Conectando dispositivos I2C a tu sistema

Circuito Inter-Integrado (I²C o más a menudo también escrito como I2C) es un bus multimaestro en serie de un solo extremo inventado por la división de semiconductores de Philips (consulte el artículo de wikipedia para obtener más información I2C) y de uso común en muchos dispositivos electrónicos modernos, incluyendo PC. I²C usa solo dos lineas bidireccionales open-drain, Serial Data Line (SDA) y Serial Clock Line(SCL), llevados al nivel lógico 1 por resistencias de pullup. Típicamente son empleadas tensiones de +5 Volt o +3,3 Volt aunque sistemas con otras tensiones son permitidas. Esto tiene una implicación interesante: el nivel lógico 1 es logrado sin hacer nada, mientras que el nivel lógico 0 debe ser llevado a tierra. Aunque existen desplazadores de nivel de tensión para I2C bidireccionales (PCA9306) puede ser factible experimentar soluciones más simples si en el bus I2C no va a tener muchos dispositivos conectados. Para más información, consulte la sección “Desplazamiento del nivel de tensión”.

Prefacio

La mayoría de las computadoras personales (PC) modernas tienen componentes internos que comunican información vital, por ejemplo temperaturas criticas de componentes, sobre un bus I2C. Este tipo de cosas están conectadas de fábrica en su PC y pueden ser tratadas con lm_sensors... lo que queremos hacer aquí es usar un bus I2C en su computadora para conectar algún sensor I2C externo como un acelerómetros. Estoy etiquetando esto en la sección de hardware de ARM porque creo que, excluyendo el tema de los sensores lm, la mayoría de la gente estará haciendo este tipo de cosas en sistemas ARM embebidos... pero los conceptos son aplicables a cualquier sistema que admita Linux con un bus I2C.

Preparando el sistema principal

Antes de empezar, es posible que desee asegurarse de que el sistema operativo tiene todo lo necesario para gestionar el bus I2C que va a utilizar. Lo primero que hay que hacer es asegurarse de que tiene el controlador de kernel correcto para cualquier implementación de capa física en su sistema. Tendrás que investigar en las hojas de datos del hardware de tu sistema... mi Pi tiene bcm2708 así que en mi caso se trataba de cargar el módulo bcm2708_i2c. También puede ser necesario cargar el módulo i2c-dev dependiendo de su configuración.

Una vez que tenga los controladores correctos, es posible que desee disponer de una herramienta de espacio de usuario que le ayude a detectar buses, presentar dispositivos y comunicarse con los dispositivos I2C de cada bus. Utilizo i2ctools para esto pero no está empaquetado entre los paquetes de Slackware y no pude encontrar un tercero que ofreciera un paquete de ARM Slackware, así que descargué las fuentes y las compilé para mí mismo. Las fuentes se pueden obtener desde aquí: I2CTools.

Conectando un dispositivo sobre el bus I2C

Siempre que haya resuelto los problemas de nivel de tensión (consulte el capítulo “Desplazamiento del nivel de tensión”), añadir un nuevo dispositivo en el bus es muy sencillo. El bus es multimaestro, lo que significa que puedes tener muchos dispositivos (hasta 101) en el mismo bus, así que todo lo que tienes que hacer es hacer 4 conexiones: Potencia, tierra, SDA y SCL. Si es el primer dispositivo que conecta en el bus, puede ser necesario instalar resistencias de pullups entre Power-SDA y Power-SCL. Es tan simple como eso y si el sistema está listo con los controladores apropiados y las utilidades del país del usuario, usted está listo para acceder al dispositivo recién conectado.

Detectando dispositivos conectados

Hay probablemente muchas formas para determinar que está conectado a un bus I2C. Elegí usar cosas del proyecto i2ctools. No pude encontrar un paquete Slackware ARM para i2ctools, así que lo compilé e instalé en mi sistema.

Lo primero que debe saber es qué buses I2C están presentes en su sistema, ya que puede haber más de uno y mirar en el bus equivocado puede ser frustrante:

```
root@pi:~# i2cdetect  -l
i2c-0   i2c             bcm2708_i2c.0                           I2C adapter
i2c-1   i2c             bcm2708_i2c.1                           I2C adapter
root@pi:~#
```

Si no estás seguro de en qué bus conectaste tus cosas, es posible que quieras hacer esto en todos los buses:

```
root@pi:~# i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 69 -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
root@pi:~#
```

Comunicación con un dispositivo I2C

La comunicación con dispositivos I2C se realiza leyendo y escribiendo en sus registros. Cada dispositivo tiene su propia lista de registros y es algo que debe buscar en la hoja de datos del dispositivo. Algunos dispositivos incluso necesitan que se realice una calibración preliminar antes de

que pueda leer cualquier dato sensible de ellos,, así que antes de empezar a usar i2cget para leer algunos registros, debe tener al menos una idea de los registros que le interesan y averiguar si su dispositivo necesita calibración antes de leer cualquier dato sensible. Los registros se pueden establecer utilizando i2cset, pero hágalo sólo si ha leído la hoja de datos.

Incluso una vez que sepa si su dispositivo necesita calibración y los registros involucrados, el contenido de los registros puede no estar en un formato conveniente para su uso inmediato. Generalmente hay varios scripts en perl o python que tratan de la calibración y gestión de datos de dispositivos específicos de I2C para que se pueda producir información legible para el ser humano.

En este punto es imposible mostrar más información sobre la comunicación I1C sin entrar en detalles sobre un dispositivo específico, así que voy a elegir uno de los dispositivos de mi pcb IMU. Al detectar previamente en mi PI encontré 4 dispositivos con las direcciones hexadecimales fluyentes: 1e,40,69 y 77. Generalmente cada dispositivo I2C tiene un rango de direcciones que pueden ser configuradas. Nos concentraremos en el dispositivo con dirección 0x69. Haciendo una búsqueda de referencias cruzadas en la hoja de datos de la pcb IMU y en las hojas de datos individuales de cada dispositivo presente en mi IMU, se revela que 0x69 debe ser la dirección de la ITG3200 (giroscopio + sensor de temperatura) y, de hecho, la hoja de datos de la ITG3200 afirma que puede tener una dirección de 0x68 o 0x69 seleccionable por nivel lógico en el pin 9. No que nosotros necesitamos es la tabla del ITG3200:

```
Addr  Addr     Register Name R/W
Hex  Decimal
  0     0      WHO_AM_I      R/W
 15    21     SMPLRT_DIV     R/W
 16    22       DLPF_FS      R/W
 17    23       INT_CFG      R/W
 1A    26     INT_STATUS     R
 1B    27     TEMP_OUT_H     R                          TEMP_OUT_H
 1C    28     TEMP_OUT_L     R                           TEMP_OUT_L
 1D    29    GYRO_XOUT_H     R                         GYRO_XOUT_H
 1E    30    GYRO_XOUT_L     R                         GYRO_XOUT_L
 1F    31    GYRO_YOUT_H     R                         GYRO_YOUT_H
 20    32    GYRO_YOUT_L     R                         GYRO_YOUT_L
 21    33    GYRO_ZOUT_H     R                         GYRO_ZOUT_H
 22    34    GYRO_ZOUT_L     R                          GYRO_ZOUT_L
 3E    62       PWR_MGM      R/W
```

Elegí el ITG3200 por que tiene un sensor de temperatura incorporado y espero poder leerlo sin tener que hacer ninguna calibración, sólo para mantener el ejemplo lo más simple posible. De acuerdo a la tabla, la dirección del registro de temperatura son 1b y 1c, así que vamos a intentar sacar algunos datos de ahí:

```
root@pi:~# i2cdump  -y -r 0x1b-0x1c 1 0x69 b
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
10:                               c0 90                         ??
root@pi:~#
```

El ejemplo de arriba muestra los registros de volcado 1b y 1c de la ITG3200, se puede lograr el mismo resultado con i2cget:

```
root@pi:~# i2cget -y  1 0x69  0x1b b
0xc0
root@pi:~# i2cget -y  1 0x69  0x1c b
0x90
root@pi:~#
```

así que tenemos el registro c090 como nuestro registro de temperatura. De acuerdo a la hoja de datos este es representado como un complemento a 2 de la temperatura. Así que vamos a tratar de averiguar qué sería eso: c090 es escrito en binario es 1100000010010000, el bit más significativo es 1 entonces el resultado debería ser:

```
16528 - 32768 = -16240
```

No pude encontrar en la hoja de datos en qué unidades se encuentra esta lectura, pero sí mencionaron que había una compensación promedio de 13200. Hice una pequeña búsqueda en Google y encontré esta fórmula:

```
35 + ((raw value + 13200) / 280))
35 + ((13200 - 16240)/280) = 24.14
```

Considerando que mi temperatura ambiente actual es cerca de 20 grados Celcius supongo que para los datos no calibrados está bien.

Si quieres un script que haga las matemáticas para ti y sólo leer la salida del sensor ITG32000 en un formato legible por un humano aquí tienes un ejemplo:

```
#!/bin/bash
BUS=1
ID=0x69
ATH=0x1b
ATL=0x1c
ARXH=0x1d
ARXL=0x1e
ARYH=0x1f
ARYL=0x20
ARZH=0x21
ARZL=0x22

#need upper case hex stripped of prefix "0x" or bc will not like the input
for VAR in TH TL RXH RXL RYH RYL RZH RZL
do
  CMD="$VAR=\$(i2cget -y $BUS $ID \$A$VAR  b |sed -e "s/^0x//" |tr "a-z" "A-
Z")"
  eval $CMD
  eval "echo $VAR = \$$VAR"
done

echo "Temp register: $(echo "ibase=16; $TH$TL" | bc -l) 0x$TH$TL ($(echo
"ibase=16; obase=2; $TH$TL" | bc -l))"
```

```
echo -n "Temp in Celcius: "
#this takes hex input and evaluates the followin formula in hexadecimal
#temp= 35 + ((raw + 13200) / 280))"
#where raw is the input reading un 2's compliment
#(to uncompliment the input I take away 0x10000 if input is larger then
0x8000)
echo "ibase=16; input=$TH$TL; if ( input >= 8000 ) { raw=input - 10000;}
else { raw=input;}; 23 + ((raw + 3390)/118);" |bc -l

echo "X Axis Rotation Register: $(echo "ibase=16; $RXH$RXL" | bc -l)
0x$RXH$RXL ($(echo "ibase=16; obase=2; $RXH$RXL" | bc -l))"
echo -n "X Axis Angula velocity degree/sec: "
echo "ibase=16; input=$RXH$RXL; if ( input >= 8000 ) { raw= input - 10000;}
else { raw=input;}; raw / E.177" |bc -l

echo "Y Axis Rotation Register: $(echo "ibase=16; $RYH$RYL" | bc -l)
0x$RYH$RYL ($(echo "ibase=16; obase=2; $RYH$RYL" | bc -l))"
echo -n "Y Axis Angula velocity degree/sec: "
echo "ibase=16; input=$RYH$RYL; if ( input >= 8000 ) { raw= input - 10000;}
else { raw=input;}; raw / E.177" |bc -l

echo "Z Axis Rotation Register: $(echo "ibase=16; $RZH$RZL" | bc -l)
0x$RZH$RZL ($(echo "ibase=16; obase=2; $RZH$RZL" | bc -l))"
echo -n "Z Axis Angula velocity degree/sec: "
echo "ibase=16; input=$RZH$RZL; if ( input >= 8000 ) { raw= input - 10000;}
else { raw=input;}; raw / E.177" |bc -l
```

El script anterior sólo hace un simple volcado de los datos de registro y convierte los valores en un formato legible para el ser humano, no hace nada con respecto a la calibración y el promedio de las vibraciones. Se lograrían lecturas giroscópicas más consistentes si se hiciera un promedio de 10 muestras de datos consecutivos, lo que permitiría obtener un promedio de la mayoría de las vibraciones ambientales.

Un script en bash no es realmente la forma más adecuada de leer datos de dispositivos I2C, una forma más rápida de gestionar los datos de los dispositivos es realmente obligatorio para realizar calibraciones, promedios de datos y, lo que es más, para que la información sea coherente y útil para cálculos posteriores. La documentación del núcleo de Linux para i2c me pareció una referencia útil (<kernel source tree>/Documentation/i2c/dev-interface); no es la única manera de que los datos puedan ser leídos, pero es un buen punto de partida.

Odio mostrar mis pobres capacidades de programación en C, pero aquí hay un código que usa i2c-dev para leer cosas de la ITG3200 y toma un promedio de más de 10 lecturas:

```
#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>
```

```
#include <errno.h>

#define I2C_DEVICE "/dev/i2c-1"

/*ITG3200*/
#define ITG3200_ADDR 0x69
#define ITG3200_SELF 0x0
#define ITG3200_INT 0x1a
#define ITG3200_TH 0x1b /*2 bytes Hight byte and Low byte*/
#define ITG3200_TL 0x1c
#define ITG3200_XRH 0x1d /*2 byte Hight byte and Low byte*/
#define ITG3200_XRL 0x1e
#define ITG3200_YRH 0x1f /*2 byte Hight byte and Low byte*/
#define ITG3200_YRL 0x20
#define ITG3200_ZRH 0x21 /*2 byte Hight byte and Low byte*/
#define ITG3200_ZRL 0x22 /*2 byte Hight byte and Low byte*/
#define ITG3200_TEMP_RAW_OFFSET 13200
#define ITG3200_TEMP_RAW_SENSITIVITY 280
#define ITG3200_TEMP_OFFSET 35
#define ITG3200_ROT_RAW_SENSITIVITY 14.375

int twosc2int(int twoscomplimentdata)
{ int retval;
  if( twoscomplimentdata > 32768 ) retval = twoscomplimentdata - 65536;
  else retval = twoscomplimentdata;
  return retval;
}

float ITG3200_rot_conv(int rawdata)
{ float retval;
  int raw;

  raw=twosc2int(rawdata);
  retval = (float)raw / (float)ITG3200_ROT_RAW_SENSITIVITY;
  return retval;
}

float ITG3200_temp_conv(int rawdata)
{ float retval;
  int raw;

  raw=twosc2int(rawdata);
  retval = (float)ITG3200_TEMP_OFFSET + (((float)raw +
ITG3200_TEMP_RAW_OFFSET) / ITG3200_TEMP_RAW_SENSITIVITY);
  return retval;
}

void ITG3200_read (int file, int *raw, int *reg_array,int size)
{ __s32 res;
  int i,j,k;
```

```
  for(i=0;i<size;i++)
  { k=0;
    for (j=0;j<2;j++)
    {
      if( (res = i2c_smbus_read_byte_data(file,*(reg_array + i + j)) )< 0 )
      { printf("Failed to read from the i2c bus.\n");
        exit(1);
      }
      if (j == 0) k=(int)res << 8;
      else
      { k += (int)res;
        *(raw + (i/2))=k;
      }
    }
    i++;
  }
}

main ()
{ int file;
  int i,j,k;
  float data[4]={0};

  int ITG3200_REGS[8]={ITG3200_TH,ITG3200_TL,ITG3200_XRH,ITG3200_XRL,
    ITG3200_YRH, ITG3200_YRL,ITG3200_ZRH,ITG3200_ZRL};
  int ITG3200_RAW_DATA[4];
  float ITG3200_DATA[4];

  if ((file = open(I2C_DEVICE, O_RDWR)) < 0)
  { perror("Failed to open the i2c bus");
    exit(1);
  }

  if (ioctl(file, I2C_SLAVE, ITG3200_ADDR) < 0)
  { printf("Failed to acquire bus access and/or talk to slave.\n");
    exit(1);
  }

/*Take an avarage over 10 consecuitve readings on the ITG3200*/
  for (i=0;i<10;i++)
  {
ITG3200_read(file,&ITG3200_RAW_DATA[0],&ITG3200_REGS[0],sizeof(ITG3200_REGS)
/sizeof(ITG3200_REGS[0]));

    data[0] += ITG3200_temp_conv(ITG3200_RAW_DATA[0]);
    data[1] += ITG3200_rot_conv(ITG3200_RAW_DATA[1]);
    data[2] += ITG3200_rot_conv(ITG3200_RAW_DATA[2]);
    data[3] += ITG3200_rot_conv(ITG3200_RAW_DATA[3]);
  }
  for(i=0;i<4;i++) data[i] /= 10;
```

```
  printf("Temp. : %2.2f \n",data[0]);
  printf("Rot. X : %2.2f \n",data[1]);
  printf("Rot. Y : %2.2f \n",data[2]);
  printf("Rot. Z : %2.2f \n",data[3]);

  close(file);
}
```

Desplazamiento del nivel de tensión

Usted puede terminar con dispositivos de nivel de voltaje heterogéneo y si tiene muchos dispositivos, la manera correcta de solucionar este problema es usando desplazadores de nivel de voltaje I2C bidireccionales como el, PCA9306, pero si sólo tiene unos pocos dispositivos agrupados en una placa de circuito impreso como la unidad IMU 10DOF, es posible que desee probar un sistema más sencillo. Así es como conecté mi PCB IMU de 5V a un bus I2C de 3.3V en mi Raspberry PI:

Tomé como supuesto que 4.4 V (5V con un diodo en serie 1N4148) seguiría siendo una tensión de alimentación tolerable para toda la pcb IMU, lo que probablemente permitiría a todos los dispositivos I2C de la pcb IMU reconocer un mínimo de 3.08 V. (4.4 * 0.7) como la tensión de nivel lógico 1 más baja y fiable, lo que le permitiría interoperar con los niveles lógicos de 3.3V de la Raspberry PI. No pude encontrar si la Raspberry Pi tiene Resistencias de pullups internas en el bus I2C o si tienen que ser colocados externamente, así que por las dudas pongo pullups de 10k entre la línea de alimentación de 4.4V y las dos líneas de datos. Fui capaz de detectar correctamente los sensores de la PCB IMU.

Fuentes

- Escrito originalmente por louigi600.
- Traducido por — *rramp 2019/07/16 22:51 (UTC)*.

howtos, hardware, arm, author louigi600