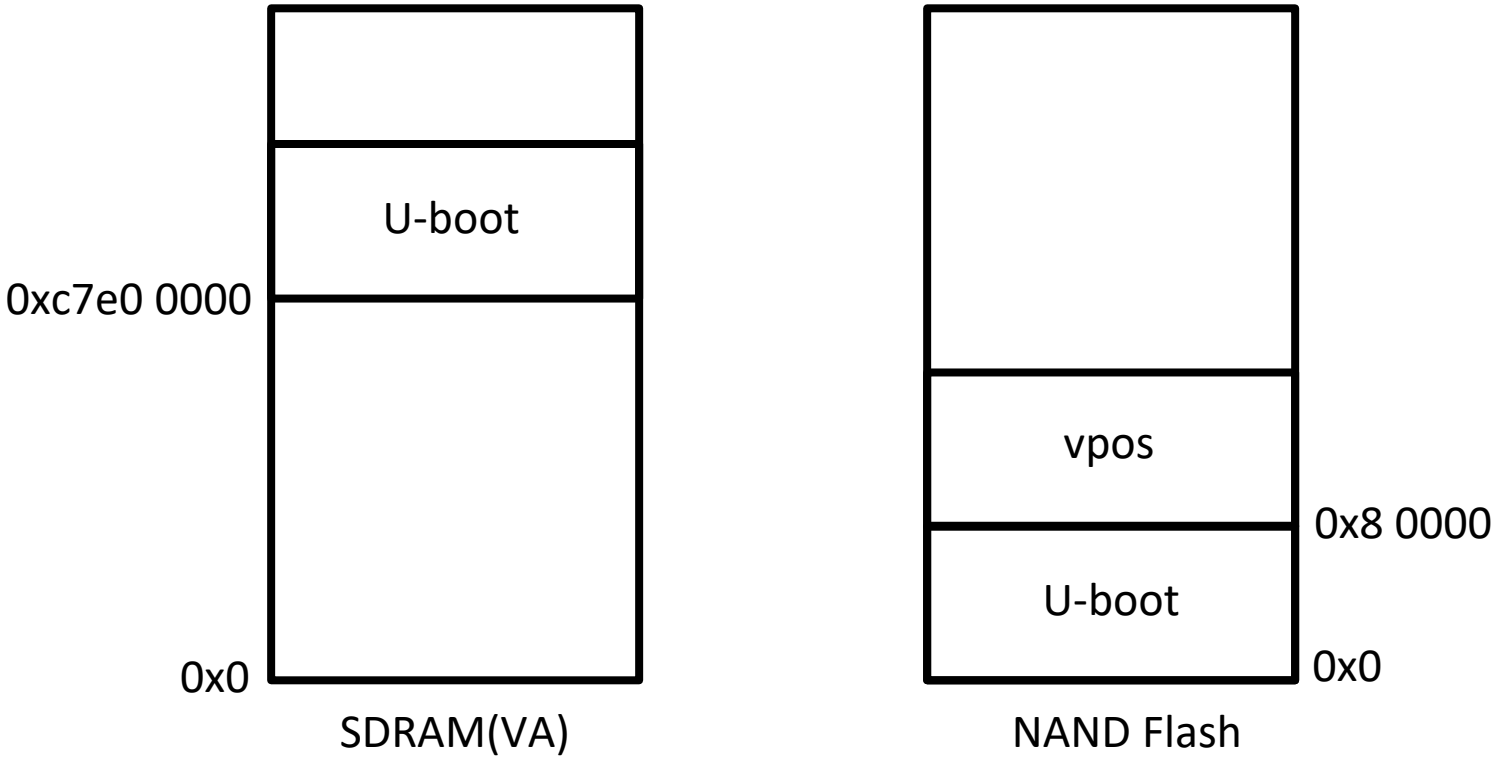
3) setenv bootcmd nand read c0008000 80000 300000\;
bootm c0008000

-> when booted, run with command defined above

4) saveenv

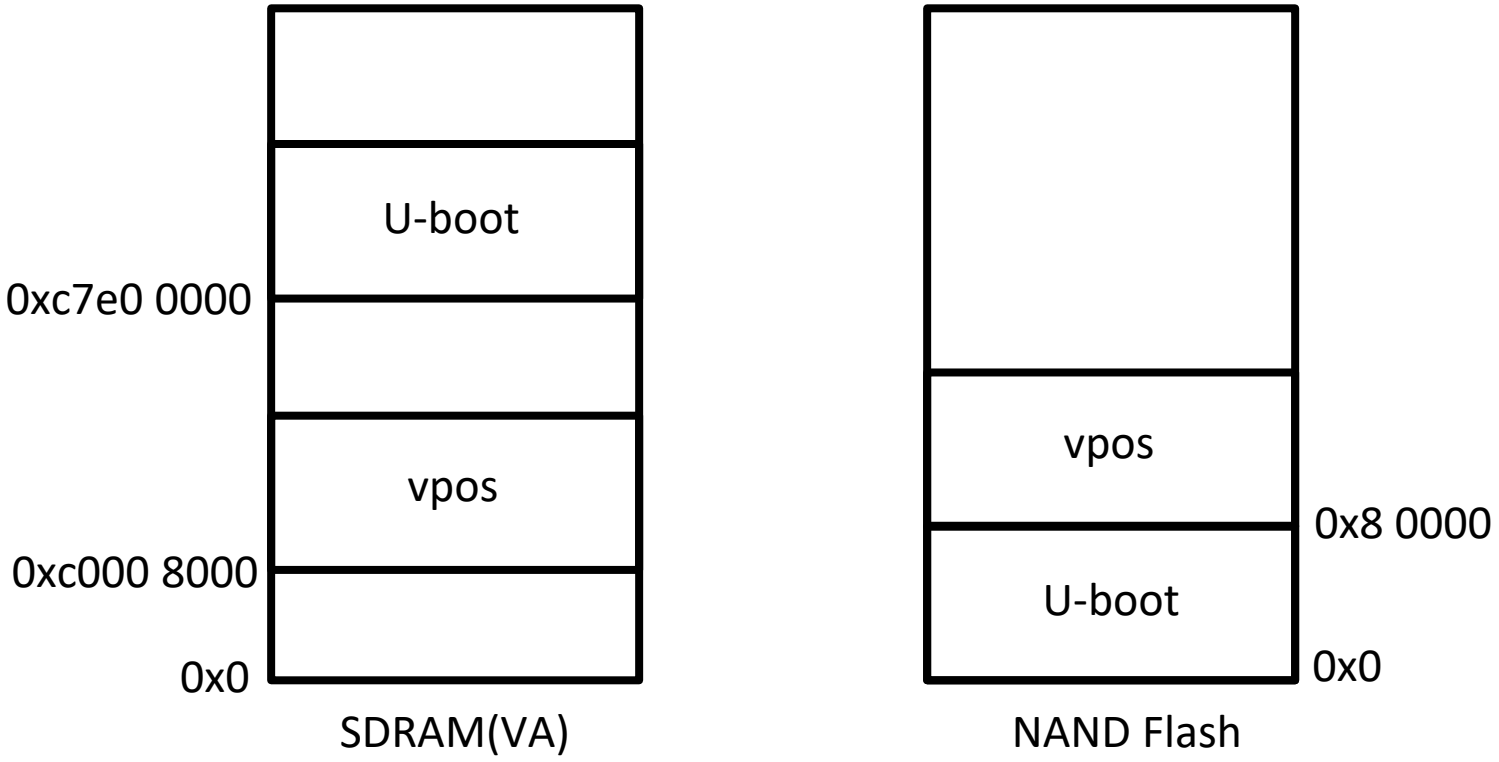
VPOS Porting (using nand)

5) turn off and turn on **U-Boot is running**



VPOS Porting (using nand)

6) 5 seconds after the u-boot starts **VPOS is running**



HARDWARE SPEC

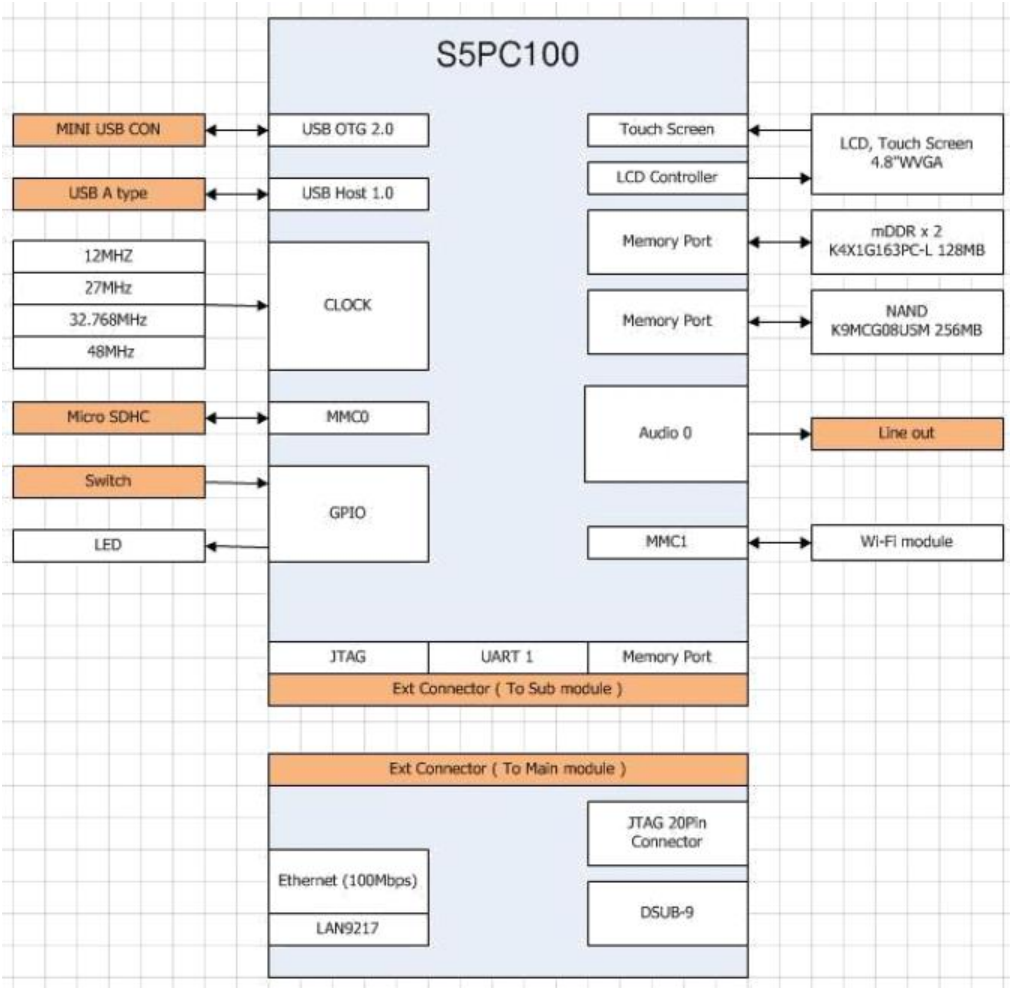
Introduction to SYS-LAB II

- **SYS-LAB II**
  - Embedded system development board with Samsung's S5PC100 processor
  - Support Embedded Linux and Google Android
  - GNU Tools for ARM
    - Support GNU-based cross-compiler

Hardware Specification

- **CPU**
  - S5PC100
    - ARM Cortex-A8 based application processor
- **Memory**
  - mDDR
    - K4X1G163PC-L(F)/GC6 (64x16) x2
  - NOR Flash
    - S29AL008D90TFI020-8Mb(1Mbyte x 8bit / 512k x 16bit)
  - NAND Flash
    - K9F4G08UOM-512M x 8bit

System block diagram of S5PC100



VPOS 2.0

VPOS 2.0

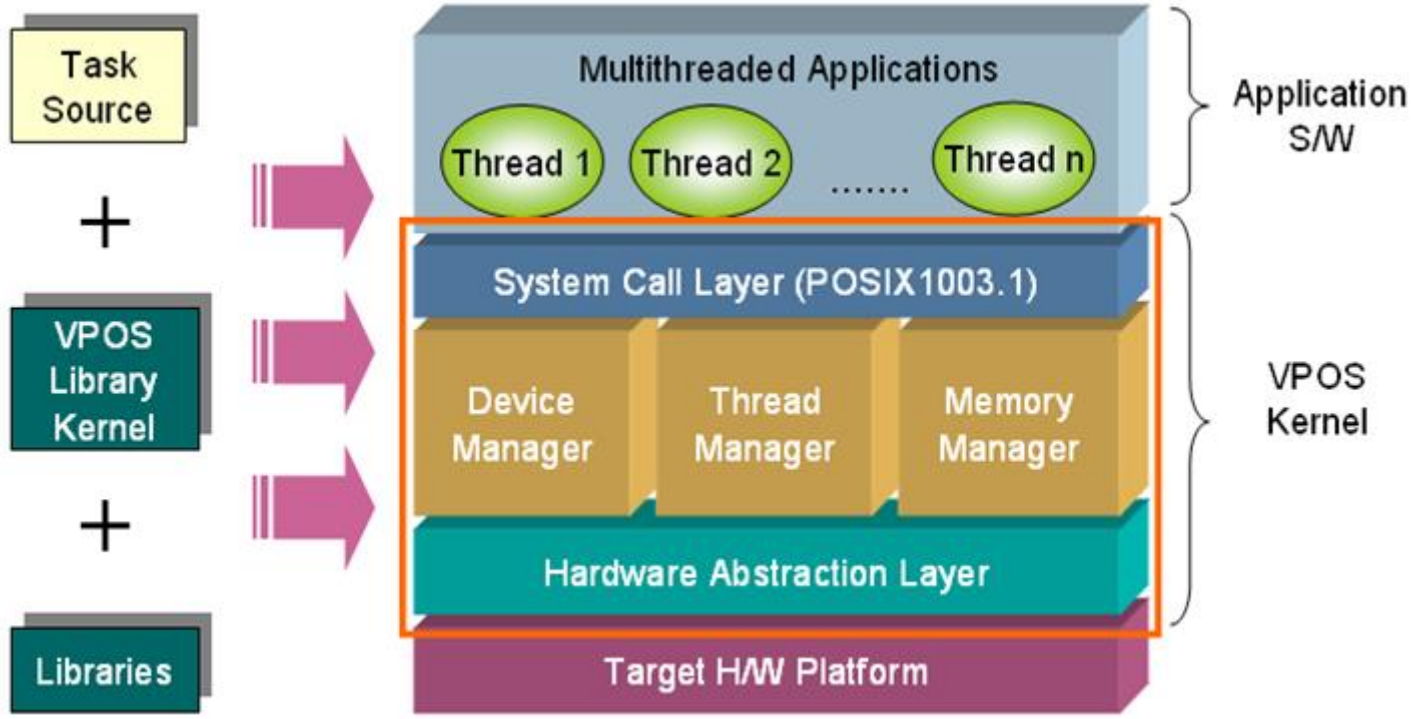
- **VPOS**
  - Verification-Purpose OS

- **Motivation**
  - More complex SOC logic design
    - Too long design and verification time
  - Traditional verification approach is too expensive
    - Complex kernel structure
    - Do not provide functions for SoC development
    - High additional costs (license fees, technical support…)

Features of VPOS 2.0

- **A small and simple kernel structure**
- **Priority based Preemptive kernel**
  - Static priority-based round-robin scheduling
- **Real-time support**
  - using PIP(Priority Inheritance Protocol) synchronization techniques
- **Use HAL(Hardware Abstraction Layer)**
- **Device driver structure for Linux compatibility support**

Structure of VPOS 2.0 kernel



DATA SHEET

Data Sheet

- **What is data sheet?**
  - Documents of performance and characteristics of components, subsystems, and software, etc.
  - Distributed by the manufacturer

- **Information on the data sheet**
  - Product characteristics
  - Simple function description
  - Pin diagram
  - Maximum/minimum values of supply voltage, power consumption, input current, etc.
  - Input/output Waveform Diagram
  - …

Data Sheet for Practice

- **DDI0344K_cortex_a8_r3p2_trm.pdf**
  - Data Sheet for ARM CORTEX A8
  - Provide information about the CORTEX A8 processor
    - Features provided by the processor
    - How to use the corresponding function through register setting
    - …

- **S5PC100_UM_REV101.pdf**
  - Data Sheet of SAMSUNG S5PC100 application processor

STARTUP CODE

Preparing to port the VPOS kernel

1. **Implement Startup code**
2. **UART Settings**
3. **TIMER Settings**
4. **Implement Hardware Interrupt Handler**
   (1) UART Interrupt
   (2) Timer Interrupt
5. **Implement Software Interrupt Entering/Leaving Routine**
6. **Kernel compile + load kernel image in RAM**

Startup code

- **Startup code?**
  - ASM code
  - Initialize embedded target board
    - Perform the initialization process that is difficult to access from C source code
  - Execute before execution of the main function of C code
    - At the end of the code, use the 'branch' command to execute the main function

- **What are the processes in the Startup Code?**
  - Variable initialization
  - PLL setting
  - Memory setting
  - Stack setting
  - Peripheral devices setting ex) UART
  - Branch to c code

- **Location of source code file**
  - hal/cpu/HAL_arch_startup.S

HAL_arch_startup File

- HAL_arch_startup file
  - Startup code and HAL related code

- **Startup code**
  - Vector table
  - Variable initialization
  - Set cache and memory policy
  - Assign stack for each CPU mode

- **HAL related code**
  - SWI
  - HWI
  - Context Switching

Analysis of HAL_arch_startup file

- **Symbol definition**
  - Define external variables, external functions, global variables, etc. to be used in the startup code

.extern : reference external symbol/label
symbol is defined in another module

.global : define global symbol/label

.equ : assign a value to a symbol
must be pre-assigned before referring to symbols

```
#include "../include/vh_cpu_hal.h"

.extern VPOS_kernel_main
.extern vk_undef_handler
.extern vh_hwi_classifier
.extern vk_swi_classifier
.extern vk_pabort_handler
.extern vk_dabort_handler
.extern vk_fiq_handler
.extern vk_not_used_handler
.extern vk_irq_test

.extern vk_sched_save_tcb_ptr

.global vh_VPOS_STARTUP

.global vk_save_swi_mode_stack_ptr
.global vk_save_swi_current_tcb_bottom
.global vk_save_irq_mode_stack_ptr
.global vk_save_irq_current_tcb_bottom
.global vk_save_pabort_current_tcb_bottom

.global vh_restore_thread_ctx
.global vh_save_thread_ctx
.global vh_save_ctx_bottom

.equ vh_USERMODE,   0x10
.equ vh_FIQMODE,    0x11
.equ vh_IRQMODE,    0x12
.equ vh_SVCMODE,    0x13
.equ vh_ABORTMODE,  0x17
.equ vh_UNDEFMODE,  0x1b
.equ vh_MODEMASK,   0x1f
.equ vh_NOINT,      0xc0

.equ vh_userstack,  0x21200000
.equ vh_svcstack,   0X21400000
.equ vh_irqstack,   0x21600000
.equ vh_abortstack, 0x21800000
.equ vh_undefstack, 0x21a00000
.equ vh_fiqstack,   0x21c00000

.equ vh_vector_base,          0x20008044// relocated vector table base address
.equ vh_VICBASE,              0xe4000000
```

Analysis of HAL_arch_startup file

- **vh_VPOS_STARTUP**
  - The first executable part of the kernel code
    - Set to ENTRY (vh_VPOS_STARTUP) in the linker script
  - nop : No-operation. A command that does nothing
  - Start the reset operation with command 'b vh_VPOS_reset' on line 58

```
.text
vh_VPOS_STARTUP:
        /* Camouflaged code for imitating linux
           Linux has a header that includes 8 nop operation, branch code, magic number, binara
ffset */
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        b       vh_VPOS_reset
        magicn: .long 0x016f2818        // Linux magic number
        startn: .long 0x00000000        // start address(offset) is 0
        endn:   .long 0x0000d8fc        // end address(offset) is file size(byte)
        /* Camouflaged code end */

        nop
        nop
        nop
        nop
```

Analysis of HAL_arch_startup file

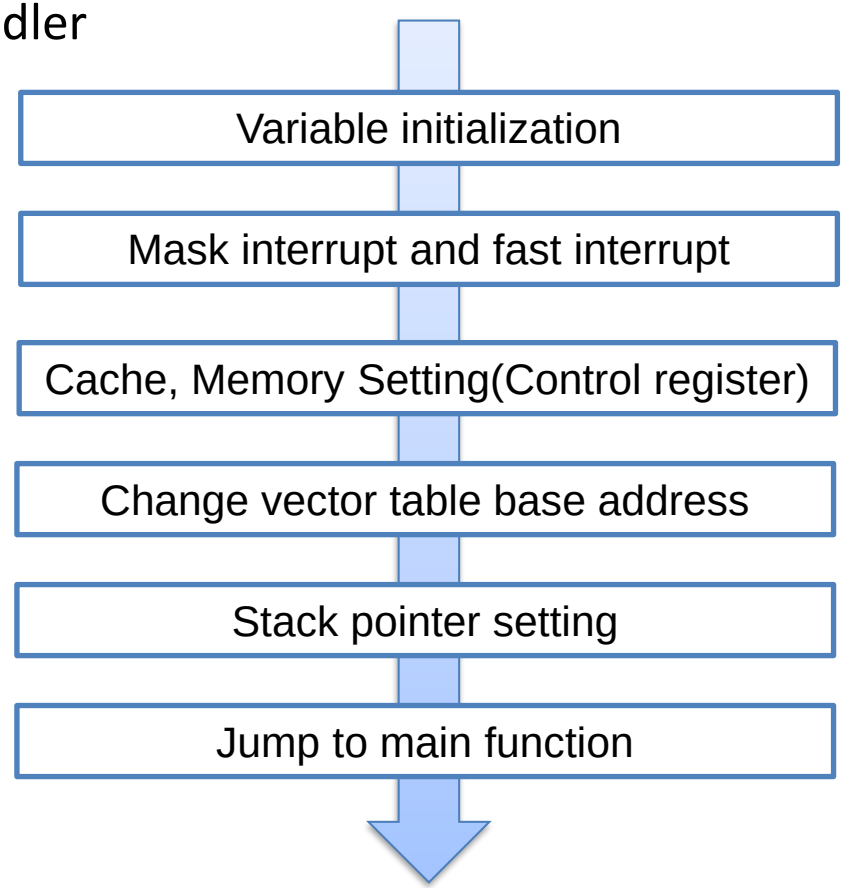
- **vh_vector_start**
  - Vector table
  - When an exception occurs,
    - CPU adds the offset of the exception to the base address of the vector table and executes the corresponding exception handler
  - If you have installed ctags, you can move to that label with move the cursor to the label and press 'ctrl +]'

Exception	Mode	Vector table offset
Reset	SVC	+0x00
Undefined Instruction	UND	+0x04
Software Interrupt(SWI)	SVC	+0x08
Prefetch Abort	ABT	+0x0c
Data Abort	ABT	+0x10
Not assigned	-	+0x14
IRQ	IRQ	+0x18
FIQ	FIQ	+0x1c

```
vh_vector_start:
        b       vh_VPOS_reset
        b       vk_undef
        b       vh_software_interrupt
        b       vh_pabort
        b       vk_dabort
        b       vk_not_used_handler
        b       vh_irq
        b       vk_fiq_handler
```

Analysis of HAL_arch_startup file

- **vh_VPOS_reset**
  - Reset handler



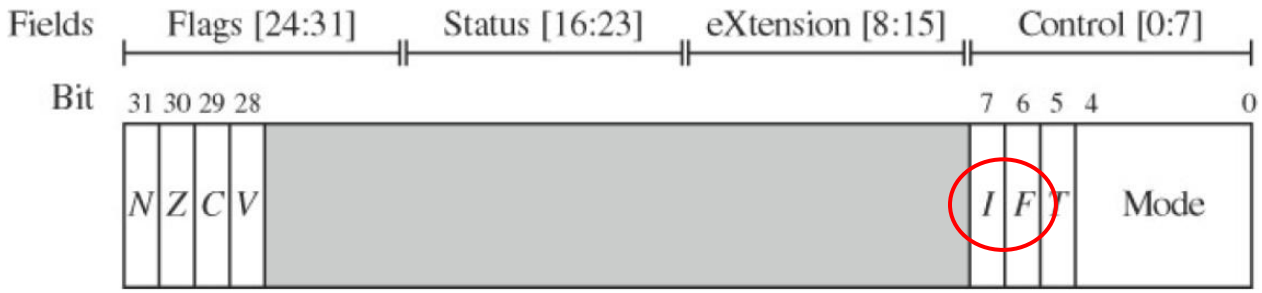
Analysis of vh_VPOS_reset

- **Variable initialization**
  - Initialize register r0 to 0
  - Initialize variables to use in HAL code

```
vh_VPOS_reset:
        // variable initialization
        mov     r0, #0x00
        str     r0, vk_save_swi_mode_stack_ptr
        str     r0, vk_save_swi_current_tcb_bottom
        str     r0, vk_save_irq_mode_stack_ptr
```

Analysis of vh_VPOS_reset

- **Mask interrupt and fast interrupt**
  - Set the I bit and F bit of CPSR to 1
  - I bit : Mask IRQ interrupt
    - 1 : Interrupt Disable
    - 0 : Interrupt Enable
  - F bit : Mask FIQ interrupt
    - 1 : Fast interrupt Disable
    - 0 : Fast interrupt Enable



Analysis of vh_VPOS_reset

- **Mask interrupt and fast interrupt**
  - Code

```
// Mask interrupt and fast interrupt
mrs     r0, cpsr          ----------> copy CPSR to r0
orr     r0, r0, #0xc0     ----------> Set bits 7 and 6 to 1
msr     cpsr, r0          ----------> Copy r0 to CPSR
```

  - m : move / r : register / s : program status register
    - mrs = mov reg PSR / msr = mov PSR reg

Analysis of vh_VPOS_reset

- **Invalidate all instruction caches**
  - Code

```
// Invalidate all instruction caches
mov     r0, #0x00
mcr     p15, 0, r0, c7, c5, 0
```

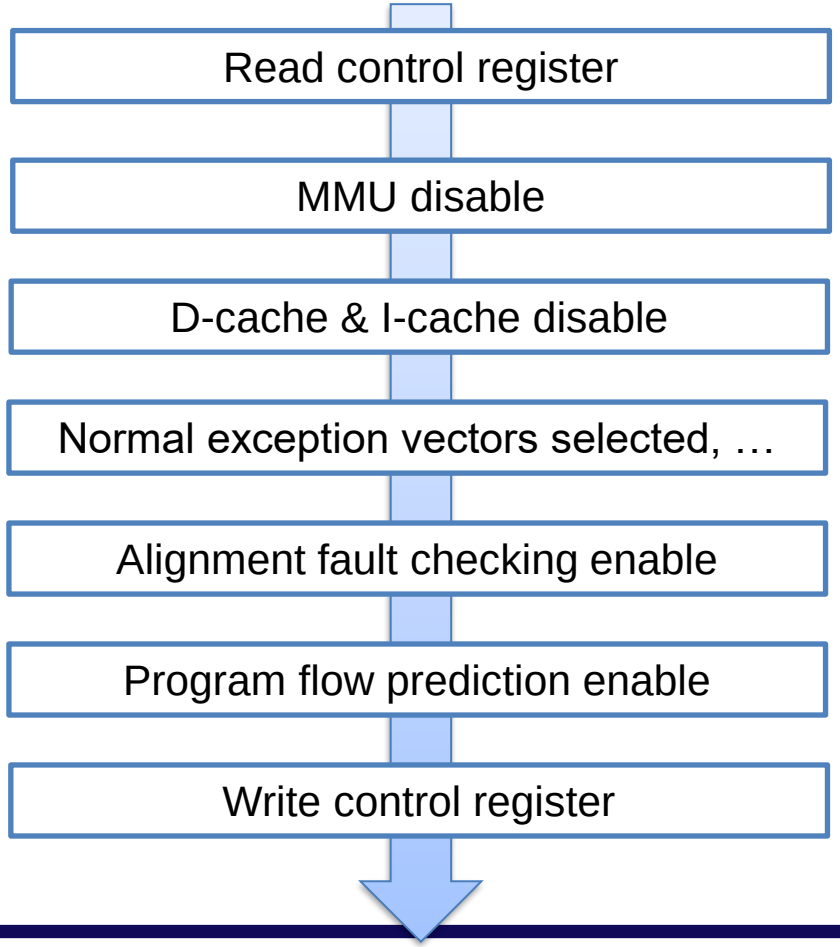
  - Cortex Data sheet (page. 88)

MCR{<cond>}, cp, opcode1, Rd, CRn, CRm {, opcode2}

CRn	Op1	CRm	Op2					
c7	0	c0	0-3	Undefined	-	-	-	-
			4	NOP (WFI)	WO	WO	-	page 3-2
			5-7	Undefined	-	-	-	-
		c1-c3	0-7	Undefined	-	-	-	-
		c4	0	Physical Address	R/W	R/W, B	0x00000000	page 3-71
			1-7	Undefined	-	-	-	-
		c5	0	Invalidate all instruction caches to point of unification	WO	WO	-	page 3-68

Analysis of vh_VPOS_reset

- **Control Register(c1) Setting**



Analysis of vh_VPOS_reset

- **Control Register(c1) Setting**
  - Data sheet
    - See page. 122 ~ 125 (P122 - 3.2.25)
    - Read control register (p.124)

      To access the Control Register, read or write CP15 with:

      ```
      MRC p15, 0, <Rd>, c1, c0, 0 ; Read Control Register
      ```

    - Write control register (p. 125)

      ```
      MCR p15, 0, <Rd>, c1, c0, 0 ; Write Control Register
      ```

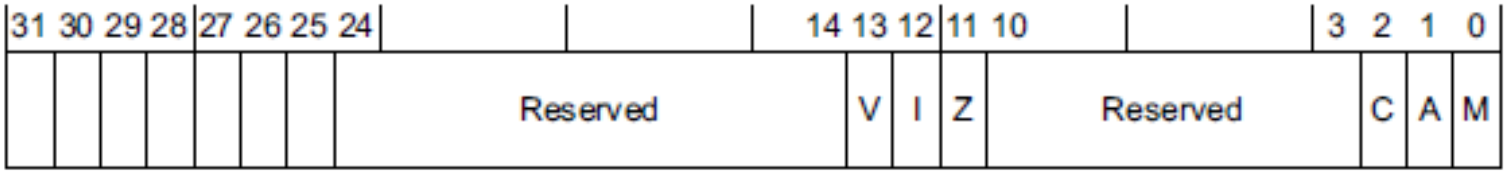
    - For the remaining settings, see Table 3-46, Control Register bit funcitons (p. 123~124)

Analysis of vh_VPOS_reset

- **Control Register(c1) Setting**
  - Code

```
// Control Register Setting
mrc     p15, 0, r0, c1, c0, 0
bic     r0, r0, #0x01
bic     r0, r0, #0x04
bic     r0, r0, #0x1000
bic     r0, r0, #0x2000
orr     r0, r0, #0x02
orr     r0, r0, #0x800
mcr     p15, 0, r0, c1, c0, 0
```

→ Read Control Register
→ MMU disable
→ D-cache disable
→ I-cache disable
→ Normal exception vectors, base address 0x00000000
→ Alignment fault checking enable
→ Program flow prediction enable
→ Write Control Register



Analysis of vh_VPOS_reset

- **Change vector table base address**
  - Store base address of vector table
  - If an exception occurs, jump to the handler by adding the offset of the exception to the stored base address
  - Data sheet
    - Page. 195~196 참고

```
MCR p15, 0, <Rd>, c12, c0, 0 ; Write Secure or Nonsecure Vector Base
                             ; Address Register
```

Analysis of vh_VPOS_reset

- **Change vector table base address**
  - Base address: 0x20008044

```
.text
vh_VPOS_STARTUP:
        /* Camouflaged code for imitating linux
           Linux has a header that includes 8 nop operation, branch code, mag
number, binaray file start offset, and file size(end offset */
        nop
        nop
```

0x20008000

```
        nop
        nop
vh_vector_start:
        b       vh_VPOS_reset
        b       vk_undef
        b       vh_software_interrupt
        b       vh_pabort
```

0x20008044

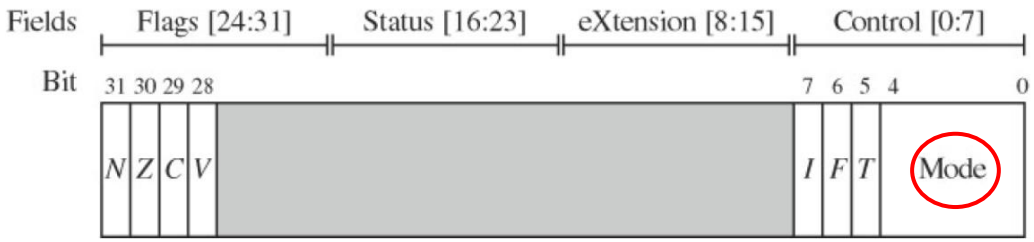
  - Code

```
// change vector table base address (0x20008044)
ldr     r0, =vh_vector_base
mcr     p15, 0, r0, c12, c0, 0
```

Analysis of vh_VPOS_reset

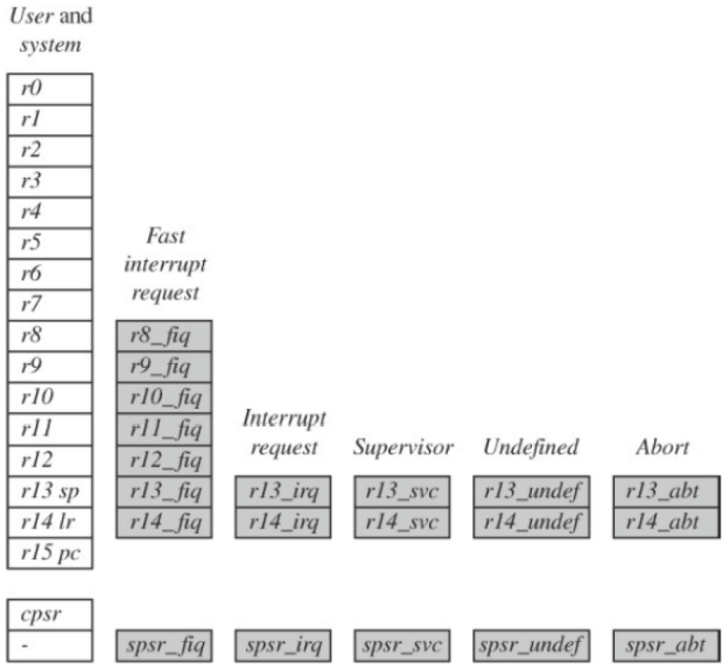
- **Stack pointer setting**
  - ARM CPU has Abort, FIQ, IRQ, Supervisor, System, Undefined, User modes
  - Set stack start position in stack pointer(r13, sp) for each mode

- **Setting Flow**
  1. Copy cpsr to r0 (using ‘mrs’ command)
  2. Change CPU mode by modifying 5 bits[4: 0] to indicate process mode
  3. Store stack start position in the stack pointer of the mode.



Analysis of vh_VPOS_reset

- **Stack pointer setting**
  - ARM has sp and lr for each mode
    - Changing sp in Undef mode does not affect sp in any other mode
    - If you want to modify sp in a specific mode, you must enter that mode



Analysis of vh_VPOS_reset

- **Stack pointer setting**
  - Code

Symbols representing each mode bit and interrupt mask bit are assigned to some values using .equ
Ex) vh_UNDEFMODE = 0x1b
vh_MODEMASK = 0x1f
vh_NOINT = 0xc0

How to change the CPU mode :

1. Get CPSR with mrs command
2. Clear mode bit and interrupt mask bit to 0
3. Set the mode bit of the desired mode and set the interrupt mask bit
4. Save to CPSR with msr command

```
// stack pointer setting
mrs r0,cpsr

bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_UNDEFMODE|vh_NOINT
msr cpsr_cxsf,r1
ldr sp,=vh_undefstack

bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_ABORTMODE|vh_NOINT
msr cpsr_cxsf,r1
ldr sp,=vh_abortstack

bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_IRQMODE|vh_NOINT
msr cpsr_cxsf,r1
ldr sp,=vh_irqstack

bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_FIQMODE|vh_NOINT
msr cpsr_cxsf,r1
ldr sp,=vh_fiqstack

bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,#vh_SVCMODE|vh_NOINT
msr cpsr_cxsf,r1
ldr sp,=vh_svcstack
```

Analysis of vh_VPOS_reset

- **User Mode Stack Setting**
  - Setting sp in USER mode
  - Using ‘vh_USERMODE‘ for mode bit
  - However, interrupt must be enabled in USER mode !!

```
.equ vh_USERMODE,    0x10
```

```
// user mode sp
bic r0,r0,#vh_MODEMASK|vh_NOINT
orr r1,r0,
msr cpsr_cxsf,r1
ldr sp,=
```

Analysis of vh_VPOS_reset

- Jump to main function
  - Code

```
b VPOS_kernel_main
```

  - Branch instruction to VPOS_kernel_main function
    - VPOS_kernel_main() : The main function of the VPOS kernel
    - Located in vpos/kernel/kernel.start.c

- **Reset process finished**

Thank you