**DESI POSITIONER FIRMWARE**

**Firmware Version: 2.0**

**Release Date: Jan 20, 2016**

**DESI-1710.v2**

Irena Gershkovich, University of Michigan

**Overview of Key Features**

This document describes the functionality of the DESI fiber positioner firmware loaded into the Cortex microcontrollers on the fiber positioner boards. Version 1.0 is based on the original ‘Open Loop’ firmware, developed by H. Heederks at LBNL. Please refer to the document ‘Open Loop Description.doc’ for a conceptual overview of the firmware and a detailed descriptions of the basic commands.

The main features that have been added since previous releases are: the ability to upload a sequence of CAN commands to each positioner and then execute the move table described by the commands (upon receipt of a sync signal), the ability to request and receive data from the positioners, and an expanded command id space that allows us to add more functionality for programming via CAN if needed. Details and instructions about the usage of these new features are provided in the sections below.

**Setup/Interface**

Appendix A provides instructions for installing and setting up JTAG programming with the KEIL μ-Vision environment. This is for loading the firmware into the positioners.

The sync signal can be provided by driving the sync pad low, we currently just do this by connecting a normally-open momentary contact switch between the sync pad and GND. When the switch is pressed the uploaded command sequence is executed. This will eventually be handled by sending a signal from the petal control computer. The sync signal is used only for commands 2-4 (move table commands) and the fiducial command 16. All other commands are executed as they are received (except when a move table is loaded/active).

**Compatibility**

Releases are backward compatible with respect to the major release number.

Version History: 1.0: Original EM release, Jan 20, 2016.

Contact: I. Gershkovich ([igershko@umich.edu](mailto:igershko@umich.edu)), M. Schubnell ([schubnel@umich.edu](mailto:schubnel@umich.edu))

**Commands and Data Structure**

Each CAN command consists of a 29-bit CAN message identifier and up to 8 bytes of data (each yellow box is 1 byte in size). The 8 least significant bits of the identifier are reserved for the command ID. This section describes each of the command of the commands implemented in the firmware.

**Command #2 – set\_currents**

The set\_currents commands is used to set the current parameter for the various stages of motor motion. Each argument is 1 byte long and expressed as a percentage (0-100).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 2**  **[7:0]** | Spin Up/Dwn  Current M0  [7:0] | Cruise Current M0  [7:0] | Creep Current M0  [7:0] | Hold Current M0  [7:0] | Spin Up/Dwn  Current M1  [7:0] | Cruise Current M1  [7:0] | Creep Current M1  [7:0] | Hold Current M!  [7:0] |

**Command #3 – set\_periods**

The set\_periods command is used to set the speed at which the creep and spin up/dwn stages of motion are performed. The command is sent with 3 arguments, each 1 byte in length.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 3**  **[7:0]** | Creep Period M0  [7:0] | Creep Period M1  [7:0] | Spin Up/Dwn Period  [7:0] |  |  |  |  |  |

**Command #4 – set\_up\_move**

Cruise steps, CW creep, and CCW creep steps are now all set with a single command, one axis at a time. This allows us to upload a larger number of steps (3 bytes) for each movement without breaking them up. Additionally a post-pause time must be specified to describe the timing of the move table. This command is sent with 6 bytes in the data field.

NOTE: For a move table of n commands, commands 1 through (n-1) will be sent with a 1 in the execute code field (2 least significant bits of the row field), and the last command (n) will be sent with a 2 in that field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 4**  **[7:0]** | Execute Code [7:4],  Axis Mode [3:0] | Move Steps  [23:16] | Move Steps  [15:8] | Move Steps  [7:0] | Post-Pause  Byte 1  (ms)  [15:8] | Post-Pause  Byte 0  (ms)  [7:0] |  |  |

Execute Code (upper 4 bits of 1st byte): 0 – single command, 1 – keep filling move table, 2 – end of move table (just [5:4] used for this)

Axis Mode:

Axis Flag (bit 2): Theta – 0, Phi – 1

Mode Flag (bit 1): Creep – 0, Cruise – 1

Direction Flag (bit 0): CW – 0, CCW – 1

Upper bit – used for pause only (no movement), all other Axis Mode bits set to 0 for pause only

**Command #5 – set\_reset\_leds**

This is a new command that is used to set and reset the signals sent to the pads that were connected to LEDs. The LEDs will not be loaded, but the signals can be used for testing purposes.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 5**  **[7:0]** | LED Mode  [7:0] |  |  |  |  |  |  |  |

LED Mode: 0 – LEDs off, 1 – both LEDs on, 2 – LED 1 on, 3 – LED 2 on

**Command #6 – run\_test\_sequence**

This is a new command implemented for the purposes of testing. This command causes the firmware to send specific waveforms to the theta/phi motor pads. The data field is left empty for this command.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 6**  **[7:0]** |  |  |  |  |  |  |  |  |

**Command #7 – execute\_move\_table**

This command is used to execute the move table upon its receipt (rather than waiting for a sync signal). The data field is left empty for this command.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 7**  **[7:0]** |  |  |  |  |  |  |  |  |

**Command #8 – get\_move\_table\_status (also clear\_move\_table)**

This command is used to receive data about the status of the move table and to send a checksum for verifying that the move table has been received without errors. If a checksum mismatch occurs, the positioner will clear its move table and reset to a state of waiting for a new one.

NOTE: The move table can be cleared by sending this command with 0's in the checksum data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 8**  **[7:0]** | Bit Sum [31:24] | Bit Sum [23:16] | Bit Sum [15:8] | Bit Sum [7:0] |  |  |  |  |

**Expected Response (from positioners):**

Status Response Code: 1 – Move table received, checksum match, 2 – Move table received checksum mismatch, reset 3 – Ready for new move table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | Bit Sum [7:0] | Bit Sum [15:8] | Bit Sum [23:16] | Bit Sum [31:24] | Status  Response  Code |  |  |  |

**Command #9 – get\_temperature**

This command is used to request the 14-bit ADC value associated with the temperature sensor. It returns 2 bytes and is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 9**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | Temp  [7:0] | Temp  [15:8] |  |  |  |  |  |  |

**Command #10 – get\_CAN\_address**

This command is used to request the CAN address of the positioner. It is sent with the global positioner id (20000/0x4e20) that all positioners are configured to accept. This command is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global**  **positioner id**  **[28:8]** | **command id: 10**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | CAN  Address  [7:0] | CAN  Address  [15:8] |  |  |  |  |  |  |

**Command #11 – get\_firmware\_version**

This command is used to request the version of the firmware running on the positioners. It returns 1 byte and is sent with an empty data field. There is assumed to be a decimal point between the last digit and the rest of the digits in the version number returned (base 10). For example, if the firmware version 0x0000000C (12) is received, the version number is 1.2.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 11**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | Firmware  Version  [7:0] |  |  |  |  |  |  |  |

**Command #12 – get\_device\_type**

This command is used to request the currently configured device type (fiducial = 1, positioner = 0). It returns 1 byte and is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 12**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | Device Type  [7:0] |  |  |  |  |  |  |  |

**Command #13 – get\_movement\_status**

This command is used to request the movement status of the positioners (moving = 1, stopped = 0). It returns 1 byte and is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 13**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | Movement  Status  [7:0] |  |  |  |  |  |  |  |

**Command #14 – get\_current\_monitor1\_val**

This command is used to request the 14-bit ADC value that corresponds to current monitor 1. It returns 1 byte and is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 14**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | IMON1  Value  [7:0] | IMON1  Value  [15:8] |  |  |  |  |  |  |

**Command #15 – get\_current\_monitor2\_val**

This command is used to request the 14-bit ADC value that corresponds to current monitor 1. It returns 1 byte and is sent with an empty data field.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 15**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | IMON2  Value  [7:0] | IMON2  Value  [15:8] |  |  |  |  |  |  |

**Command #16 – set\_device\_type**

This command is used to switch between device modes (from positioner to fiducial and vice versa). The command is sent with arguments: device type (fiducial = 1, positioner = 0), duty cycle (216=100 %, 0 = off), and the timer divide number (TIMDIV - which produces different periods). The timer divide number is normally 4000 for generating the motor waveforms (and is fixed for positioner mode regardless of what arguments are sent). It represents the number of 72 MHz clock cycles that occur between interrupts and the waveform period is given by TIMDIV/72,000,000 (sec). TIMDIV should not be set lower than 1650 (period = 23 μsec). This command is executed immediately upon receipt.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **positioner id**  **[28:8]** | **command id: 16**  **[7:0]** | Device  Type  [7:0] | Duty Cycle  [15:8] | Duty Cycle  [7:0] | TIMDIV  [31:24] | TIMDIV  [23:16] | TIMDIV  [15:8] | TIMDIV  [7:0] |  |

**Command #17 – read\_sid\_lower**

This command is used to read the lower 64 bits of the complete 96-bit unique silicon id. The command is sent without an argument in the data field and returns the lower 64 bits read from the UID register.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global**  **positioner id**  **[28:8]** | **command id: 17**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | SIDL  [7:0] | SIDL  [15:8] | SIDL  [23:16] | SIDL  [31:24] | SIDL  [39:32] | SIDL  [47:40] | SIDL  [55:48] | SIDL  [63:56] |

**Command #18 – read\_sid\_upper**

This command is used to read the highest 32 bits of the complete 96-bit unique silicon id. The command is sent without an argument in the data field and returns the upper 32 bits read from the UID register.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global positioner id**  **[28:8]** | **command id: 18**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | SIDU  [7:0] | SIDU  [15:8] | SIDU  [23:16] | SIDU  [31:24] |  |  |  |  |

**Command #19 – read\_sid\_short**

This command is used to read the abbreviated version of the unique silicon id. It is an alternative to commands 17 & 18. The command is sent without an argument in the data field and returns the unique 64 bits constructed from the contents of the UID register (based STMicro's instructions).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global**  **positioner id**  **[28:8]** | **command id: 19**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | SID[7:0] | SID[15:8] | SID[23:16] | SID[31:24] | SID[39:32] | SID[47:40] | SID[55:48] | SID[63:56] |

**Command #20 – write\_CAN\_address**

This command is used to write the positioner's CAN address to flash after verifying its unique id. Prior to sending this command, the **check\_sid** command must be sent with the unique id of the target positioner.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global**  **positioner id**  **[28:8]** | **command id: 20**  **[7:0]** | CAN address  [15:8] | CAN address  [7:0] |  |  |  |  |  |  |

**Command #21 – read\_CAN\_address**

This command is used to read the first location of page 61 in flash memory where the positioner CAN address is stored. The command is sent without an argument in the data field and returns the positioner CAN address in 2 bytes.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global**  **positioner id**  **[28:8]** | **command id: 21**  **[7:0]** |  |  |  |  |  |  |  |  |

**Expected Response (from positioners):**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **positioner id**  **[28:0]** | CAN  address  [7:0] | CAN  address  [15:8] |  |  |  |  |  |  |

**Command #22 – check\_sid\_lower**

This command is used to send the lower bits of a specific unique ID to the positioners and partially enable the target positioner's write\_CAN\_address flag. The check\_sid\_upper command must also be sent to fully enable the write\_CAN\_address operation (command 20). Alternatively, the abbreviated version (64-bit) of the unique ID can be used to set the write\_CAN\_address flag prior to sending command 20. Commands 22 and 23 allow us to use the full version of 96-bit unique ID in case that is necessary.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global positioner id**  **[28:8]** | **command id: 22**  **[7:0]** | SIDL  [63:56] | SIDL  [55:48] | SIDL  [47:40] | SIDL  [39:32] | SIDL  [31:24] | SIDL  [23:16] | SIDL  [15:8] | SIDL  [7:0] |

**Command #23 – check\_sid\_upper**

This command is used to send the highest bits of a specific unique ID to the positioners and partially enable the target positioner's write\_CAN\_address flag. The check\_sid\_lower command must also be sent to fully enable the write\_CAN\_address operation (command 20). Alternatively, the abbreviated version (64-bit) of the unique ID can be used to set the write\_CAN\_address flag prior to sending command 20. Commands 22 and 23 allow us to use the full version of 96-bit unique ID in case that is necessary.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global positioner id**  **[28:8]** | **command id: 23**  **[7:0]** | SIDU  [31:24] | SIDU  [23:16] | SIDU  [15:8] | SIDU  [7:0] |  |  |  |  |

**Command #24 – check\_sid\_short**

This command is used to send the abbreviated version of a specific unique ID to the positioners and fully enable the target positioner's write\_CAN\_address flag. Commands 22 and 23 are an alternative that allow us to use the full version of 96-bit unique ID in case that is necessary.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **global positioner id**  **[28:8]** | **command id: 24**  **[7:0]** | SID[63:56] | SID[55:48] | SID[47:40] | SID[39:32] | SID[31:24] | SID[23:16] | SID[15:8] | SID[7:0] |

**Command #25 - #n: Reserved for bootloader commands**

**APPENDIX A – JTAG Programming and Installation Instructions**

**Necessary Equipment:**  Windows machine with Keil u-Vision environment installed, Keil ULINK-ME USB programmer (or other compatible device), positioner board with power and JTAG connection

1) Download the Keil u-Vision environment onto a Windows machine (or use an existing setup).

2) Load the entire "fipos\_fw\_1.0" folder anywhere onto the machine, open the folder and then

double-click on the uVision4 project file (Open\_Loop\_Test.uvproj) that has the u-Vision logo

next to it. If using a version of u-Vision > 4, you will be prompted to download Legacy support.

Click on the link, complete the download of Legacy Support, then restart u-Vision.

3) Hardware: plug in the KEIL ULINK-ME programmer into one of the machine's USB ports. You

should see a blue LED blinking slowly once the programmer is plugged in and recognized.

4) Supply power to the board and connect it to the KEIL ULINK-ME via a JTAG connector.

6) Save the project once you do this, then rebuild by clicking Project->Rebuild all target files. This

step needs to be performed when anything is changed in the code.

7) Click Flash->Download to finish loading the firmware to the board then power cycle the board.