Linux LED

LED with Xilinx Zynq

Linux on Zynq is of course usable for much more sophisticated tasks then controlling a LED, but the basic functions should also be available?

One might think. I did. But not always are the simple things so simple.

When I tried it the first the time, it ended up in an several days long story.

Step 1: Get some Zynq based board. On my desk was a ZYNQ XC7Z7045-FFG900 based board. That should do?

There is a LED on MIO7, so from hardware side, it is prepared. Getting Linux working, well that is also simple. Start Vivado, new project, build, export to SDK, add FSBL, build BOOT.BIN, adding u-boot.elf. Done. Works first try.

Step 2: Does the LED work too?

```
cd /sys/class
cd leds
```

and nothing, LED class is not there. Not compiled in. Bad luck.

I do not want to experiment with older Kernels on the 7045, so lets switch gears, I plug TE0720 onto TE0703 simple basis and start it with pre-historic 3.09 Kernel.

```
cd /sys/class
cd leds
ls
```

And failed again, there are no LEDs available. They are missing in device-tree. Let's add the on-board LED to the device-tree. This can be done with utility called DTC. To my big surprise this tool is not at all available for Windows PC. Help... A few hours later I have DTC executable that works on Windows. My coworker who made it possible sends win32 patches to the maintainers of DTC tool, and I add the DTC.EXE to our download area.

> ⚠ It is also of course possible to run DTC from Zynq Linux prompt as well, it seems to be available on all Zynq Linux versions. DTC.EXE is just a handy tool for windows host PC.

To de-compile binary device tree file to textual version we can create a single line batch file DECOMPILE.BAT with this line inside:

```
dtc -I dtb -O dts devicetree.dtb -o devicetree.dts
```

To compile textual device tree file back to binary blob, we an create a single line batch file COMPILE.BAT with this line inside:

```
dtc -I dts -O dtb devicetree.dts -o devicetree.dtb
```

> ⚠ It is better to keep the decompile.bat in different directory after first decompile, as some changes to the textual device tree file will get lost if we accidentally decompile the blob in the same directory where the modified DTS file is.

Now lets add the GPIO LED binding to the device tree. As I have started with decompiled device tree file, so all node labels are lost, and we have no easy means to reference the nodes by symbolic names. So we add back the label to PS7_GPIO node first.

So at the line:

```
  ps7-gpio@e000a000 {
```

We added the label and semicolon:

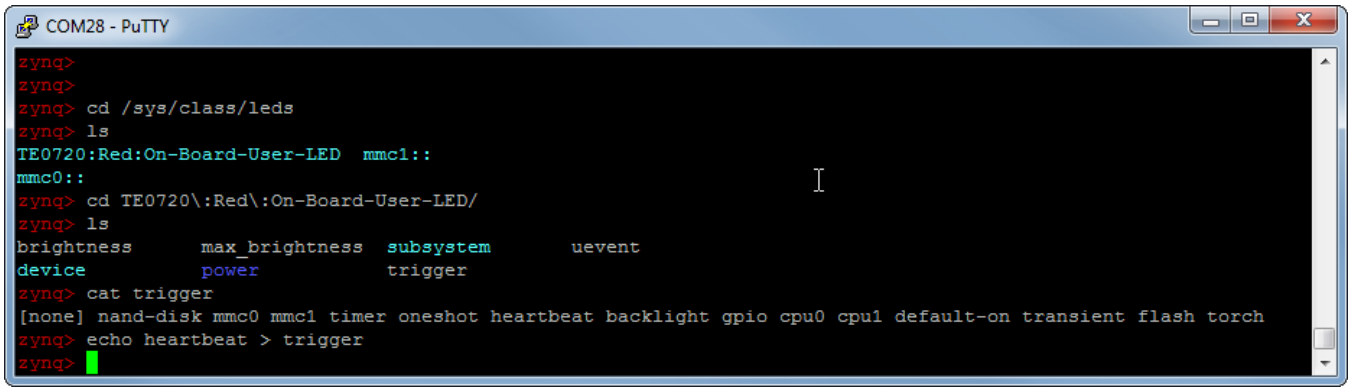
```
ps7_gpio_0: ps7-gpio@e000a000 {
```

Next, somewhere says after the memory node, we add the LEDS node that maps MIO7 to GPIO LED device like this

```
 memory@0 {
   device_type = "memory";
   reg = <0x0 0x10000000>;
 };

 /* We insert LED's node here */
 leds {
   compatible = "gpio-leds";
   red {
     label = "TE0720:Red:On-Board-User-LED";
     gpios = <&ps7_gpio_0 7 0>; /* Active High */
   };
 };
 /* End of LED's */
```

Thats all, now we invoke compile.bat and copy the devicetree.dtb file back the SD Card.



And now we have assigned Linux LED heartbeat to the Green on-board LED on TE0720.

Ups, and it works on 3.14 Kernel.

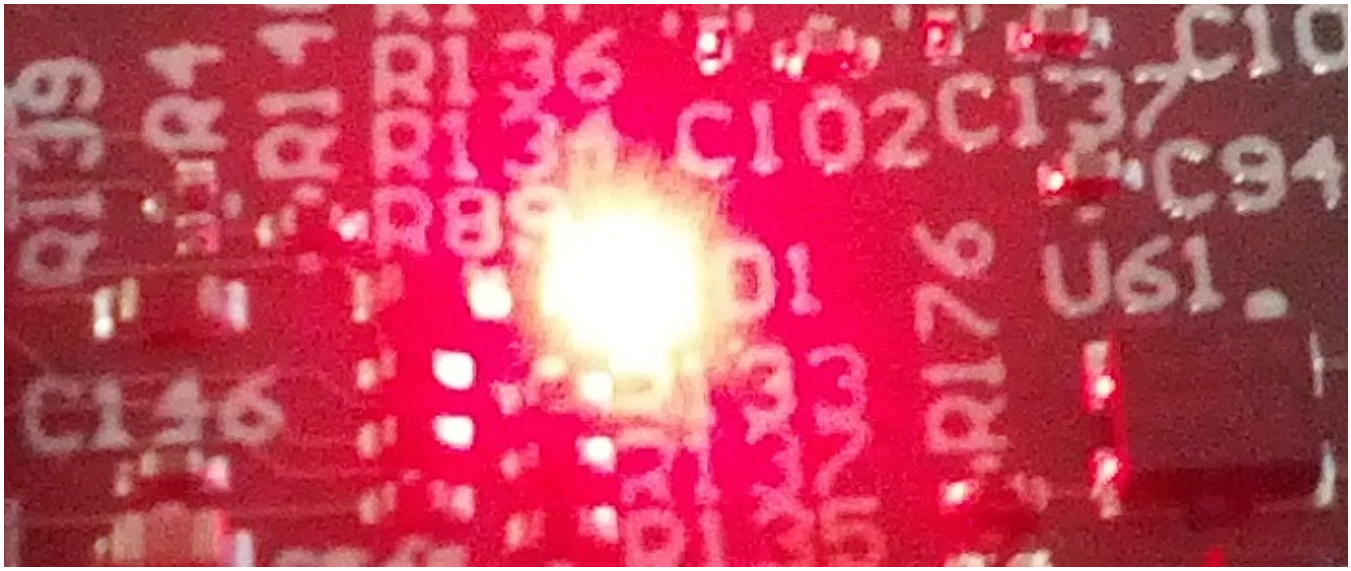
We can assign "events the LED" or just control the LED to be on or off. Please note that we can also control GPIO devices connected to I2C Bus as well (if they are I2C GPIO Class devices, or emulated I2C Device in CPLD connected to FPGA).

Why MIO7 you may ask? This is the most famous Zynq LED number ever seven: MIO7 is one of two MIO pins that are usable as OUTPUT Only and also in almost all Zynq designs MIO7 is in 3.3V MIO bank, so it is first candidate for a "User LED". In TE0720 MIO7 is connected to System Controller that as default connects to a LED. But the System Controller could also use the MIO7 as some signal from Zynq ARM Core, or route the MIO7 back to the Zynq PL Fabric, or use the MIO7 as FORCE reset or something else.

So what about the LED on the 7045 board? As I have not updated the Kernel there, the LED Class is not available, so I have to-do it the old fashion GPIO way:

```
cd /sys/class/gpio
echo 7 > export
cd gpio7
echo out > direction
echo 1 > value
```

And the LED on below 7045 is on as well...



But why was it needed to set MIO7 direction to out, if this pin is Output only? Well it is not usable as input, but Linux driver system does not know that it is output only and it listed with direction input initially.

Zedboard OOB Design

The SD Card images delivered with zedboard (and available as OOB SD Card images) include a kernel 3.09 with GPIO-LEDS enabled on MIO7 with default trigger set to MMC (SD Card) activity. Writing none to LED trigger would allow direct programmatic control over MIO7 LED on zedboard (labelled LD9!).

As newer Linux kernels for Xilinx seem to have the GPIO LED class not enabled by default it may not be possible to use Linux LED class on zedboard (if using standard kernels).

LED with RPi

The user LED on RPi is labelled ACT, and it is controllable directly using Linux LED subsystem, as default it is assigned to SD Card activity, that is why it is labelled "ACT".

First you nee to be root.

```
sudo -i<ENTER>
password root<ENTER>
```

set password for root and reboot and log in as root.

```
cd /sys/class/leds/led0
cat none > trigger
```

Now we are ready to control the user LED on Raspberry

```
cat 1 > brightness
cat 0 > brightness
cat heartbeat > trigger
```

The above 3 lines show how to control the LED on, off and set to blink..! It works that way at least with the NOOBS distribution.