Can Raspberry Pi reliably bit bang a 9600 baud serial and is there example code?

Asked 11 years, 6 months ago Modified 7 years, 3 months ago Viewed 17k times

30

I am wondering how feasible it is to use bitbanging to drive a 9600 baud serial via the GPIO pins on the Raspberry Pi.

Obviously, Linux is not a terribly good platform for bitbanging, as there is a large number of drivers and other interrupts which may block the CPU for extended periods of time (1-10ms). However, the situation has gotten a lot better recently, and some pre-emption is now regularily enabled in kernels. I also suspect a real-time patched kernel can easily be used on the Raspberry Pi, and the connected hardware and drivers can be selected carefully.

My standard for reliability is that it should stay within normal 9600 baud serial tolerances most of the time. I am currently unsure how many errors are tolerable in practice, but there are retransmissions and acknowledgements in the protocol, so it is atleast somewhat tolerant.

So, my questions are:

- Can userland software reliably bit-bang 9600 baud speed?
- Does this need a real-time patched kernel?
- How many errors should I expect in general?

Also, is there any example code doing serial via GPIO bit banging? I know of Arduino SoftSerial, but that's about it.

Just in case, to limit the question: this is not a standard serial interface and the UART of Raspberry Pi is already used for another serial link. I do not wish for answers that suggest external hardware (such as Gertboard or Arduino) nor answers that depend on the UART.

gpio serial bit-bang

Share Follow

asked Sep 24, 2012 at 16:52

Nakedible

1,521 1 13 18

1 +1 Good question. I would also be interested in the answer to this. However, the question of `reliability` could depend on the action and expectations. – Jivings Sep 24, 2012 at 18:16

2 RS-232 needs negative voltages, but this serial link does not use RS-232 voltages. I don't know but I believe that RPi requires the standard TTL to RS-232 converter, such as MAX232 or any of the dozens of choices. However, just to be clear, this isn't relevant to my question, I just wanted to answer. – Nakedible Sep 24, 2012 at 18:55

4 There is a second UART on GPIO pins 14&15 as well. It's not as full featured as the main one though – John La Rooy Sep 25, 2012 at 1:18

1 en.wikipedia.org/wiki/Bit_banging – Nakedible Sep 25, 2012 at 20:19

1 @Nakedible could you please share your work? I have a similiar problem and would like to see how you did a second serial via spi. thanks MSP – user7406 May 14, 2013 at 19:02

3 Answers

Sorted by: Highest score (default)

14

I finally solved this, but in a pretty unorthodox way. I abandoned bit-banging as too unreliable and tried to find other solutions which would allow me to the same thing without adding more hardware. I was considering writing a kernel driver, which would trigger an interrupt on GPIO and then reconfigure the pin to be SPI and use SPI to read an entire data byte - but then I got a better idea.

I use SPI to sample the lines at 20x the baud rate. I ignore the SCLK and SS pins entirely, connect the RX line to MISO and TX line to MOSI. This gives me a (1-bit) oscilloscope-like view in to the RX line and clearly see the bits being transmitted in the serial line:

```
00 00 00 00 00 00
00 00 00 00 01 FF
FF FF FF FF 00 00
01 FF FF FF FF FF
FF FF E0 00 00 00
00 00 07 FF FF FF
FF FF
```

From this, it is a simple matter of coding to figure out the correct positions from which to sample the actual data bits. The sending side is equally trivial, I just need to convert every byte in to a long stream of bits with the start bit and stop bit included.

The reason this works better than bit-banging is that SPI has its own clock that does not freeze with the kernel and the SPI send and receive lines have a 16-byte FIFO for transfer which are also independent from kernel freezes. For 9600 baud, I am using a 250kHz SPI clock and that means that I can sleep for even a millisecond between filling and draining the FIFOs without any transmission errors. However, to err on the safe side, I am using 300 µs sleeps. I briefly tested on how far I could push this and atleast a 2MHz SPI clock was still usable so this solution scales to higher baud rates as well.

The one ugly part of this solution is that the kernel SPI driver does not support such a streaming bit transfer. This means I can not do this by writing my own kernel module using the kernel SPI driver, and I also can not do it by using /dev/sdidev0.0 from the user land. However, on the Raspberry Pi, the SPI and other peripherals are accessible directly from userland by mmap():n /dev/mem, bypassing the kernel control entirely. I am not terribly happy with this, but it works perfectly and it gives the added benefit that segmentation faults in the userland can not crash the kernel (unless messing with the other peripherals by accident). As for CPU use, 300 µs sleeps seem to give me about 7% CPU use constantly, but my code is very unoptimal. Increasing the sleep duration obviously lowers the CPU usage directly.

Edit: Forgot to mention, I used the nice bcm2835 library to control the SPI from userland, extending it where necessary.

So, to summarize: **I can reliably transmit and receive on a 9600 baud serial link entirely from userland by directly using the SPI chip via /dev/mem at 250kHz on the Raspberry Pi.**

Share Follow

edited Sep 29, 2012 at 15:14

answered Sep 29, 2012 at 8:40

Nakedible
1,521 1 13 18

@Nakedible--Can you elaborate a bit or provide links on the mmap() part. I am working on the same thing. – Jay K May 17, 2013 at 4:47

Are you using the SPI hardware to do the transmit as well? – Cheetah Jun 6, 2013 at 23:34

Yes, transmitting works as well. – Nakedible Jun 17, 2013 at 8:26

3 Can you please give a step-by-step instructions how you can sent and receive code, and share any modified packages if you use something that you patched yourself...TIA! – valentt Apr 19, 2014 at 17:35

10

It would seem that atleast without the real-time patches (CONFIG_PREEMPT_RT), the Raspberry Pi cannot reliably bit-bang a 9600 baud serial.

I used a simple latency tester which configured all the linux side things optimally (sched_fifo, priority 99, cpu_dma_latench 0us, mlockall). I tried sleeping for 100 µsec (roughly 9600 baud) and checking the latency overruns on a quiet system for 2 minutes. The results were:

Min: 12 µsec Avg: 24 µsec Max: 282 µsec

This seemed like the common result. The max varied in slower measurements between 100 µsec and 300 µsec. I also checked the distribution and it seems that the vast majority are in the 24 µsec range. There only a few that go above 50 µsec, but there are almost always some. There are also sometimes huge latencies, such as 4000 µsec, but these are uncommon enough to be ignored, atleast for now.

I guess the maximum latencies should be below 50 µsec for 9600 baud to not get errors and any latency of over 100 µsec causes completely missing a bit in transmission or reception.

This is all without even touching the GPIO pins yet. Since I could not get a clean run even with just 2 seconds, it seems safe to say that without real-time patches the Raspberry Pi can not bit bang a 9600 baud serial link without generating errors for any serious amount of time.

I will be testing the real-time patches later if I get the time.

(Tool used: http://git.kernel.org/?p=linux/kernel/git/clrkwllms/rt-tests.git;a=summary)

Update: The RPi kernel hangs at boot without detecting SD card if compiled with the CONFIG_PREEMPT_RT patch set. It might be a simple thing to fix, but seeing the messy diffs in RPi source, I think I want to wait until more of it is in the mainline kernel.

So, testing this is too difficult, and I'm abandoning it.

Share Follow

edited Sep 25, 2012 at 22:23

answered Sep 25, 2012 at 9:55

Nakedible
1,521 1 13 18

0

You don't need to bit bang. You could set up an user land interrupt in the rx gpio to detect the falling of the start bit. then set a time interrupt to sample in the middle of the bits. A kernel module doing that is feasible.

Share Follow

answered Jul 9, 2015 at 18:49

Lalo UY

9 1

But that's still bit banging. Indeed that's the way you usually do bit banging. – Philippos Oct 24, 2017 at 13:55