BLDC Motor Driver

1.0.0

Generated by Doxygen 1.14.0

Chapter 1

BLDC Motor Driver Documentation

1.1 Introduction

This is a brushless direc current motor driver firmware Six-step commutation motor driver.

1.2 Features

- Layered architecture design
- MCAL abstraction via layer
- Shunt-resistor current monitor support
- VBUS monitor support
- MOSFET and motor temperature monitor via NTC thermistors

1.3 Usage

See YAZILACAK

1.4 Example

```
- YAZILACAK
```

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 commutation_step_t Struct Reference

Public Attributes

- phase_state_t u
- phase_state_t v
- phase_state_t w

4.1.1 Detailed Description

Definition at line 12 of file commutation.c.

4.1.2 Member Data Documentation

4.1.2.1 u

```
phase_state_t commutation_step_t::u
```

Definition at line 13 of file commutation.c.

4.1.2.2 v

```
phase_state_t commutation_step_t::v
```

Definition at line 14 of file commutation.c.

4.1.2.3 w

```
phase_state_t commutation_step_t::w
```

Definition at line 15 of file commutation.c.

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

- APP/Src/commutation.c

Chapter 5

File Documentation

5.1 commutation.h

```
00001 /*
00002  * commutate.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_COMMUTATION_H_
00009 #define INC_COMMUTATION_H_
00010
00011 #include <stdint.h>
00012 #include "hall_sensor_driver.h"
00013 #include "direction.h"
00014
00015 typedef enum {
00016     PHASE_OFF = 0,
00017     PHASE_LS_ON,
00018     PHASE_HS_PWM
00019 } phase_state_t;
00020
00021 void app_commutation_init(void);
00022
00023 void app_commutation_update(hall_sector_t sector, direction_t dir, uint8_t duty);
00024
00025 void app_commutation_stop(void);
00026
00027 #endif /* INC_COMMUTATION_H_ */
```

5.2 motor_control.h

```
00001 /*
00002  * motor_control.h
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_MOTOR_CONTROL_H_
00009 #define INC_MOTOR_CONTROL_H_
00010
00011 #include <stdint.h>
00012
00013 void app_motor_control_init(void);
00014
00015 void app_motor_control_task(void);
00016
00017 void app_motor_control_hall_interrupt_callback(void);
00018
00019 void app_motor_control_brake_interrupt_callback(void);
00020
00021 void app_motor_control_direction_interrupt_callback(void);
00022
00023 #endif /* INC_MOTOR_CONTROL_H_ */
```

5.3 state_machine.h

```
00001 /*
00002  * state_machine.h
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_STATE_MACHINE_H_
00009 #define INC_STATE_MACHINE_H_
00010
00011 #include <stdint.h>
00012
00013 typedef enum {
00014     STATE_INIT = 0,
00015     STATE_IDLE,
00016     STATE_ALIGN,
00017     STATE_START,
00018     STATE_RUN,
00019     STATE_STOP,
00020     STATE_BRAKE,
00021     STATE_FAULT
00022 } motor_state_t;
00023
00024 typedef enum {
00025     EVENT_NONE = 0,
00026     EVENT_THROTTLE_ACTIVE,
00027     EVENT_THROTTLE_ZERO,
00028     EVENT_SECTOR_DETECTED,
00029     EVENT_SPEED_REACHED,
00030     EVENT_SPEED_ZERO,
00031     EVENT_BRAKE_PRESSED,
00032     EVENT_BRAKE_RELEASED,
00033     EVENT_FAULT_OCCURRED,
00034     EVENT_FAULT_CLEARED
00035 } motor_event_t;
00036
00037 void app_state_machine_init(void);
00038
00039 void app_state_machine_process_event(motor_event_t event);
00040
00041 motor_state_t app_state_machine_get_state(void);
00042
00043 void app_state_machine_update(void);
00044
00045 #endif /* INC_STATE_MACHINE_H_ */
```

5.4 commutation.c

```
00001 /*
00002  * commutate.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "commutation.h"
00010 #include "pwm_driver.h"
00011
00012 typedef struct {
00013     phase_state_t u;
00014     phase_state_t v;
00015     phase_state_t w;
00016 } commutation_step_t;
00017
00018 static const commutation_step_t forward_table[8] = {
00019         [HALL_SECTOR_UNKNOWN]   = {PHASE_OFF,       PHASE_OFF,      PHASE_OFF    },
00020         [HALL_SECTOR_4]         = {PHASE_OFF,       PHASE_LS_ON,    PHASE_HS_PWM},
00021         [HALL_SECTOR_2]         = {PHASE_LS_ON,     PHASE_HS_PWM,   PHASE_OFF    },
00022         [HALL_SECTOR_3]         = {PHASE_LS_ON,     PHASE_OFF,      PHASE_HS_PWM},
00023         [HALL_SECTOR_6]         = {PHASE_HS_PWM,    PHASE_OFF,      PHASE_LS_ON },
00024         [HALL_SECTOR_5]         = {PHASE_HS_PWM,    PHASE_LS_ON,    PHASE_OFF    },
00025         [HALL_SECTOR_1]         = {PHASE_OFF,       PHASE_HS_PWM,   PHASE_LS_ON },
00026 };
00027
00028 static const commutation_step_t reverse_table[8] = {
00029         [HALL_SECTOR_UNKNOWN]   = {PHASE_OFF,       PHASE_OFF,      PHASE_OFF    },
00030         [HALL_SECTOR_4]         = {PHASE_OFF,       PHASE_HS_PWM,   PHASE_LS_ON },
00031         [HALL_SECTOR_2]         = {PHASE_HS_PWM,    PHASE_LS_ON,    PHASE_OFF    },
00032         [HALL_SECTOR_3]         = {PHASE_HS_PWM,    PHASE_OFF,      PHASE_LS_ON },
```

```
00033         [HALL_SECTOR_6]         = {PHASE_LS_ON,     PHASE_OFF,      PHASE_HS_PWM},
00034         [HALL_SECTOR_5]         = {PHASE_LS_ON,     PHASE_HS_PWM,   PHASE_OFF    },
00035         [HALL_SECTOR_1]         = {PHASE_OFF,       PHASE_LS_ON,    PHASE_HS_PWM},
00036 };
00037
00038 static void apply_phase_state(pwm_phase_t phase, phase_state_t state, uint8_t duty)
00039 {
00040     switch (state) {
00041         case PHASE_OFF:
00042             ecal_pwm_driver_stop(phase);
00043             ecal_pwm_driver_set_duty_percent(phase, 0);
00044             break;
00045
00046         case PHASE_LS_ON:
00047             ecal_pwm_driver_set_duty_percent(phase, 0);
00048             ecal_pwm_driver_start(phase);
00049             break;
00050
00051         case PHASE_HS_PWM:
00052             ecal_pwm_driver_set_duty_percent(phase, duty);
00053             ecal_pwm_driver_start(phase);
00054             break;
00055     }
00056 }
00057
00058 void app_commutation_init(void)
00059 {
00060     ecal_pwm_driver_init();
00061 }
00062
00063 void app_commutation_update(hall_sector_t sector, direction_t dir, uint8_t duty)
00064 {
00065     if (sector > HALL_SECTOR_6) {
00066         app_commutation_stop();
00067         return;
00068     }
00069
00070     const commutation_step_t *table = (dir == DIRECTION_FORWARD) ? forward_table : reverse_table;
00071     const commutation_step_t *step = &table[sector];
00072
00073     apply_phase_state(PWM_PHASE_U, step->u, duty);
00074     apply_phase_state(PWM_PHASE_V, step->v, duty);
00075     apply_phase_state(PWM_PHASE_W, step->w, duty);
00076 }
00077
00078 void app_commutation_stop(void)
00079 {
00080     ecal_pwm_driver_stop_all();
00081 }
```

5.5 motor_control.c

```
00001 /*
00002  * motor_control.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "motor_control.h"
00010 #include "state_machine.h"
00011 #include "commutation.h"
00012 #include "throttle.h"
00013 #include "hall_sensor_driver.h"
00014 #include "voltage_sensor.h"
00015 #include "current_sensor.h"
00016 #include "temperature_sensor.h"
00017 #include "brake.h"
00018 #include "direction.h"
00019 #include "hal_adc.h"
00020
00021 static uint32_t task_counter;
00022
00023 void app_motor_control_init(void)
00024 {
00025     hal_adc2_init();
00026     hal_adc3_init();
00027
00028     ecal_throttle_init();
00029     ecal_hall_sensor_init();
00030     ecal_voltage_sensor_init();
00031     ecal_current_sensor_init();
```

```
00032     ecal_temperature_sensor_init();
00033     ecal_brake_init();
00034     ecal_direction_init();
00035 
00036     app_commutation_init();
00037     app_state_machine_init();
00038 
00039     hal_adc2_start();
00040     hal_adc3_start();
00041 
00042     task_counter = 0;
00043 }
00044 
00045 void app_motor_control_task(void)
00046 {
00047     task_counter++;
00048 
00049     ecal_throttle_update();
00050     ecal_hall_sensor_update();
00051     ecal_voltage_sensor_update();
00052     ecal_current_sensor_update();
00053     ecal_temperature_sensor_update();
00054     ecal_brake_update();
00055     ecal_direction_update();
00056 
00057     motor_state_t current_state = app_state_machine_get_state();
00058     hall_sector_t sector = ecal_hall_sensor_get_sector();
00059     direction_t dir = ecal_direction_get();
00060     uint8_t throttle = ecal_throttle_get_percent();
00061 
00062     motor_event_t event = EVENT_NONE;
00063 
00064     if (ecal_brake_is_pressed()) {
00065         event = EVENT_BRAKE_PRESSED;
00066         app_state_machine_process_event(event);
00067         return;
00068     }
00069 
00070     if (throttle > 0) {
00071         event = EVENT_THROTTLE_ACTIVE;
00072     } else {
00073         event = EVENT_THROTTLE_ZERO;
00074     }
00075 
00076     if (ecal_hall_sensor_is_valid()) {
00077         app_state_machine_process_event(EVENT_SECTOR_DETECTED);
00078     }
00079 
00080     app_state_machine_process_event(event);
00081 
00082     current_state = app_state_machine_get_state();
00083 
00084     switch (current_state) {
00085         case STATE_INIT:
00086             break;
00087 
00088         case STATE_IDLE:
00089             break;
00090 
00091         case STATE_ALIGN:
00092             app_commutation_update(HALL_SECTOR_1, dir, 20);
00093             break;
00094 
00095         case STATE_START:
00096             app_commutation_update(sector, dir, throttle);
00097             break;
00098 
00099         case STATE_RUN:
00100             app_commutation_update(sector, dir, throttle);
00101             break;
00102 
00103         case STATE_STOP:
00104             break;
00105 
00106         case STATE_BRAKE:
00107             break;
00108 
00109         case STATE_FAULT:
00110             break;
00111     }
00112 }
00113 
00114 void app_motor_control_hall_interrupt_callback(void)
00115 {
00116     ecal_hall_sensor_update();
00117 
00118     hall_sector_t sector = ecal_hall_sensor_get_sector();
```

```
00119     direction_t dir = ecal_direction_get();
00120     uint8_t throttle = ecal_throttle_get_percent();
00121
00122     motor_state_t current_state = app_state_machine_get_state();
00123
00124     if (current_state == STATE_RUN || current_state == STATE_START) {
00125         app_commutation_update(sector, dir, throttle);
00126     }
00127 }
00128
00129 void app_motor_control_brake_interrupt_callback(void)
00130 {
00131     ecal_brake_update();
00132
00133     if (ecal_brake_is_pressed()) {
00134         app_state_machine_process_event(EVENT_BRAKE_PRESSED);
00135     } else {
00136         app_state_machine_process_event(EVENT_BRAKE_RELEASED);
00137     }
00138 }
00139
00140 void app_motor_control_direction_interrupt_callback(void)
00141 {
00142     ecal_direction_update();
00143 }
```

5.6 state_machine.c

```
00001 /*
00002  * state_machine.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "state_machine.h"
00010 #include "commutation.h"
00011
00012 static motor_state_t current_state;
00013 static motor_state_t previous_state;
00014
00015 static void state_transition(motor_state_t new_state);
00016 static void state_entry_init(void);
00017 static void state_entry_idle(void);
00018 static void state_entry_align(void);
00019 static void state_entry_start(void);
00020 static void state_entry_run(void);
00021 static void state_entry_stop(void);
00022 static void state_entry_brake(void);
00023 static void state_entry_fault(void);
00024
00025 void app_state_machine_init(void)
00026 {
00027     current_state = STATE_INIT;
00028     previous_state = STATE_INIT;
00029     state_entry_init();
00030 }
00031
00032 void app_state_machine_process_event(motor_event_t event)
00033 {
00034     switch (current_state) {
00035         case STATE_INIT:
00036             state_transition(STATE_IDLE);
00037             break;
00038
00039         case STATE_IDLE:
00040             if (event == EVENT_BRAKE_PRESSED) {
00041                 state_transition(STATE_BRAKE);
00042             } else if (event == EVENT_FAULT_OCCURRED) {
00043                 state_transition(STATE_FAULT);
00044             } else if (event == EVENT_THROTTLE_ACTIVE) {
00045                 if (event == EVENT_SECTOR_DETECTED) {
00046                     state_transition(STATE_START);
00047                 } else {
00048                     state_transition(STATE_ALIGN);
00049                 }
00050             }
00051             break;
00052
00053         case STATE_ALIGN:
00054             if (event == EVENT_BRAKE_PRESSED) {
00055                 state_transition(STATE_BRAKE);
```

```
00056             } else if (event == EVENT_FAULT_OCCURRED) {
00057                 state_transition(STATE_FAULT);
00058             } else if (event == EVENT_THROTTLE_ZERO) {
00059                 state_transition(STATE_IDLE);
00060             } else if (event == EVENT_SECTOR_DETECTED) {
00061                 state_transition(STATE_START);
00062             }
00063             break;
00064
00065         case STATE_START:
00066             if (event == EVENT_BRAKE_PRESSED) {
00067                 state_transition(STATE_BRAKE);
00068             } else if (event == EVENT_FAULT_OCCURRED) {
00069                 state_transition(STATE_FAULT);
00070             } else if (event == EVENT_THROTTLE_ZERO) {
00071                 state_transition(STATE_IDLE);
00072             } else if (event == EVENT_SPEED_REACHED) {
00073                 state_transition(STATE_RUN);
00074             }
00075             break;
00076
00077         case STATE_RUN:
00078             if (event == EVENT_BRAKE_PRESSED) {
00079                 state_transition(STATE_BRAKE);
00080             } else if (event == EVENT_FAULT_OCCURRED) {
00081                 state_transition(STATE_FAULT);
00082             } else if (event == EVENT_THROTTLE_ZERO) {
00083                 state_transition(STATE_STOP);
00084             }
00085             break;
00086
00087         case STATE_STOP:
00088             if (event == EVENT_BRAKE_PRESSED) {
00089                 state_transition(STATE_BRAKE);
00090             } else if (event == EVENT_FAULT_OCCURRED) {
00091                 state_transition(STATE_FAULT);
00092             } else if (event == EVENT_SPEED_ZERO) {
00093                 state_transition(STATE_IDLE);
00094             } else if (event == EVENT_THROTTLE_ACTIVE) {
00095                 state_transition(STATE_RUN);
00096             }
00097             break;
00098
00099         case STATE_BRAKE:
00100             if (event == EVENT_FAULT_OCCURRED) {
00101                 state_transition(STATE_FAULT);
00102             } else if (event == EVENT_BRAKE_RELEASED) {
00103                 if (event == EVENT_SPEED_ZERO) {
00104                     state_transition(STATE_IDLE);
00105                 } else {
00106                     state_transition(STATE_STOP);
00107                 }
00108             }
00109             break;
00110
00111         case STATE_FAULT:
00112             if (event == EVENT_FAULT_CLEARED) {
00113                 state_transition(STATE_IDLE);
00114             }
00115             break;
00116     }
00117 }
00118
00119 motor_state_t app_state_machine_get_state(void)
00120 {
00121     return current_state;
00122 }
00123
00124 void app_state_machine_update(void)
00125 {
00126
00127 }
00128
00129 static void state_transition(motor_state_t new_state)
00130 {
00131     previous_state = current_state;
00132     current_state = new_state;
00133
00134     switch (new_state) {
00135         case STATE_INIT:
00136             state_entry_init();
00137             break;
00138         case STATE_IDLE:
00139             state_entry_idle();
00140             break;
00141         case STATE_ALIGN:
00142             state_entry_align();
```

```
00143             break;
00144         case STATE_START:
00145             state_entry_start();
00146             break;
00147         case STATE_RUN:
00148             state_entry_run();
00149             break;
00150         case STATE_STOP:
00151             state_entry_stop();
00152             break;
00153         case STATE_BRAKE:
00154             state_entry_brake();
00155             break;
00156         case STATE_FAULT:
00157             state_entry_fault();
00158             break;
00159     }
00160 }
00161
00162 static void state_entry_init(void)
00163 {
00164     app_commutation_stop();
00165 }
00166
00167 static void state_entry_idle(void)
00168 {
00169     app_commutation_stop();
00170 }
00171
00172 static void state_entry_align(void)
00173 {
00174
00175 }
00176
00177 static void state_entry_start(void)
00178 {
00179
00180 }
00181
00182 static void state_entry_run(void)
00183 {
00184
00185 }
00186
00187 static void state_entry_stop(void)
00188 {
00189     app_commutation_stop();
00190 }
00191
00192 static void state_entry_brake(void)
00193 {
00194     app_commutation_stop();
00195 }
00196
00197 static void state_entry_fault(void)
00198 {
00199     app_commutation_stop();
00200 }
```

5.7 ECAL/Inc/brake.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Functions

- void ecal_brake_init (void)
- void ecal_brake_update (void)
- bool ecal_brake_is_pressed (void)

5.7.1 Detailed Description

Author

Şükrü Can Kılıç

Date

2025-12-25

Definition in file brake.h.

5.7.2 Function Documentation

5.7.2.1 ecal_brake_init()

```
void ecal_brake_init (
            void )
```

Definition at line 14 of file brake.c.

5.7.2.2 ecal_brake_is_pressed()

```
bool ecal_brake_is_pressed (
            void )
```

Definition at line 24 of file brake.c.

5.7.2.3 ecal_brake_update()

```
void ecal_brake_update (
            void )
```

Definition at line 19 of file brake.c.

5.8 brake.h

Go to the documentation of this file.

```
00001
00008
00009 #ifndef INC_BRAKE_H_
00010 #define INC_BRAKE_H_
00011
00012 #include <stdint.h>
00013 #include <stdbool.h>
00014
00015 void ecal_brake_init(void);
00016
00017 void ecal_brake_update(void);
00018
00019 bool ecal_brake_is_pressed(void);
00020
00021 #endif /* INC_BRAKE_H_ */
```

5.9 current_sensor.h

```
00001 /*
00002  * current_sensor.h
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_CURRENT_SENSOR_H_
00009 #define INC_CURRENT_SENSOR_H_
00010
00011 #include <stdint.h>
00012
00013 void ecal_current_sensor_init(void);
00014
00015 void ecal_current_sensor_update(void);
00016
00017 float ecal_current_sensor_get_phase_current(void);
00018
00019 #endif /* INC_CURRENT_SENSOR_H_ */
```

5.10 direction.h

```
00001 /*
00002  * direction.h
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_DIRECTION_H_
00009 #define INC_DIRECTION_H_
00010
00011 #include <stdint.h>
00012 #include <stdbool.h>
00013
00014 typedef enum {
00015     DIRECTION_REVERSE = 0,
00016     DIRECTION_FORWARD = 1
00017 } direction_t;
00018
00019 void ecal_direction_init(void);
00020
00021 void ecal_direction_update(void);
00022
00023 direction_t ecal_direction_get(void);
00024
00025 bool ecal_direction_is_forward(void);
00026
00027 #endif /* INC_DIRECTION_H_ */
```

5.11 hall_sensor_driver.h

```
00001 /*
00002  * hall_sensor.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_HALL_SENSOR_DRIVER_H_
00009 #define INC_HALL_SENSOR_DRIVER_H_
00010
00011 #include <stdint.h>
00012 #include <stdbool.h>
00013
00014 typedef enum {
00015     HALL_SECTOR_UNKNOWN = 0,
00016     HALL_SECTOR_1 = 1,
00017     HALL_SECTOR_2 = 2,
00018     HALL_SECTOR_3 = 3,
00019     HALL_SECTOR_4 = 4,
00020     HALL_SECTOR_5 = 5,
00021     HALL_SECTOR_6 = 6
00022 } hall_sector_t;
00023
```

```
00024 void ecal_hall_sensor_init(void);
00025
00026 void ecal_hall_sensor_update(void);
00027
00028 hall_sector_t ecal_hall_sensor_get_sector(void);
00029
00030 uint8_t ecal_hall_sensor_get_raw_state(void);
00031
00032 bool ecal_hall_sensor_is_valid(void);
00033
00034 uint32_t ecal_hall_sensor_get_speed_rpm(void);
00035
00036 #endif /* INC_HALL_SENSOR_DRIVER_H_ */
```

5.12 pwm_driver.h

```
00001 /*
00002  * pwm_driver.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_PWM_DRIVER_H_
00009 #define INC_PWM_DRIVER_H_
00010
00011 #include <stdint.h>
00012
00013 typedef enum {
00014     PWM_PHASE_U = 0,
00015     PWM_PHASE_V = 1,
00016     PWM_PHASE_W = 2
00017 } pwm_phase_t;
00018
00019 void ecal_pwm_driver_init(void);
00020
00021 void ecal_pwm_driver_start(pwm_phase_t phase);
00022
00023 void ecal_pwm_driver_stop(pwm_phase_t phase);
00024
00025 void ecal_pwm_driver_set_duty_percent(pwm_phase_t phase, uint8_t duty_percent);
00026
00027 uint8_t ecal_pwm_driver_get_duty_percent(pwm_phase_t phase);
00028
00029 void ecal_pwm_driver_stop_all(void);
00030
00031 #endif /* INC_PWM_DRIVER_H_ */
```

5.13 temperature_sensor.h

```
00001 /*
00002  * temperature.sensor.h
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_TEMPERATURE_SENSOR_H_
00009 #define INC_TEMPERATURE_SENSOR_H_
00010
00011 #include <stdint.h>
00012
00013 void ecal_temperature_sensor_init(void);
00014
00015 void ecal_temperature_sensor_update(void);
00016
00017 float ecal_temperature_sensor_get_mosfet_temp(void);
00018
00019 float ecal_temperature_sensor_get_motor_temp(void);
00020
00021 #endif /* INC_TEMPERATURE_SENSOR_H_ */
```

5.14 throttle.h

```
00001 /*
```

```
00002  * throttle.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_THROTTLE_H_
00009 #define INC_THROTTLE_H_
00010
00011 #include <stdint.h>
00012 #include <stdbool.h>
00013
00014 void ecal_throttle_init(void);
00015
00016 void ecal_throttle_update(void);
00017
00018 uint8_t ecal_throttle_get_percent(void);
00019
00020 #endif /* INC_THROTTLE_H_ */
```

5.15 voltage_sensor.h

```
00001 /*
00002  * voltage_sensor.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_VOLTAGE_SENSOR_H_
00009 #define INC_VOLTAGE_SENSOR_H_
00010
00011 #include <stdint.h>
00012
00013 void ecal_voltage_sensor_init(void);
00014
00015 void ecal_voltage_sensor_update(void);
00016
00017 float ecal_voltage_sensor_get_vbus(void);
00018
00019 #endif /* INC_VOLTAGE_SENSOR_H_ */
```

5.16 brake.c

```
00001 /*
00002  * brake.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "brake.h"
00010 #include "hal_gpio.h"
00011
00012 static bool brake_pressed;
00013
00014 void ecal_brake_init(void)
00015 {
00016     brake_pressed = false;
00017 }
00018
00019 void ecal_brake_update(void)
00020 {
00021     brake_pressed = !hal_gpio_read_brake();
00022 }
00023
00024 bool ecal_brake_is_pressed(void)
00025 {
00026     return brake_pressed;
00027 }
```

5.17 current_sensor.c

```
00001 /*
```

```
00002  * current_sensor.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "current_sensor.h"
00010 #include "hal_adc.h"
00011 #include "filter.h"
00012
00013 #define ADC_VREF            3.3f
00014 #define ADC_RESOLUTION      4096.0f
00015 #define SHUNT_RESISTANCE    0.001f
00016 #define OPAMP_GAIN          16.0f
00017 #define ADC_OFFSET          2382
00018
00019 static float phase_current;
00020
00021 void ecal_current_sensor_init(void)
00022 {
00023     phase_current = 0.0f;
00024 }
00025
00026 void ecal_current_sensor_update(void)
00027 {
00028
00029     uint16_t adc_value = (uint16_t)hal_adc3_get_raw_value();
00030     uint16_t adc_value_filtered = adc_callback(adc_value);
00031
00032     int32_t adc_diff = (int32_t)adc_value_filtered - ADC_OFFSET;
00033
00034     float voltage = ((float)adc_diff / ADC_RESOLUTION) * ADC_VREF;
00035
00036     phase_current = voltage / (SHUNT_RESISTANCE * OPAMP_GAIN);
00037 }
00038
00039 float ecal_current_sensor_get_phase_current(void)
00040 {
00041     return phase_current;
00042 }
```

5.18 direction.c

```
00001 /*
00002  * direction.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "direction.h"
00010 #include "hal_gpio.h"
00011
00012 static direction_t current_direction;
00013
00014 void ecal_direction_init(void)
00015 {
00016     current_direction = DIRECTION_FORWARD;
00017 }
00018
00019 void ecal_direction_update(void)
00020 {
00021     if (hal_gpio_read_direction()) {
00022         current_direction = DIRECTION_FORWARD;
00023     } else {
00024         current_direction = DIRECTION_REVERSE;
00025     }
00026 }
00027
00028 direction_t ecal_direction_get(void)
00029 {
00030     return current_direction;
00031 }
00032
00033 bool ecal_direction_is_forward(void)
00034 {
00035     return (current_direction == DIRECTION_FORWARD);
00036 }
```

5.19 hall_sensor_driver.c

```
00001 /*
00002  * hall_sensor.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "hall_sensor_driver.h"
00010 #include "hal_gpio.h"
00011 #include "hal_timer.h"
00012
00013 #define POLE_PAIRS 9
00014 #define INVALID_SECTOR 0
00015
00016 static const hall_sector_t hall_lookup_table[8] = {
00017     HALL_SECTOR_UNKNOWN,  // 0b000
00018     HALL_SECTOR_4,        // 0b001
00019     HALL_SECTOR_2,        // 0b010
00020     HALL_SECTOR_3,        // 0b011
00021     HALL_SECTOR_6,        // 0b100
00022     HALL_SECTOR_5,        // 0b101
00023     HALL_SECTOR_1,        // 0b110
00024     HALL_SECTOR_UNKNOWN   // 0b111
00025 };
00026
00027 static hall_sector_t current_sector;
00028 static uint8_t raw_hall_state;
00029 static uint32_t capture_value;
00030 static uint32_t speed_rpm;
00031
00032 void ecal_hall_sensor_init(void)
00033 {
00034     current_sector = HALL_SECTOR_UNKNOWN;
00035     raw_hall_state = 0;
00036     capture_value = 0;
00037     speed_rpm = 0;
00038 }
00039
00040 void ecal_hall_sensor_update(void)
00041 {
00042     raw_hall_state = hal_gpio_read_hall_sensors();
00043     current_sector = hall_lookup_table[raw_hall_state];
00044
00045     capture_value = hal_timer2_get_capture_value();
00046
00047     if (capture_value > 0) {
00048         uint32_t time_per_sector_us = capture_value;
00049         uint32_t time_per_revolution_us = time_per_sector_us * 6;
00050
00051         speed_rpm = (60000000 / time_per_revolution_us) / POLE_PAIRS;
00052     } else {
00053         speed_rpm = 0;
00054     }
00055 }
00056
00057 hall_sector_t ecal_hall_sensor_get_sector(void)
00058 {
00059     return current_sector;
00060 }
00061
00062 uint8_t ecal_hall_sensor_get_raw_state(void)
00063 {
00064     return raw_hall_state;
00065 }
00066
00067 bool ecal_hall_sensor_is_valid(void)
00068 {
00069     return (current_sector != HALL_SECTOR_UNKNOWN);
00070 }
00071
00072 uint32_t ecal_hall_sensor_get_speed_rpm(void)
00073 {
00074     return speed_rpm;
00075 }
```

5.20 pwm_driver.c

```
00001 /*
00002  * pwm_driver.c
```

```
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "pwm_driver.h"
00010 #include "hal_timer.h"
00011
00012 #define PWM_MAX_DUTY_PERCENT 100
00013
00014 static uint8_t duty_percent[3] = {0, 0, 0};
00015
00016 void ecal_pwm_driver_init(void)
00017 {
00018     duty_percent[PWM_PHASE_U] = 0;
00019     duty_percent[PWM_PHASE_V] = 0;
00020     duty_percent[PWM_PHASE_W] = 0;
00021 }
00022
00023 void ecal_pwm_driver_start(pwm_phase_t phase)
00024 {
00025     if (phase > PWM_PHASE_W) {
00026         return;
00027     }
00028
00029     hal_timer1_pwm_start(phase);
00030 }
00031
00032 void ecal_pwm_driver_stop(pwm_phase_t phase)
00033 {
00034     if (phase > PWM_PHASE_W) {
00035         return;
00036     }
00037
00038     hal_timer1_pwm_stop(phase);
00039 }
00040
00041 void ecal_pwm_driver_set_duty_percent(pwm_phase_t phase, uint8_t duty_pct)
00042 {
00043     if (phase > PWM_PHASE_W) {
00044         return;
00045     }
00046
00047     if (duty_pct > PWM_MAX_DUTY_PERCENT) {
00048         duty_pct = PWM_MAX_DUTY_PERCENT;
00049     }
00050
00051     duty_percent[phase] = duty_pct;
00052
00053     uint16_t arr = hal_timer1_get_arr();
00054     uint16_t duty_value = (uint16_t)((arr * duty_pct) / 100);
00055
00056     hal_timer1_set_duty(phase, duty_value);
00057 }
00058
00059 uint8_t ecal_pwm_driver_get_duty_percent(pwm_phase_t phase)
00060 {
00061     if (phase > PWM_PHASE_W) {
00062         return 0;
00063     }
00064
00065     return duty_percent[phase];
00066 }
00067
00068 void ecal_pwm_driver_stop_all(void)
00069 {
00070     ecal_pwm_driver_stop(PWM_PHASE_U);
00071     ecal_pwm_driver_stop(PWM_PHASE_V);
00072     ecal_pwm_driver_stop(PWM_PHASE_W);
00073
00074     ecal_pwm_driver_set_duty_percent(PWM_PHASE_U, 0);
00075     ecal_pwm_driver_set_duty_percent(PWM_PHASE_V, 0);
00076     ecal_pwm_driver_set_duty_percent(PWM_PHASE_W, 0);
00077 }
```

5.21 temperature_sensor.c

```
00001 /*
00002  * temperature_sensor.c
00003  *
00004  *  Created on: Dec 25, 2025
00005  *      Author: Can
```

```
00006  */
00007
00008 #include "temperature_sensor.h"
00009 #include "hal_adc.h"
00010 #include <math.h>
00011
00012 #define ADC_VREF            3.3f
00013 #define ADC_RESOLUTION      4096.0f
00014 #define NTC_R25             10000.0f
00015 #define NTC_B_VALUE         3950.0f
00016 #define SERIES_RESISTOR     10000.0f
00017 #define T25_KELVIN          298.15f
00018
00019 static float mosfet_temp;
00020 static float motor_temp;
00021
00022 static float calculate_temperature(uint16_t adc_value)
00023 {
00024     float adc_voltage = ((float)adc_value / ADC_RESOLUTION) * ADC_VREF;
00025
00026     float ntc_resistance = SERIES_RESISTOR * (ADC_VREF / adc_voltage - 1.0f);
00027
00028     float temp_kelvin = 1.0f / ((1.0f / T25_KELVIN) + (1.0f / NTC_B_VALUE) * logf(ntc_resistance /
     NTC_R25));
00029
00030     return temp_kelvin - 273.15f;
00031 }
00032
00033 void ecal_temperature_sensor_init(void)
00034 {
00035     mosfet_temp = 0.0f;
00036     motor_temp = 0.0f;
00037 }
00038
00039 void ecal_temperature_sensor_update(void)
00040 {
00041     uint16_t adc_mosfet = (uint16_t)hal_adc2_get_raw_value(HAL_ADC2_CH_TEMP_MOSFET);
00042     uint16_t adc_motor = (uint16_t)hal_adc2_get_raw_value(HAL_ADC2_CH_TEMP_MOTOR);
00043
00044     mosfet_temp = calculate_temperature(adc_mosfet);
00045     motor_temp = calculate_temperature(adc_motor);
00046 }
00047
00048 float ecal_temperature_sensor_get_mosfet_temp(void)
00049 {
00050     return mosfet_temp;
00051 }
00052
00053 float ecal_temperature_sensor_get_motor_temp(void)
00054 {
00055     return motor_temp;
00056 }
```

5.22 throttle.c

```
00001 /*
00002  * throttle.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "throttle.h"
00010 #include "hal_adc.h"
00011
00012 #define THROTTLE_ADC_MIN      200
00013 #define THROTTLE_ADC_MAX      4000
00014 #define THROTTLE_DEADZONE     50
00015
00016 static uint8_t throttle_percent;
00017
00018 void ecal_throttle_init(void)
00019 {
00020     throttle_percent = 0;
00021 }
00022
00023 void ecal_throttle_update(void)
00024 {
00025     uint16_t adc_value = (uint16_t)hal_adc2_get_raw_value(HAL_ADC2_CH_THROTTLE);
00026
00027     if (adc_value < THROTTLE_ADC_MIN + THROTTLE_DEADZONE) {
00028         throttle_percent = 0;
```

```
00029         return;
00030     }
00031
00032     if (adc_value > THROTTLE_ADC_MAX) {
00033         throttle_percent = 100;
00034         return;
00035     }
00036
00037     uint32_t range = THROTTLE_ADC_MAX - THROTTLE_ADC_MIN;
00038     uint32_t value = adc_value - THROTTLE_ADC_MIN;
00039
00040     throttle_percent = (uint8_t)((value * 100) / range);
00041
00042     if (throttle_percent > 100) {
00043         throttle_percent = 100;
00044     }
00045 }
00046
00047 uint8_t ecal_throttle_get_percent(void)
00048 {
00049     return throttle_percent;
00050 }
```

5.23 voltage_sensor.c

```
00001 /*
00002  * voltage_sensor.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "voltage_sensor.h"
00010 #include "hal_adc.h"
00011
00012 #define ADC_VREF            3.3f
00013 #define ADC_RESOLUTION      4096.0f
00014 #define VOLTAGE_DIVIDER     15.0f
00015
00016 static float vbus_voltage;
00017
00018 void ecal_voltage_sensor_init(void)
00019 {
00020     vbus_voltage = 0.0f;
00021 }
00022
00023 void ecal_voltage_sensor_update(void)
00024 {
00025     uint16_t adc_value = (uint16_t)hal_adc2_get_raw_value(HAL_ADC2_CH_VBUS);
00026
00027     float adc_voltage = ((float)adc_value / ADC_RESOLUTION) * ADC_VREF;
00028
00029     vbus_voltage = adc_voltage * VOLTAGE_DIVIDER;
00030 }
00031
00032 float ecal_voltage_sensor_get_vbus(void)
00033 {
00034     return vbus_voltage;
00035 }
```

5.24 hal_adc.h

```
00001 /*
00002  * hal_adc.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_HAL_ADC_H_
00009 #define INC_HAL_ADC_H_
00010
00011 #include <stdint.h>
00012
00013 #define HAL_ADC2_CH_VBUS          0
00014 #define HAL_ADC2_CH_TEMP_MOSFET   1
00015 #define HAL_ADC2_CH_TEMP_MOTOR    2
```

```
00016 #define HAL_ADC2_CH_THROTTLE      3
00017 #define HAL_ADC2_CH_COUNT         4
00018
00019 void hal_adc2_init(void);
00020 void hal_adc3_init(void);
00021 void hal_adc2_start(void);
00022 void hal_adc3_start(void);
00023 void hal_adc2_stop(void);
00024 void hal_adc3_stop(void);
00025 uint16_t hal_adc2_get_raw_value(uint8_t channel);
00026 uint16_t hal_adc3_get_raw_value(void);
00027
00028 #endif /* INC_HAL_ADC_H_ */
```

5.25 hal_gpio.h

```
00001 /*
00002  * hal_gpio.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_HAL_GPIO_H_
00009 #define INC_HAL_GPIO_H_
00010
00011 #include <stdint.h>
00012 #include <stdbool.h>
00013
00014 void hal_gpio_init(void);
00015
00016 uint8_t hal_gpio_read_hall_sensors(void);
00017
00018 bool hal_gpio_read_brake(void);
00019
00020 bool hal_gpio_read_direction(void);
00021
00022 #endif /* INC_HAL_GPIO_H_ */
```

5.26 hal_timer.h

```
00001 /*
00002  * hal_timer.h
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008 #ifndef INC_HAL_TIMER_H_
00009 #define INC_HAL_TIMER_H_
00010
00011 #include <stdint.h>
00012
00013 #define HAL_PWM_CH_U  0
00014 #define HAL_PWM_CH_V  1
00015 #define HAL_PWM_CH_W  2
00016 #define HAL_PWM_CH_COUNT 3
00017
00018 void hal_timer1_pwm_init(void);
00019
00020 void hal_timer1_pwm_start(uint8_t channel);
00021
00022 void hal_timer1_pwm_stop(uint8_t channel);
00023
00024 void hal_timer1_set_duty(uint8_t channel, uint16_t duty);
00025
00026 uint16_t hal_timer1_get_duty(uint8_t channel);
00027
00028 void hal_timer1_set_arr(uint16_t arr);
00029
00030 uint16_t hal_timer1_get_arr(void);
00031
00032 void hal_timer2_input_capture_init(void);
00033
00034 void hal_timer2_input_capture_start(void);
00035
00036 void hal_timer2_input_capture_stop(void);
00037
```

```
00038 uint32_t hal_timer2_get_capture_value(void);
00039
00040 #endif /* INC_HAL_TIMER_H_ */
```

5.27 hal_adc.c

```
00001 /*
00002  * hal_adc.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "hal_adc.h"
00010 #include "adc.h"
00011 #include "opamp.h"
00012
00013 static uint16_t adc2_dma_buffer[HAL_ADC2_CH_COUNT];
00014 static uint32_t adc3_dma_buffer;
00015
00016 void hal_adc2_init(void)
00017 {
00018     MX_ADC2_Init();
00019 }
00020
00021 void hal_adc3_init(void)
00022 {
00023     MX_OPAMP3_Init();
00024     MX_ADC3_Init();
00025 }
00026
00027 void hal_adc2_start(void)
00028 {
00029     HAL_OPAMP_Start(&hopamp3);
00030     HAL_ADC_Start_DMA(&hadc2, (uint32_t*)adc2_dma_buffer, HAL_ADC2_CH_COUNT);
00031 }
00032
00033 void hal_adc3_start(void)
00034 {
00035     HAL_ADC_Start_DMA(&hadc3, &adc3_dma_buffer, 1);
00036 }
00037
00038 uint16_t hal_adc2_get_raw_value(uint8_t channel)
00039 {
00040     if (channel < HAL_ADC2_CH_COUNT) {
00041         return adc2_dma_buffer[channel];
00042     }
00043     return 0;
00044 }
00045
00046 uint16_t hal_adc3_get_raw_value(void)
00047 {
00048     return (uint16_t)adc3_dma_buffer;
00049 }
```

5.28 hal_gpio.c

```
00001 /*
00002  * hal_gpio.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "hal_gpio.h"
00010 #include "gpio.h"
00011
00012 void hal_gpio_init(void) {
00013     MX_GPIO_Init();
00014 }
00015
00016 uint8_t hal_gpio_read_hall_sensors(void) {
00017     uint8_t hall_state = 0U;
00018     hall_state |= HAL_GPIO_ReadPin(HALL_SENSOR_U_GPIO_Port, HALL_SENSOR_U_Pin) << 2;
00019     hall_state |= HAL_GPIO_ReadPin(HALL_SENSOR_V_GPIO_Port, HALL_SENSOR_V_Pin) << 1;
00020     hall_state |= HAL_GPIO_ReadPin(HALL_SENSOR_W_GPIO_Port, HALL_SENSOR_W_Pin) << 0;
```

```
00021
00022     return hall_state & 0x07U;
00023 }
00024
00025 bool hal_gpio_read_brake(void) {
00026     return HAL_GPIO_ReadPin(BRAKE_BUTTON_GPIO_Port, BRAKE_BUTTON_Pin);
00027 }
00028
00029 bool hal_gpio_read_direction(void) {
00030     return HAL_GPIO_ReadPin(DIRECTION_BUTTON_GPIO_Port, DIRECTION_BUTTON_Pin);
00031 }
```

5.29 hal_timer.c

```
00001 /*
00002  * hal_timer.c
00003  *
00004  *  Created on: Dec 24, 2025
00005  *      Author: Can
00006  */
00007
00008
00009 #include "hal_timer.h"
00010 #include "tim.h"
00011
00012 static uint16_t pwm_duty[HAL_PWM_CH_COUNT] = {0};
00013
00014 void hal_timer1_pwm_init(void)
00015 {
00016     MX_TIM1_Init();
00017 }
00018
00019 void hal_timer1_pwm_start(uint8_t channel)
00020 {
00021     if (channel >= HAL_PWM_CH_COUNT) {
00022         return;
00023     }
00024
00025     switch (channel) {
00026         case HAL_PWM_CH_U:
00027             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
00028             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
00029             break;
00030         case HAL_PWM_CH_V:
00031             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
00032             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
00033             break;
00034         case HAL_PWM_CH_W:
00035             HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
00036             HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
00037             break;
00038     }
00039 }
00040
00041 void hal_timer1_pwm_stop(uint8_t channel)
00042 {
00043     if (channel >= HAL_PWM_CH_COUNT) {
00044         return;
00045     }
00046
00047     switch (channel) {
00048         case HAL_PWM_CH_U:
00049             HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
00050             HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_1);
00051             break;
00052         case HAL_PWM_CH_V:
00053             HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
00054             HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_2);
00055             break;
00056         case HAL_PWM_CH_W:
00057             HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
00058             HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_3);
00059             break;
00060     }
00061 }
00062
00063 void hal_timer1_set_duty(uint8_t channel, uint16_t duty)
00064 {
00065     if (channel >= HAL_PWM_CH_COUNT) {
00066         return;
00067     }
00068
00069     pwm_duty[channel] = duty;
```

```
00070
00071     switch (channel) {
00072         case HAL_PWM_CH_U:
00073             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, duty);
00074             break;
00075         case HAL_PWM_CH_V:
00076             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, duty);
00077             break;
00078         case HAL_PWM_CH_W:
00079             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, duty);
00080             break;
00081     }
00082 }
00083
00084 uint16_t hal_timer1_get_duty(uint8_t channel)
00085 {
00086     if (channel < HAL_PWM_CH_COUNT) {
00087         return pwm_duty[channel];
00088     }
00089     return 0;
00090 }
00091
00092 void hal_timer1_set_arr(uint16_t arr)
00093 {
00094     __HAL_TIM_SET_AUTORELOAD(&htim1, arr);
00095 }
00096
00097 uint16_t hal_timer1_get_arr(void)
00098 {
00099     return __HAL_TIM_GET_AUTORELOAD(&htim1);
00100 }
00101
00102 void hal_timer2_input_capture_init(void)
00103 {
00104     MX_TIM2_Init();
00105 }
00106
00107 void hal_timer2_input_capture_start(void)
00108 {
00109     HAL_TIMEx_HallSensor_Start_IT(&htim2);
00110 }
00111
00112 void hal_timer2_input_capture_stop(void)
00113 {
00114     HAL_TIMEx_HallSensor_Stop_IT(&htim2);
00115 }
00116
00117 uint32_t hal_timer2_get_capture_value(void)
00118 {
00119     return HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_1);
00120 }
```

Index