CDP Coding Challenge: Packet Converter

Generated by Doxygen 1.9.4

Chapter 1

CDP Coding Challenge: Packet Parser

Description: This program converts two different packets to a text output in response to the Coding Challenge by Cambridge Design Partnership.

Toolchains Used

- GCC version 11.2.0
- CMake version 3.18.4 (Can use as low as 3.4 to compile though)
- Doxygen version 1.9.4
- cpputest version 4.0

Building the Application

This project uses CMake to autogenerate the needed compiler configuration files to build the application. This project was built and tested with GCC; so CMake generates a GNU makefile but CMake can autogenerate build files for another compilers if so desired. To use other compilers or build systems, you will need to run CMake manually.

Building using the Bash Script

To easily build this program, open up a bash terminal in the project folder and simply run the following script:

```
./make_program.sh
```

This script will run cmake and make then outputs the executable to `${project_folder}/build/bin`

Building "by hand"

These are the steps to run CMake and build the program without the script. These steps will assume a bash terminal is used, but can be adapted to your own terminal.

1. Open a bash terminal in the project folder
2. Make a folder called "build" and enter that folder

```
mkdir ./build
cd ./build
```

3. Run CMake while pointing to the project directory

```
cmake ../
```

4. This will generate a makefile in the build folder, so next run make

```
make
```

5. If successful, this will generate an executable located `${project_folder}/build/bin`

Running the application

The program needs an input bin file containing the data to be parsed. It should be passed as the only argument to the program on the command line. In a bash terminal, it should look similar to the following if run from in the bin folder.

```
./packet_converter /path/to/input/file
```

There is a test input file located at `${project_folder}/test_input_file/CodingTest.bin` for demostration purposes.

Unit Tests

See the README.md file in the unit test folder for information on how to build and run the unit tests. If cpputest has already been installed or built in the submodule, then the script `make_and_run_unit_tests.sh` can be run from the project folder in a bash termnial. This script will build the unit tests and then run them.

Doxygen

The Doxyfile provided generates both HTML and Latex outputs. It will output all files to `${project_folder}/doc/doxy_output` and can be generated with running Doxygen on the command line like so:

```
doxygen ./Doxyfile
```

The reference manual pdf, located in the doc folder was generated from the Latex output of the doxyfile.

Chapter 2

Todo List

Global convert_array_to_uint32 (uint8_t const *const p_array, const size_t length)

Assert that the length is <=4 bytes

Global main (int argc, char *argv[])

Potentially come back and rethink if we need any reason to error out of this loop

Global time_check (uint32_t prev_ts_ms, uint32_t current_ts_ms)

There is probably a better way to handle an uint32_t overflow or underflow situation

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 batt_packet Struct Reference

Structure for holding the data related to the battery packet.

`#include <packet_constants.h>`

Data Fields

- uint32_t time_stamp
  Timestamp of the packet recieved in milliseconds.
- uint8_t batt_status
  Status of the battery, 0-3 for VLOW,LOW,MED,HIGH.
- uint8_t err_check
  Error check value of the packet.

5.1.1 Detailed Description

Structure for holding the data related to the battery packet.

Definition at line 63 of file packet_constants.h.

5.1.2 Field Documentation

5.1.2.1 batt_status

`uint8_t batt_packet::batt_status`

Status of the battery, 0-3 for VLOW,LOW,MED,HIGH.

Definition at line 68 of file packet_constants.h.

5.1.2.2 err_check

```
uint8_t batt_packet::err_check
```

Error check value of the packet.

Definition at line 70 of file packet_constants.h.

5.1.2.3 time_stamp

```
uint32_t batt_packet::time_stamp
```

Timestamp of the packet recieved in milliseconds.

Definition at line 66 of file packet_constants.h.

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

- src/packet_constants.h

5.2 pwr_packet Struct Reference

Structure for holding the data related to the power packet.

```
#include <packet_constants.h>
```

Data Fields

- uint32_t time_stamp
  Timestamp of the packet recieved in milliseconds.
- uint32_t volts
  Volts of the power packet recieved.
- uint64_t milliamps
  Milliamps of the power packet recieved.
- uint8_t err_check
  Error check value of the packet.
- uint64_t milliwatts
  The calculated power from the volts ∗ milliamps.

5.2.1 Detailed Description

Structure for holding the data related to the power packet.

Definition at line 48 of file packet_constants.h.

5.2.2 Field Documentation

5.2.2.1 err_check

`uint8_t pwr_packet::err_check`

Error check value of the packet.

Definition at line 57 of file packet_constants.h.

5.2.2.2 milliamps

`uint64_t pwr_packet::milliamps`

Milliamps of the power packet recieved.

Definition at line 55 of file packet_constants.h.

5.2.2.3 milliwatts

`uint64_t pwr_packet::milliwatts`

The calculated power from the volts $*$ milliamps.

Definition at line 59 of file packet_constants.h.

5.2.2.4 time_stamp

`uint32_t pwr_packet::time_stamp`

Timestamp of the packet recieved in milliseconds.

Definition at line 51 of file packet_constants.h.

5.2.2.5 volts

`uint32_t pwr_packet::volts`

Volts of the power packet recieved.

Definition at line 53 of file packet_constants.h.

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

- src/packet_constants.h

Chapter 6

File Documentation

6.1 README.md File Reference

6.2 src/calculations.c File Reference

Source file of the module for holding the functions used for calculating power.

`#include "calculations.h"`
Include dependency graph for calculations.c:

6.3 calculations.c

Go to the documentation of this file.

```
00001 /**
00002  * @file calculations.c
00003  *
00004  * @brief Source file of the module for holding the functions used for calculating power.
00005  */
00006 #include "calculations.h"
00007
00008 /**
00009  * @brief Function for calculating the power from the voltage and amperage given
00010  *
00011  * @param pwr_numbers Structure holding the power information
00012  * @return int Success = 0, failure = -1
00013  */
00014 int calc_power(pwr_packet_t * const pwr_numbers)
00015 {
00016     int ret_status = -1;
00017     if (NULL != pwr_numbers)
00018     {
00019         pwr_numbers->milliwatts = (uint64_t)pwr_numbers->volts * pwr_numbers->milliamps;
00020         ret_status = 0;
00021     }
00022
00023     return ret_status;
00024 }
00025
00026 /**
00027  * @brief Function for taking the modulus of the data in an array
00028  *
00029  * @param p_array Array to be used
00030  * @param length Length of that array
00031  * @return uint32_t Modulus of the array
00032  */
00033 uint32_t mod_of_array(uint8_t const * const p_array, size_t length)
00034 {
00035     uint32_t ret_mod = 0;
00036     for (size_t i = 0; length > i; ++i)
00037     {
00038         ret_mod = (ret_mod + p_array[i]);
00039         ret_mod %= 256u;
00040     }
00041     return ret_mod;
00042 }
```

6.4 src/calculations.h File Reference

Header of the module for holding the functions used for calculating power.

```
#include "packet_constants.h"
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for calculations.h: This graph shows which files directly or indirectly include this file:

Functions

- int calc_power (pwr_packet_t *const pwr_numbers)

  Function for calculating the power from the voltage and amperage given.
- uint32_t mod_of_array (uint8_t const *const p_array, size_t length)

  Function for taking the modulus of the data in an array.

6.4.1 Detailed Description

Header of the module for holding the functions used for calculating power.

Definition in file calculations.h.

6.4.2 Function Documentation

6.4.2.1 calc_power()

```
int calc_power (
            pwr_packet_t *const pwr_numbers )
```

Function for calculating the power from the voltage and amperage given.

Parameters

pwr_numbers	Structure holding the power information

Returns

int Success = 0, failure = -1

Definition at line 14 of file calculations.c.

6.4.2.2 mod_of_array()

```
uint32_t mod_of_array (
            uint8_t const *const p_array,
            size_t length )
```

Function for taking the modulus of the data in an array.

Parameters

p_array	Array to be used
length	Length of that array

Returns

uint32_t Modulus of the array

Definition at line 33 of file calculations.c.

6.5 calculations.h

Go to the documentation of this file.

```
00001 /**
00002  * @file calculations.h
00003  *
00004  * @brief Header of the module for holding the functions used for calculating power
00005  *
00006  */
00007
00008 #ifndef CALCULATIONS_H
00009 #define CALCULATIONS_H
00010
00011 #include "packet_constants.h"
00012
00013 #include <stdint.h>
00014 #include <stddef.h>
00015
00016 #ifdef __cplusplus
00017 extern "C" {
00018 #endif
00019
00020 int calc_power(pwr_packet_t * const pwr_numbers);
00021 uint32_t mod_of_array(uint8_t const * const p_array, size_t length);
00022
00023 #ifdef __cplusplus
00024 }
00025 #endif
00026
00027 #endif // CALCULATIONS_H
```

6.6 src/main.c File Reference

```
#include "calculations.h"
#include "packet_constants.h"
#include "packet_parser.h"
#include "state_handler.h"
#include <stdint.h>
#include <stdio.h>
#include <stddef.h>
#include <errno.h>
```

Include dependency graph for main.c:

Functions

- int parse_arguments (const int argc, char ∗argv[ ], FILE ∗∗p_file)

  Function for parsing the inputs to the program.
- void get_pack_from_file (FILE ∗∗p_file, uint8_t ∗out_buff, const size_t size)

  Get the pack from file object.
- int main (int argc, char ∗argv[ ])

  Main function of the program.

6.6.1 Function Documentation

6.6.1.1 get_pack_from_file()

```
void get_pack_from_file (
            FILE ** p_file,
            uint8_t * out_buff,
            const size_t size )
```

Get the pack from file object.

Parameters

p_file	Pointer to the file being processed
out_buff	Outputing the data obtained from the file
size	Size of the buffer to output data in

Definition at line 114 of file main.c.

6.6.1.2 main()

```
int main (
            int argc,
            char * argv[] )
```

Main function of the program.

Parameters

argc	Number of arguments
argv	Pointer to the list of inputed arguments

Returns

int Status of program

Todo Potentially come back and rethink if we need any reason to error out of this loop

Definition at line 21 of file main.c.

6.6.1.3 parse_arguments()

```
int parse_arguments (
            const int argc,
            char * argv[],
            FILE ** p_file )
```

Function for parsing the inputs to the program.

Parameters

argc	Number of arguments being entered
argv	Array of the arguments
p_file	Output of the file being expected to be passed. Will be NULL if invalid

Returns

int Return status, 0 for success, error code for failure

Definition at line 80 of file main.c.

6.7 main.c

Go to the documentation of this file.

```
00001 #include "calculations.h"
00002 #include "packet_constants.h"
00003 #include "packet_parser.h"
00004 #include "state_handler.h"
00005
00006 #include <stdint.h>
00007 #include <stdio.h>
00008 #include <stddef.h>
00009 #include <errno.h>
00010
00011 int parse_arguments(const int argc, char * argv[], FILE ** p_file);
00012 void get_pack_from_file(FILE ** p_file, uint8_t * out_buff, const size_t size);
00013
00014 /**
00015  * @brief Main function of the program
00016  *
00017  * @param argc Number of arguments
00018  * @param argv Pointer to the list of inputed arguments
00019  * @return int Status of program
00020  */
00021 int main(int argc, char * argv[])
00022 {
00023     int ret_status = 0;
00024     FILE * p_bin_file = NULL;
00025
00026     ret_status = parse_arguments(argc, argv, &p_bin_file);
00027
00028     uint8_t packet_buffer[MAX_PACKET_SIZE] = {0};
00029     pwr_packet_t pwr_pack = {0};
00030     batt_packet_t batt_pack = {0};
```

```
00031     if (0 == ret_status)
00032     {
00033         ///@todo Potentially come back and rethink if we need any reason to
00034         /// error out of this loop
00035         while ((0 == feof(p_bin_file)) ) //&& (0 == ret_status))
00036         {
00037             packet_buffer[START_OF_PT_LOC] = (uint8_t)fgetc(p_bin_file);
00038             packet_type_t pack_type = determine_packet_type(packet_buffer[START_OF_PT_LOC]);
00039             switch (pack_type)
00040             {
00041             case power_pack:
00042                 get_pack_from_file(&p_bin_file,
00043                                    &packet_buffer[START_OF_TS_LOC],
00044                                    (SIZE_OF_PWR_PACK - PACKET_TYPE_NUM_OF_BYTES));
00045                 ret_status = process_pwr_packet(packet_buffer, &pwr_pack);
00046                 break;
00047             case battery_pack:
00048                 get_pack_from_file(&p_bin_file,
00049                                    &packet_buffer[START_OF_TS_LOC],
00050                                    (SIZE_OF_BATT_PACK - PACKET_TYPE_NUM_OF_BYTES));
00051                 ret_status = process_batt_packet(packet_buffer, &batt_pack);
00052                 break;
00053             default:
00054                 break;
00055             }
00056
00057             if (0 == ret_status)
00058             {
00059                 ret_status = process_state_and_transitions(pack_type,
00060                                                            &pwr_pack,
00061                                                            &batt_pack);
00062             }
00063         }
00064     }
00065
00066     (void)fclose(p_bin_file);
00067
00068     return ret_status;
00069 }
00070
00071 /**
00072  * @brief Function for parsing the inputs to the program
00073  *
00074  * @param argc Number of arguments being entered
00075  * @param argv Array of the arguments
00076  * @param p_file Output of the file being expected to be passed. Will be NULL if
00077  *               invalid
00078  * @return int Return status, 0 for success, error code for failure
00079  */
00080 int parse_arguments(const int argc, char * argv[], FILE ** p_file)
00081 {
00082     int ret_status = 0;
00083     if (2 == argc)
00084     {
00085         *p_file = fopen(argv[1], "rt");
00086         if (NULL == *p_file)
00087         {
00088             perror("ERR: Invalid File Location\n");
00089             ret_status = ENFILE;
00090         }
00091     }
00092     else if (2 > argc)
00093     {
00094         printf("ERR: No imput file passed in! ");
00095         printf("ie: packet_converter /path/to/bin/file \n");
00096         ret_status = ENODATA;
00097     }
00098     else
00099     {
00100         printf("ERR: This requires only 1 input bin file ");
00101         printf("ie: packet_converter /path/to/bin/file \n");
00102         ret_status = ENFILE;
00103     }
00104     return ret_status;
00105 }
00106
00107 /**
00108  * @brief Get the pack from file object
00109  *
00110  * @param p_file Pointer to the file being processed
00111  * @param out_buff Outputing the data obtained from the file
00112  * @param size Size of the buffer to output data in
00113  */
00114 void get_pack_from_file(FILE ** p_file, uint8_t * out_buff, const size_t size)
00115 {
00116     for (size_t i = 0; size > i; ++i)
00117     {
```

```
00118         out_buff[i] = (uint8_t)fgetc(*p_file);
00119     }
00120 }
```

6.8 src/packet_constants.h File Reference

Header for information about packets (ie type and constants)

`#include <stdint.h>`
Include dependency graph for packet_constants.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct pwr_packet

  Structure for holding the data related to the power packet.
- struct batt_packet

  Structure for holding the data related to the battery packet.

Macros

- #define PACKET_TYPE_NUM_OF_BYTES 1u
- #define TIMESTAMP_NUM_OF_BYTES 4u
- #define VOLTS_NUM_OF_BYTES 4u
- #define MILLIAMP_NUM_OF_BYTES 8u
- #define ERR_CHECK_NUM_OF_BYTES 1u
- #define BATT_STAT_NUM_OF_BYTES 1u
- #define SIZE_OF_PWR_PACK
- #define SIZE_OF_BATT_PACK
- #define MAX_PACKET_SIZE
- #define PWR_PACKET_TYPE_BYTE 0x00u
- #define BATT_PACKET_TYPE_BYTE 0x01u
- #define START_OF_PT_LOC (0u)
- #define START_OF_TS_LOC (START_OF_PT_LOC + PACKET_TYPE_NUM_OF_BYTES)
- #define PWR_START_OF_VOLTS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
- #define PWR_START_OF_MILLIAMP_LOC (PWR_START_OF_VOLTS_LOC + VOLTS_NUM_OF_BYTES)
- #define PWR_START_OF_ERR_CHECK_LOC (PWR_START_OF_MILLIAMP_LOC + MILLIAMP_NUM_OF_BYTES)
- #define BATT_START_OF_BATT_STATUS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
- #define BATT_START_OF_ERR_CHECK_LOC (BATT_START_OF_BATT_STATUS_LOC + BATT_STAT_NUM_OF_BYTES)

Typedefs

- typedef struct pwr_packet pwr_packet_t

  Structure for holding the data related to the power packet.
- typedef struct batt_packet batt_packet_t

  Structure for holding the data related to the battery packet.

Enumerations

- enum packet_type_t { power_pack = 0 , battery_pack , error_type , num_of_types }
  Typedef Enum to indicate which type of packet we are working with.

6.8.1 Detailed Description

Header for information about packets (ie type and constants)

Definition in file packet_constants.h.

6.8.2 Macro Definition Documentation

6.8.2.1 BATT_PACKET_TYPE_BYTE

```
#define BATT_PACKET_TYPE_BYTE 0x01u
```

Definition at line 38 of file packet_constants.h.

6.8.2.2 BATT_START_OF_BATT_STATUS_LOC

```
#define BATT_START_OF_BATT_STATUS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
```

Definition at line 44 of file packet_constants.h.

6.8.2.3 BATT_START_OF_ERR_CHECK_LOC

```
#define BATT_START_OF_ERR_CHECK_LOC (BATT_START_OF_BATT_STATUS_LOC + BATT_STAT_NUM_OF_BYTES)
```

Definition at line 45 of file packet_constants.h.

6.8.2.4 BATT_STAT_NUM_OF_BYTES

```
#define BATT_STAT_NUM_OF_BYTES 1u
```

Definition at line 21 of file packet_constants.h.

6.8.2.5 ERR_CHECK_NUM_OF_BYTES

```
#define ERR_CHECK_NUM_OF_BYTES 1u
```

Definition at line 20 of file packet_constants.h.

6.8.2.6 MAX_PACKET_SIZE

```
#define MAX_PACKET_SIZE
```

Value:

```
((SIZE_OF_PWR_PACK > SIZE_OF_BATT_PACK) ? \
SIZE_OF_PWR_PACK : SIZE_OF_BATT_PACK)
```

Definition at line 34 of file packet_constants.h.

6.8.2.7 MILLIAMP_NUM_OF_BYTES

```
#define MILLIAMP_NUM_OF_BYTES 8u
```

Definition at line 19 of file packet_constants.h.

6.8.2.8 PACKET_TYPE_NUM_OF_BYTES

```
#define PACKET_TYPE_NUM_OF_BYTES 1u
```

Definition at line 16 of file packet_constants.h.

6.8.2.9 PWR_PACKET_TYPE_BYTE

```
#define PWR_PACKET_TYPE_BYTE 0x00u
```

Definition at line 37 of file packet_constants.h.

6.8.2.10 PWR_START_OF_ERR_CHECK_LOC

```
#define PWR_START_OF_ERR_CHECK_LOC (PWR_START_OF_MILLIAMP_LOC + MILLIAMP_NUM_OF_BYTES)
```

Definition at line 43 of file packet_constants.h.

6.8.2.11 PWR_START_OF_MILLIAMP_LOC

```
#define PWR_START_OF_MILLIAMP_LOC (PWR_START_OF_VOLTS_LOC + VOLTS_NUM_OF_BYTES)
```

Definition at line 42 of file packet_constants.h.

6.8.2.12 PWR_START_OF_VOLTS_LOC

```
#define PWR_START_OF_VOLTS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
```

Definition at line 41 of file packet_constants.h.

6.8.2.13 SIZE_OF_BATT_PACK

```
#define SIZE_OF_BATT_PACK
```

Value:

```
(PACKET_TYPE_NUM_OF_BYTES \
+ TIMESTAMP_NUM_OF_BYTES \
+ BATT_STAT_NUM_OF_BYTES \
+ ERR_CHECK_NUM_OF_BYTES)
```

Definition at line 29 of file packet_constants.h.

6.8.2.14 SIZE_OF_PWR_PACK

```
#define SIZE_OF_PWR_PACK
```

Value:

```
( PACKET_TYPE_NUM_OF_BYTES \
+ TIMESTAMP_NUM_OF_BYTES \
+ VOLTS_NUM_OF_BYTES \
+ MILLIAMP_NUM_OF_BYTES \
+ ERR_CHECK_NUM_OF_BYTES)
```

Definition at line 23 of file packet_constants.h.

6.8.2.15 START_OF_PT_LOC

```
#define START_OF_PT_LOC (0u)
```

Definition at line 39 of file packet_constants.h.

6.8.2.16 START_OF_TS_LOC

```
#define START_OF_TS_LOC (START_OF_PT_LOC + PACKET_TYPE_NUM_OF_BYTES)
```

Definition at line 40 of file packet_constants.h.

6.8.2.17 TIMESTAMP_NUM_OF_BYTES

```
#define TIMESTAMP_NUM_OF_BYTES 4u
```

Definition at line 17 of file packet_constants.h.

6.8.2.18 VOLTS_NUM_OF_BYTES

```
#define VOLTS_NUM_OF_BYTES 4u
```

Definition at line 18 of file packet_constants.h.

6.8.3 Typedef Documentation

6.8.3.1 batt_packet_t

```
typedef struct batt_packet batt_packet_t
```

Structure for holding the data related to the battery packet.

6.8.3.2 pwr_packet_t

```
typedef struct pwr_packet pwr_packet_t
```

Structure for holding the data related to the power packet.

6.8.4 Enumeration Type Documentation

6.8.4.1 packet_type_t

```
enum packet_type_t
```

Typedef Enum to indicate which type of packet we are working with.

Enumerator

power_pack	
battery_pack	
error_type	
num_of_types	

Definition at line 74 of file packet_constants.h.

6.9 packet_constants.h

Go to the documentation of this file.

```
00001 /**
00002  * @file packet_constants.h
00003  *
00004  * @brief Header for information about packets (ie type and constants)
00005  */
00006
00007 #ifndef PACKET_CONSTANTS_H
00008 #define PACKET_CONSTANTS_H
00009
00010 #include <stdint.h>
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 #define PACKET_TYPE_NUM_OF_BYTES 1u
00017 #define TIMESTAMP_NUM_OF_BYTES 4u
00018 #define VOLTS_NUM_OF_BYTES 4u
00019 #define MILLIAMP_NUM_OF_BYTES 8u
00020 #define ERR_CHECK_NUM_OF_BYTES 1u
00021 #define BATT_STAT_NUM_OF_BYTES 1u
00022
00023 #define SIZE_OF_PWR_PACK ( PACKET_TYPE_NUM_OF_BYTES \
00024                          + TIMESTAMP_NUM_OF_BYTES \
00025                          + VOLTS_NUM_OF_BYTES \
00026                          + MILLIAMP_NUM_OF_BYTES \
00027                          + ERR_CHECK_NUM_OF_BYTES)
00028
00029 #define SIZE_OF_BATT_PACK (PACKET_TYPE_NUM_OF_BYTES \
00030                           + TIMESTAMP_NUM_OF_BYTES \
00031                           + BATT_STAT_NUM_OF_BYTES \
00032                           + ERR_CHECK_NUM_OF_BYTES)
00033
00034 #define MAX_PACKET_SIZE ((SIZE_OF_PWR_PACK > SIZE_OF_BATT_PACK) ? \
00035                          SIZE_OF_PWR_PACK : SIZE_OF_BATT_PACK)
00036
00037 #define PWR_PACKET_TYPE_BYTE 0x00u
00038 #define BATT_PACKET_TYPE_BYTE 0x01u
00039 #define START_OF_PT_LOC (0u)
00040 #define START_OF_TS_LOC (START_OF_PT_LOC + PACKET_TYPE_NUM_OF_BYTES)
00041 #define PWR_START_OF_VOLTS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
00042 #define PWR_START_OF_MILLIAMP_LOC (PWR_START_OF_VOLTS_LOC + VOLTS_NUM_OF_BYTES)
00043 #define PWR_START_OF_ERR_CHECK_LOC (PWR_START_OF_MILLIAMP_LOC + MILLIAMP_NUM_OF_BYTES)
00044 #define BATT_START_OF_BATT_STATUS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
00045 #define BATT_START_OF_ERR_CHECK_LOC (BATT_START_OF_BATT_STATUS_LOC + BATT_STAT_NUM_OF_BYTES)
00046
00047 /// Structure for holding the data related to the power packet
00048 typedef struct pwr_packet
00049 {
00050     /// Timestamp of the packet recieved in milliseconds
00051     uint32_t time_stamp;
00052     /// Volts of the power packet recieved
00053     uint32_t volts;
00054     /// Milliamps of the power packet recieved
00055     uint64_t milliamps;
00056     /// Error check value of the packet
00057     uint8_t err_check;
00058     /// The calculated power from the volts * milliamps
00059     uint64_t milliwatts;
00060 } pwr_packet_t;
00061
00062 /// Structure for holding the data related to the battery packet
00063 typedef struct batt_packet
```

```
00064 {
00065     /// Timestamp of the packet recieved in milliseconds
00066     uint32_t time_stamp;
00067     /// Status of the battery, 0-3 for VLOW,LOW,MED,HIGH
00068     uint8_t batt_status;
00069     /// Error check value of the packet
00070     uint8_t err_check;
00071 } batt_packet_t;
00072
00073 /// Typedef Enum to indicate which type of packet we are working with
00074 typedef enum
00075 {
00076     power_pack = 0,
00077     battery_pack,
00078     error_type,
00079     num_of_types
00080 } packet_type_t;
00081
00082 #ifdef __cplusplus
00083 }
00084 #endif
00085
00086 #endif // PACKET_CONSTANTS_H
```

6.10 src/packet_parser.c File Reference

Source for the module for parsing the incoming packet data.

```
#include "packet_parser.h"
#include "calculations.h"
#include <stdio.h>
```

Include dependency graph for packet_parser.c:

Functions

- uint32_t convert_array_to_uint32 (uint8_t const *const p_array, const size_t length)

  Function for converting an uint8_t array to an uint32_t.
- int check_for_pack_error (const packet_type_t pack_type, uint8_t const *const p_packet)

  Helper function to check the error byte for integraty.
- packet_type_t determine_packet_type (const uint8_t first_byte_of_pack)

  Function for determining the type of the incoming packet.
- int process_pwr_packet (uint8_t const *const p_packet_buf, pwr_packet_t *const p_out_pack)

  Function for creating a power packet from an input buffer.
- int process_batt_packet (uint8_t const *const p_packet_buf, batt_packet_t *const p_out_pack)

  Function for creating a battery status packet from an input buffer.

6.10.1 Detailed Description

Source for the module for parsing the incoming packet data.

Definition in file packet_parser.c.

6.10.2 Function Documentation

6.10.2.1 check_for_pack_error()

```
int check_for_pack_error (
            const packet_type_t pack_type,
            uint8_t const *const p_packet )
```

Helper function to check the error byte for integraty.

Parameters

pack_type	Type of packet being checked
p_packet	Pointer to the packet buffer

Returns

int Success = 0

Definition at line 130 of file packet_parser.c.

6.10.2.2 convert_array_to_uint32()

```
uint32_t convert_array_to_uint32 (
            uint8_t const *const p_array,
            const size_t length )
```

Function for converting an uint8_t array to an uint32_t.

Todo Assert that the length is <=4 bytes

Parameters

p_array	Array to be converted
length	Length of the array in bytes

Returns

uint32_t

Definition at line 110 of file packet_parser.c.

6.10.2.3 determine_packet_type()

```
packet_type_t determine_packet_type (
            const uint8_t first_byte_of_pack )
```

Function for determining the type of the incoming packet.

Parameters

first_byte_of_pack	1st byte of the packet

Returns

packet_type_t Type of packet, will return the Enum error_type if it's an invalid packet type

Definition at line 23 of file packet_parser.c.

6.10.2.4 process_batt_packet()

```
int process_batt_packet (
            uint8_t const *const p_packet_buf,
            batt_packet_t *const p_out_pack )
```

Function for creating a battery status packet from an input buffer.

Parameters

packet_buf	Buffer to be converted

Returns

batt_packet_t Structure with the battery information in it

Definition at line 81 of file packet_parser.c.

6.10.2.5 process_pwr_packet()

```
int process_pwr_packet (
            uint8_t const *const p_packet_buf,
            pwr_packet_t *const p_out_pack )
```

Function for creating a power packet from an input buffer.

Parameters

packet_buf	Buffer to be converted

Returns

pwr_packet_t Structure with the power information in it

Definition at line 48 of file packet_parser.c.

6.11 packet_parser.c

Go to the documentation of this file.

```
00001 /**
00002  * @file packet_parser.c
00003  *
00004  * @brief Source for the module for parsing the incoming packet data
00005  */
00006
00007 #include "packet_parser.h"
00008 #include "calculations.h"
00009 #include <stdio.h>
00010
00011 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00012                                  const size_t length);
00013 int check_for_pack_error(const packet_type_t pack_type,
00014                          uint8_t const * const p_packet);
00015
00016 /**
00017  * @brief Function for determining the type of the incoming packet
00018  *
00019  * @param first_byte_of_pack 1st byte of the packet
00020  * @return packet_type_t Type of packet, will return the Enum error_type if it's
00021  *                       an invalid packet type
00022  */
00023 packet_type_t determine_packet_type(const uint8_t first_byte_of_pack)
00024 {
00025     packet_type_t ret_type = error_type;
00026     if (PWR_PACKET_TYPE_BYTE == first_byte_of_pack)
00027     {
00028         ret_type = power_pack;
00029     }
00030     else if (BATT_PACKET_TYPE_BYTE == first_byte_of_pack)
00031     {
00032         ret_type = battery_pack;
00033     }
00034     else
00035     {
00036         // Error so do nothing
00037     }
00038
00039     return ret_type;
00040 }
00041
00042 /**
00043  * @brief Function for creating a power packet from an input buffer
00044  *
00045  * @param packet_buf Buffer to be converted
00046  * @return pwr_packet_t Structure with the power information in it
00047  */
00048 int process_pwr_packet(uint8_t const * const p_packet_buf,
00049                        pwr_packet_t * const p_out_pack)
00050 {
00051     int ret_status = -1;
00052     if((NULL != p_packet_buf) && (NULL != p_out_pack))
00053     {
00054         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00055                                                          TIMESTAMP_NUM_OF_BYTES);
00056
00057         p_out_pack->volts = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_VOLTS_LOC],
00058                                                     VOLTS_NUM_OF_BYTES);
00059
00060         p_out_pack->milliamps = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_MILLIAMP_LOC],
00061                                                         MILLIAMP_NUM_OF_BYTES);
00062
00063         p_out_pack->err_check = (uint16_t)p_packet_buf[PWR_START_OF_ERR_CHECK_LOC];
00064
00065         ret_status = calc_power(p_out_pack);
00066
00067         if(0 == ret_status)
00068         {
00069             ret_status = check_for_pack_error(power_pack, p_packet_buf);
00070         }
00071     }
00072     return ret_status;
00073 }
00074
00075 /**
00076  * @brief Function for creating a battery status packet from an input buffer
00077  *
00078  * @param packet_buf Buffer to be converted
00079  * @return batt_packet_t Structure with the battery information in it
00080  */
00081 int process_batt_packet(uint8_t const * const p_packet_buf,
00082                         batt_packet_t * const p_out_pack)
00083 {
00084     int ret_status = -1;
00085
00086     if((NULL != p_packet_buf) && (NULL != p_out_pack))
00087     {
```

```
00088         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00089                                                          TIMESTAMP_NUM_OF_BYTES);
00090
00091         p_out_pack->batt_status = p_packet_buf[BATT_START_OF_BATT_STATUS_LOC];
00092
00093         p_out_pack->err_check = (uint16_t)p_packet_buf[BATT_START_OF_ERR_CHECK_LOC];
00094
00095         ret_status = check_for_pack_error(battery_pack, p_packet_buf);
00096     }
00097
00098     return ret_status;
00099 }
00100
00101 /**
00102  * @brief Function for converting an uint8_t array to an uint32_t
00103  *
00104  * @todo Assert that the length is <=4 bytes
00105  *
00106  * @param p_array Array to be converted
00107  * @param length Length of the array in bytes
00108  * @return uint32_t
00109  */
00110 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00111                                  const size_t length)
00112 {
00113     uint32_t ret_val = 0;
00114     for (size_t i = 0u; length > i; ++i)
00115     {
00116         ret_val «= 8u;
00117         ret_val += p_array[i];
00118     }
00119
00120     return ret_val;
00121 }
00122
00123 /**
00124  * @brief Helper function to check the error byte for integraty
00125  *
00126  * @param pack_type Type of packet being checked
00127  * @param p_packet Pointer to the packet buffer
00128  * @return int Success = 0
00129  */
00130 int check_for_pack_error(const packet_type_t pack_type,
00131                          uint8_t const * const p_packet)
00132 {
00133     int ret_status = 0;
00134     uint32_t mod_of_pack = 0;
00135     switch (pack_type)
00136     {
00137     case power_pack:
00138         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_PWR_PACK - ERR_CHECK_NUM_OF_BYTES));
00139         if (mod_of_pack != p_packet[PWR_START_OF_ERR_CHECK_LOC])
00140         {
00141             ret_status = -1;
00142             printf("ERR: Packet Failed Error Check\n");
00143         }
00144         break;
00145     case battery_pack:
00146         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_BATT_PACK - ERR_CHECK_NUM_OF_BYTES));
00147         if (mod_of_pack != p_packet[BATT_START_OF_ERR_CHECK_LOC])
00148         {
00149             ret_status = -1;
00150             printf("ERR: Packet Failed Error Check\n");
00151         }
00152         break;
00153     default:
00154         // Do nothing, but return an error.
00155         ret_status = -1;
00156         break;
00157     }
00158
00159     return ret_status;
00160 }
```

6.12 src/packet_parser.h File Reference

Header for the module for parsing the incoming packet data.

```
#include "packet_constants.h"
#include <stdint.h>
```

`#include <stddef.h>`
Include dependency graph for packet_parser.h: This graph shows which files directly or indirectly include this file:

Functions

- packet_type_t determine_packet_type (const uint8_t first_byte_of_pack)
  Function for determining the type of the incoming packet.
- int process_pwr_packet (uint8_t const *const p_packet_buf, pwr_packet_t *const p_out_pack)
  Function for creating a power packet from an input buffer.
- int process_batt_packet (uint8_t const *const p_packet_buf, batt_packet_t *const p_out_pack)
  Function for creating a battery status packet from an input buffer.

6.12.1 Detailed Description

Header for the module for parsing the incoming packet data.

Definition in file packet_parser.h.

6.12.2 Function Documentation

6.12.2.1 determine_packet_type()

```
packet_type_t determine_packet_type (
            const uint8_t first_byte_of_pack )
```

Function for determining the type of the incoming packet.

Parameters

first_byte_of_pack	1st byte of the packet

Returns

packet_type_t Type of packet, will return the Enum error_type if it's an invalid packet type

Definition at line 23 of file packet_parser.c.

6.12.2.2 process_batt_packet()

```
int process_batt_packet (
            uint8_t const *const p_packet_buf,
            batt_packet_t *const p_out_pack )
```

Function for creating a battery status packet from an input buffer.

Parameters

packet_buf	Buffer to be converted

Returns

batt_packet_t Structure with the battery information in it

Definition at line 81 of file packet_parser.c.

6.12.2.3 process_pwr_packet()

```
int process_pwr_packet (
            uint8_t const *const p_packet_buf,
            pwr_packet_t *const p_out_pack )
```

Function for creating a power packet from an input buffer.

Parameters

packet_buf	Buffer to be converted

Returns

pwr_packet_t Structure with the power information in it

Definition at line 48 of file packet_parser.c.

6.13 packet_parser.h

Go to the documentation of this file.

```
00001 /**
00002  * @file packet_parser.h
00003  *
00004  * @brief Header for the module for parsing the incoming packet data
00005  */
00006
00007 #ifndef PACKET_PARSER_H
00008 #define PACKET_PARSER_H
00009
00010 #include "packet_constants.h"
00011 #include <stdint.h>
00012 #include <stddef.h>
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 packet_type_t determine_packet_type(const uint8_t first_byte_of_pack);
00019 int process_pwr_packet(uint8_t const * const p_packet_buf,
00020                        pwr_packet_t * const p_out_pack);
00021 int process_batt_packet(uint8_t const * const p_packet_buf,
00022                         batt_packet_t * const p_out_pack);
00023
00024 #ifdef __cplusplus
00025 }
00026 #endif
00027
00028 #endif // PACKET_PARSER_H
```

6.14 src/state_handler.c File Reference

Source file of the module for handling any state related functions.

```
#include "state_handler.h"
#include <stdbool.h>
#include <stdio.h>
```

Include dependency graph for state_handler.c:

Macros

- #define STATE_0_UPPER_BOUNDS 200u
- #define STATE_1_LOWER_BOUNDS 300u
- #define STATE_1_UPPER_BOUNDS 450u
- #define STATE_2_LOWER_BOUNDS 550u
- #define STATE_2_UPPER_BOUNDS 650u
- #define STATE_3_LOWER_BOUNDS 800u
- #define STATE_3_UPPER_BOUNDS 1200u

Enumerations

- enum states_t {
  STATE_0 = 0 , STATE_1 , STATE_2 , STATE_3 ,
  NUM_OF_STATES }

Functions

- uint32_t calc_time_from_start_ms_to_sec (uint32_t current_ts_ms)

  Function for calculating the time since the start of the program.
- states_t determine_state (pwr_packet_t const ∗const pwr)

  Function for determining the state from power.
- void set_initial_timestamp (const uint32_t ts_ms)

  Set the initial timestamp object.
- bool time_check (uint32_t prev_ts_ms, uint32_t current_ts_ms)

  Function to check if 2 timestamps are greater than 10 ms apart.
- int process_state_and_transitions (packet_type_t pack_type, pwr_packet_t const ∗const pwr, batt_packet_t const ∗const batt)

  Function for processing the states of either a power or battery state.

Variables

- const char ∗ batt_states [4] = {"VLOW", "LOW", "MED", "HIGH"}
- static uint32_t inital_ts_ms = 0

6.14.1 Detailed Description

Source file of the module for handling any state related functions.

Definition in file state_handler.c.

6.14.2 Macro Definition Documentation

6.14.2.1 STATE_0_UPPER_BOUNDS

```
#define STATE_0_UPPER_BOUNDS 200u
```

Definition at line 10 of file state_handler.c.

6.14.2.2 STATE_1_LOWER_BOUNDS

```
#define STATE_1_LOWER_BOUNDS 300u
```

Definition at line 11 of file state_handler.c.

6.14.2.3 STATE_1_UPPER_BOUNDS

```
#define STATE_1_UPPER_BOUNDS 450u
```

Definition at line 12 of file state_handler.c.

6.14.2.4 STATE_2_LOWER_BOUNDS

```
#define STATE_2_LOWER_BOUNDS 550u
```

Definition at line 13 of file state_handler.c.

6.14.2.5 STATE_2_UPPER_BOUNDS

```
#define STATE_2_UPPER_BOUNDS 650u
```

Definition at line 14 of file state_handler.c.

6.14.2.6 STATE_3_LOWER_BOUNDS

```
#define STATE_3_LOWER_BOUNDS 800u
```

Definition at line 15 of file state_handler.c.

6.14.2.7 STATE_3_UPPER_BOUNDS

```
#define STATE_3_UPPER_BOUNDS 1200u
```

Definition at line 16 of file state_handler.c.

6.14.3 Enumeration Type Documentation

6.14.3.1 states_t

```
enum states_t
```

Enumerator

STATE_0	Initial State for Power reading, 0 to 200 mW.
STATE_1	1st State for Power reading, 300 to 450 mW
STATE_2	2nd State for Power reading, 550 to 650 mW
STATE_3	2nd State for Power reading, 800 to 1200 mW
NUM_OF_STATES	Number of total states.

Definition at line 18 of file state_handler.c.

6.14.4 Function Documentation

6.14.4.1 calc_time_from_start_ms_to_sec()

```
uint32_t calc_time_from_start_ms_to_sec (
            uint32_t current_ts_ms )
```

Function for calculating the time since the start of the program.

Parameters

current_ts_ms	Current timestamp in milliseconds

Returns

uint32_t Time from start of program in seconds

Definition at line 114 of file state_handler.c.

6.14.4.2 determine_state()

```
states_t determine_state (
            pwr_packet_t const *const pwr )
```

Function for determining the state from power.

Parameters

pwr	Pointer to the struct that holds the power information

Returns

states_t Current state, returns NUM_OF_STATES if invalid

Definition at line 126 of file state_handler.c.

6.14.4.3 process_state_and_transitions()

```
int process_state_and_transitions (
            packet_type_t pack_type,
            pwr_packet_t const *const pwr,
            batt_packet_t const *const batt )
```

Function for processing the states of either a power or battery state.

Parameters

pack_type	Type of packet being processed
pwr	Pointer to the struct that holds the power information
batt	Pointer to the struct that holds the battery information

Returns

int 0 == Success

Definition at line 48 of file state_handler.c.

6.14.4.4 set_initial_timestamp()

```
void set_initial_timestamp (
            const uint32_t ts_ms )
```

Set the initial timestamp object.

Parameters

ts	Initial timestampt to be saved in milliseconds

Definition at line 161 of file state_handler.c.

6.14.4.5 time_check()

```
bool time_check (
            uint32_t prev_ts_ms,
            uint32_t current_ts_ms )
```

Function to check if 2 timestamps are greater than 10 ms apart.

Todo There is probably a better way to handle an uint32_t overflow or underflow situation

Parameters

prev_ts_ms	Previous timestamp
current_ts_ms	Current timestamp

Returns

true If (current - prev) $>$ 10ms

false If (current - prev) $<$ 10ms

Definition at line 177 of file state_handler.c.

6.14.5 Variable Documentation

6.14.5.1 batt_states

```
const char* batt_states[4] = {"VLOW", "LOW", "MED", "HIGH"}
```

Definition at line 32 of file state_handler.c.

6.14.5.2 inital_ts_ms

```
uint32_t inital_ts_ms = 0  [static]
```

Definition at line 38 of file state_handler.c.

6.15 state_handler.c

Go to the documentation of this file.

```
00001 /**
00002  * @file state_handler.c
00003  *
00004  * @brief Source file of the module for handling any state related functions.
00005  */
00006 #include "state_handler.h"
00007 #include <stdbool.h>
00008 #include <stdio.h>
00009
00010 #define STATE_0_UPPER_BOUNDS 200u
00011 #define STATE_1_LOWER_BOUNDS 300u
00012 #define STATE_1_UPPER_BOUNDS 450u
00013 #define STATE_2_LOWER_BOUNDS 550u
00014 #define STATE_2_UPPER_BOUNDS 650u
00015 #define STATE_3_LOWER_BOUNDS 800u
00016 #define STATE_3_UPPER_BOUNDS 1200u
00017
00018 typedef enum
00019 {
00020     /// Initial State for Power reading, 0 to 200 mW
00021     STATE_0 = 0,
00022     /// 1st State for Power reading, 300 to 450 mW
00023     STATE_1,
00024     /// 2nd State for Power reading, 550 to 650 mW
00025     STATE_2,
00026     /// 2nd State for Power reading, 800 to 1200 mW
00027     STATE_3,
00028     /// Number of total states
00029     NUM_OF_STATES
00030 } states_t;
00031
00032 const char * batt_states[4] = {"VLOW", "LOW", "MED", "HIGH"};
00033 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms);
00034 states_t determine_state(pwr_packet_t const * const pwr);
00035 void set_initial_timestamp(const uint32_t ts_ms);
00036 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms);
00037
00038 static uint32_t inital_ts_ms = 0;
00039
00040 /**
00041  * @brief Function for processing the states of either a power or battery state
00042  *
00043  * @param pack_type Type of packet being processed
00044  * @param pwr Pointer to the struct that holds the power information
00045  * @param batt Pointer to the struct that holds the battery information
00046  * @return int 0 == Success
00047  */
00048 int process_state_and_transitions(packet_type_t pack_type,
00049                                   pwr_packet_t const * const pwr,
00050                                   batt_packet_t const * const batt)
00051 {
00052     int ret_status = 0;
00053     static uint32_t prev_ts_ms = 0;
00054     static states_t prev_state = STATE_0;
00055     static bool init_intial_ts = true;
00056     if ((NULL == pwr) || (NULL == batt))
00057     {
00058         ret_status = -1;
00059     }
00060     else
00061     {
00062         switch (pack_type)
00063         {
00064             case power_pack:
00065             {
00066                 if (init_intial_ts)
00067                 {
00068                     set_initial_timestamp(pwr->time_stamp);
00069                     init_intial_ts = false;
```

```
00070                 }
00071                 states_t current_state = determine_state(pwr);
00072                 if (current_state != prev_state)
00073                 {
00074                     if(time_check(prev_ts_ms, pwr->time_stamp))
00075                     {
00076                         printf("S;%u;%u-%u\n",
00077                                calc_time_from_start_ms_to_sec(pwr->time_stamp),
00078                                prev_state,
00079                                current_state);
00080                     }
00081                 }
00082                 prev_state = current_state;
00083                 break;
00084             }
00085             case battery_pack:
00086             {
00087                 if (init_intial_ts)
00088                 {
00089                     set_initial_timestamp(batt->time_stamp);
00090                     init_intial_ts = false;
00091                 }
00092                 printf("B;%u;%s\n",
00093                        calc_time_from_start_ms_to_sec(batt->time_stamp),
00094                        batt_states[batt->batt_status]);
00095                 break;
00096             }
00097             default:
00098             {
00099                 ret_status = -1;
00100                 break;
00101             }
00102         }
00103     }
00104
00105     return ret_status;
00106 }
00107
00108 /**
00109  * @brief Function for calculating the time since the start of the program
00110  *
00111  * @param current_ts_ms Current timestamp in milliseconds
00112  * @return uint32_t Time from start of program in seconds
00113  */
00114 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms)
00115 {
00116     uint32_t ts_ms = (current_ts_ms - inital_ts_ms);
00117     return (ts_ms / 1000);
00118 }
00119
00120 /**
00121  * @brief Function for determining the state from power
00122  *
00123  * @param pwr Pointer to the struct that holds the power information
00124  * @return states_t Current state, returns NUM_OF_STATES if invalid
00125  */
00126 states_t determine_state(pwr_packet_t const * const pwr)
00127 {
00128     states_t ret_state = NUM_OF_STATES;
00129     if (STATE_0_UPPER_BOUNDS >= pwr->milliwatts)
00130     {
00131         ret_state = STATE_0;
00132     }
00133     else if ((STATE_1_LOWER_BOUNDS <= pwr->milliwatts)
00134             && (STATE_1_UPPER_BOUNDS >= pwr->milliwatts))
00135     {
00136         ret_state = STATE_1;
00137     }
00138     else if ((STATE_2_LOWER_BOUNDS <= pwr->milliwatts)
00139             && (STATE_2_UPPER_BOUNDS >= pwr->milliwatts))
00140     {
00141         ret_state = STATE_2;
00142     }
00143     else if ((STATE_3_LOWER_BOUNDS <= pwr->milliwatts)
00144             && (STATE_3_UPPER_BOUNDS >= pwr->milliwatts))
00145     {
00146         ret_state = STATE_3;
00147     }
00148     else
00149     {
00150         //do nothing
00151     }
00152
00153     return ret_state;
00154 }
00155
00156 /**
```

```
00157  * @brief Set the initial timestamp object
00158  *
00159  * @param ts Initial timestampt to be saved in milliseconds
00160  */
00161 void set_initial_timestamp(const uint32_t ts_ms)
00162 {
00163     inital_ts_ms = ts_ms;
00164 }
00165
00166 /**
00167  * @brief Function to check if 2 timestamps are greater than 10 ms apart
00168  *
00169  * @todo There is probably a better way to handle an uint32_t overflow or
00170  *       underflow situation
00171  *
00172  * @param prev_ts_ms Previous timestamp
00173  * @param current_ts_ms Current timestamp
00174  * @return true If (current - prev) > 10ms
00175  * @return false If (current - prev) < 10ms
00176  */
00177 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms)
00178 {
00179     bool greater_than_10 = false;
00180
00181     if (prev_ts_ms < current_ts_ms)
00182     {
00183         if (10u <= (current_ts_ms - prev_ts_ms))
00184         {
00185             greater_than_10 = true;
00186         }
00187     }
00188
00189     return greater_than_10;
00190 }
```

6.16 src/state_handler.h File Reference

Header for the module for handling any state related functions.

```
#include "packet_constants.h"
#include <stdint.h>
```

Include dependency graph for state_handler.h: This graph shows which files directly or indirectly include this file:

Functions

- int process_state_and_transitions (packet_type_t pack_type, pwr_packet_t const *const pwr, batt_packet_t const *const batt)

  Function for processing the states of either a power or battery state.

6.16.1 Detailed Description

Header for the module for handling any state related functions.

Definition in file state_handler.h.

6.16.2 Function Documentation

6.16.2.1 process_state_and_transitions()

```
int process_state_and_transitions (
            packet_type_t pack_type,
            pwr_packet_t const *const pwr,
            batt_packet_t const *const batt )
```

Function for processing the states of either a power or battery state.

Parameters

pack_type	Type of packet being processed
pwr	Pointer to the struct that holds the power information
batt	Pointer to the struct that holds the battery information

Returns

int 0 == Success

Definition at line 48 of file state_handler.c.

6.17 state_handler.h

Go to the documentation of this file.

```
00001 /**
00002  * @file state_handler.h
00003  *
00004  * @brief Header for the module for handling any state related functions.
00005  */
00006
00007 #ifndef STATE_HANDLER_H
00008 #define STATE_HANDLER_H
00009
00010 #include "packet_constants.h"
00011
00012 #include <stdint.h>
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 int process_state_and_transitions(packet_type_t pack_type,
00019                                   pwr_packet_t const * const pwr,
00020                                   batt_packet_t const * const batt);
00021
00022 #ifdef __cplusplus
00023 }
00024 #endif
00025
00026 #endif // STATE_HANDLER_H
```

Index