Adding CPU Frequency Scaling for Your ARM Platform to Linux Kernel

Bartlomiej Zolnierkiewicz
b.zolnierkie@samsung.com
Samsung R&D Institute Poland
Overview and goals

CPU frequency scaling

Operating Performance Points (OPPs)

Generic DT based CPUfreq driver

OPPs for Exynos5422

CPU clock support for Exynos5422

making cpufreq-dt work on Odroid-XU3

ARM big LITTLE CPUfreq driver

Summary
Overview and goals

- Explain operation of generic cpufreq-dt driver
  - Based on v4.7 mainline kernel
- Present CPUfreq implementation for Odroid-XU3
  - Based on v4.7 & original next-20151216 patches
- cpufreq-dt vs ARM big LITTLE CPUfreq driver
  - Differences & similarities
CPU frequency scaling

- Frequency and voltage scaling possible through CPUfreq kernel subsystem
  - drivers/cpufreq/

- Generic governors for deciding transitions
  - performance – maximum frequency
  - powersave – minimum frequency
  - ondemand/conservative – heuristics based on load
  - userspace – leaves transition decisions to userspace

- Can be controlled from sysfs:
  - /sys/devices/system/cpu/cpu<n>/cpufreq
CPU specific code in separate drivers, i.e.:
  ◦ drivers/cpufreq/omap-cpufreq.c

Such driver must:
  ◦ implement operations of cpufreq_driver structure
    • init() – initialization
    • exit() – cleanup
    • verify() – verify the user-chosen policy
    • setpolicy() or target() – perform frequency change
  ◦ register it using cpufreq_register_driver()
Operating Performance Points (OPPs)

- Tuple of frequency & minimum voltage
- Described in Device Tree (DT) files
  - Documentation/devicetree/bindings/opp/opp.txt
- OPP v2 (by Viresh Kumar) merged in v.4.3
  - drivers/base/power/opp/

Examples:

```c
cpu@0 {
    compatible = "arm,cortex-a9";
    reg = <0>;
    next-level-cache = <&L2>;
    operating-points = <
        /* kHz   uV */
        792000 1100000
        396000  950000
        198000  850000
    >;
};
```
/ {
    cpus {
        #address-cells = <1>;
        #size-cells = <0>;

        cpu@0 {
            compatible = "arm,cortex-a9";
            reg = <0>;
            next-level-cache = <&L2>;
            clocks = <&clk_controller 0>;
            clock-names = "cpu";
            cpu-supply = <&cpu_supply0>;
            operating-points-v2 = <&cpu0_opp_table>;
        }
    }

    cpu@1 {
        compatible = "arm,cortex-a9";
        reg = <1>;
        next-level-cache = <&L2>;
        clocks = <&clk_controller 0>;
        clock-names = "cpu";
        cpu-supply = <&cpu_supply0>;
        operating-points-v2 = <&cpu0_opp_table>;
    }
};
cpu0_opp_table: opp_table0 {
    compatible = "operating-points-v2";
    opp-shared;

    opp@1000000000 {
        opp-hz = /bits/ 64 <1000000000>;
        opp-microvolt = <970000 975000 985000>;
        opp-microamp = <70000>;
        clock-latency-ns = <300000>;
        opp-suspend;
    };
    opp@1100000000 {
        opp-hz = /bits/ 64 <1100000000>;
        opp-microvolt = <980000 1000000 1010000>;
        opp-microamp = <80000>;
        clock-latency-ns = <310000>;
    };
    opp@1200000000 {
        opp-hz = /bits/ 64 <1200000000>;
        opp-microvolt = <1025000>;
        clock-latency-ns = <290000>;
        turbo-mode;
    };
};
Separate CPUfreq drivers for each ARM SoC

Moving toward one generic cpufreq-dt driver
- drivers/cpufreq/cpufreq-dt.c
  - formerly known as cpufreq-cpu0
  - introduced in v3.7 by Shawn Guo
  - heavily reworked by Viresh Kumar

Clock/regulator/OPP frameworks add necessary abstraction layer and make cpufreq-dt a generic solution
cpufreq-dt requires:

- Table of Operating Performance Points from Device Tree
  - OPP is a tuple of frequency and voltage
- Clock to operate on (provided by clock driver)
  - clk_get_rate()
  - clk_set_rate()
- Optionally voltage regulator (provided by regulator driver)
  - regulator_get_voltage()
  - regulator_set_voltage()
/*
 * Copyright (C) 2012 Freescale Semiconductor, Inc.
 * 
 * Copyright (C) 2014 Linaro.
 * Viresh Kumar <viresh.kumar@linaro.org>
 * 
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

#define pr_fmt(fmt) KBUILD_MODNAME "\": " fmt

#include <linux/clk.h>
#include <linux/cpumask.h>
#include <linux/of.h>
#include <linux/pm_opp.h>
#include <linux/platform_device.h>
#include <linux/regulator/consumer.h>
#include <linux/slab.h>
#include <linux/thermal.h>
Generic DT based CPUfreq driver

```c
struct private_data {
    struct device *cpu_dev;
    struct thermal_cooling_device *cdev;
    const char *reg_name;
};

static struct freq_attr *cpufreq_dt_attr[] = {
    &cpufreq_freq_attr_scaling_available_freqs,
    NULL, /* Extra space for boost-attr if required */
    NULL,
};

static int set_target(struct cpufreq_policy *policy, unsigned int index)
{
    struct private_data *priv = policy->driver_data;

    return dev_pm_opp_set_rate(priv->cpu_dev,
                                policy->freq_table[index].frequency * 1000);
}

/*
 * An earlier version of opp-v1 bindings used to name the regulator
 * "cpu0-supply", we still need to handle that for backwards compatibility.
 */
```
static const char *find_supply_name(struct device *dev) {
    struct device_node *np;
    struct property *pp;
    int cpu = dev->id;
    const char *name = NULL;

    np = of_node_get(dev->of_node);

    /* This must be valid for sure */
    if (WARN_ON(!np))
        return NULL;

    /* Try "cpu0" for older DTs */
    if (!cpu) {
        pp = of_find_property(np, "cpu0-supply", NULL);
        if (pp) {
            name = "cpu0";
            goto node_put;
        }
    }

    pp = of_find_property(np, "cpu-supply", NULL);
    if (pp) {
        name = "cpu";
        goto node_put;
    }
}

SAMSUNG
dev_dbg(dev, "no regulator for cpu%d\n", cpu);
node_put:
    of_node_put(np);
    return name;
}

static int resources_available(void)
{
    struct device *cpu_dev;
    struct regulator *cpu_reg;
    struct clk *cpu_clk;
    int ret = 0;
    const char *name;

    cpu_dev = get_cpu_device(0);
    if (!cpu_dev) {
        pr_err("failed to get cpu0 device\n");
        return -ENODEV;
    }

    cpu_clk = clk_get(cpu_dev, NULL);
    ret = PTR_ERR_OR_ZERO(cpu_clk);
    if (ret) {
        /*
         * If cpu's clk node is present, but clock is not yet
         * registered, we should try deferering probe.
         */
    }
    return ret;
}
dev_dbg(dev, "no regulator for cpu%d\n", cpu);
}

of_node_put(np);
return name;

static int resources_available(void)
{
    struct device *cpu_dev;
    struct regulator *cpu_reg;
    struct clk *cpu_clk;
    int ret = 0;
    const char *name;

    cpu_dev = get_cpu_device(0);
    if (!cpu_dev) {
        pr_err("failed to get cpu0 device\n");
        return -ENODEV;
    }

    cpu_clk = clk_get(cpu_dev, NULL);
    ret = PTR_ERR_OR_ZERO(cpu_clk);
    if (ret) {
        /*
         * If cpu's clk node is present, but clock is not yet
         * registered, we should try deffering probe.
         */
    }
}
if (ret == -EPROBE_DEFER)
    dev_dbg(cpu_dev, "clock not ready, retry\n");
else
    dev_err(cpu_dev, "failed to get clock: %d\n", ret);

return ret;

clk_put(cpu_clk);

name = find_supply_name(cpu_dev);
/* Platform doesn't require regulator */
if (!name)
    return 0;

cpu_reg = regulator_get_optional(cpu_dev, name);
ret = PTR_ERR_OR_ZERO(cpu_reg);
if (ret) {
    /*
     * If cpu's regulator supply node is present, but regulator is
     * not yet registered, we should try deferring probe.
     */
    if (ret == -EPROBE_DEFER)
        dev_dbg(cpu_dev, "cpu0 regulator not ready, retry\n");
    else
        dev_dbg(cpu_dev, "no regulator for cpu0: %d\n", ret);
static int cpufreq_init(struct cpufreq_policy *policy)
{
    struct cpufreq_frequency_table *freq_table;
    struct private_data *priv;
    struct device *cpu_dev;
    struct clk *cpu_clk;
    struct dev_pm_opp *suspend_opp;
    unsigned int transition_latency;
    bool fallback = false;
    const char *name;
    int ret;

    cpu_dev = get_cpu_device(policy->cpu);
    if (!cpu_dev) {
        pr_err("failed to get cpu%d device\n", policy->cpu);
        return -ENODEV;
    }

    ...
cpu_clk = clk_get(cpu_dev, NULL);
if (IS_ERR(cpu_clk)) {
    ret = PTR_ERR(cpu_clk);
    dev_err(cpu_dev, "%s: failed to get clk: %d\n", __func__, ret);
    return ret;
}

/* Get OPP-sharing information from "operating-points-v2" bindings */
ret = dev_pm_opp_of_get_sharing_cpus(cpu_dev, policy->cpus);
if (ret) {
    if (ret != -ENOENT)
        goto out_put_clk;

    /* operating-points-v2 not supported, fallback to old method of
    * finding shared-OPPs for backward compatibility if the
    * platform hasn't set sharing CPUs.
    */
    if (dev_pm_opp_get_sharing_cpus(cpu_dev, policy->cpus))
        fallback = true;
}

/*
 * OPP layer will be taking care of regulators now, but it needs to know
 * the name of the regulator first.
 */
name = find_supply_name(cpu_dev);
if (name) {
    ret = dev_pm_oppen_set_regulator(cpu_dev, name);
    if (ret) {
        dev_err(cpu_dev, "Failed to set regulator for cpu%d: %d\n",
                policy->cpu, ret);
        goto out_put_clk;
    }
}

/*
 * Initialize OPP tables for all policy->cpus. They will be shared by all CPUs which have marked their CPUs shared with OPP bindings.
 * For platforms not using operating-points-v2 bindings, we do this before updating policy->cpus. Otherwise, we will end up creating duplicate OPPs for policy->cpus.
 * OPPs might be populated at runtime, don't check for error here
 */
dev_pm_oppen_of_cpumask_add_table(policy->cpus);

/*
 * But we need OPP table to function so if it is not there let's give platform code chance to provide it for us.
 */
ret = dev_pm OPP_get OPP_count(cpu_dev);
if (ret <= 0) {
    dev_dbg(cpu_dev, "OPP table is not ready, deferring probe\n");
    ret = -EPROBE_DEFER;
    goto out_free_opp;
}

if (fallback) {
    cpumask_setall(policy->cpus);

    /*
     * OPP tables are initialized only for policy->cpu, do it for
     * others as well.
     */
    ret = dev_pm OPP_set OPP_sharing_cpus(cpu_dev, policy->cpus);
    if (ret)
        dev_err(cpu_dev, "%s: failed to mark OPPs as shared: %d\n", __func__, ret);
}

priv = kzalloc(sizeof(*priv), GFP_KERNEL);
if (!priv) {
    ret = -ENOMEM;
    goto out_free_opp;
}
priv->reg_name = name;

ret = dev_pm_op_init_cpu_freq_table(cpu_dev, &freq_table);
if (ret) {
    dev_err(cpu_dev, "failed to init cpu_freq table: %d\n", ret);
    goto out_free_priv;
}

priv->cpu_dev = cpu_dev;
policy->driver_data = priv;
policy->clk = cpu_clk;

rcu_read_lock();
suspend_opp = dev_pm_op_get_suspend_opp(cpu_dev);
if (suspend_opp) {
    policy->suspend_freq = dev_pm_op_get_freq(suspend_opp) / 1000;
}
rcu_read_unlock();

ret = cpu_freq_table_validate_and_show(policy, freq_table);
if (ret) {
    dev_err(cpu_dev, "%s: invalid frequency table: %d\n", __func__, ret);
    goto out_free_cpu_freq_table;
}

goto out_free_cpu_freq_table;
/* Support turbo/boost mode */
if (policy_has_boost_freq(policy)) {
    /* This gets disabled by core on driver unregister */
    ret = cpufreq_enable_boost_support();
    if (ret)
        goto out_free_cpufreq_table;
    cpufreq_dt_attr[1] = &cpufreq_freq_attr_scaling_boost_fregs;
}

transition_latency = dev_pm_opp_get_max_transition_latency(cpu_dev);
if (!transition_latency)
    transition_latency = CPUFREQ_ETERNAL;

policy->cpuinfo.transition_latency = transition_latency;
return 0;

out_free_cpufreq_table:
    dev_pm_opp_free_cpufreq_table(cpu_dev, &freq_table);
out_free_priv:
    kfree(priv);
out_free_opp:
    dev_pm_opp_of_cpumask_remove_table(policy->cpus);
if (name)
    dev_pm_opp_put_regulator(cpu_dev);
Generic DT based CPUfreq driver

```c
out_put_clk:
    clk_put(cpu_clk);
    return ret;
}

static int cpufreq_exit(struct cpufreq_policy *policy)
{
    struct private_data *priv = policy->driver_data;

    cpufreq_cooling_unregister(priv->cdev);
    dev_pm_opf_free_cpufreq_table(priv->cpu_dev, &policy->freq_table);
    dev_pm_opf_of_cpumask_remove_table(policy->related_cpus);
    if (priv->reg_name)
        dev_pm_opf_put_regulator(priv->cpu_dev);

    clk_put(policy->clk);
    kfree(priv);

    return 0;
}
```
static void cpufreq_ready(struct cpufreq_policy *policy)
{
    struct private_data *priv = policy->driver_data;
    struct device_node *np = of_node_get(priv->cpu_dev->of_node);

    if (WARN_ON(!np))
        return;

    /*
     * For now, just loading the cooling device;
     * thermal DT code takes care of matching them.
     */
    if (of_find_property(np, "#cooling-cells", NULL)) {
        u32 power_coefficient = 0;

        of_property_read_u32(np, "dynamic-power-coefficient",
                             &power_coefficient);

        priv->cdev = of_cpufreq_power_cooling_register(np,
                                                    policy->related_cpus, power_coefficient, NULL);
        if (IS_ERR(priv->cdev)) {
            dev_err(priv->cpu_dev,
                    "running cpufreq without cooling device: %ld\n",
                    PTR_ERR(priv->cdev));
            priv->cdev = NULL;
        }
    }
}
of_node_put(np);

static struct cpufreq_driver dt_cpufreq_driver = {
    .flags = CPUFREQ_STICKY | CPUFREQ_NEED_INITIAL_FREQ_CHECK,
    .verify = cpufreq_generic_frequency_table_verify,
    .target_index = set_target,
    .get = cpufreq_generic_get,
    .init = cpufreq_init,
    .exit = cpufreq_exit,
    .ready = cpufreq_ready,
    .name = "cpufreq-dt",
    .attr = cpufreq_dt_attr,
    .suspend = cpufreq_generic_suspend,
};

static int dt_cpufreq_probe(struct platform_device *pdev)
{
    int ret;

    /*
     * All per-cluster (CPUs sharing clock/voltages) initialization is done
     * from ->init(). In probe(), we just need to make sure that clk and
     * regulators are available. Else defer probe and retry.
     * 
     * FIXME: Is checking this only for CPU0 sufficient?
     */
Generic DT based CPUfreq driver

```c
ret = resources_available();
if (ret)
    return ret;

dt_cpufreq_driver.driver_data = dev_get_platdata(&pdev->dev);

ret = cpufreq_register_driver(&dt_cpufreq_driver);
if (ret)
    dev_err(&pdev->dev, "failed register driver: %d\n", ret);

return ret;
```

```c
static int dt_cpufreq_remove(struct platform_device *pdev)
{
    cpufreq_unregister_driver(&dt_cpufreq_driver);
    return 0;
}
```

```c
static struct platform_driver dt_cpufreq_platdrv = {
    .driver = {
        .name = "cpufreq-dt",
    },
    .probe = dt_cpufreq_probe,
    .remove = dt_cpufreq_remove,
};
```
Generic DT based CPUpfreq driver

```c
ret = resources_available();
if (ret)
    return ret;

dt_cpufreq_driver.driver_data = dev_get_platdata(&pdev->dev);

ret = cpufreq_register_driver(&dt_cpufreq_driver);
if (ret)
    dev_err(&pdev->dev, "failed register driver: %d\n", ret);

return ret;
}

static int dt_cpufreq_remove(struct platform_device *pdev)
{
    cpufreq_unregister_driver(&dt_cpufreq_driver);
    return 0;
}

static struct platform_driver dt_cpufreq_platdrv = {
    .driver = {
        .name = "cpufreq-dt",
    },
    .probe = dt_cpufreq_probe,
    .remove = dt_cpufreq_remove,
};
```
module_platform_driver(dt_cpufreq_platdrv);
MODULE_ALIAS("platform:cpufreq-dt");
MODULE_AUTHOR("Viresh Kumar <viresh.kumar@linaro.org>");
MODULE_AUTHOR("Shawn Guo <shawn.guo@linaro.org>");
MODULE_DESCRIPTION("Generic cpufreq driver");
MODULE_LICENSE("GPL");
OPPs for Exynos5422

- Exynos5422 is a 32-bit ARM Octa Core SoC
  - 4 x Cortex-A15 + 4 x Cortex-A7
  - Clocking:
    - 2.0GHz / 1.4GHz (Odroid-XU3 & Odroid-XU4)
    - 1.8GHz / 1.3GHz (Odroid-XU3 Lite)
Based on original patches from Thomas Abraham

Modified DT files:
- arch/arm/boot/dts/exynos5420.dtsi
- arch/arm/boot/dts/exynos5800.dtsi
- arch/arm/boot/dts/exynos5422-cpus.dtsi

Merged in v4.6 kernel
- 66a4a1fb2398
  - „ARM: dts: Add CPU OPP properties for exynos542x/5800”
- 4869710caedf
  - „ARM: dts: Extend existing CPU OPP for exynos5800”
diff --git a/arch/arm/boot/dts/exynos5420.dtsi b/arch/arm/boot/dts/exynos5420.dtsi
index 6c102c4..2a40554 100644
--- a/arch/arm/boot/dts/exynos5420.dtsi
+++ b/arch/arm/boot/dts/exynos5420.dtsi
@@ -50,6 +50,116 @@

                       usebdrdphy1 = &usbdrd_phy1;
                     }

+cluster_a15_opp_table: opp_table0 {
+    compatible = "operating-points-v2";
+    opp-shared;
+    opp@1800000000 {
+        opp-hz = /bits/ 64 <1800000000>; 
+        opp-microvolt = <1250000>; 
+        clock-latency-ns = <140000>; 
+    }
+}
+opp@1700000000 {
+    opp-hz = /bits/ 64 <1700000000>; 
+    opp-microvolt = <1212500>; 
+    clock-latency-ns = <140000>; 
+}
+opp@1600000000 {
+    opp-hz = /bits/ 64 <1600000000>; 
+    opp-microvolt = <1175000>; 
+    clock-latency-ns = <140000>; 
+}
+};
OPPs for Exynos5422

opp@15000000000 {
    opp-hz = /bits/ 64 <1500000000>;  
    opp-microvolt = <1137500>;  
    clock-latency-ns = <140000>;  
};

opp@14000000000 {
    opp-hz = /bits/ 64 <1400000000>;  
    opp-microvolt = <1112500>;  
    clock-latency-ns = <140000>;  
};

opp@13000000000 {
    opp-hz = /bits/ 64 <1300000000>;  
    opp-microvolt = <1062500>;  
    clock-latency-ns = <140000>;  
};

opp@12000000000 {
    opp-hz = /bits/ 64 <1200000000>;  
    opp-microvolt = <1037500>;  
    clock-latency-ns = <140000>;  
};

opp@11000000000 {
    opp-hz = /bits/ 64 <1100000000>;  
    opp-microvolt = <1012500>;  
    clock-latency-ns = <140000>;  
};
OPPs for Exynos5422

opp@1000000000 {
  opp-hz = /bits/ 64 <1000000000>;  
  opp-microvolt = < 987500>;  
  clock-latency-ns = <140000>;  
};

opp@900000000 {
  opp-hz = /bits/ 64 <900000000>;  
  opp-microvolt = < 962500>;  
  clock-latency-ns = <140000>;  
};

opp@800000000 {
  opp-hz = /bits/ 64 <800000000>;  
  opp-microvolt = < 937500>;  
  clock-latency-ns = <140000>;  
};

opp@700000000 {
  opp-hz = /bits/ 64 <700000000>;  
  opp-microvolt = < 912500>;  
  clock-latency-ns = <140000>;  
};
OPPs for Exynos5422

```c
+ cluster_a7_opp_table: opp_table1 {
+     compatible = "operating-points-v2";
+     opp-shared;
+     opp@1300000000 {
+         opp-hz = /bits/ 64 <1300000000>
+         opp-microvolt = <1275000>
+         clock-latency-ns = <140000>
+     }
+     opp@1200000000 {
+         opp-hz = /bits/ 64 <1200000000>
+         opp-microvolt = <1212500>
+         clock-latency-ns = <140000>
+     }
+     opp@1100000000 {
+         opp-hz = /bits/ 64 <1100000000>
+         opp-microvolt = <1162500>
+         clock-latency-ns = <140000>
+     }
+     opp@1000000000 {
+         opp-hz = /bits/ 64 <1000000000>
+         opp-microvolt = <1112500>
+         clock-latency-ns = <140000>
+     }
+ };
```
OPPs for Exynos5422

```c
+ opp@900000000 {  
+   opp-hz = /bits/ 64 <900000000>;  
+   opp-microvolt = <1062500>;  
+   clock-latency-ns = <140000>;  
+ + };  
+ opp@800000000 {  
+   opp-hz = /bits/ 64 <800000000>;  
+   opp-microvolt = <1025000>;  
+   clock-latency-ns = <140000>;  
+ + };  
+ opp@700000000 {  
+   opp-hz = /bits/ 64 <700000000>;  
+   opp-microvolt = <975000>;  
+   clock-latency-ns = <140000>;  
+ + };  
+ opp@600000000 {  
+   opp-hz = /bits/ 64 <600000000>;  
+   opp-microvolt = <937500>;  
+   clock-latency-ns = <140000>;  
+ + };  
+ +  
+ /*  
+ * The 'cpus' node is not present here but instead it is provided  
+ * by exynos5420–cpus.dtsi or exynos5422–cpus.dtsi.  
+ */
```
diff --git a/arch/arm/boot/dts/exynos5422-cpus.dtsi b/arch/arm/boot/dts/exynos5422-cpus.dtsi
index 33028ac..9b46b9f 100644
--- a/arch/arm/boot/dts/exynos5422-cpus.dtsi
+++ b/arch/arm/boot/dts/exynos5422-cpus.dtsi
@@ -28,8 +28,10 @@
    device_type = "cpu";
    compatible = "arm,cortex-a7";
    reg = <0x100>;
+    clocks = <&clock CLK_KFC_CLK>;
    clock-frequency = <1000000000>;
    cci-control-port = <&cci_control0>;
+    operating-points-v2 = <&cluster_a7_opp_table>;

    cpu1: cpu@101 {
@@ -38,6 +40,7 @@
            reg = <0x101>;
            clock-frequency = <1000000000>;
            cci-control-port = <&cci_control0>;
+            operating-points-v2 = <&cluster_a7_opp_table>;

    cpu2: cpu@102 {

SAMSUNG
OPPs for Exynos5422

```c
reg = <0x102>;  
clock-frequency = <1000000000>;  
ccci-control-port = <&cci_control0>;  
+ operating-points-v2 = <&cluster_a7_opp_table>;  
};

cpu3: cpu@103 {
reg = <0x103>;  
clock-frequency = <1000000000>;  
ccci-control-port = <&cci_control0>;  
+ operating-points-v2 = <&cluster_a7_opp_table>;  
};

cpu4: cpu@0 {
    device_type = "cpu";  
    compatible = "arm,cortex-a15";  
    + clocks = <&clock CLK_ARM_CLK>;  
    reg = <0x0>;  
    clock-frequency = <1800000000>;  
    ccci-control-port = <&cci_control1>;  
    + operating-points-v2 = <&cluster_a15_opp_table>;  
};

cpu5: cpu@1 {
```
OPPs for Exynos5422

```cpp
@@ -70,6 +77,7 @@
    reg = <0x1>; |
    clock-frequency = <1800000000>; |
    cci-control-port = <&cci_control1>; |
    + operating-points-v2 = <&cluster_a15_opp_table>;

    cpu6: cpu@2 {
@@ -78,6 +83,7 @@
    reg = <0x2>; |
    clock-frequency = <1800000000>; |
    cci-control-port = <&cci_control1>; |
    + operating-points-v2 = <&cluster_a15_opp_table>;

    cpu7: cpu@3 {
@@ -86,6 +95,7 @@
    reg = <0x3>; |
    clock-frequency = <1800000000>; |
    cci-control-port = <&cci_control1>; |
    + operating-points-v2 = <&cluster_a15_opp_table>;
```
OPPs for Exynos5422

diff --git a/arch/arm/boot/dts/exynos5800.dtsi b/arch/arm/boot/dts/exynos5800.dtsi
index c0bb356..8213016 100644
--- a/arch/arm/boot/dts/exynos5800.dtsi
+++ b/arch/arm/boot/dts/exynos5800.dtsi
@@ -23,6 +23,114 @@

compatible = "samsung,exynos5800-clock";
};

+&cluster_a15_opp_table {
+  opp@1700000000 {
+    opp-microvolt = <1250000>;
+  };
+  opp@1600000000 {
+    opp-microvolt = <1250000>;
+  };
+  opp@1500000000 {
+    opp-microvolt = <1100000>;
+  };
+  opp@1400000000 {
+    opp-microvolt = <1100000>;
+  };
+  opp@1300000000 {
+    opp-microvolt = <1100000>;
+  };
+};
OPPs for Exynos5422

```
+ opp@12000000000 {
+   opp-microvolt = <1000000>;
+ }
+ opp@11000000000 {
+   opp-microvolt = <1000000>;
+ }
+ opp@10000000000 {
+   opp-microvolt = <1000000>;
+ }
+ opp@9000000000 {
+   opp-microvolt = <900000>;
+ }
+ opp@8000000000 {
+   opp-microvolt = <900000>;
+ }
+ opp@7000000000 {
+   opp-microvolt = <900000>;
+ }
+ opp@6000000000 {
+   opp-hz = /bits/ 64 <6000000000>;
+   opp-microvolt = <900000>;
+   clock-latency-ns = <140000>;
+ }
```
OPPs for Exynos5422

```
+     opp@5000000000 {
+         opp-hz = /bits/ 64 <500000000>;  
+         opp-microvolt = <900000>; 
+         clock-lateness-ns = <140000>; 
+     
+     opp@4000000000 {
+         opp-hz = /bits/ 64 <400000000>;  
+         opp-microvolt = <900000>; 
+         clock-lateness-ns = <140000>; 
+     
+     opp@3000000000 {
+         opp-hz = /bits/ 64 <300000000>;  
+         opp-microvolt = <900000>; 
+         clock-lateness-ns = <140000>; 
+     
+     opp@2000000000 {
+         opp-hz = /bits/ 64 <200000000>;  
+         opp-microvolt = <900000>; 
+         clock-lateness-ns = <140000>; 
+     
+ };  
+ }; 
+   
```
OPPs for Exynos5422

```c
+  opp@500000000 { |
+    opp-hz = /bits/ 64 <500000000>;
+    opp-microvolt = <1000000>;
+    clock-latency-ns = <140000>;
+  };
+  opp@400000000 {
+    opp-hz = /bits/ 64 <400000000>;
+    opp-microvolt = <1000000>;
+    clock-latency-ns = <140000>;
+  };
+  opp@300000000 {
+    opp-hz = /bits/ 64 <300000000>;
+    opp-microvolt = <900000>;
+    clock-latency-ns = <140000>;
+  };
+  opp@200000000 {
+    opp-hz = /bits/ 64 <200000000>;
+    opp-microvolt = <900000>;
+    clock-latency-ns = <140000>;
+  };
+};
+&mfc {
    compatible = "samsung,mfc-v8";
};
```
CPU clock support for Exynos5422

- Based on original patches from Thomas Abraham
- Uses generic Exynos CPU clock support:
  - On Exynos SoCs:
    - some other clocks are derived from CPU clock
    - there is special CPU clock switching sequence
  - Part of Exynos clock driver
    - drivers/clk/samsung/clk-cpu.{c,h}
  - Utilizes clock framework pre/post rate notifiers
  - Merged in v4.2
    - ddeac8d968d4
      - „clk: samsung: add infrastructure to register cpu clocks”
static int exynos_cpuclk_notifier_cb(struct notifier_block *nb, unsigned long event, void *data) {
    struct clk_notifier_data *ndata = data;
    struct exynos_cpuclk *cpuclk;
    void __iomem *base;
    int err = 0;

    cpuclk = container_of(nb, struct exynos_cpuclk, clk_nb);
    base = cpuclk->ctrl_base;

    if (event == PRE_RATE_CHANGE)
        err = exynos_cpuclk_pre_rate_change(ndata, cpuclk, base);
    else if (event == POST_RATE_CHANGE)
        err = exynos_cpuclk_post_rate_change(ndata, cpuclk, base);

    return notifier_from_errno(err);
}

/*
 * This notifier function is called for the pre-rate and post-rate change
 * notifications of the parent clock of cpuclk.
 */
CPU clock support for Exynos5422

- Exynos542x specific part of Exynos clock driver:
  - drivers/clk/samsung/clk-exynos5420.c

- CPU clock support for Exynos542x merged in v4.6
  - bee4f87f01dc
    - „clk: samsung: exynos5420: add cpu clock configuration data and instantiate cpu clock”
  - 54abbdb4ee53
    - „clk: samsung: exynos542x/5800: fix cpu clock configuration data”
Other SoCs using custom CPU clocks implementations
  ◦ Rockchip RK3xxx SoCs (config ARCH_ROCKCHIP)
    • drivers/clk/rockchip/clk-cpu.c
  ◦ Amlogic Meson SoCs (config ARCH_MESON)
    • drivers/clk/meson/clk-cpu.c
Making cpufreq-dt work on Odroid-XU3

- Hardkernel Odroid-XU3 board
  - Exynos5422 based
  - „Off-the-shelf” hardware
  - Replaced by Odroid-XU4

- Hardkernel Odroid-XU4
  - More compact
  - Easily available
Optional regulator support
  ◦ SoC/board specific

Merged in v4.6
  ◦ 8b51c5e730fb
    • "ARM: dts: Add cluster regulator supply properties for exynos542x/5800"
Making cpufreq-dt work on Odroid-XU3

```diff
diff --git a/arch/arm/boot/dts/exynos5422-odroidxu3-common.dtsi  
b/arch/arm/boot/dts/exynos5422-odroidxu3-common.dtsi
index 9134217..1bd507b 100644
--- a/arch/arm/boot/dts/exynos5422-odroidxu3-common.dtsi
+++ b/arch/arm/boot/dts/exynos5422-odroidxu3-common.dtsi
@@ -67,6 +67,14 @@
                                 <19200000>;
                     +&cpu0 {
                      + cpu-supply = &buck6_reg;
+      +};
+      +&cpu4 {
+        + cpu-supply = &buck2_reg;
+      +};
+    &hdmi {
+      status = "okay";
+      hpd-gpio = &gpx3 7 GPIO_ACTIVE_HIGH;
```
Making cpufreq-dt work on Odroid-XU3

- Instantiate cpufreq-dt platform device
  - cpufreq-dt is a platform driver
- Handled from ARM platform code
- Merged in v4.6
  - 25ef3f52e59d
    - "ARM: EXYNOS: Use generic cpufreq driver for Exynos5422/5800,"
Making cpufreq-dt work on Odroid-XU3

```c
diff --git a/arch/arm/mach-exynos/exynos.c b/arch/arm/mach-exynos/exynos.c
index f1831f9..a439233 100644
--- a/arch/arm/mach-exynos/exynos.c
+++ b/arch/arm/mach-exynos/exynos.c
@@ -227,6 +227,7 @@ static const struct of_device_id exynos_cpufreq_matches[] = {
    { .compatible = "samsung,exynos5250", .data = "cpufreq-dt" },
#else CONFIG_BL_SWITCHER
    { .compatible = "samsung,exynos5420", .data = "cpufreq-dt" },
+   { .compatible = "samsung,exynos5800", .data = "cpufreq-dt" },
#endif
    { /* sentinel */ }
};
/* sentinel */
```
Nowadays handled by cpufreq-dt-platdev
  ◦ drivers/cpufreq/cpufreq-dt-platdev.c

Merged in v4.7
  ◦ f56aad1d98f1
    • "cpufreq: dt: Add generic platform-device creation support"
  ◦ 2249c00a0bf8
    • "cpufreq: exynos: Use generic platdev driver"
static int __init cpufreq_dt_platdev_init(void) {
    struct device_node *np = of_find_node_by_path("/");
    const struct of_device_id *match;

    if (!np)
        return -ENODEV;

    match = of_match_node(machines, np);
    of_node_put(np);
    if (!match)
        return -ENODEV;

    return PTR_ERR_OR_ZERO(platform_device_register_simple("cpufreq-dt", -1, NULL, 0));
}

device_initcall(cpufreq_dt_platdev_init);
static const struct of_device_id machines[] __initconst = {
    { .compatible = "allwinner,sun4i-a10", },
    { .compatible = "allwinner,sun5i-a10s", },
    { .compatible = "allwinner,sun5i-a13", },
    { .compatible = "allwinner,sun5i-r8", },
    { .compatible = "allwinner,sun6i-a31", },
    { .compatible = "allwinner,sun6i-a31s", },
    { .compatible = "allwinner,sun7i-a20", },
    { .compatible = "allwinner,sun8i-a23", },
    { .compatible = "allwinner,sun8i-a33", },
    { .compatible = "allwinner,sun8i-a83t", },
    { .compatible = "allwinner,sun8i-h3", },

    { .compatible = "hisilicon,hi6220", },

    { .compatible = "fsl,imx27", },
    { .compatible = "fsl,imx51", },
    { .compatible = "fsl,imx53", },
    { .compatible = "fsl,imx7d", },

    { .compatible = "marvell,berlin", },
};
Making cpufreq-dt work on Odroid-XU3

```c
#ifndef CONFIG_BL_SWITCHER

{ .compatible = "samsung,exynos5420", },
{ .compatible = "samsung,exynos5800", },
#endif

{ .compatible = "renesas,emev2", },
{ .compatible = "renesas,r7s72100", },
{ .compatible = "renesas,r8a73a4", },
{ .compatible = "renesas,r8a7740", },
{ .compatible = "renesas,r8a7778", },
{ .compatible = "renesas,r8a7779", },
{ .compatible = "renesas,r8a7790", },
{ .compatible = "renesas,r8a7791", },
{ .compatible = "renesas,r8a7793", },
{ .compatible = "renesas,r8a7794", },
{ .compatible = "renesas,sh73a0", },
```
Making cpufreq-dt work on Odroid-XU3

```c
{ .compatible = "rockchip,rk2928", },
{ .compatible = "rockchip,rk3036", },
{ .compatible = "rockchip,rk3066a", },
{ .compatible = "rockchip,rk3066b", },
{ .compatible = "rockchip,rk3188", },
{ .compatible = "rockchip,rk3228", },
{ .compatible = "rockchip,rk3288", },
{ .compatible = "rockchip,rk3366", },
{ .compatible = "rockchip,rk3368", },
{ .compatible = "rockchip,rk3399", },
{ .compatible = "sigma,tango4" },
{ .compatible = "ti,omap2", },
{ .compatible = "ti,omap3", },
{ .compatible = "ti,omap4", },
{ .compatible = "ti,omap5", },
{ .compatible = "xlnx,zynq-7000", },
};```
ARM big LITTLE CPUfreq driver

- Main driver code
  - drivers/cpufreq/arm_big_little.c

- „Glue” code for providing OPPs
  - drivers/cpufreq/arm_big_little_dt.c (DT)
  - drivers/cpufreq/scpi-cpufreq.c (SCPI)
  - drivers/cpufreq/vexpress-spc-cpufreq.c (SPC)
big LITTLE support using legacy big.LITTLE switcher
  ◦ Groups 1 big and 1 LITTLE CPU in „virtual” CPU

Provides „virtual” frequency for „virtual” CPU
  ◦ LITTLE CPU’s frequency is assumed to be half of the big one’s
  ◦ big CPU’s frequency is mapped 1:1

Needed till kernel scheduler has full HMP support

Limited usability in mainline kernel
  ◦ In v4.7 full support for ARM Versatile Express TC2
ARM big LITTLE CPUfreq driver

- Duplicates much of cpufreq-dt driver
  - cpufreq-dt also handles multiple CPU clusters now

- Stuff missing that is present in cpufreq-dt
  - Optional regulator support
  - Suspend OPP / frequency support
  - Software Boost functionality support

- Long-term status
  - Probably should be ported over cpufreq-dt
  - ..or just removed once scheduler handles HMP well
  - „Glue” code needs to be ported in either case
Adding CPUfreq support to new ARM platforms become relatively easy thanks to cpufreq-dt driver

- Assumption is that your platform uses CPU clock/regulator
  - IOW no special interface to request frequency/voltage change
- Some platforms may require additional CPU clock handling
  - Example implementations: Exynos, Rockchip, Meson
- Always use cpufreq-dt if possible (even for big.LITTLE CPUs)
- It is possible to easily add Turbo frequencies if desired
Thank you!
Any questions?
References

[1] "[PATCH V7 0/3] OPP: Introduce OPP (V2) bindings”, Viresh Kumar

[2] "[PATCH V3 00/16] OPP: Add code to support operating-points-v2 bindings”, Viresh Kumar

http://lists.infradead.org/pipermail/linux-arm-kernel/2012-September/117788.html


http://lkml.iu.edu/hypermail/linux/kernel/1504.0/02048.html


[7] „Introduction to Kernel Power Management”, Kevin Hilman
http://events.linuxfoundation.org/sites/events/files/slides/Intro_Kernel_PM.pdf
Boost mode (also known as Turbo mode)
- Use when CPU can run over its operating frequency
  - For a short duration of time
  - Limited by the CPU’s power, current & thermal limits
- Your platform should have
  - Thermal subsystem driver
  - „Generic CPU cooling support” enabled

„turbo-mode” DT property (requires OPPv2)
- To mark OPPs containing „turbo” frequencies
Reference implementation in upstream kernel
  ◦ Exynos 4412 based Hardkernel Odroid-U3 board
    • arch/arm/boot/dts/exynos4412.dtsi

```c
cpu0_opp_table: opp_table0 {
    compatible = "operating-points-v2";
    opp-shared;

    opp@200000000 {
        opp-hz = /bits/ 64 <200000000>;
        opp-microvolt = <900000>;
        clock-latency-ns = <200000>;
    }

    ...;

    opp@1500000000 {
        opp-hz = /bits/ 64 <1500000000>;
        opp-microvolt = <1350000>;
        clock-latency-ns = <200000>;
        turbo-mode;
    }
};
```
- Boost mode is disabled by default
- Check available frequencies using sysfs:
  - /sys/devices/system/cpu/cpu<n>/cpufreq/scaling_boost_frequencies
- Turn Boost mode on using sysfs:
  - echo 1 > /sys/devices/system/cpu/cpu<n>/cpufreq/boost