1 Arbeitsumgebung

1.1 Remote Reset

Minicom mit Reset-feature befindet sich auf der `i30cons` und dort im Verzeichnis `/opt/remote_reset/bin`.
In dieser Version kann dann der reset mit `ctrl+a r` ausgelöst werden. Alternativ kann auf der `i30cons` mit `rr pm<no>` der Rechner zurückgesetzt werden.

1.2 Kernel Build für ATOM Rechner

Um einen eigenen Kernel zu booten wurde die Datei `~/boot/menu.lst` so angelegt, dass GRUB den eigenen Kernel via Netzwerk lädt.

1. Kernel von `/home/power/pub/linux-3.1.4.tar.bz2` kopieren und entpacken oder einen aktuellen von kernel.org auschecken.
2. Die bereitgestellte Kernelkonfigurationsdatei `/home/power/pub/Atom/ATOM_small_kernel_config` in das Kernelverzeichnis kopieren und in `.config` umbenennen.
3. `LINUX-SOURCEDIR/make menuconfig` aufrufen und wieder beenden. Dabei speichern, um die soeben kopierte Konfigurationsdatei zu verwenden.
4. `LINUX-SOURCEDIR/make ARCH=i386 -j2` ausführen
5. den Kernel aus `LINUX-SOURCEDIR/arch/x86/boot/bzImage` nach `~/boot/` kopieren
6. Den Testrechner neu booten. Nun sollte der Testrechner mit dem selbst erstellten Kernel booten können.

1.3 Kernel Build für Sandy Bridge Rechner

Der Kern für die Sandy Bridge Rechner wird auf dem Zielsystem selbst kompiliert und dann automatisch in desen GRUB-Menüeintag eingetragen.

1. Einen stable-Kernel von `/home/power/pub/linux-3.1.4.tar.bz2` auf das Zielsystem herunterladen und entpacken (oder einen Kernel von kernel.org herunterladen).
2. Die bereitgestellte Kernelkonfigurationsdatei `/home/power/pub/SB_small_kernel_config` in das Kernelverzeichnis kopieren und in `.config` umbenennen.

3. `LINUX-SOURCEDIR/make menuconfig` aufrufen und wieder beenden. Dabei speichern um die soeben kopierte Konfigurationsdatei zu verwenden.

4. `LINUX-SOURCEDIR/make deb-pkg -j4 LOCALVERSION=-pmx KDEB_PKGVERSION=1` ausführen.

5. Die Datei `LINUX-SOURCEDIR/../linux-image-3.1.4_3.1.4-10.00.Custom_amd64.deb` mit `dpkg -i linux-image-3.1.4_3.1.4-10.00.Custom_amd64.deb` installieren.

6. Den Testrechner neu booten. Nun sollte der Testrechner mit dem selbst erstellten Kernel booten können.

1.4 Kernel Build für Odroid XU3

Der Kern für Odroid XU3 wird auf dem Zielsystem selbst kompiliert. Nicht jeder Schritt ist bei kleinen Änderungen notwendig.

1. Repository von github klonen mit
   `git clone --depth 1 https://github.com/hardkernel/linux.git -b odroidxu3-3.10.y odroidxu3-3.10.y`

2. Wechseln Sie in den erzeugten Ordner `linux` und führen Sie alle folgenden Anweisungen in diesem Ordner durch.

3. Den Kernel konfigurieren mit `make odroidxu3_defconfig`

4. Optional: Die Konfiguariton mit `make menuconfig` anpassen.

5. Den eignetlichen Buildvorgang mit `make -j9` starten.

6. **Den erzeugten Kern installiert:**

   (a) Das zImage kopieren:
   `cp arch/arm/boot/zImage /media/boot`

   (b) Die DTB Datei kopieren:
   `cp arch/arm/boot/dts/exynos5422-odroidxu3.dtb /media/boot`

   (c) Module installieren:
   `sudo make modules_install`

7. **Initramfs und uInitrd erzeugen:** Hierfür finden sie auch ein Skript in `home/power/pub` auf i30s7. Ausführen mit `sudo sh ./odroid-ramdisk.sh`.

   (a) Kopieren der Konfiguartion:
   `sudo cp .config /boot/config-'make kernelrelease'`

(b) Erzeugen von initramfs:
`sudo update-initramfs -c -k 'make kernelrelease'`

(c) Erzeugen von uInitrd:
`sudo mkimage -A arm -O linux -T ramdisk -C none -a 0 -e 0 -n uInitrd -d /boot/initrd.img-‘make kernelrelease‘ /boot/uInitrd-‘make kernelrelease‘`

(d) Installieren mit
`sudo cp /boot/uInitrd-‘make kernelrelease‘ /media/boot/uInitrd`

8. Den Testrechner neu booten `sudo sync && reboot`. Nun sollte der Testrechner mit dem selbst erstellten Kernel booten können.

9. Bei einem nicht funktionierenden Kern, den Rechner über minicom resetten, und danach in den Bootloader wechseln. Dort lässt sich der original Kern mit `setenv bootcmd $altbootcmd` temporr wieder aktivieren und mit `boot` starten.

1.5 PM-LAB-Tools (erfoderlich für PM2 und PM3)

Das Messgerät liefert Messdaten mit Hilfe des Messservers (i30pm1) und der PM-LAB-Tools aus. Das Programm `pmlabclient pm1 12345 pmX` verbindet sich mit dem Messserver und gibt bis zu einem `SIGINT` Signal die Spannung der Platte auf die Konsole aus. Es steht ein Skript zur Verfügung, das die Daten direkt in einen Graph plottet `./plott-watts pmX`. Der Messserver ist des weiteren über eine C-API ansprechbar.

Installation und Test der PM-LAB-TOOLS:

1. `cp /home/power/pub/pm-lab-tools.tar.gz .` in `~/` kopieren.
2. `tar -xjvf pm-lab-tools.tar.gz`
3. `cd pm-lab-tools.git/client`
4. `./build.sh client`
5. `build/pmlabclient i30pm1 12345 pmX`

Programmierschnittstelle:

- libpmlab.h # Programmatic C-API
- libpmlab.c # Programmatic C-API
- Establish Connection
  `void *pm_connect(char *server, char *port, unsigned int *channels, unsigned int num_channels);`

- Read Data
  `int pm_read(void *handle, size_t buffer_sizes, double *analog_data, digival_t *unused, unsigned int *samples_read, uint64_t *timestamp_nanos);`
- Close Connection
  `void pm_close(void *handle);`

2 PM2 + PM3: Disk Power Management

2.1 Aufgabenstellung

Die Rechner `i30pm2-i30pm4` sind mit einer zusätzlichen Festplatte (`/dev/sda1`) und einer Solid State Disk (`/dev/sdb1`) ausgestattet.

Es soll ein stromsparendes Dateisystem entwickelt werden, das neben einer Festplatte als weiteres Speichermedium eine SDD verwendet. Als Dateisystem soll FAT32 (eine Lösung die auf ext3 basieren werden ebenfalls akzeptiert) eingesetzt werden (`fs/fat`). Während es ausreicht, Meta-Informationen (FAT-Tabelle ...) nur auf der SSD zu schreiben, sollen Dateiinhalte zusätzlich auf der Festplatte gesichert werden. Bei einem lesenden Zugriff soll zunächst überprüft werden, ob sich die Festplatte im Idle-Modus befindet und nur in diesem Fall der Zugriff durchgeführt werden. Falls sich die Platte im Standby-Modus befindet, soll der Zugriff auf das Flashlaufwerk umgeleitet werden. Um lesende und schreibende Zugriffe abzufangen, müssen die Funktionen `fat_readpages()` und `fat_writepages()` angepasst werden (in der Datei `inode.c`).

Die Funktionen `fat_readpages()` und `fat_writepages()` greifen lesend bzw. schreibend auf Seiten zu, die wiederum auf der darunterliegenden Ebene als Menge von "bufferheads" verwaltet werden (s. `fs/buffer.c`). Für ein spiegelndes Schreiben bzw. für die Umlenkung eines lesenden Zugriffs kann beispielsweise die Kennung des Blockgerätes in den "bufferheads" geändert werden.

Als mögliche Erweiterung kann ein spiegelndes Schreiben der Metadaten implementiert werden. Der Zugriff auf Metadaten wird im Fat-Dateisystem über die Buffercache-Ebene durchgeführt (durch Verwendung der Funktionen `sb_bread()` und `sb_getblk()`. Die modifizierten "bufferheads" müssen zusätzlich auf die Festplatte geschrieben werden.

2.2 Stromverbrauch der Festplatte messen

Die Festplatte `/dev/sdb` ist über einen Messwiderstand an den A/D-Wandler angeschlossen und liefert Messwerte an den Messserver (`i30pm1`).

2.3 Implementierung eines Timeout-Mechanismus

Als erster Schritt soll der "DDT-Algorithmus" (*device-dependent timeout*) für die Festplatte implementiert werden:

DDT-Algorithmus

Festplatte in den Standby-Modus setzen, falls

$$t_{la} + t_{be} \leq t$$

Die Variablen bedeuten:

t : die aktuelle Zeit

t_{la} : wann wurde zuletzt auf die Festplatte zugegriffen (last_access)

t_{be} : die break-even-Zeit

Die break-even-Zeit soll über das `/proc` Dateisystem konfigurierbar sein (implementiert im Blocklayer: `block/blk-core.c` und `block/genhd.c`).

Dazu soll der Kernel-Thread `hdd_power_task` geschrieben werden, der die Moduswechsel der Festplatte kontrolliert. Dieser Thread wird aufgeweckt (z.B. über einen Mutex), wenn die Festplatte aktiviert oder deaktiviert werden soll oder sich der Modus der Festplatte ändert (z.B. durch einen Reset). Die Struktur `struct scsi_device` (`include/scsi/scsi_device.h`) soll dazu um folgende Informationen erweitert werden:

```
unsigned long break_even_period;
unsigned long last_access;
int power_mode;     /* 0=standby, 1=running */
int new_power_mode; /* 0=standby, 1=running, 2=changed, 3=no_change */
```

Die Variable `power_mode` stellt den Zustand der Festplatte dar (es wird nur zwischen `standby` und `running` unterschieden); `new_power_mode` ist der neue Zustand bei einem Moduswechsel. Bei einem Zugriff auf die Festplatte soll die Variable `last_access` aktualisiert werden. Über einen Timer soll einmal die Sekunde die Funktion `check_for_standby()` aufgerufen werden, die die aktuelle Zeit mit (`last_access + break_even_period`) vergleicht und bei einer Überschreitung die Festplatte in den Standby-Modus versetzt (`new_power_mode` auf 0 setzen, `hdd_power_task` aufwecken). In manchen Situationen ist der Zustand der Festplatte unbekannt. Dann soll der `hdd_power_ task` mit dem Modus `changed` aufgeweckt werden und selbst herausfinden, in welchem Modus sich die Festplatte befindet.

Seiten, die den Inhalt von Disk-Blöcken enthalten und modifiziert wurden (als *dirty* markiert), werden vom *pdflush*-Daemon auf die Platte zurückgeschrieben (implementiert in in `mm/pdflush.c` und `mm/page-writeback.c`). Der Daemon überprüft alle 5 Sekunden (*dirty writeback centisecs*), ob modifizierte Seiten vorliegen und schreibt diese auf Platte, sobald sie mindestens 30 Sekunden alt sind (*dirty expire centisecs*). Damit soll vermieden werden, dass bei kurz aufeinanderfolgenden Schreibzugriffen jedesmal auf die Platte zugegriffen wird. Die Zeiten können unter `/proc/sys/vm/dirty_expire_centisecs` bzw. `dirty_writeback_centisecs` geändert werden.

Dieser Mechanismus verfügt über einen *laptop mode*, welcher einen energiegewahren Update-Prozess implementiert. Dieser *laptop mode* soll so ergänzt werden, dass vor einem Herunterfahren der Platte sämtliche als *dirty* markierte Blöcke auf die Platte geschrieben werden. Damit soll ein Wiederanfahren der Platte durch den Update-Prozess vermieden werden.

Bei regelmäßigen Festplattenzugriffen mit ungefähr gleicher Größe bzw. Dauer (z.B. Audioplayer) kann der Algorithmus noch verbessert werden, indem nach einem Datentransfer sofort in den Standby-Modus gewechselt wird (und nicht gewartet wird, bis die break-even-Zeit abgelaufen ist). Der Algorithmus soll dann zum DDT/ES-Algorithmus (device-dependent timeout with early shutdown) erweitert werden:

DDT/ES-Algorithmus

Festplatte in den Standby-Modus setzen, falls

$$t_{la} + t_{be} \leq t$$

oder

$$t_{fa} + t_{lb} \leq t \leq t_{fa} + t_{lb} + t_1 \quad \text{und} \quad t_{la} + t_2 \leq t$$

Die Variablen bedeuten:

- t : die aktuelle Zeit
- t_{lb} : die Länge der letzten aktiven Phase (Dauer eines I/O-Transfers, `busy_period`)
- t_{fa} : der Zeitpunkt des ersten Zugriffs in der gerade aktiven Phase (`first_access`)
- t_{la} : wann wurde zuletzt auf die Festplatte zugegriffen
- t_{be} : die break-even-Zeit
- t_1 : Toleranzwert beim Vergleich des letzten mit dem aktuellen Interval (2 Sekunden)
- t_2 : kleiner Timeout, um das Ende der aktiven Phase zu erkennen (1 Sekunde)

Als *busy period* wird die Dauer zusammengehörender Festplattenzugriffe verstanden; sie beginnt und endet mit einem Festplattenzugriff. Dazu muss in der Struktur `gen_disk_t` (`include/linux/ide.h`) noch um folgende Variablen ergänzt werden:

```
unsigned long busy_period;
unsigned long first_access;
```

3 PM5-PM7: Energieabschätzung und Drosselung

3.1 Aufgabenstellung

Mit Hilfe von Ereigniszählern lässt sich die Leistungsaufnahme eines Prozessors abschätzen. In dieser Aufgabe sollen einzelne Tasks durch die von ihnen verursachte Leistungsaufnahme charakterisiert werden. Dazu muss zunächst ein Mechanismus zur Energieabschätzung implementiert werden. Die Abschätzung geschieht durch Zählen von Ereignissen und durch Gewichten der Zählerwerte mit Energiegewichten.

Durch einen Systemaufruf soll ein Limit (z.B.hJoule/10ms) für einen Prozess gesetzt werden können. Wir dieses Limit überschritten, wir der Prozess bis zum Ende des Intervalls blockiert.

Die Ereigniszähler des Intel Sandy Bridge Prozessors sind in der Lage, eine Vielzahl an verschiedenen Ereignissen zu zählen. Unter `/home/power/pub/SandyBridge/init_perfcounters.sh` liegt ein Script, welches die Zählregister so konfiguriert, dass sie die in der Tabelle aufgeführten Ereignisse zählen.

Ereignis	**Zählregister**	**Gewicht [nJ]**
INST_RETIRED	`IA32_FIXED_CTR0`	-0,224
CPU_CLK_UNHALTED	`IA32_FIXED_CTR1`	4,546
DSB2MITE_SWITCHES	`IA32_PMC0`	-5,972
DSB_FILL:ALL_CANCEL	`IA32_PMC1`	64,240
ILD_STALL:IQ_FULL	`IA32_PMC2`	-1,425
L2_RQSTS:PF_HIT	`IA32_PMC3`	-22,837
LD_BLOCKS:ALL_BLOCK	`IA32_PMC4`	5,504
LD_BLOCKS:DATA_UNKNOWN	`IA32_PMC5`	-6,281
UOPS_DISPATCHED:STALL_CYCLES	`IA32_PMC6`	-2,508
BR_INST_RETIRED:FAR_BRANCH	`IA32_PMC7`	-18031,295

Die Modell spezifischen Register des *Run Time Average Power Limiting Interface* (RAPL) sollen genutzt werden, um die verbrauchte Energie abzuschätzen. Hierfür sei die Lektüre von Intel 64 and IA-32 Architectures Software Developer's Guide Volume 3 Chapter 14.7 empfohlen.

Zum Charakterisieren einzelner Tasks müssen am Ende jeder Zeitscheibe und bei jedem Taskwechsel (siehe `kernel/sched/core.c`) die Ereigniszähler ausgelesen und der Energieverbrauch ermittelt werden. Dividiert man den Energieverbrauch durch die seit dem letzten Messen verstrichene Zeit, erhält man die durch den Task verursachte Leistungsaufnahme. Zum Speichern der Leistungsaufnahme soll die Datenstruktur `task_struct` (`include/linux/sched.h`) um zwei Felder `power` und `power_msr` erweitert werden. Diese Felder und der Gesamtenergieverbrauch der CPUs `cpu<n>_power` und `cpu<n>_power_msr` sollen vom Userspace aus über das proc Dateisystem auslesbar sein.

Hinweise:

- Im Linux-Kern können keine Gleitkommaoperationen ausgeführt werden. Alle Berechnungen müssen daher als Ganzzahloperationen erfolgen.
- Die Ereigniszähler umfassen 48 Bit. Unter `/home/power/pub/sb_perfcounter/rdmsrll_wrmsrll.h` finden sich Makros zum Auslesen/Schreiben dieser Register in/aus eine(r) 64-Bit Variable(n).

[1] Frank Bellosa, Andreas Weißel, Martin Waitz and Simon Kellner: Event-Driven Energy Accounting for Dynamic Thermal Management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, September 2003.
`http://www4.informatik.uni-erlangen.de/Publications/pdf/Bellosa-Weissel-Kellner-Waitz-COLP03-Thermal_Management.pdf`
[2] Johannes Weiß, Frank Bellosa: Multi-Core Energy Accounting *Studienarbeit*, Oktober 2011.
`http://tux4u.de/study-thesis-weiss.pdf`
[3] Intel 64 and IA-32 Architectures Software Developer's Guide Volume 3, September 2013.
`http://www3.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`

4 PM8-PM10: ARM Exynos5422 - Energiemessung und Management

4.1 Aufgabenstellung

Der Exynos5422 verfügt über 8 Prozessorkerne, diese unterscheiden sich in ihrem Aufbau, 4 Kerne sind als Cortex-A7(little) und 4 als Cortex-A15(big) ausgeführt. Sie unterscheiden sich daher in ihrer Leistungsaufnahme und ihrer maximalen CPU Frequenz. Die Leistungsaufnahme lässt sich mit den auf dem Board verbauten Messeinheiten auslesen. Die technischen Informationen zu den 4 verbauten Messchips finden Sie in `/home/power/pub/Exynos`. Weitere Informationen zum Odroid XU3 stehen im Forum und Wiki des Herstellers (`www.hardkernel.org`) zur Verfügung. Leider existiert momentan keine frei verfügbare Dokumentation der Samsung Exynos 5422 CPU, es steht nur die allgemeine Referenz von ARM zur Verfügung.

Ihre Aufgabe besteht darin, die Energieaufnahme der einzelnen Kerne (big und little, 1-4 Kerne) zu ermitteln, beobachten Sie bei ihren Experimenten zusätzlich den Energieverbrauch des Arbeitsspeichers (Gibt es nennenswerte Änderungen?). Um das genaue Verhalten der Messchips zu ermitteln, experimentieren Sie auch mit verschiedenen zeitlichen Auflösungen. Bestimmen Sie den maximalen und minimalen Energieverbrauch, um eine Fehlkonfiguration des Drosslungsmechanismus zu verhindern. Entwerfen Sie hierfür einfache Microbenchmarks, um die Energieaufnahme einzelner Klassen von Instruktionen (z.B. add, mov, branch, mul, ...) Energiewerte zuordnen zu können. Als eine mögliche Anwendung soll eine Energieabschätzung wie auf x86 Prozessoren entstehen. Versuchen Sie ihre Ergebnisse, wenn möglich mit den Performance Countern der CPUs zu validieren.

Der Exynos5422 findet in leicht veränderter Form auch Einsatz in Mobilgeräten, z.B. in modernen Smartphones, deshalb ist es wichtig die maximale Stromaufnahme und damit den entnommenen Batteriestrom zu limitieren. Implementieren Sie hierfür einen Mechanismus, der es ermöglicht, die maximale Leistungsaufnahme dynamisch einzustellen (z.B. über das sysfs) und diese durch geschicktes scheduling (z.B. möglichst nur die Cortex-A7 Kerne verwenden) diese Grenze nicht zu erreichen, dabei aber weiterhin notwendige Anwendungen auszuführen. Untersuchen Sie dabei auch die vom Linuxkern bereits verwendeten Stromsparfunktionen und setzen Sie diese, falls erforderlich ein (CPU frequency governor).