MCU commands

This document provides information on the low-level micro-controller commands that are sent from the Klipper "host" software and processed by the Klipper micro-controller software. This document is not an authoritative reference for these commands, nor is it an exclusive list of all available commands.

This document may be useful for developers interested in understanding the low-level micro-controller commands.

See the protocol document for more information on the format of commands and their transmission. The commands here are described using their "printf" style syntax - for those unfamiliar with that format, just note that where a '%...' sequence is seen it should be replaced with an actual integer. For example, a description with "count=%c" could be replaced with the text "count=10". Note that parameters that are considered "enumerations" (see the above protocol document) take a string value which is automatically converted to an integer value for the micro-controller. This is common with parameters named "pin" (or that have a suffix of "_pin").

Startup Commands

It may be necessary to take certain one-time actions to configure the micro-controller and its peripherals. This section lists common commands available for that purpose. Unlike most micro-controller commands, these commands run as soon as they are received and they do not require any particular setup.

Common startup commands:

- `set_digital_out pin=%u value=%c` : This command immediately configures the given pin as a digital out GPIO and it sets it to either a low level (value=0) or a high level (value=1). This command may be useful for configuring the initial value of LEDs and for configuring the initial value of stepper driver micro-stepping pins.
- `set_pwm_out pin=%u cycle_ticks=%u value=%hu` : This command will immediately configure the given pin to use hardware based pulse-width-modulation (PWM) with the given number of cycle_ticks. The "cycle_ticks" is the number of MCU clock ticks each power on and power off cycle should last. A cycle_ticks value of 1 can be used to request the fastest possible cycle time. The "value" parameter is between 0 and 255 with 0 indicating a full off state and 255 indicating a full on state. This command may be useful for enabling CPU and nozzle cooling fans.

Low-level micro-controller configuration

Most commands in the micro-controller require an initial setup before they can be successfully invoked. This section provides an overview of the configuration process. This section and the following sections are likely only of interest to developers interested in the internal details of Klipper.

When the host first connects to the micro-controller it always starts by obtaining a data dictionary (see protocol for more information). After the data dictionary is obtained the host will check if the micro-controller is in a "configured" state and configure it if not. Configuration involves the following phases:

- `get_config` : The host starts by checking if the micro-controller is already configured. The micro-controller responds to this command with a "config" response message. The micro-controller software always starts in an unconfigured state at power-on. It remains in this state until the host completes the configuration processes (by issuing a finalize_config command). If the micro-controller is already configured from a previous session (and is configured with the desired settings) then no further action is needed by the host and the configuration process ends successfully.

- `allocate_oids count=%c` : This command is issued to inform the micro-controller of the maximum number of object-ids (oid) that the host requires. It is only valid to issue this command once. An oid is an integer identifier allocated to each stepper, each endstop, and each schedulable gpio pin. The host determines in advance the number of oids it will require to operate the hardware and passes this to the micro-controller so that it may allocate sufficient memory to store a mapping from oid to internal object.

- `config_XXX oid=%c ...` : By convention any command starting with the "config_" prefix creates a new micro-controller object and assigns the given oid to it. For example, the config_digital_out command will configure the specified pin as a digital output GPIO and create an internal object that the host can use to schedule changes to the given GPIO. The oid parameter passed into the config command is selected by the host and must be between zero and the maximum count supplied in the allocate_oids command. The config commands may only be run when the micro-controller is not in a configured state (ie, prior to the host sending finalize_config) and after the allocate_oids command has been sent.

- `finalize_config crc=%u` : The finalize_config command transitions the micro-controller from an unconfigured state to a configured state. The crc parameter passed to the micro-controller is stored and provided back to the host in "config" response messages. By convention, the host takes a 32bit CRC of the configuration it will request and at the start of subsequent communication sessions it checks that the CRC stored in the micro-controller exactly matches its desired CRC. If the CRC does not match then the host knows the micro-controller has not been configured in the state desired by the host.

Common micro-controller objects

This section lists some commonly used config commands.

- `config_digital_out oid=%c pin=%u value=%c default_value=%c max_duration=%u` : This command creates an internal micro-controller object for the given GPIO 'pin'. The pin will be configured in digital output mode and set to an initial value as specified by 'value' (0 for low, 1 for high). Creating a digital_out object allows the host to schedule GPIO updates for the given pin at specified times (see the queue_digital_out command described below). Should the micro-controller software go into shutdown mode then all configured digital_out objects will be set to 'default_value'. The 'max_duration' parameter is used to implement a safety check - if it is non-zero then it is the maximum number of clock ticks that the host may set the given GPIO to a non-default value without further updates. For example, if the default_value is zero and the max_duration is 16000 then if the host sets the gpio to a value of one then it must schedule another update to the gpio pin (to either zero or one) within 16000 clock ticks. This safety feature can be used with heater pins to ensure the host does not enable the heater and then go off-line.

- `config_pwm_out oid=%c pin=%u cycle_ticks=%u value=%hu default_value=%hu max_duration=%u` : This command creates an internal object for hardware based PWM pins that the host may schedule updates for. Its usage is analogous to config_digital_out - see the description of the 'set_pwm_out' and 'config_digital_out' commands for parameter description.

- `config_analog_in oid=%c pin=%u` : This command is used to configure a pin in analog input sampling mode. Once configured, the pin can be sampled at regular interval using the query_analog_in command (see below).

- `config_stepper oid=%c step_pin=%c dir_pin=%c invert_step=%c step_pulse_ticks=%u` : This command creates an internal stepper object. The 'step_pin' and 'dir_pin' parameters specify the step and direction pins respectively; this command will configure them in digital output mode. The 'invert_step' parameter specifies whether a step occurs on a rising edge (invert_step=0) or falling edge (invert_step=1). The 'step_pulse_ticks' parameter specifies the minimum duration of the step pulse. If the mcu exports the constant 'STEPPER_BOTH_EDGE=1' then setting step_pulse_ticks=0 and invert_step=-1 will setup for stepping on both the rising and falling edges of the step pin.

- `config_endstop oid=%c pin=%c pull_up=%c stepper_count=%c` : This command creates an internal "endstop" object. It is used to specify the endstop pins and to enable "homing" operations (see the endstop_home command below). The command will configure the specified pin in digital input mode. The 'pull_up' parameter determines whether hardware provided pullup resistors for the pin (if available) will be enabled. The 'stepper_count' parameter specifies the maximum number of steppers that this endstop may need to halt during a homing operation (see endstop_home below).

- `config_spi oid=%c bus=%u pin=%u mode=%u rate=%u shutdown_msg=%*s` : This command creates an internal SPI object. It is used with spi_transfer and spi_send commands (see below). The "bus" identifies the SPI bus to use (if the micro-controller has more than one SPI bus available). The "pin" specifies the chip select (CS) pin for the device. The "mode" is the SPI mode (should be between 0 and 3). The "rate" parameter specifies the SPI bus rate (in cycles per second). Finally, the "shutdown_msg" is an SPI command to send to the given device should the micro-controller go into a shutdown state.

- `config_spi_without_cs oid=%c bus=%u mode=%u rate=%u shutdown_msg=%*s` : This command is similar to config_spi, but without a CS pin definition. It is useful for SPI devices that do not have a chip select line.

Common commands

This section lists some commonly used run-time commands. It is likely only of interest to developers looking to gain insight into Klipper.

- `set_digital_out_pwm_cycle oid=%c cycle_ticks=%u` : This command configures a digital output pin (as created by config_digital_out) to use "software PWM". The 'cycle_ticks' is the number of clock ticks for the PWM cycle. Because the output switching is implemented in the micro-controller software, it is recommended that 'cycle_ticks' correspond to a time of 10ms or greater.

- `queue_digital_out oid=%c clock=%u on_ticks=%u` : This command will schedule a change to a digital output GPIO pin at the given clock time. To use this command a 'config_digital_out' command with the same 'oid' parameter must have been issued during micro-controller configuration. If 'set_digital_out_pwm_cycle' has been called then 'on_ticks' is the on duration (in clock ticks) for the pwm cycle. Otherwise, 'on_ticks' should be either 0 (for low voltage) or 1 (for high voltage).

- `queue_pwm_out oid=%c clock=%u value=%hu` : Schedules a change to a hardware PWM output pin. See the 'queue_digital_out' and 'config_pwm_out' commands for more info.

- `query_analog_in oid=%c clock=%u sample_ticks=%u sample_count=%c rest_ticks=%u min_value=%hu max_value=%hu` : This command sets up a recurring schedule of analog input samples. To use this command a 'config_analog_in' command with the same 'oid' parameter must have been issued during micro-controller configuration. The samples will start as of 'clock' time, it will report on the obtained value every 'rest_ticks' clock ticks, it will over-sample 'sample_count' number of times, and it will pause 'sample_ticks' number of clock ticks between over-sample samples. The 'min_value' and 'max_value' parameters implement a safety feature - the micro-controller software will verify the sampled value (after any oversampling) is always between the supplied range. This is intended for use with pins attached to thermistors controlling heaters - it can be used to check that a heater is within a temperature range.

- `get_clock` : This command causes the micro-controller to generate a "clock" response message. The host sends this command once a second to obtain the value of the micro-controller clock and to estimate the drift between host and micro-controller clocks. It enables the host to accurately estimate the micro-controller clock.

Stepper commands

- `queue_step oid=%c interval=%u count=%hu add=%hi` : This command schedules 'count' number of steps for the given stepper, with 'interval' number of clock ticks between each step. The first step will be 'interval' number of clock ticks since the last scheduled step for the given stepper. If 'add' is non-zero then the interval will be adjusted by 'add' amount after each step. This command appends the given interval/count/add sequence to a per-stepper queue. There may be hundreds of these sequences queued during normal operation. New sequence are appended to the end of the queue and as each sequence completes its 'count' number of steps it is popped from the front of the queue. This system allows the micro-controller to queue potentially hundreds of thousands of steps - all with reliable and predictable schedule times.

- `set_next_step_dir oid=%c dir=%c` : This command specifies the value of the dir_pin that the next queue_step command will use.

- `reset_step_clock oid=%c clock=%u` : Normally, step timing is relative to the last step for a given stepper. This command resets the clock so that the next step is relative to the supplied 'clock' time. The host usually only sends this command at the start of a print.

- `stepper_get_position oid=%c` : This command causes the micro-controller to generate a "stepper_position" response message with the stepper's current position. The position is the total number of steps generated with dir=1 minus the total number of steps generated with dir=0.

- `endstop_home oid=%c clock=%u sample_ticks=%u sample_count=%c rest_ticks=%u pin_value=%c` : This command is used during stepper "homing" operations. To use this command a 'config_endstop' command with the same 'oid' parameter must have been issued during micro-controller configuration. When this command is invoked, the micro-controller will sample the endstop pin every 'rest_ticks' clock ticks and check if it has a value equal to 'pin_value'. If the value matches (and it continues to match for 'sample_count' additional samples spread 'sample_ticks' apart) then the movement queue for the associated stepper will be cleared and the stepper will come to an immediate halt. The host uses this command to implement homing - the host instructs the endstop to sample for the endstop trigger and then it issues a series of queue_step commands to move a stepper towards the endstop. Once the stepper hits the endstop, the trigger will be detected, the movement halted, and the host notified.

Move queue

Each queue_step command utilizes an entry in the micro-controller "move queue". This queue is allocated when it receives the "finalize_config" command, and it reports the number of available queue entries in "config" response messages.

It is the responsibility of the host to ensure that there is available space in the queue before sending a queue_step command. The host does this by calculating when each queue_step command completes and scheduling new queue_step commands accordingly.

SPI Commands

- `spi_transfer oid=%c data=%*s` : This command causes the micro-controller to send 'data' to the spi device specified by 'oid' and it generates a "spi_transfer_response" response message with the data returned during the transmission.
- `spi_send oid=%c data=%*s` : This command is similar to "spi_transfer", but it does not generate a "spi_transfer_response" message.