

Linux for StrongARM platform

2002/9/13 YSY

u9067561@cis.nctu.edu.tw



Introduction

- n StrongARM SA-1110 with SA-1 CPU Core
- n ARM V4 architecture
- n ARM mode = 32 bit, Thumb mode = 16 bit
- n 16K bytes I-cache, 8K bytes D-cache (with a mini D-cache)
I-cache/D-cache can be enabled separately.
Read buffers/Write buffers, TLB, I-MMU/D-MMU
- n Peripherals : UART, PLL, timer, Memory controller, PCMCIA controller,
LCD controller, Interrupt controller, USB controller, IrDA port,
Touch Panel, Audio control
- n Extension : SA-1111

ARM registers set

n CPSR: Current Program Status Register

bit

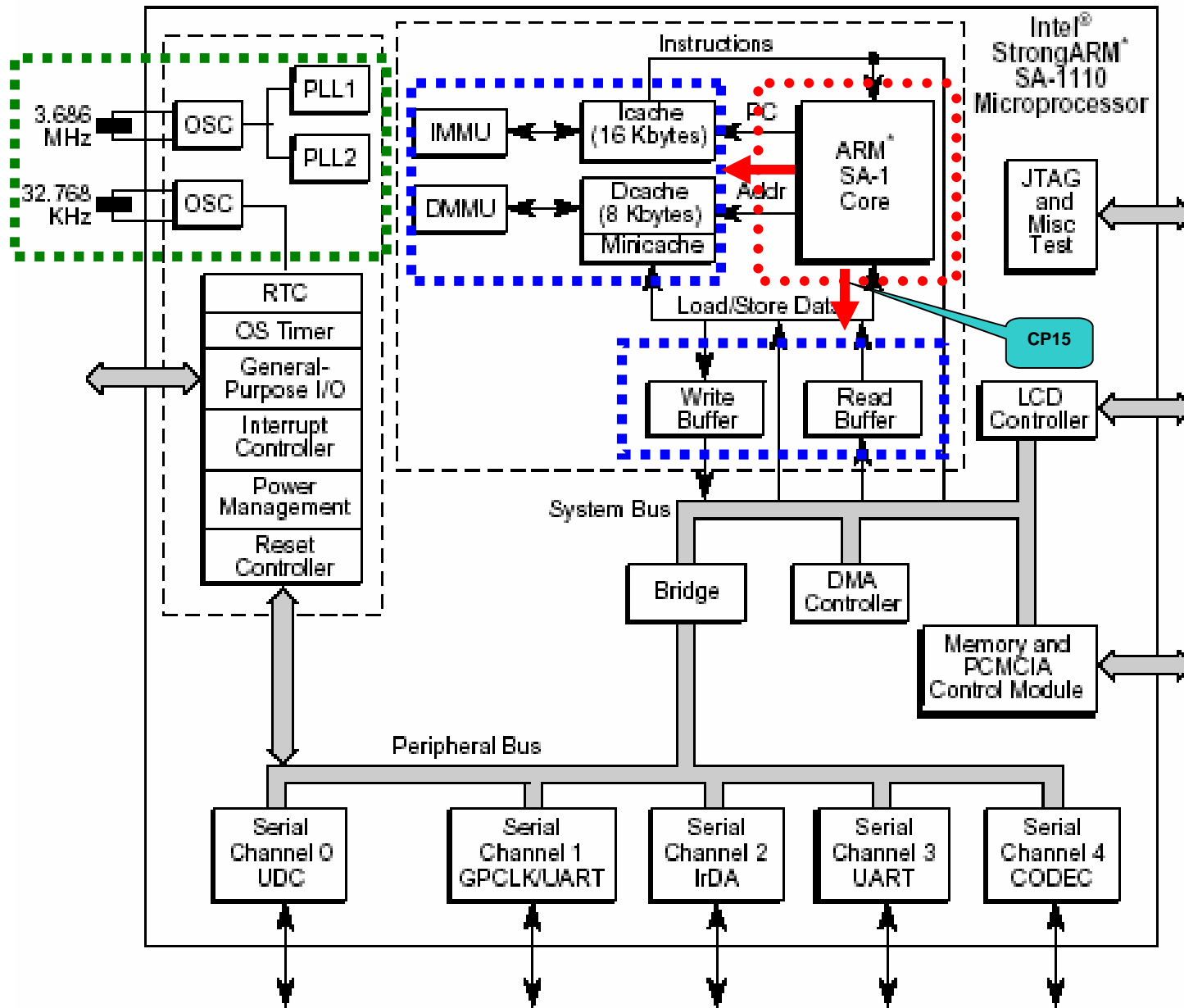
- 31 N : Negative ?
- 30 Z : the result is 0 ?
- 29 C : Carry ?
- 28 V : Overflow ?
- 27 Q : Overflow/Saturation
- 7 I : Interrupt enabled
- 6 F : Fast Interrupt enabled
- 5 T : ARM/Thumb instruction
- 4~0 M[] : Mode

n SPSR: Stored Program Status Register

n PC = Program Counter = R15

n LR = Link Register = R14

User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_fiq	SPSR_irq	SPSR_fiq



Address Range	Resource Size	Use	SA-1110 Development Board Size: Width
E800,0000-FFFF,FFFF	Reserved 384 Mbyte	Reserved	—
E000,0000-E7FF,FFFF	Zeros Bank 128 Mbyte	Cache flush	Read zeros, no bus cycle
D800,0000-DFFF,FFFF	SDRAM bank 3 128 Mbyte	Empty	—
D000,0000-D7FF,FFFF	SDRAM bank 2 128 Mbyte	Expansion SDRAM bank on SA-1111 board Clk/4	16 Mbyte/32 Mbyte/64 Mbyte: 32 bits wide
C800,0000-CFFF,FFFF	SDRAM bank 1 128 Mbyte	Empty	—
C000,0000-C7FF,FFFF	SDRAM bank 0 128 Mbyte	Main SDRAM bank on SA-1110 board Clk/2	16 Mbyte/32 Mbyte/64 Mbyte: 32 bits wide
B000,0000-BFFF,FFFF	LCD and DMA Control 256 Mbyte	Internal	—
A000,0000-AFFF,FFFF	Memory Control 256 Mbyte	Internal	—
9000,0000-9FFF,FFFF	SA-1110 System Control Module Register ^a 256 Mbyte	Internal	—
8000,0000-8FFF,FFFF	Peripheral Control 256 Mbyte	Internal	—
5000,0000-7FFF,FFFF	Reserved 768 Mbyte	Reserved	—
4800,0000-4FFF,FFFF	CS5 128 Mbyte	GFX (uses rdy)	TBD bytes: 32 bits wide
4000,0000-47FF,FFFF	CS4 128 Mbyte	SA-1111 (uses Rdy)	8KB
3000,0000-3FFF,FFFF	PCMCIA/CF Slot B 512 Mbyte	Compact Flash	2x2KBytes: 16 bits wide
2000,0000-2FFF,FFFF	PCMCIA/CF Slot A 512 Mbyte	PCMCIA (SA-1111)	3x64 MBytes: 16 bits wide
1800,0000-1FFF,FFFF	CS3 128 Mbyte	Ethernet device (uses rdy)	1Kbytes: 8 bits wide
1000,0000-17FF,FFFF	CS2 128 Mbyte	SA-1110 Development Board system registers	Various widths
0800,0000-0FFF,FFFF	CS1 128 Mbyte	Expansion flash on SA-1111 board	16/32 MBytes: 32 bits wide
0000,0000-07FF,FFFF	CS0 128 Mbyte	Boot/Application flash ROM on SA-1110 board	16/32 MBytes: 32 bits wide

Kernel Area

System register Area

Boot Loader Area



Boot sequence

n Boot loader

AngelBoot

RedBoot

Blob

Booting sequence (blob)

n Code is initially run from ROM
(or flash ROM) from address 0

n **After reset**

Core clock = 59MHz

Interrupt(IRQ, FIQ) is disabled

In supervisor mode

In ARM mode

Cache and MMU is off

n **Setup Interrupt Table**

vector location is fixed

n **Mask all interrupt sources**

those from Interrupt controller

n **Set higher core clock speed**

configure the PLL which is related to
CPU core

59MHz -> 221MHz

Total are 12 speeds, but there is
problem below 132.7MHz

```
_start : b   reset           //0x00
         b   undefined_instruction //0x04
         b   software_interrupt  //0x08
         b   abort_prefetch     //0x0c
         b   abort_data         //0x10
         b   not_used           //0x14
         b   irq                //0x18
         b   fiq                //0x1c
```

```
reset:
    // store 0 to address 0x90050004
    mov    r1, #0x90000000
    add    r1, r1, #0x50000
    mov    r2, #0x00
    str    r2, [r1, #0x4]

    // store 0x0b to address 0x90020014
    mov    r1, #0x90000000
    add    r1, r1, #0x20000
    mov    r2, #0x0b
    str    r2, [r1, #0x14]
```

Booting sequence (blob) 1/6

n Code is initially run from ROM
(or flash ROM) from address 0

n **After reset**

Core clock = 59MHz

Interrupt(IRQ, FIQ) is disabled

In supervisor mode

In ARM mode

Cache and MMU is off

n **Setup Interrupt Table**

vector location is fixed

n **Mask all interrupt sources**

those from Interrupt controller

n **Set higher core clock speed**

configure the PLL which is related to
CPU core

59MHz -> 221MHz

Total are 12 speeds, but there is
problem below 132.7MHz

```
_start : b   reset           //0x00
         b   undefined_instruction //0x04
         b   software_interrupt  //0x08
         b   abort_prefetch     //0x0c
         b   abort_data         //0x10
         b   not_used           //0x14
         b   irq                //0x18
         b   fiq                //0x1c
```

```
reset:
    // store 0 to address 0x90050004
    mov    r1, #0x90000000
    add    r1, r1, #0x50000
    mov    r2, #0x00
    str    r2, [r1, #0x4]

    // store 0x0b to address 0x90020014
    mov    r1, #0x90000000
    add    r1, r1, #0x20000
    mov    r2, #0x0b
    str    r2, [r1, #0x14]
```

Booting sequence (blob) 1/6

n Code is initially run from ROM
(or flash ROM) from address 0

n **After reset**

Core clock = 59MHz

Interrupt(IRQ, FIQ) is disabled

In supervisor mode

In ARM mode

Cache and MMU is off

n **Setup Interrupt Table**

vector location is fixed

n **Mask all interrupt sources**

those from Interrupt controller

n **Set higher core clock speed**

configure the PLL which is related to
CPU core

59MHz -> 221MHz

Total are 12 speeds, but there is
problem below 132.7MHz

```
_start : b   reset           //0x00
         b   undefined_instruction //0x04
         b   software_interrupt  //0x08
         b   abort_prefetch     //0x0c
         b   abort_data        //0x10
         b   not_used          //0x14
         b   irq               //0x18
         b   fiq               //0x1c
```

reset:

```
// store 0 to address 0x90050004
```

```
mov    r1, #0x90000000
add    r1, r1, #0x50000
mov    r2, #0x00
str    r2, [r1, #0x4]
```

Move 0x00 to
address
0x90050004

```
// store 0x0b to address 0x90020014
```

```
mov    r1, #0x90000000
add    r1, r1, #0x20000
mov    r2, #0x0b
str    r2, [r1, #0x14]
```

Booting sequence (blob) 1/6

n Code is initially run from ROM (or flash ROM) from address 0

n **After reset**

Core clock = 59MHz

Interrupt(IRQ, FIQ) is disabled

In supervisor mode

In ARM mode

Cache and MMU is off

n **Setup Interrupt Table**

vector location is fixed

n **Mask all interrupt sources**

those from Interrupt controller

n **Set higher core clock speed**

configure the PLL which is related to CPU core

59MHz -> 221MHz

Total are 12 speeds, but there is problem below 132.7MHz

```
_start : b   reset           //0x00
        b   undefined_instruction //0x04
        b   software_interrupt  //0x08
        b   abort_prefetch     //0x0c
        b   abort_data         //0x10
        b   not_used           //0x14
        b   irq                //0x18
        b   fiq                //0x1c
```

reset:

```
// store 0 to address 0x90050004
```

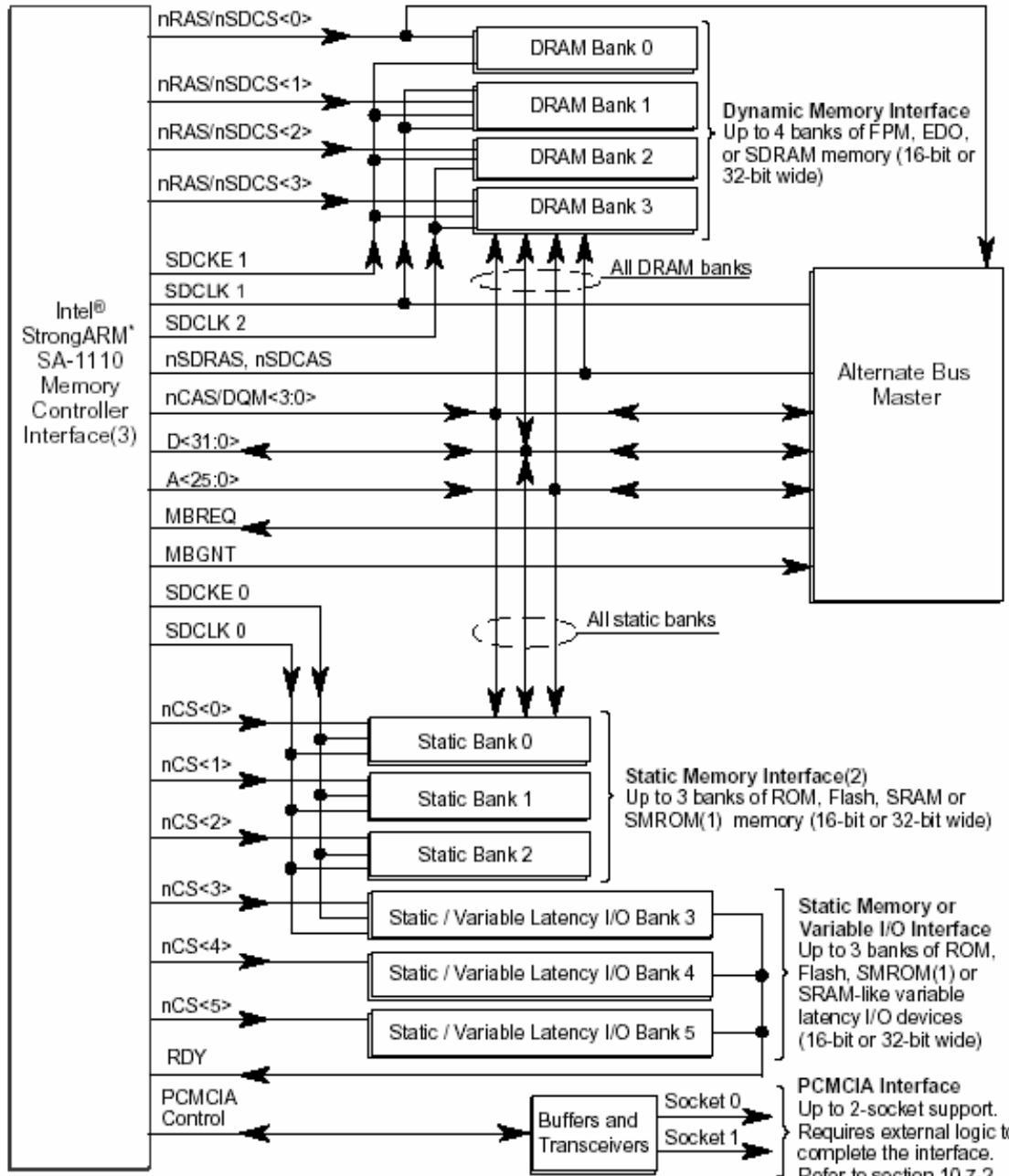
```
mov    r1, #0x90000000
add    r1, r1, #0x50000
mov    r2, #0x00
str    r2, [r1, #0x4]
```

Move 0x02 to
address
0x90050004

```
// store 0x0b to address 0x90020014
```

```
mov    r1, #0x90000000
add    r1, r1, #0x20000
mov    r2, #0x0b
str    r2, [r1, #0x14]
```

Move 0x0b to
address
0x90020014



Memory Interface Configuration

n 4 banks of DRAM

SDRAM/DRAM

n 6 banks of SRAM-like device

PROM/Flash ROM

SRAM

I/O device

n PCMCIA

Control area

Memory area

Timing Interpretations of Possible SDRAM/SMROM MDCAS Settings

Possible SDRAM/SMROM Settings for: MDCASn0[31:0] MDCASn1[31:0] MDCASn2[31:0]	SDRAM/SMROM Timing Interpretation		SDRAM LATCHING
	KnDB2 = 0	KnDB2 = 1	delayed or non-delayed read
0101 0101 0101 0101 0101 0101 0101 0111 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101	trcd=4*Tcpu tccd=2*Tcpu tac=2*Tcpu	trcd=4*Tcpu tccd=4*Tcpu tac=4*Tcpu	non-delayed read
1010 1010 1010 1010 1010 1010 1010 0111 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010	trcd=4*Tcpu tccd=2*Tcpu tac=3*Tcpu	trcd=4*Tcpu tccd=4*Tcpu tac=4*Tcpu	delayed read
0101 0101 0101 0101 0101 0101 0101 1111 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101	trcd=6*Tcpu tccd=2*Tcpu tac=2*Tcpu	Not Applicable	non-delayed read
1010 1010 1010 1010 1010 1010 1001 1111 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010	trcd=6*Tcpu tccd=2*Tcpu tac=3*Tcpu	Not Applicable	delayed read
0101 0101 0101 0101 0101 0101 0111 1111 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101 0101	trcd=8*Tcpu tccd=2*Tcpu tac=2*Tcpu	trcd=8*Tcpu tccd=4*Tcpu tac=4*Tcpu	non-delayed read
1010 1010 1010 1010 1010 1010 0111 1111 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010 1010	trcd=8*Tcpu tccd=2*Tcpu tac=3*Tcpu	trcd=8*Tcpu tccd=4*Tcpu tac=4*Tcpu	delayed read

Blob uses this Settings

Booting sequence (blob) 2/6

n Setup SDRAM parameter

Two banks are configured.

```
/* Set up the SDRAM controller*/  
mov r1, #0xA0000000 /*MDCNFG base address */
```

```
ldr r2, =0xAAAAAA7F  
str r2, [r1, #0x04] /* MDCAS00 */  
str r2, [r1, #0x20] /* MDCAS20 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x08] /* MDCAS01 */  
str r2, [r1, #0x24] /* MDCAS21 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x0C] /* MDCAS02 */  
str r2, [r1, #0x28] /* MDCAS22 */
```

```
ldr r2, =0x4dbc0327 /*MDREFR */  
str r2, [r1, #0x1C]
```

```
ldr r2, =0x72547254 /* MDCNFG */  
str r2, [r1, #0x00]
```

n Setup SRAM parameter

```
ldr r2, =0x4b90 /* MCS0 */  
orr r2,r2,r2,lsr #16  
str r2, [r1, #0x10]
```

```
ldr r2, =0x22212419 /* MCS1 */  
str r2, [r1, #0x14]
```

```
ldr r2, =0x42196669 /* MCS2 */  
str r2, [r1, #0x2C]
```

```
ldr r2, =0xafccafcc /* SMCNFG */  
str r2, [r1, #0x30]
```

Booting sequence (blob) 2/6

n Setup SDRAM parameter

Two banks are configured.

```
/* Set up the SDRAM controller*/  
mov r1, #0xA0000000 /*MDCNFG base address */
```

```
ldr r2, =0xAAAAAA7F  
str r2, [r1, #0x04] /* MDCAS00 */  
str r2, [r1, #0x20] /* MDCAS20 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x08] /* MDCAS01 */  
str r2, [r1, #0x24] /* MDCAS21 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x0C] /* MDCAS02 */  
str r2, [r1, #0x28] /* MDCAS22 */
```

```
ldr r2, =0x4dbc0327 /*MDREFR */  
str r2, [r1, #0x1C]
```

```
ldr r2, =0x72547254 /* MDCNFG */  
str r2, [r1, #0x00]
```

n Setup SRAM parameter

```
ldr r2, =0x4b90 /* MCS0 */  
orr r2,r2,r2,lsr #16  
str r2, [r1, #0x10]
```

```
ldr r2, =0x22212419 /* MCS1 */  
str r2, [r1, #0x14]
```

```
ldr r2, =0x42196669 /* MCS2 */  
str r2, [r1, #0x2C]
```

```
ldr r2, =0xafccafcc /* SMCNFG */  
str r2, [r1, #0x30]
```

Booting sequence (blob) 2/6

n Setup SDRAM parameter

Two banks are configured.

```
/* Set up the SDRAM controller*/  
mov r1, #0xA0000000 /*MDCNFG base address */
```

```
ldr r2, =0xAAAAAA7F  
str r2, [r1, #0x04] /* MDCAS00 */  
str r2, [r1, #0x20] /* MDCAS20 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x08] /* MDCAS01 */  
str r2, [r1, #0x24] /* MDCAS21 */
```

```
ldr r2, =0xAAAAAAA  
str r2, [r1, #0x0C] /* MDCAS02 */  
str r2, [r1, #0x28] /* MDCAS22 */
```

```
ldr r2, =0x4dbc0327 /*MDREFR */  
str r2, [r1, #0x1C]
```

```
ldr r2, =0x72547254 /* MDCNFG */  
str r2, [r1, #0x00]
```

n Setup SRAM parameter

MCS0 – Flash

MCS1 – SA-1111 extented flash

MCS2 – SA-1110 registers

```
ldr r2, =0x4b90 /* MCS0 */
```

```
orr r2,r2,r2,lsr #16  
str r2, [r1, #0x10]
```

```
ldr r2, =0x22212419 /* MCS1 */  
str r2, [r1, #0x14]
```

```
ldr r2, =0x42196669 /* MCS2 */  
str r2, [r1, #0x2C]
```

```
ldr r2, =0xafccafcc /* SMCNFG */  
str r2, [r1, #0x30]
```

UART serial port 1 registers

7	6	5	4	3	2	1	0
Reserved	TCE	RCE	SCE	DSS	SBS	OES	PE
Reset	?	?	?	?	?	?	?

Bits	Name	Description
0	PE	Parity enable. 0 – Parity checking on received data and parity generation on transmitted data is disabled. 1 – Parity checking on received data and parity generation on transmitted data is enabled.
1	OES	Odd/even parity select. 0 – Odd parity checking/generation selected. Parity error bit set if even number of ones counted in data field (including the parity bit). 1 – Even parity checking/generation selected. Parity error bit set if odd number of ones counted in data field (including the parity bit).
2	SBS	Stop bit select. 0 – One stop bit transmitted per frame. 1 – Two stop bits transmitted per frame. Note: Receiver not affected by SBS; always checks for one stop bit.
3	DSS	Data size select. 0 – 7-bit data. 1 – 8-bit data. Note: For 7-bit mode, the data is right justified within the FIFO entries, the MSBs in the receive FIFO are zero filled, and the MSBs in the transmit FIFO are ignored.
4	SCE	Sample clock enable. 0 – on-chip baud rate generator and digital PLL used to transmit and receive asynchronous data. 1 – A clock is input via GPIO pin 20 and is used synchronously to sample receive data and drive transmit data. Note: Serial port 1's UART uses GPIO pin 18 for the sample clock input; serial port 2 does not support the sample clock function. The user must also program the appropriate bits in the GPDR and GAFR registers within the system control module.
5	RCE	Receive clock edge select. 0 – Rising edge of clock input on GPIO pin 20 used to latch data from the receive pin if SCE=1. 1 – Falling edge of clock input on GPIO pin 20 used to latch data from the receive pin if SCE=1.
6	TCE	Transmit clock edge select. 0 – Rising edge of clock input on GPIO pin 20 used to drive data onto the transmit pin if SCE=1. 1 – Falling edge of clock input on GPIO pin 20 used to drive data onto the transmit pin if SCE=1.
7	—	Reserved.

Address: 0h 8002 0060 GPCLKR0 Read/Write

Bit	7	6	5	4	3	2	1	0
Reset	?	?	?	?	?	?	?	?

Bit	Name	Description
0	SUS	GPCLK/UART select. 0 – GPCLK mode selected. 1 – UART mode selected. Note: For SUS=0, TXD1 and RXD1 control is given to the PPC unit. If UPR is set in the PPC unit, SUS is ignored, the UART uses GPIO<14> to transmit and GPIO<15> to receive data, and serial port 1 defaults to GPLCK mode. The user must also program the GAFR and GPDR registers appropriately in the GPIO unit.
1	Reserved	Reserved for Future Expansion.
2	Reserved	Reserved for Future Expansion.
3	Reserved	Reserved for Future Expansion.
4	SCE	Sample clock enable. 0 – Sample clock disabled. 1 – Sample clock enabled.
5	SCD	Sample clock direction. 0 – If sample clock enabled, it is input on GPIO pin 16 and is not used. 1 – The sample clock which is generated within the GPCLK unit (the clock that is output after dividing the 3.6864 MHz reference by the programmable BRD field), is output to the GPIO pin 16 in the frequency range of 900 Hz – 3.6864MHz..
6	Reserved	Reserved for Future Expansion.
7	Reserved	Reserved for Future Expansion.

Booting sequence (blob) 3/6

n Setup Serial port

Assabet has port 1 and 3

Here we use port 1

port 1 has 2 mode:

GPCLK and UART

Set correct mode

Set baud rate = 9600 bps

No hardware flow control

```
// Serial port 1 on Assabet
```

```
ldr r0,=0x80020060 // GPCLK/UART select,select UART
```

```
mov r2,#1
```

```
str r2,[r0]
```

```
mov r1, #0x80000000
```

```
add r1, r1, #0x10000
```

```
mov r2, #0xFF
```

```
str r2, [r1, #0x1C]
```

```
...
```

```
// Set the serial port to sensible defaults: no break, no interrupt
```

```
// no parity, 8 databits, 1 stopbit. 8N1
```

```
mov r2, #0x00
```

```
str r2, [r1, #0x0C]
```

```
mov r2, #0x08
```

```
str r2, [r1, #0x00]
```

```
// Set BRD to 5, for a baudrate of 38k4 ([1] 11.11.4.1)
```

```
// Set BRD to 23, for a baudrate of 9k6 ([1] 11.11.4.1)
```

```
mov r2, #0x00
```

```
str r2, [r1, #0x04]
```

```
mov r2, #23
```

```
str r2, [r1, #0x08]
```

```
// Prepare to send some characters out in a loop
```

```
mov r1, #0x80000000
```

```
add r1, r1, #0x10000
```

```
// Enable the transmitter
```

```
mov r2, #0x02
```

```
str r2, [r1, #0x0C]
```

```
...
```

Booting sequence (blob) 3/6

n Setup Serial port

Assabet has port 1 and 3
Here we use port 1
port 1 has 2 mode:
GPCLK and UART
Set correct mode
Set baud rate = 9600 bps
No hardware flow control

```
// Serial port 1 on Assabet
ldr r0,=0x80020060 // GPCLK/UART select,select UART
mov r2,#1
str r2,[r0]
mov r1, #0x80000000
add r1, r1, #0x10000
mov r2, #0xFF
str r2, [r1, #0x1C]
...
// Set the serial port to sensible defaults: no break, no
interrupt
// no parity, 8 databits, 1 stopbit. 8N1
mov r2, #0x00
str r2, [r1, #0x0C]
mov r2, #0x08
str r2, [r1, #0x00]

// Set BRD to 5, for a baudrate of 38k4 ([1] 11.11.4.1)
// Set BRD to 23, for a baudrate of 9k6 ([1] 11.11.4.1)
mov r2, #0x00
str r2, [r1, #0x04]
mov r2, #23
str r2, [r1, #0x08]
// Prepare to send some characters out in a loop
mov r1, #0x80000000
add r1, r1, #0x10000
// Enable the transmitter
mov r2, #0x02
str r2, [r1, #0x0C]
...
```



Clear Status
Register

Booting sequence (blob) 3/6

n Setup Serial port

Assabet has port 1 and 3
Here we use port 1
port 1 has 2 mode:
GPCLK and UART
Set correct mode
Set baud rate = 9600 bps
No hardware flow control

```
// Serial port 1 on Assabet
ldr r0,=0x80020060 // GPCLK/UART select,select UART
mov r2,#1
str r2,[r0]
mov r1, #0x80000000
add r1, r1, #0x10000
mov r2, #0xFF
str r2, [r1, #0x1C]
...
// Set the serial port to sensible defaults: no break, no
// interrupt
// no parity, 8 databits, 1 stopbit. 8N1
mov r2, #0x00
str r2, [r1, #0x0C]
mov r2, #0x08
str r2, [r1, #0x00]

// Set BRD to 5, for a baudrate of 38k4 ([1] 11.11.4.1)
// Set BRD to 23, for a baudrate of 9k6 ([1] 11.11.4.1)
mov r2, #0x00
str r2, [r1, #0x04]
mov r2, #23
str r2, [r1, #0x08]
// Prepare to send some characters out in a loop
mov r1, #0x80000000
add r1, r1, #0x10000
// Enable the transmitter
mov r2, #0x02
str r2, [r1, #0x0C]
...
```



Disable
Tx/Rx

Booting sequence (blob) 3/6

n Setup Serial port

Assabet has port 1 and 3

Here we use port 1

port 1 has 2 mode:

GPCLK and UART

Set correct mode

Set baud rate = 9600 bps

No hardware flow control

```
// Serial port 1 on Assabet
```

```
ldr r0,=0x80020060 // GPCLK/UART select,select UART
```

```
mov r2,#1
```

```
str r2,[r0]
```

```
mov r1, #0x80000000
```

```
add r1, r1, #0x10000
```

```
mov r2, #0xFF
```

```
str r2, [r1, #0x1C]
```

```
...
```

```
// Set the serial port to sensible defaults: no break, no interrupt
```

```
// no parity, 8 databits, 1 stopbit. 8N1
```

```
mov r2, #0x00
```

```
str r2, [r1, #0x0C]
```

```
mov r2, #0x08
```

```
str r2, [r1, #0x00]
```

```
// Set BRD to 5, for a baudrate of 38k4 ([1] 11.11.4.1)
```

```
// Set BRD to 23, for a baudrate of 9k6 ([1] 11.11.4.1)
```

```
mov r2, #0x00
```

```
str r2, [r1, #0x04]
```

```
mov r2, #23
```

```
str r2, [r1, #0x08]
```

```
// Prepare to send some characters out in a loop
```

```
mov r1, #0x80000000
```

```
add r1, r1, #0x10000
```

```
// Enable the transmitter
```

```
mov r2, #0x02
```

```
str r2, [r1, #0x0C]
```

```
...
```

Booting sequence (blob) 3/6

n Setup Serial port

Assabet has port 1 and 3
Here we use port 1
port 1 has 2 mode:
GPCLK and UART
Set correct mode
Set baud rate = 9600 bps
No hardware flow control

```
// Serial port 1 on Assabet
ldr  r0,=0x80020060 // GPCLK/UART select,select UART
mov  r2,#1
str  r2,[r0]
mov  r1, #0x80000000
add  r1, r1, #0x10000
mov  r2, #0xFF
str  r2, [r1, #0x1C]
...
// Set the serial port to sensible defaults: no break, no
// interrupt
// no parity, 8 databits, 1 stopbit. 8N1
mov  r2, #0x00
str  r2, [r1, #0x0C]
mov  r2, #0x08
str  r2, [r1, #0x00]

// Set BRD to 5, for a baudrate of 38k4 ([1] 11.11.4.1)
// Set BRD to 23, for a baudrate of 9k6 ([1] 11.11.4.1)
mov  r2, #0x00
str  r2, [r1, #0x04]
mov  r2, #23
str  r2, [r1, #0x08]
// Prepare to send some characters out in a loop
mov  r1, #0x80000000
add  r1, r1, #0x10000
// Enable the transmitter
mov  r2, #0x02
str  r2, [r1, #0x0C]
...
```

Enable
Tx

Booting sequence (blob) 4/6

- n Prepare kernel image space
- n Working area =
0xC0000000 ~ 0xD0000000
(total is 256M bytes, large enough !)
kernel address = 0xc0008000
ramdisk address = 0xc0800000

Clean (write 0 to) all address

- n stmia
store multiple and increment after immediately
stmia r4!, {r0-r3,r7-r10}
r4 = start address
! = store the changed start address back
store the content of r0 to start address, the
content of r1 to start address+1, and vice
versa. Finally, add 32 to r4 and restore.

```
test_mem:
    ...
    mov    r4, #0xC0000000 //Start Address
    mov    r5, #0xD0000000 //Stop Address
    mov    r6, #0x400000   //Step size
    sub    r6, r6, #0x01
    ...
zero_loop:
    mov    r0, #0x0
    mov    r1, #0x0
    mov    r2, #0x0
    mov    r3, #0x0
    mov    r7, #0x0
    mov    r8, #0x0
    mov    r9, #0x0
    mov    r10, #0x0
zero_loop1:
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    tst    r4, r6
    bne    zero_loop1
    ...
    cmp    r4, r5
    blt    zero_loop
    ...
    mov    r4, #0xC0000000
    mov    r5, #0xD0000000
    mov    r6, r4 // Base for current block
    mov    r7, #0x0 // Diff between r4 and [r4]
```

Booting sequence (blob) 4/6

- n Prepare kernel image space
- n Working area =
0xC0000000 ~ 0xD0000000
(total is 256M bytes, large enough !)
kernel address = 0xc0008000
ramdisk address = 0xc0800000

Clean (write 0 to) all address

- n stmia
store multiple and increment after immediately
stmia r4!, {r0-r3,r7-r10}
r4 = start address
! = store the changed start address back
store the content of r0 to start address, the
content of r1 to start address+1, and vice
versa. Finally, add 32 to r4 and restore.

```

test_mem:
    ...
    mov    r4, #0xC0000000 //Start Address
    mov    r5, #0xD0000000 //Stop Address
    mov    r6, #0x400000   //Step size
    sub    r6, r6, #0x01
    ...
zero_loop:
    mov    r0, #0x0
    mov    r1, #0x0
    mov    r2, #0x0
    mov    r3, #0x0
    mov    r7, #0x0
    mov    r8, #0x0
    mov    r9, #0x0
    mov    r10, #0x0
zero_loop1:
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    tst    r4, r6
    bne    zero_loop1
    ...
    cmp    r4, r5
    blt    zero_loop
    ...
    mov    r4, #0xC0000000
    mov    r5, #0xD0000000
    mov    r6, r4 // Base for current block
    mov    r7, #0x0 // Diff between r4 and [r4]
    
```

Store 8 words
each time

128 bytes
each time

Booting sequence (blob) 4/6

- n Prepare kernel image space
- n Working area =
0xC0000000 ~ 0xD0000000
(total is 256M bytes, large enough !)
kernel address = 0xc0008000
ramdisk address = 0xc0800000

Clean (write 0 to) all address

- n stmia
store multiple and increment after immediately
stmia r4!, {r0-r3,r7-r10}
r4 = start address
! = store the changed start address back
store the content of r0 to start address, the
content of r1 to start address+1, and vice
versa. Finally, add 32 to r4 and restore.

```
test_mem:
    ...
    mov    r4, #0xC0000000 //Start Address
    mov    r5, #0xD0000000 //Stop Address
    mov    r6, #0x400000   //Step size
    sub    r6, r6, #0x01
    ...
zero_loop:
    mov    r0, #0x0
    mov    r1, #0x0
    mov    r2, #0x0
    mov    r3, #0x0
    mov    r7, #0x0
    mov    r8, #0x0
    mov    r9, #0x0
    mov    r10, #0x0
zero_loop1:
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    stmia r4!, {r0-r3, r7-r10}
    tst    r4, r6
    bne    zero_loop1
    ...
    cmp    r4, r5
    blt    zero_loop
    ...
    mov    r4, #0xC0000000
    mov    r5, #0xD0000000
    mov    r6, r4 // Base for current block
    mov    r7, #0x0 // Diff between r4 and [r4]
```

Booting sequence (blob) 5/6

```
/* Go hunting for aliases */
alias_loop:
    ldr    r8, [r4]
    cmp   r8, #0x0
    streq r4, [r4]      // If the value was zero, load r4 to it
    moveq r8, r4
    sub   r9, r4, r8
    cmp   r9, r7      // Is this a new block ?
    bne   alias_detected
    add   r4, r4, #0x1000
    cmp   r4, r5
    blt   alias_loop
    b     alias_done

alias_detected:
    sub   r8, r6, r7
    cmp   r7, #0x0    // Is this a 'new' block ?
    moveq r10, r6     // If so, remember the start of this block...
    moveq r11, r4     // ...and the start of the next block
    mov   r6, r4
    mov   r7, r9
    beq   alias_loop
    b     alias_loop

alias_done:
    sub   sp, r11, #0x04    // Set up the stack pointer

/* Arg 0 of the C code is the start of the block it can use; */
/* arg 1 is the size of that block. */
    mov   r0, r10
    sub   r1, r11, r10

jump_to_c:
    bl    c_main          // Jump to the C code
```

- n b : branch to the target address
- bl : branch and save the return address (the original address) in R14
- blt : branch if less than

Booting sequence (blob) 5/6

```
/* Go hunting for aliases */
alias_loop:
    ldr    r8, [r4]
    cmp   r8, #0x0
    streq r4, [r4]           // If the value was zero, load r4 to it
    moveq r8, r4
    sub   r9, r4, r8
    cmp   r9, r7           // Is this a new block ?
    bne   alias_detected
    add   r4, r4, #0x1000
    cmp   r4, r5
    blt   alias_loop
    b     alias_done

alias_detected:
    sub   r8, r6, r7
    cmp   r7, #0x0        // Is this a 'new' block ?
    moveq r10, r6         // If so, remember the start of this block...
    moveq r11, r4         // ...and the start of the next block
    mov   r6, r4
    mov   r7, r9
    beq   alias_loop
    b     alias_loop

alias_done:
    sub   sp, r11, #0x04   // Set up the stack pointer

/* Arg 0 of the C code is the start of the block it can use; */
/* arg 1 is the size of that block. */
    mov   r0, r10
    sub   r1, r11, r10

jump_to_c:
    bl    c_main          // Jump to the C code
```

- n b : branch to the target address
- bl : branch and save the return address (the original address) in R14
- blt : branch if less than

Booting sequence (blob) 5/6

```
/* Go hunting for aliases */
alias_loop:
  ldr    r8, [r4]
  cmp    r8, #0x0
  streq  r4, [r4]           // If the value was zero, load r4 to it
  moveq  r8, r4
  sub    r9, r4, r8
  cmp    r9, r7           // Is this a new block ?
  bne    alias_detected
  add    r4, r4, #0x1000
  cmp    r4, r5
  blt    alias_loop
  b      alias_done
```

- n b : branch to the target address
- bl : branch and save the return address (the original address) in R14
- blt : branch if less than

```
alias_detected:
  sub    r8, r6, r7
  cmp    r7, #0x0 // Is this a 'new' block ?
  moveq  r10, r6 // If so, remember the start of this block...
  moveq  r11, r4 // ...and the start of the next block
  mov    r6, r4
  mov    r7, r9
  beq    alias_loop
  b      alias_loop
```

```
alias_done:
  sub    sp, r11, #0x04 // Set up the stack pointer
```

```
/* Arg 0 of the C code is the start of the block it can use; */
/* arg 1 is the size of that block. */
  mov    r0, r10
  sub    r1, r11, r10
```

```
jump_to_c:
  bl     c_main // Jump to the C code
```

Booting sequence (blob) 6/6

n What does the other C codes of Blob do ?

Re-initialize serial port
Set a Timer for event control
Parse command
download a kernel
download a ramdisk
boot the kernel

```
void Download(char .....)  
{  
    ...  
    /* download kernel */  
    startAddress = KERNEL_RAM_BASE;  
    bufLen = status->blockSize -  
            KERNEL_BLOCK_OFFSET;  
    numRead = &status->kernelSize;  
    status->kernelType = fromDownload;  
    ...  
    /* download ramdisk */  
    startAddress = RAMDISK_RAM_BASE;  
    bufLen = status->blockSize -  
            RAMDISK_BLOCK_OFFSET;  
    numRead = &status->ramdiskSize;  
    status->ramdiskType = fromDownload;  
    ...  
    *numRead = UUDecode((char *)startAddress,  
                        bufLen);  
    ...  
}  
  
.....  
void BootKernel(char *commandline)  
{  
    void (*theKernel)(int zero, int arch) = (void (*)(int,  
int))KERNEL_RAM_BASE;  
    .....  
    SerialOutputString("\rStarting kernel ... \r\r");  
    .....  
    theKernel(0, 25);  
}
```

0xc0008000

0xc0800000

Where does the Kernel start ?

n GZIP compression

SA-1110

ZTEXTADDR = 0xc0008000

~~ZRELADDR = 0xc0008000~~

ZRELADDR = 0xc0208000 (if with SA-1111)

Vmlinux : kernel image

Decompression routine : head.S + misc.c

Piggy.o = compressed vmlinux

convert vmlinux to raw binary format

use gzip to compress

link to the relocatable binary format

zImage = head.o+misc.o+piggy.o

n MISC.c uses flate.c to decompress kernel.

Start from 0xc0208000 (head.S)

start:

```
.type start,#function
.rept 8
mov r0, r0
.endr
```

8 nop

```
b 1f
.word 0x016f2818 @ Magic numbers to help the loader
.word start @ absolute load/run zImage address
.word _edata @ zImage end address
1: mov r7, r1 @ save architecture ID
mov r8, #0 @ save r0
```

```
.text
1: adr r2, LC0
ldmia r2, {r2, r3, r4, r5, sp}
```

```
mov r0, #0
1: str r0, [r2], #4 @clear bss
str r0, [r2], #4
str r0, [r2], #4
str r0, [r2], #4
```

```
cmp r2, r3
blt 1b
mrc p15, 0, r6, c0, c0 @ get processor ID
bl cache_on
```

```
mov r1, sp @ malloc space above stack
add r2, sp, #0x10000 @ 64k max
```

```
teq r4, r5 @ will we overwrite ourselves?
moveq r5, r2 @ decompress after image
movne r5, r4 @ decompress to final location
mov r0, r5
```

```
mov r3, r7
bl SYMBOL_NAME(decompress_kernel)
```

n mrc : coprocessor instruction
mrc p15,0,r6,c0,c0
read the value of coprocessor's
register,C0,to r6

n mcr : coprocessor instruction
store value to coprocessor
registers

Start from 0xc0208000 (head.S)

start:

```
.type start,#function
.rept 8
mov r0, r0
.endr
```

```
b 1f
```

```
.word 0x016f2818 @ Magic numbers to help the loader
```

```
.word start @ absolute load/run zImage address
```

```
.word _edata @ zImage end address
```

```
1: mov r7, r1 @ save architecture ID
```

```
mov r8, #0 @ save r0
```

```
.text
```

```
1: adr r2, LC0
```

```
ldmia r2, {r2, r3, r4, r5, sp}
```

Clear BSS
segment

```
mov r0, #0
```

```
1: str r0, [r2], #4
```

@clear bss

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
cmp r2, r3
```

```
blt 1b
```

```
mrc p15, 0, r6, c0, c0 @ get processor ID
```

```
bl cache_on
```

```
mov r1, sp @ malloc space above stack
```

```
add r2, sp, #0x10000 @ 64k max
```

```
teq r4, r5 @ will we overwrite ourselves?
```

```
moveq r5, r2 @ decompress after image
```

```
movne r5, r4 @ decompress to final location
```

```
mov r0, r5
```

```
mov r3, r7
```

```
bl SYMBOL_NAME(decompress_kernel)
```

n mrc : coprocessor instruction
mrc p15,0,r6,c0,c0
read the value of coprocessor's
register,C0,to r6

n mcr : coprocessor instruction
store value to coprocessor
registers

Start from 0xc0208000 (head.S)

start:

```
.type start,#function
.rept 8
mov r0, r0
.endr
```

```
b 1f
```

```
.word 0x016f2818 @ Magic numbers to help the loader
```

```
.word start @ absolute load/run zImage address
```

```
.word _edata @ zImage end address
```

```
1: mov r7, r1 @ save architecture ID
```

```
mov r8, #0 @ save r0
```

```
.text
```

```
1: adr r2, LC0
```

```
ldmia r2, {r2, r3, r4, r5, sp}
```

```
mov r0, #0
```

```
1: str r0, [r2], #4
```

@clear bss

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
cmp r2, r3
```

```
blt 1b
```

```
mrc p15, 0, r6, c0, c0 @ get processor ID
```

```
bl cache_on
```

```
mov r1, sp @ malloc space above stack
```

```
add r2, sp, #0x10000 @ 64k max
```

```
teq r4, r5 @ will we overwrite ourselves?
```

```
moveq r5, r2 @ decompress after image
```

```
movne r5, r4 @ decompress to final location
```

```
mov r0, r5
```

```
mov r3, r7
```

```
bl SYMBOL_NAME(decompress_kernel)
```

n mrc : coprocessor instruction
mrc p15,0,r6,c0,c0
read the value of coprocessor's
register,C0,to r6

n mcr : coprocessor instruction
store value to coprocessor
registers

Enable cache
and ready to do
decompression

Start from 0xc0208000 (head.S)

start:

```
.type start,#function
.rept 8
mov r0, r0
.endr
```

```
b 1f
```

```
.word 0x016f2818 @ Magic numbers to help the loader
```

```
.word start @ absolute load/run zImage address
```

```
.word _edata @ zImage end address
```

```
1: mov r7, r1 @ save architecture ID
```

```
mov r8, #0 @ save r0
```

```
.text
```

```
1: adr r2, LC0
```

```
ldmia r2, {r2, r3, r4, r5, sp}
```

```
mov r0, #0
```

```
1: str r0, [r2], #4 @clear bss
```

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
str r0, [r2], #4
```

```
cmp r2, r3
```

```
blt 1b
```

```
mrc p15, 0, r6, c0, c0 @ get processor ID
```

```
bl cache_on
```

```
mov r1, sp @ malloc space above stack
```

```
add r2, sp, #0x10000 @ 64k max
```

```
teq r4, r5 @ will we overwrite ourselves?
```

```
moveq r5, r2 @ decompress after image
```

```
movne r5, r4 @ decompress to final location
```

```
mov r0, r5
```

```
mov r3, r7
```

```
bl SYMBOL_NAME(decompress_kernel)
```

n mrc : coprocessor instruction
mrc p15,0,r6,c0,c0
read the value of coprocessor's
register,C0,to r6

n mcr : coprocessor instruction
store value to coprocessor
registers

Start from 0xc0208000 (head.S)

```
teq  r4, r5      @ do we need to relocate
beq  call_kernel @ the kernel?

add  r0, r0, #127
bic  r0, r0, #127 @ align the kernel length
/*
 * r0  = decompressed kernel length
 * r1-r3 = unused
 * r4  = kernel execution address
 * r5  = decompressed kernel start
 * r6  = processor ID
 * r7  = architecture ID
 * r8-r14 = unused
 */
add  r1, r5, r0 @ end of decompressed kernel
adr  r2, reloc_start
adr  r3, reloc_end
1: ldmia r2!, {r8 - r13} @ copy relocation code
   stmia r1!, {r8 - r13}
   ldmia r2!, {r8 - r13}
   stmia r1!, {r8 - r13}
   cmp  r2, r3
   blt  1b

bl   cache_clean_flush
add  pc, r5, r0 @ call relocation code

.type LC0, #object
LC0: .word  __bss_start
     .word  _end
     .word  _load_addr
     .word  _start
     .word  user_stack+4096
     .size LC0, . - LC0
```

```
.align
.section ".stack"
user_stack: .space 4096
```

Cache_on (1/2)

cache_on:

```
ldr    r1, proc_sa110_type
eor    r1, r1, r6
movs  r1, r1, lsr #5  @ catch SA110 and SA1100
beq   1f
ldr    r1, proc_sa1110_type
eor    r1, r1, r6
movs  r1, r1, lsr #4
movne pc, lr
```

1:

```
sub    r3, r4, #16384      @ Page directory size
bic    r3, r3, #0xff      @ Align the pointer
bic    r3, r3, #0x3f00
mov    r0, r3
mov    r8, r0, lsr #18
mov    r8, r8, lsl #18    @ start of RAM
add    r9, r8, #0x10000000 @ a reasonable RAM size
mov    r1, #0x12
orr    r1, r1, #3 << 10
add    r2, r3, #16384
```

```
1: cmp    r1, r8            @ if virt > start of RAM
orrge  r1, r1, #0x0c      @ set cacheable, bufferable
cmp    r1, r9            @ if virt > end of RAM
bicge  r1, r1, #0x0c      @ clear cacheable, bufferable
str    r1, [r0], #4      @ 1:1 mapping
add    r1, r1, #1048576
teq    r0, r2
bne   1b
```

Cache_on (2/2)

```
mov r1, #0x1e
orr r1, r1, #3 << 10
mov r2, pc, lsr #20
orr r1, r1, r2, lsl #20
add r0, r3, r2, lsl #2
str r1, [r0], #4
add r1, r1, #1048576
str r1, [r0]
```

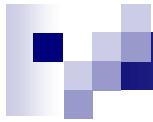
```
mov r0, #0
mcr p15, 0, r0, c7, c10, 4 @ drain write buffer
mcr p15, 0, r0, c8, c7 @ flush I,D TLBs
mcr p15, 0, r3, c2, c0 @ load page table pointer
mov r0, #-1
mcr p15, 0, r0, c3, c0 @ load domain access register
mrc p15, 0, r0, c1, c0
orr r0, r0, #0x1000 @ I-cache enable
mcr p15, 0, r0, c1, c0
mov pc, lr
```

Cache_off and Cache_clean_flush

```
cache_off:
    ldr    r1, proc_sa110_type
    eor    r1, r1, r6
    movs  r1, r1, lsr #5           @ catch SA110 and SA1100
    beq    1f
    ldr    r1, proc_sa1110_type
    eor    r1, r1, r6
    movs  r1, r1, lsr #4
    movne pc, lr
1: mrc    p15, 0, r0, c1, c0
    bic    r0, r0, #0x000d
    mcr    p15, 0, r0, c1, c0
    mov    pc, lr

cache_clean_flush:
    ldr    r1, proc_sa110_type
    eor    r1, r1, r6
    movs  r1, r1, lsr #5           @ catch SA110 and SA1100
    beq    1f
    ldr    r1, proc_sa1110_type
    eor    r1, r1, r6
    movs  r1, r1, lsr #4
    movne pc, lr
1: bic    r1, pc, #31
    add    r2, r1, #32768
1: ldr    r12, [r1], #32           @ s/w flush D cache
    teq    r1, r2
    bne    1b

    mcr    p15, 0, r1, c7, c7, 0 @ flush I cache
    mcr    p15, 0, r1, c7, c10, 4 @ drain WB
    mov    pc, lr
```



Decompress_kernel

- n Gunzip is used for decompression
gunzip() is in flate.c

- n Passing parameters
parameters from head.S
output_start = content of r0
free_mem_ptr_p = content of r1
free_mem_ptr_end_p = content of r2
arch_id = r3

```
decompress_kernel(ulg output_start, ulg
                  free_mem_ptr_p, ulg free_mem_ptr_end_p,
                  int arch_id)
{
    output_data    = (uch *)output_start;
                  /* Points to kernel start */
    free_mem_ptr   = free_mem_ptr_p;
    free_mem_ptr_end = free_mem_ptr_end_p;
    __machine_arch_type = arch_id;

    proc_decomp_setup();
    arch_decomp_setup();

    makecrc();
    puts("Uncompressing Linux...");
    gunzip();
    puts(" done, booting the kernel.\n");
    return output_ptr;
}
```

Call kernel

```
/*
 * This code is relocatable. It is relocated by the above code to the end
 * of the kernel and executed there. During this time, we have no stacks.
 *
 * r0   = decompressed kernel length
 * r1-r3 = unused
 * r4   = kernel execution address
 * r5   = decompressed kernel start
 * r6   = processor ID
 * r7   = architecture ID
 * r8-r14 = unused
 */
    .align 5
reloc_start:
    add    r8, r5, r0
    debug_reloc_start
    mov    r1, r4
1: .rept 4
    ldmia r5!, {r0, r2, r3, r9 - r13} @ relocate kernel
    stmia r1!, {r0, r2, r3, r9 - r13}
    .endr
    cmp   r5, r8
    blt   1b
    debug_reloc_end
call_kernel:
    bl    cache_clean_flush
    bl    cache_off
    mov   r0, #0
    mov   r1, r7 @ restore architecture number
    mov   pc, r4 @ call kernel
```

Head-armv.S

- n 26 bit v.s. 32 bit
26 bit is old version

We don't use 26 bits mode here.
CONFIG_CPU_26 should be disabled.
Head-armv.S : 32 bit header codes
Head-armo.S : 26 bit header codes

```
ENTRY(stext)
    mov    r12, r0
    mov    r0, #F_BIT | I_BIT | MODE_SVC
                                     @ make sure svc mode
                                     @ and all irqs disabled
    msr    cpsr_c, r0
    bl     __lookup_processor_type
    teq    r10, #0
                                     @ invalid processor?
    moveq  r0, #'p'
                                     @ yes, error 'p'
    beq    __error
    bl     __lookup_architecture_type
    teq    r7, #0
                                     @ invalid architecture?
    moveq  r0, #'a' @ yes, error 'a'
    beq    __error
    bl     __create_page_tables
    adr    lr, __ret
                                     @ return address
    add    pc, r10, #12
                                     @ initialise processor
                                     @ (return control reg)
```

```
__switch_data:
    .long  __mmap_switched
    .long  SYMBOL_NAME(compat)
    .long  SYMBOL_NAME(__bss_start)
    .long  SYMBOL_NAME(_end)
    .long  SYMBOL_NAME(processor_id)
    .long  SYMBOL_NAME(__machine_arch_type)
    .long  SYMBOL_NAME(cr_alignment)
    .long  SYMBOL_NAME(init_task_union)+8192
```

```
__ret:
    ldr    lr, __switch_data
    mcr    p15, 0, r0, c1, c0
    mov    r0, r0
    mov    r0, r0
    mov    r0, r0
    mov    pc, lr
```

Head-armv.S

```

__lookup_processor_type:
    adr    r5, 2f
    ldmia  r5, {r7, r9, r10}
    sub    r5, r5, r10
    add    r7, r7, r5
    add    r10, r9, r5
    mrc    p15, 0, r9, c0, c0
1:  ldmia  r10, {r5, r6, r8}
    and    r6, r6, r9
    teq    r5, r6
    moveq  pc, lr
    add    r10, r10, #36
    cmp    r10, r7
    blt    1b
    mov    r10, #0
    mov    pc, lr

__lookup_architecture_type:
    adr    r4, 2b
    ldmia  r4, {r2, r3, r5, r6, r7}
    sub    r5, r4, r5
    add    r4, r6, r5
    add    r7, r7, r5
1:  ldr    r5, [r4]
    teq    r5, r1
    beq    2f
    add    r4, r4, #SIZEOF_MACHINE_DESC
    cmp    r4, r7
    blt    1b
    mov    r7, #0
    mov    pc, lr
2:  ldmiB  r4, {r5, r6, r7}
    mov    r7, r7, lsr #18
    mov    pc, lr

```

2: .long __proc_info_end
 .long
 __proc_info_begin
 .long 2b
 .long __arch_info_begin
 .long __arch_info_end

@ to our address space

@ get processor id
 @ value, mask, mmuflags
 @ mask wanted bits

@ sizeof(proc_info_list)

@ unknown processor

@ throw away r2, r3
 @ convert addresses
 @ to our address space

@ get machine type

@ unknown architecture

@ found, get results
 @ pagetable byte offset

Head-armv.S

```
__mmap_switched:
    adr    r3, __switch_data + 4
    ldmia  r3, {r2, r4, r5, r6, r7, r8, sp}  @ r2 = compat
                                              @ sp = stack pointer

    str    r12, [r2]

    mov    fp, #0      @ Clear BSS (and zero fp)
1:  cmp    r4, r5
    strcc  fp, [r4], #4
    bcc    1b

    str    r9, [r6]    @ Save processor ID
    str    r1, [r7]    @ Save machine type
    bic    r2, r0, #2  @ Clear 'A' bit
    stmia  r8, {r0, r2} @ Save control register values
    b      SYMBOL_NAME(start_kernel)
```

Head-armv.S

```
__create_page_tables:
    pgtbl r4
    mov r0, r4
    mov r3, #0
    add r2, r0, #0x4000 @ 16k of page table
1: str r3, [r0], #4 @ Clear page table
    str r3, [r0], #4
    str r3, [r0], #4
    teq r0, r2
    bne 1b
    adr r2, SYMBOL_NAME(stext) @ get kernel phys start
    bic r2, r2, #0x000ff000 @ aligned down to a MB
    add r3, r8, r2 @ add mmu flags
    add r0, r4, r2, lsr #18
    str r3, [r0] @ identity mapping
    sub r2, r2, r5 @ kernel offset in RAM
    ldr r0, krnl_idx @ kernel pgtable idx
    sub r0, r0, r2, lsr #18 @ PAGE_OFFSET pgtable idx
    add r0, r0, r4 @ offset into pgtable
    add r2, r8, r5 @ phys mem + mmu flags
    add r3, r3, #4 << 20 @ kernel start + 4MB
1: str r2, [r0], #4 @ store mapping
    add r2, r2, #1 << 20
    teq r2, r3
    bne 1b

    bic r8, r8, #0x0c @ turn off cacheable
    @ and bufferable bits

    mov pc, lr
```

```
.macro pgtbl, reg
    adr \reg, stext
    sub \reg, \reg, #0x4000
.endm
```



[Reference]

- n ARM Architecture Reference Manual
- n Intel StrongARM SA-1110 Microprocessor Developer's Manual
- n ARM-Linux(www.arm.linux.org.uk) mailing list