start_kernel() ---> time_init()

arch/arm/kernel/time.c

```
void __init time_init(void)
{
        if (machine_desc->init_time) {
                machine_desc->init_time();
        } else {
#ifdef CONFIG_COMMON_CLK
                of_clk_init(NULL);
#endif
                clocksource_of_init();
        }
}
```

drivers/clk/clk.c

```
void __init of_clk_init(const struct of_device_id *matches)

/**
 * of_clk_init() - Scan and init clock providers from the DT
 * @matches: array of compatible values and init functions for providers.
 *
 * This function scans the device tree for matching clock providers
```

```
 * and calls their initialization functions. It also does it by trying
 * to follow the dependencies.
 */
void __init of_clk_init(const struct of_device_id *matches)
{
	const struct of_device_id *match;
	struct device_node *np;
	struct clock_provider *clk_provider, *next;
	bool is_init_done;
	bool force = false;

	if (!matches)
		matches = &__clk_of_table;

	/* First prepare the list of the clocks providers */
	for_each_matching_node_and_match(np, matches, &match) {
		struct clock_provider *parent =
			kzalloc(sizeof(struct clock_provider),	GFP_KERNEL);

		parent->clk_init_cb = match->data;
		parent->np = np;
		list_add_tail(&parent->node, &clk_provider_list);
	}

	while (!list_empty(&clk_provider_list)) {
```

```
        is_init_done = false;
        list_for_each_entry_safe(clk_provider, next,
                                 &clk_provider_list, node) {
            if (force || parent_ready(clk_provider->np)) {

                clk_provider->clk_init_cb(clk_provider->np);
                of_clk_set_defaults(clk_provider->np, true);

                list_del(&clk_provider->node);
                kfree(clk_provider);
                is_init_done = true;
            }
        }

        /*
         * We didn't manage to initialize any of the
         * remaining providers during the last loop, so now we
         * initialize all the remaining ones unconditionally
         * in case the clock parent was not mandatory
         */
        if (!is_init_done)
            force = true;
    }
}
```

```
extern struct of_device_id __clk_of_table;


/*
 * Struct used for matching a device
 */
struct of_device_id
{
        char name[32];
        char type[32];
        char compatible[128];
        const void *data;
};


CLK_OF_DECLARE(pegmatite_clkgate, "marvell,pegmatite-clkgate", of_pegmatite_clkgate_setup);

#define CLK_OF_DECLARE(name, compat, fn) OF_DECLARE_1(clk, name, compat, fn)


OF_DECLARE_1(clk, name, compat, fn)

OF_DECLARE_1(clk, pegmatite_clkgate ,  "marvell,pegmatite-
clkgate" ,  of_pegmatite_clkgate_setup)


#define OF_DECLARE_1(table, name, compat, fn) \
                _OF_DECLARE(table, name, compat, fn, of_init_fn_1)


_OF_DECLARE(clk, pegmatite_clkgate ,  "marvell,pegmatite-
clkgate" ,  of_pegmatite_clkgate_setup, of_init_fn_1)
```

```
typedef void (*of_init_fn_1)(struct device_node *);


#define _OF_DECLARE(table, name, compat, fn, fn_type)                \
        static const struct of_device_id __of_table_##name           \
                __used __section(__##table##_of_table)               \
                 = { .compatible = compat,                           \
                     .data = (fn == (fn_type)NULL) ? fn : fn  }


        static const struct of_device_id __of_table_pegmatite_clkgate __used
__section(__clk__of_table)

= { .compatible = compat,

    .data = (of_pegmatite_clkgate_setup ==  (of_init_fn_1)NULL) ? of_pegmatite_clkgate_setup
: of_pegmatite_clkgate_setup

  }


struct of_device_id
{
        char name[32];
        char type[32];
        char compatible[128];
        const void *data;
} __of_table_pegmatite_clkgate = {
            name = ??? 未定义 ， 0
            type =  ??? 未定义  ， 0
            compatible = "marvell,pegmatite-clkgate";
            data = of_pegmatite_clkgate_setup;

 } __used __section(__clk__of_table)
```

该struct variable存入__clk__of_table section

在poky/build/tmp/work/granite2-poky-linux-gnueabi/linux-granite-upstream/3.18.7+gitAUTOINC+71dd095d17-r0/linux-granite2-standard-build/arch/arm/kernel目录下的vmlinux.lds（奇怪不是linux）

```
 .init.data : {

  *(.init.data) *(.meminit.data) *(.init.rodata) *(.meminit.rodata) . = ALIGN(8); __clk_of_table = .; *(__clk_of_table) *(__clk_of_table_end) . = ALIGN(8); __reservedmem_of_table = .; *(__reservedmem_of_table) *(__reservedmem_of_table_end) . = ALIGN(8); __clksrc_of_table = .; *(__clksrc_of_table) *(__clksrc_of_table_end) . = ALIGN(8); __cpu_method_of_table = .; *(__cpu_method_of_table) *(__cpu_method_of_table_end) . = ALIGN(32); __dtb_start = .; *(.dtb.init.rodata) __dtb_end = .; . = ALIGN(8); __irqchip_of_table = .; *(__irqchip_of_table) *(__irqchip_of_table_end)

  . = ALIGN(16); __setup_start = .; *(.init.setup) __setup_end = .;

  __initcall_start = .; *(.initcallearly.init) __initcall0_start = .; *(.initcall0.init) *(.initcall0s.init) __initcall1_start = .; *(.initcall1.init) *(.initcall1s.init) __initcall2_start = .; *(.initcall2.init) *(.initcall2s.init) __initcall3_start = .; *(.initcall3.init) *(.initcall3s.init) __initcall4_start = .; *(.initcall4.init) *(.initcall4s.init) __initcall5_start = .; *(.initcall5.init) *(.initcall5s.init) __initcallrootfs_start = .; *(.initcallrootfs.init) *(.initcallrootfss.init) __initcall6_start = .; *(.initcall6.init) *(.initcall6s.init) __initcall7_start = .; *(.initcall7.init) *(.initcall7s.init) __initcall_end = .;

  __con_initcall_start = .; *(.con_initcall.init) __con_initcall_end = .;

  __security_initcall_start = .; *(.security_initcall.init) __security_initcall_end = .;

  . = ALIGN(4); __initramfs_start = .; *(.init.ramfs) . = ALIGN(8); *(.init.ramfs.info)

 }
```

```
static const struct of_device_id __clk_of_table_sentinel

        __used __section(__clk_of_table_end);
```

所有通过CLK_OF_DECLARE（）定义的变量都放入__clk__of_table section中，该section的首地址为__clk_of_table，而尾指针为__clk_of_table_sentinel.

/opt/armv7-marvell-linux-gnueabi-hard-4.6.4_x86_64_20140402/bin/arm-marvell-linux-gnueabi-readelf -S vmlinux

Section Headers:

```
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .head.text        PROGBITS        c0008000 008000 0002dc 00  AX  0   0  4
  [ 2] .text             PROGBITS        c0008300 008300 44c578 00  AX  0   0 64
  [ 3] .text.head        PROGBITS        c0454878 454878 000044 00  AX  0   0  4
  [ 4] sram_cpuidle      PROGBITS        c04548bc 4548bc 0000d0 00  AX  0   0  4
  [ 5] .rodata           PROGBITS        c0455000 455000 128a90 00   A  0   0 64
  [ 6] .pci_fixup        PROGBITS        c057da90 57da90 001690 00   A  0   0  4
  [ 7] __ksymtab         PROGBITS        c057f120 57f120 007040 00   A  0   0  4
  [ 8] __ksymtab_gpl     PROGBITS        c0586160 586160 0051f0 00   A  0   0  4
  [ 9] __ksymtab_strings PROGBITS        c058b350 58b350 01be7f 00   A  0   0  1
  [10] __param           PROGBITS        c05a71d0 5a71d0 000ad0 00   A  0   0  4
  [11] __modver          PROGBITS        c05a7ca0 5a7ca0 000360 00   A  0   0  4
  [12] __ex_table        PROGBITS        c05a8000 5a8000 000f58 00   A  0   0  8
  [13] .ARM.unwind_idx   ARM_EXIDX       c05a8f58 5a8f58 021a10 00  AL 18   0  4
  [14] .ARM.unwind_tab   PROGBITS        c05ca968 5ca968 0034d4 00   A  0   0  4
  [15] .notes            NOTE            c05cde3c 5cde3c 000024 00  AX  0   0  4
  [16] .vectors          PROGBITS        00000000 5d0000 000020 00  AX  0   0  4
  [17] .stubs            PROGBITS        00001000 5d1000 0002c0 00  AX  0   0 32
  [18] .init.text        PROGBITS        c05ce2e0 5d62e0 0215e8 00  AX  0   0 32
  [19] .exit.text        PROGBITS        c05ef8c8 5f78c8 000c44 00  AX  0   0  4
  [20] .init.arch.info   PROGBITS        c05f0510 5f8510 0000f0 00   A  0   0  8
```

[21] .init.tagtable PROGBITS c05f0600 5f8600 000040 00 A 0 0 4

[22] .init.smpalt PROGBITS c05f0640 5f8640 004228 00 A 0 0 4

[23] .init.pv_table PROGBITS c05f4868 5fc868 000494 00 A 0 0 1

[24] .init.data PROGBITS c05f4d00 5fcd00 00886c 00 WA 0 0 8

[25] .data..percpu PROGBITS c05fe000 606000 0020c0 00 WA 0 0 64

[26] .data PROGBITS c0602000 60a000 036728 00 WA 0 0 64

[27] .bss NOBITS c0638740 640728 0703b4 00 WA 0 0 64

[28] .comment PROGBITS 00000000 640728 000011 01 MS 0 0 1

[29] .ARM.attributes ARM_ATTRIBUTES 00000000 640739 00002f 00 0 0 1

[30] .debug_line PROGBITS 00000000 640768 4cc19b 00 0 0 1

[31] .debug_info PROGBITS 00000000 b0c903 3016df2 00 0 0 1

[32] .debug_abbrev PROGBITS 00000000 3b236f5 19a5c2 00 0 0 1

[33] .debug_aranges PROGBITS 00000000 3cbdcb8 00bba8 00 0 0 8

[34] .debug_ranges PROGBITS 00000000 3cc9860 126b70 00 0 0 8

[35] .debug_frame PROGBITS 00000000 3df03d0 0a9e1c 00 0 0 4

[36] .debug_str PROGBITS 00000000 3e9a1ec 172ef0 01 MS 0 0 1

[37] .debug_loc PROGBITS 00000000 400d0dc 275d81 00 0 0 1

[38] .shstrtab STRTAB 00000000 4282e5d 0001bc 00 0 0 1

[39] .symtab SYMTAB 00000000 4283684 123350 10 40 56039 4

[40] .strtab STRTAB 00000000 43a69d4 0e6c49 00 0 0 1

walterzh$ /opt/armv7-marvell-linux-gnueabi-hard-4.6.4_x86_64_20140402/bin/arm-marvell-linux-gnueabi-nm vmlinux | grep __clk_of_table

c05fb770 T __clk_of_table

.....

__of_table_pegmatite_clkgate

.....

c05fbe54 t __clk_of_table_sentinel

```
void __init of_clk_init(const struct of_device_id *matches)
{
  。。。。。。

        /* First prepare the list of the clocks providers */
        for_each_matching_node_and_match(np, matches, &match) {
                struct clock_provider *parent =
                        kzalloc(sizeof(struct clock_provider),     GFP_KERNEL);

                parent->clk_init_cb = match->data;
                parent->np = np;
                list_add_tail(&parent->node, &clk_provider_list);
        }
```

上面code就是一个双循环

```
for (device_node = head of device_tree; device_node; device_node = device_node->next_node)
{
    for( struct of_device_id * device_id = __clk_of_table; device_id
< __clk_of_table_sentinel; device_id++)
    {
```

```
        __of_match_node(device_id, device_node);
    }
}
```

get the device_id and device_node (they are matched)

```
            struct clock_provider *parent =
                kzalloc(sizeof(struct clock_provider),    GFP_KERNEL);

            parent->clk_init_cb = match->data;
            parent->np = np;
            list_add_tail(&parent->node, &clk_provider_list);
```

上面的code生成如下node

```
struct clock_provider {
        of_clk_init_cb_t clk_init_cb;
        struct device_node *np;
        struct list_head node;
};
```

node->clk_init_cb = of_pegmatite_clkgate_setup;

node->np = device_node; 在device tree中与该clk driver match的device node。

并把该node加入到clk_provider_list这个global list上。

```
while (!list_empty(&clk_provider_list)) {
        is_init_done = false;
        list_for_each_entry_safe(clk_provider, next,
                                 &clk_provider_list, node) {
                if (force || parent_ready(clk_provider->np)) {

                        clk_provider->clk_init_cb(clk_provider->np);
                        of_clk_set_defaults(clk_provider->np, true);

                        list_del(&clk_provider->node);
                        kfree(clk_provider);
                        is_init_done = true;
                }
        }
```

枚举clk_provider_list这个global list上的各个node，依次调用clk driver的callback。

对某个clk进行初始化前，要确保其parent已经初始化。

```
xio_clkgate: xioclkgate {
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xf9080148 0 0x8>;
        clocks = <&xio_clk>;
        reset = <1>;
        always-used;
};
```

也就是在xio_clkgate被初始化以前，xio_clk必须先已经初始化。

```
static
const struct of_device_id *__of_match_node(const struct of_device_id *matches,
                                           const struct device_node *node)
{
        const struct of_device_id *best_match = NULL;
        int score, best_score = 0;

        if (!matches)
                return NULL;

        for (; matches->name[0] || matches->type[0] || matches->compatible[0]; matches++) {
                score = __of_device_is_compatible(node, matches->compatible,
                                                  matches->type, matches->name);
                if (score > best_score) {
                        best_match = matches;
                        best_score = score;
                }
        }

        return best_match;
}
```

对drivers/clk/pegmatite/clkgate.c clock driver 而言

matches->name[0] equal 0

matches->type[0] equal 0

matches->compatible[0] != 0 ("marvell,pegmatite-clkgate")

```
__of_device_is_compatible(device_node, "marvell,pegmatite-clkgate", NULL, NULL)
```

而该devcie_node为

```
scan_clkgate: scanclkgate {
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xf9080138 0 0x8>;
        clocks = <&scan_clk>;
        reset = <1>;
};

lvds_afe_clkgate: lvdsafeclkgate {
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xf9080130 0 0x8>;
        clocks = <&scan_pll_spread>;
        reset = <1>;
};

xcpu_clkgate: xcpuclkgate {
        compatible = "marvell,pegmatite-clkgate";
        #clock-cells = <0>;
        reg = <0 0xf9080140 0 0x8>;
        clocks = <&xcpu_clk>;
        reset = <1>;
        always-used;
```

};

etc.

即of_pegmatite_clkgate_setup（struct device_node *node）会依次收到各个device node.