BioTac® C Library Manual for Cheetah

Version 1.1.0

Chia-Hsien (Gary) Lin
Tomonori Yamamoto
Jeremy Fishel

April 5, 2012

Table of Contents

1 Introduction

This document describes how to get BioTac data through a Cheetah SPI Host Adapter (Total Phase, Inc.). Some functions therefore use Cheetah API functions. Please refer to Cheetah SPI Host Adapter Datasheet for more details.

BioTac C Library for Cheetah is supported on Linux, Mac OS X, and Windows operating systems. The current Windows version has some restrictions (discussed in Windows section), and we recommend that a user test this on Linux (or Mac OS X).

2 General Data Types

The Cheetah API provides the following data types:

```
typedef unsigned char           u08;
typedef unsigned short          u16;
typedef unsigned int            u32;
typedef unsigned long long      u64;
typedef signed char             s08;
typedef signed short            s16;
typedef signed int              s32;
typedef signed long long        s64;
```

Along with these definitions, BioTac C Library for Cheetah provides the following data types:

```
typedef int                     BioTac;
typedef int                     BOOL;
```

as well as structures related to BioTac info (`bt_info`), properties (`bt_property`), and data (`bt_data`) as illustrated in Figure 1.

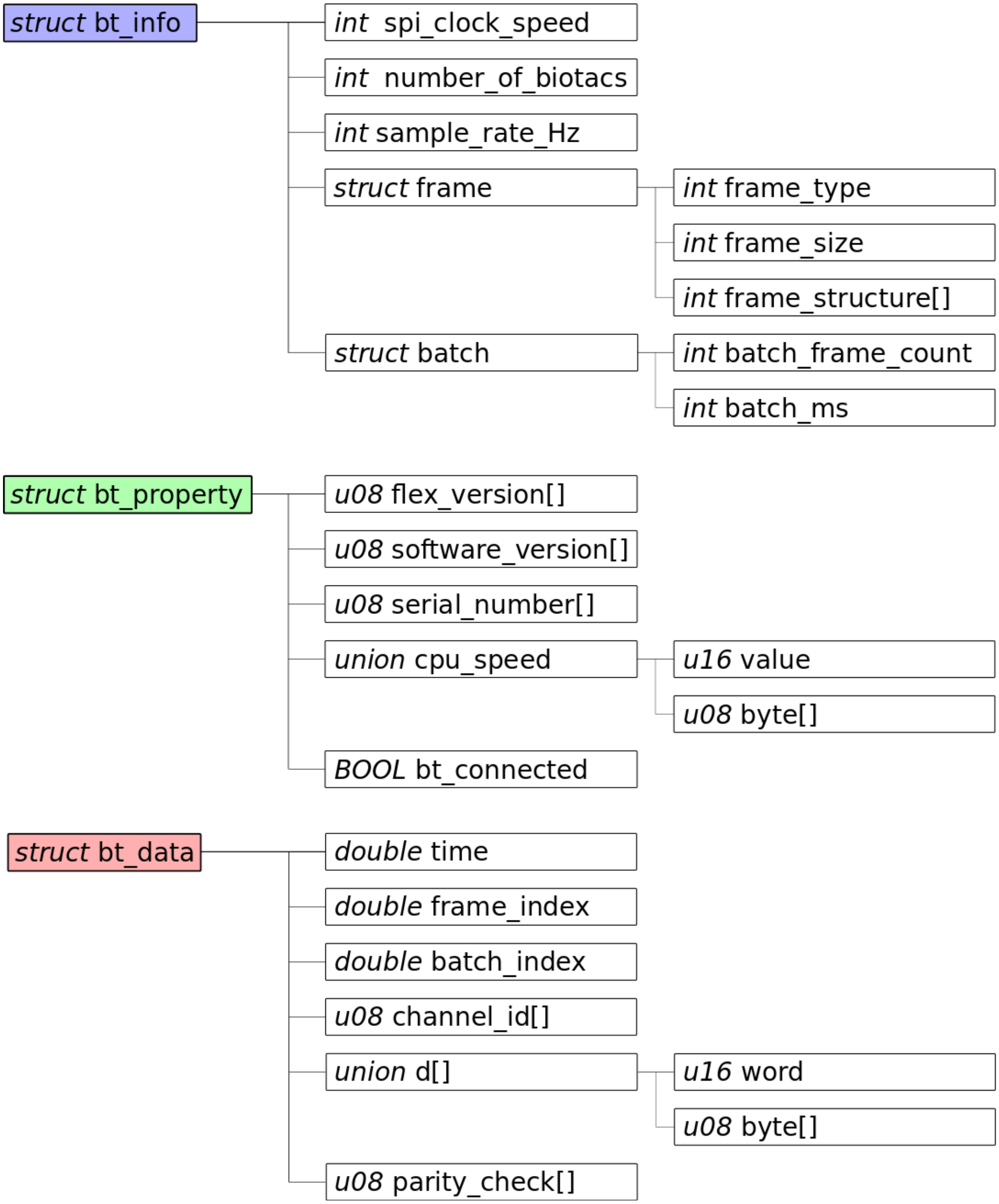

Figure 1: Structures and their member variables of BioTac info, properties, and data.

3 Functions

Initialize Cheetah SPI Device (bt_cheetah_initialize)

```
Cheetah bt_cheetah_initialize (const bt_info *biotac);
```

Open a connection to the Cheetah device and return a Cheetah handle to check errors.

Arguments

`biotac`	pointer to a BioTac information structure

Return Value

This function returns a Cheetah handle, which is guaranteed to be greater than zero if valid.

Specific Error Codes

`CH_UNABLE_TO_OPEN`

The specified port is not connected to a Cheetah device or the port is already in use.

`CH_INCOMPATIBLE_DEVICE`

There is a version mismatch between the DLL and the hardware. The DLL is not of a sufficient version for interoperability with the hardware version or vice versa. See `ch_open_ext()` in the Cheetah Datasheet for more information.

Details

This function may be used to initialize a Cheetah device to communicate with a BioTac. It configures SPI subsystems, sets a clock speed, and clears up the batch queue.

Get BioTac Properties (bt_cheetah_get_properties)

```
BioTac bt_cheetah_get_properties (  Cheetah     ch_handle,
                                    int         bt_select,
                                    bt_property *property);
```

Receive and print the properties of a selected BioTac.

Arguments

`ch_handle`	handle of a Cheetah adapter
`bt_select`	switch to choose a specific BioTac (i.e., 1, 2, …)
`property`	pointer to a BioTac property structure

Return Value

This function returns a BioTac error code, which is zero if valid.

Specific Error Codes

`BT_WRONG_NUMBER_ASSIGNED`

The assigned number exceeds the range, and there is no BioTac assigned to the number.

`BT_ERROR_UNKNOWN_COMMAND`

No such command exists. Current version of the BioTac firmware only supports six kinds of properties: Flex version, Firmware version, Serial Number, CPU speed, Sampling speed, and Sampling Pattern.

Details

This function receives and prints BioTac properties such flex circuit version, firmware version, serial number, and CPU speed, and stores those property data in `property` (the third argument of the function).

Configure batch command (bt_cheetah_configure_batch)

```
BioTac bt_cheetah_configure_batch ( Cheetah     ch_handle,
                                    bt_info     *biotac,
                                    int         num_samples);
```

Configure a batch of commands.

Arguments

`ch_handle`	handle of a Cheetah adapter
`biotac`	pointer to a BioTac information structure
`num_samples`	total number of samples

Return Value

This function returns a BioTac error code, which is zero if valid.

Specific Error Codes

`BT_DATA_SIZE_TOO_SMALL`

A value assigned to length_of_data_in_second is too small. The value should not be smaller than batch_ms that is 50ms by default.

Details

This function creates a batch of commands that is constructed by a repeated sampling frame. The function uses `batch_frame_count`, a member variable of `bt_info`, to setup the number of frames in a batch. A sampling frame is composed sequentially in the order addressed by `frame_structure[]`. By default, the following 44 sampling commands generate the frame structure:

(`BT_FRAME_STRUCTURE_DEFAULT`): E1, Pac, E2, Pac, E3, Pac, E4, Pac, E5, Pac, E6, Pac, E7, Pac, E8, Pac, E9, Pac, E10, Pac, E11, Pac, E12, Pac, E13, Pac, E14, Pac, E15, Pac, E16, Pac, E17, Pac, E18, Pac, E19, Pac, Pdc, Pac, Tac, Pac, Tdc, and Pac.

Configure save-buffer (bt_configure_save_buffer)

```
bt_data* bt_configure_save_buffer (int num_samples);
```

Configure a buffer to save data in a file.

Arguments

`num_samples`	total number of samples

Return Value

This function returns a pointer to a BioTac data structure.

Specific Error Codes

None.

Details

This function allocates memory for a buffer to store BioTac data and write them in a file later. The function uses the number of samples, which is calculated from `length_of_data_in_second`. The function has to be called when the save buffer size is changed.

Collect Data (bt_cheetah_collect_batch)

```
void bt_cheetah_collect_batch ( Cheetah         ch_handle,
                                const bt_info   *biotac,
                                bt_data         *data,
                                BOOL            print_flag);
```

Collect BioTac sampling data in batch unit.

Arguments

`ch_handle`	handle of a Cheetah adapter
`biotac`	pointer to a BioTac information structure
`data`	pointer to data array for storing BioTac sampling data
`print_flag`	flag for printing data on Terminal (Linux) or Command Prompt (Windows)

Return Value

This function does not return any value, but stores data in the `bt_data` array.

Specific Error Codes

None.

Details

This function collects a batch of data from a Cheetah SPI device and submits a batch of commands that is configured in the `bt_cheetah_get_properties` function.

By default, a batch contains 5 frames of raw data (ranging from 0 to 4095) and these data will be decoded in this function. If the result of parity check is wrong, `bt_parity` will be set to 1.

Currently, elapsed time is set to zero for all data by default (`DEFAULT_TIMER`). If the user is willing to use a timer provided by your computer, `MACHINE_TIMER` could be enabled (located at the beginning of `biotac.c`). This feature is currently experimental.

Note: Cheetah SPI device controls the sampling time according to the commands in the memory. By default, the interval timing for each sample of channel is 227µs periodically. However, collecting data would occur only after the Cheetah SPI device has finished collecting a batch of data, which is equal to 50ms of data.

Print Error (bt_display_errors)

```
    void    bt_display_errors (BioTac bt_err_code);
```

Print an error message on the display based on the error code collected in `bt_cheetah_setup` *function.*

Arguments

`bt_err_code` variable to store BioTac error information

Return Value

This function does not return any value.

Specific Error Codes

None.

Details

This function prints out "Wrong BioTac number assigned!" if wrong `bt_select number` is assigned to the `bt_cheetah_get_properties` function. For `BT_MAX_BIOTACS_PER_CHEETAH = 3`, this value should be integer 1, 2, or 3. For `BT_MAX_BIOTACS_PER_CHEETAH = 5`, this value should be integer 1, 2, 3, 4, or 5.

This function prints out "No BioTac detected!" if no BioTac is detected. A physical connection error or wrong timing of SPI sampling commands will cause this error.

This function prints out "Wrong maximum number of BioTacs assigned (should be 3 or 5)!" if `BT_MAX_BIOTACS_PER_CHEETAH` is not 3 or 5. The assigned value should be 3 or 5 only.

This function prints out "The number of samples is too small!" if variable `length_of_data_in_second` is too small. It should be greater than 50ms.

This function prints out "Something wrong occurred!" if an unknown error has occurred.

Save data (bt_save_data)

```
    void    bt_save_data (  const char      *file_name,
                            const bt_data   *data,
                            int             num_samples);
```

Save BioTac data, including batch, frame, and channel ids, raw values, parity check, etc.

Arguments

`file_name`	pointer of the string of file name such as "output.txt"
`data`	pointer to access the BioTac data
`num_samples`	numbers of sampling data saved into the file

Return Value

This function does not return any value.

Specific Error Codes

None.

Details

This function saves data in a text file whose name is `file_name`. By default, a saved data format is as follows (when `MAX_BIOTACS_PER_CHEETAH = 3`):

```
time, batch_index, frame_index, channel_id, value[0], bt_parity[0],
value[1], bt_parity[1], value[2], bt_parity[2]
```

Note that `channel_id` is saved as integer (0-3, 15-35) by default, or can be string (PAC, PDC, TAC, TDC, HAL, E01-E19) as an option. The string name is declared as `command_name[]` in `biotac.c`.

Close Cheetah Device (bt_cheetah_close)

```
void    bt_cheetah_close (Cheetah ch_handle);
```

Disable the SPI output and close the Cheetah Device.

Arguments

`ch_handle`	handle of a Cheetah adapter

Return Value

This function does not return any value.

Specific Error Codes

None.

Details

Disable the outputs. Close the connection of the Cheetah given by the handle.

4 Error Codes

Literal Name	Value
`BT_OK`	0
`BT_WRONG_NUMBER_ASSIGNED`	-1
`BT_NO_BIOTAC_DETECTED`	-2
`BT_WRONG_MAX_BIOTAC_NUMBER`	-3
`BT_DATA_SIZE_TOO_SMALL`	-4
`BT_ERROR_UNKNOWN_COMMAND`	-5

5 Code Example

The code example is compatible with Linux (recommended), Mac OS X, and Windows. **Before testing the code, we highly recommend the user try some code examples provided for a Cheetah SPI adapter. Make sure if you have downloaded an appropriate USB driver for your operating system and placed an appropriate Dynamic Linked Library (`cheetah.so` for Linux and Mac OS X and `cheetah.dll` for Windows) in a designated directory.** Without being able to run Total Phase's `detect` program, our code example will not run properly.

The cross-platform example package contains seven files:

- `biotac.c`
- `biotac.h`
- `cheetah.c`
- `cheetah.h`
- `example.c`
- `Makefile`
- `Makefile-32bit`

It is anticipated that the user only modifies `example.c`. SynTouch LLC does not provide official support for any modifications on `biotac.h`, `biotac.c`, `cheetah.h`, or `cheetah.c`. Therefore, only `example.c` is explained below.

```
int main(void)
{
	/*********************/
	/* --- Define variables --- */
	/*********************/
	bt_info biotac;
	bt_property biotac_property[MAX_BIOTACS_PER_CHEETAH];
	bt_data *data;
	BioTac bt_err_code;
	Cheetah ch_handle;

	int i;
	int length_of_data_in_second;
	int number_of_samples;
	int number_of_loops;

	/*******************************************************/
	/* --- Initialize BioTac settings (only default values are supported) --- */
	/*******************************************************/
	biotac.spi_clock_speed = BT_SPI_BITRATE_KHZ_DEFAULT;
	biotac.number_of_biotacs = 0;
	biotac.sample_rate_Hz = BT_SAMPLE_RATE_HZ_DEFAULT;
	biotac.frame.frame_type = 0;
	biotac.batch.batch_frame_count = BT_FRAMES_IN_BATCH_DEFAULT;
	biotac.batch.batch_ms = BT_BATCH_MS_DEFAULT;
```

```

// Set the duration of the run time
length_of_data_in_second = 3;
number_of_samples = (int)(BT_SAMPLE_RATE_HZ_DEFAULT * length_of_data_in_second);

// Check if any initial settings are wrong
if (MAX_BIOTACS_PER_CHEETAH != 3 && MAX_BIOTACS_PER_CHEETAH != 5)
{
    bt_err_code = BT_WRONG_MAX_BIOTAC_NUMBER;
    bt_display_errors(bt_err_code);
    exit(1);
}

/********************************/
/* --- Initialize the Cheetah devices --- */
/********************************/
ch_handle = bt_cheetah_initialize(&biotac);

/********************************************/
/* --- Get and print out properties of the BioTac(s) --- */
/********************************************/
for (i = 0; i < MAX_BIOTACS_PER_CHEETAH; i++)
{
    bt_err_code = bt_cheetah_get_properties(ch_handle, i+1, &(biotac_property[i]));

    if (biotac_property[i].bt_connected == YES)
    {
        (biotac.number_of_biotacs)++;
    }

    if (bt_err_code)
    {
        bt_display_errors(bt_err_code);
        exit(1);
    }
}

// Check if any BioTacs are detected
if (biotac.number_of_biotacs == 0)
{
    bt_err_code = BT_NO_BIOTAC_DETECTED;
    bt_display_errors(bt_err_code);
    return bt_err_code;
}
else
{
    printf("\n%d BioTac(s) detected.\n\n", biotac.number_of_biotacs);
}

// The programs stops here until it accepts [Enter] key hit
printf("Press [Enter] to continue ...");
fflush(stdout);
getchar();
```

```
120
121         /*****************************/
122         /* --- Configure the save buffer --- * /
123         /*****************************/
124         data = bt_configure_save_buffer(number_of_samples);
125
126         /************************/
127         /* --- Configure the batch --- */
128         /************************/
129         bt_err_code = bt_cheetah_configure_batch(ch_handle, &biotac, number_of_samples);
130         if (bt_err_code < 0)
131         {
132             bt_display_errors(bt_err_code);
133             exit(1);
134         }
135         else
136         {
137             printf("\nConfigured the batch\n");
138         }
139
140         /************************************************/
141         /* --- Collect the batch and display the data (if desired) --- */
142         /************************************************/
143         number_of_loops = (int)(number_of_samples / ((double)(biotac.frame.frame_size *
biotac.batch.batch_frame_count)));
144         printf("Start collecting BioTac data for %d second(s)...\n", length_of_data_in_second);
145         for (i = 0; i < number_of_loops; i++)
146         {
147             // To print out data on Terminal, set the fourth argument to YES (NO by default)
148             bt_cheetah_collect_batch(ch_handle, &biotac, data, NO);
149         }
150
151         /****************/
152         /* --- Save data --- */
153         /****************/
154         bt_save_buffer_data("output.txt", data, number_of_samples);
155
156         /*******************/
157         /* --- Close and exit --- */
158         /*******************/
159         printf("Press [Enter] to close the program");
160         fflush(stdout);
161         getchar();
162
163         free(data);
164         bt_cheetah_close(ch_handle);
165
166         return 0;
167     }
```

The `main` function for the example starts from Line 42. First, several variables are declared (Lines 47-56). Member variables of `bt_info biotac` are initialized (Lines

61-66). We suggest the user use the default values. On Lines 69-70, the user can change the duration of the run time. The Cheetah initialization function is called on Line 83. In the current version of software, if more than one Cheetah devices are hooked up to a computer, the device opening through port 0 will be detected and other devices will be ignored. Next, properties of the BioTacs such as flex circuit version, software version, serial number, and CPU speed and the number of connected BioTacs are displayed on the display (Lines 88-102). If no BioTac is detected, the program exits (Lines 105-114). The program waits before configuring the batch until it accepts a user input of [Enter] key from a keyboard (Lines 117-119).

After the [Enter] key hit, we allocate memory for a buffer to save data (Line 124), followed by a function to configure batch settings (Lines 129-138). If the user is willing to try the `MACHINE_TIMER` option, the timer will start here. Thus, pausing after the `bt_cheetah_configure_batch` function before stopping collecting data is likely to cause an error.

Next, we start collecting the batch of data (Lines 143-149). `number_of_loops` can be calculated based on `number_of_samples`, `frame_size`, and `batch_frame_count`. During data collection, if the user would like to display real-time data on Terminal, the fourth argument of `bt_cheetah_collect_batch` can be changed from `NO` to `YES`. However, keep in mind that printing data on Terminal makes the program running slower and thus the program might fail to keep up with a specified clock. Instead, you can save data to a text file using the `bt_save_data` function (Lines 154).

Finally, we close the program by calling `bt_cheetah_close` and free allocated memory.

OS Specific Instructions and Known Issues

Linux

We have tested our code on Ubuntu 10.04 (32-bit and 64-bit). `Makefile` is written for a 64-bit kernel. For 32-bit, please delete `Makefile` (or rename it to `Makefile-64bit`) and rename `Makefile-32bit` to `Makefile`. To run the example program, fist launch Terminal, move to an appropriate directory, build the project, and run the executable file:

```
>> cd /path/to/code_example/
>> make
>> ./bin/example
```

Make sure if you copy an appropriate version of `cheetah.so` in a designated folder (i.e. where your source files are).

Mac OS X

We have tested our code on Mac OS X 10.7 (64-bit). First, make sure if XCode 4 or higher is installed on your machine to run make. Then, follow the instructions for Linux.

Windows

We have tested our code using Microsoft Visual C++ 2010 Express on Windows 7 Professional (64-bit). First, create a new project (`File` -> `New` -> `Project...`) and choose `Empty Project` from Visual C++ template. Enter names of the project/solution and set the location, and click OK. After successfully creating the new project, right-click on `Header Files` on the left menu, choose `Add` -> `Existing item...` to import `biotac.h` and `cheetah.h`. Similarly, right-click on `Sources Files` and choose `Add` -> `Existing item...` to import `biotac.c`, `cheetah.c`, and `example.c`. Make sure if you copy an appropriate version of `cheetah.dll` in a designated folder. In our case, we downloaded the 32-bit version of `cheetah.dll` and copied it to `C:\Windows\System32` because Microsoft Visual C++ Express 2010 runs under the 32-bit mode.

We have observed that when `print_flag`, the fourth argument of the `bt_cheetah_collect_batch` function, is set to `YES`, only the first batch collects BioTac data. Thus, it is highly recommended that Windows users turn off the print display feature. A member variable `parity_check` in `bt_data` cab be a quick check if collected data are corrupted.

6 Revision History

(V1.1.0) April 5, 2012

- Compatible with BioTac firmware 2.x.x and 3.x.x
- New function `bt_configure_save_buffer` created
- Due to structure change, `bt_property` is not a member variable of `bt_info`

(V1.0.0) December 22, 2011

Original document created