BioTac® C Library Manual for RT-Socket-CAN

Version 0.1.0

Tomonori Yamamoto
Chia-Hsien (Gary) Lin
Jeremy Fishel

April 5, 2012

Table of Contents

1 Introduction 3
2 General Data Types 4
3 Functions 5
Initialize RT-Socket-CAN (bt_rtcan_initialize) 5
Configure save-buffer (bt_configure_save_buffer) 5
Listen to BioTac sample (bt_rtcan_listen) 6
Print Error (bt_display_errors) 6
Save data (bt_save_buffer_data) 7
4 Code Example 8
5 Revision History 12

1 Introduction

This document describes how to acquire BioTac data through a PCAN-PCI[1] card (PEAK-System Technik) on a real-time Linux kernel. We specifically use RT-Socket-CAN functions provided by Xenomai. Please refer to the following links for more details regarding RT-Socket-CAN:

- http://www.xenomai.org/documentation/trunk/html/api/rtcan_8h.html
- http://svn.gna.org/svn/xenomai/trunk/ksrc/drivers/can/README
- http://svn.gna.org/svn/xenomai/trunk/src/utils/can/README

Therefore, BioTac C Library for RT-Socket-CAN is supported only on a Linux operating system with Xenomai. We have tested our code with Ubuntu 10.04 64-bit and Xenomai 2.5.5.2, which we followed at: http://barrett.com:8080/wiki/BuildingAPC/Lucid64Install

[1] The Xenomai Peak CAN driver does not seem to support PEAK PCAN-PCI Express (PCIe) card for Xenomai 2.5.5.2. With the PCIe board, you may want to use a newer kernel version of Xenomai or rebuild the Xenomai kernel space with the modified source code: http://svn.berlios.de/wsvn/socketcan/trunk/kernel/2.6/drivers/net/can/sja1000/peak_pci.c?op=diff&rev=1182&peg=1182

2 General Data Types

BioTac C Library for RT-Socket-CAN 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;

typedef int                     BioTac;
typedef int                     BOOL;
```

as well as structures related to BioTac data (bt_data) as illustrated in Figure 1.

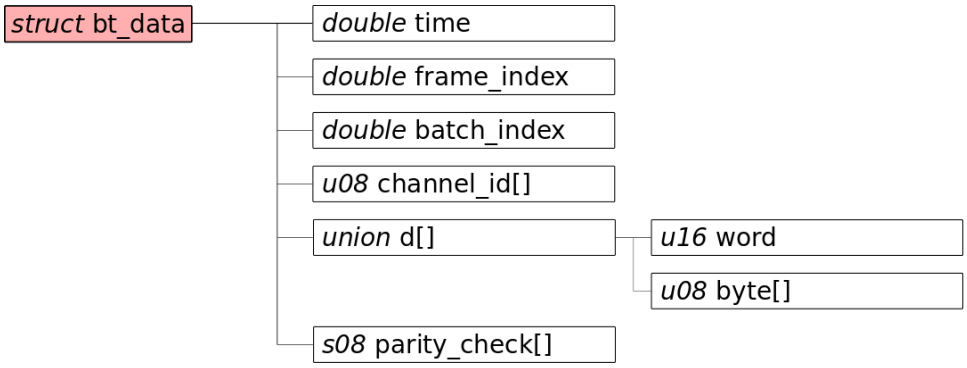

Figure 1: Structures and their member variables of BioTac info, properties, and data.

3 Functions

Initialize RT-Socket-CAN (bt_rtcan_initialize)

```
BioTac rtcan_initialize (int argc, char **argv);
```

Initialize the settings related to RT-Socket-CAN.

Arguments

`argc`	argument count (same as `argc` for the `main` function)
`argv`	argument vector (same as `argv` for the `main` function)

Return Value

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

Specific Error Codes

`BT_RTCAN_WRONG_ARGUMENT`

To run the example program, only one argument, such as `rtcan0` or `rtcan1`, is required to specify what port of the CAN board the BioTacs are connected to. You can check the activated port by typing on Terminal:
>> $ `cat /proc/rtcan/devices`

Details

This function may be used to open communicate with the BioTacs through RT-Socket-CAN. This function originates mostly from `rtcanrecv.c`.

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.

Listen to BioTac sample (bt_rtcan_listen)

```
void bt_rtcan_listen (  bt_data         *data,
                        int             num_samples,
                        BOOL            print_flag);
```

Listen to BioTac samples and collect data.

Arguments

`data`	pointer to data array for storing BioTac sampling data
`num_samples`	total number of samples
`print_flag`	flag for printing data on Terminal

Return Value

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

Specific Error Codes

None.

Details

This function receives BioTac samples and collects data through RT-Socket-CAN. Signals from the BioTac will be decoded and assigned as raw data (ranging from 0 to 4095) in this function.

For CANbus communication, a batch is not necessary, but to be consistent with other API functions provided by SynTouch, batch index number is also stored, which is just an index number. Note that parity check may be redundant for CANbus because the CAN procotol has CRC16 check.

To save data in a file, there are a few options to record time stamps: no time stamps, elapsed time, or absolute time. Elapsed time is the default setting, but it can be changed in `rt_can.c`.

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.

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 maximum number of BioTacs assigned (should be 3)!" if `BT_MAX_BIOTACS_PER_CHEETAH` is not 3. For BH8-280, it should be 3.

This function prints out "Please add either 'rtcan0' or 'rtcan1' as the argument" if the argument to run the program is none or more than 1.

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

Save data (bt_save_buffer_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:

`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 `bt_rtcan.c`.

4 Code Example

This section explains how to use the provided codes to acquire BioTac data through RT-Socket-CAN. As mentioned earlier, our environment has been only tested with Ubuntu 10.04 64-bit and Xenomai 2.5.5.2 using a PCAN-PCI card. We specifically use RT-Socket-CAN API functions provided by Xenomai. It seems installing the PCAN driver overrides the Xenomai PEAK CAN driver, so it is not recommended to install the PCAN driver (or remove the module using `rmmod`).

The code example package contains seven files and two hidden directories:

- rt_can.c
- bt_rtcan.c
- bt_rtcan_example.c
- bt_rtcan.h
- Makefile
- Makefile.am
- Makefile.in
- .deps
- .libs

Here are the steps to run the example program `bt_rtcan`:

1. First, make a directory and backup the whole `can` directory of the Xenomai user space. In our environment, it is `/usr/src/xenomai-2.5.5.2/src/utils/can`.

```
>> $ cd ~/
>> $ mkdir bt_tmp && cd bt_tmp
>> $ cp –r /usr/src/xenomai-2.5.5.2/src/utils/can ./can_backup
```

2. Next, uncompress the `bt_rtcan` package (assuming you save it under `Downloads` directory) and copy the files to the `can` directory. Note that some of the files will be overridden in the `can` directory, and that is the reason you may want to backup the original files in the first step.

```
>> $ tar xvfz ~/Downloads/bt_rtcan.tar.gz
>> $ cp bt_rtcan/* /usr/src/xenomai-2.5.5.2/src/utils/can/
```

3. To create the example program provided by SynTouch without modification, move to the `can` directory of the Xenomai user space and build, which generates a program called `bt_rtcan`.

```
>> $ cd /usr/src/xenomai-2.5.5.2/src/utils/can/
>> $ sudo make install
```

If you want to create your own program, please read the following notes.

[Warning] In the future, you may want to change `Makefile` (plus `Makefile.in` and `Makefile.am`) to build your own program. Then, you will probably need to regenerate all `Makefile` files located in the Xenomai user space. You will not have to rebuild the whole Xenomai user space, but still need to regenerate `Makefile` files to make changes in the `can` directory. In this case, you need to take a few more steps instead of just running `sudo make install`:

```
>> $ cd /usr/src/xenomai-2.5.5.2
>> $ autoreconf –fiv
>> $ sudo ./configure
>> $ cd /usr/src/xenomai-2.5.5.2/src/utils/can/
>> $ sudo make install
```

Note that these commands themselves will **NOT** rebuild the original executable programs in the Xenomai user space. Also, note that the original executable programs are copied under `/usr/xenomai/sbin` and `/usr/xenomai/bin` directories.

4. To run the program, you need to specify what port the BioTacs are connected. Make sure if it has been activated. In the following example, we assume neither ports have been activated, and then you activate `rtcan0`, assuming the BioTacs are connected to `rtcan0`. As soon as the port is activated, you should see the accumulated size of the incoming messages at RX_Counter in `rtcan0`. If you have already activated a port, you can skip this step, but make sure if you see that RT_Counter keeps increasing.

```
>> $ cat /proc/rtcan/devices
Name_____________ _Baudrate State___ TX_Counter RX_Counter ____Errors
rtcan0          undefined stopped          0          0          0
rtcan1          undefined stopped          0          0          0

>> $ sudo /usr/xenomai/sbin/rtcanconfig rtcan0 -b 1000000 -c none start

>> $ cat /proc/rtcan/devices
Name_____________ _Baudrate State___ TX_Counter RX_Counter ____Errors
rtcan0           1000000 active           0       XXXX          0
rtcan1          undefined stopped          0          0          0
```

5. Finally, you are ready to run the program. Without modifying the example code, it runs for 3 seconds (without printing data on Terminal), saves data in `output.txt`, and exits.

```
>> $ sudo ./bt_rtcan rtcan0
```

Next, `bt_rtcan_example.c` is explained below:

```
41  int main(int argc, char **argv)
42  {
43      /*****************************/
44      /* --- Define variables --- */
45      /*****************************/
46      bt_data *data;
47      BioTac bt_err_code;
48
49      int length_of_data_in_second = 3;
50      int number_of_samples = (int)(BT_SAMPLE_RATE_HZ_DEFAULT *
length_of_data_in_second);
51
52  /*****************************************************************/
53  /* Initialize BioTac settings (only default values are supported) */
54  /*****************************************************************/
55      // Check if any initial settings are wrong
56      if (MAX_BIOTACS_FOR_BARRETT != 3)
57      {
58          bt_err_code = BT_WRONG_MAX_BIOTAC_NUMBER;
59          bt_display_errors(bt_err_code);
60          exit(1);
61      }
62
63      /*********************************/
64      /* --- Initialize rtcan device --- */
65      /*********************************/
66      bt_err_code = rtcan_initialize(argc, argv);
67      if (bt_err_code)
68      {
69          bt_display_errors(bt_err_code);
70          exit(1);
71      }
72
73      /*********************************/
74      /* --- Initialize Save Buffer --- */
75      /*********************************/
76      data = bt_configure_save_buffer(number_of_samples);
77
78      /*******************************/
79      /* --- Acquire BioTac data --- */
80      /*******************************/
81      bt_rtcan_listen(data, number_of_samples, NO);
82
83      /*********************/
84      /* --- Save data --- */
85      /*********************/
86      bt_save_data("output.txt", data, number_of_samples);
87
88      return 0;
89  }
```

The `main` function for the example starts from Line 41. First, a couple of variables are declared (Lines 46-47). On Lines 49-50, the user can change the duration of the run time. Then, the constant `MAX_BIOTACS_FOR_BARRETT is checked.` As long as there is no change in `bt_rtcan.h`, this does not return any error. The initialization function is called on Line 66 and checks if there has been any error occurred during the initilization.

After allocating memory for a buffer to save data (Line 76), it starts collecting the BioTac samples (Line 81). Note that the function `bt_rtcan_listen` takes `number_of_samples` to determine how long it runs. During data collection, if the user would like to display real-time data on Terminal, the third argument of the function can be changed from `NO` to `YES`. However, keep in mind, especially in the real-time kernel, 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 86). Finally, we close the program after freeing allocated memory.

5 Revision History

(V0.1.0) April 5, 2012

Original document created