Application Note

Saving Power on the E20 by Disabling Peripherals and Reducing Running Processes

In some cases, such as powering from batteries, it may be necessary for an E20 to run in a reduced-power mode. This application note describes ways to disable unneeded peripherals, shorten boot time, and eliminate unnecessary background processes to reduce the E20's power consumption.

10 May 2016

Christopher Dail
Software Engineer

116-091504-002-A000

Disclaimers

Synapse Wireless, Inc.
6723 Odyssey Drive
Huntsville, Alabama 35806
256-852-7888
877-982-7888
256-924-7398 (fax)

www.synapse-wireless.com

License governing any code samples presented in this application note

NOTE: This equipment is meant to be installed in accordance with all applicable local, state, and national electrical codes. If the equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired. Read these instructions, keep these instructions, and heed all warnings.

Introduction

The purpose of this application note is to provide specific instructions on how to reduce the E20's power consumption by individually disabling:

- the on-board USB-A port
- the internal DE-910 cell modem
- the internal WF111 Wi-Fi module
- the built-in Ethernet interface
- the internal SNAP module

We will also explore reducing the E20's boot time by altering the U-Boot boot sequence, and disabling unnecessary processes in Linux.

Power Savings Calculations and Baseline

All testing and analysis were performed using a DC bench power supply set to 12V, powering a factory defaulted E20 directly via the DC input, using the micro-SD card image e20-1.0.5-0-gab32e2d-2015-08-20-sdcard.img (image number #114-001901-000-A003). Similar results can most likely be expected on other image versions.

The following changes were made:

- The MAC address of eth0 was set to the valid address specified on the E20's sticker (not the 00:1c:2c:ff:ff:ff default.)

From an otherwise fresh boot with all default settings and the following configuration:

- Ethernet connected
- No USB devices plugged in
- One DE-910 cell modem (including antenna) plugged into the cell modem adapter
- Antenna connected to the SNAP module
- Antenna connected to the Wi-Fi module (which by default is disabled)
- All configurable LEDs off

In this configuration, during the boot sequence the E20 draws a peak current of 360mA at 12V, and settles at approximately 180mA when it reaches the login prompt.

Note that between each test, this is the configuration to which we return. All software is returned to its default state between tests — if Ethernet was disabled for one test, it should be assumed to be turned back on at the end of that test.

Ethernet Power Savings

If you have no need for Ethernet, you can perform the following steps to disable the Ethernet port:

Unplug the Ethernet:

With the Ethernet unplugged, current consumption at 12V in our test configuration falls from 180mA to around 150mA (while idle, at the boot prompt, a 30mA savings.)

Disable eth0 Entirely:

You can prevent the device from ever being “brought up” by Linux by editing the file /etc/network/interfaces, and changing the line:

```
allow-hotplug eth0
```

to:

```
#allow-hotplug eth0
```

In our testing, this resulted in a further reduction to 130mA, for a total savings of 50mA.

Wi-Fi Power Savings

If you have no need for the Wi-Fi, simply leave the Wi-Fi disabled. By default, the Wi-Fi module is not configured to activate at boot time.

With the Wi-Fi enabled, connected to an access point and idle, the E20 draws approximately 40mA of additional current. While transmitting, we see an additional draw of 30-40mA, on average.

Cellular Power Savings

If you have a cellular modem in your E20, there are ways you can temporarily disable the cellular modem to conserve power.

Note that to perform these steps, you will need to have installed the e20 gpio scripts. They can be installed at the E20 command prompt by typing:

```
$ sudo apt-get install e20-gpio-scripts>=1.0.3
```

To disable the cell modem temporarily, you can call:

```
$ enable-cell-modem off
```

In our testing, with the modem and E20 both idle at the time it was disabled, this reduced current consumption at 12V from approximately 180mA to 150mA. This setting will persist until you reboot or until you call:

```
$ enable-cell-modem on
```

This results in a measured savings of 30mA when disabling the cell modem.

SNAP Module Power Savings

The most effective way to reduce the SNAP module's power consumption is to use the sleep functionality combined with the E20's built in ability to toggle a wakeup pin.

First, on the SNAP node, you will need a script capable of putting the node to sleep and waking it up:

```
from synapse.platforms import *
from synapse.pinWakeup import *

@setHook(HOOK_STARTUP)
def main():
    setPinDir(GPIO_F1, False)
    setPinPullup(GPIO_F1, True)
    wakeupOn(GPIO_F1, True, False)
    sleep(0, sleep_time)
```

In our test, as always, with all other things idle and in their default state with 12V input, current is lowered from approximately 180mA to approximately 160mA, a savings of 20mA.

Then, on the E20, when you desire to wake the SNAP node, call:

```
sudo wake-snap-node
```

If absolutely necessary, one can hold the SNAP node in reset by calling:

```
sudo echo 0 > /sys/class/gpio/gpio34/value
```

This holds the reset line low, and should reduce current to 150mA until you reboot the E20, or you run the command:

```
sudo echo 1 > /sys/class/gpio/gpio34/value
```

This provides a savings of up to 30mA, but reboots the SNAP node when you enable it.

USB-A Power Savings

If nothing is plugged into the USB-A port, no current is drawn. There is no need to do anything with the USB-A port to reduce power consumption. It is not recommended to power-cycle the USB-A port if you plan to have attached devices. Some USB device drivers do not gracefully recover from power cycles while running.

LED Power Savings

All user-configurable LEDs are turned off by default in the stock E20 firmware image, so any activity will be defined by the user application. However, if your application uses the LEDs, it may be valuable to know that each LED will draw approximately 15mA of current for red or green, and approximately 30mA of current in “amber” (red + green) mode.

Reducing Boot Time

There are a number of things one can do to reduce boot time, which is a relatively CPU-intensive period that translates to a high current load.

1. Networking:
    a. Ensure any network devices not in use are disabled in the file `/etc/network/interfaces`.
    b. If you need a particular interface, use a static IP if possible. DHCP requires a negotiation phase, while static IP addresses should be immediately available. However, be aware that this can cause problems if your network settings change.
2. Reduce the U-Boot boot time:
    a. By default, U-Boot waits for five seconds for user input before continuing to load and run the Ubuntu operating system. This can be overridden with the following sets of commands.

       Note that by doing this, you may not be able to get to the U-Boot command prompt without following the steps to restore the boot delay, which are also detailed below.

   i. **If you have no micro SD card in your E20**, just run the following command:

```
sudo fw_setenv bootdelay 1
```

   Replace 1 with the number of seconds you want U-Boot to wait before booting.

   ii. **If you have a micro SD card in the E20**, edit the file `/etc/fw_env.config` first, and change:

```
# Dev         #offset  env size  #sector size
/dev/mmcblk0 0x60000  0x2000    0x200
/dev/mmcblk0 0x62000  0x2000    0x200
```

   To read:

```
# Dev         #offset  env size  #sector size
/dev/mmcblk1 0x60000  0x2000    0x200
/dev/mmcblk1 0x62000  0x2000    0x200
```

   Then run the `fw_setenv` command above.

b. To restore the original boot delay, run:

```
sudo fw_setenv bootdelay 5
```

3. Additionally, prevent U-Boot from scanning for USB devices.
   a. Run the following command to disable scanning:

```
sudo fw_setenv bootmmc 'run loaduimage; run loadfdt;
run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
```

   b. To restore the normal boot configuration:

```
sudo fw_setenv bootmmc 'usb start; run loaduimage; run
loadfdt; run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
```

4. Reduce Linux boot time by disabling processes:
   a. The E20 ships with, for the most part, a typical ARM Ubuntu 14.04 installation. The modifications made to it have been relatively minimal, so that the applications you develop and install can be Ubuntu-specific, rather than E20-specific. As such, there are a number of processes and components that are <u>not strictly necessary</u> for the E20's operation.

You can disable a number of these components (and speed the boot process) with the following sets of commands. The specifics of each component are beyond the scope of this application note, however ample documentation is available online about Ubuntu and each process and service.

Most notably, the command will remove the SSH services, disable the DHCP client, along with no longer setting the system clock from the hardware clock. If you need SSH, remove `ssh.conf` and `ssh` from the command below. If you will be using DHCP, remove `udhcpd` and `S20udhcpd`. If you require hardware clock support, remove `hwlclock.conf`, `hwclock-save.conf`, and `S10hwclock`.

```
$ # setup some directories to back up what we
disable..

$ mkdir /home/snap/disabled_init/

$ mkdir /home/snap/disabled_init.d/

$ mkdir /home/snap/disabled_rc2.d/

$ # now prevent some services from coming up on boot

$ cd /etc/init/
$ sudo mv -t /home/snap/disabled_init/ console-
font.conf cron.conf dmesg.conf plymouth-upstart-
bridge.conf plymouth.conf plymouth-shutdown.conf
plymouth-log.conf plymouth-splash.conf plymouth-
stop.conf rsyslog.conf ssh.conf tty1.conf tty2.conf
tty3.conf tty4.conf tty5.conf tty6.conf hwclock.conf
hwclock-save.conf flush-early-job-log.conf
setvtrgb.conf

$ cd /etc/init.d/

$ sudo mv -t /home/snap/disabled_init.d/ udhcpd
rsyslog

$ cd /etc/rc2.d/
$ sudo mv -t /home/snap/disabled_rc2.d/ S10hwclock
S20udhcpd
```

In these commands, we disable a number of default Ubuntu services:

- console-font.conf – console setup, which has no effect on the terminal sessions used to access the E20. Disabling this does not save you much, but costs you nothing.
- cron.conf – will prevent the start of the crond tasks scheduling deamon.
- dmesg.conf – the logging of driver and display messages to /var/log/dmesg; can be disabled if you do not need access to these logs.
- plymouth-*.conf – a series of customizable startup and shutdown utilities, however by default does nothing.
- tty*.conf – establishes a getty service on the virtual terminals tty[1-7]. It is unlikely you will need these in a low-power setup.
- hwclock.conf / hwclock_save.conf – reading / saving the time from / to the hwclock. If you are not concerned with the time in your low power setup, this is safe to disable.
- flush-early-job-log.conf – flushes cached log output to the disk; if you have disabled logging, you can disable this as well.
- setvtrgb.conf – sets the virtual terminal color codings; this can be safely disabled.
- udhcpd – the DHCP client deamon; only disable if you will be using a static IP address.
- rsyslog – starts the syslog deamon of messages to /var/log/syslog. This can be disabled if you do not need access to these logs.

In testing, this reduced the average time from power-on to the “login” prompt being displayed from around 26 seconds to around 15 seconds.

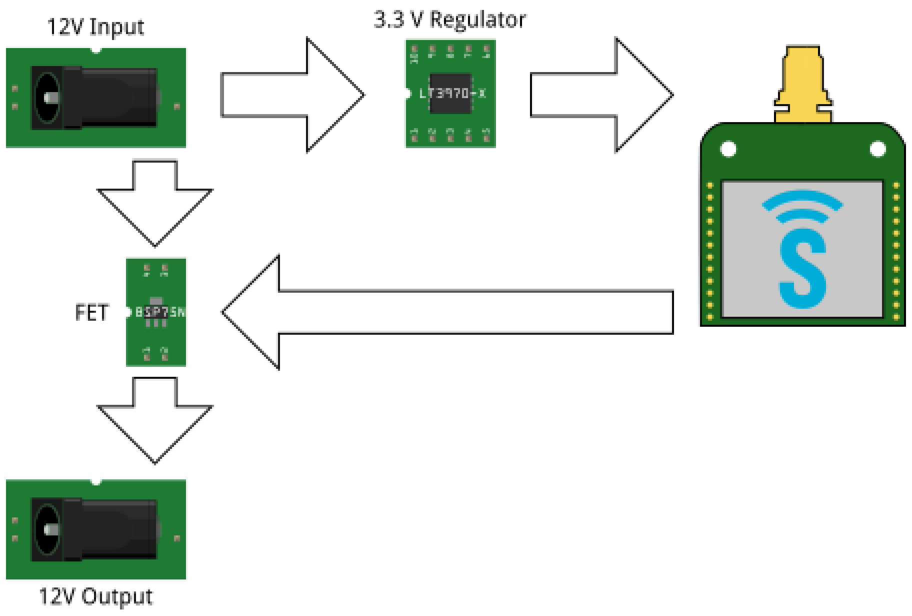
An External Approach

Adjusting your E20 configuration to minimize CPU usage and decommission attached hardware can save you about 45% on idle power consumption, but it leaves you with a gateway that has no means of communicating with its environment.

You will achieve the greatest power savings by turning off the E20 and then using an external trigger to restart it on schedule. You can accomplish this using an additional SNAP module and a field-effect transistor, controlling the power available to the E20 and powering it down between uses.

Consider the following diagram, blocked out with Fritzing[1] parts:

[1] http://fritzing.org



Note: Refer to the data sheets for the appropriate components, as PCB layouts for power supplies can be sensitive to trace lengths and component positioning.

The 12-volt incoming power is stepped down to 3.3 volts for the SNAP Engine by the LT3970 regulator. GPIO_0 from the SNAP Engine provides the input signal to the BSP75N FET that interrupts the ground line to the E20. When the SNAP Engine on the SN171 drives GPIO_0 high, the E20 receives power, boots, and functions as normal. When the SNAP Engine pulls GPIO_0 low, the power to the E20 is cut off.

You can then send the SNAP Engine itself to sleep, dropping your complete power consumption (for the SNAP Engine, the power regulator, and the FET) to less than 10 µA, *four orders of magnitude* lower than the 100 mA draw for an idle E20.

The recommended process for this is to have the SNAP node in the E20 send a message over the air to the external SNAP Engine starting a countdown to shutdown. The E20 would then perform a controlled shutdown process, and after the appropriate amount of time (30 seconds will typically be sufficient), the external SNAP Engine will shut off power to the E20 and go to sleep for the requested time. (Alternately you could have the SNAP Engine remain awake and responsive to radio signals elsewhere in your network, at a current cost on the order of 22 mA.)

When the SNAP Engine wakes on schedule (minutes or hours later), it can restore power to the E20, which will boot and can then continue with whatever data polling or processing you require. The power cost of this solution is approximately 22 mA of current (at 3.3 V) to the SNAP Engine while it is awake, though if you knew how long you expected to keep your E20 awake between sleep cycles, you could have your SNAP Engine sleep for a duration (at 1.3 µA) while the E20 performed its work, then have it wake up in time to hear the request to send the E20 (and itself) back to sleep.

A script like the following in your external SNAP Engine will serve this purpose (assuming an RF220 SNAP Engine, which performs a timed sleep at about 1.3 µA).

To invoke this, you would have the SNAP node in your E20 call `pause_and_disconnect_power(a, b)` where a is how long to wait before shutting the E20 down and b is how long to sleep before waking the E20 back up.

```
from synapse.platforms import *

@setHook(HOOK_STARTUP)
def _on_startup():
    writePin(GPIO_0, True)
    setPinDir(GPIO_0, True)

def pause_and_disconnect_power(delay_before_power_down,
                               pause_before_waking):
    """
    Use a brief sleep to control the timing of the delay,
    and then drive GPIO_0 low to disconnect external power.
    On waking, restore power through GPIO_0.

    Both delays are in seconds. The range of appropriate values
    is from 1 to 0xA8C0, or 43,200 seconds (12 hours). Because
    SNAPpy only handles 16-bit signed integers, this spans the
    range 1 to 32,767, and then -32,768 "up" to -22,336.
    Negative values between -22,335 and -1 will provide sleep
    durations measured in milliseconds, which will not be
    beneficial for power savings as the power required to reboot
    the E20 exceeds the power saved in the short sleep duration.

    See the SNAP Reference Manual for details about sleep modes.
    """

    if _validate_time(delay_before_power_down) \
        and _validate_time(pause_before_waking):
        sleep(2, delay_before_power_down)
        writePin(GPIO_0, False)
        sleep(2, pause_before_waking)
        writePin(GPIO_0, True)

def _validate_time(time):
    """Make sure time is in the range 0x0001 to 0xA8C0"""
    if type(time) == 2:  # time is an integer
        if time > 0:
            return True
        if time < 22335:
            return True
    return False
```

Summary

Disabling unnecessary peripherals is an excellent start, and can reduce current draw by up to 80mA if the Ethernet, Wi-Fi, cell, and SNAP modules are all suspended simultaneously. (This reduces current consumption to 100mA at 12V after booting into Linux — nearly 45%.) Clearly, you will periodically need some of these interfaces for the E20 to operate as a gateway to your SNAP network, but this provides a starting point for power savings by disabling unnecessary peripherals when not in use.

Even with all peripherals disabled, reducing the boot time is critical, as the current draw is much higher during this period (and the disabling of those peripherals will only take effect after Linux has fully booted.) By implementing the suggested boot configuration changes, you can reduce the E20's current draw by up to 180mA at times over a duration of 10 - 11 seconds of booting.

You may instead find it appropriate to shut the E20 down completely rather than having it go deaf to its environment and sit quietly waiting to turn things back on. Adding an external SNAP Engine to your configuration makes this fairly simple.

synapse®