Transform Cobalt Raq3 into a Raspberry Pi-powered Media Center

Anyone remember these adorable blueish 1U servers made by Cobalt Networks?

While I was never in true love with the Cobalt OS itself, I actually liked the Cobalts Raq enclosure.
So much that I salvaged one while cleaning out a data center last summer. I decided to grant it a second live as a media center box running OSMC.
And of course it's powered by a Respberry Pi. Nowadays there's simply no way around those nice little boxes ;-)

From todays perspective, the specs of the Raq3 are rather low, a 300 MHz AMD K6, barely some 256 megs of RAM and a 20 gig hard drive. Well, back in the days, this was good enough to do simple web hosting. But today, there's not much use to boxes like these.

Oh, and if anyone has a Cobalt Cube to donate, let me know. I'll surely put it to good use :-)

Anyway, for my current idea to become reality, I first needed some parts.

Parts

So here's the things for the shopping list:

- A first generation Raspberry Pi, Model B+ (the one with 4 USB ports), or the more powerful Raspberry Pi 2
- A class 9 MicroSD card with some 8 Gigs or more

1 panel mountable Ethernet socket with cable (from Adafruit)]
2 panel mountable USB Type A-M to A-F sockets (from Adafruit)]
1 panel mountable HDMI socket (Adafruit)]
USB micro-B cable (again Adafruit)]
5V Pro Trinker, also from Adafruit]
Adafruit also has some nice multicolor 16x2 LCDs (No, I'm not having stock options with them!)]

- Standalone ATmega 328P chip, eventually a DIL socket to allow for easy replacement
- 7 LEDs in different colors, 1 green, 2 yellow, 2 orange, 2 red (or whatver colors you like)
- 2 x 220 ohm resistors, 2 x potentiometers, some wires, header connectors, flat cable, some bakelite boards (or perma protoboard from Ada....), etc

Make new Room

Before the Cobalt will come to new life, we'll need to yank out the interior. Remove everything except the power supply.
The front panel needs to get disassembled as well.

I went on and cut them apart using my Dremel. I kept only the right-hand part with the buttons, and the left-hand part with the LEDs. Also keep the mounting bracket which holds the current LCD in place. I scrapped the original LCD panel, as I wanted to replace it with a new one with a blue backlight.

If you want to keep the original panel, this is fine as well. Rumour has it, that it has the same pinout than my replacement from Adafruit. Though I did not verify this myself, YMMV.

OSMC and Package Installation

So first I installed OSMC to the Pi. The procedure is straight vorward and documented here.

After OSMC is installed, we need some additional packages which are not available by default.
Since OSMC builds on Rasbian, it's easy to install additional packages.
Connect to your Raspberry Pi through SSH. The default credentials are **osmc** for both the username and password.

Now for the packages: I'm adicted to vim, which is why I installed it. You will need **build-essential** and **python-pip**. Also, **python2.7-dev** is required. Since Python 2.7 is pre-installed on OSMC I stick with it.
If you want to go for Python 3, install **python3-dev** and **python3** as well.
Also **git** and **python-smbus** are required.
apt-get update
apt-get install vim build-essential python-pip python2.7-dev python-smbus git

Then let **pip** build and install the **RPi.GPIO** library:
pip install RPi.GPIO netifaces uptime

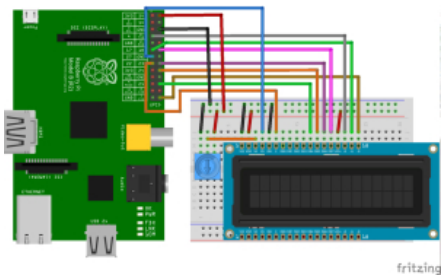
For the LCD display, an extra library from Adafruit is needed.
cd ~
git clone https://github.com/adafruit/Adafruit_Python_CharLCD.git
cd Adafruit_Python_CharLCD
sudo python setup.py install

You may want to cleanup this afterwards:
rm -rf ~/Adafruit_Python_CharLCD

Hooking up the LCD

For the LCD display, there's a great tutorial at Adafruit, which covers the basics on the wiring and more useful instructions.

I included the original wiring diagram here as a reference as well.



I used this as a starting point to wire up my LCD panel.
For the final wiring however, please have a look at the diagram further down below as decided to go a slightly different route.

It's especially important to note that I did not connect the backlight controls to the Raspi but rather preferred to hard-wire the backlight to emit blue light constantly. If you went to buy the multicolor LED, you can easily adapt this to use another color or have do the full backlight wiring. Then you will need to stick more closely to Adafruits tutorial.

Hooking up the "Glowing LED"

The front panel has this nice Cobalt logo sitting right next to the LCD panel.

An LED will be hooked up to the Raspberry Pi on GPIO#10. We will be using PWM to get a glowing / pulsating effect on the LED. To achieve this effect, I'll drive the LED at cycle rate between 10 and 100 percent, which means it will actually never be fully turned off. Also I'm using different sleep intervals to reduce the LED brightness more quickly, but give it more time to light up.

Here's a code example on how to do it:

```
for dc in range(10, 100, 5):
    p.ChangeDutyCycle(dc)
    time.sleep(0.2)
for dc in range(100, 10, -5):
    p.ChangeDutyCycle(dc)
    time.sleep(0.1)
```

Don't worry, I'll disclose my full code below. This was just to illustrate it.

The VU Meter

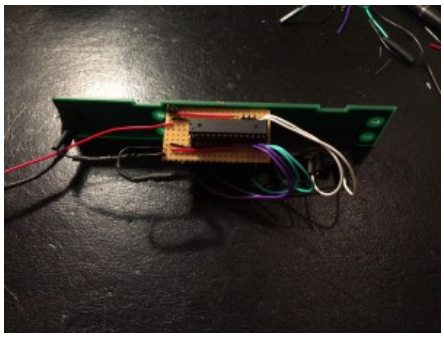
Yet more wiring is needed for the VU meter.
I decided to build my own VU meter using a ATmega 328 chip, which runs standalone besides the Raspberry Pi.
Why is that? Because the Pi doesn't have any analog pins, so I'm unable to perform the analog readout for the VU meter magic.
The standalone chip will serve this purpose, so once more I refer to my previous post about incorporating a standalone ATmega.
Essentialy it's only about some LEDs hooked up to the IC, then a potentiometer to control sensitiveness of the signal, and the 3.5mm audio jack connected to an analog pin to do the audio detection.

Right, we're grabbing audio input off the Pi's 3.5mm audio jack. This also means that you must configure Kodi for dual audio output on HDMI and analog audio. Your audio receiver should therefore be hooked up via HDMI, because there's no analogue audio to re-use, unless you add a twin adapter to the port.

Here's the sketch for it.

Later on in this article you'll find the complete wiring diagram.
Please note that I also added a DIL socket and some headers to the PCB, the former to be able to replace the IC, the latter to hook up an in-system programm (ISP) for to allow for easy software updates without removing the chip.

I have included this in my wiring diagram. You'll need extra headers for one pin on +5V, one pin for GND, and the pins 11 (MOSI), 12 (MISO) and 13 (SCK). Have also a look again here.

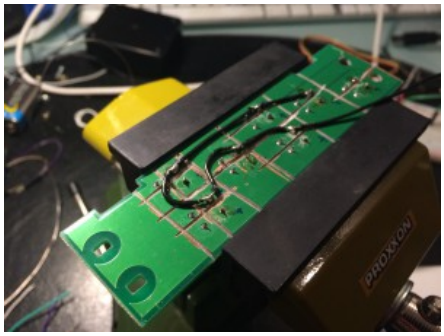


Make Front Panel Keys Functional

There's six keys on the front panel (well, actually seven, if you count the reset key as well, I'll spare this one though).
This makes it ideal to use them for navigation in OSMC, one key is to serve as ENTER key, one serves as BACKSPACE, the other for will be used as UP, DOWN, LEFT and RIGHT navigation keys.

So there's two ways to do it.
First I could connect the pins to the Raspi and have it read the key press event, then send a command to Kodi.
How to send the command? Either by doing some weird stuff by sending emulated key presses (yeeerks) or by sending the movement commands through the JSON/REST API.

But then I found the Adafruit Pro Trinket, which is cheap and still capable enough to emulate a USB keyboard.
Yes, that's right, that's exactly what we gonna do. Especially since I've never done it before, this sounded interesting to me.
So it's all about connecting the buttons from the front panel to the trinket, check out the wiring diagram further down below for details.
One thing is worth noting though: Before you solder some wires to the connector pins on the key panel, make sure that you cut the connectivity of the copper layer of the PCB, so each button has it's own "island". If you don't, the copper layer may cause undesired crosstalk which makes it hard to properly detect the key presses.



Software-wise you need only to install the Trinket USB library to your Arduino library folder. Adafruit has a wonderful tutorial about this as well.

And be very careful with the Trinket. Do not hit the reset button for too long, or you may erase the boot loader.
Also it seems not very fond of sudden disconnects. The only downside I found and yet killed the Trinket three times during my tests so far.
To restore the booloader follow the instructions over at Adafruit.

Good, but now for the actualy code (download Arduino sketch here), which I hope to be explanatory enough by itself:

```
/*
 * cobalt_keys - read cobalt front panel buttons to control Kodi
 *
 * Copyright (c) 2015, Gianpaolo Del Matto, www.phunsites.net
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
 *
 * 2. 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.
 *
 * 3. Neither the name of the copyright holder 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 pro trinket library
 *
 *   for some reason, which I had no time to resolve, this behaves very differently with my setup.
 *   according to official examples, only the single "include <ProTrinketKeyBoard.h>" is required.
 *
 *   however, if I included only the <ProTrinketKeyBoard.h> file, I would always end up with linker errors.
 *   only by adding the full path includes for <cmdline_defs.h>, <ProTrinketKeyboard.h>
 *   and <usbconfig.h>, the compiler would actually be happy with me.
 */
#include </full/path/to/Arduino/libraries/Pro_Trinket_USB_Keyboard_Library-master/cmdline_defs.h>
#include </full/path/to/Arduino/libraries/Pro_Trinket_USB_Keyboard_Library-master/ProTrinketKeyboard.h>
#include </full/path/to/Arduino/libraries/Pro_Trinket_USB_Keyboard_Library-master/usbconfig.h>
#include <ProTrinketKeyboard.h>


/* key definitions */
#define PIN_ARROW_LEFT A0
#define PIN_ARROW_RIGHT A1
#define PIN_ARROW_UP A2
#define PIN_ARROW_DOWN A3
#define PIN_ENTER A4
#define PIN_BACKSPACE A5

/* LED definition */
#define PIN_LED 13

/* arrays to track our states */
int state_KEY_ARROW_LEFT = 1;
int state_KEY_ARROW_RIGHT = 1;
int state_KEY_ARROW_UP = 1;
int state_KEY_ARROW_DOWN = 1;
int state_KEY_ENTER = 1;
int state_KEY_BACKSPACE = 1;
```

```
/* init */
void setup() {
    // enable LED pin
    pinMode(PIN_LED, OUTPUT);
    digitalWrite(PIN_LED, LOW);

    // button pins as inputs
    pinMode(PIN_ARROW_RIGHT, INPUT);
    pinMode(PIN_ARROW_LEFT, INPUT);
    pinMode(PIN_ARROW_UP, INPUT);
    pinMode(PIN_ARROW_DOWN, INPUT);
    pinMode(PIN_ENTER, INPUT);
    pinMode(PIN_BACKSPACE, INPUT);

    // enable internal pullups
    digitalWrite(PIN_ARROW_RIGHT, HIGH);
    digitalWrite(PIN_ARROW_LEFT, HIGH);
    digitalWrite(PIN_ARROW_UP, HIGH);
    digitalWrite(PIN_ARROW_DOWN, HIGH);
    digitalWrite(PIN_ENTER, HIGH);
    digitalWrite(PIN_BACKSPACE, HIGH);

    // start USB
    TrinketKeyboard.begin();
}

/* main loop */
void loop() {
    // always reset the indicator LED, switch it off and make a short sleep
    digitalWrite(PIN_LED, LOW);
    delay(5); // stay below 10 ms or the pseudo-keyboard may get disconnected, see below

    // the poll function must be called at least once every 10 ms or cause a keystroke
    // if it is not, then the computer may think that the device has stopped working, and give errors
    TrinketKeyboard.poll();

    // read all inputs at once so even concurrent presses could be detected this way,
    // though I propably wont ever need it
    //
    int r_KEY_ARROW_RIGHT = digitalRead(PIN_ARROW_RIGHT);
    int r_KEY_ARROW_LEFT = digitalRead(PIN_ARROW_LEFT);
    int r_KEY_ARROW_UP = digitalRead(PIN_ARROW_UP);
    int r_KEY_ARROW_DOWN = digitalRead(PIN_ARROW_DOWN);
    int r_KEY_ENTER = digitalRead(PIN_ENTER);
    int r_KEY_BACKSPACE = digitalRead(PIN_BACKSPACE);


    // handle ARROW RIGHT key
    // by checking inverse state, I enforce an absolute debounce,
```

```
// which means, that only one keypress is detected, no matter
// how long the key is actually pressed.
// ok, this also prevents things like "key repeat after nn ms",
// but actually I don't want this: I want to press the keys
// on the front panel repeatedly. As it turned out for me,
// the haptic feedback in Kodi seemed more tactile
// to me than without it.
//
if(r_KEY_ARROW_RIGHT != state_KEY_ARROW_RIGHT) {

  // right, we send a key press only if the input reads LOW
  // also, I'm quickly flashing the LED on the trinket,
  // that's actually only as a debugging aid to indicate, that
  // "something" was received.
  //
  if(r_KEY_ARROW_RIGHT == LOW) {
    digitalWrite(PIN_LED, HIGH);
    TrinketKeyboard.pressKey(0, KEYCODE_ARROW_RIGHT);
  }

  // save the current key state to compare key states on next
  // iteration again. we even do so when no actual processing
  // happened (INPUT did NOT read LOW)
  //
  state_KEY_ARROW_RIGHT = r_KEY_ARROW_RIGHT;
}

// next sections are identical to the one above, they follow the same logic.
// yes, I know, I did not respect the DRY (don't repeat yourself) pragma ;-)
// I should propably implement a bitwise-ANDed matrix to make this more lean.

// handle ARROW LEFT key
if(r_KEY_ARROW_LEFT != state_KEY_ARROW_LEFT) {
  if(r_KEY_ARROW_LEFT == LOW) {
    digitalWrite(PIN_LED, HIGH);
    TrinketKeyboard.pressKey(0, KEYCODE_ARROW_LEFT);
  }
  state_KEY_ARROW_LEFT = r_KEY_ARROW_LEFT;
}

// handle ARROW UP key
if(r_KEY_ARROW_UP != state_KEY_ARROW_UP) {
  if(r_KEY_ARROW_UP == LOW) {
    digitalWrite(PIN_LED, HIGH);
    TrinketKeyboard.pressKey(0, KEYCODE_ARROW_UP);
  }
  state_KEY_ARROW_UP = r_KEY_ARROW_UP;
}

// handle ARROW_DOWN key
```

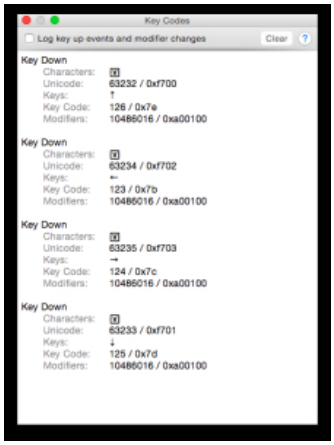
```
    if(r_KEY_ARROW_DOWN != state_KEY_ARROW_DOWN) {
      if(r_KEY_ARROW_DOWN == LOW) {
        digitalWrite(PIN_LED, HIGH);
        TrinketKeyboard.pressKey(0, KEYCODE_ARROW_DOWN);
      }
      state_KEY_ARROW_DOWN = r_KEY_ARROW_DOWN;
    }

    // handle ENTER key
    if(r_KEY_ENTER != state_KEY_ENTER) {
      if(r_KEY_ENTER == LOW) {
        digitalWrite(PIN_LED, HIGH);
        TrinketKeyboard.pressKey(0, KEYCODE_ENTER);
      }
      state_KEY_ENTER = r_KEY_ENTER;
    }


    // handle BACKSPACE key
    if(r_KEY_BACKSPACE != state_KEY_BACKSPACE) {
      if(r_KEY_BACKSPACE == LOW) {
        digitalWrite(PIN_LED, HIGH);
        TrinketKeyboard.pressKey(0, KEYCODE_BACKSPACE);
      }
      state_KEY_BACKSPACE = r_KEY_BACKSPACE;
    }

    // reset pressed key
    TrinketKeyboard.pressKey(0, 0);
}
```

Once you've uploaded the code, each key press should trigger the proper key code being sent to your computer.
If you want to be absolutely sure, that all key presses are properly sent, you should use a key code debugger.
On OS X, the Key Codes utility can be used. It's free and can be loaded directly from the App Store.

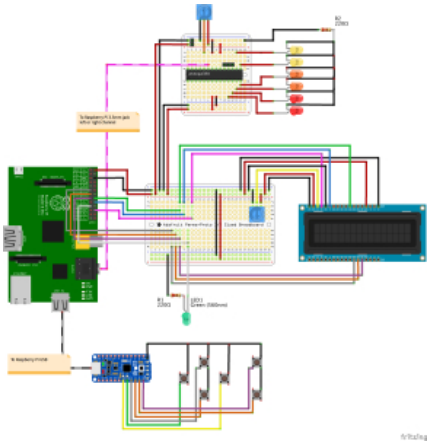


It has a simple but yet useful interface which will capture the key press and display some info about.

It will help you to see if all works fine. And if you're getting multiple key readings, although you only pressed one key, be sure to cut the PCBs copper layer properly, so the buttons are absolutely 200 % sure not be interconnected to each other in any imaginable way except for the common ground wire.

The Wiring Diagram

And now finally, here's the wiring diagram for everything I talked about so far.



cobalt_pi - The heart of it all

Now that you've probably wired and soldered everything together, there's still one more thing: *tadadadaaaaa* .... curtain opens .... cobalt_pi[tm]

The heart of it all is cobalt_pi, my little python script, which does the magic. But what exactly does it do?

Well, first of all, it's a multithreaded script intended to be run at boottime.
It will start two threads, one is responsible to drive the PWM for the glowing LED effect, the other is master to the LCD panel.
Essentially it's doing a constant poll on Kodi's state, wether there is a player active or not.
If the player is inactive, we will display some system information like IP addresses (good to know if Kodi uses DHCP-assigned IP and you need to know it for SSH login), system load, uptime and alike neerdism.
On the other hand, if the player is active, the current state is polled via Kodi's JSON/RPC API. This will then be used to show title information and time progress, as seen in many CD/DVD/BD players.

Ok, all gadgetery, but still cool. And hey, it's nice to build something like that by yourself :-)

Install cobalt_pi like this to /opt directory:

```
cd /opt
wget http://phaq.phunsites.net/files/2015/09/cobalt_pi.tgz -O- | tar -xzpvf-
```

This will create a new directory in /opt. The script itself lies beneath **/opt/cobalt_pi/sbin/cobalt_pi.py**.
It's intended to be run via a shell script, which you should add to your **/etc/rc.local** file.
Add a line like this before the **exit 0** line ...

/opt/cobalt_pi/bin/cobalt_pi.sh &

... so it will look similar to this:

Before you start it, be sure to edit the **/opt/cobalt_pi/etc/cobalt_pi.ini** config file.
All relevant settings are configurable via the ini file, so you won't really need to touch the code if you need to change the pins or Kodi's JSON/RPC url:

[global]
KODI_JSONRPC_URL=http://localhost:80/jsonrpc

[gpio]
GPIO_PL1=10
GPIO_RS=25
GPIO_EN=24
GPIO_D4=23
GPIO_D5=17
GPIO_D6=27
GPIO_D7=22
GPIO_BACKLIGHT=0

[lcd]
LCD_COLUMNS=16
LCD_ROWS=2

Bringing It All Together

Right, so once you've got everything in place, this is what you get:

Mission: Accomplished!