NAME

fdt_pinctrl - helper functions for FDT pinmux controller drivers

SYNOPSIS

#include <dev/fdt/fdt_pinctrl.h>

int
fdt_pinctrl_configure(*device_t client, u_int index*);

int
fdt_pinctrl_configure_by_name(*device_t client, const char * name*);

int
fdt_pinctrl_register(*device_t pinctrl, const char *pinprop*);

int
fdt_pinctrl_configure_tree(*device_t pinctrl*);

DESCRIPTION

fdt_pinctrl(4) provides an API for manipulating I/O pin configurations on pinmux controllers and pinmux clients. On the controller side, the standard newbus probe and attach methods are implemented. As part of handling attach, it calls the **fdt_pinctrl_register**() function to register itself as a pinmux controller. Then **fdt_pinctrl_configure_tree**() is used to walk the device tree and configure pins specified by the pinctrl-0 property for all active devices. The driver also implements the **fdt_pinctrl_configure**() method, which allows client devices to change their pin configurations after startup. If a client device requires a pin configuration change at some point of its lifecycle, it uses the **fdt_pinctrl_configure**() or **fdt_pinctrl_configure_by_name**() functions.

fdt_pinctrl_configure() is used by client device *client* to request a pin configuration described by the pinctrl-N property with index *index*.

fdt_pinctrl_configure_by_name() is used by client device *client* to request the pin configuration with name *name*.

fdt_pinctrl_register() registers a pinctrl driver so that it can be used by other devices which call **fdt_pinctrl_configure**() or **fdt_pinctrl_configure_by_name**(). It also registers each child node of the pinctrl driver's node which contains a property with the name given in *pinprop*. If *pinprop* is NULL, every descendant node is registered. It is possible for the driver to register itself as a pinmux controller for more than one pin property type by calling **fdt_pinctrl_register**() multiple types.

fdt_pinctrl_configure_tree() walks through enabled devices in the device tree. If the pinctrl-0 property contains references to child nodes of the specified pinctrl device, their pins are configured.

EXAMPLES

```
static int
foo_configure_pins(device_t dev, phandle_t cfgxref)
{
        phandle_t cfgnode;
        uint32_t *pins, *functions;
        int npins, nfunctions;

        cfgnode = OF_node_from_xref(cfgxref);
        pins = NULL;
        npins = OF_getencprop_alloc_multi(cfgnode, "foo,pins", sizeof(*pins),
           (void **)&pins);
        functions = NULL;
        nfunctions = OF_getencprop_alloc_multi(cfgnode, "foo,functions",
           sizeof(*functions), (void **)&functions);
        ...
}

static int
foo_is_gpio(device_t dev, device_t gpiodev, bool *is_gpio)
{
        return (foo_is_pin_func_gpio(is_gpio));
}

static int
foo_set_flags(device_t dev, device_t gpiodev, uint32_t pin, uint32_t flags)
{
        int rv;

        rv = foo_is_pin_func_gpio(is_gpio);
        if (rv != 0)
                return (rv);
        foo_set_flags(pin, flags);
        return (0);
}

static int
```

```
foo_get_flags(device_t dev, device_t gpiodev, uint32_t pin, uint32_t *flags)
{
        int rv;

        rv = foo_is_pin_func_gpio(is_gpio);
        if (rv != 0)
                return (rv);
        foo_get_flags(pin, flags);
        return (0);
}

static int
foo_attach(device_t dev)
{
        ...

        fdt_pinctrl_register(dev, "foo,pins");
        /*
         * It is possible to register more than one pinprop handler
         */
        fdt_pinctrl_register(dev, "bar,pins");
        fdt_pinctrl_configure_tree(dev);

        return (0);
}

static device_method_t foo_methods[] = {
        ...

        /* fdt_pinctrl interface */
        DEVMETHOD(fdt_pinctrl_configure, foo_configure_pins),
        DEVMETHOD(fdt_pinctrl_is_gpio, foo_is_gpio),
        DEVMETHOD(fdt_pinctrl_set_flags, foo_set_flags),
        DEVMETHOD(fdt_pinctrl_get_flags, foo_get_flags),

        /* Terminate method list */
        DEVMETHOD_END
};

DRIVER_MODULE(foo, simplebus, foo_driver, foo_devclass, NULL, NULL);
```

SEE ALSO

fdt_pinctrl(4)

AUTHORS

This manual page was written by Oleksandr Tymoshenko.