How to boot Linux on your SOC boards

Liang Yan (lyan)
SUSE Virtualization Team
11/14/2018

Outline

1. Background
2. Hardware
3. Software
4. Further imagination
5. Q&A

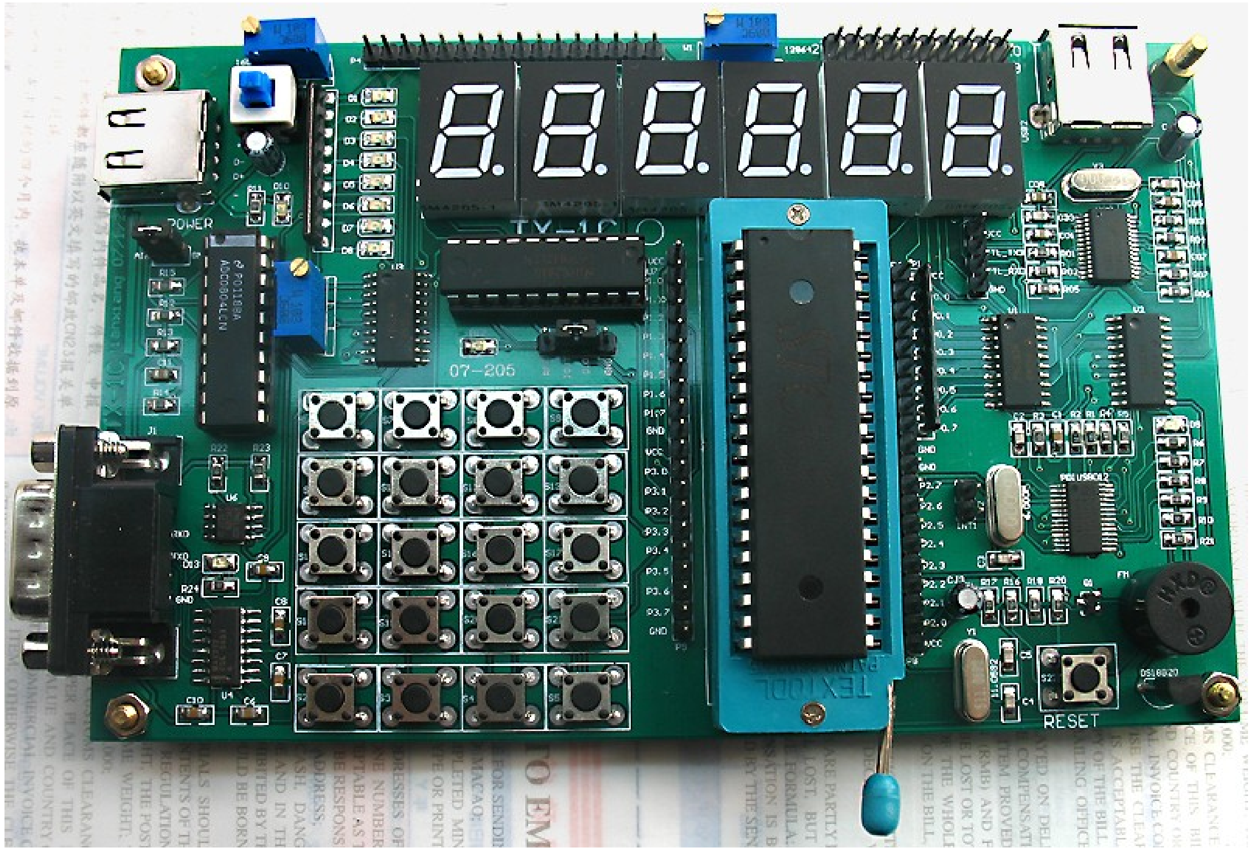
Background

SOC boards

- stc89c52

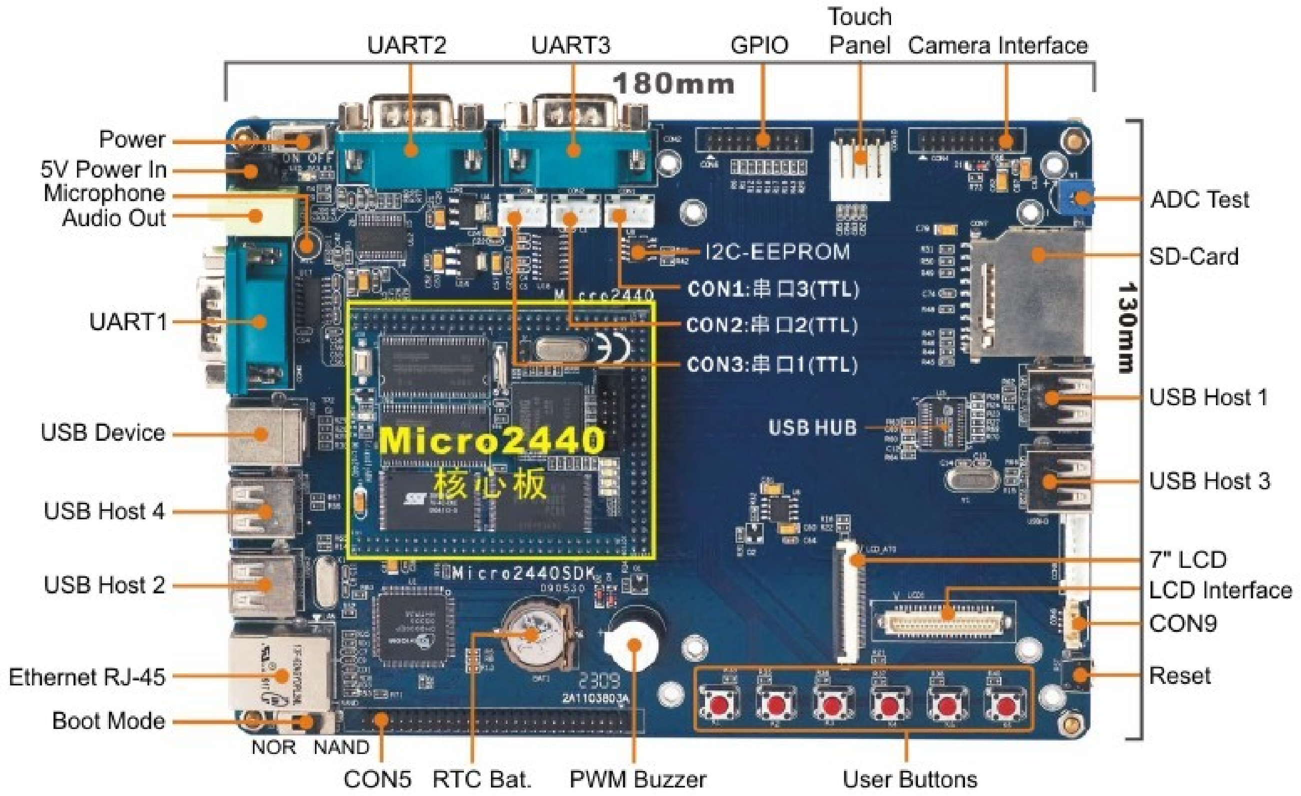
- ARM V7
  Micro 2440 board

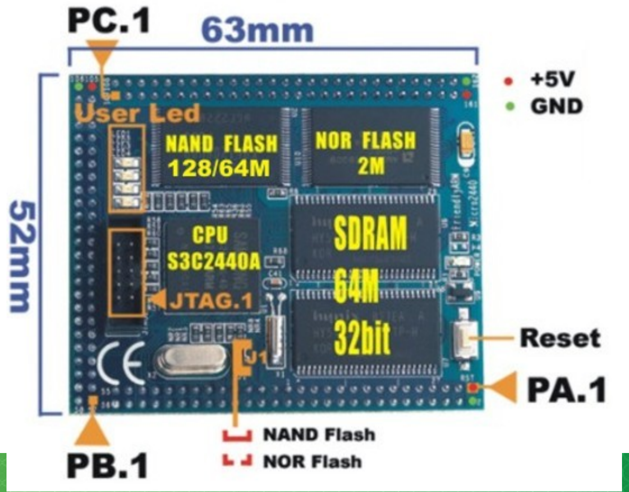
- ARM V8(AARCH64)
  Raspberry pi board



SOC boards

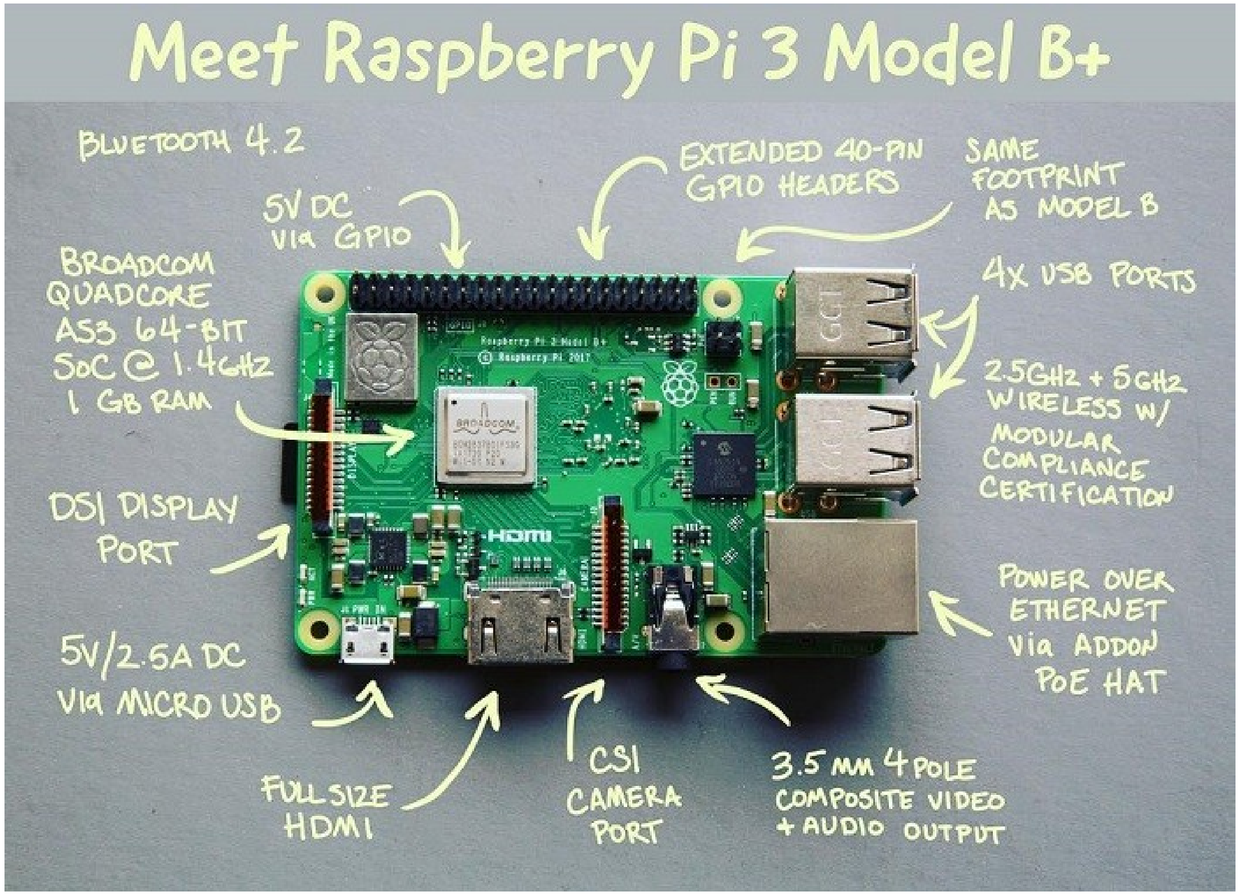
- stc89c52 (ISP/IAP)
- ARM V7 Micro 2440 board (supervivi/uboot)
- ARM V8(AARCH64) Raspberry pi board (uboot/uefi)





SOC boards

- stc89c52rc
- ARM V7
  Micro 2440 board
- ARM V8(AARCH64)
  Raspberry pi board


Meet Raspberry Pi 3 Model B+
BLUETOOTH 4.2
5V DC VIA GPIO
EXTENDED 40-PIN GPIO HEADERS
SAME FOOTPRINT AS MODEL B
BROADCOM QUADCORE A53 64-BIT SoC @ 1.4GHz 1 GB RAM
4X USB PORTS
2.5GHz + 5GHz WIRELESS W/ MODULAR COMPLIANCE CERTIFICATION
DSI DISPLAY PORT
POWER OVER ETHERNET VIA ADDON PoE HAT
5V/2.5A DC VIA MICRO USB
FULL SIZE HDMI
CSI CAMERA PORT
3.5 MM 4 POLE COMPOSITE VIDEO + AUDIO OUTPUT

Bootloader:

VIVI
SuperVIVI
U-Boot
Fastboot
Grub2
UEFI

Kernel:

Linux
Windows
Mac OS
BSD unix

Vxworks
uc-os

What should we look
If we want boot a hardware

SOC boards

- CPU(SOC)

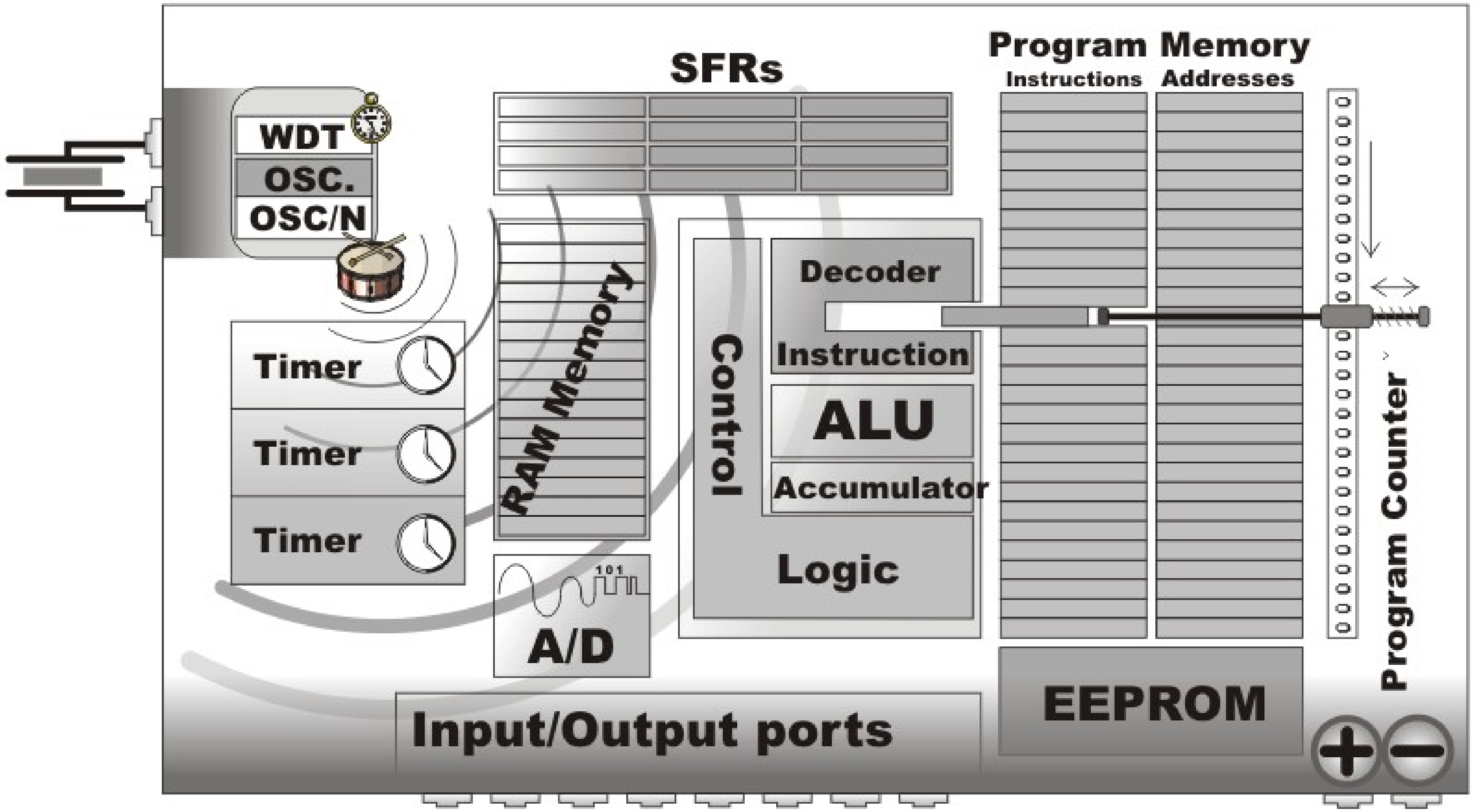
- Storage media

ROM/PROM/EEPROM

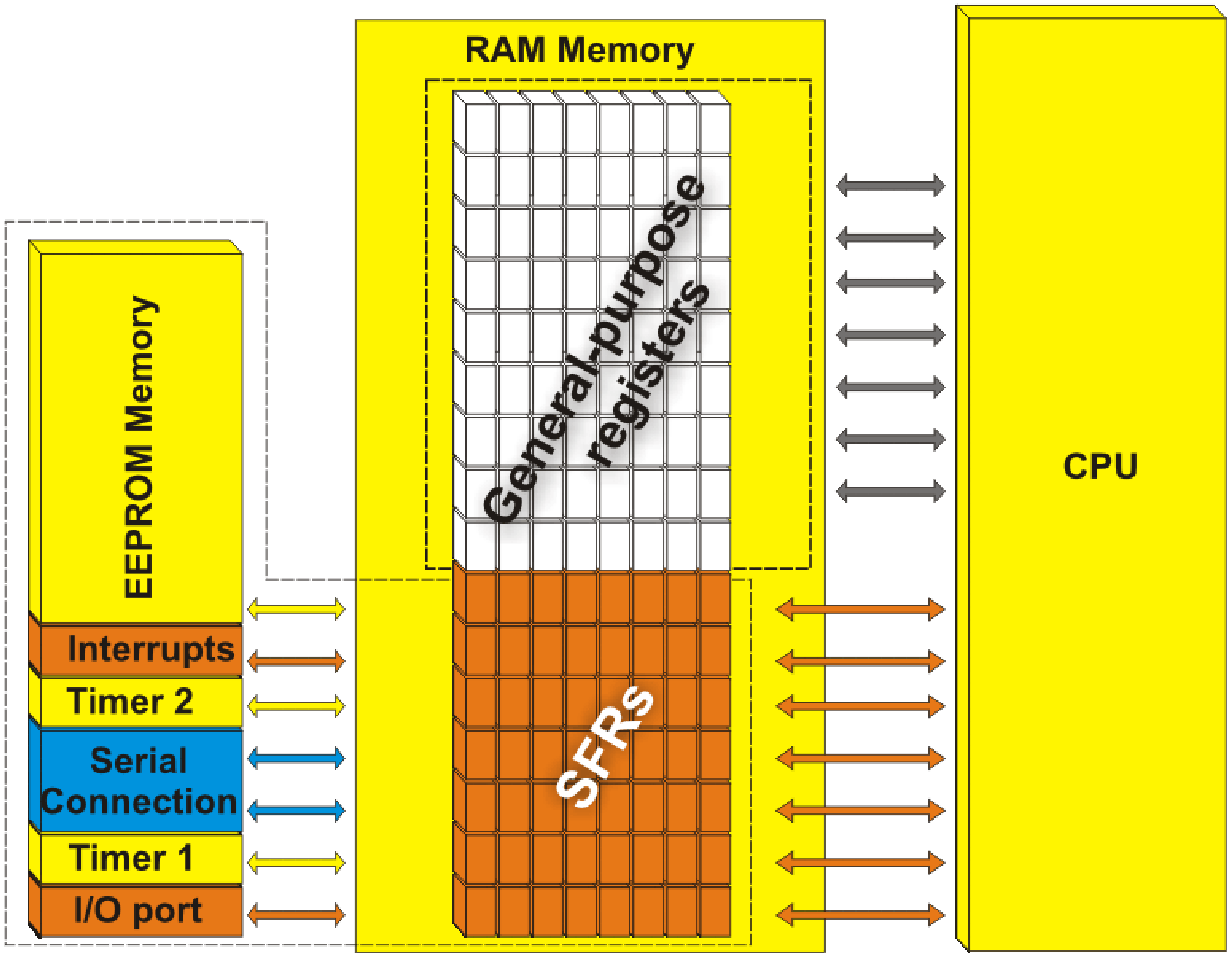
- IROM
- NAND flash
- NOR flash
- IRAM/SRAM(Steppingstone)
- SDRAM
- DDRAM

- Boards

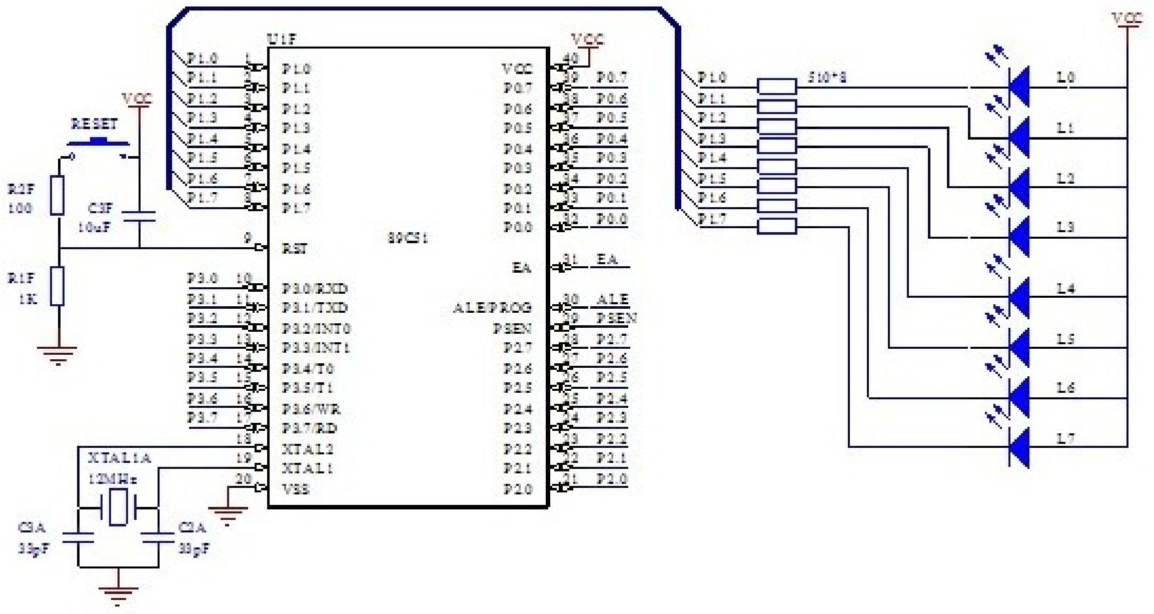
- st89c52


SFRs
Program Memory
Instructions
Addresses
WDT
OSC.
OSC/N
Timer
Timer
Timer
RAM Memory
Control
Decoder
Instruction
ALU
Accumulator
Logic
101
A/D
Program Counter
Input/Output ports
EEPROM

- stc89c52

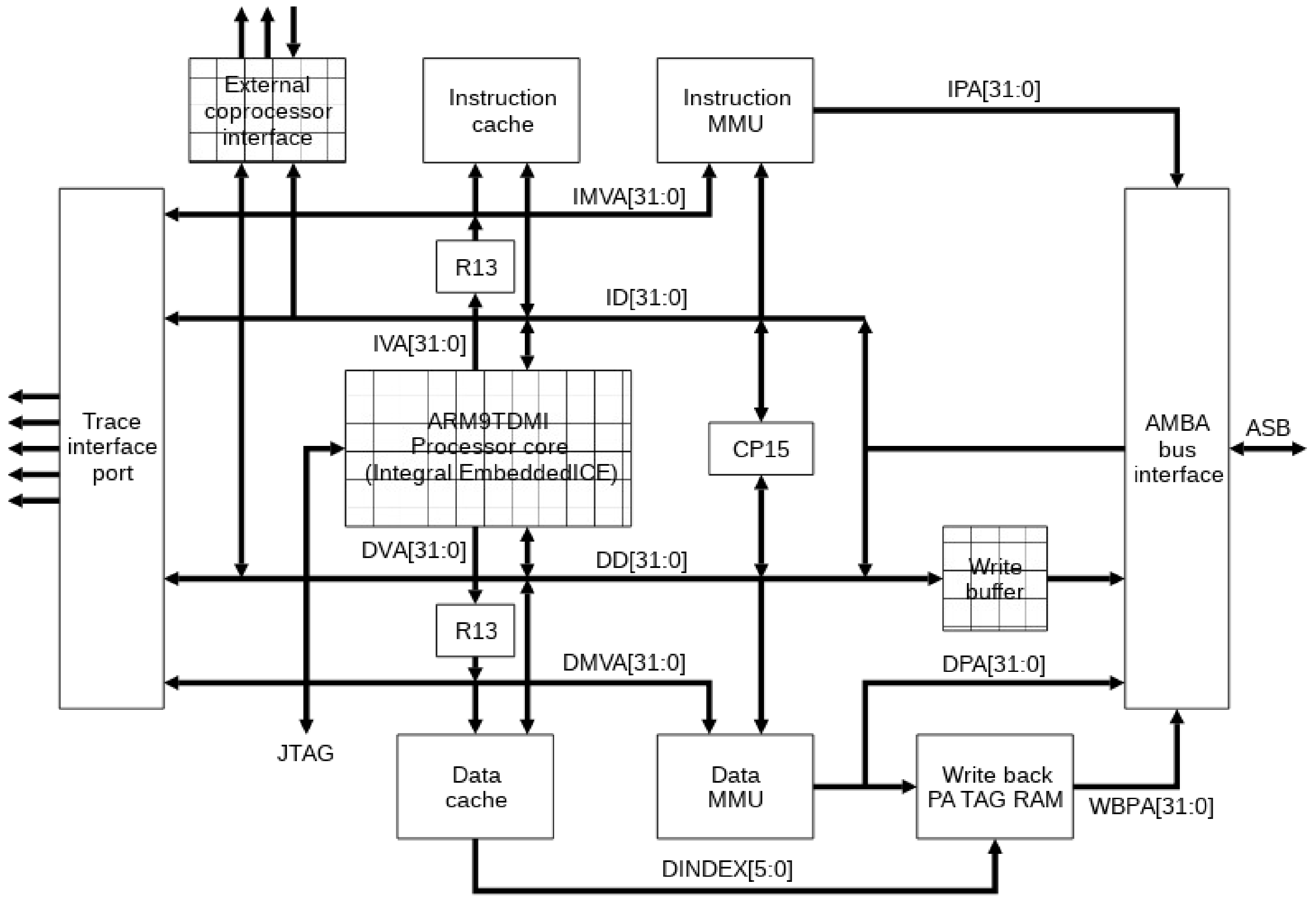

RAM Memory
General-purpose registers
SFRs
CPU
EEPROM Memory
Interrupts
Timer 2
Serial Connection
Timer 1
I/O port

- st89c52



U1F
89C51
VCC
RESET
R2F 100
C3F 10uF
R1F 1K
XTAL1A 12MHz
C3A 33pF
C2A 33pF
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
RST
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
XTAL2
XTAL1
VSS
P0.7
P0.6
P0.5
P0.4
P0.3
P0.2
P0.1
P0.0
EA
ALE/PROG
PSEN
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
510*8
L0
L1
L2
L3
L4
L5
L6
L7

Micro 2440


External coprocessor interface
Instruction cache
Instruction MMU
IPA[31:0]
IMVA[31:0]
R13
ID[31:0]
IVA[31:0]
Trace interface port
ARM9TDMI Processor core (Integral EmbeddedICE)
CP15
AMBA bus interface
ASB
DVA[31:0]
DD[31:0]
Write buffer
R13
DMVA[31:0]
DPA[31:0]
JTAG
Data cache
Data MMU
Write back PA TAG RAM
WBPA[31:0]
DINDEX[5:0]

Micro 2440

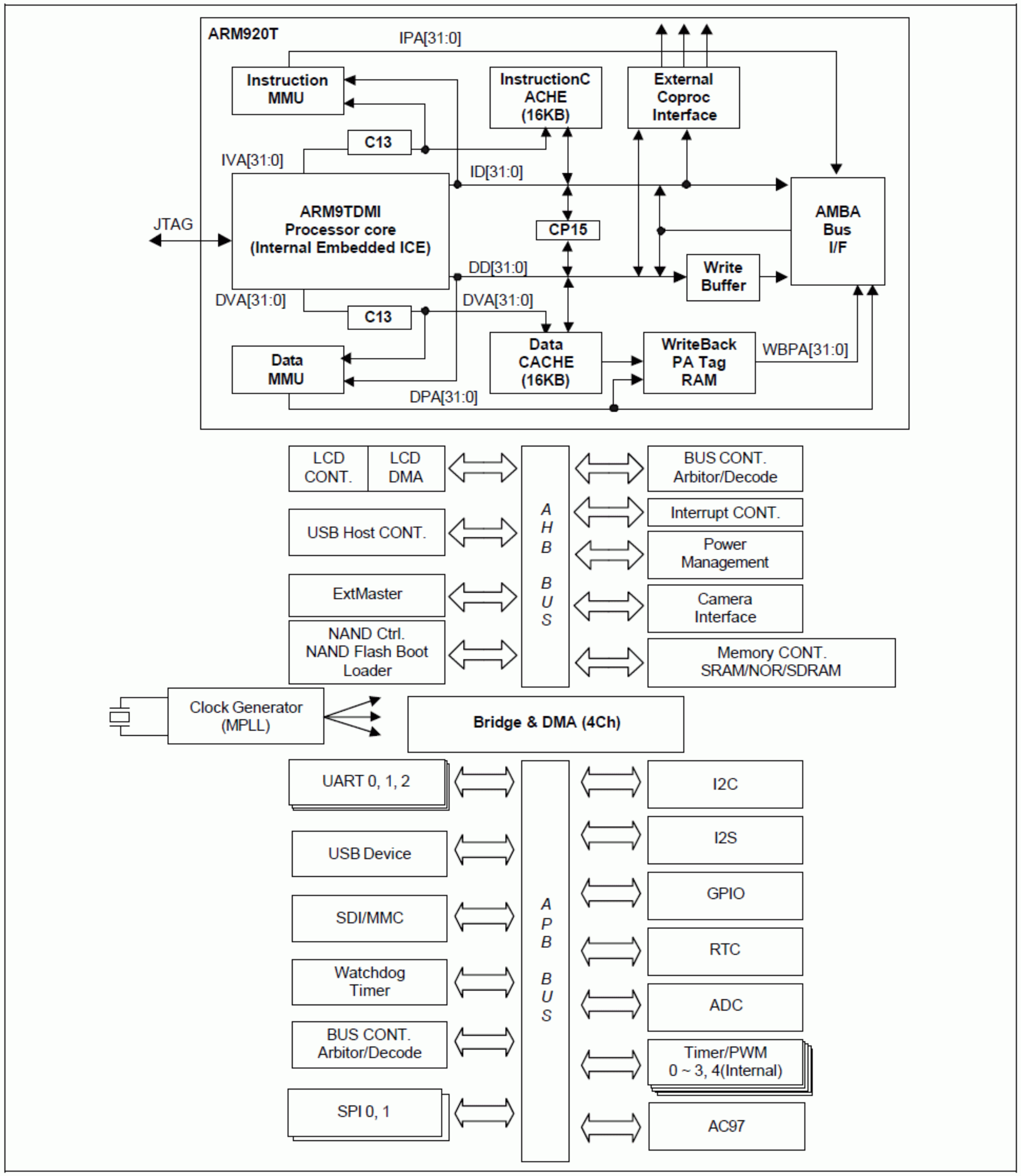
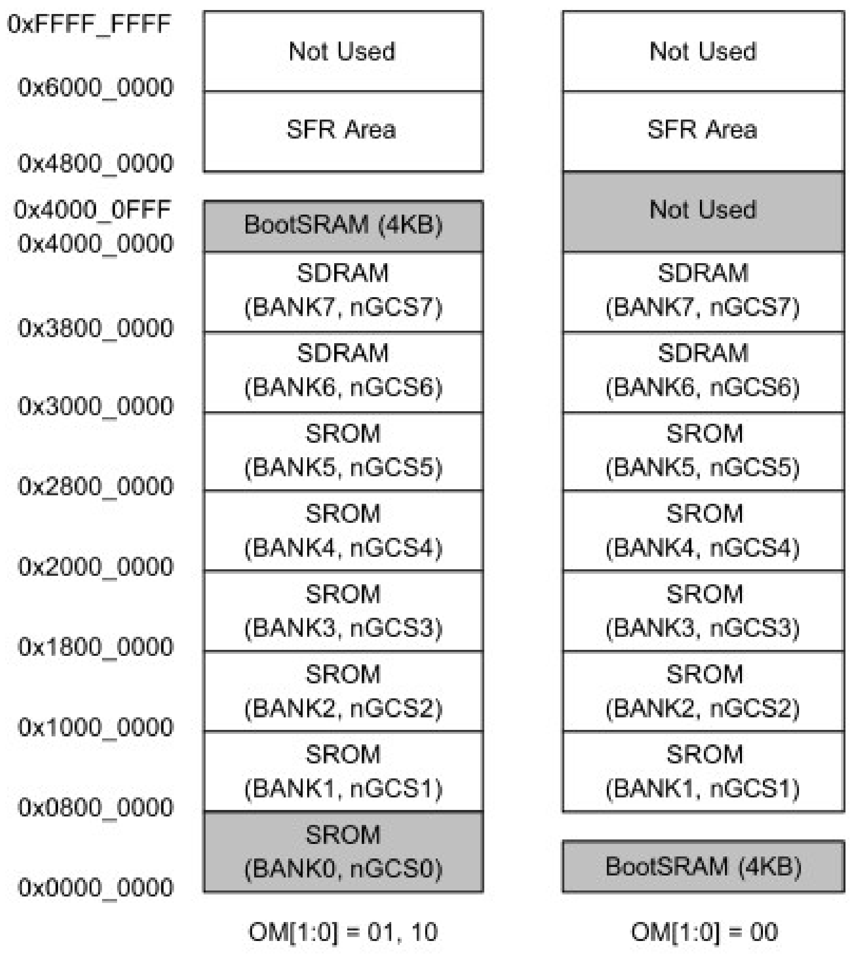


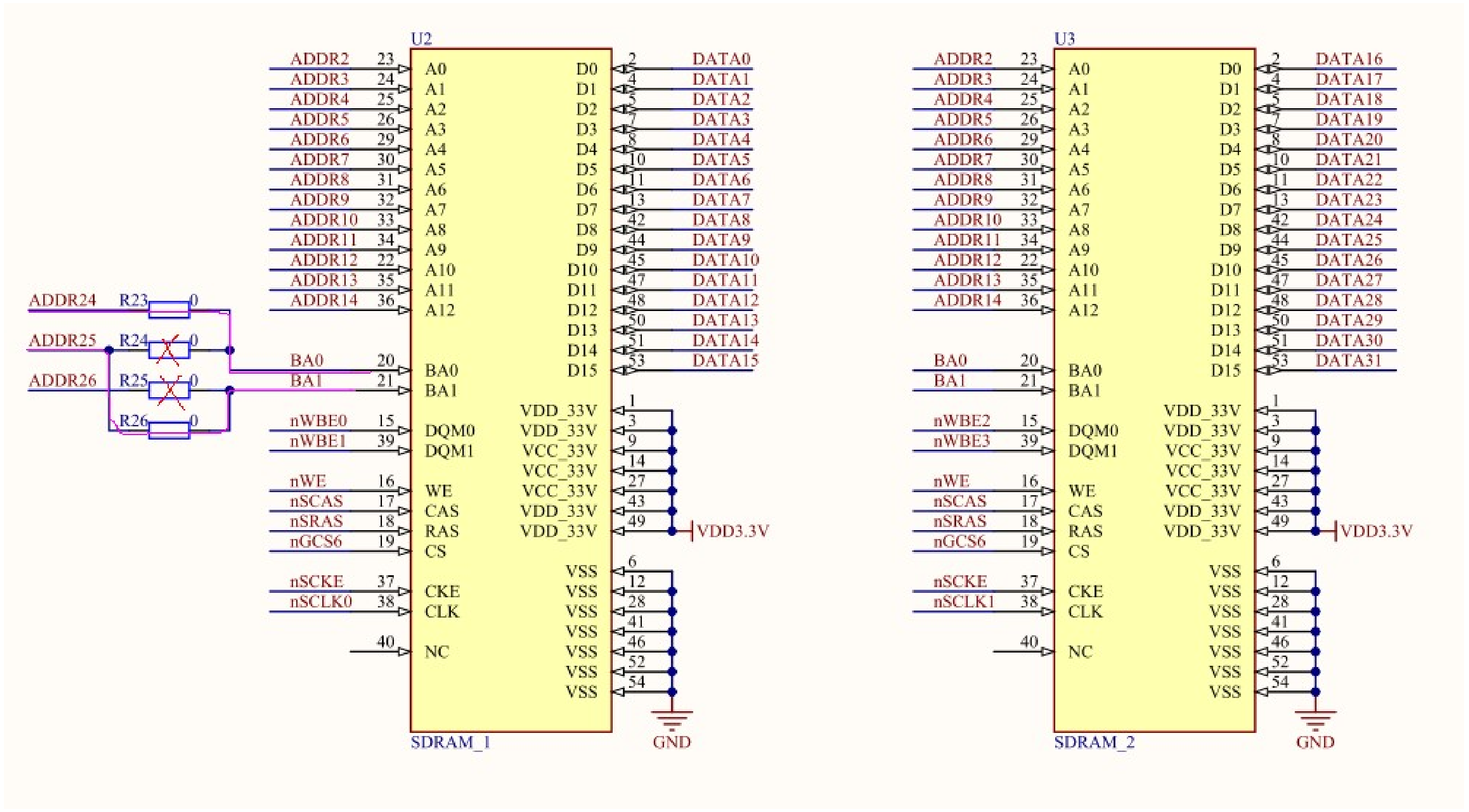
Figure 1-1. S3C2440A Block Diagram

32bit cpu could access 4G, however it only allows user accessed below 0x40000000, upper address are reserved for cpu registers and unused.

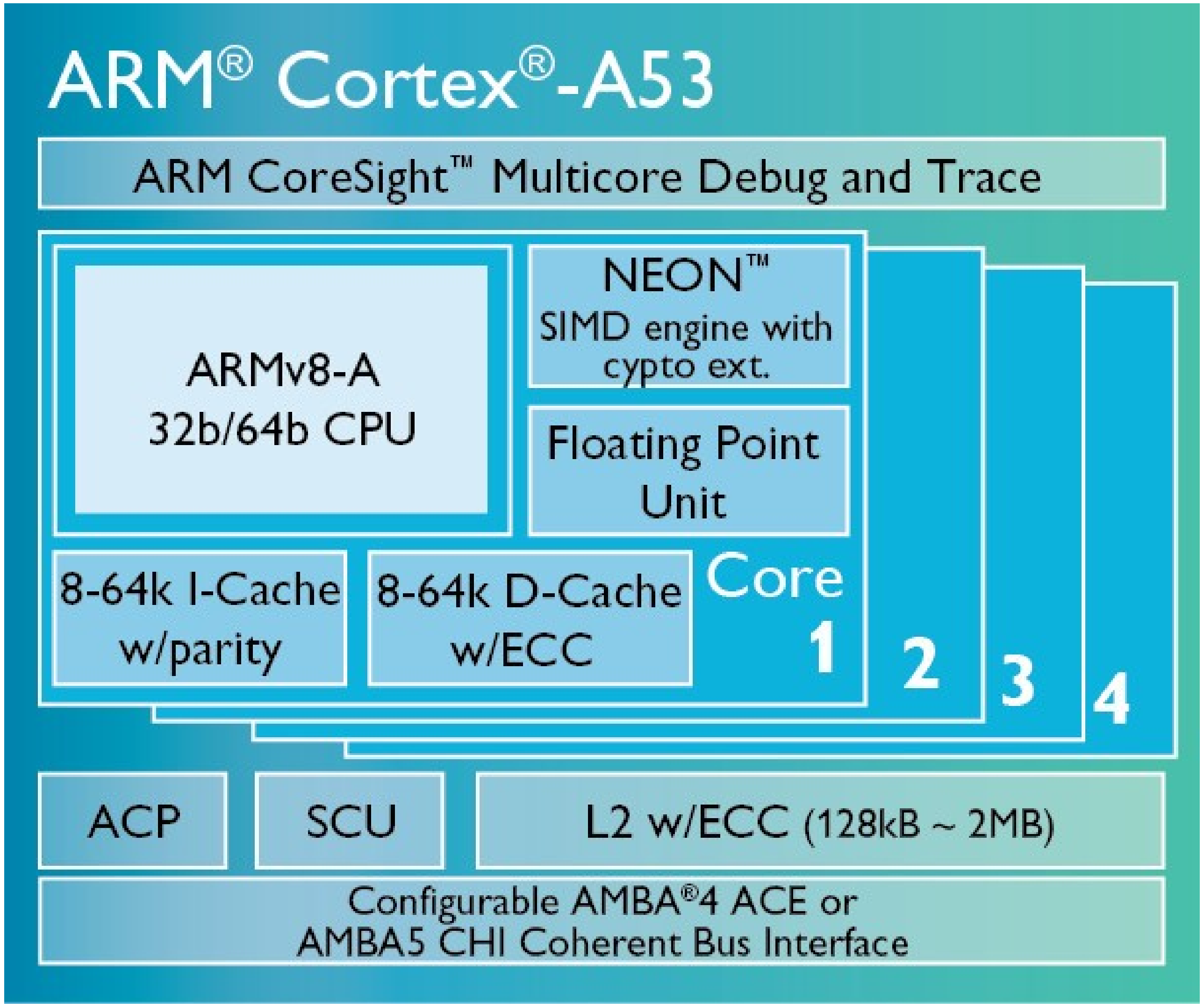
27 address line:

128 M



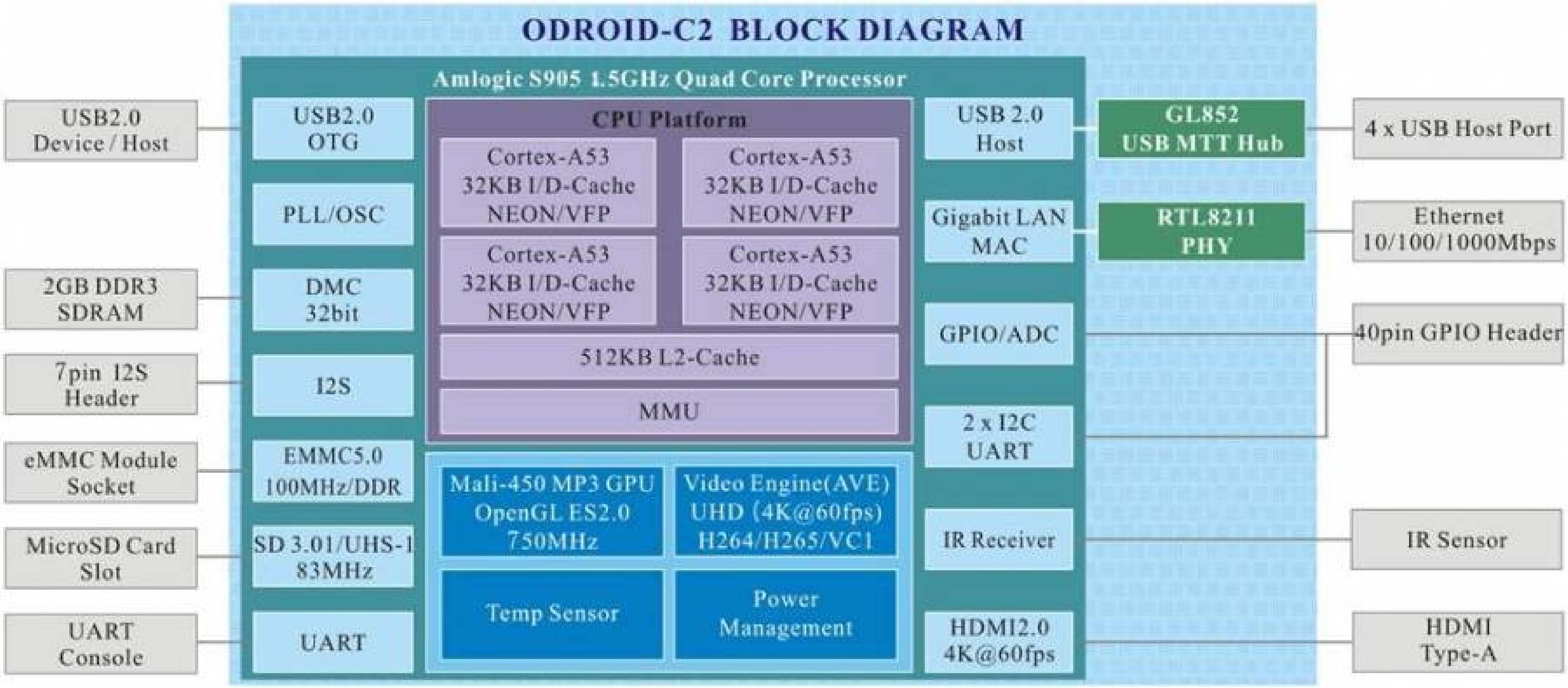

U2
SDRAM_1
U3
SDRAM_2
ADDR2 … ADDR14
A0 … A12
D0 … D15
DATA0 … DATA15
DATA16 … DATA31
ADDR24 R23 0
ADDR25 R24 0
ADDR26 R25 0
R26 0
BA0
BA1
nWBE0
nWBE1
nWBE2
nWBE3
DQM0
DQM1
nWE WE
nSCAS CAS
nSRAS RAS
nGCS6 CS
nSCKE CKE
nSCLK0 CLK
nSCLK1
NC
VDD_33V
VCC_33V
VSS
VDD3.3V
GND

ARM Cortex-A53

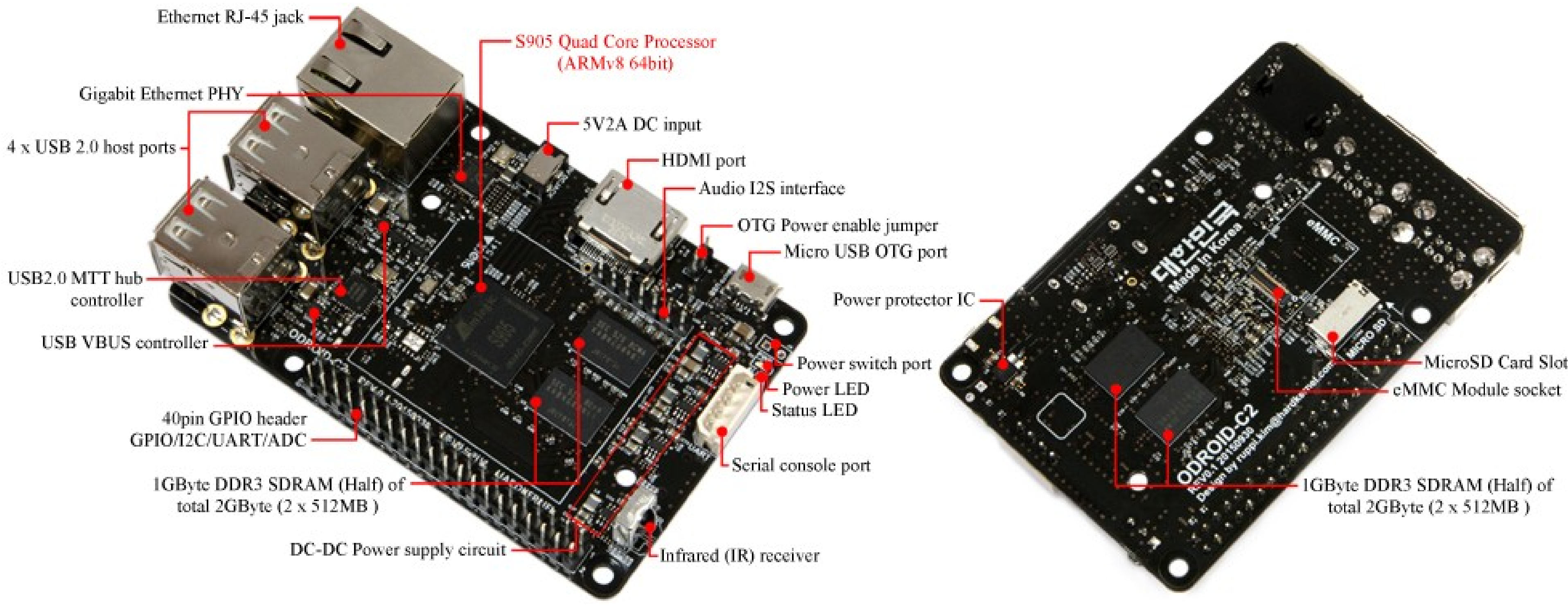

ARM® Cortex®-A53
ARM CoreSight™ Multicore Debug and Trace
ARMv8-A 32b/64b CPU
NEON™ SIMD engine with cypto ext.
Floating Point Unit
8-64k I-Cache w/parity
8-64k D-Cache w/ECC
Core 1
2
3
4
ACP
SCU
L2 w/ECC (128kB ~ 2MB)
Configurable AMBA®4 ACE or AMBA5 CHI Coherent Bus Interface

- Raspberry Pi

Odroid c2



- Odroid c2


Ethernet RJ-45 jack
S905 Quad Core Processor (ARMv8 64bit)
Gigabit Ethernet PHY
5V2A DC input
4 x USB 2.0 host ports
HDMI port
Audio I2S interface
OTG Power enable jumper
Micro USB OTG port
USB2.0 MTT hub controller
Power protector IC
USB VBUS controller
Power switch port
Power LED
Status LED
MicroSD Card Slot
eMMC Module socket
40pin GPIO header GPIO/I2C/UART/ADC
Serial console port
1GByte DDR3 SDRAM (Half) of total 2GByte (2 x 512MB)
1GByte DDR3 SDRAM (Half) of total 2GByte (2 x 512MB)
DC-DC Power supply circuit
Infrared (IR) receiver

Software:

Uboot
Kernel

Once a board is ready, everything is fixed

FSM

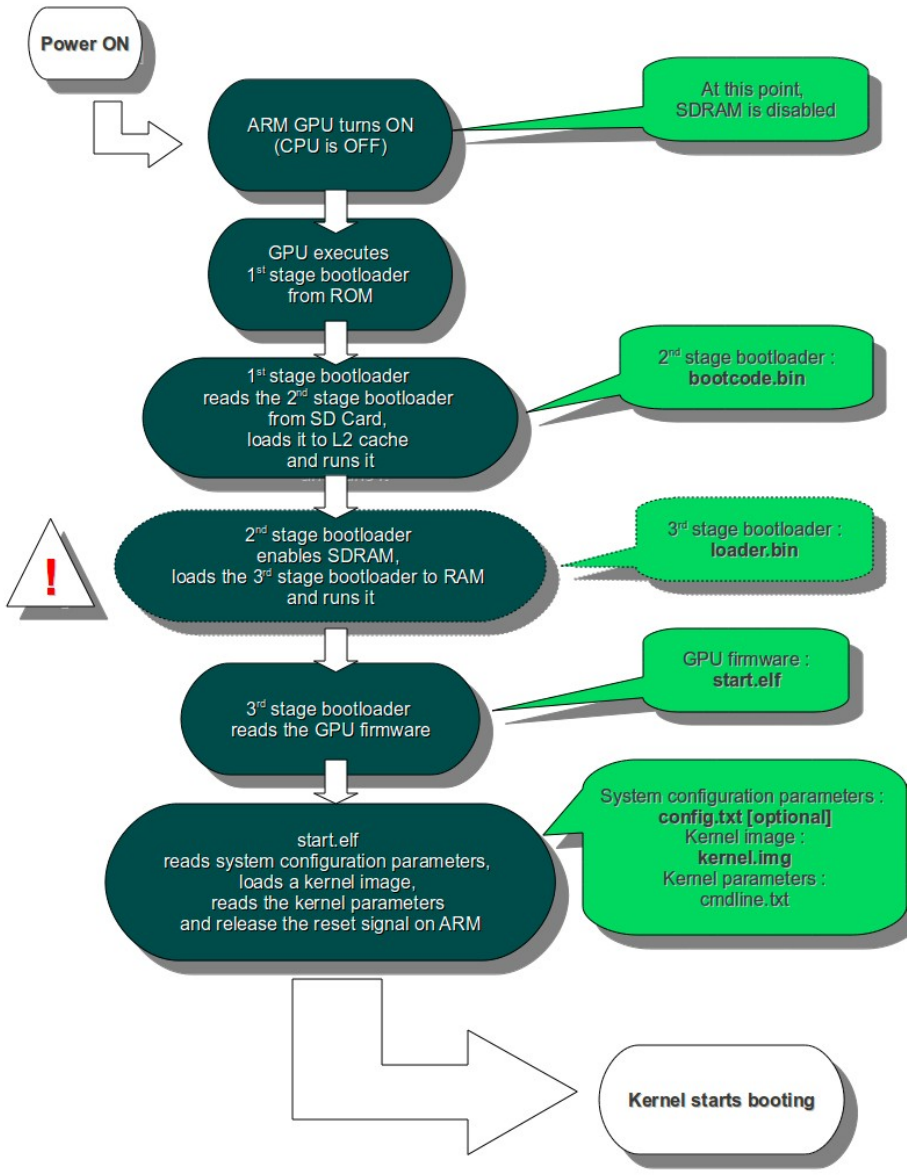
Finite state machine

Our code is based on a fixed infrastructure

What are variables?
0 and 1， once flashed or written in storage.

After that:

Input starts everything, otherwise it will in a idle loop!!!


Power ON
ARM GPU turns ON (CPU is OFF)
At this point, SDRAM is disabled
GPU executes 1st stage bootloader from ROM
1st stage bootloader reads the 2nd stage bootloader from SD Card, loads it to L2 cache and runs it
2nd stage bootloader : bootcode.bin
2nd stage bootloader enables SDRAM, loads the 3rd stage bootloader to RAM and runs it
!
3rd stage bootloader : loader.bin
3rd stage bootloader reads the GPU firmware
GPU firmware : start.elf
start.elf reads system configuration parameters, loads a kernel image, reads the kernel parameters and release the reset signal on ARM
System configuration parameters : config.txt [optional] Kernel image : kernel.img Kernel parameters : cmdline.txt
Kernel starts booting

Stage 1 boot is in the on-chip ROM. Loads Stage 2 in the L2 cache

Stage 2 is boootcode.bin. Enables SDRAM and loads Stage 3

Stage 3 is loader.bin. It knows about the .elf format and loads start.elf

start.elf loads kernel.img. It then also reads config.txt, cmdline.txt and bcm2837.dtb If the dtb file exists, it is loaded at 0×100 & kernel @ 0×8000 If disable_commandline_tags is set it loads kernel @ 0×0 Otherwise it loads kernel @ 0×8000 and put ATAGS at 0×100

kernel.img is then run on the ARM.

Everything is run on the GPU until kernel.img is loaded on the ARM.

SOC boards

- stc89c52rc:

no need for OS or any other boot, ISP/IAP

- ARM V7
  Micro 2440 board

Start support, uboot, vivi, supervivi

- ARM V8(AARCH64)
  Raspberry pi board

Uboot/uefi

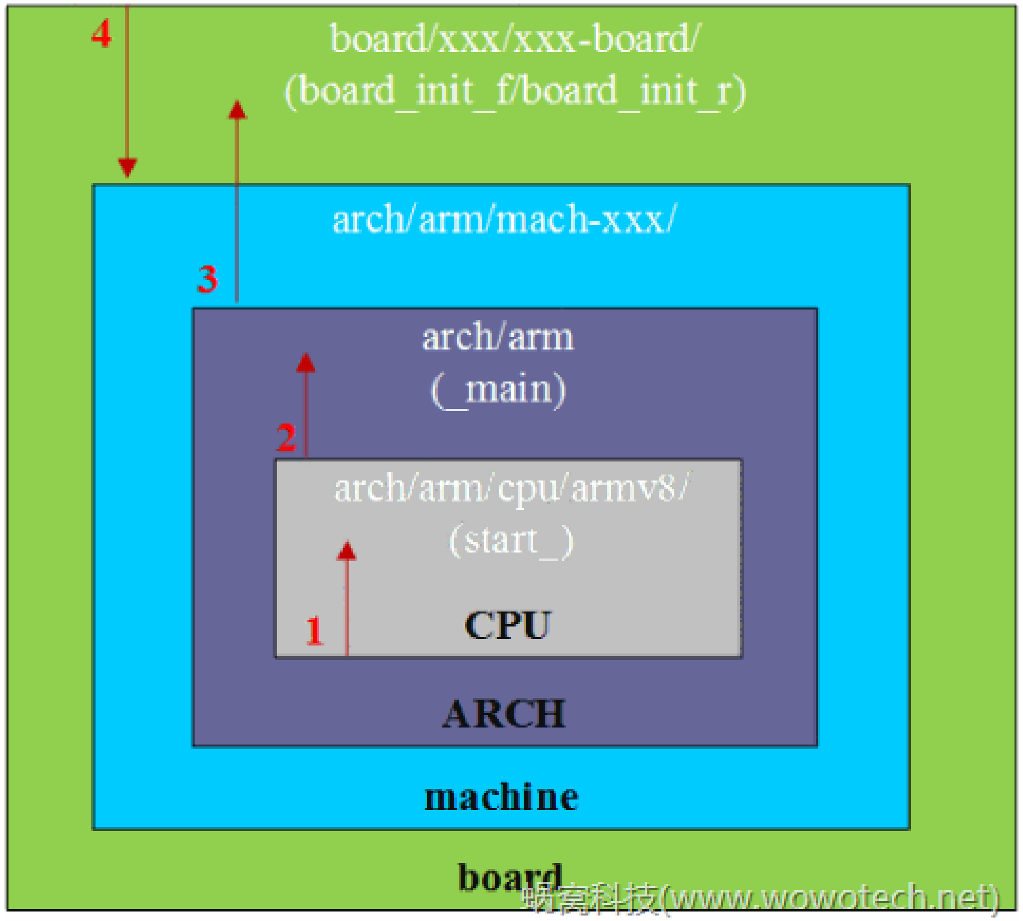
board—>machine—>arch—>cpu

Board: raspberry pi 3b

Machine: bcm2837

Arch: arm64/arm32

CPU: armv8



Stage 1
Start address: 0x00

Arch related:

start.s _start: b start_code

setup interrupt vectors
reset and set CPU to SVC
disable cache, MMU, TLBs()

Board related:

lowlevel_init.S ldr pc, _start_armboot

setup watchdog, muxing, and clocks
setup SP, pll, mux, memory(SRAM,SROM)
board initialization(uart, nand, lcd, led, nic)
clear bss
copy to ram and start from there

Stage 1
Start address: 0x00

main_loop prepare to get into kernel

1   CPU register

r0=0 r1=unique architecture number
r2= ram address for kernel parameters

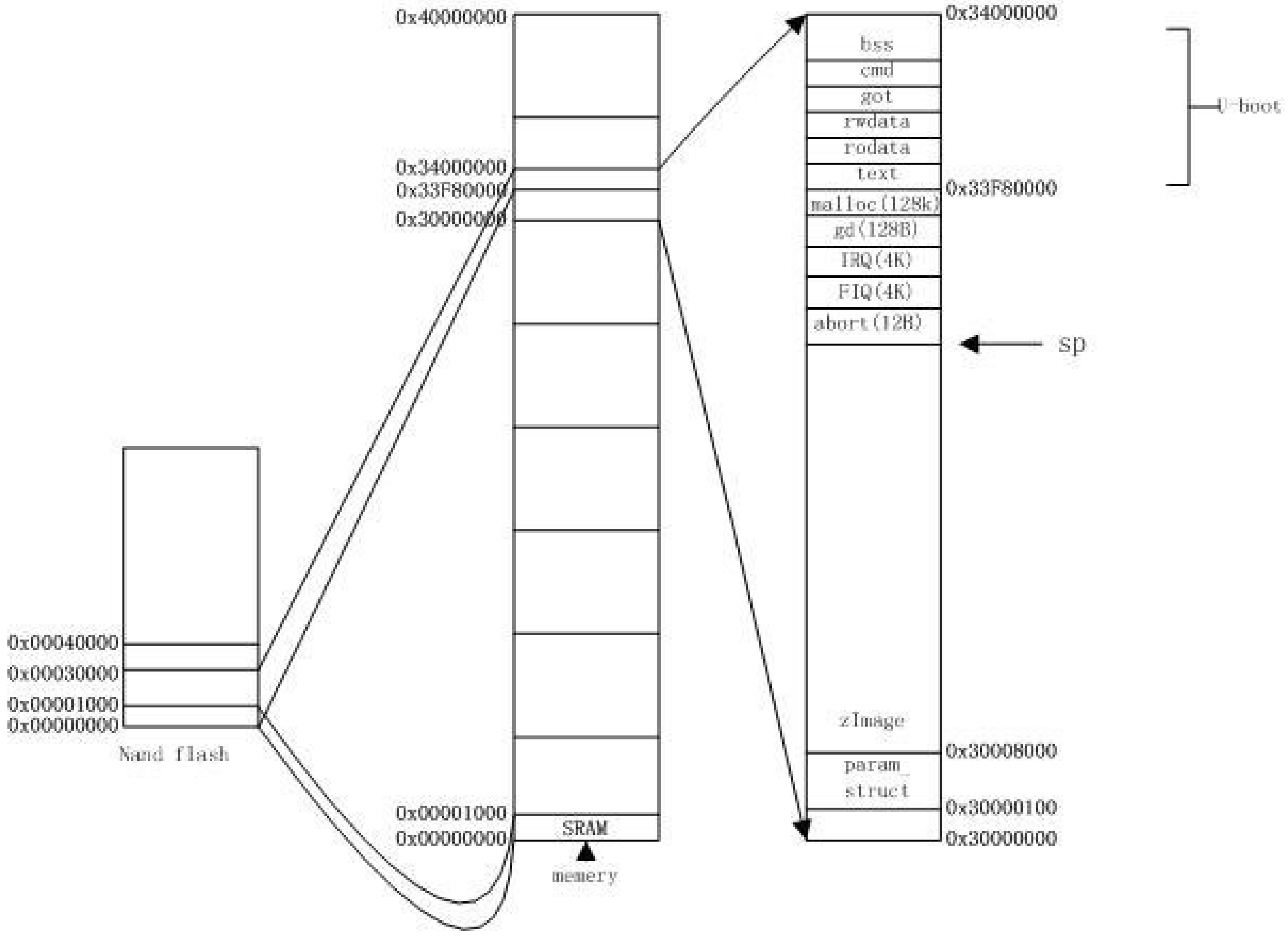
2   CPU mode
Disable IRQ and FIQ CPU SVC mode

3   disable D-Cache and I-Cache

```
1.  arch
_start————>reset—————>disable interrupt
………………………………|
……………………………….———→cpu_init_cp15————>disble MMU,TLB
………………………………|
……………………………….———→cpu_init_crit————→lowlevel_init————→ config and initialize key registers
………………………………|
……………………………….———→_main—————> jump to board

2.  board

_main—————>board_init_f_alloc_reserve —————>stack 、 GD 、 early malloc
…………|
…………—————->board_init_f_init_reserve —————>stack 、 GD 、 early malloc
…………|
…………—————->board_init_f —————>board initialization before uboot relocate, arrange relocate area
…………|
…………—————->relocate_code 、 relocate_vectors —————>relocate uboot vectors
…………|
…………—————->  clean old stack
…………|
…………—————->board_init_r —————>board initialization after uboot relocate
…………|
…………—————->run_main_loop —————> command status, waiting for input
--------------------
```



0x40000000
0x34000000
0x33F80000
0x30000000
0x00001000
0x00000000
SRAM
memery
0x00040000
0x00030000
0x00001000
0x00000000
Nand flash
bss
cmd
got
rwdata
rodata
text
malloc(128k)
gd(128B)
IRQ(4K)
FIQ(4K)
abort(12B)
zImage
param_struct
0x34000000
0x33F80000
0x30008000
0x30000100
0x30000000
U-boot
sp

Kernel:

reset and set CPU to SVC
disable interrupt

Proc ID verification
Parameters(atags/dtb) verification

Get Page table physical address and zero

remap _turn_mmu_on fuction(1:1)

remap kernel code segment

Parameters map

Initialize tlb and cache,
Save pagetable to tlb

Enable mmu

Relocate data segment

Clean BSS
Start Kernel

```
/* sorce code */    head.S
/* entry point */
ENTRY(stext)
/* program status， disable FIQ、IRQ， enable SVC mode*/
mov r0, #F_BIT | I_BIT | MODE_SVC@ make sure svc mode
/* setup current registers */
msr cpsr_c, r0 @ and all irqs disabled
/* verify CPU mode， compare current CPU Id with Linux compiled ID */
bl __lookup_processor_type
/* Jump __error */
teq r10, #0 @ invalid processor?
moveq r0, #'p' @ yes, error 'p'
beq __error
/* check Architecture Type from R1 */
bl __lookup_architecture_type
/* jump to error if invalid */
teq r7, #0 @ invalid architecture?
moveq r0, #'a' @ yes, error 'a'
beq __error
/* create page table */
bl __create_page_tables
adr lr, __ret @ return address
add pc, r10, #12 @ initialise processor
/* jump to start_kernel */
b start_kernel
```

Start Kernel:

Once kernel code is in DRAM:

Key Registers are initialized

Stack environment is setup,

Create temperate page table

Related hardware is initialized
(MMU,TLB, Cache)

===>

Create real page table (bootm, page_init, buddy, slab)
set_task_stack_end_magic(&init_task); ==> pid0, task_struct is created manually
setup_arch
trap_init
mem_init
sched_init
init_irq
rest_init(); ==> pid = kernel_thread
==> kernel_init(pid 1) ==> all user process
==> kthreadd(pid 2) ==> all kernel threads ==> cpu_idle_loop(pid0)

Further imagination:

The whole process:

- bootrom/bios
- uboot/SPL
- FDT
- Kernel
- Initrd?
- rootfs:

Init, systemd/systemV

How is your phone booted? Your router? Tablet? Laptop?

Userspace development:

- Based on different kinds of input.
- Think about a function(Algorithm), start from input and end by output

X86? Other architecture?

Question?

Thank you.

We adapt. You succeed.

REFERENCE

https://raspberrypi.stackexchange.com/questions/10442/what-is-the-boot-sequence
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/bootflow.md
http://www.friendlyarm.net/products/micro2440
https://github.com/wowotechX/u-boot
https://www.kernel.org/doc/Documentation/arm/Booting
https://blog.csdn.net/cagent_z/article/details/61441951
https://blog.csdn.net/ooonebook/article/details/52850433
http://www.friendlyarm.net/
https://developer.arm.com/products/processors/cortex-a/cortex-a53
http://wiki.100ask.org/images/c/c4/S3C2440A_UserManual_Rev13.pdf
https://dn.odroid.com/S905/DataSheet/S905_Public_Datasheet_V1.1.4.pdf
https://wiki.odroid.com/odroid-c2/odroid-c2

We adapt. You succeed.

Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org

257-000020-001

SUSE®
We adapt. You succeed.