Appendix: Leveraging a Hardware Agnostic Approach to Ease Embedded Systems Design

Author: Giacomo Paterniani, Field Applications Engineer

```
//------------------------------------------------------------//
// ADIS16500 registers
#define ADIS16500_PROD_ID                   0x4074

#define ADIS16500_REG_DIAG_STAT             0x02
#define ADIS16500_REG_X_GYRO_L              0x04
#define ADIS16500_REG_X_GYRO_OUT            0x06
#define ADIS16500_REG_Y_GYRO_L              0x08
#define ADIS16500_REG_Y_GYRO_OUT            0x0A
#define ADIS16500_REG_Z_GYRO_L              0x0C
#define ADIS16500_REG_Z_GYRO_OUT            0x0E
#define ADIS16500_REG_X_ACCEL_L             0x10
#define ADIS16500_REG_X_ACCEL_OUT           0x12
#define ADIS16500_REG_Y_ACCEL_L             0x14
#define ADIS16500_REG_Y_ACCEL_OUT           0x16
#define ADIS16500_REG_Z_ACCEL_L             0x18
#define ADIS16500_REG_Z_ACCEL_OUT           0x1A
#define ADIS16500_REG_TEMP_OUT              0x1C
#define ADIS16500_REG_TIME_STAMP            0x1E

// spi max speed in brust mode
#define ADIS16500_BURST_MAX_SPEED           1000000

// default f_odr
#define ADIS16500_DEFAULT_F_ODR             2000 // Sample per second --> [hz]

// masks
#define ADIS16500_MASK_SYNC_MODE            0x000C
#define ADIS16500_MASK_DR_POL               0x0001

// output sensitivities (acceleration and gyroscope)
#define ADIS16500_ACC_SENSITIVITY_32b       5351254.0f          // [LSB/(m/sec^2)]
#define ADIS16500_GYRO_SENSITIVITY_16b      10.0f               // [LSB/(°/sec)]
#define ADIS16500_GYRO_SENSITIVITY_32b      655360.0f           // [LSB/(°/sec)]
#define ADIS16500_TEMP_SCALE_FACT_16b       0.1f                // [°C/LSB]
#define ADIS16500_TS_SCALE_FACT_16b         49.02f              // [usec/LSB]
```

Figure 1: Macros displayed in the ADIS16500 header file (adis16500.h)

```
//---------------------------------------------------------------//
// IMU stuff --> ADIS16500 sensor
static ADIS16500_INIT _adis16500_init =
{
    .spi_tx_func    = &_spi_adis16500_tx_func,
    .spi_rx_func    = &_spi_adis16500_rx_func,
    .delay_usec_func = &_delay_usec,
    .dr_pin_pol     = ADIS16500_DR_PIN_POLARITY__active_low,
    .sync_mode      = ADIS16500_SYNC_MODE__int_sync,
    .dec_rate       = 0
};
//---------------------------------------------------------------//
static int _spi_adis16500_tx_func(uint16_t *p_val)
{
    spiSelect(&SPID1);
    spiSend(&SPID1, 1, p_val);
    spiUnselect(&SPID1);

    return 1;
}
//---------------------------------------------------------------//

//---------------------------------------------------------------//
static int _spi_adis16500_rx_func(uint16_t *p_val)
{
    spiSelect(&SPID1);
    spiReceive(&SPID1, 1, p_val);
    spiUnselect(&SPID1);

    return 1;
}
//---------------------------------------------------------------//

//---------------------------------------------------------------//
static void _delay_usec(float usec)
{
    chThdSleepMicroseconds(usec);
}
//---------------------------------------------------------------//
```

Figure 2: main application layer file example

```
// Public variables
//------------------------------------------------------------//
typedef int  (*ADIS16500_SPI_TX_FUNC) (uint16_t *p_val);
typedef int  (*ADIS16500_SPI_RX_FUNC) (uint16_t *p_val);
typedef void (*ADIS16500_DELAY_FUNC) (float delay);

typedef enum
{
    ADIS16500_RET_VAL__ERROR    = 0,
    ADIS16500_RET_VAL__OK       = 1,
} ADIS16500_RET_VAL;

typedef struct
{
    float x;
    float y;
    float z;
} ADIS16500_XL_OUT;

typedef struct
{
    float x;
    float y;
    float z;
} ADIS16500_GYRO_OUT;

typedef struct
{
    float t;
} ADIS16500_TEMP_OUT;

typedef struct
{
    float ts;
} ADIS16500_TS_OUT;

typedef enum
{
    ADIS16500_SYNC_MODE__int_sync = 0,
    ADIS16500_SYNC_MODE__direct_input_sync,
    ADIS16500_SYNC_MODE__scaled_sync,
} ADIS16500_SYNC_MODE;

typedef enum
{
    ADIS16500_DR_PIN_POLARITY__active_low = 0,
    ADIS16500_DR_PIN_POLARITY__active_high,
} ADIS16500_DR_PIN_POLARITY;

typedef struct
{
    bool datapath_overrun;
    bool flash_memory_update_failure;
    bool spi_comm_error;
    bool standby_mode;
    bool sensor_failure;
    bool memory_failure;
    bool clock_error;
    bool gyroscope1_failure;
    bool gyroscope2_failure;
    bool accelerometer_failure;
} ADIS16500_ERROR_FLAGS;

typedef struct
{
    ADIS16500_SPI_TX_FUNC       spi_tx_func;
    ADIS16500_SPI_RX_FUNC       spi_rx_func;
    ADIS16500_DELAY_FUNC        delay_usec_func;
    ADIS16500_DR_PIN_POLARITY   dr_pin_pol;
    ADIS16500_SYNC_MODE         sync_mode;
    uint16_t                    dec_rate;
} ADIS16500_INIT;
//------------------------------------------------------------//
```

Figure 3: ADIS16500 public variables

```
// Public functions --> adis16500_rd_acc, adis16500_rd_gyro, adis16500_rd_temp,
//→ → → → → → adis16500_get_ts_usec, adis16500_wr_acc_calib, adis16500_wr_gyro_calib
//→ → → → → → functions must be called after adis16500_init
//------------------------------------------------------------//
int adis16500_init(const ADIS16500_INIT *p_init);
int adis16500_rd_reg16(uint8_t ad, uint16_t *p_reg_val);
int adis16500_wr_reg16(uint8_t ad, uint16_t reg_val);
int adis16500_rd_acc(ADIS16500_XL_OUT *p_acc);
int adis16500_rd_gyro(ADIS16500_GYRO_OUT *p_gyro);
ADIS16500_ERROR_FLAGS adis16500_rd_error_flag(void);
int adis16500_rd_temp(ADIS16500_TEMP_OUT *p_temp);
int adis16500_get_ts_usec(ADIS16500_TS_OUT *p_ts);
uint16_t adis16500_rd_data_cntr(void);
int adis16500_wr_acc_calib(ADIS16500_XL_OUT *p_acc_calib);
int adis16500_wr_gyro_calib(ADIS16500_GYRO_OUT *p_gyro_calib);
//------------------------------------------------------------//
#endif /* __ADIS16500_H__ */
```

Figure 4: ADIS16500 public functions

```
typedef struct
{
→bool→ → → → → → → initialized;
→ADIS16500_SPI_TX_FUNC→ → spi_tx_func;
→ADIS16500_SPI_RX_FUNC→ → spi_rx_func;
→ADIS16500_DELAY_FUNC→ → delay_usec_func;
→ADIS16500_DR_PIN_POLARITY→dr_pin_pol;
→ADIS16500_SYNC_MODE→ → sync_mode;
→uint16_t→ → → → → → dec_rate;
} ADIS16500_PRIV;

static ADIS16500_PRIV _adis16500_priv;
//------------------------------------------------------------//


// Public functions implementations
//------------------------------------------------------------//
int adis16500_init(const ADIS16500_INIT *p_init)
{
→_adis16500_priv.initialized = false;
→_adis16500_priv.spi_tx_func = p_init->spi_tx_func;
→_adis16500_priv.spi_rx_func = p_init->spi_rx_func;
→_adis16500_priv.delay_usec_func = p_init->delay_usec_func;

→// Check Prod ID
→adis16500_rd_reg16(ADIS16500_REG_PROD_ID, &_reg_val);
→if(_reg_val != ADIS16500_PROD_ID)
→{
→→return ADIS16500_RET_VAL__ERROR;
→}
→_reg_val = 0;

→// set DR pin polarity
→_adis16500_priv.dr_pin_pol = p_init->dr_pin_pol;
→adis16500_wr_reg16(ADIS16500_REG_MSC_CTRL, _adis16500_priv.dr_pin_pol);

→// set sync mode
→_adis16500_priv.sync_mode = p_init->sync_mode & ADIS16500_MASK_SYNC_MODE;
→adis16500_wr_reg16(ADIS16500_REG_MSC_CTRL, _adis16500_priv.sync_mode);

→// set decimation rate
→_adis16500_priv.dec_rate = p_init->dec_rate;
→adis16500_wr_reg16(ADIS16500_REG_DEC_RATE, _adis16500_priv.dec_rate);

→_adis16500_priv.initialized = true;

→return ADIS16500_RET_VAL__OK;
}
//------------------------------------------------------------//
```

Figure 5: ADIS16500 initialization function implementation

```
//---------------------------------------------------------------//
int adis16500_rd_reg16(uint8_t ad, uint16_t *p_reg_val)
{
    uint16_t txbuf[1] = {(AD_RD(ad) << 8) & 0xFF00};

    // Transmit R/W bit, register address shifted by 8 bit
    _adis16500_priv.spi_tx_func(&txbuf[0]);

    // Delay
    _adis16500_priv.delay_usec_func(0.01);

    // Read the register value
    _adis16500_priv.spi_rx_func(p_reg_val);

    return ADIS16500_RET_VAL__OK;
}
//---------------------------------------------------------------//

//---------------------------------------------------------------//
int adis16500_wr_reg16(uint8_t ad, uint16_t reg_val)
{
    uint16_t reg_val_l = (reg_val) & (0x00FF);
    uint16_t reg_val_h = ((reg_val) & (0xFF00)) >> 8;

    uint16_t ad_l = ((uint16_t)AD_WR(ad) << 8) & 0xFF00;
    uint16_t ad_h = ((uint16_t)AD_WR(ad + 1) << 8) & 0xFF00;

    uint16_t txbuf1[1] = {ad_l | reg_val_l};
    uint16_t txbuf2[1] = {ad_h | reg_val_h};

    // transmit R/W bit, register address (low) shifted by 8 bit and low reg_val
    _adis16500_priv.spi_tx_func(&txbuf1[0]);

    // Delay
    _adis16500_priv.delay_usec_func(0.01);

    // transmit R/W bit, register address (high) shifted by 8 bit and high reg_val
    _adis16500_priv.spi_tx_func(&txbuf2[0]);

    return ADIS16500_RET_VAL__OK;
}
//---------------------------------------------------------------//
```

Figure 6: ADIS16500 read and write register functions implementation

```
//-------------------------------------------------------------//
int adis16500_rd_acc(ADIS16500_XL_OUT *p_acc)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    // read x-axis acc
    adis16500_rd_reg16(ADIS16500_REG_X_ACCEL_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_X_ACCEL_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp -= OFFSET_CONVERSION_VALUE_32b;
        p_acc->x = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    else
    {
        p_acc->x = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    _reg_val = 0;

    // read y-axis acc
    adis16500_rd_reg16(ADIS16500_REG_Y_ACCEL_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_Y_ACCEL_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp = _temp - OFFSET_CONVERSION_VALUE_32b;
        p_acc->y = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    else
    {
        p_acc->y = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    _reg_val = 0;

    // read z-axis acc
    adis16500_rd_reg16(ADIS16500_REG_Z_ACCEL_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_Z_ACCEL_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp = _temp - OFFSET_CONVERSION_VALUE_32b;
        p_acc->z = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    else
    {
        p_acc->z = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
    }
    _reg_val = 0;

    return ADIS16500_RET_VAL__OK;
}
//-------------------------------------------------------------//
```

Figure 7: ADIS16500 read acceleration function implementation

```
//-------------------------------------------------------------//
int adis16500_rd_gyro(ADIS16500_GYRO_OUT *p_gyro)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    // read x-axis gyro
    adis16500_rd_reg16(ADIS16500_REG_X_GYRO_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_X_GYRO_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp = _temp - OFFSET_CONVERSION_VALUE_32b;
        p_gyro->x = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    else
    {
        p_gyro->x = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    _reg_val = 0;

    // read y-axis gyro
    adis16500_rd_reg16(ADIS16500_REG_Y_GYRO_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_Y_GYRO_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp = _temp - OFFSET_CONVERSION_VALUE_32b;
        p_gyro->y = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    else
    {
        p_gyro->y = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    _reg_val = 0;

    // read z-axis gyro
    adis16500_rd_reg16(ADIS16500_REG_Z_GYRO_OUT, &_reg_val);
    _temp = ((uint32_t)_reg_val << 16) & MASK_16MSB;
    _reg_val = 0;
    adis16500_rd_reg16(ADIS16500_REG_Z_GYRO_L, &_reg_val);
    _temp += (uint32_t)_reg_val;
    if(_temp > TH_CONVERSION_VALUE_32b)
    {
        _temp = _temp - OFFSET_CONVERSION_VALUE_32b;
        p_gyro->z = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    else
    {
        p_gyro->z = (float)_temp / ADIS16500_GYRO_SENSITIVITY_32b;
    }
    _reg_val = 0;

    return ADIS16500_RET_VAL__OK;
}
//-------------------------------------------------------------//
```

Figure 8: ADIS16500 read gyroscope function implementation

```
//---------------------------------------------------------------//
int adis16500_rd_temp(ADIS16500_TEMP_OUT *p_temp)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    // read temp
    adis16500_rd_reg16(ADIS16500_REG_TEMP_OUT, &_reg_val);
    if(_reg_val > TH_CONVERSION_VALUE_16b)
    {
        _reg_val = _reg_val - OFFSET_CONVERSION_VALUE_16b;
        p_temp->t = (float)_reg_val * ADIS16500_TEMP_SCALE_FACT_16b;
    }
    else
    {
        p_temp->t = (float)_reg_val * ADIS16500_TEMP_SCALE_FACT_16b;
    }
    _reg_val = 0;

    return ADIS16500_RET_VAL__OK;
}
//---------------------------------------------------------------//

//---------------------------------------------------------------//
int adis16500_get_ts_usec(ADIS16500_TS_OUT *p_ts)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    // read timestamp
    adis16500_rd_reg16(ADIS16500_REG_TIME_STAMP, &_reg_val);
    if(_reg_val > TH_CONVERSION_VALUE_16b)
    {
        _reg_val = _reg_val - OFFSET_CONVERSION_VALUE_16b;
        p_ts->ts = (float)_reg_val * ADIS16500_TS_SCALE_FACT_16b;
    }
    else
    {
        p_ts->ts = (float)_reg_val * ADIS16500_TS_SCALE_FACT_16b;
    }
    _reg_val = 0;

    return ADIS16500_RET_VAL__OK;
}
//---------------------------------------------------------------//

//---------------------------------------------------------------//
uint16_t adis16500_rd_data_cntr(void)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    // read data counter
    adis16500_rd_reg16(ADIS16500_REG_DATA_CNTR, &_reg_val);

    return _reg_val;
}
//---------------------------------------------------------------//
```

Figure 9: ADIS16500 read temperature, timestamp, data counter functions implementation

```
//-----------------------------------------------------------//
ADIS16500_ERROR_FLAGS adis16500_rd_error_flag(void)
{
    ADIS16500_ERROR_FLAGS adis16500_error_flags;

    adis16500_error_flags.datapath_overrun          = false;
    adis16500_error_flags.flash_memory_update_failure = false;
    adis16500_error_flags.spi_comm_error            = false;
    adis16500_error_flags.standby_mode              = false;
    adis16500_error_flags.sensor_failure            = false;
    adis16500_error_flags.memory_failure            = false;
    adis16500_error_flags.clock_error               = false;
    adis16500_error_flags.gyroscope1_failure        = false;
    adis16500_error_flags.gyroscope2_failure        = false;
    adis16500_error_flags.accelerometer_failure     = false;

    adis16500_rd_reg16(ADIS16500_REG_DIAG_STAT, &_reg_val);

    if(_reg_val & ADIS16500_REG_DIAG_STAT_datapath_overrun)
    {
        adis16500_error_flags.datapath_overrun = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_flash_memory_update_failure)
    {
        adis16500_error_flags.flash_memory_update_failure = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_spi_comm_error)
    {
        adis16500_error_flags.spi_comm_error = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_standby_mode)
    {
        adis16500_error_flags.standby_mode = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_sensor_failure)
    {
        adis16500_error_flags.sensor_failure = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_memory_failure)
    {
        adis16500_error_flags.memory_failure = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_clock_error)
    {
        adis16500_error_flags.clock_error = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_gyroscope1_failure)
    {
        adis16500_error_flags.gyroscope1_failure = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_gyroscope2_failure)
    {
        adis16500_error_flags.gyroscope2_failure = true;
    }
    if(_reg_val & ADIS16500_REG_DIAG_STAT_accelerometer_failure)
    {
        adis16500_error_flags.accelerometer_failure = true;
    }

    return adis16500_error_flags;
}
//-----------------------------------------------------------//
```

Figure 10: ADIS16500 read error flags function implementation

```
//-----------------------------------------------------------//
int adis16500_wr_acc_calib(ADIS16500_XL_OUT *p_acc_calib)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    uint16_t calib_l, calib_h;

    // write x-axis acc calibration value
    if(p_acc_calib->x >= 0)
    {
        _temp = (int32_t)(p_acc_calib->x * ADIS16500_ACC_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_acc_calib->x * ADIS16500_ACC_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_X_ACCEL_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_X_ACCEL_BIAS_H, calib_h);

    // write y-axis acc calibration value
    if(p_acc_calib->y >= 0)
    {
        _temp = (int32_t)(p_acc_calib->y * ADIS16500_ACC_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_acc_calib->y * ADIS16500_ACC_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_Y_ACCEL_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_Y_ACCEL_BIAS_H, calib_h);

    // write z-axis acc calibration value
    if(p_acc_calib->z >= 0)
    {
        _temp = (int32_t)(p_acc_calib->z * ADIS16500_ACC_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_acc_calib->z * ADIS16500_ACC_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_Z_ACCEL_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_Z_ACCEL_BIAS_H, calib_h);

    return ADIS16500_RET_VAL__OK;
}
//-----------------------------------------------------------//
```

Figure 11: ADIS16500 write acceleration calibration function implementation

```
//-------------------------------------------------------------//
int adis16500_wr_gyro_calib(ADIS16500_GYRO_OUT *p_gyro_calib)
{
    if(!_adis16500_priv.initialized)
    {
        return ADIS16500_RET_VAL__ERROR;
    }

    uint16_t calib_l, calib_h;

    // write x-axis gyro calibration value
    if(p_gyro_calib->x >= 0)
    {
        _temp = (int32_t)(p_gyro_calib->x * ADIS16500_GYRO_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_gyro_calib->x * ADIS16500_GYRO_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_X_GYRO_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_X_GYRO_BIAS_H, calib_h);

    // write y-axis acc calibration value
    if(p_gyro_calib->y >= 0)
    {
        _temp = (int32_t)(p_gyro_calib->y * ADIS16500_GYRO_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_gyro_calib->y * ADIS16500_GYRO_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_Y_GYRO_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_Y_GYRO_BIAS_H, calib_h);

    // write z-axis acc calibration value
    if(p_gyro_calib->z >= 0)
    {
        _temp = (int32_t)(p_gyro_calib->z * ADIS16500_GYRO_SENSITIVITY_32b);
    }
    else
    {
        _temp = (int32_t)(p_gyro_calib->z * ADIS16500_GYRO_SENSITIVITY_32b);
        _temp += OFFSET_CONVERSION_VALUE_32b;
    }
    calib_l = (uint16_t)(_temp & MASK_16LSB);
    calib_h = (uint16_t)((_temp & MASK_16MSB) >> 16);
    adis16500_wr_reg16(ADIS16500_REG_Z_GYRO_BIAS_L, calib_l);
    adis16500_wr_reg16(ADIS16500_REG_Z_GYRO_BIAS_H, calib_h);

    return ADIS16500_RET_VAL__OK;
}
//-------------------------------------------------------------//
```

Figure 12: ADIS16500 write gyroscope calibration function implementation