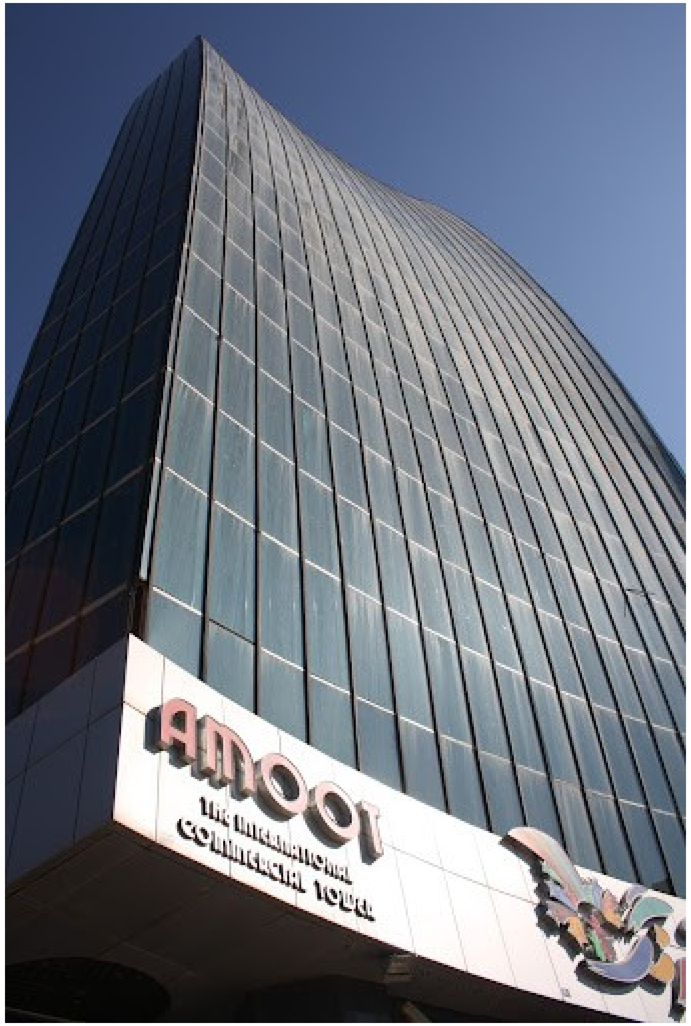
Power Saving on Embedded Linux

>whoami

SANA programmer

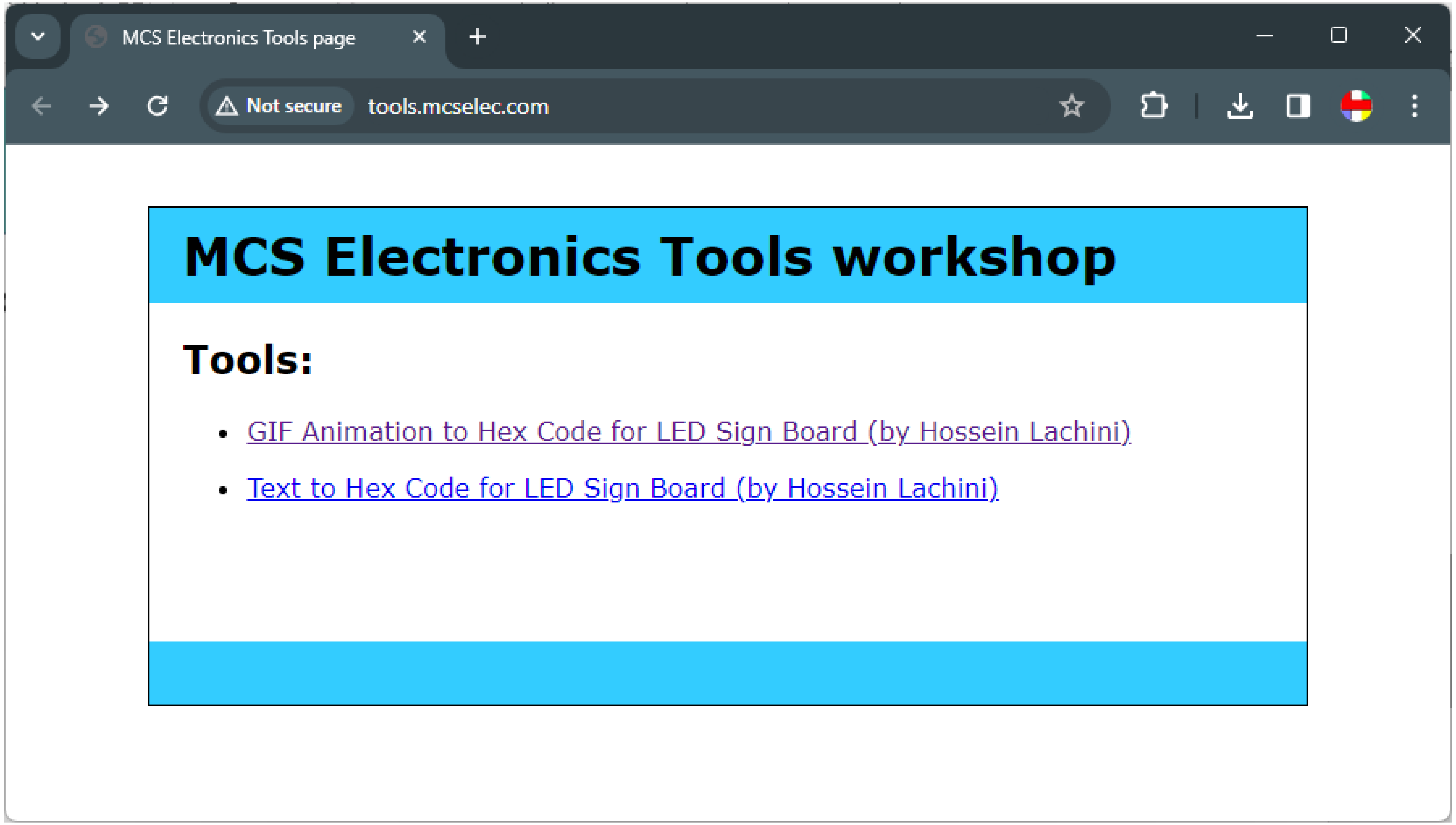
>whoami

Amoot tower (Karaj) lighting.



>whoami

AN #204 – GIF Animation to Hex code for LED Sign Board – MCS Electronic (BASCOM AVR)



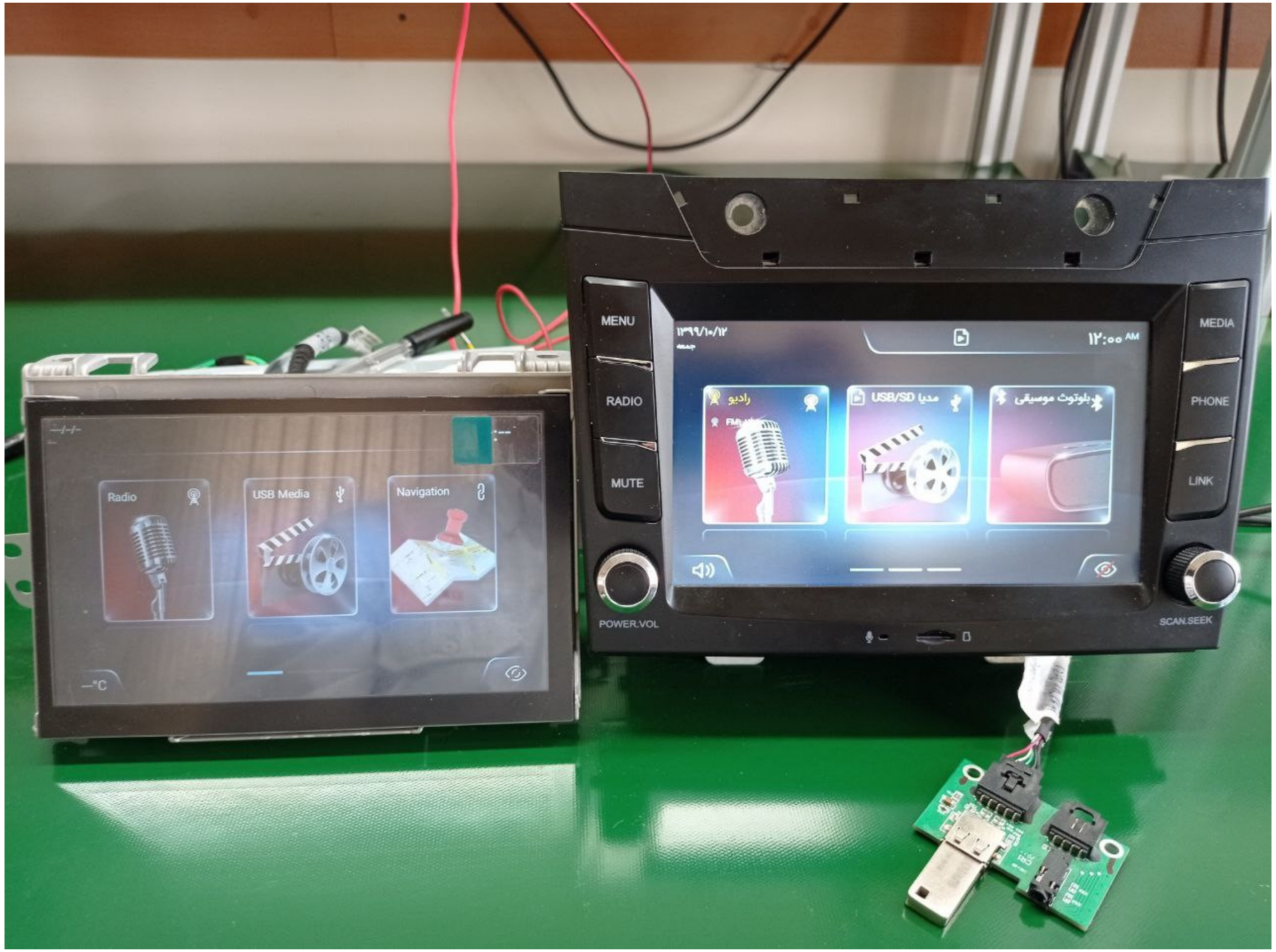
MCS Electronics Tools workshop

Tools:

- GIF Animation to Hex Code for LED Sign Board (by Hossein Lachini)
- Text to Hex Code for LED Sign Board (by Hossein Lachini)

>whoami

Manger of Electronic Product Engineering Department at RKA



>whoami

New Products Manger at ARG Telecom

>info ./Agenda

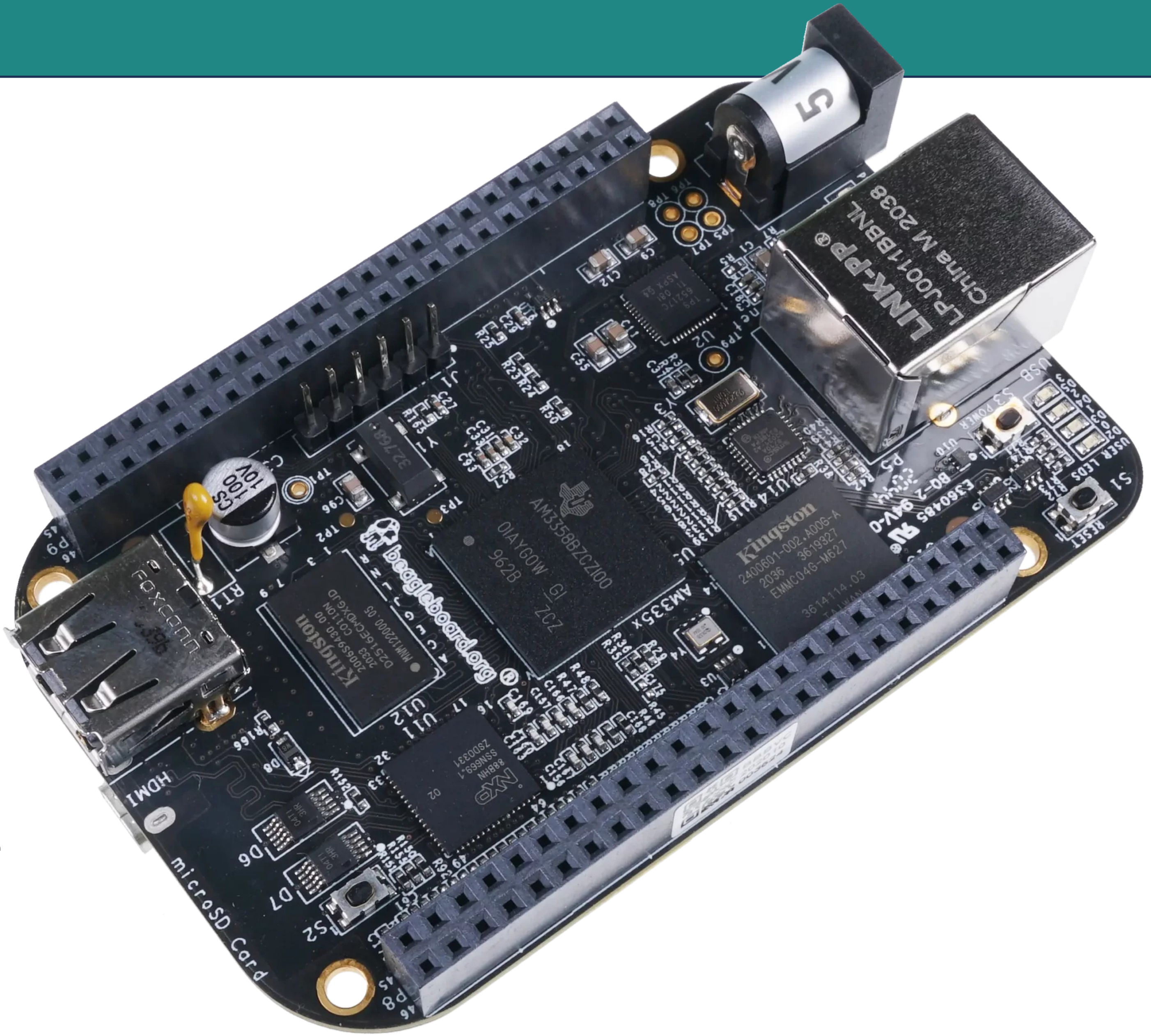
- Hardware for this course
- CPU
- Peripherals
- System Sleep
- References

>info ./Hardware

BeagleBone Black

```
# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

```
# cat /proc/version
Linux version 4.19.94-ti-r42 (voodoo@x3-am57xx-beagle-x15-2gb) (gcc version 8.3.0 (Debian 8.3.0-6)) #1buster SMP PREEMPT Tue Mar 31 19:38:29 UTC 2020
```



>info ./Hardware

USB to UART

>info ./Hardware

USB Power Meter

>info ./Energizer-Power-Max-P18K-Pop

>info ./Energizer-Power-Max-P18K-Pop

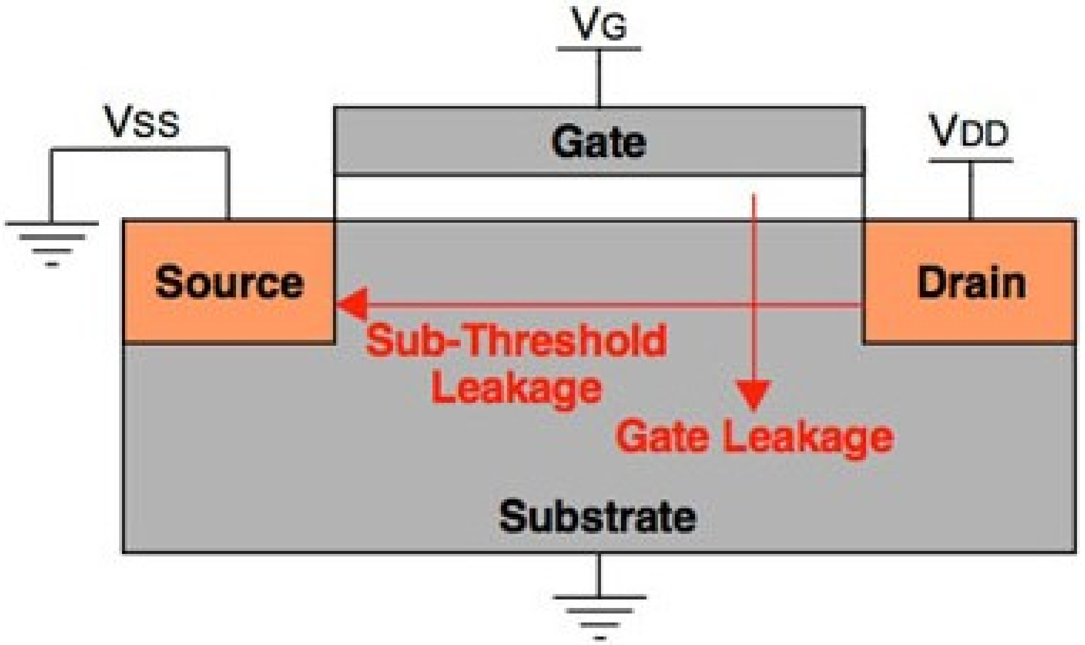
0.5Kg

>info ./power

$P_{cpu} = P_{static} + P_{dyn}$

Static Power (gate leakage) :

Static power is defined as the power consumed by the different devices on-chip when they are not switching.



Dynamic Power :

The dynamic power component is dependent on the total capacitance of the logic gates **being switched**, the clock frequency, and the square of the voltage:

$$P_{dyn} = CfV^2$$

From this, we can see that changing the frequency by itself is not going to save any power because the same number of CPU cycles have to be completed in order to execute a given subroutine. If we reduce the frequency by half, it will take twice as long to complete the calculation, but the total power consumed due to the dynamic power component will be the same.

Therefore, if we want to save power, we must be able to change the voltage that the CPU core operates at. But for any given voltage, there is a maximum frequency beyond which the switching of the gates becomes unreliable. Higher frequencies need higher voltages, and so the two need to be adjusted together.

Many SoCs implement such a feature: it is called **Dynamic Voltage and Frequency Scaling**, or **DVFS**. Manufacturers calculate optimum combinations of core frequency and voltage. Each combination is called **Operating Performance Point**, or **OPP**.

>info ./CPUFreq

Linux has a component named **CPUFreq** that manages the transitions between OPPs.CPUFreq consists of drivers in **drivers/cpufreq/**, which make the transition from one OPP to another, and a set of governors that implement the policy of when to switch. It is controlled per-CPU via the **/sys/devices/system/cpu/cpuN/cpufreq** directory, with **N** being the CPU number. In there, we find a number of files, the most interesting of which are as follows:

- **cpuinfo_cur_freq**, **cpuinfo_max_freq**, and **cpuinfo_min_freq**: The current frequency for this CPU, together with the maximum and minimum, measured in KHz.
- **cpuinfo_transition_latency**: The time, in nanoseconds, to switch from one OPP to another. If the value is unknown, it is set to -1.
- **scaling_available_frequencies**: A list of OPP frequencies available on this CPU.
- **scaling_available_governors**: A list of governors available on this CPU.
- **scaling_governor**: The CPUFreq governor currently being used.
- **scaling_cur_freq** : Current frequency of the CPU as determined by the governor and cpufreq core, in KHz. This is the frequency the kernel thinks the CPU runs at.
- **scaling_max_freq** and **scaling_min_freq**: The range of frequencies available to the governor in KHz.
- **scaling_setspeed**: A file that allows you to manually set the frequency when the governor is **userspace.**

>info ./CPUFreq

The governor sets the policy to change the OPP. It can set the frequency between the limits of **scaling_min_freq** and **scaling_max_freq**. The governors are named as follows:

- **powersave**: sets the CPU **statically to the lowest** frequency within the borders of scaling_min_freq and scaling_max_freq.
- **performance**: sets the CPU **statically to the highest** frequency within the borders of scaling_min_freq and scaling_max_freq.
- **ondemand**: Changes frequency based on the CPU utilization. If the CPU is idle less than 20% of the time, it sets the frequency to the maximum; if it is idle more than 30% of the time, it decrements the frequency by 5%.
- **conservative**: Like ondemand, but switches to higher frequencies in 5% steps rather than going immediately to the maximum.
- **schedutil** : much like the "**ondemand**" governor, It differs in behaviour in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU. This behaviour is more suitable in a battery powered environment.
- **userspace**: allows the user, or any userspace program running with UID "**root**", to set the CPU to a specific frequency by making a sysfs file "**scaling_setspeed**" available in the CPU-device directory.

>info ./CPUFreq

```
# echo userspace > /sys/bus/cpu/devices/cpu0/cpufreq/scaling_governor
# cat /sys/bus/cpu/devices/cpu0/cpufreq/scaling_available_frequencies
300000 600000 720000 800000 1000000
# cat /sys/bus/cpu/devices/cpu0/cpufreq/scaling_cur_freq
300000
# echo 720000 > /sys/bus/cpu/devices/cpu0/cpufreq/scaling_setspeed
# cat /sys/bus/cpu/devices/cpu0/cpufreq/scaling_cur_freq
720000
# cat /sys/bus/cpu/devices/cpu0/cpufreq/stats/time_in_state
300000 73960
600000 165
720000 11681
800000 0
1000000 137326
```

>info ./RPM

Runtime power management system : It is dynamic and should be trasparent to user space. Typically, it would include turning off the clock to the subsystem.
In **sysfs**, each device has a subdirectory named **power**, in which you will find these files:

- **control** : This allows user space to determine whether runtime pm is used on this device. If it is set to **auto**, then runtime pm is enabled, but by setting it to **on**, the device is always on and does not use runtime pm.
- **runtime_enabled** : This reports that runtime pm is **enabled**, **disabled**, or, if **control** is **on**, it reports **forbidden**.
- **runtime_status** : This reports the current state of the device. It may be **active**, **suspended**, or **unsupported**.
- **autosuspend_delay_ms** : This is the time before the device is suspended. **-1** means waiting forever. Some drivers implement this if there is a significant cost to suspending the device hardware since it prevents rapid suspend/resume cycles.

>info ./USB

>info ./USB

```
# cd /sys/bus/usb/devices/usb1/power/
# grep “” *
active_duration:108
async:enabled
autosuspend:0
autosuspend_delay_ms:0
connected_duration:2457916
control:auto
level:auto
runtime_active_kids:0
runtime_active_time:108
runtime_enabled:enabled
runtime_status:suspended
runtime_suspended_time:2457808
runtime_usage:0
wakeup:disabled
wakeup_abort_count:
wakeup_active:
wakeup_active_count:
wakeup_count:
wakeup_expire_count:
wakeup_last_time_ms:
wakeup_max_time_ms:
wakeup_prevent_sleep_time_ms:
wakeup_total_time_ms:
```

>info ./USB

```
# grep “” *
active_duration:4504
async:enabled
autosuspend:0
autosuspend_delay_ms:0
connected_duration:2583136
control:auto
level:auto
runtime_active_kids:1
runtime_active_time:4504
runtime_enabled:enabled
runtime_status:active
runtime_suspended_time:2578632
runtime_usage:0
wakeup:disabled
wakeup_abort_count:
wakeup_active:
wakeup_active_count:
wakeup_count:
wakeup_expire_count:
wakeup_last_time_ms:
wakeup_max_time_ms:
wakeup_prevent_sleep_time_ms:
wakeup_total_time_ms:
```

>cat /sys/power/state

The states are represented by strings that can be read or written to the **/sys/power/state** file. Those strings may be **mem**, **standby**, **freeze** and **disk**, where the last three always represent Power-On Suspend (if supported), Suspend-To-Idle and hibernation (Suspend-To-Disk), respectively.

The meaning of the **mem** string is controlled by the **/sys/power/mem_sleep** file. It contains strings representing the available modes of system suspend that may be triggered by writing **mem** to **/sys/power/state**. These modes are **s2idle** (Suspend-To-Idle), **shallow** (Power-On Suspend) and **deep** (Suspend-To-RAM).
The **s2idle** mode is always available, while the other ones are only available if supported by the platform (if not supported, the strings representing them are not present in **/sys/power/mem_sleep**).
The string representing the suspend mode to be used subsequently is enclosed in square brackets.
Writing one of the other strings present in **/sys/power/mem_sleep** to it causes the suspend mode to be used subsequently to change to the one represented by that string.

>info ./freeze

Stops all activity in user space and put all I/O Devices into low-power state, CPU and memory are operating as normal.

There are two ways to cause the system to go into the Suspend-To-Idle sleep state. The first one is to write **freeze** directly to **/sys/power/state**.

```
# echo freeze > /sys/power/state
```

The second one is to write **s2idle** to **/sys/power/mem_sleep** and then to write **mem** to **/sys/power/state**.

```
# echo s2idle > /sys/power/mem_sleep
# echo mem > /sys/power/state
```

>info ./standby

Just like **freeze**, but additionally takes all CPUs offline except the boot CPU.

Similarly, there are two ways to cause the system to go into the Power-On Suspend sleep state if that state is supported by the platform.
Write **standby** directly to **/sys/power/state**.

```
# echo standby > /sys/power/state
```

The second one is to write **shallow** to **/sys/power/mem_sleep** and then to write **mem** to **/sys/power/state**.

```
# echo shallow > /sys/power/mem_sleep
# echo mem > /sys/power/state
```

>info ./STR

Suspend-to-RAM : Powers down the system and puts the memory in the self-refresh mode.

In turn, there is only one way to cause the system to go into the Suspend-To-RAM state (write **deep** into **/sys/power/mem_sleep** and **mem** into **/sys/power/state**).

```
# echo deep > /sys/power/mem_sleep
# echo mem > /sys/power/state
```

>info ./STD

Hibernation (Suspend-to-disk) : This state operates similarly to Suspend-to-RAM, but includes a final step of writing memory contents to disk.

For suspend-to-disk, a mechanism called 'swsusp' (Swap Suspend) is used to write memory contents to free swap space. swsusp has some restrictive requirements, but should work in most cases. Some, albeit outdated, documentation can be found in Documentation/power/swsusp.txt.
Alternatively, userspace can do most of the actual suspend to disk work, see userland-swsusp.txt.

In turn, there is only one way to cause the system to go into the Suspend-To-disk state (write **disk** into **/sys/power/state**).

```
# echo disk > /sys/power/state
```

>echo standby > /sys/power/state

```
# echo shallow > /sys/power/mem_sleep
# echo mem > /sys/power/state
[  774.052077] PM: suspend entry (deep)
[  774.055790] PM: Syncing filesystems ... done.
[  774.077831] Freezing user space processes ... (elapsed 0.002 seconds) done.
[  774.087442] OOM killer disabled.
[  774.090690] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  774.099649] Suspending console(s) (use no_console_suspend to debug)
```

The device powers down! But how do I wake it up again?

>cat /sys/kernel/debug/wakeup_sources

Before you suspend a device, you must have a method of waking it again. To see wake up sources :

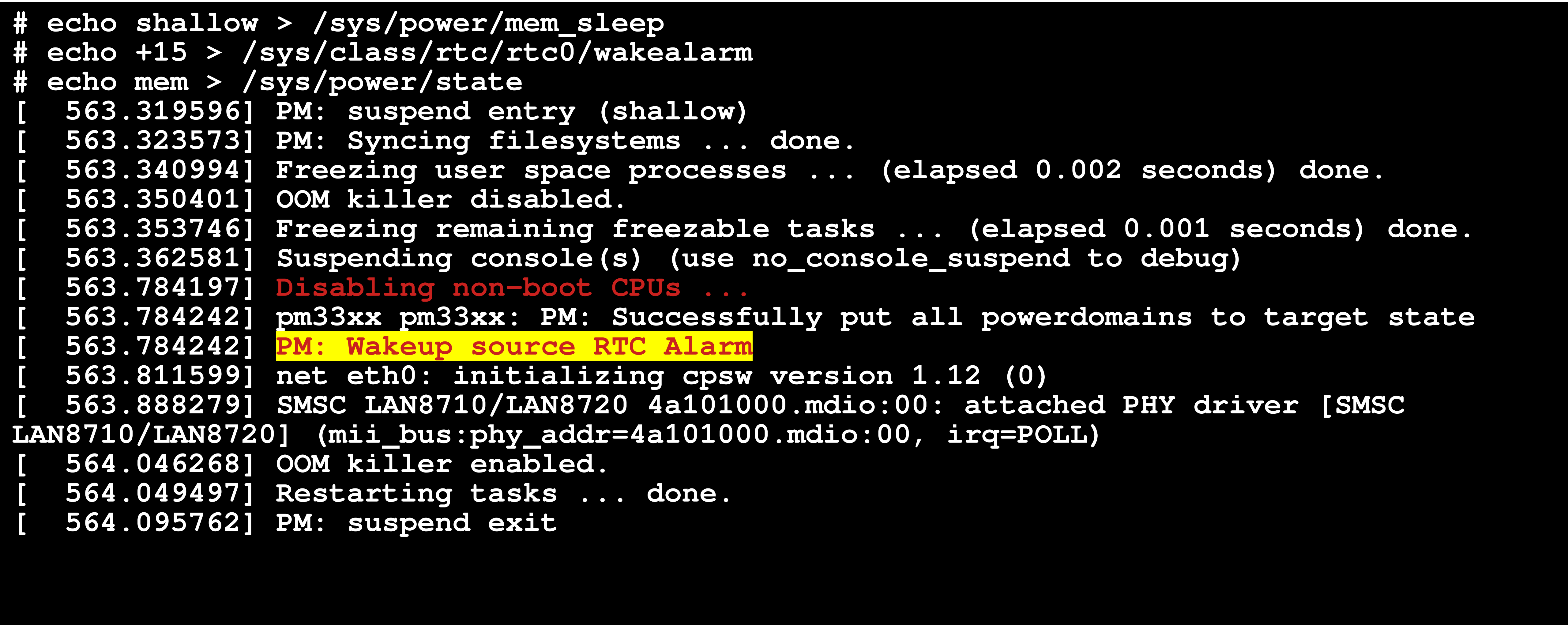
```
# cat /sys/kernel/debug/wakeup_sources
```

```
# cat /sys/kernel/debug/wakeup_sources
musb-hdrc.0
tps65217-pwrbutton
alarmtimer
44e3e000.rtc
musb-hdrc.1
481d8000.mmc
48060000.mmc
481aa000.serial
481a8000.serial
481a6000.serial
48024000.serial
48022000.serial
44e09000.serial
autosleep
```

>echo +15 > /sys/class/rtc/rtc0/wakealarm

To set RTC wake up alarm :

```
# echo +15 > /sys/class/rtc/rtc0/wakealarm
```



```
# echo shallow > /sys/power/mem_sleep
# echo +15 > /sys/class/rtc/rtc0/wakealarm
# echo mem > /sys/power/state
[  563.319596] PM: suspend entry (shallow)
[  563.323573] PM: Syncing filesystems ... done.
[  563.340994] Freezing user space processes ... (elapsed 0.002 seconds) done.
[  563.350401] OOM killer disabled.
[  563.353746] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  563.362581] Suspending console(s) (use no_console_suspend to debug)
[  563.784197] Disabling non-boot CPUs ...
[  563.784242] pm33xx pm33xx: PM: Successfully put all powerdomains to target state
[  563.784242] PM: Wakeup source RTC Alarm
[  563.811599] net eth0: initializing cpsw version 1.12 (0)
[  563.888279] SMSC LAN8710/LAN8720 4a101000.mdio:00: attached PHY driver [SMSC
LAN8710/LAN8720] (mii_bus:phy_addr=4a101000.mdio:00, irq=POLL)
[  564.046268] OOM killer enabled.
[  564.049497] Restarting tasks ... done.
[  564.095762] PM: suspend exit
```

>echo enabled > /sys/class/tty/ttyS0/power/wakeup

To set UART wake up alarm :

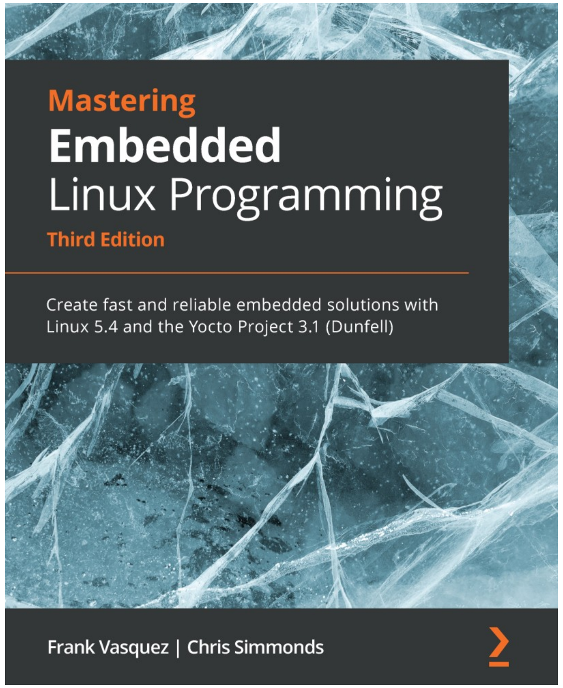
```
# echo enabled > /sys/class/tty/ttyS0/power/wakeup
```

```
# echo enabled > /sys/class/tty/ttyS0/power/wakeup
# echo mem > /sys/power/state
[  182.280068] PM: suspend entry (shallow)
[  182.284037] PM: Syncing filesystems ... done.
[  182.303021] Freezing user space processes ... (elapsed 0.002 seconds) done.
[  182.312950] OOM killer disabled.
[  182.316311] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  182.325314] Suspending console(s) (use no_console_suspend to debug)
[  183.409222] Disabling non-boot CPUs ...
[  183.409266] pm33xx pm33xx: PM: Successfully put all powerdomains to target state
[  183.409266] PM: Wakeup source UART
[  183.436515] net eth0: initializing cpsw version 1.12 (0)
[  183.512841] SMSC LAN8710/LAN8720 4a101000.mdio:00: attached PHY driver [SMSC
LAN8710/LAN8720] (mii_bus:phy_addr=4a101000.mdio:00, irq=POLL)
[  183.670304] OOM killer enabled.
[  183.673502] Restarting tasks ... done.
[  183.708462] PM: suspend exit
```

>info ./References

https://www.kernel.org/doc/Documentation/

https://www.youtube.com/watch?v=DJDrfsSgqO0



Mastering
Embedded
Linux Programming
Third Edition

Create fast and reliable embedded solutions with
Linux 5.4 and the Yocto Project 3.1 (Dunfell)

Frank Vasquez | Chris Simmonds

>poweroff

```
~# poweroff
```