Smoke Sensor Data Deading

In the development board using qca4020, there are quite a few demos. According to the instruction manual, this experiment uses QCLI\_demo. In this demo, there is a Peripherals module, which contains various sensor data for reading the ADC board and contains settings. The voltage value and interrupt value of GPIO, so the main idea of this smoke alarm is: use the peripheral module in QCLI\_demo, use its ADC interface to read the value of the smoke sensor, and the value of the smoke alarm is higher than a certain voltage range. Set the voltage value of a GPIO on the board to a high level.

1. open the Peripherals module

In actual use, the peripherals module is not fully open, so code modification is required to open and use the peripherals module. The module is opened as follows:  
(1) modify build.bat and env.config

**diff --git a/build/gcc/build.bat b/build/gcc/build.bat**

**index d3a8ccc..ae5a568 100644**

**--- a/build/gcc/build.bat**

**+++ b/build/gcc/build.bat**

@@ -20,7 +20,7 @@ IF /I "%CFG\_FEATURE\_FS%" == "" (SET CFG\_FEATURE\_FS=true)

IF /I "%CFG\_FEATURE\_SECUREFS%" == "" (SET CFG\_FEATURE\_SECUREFS=true)

IF /I "%CFG\_FEATURE\_THREAD%" == "" (SET CFG\_FEATURE\_THREAD=true)

IF /I "%CFG\_FEATURE\_I2S%" == "" (SET CFG\_FEATURE\_I2S=false)

-IF /I "%CFG\_FEATURE\_PERIPHERALS%" == "" (SET CFG\_FEATURE\_PERIPHERALS=false)

+IF /I "%CFG\_FEATURE\_PERIPHERALS%" == "" (SET CFG\_FEATURE\_PERIPHERALS=true)

**diff --git a/build/gcc/env.config b/build/gcc/env.config**

**index ed6f116..0fd8336 100644**

**--- a/build/gcc/env.config**

**+++ b/build/gcc/env.config**

@@ -36,7 +36,7 @@ CFG\_FEATURE\_ZIGBEE=true

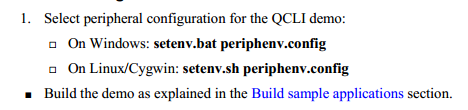
CFG\_FEATURE\_THREAD=true

### Peripherals Demo ###

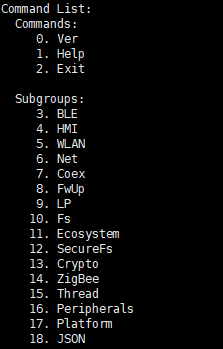
-CFG\_FEATURE\_PERIPHERALS=false

+CFG\_FEATURE\_PERIPHERALS=true

(2)set the compilation environment

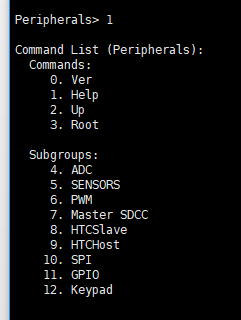


(3)then compile the code and download  
 Go to the ./build/gcc/ directory and execute the following command:  
.\build.bat prepare 4020 cdb  
.\build.bat t 4020 cdb  
Using the shorting cap, short the J34 12 port, use usb to download, the download command is as follows:  
Python ..\..\..\..\..\build\tools\flash\qflash.py --comm \*\*--app ./  
After programming, unplug the shorting cap to open the serial port and the following directory appears:

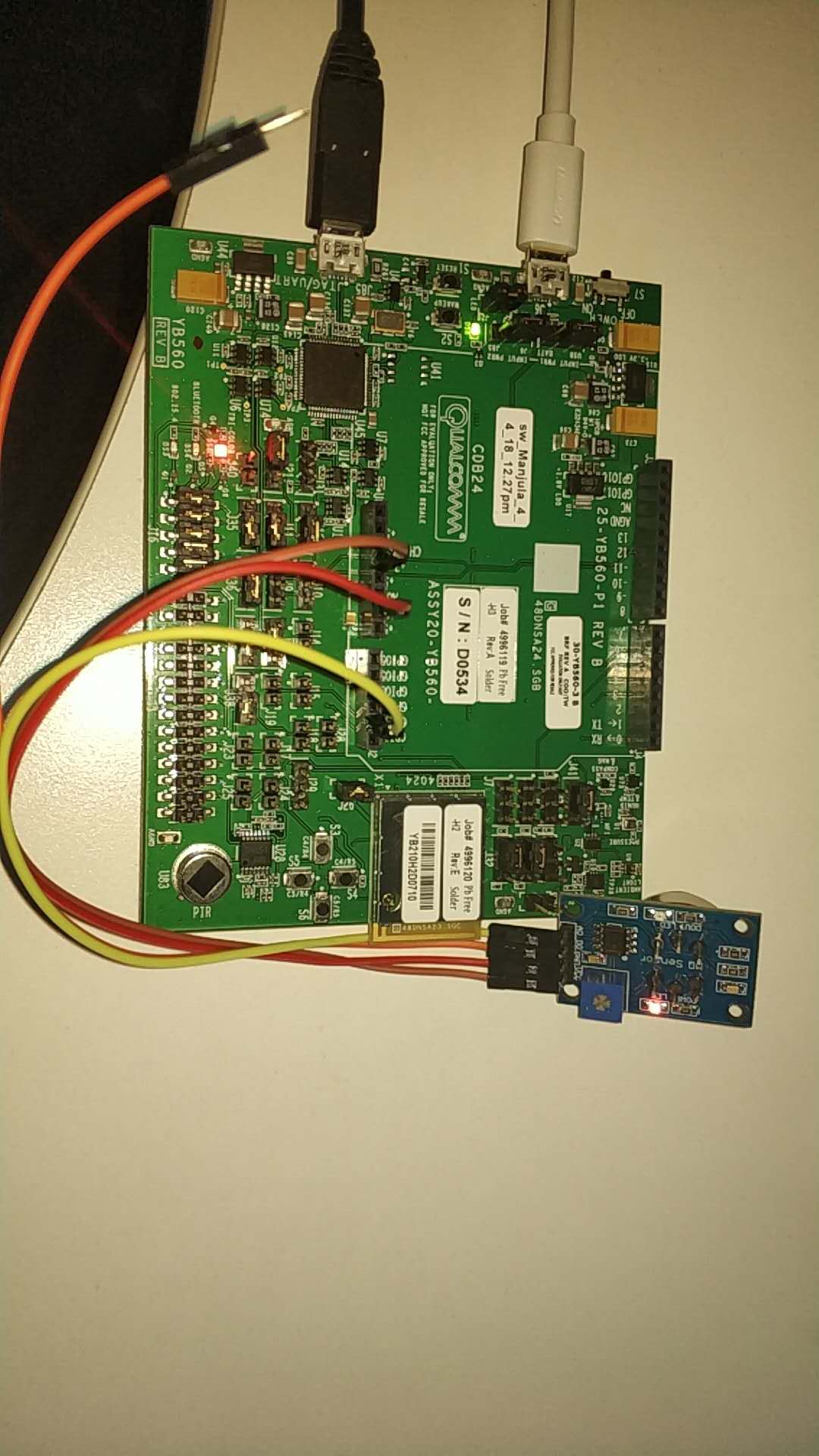


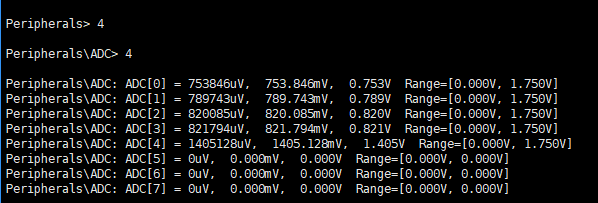
This shows that the compilation was successful.

2. Read the ADC data value of the smoke sensor  
2.1 Single reading of ADC data  
 In reading the data of the sensor, the value of the sensor can be read directly by using the command, and the instruction 16 is input, and the display interface is as follows:



Connect the sensor to the board and enter command 4 4 to read the smoke sensor data as shown:





This method of operation can only read the data of the adc once, and can not read the data of the ADC continuously, so the code needs to be modified accordingly.

2.2 Continuous reading of ADC data  
 In the process of reading the ADC data in a single time, the relevant data instructions are used, so the continuous reading of the values can be realized by the continuous instruction input. Queryed in the demo source code qal.c, thread  
QCLI\_Thread implements the instruction receiving and processing. The QCLI\_Process\_Input\_Data() function is responsible for receiving and processing related instructions. Therefore, the modification is mainly performed in this thread, and the modifications are as follows:  
First, the QCLI\_Adc function will be encapsulated, and the data read by the adc demo will be put into the thread, and the interface for the future smoke value judgment will be written, and the modifications are as follows:  
Modify in adc\_demo.c, add Qcli\_Adc interface, and add print interface

+uint32\_t Qcli\_Adc()

+{

+ qapi\_ADC\_Read\_Result\_t \*result;

+ uint32\_t i;

+ //for (i=0; i < MAX\_CHANNELS; i++)

+ //{

+ result = &chan\_result[7].chan\_result;

+

+ QCLI\_Printf(qcli\_adc\_group, "ADC[7] = %ddmV \n", result->microvolts/1000);

+ return result->microvolts/1000;

+ //}

+}

Add a print interface:

+void printf\_a(uint32\_t result\_adc)

+{

+ QCLI\_Printf(qcli\_adc\_group, "ADC = %ddmV \n", result\_adc);

+}

Add an interface function in adc\_demo.h and reference it in pal.c

**diff --git a/src/adc/adc\_demo.h b/src/adc/adc\_demo.h**

**index 60d9812..26e2869 100644**

**--- a/src/adc/adc\_demo.h**

**+++ b/src/adc/adc\_demo.h**

@@ -15,6 +15,8 @@ typedef struct \_adc\_result {

} adc\_result\_t;

extern adc\_result\_t chan\_result[MAX\_CHANNELS];

+extern uint32\_t Qcli\_Adc();

+extern void printf\_a(uint32\_t result\_adc);

/\*\*

@brief This function registers the ADC demo commands with QCLI.

**diff --git a/src/qcli/pal.c b/src/qcli/pal.c**

**index f8c833d..bd18e6c 100644**

**--- a/src/qcli/pal.c**

**+++ b/src/qcli/pal.c**

@@ -45,6 +45,7 @@

#include "ecosystem\_demo.h"

#include "json\_demo.h"

#include "kpi\_demo.h"

+#include "adc\_demo.h"

#ifdef CONFIG\_QMESH\_DEMO

#include "qmesh\_demo\_menu.h"

According to the previous instruction, in the whir loop statement, the value of the smoke sensor can be read all the time. Each time the sleep is read for 200ms, the code is as follows:

while(1) {

+ QCLI\_Process\_Input\_Data(2, "16");

+ Process\_Command();

+ clear\_buffer();

+

+ //QCLI\_Process\_Input\_Data(1, " ");

+ QCLI\_Process\_Input\_Data(1, "4");

+ Process\_Command();

+

+ //QCLI\_Process\_Input\_Data(1, " ");

+ QCLI\_Process\_Input\_Data(1, "4");

+ Process\_Command();

+

**+ ADC=Qcli\_Adc();**

**+ printf\_a(ADC);**

+ //QCLI\_Process\_Input\_Data(1, " ");

+ QCLI\_Process\_Input\_Data(1, "3");

+ Process\_Command();

+ qurt\_thread\_sleep(200);

+ };

3. Control the high and low levels of GPIO by the value of the smoke sensor  
 By reading the value of the ADC, it is generally 850 mv in the case of smoke, so this value is set to the alarm threshold. Above this threshold is to control GPIO4 to be high, below which GPIO4 is set low. Set the GPIO high and low level, you can still use the command to get the code, the code is as follows:

+ if(ADC > 850)

+ {

+ QCLI\_Process\_Input\_Data(1, "3");

+ Process\_Command();

+ clear\_buffer();

+

+ QCLI\_Process\_Input\_Data(2, "16");

+ Process\_Command();

+ clear\_buffer();

+

+ QCLI\_Process\_Input\_Data(2, "11");

+ Process\_Command();

+ clear\_buffer();

+

+

+ QCLI\_Process\_Input\_Data(20, "gpio\_tlmm 4 10 1 2 1");

+ Process\_Command();

+ clear\_buffer();

+

+ } else {

+

+ QCLI\_Process\_Input\_Data(1, "3");

+ Process\_Command();

+ clear\_buffer();

+

+ QCLI\_Process\_Input\_Data(2, "16");

+ Process\_Command();

+ clear\_buffer();

+

+ QCLI\_Process\_Input\_Data(2, "11");

+ Process\_Command();

+ clear\_buffer();

+

+ QCLI\_Process\_Input\_Data(20, "gpio\_tlmm 4 10 1 1 0");

+ Process\_Command();

+ clear\_buffer();

+ }