Hacking the CBC Botball Controller:
Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked
Part 2
Jeremy Rand, Matt Thompson, and Braden McDorman
Norman High School, Nease High School, Norman High School
jeremy@asofok.org, matthewbot@gmail.com, bmcdorman@gmail.com

Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked Part 2

11. Welcome to Part 2!

Thanks for sticking with us for Part 2! Here, we'll give more information on hacking the CBC than we could fit into the first paper. But first, the obligatory disclaimer:

DISCLAIMER: Hacking the CBC's file system in any way can brick your CBC if done improperly. While we have tried to be as accurate as possible, we are not responsible for any damage that may result from the use of this information. As with any Linux system that gives you root access, you should not mess with files whose purpose you do not understand. KIPR will most likely charge you money to repair a CBC which you brick through improper use of these methods, and don't expect us to fix your CBC for you either. If this scares you, that should be a hint that CBC hacking is probably not for you.

12. Network Program Downloads

Running shell commands over the network is fun, but what about something more practical? The TAR command, combined with pipes and SSH, can be used to download KISS-C programs (or any other files) to the CBC, without requiring the USB cable. Why is this such a nice feature? The USB download cable is a serial port with an FTDI USB-to-Serial chip. This chip is limited to 921.6 kbaud, or 115.2 kbyte/s. And the actual serial baud rate that the chip is configured with is far less than that. In contrast, we routinely get 500 kbyte/s through Wi-Fi -- and of course, Wi-Fi doesn't need a cable. Here are the files you'll need to create on your PC:

blank/blank.c:

```
int main()
{
	printf("Blank program.\n");
	printf("Press B to exit....\n");
	while(! b_button()) msleep(20);
	return(0);
}
```

```
new:

#! /bin/bash
if [ $# -eq 1 ]
then
    SOURCE=blank
fi
if [ $# -eq 2 ]
then
    SOURCE=$2
fi
cp -R $SOURCE $1
mv $1/$SOURCE.c $1/$1.c

download:

#! /bin/bash
CBCIP=`cat cbcip`
tar -czf - $1/* --exclude=".*" --exclude="*~" -h -p | ssh root@$CBCIP "rm -rf
/mnt/user/code/$1/* > /dev/null; rmdir /mnt/user/code/$1/ > /dev/null; tar -C
/mnt/user/code/ -xzf - -p; /mnt/kiss/usercode/compile /mnt/user/code/$1/$1.c"
```

To create a new program, type `./new new_project_name` on your PC. It will generate the new project as a copy of the `blank` program. Or, if you want to copy a program named `custom_program` instead, type `./new new_project_name custom_program`. Unlike with the KISS-C IDE, you can add any file type to a project, e.g. JPEG images or MP3 songs (both of which we will elaborate on later in this paper); you are not restricted to .c files.

To download your program to your CBC, type `./download project_name`. This extremely ugly script uses the TAR command to compress your project into one file, sends it over SSH to the CBC, and has the CBC extract the files and put them in the proper folder.

13. Copying Files with SCP

Using TAR with SSH to download programs works, but it's unwieldy for copying single files to your CBC, or copying files from your CBC to your PC. SCP (Secure Copy) handles this well. Use these shell scripts from your PC:

```
scpto:

#! /bin/bash
CBCIP=`cat cbcip`
scp $1 root@$CBCIP:$2

scpfrom:

#! /bin/bash
CBCIP=`cat cbcip`
scp root@$CBCIP:$1 $2
```

`scpto` is used to download to the CBC; `scpfrom` is used to upload from the CBC. The arguments are the start path and the destination path. For example, if you want to copy `/cygdrive/C/file.bin` on your PC to `/mnt/user/file.bin` on your CBC, you would use `./scpto /cygdrive/C/file.bin /mnt/user/file.bin` . To copy from your CBC to your PC, you would use `./scpfrom /mnt/user/file.bin /cygdrive/C/file.bin` .

The `/mnt/usb` and `/mnt/kiss` partitions are read-only by default. If you want to write to them with either SCP or any other means, you will need to remount them using something like this shell command on the CBC:

```
mount -n -o remount,rw /mnt/kiss
```

14. Debugging with GDB

Interactive C was slow and inefficient, but it did have a major advantage over standard C: the ability to interactively type a command, and have it executed immediately. This made debugging much easier, since it was unnecessary to write complex debugging programs just to see if a function returned the value or had the effect that it was supposed to. KISS-C does not by default support such interaction, but it is possible to get similar (though not identical) interaction capabilities through an SSH connection, using GDB.

GDB is the GNU Debugger, and it comes preloaded on the CBC. To debug your programs, you will have to modify the compile script. These modifications are included in our installer.

Once the scripts are present on the CBC, open an SSH session to your CBC, navigate to the `/mnt/user/nhs` folder, and type `./debug example`, where `example` is the name of your program (as you specified when downloading it to the CBC). You will be given a GDB prompt. Run your program by typing `run`. To pause the program so that you can interact, hit `Ctrl+Z`. You will be back at the GDB prompt. To evaluate an expression, type `call digital(0)`, where `digital(0)` is the expression you wish to evaluate. When you're done interacting, type `continue`, and your program will resume where it left off. To exit completely, type `quit`.

If you just want to test the CBC libs (e.g. to calibrate servo values), the `blank.c` program listed above is a good program to debug for this purpose.

A few differences between GDB and IC: GDB's interaction commands execute while the program is paused, not simultaneously in another thread like IC. We were unable to get `#define` macros to work with GDB, but that may just be due to our incompetence. Also, when you pause the program, cbcui and the Breakout Board will continue to run, so motors will continue to move. GDB is much more powerful than IC; see the GDB manual [4] for a full reference.

15. Remote Desktop with VNC

One important decision KIPR made when designing the CBC was to use Qt for the Graphical User Interface (GUI). This allows us to view and control the CBC screen over a network

connection using a VNC plugin for Qt. Virtual Network Computing (VNC) is software which works similarly to Windows XP's Remote Desktop feature. VNC could be used for several purposes on the CBC. One possibility is interaction with the CBC from a computer using the CBC's standard GUI. Another use is getting high-quality screenshots or videos of your CBC screen, e.g to demonstrate something. There is one major caveat when using Qt VNC: Qt VNC disables the CBC screen. This may present problems for interoperability between multiple people testing a single robot.

To start a VNC session, run the following script from a Cygwin or bash prompt on your PC:

`vnc:`

```
#! /bin/bash
CBCIP=`cat cbcip`
ssh root@$CBCIP "/mnt/user/nhs/vnc"
```

Make sure to append an ampersand when calling this script, e.g. `./vnc &` . This ensures that you will be able to continue typing new commands; otherwise, you'll get a bunch of cbcui output endlessly spewed at you.

Once you've run the script, point your favorite VNC client (we tested with RealVNC on Windows) to your CBC's IP address. You'll see your CBC's screen on your PC! Feel free to click around; all the menus should work just like on the actual CBC.

16. Camera Driver Settings

The default CBC camera driver settings do not produce an optimal framerate. This is because they rely on auto-exposure control, and they typically make the exposure time much too long. This results in a framerate of about 3fps, with significant motion blur.

It is, however, very easy to change these settings to produce a high framerate and little to no motion blur, with no noticible (to us) drop in picture quality. Simply execute these commands from your CBC program:

```
system("echo 0 > /sys/class/video4linux/video0/auto_exposure");
system("echo 0 > /sys/class/video4linux/video0/exposure");
```

This code disables auto-exposure control, and forces exposure to the lowest allowable value. We measure around 11-12fps with these settings.

You can also change the brightness and contrast of the camera similarly:

```
system("echo 0 > /sys/class/video4linux/video0/brightness"); // minimum
brightness
system("echo ffff > /sys/class/video4linux/video0/contrast"); // maximum
contrast
```

The values are in hexadecimal, and seem to range from `0` to `ffff`.

There are a few other values in there too, such as `sharpness` and `rgb_gain`, which probably aren't particularly useful for Botball but are mentioned here for completeness.

17. Binary Caching

Having multiple programs on a CBC is nice, but recompiling every time you want to switch programs is a waste of time. A solution is to modify the compile script to cache each program's binary, so that a program only has to be compiled the first time it is run. Our installer adds this capability. The modified code is quite mundane, so we won't waste your time explaining it here. If you're interested in how we implemented it, take a look at the compile script in our installer.

Our script also caches `robot.c` files from USB flash drives, under the program name `usb_stick_cached`. So long as your program doesn't access files on the USB drive, this allows you to compile a program from a USB drive, and then run the program with the USB stick removed, even if you compile other programs. (Only one USB drive program can be cached at a time.)

18. Fixing the Shutdown Button

The Stop and Shutdown buttons in the CBC GUI don't work particularly well. They kill your program's process, but servos stay on, and the Shutdown button doesn't even turn off motors. In addition, the motor PID gains don't get reset, so if one of your programs messed with them, any subsequent programs which expect the defaults will encounter issues. We partially fixed this issue with a rather crude hack that nevertheless works acceptably. We created a KISS-C program called `stop.c`, which gets installed and compiled by the auto-installer. When `stop.c` executes, all motors and servos are disabled, and the PID gains are reset to their default. We then modified the stop and shutdown scripts to run the stop program. Unfortunately, we found that the current CBC firmware doesn't actually use the stop script; the Stop button functionality is built into cbcui. So for now, this only fixes the Shutdown button. We hope to find a way to fix the Stop button as well; we'll make an announcement if we find a way to fix it. In the interim, if a program leaves your servos turned on, you can manually run the stop program from the Programs menu. Not only does this fix issues with programs leaving servos on, but it even stops motors or servos that have been turned on by interaction through GDB.

19. Programming in C++

Sometimes C's feature set is enough. Other times, the object-oriented features of C++ may be useful. `g++`, the GNU C++ compiler, was present on the 0.6 CBC firmware, but was removed from subsequent versions to save space and decrease the required time to do a firmware update. Our installer restores `g++` to the CBC, and also adds support for C++ to the compile script. This was mostly grunt work (replace `gcc` with `g++`); the only particularly interesting parts of this hack are that we had to restore various library files (e.g. `libstdc++.a`) as well as use the `g++` flag `-fno-exceptions` . We're not entirely sure why disabling exceptions is required, but it seems to fix a lot of compile errors that we had with syntactically valid C++ code.

To use C++ with our modified compile script, give the main source code file the `.cpp` or `.cxx` extension rather than `.c`. The compile script takes care of the rest.

20. JPEG and PNG Overlays

JPEG and PNG images can be displayed by the Chumby as an overlay. To display an image, you might use this code in your CBC program:

```
system("imgtool --mode=draw --fb=1 /www/images/chumby_logo.png"); // Loads
the image into the overlay framebuffer
system("echo 1 > /proc/driver/imxfb/enable"); // enable display of the
framebuffer
system("echo 0x77 > /proc/driver/imxfb/alpha"); // set opacity to 0x77,
opaque is 0xFF
```

This can come in handy if you want to have a diagram for setting up your bots but don't want to carry it around on a piece of paper. (Well, maybe not that useful. But fun!)

21. Playing MP3 Files

Just about every team has tried to make their bot play music during the game rounds back when the Handy Board and XBC were in use. Some teams just used tone(). A couple teams used converted .wav files on the XBC (with hacked firmware). But so far, no one has made music play on a CBC. Until now. Playing MP3 files on a CBC is quite easy. First, enable the music player daemon with these CBC commands:

```
system("btplayd > /dev/null &");
msleep(1000); // Wait for it to get ready
```

This should only have to be done once until you reboot your CBC. Even so, we recommend putting it at the start of all programs that use MP3 playback.

Then, you can play MP3 files using this CBC command: (And yes, we really did test this code using the Bill Nye theme.)

```
system("btplay /mnt/user/nhs/billnye.mp3 > /dev/null &");
```

We don't know of an easy way to play multiple files simultaneously; playing a file will stop any currently playing file. It should be noted that playing MP3 files will only work if the KISS-C `beep()` function works properly for you. It seems that the CBC's speaker works rather randomly, so if `beep()` doesn't produce audio, `btplay` won't either. We will make an announcement if we find more information on making the speaker more reliable.

22. Dimming the Backlight

If you think the screen is too bright, or you want to save battery life while your robot is running, it is possible to adjust the brightness of the Chumby screen's backlight. Use this CBC command:

```
system("echo 0x7777 > /proc/sys/sense1/brightness"); // sets brightness to
0x7777; default is the max, 0xffff
```

23. Conclusion

We hope you enjoyed and learned something from this two-part paper. CBC hacking is still fairly young (let's face it, the controller is less than 5 months old as of this writing), so we need more people working on it. If you've made an awesome hack on the CBC, we'd like to hear about it. We can be found at the Botballer's Chat and Botball Forums on the Botball Community website [5].

Happy Hacking!

References

[4] Free Software Foundation, Inc. Debugging with GDB. http://sourceware.org/gdb/current/onlinedocs/gdb_toc.html , 2009.
[5] Botball Youth Advisory Council. Botball Community. http://community.botball.org/, 2009.