# FPGA Manager

## Overview

The FPGA manager core exports a set of functions for programming an FPGA with an image. The API is manufacturer agnostic. All manufacturer specifics are hidden away in a low level driver which registers a set of ops with the core. The FPGA image data itself is very manufacturer specific, but for our purposes it's just binary data. The FPGA manager core won't parse it.

The FPGA image to be programmed can be in a scatter gather list, a single contiguous buffer, or a firmware file. Because allocating contiguous kernel memory for the buffer should be avoided, users are encouraged to use a scatter gather list instead if possible.

The particulars for programming the image are presented in a structure (struct fpga_image_info). This struct contains parameters such as pointers to the FPGA image as well as image-specific particulars such as whether the image was built for full or partial reconfiguration.

## How to support a new FPGA device

To add another FPGA manager, write a driver that implements a set of ops. The probe function calls fpga_mgr_register() or fpga_mgr_register_full(), such as:

```
static const struct fpga_manager_ops socfpga_fpga_ops = {
        .write_init = socfpga_fpga_ops_configure_init,
        .write = socfpga_fpga_ops_configure_write,
        .write_complete = socfpga_fpga_ops_configure_complete,
        .state = socfpga_fpga_ops_state,
};

static int socfpga_fpga_probe(struct platform_device *pdev)
{
        struct device *dev = &pdev->dev;
        struct socfpga_fpga_priv *priv;
        struct fpga_manager *mgr;
        int ret;

        priv = devm_kzalloc(dev, sizeof(*priv), GFP_KERNEL);
        if (!priv)
                return -ENOMEM;

        /*
         * do ioremaps, get interrupts, etc. and save
         * them in priv
         */

        mgr = fpga_mgr_register(dev, "Altera SOCFPGA FPGA Manager",
                                &socfpga_fpga_ops, priv);
        if (IS_ERR(mgr))
                return PTR_ERR(mgr);

        platform_set_drvdata(pdev, mgr);

        return 0;
}

static int socfpga_fpga_remove(struct platform_device *pdev)
{
        struct fpga_manager *mgr = platform_get_drvdata(pdev);

        fpga_mgr_unregister(mgr);

        return 0;
}
```

Alternatively, the probe function could call one of the resource managed register functions, devm_fpga_mgr_register() or devm_fpga_mgr_register_full(). When these functions are used, the parameter syntax is the same, but the call to fpga_mgr_unregister() should be removed. In the above example, the socfpga_fpga_remove() function would not be required.

The ops will implement whatever device specific register writes are needed to do the programming sequence for this particular FPGA. These ops return 0 for success or negative error codes otherwise.

The programming sequence is::

1. .write_init
2. .write or .write_sg (may be called once or multiple times)
3. .write_complete

The .write_init function will prepare the FPGA to receive the image data. The buffer passed into .write_init will be at most

.initial_header_size bytes long; if the whole bitstream is not immediately available then the core code will buffer up at least this much before starting.

The .write function writes a buffer to the FPGA. The buffer may be contain the whole FPGA image or may be a smaller chunk of an FPGA image. In the latter case, this function is called multiple times for successive chunks. This interface is suitable for drivers which use PIO.

The .write_sg version behaves the same as .write except the input is a sg_table scatter list. This interface is suitable for drivers which use DMA.

The .write_complete function is called after all the image has been written to put the FPGA into operating mode.

The ops include a .state function which will determine the state the FPGA is in and return a code of type enum fpga_mgr_states. It doesn't result in a change in state.

## API for implementing a new FPGA Manager driver

- `fpga_mgr_states` - Values for :c:expr:`fpga_manager->state`.

  **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 109); *backlink***

  Unknown interpreted text role "c:expr".

- struct fpga_manager - the FPGA manager struct
- struct fpga_manager_ops - Low level FPGA manager driver ops
- struct fpga_manager_info - Parameter structure for fpga_mgr_register_full()
- fpga_mgr_register_full() - Create and register an FPGA manager using the fpga_mgr_info structure to provide the full flexibility of options
- fpga_mgr_register() - Create and register an FPGA manager using standard arguments
- devm_fpga_mgr_register_full() - Resource managed version of fpga_mgr_register_full()
- devm_fpga_mgr_register() - Resource managed version of fpga_mgr_register()
- fpga_mgr_unregister() - Unregister an FPGA manager

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 122)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/fpga/fpga-mgr.h
   :functions: fpga_mgr_states
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 125)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/fpga/fpga-mgr.h
   :functions: fpga_manager
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 128)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/fpga/fpga-mgr.h
   :functions: fpga_manager_ops
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api]

**[fpga]fpga-mgr.rst, line 131)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/fpga/fpga-mgr.h
   :functions: fpga_manager_info
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 134)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/fpga/fpga-mgr.c
   :functions: fpga_mgr_register_full
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 137)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/fpga/fpga-mgr.c
   :functions: fpga_mgr_register
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 140)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/fpga/fpga-mgr.c
   :functions: devm_fpga_mgr_register_full
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 143)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/fpga/fpga-mgr.c
   :functions: devm_fpga_mgr_register
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\fpga\[linux-master][Documentation][driver-api][fpga]fpga-mgr.rst, line 146)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/fpga/fpga-mgr.c
   :functions: fpga_mgr_unregister
```