

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

Author: Giacomo Paterniani, Field Applications Engineer

```
-----//
// ADIS16500 registers
#define ADIS16500 PROD ID \longrightarrow \longrightarrow 0x4074
#define ADIS16500 REG_DIAG_STAT \longrightarrow \longrightarrow 0x02
#define \cdot ADIS16500 REG Y_GYRO L \rightarrow \longrightarrow 0x08
#define ADIS16500 REG Z GYRO L \rightarrow \longrightarrow 0 \times 0 \times 0 \times 0
#define ADIS16500 REG X ACCEL OUT \longrightarrow 0x12
#define ADIS16500 REG Y ACCEL L \rightarrow \rightarrow 0x14
#define ADIS16500 REG Y ACCEL OUT \longrightarrow Ox16
 \begin{array}{c} \texttt{#define ADIS16500 REG Z ACCEL L} \longrightarrow \longrightarrow 0 \texttt{x18} \\ \texttt{#define ADIS16500 REG Z ACCEL OUT} \longrightarrow \longrightarrow 0 \texttt{x1A} \\ \end{array} 
#define ADIS16500 REG_TEMP_OUT \rightarrow \longrightarrow 0x1C
// spi max speed in brust mode
// default f_odr
#define ADIS16500 DEFAULT F ODR> ----> 2000 // Sample per second ---> [hz]
// masks
#define ADIS16500 MASK DR POL\longrightarrow\longrightarrow0x0001
// output sensitivities (acceleration and gyroscope)
```

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



```
//-----
                         -----//
 // ·IMU · stuff ·--> · ADIS16500 · sensor
 static ADIS16500_INIT _adis16500_init =
 \begin{array}{l} & \longrightarrow . \text{spi_tx_func} \longrightarrow = \pounds \text{spi_adis16500_tx_func,} \\ & \longrightarrow . \text{spi_rx_func} \longrightarrow = \pounds \text{spi_adis16500_rx_func,} \\ & \longrightarrow . \text{delay_usec_func} = \pounds \text{delay_usec,} \end{array} 
 .dr_pin_pol> → = ADIS16500_DR_PIN_POLARITY_active_low,
→.sync_mode→ → = ADIS16500_SYNC_MODE_int_sync,
  \longrightarrow.dec_rate\longrightarrow
L};
 //-----//
 static int _ spi_adis16500_tx_func (uint16_t * p_val)
⊟{
   spiSelect (&SPID1);
  spiSend(&SPID1, 1, p_val);
  \longrightarrowreturn\cdot 1;
 3
 .
//-----//
 //-----//
 static int _spi_adis16500_rx_func(uint16_t *p_val)
spiSelect(&SPID1);
spiReceive(&SPID1, 1, p_val);
  ——>spiUnselect(&SPID1);
  \longrightarrowreturn\cdot 1;
 -}
 .
//-----//
 //-----//
 static void __delay_usec(float usec)
- (
  ——>chThdSleepMicroseconds(usec);
L}
 //---
                       -----//
```

Figure 2: main application layer file example



```
// Public variables
11 .-
                                                                                               -----//
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
1
      ADIS16500 RET_VAL_ERROR == 0,
ADIS16500 RET_VAL_OK == 1,
} ADIS16500 RET VAL;
typedef struct
ł
     >float x;
    \rightarrow float y;
\rightarrow float z;
} ADIS16500_XL_OUT;
typedef struct
     -float x;
    \rightarrow float y;
\rightarrow float z;
} ADIS16500_GYRO_OUT;
typedef struct
ł
     float t;
ADIS16500_TEMP_OUT;
typedef struct
1
     float ts;
} ADIS16500 TS_OUT;
typedef enum
ł
     ADIS16500_SYNC_MODE__int_sync = 0,
ADIS16500_SYNC_MODE__direct_input_sync,
ADIS16500_SYNC_MODE__scaled_sync,
} ADIS16500 SYNC MODE;
typedef enum
ł
     ADIS16500_DR_PIN_POLARITY__active_low = 0,
ADIS16500_DR_PIN_POLARITY__active_high,
} ADIS16500_DR_PIN_POLARITY;
typedef struct
ł
      -bool datapath_overrun;
    →bool flash_memory_update_failure;
 bool spi_comm_error;
bool standby_mode;
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
H.
      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;
                                                   >dec rate;
     Juint16 t
} ADIS16500 INIT;
                                                                                  -----//
```

Figure 3: ADIS16500 public variables



Figure 4: ADIS16500 public functions



Figure 5: ADIS16500 initialization function implementation



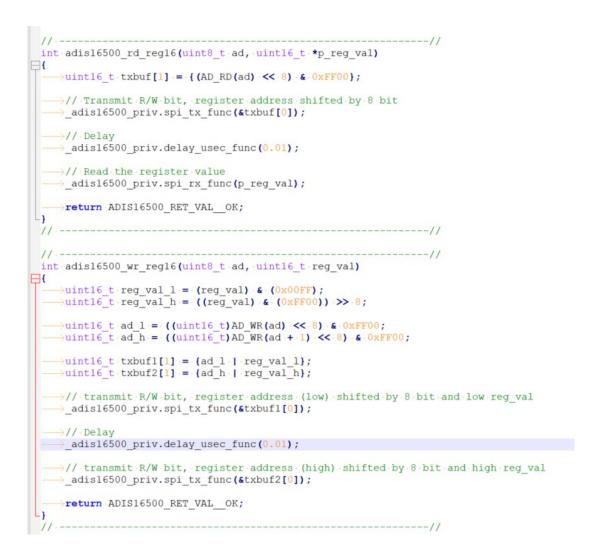


Figure 6: ADIS16500 read and write register functions implementation



```
--11
 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;</pre>
       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;
      1
      else
     •{
          p_acc->x = (float)_temp / ADIS16500_ACC_SENSITIVITY_32b;
     -1
     >_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;
    \rightarrow_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;</pre>
       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



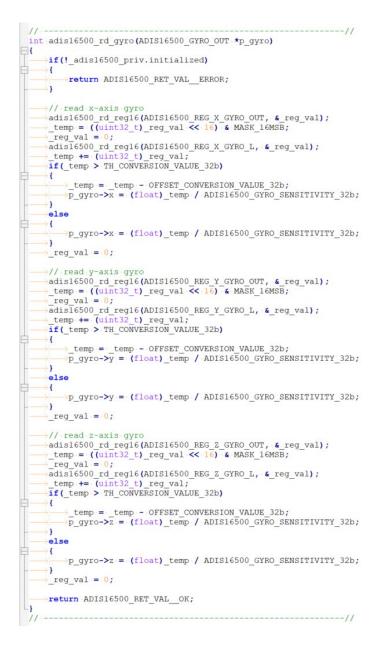


Figure 8: ADIS16500 read gyroscope function implementation



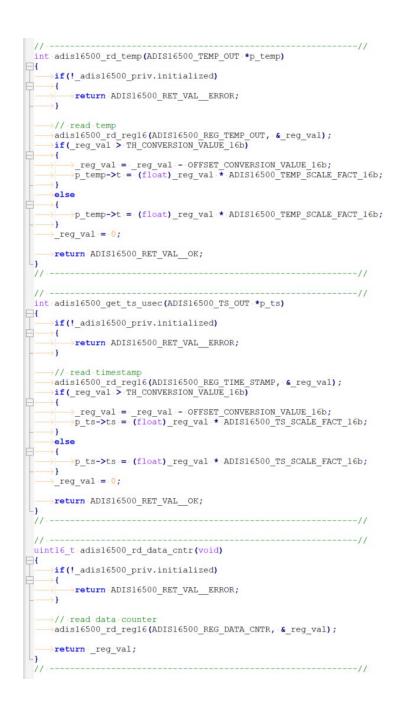


Figure 9: ADIS16500 read temperature, timestamp, data counter functions implementation



<pre>ADIS16500_ERROR_FLAGS adis16500_error_flags;</pre>
⇒adis16500 error flags.datapath overrun→→→→= false;
<pre>>adis16500_error_flags.flash_memory_update_failure</pre>
adis16500 error flags.spi comm error \rightarrow = false;
adis16500_error_flags.standby_mode >>> = false;
adis16500_error_flags.sensor_failure
adis16500 error flags.memory failure
→adis16500 error flags.clock error → → → →= false;
adis16500 error flags.gyroscopel failure
<pre>>adis16500_error_flags.gyroscope2_failure</pre>
adis16500_error_flags.accelerometer_failure>>= false;
adis16500_rd_reg16(ADIS16500_REG_DIAG_STAT, &_reg_val);
<pre>if(_reg_val & ADIS16500_REG_DIAG_STAT_datapath_overrun)</pre>
<pre>>{</pre>
<pre>}} +if(_reg_val ⋅ & ADIS16500_REG_DIAG_STAT_flash_memory_update_fai</pre>
<pre> >{ >adis16500_error_flags.flash_memory_update_failure = true; </pre>
<pre>}} if(_reg_val ⋅ & ADIS16500_REG_DIAG_STAT_spi_comm_error)</pre>
<pre>>{ adis16500_error_flags.spi_comm_error = true;</pre>
<pre>}} if(_reg_val & ADIS16500_REG_DIAG_STAT_standby_mode)</pre>
<pre>>{</pre>
→} →if(reg val & ADIS16500 REG DIAG STAT sensor failure)
<pre>→{ →adis16500_error_flags.sensor_failure = true;</pre>
<pre> >) >if(reg val & ADIS16500 REG DIAG STAT memory failure) </pre>
<pre>>{</pre>
<pre> →} →if(reg val & ADIS16500 REG DIAG STAT clock error) </pre>
÷{
<pre>>>adis16500_error_flags.clock_error = true; >}</pre>
<pre>>if(_reg_val & ADIS16500_REG_DIAG_STAT_gyroscope1_failure) >{</pre>
<pre>>adis16500_error_flags.gyroscope1_failure = true; >}</pre>
<pre>→if(_reg_val & ADIS16500_REG_DIAG_STAT_gyroscope2_failure) →{</pre>
<pre>adis16500_error_flags.gyroscope2_failure = true; }</pre>
<pre>>if(_reg_val & ADIS16500_REG_DIAG_STAT_accelerometer_failure) >{</pre>
<pre>adis16500_error_flags.accelerometer_failure = true; }</pre>
→return adis16500 error_flags;

Figure 10: ADIS16500 read error flags function implementation



```
11.
  int adis16500_wr_acc_calib(ADIS16500_XL_OUT *p_acc_calib)
if(!_adis16500_priv.initialized)
     \rightarrow{
           return ADIS16500 RET VAL ERROR;
     \rightarrow}
     -)uint16 t calib 1, calib h;
      >// write x-axis acc calibration value
   temp = (int32 t) (p acc calib->x * ADIS16500 ACC SENSITIVITY 32b);
    \rightarrow
     else
   \rightarrow{
             temp = (int32 t) (p acc calib->x * ADIS16500 ACC SENSITIVITY 32b);
          -_temp += OFFSET_CONVERSION_VALUE 32b;
      -1
     >calib_l = (uint16_t) (_temp & MASK_16LSB);
>calib_h = (uint16_t) ((_temp & MASK_16MSB) >> 16);
>adis16500_wr_reg16(ADI$16500_REG_X_ACCEL_BIAS_L, calib_1);
      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);
      - 1
     else
     -1
           temp = (int32 t) (p acc calib->y * ADIS16500 ACC SENSITIVITY 32b);
           += OFFSET_CONVERSION VALUE 32b;
     >calib_1 = (uint16_t) (_temp & MASK_16LSB);
>calib_h = (uint16_t) ((_temp & MASK_16MSB) >> 16);
>adis16500_wr_reg16(ADIS16500_REG_Y_ACCEL_BIAS_L, calib_1);
>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);
     \rightarrow}
      else
    \rightarrow
          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;
 1
11 .-
                                                    -----//
```

Figure 11: ADIS16500 write acceleration calibration function implementation



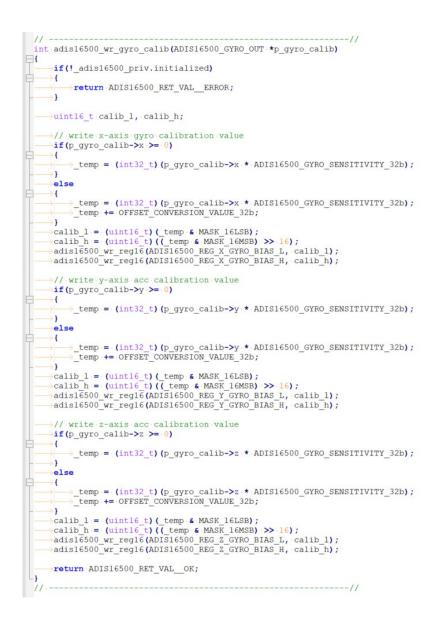


Figure 12: ADIS16500 write gyroscope calibration function implementation