

Chumbot: Introduction to the BaseBoard4

By Bob Smith

Scope

This document contains the speaker notes for the talk at the Home-Brew Robotics Club meeting of July 28, 2010. These notes are for the section of the talk on the BaseBoard4. The notes for the Chumby part of the are available here: http://www.linuxtoys.org/chumbot/chumby_as_robot.pdf. This part of the talk is intended as a hands-on introduction to the BaseBoard4 robotic controller card.

Problem Statement

Photo 1 shows a Chumby based robot with the following features:

- Single USB interface to the host
- A six-channel RC receiver
- Dual H-bridge motor controllers
- Quadrature wheel sensors
- Four bumper switches
- A Maxbotix range sensor
- A servo to aim the range sensor

How long would it take you to design and build an electronics board with these peripherals? Would you program an AVR for each peripheral and then try to network them with yet another AVR that provides the USB interface? Would you use CAN Bus or six separate serial ports? How long would it take to write the software for each peripheral? How long to define and write the software for the inter-AVR protocol and to write the host side protocol code?

We used a Demand Peripherals BaseBoard4 for the peripherals. Our electronics board is shown in Photo 2 and the schematic for it in Figure 1. As you can see the electronics board has connectors and resistors and not much else. The key to this simplicity was putting all of the peripherals into the FPGA of the BaseBoard4.

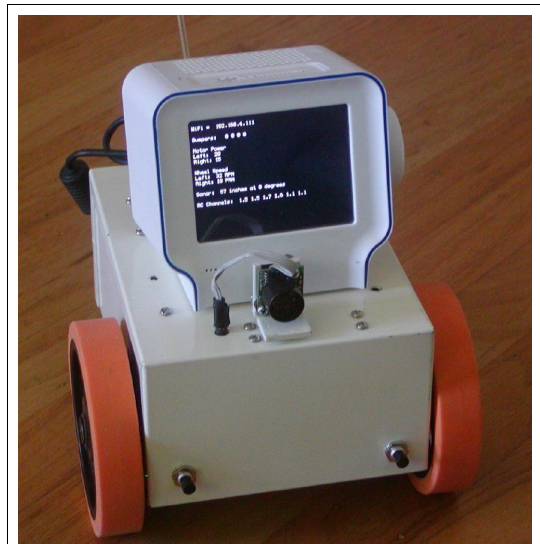


Photo 1: A Linux Based Robot

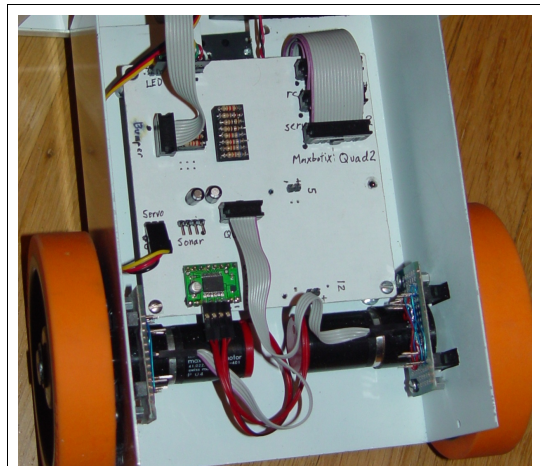


Photo 2: The Electronics Board

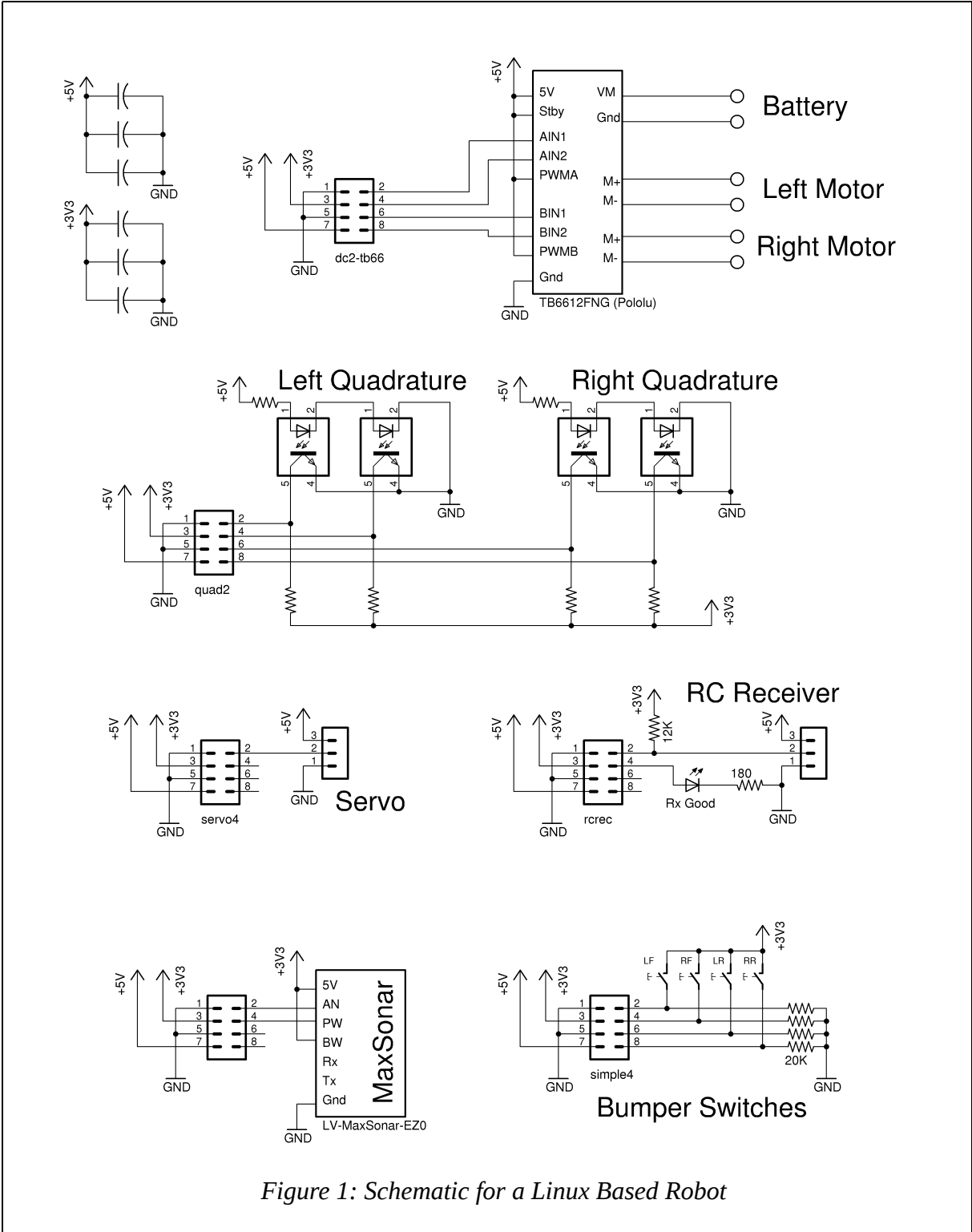


Figure 1: Schematic for a Linux Based Robot

The BaseBoard4: Robotic Peripherals On Demand

The BaseBoard4 is based on an FPGA, which allows the user to select up to nine different peripherals to use when the firmware is downloaded to the board. All peripherals come with Linux drivers. Ralph Gnauck is working on Windows drivers that have an identical (or at least similar) API. Eight of the nine peripherals have four FPGA pins and one peripheral has three pins. Photo 3 shows the assignment for the "slots" for the nine peripherals.

The underlying concept is that a peripheral has a set of addressable read/write registers to control it. If you've worked with PIC or AVR micro-controllers you are already familiar with registers.

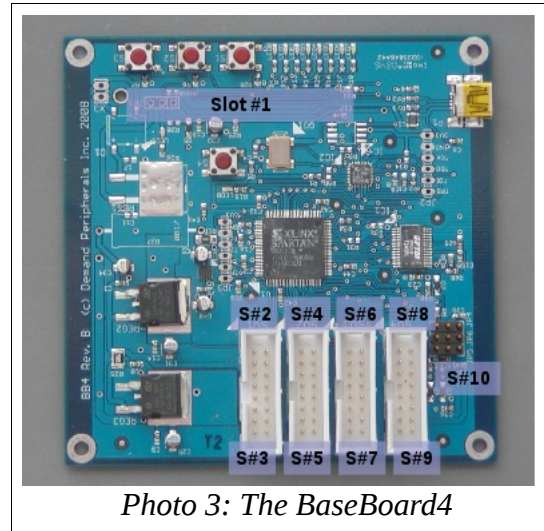


Photo 3: The BaseBoard4

BaseBoard4 and DPCore

Figure 2 shows the logical components in a BaseBoard4. The hardware is the BaseBoard4 and the firmware containing the peripheral logic is called *DPCore*. The host sends register read and write commands over a USB/serial link to an FTDI FT245 on the BaseBoard4. The FPGA reads the commands from the FTDI and puts the commands on the 16 bit address and 16 data buses. Figure 2 shows the peripherals built into the demo version of DPCore used for the talk.

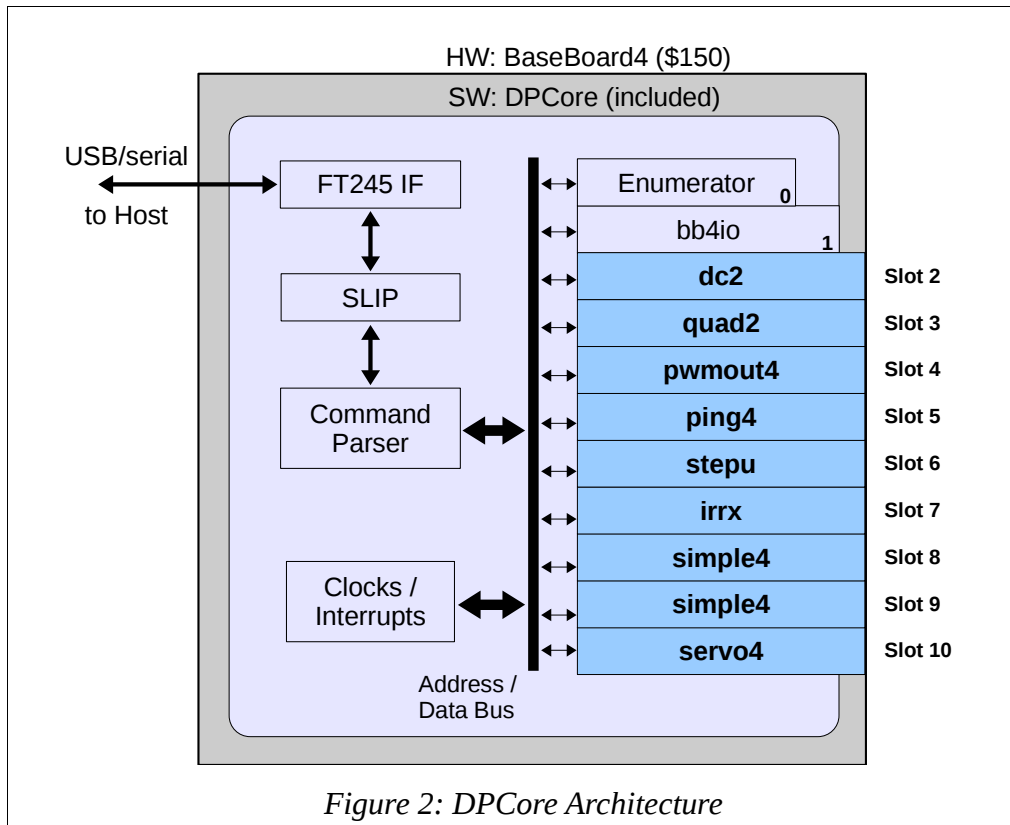


Figure 2: DPCore Architecture

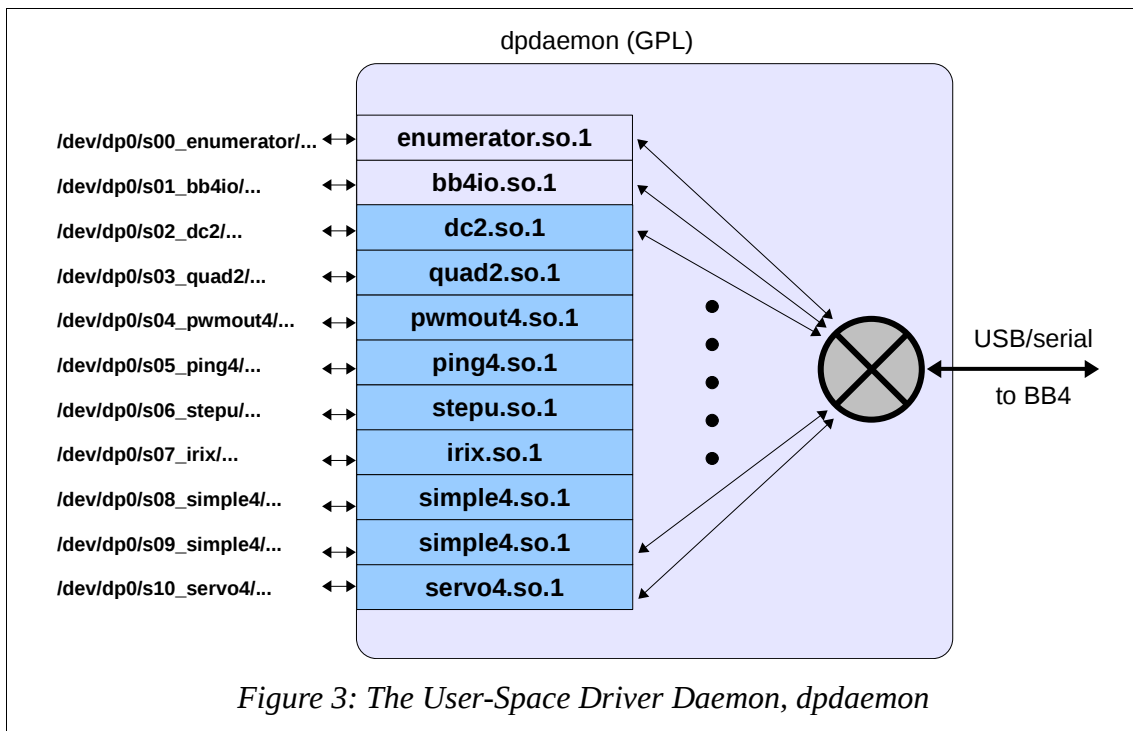
There are two peripherals that are in every build of DPCore. The Enumerator is a ROM that contains a list of the peripherals in the other slots. Host software reads the Enumerator at start-up in order to

know what drivers to load. The other peripheral in all builds of DPCore is BB4IO, the peripheral that deals with the buttons and LEDs on the BaseBoard4 itself. Nine peripherals are selected by the user. (We'll see how in just a minute.) The data sheet for each peripheral describes the pin-out for the peripheral, and many of the data sheets have a reference design and schematic to make the peripheral pin assignments clear. The peripheral does any precise timing required and the user-space driver maps the user visible API into register reads and writes.

Dpdaemon: A User-Space Driver

Figure 3 shows the architecture of the user-space driver daemon, *dpdaemon*. At power-up *dpdaemon* reads the Enumerator and loads the appropriate driver shared object file (.so) for the peripherals built into DPCore. As each driver .so file is loaded it sends commands to the BaseBoard4 to initialize the peripheral and it creates the device nodes that other programs use to access the peripheral.

Most peripheral registers map onto one device node under *dpdaemon*'s control, and it is *dpdaemon* that multiplexes the read and write requests over the USB/serial link to the BaseBoard4. The multiplexing action by *dpdaemon* means that each /dev node is independent of the other and you no



longer need to build your robot software as one big monolithic piece of code.

The best part of having real device nodes for the peripherals is that you can use any programming language that you want. We are using shell for the examples but C/C++, Java, Perl, Python all work equally well.

Using the -s option to *dpdaemon* you can replace the supplied .so driver file with one of your own. For example, you could select a Simple8 peripheral and tie a two line LCD display to it. Then you could write your own custom driver for it so that your applications would just write to something like /dev/dp0/s06_myLCD/text to send text to the LCD display.

Select and Install Your Peripherals

You can specify which nine peripherals to put into your version of DPCore. Backend software at Demand Peripherals uses your list of peripherals to select which FPGA files to include in the Verilog build of DPCore. This build service is included in the price of the BaseBoard4 hardware. You can request as many DPCore builds as you want.

You specify your list of peripherals by visiting the DP web site and selecting "Support" followed by "Build Your FPGA Image". Select enough peripherals to fill all nine slots. Accept the license and tell us where to email the DPCore file. For this tutorial you can use the following command to get a pre-built version:

```
cd /tmp
wget http://www.linuxoys.com/chumbot/DPCore.bin
```

At each power cycle you need to do the following to load DPCore onto the BaseBoard4:

```
stty -opost < /dev/ttyUSB0
cat DPCore.bin > /dev/ttyUSB0
```

Dpdaemon Kernel Driver Installation

Enter the following commands to download, build, and install the kernel driver components.

```
wget http://www.demandperipherals.com/downloads/proxy-1.0.tar.gz
tar -xzf proxy-1.0.tar.gz
cd proxy-1.0
make
sudo make install
sudo depmod -a
sudo modprobe fanout           # do this at each boot
sudo modprobe proxy           # do this at each boot
```

Dpdaemon User-Space Driver Installation

When you have the kernel modules loaded and DPCore loaded into the BaseBoard4 you can enter the following commands to download, build, and start dpdaemon.

```
cd /tmp
wget http://www.demandperipherals.com/downloads/dpdaemon-0.9.0.tar.gz
tar -xzf dpdaemon-0.9.0.tar.gz
cd dpdaemon-0.9.0
make
sudo LD_LIBRARY_PATH=/tmp/dpdaemon-0.9.0/lib ./bin/dpdaemon /dev/ttyUSB0
ls /dev/dp0
echo cc > /dev/dp0/s01_bb4io/leds
cat /dev/dp0/s01_bb4io/buttons &           # press some buttons
cat /dev/dp0/s10_servo4/README
echo 2000000,0 > /dev/dp0/s10_servo4/servo0 # for a servo on slot 10
echo 1000000,0 > /dev/dp0/s10_servo4/servo0
```

DP Direction

Demand Peripherals is dedicated to building robotic and industrial automation peripherals that include Linux drivers. Pin Peripherals (so named because they end at the FPGA pins) are our current offering and we will be expanding the selection and type of Pin Peripherals. We will also offer Card Peripheral which will contain the analog or drive electronics for the peripheral. Photo 4 shows a selection of card peripherals which we have at the prototype stage.

Please use the forums on the company web site to help us select and define new peripherals to bring to market.

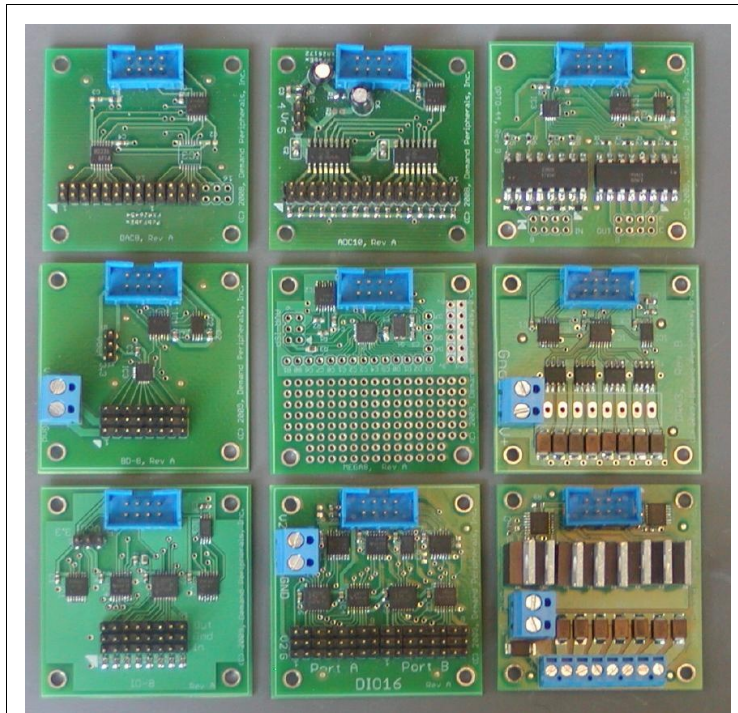


Photo 4: Prototype Card Peripherals