[Google Git](https://android.googlesource.com/)

[Sign in](https://accounts.google.com/AccountChooser?service=gerritcodereview&continue=https://android.googlesource.com/login/kernel/msm.git/%2B/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c)

[android](https://android.googlesource.com/?format=HTML) / [kernel](https://android.googlesource.com/kernel/) / [msm.git](https://android.googlesource.com/kernel/msm.git/) / [eaf36994a3992b8f918c18e4f7411e8b2320a35f](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f) / [.](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/) / [drivers](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers) / [misc](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc) / [mpu6050](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050) / **mldl\_cfg.c**

blob: 7f806f33f2e29e8a5e6ff6f75f0e7b3edac55cf5 [[file](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c)] [[log](https://android.googlesource.com/kernel/msm.git/+log/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c)] [[blame](https://android.googlesource.com/kernel/msm.git/+blame/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c)]

|  |  |
| --- | --- |
|  | /\* |
|  | $License: |
|  | Copyright (C) 2011 InvenSense Corporation, All Rights Reserved. |
|  | Copyright (C) 2012 Sony Mobile Communications AB. |
|  |  |
|  | 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; either version 2 of the License, or |
|  | (at your option) any later version. |
|  |  |
|  | 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/>. |
|  | $ |
|  | \*/ |
|  |  |
|  | /\*\* |
|  | \* @addtogroup MLDL |
|  | \* |
|  | \* @{ |
|  | \* @file mldl\_cfg.c |
|  | \* @brief The Motion Library Driver Layer. |
|  | \*/ |
|  |  |
|  | /\* -------------------------------------------------------------------------- \*/ |
|  | #include <linux/delay.h> |
|  | #include <linux/slab.h> |
|  |  |
|  | #include <stddef.h> |
|  |  |
|  | #include "mldl\_cfg.h" |
|  | #include <linux/mpu.h> |
|  | #include "mpu6050.h" |
|  |  |
|  | #include "mlsl.h" |
|  | #include "mldl\_print\_cfg.h" |
|  | #include "log.h" |
|  | #undef MPL\_LOG\_TAG |
|  | #define MPL\_LOG\_TAG "mldl\_cfg:" |
|  |  |
|  | /\* -------------------------------------------------------------------------- \*/ |
|  |  |
|  | #define SLEEP 0 |
|  | #define WAKE\_UP 7 |
|  | #define RESET 1 |
|  | #define STANDBY 1 |
|  |  |
|  | #define CHARGEPUMP\_WAKE 10 |
|  |  |
|  | /\* -------------------------------------------------------------------------- \*/ |
|  |  |
|  | /\*\* |
|  | \* @brief Stop the DMP running |
|  | \* |
|  | \* @return INV\_SUCCESS or non-zero error code |
|  | \*/ |
|  | static int dmp\_stop(struct mldl\_cfg \*mldl\_cfg, void \*gyro\_handle) |
|  | { |
|  | unsigned char user\_ctrl\_reg; |
|  | int result; |
|  |  |
|  | if (mldl\_cfg->inv\_mpu\_state->status & MPU\_DMP\_IS\_SUSPENDED) |
|  | return INV\_SUCCESS; |
|  |  |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, 1, &user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | user\_ctrl\_reg = (user\_ctrl\_reg & (~BIT\_FIFO\_EN)) | BIT\_FIFO\_RST; |
|  | user\_ctrl\_reg = (user\_ctrl\_reg & (~BIT\_DMP\_EN)) | BIT\_DMP\_RST; |
|  |  |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status |= MPU\_DMP\_IS\_SUSPENDED; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @brief Starts the DMP running |
|  | \* |
|  | \* @return INV\_SUCCESS or non-zero error code |
|  | \*/ |
|  | static int dmp\_start(struct mldl\_cfg \*mldl\_cfg, void \*mlsl\_handle) |
|  | { |
|  | unsigned char user\_ctrl\_reg; |
|  | int result; |
|  |  |
|  | if ((!(mldl\_cfg->inv\_mpu\_state->status & MPU\_DMP\_IS\_SUSPENDED) && |
|  | mldl\_cfg->mpu\_gyro\_cfg->dmp\_enable) |
|  | || |
|  | ((mldl\_cfg->inv\_mpu\_state->status & MPU\_DMP\_IS\_SUSPENDED) && |
|  | !mldl\_cfg->mpu\_gyro\_cfg->dmp\_enable)) |
|  | return INV\_SUCCESS; |
|  |  |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, 1, &user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, |
|  | ((user\_ctrl\_reg & (~BIT\_FIFO\_EN)) |
|  | | BIT\_FIFO\_RST)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, 1, &user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | user\_ctrl\_reg |= BIT\_DMP\_EN; |
|  |  |
|  | if (mldl\_cfg->mpu\_gyro\_cfg->fifo\_enable) |
|  | user\_ctrl\_reg |= BIT\_FIFO\_EN; |
|  | else |
|  | user\_ctrl\_reg &= ~BIT\_FIFO\_EN; |
|  |  |
|  | user\_ctrl\_reg |= BIT\_DMP\_RST; |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, user\_ctrl\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_DMP\_IS\_SUSPENDED; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @brief enables/disables the I2C bypass to an external device |
|  | \* connected to MPU's secondary I2C bus. |
|  | \* @param enable |
|  | \* Non-zero to enable pass through. |
|  | \* @return INV\_SUCCESS if successful, a non-zero error code otherwise. |
|  | \*/ |
|  | static int mpu6050b1\_set\_i2c\_bypass(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*mlsl\_handle, unsigned char enable) |
|  | { |
|  | unsigned char reg; |
|  | int result; |
|  | unsigned char status = mldl\_cfg->inv\_mpu\_state->status; |
|  | if ((status & MPU\_GYRO\_IS\_BYPASSED && enable) || |
|  | (!(status & MPU\_GYRO\_IS\_BYPASSED) && !enable)) |
|  | return INV\_SUCCESS; |
|  |  |
|  | /\*---- get current 'USER\_CTRL' into b ----\*/ |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, 1, &reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (!enable) { |
|  | /\* setting int\_config with the property flag BIT\_BYPASS\_EN |
|  | should be done by the setup functions \*/ |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_INT\_PIN\_CFG, |
|  | (mldl\_cfg->pdata->int\_config & ~(BIT\_BYPASS\_EN))); |
|  | if (!(reg & BIT\_I2C\_MST\_EN)) { |
|  | result = |
|  | inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, |
|  | (reg | BIT\_I2C\_MST\_EN)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | } else if (enable) { |
|  | if (reg & BIT\_AUX\_IF\_EN) { |
|  | result = |
|  | inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_USER\_CTRL, |
|  | (reg & (~BIT\_I2C\_MST\_EN))); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* To avoid hanging the bus we must sleep until all |
|  | \* slave transactions have been completed. |
|  | \* 24 bytes max slave reads |
|  | \* +1 byte possible extra write |
|  | \* +4 max slave address |
|  | \* --- |
|  | \* 33 Maximum bytes |
|  | \* x9 Approximate bits per byte |
|  | \* --- |
|  | \* 297 bits. |
|  | \* 2.97 ms minimum @ 100kbps |
|  | \* 0.75 ms minimum @ 400kbps. |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  | msleep(3); |
|  | } |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_INT\_PIN\_CFG, |
|  | (mldl\_cfg->pdata->int\_config | BIT\_BYPASS\_EN)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | if (enable) |
|  | mldl\_cfg->inv\_mpu\_state->status |= MPU\_GYRO\_IS\_BYPASSED; |
|  | else |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_GYRO\_IS\_BYPASSED; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @brief enables/disables the I2C bypass to an external device |
|  | \* connected to MPU's secondary I2C bus. |
|  | \* @param enable |
|  | \* Non-zero to enable pass through. |
|  | \* @return INV\_SUCCESS if successful, a non-zero error code otherwise. |
|  | \*/ |
|  | static int mpu\_set\_i2c\_bypass(struct mldl\_cfg \*mldl\_cfg, void \*mlsl\_handle, |
|  | unsigned char enable) |
|  | { |
|  | return mpu6050b1\_set\_i2c\_bypass(mldl\_cfg, mlsl\_handle, enable); |
|  | } |
|  |  |
|  |  |
|  | #define NUM\_OF\_PROD\_REVS (ARRAY\_SIZE(prod\_rev\_map)) |
|  | #define NOTFOUND\_PROD\_REVS -1 |
|  |  |
|  | /\* NOTE : when not indicated, product revision |
|  | is considered an 'npp'; non production part \*/ |
|  |  |
|  | /\* produces an unique identifier for each device based on the |
|  | combination of product version and product revision \*/ |
|  | struct prod\_rev\_map\_t { |
|  | unsigned short mpl\_product\_key; |
|  | unsigned char silicon\_rev; |
|  | unsigned short gyro\_trim; |
|  | unsigned short accel\_trim; |
|  | }; |
|  |  |
|  | /\* NOTE: product entries are in chronological order \*/ |
|  | static struct prod\_rev\_map\_t prod\_rev\_map[] = { |
|  | /\* prod\_ver = 0 \*/ |
|  | {MPL\_PROD\_KEY(0, 1), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 2), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 3), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 4), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 5), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 6), MPU\_SILICON\_REV\_A2, 131, 16384}, /\* (A2/C2-1) \*/ |
|  | /\* prod\_ver = 1, forced to 0 for MPU6050 A2 \*/ |
|  | {MPL\_PROD\_KEY(0, 7), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 8), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 9), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 10), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 11), MPU\_SILICON\_REV\_A2, 131, 16384}, /\* (A2/D2-1) \*/ |
|  | {MPL\_PROD\_KEY(0, 12), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 13), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 14), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 15), MPU\_SILICON\_REV\_A2, 131, 16384}, |
|  | {MPL\_PROD\_KEY(0, 27), MPU\_SILICON\_REV\_A2, 131, 16384}, /\* (A2/D4) \*/ |
|  | /\* prod\_ver = 1 \*/ |
|  | {MPL\_PROD\_KEY(1, 16), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D2-1) \*/ |
|  | {MPL\_PROD\_KEY(1, 17), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D2-2) \*/ |
|  | {MPL\_PROD\_KEY(1, 18), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D2-3) \*/ |
|  | {MPL\_PROD\_KEY(1, 19), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D2-4) \*/ |
|  | {MPL\_PROD\_KEY(1, 20), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D2-5) \*/ |
|  | {MPL\_PROD\_KEY(1, 28), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/D4) \*/ |
|  | {MPL\_PROD\_KEY(1, 1), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-1) \*/ |
|  | {MPL\_PROD\_KEY(1, 2), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-2) \*/ |
|  | {MPL\_PROD\_KEY(1, 3), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-3) \*/ |
|  | {MPL\_PROD\_KEY(1, 4), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-4) \*/ |
|  | {MPL\_PROD\_KEY(1, 5), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-5) \*/ |
|  | {MPL\_PROD\_KEY(1, 6), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B1/E1-6) \*/ |
|  | /\* prod\_ver = 2 \*/ |
|  | {MPL\_PROD\_KEY(2, 7), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-1) \*/ |
|  | {MPL\_PROD\_KEY(2, 8), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-2) \*/ |
|  | {MPL\_PROD\_KEY(2, 9), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-3) \*/ |
|  | {MPL\_PROD\_KEY(2, 10), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-4) \*/ |
|  | {MPL\_PROD\_KEY(2, 11), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-5) \*/ |
|  | {MPL\_PROD\_KEY(2, 12), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E1-6) \*/ |
|  | {MPL\_PROD\_KEY(2, 29), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/D4) \*/ |
|  | /\* prod\_ver = 3 \*/ |
|  | {MPL\_PROD\_KEY(3, 30), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B2/E2) \*/ |
|  | /\* prod\_ver = 4 \*/ |
|  | {MPL\_PROD\_KEY(4, 31), MPU\_SILICON\_REV\_B1, 131, 8192}, /\* (B2/F1) \*/ |
|  | {MPL\_PROD\_KEY(4, 1), MPU\_SILICON\_REV\_B1, 131, 8192}, /\* (B3/F1) \*/ |
|  | {MPL\_PROD\_KEY(4, 3), MPU\_SILICON\_REV\_B1, 131, 8192}, /\* (B4/F1) \*/ |
|  | /\* prod\_ver = 6 \*/ |
|  | {MPL\_PROD\_KEY(6, 19), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B5/E2) \*/ |
|  | /\* prod\_ver = 7 \*/ |
|  | {MPL\_PROD\_KEY(7, 19), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B5/E2) \*/ |
|  | /\* prod\_ver = 8 \*/ |
|  | {MPL\_PROD\_KEY(8, 19), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B5/E2) \*/ |
|  | /\* prod\_ver = 9\*/ |
|  | {MPL\_PROD\_KEY(9, 19), MPU\_SILICON\_REV\_B1, 131, 16384}, /\* (B5/E2) \*/ |
|  | /\* prod\_ver = 10 \*/ |
|  | {MPL\_PROD\_KEY(10, 19), MPU\_SILICON\_REV\_B1, 131, 16384} /\* (B5/E2) \*/ |
|  | }; |
|  |  |
|  | /\* |
|  | List of product software revisions |
|  |  |
|  | NOTE : |
|  | software revision 0 falls back to the old detection method |
|  | based off the product version and product revision per the |
|  | table above |
|  | \*/ |
|  | static struct prod\_rev\_map\_t sw\_rev\_map[] = { |
|  | {0, 0, 0, 0}, |
|  | {1, MPU\_SILICON\_REV\_B1, 131, 8192}, /\* rev C \*/ |
|  | {2, MPU\_SILICON\_REV\_B1, 131, 16384} /\* rev D \*/ |
|  | }; |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @internal |
|  | \* @brief Inverse lookup of the index of an MPL product key . |
|  | \* @param key |
|  | \* the MPL product indentifier also referred to as 'key'. |
|  | \* @return the index position of the key in the array, -1 if not found. |
|  | \*/ |
|  | short index\_of\_key(unsigned short key) |
|  | { |
|  | int i; |
|  | for (i = 0; i < NUM\_OF\_PROD\_REVS; i++) |
|  | if (prod\_rev\_map[i].mpl\_product\_key == key) |
|  | return (short)i; |
|  | return NOTFOUND\_PROD\_REVS; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @internal |
|  | \* @brief Get the product revision and version for MPU6050 and |
|  | \* extract all per-part specific information. |
|  | \* The product version number is read from the PRODUCT\_ID register in |
|  | \* user space register map. |
|  | \* The product revision number is in read from OTP bank 0, ADDR6[7:2]. |
|  | \* These 2 numbers, combined, provide an unique key to be used to |
|  | \* retrieve some per-device information such as the silicon revision |
|  | \* and the gyro and accel sensitivity trim values. |
|  | \* |
|  | \* @param mldl\_cfg |
|  | \* a pointer to the mldl config data structure. |
|  | \* @param mlsl\_handle |
|  | \* an file handle to the serial communication device the |
|  | \* device is connected to. |
|  | \* |
|  | \* @return 0 on success, a non-zero error code otherwise. |
|  | \*/ |
|  | static int inv\_get\_silicon\_rev\_mpu6050( |
|  | struct mldl\_cfg \*mldl\_cfg, void \*mlsl\_handle) |
|  | { |
|  | unsigned char prod\_ver, prod\_rev; |
|  | struct prod\_rev\_map\_t \*p\_rev; |
|  | unsigned sw\_rev; |
|  | unsigned short key; |
|  | unsigned char bank = |
|  | (BIT\_PRFTCH\_EN | BIT\_CFG\_USER\_BANK | MPU\_MEM\_OTP\_BANK\_0); |
|  | unsigned short mem\_addr = ((bank << 8) | 0x06); |
|  | short index; |
|  | unsigned char regs[5]; |
|  | struct mpu\_chip\_info \*mpu\_chip\_info = mldl\_cfg->mpu\_chip\_info; |
|  | int result; |
|  |  |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PRODUCT\_ID, 1, &prod\_ver); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | prod\_ver &= 0xF; |
|  |  |
|  | result = inv\_serial\_read\_mem(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | mem\_addr, 1, &prod\_rev); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | prod\_rev >>= 2; |
|  |  |
|  | /\* clean the prefetch and cfg user bank bits \*/ |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_BANK\_SEL, 0); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* get the software-product version \*/ |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_XA\_OFFS\_L, 5, regs); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | sw\_rev = (regs[4] & 0x01) << 2 | /\* 0x0b, bit 0 \*/ |
|  | (regs[2] & 0x01) << 1 | /\* 0x09, bit 0 \*/ |
|  | (regs[0] & 0x01); /\* 0x07, bit 0 \*/ |
|  |  |
|  | /\* if 0, use the product key to determine the type of part \*/ |
|  | if (sw\_rev == 0) { |
|  | key = MPL\_PROD\_KEY(prod\_ver, prod\_rev); |
|  | if (key == 0) { |
|  | MPL\_LOGE("Product id read as 0 " |
|  | "indicates device is either " |
|  | "incompatible or an MPU3050\n"); |
|  | return INV\_ERROR\_INVALID\_MODULE; |
|  | } |
|  | index = index\_of\_key(key); |
|  | if (index == -1 || index >= NUM\_OF\_PROD\_REVS) { |
|  | MPL\_LOGE("Unsupported product key %d in MPL\n", key); |
|  | return INV\_ERROR\_INVALID\_MODULE; |
|  | } |
|  | /\* check MPL is compiled for this device \*/ |
|  | if (prod\_rev\_map[index].silicon\_rev != MPU\_SILICON\_REV\_B1) { |
|  | MPL\_LOGE("MPL compiled for MPU6050B1 support " |
|  | "but device is not MPU6050B1 (%d)\n", key); |
|  | return INV\_ERROR\_INVALID\_MODULE; |
|  | } |
|  | p\_rev = &prod\_rev\_map[index]; |
|  |  |
|  | /\* if valid, use the software product key \*/ |
|  | } else if (sw\_rev < ARRAY\_SIZE(sw\_rev\_map)) { |
|  | p\_rev = &sw\_rev\_map[sw\_rev]; |
|  |  |
|  | } else { |
|  | MPL\_LOGE("Software revision key is outside of known " |
|  | "range [0..%d] : %d\n", ARRAY\_SIZE(sw\_rev\_map), sw\_rev); |
|  | return INV\_ERROR\_INVALID\_MODULE; |
|  | } |
|  |  |
|  | mpu\_chip\_info->product\_id = prod\_ver; |
|  | mpu\_chip\_info->product\_revision = prod\_rev; |
|  | mpu\_chip\_info->silicon\_revision = p\_rev->silicon\_rev; |
|  | mpu\_chip\_info->gyro\_sens\_trim = p\_rev->gyro\_trim; |
|  | mpu\_chip\_info->accel\_sens\_trim = p\_rev->accel\_trim; |
|  |  |
|  | return result; |
|  | } |
|  | #define inv\_get\_silicon\_rev inv\_get\_silicon\_rev\_mpu6050 |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @brief Enable / Disable the use MPU's secondary I2C interface level |
|  | \* shifters. |
|  | \* When enabled the secondary I2C interface to which the external |
|  | \* device is connected runs at VDD voltage (main supply). |
|  | \* When disabled the 2nd interface runs at VDDIO voltage. |
|  | \* See the device specification for more details. |
|  | \* |
|  | \* @note using this API may produce unpredictable results, depending on how |
|  | \* the MPU and slave device are setup on the target platform. |
|  | \* Use of this API should entirely be restricted to system |
|  | \* integrators. Once the correct value is found, there should be no |
|  | \* need to change the level shifter at runtime. |
|  | \* |
|  | \* @pre Must be called after inv\_serial\_start(). |
|  | \* @note Typically called before inv\_dmp\_open(). |
|  | \* |
|  | \* @param[in] enable: |
|  | \* 0 to run at VDDIO (default), |
|  | \* 1 to run at VDD. |
|  | \* |
|  | \* @return INV\_SUCCESS if successfull, a non-zero error code otherwise. |
|  | \*/ |
|  | static int inv\_mpu\_set\_level\_shifter\_bit(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*mlsl\_handle, unsigned char enable) |
|  | { |
|  | int result; |
|  | unsigned char regval; |
|  |  |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_YG\_OFFS\_TC, 1, &regval); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | if (enable) |
|  | regval |= BIT\_I2C\_MST\_VDDIO; |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_YG\_OFFS\_TC, regval); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @internal |
|  | \* @brief MPU6050 B1 power management functions. |
|  | \* @param mldl\_cfg |
|  | \* a pointer to the internal mldl\_cfg data structure. |
|  | \* @param mlsl\_handle |
|  | \* a file handle to the serial device used to communicate |
|  | \* with the MPU6050 B1 device. |
|  | \* @param reset |
|  | \* 1 to reset hardware. |
|  | \* @param sensors |
|  | \* Bitfield of sensors to leave on |
|  | \* |
|  | \* @return 0 on success, a non-zero error code on error. |
|  | \*/ |
|  | static int mpu60xx\_pwr\_mgmt(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*mlsl\_handle, |
|  | unsigned int reset, unsigned long sensors) |
|  | { |
|  | unsigned char pwr\_mgmt[2]; |
|  | unsigned char pwr\_mgmt\_prev[2]; |
|  | int result; |
|  | int sleep = !(sensors & (INV\_THREE\_AXIS\_GYRO | INV\_THREE\_AXIS\_ACCEL |
|  | | INV\_DMP\_PROCESSOR)); |
|  |  |
|  | if (reset) { |
|  | MPL\_LOGI("Reset MPU6050 B1\n"); |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGMT\_1, BIT\_H\_RESET); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_GYRO\_IS\_BYPASSED; |
|  | msleep(100); |
|  | } |
|  |  |
|  | /\* NOTE : reading both PWR\_MGMT\_1 and PWR\_MGMT\_2 for efficiency because |
|  | they are accessible even when the device is powered off \*/ |
|  | result = inv\_serial\_read(mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGMT\_1, 2, pwr\_mgmt\_prev); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | pwr\_mgmt[0] = pwr\_mgmt\_prev[0]; |
|  | pwr\_mgmt[1] = pwr\_mgmt\_prev[1]; |
|  |  |
|  | if (sleep) { |
|  | mldl\_cfg->inv\_mpu\_state->status |= MPU\_DEVICE\_IS\_SUSPENDED; |
|  | pwr\_mgmt[0] |= BIT\_SLEEP; |
|  | } else { |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_DEVICE\_IS\_SUSPENDED; |
|  | pwr\_mgmt[0] &= ~BIT\_SLEEP; |
|  | } |
|  | if (pwr\_mgmt[0] != pwr\_mgmt\_prev[0]) { |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGMT\_1, pwr\_mgmt[0]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | msleep(CHARGEPUMP\_WAKE); |
|  |  |
|  | pwr\_mgmt[1] &= ~(BIT\_STBY\_XG | BIT\_STBY\_YG | BIT\_STBY\_ZG); |
|  | if (!(sensors & INV\_X\_GYRO)) |
|  | pwr\_mgmt[1] |= BIT\_STBY\_XG; |
|  | if (!(sensors & INV\_Y\_GYRO)) |
|  | pwr\_mgmt[1] |= BIT\_STBY\_YG; |
|  | if (!(sensors & INV\_Z\_GYRO)) |
|  | pwr\_mgmt[1] |= BIT\_STBY\_ZG; |
|  |  |
|  | if (pwr\_mgmt[1] != pwr\_mgmt\_prev[1]) { |
|  | result = inv\_serial\_single\_write( |
|  | mlsl\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGMT\_2, pwr\_mgmt[1]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | if ((pwr\_mgmt[1] & (BIT\_STBY\_XG | BIT\_STBY\_YG | BIT\_STBY\_ZG)) == |
|  | (BIT\_STBY\_XG | BIT\_STBY\_YG | BIT\_STBY\_ZG)) { |
|  | mldl\_cfg->inv\_mpu\_state->status |= MPU\_GYRO\_IS\_SUSPENDED; |
|  | } else { |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_GYRO\_IS\_SUSPENDED; |
|  | } |
|  |  |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @brief sets the clock source for the gyros. |
|  | \* @param mldl\_cfg |
|  | \* a pointer to the struct mldl\_cfg data structure. |
|  | \* @param gyro\_handle |
|  | \* an handle to the serial device the gyro is assigned to. |
|  | \* @return ML\_SUCCESS if successful, a non-zero error code otherwise. |
|  | \*/ |
|  | static int mpu\_set\_clock\_source(void \*gyro\_handle, struct mldl\_cfg \*mldl\_cfg) |
|  | { |
|  | int result; |
|  | unsigned char cur\_clk\_src; |
|  | unsigned char reg; |
|  |  |
|  | /\* clock source selection \*/ |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGM, 1, &reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | cur\_clk\_src = reg & BITS\_CLKSEL; |
|  | reg &= ~BITS\_CLKSEL; |
|  |  |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_PWR\_MGM, mldl\_cfg->mpu\_gyro\_cfg->clk\_src | reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* ERRATA: |
|  | workaroud to switch from any MPU\_CLK\_SEL\_PLLGYROx to |
|  | MPU\_CLK\_SEL\_INTERNAL and XGyro is powered up: |
|  | 1) Select INT\_OSC |
|  | 2) PD XGyro |
|  | 3) PU XGyro |
|  | \*/ |
|  | if ((cur\_clk\_src == MPU\_CLK\_SEL\_PLLGYROX |
|  | || cur\_clk\_src == MPU\_CLK\_SEL\_PLLGYROY |
|  | || cur\_clk\_src == MPU\_CLK\_SEL\_PLLGYROZ) |
|  | && mldl\_cfg->mpu\_gyro\_cfg->clk\_src == MPU\_CLK\_SEL\_INTERNAL |
|  | && mldl\_cfg->inv\_mpu\_cfg->requested\_sensors & INV\_X\_GYRO) { |
|  | unsigned char first\_result = INV\_SUCCESS; |
|  | mldl\_cfg->inv\_mpu\_cfg->requested\_sensors &= ~INV\_X\_GYRO; |
|  | result = mpu60xx\_pwr\_mgmt( |
|  | mldl\_cfg, gyro\_handle, |
|  | false, mldl\_cfg->inv\_mpu\_cfg->requested\_sensors); |
|  | ERROR\_CHECK\_FIRST(first\_result, result); |
|  | mldl\_cfg->inv\_mpu\_cfg->requested\_sensors |= INV\_X\_GYRO; |
|  | result = mpu60xx\_pwr\_mgmt( |
|  | mldl\_cfg, gyro\_handle, |
|  | false, mldl\_cfg->inv\_mpu\_cfg->requested\_sensors); |
|  | ERROR\_CHECK\_FIRST(first\_result, result); |
|  | result = first\_result; |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Configures the MPU I2C Master |
|  | \* |
|  | \* @mldl\_cfg Handle to the configuration data |
|  | \* @gyro\_handle handle to the gyro communictation interface |
|  | \* @slave Can be Null if turning off the slave |
|  | \* @slave\_pdata Can be null if turning off the slave |
|  | \* @slave\_id enum ext\_slave\_type to determine which index to use |
|  | \* |
|  | \* |
|  | \* This fucntion configures the slaves by: |
|  | \* 1) Setting up the read |
|  | \* a) Read Register |
|  | \* b) Read Length |
|  | \* 2) Set up the data trigger (MPU6050 only) |
|  | \* a) Set trigger write register |
|  | \* b) Set Trigger write value |
|  | \* 3) Set up the divider (MPU6050 only) |
|  | \* 4) Set the slave bypass mode depending on slave |
|  | \* |
|  | \* returns INV\_SUCCESS or non-zero error code |
|  | \*/ |
|  |  |
|  | static int mpu\_set\_slave\_mpu60xx(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | struct ext\_slave\_descr \*slave, |
|  | struct ext\_slave\_platform\_data \*slave\_pdata, |
|  | int slave\_id) |
|  | { |
|  | int result; |
|  | unsigned char reg; |
|  | /\* Slave values \*/ |
|  | unsigned char slave\_reg; |
|  | unsigned char slave\_len; |
|  | unsigned char slave\_endian; |
|  | unsigned char slave\_address; |
|  | /\* Which MPU6050 registers to use \*/ |
|  | unsigned char addr\_reg; |
|  | unsigned char reg\_reg; |
|  | unsigned char ctrl\_reg; |
|  | /\* Which MPU6050 registers to use for the trigger \*/ |
|  | unsigned char addr\_trig\_reg; |
|  | unsigned char reg\_trig\_reg; |
|  | unsigned char ctrl\_trig\_reg; |
|  |  |
|  | unsigned char bits\_slave\_delay = 0; |
|  | /\* Divide down rate for the Slave, from the mpu rate \*/ |
|  | unsigned char d0\_trig\_reg; |
|  | unsigned char delay\_ctrl\_orig; |
|  | unsigned char delay\_ctrl; |
|  | long divider; |
|  |  |
|  | if (NULL == slave || NULL == slave\_pdata) { |
|  | slave\_reg = 0; |
|  | slave\_len = 0; |
|  | slave\_endian = 0; |
|  | slave\_address = 0; |
|  | } else { |
|  | slave\_reg = slave->read\_reg; |
|  | slave\_len = slave->read\_len; |
|  | slave\_endian = slave->endian; |
|  | slave\_address = slave\_pdata->address; |
|  | slave\_address |= BIT\_I2C\_READ; |
|  | } |
|  |  |
|  | switch (slave\_id) { |
|  | case EXT\_SLAVE\_TYPE\_ACCEL: |
|  | addr\_reg = MPUREG\_I2C\_SLV1\_ADDR; |
|  | reg\_reg = MPUREG\_I2C\_SLV1\_REG; |
|  | ctrl\_reg = MPUREG\_I2C\_SLV1\_CTRL; |
|  | addr\_trig\_reg = 0; |
|  | reg\_trig\_reg = 0; |
|  | ctrl\_trig\_reg = 0; |
|  | bits\_slave\_delay = BIT\_SLV1\_DLY\_EN; |
|  | break; |
|  | case EXT\_SLAVE\_TYPE\_COMPASS: |
|  | addr\_reg = MPUREG\_I2C\_SLV0\_ADDR; |
|  | reg\_reg = MPUREG\_I2C\_SLV0\_REG; |
|  | ctrl\_reg = MPUREG\_I2C\_SLV0\_CTRL; |
|  | addr\_trig\_reg = MPUREG\_I2C\_SLV2\_ADDR; |
|  | reg\_trig\_reg = MPUREG\_I2C\_SLV2\_REG; |
|  | ctrl\_trig\_reg = MPUREG\_I2C\_SLV2\_CTRL; |
|  | d0\_trig\_reg = MPUREG\_I2C\_SLV2\_DO; |
|  | bits\_slave\_delay = BIT\_SLV2\_DLY\_EN | BIT\_SLV0\_DLY\_EN; |
|  | break; |
|  | case EXT\_SLAVE\_TYPE\_PRESSURE: |
|  | addr\_reg = MPUREG\_I2C\_SLV3\_ADDR; |
|  | reg\_reg = MPUREG\_I2C\_SLV3\_REG; |
|  | ctrl\_reg = MPUREG\_I2C\_SLV3\_CTRL; |
|  | addr\_trig\_reg = MPUREG\_I2C\_SLV4\_ADDR; |
|  | reg\_trig\_reg = MPUREG\_I2C\_SLV4\_REG; |
|  | ctrl\_trig\_reg = MPUREG\_I2C\_SLV4\_CTRL; |
|  | bits\_slave\_delay = BIT\_SLV4\_DLY\_EN | BIT\_SLV3\_DLY\_EN; |
|  | break; |
|  | default: |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_PARAMETER); |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  | }; |
|  |  |
|  | /\* return if this slave has already been set \*/ |
|  | if ((slave\_address && |
|  | ((mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled & bits\_slave\_delay) |
|  | == bits\_slave\_delay)) || |
|  | (!slave\_address && |
|  | (mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled & bits\_slave\_delay) == |
|  | 0)) |
|  | return 0; |
|  |  |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, true); |
|  |  |
|  | /\* Address \*/ |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | addr\_reg, slave\_address); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | /\* Register \*/ |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | reg\_reg, slave\_reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Length, byte swapping, grouping & enable \*/ |
|  | if (slave\_len > BITS\_SLV\_LENG) { |
|  | MPL\_LOGW("Limiting slave burst read length to " |
|  | "the allowed maximum (15B, req. %d)\n", slave\_len); |
|  | slave\_len = BITS\_SLV\_LENG; |
|  | return INV\_ERROR\_INVALID\_CONFIGURATION; |
|  | } |
|  | reg = slave\_len; |
|  | if (slave\_endian == EXT\_SLAVE\_LITTLE\_ENDIAN) { |
|  | reg |= BIT\_SLV\_BYTE\_SW; |
|  | if (slave\_reg & 1) |
|  | reg |= BIT\_SLV\_GRP; |
|  | } |
|  | if (slave\_address) |
|  | reg |= BIT\_SLV\_ENABLE; |
|  |  |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | ctrl\_reg, reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Trigger \*/ |
|  | if (addr\_trig\_reg) { |
|  | /\* If slave address is 0 this clears the trigger \*/ |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | addr\_trig\_reg, |
|  | slave\_address & ~BIT\_I2C\_READ); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | if (slave && slave->trigger && reg\_trig\_reg) { |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | reg\_trig\_reg, |
|  | slave->trigger->reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | ctrl\_trig\_reg, |
|  | BIT\_SLV\_ENABLE | 0x01); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | d0\_trig\_reg, |
|  | slave->trigger->value); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } else if (ctrl\_trig\_reg) { |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | ctrl\_trig\_reg, 0x00); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | /\* Data rate \*/ |
|  | if (slave) { |
|  | struct ext\_slave\_config config; |
|  | long data; |
|  | config.key = MPU\_SLAVE\_CONFIG\_ODR\_RESUME; |
|  | config.len = sizeof(long); |
|  | config.apply = false; |
|  | config.data = &data; |
|  | if (!(slave->get\_config)) |
|  | return INV\_ERROR\_INVALID\_CONFIGURATION; |
|  |  |
|  | result = slave->get\_config(NULL, slave, slave\_pdata, &config); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | MPL\_LOGI("Slave %d ODR: %ld Hz\n", slave\_id, data / 1000); |
|  | divider = ((1000 \* inv\_mpu\_get\_sampling\_rate\_hz( |
|  | mldl\_cfg->mpu\_gyro\_cfg)) |
|  | / data) - 1; |
|  | } else { |
|  | divider = 0; |
|  | } |
|  |  |
|  | result = inv\_serial\_read(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_I2C\_MST\_DELAY\_CTRL, |
|  | 1, &delay\_ctrl\_orig); |
|  | delay\_ctrl = delay\_ctrl\_orig; |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (divider > 0 && divider <= MASK\_I2C\_MST\_DLY) { |
|  | result = inv\_serial\_read(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_I2C\_SLV4\_CTRL, 1, &reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | if ((reg & MASK\_I2C\_MST\_DLY) && |
|  | ((long)(reg & MASK\_I2C\_MST\_DLY) != |
|  | (divider & MASK\_I2C\_MST\_DLY))) { |
|  | MPL\_LOGW("Changing slave divider: %ld to %ld\n", |
|  | (long)(reg & MASK\_I2C\_MST\_DLY), |
|  | (divider & MASK\_I2C\_MST\_DLY)); |
|  |  |
|  | } |
|  | reg |= (unsigned char)(divider & MASK\_I2C\_MST\_DLY); |
|  | result = inv\_serial\_single\_write(gyro\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_I2C\_SLV4\_CTRL, |
|  | reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | delay\_ctrl |= bits\_slave\_delay; |
|  | } else { |
|  | delay\_ctrl &= ~(bits\_slave\_delay); |
|  | } |
|  | if (delay\_ctrl != delay\_ctrl\_orig) { |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_I2C\_MST\_DELAY\_CTRL, |
|  | delay\_ctrl); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | if (slave\_address) |
|  | mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled |= |
|  | bits\_slave\_delay; |
|  | else |
|  | mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled &= |
|  | ~bits\_slave\_delay; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | static int mpu\_set\_slave(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | struct ext\_slave\_descr \*slave, |
|  | struct ext\_slave\_platform\_data \*slave\_pdata, |
|  | int slave\_id) |
|  | { |
|  | return mpu\_set\_slave\_mpu60xx(mldl\_cfg, gyro\_handle, slave, |
|  | slave\_pdata, slave\_id); |
|  | } |
|  | /\*\* |
|  | \* Check to see if the gyro was reset by testing a couple of registers known |
|  | \* to change on reset. |
|  | \* |
|  | \* @mldl\_cfg mldl configuration structure |
|  | \* @gyro\_handle handle used to communicate with the gyro |
|  | \* |
|  | \* @return INV\_SUCCESS or non-zero error code |
|  | \*/ |
|  | static int mpu\_was\_reset(struct mldl\_cfg \*mldl\_cfg, void \*gyro\_handle) |
|  | { |
|  | int result = INV\_SUCCESS; |
|  | unsigned char reg; |
|  |  |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_DMP\_CFG\_2, 1, &reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg2 != reg) |
|  | return true; |
|  |  |
|  | if (0 != mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg1) |
|  | return false; |
|  |  |
|  | /\* Inconclusive assume it was reset \*/ |
|  | return true; |
|  | } |
|  |  |
|  |  |
|  | int inv\_mpu\_set\_firmware(struct mldl\_cfg \*mldl\_cfg, void \*mlsl\_handle, |
|  | const unsigned char \*data, int size) |
|  | { |
|  | int bank, offset, write\_size; |
|  | int result; |
|  | unsigned char read[MPU\_MEM\_BANK\_SIZE]; |
|  |  |
|  | if (mldl\_cfg->inv\_mpu\_state->status & MPU\_DEVICE\_IS\_SUSPENDED) { |
|  | #if INV\_CACHE\_DMP == 1 |
|  | memcpy(mldl\_cfg->mpu\_ram->ram, data, size); |
|  | return INV\_SUCCESS; |
|  | #else |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_MEMORY\_SET); |
|  | return INV\_ERROR\_MEMORY\_SET; |
|  | #endif |
|  | } |
|  |  |
|  | if (!(mldl\_cfg->inv\_mpu\_state->status & MPU\_DMP\_IS\_SUSPENDED)) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_MEMORY\_SET); |
|  | return INV\_ERROR\_MEMORY\_SET; |
|  | } |
|  | /\* Write and verify memory \*/ |
|  | for (bank = 0; size > 0; bank++, |
|  | size -= write\_size, |
|  | data += write\_size) { |
|  | if (size > MPU\_MEM\_BANK\_SIZE) |
|  | write\_size = MPU\_MEM\_BANK\_SIZE; |
|  | else |
|  | write\_size = size; |
|  |  |
|  | result = inv\_serial\_write\_mem(mlsl\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | ((bank << 8) | 0x00), |
|  | write\_size, |
|  | data); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | MPL\_LOGE("Write mem error in bank %d\n", bank); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_read\_mem(mlsl\_handle, |
|  | mldl\_cfg->mpu\_chip\_info->addr, |
|  | ((bank << 8) | 0x00), |
|  | write\_size, |
|  | read); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | MPL\_LOGE("Read mem error in bank %d\n", bank); |
|  | return result; |
|  | } |
|  |  |
|  | #define ML\_SKIP\_CHECK 38 |
|  | for (offset = 0; offset < write\_size; offset++) { |
|  | /\* skip the register memory locations \*/ |
|  | if (bank == 0 && offset < ML\_SKIP\_CHECK) |
|  | continue; |
|  | if (data[offset] != read[offset]) { |
|  | result = INV\_ERROR\_SERIAL\_WRITE; |
|  | break; |
|  | } |
|  | } |
|  | if (result != INV\_SUCCESS) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | MPL\_LOGE("Read data mismatch at bank %d, offset %d\n", |
|  | bank, offset); |
|  | return result; |
|  | } |
|  | } |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  | static int gyro\_resume(struct mldl\_cfg \*mldl\_cfg, void \*gyro\_handle, |
|  | unsigned long sensors) |
|  | { |
|  | int result; |
|  | int ii; |
|  | unsigned char reg; |
|  | unsigned char regs[7]; |
|  |  |
|  | /\* Wake up the part \*/ |
|  | result = mpu60xx\_pwr\_mgmt(mldl\_cfg, gyro\_handle, false, sensors); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Always set the INT\_ENABLE and DIVIDER as the Accel Only mode for 6050 |
|  | can set these too \*/ |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_INT\_ENABLE, (mldl\_cfg->mpu\_gyro\_cfg->int\_config)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_SMPLRT\_DIV, mldl\_cfg->mpu\_gyro\_cfg->divider); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (!(mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_NEEDS\_CONFIG) && |
|  | !mpu\_was\_reset(mldl\_cfg, gyro\_handle)) { |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  | /\* Configure the MPU \*/ |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = mpu\_set\_clock\_source(gyro\_handle, mldl\_cfg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | reg = MPUREG\_GYRO\_CONFIG\_VALUE(0, 0, 0, |
|  | mldl\_cfg->mpu\_gyro\_cfg->full\_scale); |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_GYRO\_CONFIG, reg); |
|  | reg = MPUREG\_CONFIG\_VALUE(mldl\_cfg->mpu\_gyro\_cfg->ext\_sync, |
|  | mldl\_cfg->mpu\_gyro\_cfg->lpf); |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_CONFIG, reg); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_DMP\_CFG\_1, mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_DMP\_CFG\_2, mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg2); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Write and verify memory \*/ |
|  | #if INV\_CACHE\_DMP != 0 |
|  | inv\_mpu\_set\_firmware(mldl\_cfg, gyro\_handle, |
|  | mldl\_cfg->mpu\_ram->ram, mldl\_cfg->mpu\_ram->length); |
|  | #endif |
|  |  |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_XG\_OFFS\_TC, |
|  | ((mldl\_cfg->mpu\_offsets->tc[0] << 1) & BITS\_XG\_OFFS\_TC)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | regs[0] = ((mldl\_cfg->mpu\_offsets->tc[1] << 1) & BITS\_YG\_OFFS\_TC); |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_YG\_OFFS\_TC, regs[0]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_single\_write( |
|  | gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_ZG\_OFFS\_TC, |
|  | ((mldl\_cfg->mpu\_offsets->tc[2] << 1) & BITS\_ZG\_OFFS\_TC)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | regs[0] = MPUREG\_X\_OFFS\_USRH; |
|  | for (ii = 0; ii < ARRAY\_SIZE(mldl\_cfg->mpu\_offsets->gyro); ii++) { |
|  | regs[1 + ii \* 2] = |
|  | (unsigned char)(mldl\_cfg->mpu\_offsets->gyro[ii] >> 8) |
|  | & 0xff; |
|  | regs[1 + ii \* 2 + 1] = |
|  | (unsigned char)(mldl\_cfg->mpu\_offsets->gyro[ii] & 0xff); |
|  | } |
|  | result = inv\_serial\_write(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | 7, regs); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Configure slaves \*/ |
|  | result = inv\_mpu\_set\_level\_shifter\_bit(mldl\_cfg, gyro\_handle, |
|  | mldl\_cfg->pdata->level\_shifter); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_GYRO\_NEEDS\_CONFIG; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | int gyro\_config(void \*mlsl\_handle, |
|  | struct mldl\_cfg \*mldl\_cfg, |
|  | struct ext\_slave\_config \*data) |
|  | { |
|  | struct mpu\_gyro\_cfg \*mpu\_gyro\_cfg = mldl\_cfg->mpu\_gyro\_cfg; |
|  | struct mpu\_chip\_info \*mpu\_chip\_info = mldl\_cfg->mpu\_chip\_info; |
|  | struct mpu\_offsets \*mpu\_offsets = mldl\_cfg->mpu\_offsets; |
|  | int ii; |
|  |  |
|  | if (!data->data) |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  |  |
|  | switch (data->key) { |
|  | case MPU\_SLAVE\_INT\_CONFIG: |
|  | mpu\_gyro\_cfg->int\_config = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_EXT\_SYNC: |
|  | mpu\_gyro\_cfg->ext\_sync = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_FULL\_SCALE: |
|  | mpu\_gyro\_cfg->full\_scale = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_LPF: |
|  | mpu\_gyro\_cfg->lpf = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_CLK\_SRC: |
|  | mpu\_gyro\_cfg->clk\_src = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_DIVIDER: |
|  | mpu\_gyro\_cfg->divider = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_ENABLE: |
|  | mpu\_gyro\_cfg->dmp\_enable = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_FIFO\_ENABLE: |
|  | mpu\_gyro\_cfg->fifo\_enable = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_CFG1: |
|  | mpu\_gyro\_cfg->dmp\_cfg1 = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_CFG2: |
|  | mpu\_gyro\_cfg->dmp\_cfg2 = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_TC: |
|  | for (ii = 0; ii < GYRO\_NUM\_AXES; ii++) |
|  | mpu\_offsets->tc[ii] = ((\_\_u8 \*)data->data)[ii]; |
|  | break; |
|  | case MPU\_SLAVE\_GYRO: |
|  | for (ii = 0; ii < GYRO\_NUM\_AXES; ii++) |
|  | mpu\_offsets->gyro[ii] = ((\_\_u16 \*)data->data)[ii]; |
|  | break; |
|  | case MPU\_SLAVE\_ADDR: |
|  | mpu\_chip\_info->addr = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_PRODUCT\_REVISION: |
|  | mpu\_chip\_info->product\_revision = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_SILICON\_REVISION: |
|  | mpu\_chip\_info->silicon\_revision = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_PRODUCT\_ID: |
|  | mpu\_chip\_info->product\_id = \*((\_\_u8 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_GYRO\_SENS\_TRIM: |
|  | mpu\_chip\_info->gyro\_sens\_trim = \*((\_\_u16 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_ACCEL\_SENS\_TRIM: |
|  | mpu\_chip\_info->accel\_sens\_trim = \*((\_\_u16 \*)data->data); |
|  | break; |
|  | case MPU\_SLAVE\_RAM: |
|  | if (data->len != mldl\_cfg->mpu\_ram->length) |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  |  |
|  | memcpy(mldl\_cfg->mpu\_ram->ram, data->data, data->len); |
|  | break; |
|  | default: |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_FEATURE\_NOT\_IMPLEMENTED); |
|  | return INV\_ERROR\_FEATURE\_NOT\_IMPLEMENTED; |
|  | }; |
|  | mldl\_cfg->inv\_mpu\_state->status |= MPU\_GYRO\_NEEDS\_CONFIG; |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  | int gyro\_get\_config(void \*mlsl\_handle, |
|  | struct mldl\_cfg \*mldl\_cfg, |
|  | struct ext\_slave\_config \*data) |
|  | { |
|  | struct mpu\_gyro\_cfg \*mpu\_gyro\_cfg = mldl\_cfg->mpu\_gyro\_cfg; |
|  | struct mpu\_chip\_info \*mpu\_chip\_info = mldl\_cfg->mpu\_chip\_info; |
|  | struct mpu\_offsets \*mpu\_offsets = mldl\_cfg->mpu\_offsets; |
|  | int ii; |
|  |  |
|  | if (!data->data) |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  |  |
|  | switch (data->key) { |
|  | case MPU\_SLAVE\_INT\_CONFIG: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->int\_config; |
|  | break; |
|  | case MPU\_SLAVE\_EXT\_SYNC: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->ext\_sync; |
|  | break; |
|  | case MPU\_SLAVE\_FULL\_SCALE: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->full\_scale; |
|  | break; |
|  | case MPU\_SLAVE\_LPF: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->lpf; |
|  | break; |
|  | case MPU\_SLAVE\_CLK\_SRC: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->clk\_src; |
|  | break; |
|  | case MPU\_SLAVE\_DIVIDER: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->divider; |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_ENABLE: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->dmp\_enable; |
|  | break; |
|  | case MPU\_SLAVE\_FIFO\_ENABLE: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->fifo\_enable; |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_CFG1: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->dmp\_cfg1; |
|  | break; |
|  | case MPU\_SLAVE\_DMP\_CFG2: |
|  | \*((\_\_u8 \*)data->data) = mpu\_gyro\_cfg->dmp\_cfg2; |
|  | break; |
|  | case MPU\_SLAVE\_TC: |
|  | for (ii = 0; ii < GYRO\_NUM\_AXES; ii++) |
|  | ((\_\_u8 \*)data->data)[ii] = mpu\_offsets->tc[ii]; |
|  | break; |
|  | case MPU\_SLAVE\_GYRO: |
|  | for (ii = 0; ii < GYRO\_NUM\_AXES; ii++) |
|  | ((\_\_u16 \*)data->data)[ii] = mpu\_offsets->gyro[ii]; |
|  | break; |
|  | case MPU\_SLAVE\_ADDR: |
|  | \*((\_\_u8 \*)data->data) = mpu\_chip\_info->addr; |
|  | break; |
|  | case MPU\_SLAVE\_PRODUCT\_REVISION: |
|  | \*((\_\_u8 \*)data->data) = mpu\_chip\_info->product\_revision; |
|  | break; |
|  | case MPU\_SLAVE\_SILICON\_REVISION: |
|  | \*((\_\_u8 \*)data->data) = mpu\_chip\_info->silicon\_revision; |
|  | break; |
|  | case MPU\_SLAVE\_PRODUCT\_ID: |
|  | \*((\_\_u8 \*)data->data) = mpu\_chip\_info->product\_id; |
|  | break; |
|  | case MPU\_SLAVE\_GYRO\_SENS\_TRIM: |
|  | \*((\_\_u16 \*)data->data) = mpu\_chip\_info->gyro\_sens\_trim; |
|  | break; |
|  | case MPU\_SLAVE\_ACCEL\_SENS\_TRIM: |
|  | \*((\_\_u16 \*)data->data) = mpu\_chip\_info->accel\_sens\_trim; |
|  | break; |
|  | case MPU\_SLAVE\_RAM: |
|  | if (data->len != mldl\_cfg->mpu\_ram->length) |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  |  |
|  | memcpy(data->data, mldl\_cfg->mpu\_ram->ram, data->len); |
|  | break; |
|  | default: |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_FEATURE\_NOT\_IMPLEMENTED); |
|  | return INV\_ERROR\_FEATURE\_NOT\_IMPLEMENTED; |
|  | }; |
|  |  |
|  | return INV\_SUCCESS; |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* Exported functions |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | /\*\* |
|  | \* Initializes the pdata structure to defaults. |
|  | \* |
|  | \* Opens the device to read silicon revision, product id and whoami. |
|  | \* |
|  | \* @mldl\_cfg |
|  | \* The internal device configuration data structure. |
|  | \* @mlsl\_handle |
|  | \* The serial communication handle. |
|  | \* |
|  | \* @return INV\_SUCCESS if silicon revision, product id and woami are supported |
|  | \* by this software. |
|  | \*/ |
|  | int inv\_mpu\_open(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*accel\_handle, |
|  | void \*compass\_handle, void \*pressure\_handle) |
|  | { |
|  | int result; |
|  | void \*slave\_handle[EXT\_SLAVE\_NUM\_TYPES]; |
|  | int ii; |
|  |  |
|  | /\* Default is Logic HIGH, pushpull, latch disabled, anyread to clear \*/ |
|  | ii = 0; |
|  | mldl\_cfg->inv\_mpu\_cfg->ignore\_system\_suspend = false; |
|  | mldl\_cfg->mpu\_gyro\_cfg->int\_config = BIT\_DMP\_INT\_EN; |
|  | mldl\_cfg->mpu\_gyro\_cfg->clk\_src = MPU\_CLK\_SEL\_PLLGYROZ; |
|  | mldl\_cfg->mpu\_gyro\_cfg->lpf = MPU\_FILTER\_42HZ; |
|  | mldl\_cfg->mpu\_gyro\_cfg->full\_scale = MPU\_FS\_2000DPS; |
|  | mldl\_cfg->mpu\_gyro\_cfg->divider = 4; |
|  | mldl\_cfg->mpu\_gyro\_cfg->dmp\_enable = 1; |
|  | mldl\_cfg->mpu\_gyro\_cfg->fifo\_enable = 1; |
|  | mldl\_cfg->mpu\_gyro\_cfg->ext\_sync = 0; |
|  | mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg1 = 0; |
|  | mldl\_cfg->mpu\_gyro\_cfg->dmp\_cfg2 = 0; |
|  | mldl\_cfg->inv\_mpu\_state->status = |
|  | MPU\_DMP\_IS\_SUSPENDED | |
|  | MPU\_GYRO\_IS\_SUSPENDED | |
|  | MPU\_ACCEL\_IS\_SUSPENDED | |
|  | MPU\_COMPASS\_IS\_SUSPENDED | |
|  | MPU\_PRESSURE\_IS\_SUSPENDED | |
|  | MPU\_DEVICE\_IS\_SUSPENDED; |
|  | mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled = 0; |
|  |  |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_GYROSCOPE] = gyro\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_ACCEL] = accel\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_COMPASS] = compass\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_PRESSURE] = pressure\_handle; |
|  |  |
|  | if (mldl\_cfg->mpu\_chip\_info->addr == 0) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_PARAMETER); |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  | } |
|  |  |
|  | /\* |
|  | \* Reset, |
|  | \* Take the DMP out of sleep, and |
|  | \* read the product\_id, sillicon rev and whoami |
|  | \*/ |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~MPU\_GYRO\_IS\_BYPASSED; |
|  | result = mpu60xx\_pwr\_mgmt(mldl\_cfg, gyro\_handle, true, |
|  | INV\_THREE\_AXIS\_GYRO); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | result = inv\_get\_silicon\_rev(mldl\_cfg, gyro\_handle); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Get the factory temperature compensation offsets \*/ |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_XG\_OFFS\_TC, 1, |
|  | &mldl\_cfg->mpu\_offsets->tc[0]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_YG\_OFFS\_TC, 1, |
|  | &mldl\_cfg->mpu\_offsets->tc[1]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_serial\_read(gyro\_handle, mldl\_cfg->mpu\_chip\_info->addr, |
|  | MPUREG\_ZG\_OFFS\_TC, 1, |
|  | &mldl\_cfg->mpu\_offsets->tc[2]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Into bypass mode before sleeping and calling the slaves init \*/ |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, true); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = inv\_mpu\_set\_level\_shifter\_bit(mldl\_cfg, gyro\_handle, |
|  | mldl\_cfg->pdata->level\_shifter); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | for (ii = 0; ii < GYRO\_NUM\_AXES; ii++) { |
|  | mldl\_cfg->mpu\_offsets->tc[ii] = |
|  | (mldl\_cfg->mpu\_offsets->tc[ii] & BITS\_XG\_OFFS\_TC) >> 1; |
|  | } |
|  |  |
|  | #if INV\_CACHE\_DMP != 0 |
|  | result = mpu60xx\_pwr\_mgmt(mldl\_cfg, gyro\_handle, false, 0); |
|  | #endif |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  |  |
|  | return result; |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Close the mpu interface |
|  | \* |
|  | \* @mldl\_cfg pointer to the configuration structure |
|  | \* @mlsl\_handle pointer to the serial layer handle |
|  | \* |
|  | \* @return INV\_SUCCESS or non-zero error code |
|  | \*/ |
|  | int inv\_mpu\_close(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*accel\_handle, |
|  | void \*compass\_handle, |
|  | void \*pressure\_handle) |
|  | { |
|  | return 0; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @brief resume the MPU device and all the other sensor |
|  | \* devices from their low power state. |
|  | \* |
|  | \* @mldl\_cfg |
|  | \* pointer to the configuration structure |
|  | \* @gyro\_handle |
|  | \* the main file handle to the MPU device. |
|  | \* @accel\_handle |
|  | \* an handle to the accelerometer device, if sitting |
|  | \* onto a separate bus. Can match mlsl\_handle if |
|  | \* the accelerometer device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @compass\_handle |
|  | \* an handle to the compass device, if sitting |
|  | \* onto a separate bus. Can match mlsl\_handle if |
|  | \* the compass device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @pressure\_handle |
|  | \* an handle to the pressure sensor device, if sitting |
|  | \* onto a separate bus. Can match mlsl\_handle if |
|  | \* the pressure sensor device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @resume\_gyro |
|  | \* whether resuming the gyroscope device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @resume\_accel |
|  | \* whether resuming the accelerometer device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @resume\_compass |
|  | \* whether resuming the compass device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @resume\_pressure |
|  | \* whether resuming the pressure sensor device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @return INV\_SUCCESS or a non-zero error code. |
|  | \*/ |
|  | int inv\_mpu\_resume(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*accel\_handle, |
|  | void \*compass\_handle, |
|  | void \*pressure\_handle, |
|  | unsigned long sensors) |
|  | { |
|  | int result = INV\_SUCCESS; |
|  | int ii; |
|  | bool resume\_slave[EXT\_SLAVE\_NUM\_TYPES]; |
|  | bool resume\_dmp = sensors & INV\_DMP\_PROCESSOR; |
|  | void \*slave\_handle[EXT\_SLAVE\_NUM\_TYPES]; |
|  | resume\_slave[EXT\_SLAVE\_TYPE\_GYROSCOPE] = |
|  | (sensors & (INV\_X\_GYRO | INV\_Y\_GYRO | INV\_Z\_GYRO)); |
|  | resume\_slave[EXT\_SLAVE\_TYPE\_ACCEL] = |
|  | sensors & INV\_THREE\_AXIS\_ACCEL; |
|  | resume\_slave[EXT\_SLAVE\_TYPE\_COMPASS] = |
|  | sensors & INV\_THREE\_AXIS\_COMPASS; |
|  | resume\_slave[EXT\_SLAVE\_TYPE\_PRESSURE] = |
|  | sensors & INV\_THREE\_AXIS\_PRESSURE; |
|  |  |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_GYROSCOPE] = gyro\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_ACCEL] = accel\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_COMPASS] = compass\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_PRESSURE] = pressure\_handle; |
|  |  |
|  |  |
|  | mldl\_print\_cfg(mldl\_cfg); |
|  |  |
|  | /\* Skip the Gyro since slave[EXT\_SLAVE\_TYPE\_GYROSCOPE] is NULL \*/ |
|  | for (ii = EXT\_SLAVE\_TYPE\_ACCEL; ii < EXT\_SLAVE\_NUM\_TYPES; ii++) { |
|  | if (resume\_slave[ii] && |
|  | ((!mldl\_cfg->slave[ii]) || |
|  | (!mldl\_cfg->slave[ii]->resume))) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_PARAMETER); |
|  | return INV\_ERROR\_INVALID\_PARAMETER; |
|  | } |
|  | } |
|  |  |
|  | if ((resume\_slave[EXT\_SLAVE\_TYPE\_GYROSCOPE] || resume\_dmp) |
|  | && ((mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_IS\_SUSPENDED) || |
|  | (mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_NEEDS\_CONFIG))) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = dmp\_stop(mldl\_cfg, gyro\_handle); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = gyro\_resume(mldl\_cfg, gyro\_handle, sensors); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | for (ii = 0; ii < EXT\_SLAVE\_NUM\_TYPES; ii++) { |
|  | if (!mldl\_cfg->slave[ii] || |
|  | !mldl\_cfg->pdata\_slave[ii] || |
|  | !resume\_slave[ii] || |
|  | !(mldl\_cfg->inv\_mpu\_state->status & (1 << ii))) |
|  | continue; |
|  |  |
|  | if (EXT\_SLAVE\_BUS\_SECONDARY == |
|  | mldl\_cfg->pdata\_slave[ii]->bus) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, |
|  | true); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | result = mldl\_cfg->slave[ii]->resume(slave\_handle[ii], |
|  | mldl\_cfg->slave[ii], |
|  | mldl\_cfg->pdata\_slave[ii]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status &= ~(1 << ii); |
|  | } |
|  |  |
|  | for (ii = 0; ii < EXT\_SLAVE\_NUM\_TYPES; ii++) { |
|  | if (resume\_dmp && |
|  | !(mldl\_cfg->inv\_mpu\_state->status & (1 << ii)) && |
|  | mldl\_cfg->pdata\_slave[ii] && |
|  | EXT\_SLAVE\_BUS\_SECONDARY == mldl\_cfg->pdata\_slave[ii]->bus) { |
|  | result = mpu\_set\_slave(mldl\_cfg, |
|  | gyro\_handle, |
|  | mldl\_cfg->slave[ii], |
|  | mldl\_cfg->pdata\_slave[ii], |
|  | mldl\_cfg->slave[ii]->type); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | /\* Turn on the master i2c iterface if necessary \*/ |
|  | if (resume\_dmp) { |
|  | result = mpu\_set\_i2c\_bypass( |
|  | mldl\_cfg, gyro\_handle, |
|  | !(mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | /\* Now start \*/ |
|  | result = dmp\_start(mldl\_cfg, gyro\_handle); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | mldl\_cfg->inv\_mpu\_cfg->requested\_sensors = sensors; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @brief suspend the MPU device and all the other sensor |
|  | \* devices into their low power state. |
|  | \* @mldl\_cfg |
|  | \* a pointer to the struct mldl\_cfg internal data |
|  | \* structure. |
|  | \* @gyro\_handle |
|  | \* the main file handle to the MPU device. |
|  | \* @accel\_handle |
|  | \* an handle to the accelerometer device, if sitting |
|  | \* onto a separate bus. Can match gyro\_handle if |
|  | \* the accelerometer device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @compass\_handle |
|  | \* an handle to the compass device, if sitting |
|  | \* onto a separate bus. Can match gyro\_handle if |
|  | \* the compass device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @pressure\_handle |
|  | \* an handle to the pressure sensor device, if sitting |
|  | \* onto a separate bus. Can match gyro\_handle if |
|  | \* the pressure sensor device operates on the same |
|  | \* primary bus of MPU. |
|  | \* @accel |
|  | \* whether suspending the accelerometer device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @compass |
|  | \* whether suspending the compass device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @pressure |
|  | \* whether suspending the pressure sensor device is |
|  | \* actually needed (if the device supports low power |
|  | \* mode of some sort). |
|  | \* @return INV\_SUCCESS or a non-zero error code. |
|  | \*/ |
|  | int inv\_mpu\_suspend(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*accel\_handle, |
|  | void \*compass\_handle, |
|  | void \*pressure\_handle, |
|  | unsigned long sensors) |
|  | { |
|  | int result = INV\_SUCCESS; |
|  | int ii; |
|  | struct ext\_slave\_descr \*\*slave = mldl\_cfg->slave; |
|  | struct ext\_slave\_platform\_data \*\*pdata\_slave = mldl\_cfg->pdata\_slave; |
|  | bool suspend\_dmp = ((sensors & INV\_DMP\_PROCESSOR) == INV\_DMP\_PROCESSOR); |
|  | bool suspend\_slave[EXT\_SLAVE\_NUM\_TYPES]; |
|  | void \*slave\_handle[EXT\_SLAVE\_NUM\_TYPES]; |
|  |  |
|  | suspend\_slave[EXT\_SLAVE\_TYPE\_GYROSCOPE] = |
|  | ((sensors & (INV\_X\_GYRO | INV\_Y\_GYRO | INV\_Z\_GYRO)) |
|  | == (INV\_X\_GYRO | INV\_Y\_GYRO | INV\_Z\_GYRO)); |
|  | suspend\_slave[EXT\_SLAVE\_TYPE\_ACCEL] = |
|  | ((sensors & INV\_THREE\_AXIS\_ACCEL) == INV\_THREE\_AXIS\_ACCEL); |
|  | suspend\_slave[EXT\_SLAVE\_TYPE\_COMPASS] = |
|  | ((sensors & INV\_THREE\_AXIS\_COMPASS) == INV\_THREE\_AXIS\_COMPASS); |
|  | suspend\_slave[EXT\_SLAVE\_TYPE\_PRESSURE] = |
|  | ((sensors & INV\_THREE\_AXIS\_PRESSURE) == |
|  | INV\_THREE\_AXIS\_PRESSURE); |
|  |  |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_GYROSCOPE] = gyro\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_ACCEL] = accel\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_COMPASS] = compass\_handle; |
|  | slave\_handle[EXT\_SLAVE\_TYPE\_PRESSURE] = pressure\_handle; |
|  |  |
|  | if (suspend\_dmp) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | result = dmp\_stop(mldl\_cfg, gyro\_handle); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | /\* Gyro \*/ |
|  | if (suspend\_slave[EXT\_SLAVE\_TYPE\_GYROSCOPE] && |
|  | !(mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_IS\_SUSPENDED)) { |
|  | result = mpu60xx\_pwr\_mgmt(mldl\_cfg, gyro\_handle, false, |
|  | ((~sensors) & INV\_ALL\_SENSORS)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | for (ii = 0; ii < EXT\_SLAVE\_NUM\_TYPES; ii++) { |
|  | bool is\_suspended = mldl\_cfg->inv\_mpu\_state->status & (1 << ii); |
|  | if (!slave[ii] || !pdata\_slave[ii] || |
|  | is\_suspended || !suspend\_slave[ii]) |
|  | continue; |
|  |  |
|  | if (EXT\_SLAVE\_BUS\_SECONDARY == pdata\_slave[ii]->bus) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | result = slave[ii]->suspend(slave\_handle[ii], |
|  | slave[ii], |
|  | pdata\_slave[ii]); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | if (EXT\_SLAVE\_BUS\_SECONDARY == pdata\_slave[ii]->bus) { |
|  | result = mpu\_set\_slave(mldl\_cfg, gyro\_handle, |
|  | NULL, NULL, |
|  | slave[ii]->type); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | mldl\_cfg->inv\_mpu\_state->status |= (1 << ii); |
|  | } |
|  |  |
|  | /\* Re-enable the i2c master if there are configured slaves and DMP \*/ |
|  | if (!suspend\_dmp) { |
|  | result = mpu\_set\_i2c\_bypass( |
|  | mldl\_cfg, gyro\_handle, |
|  | !(mldl\_cfg->inv\_mpu\_state->i2c\_slaves\_enabled)); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | mldl\_cfg->inv\_mpu\_cfg->requested\_sensors = (~sensors) & INV\_ALL\_SENSORS; |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | int inv\_mpu\_slave\_read(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*slave\_handle, |
|  | struct ext\_slave\_descr \*slave, |
|  | struct ext\_slave\_platform\_data \*pdata, |
|  | unsigned char \*data) |
|  | { |
|  | int result; |
|  | int bypass\_result; |
|  | int remain\_bypassed = true; |
|  |  |
|  | if (NULL == slave || NULL == slave->read) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_CONFIGURATION); |
|  | return INV\_ERROR\_INVALID\_CONFIGURATION; |
|  | } |
|  |  |
|  | if ((EXT\_SLAVE\_BUS\_SECONDARY == pdata->bus) |
|  | && (!(mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_IS\_BYPASSED))) { |
|  | remain\_bypassed = false; |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | result = slave->read(slave\_handle, slave, pdata, data); |
|  |  |
|  | if (!remain\_bypassed) { |
|  | bypass\_result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 0); |
|  | if (bypass\_result) { |
|  | LOG\_RESULT\_LOCATION(bypass\_result); |
|  | return bypass\_result; |
|  | } |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | int inv\_mpu\_slave\_config(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*slave\_handle, |
|  | struct ext\_slave\_config \*data, |
|  | struct ext\_slave\_descr \*slave, |
|  | struct ext\_slave\_platform\_data \*pdata) |
|  | { |
|  | int result; |
|  | int remain\_bypassed = true; |
|  |  |
|  | if (NULL == slave || NULL == slave->config) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_CONFIGURATION); |
|  | return INV\_ERROR\_INVALID\_CONFIGURATION; |
|  | } |
|  |  |
|  | if (data->apply && (EXT\_SLAVE\_BUS\_SECONDARY == pdata->bus) |
|  | && (!(mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_IS\_BYPASSED))) { |
|  | remain\_bypassed = false; |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | result = slave->config(slave\_handle, slave, pdata, data); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (!remain\_bypassed) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 0); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | int inv\_mpu\_get\_slave\_config(struct mldl\_cfg \*mldl\_cfg, |
|  | void \*gyro\_handle, |
|  | void \*slave\_handle, |
|  | struct ext\_slave\_config \*data, |
|  | struct ext\_slave\_descr \*slave, |
|  | struct ext\_slave\_platform\_data \*pdata) |
|  | { |
|  | int result; |
|  | int remain\_bypassed = true; |
|  |  |
|  | if (NULL == slave || NULL == slave->get\_config) { |
|  | LOG\_RESULT\_LOCATION(INV\_ERROR\_INVALID\_CONFIGURATION); |
|  | return INV\_ERROR\_INVALID\_CONFIGURATION; |
|  | } |
|  |  |
|  | if (data->apply && (EXT\_SLAVE\_BUS\_SECONDARY == pdata->bus) |
|  | && (!(mldl\_cfg->inv\_mpu\_state->status & MPU\_GYRO\_IS\_BYPASSED))) { |
|  | remain\_bypassed = false; |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 1); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | result = slave->get\_config(slave\_handle, slave, pdata, data); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  |  |
|  | if (!remain\_bypassed) { |
|  | result = mpu\_set\_i2c\_bypass(mldl\_cfg, gyro\_handle, 0); |
|  | if (result) { |
|  | LOG\_RESULT\_LOCATION(result); |
|  | return result; |
|  | } |
|  | } |
|  | return result; |
|  | } |

Powered by [Gitiles](https://gerrit.googlesource.com/gitiles/)[txt](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c?format=TEXT)[json](https://android.googlesource.com/kernel/msm.git/+/eaf36994a3992b8f918c18e4f7411e8b2320a35f/drivers/misc/mpu6050/mldl_cfg.c?format=JSON)