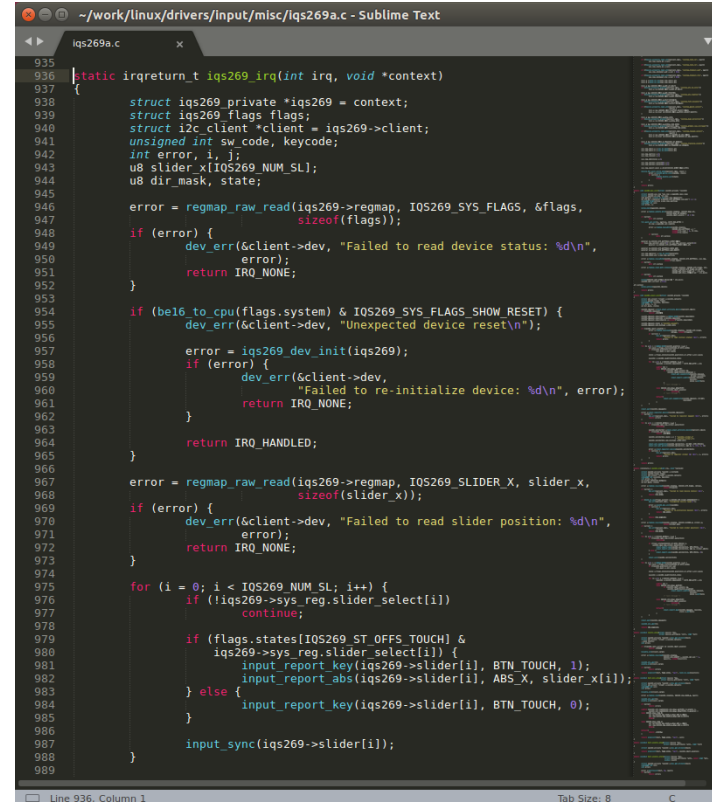


Azoteq IQS269A Linux Kernel Driver

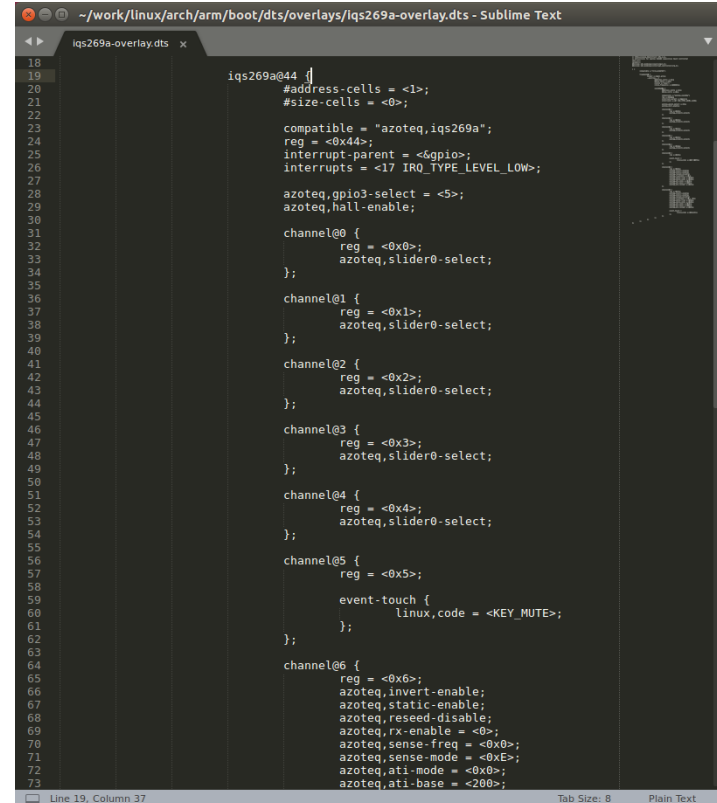
- Enables the IQS269A in Android and other embedded Linux applications
- Interfaces to the Linux input core for direct communication with the Android EventHub
- Efficient use of existing Linux frameworks simplifies integration and system bring-up

- Handles all low-level communication (I²C transactions and RDY/interrupt handling)
- Registers up to 3 input devices with the Linux kernel
 - Keypad for individual sensing channel events
 - Slider 0 and 1 UIs
- All events can be assigned a Linux input event “key code” (KEY_MUTE, etc.)
 - Proximity, touch and deep touch events
 - Positive or negative delta
- Controls power mode based on system state

A screenshot of a code editor window titled "iqs269a.c" showing C code for an I2C driver. The code includes a static irqreturn_t function iqs269_irq that handles device status, initialization, and slider position reads. It uses regmap_raw_read to access hardware registers and input_report_key to generate Linux input events. The code is line-numbered from 935 to 989.

```
935 static irqreturn_t iqs269_irq(int irq, void *context)
936 {
937     struct iqs269 private *iqs269 = context;
938     struct iqs269 flags flags;
939     struct i2c_client *client = iqs269->client;
940     unsigned int sw_code, keycode;
941     int error, i, j;
942     u8 slider_x[IQS269_NUM_SL];
943     u8 dir_mask, state;
944
945     error = regmap_raw_read(iqs269->regmap, IQS269_SYS_FLAGS, &flags,
946                             sizeof(flags));
947     if (error) {
948         dev_err(&client->dev, "Failed to read device status: %d\n",
949                 error);
950         return IRQ_NONE;
951     }
952
953     if (be16_to_cpu(flags.system) & IQS269_SYS_FLAGS_SHOW_RESET) {
954         dev_err(&client->dev, "Unexpected device reset\n");
955     }
956     error = iqs269_dev_init(iqs269);
957     if (error) {
958         dev_err(&client->dev,
959                 "Failed to re-initialize device: %d\n", error);
960         return IRQ_NONE;
961     }
962     return IRQ_HANDLED;
963 }
964
965 error = regmap_raw_read(iqs269->regmap, IQS269_SLIDER_X, slider_x,
966                         sizeof(slider_x));
967 if (error) {
968     dev_err(&client->dev, "Failed to read slider position: %d\n",
969             error);
970     return IRQ_NONE;
971 }
972
973 for (i = 0; i < IQS269_NUM_SL; i++) {
974     if (!iqs269->sys_reg.slider_select[i])
975         continue;
976
977     if (flags.states[IQS269_ST_OFFS_TOUCH] &
978         iqs269->sys_reg.slider_select[i]) {
979         input_report_key(iqs269->slider[i], BTN_TOUCH, 1);
980         input_report_abs(iqs269->slider[i], ABS_X, slider_x[i]);
981     } else {
982         input_report_key(iqs269->slider[i], BTN_TOUCH, 0);
983     }
984     input_sync(iqs269->slider[i]);
985 }
986
987 }
988
989 }
```

- Compile-time control of nearly every register
 - All parameters exposed as device tree properties
 - Device tree is a ubiquitous data structure that describes hardware
- All 8 channels represented as fully configurable device tree child nodes
- Run-time control of ATI-specific registers
 - Mirrored to user space through sysfs attributes (i.e. R/O or R/W “files”)
 - Facilitates production-line calibration of Hall sensor

A screenshot of a Sublime Text editor window showing a device tree overlay file named "iqs269a-overlay.dts". The code defines a device node "iqs269a@44" with various properties and child nodes. The child nodes are "channel@0" through "channel@6", each with its own set of properties. The "channel@6" node includes an "event-touch" sub-node with a "linux,code" property.

```
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

iqs269a@44 {
    #address-cells = <1>;
    #size-cells = <0>;

    compatible = "azoteq,iqs269a";
    reg = <0x44>;
    interrupt-parent = <6gpio>;
    interrupts = <17 IRQ_TYPE_LEVEL_LOW>;

    azoteq,gpio3-select = <5>;
    azoteq,hall-enable;

    channel@0 {
        reg = <0x0>;
        azoteq,slider0-select;
    };

    channel@1 {
        reg = <0x1>;
        azoteq,slider0-select;
    };

    channel@2 {
        reg = <0x2>;
        azoteq,slider0-select;
    };

    channel@3 {
        reg = <0x3>;
        azoteq,slider0-select;
    };

    channel@4 {
        reg = <0x4>;
        azoteq,slider0-select;
    };

    channel@5 {
        reg = <0x5>;

        event-touch {
            linux,code = <KEY_MUTE>;
        };
    };

    channel@6 {
        reg = <0x6>;
        azoteq,invert-enable;
        azoteq,static-enable;
        azoteq,reset-disable;
        azoteq,rx-enable = <0>;
        azoteq,sense-freq = <0x0>;
        azoteq,sense-mode = <0xE>;
        azoteq,ati-mode = <0x0>;
        azoteq,ati-base = <200>;
    };
};
```



Demonstration Platform

IQS269A Stamp Module

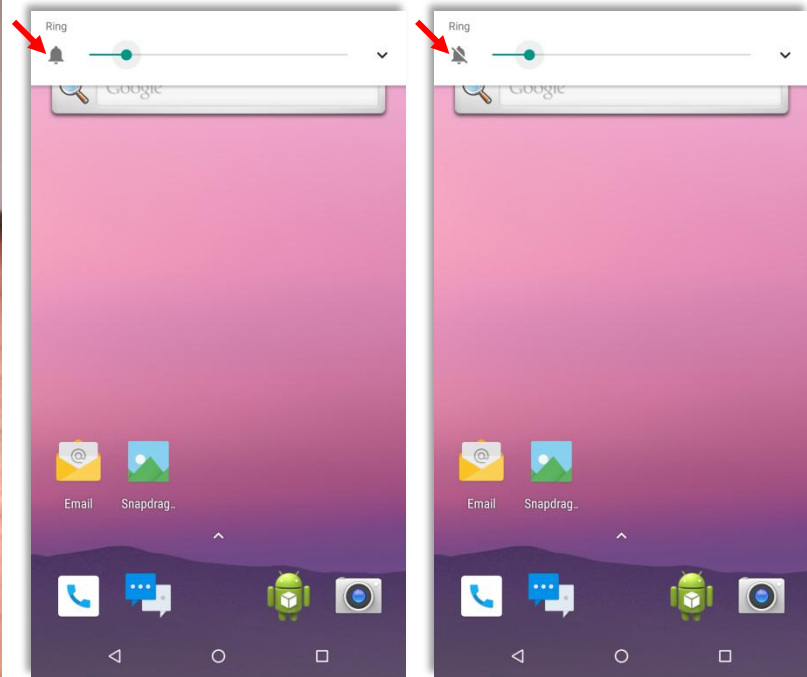
8-Channel Capacitive Keypad/Slider

Power/I²C/GPIO Header

Android Development Kit



Ring/Vibrate Touch Key





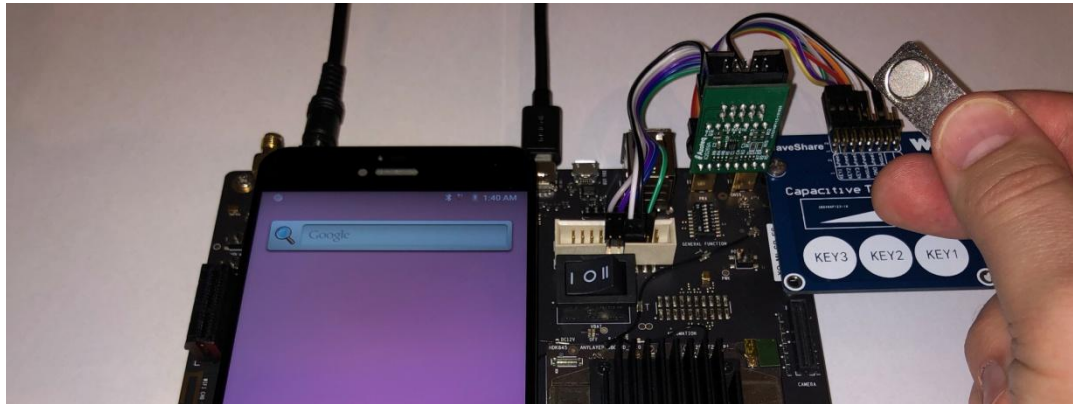
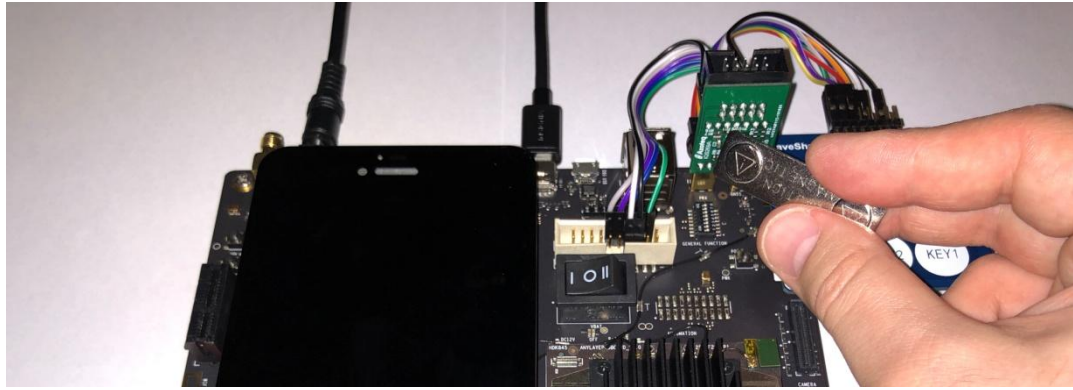
Generic Touch Slider

- Slider activity reported using input event codes commonly used for axial sliders

```
jlabundy@nixie71: ~  
  
0001 014a 00000001 ← EV_KEY: touch start (BTN_TOUCH = 0x014A = 1)  
0003 0000 0000002a  
0000 0000 00000000  
0003 0000 00000032  
0000 0000 00000000  
0003 0000 00000045  
0000 0000 00000000  
0003 0000 0000004d ← EV_ABS: absolute coordinate change (0-255)  
0000 0000 00000000  
0003 0000 00000055  
0000 0000 00000000  
0003 0000 0000005c  
0000 0000 00000000  
0003 0000 00000063  
0000 0000 00000000  
0003 0000 00000069  
0000 0000 00000000  
0003 0000 0000006d  
0000 0000 00000000  
0001 014a 00000000 ← EV_KEY: touch stop (BTN_TOUCH = 0x014A = 0)  
0000 0000 00000000
```

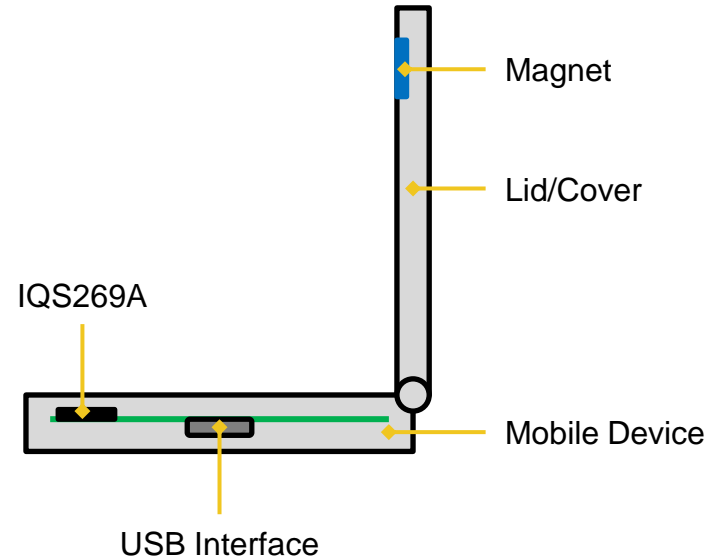

Magnetic Lid Switch

- Channel 7 events reported as change in switch state (EV_SW) instead of key press/release (EV_KEY) if Hall UI is enabled
- Some Linux switch codes (e.g. SW_LID, SW_DOCK) invoke preset behaviors in Android (e.g. screen on/off)



Production-Line Calibration Overview

- Driver provides means to derive unit-specific ATI target (N_T) for Hall channel pair during production
- Calibration is performed using shell scripts executed on host via Android Debug Bridge over USB
- N_T is written to target's nonvolatile memory during production and passed to driver each time target is booted





1. Set compile-time properties in device tree (see iqs269a.yaml)

```
iqs269a@44 {
    [...]
    azoteg,hall-enable;

    channel@6 {
        reg = <0x6>;
        azoteg,invert-enable;
        azoteg,static-enable;
        azoteg,reed-disable;
        azoteg,rx-enable = <0>;
        azoteg,sense-freq = <0x0>;
        azoteg,sense-mode = <0xE>;
        azoteg,ati-mode = <0x0>;
        azoteg,ati-base = <200>;
        azoteg,ati-target = <320>;
    };

    channel@7 {
        reg = <0x7>;
        azoteg,invert-enable;
        azoteg,static-enable;
        azoteg,reed-disable;
        azoteg,rx-enable = <0>, <6>;
        azoteg,sense-freq = <0x0>;
        azoteg,sense-mode = <0xE>;
        azoteg,ati-mode = <0x3>;
        azoteg,ati-base = <200>;
        azoteg,ati-target = <320>;

        event-touch {
            linux,code = <SW_LID>;
        };
    };
};
```

2. Override relevant properties in user space

```
echo 0 > /sys/bus/i2c/devices/1-0044/hall_enable
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
echo 3 > /sys/bus/i2c/devices/1-0044/ati_mode
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
echo 3 > /sys/bus/i2c/devices/1-0044/ati_mode
```

3. Open lid (i.e. remove magnet)
4. Update registers and trigger ATI

```
echo 1 > /sys/bus/i2c/devices/1-0044/ati_trigger
```

5. Close lid (i.e. apply magnet)
6. Read counts, ATI base/target and Hall pad bin number

```
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
cat /sys/bus/i2c/devices/1-0044/counts
302
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
cat /sys/bus/i2c/devices/1-0044/counts
342
cat /sys/bus/i2c/devices/1-0044/ati_base
200
cat /sys/bus/i2c/devices/1-0044/ati_target
320
cat /sys/bus/i2c/devices/1-0044/hall_bin
8
```

7. Ensure neither inverting nor non-inverting counts reach 8192

8. Calculate i_a

$$i_a = IN_B \left| \frac{1}{N_T} - \frac{1}{n} \right| = 6.25 \times 200 \times \left| \frac{1}{320} - \frac{1}{342} \right| = 0.25 \mu A$$

9. Calculate N_T based on desired counts (e.g. $n_z = 500$)

$$N_T = \frac{1}{\frac{1}{n_z} + \frac{i_a}{IN_B}} = \frac{1}{\frac{1}{500} + \frac{0.25}{6.25 \times 200}} = 454$$

10. Write N_T to channels 6 and 7

```
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
echo 454 > /sys/bus/i2c/devices/1-0044/ati_target
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
echo 454 > /sys/bus/i2c/devices/1-0044/ati_target
```

11. Open lid (i.e. remove magnet)

12. Update registers and trigger ATI

```
echo 1 > /sys/bus/i2c/devices/1-0044/ati_trigger
```

13. Close lid (i.e. apply magnet)

14. Read updated counts

```
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
cat /sys/bus/i2c/devices/1-0044/counts
414
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
cat /sys/bus/i2c/devices/1-0044/counts
490
```

15. Ensure channel 7 (EV_SW reporting) counts are reasonably close to n_z

16. Write N_T to nonvolatile memory (e.g. persist partition)

17. Restore compile-time properties

```
echo 1 > /sys/bus/i2c/devices/1-0044/hall_enable
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
echo 0 > /sys/bus/i2c/devices/1-0044/ati_mode
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
echo 3 > /sys/bus/i2c/devices/1-0044/ati_mode
```

18. Open lid (i.e. remove magnet)

19. Update registers and trigger ATI

```
echo 1 > /sys/bus/i2c/devices/1-0044/ati_trigger
```

Post-Calibration Boot Sequence

1. Read N_T from nonvolatile memory (e.g. persist partition)
2. Write N_T to channels 6 and 7 via init.rc

```
echo 6 > /sys/bus/i2c/devices/1-0044/ch_number
echo $NT > /sys/bus/i2c/devices/1-0044/ati_target
echo 7 > /sys/bus/i2c/devices/1-0044/ch_number
echo $NT > /sys/bus/i2c/devices/1-0044/ati_target
```

3. Update registers and trigger ATI via init.rc

```
echo 1 > /sys/bus/i2c/devices/1-0044/ati_trigger
```



User-Space Control Summary

Name	Access	Description
ch_number	R/W	Channel number selection (0–7)
rx_enable	R/W*	Sensing pin enable/disable for the selected channel (CRX[7:0])
counts	R/O	Filtered counts for the selected channel
hall_bin	R/O	Bin number for the Hall pad selected by rx_enable[6] and rx_enable[7] (both must agree)
hall_enable	R/W*	Hall UI enable/disable
ati_mode	R/W*	ATI mode for the selected channel (0 = disabled, 1 = semi-partial, 2 = partial, 3 = full)
ati_base	R/W*	ATI base for the selected channel (75, 100, 150 or 200)
ati_target	R/W*	ATI target for the selected channel (0–2016)
ati_trigger	R/W	R: non-zero value indicates all registers are up-to-date W: non-zero value updates all registers and triggers ATI

* Registers are not updated until ati_trigger is written with a non-zero value



LABUNDY.COM