Bit- Banging in Galileo with WS2812 Report

This report summarizes our attempts on using the bit banging technique to send the data to a WS2812 device. The WS2812 device considers the signal it receives as “1” if the signal is HIGH for 0.7 us and LOW for 0.6 us. It is a “0” if the signal is HIGH for 0.35 us and LOW for 0.8 us.

We initially used gpio_set_value_cansleep() function to set the pin high or low. We checked the time taken for the function and saw that it took about 6 microseconds. We used the time stamp counter to measure the time taken for it. So we implemented a memory mapped IO writing to speed up the process thus directly writing to the register. The time taken for this was approximately 1.5 us. We believe this time taken includes the overhead from accessing the TSC register and storing into the variable. But this did help us to come to conclusion that memory mapped IO is way faster almost 4 times.

With a faster IO write mechanism we sent the information to the WS2812 bit by bit using ndelay. While we were able to light up the LED we were not able to switch it off thus indicating that we were not able to send bit 0. we probed the ndelay () function with 3 mechanisms to find if it generated the the right amount of delay. We first used RDTSC to find the number of ticks and used frequency to find the delay. Just to be sure we used other mechanisms as well. We used the kernel timer ktime_get() and used it to measure the time difference of the delay generated. We also used the high resolution timer to measure the time. We used the 3 mechanisms to see if a 350 nano second delay can be generated. The RDTSC and Ktime indicated it takes

about .8 microseconds for a 350 nano second delay and hr timer indicates 4 microsecond delay for the same. We believe the HR timer will have a little extra over head compared to the other mechanisms. RDTSC is assembly and it should take minimum overhead. So even with a generous margin for overhead error we think that the ndelay mechanism does not provided the delay accurately and produces a delay much more than the intended.

We followed the same procedure for hrtimer. We used Kernel timer and RDTSC to measure. We tried to generate a delay of 350 nanoseconds. The method that used rdtsc gave absurd results. We attribute this to the fact that the TSC is disabled when hrtimer is used. This can be seen over here.
https://lwn.net/Articles/209101/

For this reason, the recently-updated high-resolution timers and dynamic tick patch set includes a change which disables use of the TSC. It seems that the high-resolution timers and dynamic tick features are incompatible with the TSC - and that people configuring kernels must choose between the two. Since the TSC does have real performance benefits, disabling it has predictably made some people unhappy, to the point that some would prefer to see the timer patches remain out of the kernel for now.

In response to the objections, Ingo Molnar has explained things this way:

> We just observed that in the past 10 years no generally working TSC-based gettimeofday was written (and i wrote the first version of it for the Pentium, so the blame is on me too), and that we might be better off without it. If someone can pull off a working TSC-based gettimeofday() implementation then there's no objection from us.

The method that used ktime also gave high absurd results. We think this is due to the fact that ktime_get is used inside the hrtimer api. This might cause conflict and thereby resulting in wrong values.This we observed in the following link.

http://elixir.free-electrons.com/linux/v4.0/source/kernel/time/hrtimer.c

Our attempts in sending in zeros to the WS2812 using hrtimer failed and instead lit up the Leds.

To conclude we think both hrtimer and ndelay are not capable of producing a nanosecond delay as less as 350 nanoseconds. Both the timer mechanisms however are capable of producing accurate delays when it is in micro or milliseconds value. This incapability to produce small nano second delays makes it almost impossible to send in zero bit and as a result we are unable to turn off the LEDs. But we are able to turn on the LEDs. The program that we have provided shows that we can turn on the LEDs to all white one by one. Thus the timing for the WS2812 cannot be met.

COMPILATION INSTRUCTIONS

Hardware connections: Connect the vcc and ground of the WS2812 to the respective pins. Connect the data in of the WS2812 to IO1 of the board.

Software:
Navigate to directory containing the Makefile, timeprobe.c . type in:
make

You should be able to see the timprobe.ko file.

Transfer to the galileo board:
sudo scp timeprobe.ko root@192.168.1.5

Install the kernel module.
Insmod timeprobe.ko

OUTPUT:

On the hardware you should be able to see the lights light up one by one.

On the terminal you will see the following:

- time taken for gpio_set_value_can_sleep().
- time taken for memory mapped IO control.
- Time taken for ndelay(350) measured using RDTSC.
- Time taken for ndelay(350) measured using Ktimer.
- Time taken for ndelay(350) measured using hrtimer.
- Time taken for hrtimer(350) measured using RDTSC.
- Time taken for hrtimer(350) measured using ktimer.