Triggered Buffers

Now that we know what buffers and triggers are let's see how they work together.

IIO triggered buffer setup

- :c:func:`iio_triggered_buffer_setup` â€" Setup triggered buffer and pollfunc

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 10); *backlink*

  Unknown interpreted text role "c:func".

- :c:func:`iio_triggered_buffer_cleanup` â€" Free resources allocated by :c:func:`iio_triggered_buffer_setup`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 11); *backlink*

  Unknown interpreted text role "c:func".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 11); *backlink*

  Unknown interpreted text role "c:func".

- struct iio_buffer_setup_ops â€" buffer setup related callbacks

A typical triggered buffer setup looks like this:

```
const struct iio_buffer_setup_ops sensor_buffer_setup_ops = {
  .preenable    = sensor_buffer_preenable,
  .postenable   = sensor_buffer_postenable,
  .postdisable  = sensor_buffer_postdisable,
  .predisable   = sensor_buffer_predisable,
};

irqreturn_t sensor_iio_pollfunc(int irq, void *p)
{
    pf->timestamp = iio_get_time_ns((struct indio_dev *)p);
    return IRQ_WAKE_THREAD;
}

irqreturn_t sensor_trigger_handler(int irq, void *p)
{
    u16 buf[8];
    int i = 0;

    /* read data for each active channel */
    for_each_set_bit(bit, active_scan_mask, masklength)
        buf[i++] = sensor_get_data(bit)

    iio_push_to_buffers_with_timestamp(indio_dev, buf, timestamp);

    iio_trigger_notify_done(trigger);
    return IRQ_HANDLED;
}

/* setup triggered buffer, usually in probe function */
iio_triggered_buffer_setup(indio_dev, sensor_iio_polfunc,
                           sensor_trigger_handler,
                           sensor_buffer_setup_ops);
```

The important things to notice here are:

- :c:type:`iio_buffer_setup_ops`, the buffer setup functions to be called at predefined points in the buffer configuration sequence (e.g. before enable, after disable). If not specified, the IIO core uses the default iio_triggered_buffer_setup_ops.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 52); *backlink*

Unknown interpreted text role "c:type".

- **sensor_iio_pollfunc**, the function that will be used as top half of poll function. It should do as little processing as possible, because it runs in interrupt context. The most common operation is recording of the current timestamp and for this reason one can use the IIO core defined :c:func:`iio_pollfunc_store_time` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 56); *backlink*

Unknown interpreted text role "c:func".

- **sensor_trigger_handler**, the function that will be used as bottom half of the poll function. This runs in the context of a kernel thread and all the processing takes place here. It usually reads data from the device and stores it in the internal buffer together with the timestamp recorded in the top half.

More details

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\iio\[linux-master][Documentation][driver-api][iio]triggered-buffers.rst, line 69)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/iio/buffer/industrialio-triggered-buffer.c
```