fpga manager & device tree overlays

...

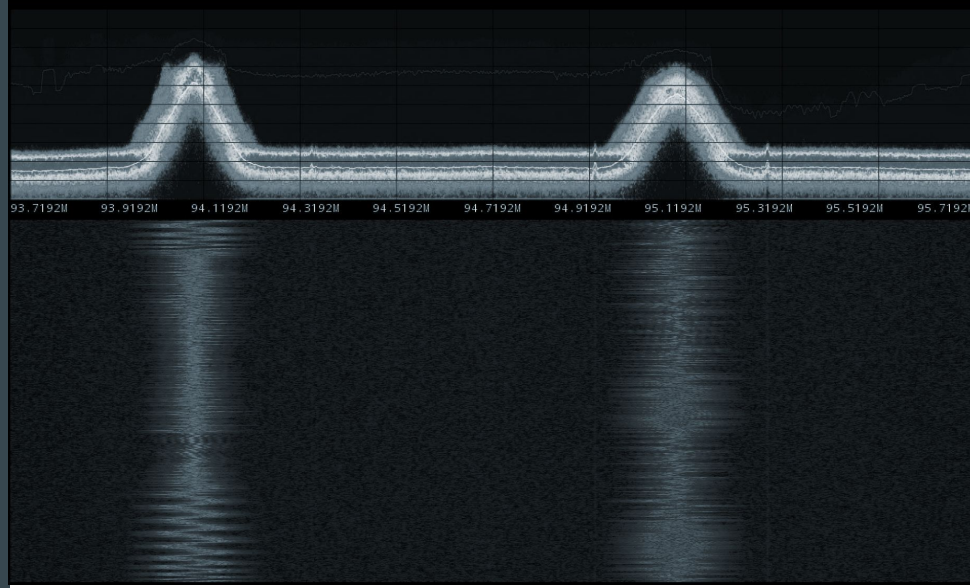
FOSDEM'16

moritz fischer

moritz.fischer@ettus.com

mfischer

embedded sdr

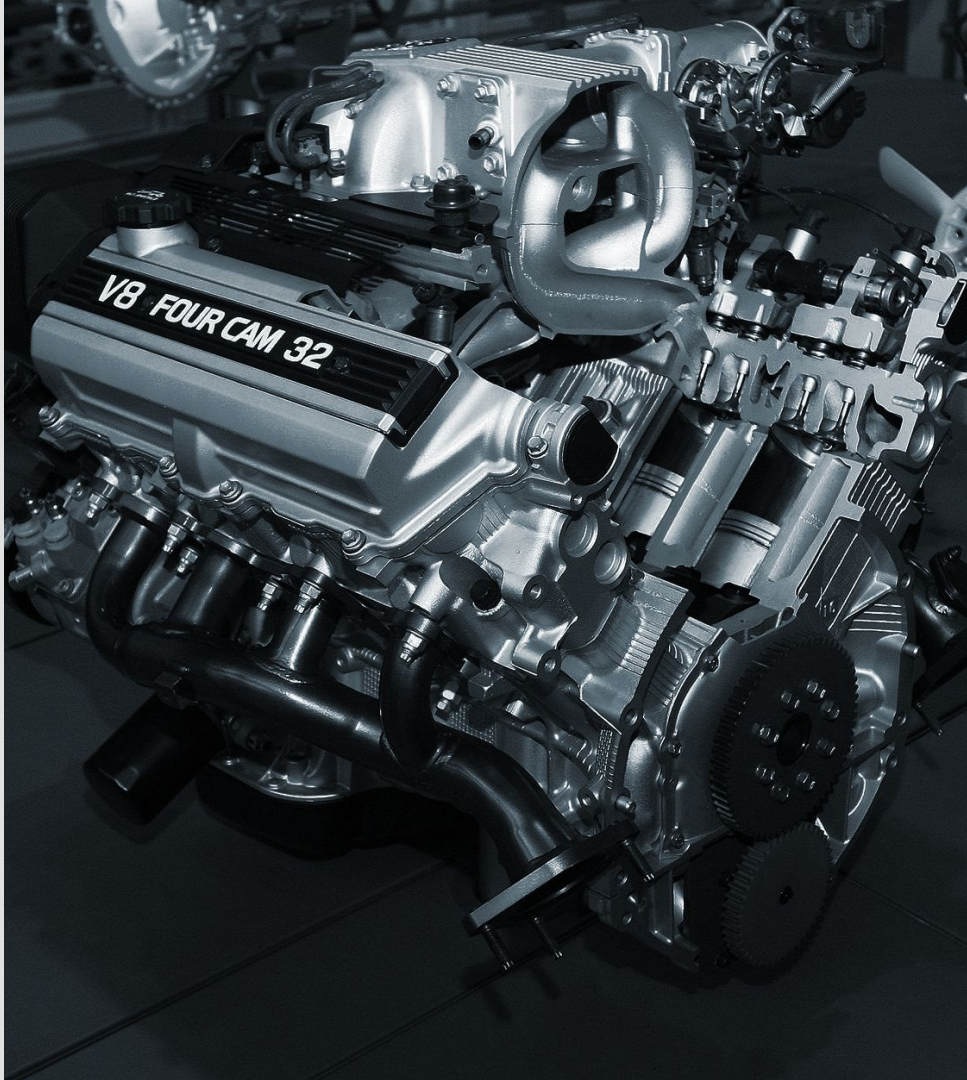


wtf?! what does that even mean embedded sdr?

come see other talks tomorrow @ sdr track

so why care about fpgas?

performance



so why care about fpgas?

reconfigurability

so why care about fpgas?

also, they're awesome

i won’t go into details of fpga design

bitstream (firmware) contains hardware behavior

so how to configure an fpga in a sane way?

let’s start off with a bit of history

vendor solutions

altera

```
$ cat design.rbf > /dev/fpga0
```

xilinx

```
$ cat design.bin > /dev/xdevcfg
```

what could possibly go wrong?

well … if you have more than one device implemented in the fpga

userland just goes ahead and reloads the fpga

you maybe have a kernel driver using fpga resources as well ...

ehrm … whoopsie

should the user really care what fpga is in the system?

what if you had more than one fpga?

even worse, hierarchy?

(i’m not making these up...)

partial reconfiguration anyone?

fpga manager is vendor neutral as part of linux 4.4 basic support for socfpga and zynq

api - driver ops

```
write_init() /* prepare fpga for reload */
write() /* reconfigure fpga */
write_complete() /* callback when done */
state() /* returns framework internal state */
fpga_remove() /* called when removed */
```

api usage (kernel)

```
/* get reference from device node */
struct fpga_manager *mgr = of_fpga_mgr_get(dn);
/* load bitstream via fw layer*/
fpga_mgr_firmware_load(mgr, flags, “fw.bin”);
/* drop reference */
fpga_mgr_put(mgr);
```

this covers the simple usecase: driver needs fpga bitstream loaded

but we can do better than that ...

let’s talk about device tree overlays

device tree describes hardware, but what if hardware changes?

device tree overlays allow us to add, remove, and modify nodes of the live tree

example to modify status property

– foo.dts (abbrev.)–

```
foo0: foo@0 {
    compatible = “linux,foo”;
    status = “disabled”;
};
```

– overlay.dts (abbrev.) –

```
fragment@0 {
    target = <&foo0>;
    __overlay__ {
        status = “okay”;
    };
};
```

example to add bar child

– foo.dts (abbrev.) –

```
foo0: foo@0 {
    compatible = “linux,foo”;
    [...]
};
```

– overlay.dts (abbrev.) –

```
fragment@0 {
    target = <&foo0>;
    __overlay__ {
        bar0: bar@42 {
            compatible = “linux,bar”;
        };
    };
};
```


YES YES
THATS MORE LIKE IT

seriously now, that's pretty close to what we want, right?

fpga area (still in dev)

so

DO NOT TRUST THE SLIDES

will look somewhat like this

– overlay.dts (abbrev.) –

```
fragment@0 {
    target = <&fpga_mgr0>;
    __overlay__ {
        area0: area@40000000 {
            compatible = “fpga-area”;
            firmware-name = “foo.bin”;

            c0: child@0 {
                compatible = “linux,foo”
            };

            c1: child@4 {
                compatible = “linux,bar”
            };
        };

    };
};
```

discussion still ongoing,
if you care about fpga
join the discussion on
lkml

some open issues, but seem mostly solvable

what if fpga is pass-through, i.e. soc spi routed through fabric out to a pin?

notifiers?

trying to let driver know device is gonna be gone for a bit

fw subsystem doesn’t support (yet) streaming fw for wimpy systems

buckle up …
demo time

if we got here, we're probably out of time…
questions?

thanks to these guys

alan tull - fpga mgr core, socfpga driver, reviews
gregkh - taking my patches
pantelis antoniou - dt overlays
michal simek - reviews, initial fpga mgr
josh cartwright - reviews

– foo.dts –

```
btn0: button@0 {
    compatible = “ettus,
e3x0-button”;
    status = “disabled”;
};
```

– overlay.dts (abbrev.) –

```
fragment@0 {
    target = <&btn0>;
    __overlay__ {
        status = “okay”;
    };
};
```

example to modify status property