Coral

Power Savings Application Note

Author: michaelbrooks@

PRELIMINARY DRAFT (July 15, 2020)

Overview

The Coral Dev Board was designed for performance, it runs the system to get the maximum performance for showcasing the TPU. However, when the SoM is used in production designs this may not always be the correct assumption. The SoM doesn't include a fan; many designs will be built around passive cooling and battery power. This document discusses power saving techniques that can be used to reduce the power of the SoM in an application-specific manner. While most of these recommendations are intended for SoM customers, they are tested on the Dev Board and can easily be evaluated on that platform.

iMX8M Power Reduction

When not actually running ML workloads, the iMX8M dominates the total power consumption of the system. Some techniques are already in place for reducing the power consumption of the iMX8M including:

- DVFS (Dynamic Voltage and Frequency Scaling) for the CPU, GPU, and VPU
- cpufreq frequency management (default governor is ondemand)
- Power gating of GPU/VPU when unused
- cpuidle+hotplug (allowing individual CPU cores to be turned off)

That being said there are still significant power draws that remain on even in idle including:

- Memory / Bus Frequency always runs at max speed
- PCIe (including the PHYs)
- Unused components remain on (for example USB, ethernet)

- Leakage from improperly configured pins

Disabling Unused Peripherals

The first step that should be taken for power savings is disabling peripherals that aren't needed in the device tree. The device tree has been split into two files, one for the SoM and one for the baseboard. This is deliberate, the purpose is that when designing your own baseboard you would import the SoM one and create your own baseboard device tree. You can disable a specific resource by referencing the node and setting the status to "disabled". For example, to disable Ethernet:

```
&fec1 {
        status = "disabled";
}
```

Estimated Power Savings: Varies, depending on which components are disabled. Can be significant for large blocks (display, ethernet, USB, etc).

Reducing Memory / Bus Frequency

Busfreq is used to dynamically reduce the supply clock for the system bus as well as the DDR controller. On the Dev Board, this is explicitly disabled to ensure that the memory always runs at full speed. Busfreq works by allowing kernel drivers to request the bus speed they need in order to operate. In order to enable this functionality, a few changes are needed:

- Enabling busfreq, by setting the disabled reference in the device tree to "okay".
- Compiling the kernel with the config CONFIG_IMX8MQ_PHANBELL_POWERSAVE enabled in the defconfig (in the tree this is located at packages/linux-imx/debian/defconfig)

Why is the defconfig change needed? If you look for references to this config (CONFIG_IMX8MQ_PHANBELL_POWERSAVE) in the source tree you'll see that certain modules (ethernet, USB, SDIO) will hold BUS_FREQ_HIGH when the module is probed (for example, ethernet) and this won't be released until the module suspends. This is required to enable full performance; the ethernet port won't be able to run at gigabit speeds if the bus frequency is only 200 MHz. For the CONFIG_IMX8MQ_PHANBELL_POWERSAVE we are willing to sacrifice peak performance of these modules in order to allow the bus and DDR to underclock. For your application, please verify the peripherals affected do not need to hold a permanent high clock request.

Note: Busfreq only works for the 1G boards. If you have a 2/4G SoM or Dev Board the memory timings do not operate at lower frequencies. This is a known issue.

Estimated Power Savings: TBD (~0.5W)

Entering Suspend

If the system is designed in a way where the entire system can suspend (for example a periodic wake, external wake source, or M4 wake) significant savings can be had by putting the system into suspend to RAM mode. This will turn off all the CPU cores, suspend all the peripherals, and copy the current state to RAM. You'll need to configure a proper wakeup source in order to bring the system back to life.

For example, we can use the RTC to tell the system to wake in 20 seconds and then put the system into suspend to RAM mode:

```
mendel@silly-zebra:~$ echo +20 | sudo tee /sys/class/rtc/rtc0/wakealarm
+20
mendel@silly-zebra:~$ echo mem | sudo tee /sys/power/state
mem
```

After executing that command, the system enters suspend for 20 seconds.

Estimated Power Savings: TBD (Significant)

GPIO Audit

The IOs used in the system may not apply for a custom baseboard. Many of these IOs will have pullups enabled, and because every pin has a weak pull down - there is power consumption. The pinmux settings are defined in the baseboard device tree. The device tree configuration can be seen in the kernel documentation, most notably it is important to verify the usage of ODE (open drain enable), defined as the 5th bit:

```
PAD_CTL_ODE             (1 << 5)
```

This can be done with the iMX pinmux tool, where the exact pin mux configuration can be determined (note the SOC is MIMX8MQ6DVAJZ).

Estimated Power Savings: TBD (Minimal)

TPU Power Reduction

The TPU itself is intended to consume about 0.5W per TOP, but this number is an average. The TPU workload, for most models, consumes significant current quickly and then returns to idle (as most of the computation is extremely fast). The large spike requires special power considerations for things like power supply design and battery operation. But for the purpose of this document, we will focus on the idle; the high power spike while running a model is expected and acceptable. For power savings we want to ensure that we return to the best possible idle.

Apex Power

We've added a character device, apex_power, that allows complete control of the power to the TPU. This must be done as a separate device (than apex_0, for example) for the most power savings; by toggling this device we can turn off the upstream PCIe bus, PHYs, and the power supply to the TPU itself.

This driver is not included in the latest Mendel release as of this publishing (Eagle), but it is on the master branch of linux-imx. In order to enable this driver, it must be explicitly enabled:

- Compile the kernel with the config CONFIG_GOOGLE_APEX_POWER enabled in the defconfig (in the tree this is located at packages/linux-imx/debian/defconfig)
- Enable apex_power, by setting the disabled reference in the device tree to "okay".

With apex_power enabled, on kernel boot the TPU will come up and then immediately be turned off. The upstream PCIe bus as well as the power supply for the TPU will also be removed. If you look in /dev, the TPU itself (apex_0) is missing:

```
mendel@silly-zebra:~$ ls /dev | grep apex
apex_power
```

In order to turn the TPU on, this file descriptor must be open. This is likely done in your application code, for example opening and closing the file around when you need the TPU. While it is theoretically possible to toggle this on every inference, it's not a good idea - you'll be constantly remunerating PCIe devices. Rather the application would be more if only periodic TPU usage is needed (such as an external trigger) or if the input data isn't always streaming. A simple example of this would be opening the file as such:

```
mendel@silly-zebra:~$ ls /dev | grep apex
apex_power

mendel@silly-zebra:~$ tail -f /dev/null | sudo tee /dev/apex_power &
[1] 4078

mendel@silly-zebra:~$ ls /dev | grep apex
apex_0
apex_power

mendel@silly-zebra:~$ sudo kill 4078
mendel@silly-zebra:~$ ls /dev | grep apex
apex_power
```

Note that the TPU module only exists when the apex_power device is open. The kernel log reflects this:

```
[  894.512277] ddrc freq set to high bus mode
[  894.512392] pci 0001:00:00.0: [16c3:abcd] type 01 class 0x060400
...
[  894.594314] pcieport 0001:00:00.0: AER enabled with IRQ 368
[  894.600780] apex 0001:01:00.0: enabling device (0000 -> 0002)
[  894.607100] Can't support > 32 bit dma.
[  894.611137] Can't support > 32 bit dma.

[  919.915961] pci_bus 0001:01: busn_res: [bus 01] is released
[  920.939714] ddrc freq set to audio bus mode
```

Estimated Power Savings: Substantial (~0.5W)

Apex Module Parameters

The TPU module (apex) does have three properties for power savings:

```
mendel@silly-zebra:~$ cd /sys/module/apex/parameters
mendel@silly-zebra:/sys/module/apex/parameters$ ls
allow_hw_clock_gating  allow_sw_clock_gating  hw_temp_warn1     hw_temp_warn2     temp_poll_interval
trip_point1_temp
allow_power_save       bypass_top_level       hw_temp_warn1_en  hw_temp_warn2_en  trip_point0_temp
trip_point2_temp

mendel@silly-zebra:/sys/module/apex/parameters$ find allow* -type f -print -exec cat {} \;
allow_hw_clock_gating
1
allow_power_save
1
allow_sw_clock_gating
0
```

For additional power savings, you can enable allow_sw_clock_gating. This is set at module load time, so it should be set in the kernel command line by adding the line:

```
apex.allow_sw_clock_gating=1
```

Most likely this will be accomplished by adding to the command line provided in UBoot. For Mendel, this is best set either in the boot.txt (located in the tree at packages/uboot-imx/debian/boot.txt) or by setting the extra_bootargs environmental variable in UBoot and saving:

```
u-boot=> setenv extra_bootargs apex.allow_sw_clock_gating=1
u-boot=> saveenv
```

You can verify this has been applied once the kernel boots:

```
mendel@silly-zebra:~$ cat /proc/cmdline
console=ttymxc0,115200 console=tty0 earlycon=ec_imx6q,0x30860000,115200
root=PARTUUID=70672ec3-5eee-49ff-b3b1-eb1fbd406bf5 rootfstype=ext4 rw rootwait init=/sbin/init net.ifnames=0
pci=pcie_bus_perf apex.allow_sw_clock_gating=1

mendel@silly-zebra:~$ cat /sys/module/apex/parameters/allow_sw_clock_gating
1
```

Estimated Power Savings: TBD (Minimal)

Battery Power Considerations

With all the above power savings it is possible to run the system from a battery but some considerations are needed.

First, many of the outputs of the PMIC for the iMX and the TPU (as well as all the IOs) run at 3.3V. This means that at low battery voltages (for example 3-3.3V) the device may not work as expected - these rails will only be able to reach the input voltage. While there is little capacity in the lower voltage, if a battery is being used it is recommended to consider the battery empty earlier than a gas gauge may suggest.

The other major concern is the high current spike when running an inference. This may cause the battery voltage to droop significantly - possibly triggering low voltage protection or inaccurate gauging.

Document revisions

Table 1. History of changes to this document

Version	Changes
DRAFT (July 2020)	Initial release