CDP Coding Challenge: Packet Converter

Generated by Doxygen 1.9.4

Chapter 1

CDP Coding Challenge: Packet Parser

Coding Challenge for Cambridge Design Partnership where this program converts two different packets to a text output.

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 autogenate build files for another compilers just as easily if so desired. To use other compilers or build systems, you will need to run CMake manual.

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 should generate a makefile in the build folder, so next run make

```
make
```

5. If successful, this should 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 something like (assuming run from in the bin folder):

```
./pwr_and_batt_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 runs them.

Doxygen

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

`#include <packet_constants.h>`

Data Fields

- uint32_t time_stamp
- uint8_t batt_status
- uint16_t err_check

5.1.1 Detailed Description

Definition at line 56 of file packet_constants.h.

5.1.2 Field Documentation

5.1.2.1 batt_status

`uint8_t batt_packet::batt_status`

Definition at line 59 of file packet_constants.h.

5.1.2.2 err_check

`uint16_t batt_packet::err_check`

Definition at line 60 of file packet_constants.h.

5.1.2.3 time_stamp

```
uint32_t batt_packet::time_stamp
```

Definition at line 58 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

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

Data Fields

- uint32_t time_stamp
- uint32_t volts
- uint64_t milliamps
- uint16_t err_check
- uint64_t milliwatts

5.2.1 Detailed Description

Definition at line 47 of file packet_constants.h.

5.2.2 Field Documentation

5.2.2.1 err_check

```
uint16_t pwr_packet::err_check
```

Definition at line 52 of file packet_constants.h.

5.2.2.2 milliamps

```
uint64_t pwr_packet::milliamps
```

Definition at line 51 of file packet_constants.h.

5.2.2.3 milliwatts

```
uint64_t pwr_packet::milliwatts
```

Definition at line 53 of file packet_constants.h.

5.2.2.4 time_stamp

```
uint32_t pwr_packet::time_stamp
```

Definition at line 49 of file packet_constants.h.

5.2.2.5 volts

```
uint32_t pwr_packet::volts
```

Definition at line 50 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

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 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 Struct 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

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

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	Struct 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 Module for holding the functions used for calculating power.
00005  */
00006
00007 #ifndef CALCULATIONS_H
00008 #define CALCULATIONS_H
00009
00010 #include "packet_constants.h"
00011
00012 #include <stdint.h>
00013 #include <stddef.h>
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 int calc_power(pwr_packet_t * const pwr_numbers);
00020 uint32_t mod_of_array(uint8_t const * const p_array, size_t length);
00021
00022 #ifdef __cplusplus
00023 }
00024 #endif
00025
00026 #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 112 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!\n");
00095         ret_status = ENODATA;
00096     }
00097     else
00098     {
00099         printf("ERR: This requires only 1 input bin file\n");
00100         ret_status = ENFILE;
00101     }
00102     return ret_status;
00103 }
00104
00105 /**
00106  * @brief Get the pack from file object
00107  *
00108  * @param p_file Pointer to the file being processed
00109  * @param out_buff Outputing the data obtained from the file
00110  * @param size size of the buffer to output data in.
00111  */
00112 void get_pack_from_file(FILE ** p_file, uint8_t * out_buff, const size_t size)
00113 {
00114     for (size_t i = 0; size > i; ++i)
00115     {
00116         out_buff[i] = (uint8_t)fgetc(*p_file);
00117     }
```

```
00118 }
```

6.8 src/packet_constants.h File Reference

Header for information about packets like 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
- struct batt_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
- typedef struct batt_packet batt_packet_t

Enumerations

- enum packet_type_t { power_pack = 0 , battery_pack , error_type , num_of_types }

6.8.1 Detailed Description

Header for information about packets like 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
```

6.8.3.2 pwr_packet_t

```
typedef struct pwr_packet pwr_packet_t
```

6.8.4 Enumeration Type Documentation

6.8.4.1 packet_type_t

```
enum packet_type_t
```

Enumerator

power_pack	
battery_pack	
error_type	
num_of_types	

Definition at line 63 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 like 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 typedef struct pwr_packet
00048 {
00049     uint32_t time_stamp;
00050     uint32_t volts;
00051     uint64_t milliamps;
00052     uint16_t err_check;
00053     uint64_t milliwatts;
00054 } pwr_packet_t;
00055
00056 typedef struct batt_packet
00057 {
00058     uint32_t time_stamp;
00059     uint8_t batt_status;
00060     uint16_t err_check;
00061 } batt_packet_t;
00062
00063 typedef enum
00064 {
00065     power_pack = 0,
00066     battery_pack,
00067     error_type,
00068     num_of_types
00069 } packet_type_t;
00070
00071 #ifdef __cplusplus
00072 }
00073 #endif
00074
00075 #endif // PACKET_CONSTANTS_H
```

6.10 src/packet_parser.c File Reference

Module for parsing the incoming packets.

```
#include "packet_parser.h"
#include "calculations.h"
#include <stdbool.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.

Variables

- static uint32_t prev_pack_modulus = 0xEFu

6.10.1 Detailed Description

Module for parsing the incoming packets.

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 133 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 113 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 26 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 Struct with the battery information in it.

Definition at line 84 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 Struct with the power information in it

Definition at line 51 of file packet_parser.c.

6.10.3 Variable Documentation

6.10.3.1 prev_pack_modulus

```
uint32_t prev_pack_modulus = 0xEFu  [static]
```

Definition at line 12 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 Module for parsing the incoming packets
00005  */
00006
00007 #include "packet_parser.h"
00008 #include "calculations.h"
00009 #include <stdbool.h>
00010 #include <stdio.h>
00011
00012 static uint32_t prev_pack_modulus = 0xEFu;
00013
00014 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00015                                  const size_t length);
00016 int check_for_pack_error(const packet_type_t pack_type,
00017                          uint8_t const * const p_packet);
00018
00019 /**
00020  * @brief Function for determining the type of the incoming packet
00021  *
00022  * @param first_byte_of_pack 1st byte of the packet
00023  * @return packet_type_t Type of packet, will return the Enum error_type if it's
00024  *                       an invalid packet type.
00025  */
00026 packet_type_t determine_packet_type(const uint8_t first_byte_of_pack)
00027 {
00028     packet_type_t ret_type = error_type;
00029     if (PWR_PACKET_TYPE_BYTE == first_byte_of_pack)
00030     {
00031         ret_type = power_pack;
00032     }
00033     else if (BATT_PACKET_TYPE_BYTE == first_byte_of_pack)
00034     {
00035         ret_type = battery_pack;
00036     }
00037     else
00038     {
00039         // Error so do nothing
00040     }
00041
00042     return ret_type;
00043 }
00044
00045 /**
00046  * @brief Function for creating a power packet from an input buffer
00047  *
00048  * @param packet_buf Buffer to be converted
00049  * @return pwr_packet_t Struct with the power information in it
00050  */
00051 int process_pwr_packet(uint8_t const * const p_packet_buf,
00052                        pwr_packet_t * const p_out_pack)
00053 {
00054     int ret_status = -1;
00055     if((NULL != p_packet_buf) && (NULL != p_out_pack))
00056     {
00057         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00058                                                          TIMESTAMP_NUM_OF_BYTES);
00059
00060         p_out_pack->volts = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_VOLTS_LOC],
00061                                                     VOLTS_NUM_OF_BYTES);
00062
00063         p_out_pack->milliamps = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_MILLIAMP_LOC],
00064                                                         MILLIAMP_NUM_OF_BYTES);
00065
00066         p_out_pack->err_check = (uint16_t)p_packet_buf[PWR_START_OF_ERR_CHECK_LOC];
00067
00068         ret_status = calc_power(p_out_pack);
00069
00070         if(0 == ret_status)
00071         {
00072             ret_status = check_for_pack_error(power_pack, p_packet_buf);
00073         }
00074     }
00075     return ret_status;
00076 }
00077
00078 /**
00079  * @brief Function for creating a battery status packet from an input buffer
00080  *
00081  * @param packet_buf Buffer to be converted
00082  * @return batt_packet_t Struct with the battery information in it.
```

```
00083  */
00084 int process_batt_packet(uint8_t const * const p_packet_buf,
00085                         batt_packet_t * const p_out_pack)
00086 {
00087     int ret_status = -1;
00088 
00089     if((NULL != p_packet_buf) && (NULL != p_out_pack))
00090     {
00091         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00092                                                          TIMESTAMP_NUM_OF_BYTES);
00093 
00094         p_out_pack->batt_status = p_packet_buf[BATT_START_OF_BATT_STATUS_LOC];
00095 
00096         p_out_pack->err_check = (uint16_t)p_packet_buf[BATT_START_OF_ERR_CHECK_LOC];
00097 
00098         ret_status = check_for_pack_error(battery_pack, p_packet_buf);
00099     }
00100 
00101     return ret_status;
00102 }
00103 
00104 /**
00105  * @brief Function for converting an uint8_t array to an uint32_t
00106  *
00107  * @todo assert that the length is <=4 bytes
00108  *
00109  * @param p_array Array to be converted
00110  * @param length length of the array in bytes
00111  * @return uint32_t
00112  */
00113 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00114                                  const size_t length)
00115 {
00116     uint32_t ret_val = 0;
00117     for (size_t i = 0u; length > i; ++i)
00118     {
00119         ret_val «= 8u;
00120         ret_val += p_array[i];
00121     }
00122 
00123     return ret_val;
00124 }
00125 
00126 /**
00127  * @brief Helper function to check the error byte for integraty
00128  *
00129  * @param pack_type Type of packet being checked
00130  * @param p_packet pointer to the packet buffer
00131  * @return int Success = 0
00132  */
00133 int check_for_pack_error(const packet_type_t pack_type,
00134                          uint8_t const * const p_packet)
00135 {
00136     int ret_status = 0;
00137     uint32_t mod_of_pack = 0;
00138     bool save_modulus = true;
00139     switch (pack_type)
00140     {
00141     case power_pack:
00142         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_PWR_PACK - ERR_CHECK_NUM_OF_BYTES));
00143         if (mod_of_pack != p_packet[PWR_START_OF_ERR_CHECK_LOC])
00144         {
00145             ret_status = -1;
00146             printf("ERR: Packet Failed Error Check\n");
00147         }
00148         break;
00149     case battery_pack:
00150         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_BATT_PACK - ERR_CHECK_NUM_OF_BYTES));
00151         if (mod_of_pack != p_packet[BATT_START_OF_ERR_CHECK_LOC])
00152         {
00153             ret_status = -1;
00154             printf("ERR: Packet Failed Error Check\n");
00155         }
00156         break;
00157     default:
00158         // Should never be able to call this but should guard for it just in case
00159         save_modulus = false;
00160         break;
00161     }
00162 
00163     if(save_modulus)
00164     {
00165         prev_pack_modulus = mod_of_pack;
00166     }
00167 
00168     return ret_status;
00169 }
```

6.12 src/packet_parser.h File Reference

Module for parsing the incoming packets.

```
#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

Module for parsing the incoming packets.

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 26 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 Struct with the battery information in it.

Definition at line 84 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 Struct with the power information in it

Definition at line 51 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 Module for parsing the incoming packets
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

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)

  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

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	
STATE_1	
STATE_2	
STATE_3	
NUM_OF_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 109 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 121 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 43 of file state_handler.c.

6.14.4.4 set_initial_timestamp()

```
void set_initial_timestamp (
            const uint32_t ts )
```

Set the initial timestamp object.

Parameters

ts	

Definition at line 156 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 172 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 27 of file state_handler.c.

6.14.5.2 inital_ts_ms

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

Definition at line 33 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 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     STATE_0 = 0,
00021     STATE_1,
00022     STATE_2,
00023     STATE_3,
00024     NUM_OF_STATES
00025 } states_t;
00026
00027 const char * batt_states[4] = {"VLOW", "LOW", "MED", "HIGH"};
00028 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms);
00029 states_t determine_state(pwr_packet_t const * const pwr);
00030 void set_initial_timestamp(const uint32_t ts);
00031 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms);
00032
00033 static uint32_t inital_ts_ms = 0;
```

```
00034
00035 /**
00036  * @brief Function for processing the states of either a power or battery state
00037  *
00038  * @param pack_type Type of packet being processed
00039  * @param pwr Pointer to the struct that holds the power information
00040  * @param batt Pointer to the struct that holds the battery information
00041  * @return int 0 == Success
00042  */
00043 int process_state_and_transitions(packet_type_t pack_type,
00044                                   pwr_packet_t const * const pwr,
00045                                   batt_packet_t const * const batt)
00046 {
00047     int ret_status = 0;
00048     static uint32_t prev_ts_ms = 0;
00049     static states_t prev_state = STATE_0;
00050     static bool init_intial_ts = true;
00051     if ((NULL == pwr) || (NULL == batt))
00052     {
00053         ret_status = -1;
00054     }
00055     else
00056     {
00057         switch (pack_type)
00058         {
00059             case power_pack:
00060             {
00061                 if (init_intial_ts)
00062                 {
00063                     set_initial_timestamp(pwr->time_stamp);
00064                     init_intial_ts = false;
00065                 }
00066                 states_t current_state = determine_state(pwr);
00067                 if (current_state != prev_state)
00068                 {
00069                     if(time_check(prev_ts_ms, pwr->time_stamp))
00070                     {
00071                         printf("S;%u;%u-%u\n",
00072                                calc_time_from_start_ms_to_sec(pwr->time_stamp),
00073                                prev_state,
00074                                current_state);
00075                     }
00076                 }
00077                 prev_state = current_state;
00078                 break;
00079             }
00080             case battery_pack:
00081             {
00082                 if (init_intial_ts)
00083                 {
00084                     set_initial_timestamp(batt->time_stamp);
00085                     init_intial_ts = false;
00086                 }
00087                 printf("B;%u;%s\n",
00088                        calc_time_from_start_ms_to_sec(batt->time_stamp),
00089                        batt_states[batt->batt_status]);
00090                 break;
00091             }
00092             default:
00093             {
00094                 ret_status = -1;
00095                 break;
00096             }
00097         }
00098     }
00099
00100     return ret_status;
00101 }
00102
00103 /**
00104  * @brief Function for calculating the time since the start of the program
00105  *
00106  * @param current_ts_ms Current timestamp in milliseconds
00107  * @return uint32_t Time from start of program in seconds
00108  */
00109 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms)
00110 {
00111     uint32_t ts_ms = (current_ts_ms - inital_ts_ms);
00112     return (ts_ms / 1000);
00113 }
00114
00115 /**
00116  * @brief Function for determining the state from power
00117  *
00118  * @param pwr Pointer to the struct that holds the power information
00119  * @return states_t Current state, returns NUM_OF_STATES if invalid
00120  */
```

```
00121 states_t determine_state(pwr_packet_t const * const pwr)
00122 {
00123     states_t ret_state = NUM_OF_STATES;
00124     if (STATE_0_UPPER_BOUNDS >= pwr->milliwatts)
00125     {
00126         ret_state = STATE_0;
00127     }
00128     else if ((STATE_1_LOWER_BOUNDS <= pwr->milliwatts)
00129             && (STATE_1_UPPER_BOUNDS >= pwr->milliwatts))
00130     {
00131         ret_state = STATE_1;
00132     }
00133     else if ((STATE_2_LOWER_BOUNDS <= pwr->milliwatts)
00134             && (STATE_2_UPPER_BOUNDS >= pwr->milliwatts))
00135     {
00136         ret_state = STATE_2;
00137     }
00138     else if ((STATE_3_LOWER_BOUNDS <= pwr->milliwatts)
00139             && (STATE_3_UPPER_BOUNDS >= pwr->milliwatts))
00140     {
00141         ret_state = STATE_3;
00142     }
00143     else
00144     {
00145         //do nothing
00146     }
00147
00148     return ret_state;
00149 }
00150
00151 /**
00152  * @brief Set the initial timestamp object
00153  *
00154  * @param ts
00155  */
00156 void set_initial_timestamp(const uint32_t ts)
00157 {
00158     inital_ts_ms = ts;
00159 }
00160
00161 /**
00162  * @brief Function to check if 2 timestamps are greater than 10 ms apart
00163  *
00164  * @todo There is probably a better way to handle an uint32_t overflow or
00165  *       underflow situation.
00166  *
00167  * @param prev_ts_ms Previous timestamp
00168  * @param current_ts_ms Current timestamp
00169  * @return true If (current - prev) > 10ms
00170  * @return false If (current - prev) < 10ms
00171  */
00172 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms)
00173 {
00174     bool greater_than_10 = false;
00175
00176     if (prev_ts_ms < current_ts_ms)
00177     {
00178         if (10u < (current_ts_ms - prev_ts_ms))
00179         {
00180             greater_than_10 = true;
00181         }
00182     }
00183
00184     return greater_than_10;
00185 }
```

6.16 src/state_handler.h File Reference

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

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 43 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 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