国立大学法人
電気通信大学
The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

「Course」 RISC-V Computer System Integration

「Lecture 4」 Rocket Computer System: Boot Sequence and Software

Hoang Trong Thuc

2022/11

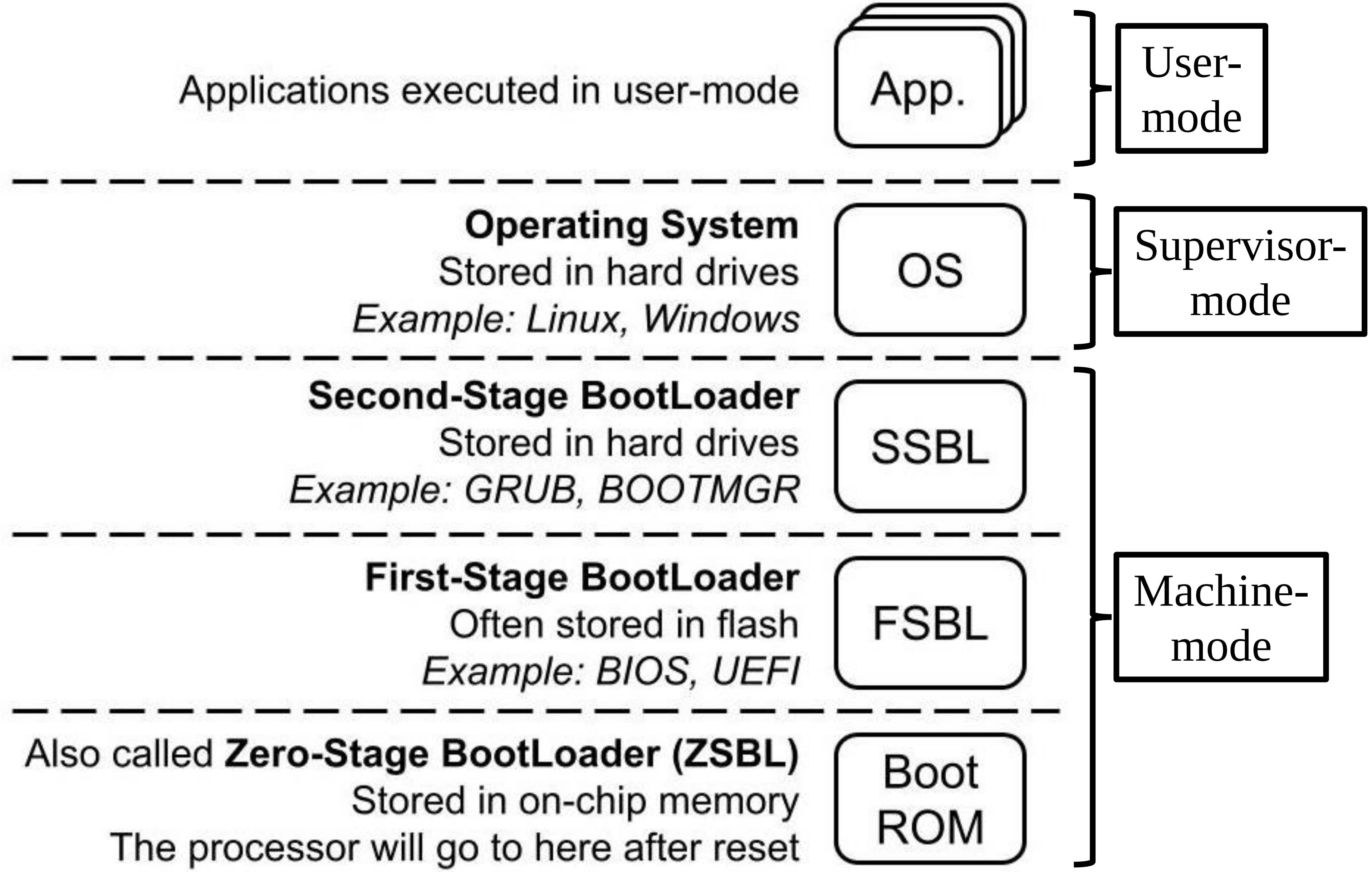
Outline

1. Generic boot sequence
2. Our system's boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

Outline

1. Generic boot sequence
2. Our system's boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

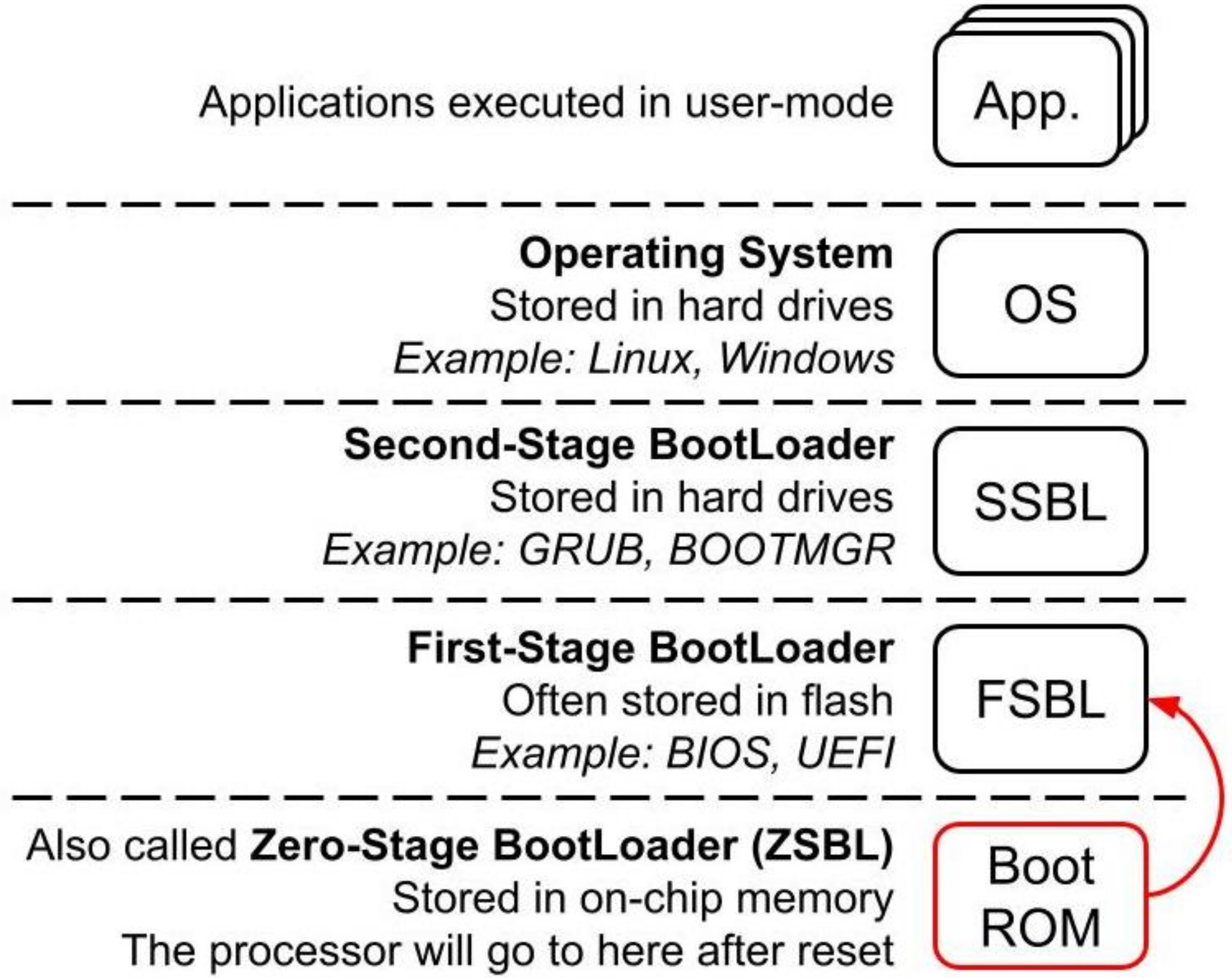
1. Generic boot sequence (1/7) Boot flow



A typical PC boot sequence will look like this.

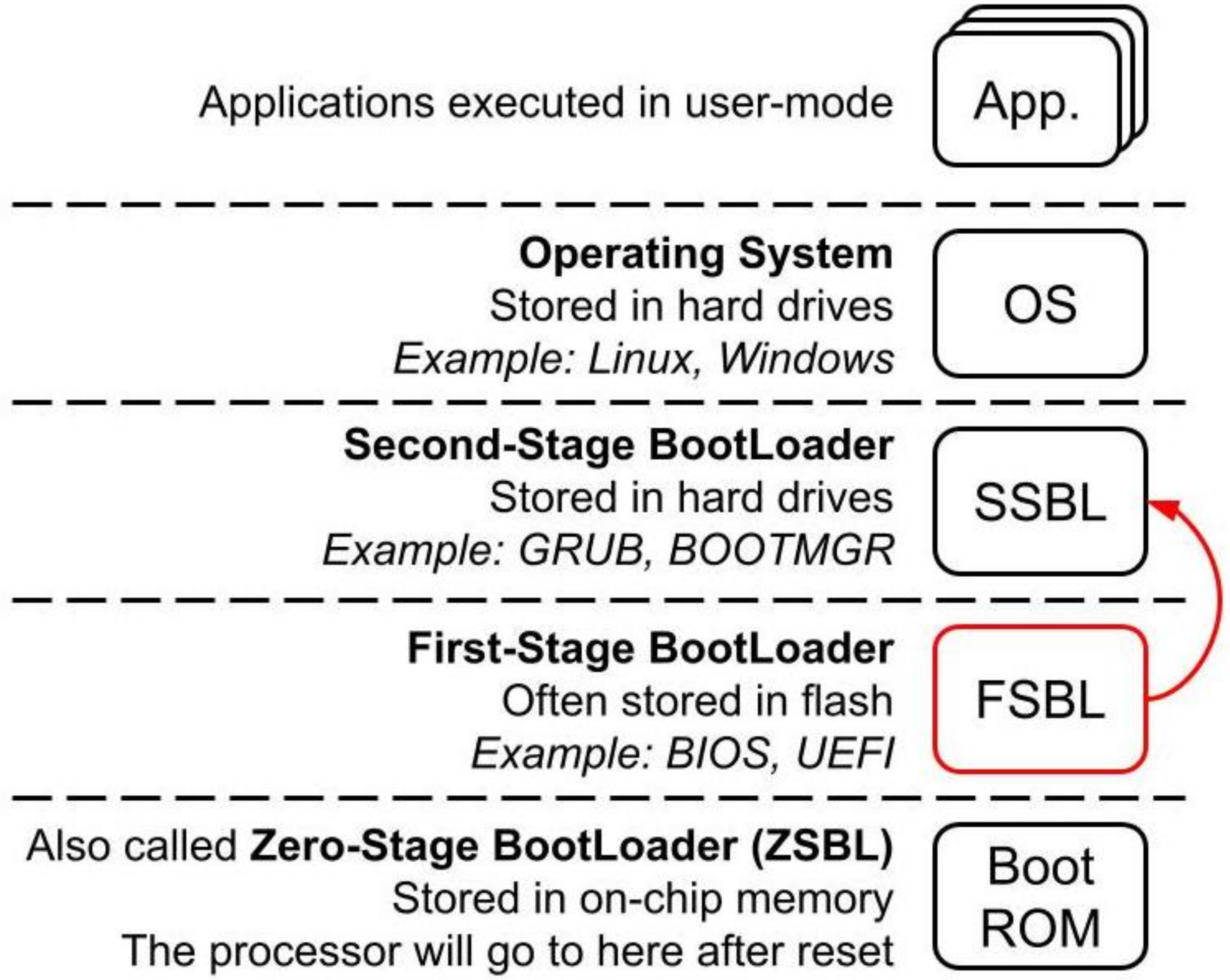
1. It will begin from the boot ROM,
2. then goes to several bootloaders,
3. and finally, boot to an Operating System (OS)

1. Generic boot sequence (2/7) Boot ROM



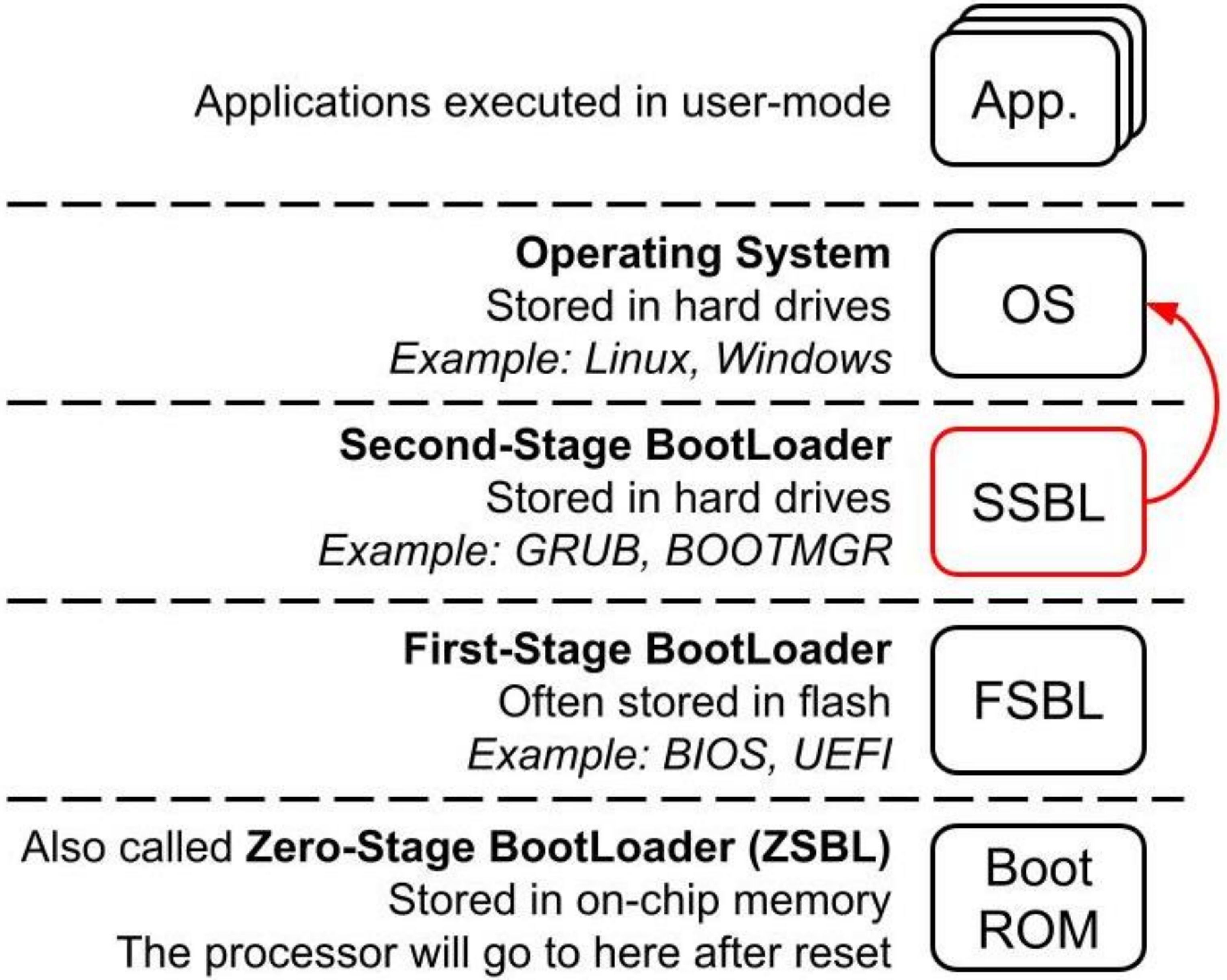
- Boot ROM *(also called ZSBL)* is usually very small and simple.
- Its main task is just carrying the device tree file.
- At reset, the processor core will go here.
- Normally, its primary function is to go to the FSBL for more complicated processing.

1. Generic boot sequence (3/7) FSBL



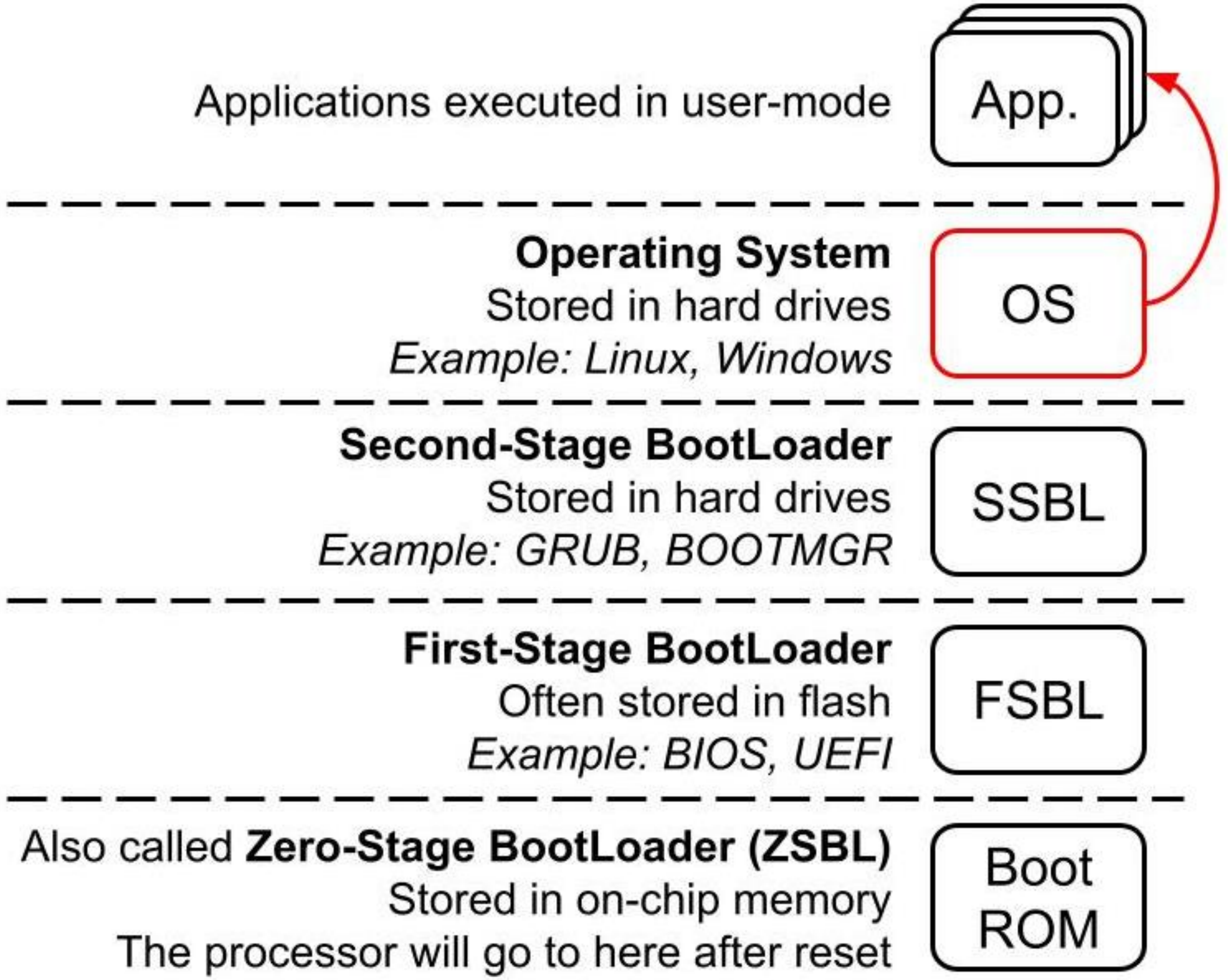
- FSBL's main task is preparing the necessary files and environment for later bootloaders.
- FSBL also checks hardware memories and devices based on the given device tree file.
  → and also configs some of the devices, such as UART and SPI.

1. Generic boot sequence (4/7) SSBL


Applications executed in user-mode
App.
Operating System
Stored in hard drives
Example: Linux, Windows
OS
Second-Stage BootLoader
Stored in hard drives
Example: GRUB, BOOTMGR
SSBL
First-Stage BootLoader
Often stored in flash
Example: BIOS, UEFI
FSBL
Also called **Zero-Stage BootLoader (ZSBL)**
Stored in on-chip memory
The processor will go to here after reset
Boot ROM

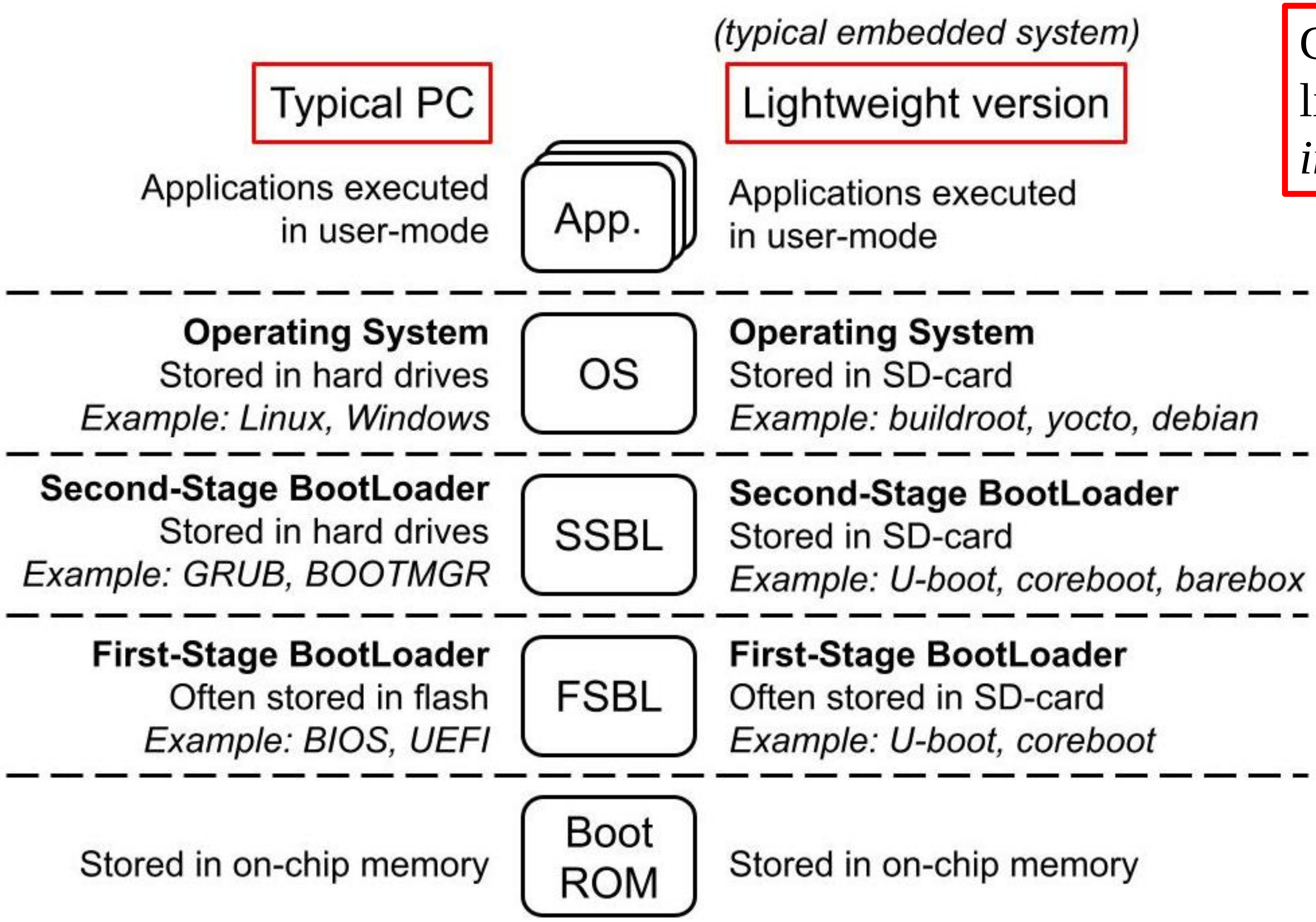
- SSBL's goal is to set up the runtime environment for a specific Operating System (OS).
- SSBL also allocates memory for the OS and copies system drivers and standard libraries.

1. Generic boot sequence (5/7) OS



- OS's main goal is to set up the environment for user interaction.
- System and application drivers are installed in this stage.

1. Generic boot sequence (6/7) Lightweight version

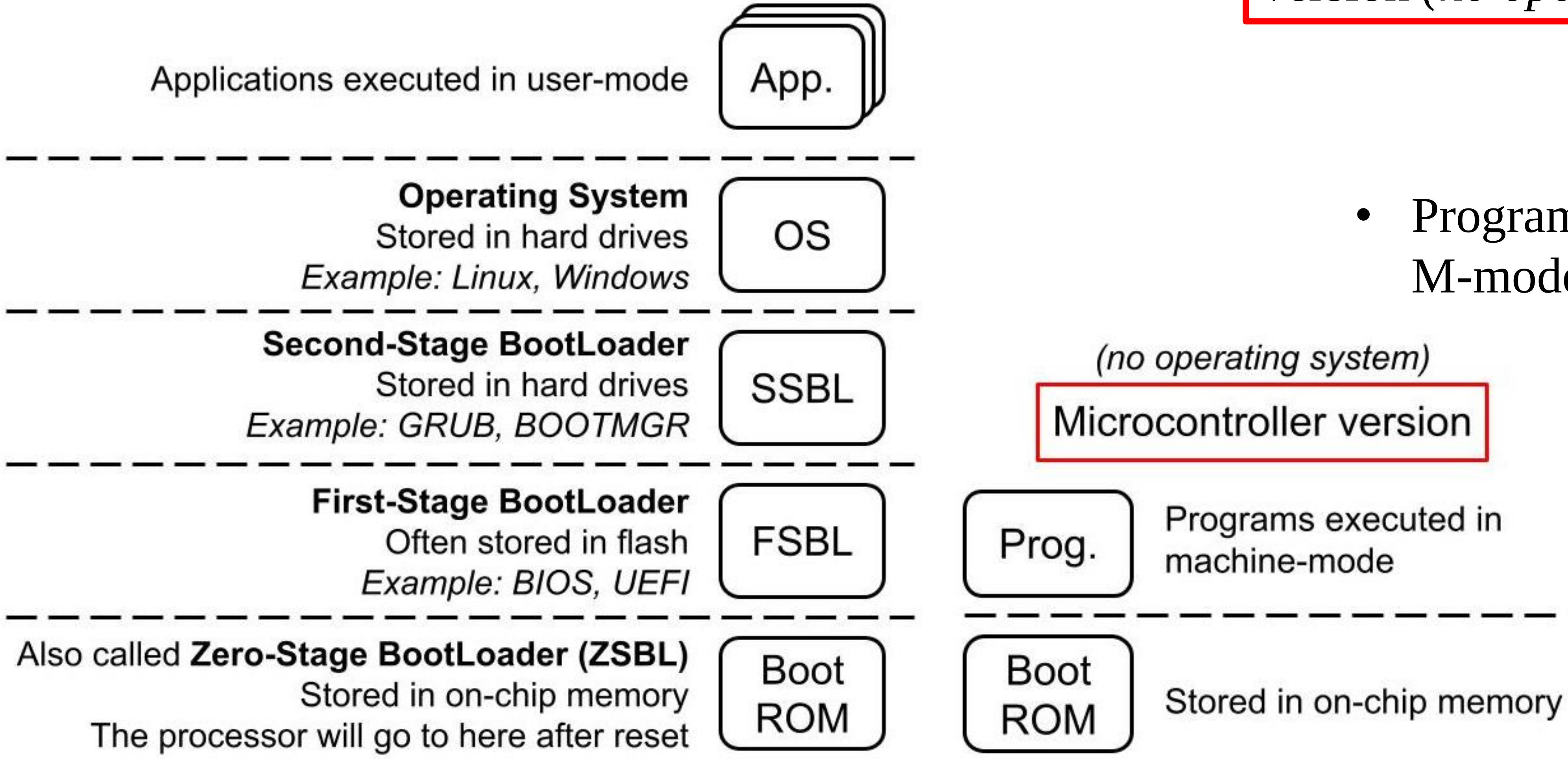


Comparing to the lightweight version *(usually in embedded systems)*

- Bootloaders and OS data are often stored in SD-card.
- Simpler bootloaders are used.
- Compact OSes are used *(usually an UNIX system)*.

1. Generic boot sequence (7/7) Microcontroller version

Comparing to the microcontroller version *(no operating system)*

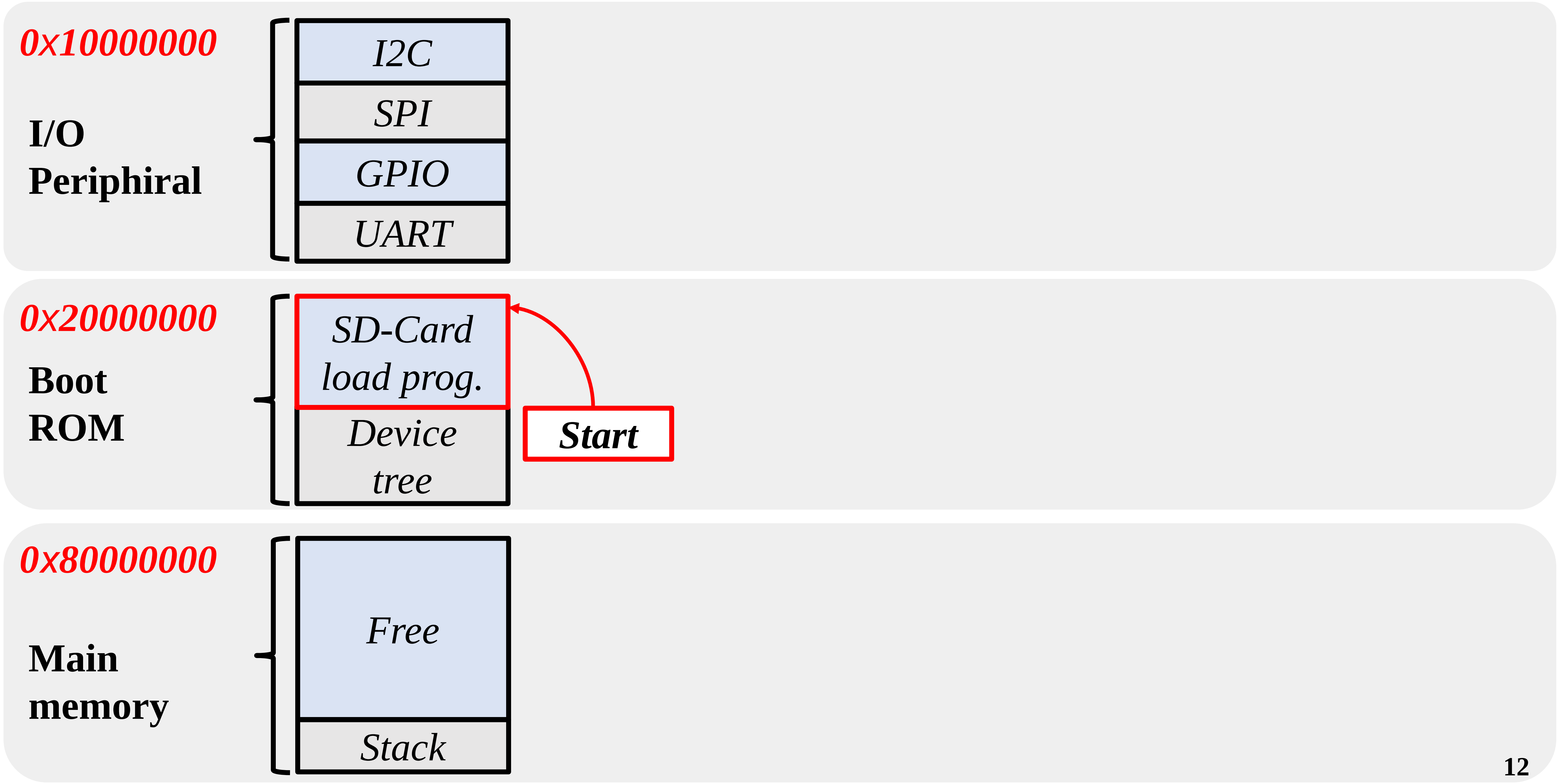


- Programs are executed at M-mode *(not U-mode)*
  - Because there is no OS, only one program can be executed at a time *(no multi-thread)*

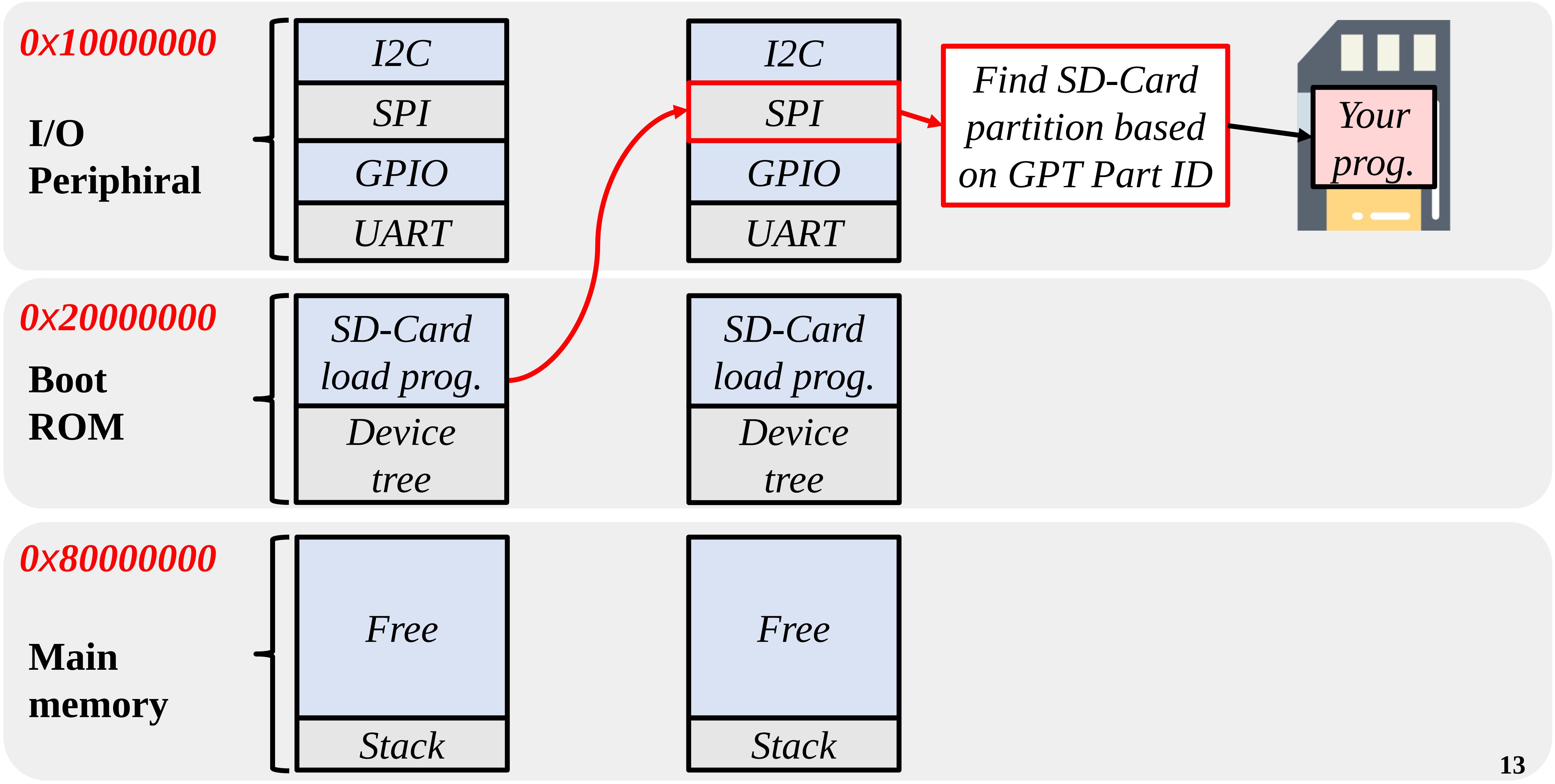
Outline

1. Generic boot sequence
2. Our system’s boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

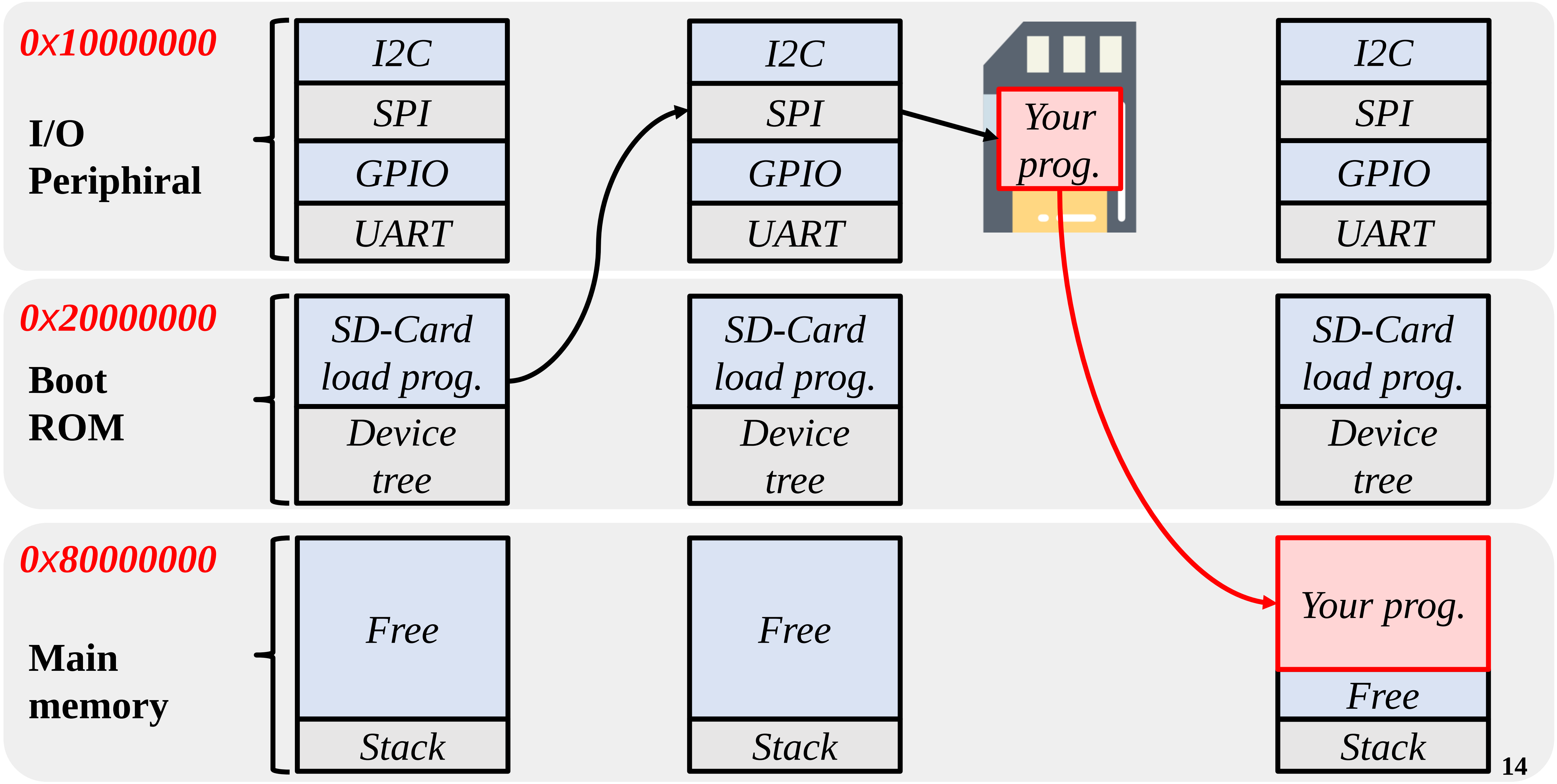
2. Our system's boot sequence (1/5) Start at boot ROM



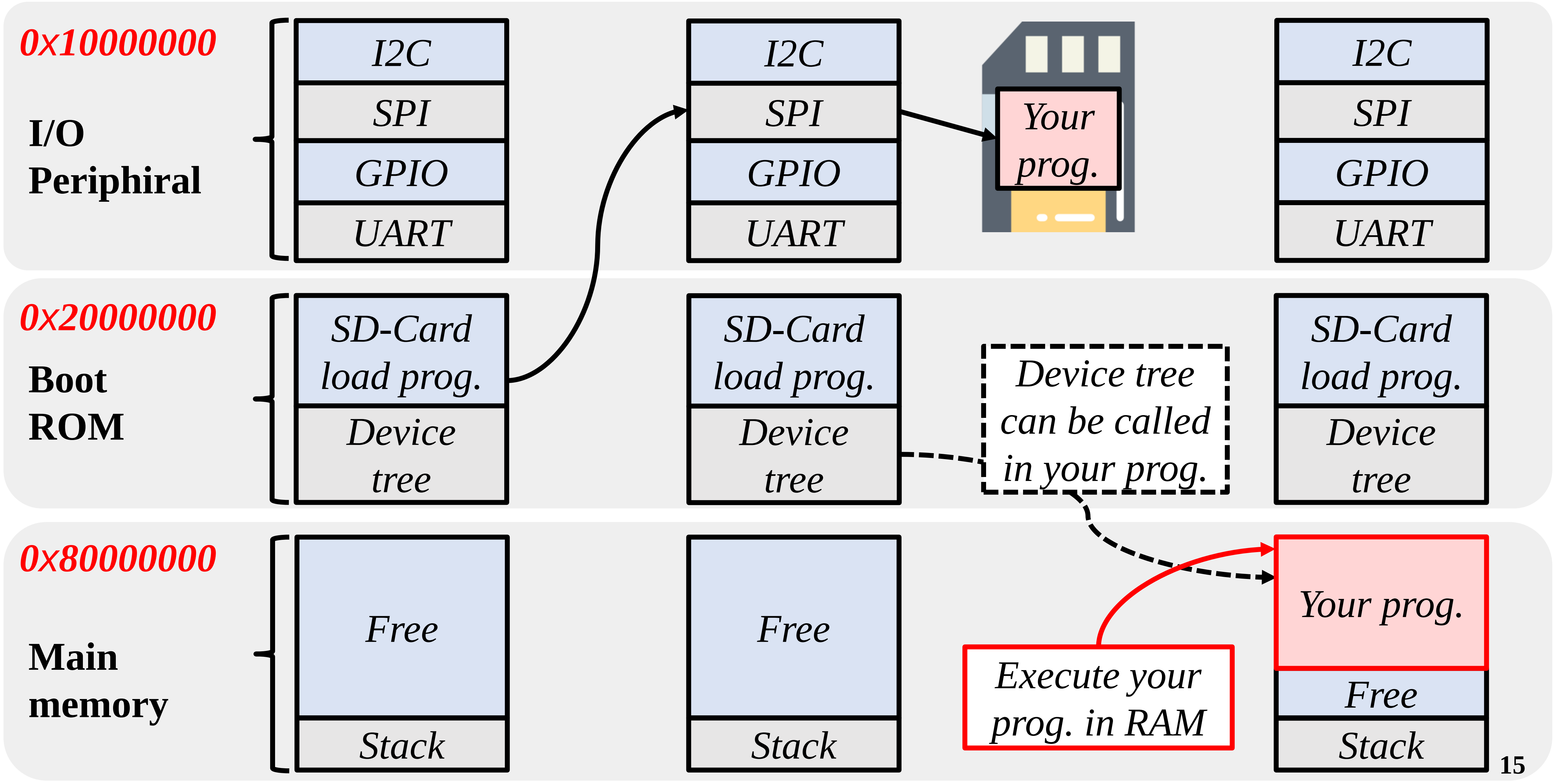
2. Our system's boot sequence (2/5) Find SD-card partition



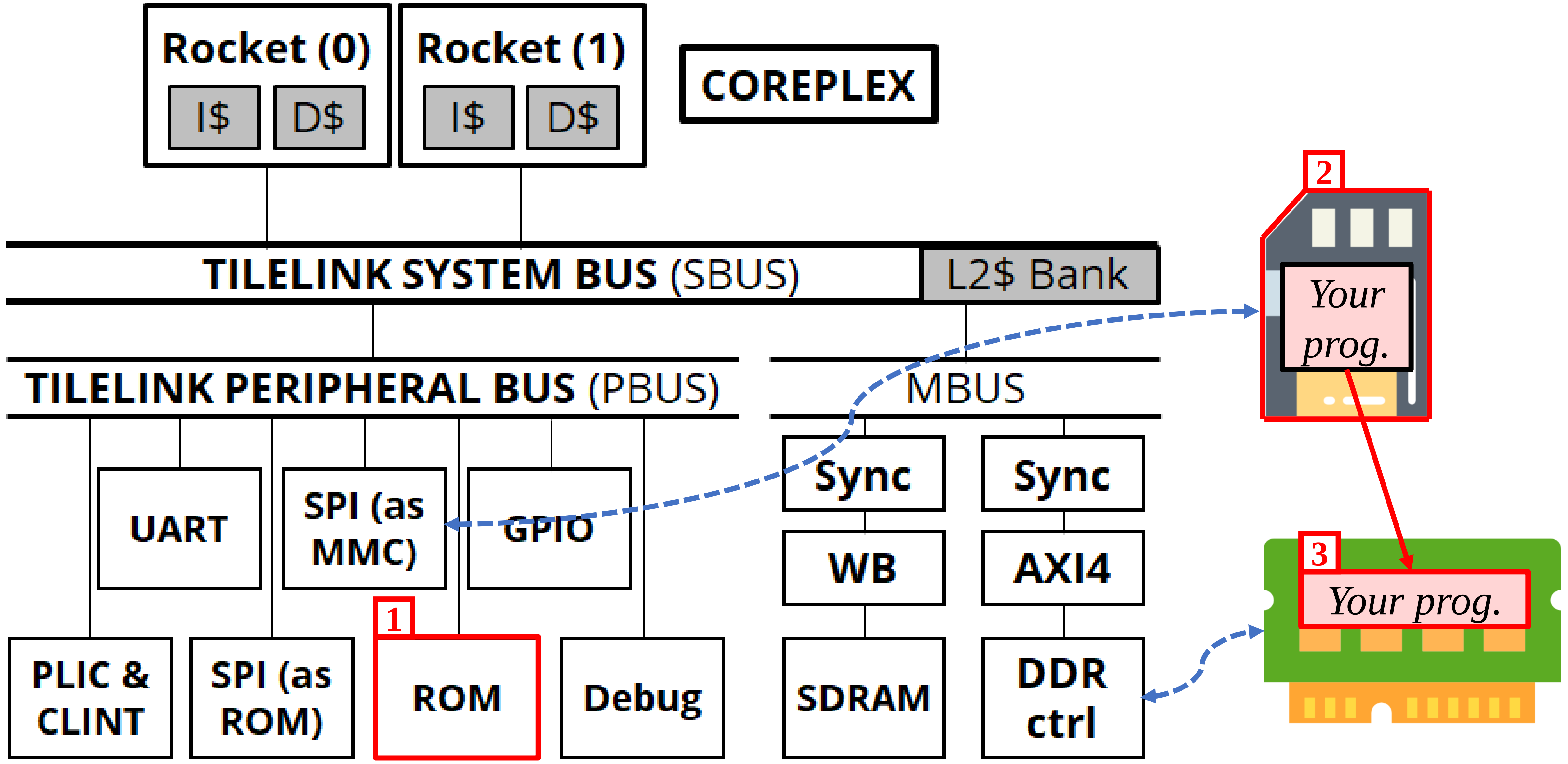
2. Our system's boot sequence (3/5) Copy to RAM



2. Our system's boot sequence (4/5) Execute in RAM



2. Our system's boot sequence (5/5) Architecture view



国立大学法人
電気通信大学
The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

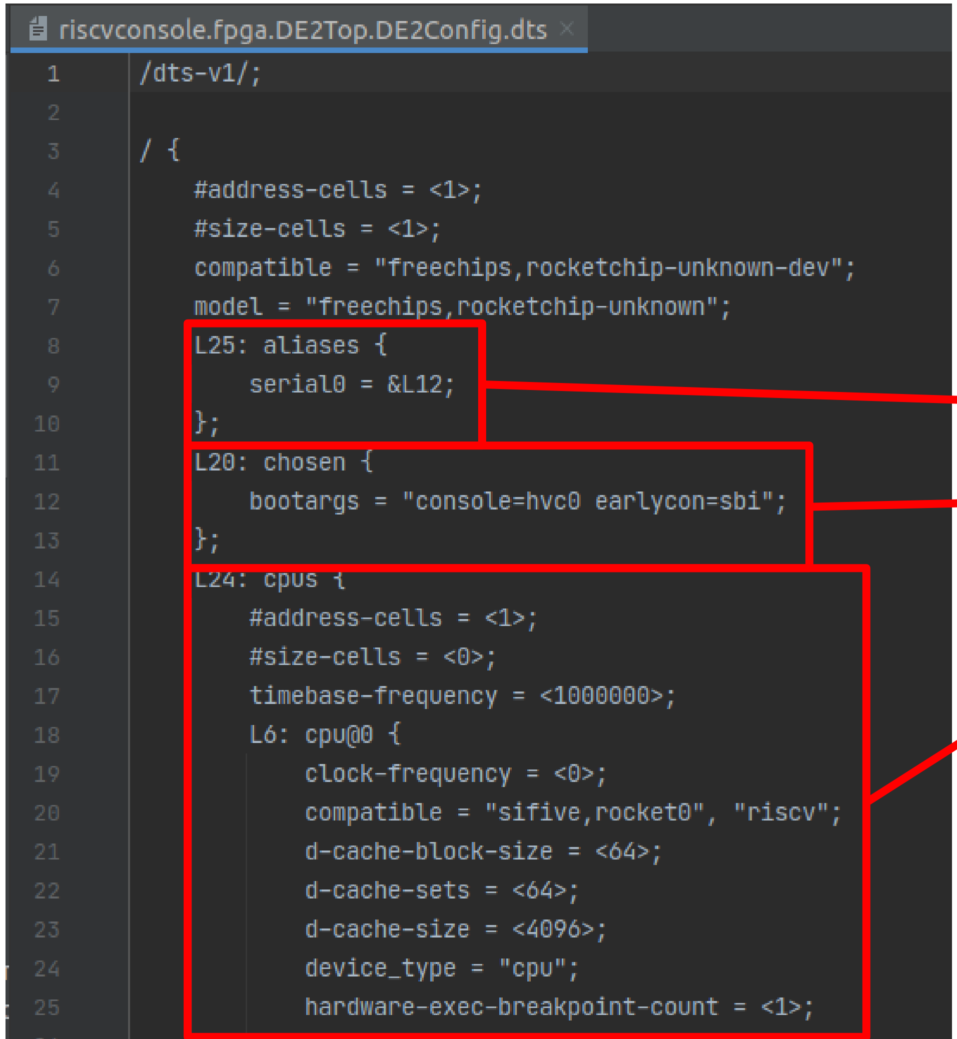
Outline

1. Generic boot sequence
2. Our system's boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

3. Device tree (1/7) The .dts file

After `$ make default`, the device tree files *(**.dts** and **.dtb**)* are generated together with other files *(Verilog, FIRRTL, boot ROM, etc.)* under the **generated-src** folder.

The **.dts** file:



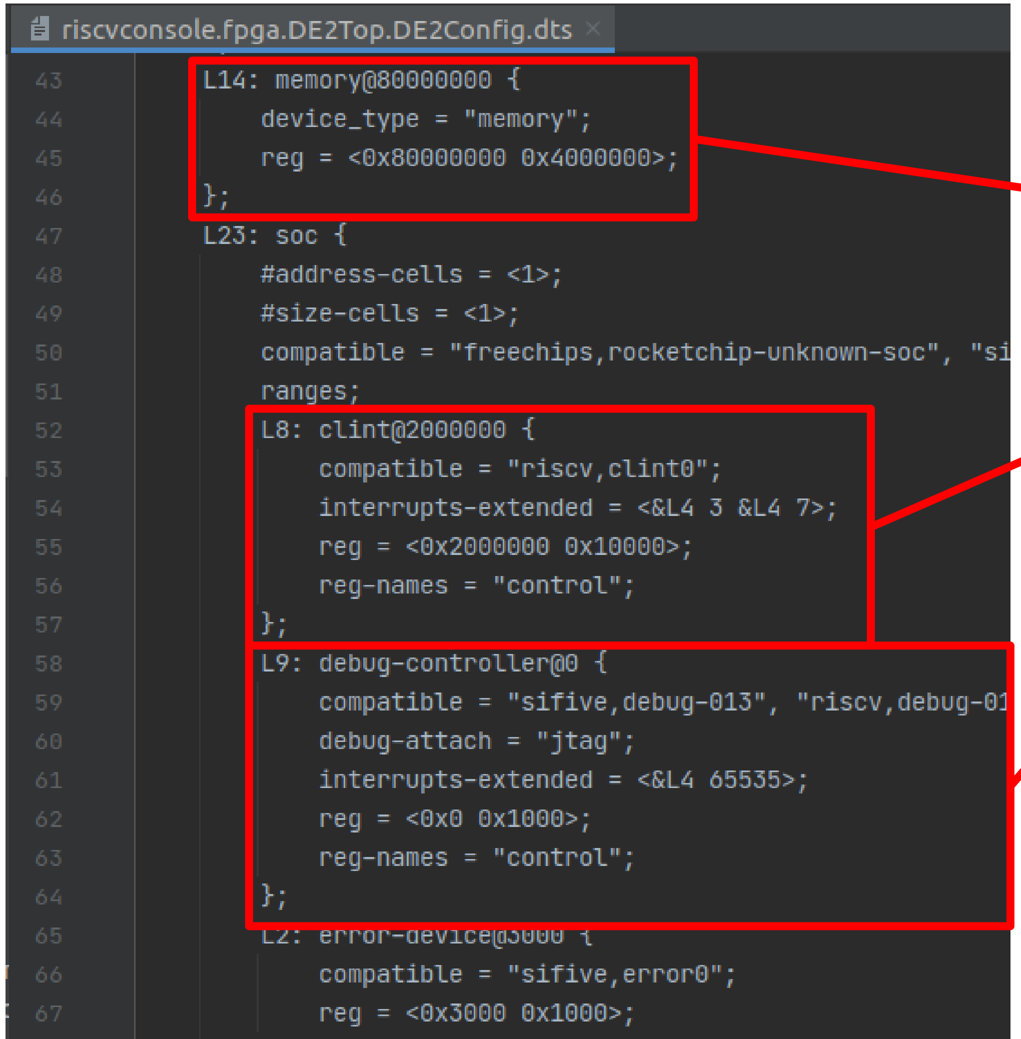
riscvconsole.fpga.DE2Top.DE2Config.dts

```
/dts-v1/;

/ {
	#address-cells = <1>;
	#size-cells = <1>;
	compatible = "freechips,rocketchip-unknown-dev";
	model = "freechips,rocketchip-unknown";
	L25: aliases {
		serial0 = &L12;
	};
	L20: chosen {
		bootargs = "console=hvc0 earlycon=sbi";
	};
	L24: cpus {
		#address-cells = <1>;
		#size-cells = <0>;
		timebase-frequency = <1000000>;
		L6: cpu@0 {
			clock-frequency = <0>;
			compatible = "sifive,rocket0", "riscv";
			d-cache-block-size = <64>;
			d-cache-sets = <64>;
			d-cache-size = <4096>;
			device_type = "cpu";
			hardware-exec-breakpoint-count = <1>;
```

- Main serial interface (**/aliases**)
- Boot arguments for linux (**/chosen**)
- Processors (**/cpus**)

3. Device tree (2/7) The .dts file

After `$ make default`, the device tree files *(**.dts** and **.dtb**)* are generated together with other files *(Verilog, FIRRTL, boot ROM, etc.)* under the **generated-src** folder.
The **.dts** file:



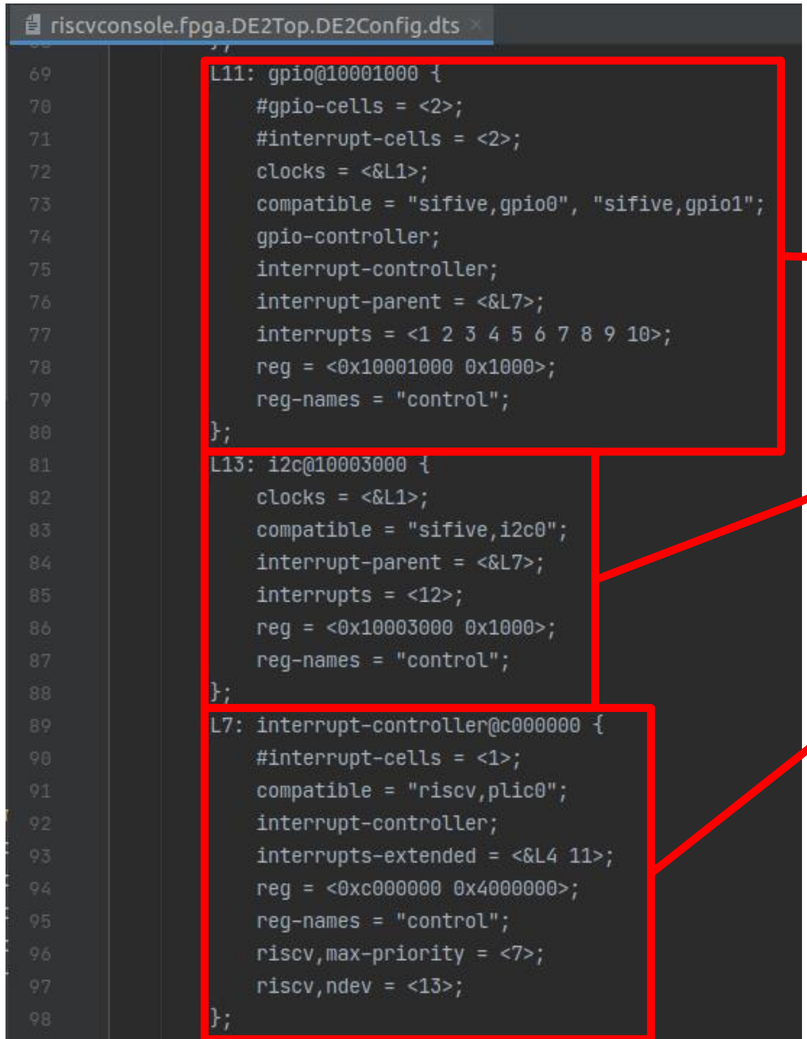
riscvconsole.fpga.DE2Top.DE2Config.dts

```
        L14: memory@80000000 {
            device_type = "memory";
            reg = <0x80000000 0x4000000>;
        };
        L23: soc {
            #address-cells = <1>;
            #size-cells = <1>;
            compatible = "freechips,rocketchip-unknown-soc", "si
            ranges;
            L8: clint@2000000 {
                compatible = "riscv,clint0";
                interrupts-extended = <&L4 3 &L4 7>;
                reg = <0x2000000 0x10000>;
                reg-names = "control";
            };
            L9: debug-controller@0 {
                compatible = "sifive,debug-013", "riscv,debug-01
                debug-attach = "jtag";
                interrupts-extended = <&L4 65535>;
                reg = <0x0 0x1000>;
                reg-names = "control";
            };
            L2: error-device@3000 {
                compatible = "sifive,error0";
                reg = <0x3000 0x1000>;
```

- Memories (**/memory**)
- Core-Local Interrupts (**/soc/clint**)
- Debug controller (**/soc/debug**)

3. Device tree (3/7) The .dts file

After `$ make default`, the device tree files *(**.dts** and **.dtb**)* are generated together with other files *(Verilog, FIRRTL, boot ROM, etc.)* under the **generated-src** folder.
The **.dts** file:

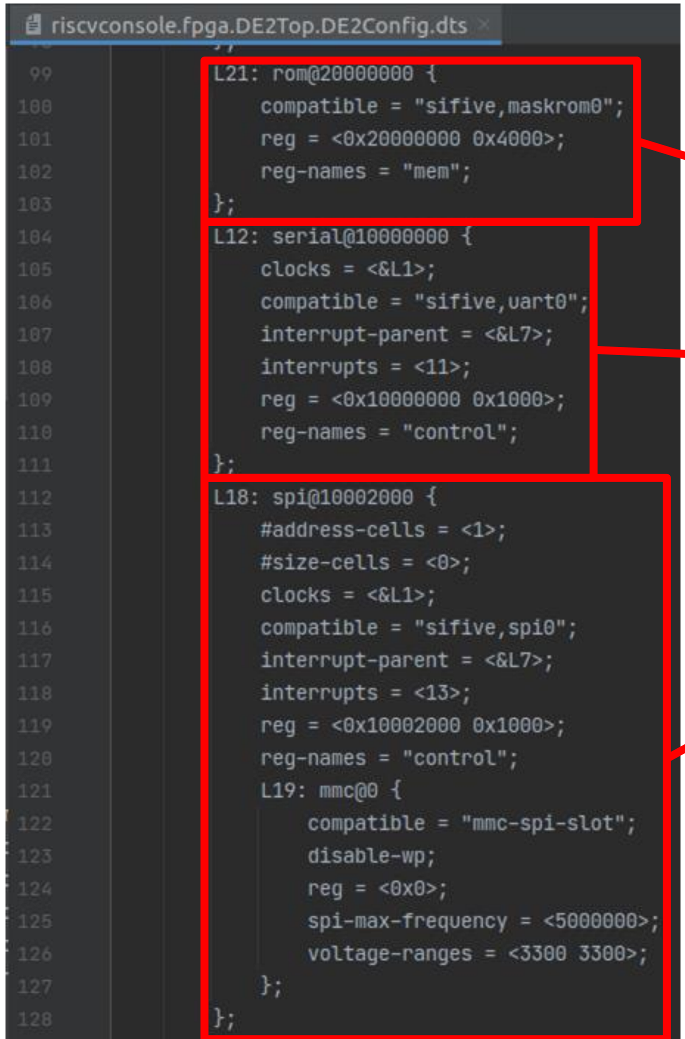


```
riscvconsole.fpga.DE2Top.DE2Config.dts
L11: gpio@10001000 {
    #gpio-cells = <2>;
    #interrupt-cells = <2>;
    clocks = <&L1>;
    compatible = "sifive,gpio0", "sifive,gpio1";
    gpio-controller;
    interrupt-controller;
    interrupt-parent = <&L7>;
    interrupts = <1 2 3 4 5 6 7 8 9 10>;
    reg = <0x10001000 0x1000>;
    reg-names = "control";
};
L13: i2c@10003000 {
    clocks = <&L1>;
    compatible = "sifive,i2c0";
    interrupt-parent = <&L7>;
    interrupts = <12>;
    reg = <0x10003000 0x1000>;
    reg-names = "control";
};
L7: interrupt-controller@c000000 {
    #interrupt-cells = <1>;
    compatible = "riscv,plic0";
    interrupt-controller;
    interrupts-extended = <&L4 11>;
    reg = <0xc000000 0x4000000>;
    reg-names = "control";
    riscv,max-priority = <7>;
    riscv,ndev = <13>;
};
```

- GPIO (**/soc/gpio**)
- I2C (**/soc/i2c**)
- Platform Level Interrupts (**/soc/interrupt-controller**)

3. Device tree (4/7) The .dts file

After `$ make default`, the device tree files *(**.dts** and **.dtb**)* are generated together with other files *(Verilog, FIRRTL, boot ROM, etc.)* under the **generated-src** folder.
The **.dts** file:



riscvconsole.fpga.DE2Top.DE2Config.dts

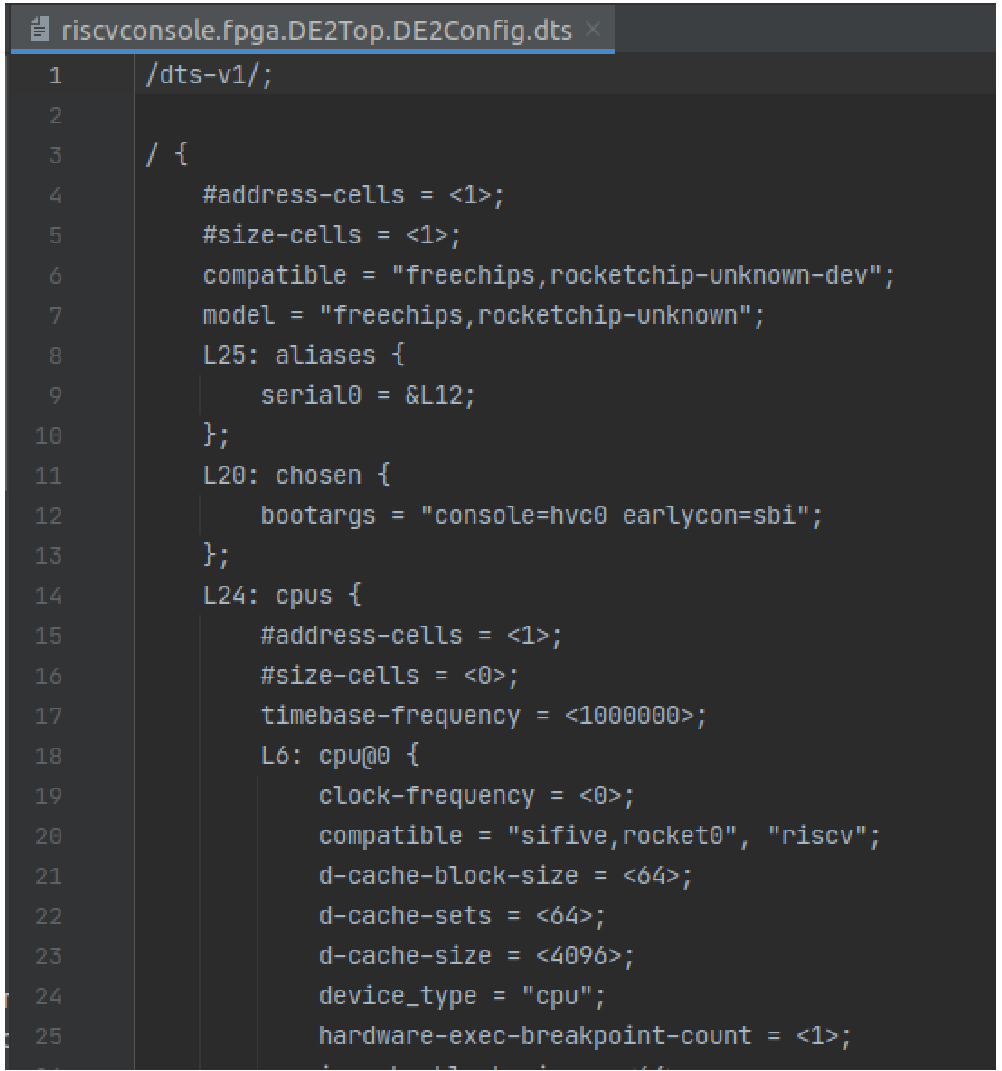
```
L21: rom@20000000 {
    compatible = "sifive,maskrom0";
    reg = <0x20000000 0x4000>;
    reg-names = "mem";
};
L12: serial@10000000 {
    clocks = <&L1>;
    compatible = "sifive,uart0";
    interrupt-parent = <&L7>;
    interrupts = <11>;
    reg = <0x10000000 0x1000>;
    reg-names = "control";
};
L18: spi@10002000 {
    #address-cells = <1>;
    #size-cells = <0>;
    clocks = <&L1>;
    compatible = "sifive,spi0";
    interrupt-parent = <&L7>;
    interrupts = <13>;
    reg = <0x10002000 0x1000>;
    reg-names = "control";
    L19: mmc@0 {
        compatible = "mmc-spi-slot";
        disable-wp;
        reg = <0x0>;
        spi-max-frequency = <5000000>;
        voltage-ranges = <3300 3300>;
    };
};
```

- ROM (**/soc/rom**)
- UART (**/soc/serial**)
- SPI (**/soc/spi**)
  (for SD-card)

3. Device tree (5/7) The .dtb file

In the compiled program file, the device tree is attached by its binary version, the **.dtb**.

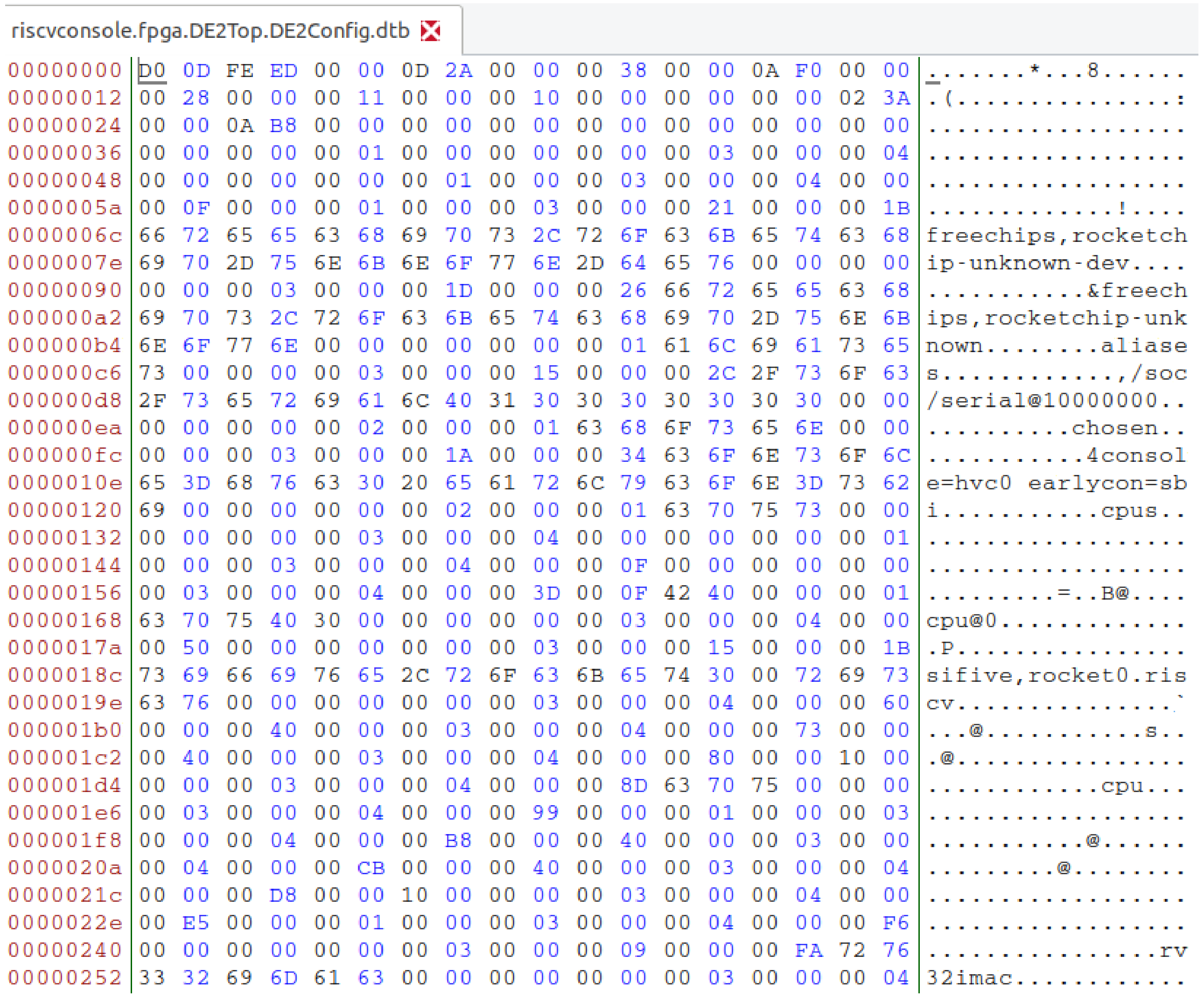
The **.dts** file:



riscvconsole.fpga.DE2Top.DE2Config.dts

```
/dts-v1/;

/ {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "freechips,rocketchip-unknown-dev";
    model = "freechips,rocketchip-unknown";
    L25: aliases {
        serial0 = &L12;
    };
    L20: chosen {
        bootargs = "console=hvc0 earlycon=sbi";
    };
    L24: cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        timebase-frequency = <1000000>;
        L6: cpu@0 {
            clock-frequency = <0>;
            compatible = "sifive,rocket0", "riscv";
            d-cache-block-size = <64>;
            d-cache-sets = <64>;
            d-cache-size = <4096>;
            device_type = "cpu";
            hardware-exec-breakpoint-count = <1>;
```

The **.dtb** file:

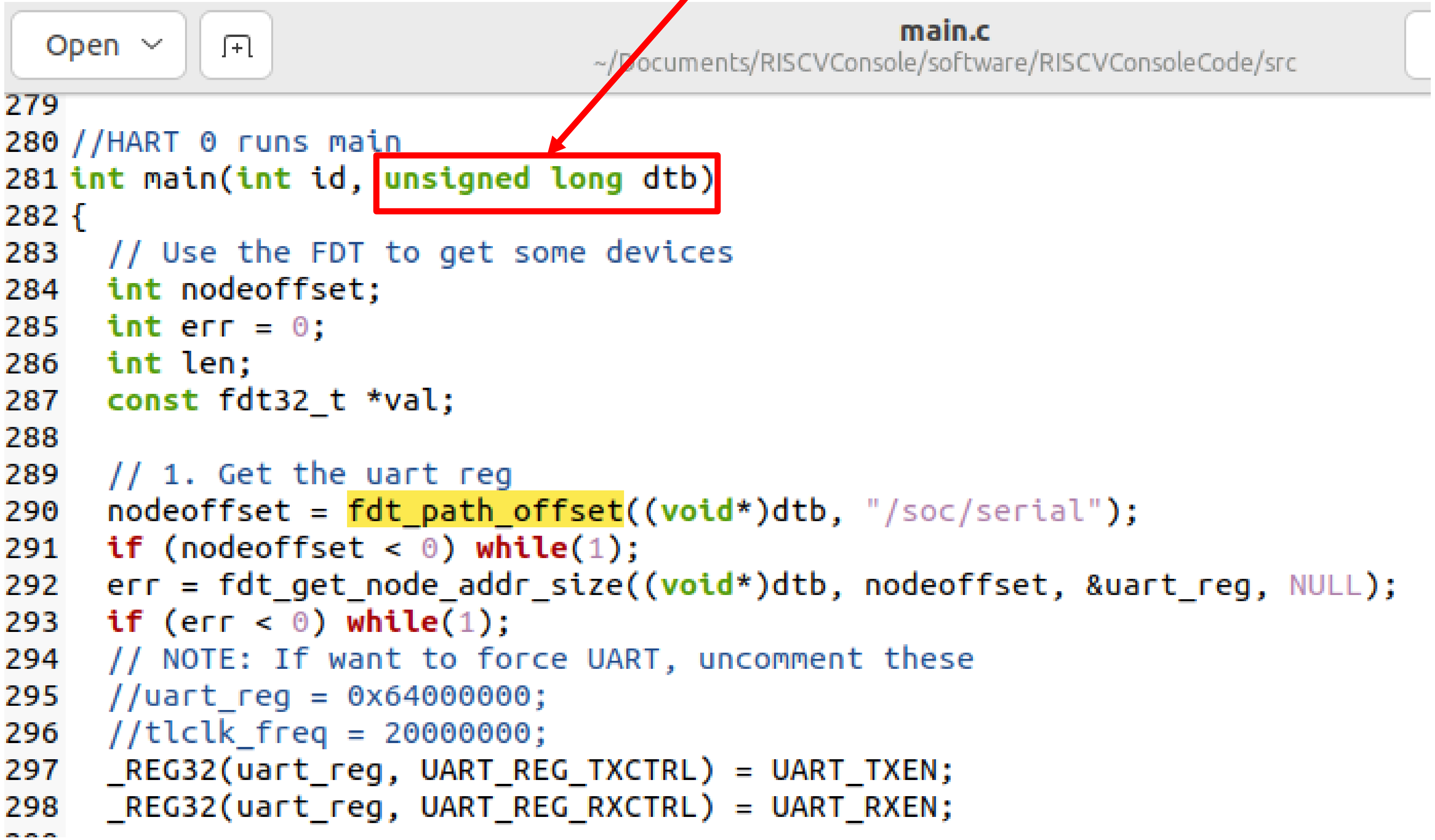


riscvconsole.fpga.DE2Top.DE2Config.dtb

```
00000000 D0 0D FE ED 00 00 0D 2A 00 00 00 38 00 00 0A F0 00 00 .......*...8......
00000012 00 28 00 00 00 11 00 00 00 10 00 00 00 00 00 00 02 3A .(...............:
00000024 00 00 0A B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 00 00 00 00 00 01 00 00 00 00 00 00 00 03 00 00 00 04 ..................
00000048 00 00 00 00 00 00 00 01 00 00 00 03 00 00 00 04 00 00 ..................
0000005a 00 0F 00 00 00 01 00 00 00 03 00 00 00 21 00 00 00 1B .............!....
0000006c 66 72 65 65 63 68 69 70 73 2C 72 6F 63 6B 65 74 63 68 freechips,rocketch
0000007e 69 70 2D 75 6E 6B 6E 6F 77 6E 2D 64 65 76 00 00 00 00 ip-unknown-dev....
00000090 00 00 00 03 00 00 00 1D 00 00 00 26 66 72 65 65 63 68 ...........&freech
000000a2 69 70 73 2C 72 6F 63 6B 65 74 63 68 69 70 2D 75 6E 6B ips,rocketchip-unk
000000b4 6E 6F 77 6E 00 00 00 00 00 00 00 01 61 6C 69 61 73 65 nown........aliase
000000c6 73 00 00 00 00 03 00 00 00 15 00 00 00 2C 2F 73 6F 63 s............,/soc
000000d8 2F 73 65 72 69 61 6C 40 31 30 30 30 30 30 30 30 00 00 /serial@10000000..
000000ea 00 00 00 00 00 02 00 00 00 01 63 68 6F 73 65 6E 00 00 ..........chosen..
000000fc 00 00 00 03 00 00 00 1A 00 00 00 34 63 6F 6E 73 6F 6C ...........4consol
0000010e 65 3D 68 76 63 30 20 65 61 72 6C 79 63 6F 6E 3D 73 62 e=hvc0 earlycon=sb
00000120 69 00 00 00 00 00 00 02 00 00 00 01 63 70 75 73 00 00 i...........cpus..
00000132 00 00 00 00 00 03 00 00 00 04 00 00 00 00 00 00 00 01 ..................
00000144 00 00 00 03 00 00 00 04 00 00 00 0F 00 00 00 00 00 00 ..................
00000156 00 03 00 00 00 04 00 00 00 3D 00 0F 42 40 00 00 00 01 .........=..B@....
00000168 63 70 75 40 30 00 00 00 00 00 00 03 00 00 00 04 00 00 cpu@0.............
0000017a 00 50 00 00 00 00 00 00 00 03 00 00 00 15 00 00 00 1B .P................
0000018c 73 69 66 69 76 65 2C 72 6F 63 6B 65 74 30 00 72 69 73 sifive,rocket0.ris
0000019e 63 76 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00 60 cv...............`
000001b0 00 00 00 40 00 00 00 03 00 00 00 04 00 00 00 73 00 00 ...@...........s..
000001c2 00 40 00 00 00 03 00 00 00 04 00 00 00 80 00 00 10 00 .@................
000001d4 00 00 00 03 00 00 00 04 00 00 00 8D 63 70 75 00 00 00 ............cpu...
000001e6 00 03 00 00 00 04 00 00 00 99 00 00 00 01 00 00 00 03 ..................
000001f8 00 00 00 04 00 00 00 B8 00 00 00 40 00 00 00 03 00 00 ...........@......
0000020a 00 04 00 00 00 CB 00 00 00 40 00 00 00 03 00 00 00 04 .........@........
0000021c 00 00 00 D8 00 00 10 00 00 00 00 03 00 00 00 04 00 00 ..................
0000022e 00 E5 00 00 00 01 00 00 00 03 00 00 00 04 00 00 00 F6 ..................
00000240 00 00 00 00 00 00 00 03 00 00 00 09 00 00 00 FA 72 76 ................rv
00000252 33 32 69 6D 61 63 00 00 00 00 00 00 00 03 00 00 00 04 32imac............
```

3. **Device tree** (6/7) To call **dtb** in software

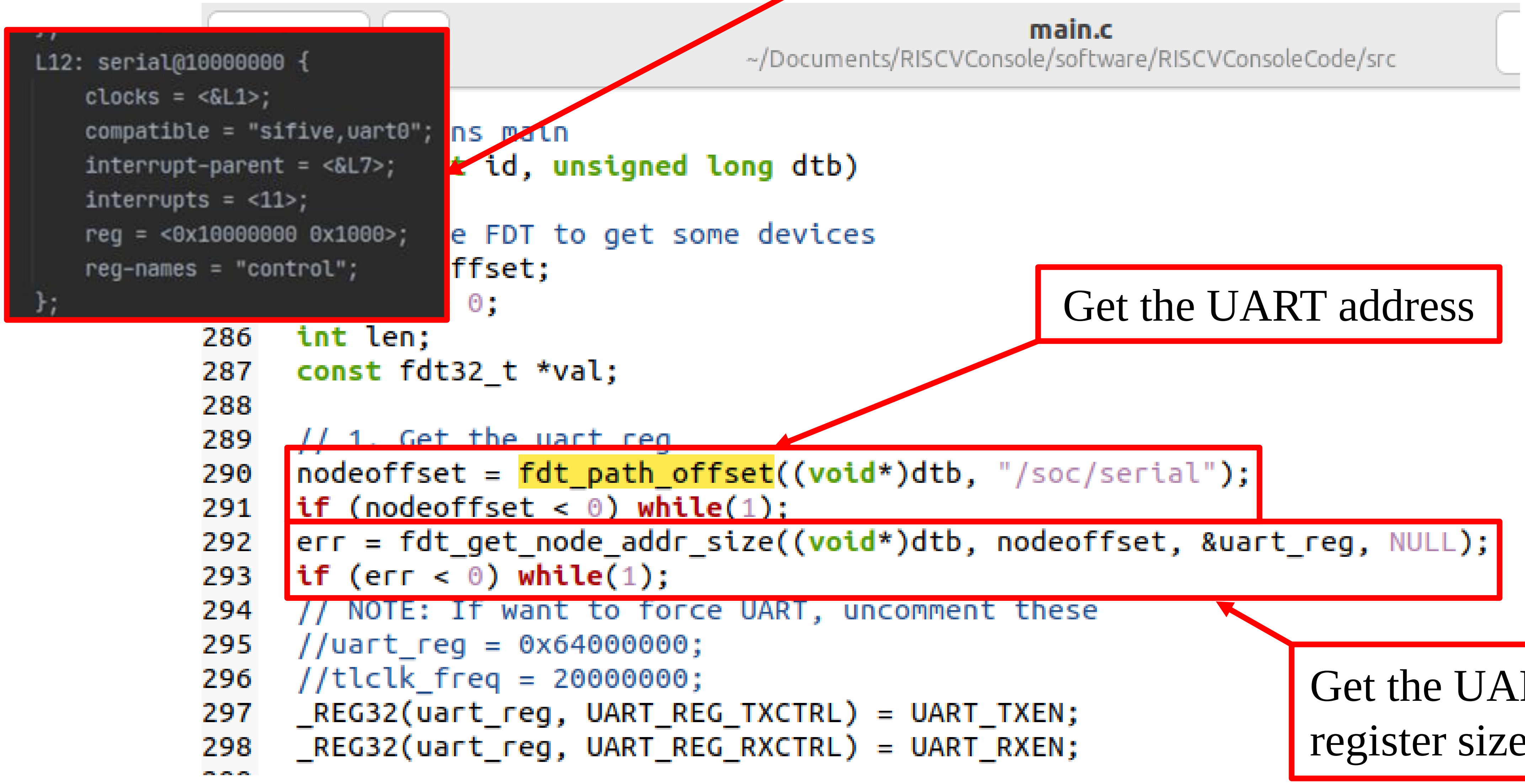
The pointer of **.dtb** will be passed to the `main()` by the bootloader.



```

//HART 0 runs main
int main(int id, unsigned long dtb)
{
  // Use the FDT to get some devices
  int nodeoffset;
  int err = 0;
  int len;
  const fdt32_t *val;

  // 1. Get the uart reg
  nodeoffset = fdt_path_offset((void*)dtb, "/soc/serial");
  if (nodeoffset < 0) while(1);
  err = fdt_get_node_addr_size((void*)dtb, nodeoffset, &uart_reg, NULL);
  if (err < 0) while(1);
  // NOTE: If want to force UART, uncomment these
  //uart_reg = 0x64000000;
  //tlclk_freq = 20000000;
  _REG32(uart_reg, UART_REG_TXCTRL) = UART_TXEN;
  _REG32(uart_reg, UART_REG_RXCTRL) = UART_RXEN;
```

3. Device tree (7/7) To call dtb in software

For example: to use the UART from the device-tree



```
L12: serial@10000000 {
    clocks = <&L1>;
    compatible = "sifive,uart0";
    interrupt-parent = <&L7>;
    interrupts = <11>;
    reg = <0x10000000 0x1000>;
    reg-names = "control";
};
```

```
ns main
t id, unsigned long dtb)

e FDT to get some devices
ffset;
 0;
int len;
const fdt32_t *val;

// 1. Get the uart reg
nodeoffset = fdt_path_offset((void*)dtb, "/soc/serial");
if (nodeoffset < 0) while(1);
err = fdt_get_node_addr_size((void*)dtb, nodeoffset, &uart_reg, NULL);
if (err < 0) while(1);
// NOTE: If want to force UART, uncomment these
//uart_reg = 0x64000000;
//tlclk_freq = 20000000;
_REG32(uart_reg, UART_REG_TXCTRL) = UART_TXEN;
_REG32(uart_reg, UART_REG_RXCTRL) = UART_RXEN;
```

Get the UART address

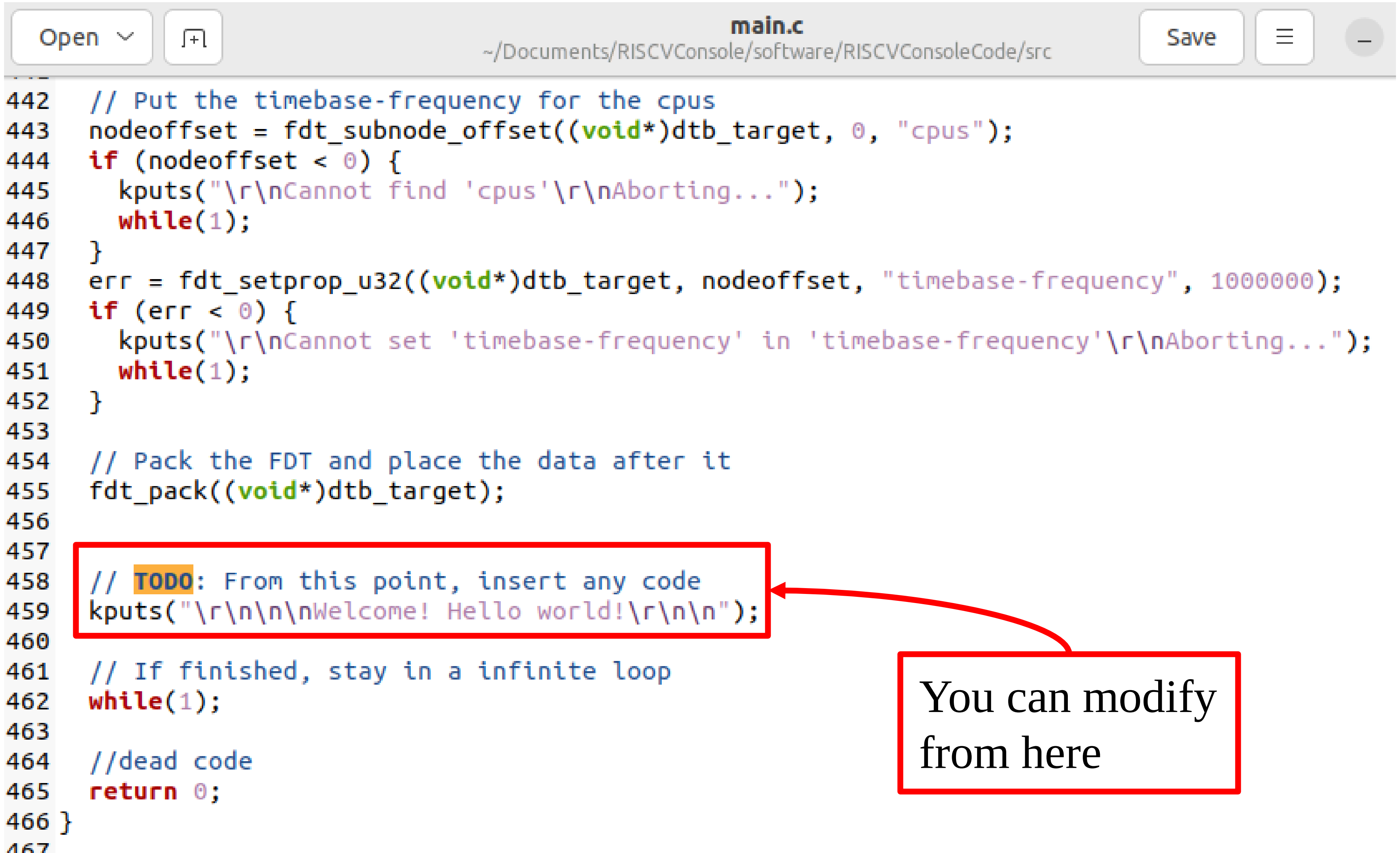
Get the UART register size

Outline

1. Generic boot sequence
2. Our system's boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

4. Modifying software (1/4) Modifying from the example

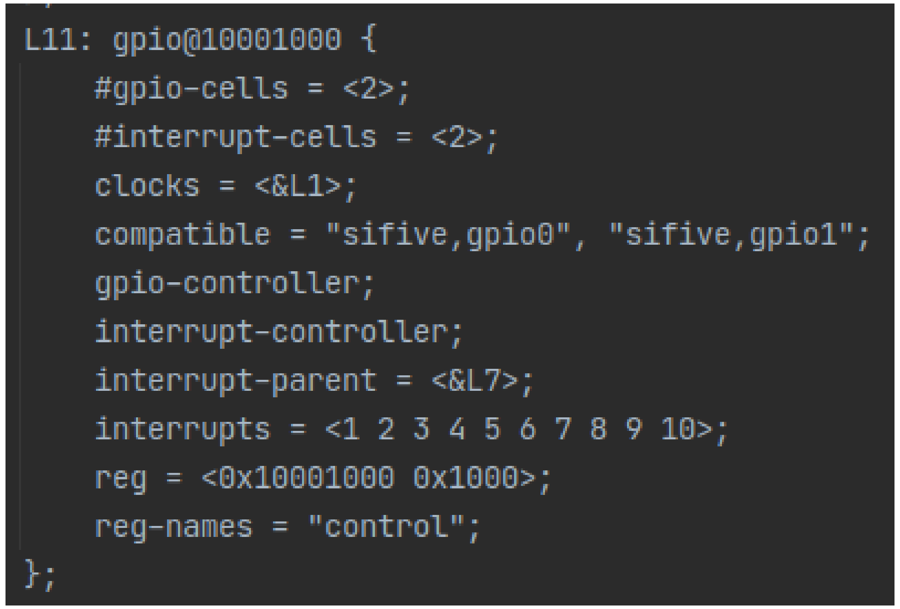
The source is at: `RISCVConsole/software/RISCVConsoleCode/src/main.c`



```c
  // Put the timebase-frequency for the cpus
  nodeoffset = fdt_subnode_offset((void*)dtb_target, 0, "cpus");
  if (nodeoffset < 0) {
    kputs("\r\nCannot find 'cpus'\r\nAborting...");
    while(1);
  }
  err = fdt_setprop_u32((void*)dtb_target, nodeoffset, "timebase-frequency", 1000000);
  if (err < 0) {
    kputs("\r\nCannot set 'timebase-frequency' in 'timebase-frequency'\r\nAborting...");
    while(1);
  }

  // Pack the FDT and place the data after it
  fdt_pack((void*)dtb_target);


  // TODO: From this point, insert any code
  kputs("\r\n\n\nWelcome! Hello world!\r\n\n");

  // If finished, stay in a infinite loop
  while(1);

  //dead code
  return 0;
}
```

You can modify from here

4. Modifying software (2/4) To use the GPIO



```
L11: gpio@10001000 {
    #gpio-cells = <2>;
    #interrupt-cells = <2>;
    clocks = <&L1>;
    compatible = "sifive,gpio0", "sifive,gpio1";
    gpio-controller;
    interrupt-controller;
    interrupt-parent = <&L7>;
    interrupts = <1 2 3 4 5 6 7 8 9 10>;
    reg = <0x10001000 0x1000>;
    reg-names = "control";
};
```

Example: using GPIO

Get the GPIO address

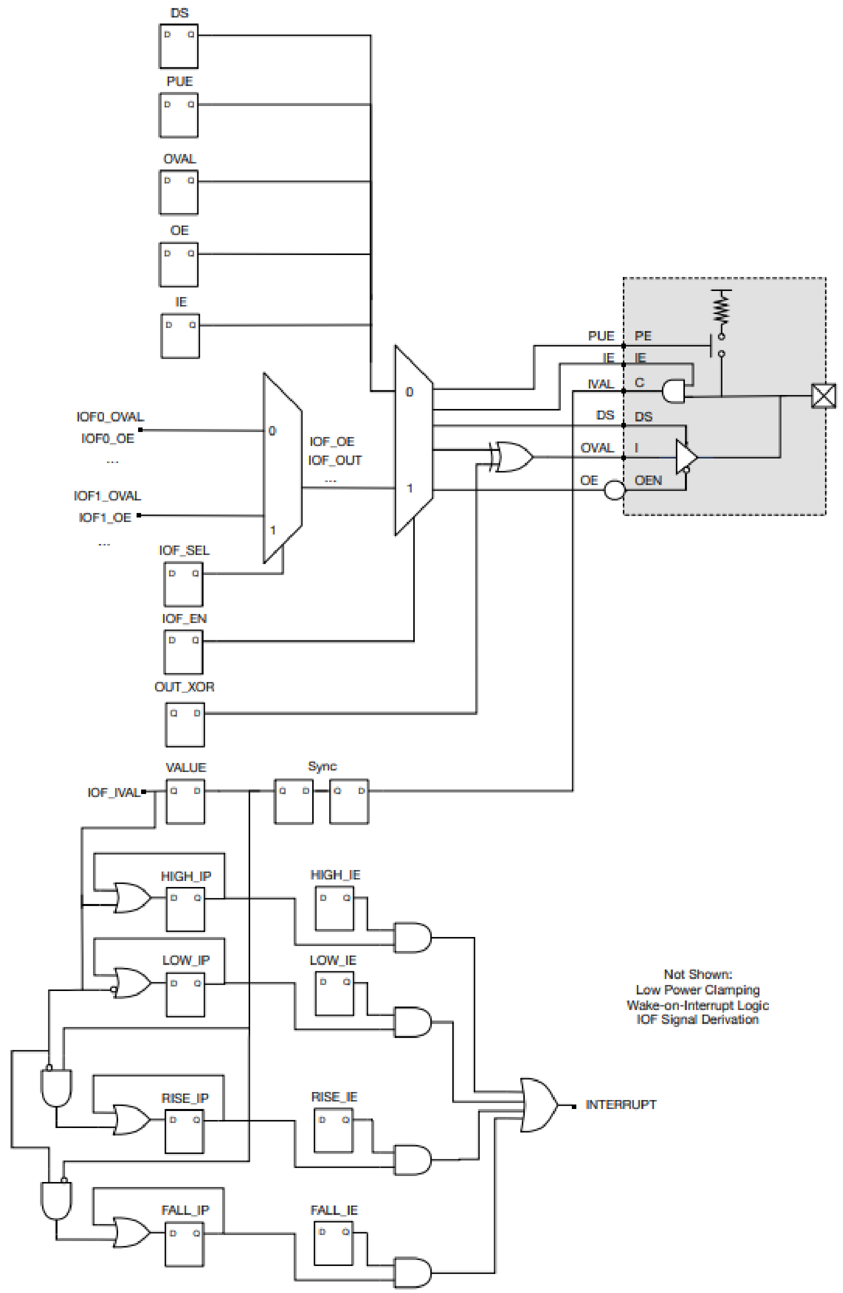
Check if the address exists, report and freeze if not.

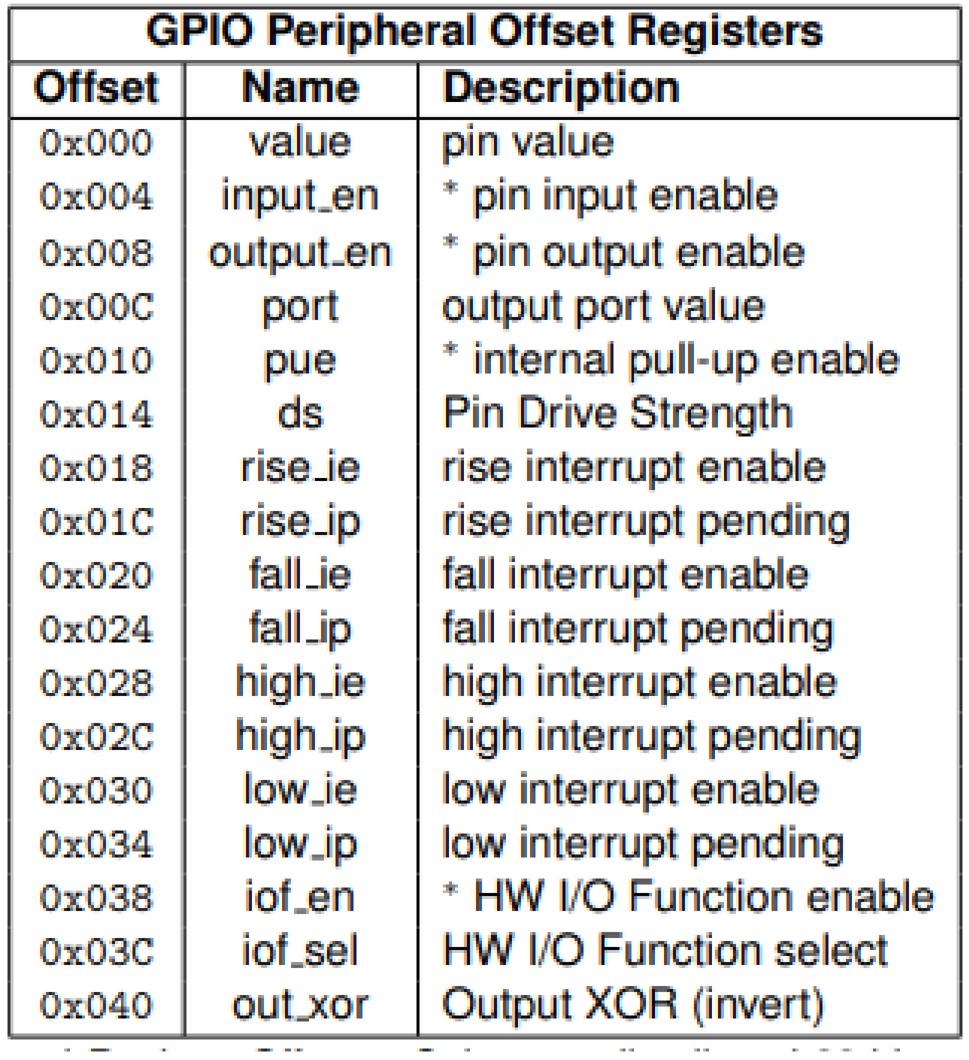
Get the GPIO size

Report and freeze if cannot detect the size.

```
// Detect the GPIO
unsigned long gpio_reg;
nodeoffset = fdt_path_offset((void*)dtb_target, "/soc/gpio");
      if (nodeoffset < 0) {
        kputs("\r\nCannot find '/soc/gpio'\r\nAborting...");
  while(1);
      }
      err = fdt_get_node_addr_size((void*)dtb_target, nodeoffset, &gpio_reg, NULL);
if (err < 0) {
  kputs("\r\nCannot get reg space from '/soc/gpio'\r\nAborting...");
  while(1);
}
```

4. Modifying software (3/4) GPIO memory map



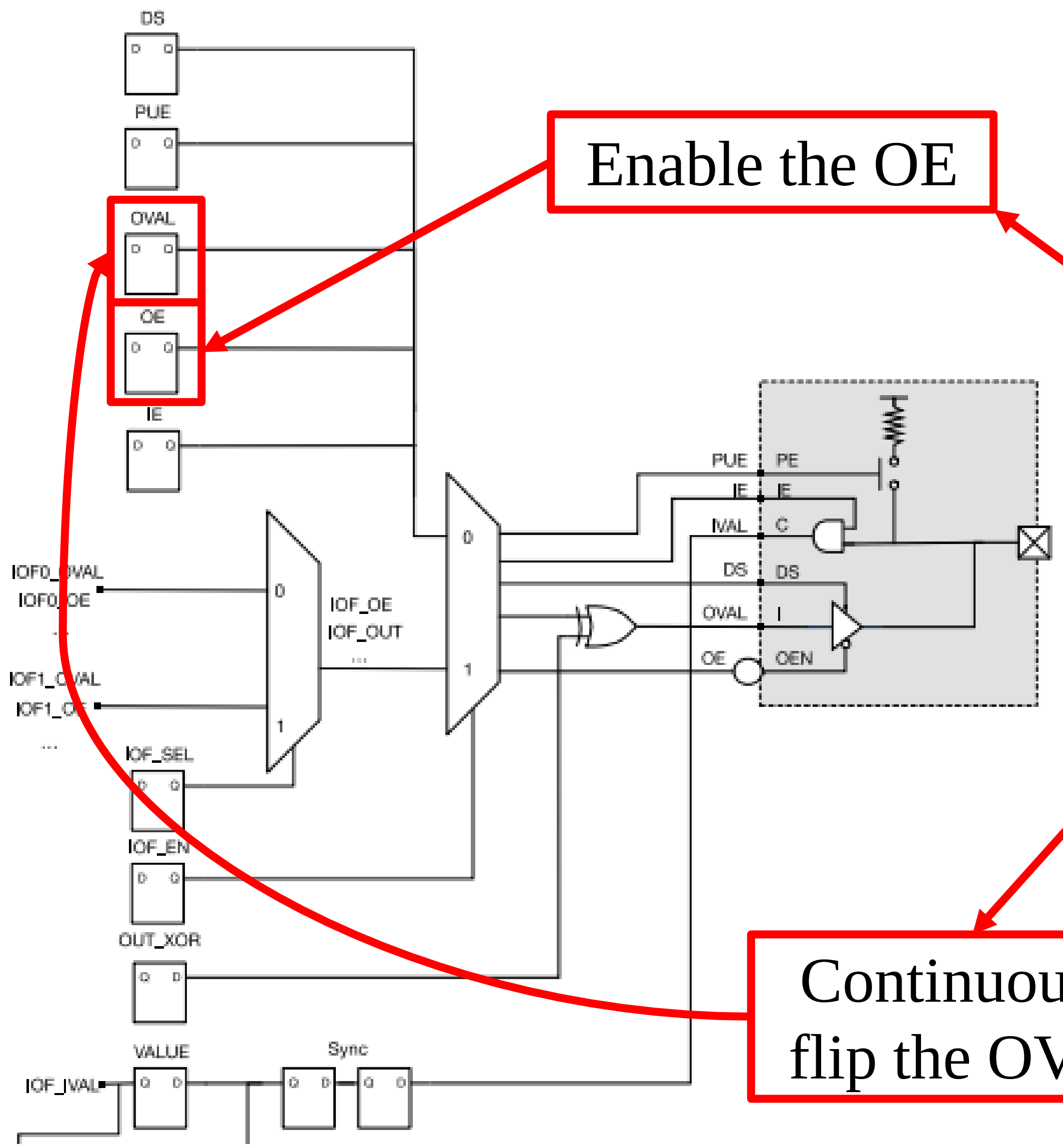


GPIO Peripheral Offset Registers

Offset	Name	Description
0x000	value	pin value
0x004	input_en	* pin input enable
0x008	output_en	* pin output enable
0x00C	port	output port value
0x010	pue	* internal pull-up enable
0x014	ds	Pin Drive Strength
0x018	rise_ie	rise interrupt enable
0x01C	rise_ip	rise interrupt pending
0x020	fall_ie	fall interrupt enable
0x024	fall_ip	fall interrupt pending
0x028	high_ie	high interrupt enable
0x02C	high_ip	high interrupt pending
0x030	low_ie	low interrupt enable
0x034	low_ip	low interrupt pending
0x038	iof_en	* HW I/O Function enable
0x03C	iof_sel	HW I/O Function select
0x040	out_xor	Output XOR (invert)

Reference link *(page 56, Chapter 17)*:
https://static.dev.sifive.com/FE310-G000.pdf

4. Modifying software (4/4) GPIO code example



Enable the OE

Continuously flip the OVAL

```
// Enable the GPIO outputs
_REG32(gpio_reg, GPIO_OUTPUT_EN) = 0x3FF;

// Switch the GPIO in an infinite value
uint32_t v = 0;
while(1) {
 _REG32(gpio_reg, GPIO_OUTPUT_VAL) = v;
 v = ~v;
 for(int i = 0; i < 50000000; i++);
}
```

国立大学法人 電気通信大学 The University of Electro-Communications

Pham Laboratory Integrated circuit design laboratory

Outline

1. Generic boot sequence
2. Our system's boot sequence
3. Device tree
4. Modifying software after boot
5. Practice: using GPIO

5. Practice: using GPIO (1/1)

- Our system has 8-bit GPIOs with [3:0] is mapped to four LEDs and [7:4] is mapped to four switches.
- Header file can be found at:
  `RISCVConsole/software/RISCVConsoleCode/include/devices/gpio.h`

Exercise 1:

Map four switches to four LEDs.

Exercise 2:

Make 1 LED flashing with about 1 second interval.

国立大学法人
電気通信大学
The University of Electro-Communications

Pham Laboratory
Integrated circuit design laboratory

THANK YOU

2022/11