|  |
| --- |
| /\* |
|  | \* This file is an MPU6050 demonstration. |
|  | \* https://openest.io/en/2020/01/21/mpu6050-accelerometer-on-raspberry-pi/ |
|  | \* Copyright (c) 2020 Julien Grossholtz - https://openest.io. |
|  | \* |
|  | \* This program is free software: you can redistribute it and/or modify |
|  | \* it under the terms of the GNU General Public License as published by |
|  | \* the Free Software Foundation, version 3. |
|  | \* |
|  | \* This program is distributed in the hope that it will be useful, but |
|  | \* WITHOUT ANY WARRANTY; without even the implied warranty of |
|  | \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU |
|  | \* General Public License for more details. |
|  | \* |
|  | \* You should have received a copy of the GNU General Public License |
|  | \* along with this program. If not, see <http://www.gnu.org/licenses/>. |
|  | \*/ |
|  |  |
|  | #include <linux/i2c-dev.h> |
|  | #include <sys/ioctl.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <stdlib.h> |
|  | #include <stdio.h> |
|  | #include <unistd.h> |
|  | #include <stdint.h> |
|  |  |
|  | #define MPU6050\_I2C\_ADDR 0x68 |
|  |  |
|  | #define REG\_ACCEL\_ZOUT\_H 0x3F |
|  | #define REG\_ACCEL\_ZOUT\_L 0x40 |
|  | #define REG\_PWR\_MGMT\_1 0x6B |
|  | #define REG\_ACCEL\_CONFIG 0x1C |
|  | #define REG\_SMPRT\_DIV 0x19 |
|  | #define REG\_CONFIG 0x1A |
|  | #define REG\_FIFO\_EN 0x23 |
|  | #define REG\_USER\_CTRL 0x6A |
|  | #define REG\_FIFO\_COUNT\_L 0x72 |
|  | #define REG\_FIFO\_COUNT\_H 0x73 |
|  | #define REG\_FIFO 0x74 |
|  | #define REG\_WHO\_AM\_I 0x75 |
|  |  |
|  | int file = -1; |
|  |  |
|  | // Please note, this is not the recommanded way to write data |
|  | // to i2c devices from user space. |
|  | void i2c\_write(\_\_u8 reg\_address, \_\_u8 val) { |
|  | char buf[2]; |
|  | if(file < 0) { |
|  | printf("Error, i2c bus is not available\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  | buf[0] = reg\_address; |
|  | buf[1] = val; |
|  |  |
|  | if (write(file, buf, 2) != 2) { |
|  | printf("Error, unable to write to i2c device\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | // Please note, this is not thre recommanded way to read data |
|  | // from i2c devices from user space. |
|  | char i2c\_read(uint8\_t reg\_address) { |
|  | char buf[1]; |
|  | if(file < 0) { |
|  | printf("Error, i2c bus is not available\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  | buf[0] = reg\_address; |
|  |  |
|  | if (write(file, buf, 1) != 1) { |
|  | printf("Error, unable to write to i2c device\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  |  |
|  | if (read(file, buf, 1) != 1) { |
|  | printf("Error, unable to read from i2c device\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  | return buf[0]; |
|  |  |
|  | } |
|  |  |
|  | uint16\_t merge\_bytes( uint8\_t LSB, uint8\_t MSB) { |
|  | return (uint16\_t) ((( LSB & 0xFF) << 8) | MSB); |
|  | } |
|  |  |
|  | // 16 bits data on the MPU6050 are in two registers, |
|  | // encoded in two complement. So we convert those to int16\_t |
|  | int16\_t two\_complement\_to\_int( uint8\_t LSB, uint8\_t MSB) { |
|  | int16\_t signed\_int = 0; |
|  | uint16\_t word; |
|  |  |
|  | word = merge\_bytes(LSB, MSB); |
|  |  |
|  | if((word & 0x8000) == 0x8000) { // negative number |
|  | signed\_int = (int16\_t) -(~word); |
|  | } else { |
|  | signed\_int = (int16\_t) (word & 0x7fff); |
|  | } |
|  |  |
|  | return signed\_int; |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | int adapter\_nr = 1; /\* probably dynamically determined \*/ |
|  | char bus\_filename[250]; |
|  | char accel\_x\_h,accel\_x\_l,accel\_y\_h,accel\_y\_l,accel\_z\_h,accel\_z\_l,temp\_h,temp\_l; |
|  | uint16\_t fifo\_len = 0; |
|  | int16\_t x\_accel = 0; |
|  | int16\_t y\_accel = 0; |
|  | int16\_t z\_accel = 0; |
|  | int16\_t temp = 0; |
|  | float x\_accel\_g, y\_accel\_g, z\_accel\_g, temp\_f; |
|  |  |
|  | snprintf(bus\_filename, 250, "/dev/i2c-1", adapter\_nr); |
|  | file = open(bus\_filename, O\_RDWR); |
|  | if (file < 0) { |
|  | /\* ERROR HANDLING; you can check errno to see what went wrong \*/ |
|  | exit(1); |
|  | } |
|  |  |
|  |  |
|  | if (ioctl(file, I2C\_SLAVE, MPU6050\_I2C\_ADDR) < 0) { |
|  | /\* ERROR HANDLING; you can check errno to see what went wrong \*/ |
|  | exit(1); |
|  | } |
|  |  |
|  | i2c\_write(REG\_PWR\_MGMT\_1, 0x01); |
|  | i2c\_write(REG\_ACCEL\_CONFIG, 0x00); |
|  | i2c\_write(REG\_SMPRT\_DIV, 0x07); |
|  | i2c\_write(REG\_CONFIG, 0x00); |
|  | i2c\_write(REG\_FIFO\_EN, 0x88); |
|  | i2c\_write(REG\_USER\_CTRL, 0x44); |
|  |  |
|  | while(fifo\_len != 1024) { |
|  | accel\_x\_h = i2c\_read(REG\_FIFO\_COUNT\_L); |
|  | accel\_x\_l = i2c\_read(REG\_FIFO\_COUNT\_H); |
|  | fifo\_len = merge\_bytes(accel\_x\_h,accel\_x\_l); |
|  |  |
|  | if(fifo\_len == 1024) { |
|  | printf("fifo overflow !\n"); |
|  | i2c\_write(REG\_USER\_CTRL, 0x44); |
|  | continue; |
|  | } |
|  |  |
|  | if(fifo\_len >= 8) { |
|  | accel\_x\_h = i2c\_read(REG\_FIFO); |
|  | accel\_x\_l = i2c\_read(REG\_FIFO); |
|  | accel\_y\_h = i2c\_read(REG\_FIFO); |
|  | accel\_y\_l = i2c\_read(REG\_FIFO); |
|  | accel\_z\_h = i2c\_read(REG\_FIFO); |
|  | accel\_z\_l = i2c\_read(REG\_FIFO); |
|  | temp\_h = i2c\_read(REG\_FIFO); |
|  | temp\_l= i2c\_read(REG\_FIFO); |
|  |  |
|  | x\_accel= two\_complement\_to\_int(accel\_x\_h,accel\_x\_l); |
|  | x\_accel\_g = ((float) x\_accel)/16384; |
|  |  |
|  | y\_accel= two\_complement\_to\_int(accel\_y\_h,accel\_y\_l); |
|  | y\_accel\_g = ((float) y\_accel)/16384; |
|  |  |
|  | z\_accel= two\_complement\_to\_int(accel\_z\_h,accel\_z\_l); |
|  | z\_accel\_g = ((float) z\_accel)/16384; |
|  |  |
|  | temp = two\_complement\_to\_int(temp\_h, temp\_l); |
|  | temp\_f = (float)temp/340 + 36.53; // calculated as described in the MPU60%) register map document |
|  |  |
|  | printf("x\_accel %.3fg y\_accel %.3fg z\_accel %.3fg temp=%.1fc \r", x\_accel\_g, y\_accel\_g, z\_accel\_g, temp\_f); |
|  | } else { |
|  | usleep(10000); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | return 0; |
|  | } |