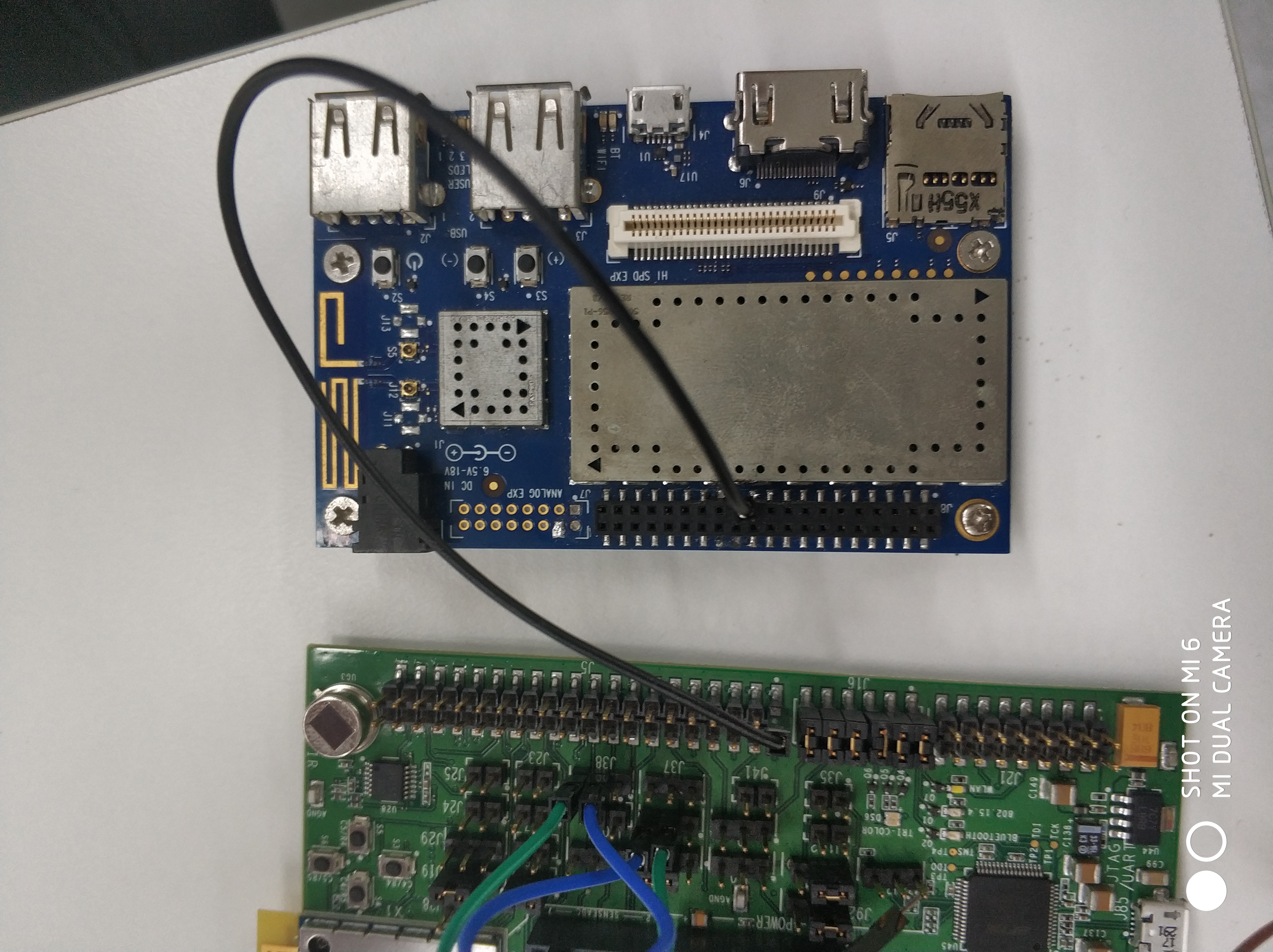
410c receives data from qca4020  
  
0. Preface  
The QCA4020 monitors the status of the smoke sensor. When the monitored value is greater than a certain threshold, the level value of gpio is set to indicate that an event has occurred. For specific implementation, please see QCA4020 Smoke Sensor Data Read.docx  
  
1. Purpose  
This article aims to achieve:  
(1) The 410c development board monitors the state change of the gpio port to which the QCA4020 is connected, thereby generating an interruption.  
(2) Provide a node for the upper layer to view the status of the smoke sensor. One of them indicates that an event has occurred. You can set the node value to 0 by writing to this node.  
  
2. Connection diagram



3. Code  
(1) Configure the device tree  
Add node

/\*arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi\*/

qca4020 {

           compatible = "linux,QCA4020";

           interrupt-parent = <&msmgpio>;

           interrupts = <0 36 2>;

           gpio\_int = <&msmgpio 36 0>;

};

（2）Configuring Makefile

/\*drivers/misc/Makefile\*/

obj-y += qca4020.o

（3）Drive implementation

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/sysfs.h>

#include <linux/delay.h>

#include <linux/platform\_device.h>

#include <linux/err.h>

#include <linux/device.h>

#include <linux/interrupt.h>

#include <linux/irq.h>

#include <linux/of\_gpio.h>

#include <asm/uaccess.h>

#include <linux/kdev\_t.h>

#include <linux/slab.h>

#include <linux/workqueue.h>

struct qca4020\_data {

   struct platform\_device \*pdev;

   int flag;

   int gpio\_int;

   int irq;

};

static struct qca4020\_data\* data;

static irqreturn\_t qca4020\_interrupt\_handler(int irq, void \*ptr)

{

   data->flag = 1;

   return IRQ\_HANDLED;

}

static ssize\_t qca4020\_value\_store(struct device \*dev, struct device\_attribute\* attr,

                                           const char \*buf, size\_t len)

{

   data->flag = 0;

   return len;

}

static ssize\_t qca4020\_value\_show(struct device \*dev, struct device\_attribute\* attr,

                                           char \*buf)

{

   ssize\_t ret = sprintf(buf, "%d\n", data->flag);

   return ret;

}

static DEVICE\_ATTR(value, 0664, qca4020\_value\_show, qca4020\_value\_store);

static int QCA4020\_probe(struct platform\_device \*pdev)

{

   int result;

   struct device\_node\* node = pdev->dev.of\_node;

   printk("qca4020 probe enter\n");

   data = devm\_kzalloc(&pdev->dev, sizeof(struct qca4020\_data), GFP\_KERNEL);

   if (!data) {

       pr\_err("%s kzalloc error\n", \_\_FUNCTION\_\_);

       return -ENOMEM;

  }

   dev\_set\_drvdata(&pdev->dev, data);

   data->gpio\_int = of\_get\_named\_gpio(node, "gpio\_int", 0);

   if (!gpio\_is\_valid(data->gpio\_int)) {

       pr\_err("gpio\_int not specified\n");

       goto err;

  } else {

       result = gpio\_request(data->gpio\_int, "qca\_gpio");

       if (result < 0) {

           pr\_err("Unable to request qca\_gpio\n");

           goto err;

      } else {

           gpio\_direction\_input(data->gpio\_int);

      }

  }

   data->irq = gpio\_to\_irq(data->gpio\_int);

   result = request\_irq(data->irq, qca4020\_interrupt\_handler,

                                                IRQF\_TRIGGER\_RISING | IRQF\_TRIGGER\_FALLING, "qca4020\_intr", data);

   if (result < 0) {

       pr\_err("Unable to request irq\n");

       goto err;

  }

   result = sysfs\_create\_file(&pdev->dev.kobj, &dev\_attr\_value.attr);

   if (result < 0) {

       printk("sysfs create file failed\n");

       goto err;

  }

   printk(KERN\_INFO "QCA4020 probe success\n");

   return 0;

   err:

   kfree(data);

   printk(KERN\_ERR "QCA4020 probe failed\n");

   return -EINVAL;

}

static int QCA4020\_remove(struct platform\_device \*pdev)

{

       gpio\_free(data->gpio\_int);

       kfree(data);

       return 0;

}

static struct of\_device\_id mach\_match\_table[] = {

  { .compatible = "linux,QCA4020"},

  { },

};

static struct platform\_driver QCA4020\_driver = {

  .probe = QCA4020\_probe,

  .remove = QCA4020\_remove,

  .driver = {

      .owner = THIS\_MODULE,

      .name = "QCA4020GPIO",

      .of\_match\_table = mach\_match\_table,

  },

};

module\_platform\_driver(QCA4020\_driver);

MODULE\_AUTHOR("yanggh0703@thundersoft.com");

MODULE\_LICENSE("GPL");