NVMEM Subsystem

Srinivas Kandagatla <srinivas.kandagatla@linaro.org>

This document explains the NVMEM Framework along with the APIs provided, and how to use it.

1. Introduction

NVMEM is the abbreviation for Non Volatile Memory layer. It is used to retrieve configuration of SOC or Device specific data from non volatile memories like eeprom, efuses and so on.

Before this framework existed, NVMEM drivers like eeprom were stored in drivers/misc, where they all had to duplicate pretty much the same code to register a sysfs file, allow in-kernel users to access the content of the devices they were driving, etc.

This was also a problem as far as other in-kernel users were involved, since the solutions used were pretty much different from one driver to another, there was a rather big abstraction leak.

This framework aims at solve these problems. It also introduces DT representation for consumer devices to go get the data they require (MAC Addresses, SoC/Revision ID, part numbers, and so on) from the NVMEMs.

NVMEM Providers

NVMEM provider refers to an entity that implements methods to initialize, read and write the non-volatile memory.

2. Registering/Unregistering the NVMEM provider

A NVMEM provider can register with NVMEM core by supplying relevant nvmem configuration to nvmem_register(), on success core would return a valid nvmem_device pointer.

nvmem_unregister(nvmem) is used to unregister a previously registered provider.

For example, a simple nvram case:

```
static int brcm_nvram_probe(struct platform_device *pdev)
{
      struct nvmem_config config = {
              .name = "brcm-nvram",
              .reg_read = brcm_nvram_read,
      };
      ...
      config.dev = &pdev->dev;
      config.priv = priv;
      config.size = resource_size(res);

      devm_nvmem_register(&config);
}
```

Users of board files can define and register nvmem cells using the nvmem_cell_table struct:

```
static struct nvmem_cell_info foo_nvmem_cells[] = {
      {
              .name           = "macaddr",
              .offset         = 0x7f00,
              .bytes          = ETH_ALEN,
      }
};

static struct nvmem_cell_table foo_nvmem_cell_table = {
      .nvmem_name             = "i2c-eeprom",
      .cells                  = foo_nvmem_cells,
      .ncells                 = ARRAY_SIZE(foo_nvmem_cells),
};

nvmem_add_cell_table(&foo_nvmem_cell_table);
```

Additionally it is possible to create nvmem cell lookup entries and register them with the nvmem framework from machine code as shown in the example below:

```
static struct nvmem_cell_lookup foo_nvmem_lookup = {
      .nvmem_name             = "i2c-eeprom",
      .cell_name              = "macaddr",
      .dev_id                 = "foo_mac.0",
      .con_id                 = "mac-address",
};
```

```
nvmem_add_cell_lookups(&foo_nvmem_lookup, 1);
```

NVMEM Consumers

NVMEM consumers are the entities which make use of the NVMEM provider to read from and to NVMEM.

3. NVMEM cell based consumer APIs

NVMEM cells are the data entries/fields in the NVMEM. The NVMEM framework provides 3 APIs to read/write NVMEM cells:

```
struct nvmem_cell *nvmem_cell_get(struct device *dev, const char *name);
struct nvmem_cell *devm_nvmem_cell_get(struct device *dev, const char *name);

void nvmem_cell_put(struct nvmem_cell *cell);
void devm_nvmem_cell_put(struct device *dev, struct nvmem_cell *cell);

void *nvmem_cell_read(struct nvmem_cell *cell, ssize_t *len);
int nvmem_cell_write(struct nvmem_cell *cell, void *buf, ssize_t len);
```

**nvmem_cell_get()* apis will get a reference to nvmem cell for a given id, and nvmem_cell_read/write() can then read or write to the cell. Once the usage of the cell is finished the consumer should call **nvmem_cell_put()* to free all the allocation memory for the cell.

4. Direct NVMEM device based consumer APIs

In some instances it is necessary to directly read/write the NVMEM. To facilitate such consumers NVMEM framework provides below apis:

```
struct nvmem_device *nvmem_device_get(struct device *dev, const char *name);
struct nvmem_device *devm_nvmem_device_get(struct device *dev,
                                           const char *name);
struct nvmem_device *nvmem_device_find(void *data,
                     int (*match)(struct device *dev, const void *data));
void nvmem_device_put(struct nvmem_device *nvmem);
int nvmem_device_read(struct nvmem_device *nvmem, unsigned int offset,
                      size_t bytes, void *buf);
int nvmem_device_write(struct nvmem_device *nvmem, unsigned int offset,
                       size_t bytes, void *buf);
int nvmem_device_cell_read(struct nvmem_device *nvmem,
                           struct nvmem_cell_info *info, void *buf);
int nvmem_device_cell_write(struct nvmem_device *nvmem,
                            struct nvmem_cell_info *info, void *buf);
```

Before the consumers can read/write NVMEM directly, it should get hold of nvmem_controller from one of the **nvmem_device_get()* api.

The difference between these apis and cell based apis is that these apis always take nvmem_device as parameter.

5. Releasing a reference to the NVMEM

When a consumer no longer needs the NVMEM, it has to release the reference to the NVMEM it has obtained using the APIs mentioned in the above section. The NVMEM framework provides 2 APIs to release a reference to the NVMEM:

```
void nvmem_cell_put(struct nvmem_cell *cell);
void devm_nvmem_cell_put(struct device *dev, struct nvmem_cell *cell);
void nvmem_device_put(struct nvmem_device *nvmem);
void devm_nvmem_device_put(struct device *dev, struct nvmem_device *nvmem);
```

Both these APIs are used to release a reference to the NVMEM and devm_nvmem_cell_put and devm_nvmem_device_put destroys the devres associated with this NVMEM.

Userspace

6. Userspace binary interface

Userspace can read/write the raw NVMEM file located at:

```
/sys/bus/nvmem/devices/*/nvmem
```

ex:

```
hexdump /sys/bus/nvmem/devices/qfprom0/nvmem

0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
00000a0 db10 2240 0000 e000 0c00 0c00 0000 0c00
```

```
0000000 0000 0000 0000 0000 0000 0000 0000 0000
...
*
0001000
```

7. DeviceTree Binding

See Documentation/devicetree/bindings/nvmem/nvmem.txt