Lott/Jayaraman Animal Treadmill SDK

Principle Engineer

Gus K Lott III, PhD

Neurobiological Instrumentation Engineer

Email: lottg@janelia.hhmi.org

Voice: 571.209.4362

Abstract

This document describes the software interface, in C, to the treadmill system via the FTDI serial interface using the FTD2xx C driver instead of the virtual COM port driver. We found that the virtual COM port interface was not reliable for these high data rates. The SDK consists of a short description of the cross platform FTDI drivers, the command codes supported by the treadmill system, and a description of the data streams returned from the treadmill system during operation.

Revision	Date	Comment
0	3/25/10	

SDK Documentation – Rev 0

Prepared by Gus Lott

March 25, 2010

HHMI Janelia Farm Research Campus

19700 Helix Dr.

Ashburn, VA 20147

FTDI Driver interface

The system is communicated with via the FTDI D2xx Driver library. This C interface is supported across platforms including Windows, Linux, Mac OS X, and a variety of embedded platforms as well.

FTDI D2xx Driver Can be downloaded from:

- http://www.ftdichip.com/Drivers/D2XX.htm

This is a general purpose serial to USB bridge. There is not a unique ID associated with the device, so if you have several FTDI driven interfaces, you'll need to explore a connection (by sending query commands and reading responses such as the "dump register" command) until you find the treadmill.

The D2xx SDK Documentation is located:

- http://www.ftdichip.com/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf

Functions of interest are basically limited to:

- FT_Open – Connect to a device
- FT_ResetDevice – Reset the FTDI chip and the serial interface
- FT_SetTimeouts – Define timeouts for blocking reads
- FT_SetDataCharacteristics – Configure Serial Data Stream
    - **FT_BITS_8**
    - **FT_STOP_BITS_1**
    - **FT_PARITY_NONE**
- FT_SetFlowControl – Configure no handshaking
    - **FT_FLOW_NONE**
- FT_SetBaudRate – Set data stream baud rate
    - **1250000 Baud**
- FT_GetQueueStatus – Determine number of bytes available for reading (optional)
- FT_Purge – Flush input and output buffers
- FT_Write – Write data to the treadmill
- FT_Read – Read data from the treadmill
- FT_Close – Close inteface

The SDK programming guide for FTDI's serial interface is clearly written with example code. They also support an "event driven read" from the driver as well.

Byte Command Codes over Serial Interface

All control and data transfer to and from the treadmill system is achieved by raw multi-byte commands sent over the FTDI serial interface. The serial input will reset within 500ms of the last byte sent, so entire commands should be handed to the output buffer of the FTDI driver with a single call to FT_Write. Byte commands are summarized in the following table.

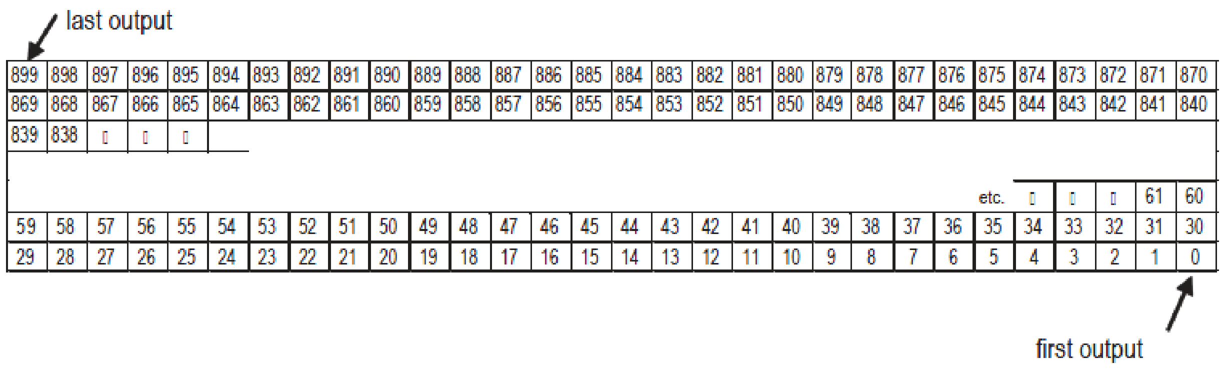
Serial Command Summary

[250 0]	**Turn off Video Stream**
[251 0]	**Initiate video stream**
[252 0]	Dump Internal Registers
[254 0]	**Stop Motion Data Acquisition**
[255 0]	**Start Motion Data Acquisition**

There are several other commands that allow the user to access the internal registers in the ADNS cameras and such, but they are not included in this document due to bugs in the treadmill firmware

Serial Command Detail

- **[251 0] – Initiate Video Stream from Cameras**
  - This command initiates a rapid dump of pixels from both cameras at 20Hz for visual display to a user for alignment and focusing purposes. The pixel streams are interleaved as received on the PC.
  - The stream begins with the bottom right pixel of each frame and continues across rows to the left and then up to the top row as illustrated in this figure from the data sheet:



Return data stream:

Data Stream: C0P0,C1P0,C0P1,C1P1,C0P2,C1P2,…,C0P899,C1P899

Frame Size per camera: 30x30 = 900 pixels

Data Stream chunk per 50ms video interval: 1800 pixels

The length of 1 frame is 900 pixels (30x30 image). So one complete from from both cameras consists of 1800 bytes returned from the chip. No effort is made to indicate when the frame starts, it is assumed that no bytes are lost and that the PC software basically poll for 1800 byte chunks.

- **[250 0] – Stop Video Stream from Cameras**
  - There is a bug in the firmware that requires the system to be physically reset after stopping the video stream in order to enter into motion tracking mode. A user should only have to enter video mode infrequently for setup and calibration purposes, so this is not a critical bug, but can be fixed in a future release.

- **[252 0] – Dump Internal Registers**
  - This command may be used to poll a candidate FTDI interface to see if it is a treadmill and also returns status info from the internal registers in the ADNS camera chip
  - This command should rapidly return 50 bytes read from registers within each camera chip (interleaved in the data stream).
  - Byte order is:

1. Product ID	9. Resolution	17. Frame Period Max Bound L
2. Revision ID	10. Configuration Bits	18. Frame Period Max Bound U
3. Motion	11. Extended Config	19. Frame Period Min Bound L
4. Delta_X	12. Shutter Lower	20. Frame Period Min Bound U
5. Delta_Y	13. Shutter Upper	21. Shutter Max Bound L
6. SQUAL	14. Frame Period Lower	22. Shutter Max Bound U
7. Pixel Sum	15. Frame Period Upper	23. LP_CFG0
8. Maximum Pixel	16. Configuration II	24. LP_CFG1
		25. Observation

- **[255 0] – Start Motion Data Acquisition**
  - This command initiates the 4kHz Motion data stream from the cameras. The data is returned in 12 byte packets per sample. Each sample consists of motion and status data from each of the cameras.
  - A single sample packet consists of the following data:
    1. Zero (0)
    2. A counter that counts from 1 to 255 and then loops back to 1
    3. Delta_X from camera 0 – zero centered on 128
    4. Delta_Y from camera 0 – zero centered on 128
    5. Delta_X from camera 1 – zero centered on 128
    6. Delta_Y from camera 1 – zero centered on 128
    7. Surface Quality from Camera 0
    8. Surface Quality from Camera 1
    9. High Byte of Shutter Speed, Camera 0

10. Low Byte of Shutter Speed, Camera 0
11. High Byte of Shutter Speed, Camera 1
12. Low Byte of Shutter Speed, Camera 1

    - Only the initial byte is ever allowed to be zero. This was designed to verify that the data stream never misses a byte and that packets remain aligned to the 12 byte width. It's essentially a byte header.
    - Motion values are "movement since last measurement" and are in units of arbitrary counts derived from some level of pixel interpolation on the camera chip. Actual correspondence to physical displacement in millimeters must be calibrated as it is a function of optics.
    - Turn motion values into signed displacements by subtracting 128 from the unsigned number.
    - Subtract one from the quality numbers to get the reported value of "features detected" in the camera field of view. This number represents a confidence in the tracking result. This number is higher when the field of view has more structure. See ADNS6090 data sheet for more details. This is the readout of the SQUAL register.
    - The shutter speed is a number of clock cycles that the shutter is held open for. The high and low bytes are prevented from ever being 1 (to preserve the packet header). The camera clock is 24MHz, so the actual shutter speed is nominally:
      ShutterSpeed = ((HighByte-1)*256+lowByte)/24MHz
    - **The shutter speed must be faster than 250us** in order to prevent quantization error from appearing in the 4kHz data stream. If the shutter speed is too long, then motion will jitter across many samples. For example, a 500us shutter speed would return a zero and then the detected motion in the 4kHz data stream.
    - The shutter speed is a function of the illumination and lens light gathering capacity and is automatically adjusted in the camera to lighting conditions. Typically, this is something that will be setup once and then ignored (similar to SQUAL).
- **[254 0] – Stop Motion Data Stream**
    - The system will complete the current motion packet and stop sending data.