Industrial I/O Subsystem: The Home of Linux Sensors

Daniel Baluta

Intel
daniel.baluta@intel.com

October 5, 2015
Why Industrial I/O?

- past - industrial process control or scientific research
- present - all kinds of devices: phones, tablets, laptops, TVs
- fill the gap between input and hwmon subsystems
  - hwmon - low sample rate sensors used to control/monitor the system itself (fan speed control, temperature)
  - input - human interaction input devices (keyboard, mouse, touchscreen)
- Industrial I/O (IIO) - de facto standard for sensors
- many drivers in Android use input for sensors - this should be changed
What is Industrial I/O?

- devices that in some sense are Analog to Digital Converters (ADC)
- support for Digital to Analog converters (DACs)
- unified framework for different types of embedded sensors
- started by Jonathan Cameron
- in staging from 2.6.32 in 2009
- merged in Linux kernel from 3.15 in 2012
- currently, in 4.3-rc3 there are around 184 IIO drivers
Industrial I/O supported sensor types

- accelerometers
- magnetometers
- gyroscopes
- pressure
- humidity
- temperature
- light and proximity
- activity
- chemical
- heart rate monitors
- potentiometers and rheostats
Industrial I/O architecture overview

```
user   Application
    -------
      fs
        char device  sysfs
      core
        iio-core
        iio-trigger  iio-buffer  iio-triggered-buffer  iio-events
      driver
        IIO driver
      bus
      hardware
        hardware device
```
Industrial I/O devices

- an IIO device is a representation of a single hardware sensor
- struct iio_dev
  - operating modes
    - DIRECT, BUFFERSOFTWARE, BUFFER_HARDWARE, BUFFER_TRIGGERED
  - chrdev
  - sysfs attributes
  - channels
  - buffers
  - triggers
  - events
- iio_device_alloc / iio_device_free
- iio_device_register / iio_device_unregister
Industrial I/O interface with user space

- **sysfs**
  - Documentation/ABI/testing/sysfs-bus-iio
  - used for configuration and raw data readings
  - /sys/bus/iio/devices/iio:deviceX
    - name - usually part number
    - dev - device node id (major:minor)
    - device configuration attributes (sampling_frequency_available)
    - data channel access attributes (in_resistance_raw)
    - buffer/, events/, trigger/, scan_elements/
  - /sys/bus/iio/devices/iio:triggerY

- **character device** - /dev/iio:deviceX
  - access to the kernel buffers of data samples/events
represents a single data source from the device

struct iio_chan_spec
- type (IIO_ACCEL, IIO_INTENSITY)
- channel - a number assigned to the channel
- modifiers (IIO_MOD_X, IIO_MOD_LIGHT_RED)
- channels attributes are specified as bit masks (IIO_CHAN_INFO_SCALE)
- scan_index - ordering of this sample in the buffer
- events are associated with the channel via struct iio_event_spec

data access attributes generic form: {direction}_{type}_{index}_{modifier}_{info}
- scaled angular velocity about the X axis: in_anglvel_x_input
- raw voltage measurement from channel 0: in_voltage0_raw
IIO channel definition for a temperature sensor

```c
struct iio_chan_spec temp_channel[] = {
    {
        .type = IIO_TEMP,
        .info_mask_separate = BIT(IIO_CHAN_INFO_PROCESSED),
    },
};
```

- `/sys/bus/iio/devices/iio:device0/in_temp_input`
IIO channels definition for a 3-axis compass

```c
struct iio_chan_spec magn_channels[] = {
    {
        .type = IIO_MAGN,
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW),
        .info_mask_shared_by_type = BIT(IIO_CHAN_INFO_SCALE),
        .modified = 1,
        .channel2 = IIO_MOD_X,
    },
    /* Y, Z axis channel definitions */
};
```

- `/sys/bus/iio/devices/iio:device0/in_magn_x_raw`
- `/sys/bus/iio/devices/iio:device0/in_magn_scale`
const struct iio_info magn_info = {
    .read_raw = magn_read_raw,
    .write_raw = magn_write_raw,
};

int magn_read_raw(indio_dev, chan, val, val2, mask)
{
    switch (mask) {
        case IIO_CHAN_INFO_RAW:
            val = read_magn(chan->address);
            return IIO_VAL_INT;
        case IIO_CHAN_INFO_SCALE:
            *val = 1;
            *val2 = 500000;
            return IIO_VAL_INT_PLUS_MICRO;
    }
    return -EINVAL;
}

/* on IIO device init */
indio_dev->info = &magn_info;
Industrial I/O buffers

- `struct iio_buffer`
- on chip hardware FIFO buffers
  - reduce the load on host CPU
- software buffers
  - continuous data capture fired by a trigger
- data retrieved from the char device node
  - `/dev/iio:deviceX`
items placed in buffers are called scans
- sysfs meta information + actual sample data in buffer

/sys/bus/iio/devices/iio:devices/scan_elements
- per channel enable attribute
  - echo 1 > /sys/.../iio:device0/scan_elements/in.accel_x.en
- per sensor type scans description
  - /sys/.../iio:device0/scan_elements/in.accel_type
  - [be|le]:[s|u]bits/storagebitsXrepeat[>>shift]

/sys/bus/iio/devices/iio:devices0/buffer
- length - buffer capacity in number of scans
- enable - activate buffer capture
setup built-in IIO device registration

buffer support is specified per channel via `scan_index`

3-axis accelerometer, 12 bits resolution, two 8-bit data registers

```
| D3 | D2 | D1 | D0 | X | X | X | X | (LOW byte, address 0x06)
```

```
| D11| D10| D9 | D8 | D7 | D6 | D5 | D4 | (HIGH byte, address 0x07)
```
```c
struct iio_chan_spec accel_channels[] = {
    {
        .type = IIO_ACCEL,
        /* ... */
        .scan_index = 0,
        .scan_type = {
            .sign = 'u',
            .realbits = 12, /* valid data bits */
            .storagebits = 16,
            .shift = 4,
            .endianness = IIO_LE,
        },
    },
    /* Y, Z axis channels definition */
};
```
Industrial I/O triggers

- alternative to polling for data available
- trigger readings based on an external interrupt source
  - hardware interrupt (IRQ pins)
  - software interrupts (periodic timers, sysfs triggers)
- multiple consumers - a trigger may be used by multiple devices
- `iio_trigger_alloc / iio_trigger_free`
- `iio_trigger_register / iio_trigger_unregister`
- `struct iio_trigger_ops`
  - `set_trigger_state` - trigger config (e.g. configure interrupts)
  - `validate_device`
/sys/bus/iio/devices/triggerX
  - name - used to identify the driver
  - various parameters - depending on trigger source

/sys/bus/iio/devices/iio:device0/trigger/
  - current_trigger - trigger associated with this device
  - link between triggers and buffers is done with triggered buffers
Industrial I/O software triggers

- interrupt trigger
- sysfs trigger
- proposal for configfs interface to create triggers
  - /config/iio/triggers
  - mkdir hrtimer
  - mkdir hrtimer/trigger0
  - work in progress
Industrial I/O triggered buffers

- `iio_triggered_buffer_setup`, `iio_triggered_buffer_cleanup`
  - `@h` - function to be called when IRQ occurs
  - `@thread` - function to be called from the IRQ handler thread

- `buffer_setup_ops`
  - `preenable` - user defined (usually powers on chip)
  - `postenable` - attaches poll functions to the trigger
  - `predisable` - detaches poll functions to the trigger
  - `postdisable` - user defined (usually powers off chip)

- `iio_pollfunc_storetime`
  - predefined top half function that stores the current time stamp
Industrial I/O triggered buffers setup

```bash
# go to I/O dir
$ cd /sys/bus/iio/devices/
# list available triggers
$ ls trigger*
  trigger0  trigger1
# set trigger0 as current trigger for device0
$ echo trigger0 > iio:device0/trigger/current_trigger
# activate channels
$ echo 1 > iio:device0/scan_elements/in_magazines
$ echo 1 > iio:device0/scan_elements/in_magnets
$ echo 1 > iio:device0/scan_elements/in_magnets
# check buffer capacity (number of samples)
$ cat iio:device0/buffer/length
  2
# final step: enable buffer
$ echo 1 > iio:device0/buffer/enable
```
Industrial I/O events

- pass out of band information to user space
- correspond to some thresholds based on sensor raw readings
  - direct crossing voltage threshold
  - crossing a rate of change threshold
  - entering/leaving an activity state
- configured via sysfs interface
- information retrieved via a special fd obtained from /dev/iio:deviceX
Events support for a proximity sensor (1)

```c
struct iio_event_spec prox_event = {
    .type = IIO_EV_TYPE_THRESHOLD,
    .dir = IIO_EV_DIR_EITHER, /* rising or falling */
    .mask_separate = IIO_EV_INFO_ENABLE | IIO_EV_INFO_VALUE,
};

struct iio_chan_spec prox_channels[] = {
    .type = IIO_PROXIMITY,
    /* .. */
    .event_spec = &prox_event,
};
```

- echo 100 > /sys/.../iio:device0/events/in_proximity_thresh_rising_value
- echo 1 > /sys/.../iio:device0/events/in_proximity_thresh_rising_en
static const struct iio_info prox_info = {
    /* ... */
    .read_event_value = prox_read_event_value,
    .write_event_value = prox_write_event_value,
    .read_event_config = prox_read_event_config,
    .write_event_config = prox_write_event_config,
};

/* on IIO device init */
indio_dev->info = &prox_info;

- callbacks used for handling events sysfs reads/writes operations
- {read/write}_event_config, handles events enabling
- {read/write}_event_value, handles events configuration
IIO events path

User

/dev/iio:device0

User

read(event_fd, ...)

sysfs config files

Kernel

fd

get_event_fd

kfifo events buffer

irq handler

generate timestamp

copy to buffer

thread handler

hardware device

IRQ
Delivering IIO events to user space

- usually handled using threaded IRqs
  - because bus access functions might sleep
- `iio_push_events(indio_dev, ev_code, timestamp)`
  - event code contains channel type, modifier, direction, event type
  - macros for packing/unpacking event codes
    - `IIO_MOD_EVENT_CODE`
    - `IIO_EVENT_CODE_EXTRACT`
- applications can read events via a special file descriptor
- ioctl command `IIO_GET_EVENT_FD_IOCTL` on `/dev/iio:deviceX` fd
Industrial I/O testing utilities

- tools/iio/
  - generic_buffer.c
  - iio_event_monitor.c
  - lsiio.c

- IIO dummy module
- IIO event generator module
New things in IIO

- chemical sensors
- potentiometer
- software triggers
- heart rate monitors
- input - IIO bridge
- IIO DMA buffer
- IIO dummy module move out of staging
linux-iio@vger.kernel.org