GP2X Demo Development
Part 1

Code & Concept
Dzz

Copy & Paste
synkro

Last Revision: April 26, 2006

Motivation

As the sponsor of the GP2X demo competition, I am ineligible to win a prize, but just for fun I will develop my own demo over the next few months anyway. I'll document my progress in a series of articles, in case the things I discover will be of use to other developers.

Looking at the task, I definitely want to aim for a 64k demo. Preferably, it would be even smaller than that, because striving for tininess is a cool thing to do in a demo. There's nothing wrong with using conventional methods to make a bigger demo, of course, and I hope a lot of people give that a try! Just my personal interest. I'm not elite enough (yet) to contemplate something like a 4k demo on the GP2X, so I'll be satisfied with simply keeping it as small as I can. – Dzz, Apr 11 2006

1 Introduction

This article series will introduce various ways to access the GP2X in a low level fashion. Those techniques are required by anyone who wants to push the GP2X to its limits. We can't talk about everything in every detail, if you have ideas, questions or some random rants than drop us a line[1]. So, what's available on the machine that might be fun to use?

- Two 200MHz processors! That's a lot of power and it would be a shame to not put it all to good use.
- Graphical display. By default, 16 bits of color... but 24 bits could conceivably be available. It might be interesting to play around with that a little bit.
- Hardware blitter. It could be fun to play around with the capabilities of the blitter.
- Video decoder. Can any good use be made of the video decoding hardware?
- Audio. This looks fairly routine but it will at least have to be used properly.
- TV-Out! It can't be easier to show off your demo on the next party on the big screen.

2 Keep it small

First, though, some thoughts about program size. Even in times of multi-giga-byte machines and SD-cards 64kB is still the size to keep. No better way to show off your skills when you keep it small. There are two possible ways to go about keeping the size small:

1. Dynamically link to features that can be found on the GP2X itself. The advantage of this approach is that we can leverage some big pieces of functionality in this way. A quick look at libs folder on the GP2X shows the basic standard libraries plus compression, ogg decoding, the extensive SDL library with sound and font support, and PNG file support[2]. That's a lot of stuff!
2. Do not use any external libraries – instead, build all functionality into the program itself.

Approach (1) has some huge advantages obviously, but what are the disadvantages? The only one that springs to mind is that we'd have to use the particular versions of the libraries that exist on the GP2X firmware releases. So we'd have to research what versions are available in the different firmware builds and live with that. Another disadvantage of dynamically linking is that we can only use upto a certain version of GCC. The libraries on the GP2X were built using GCC 2.95, so we can probably only go upto GCC 3.4. For static builds, we can use any version. GCC 4 seems to produce the fastest executables - in our experience the executables are approximately twice as fast as those created by 2.95.

But approach (2) is more interesting and might have the advantage of being more technically difficult. Also, if we get married to the SDL that comes with the firmware, it could be rather difficult to do things like use the hardware

[1] `www.gp32x.com`, `wiki.gp2x.org` or #gp2xdev (EFNET) are the places to be for any interested coder.

[2] It's a common procedure among devs not to use dynamic linking for normal applications on the GP2X.

blitter if we want to. Plus, we've never thought about the issues involved with having no reliance on any library before which means there's a bunch of fun (?) stuff to learn.

3 Let's roll

So, having cast aside the stifling comfort of libgcc, libc, libm, libsdl, and other trifles, it's time to answer this question: How small of a Hello-World-program can we make? It should print out `Hello, world!` to the serial port and cleanly exit back to the GP2X menu.

First, a blank program that doesn't do anything. To avoid fancy linking, we'll use `gp2x-ld`[3] directly and define our own entry point to stop it from whining about the `main`-function. So here is our entire first program:

```
void entry()
{
}
```

And to compile it:

```
gp2x-gcc -Wall -Werror -O2 -c entry.c
gp2x-ld -e entry entry.o
  -static -s -o demo.gpe
```

Sure enough, it produces an actual program, which crashes the GP2X if we run it. Not to worry about that for now, we need to be friendly and exit in one of the proper ways.

Next, to get some output. This will come in handy later for debugging purposes as the demo gets built. To produce output, we'll write the string to file descriptor 1 (standard output), which will send it out the serial port where we can see it. So the linux call we want to make is:

```
write(1, "Hello, world!\n",
      strlen("Hello, world!\n"));
```

After much googling, experimenting, and fits of foul language, we worked out this routines.

```
void print(char *string)
{
  int length;
  length = getStringLength(string);

  asm volatile
  (
    "mov r0, #1\n"      // stdout
    "mov r1, %0\n"      // the string
    "mov r2, %1\n"      // the length
    "swi #0x900004\n"   // write
    :  // no output
    : "r"(string), "r"(length)
    : "r0", "r1", "r2"
  );
}
```

This illustrates the general method for making system calls, which turns out to be pretty easy. `getStringLength` is a simple function that returns the length of a string.

```
int getStringLength(char *string)
{
  unsigned int i = 0;
  while(string[i] != 0)
  {
    i++;
  }
  return i;
}
```

Similarly, to restart the menu on exit:

```
void restartMenu(void)
{
  char *pszMenuDir = "/usr/gp2x";
  char *pszMenuCmd = "/usr/gp2x/gp2xmenu";

  asm volatile
  (
    "mov r0, %0\n"    // directory
    "swi #0x90000C\n" // chdir

    "mov r0, %1\n"    // program to execute
    "mov r1, #0\n"    // arg 2 = NULL
    "mov r2, #0\n"    // arg 3 = NULL
    "swi #0x90000B\n" // execve
    :   // no output
    : "r"(pszMenuDir), "r"(pszMenuCmd)
    : "r0", "r1", "r2"
  );
}
```

The `restartMenu` function switches to the given directory and executes the GP2X menu. This leaves me with just the following *main* function (which we call `entry` just for fun):

```
void entry(void)
{
  print("Hello, world!\n");
  restartMenu();
}
```

It works! Size so far of demo.gpe: 652 Bytes. In the next installment we'll look into getting access to the GP2X screen so we can show off my mad grafix skillz.

[3] We will refer to all GP2X-specific commands and programs with the prefix *gp2x*. This might differ from the names of the tools you installed.