# Table of Contents

## Overview
- Hardware Specs
- Mini Tutorials
- Pictures!

## Power
- Power Usage
- Power Jack
- Lithium Battery

## Connecting with TTL serial
- Make the Cable
- Connect

## Compiler
- Toolchain

## i2c Sensor
- i2c Twiddler
- Reading the MMA7455L accelerometer
- Acceleromate!
- Basic i2c twiddler Code
- MMA7455L reader

## WiFi
- Alternate Driver Installation For 8192.ko
- Alternate Driver Installation For RT2800
- Connecting to your WiFi network
- Start DHCP
- Set up IP & DNS
- Easy Way

## Ethernet
- Plug-in Ethernet
- With DHCP
- Without DHCP

## Audio
- Playing Local MP3s
- Playing Remote MP3s
- Splitting out the A/V
- Using the Mixer
- Speaker

## SD Card
- Cloning/Copying
- Fixing the partition table
- Resizing the partitions

## Support Forums
Overview

This page is a collection of mini-tutorials on doing stuff with the Chumby Hacker Board (for brevity we will refer to it as the CHB)! The CHB is a cool single board Linux computer that has much of the same hardware as the famous Chumby One. It's great for people who are experienced with Linux and want to have the power of a microcomputer with audio and video output while at the same time getting all the peripherals of a microcontroller such as analog-to-digital conversion, PWM outputs, sensors, bit twiddling, and broken-out GPIOs!

While we believe that the CHB is a fairly easy-to-use Single Board Computer (SBC) having a pre-installed OS on the included uSD card, and ready drivers for the peripherals, its not designed for beginners! The board is best used by those with previous Linux experience: the good news is you don't have to have another Linux computer to set up the CHB but you should have familiarity with shells and shell scripting, gcc, make, dmesg, etc. We also suggest having had some poking around with microcontrollers such as BASIC Stamp, Arduino, AVR, PIC, 8051, etc. So that when we say "i2c" and "not 5v tolerant I/Os" you can follow along.

The CHB is not in any way officially supported by Chumby Industries! Chumby has generously offered a Forum and Wiki where they will try to share information but there is absolutely no tech support or guarantee that the CHB will meet your project needs. Please do not contact Chumby directly either by email or phone for help with your CHB. If you have questions, please post to their forums to receive help from others and the occasional assistance from a CHB developer.
Want to pick one up? We have Chumby Hacker Boards and accessories in stock at the adafruit shop!

Hardware Specs

- Freescale iMX.233 processor running at 454 MHz
- 64 MB onboard RAM
- Comes with 512 MB uSD card with 100 MB Linux installation all ready to go
- 3.3V I/O pins can talk to most sensors, motor drivers, etc. No struggling with 1.8V levels.
- Low power, fanless ARM926 core draws only 200-300 mA
- Onboard GL850G USB hub draws 100-200mA
- Built-in Lithium Ion/Polymer battery charger and 5V boost converter for portable projects
- Three USB port jacks!
- 1.5W mono 4-16 ohm speaker amplifier (0.1" JST onboard connector)
- Microphone input (0.05" JST onboard connector)
- LCD controller with 2mm output port
- 3.5mm A/V output jack with stereo audio and NTSC/PAL composite video
- Back of board has GPIO outputs on 0.1" header spacing, plug in an Arduino proto shield (Beta version only, Final boards don't have this)
- Quadrature encoder connections onboard
- 5-way joystick on-board
- MMA7455 3-axis +/-2G to +/-8G accelerometer on-board
- 3.3V TTL serial port for easy shell access
- Full GCC toolchain is ready for you to download and get crackin'!

Mini Tutorials

- Power supplies - How to power your Chumby Hacker board!
- Serial port - How to connect to the serial terminal port for shell access
- Compiler - Installing the Falconwing GCC toolchain
- Accessing i2c - Connecting to i2c chips including the on-board accelerometer!
- WiFi networking - Getting to the Internet via wireless 802.11b/g (in progress)
- Stand-Alone WiFi AP and SSH - Turn your CHB into a stand-alone DHCP server
- Ethernet networking - Getting to the Internet via wired Ethernet
- Audio - Playing audio through the headphone/AV jack and onboard 2W speaker amp
- microSD - Cloning, copying and expanding the space on the card (in progress)
- VGA - VGA video out (totally in progress!)
• EC2 instance for building full, from-scratch images of their own for the hacker board () - Thanks Chumby!

More info may be found at the official Chumby Hacker Board Wiki ()

Pictures!
Click for large photos of the v1.0 PCBs

Click for large photos of the beta PCBs
Power

Power Usage

The Chumby Hacker board doesn't take a lot of power on its own but adding goodies can really boost up the draw, here are some things to look out for.

- The main system itself (processor, RAM, SD card) draws 200 to 300 mA depending on how hard its thinking
• The USB hub draws 100-200mA (not including power to the USB devices) depending on how many ports are active
• Having a speaker plugged into the 2W output adds as much as 300mA (louder speakers = more current draw)
• Charging an attached Lipoly battery will take another 500mA max
• Plug-in LCD (we don’t have one yet but say it existed) will draw 100-200mA depending on make and backlight brightness

Power Jack

The Hackerboard requires a regulated 5VDC power supply, positive tip, via a 3.5mm OD, 1.4mm ID barrel jack (sometimes referred to as a 3.8mm or 4mm connector). To connect you will need a matching plug, such as this (). We suggest a regulated switching supply () with over 1000mA output capability. The power required by the board itself is not that high (see above) but if you plug in a Wifi dongle into the USB port, or an LCD into the LCD expansion port, the power required can easily climb to a 1000 mA. Since the connector is the same as those used in the PSP, you can use a PSP charger.

Under no circumstances should you power the device with anything higher than 5V, you will destroy it!

Lithium Battery

The board can also use a standard 3.7-4.2V lithium ion/polymer battery for power. The board even has a built-in lipoly charger! It is activated by default and if you plug in a Lipoly battery you’ll see the kernel recognize it and start charging at up to 500mA.
The Lipoly breakout is a [JST B3B-EH-A connector](https://www.jst-connectors.com/) on the same side as the power and A/V jack, next to the single USB port. To connect, use a 0.1" or 2.5mm spaced female socket - the official one is [JST EHR-3](https://www.jst-connectors.com/) and crimp the matching crimps on a red, yellow/white and black wire (28 AWG, stranded is best). Slip in the crimped wires so that the board markings match up such that red is +, T is yellow, and - is black. The markings are on the opposite side of the PCB.

If your battery does not have a temperature output, you can simply solder a 10K resistor between yellow (temp wire) and black (ground). You'll probably not need a temperature sensor but in case you do, use a 10K NTC.

If powering from a lipoly, the chip will boost it to 5V to supply power to one of the USB ports (the top one, on the same side as the power jack) but not to the other two. There is a fourth USB connection on the header where I2C and other DIO pins are - this one is also powered by the lipoly boost system.

The lipoly is also used for the RTC backup! Any lipoly battery, when charged, will let the RTC run for years.

---

**Connecting with TTL serial**

The chumby has a four-pin, 3v level TTL serial connection port running at 115.2 Kbps. You cannot just connect this to your computer's serial port, parallel port or USB port without a converter. We think the best way to connect is using an [FTDI cable](https://www.ftdichip.com/Support/Documents/DataSheets/DS3050A.pdf) which
can be easily modified.

Make the Cable

You must use a 3.3v logic level FTDI cable, look for the letters "3V3" or similar on the cable. We also suggest checking with a multimeter that the voltage on the logic pins (everything but Red) is no higher than 3.3V. 5V logic will damage the chumby board, and it would be a shame if it broke before you got to have some fun!

Also, make sure you have an adapter that's a cable, so that you can move around the pins.

Use tweezers to lift up the black connector tabs.
Then gently pull out the wires
Repeat!

Rearrange the wires as shown, so you have Black (GND) then a space, Orange (TX) and Yellow (RX). You can clip the unused pins or cover them with heatshrink. They wont be used. Just make sure you don’t have the conductive pins accidentally touch your Hacker Board!!!
Connect

Plug in your FTDI cable, and install any drivers. See our instructions for driver installation for Windows, Mac and Linux.

Once the driver is installed, verify the COM or tty port. For example, the COM port for our FTDI cable is COM34 (we have a lot of FTDI cables!). Open up a Serial Terminal program and connect to that COM or tty port at 115.2Kbps without hardware handshaking.
Connect a 5V regulated supply into the chumby and your FTDI cable so that the black wire lines up with the GND labeled pin.

You should see the following on your terminal:
Hit return a few times to get the shell!

Compiler

Toolchain

You can easily install an open source toolchain to get started developing right on the Chumby Hacker board!

Step one is to download the following file [http://files.chumby.com/hacks/falconwing_toolchain.sh](http://files.chumby.com/hacks/falconwing_toolchain.sh) (20M) and save it onto any USB key. Then insert that USB into one of the CHB's USB ports.
Run `df` to find the name of the USB key

Copy the file from the USB key onto `/mnt/storage`

Then cd into `/mnt/storage` and run the `/falconwing_toolchain.sh` script to install the toolchain.
When you are done, you can rm falconwing_toolchain.sh to free up some space.

Please save a backup of the falconwing_toolchain.sh file somewhere so you don't end up having to re-download it (Chumby is kindly hosting this file but its very large!)

---

### i2c Sensor

### i2c Twiddler

First, you'll need to have a toolchain installed so make sure you go back and install it!

Sean Cross wrote a great i2c twiddler-tool. This allows you to poke and peek at i2c chips right from the command line. Nice!

Scroll down and copy his i2c C code. Open up your terminal to the CHB and type in m kdir /mnt/storage/dev (or whatever place you want to store your code), then cd /mnt/storage/dev and finally cat > i2c.c into the terminal and hit return. Then paste in the code and finish by typing Control-D
Then compile the code with gcc by typing in `gcc -o i2c i2c.c` to create the i2c executable. Then run it and make sure you get the response below.

```
Reading the MMA7455L accelerometer

There's a Freescale +2G to +8G 3-access accelerometer on the CHB for you to play with, lets get some readings. First off, we need to know what the i2c address is. Open up the datasheet () and look for the section called "i2c Secondary Address".

$1D means hexadecimal 0x1D which is the same as decimal 29. Great! Lets read byte #0 from the accelerometer by typing in ./i2c r 29 0
```
We got an error that the register was not readable. This means the chip could not be found on the I2C bus. :( But then we remember that I2C addresses are 7 bits long and are transmitted in the upper bits of an 8-bit byte. So we actually need to shift the address up by 1 bit. That's easy to do, though, just multiply by 2 to get I2C address 58.

Rock! Now we need to figure out what registers we can read, looking at the datasheet we see:

There's a lot of stuff! Let's start with one we know is going to work, like $0D (hex 0x0D = dec 13)
It in fact returns the value 0x1D which we know is the i2c address.

**Acceleromate!**

OK so now we want to get that XYZ data, right? Looking at the register file it seems like the first 6 bytes are used for 10-bit readings, but we can get single 8 bit readings from registers number 6, 7 and 8.

If you're careful you can read those registers while having a friend gently shake the board, you'll see different values returned.

However, wouldn't it be great if you didn't need a friend to shake the board while you pressed Up-arrow & Return? Lets edit Sean's code. To begin we will put #define's in for the address and registers.

```c
// The 'raw' 7 bit address shifted up
#define MMA7455_I2CADDR (0x1D * 2)

// The registers to read!
#define MMA7455_XOUT8 6
#define MMA7455_YOUT8 7
#define MMA7455_ZOUT8 8
```
Then replace the main() function with our own which is shorter and only reads those registers to print out the values.

```c
int main(int argc, char **argv) {
    int i2c_file;
    int8_t x, y, z;  // the readings are 8 bits and signed!

    // Open a connection to the I2C userspace control file.
    if ((i2c_file = open(I2C_FILE_NAME, O_RDWR)) < 0) {
        perror("Unable to open i2c control file");
        exit(1);
    }

    // ignore arguments!
    while (1) {

        // read X and Y and Z from the register
        if( get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_XOUT8, &x) \\
            ||
            get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_YOUT8, &y) \\
            ||
            get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_ZOUT8, &z) ) {
            printf("Unable to read register!\n");
            return -1;
        }

        printf("X = %d	Y = %d	Z = %d\n", x, y, z);
    }

    close(i2c_file);
    return 0;
}
```

Note the while() loop, and that we read all three registers and stick the results into 8-bit signed variables. Then we printf() 'em all and loop again.

You can grab all of the code below.

Stick the code in a new file called mma7455.c by copying and pasting as before. Then compile by running `gcc -o mma7455 mma7455.c` and run with `.mma7455`. Now shake it!
There you go! Now you can talk to the accelerometer to get motion data, and this code is easily adaptable for any i2c chip you may want to use. Enjoy!

Basic i2c twiddler Code

/*
 * This software uses a BSD license.

Copyright (c) 2010, Sean Cross / chumby industries
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the
  distribution.
* Neither the name of Sean Cross / chumby industries nor the names
  of its contributors may be used to endorse or promote products
  derived from this software without specific prior written
  permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/
#include <stdio.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#define I2C_FILE_NAME "/dev/i2c-0"
#define USAGE_MESSAGE \
  "Usage:\n    %s r [addr] [register] to read value from [register]\n    %s w [addr] [register] [value] to write a value [value] to register [register]\n   " \

static int set_i2c_register(int file, unsigned char addr, unsigned char reg, unsigned char value) {
  unsigned char outbuf[2];
  struct i2c_rdwr_ioctl_data packets;
  struct i2c_msg messages[1];

  messages[0].addr = addr;
  messages[0].flags = 0;
  messages[0].len = sizeof(outbuf);
  messages[0].buf = outbuf;

  /* The first byte indicates which register we'll write */
  outbuf[0] = reg;

  /*
  * The second byte indicates the value to write. Note that for many
  * devices, we can write multiple, sequential registers at once by
  * simply making outbuf bigger.
  */
  outbuf[1] = value;

  /* Transfer the i2c packets to the kernel and verify it worked */
  packetsmsgs = messages;
  packetsmsgs = 1;
  if(ioctl(file, I2C_RDWR, &packets) &lt; 0) {
    perror("Unable to send data");
    return 1;
  }

  return 0;
}

static int get_i2c_register(int file, unsigned char addr, unsigned char reg, unsigned char *val) {
  unsigned char inbuf, outbuf;
  struct i2c_rdwr_ioctl_data packets;
struct i2c_msg messages[2];

/* In order to read a register, we first do a "dummy write" by writing
 * 0 bytes to the register we want to read from. This is similar to
 * the packet in set_i2c_register, except it's 1 byte rather than 2. */
 outbuf = reg;
 messages[0].addr  = addr;
 messages[0].flags = 0;
 messages[0].len   = sizeof(outbuf);
 messages[0].buf   = &outbuf;

/* The data will get returned in this structure */
 messages[1].addr  = addr;
 messages[1].flags = I2C_M_RD/* | I2C_M_NOSTART*/;
 messages[1].len   = sizeof(inbuf);
 messages[1].buf   = &inbuf;

/* Send the request to the kernel and get the result back */
 packets.msgs      = messages;
 packets.nmsgs     = 2;
 if(ioctl(file, I2C_RDWR, &packets) &lt; 0) {
   perror("Unable to send data");
   return 1;
 }
 *val = inbuf;
 return 0;

int main(int argc, char **argv) {
  int i2c_file;

  // Open a connection to the I2C userspace control file.
  if ((i2c_file = open(I2C_FILE_NAME, O_RDWR)) &lt; 0) {
    perror("Unable to open i2c control file");
    exit(1);
  }

  if(argc &gt; 3 &amp;&amp; !strcmp(argv[1], "r")) {
    int addr = strtol(argv[2], NULL, 0);
    int reg = strtol(argv[3], NULL, 0);
    unsigned char value;
    if(get_i2c_register(i2c_file, addr, reg, &amp;value)) {
      printf("Unable to get register!
");
    } else {
      printf("Register %d: %d (%x)", reg, (int)value, (int)value);
    }
  }
  else if(argc &gt; 4 &amp;&amp; !strcmp(argv[1], "w")) {
    int addr = strtol(argv[2], NULL, 0);
    int reg = strtol(argv[3], NULL, 0);
    int value = strtol(argv[4], NULL, 0);
    if(set_i2c_register(i2c_file, addr, reg, value)) {
      printf("Unable to get register!
");
    } else {
      printf("Set register %x: %d (%x)", reg, value, value);
    }
  } else {
    fprintf(stderr, USAGE_MESSAGE, argv[0], argv[0]);
  }
  return 0;
}
MMA7455L reader

/*
 * This software uses a BSD license.

Copyright (c) 2010, Sean Cross / chumby industries & Limor Fried / adafruit
industries (we are both industrious people, eh?)

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
 notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
 notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.
* Neither the name of Sean Cross / chumby industries nor the names
 of its contributors may be used to endorse or promote products
developed from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/

#include <stdio.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>

// The 'raw' 7 bit address shifted up
#define MMA7455_I2CADDR (0x1D * 2)

// The registers to read!
#define MMA7455_XOUT8 6
#define MMA7455_YOUT8 7
#define MMA7455_ZOUT8 8

#define I2C_FILE_NAME "/dev/i2c-0"
static int get_i2c_register(int file,
    unsigned char addr,
    unsigned char reg,
    unsigned char *val) {
    unsigned char inbuf, outbuf;
    struct i2c_rdwr_ioctl_data packets;
    struct i2c_msg messages[2];

    /*
    * In order to read a register, we first do a "dummy write" by writing
    * 0 bytes to the register we want to read from. This is similar to
    * the packet in set_i2c_register, except it's 1 byte rather than 2.
    */
    outbuf = reg;
    messages[0].addr = addr;
    messages[0].flags = 0;
    messages[0].len = sizeof(outbuf);
    messages[0].buf = &outbuf;

    /* The data will get returned in this structure */
    messages[1].addr = addr;
    messages[1].flags = I2C_M_RD/* | I2C_M_NOSTART*/;
    messages[1].len = sizeof(inbuf);
    messages[1].buf = &inbuf;

    /* Send the request to the kernel and get the result back */
    packets.msgs = messages;
    packets.nmsgs = 2;
    if(ioctl(file, I2C_RDWR, &packets) < 0) {
        perror("Unable to send data");
        return 1;
    }
    *val = inbuf;
    return 0;
}

int main(int argc, char **argv) {
    int i2c_file;
    int8_t x, y, z;  // the readings are 8 bits and signed!

    // Open a connection to the I2C userspace control file.
    if ((i2c_file = open(I2C_FILE_NAME, O_RDWR)) < 0) {
        perror("Unable to open i2c control file");
        exit(1);
    }

    // Ignore arguments!
    while (1) {

        // Read X and Y and Z from the register
        if( get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_XOUT8, &x) ||
            get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_YOUT8, &y) ||
            get_i2c_register(i2c_file, MMA7455_I2CADDR, MMA7455_ZOUT8, &z) ) {
            printf("Unable to read register!\n");
            return -1;
        }

        printf("X = %d\tY = %d\tZ = %d\n", x, y, z);
    }
    close(i2c_file);
    return 0;
}
The Chumby hacker board does not have Wireless capability built in, but its very easy to add! The best dongle to use is the **TP-LINK TL-WN321G (or any other RT73-based dongle)** which has drivers ready to go. Note that many dongles probably won't work - if you have one kicking around you can try it but don't be surprised when it is not recognized!

If you know of an adapter that does/doesn't work, you should post it here:

- **Works - RT73 based:**
  - Hawking HWUG1 with external antenna adapter. FCC ID: NDD957318S607

- **Works with alternate driver:**
  - **This** mini-dongle from Deal Extreme works using the **8192cu.ko** module.
  - **This** generic 802.11 b/g from Amazon works using the same driver as the TP link - the same RT2800 instructions apply.
  - **<yours here>**

- **Doesn't work**
  - **<here>**
Alternate Driver Installation For 8192.ko

Download the module and copy it to your chumby, say /mnt/storage/. Then run the following commands.

```bash
$ cd /mnt/storage/
$ mount -oremount,ro /
$ install -p -m 644 8192cu.ko /lib/modules/2.6.28-chumby/kernel/drivers/net/wireless/
$ depmod -a 2.6.28-chumby
$ mount -oremount,ro /
```

Alternate Driver Installation For RT2800

The TP-Links changed from RT73 to RT2800. You can tell which is which because when you plug in the RT2800 a green LED lights up in the casing.

Download this compressed file

Uncompress it and put it on a USB key. Plug the USB key into the CHB and copy it to /mnt/storage:

```bash
$ cd /mnt/storage/
$ mount -oremount,ro /
$ cp /mnt/usb-XXXXX/update-modules.sh .
$ ./update-modules.sh
```

```
chumby:/mnt/storage # ./update-modules.sh
1306+1 records in
1306+1 records out
663723 bytes (654.0KB) copied, 2.176775 seconds, 300.4KB/s

comcast-wireless updated. Reboot to use it.
chumby:/mnt/storage #
```

© Adafruit Industries
Connecting to your WiFi network

Plug in the dongle

Run iwlist wlan0 scan to list all the local networks

If you don't get anything, check dmesg in case you don't have the driver necessary, this can be a little confusing. See the driver list above. Find your network, we'll connect to adafruit (not listed here, just substitute your own connection name).

Configure the wlan adapter (we'll be using WEP here, WPA is more complex).

- Set the WiFi password with `iwconfig wlan0 enc XXXXXXXXXX` where XXXXXXXXXX is your 10 digit hexadecimal WEP password (ps. our password isn't really DEADBEEF00)
- Set the channel to match the one in the iwlist output with `iwconfig wlan0 channel NN`
• Set the network you want to connect to with iwconfig wlan0 essid adafruit where adafruit is substituted with your network name

You should get a [1932.400000] ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready response from the kernel.

Start DHCP

DHCP will let you get your IP, route and DNS right from the router.

You'll need to remount root so that dhclient can edit it.

Start up the wlan0 connection

Then run dhclient wlan0 (the image doesn't show the wlan0 because I had already run dhclient to do this tutorial, make sure to stick the wlan0 part on!)

You should be able to ping!
Set up IP & DNS

Say you don’t have DHCP or don’t want to use DHCP for some reason.

Pick an IP address, and ifconfig wlan0 <ipaddress> up to give yourself an IP address

Don’t forget to add a default route, through your router.

If you want DNS you’ll have to remount the root directory rw and edit /etc/resolv.conf

If you need to, set up your resolv.conf so that you have one or two DNS servers in there
When you're done, be tidy and remount / as read only, k?

Easy Way
In this post by ChumbyLurker you'll find an easy way to configure your wireless, WPA and all.

Ethernet

The Chumby hacker board does not have Ethernet capability built in, but its very easy to add! We have found that ASIX AX88772 chipset modules work. Note that many dongles probably won't work - if you have one kicking around you can try it but don't be surprised when it is not recognized!

<if you know of an adapter that does/doesn't work, you should post it here>
Linksys USB300M works (uses ASIX AX88772 chipset)

Apple USB 10/100 Ethernet adapter (ASIX AX88772) works fine

**Plugable USB 10/100** works (another ASIX AX88772 based adapter).

**Plug-in Ethernet**

Get started by plugging in your USB-Ethernet adapter.

You can run `ifconfig` to verify all is OK

With DHCP

If you have DHCP running on your router, you can just run `dhclient eth0` to get going

You'll need to remount root so that `dhclient` can edit it.
Wait a bit, you should be able to ping!

When you're done, be tidy and remount / as read only, k?

Without DHCP

If you don't have DHCP or you want to pick your own IP address, simply run `ifconfig` on eth0 to set the ip address. Don't forget to set your default route as well!

If you need to, set up your `resolv.conf` so that you have one or two DNS servers in there.

When you're done, be tidy and remount / as read only, k?
Audio

Playing Local MP3s

You can play MP3's off of your USB key or if you've copied any to the /mnt/storage space with btplay –start-daemon file.mp3

MP3's will play in the background. To kill btplay run killall btplay

Audio will play through the headphone jack at max volume, just plug in your favorite headphones! (Warning: it may be really loud!)

Playing Remote MP3s

You can also stream music with btplay, just stick a URL in there! Of course, you will have to have network capability up and running first

Just as before, just plug into the headphone jack to verify its working. (Warning: it may be really loud!)
Splitting out the A/V

The headphone jack is not actually just audio - it's actually Audio and Video. That means that you if you just plug headphones you'll notice some buzzing - that's you listening to the video! Now if you're in a Cronenberg film this is probably OK but for most of us it's very annoying! There are two solutions. One is to split out the 3 signals into RCA jacks (with a camcorder cable)

![A/V cable](https://example.com/a-v-cable.png)

and then recombining with an RCA Y cable (such as [a Y female-RCA to male 3.5mm](https://example.com/y-cable) or [this](https://example.com/this-cable))

Note our A/V cables split out the signals as such - Red is Video, White and Yellow are stereo line-level audio which is not what you would expect!

Note apparently like every A/V cable is slightly different. If you get one, try different colors to figure out which ones have audio coming out of them and which have video.

Using the Mixer

If you want to change the volume of the headphones, you'll need to run the audio mixer software amixer
You can get and change the HeadPhone volume (HP) by running amixer sget HP and amixer sset HP

To set the volume to 1/2 for example, do this amixer sset HP 64
Speaker

The iMX.233 has a built-in mono speaker amplifier that can push up to 2W into a 4 ohm speaker! To use, connect the two speaker inputs into the Spkr 0.1” JST connection near the power jack. You can use either pin, they are not polarized as it is a mono output. Don't ground either of the speaker pins, they are "bridge tied" to improve output volume.

The chumby is 'switched' by default - if you connect headphones into the jack, the speaker will turn off and vice versa.

If you want the speaker on even with the headphone plugged in, you can use amixer cset

For example, the Speaker Playback Switch is off, so we turn it on with amixer cset numid=4 on to reactivate.

SD Card

The easiest way to do things for now is to use a linux machine or a bootable linux CD/DVD, or even a virtual machine running under Windows. Macintosh has most of these commands available too in a root window. Some things can be done on the Chumby hackerboard itself since dd (and fdisk) might be available.

Cloning/Copying

The original image is available from a link off the forum:


An earlier message references windiskimager:

https://launchpad.net/win32-image-writer/+download

The image in the .zip file has the partition table overlap problem, but it should be the
original boot image.

You can clone the 1G original microSD card if you have made modifications or otherwise don't want to use the image file. It can be written to any same or larger card. The standard way is to use "dd". Google "dd windows" for a version that works there, but basically you need to do a byte-for-byte image copy from the original device to the new one. You probably also want to make a backup of the card as well.

Cloning can be done on the hackerboard itself if you have a USB to microSD adapter. This will be something like:

```
dd if=/dev/mmcblk0 of=/dev/sda bs=64M
```

Since the board uses mmcblk0 for internal flash.

**NOTE:** DD IS DANGEROUS SINCE IT ACCESSES THE HARDWARE DIRECTLY - YOU CAN END UP OVERWRITING YOUR WINDOWS DRIVE IF YOU AREN'T CAREFUL. Make sure everything is backed up or you know exactly what you are doing.

```
dd if=/dev/sdX of=/dev/sdY bs=64M
```

Windows will have something different for the /dev entries. /dev/sdX is the original card. Under linux, if you do "cat /proc/partitions" you will find a list - /dev/sda is typically your system drive. You can do this before inserting the original card, then look what appears after about 10 seconds - the new partitions will be from the new drive, which should be the chumby hacker image. You can also do it directly on the Chumby - in this case the values should be if=/dev/mmcblk0, and of=/dev/sda but I would still check /proc/partitions to see what was there.

If you can access your main hard drive, I would first back things up (if you have 1G of free space) by using dd, "dd if=/dev/sdX of=/media/c-drive/chb.img bs=64M". /media/cdrive is wherever the hard drive shows up under linux, or something like "C:\chb.img" under windows. This will create an image file of the uSD so it can be restored if anything goes wrong. The "bs=64M" copies 64 megabyte chunks which will be a lot faster as it normally copies bytes. Then you can remove the chumby SD and insert the clone target and just reverse "if" and "of": "dd if=/media/c-drive/chb.img of=/dev/sdX bs=64M". Then test the clone.

You can also do so with the rom image:

```
dd if=rom-falconwing.img of=/dev/sdX bs=64M
```
Fixing the partition table
Before you can change the partition sizes you need to fix the partition table.

The limited fdisk on the Chumby hacker board can't be used for this - you will need to boot into linux.

(note, it is probably easier to patch a few bytes with a hex editor, especially the original - I need to compare the original with the fixed MBR table and document the changed bytes in the first partition entry).

Reinsert the clone after the copy (remove and reinsert to get the updated disk structure). Run "fdisk /dev/sdX". This is a basic partition editor. It will complain because the structure is broken. Type "p" to print. You will see the first partition overlaps the next two. do "d" then "1" to delete the partition from the table (it isn't reformatted or anything), then "n", "p", it will default to partiton 1, then use the defaults for first and last. Do "t" then "1" then "53(return)" to reset the type, type "p" to see the changes (no overlaps!), then "w" to write it to the disk.

Resizing the partitions
I use gparted, available on some bootable fixit CDs or almost every bootable linux CD or USB stick.

Partition 4 is the extended container, and within, partition 6 is the "storage" partion". You will first need to move and/or resize partition 4 to make partition 6 larger. I would suggest also moving it on larger cards so you can make the other (root) partitions larger too.

Run gparted from the menu or "sudo gparted /dev/sdX" from the command line. Click on partition 4 and do move/resize, and slide it to give free space before if you want to expand the other partitions, but make sure it ends at the end of the disk. Then select the storage partition within and stretch it to the end of the new space. Tell it to apply changes.

I have a 16Gb image with plenty of room for everything.

(Note from BBotany on a different route to expanding the filesystem.) If you are working on a system that does not contain gparted, it is possible to use cfdisk to expand the final partition. Simply select the partition, and apply the "maximize" option. This will also extend the extended partition container (aka "partition 4") automatically. However, you will need to resize the filesystem that is on the partition separately; it is
easy to use "resize2fs" to do this. If this is not on the machine you changed partition sizes on (or you forgot to expand the filesytem before reinserting the SD card in the hackerboard), it is present on the hackerboard by default. After the partition table has been modified, simply run resize2fs on the partition. If you are on your own linux host, you presumably know where this is. If you are on the hackerboard itself, then the commands are as follows: unmount the partition with "umount /mnt/storage/", run the resize as: "resize2fs /dev/mmcblk0p6", and then remount it with "mount /mnt/storage/". When I upgraded to an 8GB SD card, this gave me a /mnt/storage of 6.4GB.

Support Forums

Support Forums ()