Digital TV Frontend kABI

Digital TV Frontend

The Digital TV Frontend kABI defines a driver-internal interface for registering low-level, hardware specific driver to a hardware independent frontend layer. It is only of interest for Digital TV device driver writers. The header file for this API is named `dvb_frontend.h` and located in `include/media/`.

Demodulator driver

The demodulator driver is responsible for talking with the decoding part of the hardware. Such driver should implement :c:type:`dvb_frontend_ops`, which tells what type of digital TV standards are supported, and points to a series of functions that allow the DVB core to command the hardware via the code under `include/media/dvb_frontend.c`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 18); *backlink***
>
> Unknown interpreted text role "c:type".

A typical example of such struct in a driver `foo` is:

```
static struct dvb_frontend_ops foo_ops = {
        .delsys = { SYS_DVBT, SYS_DVBT2, SYS_DVBC_ANNEX_A },
        .info = {
                .name   = "foo DVB-T/T2/C driver",
                .caps = FE_CAN_FEC_1_2 |
                        FE_CAN_FEC_2_3 |
                        FE_CAN_FEC_3_4 |
                        FE_CAN_FEC_5_6 |
                        FE_CAN_FEC_7_8 |
                        FE_CAN_FEC_AUTO |
                        FE_CAN_QPSK |
                        FE_CAN_QAM_16 |
                        FE_CAN_QAM_32 |
                        FE_CAN_QAM_64 |
                        FE_CAN_QAM_128 |
                        FE_CAN_QAM_256 |
                        FE_CAN_QAM_AUTO |
                        FE_CAN_TRANSMISSION_MODE_AUTO |
                        FE_CAN_GUARD_INTERVAL_AUTO |
                        FE_CAN_HIERARCHY_AUTO |
                        FE_CAN_MUTE_TS |
                        FE_CAN_2G_MODULATION,
                .frequency_min = 42000000, /* Hz */
                .frequency_max = 1002000000, /* Hz */
                .symbol_rate_min = 870000,
                .symbol_rate_max = 11700000
        },
        .init = foo_init,
        .sleep = foo_sleep,
        .release = foo_release,
        .set_frontend = foo_set_frontend,
        .get_frontend = foo_get_frontend,
        .read_status = foo_get_status_and_stats,
        .tune = foo_tune,
        .i2c_gate_ctrl = foo_i2c_gate_ctrl,
        .get_frontend_algo = foo_get_algo,
};
```

A typical example of such struct in a driver `bar` meant to be used on Satellite TV reception is:

```
static const struct dvb_frontend_ops bar_ops = {
        .delsys = { SYS_DVBS, SYS_DVBS2 },
        .info = {
                .name           = "Bar DVB-S/S2 demodulator",
                .frequency_min  = 500000, /* KHz */
                .frequency_max  = 2500000, /* KHz */
                .frequency_stepsize     = 0,
                .symbol_rate_min = 1000000,
                .symbol_rate_max = 45000000,
                .symbol_rate_tolerance = 500,
                .caps = FE_CAN_INVERSION_AUTO |
                        FE_CAN_FEC_AUTO |
```

```
                        FE_CAN_QPSK,
        },
        .init = bar_init,
        .sleep = bar_sleep,
        .release = bar_release,
        .set_frontend = bar_set_frontend,
        .get_frontend = bar_get_frontend,
        .read_status = bar_get_status_and_stats,
        .i2c_gate_ctrl = bar_i2c_gate_ctrl,
        .get_frontend_algo = bar_get_algo,
        .tune = bar_tune,

        /* Satellite-specific */
        .diseqc_send_master_cmd = bar_send_diseqc_msg,
        .diseqc_send_burst = bar_send_burst,
        .set_tone = bar_set_tone,
        .set_voltage = bar_set_voltage,
};
```

Note

1. For satellite digital TV standards (DVB-S, DVB-S2, ISDB-S), the frequencies are specified in kHz, while, for terrestrial and cable standards, they're specified in Hz. Due to that, if the same frontend supports both types, you'll need to have two separate :c:type:`dvb_frontend_ops` structures, one for each standard.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 100); *backlink*

   Unknown interpreted text role "c:type".

2. The `.i2c_gate_ctrl` field is present only when the hardware has allows controlling an I2C gate (either directly of via some GPIO pin), in order to remove the tuner from the I2C bus after a channel is tuned.
3. All new drivers should implement the :ref:`DVBv5 statistics <dvbv5_stats>` via `.read_status`. Yet, there are a number of callbacks meant to get statistics for signal strength, S/N and UCB. Those are there to provide backward compatibility with legacy applications that don't support the DVBv5 API. Implementing those callbacks are optional. Those callbacks may be removed in the future, after we have all existing drivers supporting DVBv5 stats.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 109); *backlink*

   Unknown interpreted text role "ref".

4. Other callbacks are required for satellite TV standards, in order to control LNBf and DiSEqC: `.diseqc_send_master_cmd`, `.diseqc_send_burst`, `.set_tone`, `.set_voltage`.

The `include/media/dvb_frontend.c` has a kernel thread which is responsible for tuning the device. It supports multiple algorithms to detect a channel, as defined at enum :c:func:`dvbfe_algo`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 123); *backlink*

Unknown interpreted text role "c:func".

The algorithm to be used is obtained via `.get_frontend_algo`. If the driver doesn't fill its field at struct dvb_frontend_ops, it will default to `DVBFE_ALGO_SW`, meaning that the dvb-core will do a zigzag when tuning, e. g. it will try first to use the specified center frequency `f`, then, it will do `f` + Δ, `f` - Δ, `f` + 2 x Δ, `f` - 2 x Δ and so on.

If the hardware has internally a some sort of zigzag algorithm, you should define a `.get_frontend_algo` function that would return `DVBFE_ALGO_HW`.

Note

The core frontend support also supports a third type (`DVBFE_ALGO_CUSTOM`), in order to allow the driver to define its own hardware-assisted algorithm. Very few hardware need to use it nowadays. Using `DVBFE_ALGO_CUSTOM` require to provide other function callbacks at struct dvb_frontend_ops.

Attaching frontend driver to the bridge driver

Before using the Digital TV frontend core, the bridge driver should attach the frontend demod, tuner and SEC devices and call :c:func:`dvb_register_frontend()`, in order to register the new frontend at the subsystem. At device detach/removal, the bridge driver should call :c:func:`dvb_unregister_frontend()` to remove the frontend from the core and then :c:func:`dvb_frontend_detach()` to free the memory allocated by the frontend drivers.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 148); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 148); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 148); *backlink*

Unknown interpreted text role "c:func".

The drivers should also call :c:func:`dvb_frontend_suspend()` as part of their handler for the :c:type:`device_driver`.``suspend()``, and :c:func:`dvb_frontend_resume()` as part of their handler for :c:type:`device_driver`.``resume()``.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 157); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 157); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 157); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 157); *backlink*

Unknown interpreted text role "c:type".

A few other optional functions are provided to handle some special cases.

Digital TV Frontend statistics

Introduction

Digital TV frontends provide a range of :ref:`statistics <frontend-stat-properties>` meant to help tuning the device and measuring the quality of service.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 172); *backlink*

Unknown interpreted text role "ref".

For each statistics measurement, the driver should set the type of scale used, or `FE_SCALE_NOT_AVAILABLE` if the statistics is not available on a given time. Drivers should also provide the number of statistics for each type. that's usually 1 for most video standards [1].

Drivers should initialize each statistic counters with length and scale at its init code. For example, if the frontend provides signal strength, it should have, on its init code:

```
struct dtv_frontend_properties *c = &state->fe.dtv_property_cache;

c->strength.len = 1;
c->strength.stat[0].scale = FE_SCALE_NOT_AVAILABLE;
```

And, when the statistics got updated, set the scale:

```
c->strength.stat[0].scale = FE_SCALE_DECIBEL;
c->strength.stat[0].uvalue = strength;
```

[1] For ISDB-T, it may provide both a global statistics and a per-layer set of statistics. On such cases, len should be equal to 4. The first value corresponds to the global stat; the other ones to each layer, e. g.:

- c->cnr.stat[0] for global S/N carrier ratio,
- c->cnr.stat[1] for Layer A S/N carrier ratio,
- c->cnr.stat[2] for layer B S/N carrier ratio,
- c->cnr.stat[3] for layer C S/N carrier ratio.

Note

Please prefer to use `FE_SCALE_DECIBEL` instead of `FE_SCALE_RELATIVE` for signal strength and CNR measurements.

Groups of statistics

There are several groups of statistics currently supported:

Signal strength (:ref:`DTV-STAT-SIGNAL-STRENGTH`)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 230); *backlink*

Unknown interpreted text role "ref".

- Measures the signal strength level at the analog part of the tuner or demod.
- Typically obtained from the gain applied to the tuner and/or frontend in order to detect the carrier. When no carrier is detected, the gain is at the maximum value (so, strength is on its minimal).
- As the gain is visible through the set of registers that adjust the gain, typically, this statistics is always available [2].
- Drivers should try to make it available all the times, as these statistics can be used when adjusting an antenna position and to check for troubles at the cabling.

[2] On a few devices, the gain keeps floating if there is no carrier. On such devices, strength report should check first if carrier is detected at the tuner (`FE_HAS_CARRIER`, see :c:type:`fe_status`), and otherwise return the lowest possible value.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 227); *backlink*

Unknown interpreted text role "c:type".

Carrier Signal to Noise ratio (:ref:`DTV-STAT-CNR`)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 244); *backlink*

Unknown interpreted text role "ref".

- Signal to Noise ratio for the main carrier.

- Signal to Noise measurement depends on the device. On some hardware, it is available when the main carrier is detected. On those hardware, CNR measurement usually comes from the tuner (e. g. after FE_HAS_CARRIER, see :c:type:`fe_status`).

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 235); *backlink*

  Unknown interpreted text role "c:type".

  On other devices, it requires inner FEC decoding, as the frontend measures it indirectly from other parameters (e. g. after FE_HAS_VITERBI, see :c:type:`fe_status`).

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 240); *backlink*

  Unknown interpreted text role "c:type".

  Having it available after inner FEC is more common.

Bit counts post-FEC (:ref:`DTV-STAT-POST-ERROR-BIT-COUNT` and :ref:`DTV-STAT-POST-TOTAL-BIT-COUNT`)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 253); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 253); *backlink*

Unknown interpreted text role "ref".

- Those counters measure the number of bits and bit errors after the forward error correction (FEC) on the inner coding block (after Viterbi, LDPC or other inner code).
- Due to its nature, those statistics depend on full coding lock (e. g. after FE_HAS_SYNC or after FE_HAS_LOCK, see :c:type:`fe_status`).

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 251); *backlink*

  Unknown interpreted text role "c:type".

Bit counts pre-FEC (:ref:`DTV-STAT-PRE-ERROR-BIT-COUNT` and :ref:`DTV-STAT-PRE-TOTAL-BIT-COUNT`)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 263); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 263); *backlink*

Unknown interpreted text role "ref".

- Those counters measure the number of bits and bit errors before the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).
- Not all frontends provide this kind of statistics.
- Due to its nature, those statistics depend on inner coding lock (e. g. after FE_HAS_VITERBI, see :c:type:`fe_status`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 262); *backlink*

Unknown interpreted text role "c:type".

Block counts (:ref:`DTV-STAT-ERROR-BLOCK-COUNT` and :ref:`DTV-STAT-TOTAL-BLOCK-COUNT`)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 272); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 272); *backlink*

Unknown interpreted text role "ref".

- Those counters measure the number of blocks and block errors after the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).
- Due to its nature, those statistics depend on full coding lock (e. g. after `FE_HAS_SYNC` or after `FE_HAS_LOCK`, see :c:type:`fe_status`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 270); *backlink*

Unknown interpreted text role "c:type".

Note

All counters should be monotonically increased as they're collected from the hardware.

A typical example of the logic that handle status and statistics is:

```
static int foo_get_status_and_stats(struct dvb_frontend *fe)
{
        struct foo_state *state = fe->demodulator_priv;
        struct dtv_frontend_properties *c = &fe->dtv_property_cache;

        int rc;
        enum fe_status *status;

        /* Both status and strength are always available */
        rc = foo_read_status(fe, &status);
        if (rc < 0)
                return rc;

        rc = foo_read_strength(fe);
        if (rc < 0)
                return rc;

        /* Check if CNR is available */
        if (!(fe->status & FE_HAS_CARRIER))
                return 0;

        rc = foo_read_cnr(fe);
        if (rc < 0)
                return rc;

        /* Check if pre-BER stats are available */
        if (!(fe->status & FE_HAS_VITERBI))
                return 0;

        rc = foo_get_pre_ber(fe);
        if (rc < 0)
                return rc;

        /* Check if post-BER stats are available */
        if (!(fe->status & FE_HAS_SYNC))
```

```
                return 0;

        rc = foo_get_post_ber(fe);
        if (rc < 0)
                return rc;
}

static const struct dvb_frontend_ops ops = {
        /* ... */
        .read_status = foo_get_status_and_stats,
};
```

Statistics collection

On almost all frontend hardware, the bit and byte counts are stored by the hardware after a certain amount of time or after the total bit/block counter reaches a certain value (usually programmable), for example, on every 1000 ms or after receiving 1,000,000 bits.

So, if you read the registers too soon, you'll end by reading the same value as in the previous reading, causing the monotonic value to be incremented too often.

Drivers should take the responsibility to avoid too often reads. That can be done using two approaches:

if the driver have a bit that indicates when a collected data is ready

Driver should check such bit before making the statistics available.

An example of such behavior can be found at this code snippet (adapted from mb86a20s driver's logic):

```
static int foo_get_pre_ber(struct dvb_frontend *fe)
{
        struct foo_state *state = fe->demodulator_priv;
        struct dtv_frontend_properties *c = &fe->dtv_property_cache;
        int rc, bit_error;

        /* Check if the BER measures are already available */
        rc = foo_read_u8(state, 0x54);
        if (rc < 0)
                return rc;

        if (!rc)
                return 0;

        /* Read Bit Error Count */
        bit_error = foo_read_u32(state, 0x55);
        if (bit_error < 0)
                return bit_error;

        /* Read Total Bit Count */
        rc = foo_read_u32(state, 0x51);
        if (rc < 0)
                return rc;

        c->pre_bit_error.stat[0].scale = FE_SCALE_COUNTER;
        c->pre_bit_error.stat[0].uvalue += bit_error;
        c->pre_bit_count.stat[0].scale = FE_SCALE_COUNTER;
        c->pre_bit_count.stat[0].uvalue += rc;

        return 0;
}
```

If the driver doesn't provide a statistics available check bit

A few devices, however, may not provide a way to check if the stats are available (or the way to check it is unknown). They may not even provide a way to directly read the total number of bits or blocks.

On those devices, the driver need to ensure that it won't be reading from the register too often and/or estimate the total number of bits/blocks.

On such drivers, a typical routine to get statistics would be like (adapted from dib8000 driver's logic):

```
struct foo_state {
        /* ... */

        unsigned long per_jiffies_stats;
}

static int foo_get_pre_ber(struct dvb_frontend *fe)
{
        struct foo_state *state = fe->demodulator_priv;
        struct dtv_frontend_properties *c = &fe->dtv_property_cache;
        int rc, bit_error;
```

```
        u64 bits;

        /* Check if time for stats was elapsed */
        if (!time_after(jiffies, state->per_jiffies_stats))
                return 0;

        /* Next stat should be collected in 1000 ms */
        state->per_jiffies_stats = jiffies + msecs_to_jiffies(1000);

        /* Read Bit Error Count */
        bit_error = foo_read_u32(state, 0x55);
        if (bit_error < 0)
                return bit_error;

        /*
         * On this particular frontend, there's no register that
         * would provide the number of bits per 1000ms sample. So,
         * some function would calculate it based on DTV properties
         */
        bits = get_number_of_bits_per_1000ms(fe);

        c->pre_bit_error.stat[0].scale = FE_SCALE_COUNTER;
        c->pre_bit_error.stat[0].uvalue += bit_error;
        c->pre_bit_count.stat[0].scale = FE_SCALE_COUNTER;
        c->pre_bit_count.stat[0].uvalue += bits;

        return 0;
}
```

Please notice that, on both cases, we're getting the statistics using the :c:type:`dvb_frontend_ops` `.read_status` callback. The rationale is that the frontend core will automatically call this function periodically (usually, 3 times per second, when the frontend is locked).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 434); *backlink***
>
> Unknown interpreted text role "c:type".

That warrants that we won't miss to collect a counter and increment the monotonic stats at the right time.

Digital TV Frontend functions and types

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)dtv-frontend.rst, line 445)**
>
> Unknown directive type "kernel-doc".
>
> ```
> .. kernel-doc:: include/media/dvb_frontend.h
> ```