EMS

V1

Generated by Doxygen 1.8.3.1

Mon Feb 24 2014 15:21:43

Contents

Chapter 1

Main Page

PROGRAM NAME:

Energy Meter Driver

PURPOSE:

The Program is made to configure and read Registers of an Energy Meter IC ADE7880, for the purpose of obtaining instantaneous values of Voltage, Current, Power and watt hour values read at specific measurement point.

***VERSIONS*:**

***Version* history:**

Version

Version Number 1

Date

xx.xx.2014/

Author

TEAM HW

Added fun :

Additional functinality is add to the programm in order to read and write to or from a commen file used between the Python script and this module

Chapter 2

IMPORTANT MACROS

- *CALIBRATION* CONSTANTS: This constants calculated so that readings fetched from target registers can be converted into meaningful values such as amps , volts, kwh, watt.

  The constants should be adjusted if there is any change to Hardware design

  The calibration steps can be found from this calibration document

  http://www.analog.com/static/imported-files/application_notes/AN-1171.-pdf .

  Note

  Since all phases on the meter are matched, the same constant can be used for all energy, power,current rms and voltage rms readings on any Rpi

```
VLSB_CONST              211184.71337579617834394904458599       LSB constant for voltage reading 24 volt
ILSB_CONST              353022.88732394                         LSB constant for current reading 30 Amp
WHLSB_CONST             (67.28)*2.275                           LSB constant for energy reading
WATTLSB_CONST           8505                                    LSB constant for power reading
```

- *TRANSFORMER_RATIO*

  This value is the ratio of the voltage transformer used to step down the line voltage to lower level. the value should be adjusted if different transformer is used.

```
TRANSFORMER_RATIO         20.56621005
```

- *CONFIGURATION* COMMANDS This macros determine the common configuration file between the phyton script and this software. And the configuration command structure.

  You only need to adjust this values, to obtain the corresponding changes on the program execution.

```
CONFIG_FILE_NAME        "config.csv"            defines configuration
CONFIG_CMD_FORMAT       "%hd %s %lu %hhu %hhu"  configuration command format specifier
```

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 USER_INTERFACE

- int8_t wait_new_conversion (void)
- int16_t main (int argc, const char **argv)
- void reading_loop (void)
- int8_t save_to_file (uint16_t rpi_address, measured_data_t data, const char *fileName, uint8_t clean)
- int16_t config_recheck (const char *config_file, const char *format,...)

6.1.1 Detailed Description

6.1.1.1 USER INTERFACE

Top most module in the application, responsible for handling commands received

from the shell or via config.csv file.

In order that, the program runs accordingly.:

```
[executable] + command
```

The followoing requests are supported:

```
driver_ade7880  config
```

-> Runs Default configuration procedure. As described in the device datasheet section QUICK SETUP AS ENERGY METER

(Outputs resulting message to consol on success or failure.)

```
driver_ade7880  read  [register phisical address (HEX eg. 0xFFFF)] [register size in byte (decimal)]
```

-> Reads Specific register value

(Outputs resulting message to consol on success or failure)

```
driver_ade7880  write [register phisical address (HEX eg. 0xFFFF)] [value to be stored (HEX eg. 0xFFFF)] [regi
```

-> Writes the given value to Specific register

(Outputs resulting message to consol on success or failure)

```
driver_ade7880  run
```

-> Runs the normal program execution routine

(Outputs resulting message to consol on success or failure.)

There are also common commands between Python script and this module used to alter the normal

execution routine via common configuration csv format interface file . i.e. the program continuously

Monitor this file so as to act accordingly if there is any change.

The structure of the command used is simple

```
id(int);fileName;Sampling_Rate(uint32_t);loop_control(bool);pause(bool)
   |       |            |                        |                     |
   |       |            |                        |                     |___ Control flag:
   |       |            |                        |                          if set to 1 pause reading
   |       |            |                        |                          and set to 0 resume reading
   |       |            |                        |_____________________ Control flag:
   |       |            |                                                 if set to 1, the program will
   |       |            |                                                 exit reading loop and end execution
   |       |            |_____________________________________________ Delay_control:
   |       |                                                              Adds the specified number of
   |       |                                                              millsecond delay on the
   |       |                                                              main reading loop
   |       |_____________________________________________________________ Sets file name
   |                                                                      where data to be saved
   |_____________________________________________________________________ Sets device id
```

6.1.2 Function Documentation

6.1.2.1 int16_t config_recheck (const char * *config_file*, const char * *format*, ...)

Parameters

in	*config_file*	pointer to the file Name array
in	*format*	String format specifying configuration command structure anything used on printf function can also be used here
in,out	...	the function expects at least as many additional arguments as specified by format.

Function reads the specified file and fills into passed arguments.

Returns

function return 0 or -1, on success or failure respectively.

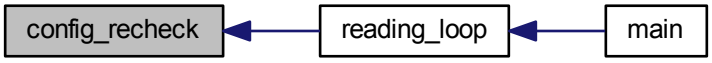
Definition at line 482 of file ade7880_driver.c.

Referenced by reading_loop().

```
00483 {
00484
00485    int16_t result;
```

```
00486    int16_t ii,cc;
00487    FILE *f =fopen(config_file,"r");
00488    if (f == NULL) return -1;
00489
00490   char cmd_line[100];
00491   va_list args;
00492   va_start (args, format);
00493
00494   if(fgets(cmd_line,100,f)!=NULL){
00495       for(ii=0,cc=0;*(cmd_line+ii)!='\0';ii++)
00496         if(*(cmd_line+ii)==';'){*(cmd_line+ii)= ' '; ++cc;}
00497
00498     result=vsscanf (cmd_line, format, args);
00499     result = (result == cc)?0:-1;
00500   }
00501
00502   va_end (args);
00503   fclose(f);
00504   return result;
00505 }
```

Here is the caller graph for this function:



6.1.2.2 int16_t main (int *argc*, const char ** *argv*)

Subroutine:main program subroutine

Parameters

in	*argc*	number of arguments sent from the shell
out	*argv*	pointer to the argument buffer

Function commonly perform library Initialization, Setup Rpi gpio and spi interface. Then run the requested procedure based on the argument passed from shall.

Returns

function return 0 or -1, on success or failure respectively.

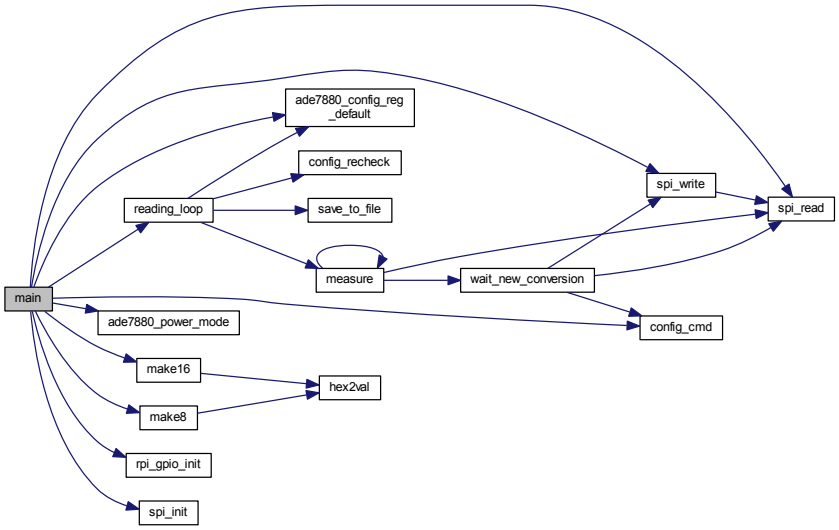
Definition at line 198 of file ade7880_driver.c.

References ade7880_config_reg_default(), ade7880_power_mode(), CHIP_ADDRESS1, config_cmd(), DISABLE, ENABLE, make16(), make8(), PIN_SS, PSM0, reading_loop(), rpi_gpio_init(), SET_RAM_WR_PROTECTION, spi_init(), spi_read(), and spi_write().

```
00198                                        {
00199
00200
00201
00202     int8_t cc =0;
00203     int16_t result =-1;
00204
00205     struct timeval tv;
00206
00207      if((argc < 1)|| !bcm2835_init()/*library has to be initialized*/)
00208           return -1;
00209
```

```
00210     rpi_gpio_init();
00211     ade7880_power_mode(PSM0);
00212     usleep(50);
00213
00214     if(spi_init(BCM2835_SPI_CS0)!= 0)
00215     return -1;
00216         #ifdef DEBUG
00217         uint16_t cp=0;
00218         printf("\n
      ######################################%d############################################################\n",cp++);
00219         #endif
00220
00221
00222      if(strcmp (argv[1],"config")==0)
00223      {
00224         if(ade7880_config_reg_default()!=-1)
00225               printf("\nDevice Ready to use\n");
00226      }
00227      else
00228      if(strcmp(argv[1],"read")==0)
00229      {
00230         if(argc<4){printf("\nCMD ERROR: RD\n");return-1;}
00231
00232         for(cc = 0;cc<((argc>=5)?atoi(argv[4]):1);cc++){
00233          spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,make16((uint8_t *)argv[2],2),
      atoi(argv[3]));
00234          gettimeofday(&tv,NULL);
00235          printf("<------------------------------------------------------------ %d timesamp: %lf\n",
00236          cc+1,(double)(tv.tv_sec + tv.tv_usec/(double)10E6));
00237         }
00238      return 0;
00239      }
00240      else
00241      if(strcmp(argv[1],"write")==0 )
00242      {
00243         if(argc<4){printf("\nCMD ERROR: WR\n");return-1;}
00244     int ii;
00245        for(ii=0;ii<argc;ii++)
00246             printf(" argval %d, = %s",ii,argv[ii]);
00247
00248         result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,DISABLE))== -1)&& (cc++ < 3));
00249         if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,make16((uint8_t *)argv[2],2)
00250         ,make8((uint8_t *)argv[3],2),atoi(argv[4]))!=0)
00251           result =-1;
00252          result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,ENABLE))== -1)&& (cc++ < 3));
00253          if(result == -1){
00254                     printf("ERROR: Couldn't write");
00255                       return -1;
00256          }
00257
00258      return 0;
00259      }
00260      else
00261      if(strcmp(argv[1],"run")==0)
00262      {
00263         #ifdef DEBUG
00264         if(argc >=3){
00265            if((strcmp(argv[2],"dprint=off")==0))
00266                spi_enable_msg_debug_print(DISABLE);
00267             else if((strcmp(argv[2],"dprint=on")==0))
00268                 spi_enable_msg_debug_print(ENABLE);
00269         }
00270         printf("\n
      ######################################%d############################################################\n",cp++);
00271         #endif
00272
00273         if(argc<2){printf("\nCMD ERROR: measure\n");return-1;}
00274
00275            reading_loop();
00276      }
00277
00278
00279      bcm2835_spi_end();
00280
00281      bcm2835_gpio_write(PIN_SS, HIGH);
00282
00283      bcm2835_close();
00284
00285 return 0;
00286 }
```

Here is the call graph for this function:



6.1.2.3 void reading_loop (void)

Function reads measured values of main interest form ADE7880, checks the current configuration command form config.csv, and saves the data to the specified interface file. The configuration command affect the behavior at which this subroutine execute, please use this document for more detail.

Returns

none.

Note

on startup new file will be used to save data,
otherwise data will be appended to the existing
interface file.
This is required due to the reason that, after power down
on reset the file could be corrupted.

Definition at line 302 of file ade7880_driver.c.

References ade7880_config_reg_default(), CONFIG_CMD_FORMAT, CONFIG_FILE_NAME, config_recheck(), ENABLE, phase_data_t::IRMS, measure(), PHASE_A, measured_data_t::phase_a, PHASE_ACTIVE_POWER, PHASE_ACTIVE_WH, PHASE_B, measured_data_t::phase_b, PHASE_C, measured_data_t::phase_c, PHASE_IRMS, PHASE_VRMS, phase_data_t::POWER, save_to_file(), phase_data_t::VRMS, and phase_data_t::WH.

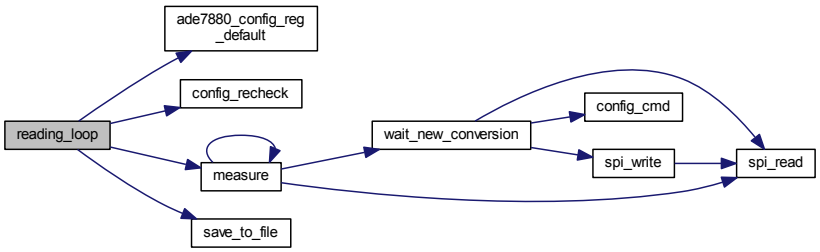
Referenced by main().

```
00302                       {
00303
00304   int16_t   rpi_address;
00305   char *    filename;
00306   uint32_t  cyc_time;
00307   uint8_t   loop_ctrl;
00308   uint8_t   pause;
```

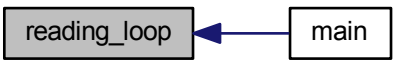
```
00309   int8_t    result=0;
00310   int8_t startup = 1;
00311
00312     measured_data_t data;
00313
00314         #ifdef DEBUG
00315         printf("\n\n");
00316         #endif
00317          printf("\nEntering Main loop ...\n");
00318          while(1){
00319          printf("\n");
00320
00321          printf("\nReading Phase A values ...\n");
00322          if((data.phase_a.IRMS           = measure(PHASE_IRMS,
      PHASE_A,100))==-1)result = -1;
00323          if((data.phase_a.VRMS           = measure(PHASE_VRMS,
      PHASE_A,100))==-1)result = -1;
00324          if((data.phase_a.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_A,1))==-1)result = -1;
00325          if((data.phase_a.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_A,1))==-1)result = -1;
00326
00327          printf("\nReading Phase B values ...\n");
00328          if((data.phase_b.IRMS           = measure(PHASE_IRMS,
      PHASE_B,100))==-1)result = -1;
00329          if((data.phase_b.VRMS           = measure(PHASE_VRMS,
      PHASE_B,100))==-1)result = -1;
00330          if((data.phase_b.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_B,1))==-1)result = -1;
00331          if((data.phase_b.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_B,1))==-1)result = -1;
00332
00333          printf("\nReading Phase C values ...\n");
00334          if((data.phase_c.IRMS           = measure(PHASE_IRMS,
      PHASE_C,100))==-1)result = -1;
00335          if((data.phase_c.VRMS           = measure(PHASE_VRMS,
      PHASE_C,100))==-1)result = -1;
00336          if((data.phase_c.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_C,1))==-1)result = -1;
00337          if((data.phase_c.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_C,1))==-1)result = -1;
00338
00339         printf("\n\n\nREADINGS:\n");
00340         printf("\n");
00341         printf("\n-------------------------------------------------------------------------PHASE A KWH  : %f\n"
      ,data.phase_a.WH);
00342         printf("\n-------------------------------------------------------------------------PHASE A POWER: %f\n"
      ,data.phase_a.POWER);
00343         printf("\n-------------------------------------------------------------------------PHASE A VRMS : %f\n"
      ,data.phase_a.VRMS);
00344         printf("\n-------------------------------------------------------------------------PHASE A IRMS : %f\n"
      ,data.phase_a.IRMS);
00345         printf("\n");
00346         printf("\n-------------------------------------------------------------------------PHASE B KWH  : %f\n"
      ,data.phase_b.WH);
00347         printf("\n-------------------------------------------------------------------------PHASE B POWER: %f\n"
      ,data.phase_b.POWER);
00348         printf("\n-------------------------------------------------------------------------PHASE B VRMS : %f\n"
      ,data.phase_b.VRMS);
00349         printf("\n-------------------------------------------------------------------------PHASE B IRMS : %f\n"
      ,data.phase_b.IRMS);
00350         printf("\n");
00351         printf("\n-------------------------------------------------------------------------PHASE C KWH  : %f\n"
      ,data.phase_c.WH);
00352         printf("\n-------------------------------------------------------------------------PHASE C POWER: %f\n"
      ,data.phase_c.POWER);
00353         printf("\n-------------------------------------------------------------------------PHASE C VRMS : %f\n"
      ,data.phase_c.VRMS);
00354         printf("\n-------------------------------------------------------------------------PHASE C IRMS : %f\n"
      ,data.phase_c.IRMS);
00355
00356         if(result == -1){
00357             ade7880_config_reg_default(); //this is the only thing we do for now
00358         //  continue;
00359         }
00360
00361         do{
00362         if(config_recheck (CONFIG_FILE_NAME,
      CONFIG_CMD_FORMAT,&rpi_address ,filename, &cyc_time,&loop_ctrl,&pause)==0){
00363
00364         if(pause==0){
00365
00366           while(save_to_file(rpi_address,data, filename,
00367           startup
00376           )!=0)
00377
00378           startup = 0;
```

```
00379
00380           }
00381
00382           }
00383
00384           usleep(cyc_time);
00385         }while(pause==1);
00386
00387         if(loop_ctrl == 1)
00388         break;
00389
00390
00391
00392
00393         #ifdef DEBUG
00394         if(result!=-1)
00395         printf("\nREADING SUCESS\n");
00396         spi_enable_msg_debug_print(ENABLE);
00397         #endif
00398
00399
00400       }
00401
00402 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.4 int8_t save_to_file (uint16_t *rpi_address*, measured_data_t *data*, const char * *fileName*, uint8_t *clean*)

Parameters

in	*rpi_address*	address of the Rpi
in	*data*	data structure with the data to be saved to the file
in	*fileName*	pointer to the fileName array
in	*clean*	boolean flag, if 0 data will be appended to the file, if 1 new file will be used

function saves data to the specified file in csv file format.

Returns

function return 0 or -1, on success or failure respectively.

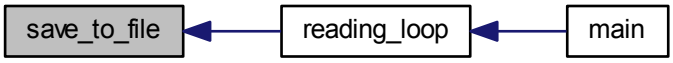
Definition at line 421 of file ade7880_driver.c.

References DUMMY_MSG, phase_data_t::IRMS, measured_data_t::phase_a, measured_data_t::phase_b, measured_data_t::phase_c, phase_data_t::POWER, phase_data_t::VRMS, and phase_data_t::WH.

Referenced by reading_loop().

```
00422 {
00423 uint32_t dummy=0;
00424   FILE *f = fopen(fileName, (clean == 1)?"w":"a");
00425   errno = 0;
00426   if (f == NULL){
00427    warn("%s: Couldn't open file %s; %s\n",fileName, strerror (errno));
00428
00429   return -1;
00430   }
00431
00432
00433   fprintf(f, "%d;%lu;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%d;%d;%d;%d;%d;%d\n",
00434           rpi_address,
00435           (unsigned)time(NULL),
00436
00437                data.phase_a.IRMS,
00438                data.phase_a.VRMS,
00439                data.phase_a.WH,
00440                data.phase_a.POWER,
00441
00442                data.phase_b.IRMS,
00443                data.phase_b.VRMS,
00444                data.phase_b.WH,
00445                data.phase_b.POWER,
00446
00447                data.phase_c.IRMS,
00448                data.phase_c.VRMS,
00449                data.phase_c.WH,
00450                data.phase_c.POWER,
00451                DUMMY_MSG,
00452                DUMMY_MSG,
00453                DUMMY_MSG,
00454                DUMMY_MSG,
00455                DUMMY_MSG,
00456                DUMMY_MSG
00457   );
00458
00459
00460   fclose(f);
00461   return 0;
00462 }
```

Here is the caller graph for this function:



6.1.2.5 int8_t wait_new_conversion (void)

Function clears STATUS0 register Data-Ready bit and waits until the DSP set

it back again. this verifies new reading is loaded to value registers.

Returns

function return 0 or -1 on success or failure respectively.

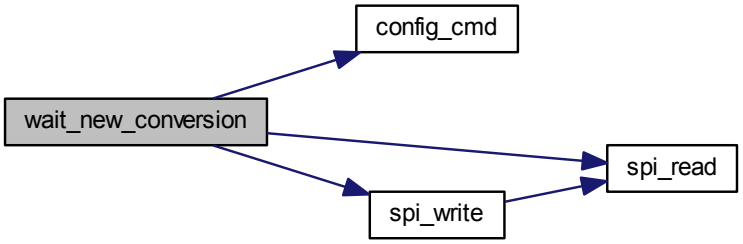
Definition at line 122 of file ade7880_driver.c.

References CHIP_ADDRESS1, config_cmd(), DISABLE, ENABLE, status0_reg_u::reg_all, SET_RAM_WR_PROTECTION, spi_read(), spi_write(), and STATUS0.

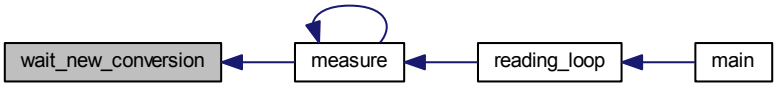
Referenced by measure().

```
00122                                {
00123
00124           int8_t cc =0;
00125           int16_t result =-1;
00126
00127           struct timeval tv;
00128           status0_reg_u status0;
00129           status0.reg_all = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,STATUS0,sizeof(uint32_t));
00130
00131
00132
00133           if(status0.bits.DREADY==1){
00134
00135           result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,DISABLE))== -1)&& (cc++ < 3));
00136           //put back the value to clears status flags
00137           if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,status0.reg_all,sizeof(uint32_t))!=0)result =-1;
00138           status0.reg_all = spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,sizeof(uint32_t));
00139           result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,ENABLE))== -1)&& (cc++ < 3));
00140
00141
00142           if(result == -1){
00143                   printf("\nERROR: Couldn't write\n");
00144                   return -1;
00145           }
00146
00147
00148           }
00149
00150
00151
00152
00153           gettimeofday(&tv,NULL);
00154           uint32_t t1,t2;
00155           t1=t2 = tv.tv_usec;
00156           #ifdef DEBUG
00157           printf("\nSTATUS REG VALUE %08X,---------------------------------------- us time %lu\n",
      status0.reg_all,t1);
00158           #endif
00159           while(status0.bits.DREADY==0){ //wait till conversion is done
00160
00161             status0.reg_all = spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,sizeof(uint32_t));
00162             gettimeofday(&tv,NULL);
00163
00164             t2 = tv.tv_usec;
00165             #ifdef DEBUG
00166             printf("\nSTATUS REG VALUE %08X,---------------------------------------- us time %lu\n",
      status0.reg_all,t2);
00167             #endif
00168             if((t2-t1)>20000)
00169             return -1;
00170
00171           }
00172
00173
00174
00175   return 0;
00176 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.2 DEVICE_CONFIGERATION

Functions

- int8_t config_cmd (uint32_t cmd, uint8_t arg,...)
- int8_t ade7880_power_mode (uint8_t mode)
- int8_t ade7880_config_reg_default ()
- int8_t spi_init (uint8_t chip_select)
- int8_t rpi_gpio_init (void)

6.2.1 Detailed Description

6.2.1.1 DEVICE CONFIGERATION

The module contains functions that are prepared and used to configure the

Raspberry pi GPIO and spi interface, As well as ADE7880 Power mode,

Serial Port selection (that is SPI port activation) and Default configuration

procedure as described in the device datasheet section QUICK SETUP AS ENERGY METER.

ADE7880 SPI PORT ACTIVATION

- TOGEL SS PIN 3 TIMES or execute three SPI write operations to a location in the address space that is not allocated to a specific register
- Write to CONFIG2 register to lock the port.

POWER MODE SELECTION

power_mode.PNG

The ADE7880 has four modes of operation, determined by the state of the PM0 and PM1 pins

ADE7880 DEFAULT CONFIGERATION

This is a procedure adapted from the device datasheet section QUICK SETUP AS ENERGY METER based on the application.

- Configer gain registers
- Initialize WTHR, VARTHR, VATHR, VLEVEL and VNOM registers based Equation 26, Equation 37, Equation 44, Equation 22, and Equation 42, respectively, Please see the datasheet
  ADE7880 http://www.analog.com/static/imported-files/data_sheets/ADE7880.-pdf.
- Enable the data memory RAM protection, by writing 0xAD to an internal 8-bit register located
  at Address 0xE7FE, followed by a write of 0x80 to an internal 8-bit register located at Address 0xE7E3.
- Start the DSP by setting Run = 1.

6.2.2 Function Documentation

6.2.2.1 int8_t ade7880_config_reg_default (void)

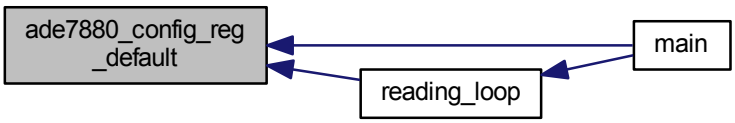
configure ADE7880 as 3-Phase, 4-Wire, Wye Distribution Systems energy meter

with the default values .

function return 0 or -1, on success or failure respectively .

Referenced by main(), and reading_loop().

Here is the caller graph for this function:



6.2.2.2 int8_t ade7880_power_mode (uint8_t *mode*)

Parameters

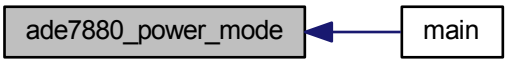
in	*mode*	indicates the selected power mode

Function to set up power mode of ADE7880.

function return 0 or -1, on success or failure respectively.

Referenced by main().

Here is the caller graph for this function:



6.2.2.3 int8_t config_cmd (uint32_t *cmd*, uint8_t *arg*, ...)

Parameters

in	*cmd*	Indicates specific configuration procedure to be execute
in	*arg*	Additional argument to control the execution
in,out	...	argument list attachment that is required by the procedure to be executed

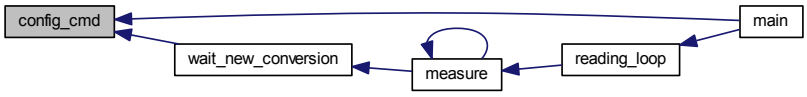
Function runs the selected configuration procedure on ADE7880.

Returns

function return 0 or -1, on success or failure respectively.

Referenced by main(), and wait_new_conversion().

Here is the caller graph for this function:



6.2.2.4 int8_t rpi_gpio_init (void)

configure used Rpi gpio pins. Pins used by the spi, will be setup up when

the module is initialized.

Returns

function return 0 or -1, on success or failure respectively.

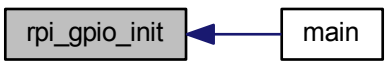
Definition at line 63 of file rpi_hwio.c.

References PIN_IRQ0, PIN_IRQ1, PIN_PM0, PIN_PM1, and PIN_SS.

Referenced by main().

```
00063                           {
00064
00065         bcm2835_gpio_fsel(PIN_SS,   BCM2835_GPIO_FSEL_OUTP);
00066         bcm2835_gpio_fsel(PIN_PM0,  BCM2835_GPIO_FSEL_OUTP);
00067         bcm2835_gpio_fsel(PIN_PM1,  BCM2835_GPIO_FSEL_OUTP);
00068         bcm2835_gpio_fsel(PIN_IRQ0, BCM2835_GPIO_FSEL_INPT);
00069         bcm2835_gpio_fsel(PIN_IRQ1, BCM2835_GPIO_FSEL_INPT);
00070
00071 return 0;
00072 }
```

Here is the caller graph for this function:



6.2.2.5 int8_t spi_init (uint8_t *chip_select*)

Parameters

in	*chip_select*	Indicates the selected chip for communication

function initializ the Rpi SPI interfce.

Returns

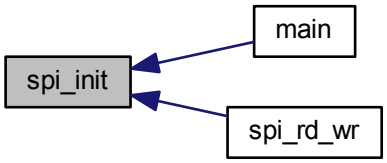
function return 0 or -1, on success or failure respectively.

Definition at line 38 of file rpi_hwio.c.

Referenced by main(), and spi_rd_wr().

```
00038                                   {
00039
00040     bcm2835_spi_begin();
00041     bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
00042     bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);                     // Data are propagated on a falling
       edge
00043
00044     bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);     // 65536 = 262.144us = 3.814697260kHz
00045     bcm2835_spi_chipSelect(chip_select);
00046     bcm2835_spi_setChipSelectPolarity(chip_select, LOW);            // SS/HSA chip pin of ade7880 should be
       low for comminication
00047
00048 return 0;
00049 }
```

Here is the caller graph for this function:



6.3 SERIAL_INTERFACE

- int8_t spi_rd_wr (uint8_t chip_select, uint8_t *tx_buffer, uint16_t len)
- uint32_t spi_read (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, uint8_t reg_len)
- int8_t spi_write (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, int32_t value, uint8_t reg_len)
- int8_t spi_ram_protection (uint8_t cmd, uint8_t chip_address)

6.3.1 Detailed Description

6.3.1.1 SERIAL INTERFACE

spi driver application layer, The module handles the data transfer between the Rpi and ADE7880":

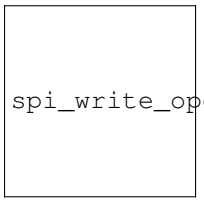
- The ADE7880 is always a slave of the communication
- The maximum serial clock frequency supported by this interface is 2.5 MHz

6.3.2 SPI Read Operation procedure

- The read operation initiated when SS/HSA pin is set to low and begins sending one byte, representing the address of the chip , on the MOSI line.
  Notice:(Bit 0 of the address byte must be 1 for a read operation)
- Next, 16-bit address of the register that is read will be sent.
- Then ADE7880 after it receives the last bit of address of the register on a low-to-high transition of SCLK, it begins to transmit its contents on the MISO line when the next SCLK high-to-low transition occurs;
- After the last bit is received, set the SS and SCLK lines high and the communication ends.
- The data lines, MOSI and MISO, go into a high impedance state.

Please see the datasheet ADE7880 http://www.analog.com/static/imported-files/data_sheets/ADE7880.pdf for more details.

6.3.3 SPI Wirte Operation procedure



- The write operation initiated when SS/HSA pin is set to low and begins sending one byte, representing the address of the chip , on the MOSI line.
  Notice:(Bit 0 of the address byte must be 0 for a read operation)
- Next, 16-bit address of the register that is written will be sent.
- And then without losing any SCLK cycle, the 32-, 16-, or 8-bit value will be sent
- After the last bit is received, set the SS and SCLK lines high and the communication ends.
- The data lines, MOSI and MISO, go into a high impedance state.

Please see the datasheet ADE7880 http://www.analog.com/static/imported-files/data_sheets/ADE7880.pdf for more details.

6.3.4 Function Documentation

6.3.4.1 int8_t spi_ram_protection (uint8_t *cmd*, uint8_t *chip_address*)

Parameters

in	*cmd*	protection enable/disable command
in	*chip_select*	Indicates the selected chip for communication

Function runs RAM protection enable or disable procedure on ADE7880

the function also verifies the communication by reading the

LAST OPERATION,LAST ADDRESS and LAST READ WRITE DATA registers and

outputs message to console if there is a failure.

Returns

function return 0 or -1, on success or failure respectively .

Definition at line 343 of file spi_ade7880_protocol.c.

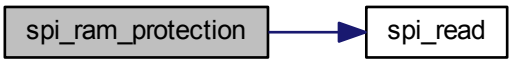
References address_byte_ut::address_all, ade7880_ram_lock_msg_ut::address_byte, ADE7880_WR, address_byte_ut::bits, CHIP_ADDRESS1, DISABLE, ENABLE, LAST_ADD, LAST_OP, LAST_RWDATA8, RAM_LOCK_MSG_LENGTH, address_byte_ut::RD_WR, spi_read(), value, and ade7880_ram_lock_msg_ut::value.

```
00343                                                                {
00344
00345
00346     int8_t result = 0;
00347
00348     uint8_t value = (cmd==ENABLE)?0x80:0x00;
00349
00350      ade7880_ram_lock_msg_ut tx_buff_1st_msg,tx_buff_2nd_msg;
00351
```

```
00352      tx_buff_1st_msg.msg_fields.address_byte.address_all = chip_address;
00353      tx_buff_1st_msg.msg_fields.address_byte.bits.RD_WR =
     ADE7880_WR;
00354      tx_buff_1st_msg.msg_fields.target_register = htons(0xE7FE);
00355      tx_buff_1st_msg.msg_fields.value = 0xAD;
00356
00357
00358      tx_buff_2nd_msg.msg_fields.address_byte.address_all = chip_address;
00359      tx_buff_2nd_msg.msg_fields.address_byte.bits.RD_WR =
     ADE7880_WR;
00360      tx_buff_2nd_msg.msg_fields.target_register = htons(0xE7E3);
00361      tx_buff_2nd_msg.msg_fields.value = value;
00362
00363
00364     #ifdef DEBUG
00365        if(debug_print&0x01){
00366        printf("\nMSG SENT: WR\n");
00367        printf("       Chip Address %02X\n",
00368        tx_buff_1st_msg.msg_fields.address_byte.address_all);
00369        printf("       1st Target Register %04X\n",
00370        ntohs(tx_buff_1st_msg.msg_fields.target_register)
00371        );
00372        printf("       Value sent 1st  %1X\n",
00373        ntohl(tx_buff_1st_msg.msg_fields.value));
00374
00375        printf("       2nd Target Register %04X\n",
00376        ntohs(tx_buff_2nd_msg.msg_fields.target_register)
00377        );
00378        printf("       Value sent  %01X\n",
00379        ntohl(tx_buff_2nd_msg.msg_fields.value));
00380        }
00381     #endif
00382
00383
00384     bcm2835_spi_transfern(tx_buff_1st_msg.msg_all,RAM_LOCK_MSG_LENGTH);
00385     bcm2835_spi_transfern(tx_buff_2nd_msg.msg_all,RAM_LOCK_MSG_LENGTH);
00386
00387
00388     uint8_t  result8  =0;
00389     uint16_t result16 =0;
00390
00391      #ifdef DEBUG
00392      if(!(debug_print&0x80))
00393      debug_print = DISABLE;
00394        printf("\nvarifying last operation  ...\n");
00395      #endif
00396      if((result8 =(uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
     LAST_OP,sizeof(uint16_t))))
00397      !=0xCA /*LAST_OP!=WR*/){
00398
00399      #ifdef DEBUG
00400        printf("\n                                                    <---  WR failure : LAST_OP \n");
00401        printf("        LAST_OP value  :%02X\n",result8);
00402      #endif
00403      result = -1;
00404      }else{
00405       #ifdef DEBUG
00406        printf("        LAST_OP value  :%02X\n",result8);
00407      #endif
00408      }
00409
00410      #ifdef DEBUG
00411        printf("\nvarifying last accessed register  ...\n");
00412      #endif
00413      if((result16 =(uint16_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
     LAST_ADD,sizeof(uint16_t))))!=0xE7E3){
00414
00415      #ifdef DEBUG
00416        printf("\n                                                    <---  WR failure : LAST_ADD\n");
00417        printf("        Target Register :%04X\n", 0xE7E3);
00418        printf("        LAST_ADD value  :%04X\n",result16);
00419      #endif
00420      result = -1;
00421      }else{
00422       #ifdef DEBUG
00423         printf("       LAST_ADD value  :%04X\n",result16);
00424       #endif
00425      }
00426
00427
00428      #ifdef DEBUG
00429        printf("\nvarifying last accessed register  value:\n");
00430      #endif
00431
00432      result8 = (uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
     LAST_RWDATA8,sizeof(uint8_t))&0x000000FF);
00433
```

```
00434
00435       if(result8!=value)
00436           result = -1;
00437       #ifdef DEBUG
00438         if(result8!=value)
00439         printf("\n                                                    <---  WR failure : LAST_RWDATA\n");
00440         printf("       Sent Value        :%01X\n", value);
00441         printf("       LAST_RWDATA value :%01X\n",result8);
00442
00443         if(!(debug_print&0x80))
00444             debug_print = ENABLE;
00445       #endif
00446
00447
00448
00449     return result;
00450 }
```

Here is the call graph for this function:



6.3.4.2 int8_t spi_rd_wr (uint8_t *chip_select*, uint8_t * *tx_buffer*, uint16_t *len*)

Parameters

in	*chip_select*	Indicates the selected chip for communication
in	*tx_buffer*	Pointer to transmit buffer
in	*len*	length of the data

initalizes spi of RPi , sends out data from tx_buffer and also receive back data into rc_buffer and end spi to return the normal gpio functionality.

Returns

function return 0 or -1, on success or failure respectively.

Definition at line 114 of file spi_ade7880_protocol.c.

References PIN_SS, and spi_init().

```
00115 {
00116
00117
00118         if(!spi_init(chip_select)){
00119
00120            bcm2835_spi_transfern(tx_buffer,sizeof(tx_buffer));
00121            return 0;
00122         }else
00123            return -1;
00124
00125         bcm2835_spi_end();
00126
00127         bcm2835_gpio_write(PIN_SS, HIGH);
00128 }
```

Here is the call graph for this function:

6.3.4.3 uint32_t spi_read (uint8_t *chip_select,* uint8_t *chip_address,* uint16_t *target_register,* uint8_t *reg_len*)

Parameters

in	*chip_select*	Indicates the selected chip for communication
in	*chip_address*	the chip address or id of the used ADE7880
in	*target_register*	the address of the register to read from
in	*reg_len*	the size of the register in byte

reads the specified register value from the selected ade7880 chip, according to the recommended SPI read Operation procedure, see the device datasheet for more detail.

Returns

function return ther resulting value on success and on failure 0x0F000000.

Parameters

in	*chip_select*	Indicates the selected chip for communication
in	*chip_address*	the chip address or id of the used ADE7880
in	*target_register*	the address of the register to update
in	*value*	value to be written
in	*reg_len*	the size of the register in byte

update the specified register value on the selected ade7880 chip

according to the recommended SPI write Operation procedure,

see the device datasheet for more detail

the function will also verify the communication by reading the

LAST OPERATION,LAST ADDRESS and LAST READ WRITE DATA registers and

outputs message to console if there is a failure.

Remarks

the function dosen't consider 24bit signed value regisers on the device,
if it become necessory to wright to regisers of this kind remember to modify.

Returns

function return 0 or -1, on success or failure respectively .

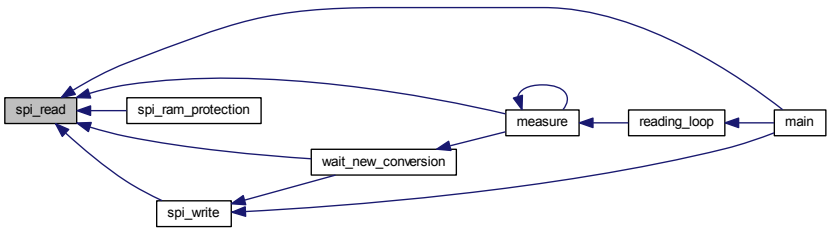
Definition at line 147 of file spi_ade7880_protocol.c.

References address_byte_ut::address_all, ade7880_read_tx_buff_ut::address_byte, ADE7880_RD, address_byte_ut::bits, DUMMY_MSG, RD_MSG_LENGTH, address_byte_ut::RD_WR, ade7880_read_tx_buff_ut::reg16, ade7880_read_tx_buff_ut::reg32, and ade7880_read_tx_buff_ut::reg8.

Referenced by main(), measure(), spi_ram_protection(), spi_write(), and wait_new_conversion().

```
00147                                                                                                                {
00148
00149   uint32_t result= 0;
00150
00151     ade7880_read_tx_buff_ut tx_buff;
00152
00153     tx_buff.msg_fields.address_byte.address_all = chip_address;
00154     tx_buff.msg_fields.address_byte.bits.RD_WR = ADE7880_RD;
00155     tx_buff.msg_fields.target_register = htons(target_register);
00156     tx_buff.msg_fields.value.reg32 = DUMMY_MSG;
00157
00158     #ifdef DEBUG
00159      if((debug_print&0x01)){
00160        printf("\nMSG SENT  : RD\n");
00161
00162        printf("        Chip Address %02X\n",tx_buff.msg_fields.address_byte.
      address_all);
00163        printf("        Target Register %04X\n",ntohs(tx_buff.msg_fields.target_register));
00164        printf("        DUMMY %08X \n",tx_buff.msg_fields.value.reg32);
00165        }
00166     #endif
00167
00168     bcm2835_spi_transfern(tx_buff.msg_all,RD_MSG_LENGTH + reg_len);
00169
00170
00171
00172
00173
00174
00175     result =  (reg_len == sizeof(uint32_t))?((uint32_t)ntohl(tx_buff.msg_fields.value.
      reg32))
00176               :((reg_len == sizeof(uint16_t))?((uint32_t)ntohs(tx_buff.msg_fields.value.
      reg16))
00177               :((reg_len == sizeof(uint8_t))?((uint32_t)tx_buff.msg_fields.value.
      reg8):0x0F000000/*ERROR*/));
00178
00179
00180     #ifdef DEBUG
00181      if(debug_print&0x01 || 1){
00182        printf("\nMSG REPLAY : RD\n");
00183        printf("        REPLAY :%X\n",result);
00184      }
00185     #endif
00186
00187
00188
00189
00190     return result;
00191
00192 }
```

Here is the caller graph for this function:



6.3.4.4 int8_t spi_write (uint8_t *chip_select*, uint8_t *chip_address*, uint16_t *target_register*, int32_t *value*, uint8_t *reg_len*)

Definition at line 221 of file spi_ade7880_protocol.c.

References address_byte_ut::address_all, ade7880_write_tx_buff_ut::address_byte, ADE7880_WR, address_byte_ut::bits, CHIP_ADDRESS1, DISABLE, ENABLE, LAST_ADD, LAST_OP, LAST_RWDATA16, LAST_RWDATA32, LAST_RWDATA8, address_byte_ut::RD_WR, ade7880_write_tx_buff_ut::reg16, ade7880_write_tx_buff_ut::reg32, ade7880_write_tx_buff_ut::reg8, spi_read(), value, and WR_MSG_LENGTH.

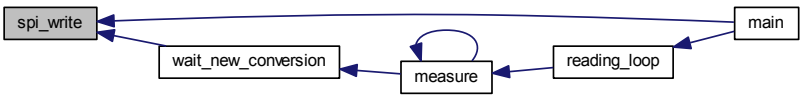
Referenced by main(), and wait_new_conversion().

```
00221
        {
00222
00223
00224     int8_t result = 0;
00225     ade7880_write_tx_buff_ut  tx_buff;
00226
00227      tx_buff.msg_fields.address_byte.address_all = chip_address;
00228      tx_buff.msg_fields.address_byte.bits.RD_WR =
      ADE7880_WR;
00229      tx_buff.msg_fields.target_register = htons(target_register);
00230
00231     if(reg_len == sizeof(uint8_t))
00232       tx_buff.msg_fields.value.reg8  = (uint8_t)value;
00233     else if(reg_len == sizeof(uint16_t))
00234       tx_buff.msg_fields.value.reg16 = htons((uint16_t)value);
00235     else if(reg_len == sizeof(uint32_t))
00236       tx_buff.msg_fields.value.reg32 = htonl((uint32_t)value);
00237
00238     #ifdef DEBUG
00239       if(debug_print&0x01){
00240        printf("\nMSG SENT: WR\n");
00241        printf("       Chip Address %02X\n",
00242        tx_buff.msg_fields.address_byte.address_all);
00243        printf("       Target Register %04X\n",
00244        ntohs(tx_buff.msg_fields.target_register));
00245        printf("       Value sent  %08X\n",
00246        ntohl(tx_buff.msg_fields.value.reg32));
00247        }
00248     #endif
00249
00250     bcm2835_spi_transfern(tx_buff.msg_all,WR_MSG_LENGTH + reg_len);
00251
00252
00253
00254     uint8_t  result8  =0;
00255     uint16_t result16 =0;
00256     int32_t result32 =0;
00257
00258      #ifdef DEBUG
00259      if(!(debug_print&0x80))
00260      debug_print = DISABLE;
00261        printf("\nvarifying last operation  ...\n");
00262      #endif
00263      if((result8 =(uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_OP,sizeof(uint16_t))))
00264      !=0xCA /*LAST_OP!=WR*/){
00265
00266      #ifdef DEBUG
00267        printf("\n                                                  <---  WR failure : LAST_OP \n");
00268        printf("        LAST_OP value  :%02X\n",result8);
00269      #endif
00270      result = -1;
00271      }else{
00272       #ifdef DEBUG
00273        printf("        LAST_OP value  :%02X\n",result8);
00274      #endif
00275      }
00276
00277      #ifdef DEBUG
00278        printf("\nvarifying last accessed register  ...\n");
00279      #endif
00280      if((result16 =(uint16_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_ADD,sizeof(uint16_t))))!=target_register){
00281
00282      #ifdef DEBUG
00283        printf("\n                                                  <---  WR failure : LAST_ADD\n");
00284        printf("        Target Register :%04X\n", target_register);
00285        printf("        LAST_ADD value  :%04X\n",result16);
00286      #endif
00287      result = -1;
00288      }else{
00289       #ifdef DEBUG
00290         printf("       LAST_ADD value  :%04X\n",result16);
00291       #endif
00292      }
00293
00294
```

```
00295      #ifdef DEBUG
00296        printf("\nvarifying last accessed register  value:\n");
00297      #endif
00298     if(reg_len == sizeof(uint8_t)){
00299         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA8,sizeof(uint8_t))&0x000000FF);
00300     }else
00301     if(reg_len == sizeof(uint16_t)){
00302         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA16,sizeof(uint16_t))&0x0000FFFF);
00303     }else
00304     if(reg_len == sizeof(uint32_t)){
00305         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA32,sizeof(uint32_t)));
00306     }
00307     //if
00308
00309      if(result32!=value)
00310          result = -1;
00311      #ifdef DEBUG
00312        if(result32!=value)
00313        printf("\n                                                    <---  WR failure : LAST_RWDATA\n");
00314        printf("       Sent Value        :%08X\n", value);
00315        printf("       LAST_RWDATA value :%08X\n",result32);
00316       if(!(debug_print&0x80))
00317        debug_print = ENABLE;
00318      #endif
00319
00320
00321     return result;
00322 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.4 SERVICE

- float measure (uint8_t cmd, uint8_t channel, float samples)
- uint16_t hex2val (uint8_t cc)
- uint16_t make16 (uint8_t *buf, uint16_t idx)
- uint8_t make8 (uint8_t *buf, uint16_t idx)

6.4.1 Detailed Description

6.4.1.1 SERVICE

This is service module containing recommended measurement reading procedures used when fetching measured data form ADE7880.And also, additional service functions used throught the program are included here

6.4.2 Function Documentation

6.4.2.1 uint16_t hex2val (uint8_t *cc*)

Parameters

in	*cc*	Hexadecimal character value to be converted to integer

Function converts hex character to the corresponding integer value.

Returns

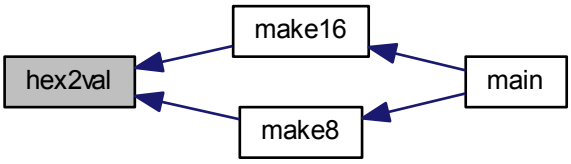
on success returns the resulting 16 bit unsigned integer value.

Definition at line 165 of file srv_cmd_handler.c.

Referenced by make16(), and make8().

```
00166 {
00167     return  (uint16_t)(((cc >= '0' ) && (cc <= '9' )) ? (cc - '0') : (cc - 'A' + 10) );
00168
00169 }
```

Here is the caller graph for this function:



6.4.2.2 uint16_t make16 (uint8_t * *buf*, uint16_t *idx*)

Parameters

in	*buf*	pointer to Hexadecimal character buffer
in	*idx*	index to first character of 16 bit hex number string (four charaters used).

Function converts string of hex character to the corresponding 16 bit unsigned integer value.

Returns

on success returns the resulting 16 bit unsigned integer value.

Definition at line 185 of file srv_cmd_handler.c.

References hex2val().

Referenced by main().

```
00186 {
00187     return ((hex2val(buf[idx+0])<<12)|(hex2val(buf[idx+1])<<8)|(
      hex2val(buf[idx+2])<<4)|(hex2val(buf[idx+3])<<0));
00188
00189 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.4.2.3 uint8_t make8 (uint8_t * *buf*, uint16_t *idx*)

Parameters

in	*buf*	pointer to Hexadecimal character buffer
in	*idx*	index to first character of 16 bit hex number string (two charaters used)

Function converts string of hex character to the corresponding 8 bit unsigned integer value.

Returns

on success returns the resulting 8 bit unsigned integer value.

Definition at line 207 of file srv_cmd_handler.c.

References hex2val().

Referenced by main().

```
00208 {
00209     return (uint8_t)((hex2val(buf[idx+0])<<4)|(hex2val(buf[idx+1]))<<0);
00210 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.4.2.4 float measure (uint8_t *cmd*, uint8_t *channel*, float *samples*)

Parameters

in	*cmd*	Indicates the target register value to read
in	*channel*	Indicates the target channel
in	*samples*	Indicates the number of samples read by the function

The Function runs recommended reading procedures designed, to enhance the accuracy of measurements by taking the average of readings taken at different instance.

Returns

on success returns the avarge of the measured value, on failure return -1 .

Definition at line 40 of file srv_cmd_handler.c.

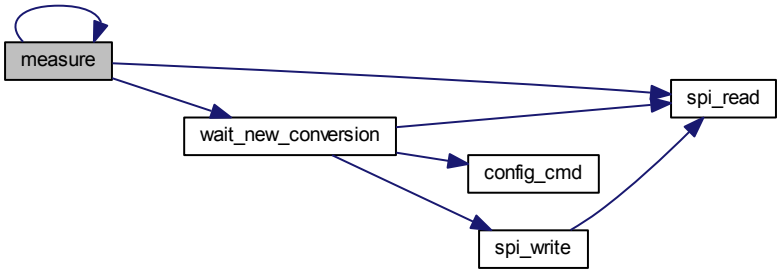
References AIRMS, AVRMS, AWATT, AWATTHR, BIRMS, BVRMS, BWATT, BWATTHR, CHIP_ADDRESS1, CIRMS, CVRMS, CWATT, CWATTHR, ILSB_CONST, measure(), PHASE_A, PHASE_ACTIVE_POWER, PHASE_ACTIVE_WH, PHASE_B, PHASE_C, PHASE_IRMS, PHASE_VRMS, spi_read(), TOTAL_ACTIVE_POWER, TOTAL_ACTIVE_WH, TRANSFORMER_RATIO, VLSB_CONST, wait_new_conversion(), WATTLSB_CONST, and WHLSB_CONST.

Referenced by measure(), and reading_loop().

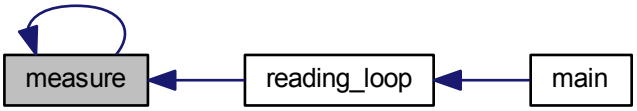
```
00040                                                       {
00041
00042
00043     float measured_val=0;
00044
00045
00046          switch(cmd)
00047         {
00048
00049                 case  PHASE_VRMS:
00050                 {
00051
00052                     uint16_t ii;
00053                     uint16_t target_reg =  (channel==PHASE_A )?AVRMS
00054                                           :(channel==PHASE_B )?BVRMS
00055                                           :(channel==PHASE_C )?CVRMS
00056                                           :0;
00057
00058                     if(target_reg==0)return -1;
00059                     for(ii=0;ii<samples;ii++){
00060                     if(wait_new_conversion()==-1)return -1;
00061                     measured_val += (uint32_t)(spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t))&0xFFFFFF)*
      TRANSFORMER_RATIO;
00062                     if(ii%50 == 0)printf("#\n");else printf("#");
00063                     }
00064                     measured_val /=samples;
00065
00066
00067                     return measured_val/VLSB_CONST;
00068
00069                 }break;
00070
00071                 case  PHASE_IRMS:
00072                 {
00073                     uint16_t ii;
00074                     uint16_t target_reg =  (channel==PHASE_A )?AIRMS
00075                                           :(channel==PHASE_B )?BIRMS
00076                                           :(channel==PHASE_C )?CIRMS
00077                                           :0;
00078                     if(target_reg==0)return -1;
00079                     for(ii=0;ii<samples;ii++){
00080                     if(wait_new_conversion()==-1)return -1;
00081                     measured_val += spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00082                     if(ii%50 == 0)printf("#\n");else printf("#");
00083
00084                     }
00085                     measured_val /=samples;
00086
00087                     return measured_val/ILSB_CONST;
00088
```

```
00089                 }break;
00090
00091
00092                 case  PHASE_ACTIVE_WH:/*considers also harmoics , trouble? then we change it
      to fundamental*/
00093                 {
00094
00095                     uint16_t target_reg =  (channel==PHASE_A )?AWATTHR
00096                                           :(channel==PHASE_B )?BWATTHR
00097                                           :(channel==PHASE_C )?CWATTHR
00098                                           :0;
00099
00100                     measured_val = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00101                     printf("#");
00102                     return measured_val/WHLSB_CONST;
00103
00104                 }break;
00105
00106                 case  TOTAL_ACTIVE_WH:/*considers also harmoics */
00107                 {
00108
00109                     uint16_t target_reg =  (channel==PHASE_A )?AWATTHR
00110                                           :(channel==PHASE_B )?BWATTHR
00111                                           :(channel==PHASE_C )?CWATTHR
00112                                           :0;
00113
00114                      return  measure(PHASE_ACTIVE_WH,
      PHASE_A,1)
00115                             +measure(PHASE_ACTIVE_WH,
      PHASE_B,1)
00116                             +measure(PHASE_ACTIVE_WH,
      PHASE_C,1);
00117
00118
00119                 }break;
00120
00121                 case  PHASE_ACTIVE_POWER:
00122                 {
00123                     uint16_t target_reg =    (channel==PHASE_A )?AWATT
00124                                             :(channel==PHASE_B )?BWATT
00125                                             :(channel==PHASE_C )?CWATT
00126                                             :0;
00127
00128                     measured_val = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00129                     printf("#");
00130                     return measured_val/WATTLSB_CONST;
00131
00132                 }break;
00133
00134                 case  TOTAL_ACTIVE_POWER:
00135                 {
00136
00137
00138                      return measure(PHASE_ACTIVE_POWER,
      PHASE_A,1)
00139                             +measure(PHASE_ACTIVE_POWER,
      PHASE_B,1);
00140                             +measure(PHASE_ACTIVE_POWER,
      PHASE_C,1);
00141
00142                 }break;
00143
00144                 default:
00145                 {
00146                   // return -1;
00147                 }break;
00148
00149         }
00150 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



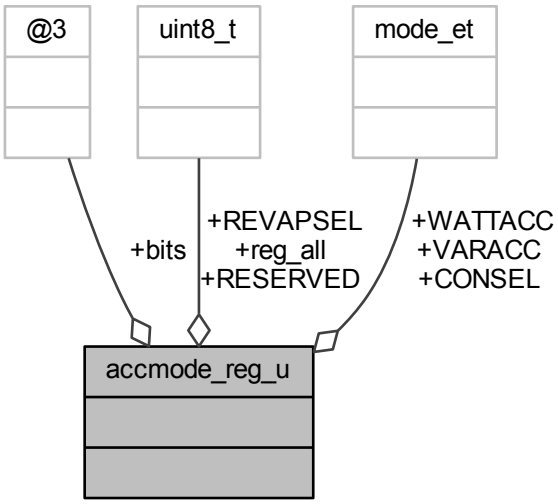
Chapter 7

Data Structure Documentation

7.1 accmode_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for accmode_reg_u:



Data Fields

- uint8_t reg_all
- struct {
  mode_et WATTACC:2
  mode_et VARACC:2
  mode_et CONSEL:2
  uint8_t REVAPSEL:1
  uint8_t RESERVED:1
  } bits

7.1.1 Detailed Description

Definition at line 134 of file ade7880_registers.h.

7.1.2 Field Documentation

7.1.2.1 struct { ... } bits

7.1.2.2 mode_et CONSEL

Definition at line 141 of file ade7880_registers.h.

7.1.2.3 uint8_t reg_all

Definition at line 136 of file ade7880_registers.h.

7.1.2.4 uint8_t RESERVED

Definition at line 143 of file ade7880_registers.h.

7.1.2.5 uint8_t REVAPSEL

Definition at line 142 of file ade7880_registers.h.

7.1.2.6 mode_et VARACC

Definition at line 140 of file ade7880_registers.h.

7.1.2.7 mode_et WATTACC

Definition at line 139 of file ade7880_registers.h.

The documentation for this union was generated from the following file:

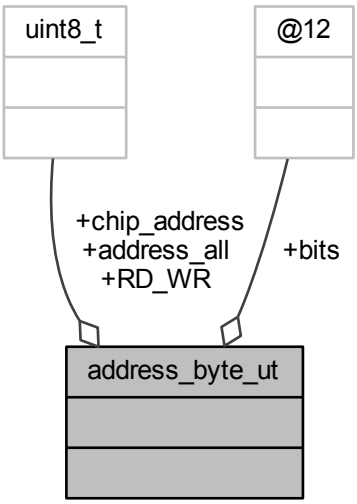
- ade7880_registers.h

7.2 address_byte_ut Union Reference

address byte of msg structure RPi to ADE7880

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for address_byte_ut:



Data Fields

- uint8_t address_all
- struct {
  uint8_t RD_WR:1
  uint8_t chip_address:7
  } bits

7.2.1 Detailed Description

address byte of msg structure RPi to ADE7880

Definition at line 21 of file spi_ade7880_protocol.h.

7.2.2 Field Documentation

7.2.2.1 uint8_t address_all

Definition at line 23 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection(), spi_read(), and spi_write().

7.2.2.2 struct { ... } bits

Referenced by spi_ram_protection(), spi_read(), and spi_write().

7.2.2.3 uint8_t chip_address

Definition at line 27 of file spi_ade7880_protocol.h.

7.2.2.4 uint8_t RD_WR

Definition at line 26 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection(), spi_read(), and spi_write().

The documentation for this union was generated from the following file:

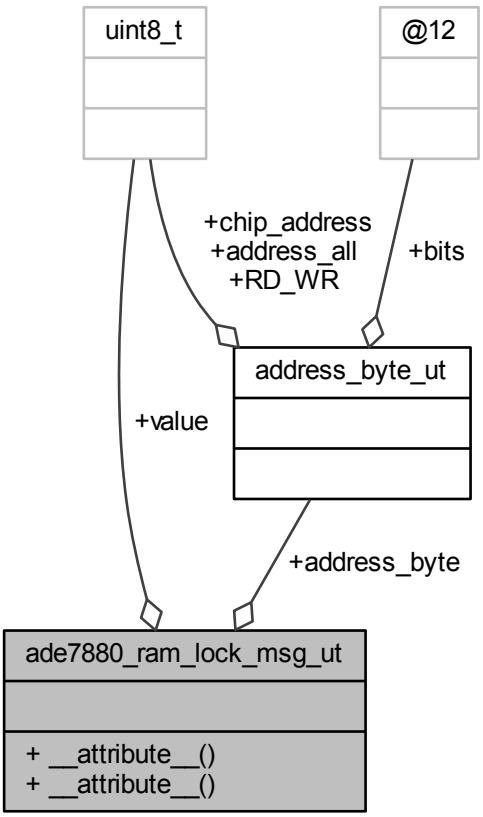
- spi_ade7880_protocol.h

7.3 ade7880_ram_lock_msg_ut Union Reference

ADE7880 write cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_ram_lock_msg_ut:



Public Member Functions

- uint8_t msg_all[sizeof(uint16_t)+2 ∗sizeof(uint8_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  uint8_t value
  } __attribute__ ((aligned))

7.3.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 183 of file spi_ade7880_protocol.h.

7.3.2 Member Function Documentation

7.3.2.1 uint8_t msg_all [sizeof(uint16_t)+ 2∗sizeof(uint8_t)] __attribute__ ((aligned))

7.3.2.2 struct ade7880_ram_lock_msg_ut::@27 __attribute__ ((aligned))

7.3.3 Field Documentation

7.3.3.1 address_byte_ut address_byte

Definition at line 189 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection().

7.3.3.2 uint8_t value

Definition at line 191 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection().

The documentation for this union was generated from the following file:

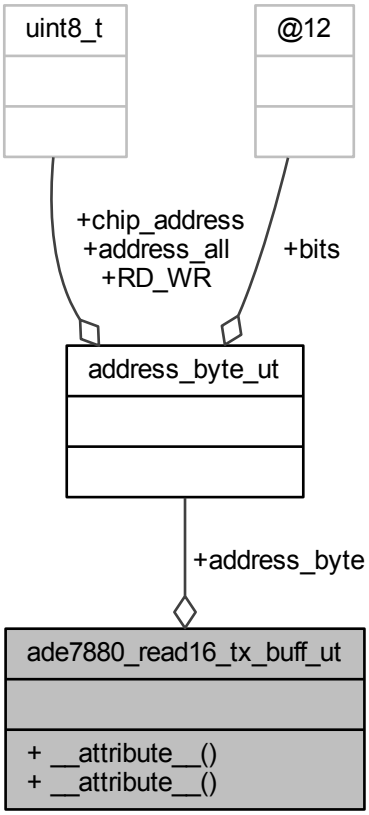
- spi_ade7880_protocol.h

7.4 ade7880_read16_tx_buff_ut Union Reference

ADE7880 read cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_read16_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[RD_MSG_LENGTH+sizeof(uint16_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  } __attribute__ ((aligned))

7.4.1 Detailed Description

ADE7880 read cmd msg structure.

Definition at line 54 of file spi_ade7880_protocol.h.

7.4.2 Member Function Documentation

7.4.2.1 uint8_t msg_all [RD_MSG_LENGTH + sizeof(uint16_t)] __attribute__ ((aligned))

7.4.2.2 struct ade7880_read16_tx_buff_ut::@14 __attribute__ ((aligned))

7.4.3 Field Documentation

7.4.3.1 address_byte_ut address_byte

Definition at line 60 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

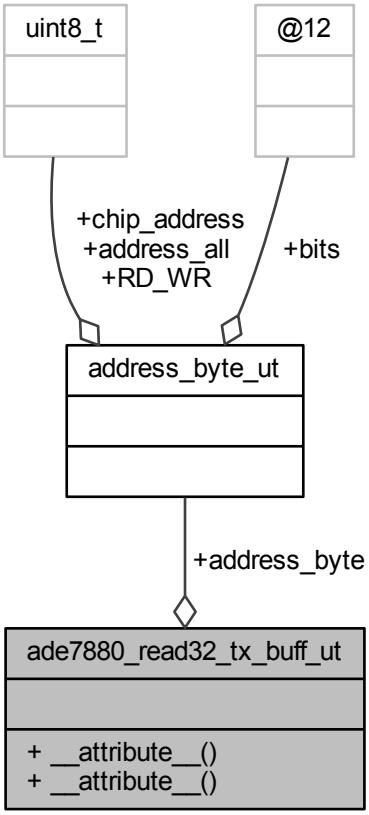
- spi_ade7880_protocol.h

7.5 ade7880_read32_tx_buff_ut Union Reference

ADE7880 read cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_read32_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[RD_MSG_LENGTH+sizeof(uint32_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte

} __attribute__ ((aligned))

7.5.1 Detailed Description

ADE7880 read cmd msg structure.

Definition at line 39 of file spi_ade7880_protocol.h.

7.5.2 Member Function Documentation

7.5.2.1 uint8_t msg_all [RD_MSG_LENGTH + sizeof(uint32_t)] __attribute__ ((aligned))

7.5.2.2 struct ade7880_read32_tx_buff_ut::@13 __attribute__ ((aligned))

7.5.3 Field Documentation

7.5.3.1 address_byte_ut address_byte

Definition at line 45 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

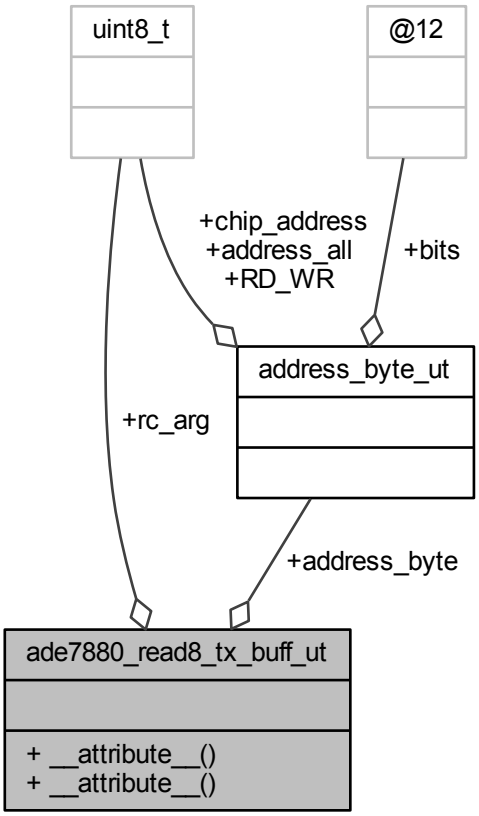
- spi_ade7880_protocol.h

7.6 ade7880_read8_tx_buff_ut Union Reference

ADE7880 read cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_read8_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[RD_MSG_LENGTH+sizeof(uint8_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  uint8_t rc_arg
  } __attribute__ ((aligned))

7.6.1 Detailed Description

ADE7880 read cmd msg structure.

Definition at line 69 of file spi_ade7880_protocol.h.

7.6.2 Member Function Documentation

7.6.2.1 uint8_t msg_all [RD_MSG_LENGTH + sizeof(uint8_t)] __attribute__ ((aligned))

7.6.2.2 struct ade7880_read8_tx_buff_ut::@15 __attribute__ ((aligned))

7.6.3 Field Documentation

7.6.3.1 address_byte_ut address_byte

Definition at line 75 of file spi_ade7880_protocol.h.

7.6.3.2 uint8_t rc_arg

Definition at line 77 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

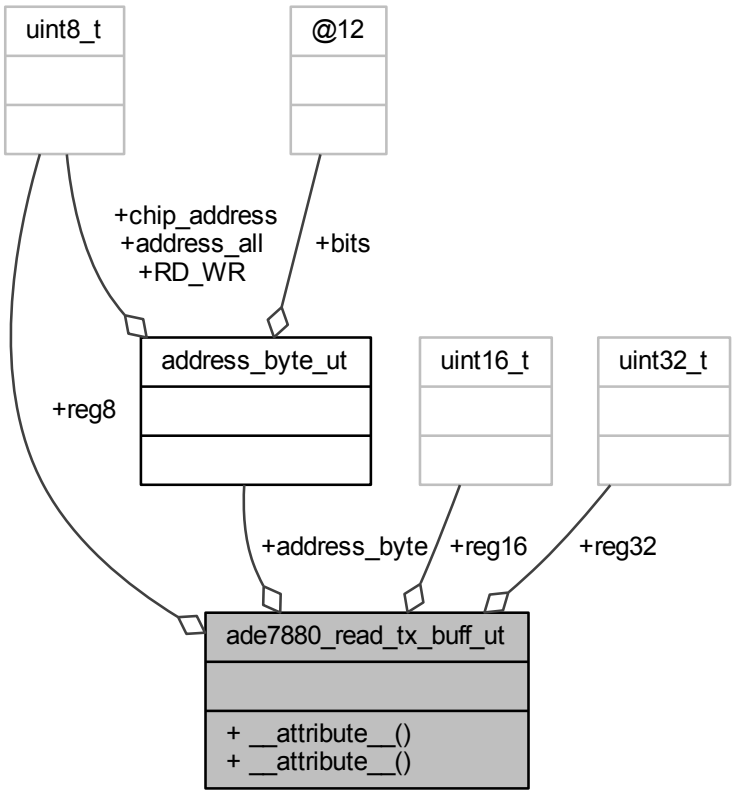
- spi_ade7880_protocol.h

7.7 ade7880_read_tx_buff_ut Union Reference

ADE7880 write cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_read_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[WR_MSG_LENGTH+sizeof(uint32_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  } __attribute__ ((aligned))

7.7.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 140 of file spi_ade7880_protocol.h.

7.7.2 Member Function Documentation

7.7.2.1 uint8_t msg_all [WR_MSG_LENGTH + sizeof(uint32_t)] __attribute__ ((aligned))

7.7.2.2 struct ade7880_read_tx_buff_ut::@21 __attribute__ ((aligned))

7.7.3 Field Documentation

7.7.3.1 address_byte_ut address_byte

Definition at line 146 of file spi_ade7880_protocol.h.

Referenced by spi_read().

7.7.3.2 uint16_t reg16

Definition at line 150 of file spi_ade7880_protocol.h.

Referenced by spi_read().

7.7.3.3 uint32_t reg32

Definition at line 151 of file spi_ade7880_protocol.h.

Referenced by spi_read().

7.7.3.4 uint8_t reg8

Definition at line 149 of file spi_ade7880_protocol.h.

Referenced by spi_read().

The documentation for this union was generated from the following file:

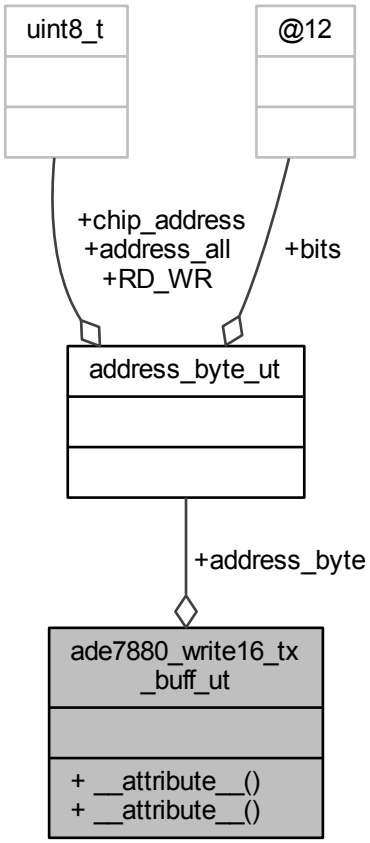
- spi_ade7880_protocol.h

7.8 ade7880_write16_tx_buff_ut Union Reference

ADE7880 write cmd msg structure.

```
#include <spi_ade7880_protocol.h>
```

Collaboration diagram for ade7880_write16_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[WR_MSG_LENGTH+sizeof(uint16_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  } __attribute__ ((aligned))

7.8.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 101 of file spi_ade7880_protocol.h.

7.8.2 Member Function Documentation

7.8.2.1 uint8_t msg_all [WR_MSG_LENGTH + sizeof(uint16_t)] __attribute__ ((aligned))

7.8.2.2 struct ade7880_write16_tx_buff_ut::@17 __attribute__ ((aligned))

7.8.3 Field Documentation

7.8.3.1 address_byte_ut address_byte

Definition at line 107 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

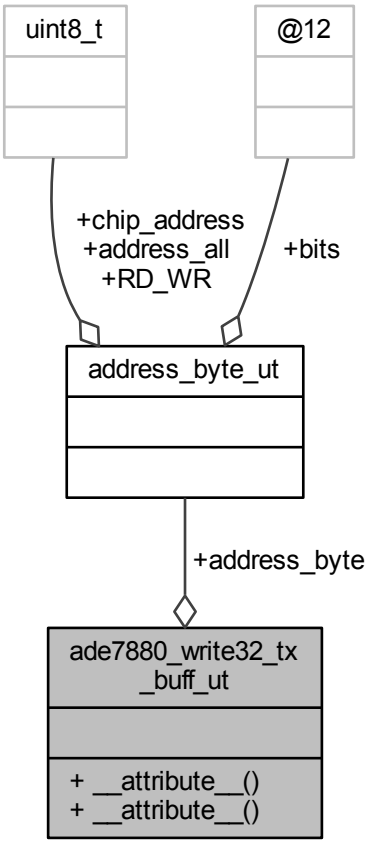
- spi_ade7880_protocol.h

7.9 ade7880_write32_tx_buff_ut Union Reference

ADE7880 write cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_write32_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[WR_MSG_LENGTH+sizeof(uint32_t)] __attribute__ ((aligned))
- struct {

address_byte_ut address_byte
} __attribute__ ((aligned))

7.9.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 85 of file spi_ade7880_protocol.h.

7.9.2 Member Function Documentation

7.9.2.1 uint8_t msg_all [WR_MSG_LENGTH + sizeof(uint32_t)] __attribute__ ((aligned))

7.9.2.2 struct ade7880_write32_tx_buff_ut::@16 __attribute__ ((aligned))

7.9.3 Field Documentation

7.9.3.1 address_byte_ut address_byte

Definition at line 91 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

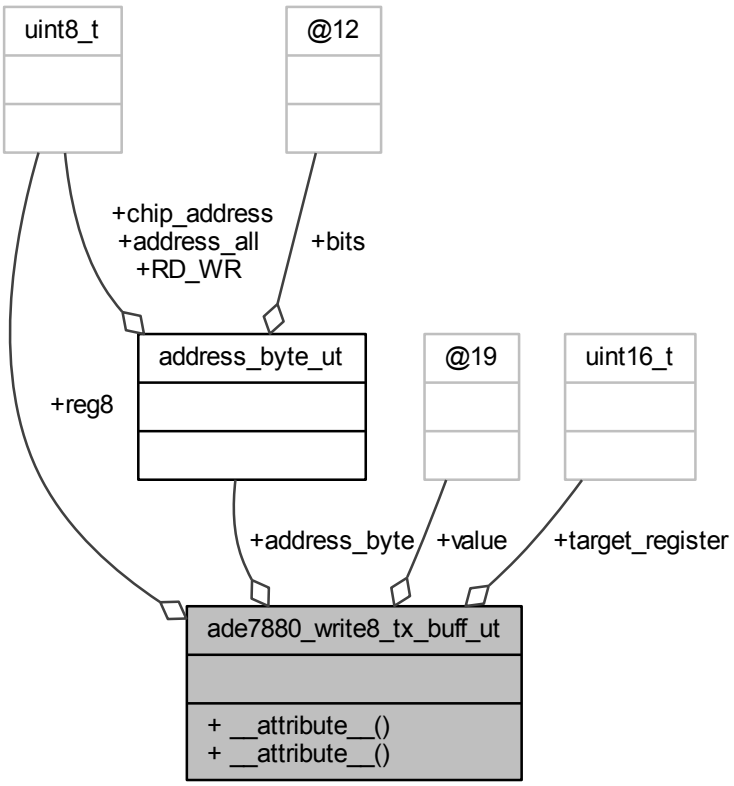
- spi_ade7880_protocol.h

7.10 ade7880_write8_tx_buff_ut Union Reference

ADE7880 write cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_write8_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[WR_MSG_LENGTH+sizeof(uint32_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  uint16_t target_register
  union {
  uint8_t reg8
  } value
  } __attribute__ ((aligned))

7.10.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 119 of file spi_ade7880_protocol.h.

7.10.2 Member Function Documentation

7.10.2.1 uint8_t msg_all [WR_MSG_LENGTH + sizeof(uint32_t)] __attribute__ ((aligned))

7.10.2.2 struct ade7880_write8_tx_buff_ut::@18 __attribute__ ((aligned))

7.10.3 Field Documentation

7.10.3.1 address_byte_ut address_byte

Definition at line 125 of file spi_ade7880_protocol.h.

7.10.3.2 uint8_t reg8

Definition at line 128 of file spi_ade7880_protocol.h.

7.10.3.3 uint16_t target_register

Definition at line 126 of file spi_ade7880_protocol.h.

7.10.3.4 union { ... } value

The documentation for this union was generated from the following file:

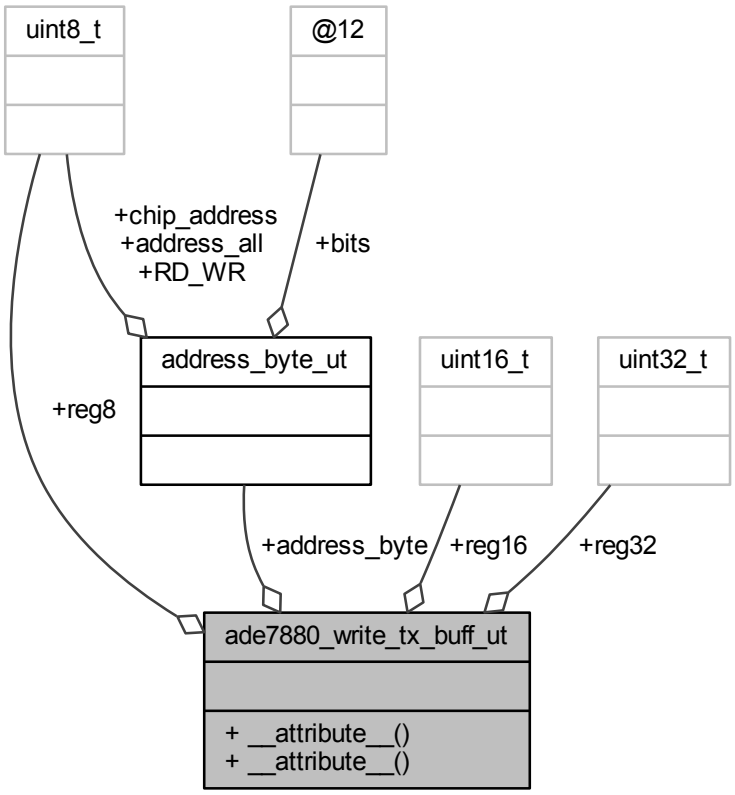
- spi_ade7880_protocol.h

7.11 ade7880_write_tx_buff_ut Union Reference

ADE7880 write cmd msg structure.

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for ade7880_write_tx_buff_ut:



Public Member Functions

- uint8_t msg_all[WR_MSG_LENGTH+sizeof(uint32_t)] __attribute__ ((aligned))
- struct {
  address_byte_ut address_byte
  } __attribute__ ((aligned))

7.11.1 Detailed Description

ADE7880 write cmd msg structure.

Definition at line 162 of file spi_ade7880_protocol.h.

7.11.2 Member Function Documentation

7.11.2.1 uint8_t msg_all [WR_MSG_LENGTH + sizeof(uint32_t)] __attribute__ ((aligned))

7.11.2.2 struct ade7880_write_tx_buff_ut::@24 __attribute__ ((aligned))

7.11.3 Field Documentation

7.11.3.1 address_byte_ut address_byte

Definition at line 168 of file spi_ade7880_protocol.h.

Referenced by spi_write().

7.11.3.2 uint16_t reg16

Definition at line 172 of file spi_ade7880_protocol.h.

Referenced by spi_write().

7.11.3.3 uint32_t reg32

Definition at line 173 of file spi_ade7880_protocol.h.

Referenced by spi_write().

7.11.3.4 uint8_t reg8

Definition at line 171 of file spi_ade7880_protocol.h.

Referenced by spi_write().

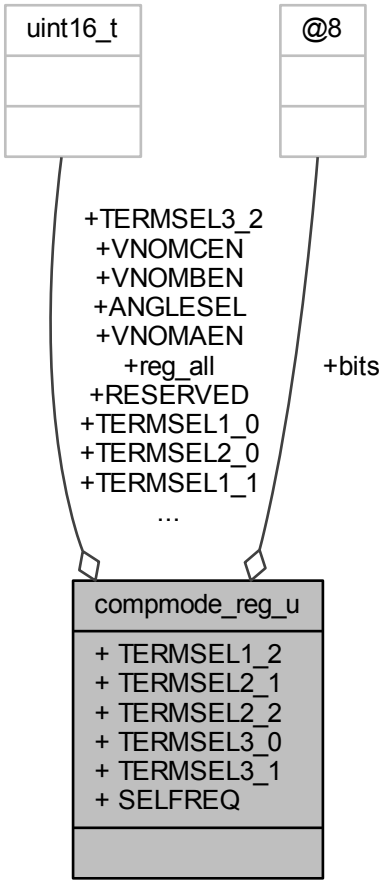
The documentation for this union was generated from the following file:

- spi_ade7880_protocol.h

7.12 compmode_reg_u Union Reference

```
#include <ade7880_registers.h>
```

Collaboration diagram for compmode_reg_u:



Data Fields

- uint16_t reg_all
- struct {
  uint16_t TERMSEL1_0:1
  uint16_t TERMSEL1_1:1
  uint16_t TERMSEL1_2:1
  uint16_t TERMSEL2_0:1
  uint16_t TERMSEL2_1:1
  uint16_t TERMSEL2_2:1
  uint16_t TERMSEL3_0:1
  uint16_t TERMSEL3_1:1
  uint16_t TERMSEL3_2:1
  uint16_t ANGLESEL:2
  uint16_t VNOMAEN:1
  uint16_t VNOMBEN:1
  uint16_t VNOMCEN:1
  uint16_t SELFREQ:1
  uint16_t RESERVED:2

} bits

7.12.1 Detailed Description

Definition at line 238 of file ade7880_registers.h.

7.12.2 Field Documentation

7.12.2.1 uint16_t ANGLESEL

Definition at line 252 of file ade7880_registers.h.

7.12.2.2 struct { ... } bits

7.12.2.3 uint16_t reg_all

Definition at line 240 of file ade7880_registers.h.

7.12.2.4 uint16_t RESERVED

Definition at line 257 of file ade7880_registers.h.

7.12.2.5 uint16_t SELFREQ

Definition at line 256 of file ade7880_registers.h.

7.12.2.6 uint16_t TERMSEL1_0

Definition at line 243 of file ade7880_registers.h.

7.12.2.7 uint16_t TERMSEL1_1

Definition at line 244 of file ade7880_registers.h.

7.12.2.8 uint16_t TERMSEL1_2

Definition at line 245 of file ade7880_registers.h.

7.12.2.9 uint16_t TERMSEL2_0

Definition at line 246 of file ade7880_registers.h.

7.12.2.10 uint16_t TERMSEL2_1

Definition at line 247 of file ade7880_registers.h.

7.12.2.11 uint16_t TERMSEL2_2

Definition at line 248 of file ade7880_registers.h.

7.12.2.12 uint16_t TERMSEL3_0

Definition at line 249 of file ade7880_registers.h.

7.12.2.13 uint16_t TERMSEL3_1

Definition at line 250 of file ade7880_registers.h.

7.12.2.14 uint16_t TERMSEL3_2

Definition at line 251 of file ade7880_registers.h.

7.12.2.15 uint16_t VNOMAEN

Definition at line 253 of file ade7880_registers.h.

7.12.2.16 uint16_t VNOMBEN

Definition at line 254 of file ade7880_registers.h.

7.12.2.17 uint16_t VNOMCEN

Definition at line 255 of file ade7880_registers.h.

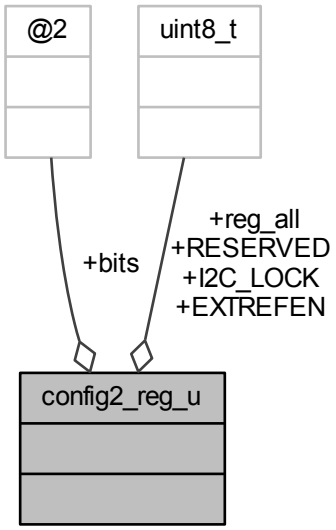
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.13 config2_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for config2_reg_u:



Data Fields

- uint8_t reg_all
- struct {
  uint8_t EXTREFEN:1
  uint8_t I2C_LOCK:1
  uint8_t RESERVED:6
  } bits

7.13.1 Detailed Description

Definition at line 109 of file ade7880_registers.h.

7.13.2 Field Documentation

7.13.2.1 struct { ... } bits

7.13.2.2 uint8_t EXTREFEN

Definition at line 114 of file ade7880_registers.h.

7.13.2.3 uint8_t I2C_LOCK

Definition at line 115 of file ade7880_registers.h.

7.13.2.4 uint8_t reg_all

Definition at line 111 of file ade7880_registers.h.

7.13.2.5 uint8_t RESERVED

Definition at line 116 of file ade7880_registers.h.

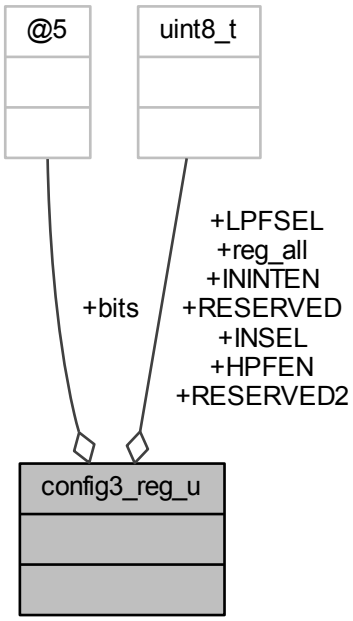
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.14 config3_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for config3_reg_u:



Data Fields

- uint8_t reg_all
- struct {
  uint8_t HPFEN:1
  uint8_t LPFSEL:1
  uint8_t INSEL:1
  uint8_t ININTEN:1
  uint8_t RESERVED:1
  uint8_t RESERVED2:3

} bits

7.14.1 Detailed Description

Definition at line 179 of file ade7880_registers.h.

7.14.2 Field Documentation

7.14.2.1 struct { ... } bits

7.14.2.2 uint8_t HPFEN

Definition at line 184 of file ade7880_registers.h.

7.14.2.3 uint8_t ININTEN

Definition at line 187 of file ade7880_registers.h.

7.14.2.4 uint8_t INSEL

Definition at line 186 of file ade7880_registers.h.

7.14.2.5 uint8_t LPFSEL

Definition at line 185 of file ade7880_registers.h.

7.14.2.6 uint8_t reg_all

Definition at line 181 of file ade7880_registers.h.

7.14.2.7 uint8_t RESERVED

Definition at line 188 of file ade7880_registers.h.

7.14.2.8 uint8_t RESERVED2

Definition at line 189 of file ade7880_registers.h.

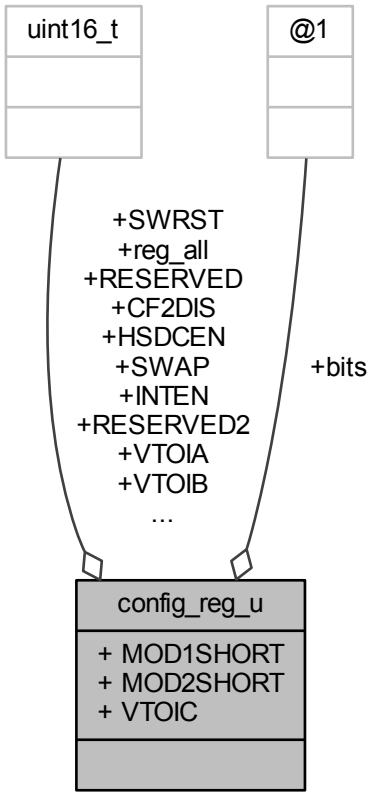
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.15 config_reg_u Union Reference

```
#include <ade7880_registers.h>
```

Collaboration diagram for config_reg_u:



Data Fields

- uint16_t reg_all
- struct {
  uint16_t INTEN:1
  uint16_t RESERVED:1
  uint16_t CF2DIS:1
  uint16_t SWAP:1
  uint16_t MOD1SHORT:1
  uint16_t MOD2SHORT:1
  uint16_t HSDCEN:1
  uint16_t SWRST:1
  uint16_t VTOIA:2
  uint16_t VTOIB:2
  uint16_t VTOIC:2
  uint16_t RESERVED2:2
  } bits

7.15.1 Detailed Description

Definition at line 84 of file ade7880_registers.h.

7.15.2 Field Documentation

7.15.2.1 struct { ... } bits

7.15.2.2 uint16_t CF2DIS

Definition at line 91 of file ade7880_registers.h.

7.15.2.3 uint16_t HSDCEN

Definition at line 95 of file ade7880_registers.h.

7.15.2.4 uint16_t INTEN

Definition at line 89 of file ade7880_registers.h.

7.15.2.5 uint16_t MOD1SHORT

Definition at line 93 of file ade7880_registers.h.

7.15.2.6 uint16_t MOD2SHORT

Definition at line 94 of file ade7880_registers.h.

7.15.2.7 uint16_t reg_all

Definition at line 86 of file ade7880_registers.h.

7.15.2.8 uint16_t RESERVED

Definition at line 90 of file ade7880_registers.h.

7.15.2.9 uint16_t RESERVED2

Definition at line 100 of file ade7880_registers.h.

7.15.2.10 uint16_t SWAP

Definition at line 92 of file ade7880_registers.h.

7.15.2.11 uint16_t SWRST

Definition at line 96 of file ade7880_registers.h.

7.15.2.12 uint16_t VTOIA

Definition at line 97 of file ade7880_registers.h.

7.15.2.13 uint16_t VTOIB

Definition at line 98 of file ade7880_registers.h.

7.15.2.14 uint16_t VTOIC

Definition at line 99 of file ade7880_registers.h.

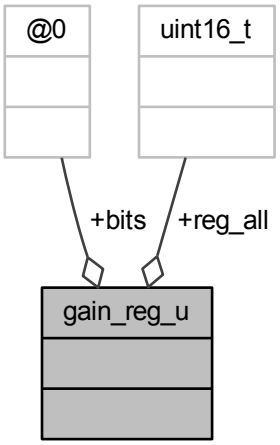
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.16 gain_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for gain_reg_u:



Data Fields

- uint16_t reg_all
- struct {
  } bits

7.16.1 Detailed Description

Definition at line 70 of file ade7880_registers.h.

7.16.2 Field Documentation

7.16.2.1 struct { ... } bits

7.16.2.2 uint16_t reg_all

Definition at line 72 of file ade7880_registers.h.

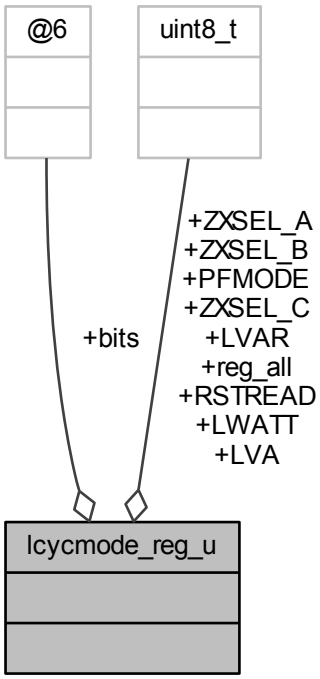
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.17 lcycmode_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for lcycmode_reg_u:



Data Fields

- uint8_t reg_all
- struct {
  uint8_t LWATT:1
  uint8_t LVAR:1
  uint8_t LVA:1
  uint8_t ZXSEL_A:1
  uint8_t ZXSEL_B:1

```
    uint8_t ZXSEL_C:1
    uint8_t RSTREAD:1
    uint8_t PFMODE:1
  } bits
```

7.17.1 Detailed Description

Definition at line 198 of file ade7880_registers.h.

7.17.2 Field Documentation

7.17.2.1 struct { ... } bits

7.17.2.2 uint8_t LVA

Definition at line 205 of file ade7880_registers.h.

7.17.2.3 uint8_t LVAR

Definition at line 204 of file ade7880_registers.h.

7.17.2.4 uint8_t LWATT

Definition at line 203 of file ade7880_registers.h.

7.17.2.5 uint8_t PFMODE

Definition at line 210 of file ade7880_registers.h.

7.17.2.6 uint8_t reg_all

Definition at line 200 of file ade7880_registers.h.

7.17.2.7 uint8_t RSTREAD

Definition at line 209 of file ade7880_registers.h.

7.17.2.8 uint8_t ZXSEL_A

Definition at line 206 of file ade7880_registers.h.

7.17.2.9 uint8_t ZXSEL_B

Definition at line 207 of file ade7880_registers.h.

7.17.2.10 uint8_t ZXSEL_C

Definition at line 208 of file ade7880_registers.h.

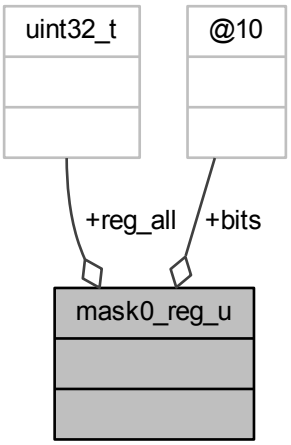
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.18 mask0_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for mask0_reg_u:



Data Fields

- uint32_t reg_all
- struct {
  } bits

7.18.1 Detailed Description

Definition at line 280 of file ade7880_registers.h.

7.18.2 Field Documentation

7.18.2.1 struct { ... } bits

7.18.2.2 uint32_t reg_all

Definition at line 282 of file ade7880_registers.h.

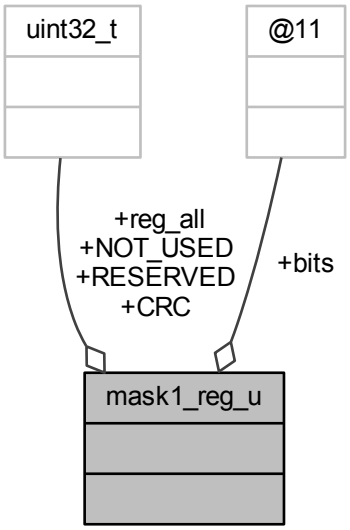
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.19 mask1_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for mask1_reg_u:



Data Fields

- uint32_t reg_all
- struct {
  uint32_t NOT_USED:24
  uint32_t CRC:1
  uint32_t RESERVED:6
  } bits

7.19.1 Detailed Description

Definition at line 294 of file ade7880_registers.h.

7.19.2 Field Documentation

7.19.2.1 struct { ... } bits

7.19.2.2 uint32_t CRC

Definition at line 300 of file ade7880_registers.h.

7.19.2.3 uint32_t NOT_USED

Definition at line 299 of file ade7880_registers.h.

7.19.2.4 uint32_t reg_all

Definition at line 296 of file ade7880_registers.h.

7.19.2.5 uint32_t RESERVED

Definition at line 301 of file ade7880_registers.h.

The documentation for this union was generated from the following file:

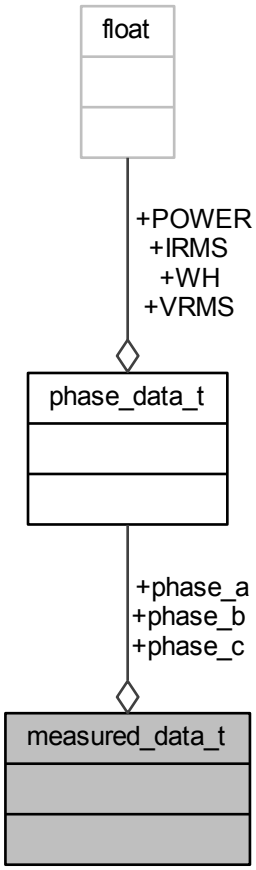
- ade7880_registers.h

7.20 measured_data_t Struct Reference

Internal.

`#include <includes.h>`

Collaboration diagram for measured_data_t:



Data Fields

- phase_data_t phase_a
- phase_data_t phase_b
- phase_data_t phase_c

7.20.1 Detailed Description

Internal.

Definition at line 25 of file includes.h.

7.20.2 Field Documentation

7.20.2.1 phase_data_t phase_a

Definition at line 28 of file includes.h.

Referenced by reading_loop(), and save_to_file().

7.20.2.2 phase_data_t phase_b

Definition at line 29 of file includes.h.

Referenced by reading_loop(), and save_to_file().

7.20.2.3 phase_data_t phase_c

Definition at line 30 of file includes.h.

Referenced by reading_loop(), and save_to_file().

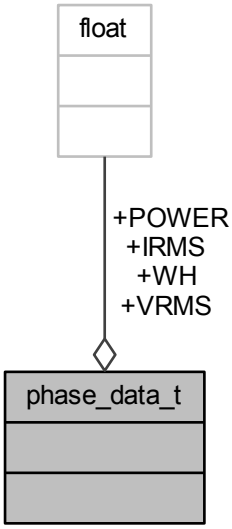
The documentation for this struct was generated from the following file:

- includes.h

7.21 phase_data_t Struct Reference

`#include <includes.h>`

Collaboration diagram for phase_data_t:



Data Fields

- float IRMS
- float VRMS
- float WH
- float POWER

7.21.1 Detailed Description

Definition at line 15 of file includes.h.

7.21.2 Field Documentation

7.21.2.1 float IRMS

Definition at line 17 of file includes.h.

Referenced by reading_loop(), and save_to_file().

7.21.2.2 float POWER

Definition at line 20 of file includes.h.

Referenced by reading_loop(), and save_to_file().

7.21.2.3 float VRMS

Definition at line 18 of file includes.h.

Referenced by reading_loop(), and save_to_file().

7.21.2.4 float WH

Definition at line 19 of file includes.h.

Referenced by reading_loop(), and save_to_file().

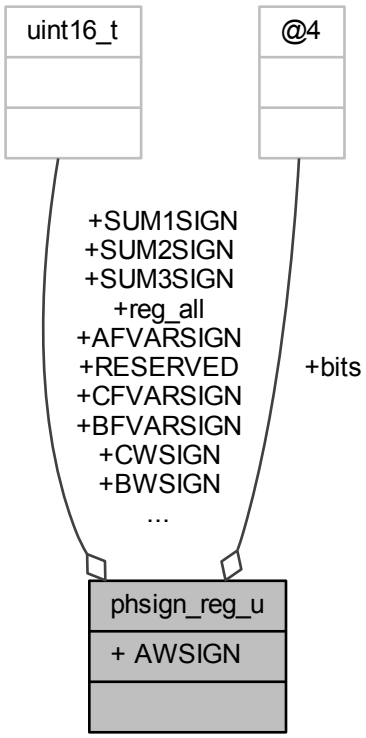
The documentation for this struct was generated from the following file:

- includes.h

7.22 phsign_reg_u Union Reference

```
#include <ade7880_registers.h>
```

Collaboration diagram for phsign_reg_u:



Data Fields

- uint16_t reg_all
- struct {
  uint16_t AWSIGN:1
  uint16_t BWSIGN:1
  uint16_t CWSIGN:1
  uint16_t SUM1SIGN:1
  uint16_t AFVARSIGN:1
  uint16_t BFVARSIGN:1
  uint16_t CFVARSIGN:1
  uint16_t SUM2SIGN:1
  uint16_t SUM3SIGN:1
  uint16_t RESERVED:7
  } bits

7.22.1 Detailed Description

Definition at line 151 of file ade7880_registers.h.

7.22.2 Field Documentation

7.22.2.1 uint16_t AFVARSIGN

Definition at line 160 of file ade7880_registers.h.

7.22.2.2 uint16_t AWSIGN

Definition at line 156 of file ade7880_registers.h.

7.22.2.3 uint16_t BFVARSIGN

Definition at line 161 of file ade7880_registers.h.

7.22.2.4 struct { ... } bits

7.22.2.5 uint16_t BWSIGN

Definition at line 157 of file ade7880_registers.h.

7.22.2.6 uint16_t CFVARSIGN

Definition at line 162 of file ade7880_registers.h.

7.22.2.7 uint16_t CWSIGN

Definition at line 158 of file ade7880_registers.h.

7.22.2.8 uint16_t reg_all

Definition at line 153 of file ade7880_registers.h.

7.22.2.9 uint16_t RESERVED

Definition at line 165 of file ade7880_registers.h.

7.22.2.10 uint16_t SUM1SIGN

Definition at line 159 of file ade7880_registers.h.

7.22.2.11 uint16_t SUM2SIGN

Definition at line 163 of file ade7880_registers.h.

7.22.2.12 uint16_t SUM3SIGN

Definition at line 164 of file ade7880_registers.h.

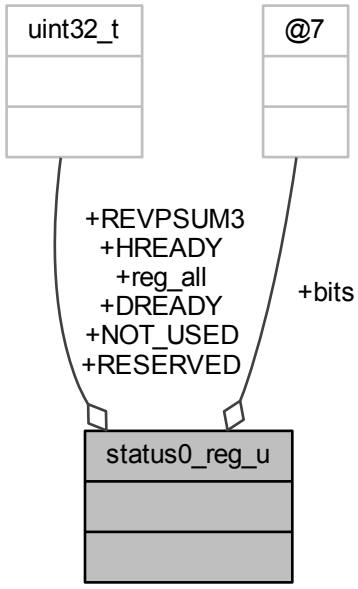
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.23 status0_reg_u Union Reference

`#include <ade7880_registers.h>`

Collaboration diagram for status0_reg_u:



Data Fields

- uint32_t reg_all
- struct {
  uint32_t NOT_USED:17
  uint32_t DREADY:1
  uint32_t REVPSUM3:1
  uint32_t HREADY:1
  uint32_t RESERVED:13
  } bits

7.23.1 Detailed Description

Definition at line 217 of file ade7880_registers.h.

7.23.2 Field Documentation

7.23.2.1 struct { ... } bits

7.23.2.2 uint32_t DREADY

Definition at line 223 of file ade7880_registers.h.

7.23.2.3 uint32_t HREADY

Definition at line 225 of file ade7880_registers.h.

7.23.2.4 uint32_t NOT_USED

Definition at line 222 of file ade7880_registers.h.

7.23.2.5 uint32_t reg_all

Definition at line 219 of file ade7880_registers.h.

Referenced by wait_new_conversion().

7.23.2.6 uint32_t RESERVED

Definition at line 226 of file ade7880_registers.h.

7.23.2.7 uint32_t REVPSUM3

Definition at line 224 of file ade7880_registers.h.

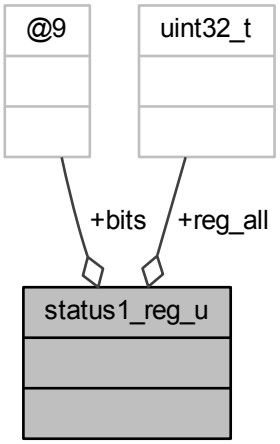
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.24 status1_reg_u Union Reference

```
#include <ade7880_registers.h>
```

Collaboration diagram for status1_reg_u:



Data Fields

- uint32_t reg_all
- struct {
  } bits

7.24.1 Detailed Description

Definition at line 266 of file ade7880_registers.h.

7.24.2 Field Documentation

7.24.2.1 struct { ... } bits

7.24.2.2 uint32_t reg_all

Definition at line 268 of file ade7880_registers.h.

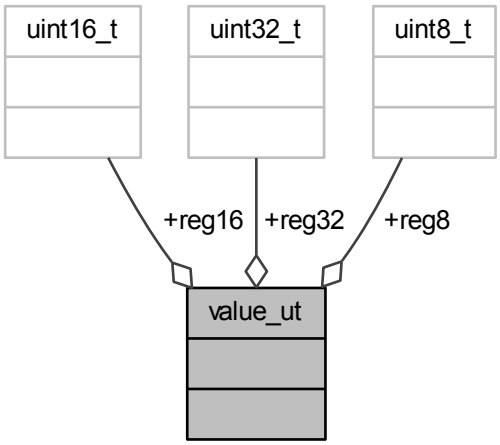
The documentation for this union was generated from the following file:

- ade7880_registers.h

7.25 value_ut Union Reference

`#include <spi_ade7880_protocol.h>`

Collaboration diagram for value_ut:



Data Fields

- uint8_t reg8

- uint16_t reg16
- uint32_t reg32

7.25.1 Detailed Description

Definition at line 32 of file spi_ade7880_protocol.h.

7.25.2 Field Documentation

7.25.2.1 uint16_t reg16

Definition at line 34 of file spi_ade7880_protocol.h.

7.25.2.2 uint32_t reg32

Definition at line 35 of file spi_ade7880_protocol.h.

7.25.2.3 uint8_t reg8

Definition at line 33 of file spi_ade7880_protocol.h.

The documentation for this union was generated from the following file:

- spi_ade7880_protocol.h

Chapter 8

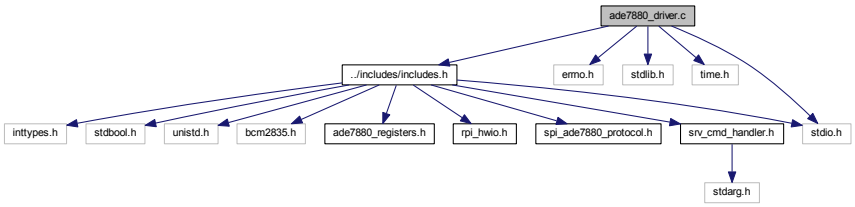
File Documentation

8.1 ade7880_driver.c File Reference

File: ade7880_driver.c

```
#include "../includes/includes.h"
#include <errno.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
```

Include dependency graph for ade7880_driver.c:



Functions

- int8_t wait_new_conversion (void)
- int16_t main (int argc, const char **argv)
- void reading_loop (void)
- int8_t save_to_file (uint16_t rpi_address, measured_data_t data, const char *fileName, uint8_t clean)
- int16_t config_recheck (const char *config_file, const char *format,...)

8.1.1 Detailed Description

File: ade7880_driver.c

Purpose:

Definition in file ade7880_driver.c.

8.2 ade7880_driver.c

```
00001
00006 #include "../includes/includes.h"
00007 #include <errno.h>
00008 #include <stdlib.h>
00009 #include <time.h>
00010 #include <stdio.h>
00011
00012
00054
00055
00121
00122 int8_t wait_new_conversion(void){
00123
00124           int8_t cc =0;
00125           int16_t result =-1;
00126
00127           struct timeval tv;
00128           status0_reg_u status0;
00129           status0.reg_all = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,STATUS0,sizeof(uint32_t));
00130
00131
00132
00133           if(status0.bits.DREADY==1){
00134
00135           result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,DISABLE))== -1)&& (cc++ < 3));
00136           //put back the value to clears status flags
00137           if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,status0.reg_all,sizeof(uint32_t))!=0)result =-1;
00138           status0.reg_all = spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,sizeof(uint32_t));
00139           result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,ENABLE))== -1)&& (cc++ < 3));
00140
00141
00142           if(result == -1){
00143                   printf("\nERROR: Couldn't write\n");
00144                   return -1;
00145           }
00146
00147
00148           }
00149
00150
00151
00152
00153           gettimeofday(&tv,NULL);
00154           uint32_t t1,t2;
00155           t1=t2 = tv.tv_usec;
00156           #ifdef DEBUG
00157           printf("\nSTATUS REG VALUE %08X,---------------------------------------- us time %lu\n",
      status0.reg_all,t1);
00158           #endif
00159           while(status0.bits.DREADY==0){ //wait till conversion is done
00160
00161             status0.reg_all = spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      STATUS0,sizeof(uint32_t));
00162             gettimeofday(&tv,NULL);
00163
00164             t2 = tv.tv_usec;
00165             #ifdef DEBUG
00166             printf("\nSTATUS REG VALUE %08X,---------------------------------------- us time %lu\n",
      status0.reg_all,t2);
00167             #endif
00168             if((t2-t1)>20000)
00169             return -1;
00170
00171           }
00172
00173
00174
```

```
00175   return 0;
00176 }
00177
00179
00180
00196
00197
00198 int16_t main(int argc, const char **argv){
00199
00200
00201
00202     int8_t cc =0;
00203     int16_t result =-1;
00204
00205     struct timeval tv;
00206
00207      if((argc < 1)|| !bcm2835_init()/*library has to be initialized*/)
00208           return -1;
00209
00210     rpi_gpio_init();
00211     ade7880_power_mode(PSM0);
00212     usleep(50);
00213
00214     if(spi_init(BCM2835_SPI_CS0)!= 0)
00215     return -1;
00216         #ifdef DEBUG
00217         uint16_t cp=0;
00218         printf("\n
      ########################################%d############################################################\n",cp++);
00219         #endif
00220
00221
00222      if(strcmp (argv[1],"config")==0)
00223      {
00224         if(ade7880_config_reg_default()!=-1)
00225              printf("\nDevice Ready to use\n");
00226      }
00227      else
00228      if(strcmp(argv[1],"read")==0)
00229      {
00230         if(argc<4){printf("\nCMD ERROR: RD\n");return-1;}
00231
00232         for(cc = 0;cc<((argc>=5)?atoi(argv[4]):1);cc++){
00233          spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,make16((uint8_t *)argv[2],2),
      atoi(argv[3]));
00234          gettimeofday(&tv,NULL);
00235          printf("<---------------------------------------------------------- %d timesamp: %lf\n",
00236          cc+1,(double)(tv.tv_sec + tv.tv_usec/(double)10E6));
00237         }
00238      return 0;
00239      }
00240      else
00241      if(strcmp(argv[1],"write")==0 )
00242      {
00243         if(argc<4){printf("\nCMD ERROR: WR\n");return-1;}
00244     int ii;
00245        for(ii=0;ii<argc;ii++)
00246            printf(" argval %d, = %s",ii,argv[ii]);
00247
00248         result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,DISABLE))== -1)&& (cc++ < 3));
00249         if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,make16((uint8_t *)argv[2],2)
00250         ,make8((uint8_t *)argv[3],2),atoi(argv[4]))!=0)
00251           result =-1;
00252          result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,ENABLE))== -1)&& (cc++ < 3));
00253          if(result == -1){
00254                     printf("ERROR: Couldn't write");
00255                       return -1;
00256          }
00257
00258      return 0;
00259      }
00260      else
00261      if(strcmp(argv[1],"run")==0)
00262      {
00263         #ifdef DEBUG
00264         if(argc >=3){
00265            if((strcmp(argv[2],"dprint=off")==0))
00266                spi_enable_msg_debug_print(DISABLE);
00267             else if((strcmp(argv[2],"dprint=on")==0))
00268                 spi_enable_msg_debug_print(ENABLE);
00269         }
00270         printf("\n
      ########################################%d############################################################\n",cp++);
00271         #endif
00272
```

```
00273         if(argc<2){printf("\nCMD ERROR: measure\n");return-1;}
00274
00275            reading_loop();
00276      }
00277
00278
00279      bcm2835_spi_end();
00280
00281      bcm2835_gpio_write(PIN_SS, HIGH);
00282
00283      bcm2835_close();
00284
00285 return 0;
00286 }
00288
00300
00301
00302 void reading_loop(void){
00303
00304   int16_t   rpi_address;
00305   char *    filename;
00306   uint32_t  cyc_time;
00307   uint8_t   loop_ctrl;
00308   uint8_t   pause;
00309   int8_t    result=0;
00310   int8_t startup = 1;
00311
00312     measured_data_t data;
00313
00314         #ifdef DEBUG
00315         printf("\n\n");
00316         #endif
00317          printf("\nEntering Main loop ...\n");
00318          while(1){
00319          printf("\n");
00320
00321          printf("\nReading Phase A values ...\n");
00322          if((data.phase_a.IRMS           = measure(PHASE_IRMS,
      PHASE_A,100))==-1)result = -1;
00323          if((data.phase_a.VRMS           = measure(PHASE_VRMS,
      PHASE_A,100))==-1)result = -1;
00324          if((data.phase_a.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_A,1))==-1)result = -1;
00325          if((data.phase_a.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_A,1))==-1)result = -1;
00326
00327          printf("\nReading Phase B values ...\n");
00328          if((data.phase_b.IRMS           = measure(PHASE_IRMS,
      PHASE_B,100))==-1)result = -1;
00329          if((data.phase_b.VRMS           = measure(PHASE_VRMS,
      PHASE_B,100))==-1)result = -1;
00330          if((data.phase_b.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_B,1))==-1)result = -1;
00331          if((data.phase_b.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_B,1))==-1)result = -1;
00332
00333          printf("\nReading Phase C values ...\n");
00334          if((data.phase_c.IRMS           = measure(PHASE_IRMS,
      PHASE_C,100))==-1)result = -1;
00335          if((data.phase_c.VRMS           = measure(PHASE_VRMS,
      PHASE_C,100))==-1)result = -1;
00336          if((data.phase_c.WH           = measure(PHASE_ACTIVE_WH,
      PHASE_C,1))==-1)result = -1;
00337          if((data.phase_c.POWER         = measure(
      PHASE_ACTIVE_POWER,PHASE_C,1))==-1)result = -1;
00338
00339         printf("\n\n\nREADINGS:\n");
00340         printf("\n");
00341         printf("\n----------------------------------------------------------------------PHASE A KWH  : %f\n"
      ,data.phase_a.WH);
00342         printf("\n----------------------------------------------------------------------PHASE A POWER: %f\n"
      ,data.phase_a.POWER);
00343         printf("\n----------------------------------------------------------------------PHASE A VRMS : %f\n"
      ,data.phase_a.VRMS);
00344         printf("\n----------------------------------------------------------------------PHASE A IRMS : %f\n"
      ,data.phase_a.IRMS);
00345         printf("\n");
00346         printf("\n----------------------------------------------------------------------PHASE B KWH  : %f\n"
      ,data.phase_b.WH);
00347         printf("\n----------------------------------------------------------------------PHASE B POWER: %f\n"
      ,data.phase_b.POWER);
00348         printf("\n----------------------------------------------------------------------PHASE B VRMS : %f\n"
      ,data.phase_b.VRMS);
00349         printf("\n----------------------------------------------------------------------PHASE B IRMS : %f\n"
      ,data.phase_b.IRMS);
00350         printf("\n");
00351         printf("\n----------------------------------------------------------------------PHASE C KWH  : %f\n"
```

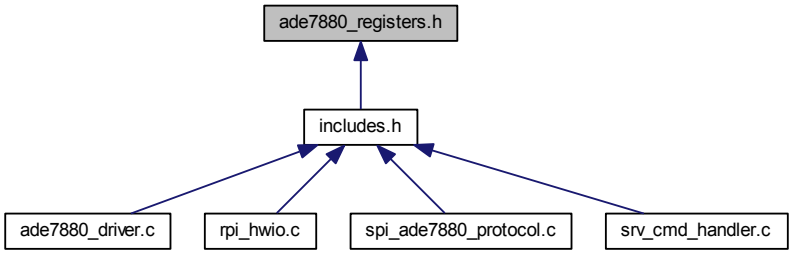
```
    ,data.phase_c.WH);
00352         printf("\n-------------------------------------------------------------------------PHASE C POWER: %f\n"
    ,data.phase_c.POWER);
00353         printf("\n-------------------------------------------------------------------------PHASE C VRMS : %f\n"
    ,data.phase_c.VRMS);
00354         printf("\n-------------------------------------------------------------------------PHASE C IRMS : %f\n"
    ,data.phase_c.IRMS);
00355
00356         if(result == -1){
00357             ade7880_config_reg_default(); //this is the only thing we do for now
00358         //  continue;
00359         }
00360
00361         do{
00362         if(config_recheck (CONFIG_FILE_NAME,
    CONFIG_CMD_FORMAT,&rpi_address ,filename, &cyc_time,&loop_ctrl,&pause)==0){
00363
00364         if(pause==0){
00365
00366           while(save_to_file(rpi_address,data, filename,
00367           startup
00376           )!=0)
00377
00378           startup = 0;
00379
00380          }
00381
00382          }
00383
00384          usleep(cyc_time);
00385         }while(pause==1);
00386
00387         if(loop_ctrl == 1)
00388         break;
00389
00390
00391
00392
00393         #ifdef DEBUG
00394         if(result!=-1)
00395         printf("\nREADING SUCESS\n");
00396         spi_enable_msg_debug_print(ENABLE);
00397         #endif
00398
00399
00400      }
00401
00402 }
00404
00419
00420
00421 int8_t save_to_file(uint16_t rpi_address,measured_data_t data, const char *
    fileName,uint8_t clean)
00422 {
00423 uint32_t dummy=0;
00424   FILE *f = fopen(fileName, (clean == 1)?"w":"a");
00425   errno = 0;
00426   if (f == NULL){
00427    warn("%s: Couldn't open file %s; %s\n",fileName, strerror (errno));
00428
00429   return -1;
00430   }
00431
00432
00433   fprintf(f, "%d;%lu;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%d;%d;%d;%d;%d;%d\n",
00434           rpi_address,
00435           (unsigned)time(NULL),
00436
00437                data.phase_a.IRMS,
00438                data.phase_a.VRMS,
00439                data.phase_a.WH,
00440                data.phase_a.POWER,
00441
00442                data.phase_b.IRMS,
00443                data.phase_b.VRMS,
00444                data.phase_b.WH,
00445                data.phase_b.POWER,
00446
00447                data.phase_c.IRMS,
00448                data.phase_c.VRMS,
00449                data.phase_c.WH,
00450                data.phase_c.POWER,
00451                DUMMY_MSG,
00452                DUMMY_MSG,
00453                DUMMY_MSG,
00454                DUMMY_MSG,
00455                DUMMY_MSG,
```

```
00456                 DUMMY_MSG
00457   );
00458
00459
00460   fclose(f);
00461   return 0;
00462 }
00464
00480
00481
00482 int16_t config_recheck(const char * config_file, const char * format, ...)
00483 {
00484
00485    int16_t result;
00486    int16_t ii,cc;
00487    FILE *f =fopen(config_file,"r");
00488    if (f == NULL) return -1;
00489
00490   char cmd_line[100];
00491   va_list args;
00492   va_start (args, format);
00493
00494   if(fgets(cmd_line,100,f)!=NULL){
00495       for(ii=0,cc=0;*(cmd_line+ii)!='\0';ii++)
00496         if(*(cmd_line+ii)==';'){*(cmd_line+ii)= ' '; ++cc;}
00497
00498     result=vsscanf (cmd_line, format, args);
00499     result = (result == cc)?0:-1;
00500   }
00501
00502   va_end (args);
00503   fclose(f);
00504   return result;
00505 }
00508
00509
00510
00511
00512
00513
00514
```

8.3 ade7880_registers.h File Reference

File: srv_ade7880_registers.h

This graph shows which files directly or indirectly include this file:



Data Structures

- union gain_reg_u
- union config_reg_u
- union config2_reg_u
- union accmode_reg_u
- union phsign_reg_u

- union config3_reg_u
- union lcycmode_reg_u
- union status0_reg_u
- union compmode_reg_u
- union status1_reg_u
- union mask0_reg_u
- union mask1_reg_u

Macros

- #define GAIN_1 0b000

  see the datasheet about this const ADE7880 http://www.analog.com/static/imported-files/data_sheets/ADE7880.pdf .
- #define GAIN_2 0b001
- #define GAIN_4 0b010
- #define GAIN_8 0b011
- #define GAIN_16 0b100
- #define MODE_0_0 0b00
- #define MODE_0_1 0b01
- #define MODE_1_0 0b10
- #define MODE_1_1 0b11
- #define DUMMIY 0xEBFF
- #define CHECKSUM 0xE51F
- #define WTHR_DEFULT 0x03

  Threshold registers.
- #define VARTHR_DEFULT 0x03

  Threshold used in phase total /fundamental reactive power data path.
- #define VATHR_DEFULT 0x03

  Threshold used in phase apparent power data path.
- #define VLEVEL_DEFULT 0x38000

  use Equation 26, Equation 37, Equation 44, Equation 22, and Equation 42 in the datasheet to determine this values
- #define WTHR 0xEA02

  Physical Address.
- #define VARTHR 0xEA03
- #define VATHR 0xEA04
- #define VLEVEL 0x439F
- #define LAST_RWDATA8 0xE7FD
- #define LAST_RWDATA16 0xE9FF
- #define LAST_RWDATA32 0xE5FF
- #define RUN 0xE228

  DSP run control register.
- #define GAIN 0xE60F
- #define CONFIG 0xE618

  ADE7880 PHSIGN Register Physical address and structure.
- #define CONFIG2 0xEC01

  ADE7880 CONFIG2 Register Physical address and structure.
- #define ACCMODE 0xE701
- #define PHSIGN 0xE617

  ADE7880 PHSIGN Register Physical address and structure.
- #define CONFIG3 0xEA00

  ADE7880 CONFIG3 Register Physical address and structure.
- #define CONFIG3_DEFULT 0x01

- #define LCYCMODE 0xE702

  ADE7880 LCYCMODE Register Physical address and structure.
- #define STATUS0 0xE502
- #define COMPMODE 0xE60E
- #define STATUS1 0xE503
- #define MASK0 0xE50A
- #define MASK1 0xE50B
- #define LAST_ADD 0xE9FE

  Registers.
- #define LAST_OP 0xEA01
- #define LAST_RWDATA_8bit 0xE7FD
- #define LAST_RWDATA_16bit 0xE9FF
- #define LAST_RWDATA_24bit 0xE5FF
- #define AVA 0xE519

  Energy Registers.
- #define BVA 0xE51A
- #define CVA 0xE51B
- #define AIGAIN 0x4380
- #define AVGAIN 0x4381
- #define BIGAIN 0x4382
- #define BVGAIN 0x4383
- #define CIGAIN 0x4384
- #define CVGAIN 0x4385
- #define NIGAIN 0x4386
- #define DICOEFF 0x4388
- #define APGAIN 0x4389
- #define AWATTOS 0x438A
- #define BPGAIN 0x438B
- #define BWATTOS 0x438C
- #define CPGAIN 0x438D
- #define CWATTOS 0x438E
- #define AIRMSOS 0x438F
- #define AVRMSOS 0x4390
- #define BIRMSOS 0x4391
- #define BVRMSOS 0x4392
- #define CIRMSOS 0x4393
- #define CVRMSOS 0x4394
- #define NIRMSOS 0x4395
- #define HPGAIN 0x4398
- #define ISUMLVL 0x4399
- #define AFWATTOS 0x43A2
- #define BFWATTOS 0x43A3
- #define CFWATTOS 0x43A4
- #define AFVAROS 0x43A5
- #define BFVAROS 0x43A6
- #define CFVAROS 0x43A7
- #define AFIRMSOS 0x43A8
- #define BFIRMSOS 0x43A9
- #define CFIRMSOS 0x43AA
- #define AFVRMSOS 0x43AB
- #define BFVRMSOS 0x43AC
- #define CFVRMSOS 0x43AD
- #define HXWATTOS 0x43AE
- #define HYWATTOS 0x43AF

- #define HZWATTOS 0x43B0
- #define HXVAROS 0x43B1
- #define HYVAROS 0x43B2
- #define HZVAROS 0x43B3
- #define HXIRMSOS 0x43B4
- #define HYIRMSOS 0x43B5
- #define HZIRMSOS 0x43B6
- #define HXVRMSOS 0x43B7
- #define HYVRMSOS 0x43B8
- #define HZVRMSOS 0x43B9
- #define AIRMS 0x43C0
- #define AVRMS 0x43C1
- #define BIRMS 0x43C2
- #define BVRMS 0x43C3
- #define CIRMS 0x43C4
- #define CVRMS 0x43C5
- #define NIRMS 0x43C6
- #define ISUM 0x43C7
- #define AWATTHR 0xE400
- #define BWATTHR 0xE401
- #define CWATTHR 0xE402
- #define AFWATTHR 0xE403
- #define BFWATTHR 0xE404
- #define CFWATTHR 0xE405
- #define AFVARHR 0xE409
- #define BFVARHR 0xE40A
- #define CFVARHR 0xE40B
- #define AVAHR 0xE40C
- #define BVAHR 0xE40D
- #define CVAHR 0xE40E
- #define IPEAK 0xE500
- #define VPEAK 0xE501
- #define AIMAV 0xE504
- #define BIMAV 0xE505
- #define CIMAV 0xE506
- #define OILVL 0xE507
- #define OVLVL 0xE508
- #define SAGLVL 0xE509
- #define IAWV 0xE50C
- #define IBWV 0xE50D
- #define ICWV 0xE50E
- #define INWV 0xE50F
- #define VAWV 0xE510
- #define VBWV 0xE511
- #define VCWV 0xE512
- #define AWATT 0xE513
- #define BWATT 0xE514
- #define CWATT 0xE515
- #define AFVAR 0xE516
- #define BFVAR 0xE517
- #define CFVAR 0xE518
- #define VNOM 0xE520
- #define PHSTATUS 0xE600
- #define ANGLE0 0xE601
- #define ANGLE1 0xE602

- #define ANGLE2 0xE603
- #define PHNOLOAD 0xE608
- #define LINECYC 0xE60C
- #define ZXTOUT 0xE60D
- #define CFMODE 0xE610
- #define CF1DEN 0xE611
- #define CF2DEN 0xE612
- #define CF3DEN 0xE613
- #define APHCAL 0xE614
- #define BPHCAL 0xE615
- #define CPHCAL 0xE616
- #define MMODE 0xE700
- #define PEAKCYC 0xE703
- #define SAGCYC 0xE704
- #define CFCYC 0xE705
- #define HSDC_CFG 0xE706
- #define Version 0xE707
- #define FVRMS 0xE880
- #define FIRMS 0xE881
- #define FWATT 0xE882
- #define FVAR 0xE883
- #define FVA 0xE884
- #define FPF 0xE885
- #define VTHDN 0xE886
- #define ITHDN 0xE887
- #define HXVRMS 0xE888
- #define HXIRMS 0xE889
- #define HXWATT 0xE88A
- #define HXVAR 0xE88B
- #define HXVA 0xE88C
- #define HXPF 0xE88D
- #define HXVHD 0xE88E
- #define HXIHD 0xE88F
- #define HYVRMS 0xE890
- #define HYIRMS 0xE891
- #define HYWATT 0xE892
- #define HYVAR 0xE893
- #define HYVA 0xE894
- #define HYPF 0xE895
- #define HYVHD 0xE896
- #define HYIHD 0xE897
- #define HZVRMS 0xE898
- #define HZIRMS 0xE899
- #define HZWATT 0xE89A
- #define HZVAR 0xE89B
- #define HZVA 0xE89C
- #define HZPF 0xE89D
- #define HZVHD 0xE89E
- #define HZIHD 0xE89F
- #define HCONFIG 0xE900
- #define APF 0xE902
- #define BPF 0xE903
- #define CPF 0xE904
- #define APERIOD 0xE905
- #define BPERIOD 0xE906

- #define CPERIOD 0xE907
- #define APNOLOAD 0xE908
- #define VARNOLOAD 0xE909
- #define VANOLOAD 0xE90A
- #define HX_reg 0xEA08
- #define HY_reg 0xEA09
- #define HZ_reg 0xEA0A
- #define LPOILVL 0xEC00

Typedefs

- typedef enum pga_et pga_et

  ADE7880 Gain Register Physical address and structure.
- typedef union accmode_reg_u accmode_reg_u

Enumerations

- enum pga_et {
  gain_1 = GAIN_1, gain_2 = GAIN_2, gain_4 = GAIN_4, gain_8 = GAIN_8,
  gain_16 = GAIN_16 }

  ADE7880 Gain Register Physical address and structure.
- enum mode_et { mode_0_0 = MODE_0_0, mode_0_1 = MODE_0_1, mode_1_0 = MODE_1_0, mode_1_1 = MODE_1_1 }

  ADE7880 Register Physical address and structure.

8.3.1 Detailed Description

File: srv_ade7880_registers.h

Purpose:

Definition in file ade7880_registers.h.

8.3.2 Macro Definition Documentation

8.3.2.1 #define ACCMODE 0xE701

Definition at line 133 of file ade7880_registers.h.

8.3.2.2 #define AFIRMSOS 0x43A8

Definition at line 364 of file ade7880_registers.h.

8.3.2.3 #define AFVAR 0xE516

Definition at line 423 of file ade7880_registers.h.

8.3.2.4 #define AFVARHR 0xE409

Definition at line 397 of file ade7880_registers.h.

8.3.2.5 #define AFVAROS 0x43A5

Definition at line 361 of file ade7880_registers.h.

8.3.2.6 #define AFVRMSOS 0x43AB

Definition at line 367 of file ade7880_registers.h.

8.3.2.7 #define AFWATTHR 0xE403

Definition at line 394 of file ade7880_registers.h.

8.3.2.8 #define AFWATTOS 0x43A2

Definition at line 358 of file ade7880_registers.h.

8.3.2.9 #define AIGAIN 0x4380

Definition at line 334 of file ade7880_registers.h.

8.3.2.10 #define AIMAV 0xE504

Definition at line 406 of file ade7880_registers.h.

8.3.2.11 #define AIRMS 0x43C0

Definition at line 382 of file ade7880_registers.h.

Referenced by measure().

8.3.2.12 #define AIRMSOS 0x438F

Definition at line 348 of file ade7880_registers.h.

8.3.2.13 #define ANGLE0 0xE601

Definition at line 430 of file ade7880_registers.h.

8.3.2.14 #define ANGLE1 0xE602

Definition at line 431 of file ade7880_registers.h.

8.3.2.15 #define ANGLE2 0xE603

Definition at line 432 of file ade7880_registers.h.

8.3.2.16 #define APERIOD 0xE905

Definition at line 492 of file ade7880_registers.h.

8.3.2.17 #define APF 0xE902

Definition at line 489 of file ade7880_registers.h.

8.3.2.18 #define APGAIN 0x4389

Definition at line 342 of file ade7880_registers.h.

8.3.2.19 #define APHCAL 0xE614

Definition at line 442 of file ade7880_registers.h.

8.3.2.20 #define APNOLOAD 0xE908

Definition at line 495 of file ade7880_registers.h.

8.3.2.21 #define AVA 0xE519

Energy Registers.

phase magnitude registers

Definition at line 328 of file ade7880_registers.h.

8.3.2.22 #define AVAHR 0xE40C

Definition at line 400 of file ade7880_registers.h.

8.3.2.23 #define AVGAIN 0x4381

Definition at line 335 of file ade7880_registers.h.

8.3.2.24 #define AVRMS 0x43C1

Definition at line 383 of file ade7880_registers.h.

Referenced by measure().

8.3.2.25 #define AVRMSOS 0x4390

Definition at line 349 of file ade7880_registers.h.

8.3.2.26 #define AWATT 0xE513

Definition at line 420 of file ade7880_registers.h.

Referenced by measure().

8.3.2.27 #define AWATTHR 0xE400

Definition at line 391 of file ade7880_registers.h.

Referenced by measure().

8.3.2.28 #define AWATTOS 0x438A

Definition at line 343 of file ade7880_registers.h.

8.3.2.29 #define BFIRMSOS 0x43A9

Definition at line 365 of file ade7880_registers.h.

8.3.2.30 #define BFVAR 0xE517

Definition at line 424 of file ade7880_registers.h.

8.3.2.31 #define BFVARHR 0xE40A

Definition at line 398 of file ade7880_registers.h.

8.3.2.32 #define BFVAROS 0x43A6

Definition at line 362 of file ade7880_registers.h.

8.3.2.33 #define BFVRMSOS 0x43AC

Definition at line 368 of file ade7880_registers.h.

8.3.2.34 #define BFWATTHR 0xE404

Definition at line 395 of file ade7880_registers.h.

8.3.2.35 #define BFWATTOS 0x43A3

Definition at line 359 of file ade7880_registers.h.

8.3.2.36 #define BIGAIN 0x4382

Definition at line 336 of file ade7880_registers.h.

8.3.2.37 #define BIMAV 0xE505

Definition at line 407 of file ade7880_registers.h.

8.3.2.38 #define BIRMS 0x43C2

Definition at line 384 of file ade7880_registers.h.

Referenced by measure().

8.3.2.39 #define BIRMSOS 0x4391

Definition at line 350 of file ade7880_registers.h.

8.3.2.40 #define BPERIOD 0xE906

Definition at line 493 of file ade7880_registers.h.

8.3.2.41 #define BPF 0xE903

Definition at line 490 of file ade7880_registers.h.

8.3.2.42 #define BPGAIN 0x438B

Definition at line 344 of file ade7880_registers.h.

8.3.2.43 #define BPHCAL 0xE615

Definition at line 443 of file ade7880_registers.h.

8.3.2.44 #define BVA 0xE51A

Definition at line 329 of file ade7880_registers.h.

8.3.2.45 #define BVAHR 0xE40D

Definition at line 401 of file ade7880_registers.h.

8.3.2.46 #define BVGAIN 0x4383

Definition at line 337 of file ade7880_registers.h.

8.3.2.47 #define BVRMS 0x43C3

Definition at line 385 of file ade7880_registers.h.

Referenced by measure().

8.3.2.48 #define BVRMSOS 0x4392

Definition at line 351 of file ade7880_registers.h.

8.3.2.49 #define BWATT 0xE514

Definition at line 421 of file ade7880_registers.h.

Referenced by measure().

8.3.2.50 #define BWATTHR 0xE401

Definition at line 392 of file ade7880_registers.h.

Referenced by measure().

8.3.2.51 #define BWATTOS 0x438C

Definition at line 345 of file ade7880_registers.h.

8.3.2.52 #define CF1DEN 0xE611

Definition at line 439 of file ade7880_registers.h.

8.3.2.53 #define CF2DEN 0xE612

Definition at line 440 of file ade7880_registers.h.

8.3.2.54 #define CF3DEN 0xE613

Definition at line 441 of file ade7880_registers.h.

8.3.2.55 #define CFCYC 0xE705

Definition at line 452 of file ade7880_registers.h.

8.3.2.56 #define CFIRMSOS 0x43AA

Definition at line 366 of file ade7880_registers.h.

8.3.2.57 #define CFMODE 0xE610

Definition at line 438 of file ade7880_registers.h.

8.3.2.58 #define CFVAR 0xE518

Definition at line 425 of file ade7880_registers.h.

8.3.2.59 #define CFVARHR 0xE40B

Definition at line 399 of file ade7880_registers.h.

8.3.2.60 #define CFVAROS 0x43A7

Definition at line 363 of file ade7880_registers.h.

8.3.2.61 #define CFVRMSOS 0x43AD

Definition at line 369 of file ade7880_registers.h.

8.3.2.62 #define CFWATTHR 0xE405

Definition at line 396 of file ade7880_registers.h.

8.3.2.63 #define CFWATTOS 0x43A4

Definition at line 360 of file ade7880_registers.h.

8.3.2.64 #define CHECKSUM 0xE51F

Definition at line 32 of file ade7880_registers.h.

8.3.2.65 #define CIGAIN 0x4384

Definition at line 338 of file ade7880_registers.h.

8.3.2.66 #define CIMAV 0xE506

Definition at line 408 of file ade7880_registers.h.

8.3.2.67 #define CIRMS 0x43C4

Definition at line 386 of file ade7880_registers.h.

Referenced by measure().

8.3.2.68 #define CIRMSOS 0x4393

Definition at line 352 of file ade7880_registers.h.

8.3.2.69 #define COMPMODE 0xE60E

Definition at line 236 of file ade7880_registers.h.

8.3.2.70 #define CONFIG 0xE618

ADE7880 PHSIGN Register Physical address and structure.

Definition at line 83 of file ade7880_registers.h.

8.3.2.71 #define CONFIG2 0xEC01

ADE7880 CONFIG2 Register Physical address and structure.

Definition at line 108 of file ade7880_registers.h.

8.3.2.72 #define CONFIG3 0xEA00

ADE7880 CONFIG3 Register Physical address and structure.

Definition at line 177 of file ade7880_registers.h.

8.3.2.73 #define CONFIG3_DEFULT 0x01

Definition at line 178 of file ade7880_registers.h.

8.3.2.74 #define CPERIOD 0xE907

Definition at line 494 of file ade7880_registers.h.

8.3.2.75 #define CPF 0xE904

Definition at line 491 of file ade7880_registers.h.

8.3.2.76 #define CPGAIN 0x438D

Definition at line 346 of file ade7880_registers.h.

8.3.2.77 #define CPHCAL 0xE616

Definition at line 444 of file ade7880_registers.h.

8.3.2.78 #define CVA 0xE51B

Definition at line 330 of file ade7880_registers.h.

8.3.2.79 #define CVAHR 0xE40E

Definition at line 402 of file ade7880_registers.h.

8.3.2.80 #define CVGAIN 0x4385

Definition at line 339 of file ade7880_registers.h.

8.3.2.81 #define CVRMS 0x43C5

Definition at line 387 of file ade7880_registers.h.

Referenced by measure().

8.3.2.82 #define CVRMSOS 0x4394

Definition at line 353 of file ade7880_registers.h.

8.3.2.83 #define CWATT 0xE515

Definition at line 422 of file ade7880_registers.h.

Referenced by measure().

8.3.2.84 #define CWATTHR 0xE402

Definition at line 393 of file ade7880_registers.h.

Referenced by measure().

8.3.2.85 #define CWATTOS 0x438E

Definition at line 347 of file ade7880_registers.h.

8.3.2.86 #define DICOEFF 0x4388

Definition at line 341 of file ade7880_registers.h.

8.3.2.87 #define DUMMIY 0xEBFF

Definition at line 31 of file ade7880_registers.h.

8.3.2.88 #define FIRMS 0xE881

Definition at line 457 of file ade7880_registers.h.

8.3.2.89 #define FPF 0xE885

Definition at line 461 of file ade7880_registers.h.

8.3.2.90 #define FVA 0xE884

Definition at line 460 of file ade7880_registers.h.

8.3.2.91 #define FVAR 0xE883

Definition at line 459 of file ade7880_registers.h.

8.3.2.92 #define FVRMS 0xE880

Definition at line 456 of file ade7880_registers.h.

8.3.2.93 #define FWATT 0xE882

Definition at line 458 of file ade7880_registers.h.

8.3.2.94 #define GAIN 0xE60F

Definition at line 69 of file ade7880_registers.h.

8.3.2.95 #define GAIN_1 0b000

see the datasheet about this const ADE7880 http://www.analog.com/static/imported-files/data_sheets/ADE7880.pdf .

Definition at line 17 of file ade7880_registers.h.

8.3.2.96 #define GAIN_16 0b100

Definition at line 21 of file ade7880_registers.h.

8.3.2.97 #define GAIN_2 0b001

Definition at line 18 of file ade7880_registers.h.

8.3.2.98 #define GAIN_4 0b010

Definition at line 19 of file ade7880_registers.h.

8.3.2.99 #define GAIN_8 0b011

Definition at line 20 of file ade7880_registers.h.

8.3.2.100 #define HCONFIG 0xE900

Definition at line 488 of file ade7880_registers.h.

8.3.2.101 #define HPGAIN 0x4398

Definition at line 355 of file ade7880_registers.h.

8.3.2.102 #define HSDC_CFG 0xE706

Definition at line 453 of file ade7880_registers.h.

8.3.2.103 #define HX_reg 0xEA08

Definition at line 500 of file ade7880_registers.h.

8.3.2.104 #define HXIHD 0xE88F

Definition at line 471 of file ade7880_registers.h.

8.3.2.105 #define HXIRMS 0xE889

Definition at line 465 of file ade7880_registers.h.

8.3.2.106 #define HXIRMSOS 0x43B4

Definition at line 376 of file ade7880_registers.h.

8.3.2.107 #define HXPF 0xE88D

Definition at line 469 of file ade7880_registers.h.

8.3.2.108 #define HXVA 0xE88C

Definition at line 468 of file ade7880_registers.h.

8.3.2.109 #define HXVAR 0xE88B

Definition at line 467 of file ade7880_registers.h.

8.3.2.110 #define HXVAROS 0x43B1

Definition at line 373 of file ade7880_registers.h.

8.3.2.111 #define HXVHD 0xE88E

Definition at line 470 of file ade7880_registers.h.

8.3.2.112 #define HXVRMS 0xE888

Definition at line 464 of file ade7880_registers.h.

8.3.2.113 #define HXVRMSOS 0x43B7

Definition at line 379 of file ade7880_registers.h.

8.3.2.114 #define HXWATT 0xE88A

Definition at line 466 of file ade7880_registers.h.

8.3.2.115 #define HXWATTOS 0x43AE

Definition at line 370 of file ade7880_registers.h.

8.3.2.116 #define HY_reg 0xEA09

Definition at line 501 of file ade7880_registers.h.

8.3.2.117 #define HYIHD 0xE897

Definition at line 479 of file ade7880_registers.h.

8.3.2.118 #define HYIRMS 0xE891

Definition at line 473 of file ade7880_registers.h.

8.3.2.119 #define HYIRMSOS 0x43B5

Definition at line 377 of file ade7880_registers.h.

8.3.2.120 #define HYPF 0xE895

Definition at line 477 of file ade7880_registers.h.

8.3.2.121 #define HYVA 0xE894

Definition at line 476 of file ade7880_registers.h.

8.3.2.122 #define HYVAR 0xE893

Definition at line 475 of file ade7880_registers.h.

8.3.2.123 #define HYVAROS 0x43B2

Definition at line 374 of file ade7880_registers.h.

8.3.2.124 #define HYVHD 0xE896

Definition at line 478 of file ade7880_registers.h.

8.3.2.125 #define HYVRMS 0xE890

Definition at line 472 of file ade7880_registers.h.

8.3.2.126 #define HYVRMSOS 0x43B8

Definition at line 380 of file ade7880_registers.h.

8.3.2.127 #define HYWATT 0xE892

Definition at line 474 of file ade7880_registers.h.

8.3.2.128 #define HYWATTOS 0x43AF

Definition at line 371 of file ade7880_registers.h.

8.3.2.129 #define HZ_reg 0xEA0A

Definition at line 502 of file ade7880_registers.h.

8.3.2.130 #define HZIHD 0xE89F

Definition at line 487 of file ade7880_registers.h.

8.3.2.131 #define HZIRMS 0xE899

Definition at line 481 of file ade7880_registers.h.

8.3.2.132 #define HZIRMSOS 0x43B6

Definition at line 378 of file ade7880_registers.h.

8.3.2.133 #define HZPF 0xE89D

Definition at line 485 of file ade7880_registers.h.

8.3.2.134 #define HZVA 0xE89C

Definition at line 484 of file ade7880_registers.h.

8.3.2.135 #define HZVAR 0xE89B

Definition at line 483 of file ade7880_registers.h.

8.3.2.136 #define HZVAROS 0x43B3

Definition at line 375 of file ade7880_registers.h.

8.3.2.137 #define HZVHD 0xE89E

Definition at line 486 of file ade7880_registers.h.

8.3.2.138 #define HZVRMS 0xE898

Definition at line 480 of file ade7880_registers.h.

8.3.2.139 #define HZVRMSOS 0x43B9

Definition at line 381 of file ade7880_registers.h.

8.3.2.140 #define HZWATT 0xE89A

Definition at line 482 of file ade7880_registers.h.

8.3.2.141 #define HZWATTOS 0x43B0

Definition at line 372 of file ade7880_registers.h.

8.3.2.142 #define IAWV 0xE50C

Definition at line 413 of file ade7880_registers.h.

8.3.2.143 #define IBWV 0xE50D

Definition at line 414 of file ade7880_registers.h.

8.3.2.144 #define ICWV 0xE50E

Definition at line 415 of file ade7880_registers.h.

8.3.2.145 #define INWV 0xE50F

Definition at line 416 of file ade7880_registers.h.

8.3.2.146 #define IPEAK 0xE500

Definition at line 403 of file ade7880_registers.h.

8.3.2.147 #define ISUM 0x43C7

Definition at line 389 of file ade7880_registers.h.

8.3.2.148 #define ISUMLVL 0x4399

Definition at line 356 of file ade7880_registers.h.

8.3.2.149 #define ITHDN 0xE887

Definition at line 463 of file ade7880_registers.h.

8.3.2.150 #define LAST_ADD 0xE9FE

Registers.

Definition at line 314 of file ade7880_registers.h.

Referenced by spi_ram_protection(), and spi_write().

8.3.2.151 #define LAST_OP 0xEA01

Definition at line 316 of file ade7880_registers.h.

Referenced by spi_ram_protection(), and spi_write().

8.3.2.152 #define LAST_RWDATA16 0xE9FF

Definition at line 49 of file ade7880_registers.h.

Referenced by spi_write().

8.3.2.153 #define LAST_RWDATA32 0xE5FF

Definition at line 50 of file ade7880_registers.h.

Referenced by spi_write().

8.3.2.154 #define LAST_RWDATA8 0xE7FD

Definition at line 48 of file ade7880_registers.h.

Referenced by spi_ram_protection(), and spi_write().

8.3.2.155 #define LAST_RWDATA_16bit 0xE9FF

Definition at line 320 of file ade7880_registers.h.

8.3.2.156 #define LAST_RWDATA_24bit 0xE5FF

Definition at line 322 of file ade7880_registers.h.

8.3.2.157 #define LAST_RWDATA_8bit 0xE7FD

Definition at line 318 of file ade7880_registers.h.

8.3.2.158 #define LCYCMODE 0xE702

ADE7880 LCYCMODE Register Physical address and structure.

Definition at line 197 of file ade7880_registers.h.

8.3.2.159 #define LINECYC 0xE60C

Definition at line 434 of file ade7880_registers.h.

8.3.2.160 #define LPOILVL 0xEC00

Definition at line 503 of file ade7880_registers.h.

8.3.2.161 #define MASK0 0xE50A

Definition at line 279 of file ade7880_registers.h.

8.3.2.162 #define MASK1 0xE50B

Definition at line 293 of file ade7880_registers.h.

8.3.2.163 #define MMODE 0xE700

Definition at line 447 of file ade7880_registers.h.

8.3.2.164 #define MODE_0_0 0b00

Definition at line 26 of file ade7880_registers.h.

8.3.2.165 #define MODE_0_1 0b01

Definition at line 27 of file ade7880_registers.h.

8.3.2.166 #define MODE_1_0 0b10

Definition at line 28 of file ade7880_registers.h.

8.3.2.167 #define MODE_1_1 0b11

Definition at line 29 of file ade7880_registers.h.

8.3.2.168 #define NIGAIN 0x4386

Definition at line 340 of file ade7880_registers.h.

8.3.2.169 #define NIRMS 0x43C6

Definition at line 388 of file ade7880_registers.h.

8.3.2.170 #define NIRMSOS 0x4395

Definition at line 354 of file ade7880_registers.h.

8.3.2.171 #define OILVL 0xE507

Definition at line 409 of file ade7880_registers.h.

8.3.2.172 #define OVLVL 0xE508

Definition at line 410 of file ade7880_registers.h.

8.3.2.173 #define PEAKCYC 0xE703

Definition at line 450 of file ade7880_registers.h.

8.3.2.174 #define PHNOLOAD 0xE608

Definition at line 433 of file ade7880_registers.h.

8.3.2.175 #define PHSIGN 0xE617

ADE7880 PHSIGN Register Physical address and structure.

Definition at line 150 of file ade7880_registers.h.

8.3.2.176 #define PHSTATUS 0xE600

Definition at line 429 of file ade7880_registers.h.

8.3.2.177 #define RUN 0xE228

DSP run control register.

Definition at line 53 of file ade7880_registers.h.

8.3.2.178 #define SAGCYC 0xE704

Definition at line 451 of file ade7880_registers.h.

8.3.2.179 #define SAGLVL 0xE509

Definition at line 411 of file ade7880_registers.h.

8.3.2.180 #define STATUS0 0xE502

Definition at line 216 of file ade7880_registers.h.

Referenced by wait_new_conversion().

8.3.2.181 #define STATUS1 0xE503

Definition at line 265 of file ade7880_registers.h.

8.3.2.182 #define VANOLOAD 0xE90A

Definition at line 497 of file ade7880_registers.h.

8.3.2.183 #define VARNOLOAD 0xE909

Definition at line 496 of file ade7880_registers.h.

8.3.2.184 #define VARTHR 0xEA03

Definition at line 45 of file ade7880_registers.h.

8.3.2.185 #define VARTHR_DEFULT 0x03

Threshold used in phase total /fundamental reactive power data path.

Definition at line 38 of file ade7880_registers.h.

8.3.2.186 #define VATHR 0xEA04

Definition at line 46 of file ade7880_registers.h.

8.3.2.187 #define VATHR_DEFULT 0x03

Threshold used in phase apparent power data path.

Definition at line 39 of file ade7880_registers.h.

8.3.2.188 #define VAWV 0xE510

Definition at line 417 of file ade7880_registers.h.

8.3.2.189 #define VBWV 0xE511

Definition at line 418 of file ade7880_registers.h.

8.3.2.190 #define VCWV 0xE512

Definition at line 419 of file ade7880_registers.h.

8.3.2.191 #define Version 0xE707

Definition at line 454 of file ade7880_registers.h.

8.3.2.192 #define VLEVEL 0x439F

Definition at line 47 of file ade7880_registers.h.

8.3.2.193 #define VLEVEL_DEFULT 0x38000

use Equation 26, Equation 37, Equation 44, Equation 22, and Equation 42 in the datasheet to determine this values

Definition at line 40 of file ade7880_registers.h.

8.3.2.194 #define VNOM 0xE520

Definition at line 427 of file ade7880_registers.h.

8.3.2.195 #define VPEAK 0xE501

Definition at line 404 of file ade7880_registers.h.

8.3.2.196 #define VTHDN 0xE886

Definition at line 462 of file ade7880_registers.h.

8.3.2.197 #define WTHR 0xEA02

Physical Address.

Definition at line 44 of file ade7880_registers.h.

8.3.2.198 #define WTHR_DEFULT 0x03

Threshold registers.

Threshold used in phase total /fundamental active power data path.

Definition at line 37 of file ade7880_registers.h.

8.3.2.199 #define ZXTOUT 0xE60D

Definition at line 435 of file ade7880_registers.h.

8.3.3 Typedef Documentation

8.3.3.1 typedef union accmode_reg_u accmode_reg_u

8.3.3.2 typedef enum pga_et pga_et

ADE7880 Gain Register Physical address and structure.

8.3.4 Enumeration Type Documentation

8.3.4.1 enum mode_et

ADE7880 Register Physical address and structure.

Enumerator

mode_0_0

mode_0_1

mode_1_0

mode_1_1

Definition at line 124 of file ade7880_registers.h.

```
00124 {  //see the datasheet about this const
00125    mode_0_0 = MODE_0_0,
00126    mode_0_1 = MODE_0_1,
00127    mode_1_0 = MODE_1_0,
00128    mode_1_1 = MODE_1_1
00129
00130 }mode_et;
00131
```

8.3.4.2 enum pga_et

ADE7880 Gain Register Physical address and structure.

Enumerator

gain_1

gain_2

gain_4

gain_8

gain_16

Definition at line 60 of file ade7880_registers.h.

```
00060 {
00061    gain_1  = GAIN_1,
00062    gain_2  = GAIN_2,
00063    gain_4  = GAIN_4,
00064    gain_8  = GAIN_8,
00065    gain_16 = GAIN_16
00066 }pga_et;
00067
```

8.4 ade7880_registers.h

```
00001
00006 #ifndef __ADE7880_REGISTERS_H
00007 #define __ADE7880_REGISTERS_H
00008
00009
00010
```

```
00011
00012
00013
00017 #define GAIN_1              0b000
00018 #define GAIN_2              0b001
00019 #define GAIN_4              0b010
00020 #define GAIN_8              0b011
00021 #define GAIN_16             0b100
00022
00023
00024
00025
00026 #define MODE_0_0            0b00
00027 #define MODE_0_1            0b01
00028 #define MODE_1_0            0b10
00029 #define MODE_1_1            0b11
00030
00031 #define DUMMIY   0xEBFF
00032 #define CHECKSUM 0xE51F
00033
00034
00035
00037 #define WTHR_DEFULT         0x03
00038 #define VARTHR_DEFULT       0x03
00039 #define VATHR_DEFULT        0x03
00040 #define VLEVEL_DEFULT       0x38000
00042
00043 #define WTHR                0xEA02
00044 #define VARTHR              0xEA03
00045 #define VATHR               0xEA04
00046 #define VLEVEL              0x439F
00047 #define LAST_RWDATA8        0xE7FD
00048 #define LAST_RWDATA16       0xE9FF
00049 #define LAST_RWDATA32       0xE5FF
00050
00052 #define RUN                 0xE228
00053
00054
00055
00056
00057
00059 typedef enum pga_et
00060 {
00061    gain_1  = GAIN_1,
00062    gain_2  = GAIN_2,
00063    gain_4  = GAIN_4,
00064    gain_8  = GAIN_8,
00065    gain_16 = GAIN_16
00066 }pga_et;
00067
00068 #define GAIN 0xE60F
00069 typedef union
00070 {
00071     uint16_t reg_all;
00072     struct
00073     {
00074        enum pga_et PGA1:3;
00075        enum pga_et PGA2:3;
00076        enum pga_et PGA3:3;
00077     }bits;
00078
00079 }gain_reg_u;
00080
00082 #define CONFIG 0xE618
00083 typedef union
00084 {
00085     uint16_t reg_all;
00086     struct
00087     {
00088        uint16_t INTEN       :1;
00089        uint16_t RESERVED    :1;
00090        uint16_t CF2DIS      :1;
00091        uint16_t SWAP        :1;
00092        uint16_t MOD1SHORT   :1;
00093        uint16_t MOD2SHORT   :1;
00094        uint16_t HSDCEN      :1;
00095        uint16_t SWRST       :1;
00096        uint16_t VTOIA       :2;
00097        uint16_t VTOIB       :2;
00098        uint16_t VTOIC       :2;
00099        uint16_t RESERVED2   :2;
00100
00101     }bits;
00102
00103 }config_reg_u;
00104
00105
```

```
00107 #define CONFIG2 0xEC01
00108 typedef union
00109 {
00110     uint8_t reg_all;
00111     struct
00112     {
00113         uint8_t EXTREFEN:1;
00114         uint8_t I2C_LOCK:1;
00115         uint8_t RESERVED:6;
00116     }bits;
00117
00118 }config2_reg_u;
00119
00120
00121
00123 typedef enum
00124 {  //see the datasheet about this const
00125    mode_0_0 = MODE_0_0,
00126    mode_0_1 = MODE_0_1,
00127    mode_1_0 = MODE_1_0,
00128    mode_1_1 = MODE_1_1
00129
00130 }mode_et;
00131
00132 #define ACCMODE 0xE701
00133 typedef union accmode_reg_u
00134 {
00135     uint8_t reg_all;
00136     struct
00137     {
00138        mode_et       WATTACC :2;
00139        mode_et       VARACC  :2;
00140        mode_et       CONSEL  :2;
00141        uint8_t      REVAPSEL:1;
00142        uint8_t      RESERVED:1;
00143
00144     }bits;
00145
00146 }accmode_reg_u;
00147
00149 #define PHSIGN 0xE617
00150 typedef union
00151 {
00152     uint16_t reg_all;
00153     struct
00154     {
00155        uint16_t AWSIGN      :1;
00156        uint16_t BWSIGN      :1;
00157        uint16_t CWSIGN      :1;
00158        uint16_t SUM1SIGN    :1;
00159        uint16_t AFVARSIGN   :1;
00160        uint16_t BFVARSIGN   :1;
00161        uint16_t CFVARSIGN   :1;
00162        uint16_t SUM2SIGN    :1;
00163        uint16_t SUM3SIGN    :1;
00164        uint16_t RESERVED    :7;
00165
00166     }bits;
00167
00168 }phsign_reg_u;
00169
00170
00171
00172
00173
00175
00176 #define CONFIG3 0xEA00
00177 #define CONFIG3_DEFULT         0x01
00178 typedef union
00179 {
00180     uint8_t reg_all;
00181     struct
00182     {
00183         uint8_t HPFEN       :1;
00184         uint8_t LPFSEL      :1;
00185         uint8_t INSEL       :1;
00186         uint8_t ININTEN     :1;
00187         uint8_t RESERVED    :1;
00188         uint8_t RESERVED2   :3;
00189
00190     }bits;
00191
00192 }config3_reg_u;
00193
00194
00196 #define LCYCMODE 0xE702
00197 typedef union
```

```
00198 {
00199     uint8_t reg_all;
00200     struct
00201     {
00202         uint8_t LWATT       :1;
00203         uint8_t LVAR        :1;
00204         uint8_t LVA         :1;
00205         uint8_t ZXSEL_A     :1;
00206         uint8_t ZXSEL_B     :1;
00207         uint8_t ZXSEL_C     :1;
00208         uint8_t RSTREAD     :1;
00209         uint8_t PFMODE      :1;
00210     }bits;
00211
00212 }lcycmode_reg_u;
00213
00214
00215 #define STATUS0 0xE502
00216 typedef union
00217 {
00218     uint32_t reg_all;
00219     struct
00220     {
00221       uint32_t NOT_USED:17;
00222       uint32_t DREADY  :1;
00223       uint32_t REVPSUM3:1;
00224       uint32_t HREADY  :1;
00225       uint32_t RESERVED:13;
00226
00227
00228     }bits;
00229
00230 }status0_reg_u;
00231
00232
00233
00234
00235 #define COMPMODE 0xE60E
00236
00237 typedef union
00238 {
00239     uint16_t reg_all;
00240     struct
00241     {
00242         uint16_t TERMSEL1_0 :1;
00243         uint16_t TERMSEL1_1 :1;
00244         uint16_t TERMSEL1_2 :1;
00245         uint16_t TERMSEL2_0 :1;
00246         uint16_t TERMSEL2_1 :1;
00247         uint16_t TERMSEL2_2 :1;
00248         uint16_t TERMSEL3_0 :1;
00249         uint16_t TERMSEL3_1 :1;
00250         uint16_t TERMSEL3_2 :1;
00251         uint16_t ANGLESEL   :2;
00252         uint16_t VNOMAEN    :1;
00253         uint16_t VNOMBEN    :1;
00254         uint16_t VNOMCEN    :1;
00255         uint16_t SELFREQ    :1;
00256         uint16_t RESERVED   :2;
00257
00258
00259     }bits;
00260
00261 }compmode_reg_u;
00262
00263
00264 #define STATUS1 0xE503
00265 typedef union
00266 {
00267     uint32_t reg_all;
00268     struct
00269     {
00270
00271
00272     }bits;
00273
00274 }status1_reg_u;
00275
00276
00277
00278 #define MASK0 0xE50A
00279 typedef union
00280 {
00281     uint32_t reg_all;
00282     struct
00283     {
00284
```

```
00285
00286
00287     }bits;
00288
00289 }mask0_reg_u;
00290
00291
00292 #define MASK1 0xE50B
00293 typedef union
00294 {
00295     uint32_t reg_all;
00296     struct
00297     {
00298      uint32_t NOT_USED :24;
00299      uint32_t CRC      :1;
00300      uint32_t RESERVED :6;
00301
00302     }bits;
00303
00304 }mask1_reg_u;
00305
00306
00307
00309
00310
00311
00312
00313 #define LAST_ADD 0xE9FE
00314
00315 #define LAST_OP 0xEA01
00316
00317 #define LAST_RWDATA_8bit 0xE7FD
00318
00319 #define LAST_RWDATA_16bit 0xE9FF
00320
00321 #define LAST_RWDATA_24bit 0xE5FF
00322
00323
00325
00327 #define AVA 0xE519
00328 #define BVA 0xE51A
00329 #define CVA 0xE51B
00330
00331
00332
00333 #define AIGAIN 0x4380
00334 #define AVGAIN 0x4381
00335 #define BIGAIN 0x4382
00336 #define BVGAIN 0x4383
00337 #define CIGAIN 0x4384
00338 #define CVGAIN 0x4385
00339 #define NIGAIN 0x4386
00340 #define DICOEFF 0x4388
00341 #define APGAIN 0x4389
00342 #define AWATTOS 0x438A
00343 #define BPGAIN 0x438B
00344 #define BWATTOS 0x438C
00345 #define CPGAIN 0x438D
00346 #define CWATTOS 0x438E
00347 #define AIRMSOS 0x438F
00348 #define AVRMSOS 0x4390
00349 #define BIRMSOS 0x4391
00350 #define BVRMSOS 0x4392
00351 #define CIRMSOS 0x4393
00352 #define CVRMSOS 0x4394
00353 #define NIRMSOS 0x4395
00354 #define HPGAIN 0x4398
00355 #define ISUMLVL 0x4399
00356
00357 #define AFWATTOS 0x43A2
00358 #define BFWATTOS 0x43A3
00359 #define CFWATTOS 0x43A4
00360 #define AFVAROS 0x43A5
00361 #define BFVAROS 0x43A6
00362 #define CFVAROS 0x43A7
00363 #define AFIRMSOS 0x43A8
00364 #define BFIRMSOS 0x43A9
00365 #define CFIRMSOS 0x43AA
00366 #define AFVRMSOS 0x43AB
00367 #define BFVRMSOS 0x43AC
00368 #define CFVRMSOS 0x43AD
00369 #define HXWATTOS 0x43AE
00370 #define HYWATTOS 0x43AF
00371 #define HZWATTOS 0x43B0
00372 #define HXVAROS 0x43B1
00373 #define HYVAROS 0x43B2
00374 #define HZVAROS 0x43B3
```

```
00375 #define HXIRMSOS 0x43B4
00376 #define HYIRMSOS 0x43B5
00377 #define HZIRMSOS 0x43B6
00378 #define HXVRMSOS 0x43B7
00379 #define HYVRMSOS 0x43B8
00380 #define HZVRMSOS 0x43B9
00381 #define AIRMS 0x43C0
00382 #define AVRMS 0x43C1
00383 #define BIRMS 0x43C2
00384 #define BVRMS 0x43C3
00385 #define CIRMS 0x43C4
00386 #define CVRMS 0x43C5
00387 #define NIRMS 0x43C6
00388 #define ISUM 0x43C7
00389
00390 #define AWATTHR 0xE400
00391 #define BWATTHR 0xE401
00392 #define CWATTHR 0xE402
00393 #define AFWATTHR 0xE403
00394 #define BFWATTHR 0xE404
00395 #define CFWATTHR 0xE405
00396 #define AFVARHR 0xE409
00397 #define BFVARHR 0xE40A
00398 #define CFVARHR 0xE40B
00399 #define AVAHR 0xE40C
00400 #define BVAHR 0xE40D
00401 #define CVAHR 0xE40E
00402 #define IPEAK 0xE500
00403 #define VPEAK 0xE501
00404
00405 #define AIMAV 0xE504
00406 #define BIMAV 0xE505
00407 #define CIMAV 0xE506
00408 #define OILVL 0xE507
00409 #define OVLVL 0xE508
00410 #define SAGLVL 0xE509
00411
00412 #define IAWV 0xE50C
00413 #define IBWV 0xE50D
00414 #define ICWV 0xE50E
00415 #define INWV 0xE50F
00416 #define VAWV 0xE510
00417 #define VBWV 0xE511
00418 #define VCWV 0xE512
00419 #define AWATT 0xE513
00420 #define BWATT 0xE514
00421 #define CWATT 0xE515
00422 #define AFVAR 0xE516
00423 #define BFVAR 0xE517
00424 #define CFVAR 0xE518
00425
00426 #define VNOM 0xE520
00427
00428 #define PHSTATUS 0xE600
00429 #define ANGLE0 0xE601
00430 #define ANGLE1 0xE602
00431 #define ANGLE2 0xE603
00432 #define PHNOLOAD 0xE608
00433 #define LINECYC 0xE60C
00434 #define ZXTOUT 0xE60D
00435
00436
00437 #define CFMODE 0xE610
00438 #define CF1DEN 0xE611
00439 #define CF2DEN 0xE612
00440 #define CF3DEN 0xE613
00441 #define APHCAL 0xE614
00442 #define BPHCAL 0xE615
00443 #define CPHCAL 0xE616
00444
00445
00446 #define MMODE 0xE700
00447
00448
00449 #define PEAKCYC 0xE703
00450 #define SAGCYC 0xE704
00451 #define CFCYC 0xE705
00452 #define HSDC_CFG 0xE706
00453 #define Version 0xE707
00454
00455 #define FVRMS 0xE880
00456 #define FIRMS 0xE881
00457 #define FWATT 0xE882
00458 #define FVAR 0xE883
00459 #define FVA 0xE884
00460 #define FPF 0xE885
00461 #define VTHDN 0xE886
```

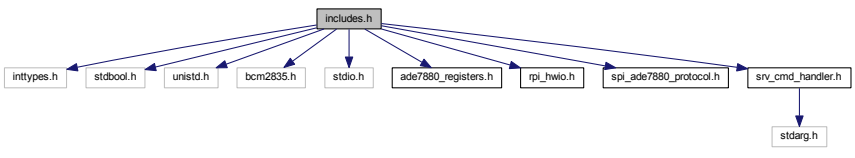
```
00462 #define ITHDN 0xE887
00463 #define HXVRMS 0xE888
00464 #define HXIRMS 0xE889
00465 #define HXWATT 0xE88A
00466 #define HXVAR 0xE88B
00467 #define HXVA 0xE88C
00468 #define HXPF 0xE88D
00469 #define HXVHD 0xE88E
00470 #define HXIHD 0xE88F
00471 #define HYVRMS 0xE890
00472 #define HYIRMS 0xE891
00473 #define HYWATT 0xE892
00474 #define HYVAR 0xE893
00475 #define HYVA 0xE894
00476 #define HYPF 0xE895
00477 #define HYVHD 0xE896
00478 #define HYIHD 0xE897
00479 #define HZVRMS 0xE898
00480 #define HZIRMS 0xE899
00481 #define HZWATT 0xE89A
00482 #define HZVAR 0xE89B
00483 #define HZVA 0xE89C
00484 #define HZPF 0xE89D
00485 #define HZVHD 0xE89E
00486 #define HZIHD 0xE89F
00487 #define HCONFIG 0xE900
00488 #define APF 0xE902
00489 #define BPF 0xE903
00490 #define CPF 0xE904
00491 #define APERIOD 0xE905
00492 #define BPERIOD 0xE906
00493 #define CPERIOD 0xE907
00494 #define APNOLOAD 0xE908
00495 #define VARNOLOAD 0xE909
00496 #define VANOLOAD 0xE90A
00497
00498
00499 #define HX_reg 0xEA08
00500 #define HY_reg 0xEA09
00501 #define HZ_reg 0xEA0A
00502 #define LPOILVL 0xEC00
00503
00504
00505 #endif /* __ADE7880_REGISTERS_H */
00506
00507
00508
00509
00510
00511
```

8.5 includes.h File Reference

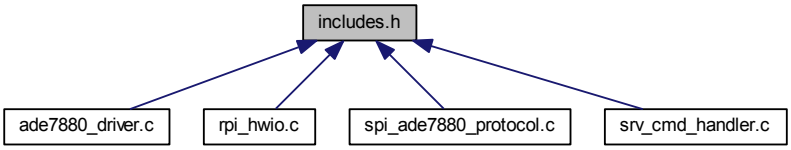
File: Includes.h

```
#include <inttypes.h>
#include <stdbool.h>
#include <unistd.h>
#include <bcm2835.h>
#include <stdio.h>
#include "ade7880_registers.h"
#include "rpi_hwio.h"
#include "spi_ade7880_protocol.h"
#include "srv_cmd_handler.h"
```

Include dependency graph for includes.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct phase_data_t
- struct measured_data_t

  Internal.

Macros

- #define TYPE(cc) (((cc >= '1')) ? ((uint8_t)) : ((uint16_t)))
- #define CHIP_ADDRESS1 0x70
- #define ENABLE 1
- #define DISABLE 0
- #define DSP_START 0xFF
- #define _VNOMAEN 0

  enable disable Apparent Power Calculation Using VNOM register value
- #define PSM0 0

  internal use power mode select macros
- #define PSM1 1
- #define PSM2 2
- #define PSM3 3
- #define VLSB_CONST 211184.71337579617834394904458599

  LSB constant for voltage reading 24 volt max is assumed.
- #define ILSB_CONST 353022.88732394

  LSB constant for current reading 30 Amp max is assumed.
- #define WHLSB_CONST (67.28)∗2.275

  LSB constant for energy reading.
- #define WATTLSB_CONST 8505

  LSB constant for power reading.
- #define TRANSFORMER_RATIO 20.56621005
- #define CONFIG_FILE_NAME "config.csv"

  defines configuration file
- #define CONFIG_CMD_FORMAT "%hd %s %lu %hhu %hhu"

  configuration command format specifier

Functions

- int8_t save_to_file (uint16_t rpi_address, measured_data_t data, const char *fileName, uint8_t clean)
- int8_t wait_new_conversion (void)
- int16_t config_recheck (const char *config_file, const char *format,...)
- void reading_loop ()

8.5.1 Detailed Description

File: Includes.h

Purpose:

contains all header files

Definition in file includes.h.

8.5.2 Macro Definition Documentation

8.5.2.1 #define _VNOMAEN 0

enable disable Apparent Power Calculation Using VNOM register value

Definition at line 46 of file includes.h.

8.5.2.2 #define CHIP_ADDRESS1 0x70

Definition at line 39 of file includes.h.

Referenced by main(), measure(), spi_ram_protection(), spi_write(), and wait_new_conversion().

8.5.2.3 #define CONFIG_CMD_FORMAT "%hd %s %lu %hhu %hhu"

configuration command format specifier

Definition at line 115 of file includes.h.

Referenced by reading_loop().

8.5.2.4 #define CONFIG_FILE_NAME "config.csv"

defines configuration file

Definition at line 114 of file includes.h.

Referenced by reading_loop().

8.5.2.5 #define DISABLE 0

Definition at line 41 of file includes.h.

Referenced by main(), spi_ram_protection(), spi_write(), and wait_new_conversion().

8.5.2.6 #define DSP_START 0xFF

Definition at line 45 of file includes.h.

8.5.2.7 #define ENABLE 1

Definition at line 40 of file includes.h.

Referenced by main(), reading_loop(), spi_ram_protection(), spi_write(), and wait_new_conversion().

8.5.2.8 #define ILSB_CONST 353022.88732394

LSB constant for current reading 30 Amp max is assumed.

Definition at line 79 of file includes.h.

Referenced by measure().

8.5.2.9 #define PSM0 0

internal use power mode select macros

Definition at line 50 of file includes.h.

Referenced by main().

8.5.2.10 #define PSM1 1

Definition at line 51 of file includes.h.

8.5.2.11 #define PSM2 2

Definition at line 52 of file includes.h.

8.5.2.12 #define PSM3 3

Definition at line 53 of file includes.h.

8.5.2.13 #define TRANSFORMER_RATIO 20.56621005

Definition at line 95 of file includes.h.

Referenced by measure().

8.5.2.14 #define TYPE(*cc*) (((cc >= '1')) ? ((uint8_t)) : ((uint16_t)))

Definition at line 13 of file includes.h.

8.5.2.15 #define VLSB_CONST 211184.71337579617834394904458599

LSB constant for voltage reading 24 volt max is assumed.

Definition at line 78 of file includes.h.

Referenced by measure().

8.5.2.16 #define WATTLSB_CONST 8505

LSB constant for power reading.

Definition at line 81 of file includes.h.

Referenced by measure().

8.5.2.17 #define WHLSB_CONST (67.28)*2.275

LSB constant for energy reading.

Definition at line 80 of file includes.h.

Referenced by measure().

8.6 includes.h

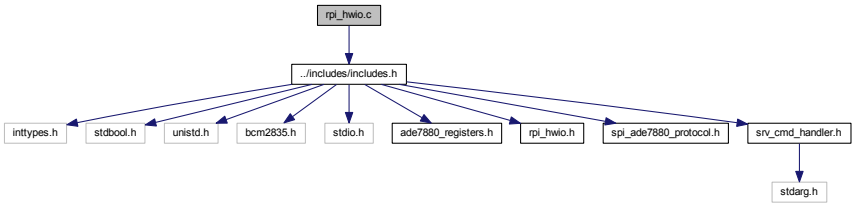
```
00001
00007 #ifndef __INCLUDES_H
00008 #define __INCLUDES_H
00009
00010
00011 #include <inttypes.h>
00012
00013 #define TYPE(cc)  (((cc >= '1' )) ? ((uint8_t)) : ((uint16_t)))
00014
00015 typedef struct
00016 {
00017      float IRMS;
00018      float VRMS;
00019      float WH;
00020      float POWER;
00021
00022 }phase_data_t;
00023
00025 typedef struct
00026 {
00027
00028      phase_data_t phase_a;
00029      phase_data_t phase_b;
00030      phase_data_t phase_c;
00031 }measured_data_t;
00032
00033
00034
00035
00036
00037
00038
00039 #define CHIP_ADDRESS1           0x70
00040 #define ENABLE                  1
00041 #define DISABLE                 0
00042
00043
00044
00045 #define DSP_START               0xFF
00046 #define _VNOMAEN                0
00049
00050 #define PSM0                    0
00051 #define PSM1                    1
00052 #define PSM2                    2
00053 #define PSM3                    3
00054
00055
00056
00078 #define VLSB_CONST              211184.71337579617834394904458599
00079 #define ILSB_CONST              353022.88732394
00080 #define WHLSB_CONST             (67.28)*2.275
00081 #define WATTLSB_CONST           8505
00095 #define TRANSFORMER_RATIO       20.56621005
00096
00097
00098
00099
00114 #define CONFIG_FILE_NAME        "config.csv"
00115 #define CONFIG_CMD_FORMAT       "%hd %s %lu %hhu %hhu"
00120 #include <stdbool.h>
```

```
00121 #include <unistd.h>
00122 #include <bcm2835.h>
00123 #include <stdio.h>
00124
00125
00126
00127 #include "ade7880_registers.h"
00128 #include "rpi_hwio.h"
00129 #include "spi_ade7880_protocol.h"
00130 #include "srv_cmd_handler.h"
00131
00132
00133
00134 int8_t      save_to_file(uint16_t rpi_address,measured_data_t data, const char *
      fileName,uint8_t clean) ;
00135 int8_t      wait_new_conversion(void);
00136 int16_t     config_recheck(const char * config_file, const char * format, ...);
00137 void        reading_loop();
00138
00139
00140 #endif
00141
```

8.7 rpi_hwio.c File Reference

File: rpi_hwio.c

`#include "../includes/includes.h"`
Include dependency graph for rpi_hwio.c:



Functions

- int8_t spi_init (uint8_t chip_select)

- int8_t rpi_gpio_init (void)

8.7.1 Detailed Description

File: rpi_hwio.c

Purpose:

Definition in file rpi_hwio.c.

8.8 rpi_hwio.c

```
00001
00002
00009 #include "../includes/includes.h"
```

```
00010
00015
00016
00037
00038 int8_t spi_init(uint8_t chip_select){
00039
00040     bcm2835_spi_begin();
00041     bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);
00042     bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);                      // Data are propagated on a falling
      edge
00043
00044     bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);      // 65536 = 262.144us = 3.814697260kHz
00045     bcm2835_spi_chipSelect(chip_select);
00046     bcm2835_spi_setChipSelectPolarity(chip_select, LOW);             // SS/HSA chip pin of ade7880 should be
      low for comminication
00047
00048 return 0;
00049 }
00051
00061
00062
00063 int8_t rpi_gpio_init(void){
00064
00065         bcm2835_gpio_fsel(PIN_SS,  BCM2835_GPIO_FSEL_OUTP);
00066         bcm2835_gpio_fsel(PIN_PM0,  BCM2835_GPIO_FSEL_OUTP);
00067         bcm2835_gpio_fsel(PIN_PM1,  BCM2835_GPIO_FSEL_OUTP);
00068         bcm2835_gpio_fsel(PIN_IRQ0, BCM2835_GPIO_FSEL_INPT);
00069         bcm2835_gpio_fsel(PIN_IRQ1, BCM2835_GPIO_FSEL_INPT);
00070
00071 return 0;
00072 }
00073
00075
00076
00077
00091
00092 int8_t config_cmd(uint32_t cmd,uint8_t arg,...){
00093
00094           va_list va;
00095           va_start(va,arg);
00096           uint8_t cc = 0;
00097           int8_t result = -1;
00098         switch(cmd)
00099         {
00100
00101                 case  ADE7880_SPI_PORT_ACTIVATE:
00102                 {
00110                     uint8_t ii;
00111                     bool PIN_SET = 0;
00112                     config2_reg_u config2_reg;
00113
00114
00115                     for(ii=0;ii<3/*TOGEL SS PIN 3 TIMES*/;ii++){
00116
00117                         if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      DUMMIY,0xFF,sizeof(uint8_t))!=0)
00118                             return -1;
00119
00120                         usleep(1);
00121                     }
00122
00123
00124                      uint8_t cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,DISABLE))== -1)&& (cc++ < 3));if(result==-1)return -1;
00125
00126                       config2_reg.reg_all=0;
00127
00128                      config2_reg.reg_all = (uint8_t)spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,CONFIG2,sizeof(config2_reg.reg_all));
00129
00130
00131                    //put it back, this will lock the spi port antill the next device reset
00132                      if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      CONFIG2,config2_reg.reg_all,sizeof(config2_reg.reg_all))!=0)
00133                             return -1;
00134
00135
00136                 }break;
00137
00138
00139
00140                 case DSP_RUN:
00141                 {
00142
00143                     if(arg != 1)
00144                      return -1;
00145
```

```
00146                     uint8_t prt_arg = va_arg(va,int);
00147                     uint16_t ii;
00148                     for(ii=0;ii<3;ii++){
00149                     if(prt_arg == ENABLE){
00150                     if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      RUN,0x0001,sizeof(uint16_t))!=0)
00151                     return -1;
00152                     }else{
00153                     if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      RUN,0x0000,sizeof(uint16_t))!=0)
00154                     return -1;
00155                     }
00156                     }
00157
00158
00159                 }break;
00160
00161                 case  ADE7880_RESET:
00162                 {
00163
00164
00165
00166                 }break;
00167
00168
00169
00170                 case  GAIN:
00171                 {
00172                     if(arg != 3)
00173                         return -1;
00174
00175                     gain_reg_u gain;
00176                     gain.reg_all = 0;
00177
00178                     gain.bits.PGA1 = (pga_et)va_arg(va,int);
00179                     gain.bits.PGA2 = (pga_et)va_arg(va,int);
00180                     gain.bits.PGA3 = (pga_et)va_arg(va,int);
00181
00182                     if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      GAIN,gain.reg_all,sizeof(gain.reg_all))!=0)
00183                     return -1;
00184
00185                 }break;
00186
00187
00188                 case  POWER_MODE:
00189                 {
00190                     if(arg != 1)
00191                       return -1;
00192
00193                     uint8_t mode = va_arg(va,int);
00194
00195                    if(ade7880_power_mode(mode)!=0)
00196                         return -1;
00197
00198                 }break;
00199
00200                 case VLEVEL:
00201                 {
00202                     if(arg !=1)
00203                       return -1;
00204
00205                     uint32_t vlevel_arg = va_arg(va,int);
00206
00207                     if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      VLEVEL,vlevel_arg,sizeof(uint32_t))!=0)
00208                         return -1;
00209                 }break;
00210
00211
00212                 case COMPMODE:
00213                 {
00214                         if(arg != 1)
00215                           return -1;
00216
00217                         uint16_t compmode_arg = va_arg(va,int);
00218
00219                         if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      COMPMODE,compmode_arg,sizeof(uint16_t))!=0)
00220                             return -1;
00221                 }break;
00222
00223
00224                 case VNOM:
00225                 {
00226                         if(arg != 1)
00227                           return -1;
```

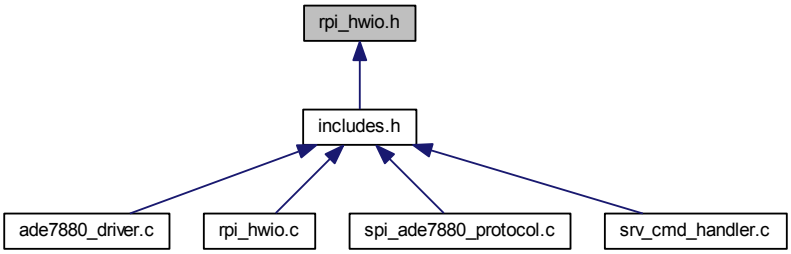
```
00228
00229                           uint32_t vnom_arg = va_arg(va,int);
00230
00231                           if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      VNOM,vnom_arg,sizeof(uint32_t))!=0)
00232                               return -1;
00233                   }break;
00234
00235
00236
00237
00238                   case  THRESHOLD:
00239                   {
00240                       if(arg != 3)
00241                         return -1;
00242
00243                        uint8_t wthr_arg = va_arg(va,int);
00244                        uint8_t varthr_arg = va_arg(va,int);
00245                        uint8_t vathr_arg = va_arg(va,int);
00246
00247
00248                       if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      WTHR,WTHR_DEFULT,sizeof(uint8_t))!=0)
00249                         return -1;
00250
00251
00252
00253                       if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      VARTHR,VARTHR_DEFULT,sizeof(uint8_t))!=0)
00254                         return -1;
00255
00256
00257                       if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      VATHR,VATHR_DEFULT,sizeof(uint8_t))!=0)
00258                          return -1;
00259
00260
00261                   }break;
00262
00263
00264
00265
00266                   //Enables/Disable data memory protection
00267                   case SET_RAM_WR_PROTECTION:
00268                   {
00269
00270                       if(arg != 1)
00271                        return -1;
00272
00273                       uint8_t prt_arg = va_arg(va,int);
00274
00275                          // spi_ram_protection(prt_arg,CHIP_ADDRESS1);
00276
00277                   }break;
00278
00279
00280
00281
00282
00283
00284
00285                   default:
00286                   {
00287                     // return -1;
00288                   }break;
00289
00290           }
00291
00292           va_end(va);
00293
00294 return 0;
00295 }
00296
00298
00308
00309 int8_t ade7880_power_mode(uint8_t mode){
00319            if(!bcm2835_init())
00320              return -1;
00321
00322           switch(mode)
00323           {
00324
00325                   case  PSM0:
00326                   {
00328                        bcm2835_gpio_write(PIN_PM0, HIGH);
00329                        bcm2835_gpio_write(PIN_PM1, LOW);
00330
```

```
00331                 }break;
00332
00333                  case  PSM1:
00334                 {
00336                       bcm2835_gpio_write(PIN_PM0, LOW);
00337                       bcm2835_gpio_write(PIN_PM1, LOW);
00338                 }break;
00339
00340                  case  PSM2:
00341                 {
00343                       bcm2835_gpio_write(PIN_PM0, LOW);
00344                       bcm2835_gpio_write(PIN_PM1, HIGH);
00345
00346                 }break;
00347
00348
00349                  case  PSM3:
00350                 {
00352                       bcm2835_gpio_write(PIN_PM0, HIGH);
00353                       bcm2835_gpio_write(PIN_PM1, HIGH);
00354
00355                 }break;
00356
00357                 default:
00358                 {
00359                    return -1;
00360                 }break;
00361
00362         }
00363
00364      //we may need to check the state of the pin's here to verify
00365
00366 return 0;
00367 }
00369
00378
00379
00380 int8_t ade7880_config_reg_default(void){
00381
00398     uint16_t cc=0;
00399     int8_t result=0;
00400     compmode_reg_u compmode_reg;
00401
00402
00403             //SET gain registers
00404             if(config_cmd(GAIN,3,GAIN_1,GAIN_1,GAIN_1) != 0)return -1;
00405
00406             if(config_cmd(THRESHOLD,3,WTHR_DEFULT,
      VARTHR_DEFULT,VATHR_DEFULT) != 0)return -1;
00407
00408             if(spi_write(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      CONFIG3,CONFIG3_DEFULT,sizeof(uint8_t))!=0)return -1;
00409
00410             #if _VNOMAEN
00411             compmode_reg.reg_all = 0x0001;
00412             compmode_reg.bits.VNOMAEN    = 0;
00413             compmode_reg.bits.VNOMBEN    = 0;
00414             compmode_reg.bits.VNOMCEN    = 0;
00415
00416
00417             if(config_cmd(COMPMODE,1,compmode_reg.reg_all) != 0)
00418             return -1;
00419             #endif
00420
00421
00422             while(++cc<3)
00423             if(config_cmd(VLEVEL,1,VLEVEL_DEFULT) != 0)
00424             return -1;
00425
00426
00427              result=0;cc = 0;while(((result = config_cmd(
      SET_RAM_WR_PROTECTION,1,ENABLE))== -1)&& (cc++ < 3));if(result==-1)return -1;
00428              result=0;cc = 0;while(((result = config_cmd(DSP_RUN,1,
      ENABLE))== -1)&& (cc++ < 3));if(result==-1)return -1;
00429
00430
00431
00432
00433         return 0;
00434  }
00437
00438
00439
```

8.9 rpi_hwio.h File Reference

File: rpi_hwio.h

This graph shows which files directly or indirectly include this file:



Macros

- #define PIN_SS RPI_GPIO_P1_24

  Rpi chip select pin.
- #define PIN_SS2 RPI_GPIO_P1_26

  Rpi second chip select pin.
- #define PIN_PM0 RPI_GPIO_P1_11

  ADE7880 PM0 connection pin.
- #define PIN_PM1 RPI_GPIO_P1_13

  ADE7880 PM1 connection pin.
- #define PIN_IRQ0 RPI_GPIO_P1_16

  ADE7880 IRQ0 connection pin.
- #define PIN_IRQ1 RPI_GPIO_P1_18

  ADE7880 IRQ1 connection pin.

Functions

- int8_t rpi_gpio_init ()
- int8_t ade7880_power_mode (uint8_t mode)
- int8_t ade7880_config_reg_default ()
- int8_t config_cmd (uint32_t cmd, uint8_t arg,...)

8.9.1 Detailed Description

File: rpi_hwio.h

Purpose:

Definition in file rpi_hwio.h.

8.9.2 Macro Definition Documentation

8.9.2.1 #define PIN_IRQ0 RPI_GPIO_P1_16

ADE7880 IRQ0 connection pin.

Definition at line 15 of file rpi_hwio.h.

Referenced by rpi_gpio_init().

8.9.2.2 #define PIN_IRQ1 RPI_GPIO_P1_18

ADE7880 IRQ1 connection pin.

Definition at line 16 of file rpi_hwio.h.

Referenced by rpi_gpio_init().

8.9.2.3 #define PIN_PM0 RPI_GPIO_P1_11

ADE7880 PM0 connection pin.

Definition at line 13 of file rpi_hwio.h.

Referenced by rpi_gpio_init().

8.9.2.4 #define PIN_PM1 RPI_GPIO_P1_13

ADE7880 PM1 connection pin.

Definition at line 14 of file rpi_hwio.h.

Referenced by rpi_gpio_init().

8.9.2.5 #define PIN_SS RPI_GPIO_P1_24

Rpi chip select pin.

Definition at line 10 of file rpi_hwio.h.

Referenced by main(), rpi_gpio_init(), and spi_rd_wr().

8.9.2.6 #define PIN_SS2 RPI_GPIO_P1_26

Rpi second chip select pin.

Definition at line 11 of file rpi_hwio.h.

8.10 rpi_hwio.h

```
00001
00007 #ifndef __RPI_GPIO_H
00008 #define __RPI_GPIO_H
00009
00010   #define PIN_SS   RPI_GPIO_P1_24
00011   #define PIN_SS2  RPI_GPIO_P1_26
00013   #define PIN_PM0  RPI_GPIO_P1_11
00014   #define PIN_PM1  RPI_GPIO_P1_13
00015   #define PIN_IRQ0 RPI_GPIO_P1_16
00016   #define PIN_IRQ1 RPI_GPIO_P1_18
00019   int8_t  rpi_gpio_init();
00020   int8_t  ade7880_power_mode(uint8_t mode);
00021   int8_t  ade7880_config_reg_default();
```

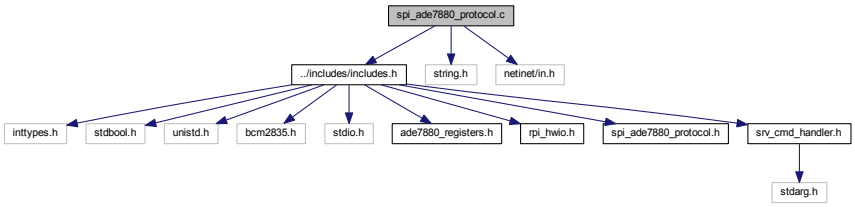
```
00022   int8_t  config_cmd(uint32_t cmd,uint8_t arg,...);
00023
00024 #endif
00025
```

8.11 spi_ade7880_protocol.c File Reference

File: spi_ade7880_protocol.c

```
#include "../includes/includes.h"
#include <string.h>
#include <netinet/in.h>
```

Include dependency graph for spi_ade7880_protocol.c:



Functions

- int8_t spi_rd_wr (uint8_t chip_select, uint8_t *tx_buffer, uint16_t len)
- uint32_t spi_read (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, uint8_t reg_len)
- int8_t spi_write (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, int32_t value, uint8_t reg_len)
- int8_t spi_ram_protection (uint8_t cmd, uint8_t chip_address)

8.11.1 Detailed Description

File: spi_ade7880_protocol.c

Purpose:

Definition in file spi_ade7880_protocol.c.

8.12 spi_ade7880_protocol.c

```
00001
00008 #include "../includes/includes.h"
00009 #include <string.h>
00010 #include <netinet/in.h>
00011
00012
00013 #ifdef DEBUG
00014 uint8_t debug_print = ENABLE;
```

```
00015 #endif
00016
00021
00022
00086 #ifdef DEBUG
00087
00093 void spi_enable_msg_debug_print(uint8_t cmd){
00094     debug_print =(cmd==DISABLE)?(0x08|DISABLE):ENABLE; //bit 4 is set to indecate
       external setting
00095 }
00096 #endif
00097
00098
00112
00113
00114 int8_t  spi_rd_wr(uint8_t chip_select,uint8_t * tx_buffer,uint16_t len)
00115 {
00116
00117
00118        if(!spi_init(chip_select)){
00119
00120           bcm2835_spi_transfern(tx_buffer,sizeof(tx_buffer));
00121           return 0;
00122        }else
00123           return -1;
00124
00125        bcm2835_spi_end();
00126
00127        bcm2835_gpio_write(PIN_SS, HIGH);
00128 }
00130
00146
00147 uint32_t spi_read(uint8_t chip_select,uint8_t chip_address,uint16_t
      target_register,uint8_t reg_len){
00148
00149   uint32_t result= 0;
00150
00151     ade7880_read_tx_buff_ut tx_buff;
00152
00153     tx_buff.msg_fields.address_byte.address_all = chip_address;
00154     tx_buff.msg_fields.address_byte.bits.RD_WR = ADE7880_RD;
00155     tx_buff.msg_fields.target_register = htons(target_register);
00156     tx_buff.msg_fields.value.reg32 = DUMMY_MSG;
00157
00158     #ifdef DEBUG
00159      if((debug_print&0x01)){
00160        printf("\nMSG SENT  : RD\n");
00161
00162        printf("        Chip Address %02X\n",tx_buff.msg_fields.address_byte.
      address_all);
00163        printf("        Target Register %04X\n",ntohs(tx_buff.msg_fields.target_register));
00164        printf("        DUMMY %08X \n",tx_buff.msg_fields.value.reg32);
00165        }
00166     #endif
00167
00168     bcm2835_spi_transfern(tx_buff.msg_all,RD_MSG_LENGTH + reg_len);
00169
00170
00171
00172
00173
00174
00175     result =  (reg_len == sizeof(uint32_t))?((uint32_t)ntohl(tx_buff.msg_fields.value.
      reg32))
00176               :((reg_len == sizeof(uint16_t))?((uint32_t)ntohs(tx_buff.msg_fields.value.
      reg16))
00177               :((reg_len == sizeof(uint8_t))?((uint32_t)tx_buff.msg_fields.value.
      reg8):0x0F000000/*ERROR*/));
00178
00179
00180     #ifdef DEBUG
00181      if(debug_print&0x01 || 1){
00182        printf("\nMSG REPLAY : RD\n");
00183        printf("        REPLAY :%X\n",result);
00184      }
00185     #endif
00186
00187
00188
00189
00190     return result;
00191
00192 }
00194
00195
00196
00220
```

```
00221 int8_t spi_write(uint8_t chip_select,uint8_t chip_address,uint16_t
      target_register,int32_t value,uint8_t reg_len){
00222
00223
00224     int8_t result = 0;
00225     ade7880_write_tx_buff_ut  tx_buff;
00226
00227      tx_buff.msg_fields.address_byte.address_all = chip_address;
00228      tx_buff.msg_fields.address_byte.bits.RD_WR =
      ADE7880_WR;
00229      tx_buff.msg_fields.target_register = htons(target_register);
00230
00231     if(reg_len == sizeof(uint8_t))
00232       tx_buff.msg_fields.value.reg8  = (uint8_t)value;
00233     else if(reg_len == sizeof(uint16_t))
00234       tx_buff.msg_fields.value.reg16 = htons((uint16_t)value);
00235     else if(reg_len == sizeof(uint32_t))
00236       tx_buff.msg_fields.value.reg32 = htonl((uint32_t)value);
00237
00238     #ifdef DEBUG
00239       if(debug_print&0x01){
00240        printf("\nMSG SENT: WR\n");
00241        printf("       Chip Address %02X\n",
00242        tx_buff.msg_fields.address_byte.address_all);
00243        printf("       Target Register %04X\n",
00244        ntohs(tx_buff.msg_fields.target_register));
00245        printf("       Value sent  %08X\n",
00246        ntohl(tx_buff.msg_fields.value.reg32));
00247        }
00248     #endif
00249
00250     bcm2835_spi_transfern(tx_buff.msg_all,WR_MSG_LENGTH + reg_len);
00251
00252
00253
00254     uint8_t  result8  =0;
00255     uint16_t result16 =0;
00256     int32_t result32 =0;
00257
00258      #ifdef DEBUG
00259      if(!(debug_print&0x80))
00260      debug_print = DISABLE;
00261        printf("\nvarifying last operation  ...\n");
00262      #endif
00263      if((result8 =(uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_OP,sizeof(uint16_t))))
00264      !=0xCA /*LAST_OP!=WR*/){
00265
00266      #ifdef DEBUG
00267        printf("\n                                                  <---  WR failure : LAST_OP \n");
00268        printf("        LAST_OP value  :%02X\n",result8);
00269      #endif
00270      result = -1;
00271      }else{
00272       #ifdef DEBUG
00273        printf("        LAST_OP value  :%02X\n",result8);
00274      #endif
00275      }
00276
00277      #ifdef DEBUG
00278        printf("\nvarifying last accessed register  ...\n");
00279      #endif
00280      if((result16 =(uint16_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_ADD,sizeof(uint16_t))))!=target_register){
00281
00282      #ifdef DEBUG
00283        printf("\n                                                  <---  WR failure : LAST_ADD\n");
00284        printf("        Target Register :%04X\n", target_register);
00285        printf("        LAST_ADD value  :%04X\n",result16);
00286      #endif
00287      result = -1;
00288      }else{
00289       #ifdef DEBUG
00290         printf("       LAST_ADD value  :%04X\n",result16);
00291       #endif
00292      }
00293
00294
00295      #ifdef DEBUG
00296        printf("\nvarifying last accessed register  value:\n");
00297      #endif
00298     if(reg_len == sizeof(uint8_t)){
00299         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA8,sizeof(uint8_t))&0x000000FF);
00300     }else
00301     if(reg_len == sizeof(uint16_t)){
00302         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
```

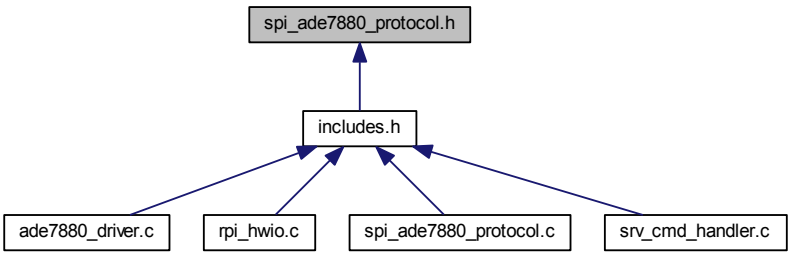
```
      LAST_RWDATA16,sizeof(uint16_t))&0x0000FFFF);
00303     }else
00304     if(reg_len == sizeof(uint32_t)){
00305         result32 = (int32_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA32,sizeof(uint32_t)));
00306     }
00307     //if
00308
00309      if(result32!=value)
00310          result = -1;
00311      #ifdef DEBUG
00312        if(result32!=value)
00313        printf("\n                                                  <---  WR failure : LAST_RWDATA\n");
00314        printf("       Sent Value        :%08X\n", value);
00315        printf("       LAST_RWDATA value :%08X\n",result32);
00316       if(!(debug_print&0x80))
00317        debug_print = ENABLE;
00318      #endif
00319
00320
00321     return result;
00322 }
00323
00325
00326
00327
00342
00343 int8_t spi_ram_protection(uint8_t cmd,uint8_t chip_address){
00344
00345
00346     int8_t result = 0;
00347
00348     uint8_t value = (cmd==ENABLE)?0x80:0x00;
00349
00350      ade7880_ram_lock_msg_ut tx_buff_1st_msg,tx_buff_2nd_msg;
00351
00352      tx_buff_1st_msg.msg_fields.address_byte.address_all = chip_address;
00353      tx_buff_1st_msg.msg_fields.address_byte.bits.RD_WR =
      ADE7880_WR;
00354      tx_buff_1st_msg.msg_fields.target_register = htons(0xE7FE);
00355      tx_buff_1st_msg.msg_fields.value = 0xAD;
00356
00357
00358      tx_buff_2nd_msg.msg_fields.address_byte.address_all = chip_address;
00359      tx_buff_2nd_msg.msg_fields.address_byte.bits.RD_WR =
      ADE7880_WR;
00360      tx_buff_2nd_msg.msg_fields.target_register = htons(0xE7E3);
00361      tx_buff_2nd_msg.msg_fields.value = value;
00362
00363
00364     #ifdef DEBUG
00365        if(debug_print&0x01){
00366        printf("\nMSG SENT: WR\n");
00367        printf("       Chip Address %02X\n",
00368        tx_buff_1st_msg.msg_fields.address_byte.address_all);
00369        printf("       1st Target Register %04X\n",
00370        ntohs(tx_buff_1st_msg.msg_fields.target_register)
00371        );
00372        printf("       Value sent 1st  %1X\n",
00373        ntohl(tx_buff_1st_msg.msg_fields.value));
00374
00375        printf("       2nd Target Register %04X\n",
00376        ntohs(tx_buff_2nd_msg.msg_fields.target_register)
00377        );
00378        printf("       Value sent  %01X\n",
00379        ntohl(tx_buff_2nd_msg.msg_fields.value));
00380        }
00381     #endif
00382
00383
00384     bcm2835_spi_transfern(tx_buff_1st_msg.msg_all,RAM_LOCK_MSG_LENGTH);
00385     bcm2835_spi_transfern(tx_buff_2nd_msg.msg_all,RAM_LOCK_MSG_LENGTH);
00386
00387
00388     uint8_t  result8  =0;
00389     uint16_t result16 =0;
00390
00391      #ifdef DEBUG
00392      if(!(debug_print&0x80))
00393      debug_print = DISABLE;
00394        printf("\nvarifying last operation  ...\n");
00395      #endif
00396      if((result8 =(uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_OP,sizeof(uint16_t))))
00397      !=0xCA /*LAST_OP!=WR*/){
00398
00399      #ifdef DEBUG
```

```
00400         printf("\n                                                  <---  WR failure : LAST_OP \n");
00401         printf("         LAST_OP value  :%02X\n",result8);
00402       #endif
00403       result = -1;
00404       }else{
00405        #ifdef DEBUG
00406         printf("         LAST_OP value  :%02X\n",result8);
00407       #endif
00408       }
00409
00410       #ifdef DEBUG
00411         printf("\nvarifying last accessed register  ...\n");
00412       #endif
00413       if((result16 =(uint16_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_ADD,sizeof(uint16_t))))!=0xE7E3){
00414
00415       #ifdef DEBUG
00416         printf("\n                                                  <---  WR failure : LAST_ADD\n");
00417         printf("         Target Register :%04X\n", 0xE7E3);
00418         printf("         LAST_ADD value  :%04X\n",result16);
00419       #endif
00420       result = -1;
00421       }else{
00422        #ifdef DEBUG
00423          printf("       LAST_ADD value  :%04X\n",result16);
00424        #endif
00425       }
00426
00427
00428       #ifdef DEBUG
00429         printf("\nvarifying last accessed register  value:\n");
00430       #endif
00431
00432       result8 = (uint8_t)(spi_read(BCM2835_SPI_CS0,CHIP_ADDRESS1,
      LAST_RWDATA8,sizeof(uint8_t))&0x000000FF);
00433
00434
00435       if(result8!=value)
00436           result = -1;
00437       #ifdef DEBUG
00438         if(result8!=value)
00439         printf("\n                                                  <---  WR failure : LAST_RWDATA\n");
00440         printf("       Sent Value        :%01X\n", value);
00441         printf("       LAST_RWDATA value :%01X\n",result8);
00442
00443         if(!(debug_print&0x80))
00444             debug_print = ENABLE;
00445       #endif
00446
00447
00448
00449      return result;
00450 }
00451
00454
00455
00456
00457
00458
00459
00460
```

8.13 spi_ade7880_protocol.h File Reference

File: spi_ade7880_protocol.h

This graph shows which files directly or indirectly include this file:



Data Structures

- union address_byte_ut

  address byte of msg structure RPi to ADE7880
- union value_ut
- union ade7880_read32_tx_buff_ut

  ADE7880 read cmd msg structure.
- union ade7880_read16_tx_buff_ut

  ADE7880 read cmd msg structure.
- union ade7880_read8_tx_buff_ut

  ADE7880 read cmd msg structure.
- union ade7880_write32_tx_buff_ut

  ADE7880 write cmd msg structure.
- union ade7880_write16_tx_buff_ut

  ADE7880 write cmd msg structure.
- union ade7880_write8_tx_buff_ut

  ADE7880 write cmd msg structure.
- union ade7880_read_tx_buff_ut

  ADE7880 write cmd msg structure.
- union ade7880_write_tx_buff_ut

  ADE7880 write cmd msg structure.
- union ade7880_ram_lock_msg_ut

  ADE7880 write cmd msg structure.

Macros

- #define ADE7880_RD 1
- #define ADE7880_WR 0
- #define DUMMY_MSG 0x00000000
- #define RD_MSG_LENGTH 3
- #define WR_MSG_LENGTH 3
- #define RAM_LOCK_MSG_LENGTH 4
- #define REG_LENGTH(_x) (sizeof(_x))

Functions

- uint16_t target_register __attribute__ ((packed))
- uint32_t spi_read (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, uint8_t reg_len)
- int8_t spi_write (uint8_t chip_select, uint8_t chip_address, uint16_t target_register, int32_t value, uint8_t reg_len)

Variables

- address_byte_ut address_byte
- uint8_t rc_arg
- uint16_t target_register
- union {
  uint8_t reg8
  } value
- uint16_t reg16
- uint32_t reg32

8.13.1 Detailed Description

File: spi_ade7880_protocol.h

Purpose:

Definition in file spi_ade7880_protocol.h.

8.13.2 Macro Definition Documentation

8.13.2.1 #define ADE7880_RD 1

Definition at line 12 of file spi_ade7880_protocol.h.

Referenced by spi_read().

8.13.2.2 #define ADE7880_WR 0

Definition at line 13 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection(), and spi_write().

8.13.2.3 #define DUMMY_MSG 0x00000000

Definition at line 14 of file spi_ade7880_protocol.h.

Referenced by save_to_file(), and spi_read().

8.13.2.4 #define RAM_LOCK_MSG_LENGTH 4

Definition at line 17 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection().

8.13.2.5 #define RD_MSG_LENGTH 3

Definition at line 15 of file spi_ade7880_protocol.h.

Referenced by spi_read().

8.13.2.6 #define REG_LENGTH(_x) (sizeof(_x))

Definition at line 18 of file spi_ade7880_protocol.h.

8.13.2.7 #define WR_MSG_LENGTH 3

Definition at line 16 of file spi_ade7880_protocol.h.

Referenced by spi_write().

8.13.3 Function Documentation

8.13.3.1 uint16_t target_register __attribute__ ((packed))

8.13.4 Variable Documentation

8.13.4.1 address_byte_ut address_byte

Definition at line 49 of file spi_ade7880_protocol.h.

8.13.4.2 uint8_t rc_arg

Definition at line 81 of file spi_ade7880_protocol.h.

8.13.4.3 uint16_t reg16

Definition at line 154 of file spi_ade7880_protocol.h.

8.13.4.4 uint32_t reg32

Definition at line 155 of file spi_ade7880_protocol.h.

8.13.4.5 uint8_t reg8

Definition at line 136 of file spi_ade7880_protocol.h.

8.13.4.6 uint16_t target_register

Definition at line 134 of file spi_ade7880_protocol.h.

8.13.4.7 uint8_t value

Definition at line 195 of file spi_ade7880_protocol.h.

Referenced by spi_ram_protection(), and spi_write().

8.14 spi_ade7880_protocol.h

```
00001
00006 #ifndef __SRV_ADE7880_PROTOCOL_H
00007 #define __SRV_ADE7880_PROTOCOL_H
00008
00009
00010
00011
00012 #define ADE7880_RD                  1
00013 #define ADE7880_WR                  0
00014 #define DUMMY_MSG                   0x00000000
00015 #define RD_MSG_LENGTH               3
00016 #define WR_MSG_LENGTH               3
00017 #define RAM_LOCK_MSG_LENGTH         4
00018 #define REG_LENGTH(_x)              (sizeof(_x))
00019
00021 typedef union
00022 {
00023     uint8_t address_all;
00024     struct
00025     {
00026         uint8_t RD_WR:1;
00027         uint8_t chip_address:7;
00028     }bits;
00029
00030 }address_byte_ut;
00031
00032 typedef union{
00033 uint8_t         reg8;
00034 uint16_t        reg16;
00035 uint32_t        reg32;
00036 }value_ut;
00037
00039 typedef union
00040 {
00041     uint8_t msg_all[RD_MSG_LENGTH + sizeof(uint32_t)]
    __attribute__ ((aligned));
00042
00043     struct
00044     {
00045     address_byte_ut address_byte ;
00046     uint16_t        target_register __attribute__ ((packed));
00047     uint32_t        rc_arg __attribute__ ((packed));
00048     }msg_fields  __attribute__ ((aligned));
00049
00050 }ade7880_read32_tx_buff_ut;
00051
00052
00054 typedef union
00055 {
00056     uint8_t msg_all[RD_MSG_LENGTH + sizeof(uint16_t)]
    __attribute__ ((aligned));
00057
00058     struct
00059     {
00060     address_byte_ut address_byte;
00061     uint16_t        target_register __attribute__ ((packed));
00062     uint16_t        rc_arg __attribute__ ((packed));
00063     }msg_fields  __attribute__ ((aligned));
00064
00065 }ade7880_read16_tx_buff_ut;
00066
00067
00069 typedef union
00070 {
00071     uint8_t msg_all[RD_MSG_LENGTH + sizeof(uint8_t)]  __attribute__ ((aligned));
00072
00073     struct
00074     {
00075     address_byte_ut address_byte;
00076     uint16_t        target_register __attribute__ ((packed));
00077     uint8_t         rc_arg;
00078     }msg_fields  __attribute__ ((aligned));
00079
00080 }ade7880_read8_tx_buff_ut;
00081
00082
00083
00085 typedef union
00086 {
00087     uint8_t msg_all[WR_MSG_LENGTH + sizeof(uint32_t)]
    __attribute__ ((aligned));
00088
00089     struct
00090     {
```

```
00091     address_byte_ut address_byte;
00092     uint16_t        target_register __attribute__ ((packed));
00093     uint32_t        value __attribute__ ((packed));
00094     }msg_fields  __attribute__ ((aligned));
00095
00096 }ade7880_write32_tx_buff_ut;
00097
00098
00099
00101 typedef union
00102 {
00103     uint8_t msg_all[WR_MSG_LENGTH + sizeof(uint16_t)]
      __attribute__ ((aligned));
00104
00105     struct
00106     {
00107     address_byte_ut address_byte;
00108     uint16_t        target_register __attribute__ ((packed));
00109     uint16_t        value __attribute__ ((packed));
00110     }msg_fields  __attribute__ ((aligned));
00111
00112 }ade7880_write16_tx_buff_ut;
00113
00114
00115
00116
00117
00119 typedef union
00120 {
00121     uint8_t msg_all[WR_MSG_LENGTH + sizeof(uint32_t)]
      __attribute__ ((aligned));
00122
00123     struct
00124     {
00125     address_byte_ut address_byte;
00126     uint16_t        target_register;
00127         union{
00128         uint8_t         reg8;
00129         uint16_t        reg16 __attribute__ ((packed));
00130         uint32_t        reg32 __attribute__ ((packed));
00131         }value;
00132     }msg_fields  __attribute__ ((aligned));
00133
00134 }ade7880_write8_tx_buff_ut;
00135
00136
00137
00138
00140 typedef union
00141 {
00142     uint8_t msg_all[WR_MSG_LENGTH + sizeof(uint32_t)]
      __attribute__ ((aligned));
00143
00144     struct
00145     {
00146     address_byte_ut address_byte;
00147     uint16_t        target_register __attribute__ ((packed));
00148     union{
00149         uint8_t         reg8;
00150         uint16_t        reg16;
00151         uint32_t        reg32;
00152     }value __attribute__ ((packed));
00153
00154     }msg_fields  __attribute__ ((aligned));
00155
00156 }ade7880_read_tx_buff_ut;
00157
00158
00159
00160
00162 typedef union
00163 {
00164     uint8_t msg_all[WR_MSG_LENGTH + sizeof(uint32_t)]
      __attribute__ ((aligned));
00165
00166     struct
00167     {
00168     address_byte_ut address_byte;
00169     uint16_t        target_register __attribute__ ((packed));
00170     union{
00171         uint8_t         reg8;
00172         uint16_t        reg16;
00173         uint32_t        reg32;
00174     }value __attribute__ ((packed));
00175
00176     }msg_fields  __attribute__ ((aligned));
00177
```

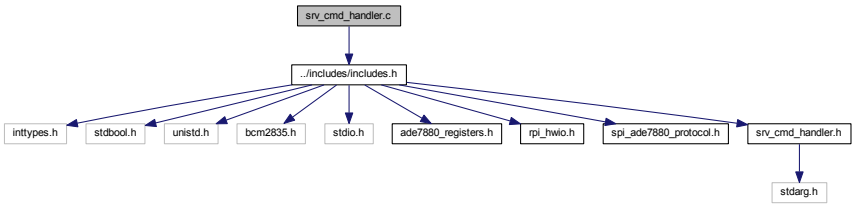
```
00178 }ade7880_write_tx_buff_ut;
00179
00180
00181
00183 typedef union
00184 {
00185     uint8_t msg_all[sizeof(uint16_t)+ 2*sizeof(uint8_t)]  __attribute__ ((aligned));
00186
00187     struct
00188     {
00189     address_byte_ut address_byte;
00190     uint16_t        target_register __attribute__ ((packed));
00191     uint8_t         value;
00192     }msg_fields  __attribute__ ((aligned));
00193
00194 }ade7880_ram_lock_msg_ut;
00195
00196
00197
00198
00199 uint32_t spi_read(uint8_t chip_select,uint8_t chip_address,uint16_t target_register,uint8_t reg_len
    );
00200 int8_t   spi_write(uint8_t chip_select,uint8_t chip_address,uint16_t target_register,int32_t
    value,uint8_t reg_len);
00201
00202 #ifdef DEBUG
00203 void spi_enable_msg_debug_print(uint8_t cmd);
00204 #endif
00205
00206
00207
00208
00209
00210
00211
00212 #endif
00213
00214
```

8.15 srv_cmd_handler.c File Reference

File: srv_cmd_handler.c

```
#include "../includes/includes.h"
```

Include dependency graph for srv_cmd_handler.c:



Functions

- float measure (uint8_t cmd, uint8_t channel, float samples)
- uint16_t hex2val (uint8_t cc)
- uint16_t make16 (uint8_t *buf, uint16_t idx)

- uint8_t make8 (uint8_t *buf, uint16_t idx)

8.15.1 Detailed Description

File: srv_cmd_handler.c

Purpose:

Definition in file srv_cmd_handler.c.

8.16 srv_cmd_handler.c

```
00001
00006 #include "../includes/includes.h"
00007
00008
00013
00014
00039
00040 float measure(uint8_t cmd,uint8_t channel,float samples){
00041
00042
00043     float measured_val=0;
00044
00045
00046          switch(cmd)
00047         {
00048
00049                 case  PHASE_VRMS:
00050                 {
00051
00052                     uint16_t ii;
00053                     uint16_t target_reg =  (channel==PHASE_A )?AVRMS
00054                                           :(channel==PHASE_B )?BVRMS
00055                                           :(channel==PHASE_C )?CVRMS
00056                                           :0;
00057
00058                     if(target_reg==0)return -1;
00059                     for(ii=0;ii<samples;ii++){
00060                     if(wait_new_conversion()==-1)return -1;
00061                     measured_val += (uint32_t)(spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t))&0xFFFFFF)*
      TRANSFORMER_RATIO;
00062                     if(ii%50 == 0)printf("#\n");else printf("#");
00063                     }
00064                     measured_val /=samples;
00065
00066
00067                     return measured_val/VLSB_CONST;
00068
00069                 }break;
00070
00071                 case  PHASE_IRMS:
00072                 {
00073                     uint16_t ii;
00074                     uint16_t target_reg =  (channel==PHASE_A )?AIRMS
00075                                           :(channel==PHASE_B )?BIRMS
00076                                           :(channel==PHASE_C )?CIRMS
00077                                           :0;
00078                     if(target_reg==0)return -1;
00079                     for(ii=0;ii<samples;ii++){
00080                     if(wait_new_conversion()==-1)return -1;
00081                     measured_val += spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00082                     if(ii%50 == 0)printf("#\n");else printf("#");
00083
00084                     }
00085                     measured_val /=samples;
00086
00087                     return measured_val/ILSB_CONST;
00088
00089                 }break;
00090
00091
00092                 case  PHASE_ACTIVE_WH:/*considers also harmoics , trouble? then we change it
       to fundamental*/
00093                 {
```

```
00094
00095                     uint16_t target_reg =  (channel==PHASE_A )?AWATTHR
00096                                           :(channel==PHASE_B )?BWATTHR
00097                                           :(channel==PHASE_C )?CWATTHR
00098                                           :0;
00099
00100                     measured_val = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00101                     printf("#");
00102                     return measured_val/WHLSB_CONST;
00103
00104                 }break;
00105
00106                 case  TOTAL_ACTIVE_WH:/*considers also harmoics */
00107                 {
00108
00109                     uint16_t target_reg =  (channel==PHASE_A )?AWATTHR
00110                                           :(channel==PHASE_B )?BWATTHR
00111                                           :(channel==PHASE_C )?CWATTHR
00112                                           :0;
00113
00114                      return  measure(PHASE_ACTIVE_WH,
      PHASE_A,1)
00115                             +measure(PHASE_ACTIVE_WH,
      PHASE_B,1)
00116                             +measure(PHASE_ACTIVE_WH,
      PHASE_C,1);
00117
00118
00119                 }break;
00120
00121                 case  PHASE_ACTIVE_POWER:
00122                 {
00123                     uint16_t target_reg =    (channel==PHASE_A )?AWATT
00124                                             :(channel==PHASE_B )?BWATT
00125                                             :(channel==PHASE_C )?CWATT
00126                                             :0;
00127
00128                     measured_val = spi_read(BCM2835_SPI_CS0,
      CHIP_ADDRESS1,target_reg,sizeof(uint32_t));
00129                     printf("#");
00130                     return measured_val/WATTLSB_CONST;
00131
00132                 }break;
00133
00134                 case  TOTAL_ACTIVE_POWER:
00135                 {
00136
00137
00138                      return measure(PHASE_ACTIVE_POWER,
      PHASE_A,1)
00139                             +measure(PHASE_ACTIVE_POWER,
      PHASE_B,1);
00140                             +measure(PHASE_ACTIVE_POWER,
      PHASE_C,1);
00141
00142                 }break;
00143
00144                 default:
00145                 {
00146                   // return -1;
00147                 }break;
00148
00149         }
00150 }
00152
00153
00164
00165 uint16_t  hex2val(uint8_t cc)
00166 {
00167     return  (uint16_t)(((cc >= '0' ) && (cc <= '9' )) ? (cc - '0') : (cc - 'A' + 10) );
00168
00169 }
00171
00184
00185 uint16_t  make16(uint8_t* buf,uint16_t idx)
00186 {
00187     return ((hex2val(buf[idx+0])<<12)|(hex2val(buf[idx+1])<<8)|(
      hex2val(buf[idx+2])<<4)|(hex2val(buf[idx+3])<<0));
00188
00189 }
00191
00205
00206
00207 uint8_t  make8(uint8_t* buf,uint16_t idx)
00208 {
00209     return (uint8_t)((hex2val(buf[idx+0])<<4)|(hex2val(buf[idx+1]))<<0);
```

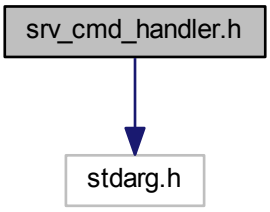
```
00210 }
00211
00213
```

8.17 srv_cmd_handler.h File Reference

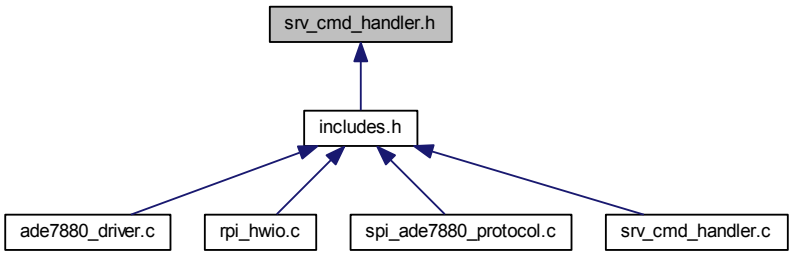
File: rpi_gpio.h

```
#include <stdarg.h>
```

Include dependency graph for srv_cmd_handler.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ADE7880_SPI_PORT_ACTIVATE 0x01

  internal use config cmd macros, make sure the values will not
- #define ADE7880_RESET 0x02
- #define POWER_MODE 0x03
- #define THRESHOLD 0x04
- #define SET_RAM_WR_PROTECTION 0x05
- #define DSP_RUN 0x06
- #define PHASE_A 0x0A

  internal use measurment read cmd macros
- #define PHASE_B 0x0B
- #define PHASE_C 0x0C

- #define LINE 0x00
- #define PHASE_VRMS 0x01
- #define PHASE_IRMS 0x02
- #define PHASE_ACTIVE_WH 0x03
- #define TOTAL_ACTIVE_WH 0x04
- #define PHASE_ACTIVE_POWER 0x05
- #define TOTAL_ACTIVE_POWER 0x06

Functions

- float measure (uint8_t cmd, uint8_t channel, float samples)
- uint16_t hex2val (uint8_t cc)
- uint16_t make16 (uint8_t *buf, uint16_t idx)
- uint8_t make8 (uint8_t *buf, uint16_t idx)
- uint32_t mask24 (uint32_t val)

8.17.1 Detailed Description

File: rpi_gpio.h

Purpose:

Definition in file srv_cmd_handler.h.

8.17.2 Macro Definition Documentation

8.17.2.1 #define ADE7880_RESET 0x02

Definition at line 13 of file srv_cmd_handler.h.

8.17.2.2 #define ADE7880_SPI_PORT_ACTIVATE 0x01

internal use config cmd macros, make sure the values will not

Definition at line 12 of file srv_cmd_handler.h.

8.17.2.3 #define DSP_RUN 0x06

Definition at line 17 of file srv_cmd_handler.h.

8.17.2.4 #define LINE 0x00

Definition at line 24 of file srv_cmd_handler.h.

8.17.2.5 #define PHASE_A 0x0A

internal use measurment read cmd macros

Definition at line 21 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.6 #define PHASE_ACTIVE_POWER 0x05

Definition at line 30 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.7 #define PHASE_ACTIVE_WH 0x03

Definition at line 28 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.8 #define PHASE_B 0x0B

Definition at line 22 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.9 #define PHASE_C 0x0C

Definition at line 23 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.10 #define PHASE_IRMS 0x02

Definition at line 27 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.11 #define PHASE_VRMS 0x01

Definition at line 26 of file srv_cmd_handler.h.

Referenced by measure(), and reading_loop().

8.17.2.12 #define POWER_MODE 0x03

Definition at line 14 of file srv_cmd_handler.h.

8.17.2.13 #define SET_RAM_WR_PROTECTION 0x05

Definition at line 16 of file srv_cmd_handler.h.

Referenced by main(), and wait_new_conversion().

8.17.2.14 #define THRESHOLD 0x04

Definition at line 15 of file srv_cmd_handler.h.

8.17.2.15 #define TOTAL_ACTIVE_POWER 0x06

Definition at line 31 of file srv_cmd_handler.h.

Referenced by measure().

8.17.2.16 #define TOTAL_ACTIVE_WH 0x04

Definition at line 29 of file srv_cmd_handler.h.

Referenced by measure().

8.17.3 Function Documentation

8.17.3.1 uint32_t mask24 (uint32_t *val*)

8.18 srv_cmd_handler.h

```
00001
00007 #ifndef __SRV_CMD_HANDLER_H
00008 #define __SRV_CMD_HANDLER_H
00009
00010
00012 #define ADE7880_SPI_PORT_ACTIVATE 0x01
00013 #define ADE7880_RESET             0x02
00014 #define POWER_MODE                0x03
00015 #define THRESHOLD                 0x04
00016 #define SET_RAM_WR_PROTECTION     0x05
00017 #define DSP_RUN                   0x06
00018
00019
00021 #define PHASE_A           0x0A
00022 #define PHASE_B           0x0B
00023 #define PHASE_C           0x0C
00024 #define LINE              0x00
00025
00026 #define PHASE_VRMS          0x01
00027 #define PHASE_IRMS          0x02
00028 #define PHASE_ACTIVE_WH     0x03
00029 #define TOTAL_ACTIVE_WH     0x04
00030 #define PHASE_ACTIVE_POWER  0x05
00031 #define TOTAL_ACTIVE_POWER  0x06
00032
00033
00034 #include <stdarg.h>
00035
00036 float       measure(uint8_t cmd,uint8_t channel,float samples);
00037 uint16_t    hex2val(uint8_t cc);
00038 uint16_t    make16(uint8_t* buf,uint16_t idx);
00039 uint8_t     make8(uint8_t* buf,uint16_t idx);
00040 uint32_t    mask24(uint32_t val);
00041
00042
00043 #endif
```