Claw Device Controller

Generated by Doxygen 1.16.0

Chapter 1

claw

Raspberry pico 2 code to drive stepper motor driven claw

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 stepper_state Struct Reference

Structure to hold stepper motor state.

`#include <stepper.h>`

Public Attributes

- int **current_position**

  Current position in steps.
- int **target_position**

  Target position in steps.
- int **step_period**

  Step period in TIMMER_INTERVAL_US units.
- bool **moving**

  Is the stepper currently moving.
- bool **enabled**

  Is the stepper enabled.

4.1.1 Detailed Description

Structure to hold stepper motor state.

The documentation for this struct was generated from the following file:

- stepper.h

Chapter 5

File Documentation

5.1 claw.c File Reference

Firmware to control a claw device with stepper motors via USB serial commands.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include <string.h>
#include "pico/assert.h"
#include <stdlib.h>
#include "hardware/gpio.h"
#include <ctype.h>
#include <math.h>
#include "sys_timer.h"
#include "stepper.h"
#include "led.h"
#include "command_processor.h"
```

Functions

- int main ()

  Main function.

5.1.1 Detailed Description

Firmware to control a claw device with stepper motors via USB serial commands.

Author

Jon Wade

Date

19 Dec 2025

Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

Command interface for controlling stepper motors and other functions associated with the claw device. This file implements a simple command interface over USB serial to control the claw device.

5.1.2 Function Documentation

5.1.2.1 main()

```
int main ()
```

Main function.

Parameters

	none

Returns

: none

5.2 command_processor.c File Reference

implementation of command processing functions

```
#include <stdio.h>
#include "pico/stdlib.h"
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include "sys_timer.h"
#include "stepper.h"
#include "led.h"
#include "command_processor.h"
```

Macros

- #define **MAX_COMMAND_LENGTH** 50
- #define **CLAW_SET_POSITION_COMMAND** "claw_set "
- #define **LED_PERIOD_COMMAND** "led_period "
- #define **SET_STEPPER_PERIOD_COMMAND** "set_stepper_period "
- #define **SET_STEPPER_ZERO_COMMAND** "set_stepper_zero"
- #define **MOVE_STEPPER_ABSOLUTE_COMMAND** "move_stepper_absolute "
- #define **MOVE_STEPPER_RELATIVE_COMMAND** "move_stepper_relative "
- #define **MOVE_STEPPER_ROTATIONS_COMMAND** "move_stepper_rotations "
- #define **MOVE_STEPPER_BUMP_DOWN_COMMAND** "move_stepper_bump_down"
- #define **STOP_STEPPER_COMMAND** "stop_stepper"
- #define **GET_STEPPER_STATUS_COMMAND** "get_stepper_status"
- #define **ENABLE_STEPPER_COMMAND** "enable_stepper"
- #define **DISABLE_STEPPER_COMMAND** "disable_stepper"

Functions

- bool process_command (const char *cmd, stepper_state_t *stepper)

  Process a command string.
- char * process_stdin_input (void)

  Process stdin input.
- bool command_get_stepper_status (stepper_state_t *stepper)

  Command helper function to get stepper status.
- bool command_claw_set_position (stepper_state_t *stepper, const char *cmd)

  Command helper function to set claw position.
- bool command_set_led_period (const char *cmd)

  Command helper function to set LED period.
- bool command_set_stepper_period (stepper_state_t *stepper, const char *cmd)

  Command helper function to set stepper period.
- bool command_set_stepper_zero (stepper_state_t *stepper)

  Command helper function to set stepper position to zero.
- bool command_move_stepper_absolute (stepper_state_t *stepper, const char *cmd)

  Command helper function to move stepper to an absolute position.
- bool command_move_stepper_relative (stepper_state_t *stepper, const char *cmd)

  Command helper function to move stepper by a relative amount.
- bool command_move_stepper_rotations (stepper_state_t *stepper, const char *cmd)

  Command helper function to move stepper by a relative amount.
- bool command_move_stepper_bump_down (stepper_state_t *stepper)

  Command helper function to move stepper by a relative amount.
- bool command_stop_stepper (stepper_state_t *stepper)

  Command helper function to stop stepper movement.

Variables

- const char * help_message

  Help message.

5.2.1 Detailed Description

implementation of command processing functions

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the implementation of functions for processing commands.

5.2.2 Function Documentation

5.2.2.1 command_claw_set_position()

```
bool command_claw_set_position (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to set claw position.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.2 command_get_stepper_status()

```
bool command_get_stepper_status (
            stepper_state_t * stepper)
```

Command helper function to get stepper status.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.2.2.3 command_move_stepper_absolute()

```
bool command_move_stepper_absolute (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper to an absolute position.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.4 command_move_stepper_bump_down()

```
bool command_move_stepper_bump_down (
            stepper_state_t * stepper)
```

Command helper function to move stepper by a relative amount.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.2.2.5 command_move_stepper_relative()

```
bool command_move_stepper_relative (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper by a relative amount.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.6 command_move_stepper_rotations()

```
bool command_move_stepper_rotations (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper by a relative amount.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.7 command_set_led_period()

```
bool command_set_led_period (
            const char * cmd)
```

Command helper function to set LED period.

Note

: Function is not completely safe, assumes valid command string

Parameters

cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.8 command_set_stepper_period()

```
bool command_set_stepper_period (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to set stepper period.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.2.2.9 command_set_stepper_zero()

```
bool command_set_stepper_zero (
            stepper_state_t * stepper)
```

Command helper function to set stepper position to zero.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.2.2.10 command_stop_stepper()

```
bool command_stop_stepper (
            stepper_state_t * stepper)
```

Command helper function to stop stepper movement.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.2.2.11 process_command()

```
bool process_command (
            const char * cmd,
            stepper_state_t * stepper)
```

Process a command string.

Note

: This function parses the command string and calls the appropriate command helper function. It checks for

Parameters

cmd	pointer to command string
stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.2.2.12 process_stdin_input()

```
char * process_stdin_input (
            void )
```

Process stdin input.

Note

: This function reads characters from stdin, builds commands, and returns complete command strings.

This function has a simple lock to prevent re-entrancy.

Parameters

	none

Returns

: non-null pointer to command string when a complete command is received,

5.2.3 Variable Documentation

5.2.3.1 help_message

```
const char* help_message
```

Initial value:

```
                  =
"\n"
"Available commands:\n"
"  claw_set <position>               - Set the claw position 0 to 100\n"
"  led_period <ms>                   - Set the LED blink period in milliseconds\n"
"  set_stepper_period <us>           - Set the stepper motor step period in us\n"
"  set_stepper_zero                  - Set the current position to zero\n"
"  move_stepper_absolute <steps>     - Move the stepper to an absolute position\n"
"  move_stepper_relative <steps>     - Move the stepper by a relative number of steps\n"
"  move_stepper_rotations <rotations> - Move the stepper by a number of rotations\n"
"  move_stepper_bump_down            - Move the stepper down by a small fixed amount\n"
"  stop_stepper                      - Stop the stepper motor\n"
"  get_stepper_status                - Get the current status of the stepper motor\n"
"  enable_stepper                    - Enable the stepper motor\n"
"  disable_stepper                   - Disable the stepper motor\n"
"  help                              - Show this help message\n"
"-----\n"
```

Help message.

This message is displayed when the user requests help or enters an unknown command.

5.3 command_processor.h File Reference

Definitions, functions and variables for command processing.

```
#include "stepper.h"
```

Functions

- bool process_command (const char *cmd, stepper_state_t *stepper)
  Process a command string.
- char * process_stdin_input (void)
  Process stdin input.
- bool command_claw_set_position (stepper_state_t *stepper, const char *cmd)
  Command helper function to set claw position.
- bool command_set_led_period (const char *cmd)
  Command helper function to set LED period.
- bool command_set_stepper_period (stepper_state_t *stepper, const char *cmd)
  Command helper function to set stepper period.
- bool command_set_stepper_zero (stepper_state_t *stepper)
  Command helper function to set stepper position to zero.
- bool command_move_stepper_absolute (stepper_state_t *stepper, const char *cmd)
  Command helper function to move stepper to an absolute position.
- bool command_move_stepper_relative (stepper_state_t *stepper, const char *cmd)
  Command helper function to move stepper by a relative amount.
- bool command_move_stepper_rotations (stepper_state_t *stepper, const char *cmd)
  Command helper function to move stepper by a relative amount.
- bool command_move_stepper_bump_down (stepper_state_t *stepper)
  Command helper function to move stepper by a relative amount.
- bool command_stop_stepper (stepper_state_t *stepper)
  Command helper function to stop stepper movement.
- bool command_get_stepper_status (stepper_state_t *stepper)
  Command helper function to get stepper status.

5.3.1 Detailed Description

Definitions, functions and variables for command processing.

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the definitions, functions and variables for processing commands.

5.3.2 Function Documentation

5.3.2.1 command_claw_set_position()

```
bool command_claw_set_position (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to set claw position.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.2 command_get_stepper_status()

```
bool command_get_stepper_status (
            stepper_state_t * stepper)
```

Command helper function to get stepper status.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.3.2.3 command_move_stepper_absolute()

```
bool command_move_stepper_absolute (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper to an absolute position.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.4 command_move_stepper_bump_down()

```
bool command_move_stepper_bump_down (
            stepper_state_t * stepper)
```

Command helper function to move stepper by a relative amount.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.3.2.5 command_move_stepper_relative()

```
bool command_move_stepper_relative (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper by a relative amount.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.6 command_move_stepper_rotations()

```
bool command_move_stepper_rotations (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to move stepper by a relative amount.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.7 command_set_led_period()

```
bool command_set_led_period (
            const char * cmd)
```

Command helper function to set LED period.

Note

: Function is not completely safe, assumes valid command string

Parameters

cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.8 command_set_stepper_period()

```
bool command_set_stepper_period (
            stepper_state_t * stepper,
            const char * cmd)
```

Command helper function to set stepper period.

Note

: Function is not completely safe, assumes valid command string

Parameters

stepper	pointer to stepper state structure
cmd	pointer to command string

Returns

: true on success, false on failure

5.3.2.9 command_set_stepper_zero()

```
bool command_set_stepper_zero (
            stepper_state_t * stepper)
```

Command helper function to set stepper position to zero.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.3.2.10 command_stop_stepper()

```
bool command_stop_stepper (
            stepper_state_t * stepper)
```

Command helper function to stop stepper movement.

Parameters

stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.3.2.11 process_command()

```
bool process_command (
            const char * cmd,
            stepper_state_t * stepper)
```

Process a command string.

Note

: This function parses the command string and calls the appropriate command helper function. It checks for

Parameters

cmd	pointer to command string
stepper	pointer to stepper state structure

Returns

: true on success, false on failure

5.3.2.12 process_stdin_input()

```
char * process_stdin_input (
            void )
```

Process stdin input.

Note

: This function reads characters from stdin, builds commands, and returns complete command strings.

This function has a simple lock to prevent re-entrancy.

Parameters

	none

Returns

: non-null pointer to command string when a complete command is received,

5.4 command_processor.h

Go to the documentation of this file.

```
00001
00011
00012 #ifndef COMMAND_PROCESSOR_H
00013 #define COMMAND_PROCESSOR_H
00014
00015 #include "stepper.h"
00016
00027 bool process_command(const char* cmd, stepper_state_t* stepper);
00028
00040 char* process_stdin_input(void);
00041
00051 bool command_claw_set_position(stepper_state_t* stepper, const char* cmd);
00052
00061 bool command_set_led_period(const char* cmd);
00062
00072 bool command_set_stepper_period(stepper_state_t* stepper, const char* cmd);
00073
00080 bool command_set_stepper_zero(stepper_state_t* stepper);
00081
00091 bool command_move_stepper_absolute(stepper_state_t* stepper, const char* cmd);
00092
00102 bool command_move_stepper_relative(stepper_state_t* stepper, const char* cmd);
00103
00113 bool command_move_stepper_rotations(stepper_state_t* stepper, const char* cmd);
00114
00121 bool command_move_stepper_bump_down(stepper_state_t* stepper);
00122
00129 bool command_stop_stepper(stepper_state_t* stepper);
00130
00137 bool command_get_stepper_status(stepper_state_t* stepper);
00138
00139 #endif // COMMAND_PROCESSOR_H
```

5.5 led.c File Reference

Implementations and variables for LED control.

```
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "led.h"
```

Functions

- int pico_led_init (void)
  Initialise the LED.
- void pico_set_led (bool led_on)
  Turn the LED on or off.
- void process_led_tick (void)
  Process LED timing tick.

Variables

- volatile int led_period = LED_DELAY_MS
  LED blink period in milliseconds.

5.5.1 Detailed Description

Implementations and variables for LED control.

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the implementations and variables for controlling an LED.

5.5.2 Function Documentation

5.5.2.1 pico_led_init()

```
int pico_led_init (
            void )
```


Initialise the LED.

Parameters

	none

Returns

: PICO_OK on success, error code on failure

5.5.2.2 pico_set_led()

```
void pico_set_led (
            bool led_on)
```

Turn the LED on or off.

Parameters

led_on	true to turn on, false to turn off

Returns

: none

5.5.2.3 process_led_tick()

```
void process_led_tick (
            void )
```

Process LED timing tick.

Parameters

	none

Returns

: none

5.5.3 Variable Documentation

5.5.3.1 led_period

```
volatile int led_period = LED_DELAY_MS
```

LED blink period in milliseconds.

This variable can be modified via command interface to change the LED blink rate.

5.6 led.h File Reference

Definitions, function definitions and variables for LED control.

Macros

- #define **LED_DELAY_MS** 1000

Functions

- int pico_led_init (void)
  Initialise the LED.
- void pico_set_led (bool led_on)
  Turn the LED on or off.
- void process_led_tick (void)
  Process LED timing tick.

Variables

- volatile int led_period
  LED blink period in milliseconds.

5.6.1 Detailed Description

Definitions, function definitions and variables for LED control.

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the definitions, function definitions and variables for controlling an LED.

5.6.2 Function Documentation

5.6.2.1 pico_led_init()

```
int pico_led_init (
                void )
```


Initialise the LED.

Parameters

	none

Returns

: PICO_OK on success, error code on failure

5.6.2.2 pico_set_led()

```
void pico_set_led (
            bool led_on)
```

Turn the LED on or off.

Parameters

led_on	true to turn on, false to turn off

Returns

: none

5.6.2.3 process_led_tick()

```
void process_led_tick (
            void )
```

Process LED timing tick.

Parameters

	none

Returns

: none

5.6.3 Variable Documentation

5.6.3.1 led_period

```
volatile int led_period  [extern]
```

LED blink period in milliseconds.

This variable can be modified via command interface to change the LED blink rate.

5.7 led.h

Go to the documentation of this file.

```
00001
00011
00012 #ifndef LED_H
00013 #define LED_H
00014
00015 // Define default LED delay if not defined
00016 #ifndef LED_DELAY_MS
00017 #define LED_DELAY_MS 1000
00018 #endif
00019
00025 extern volatile int led_period;
00026
00033 int pico_led_init(void);
00034
00041 void pico_set_led(bool led_on);
00042
00043
00044
00051 void process_led_tick(void);
00052
00053 #endif
```

5.8 stepper.c File Reference

implementation of stepper motor control and related global variables

```
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "stepper.h"
#include "sys_timer.h"
```

Functions

- bool stepper_init (stepper_state_t *stepper, int initial_position, int step_period)
  Initialize the stepper state.
- bool stepper_set_target_position (stepper_state_t *stepper, int target_position)
  Set the target position for the stepper motor.
- bool stepper_set_step_period (stepper_state_t *stepper, int step_period_us)
  Set the step period for the stepper motor.
- bool stepper_stop (stepper_state_t *stepper)
  Stop the stepper motor, setting target position to current position.
- bool stepper_enable (stepper_state_t *stepper, bool enable)
  Enable the stepper motor.
- bool process_stepper_movement (stepper_state_t *stepper)
  Process stepper movement.

5.8.1 Detailed Description

implementation of stepper motor control and related global variables

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the implementation of functions for controlling a stepper motor.

5.8.2 Function Documentation

5.8.2.1 process_stepper_movement()

```
bool process_stepper_movement (
            stepper_state_t * stepper)
```

Process stepper movement.

Parameters

stepper	pointer to stepper state structure

Returns

: true if stepper is still moving, false if it has reached target

5.8.2.2 stepper_enable()

```
bool stepper_enable (
            stepper_state_t * stepper,
            bool enable)
```

Enable the stepper motor.

Parameters

stepper	pointer to stepper state structure, must not be NULL
enable	true to enable, false to disable

Returns

: true on success, false on failure

5.8.2.3 stepper_init()

Parameters

stepper	pointer to stepper state structure to initialize, must not be NULL
initial_position	initial position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_↩ POSITION
step_period	step period in TIMER_INTERVAL_US must be greater than 1 ms

Returns

: true on success, false on failure

5.8.2.4 stepper_set_step_period()

```
bool stepper_set_step_period (
            stepper_state_t * stepper,
            int step_period_us)
```

Set the step period for the stepper motor.

Parameters

stepper	pointer to stepper state structure, must not be NULL
step_period	step period in microseconds must be greater than MIN_STEPPER_PERIOD

Returns

: true on success, false on failure

5.8.2.5 stepper_set_target_position()

```
bool stepper_set_target_position (
            stepper_state_t * stepper,
            int target_position)
```

Set the target position for the stepper motor.

Parameters

stepper	pointer to stepper state structure, must not be NULL
target_position	target position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER↩ _POSITION

Returns

: true on success, false on failure

5.8.2.6 stepper_stop()

```
bool stepper_stop (
```

Parameters

stepper	pointer to stepper state structure, must not be NULL

Returns

: true on success, false on failure

5.9 stepper.h File Reference

Definitions, functions and variables for stepper motor control.

Classes

- struct stepper_state

  Structure to hold stepper motor state.

Macros

- #define **DEFAULT_STEPPER_PERIOD** 4
- #define **MIN_STEPPER_PERIOD** 4
- #define **STEPPER_STEP_PIN** 2
- #define **STEPPER_DIR_PIN** 3
- #define **STEPPER_ENABLE_PIN** 4
- #define **STEPPER_ENABLE_PIN_INVERTED** true
- #define **STEPPER_DIRECTION_FORWARD** 1
- #define **STEPPER_DIRECTION_BACKWARD** 0
- #define **STEPPER_STEPS_PER_REV** 3200
- #define **STEPPER_MAX_REVOLUTIONS** 12
- #define **STEPPER_BUMP_STEPS** (STEPPER_STEPS_PER_REV / 4)
- #define **MAX_STEPPER_POSITION** (STEPPER_STEPS_PER_REV ∗ STEPPER_MAX_REVOLUTIONS)
- #define **MIN_STEPPER_POSITION** 0

Typedefs

- typedef struct stepper_state **stepper_state_t**

Functions

- bool stepper_init (stepper_state_t ∗stepper, int initial_position, int step_period)

  Initialize the stepper state.
- bool stepper_set_target_position (stepper_state_t ∗stepper, int target_position)

  Set the target position for the stepper motor.
- bool stepper_set_step_period (stepper_state_t ∗stepper, int step_period_us)

  Set the step period for the stepper motor.
- bool stepper_stop (stepper_state_t ∗stepper)

  Stop the stepper motor, setting target position to current position.
- bool stepper_enable (stepper_state_t ∗stepper, bool enable)

  Enable the stepper motor.
- bool process_stepper_movement (stepper_state_t ∗stepper)

  Process stepper movement.

5.9.1 Detailed Description

Definitions, functions and variables for stepper motor control.

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the definitions, functions and variables for controlling a stepper motor.

5.9.2 Function Documentation

5.9.2.1 process_stepper_movement()

```
bool process_stepper_movement (
            stepper_state_t * stepper)
```

Process stepper movement.

Parameters

stepper	pointer to stepper state structure

Returns

: true if stepper is still moving, false if it has reached target

5.9.2.2 stepper_enable()

```
bool stepper_enable (
            stepper_state_t * stepper,
            bool enable)
```

Enable the stepper motor.

Parameters

stepper	pointer to stepper state structure, must not be NULL

enable	true to enable, false to disable

Returns

: true on success, false on failure

5.9.2.3 stepper_init()

```
bool stepper_init (
            stepper_state_t * stepper,
            int initial_position,
            int step_period)
```

Initialize the stepper state.

Parameters

stepper	pointer to stepper state structure to initialize, must not be NULL
initial_position	initial position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER_↩ POSITION
step_period	step period in TIMER_INTERVAL_US must be greater than 1 ms

Returns

: true on success, false on failure

5.9.2.4 stepper_set_step_period()

```
bool stepper_set_step_period (
            stepper_state_t * stepper,
            int step_period_us)
```

Set the step period for the stepper motor.

Parameters

stepper	pointer to stepper state structure, must not be NULL
step_period	step period in microseconds must be greater than MIN_STEPPER_PERIOD

Returns

: true on success, false on failure

5.9.2.5 stepper_set_target_position()


```
bool stepper_set_target_position (
            stepper_state_t * stepper,
            int target_position)
```

Parameters

stepper	pointer to stepper state structure, must not be NULL
target_position	target position in steps must be between MIN_STEPPER_POSITION and MAX_STEPPER↩ _POSITION

Returns

: true on success, false on failure

5.9.2.6 stepper_stop()

```
bool stepper_stop (
            stepper_state_t * stepper)
```

Stop the stepper motor, setting target position to current position.

Parameters

stepper	pointer to stepper state structure, must not be NULL

Returns

: true on success, false on failure

5.10 stepper.h

Go to the documentation of this file.

```
00001
00011
00012 #ifndef STEPPER_H
00013 #define STEPPER_H
00014
00015 // Stepper motor configuration
00016 #define DEFAULT_STEPPER_PERIOD          4       // Default step period in TIMER_INTERVAL_US units (4 *
      10 us = 40 us = 25 kHz)
00017 #define MIN_STEPPER_PERIOD              4       // Minimum step period in TIMER_INTERVAL_US units (4 *
      10 us = 40 us = 25 kHz)
00018 #define STEPPER_STEP_PIN                2       // GPIO pin for stepper step control
00019 #define STEPPER_DIR_PIN                 3       // GPIO pin for stepper direction control
00020 #define STEPPER_ENABLE_PIN              4       // GPIO pin for stepper enable control
00021 #define STEPPER_ENABLE_PIN_INVERTED     true    // Set to true if enable pin is active low
00022 #define STEPPER_DIRECTION_FORWARD       1
00023 #define STEPPER_DIRECTION_BACKWARD      0
00024 #define STEPPER_STEPS_PER_REV           3200    // Number of steps per revolution for the stepper
      motor
00025                                                 // 16 microsteps / 1.8 degree step angle * 360 degrees
      = 3200 steps
00026 #define STEPPER_MAX_REVOLUTIONS         12      // Maximum number of revolutions the stepper can move
00027                                                 // 20 TPI lead screw with 3/4 inch travel = 15
      revolutions
00028 #define STEPPER_BUMP_STEPS              (STEPPER_STEPS_PER_REV / 4) // Number of steps to move for a
      bump down command (1/4 revolution)
00029 #define MAX_STEPPER_POSITION            (STEPPER_STEPS_PER_REV * STEPPER_MAX_REVOLUTIONS)
00030 #define MIN_STEPPER_POSITION            0
00031
00035 typedef struct stepper_state
00036 {
00037     int current_position;
00038     int target_position;
00039     int step_period;
```

```
00040     bool moving;
00041     bool enabled;
00042 } stepper_state_t;
00043
00044 // Function prototypes
00045
00054 bool stepper_init(stepper_state_t* stepper, int initial_position, int step_period);
00055
00063 bool stepper_set_target_position(stepper_state_t* stepper, int target_position);
00064
00072 bool stepper_set_step_period(stepper_state_t* stepper, int step_period_us);
00073
00080 bool stepper_stop(stepper_state_t* stepper);
00081
00089 bool stepper_enable(stepper_state_t* stepper, bool enable);
00090
00097 bool process_stepper_movement(stepper_state_t* stepper);
00098
00099 #endif // STEPPER_H
```

5.11 sys_timer.c File Reference

implementation of system timer and related global variables

```
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include "sys_timer.h"
```

Functions

- bool timer_callback (struct repeating_timer *t)
  Millisecond timer callback.

Variables

- volatile int ten_us_ticks_count = 0
  Global ten microsecond ticks count.
- volatile int ms_ticks_count = 0
  Global millisecond ticks count.

5.11.1 Detailed Description

implementation of system timer and related global variables

Author

Jon Wade

Date

20 Dec 2025

Copyright

(c) 2025 Jon Wade. Standard MIT License applies. See LICENSE file.

This file contains the timer callback function and related definitions for the system timer.

5.11.2 Function Documentation

5.11.2.1 timer_callback()

```
bool timer_callback (
            struct repeating_timer * t)
```

Millisecond timer callback.

Note

: This function is called every millisecond by the repeating timer.

Parameters

t	pointer to repeating_timer struct

Returns

: true to keep repeating, false to stop

5.11.3 Variable Documentation

5.11.3.1 ms_ticks_count

```
volatile int ms_ticks_count = 0
```

Global millisecond ticks count.

This variable is incremented by the timer callback every 100 calls (1 ms = 100 $*$ 10 us) and decremented in the main loop to track when the millisecond tasks should run.

5.11.3.2 ten_us_ticks_count

```
volatile int ten_us_ticks_count = 0
```

Global ten microsecond ticks count.

This variable is incremented by the timer callback and decremented in the main loop to track when the ten microsecond tasks should run.

5.12 sys_timer.h File Reference

Definitions, functions and variables for system timer.

Macros

- #define **TIMER_INTERVAL_US** 10

Functions

- bool timer_callback (struct repeating_timer *t)

  Millisecond timer callback.

Variables

- volatile int ten_us_ticks_count

  Global ten microsecond ticks count.
- volatile int ms_ticks_count

  Global millisecond ticks count.

5.12.1 Detailed Description

Definitions, functions and variables for system timer.

Author

Jon Wade

Date

20 Dec 2025

Copyright

This file contains the timer callback function and related definitions for the system timer.

5.12.2 Function Documentation

5.12.2.1 timer_callback()

```
bool timer_callback (
            struct repeating_timer * t)
```

Millisecond timer callback.

Note

: This function is called every millisecond by the repeating timer.

Parameters

t	pointer to repeating_timer struct

Returns

: true to keep repeating, false to stop

5.12.3 Variable Documentation

5.12.3.1 ms_ticks_count

```
volatile int ms_ticks_count  [extern]
```

Global millisecond ticks count.

This variable is incremented by the timer callback every 100 calls (1 ms = 100 ∗ 10 us) and decremented in the main loop to track when the millisecond tasks should run.

5.12.3.2 ten_us_ticks_count

```
volatile int ten_us_ticks_count  [extern]
```

Global ten microsecond ticks count.

This variable is incremented by the timer callback and decremented in the main loop to track when the ten microsecond tasks should run.

5.13 sys_timer.h

Go to the documentation of this file.

```
00001
00011
00012 #ifndef SYS_TIMER_H
00013 #define SYS_TIMER_H
00014
00015 #define TIMER_INTERVAL_US              10       // Timer interval in microseconds
00016
00023 extern volatile int ten_us_ticks_count;
00024
00031
00032 extern volatile int ms_ticks_count;
00033
00042 bool timer_callback(struct repeating_timer *t);
00043
00044 #endif // SYS_TIMER_H
```

Index